Bottom-Up Parsing

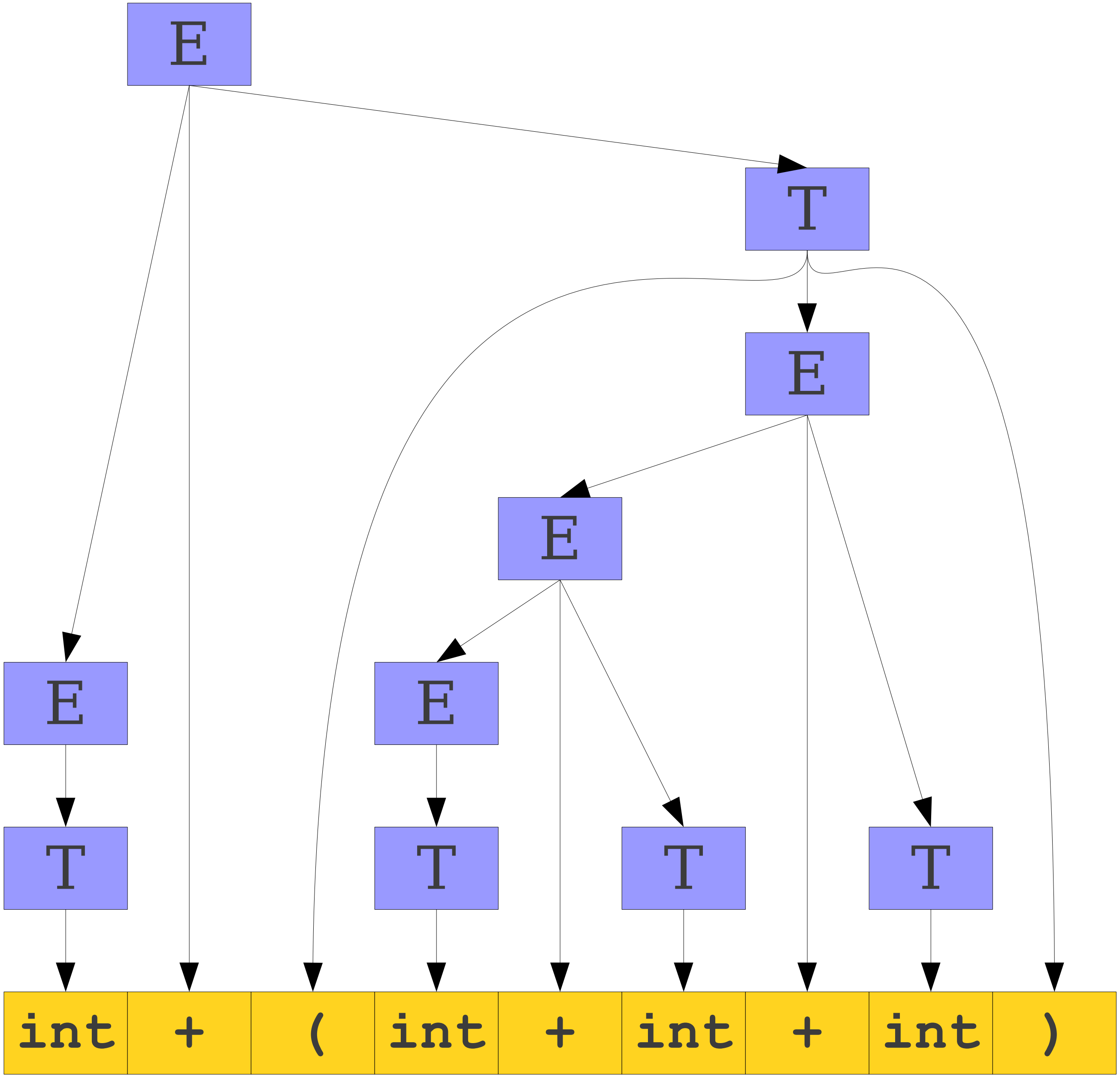
What is Bottom-Up Parsing?

- Idea: Apply productions **in reverse** to convert the user's program to the start symbol.
- As with top-down, could be done with a DFS or BFS, though this is rarely done in practice.
- We'll be exploring four **directional**, **predictive** bottom-up parsing techniques:
  - **Directional**: Scan the input from left-to-right.
  - **Predictive**: Guess which production should be inverted.

Bottoms Up!

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.

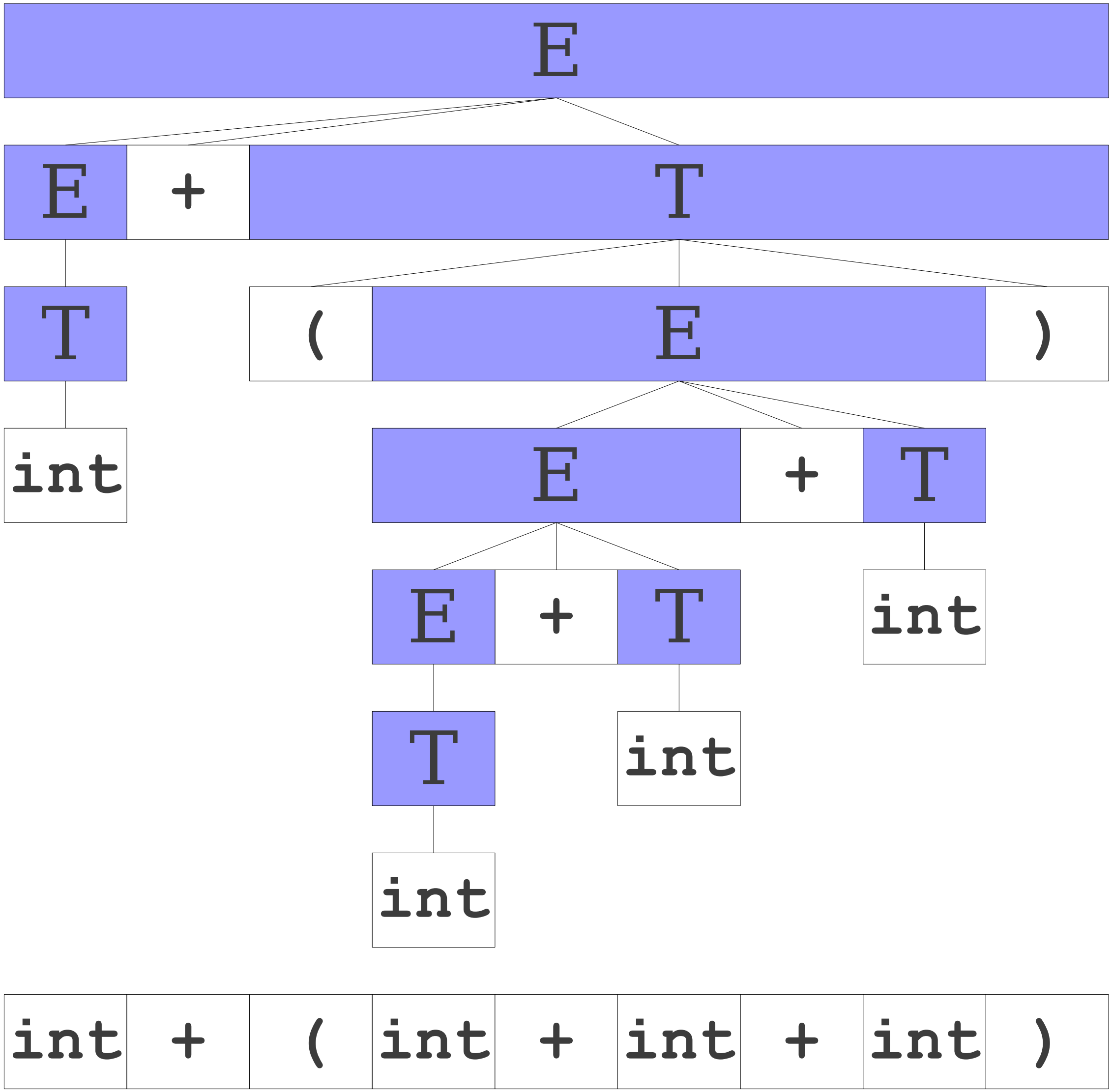
A Third View of a Bottom-Up Parse

```
  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
```

Each step in this bottom-up parse is called a **reduction**. We **reduce** a substring of the sentential form back to a nonterminal.

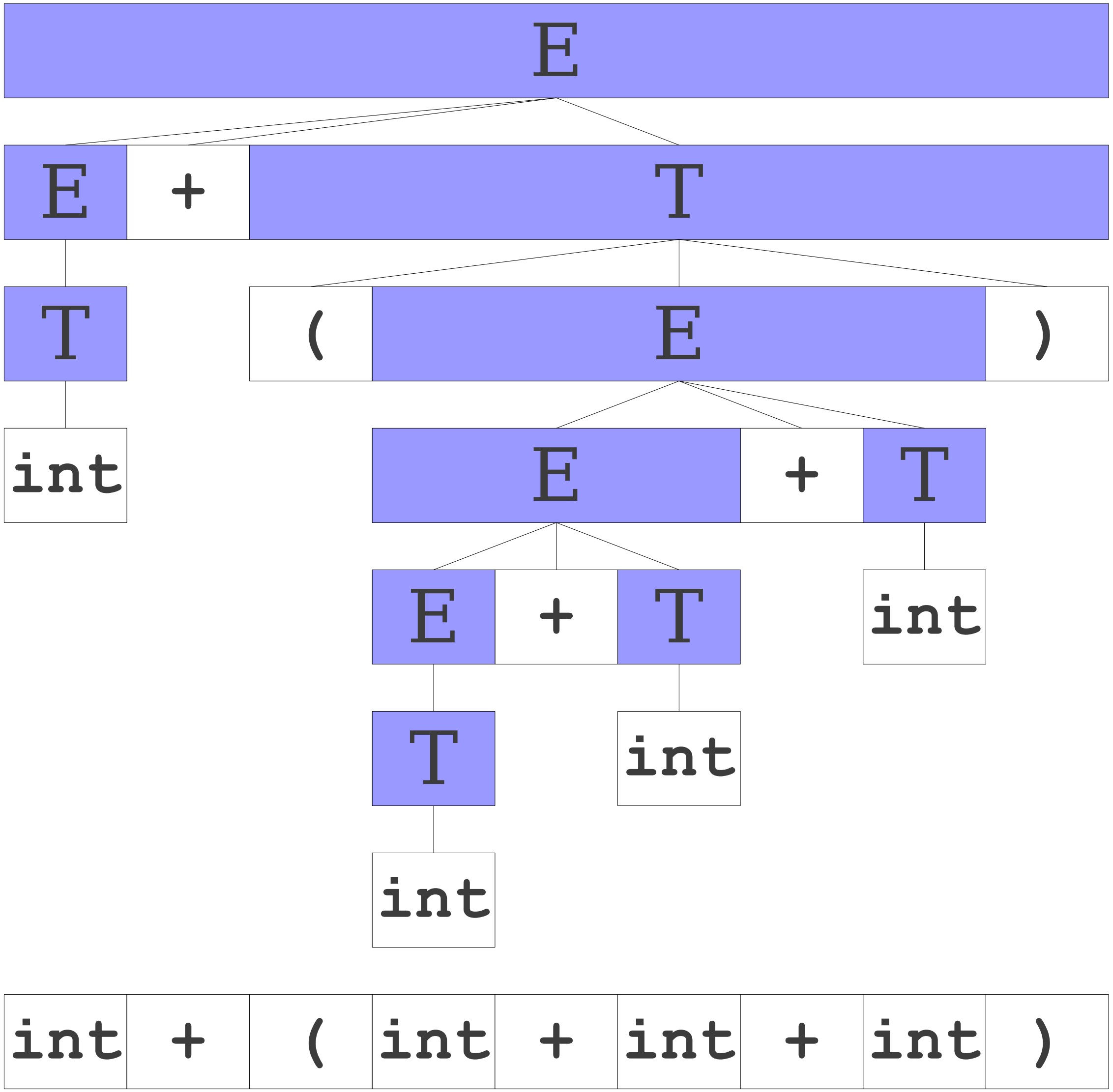
A Third View of a Bottom-Up Parse

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



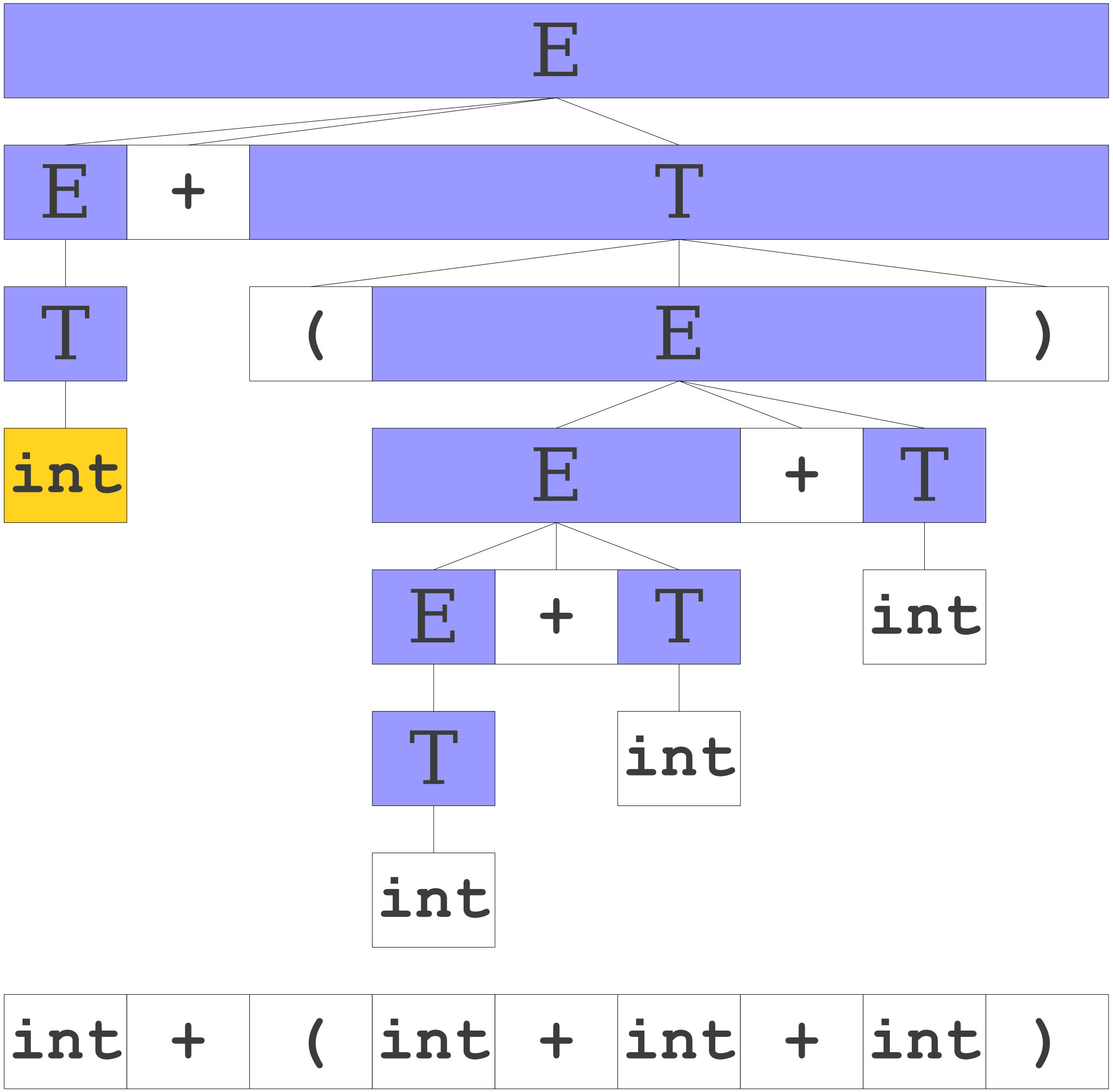
A Third View of a Bottom-Up Parse

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



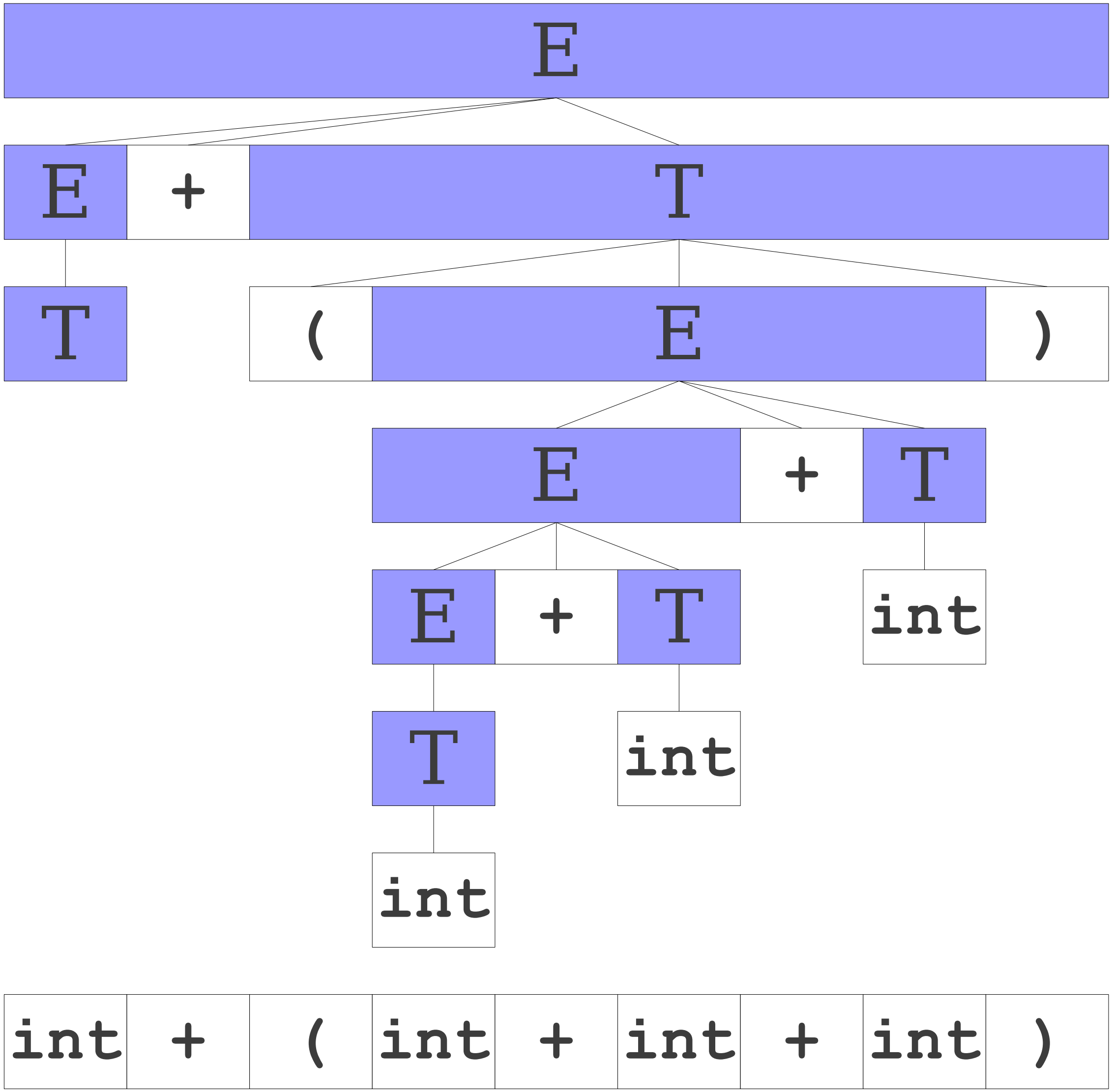
A Third View of a Bottom-Up Parse

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



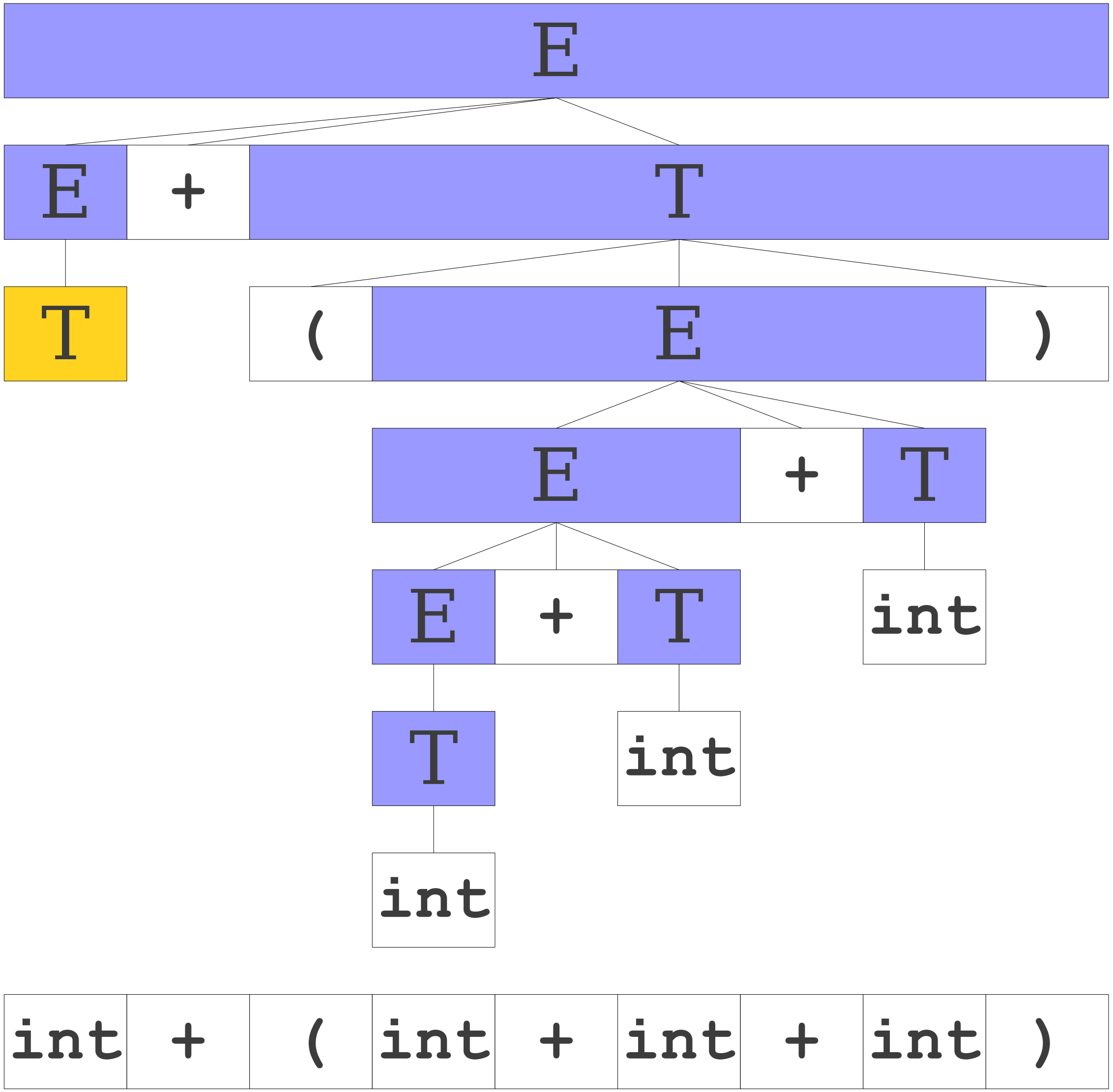
A Third View of a Bottom-Up Parse

⇒ 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



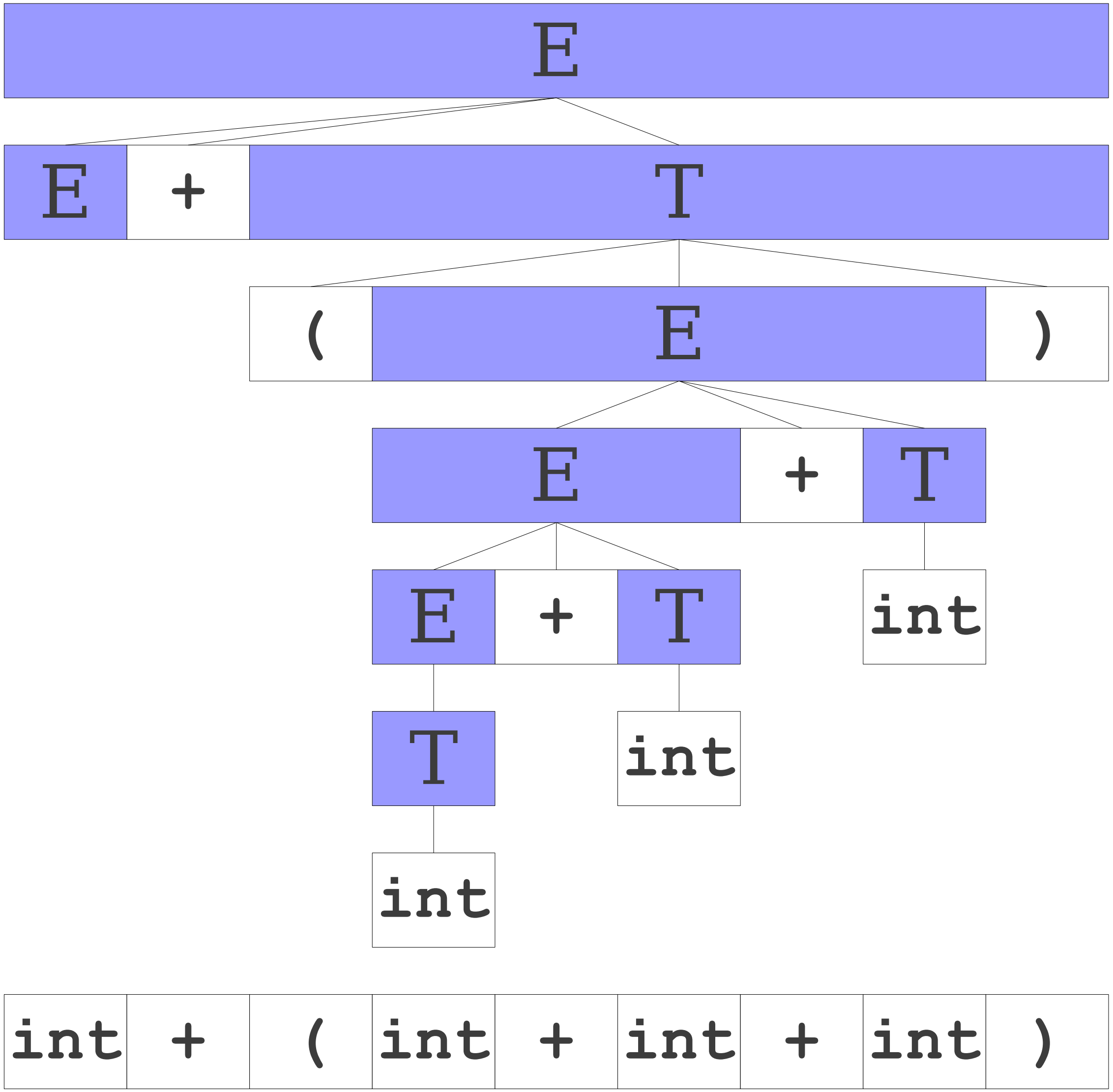
A Third View of a Bottom-Up Parse

⇒ **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 Third View of a Bottom-Up Parse

⇒ 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 Third View of a Bottom-Up Parse

⇒ **E** + `(`**`int`**` + int + int)`

⇒ **E** + (**T** + `int + int`)

⇒ **E** + (**E** + `int + int`)

⇒ **E** + (**E** + **T** + `int`)

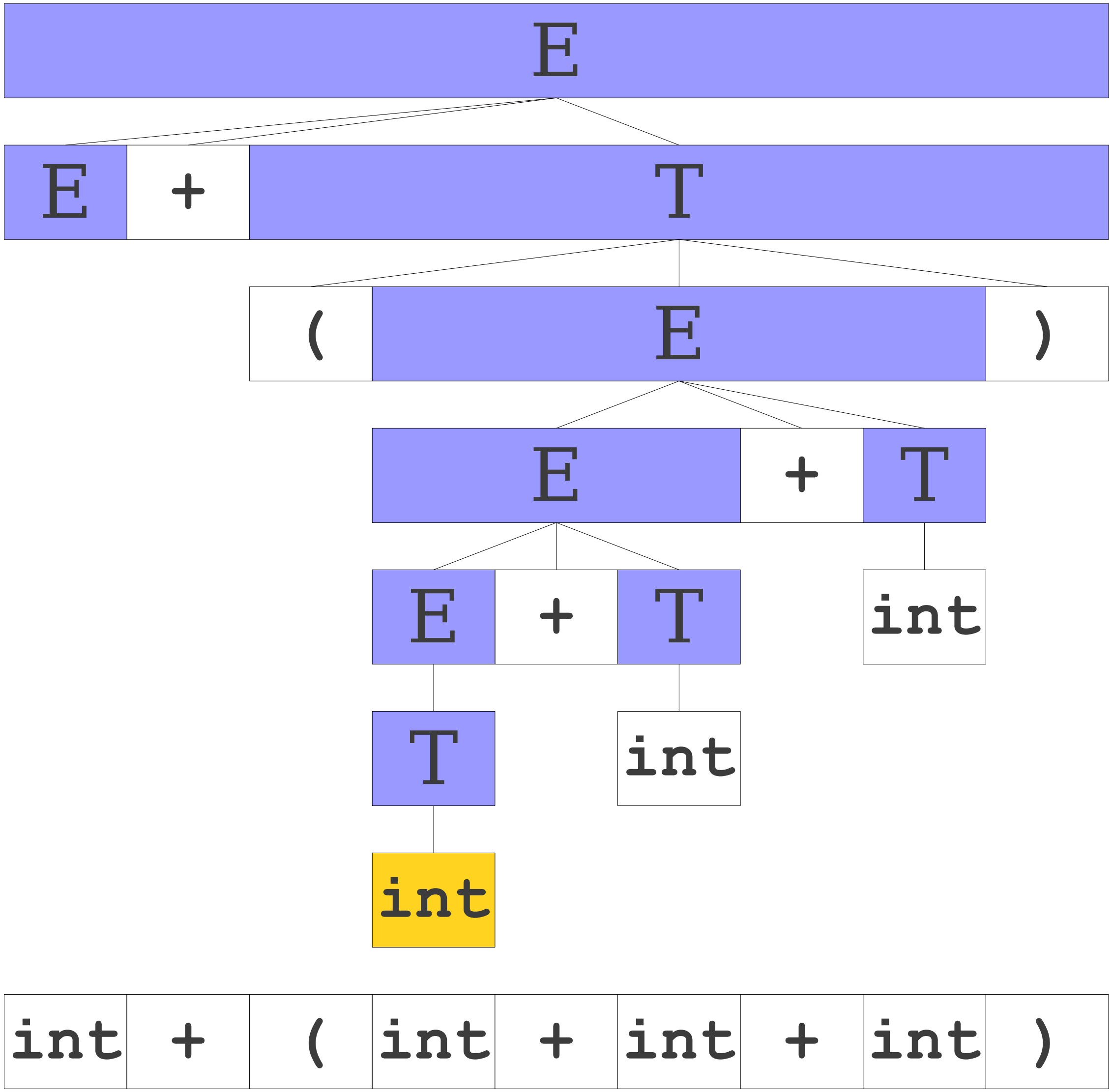
⇒ **E** + (**E** + `int`)

⇒ **E** + (**E** + **T**)

⇒ **E** + (**E**)

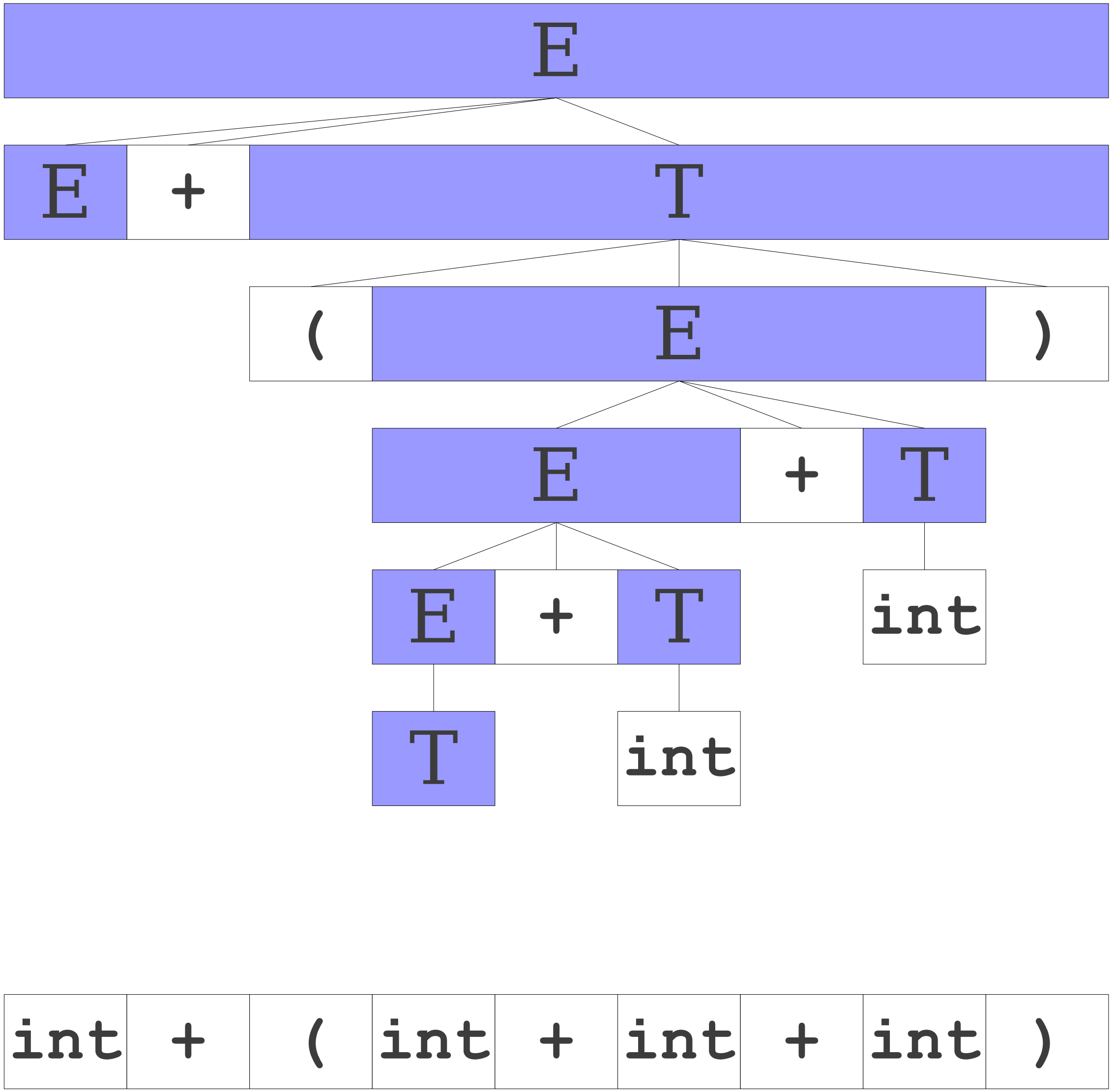
⇒ **E** + **T**

⇒ **E**



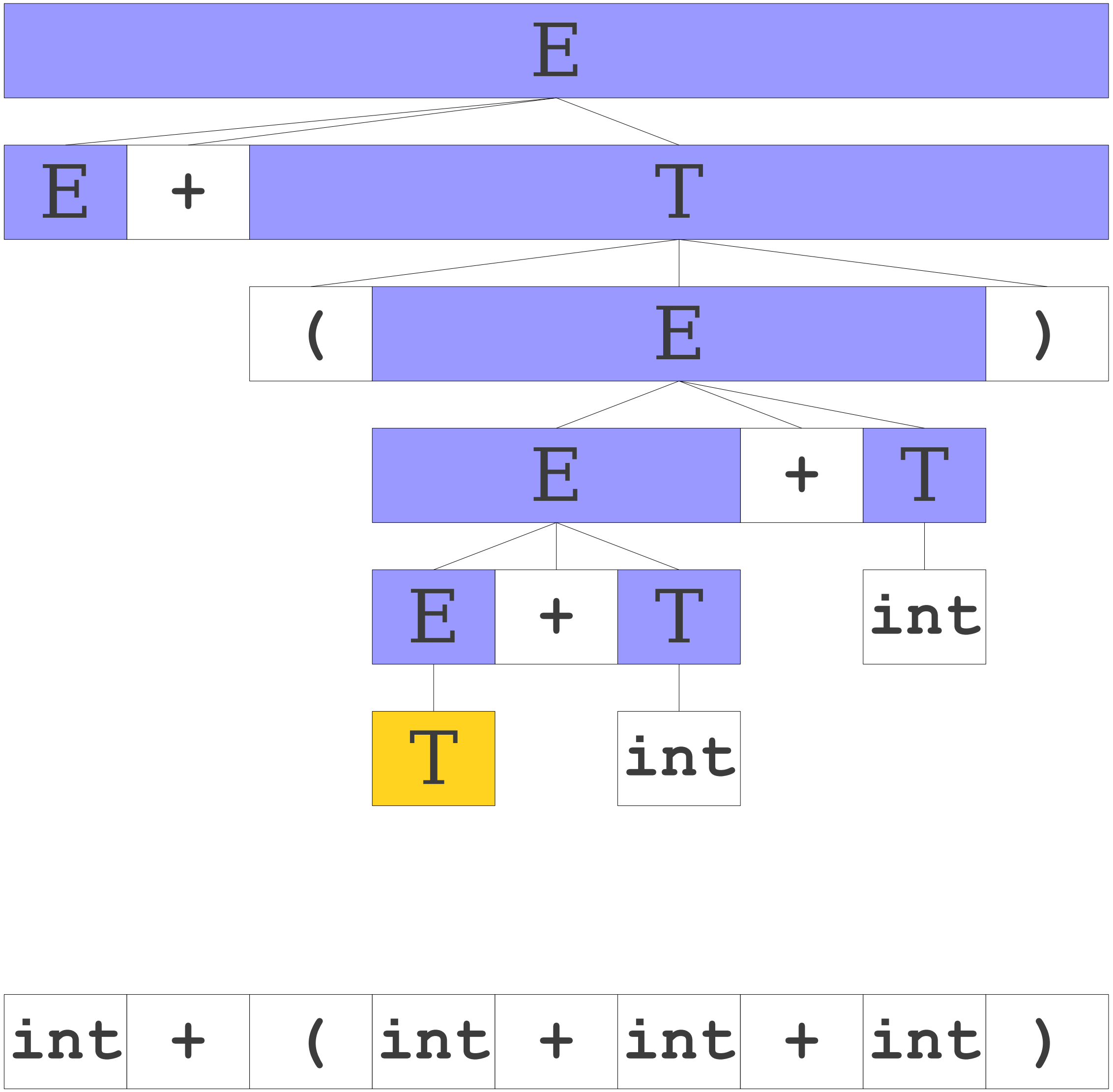
A Third View of a Bottom-Up Parse

⇒ **E** + (**T** + int + int)
⇒ **E** + (**E** + int + int)
⇒ **E** + (**E** + **T** + int)
⇒ **E** + (**E** + int)
⇒ **E** + (**E** + **T**)
⇒ **E** + (**E**)
⇒ **E** + **T**
⇒ **E**

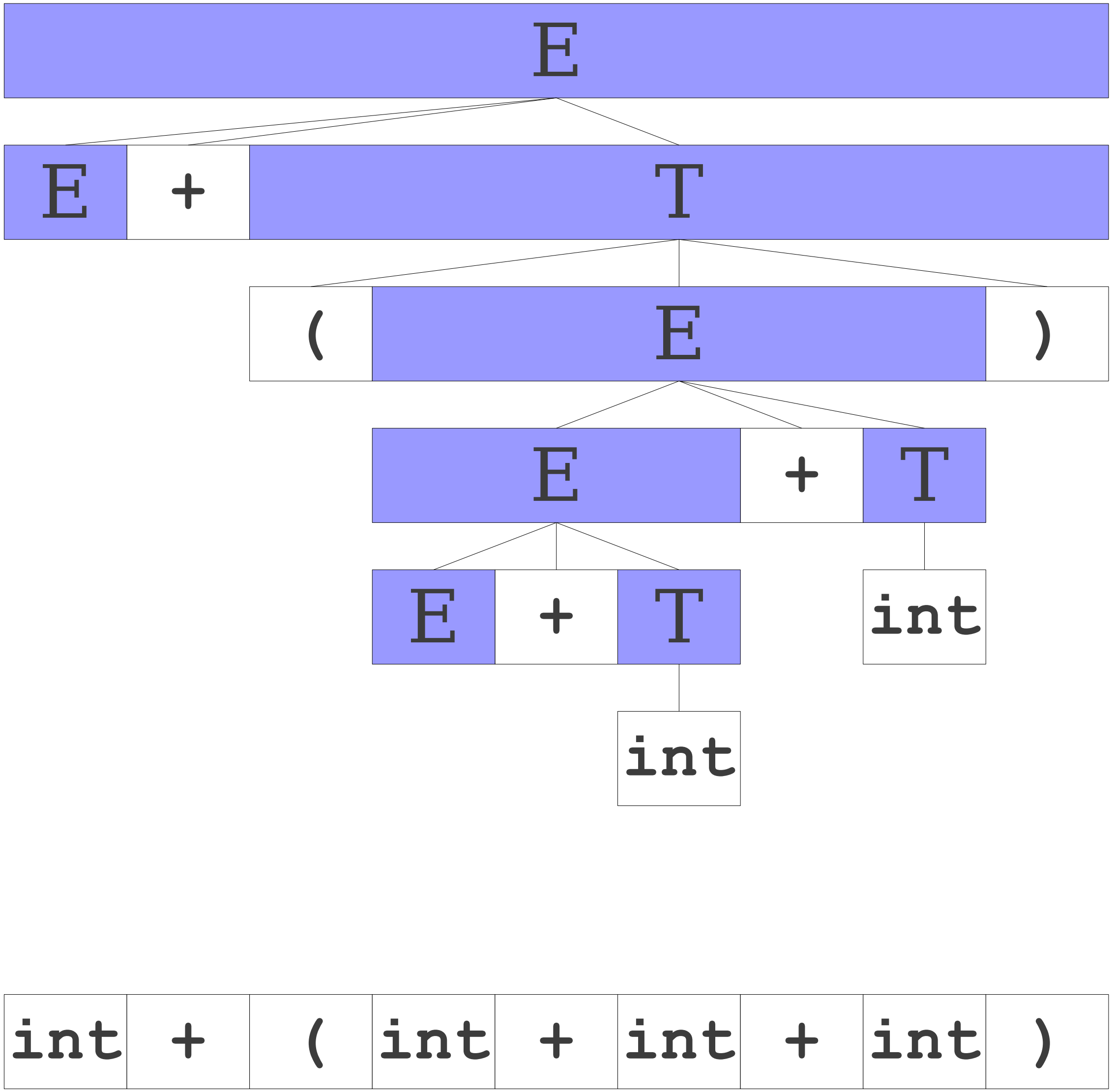


A Third View of a Bottom-Up Parse

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



A Third View of a Bottom-Up Parse



⇒ **E** `+ (`**E** `+ int + int)`

⇒ **E** `+ (`**E** `+` **T** `+ int)`

⇒ **E** `+ (`**E** `+ int)`

⇒ **E** `+ (`**E** `+` **T**`)`

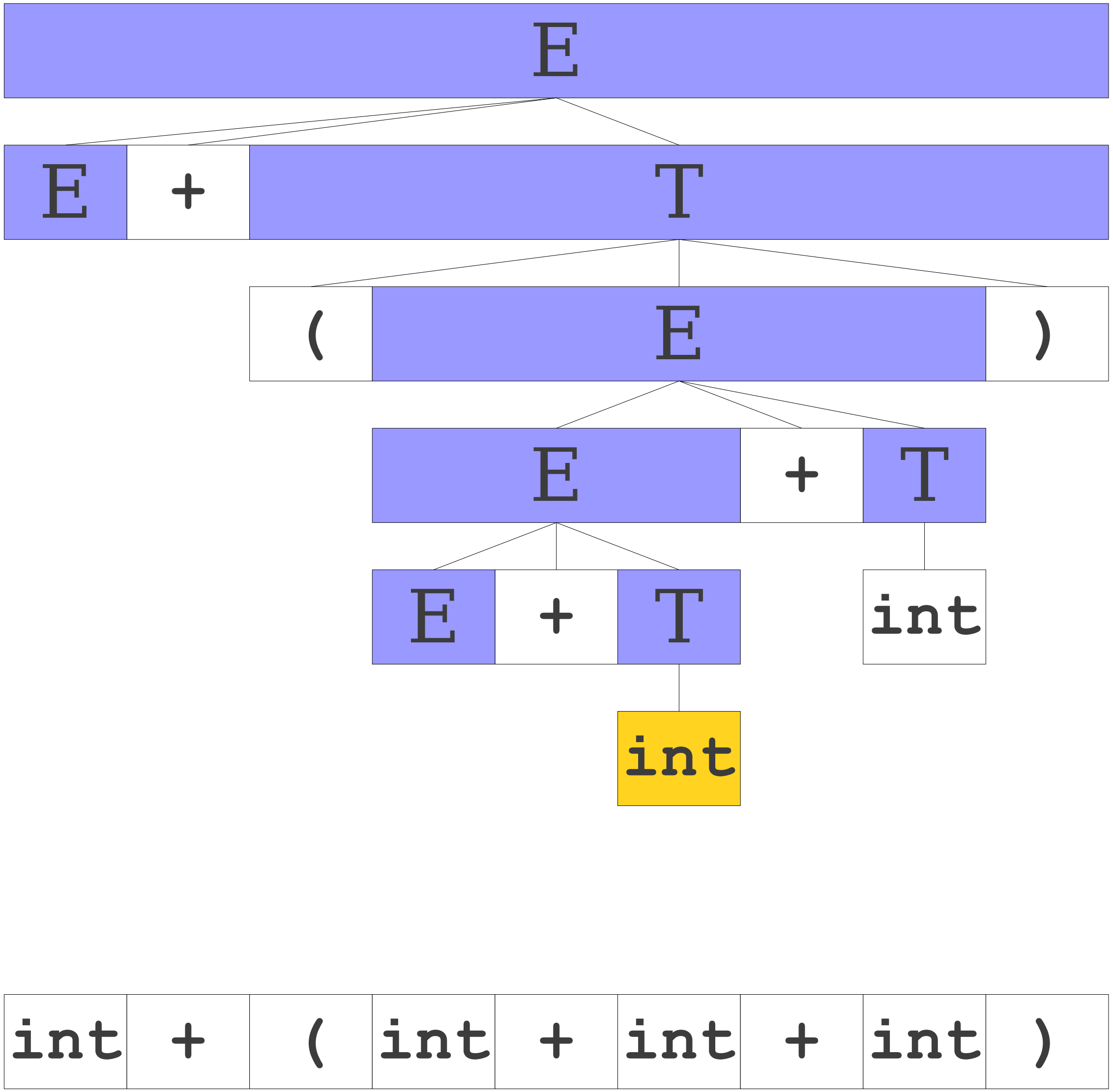
⇒ **E** `+ (`**E**`)`

⇒ **E** `+` **T**

⇒ **E**

A Third View of a Bottom-Up Parse

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



A Third View of a Bottom-Up Parse

⇒ **E + (E + T + int)**

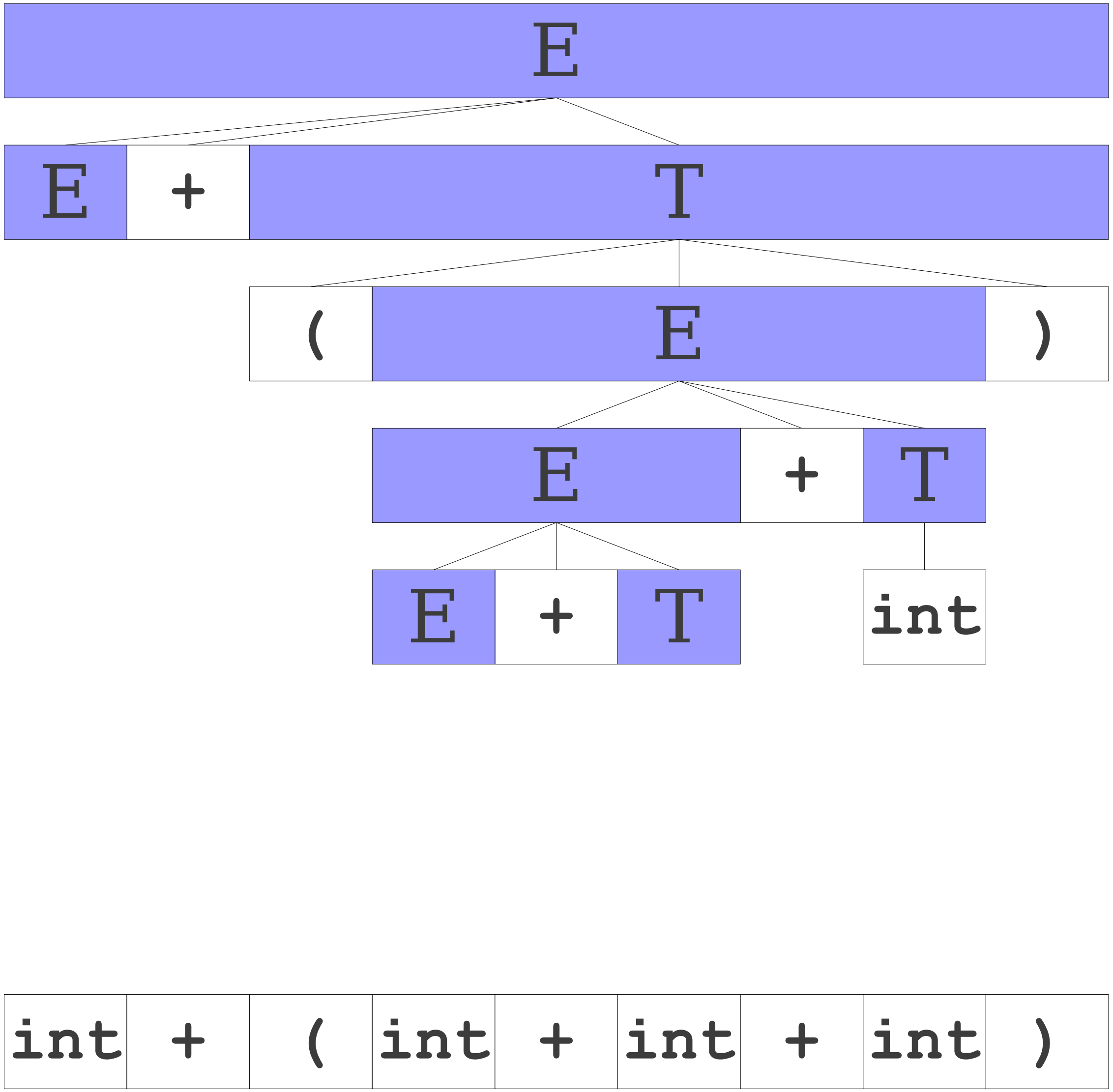
⇒ **E + (E + int)**

⇒ **E + (E + T)**

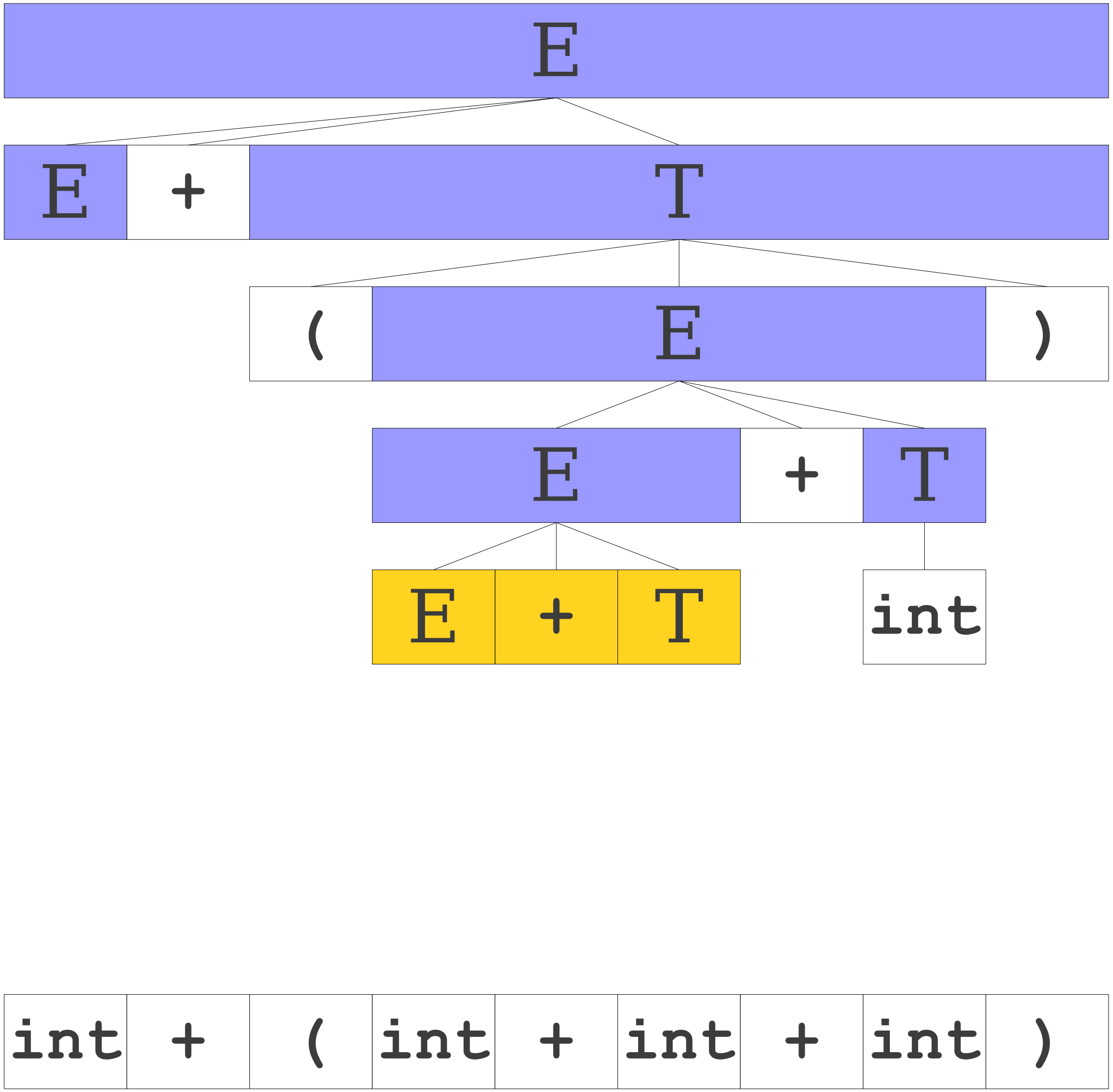
⇒ **E + (E)**

⇒ **E + T**

⇒ **E**

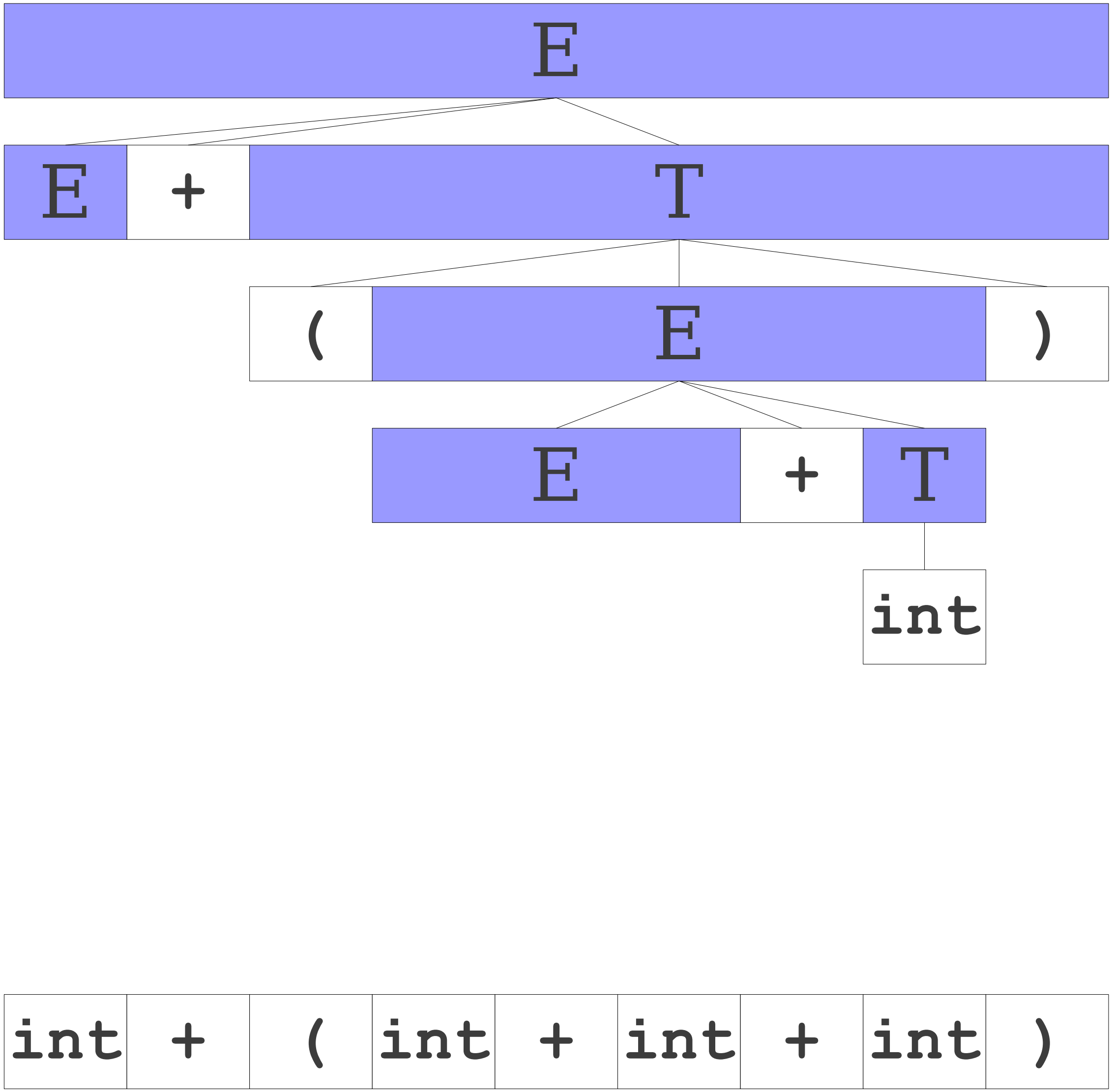


A Third View of a Bottom-Up Parse



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

A Third View of a Bottom-Up Parse



⇒ E + (E + int)

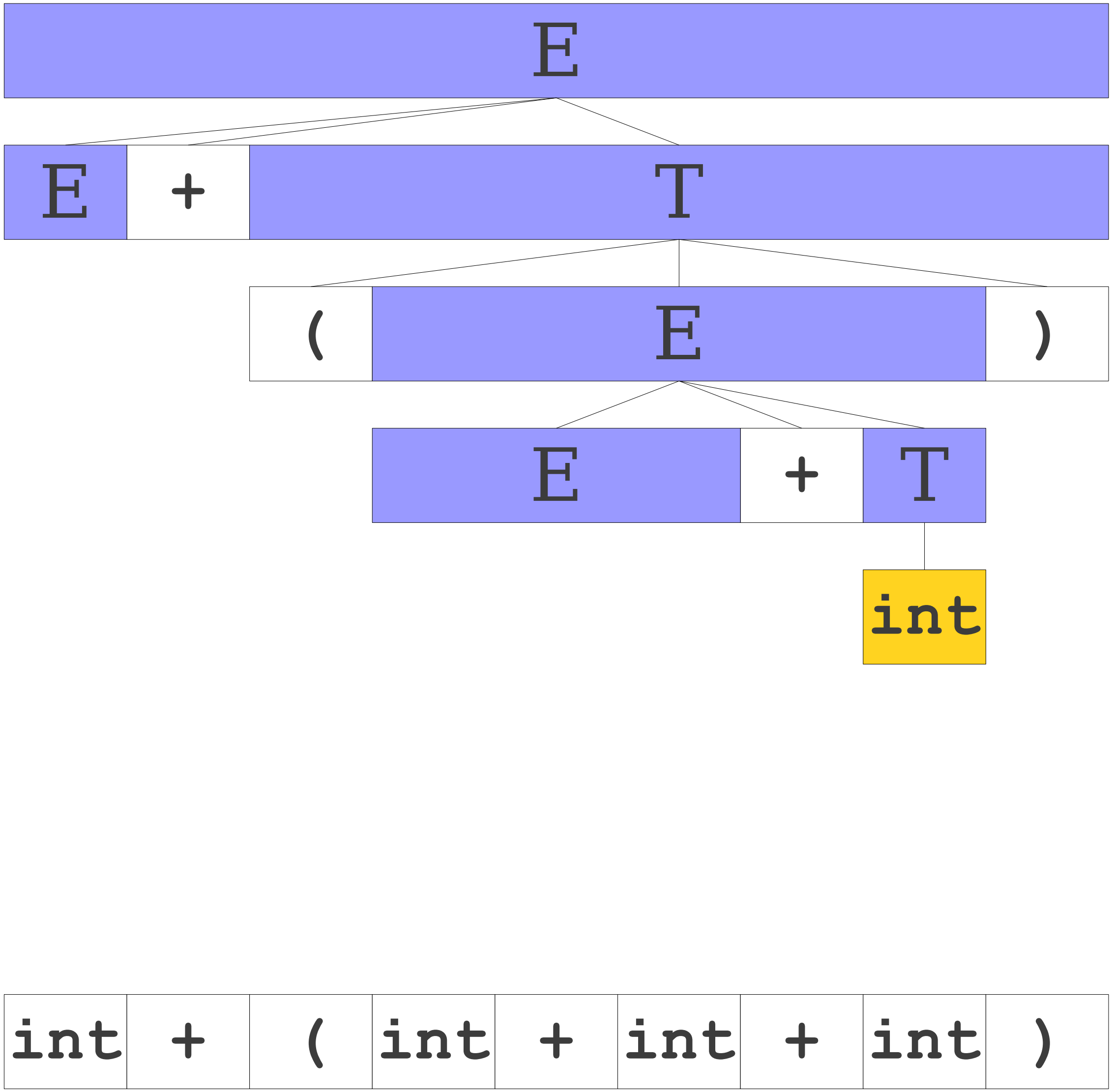
⇒ E + (E + T)

⇒ E + (E)

⇒ E + T

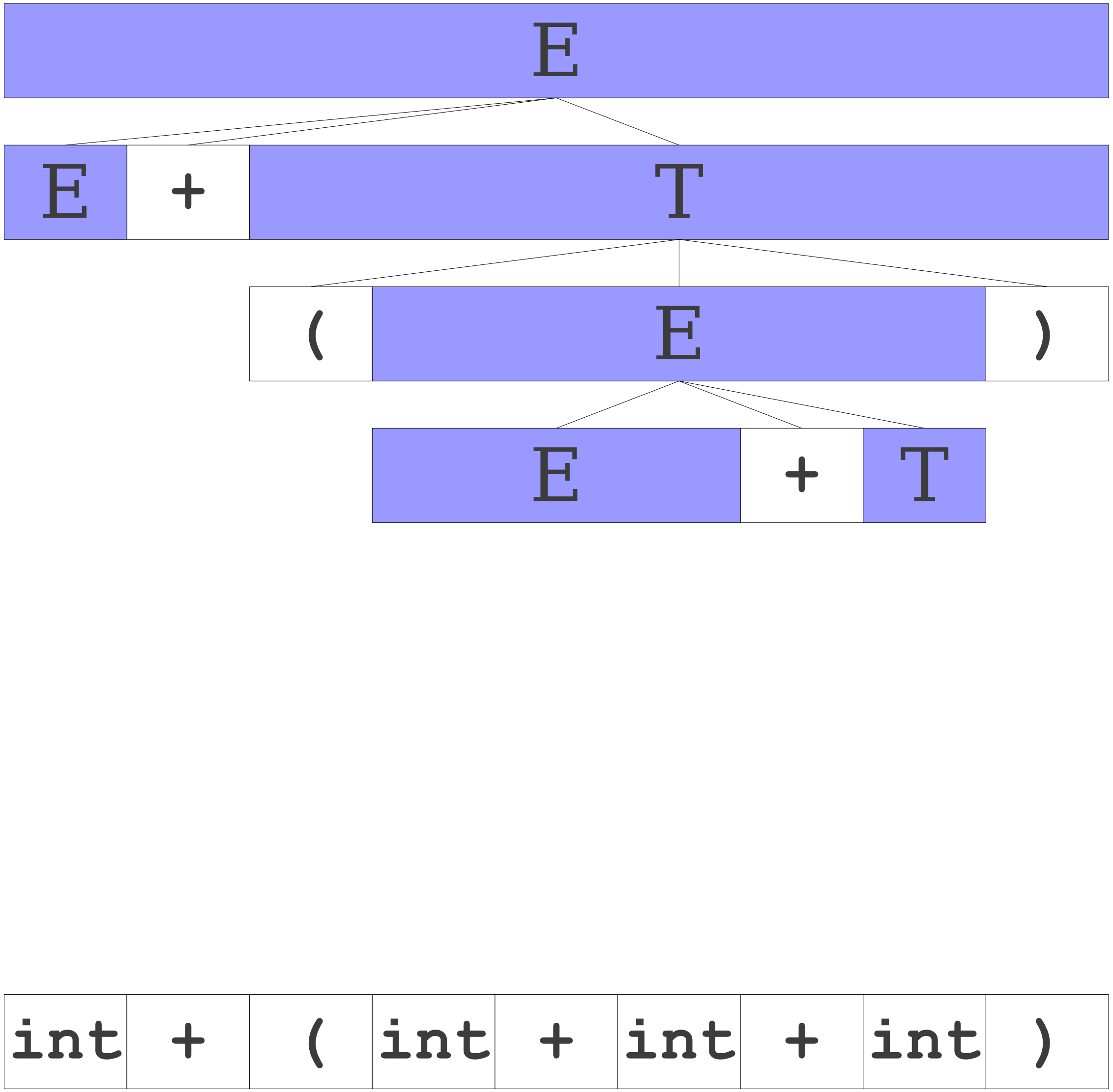
⇒ E

A Third View of a Bottom-Up Parse



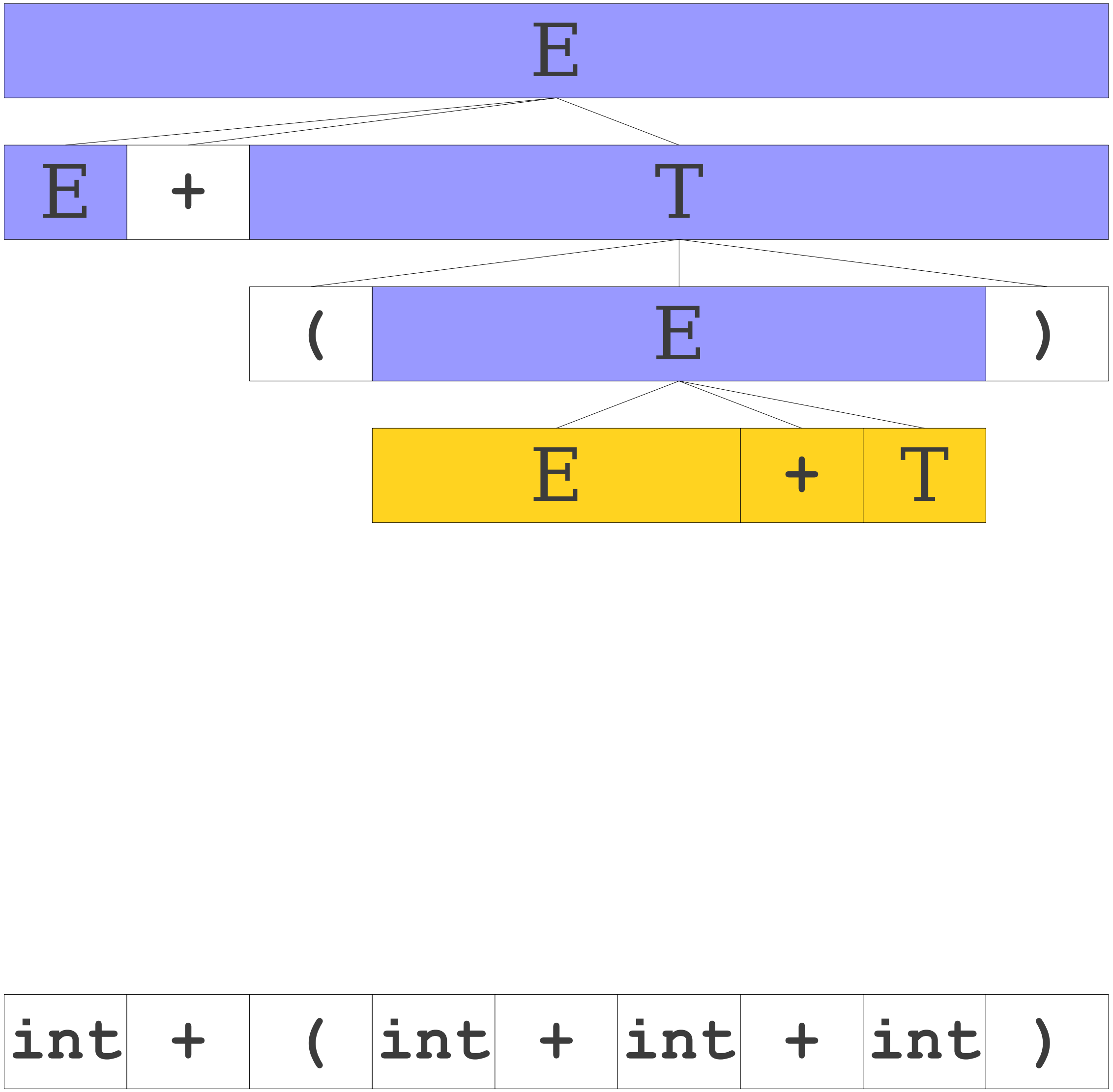
⇒ **E** + (**E** + `int`)
⇒ **E** + (**E** + **T**)
⇒ **E** + (**E**)
⇒ **E** + **T**
⇒ **E**

A Third View of a Bottom-Up Parse



⇒ E + (E + T)
⇒ E + (E)
⇒ E + T
⇒ E

A Third View of a Bottom-Up Parse



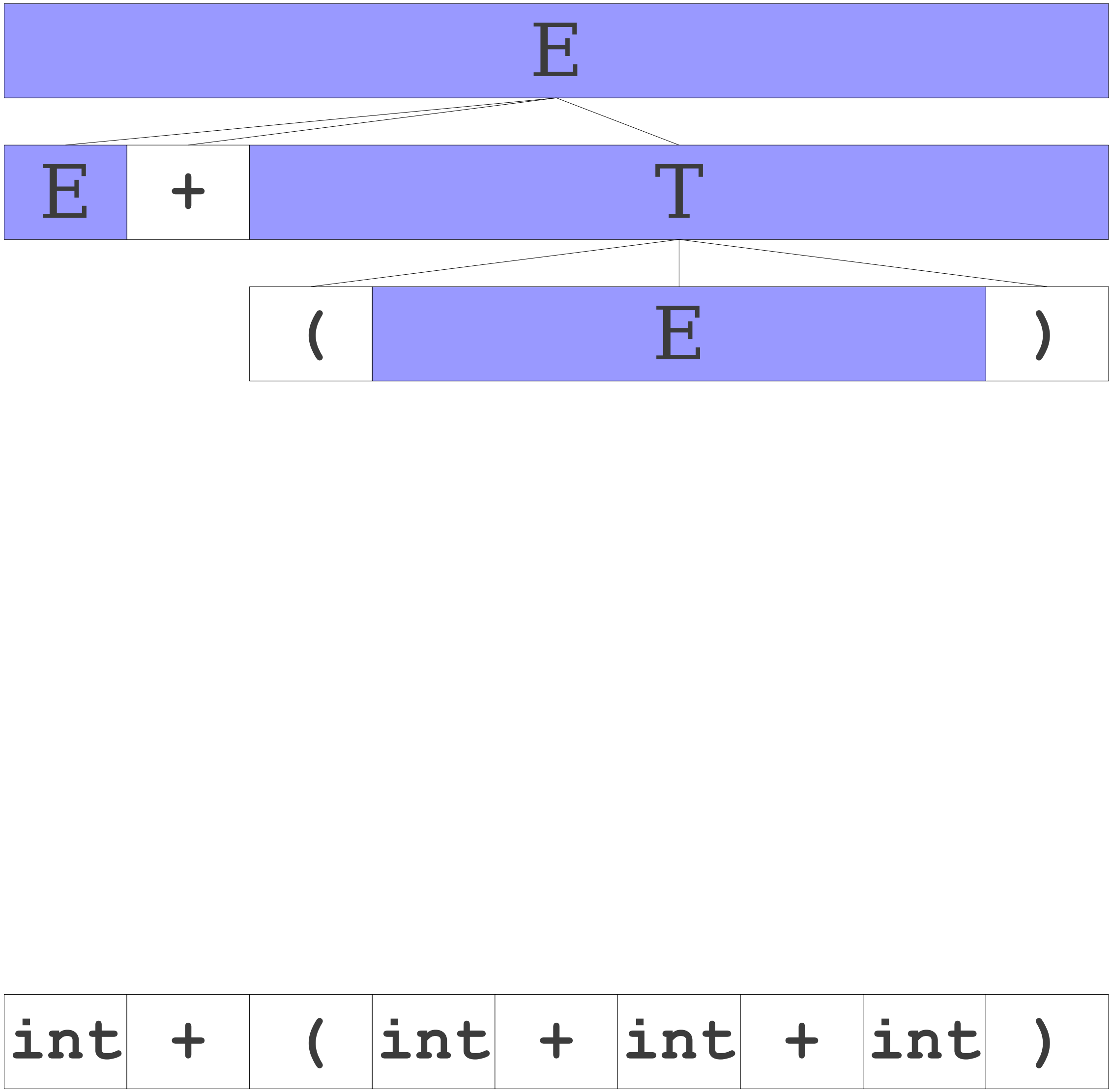
⇒ **E + (E + T)**
⇒ **E + (E)**
⇒ **E + T**
⇒ **E**

A Third View of a Bottom-Up Parse

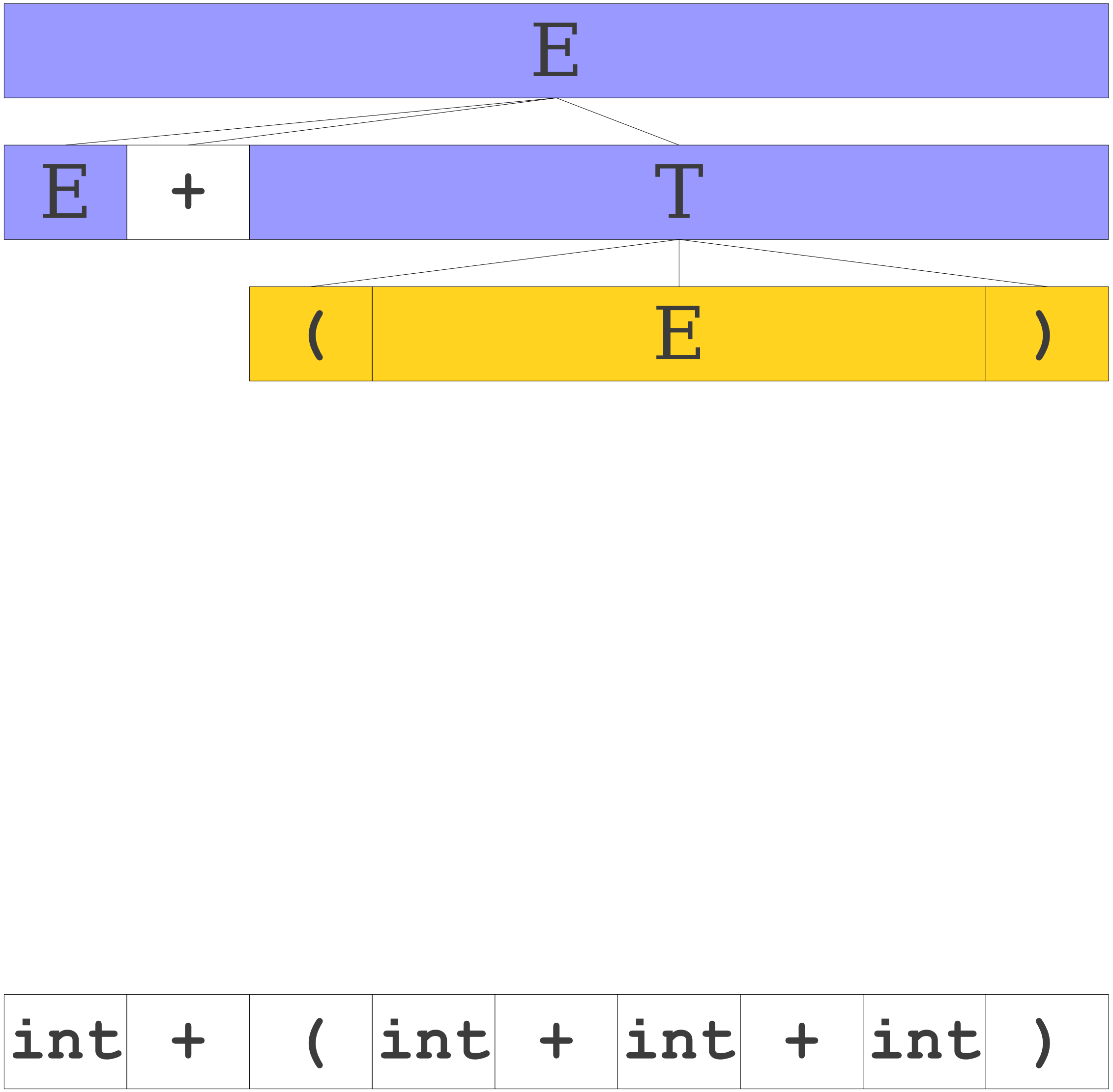
⇒ **E + (E)**

⇒ **E + T**

⇒ **E**



A Third View of a Bottom-Up Parse

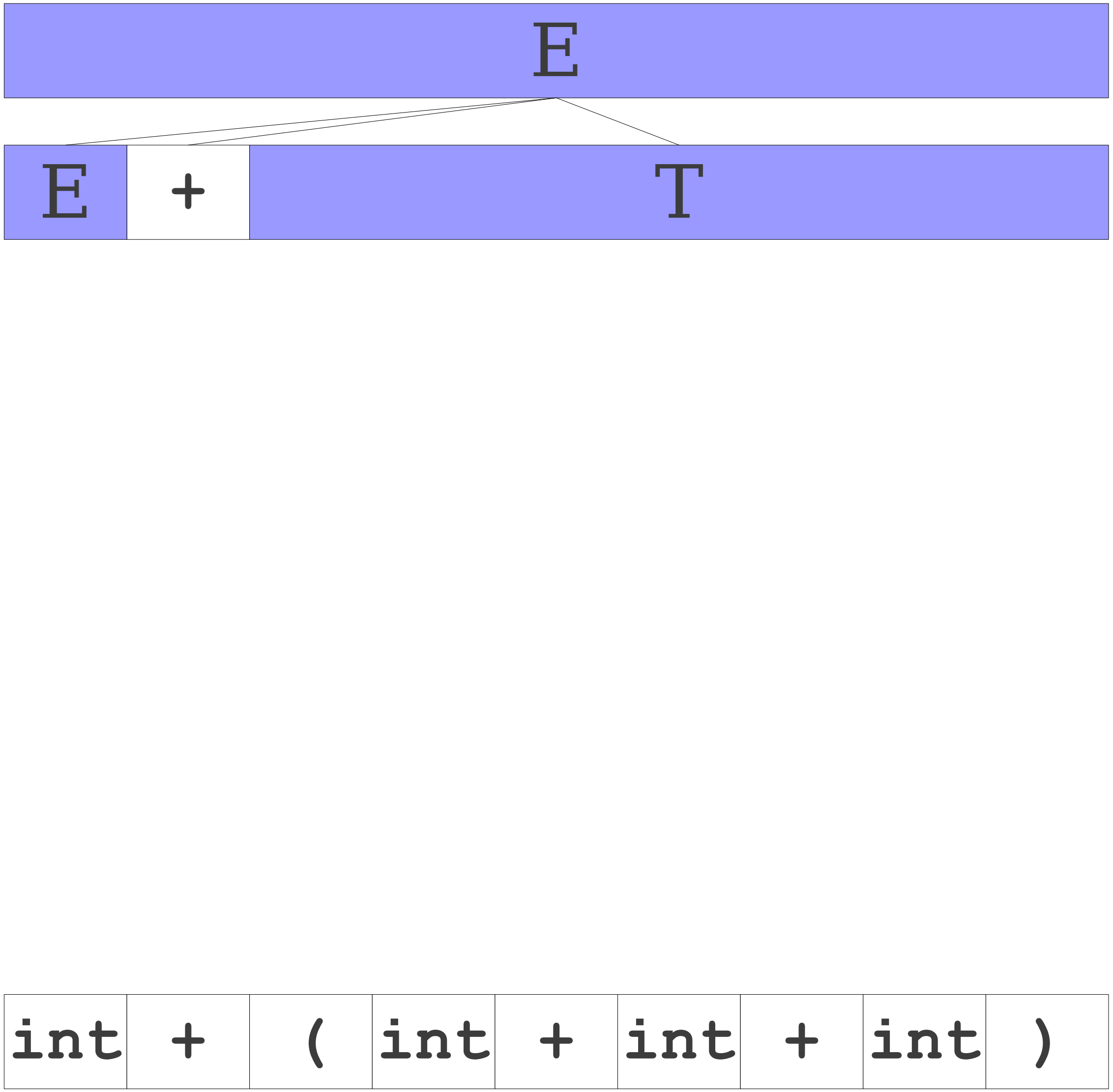


⇒ **E + (E)**

⇒ **E + T**

⇒ **E**

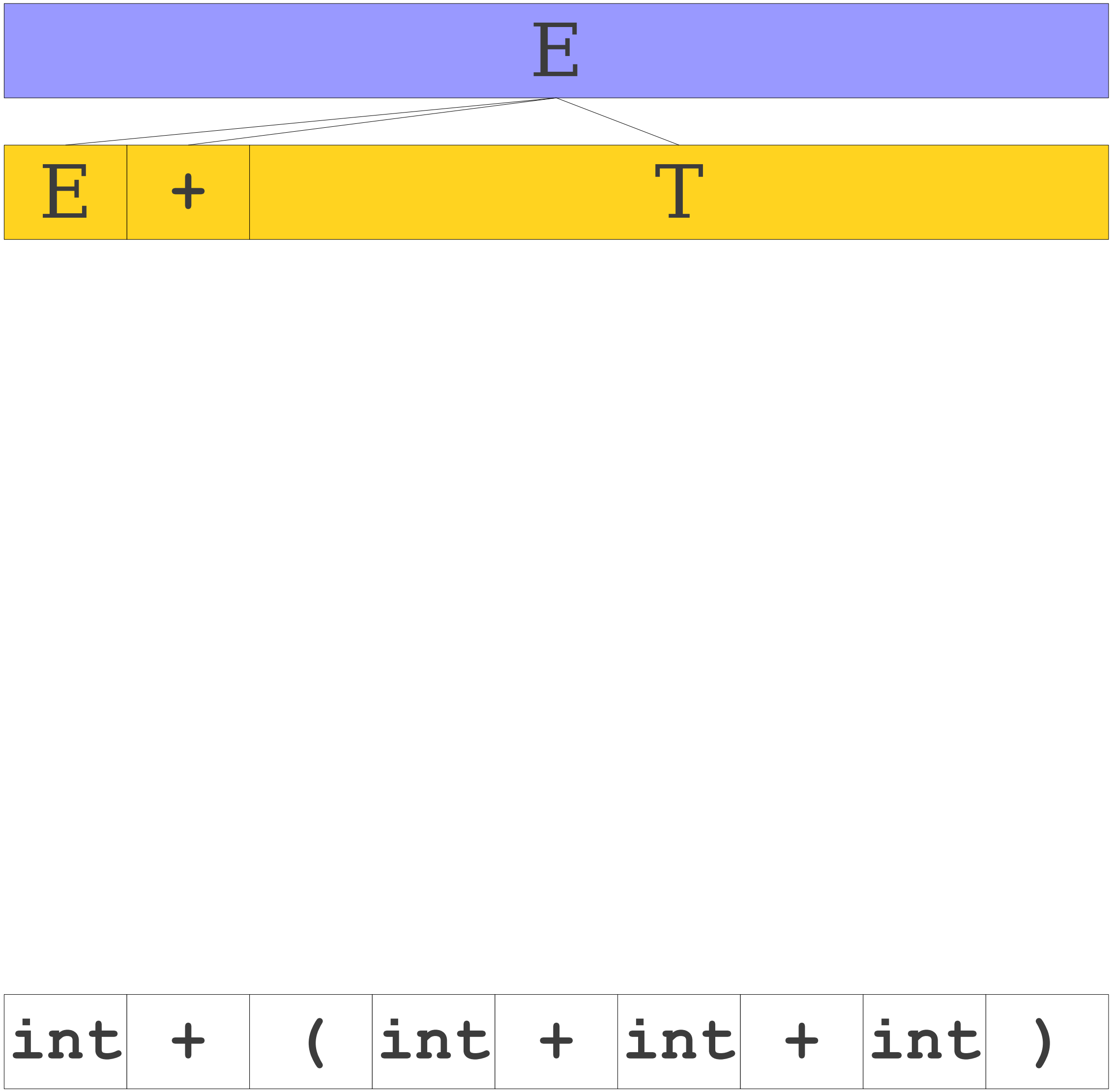
A Third View of a Bottom-Up Parse



⇒ **E + T**

⇒ **E**

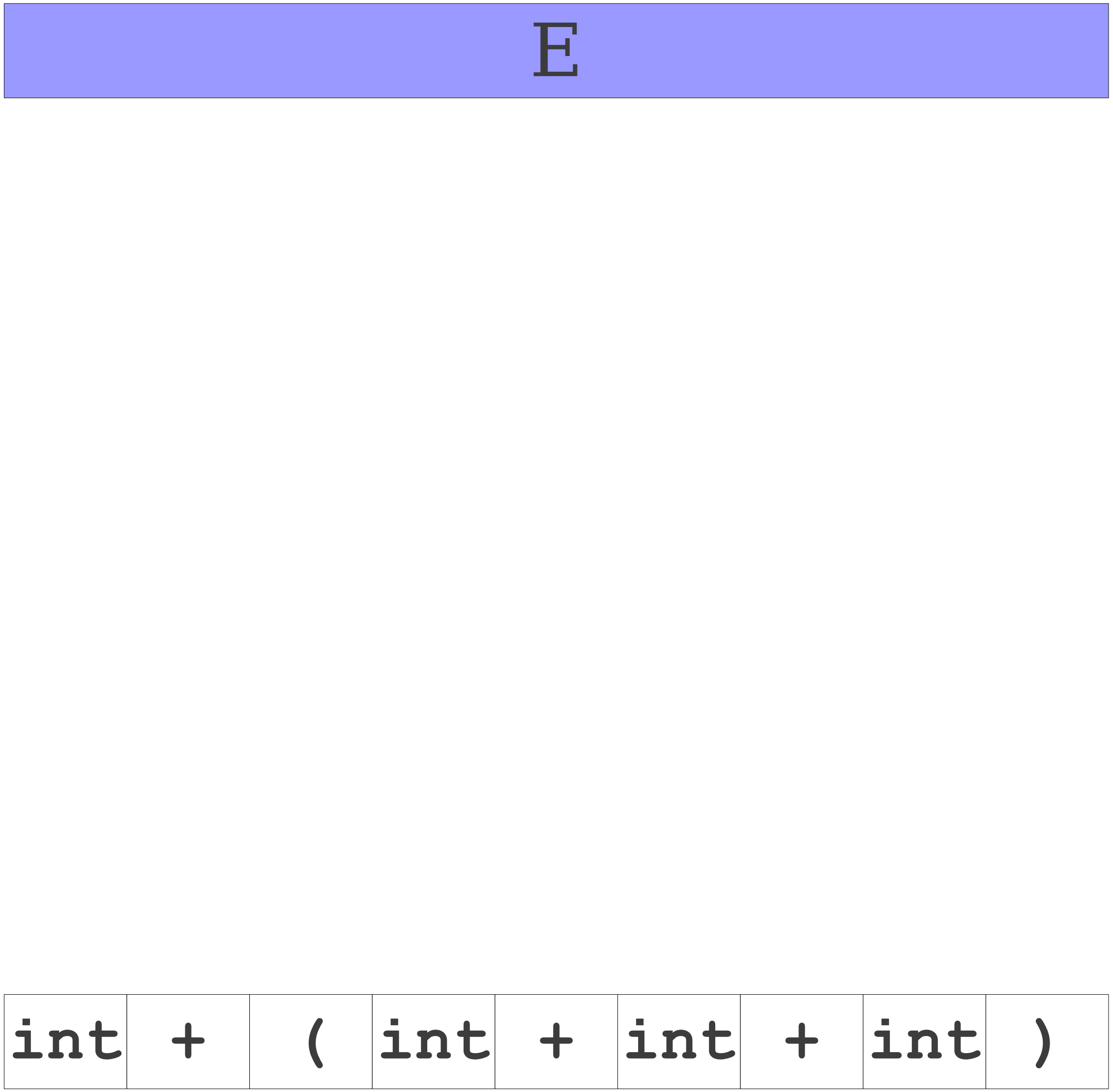
A Third View of a Bottom-Up Parse



⇒ **E + T**

⇒ **E**

A Third View of a Bottom-Up Parse



⇒ **E**

Handles

- The **handle** of a parse tree *T* is the leftmost complete cluster of leaf nodes.
- A left-to-right, bottom-up parse works by iteratively searching for a handle, then reducing the handle.

Summarizing Our Intuition

- Our first intuition (reconstructing the parse tree bottom-up) motivates how the parsing should work.
- Our second intuition (rightmost derivation in reverse) describes the order in which we should build the parse tree.
- Our third intuition (handle pruning) is the basis for the bottom-up parsing algorithms we will explore.

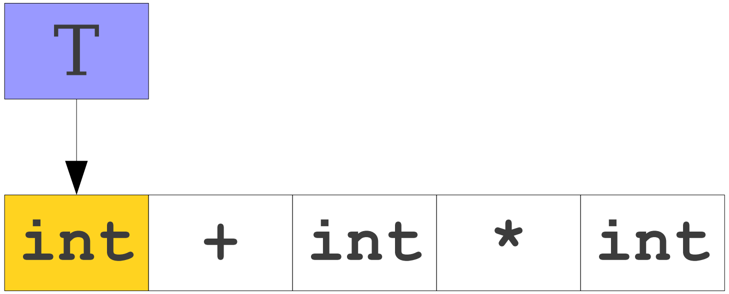
A Detail about Handles

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

int	+	int	*	int

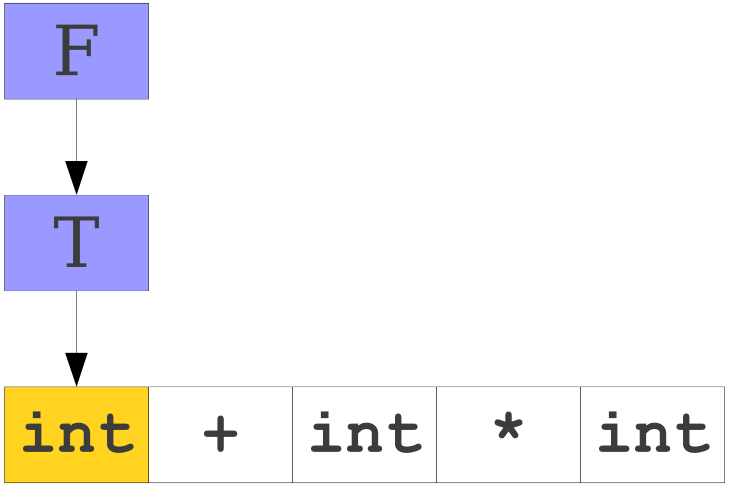
A Detail about Handles

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



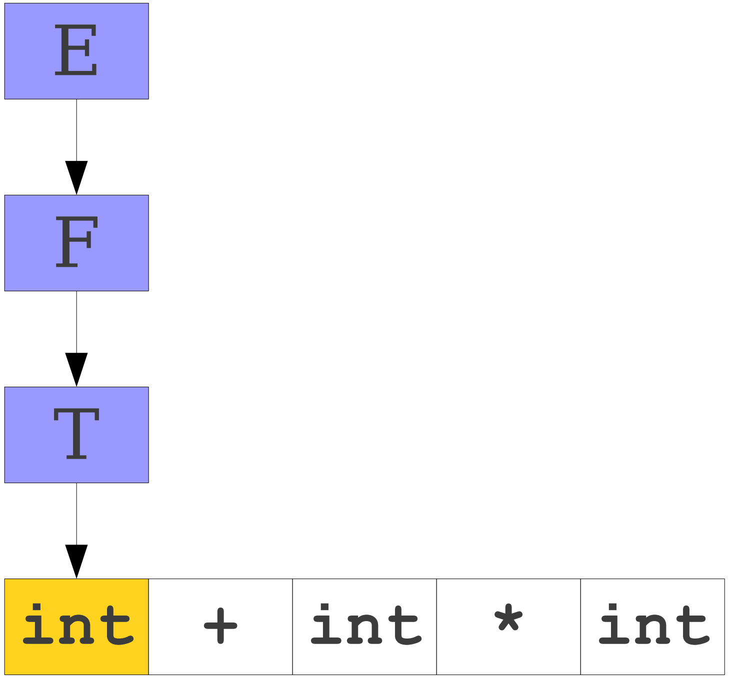
A Detail about Handles

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



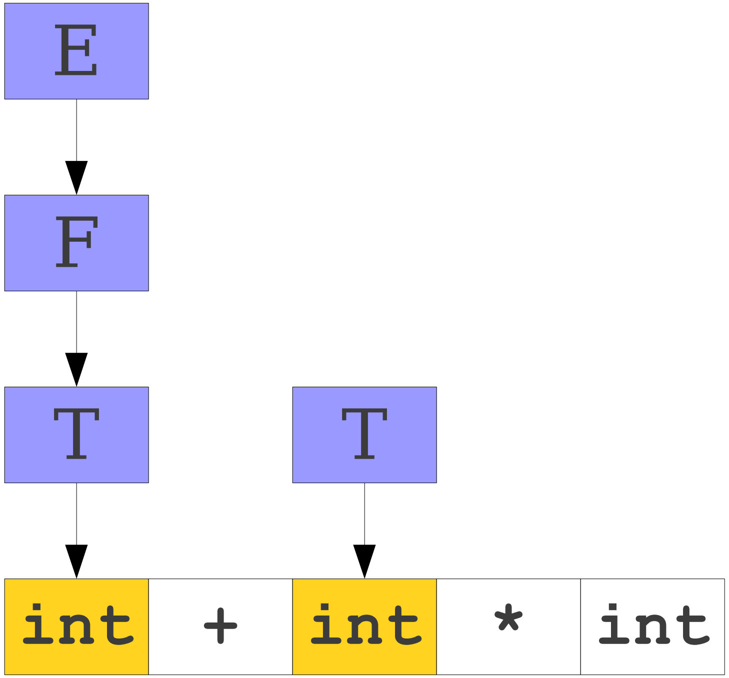
A Detail about Handles

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



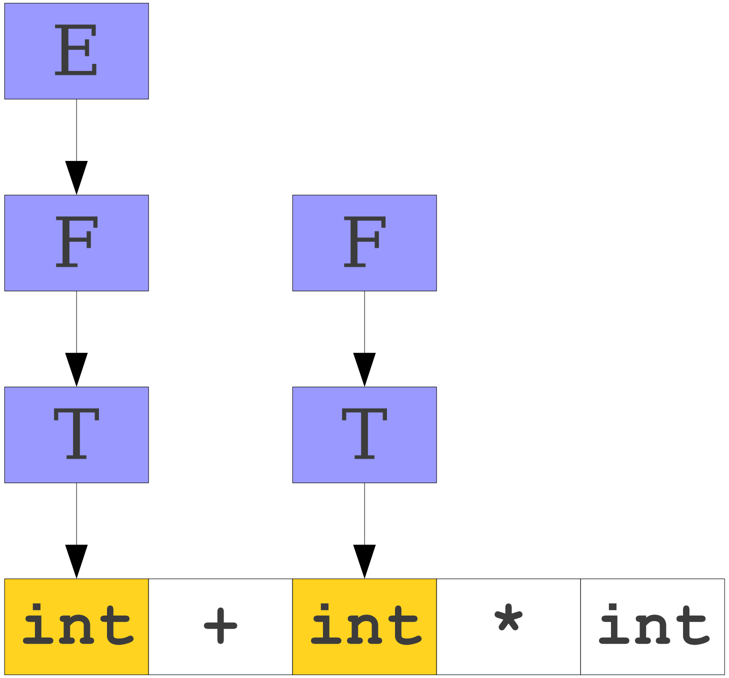
A Detail about Handles

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



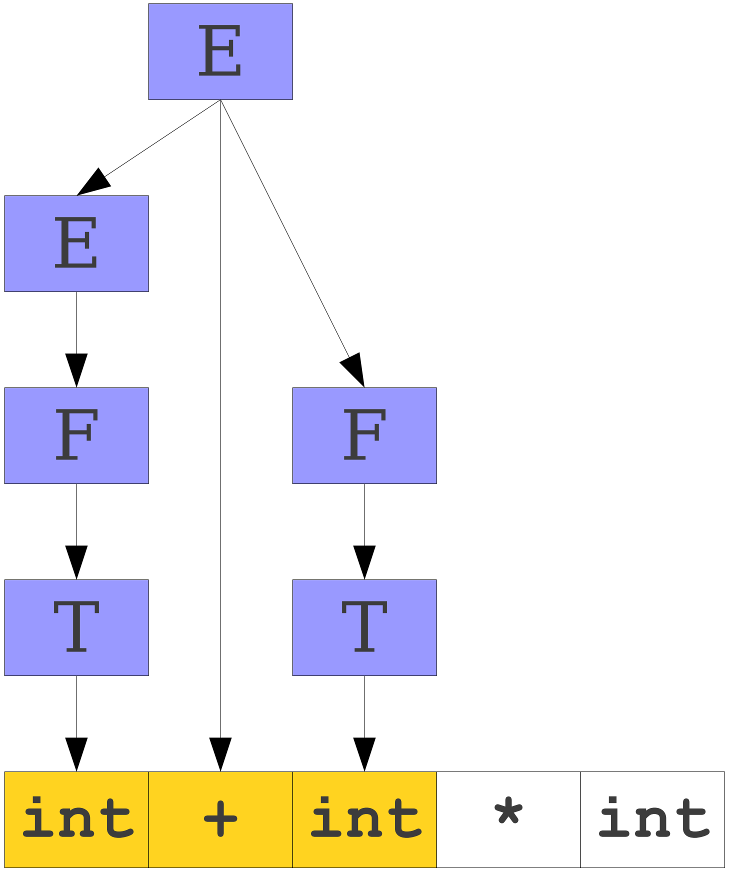
A Detail about Handles

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



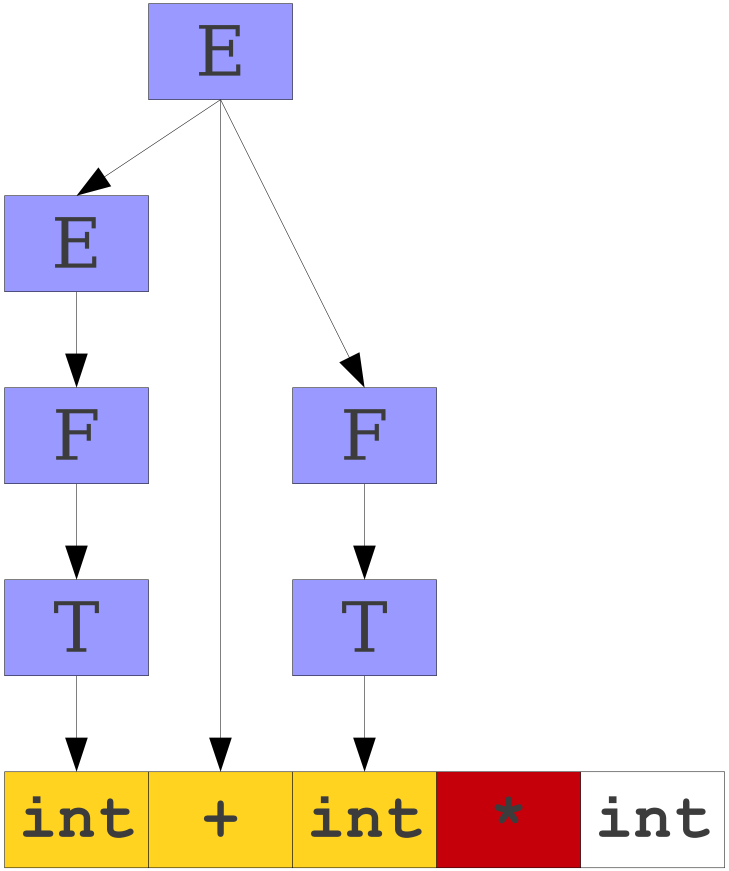
A Detail about Handles

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



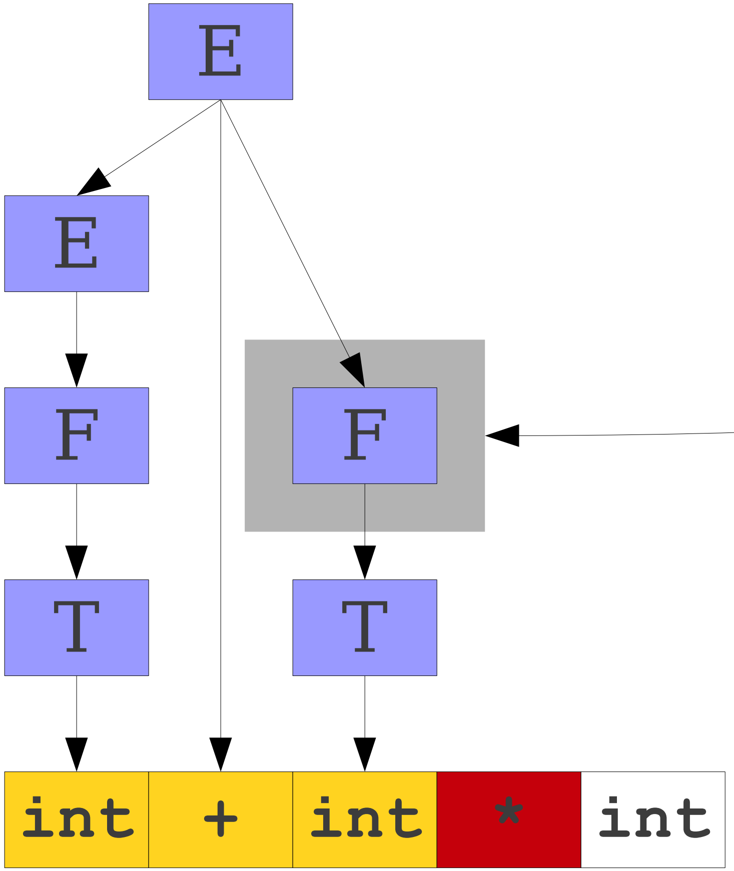
A Detail about Handles

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



A Detail about Handles

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



This reduction wasn't a handle!

The leftmost reduction
isn't always the handle.

Finding Handles

- Where do we look for handles?
  - Where in the string might the handle be?
- How do we search for possible handles?
  - Once we know where to search, how do we identify candidate handles?
- How do we recognize handles?
  - Once we've found a candidate handle, how do we check that it really is the handle?

Question One:

Where are handles?

Where are Handles?

- Recall: A left-to-right, bottom-up parse traces a rightmost derivation in reverse.
- Each time we do a reduction, we are reversing a production applied to the *rightmost* nonterminal symbol.
- Suppose that our current sentential form is **$\alpha\gamma\omega$**, where **γ** is the handle and **A** → **γ** is a production rule.
- After reducing **γ** back to **A**, we have the string **$\alpha A\omega$**.
- Thus **ω** must consist purely of terminals, since otherwise the reduction we just did was not for the rightmost terminal.

Why This Matters

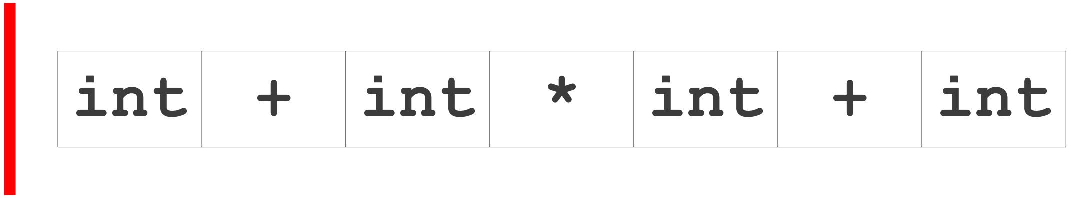
- Suppose we want to parse the string **γ**.
- We will break **γ** into two parts, **α** and **ω**, where
  - **α** consists of both terminals and nonterminals, and
  - **ω** consists purely of terminals.
- Our search for handles will concentrate purely in **α**.
- As necessary, we will start moving terminals from **ω** over into **α**.

Shift/Reduce Parsing

- The bottom-up parsers we will consider are called **shift/reduce** parsers.
  - Contrast with the LL(1) **predict/match** parser.
- Idea: Split the input into two parts:
  - Left substring is our work area; all handles must be here.
  - Right substring is input we have not yet processed; consists purely of terminals.
- At each point, decide whether to:
  - Move a terminal across the split (**shift**)
  - Reduce a handle (**reduce**)

A Sample Shift/Reduce Parse

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



int	+	int	*	int	+	int

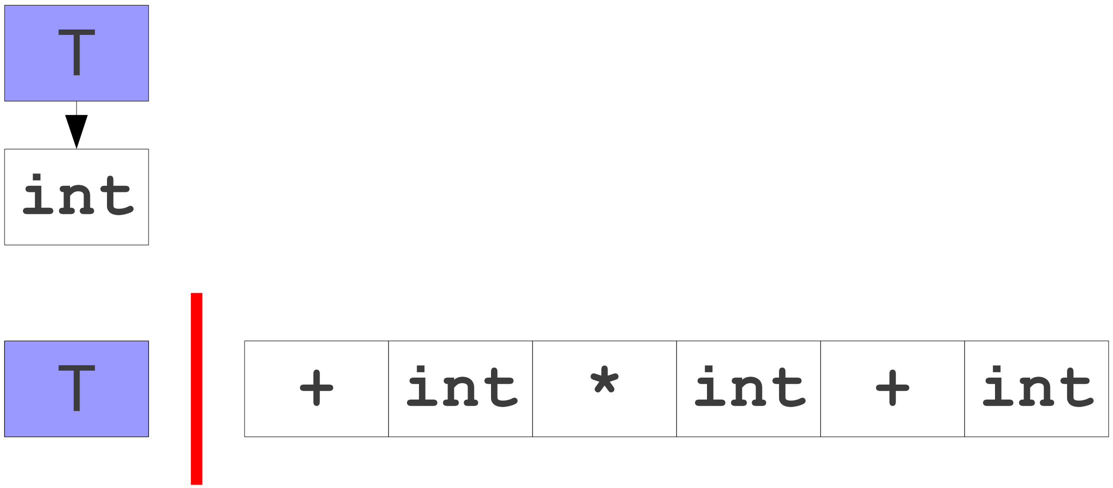
A Sample Shift/Reduce Parse

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

int | + int * int + int

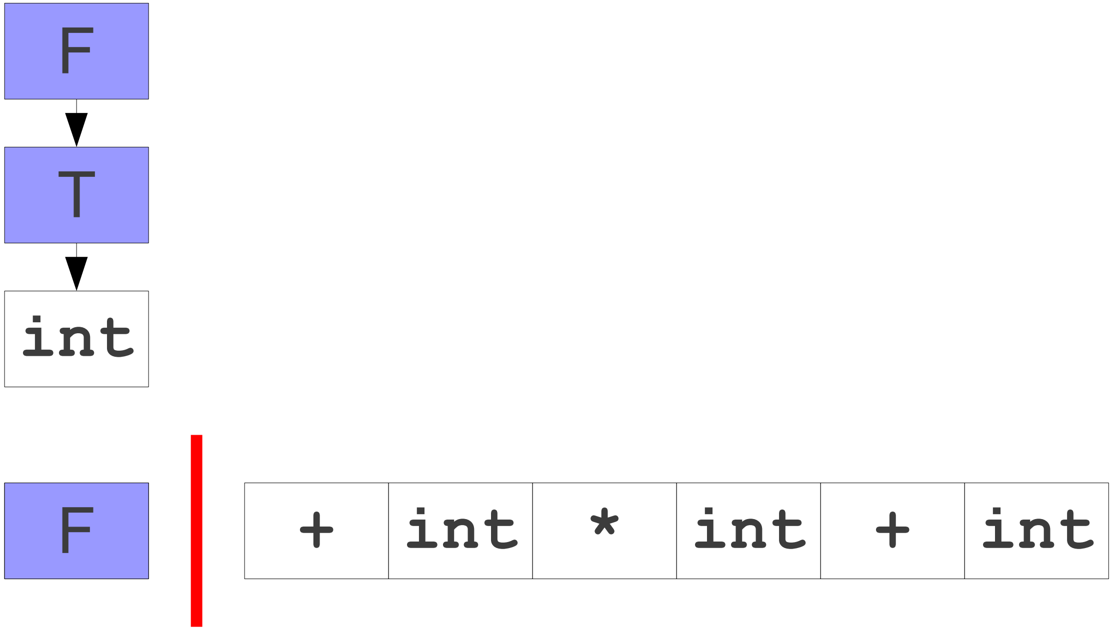
A Sample Shift/Reduce Parse

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



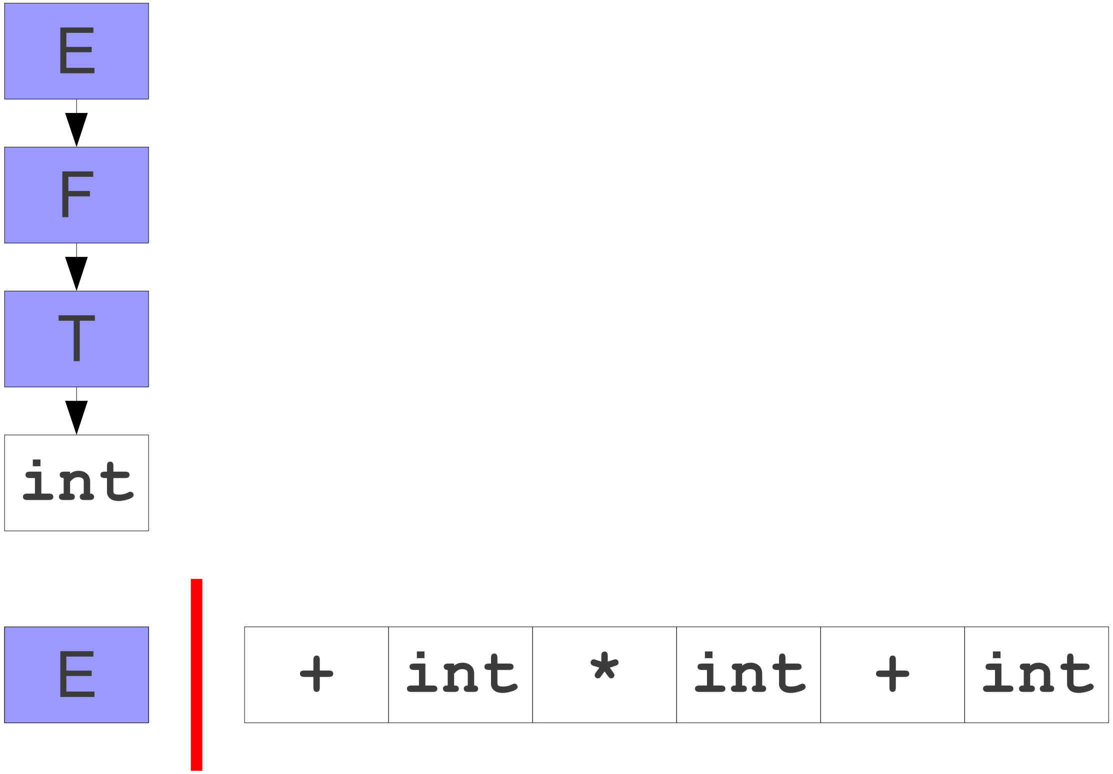
A Sample Shift/Reduce Parse

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



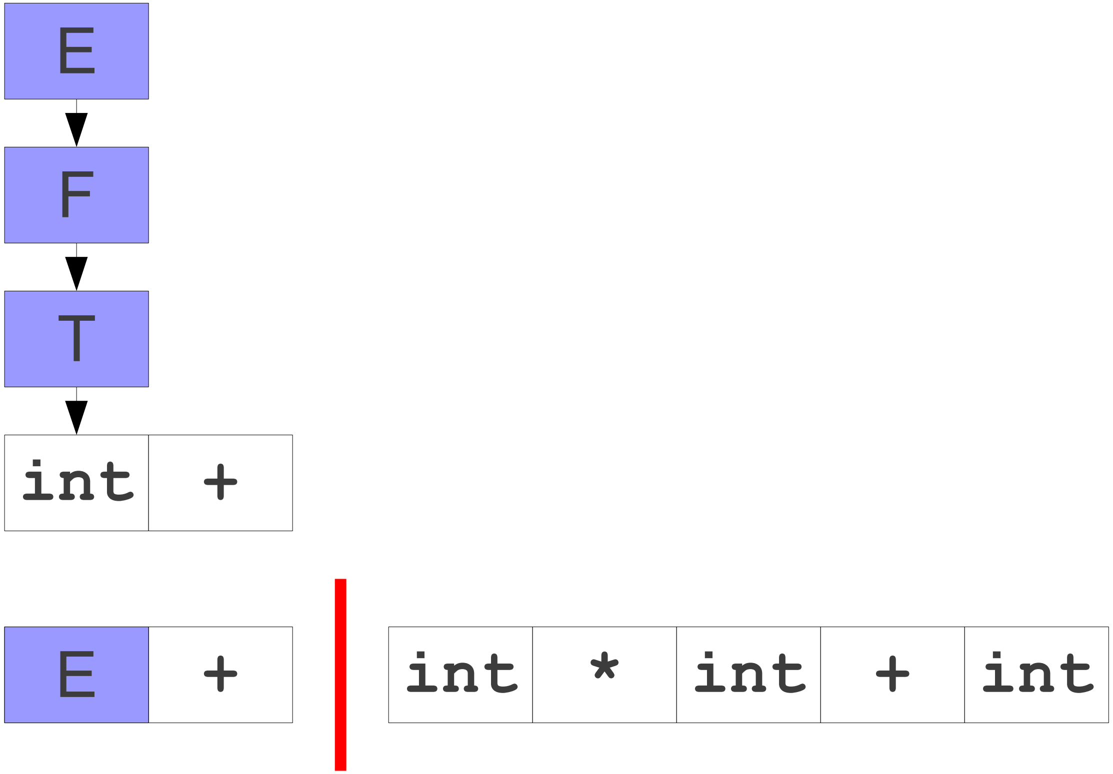
A Sample Shift/Reduce Parse

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



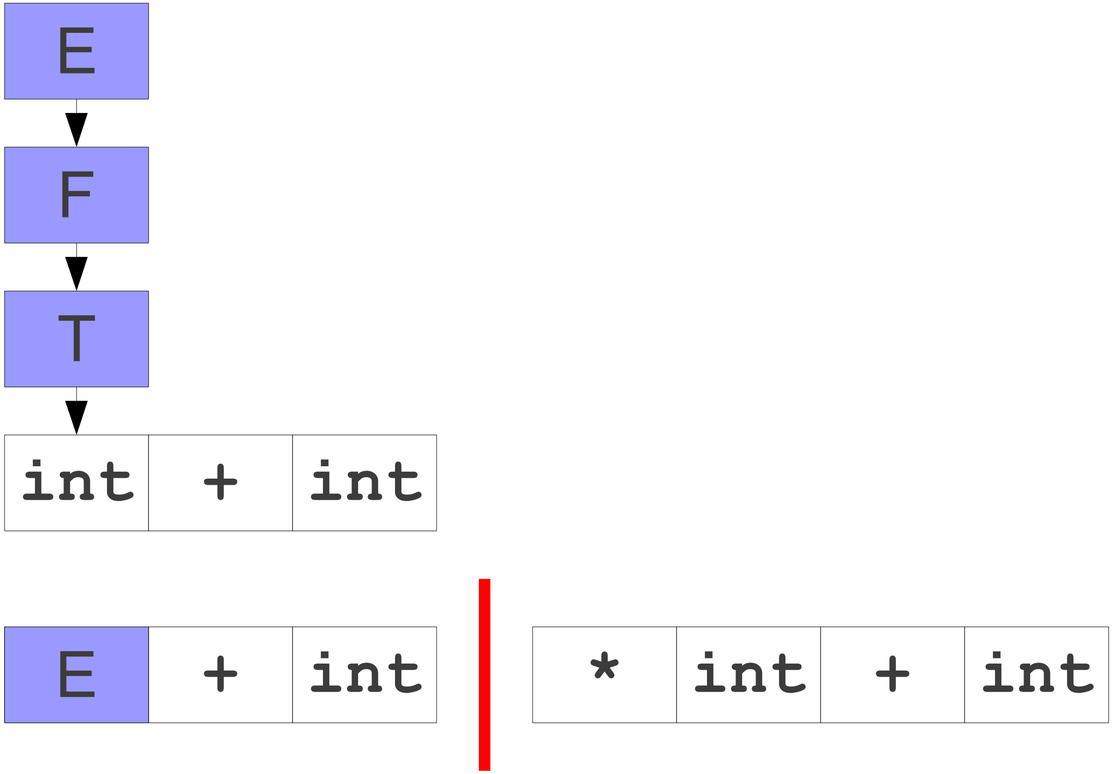
A Sample Shift/Reduce Parse

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



A Sample Shift/Reduce Parse

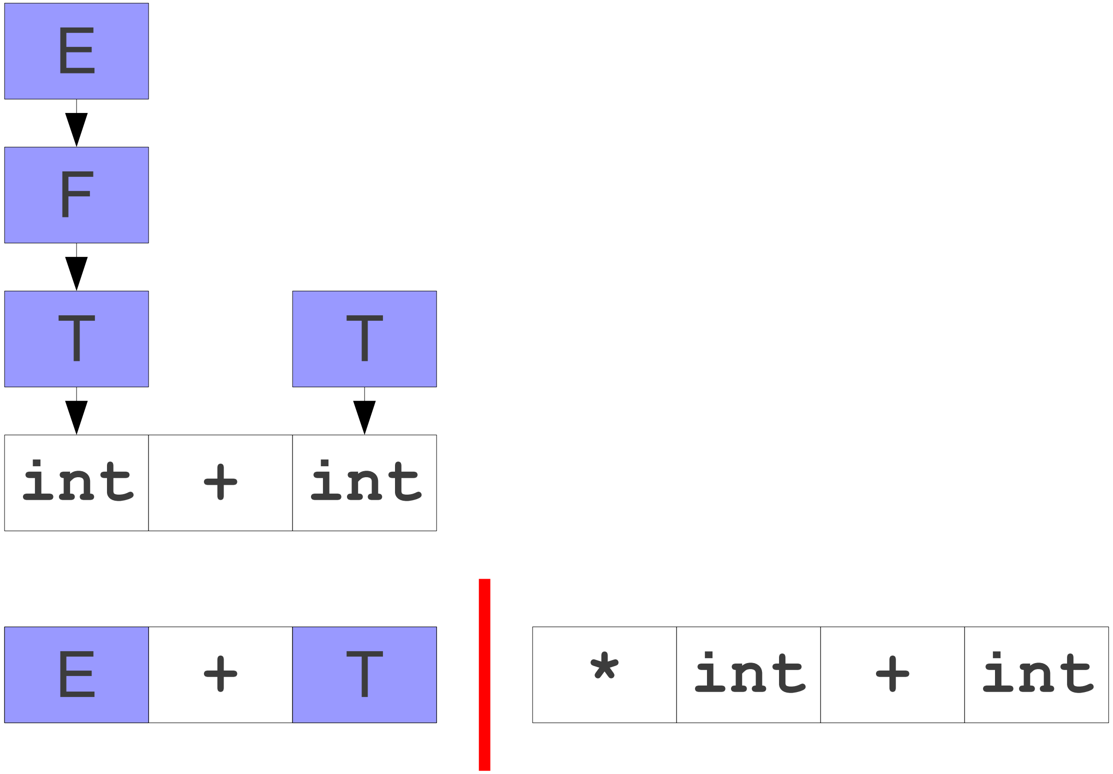
E → **F**
E → **E** + **F**
F → **F** * **T**
F → **T**
T → `int`
T → (**E**)


E
F
T
int + int

E + int | * int + int

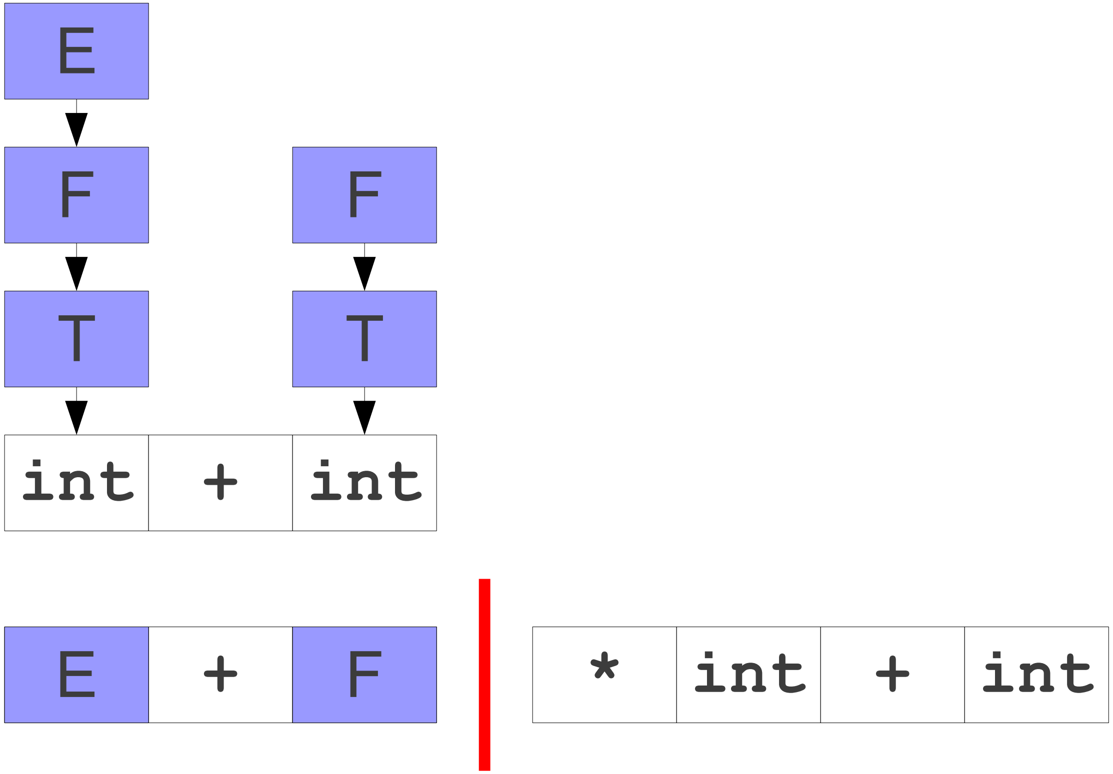
A Sample Shift/Reduce Parse

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



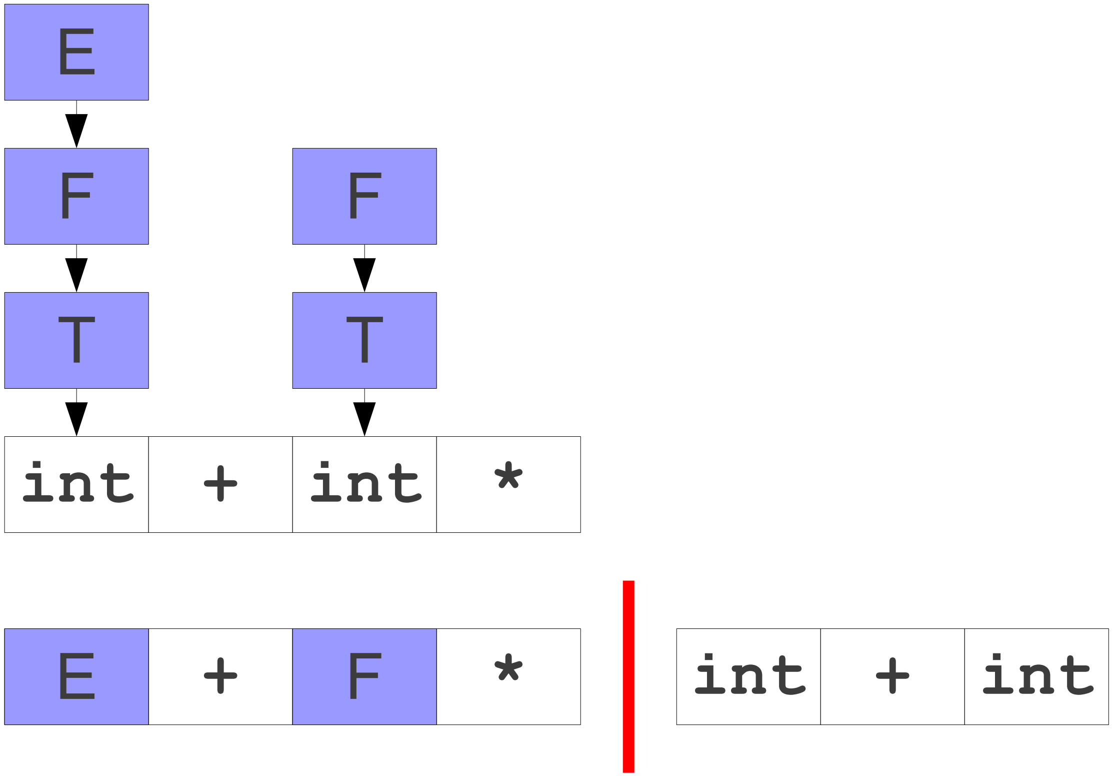
A Sample Shift/Reduce Parse

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



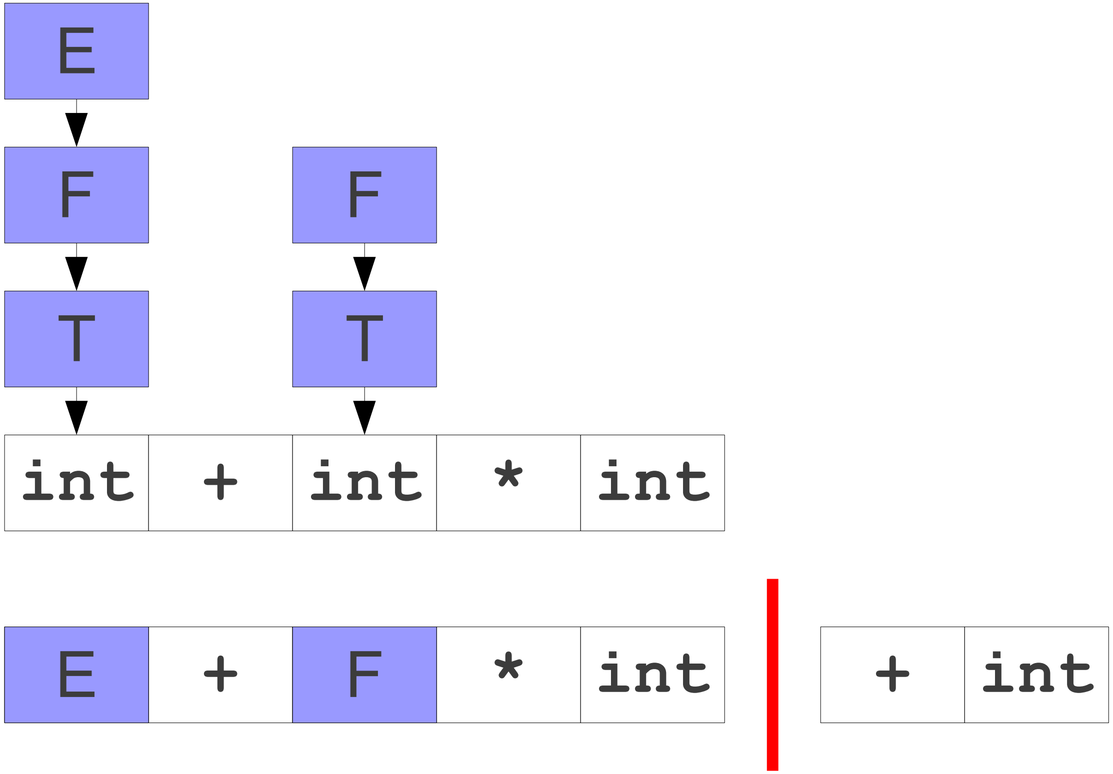
A Sample Shift/Reduce Parse

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**)

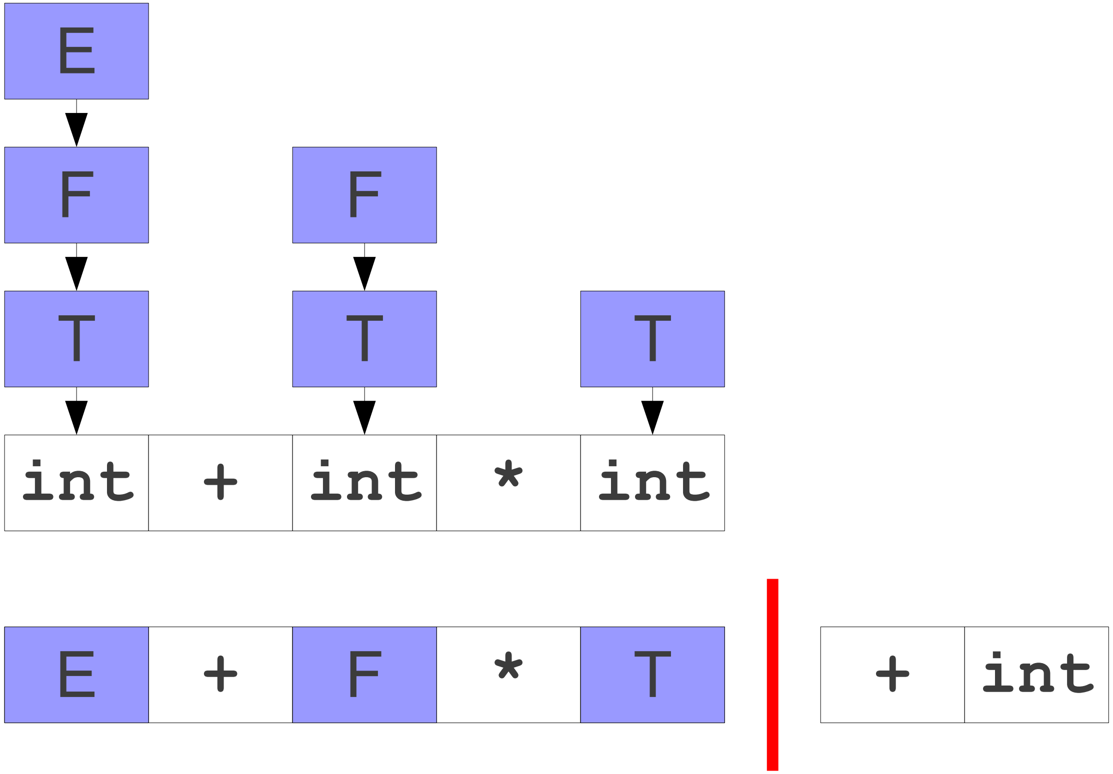


E
F F
T T
int + int * int

E + F * int | + int

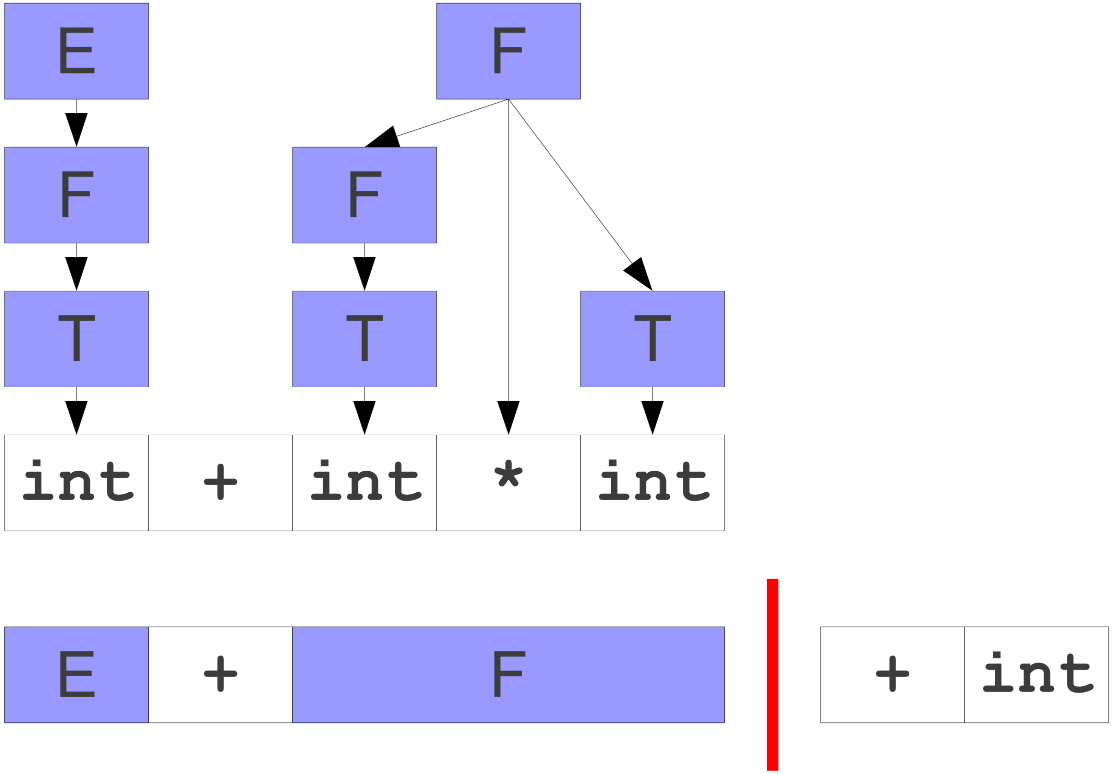
A Sample Shift/Reduce Parse

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



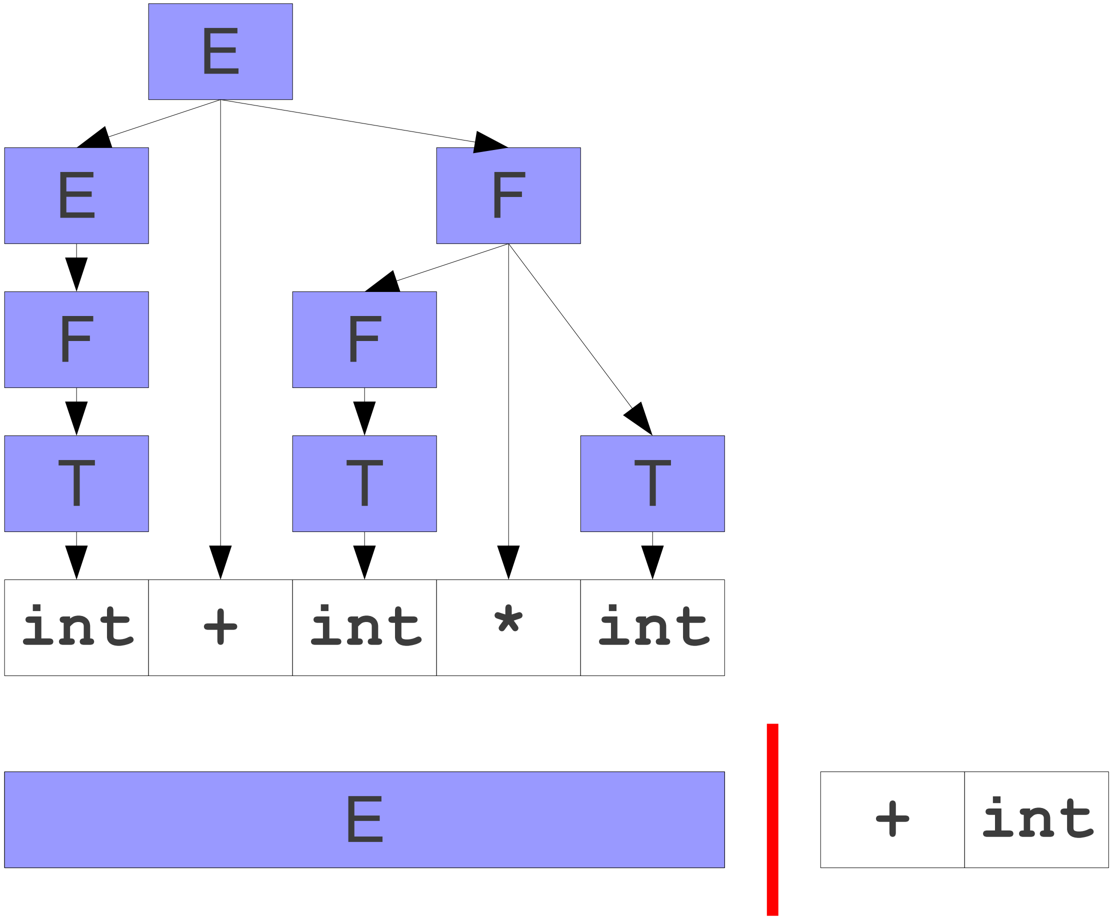
A Sample Shift/Reduce Parse

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



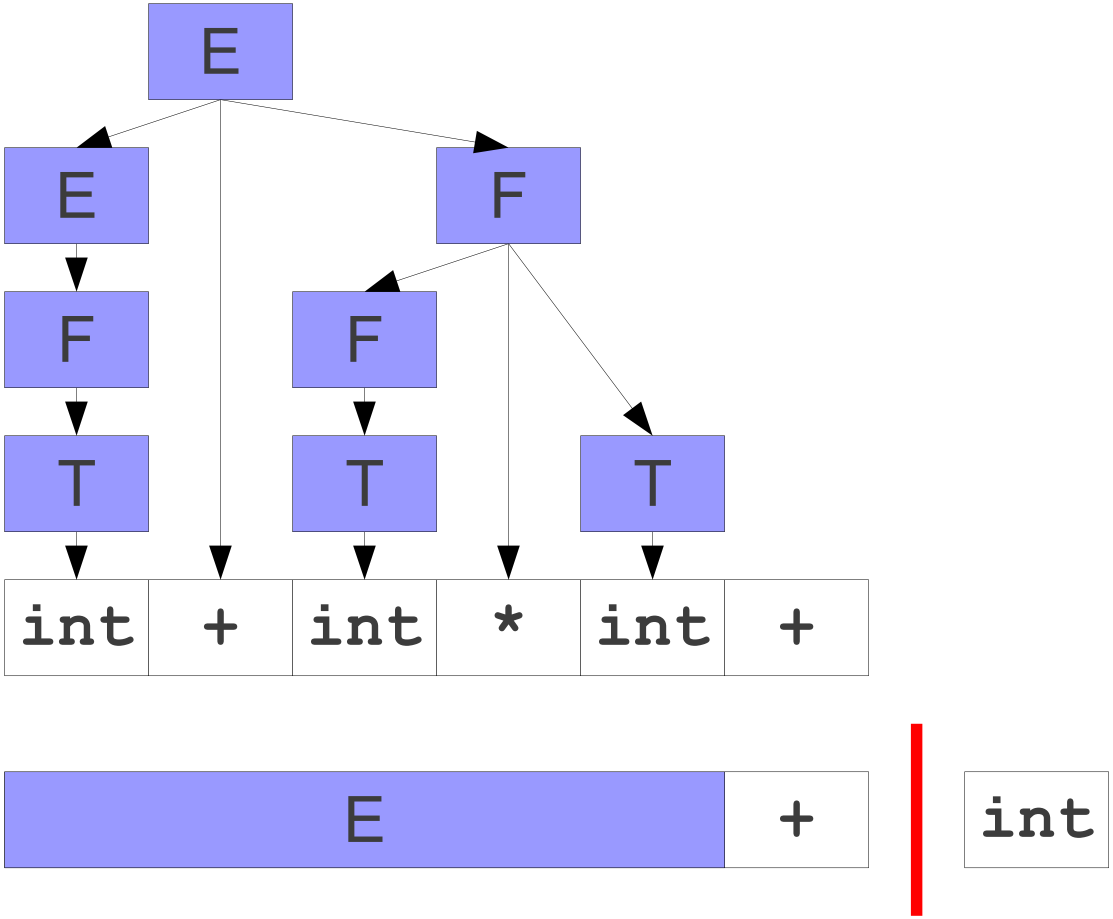
A Sample Shift/Reduce Parse

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



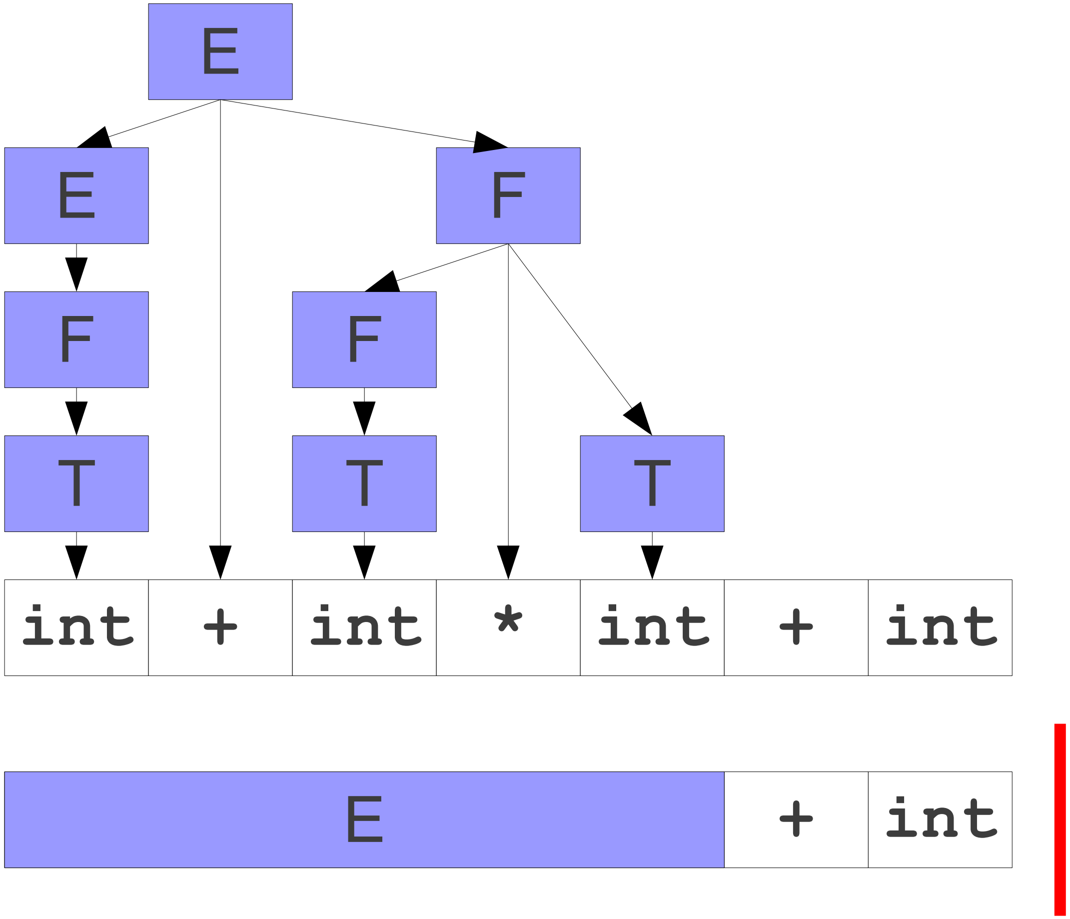
A Sample Shift/Reduce Parse

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



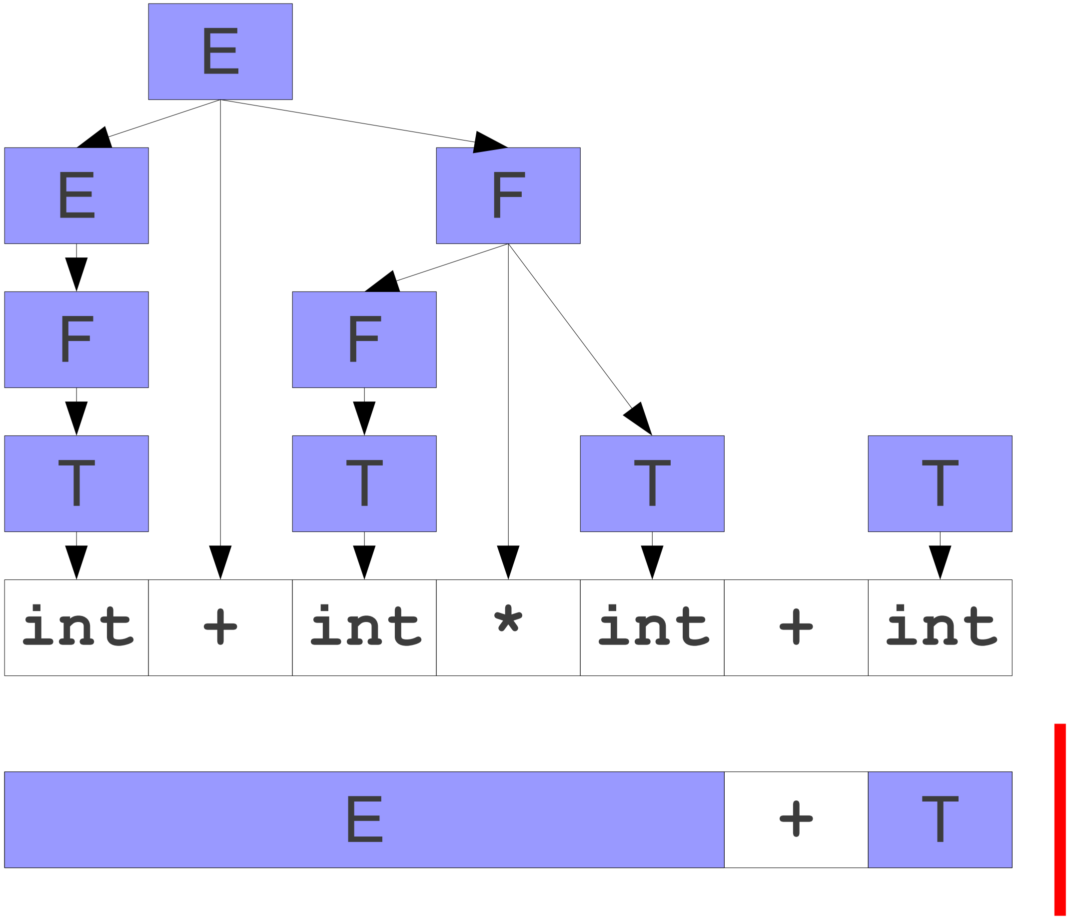
A Sample Shift/Reduce Parse

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



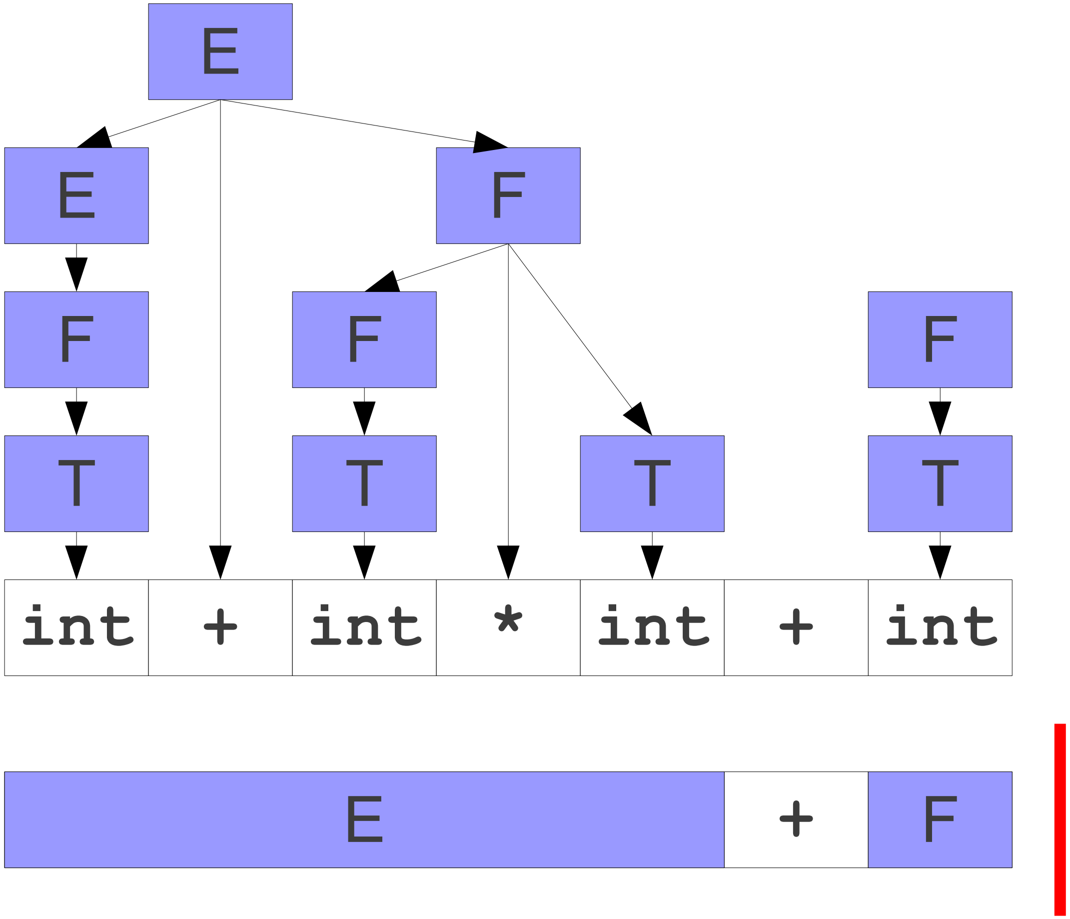
A Sample Shift/Reduce Parse

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



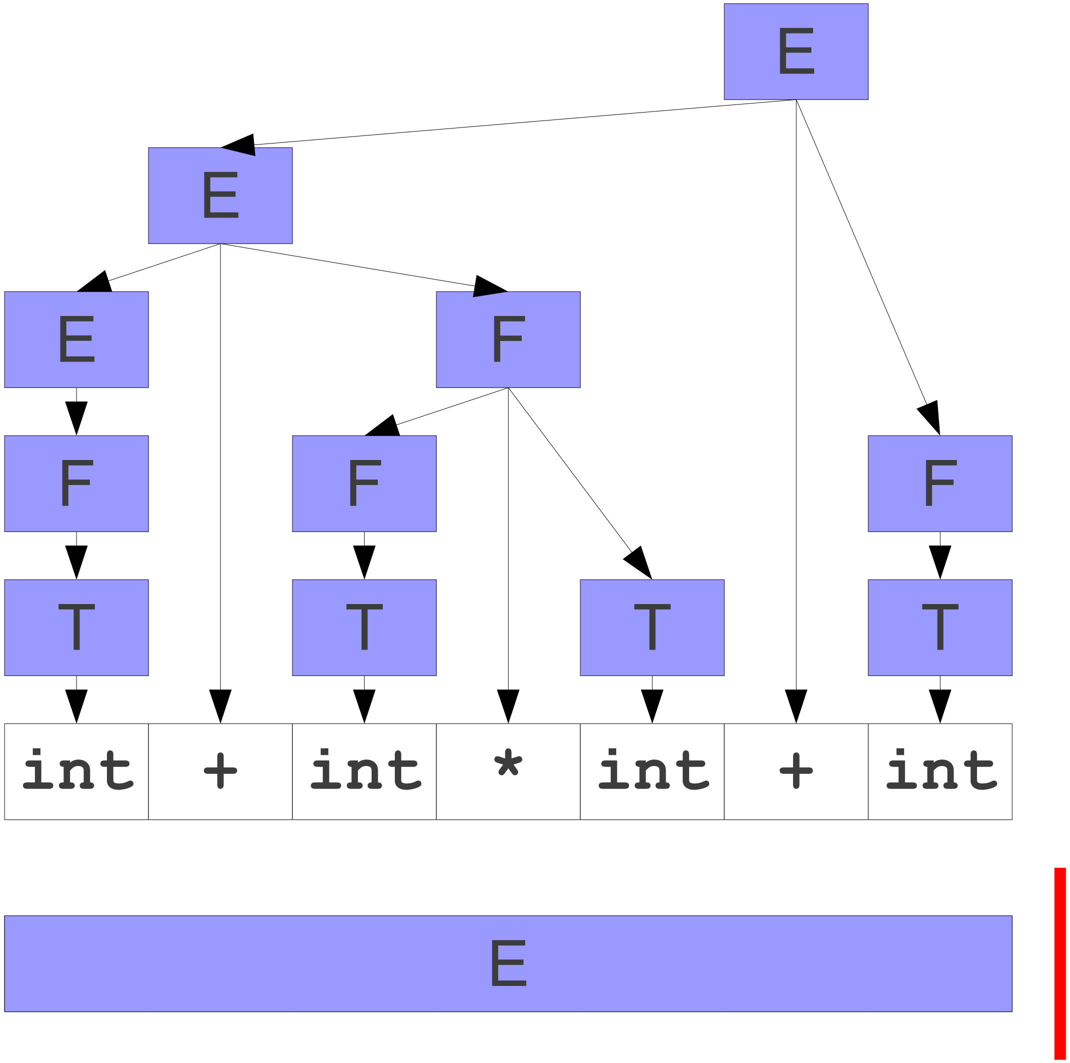
A Sample Shift/Reduce Parse

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**)



An Important Observation

- All of the reductions we applied were to the far right end of the left area.
- This is not a coincidence; all reductions are always applied all the way to the end of the left area.
- Inductive proof sketch:
    - After no reduces, the first reduction can be done at the right end of the left area.
    - After at least one reduce, the very right of the left area is a nonterminal. This nonterminal must be part of the next reduction, since we're tracing a rightmost derivation backwards.

An Important Corollary

- Since reductions are always at the right side of the left area, we never need to shift from the left to the right.
- No need to “uncover” something to do a reduction.
- Consequently, 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.

Simplifying our Terminology

- All activity in a shift/reduce parser is at the far right end of the left area.
- Idea: Represent the left area as a stack.
- Shift: Push the next terminal onto the stack.
- Reduce: Pop some number of symbols from the stack, then push the appropriate nonterminal.

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?

Searching for Handles

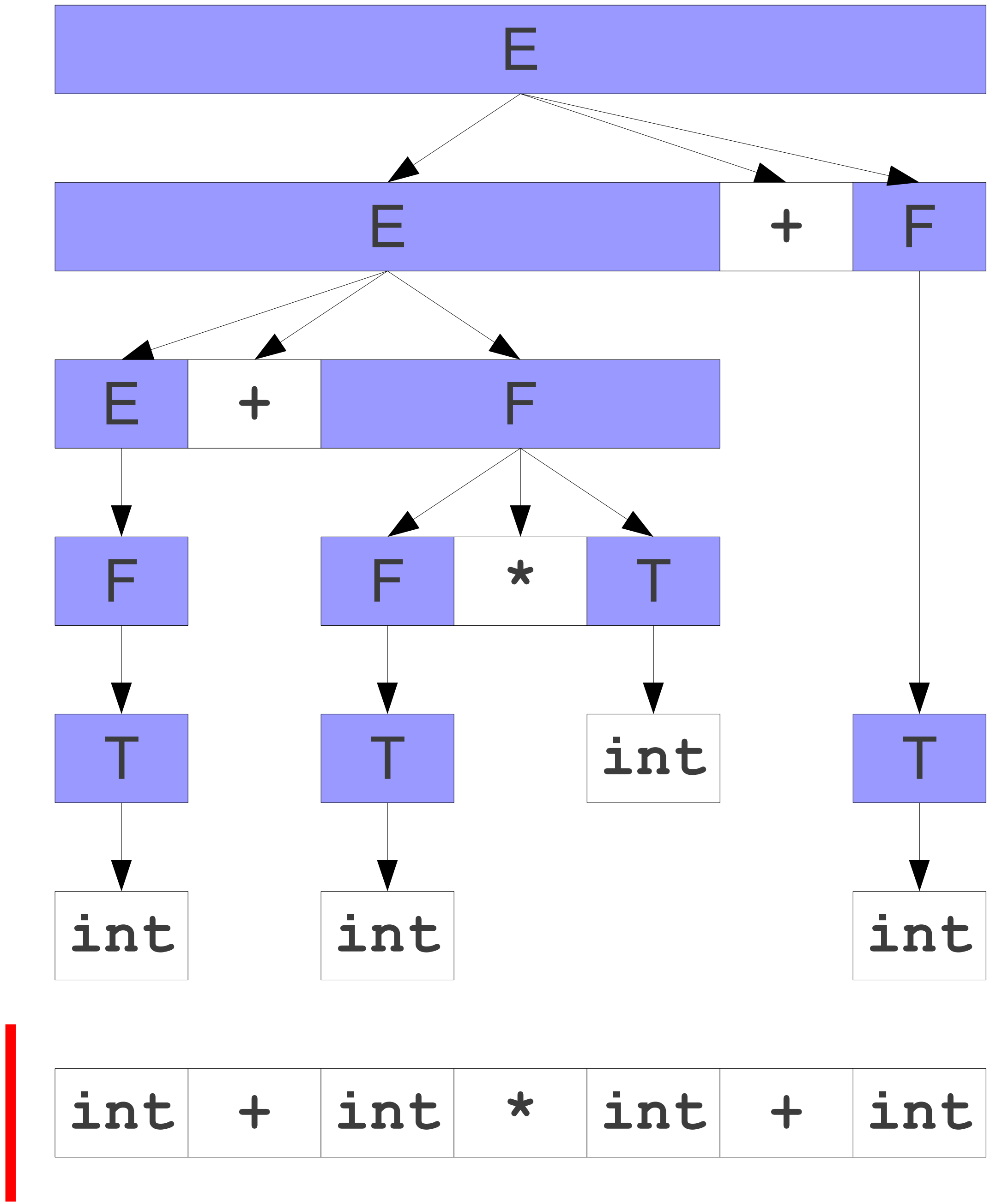
- When using a shift/reduce parser, we must decide whether to shift or reduce at each point.
- We only want to reduce when we know we have a handle.
- **Question:** How can we tell that we might be looking at a handle?

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.

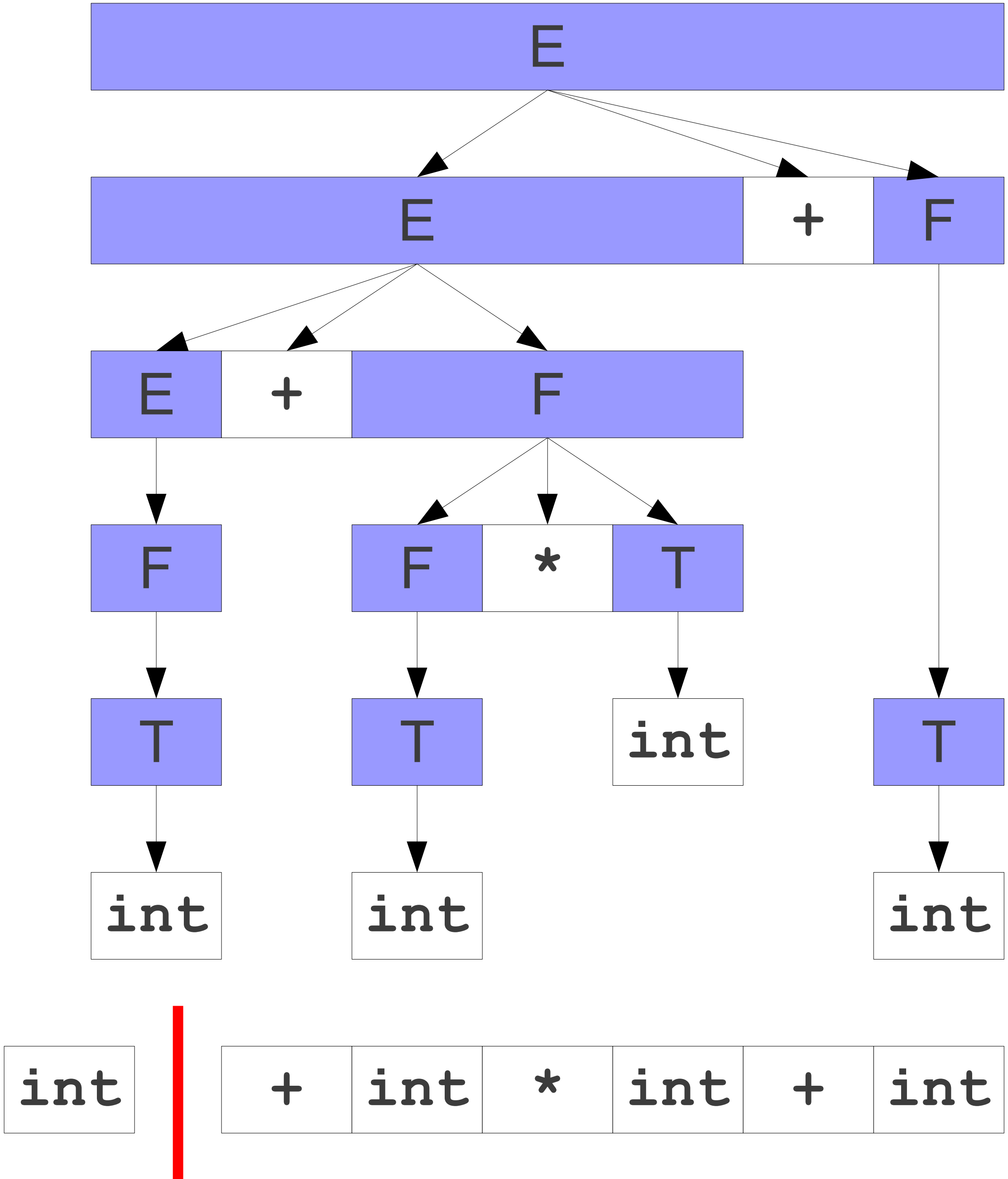
Another Look at Handles

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



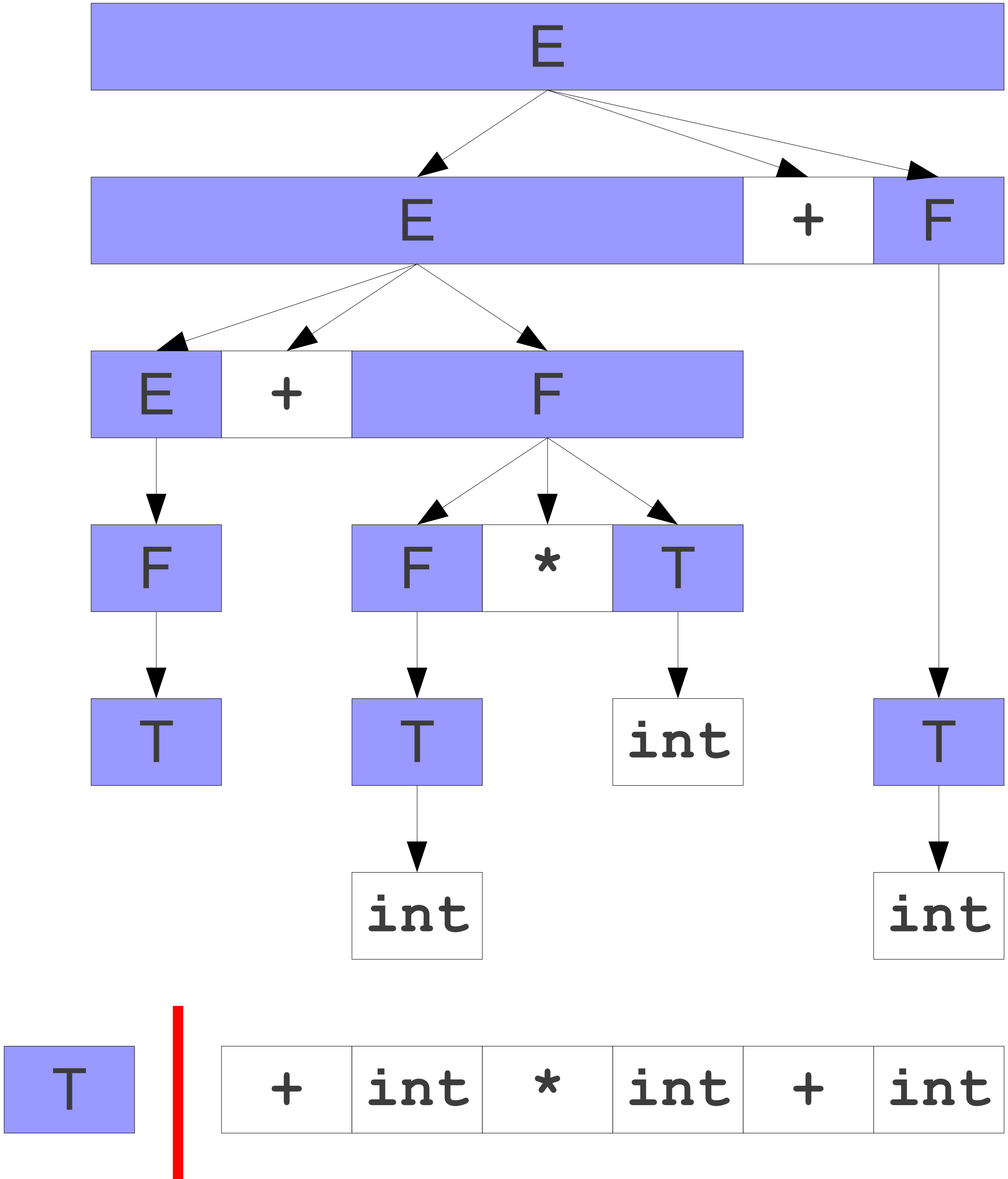
Another Look at Handles

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



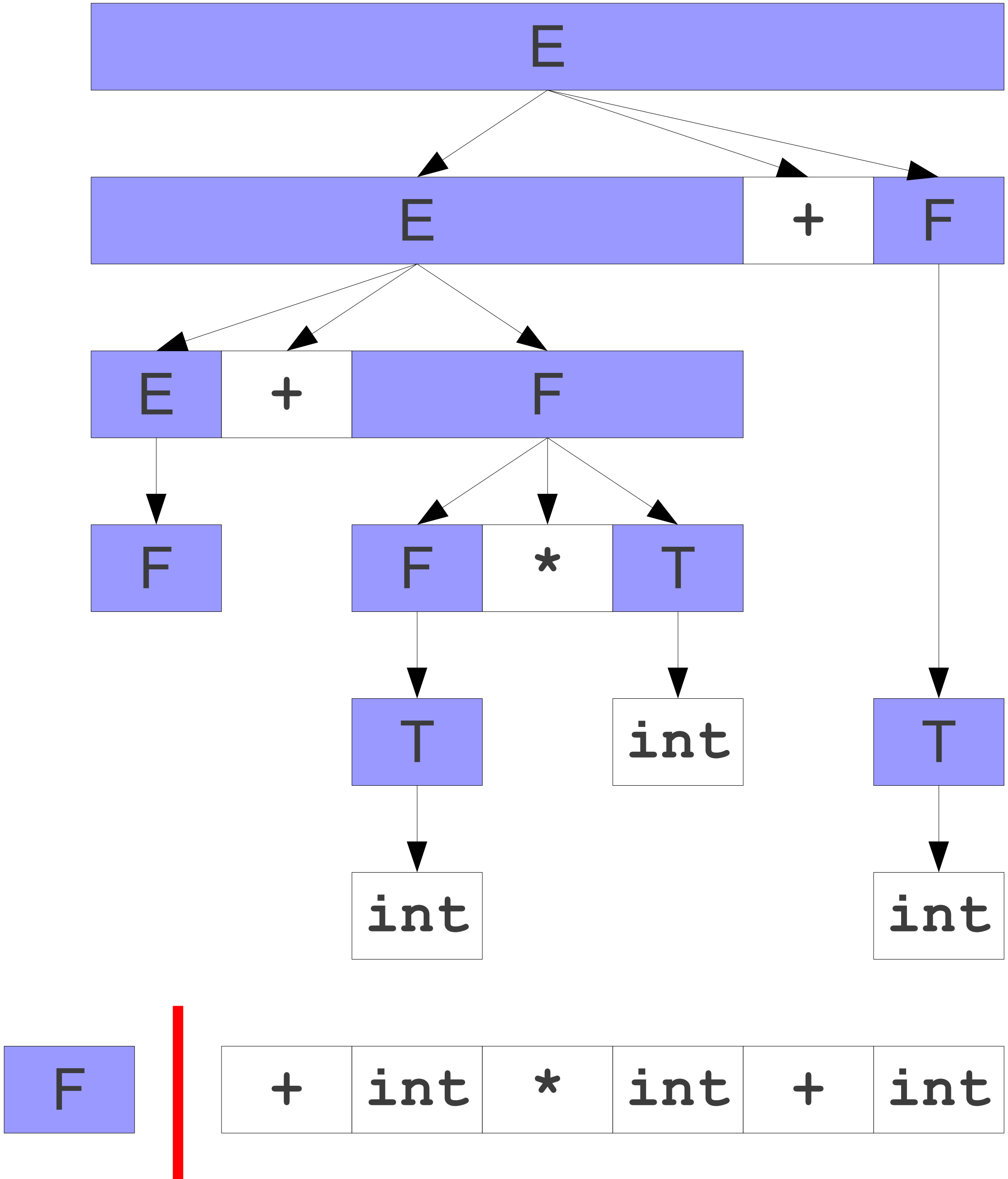
Another Look at Handles

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



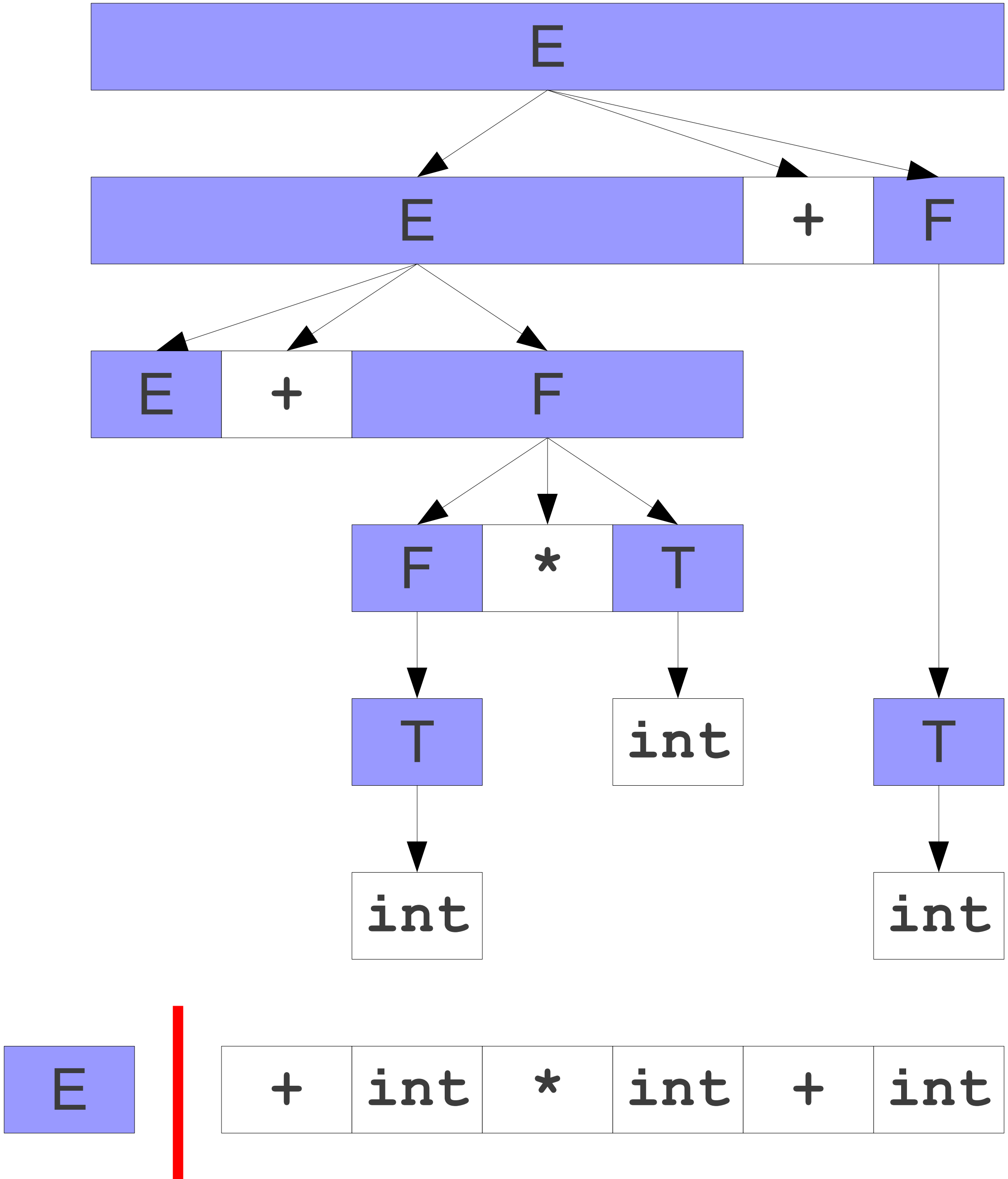
Another Look at Handles

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



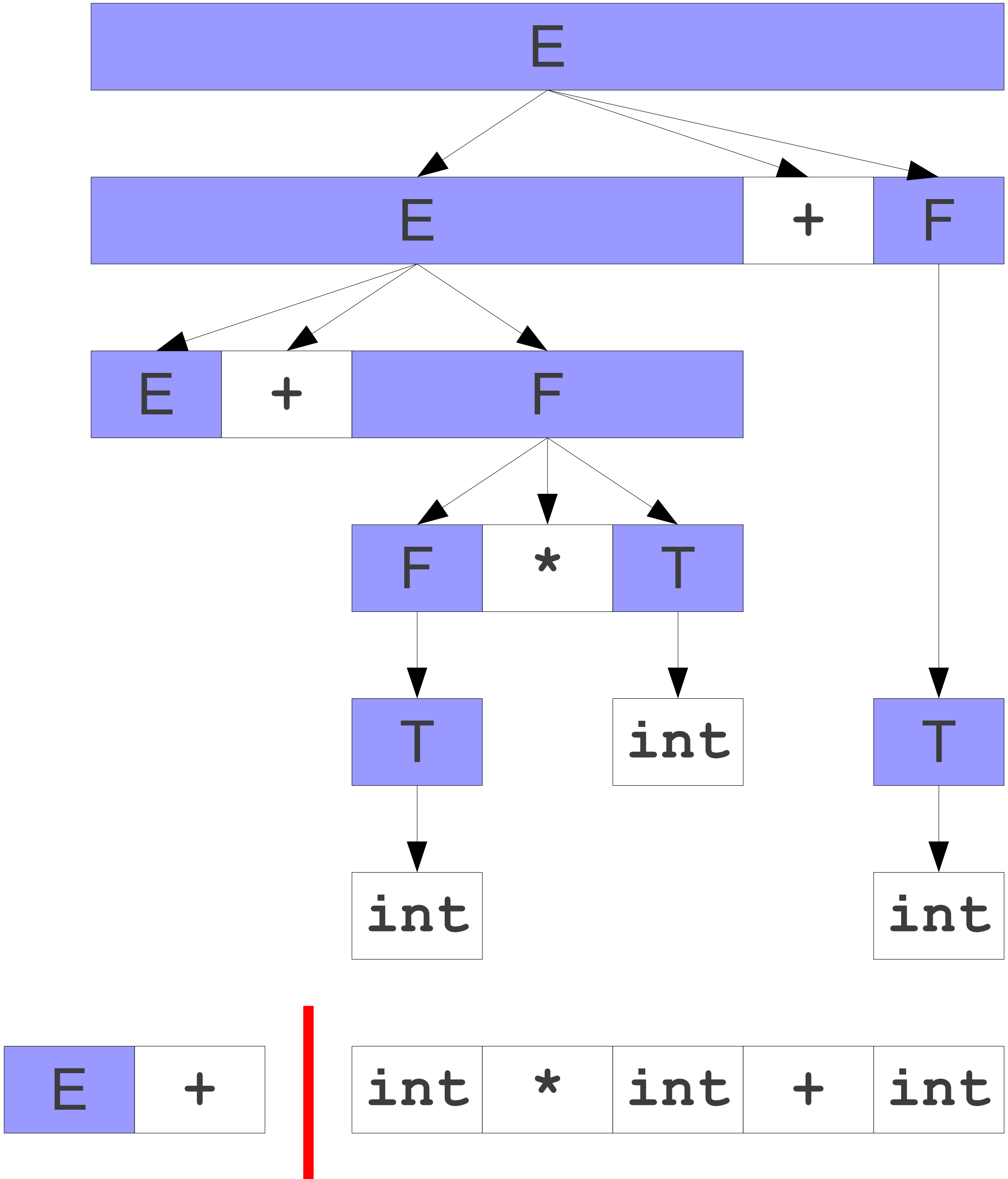
Another Look at Handles

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



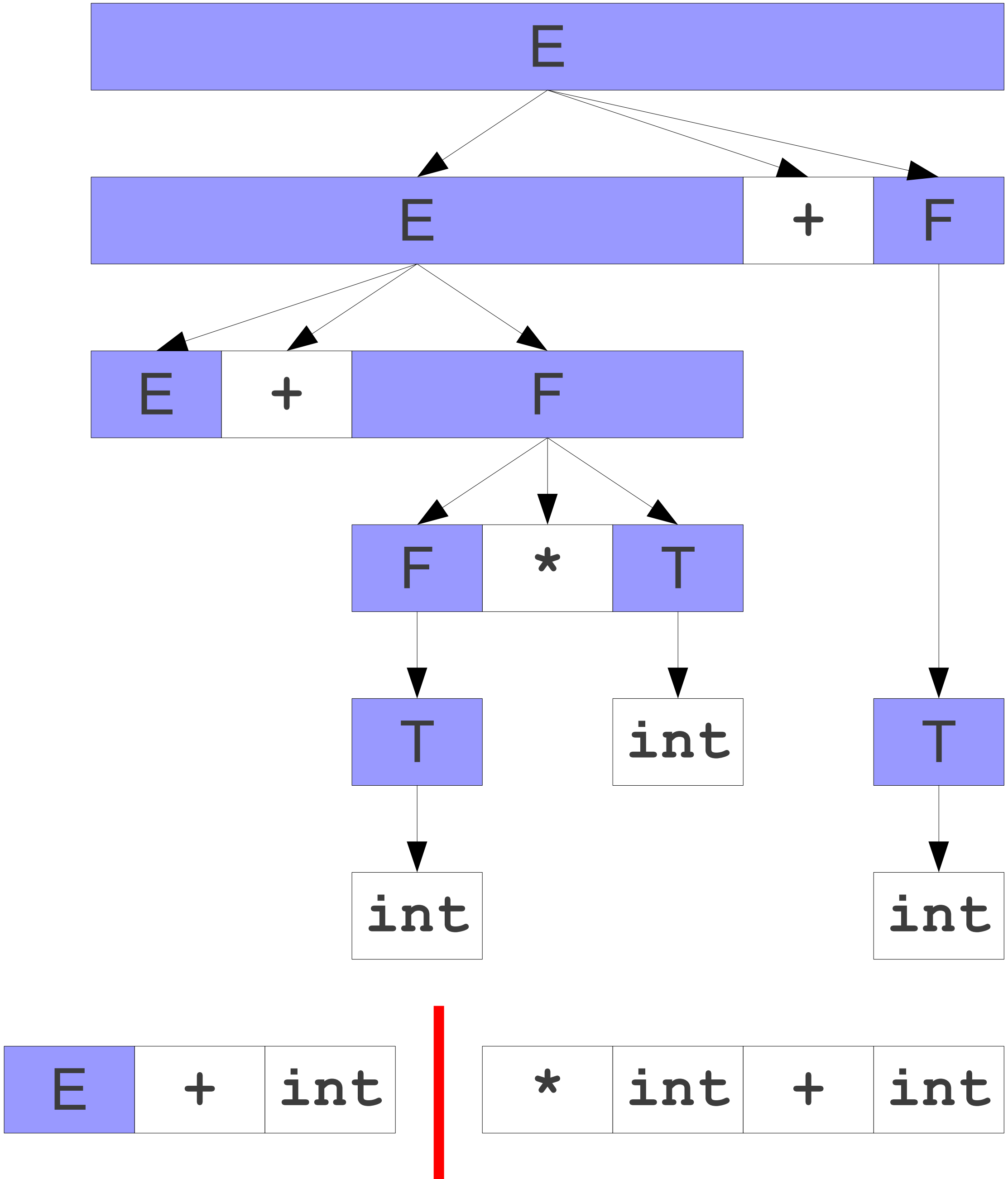
Another Look at Handles

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



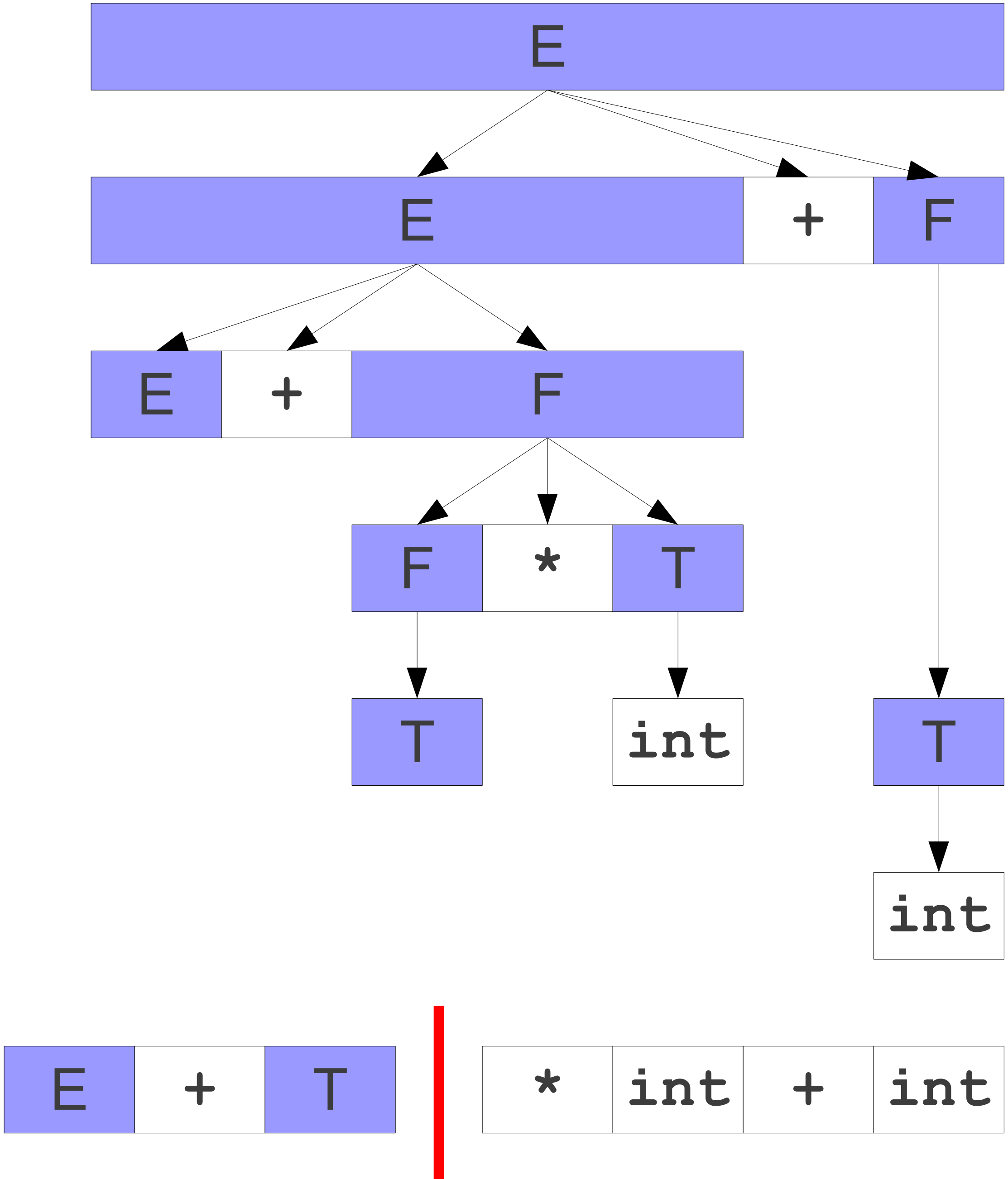
Another Look at Handles

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



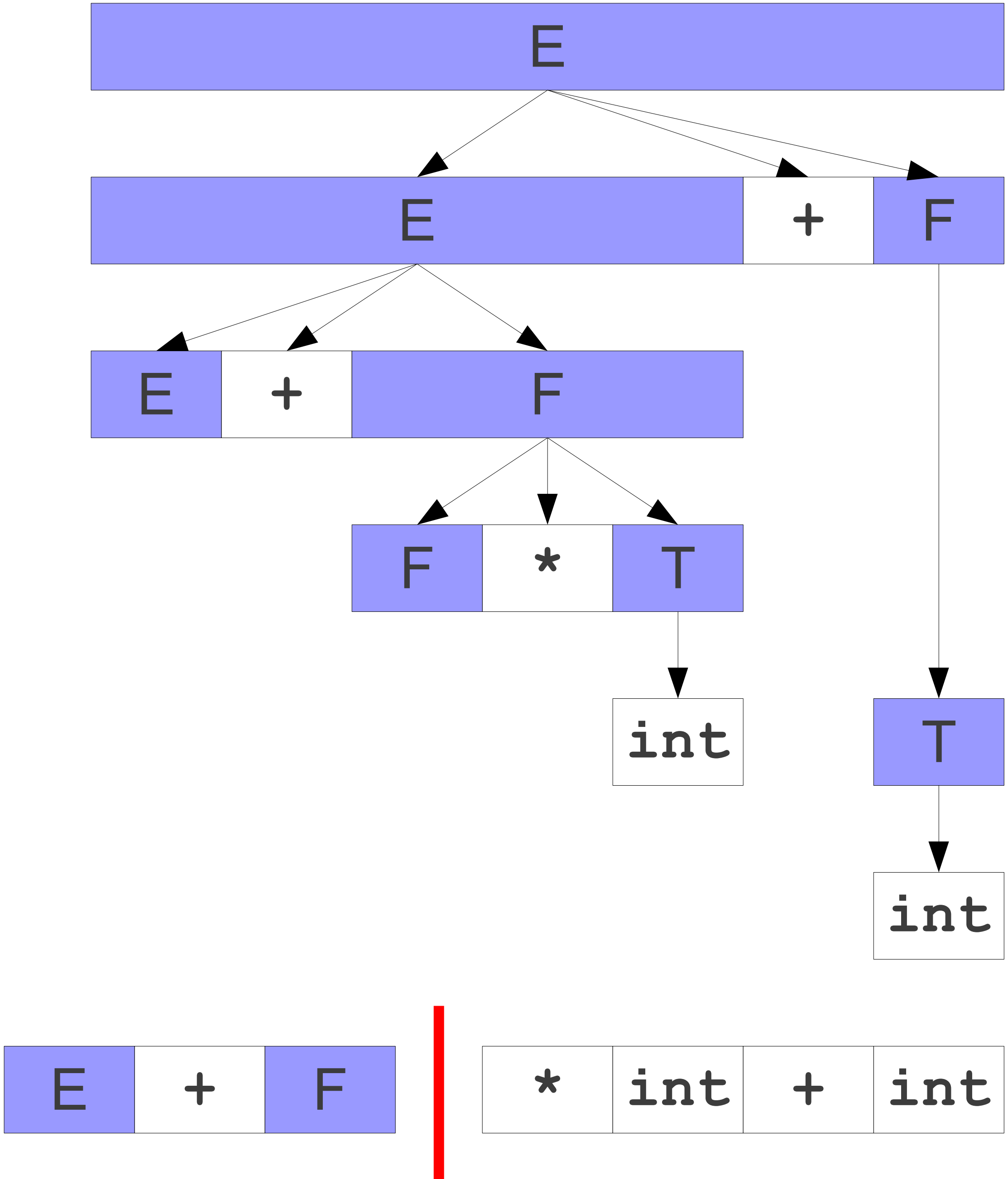
Another Look at Handles

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



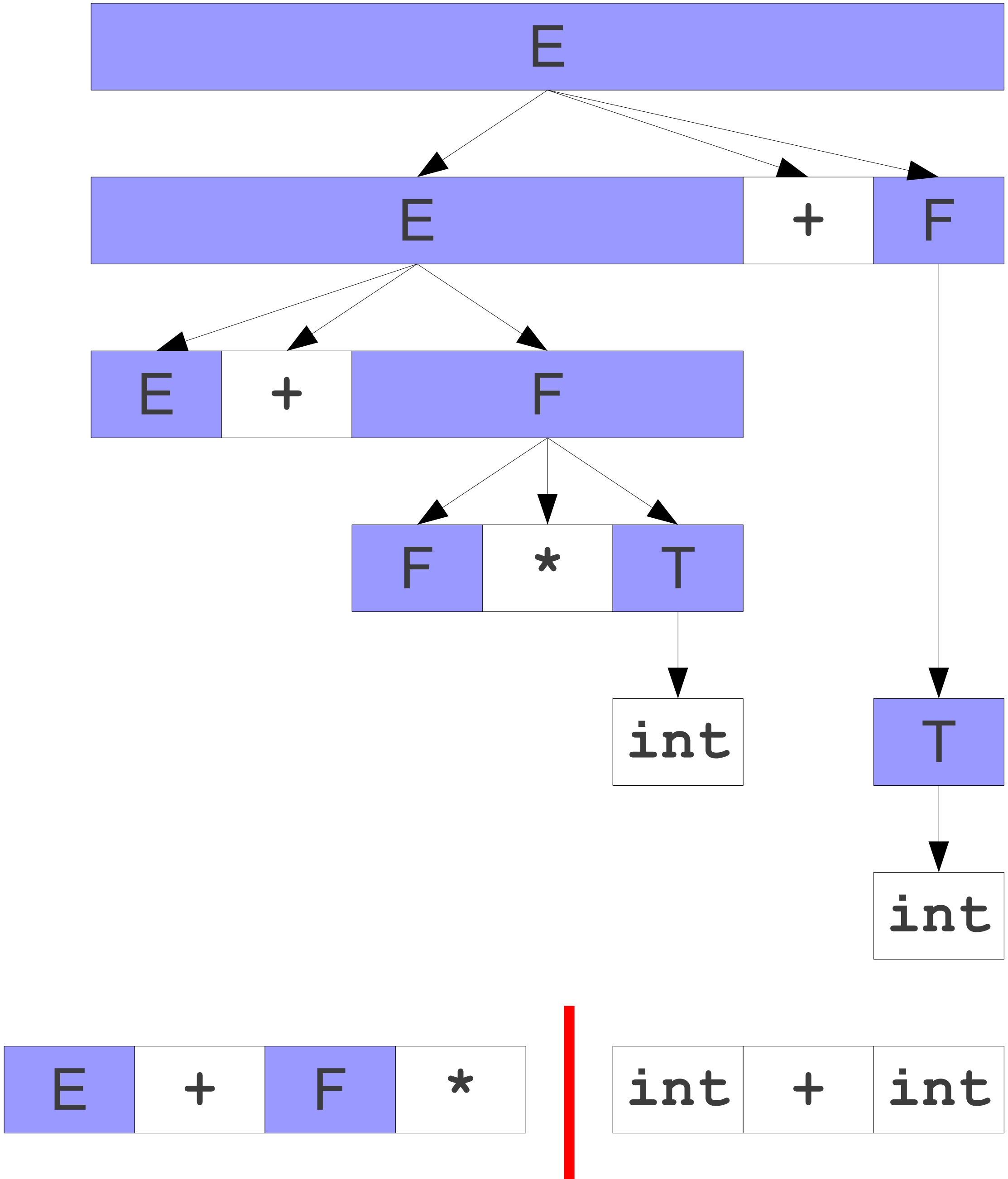
Another Look at Handles

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



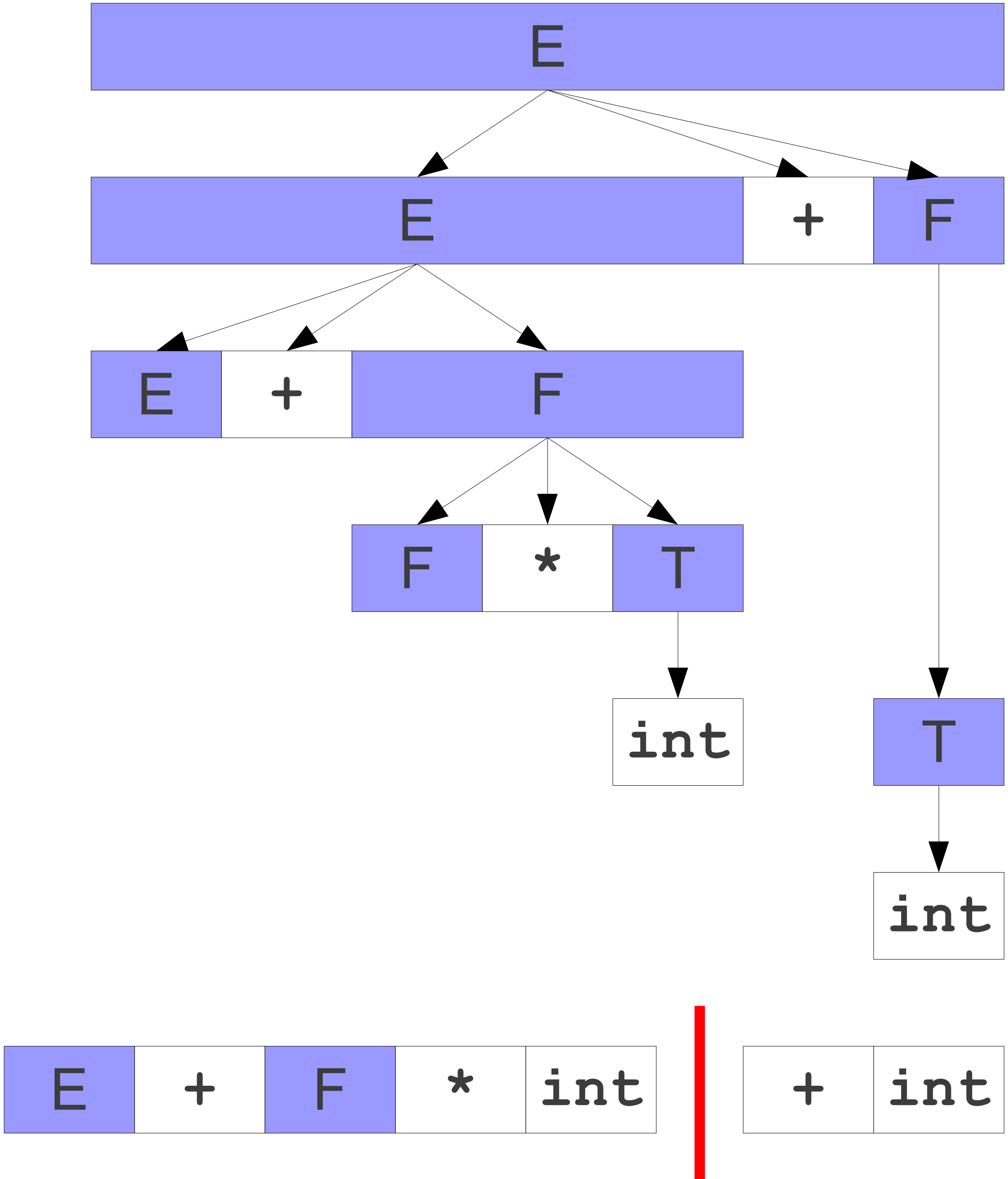
Another Look at Handles

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



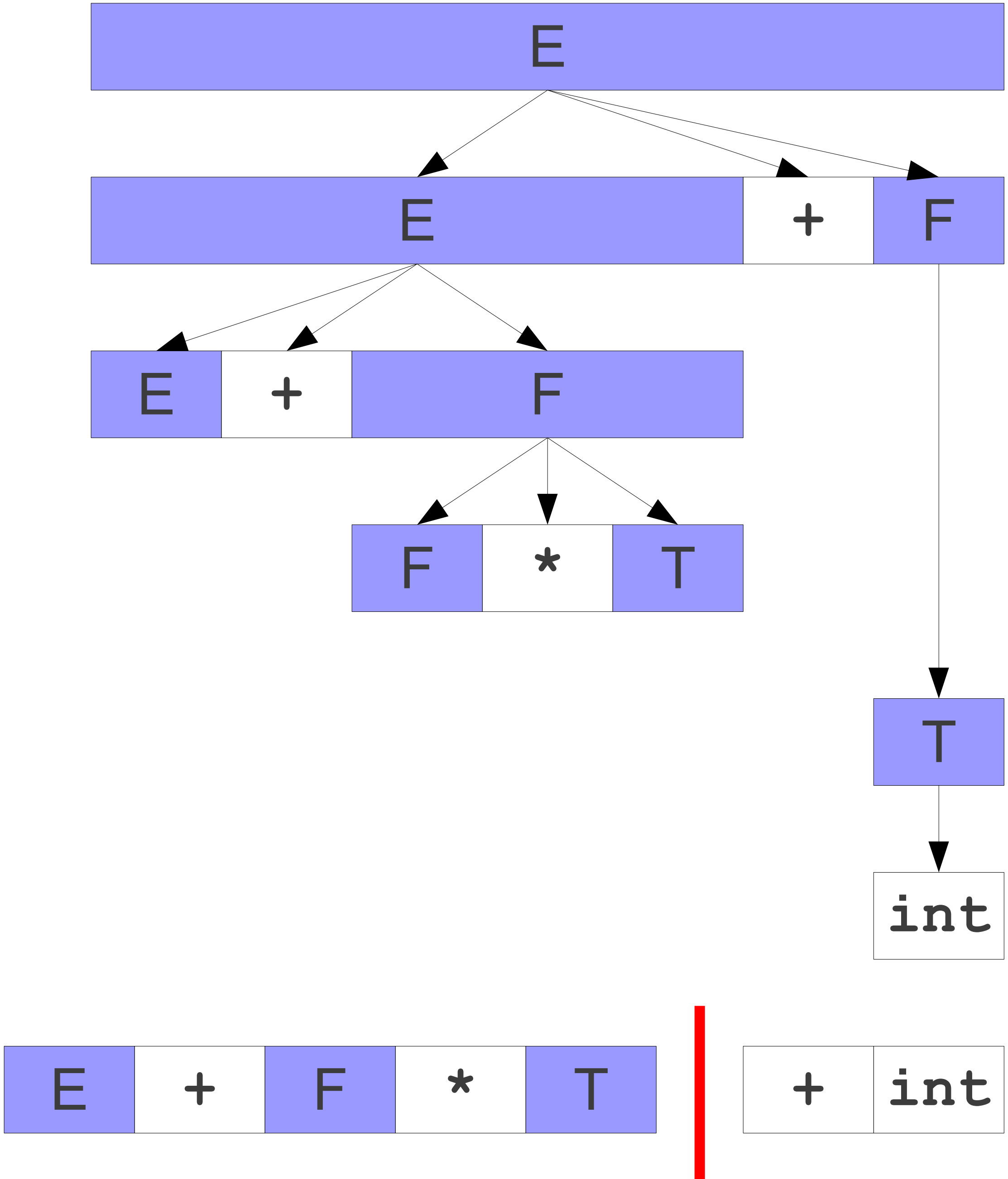
Another Look at Handles

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



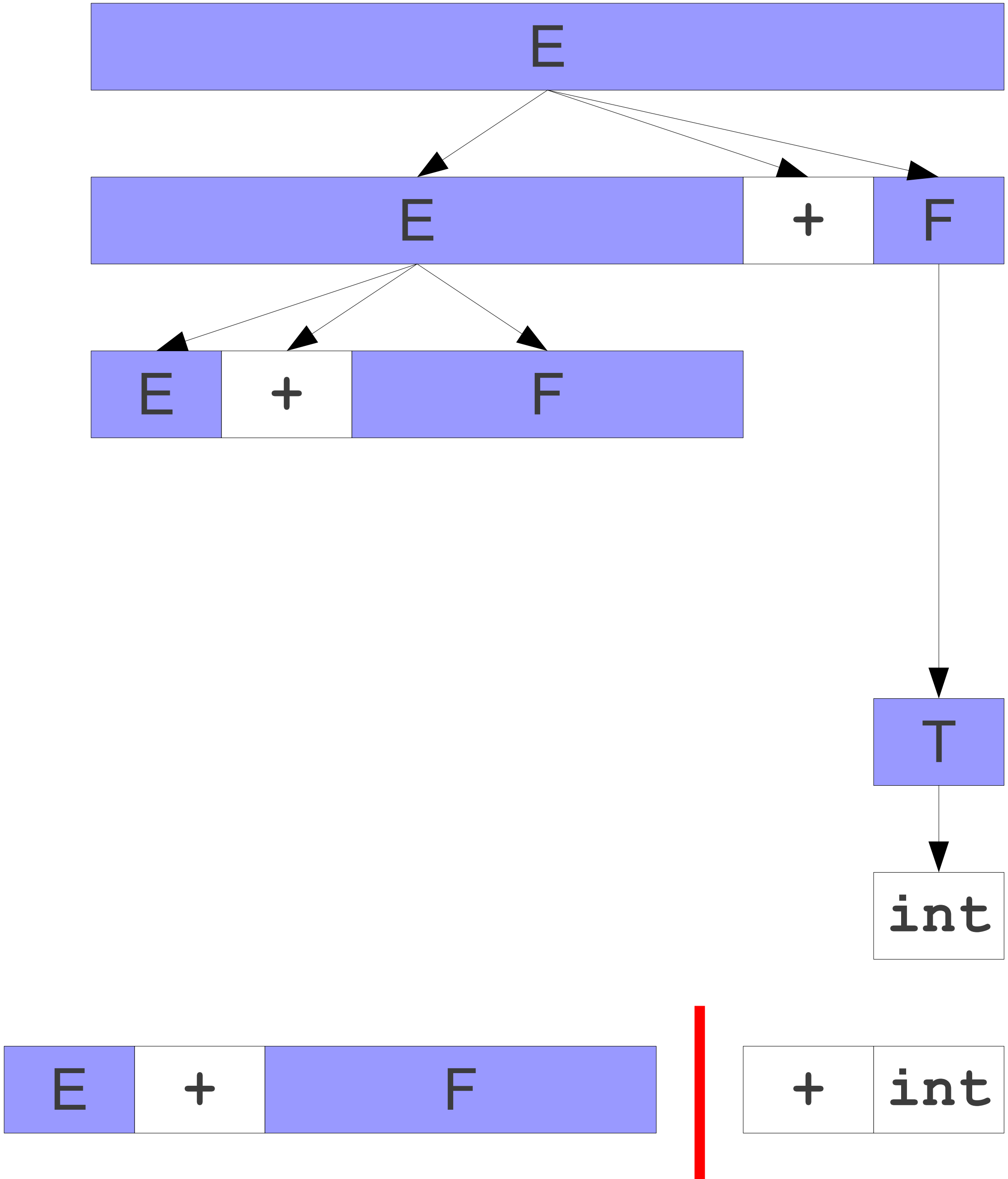
Another Look at Handles

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



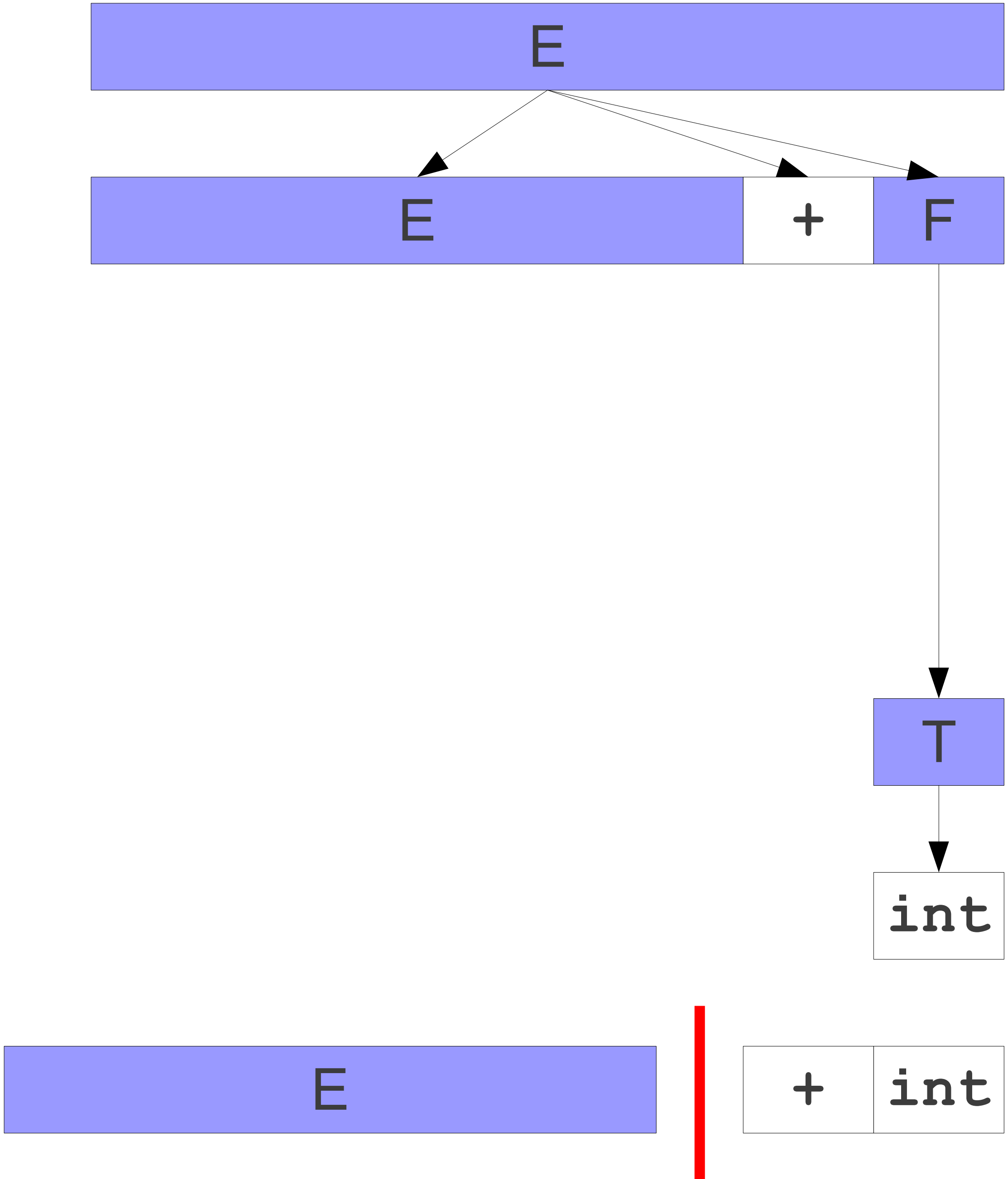
Another Look at Handles

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



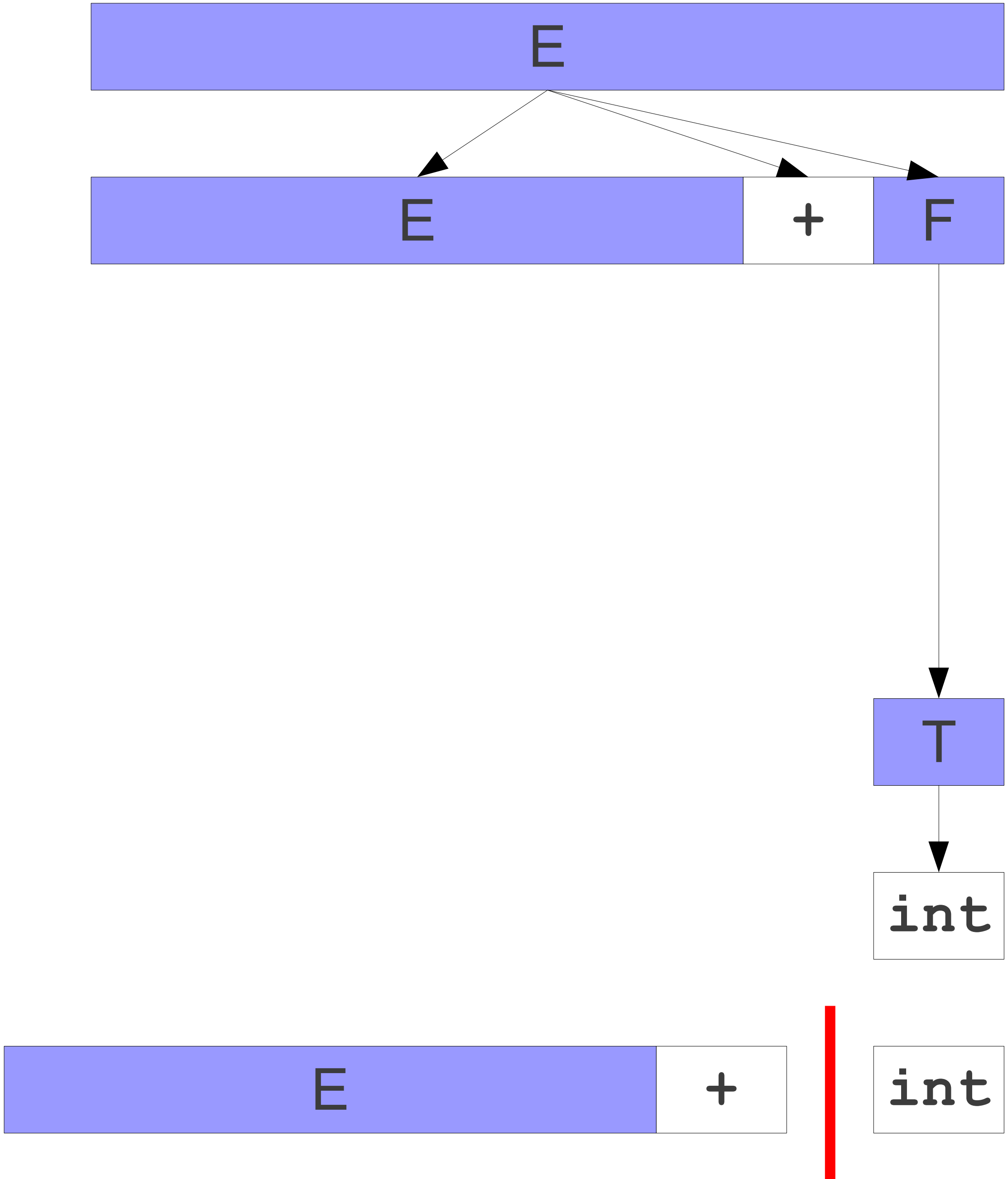
Another Look at Handles

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



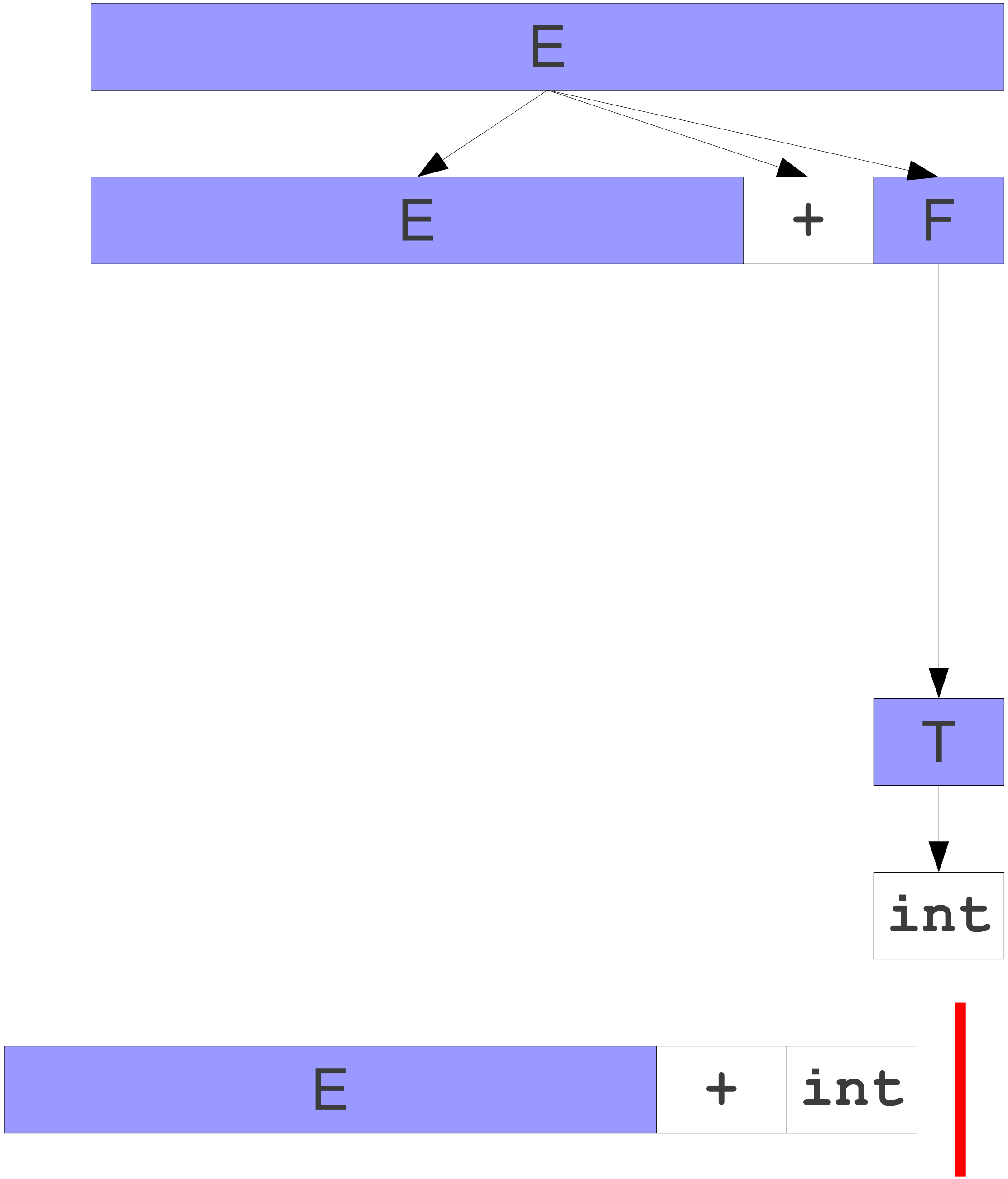
Another Look at Handles

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

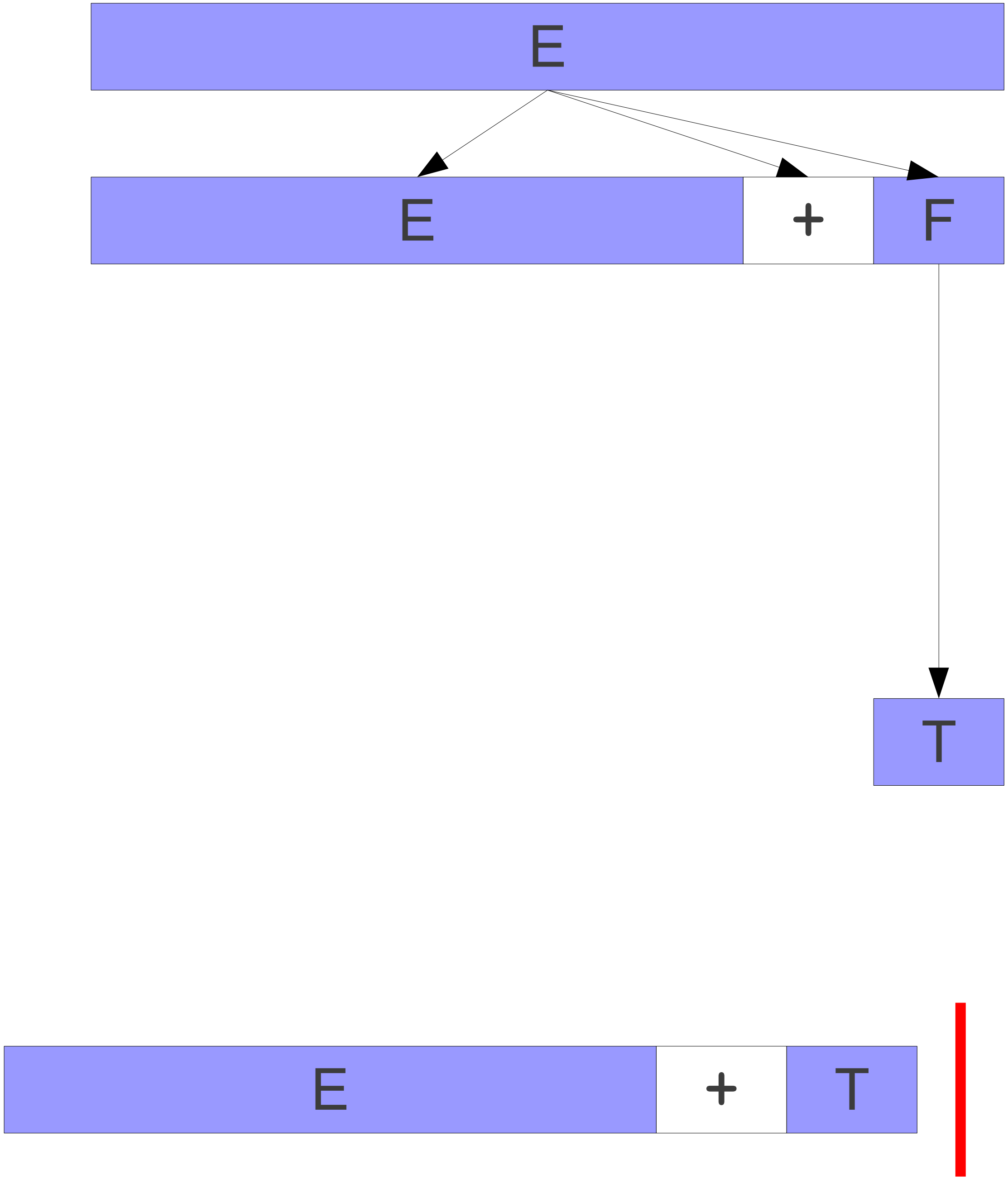


Another Look at Handles

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

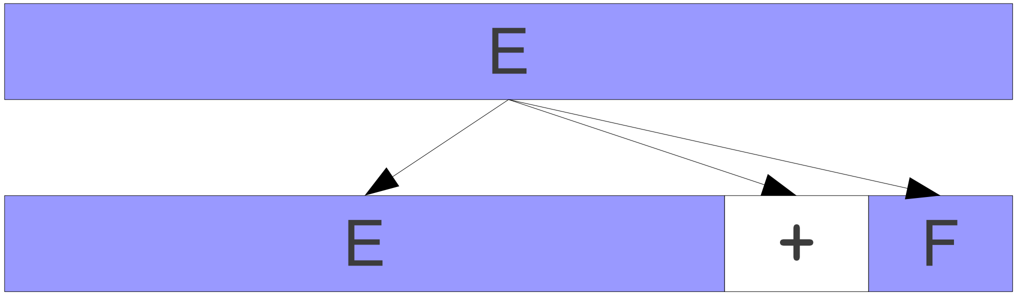


Another Look at Handles

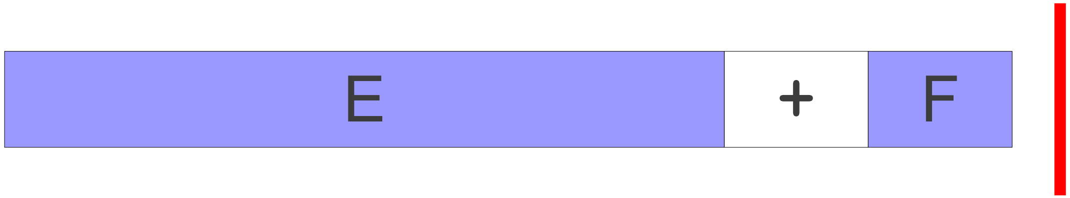


E → **F**
E → **E** + **F**
F → **F** * **T**
F → **T**
T → `int`
T → `(`**E**`)`

Another Look at Handles

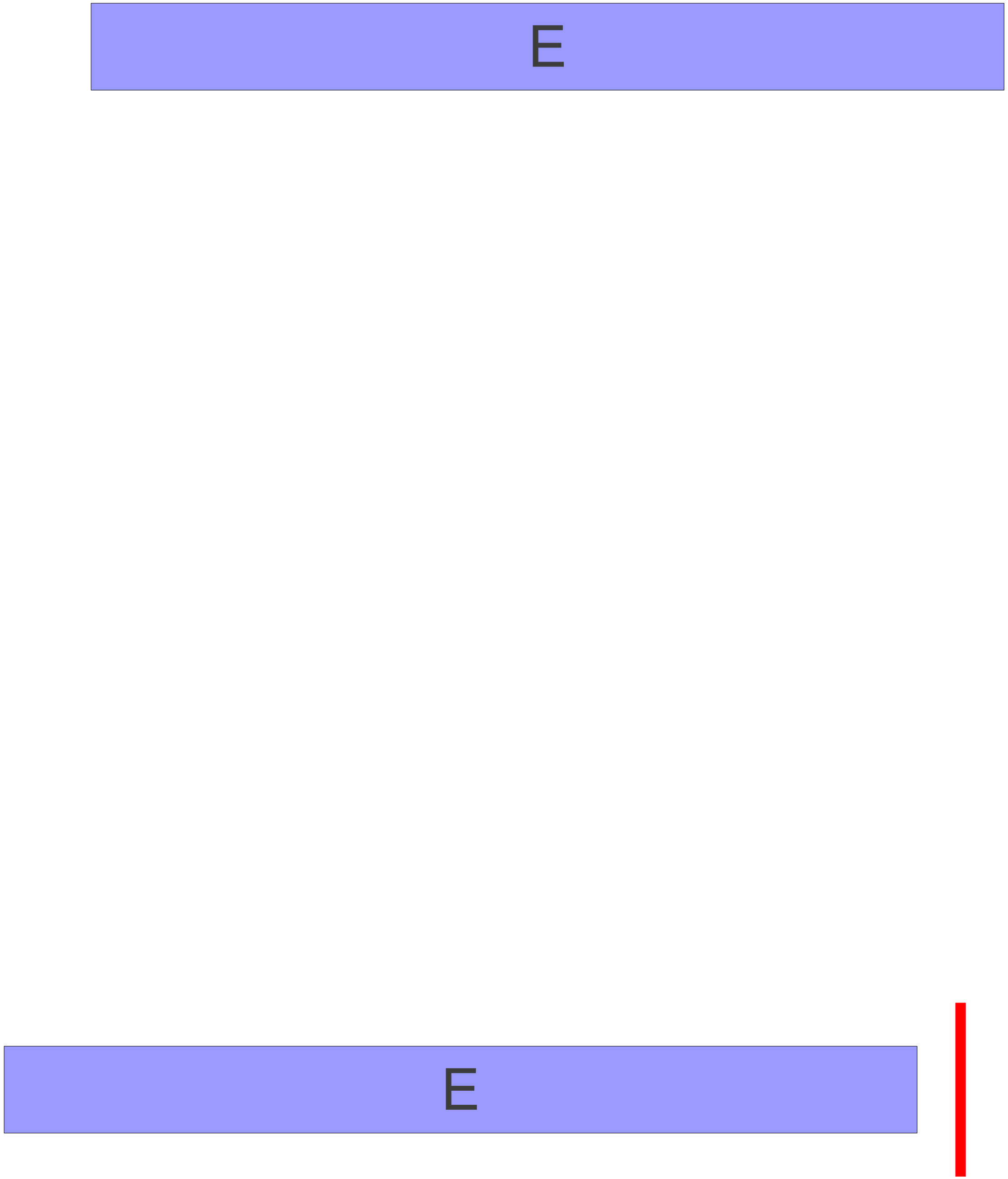


E → **F**
E → **E** **+** **F**
F → **F** ***** **T**
F → **T**
T → **int**
T → **(E)**



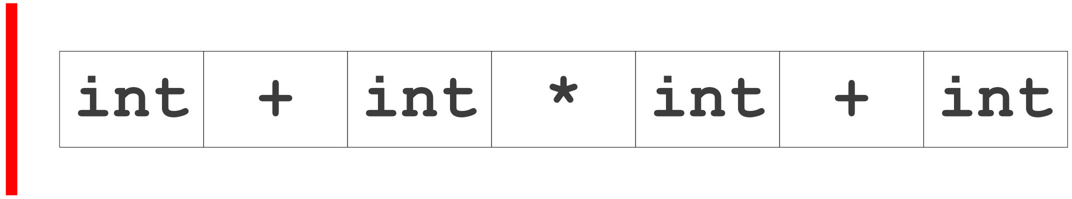
Another Look at Handles

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



Tracking Our Position

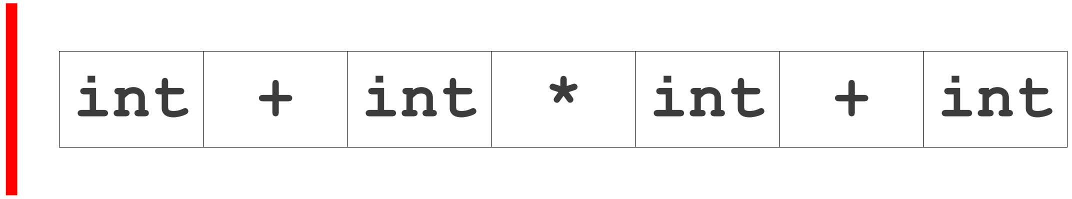
$E \rightarrow F$
$E \rightarrow E$ **+** F
$F \rightarrow F$ ***** T
$F \rightarrow T$
$T \rightarrow$ **int**
$T \rightarrow$ **(**E**)**



int	+	int	*	int	+	int

Tracking Our Position

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

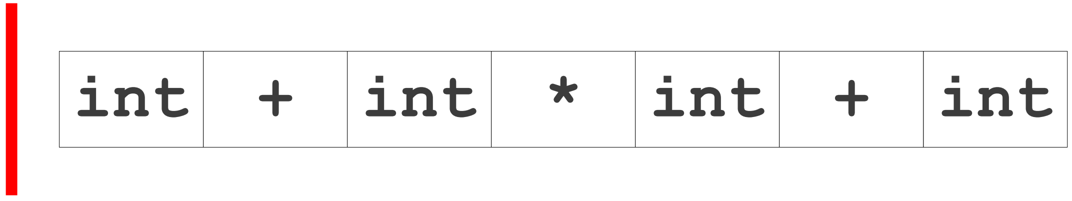


int	+	int	*	int	+	int

Tracking Our Position

S → · E

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



`int`	`+`	`int`	`*`	`int`	`+`	`int`

Tracking Our Position

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

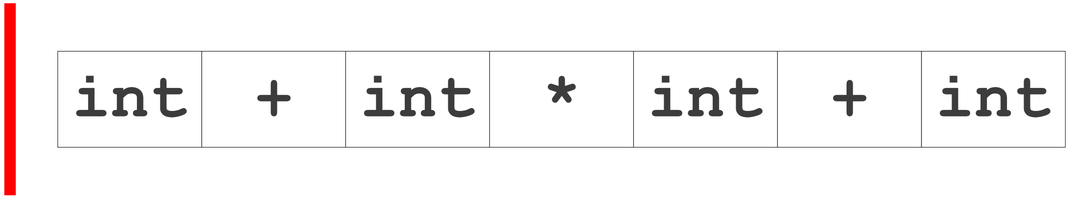
S → · E
E → · E + F

`int`	`+`	`int`	`*`	`int`	`+`	`int`

Tracking Our Position

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

S → · E
E → · E + F
E → · E + F

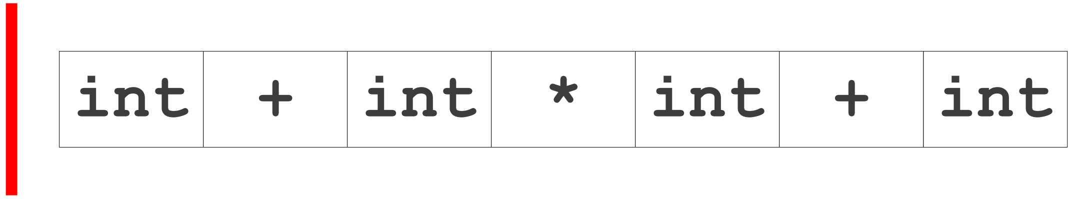


`int`	`+`	`int`	`*`	`int`	`+`	`int`

Tracking Our Position

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	*	int	+	int

Tracking Our Position

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

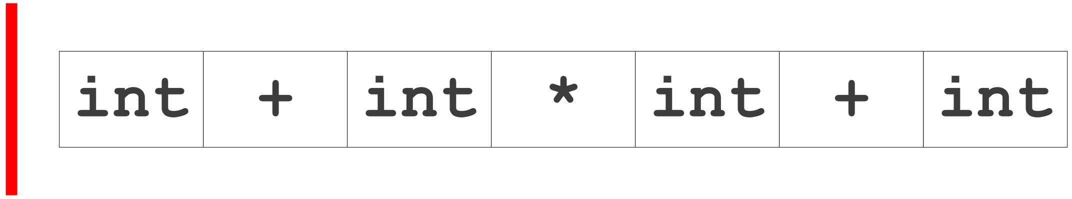
S → · E
E → · E + F
E → · E + F
E → · F
F → · T

`int`	`+`	`int`	`*`	`int`	`+`	`int`

Tracking Our Position

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
F → · T
T → · `int`

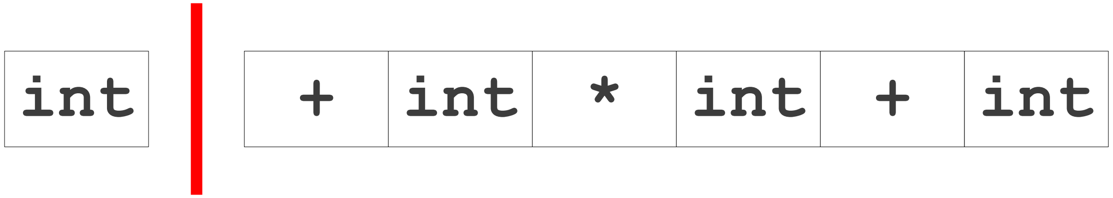


`int`	`+`	`int`	`*`	`int`	`+`	`int`

Tracking Our Position

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
F → · T
T → `int` ·



Tracking Our Position

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

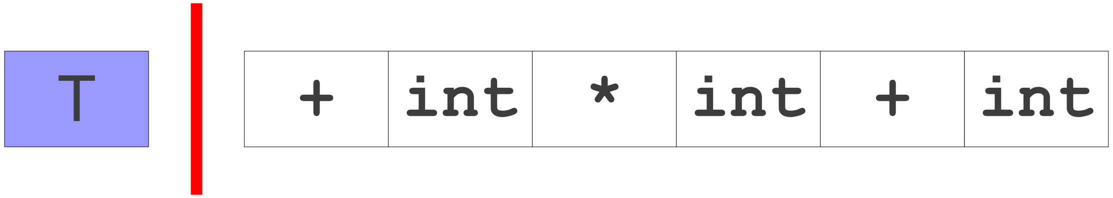
$S \rightarrow \cdot E$
$E \rightarrow \cdot E + F$
$E \rightarrow \cdot E + F$
$E \rightarrow \cdot F$
$F \rightarrow \cdot T$

T	|	+	int	*	int	+	int

Tracking Our Position

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
F → T ·



T		+	int	*	int	+	int

Tracking Our Position

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

F	|	+	int	*	int	+	int

Tracking Our Position

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 ·

F	|	+	int	*	int	+	int

Tracking Our Position

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

$S \to \cdot E$
$E \to \cdot E + F$
$E \to \cdot E + F$

E	|	+	int	*	int	+	int

Tracking Our Position

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

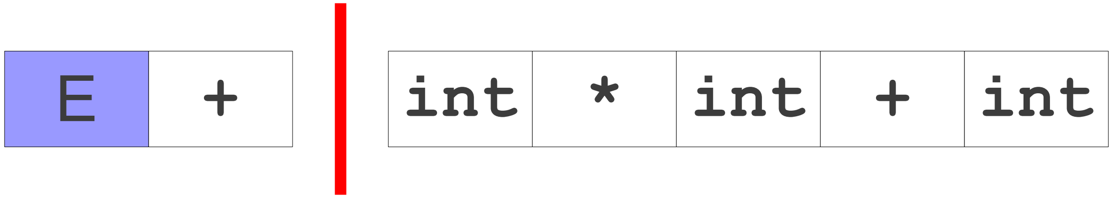
$S \to \cdot E$
$E \to \cdot E + F$
$E \to E \cdot + F$

E	`+`	`int`	`*`	`int`	`+`	`int`

Tracking Our Position

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

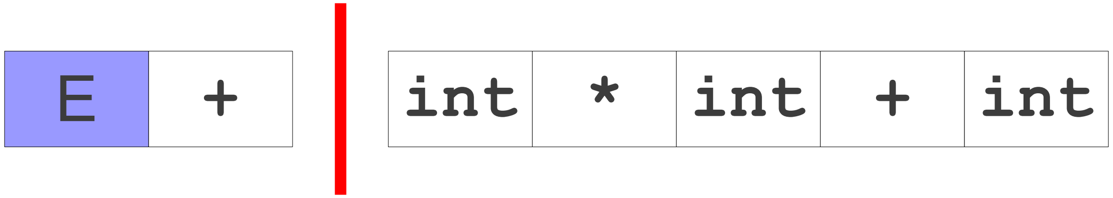
S → · E
E → · E + F
E → E + · F



Tracking Our Position

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

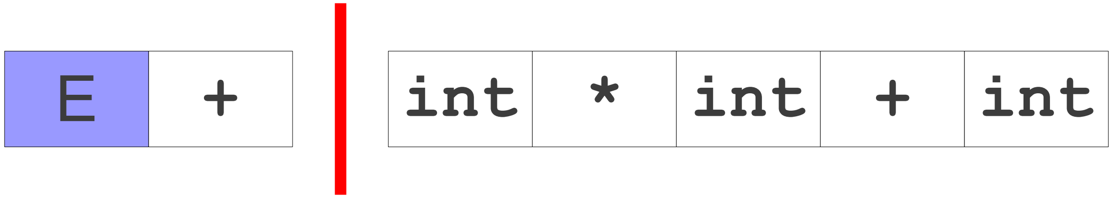
S → · E
E → · E + F
E → E + · F
F → · F * T



Tracking Our Position

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

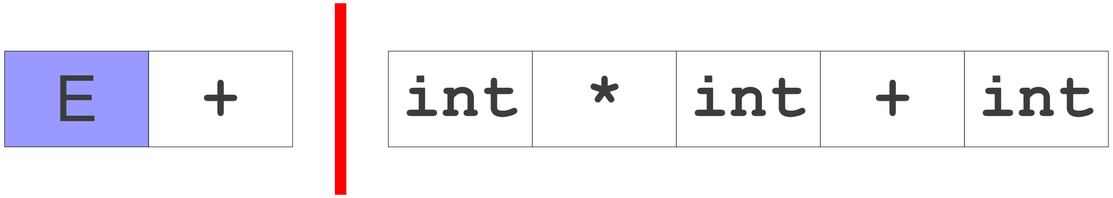
S → · E
E → · E + F
E → E + · F
F → · F * T
F → · T



Tracking Our Position

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

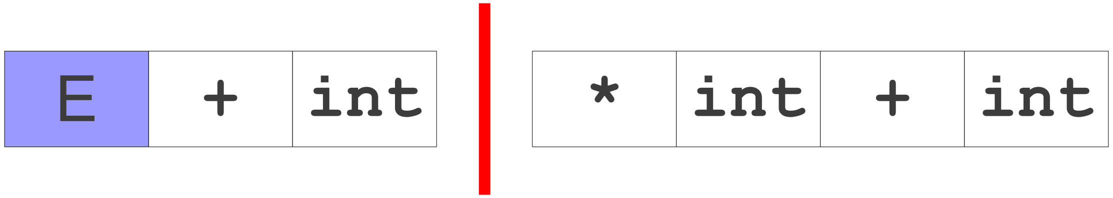
S → · E
E → · E + F
E → E + · F
F → · F `*` T
F → · T
T → · `int`



Tracking Our Position

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

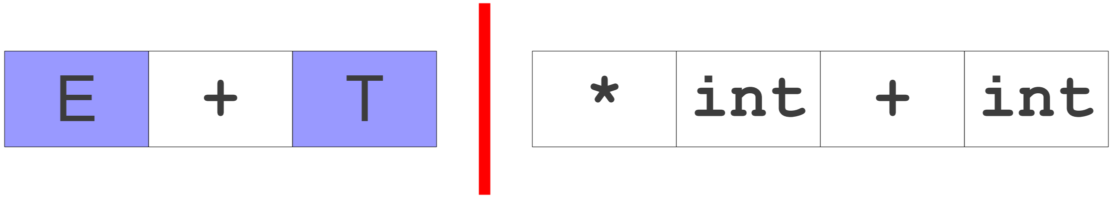
S → · E
E → · E + F
E → E + · F
F → · F * T
F → · T
T → `int` ·



Tracking Our Position

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

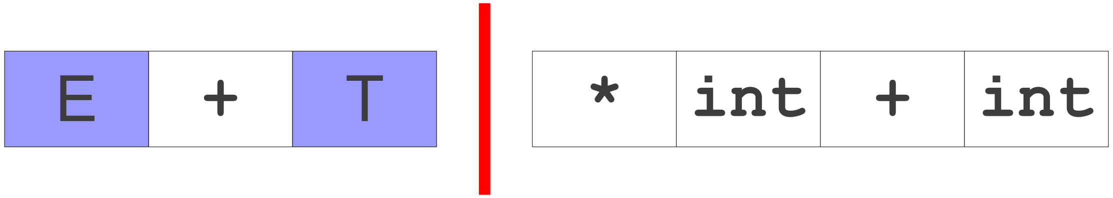
S → · E
E → · E + F
E → E + · F
F → · F * T
F → · T



Tracking Our Position

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

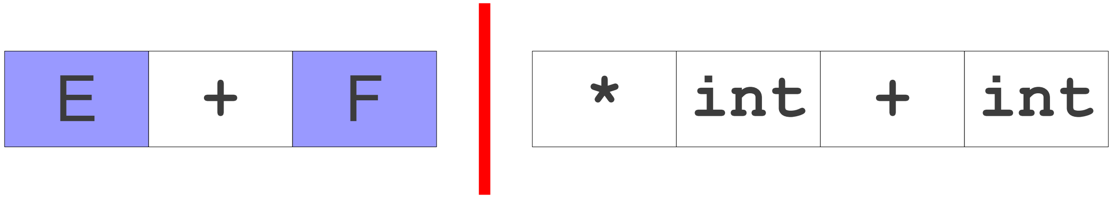
$S \rightarrow \cdot E$
$E \rightarrow \cdot E + F$
$E \rightarrow E + \cdot F$
$F \rightarrow \cdot F * T$
$F \rightarrow T \cdot$



Tracking Our Position

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

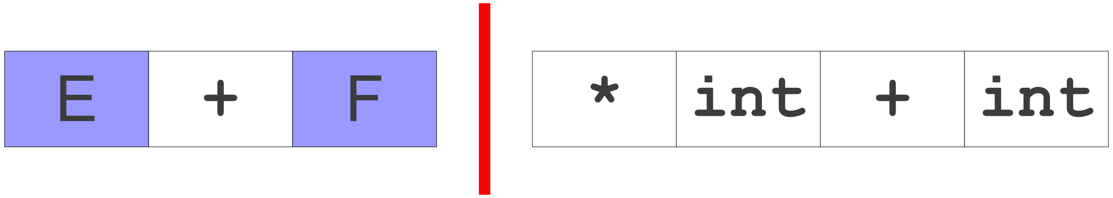
S → · E
E → · E + F
E → E + · F
F → · F * T



Tracking Our Position

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

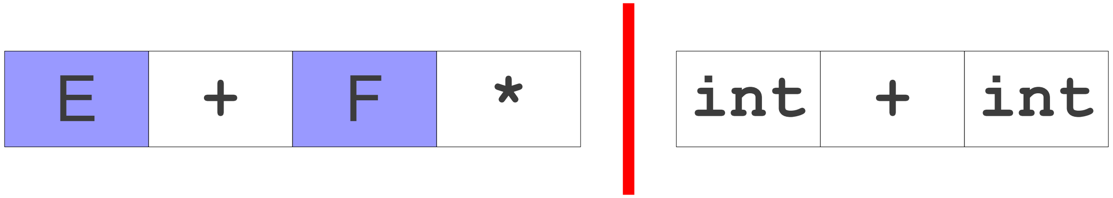
S → · E
E → · E + F
E → E + · F
F → F · * T



Tracking Our Position

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

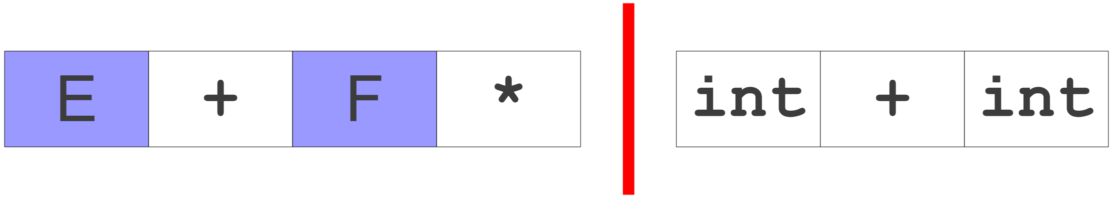
S → · E
E → · E + F
E → E + · F
F → F * · T



Tracking Our Position

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`

Tracking Our Position

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

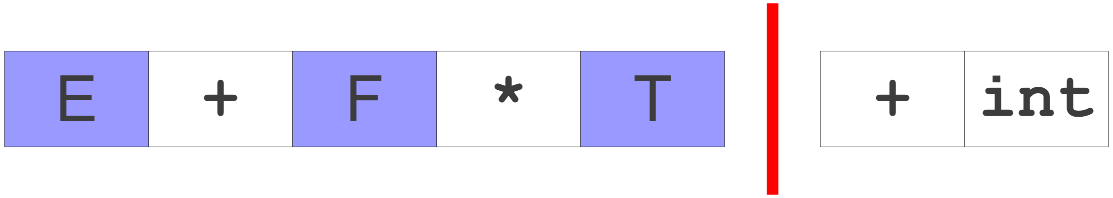
S → · E
E → · E + F
E → E + · F
F → F * · T
T → `int` ·

E	+	F	*	`int`	+	`int`

Tracking Our Position

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



Tracking Our Position

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

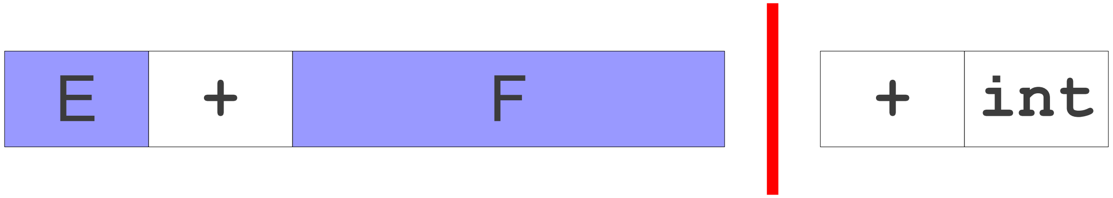
S → · E
E → · E + F
E → E + · F
F → F * T ·

E	+	F	*	T	|	+	int

Tracking Our Position

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

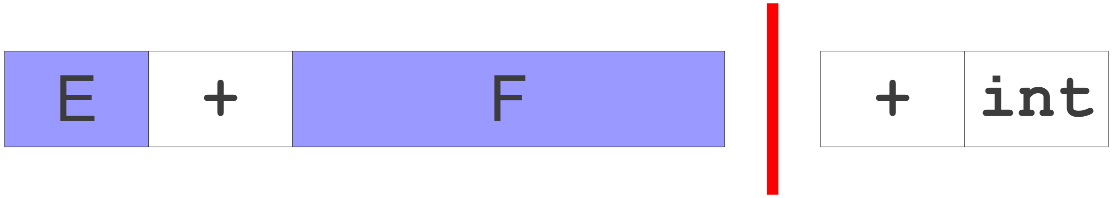
S → · E
E → · E + F
E → E + · F



Tracking Our Position

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

Tracking Our Position

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

$S \rightarrow \cdot E$

$E \rightarrow \cdot E + F$

Tracking Our Position

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

S → · E
E → E · + F


E

\+ `int`

Tracking Our Position

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

S → · E

E → E + · F

Tracking Our Position

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

S → · E
E → E + · F
F → · T

Tracking Our Position

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

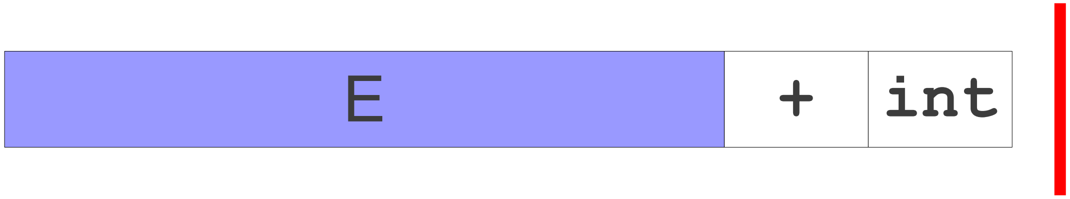
S → · E
E → E + · F
F → · T
T → · int


E
+
int

Tracking Our Position

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

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



Tracking Our Position

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

S → · E
E → E + · F
F → · T

Tracking Our Position

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

S → · E
E → E + · F
F → T ·

Tracking Our Position

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

S → · E

E → E + · F

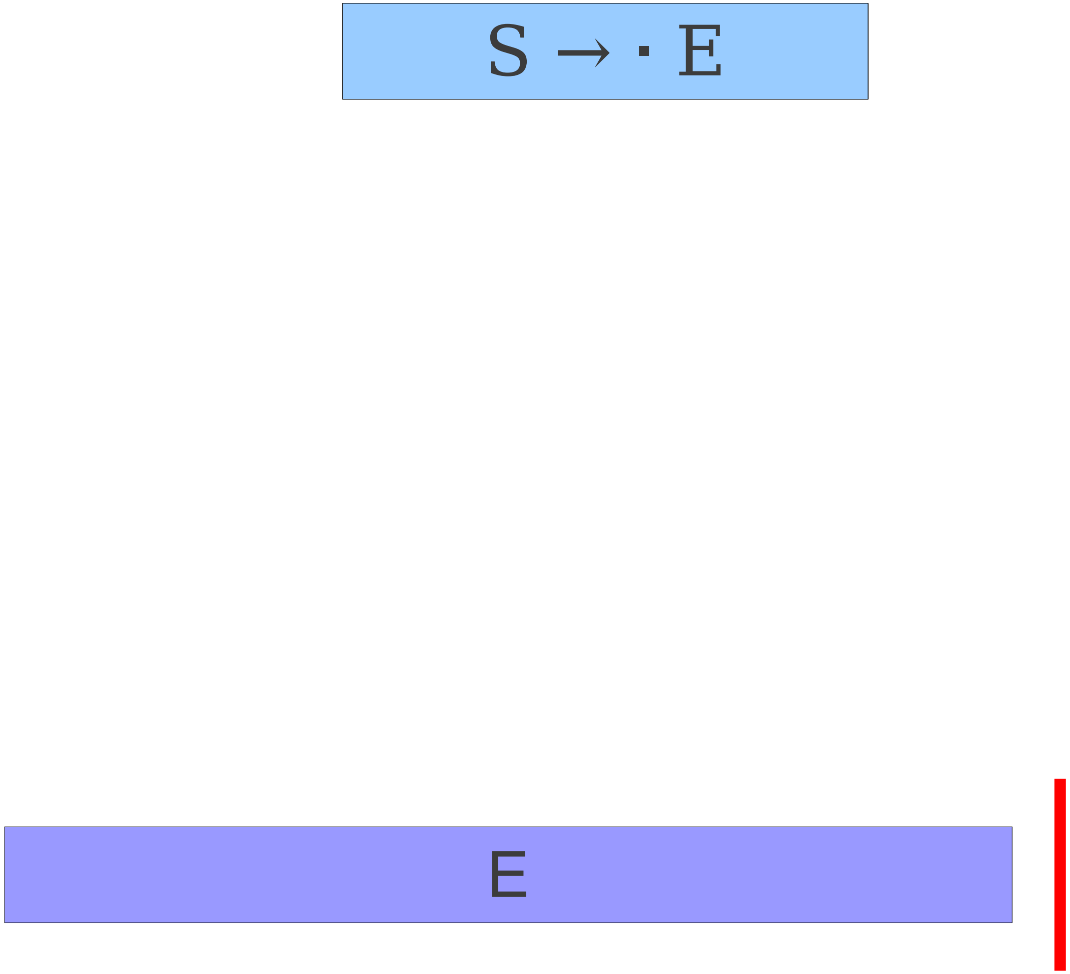
Tracking Our Position

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

S → · E
E → E + F ·

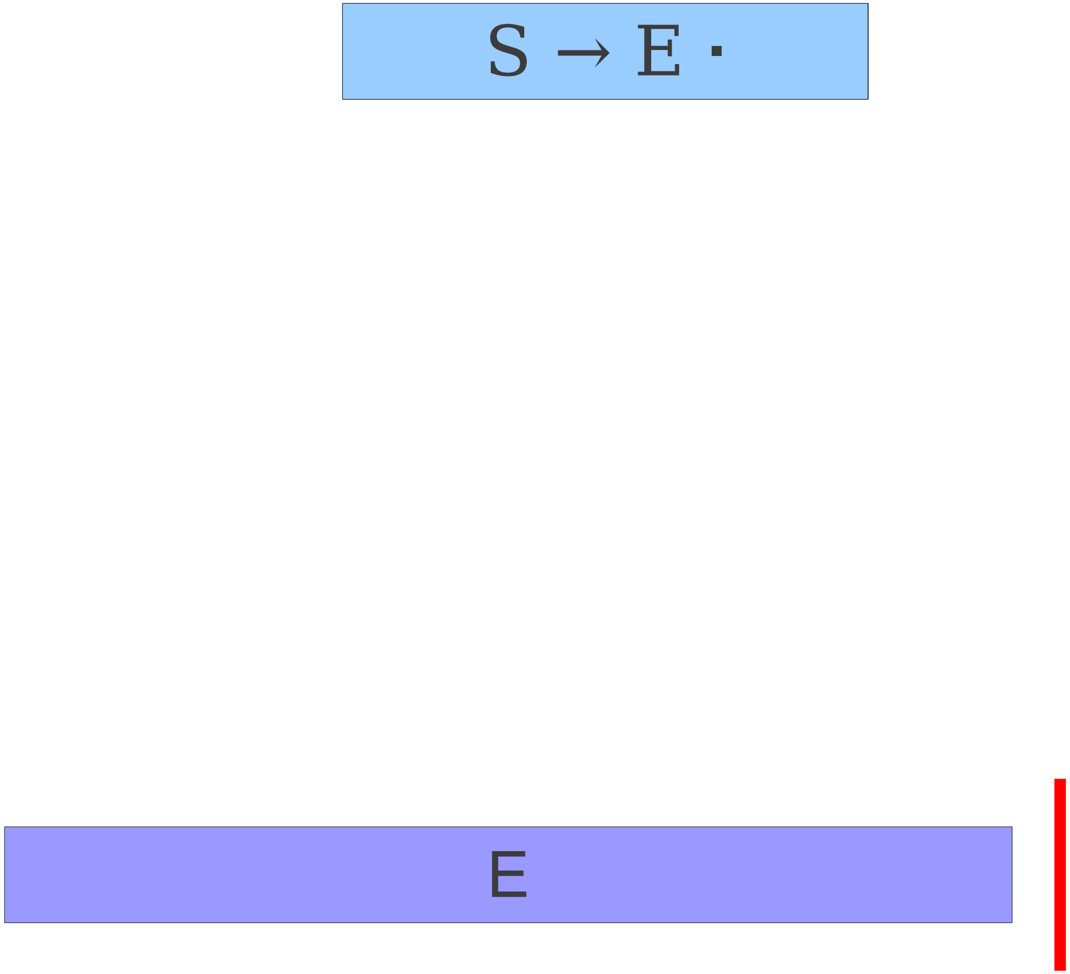
Tracking Our Position

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



Tracking Our Position

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



S → E ·

E

Generating Left-Hand Sides

- At any instant in time, the contents of the left side of the parser can be described using the following process:
  - Trace out, from the start symbol, the series of productions that have not yet been completed and where we are in each production.
  - For each production, in order, output all of the symbols up to the point where we change from one production to the next.

Recognizing Left-Hand Sides

- Given that we have a procedure for *generating* left-hand sides, can we build a procedure for *recognizing* those 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 of the candidate left-hand side with the next symbol in the current production, or
  - If the next symbol of the candidate left-hand side is a nonterminal, nondeterministically guess which production to try next.

Recognizing Left-Hand Sides

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

Recognizing Left-Hand Sides

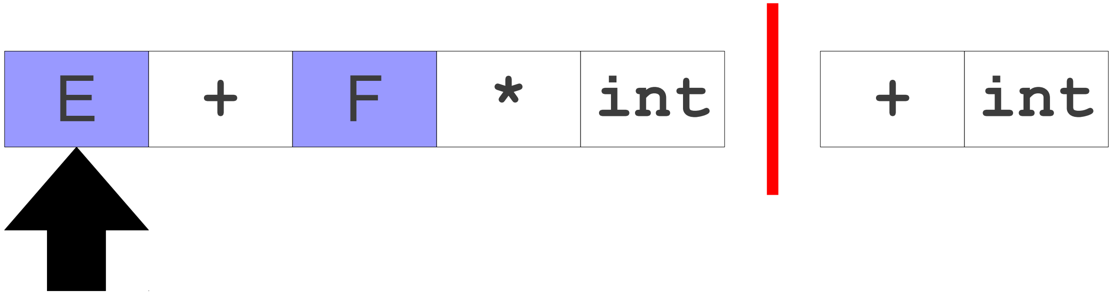
S → · E

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**)

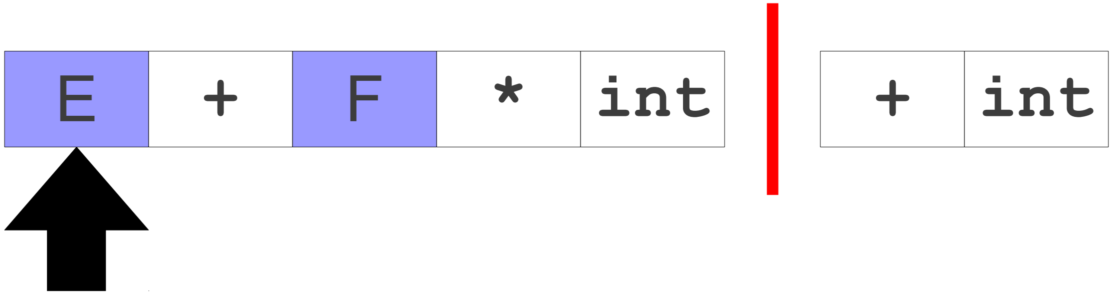
S → · E



Recognizing Left-Hand Sides

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

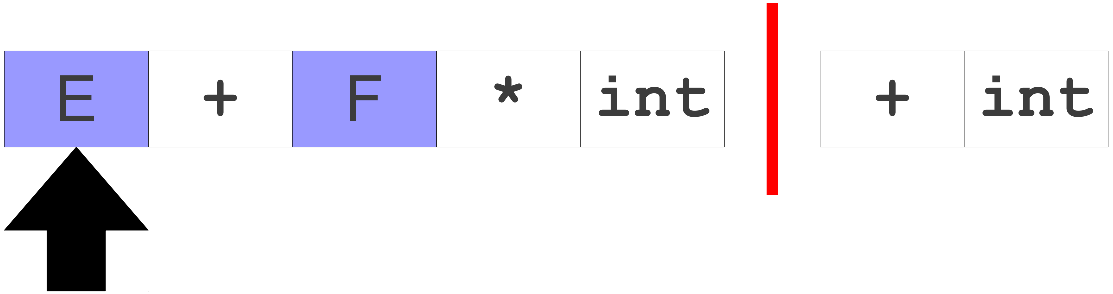
S → · E
E → · E + F


E + F * int | + int

Recognizing Left-Hand Sides

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

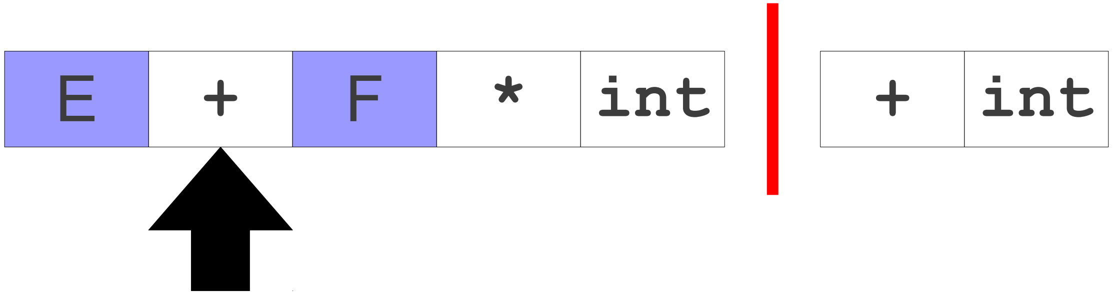
S → · E
E → · E + F
E → · E + F



Recognizing Left-Hand Sides

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

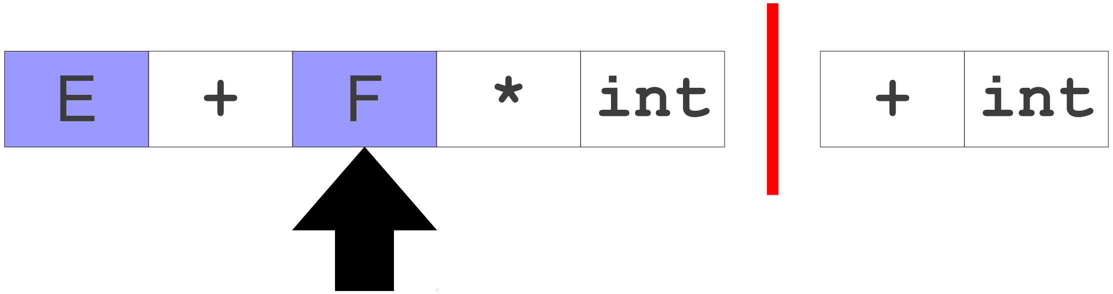
S → · E
E → · E + F
E → E · + F



Recognizing Left-Hand Sides

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

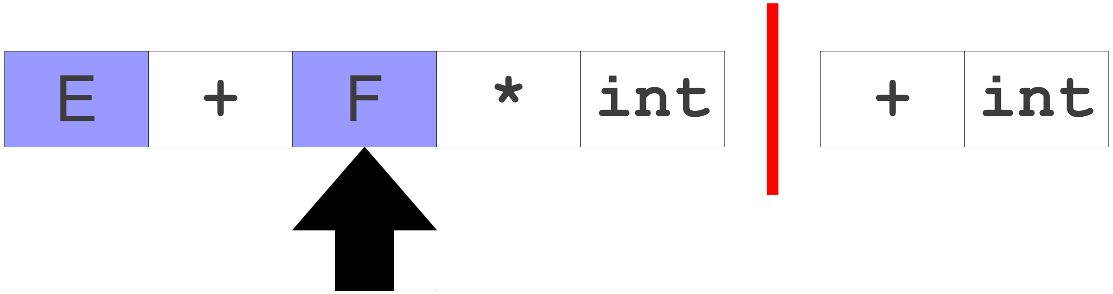
S → · E
E → · E + F
E → E + · F



Recognizing Left-Hand Sides

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

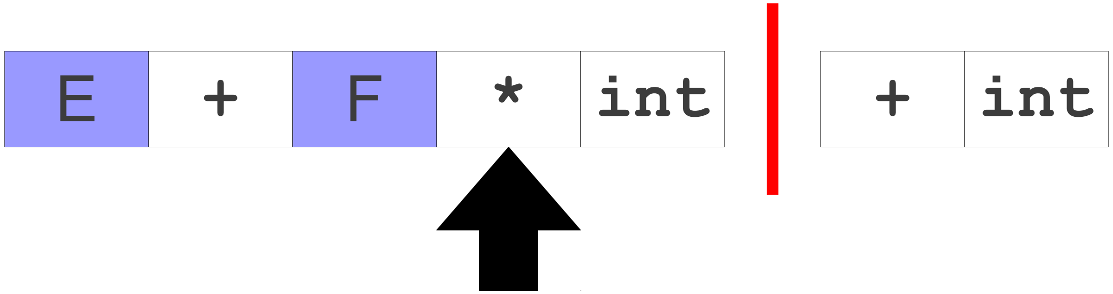
S → · E
E → · E + F
E → E + · F
F → · F * T


E | + | F | * | int || + | int

Recognizing Left-Hand Sides

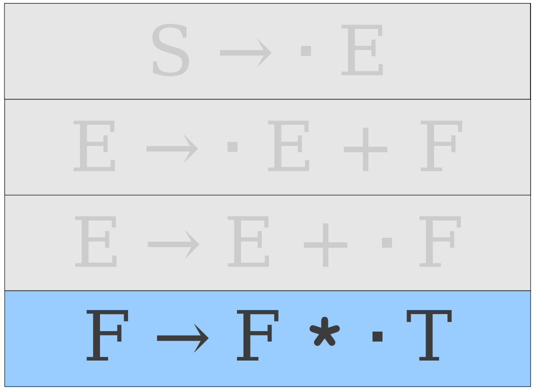
S → **E**
E → **F**
E → **E** + **F**
F → **F** * **T**
F → **T**
T → `int`
T → (**E**)

S → · E
E → · E + F
E → E + · F
F → F · * T

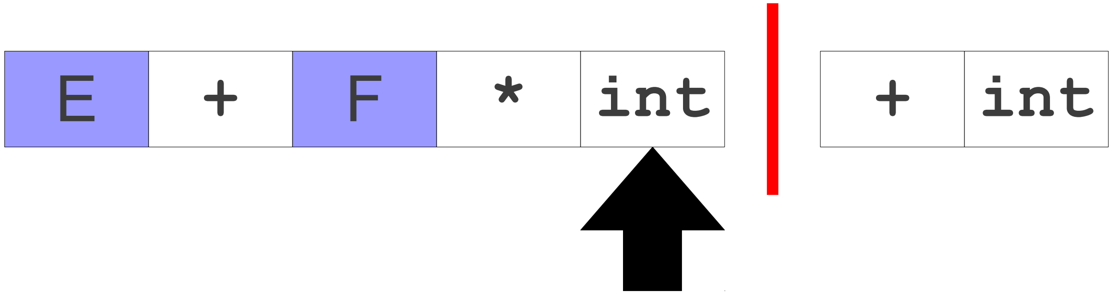


Recognizing Left-Hand Sides

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



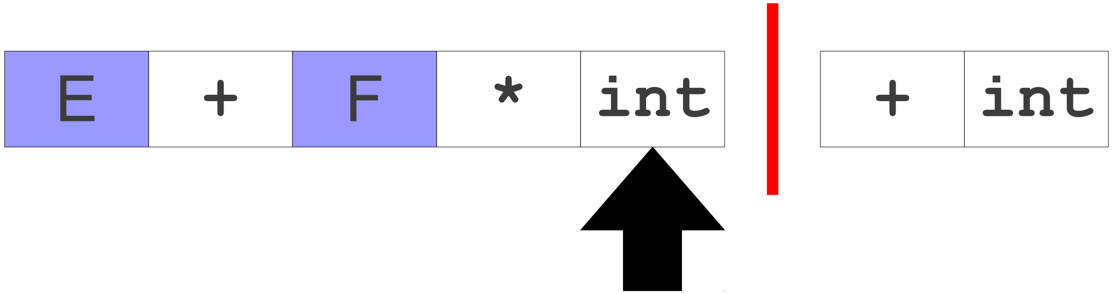
S → · E
E → · E + F
E → E + · F
F → F * · T



Recognizing Left-Hand Sides

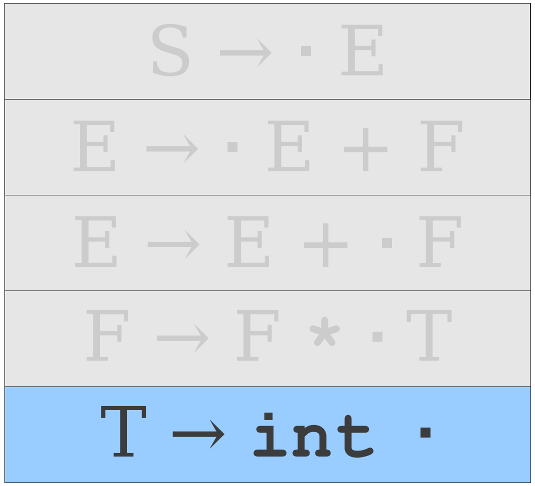
S → **E**
E → **F**
E → **E** + **F**
F → **F** * **T**
F → **T**
T → `int`
T → (**E**)

S → · E
E → · E + F
E → E + · F
F → F * · T
T → · `int`



Recognizing Left-Hand Sides

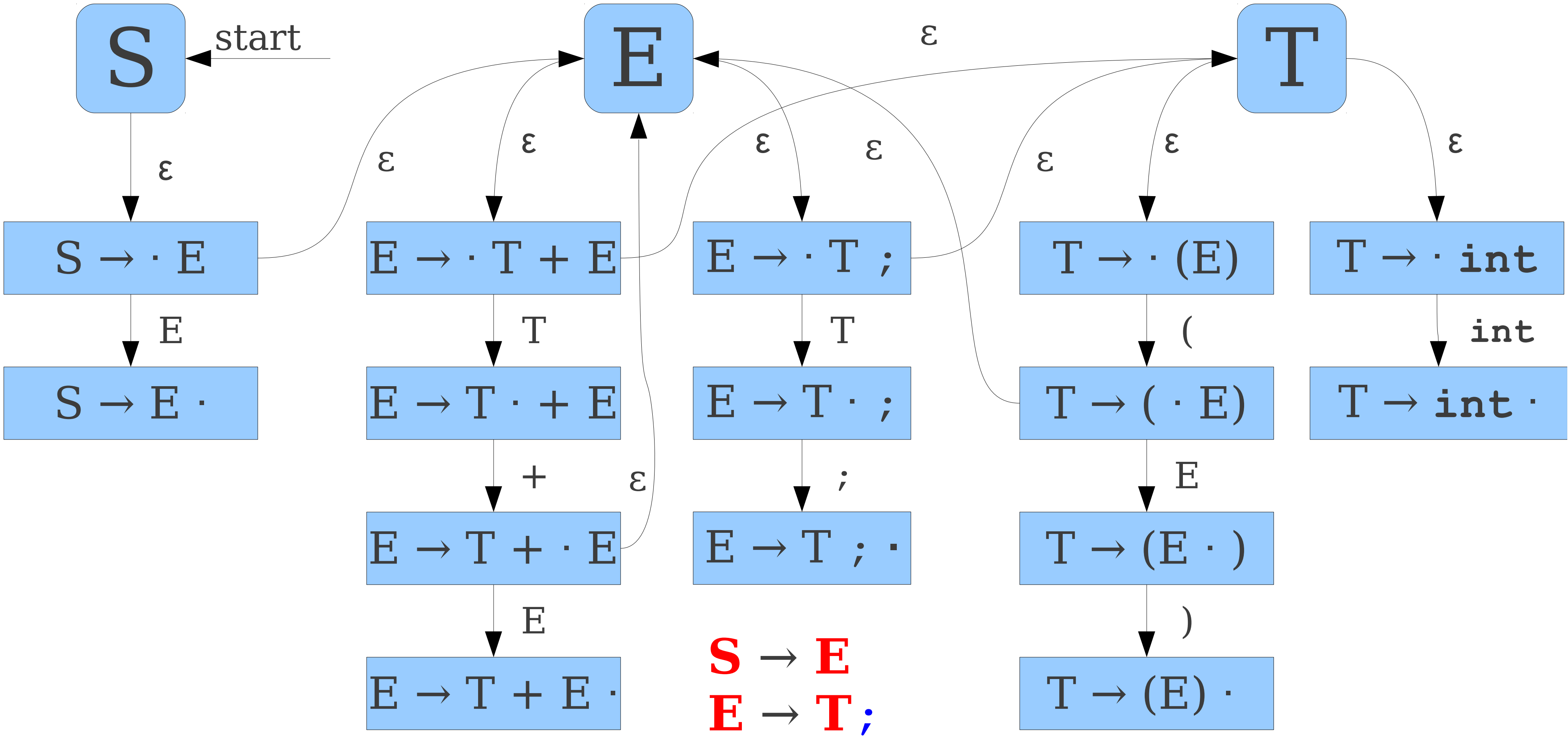
S → **E**
E → **F**
E → **E** `+` **F**
F → **F** `*` **T**
F → **T**
T → `int`
T → `(`**E**`)`


S → · E
E → · E + F
E → E + · F
F → F * · T
T → `int` ·

An Important Result

- There are only finitely many productions, and within those productions only finitely many positions.
- At any point in time, we only need to track where we are in one production.
- There are only finitely many options we can take at any one point.
- **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 $\mathbf{A} \to \boldsymbol{\gamma}$:
  - Construct states $\mathbf{A} \to \boldsymbol{\alpha} \cdot \boldsymbol{\omega}$ for each possible way of splitting $\boldsymbol{\gamma}$ into two substrings $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$.
  - Add transitions on $\boldsymbol{x}$ between $\mathbf{A} \to \boldsymbol{\alpha} \cdot \boldsymbol{x\omega}$ and $\mathbf{A} \to \boldsymbol{\alpha x} \cdot \boldsymbol{\omega}$.
- For each state $\mathbf{A} \to \boldsymbol{\alpha} \cdot \mathbf{B}\boldsymbol{\omega}$ for nonterminal $\mathbf{B}$, add an ε-transition from $\mathbf{A} \to \boldsymbol{\alpha} \cdot \mathbf{B}\boldsymbol{\omega}$ to $\mathbf{B}$.

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

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

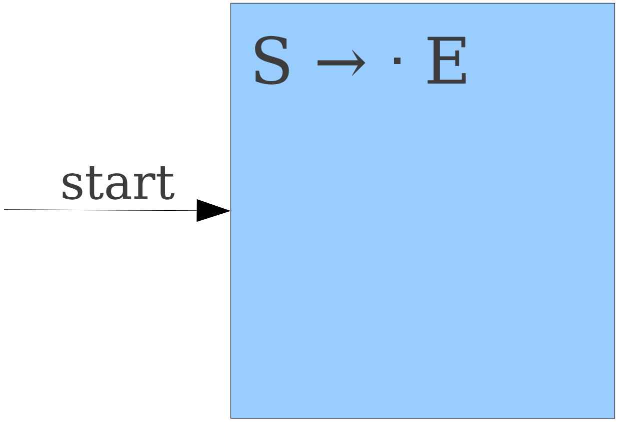
- Then we might be looking at a handle.
- This automaton can be used to discover possible handle locations!

Adding Determinism

- Typically, this handle-finding automaton is implemented deterministically.
- We could construct a deterministic parsing automaton by constructing the nondeterministic automaton and applying the subset construction, but there is a more direct approach.

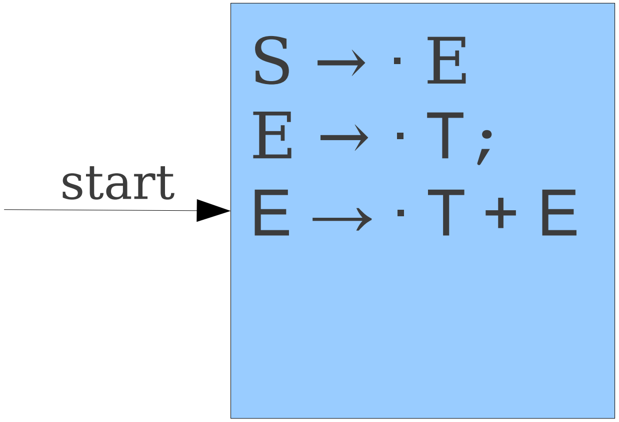
A Deterministic Automaton

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**`)`

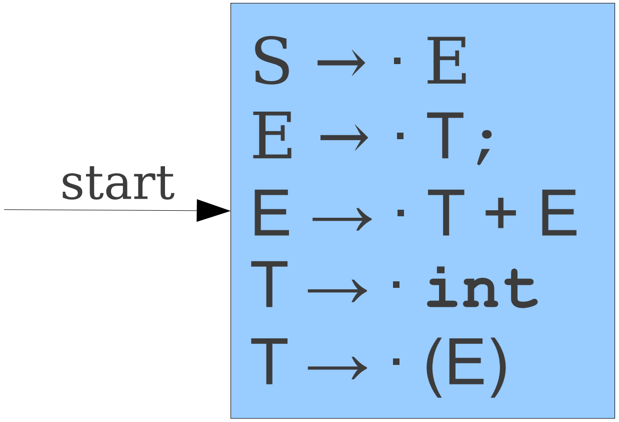


start

S → · E
E → · T;
E → · T + E

A Deterministic Automaton

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

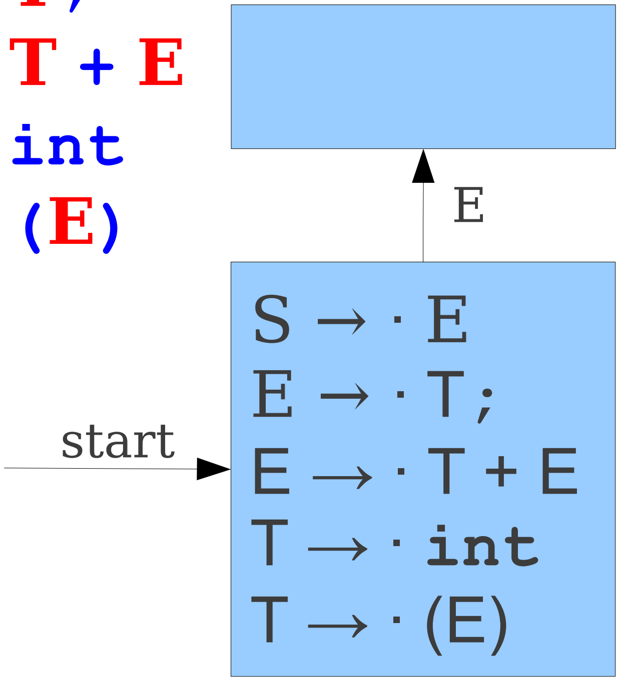


start

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

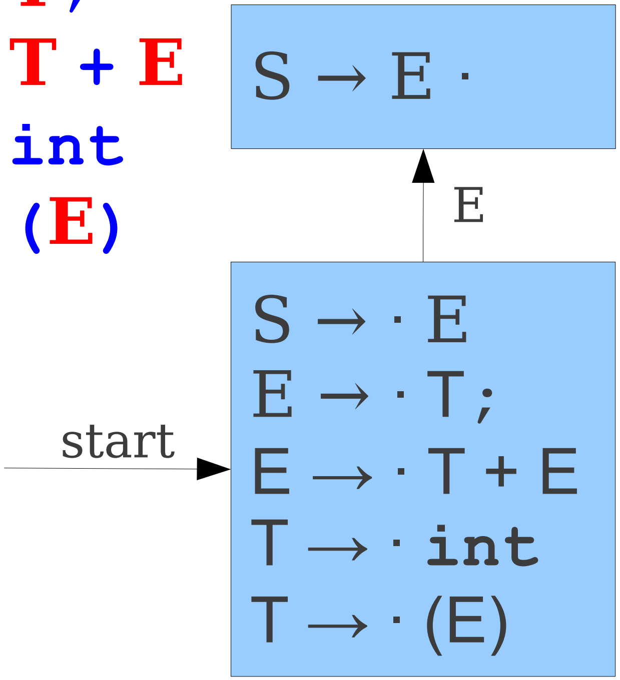
A Deterministic Automaton

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



A Deterministic Automaton

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



S → E ·

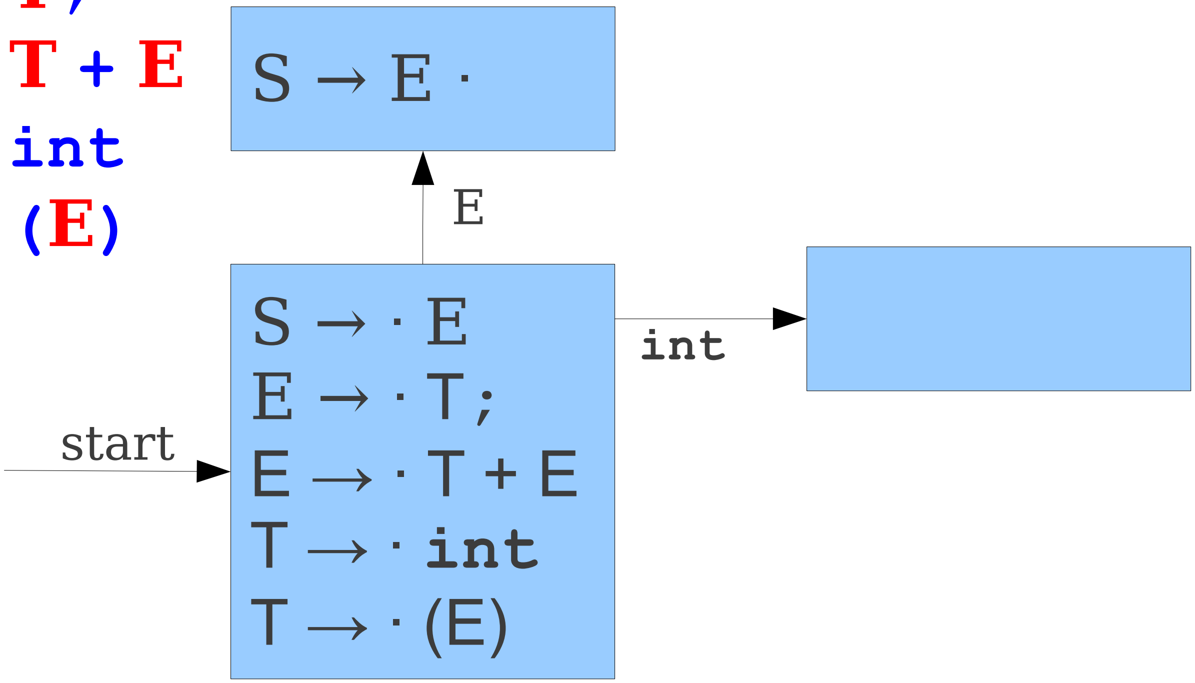
E

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**)

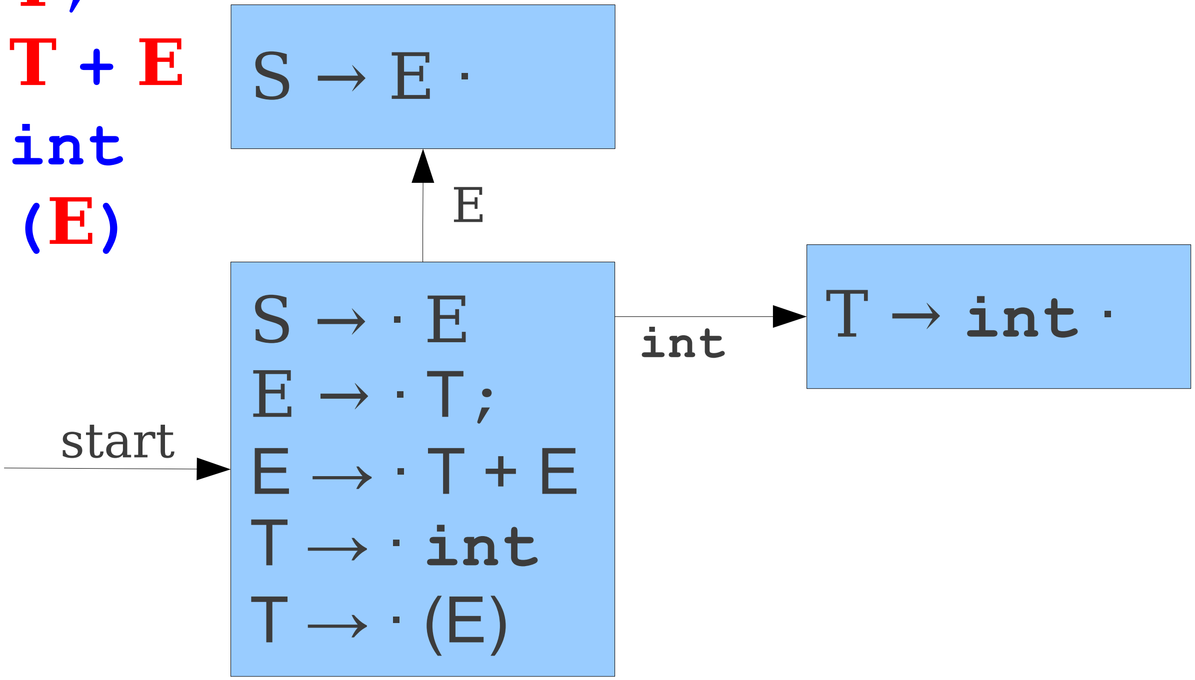


S → E ·

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

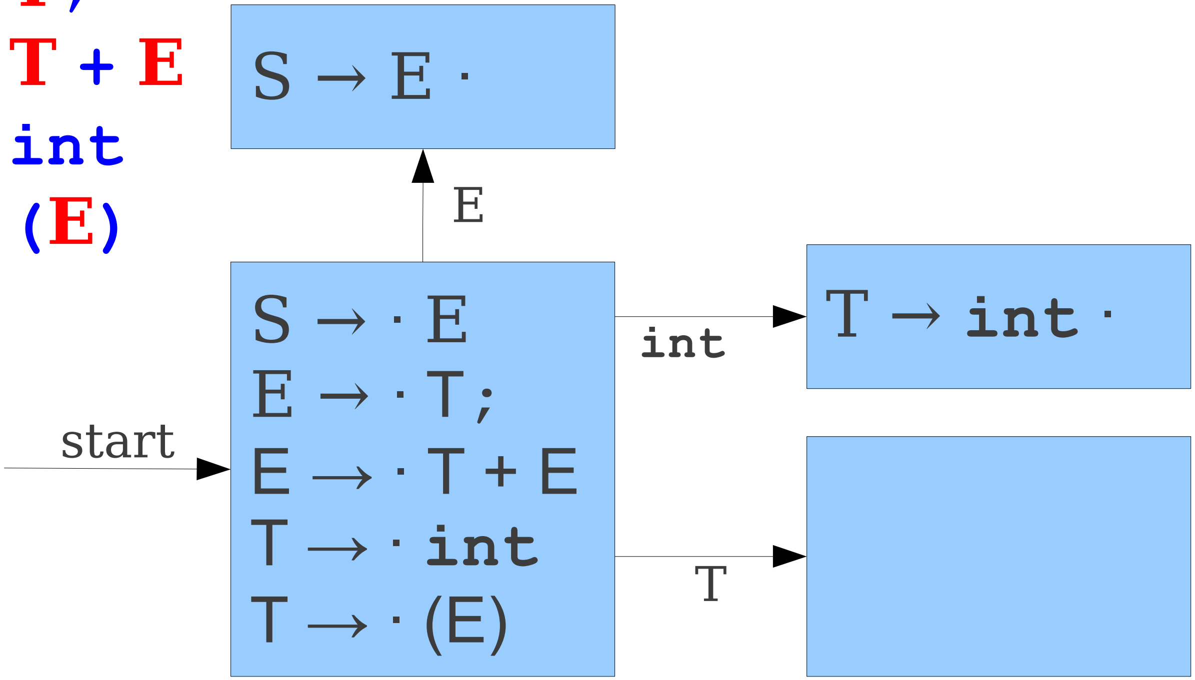
A Deterministic Automaton

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



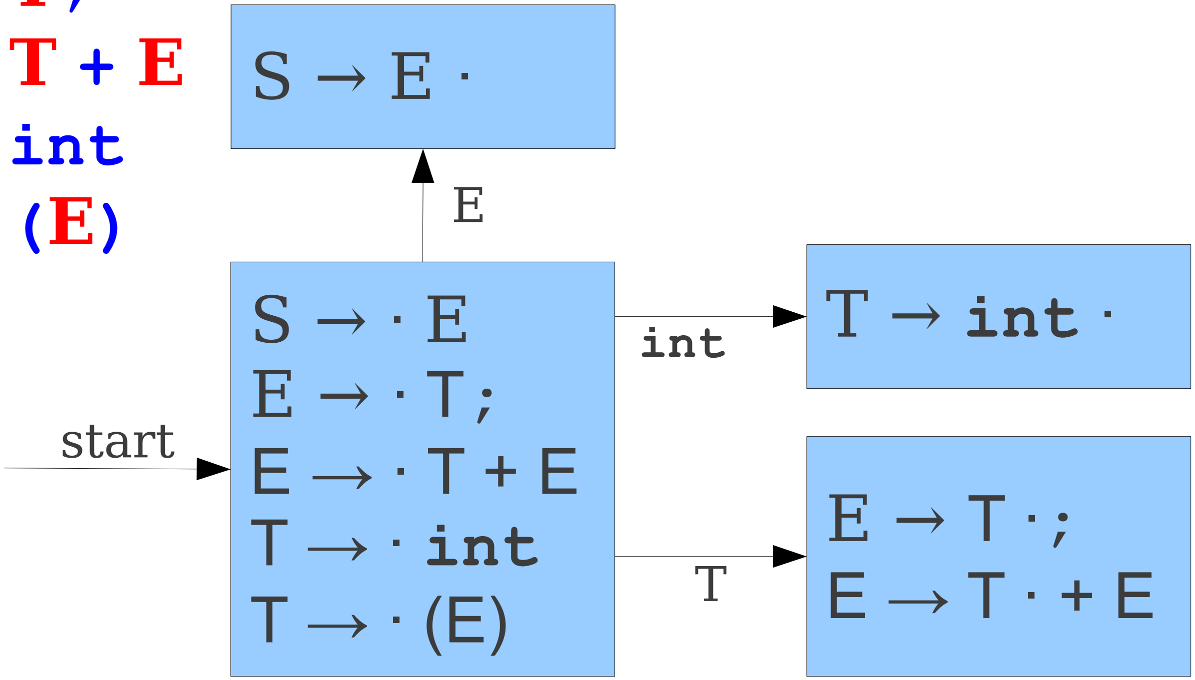
A Deterministic Automaton

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)

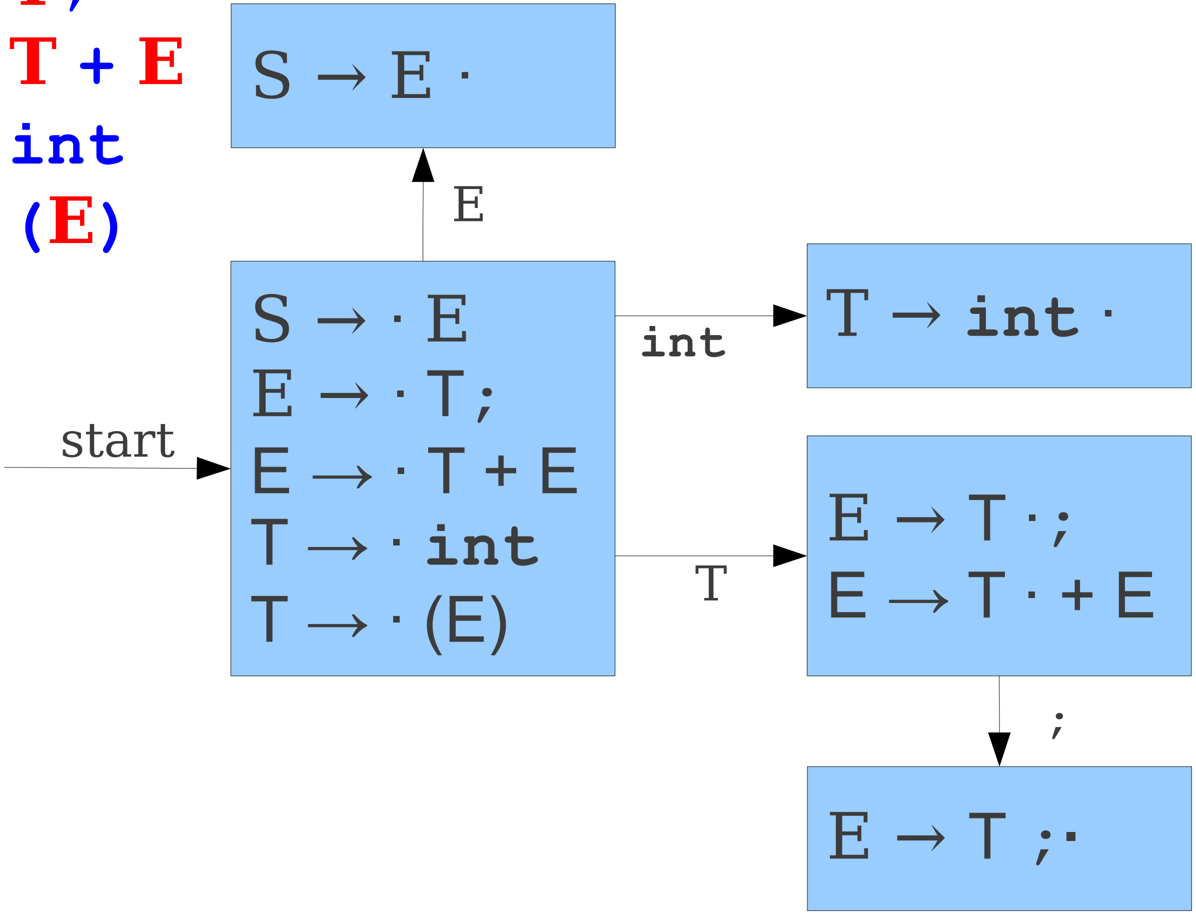
S → E ·

T → int ·

E → T · ;
E → T · + E

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)

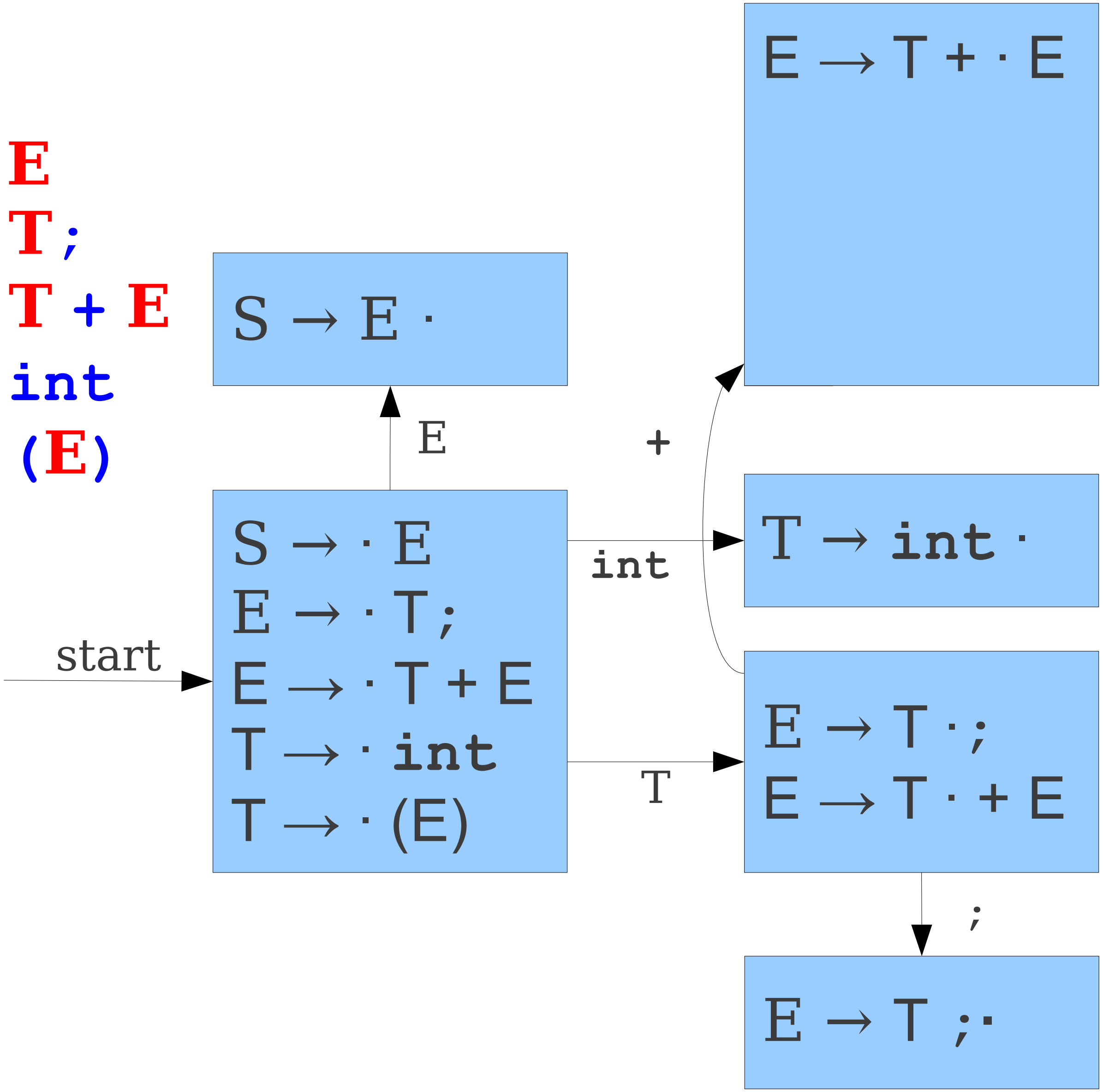
T → `int` ·

E → T · ;
E → T · + E

E → T ; ·

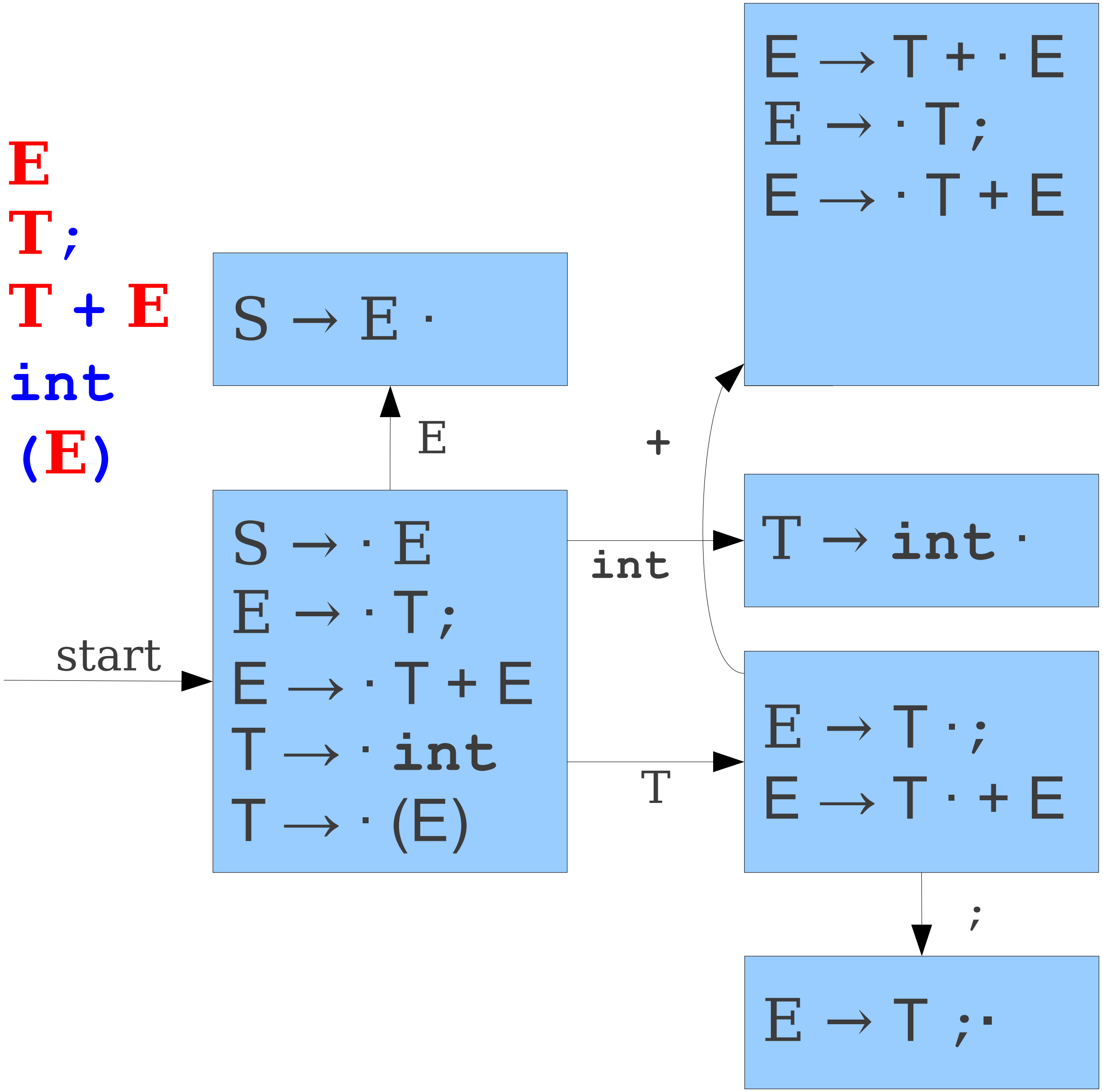
A Deterministic Automaton

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



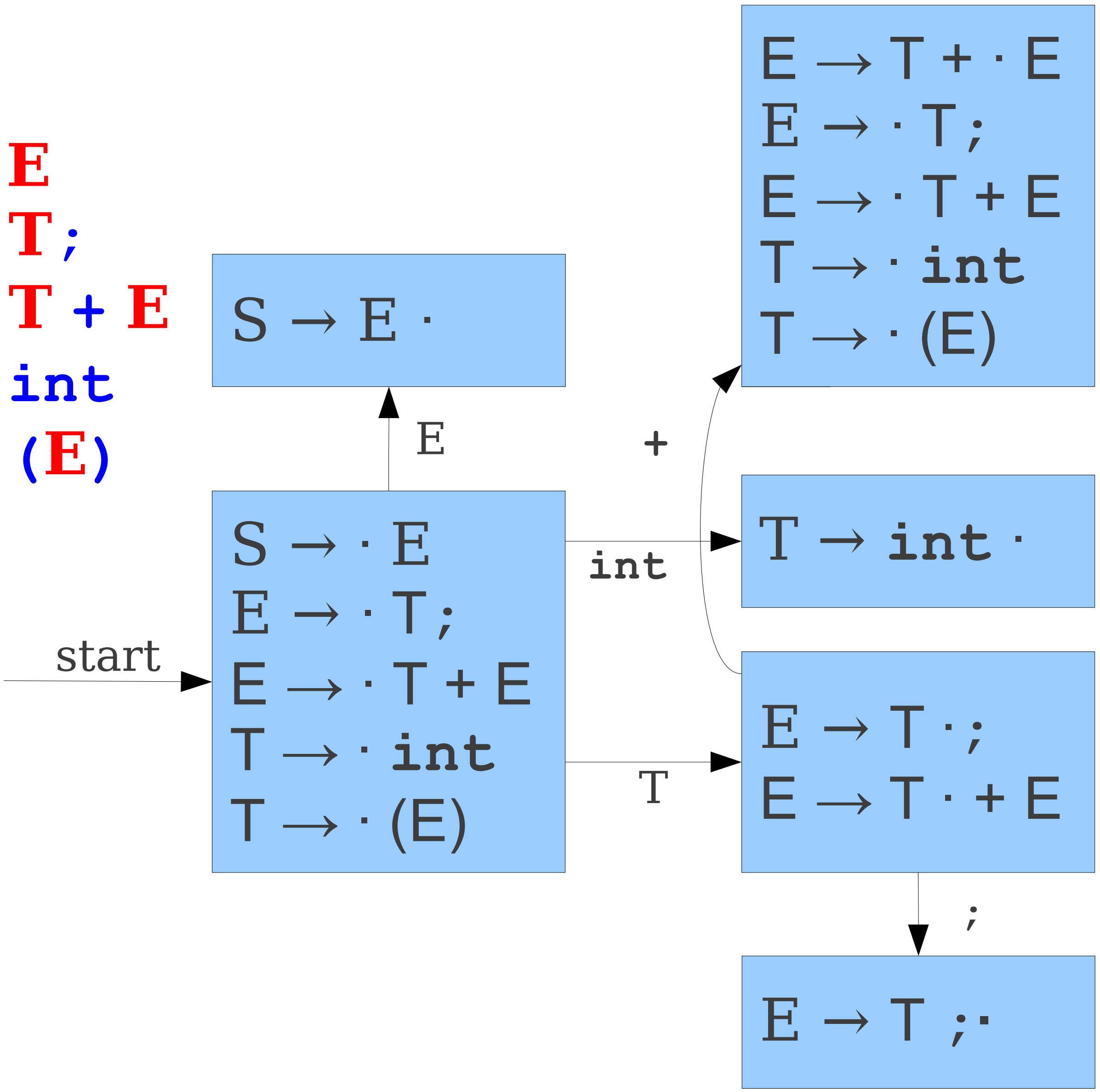
A Deterministic Automaton

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



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`
T → · (E)

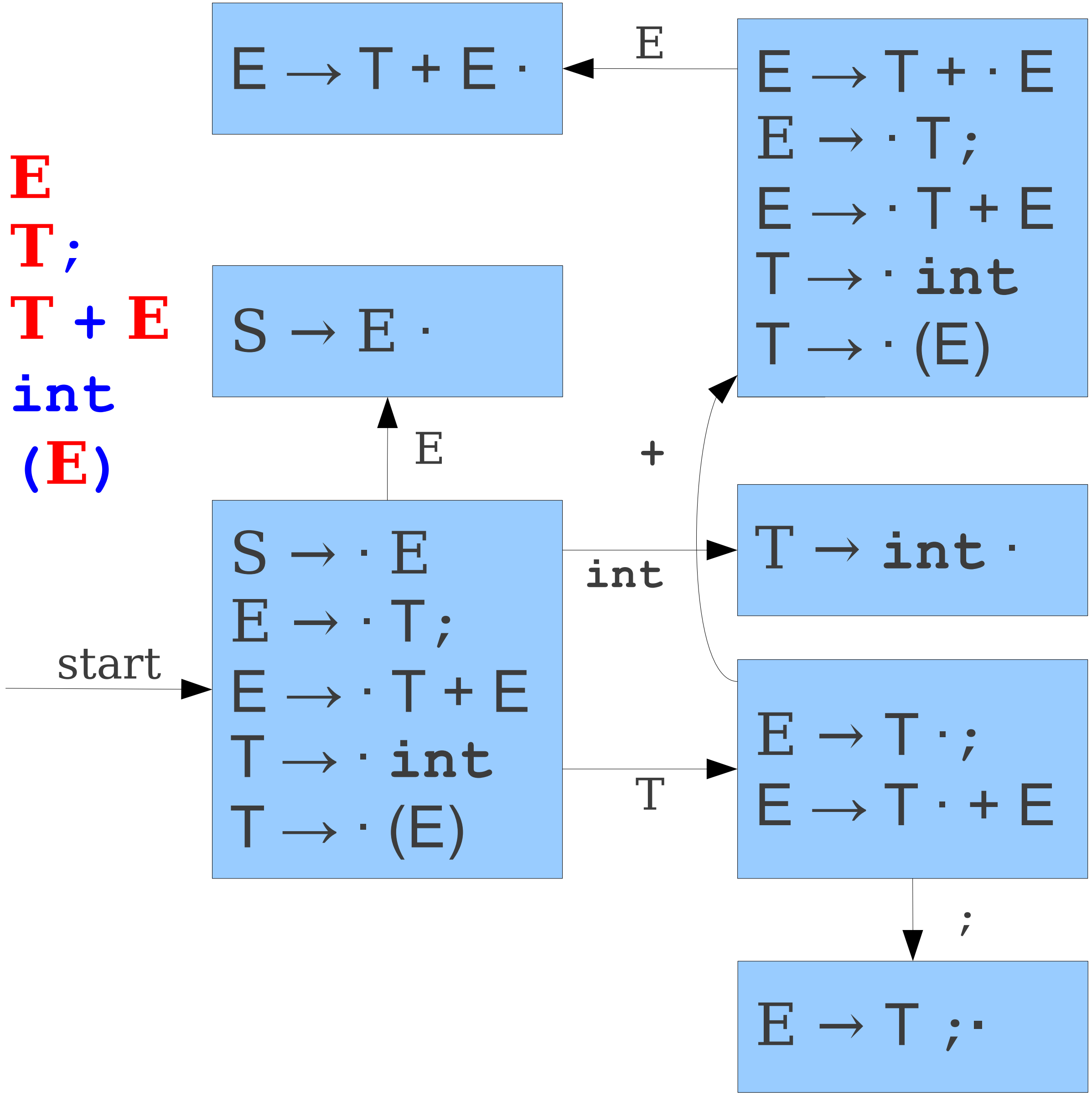
T → `int` ·

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)

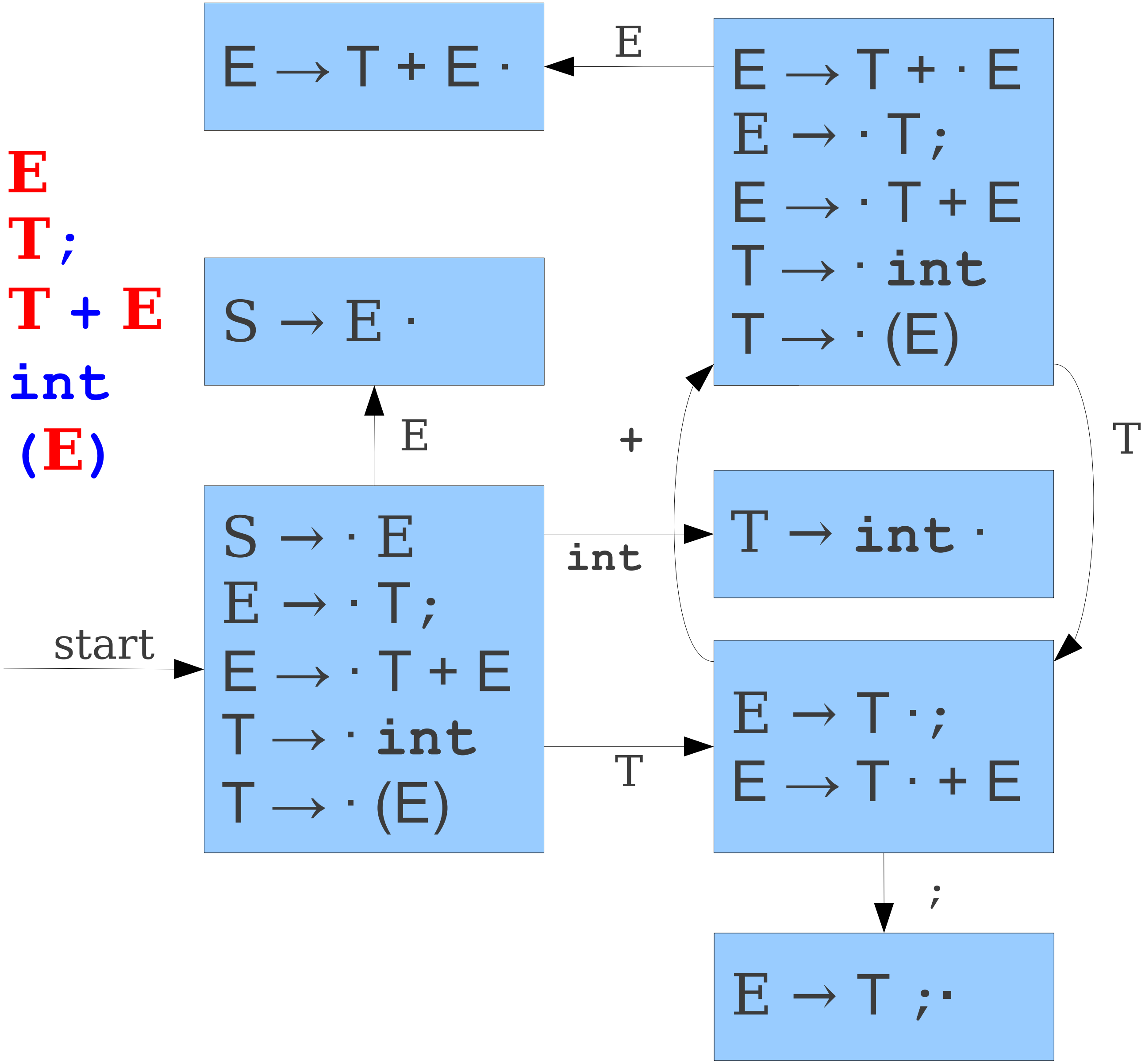
T → `int` ·

E → T · ;
E → T · + E

E → T ;·

A Deterministic Automaton

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



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)

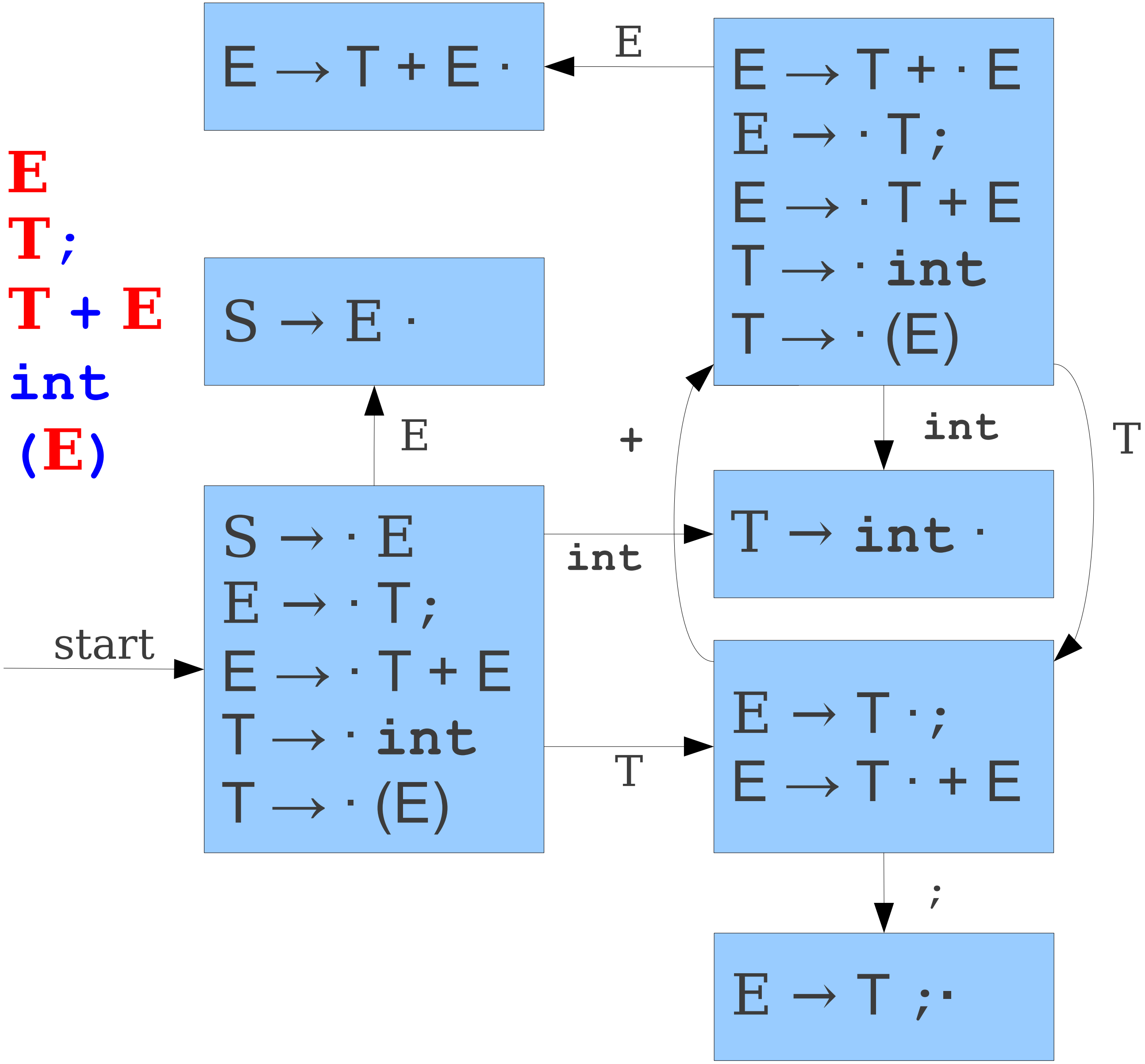
T → `int` ·

E → T · ;
E → T · + E

E → T ; ·

A Deterministic Automaton

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



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)

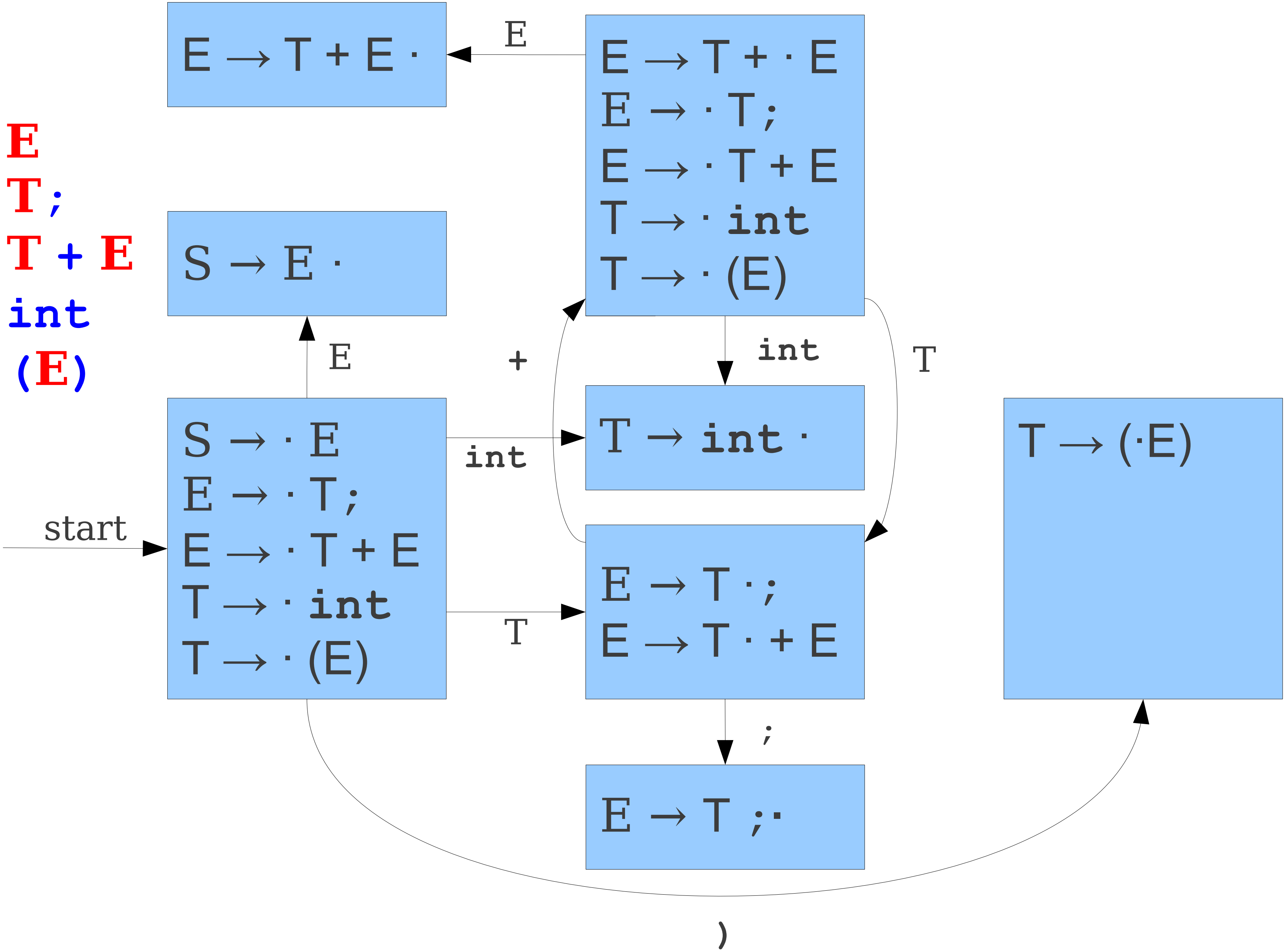
T → int ·

E → T · ;
E → T · + E

E → T ; ·

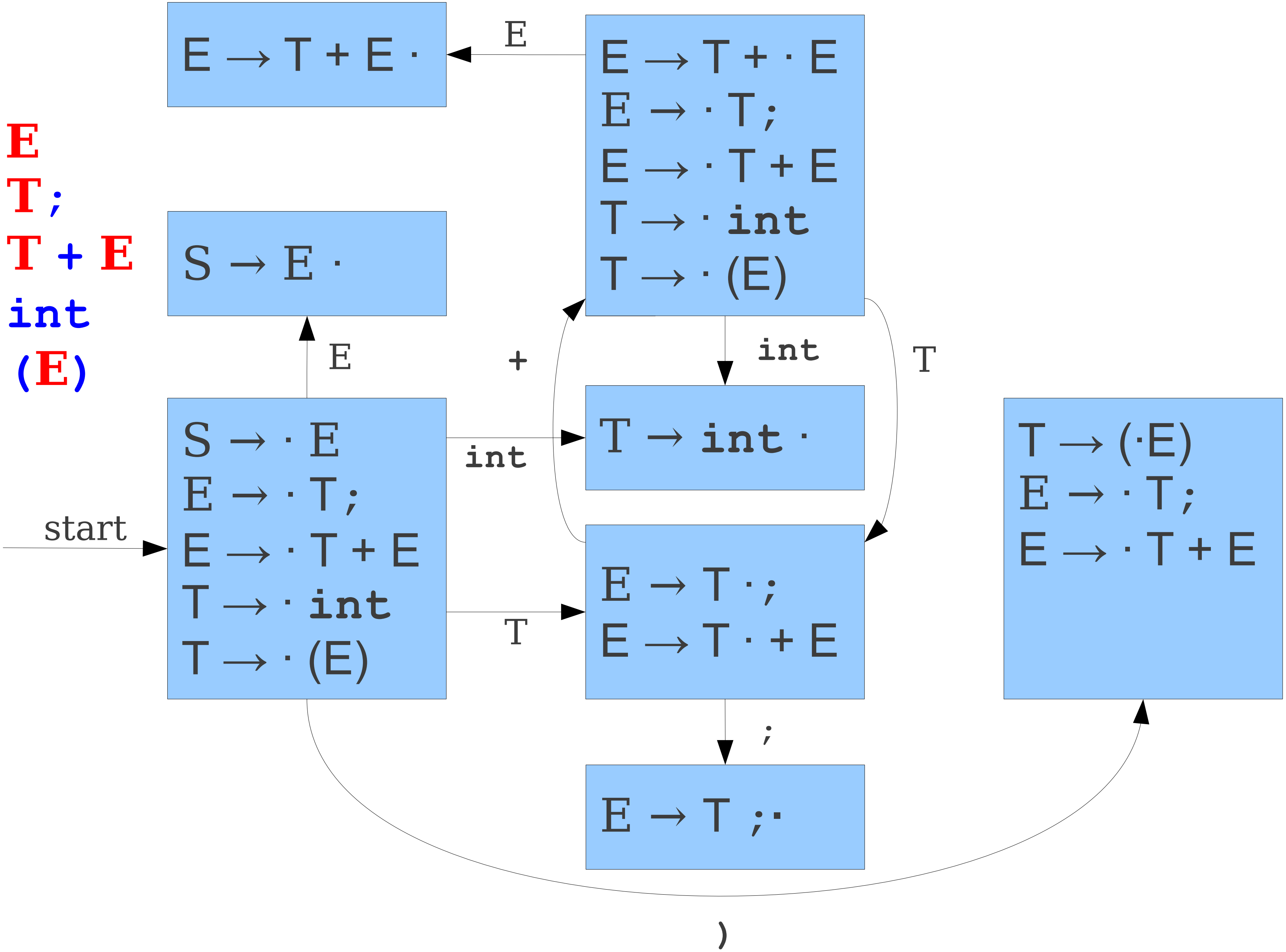
A Deterministic Automaton

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



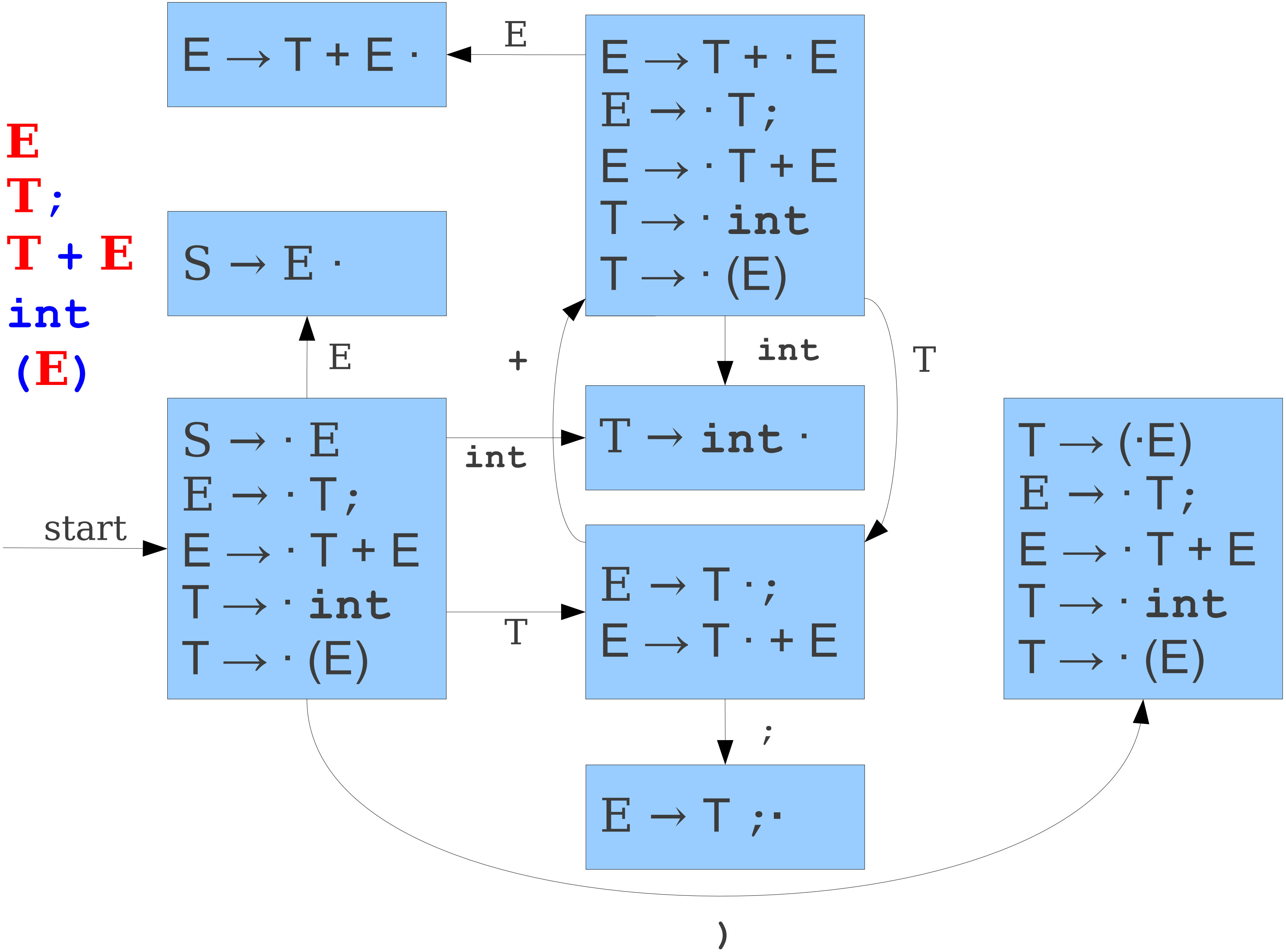
A Deterministic Automaton

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 ·

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` ·

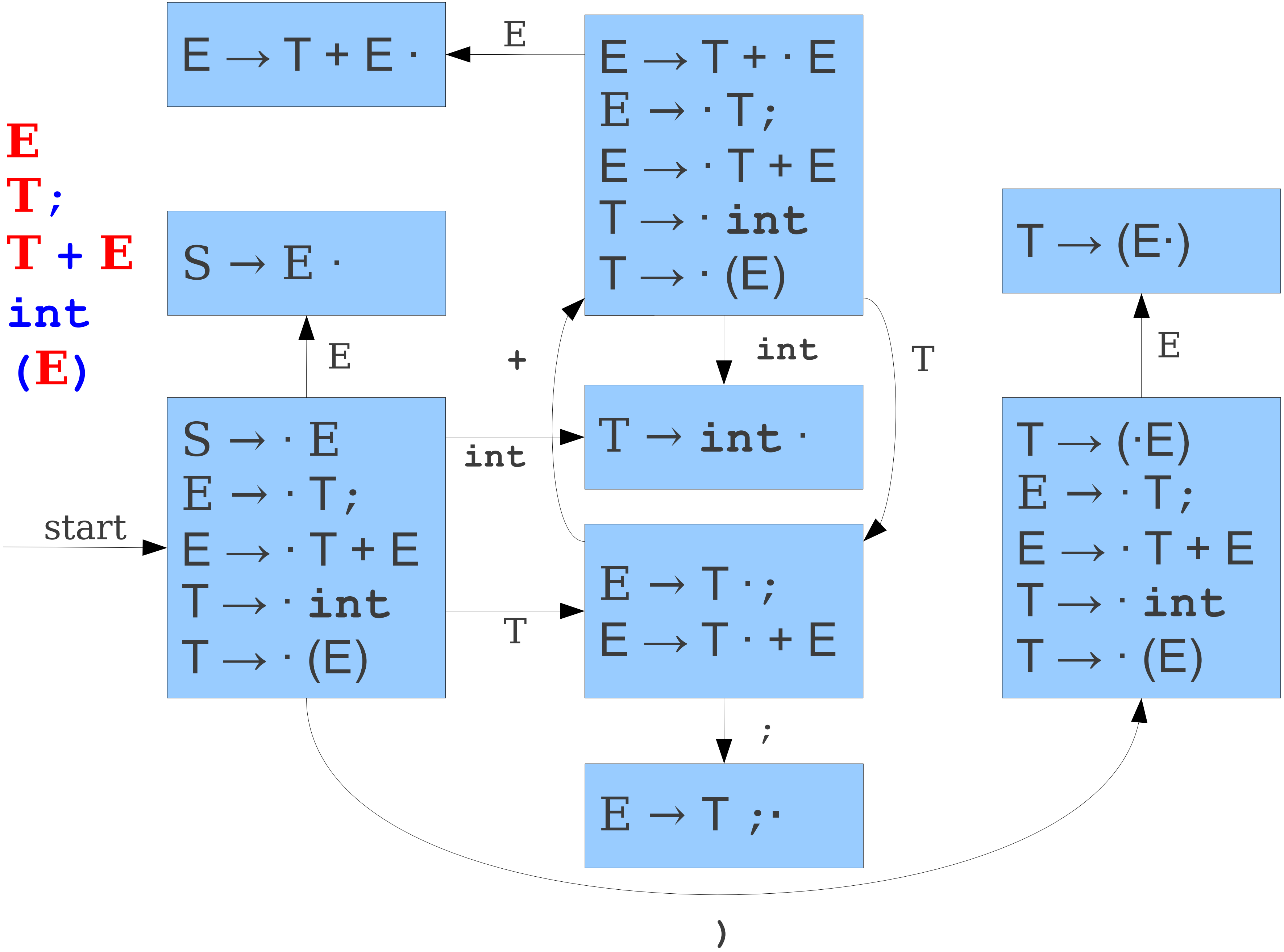
E → T ·;
E → T · + E

E → T ;·

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

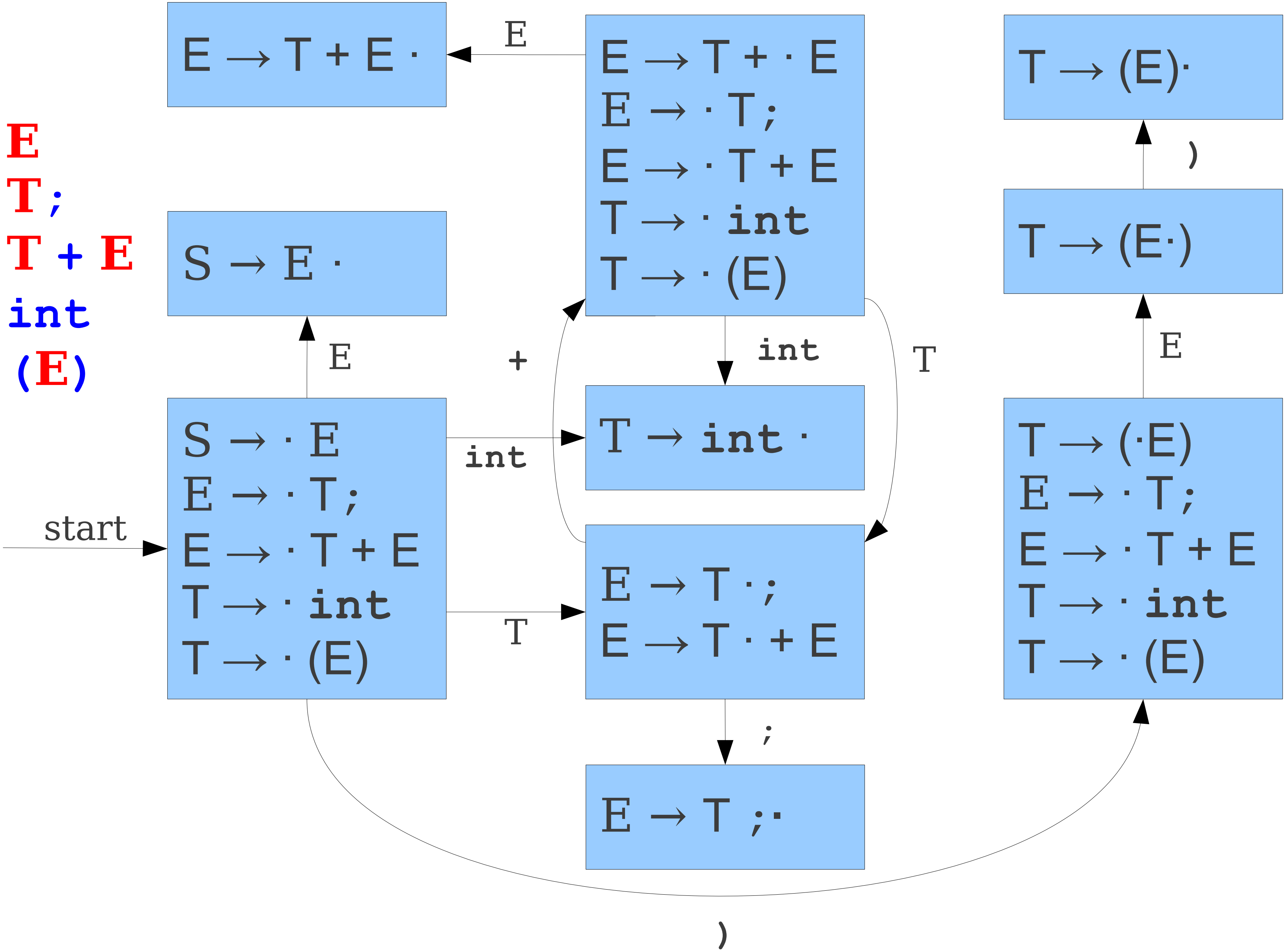
A Deterministic Automaton

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)**



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 ·

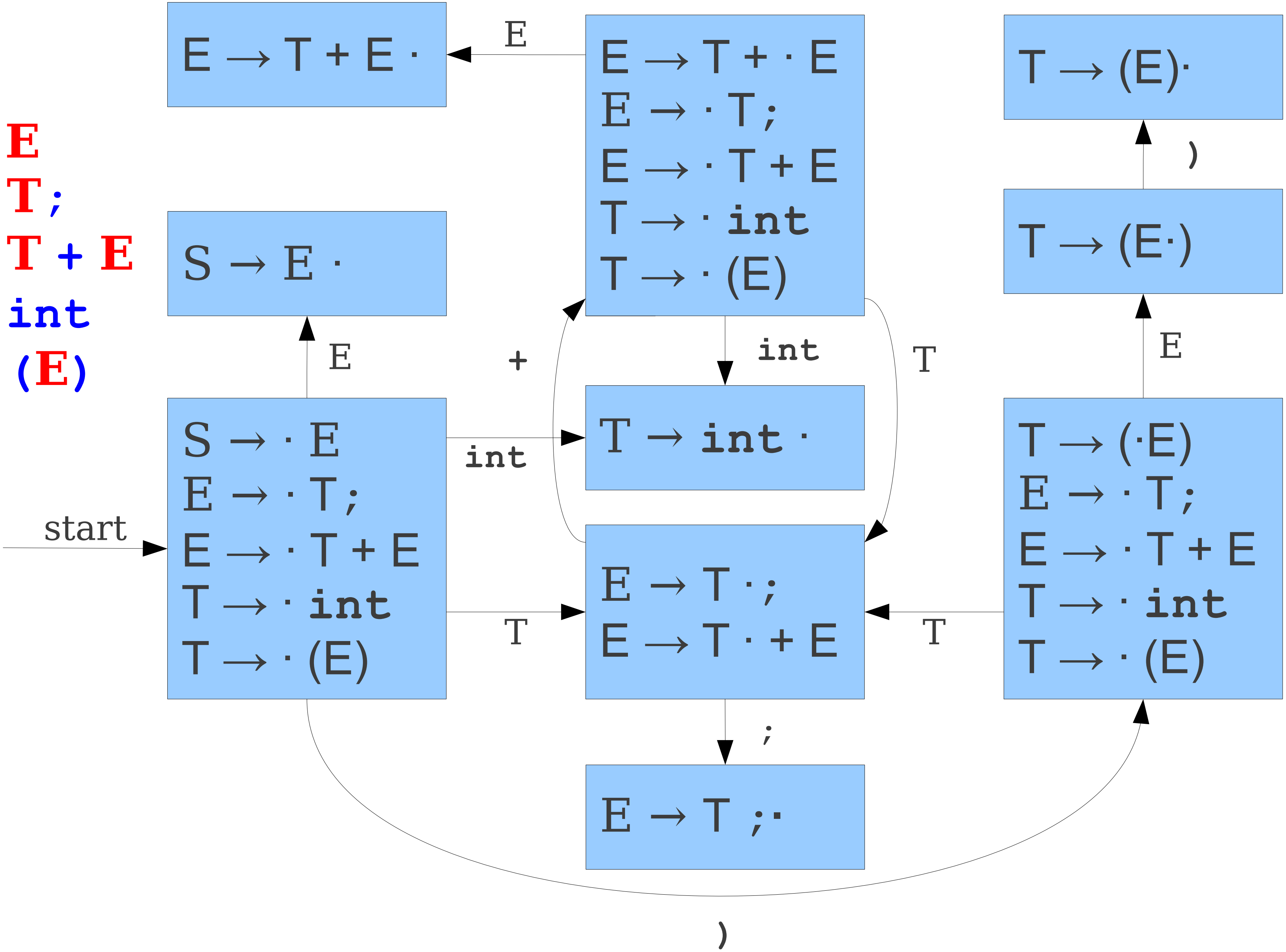
E → T · ;
E → T · + E

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

E → T ;·

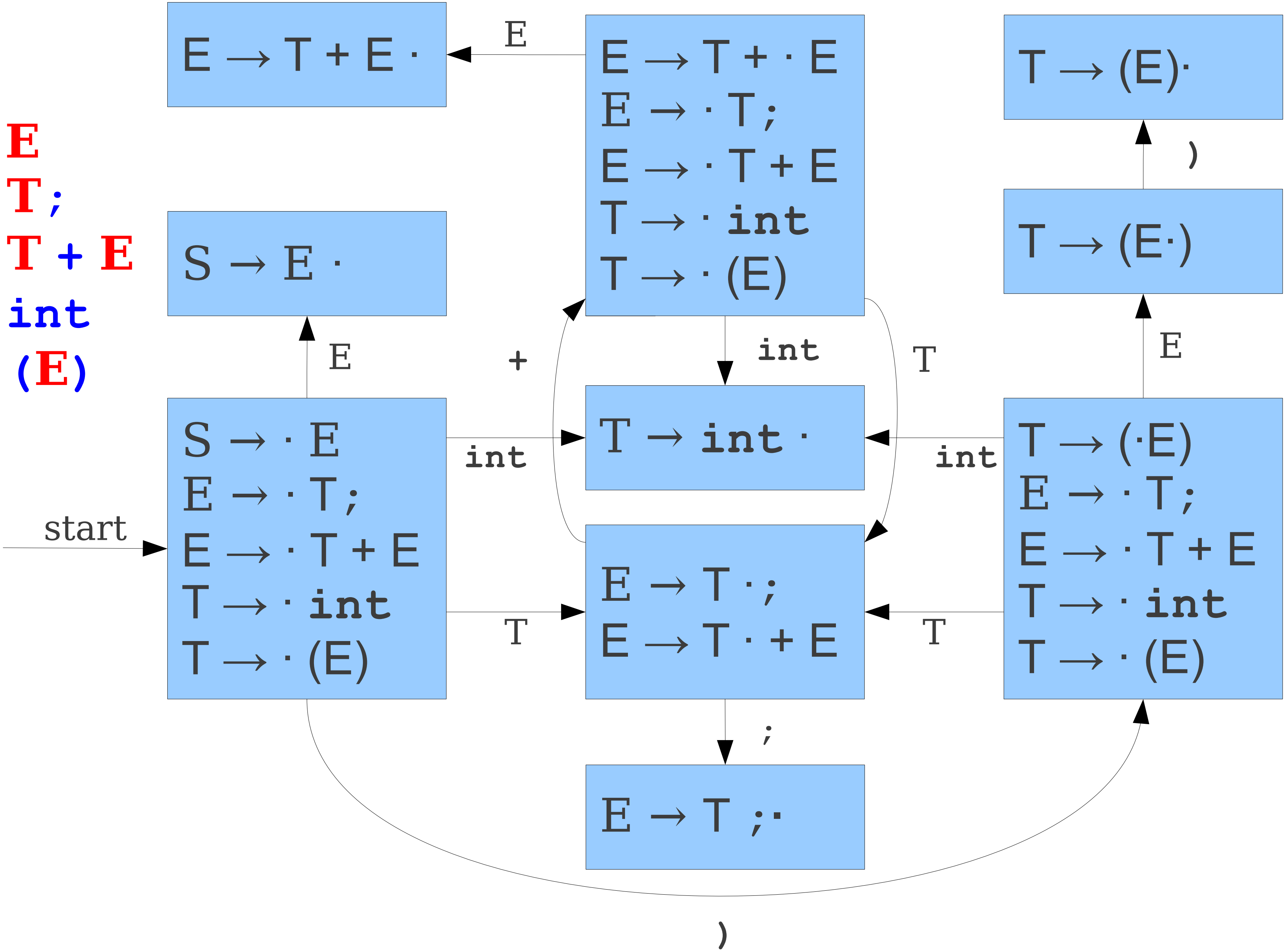
A Deterministic Automaton

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



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 ·

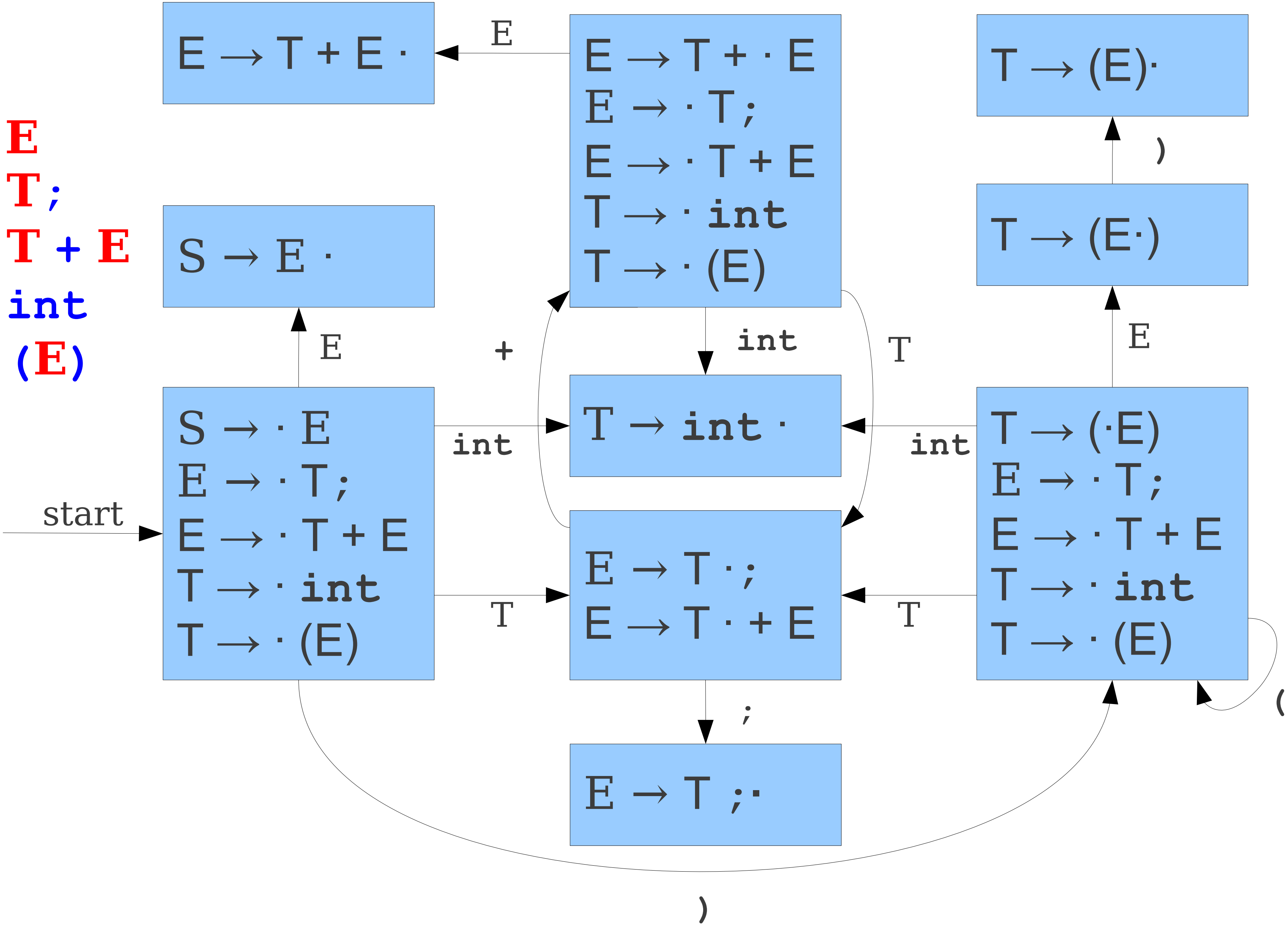
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

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)·

S → E ·

T → (E·)

T → int ·

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

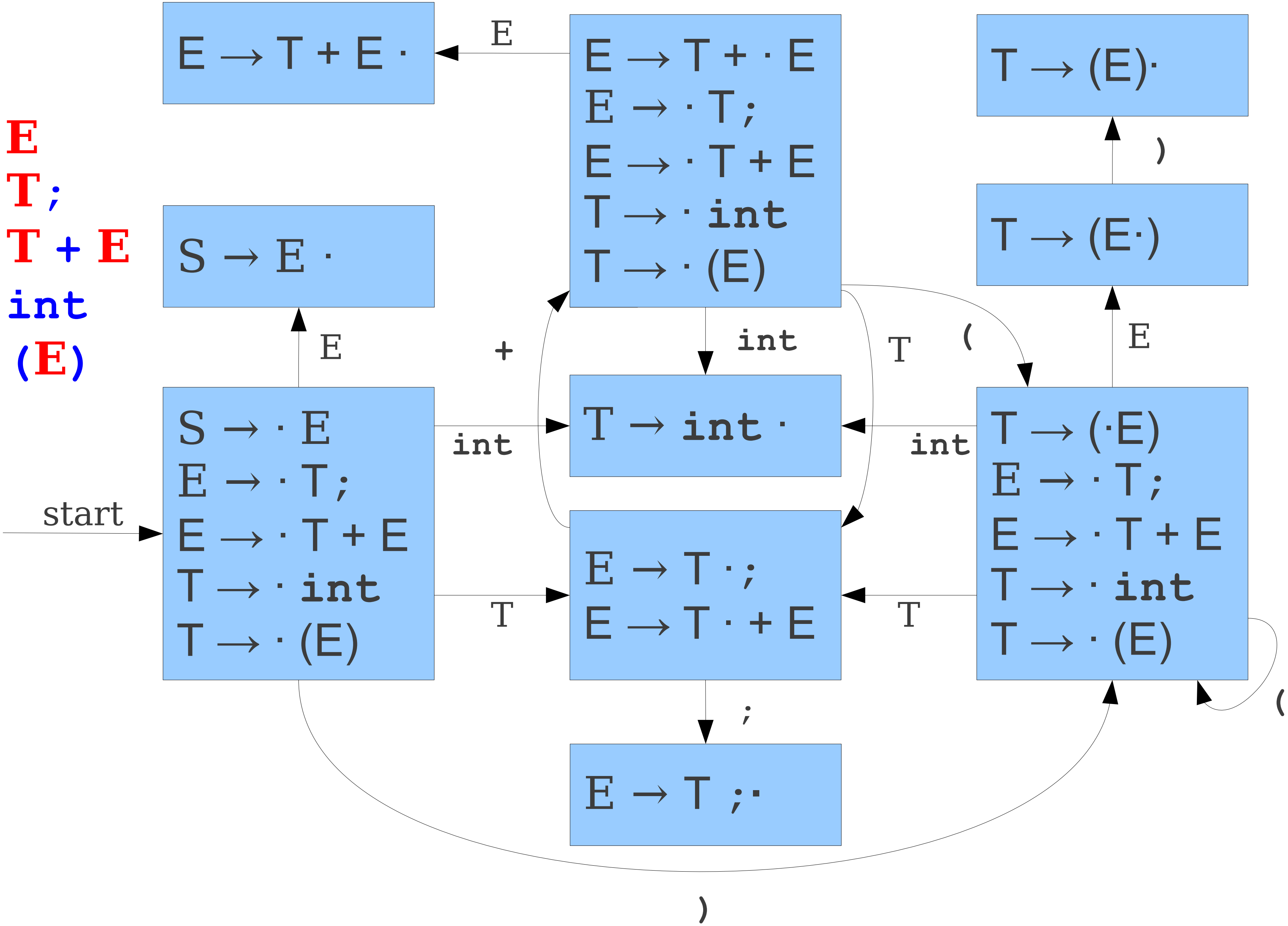
E → T · ;
E → T · + E

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

E → T ;·

A Deterministic Automaton

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



Constructing the Automaton II

- Begin in a state containing $\mathbf{S} \to \cdot\, \mathbf{A}$, where **S** is the augmented start symbol.
- Compute the **closure** of the state:
  - If $\mathbf{A} \to \boldsymbol{\alpha} \cdot \mathbf{B}\boldsymbol{\omega}$ is in the state, add $\mathbf{B} \to \cdot\, \boldsymbol{\gamma}$ to the state for each production $\mathbf{B} \to \boldsymbol{\gamma}$.
  - Yet another fixed-point iteration!
- Repeat until no new states are added:
  - If a state contains a production $\mathbf{A} \to \boldsymbol{\alpha} \cdot \boldsymbol{x}\boldsymbol{\omega}$ for symbol $\boldsymbol{x}$, add a transition on $\boldsymbol{x}$ from that state to the state containing the closure of $\mathbf{A} \to \boldsymbol{\alpha}\boldsymbol{x} \cdot \boldsymbol{\omega}$
- This is equivalent to a subset construction on the NFA.

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.

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.

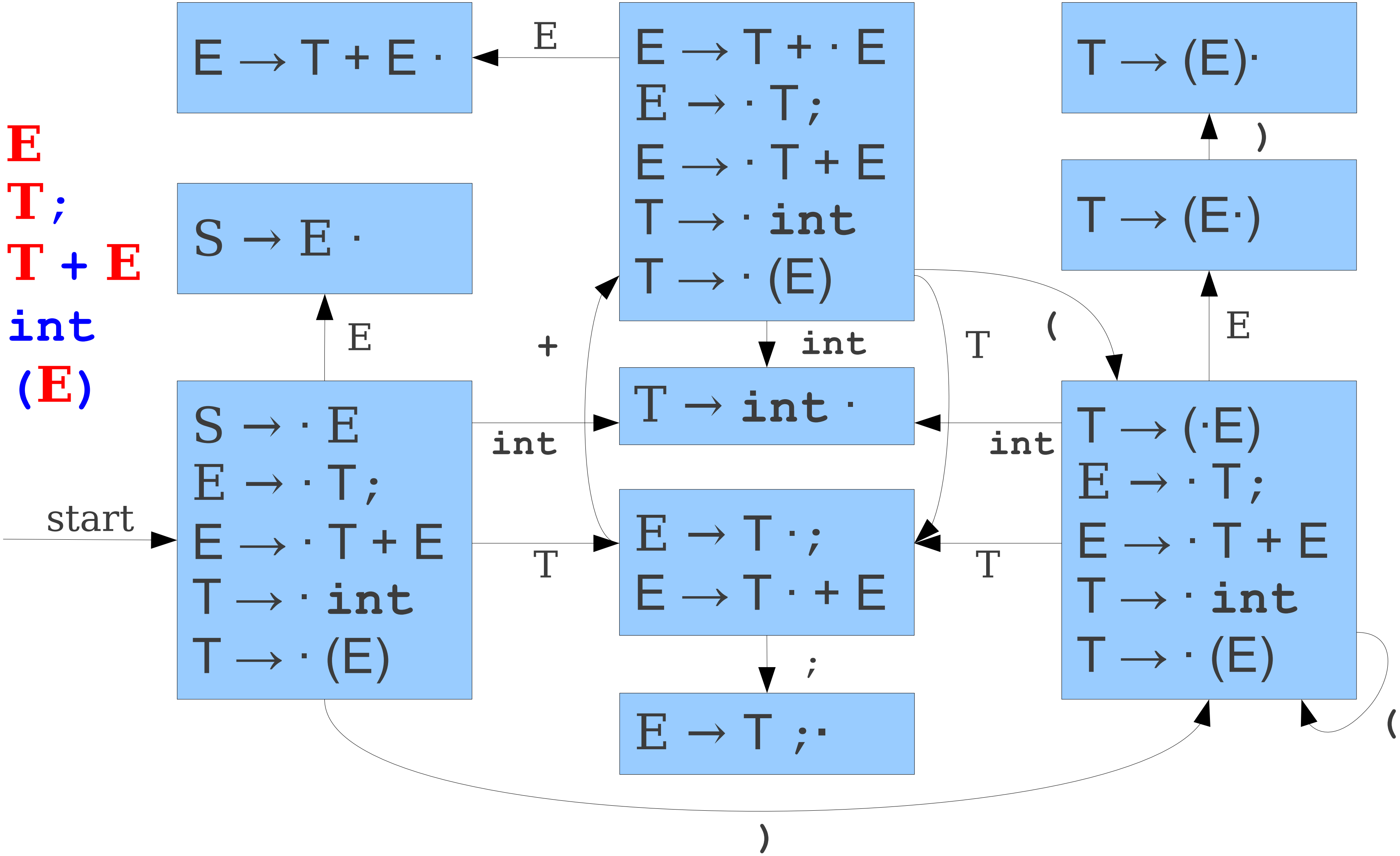
LR(0) Parsing

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)



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) ·

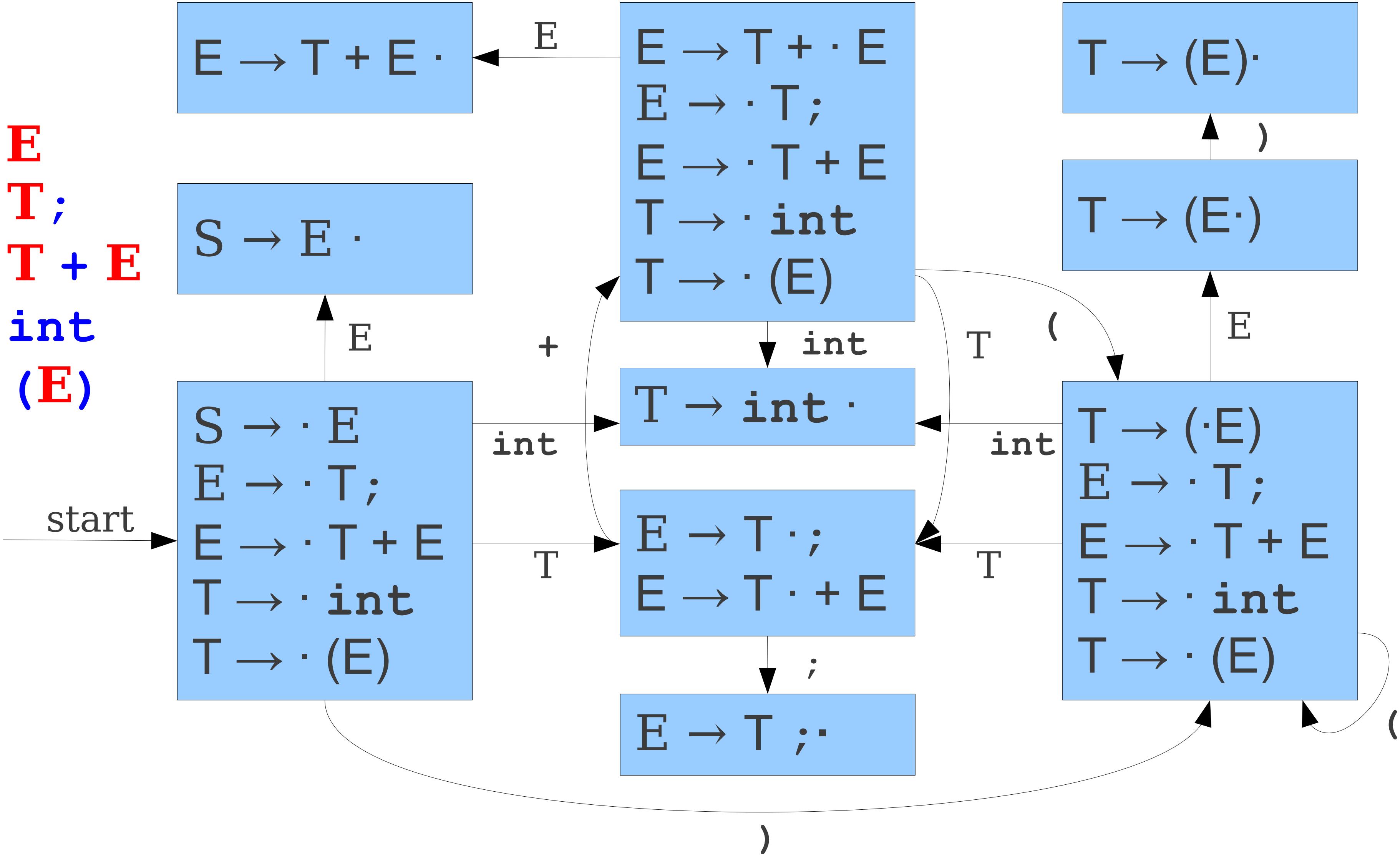
T → (E ·)

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

int	+	(	int	+	int	;	)	;

LR(0) Parsing

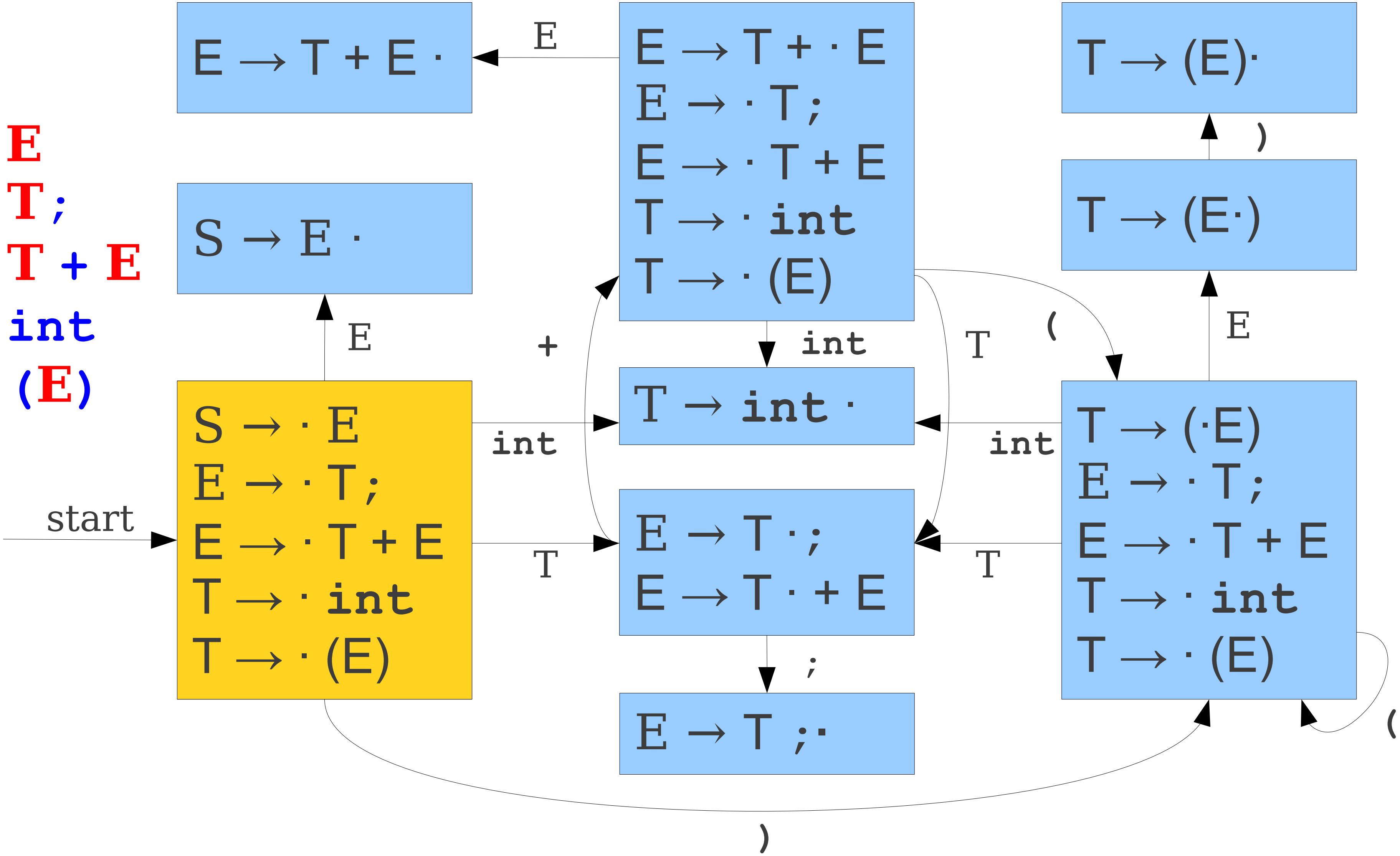
S → **E**
E → **T** `;`
E → **T** + **E**
T → `int`
T → `(`**E**`)`



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

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)·

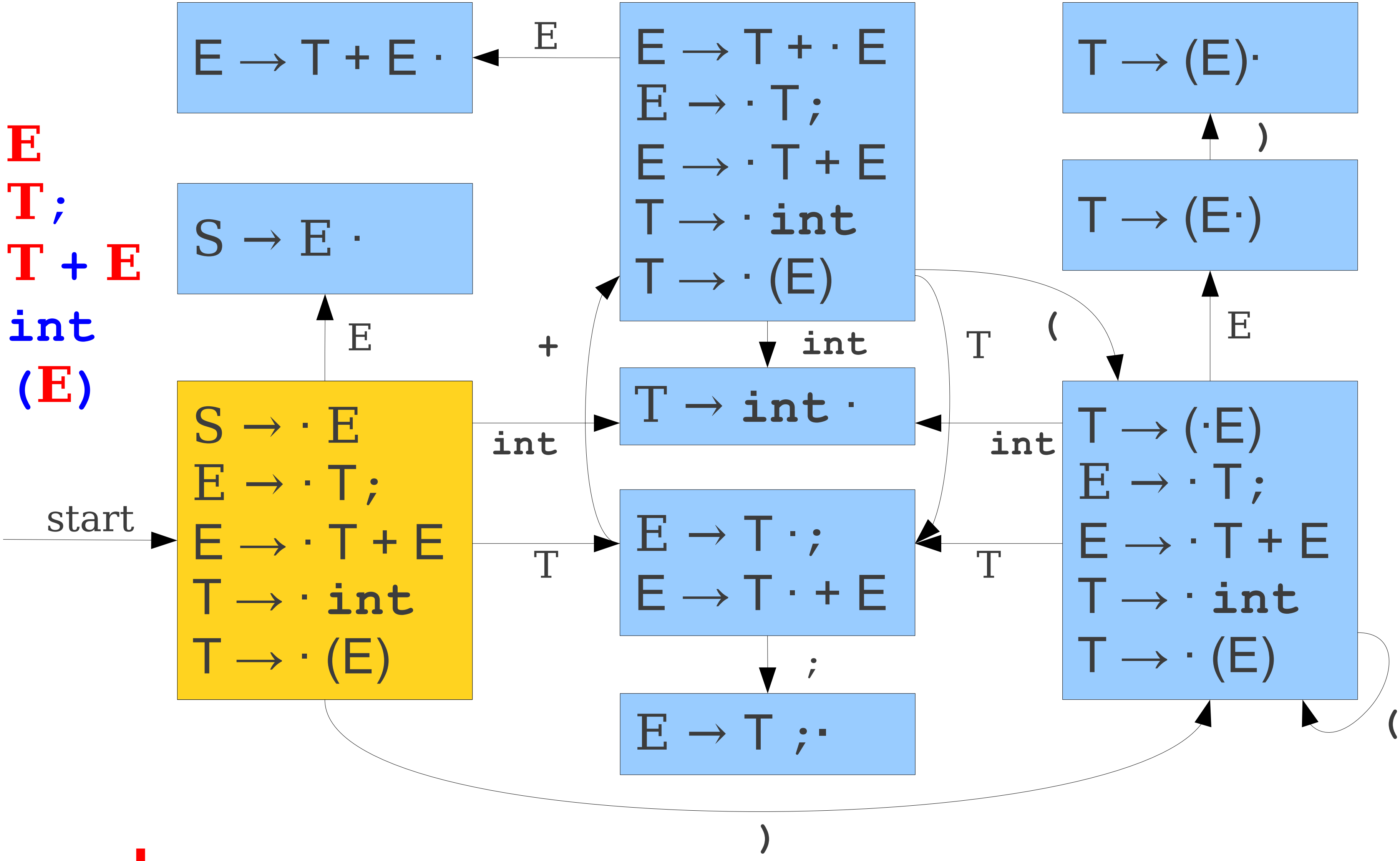
T → (E·)

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)`



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)·

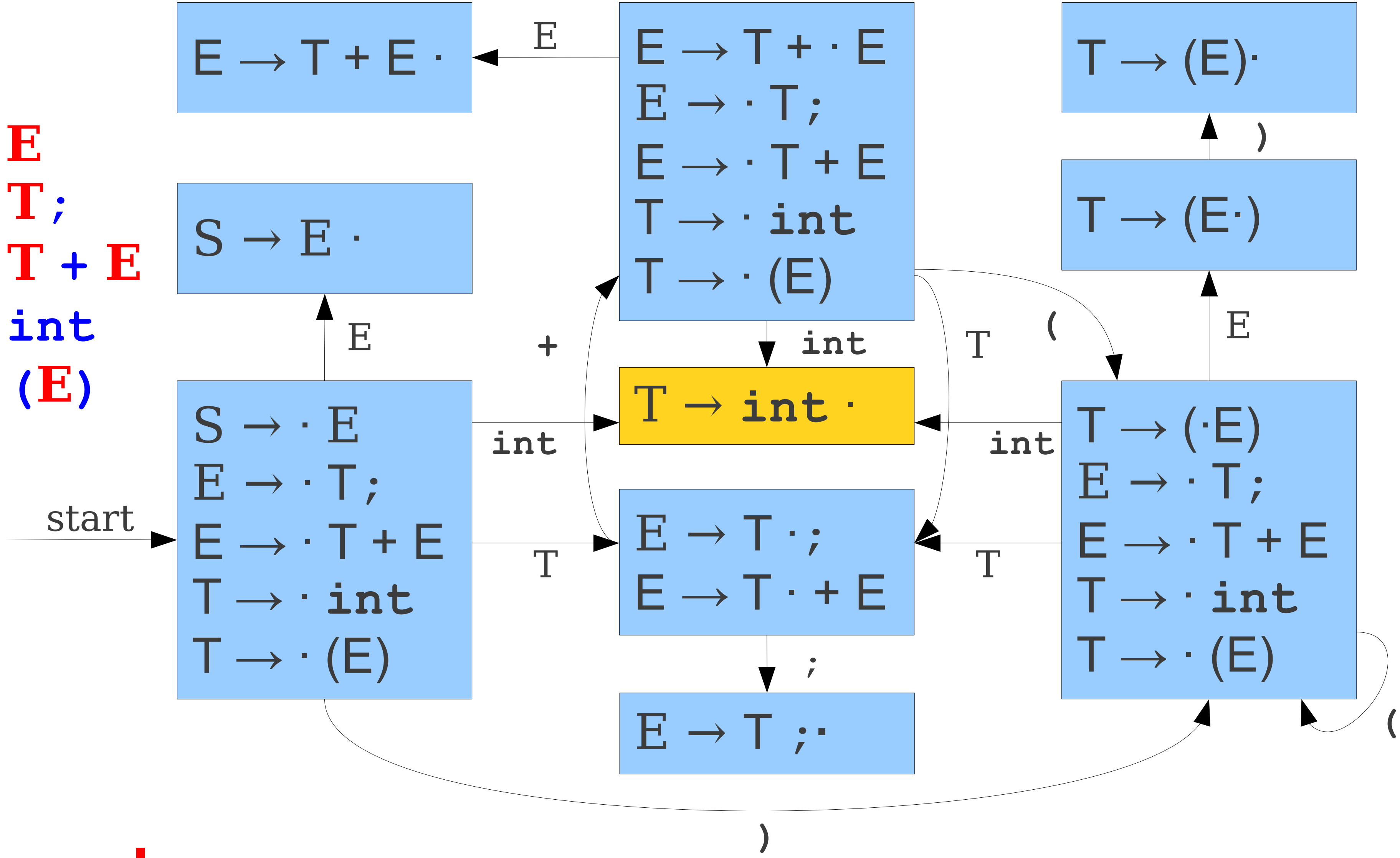
T → (E·)

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

`int`	+	(	`int`	+	`int`	;	)	;

LR(0) Parsing

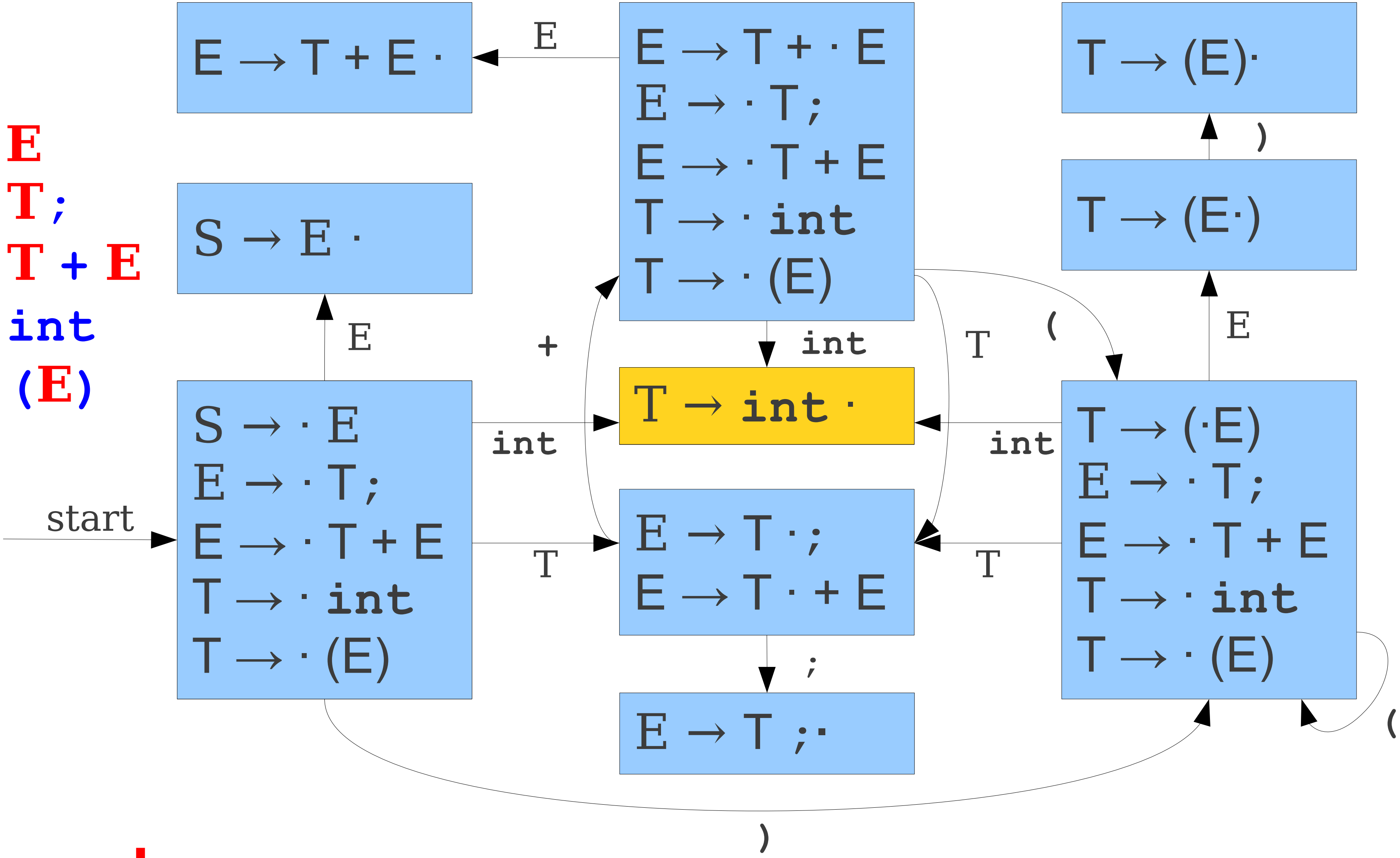
S → E
E → T ;
E → T + E
T → int
T → (E)



int	|	+	(	int	+	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)

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)·

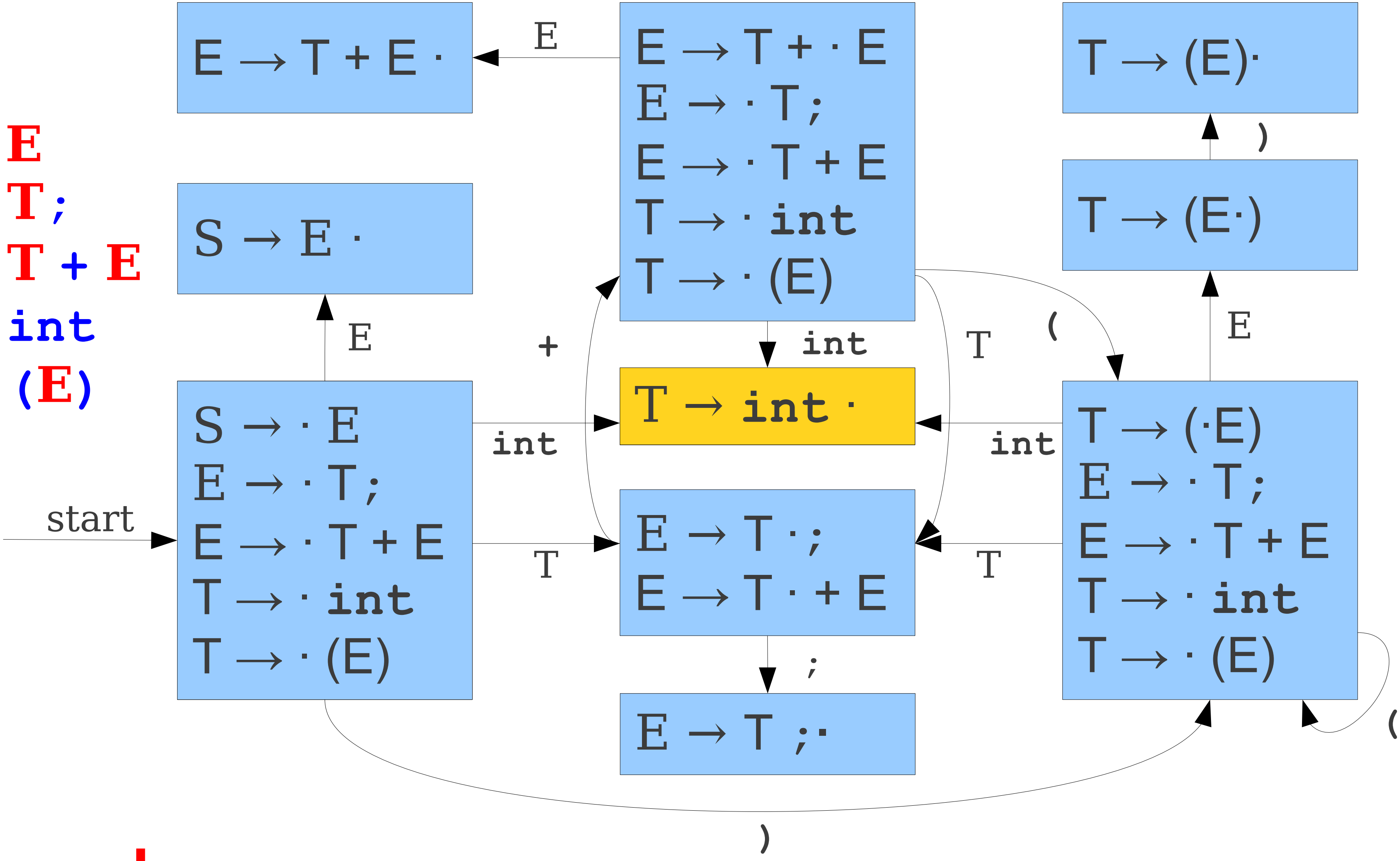
T → (E·)

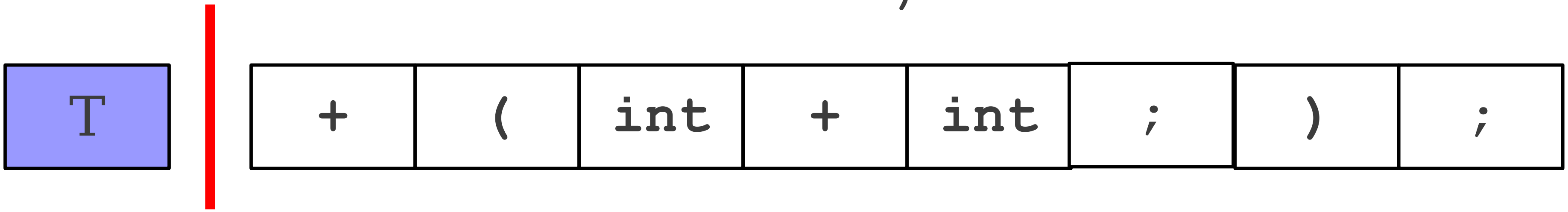
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

+	(	int	+	int	;	)	;

LR(0) Parsing

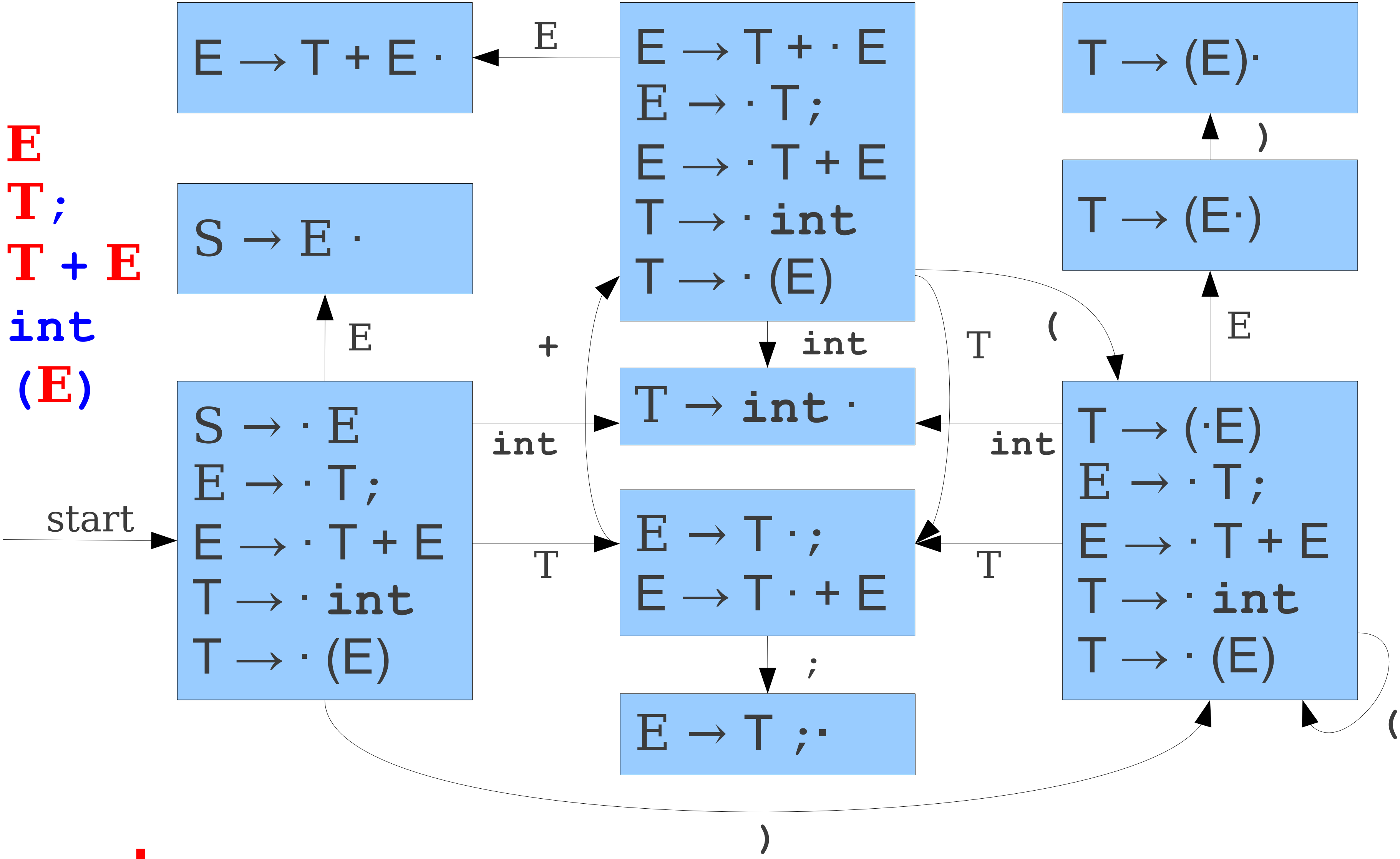
S → **E**
E → **T** **;**
E → **T** **+** **E**
T → **int**
T → **(E)**





LR(0) Parsing

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



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)·

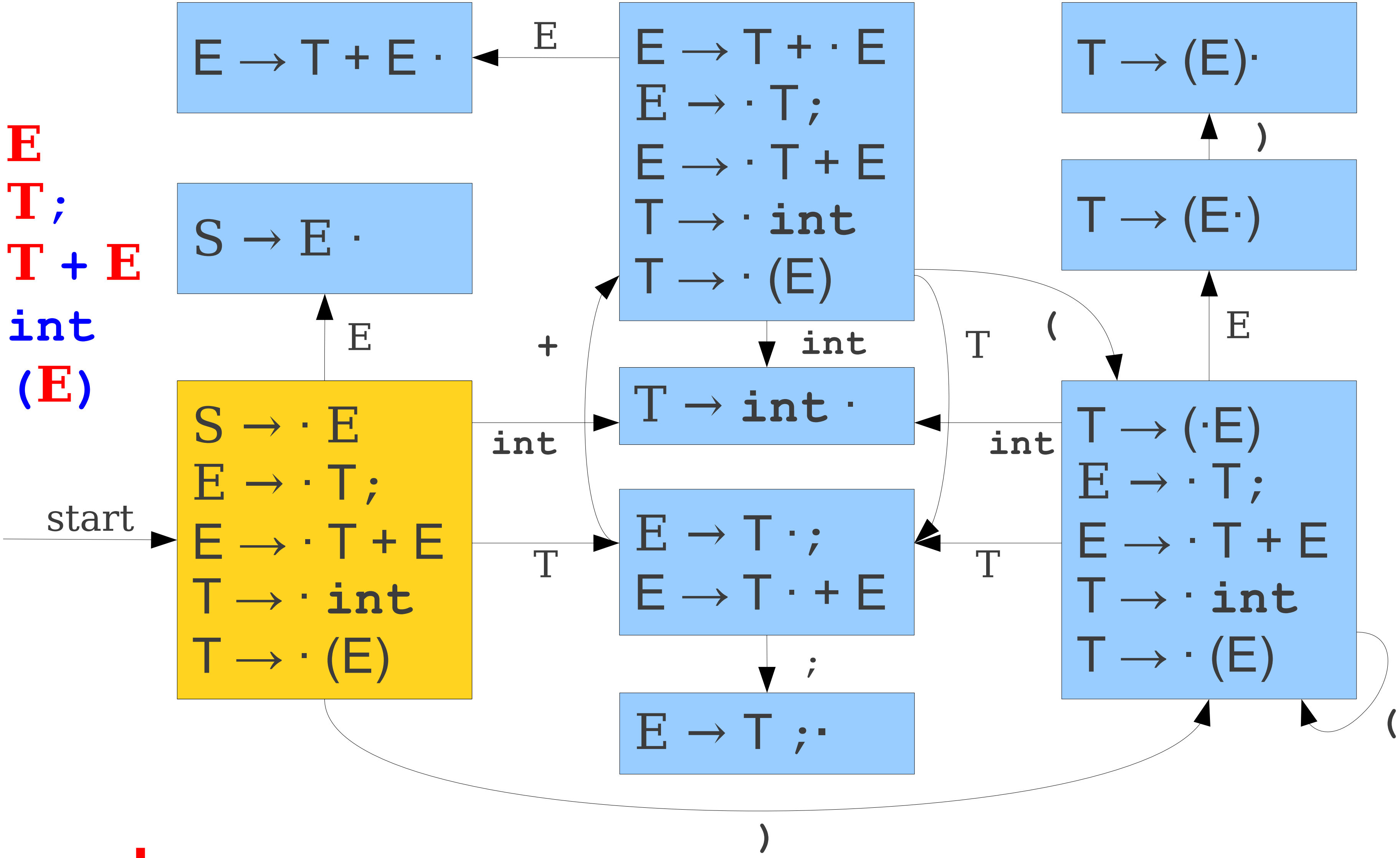
T → (E·)

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

T	+	(	int	+	int	;	)	;

LR(0) Parsing

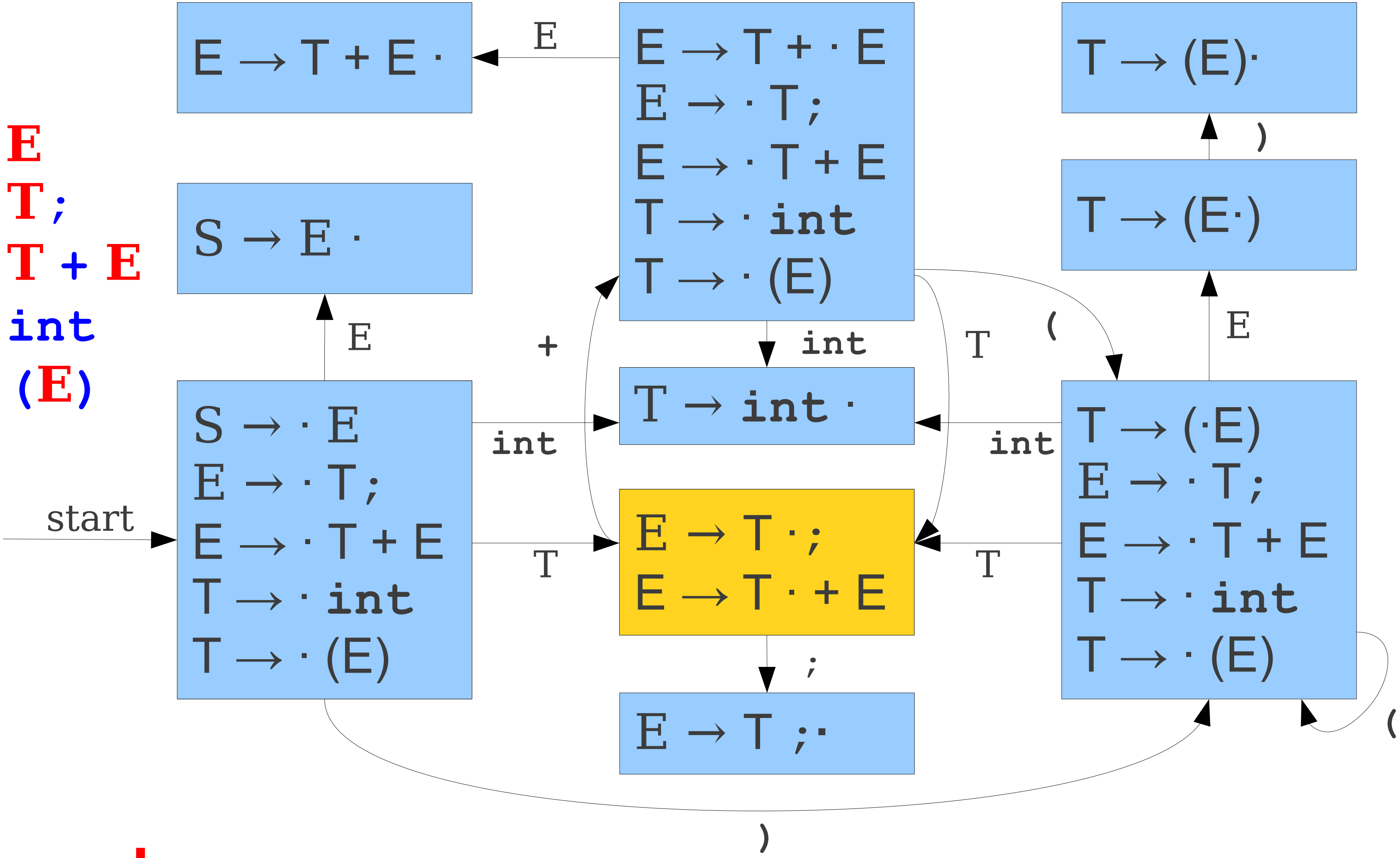
S → **E**
E → **T** `;`
E → **T** `+` **E**
T → `int`
T → `(`**E**`)`



T		+	(	int	+	int	;	)	;

LR(0) Parsing

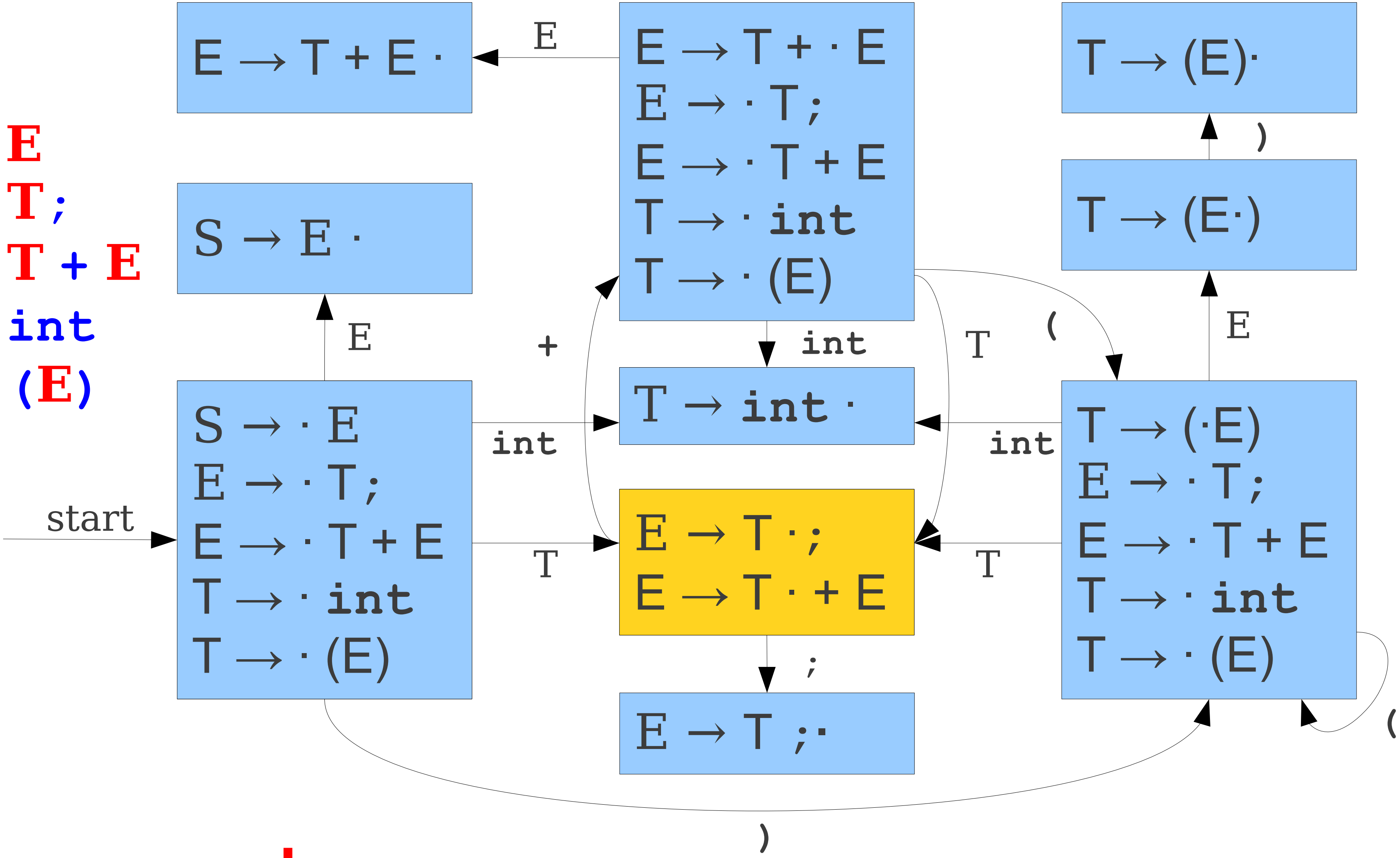
S → **E**
E → **T** `;`
E → **T** `+` **E**
T → `int`
T → `(`**E**`)`



T		+	(	int	+	int	;	)	;

LR(0) Parsing

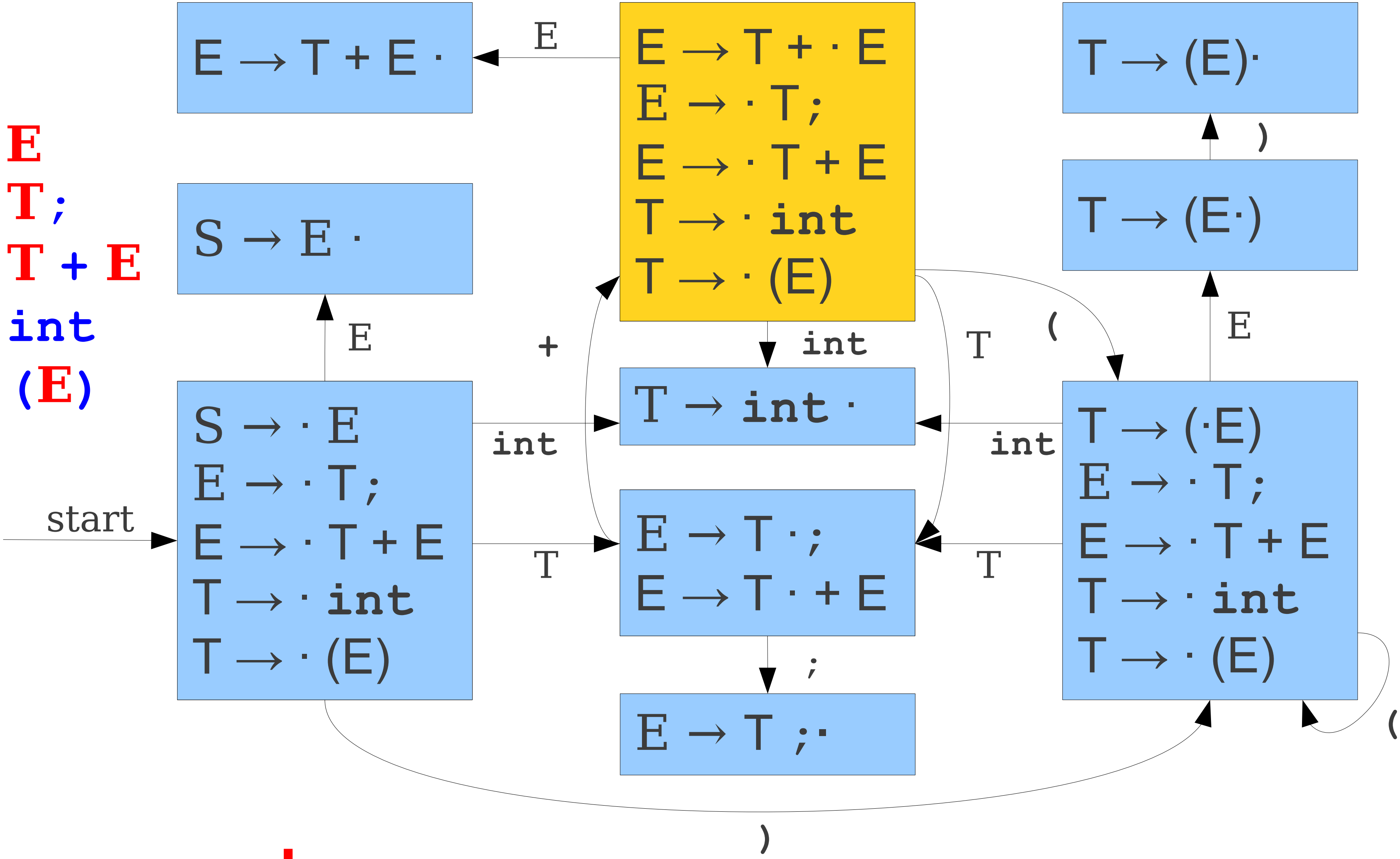
S → **E**
E → **T** `;`
E → **T** + **E**
T → `int`
T → `(`**E**`)`



T	+		(	int	+	int	;	)	;

LR(0) Parsing

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



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

S → E ·

E → T · ;
E → T · + E

E → T ; ·

T → int ·

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

E → T + E ·

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

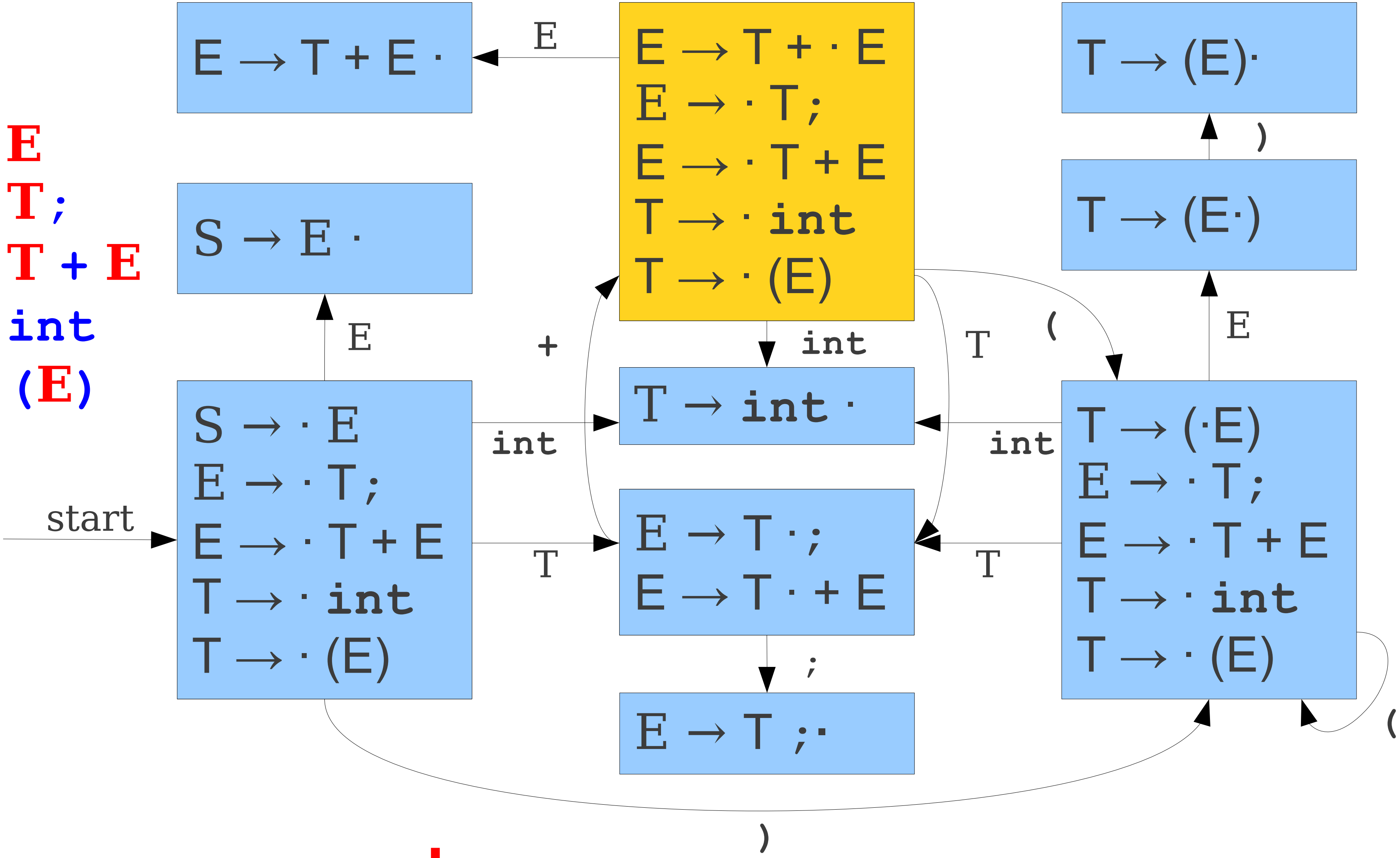
T → (E·)

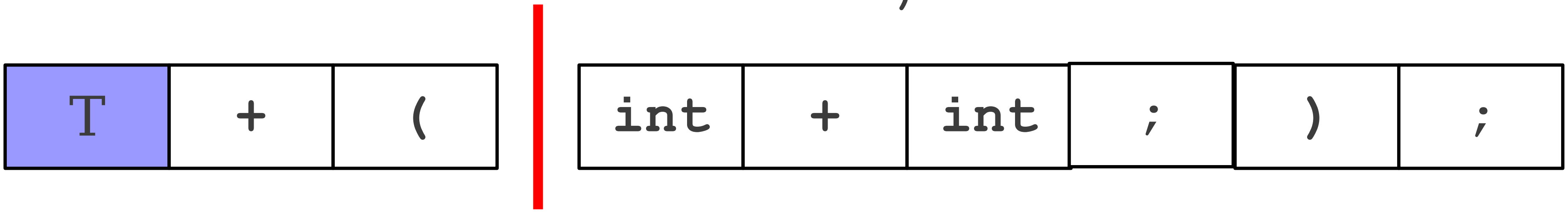
T → (E)·

T	+		(	int	+	int	;	)	;

LR(0) Parsing

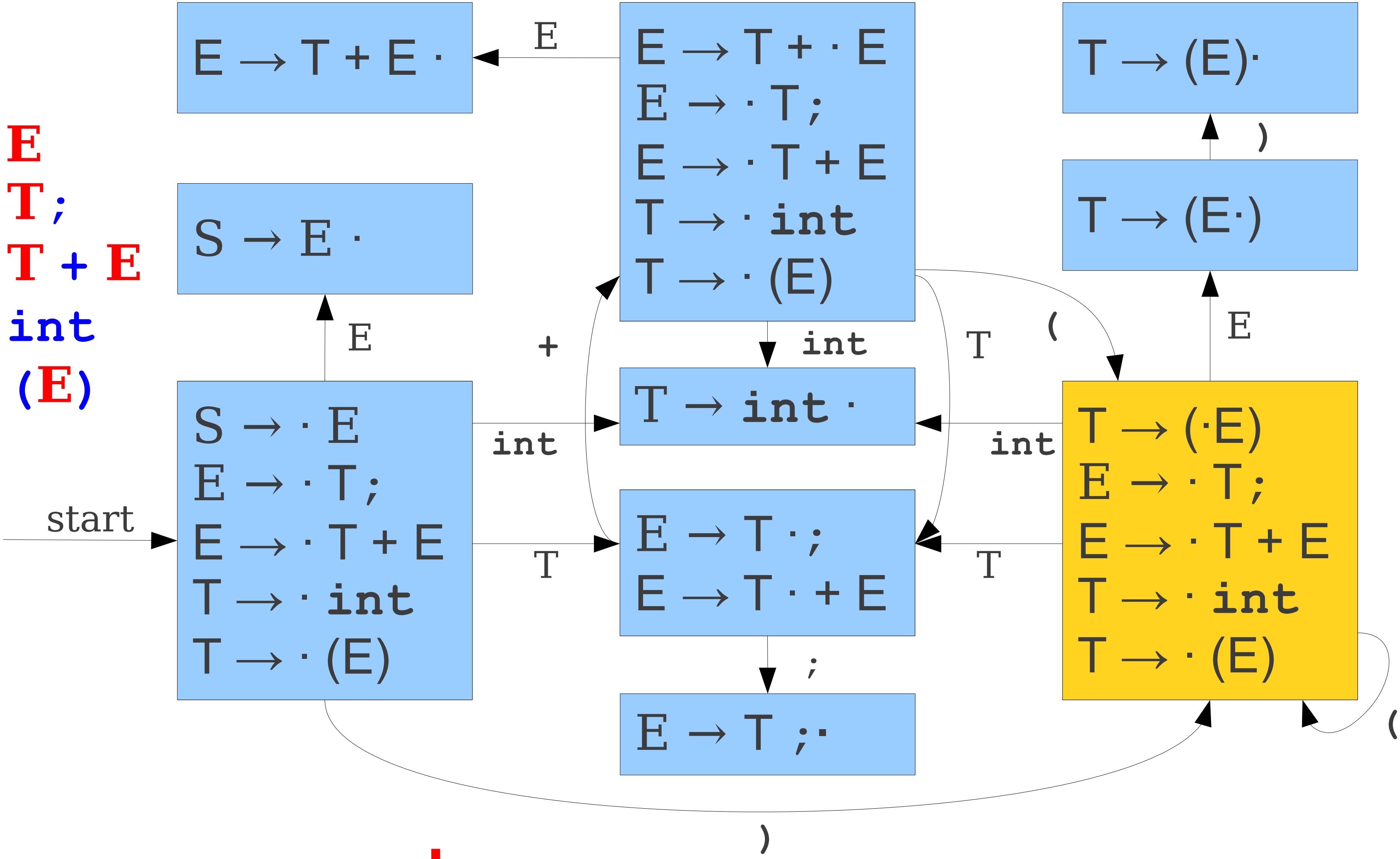
S → **E**
E → **T** `;`
E → **T** `+` **E**
T → `int`
T → `(`**E**`)`





LR(0) Parsing

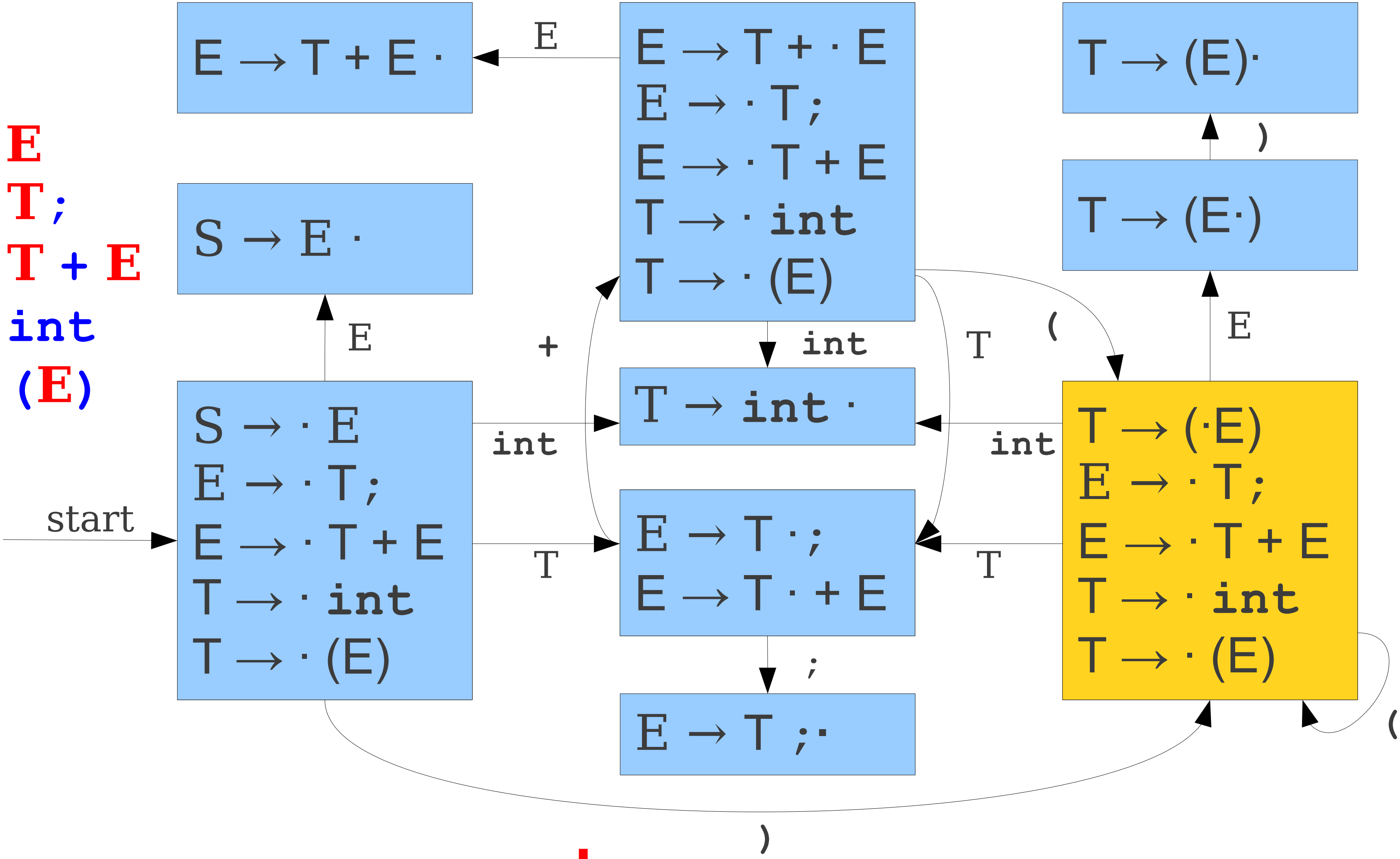
S → **E**
E → **T** `;`
E → **T** `+` **E**
T → `int`
T → `(`**E**`)`



T	+	(	int	+	int	;	)	;

LR(0) Parsing

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



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 ·

E → T ·;
E → T · + E

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

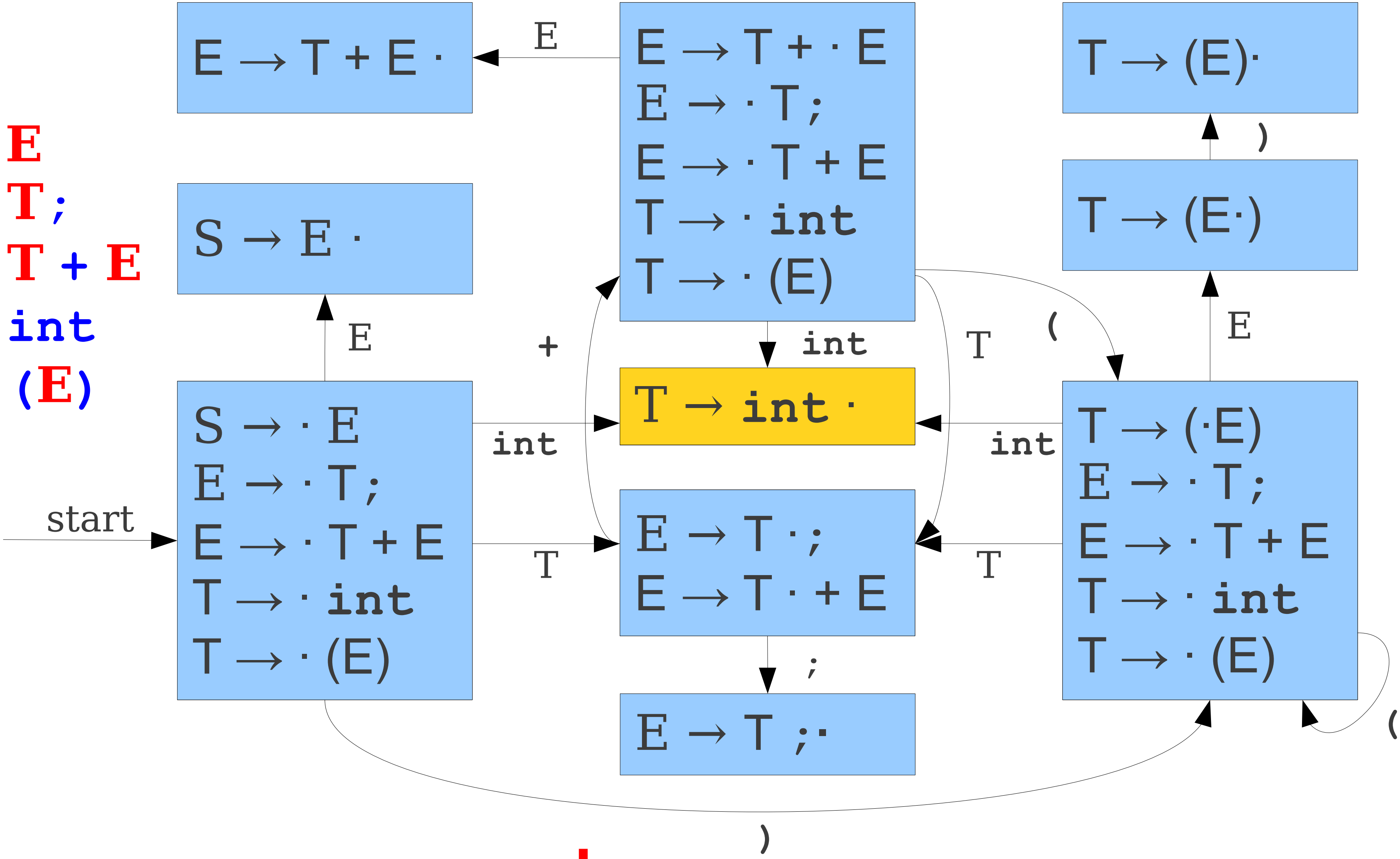
E → T ;·

T	+	(	int

+	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)

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)·

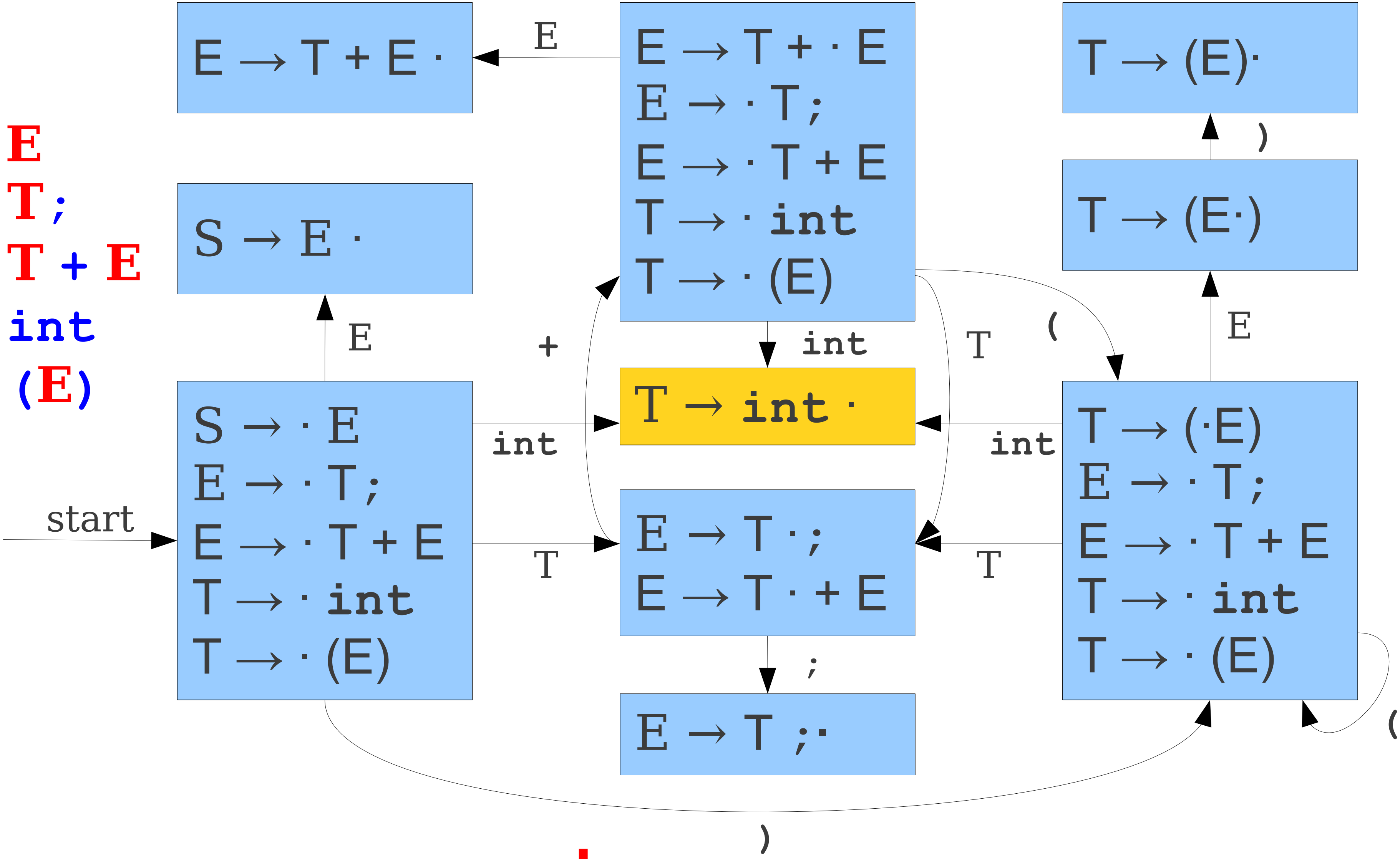
T → (E·)

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

T	+	(	`int`	+	`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)

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·)

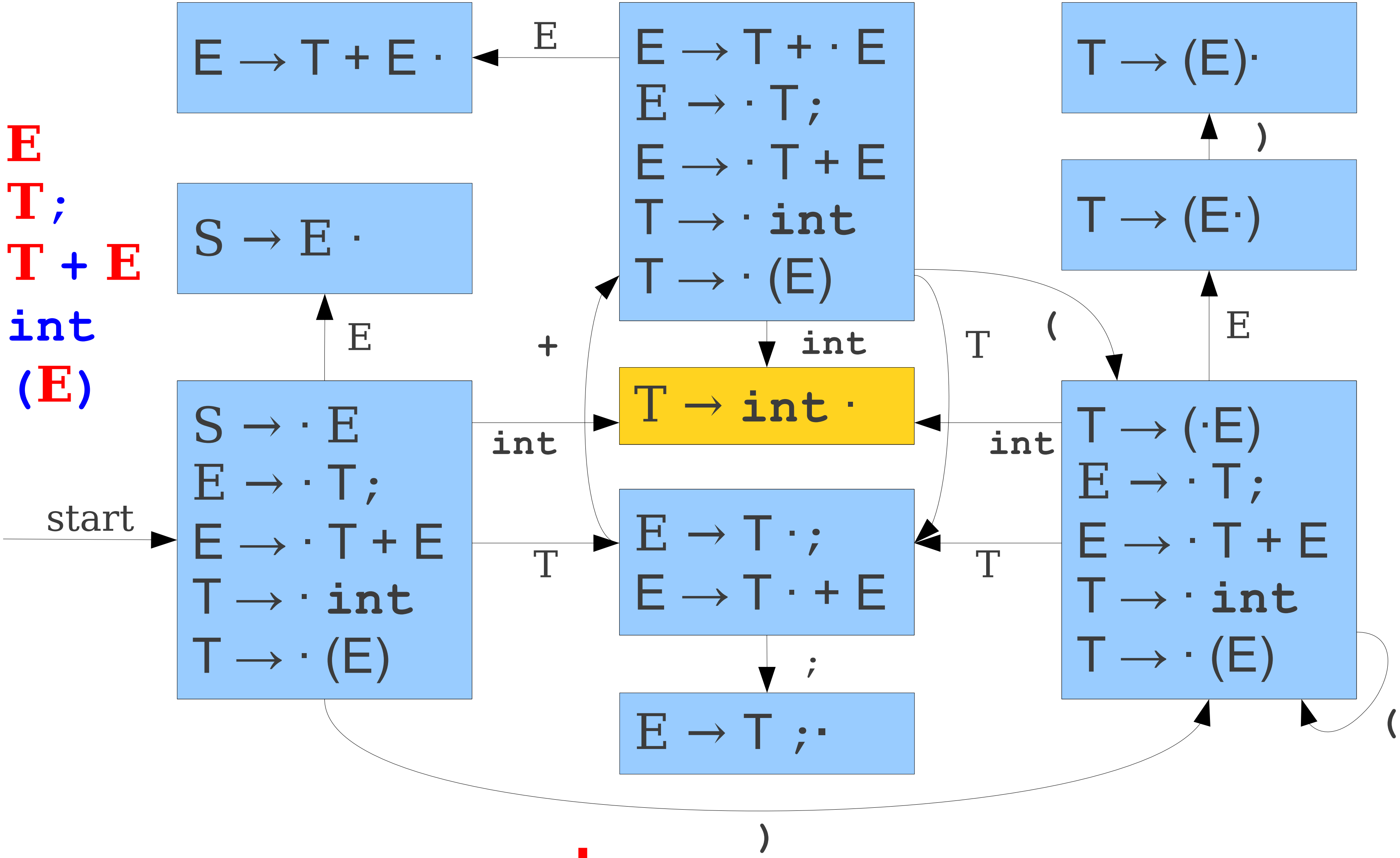
T → (E)·

T	+	(

+	`int`	;	)	;

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·)

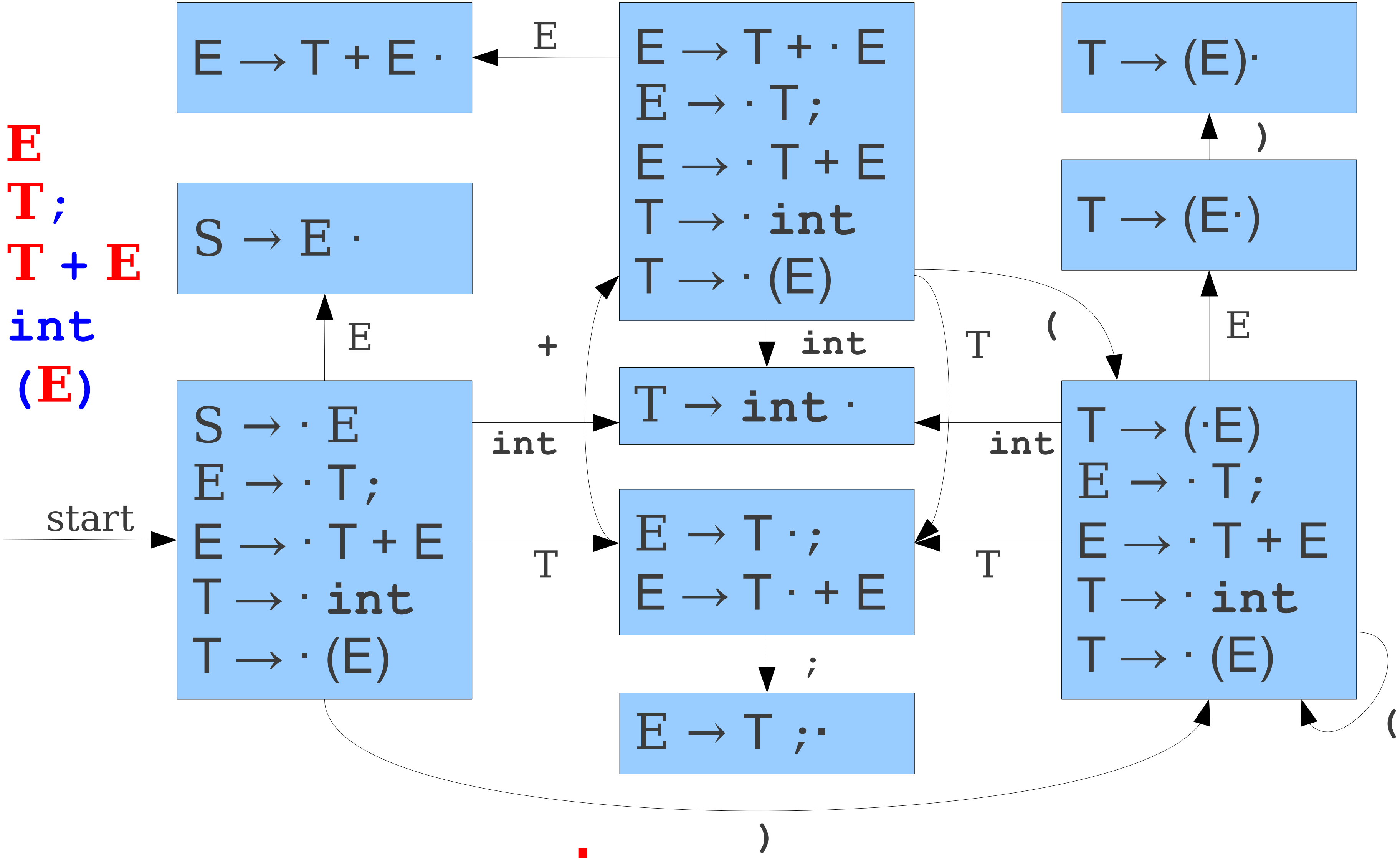
T → (·E)
E → · T;
E → · T + E
T → · `int`
T → · (E)

T	+	(	T

+	`int`	;	)	;

LR(0) Parsing

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



E → T + E ·

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)

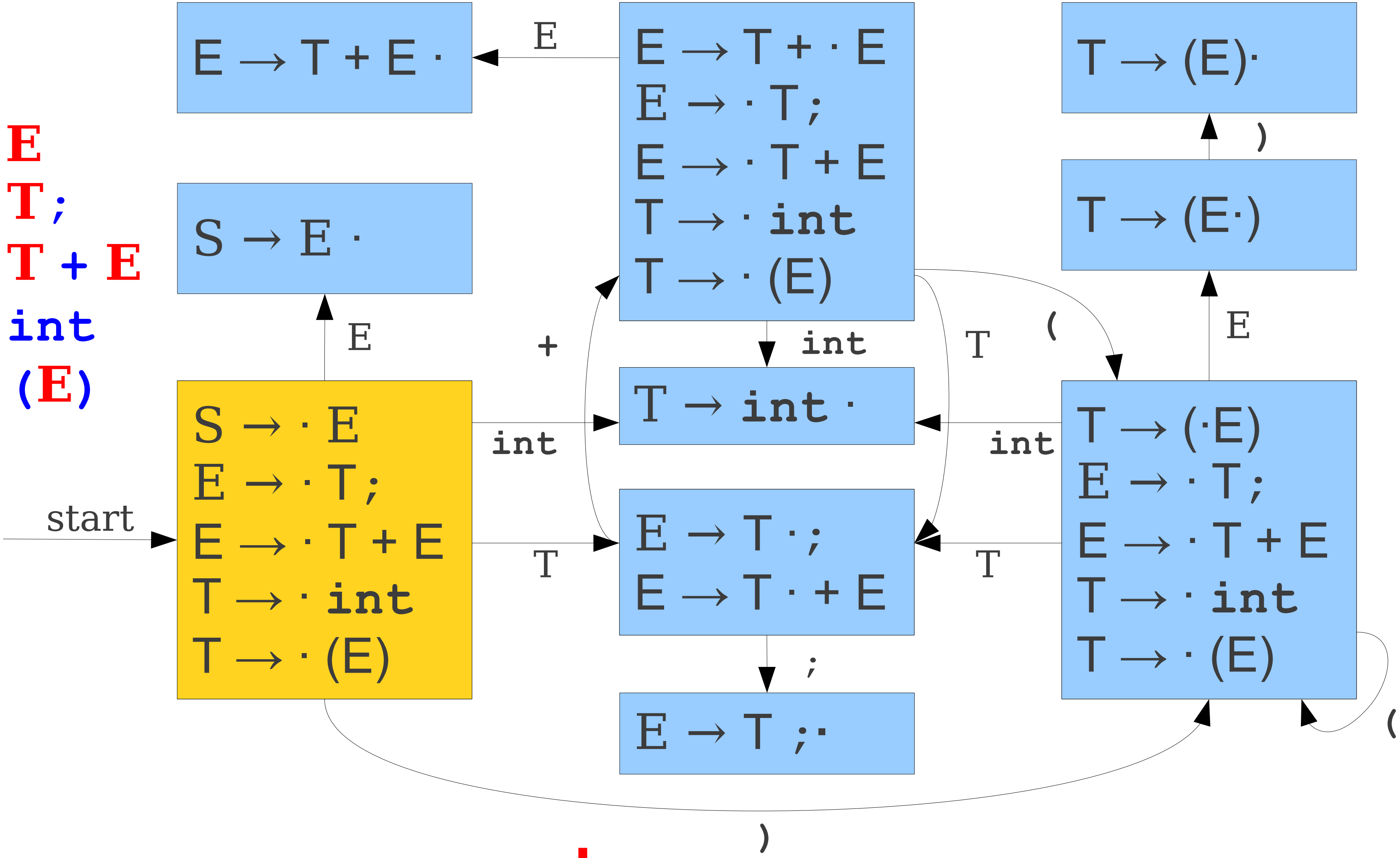
S → · E
E → · T;
E → · T + E
T → · int
T → · (E)

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
T → · **int**
T → · (E)

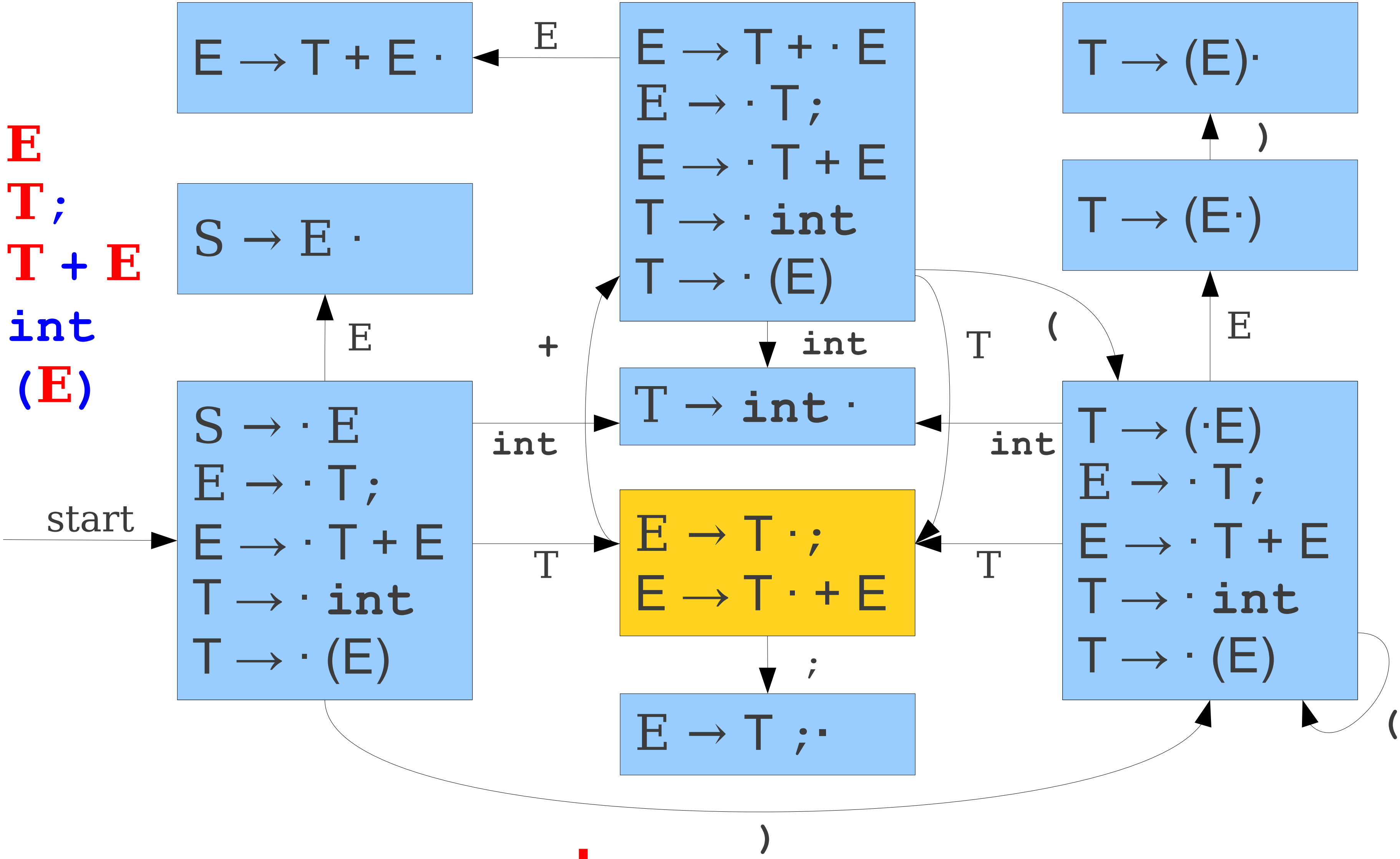
E → T ·;
E → T · + E

E → T ;·

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

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

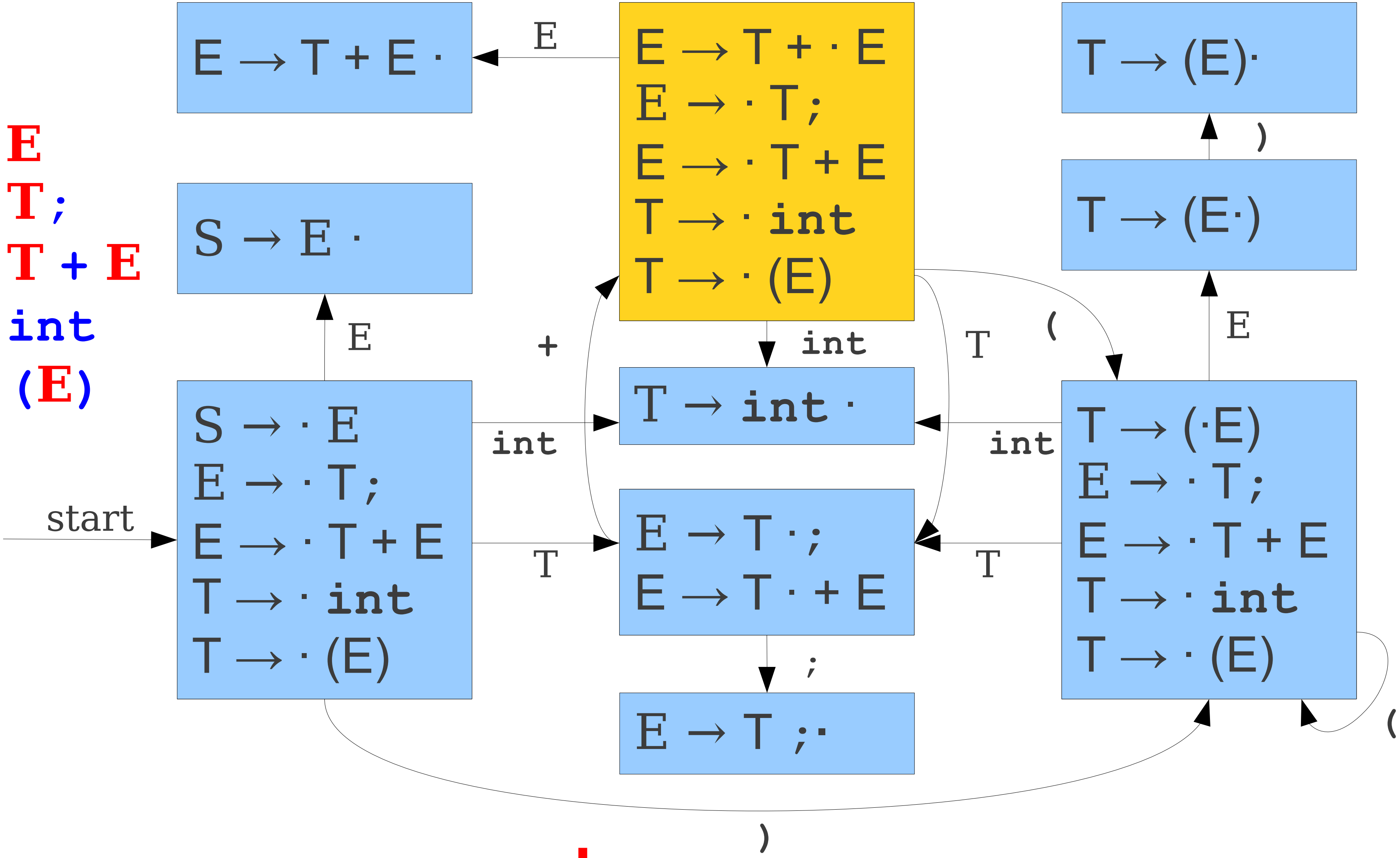
E → T ;·

T	+	(	T

+	int	;	)	;

LR(0) Parsing

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



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)

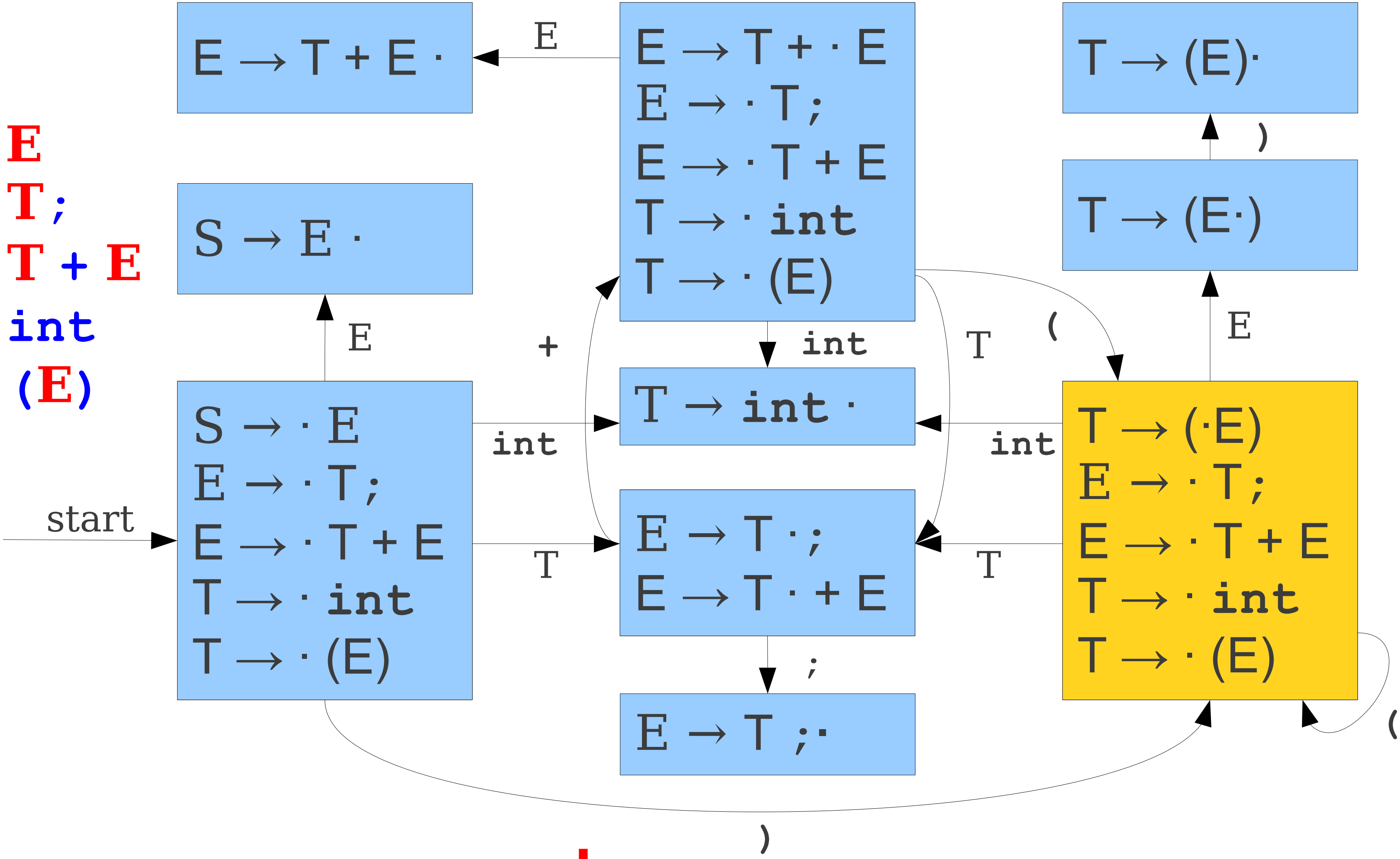
T → (E·)

T → (E)·

T	+	(	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)

S → E ·

E → T + E ·

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

T → `int` ·

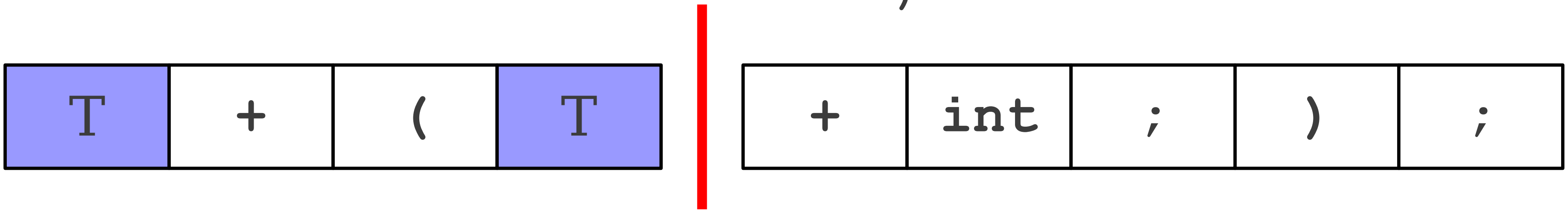
E → T · ;
E → T · + E

E → T ; ·

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

T → (E·)

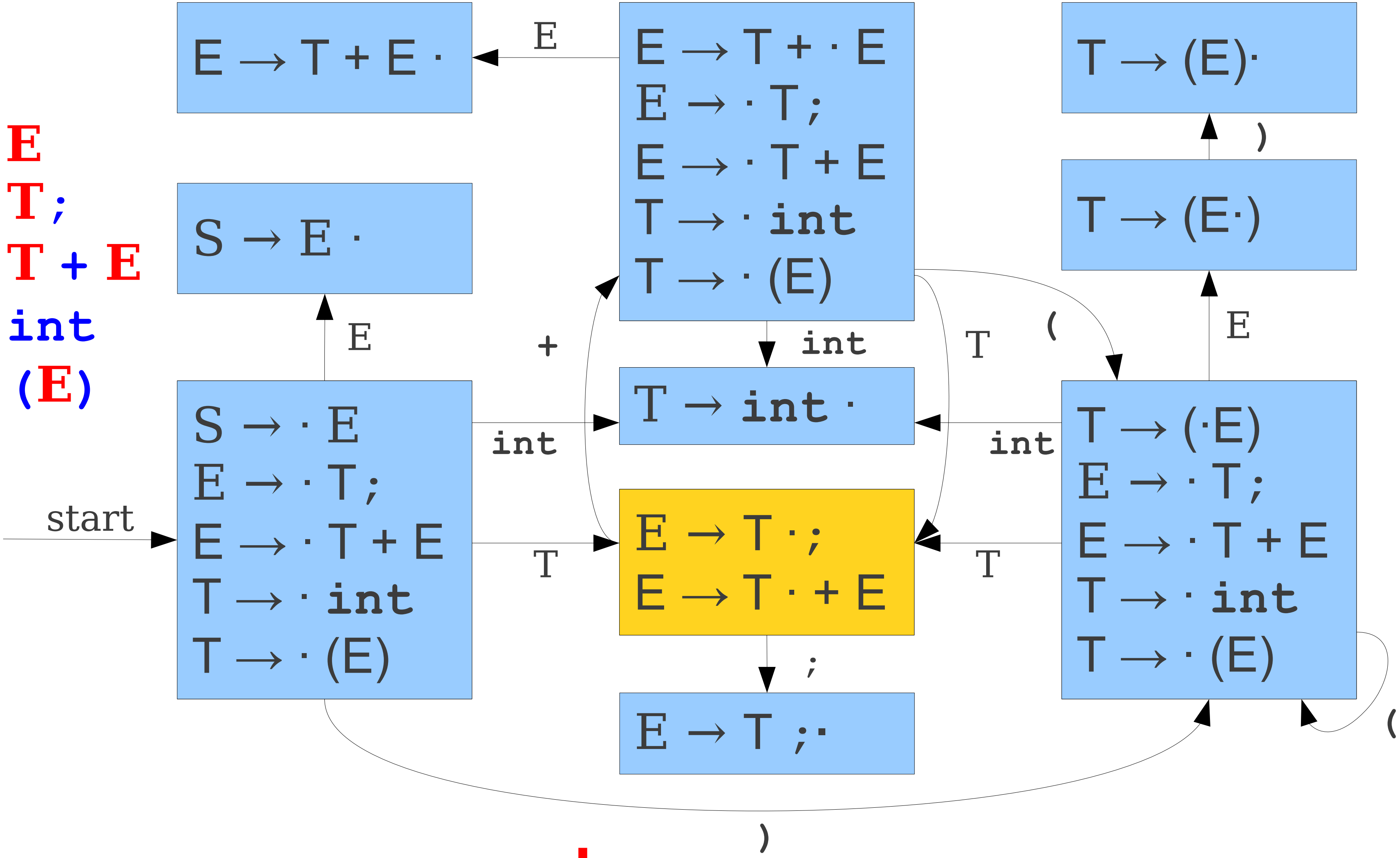
T → (E)·



T	+	(	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)

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)·

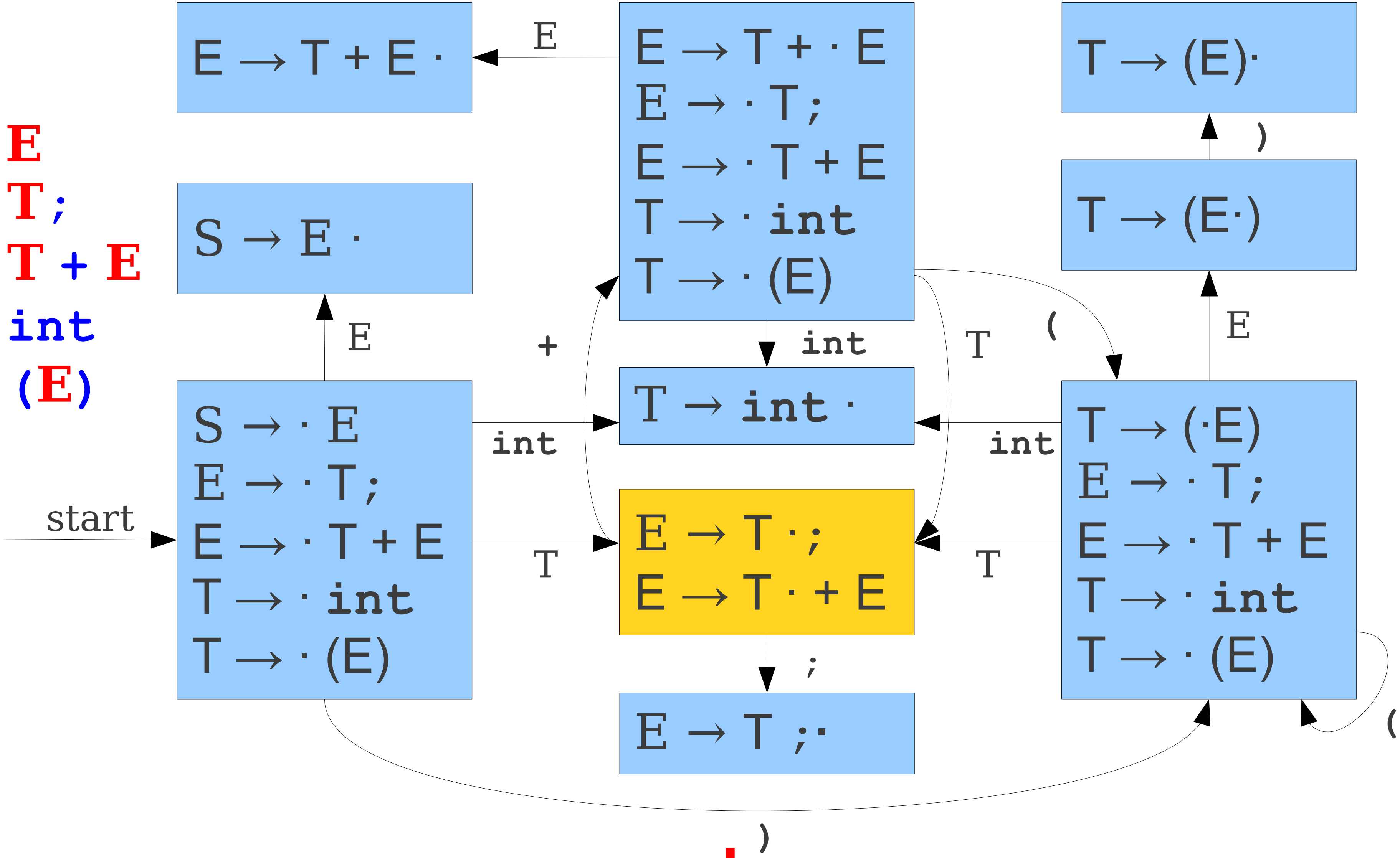
T → (E·)

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

T	+	(	T	|	+	`int`	;	)	;

LR(0) Parsing

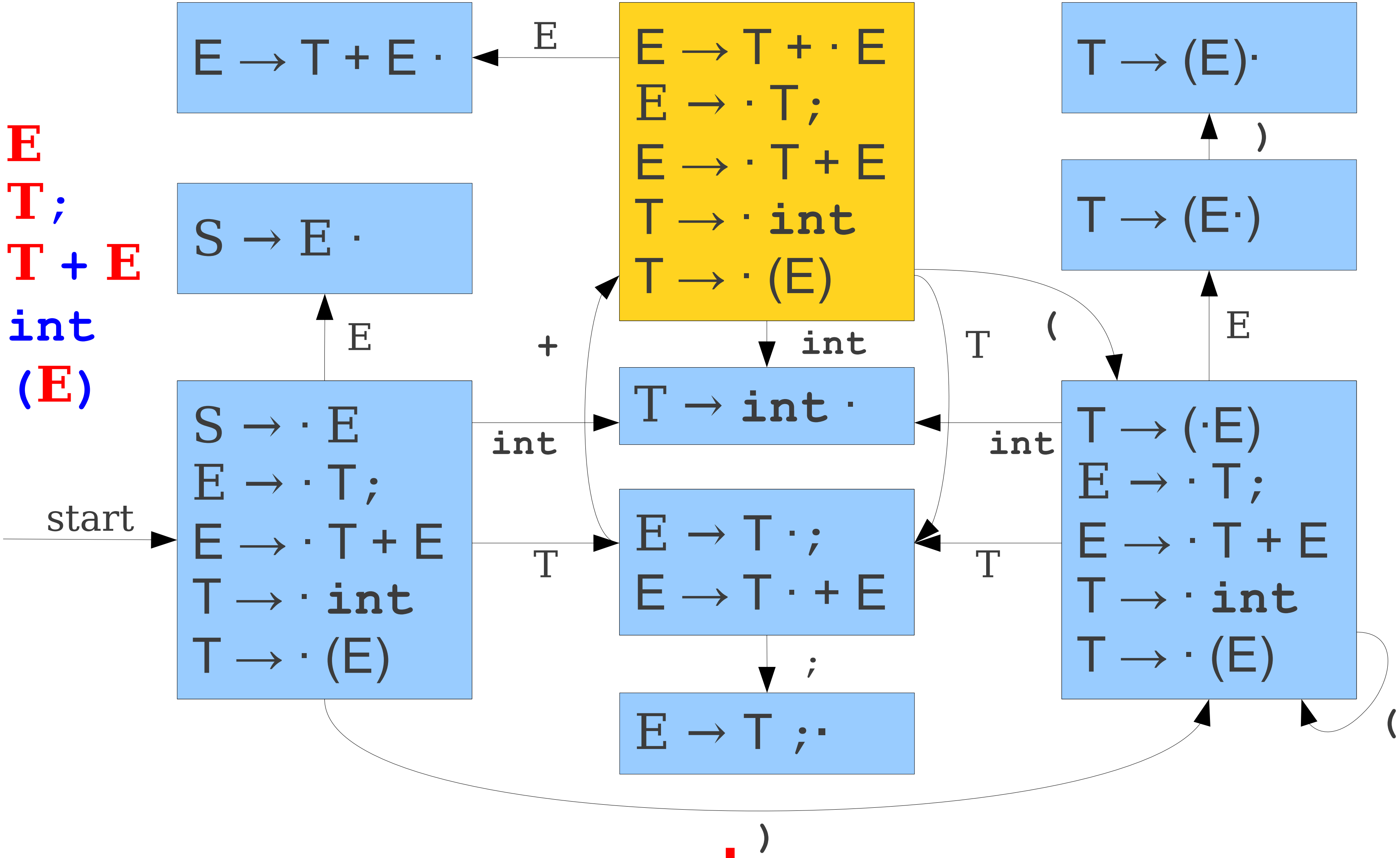
S → **E**
E → **T** ;
E → **T** + **E**
T → int
T → (**E**)



T	+	(	T	+	int	;	)	;

LR(0) Parsing

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

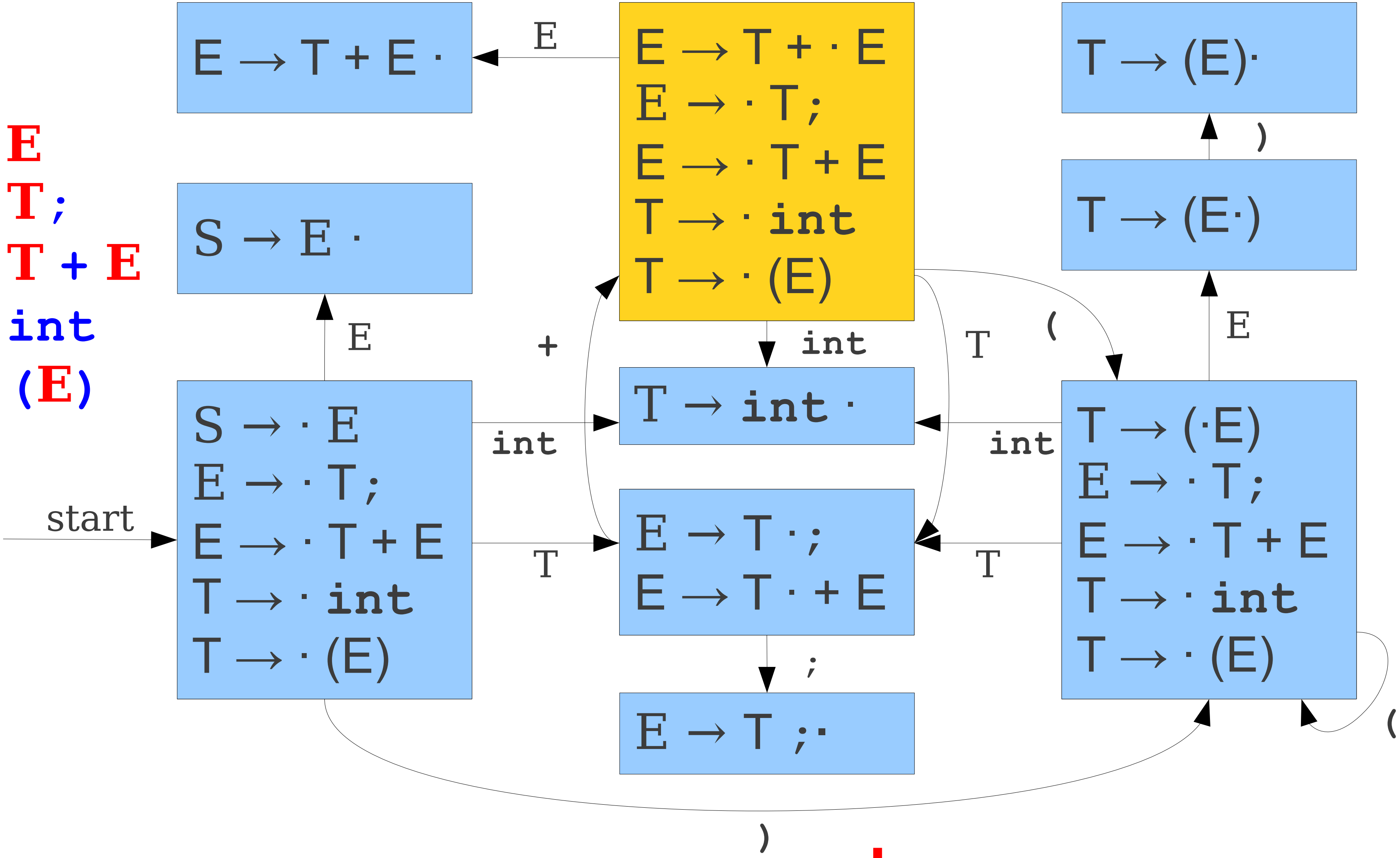


T	+	(	T	+

`int`	;	)	;

LR(0) Parsing

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



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)·

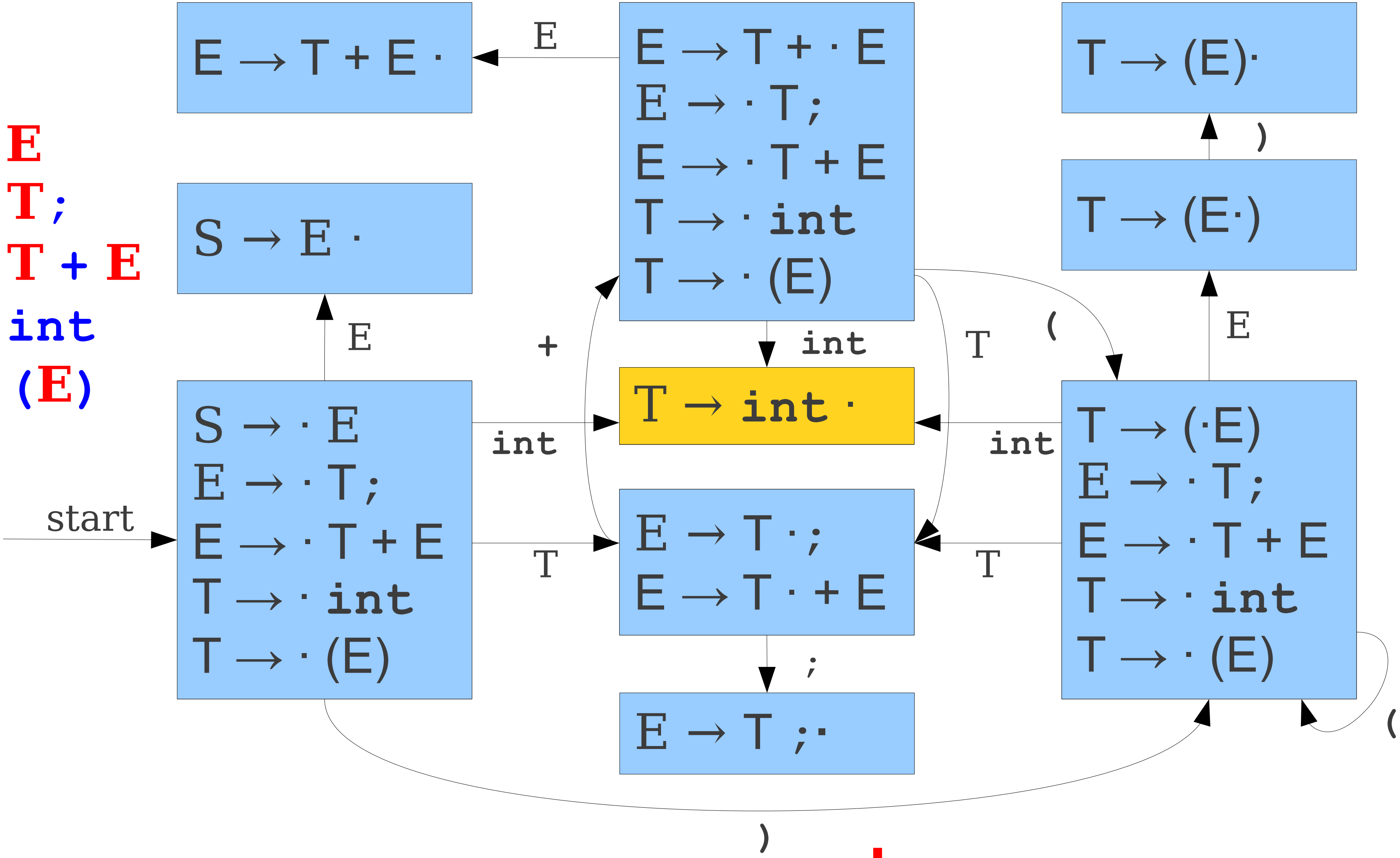
T → (E·)

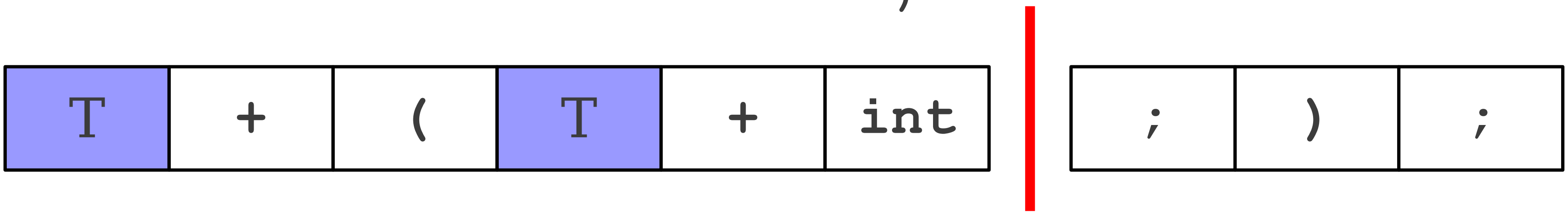
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

T	+	(	T	+	int	;	)	;

LR(0) Parsing

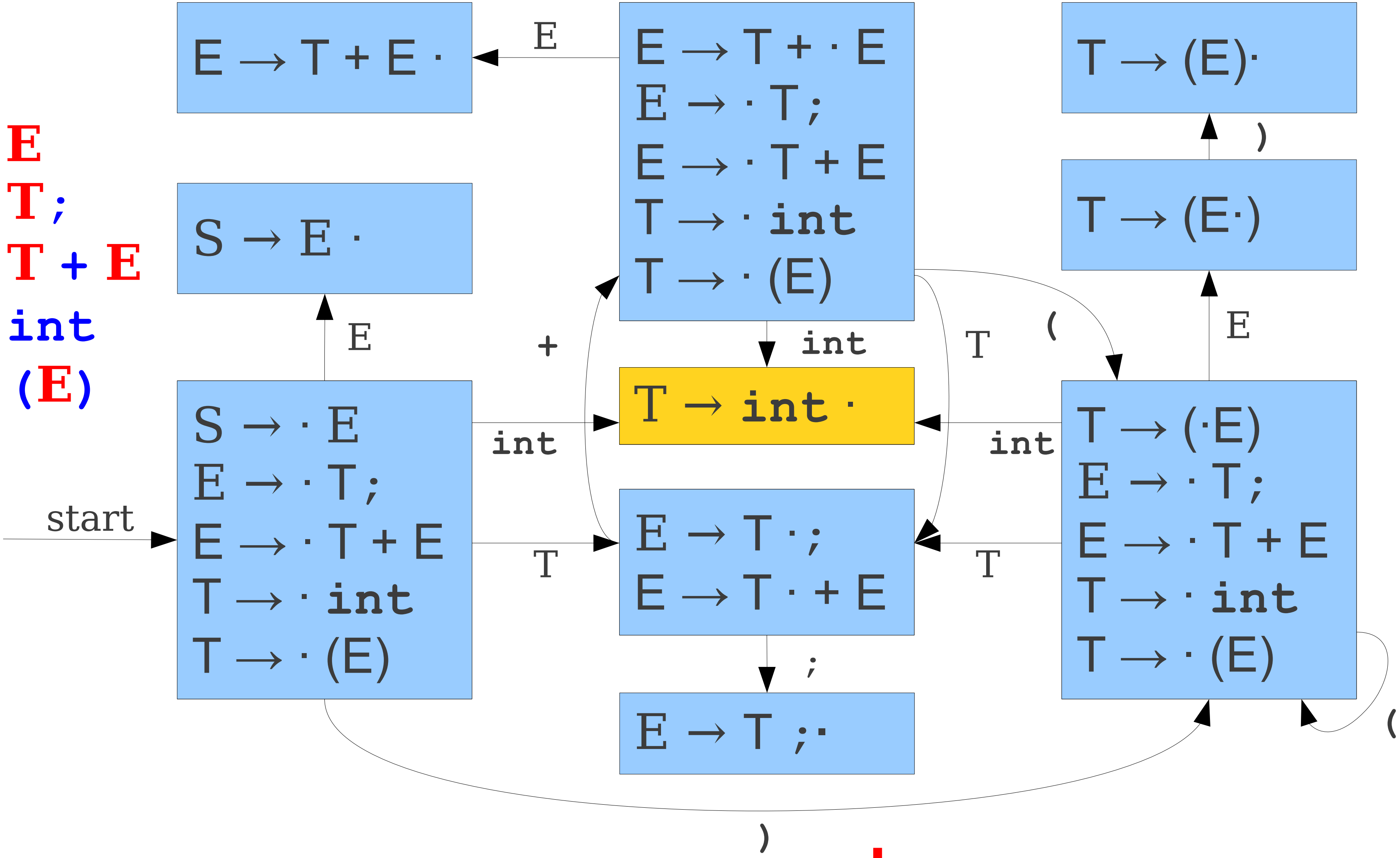
S → **E**
E → **T** `;`
E → **T** `+` **E**
T → `int`
T → `(`**E**`)`





LR(0) Parsing

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



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

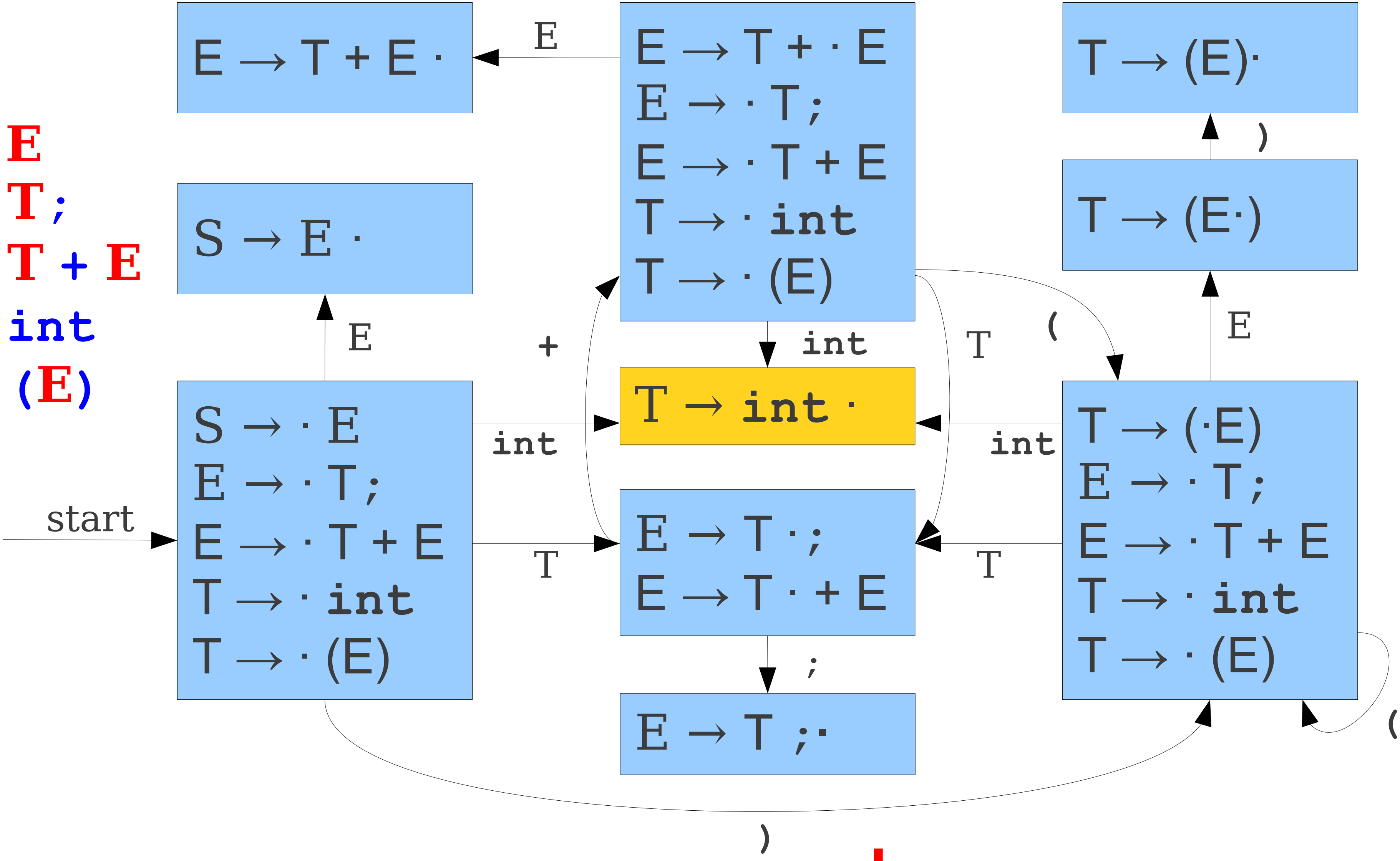
T → (·E)
E → · T;
E → · T + E
T → · `int`
T → · (E)

E → T ;·

T	+	(	T	+	|	;	)	;

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)

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)·

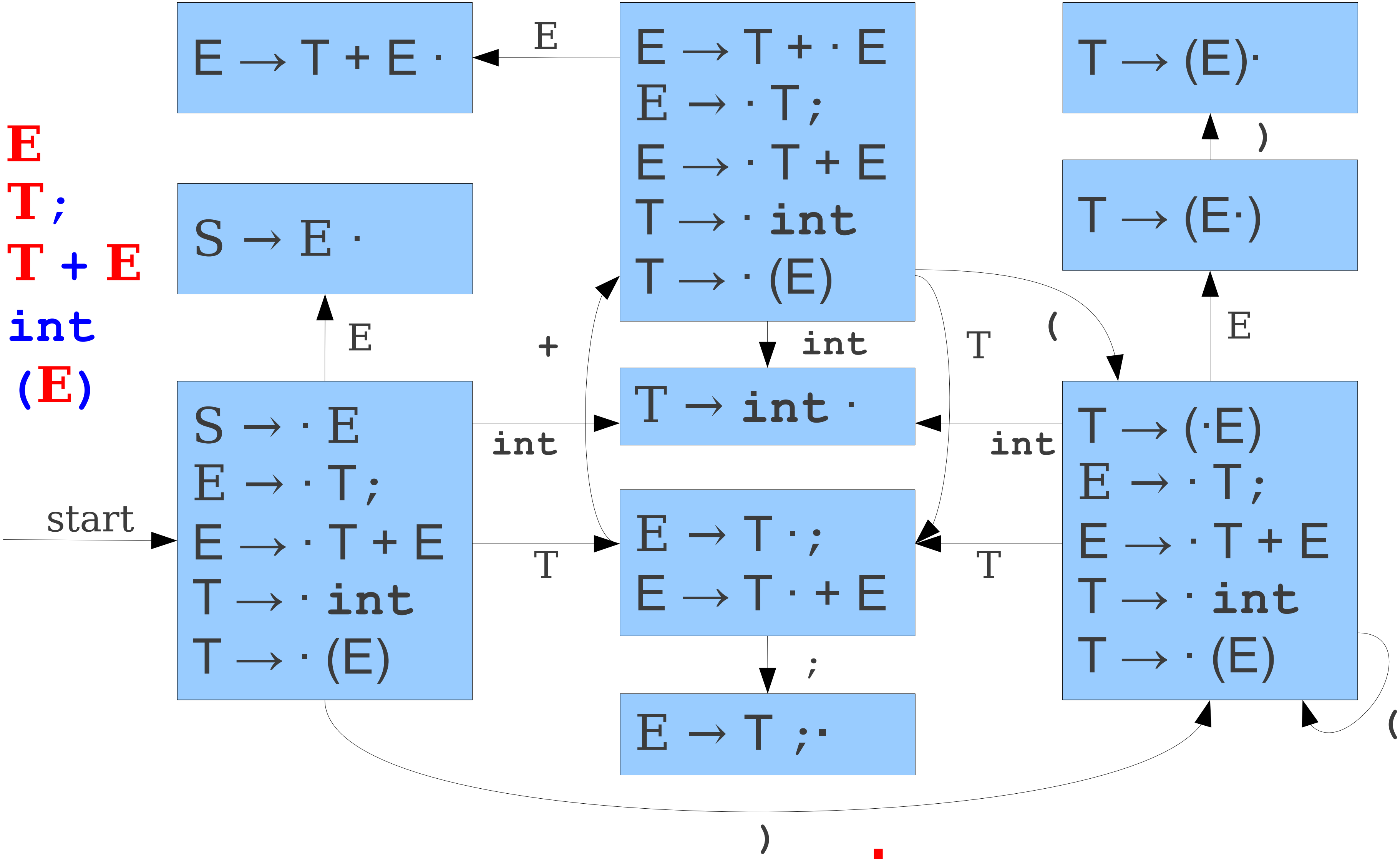
T → (E·)

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

T	+	(	T	+	T	|	;	)	;

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)

S → E ·

E → T · ;
E → T · + E

E → T ; ·

T → `int` ·

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

E → T + E ·

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

T → (E·)

T → (E)·

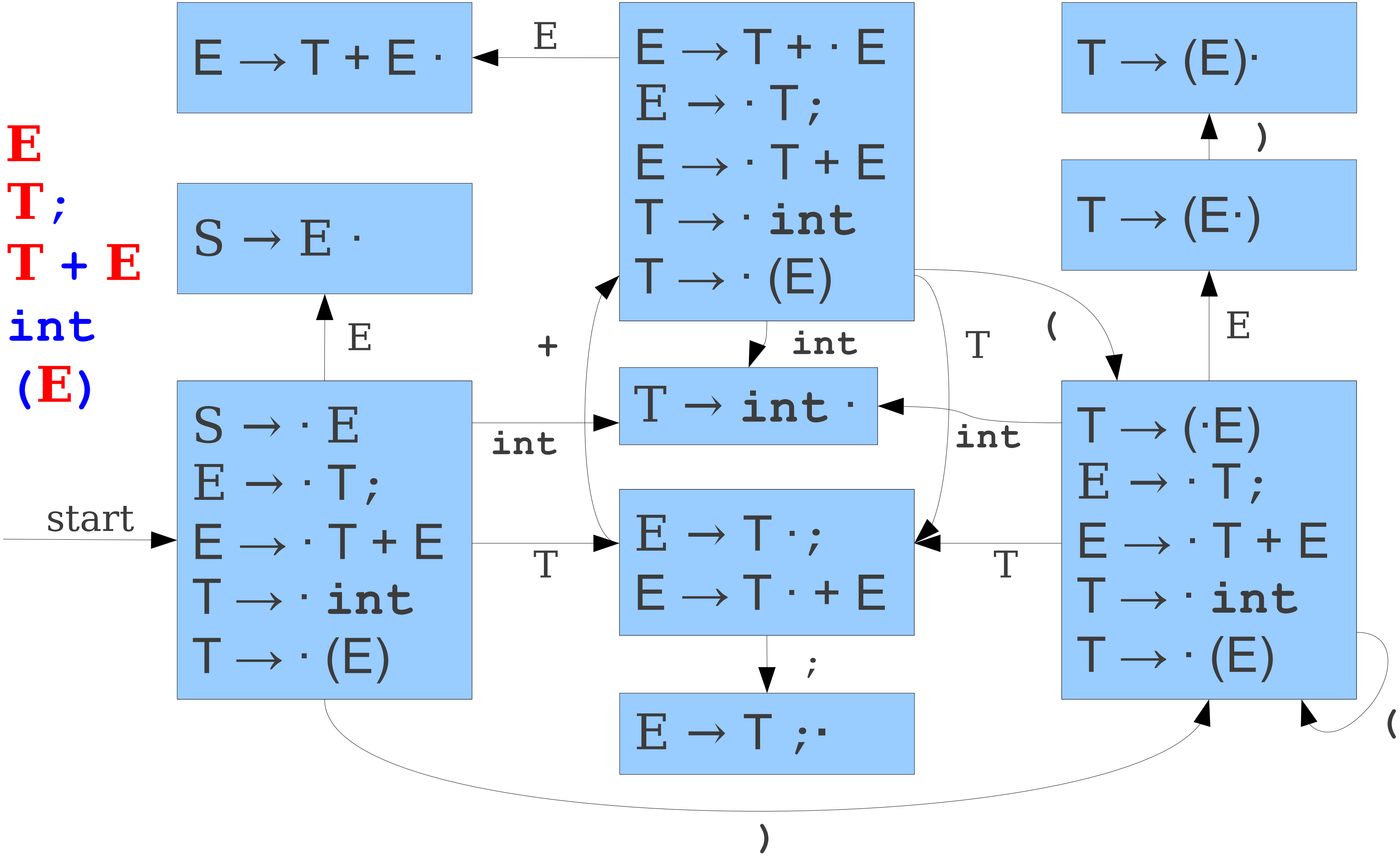
T	+	(	T	+	T	|	;	)	;

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**`)`



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

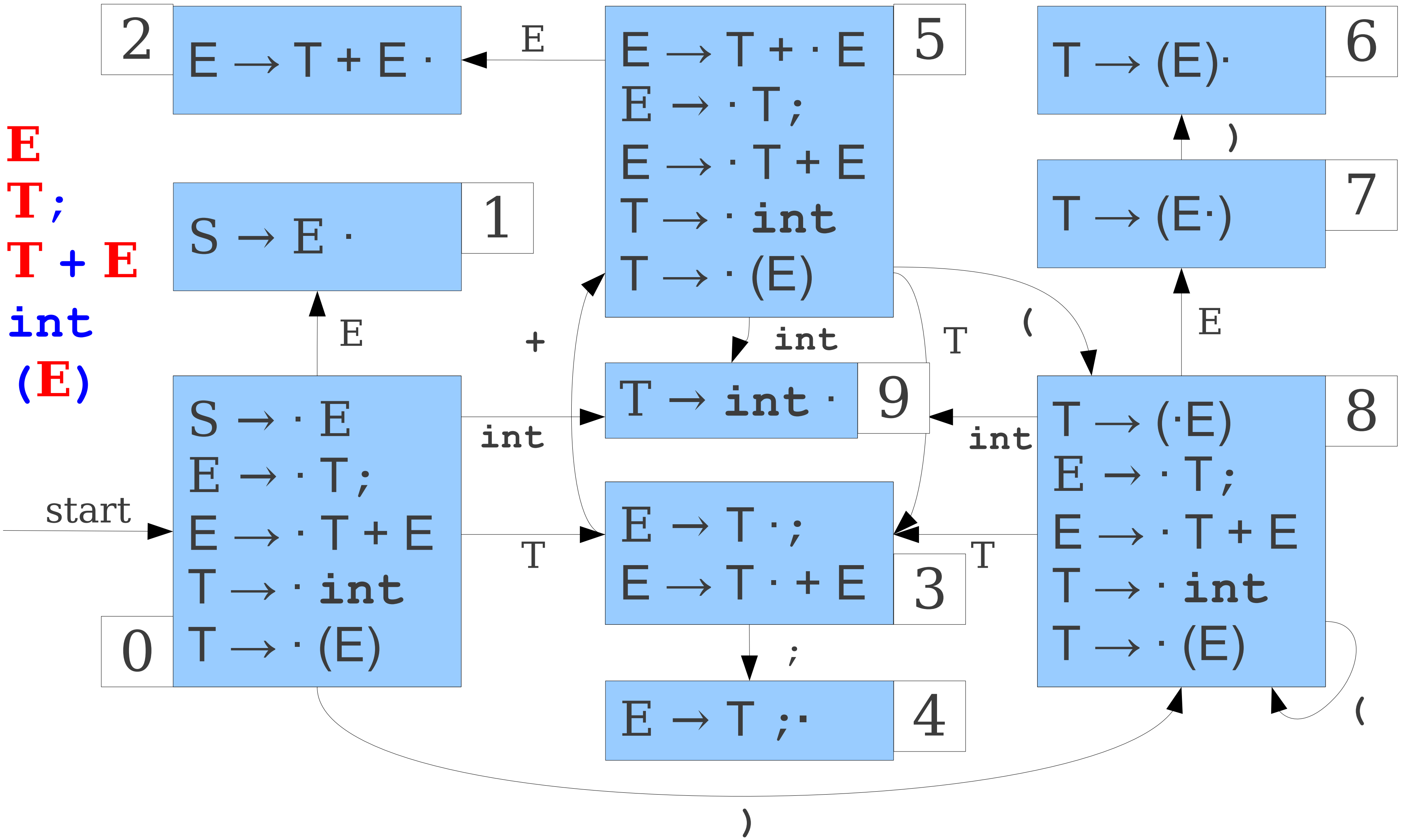
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

E → T ;·

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·)

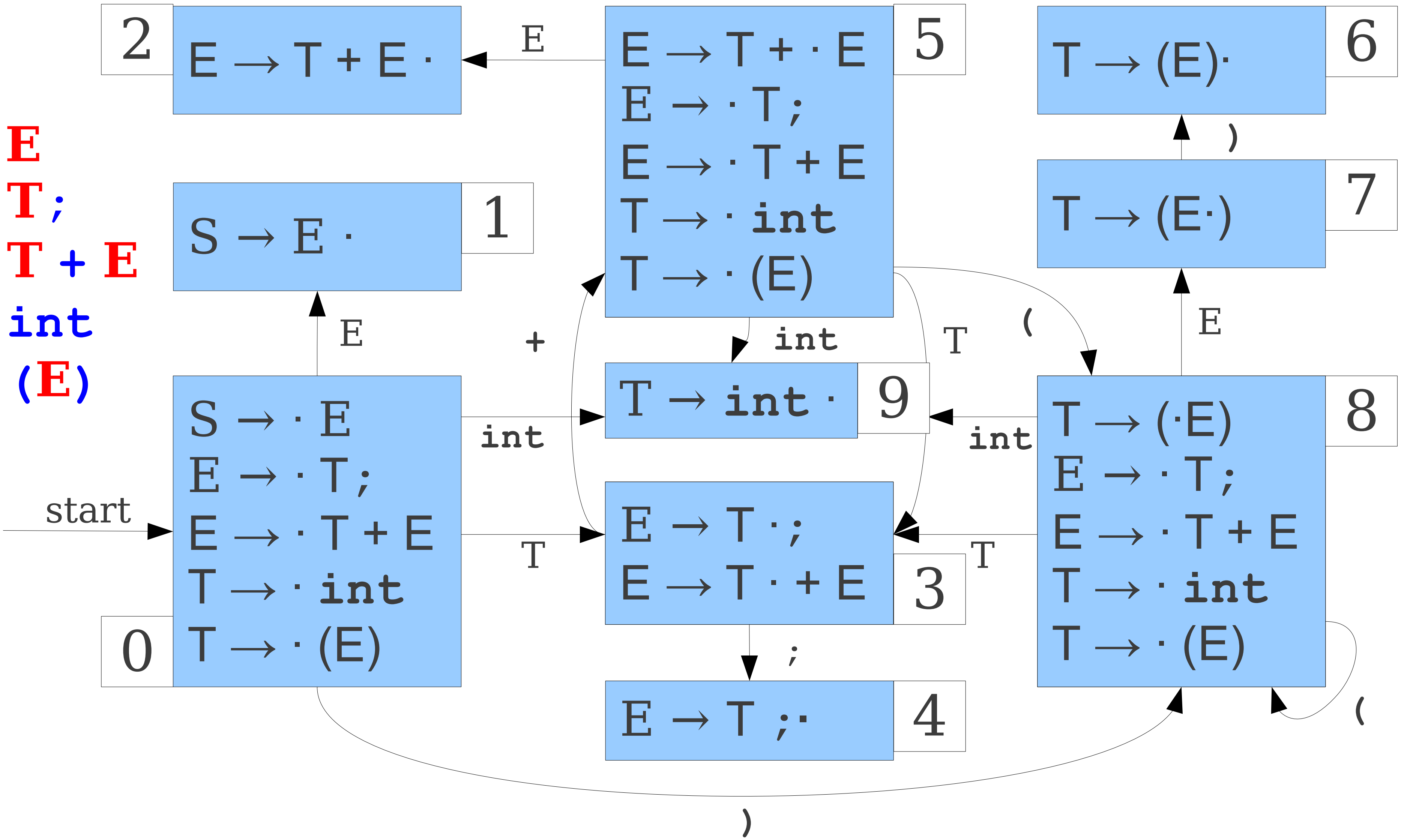
8
T → (·E)
E → · T;
E → · T + E
T → · `int`
T → · (E)

9
T → `int` ·

`int`	`+`	`(`	`int`	`+`	`int`	`;`	`)`	`;`

LR(0) Parsing

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

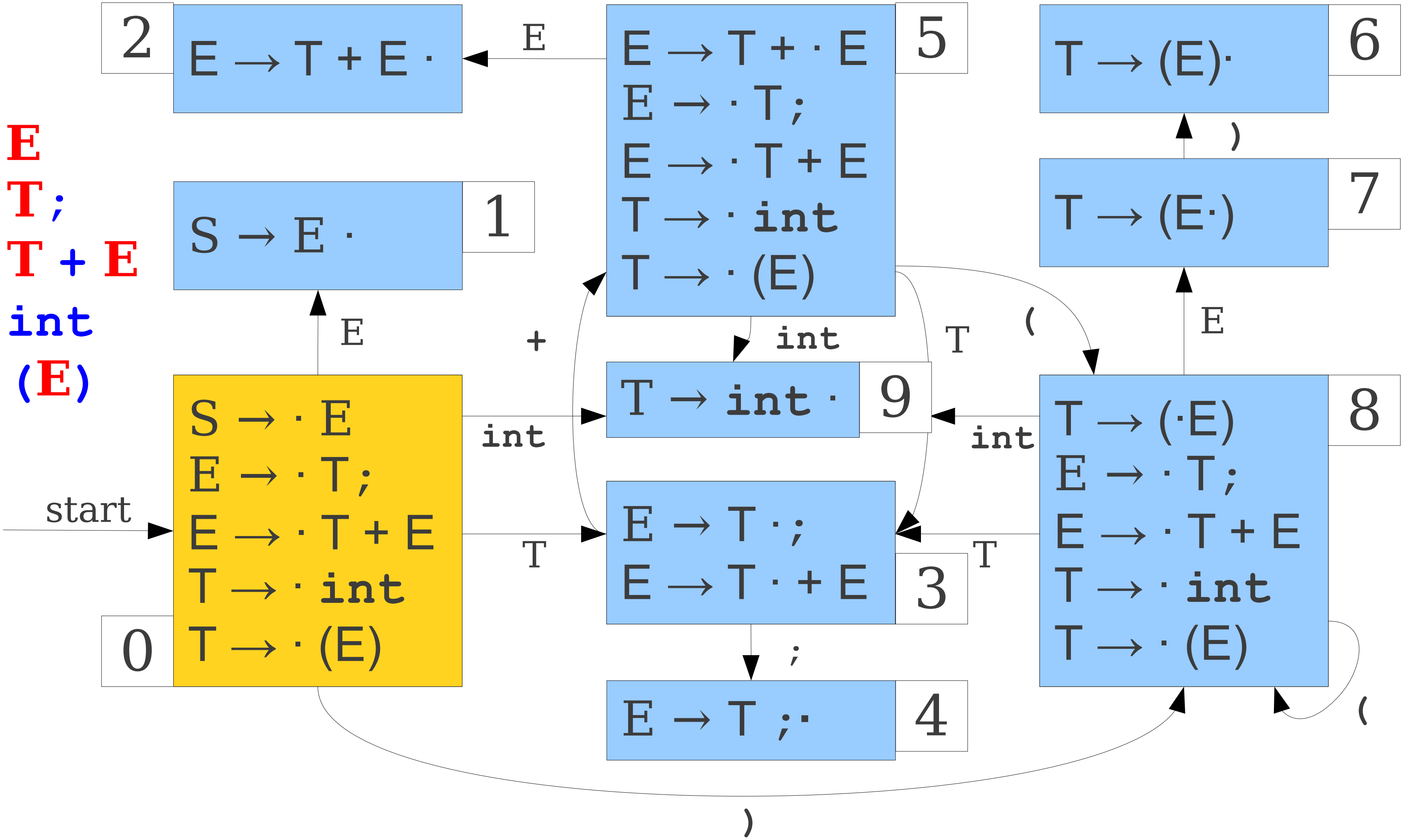


int	+	(	int	+	int	;	)	;

0

LR(0) Parsing

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



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)

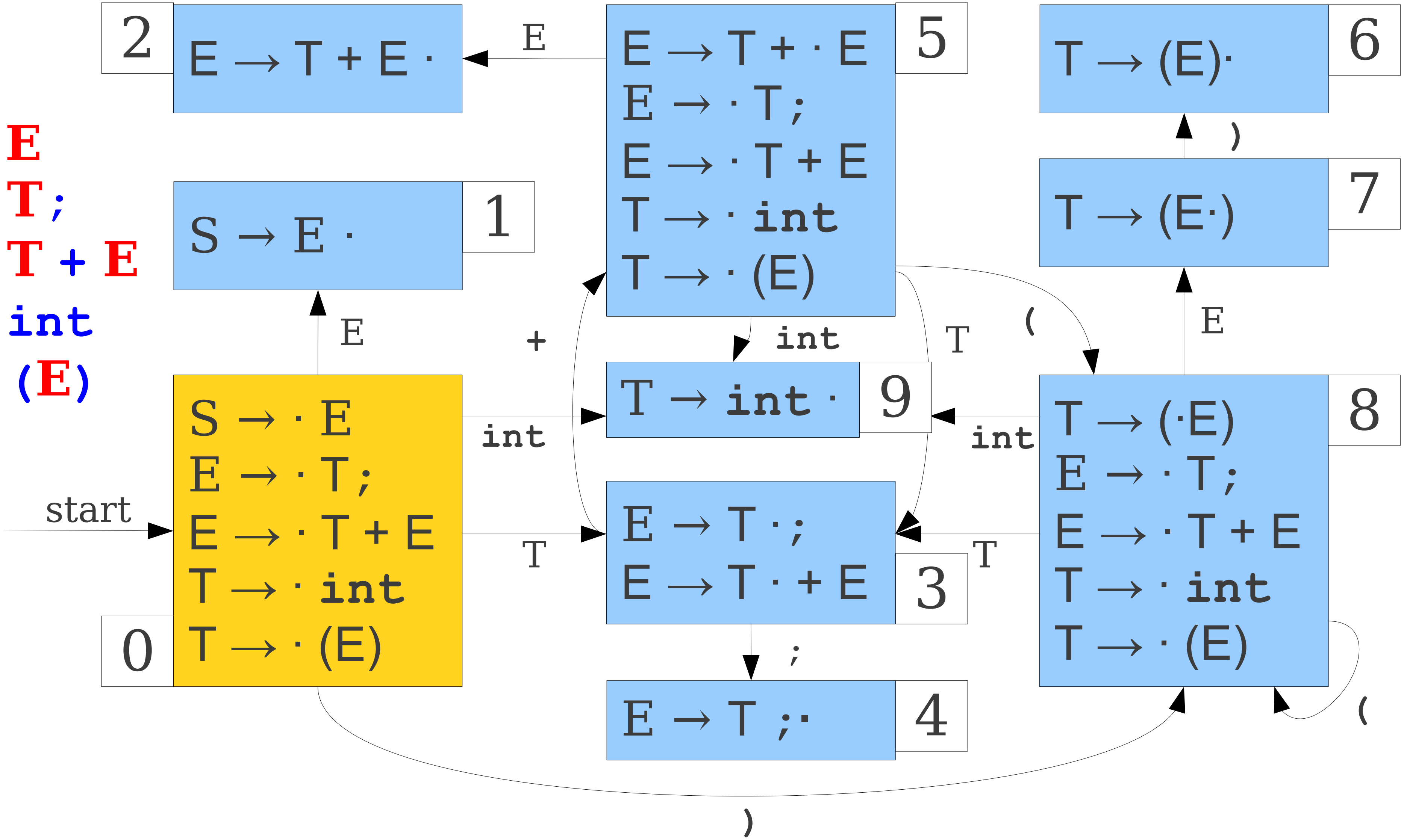
9
T → **int** ·

`int`	+	(	`int`	+	`int`	;	)	;

0

LR(0) Parsing

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

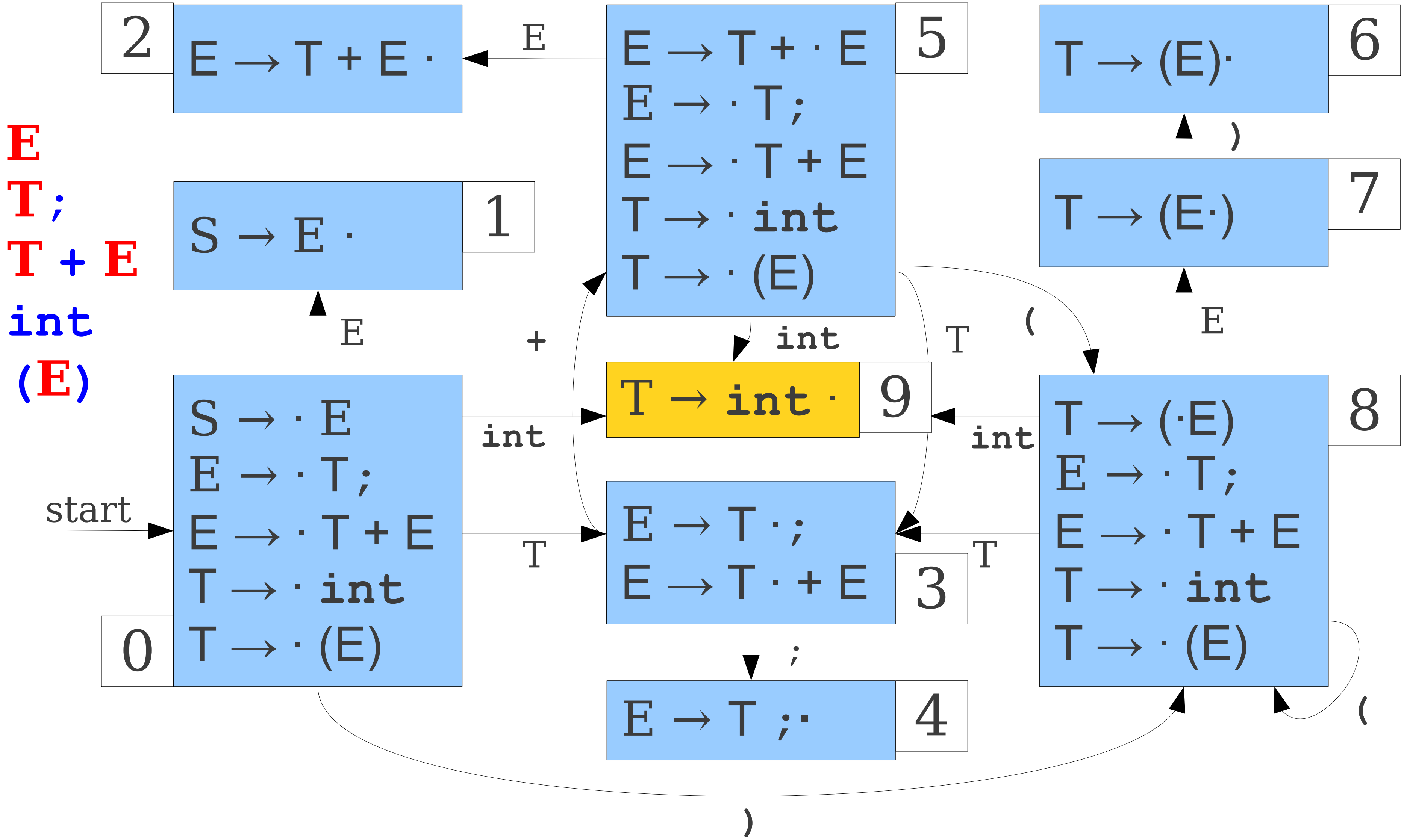


`int` | `+` `(` `int` `+` `int` `;` `)` `;`

0

LR(0) Parsing

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

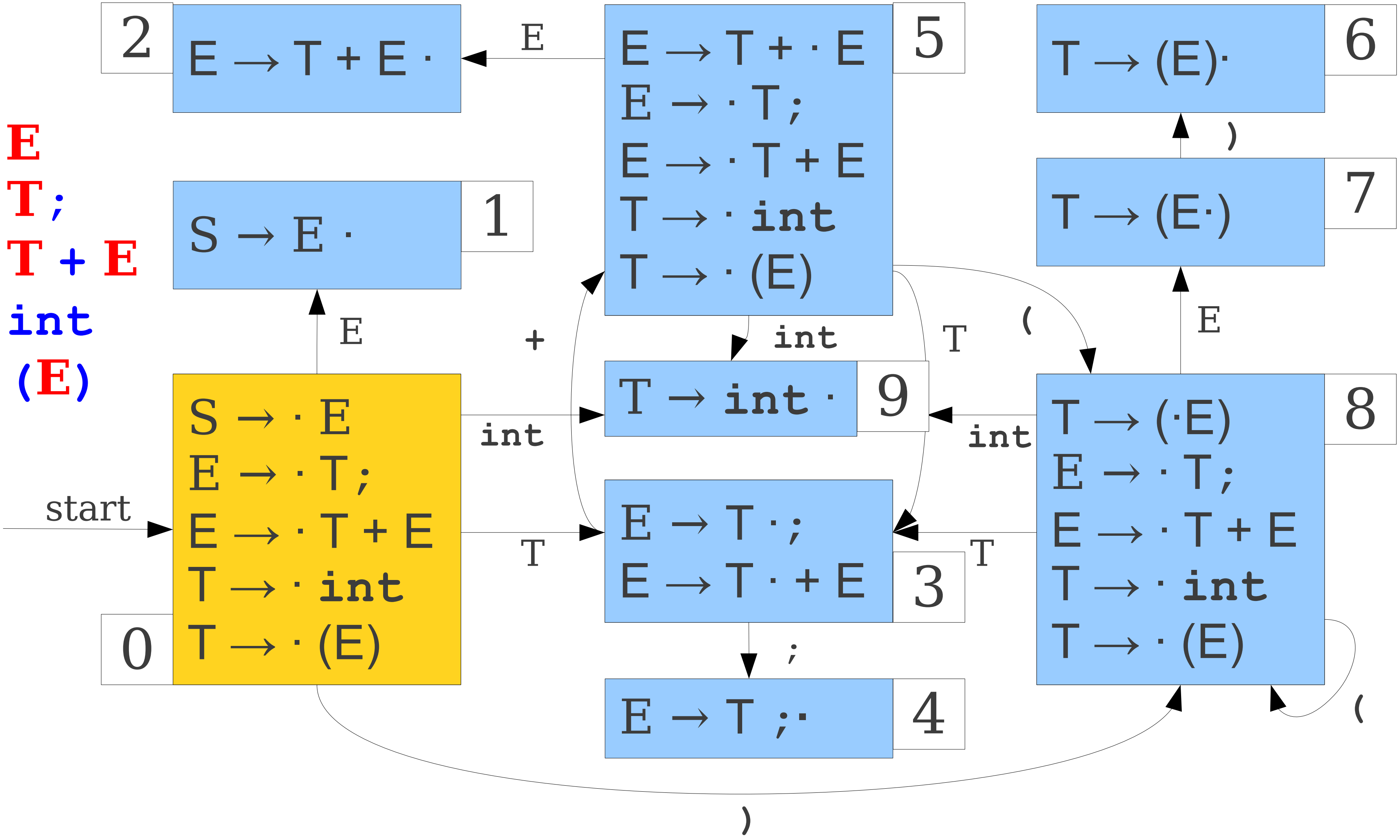


`int`		+	(	`int`	+	`int`	;	)	;

0

LR(0) Parsing

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



2
E → T + E ·

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)

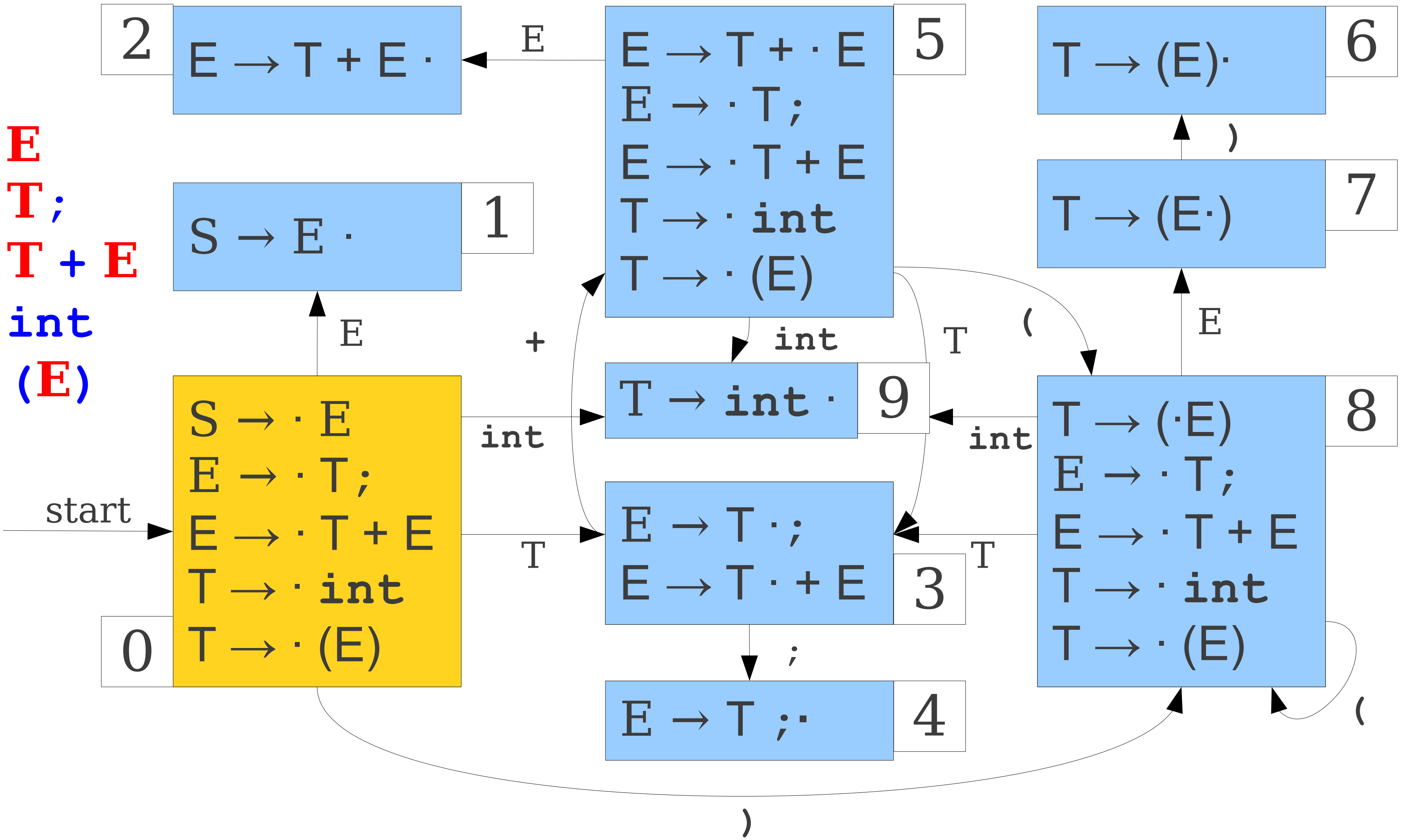
4
E → T ;·

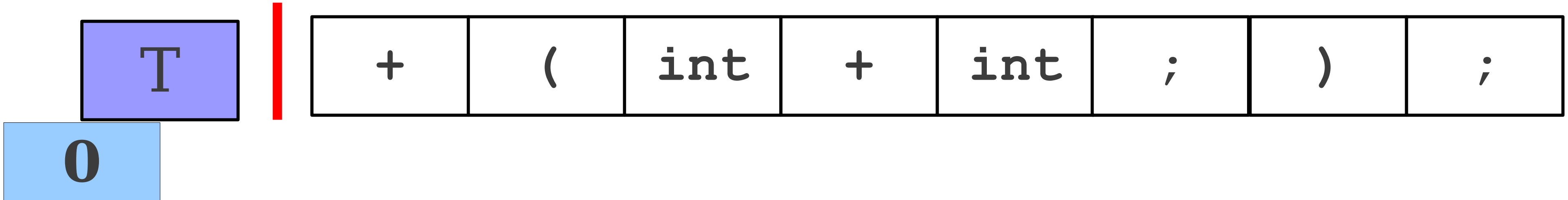
+	(	`int`	+	`int`	;	)	;

0

LR(0) Parsing

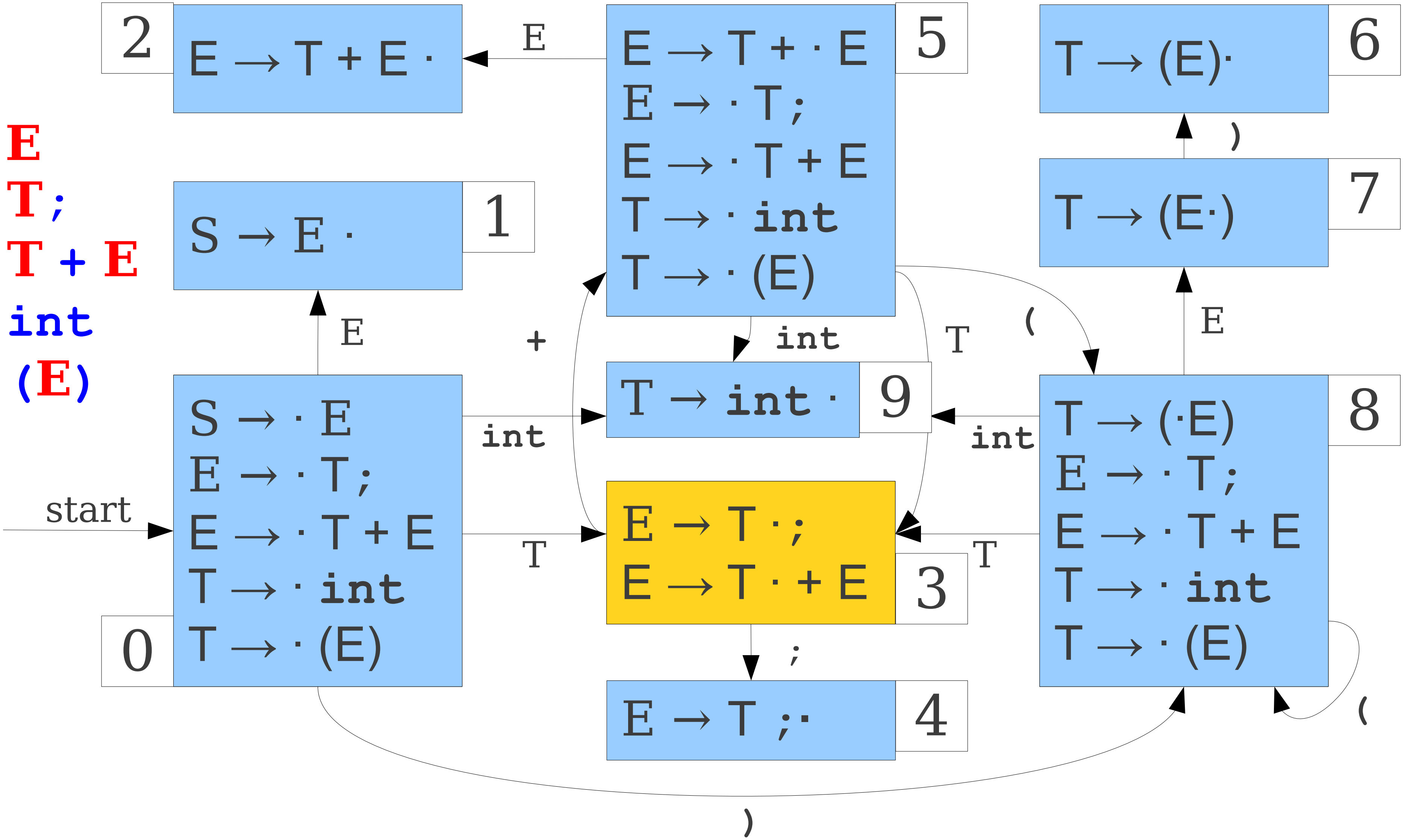
S → **E**
E → **T** ;
E → **T** + **E**
T → int
T → (**E**)





LR(0) Parsing

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



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)

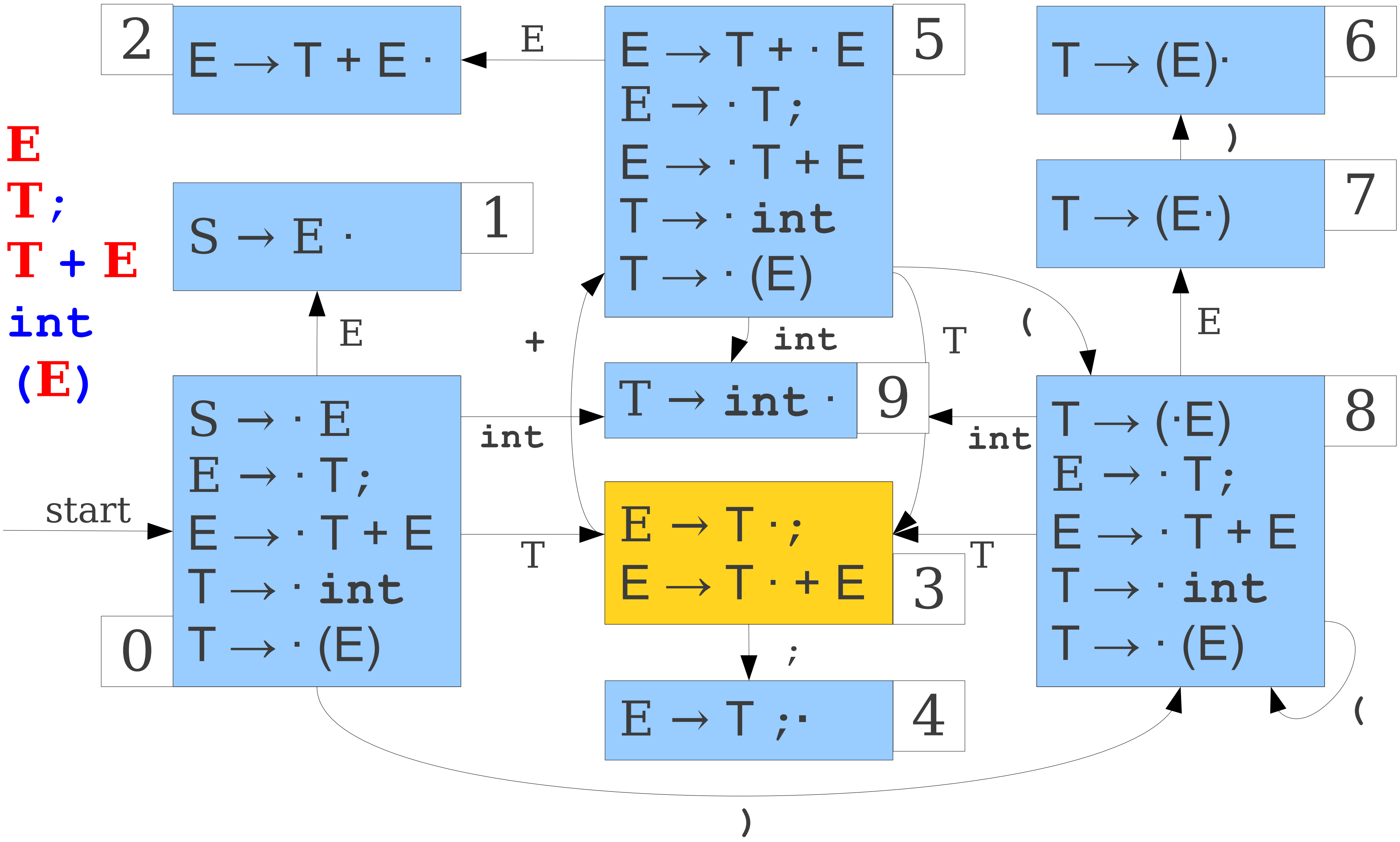
9
T → `int` ·

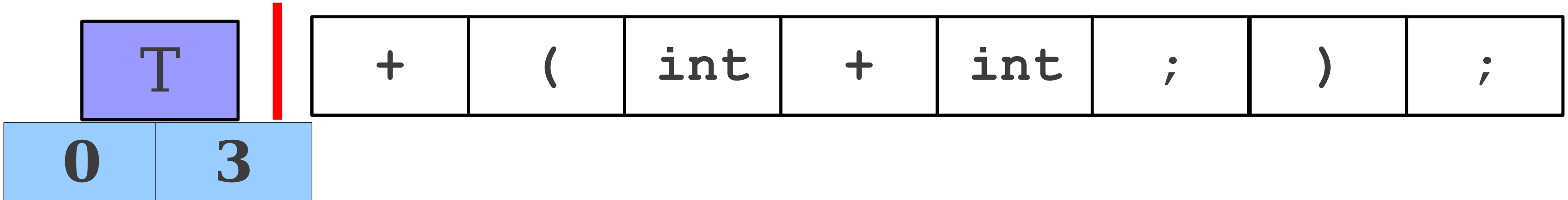
T
0

\+ (`int` + `int` ;) ;

LR(0) Parsing

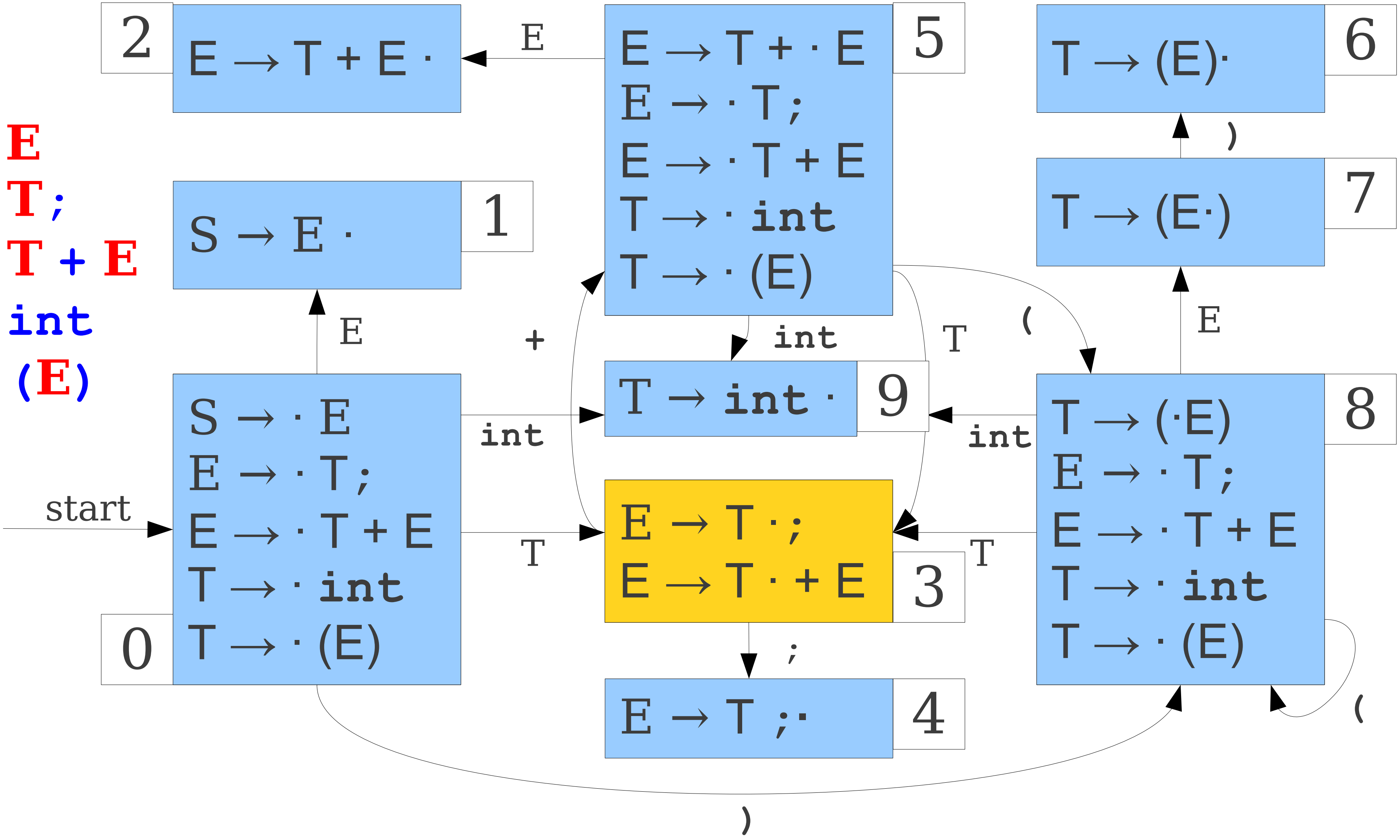
S → **E**
E → **T ;**
E → **T + E**
T → **int**
T → **(E)**





LR(0) Parsing

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

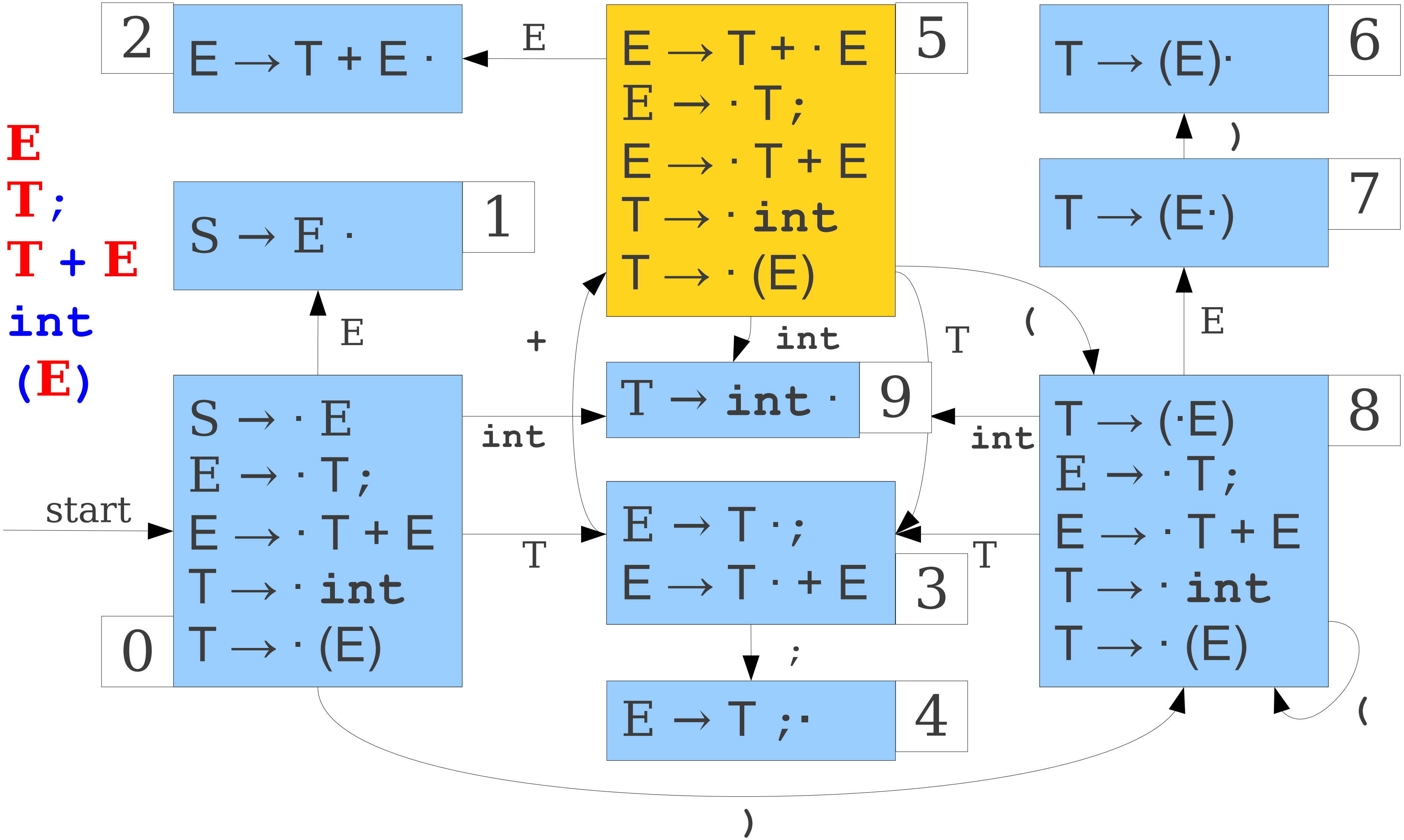


T	+
0	3

(	`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)

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)

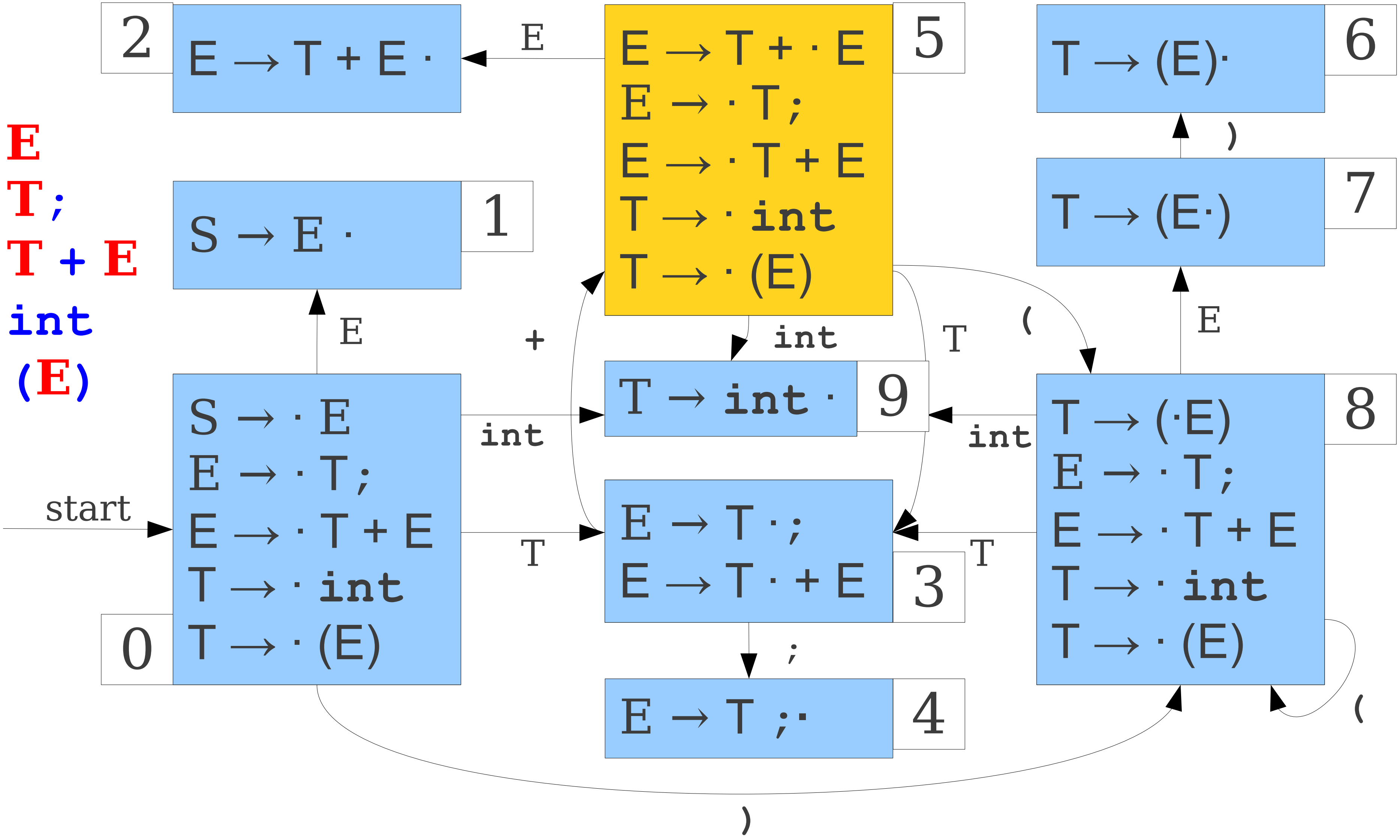
9
T → int ·

T	+
0	3

(	int	+	int	;	)	;

LR(0) Parsing

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

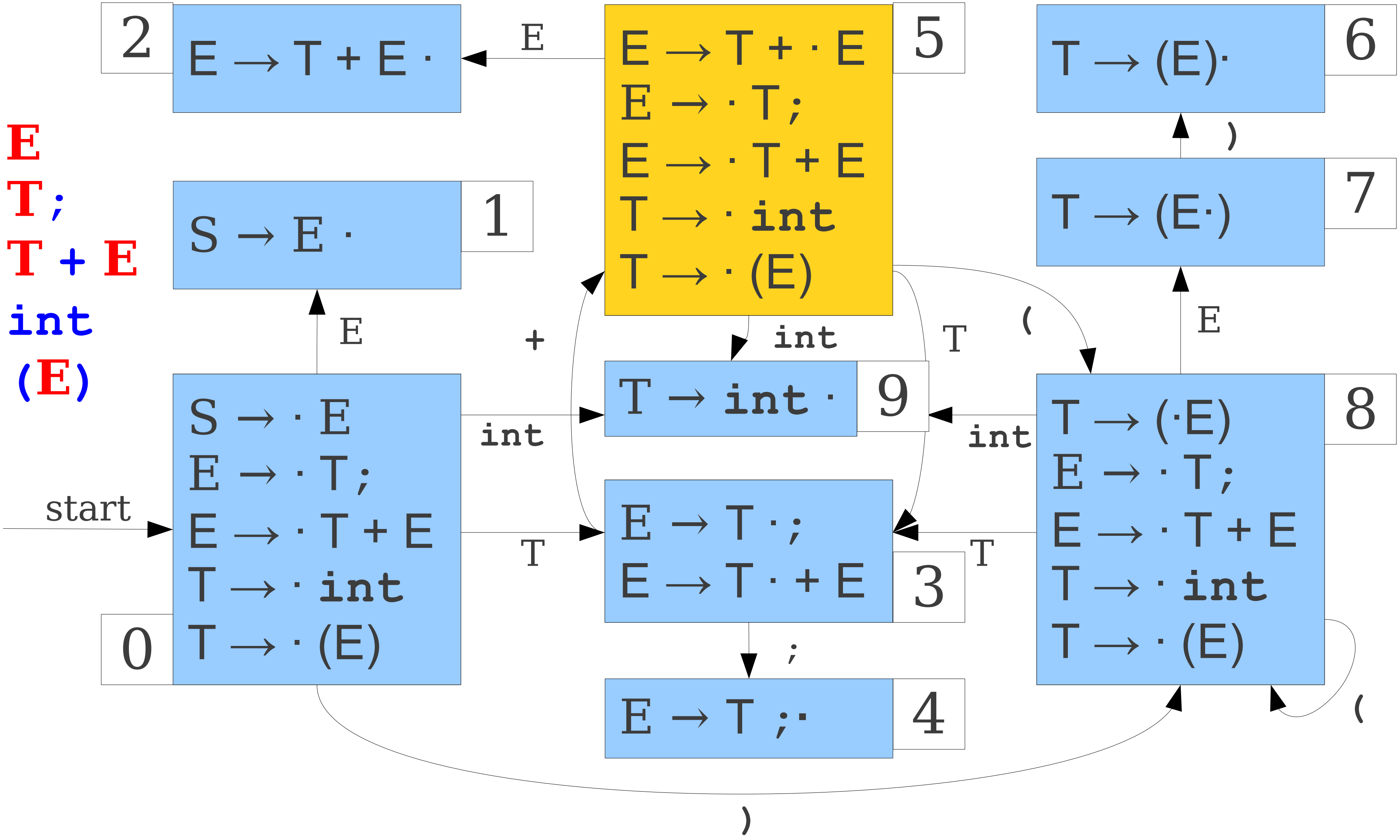


T	+
0	3	5

(	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·)

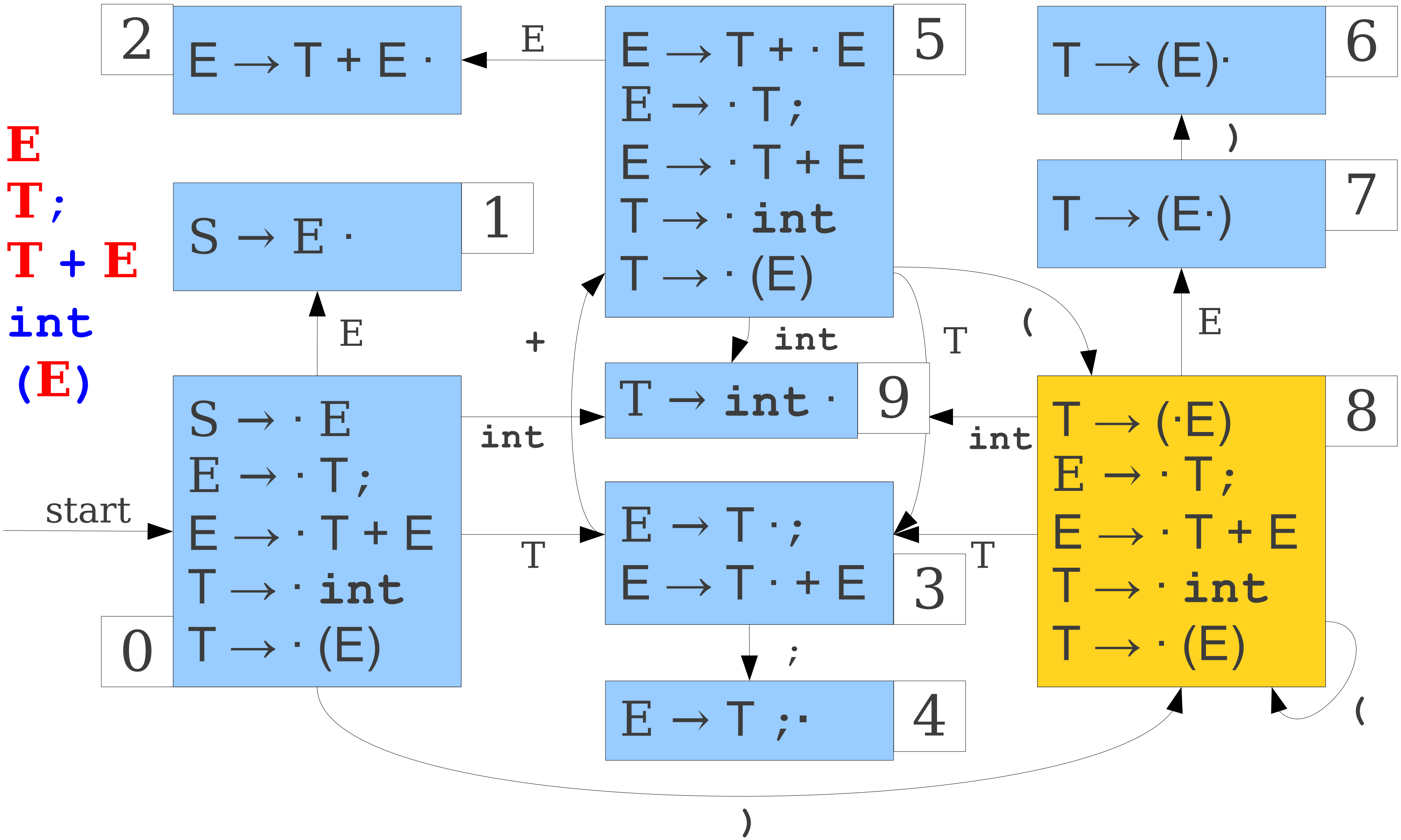
8: T → (·E)
E → · T;
E → · T + E
T → · `int`
T → · (E)

T	+	(
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)

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)

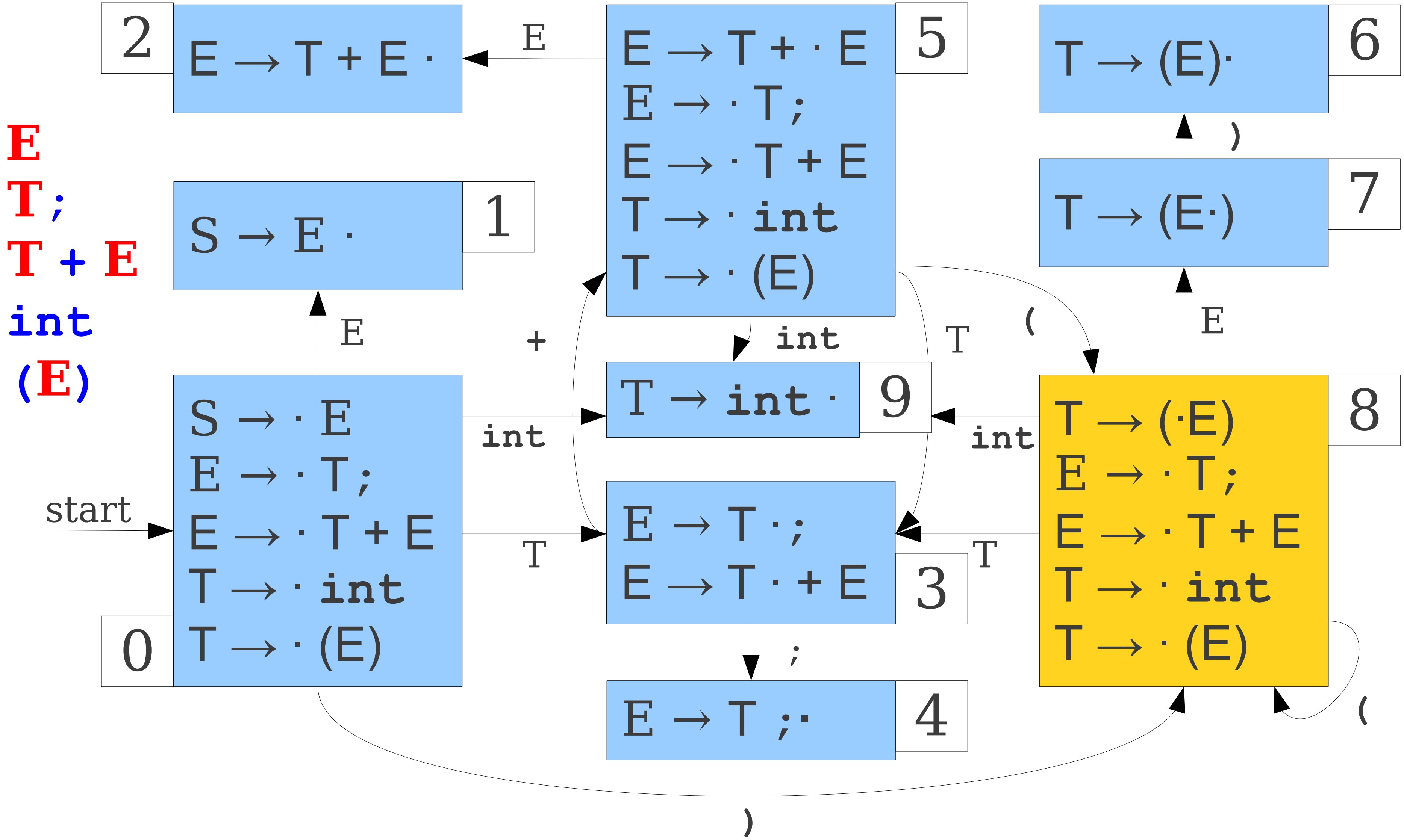
9
T → **int** ·

T	+	(
0	3	5

int	+	int	;	)	;

LR(0) Parsing

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

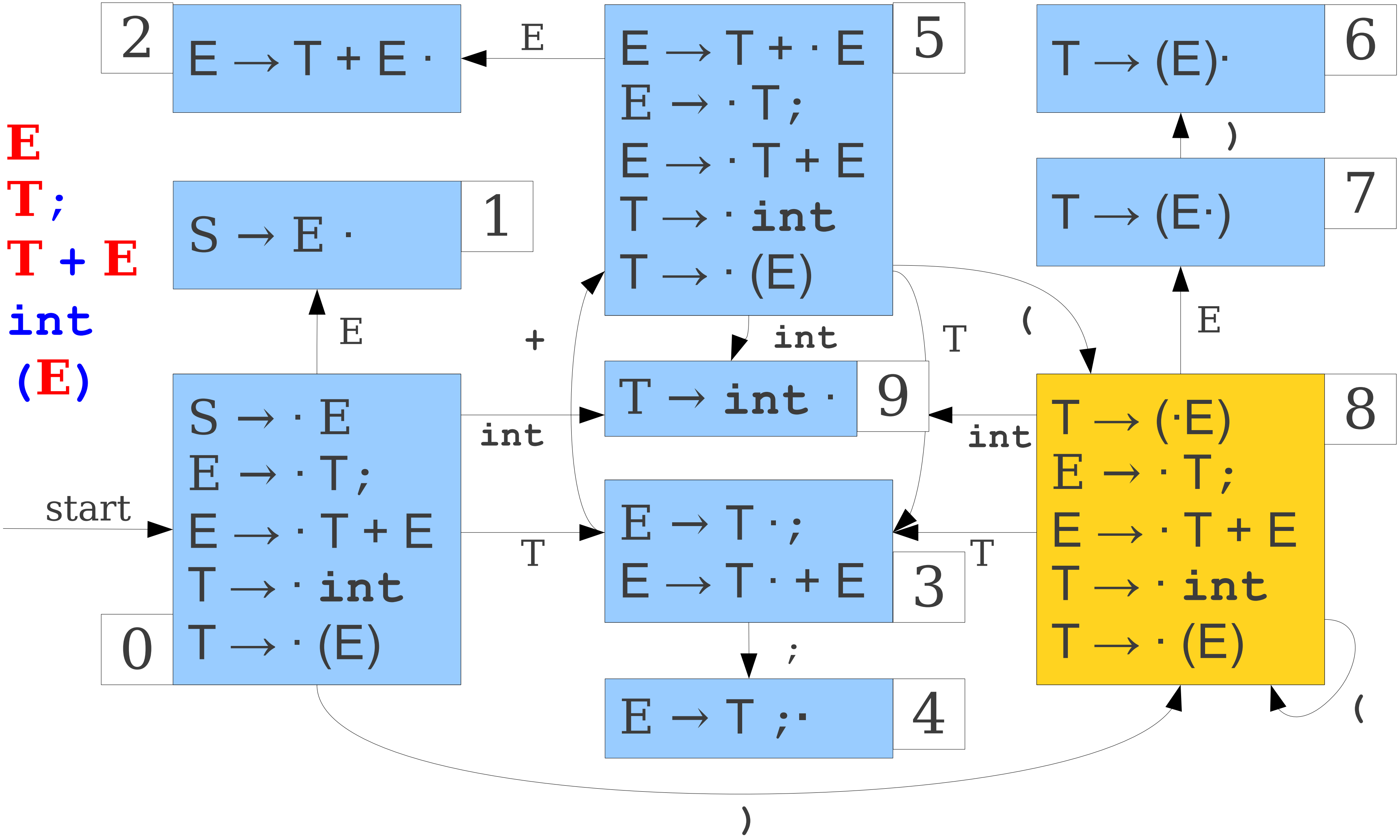


	T	+	(
0	3	5	8

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)

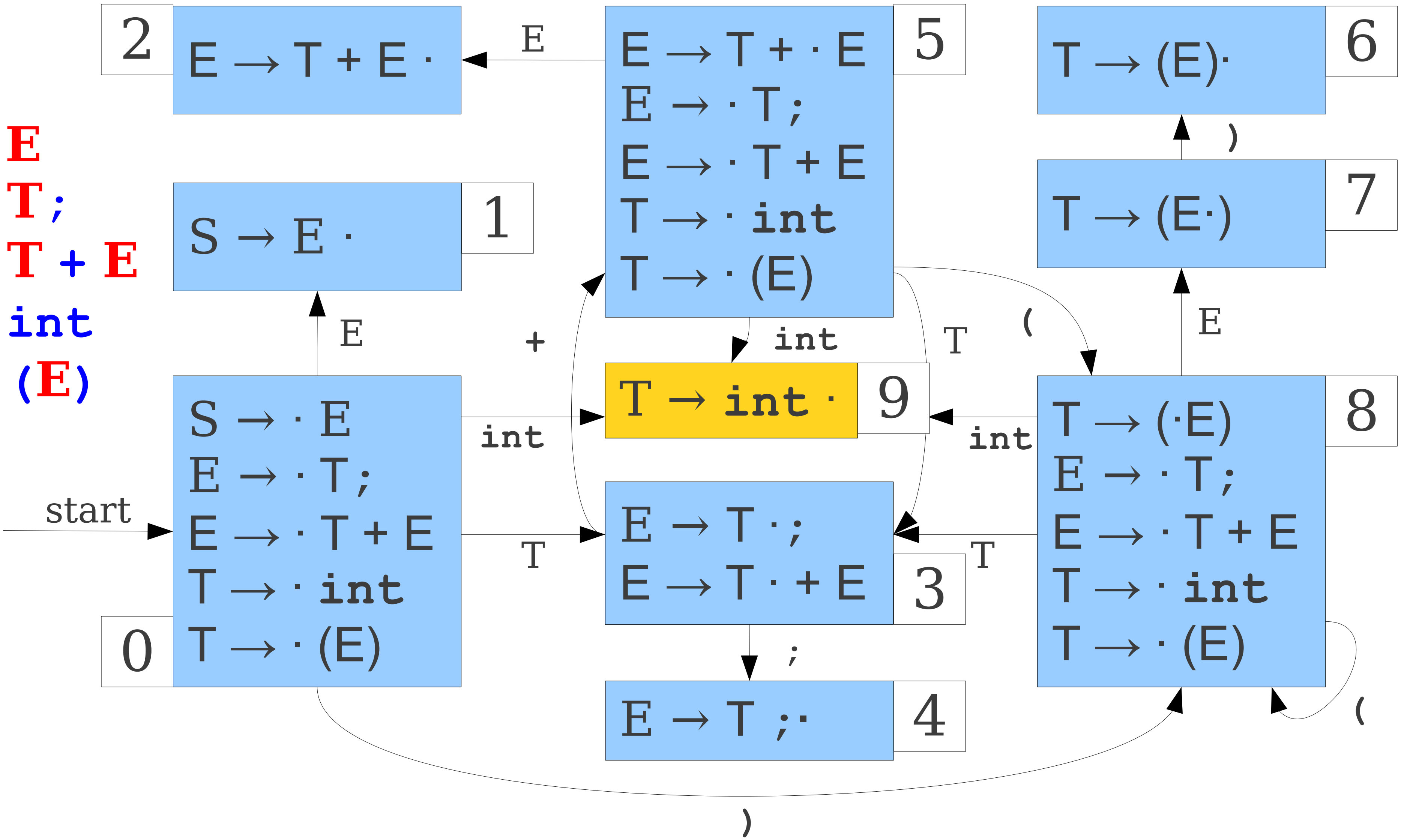
9
T → int ·

T	+	(	int		+	int	;	)	;

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 ·

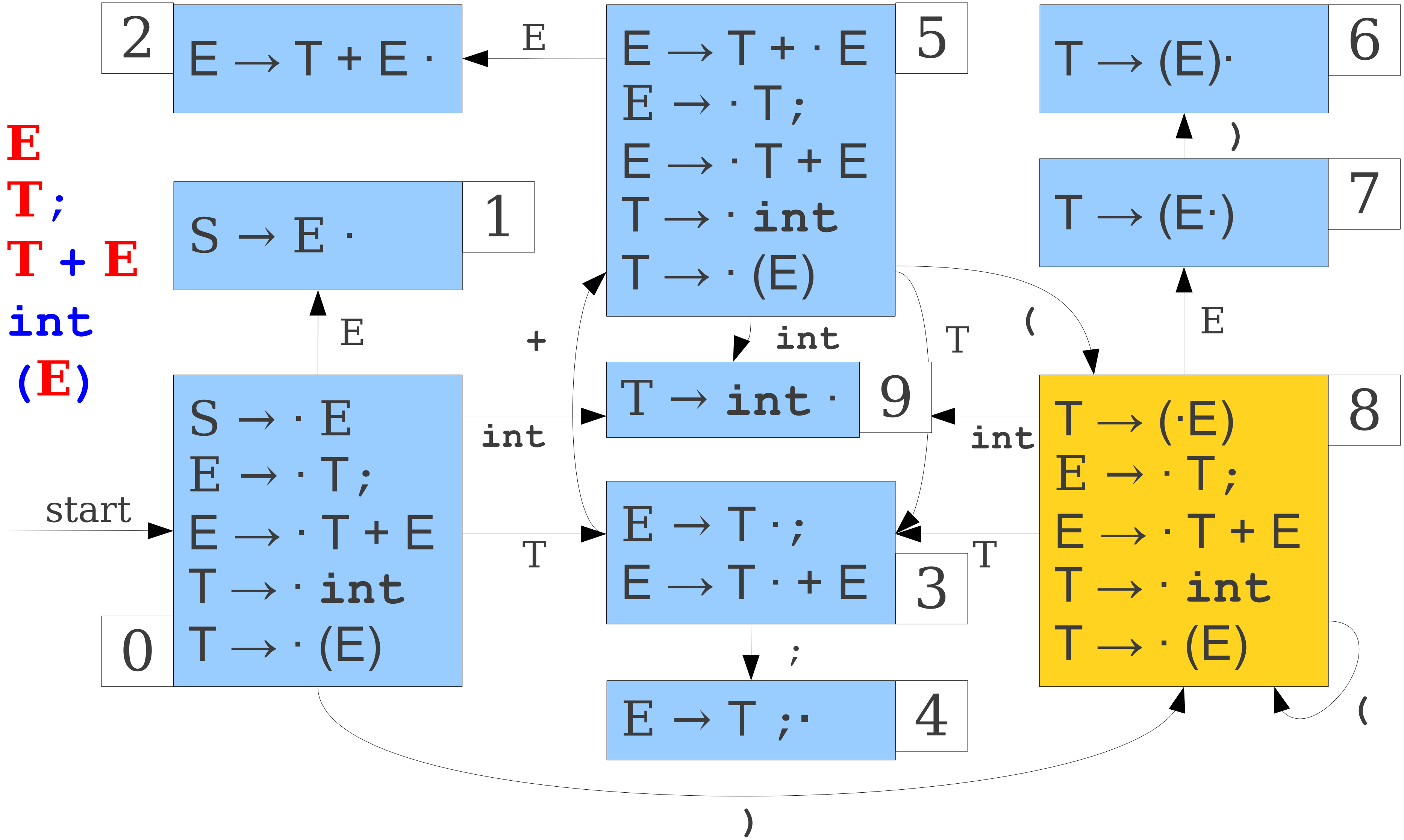
T	+	(	int

+	int	;	)	;

0	3	5	8

LR(0) Parsing

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



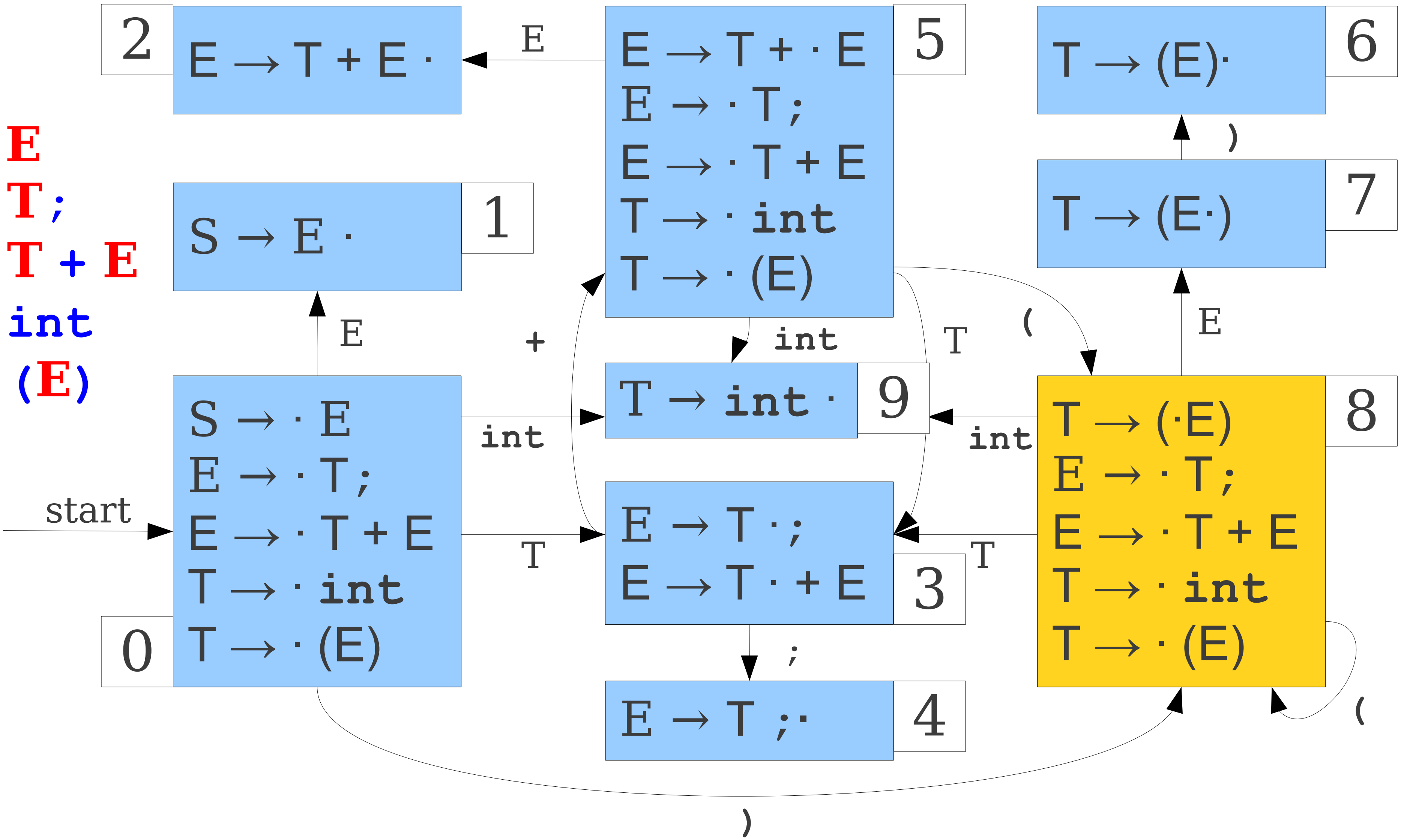
T	+	(

0	**3**	**5**	**8**

+	`int`	;	)	;

LR(0) Parsing

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

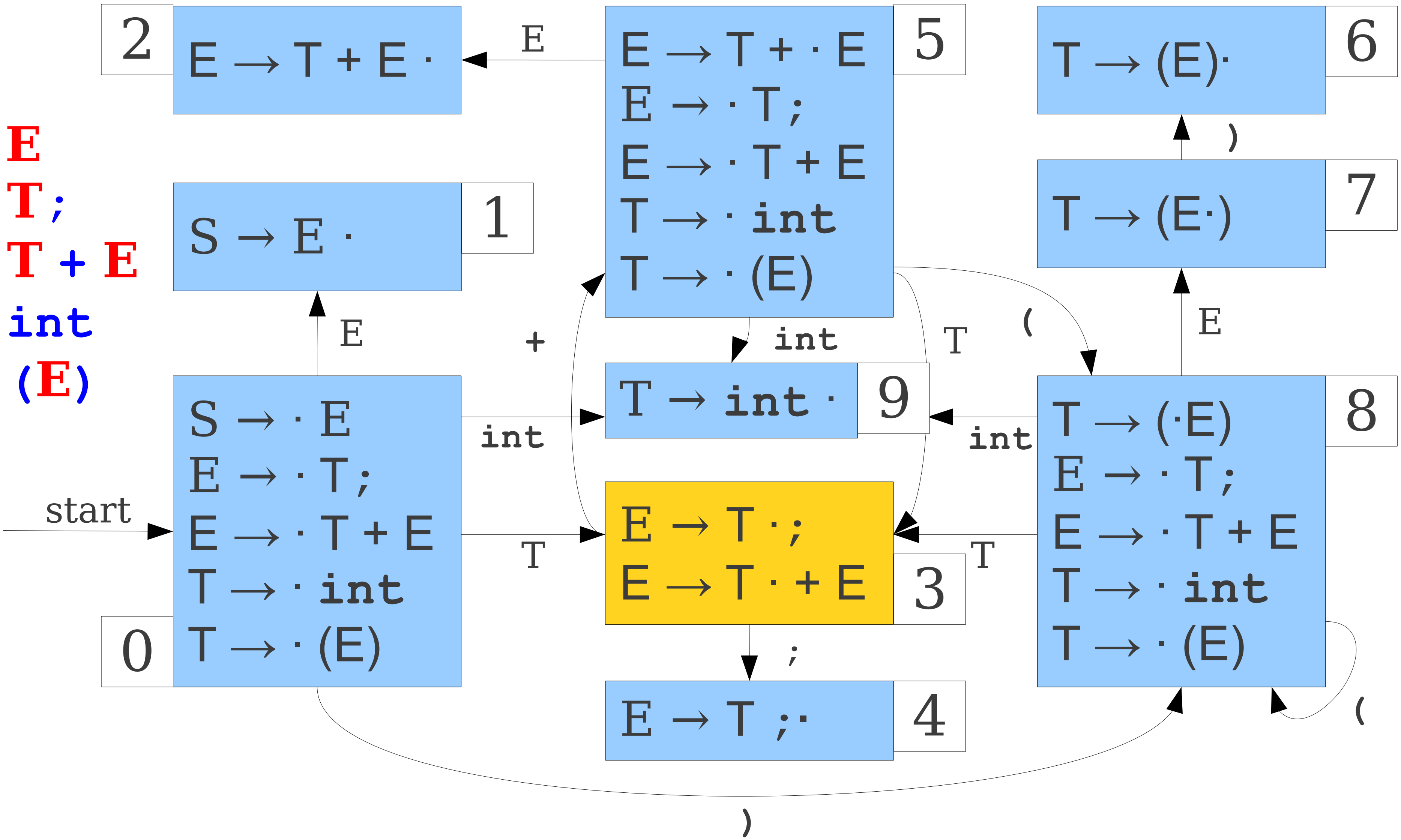


T	+	(	T
0	**3**	**5**	**8**

+	`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)

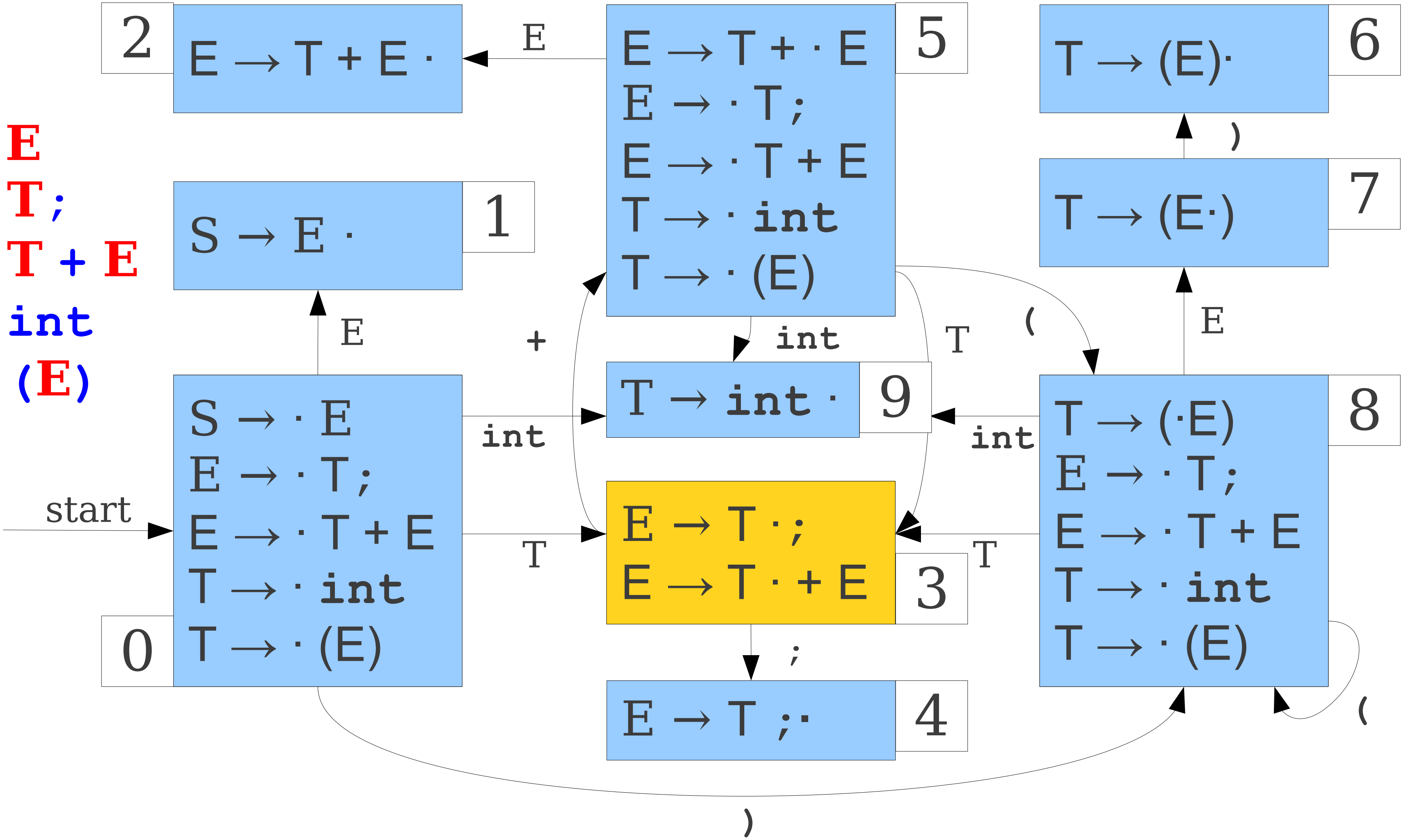
9
T → `int` ·

T	+	(	T
0	3	5	8

+	`int`	;	)	;

LR(0) Parsing

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



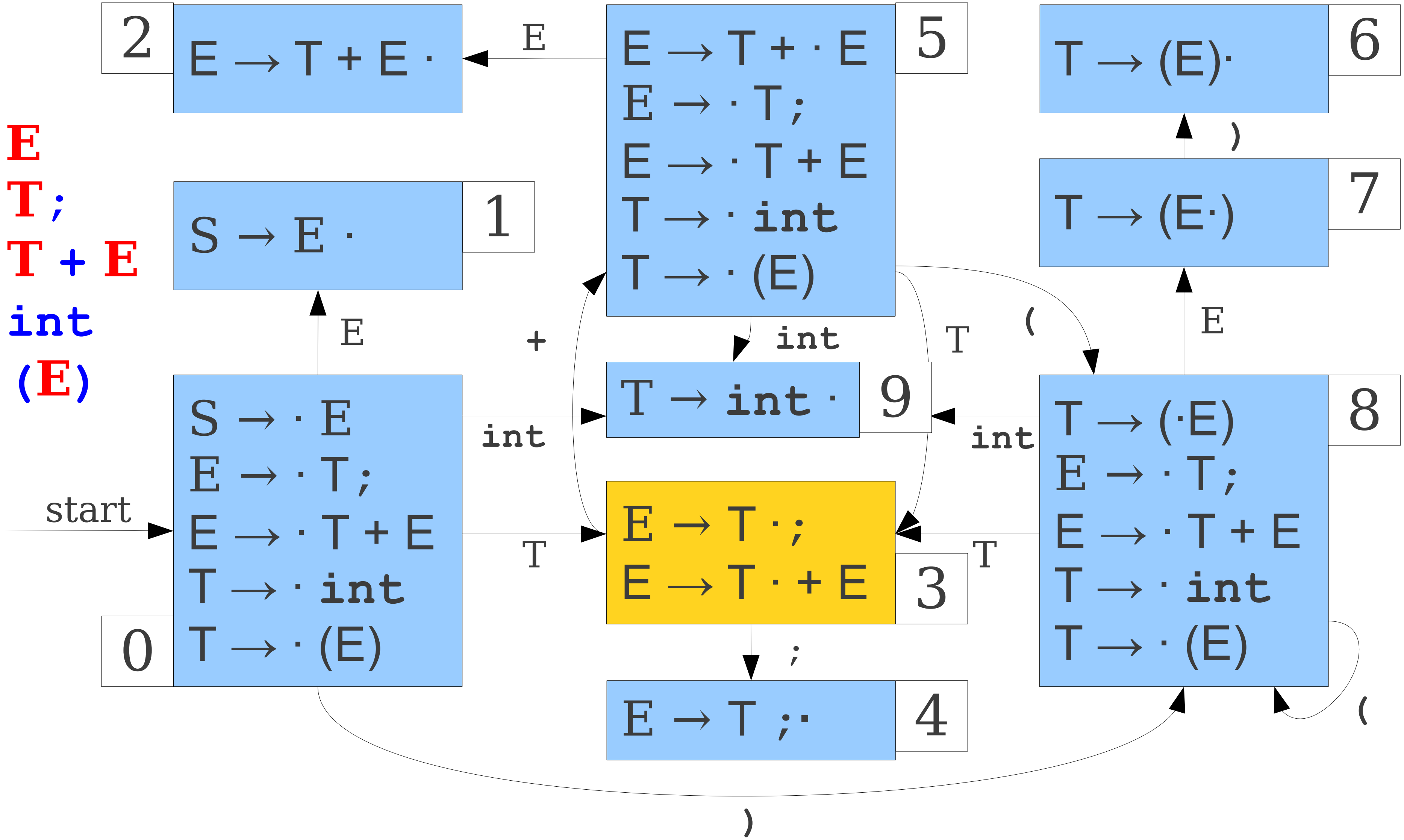
T	+	(	T

+	int	;	)	;

0	3	5	8	3

LR(0) Parsing

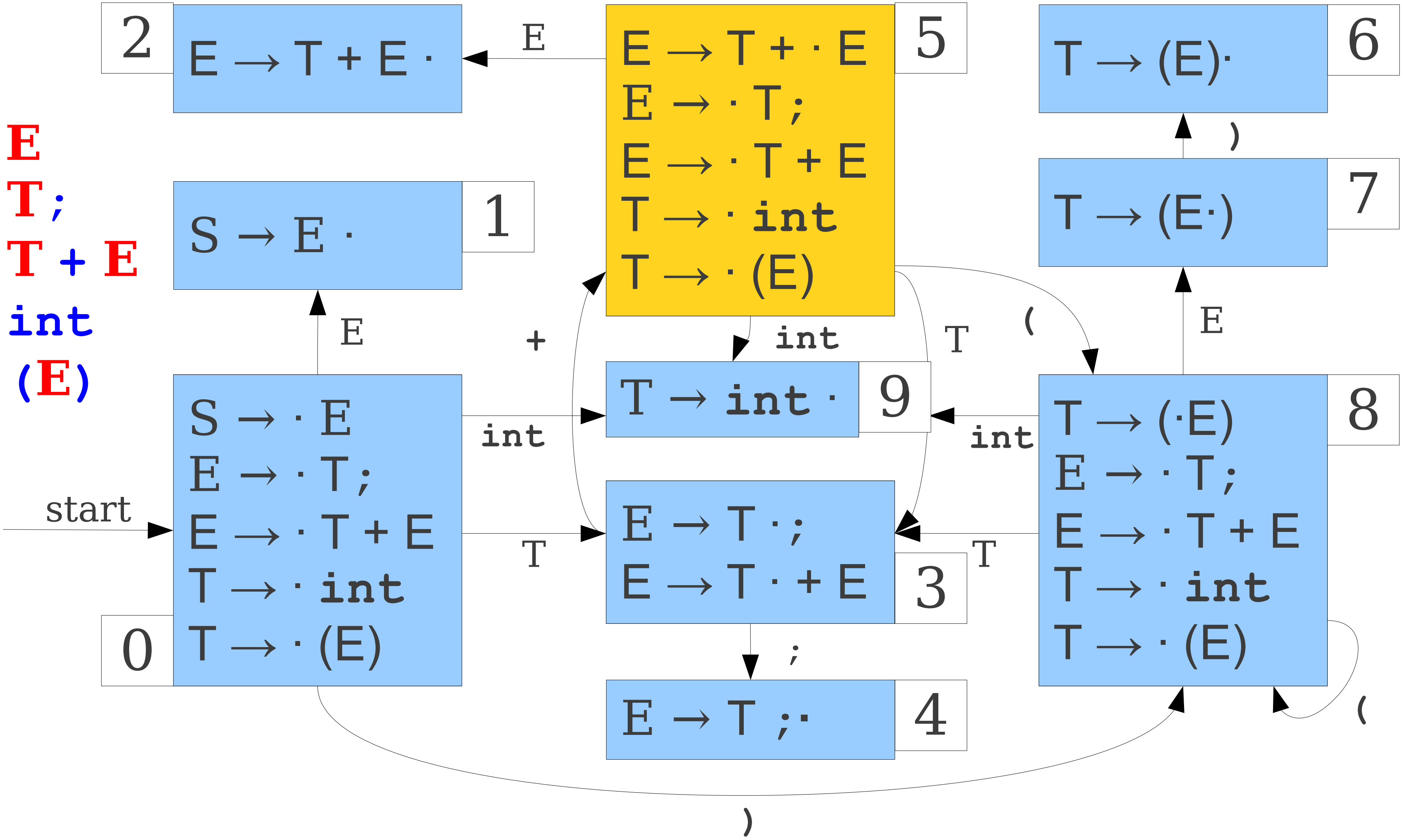
S → **E**
E → **T;**
E → **T + E**
T → **`int`**
T → **`(E)`**



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)·

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)

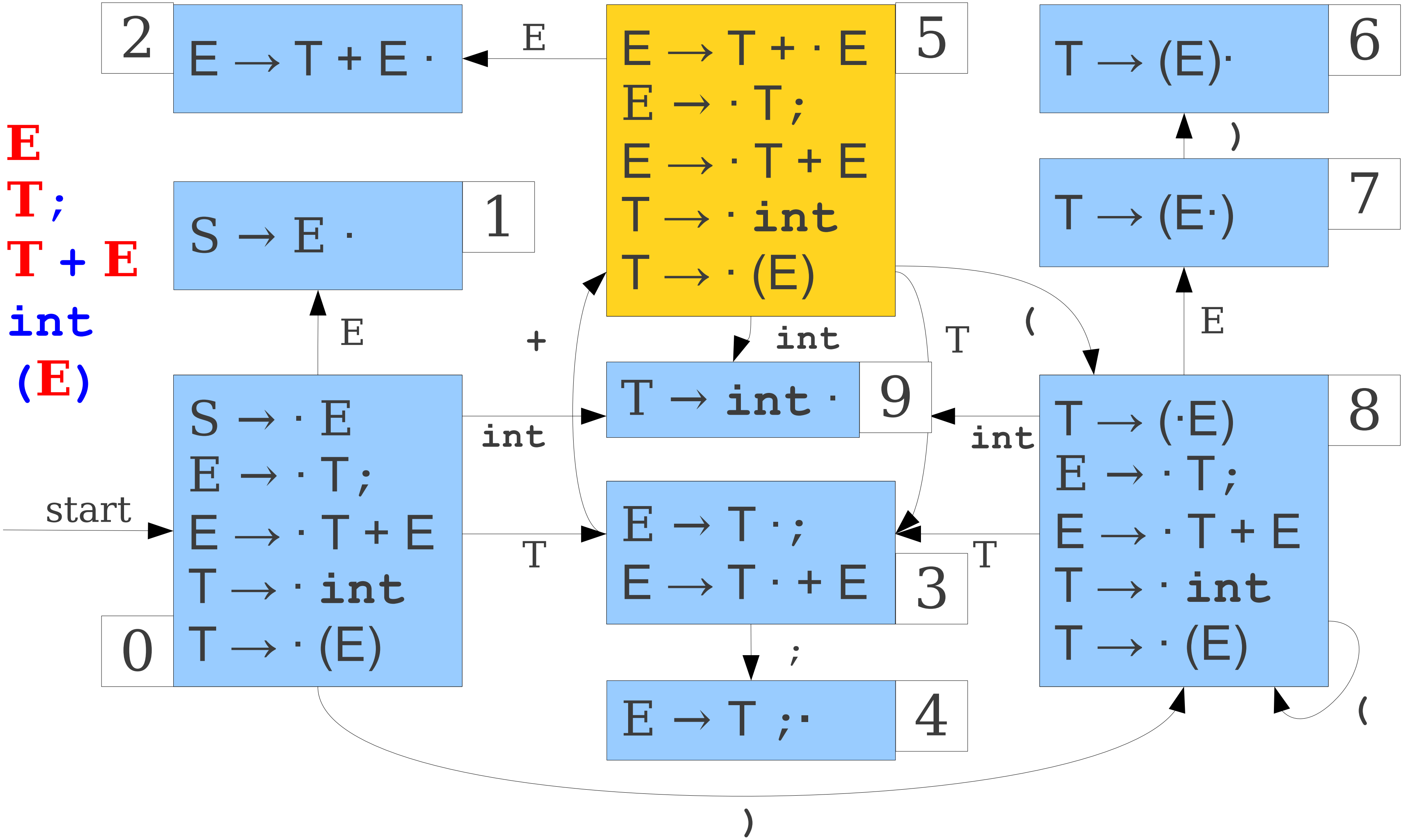
4
E → T ;·

T	+	(	T	+		`int`	;	)	;

0	**3**	**5**	**8**	**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)

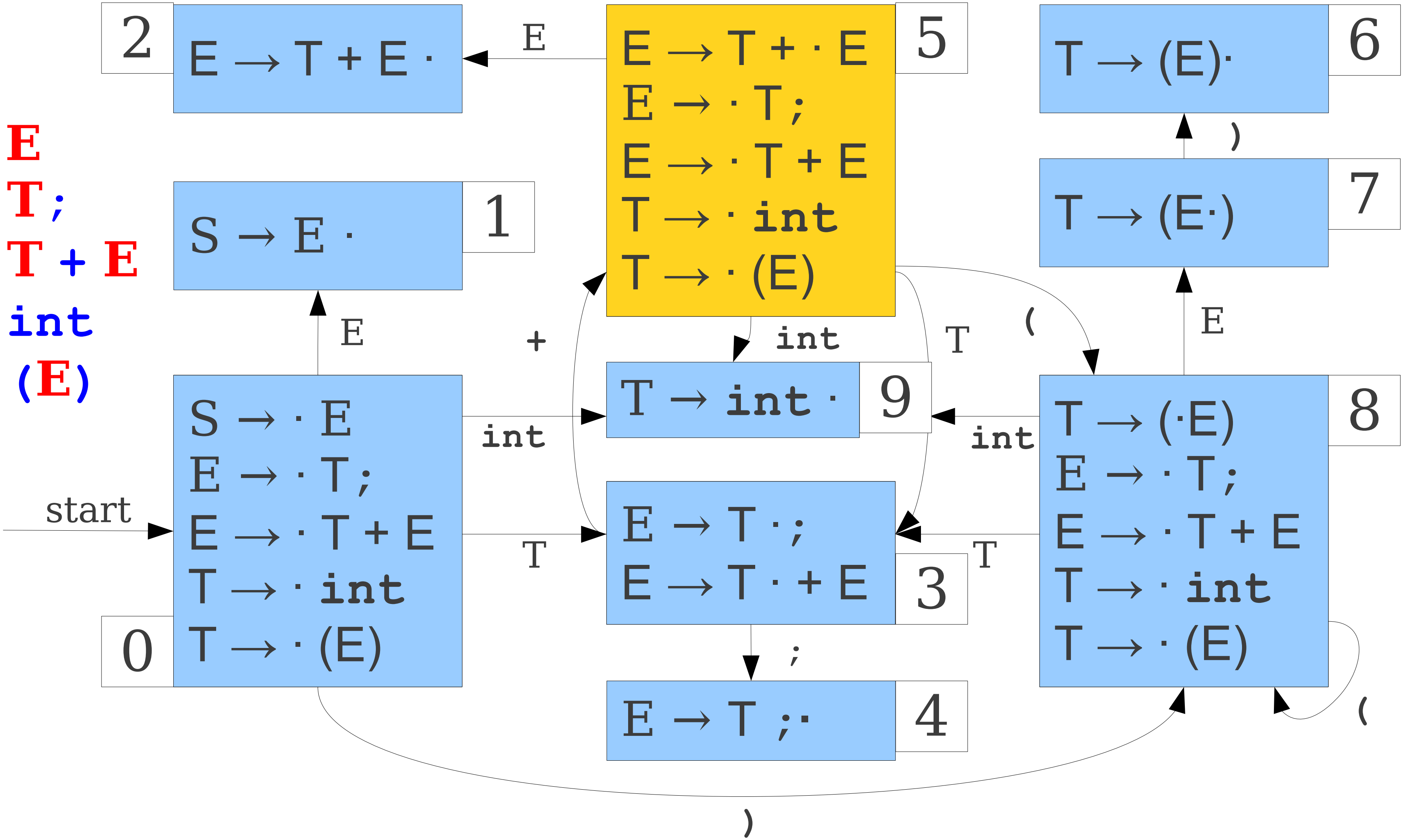
9
T → `int` ·

T	+	(	T	+
0	3	5	8	3	5

`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)

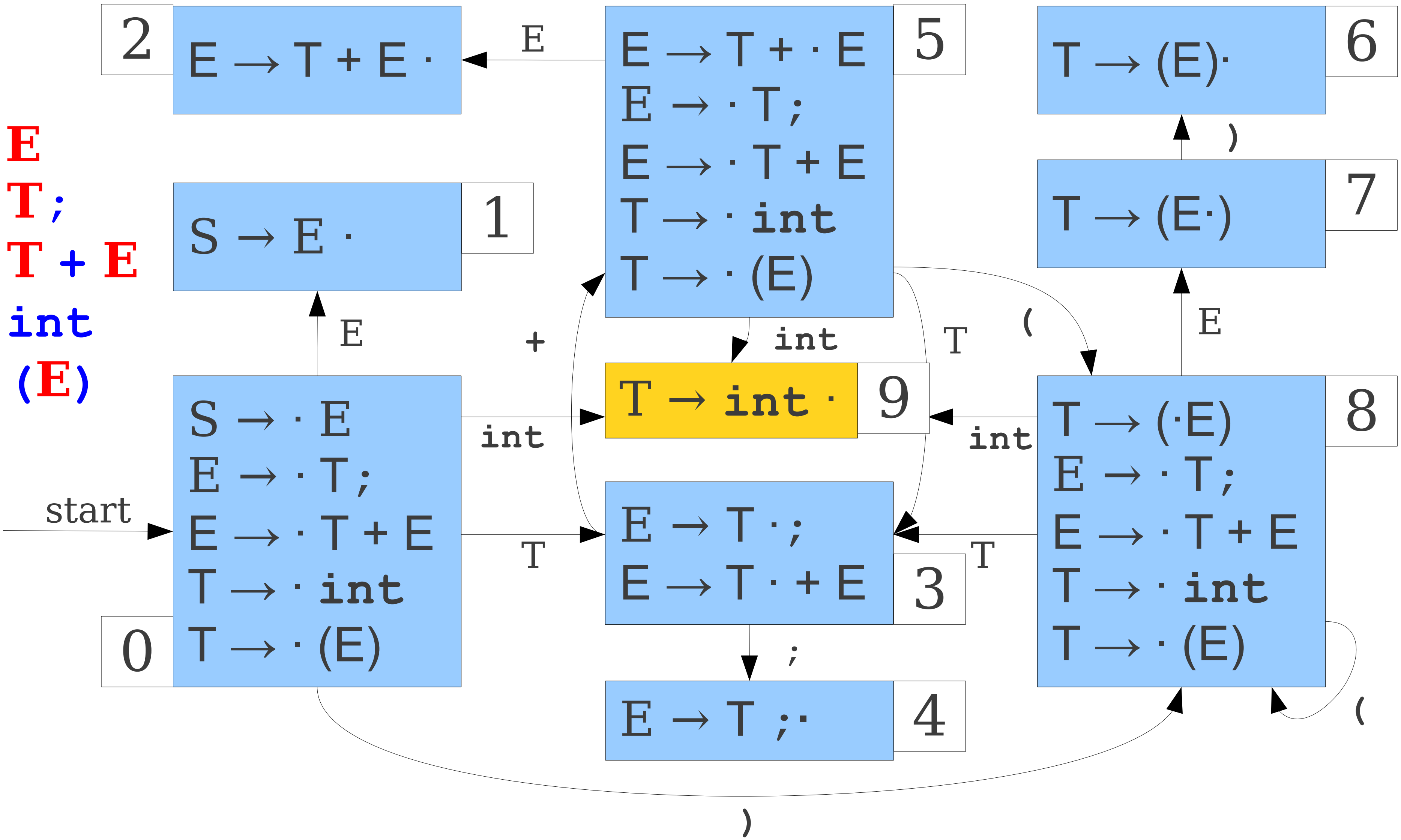
9
T → `int` ·

T	+	(	T	+	`int`	|	;	)	;

0	**3**	**5**	**8**	**3**	**5**

LR(0) Parsing

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

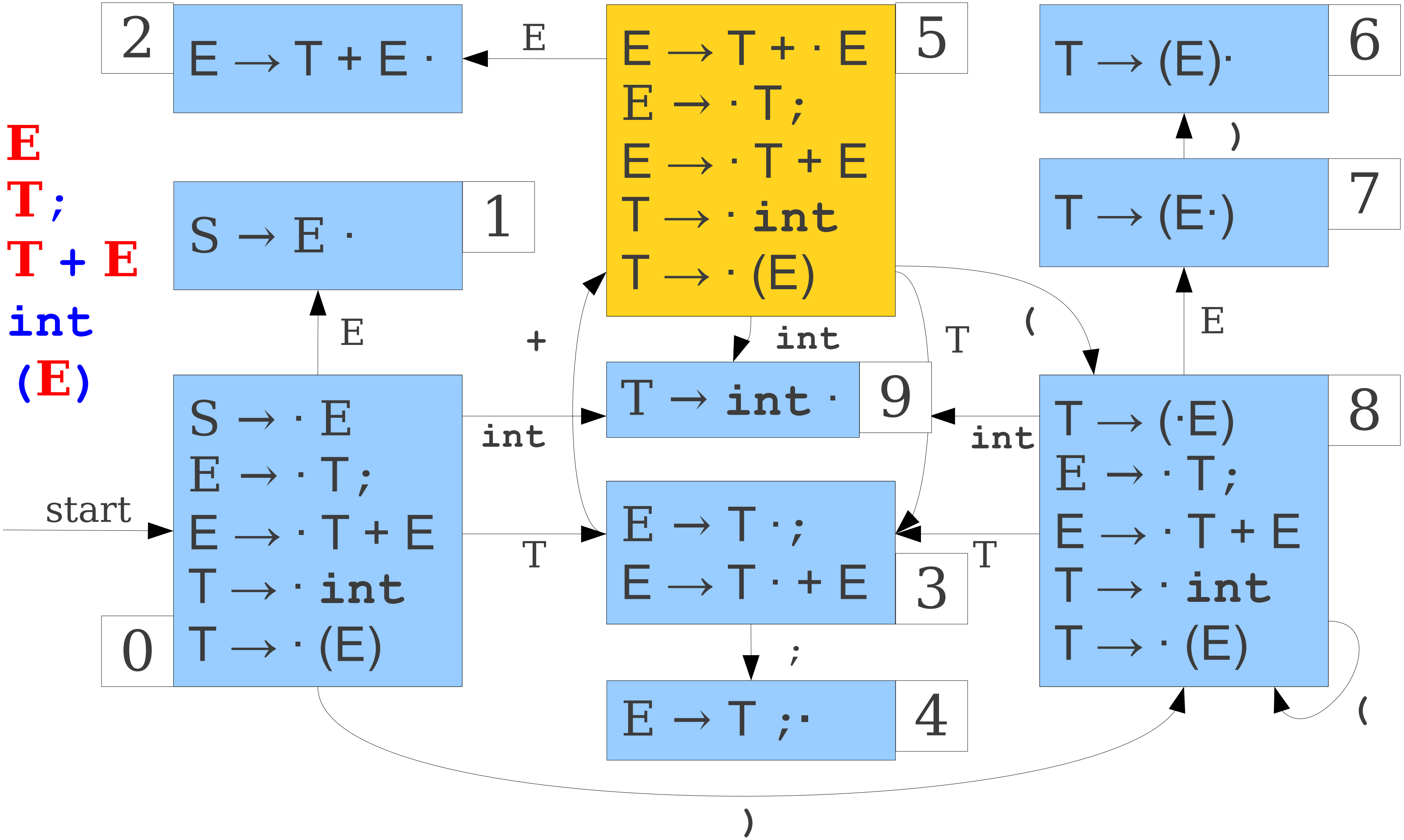


T	+	(	T	+	int	|	;	)	;

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)·

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)

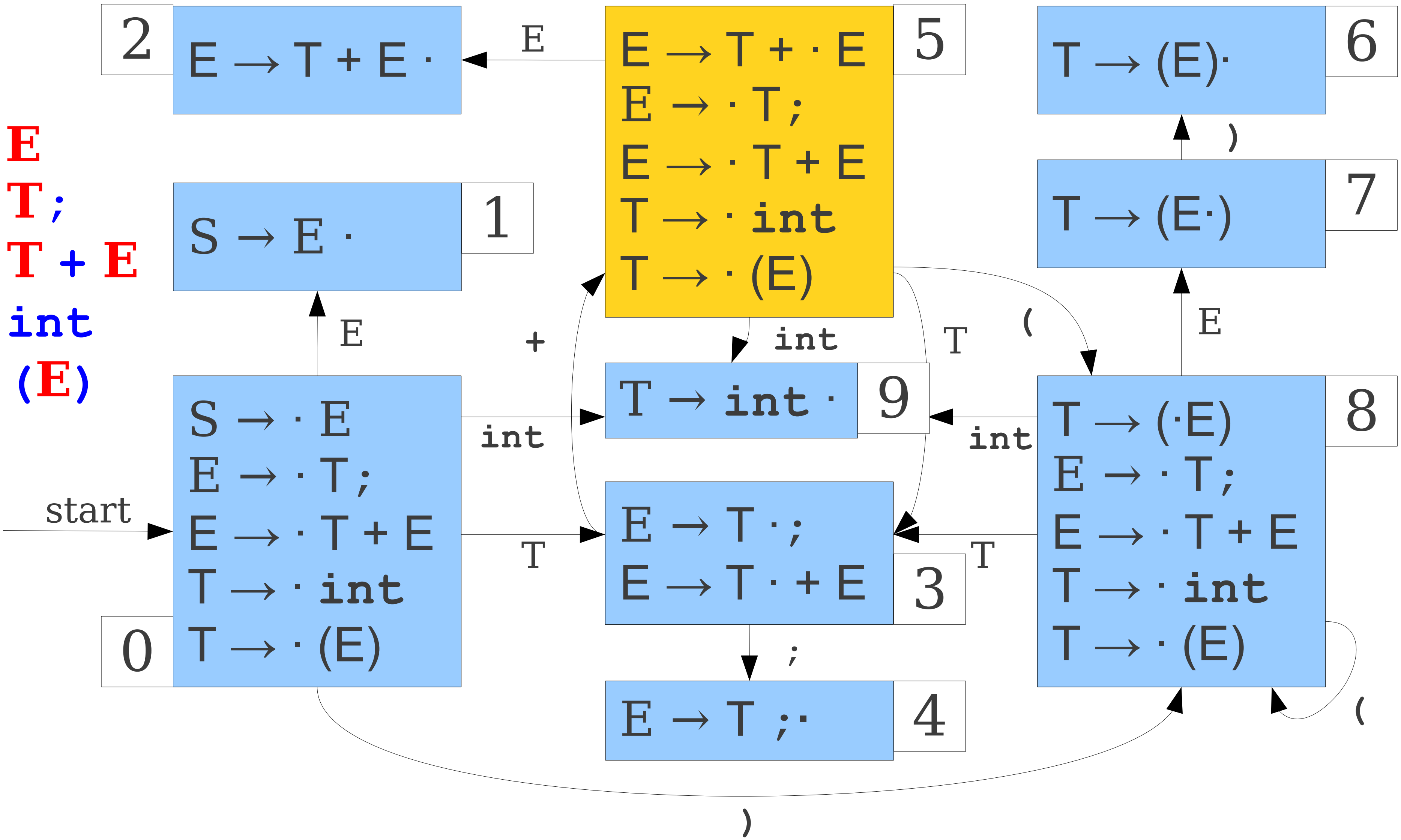
4
E → T ;·

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

;	)	;

LR(0) Parsing

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



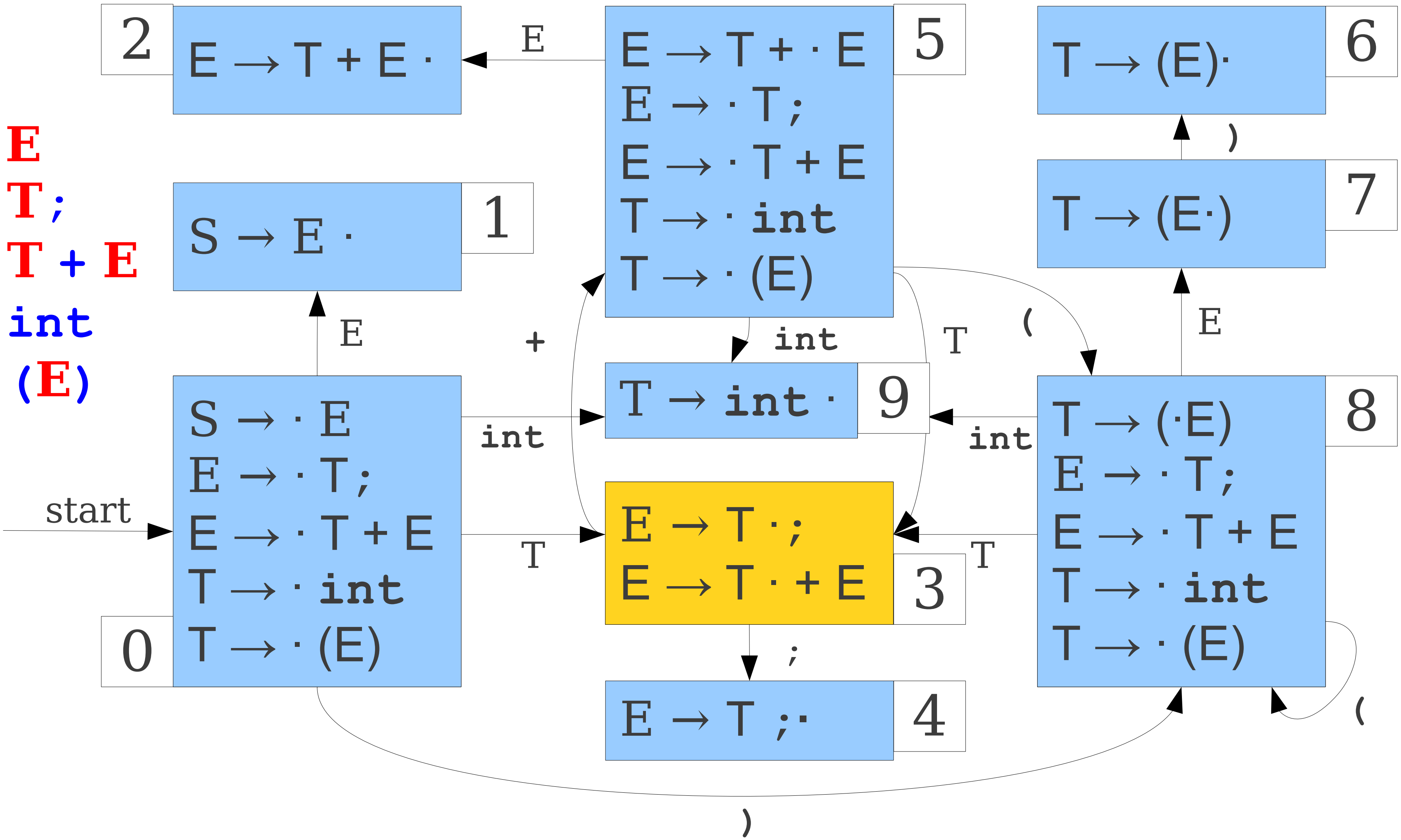
T	+	(	T	+	T

;	)	;

0	**3**	**5**	**8**	**3**	**5**

LR(0) Parsing

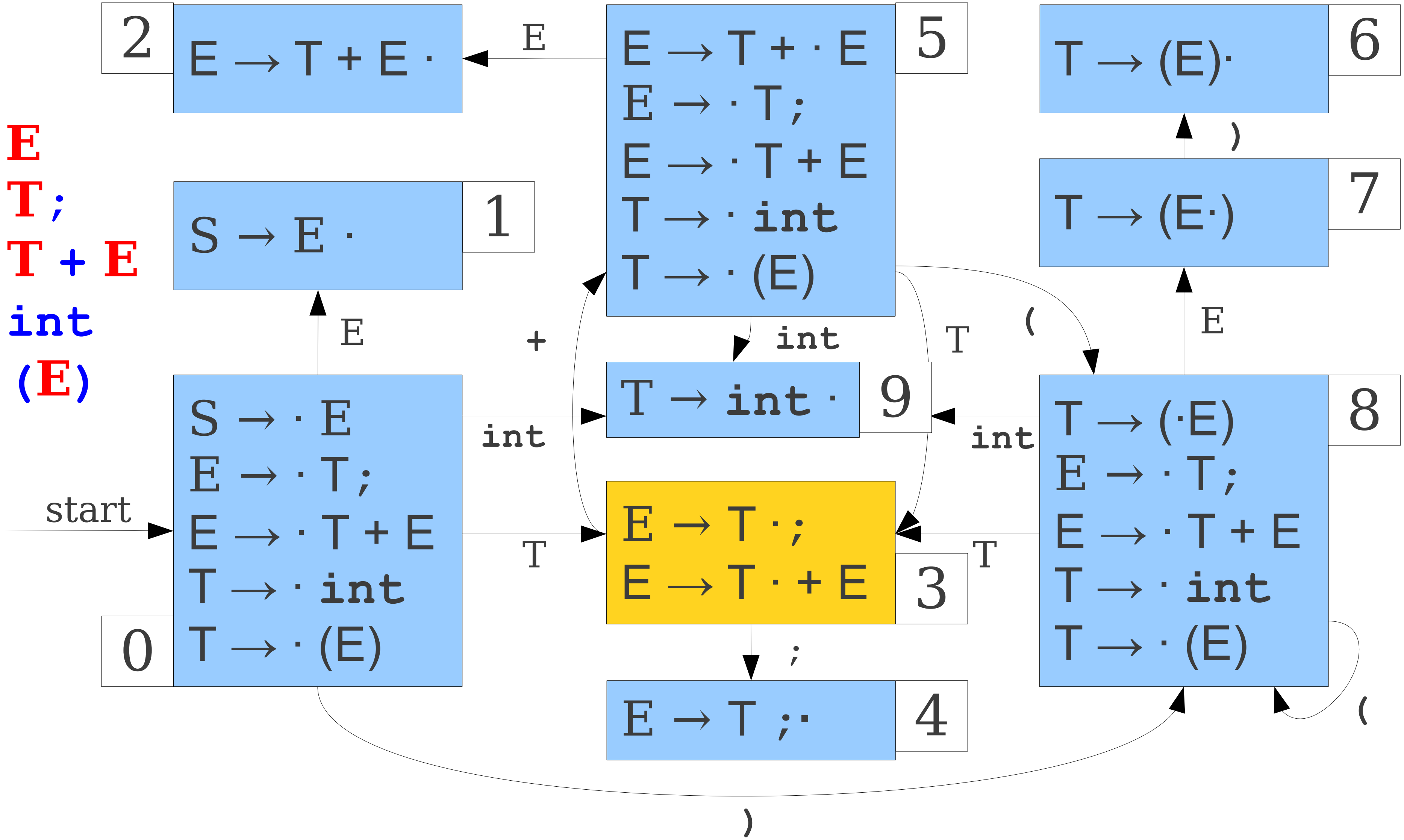
S → **E**
E → **T** `;`
E → **T** + **E**
T → `int`
T → `(`**E**`)`



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

LR(0) Parsing

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

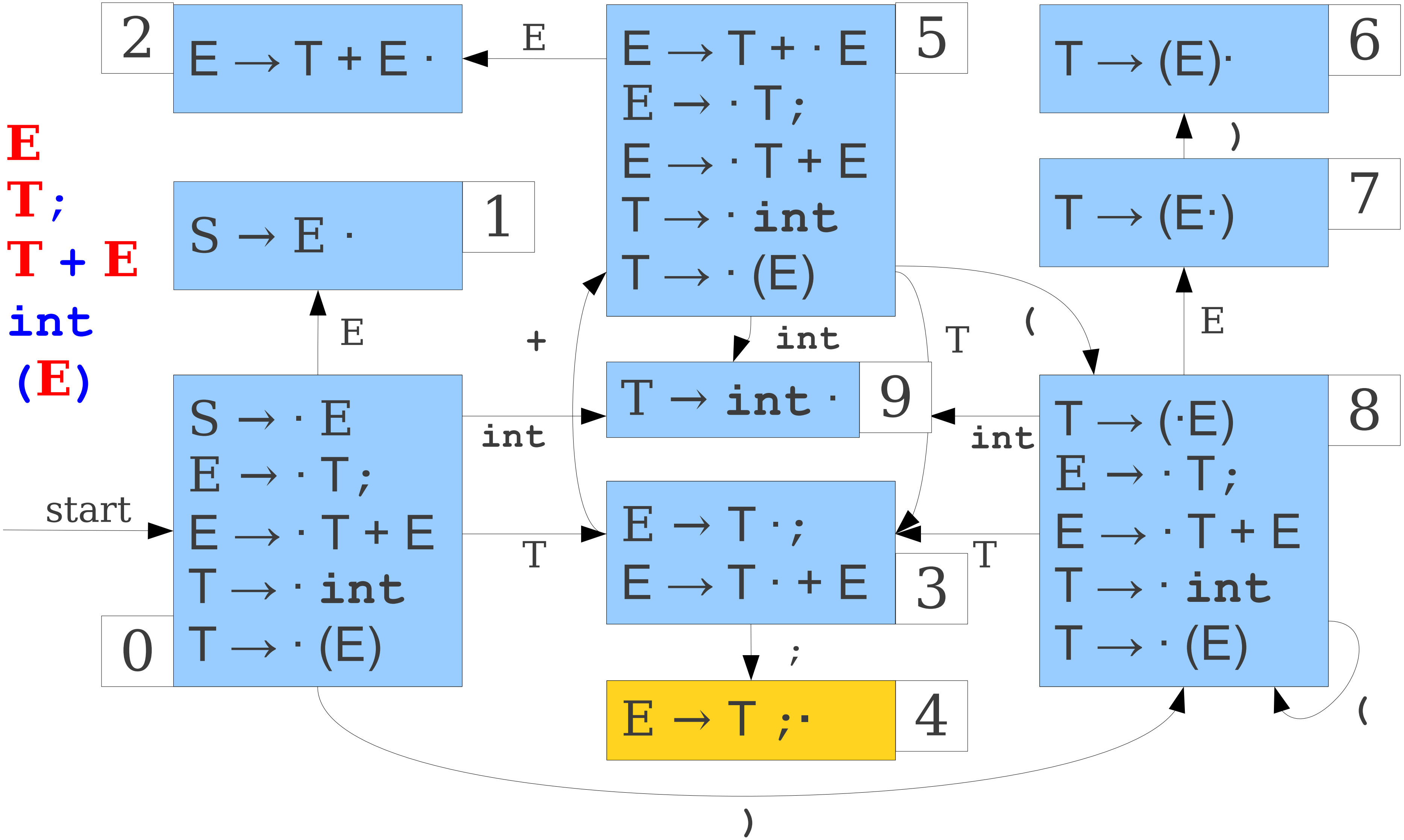


T	+	(	T	+	T	;	)	;

0 3 5 8 3 5 3

LR(0) Parsing

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

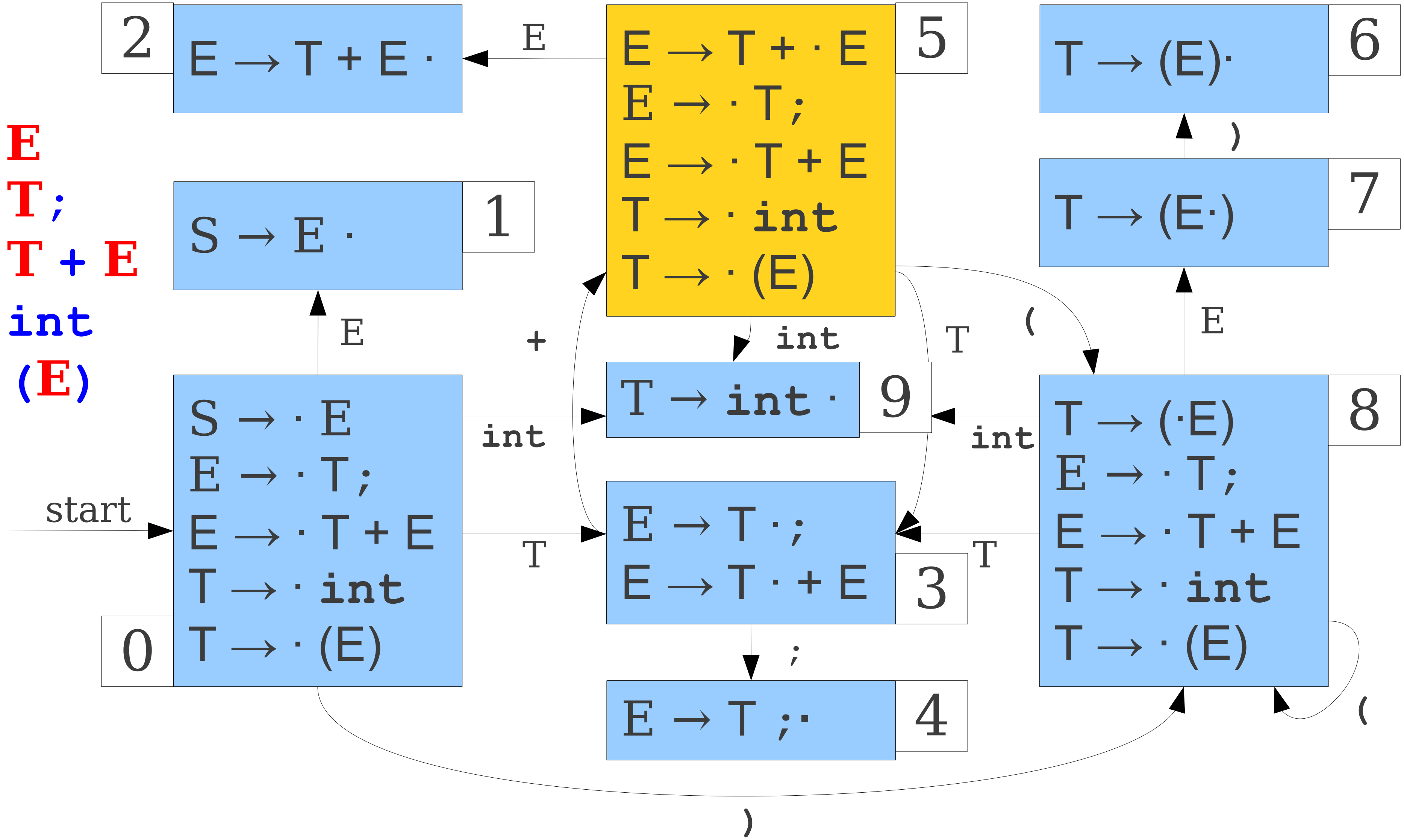


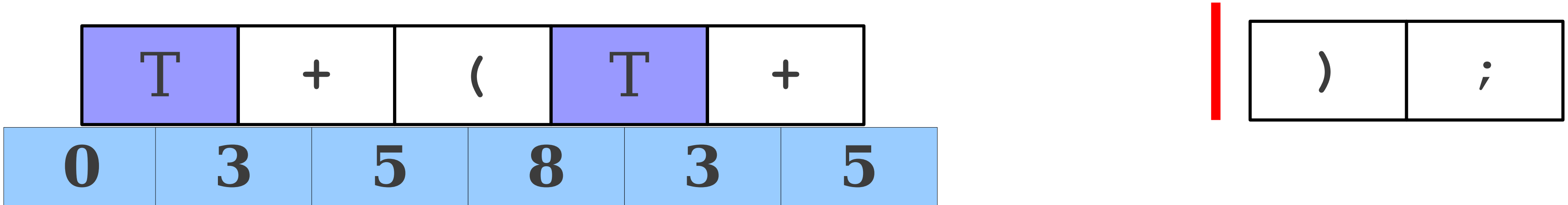
T	+	(	T	+	T	;
0	3	5	8	3	5	3

)	;

LR(0) Parsing

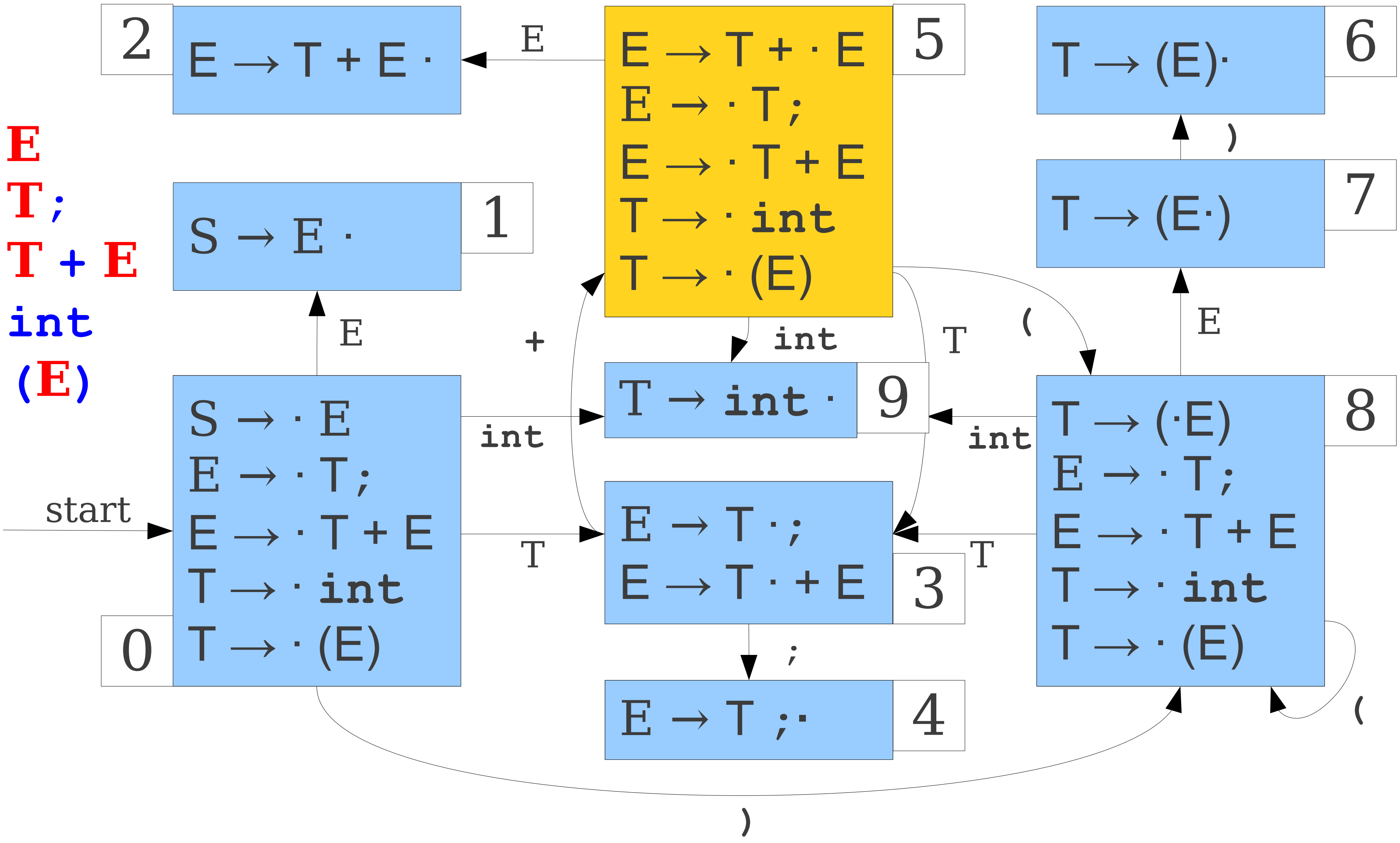
S → **E**
E → **T**;
E → **T** + **E**
T → `int`
T → (**E**)





LR(0) Parsing

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



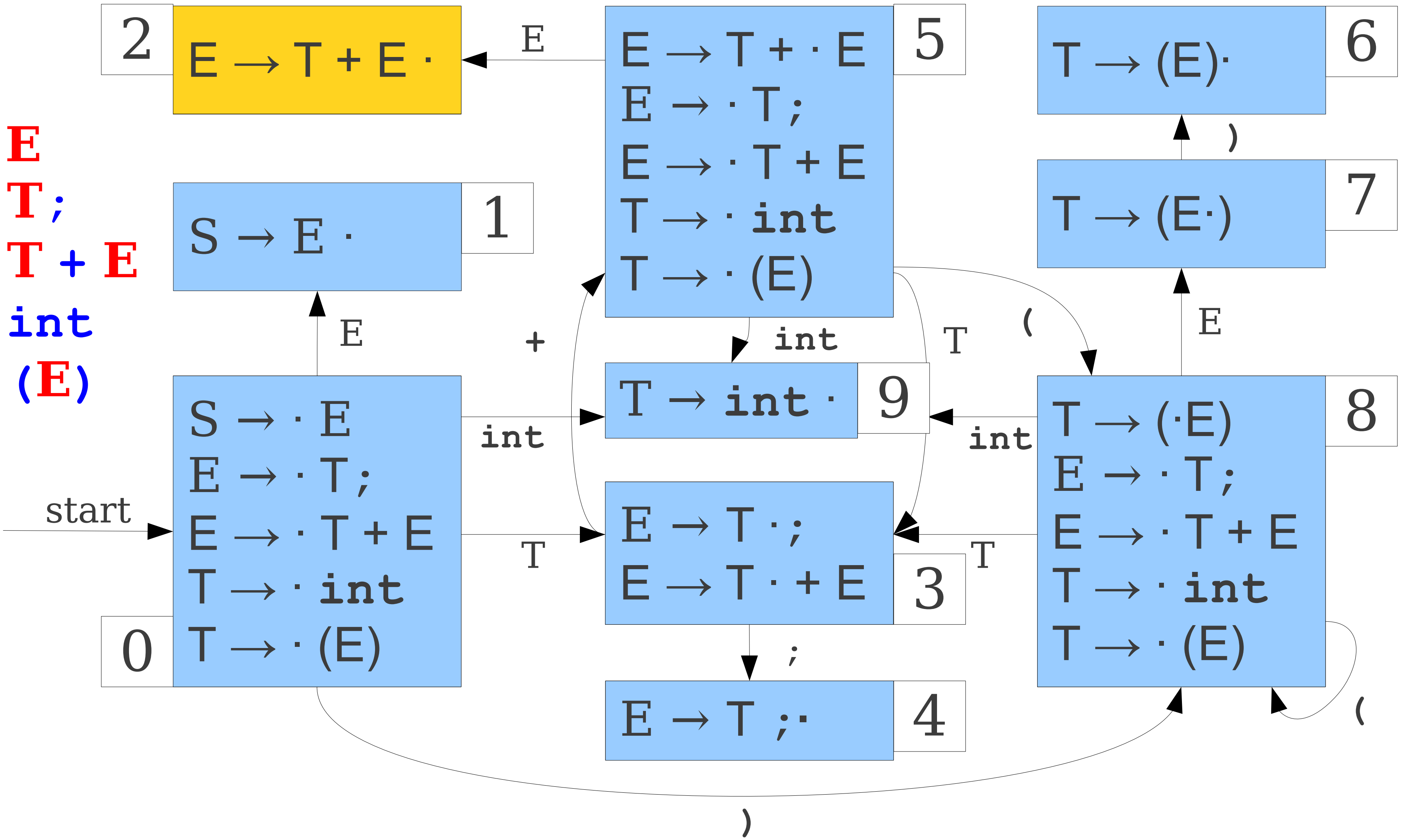
T	+	(	T	+	E

)	;

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)

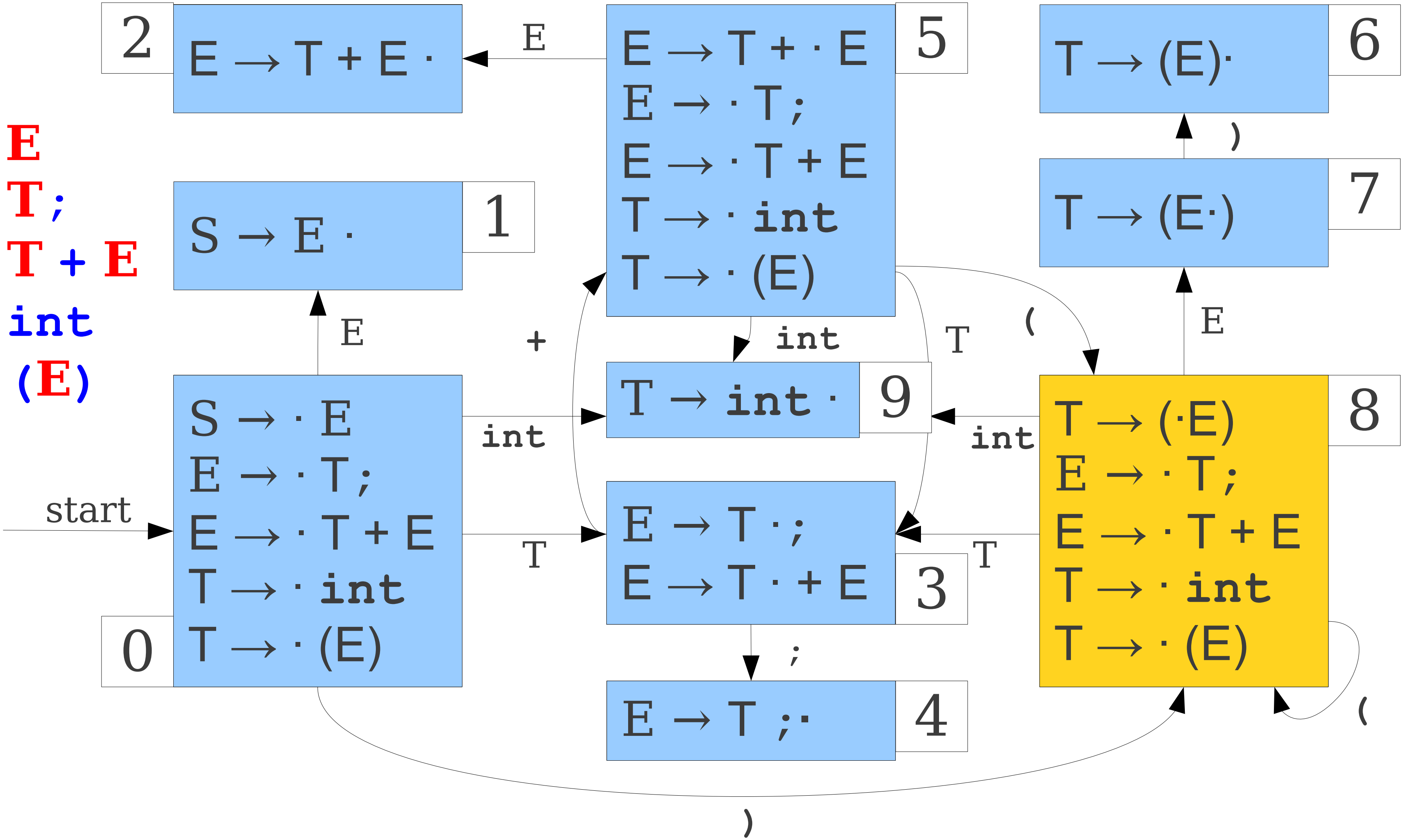
9
T → `int` ·

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

)	;

LR(0) Parsing

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



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 ;·

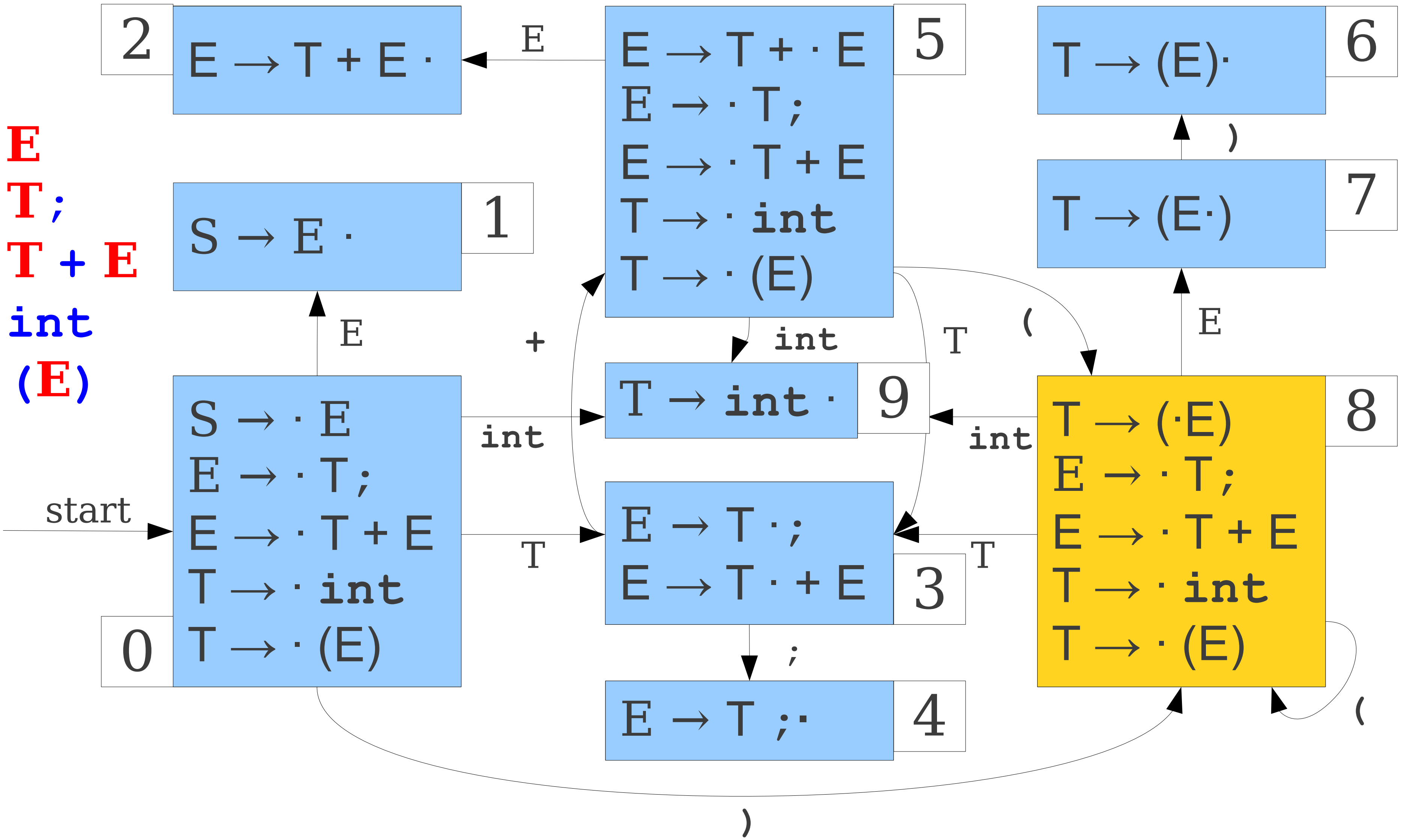
T	+	(

0	**3**	**5**	**8**

)	;

LR(0) Parsing

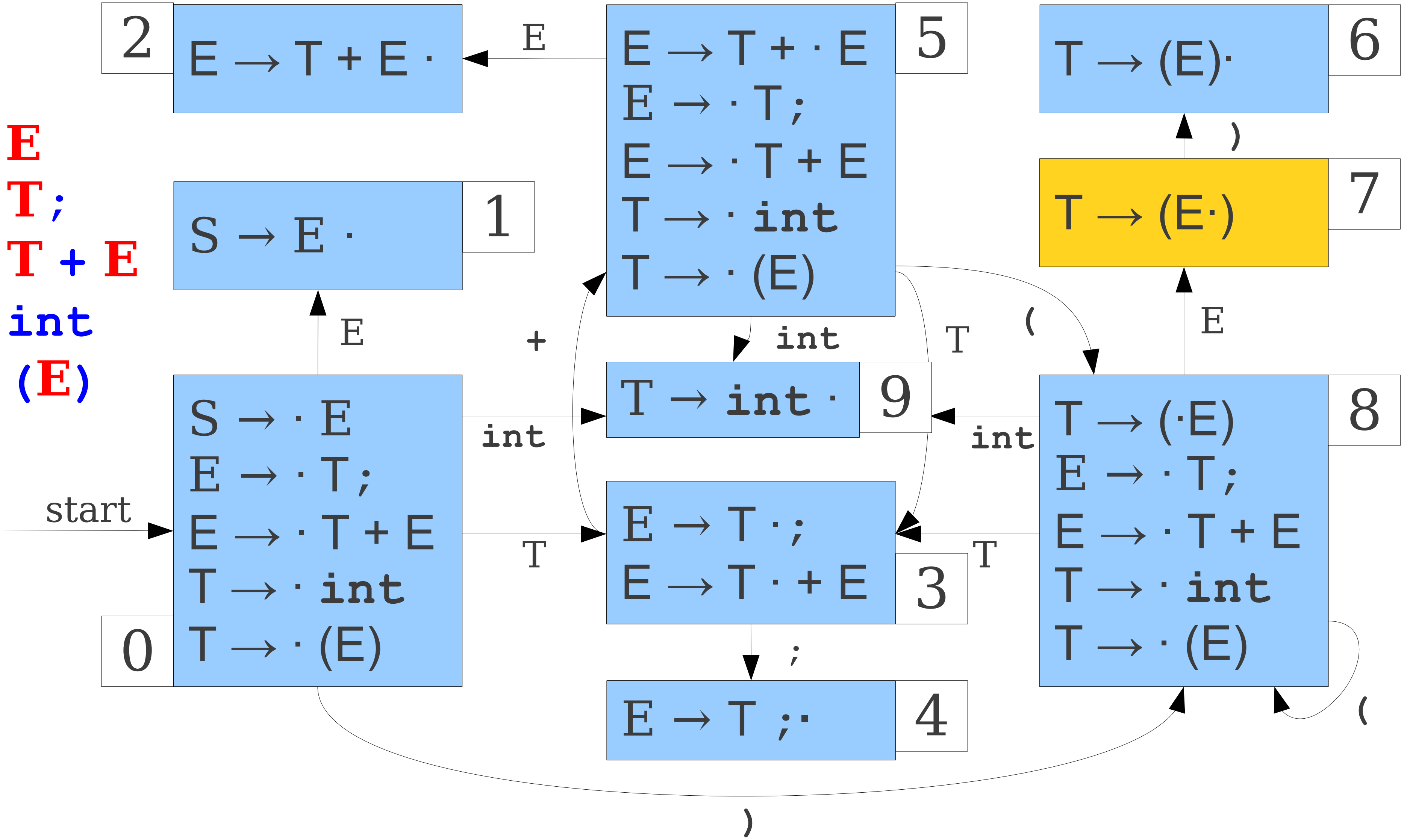
S → **E**
E → **T** ;
E → **T** + **E**
T → `int`
T → **(E)**



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)

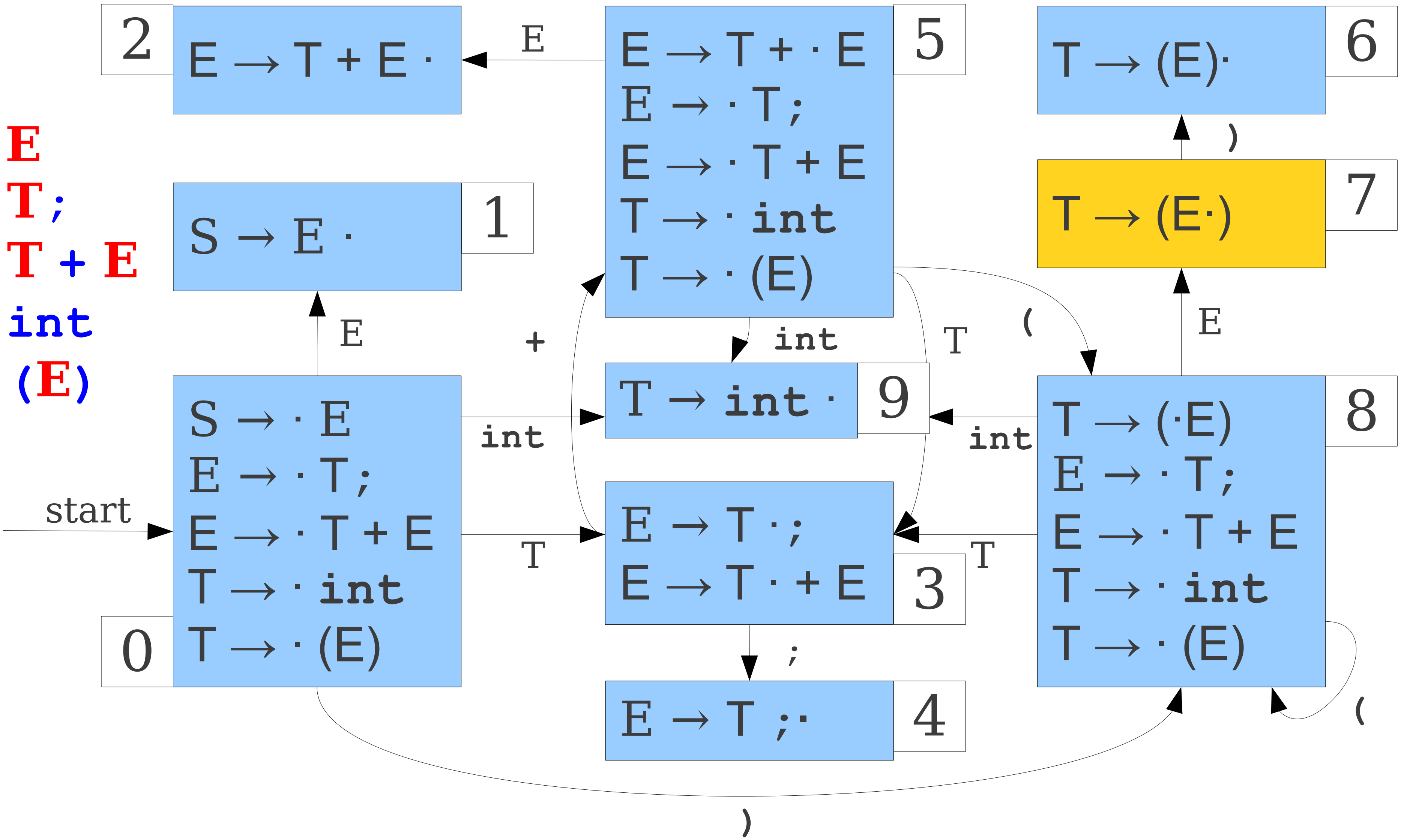
9
T → `int` ·

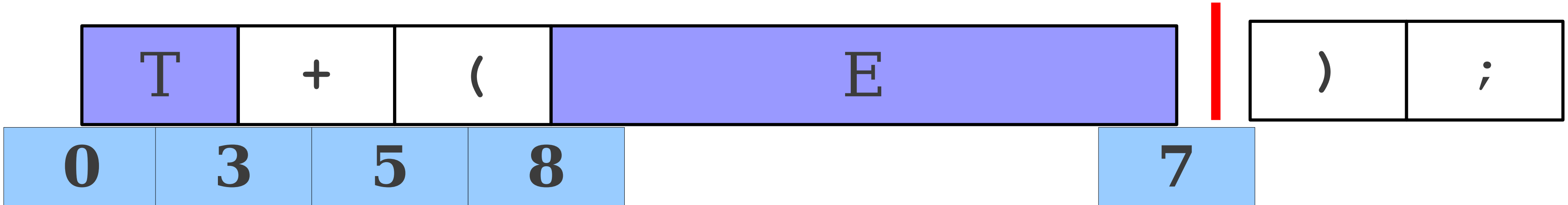
T	+	(	E	)	;

0	**3**	**5**	**8**

LR(0) Parsing

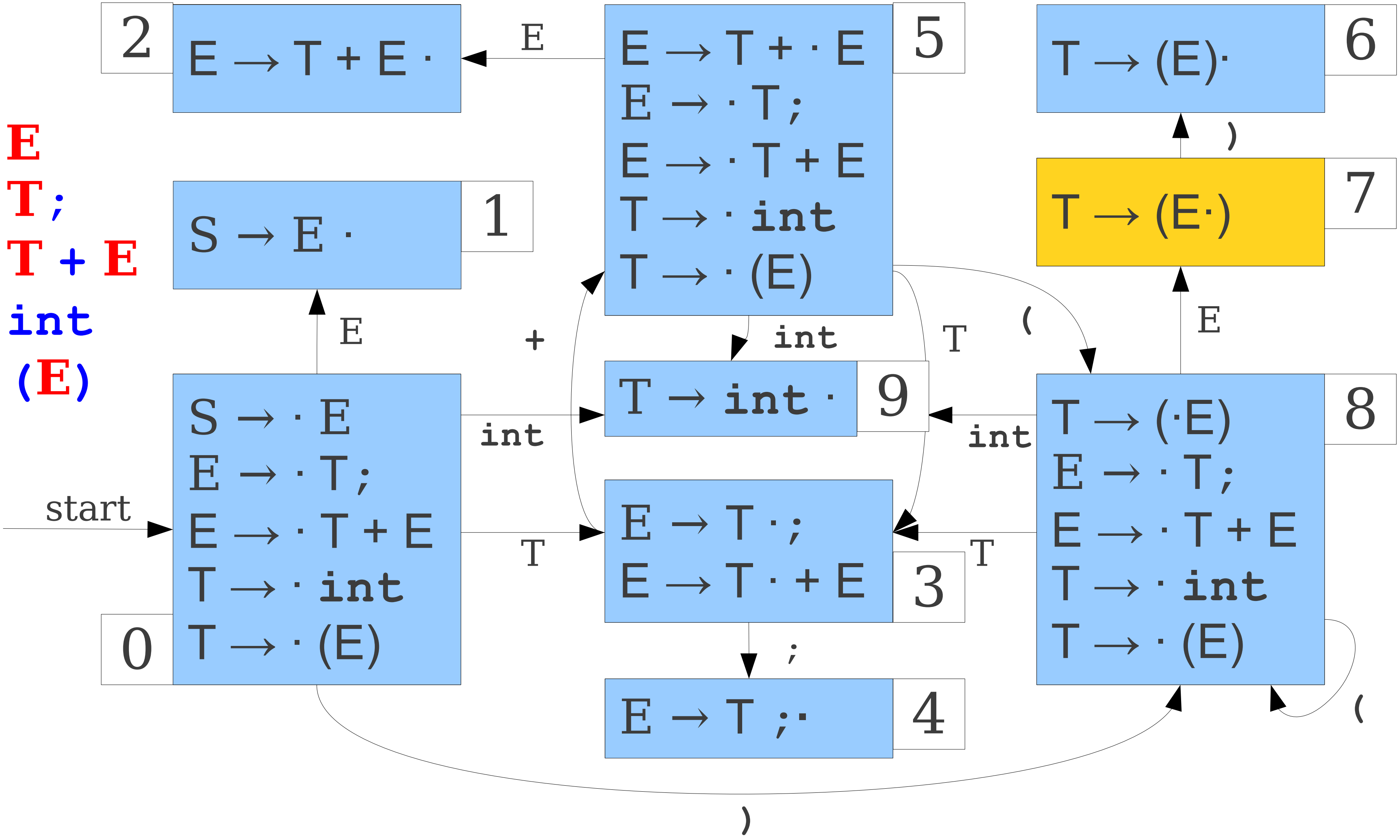
S → **E**
E → **T** ;
E → **T** + **E**
T → **int**
T → **(E)**

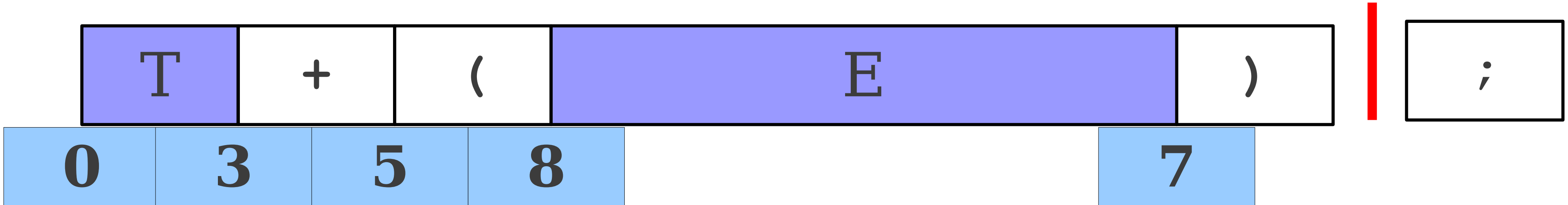




LR(0) Parsing

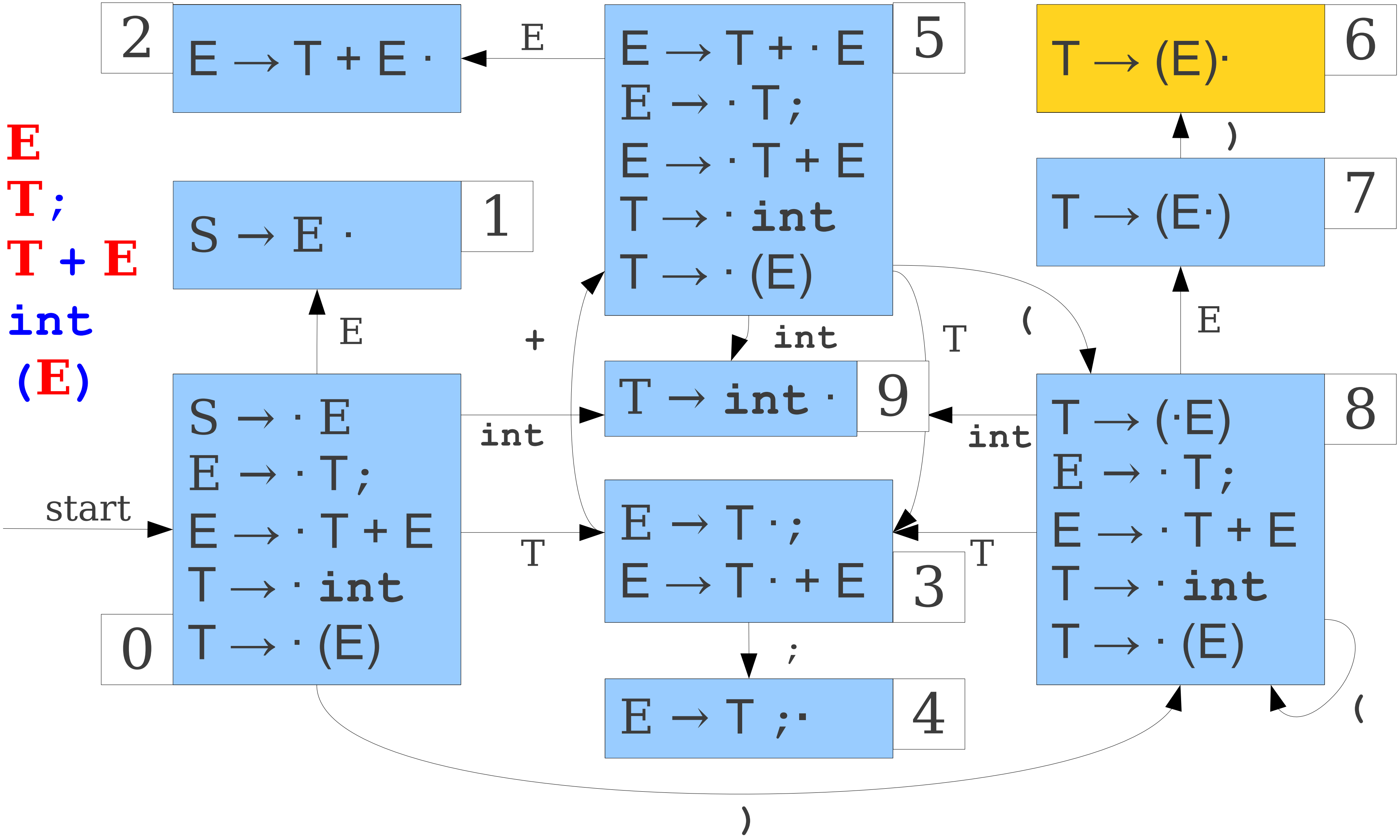
S → **E**
E → **T** ;
E → **T** + **E**
T → `int`
T → (**E**)





LR(0) Parsing

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



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 ; ·

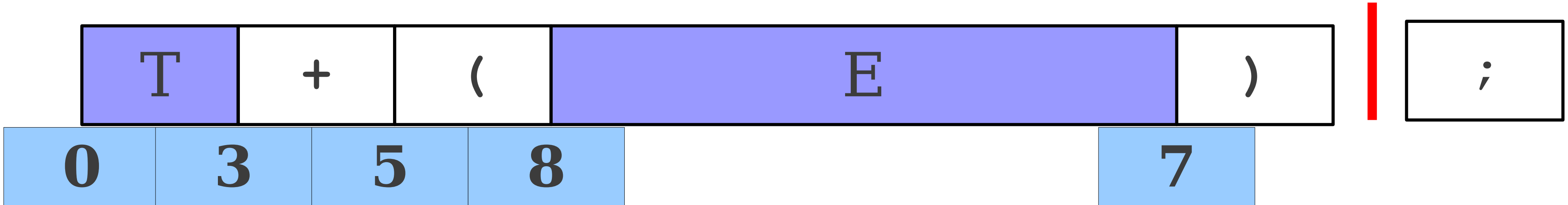
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6
T → (E)·

7
T → (E·)

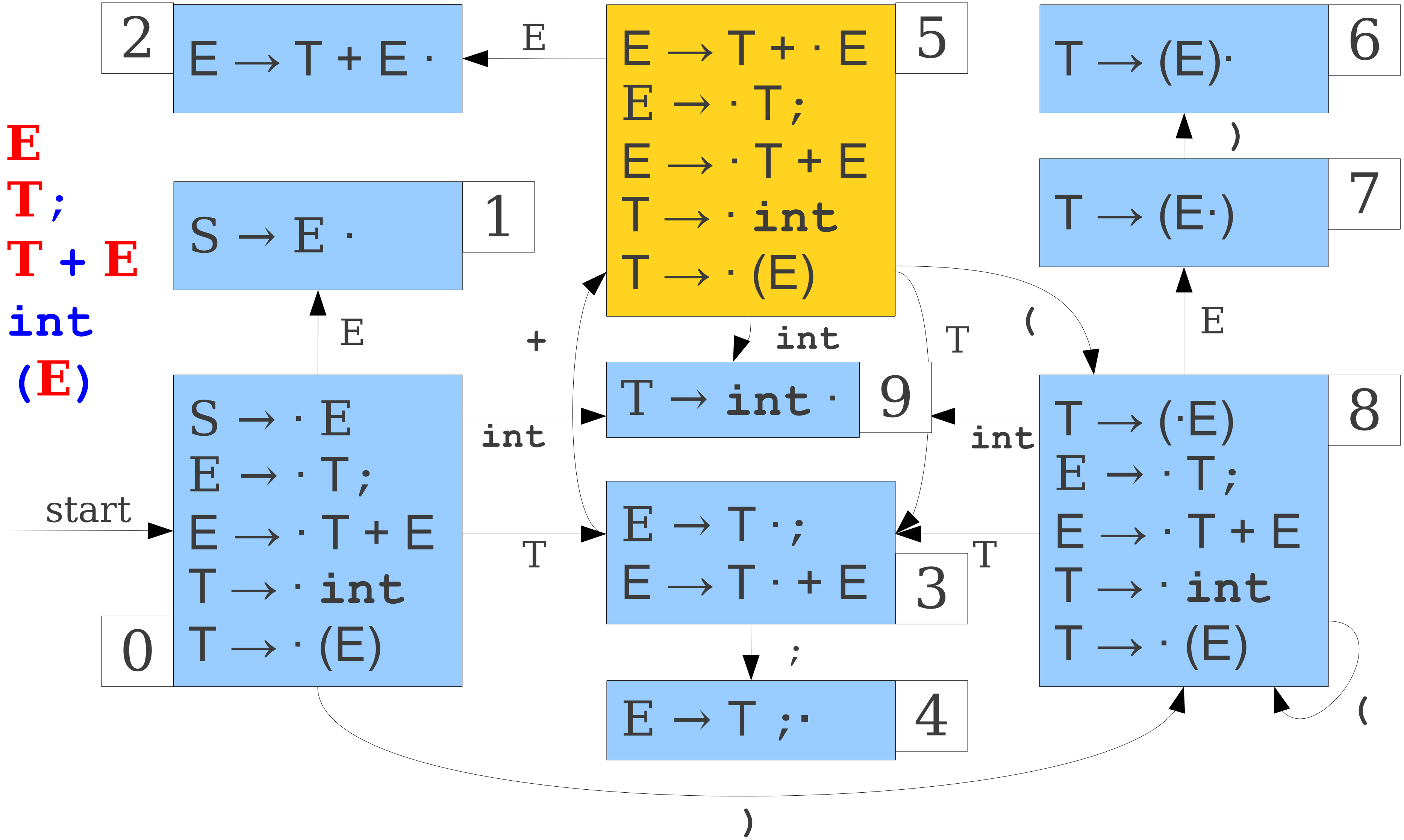
8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

9
T → int ·



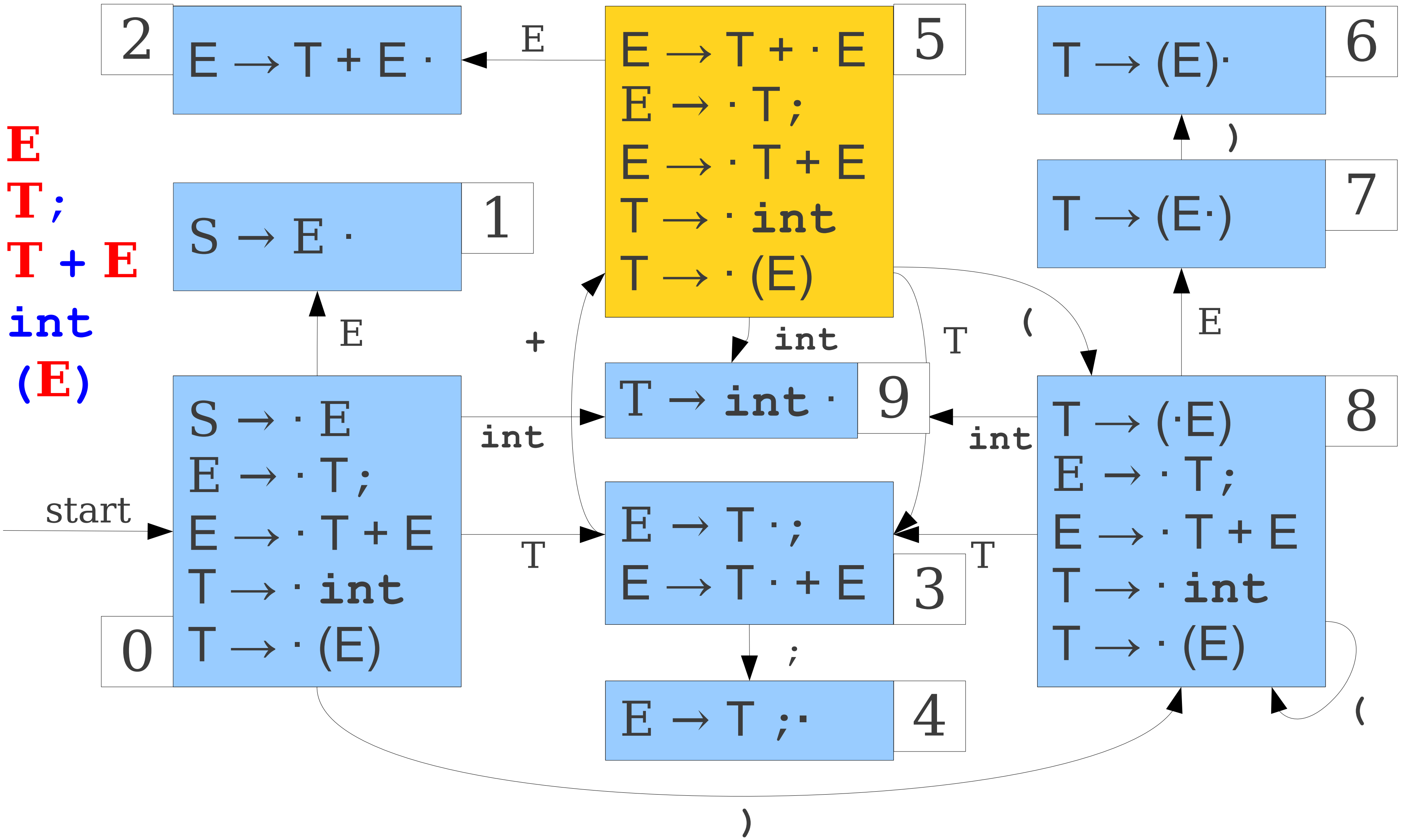
LR(0) Parsing

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



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)

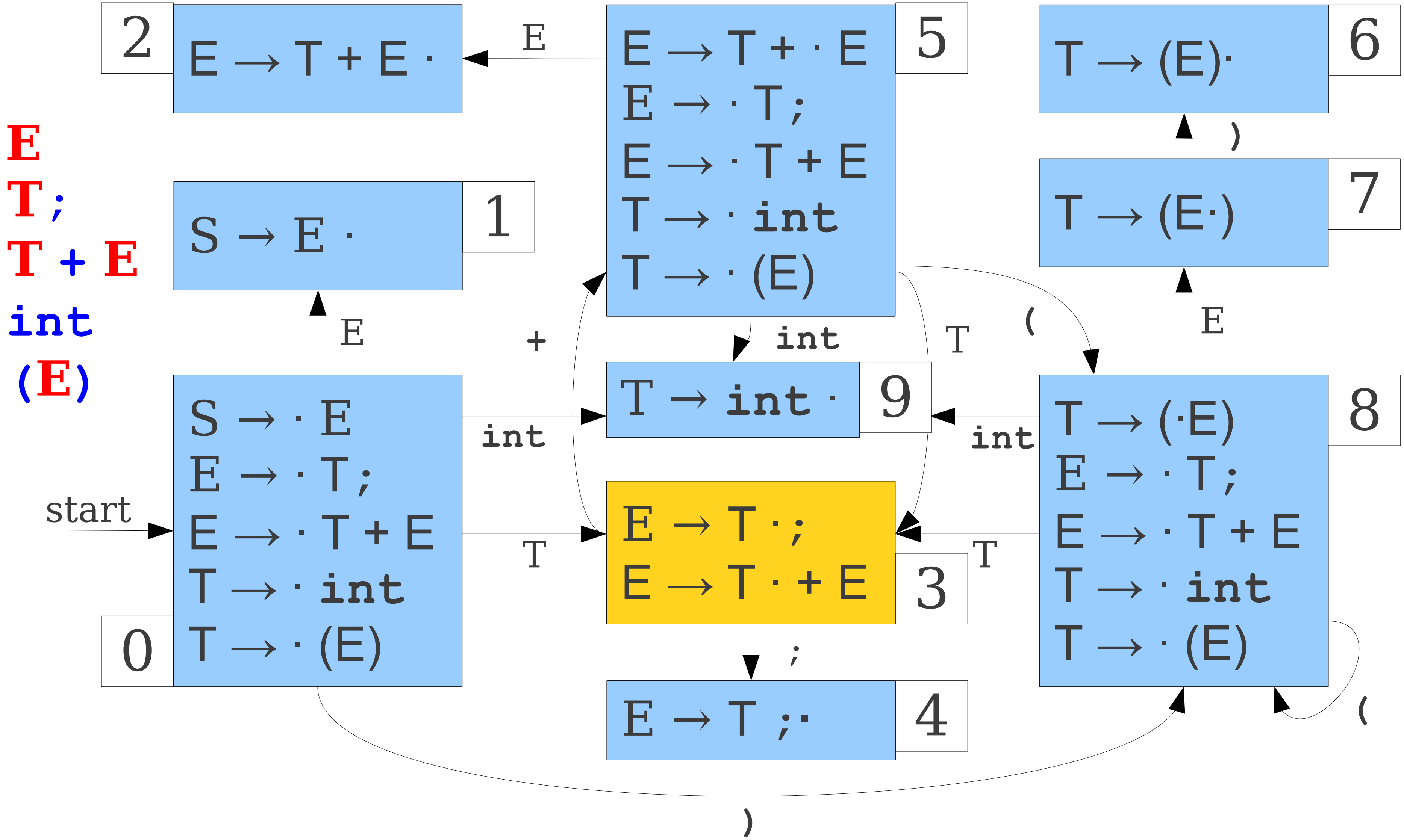
4
E → T ;·

T	+	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

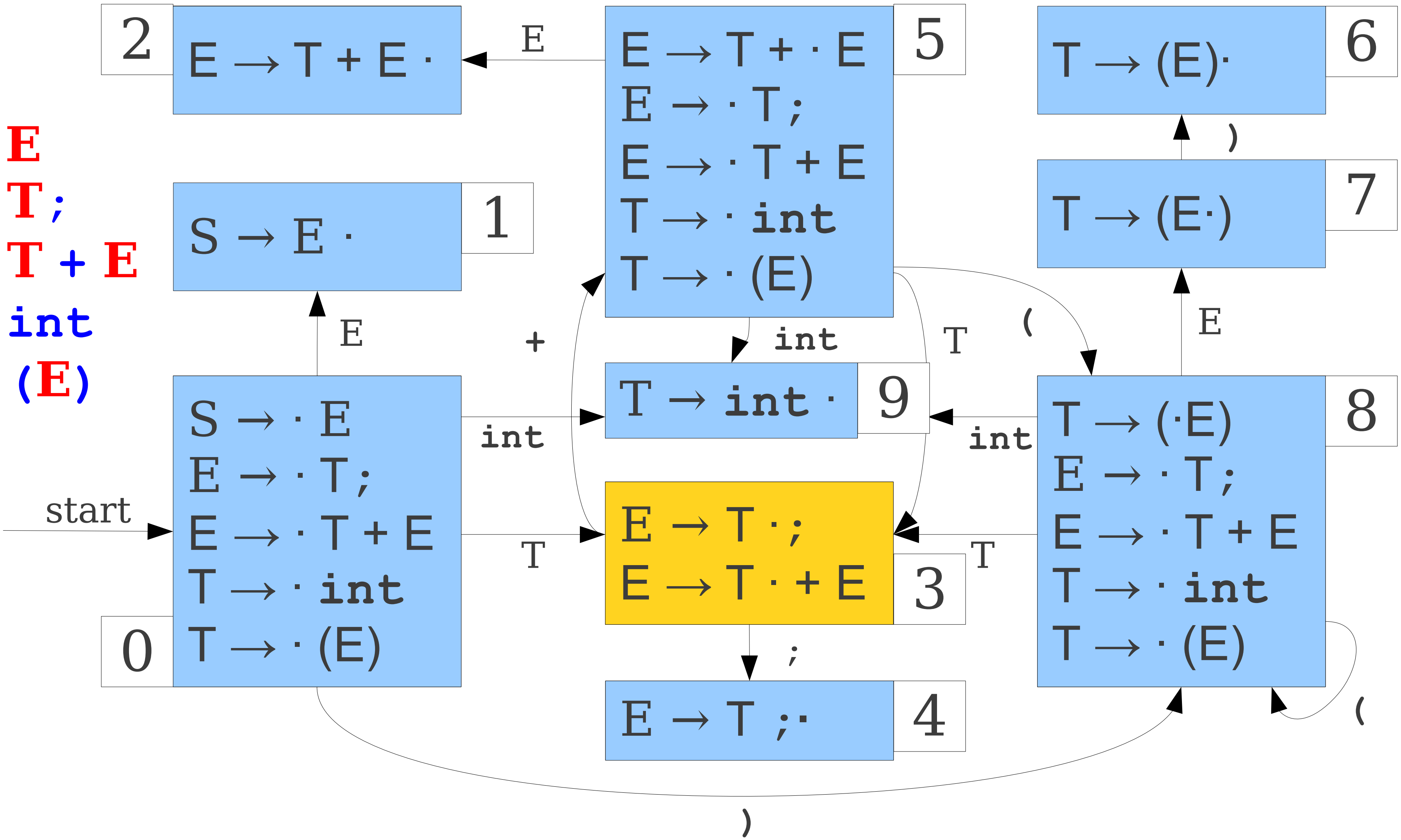
8
T → (·E)
E → · T;
E → · T + E
T → · `int`
T → · (E)

4
E → T ;·

T	+	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)

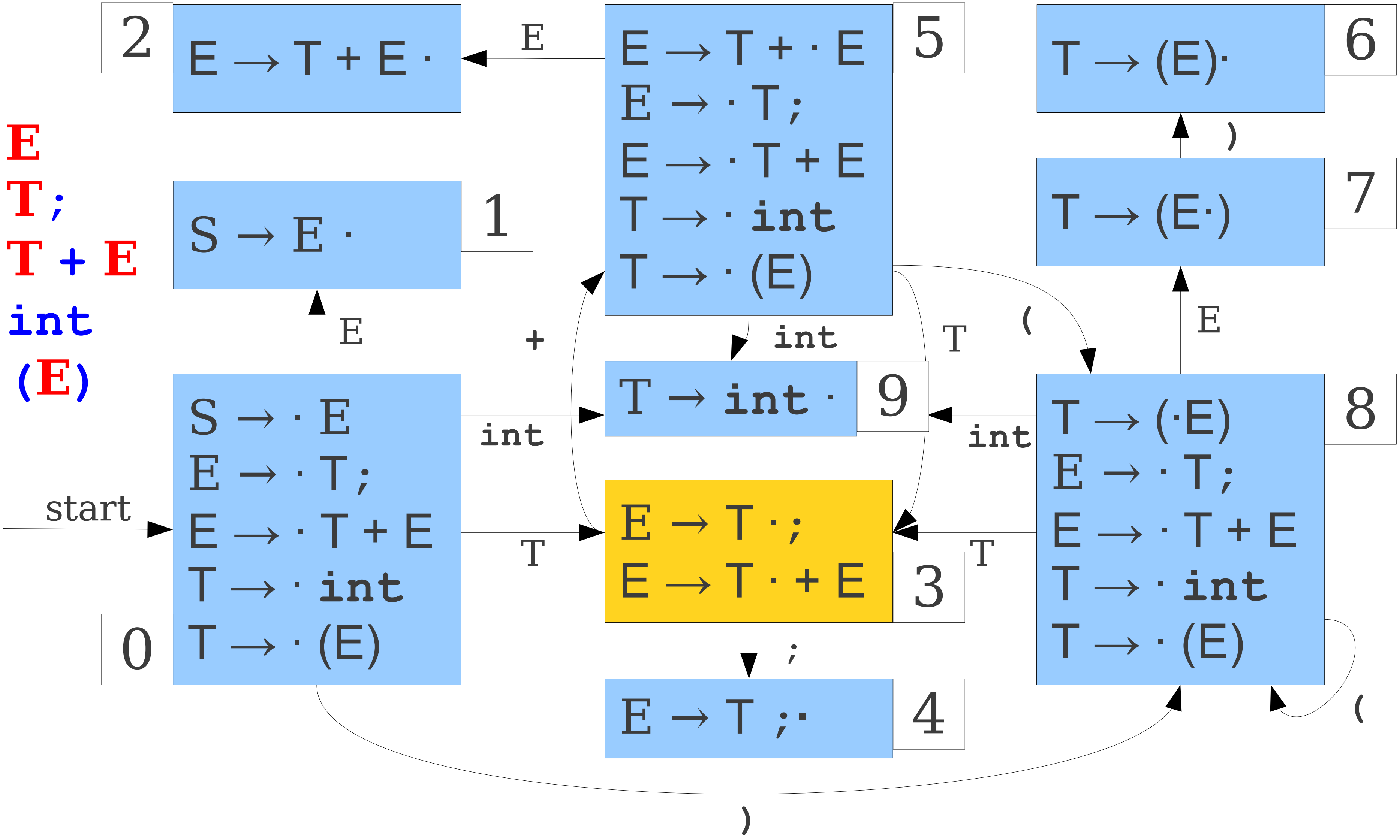
4: E → T ;·

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)

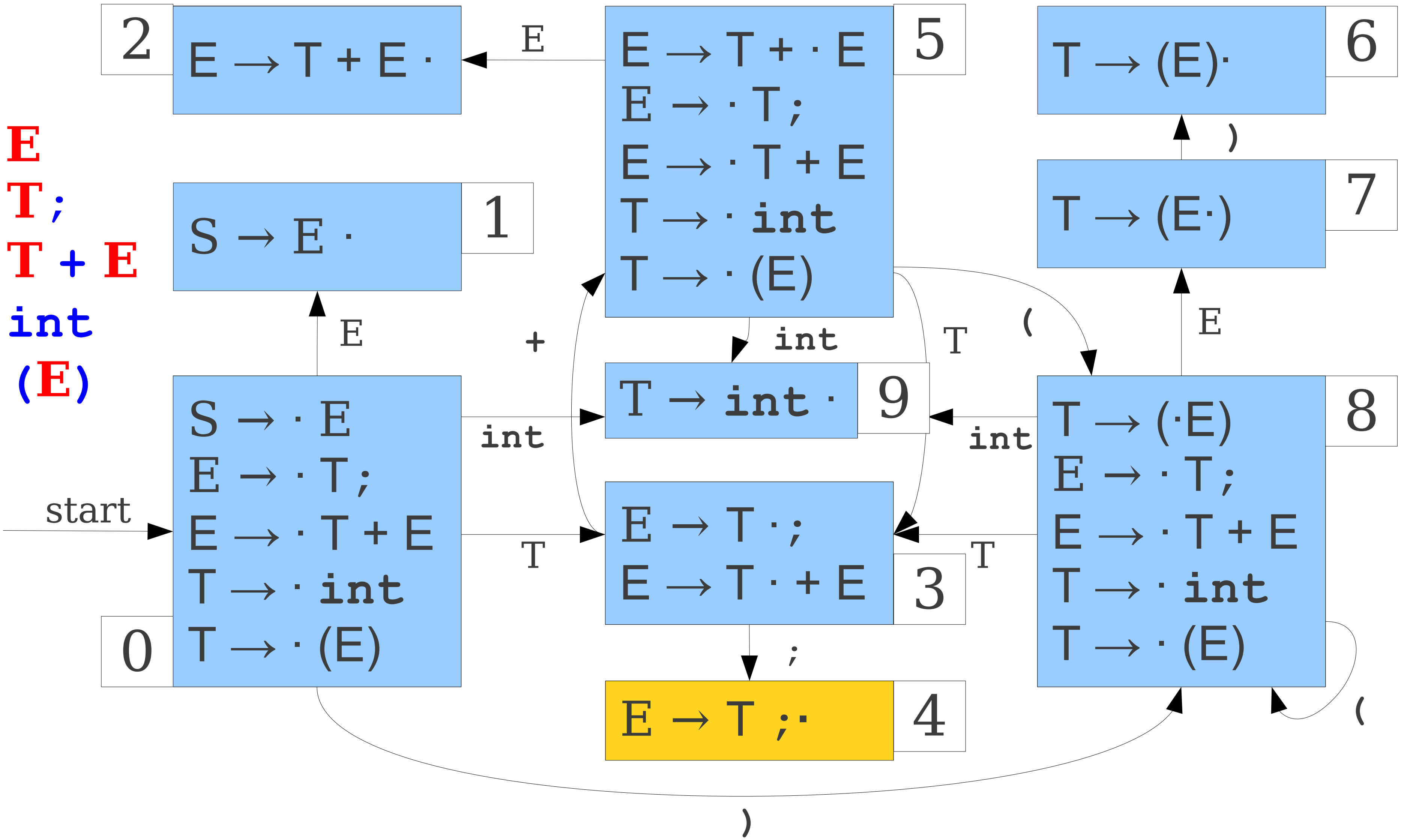
9
T → `int` ·

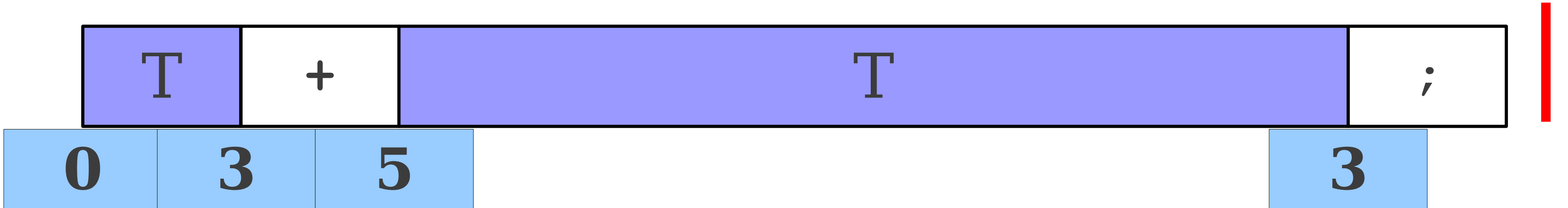
T | + | T | ;

0 3 5 3

LR(0) Parsing

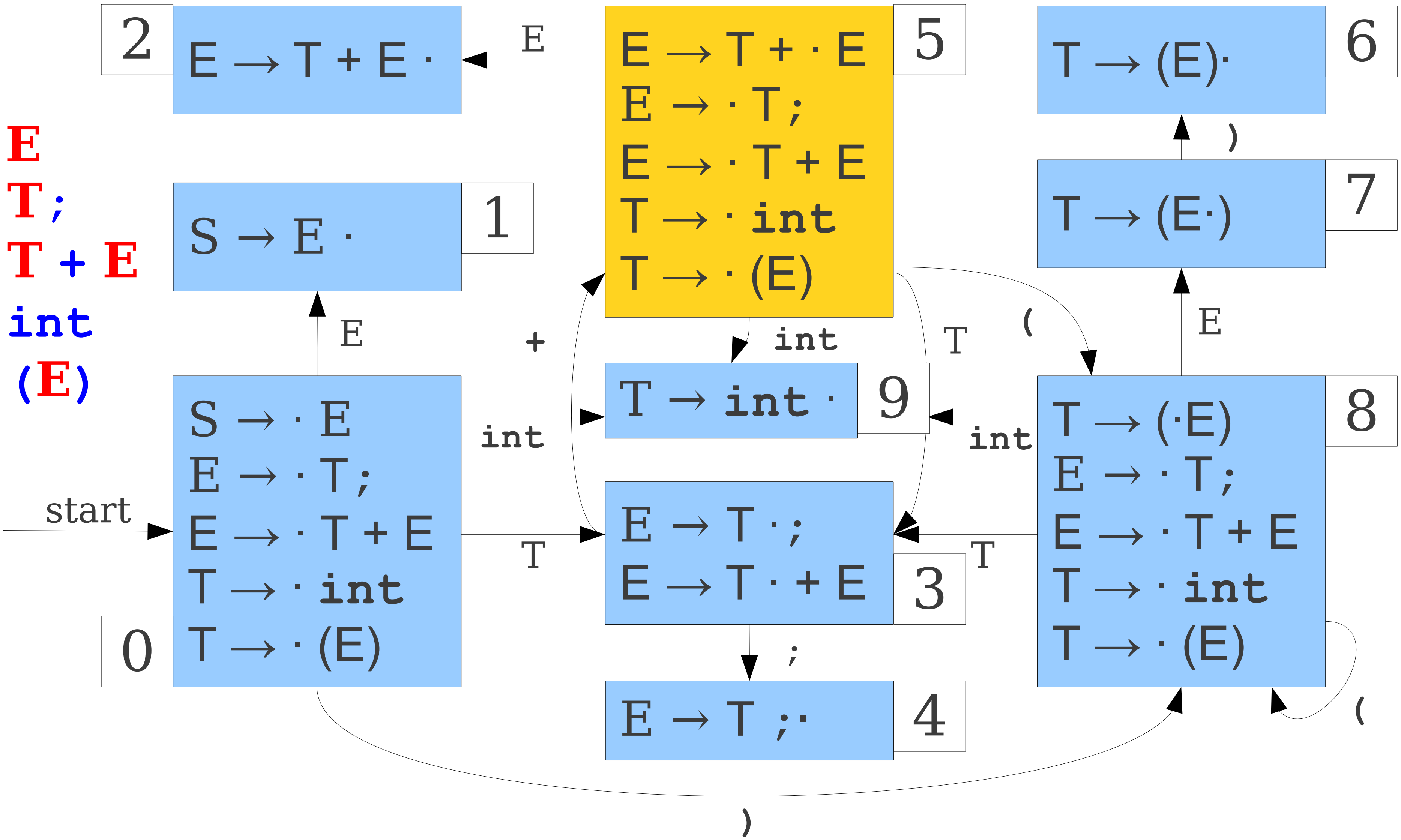
S → **E**
E → **T** ;
E → **T** + **E**
T → **int**
T → **(E)**





LR(0) Parsing

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



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·)

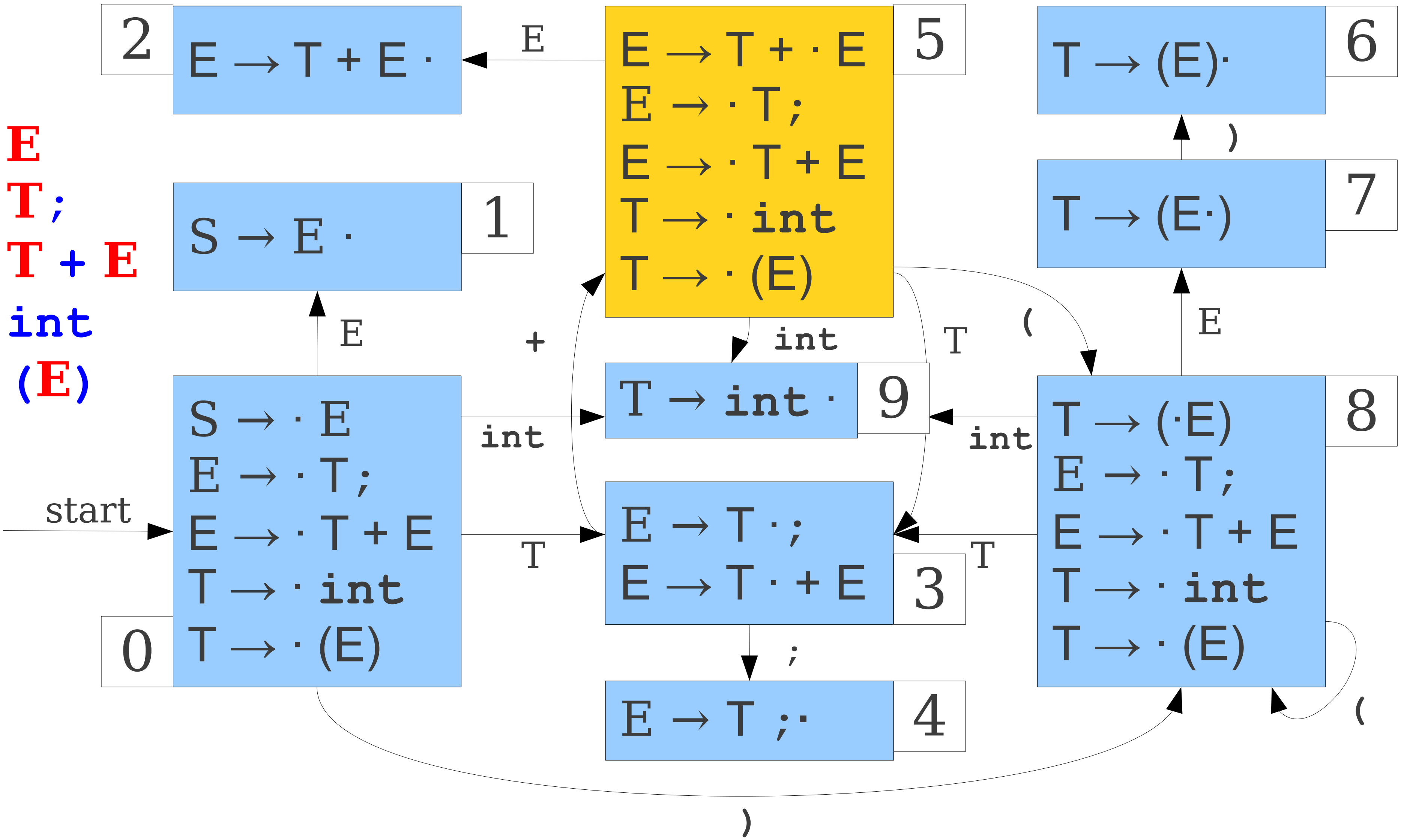
8
T → (·E)
E → · T;
E → · T + E
T → · `int`
T → · (E)

9
T → `int` ·

T	+	
0	3	5

LR(0) Parsing

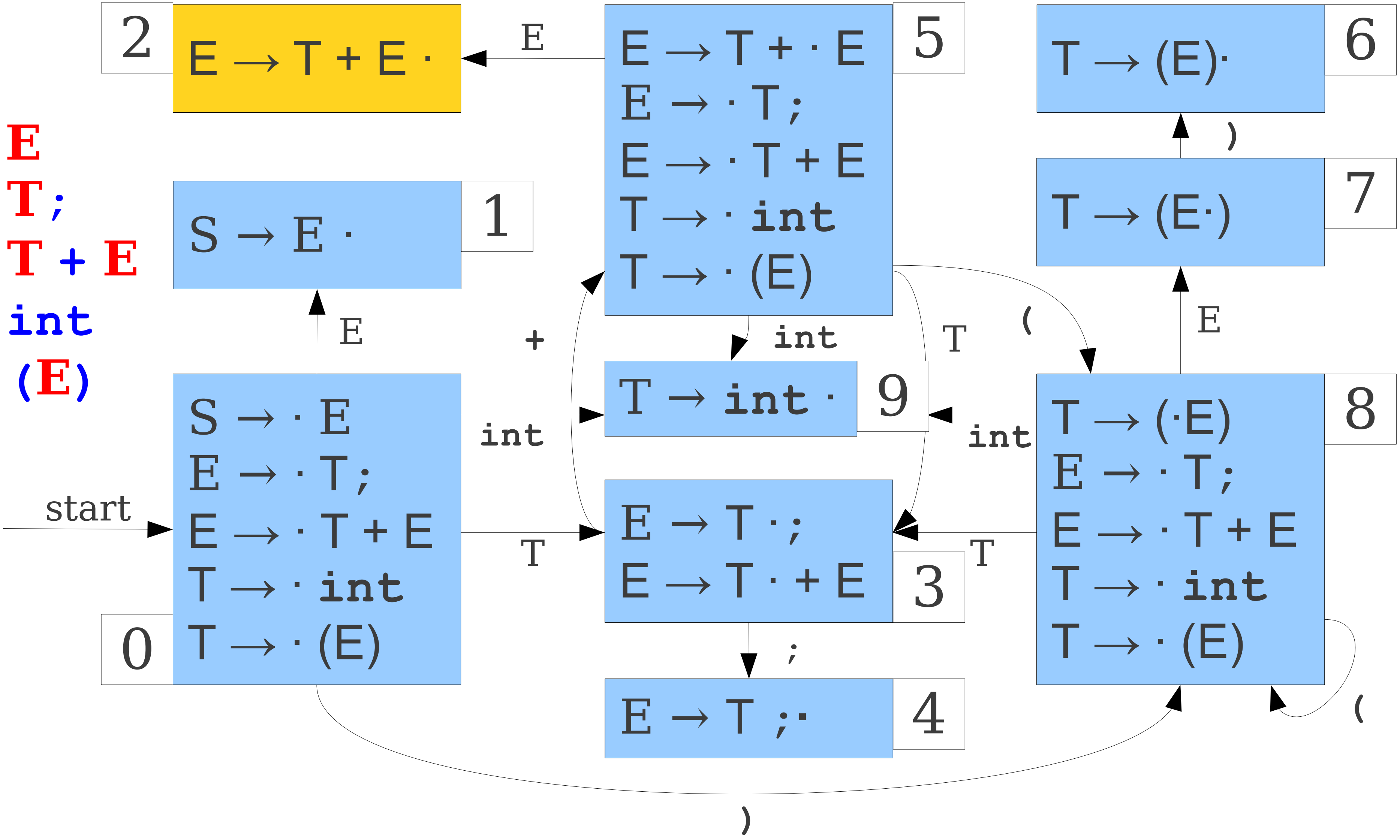
S → **E**
E → **T**;
E → **T** + **E**
T → `int`
T → (**E**)



T	+	E
0	3	5

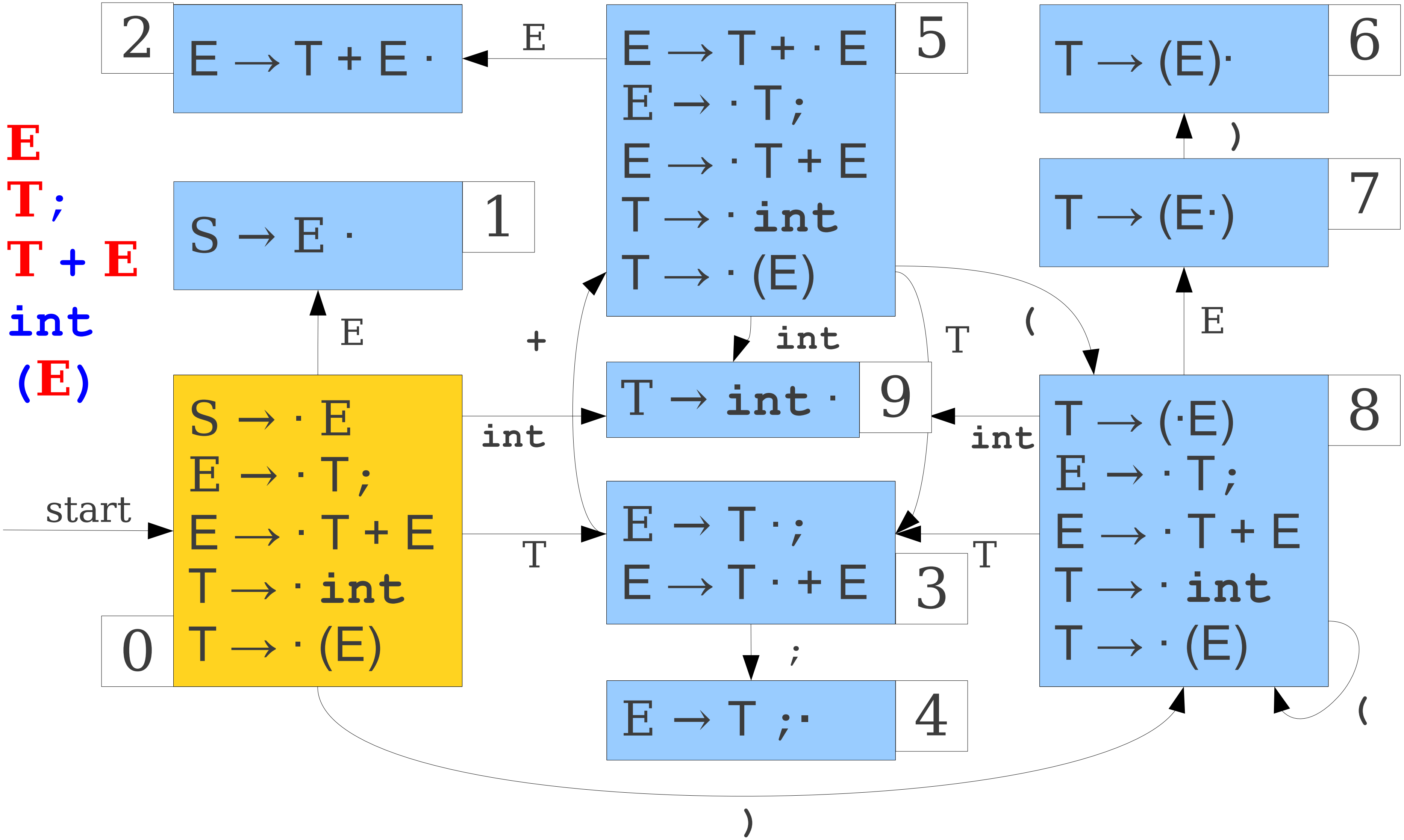
LR(0) Parsing

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**`)`



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·)

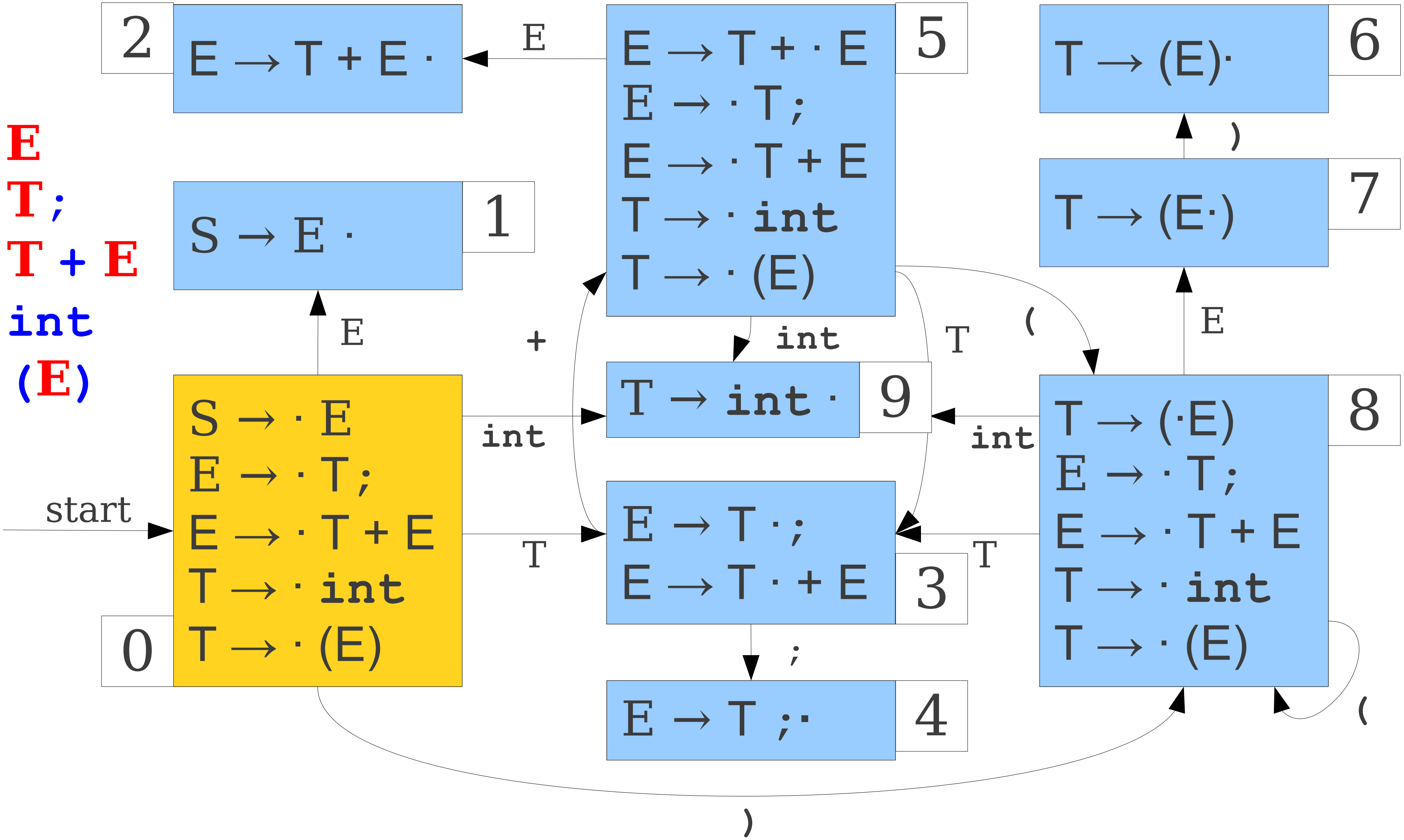
8
T → (·E)
E → · T;
E → · T + E
T → · `int`
T → · (E)

9
T → `int` ·

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 ;·

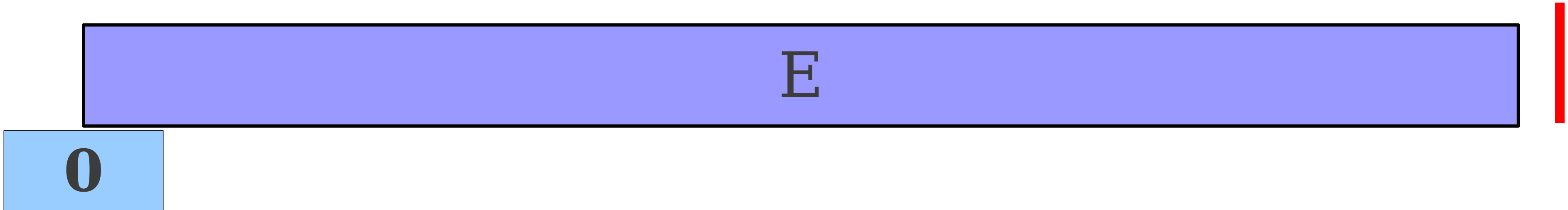
5
E → T + · E
E → · T;
E → · T + E
T → · int
T → · (E)

6
T → (E)·

7
T → (E·)

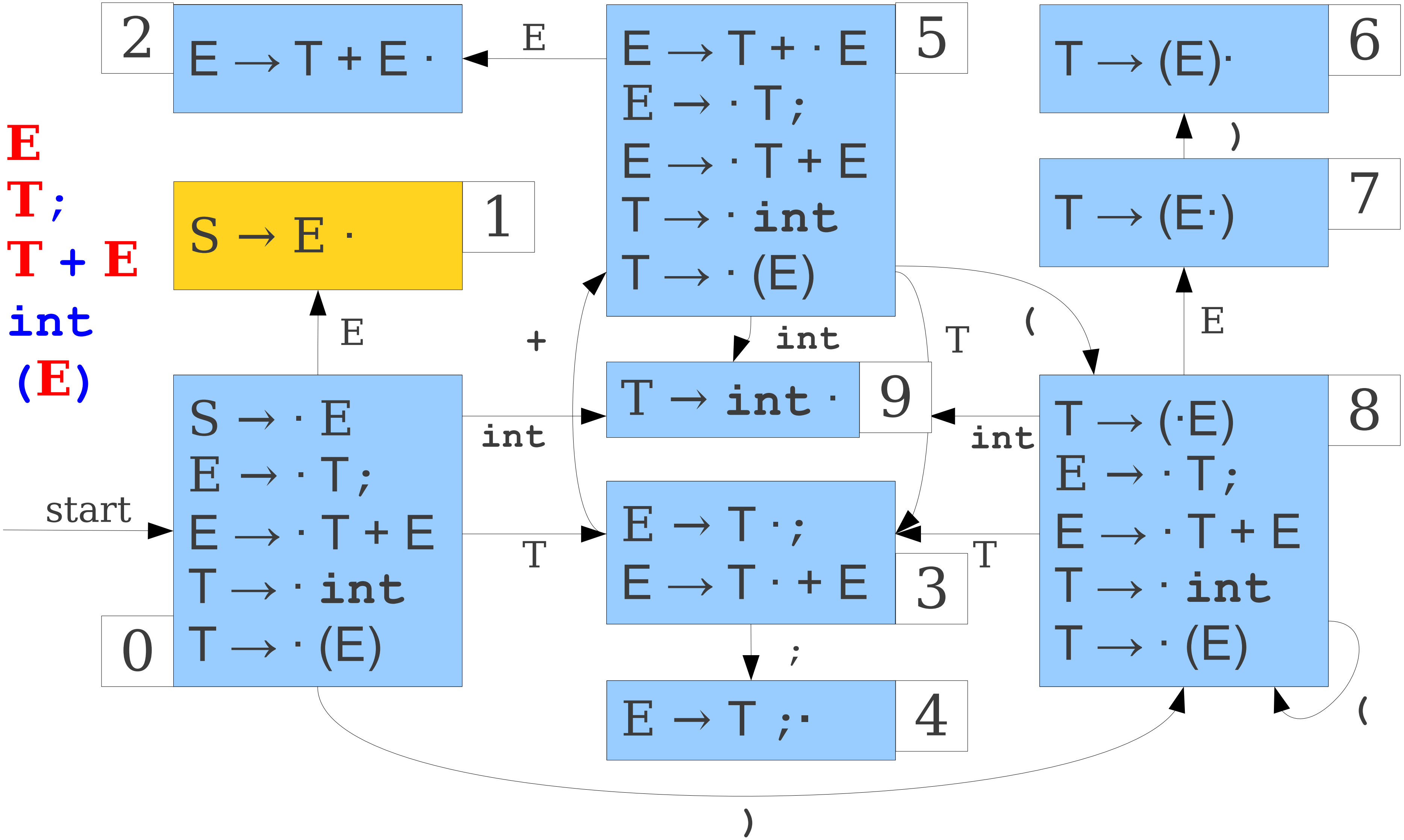
8
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

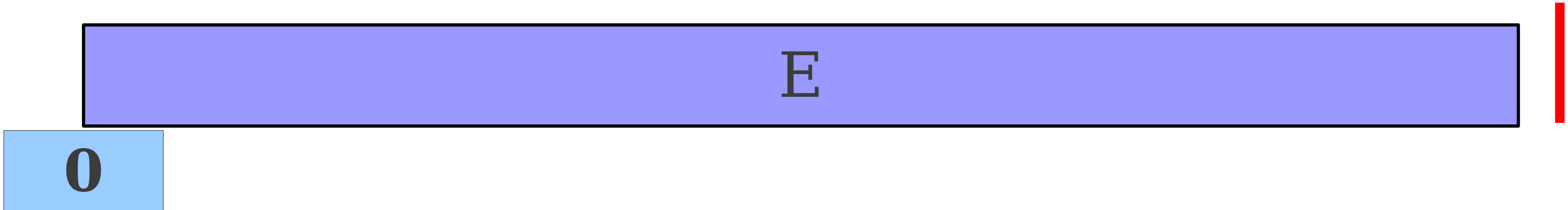
9
T → int ·



LR(0) Parsing

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



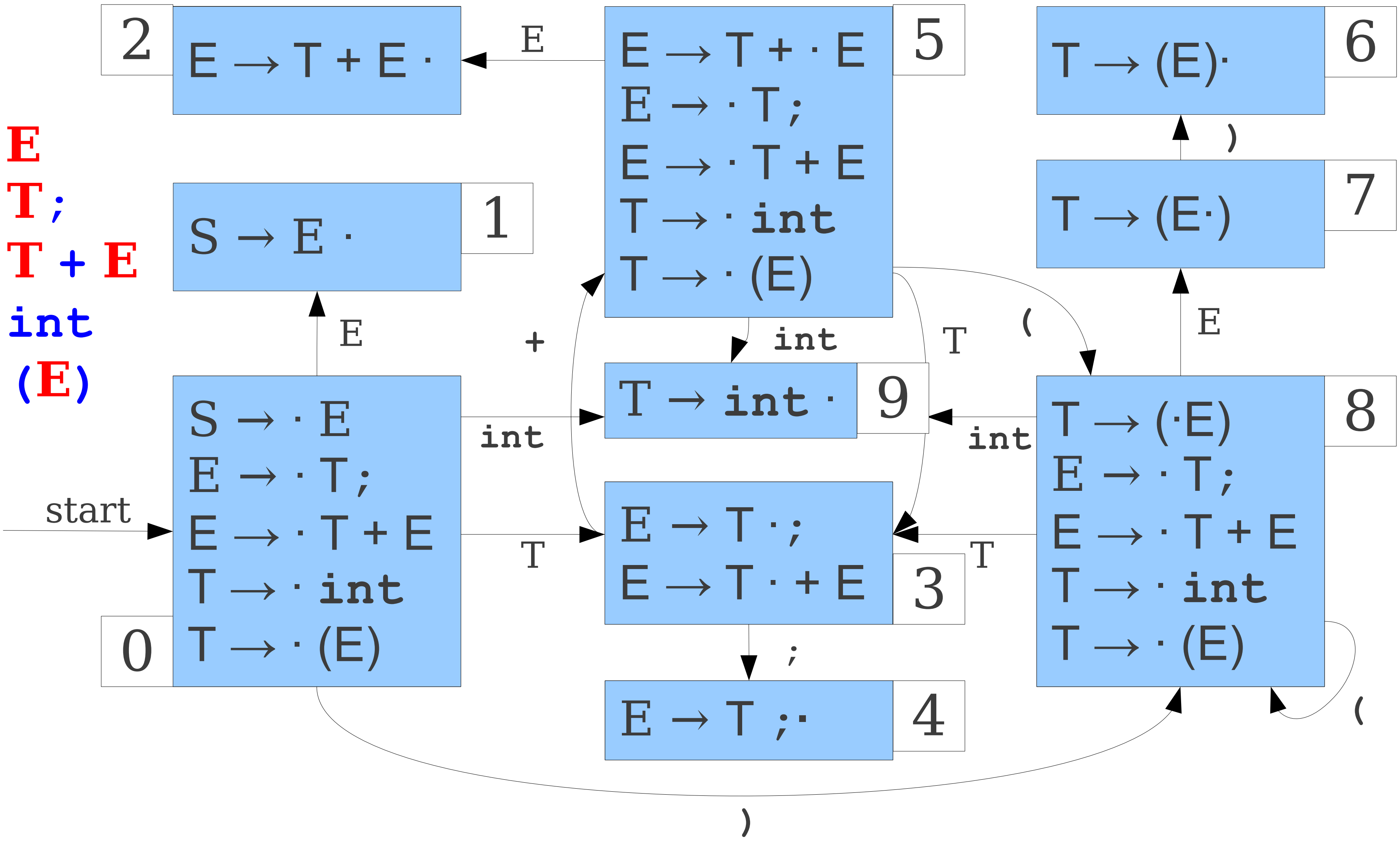


Representing the Automaton

- 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** $A \to \omega$, which performs reduction $A \to \omega$.
  - Any state of the form $A \to \omega \cdot$ 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**

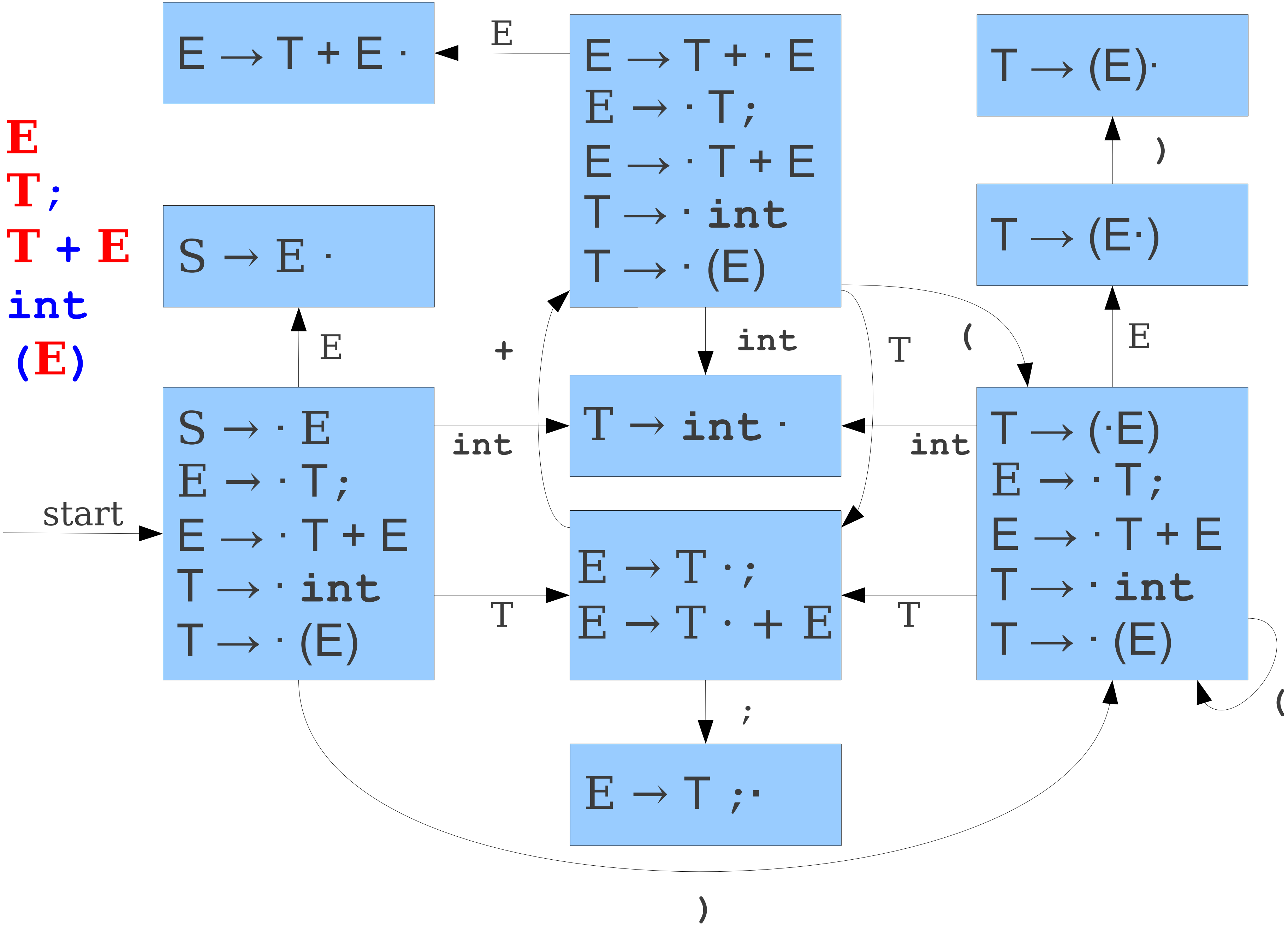
The LR(0) Algorithm

- 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**`)`



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 ;·

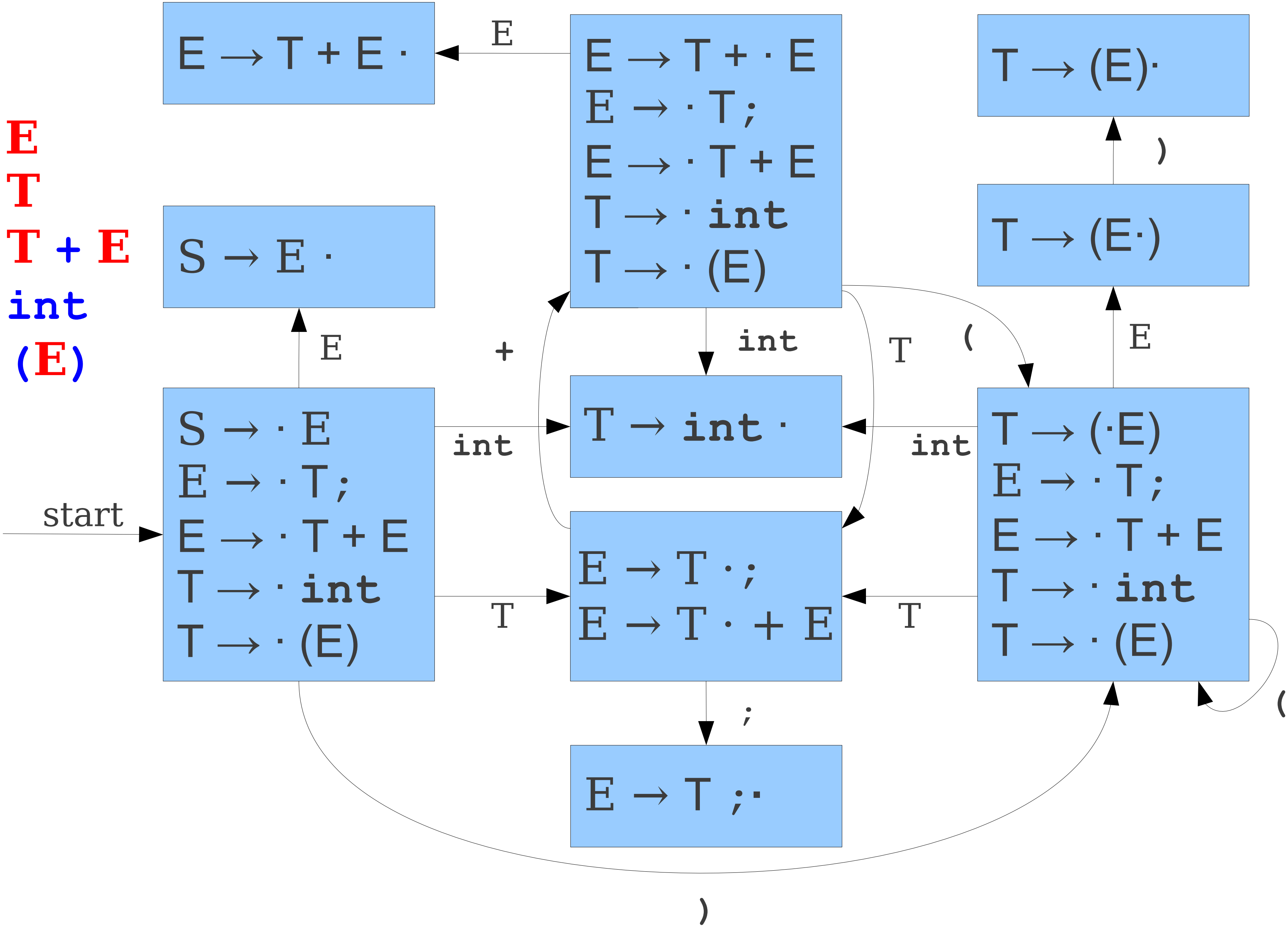
T → (·E)
E → · T;
E → · T + E
T → · int
T → · (E)

T → (E·)

T → (E)·

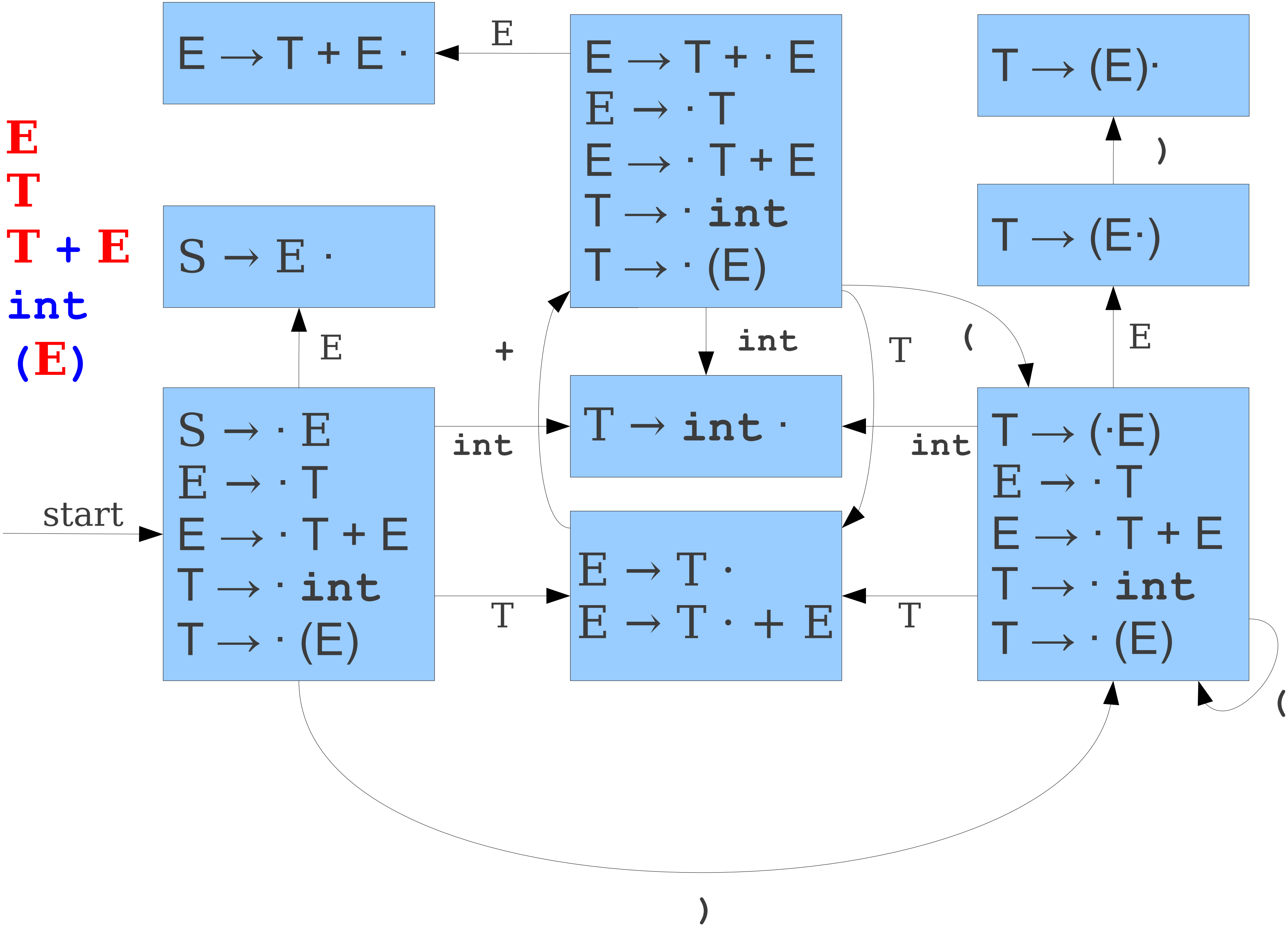
A Non-LR(0) Grammar

S → **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)**



E → T + E ·

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

T → (E)·

S → E ·

T → (E·)

T → int ·

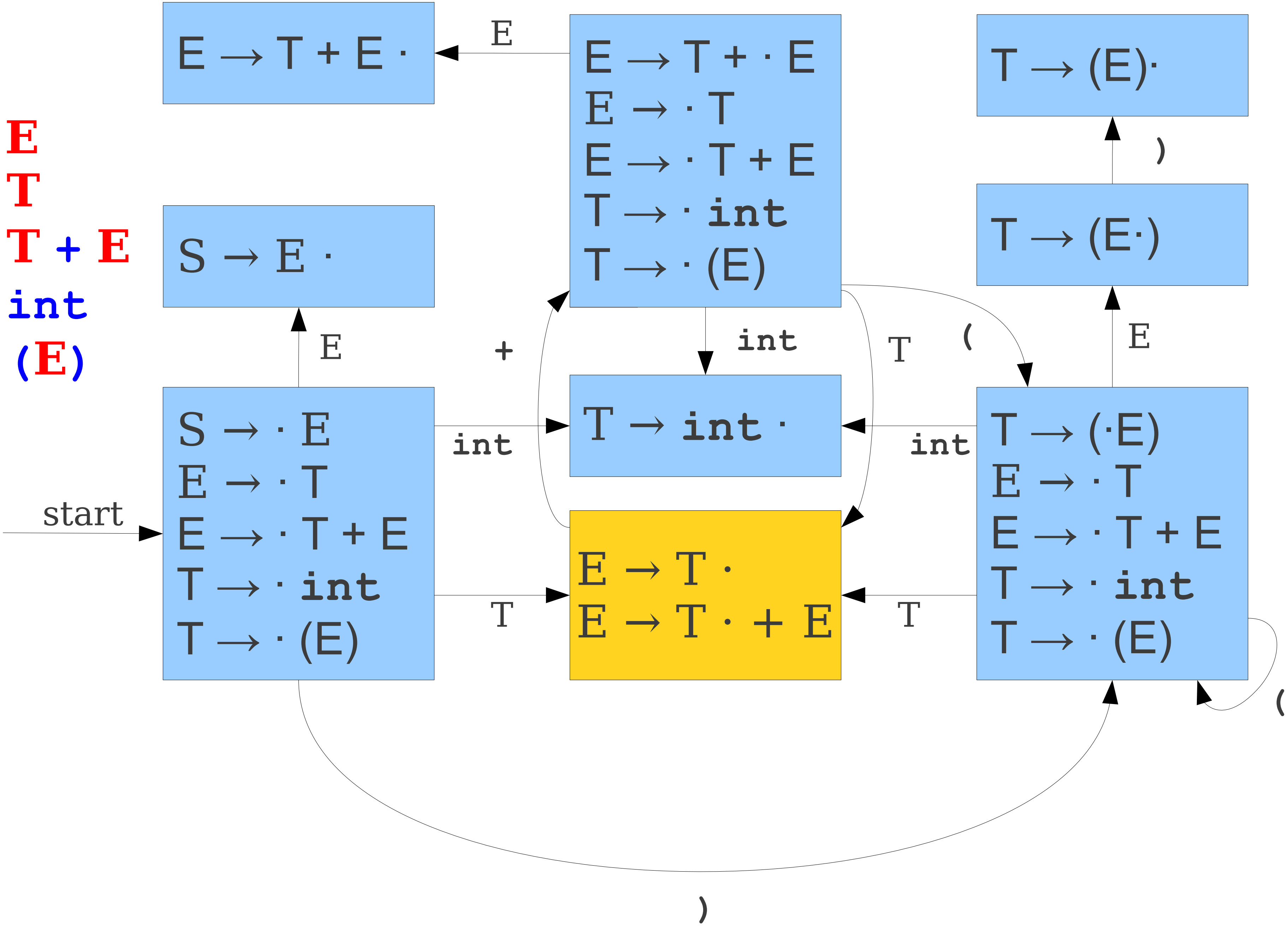
S → · E
E → · T
E → · T + E
T → · int
T → · (E)

E → T ·
E → T · + E

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

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 → · T + E
T → · **int**
T → · (E)

E → T + E ·

T → **int** ·

E → T ·
E → T · + E

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

T → (E·)

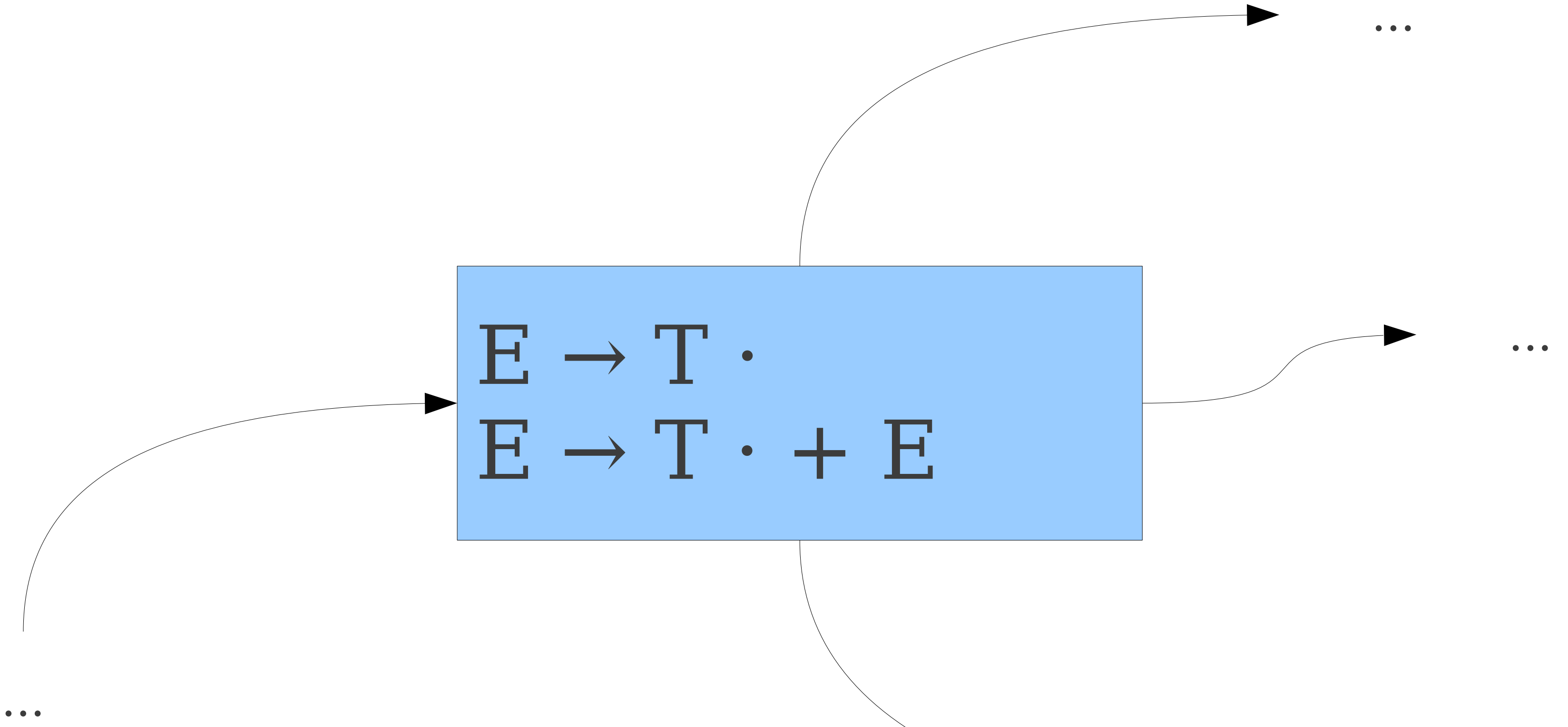
T → (E)·

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.
- 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 error is this?

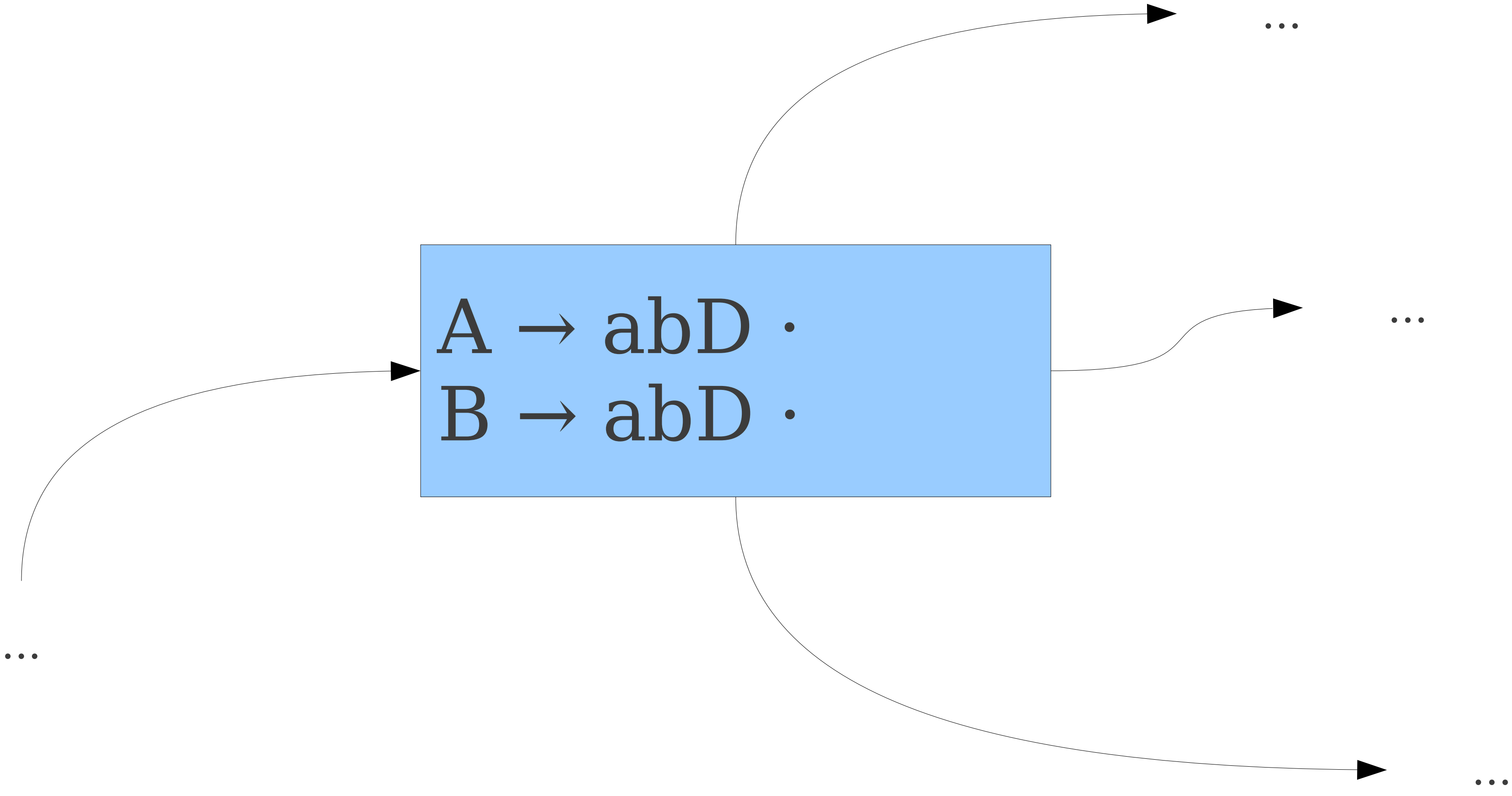


$E \rightarrow T \cdot$

$E \rightarrow T \cdot + E$

What about this?

What about this?



A → abD ·
B → abD ·

And what about this?

And what about this?

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**.
- Our shift/reduce parser only looks to the left of the handle, not to the right.
- How do we exploit the tokens after a possible handle to determine what to do?

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

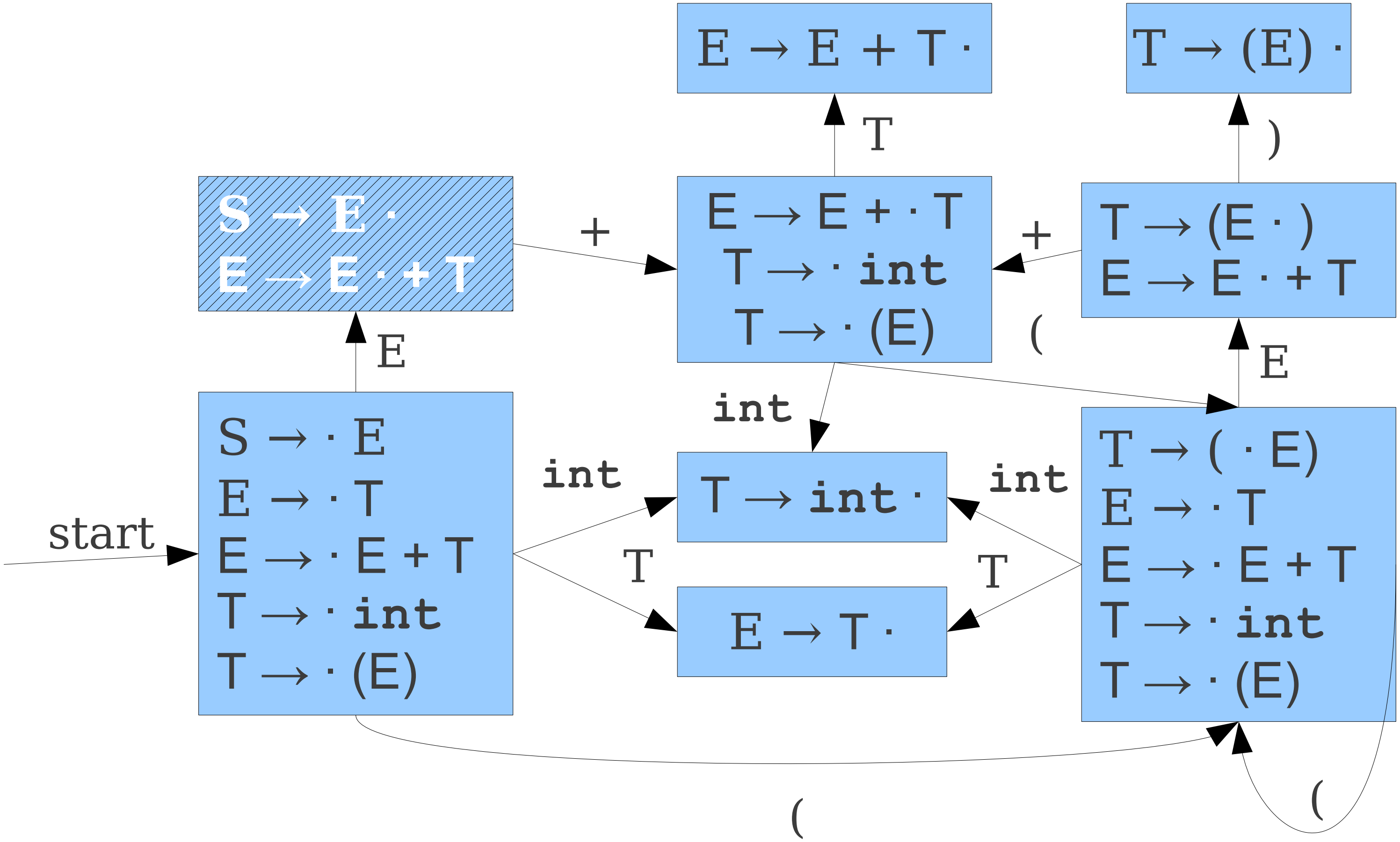
- Bottom-up predictive parsing with
  - **L**: **L**eft-to-right scan
  - **R**: **R**ightmost derivation
  - (**1**): One token lookahead
- *Substantially* more powerful than the other methods we've covered so far (more on that later).
- Tries to more intelligently find handles by using a lookahead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

LR(1) Parsing: The Intuition

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

LR(1) Parsing: The Intuition

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



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 ·

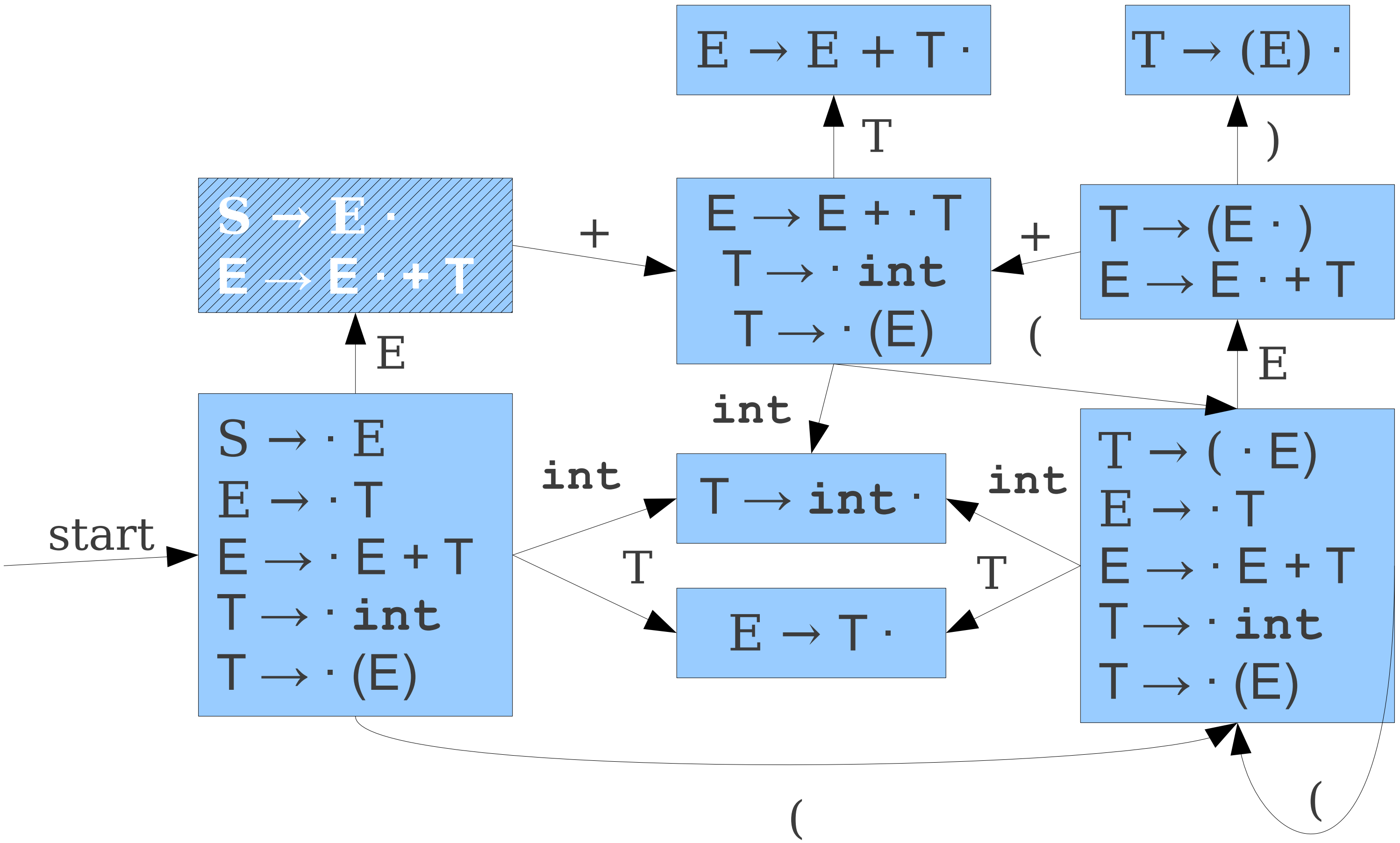
T → (E ·)
E → E · + T

T → (E) ·

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

LR(1) Parsing: The Intuition

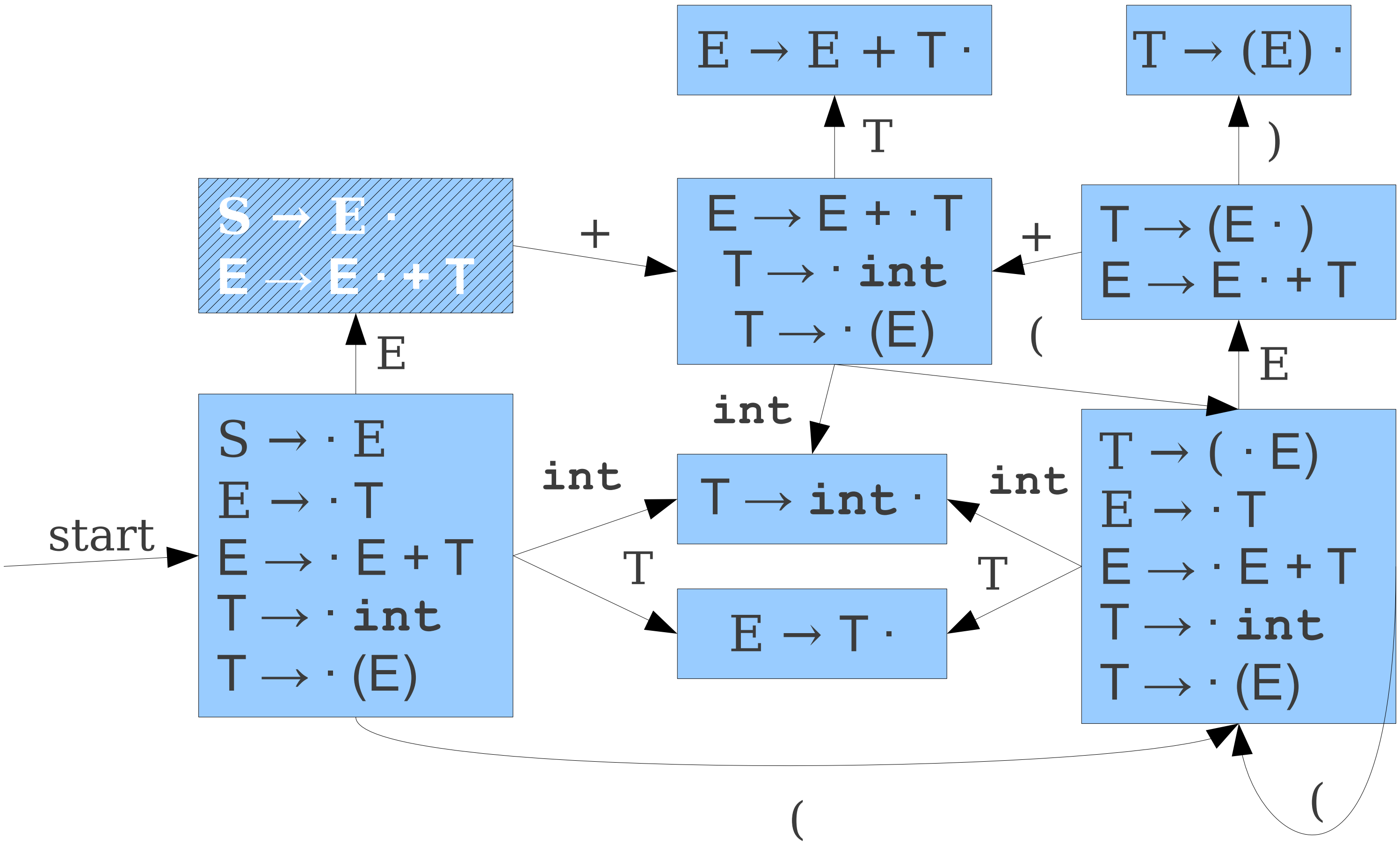
S → **E**
E → **T**
E → **E + T**
T → **`int`**
T → **(E)**



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

LR(1) Parsing: The Intuition

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



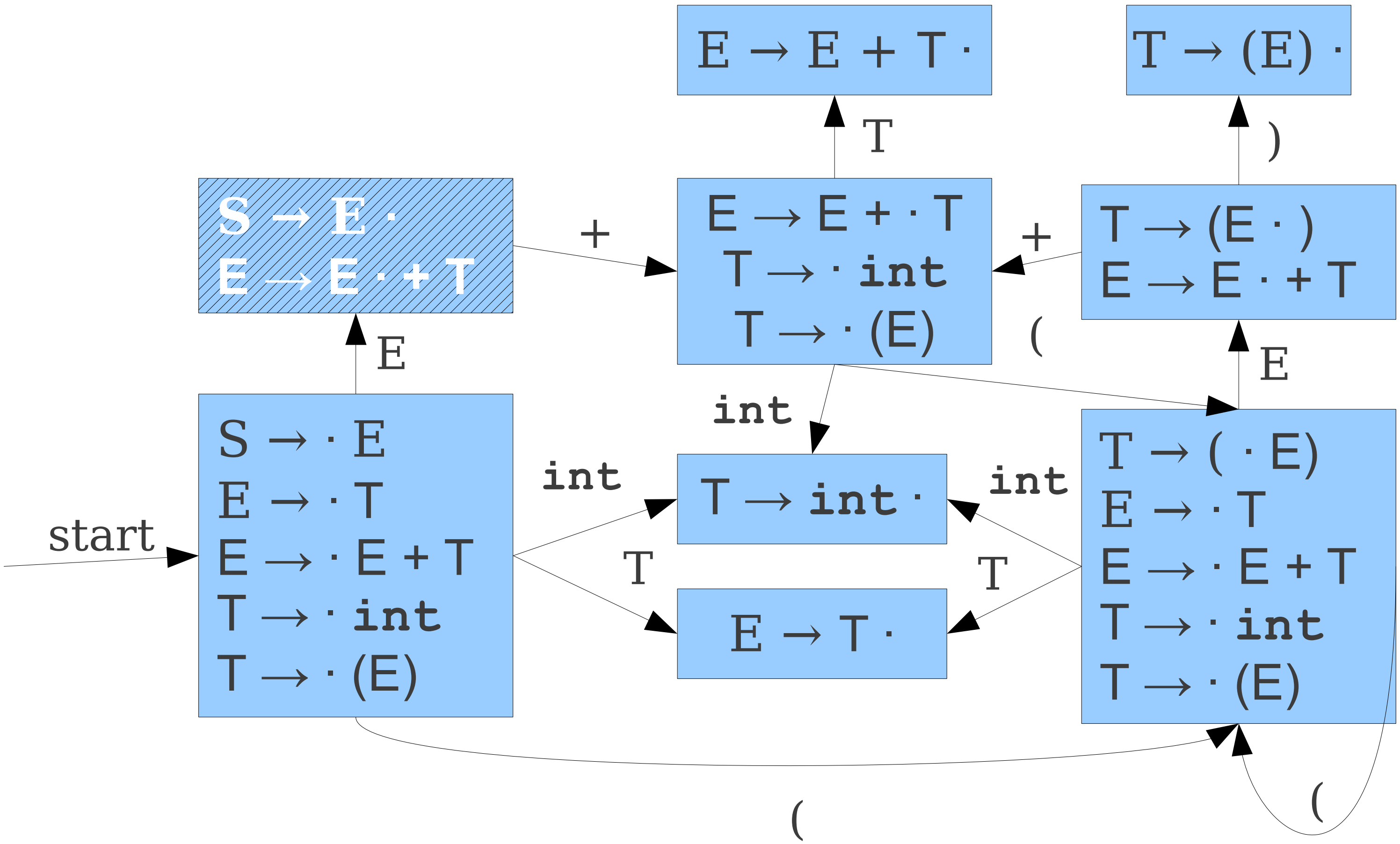
S → · E	$

int	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

S → · E	$
E → · E + T	$



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)

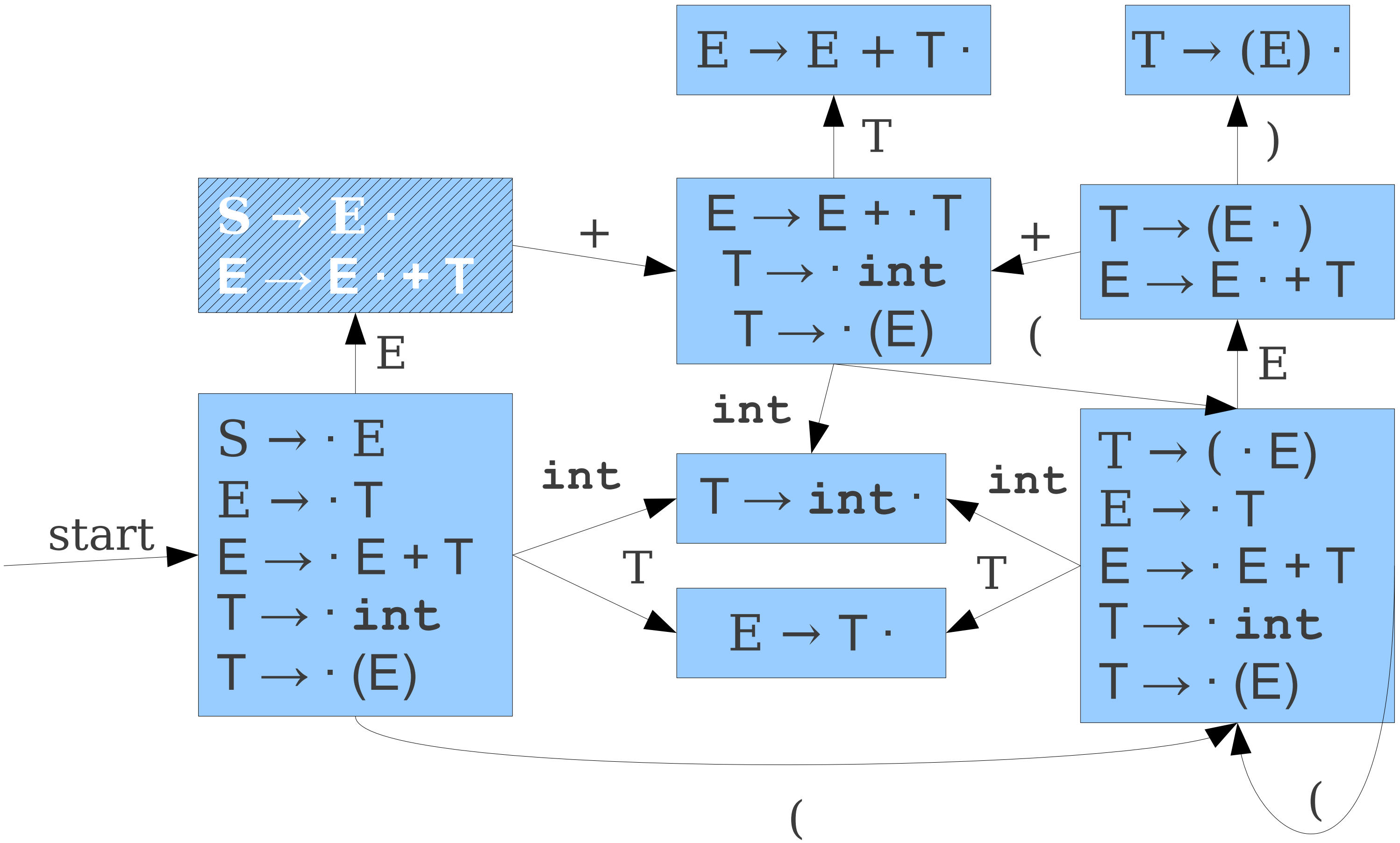
T → (E ·)
E → E · + T

T → (E) ·

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

LR(1) Parsing: The Intuition

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

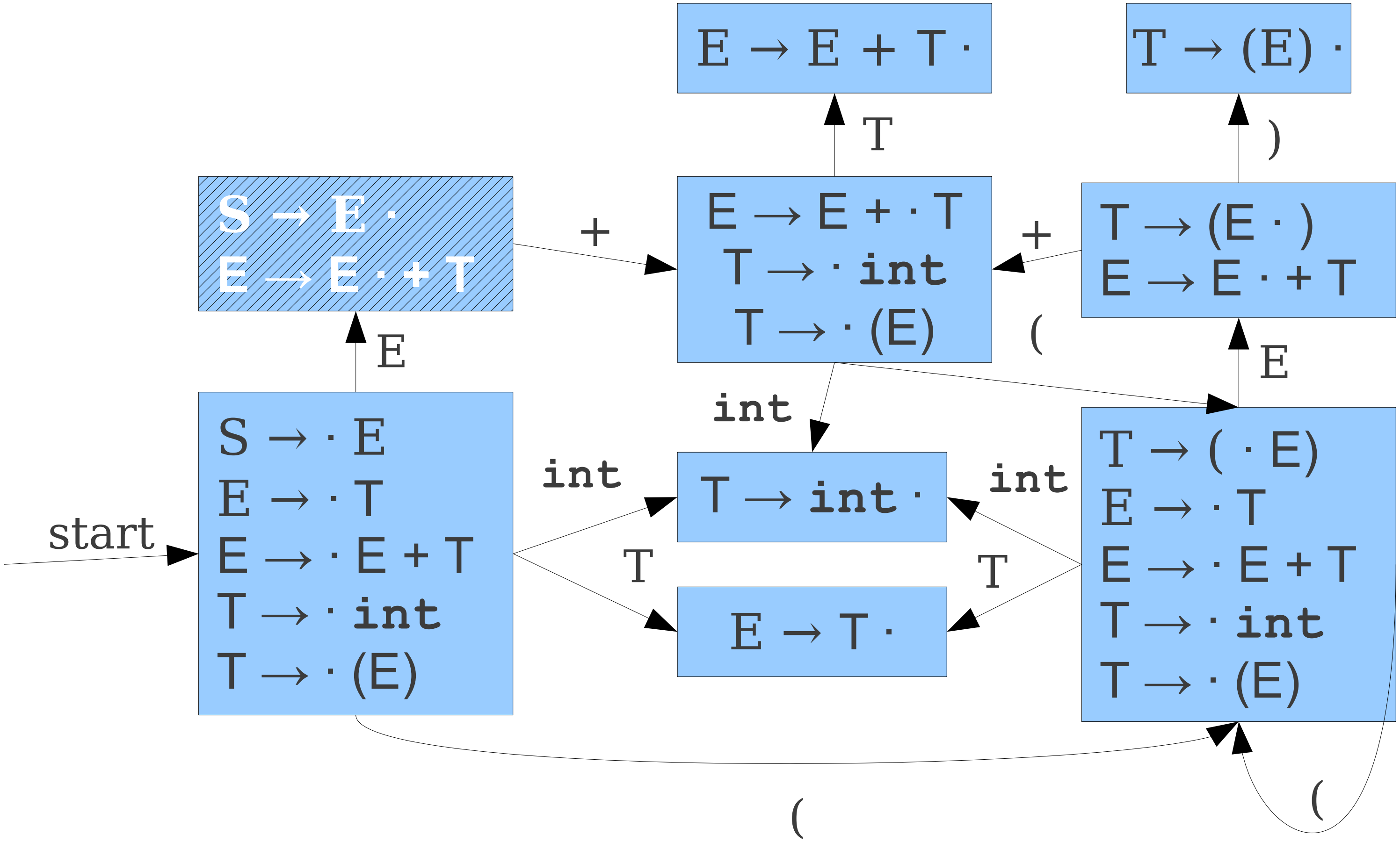


S → · E	$
E → · E + T	$
E → · T	+

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

LR(1) Parsing: The Intuition

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



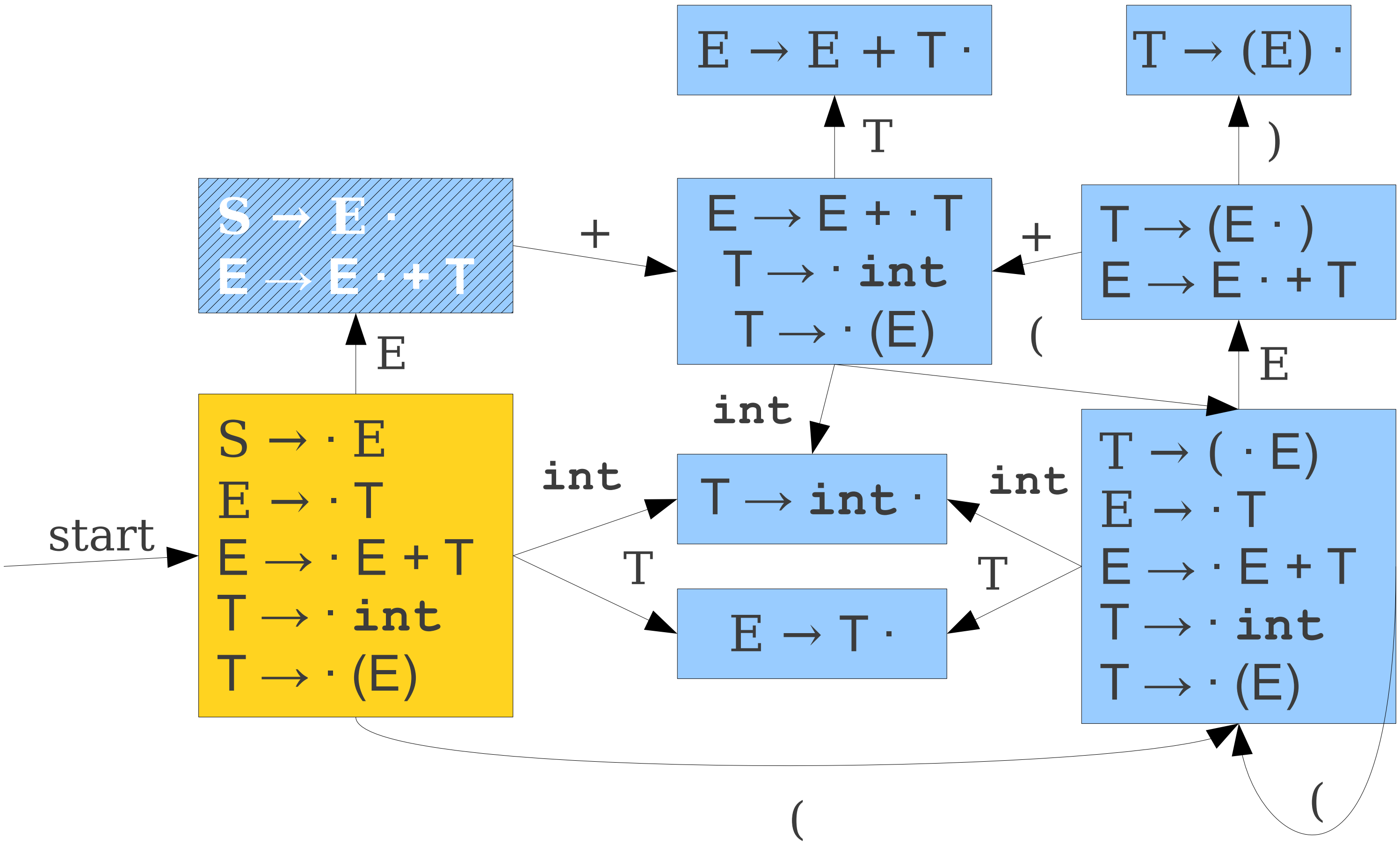
S → · E	$
E → · E + T	$
E → · T	+
T → · int	+

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

LR(1) Parsing: The Intuition

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

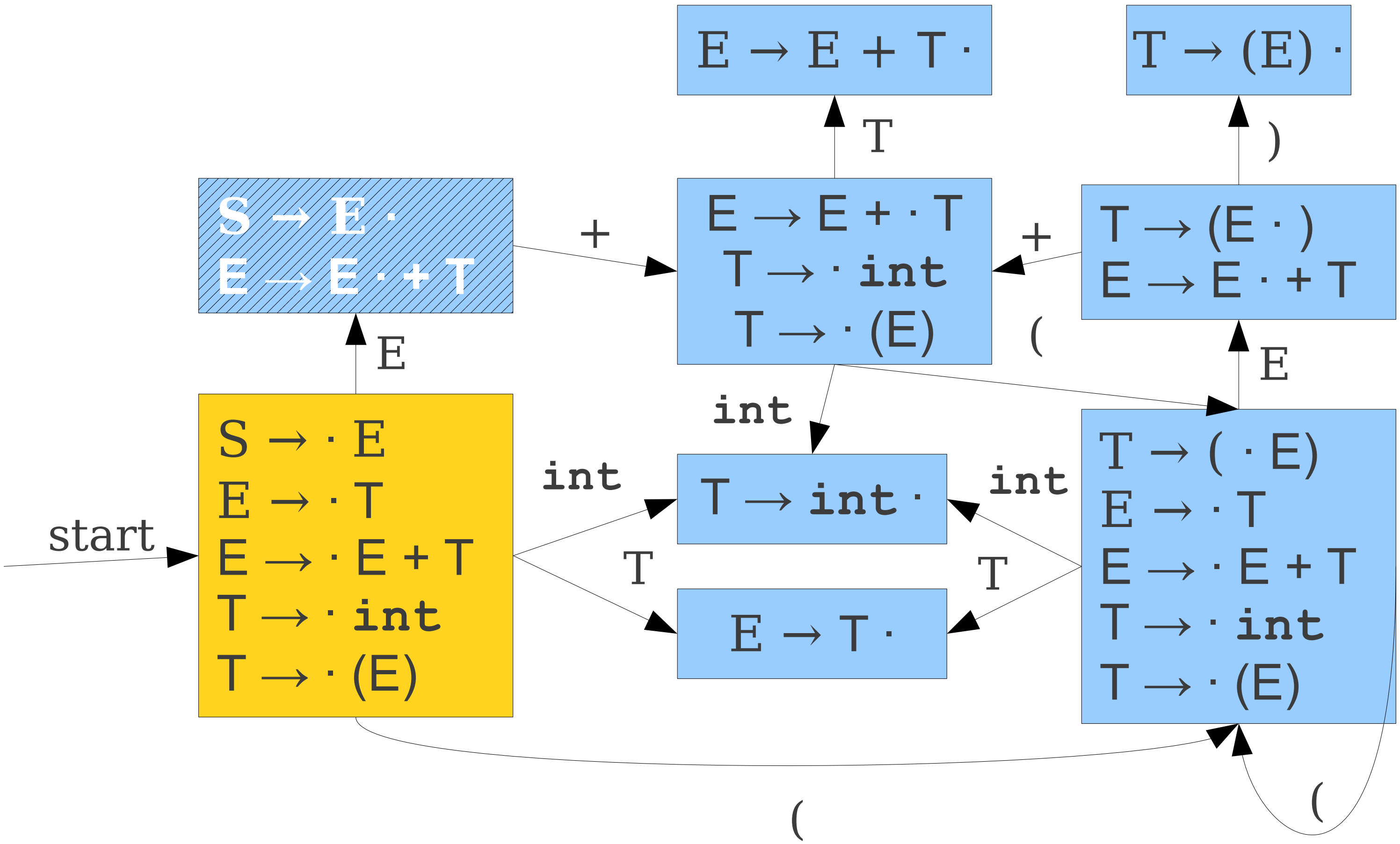
S → · E	$
E → · E + T	$
E → · T	+
T → · `int`	+



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

LR(1) Parsing: The Intuition

S → E
E → T
E → E + T
T → int
T → (E)

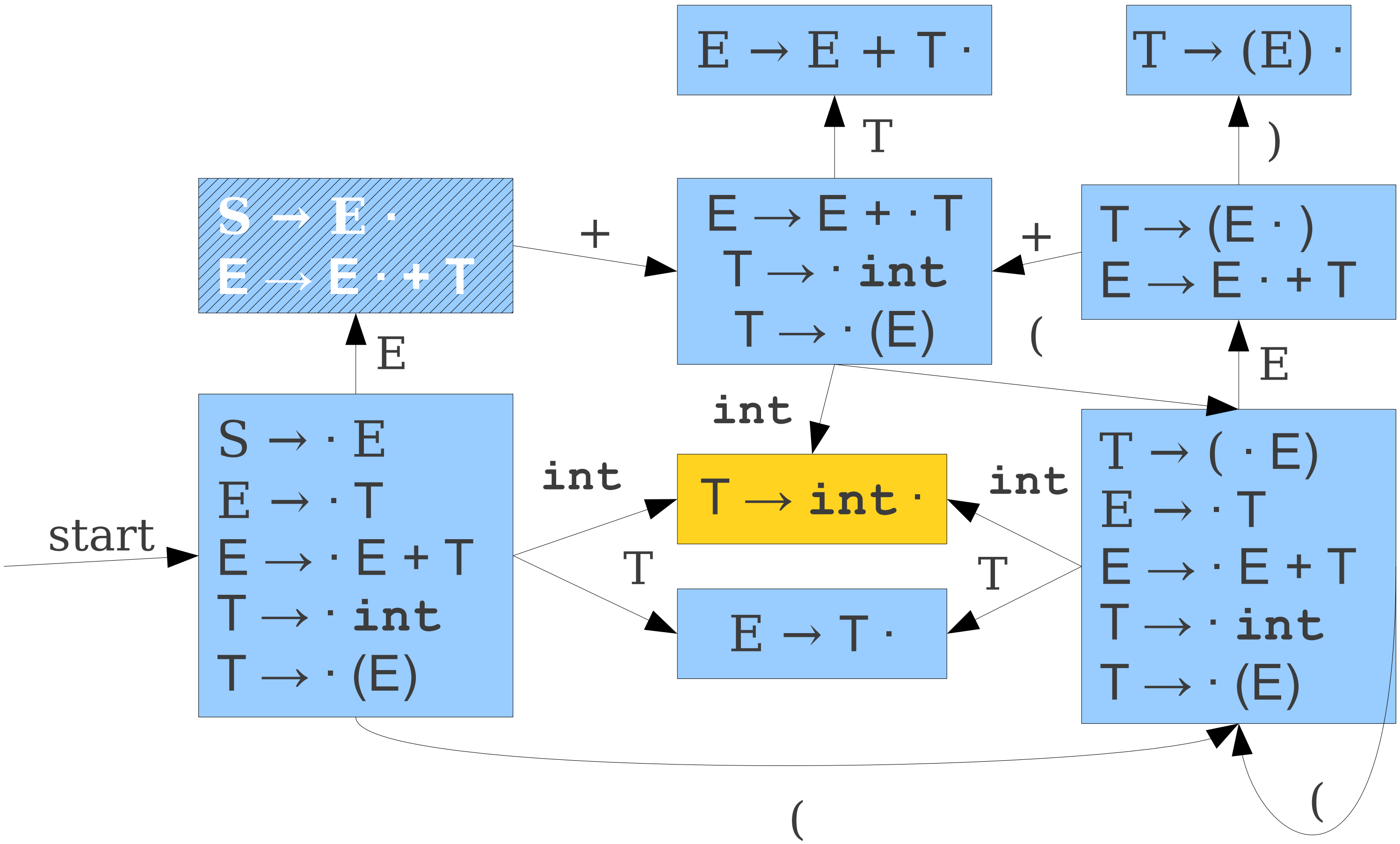


S → · E	$
E → · E + T	$
E → · T	+
T → · int	+

int	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

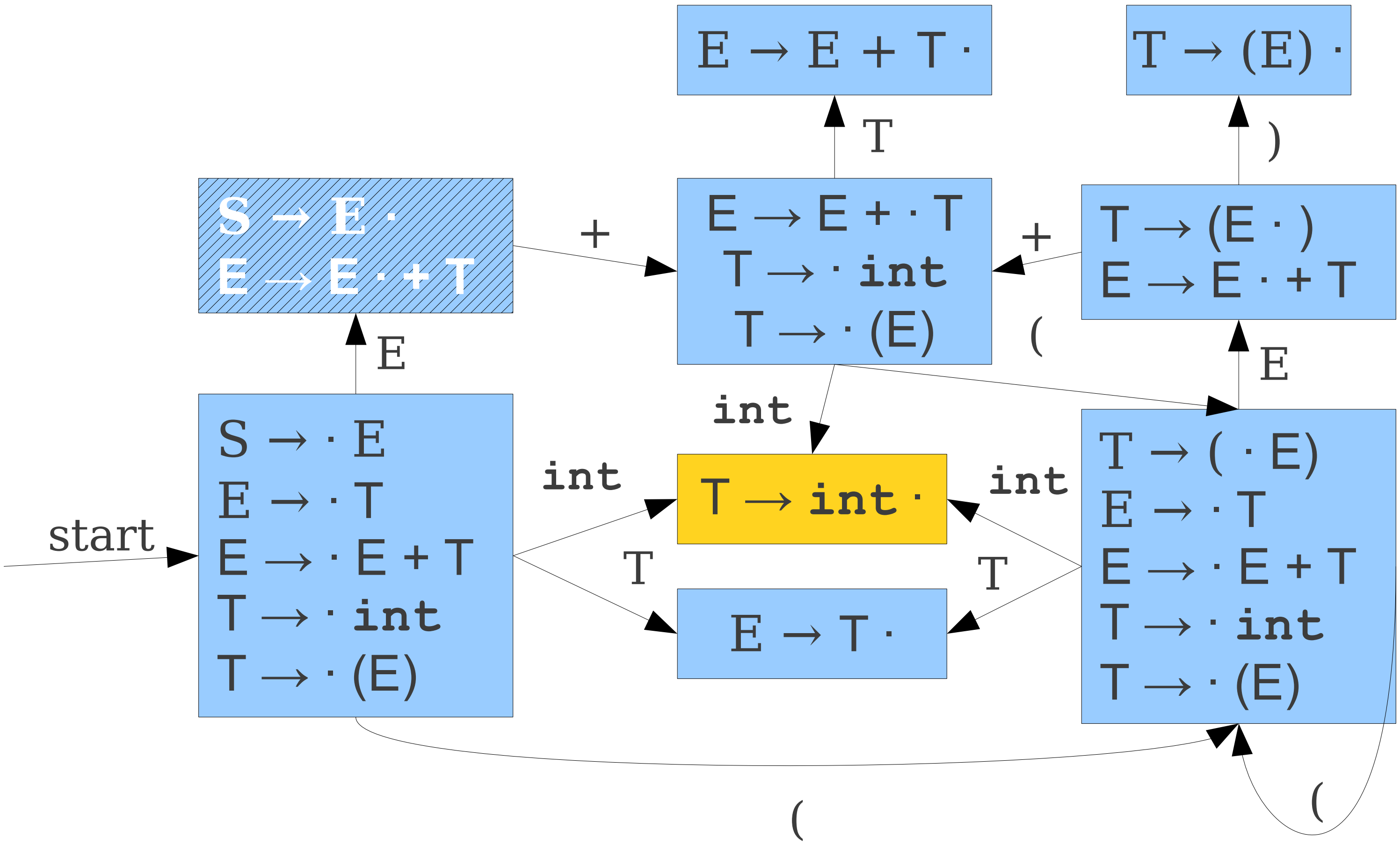


S → · E	$
E → · E + T	$
E → · T	+
T → · int	+

int	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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



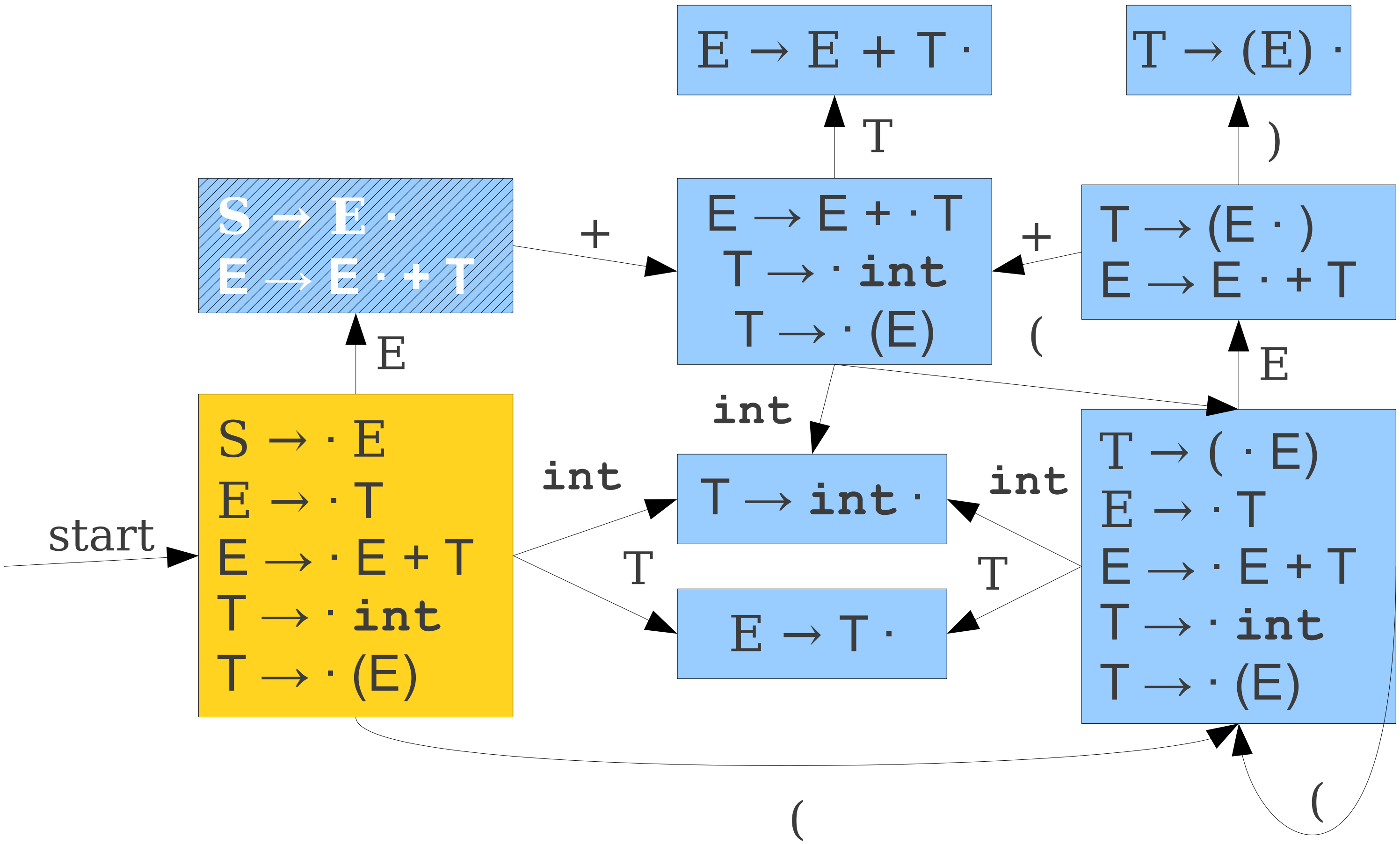
S → · E	$
E → · E + T	$
E → · T	+
T → `int` ·	+

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

LR(1) Parsing: The Intuition

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

S → · E	$
E → · E + T	$
E → · T	+



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)

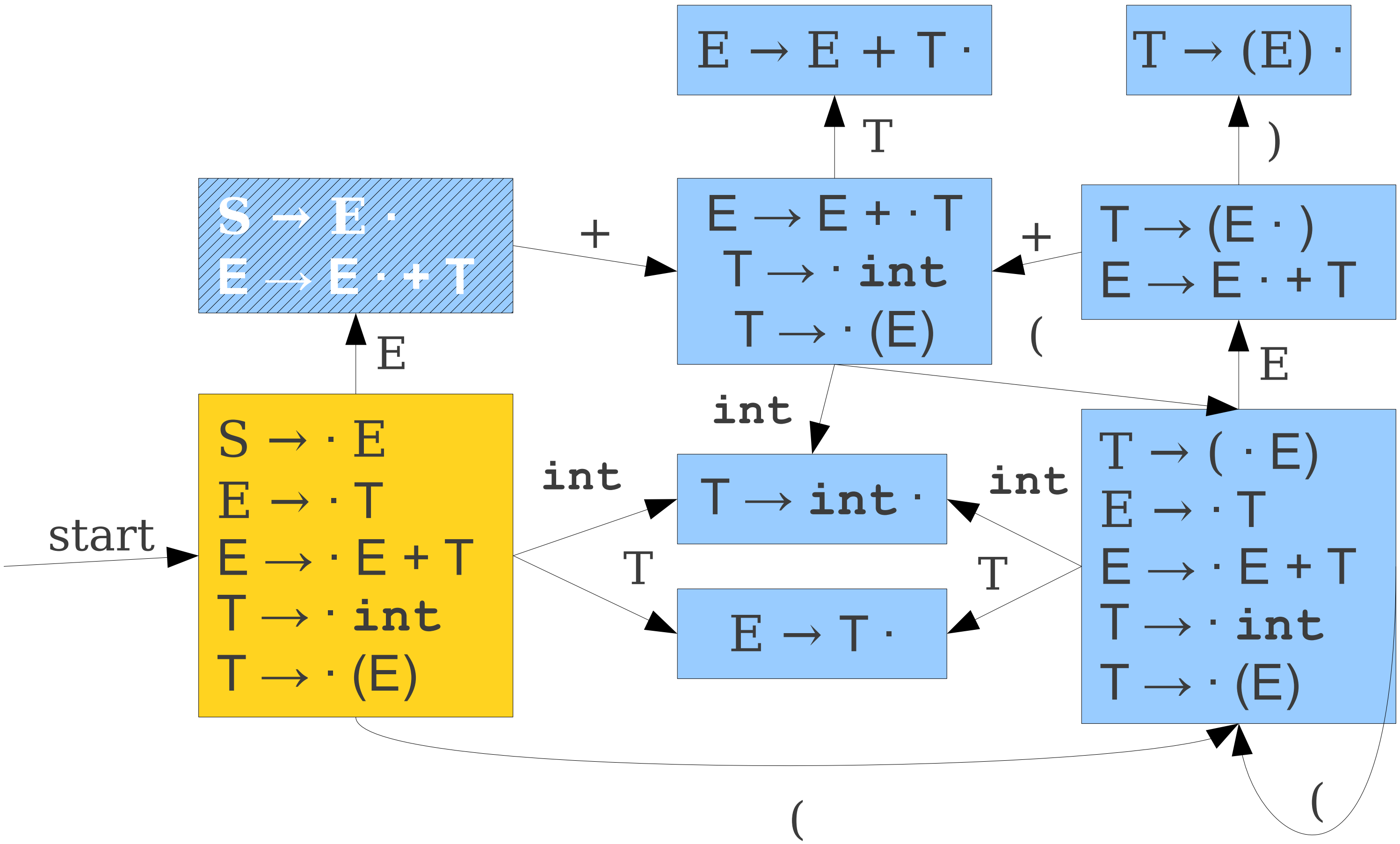
T → (E ·)
E → E · + T

T → (E) ·

+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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



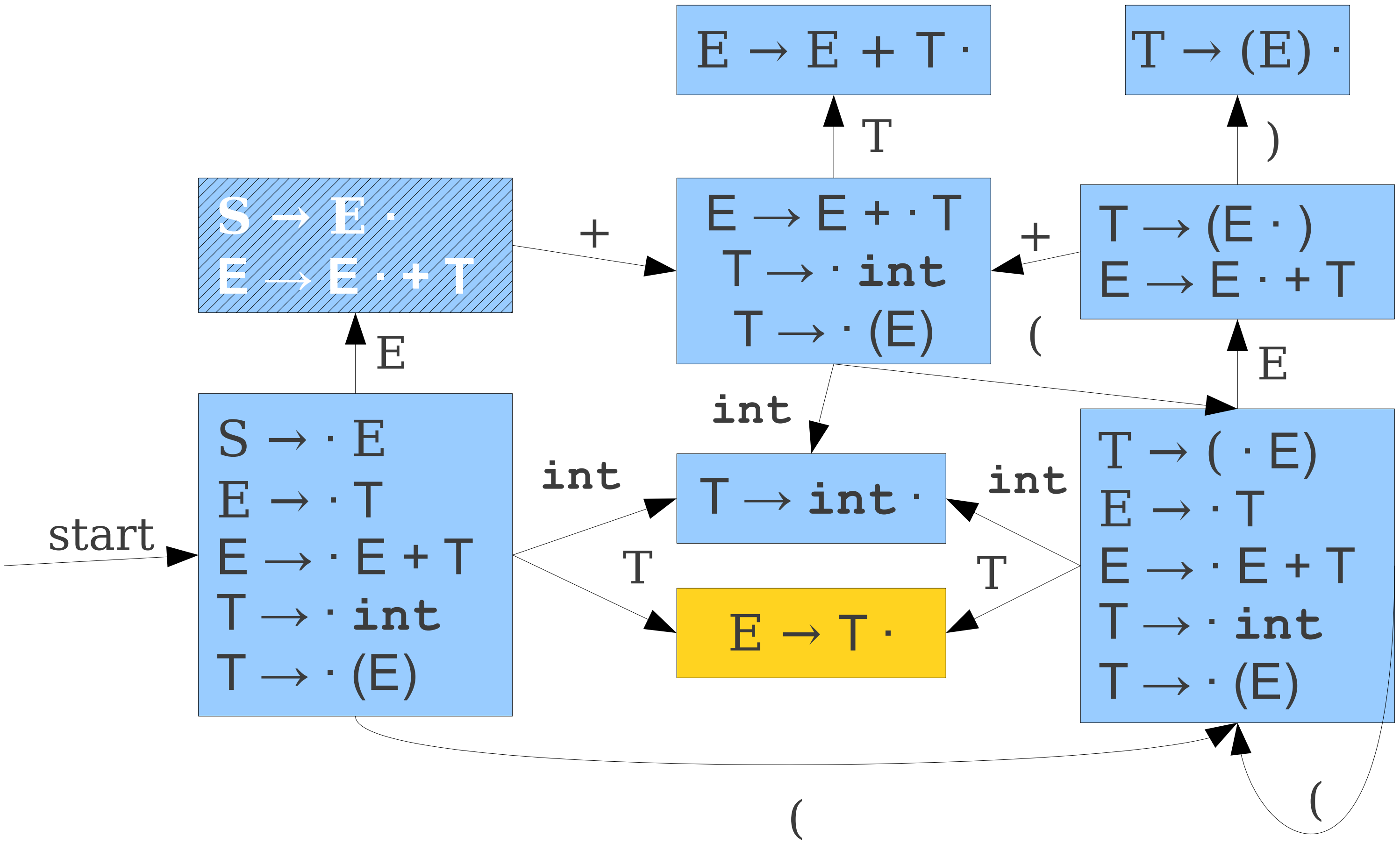
S → · E	$
E → · E + T	$
E → · T	+

T	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → E
E → T
E → E + T
T → int
T → (E)

S → · E	$
E → · E + T	$
E → T ·	+

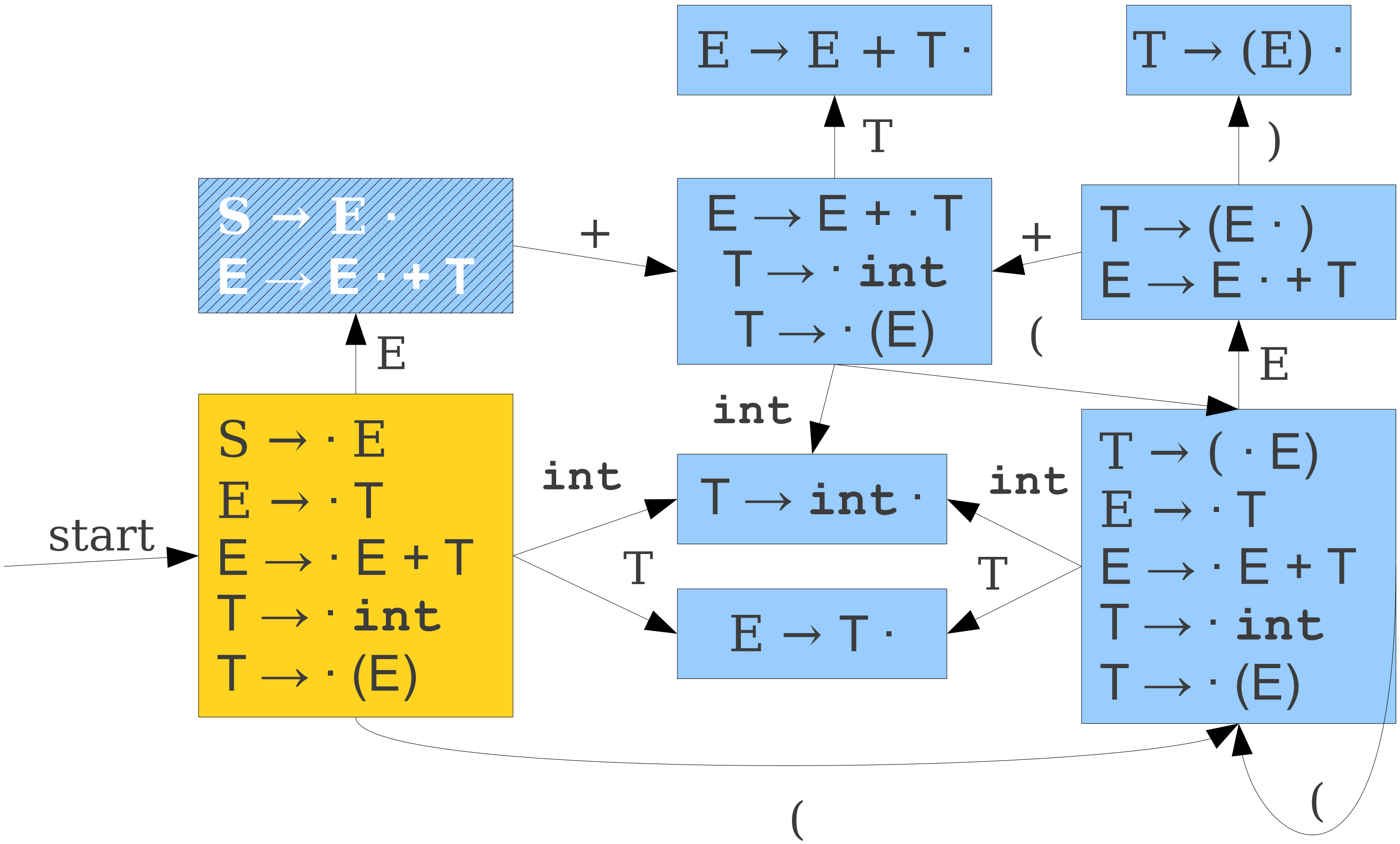


T	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → E
E → T
E → E + T
T → int
T → (E)

S → · E	$
E → · E + T	$

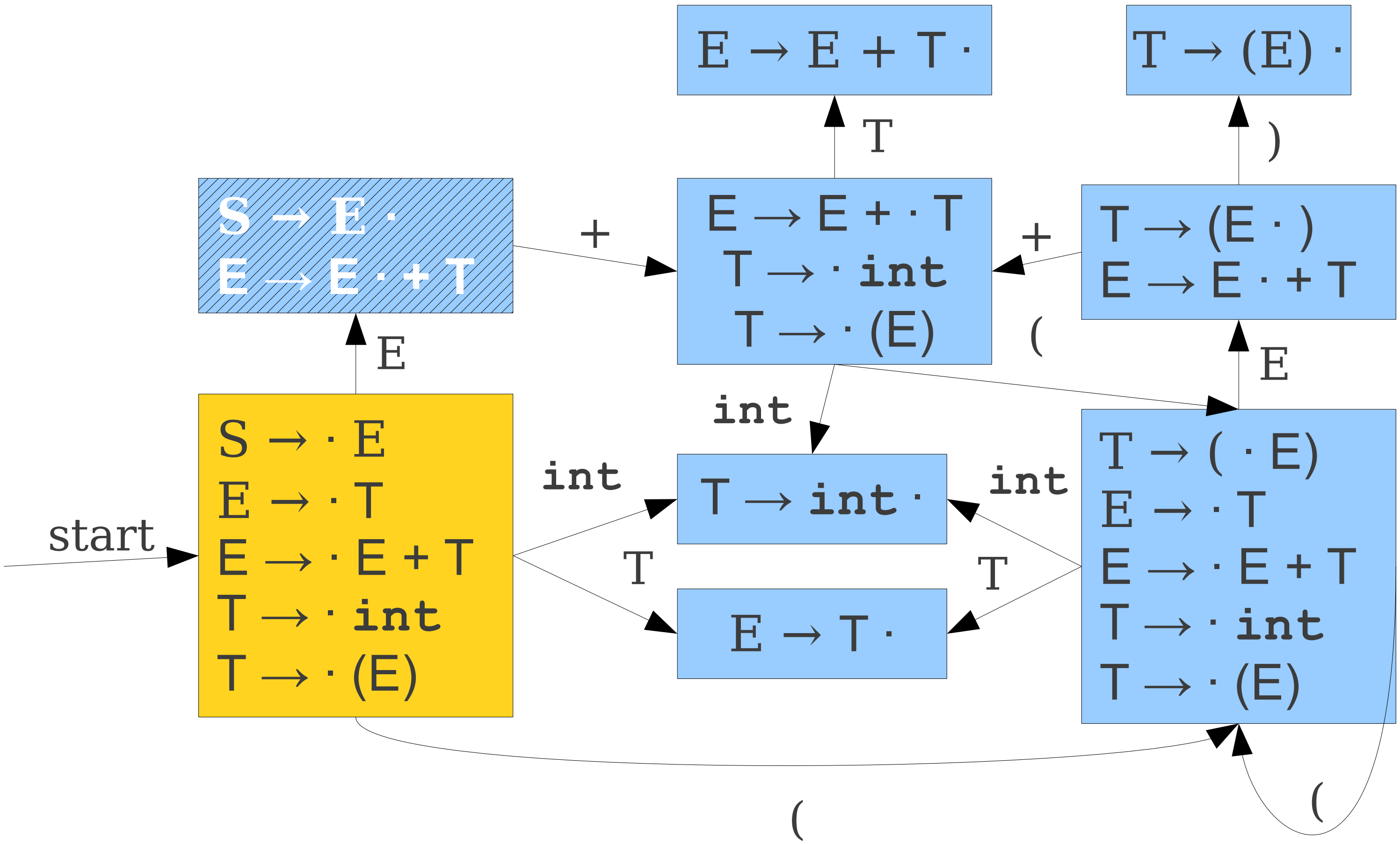


+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → E
E → T
E → E + T
T → int
T → (E)

S → · E	$
E → · E + T	$



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 → (E) ·

T → (E ·)
E → E · + T

T → int ·

E → T ·

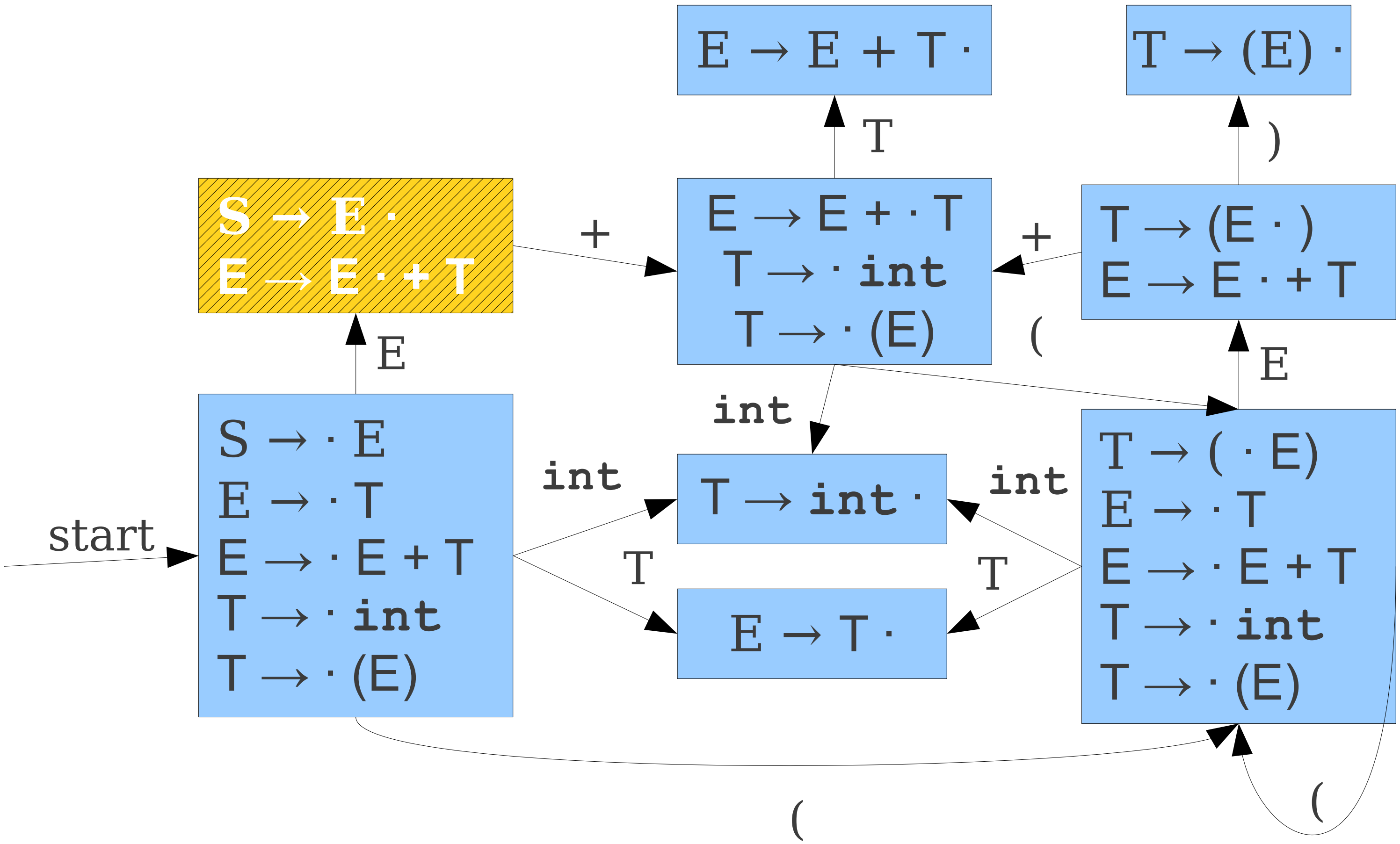
T → (· E)
E → · T
E → · E + T
T → · int
T → · (E)

E	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

S → · E	$
E → E · + T	$

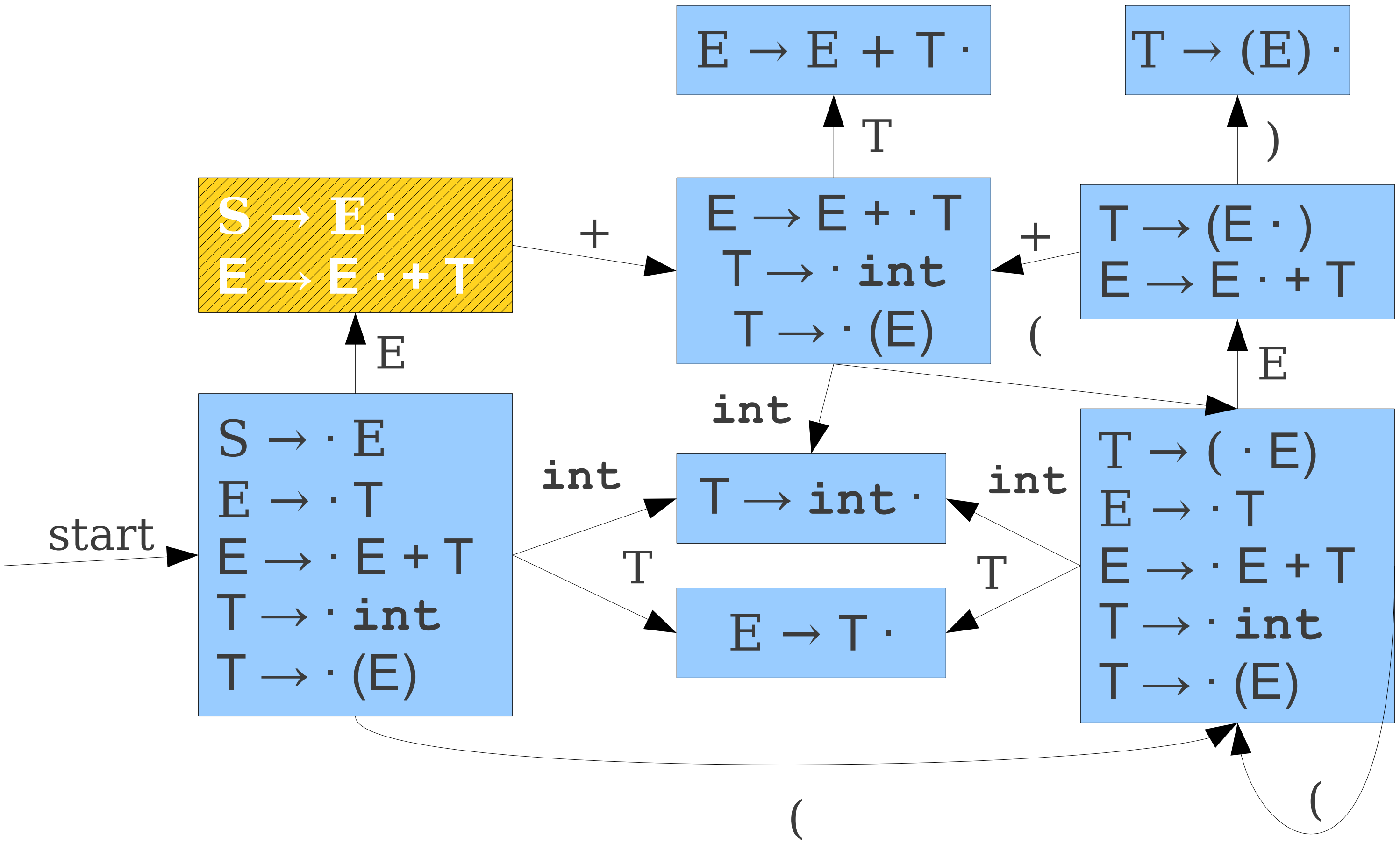


E	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

S → · E	$
E → E · + T	$

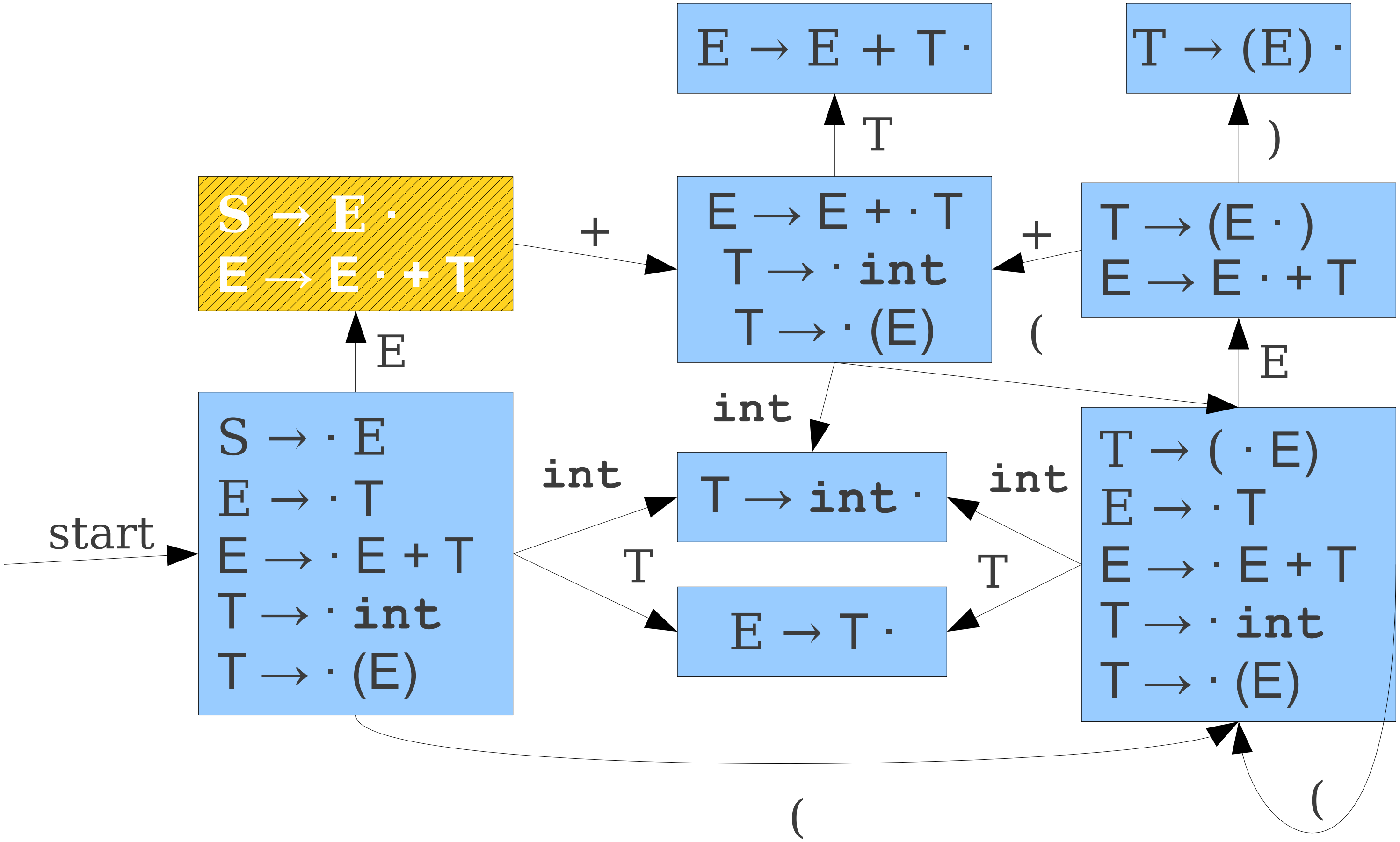


E	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → E
E → T
E → E + T
T → int
T → (E)

S → · E	$
E → E · + T	$

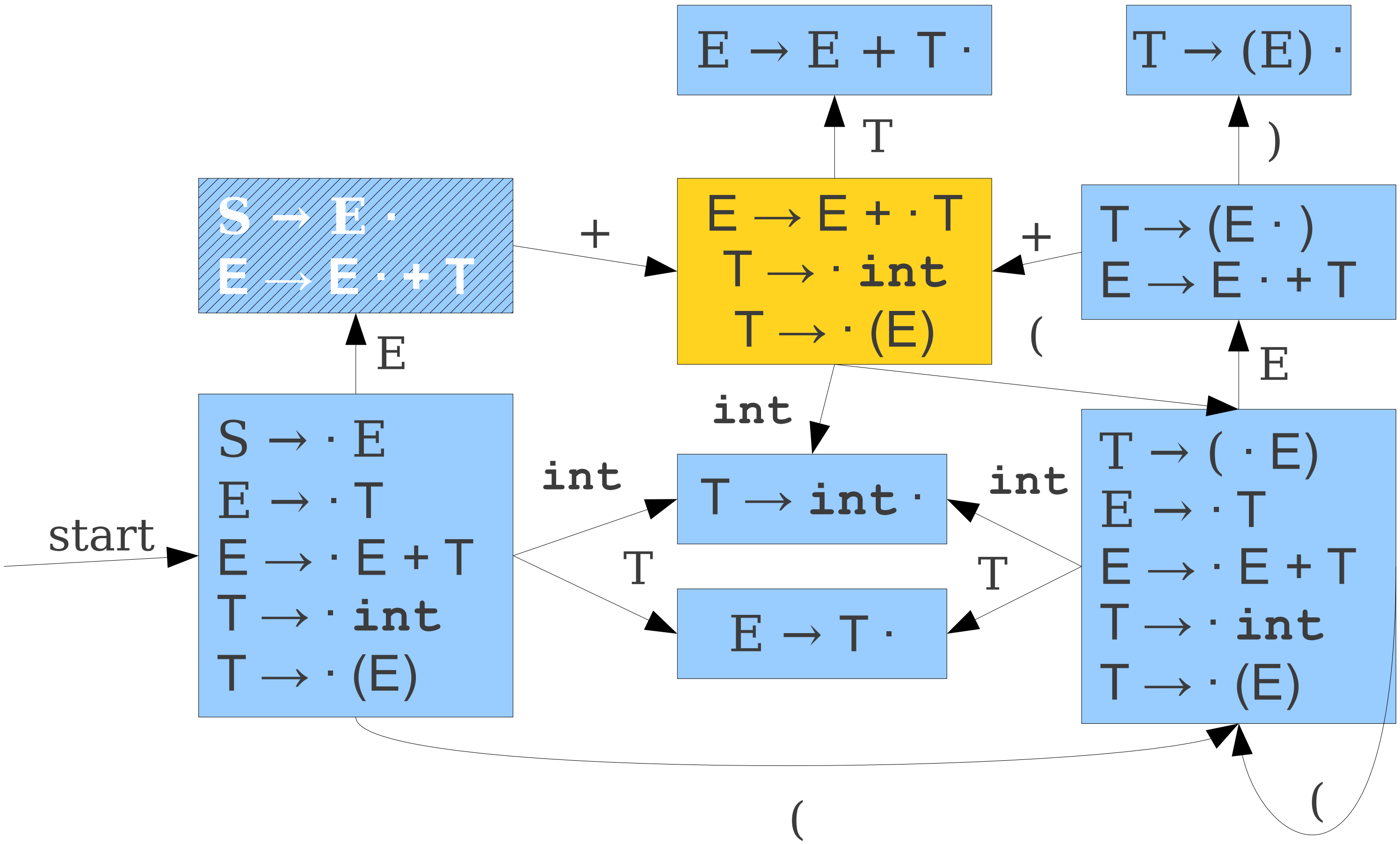


E	+	|	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

S → · E	$
E → E + · T	$



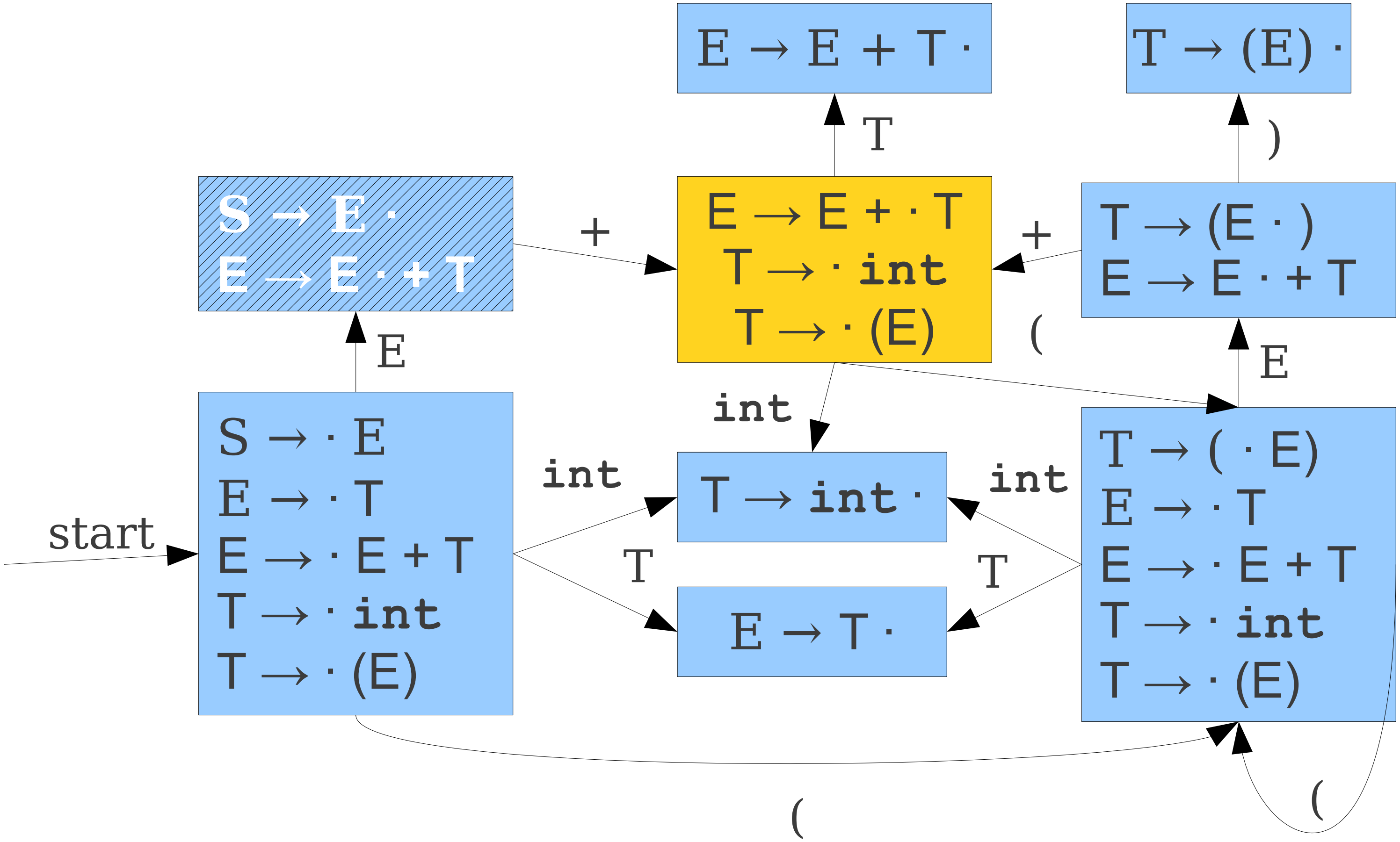
E	+

(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

S → E
E → T
E → E + T
T → int
T → (E)

S → · E	$
E → E + · T	$
T → · (E)	$

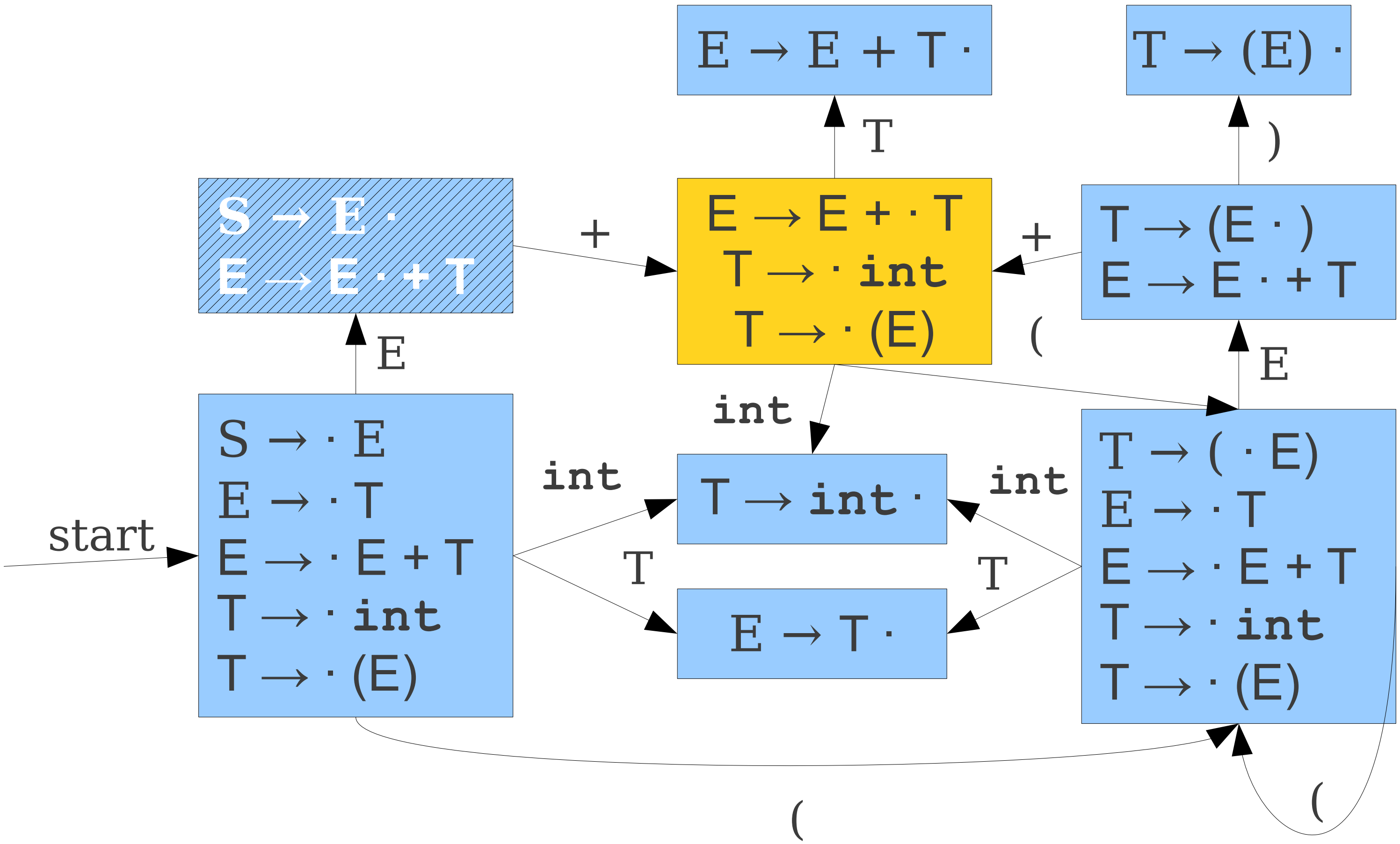


E	+		(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

S → · E	$
E → E + · T	$
T → · (E)	$



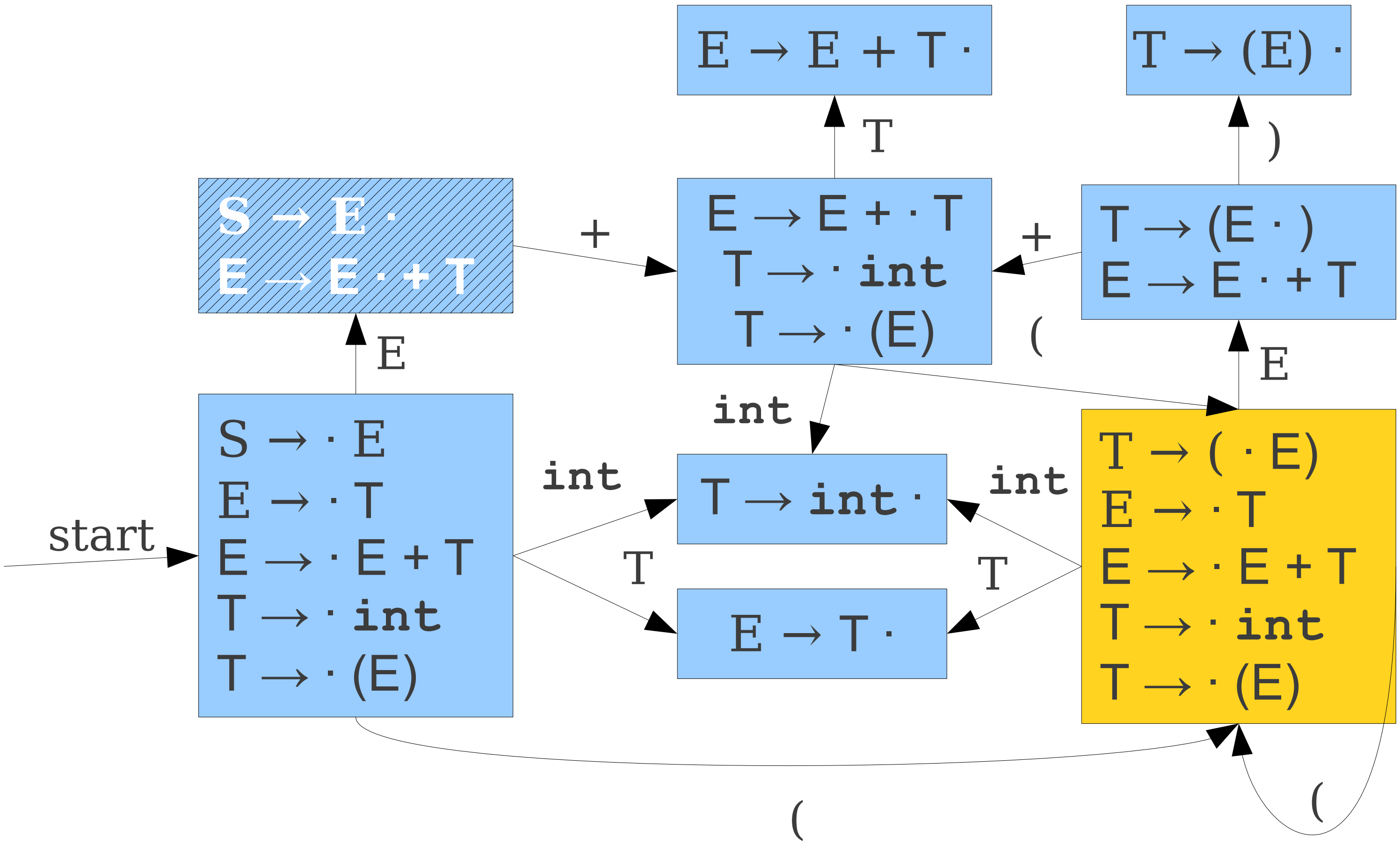
E	+	(

int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

S → · E	$
E → E + · T	$
T → (· E)	$



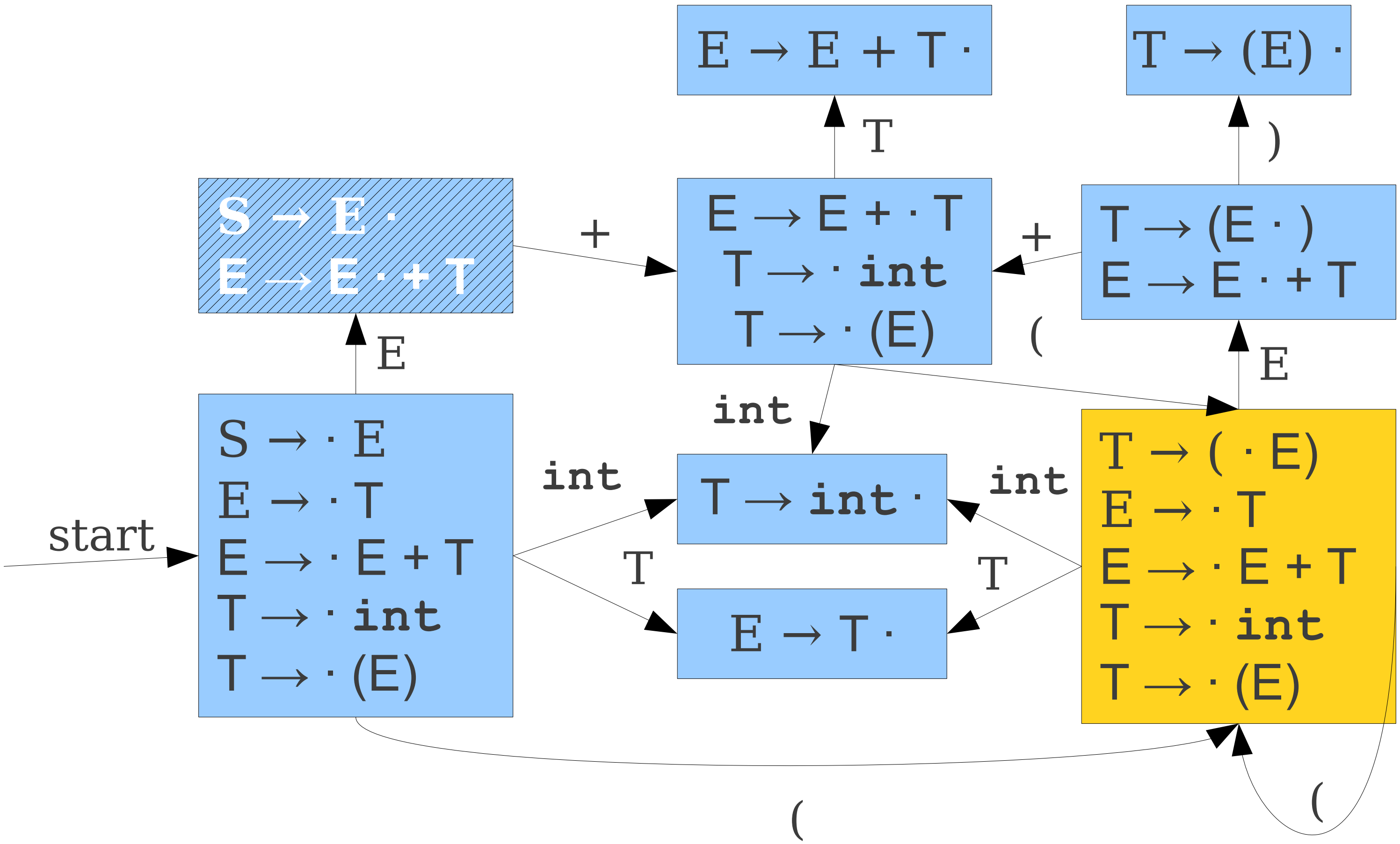
E	+	(

int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

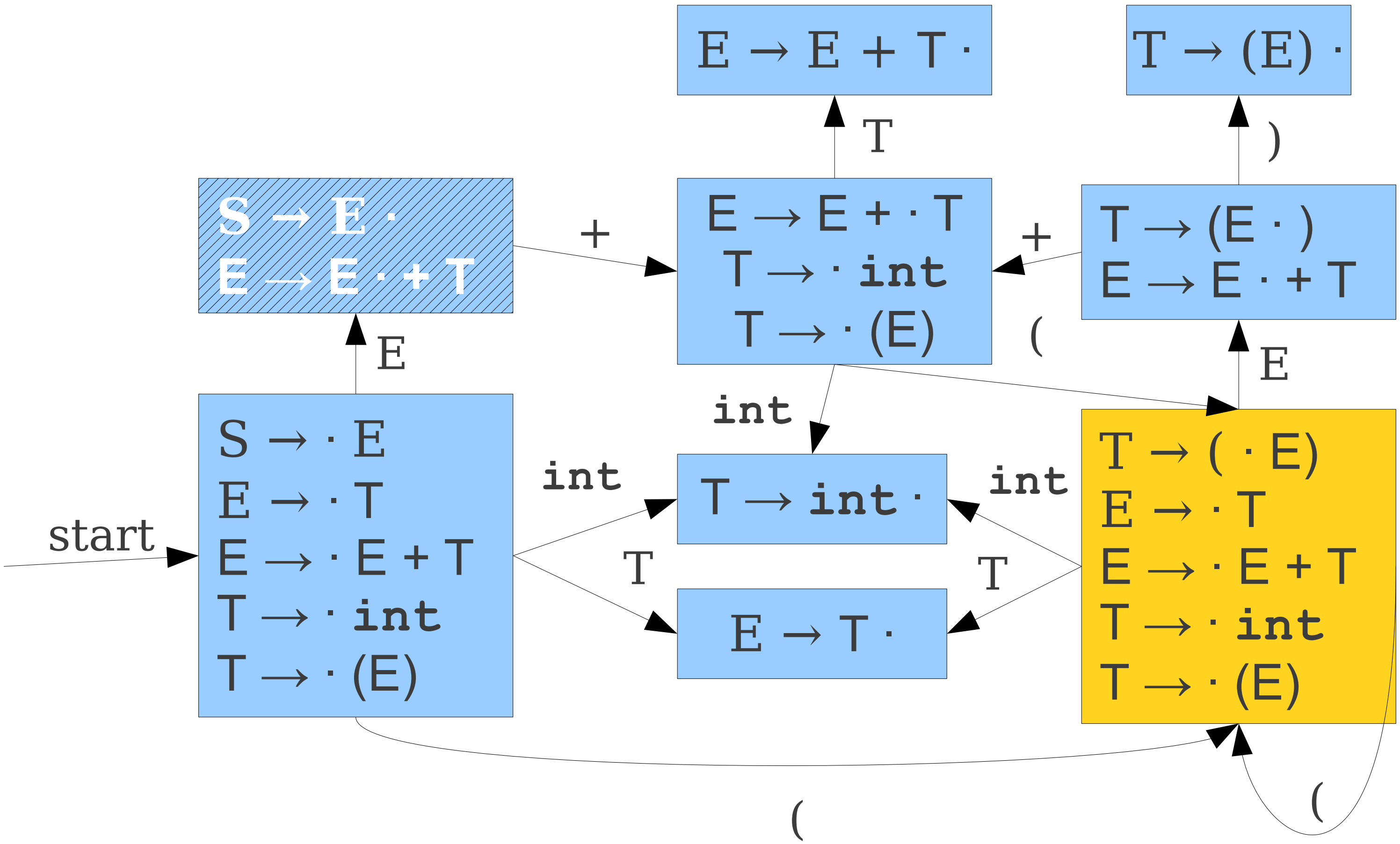
S → · E	$
E → E + · T	$
T → (· E)	$
E → · E + T	)



E + (| int + int + int) $

LR(1) Parsing: The Intuition

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

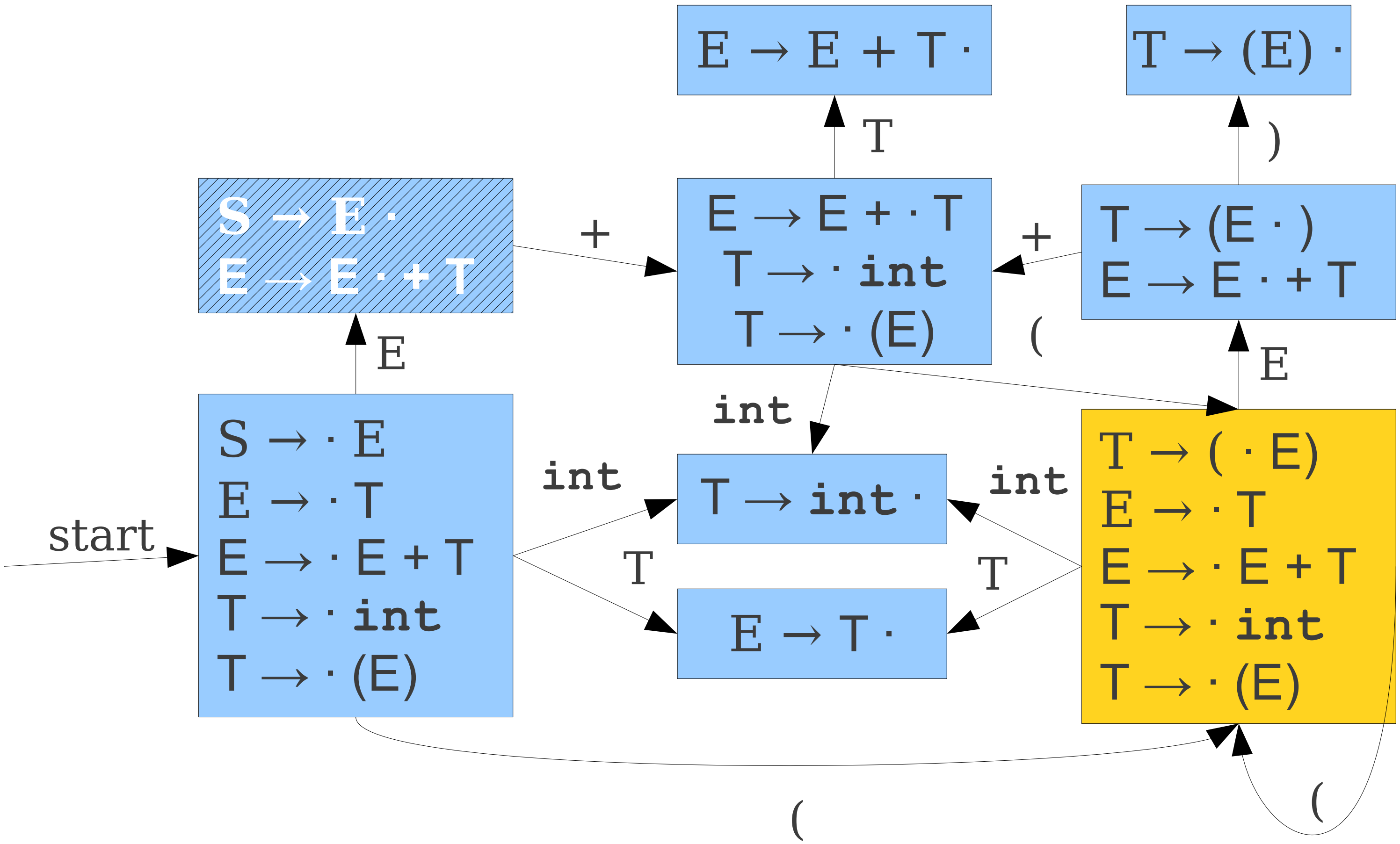


S → · E	$
E → E + · T	$
T → (· E)	$
E → · E + T	)
E → · T	+

E	+	(		int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

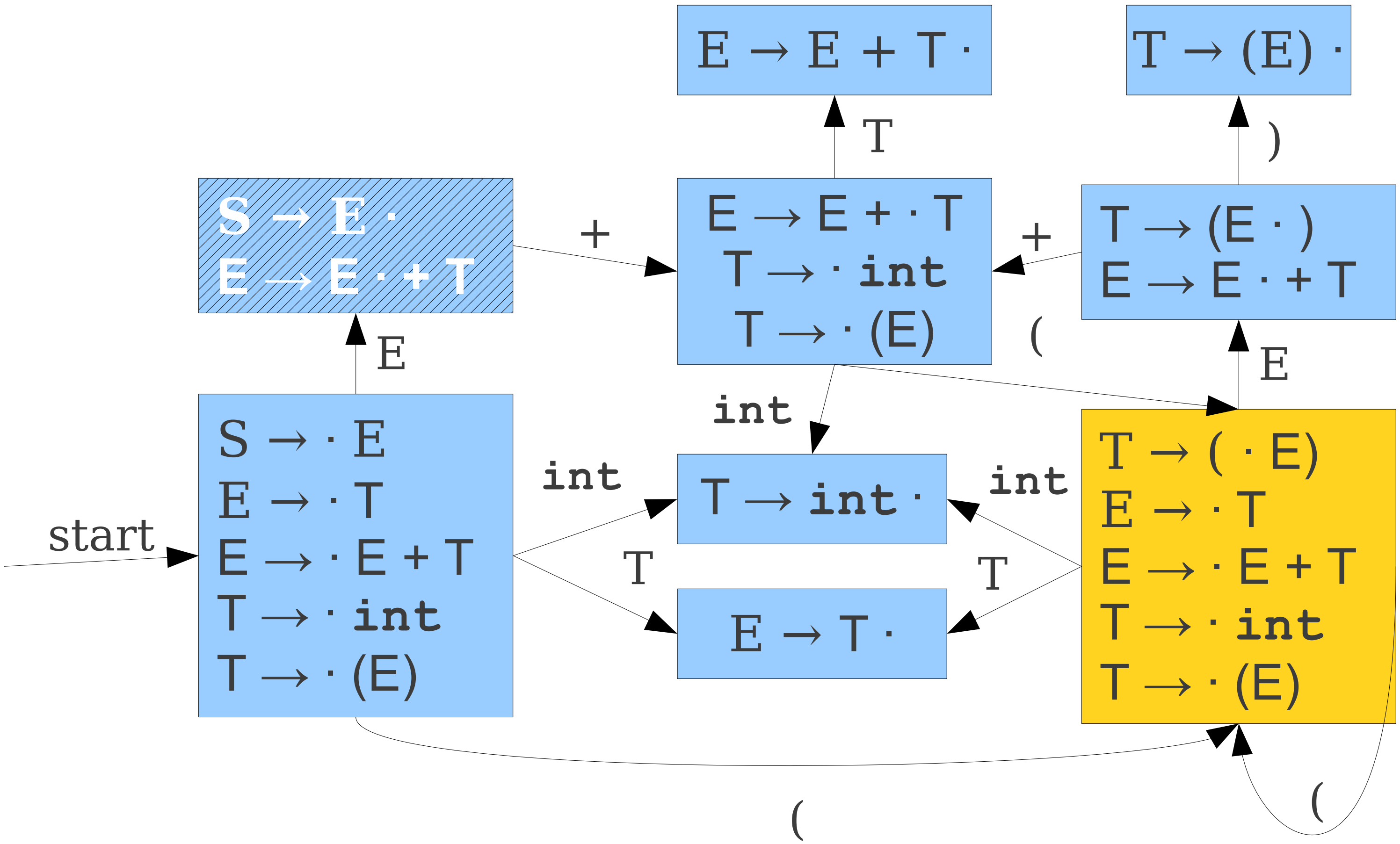


S → · E	$
E → E + · T	$
T → (· E)	$
E → · E + T	)
E → · T	+
T → · int	+

E	+	(	int	+	int	+	int	)	$

LR(1) Parsing: The Intuition

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

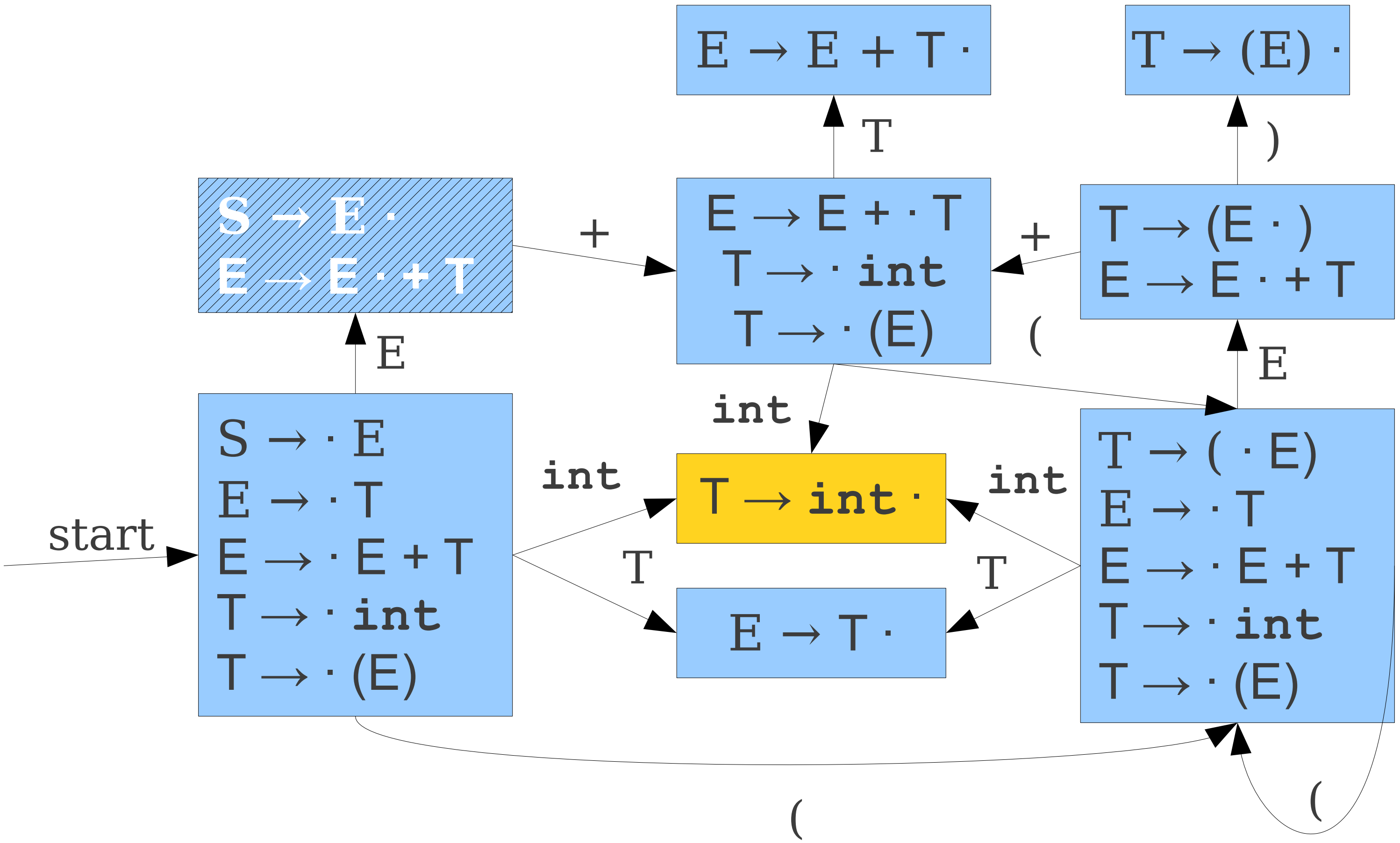


S → · E	$
E → E + · T	$
T → (· E)	$
E → · E + T	)
E → · T	+
T → · int	+

E	+	(	int		+	int	+	int	)	$

LR(1) Parsing: The Intuition

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



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

E	+	(	int	+	int	+	int	)	$

The Intuition behind LR(1)

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

Tracking Lookaheads

- How do we know what lookahead to expect at each state?
- Observation:
    - There are only finitely many productions we can be in at any point.
    - There are only finitely many positions we can be in each production.
    - **There are only finitely many lookahead sets** at each point.
- Construct an automaton to track lookaheads!

Constructing LR(1) Automata

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

Constructing LR(1) Automata

S → **E**

E → **T**

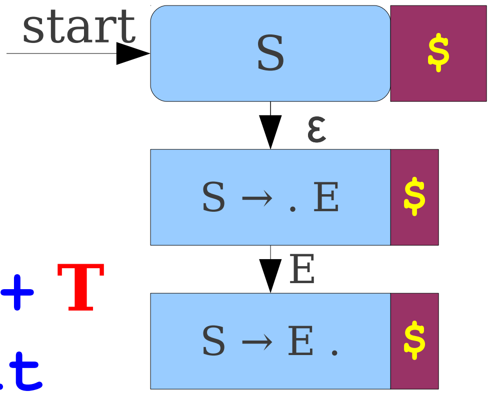
E → **E** **+** **T**

T → **`int`**

T → **`(`E`)`**

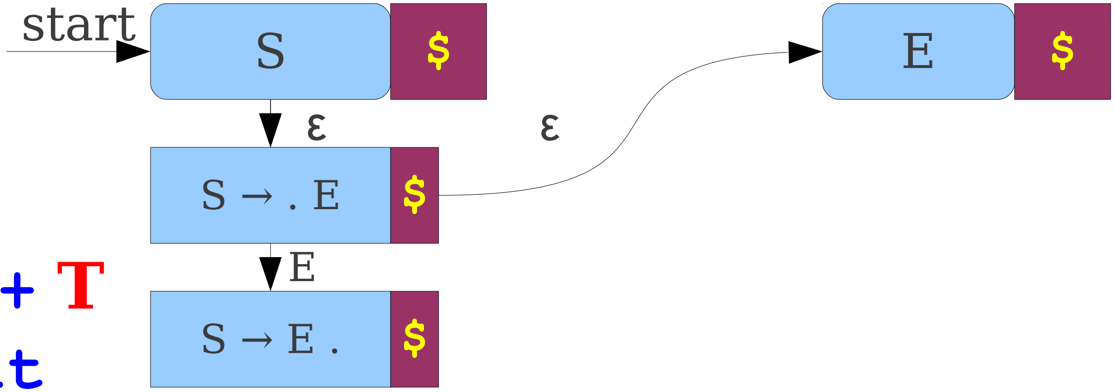
Constructing LR(1) Automata

S → E
E → T
E → E + T
T → int
T → (E)



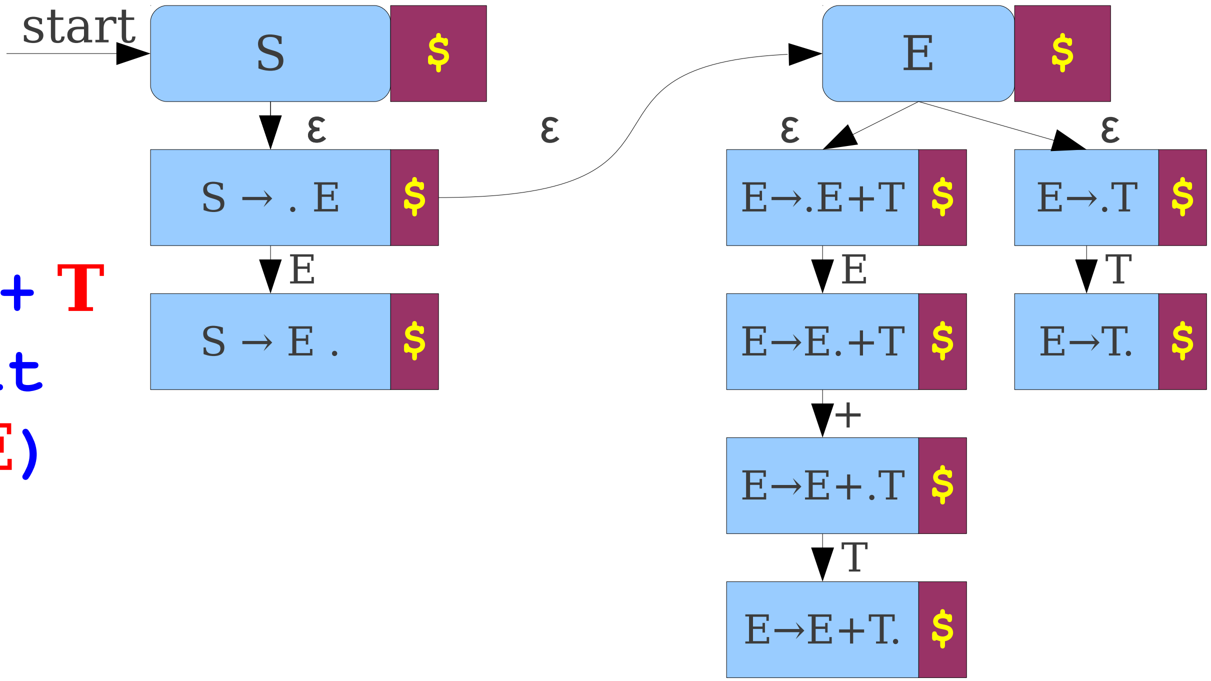
Constructing LR(1) Automata

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



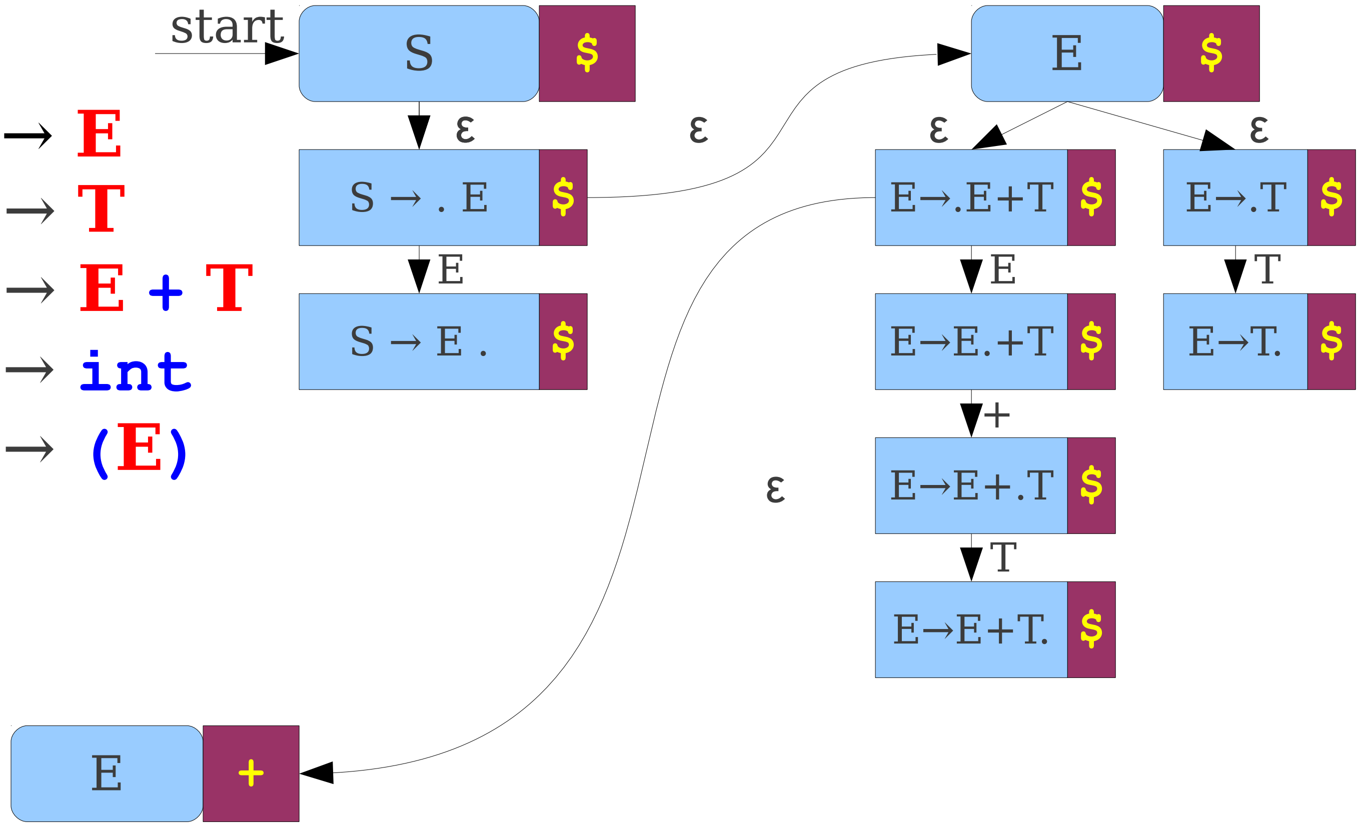
Constructing LR(1) Automata

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



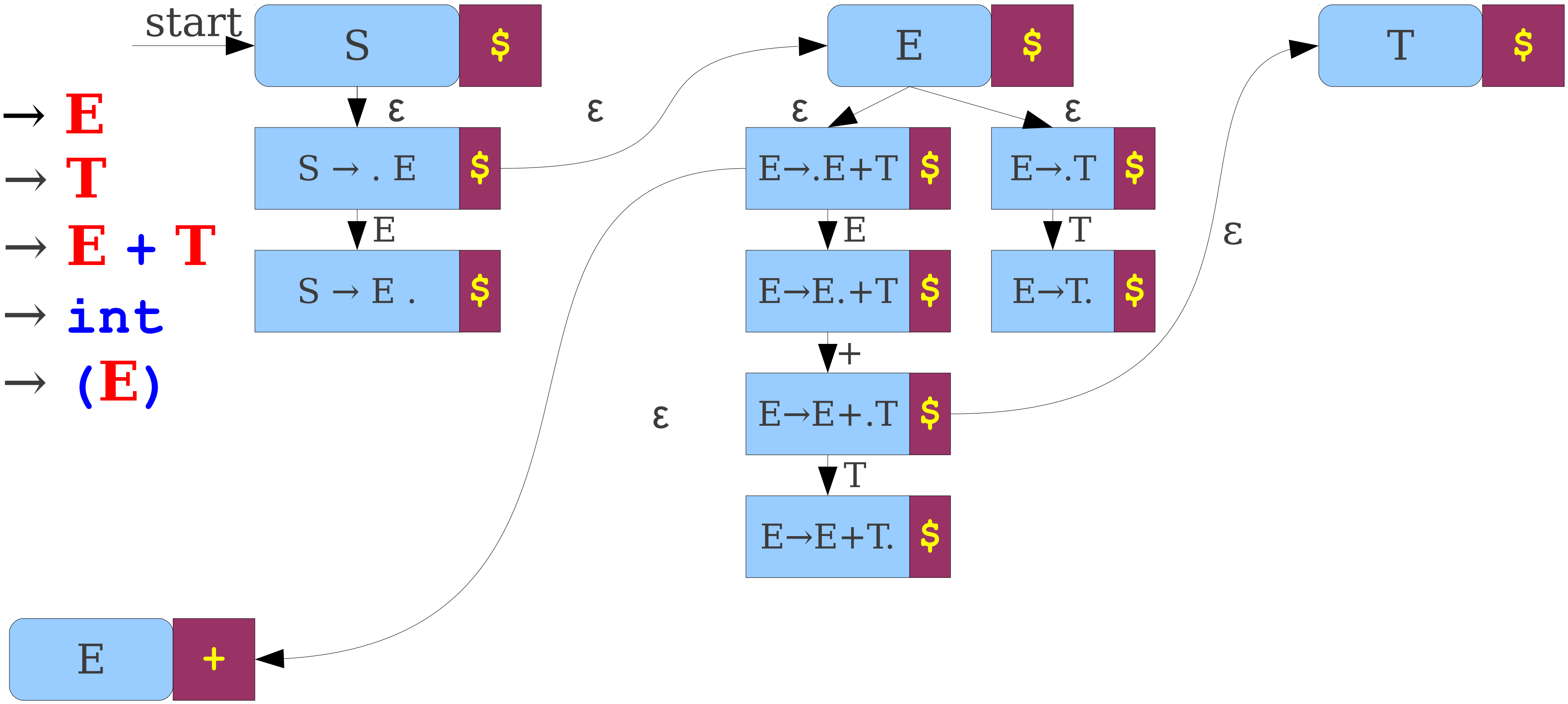
Constructing LR(1) Automata

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



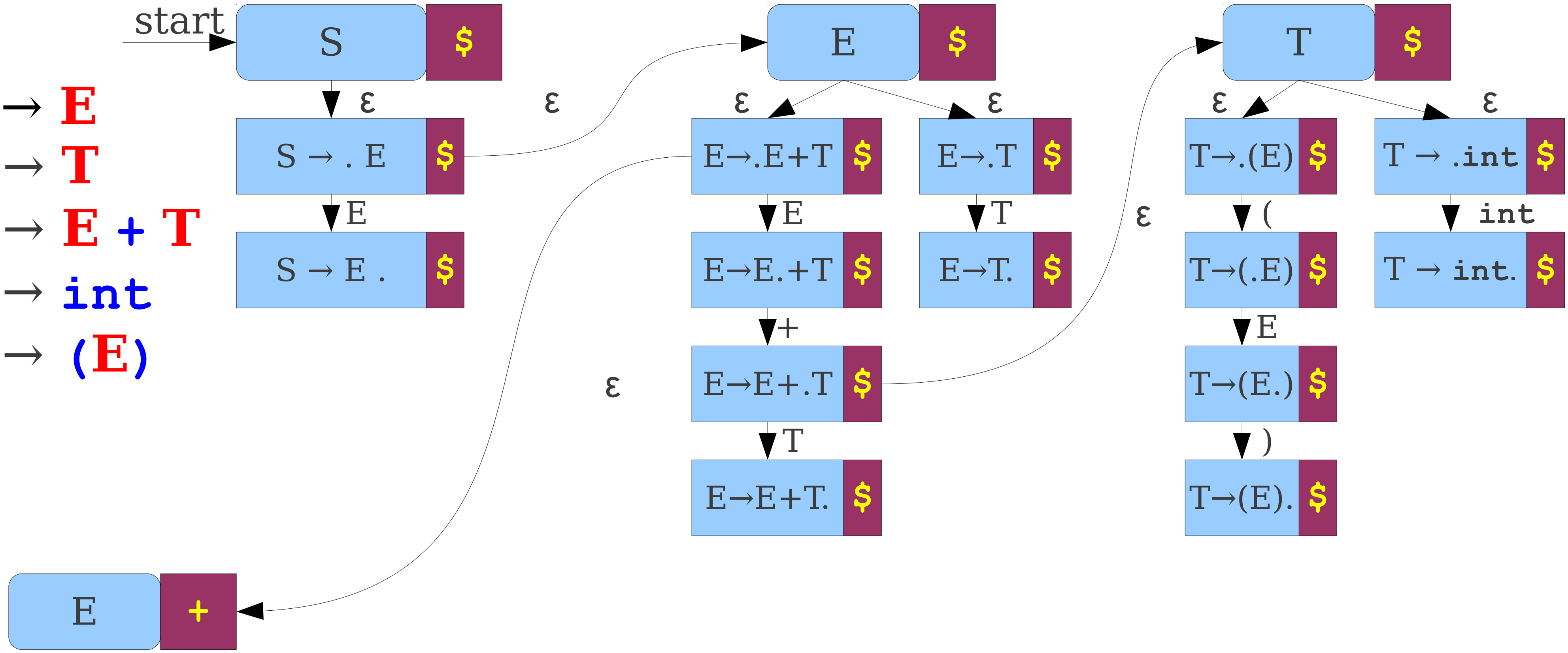
Constructing LR(1) Automata

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



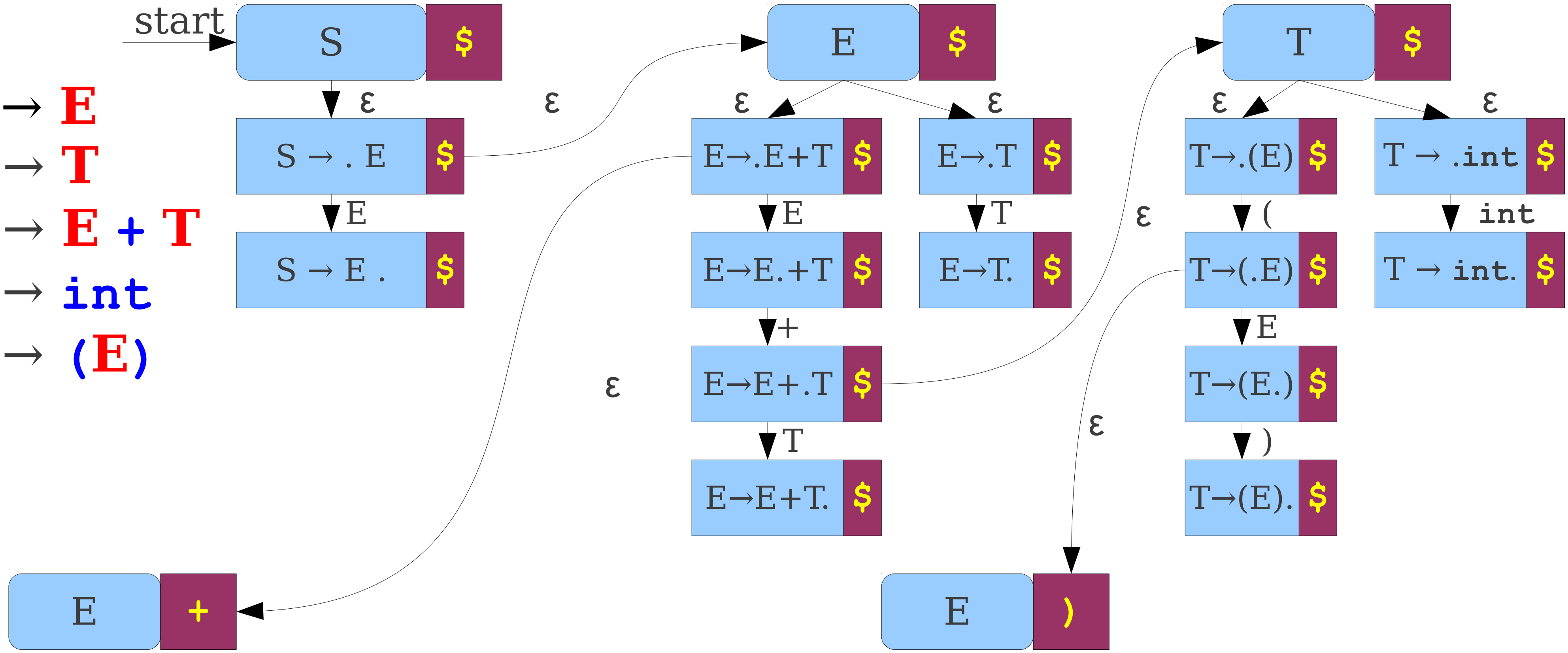
Constructing LR(1) Automata

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



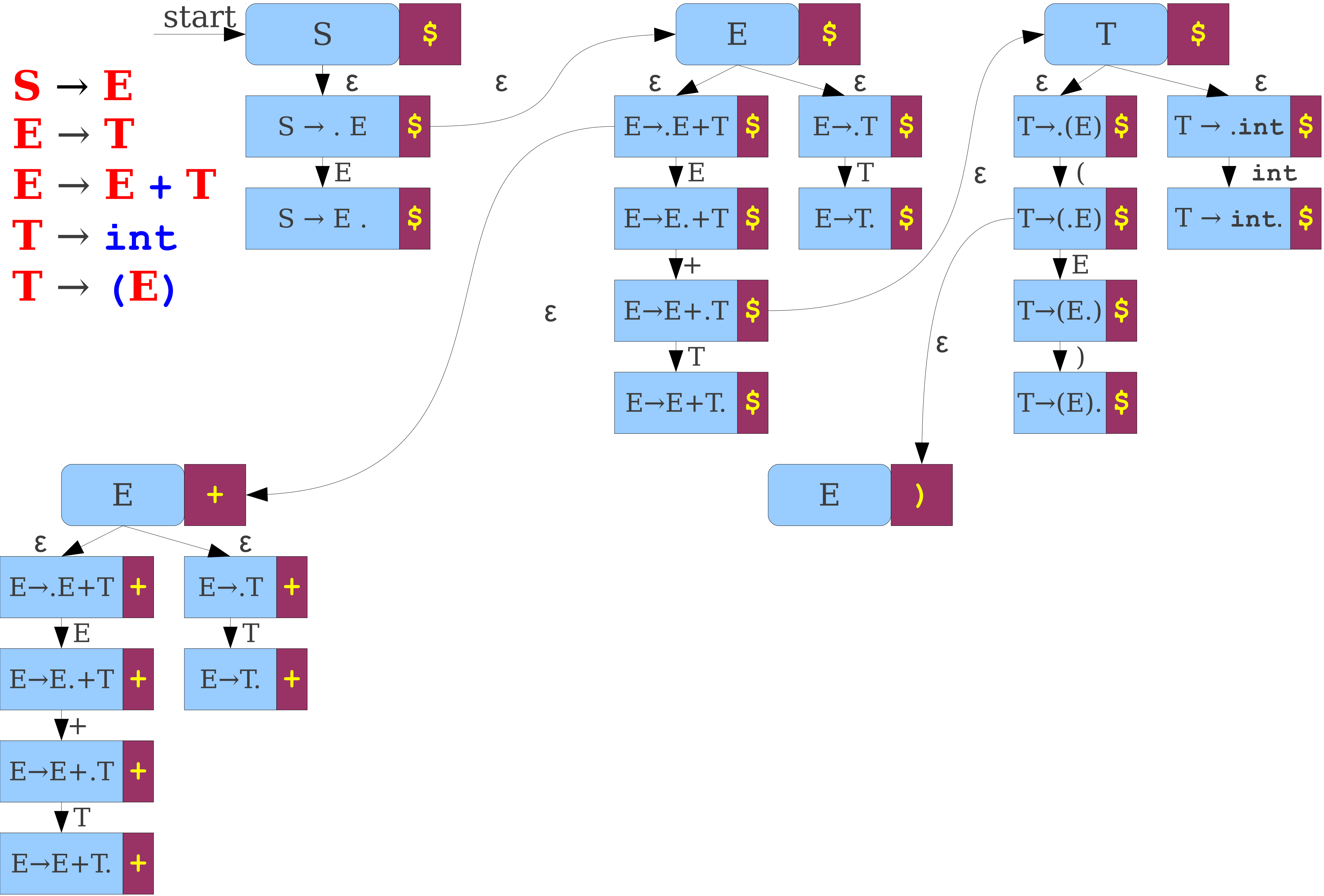
Constructing LR(1) Automata

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



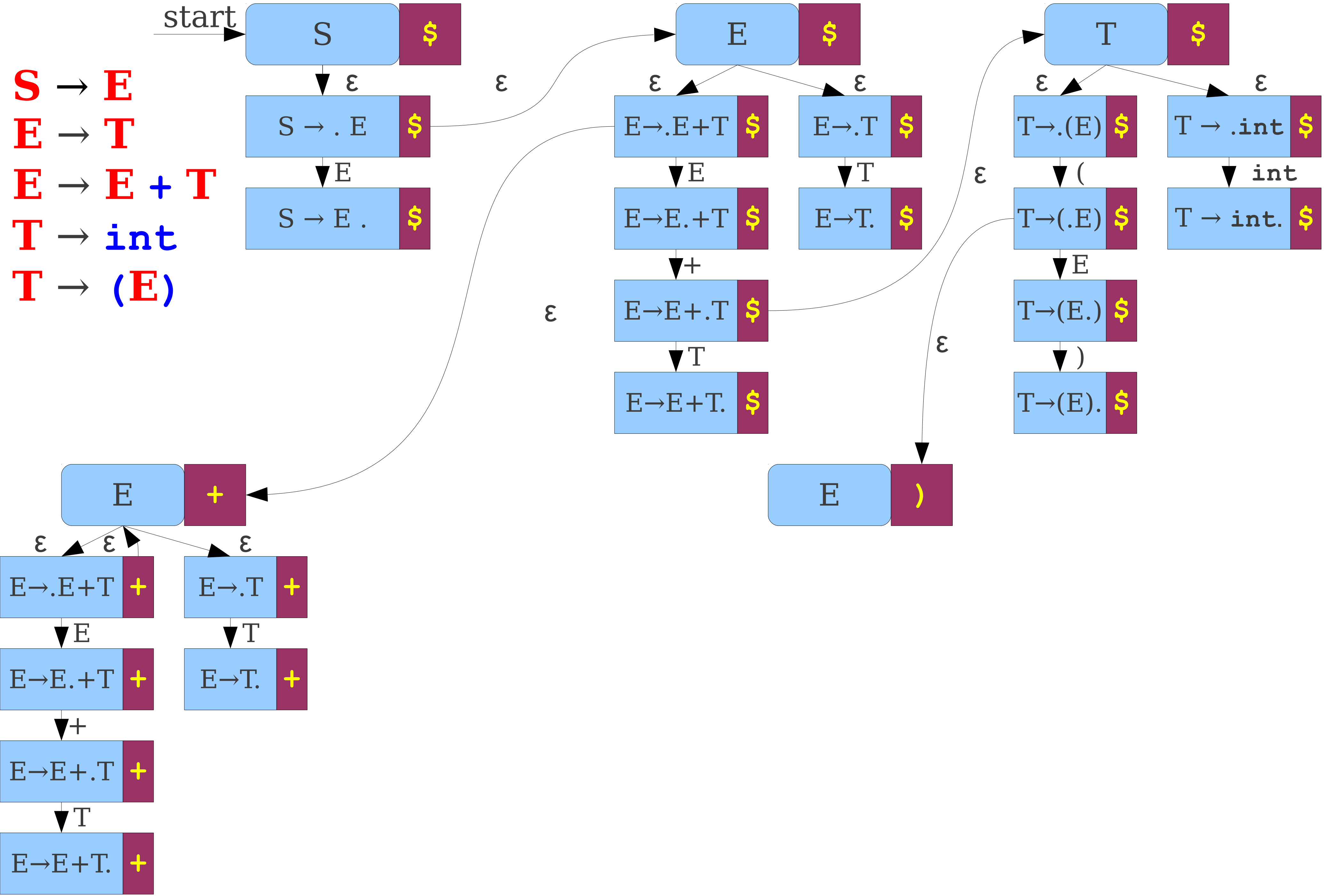
Constructing LR(1) Automata

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



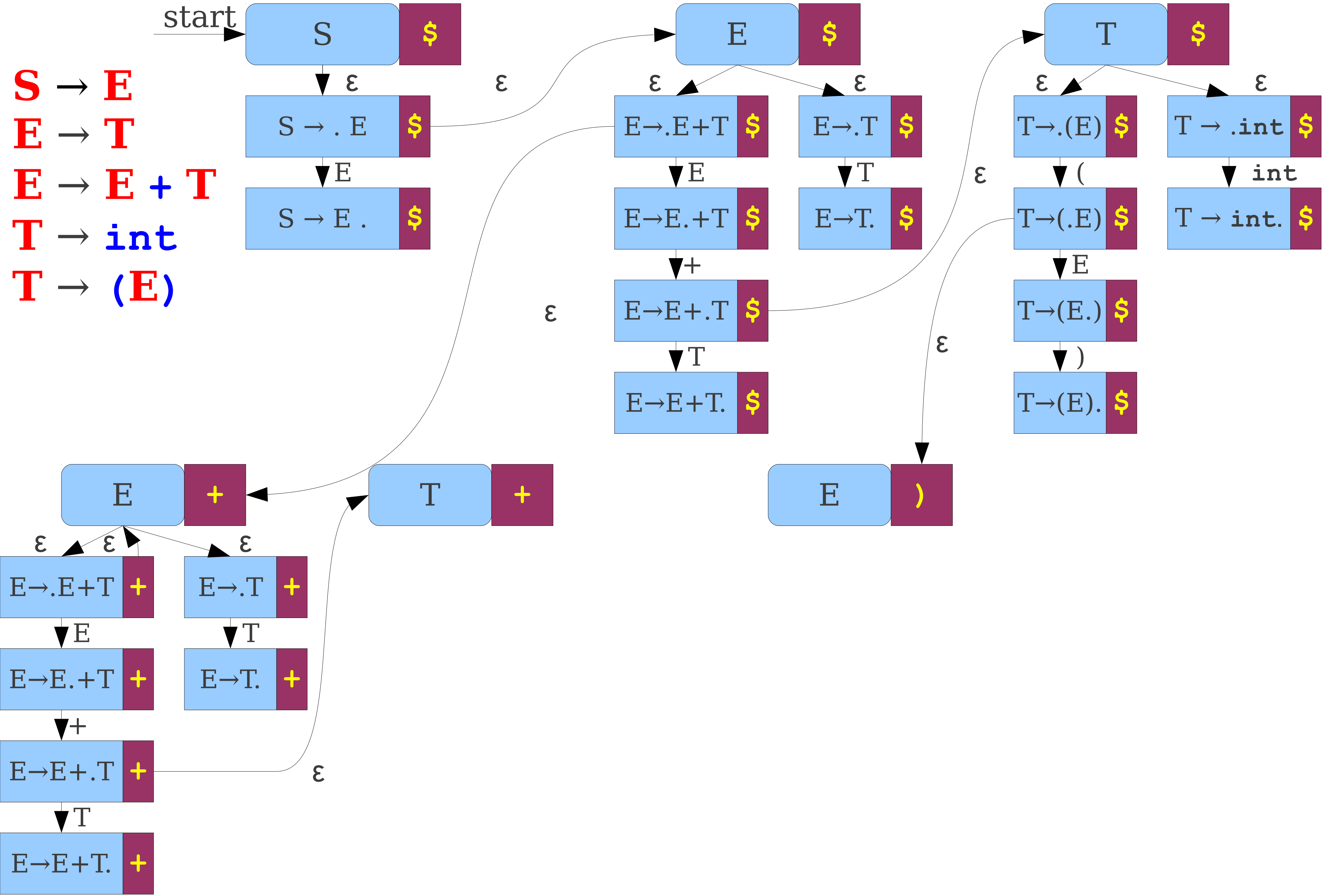
Constructing LR(1) Automata

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



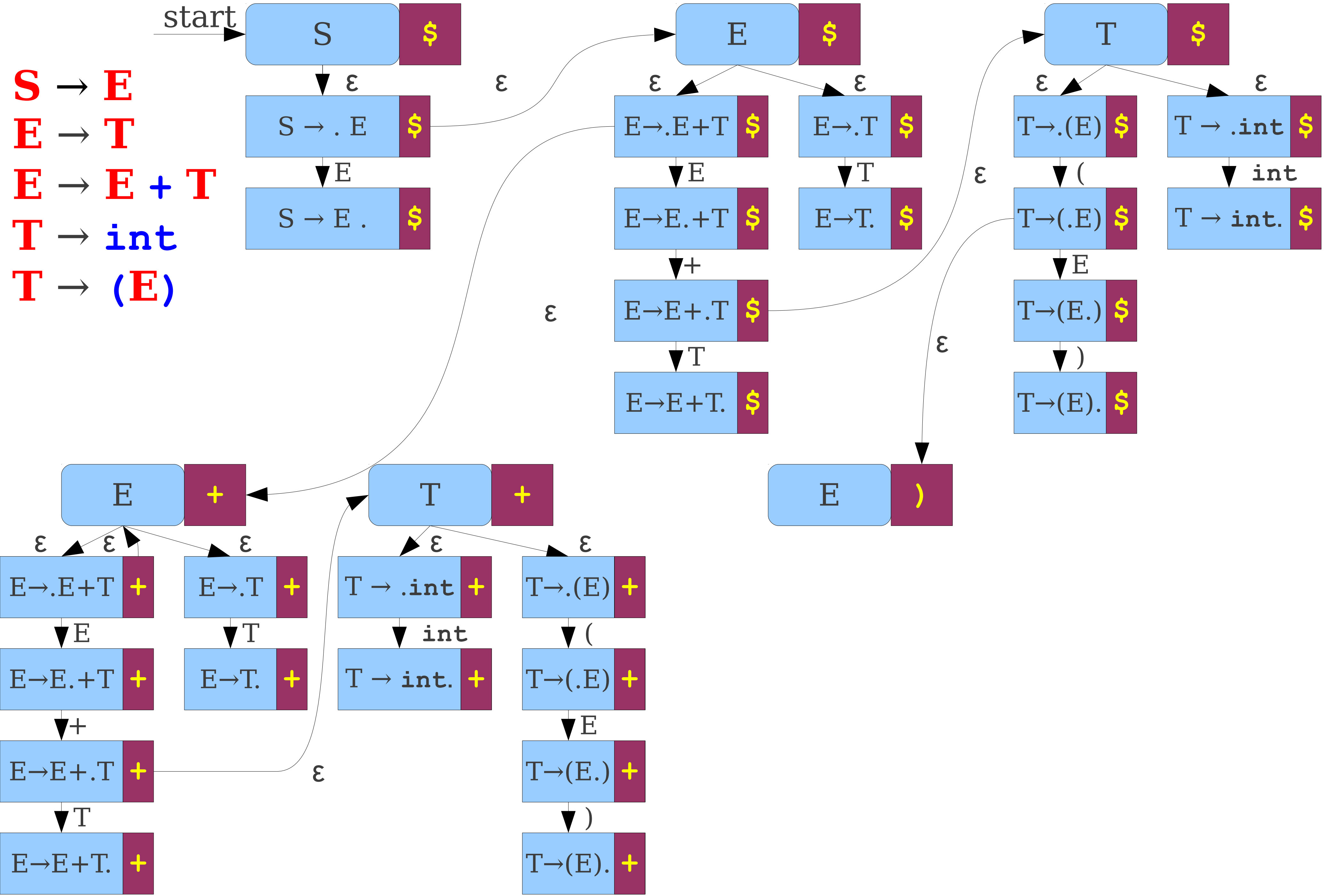
Constructing LR(1) Automata

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



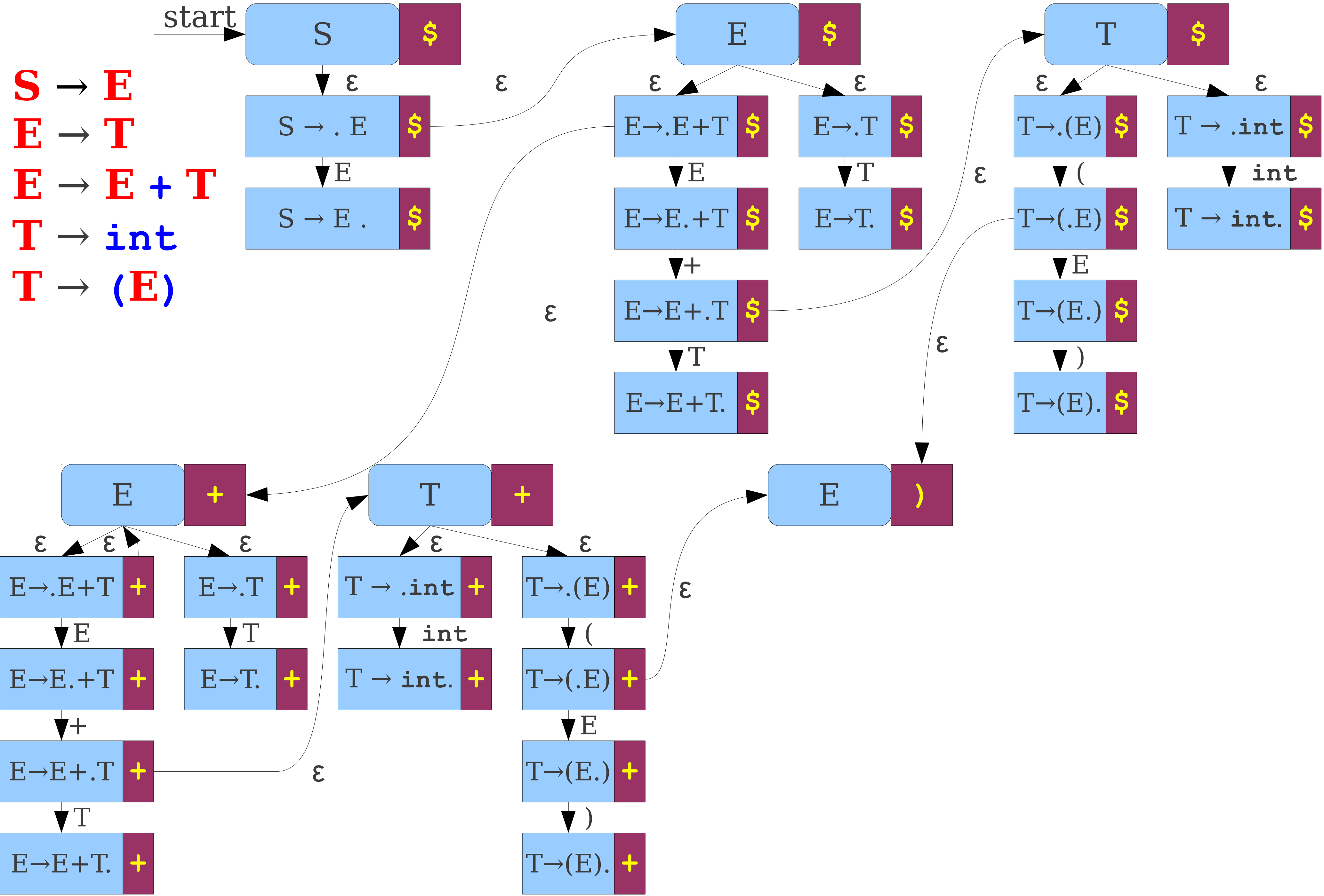
Constructing LR(1) Automata

S → E
E → T
E → E + T
T → int
T → (E)



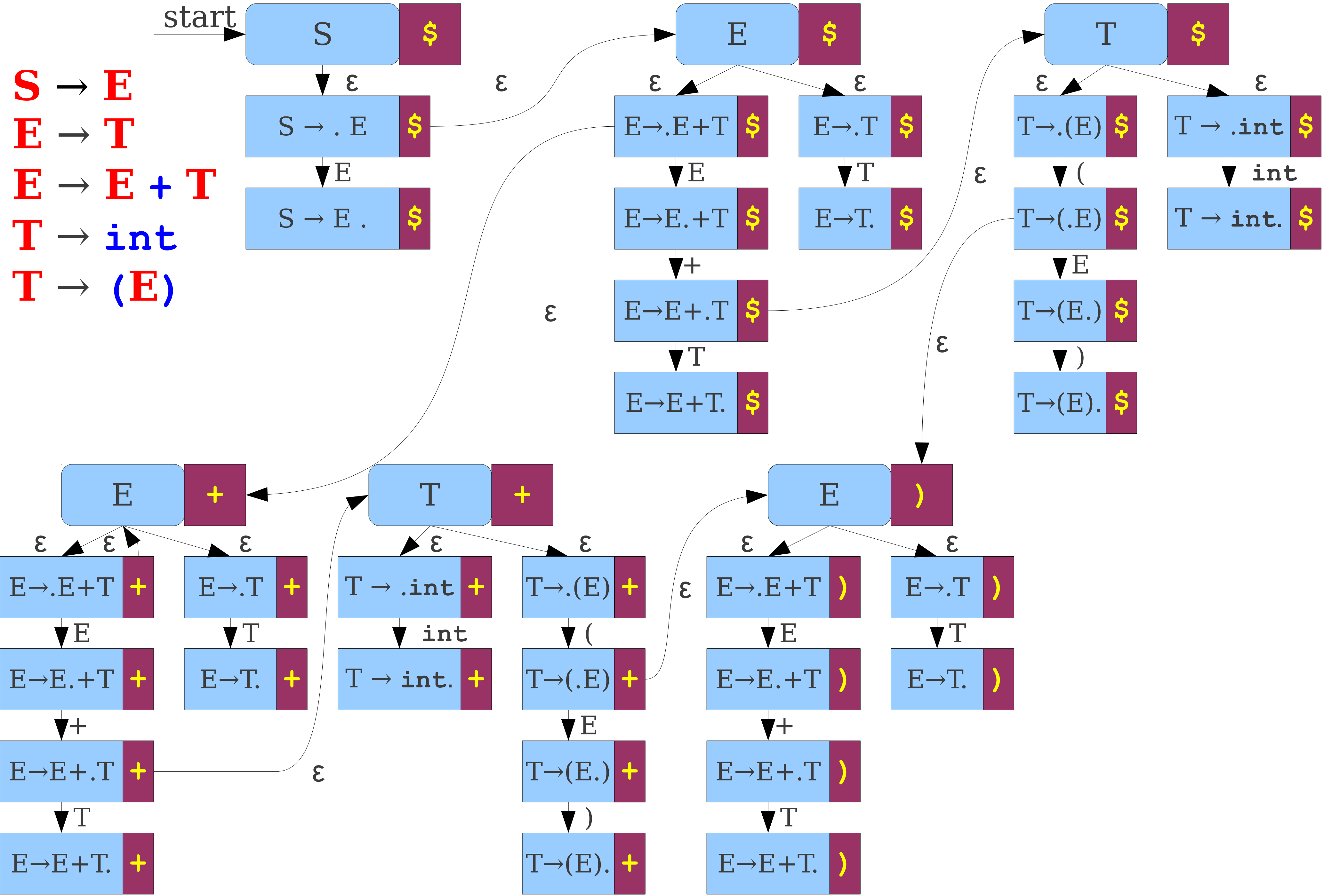
Constructing LR(1) Automata

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



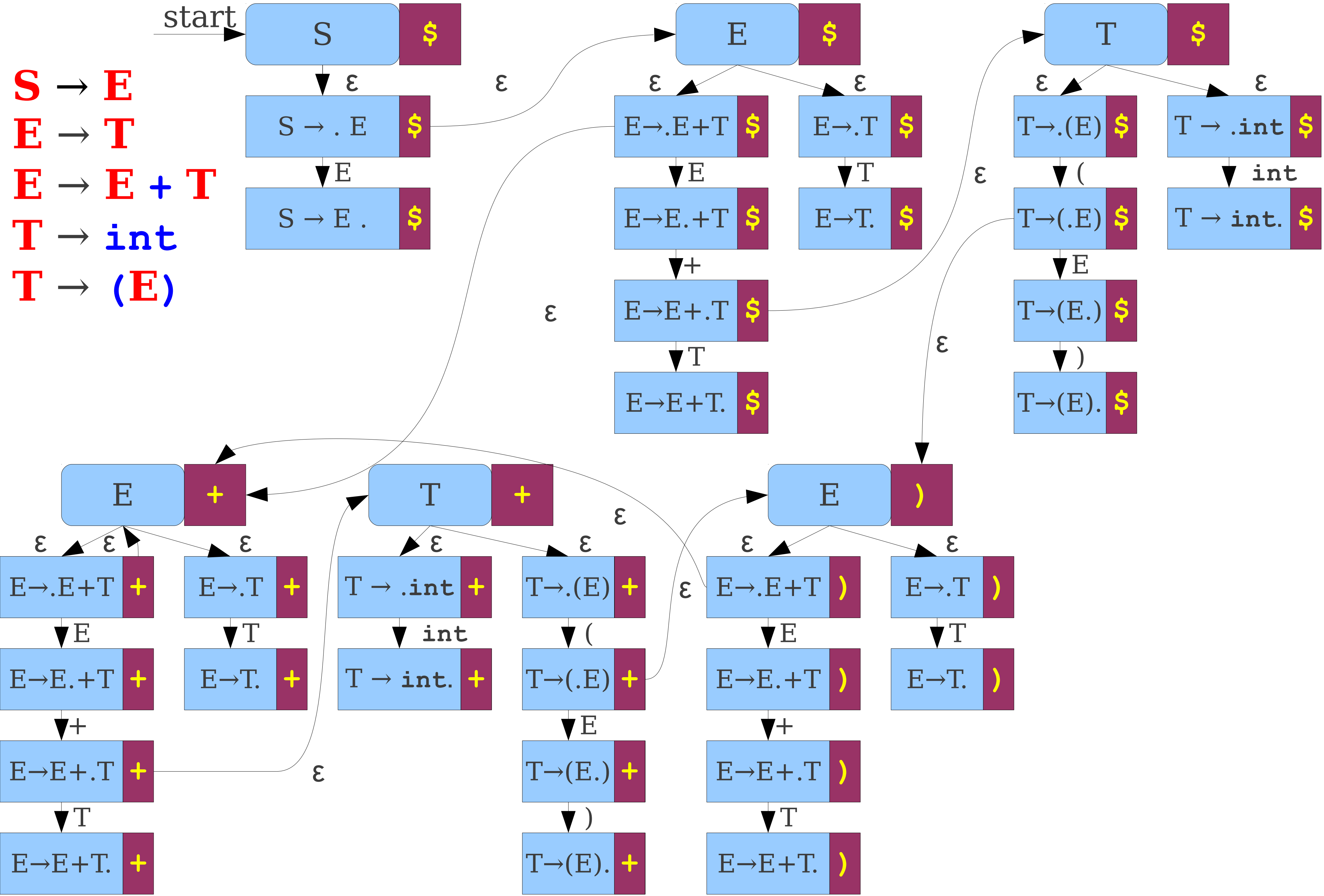
Constructing LR(1) Automata

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



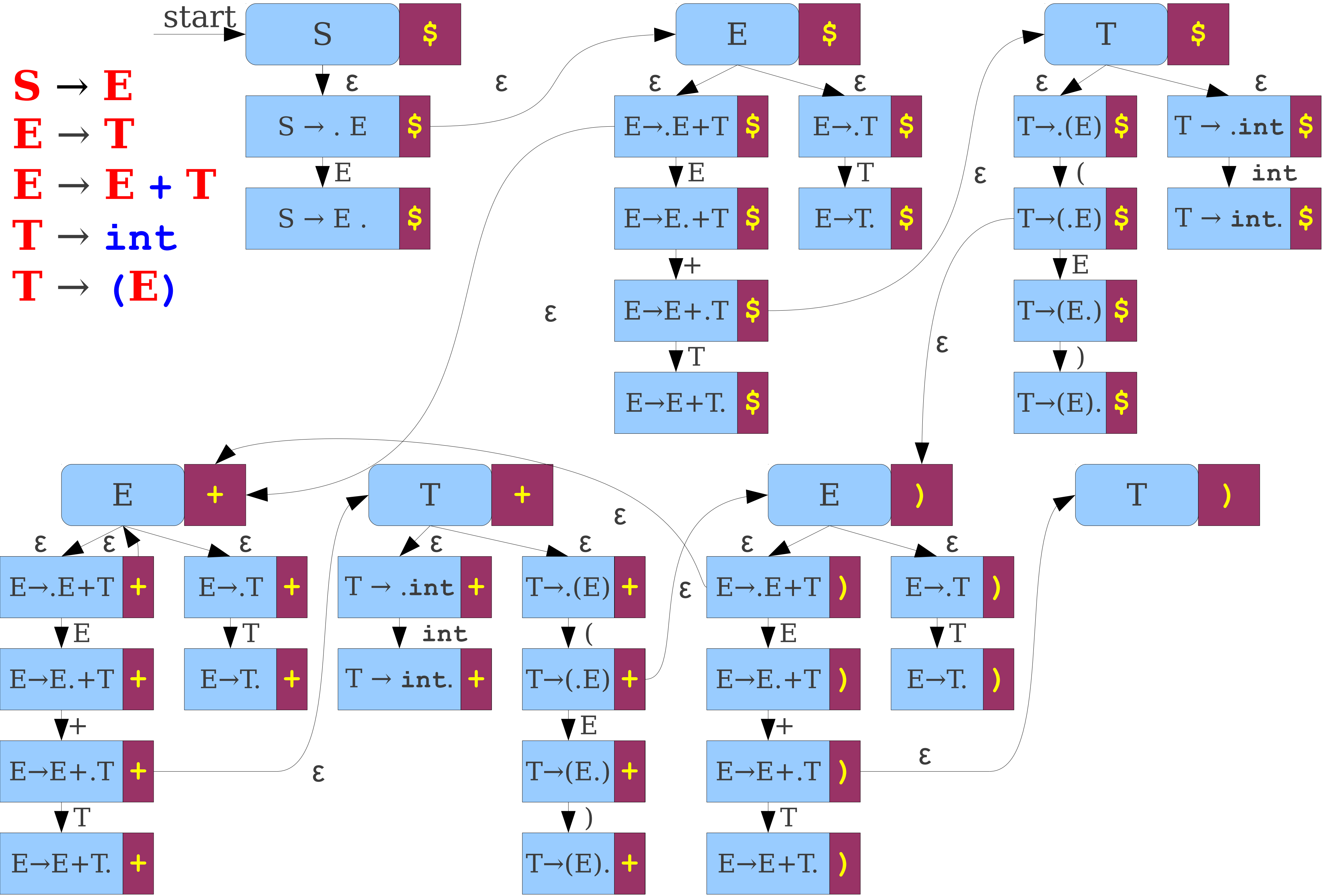
Constructing LR(1) Automata

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



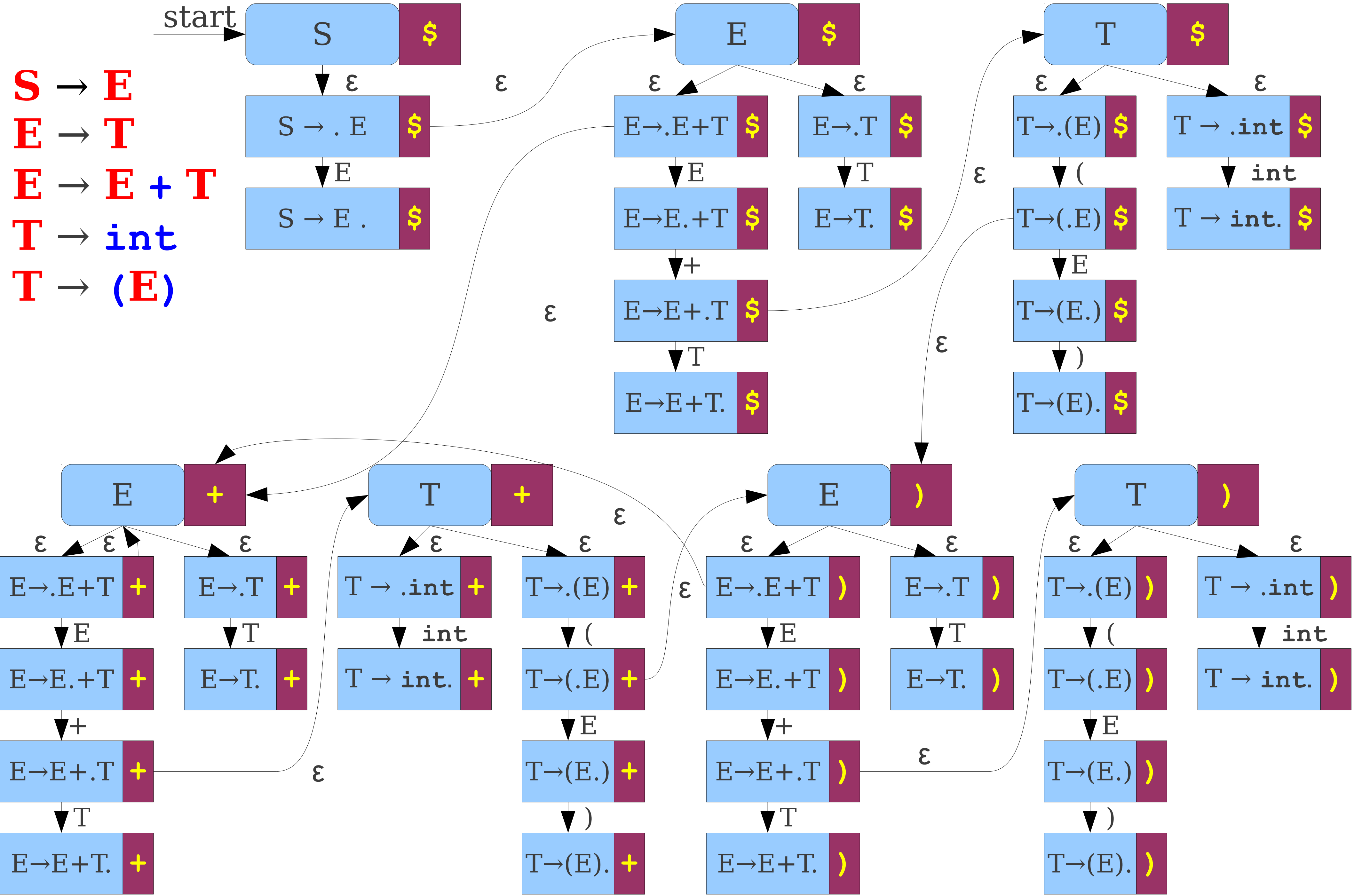
Constructing LR(1) Automata

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



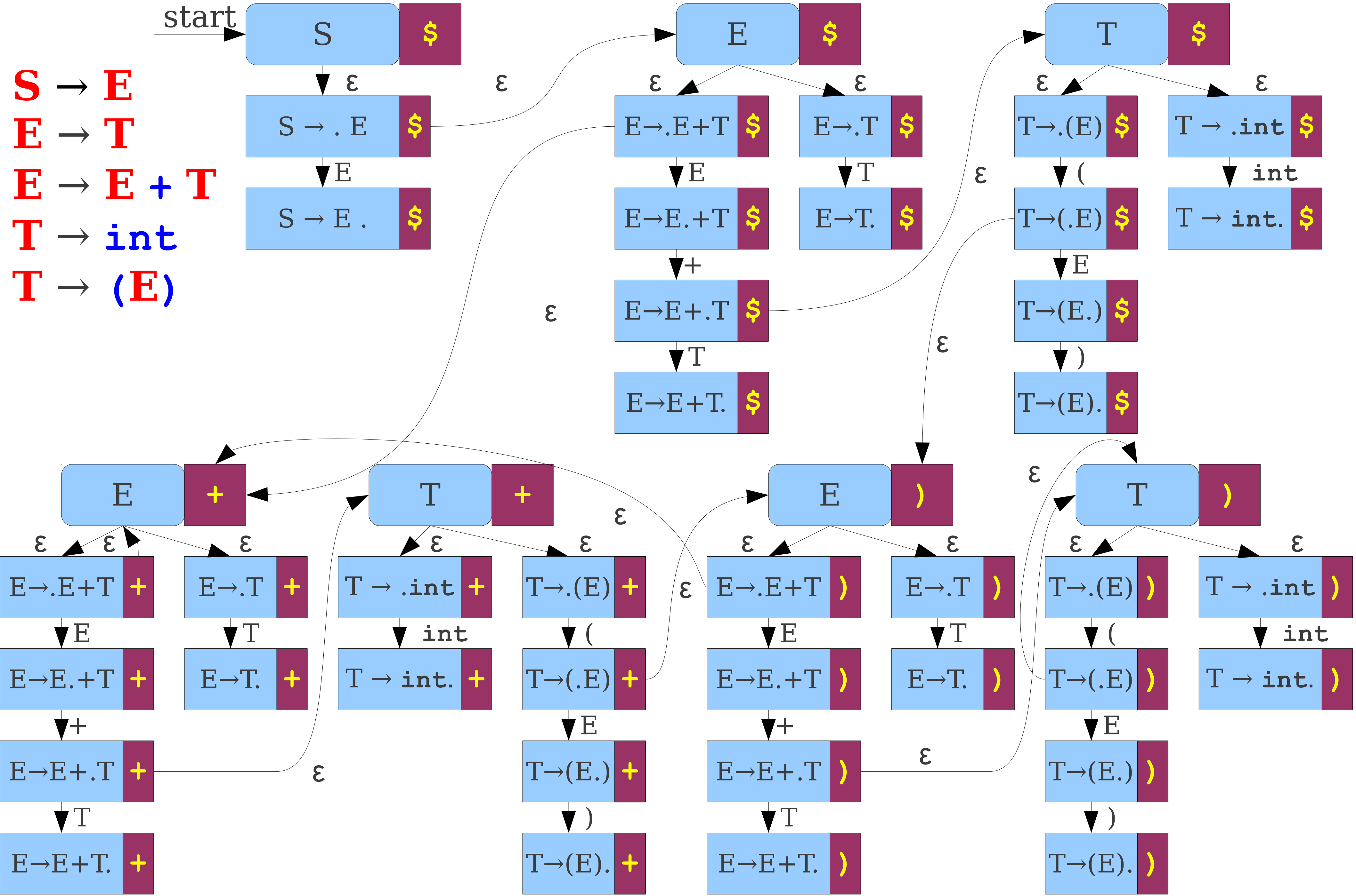
Constructing LR(1) Automata

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



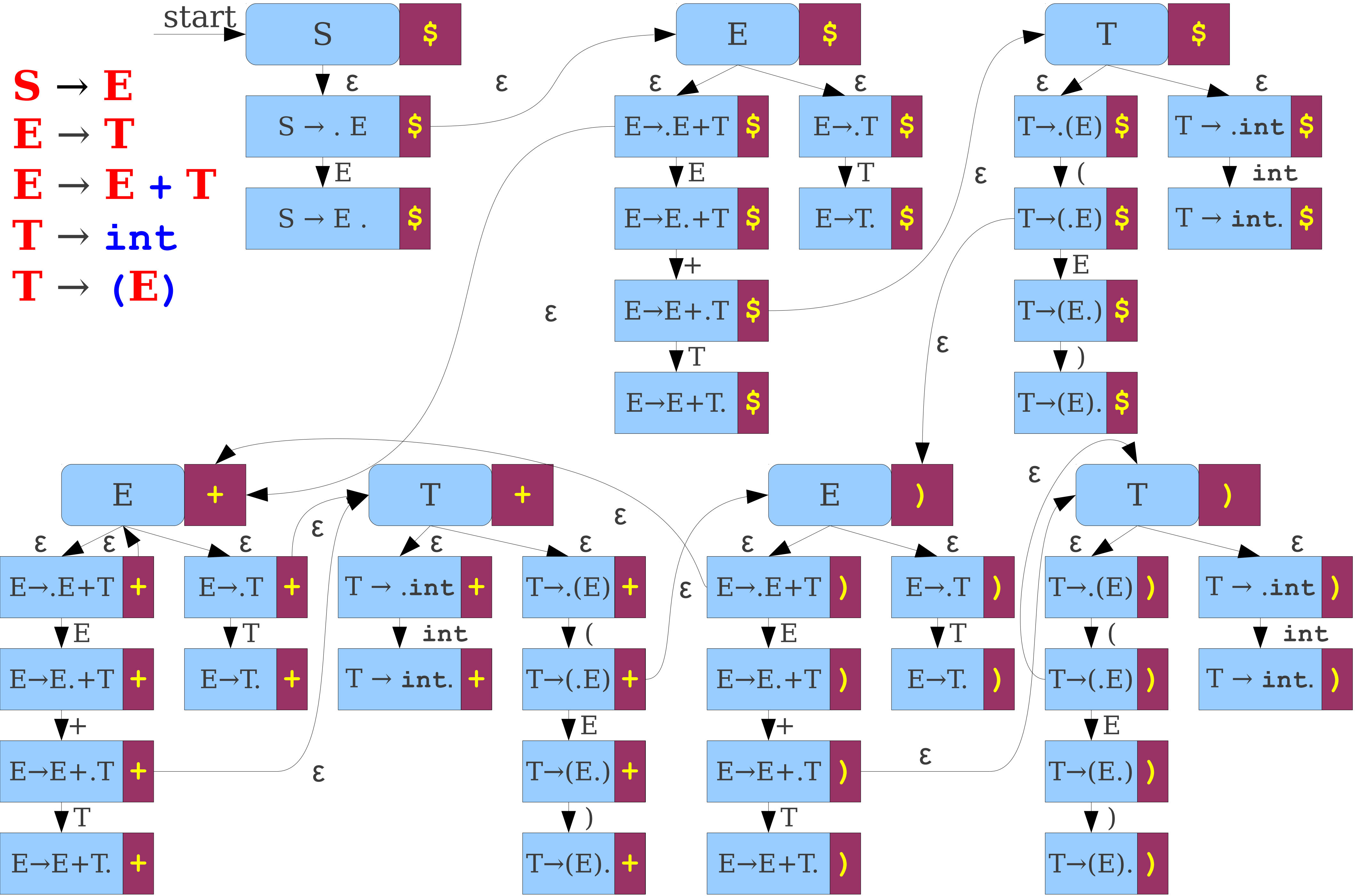
Constructing LR(1) Automata

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



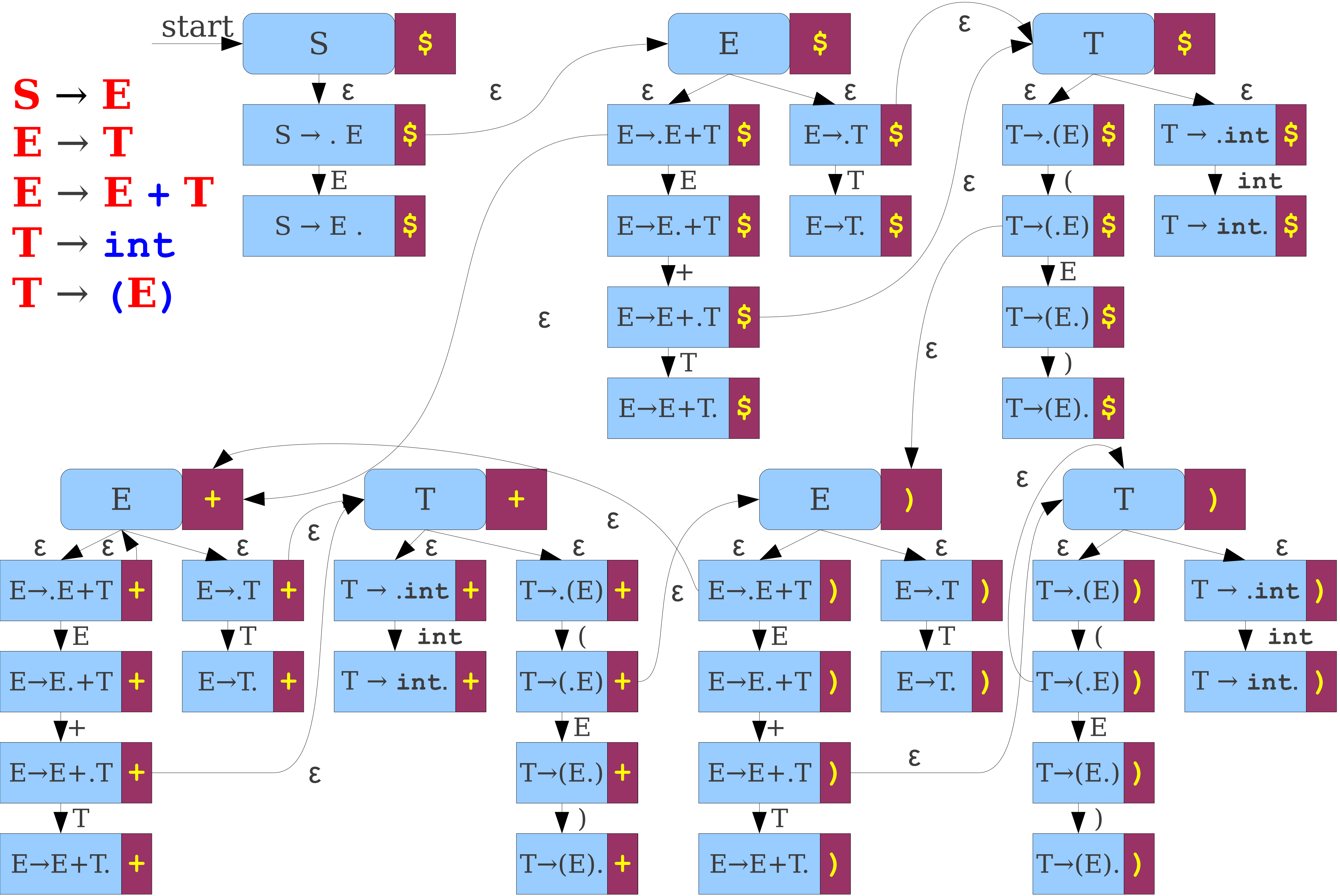
Constructing LR(1) Automata

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



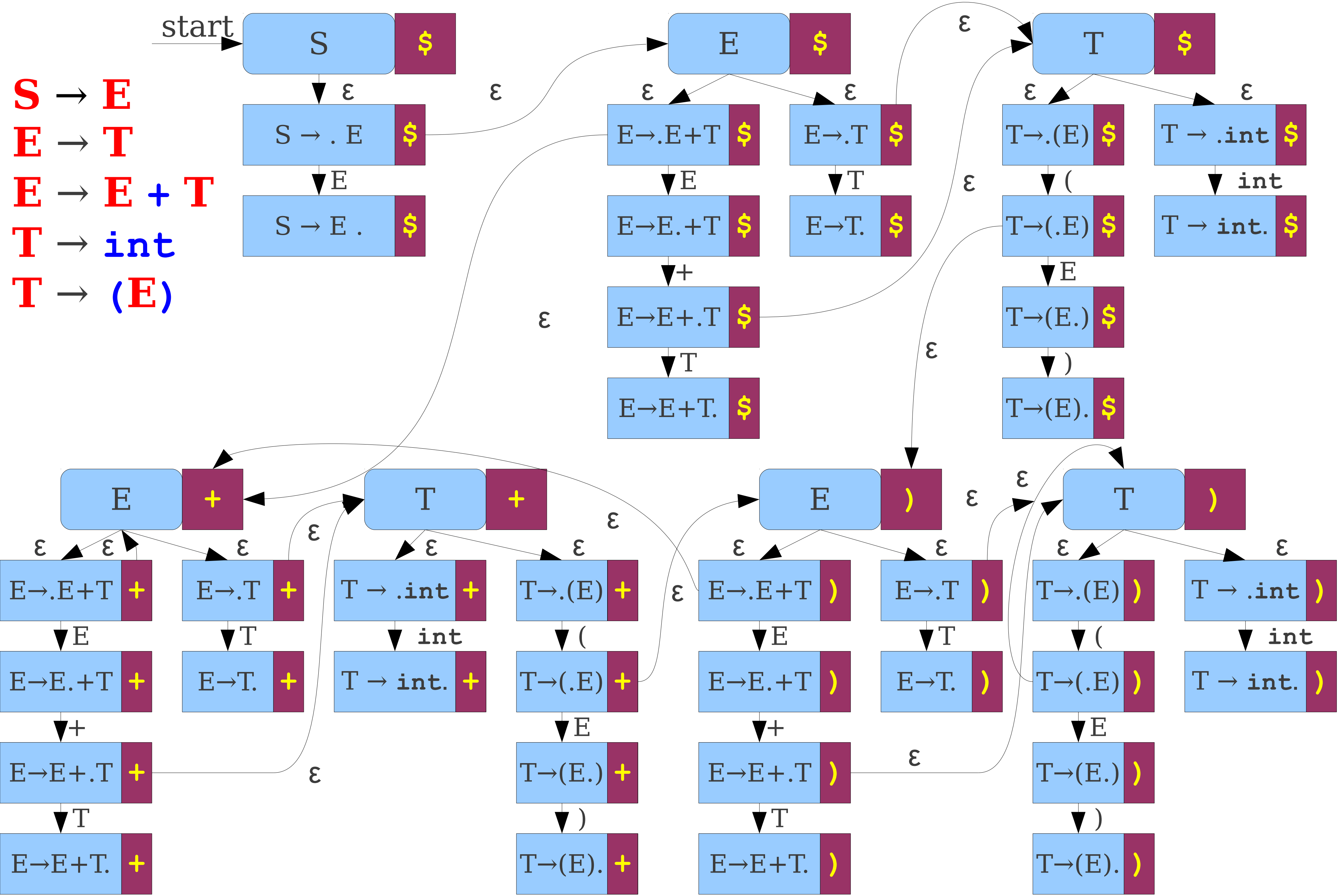
Constructing LR(1) Automata

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



Constructing LR(1) Automata

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

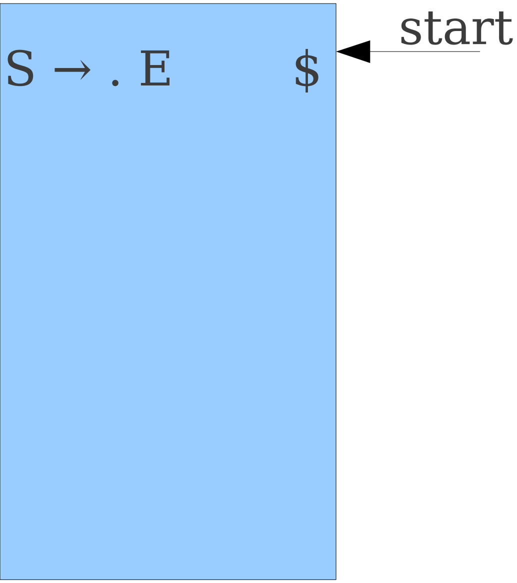


Constructing LR(1) Automata

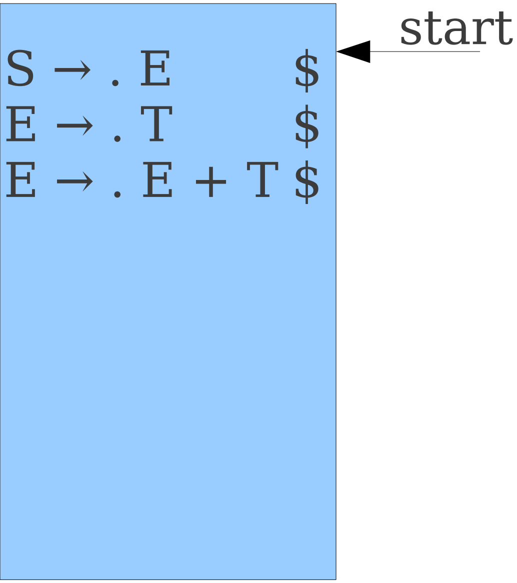
- Begin with a state $\mathbf{S}$ [`$`].
- For each state $\mathbf{A}$ [`t`], for each production $\mathbf{A} \to \gamma$:
  - Construct states $\mathbf{A} \to \alpha\cdot\omega$ [`t`] for all possible ways of splitting $\gamma = \alpha\omega$.
  - Add an ε-transition from $\mathbf{A}$ [`t`] to each of these states.
  - Add transitions on x between $\mathbf{A} \to \alpha \cdot x\omega$ [`t`] and $\mathbf{A} \to \alpha x \cdot \omega$ [`t`]
- For each state $\mathbf{A} \to \alpha \cdot \mathbf{B}\omega$ [`t`], add an ε-transition from $\mathbf{A} \to \alpha \cdot \mathbf{B}\omega$ [`t`] to $\mathbf{B}$ [`r`] for each terminal $\mathtt{r} \in \text{FIRST*}(\omega\mathtt{t})$.

Deterministic LR(1) Automata

Deterministic LR(1) Automata



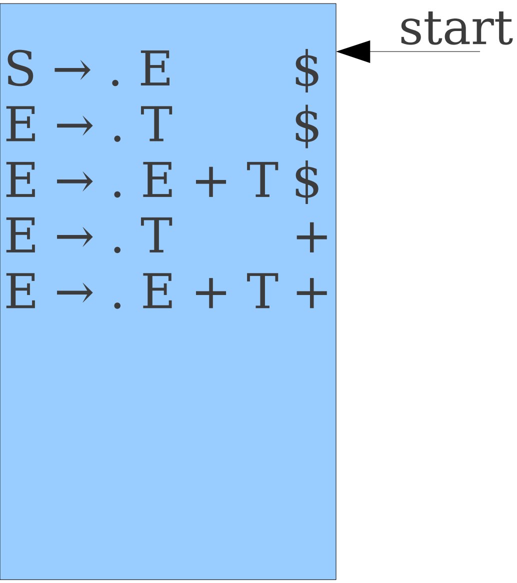
Deterministic LR(1) Automata



start

S → . E $
E → . T $
E → . E + T $

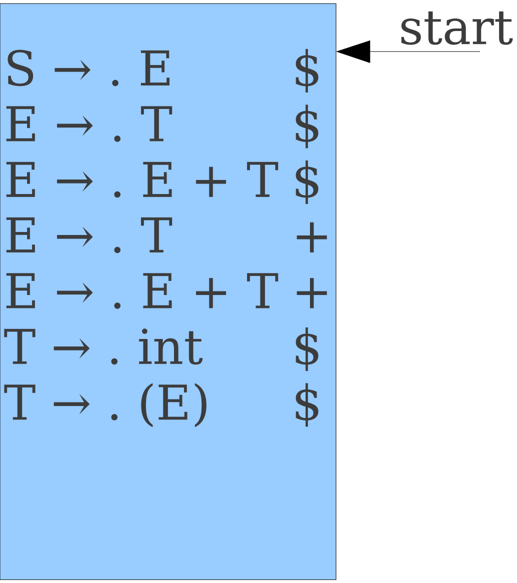
Deterministic LR(1) Automata



start

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

Deterministic LR(1) Automata



S → . E $

E → . T $

E → . E + T $

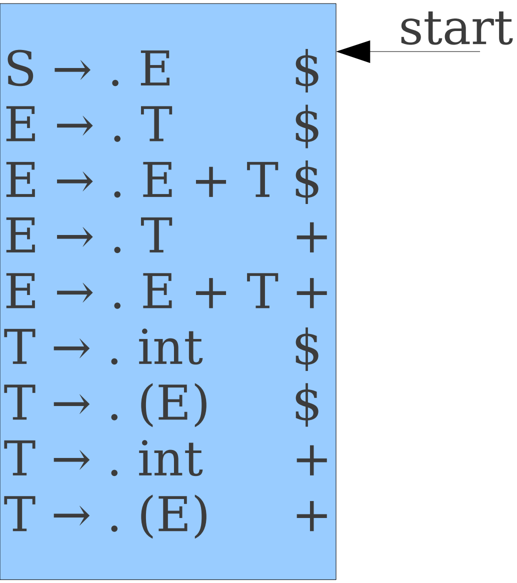
E → . T +

E → . E + T +

T → . int $

T → . (E) $

Deterministic LR(1) Automata



start

S → . E $

E → . T $

E → . E + T $

E → . T +

E → . E + T +

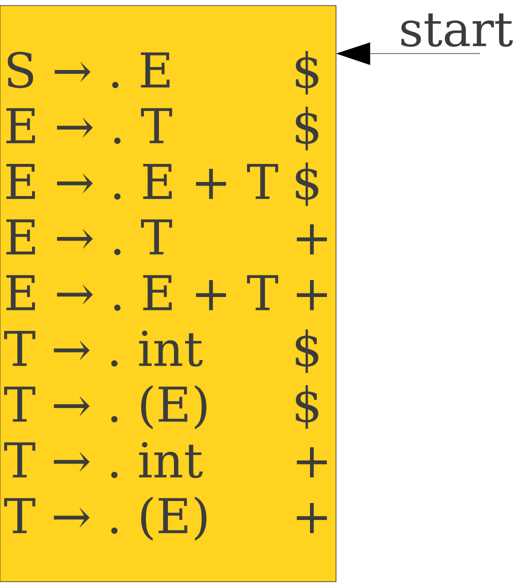
T → . int $

T → . (E) $

T → . int +

T → . (E) +

Deterministic LR(1) Automata



S → . E $

E → . T $

E → . E + T $

E → . T +

E → . E + T +

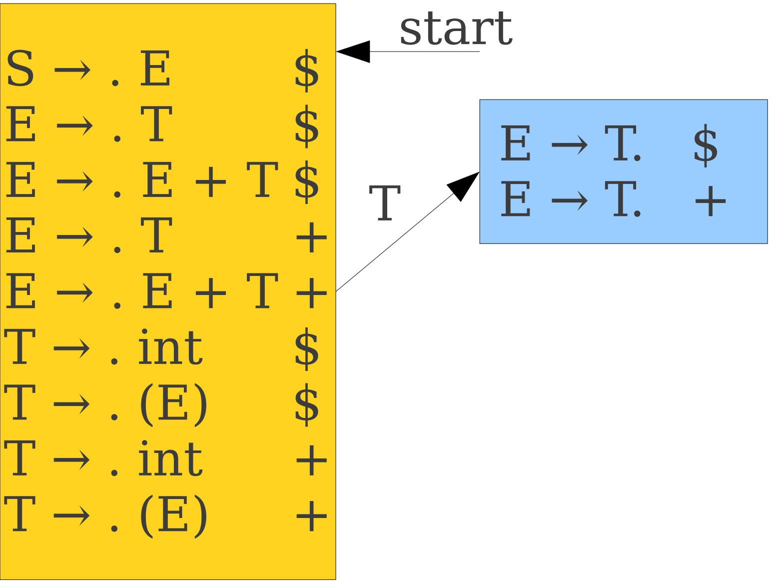
T → . int $

T → . (E) $

T → . int +

T → . (E) +

Deterministic LR(1) Automata



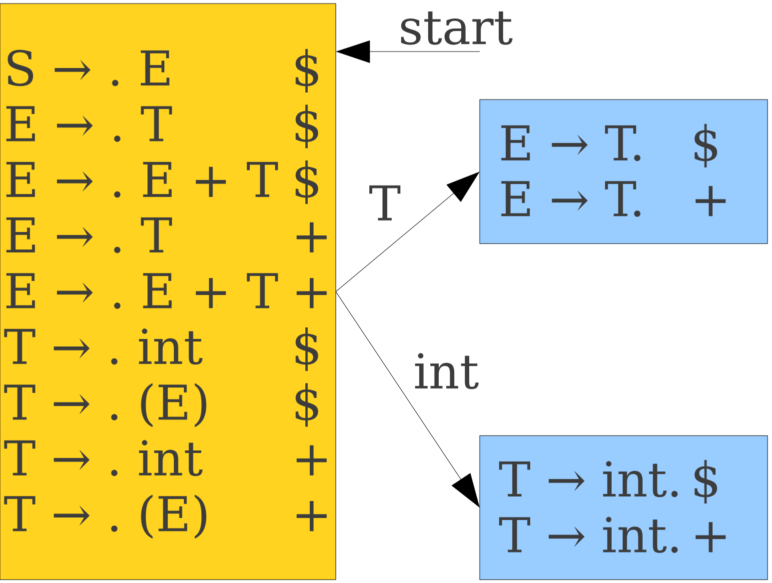
start

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

T

E → T. $
E → T. +

Deterministic LR(1) Automata



start

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

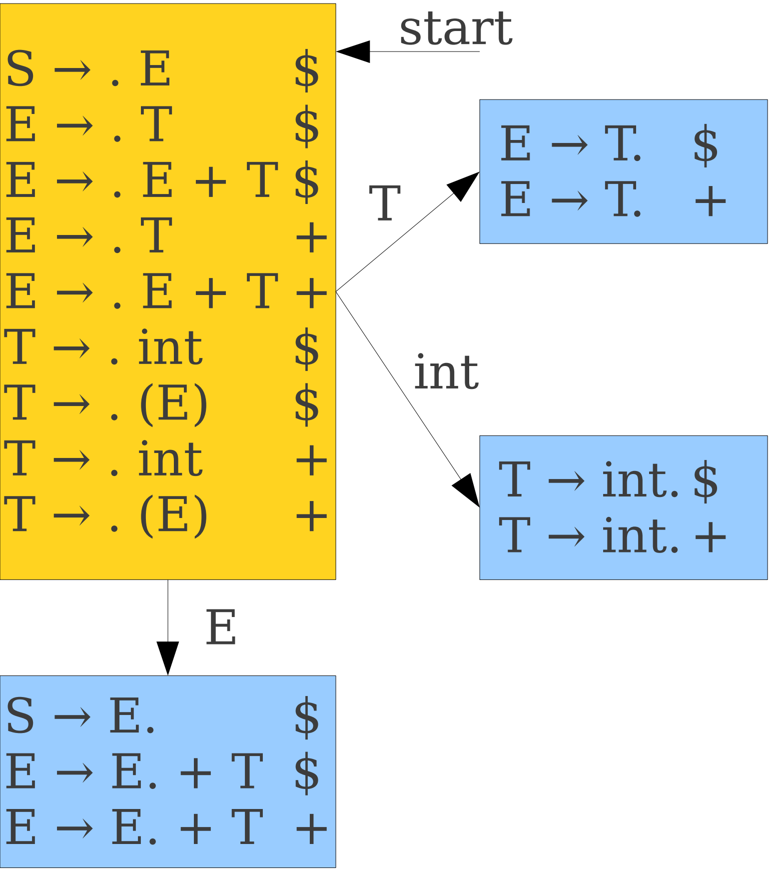
T

E → T. $
E → T. +

int

T → int. $
T → int. +

Deterministic LR(1) Automata



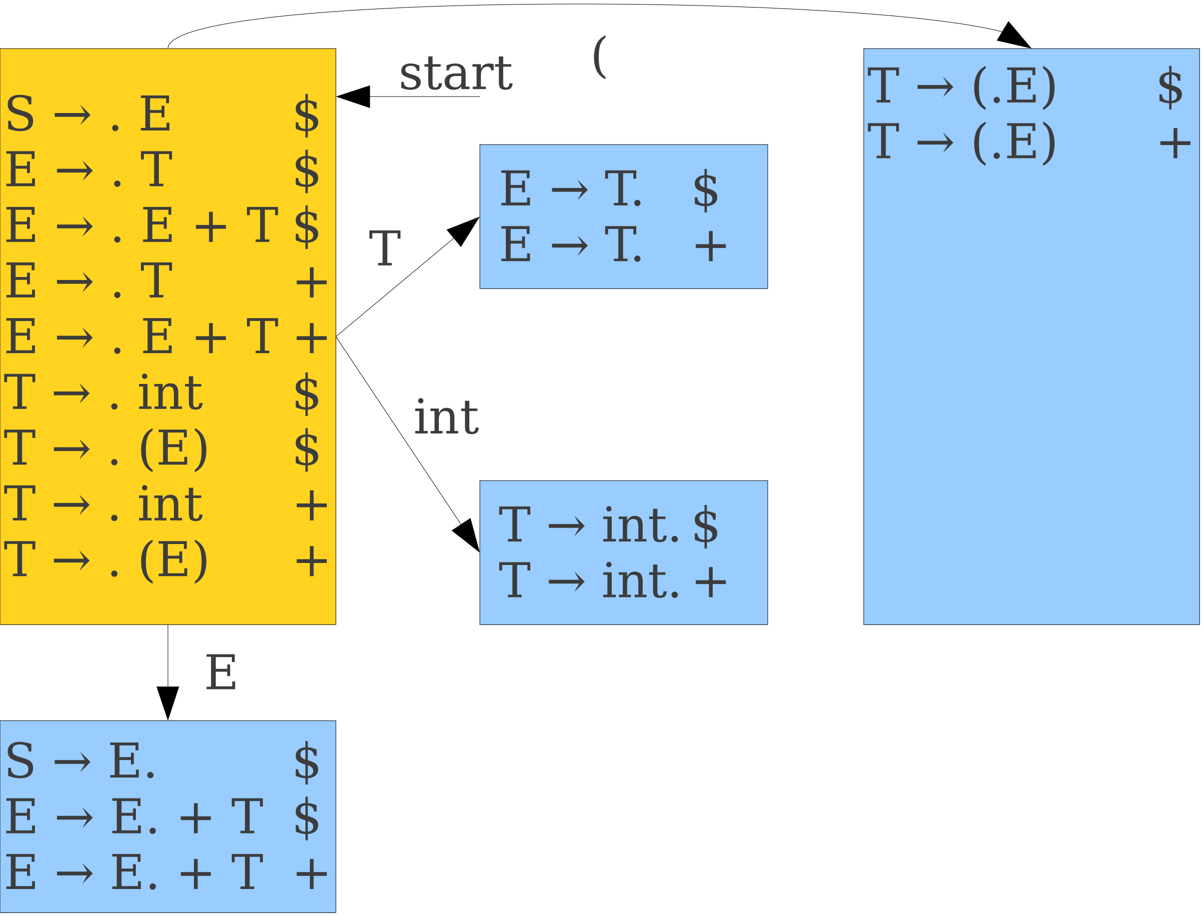
S → . E $
E → . T $
E → . E + T $
E → . T +
E → . E + T +
T → . int $
T → . (E) $
T → . int +
T → . (E) +

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

Deterministic LR(1) Automata



start

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

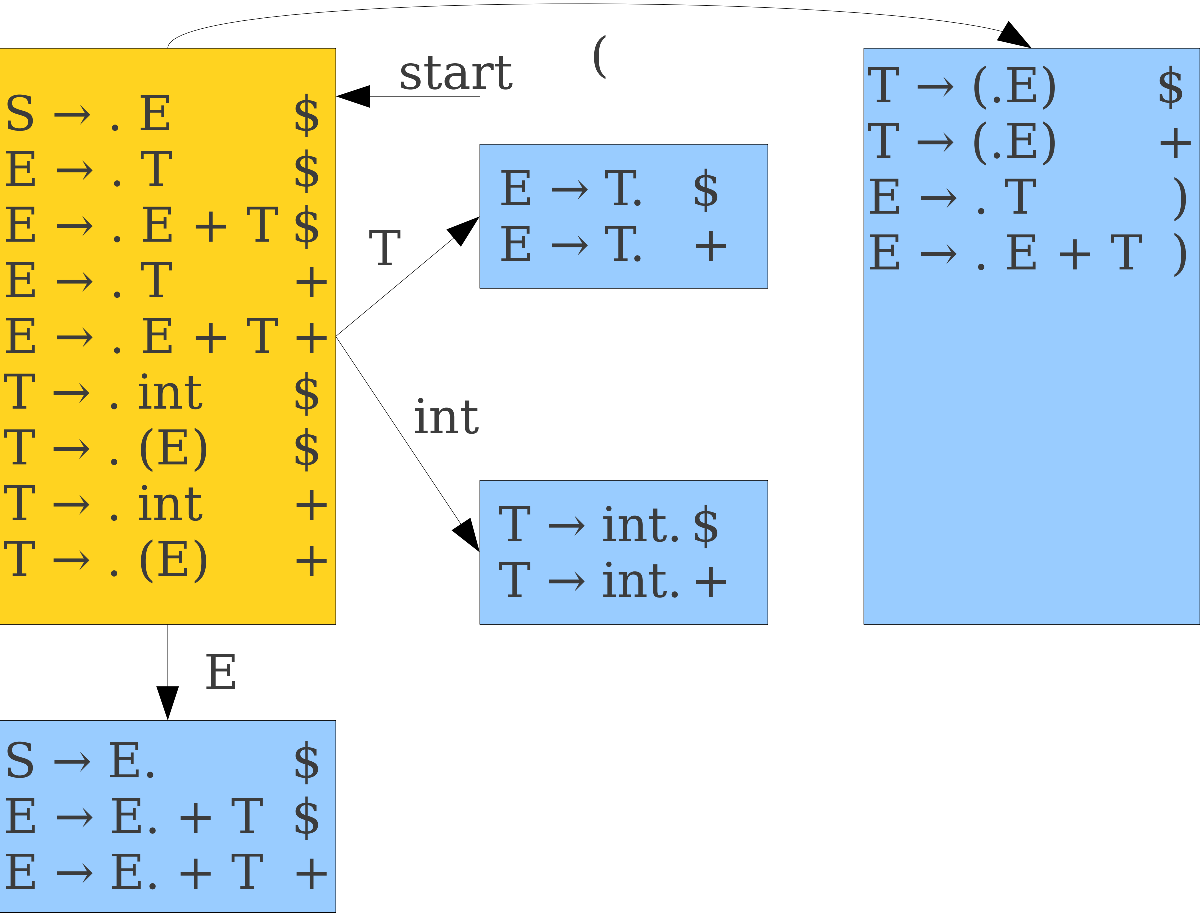
(

T → (.E) $
T → (.E) +

E

S → E. $
E → E. + T $
E → E. + T +

Deterministic LR(1) Automata



start

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

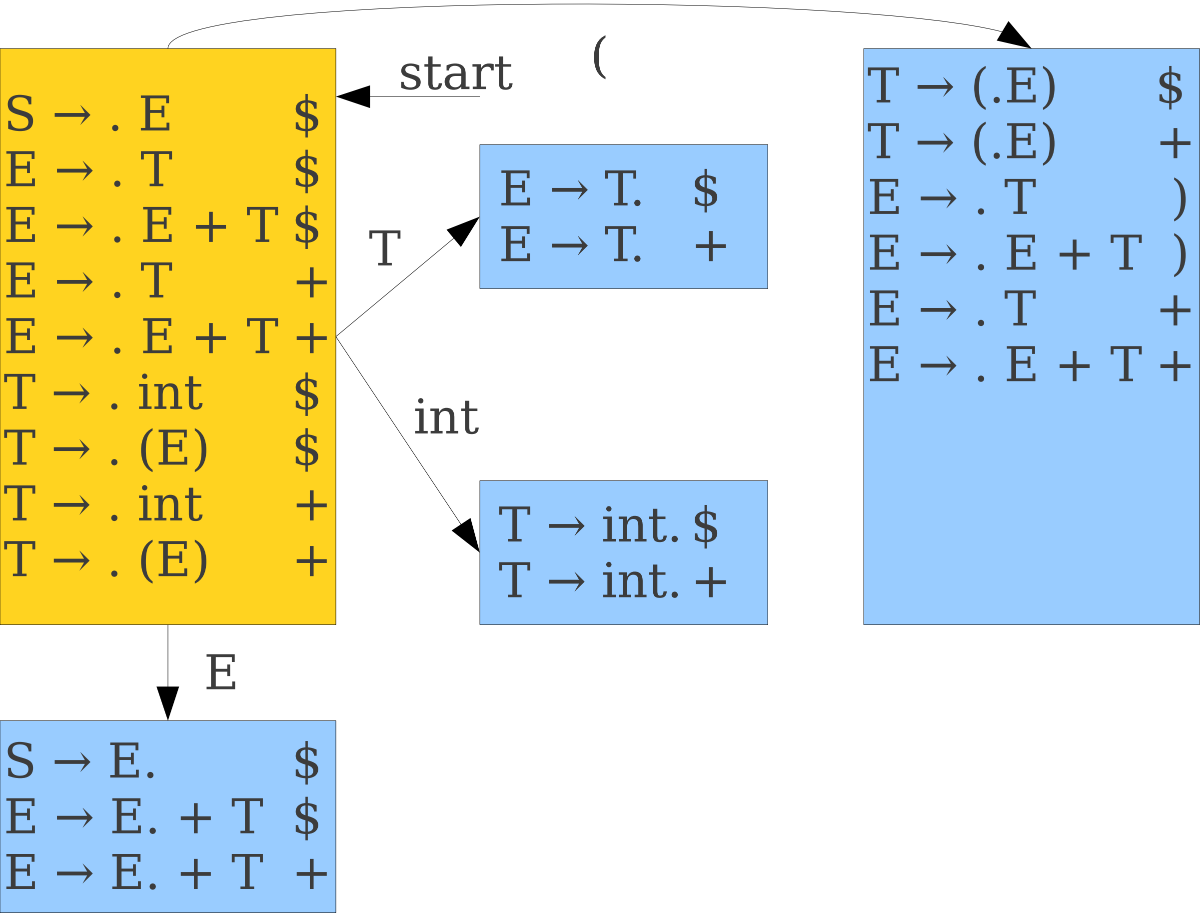
(

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

E

S → E. $
E → E. + T $
E → E. + T +

Deterministic LR(1) Automata



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

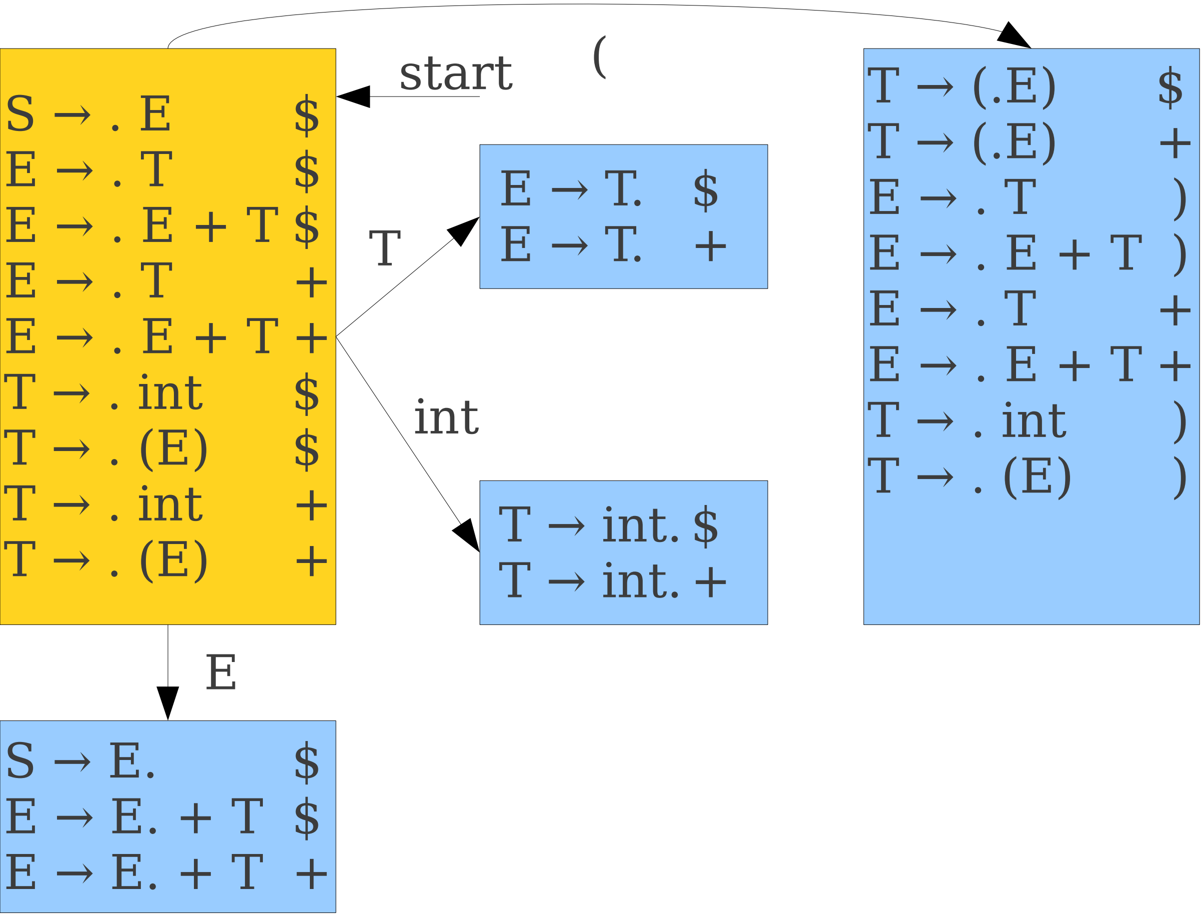
E → T. $
E → T. +

T → int. $
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

Deterministic LR(1) Automata



start

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

(

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

T

E → T. $
E → T. +

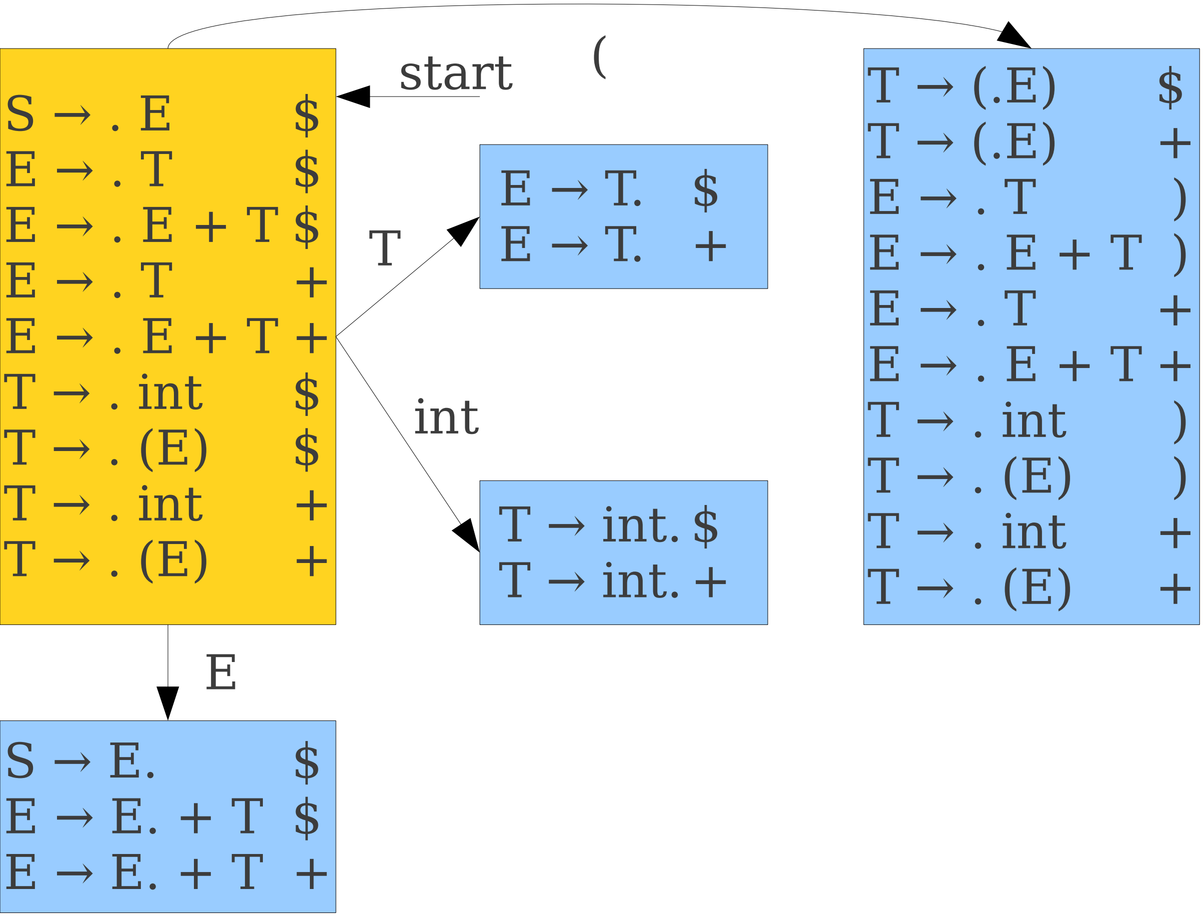
int

T → int. $
T → int. +

E

S → E. $
E → E. + T $
E → E. + T +

Deterministic LR(1) Automata



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

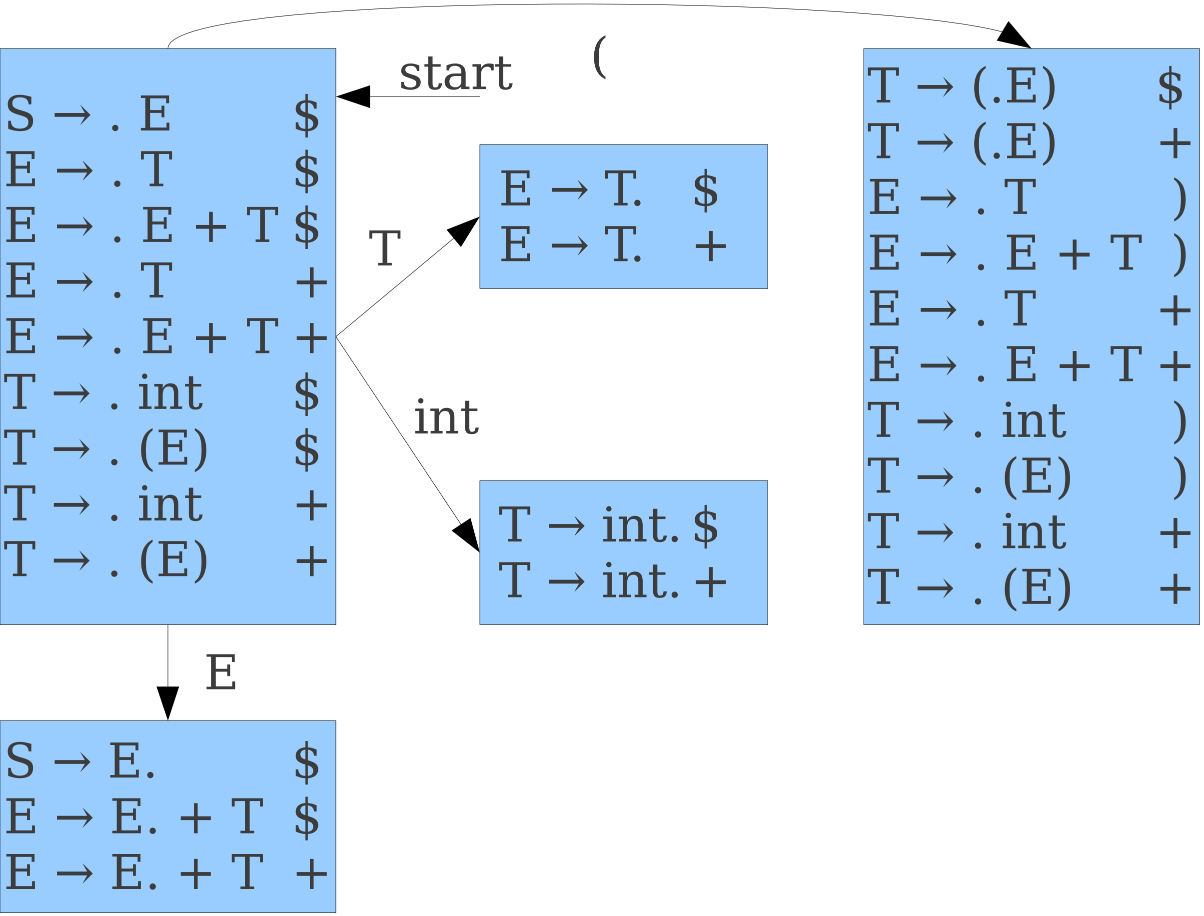
E → T. $
E → T. +

T → int. $
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

Deterministic LR(1) Automata



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

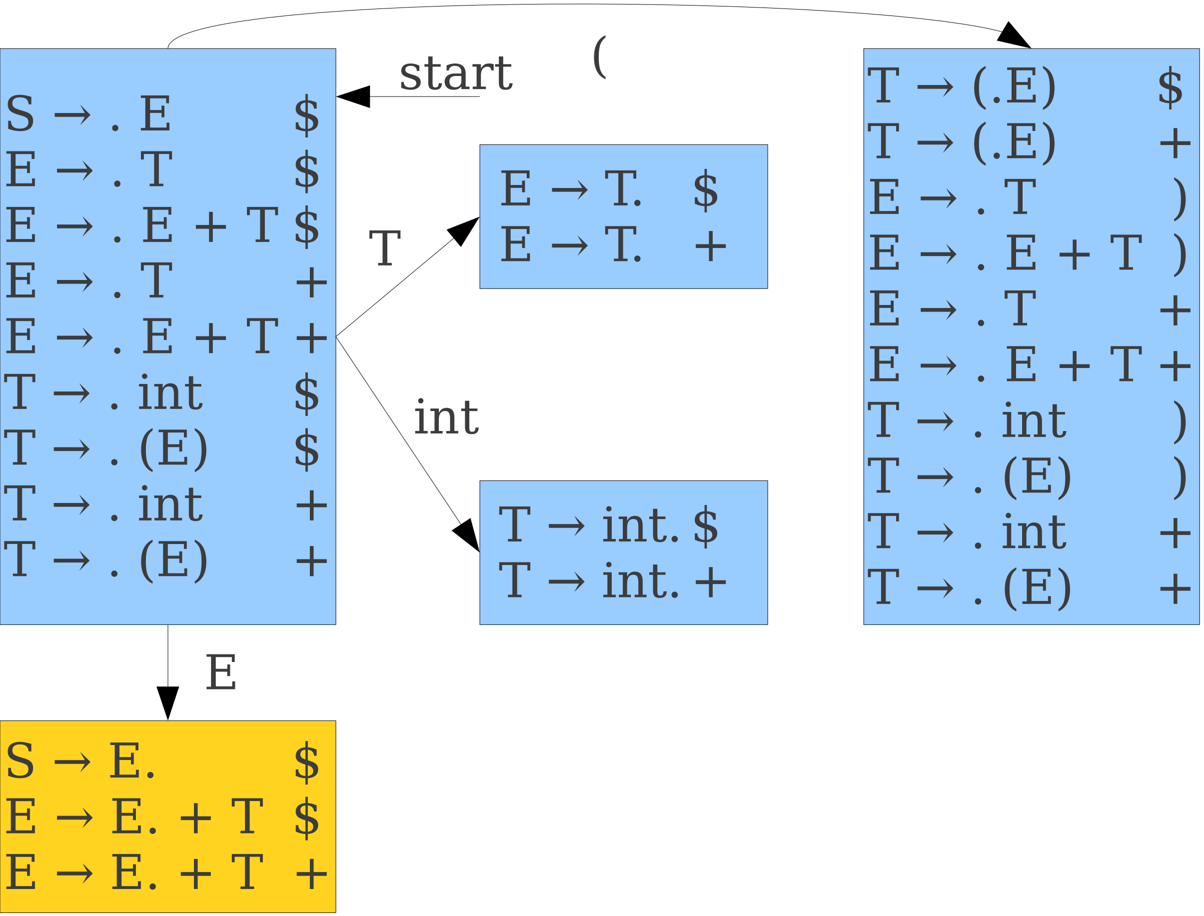
E → T. $
E → T. +

T → int. $
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

Deterministic LR(1) Automata



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

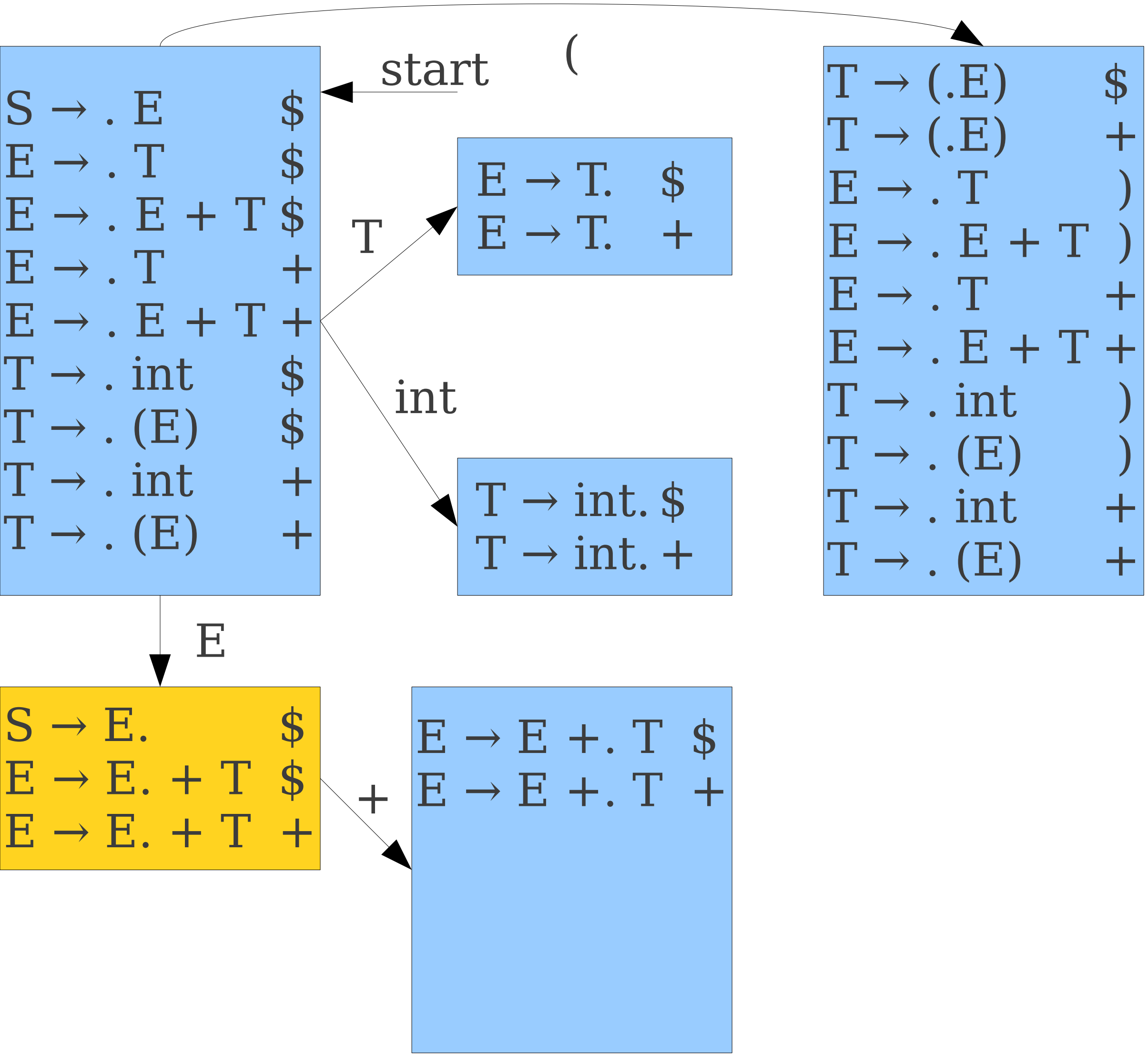
E → T. $
E → T. +

T → int. $
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

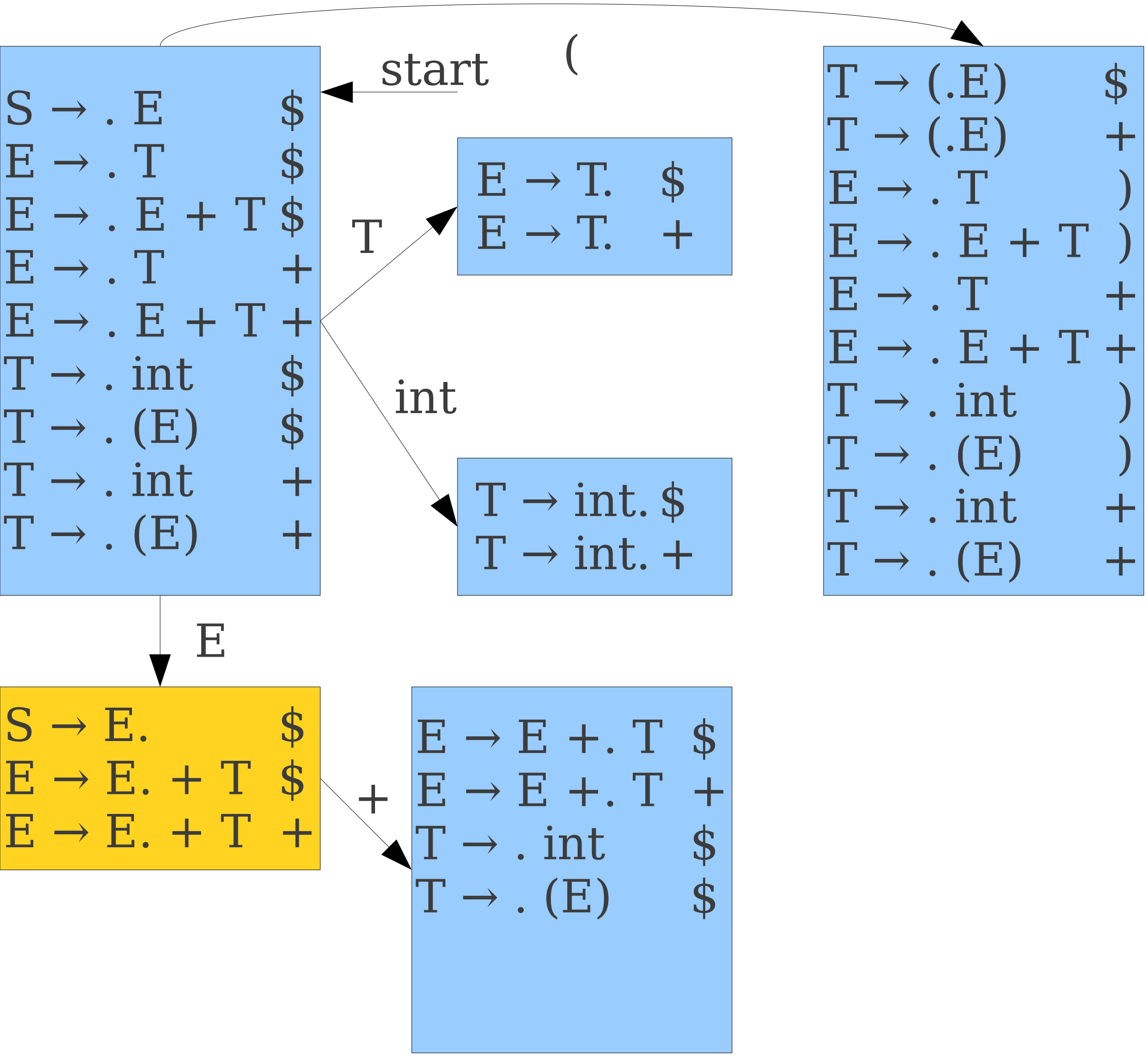
T → int. $
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

E → E +. T $
E → E +. T +

Deterministic LR(1) Automata



start

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

(

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

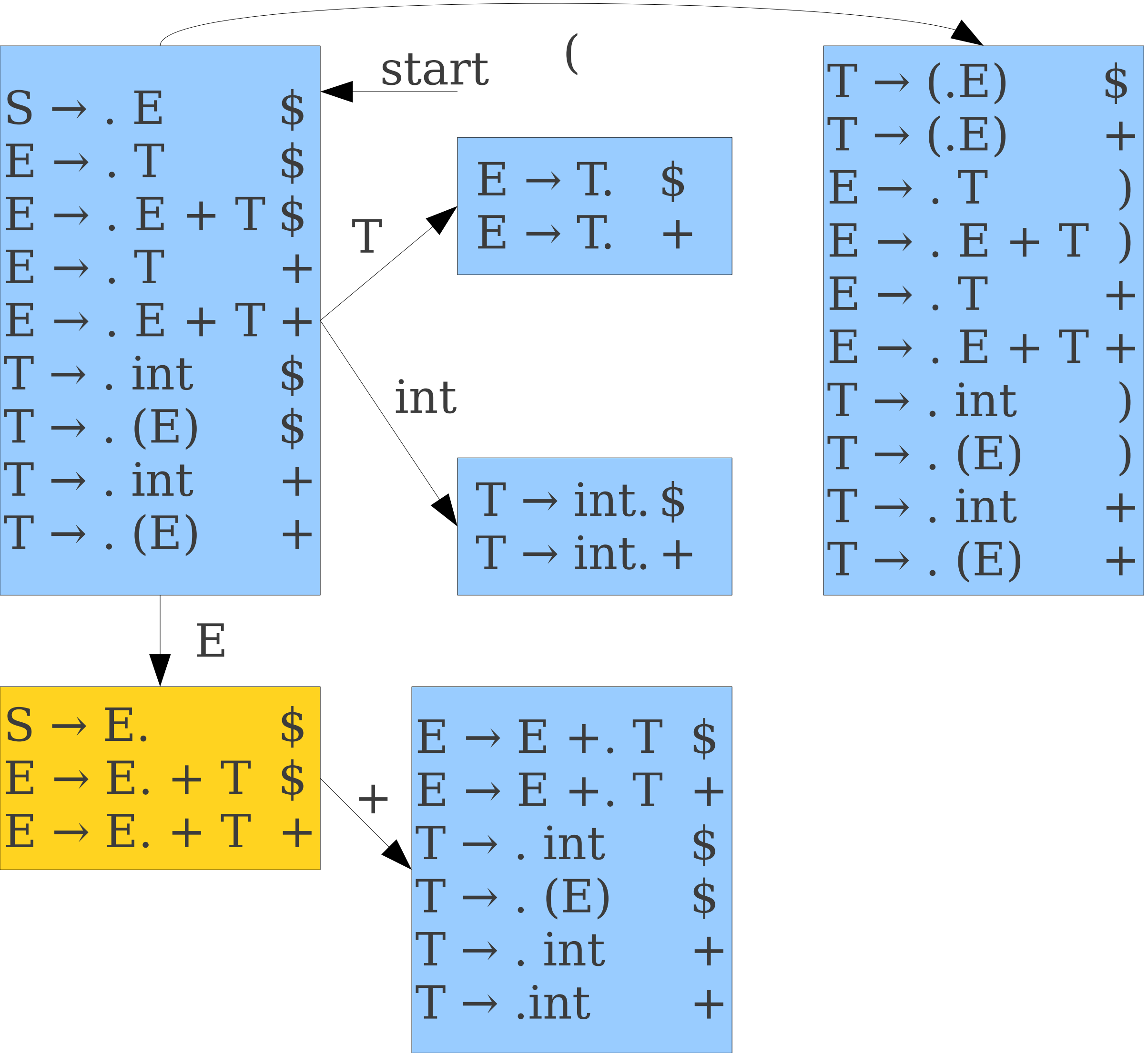
E

S → E. $
E → E. + T $
E → E. + T +

\+

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

Deterministic LR(1) Automata



start

(

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

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

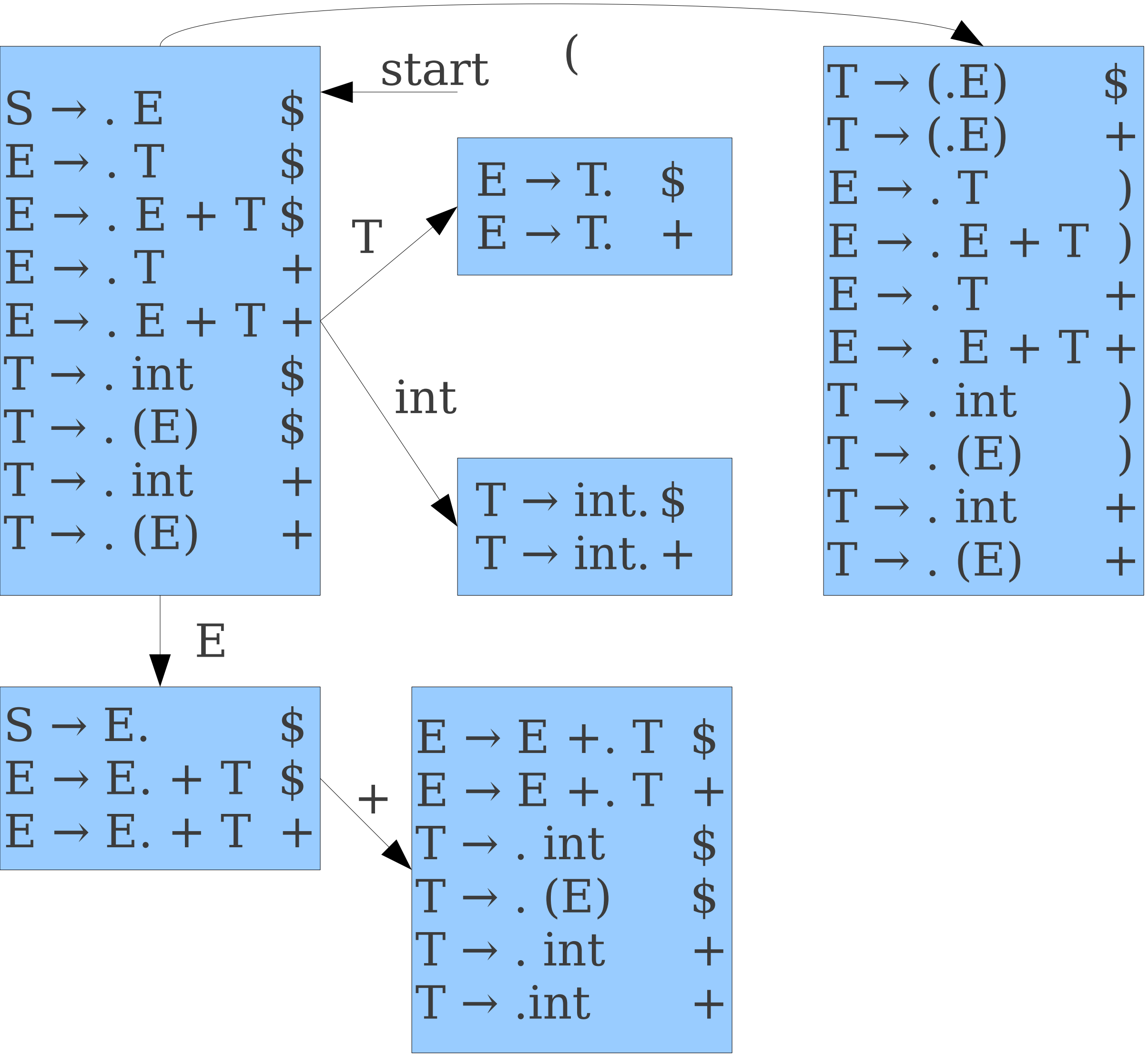
E

S → E. $
E → E. + T $
E → E. + T +

\+

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

Deterministic LR(1) Automata



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

E → T. $
E → T. +

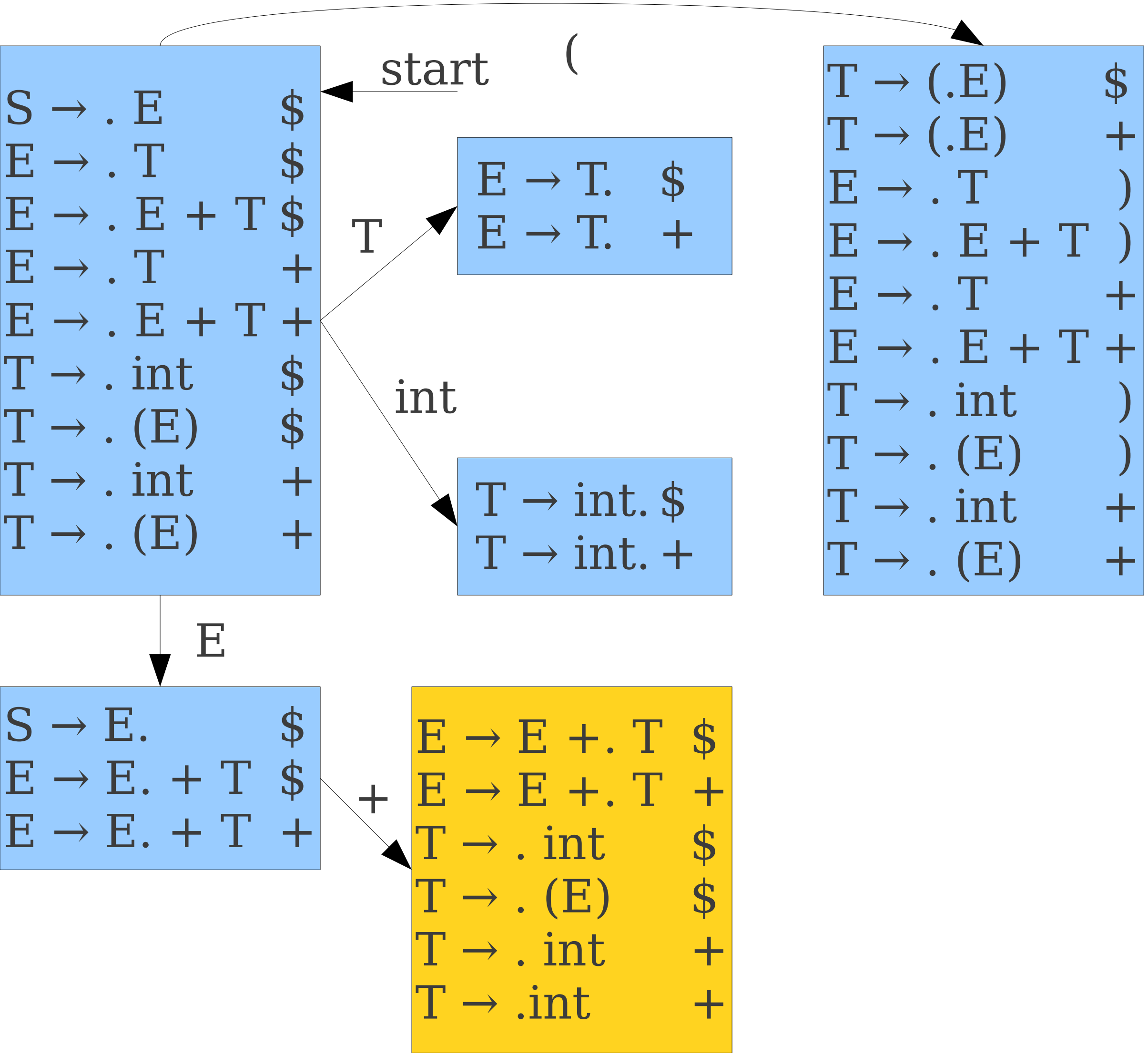
T → int. $
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

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

Deterministic LR(1) Automata



start

(

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

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

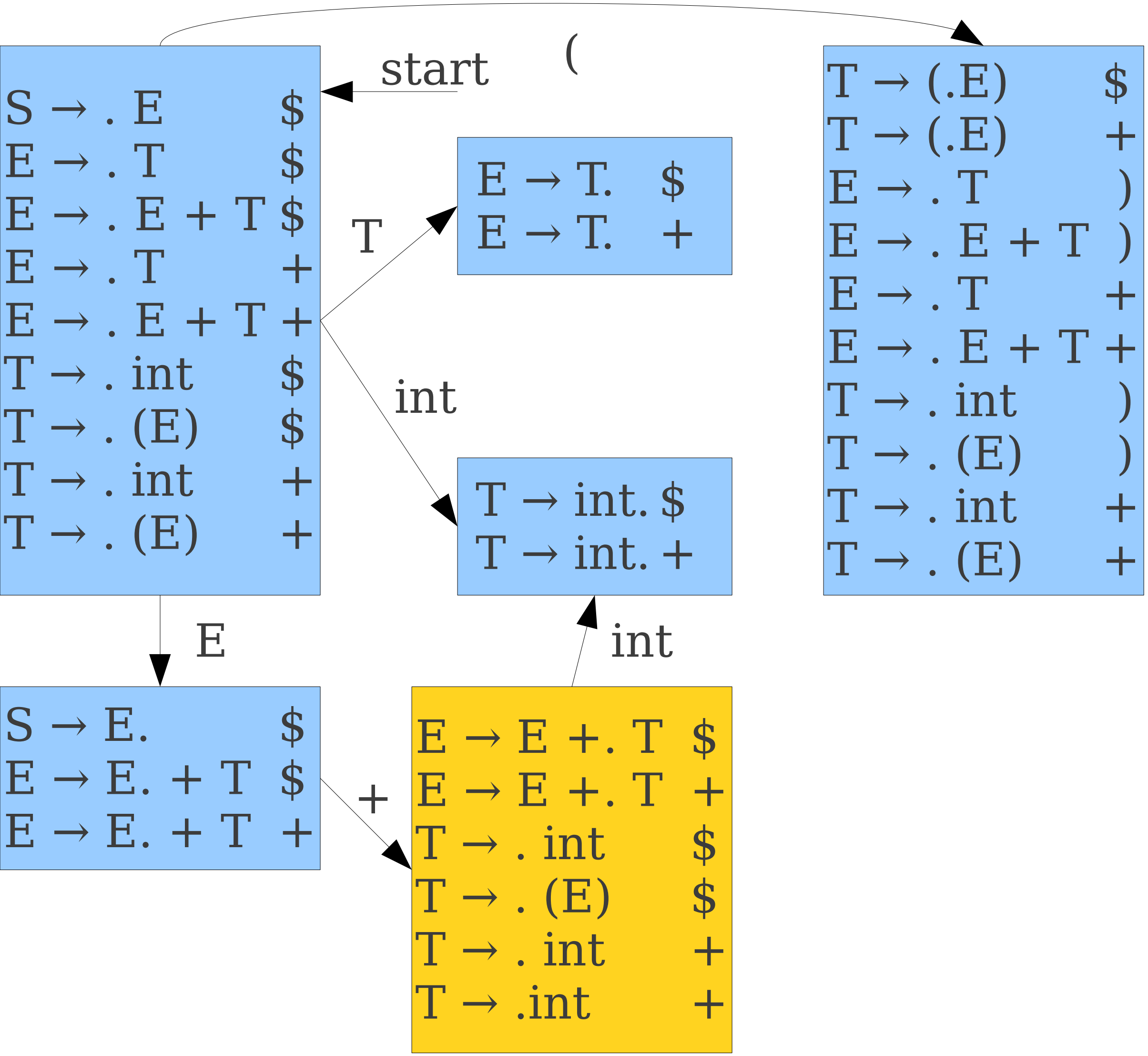
E

S → E. $
E → E. + T $
E → E. + T +

\+

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

Deterministic LR(1) Automata



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

E → T. $
E → T. +

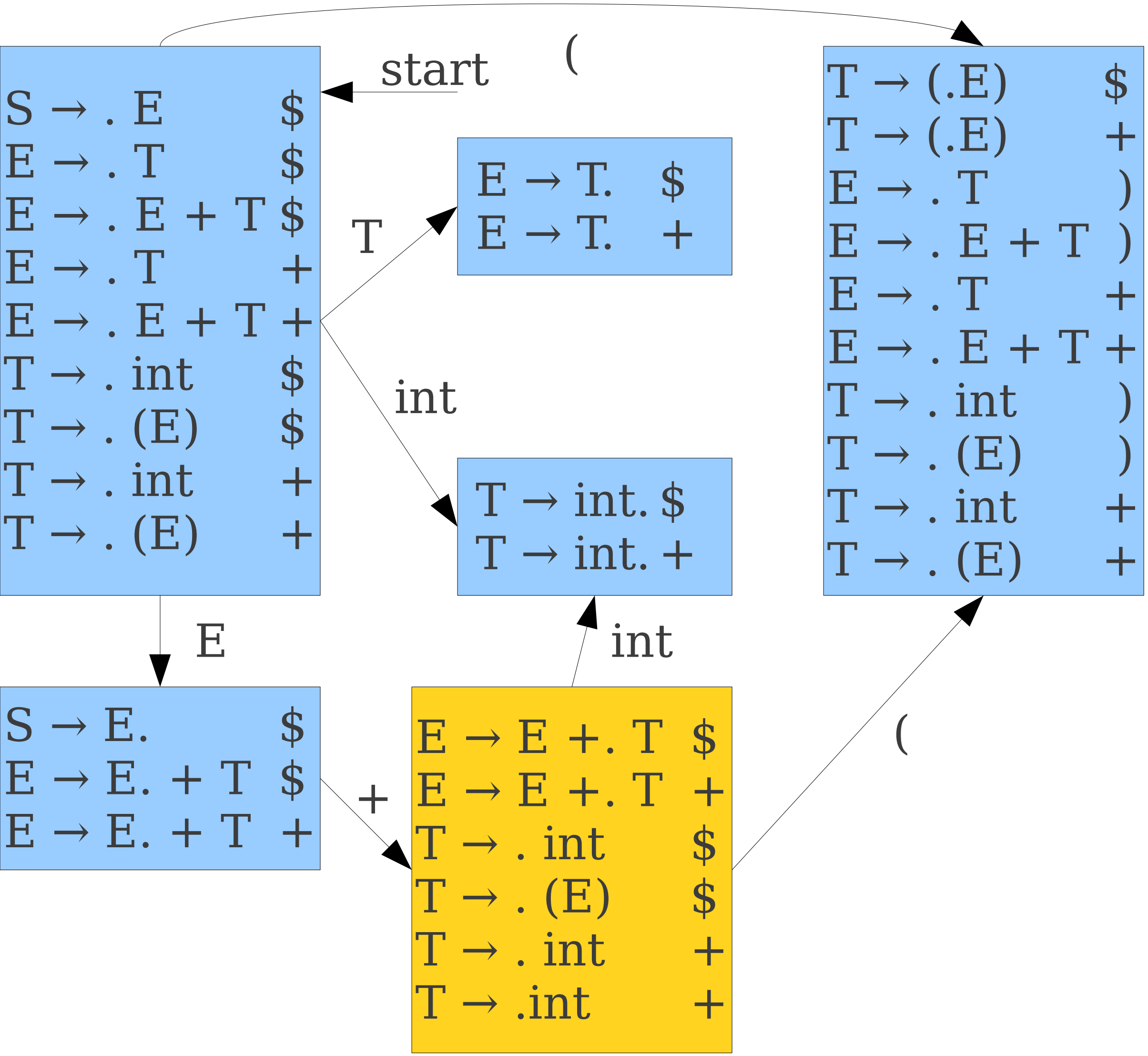
T → int. $
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

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

Deterministic LR(1) Automata



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

E → T. $
E → T. +

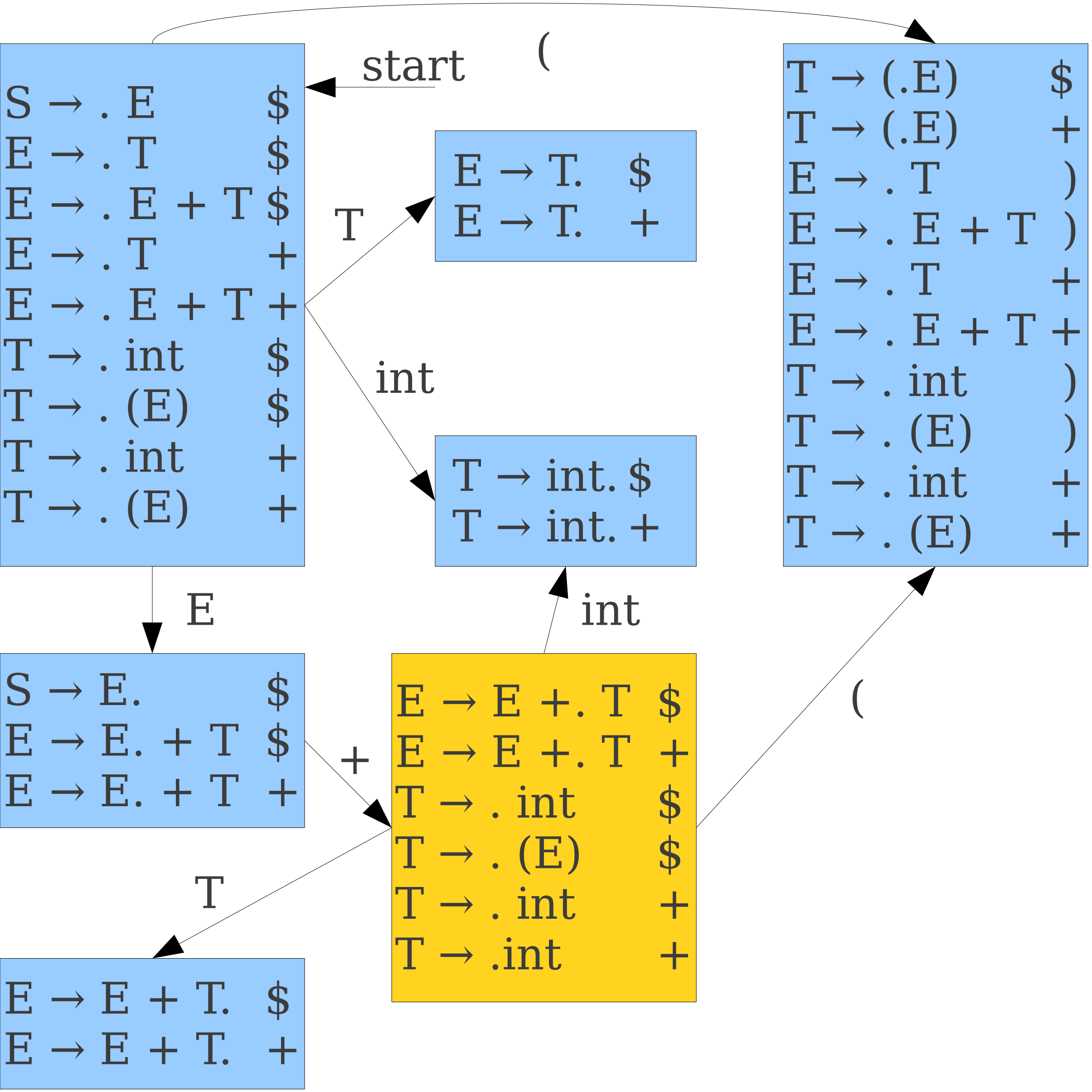
T → int. $
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

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

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

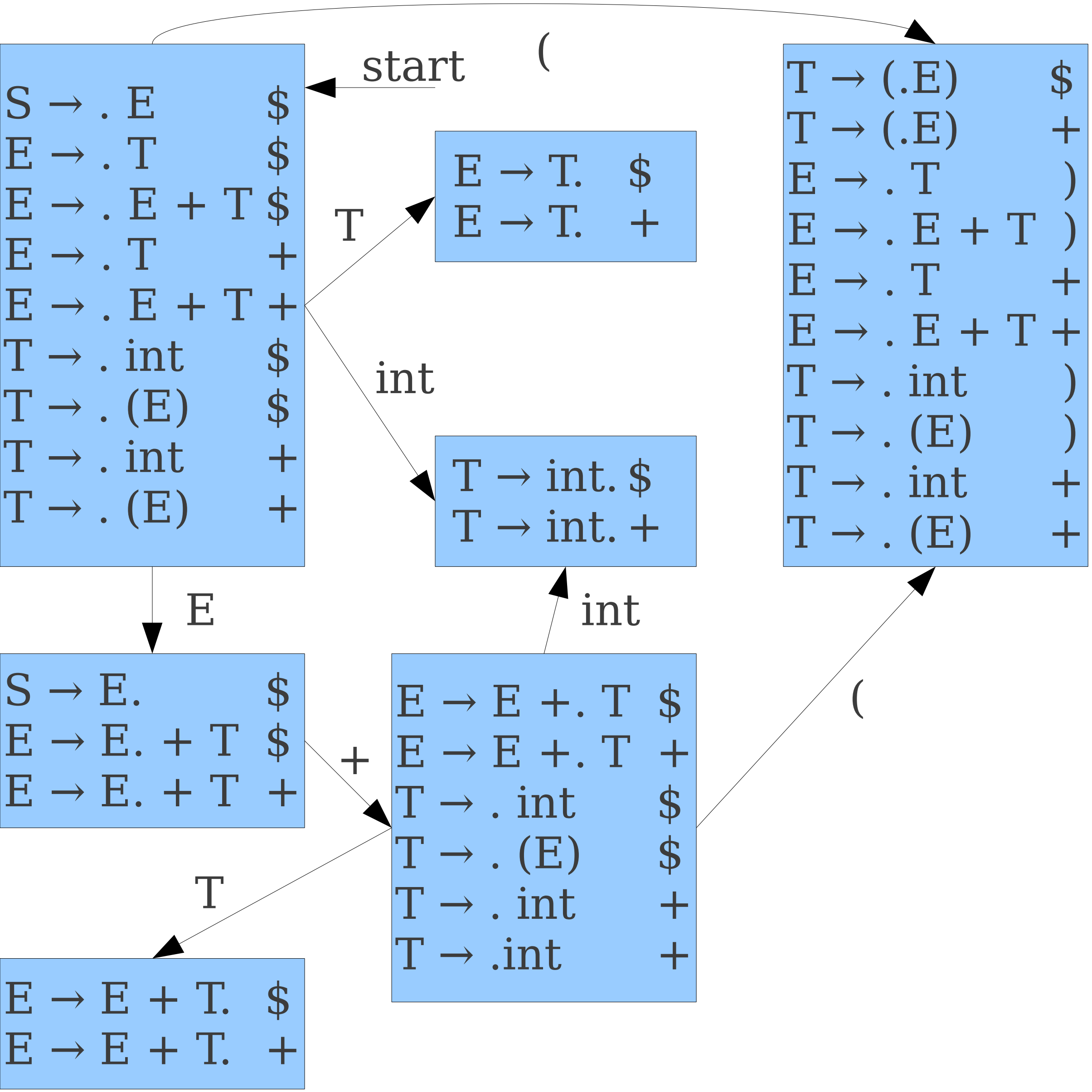
T → (.E) $
T → (.E) +
E → . T)
E → . E + T)
E → . T +
E → . E + T +
T → . int)
T → . (E))
T → . int +
T → . (E) +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

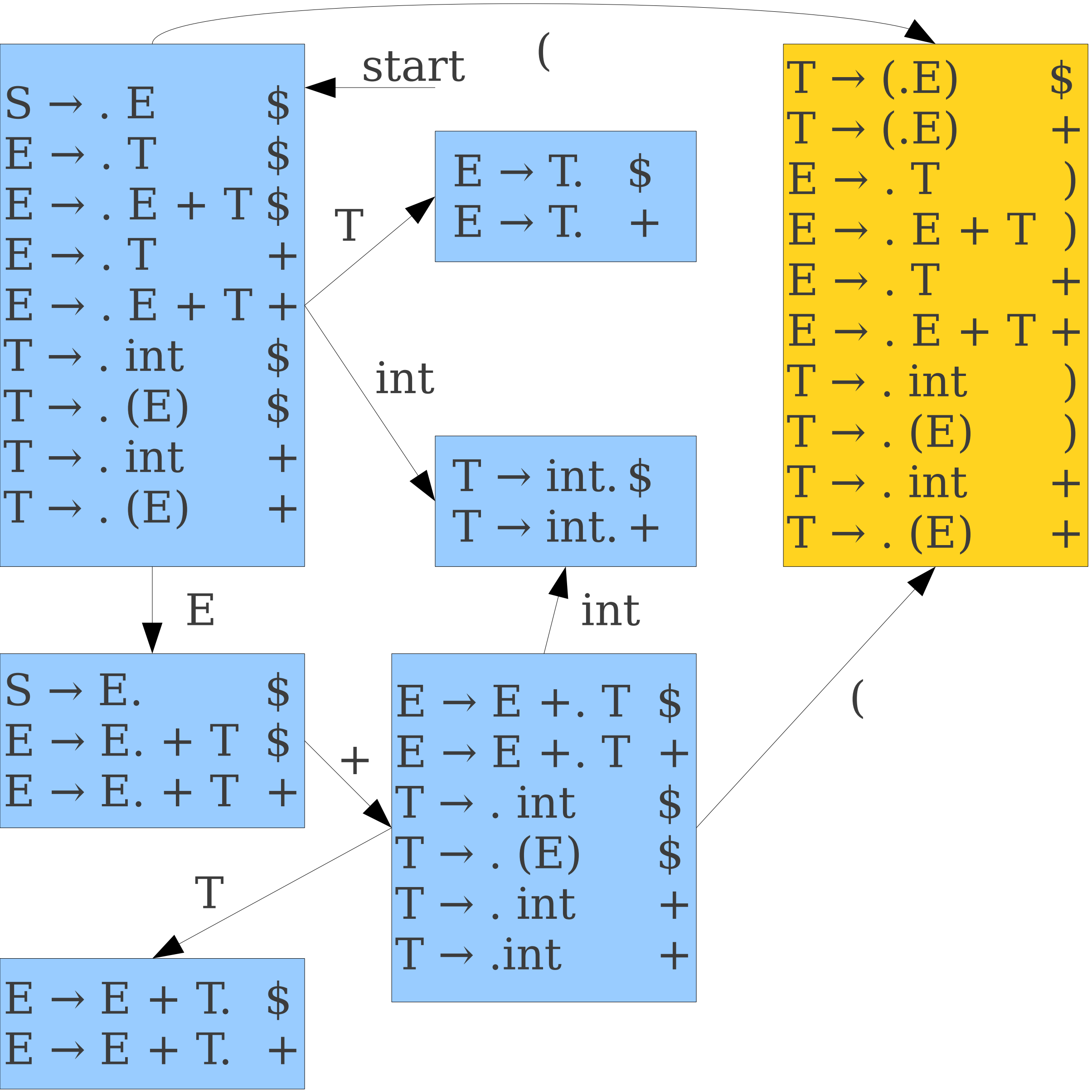
T → (.E) $
T → (.E) +
E → . T)
E → . E + T)
E → . T +
E → . E + T +
T → . int)
T → . (E))
T → . int +
T → . (E) +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

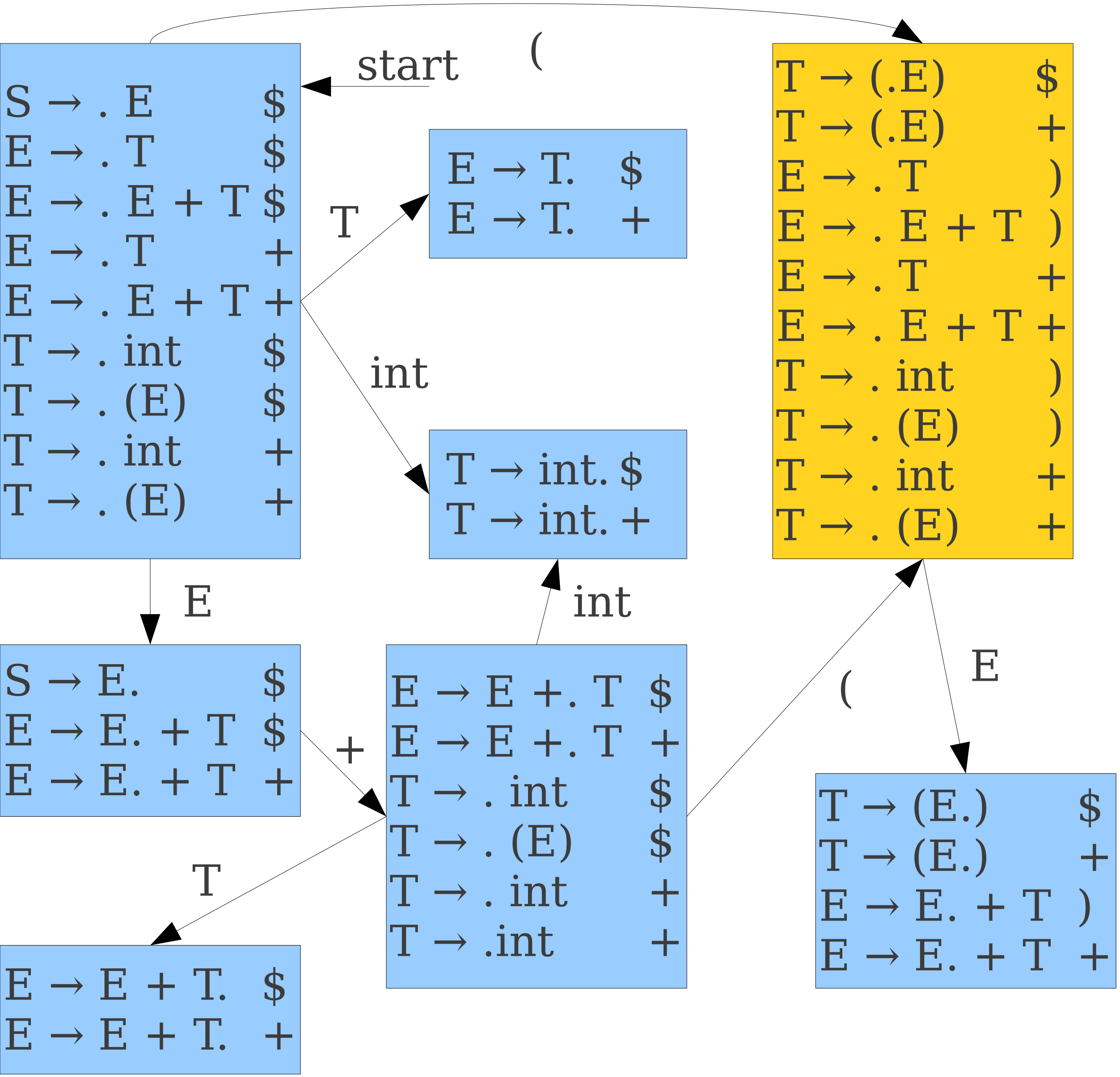
T → (.E) $
T → (.E) +
E → . T)
E → . E + T)
E → . T +
E → . E + T +
T → . int)
T → . (E))
T → . int +
T → . (E) +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

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

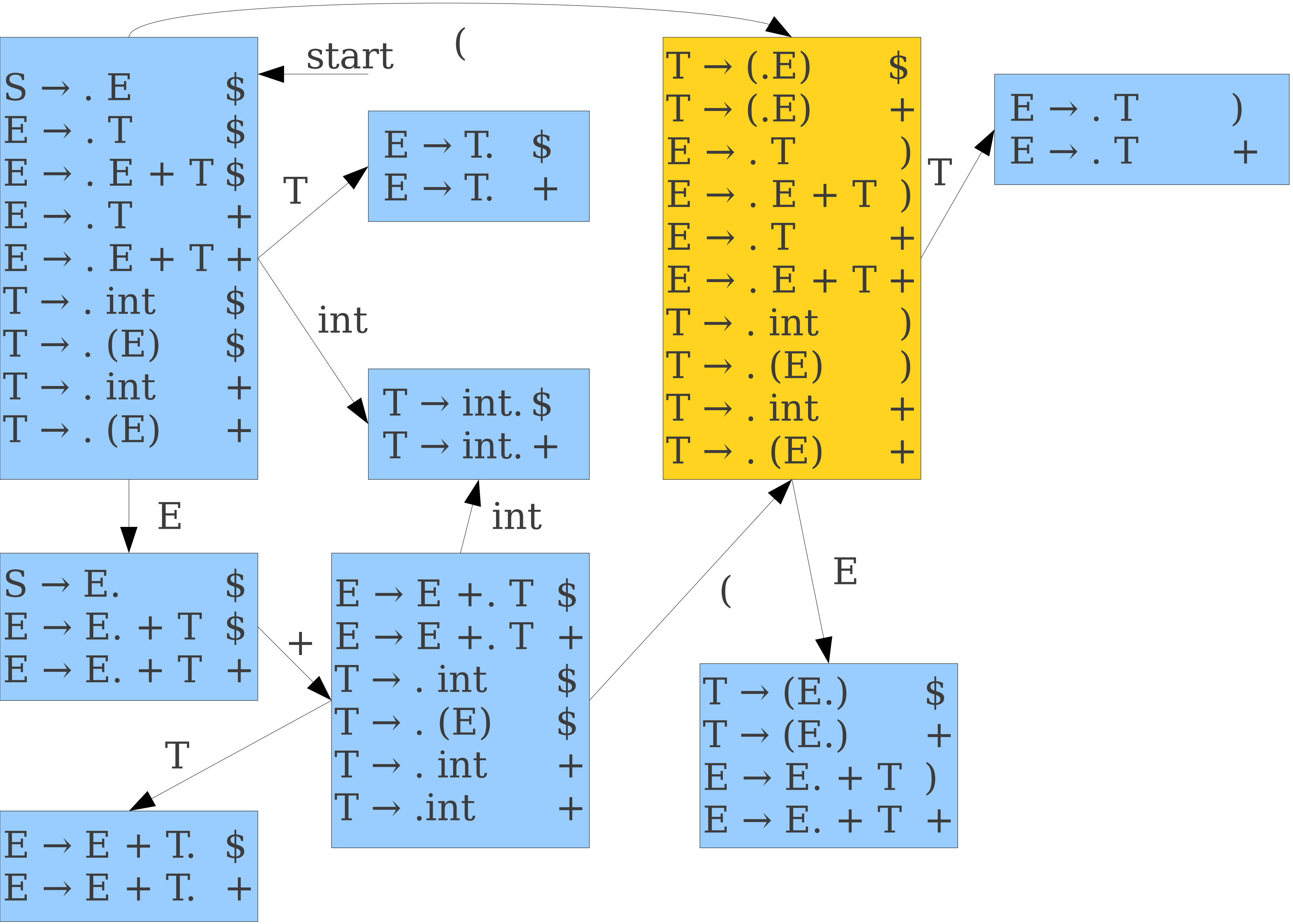
S → E. $
E → E. + T $
E → E. + T +

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

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

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

E → . T)
E → . T +

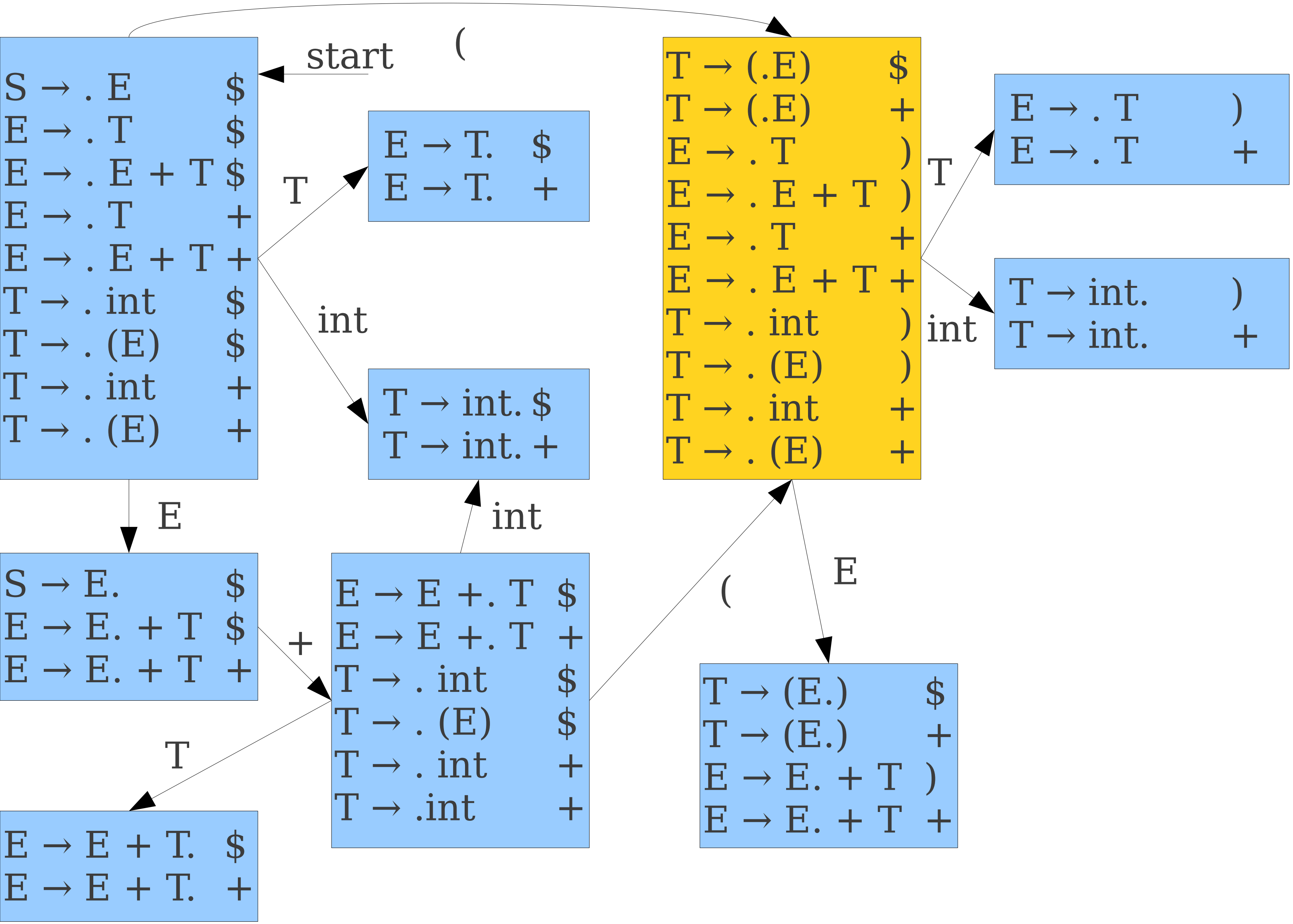
S → E. $
E → E. + T $
E → E. + T +

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

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

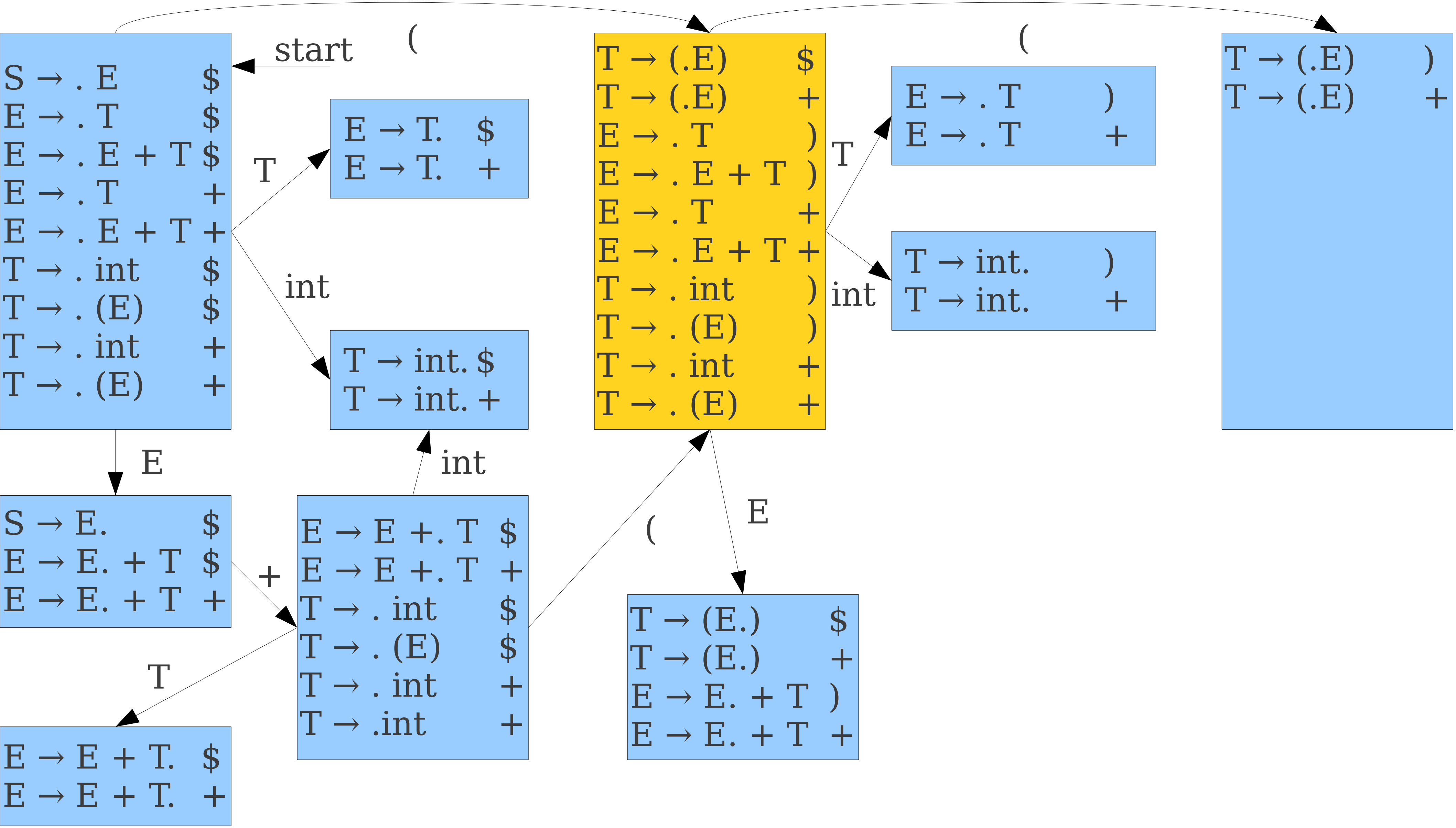
T → (.E) $
T → (.E) +
E → . T)
E → . E + T)
E → . T +
E → . E + T +
T → . int)
T → . (E))
T → . int +
T → . (E) +

E → . T)
E → . T +

T → int.)
T → int. +

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

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

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

E → . T)
E → . T +

T → int.)
T → int. +

T → (.E))
T → (.E) +

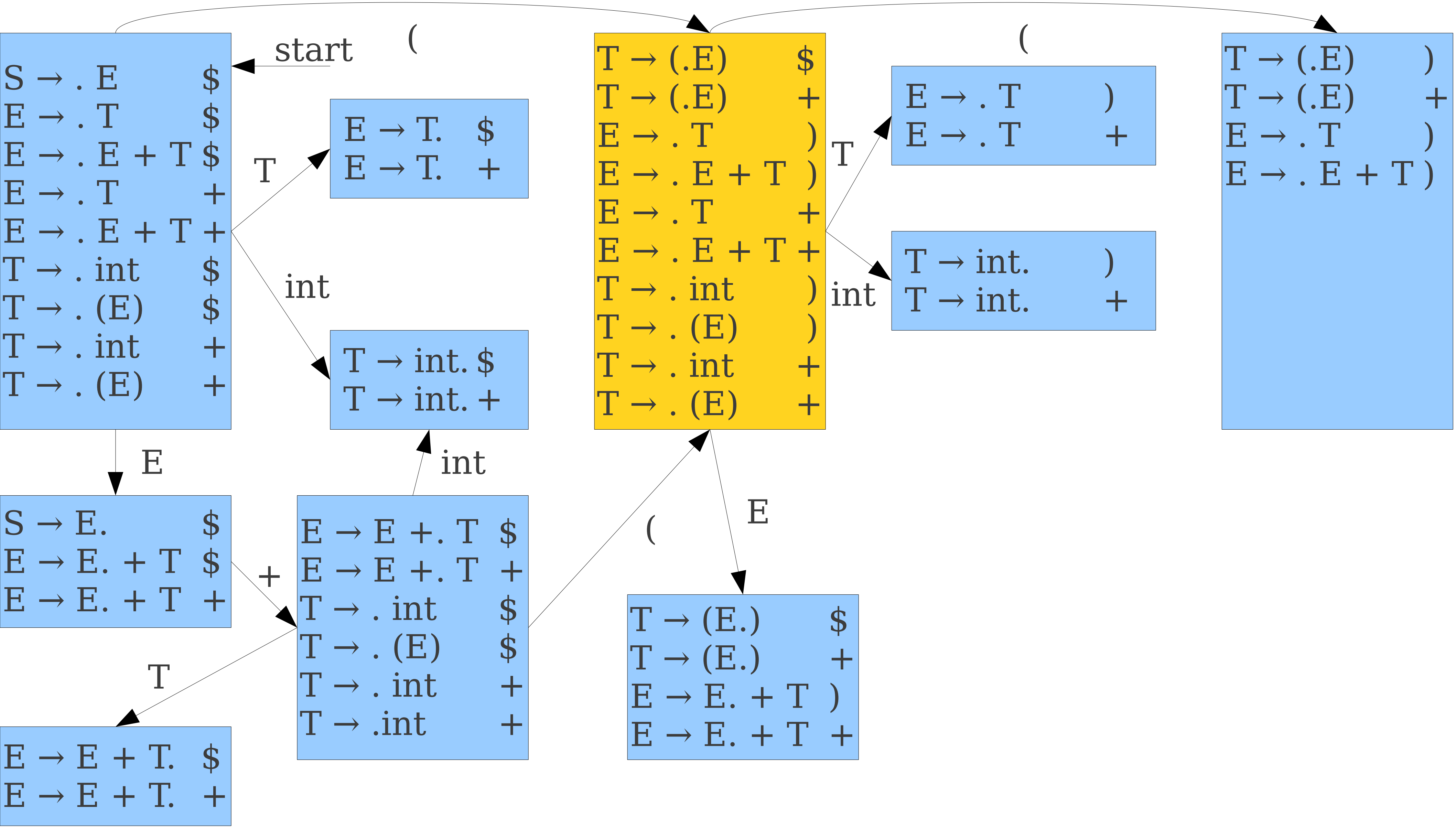
S → E. $
E → E. + T $
E → E. + T +

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

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

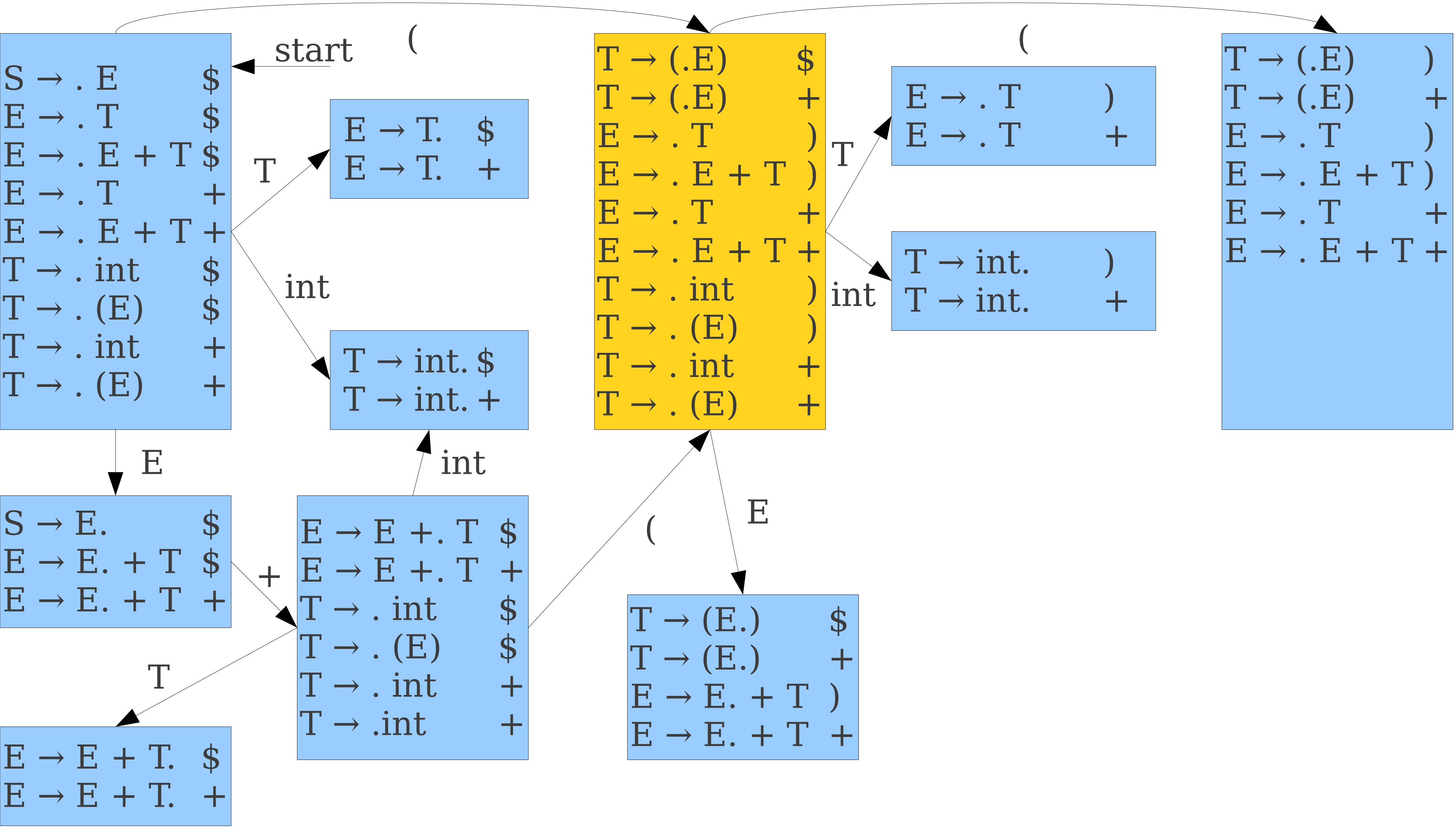
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

E → . T)
E → . T +

T → int.)
T → int. +

T → (.E))
T → (.E) +
E → . T)
E → . E + T)

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

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

E → . T)
E → . T +

T → int.)
T → int. +

T → (.E))
T → (.E) +
E → . T)
E → . E + T)
E → . T +
E → . E + T +

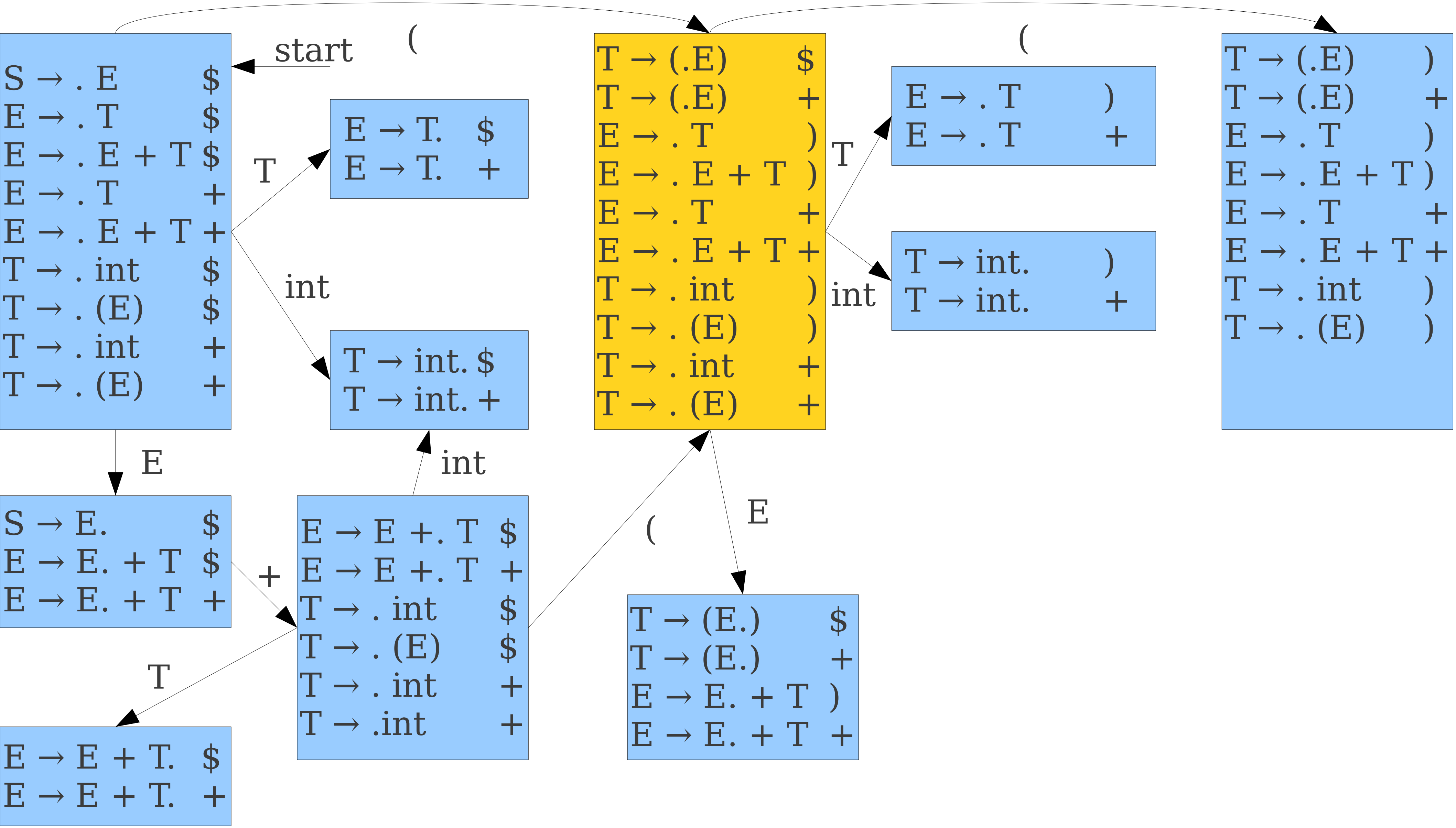
S → E. $
E → E. + T $
E → E. + T +

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

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

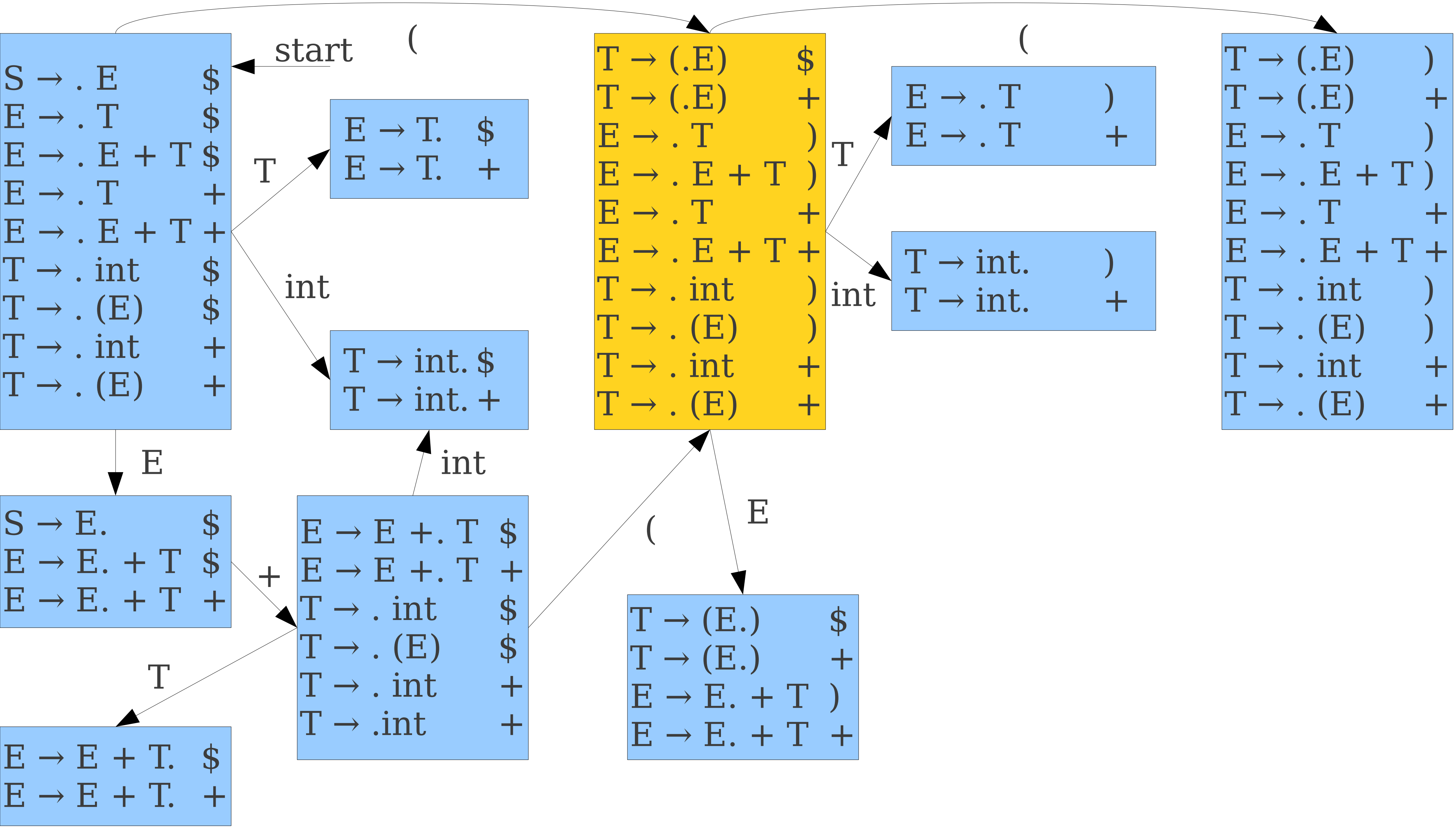
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

E → . T)
E → . T +

T → int.)
T → int. +

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

Deterministic LR(1) Automata



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

start

(

T

E → T. $
E → T. +

int

T → int. $
T → int. +

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

T

int

E → . T)
E → . T +

T → int.)
T → int. +

(

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

E

S → E. $
E → E. + T $
E → E. + T +

\+

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

int

(

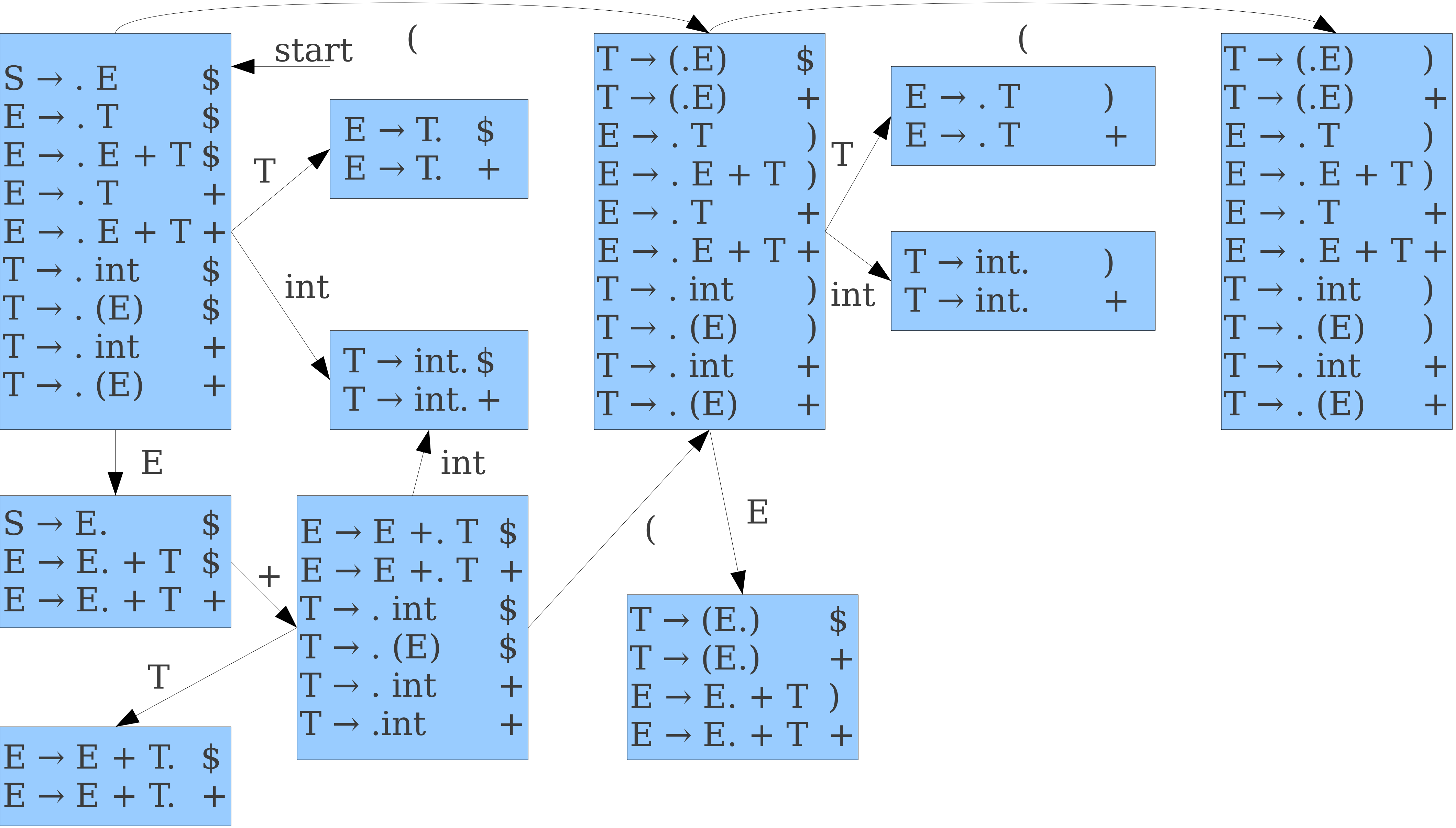
E

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

T

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

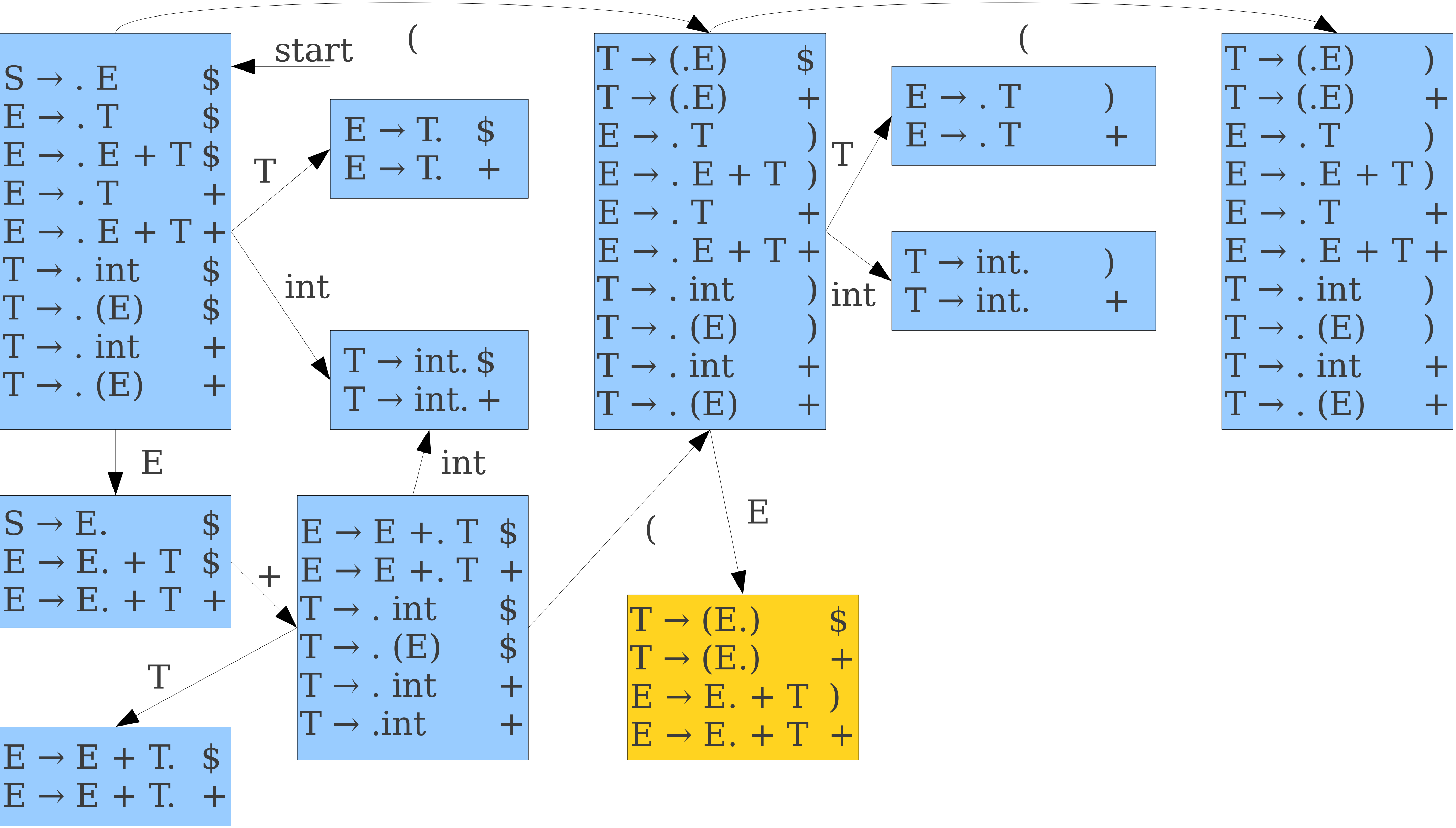
S → E. $
E → E. + T $
E → E. + T +

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

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

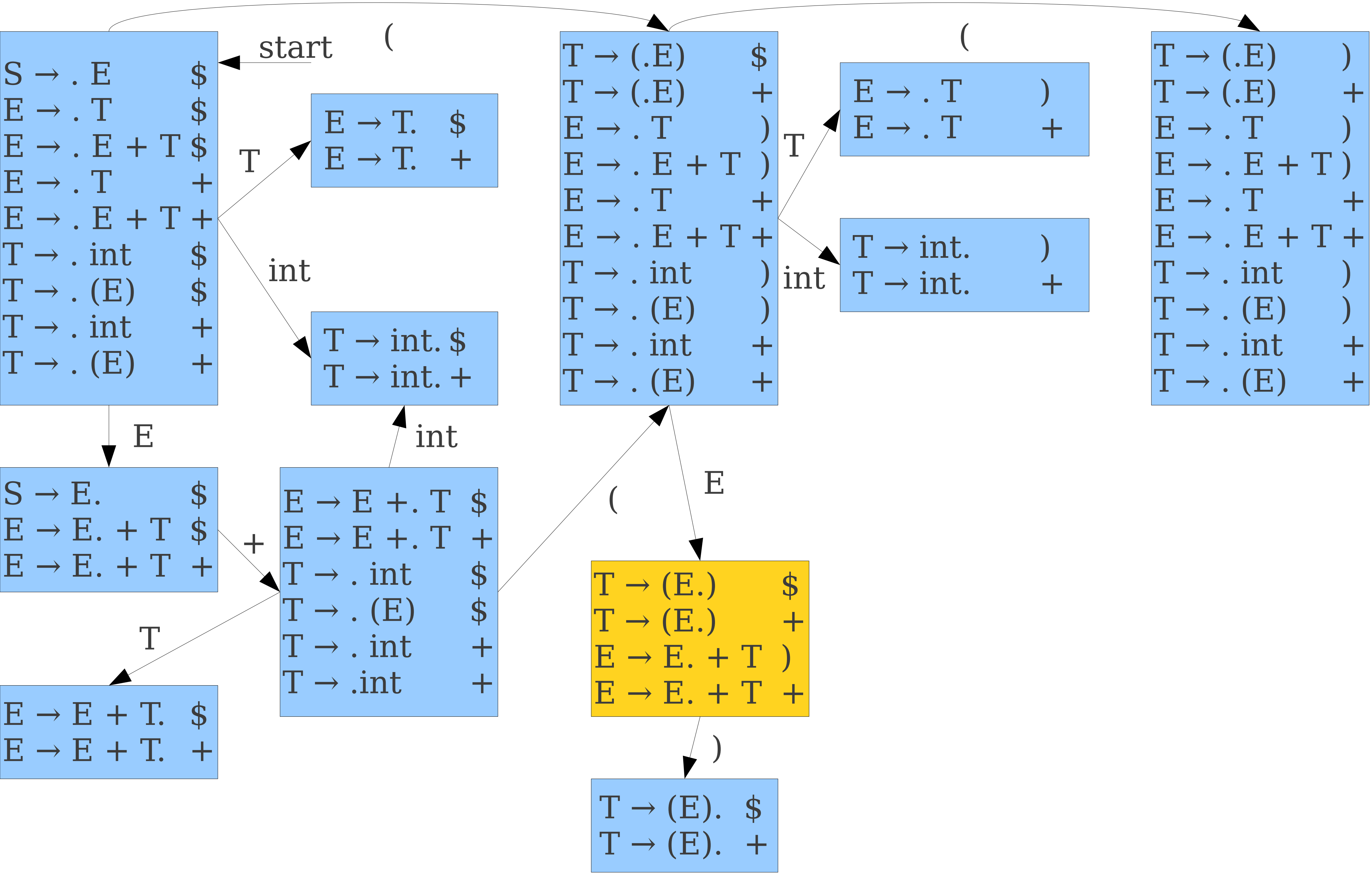
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

E → . T)
E → . T +

T → int.)
T → int. +

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

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

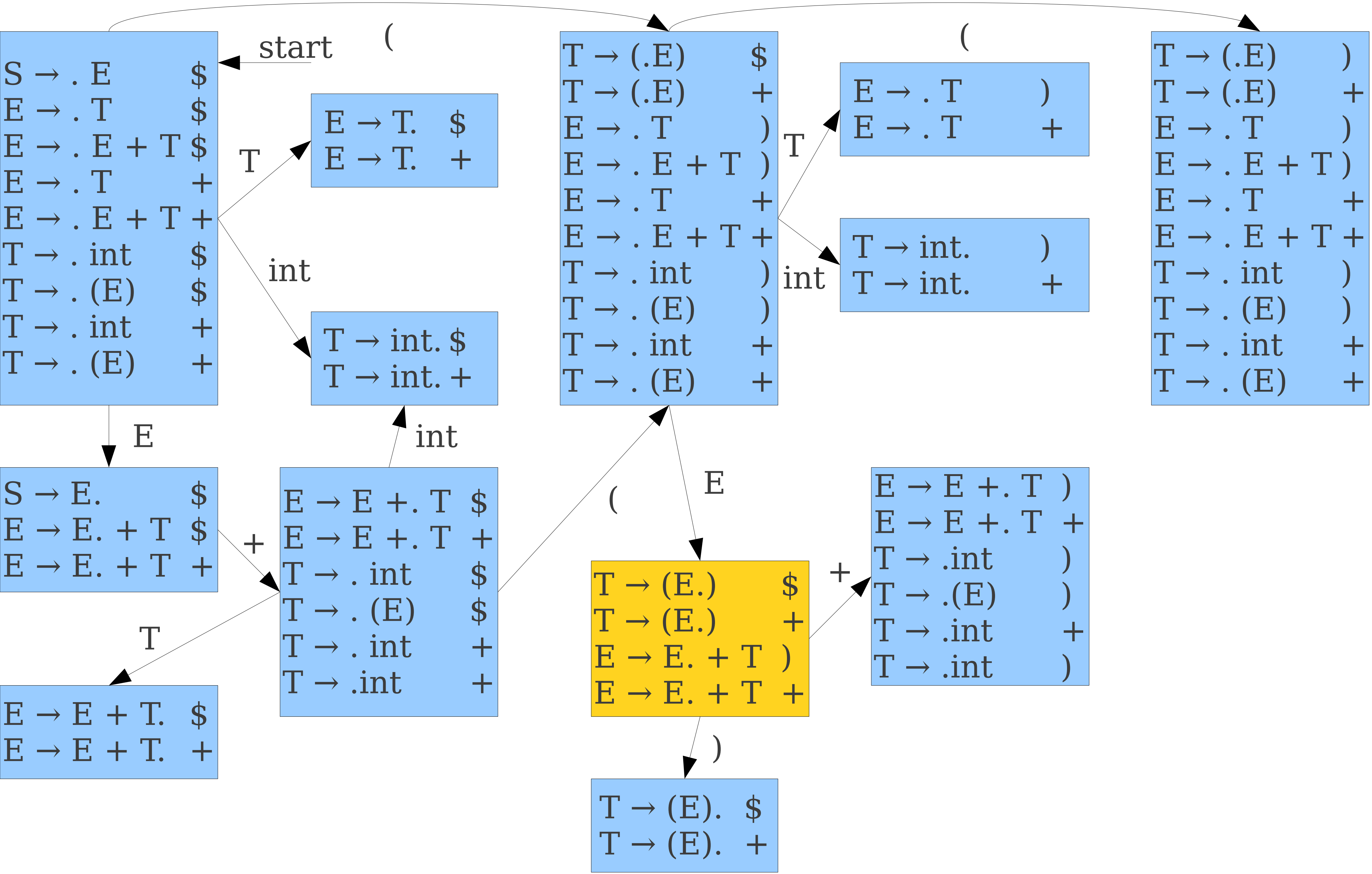
T → int.)
T → int. +

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

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

T → (E). $
T → (E). +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

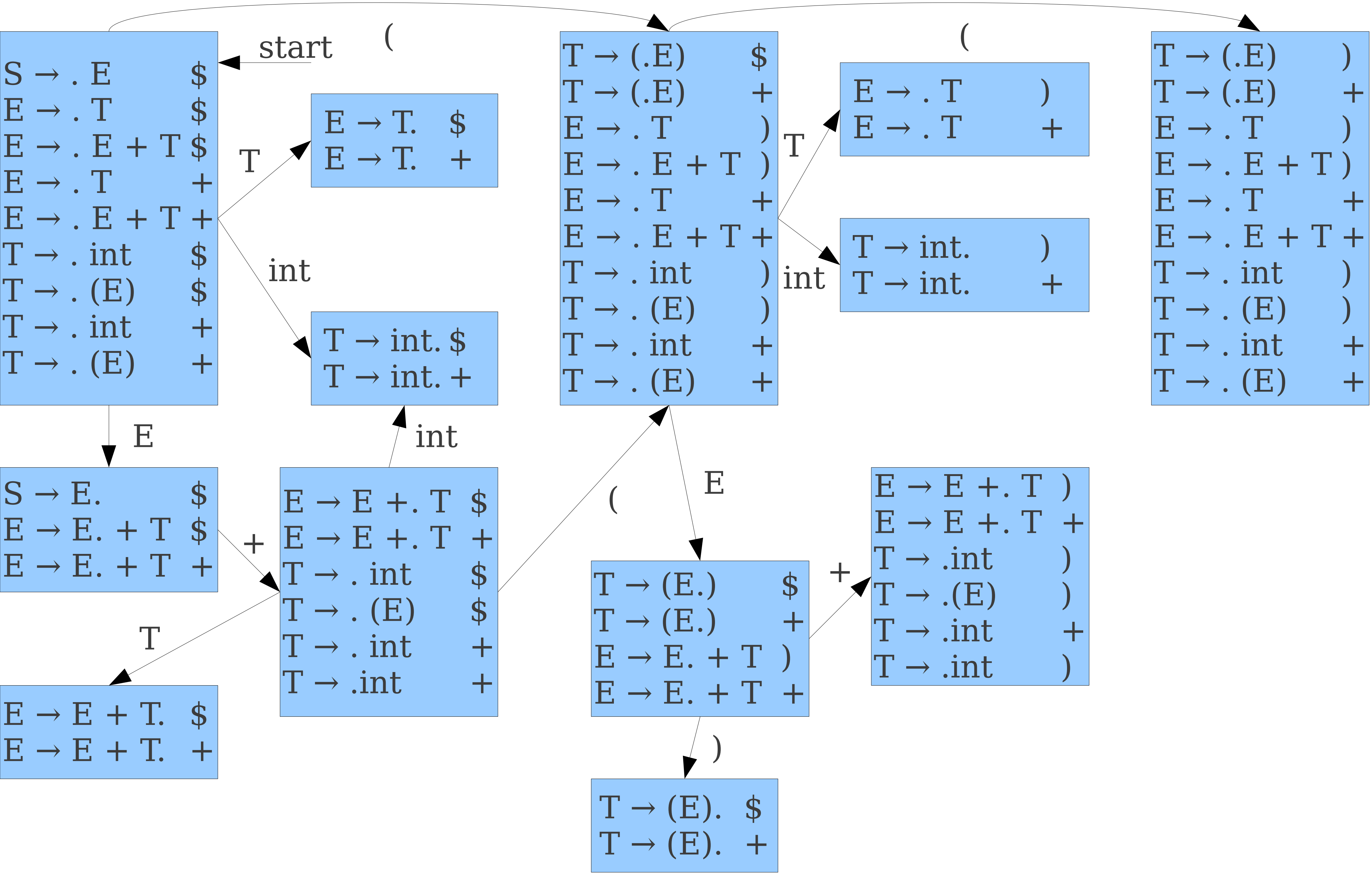
T → (.E))
T → (.E) +
E → . T)
E → . E + T)
E → . T +
E → . E + T +
T → . int)
T → . (E))
T → . int +
T → . (E) +

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

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

T → (E). $
T → (E). +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

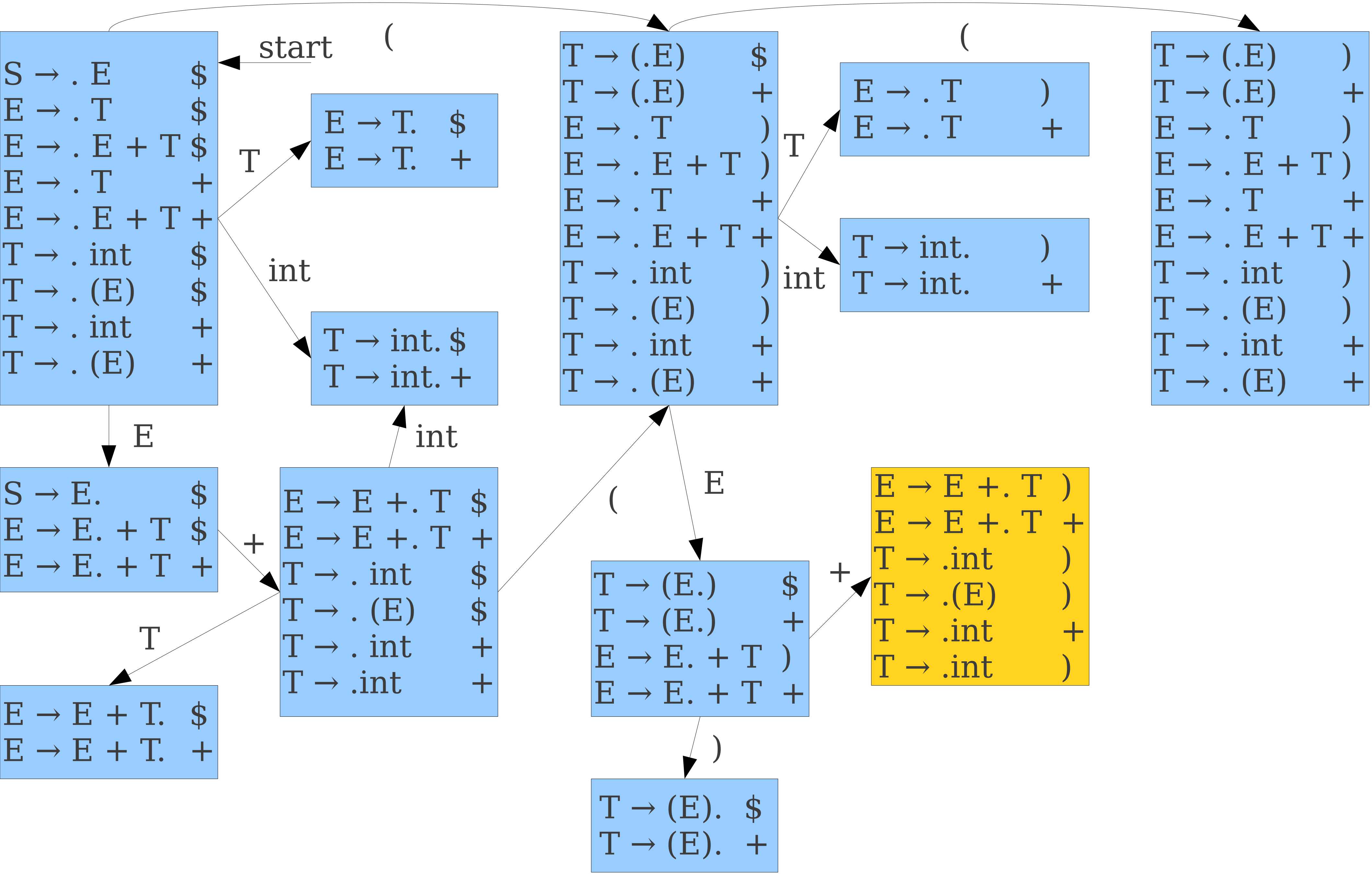
T → (.E))
T → (.E) +
E → . T)
E → . E + T)
E → . T +
E → . E + T +
T → . int)
T → . (E))
T → . int +
T → . (E) +

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

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

T → (E). $
T → (E). +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

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

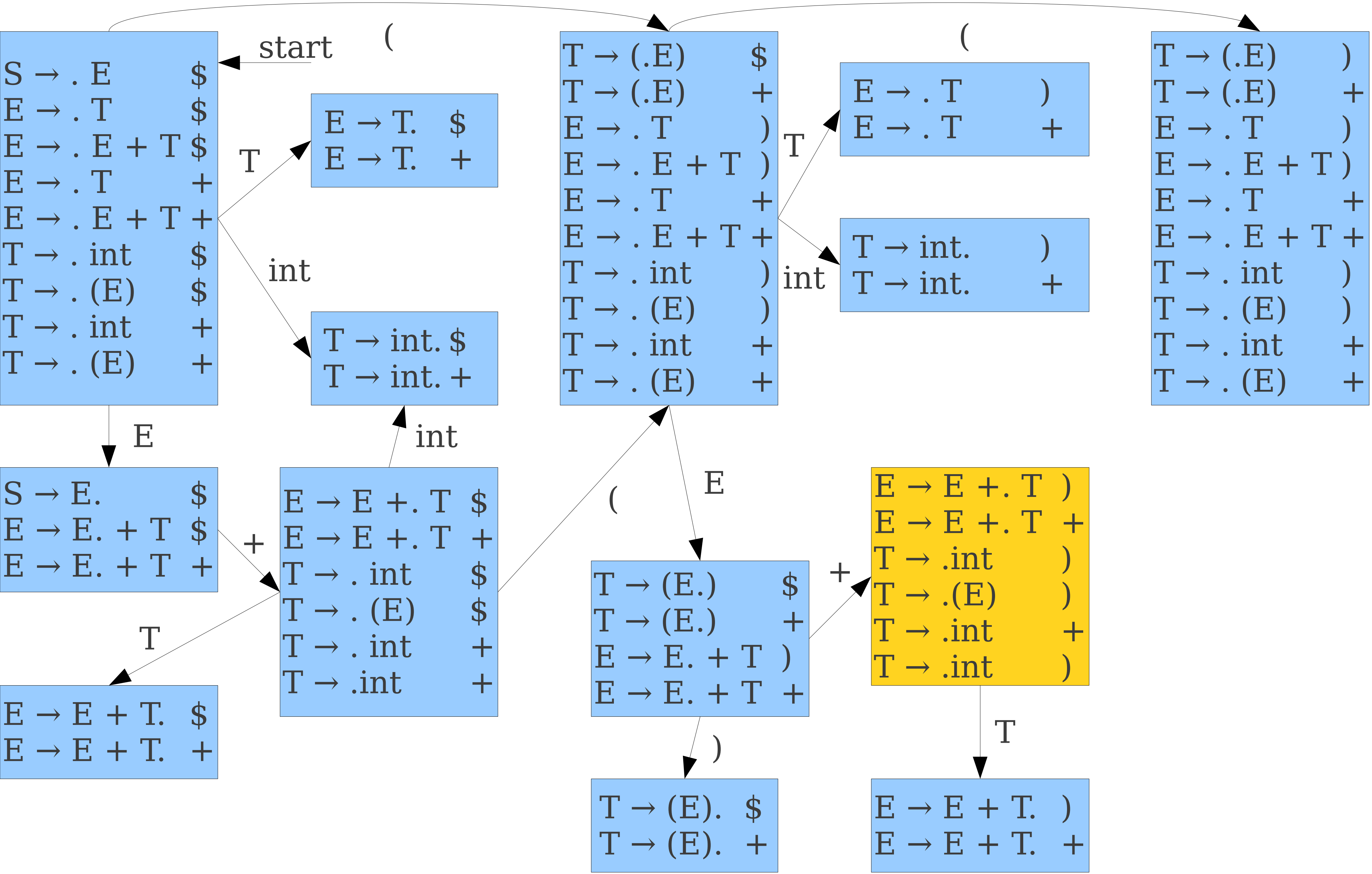
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

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

E → E + T. $
E → E + T. +

T → (E). $
T → (E). +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

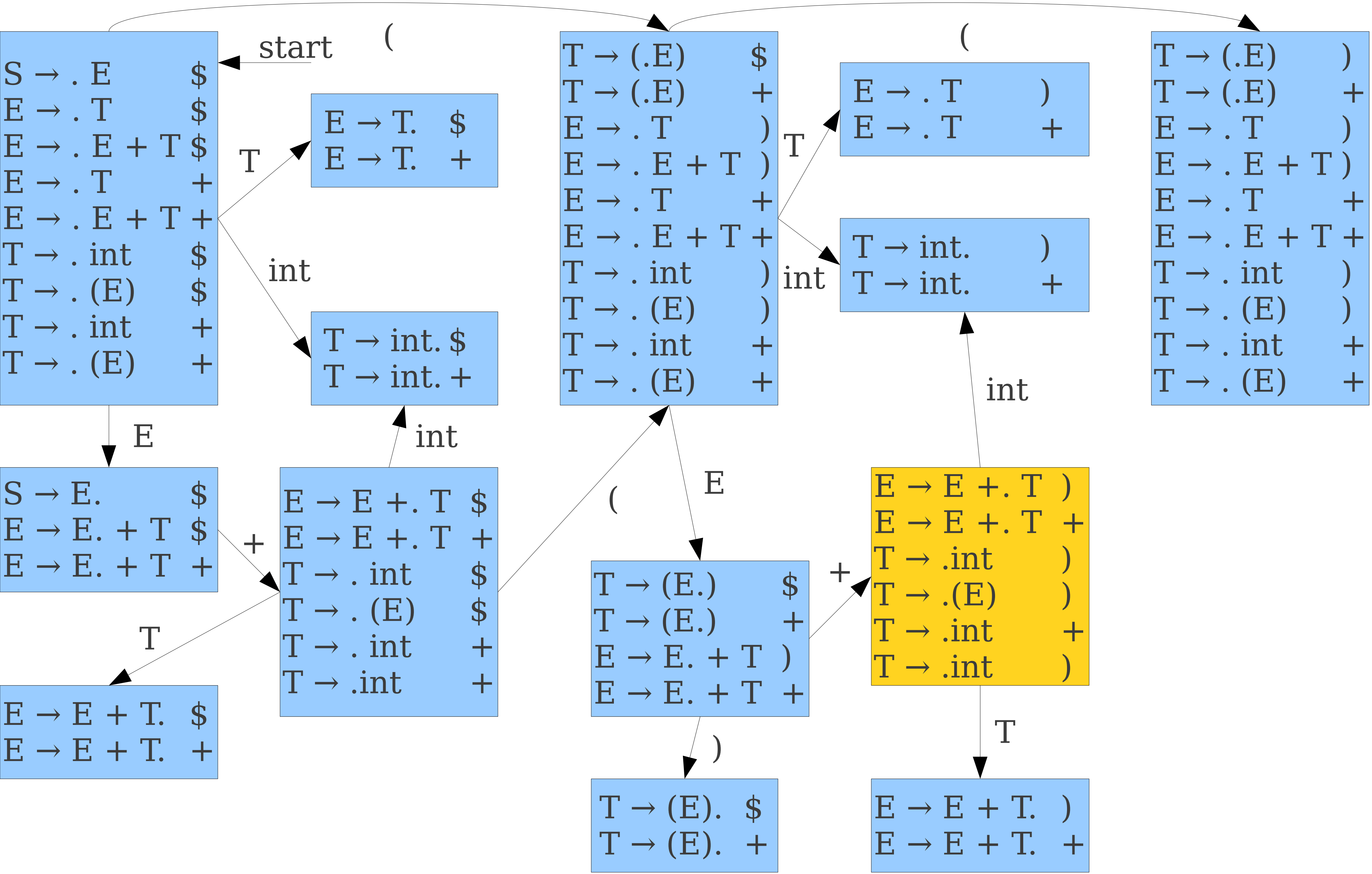
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

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

T → (E). $
T → (E). +

E → E + T.)
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T.   +

T → int. $
T → int. +

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

E → . T   )
E → . T   +

T → int.   )
T → int.   +

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

S → E. $
E → E. + T $
E → E. + T   +

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

E → E + T. $
E → E + T.   +

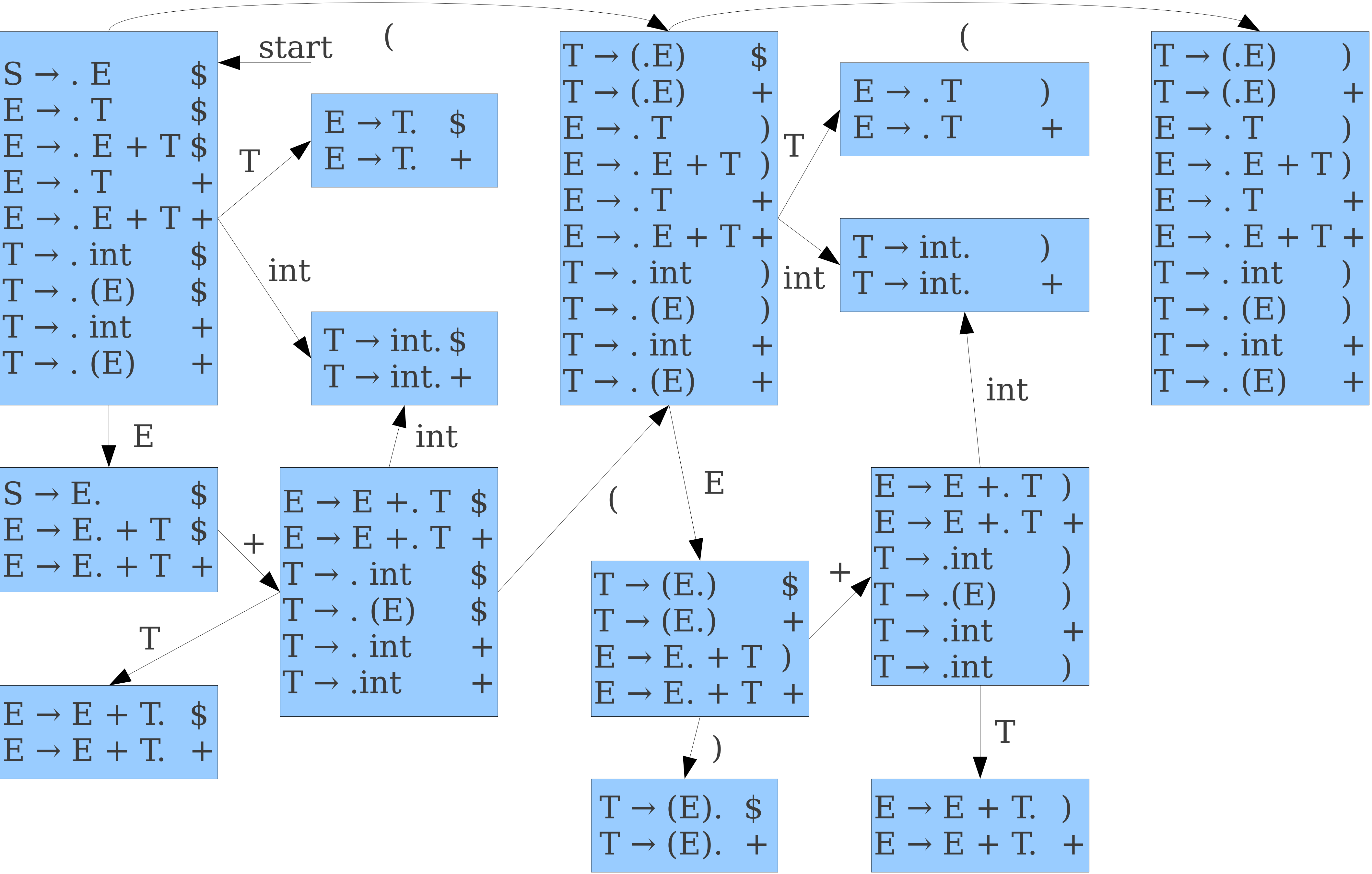
T → (E.) $
T → (E.)   +
E → E. + T)
E → E. + T +

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

T → (E). $
T → (E).   +

E → E + T.   )
E → E + T.   +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

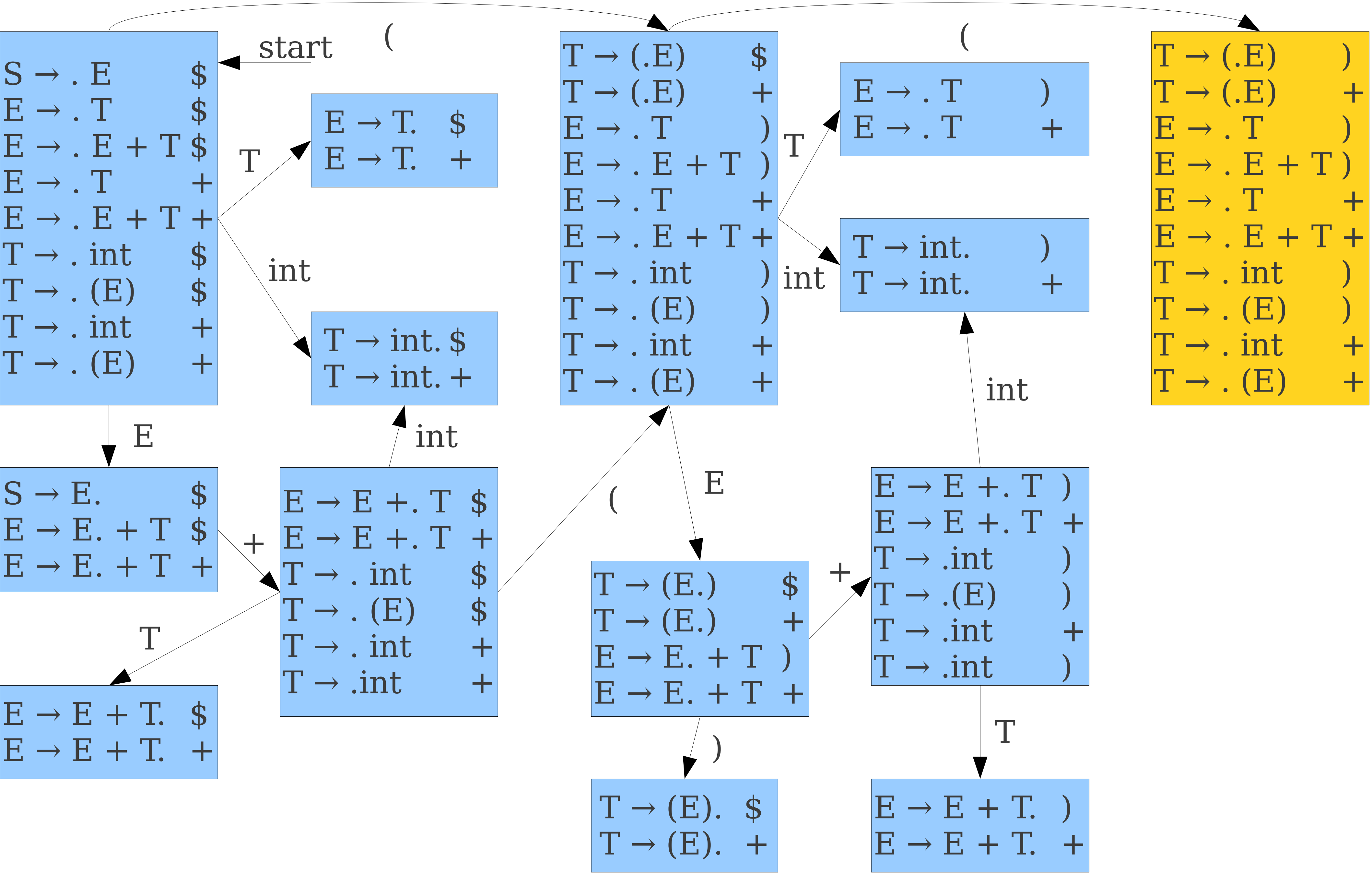
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

T → (E). $
T → (E). +

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

E → E + T.)
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

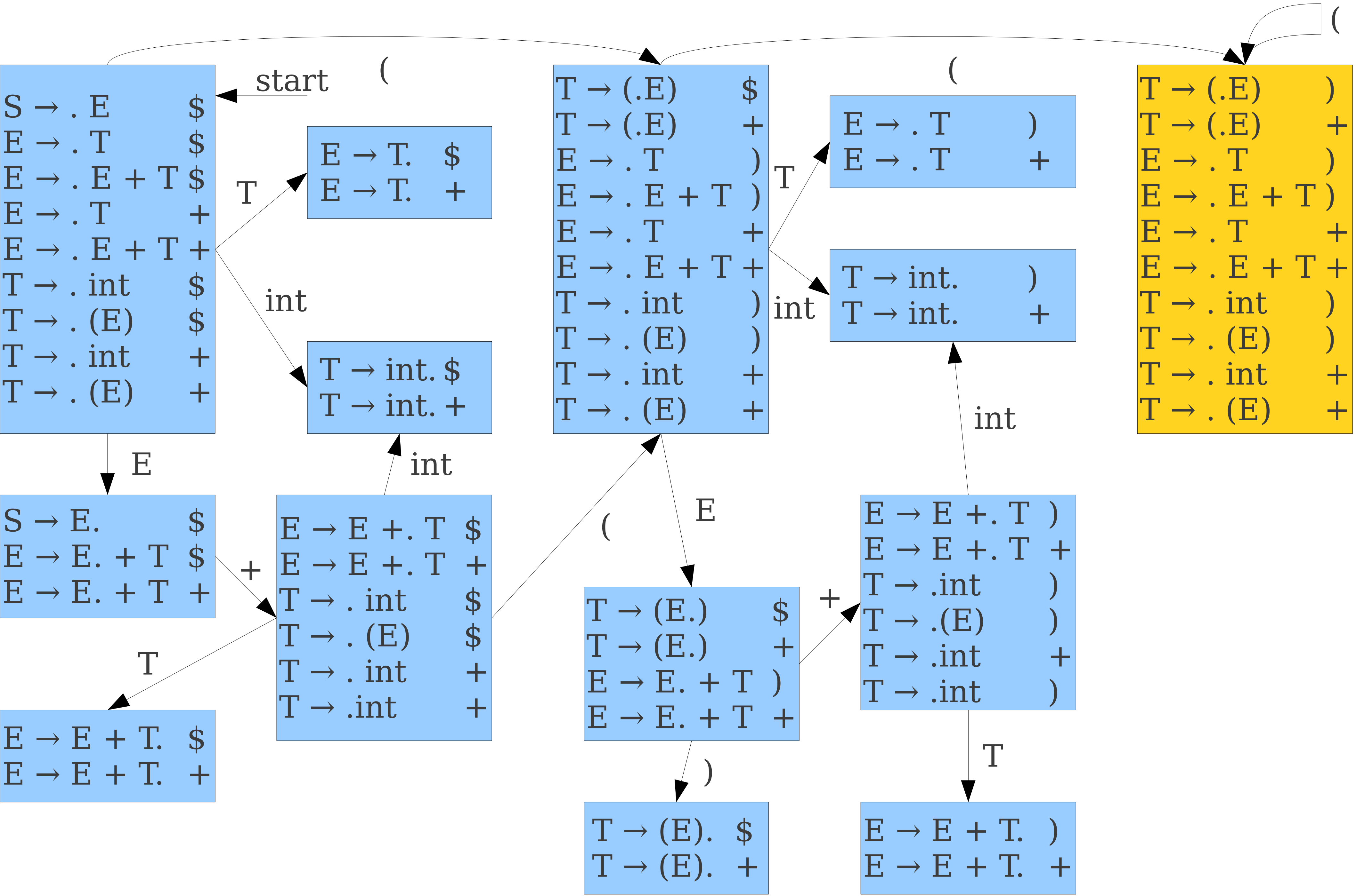
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

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

T → (E). $
T → (E). +

E → E + T.)
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

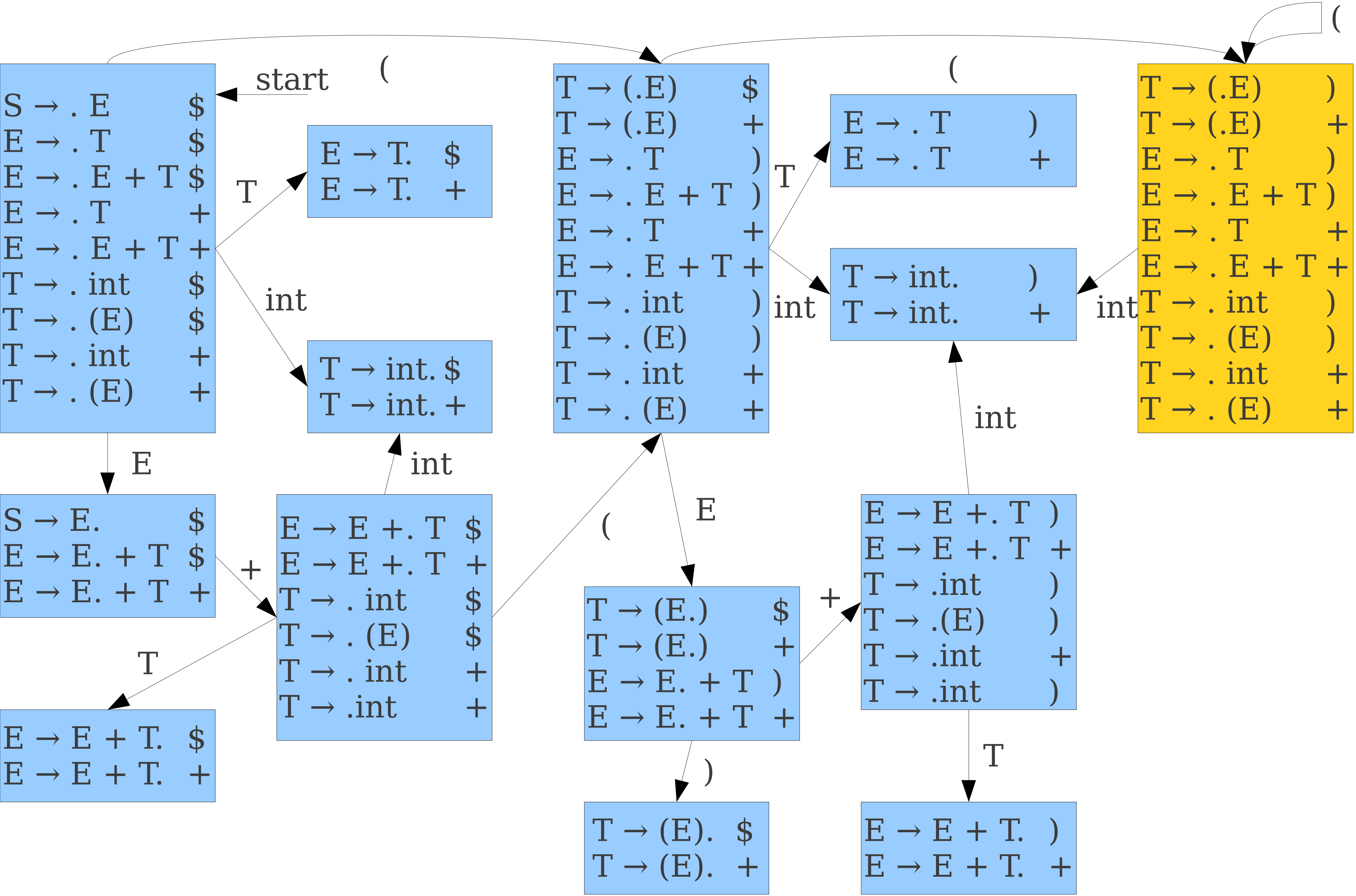
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

T → (E). $
T → (E). +

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

E → E + T.)
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

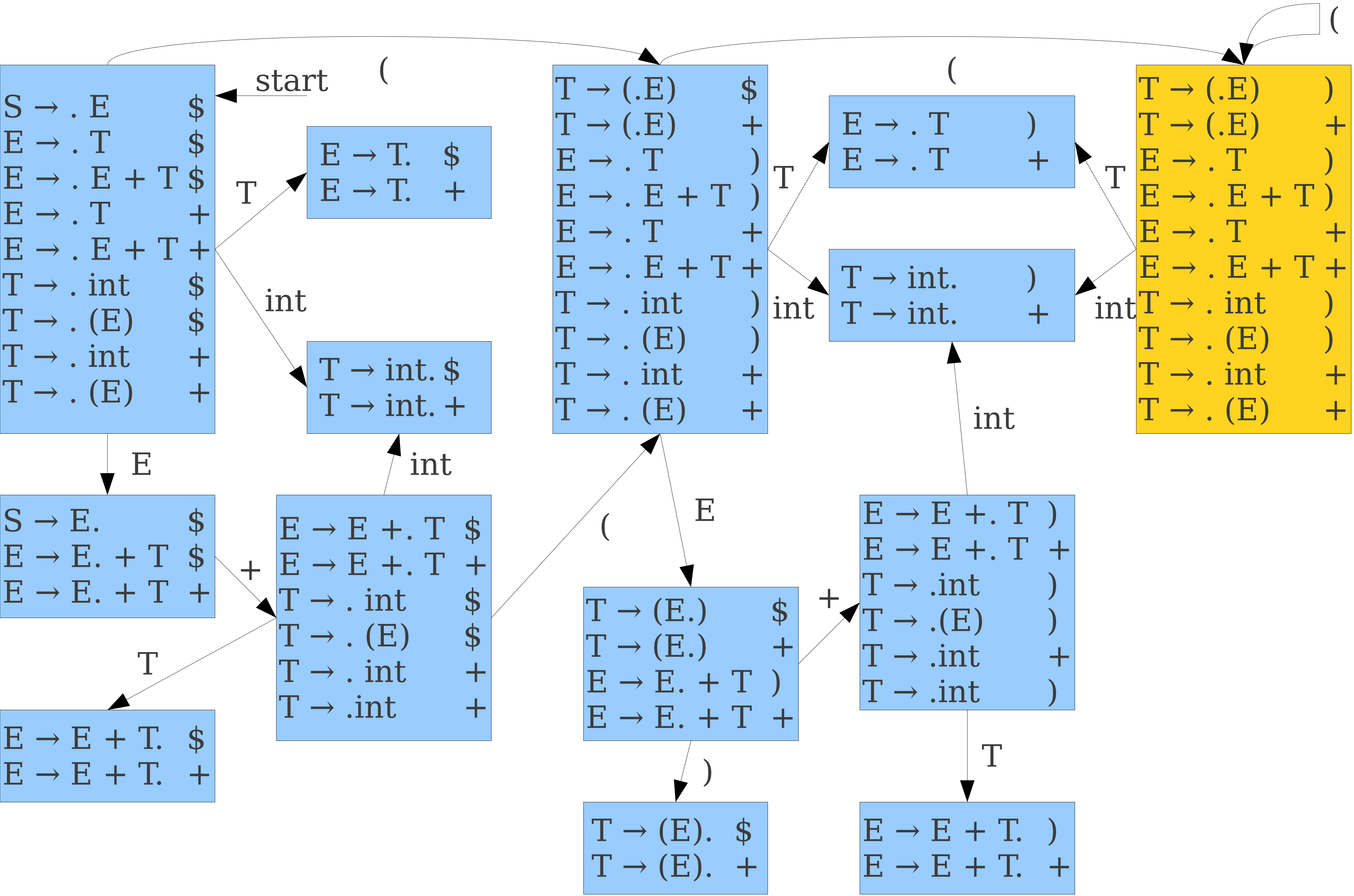
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

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

T → (E). $
T → (E). +

E → E + T.)
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

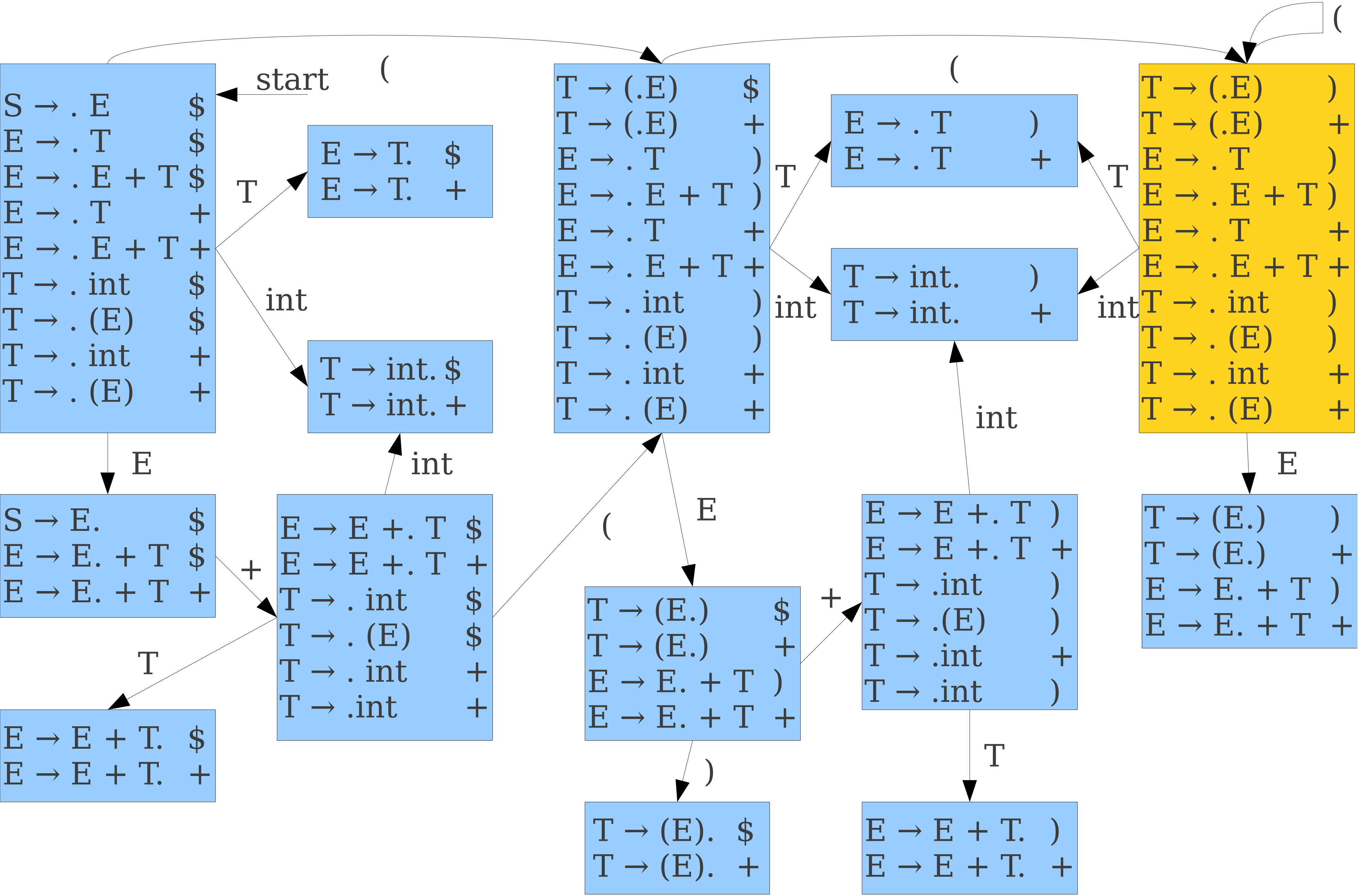
T → (E.) $
T → (E.) +
E → E. + T)
E → E. + T +

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

T → (E). $
T → (E). +

E → E + T.)
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

S → E. $
E → E. + T $
E → E. + T +

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

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

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

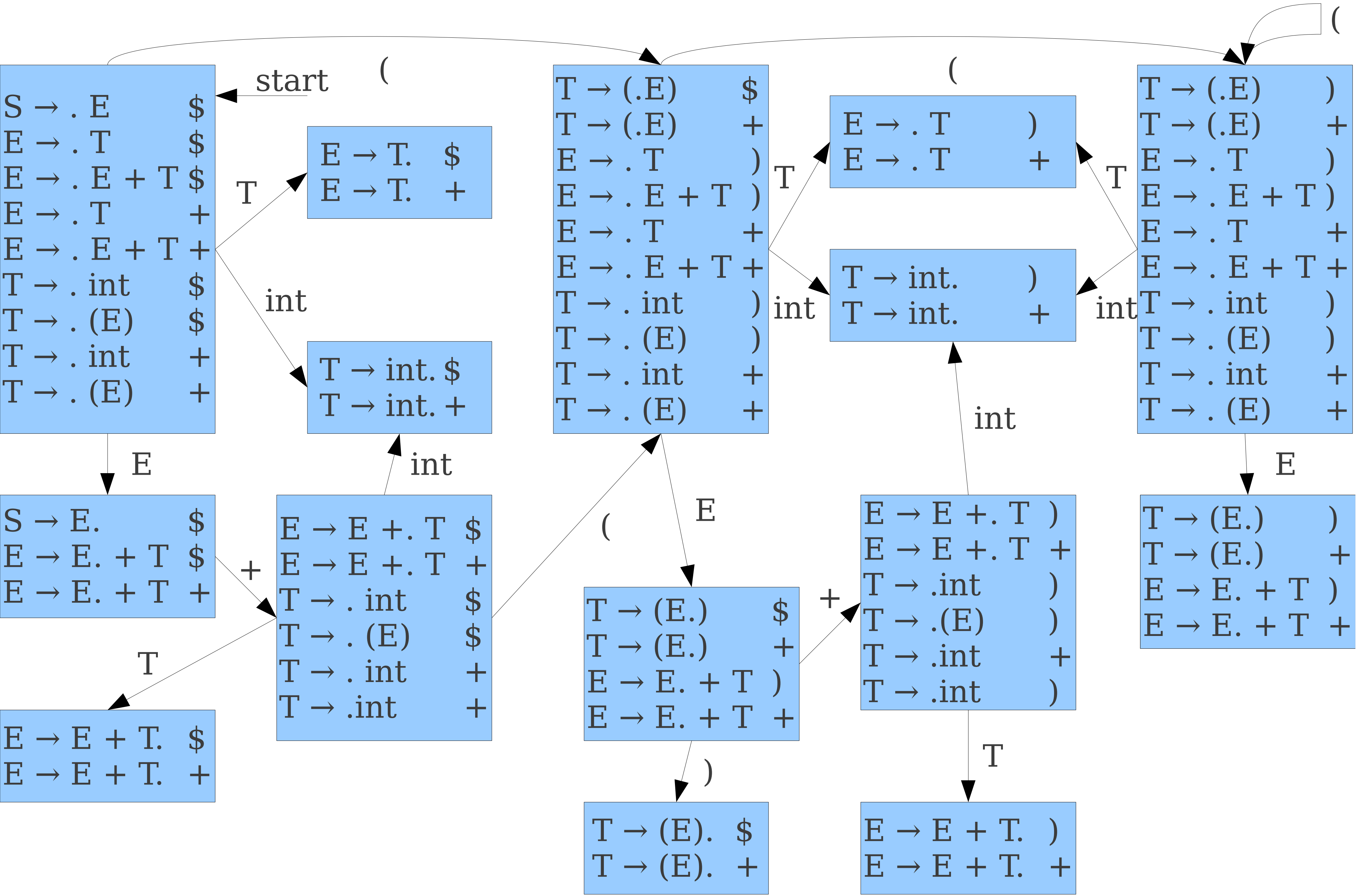
T → (E.))
T → (E.) +
E → E. + T)
E → E. + T +

E → E + T. $
E → E + T. +

T → (E). $
T → (E). +

E → E + T.)
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

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

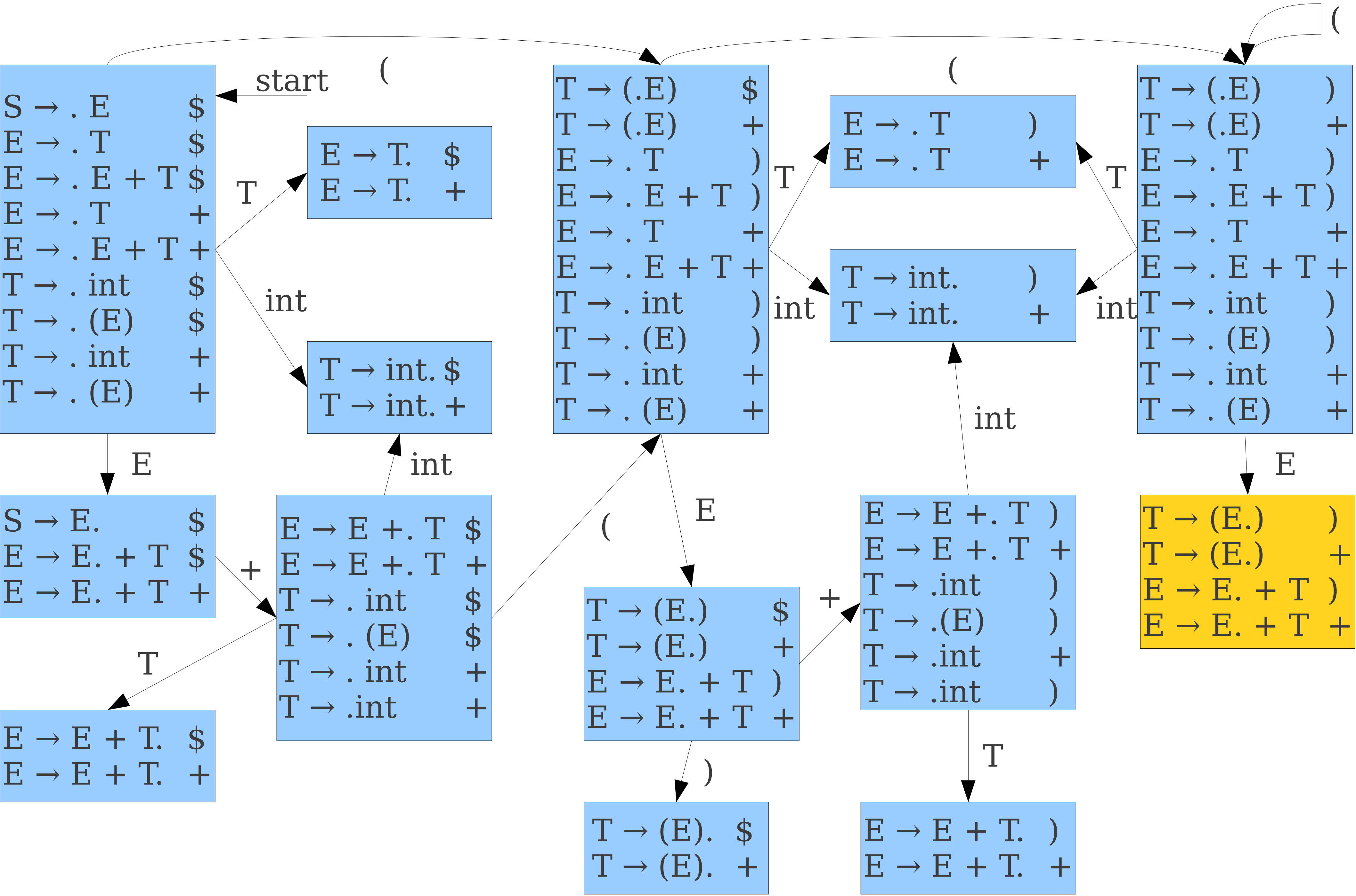
E → E +. T)
E → E +. T +
T → .int)
T → .(E))
T → .int +
T → .int)

T → (E). $
T → (E). +

E → E + T.)
E → E + T. +

T → (E.))
T → (E.) +
E → E. + T)
E → E. + T +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

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

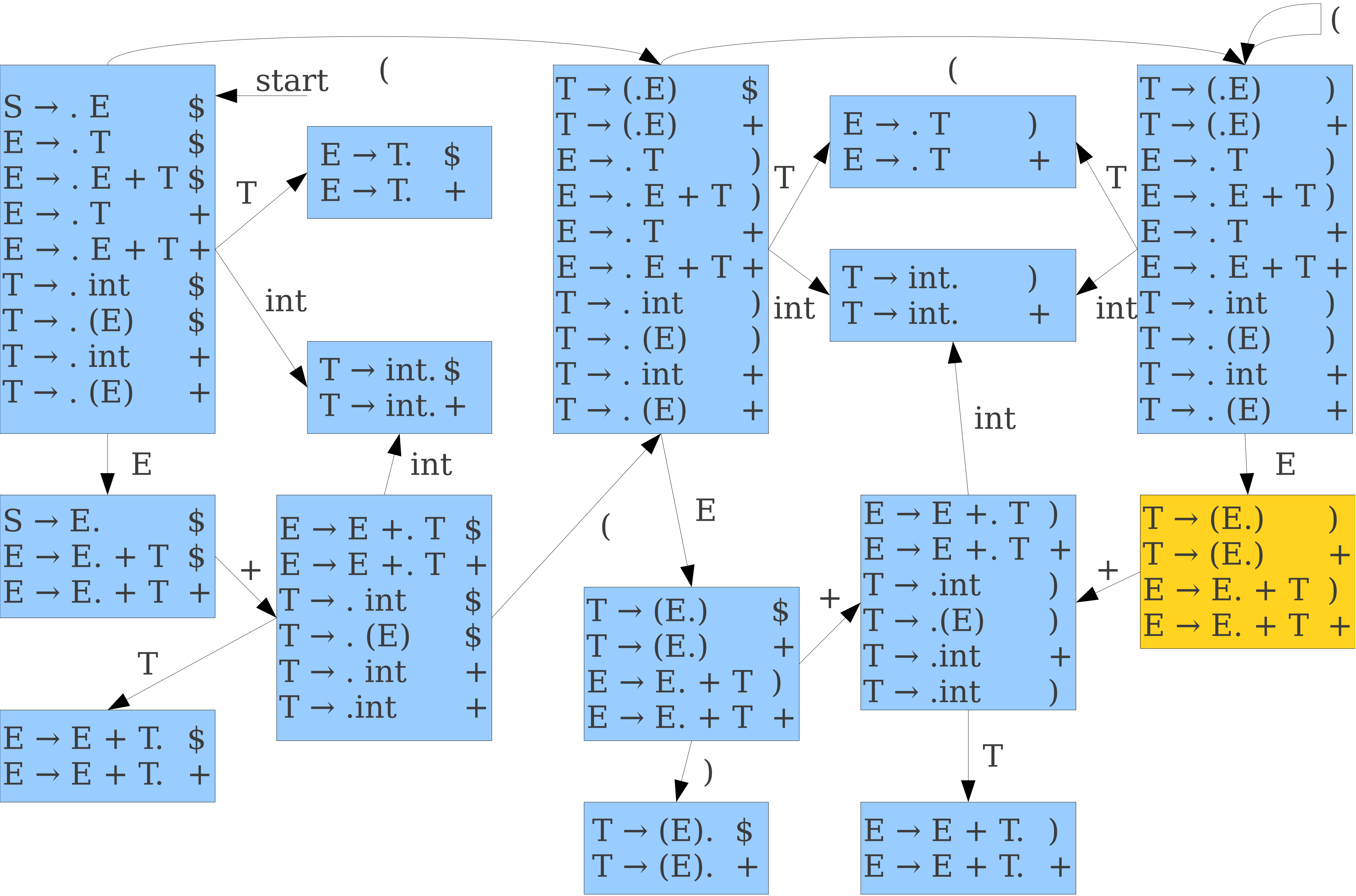
E → E +. T)
E → E +. T +
T → .int)
T → .(E))
T → .int +
T → .int)

T → (E). $
T → (E). +

E → E + T.)
E → E + T. +

T → (E.))
T → (E.) +
E → E. + T)
E → E. + T +

Deterministic LR(1) Automata



start

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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

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

T → (E). $
T → (E). +

E → . T)
E → . T +

T → int.)
T → int. +

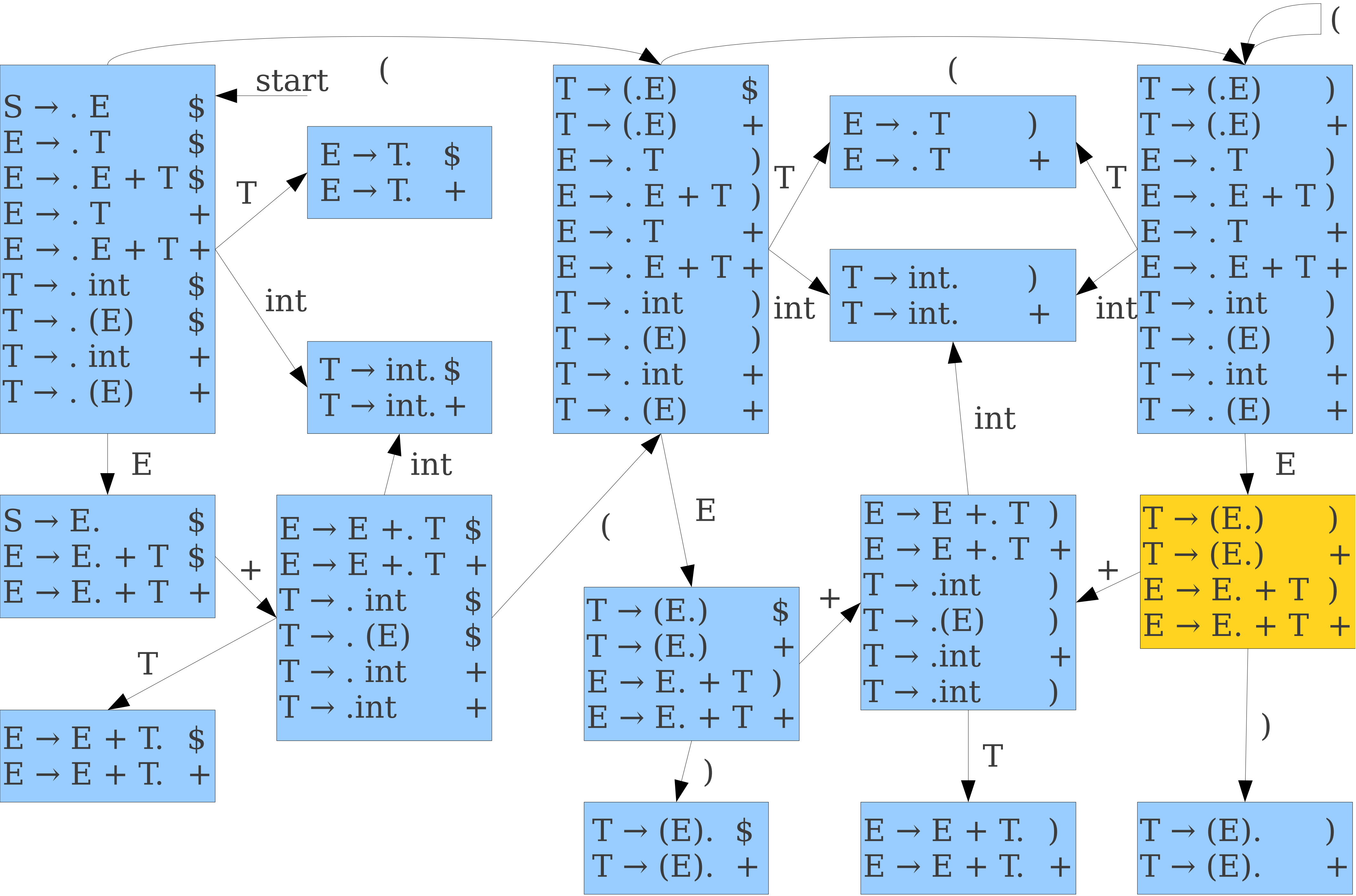
E → E +. T)
E → E +. T +
T → .int)
T → .(E))
T → .int +
T → .int)

E → E + T.)
E → E + T. +

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

T → (E.))
T → (E.) +
E → E. + T)
E → E. + T +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

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

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

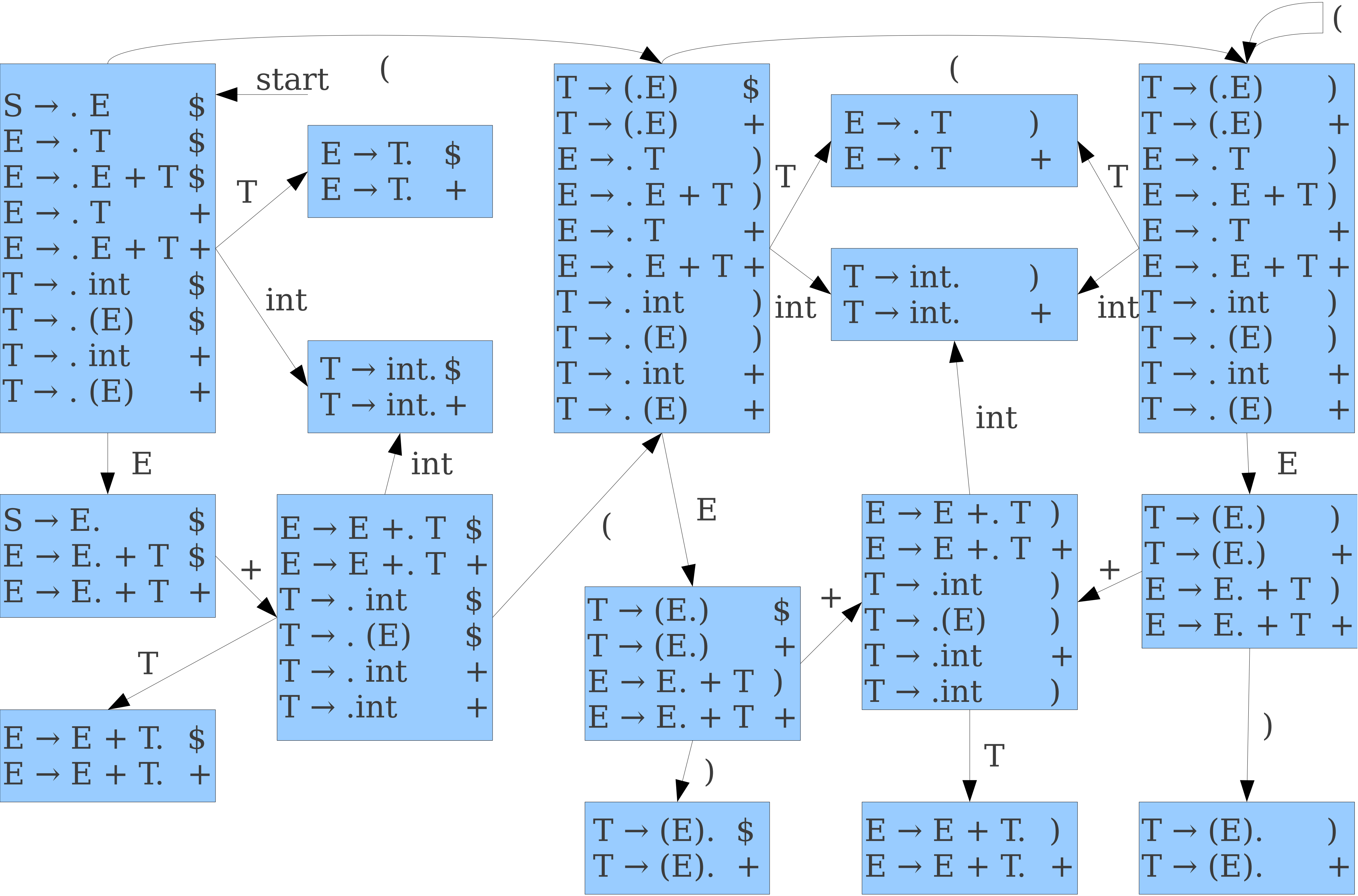
T → (E.))
T → (E.) +
E → E. + T)
E → E. + T +

T → (E). $
T → (E). +

E → E + T.)
E → E + T. +

T → (E).)
T → (E). +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

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

E → . T)
E → . T +

T → int.)
T → int. +

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

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

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

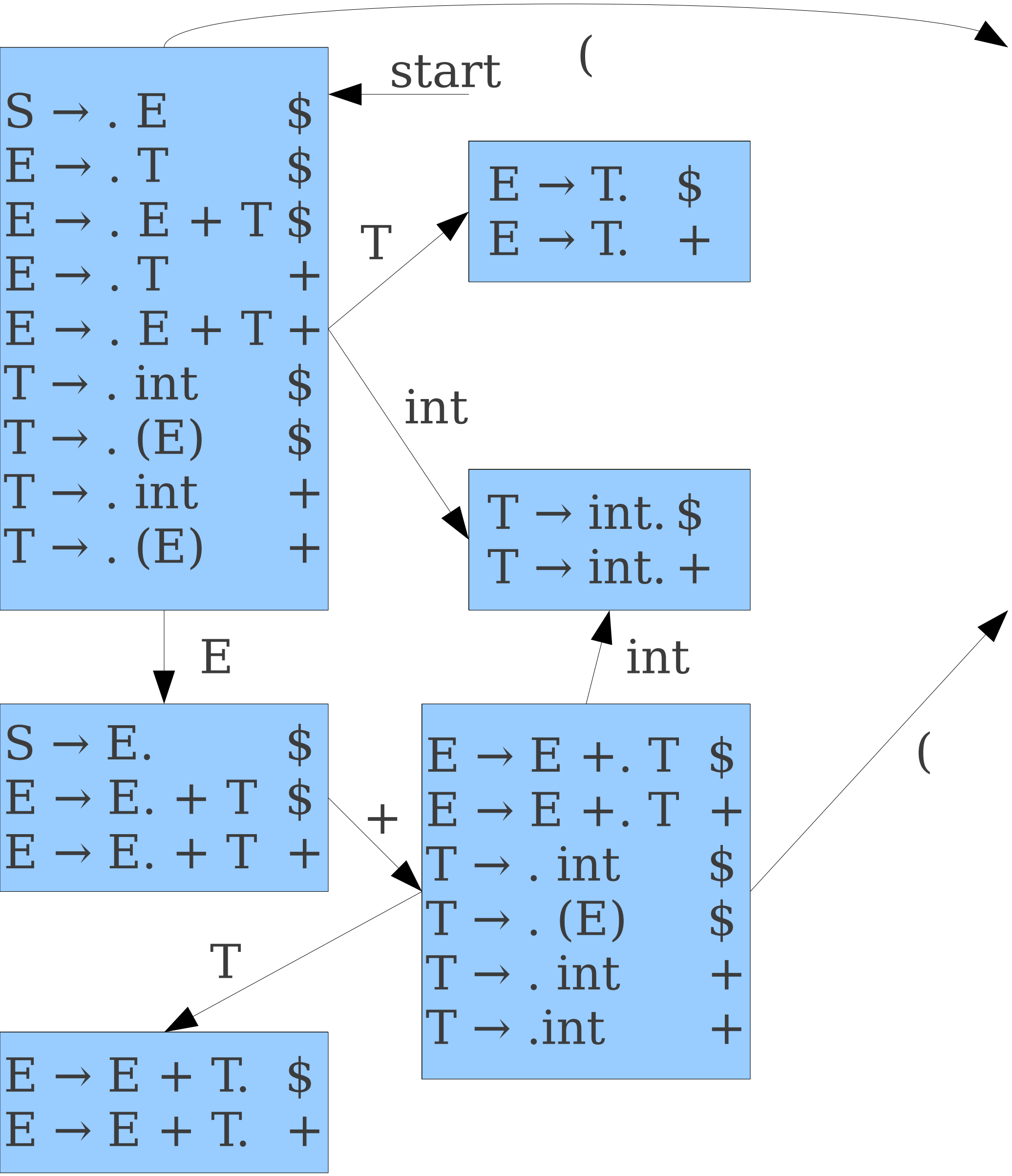
T → (E.))
T → (E.) +
E → E. + T)
E → E. + T +

T → (E). $
T → (E). +

E → E + T.)
E → E + T. +

T → (E).)
T → (E). +

Deterministic LR(1) Automata



start

(

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

E

S → E. $
E → E. + T $
E → E. + T +

\+

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

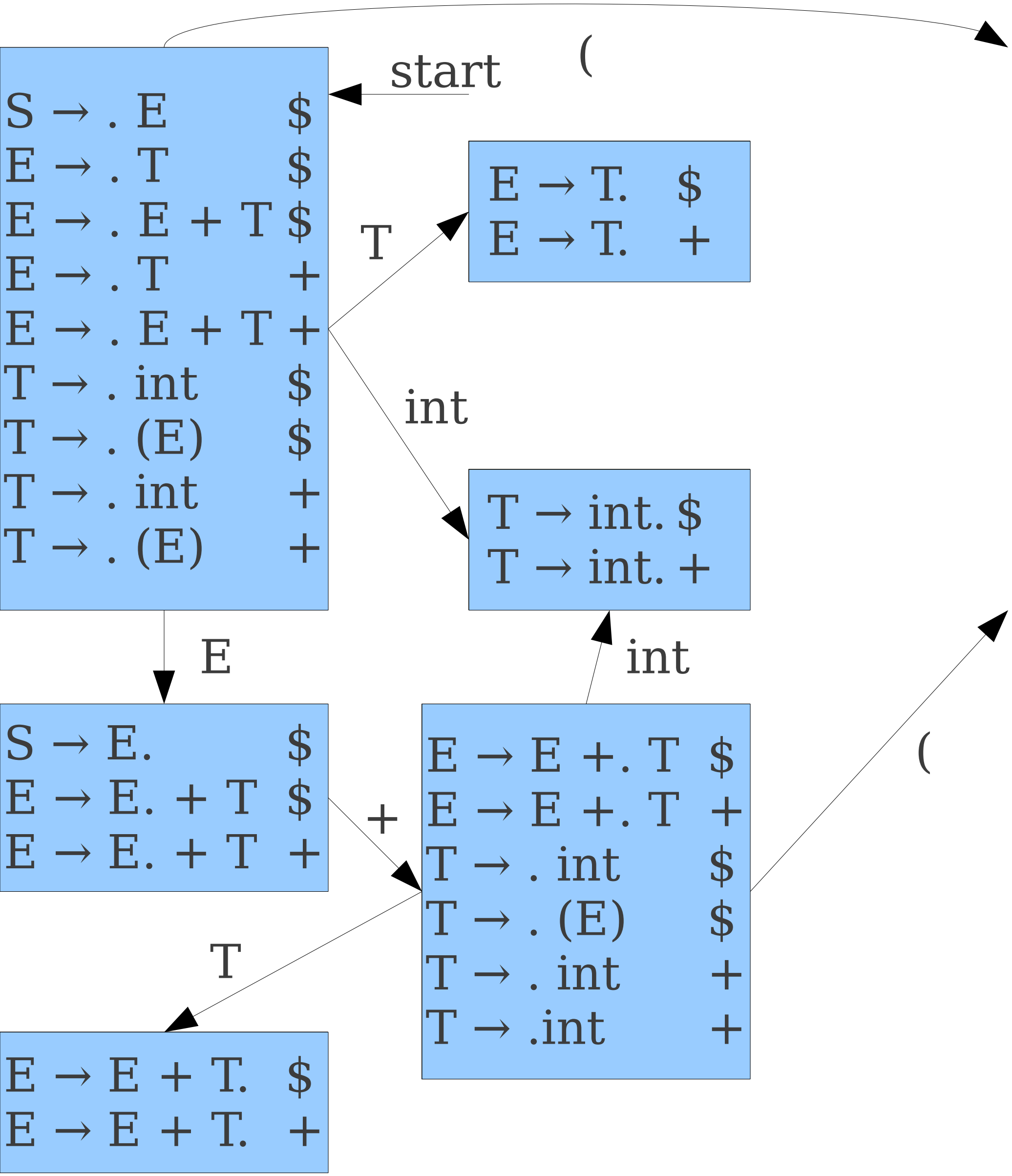
int

(

T

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

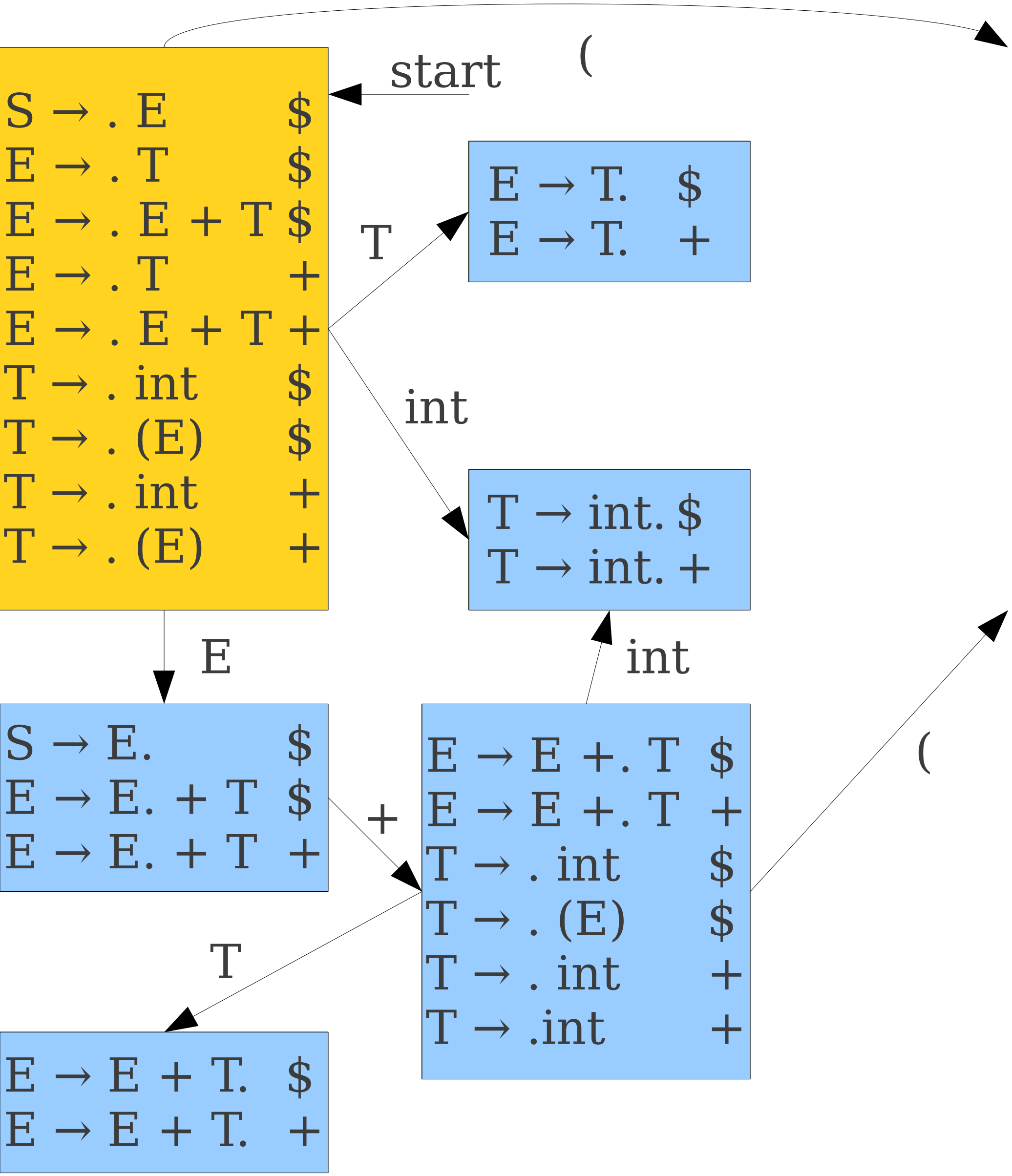
S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

`int`	`+`	`int`	`$`

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

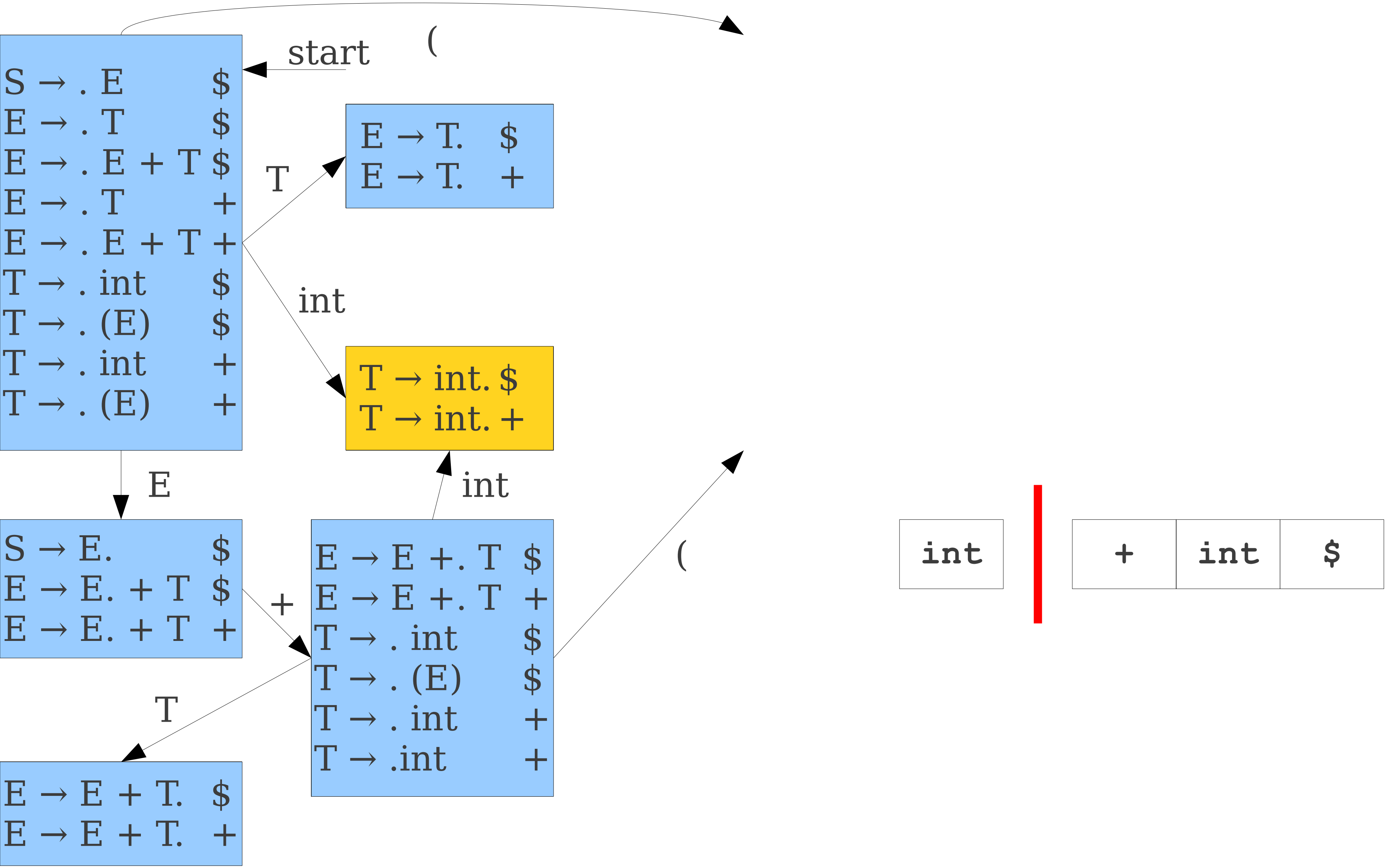
S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

`int`	`+`	`int`	`$`

Deterministic LR(1) Automata



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

E → T. $
E → T. +

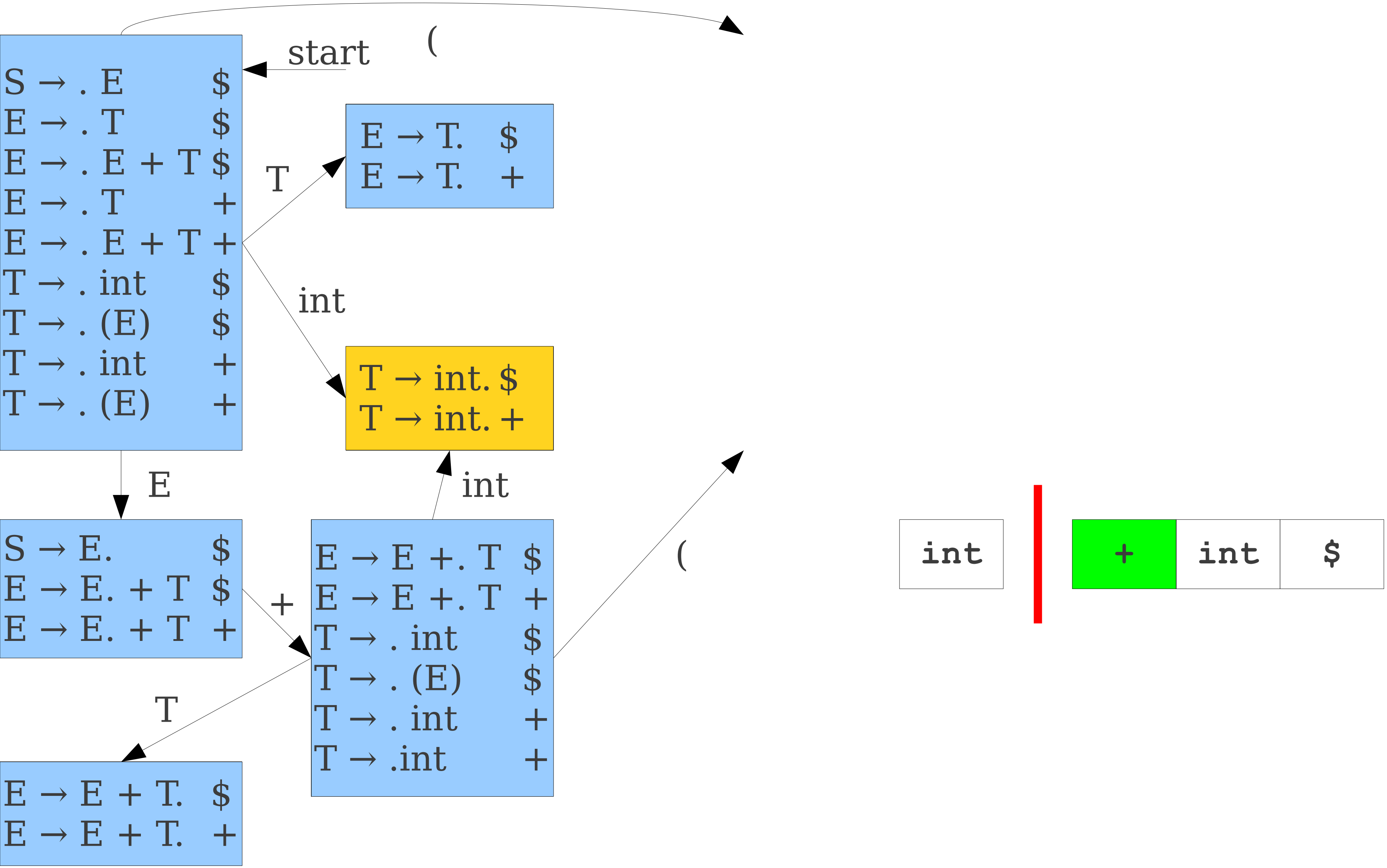
T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



start

(

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

E

S → E. $
E → E. + T $
E → E. + T +

+

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

int

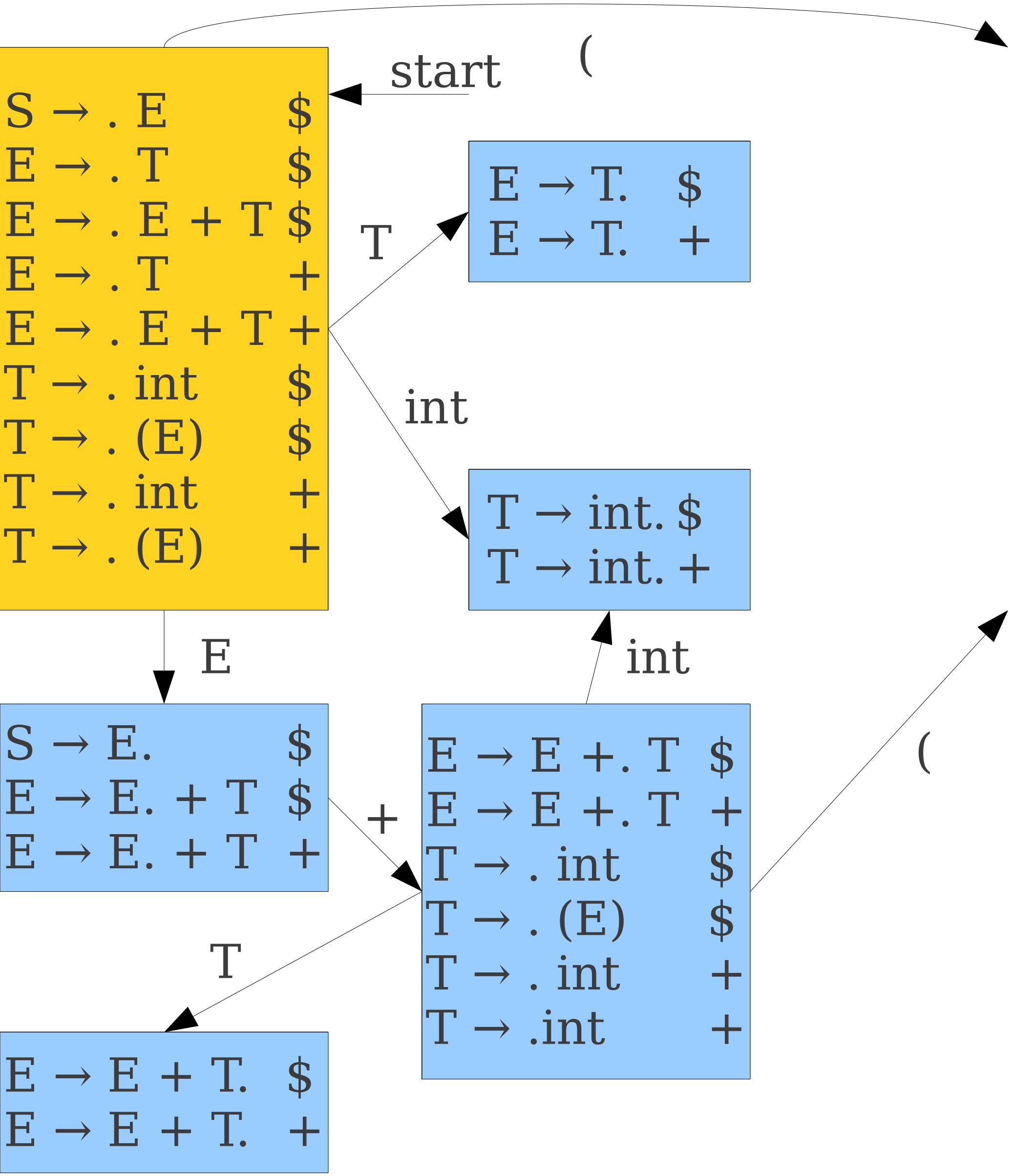
(

T

E → E + T. $
E → E + T. +

int		+	int	$

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

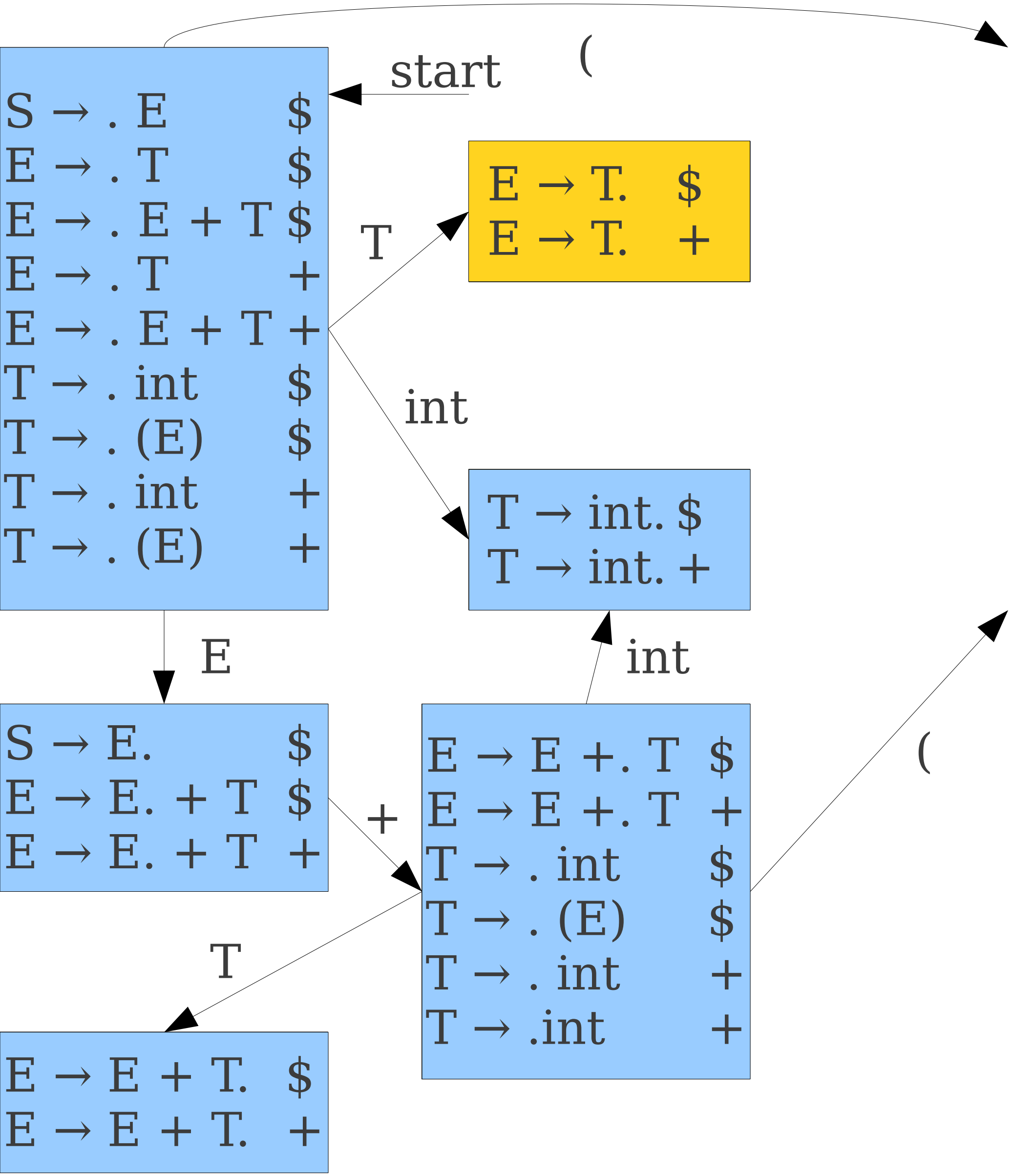
S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

T	|	+	int	$

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

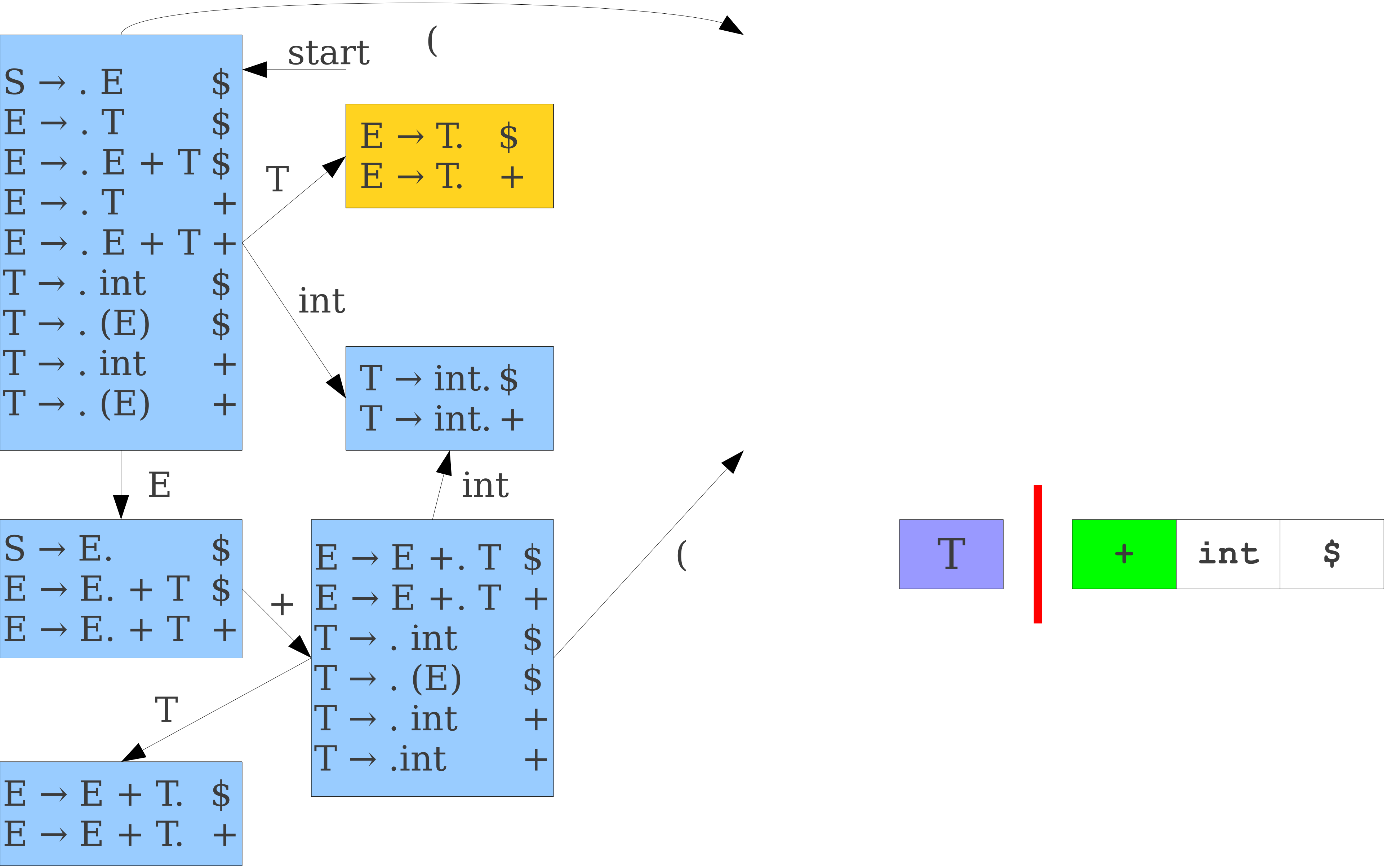
S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

T | + int $

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

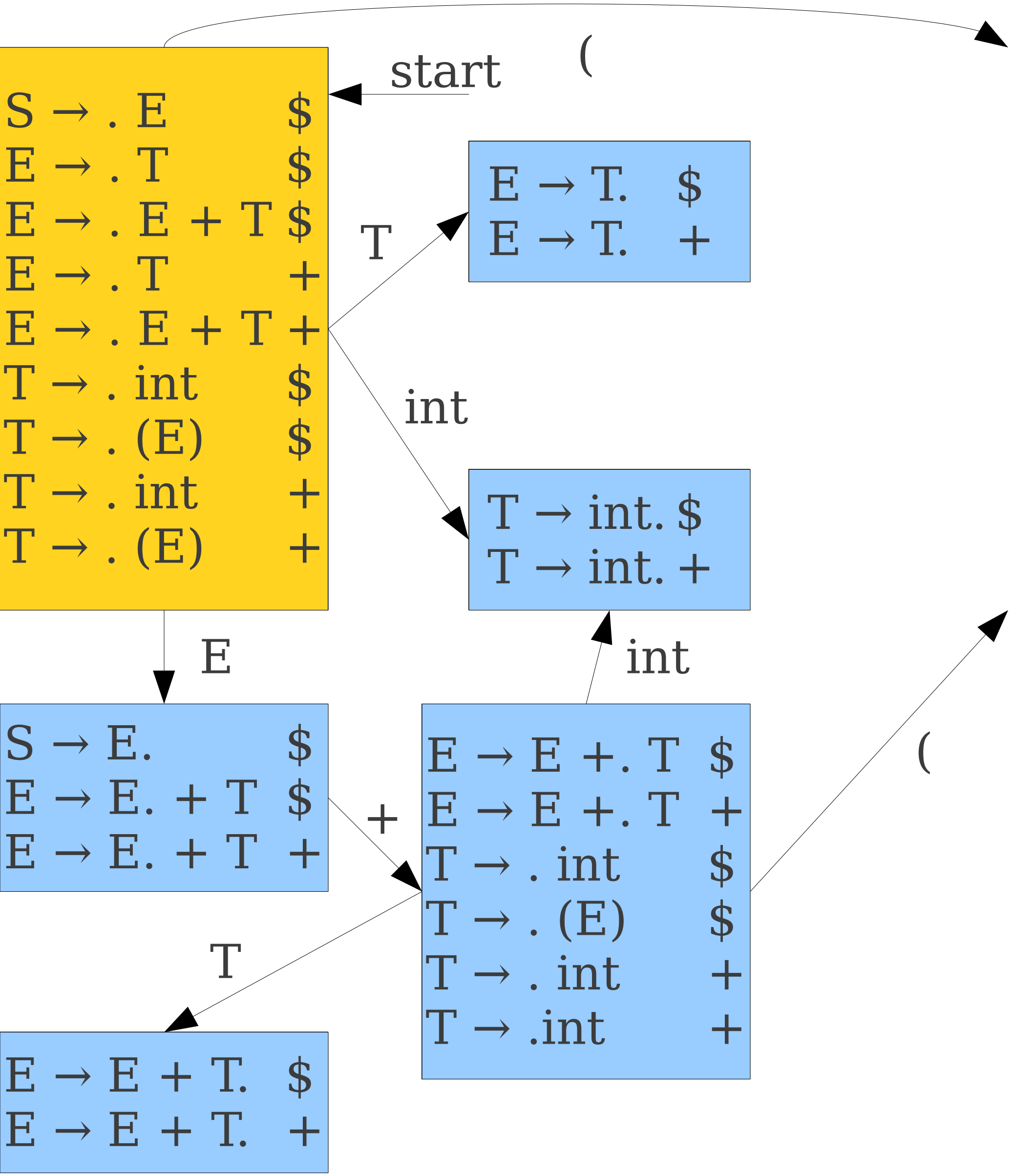
S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

T		+	`int`	`$`

Deterministic LR(1) Automata



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

E → T. $
E → T. +

T → int. $
T → int. +

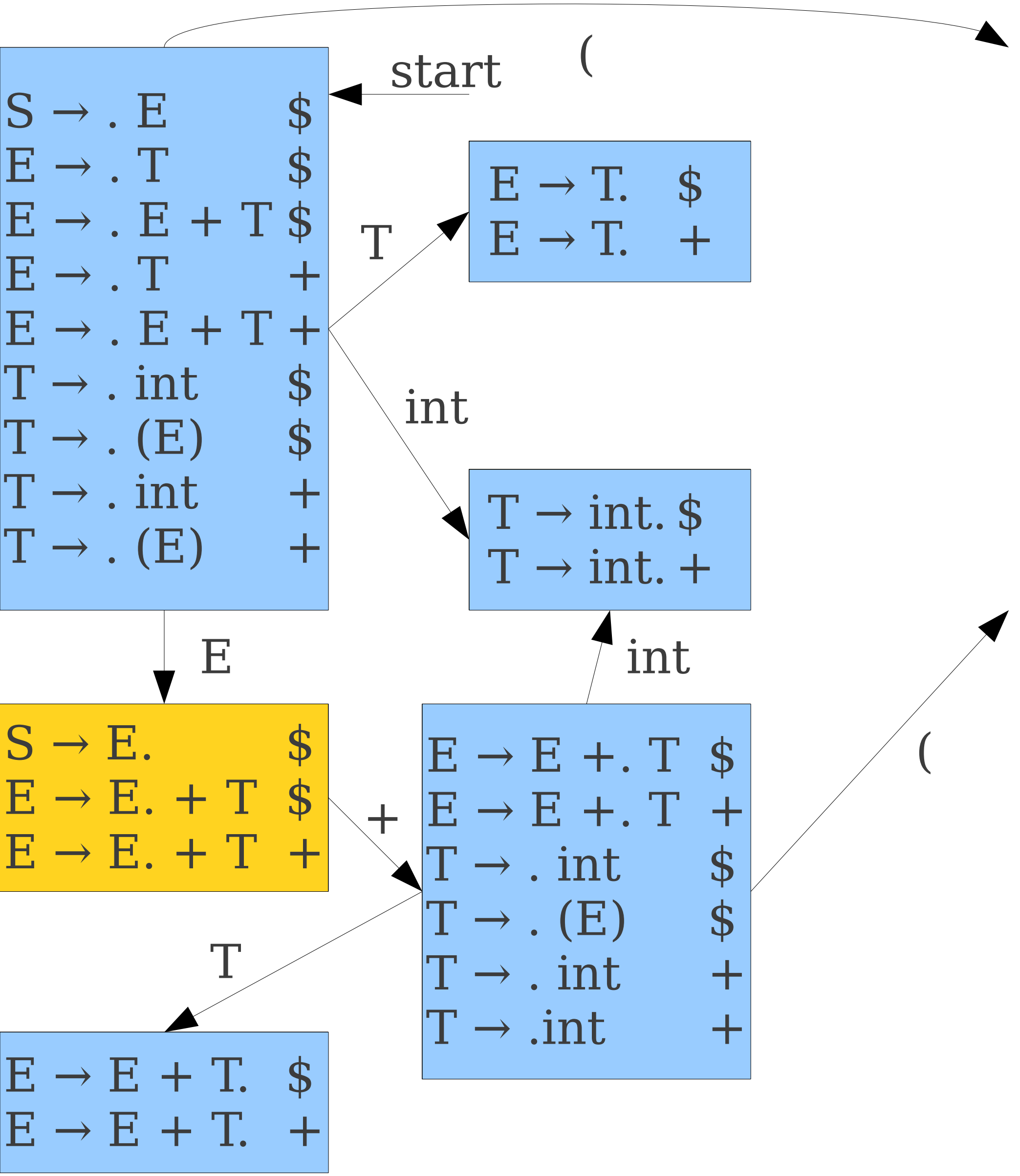
S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

E | `+` `int` `$`

Deterministic LR(1) Automata



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

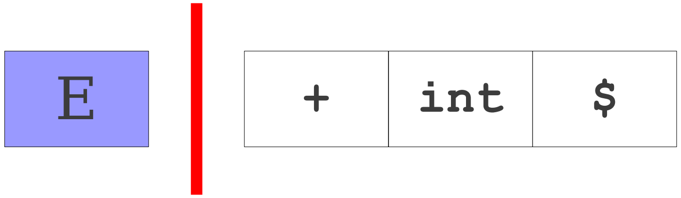
E → T. $
E → T. +

T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

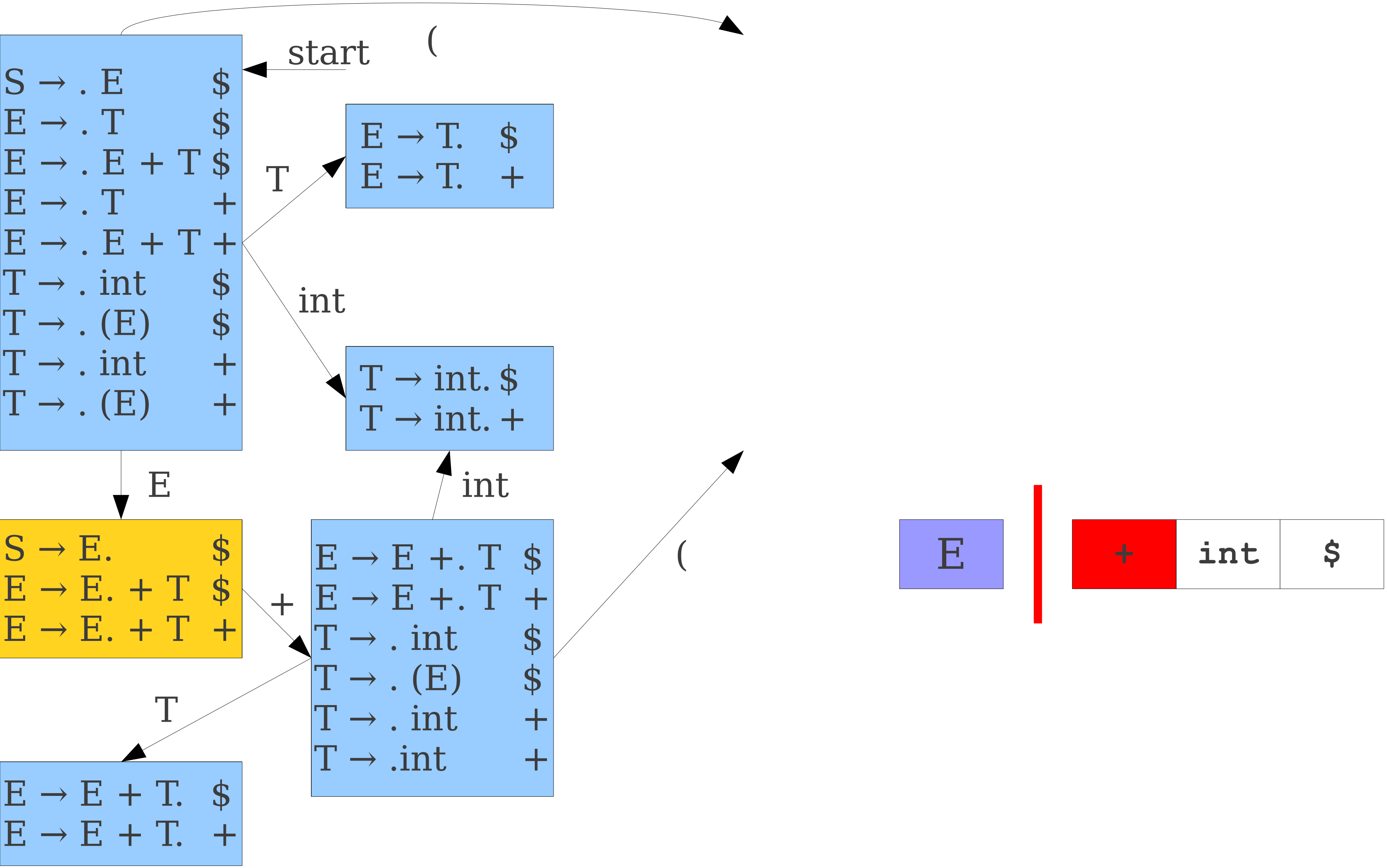
E → E +. T $
E → E +. T +
T → . int $
T → . (E) $
T → . int +
T → .int +

E → E + T. $
E → E + T. +



E	+	int	$

Deterministic LR(1) Automata



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

E → T. $
E → T. +

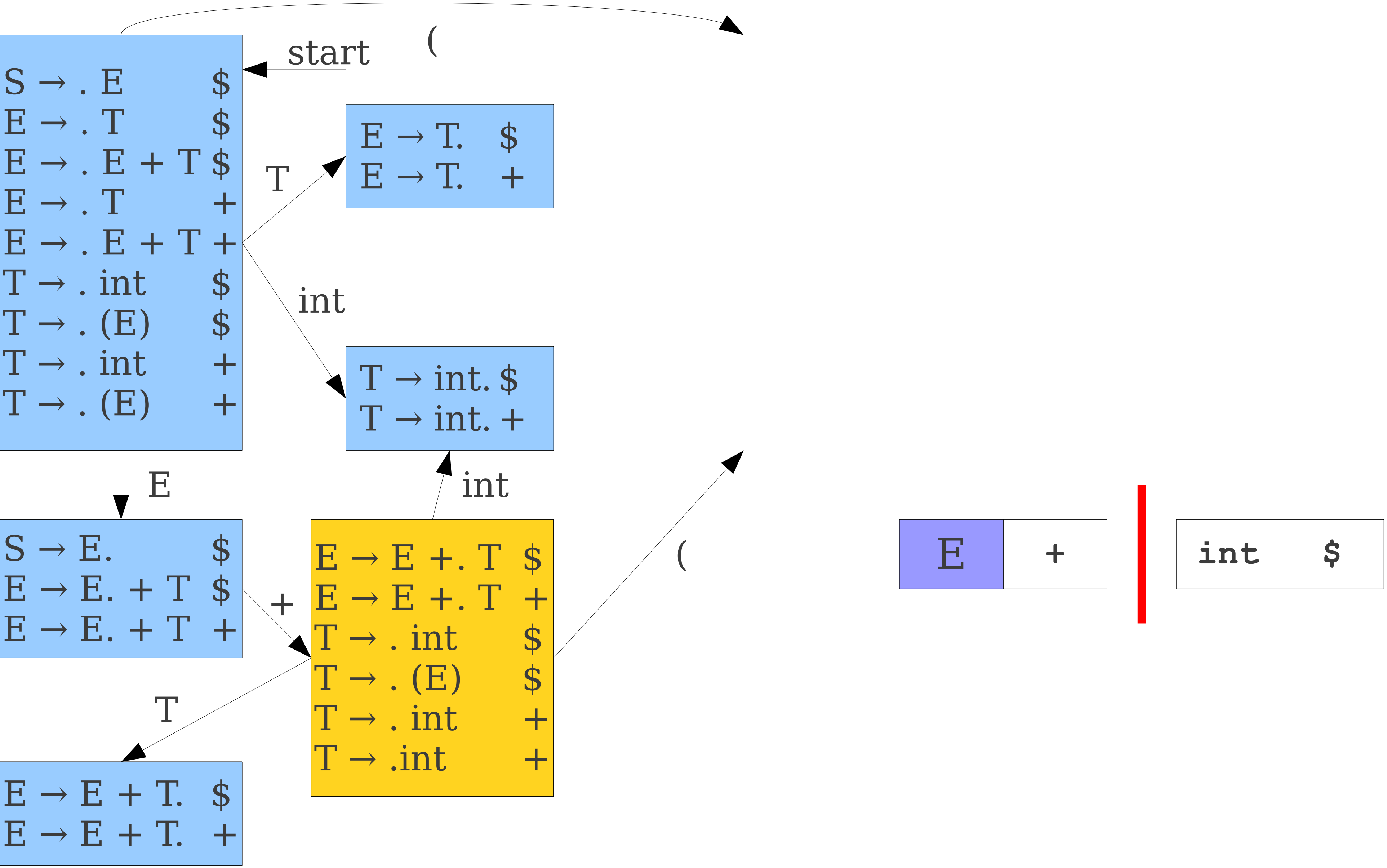
T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



start

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

(

E

S → E. $
E → E. + T $
E → E. + T +

+

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

int

(

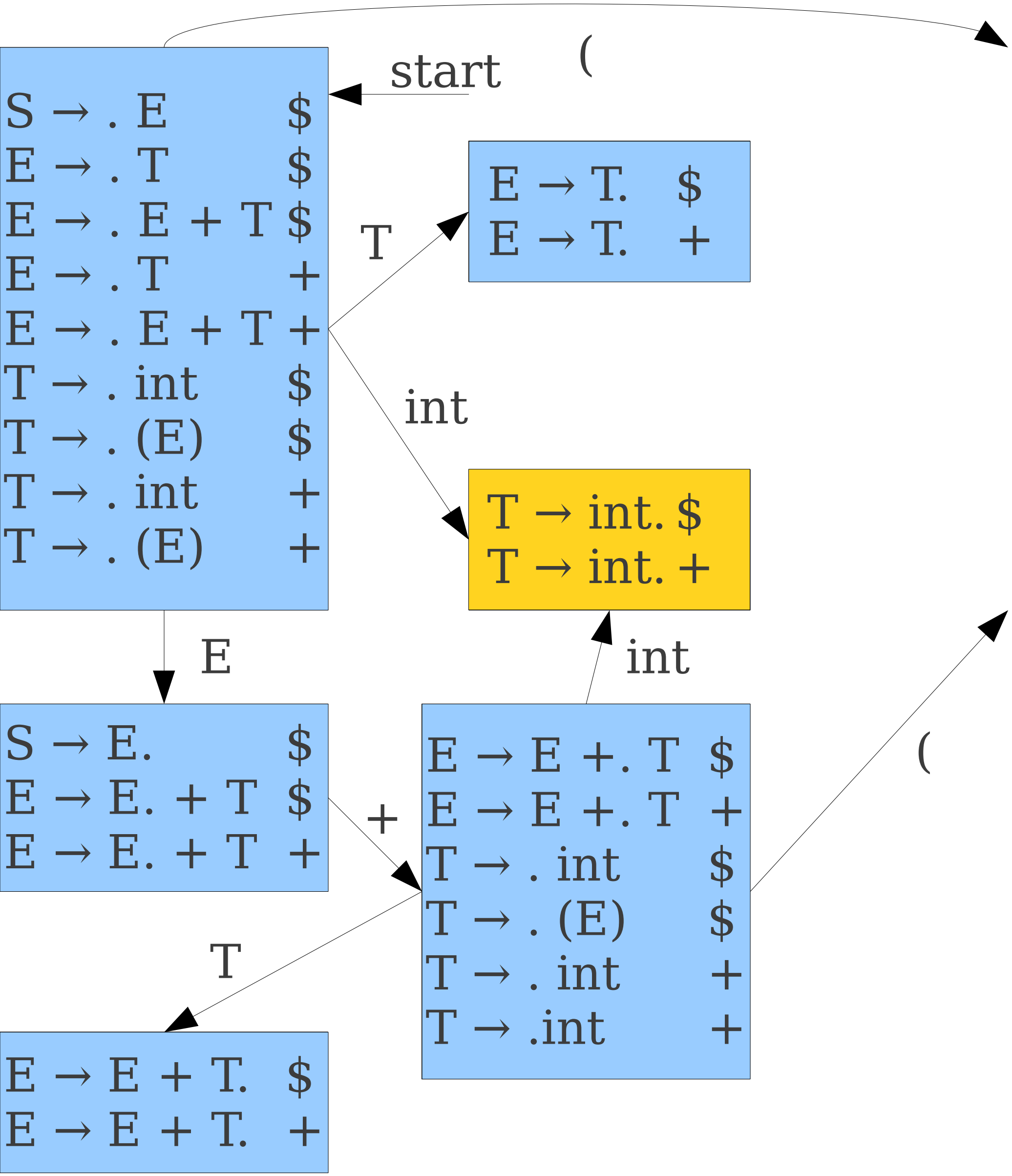
T

E → E + T. $
E → E + T. +

E | +

int | $

Deterministic LR(1) Automata



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

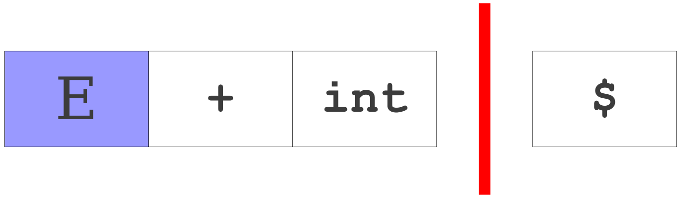
E → T. $
E → T. +

T → int. $
T → int. +

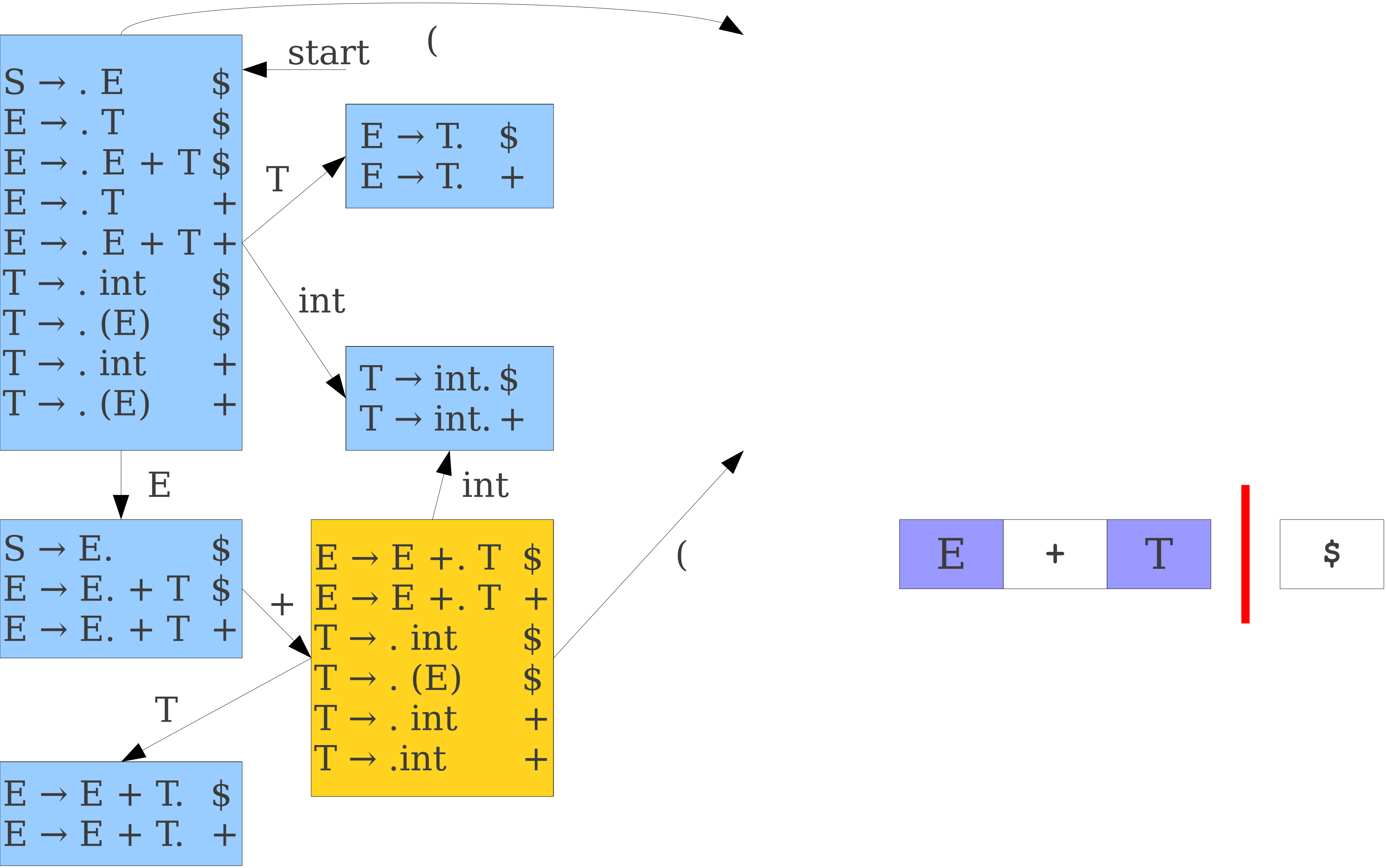
S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +



Deterministic LR(1) Automata



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

E → T. $
E → T. +

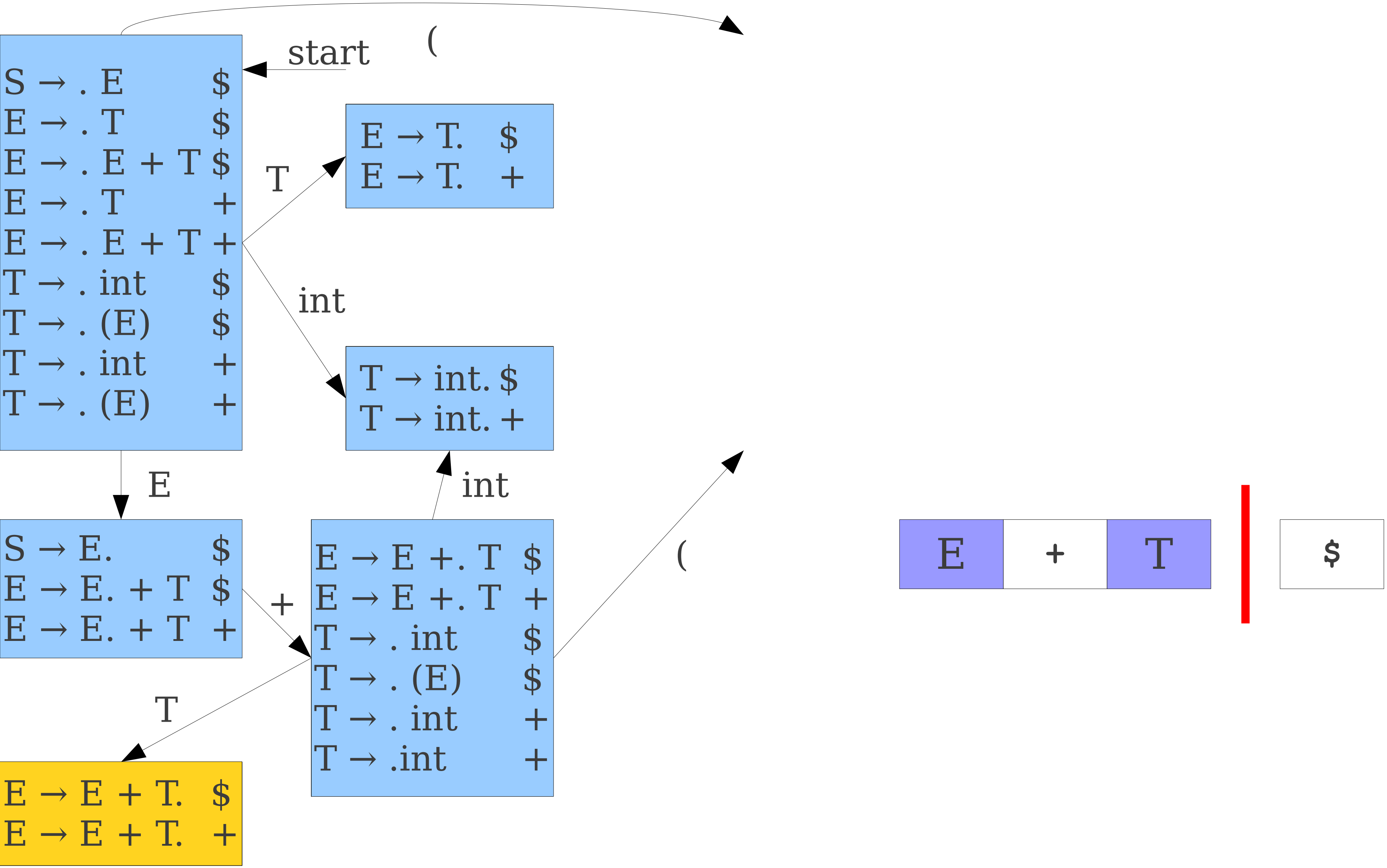
T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



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

E → T. $
E → T. +

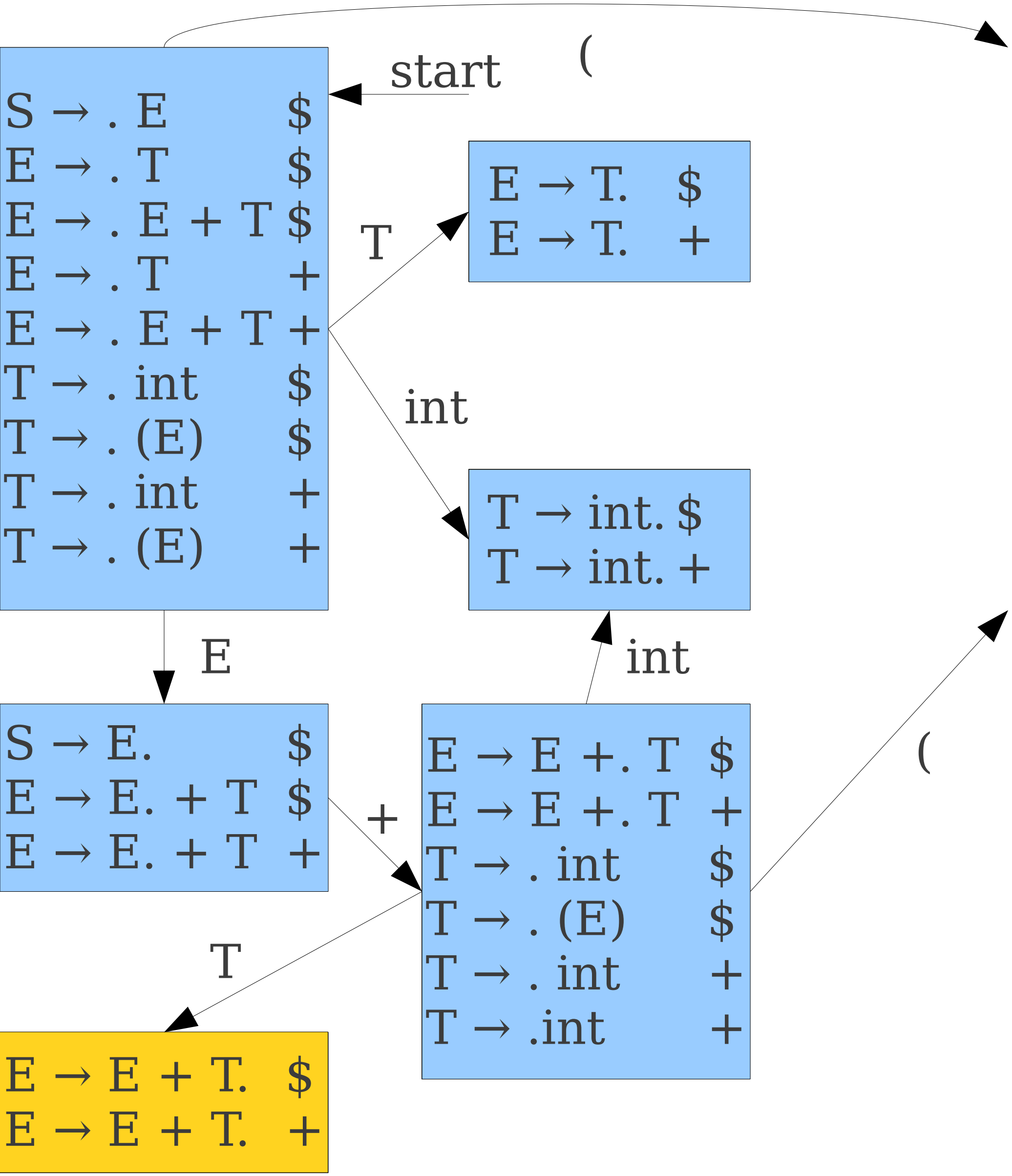
T → int. $
T → int. +

S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +

Deterministic LR(1) Automata



start

(

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

E

S → E. $
E → E. + T $
E → E. + T +

+

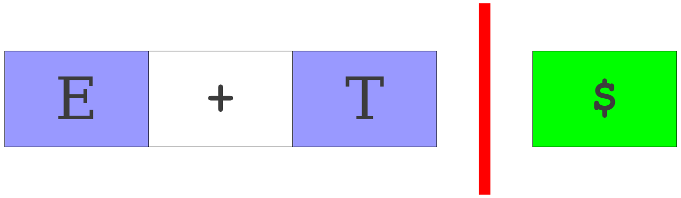
int

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

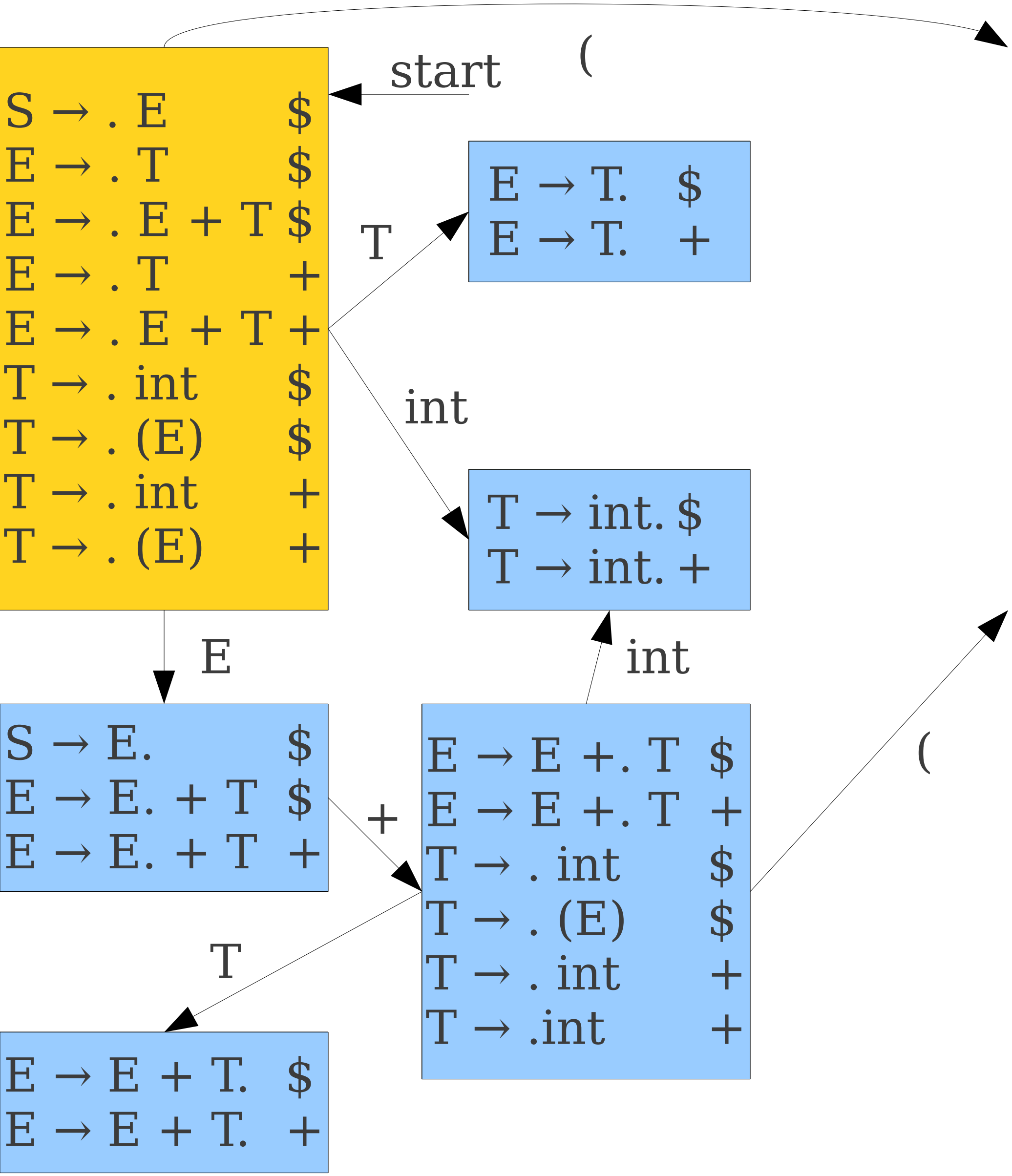
(

T

E → E + T. $
E → E + T. +



Deterministic LR(1) Automata



start

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

(

T

E → T. $
E → T. +

int

T → int. $
T → int. +

E

int

S → E. $
E → E. + T $
E → E. + T +

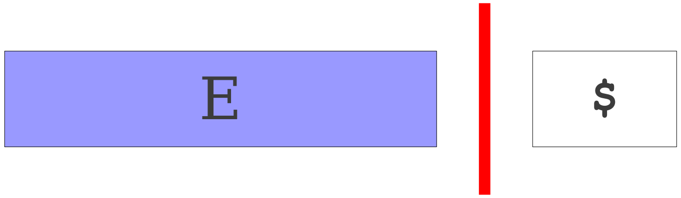
+

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

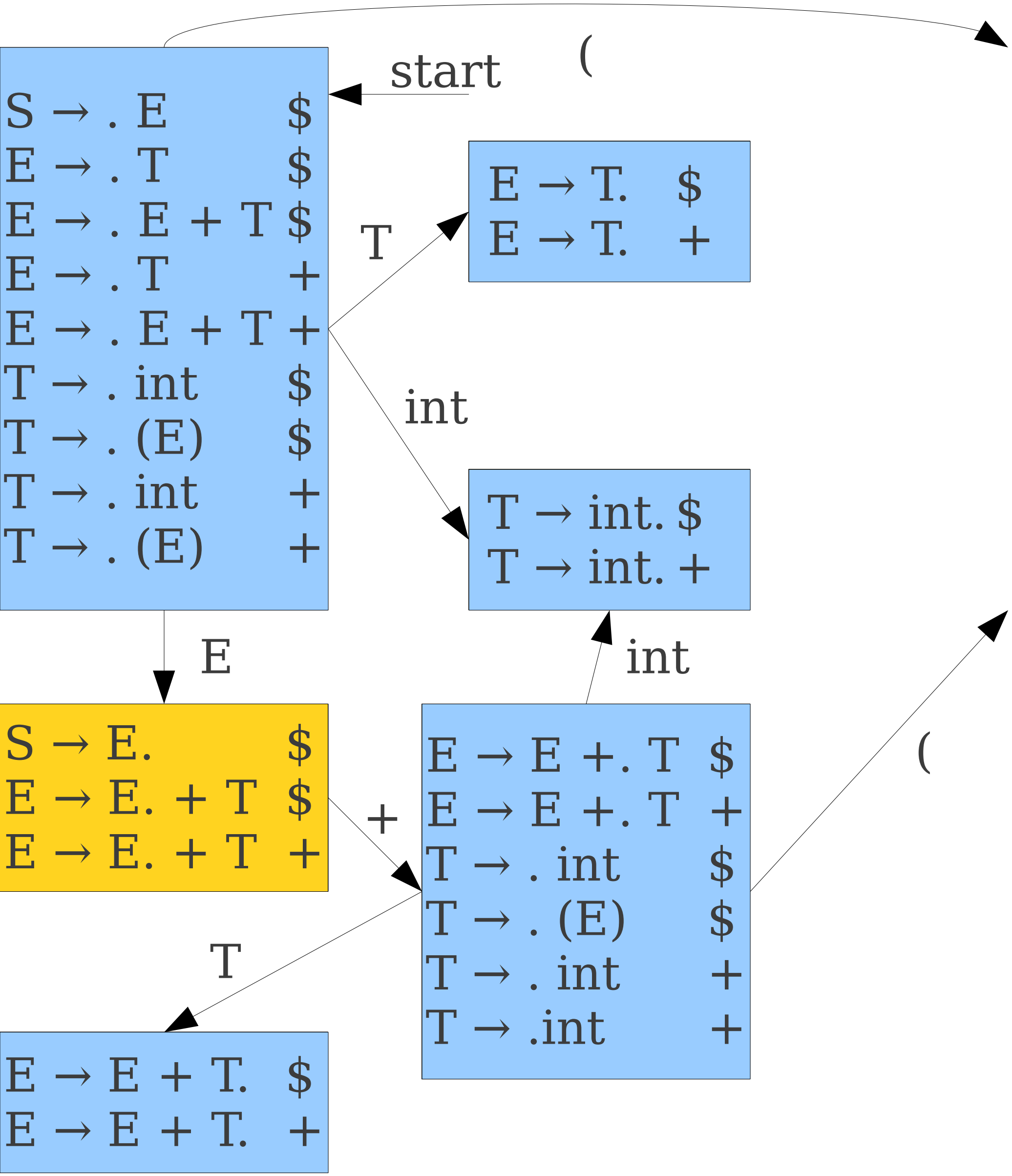
(

T

E → E + T. $
E → E + T. +



Deterministic LR(1) Automata



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

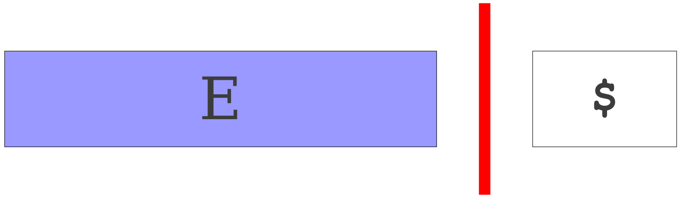
E → T. $
E → T. +

T → int. $
T → int. +

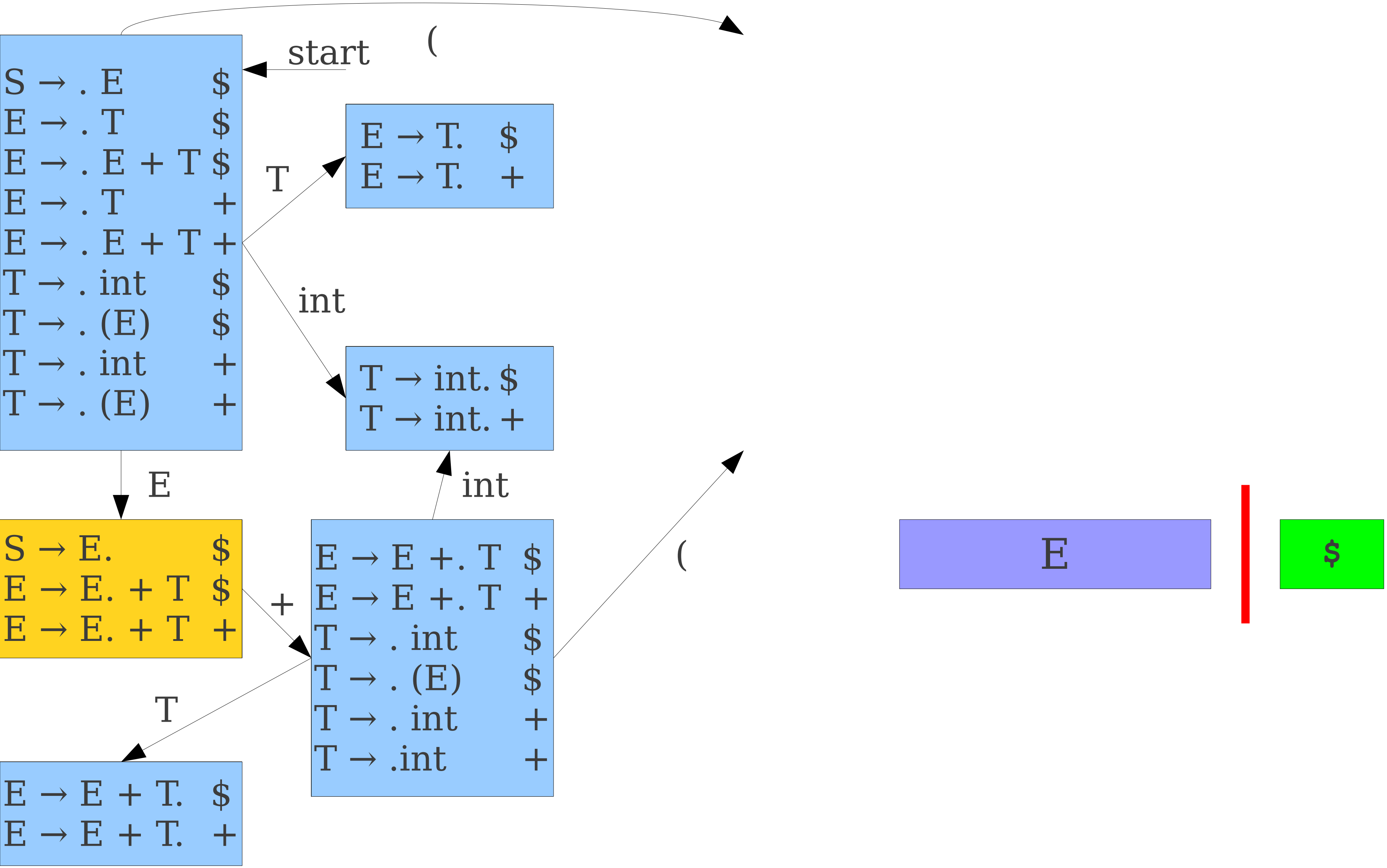
S → E. $
E → E. + T $
E → E. + T +

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

E → E + T. $
E → E + T. +



Deterministic LR(1) Automata



start

(

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

T

E → T. $
E → T. +

int

T → int. $
T → int. +

E

int

S → E. $
E → E. + T $
E → E. + T +

+

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

(

T

E → E + T. $
E → E + T. +

E

$

Constructing LR(1) Automata II

- Begin in a state containing **S** → · **E** [`$`], where **S** is the start symbol.
- Compute the **closure** of the state:
  - If **A** → α · **B**ω [`t`] is in the state, add **B** → · γ [`t`] to the state for each production **B** → γ and for each terminal `t` ∈ FIRST*(ω`t`)
- Repeat until no new states are added:
  - If a state contains a production **A** → α · $x\omega$ [`t`], add a transition on x from that state to the state containing the closure of **A** → αx · ω [`t`].

Structure of LR(1) Automata

- Every LR(1) automaton simulates two processes simultaneously:
    - An **LR(0) automaton** for finding handles.
    - A **lookahead tracker** for determining what the lookahead is.
- Removing the lookaheads from an LR(1) automaton results in a (much larger) LR(0) automaton for the same grammar.

Representing LR(1) Automata

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

The LR(1) Parsing Algorithm

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

Constructing LR(1) Parse Tables

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

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

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

The Power of LR(1)

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

LR(1) Automata are **Huge**

- In a grammar with *n* terminals, could in theory be $O(2^n)$ times as large as the LR(0) automaton.
  - Replicate each state with all $O(2^n)$ possible lookaheads.
- LR(1) tables for practical programming languages can have hundreds of thousands or even *millions* of states.
- Consequently, LR(1) parsers are rarely used in practice.

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

Why is LR(1) so powerful?

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

Revisiting Shift/Reduce Conflicts

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

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

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

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

SLR(1)

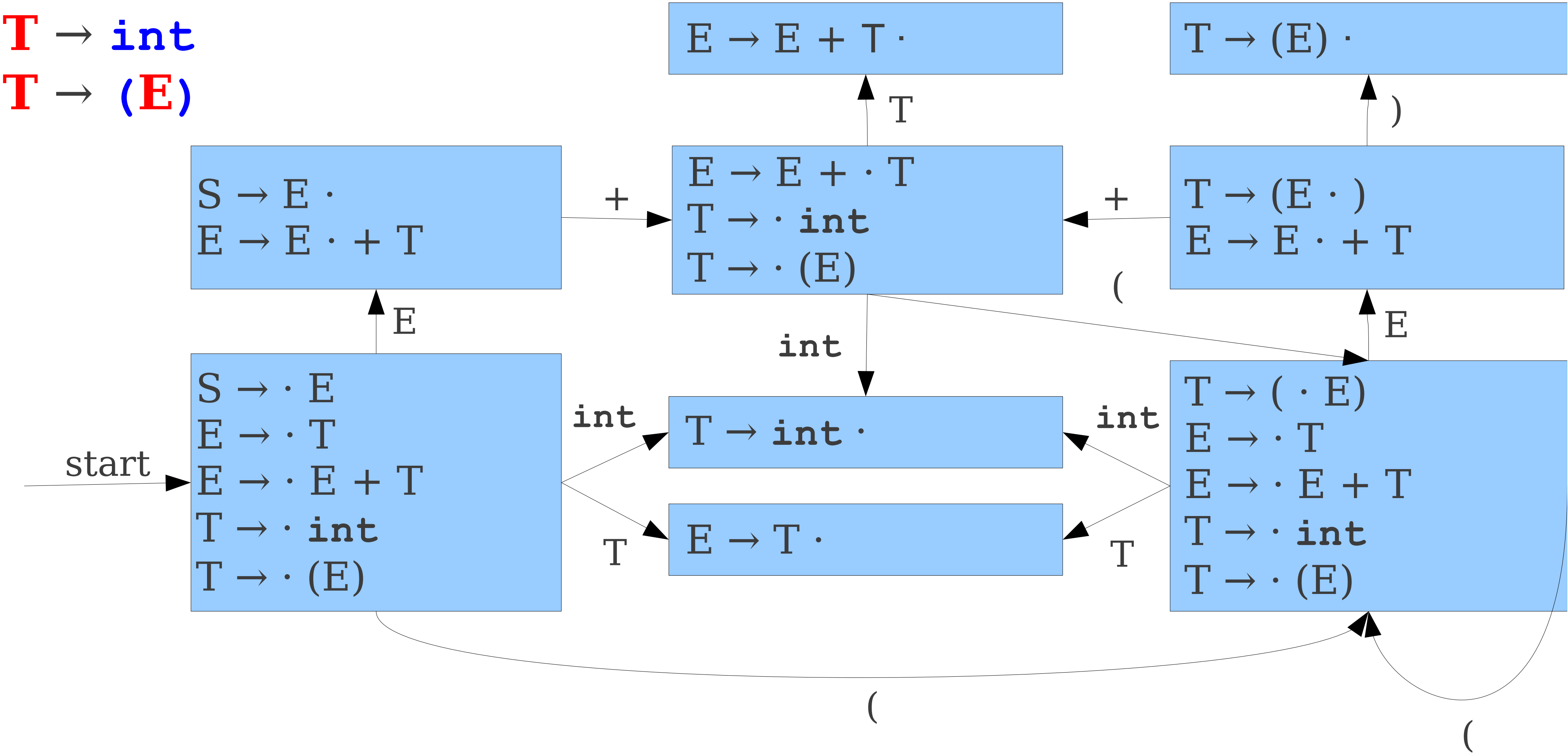
- **Simple LR(1)**
- Minor modification to LR(0) automaton that uses lookahead to avoid shift/reduce conflicts.
- Idea: Only reduce **A** → ***ω*** if the next token **`t`** is in FOLLOW(**A**).
- Automaton identical to LR(0) automaton; only change is when we choose to reduce.

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

SLR(1) Parsing

SLR(1) Parsing

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



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 ·

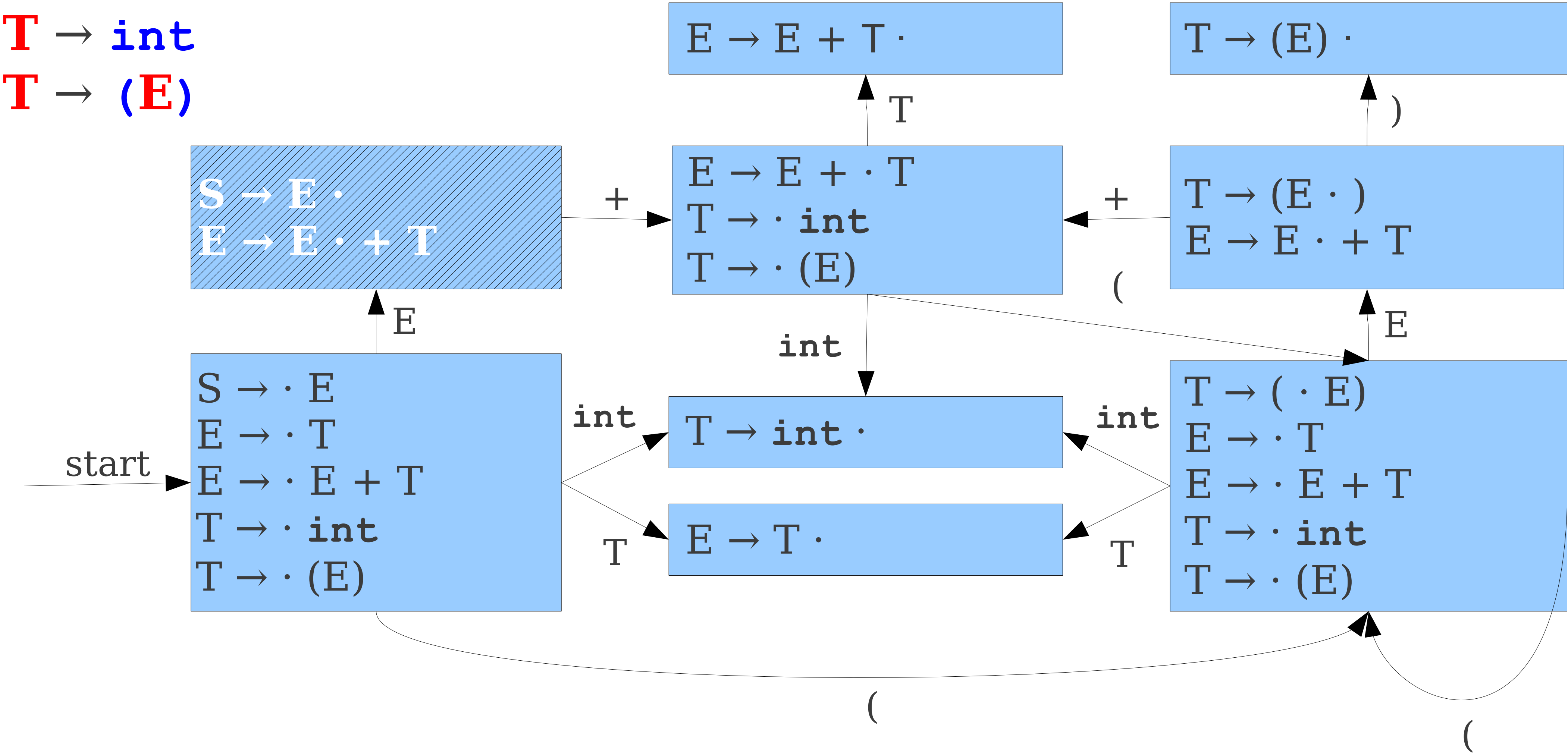
T → (· E)
E → · T
E → · E + T
T → · `int`
T → · (E)

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)

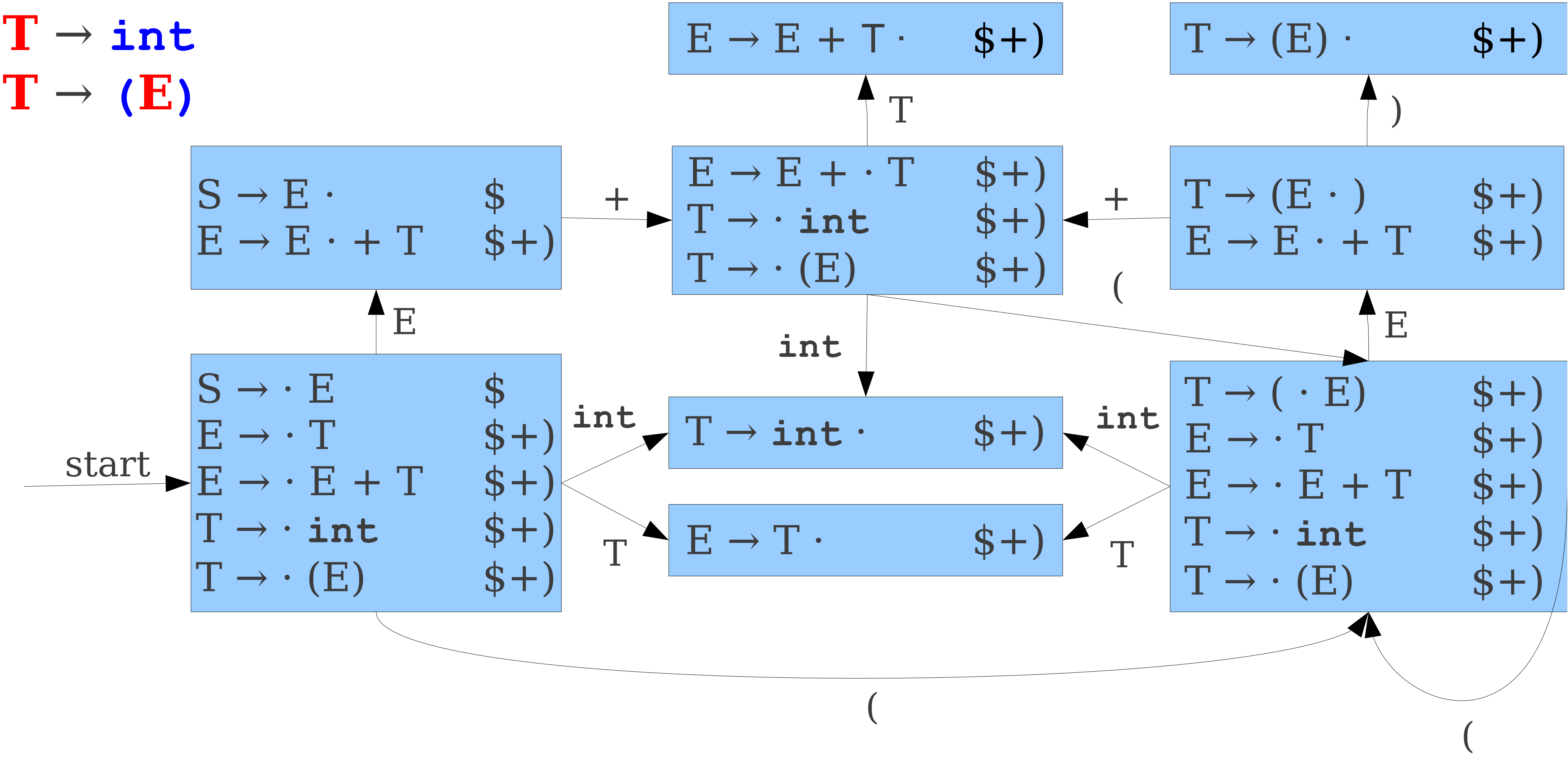
T → **`int`** ·

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) $+)

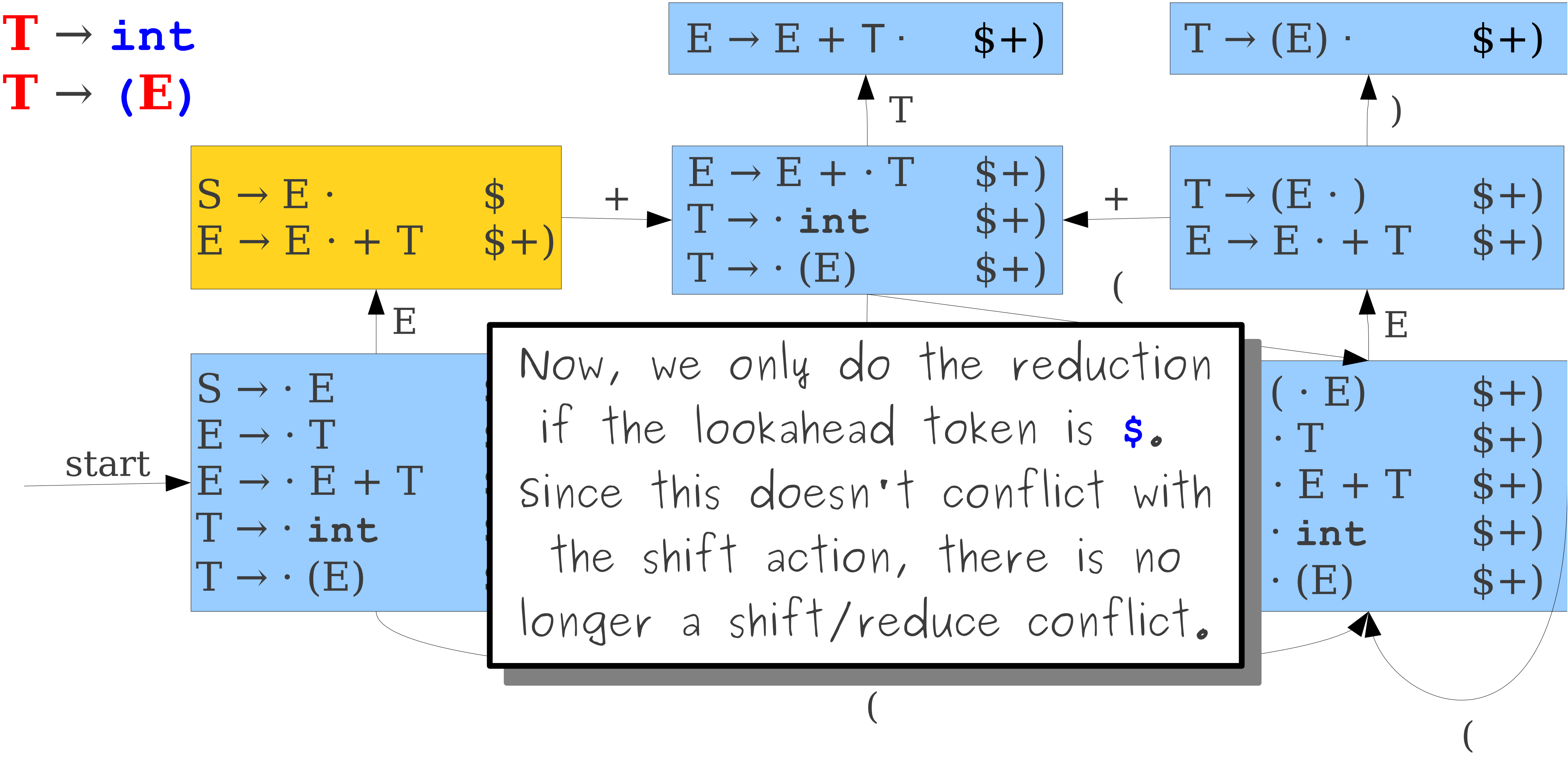
T → **`int`** · $+)

E → T · $+)

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

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

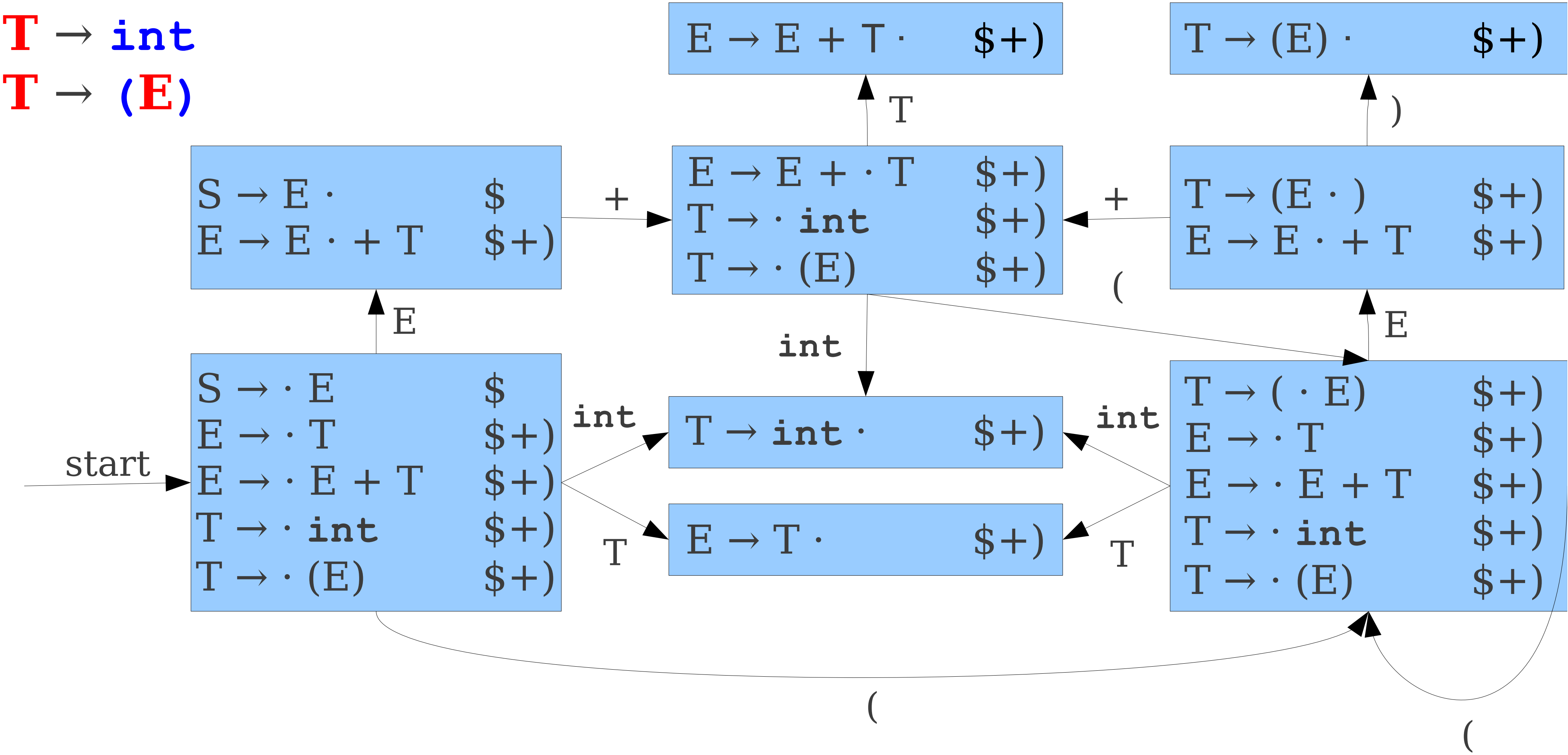
SLR(1) Parsing



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

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

SLR(1) Parsing



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` · $+)

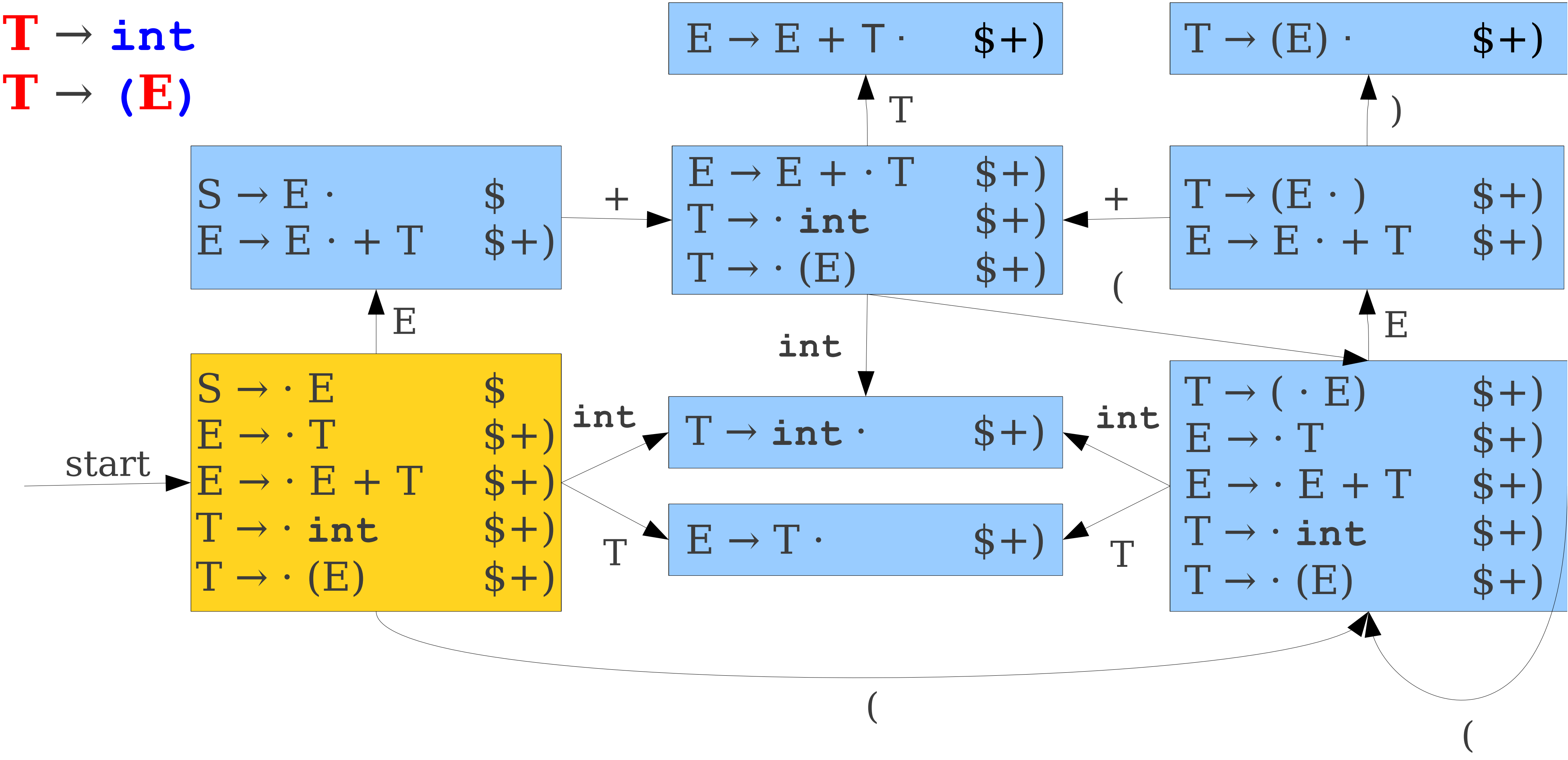
E → T · $+)

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)**

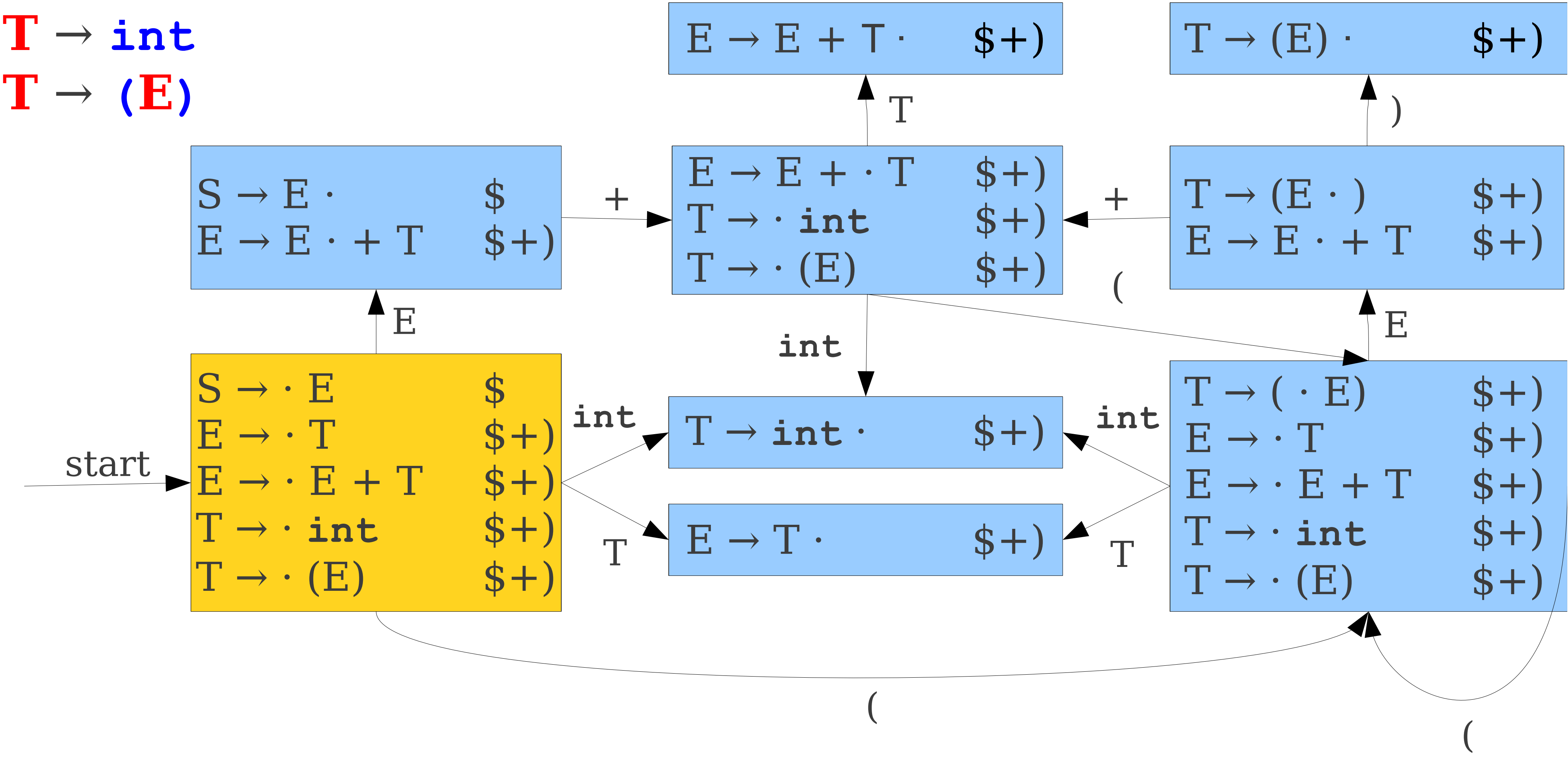
SLR(1) Parsing



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

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

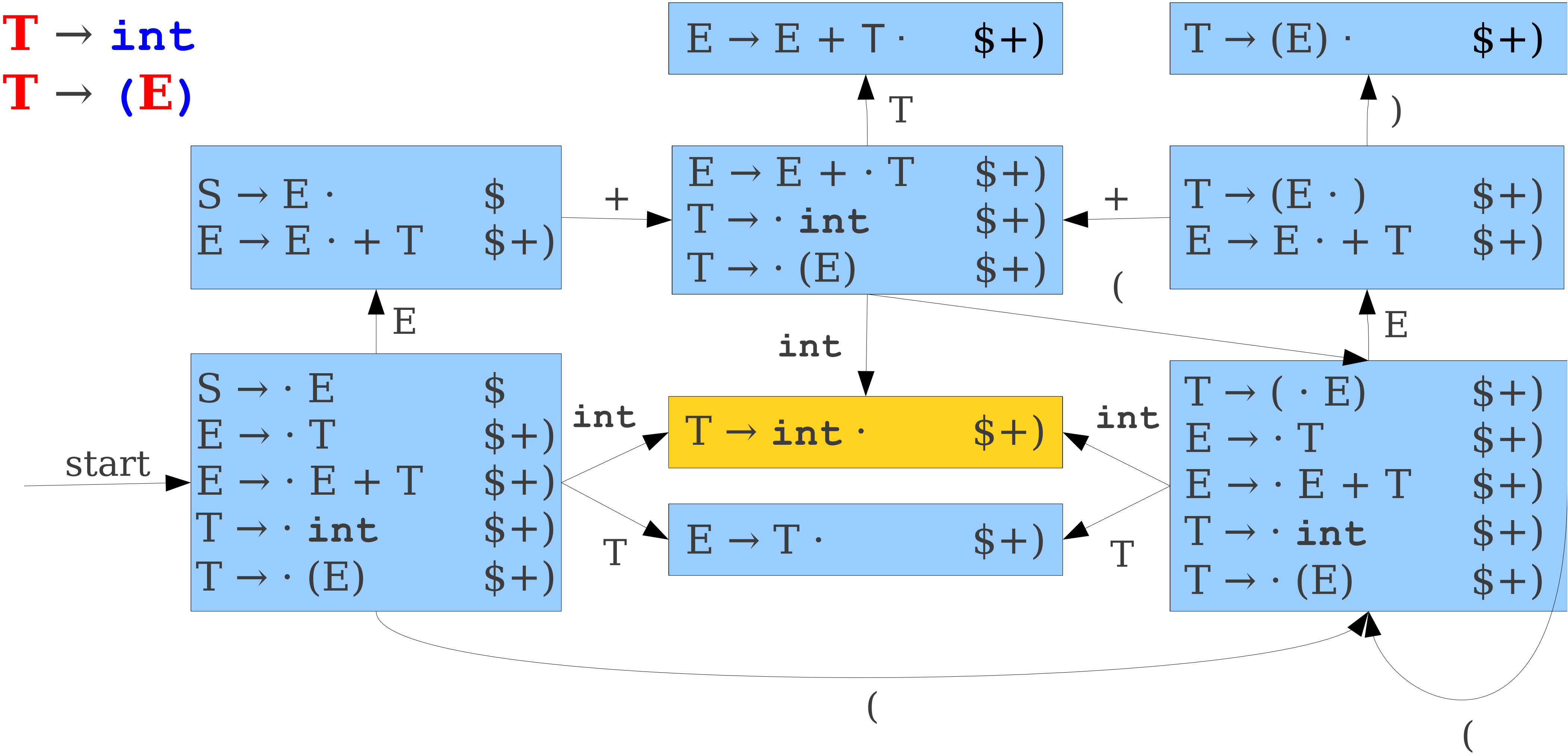
SLR(1) Parsing



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) $+)

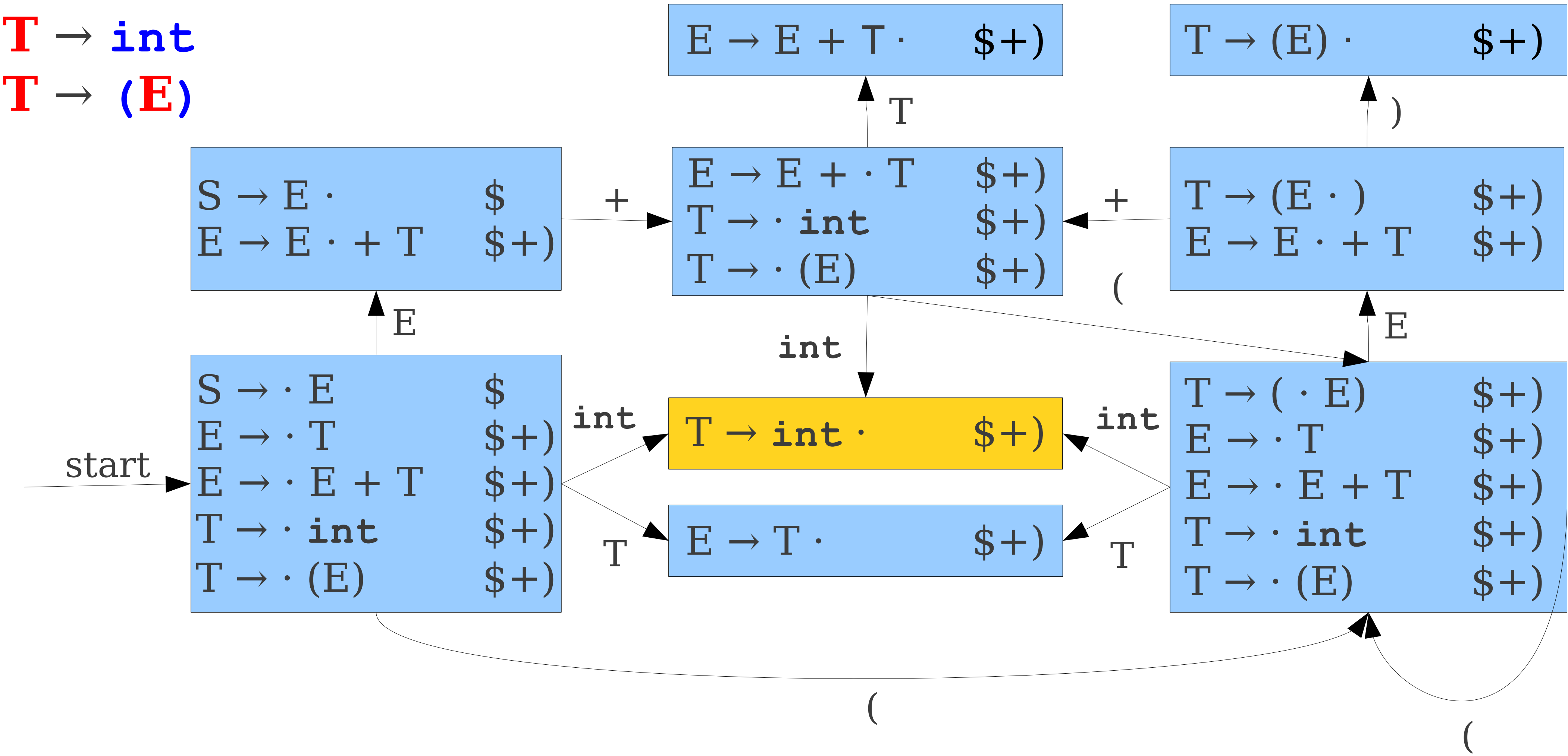
T → (E ·) $+)
E → E · + T $+)

T → (E) · $+)

int	|	**+**	**(**	**int**	**+**	**int**	**+**	**int**	**)**	**$**

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

SLR(1) Parsing


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) $+)

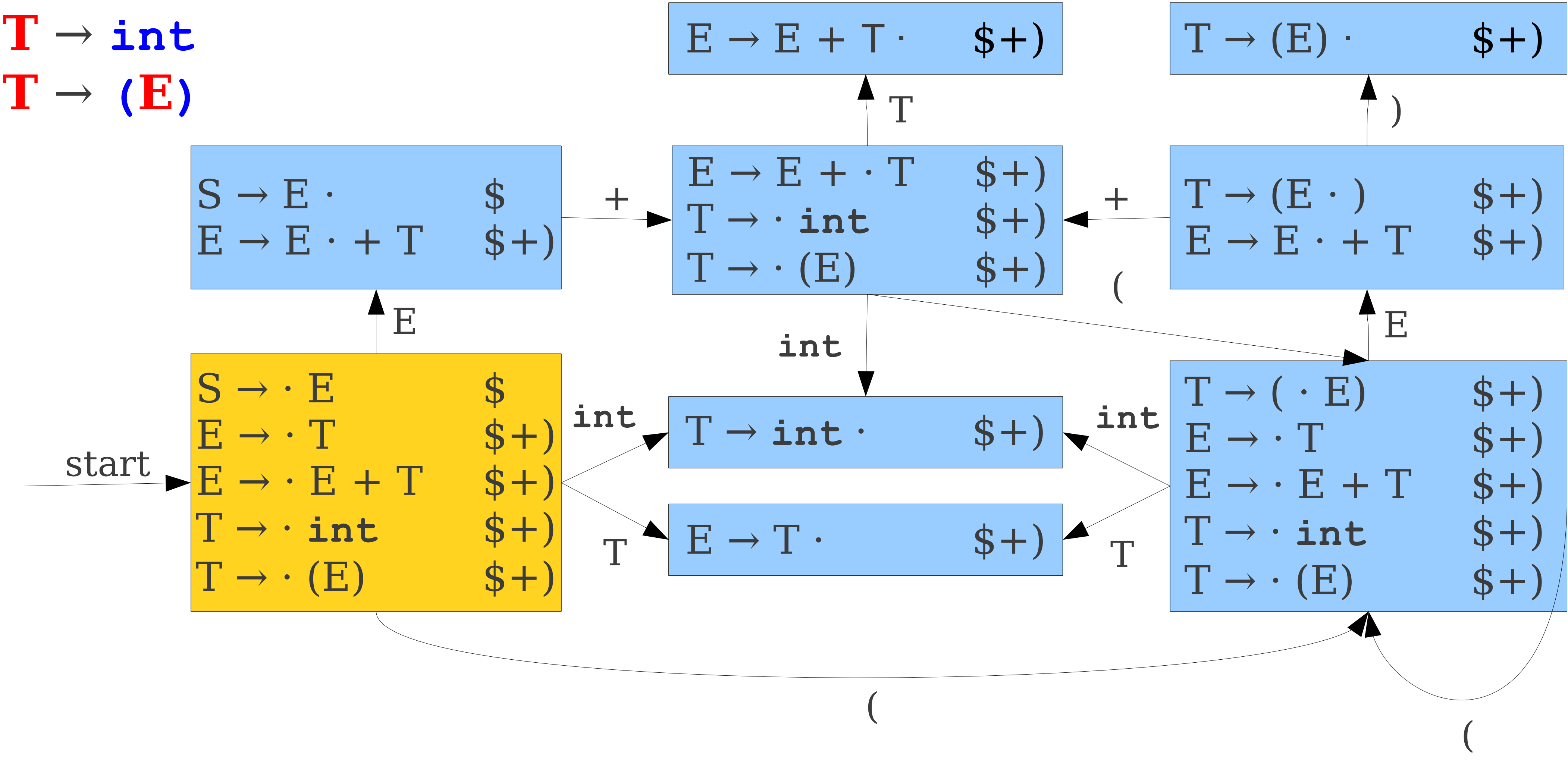
T → (E ·) $+)
E → E · + T $+)

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` · $+)

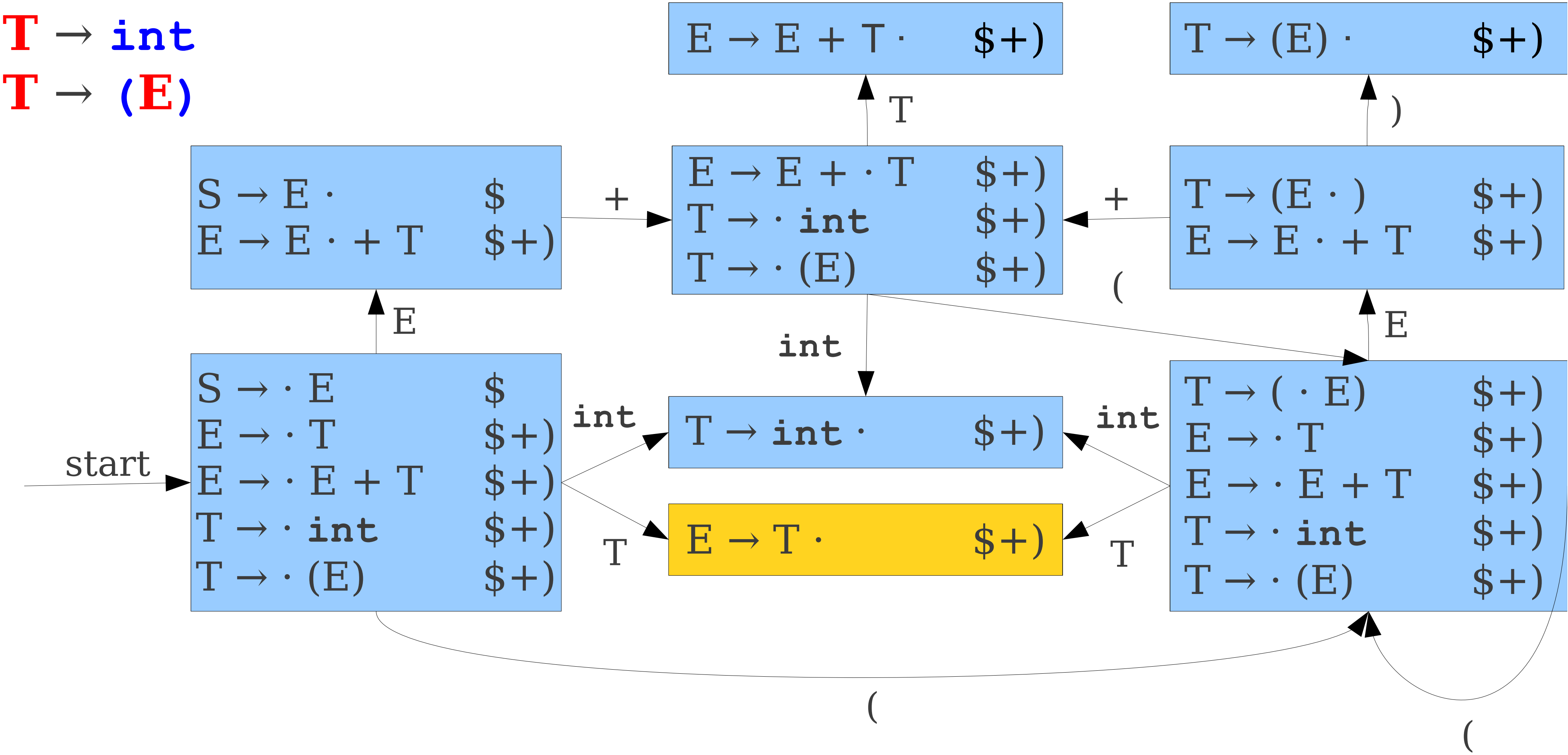
E → T · $+)

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

T		+	(	int	+	int	+	int	)	$

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

SLR(1) Parsing



E → E + T · $+)

T → (E) · $+)

S → E · $
E → E · + T $+)

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

T → (E ·) $+)
E → E · + T $+)

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

T → `int` · $+)

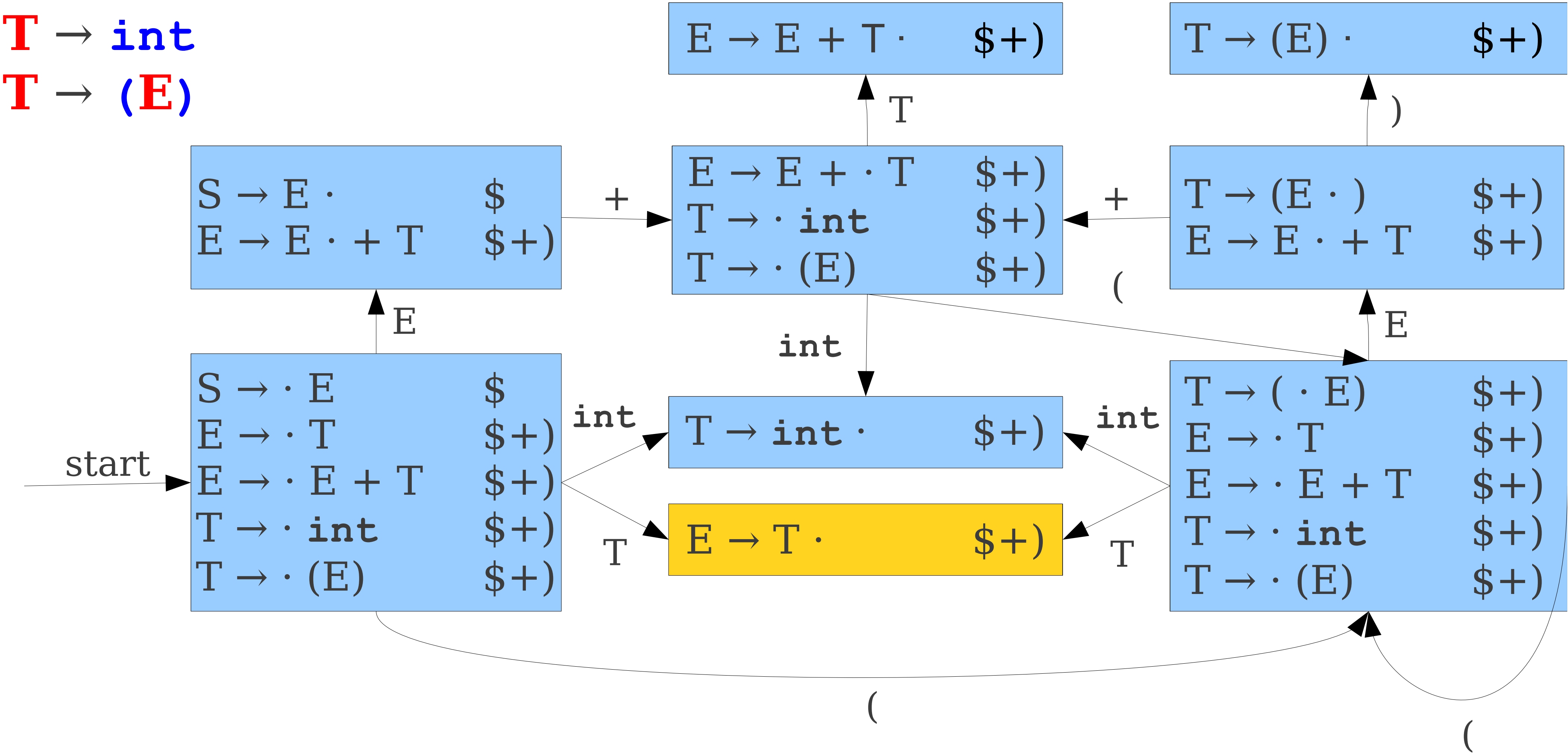
E → T · $+)

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

T	+	(	int	+	int	+	int	)	$

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

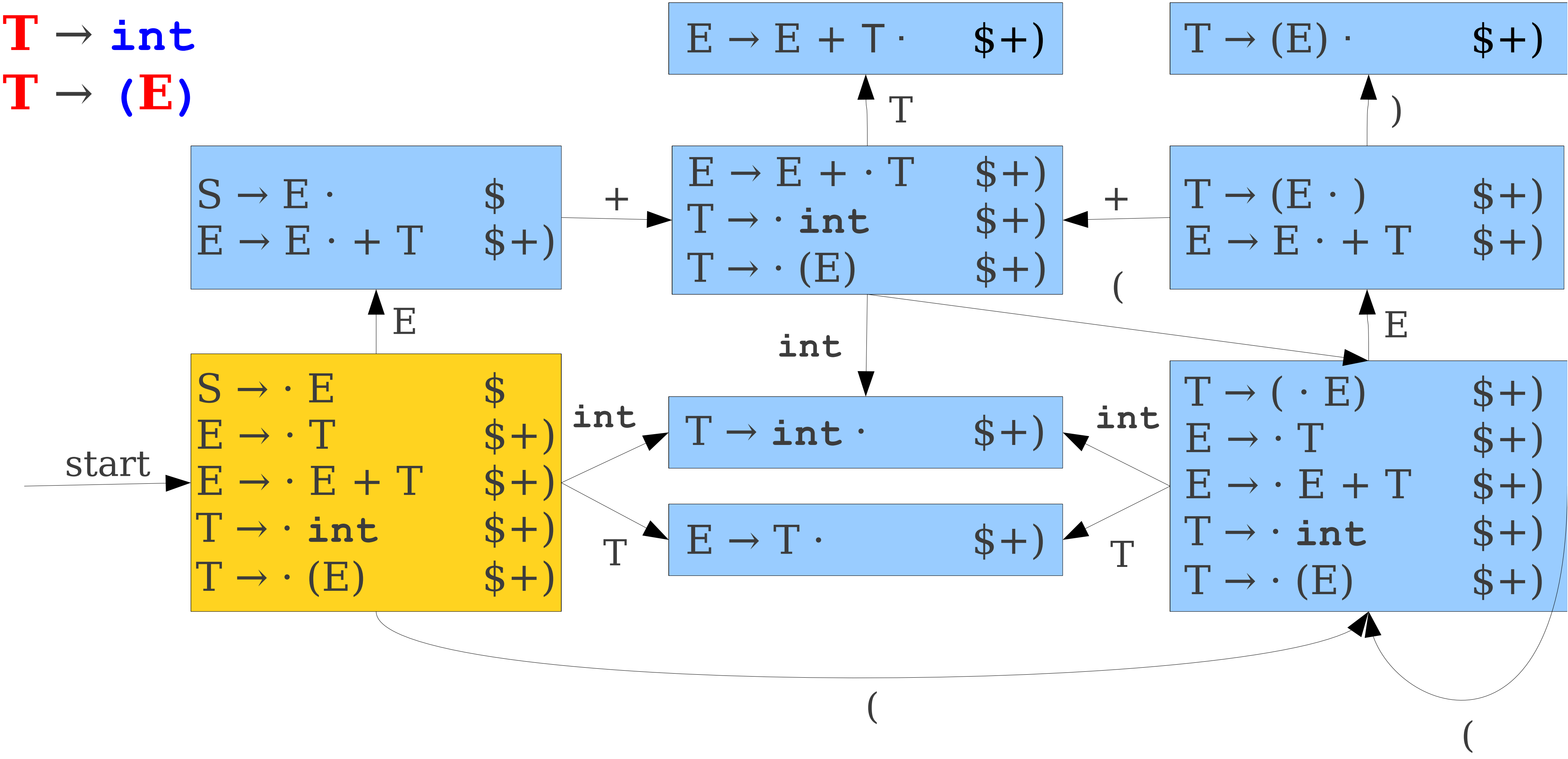
SLR(1) Parsing



T		+	(	int	+	int	+	int	)	$

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

SLR(1) Parsing



E → E + T · $+)

T → (E) · $+)

S → E · $
E → E · + T $+)

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

T → (E ·) $+)
E → E · + T $+)

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

T → **int** · $+)

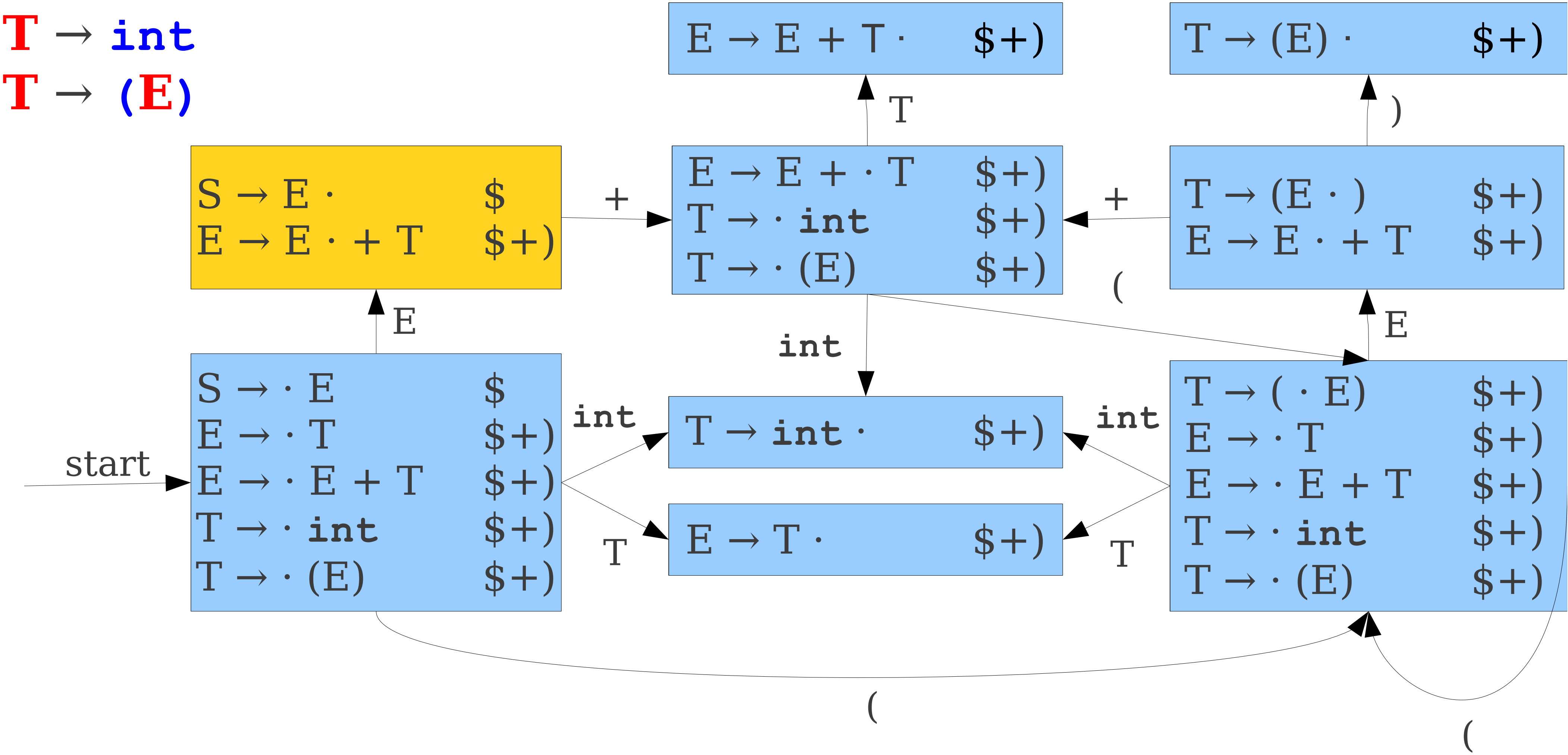
E → T · $+)

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

E		+	(	int	+	int	+	int	)	$

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

SLR(1) Parsing



E → E + T · $+)

T → (E) · $+)

S → E · $
E → E · + T $+)

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

T → (E ·) $+)
E → E · + T $+)

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

T → **`int`** · $+)

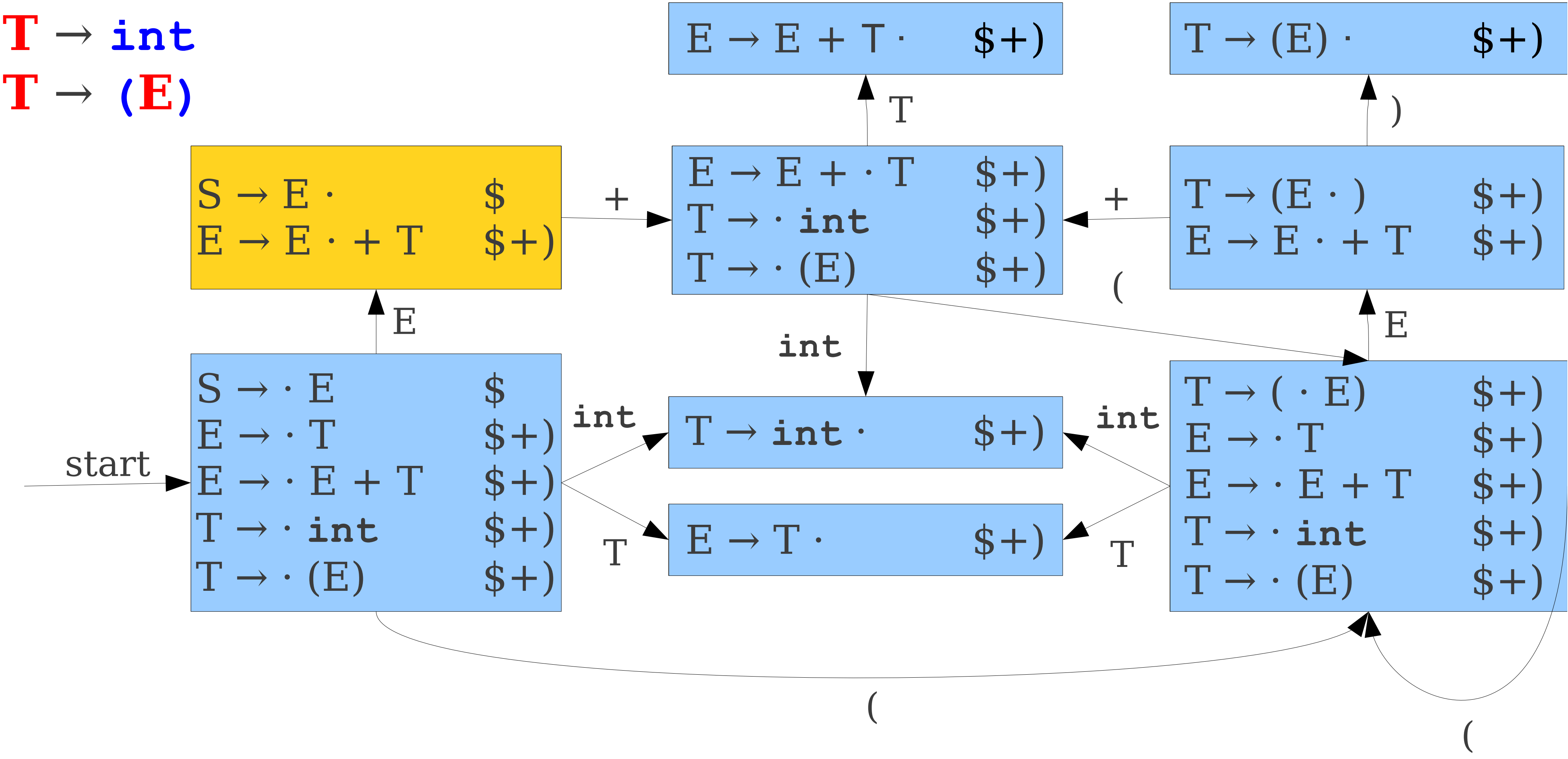
E → T · $+)

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

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

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

SLR(1) Parsing



E → 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 · $+)

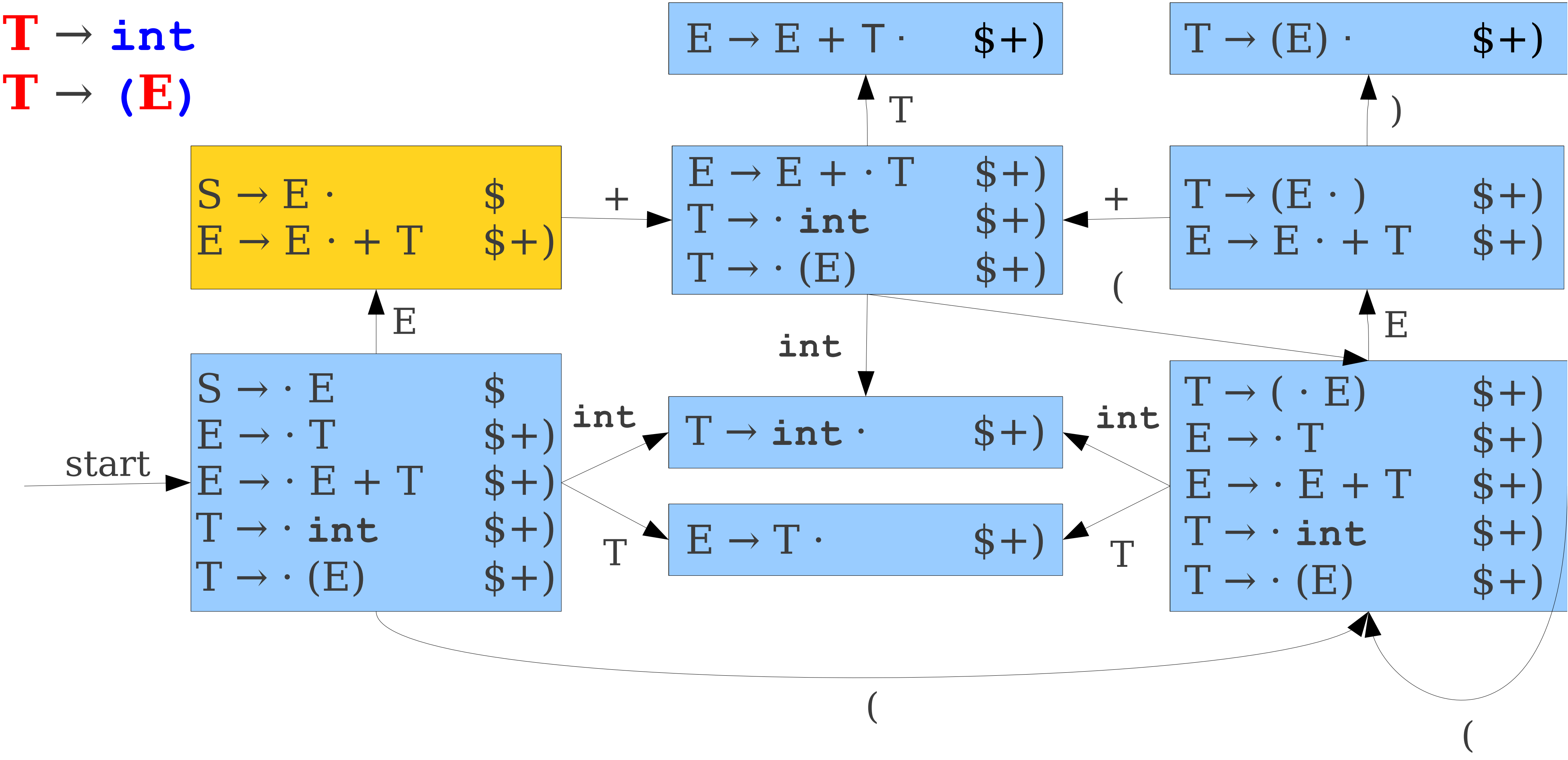
E → T · $+)

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 → · 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 $+)

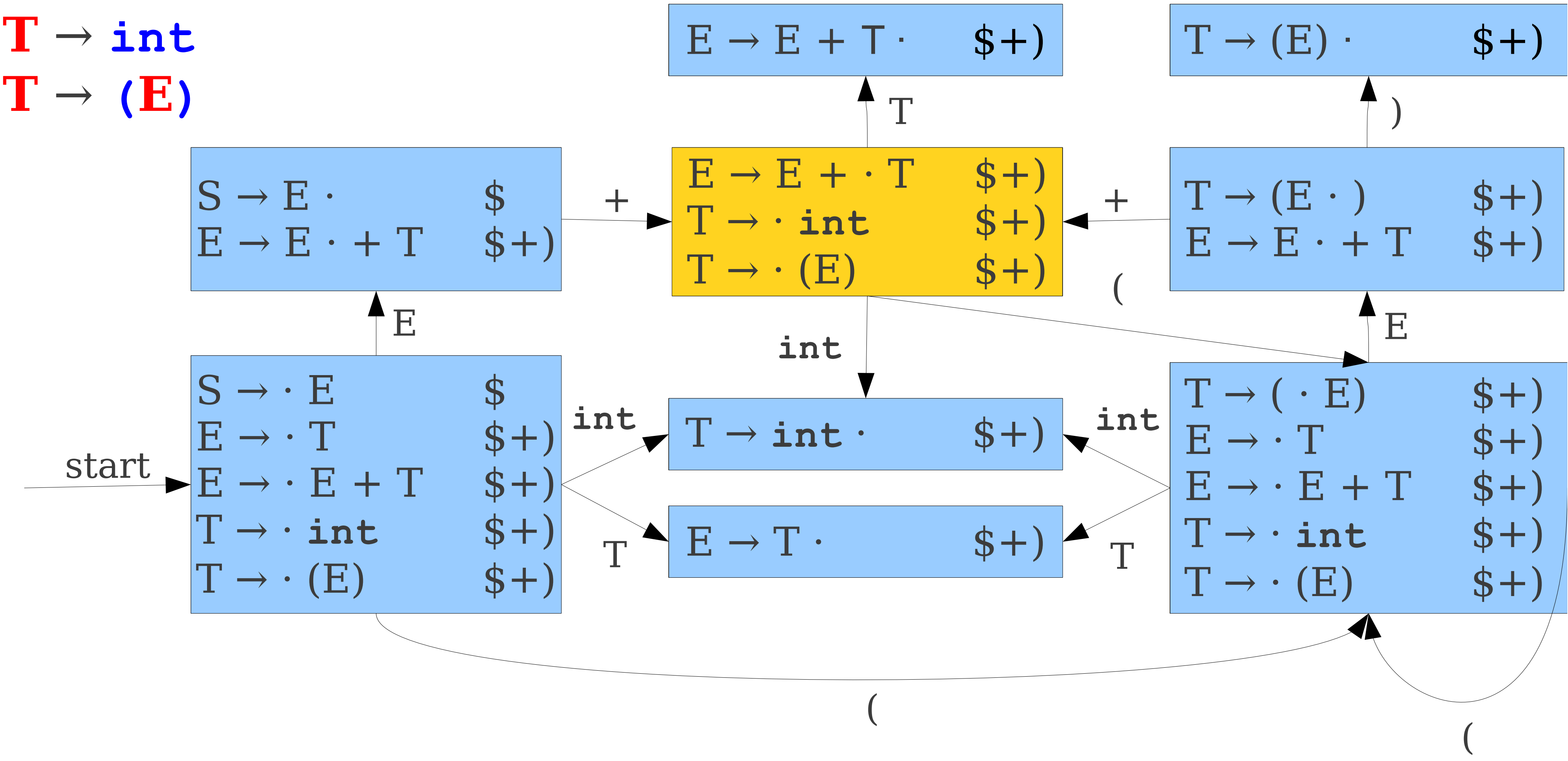
T → (E) · $+)

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** ·　　$+)

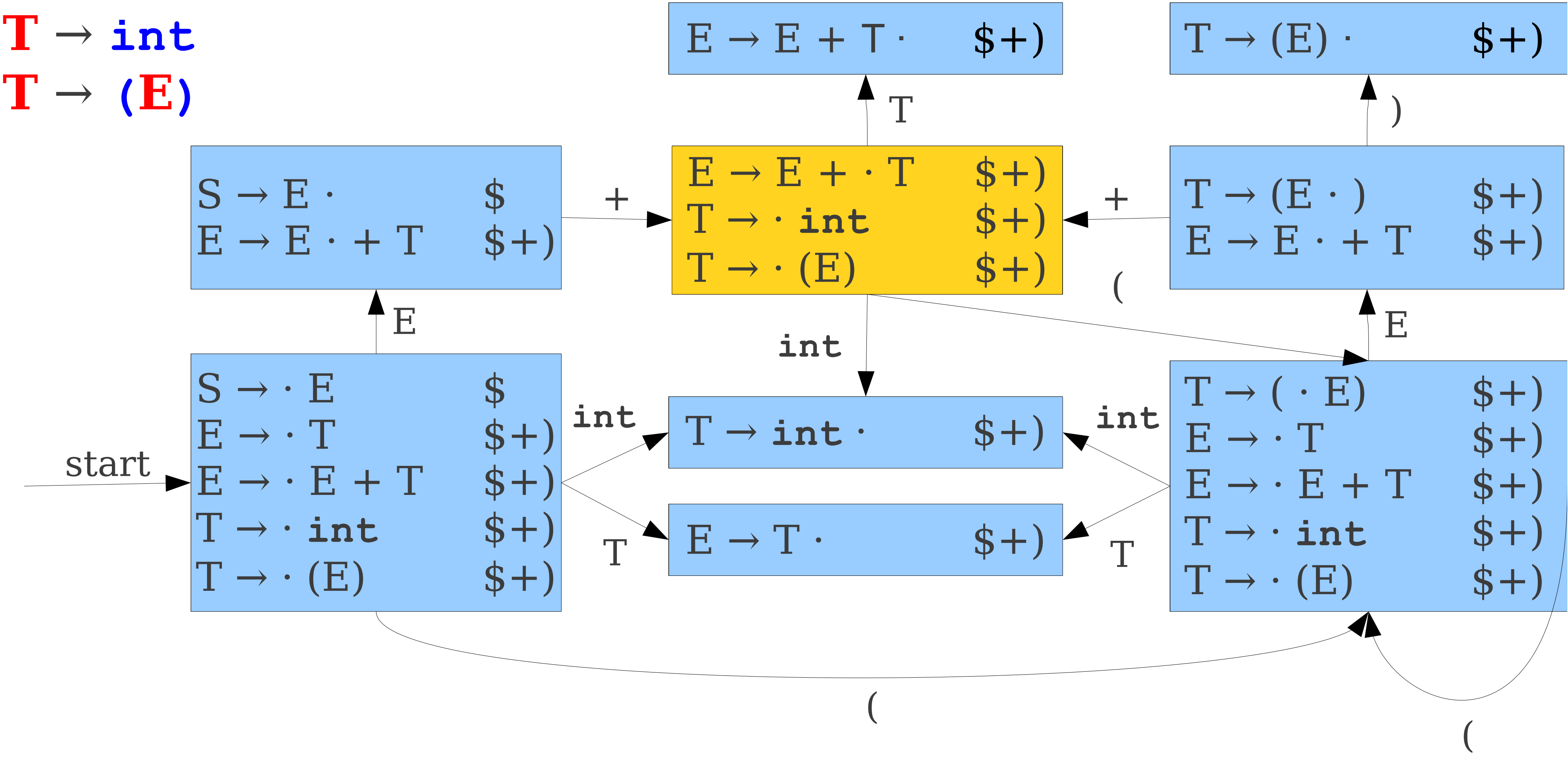
E → T ·　　$+)

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 · $+)

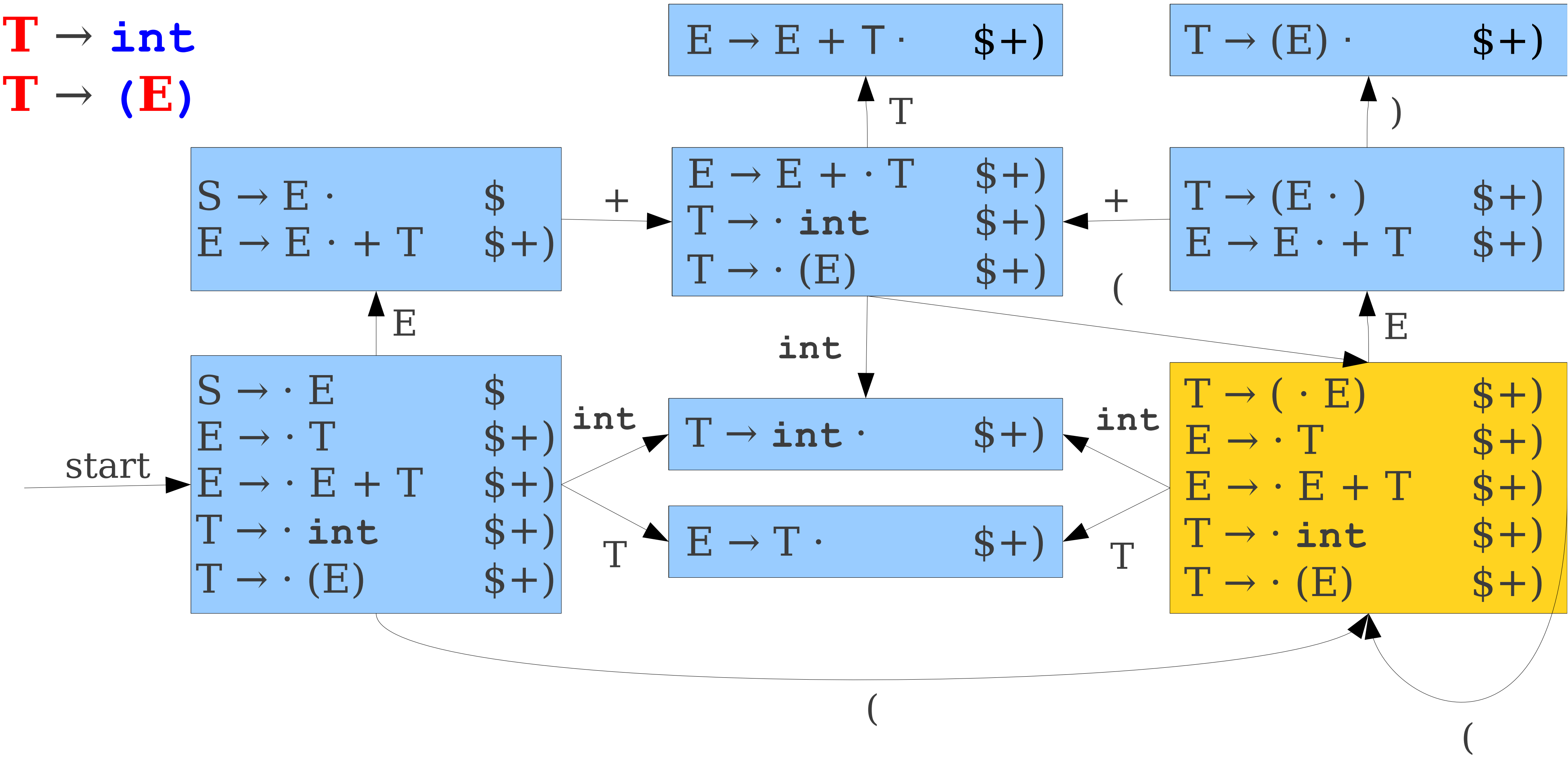
T → (· E) $+)
E → · T $+)
E → · E + T $+)
T → · **int** $+)
T → · (E) $+)

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`	)	$

Analysis of SLR(1)

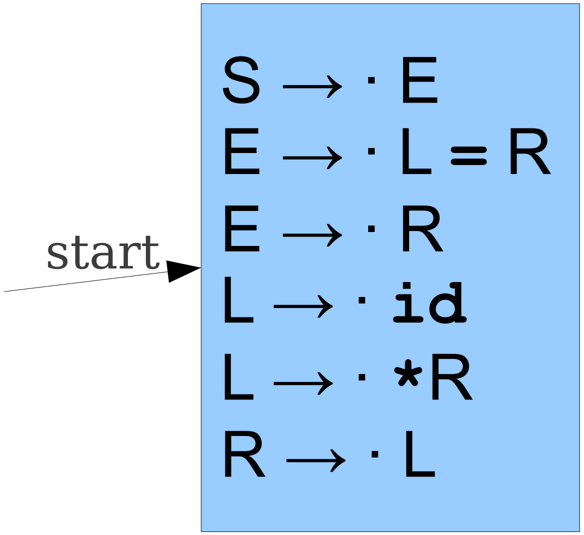
- Exploits lookahead in a small space.
  - Small automaton – same number of states as in as LR(0).
  - Works on many more grammars than LR(0)
- Too weak for most grammars: lose context from not having extra states.

The Limits of SLR(1)

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

The Limits of SLR(1)

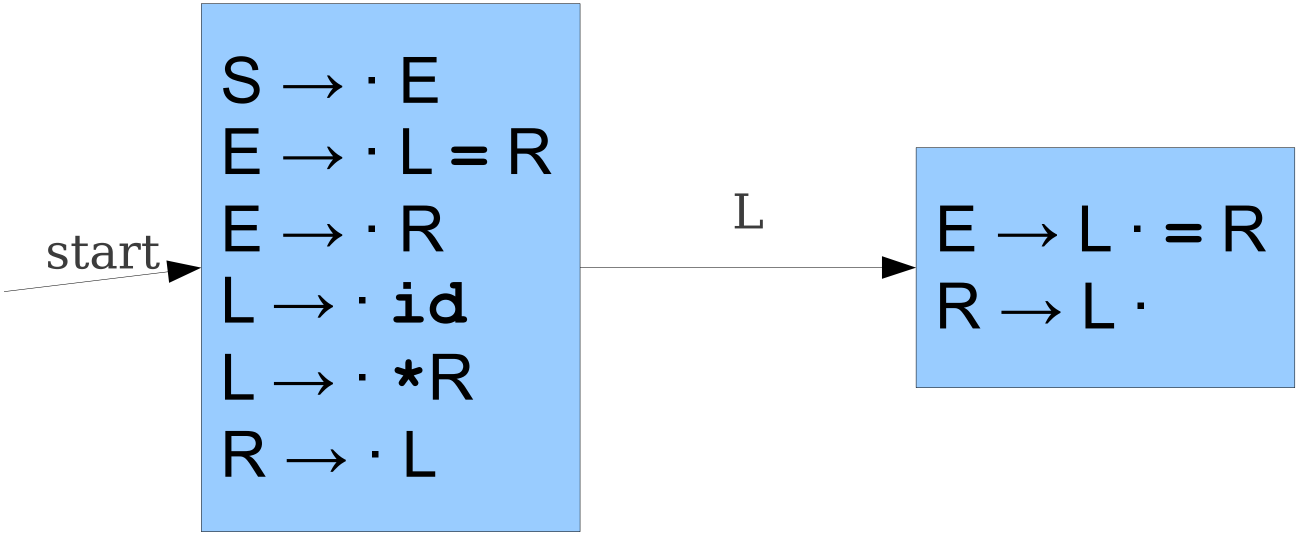
S → E
E → L = R
E → R
L → id
L → *R
R → L



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

The Limits of SLR(1)

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



start

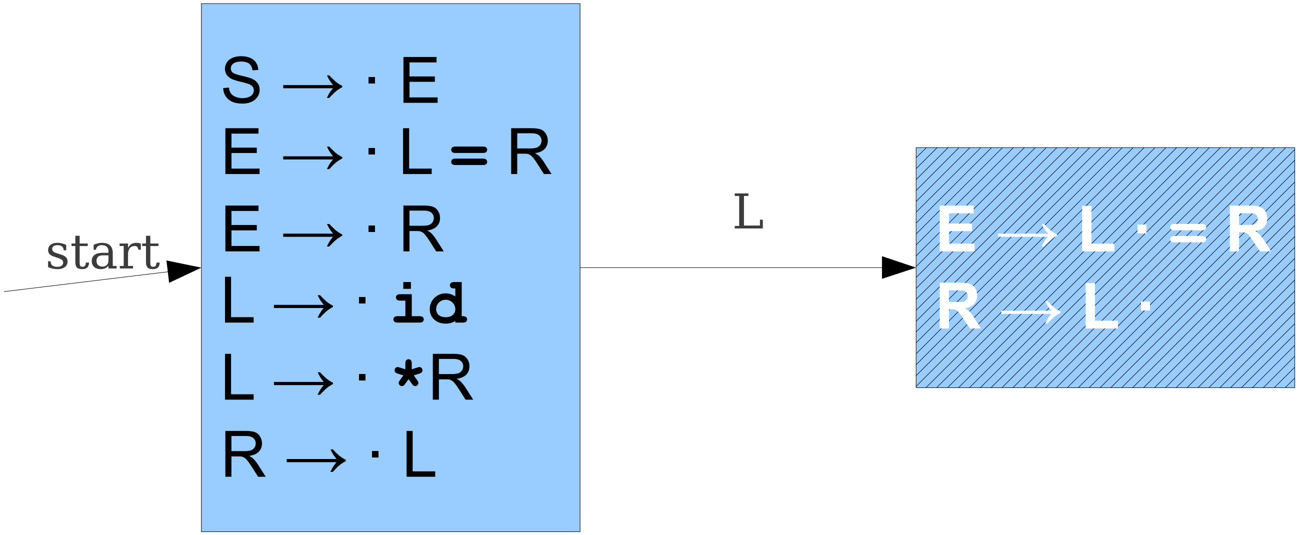
S → · E
E → · L = R
E → · R
L → · **id**
L → · ***R**
R → · L

L

E → L · = R
R → L ·

The Limits of SLR(1)

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



start

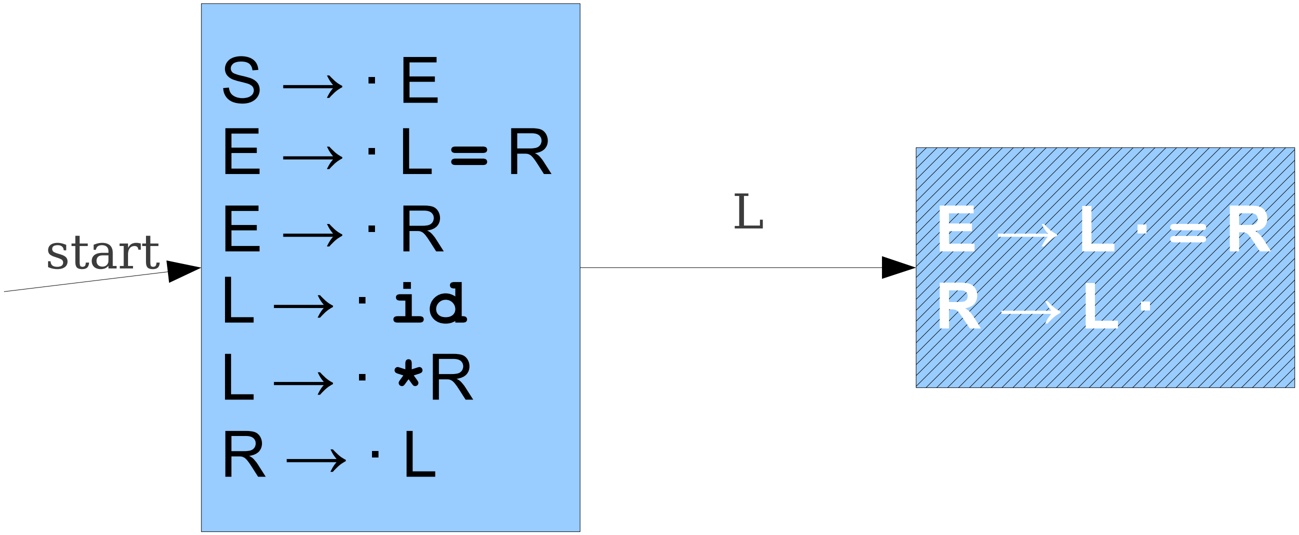
S → · E
E → · L = R
E → · R
L → · id
L → · *R
R → · L

L

E → L · = R
R → L ·

The Limits of SLR(1)

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



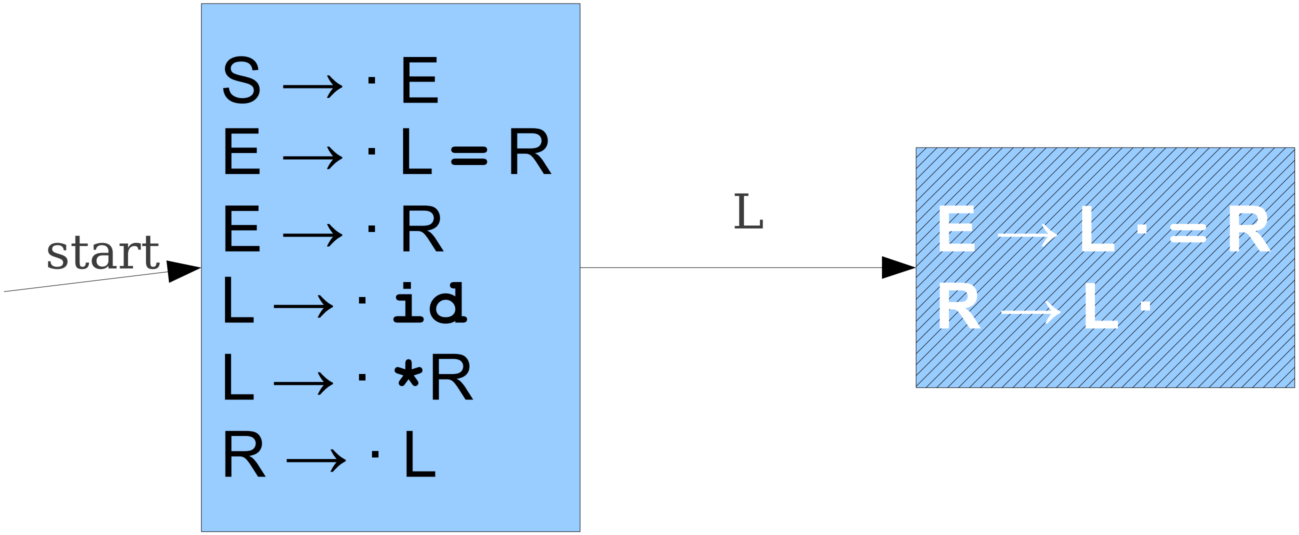
S → · E
E → · L = R
E → · R
L → · id
L → · *R
R → · L

E → L · = R
R → L ·

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

The Limits of SLR(1)

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



start

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

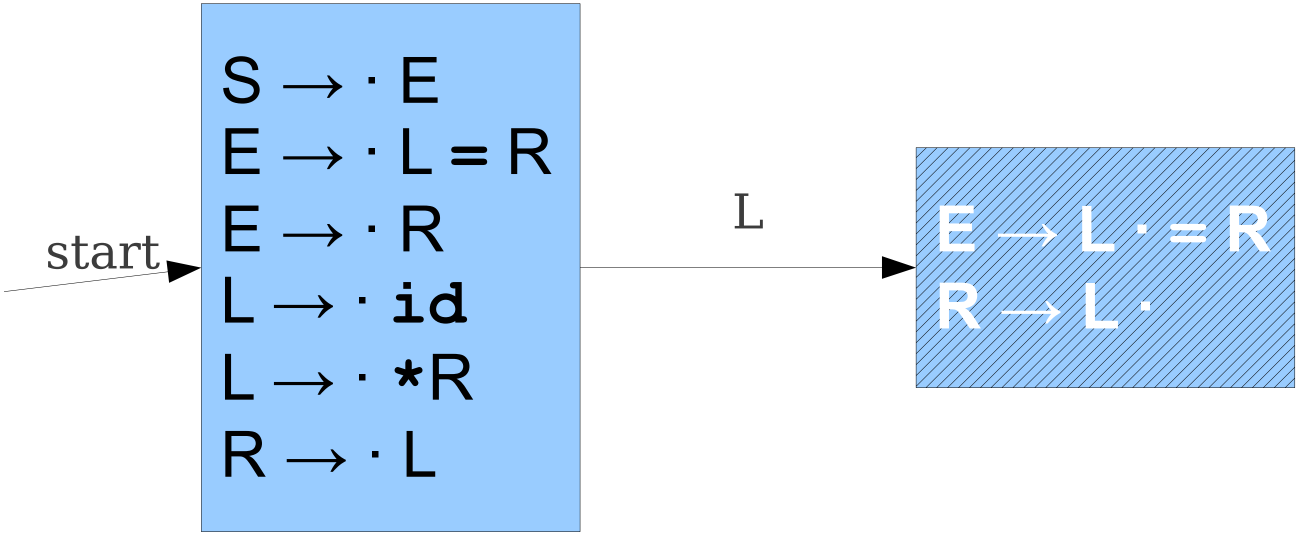
L

E → L · = R
R → L ·

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

The Limits of SLR(1)

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



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

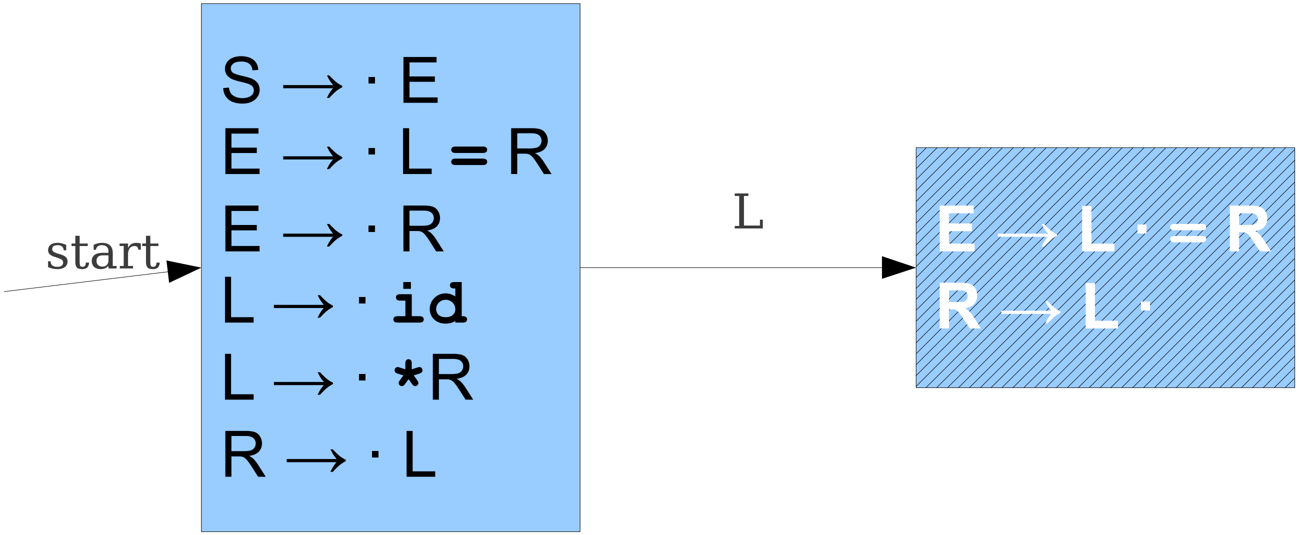
E → L · = R
R → L ·

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

= ∈ FOLLOW(**R**).

The Limits of SLR(1)

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



start

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

L

E → L · = R
R → L ·

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

= ∈ FOLLOW(**R**).

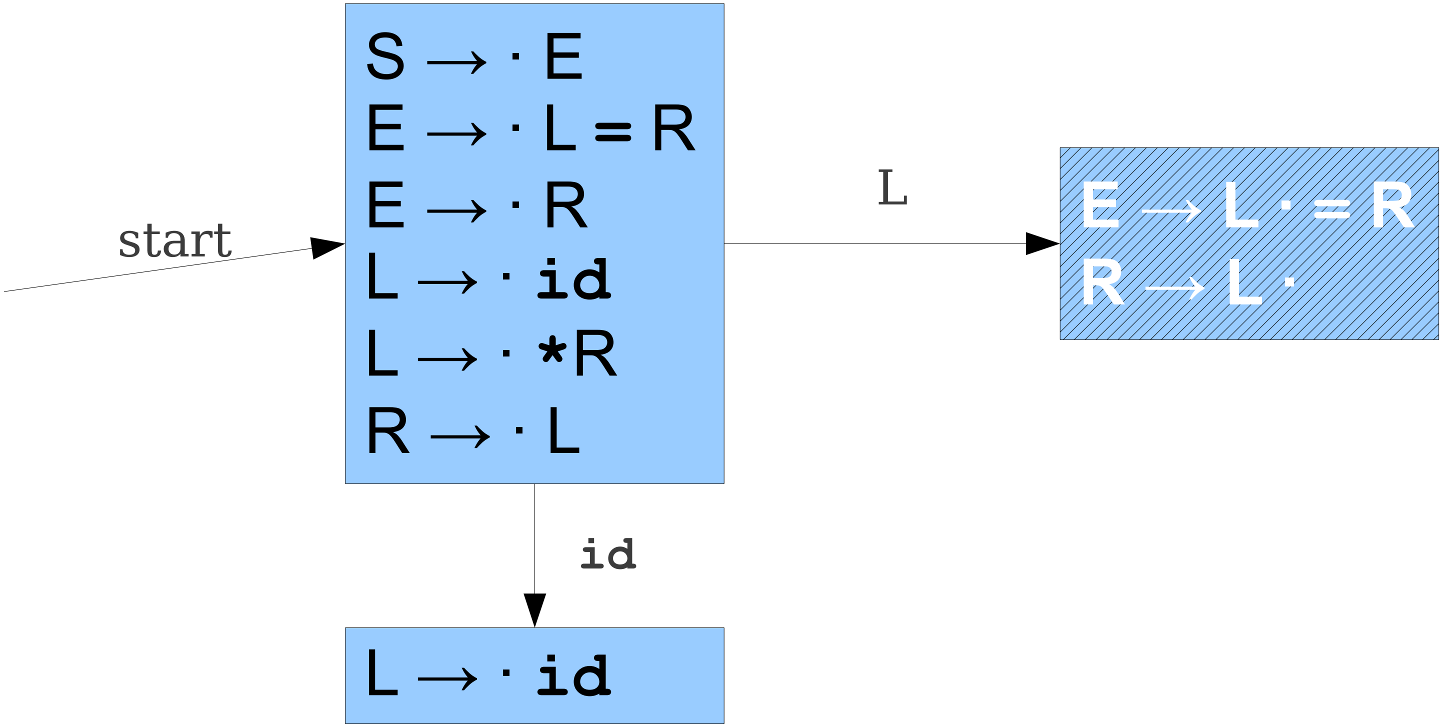
We have a conflict!

Why is SLR(1) Weak?

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

A Lack of Context

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



start

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

L

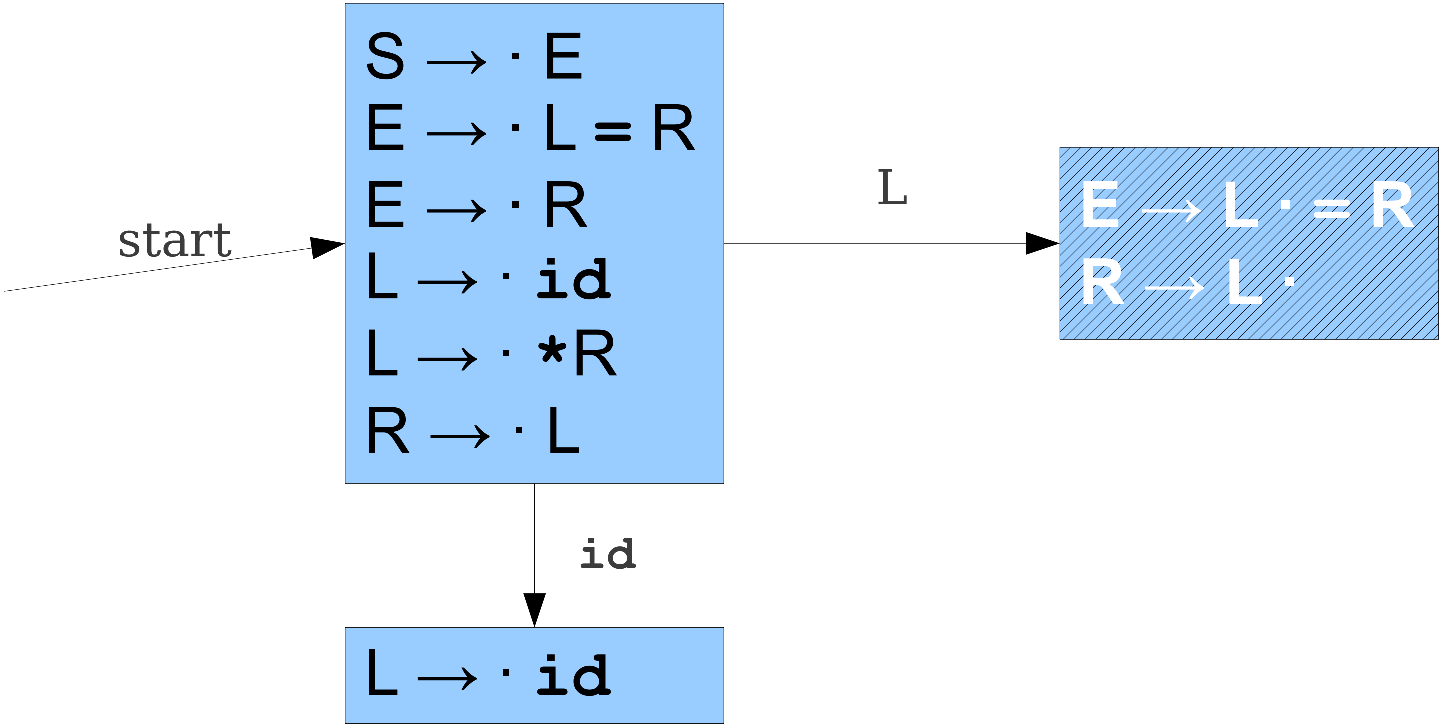
E → L · = R
R → L ·

id

L → · id

A Lack of Context

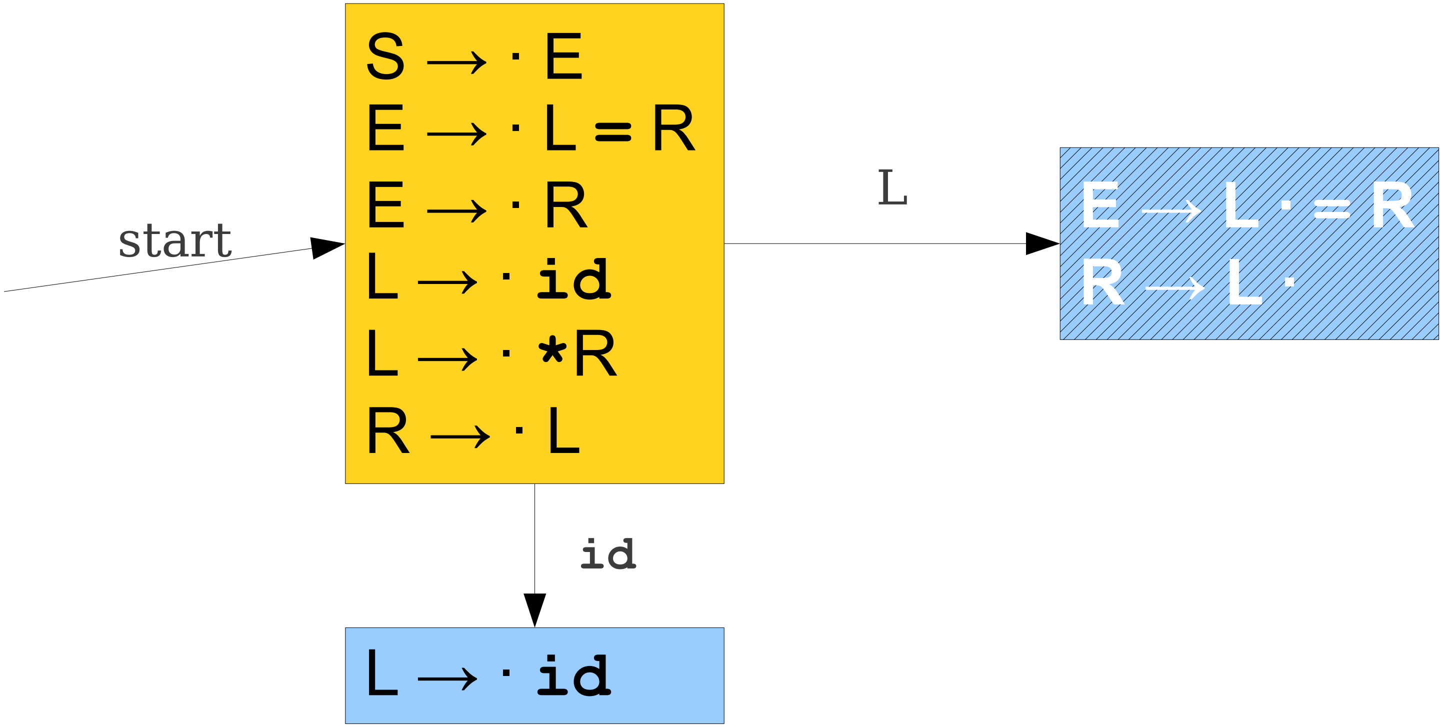
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



id	=	*	id

A Lack of Context

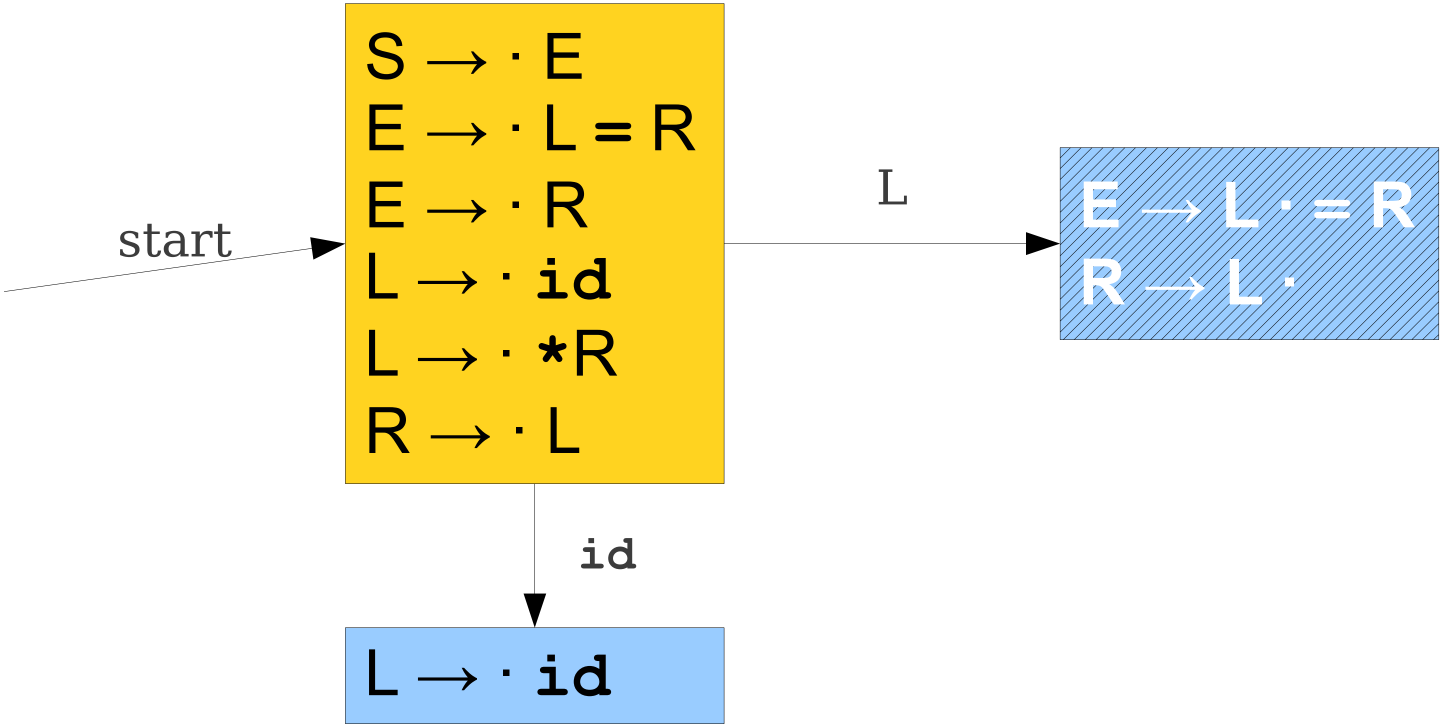
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



id	=	*	id

A Lack of Context

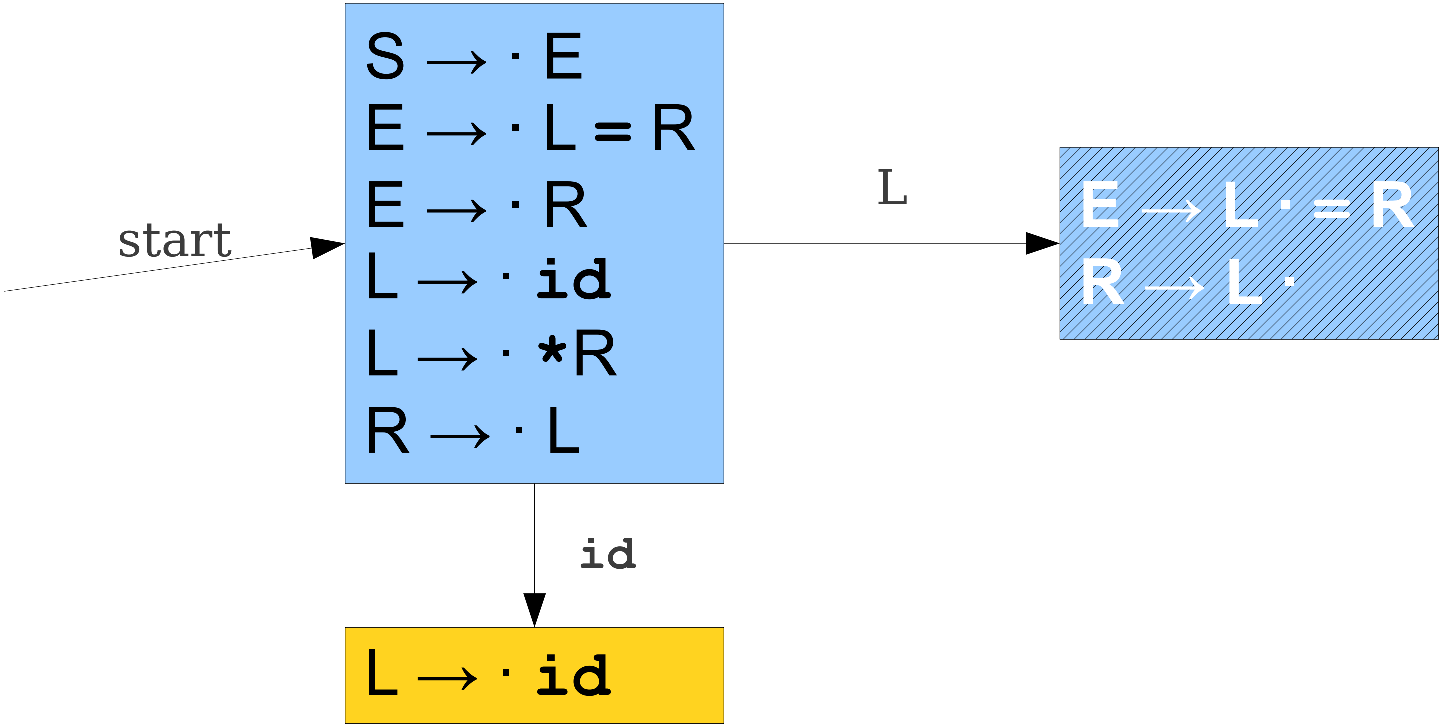
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



id	|	=	*	id

A Lack of Context

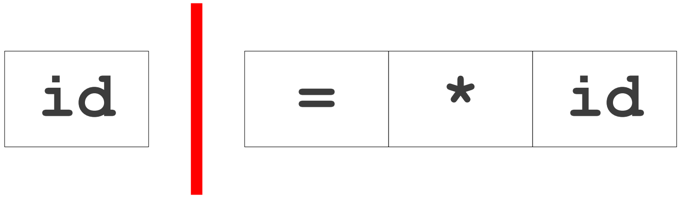
S → E
E → L = R
E → R
L → id
L → *R
R → L



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

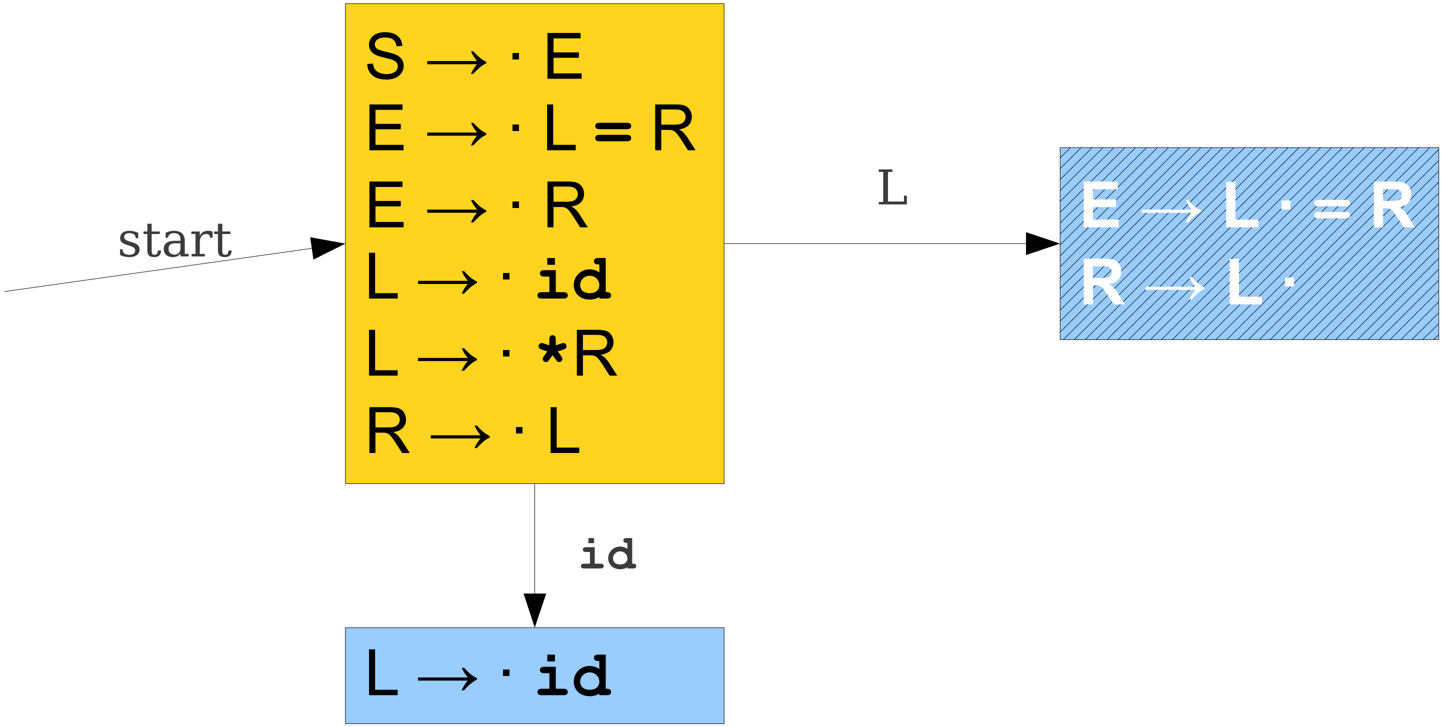
E → L · = R
R → L ·

L → · **id**



A Lack of Context

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



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

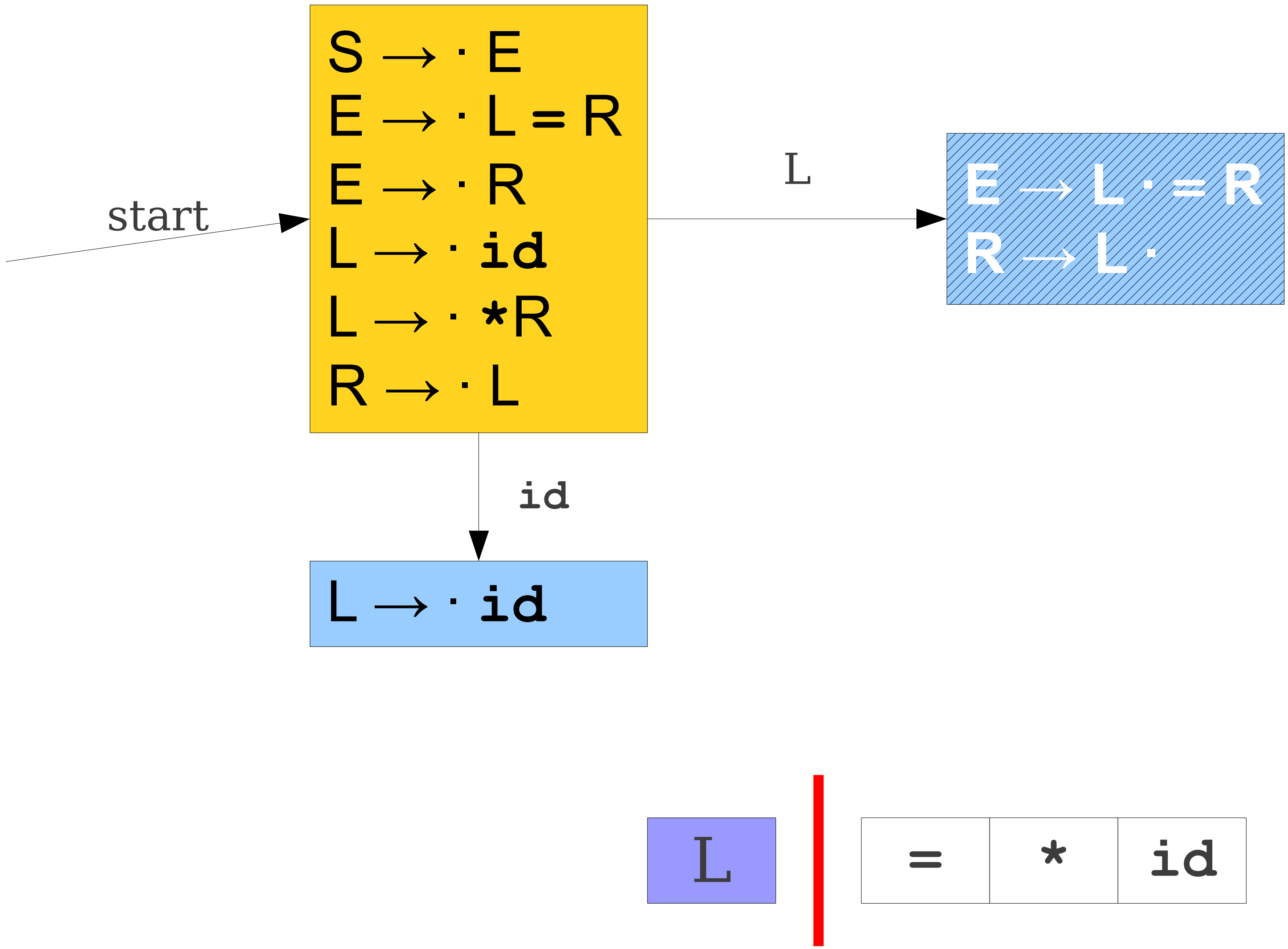
$E \rightarrow L \cdot = R$
$R \rightarrow L \cdot$

$L \rightarrow \cdot$ `id`

=	*	id

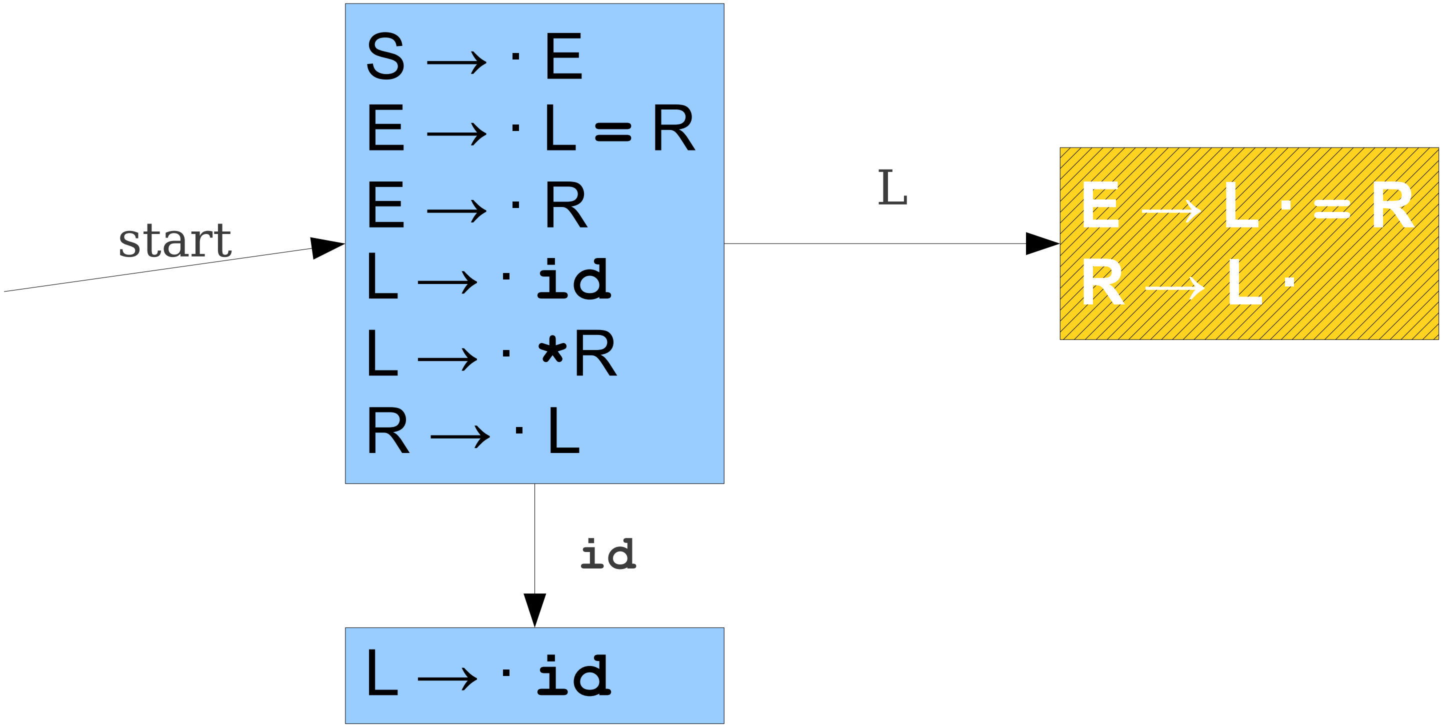
A Lack of Context

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

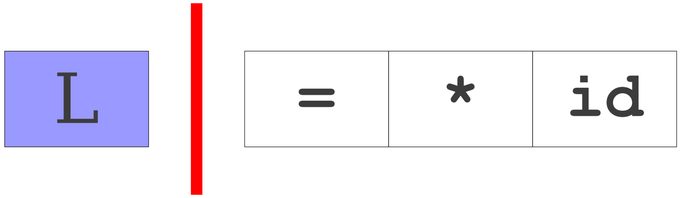
S → · E
E → · L = R
E → · R
L → · `id`
L → · `*`R
R → · L

L

E → L · = R
R → L ·

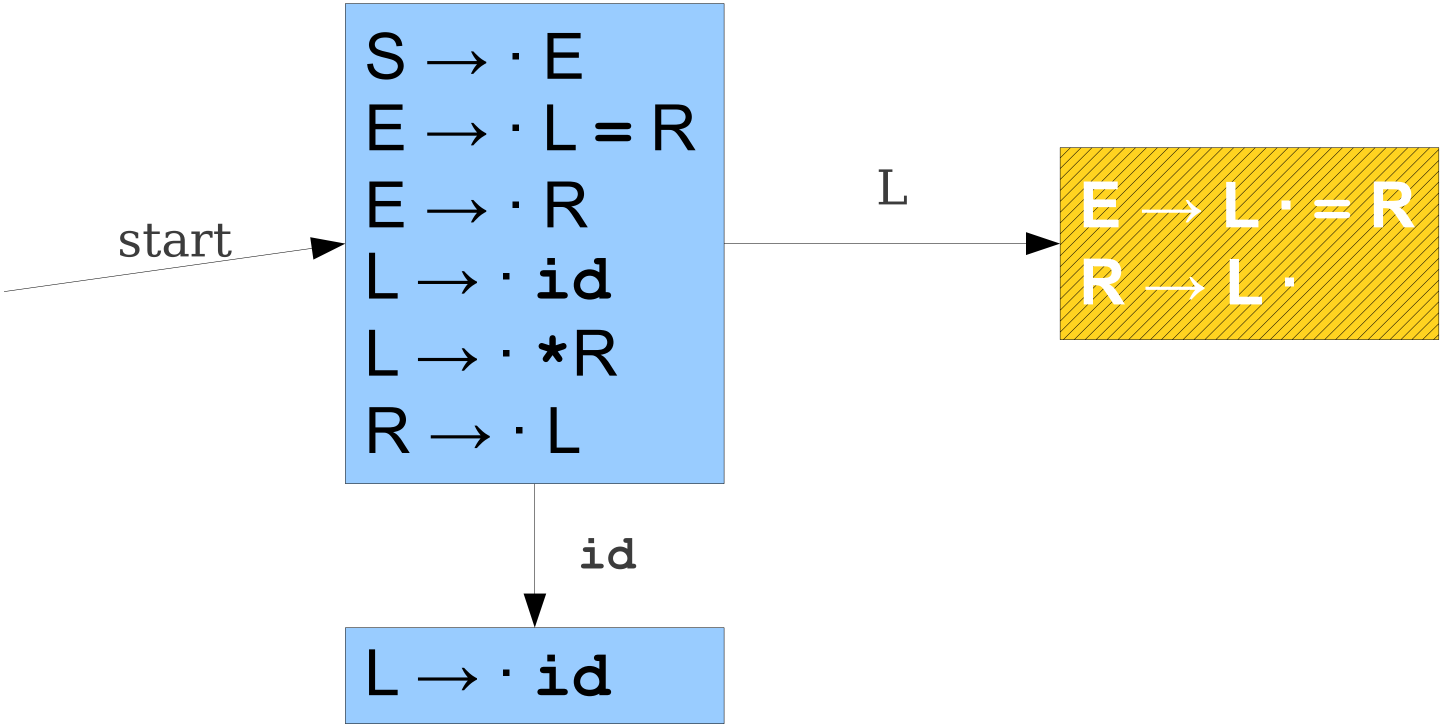
`id`

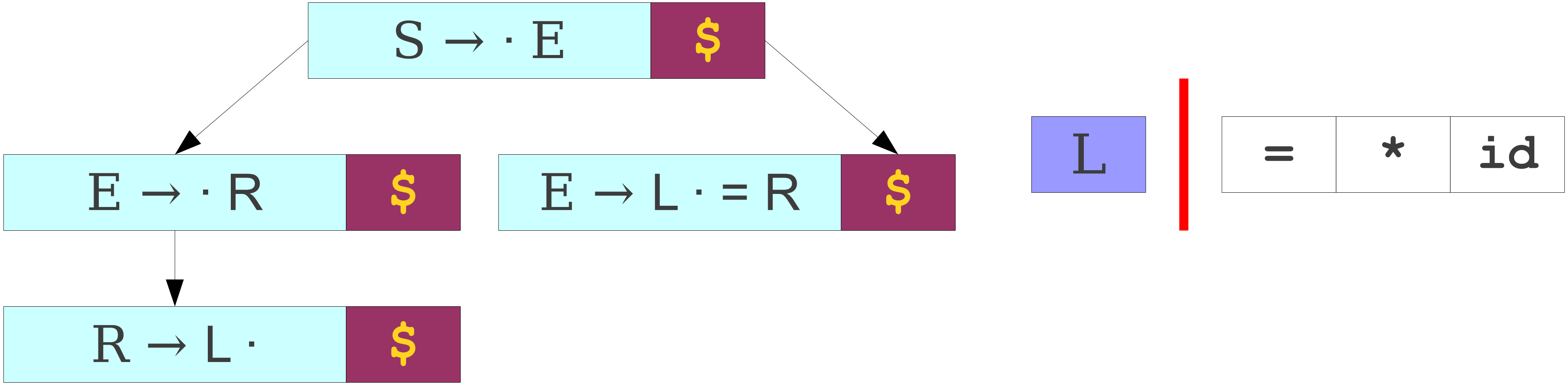
L → · `id`



A Lack of Context

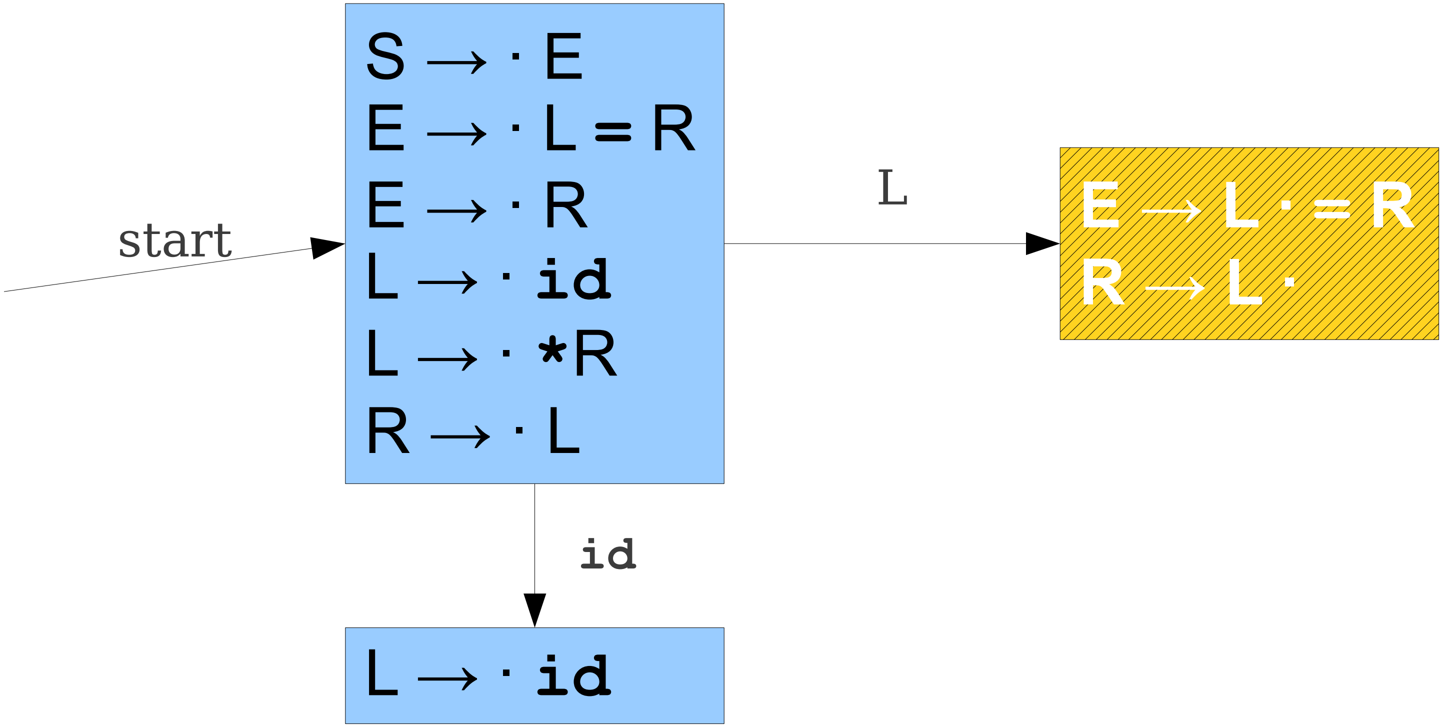
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**





A Lack of Context

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



start

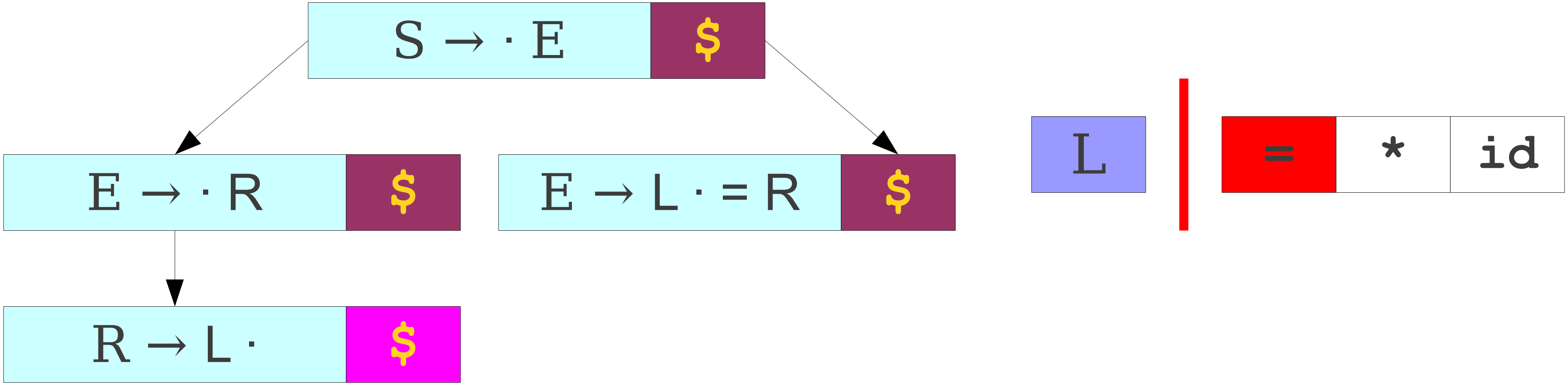
$S \to \cdot E$
$E \to \cdot L = R$
$E \to \cdot R$
$L \to \cdot$ `id`
$L \to \cdot$ `*`R
$R \to \cdot L$

L

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

`id`

$L \to \cdot$ `id`



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

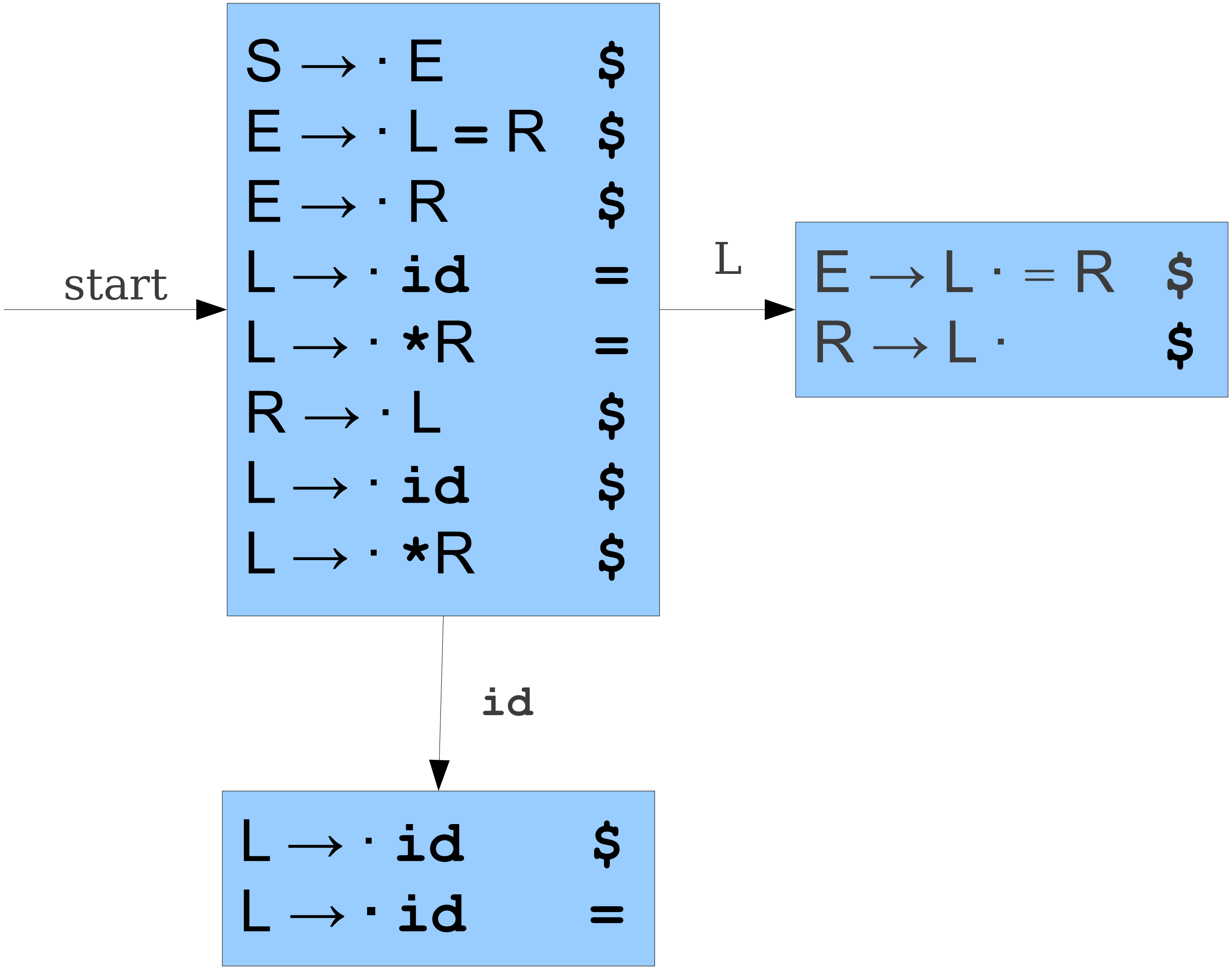
L	|	`=`	`*`	`id`

For Reference: LR(1) States

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

For Reference: LR(1) States

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



start

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

L

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

id

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

LR(1) and SLR(1)

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

Review of LR(1)

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

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

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

A Surprisingly Powerful Idea

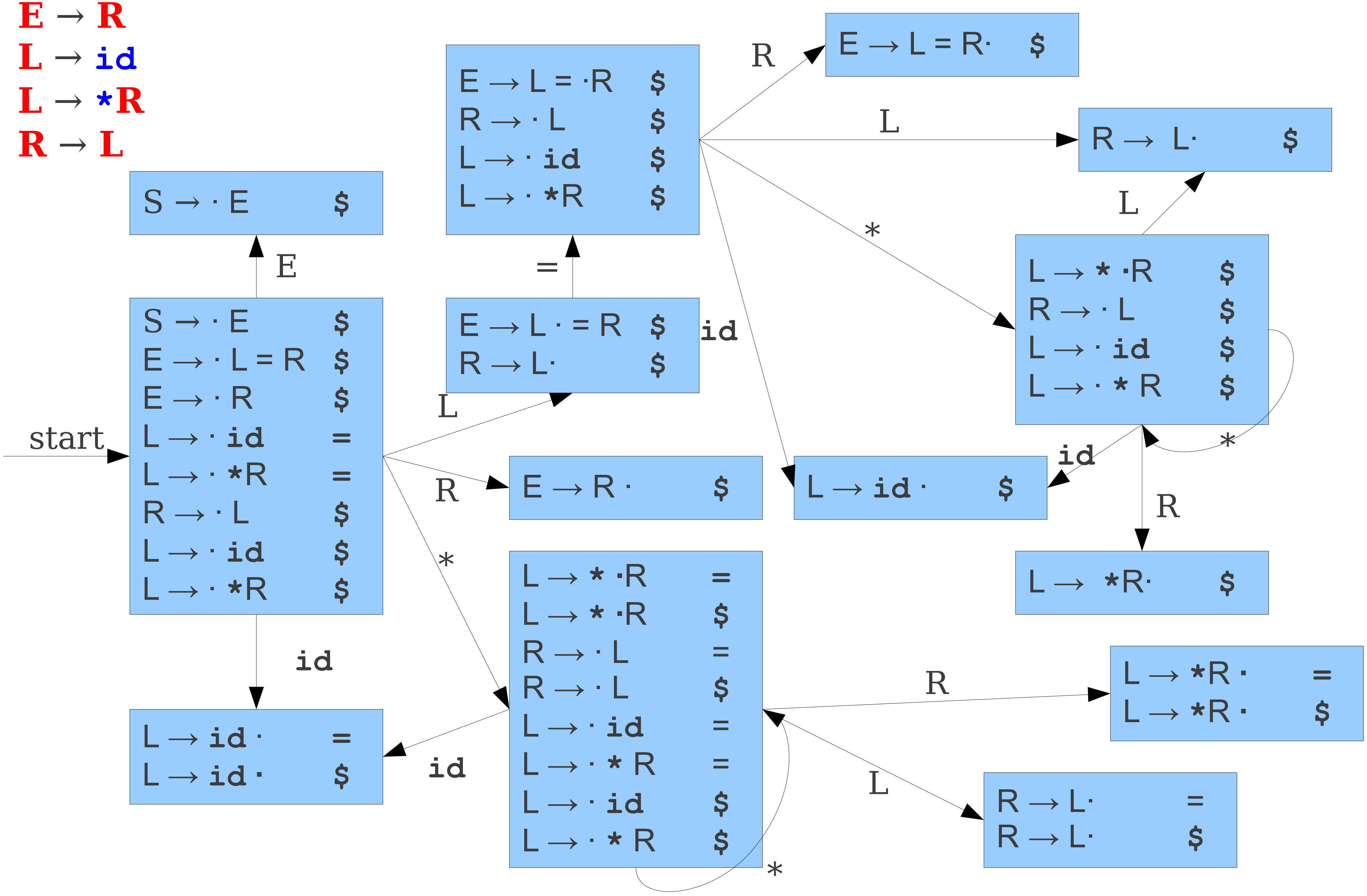
- In an LR(1) automaton, we have multiple states with the same core but different lookahead.
- What if we merge all these states together?
- This is called **LALR(1)**
    - **Lookahead(1) LR(0)**

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

From LR(1) to LALR(1)

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

From LR(1) to LALR(1)



S → ·E $

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

E → L = R· $

R → L· $

L → ***** ·R $
R → · L $
L → · **id** $
L → · ***** R $

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

E → L · = R $
R → L· $

E → R · $

L → **id** · $

L → ***R**· $

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

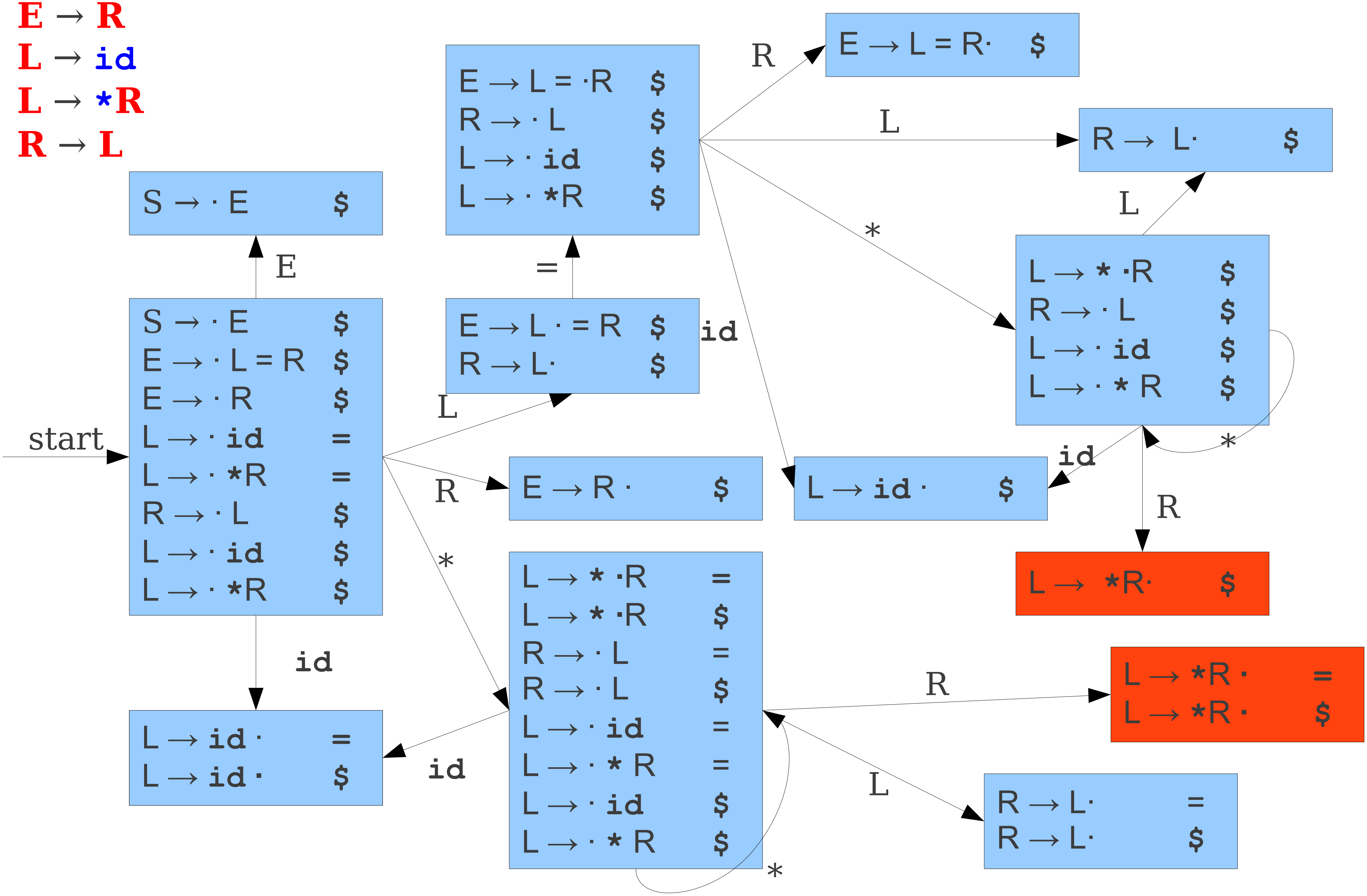
L → **id** ·   =
L → **id** · $

L → ***R** ·   =
L → ***R** · $

R → L·   =
R → L· $

From LR(1) to LALR(1)

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



S → · E $

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

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

E → L · = R $
R → L· $

E → L = R· $

R → L· $

L → * ·R $
R → · L $
L → · id $
L → · * R $

E → R · $

L → id · $

L → *R· $

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

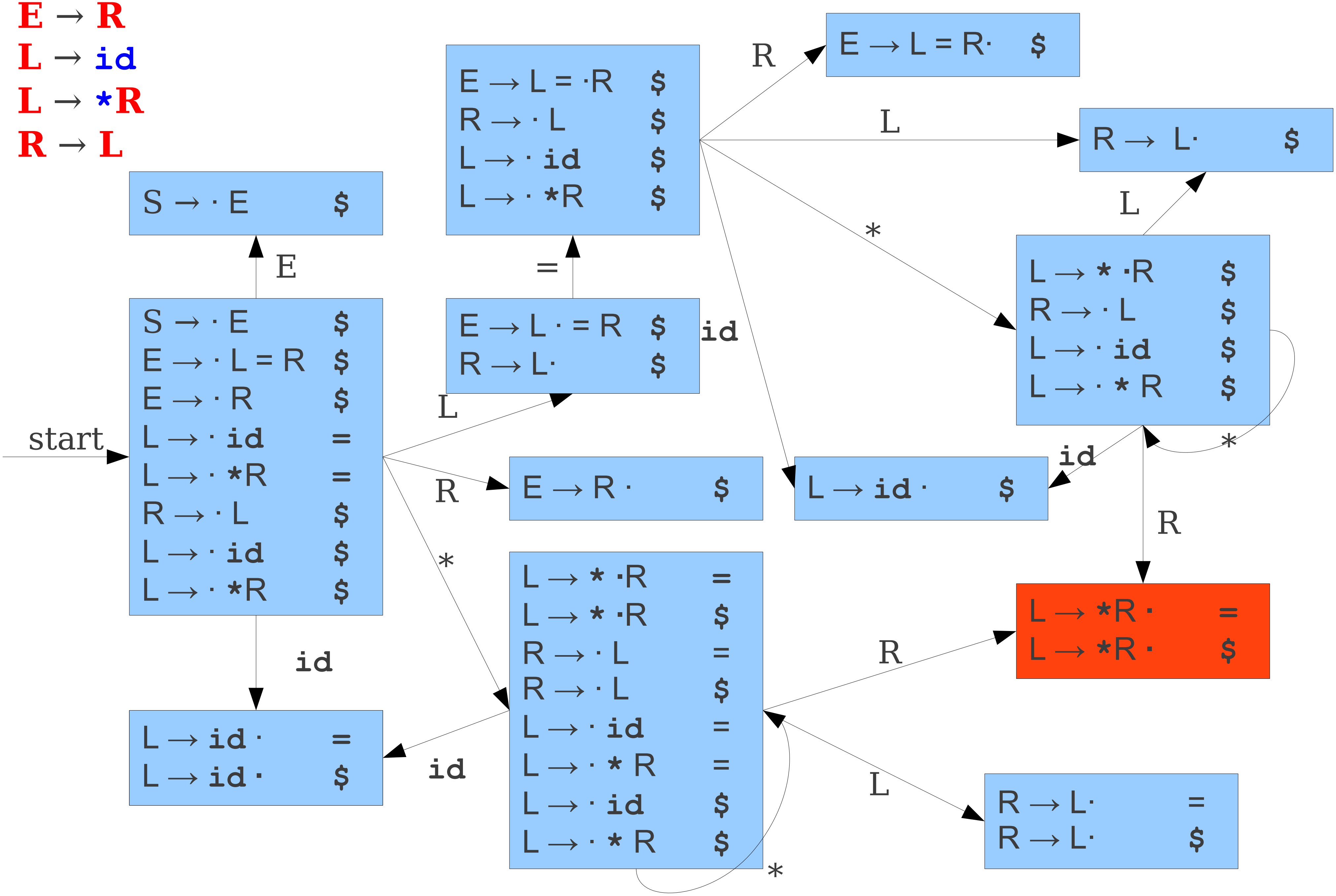
L → *R · =
L → *R · $

L → id · =
L → id · $

R → L· =
R → L· $

From LR(1) to LALR(1)

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



S → · E $

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

E → L = R· $

R → L· $

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

E → L · = R $
R → L· $

L → ***** ·R $
R → · L $
L → · **id** $
L → · ***** R $

E → R · $

L → **id** · $

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

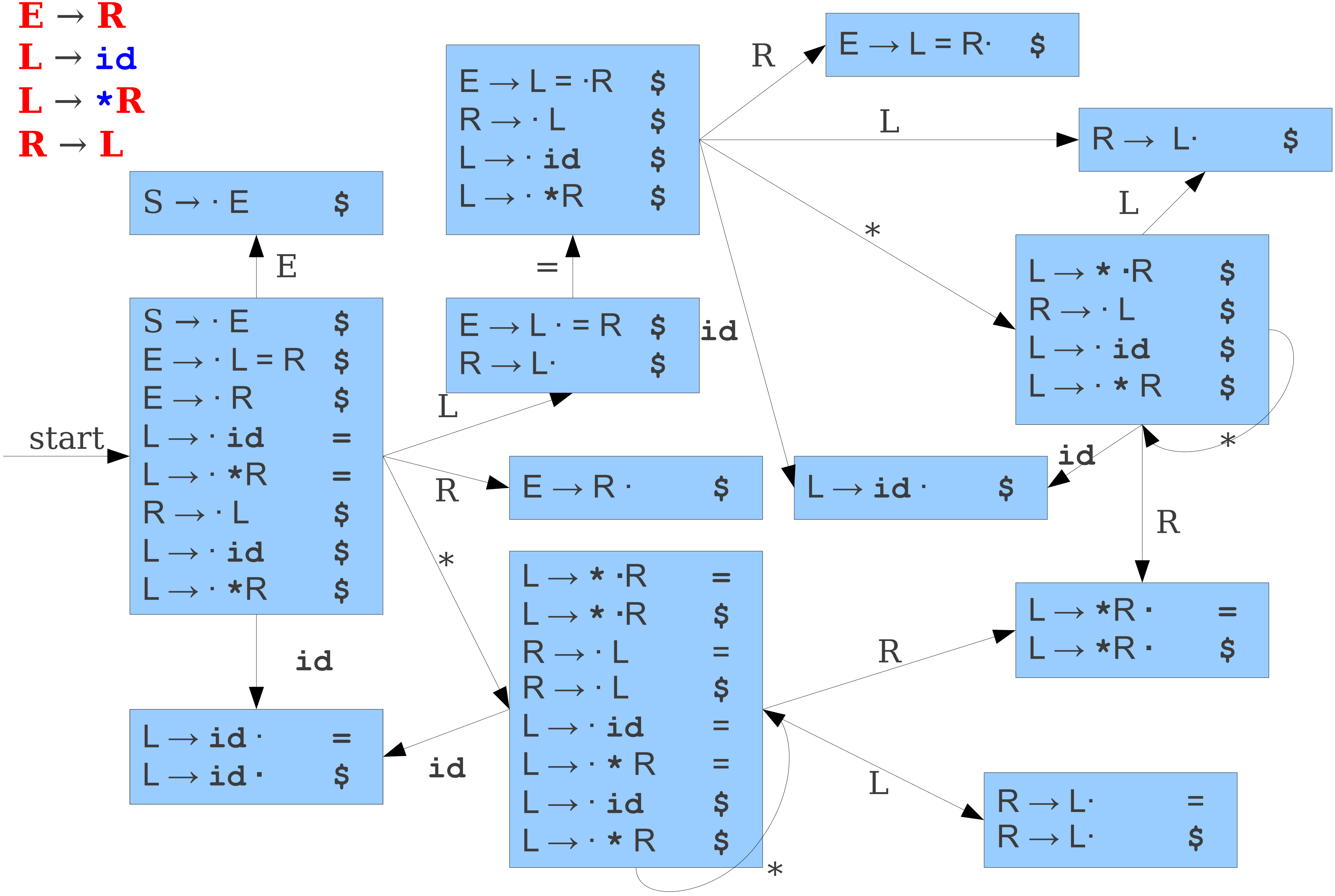
L → *****R · =
L → *****R · $

L → **id** · =
L → **id** · $

R → L· =
R → L· $

From LR(1) to LALR(1)

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



E → L = R· $

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

R → L· $

S → · E $

L → * ·R $
R → · L $
L → · id $
L → · * R $

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

E → L · = R $
R → L· $

E → R · $

L → id · $

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

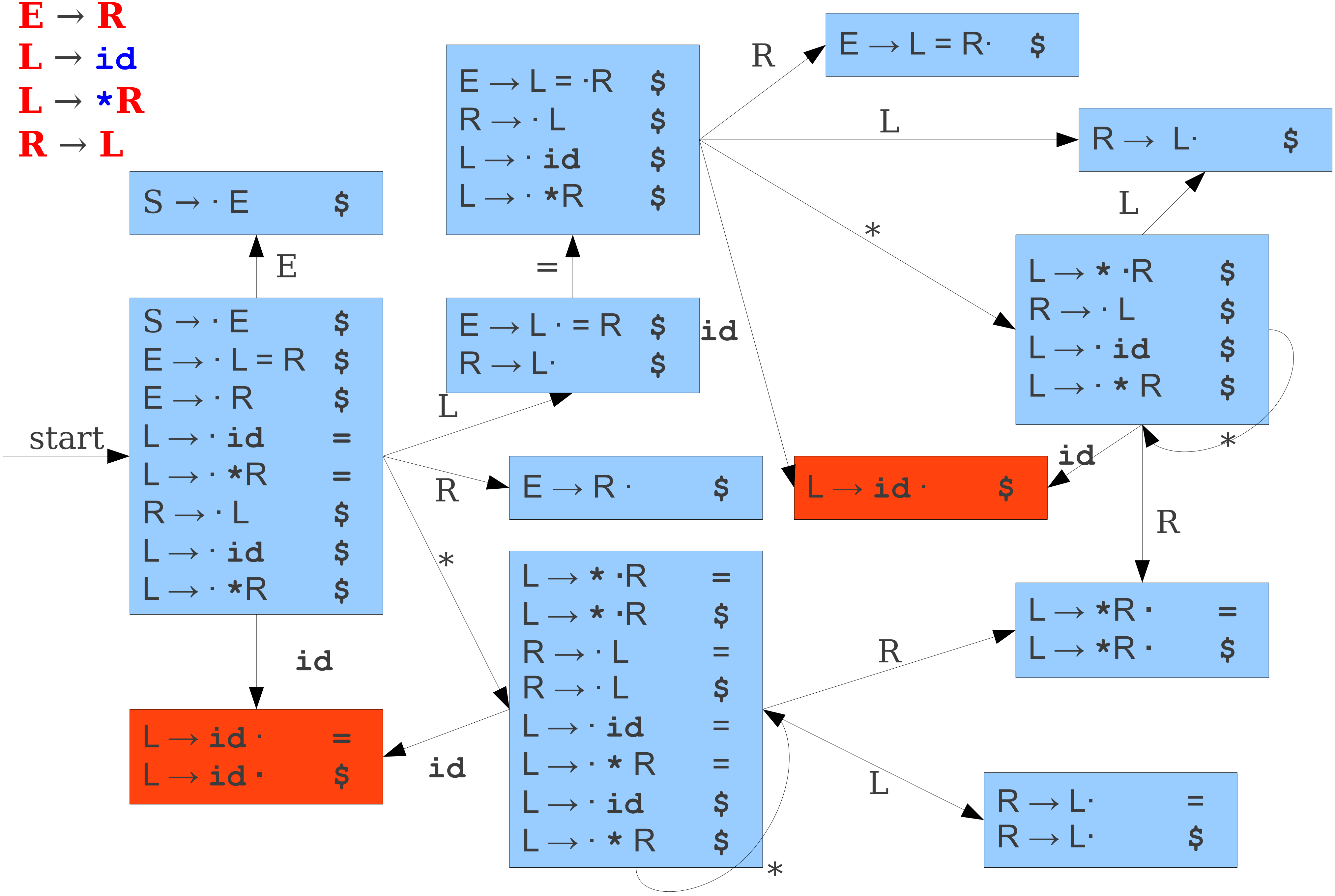
L → *R · =
L → *R · $

L → id · =
L → id · $

R → L· =
R → L· $

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

From LR(1) to LALR(1)


S → · E $

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

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

E → L · = R $
R → L· $

E → L = R· $

R → L· $

L → * ·R $
R → · L $
L → · id $
L → · * R $

E → R · $

L → id · $

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

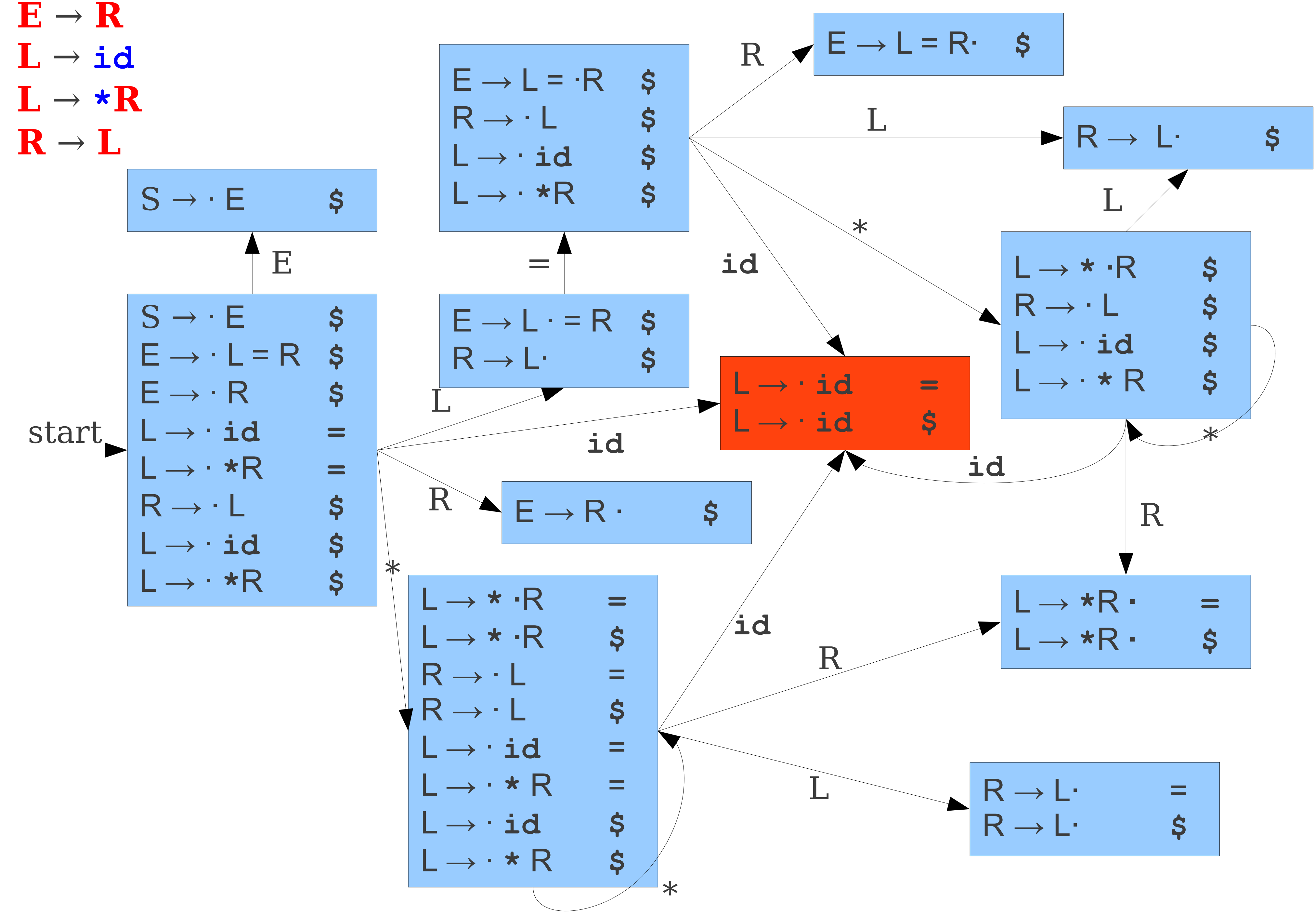
L → *R ·   =
L → *R · $

L → id ·   =
L → id · $

R → L·   =
R → L· $

From LR(1) to LALR(1)

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



S → · E $

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

E → L = R· $

R → L· $

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

E → L · = R $
R → L· $

L → · **id** =
L → · **id** $

L → * ·R $
R → · L $
L → · **id** $
L → · * R $

E → R · $

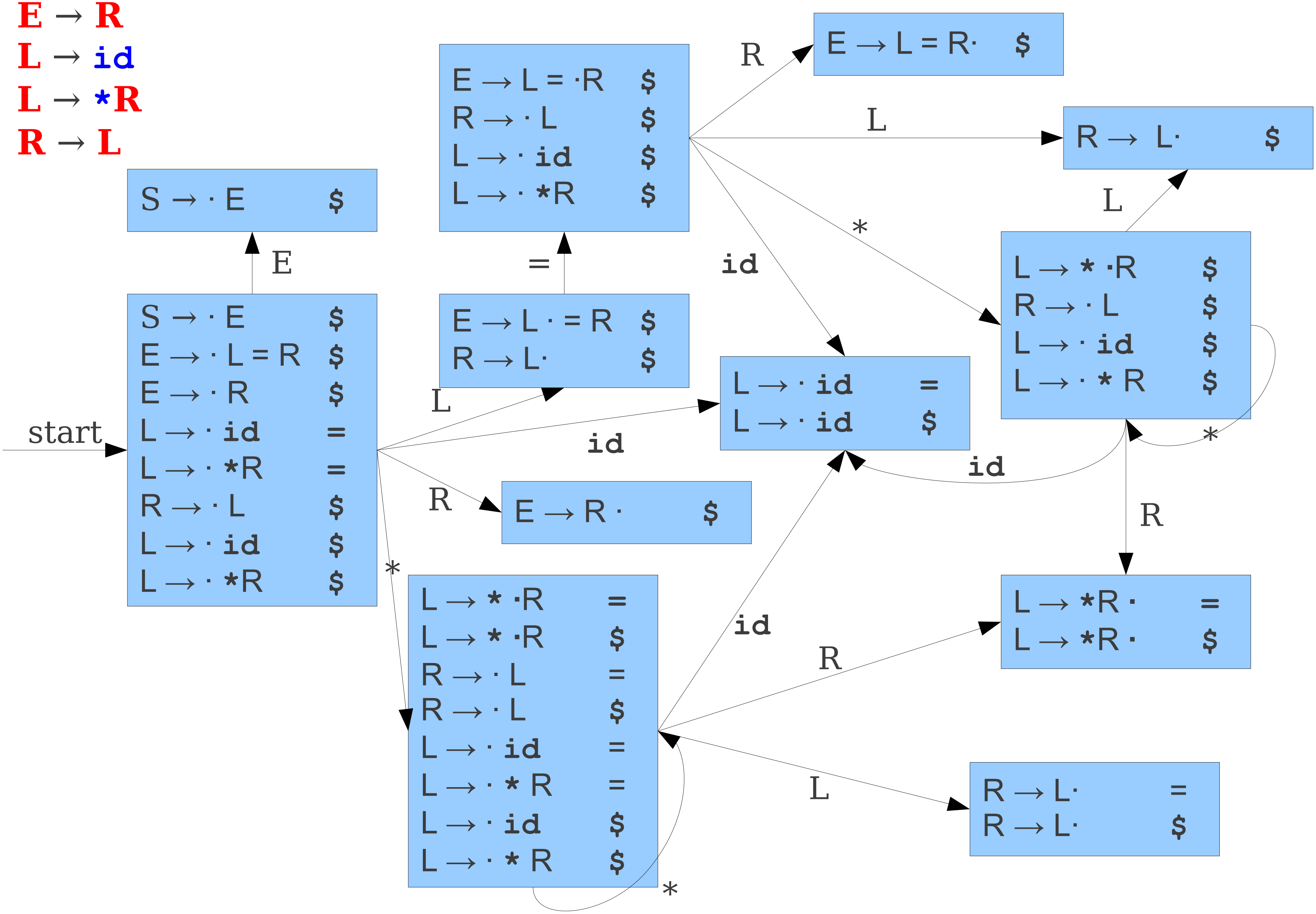
L → * ·R =
L → * ·R $
R → · L =
R → · L $
L → · **id** =
L → · * R =
L → · **id** $
L → · * R $

L → *R · =
L → *R · $

R → L· =
R → L· $

From LR(1) to LALR(1)

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



S → · E $

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

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

E → L · = R $
R → L· $

E → L = R· $

R → L· $

L → · id =
L → · id $

E → R · $

L → * ·R $
R → · L $
L → · id $
L → · * R $

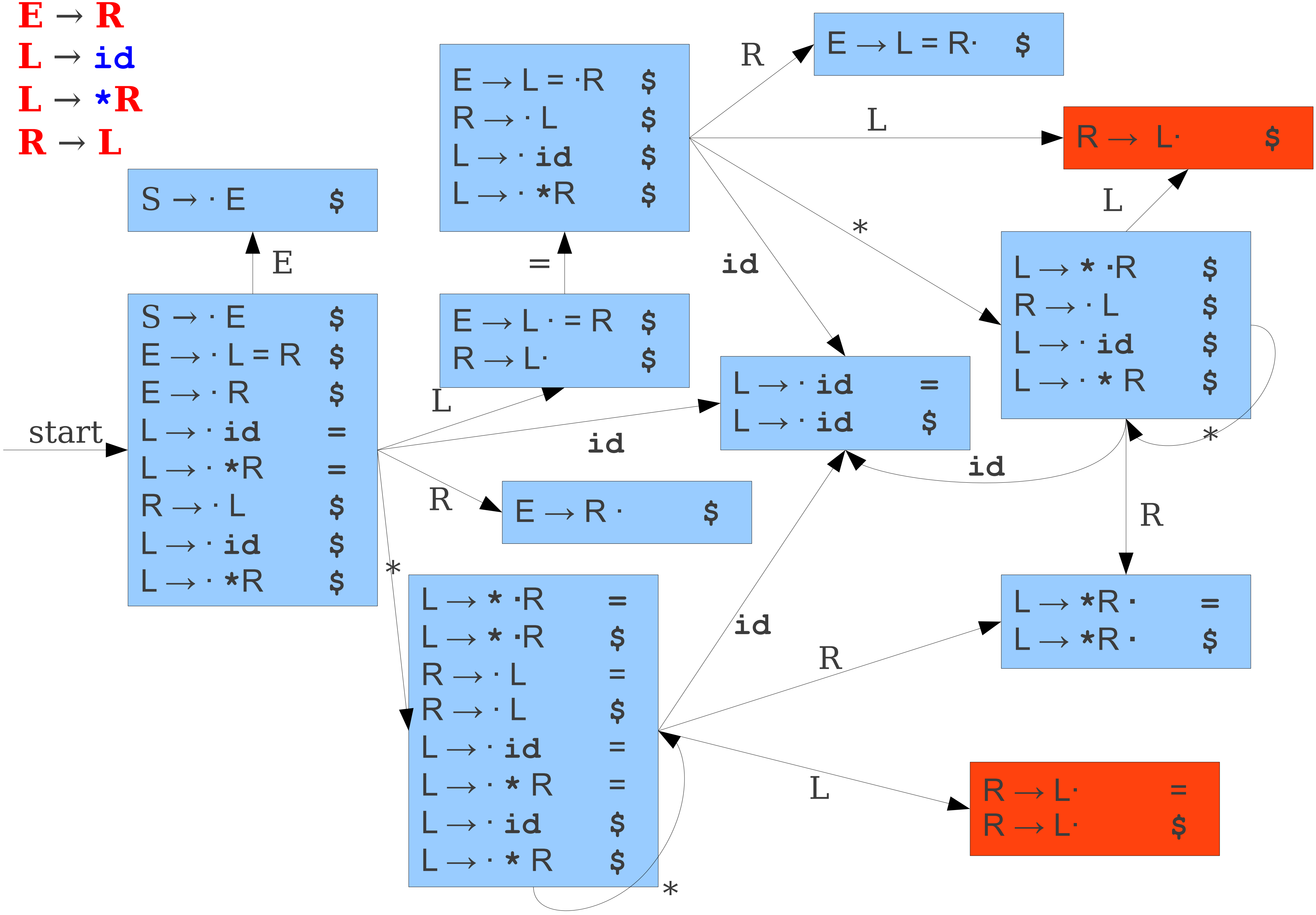
L → * ·R =
L → * ·R $
R → · L =
R → · L $
L → · id =
L → · * R =
L → · id $
L → · * R $

L → *R · =
L → *R · $

R → L· =
R → L· $

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

From LR(1) to LALR(1)



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

E → L = R· $

R → L· $

S → · E $

L → * ·R $
R → · L $
L → · id $
L → · * R $

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

E → L · = R $
R → L· $

L → · id =
L → · id $

E → R · $

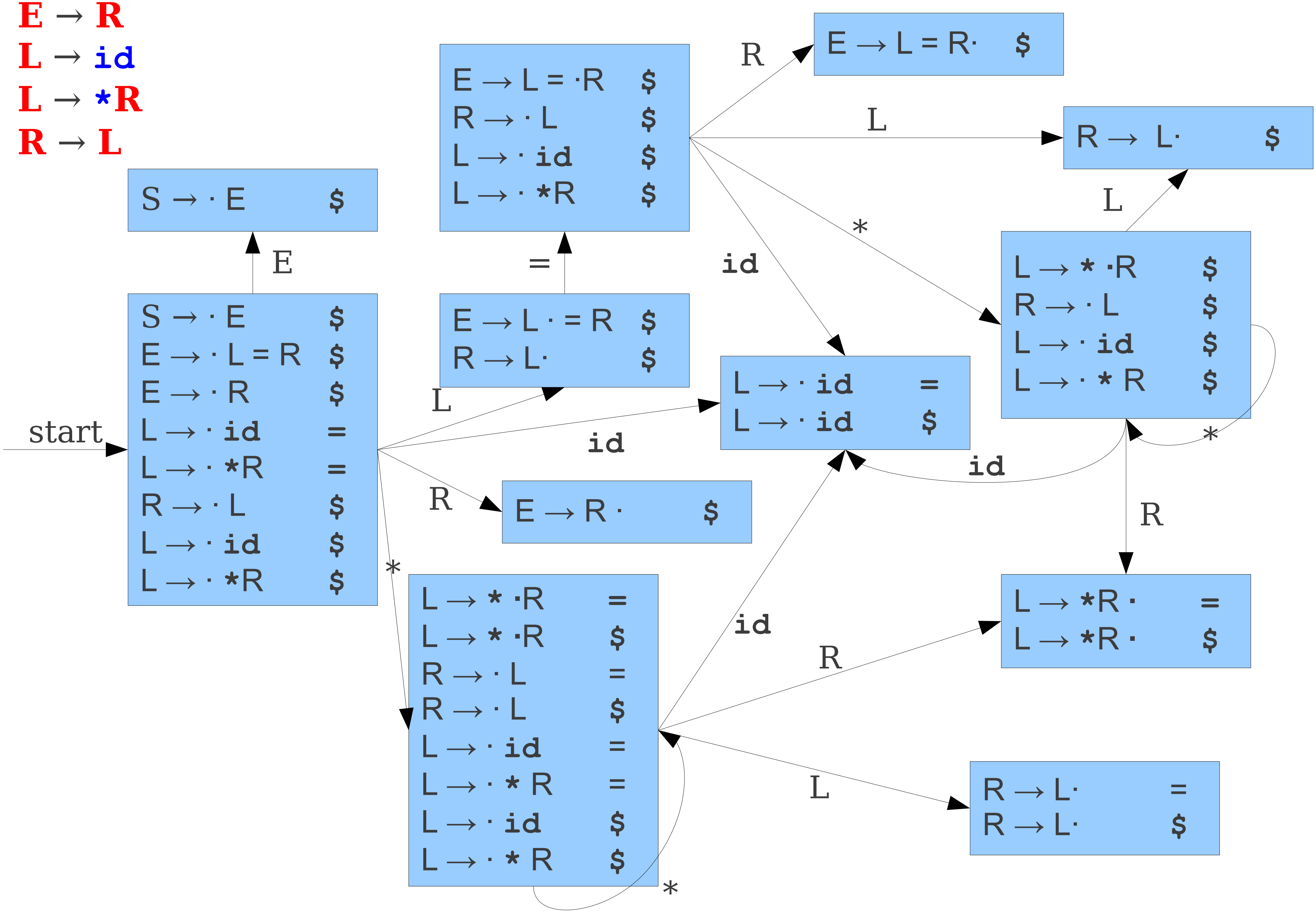
L → * ·R =
L → * ·R $
R → · L =
R → · L $
L → · id =
L → · * R =
L → · id $
L → · * R $

L → *R · =
L → *R · $

R → L· =
R → L· $

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

From LR(1) to LALR(1)



S → · E $

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

E → L · = R $
R → L · $

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

E → L = R· $

R → L· $

L → * ·R $
R → · L $
L → · id $
L → · * R $

L → · id   =
L → · id $

E → R · $

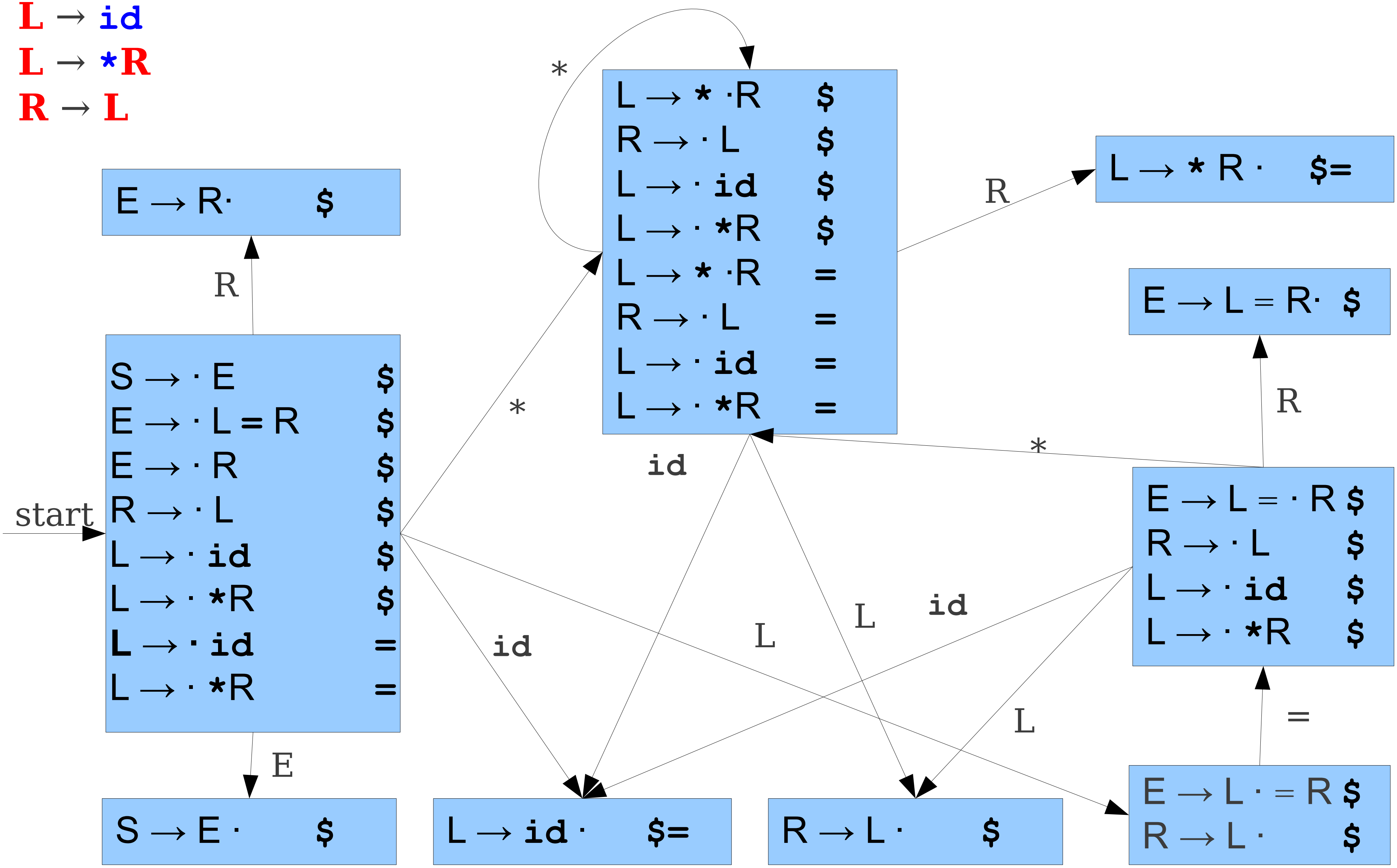
L → * ·R   =
L → * ·R $
R → · L   =
R → · L $
L → · id   =
L → · * R   =
L → · id $
L → · * R $

L → *R ·   =
L → *R · $

R → L·   =
R → L· $

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

From LR(1) to LALR(1)



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

E → R· $

S → E · $

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

L → * R · $=

L → id · $=

R → L · $

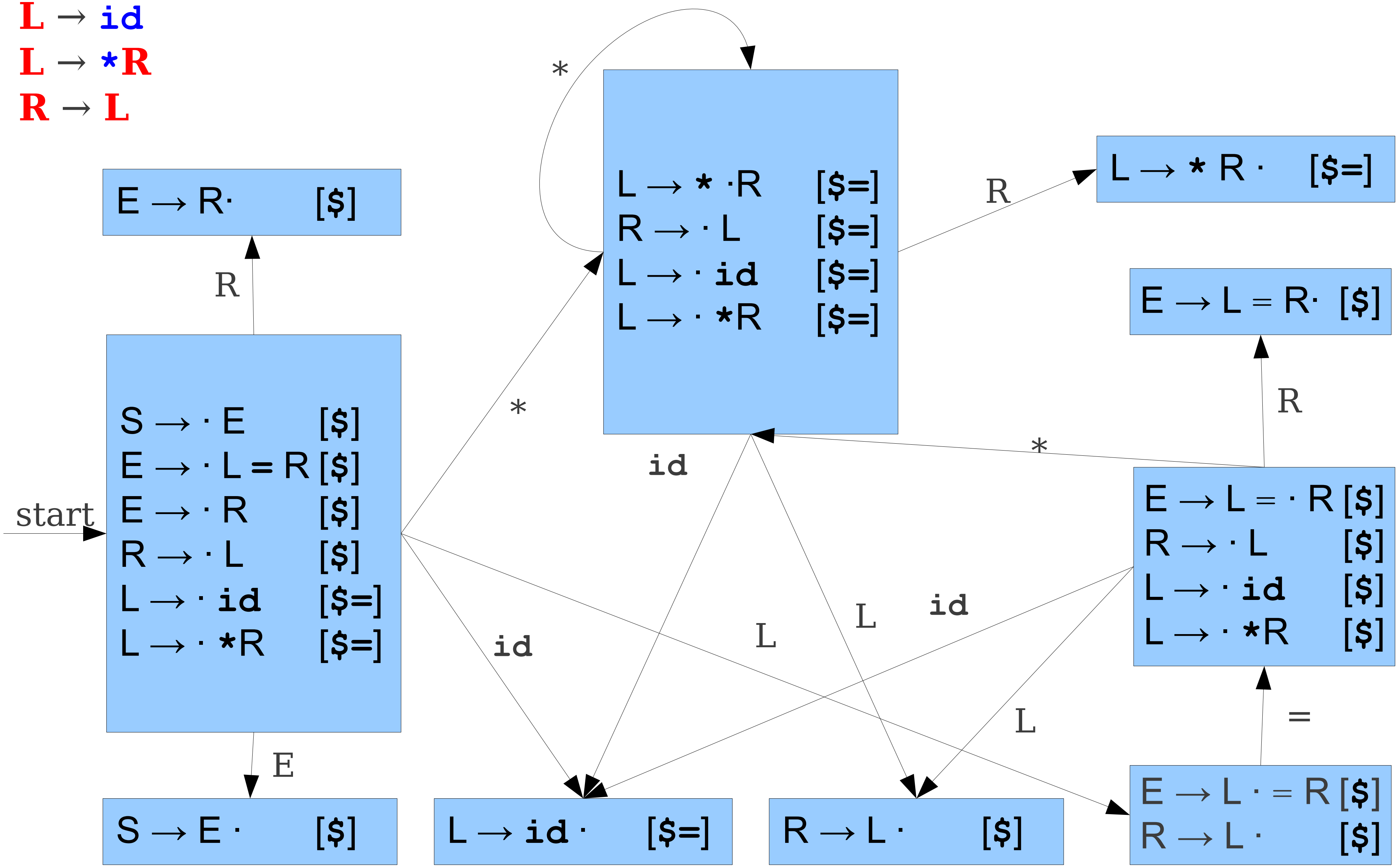
E → L = R· $

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

E → L · = R $
R → L · $

From LR(1) to LALR(1)

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



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

E → R· [$]

S → E · [$]

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

L → ***** R · [$=]

L → **id** · [$=]

R → L · [$]

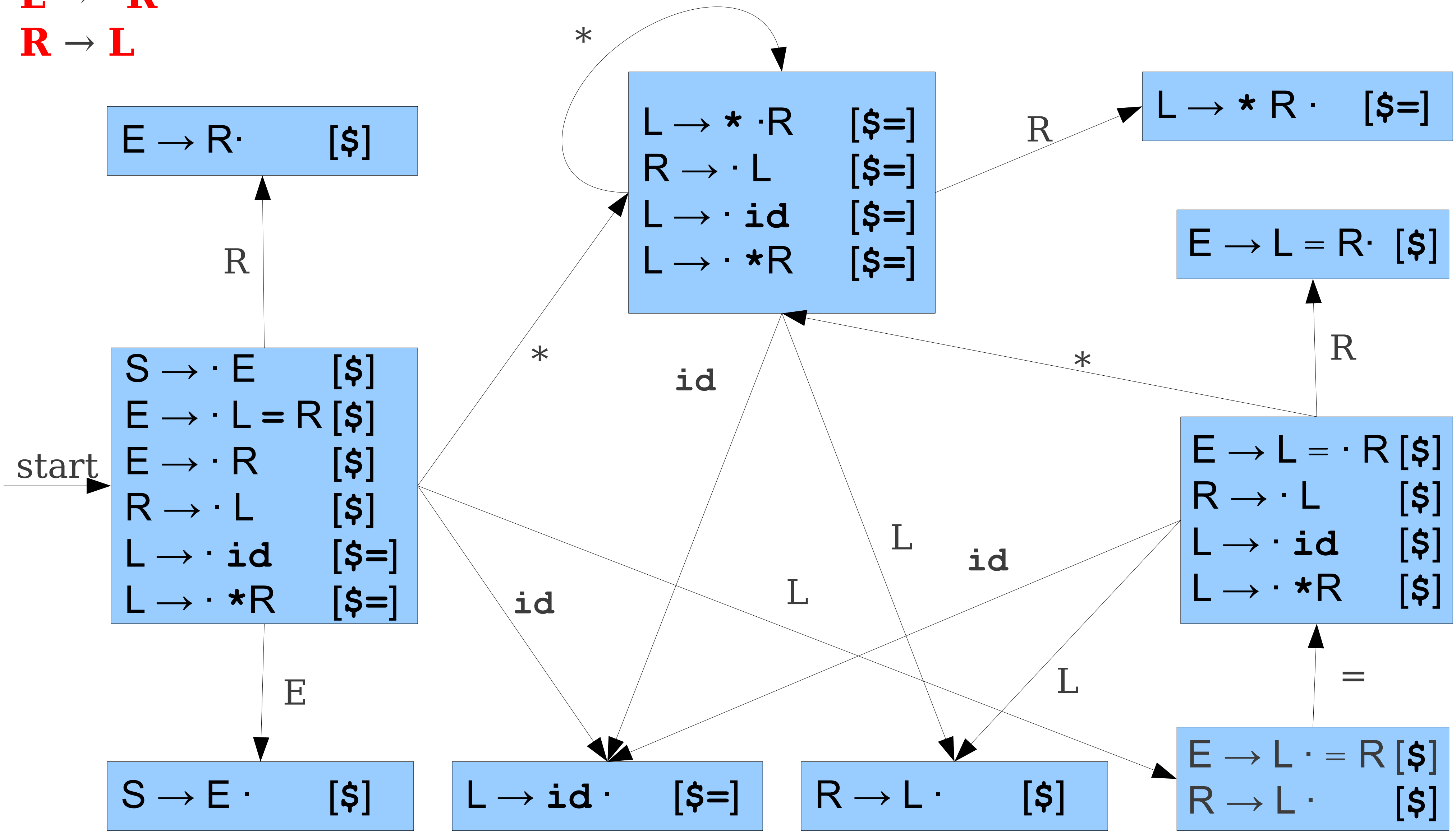
E → L = R· [$]

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

E → L · = R [$]
R → L · [$]

From LR(1) to LALR(1)

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



E → R· [$]

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

L → * R · [$=]

E → L = R· [$]

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

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

S → E · [$]

L → id · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

Advantages of LALR(1)

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

LALR(1) is Powerful

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

LALR(1) isn't LR(1)

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

Constructing LALR(1) Automata

- It's not a good idea to build LALR(1) automata from LR(1) automata.
- **Why?**
- LR(1) automata are impractically large.
- Are there more efficient methods for LALR(1) automata construction?
- **Yes**; we'll see two.

The “Lazy Merging” Technique

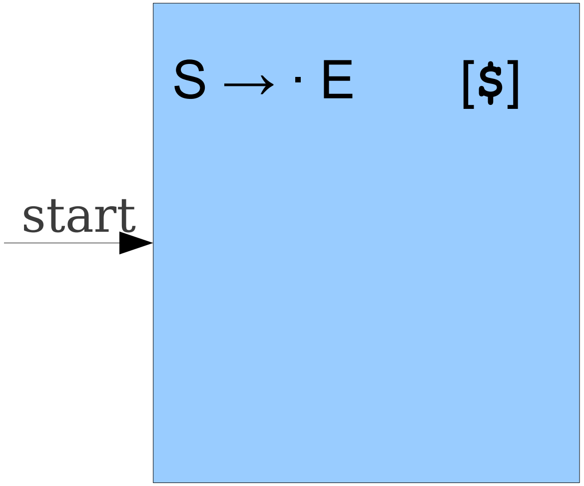
- Idea: Merge together LR(1) states as they're generated.
- Maintain a worklist of states to process; begin with the initial LR(1) state.
- When adding a new state, if it has the same core as an old state, update the old state and put it back in the worklist.

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

LALR(1) Construction

LALR(1) Construction

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

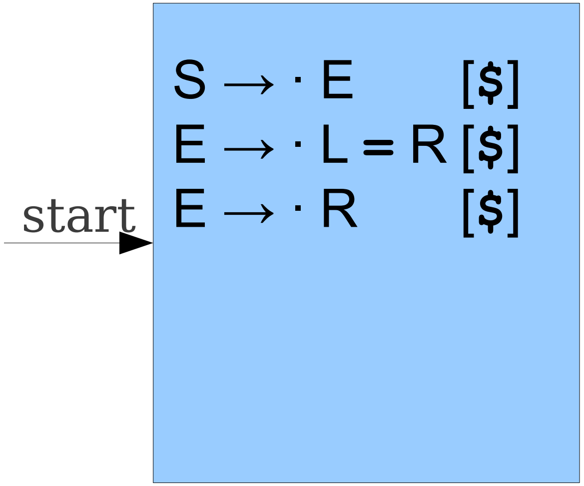


S → · E [$]

start

LALR(1) Construction

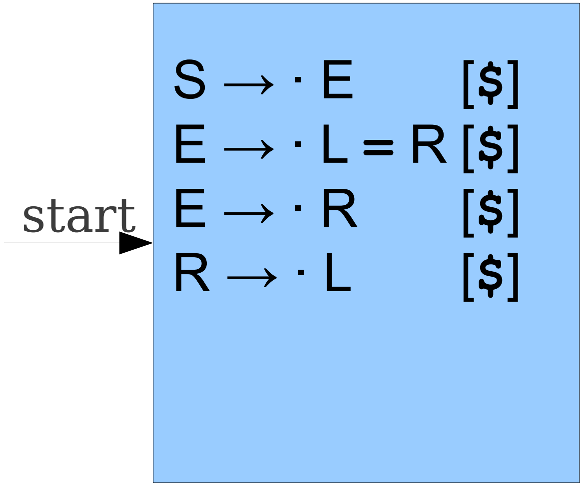
S → **E**
E → **L** `=` **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



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

LALR(1) Construction

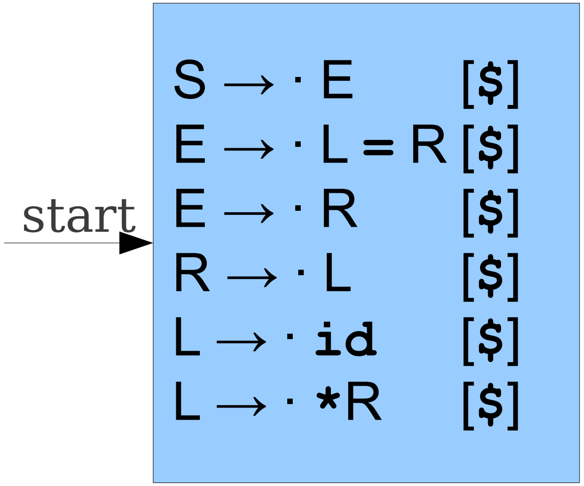
S → **E**
E → **L = R**
E → **R**
L → **id**
L → ***R**
R → **L**



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

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

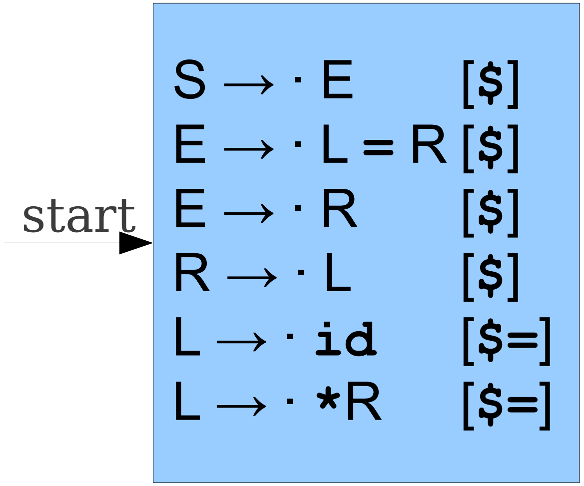
LALR(1) Construction



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

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

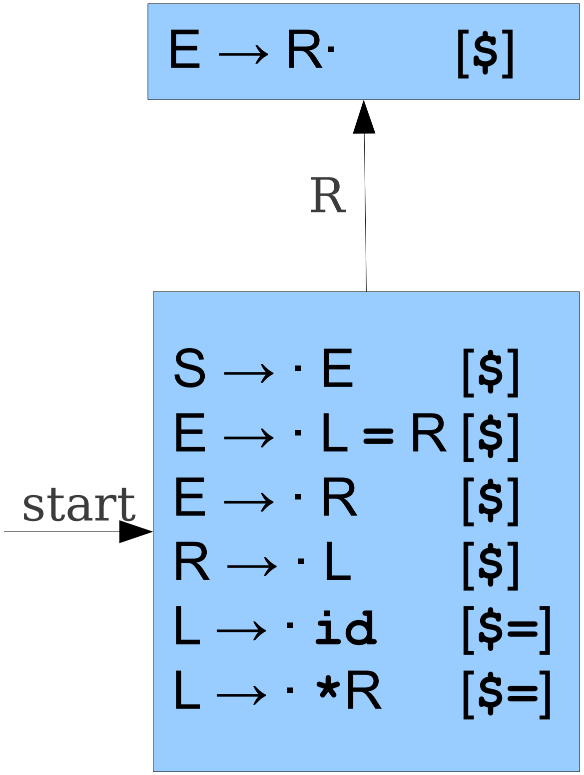
LALR(1) Construction



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

LALR(1) Construction

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

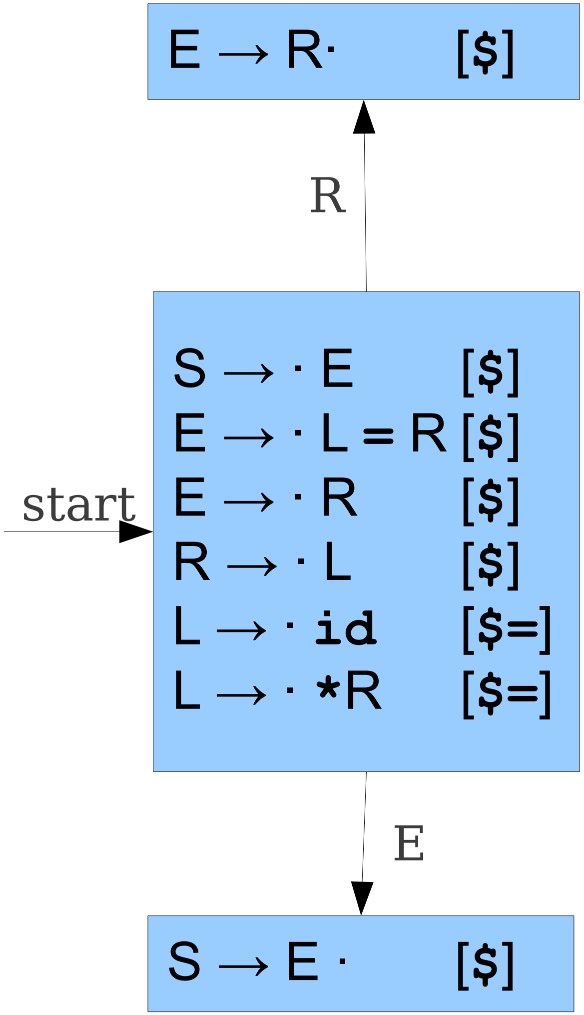


E → R·   [$]

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

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

LALR(1) Construction



E → R· [$]

R

start

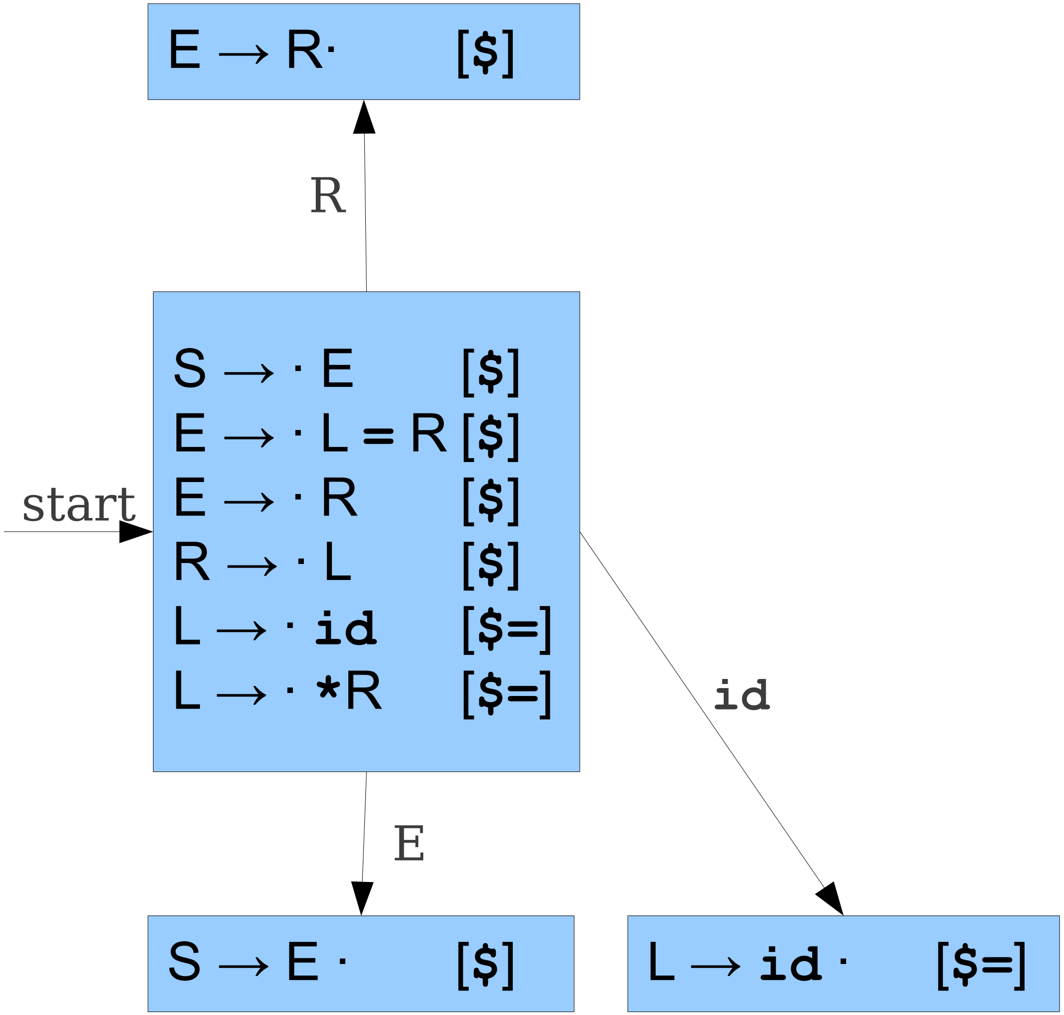
S → · E [$]
E → · L = R [$]
E → · R [$]
R → · L [$]
L → · id [$=]
L → · *R [$=]

E

S → E · [$]

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

LALR(1) Construction



E → R·   [$]

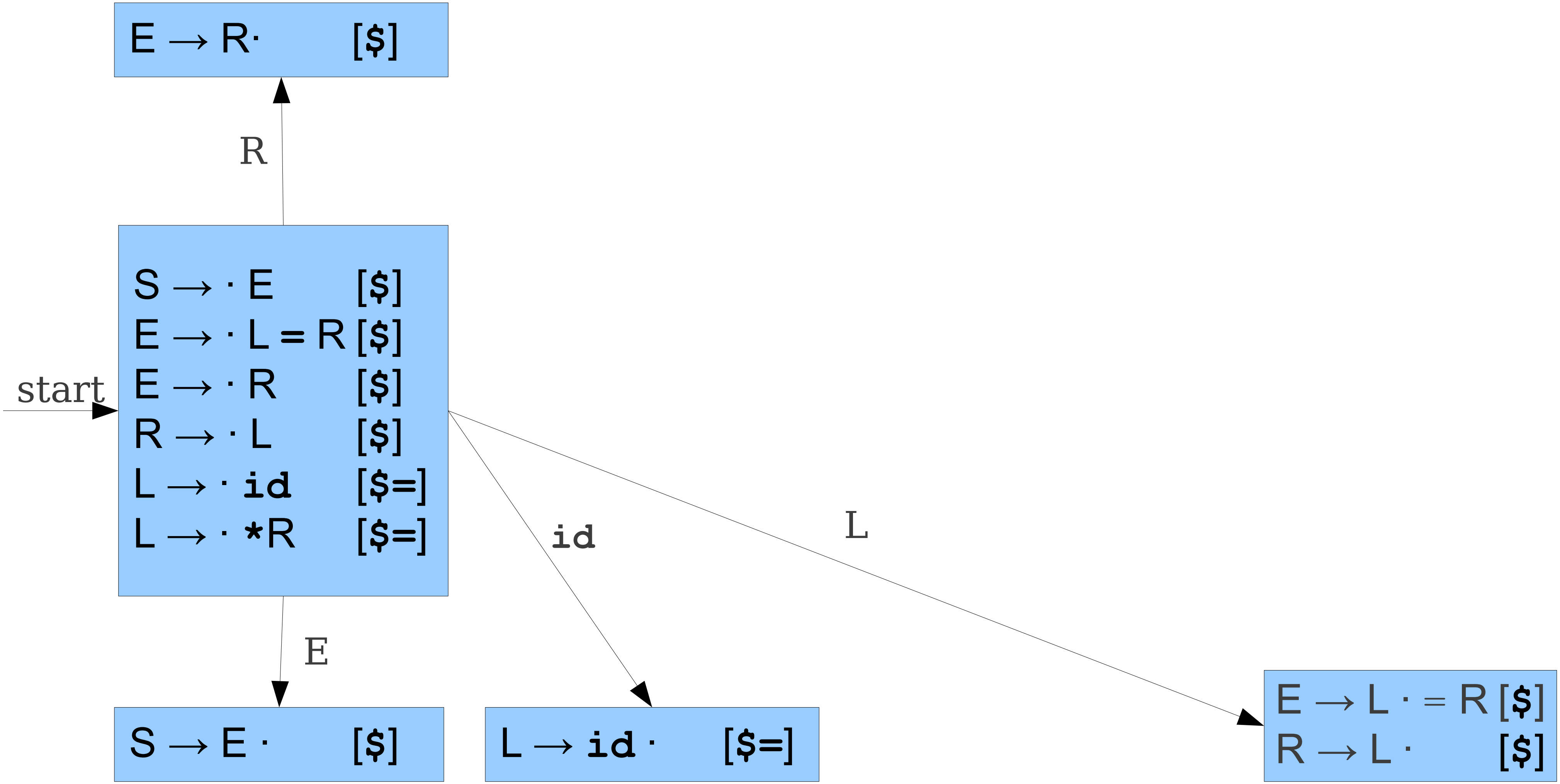
S → · E   [$]
E → · L = R [$]
E → · R   [$]
R → · L   [$]
L → · **id**   [$=]
L → · *****R   [$=]

S → E ·   [$]

L → **id** ·   [$=]

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

LALR(1) Construction



E → R· [$]

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

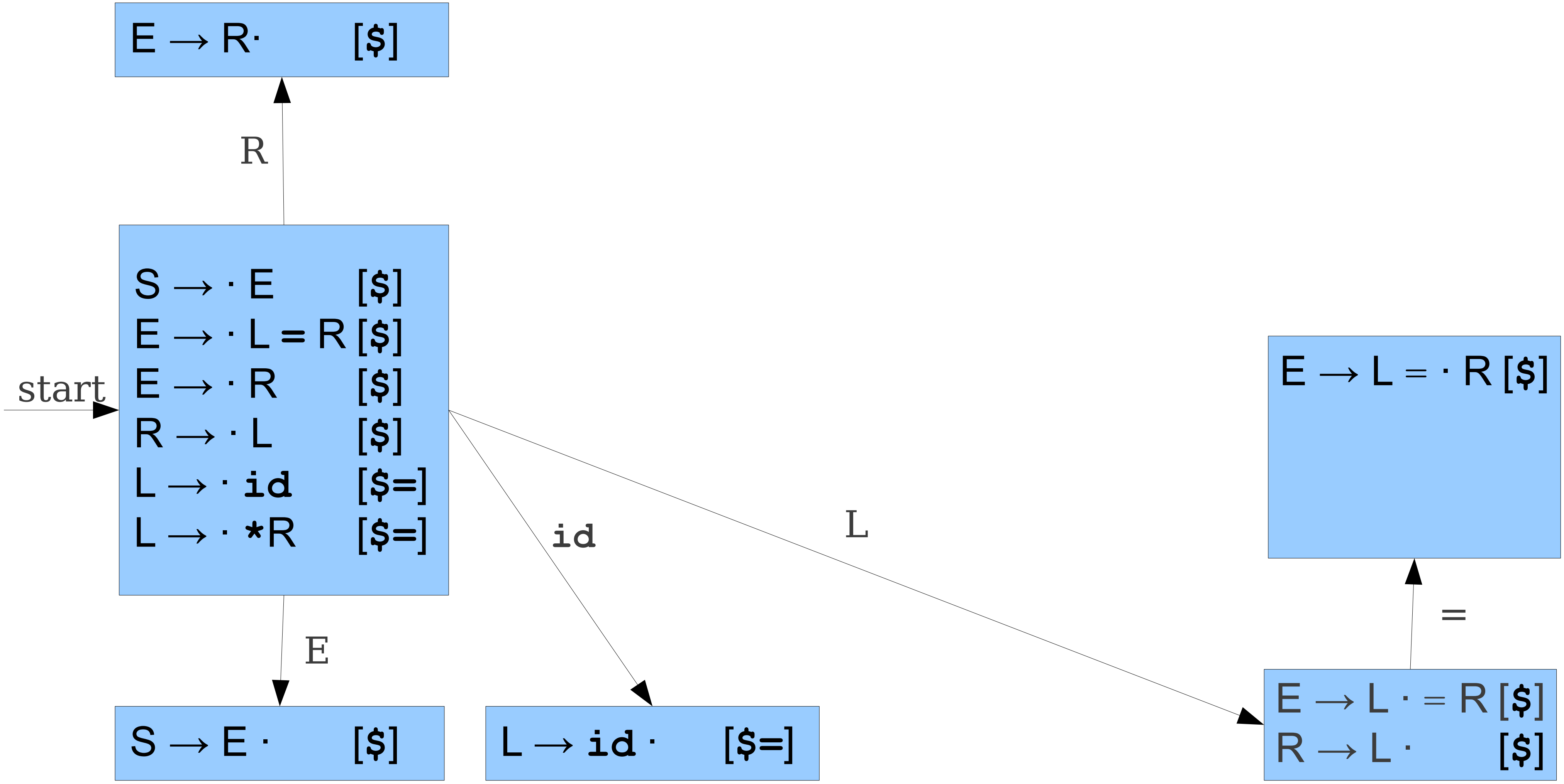
S → E · [$]

L → **id** · [$=]

E → L · = R [$]
R → L · [$]

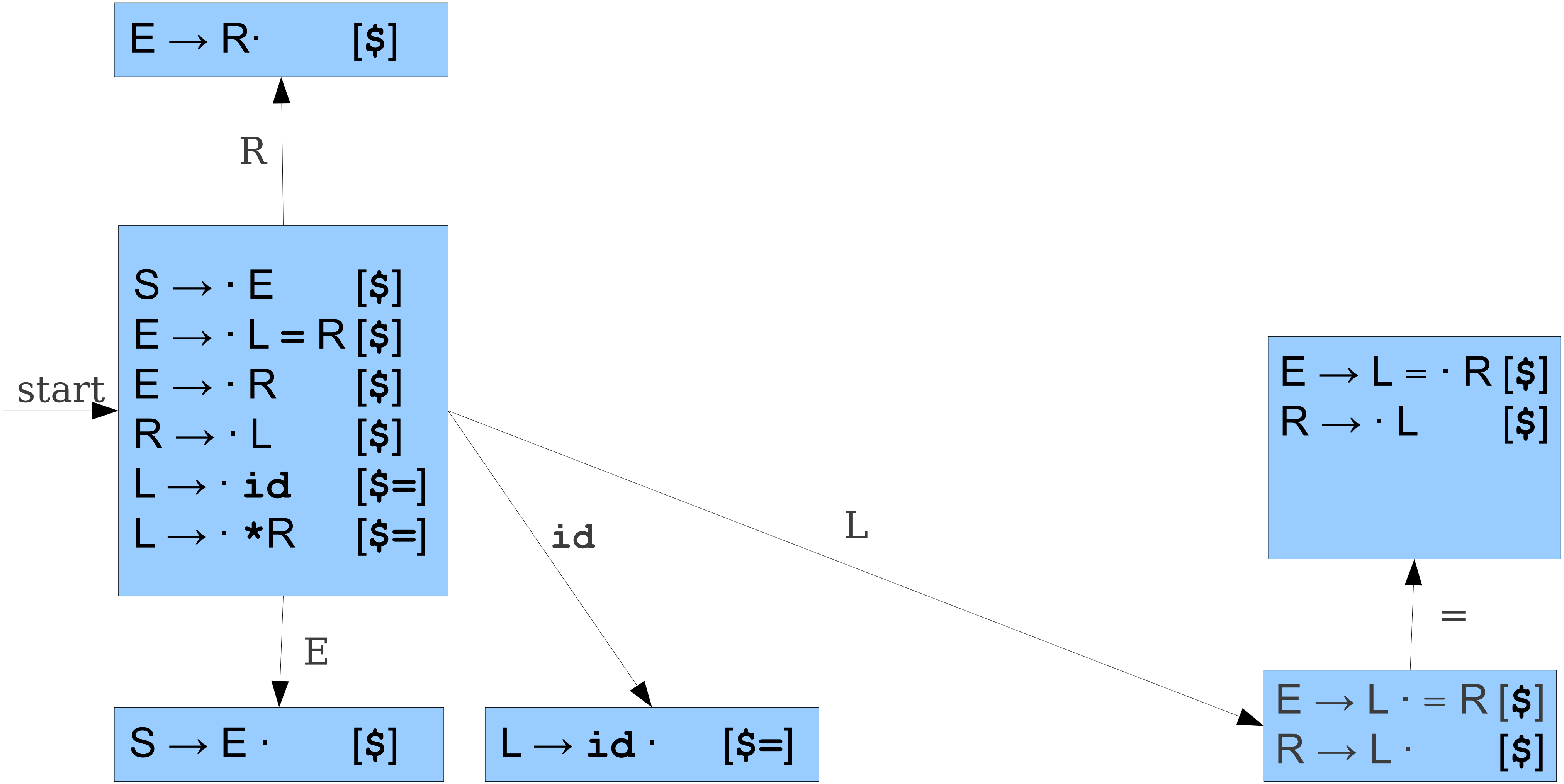
S → **E**
E → **L = R**
E → **R**
L → **id**
L → ***R**
R → **L**

LALR(1) Construction



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

LALR(1) Construction



E → R· [$]

R

start

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

E

id

L

S → E · [$]

L → **id** · [$=]

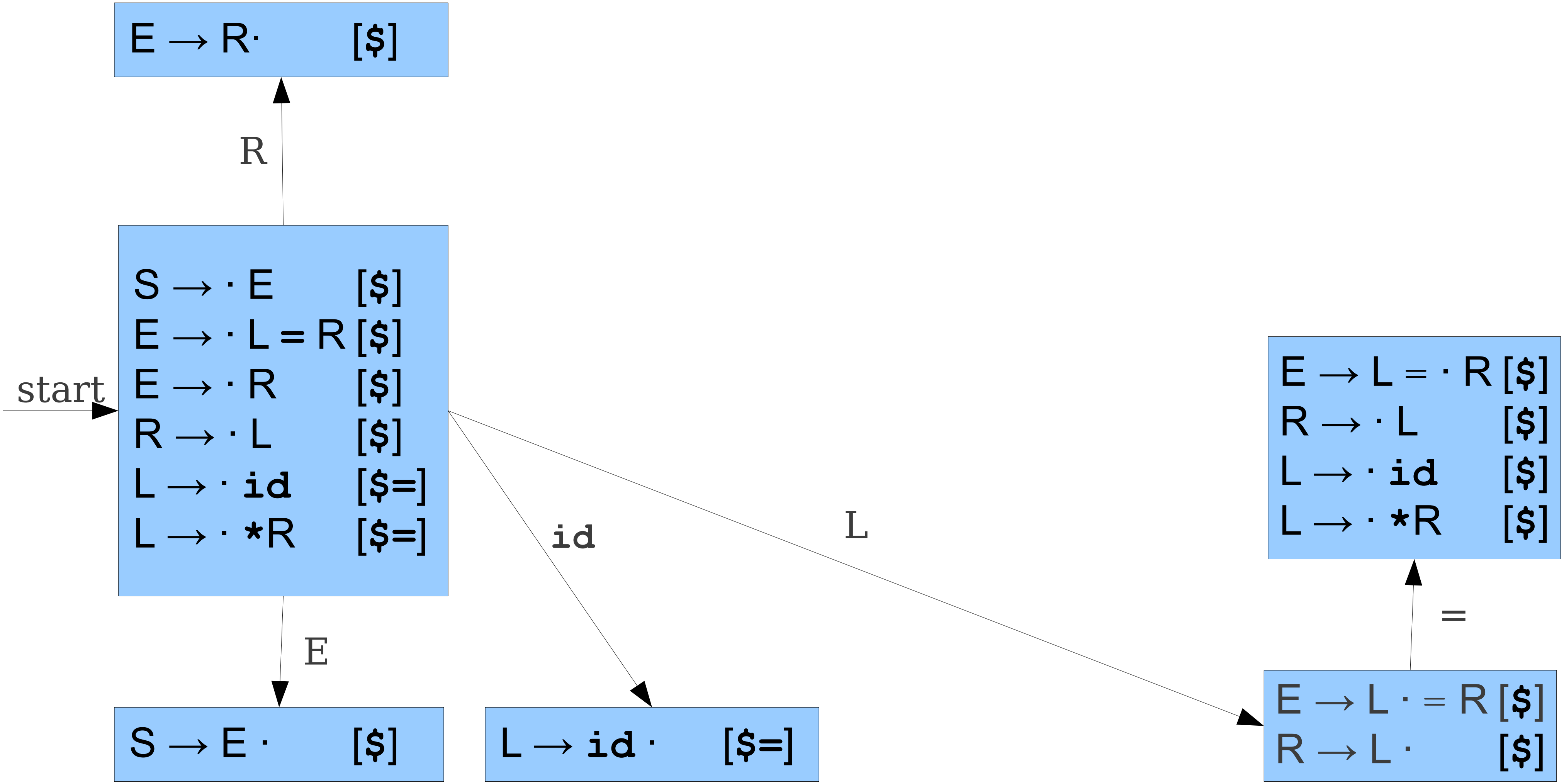
E → L · = R [$]
R → L · [$]

=

E → L = · R [$]
R → · L [$]

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

LALR(1) Construction



E → R· [$]

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

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

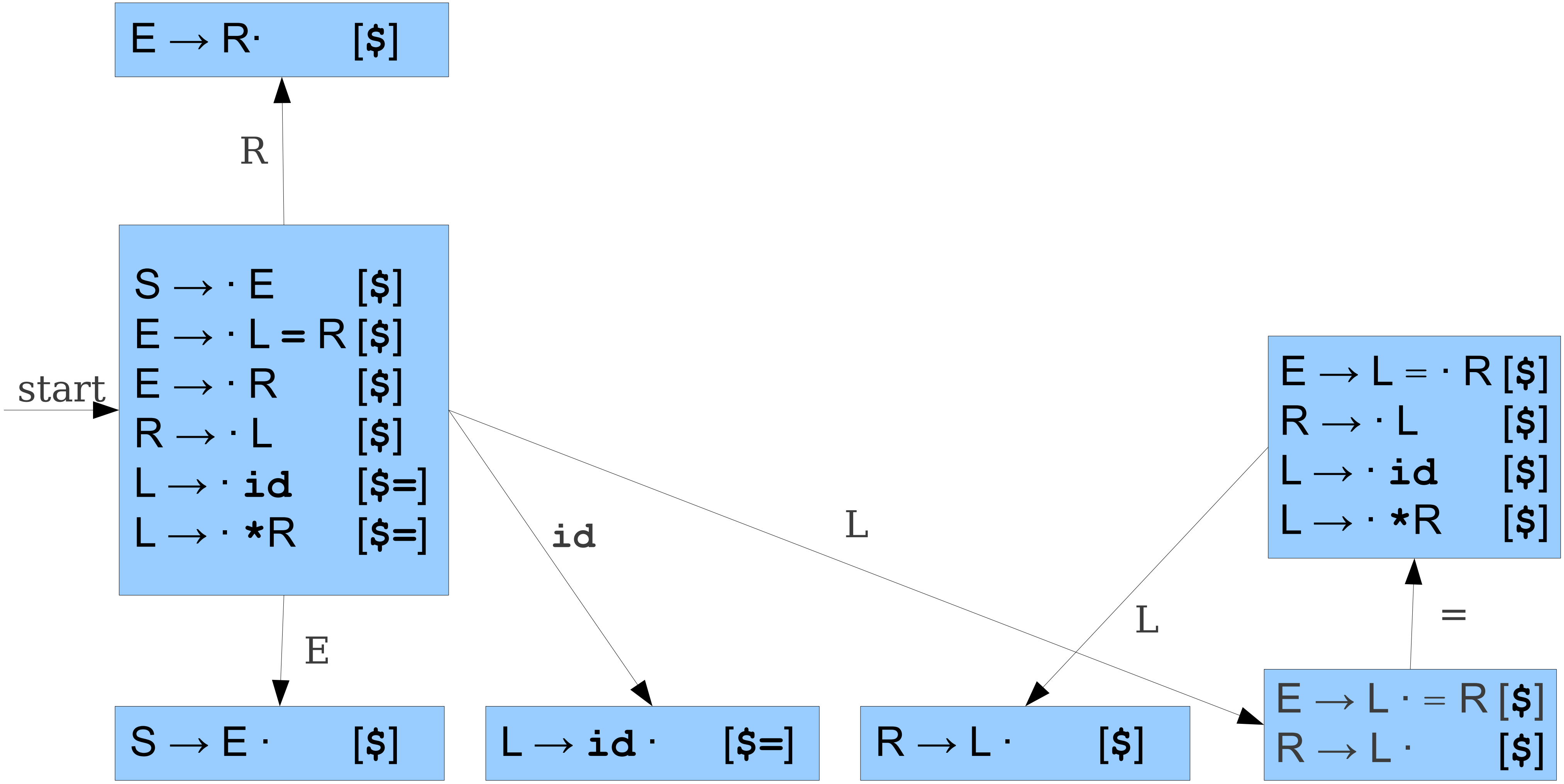
E → L · = R [$]
R → L · [$]

S → E · [$]

L → `id` · [$=]

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

LALR(1) Construction



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

E → R· [$]

S → E · [$]

L → id · [$=]

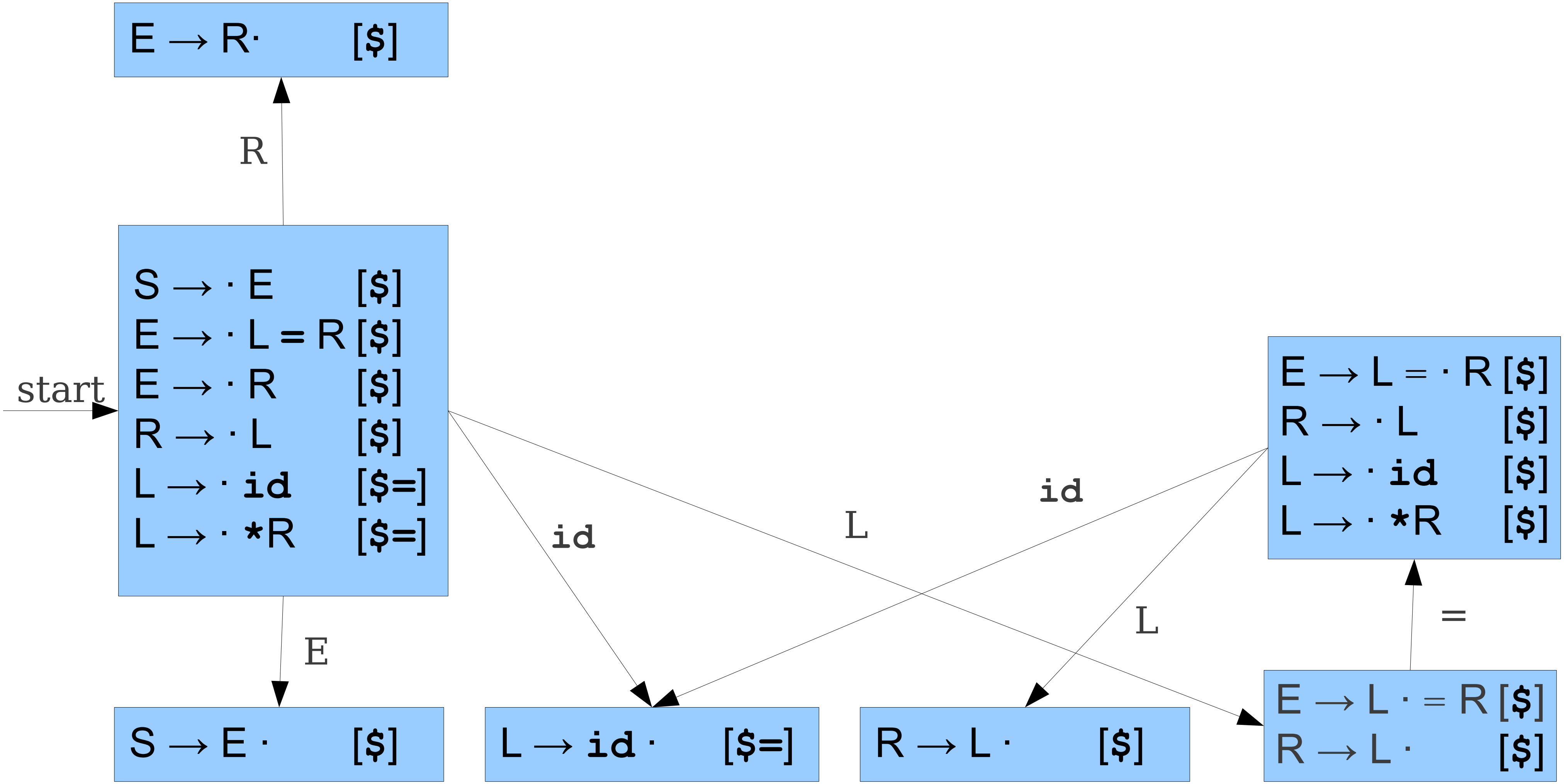
R → L · [$]

E → L · = R [$]
R → L · [$]

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

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

LALR(1) Construction



E → R· [$]

R

start

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

E

S → E · [$]

`id`

L → `id` · [$=]

L

`id`

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

L

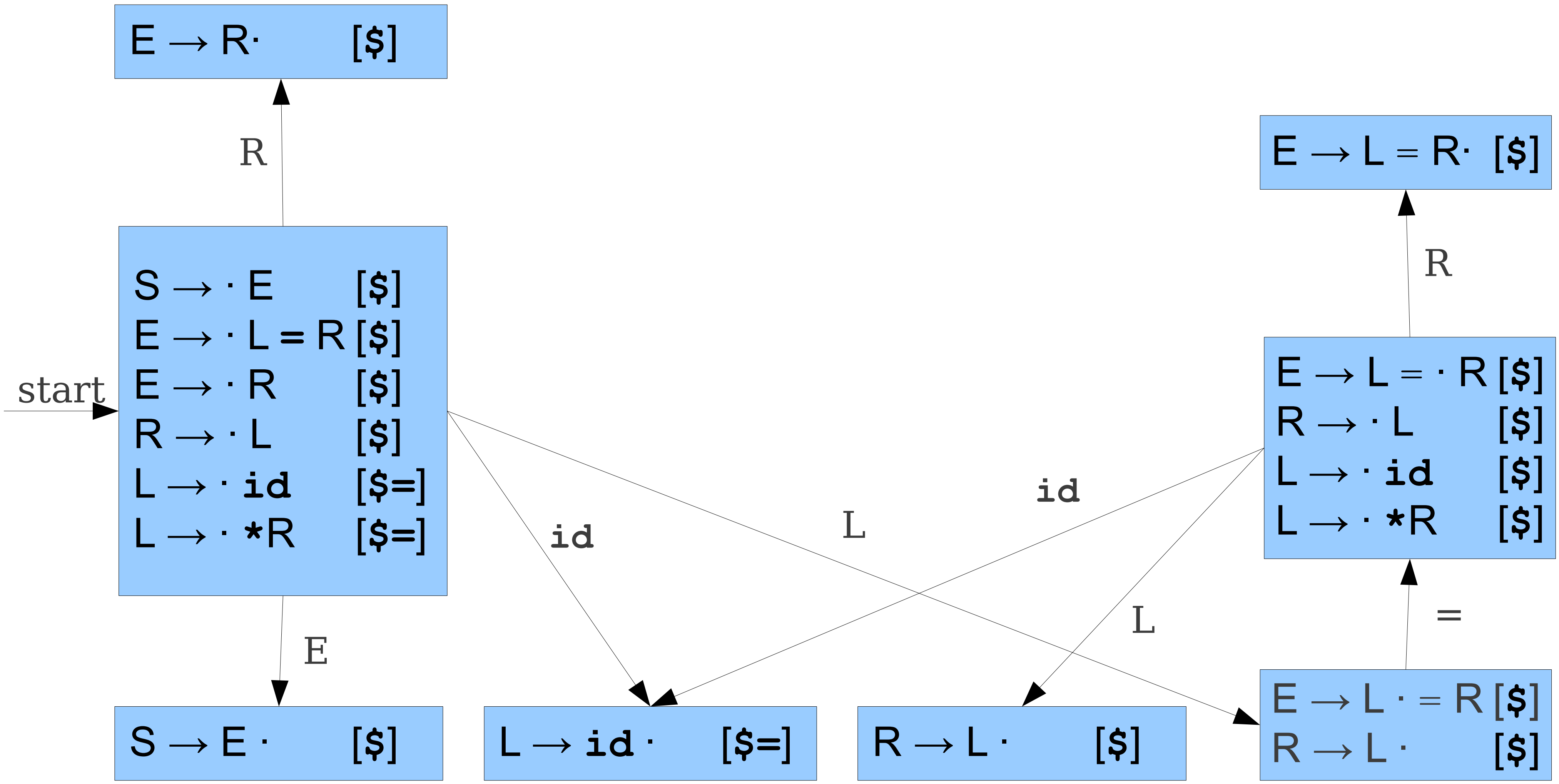
R → L · [$]

=

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

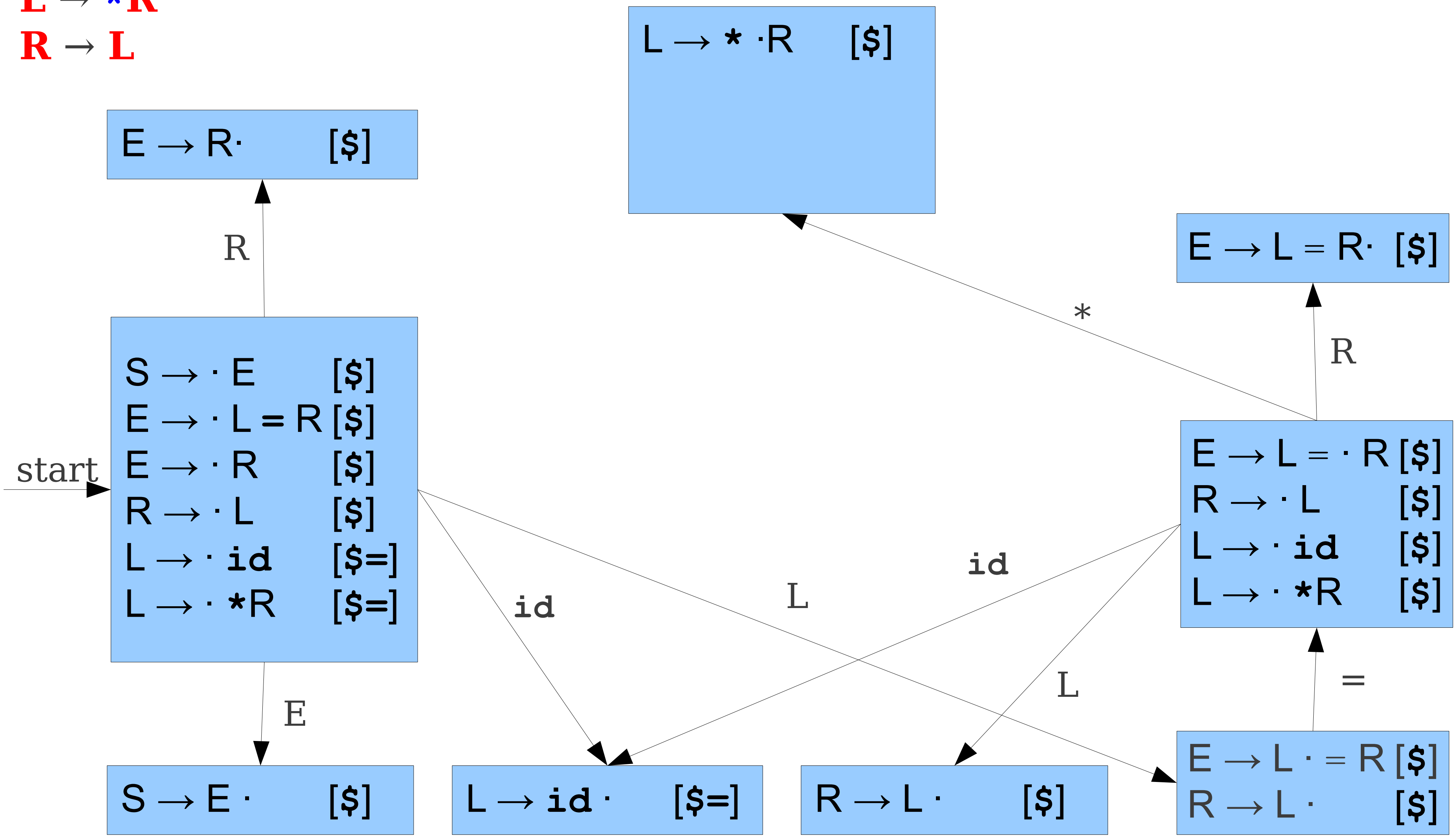
L → **`id`** · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



L → * ·R [$]

E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

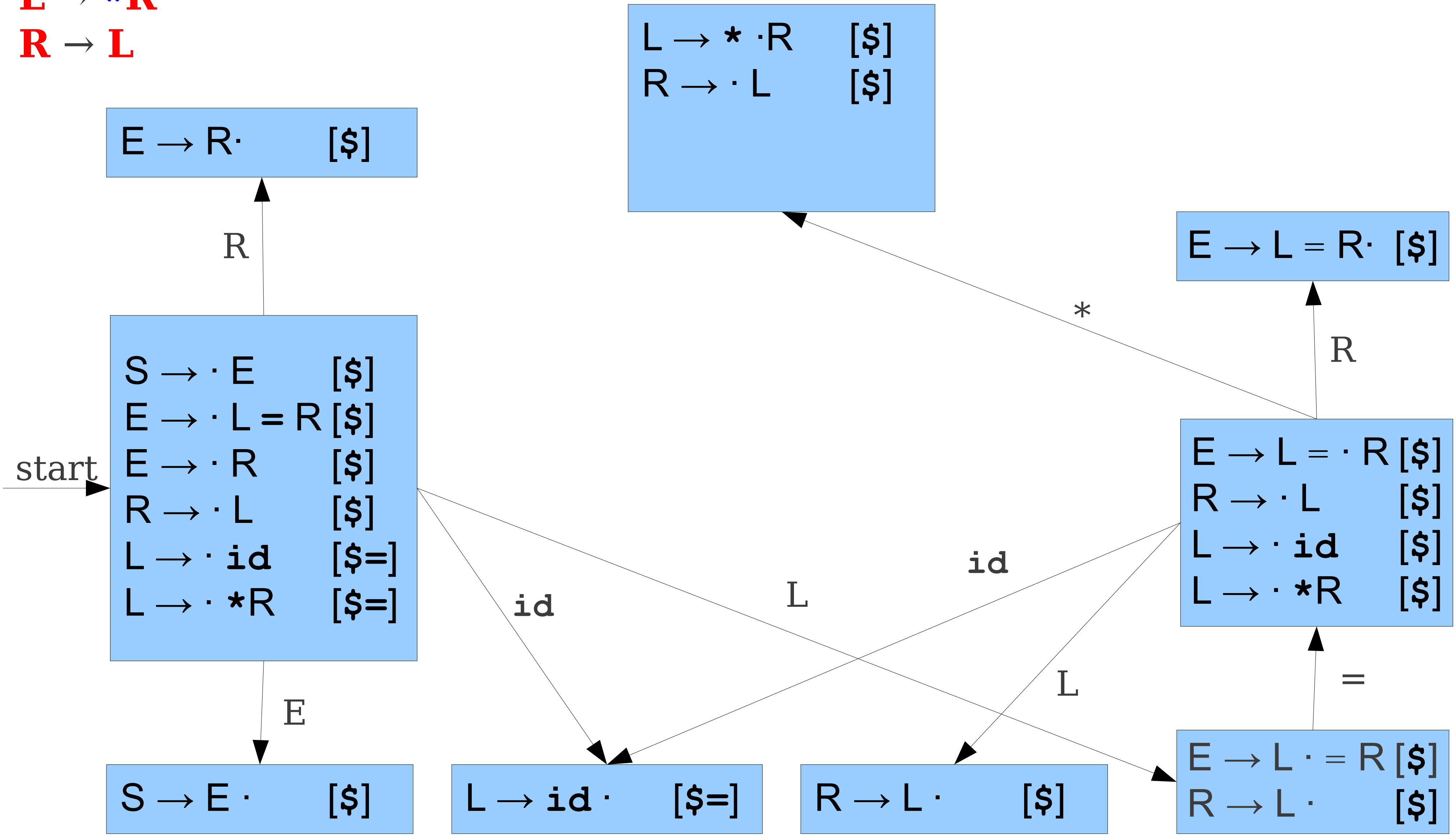
L → id · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



L → * ·R [$]
R → · L [$]

E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

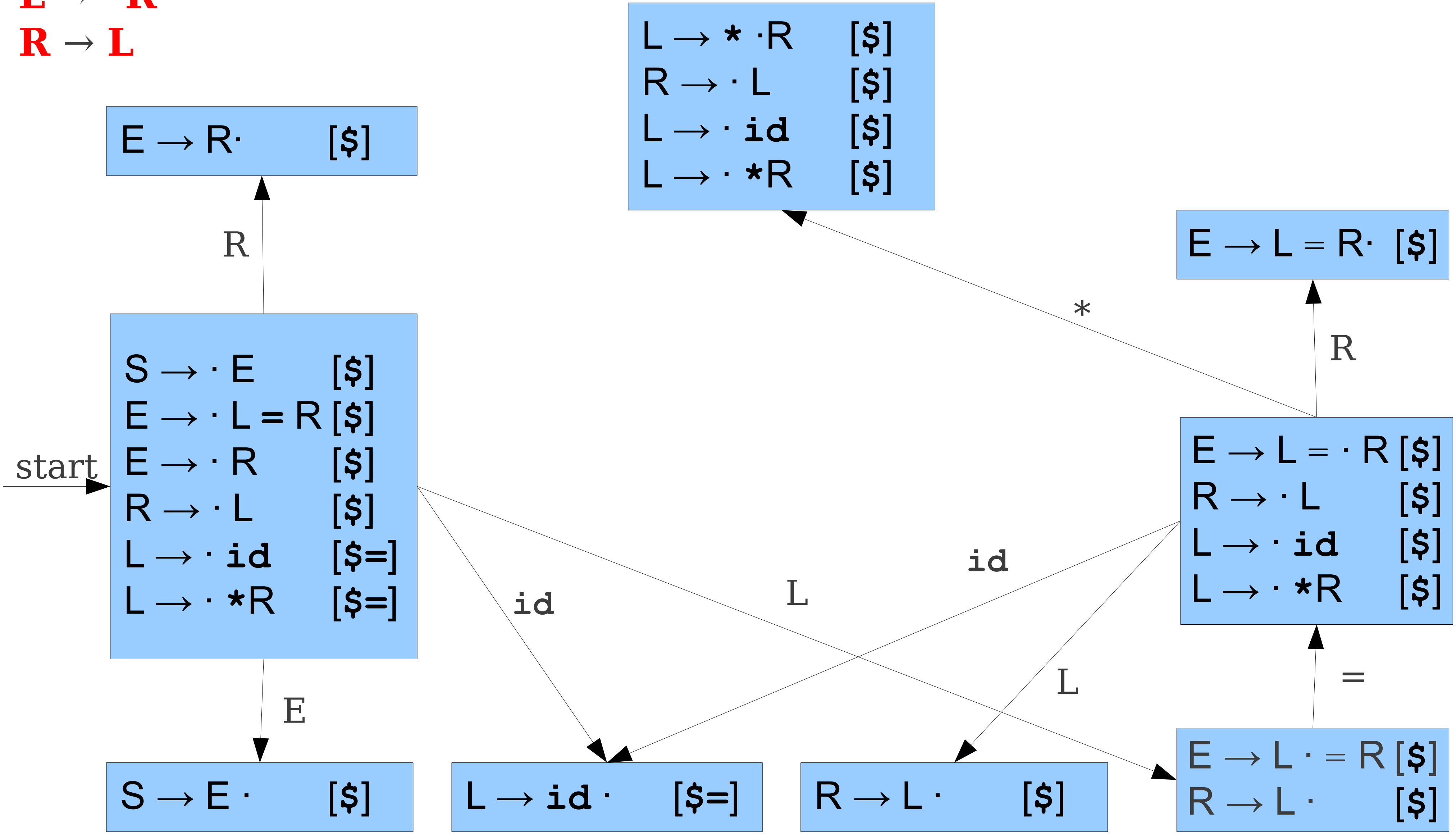
L → id · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



L → * ·R [$]
R → · L [$]
L → · **id** [$]
L → · ***R** [$]

E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

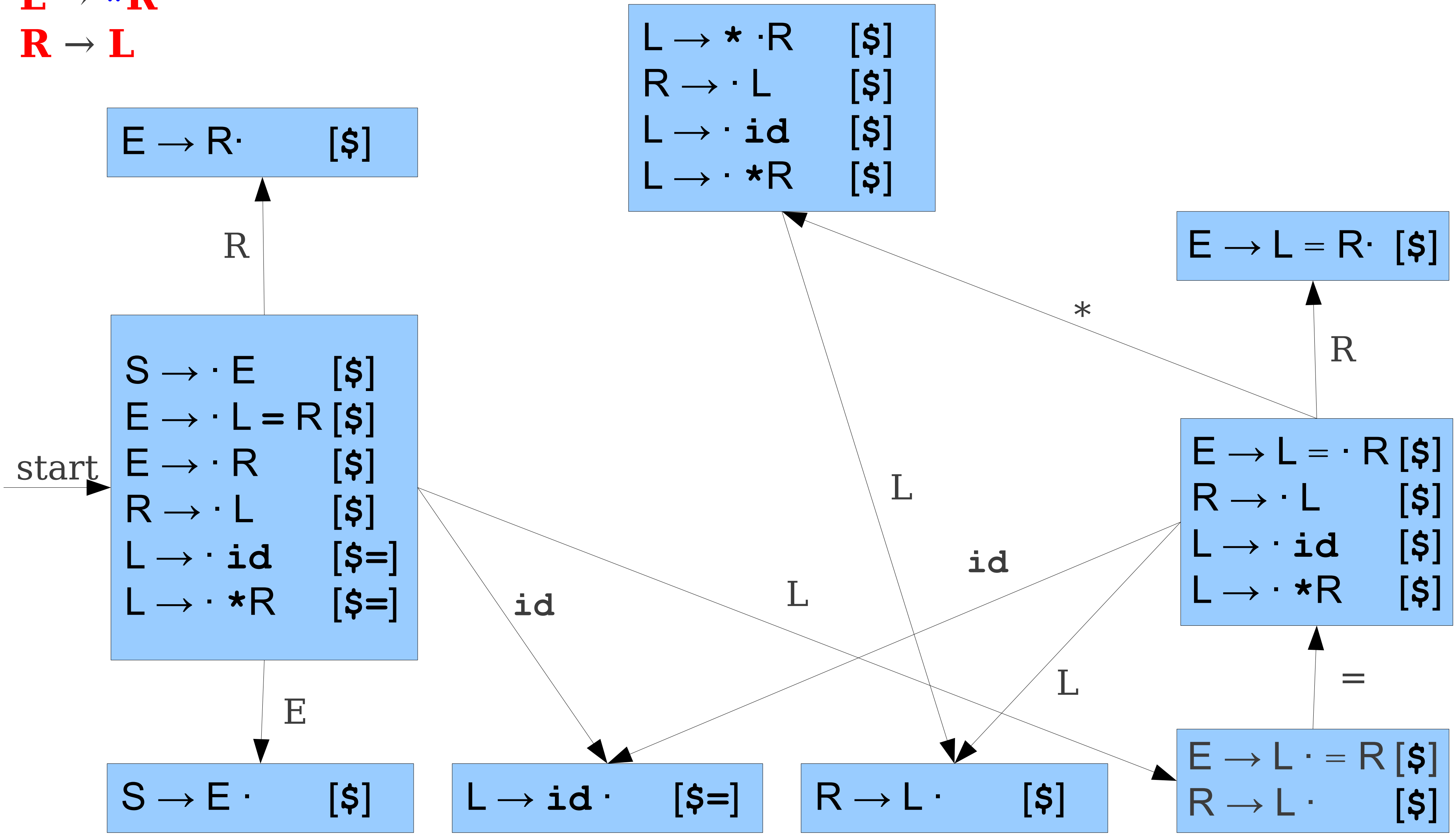
L → **id** · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



L → * ·R [$]
R → · L [$]
L → · **id** [$]
L → · ***R** [$]

E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

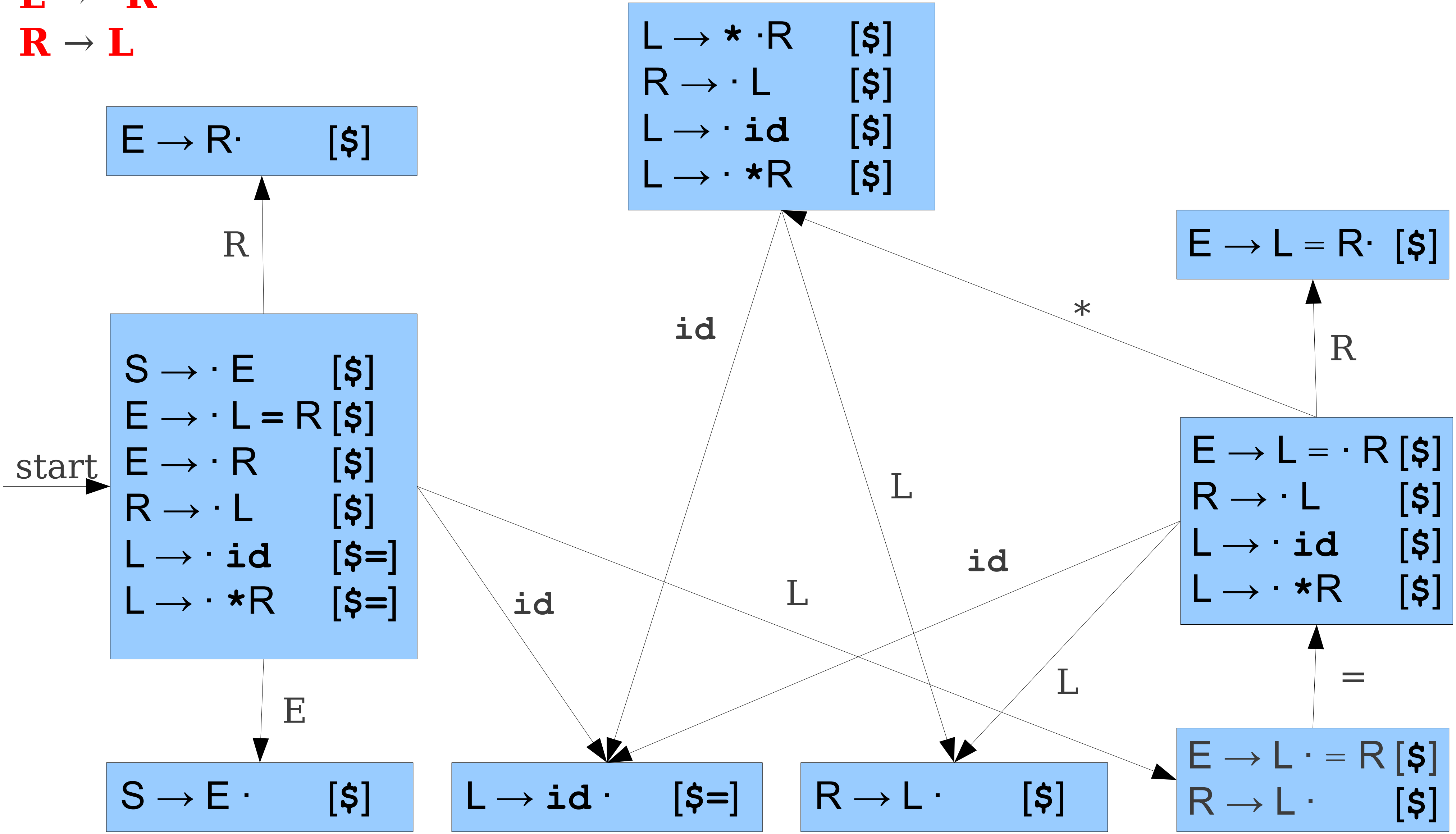
L → **id** · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



L → * ·R [$]
R → · L [$]
L → · id [$]
L → · *R [$]

E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

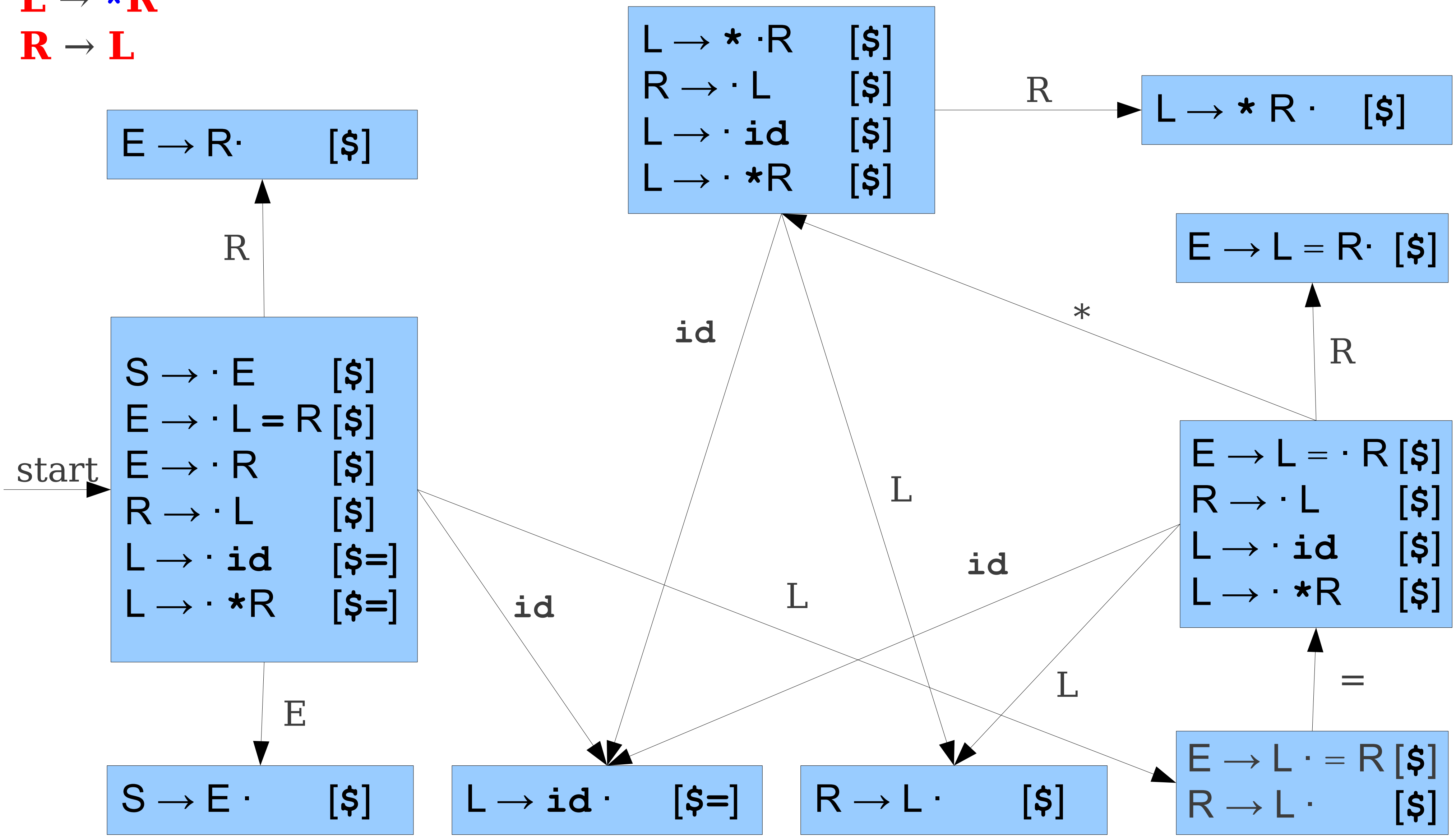
L → id · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



L → * ·R [$]
R → · L [$]
L → · **id** [$]
L → · *R [$]

L → * R · [$]

E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

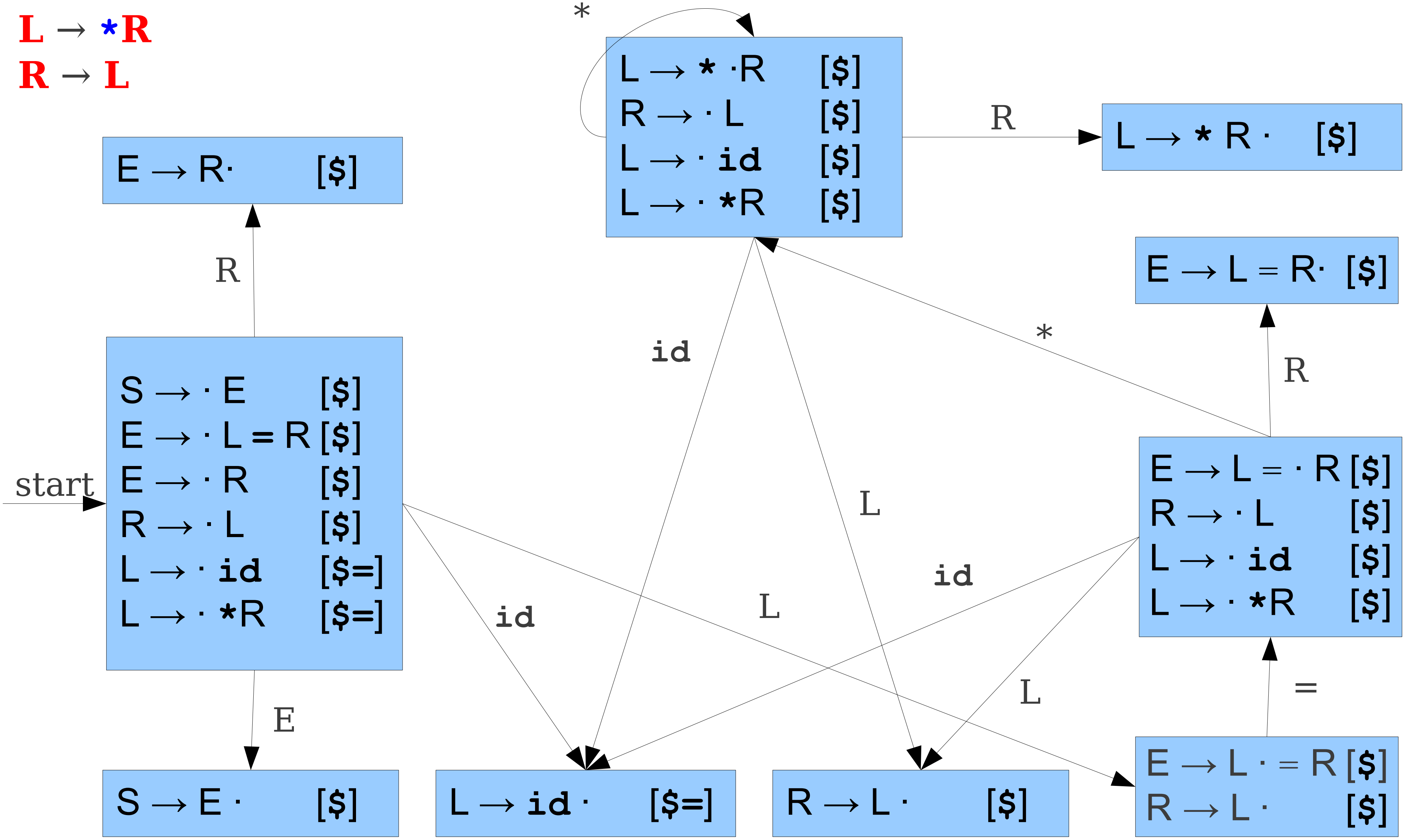
L → **id** · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



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

E → R· [$]

S → E · [$]

L → id · [$=]

L → * ·R [$]
R → · L [$]
L → · id [$]
L → · *R [$]

L → * R · [$]

R → L · [$]

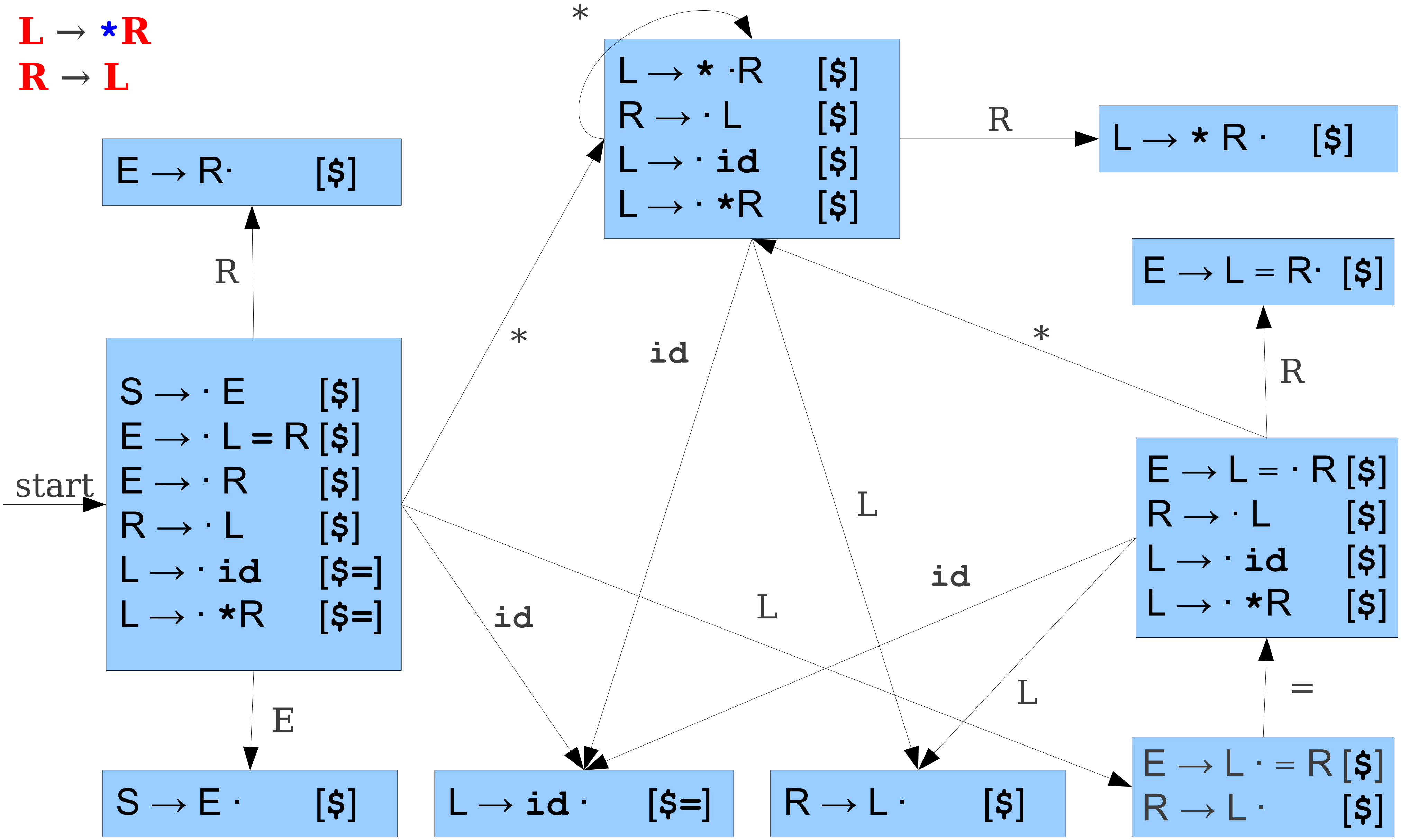
E → L = R· [$]

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

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



L → * ·R [$]
R → · L [$]
L → · id [$]
L → · *R [$]

L → * R · [$]

E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

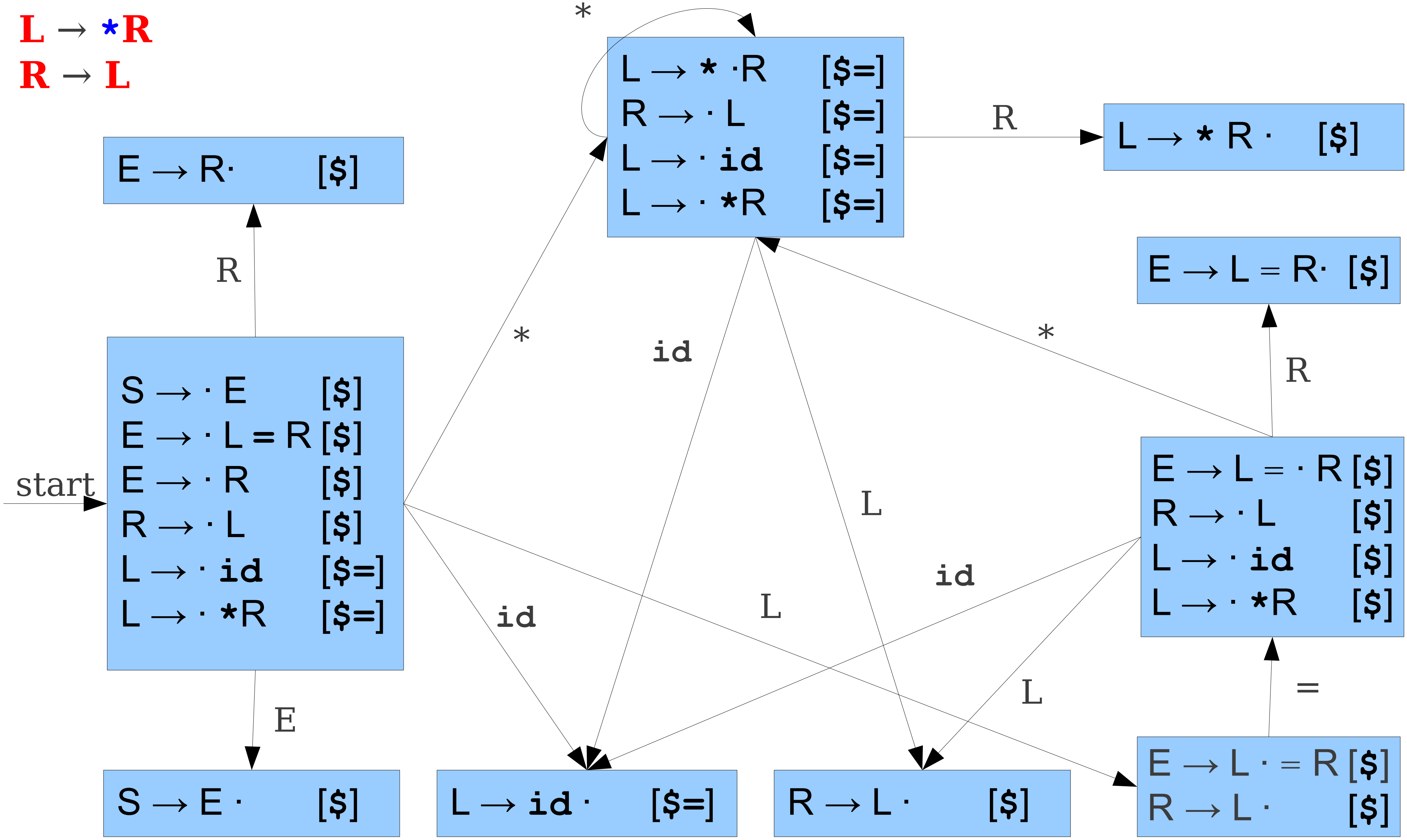
L → id · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

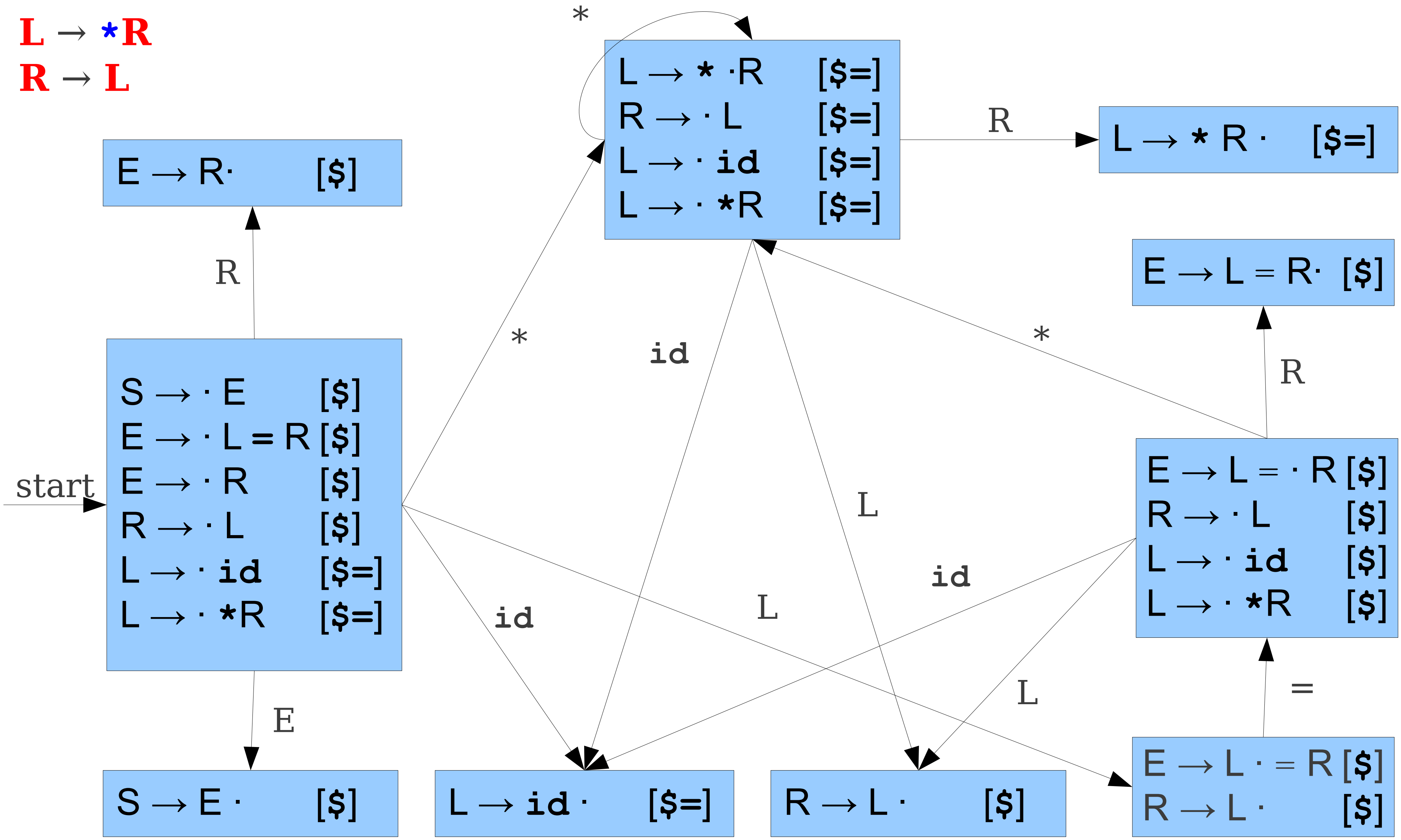
LALR(1) Construction

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



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

LALR(1) Construction



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

L → * R · [$=]

E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

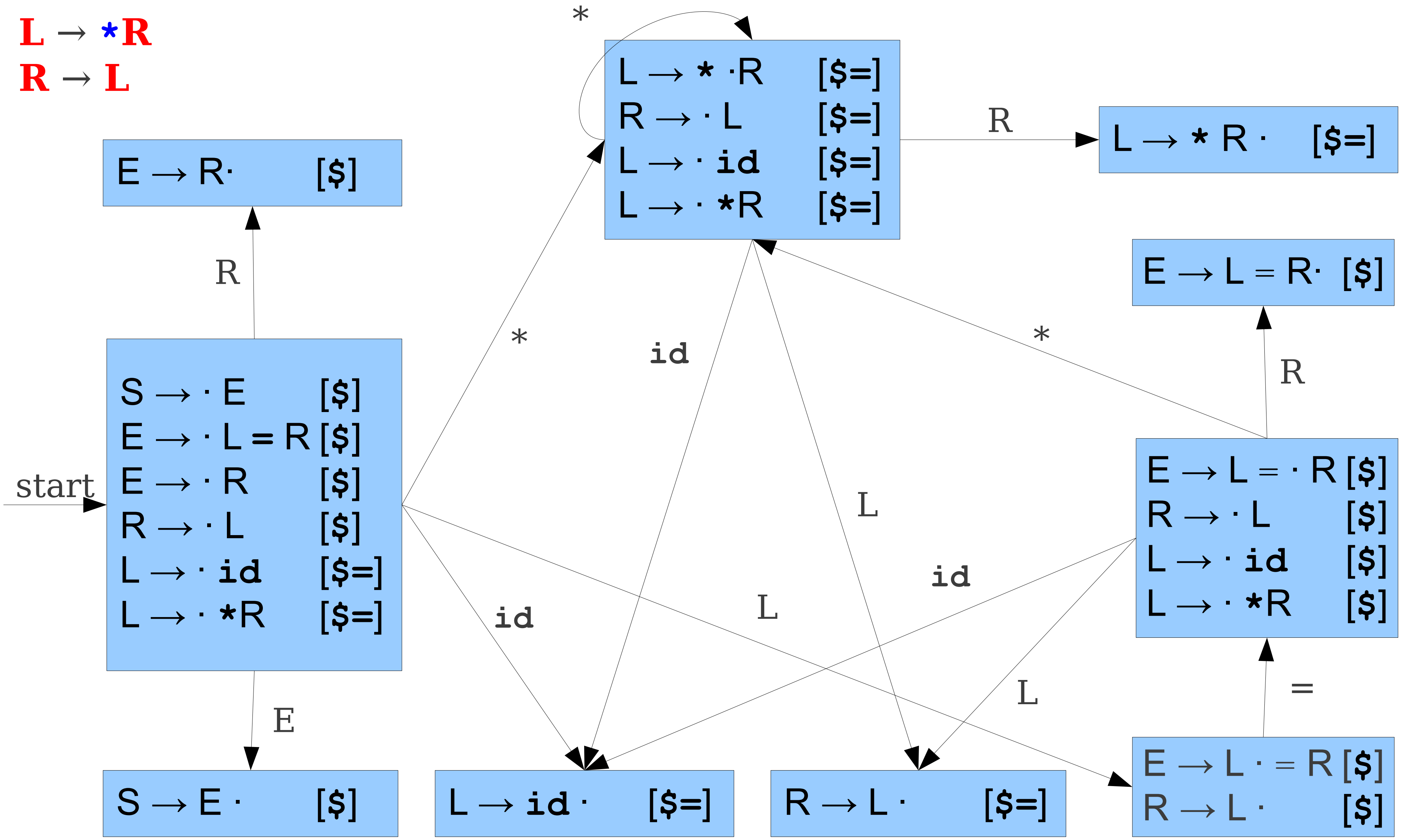
L → id · [$=]

R → L · [$]

E → L · = R [$]
R → L · [$]

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

LALR(1) Construction



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

L → * R · [$=]

E → R· [$]

E → L = R· [$]

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

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

S → E · [$]

L → **id** · [$=]

R → L · [$=]

E → L · = R [$]
R → L · [$]

Analysis of our Algorithm

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

SLR uses FOLLOW sets

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

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

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

LALR uses LA sets

- Given an LR(0) state q and a production $\mathbf{A} \to \boldsymbol{\gamma}$, the **lookahead set** $\text{LA}(q, \mathbf{A} \to \boldsymbol{\gamma})$ is defined as

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

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

LA and FOLLOW

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

  LA(q, **A** → $\boldsymbol{\gamma}$) = { `t` | **S** ⇒* $\boldsymbol{\alpha}$**A**`t`$\boldsymbol{\omega}$ and $\boldsymbol{\alpha\gamma}$ reaches q }

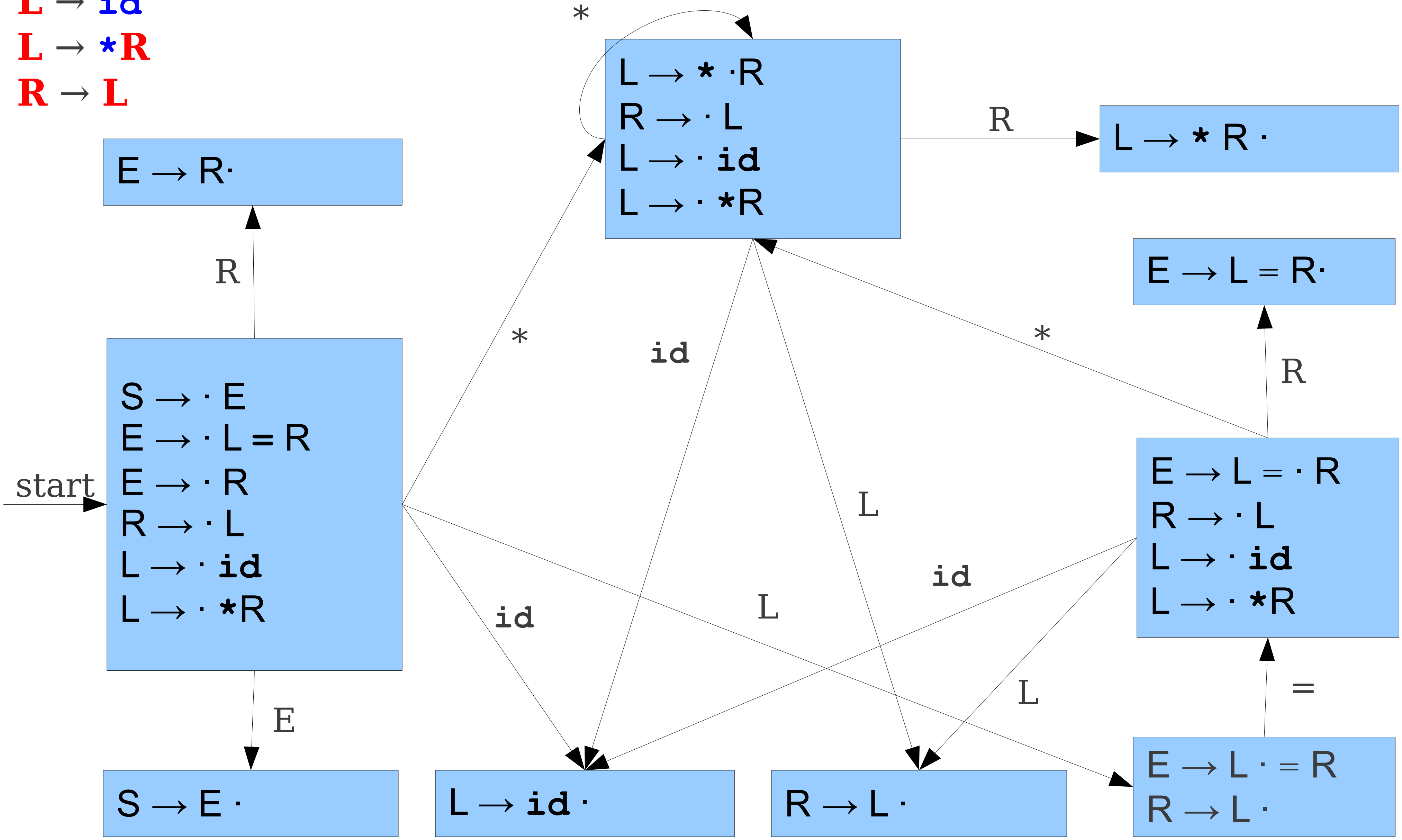
- The **follow set** FOLLOW(**A**) is defined as

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

- Note that LA(q, **A** → $\boldsymbol{\gamma}$) ⊆ FOLLOW(**A**); that is, LA sets are "more precise" than FOLLOW sets.
- If we can compute LA from FOLLOW, we can construct a LALR(1) parser efficiently.

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

An LR(0) Automaton



L → * ·R
R → · L
L → · id
L → · *R

L → * R ·

E → R·

E → L = R·

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

E → L = · R
R → · L
L → · id
L → · *R

S → E ·

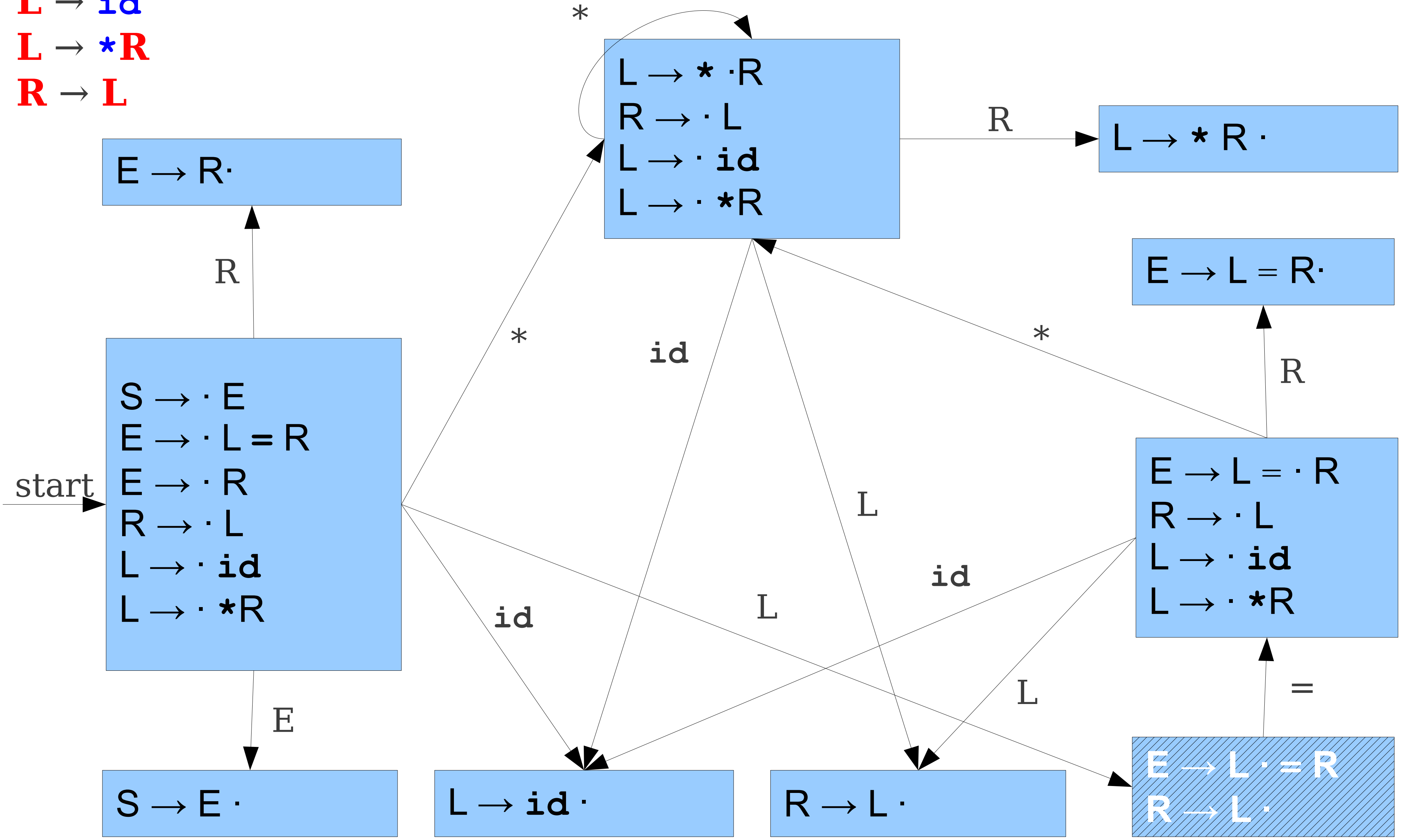
L → id ·

R → L ·

E → L · = R
R → L ·

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

An LR(0) Automaton



L → * ·R
R → · L
L → · **id**
L → · *R

L → * R ·

E → R·

E → L = R·

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

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

S → E ·

L → **id** ·

R → L ·

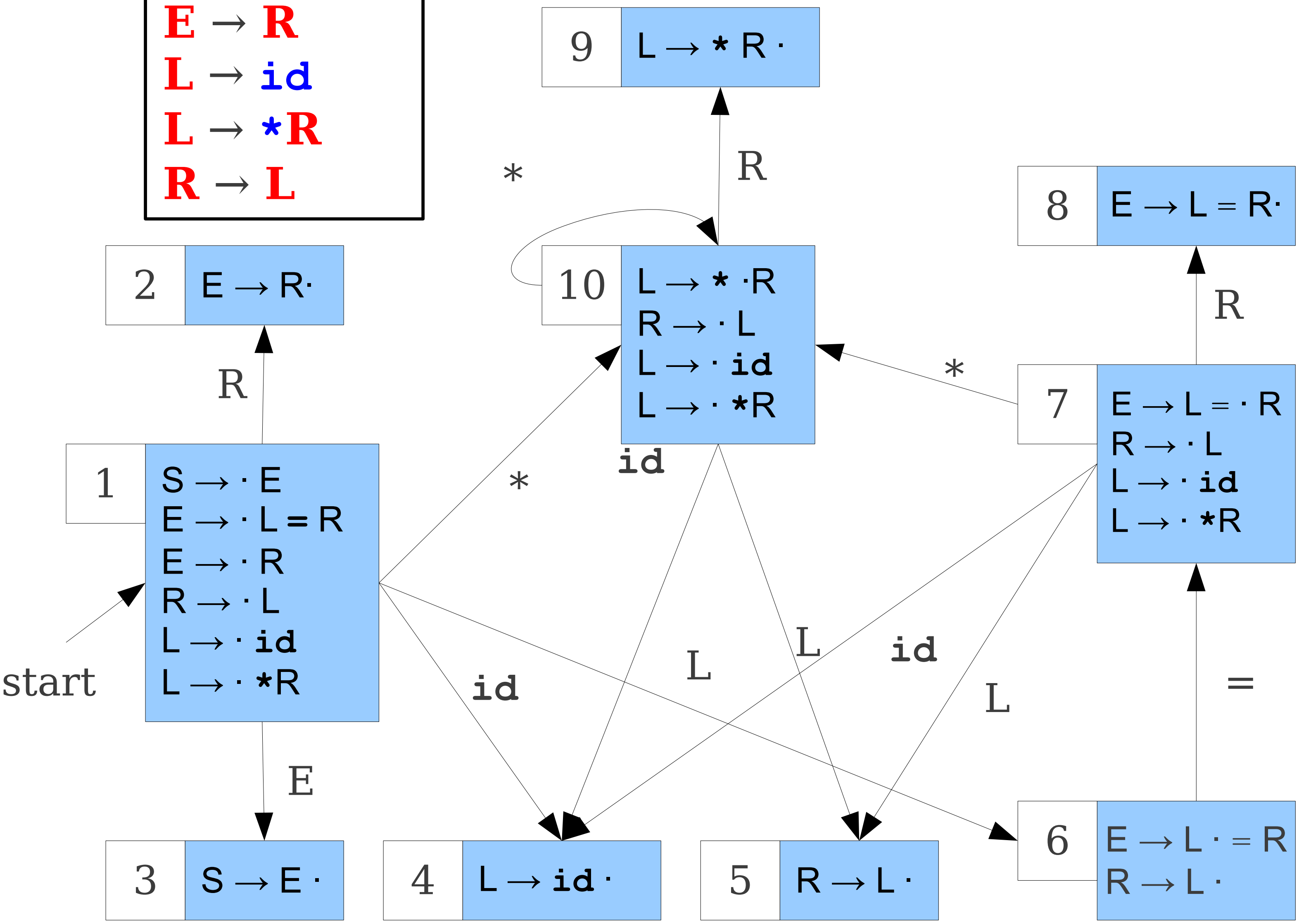
E → L · = R
R → L ·

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

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

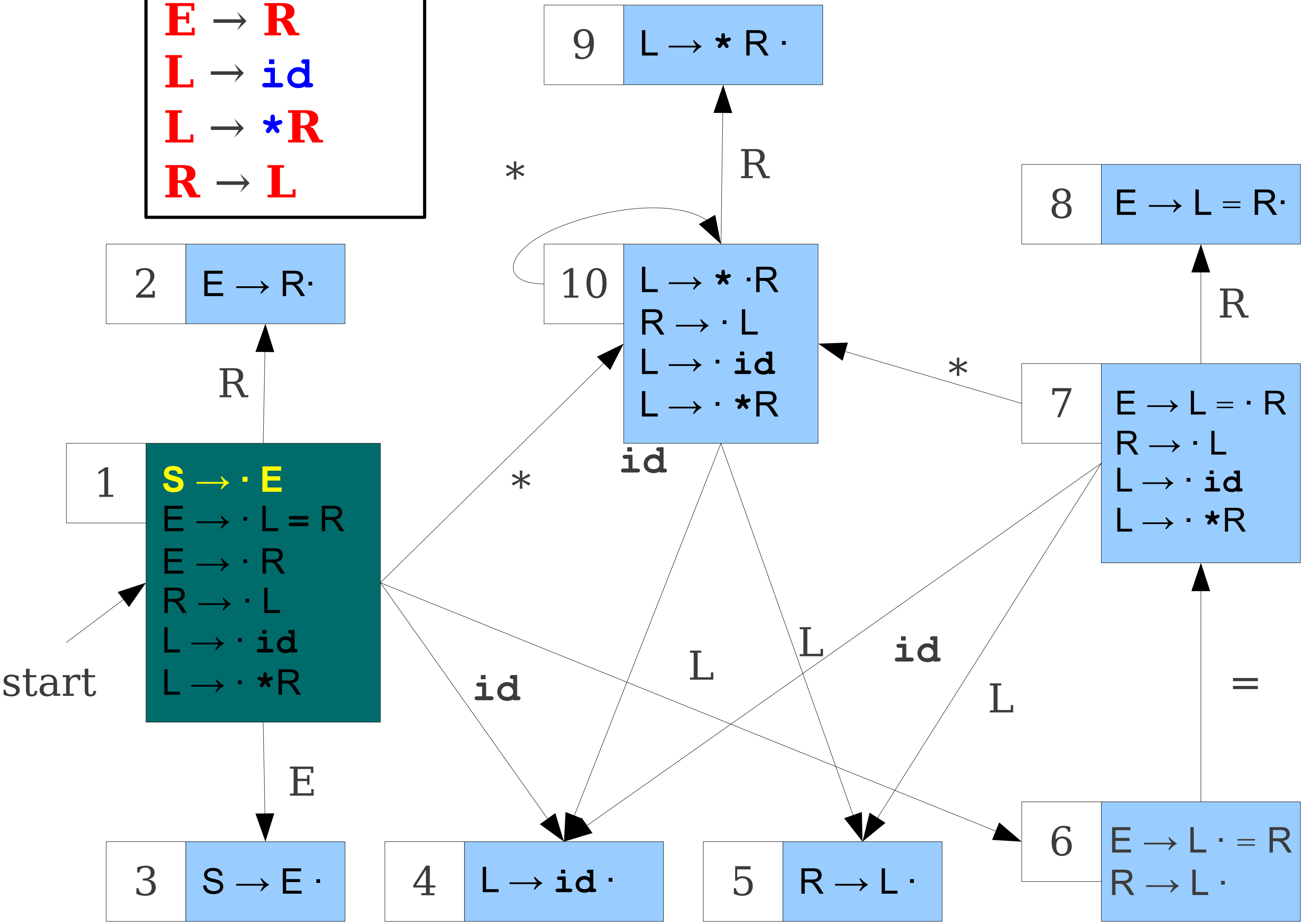
Augmenting the Grammar

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



Augmenting the Grammar

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



1
S → · E
E → · L = R
E → · R
R → · L
L → · **id**
L → · ***R**

2
E → R·

3
S → E ·

4
L → **id** ·

5
R → L ·

6
E → L · = R
R → L ·

7
E → L = · R
R → · L
L → · **id**
L → · ***R**

8
E → L = R·

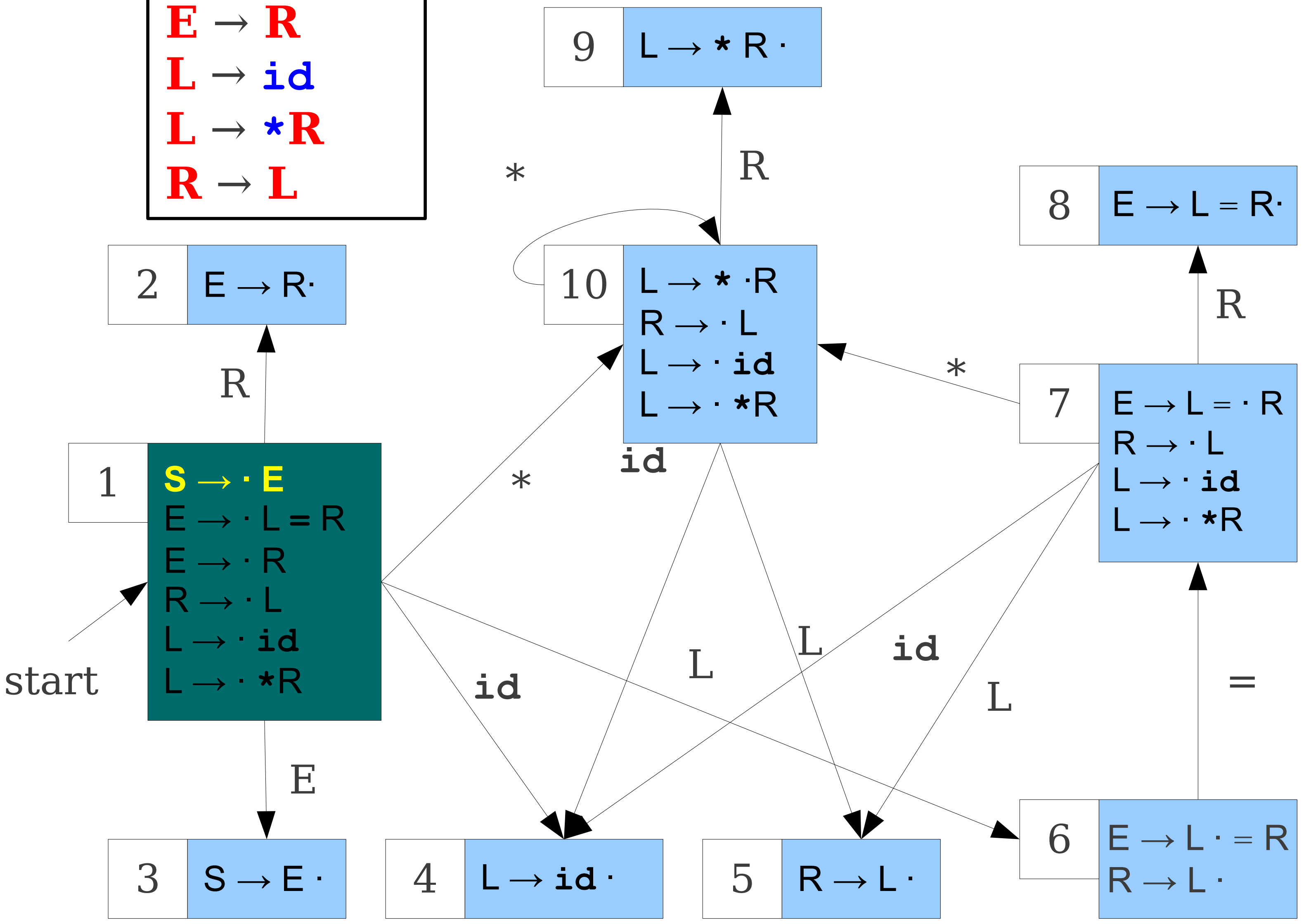
9
L → ***** R ·

10
L → ***** ·R
R → · L
L → · **id**
L → · ***R**

Augmenting the Grammar

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

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



9
L → * R ·

8
E → L = R·

10
L → * ·R
R → · L
L → · **id**
L → · *R

2
E → R·

7
E → L = · R
R → · L
L → · **id**
L → · *R

1
S → · E
E → · L = R
E → · R
R → · L
L → · **id**
L → · *R

6
E → L · = R
R → L ·

3
S → E ·

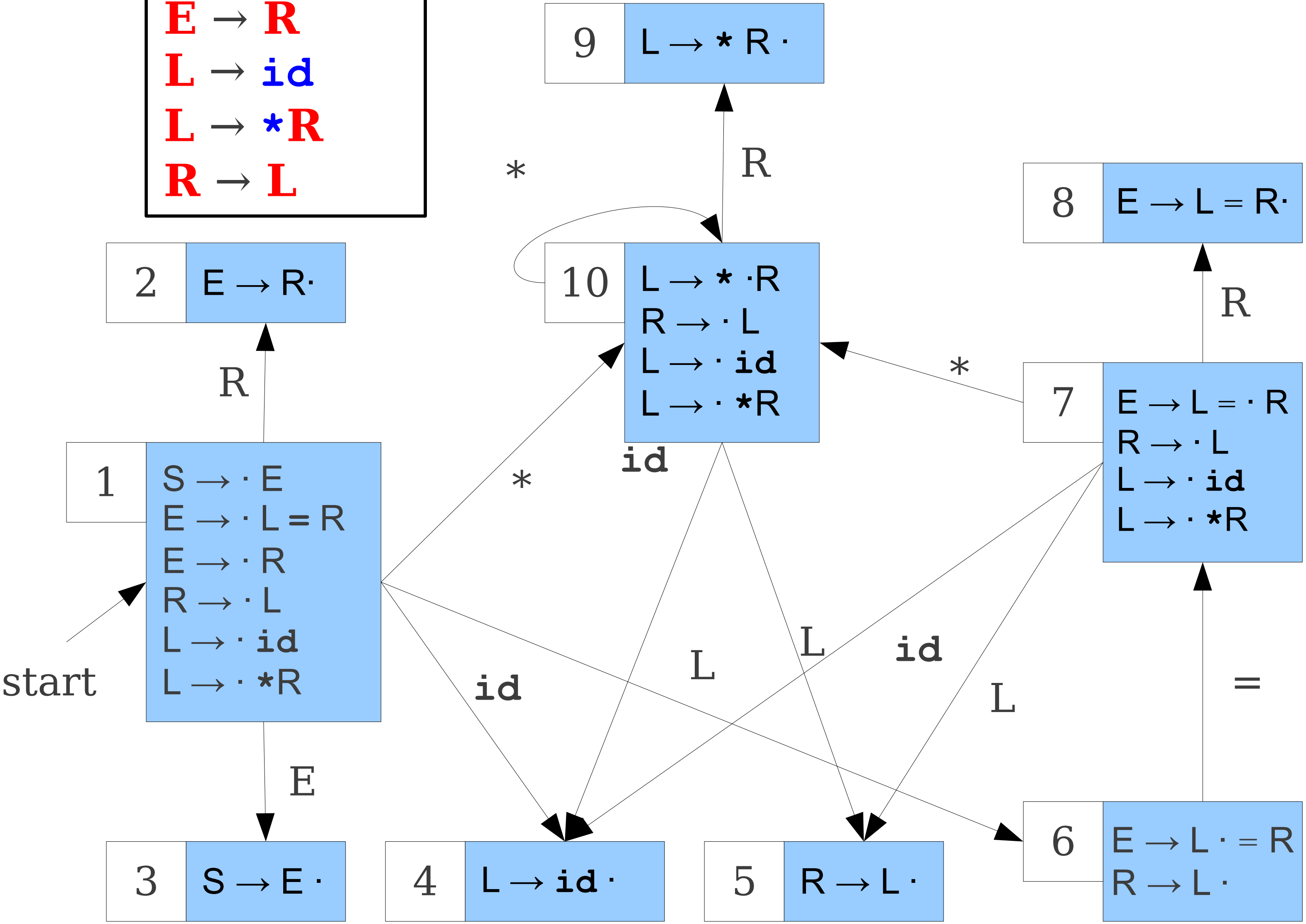
4
L → **id** ·

5
R → L ·

Augmenting the Grammar

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

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



9 L → ***** R ·

8 E → L = R·

2 E → R·

10
L → ***** ·R
R → · L
L → · **id**
L → · *****R

7
E → L = · R
R → · L
L → · **id**
L → · *****R

1
S → · E
E → · L = R
E → · R
R → · L
L → · **id**
L → · *****R

3 S → E ·

4 L → **id** ·

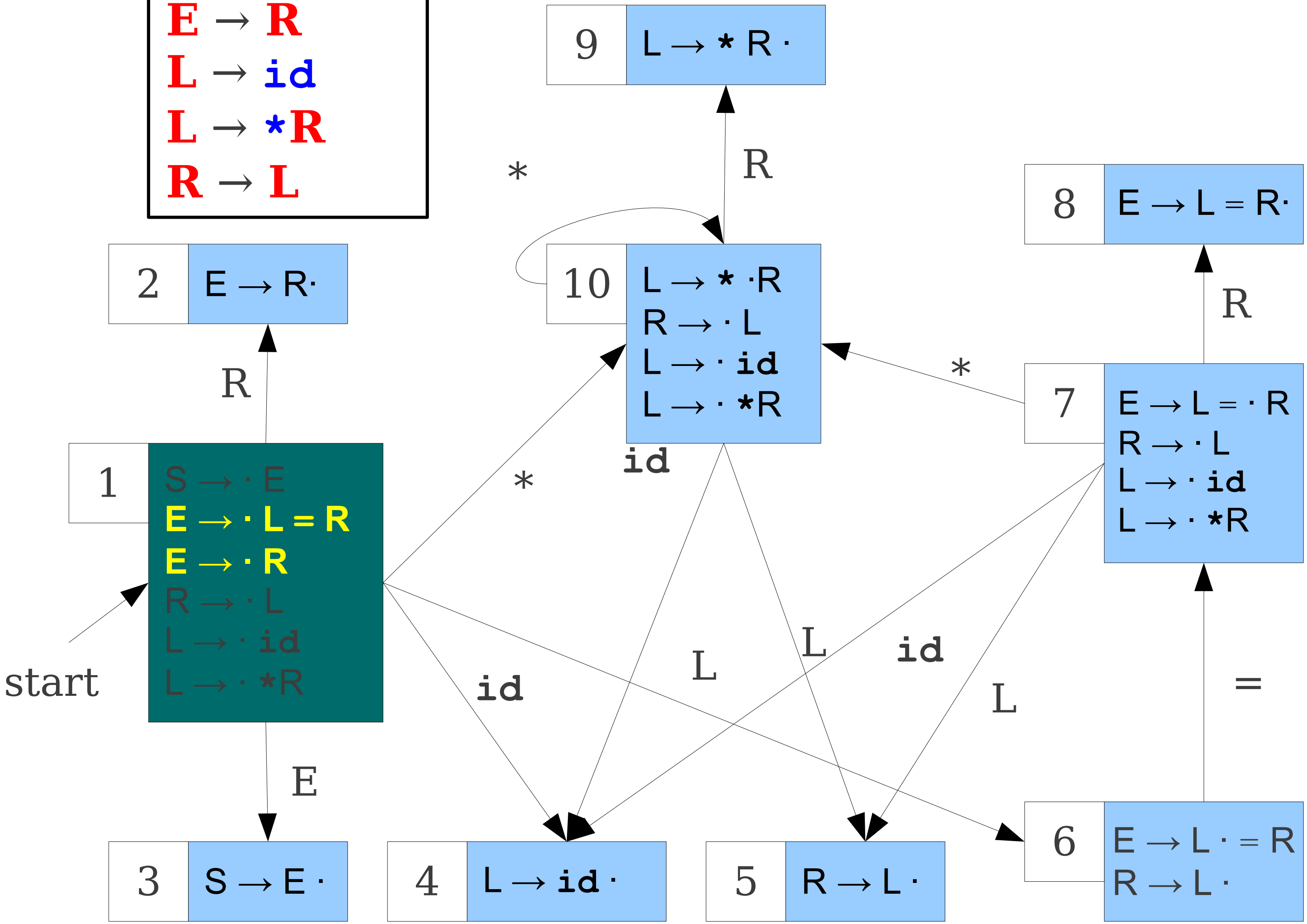
5 R → L ·

6
E → L · = R
R → L ·

Augmenting the Grammar

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

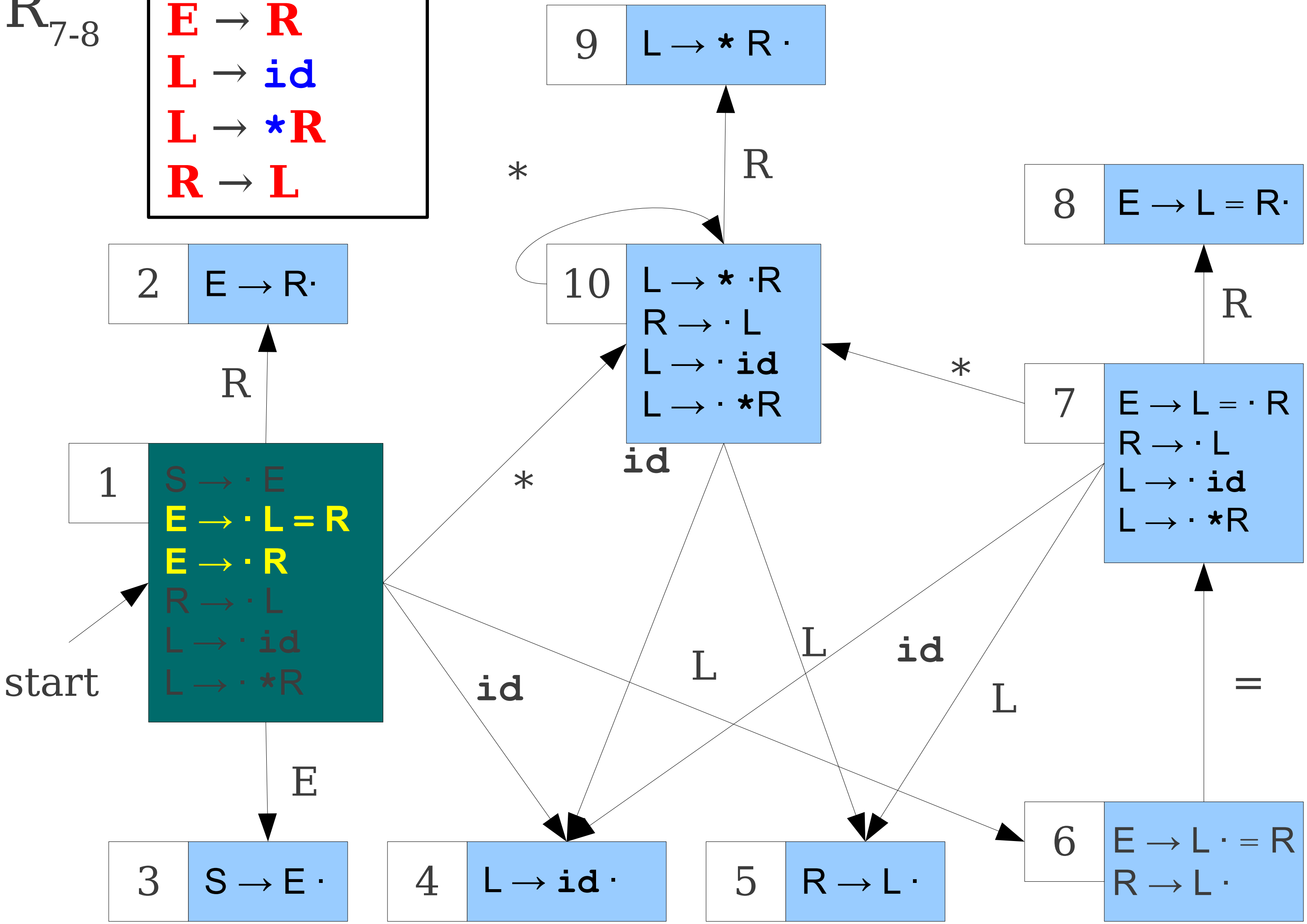
S → **E**
E → **L** **=** **R**
E → **R**
L → **`id`**
L → ***R**
R → **L**



Augmenting the Grammar

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

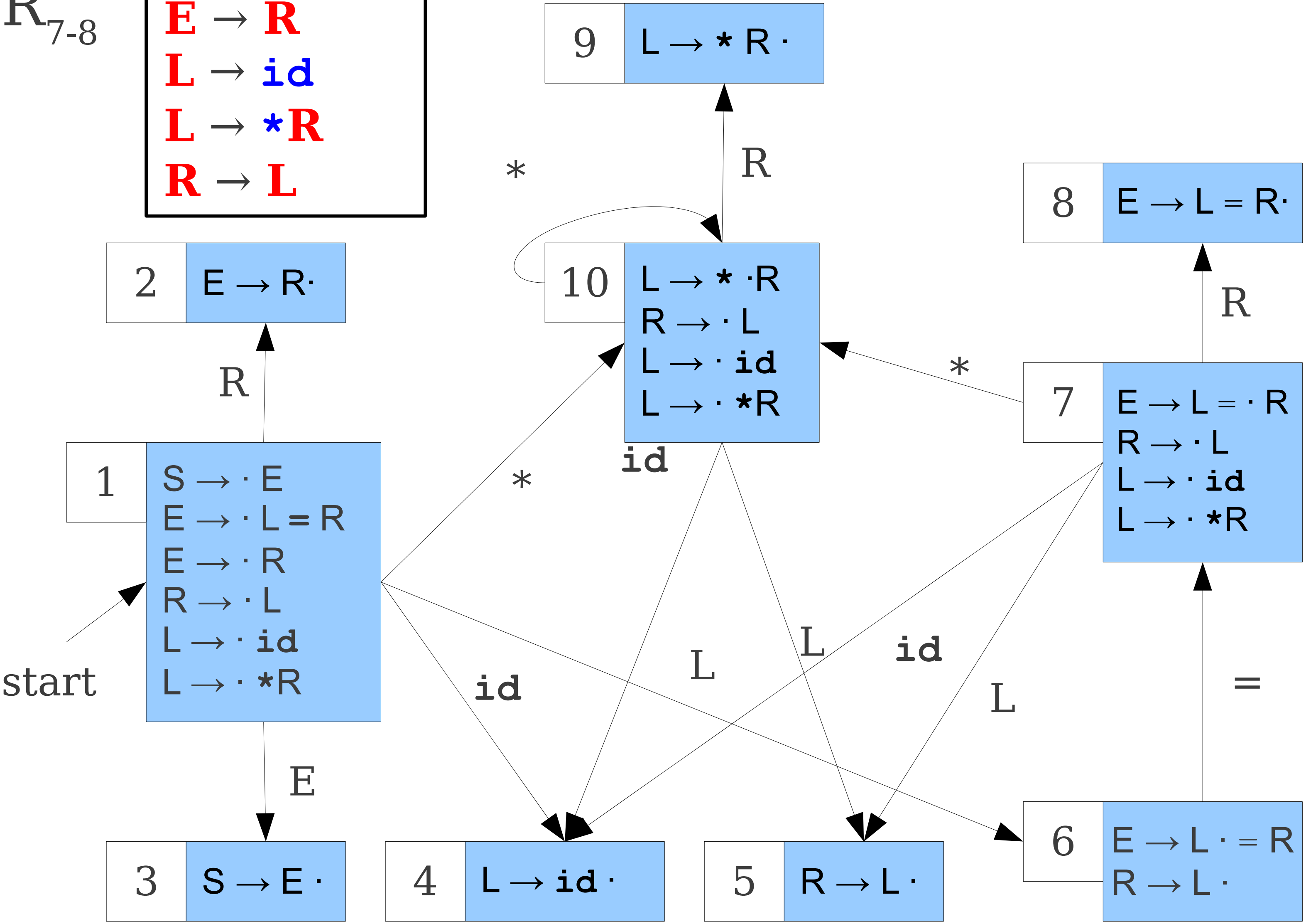
S → **E**
E → **L** **=** **R**
E → **R**
L → **`id`**
L → ***R**
R → **L**



Augmenting the Grammar

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

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



1
S → · E
E → · L = R
E → · R
R → · L
L → · **id**
L → · *****R

2
E → R·

3
S → E ·

4
L → **id** ·

5
R → L ·

6
E → L · = R
R → L ·

7
E → L = · R
R → · L
L → · **id**
L → · *****R

8
E → L = R·

9
L → ***** R ·

10
L → ***** ·R
R → · L
L → · **id**
L → · *****R

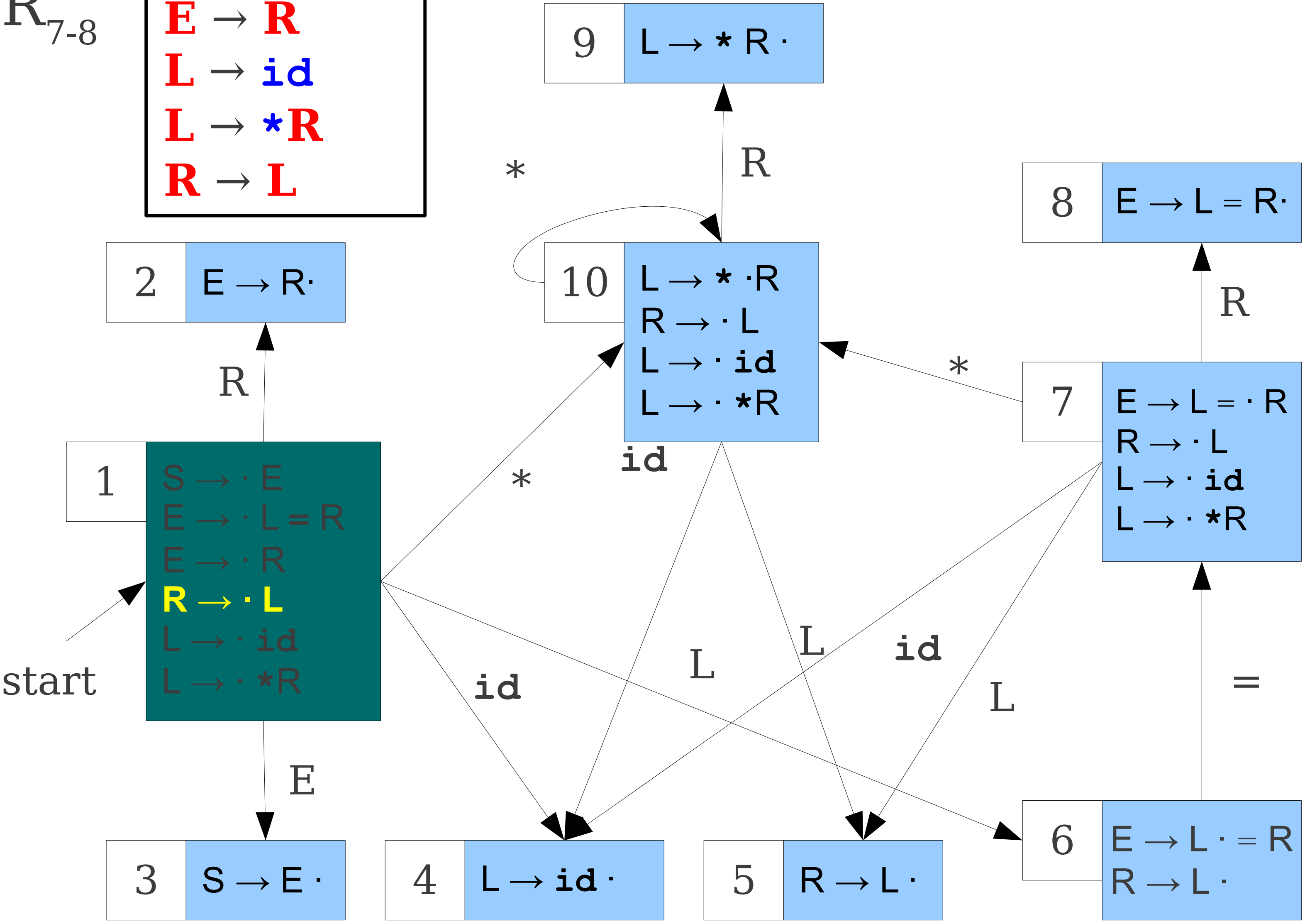
Augmenting the Grammar

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

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

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

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



Augmenting the Grammar

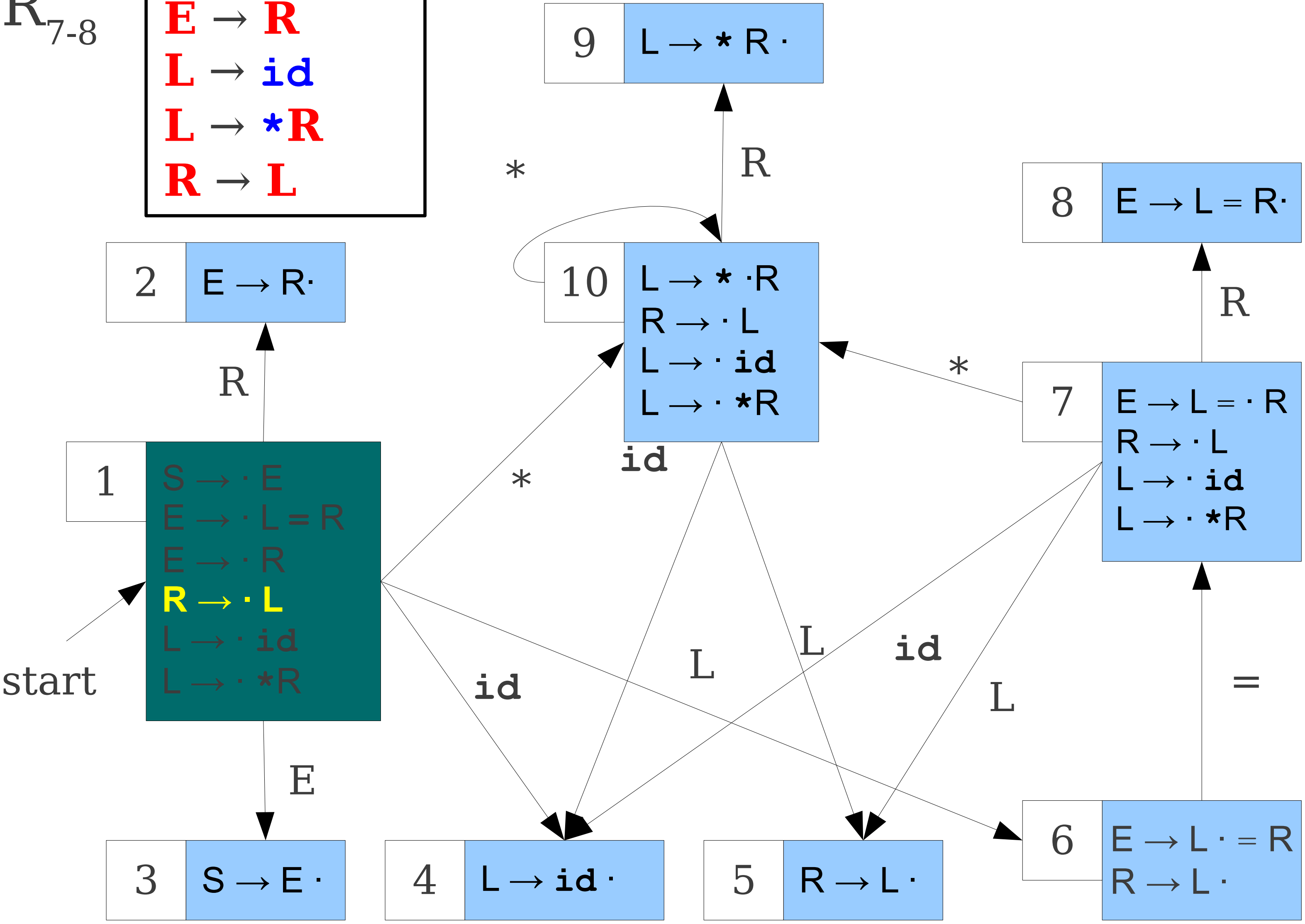
$S_1 \to E_{1\text{-}3}$

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

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

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

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



9 L → * R ·

2 E → R·

10 L → * ·R
R → · L
L → · **id**
L → · *R

8 E → L = R·

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

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

start

3 S → E ·

4 L → **id** ·

5 R → L ·

6 E → L · = R
R → L ·

Augmenting the Grammar

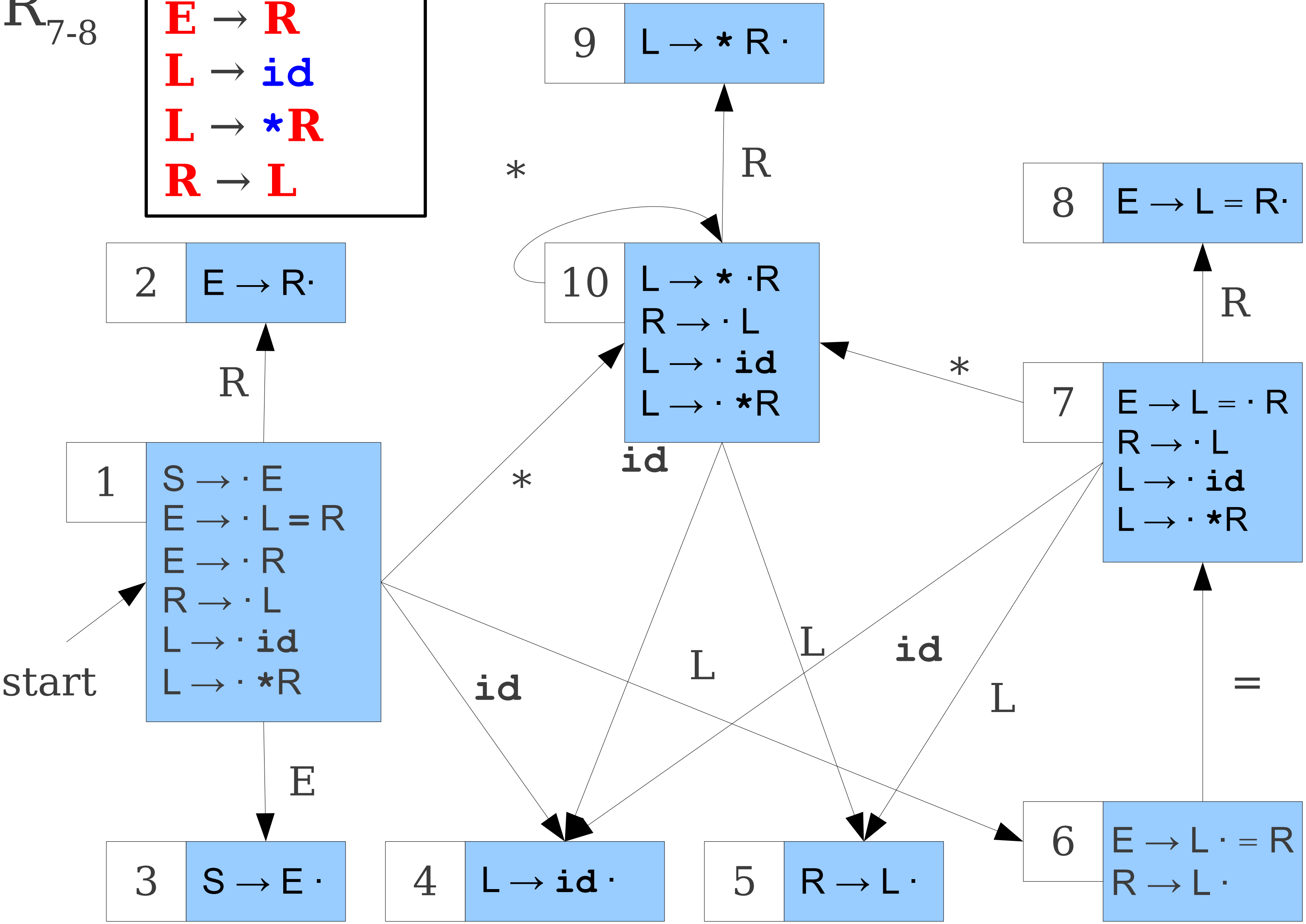
$S_1 \to E_{1\text{-}3}$

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

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

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

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



1
S → · E
E → · L = R
E → · R
R → · L
L → · **`id`**
L → · *****R

2
E → R·

3
S → E ·

4
L → **`id`** ·

5
R → L ·

6
E → L · = R
R → L ·

7
E → L = · R
R → · L
L → · **`id`**
L → · *****R

8
E → L = R·

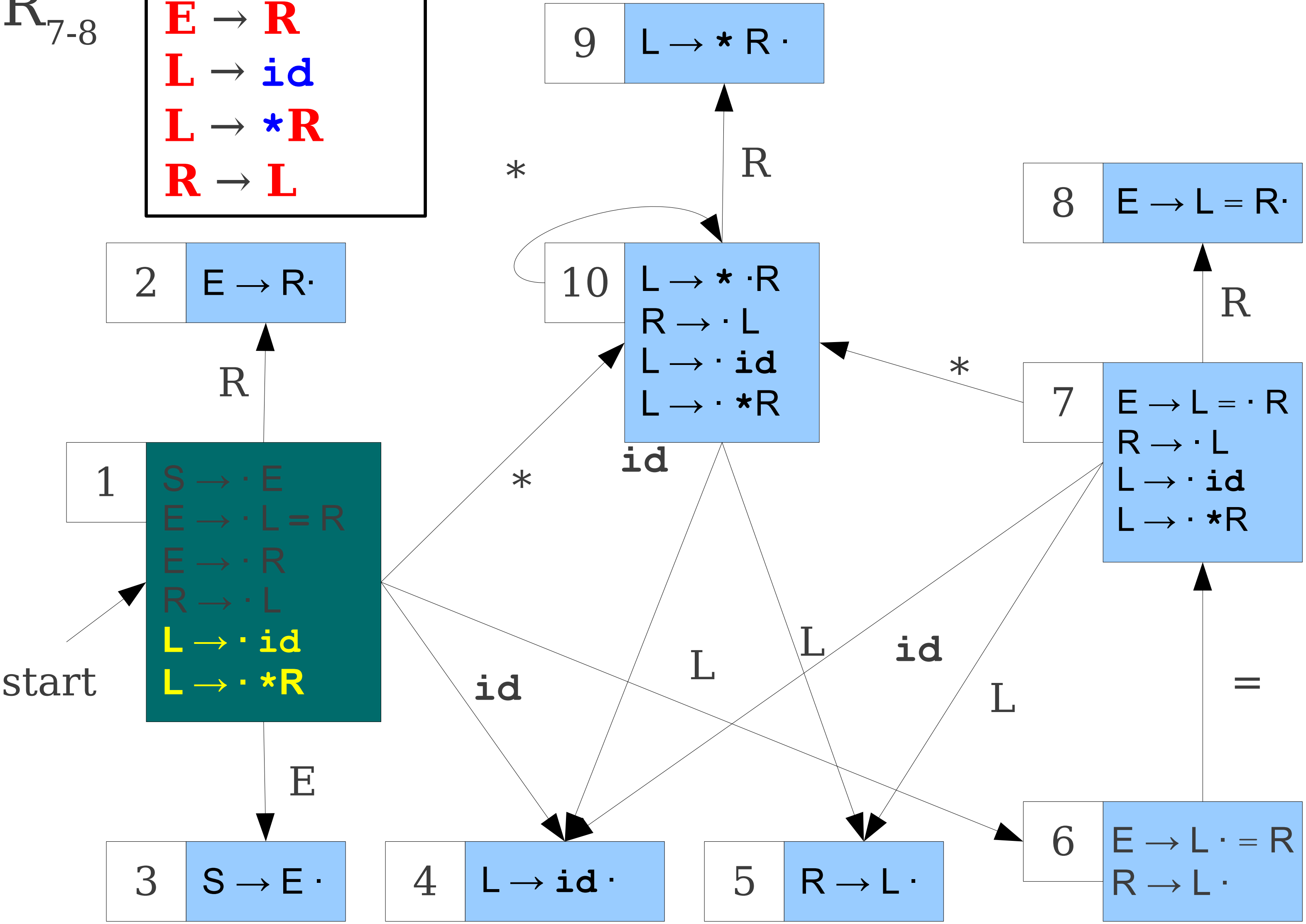
9
L → ***** R ·

10
L → ***** ·R
R → · L
L → · **`id`**
L → · *****R

Augmenting the Grammar

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

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



Augmenting the Grammar

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

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

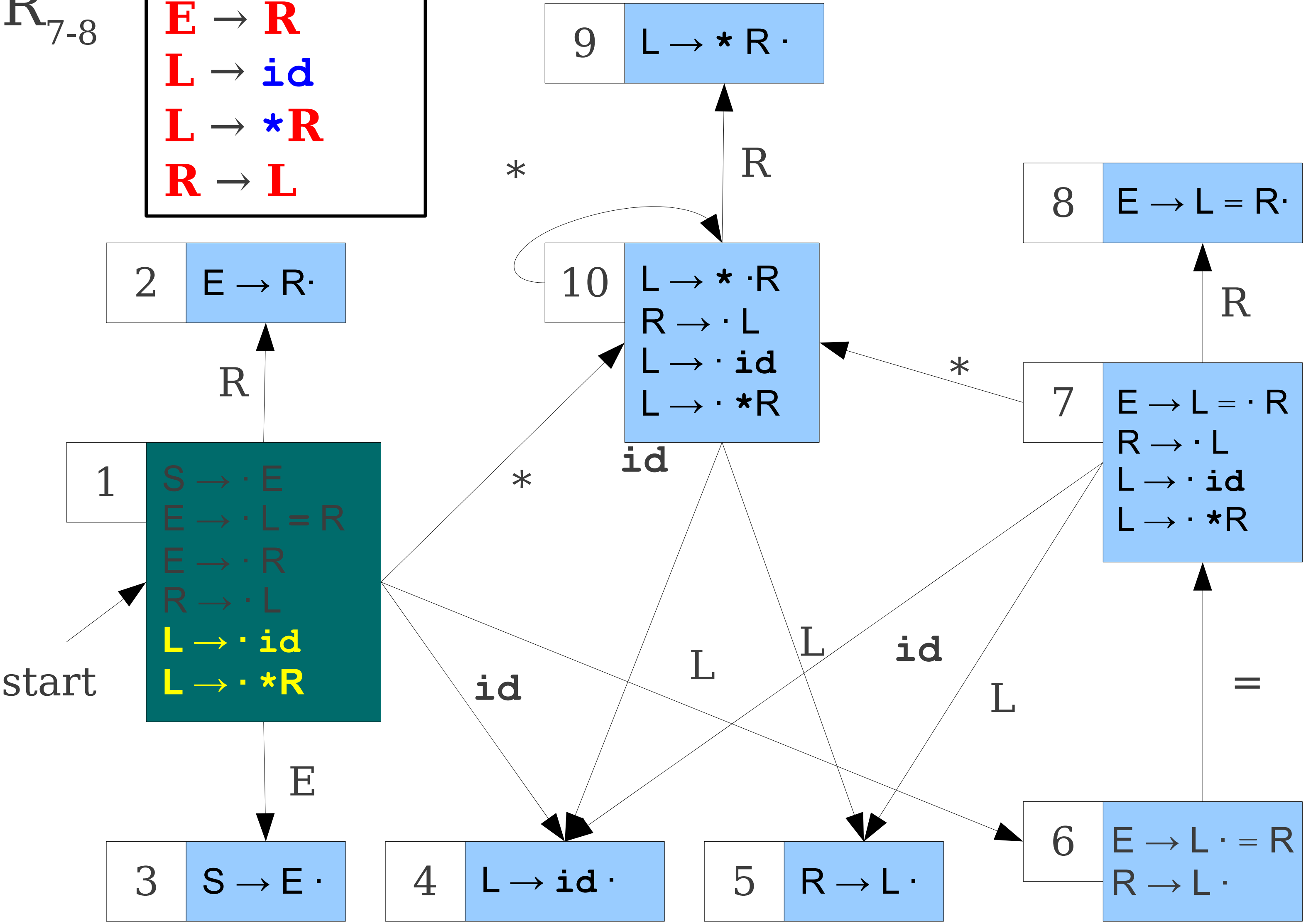
$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

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

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

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

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



Augmenting the Grammar

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

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

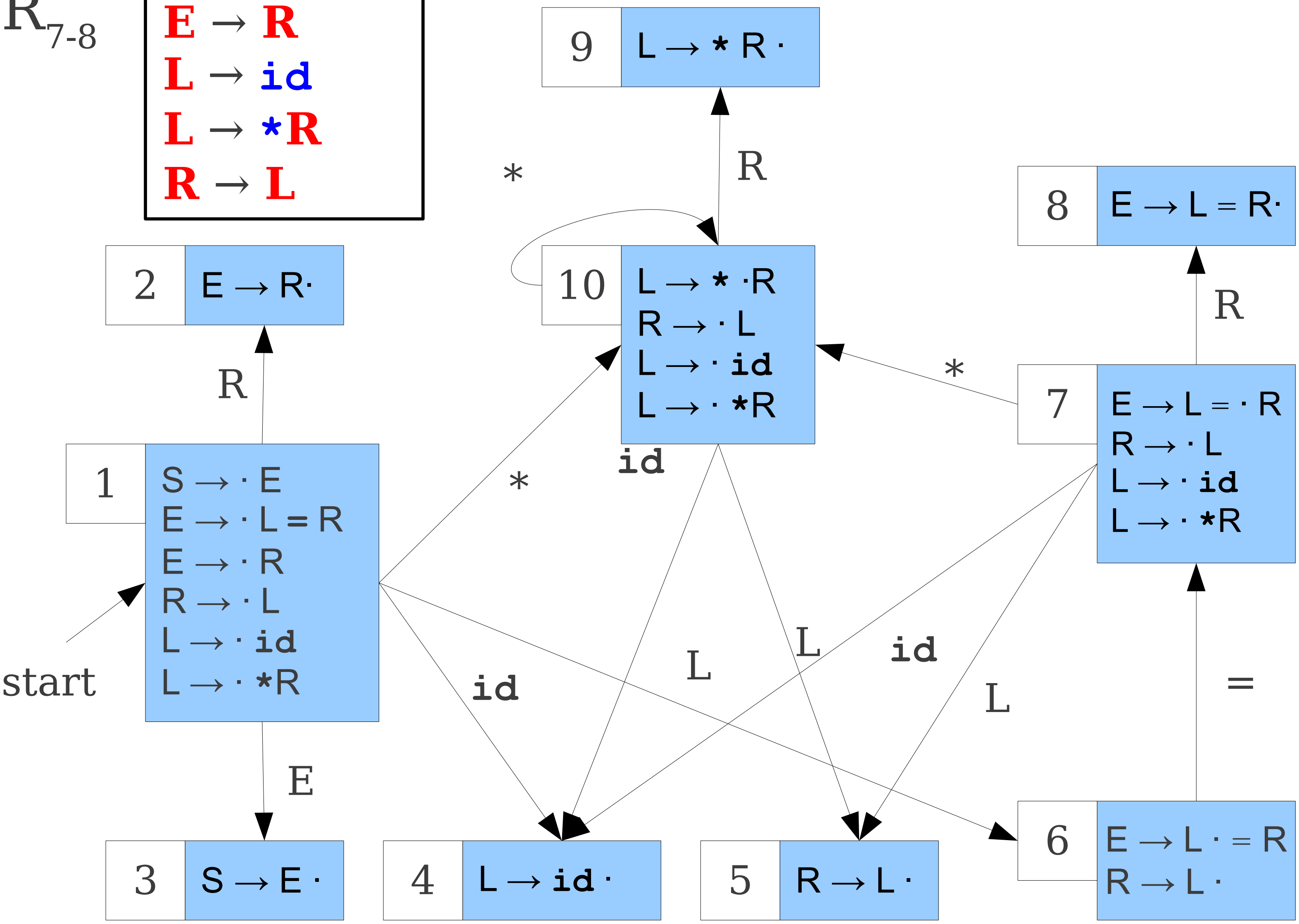
$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

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

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

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

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



Augmenting the Grammar

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

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

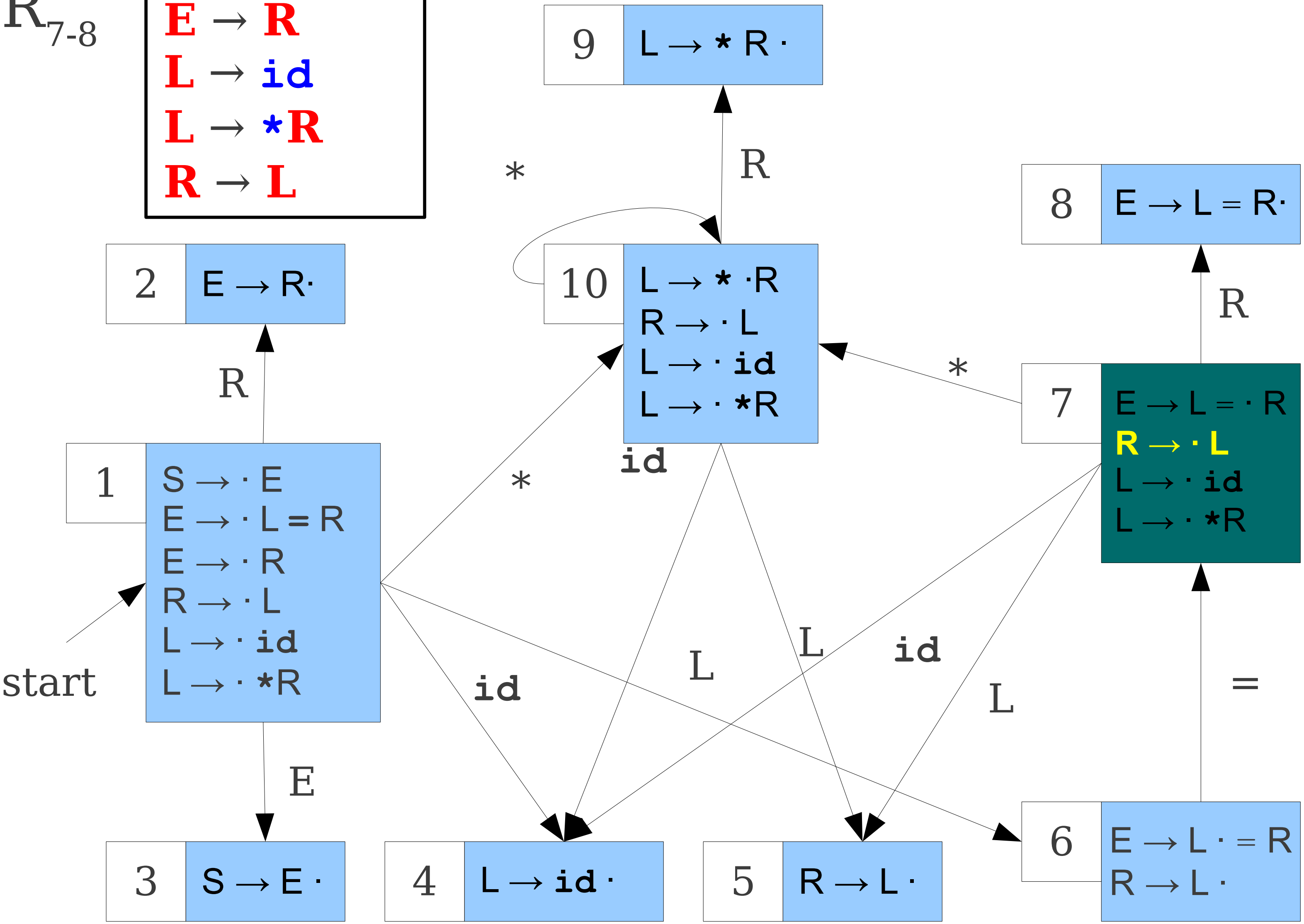
$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

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

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

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

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



Augmenting the Grammar

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

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

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

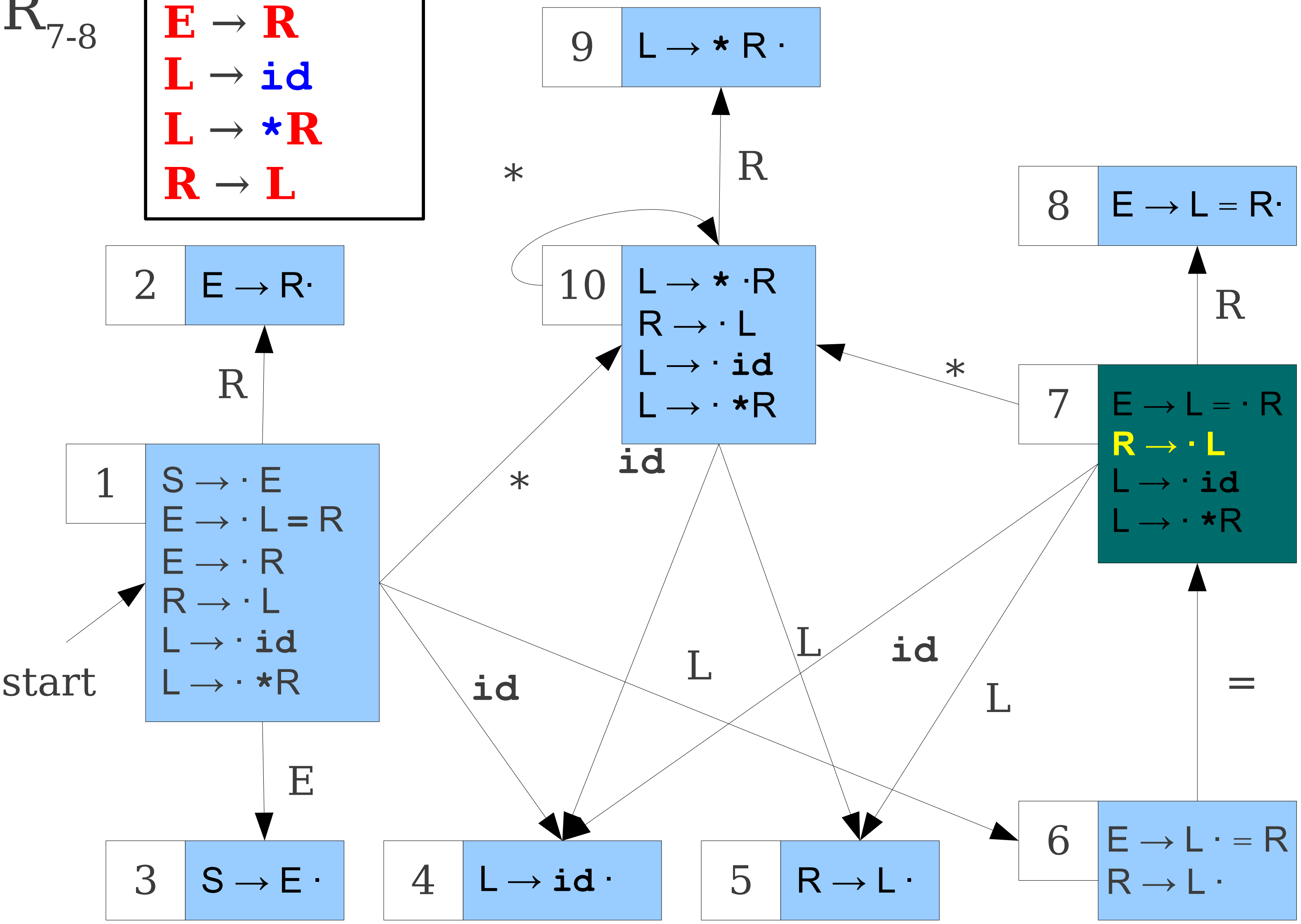
$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

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

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

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

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



Augmenting the Grammar

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

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

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

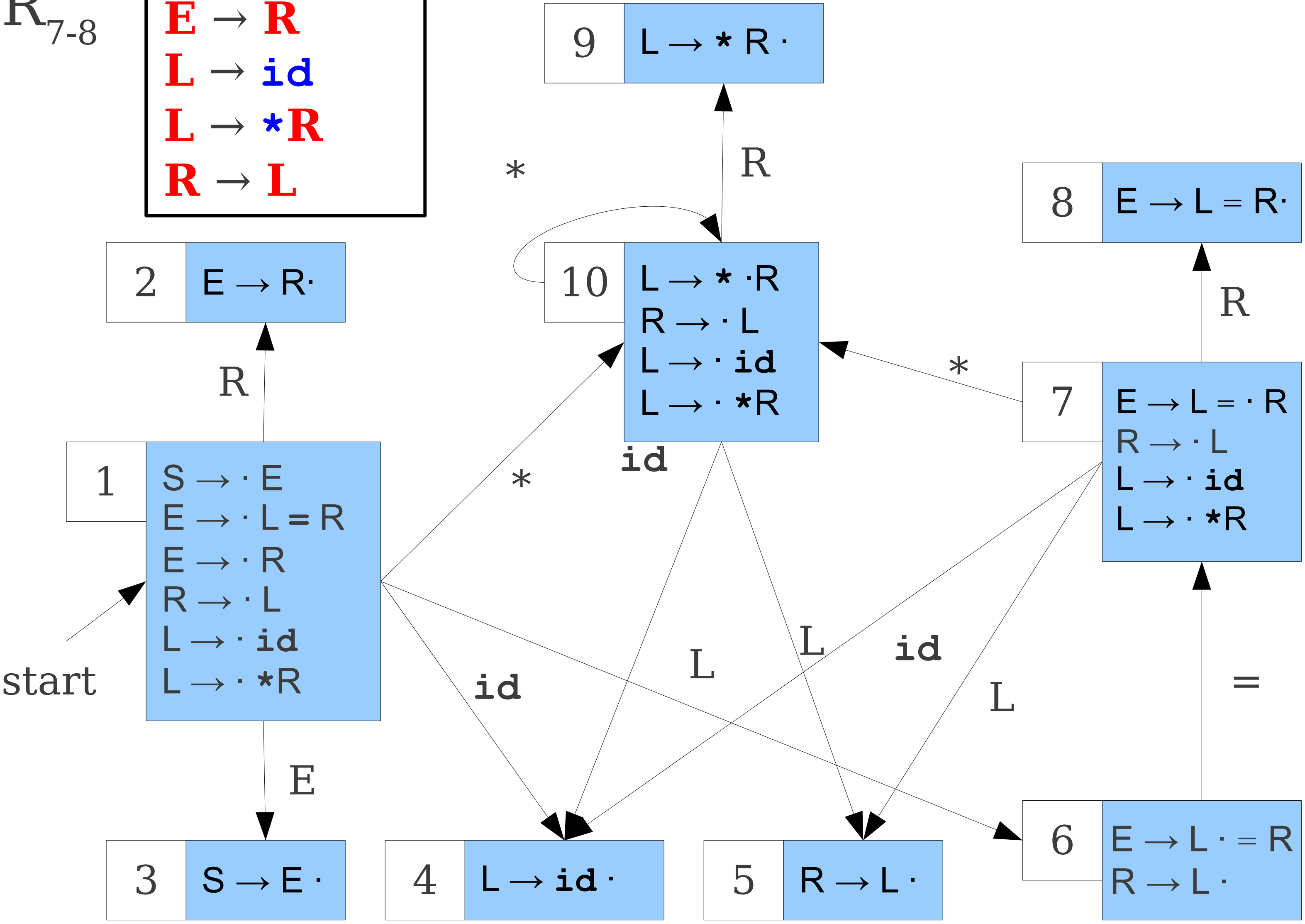
$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

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

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

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

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



Augmenting the Grammar

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

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

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

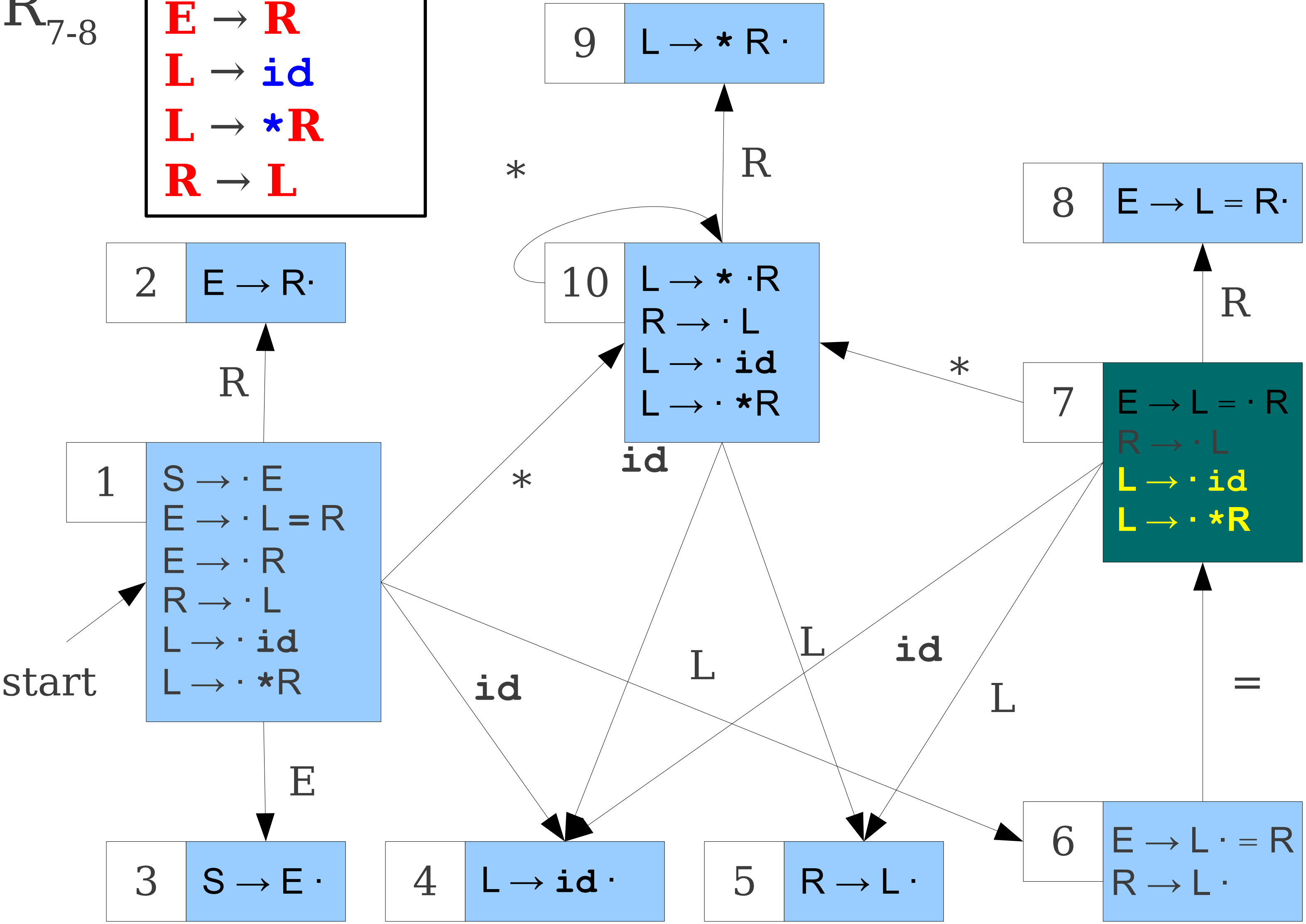
$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

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

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

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

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



Augmenting the Grammar

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

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

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

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

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

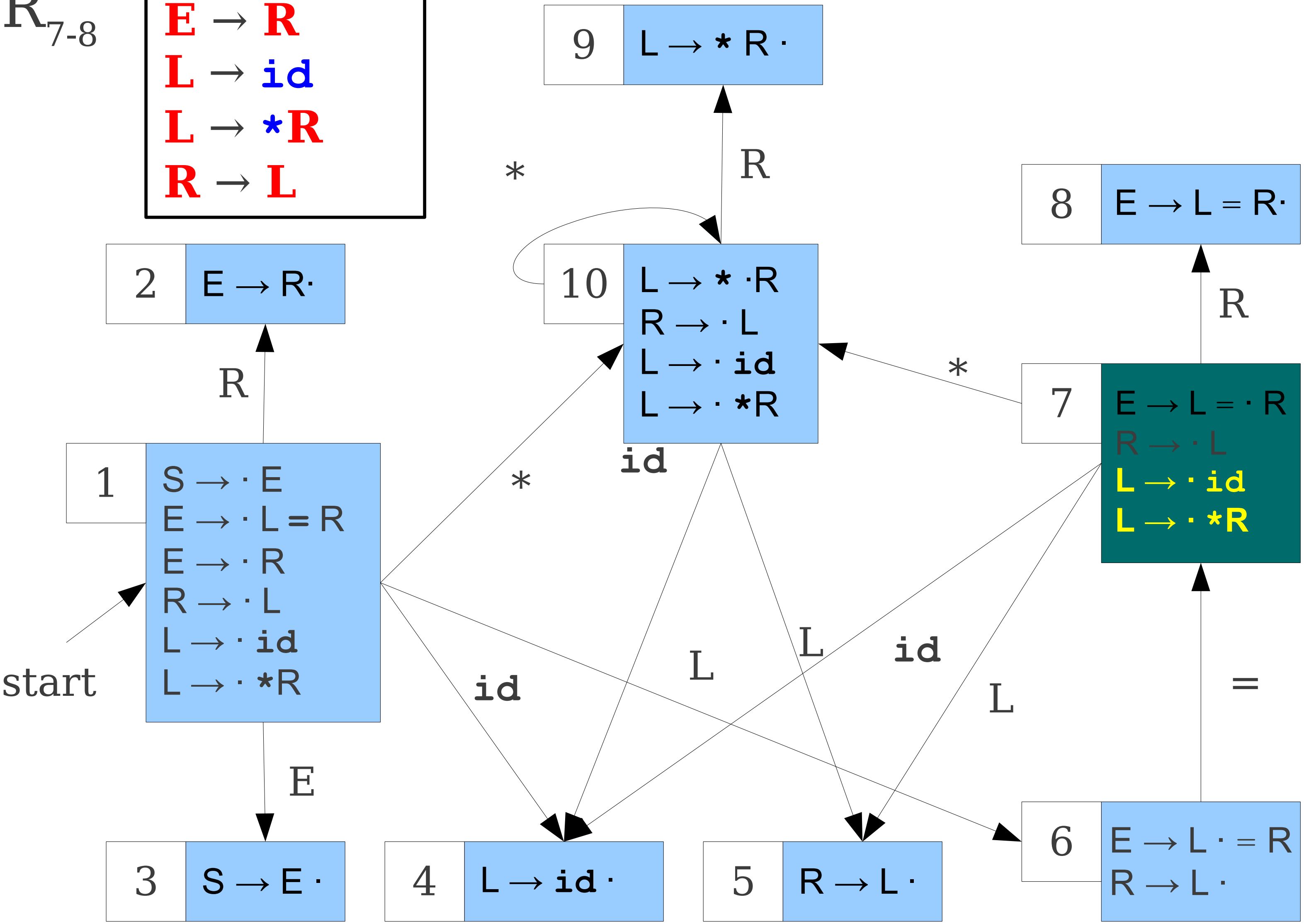
$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$

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

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

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

S → **E**
E → **L** = **R**
E → **R**
L → **`id`**
L → ***R**
R → **L**



1
S → · E
E → · L = R
E → · R
R → · L
L → · **id**
L → · ***R**

2
E → R·

3
S → E ·

4
L → **id** ·

5
R → L ·

6
E → L · = R
R → L ·

7
E → L = · R
R → · L
L → · **id**
L → · ***R**

8
E → L = R·

9
L → * R ·

10
L → * ·R
R → · L
L → · **id**
L → · ***R**

Augmenting the Grammar

$S_1 \to E_{1\text{-}3}$

$E_{1\text{-}3} \to L_{1\text{-}6} = R_{7\text{-}8}$

$E_{1\text{-}3} \to R_{1\text{-}2}$

$R_{1\text{-}2} \to L_{1\text{-}6}$

$L_{1\text{-}6} \to \mathbf{id}$

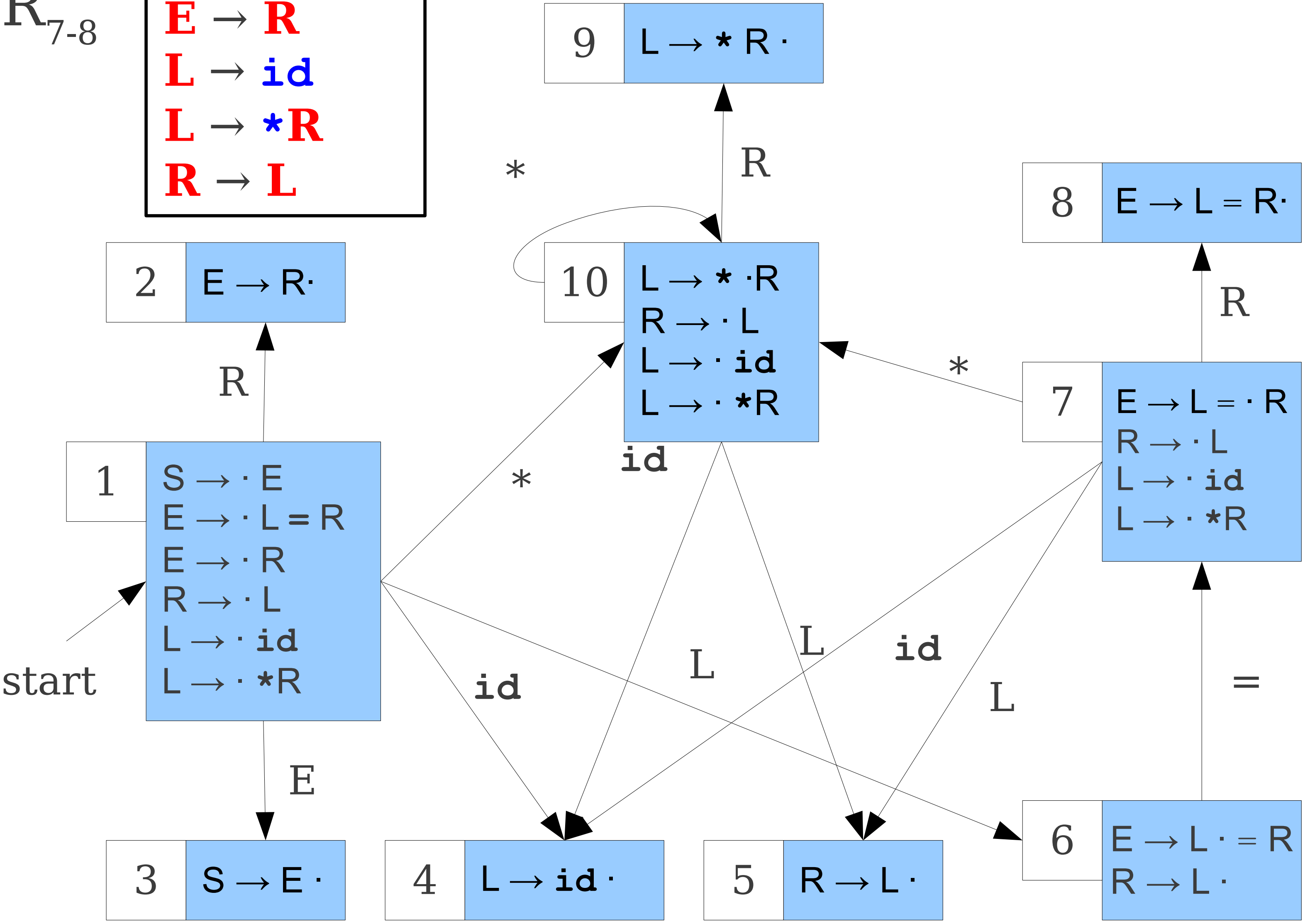
$L_{1\text{-}6} \to * \, R_{10\text{-}9}$

$R_{7\text{-}8} \to L_{7\text{-}5}$

$L_{7\text{-}5} \to \mathbf{id}$

$L_{7\text{-}5} \to *R_{10\text{-}9}$

S → **E**
E → **L** **=** **R**
E → **R**
L → **id**
L → ***R**
R → **L**



Augmenting the Grammar

$S_1 \to E_{1\text{-}3}$

$E_{1\text{-}3} \to L_{1\text{-}6} = R_{7\text{-}8}$

$E_{1\text{-}3} \to R_{1\text{-}2}$

$R_{1\text{-}2} \to L_{1\text{-}6}$

$L_{1\text{-}6} \to \texttt{id}$

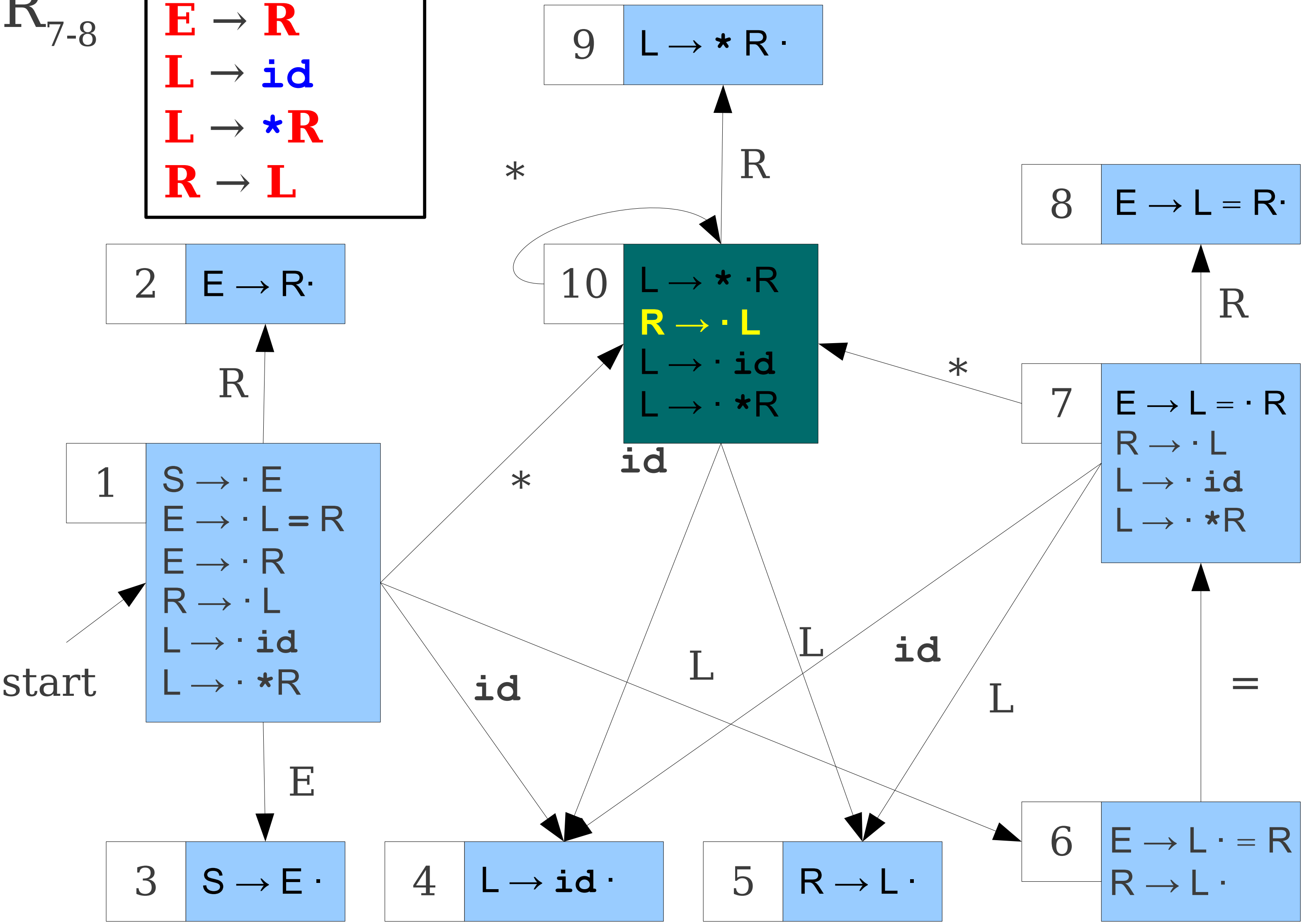
$L_{1\text{-}6} \to * R_{10\text{-}9}$

$R_{7\text{-}8} \to L_{7\text{-}5}$

$L_{7\text{-}5} \to \texttt{id}$

$L_{7\text{-}5} \to *R_{10\text{-}9}$

S → **E**
E → **L** **=** **R**
E → **R**
L → **`id`**
L → ***R**
R → **L**



Augmenting the Grammar

$S_1 \rightarrow E_{1\text{-}3}$

$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$

$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

$L_{1\text{-}6} \rightarrow$ **`id`**

$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$

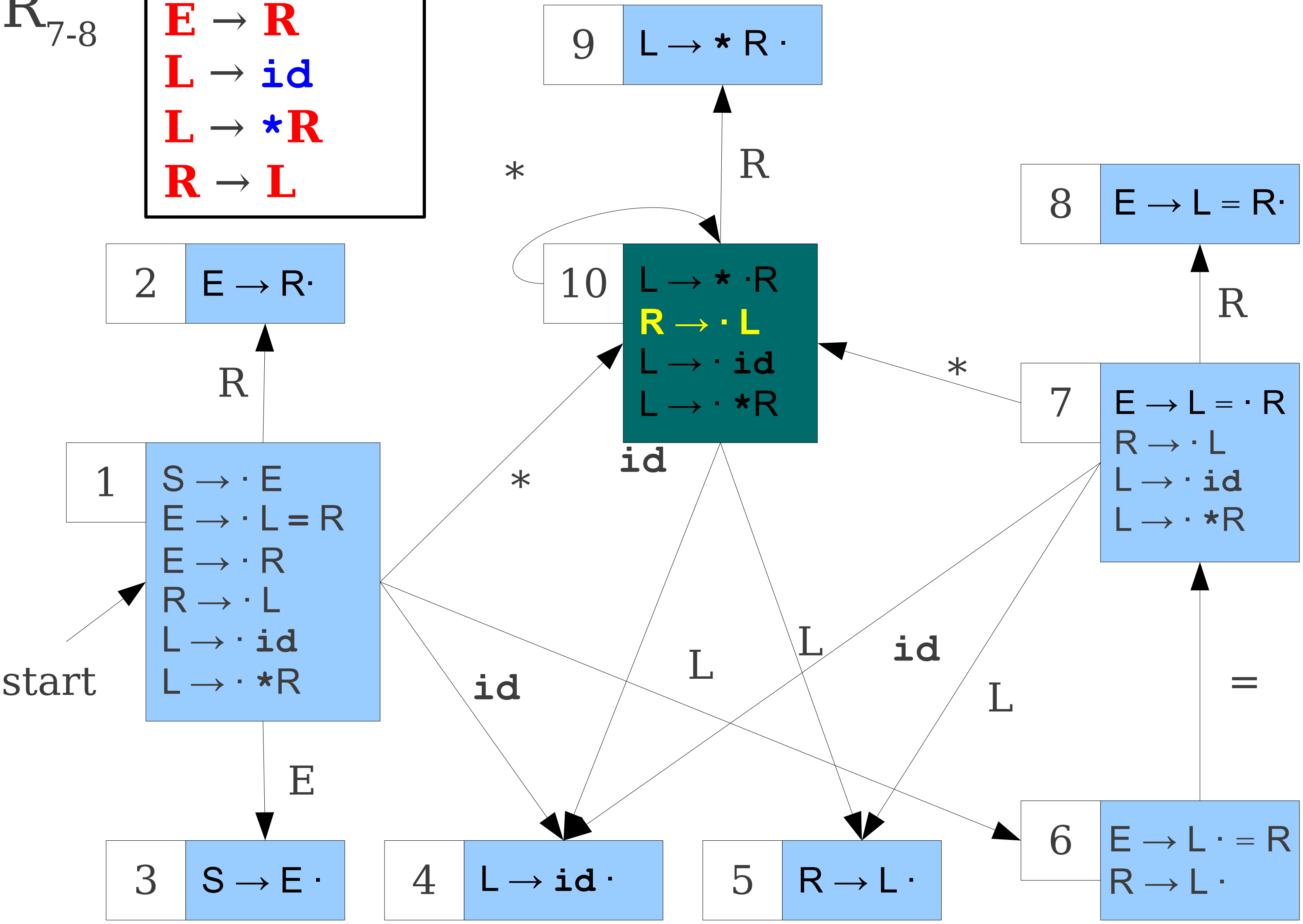
$R_{7\text{-}8} \rightarrow L_{7\text{-}5}$

$L_{7\text{-}5} \rightarrow$ **`id`**

$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

$R_{10\text{-}9} \rightarrow L_{10\text{-}5}$

S → **E**
E → **L** **=** **R**
E → **R**
L → **`id`**
L → ***R**
R → **L**



1
S → · E
E → · L = R
E → · R
R → · L
L → · id
L → · *R

2
E → R·

3
S → E ·

4
L → id ·

5
R → L ·

6
E → L · = R
R → L ·

7
E → L = · R
R → · L
L → · id
L → · *R

8
E → L = R·

9
L → * R ·

10
L → * ·R
R → · L
L → · id
L → · *R

Augmenting the Grammar

$S_{1} \rightarrow E_{1\text{-}3}$

$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$

$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

$L_{1\text{-}6} \rightarrow$ **`id`**

$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$

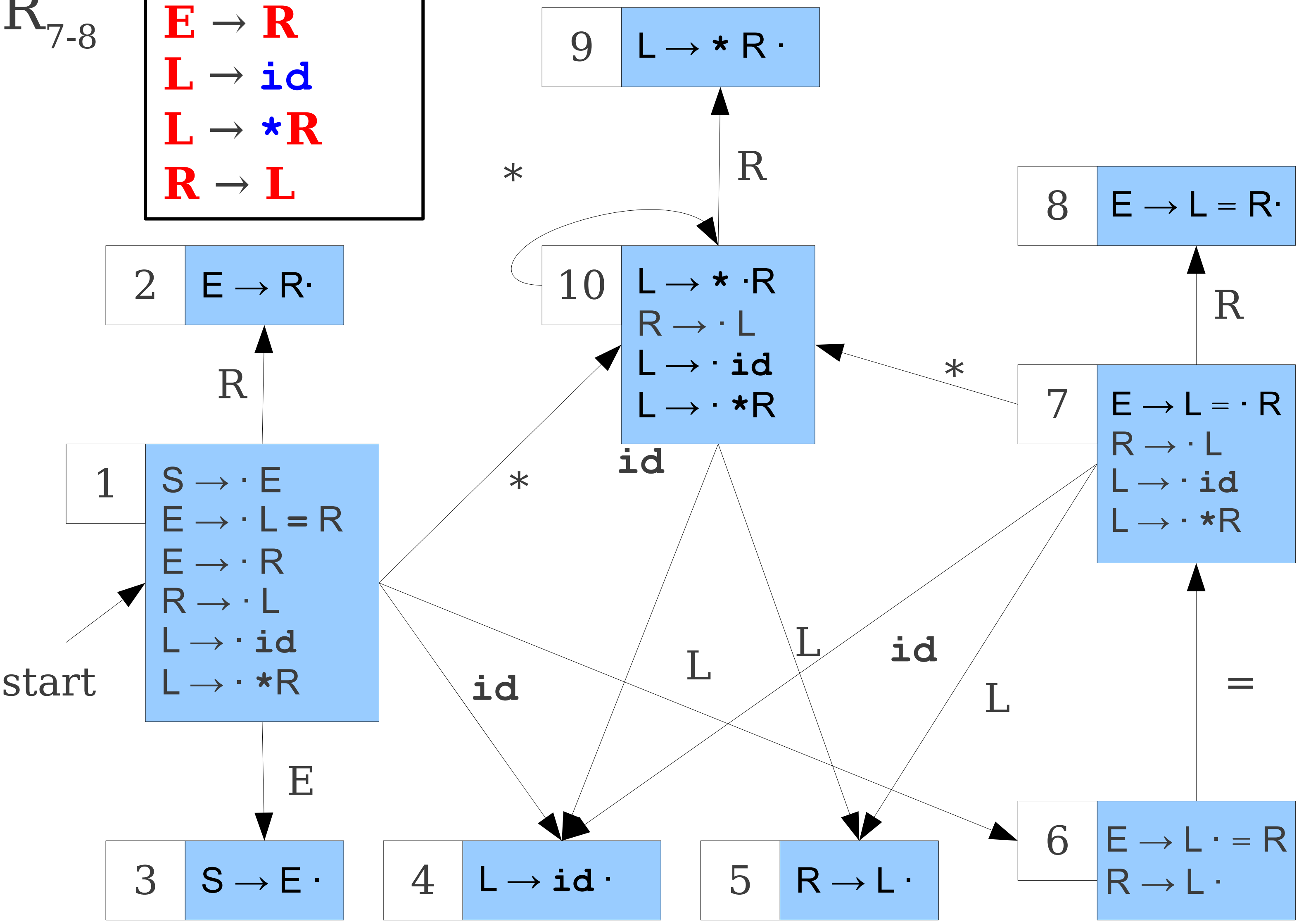
$R_{7\text{-}8} \rightarrow L_{7\text{-}5}$

$L_{7\text{-}5} \rightarrow$ **`id`**

$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

$R_{10\text{-}9} \rightarrow L_{10\text{-}5}$

S → **E**
E → **L** = **R**
E → **R**
L → `id`
L → ***R**
R → **L**


1
S → · E
E → · L = R
E → · R
R → · L
L → · **id**
L → · *R

2
E → R·

3
S → E ·

4
L → **id** ·

5
R → L ·

6
E → L · = R
R → L ·

7
E → L = · R
R → · L
L → · **id**
L → · *R

8
E → L = R·

9
L → * R ·

10
L → * ·R
R → · L
L → · **id**
L → · *R

Augmenting the Grammar

$S_1 \rightarrow E_{1\text{-}3}$

$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$

$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

$L_{1\text{-}6} \rightarrow \texttt{id}$

$L_{1\text{-}6} \rightarrow * \, R_{10\text{-}9}$

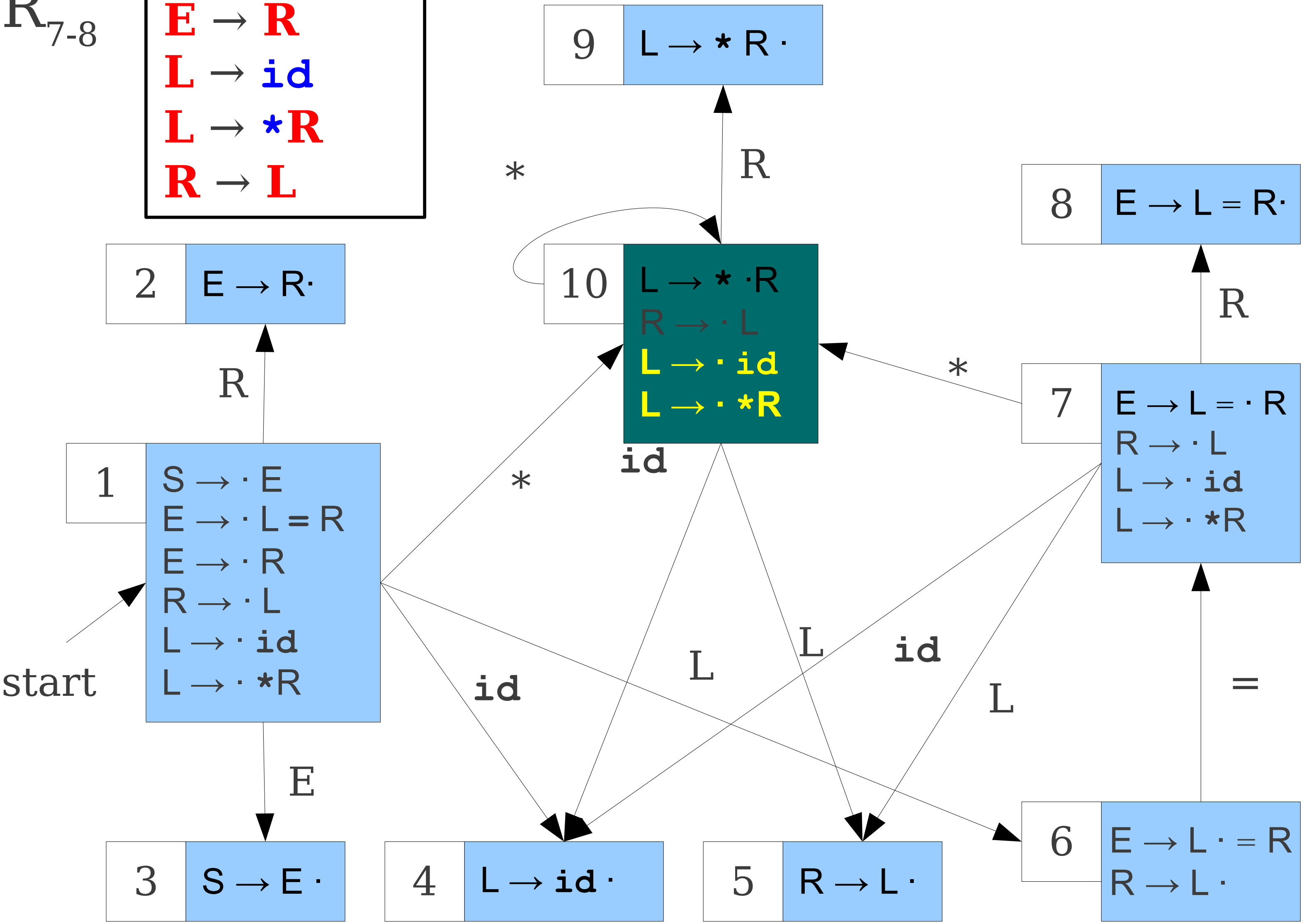
$R_{7\text{-}8} \rightarrow L_{7\text{-}5}$

$L_{7\text{-}5} \rightarrow \texttt{id}$

$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

$R_{10\text{-}9} \rightarrow L_{10\text{-}5}$

S → **E**
E → **L** **=** **R**
E → **R**
L → **id**
L → ***R**
R → **L**



Augmenting the Grammar

$S_{1} \rightarrow E_{1\text{-}3}$

$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$

$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

$L_{1\text{-}6} \rightarrow$ **`id`**

$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$

$R_{7\text{-}8} \rightarrow L_{7\text{-}5}$

$L_{7\text{-}5} \rightarrow$ **`id`**

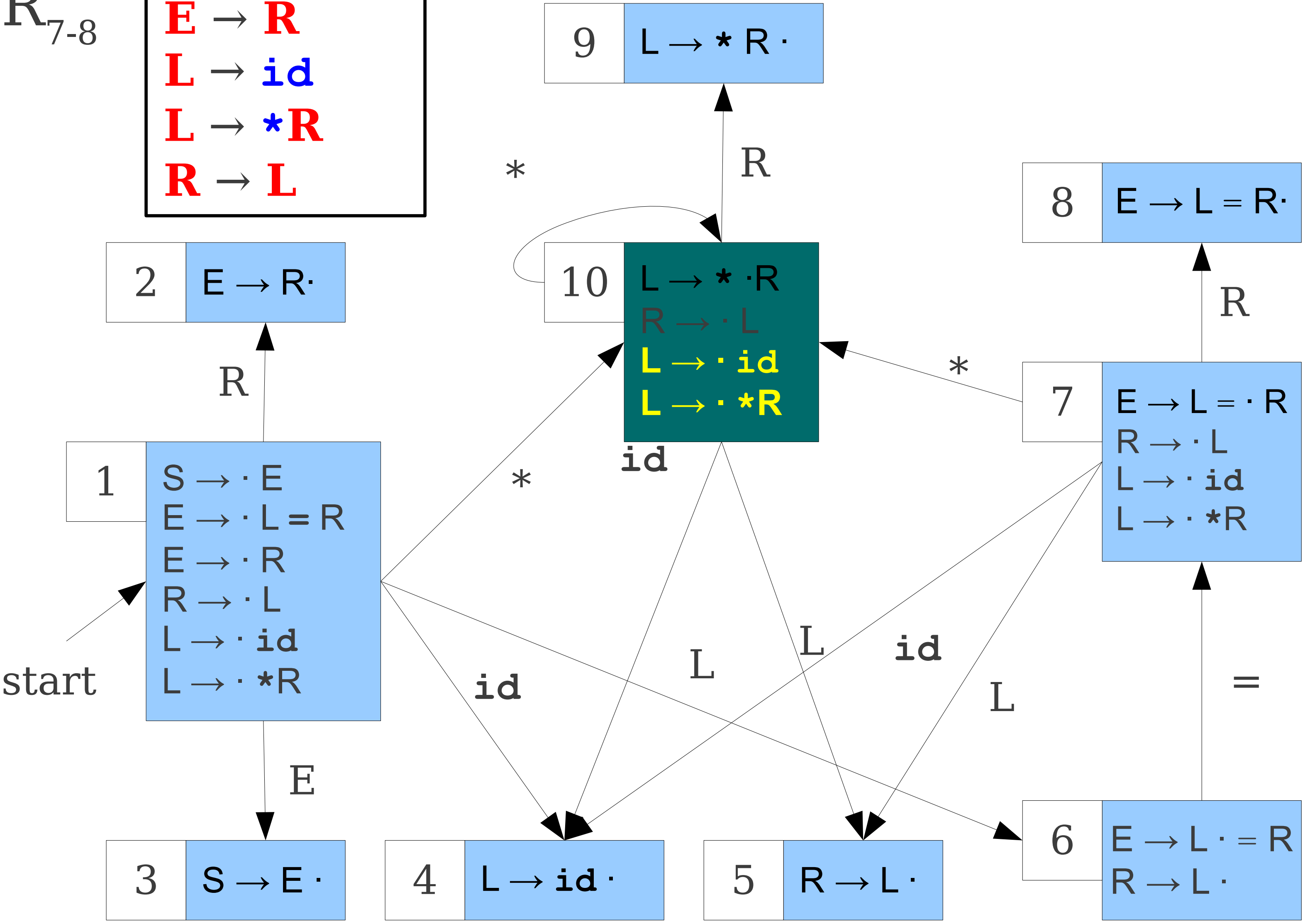
$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

$R_{10\text{-}9} \rightarrow L_{10\text{-}5}$

$L_{10\text{-}5} \rightarrow$ **`id`**

$L_{10\text{-}5} \rightarrow *R_{10\text{-}9}$

S → **E**
E → **L** **=** **R**
E → **R**
L → **`id`**
L → ***R**
R → **L**



1
S → · E
E → · L = R
E → · R
R → · L
L → · **id**
L → · *R

2
E → R·

3
S → E ·

4
L → **id** ·

5
R → L ·

6
E → L · = R
R → L ·

7
E → L = · R
R → · L
L → · **id**
L → · *R

8
E → L = R·

9
L → * R ·

10
L → * ·R
R → · L
L → · **id**
L → · *R

Augmenting the Grammar

$S_1 \rightarrow E_{1\text{-}3}$

$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$

$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

$L_{1\text{-}6} \rightarrow \mathtt{id}$

$L_{1\text{-}6} \rightarrow * \; R_{10\text{-}9}$

$R_{7\text{-}8} \rightarrow L_{7\text{-}5}$

$L_{7\text{-}5} \rightarrow \mathtt{id}$

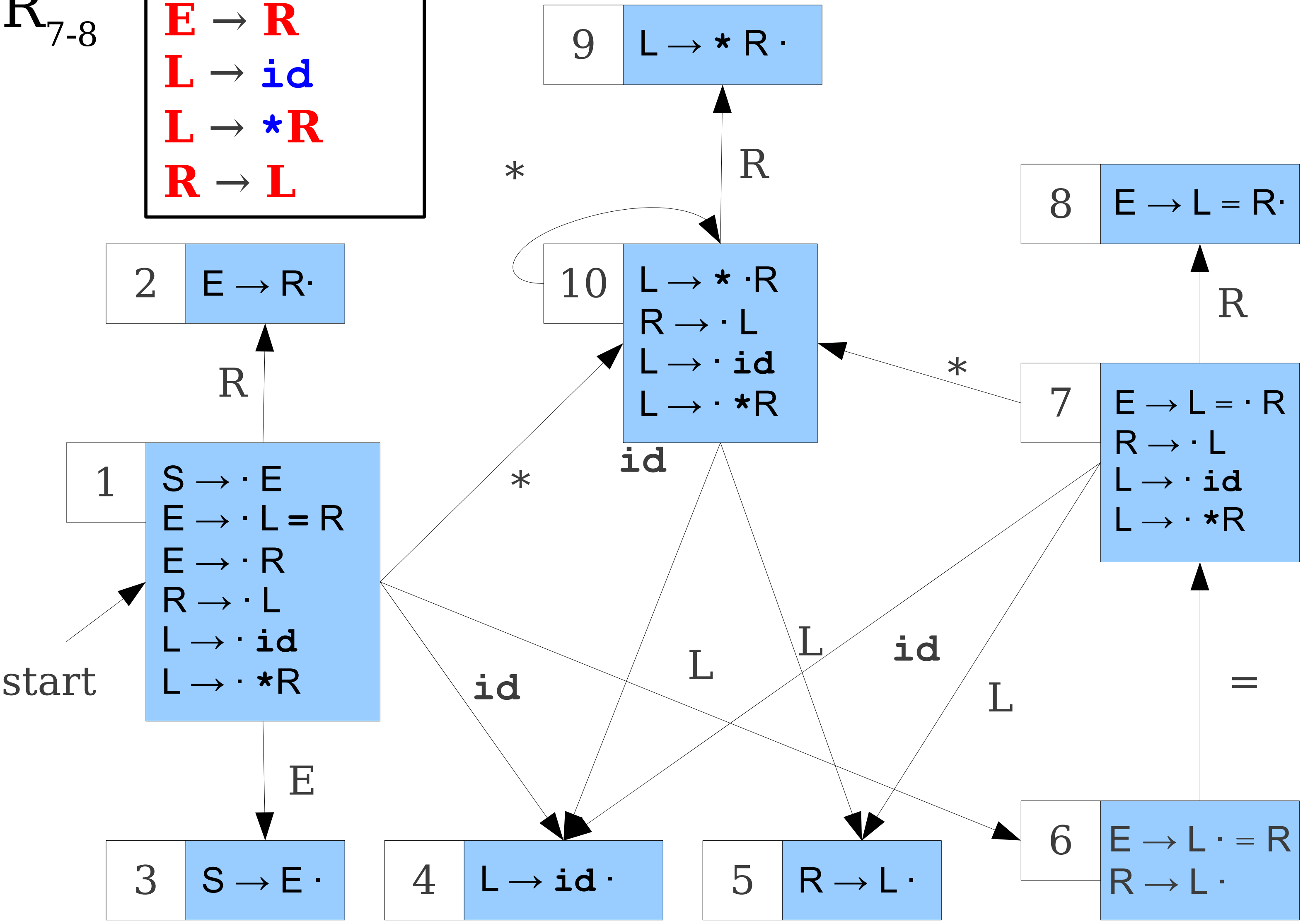
$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

$R_{10\text{-}9} \rightarrow L_{10\text{-}5}$

$L_{10\text{-}5} \rightarrow \mathtt{id}$

$L_{10\text{-}5} \rightarrow *R_{10\text{-}9}$

S → **E**
E → **L** **=** **R**
E → **R**
L → `id`
L → `*`**R**
R → **L**



1
S → · E
E → · L = R
E → · R
R → · L
L → · `id`
L → · `*`R

2
E → R·

3
S → E ·

4
L → `id` ·

5
R → L ·

6
E → L · = R
R → L ·

7
E → L = · R
R → · L
L → · `id`
L → · `*`R

8
E → L = R·

9
L → `*` R ·

10
L → `*` ·R
R → · L
L → · `id`
L → · `*`R

Constructing Augmented Grammars

- For each item **A** → $\cdot\boldsymbol{\omega}$ in some state *q*:
  - Trace out the path $\boldsymbol{\omega}$ takes through the LR(0) automaton starting at *q*.
  - Replace each nonterminal in $\boldsymbol{\omega}$ with a nonterminal annotated with the state transitioned to and from by the edge labeled with that nonterminal.
  - Replace **A** with a nonterminal annotated with the start and end state of the transition on **A** out of *q*.
- Result is a larger grammar with more precise productions.

Why is this Grammar Useful?

- At a high-level, separates out the nonterminals based on their context.
- This makes the FOLLOW sets more precise for their nonterminals.
- In fact, the FOLLOW sets are surprisingly precise.

Augmented FOLLOW Sets

$S_{1} \rightarrow E_{1\text{-}3}$

$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$

$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

$L_{1\text{-}6} \rightarrow$ **`id`**

$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$

$R_{7\text{-}8} \rightarrow L_{7\text{-}5}$

$L_{7\text{-}5} \rightarrow$ **`id`**

$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

$R_{10\text{-}9} \rightarrow L_{10\text{-}5}$

$L_{10\text{-}5} \rightarrow$ **`id`**

$L_{10\text{-}5} \rightarrow *R_{10\text{-}9}$

Augmented FOLLOW Sets

$S_1 \rightarrow E_{1\text{-}3}$

$E_{1\text{-}3} \rightarrow L_{1\text{-}6} = R_{7\text{-}8}$

$E_{1\text{-}3} \rightarrow R_{1\text{-}2}$

$R_{1\text{-}2} \rightarrow L_{1\text{-}6}$

$L_{1\text{-}6} \rightarrow$ **`id`**

$L_{1\text{-}6} \rightarrow * R_{10\text{-}9}$

$R_{7\text{-}8} \rightarrow L_{7\text{-}5}$

$L_{7\text{-}5} \rightarrow$ **`id`**

$L_{7\text{-}5} \rightarrow *R_{10\text{-}9}$

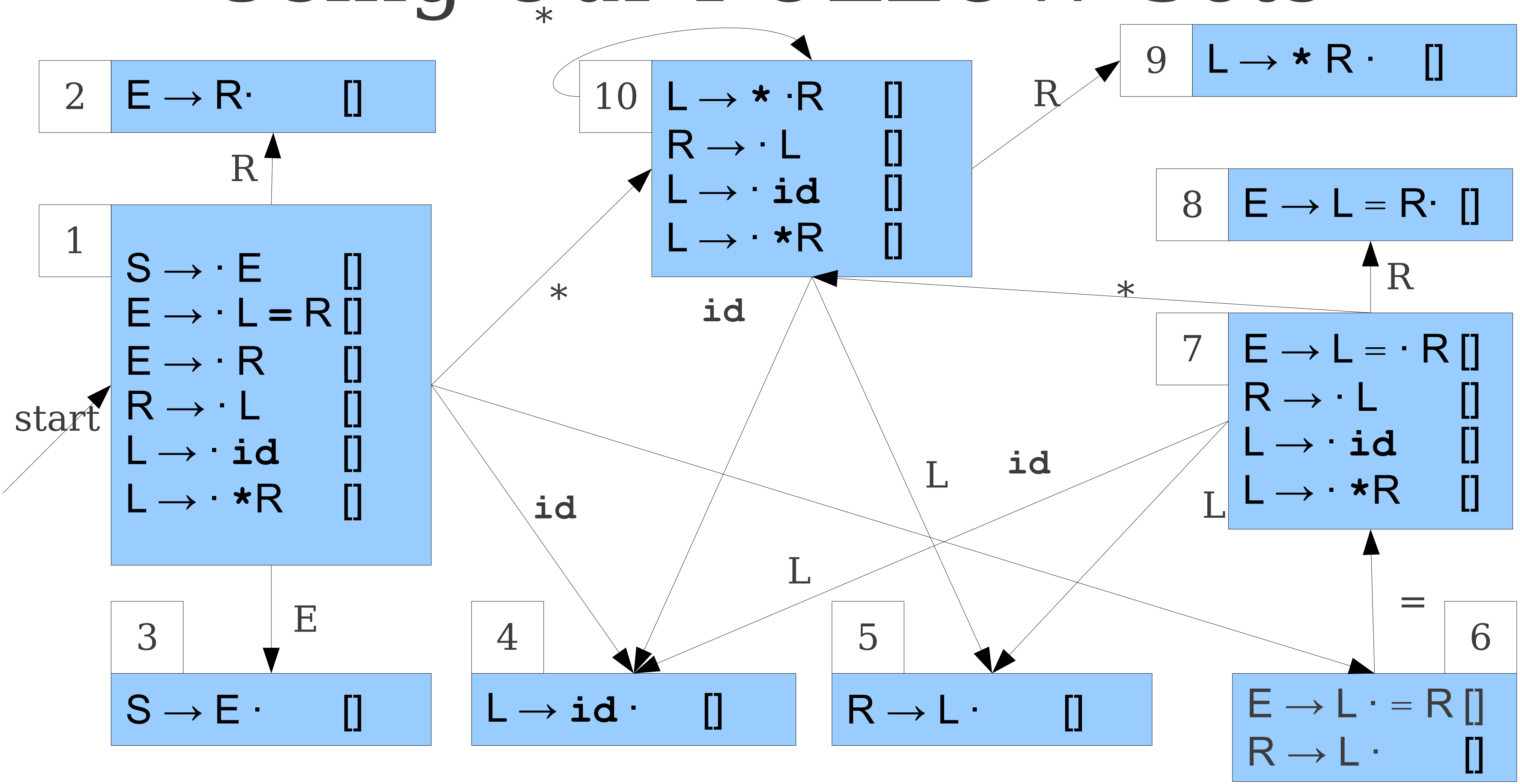
$R_{10\text{-}9} \rightarrow L_{10\text{-}5}$

$L_{10\text{-}5} \rightarrow$ **`id`**

$L_{10\text{-}5} \rightarrow *R_{10\text{-}9}$

S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	**$**	**=**	**$**	**=**	**$**	**$**	**=**

Using Our FOLLOW Sets



1

S → · E []

E → · L = R []

E → · R []

R → · L []

L → · **id** []

L → · *****R []

2

E → R· []

3

S → E · []

4

L → **id** · []

5

R → L · []

6

E → L · = R []

R → L · []

7

E → L = · R []

R → · L []

L → · **id** []

L → · *****R []

8

E → L = R· []

9

L → ***** R · []

10

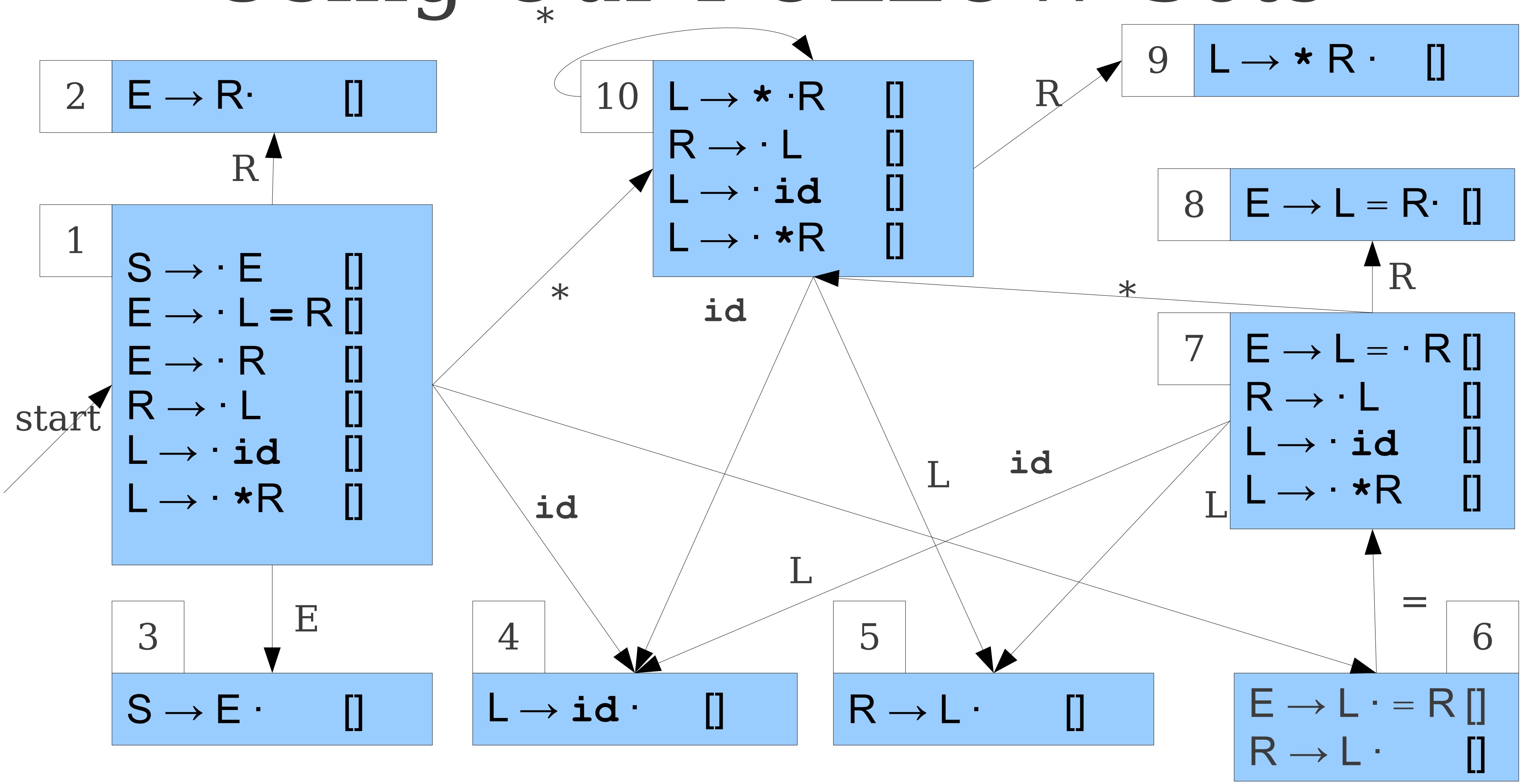
L → ***** ·R []

R → · L []

L → · **id** []

L → · *****R []

Using Our FOLLOW Sets



1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · **id** []
L → · ***R** []

2
E → R· []

3
S → E · []

4
L → **id** · []

5
R → L · []

6
E → L · = R []
R → L · []

7
E → L = · R []
R → · L []
L → · **id** []
L → · ***R** []

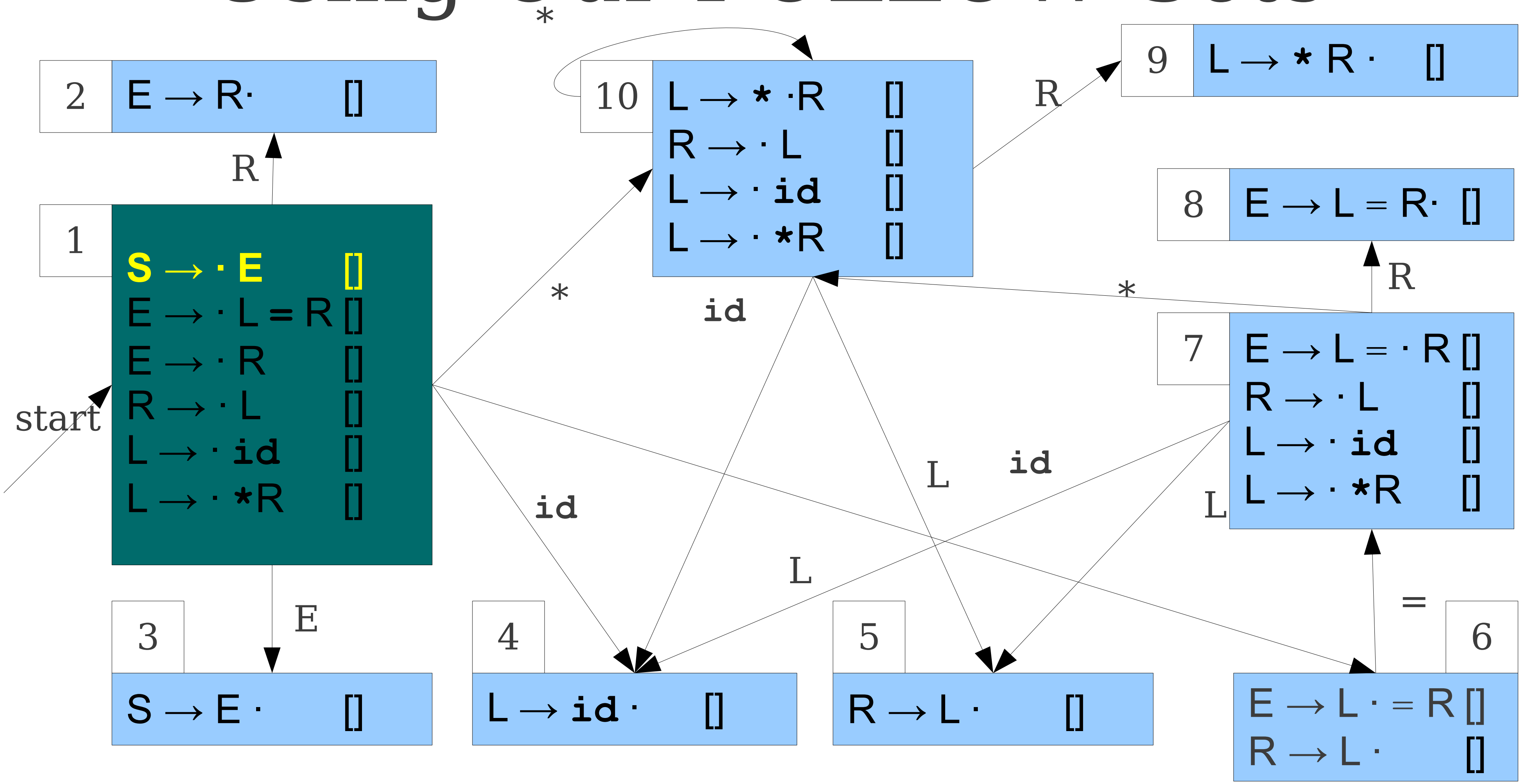
8
E → L = R· []

9
L → * R · []

10
L → * ·R []
R → · L []
L → · **id** []
L → · ***R** []

S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



2
E → R· []

10
L → * ·R []
R → · L []
L → · **id** []
L → · *R []

9
L → * R · []

1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · **id** []
L → · *R []

8
E → L = R· []

7
E → L = · R []
R → · L []
L → · **id** []
L → · *R []

3
S → E · []

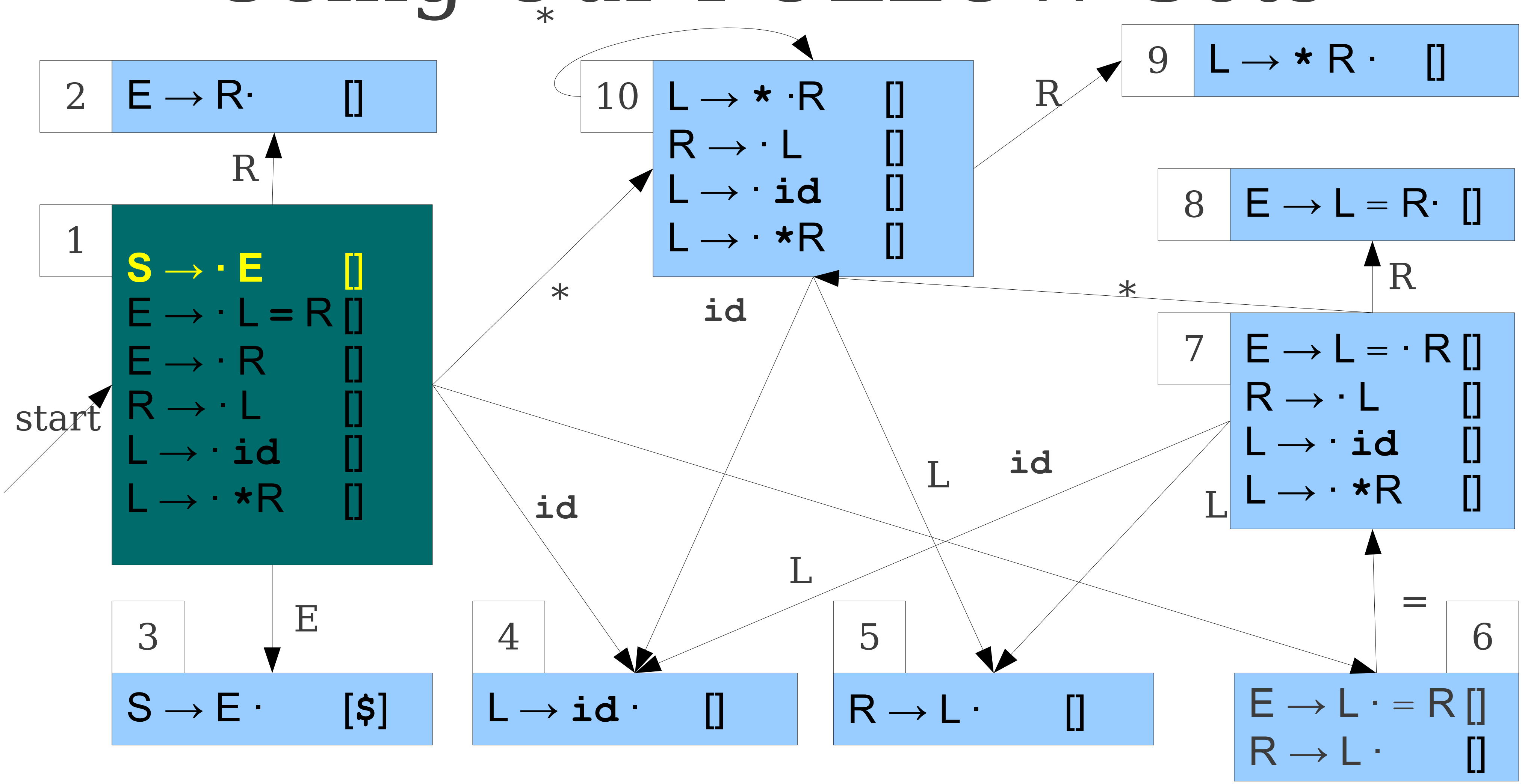
4
L → **id** · []

5
R → L · []

6
E → L · = R []
R → L · []

S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1

S → · E []

E → · L = R []

E → · R []

R → · L []

L → · **id** []

L → · *****R []

2

E → R· []

3

S → E · [$]

4

L → **id** · []

5

R → L · []

6

E → L · = R []

R → L · []

7

E → L = · R []

R → · L []

L → · **id** []

L → · *****R []

8

E → L = R· []

9

L → ***** R · []

10

L → ***** ·R []

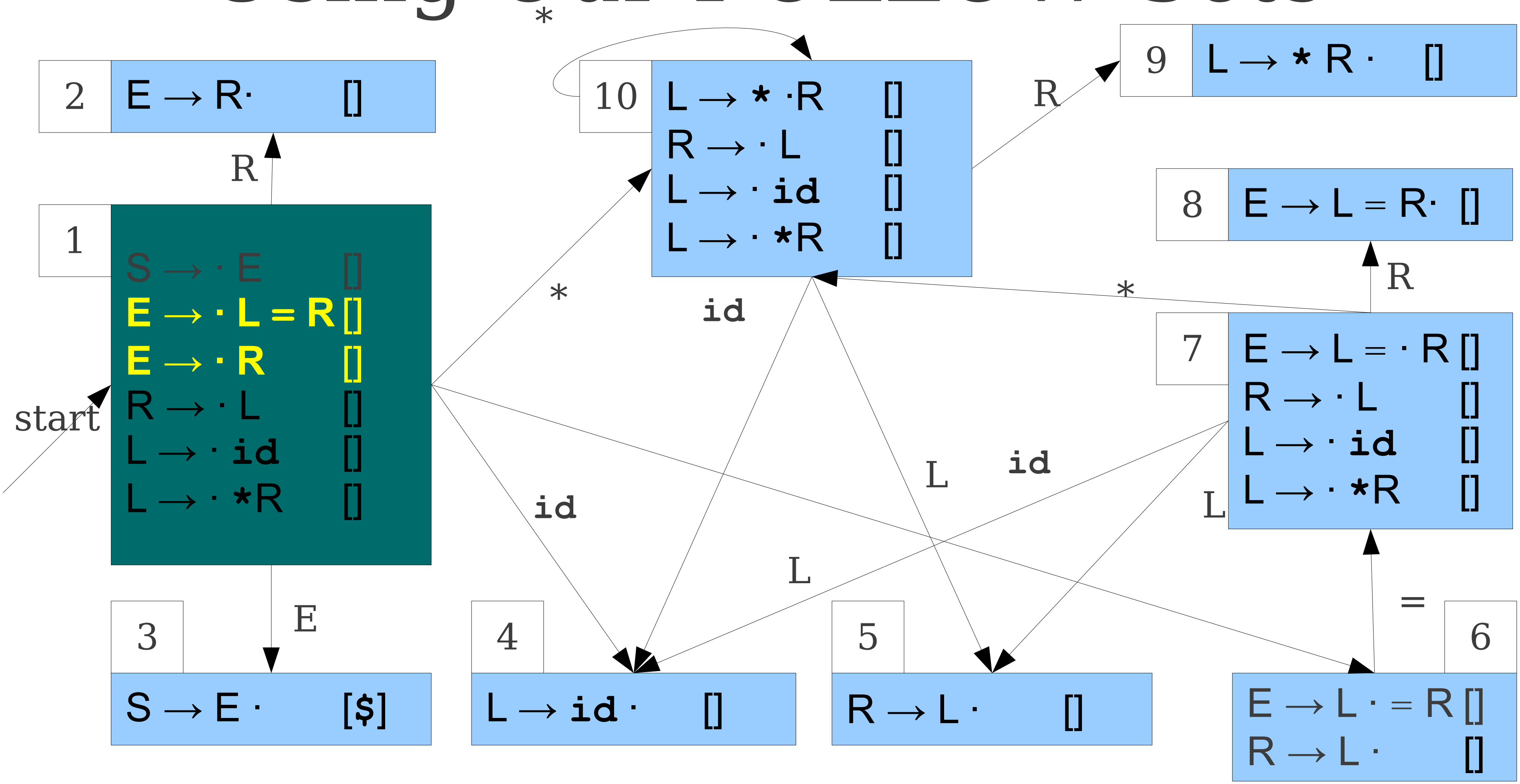
R → · L []

L → · **id** []

L → · *****R []

S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · **id** []
L → · ***R** []

2
E → R· []

10
L → * ·R []
R → · L []
L → · **id** []
L → · ***R** []

9
L → * R · []

8
E → L = R· []

7
E → L = · R []
R → · L []
L → · **id** []
L → · ***R** []

3
S → E · [$]

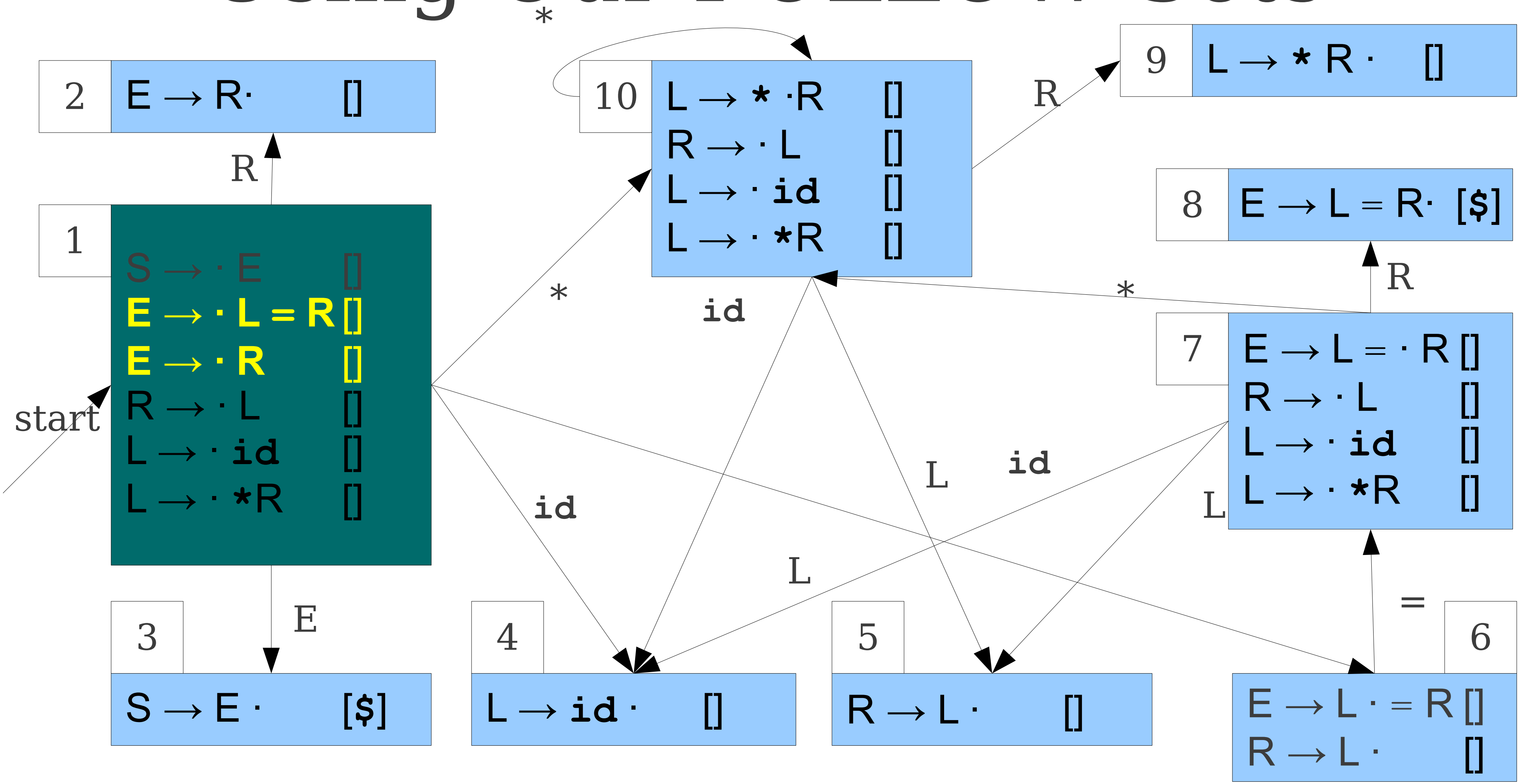
4
L → **id** · []

5
R → L · []

6
E → L · = R []
R → L · []

S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1

S → · E []

E → · L = R []

E → · R []

R → · L []

L → · **id** []

L → · *****R []

2

E → R· []

3

S → E · [$]

4

L → **id** · []

5

R → L · []

6

E → L · = R []

R → L · []

7

E → L = · R []

R → · L []

L → · **id** []

L → · *****R []

8

E → L = R· [$]

9

L → ***** R · []

10

L → ***** ·R []

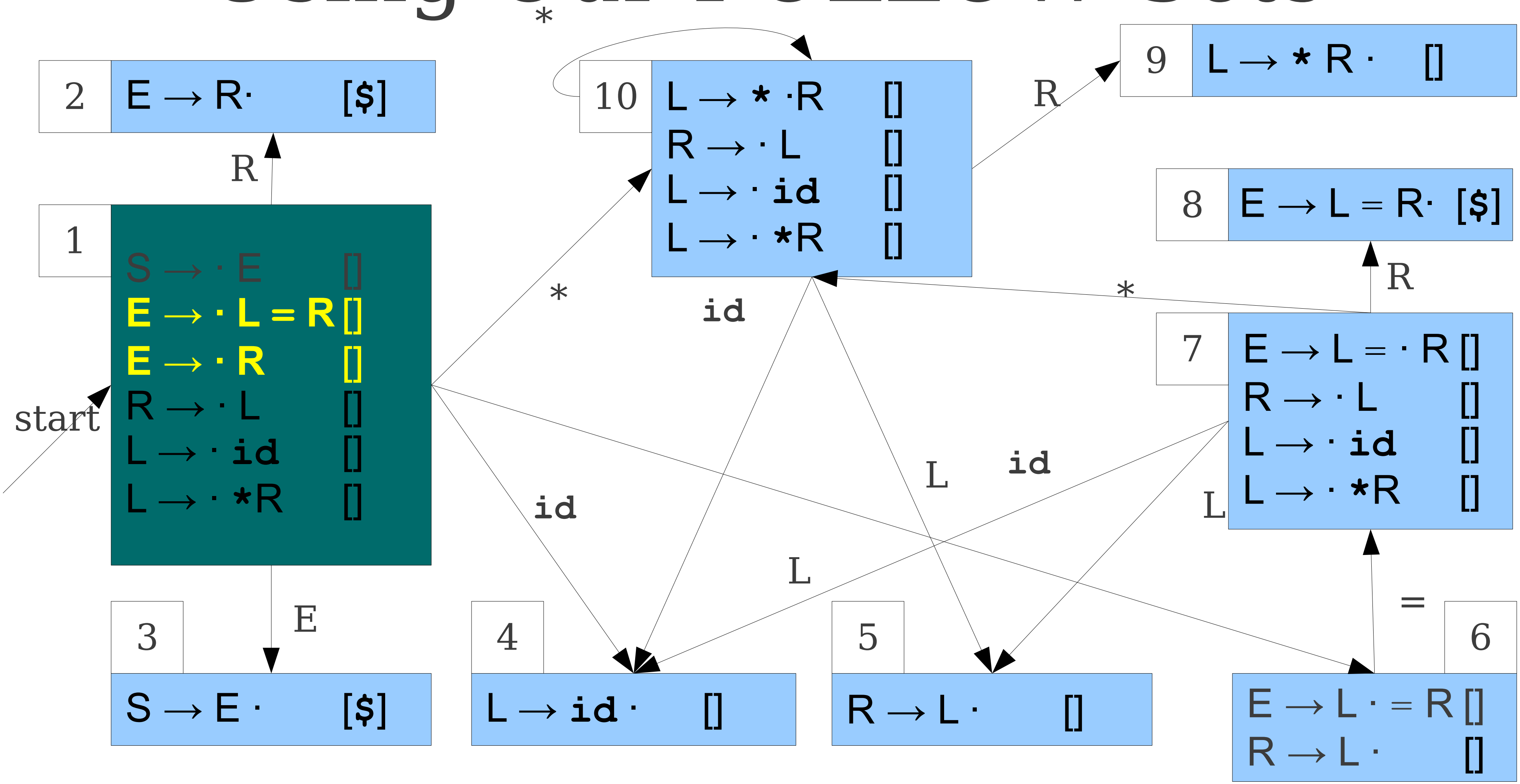
R → · L []

L → · **id** []

L → · *****R []

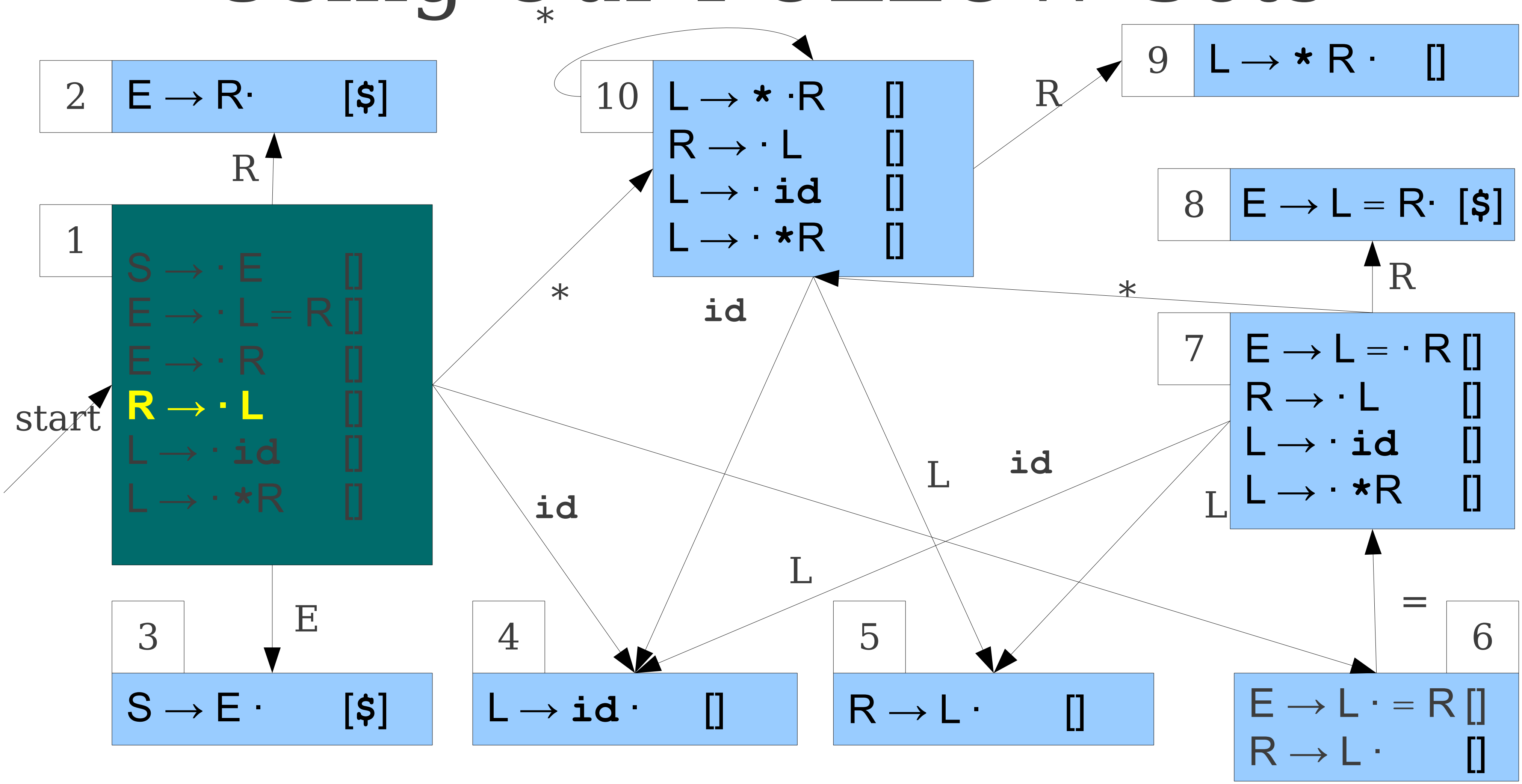
S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · **id** []
L → · ***R** []

2
E → R· [$]

3
S → E · [$]

4
L → **id** · []

5
R → L · []

6
E → L · = R []
R → L · []

7
E → L = · R []
R → · L []
L → · **id** []
L → · ***R** []

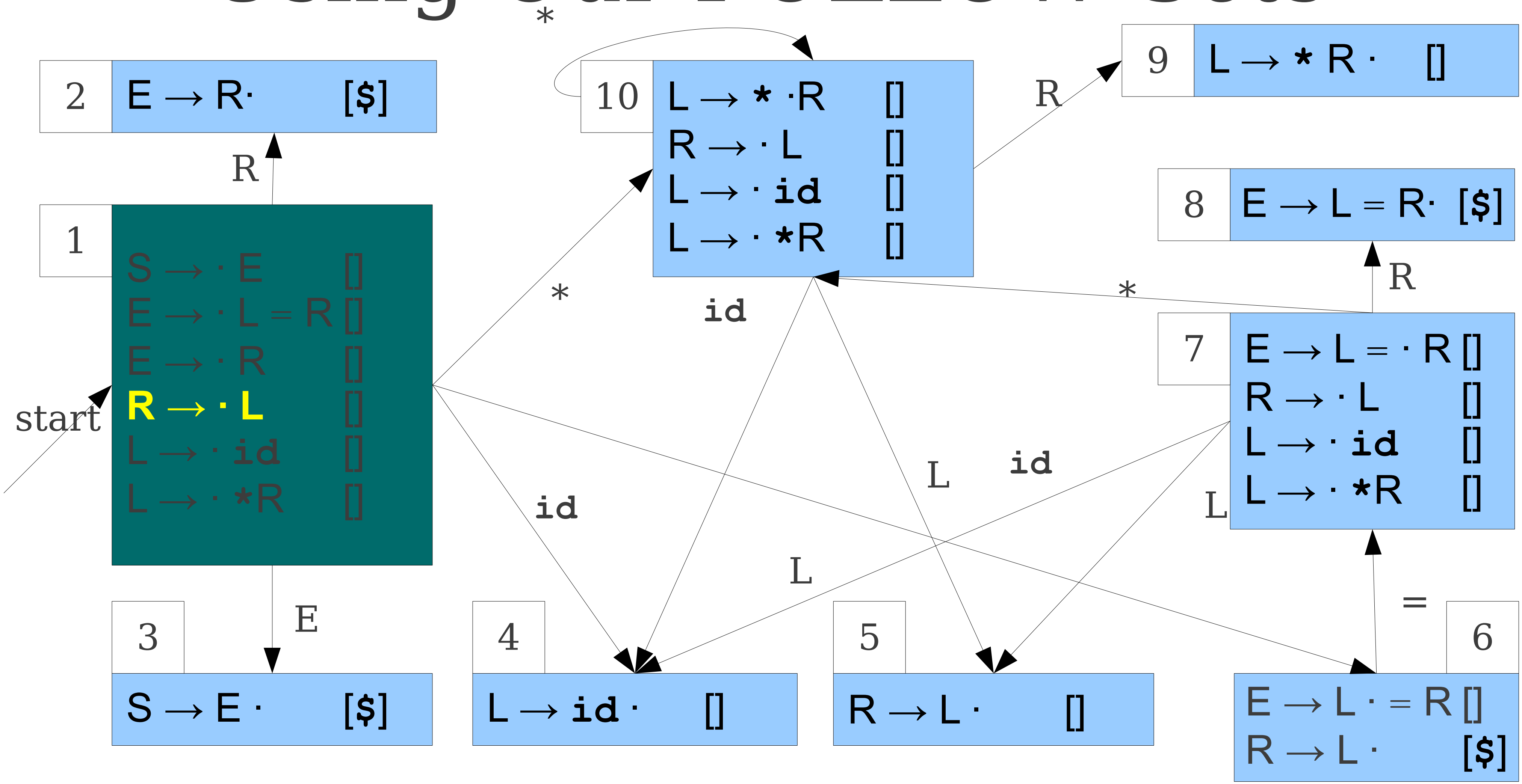
8
E → L = R· [$]

9
L → * R · []

10
L → * ·R []
R → · L []
L → · **id** []
L → · ***R** []

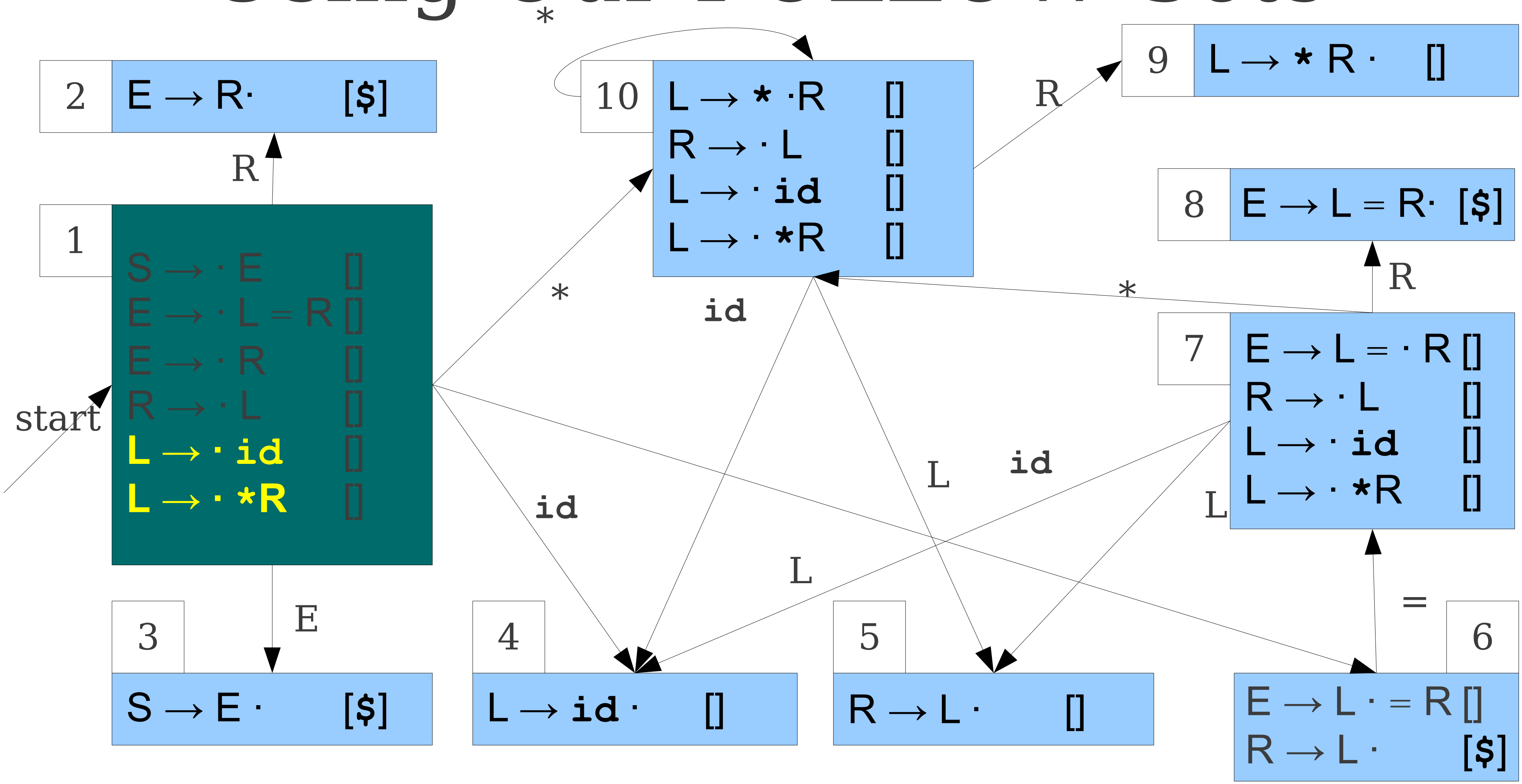
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · id []
L → · *R []

2
E → R· [$]

3
S → E · [$]

4
L → **id** · []

5
R → L · []

6
E → L · = R []
R → L · [$]

7
E → L = · R []
R → · L []
L → · **id** []
L → · *****R []

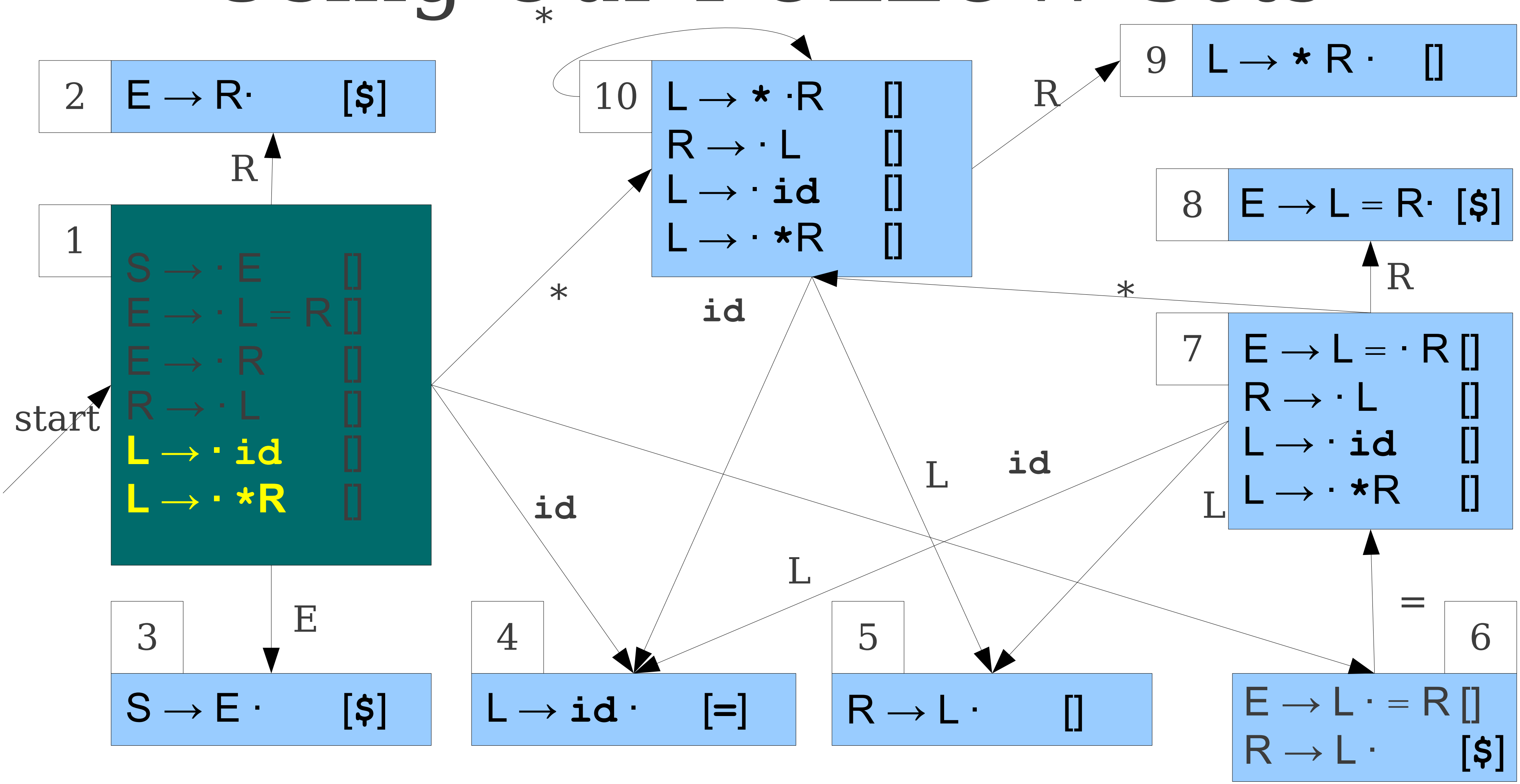
8
E → L = R· [$]

9
L → ***** R · []

10
L → ***** ·R []
R → · L []
L → · **id** []
L → · *****R []

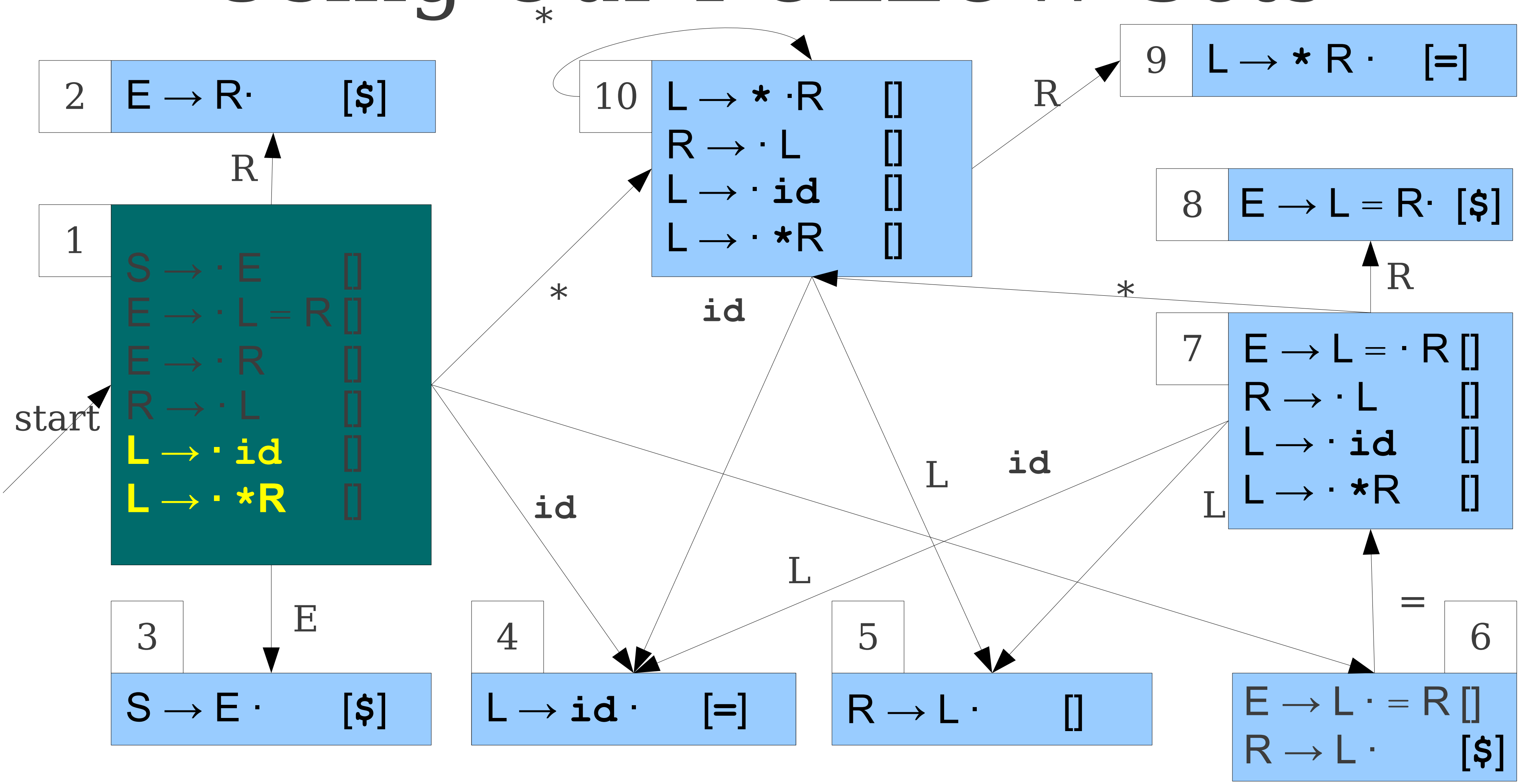
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · `id` []
L → · *R []

2
E → R· [$]

3
S → E · [$]

4
L → `id` · [=]

5
R → L · []

6
E → L · = R []
R → L · [$]

7
E → L = · R []
R → · L []
L → · `id` []
L → · *R []

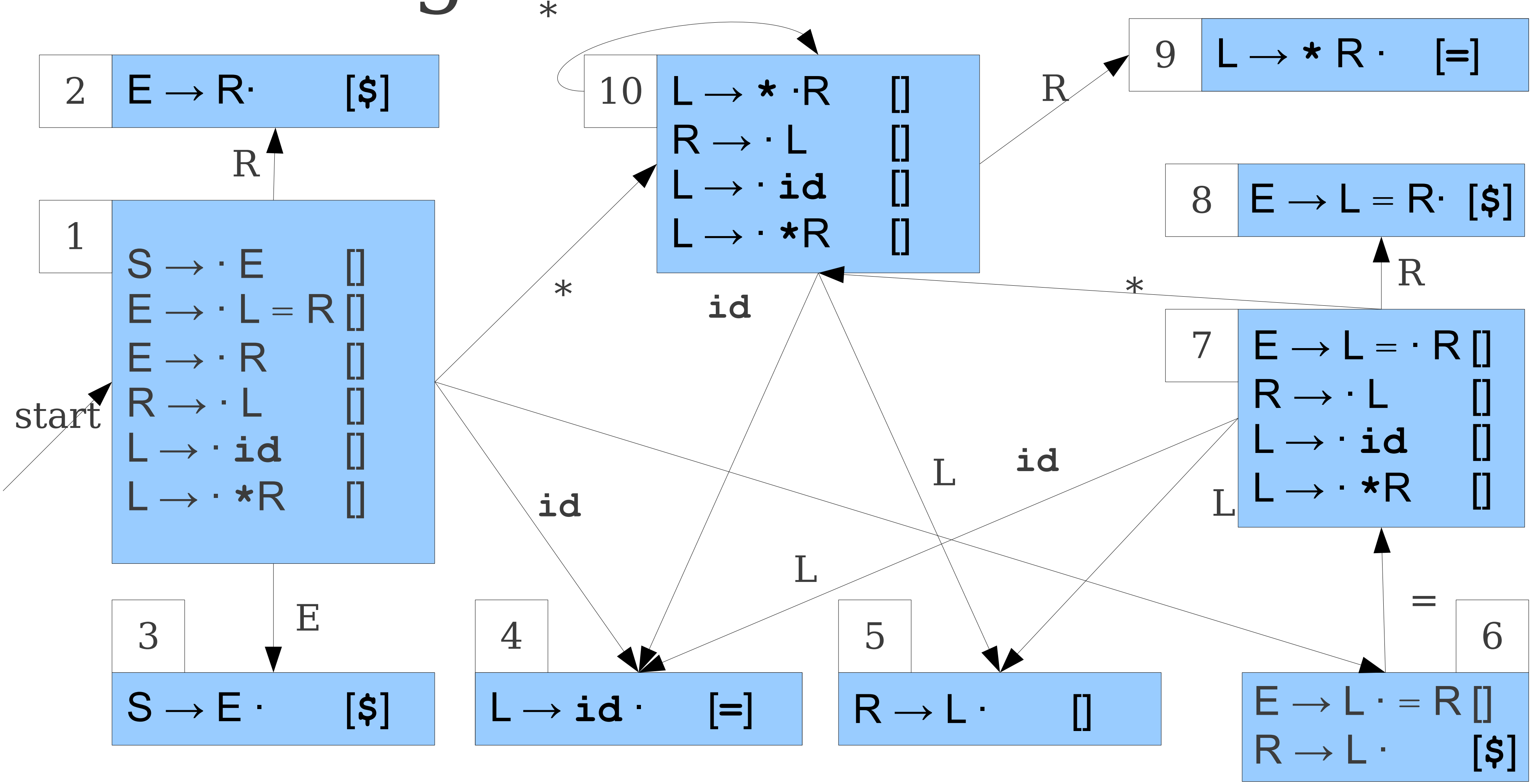
8
E → L = R· [$]

9
L → * R · [=]

10
L → * ·R []
R → · L []
L → · `id` []
L → · *R []

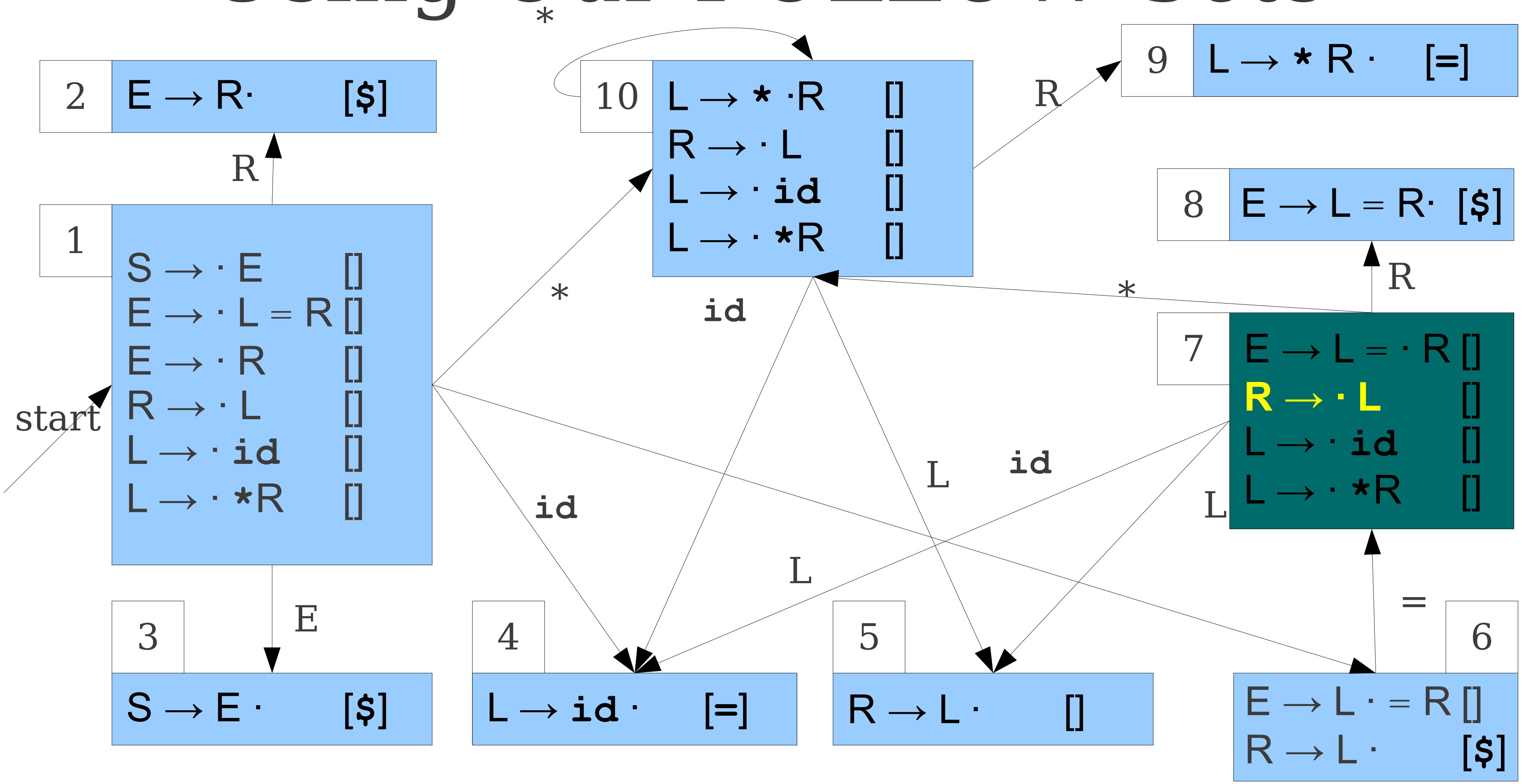
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · **id** []
L → · *R []

2
E → R· [$]

3
S → E · [$]

4
L → **id** · [=]

5
R → L · []

6
E → L · = R []
R → L · [$]

7
E → L = · R []
R → · L []
L → · **id** []
L → · *R []

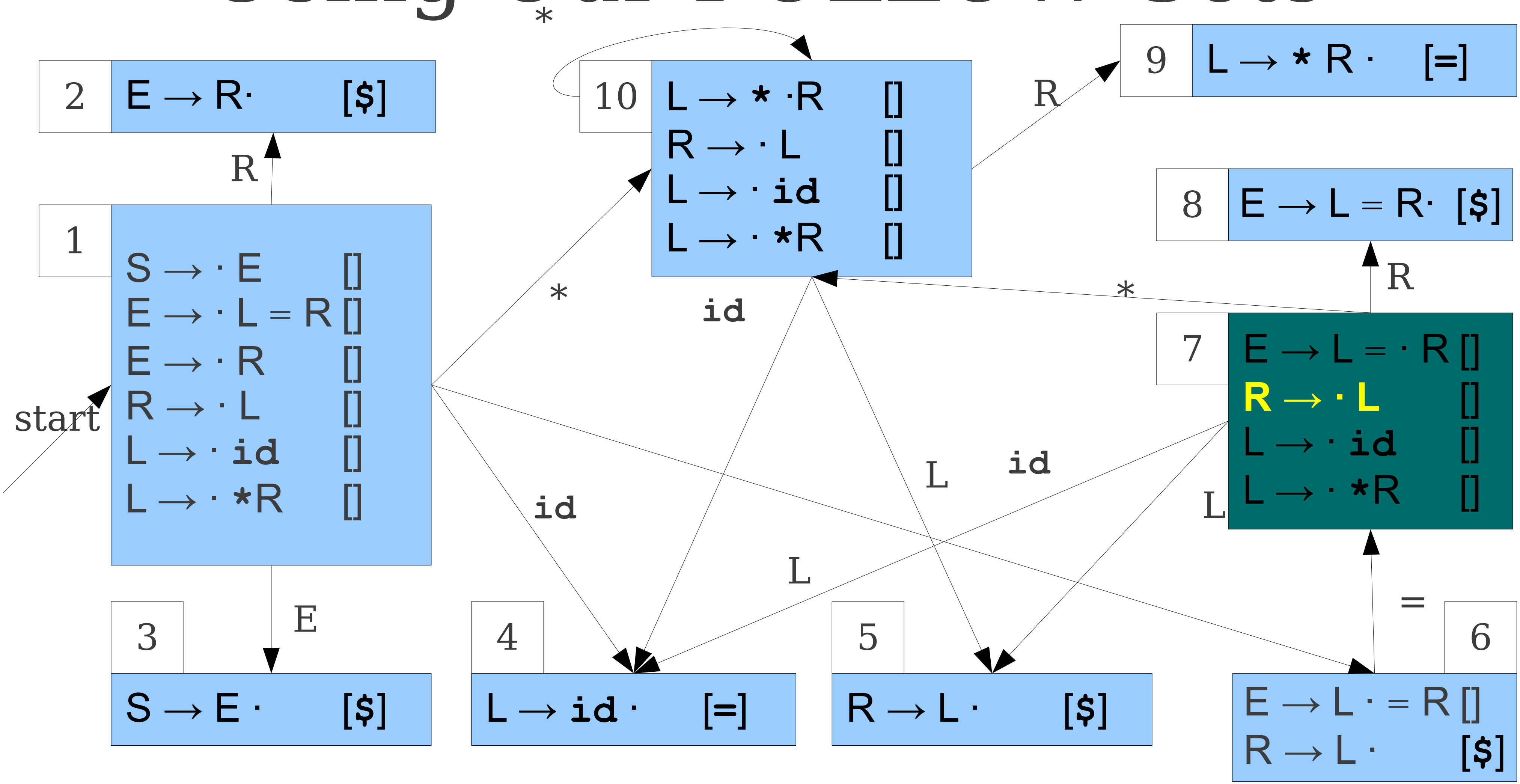
8
E → L = R· [$]

9
L → * R · [=]

10
L → * ·R []
R → · L []
L → · **id** []
L → · *R []

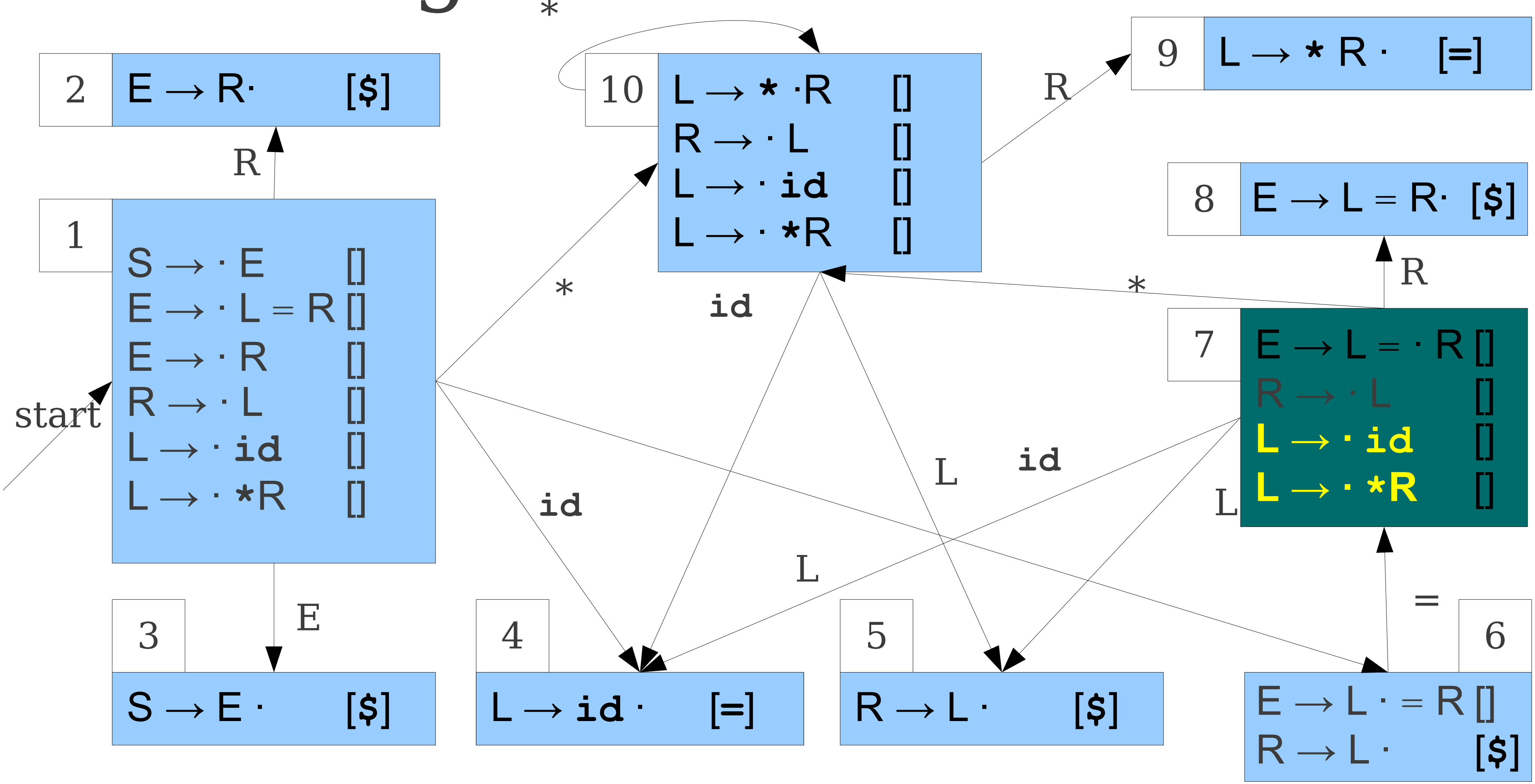
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



2
E → R· [$]

10
L → * ·R []
R → · L []
L → · **id** []
L → · *R []

9
L → * R · [=]

8
E → L = R· [$]

1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · **id** []
L → · *R []

7
E → L = · R []
R → · L []
L → · **id** []
L → · *R []

3
S → E · [$]

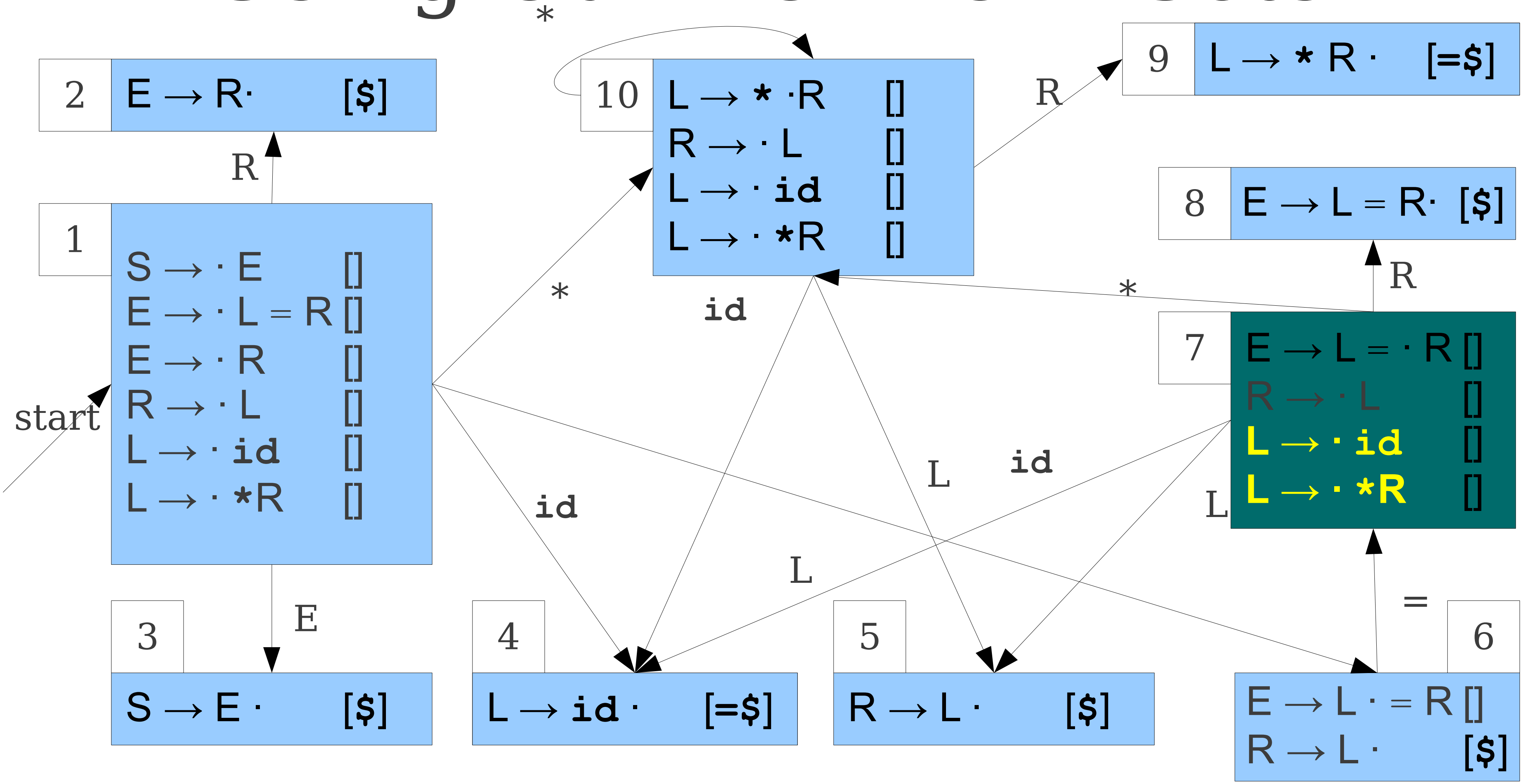
4
L → **id** · [=]

5
R → L · [$]

6
E → L · = R []
R → L · [$]

S_1	E_{1-3}	L_{1-6}	L_{7-5}	L_{10-5}	R_{1-2}	R_{7-8}	R_{10-9}
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1

S → · E []

E → · L = R []

E → · R []

R → · L []

L → · **id** []

L → · *R []

2

E → R· [$]

3

S → E · [$]

4

L → **id** · [=$]

5

R → L · [$]

6

E → L · = R []

R → L · [$]

7

E → L = · R []

R → · L []

L → · **id** []

L → · *R []

8

E → L = R· [$]

9

L → * R · [=$]

10

L → * ·R []

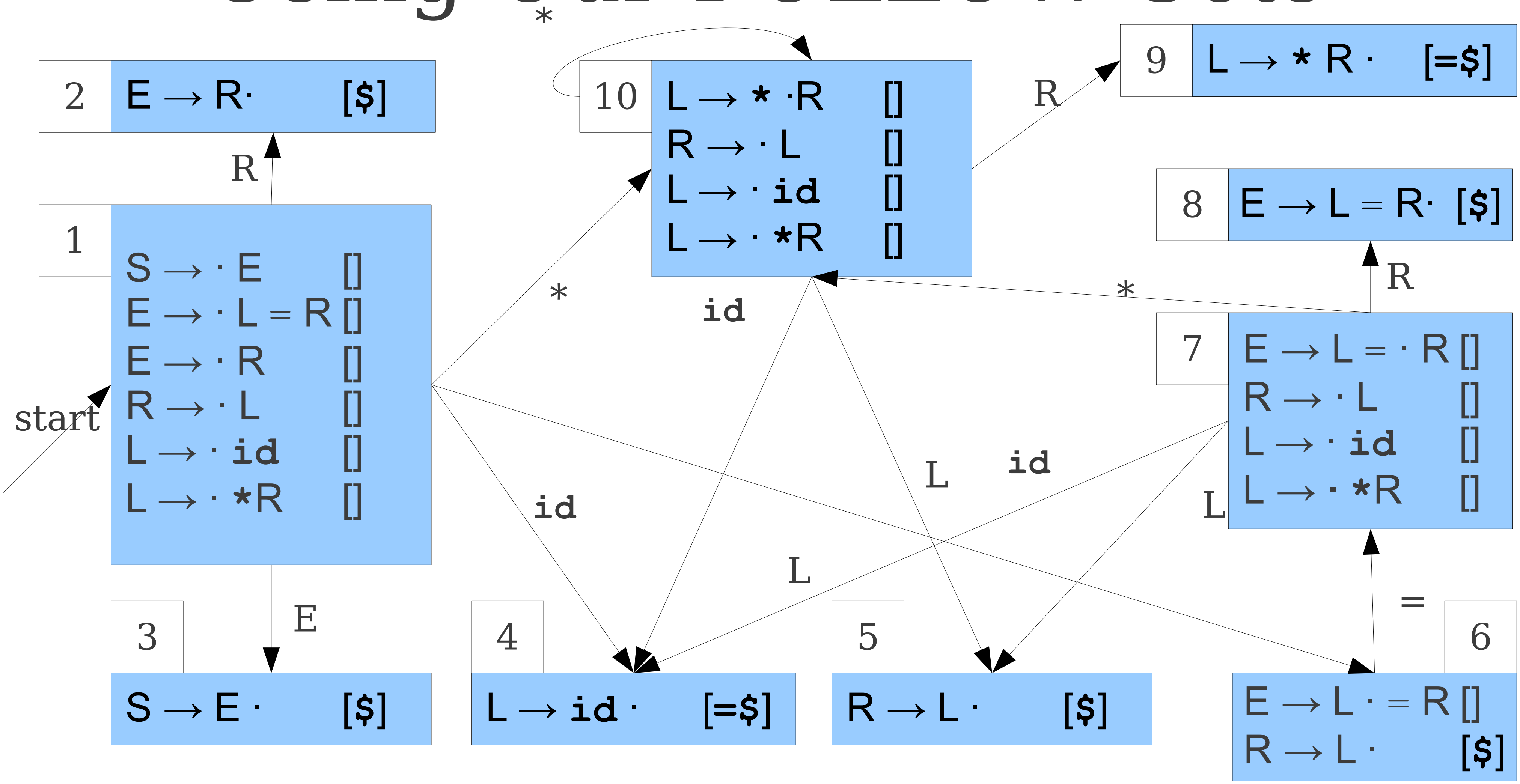
R → · L []

L → · **id** []

L → · *R []

S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · **id** []
L → · ***R** []

2
E → R· [$]

3
S → E · [$]

4
L → **id** · [=$]

5
R → L · [$]

6
E → L · = R []
R → L · [$]

7
E → L = · R []
R → · L []
L → · **id** []
L → · ***R** []

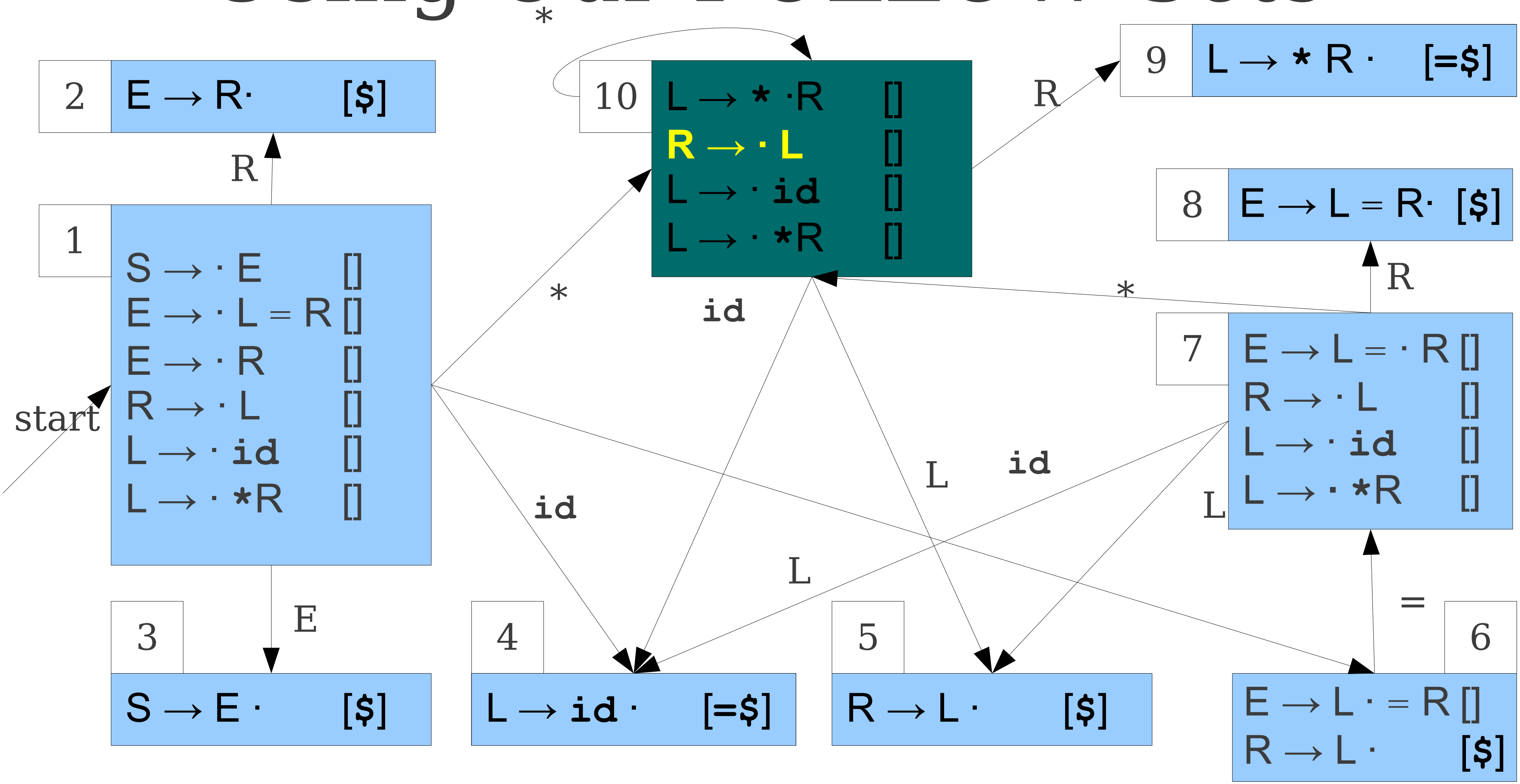
8
E → L = R· [$]

9
L → * R · [=$]

10
L → * ·R []
R → · L []
L → · **id** []
L → · ***R** []

S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · **id** []
L → · ***R** []

2
E → R· [$]

3
S → E · [$]

4
L → **id** · [=$]

5
R → L · [$]

6
E → L · = R []
R → L · [$]

7
E → L = · R []
R → · L []
L → · **id** []
L → · ***R** []

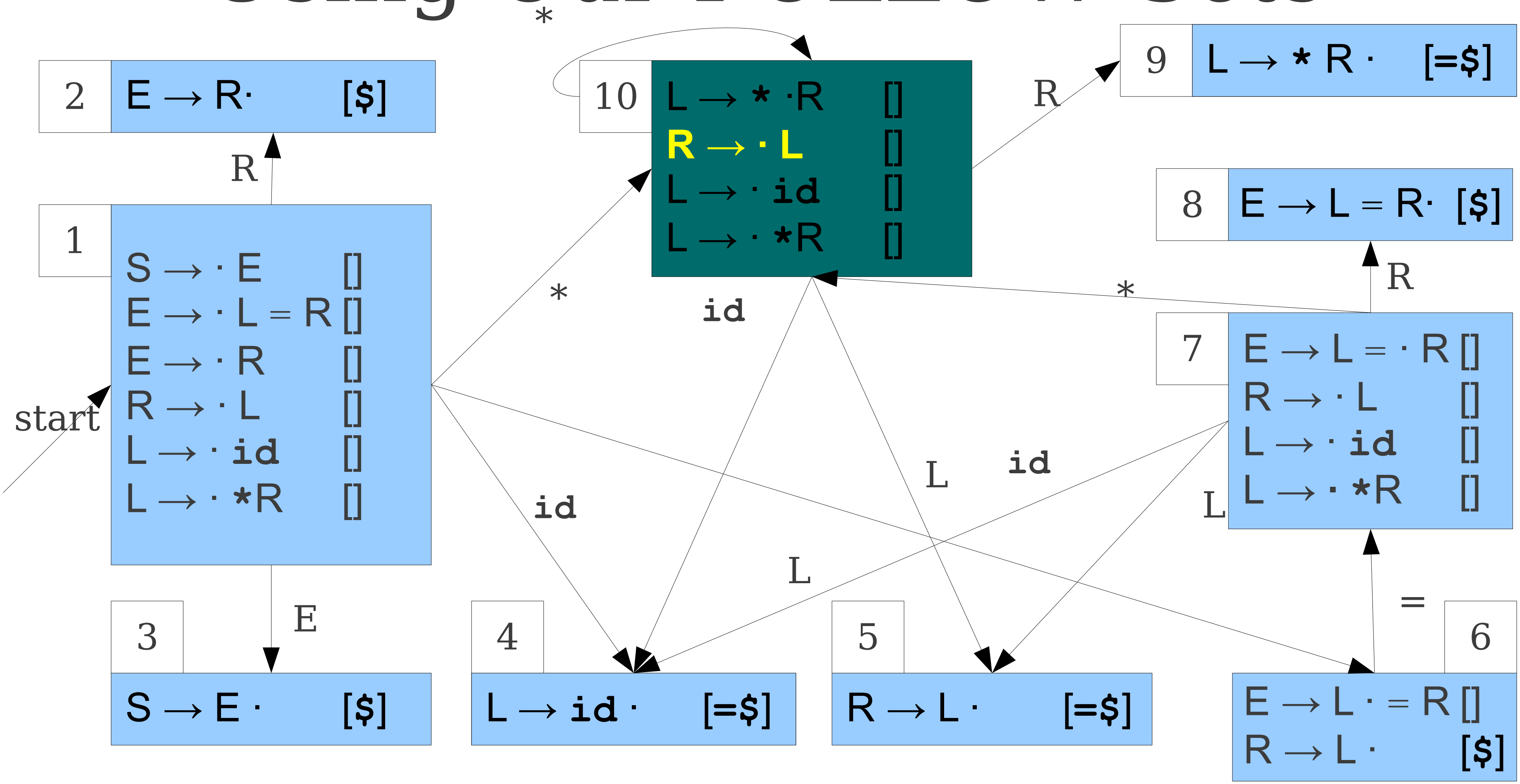
8
E → L = R· [$]

9
L → * R · [=$]

10
L → * ·R []
R → · L []
L → · **id** []
L → · ***R** []

S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



1
S → · E []
E → · L = R []
E → · R []
R → · L []
L → · **id** []
L → · *****R []

2
E → R· [$]

3
S → E · [$]

4
L → **id** · [=$]

5
R → L · [=$]

6
E → L · = R []
R → L · [$]

7
E → L = · R []
R → · L []
L → · **id** []
L → · *****R []

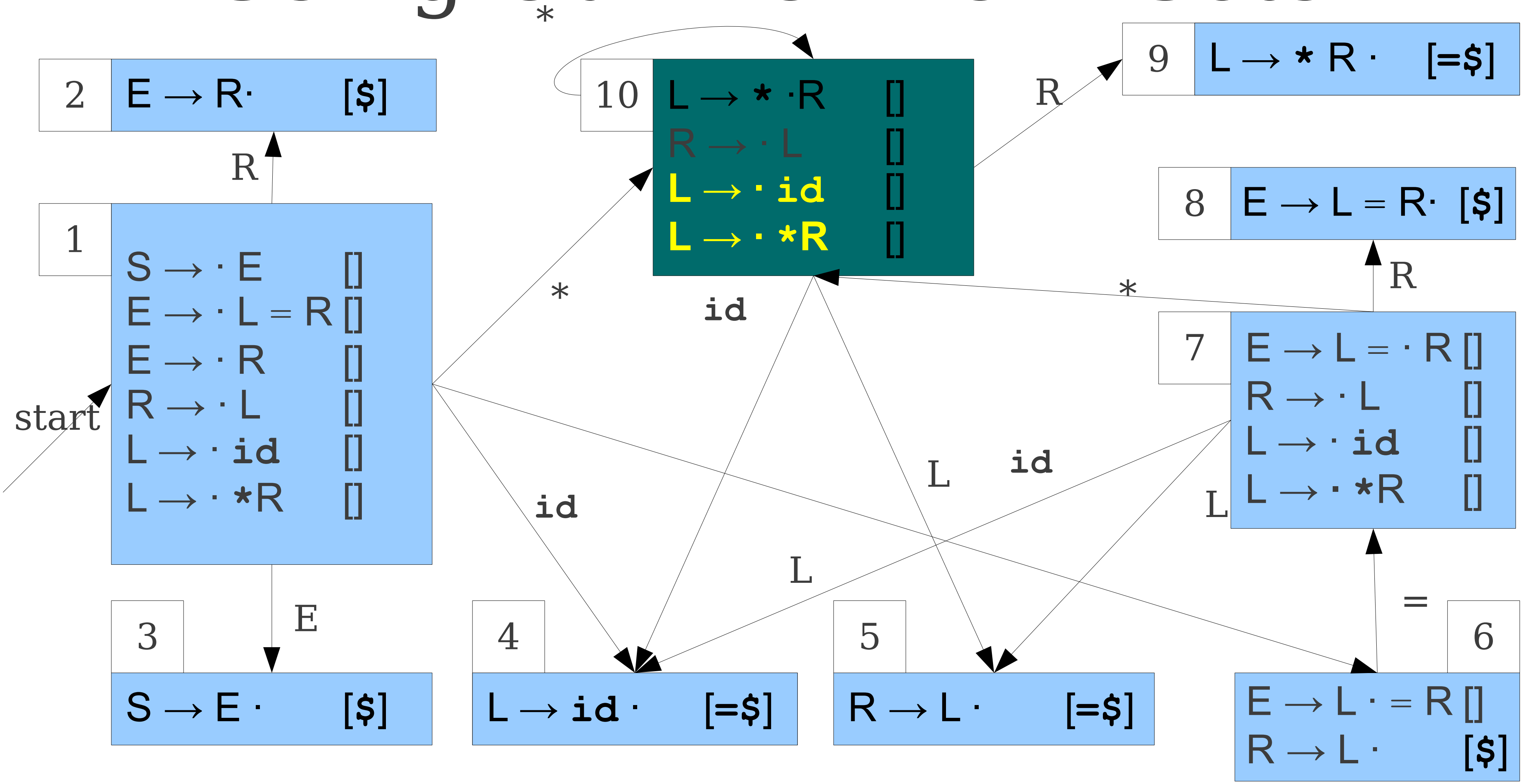
8
E → L = R· [$]

9
L → ***** R · [=$]

10
L → ***** ·R []
R → · L []
L → · **id** []
L → · *****R []

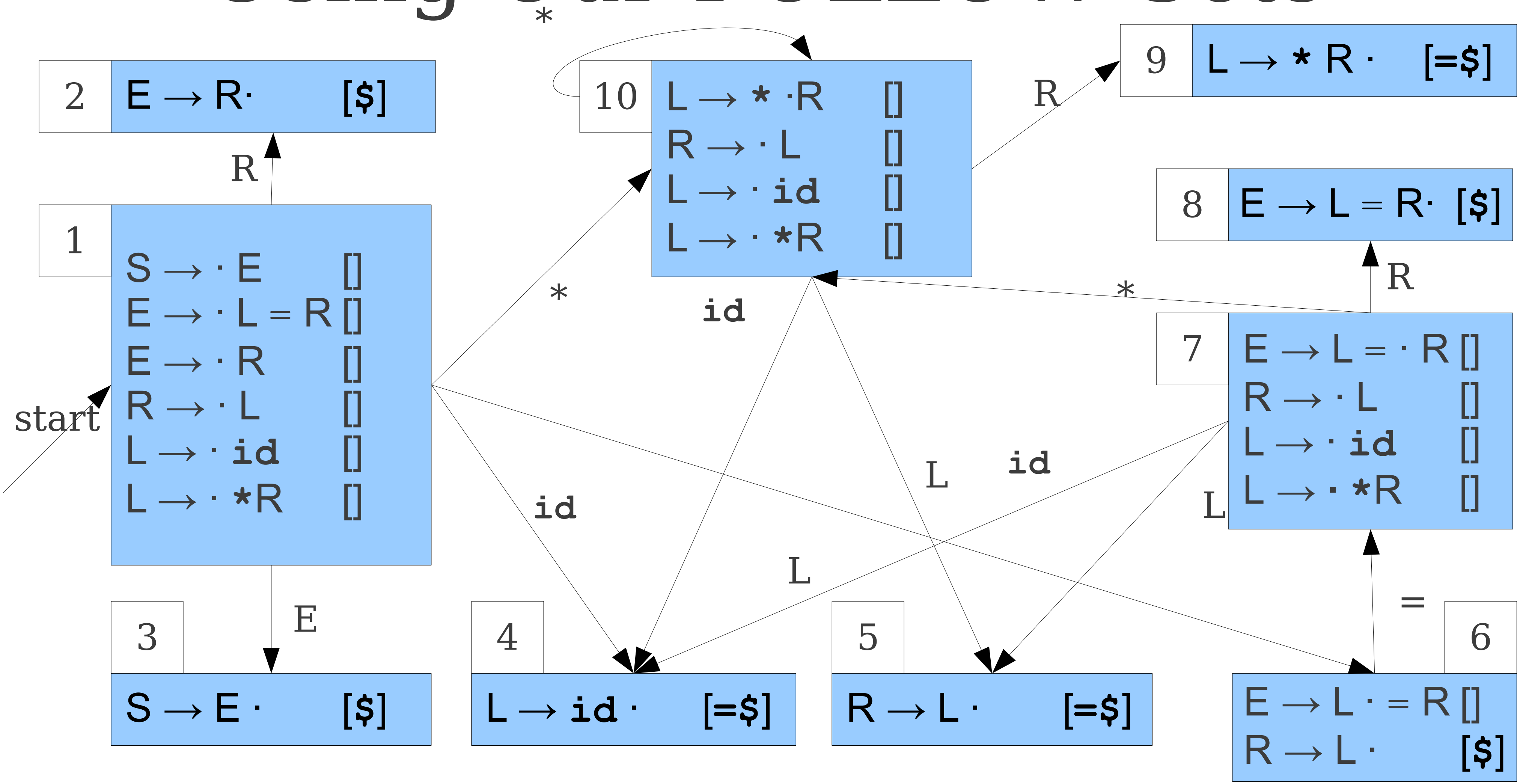
S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Using Our FOLLOW Sets



S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
\$	\$	=	\$	=	\$	\$	=

Using Our FOLLOW Sets



1
S → ·E []
E → ·L = R []
E → ·R []
R → ·L []
L → ·**id** []
L → ·***R** []

2
E → R· [$]

3
S → E· [$]

4
L → **id**· [=$]

5
R → L· [=$]

6
E → L· = R []
R → L· [$]

7
E → L = ·R []
R → ·L []
L → ·**id** []
L → ·***R** []

8
E → L = R· [$]

9
L → ***** R· [=$]

10
L → ***** ·R []
R → ·L []
L → ·**id** []
L → ·***R** []

S_1	$E_{1\text{-}3}$	$L_{1\text{-}6}$	$L_{7\text{-}5}$	$L_{10\text{-}5}$	$R_{1\text{-}2}$	$R_{7\text{-}8}$	$R_{10\text{-}9}$
$	$	=	$	=	$	$	=

Propagating Changes

- For each item $\mathbf{A} \rightarrow \cdot\boldsymbol{\omega}$ in a state q:
  - Let $\mathbf{A}_{q - r}$ be the nonterminal corresponding to $\mathbf{A}$ following the transition out of q into some state r.
  - Trace through the automaton along the path labeled by $\boldsymbol{\omega}$. This will lead to a state containing an item $\mathbf{A} \rightarrow \boldsymbol{\omega}\cdot$.
  - Add to the lookahead of $\mathbf{A} \rightarrow \boldsymbol{\omega} \cdot$ the contents of FOLLOW($\mathbf{A}_{q - r}$)

LALR(1)-by-SLR(1)

- Fast and simple construction of LALR(1) lookaheads:
- Construct the LR(0) automaton for the grammar.
    - Construct the augmented grammar by replacing nonterminals with new nonterminals based on the LR(0) transitions.
    - Compute the FOLLOW sets for these nonterminals.
    - Propagate changes through the LR(0) automaton.
- **Theorem** *(Bermudez and Logothetis)*: This correctly computes LALR(1) lookaheads.

Summary of LALR(1)

- Along with LL(*k*), one of the most popular parsing algorithms in use today.
- Produced by the **`bison`** parser generator; rarely generated by hand.
- Can handle most, but not all, LR(1) languages.

Practical Concerns

Where Theory Meets Practice

- We've just covered six powerful parsing algorithms:
  - Leftmost DFS
  - LL(1)
  - LR(0)
  - SLR(1)
  - LALR(1)
  - LR(1)
- How do we make them work in practice?

Two Practical Concerns

- **Ambiguity**
  - Real grammars are often ambiguous.
  - Programmers are *terrible* at eliminating it.
  - How do you build a parser to try to combat it?
- **Error-handling**
  - How do you report errors intelligently?
  - How do you continue parsing after an error?

Ambiguity and Predictive Parsing

- The predictive parsers we have seen so far (LL(1), LR(0), SLR(1), LALR(1), LR(1)) only work on unambiguous grammars.
  - Intuitively: if grammar is ambiguous, cannot uniquely guess which production/reduction to use.
  - Formally proving this is somewhat involved.
- Most grammars for programming languages, unless cleverly written, are ambiguous.
- How can we handle this?

Parsing Ambiguous Grammars

- Consider this simple grammar for arithmetic expressions:

  S → **E**

  E → **E** `+` **E**

  E → **E** `*` **E**

  E → `int`

  E → `(`**E**`)`

- This grammar is ambiguous.
  - e.g. Two trees for `int + int * int`
- What happens if we try parsing it?

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

FOLLOW(**S**) = { $ }
FOLLOW(**E**) = { +, *,), $ }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10						
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2							
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3					
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

FOLLOW(S) = { $ }
FOLLOW(E) = { +, *,), $ }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4				
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3							
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10						
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }

FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4							
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

FOLLOW(**S**) = { $ }
FOLLOW(**E**) = { +, *,) , $ }

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4							
5							
6							
7							
8							
9							
10							

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

SLR(1) Parsing with Ambiguity

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

FOLLOW(S) = { $ }
FOLLOW(E) = { +, *,), $ }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10						
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

FOLLOW(**S**) = { $ }
FOLLOW(**E**) = { +, *,), $ }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5							
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3					
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3	s4				
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4				
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2				
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2		
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

FOLLOW(**S**) = { $ }
FOLLOW(**E**) = { +, *,), $ }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6							
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3					
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → `int`

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3	s4				
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4				
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3				
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

FOLLOW(**S**) = { $ }
FOLLOW(**E**) = { +, *,), $ }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3		
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. S → E
2. E → E + E
3. E → E * E
4. E → (E)
5. E → int

FOLLOW(S) = { $ }
FOLLOW(E) = { +, *,), $ }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7							
8							
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10						
8							
9							
10							

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E + E**
3. **E** → **E * E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			
8							
9							
10							

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8							
9							
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3					
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4				
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)**, **$** }

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9							
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4					
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4				
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1
S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2
S → E ·
E → E ·+ E
E → E ·* E

3
E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4
E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5
E → E + E ·
E → E · + E
E → E · * E

6
E → E * E ·
E → E · + E
E → E · * E

7
E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8
E → (E ·)
E → E ·+ E
E → E ·* E

9
E → (E) ·

10
E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4		
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10							

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10							

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5					

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

FOLLOW(**S**) = { $ }

FOLLOW(**E**) = { +, *,), $ }

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5				

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

FOLLOW(**S**) = { $ }
FOLLOW(**E**) = { +, *,), $ }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5		

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E**
3. **E** → **E** ***** **E**
4. **E** → **(E)**
5. **E** → **int**

FOLLOW(**S**) = { **$** }
FOLLOW(**E**) = { **+**, *****, **)** , **$** }

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

Resolving Ambiguity

- Although the grammar is ambiguous, there is clearly one intended parse tree because of operator precedence.
- How can we use this precedence information to avoid LR conflicts?

Precedence Declarations

- Tell the parser generator about the *associativity* and *precedence* of certain rules.
- Productions can be left-associative, right-associative, or nonassociative.
- Productions can have their priorities ranked against one another.

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** + **E**
3. **E** → **E** * **E**
4. **E** → (**E**)
5. **E** → int

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E** (Left-assoc, pri. 0)
3. **E** → **E** ***** **E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4 r2		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → `(`**E**`)`
5. **E** → `int`

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E** (Left-assoc, pri. 0)
3. **E** → **E** ***** **E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E** (Left-assoc, pri. 0)
3. **E** → **E** ***** **E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		s3 r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S** → **E**
2. **E** → **E** **+** **E** (Left-assoc, pri. 0)
3. **E** → **E** ***** **E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E** (Left-assoc, pri. 0)
3. **E** → **E** ***** **E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		r2	s4		r2	r2	
6		r3	r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E** (Left-assoc, pri. 0)
3. **E** → **E** ***** **E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		r2	s4		r2	r2	
6		r3	r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Rgt.-assoc, pri. 0)
3. **E** → **E * E** (Rgt.-assoc, pri. 1)
4. **E** → **`(E)`**
5. **E** → **`int`**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Rgt.-assoc, pri. 0)
3. **E** → **E * E** (Rgt.-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3 r2	s4		r2	r2	
6		r3	s4 r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

1. **S → E**
2. **E → E + E** (Rgt.-assoc, pri. 0)
3. **E → E * E** (Rgt.-assoc, pri. 1)
4. **E → (E)**
5. **E → int**

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3	s4		r2	r2	
6		r3	s4		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Rgt.-assoc, pri. 0)
3. **E** → **E * E** (Rgt.-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		s3	s4		r2	r2	
6		r3	s4		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

Resolving Conflicts with Precedence

- When choosing whether to reduce a rule containing `t` or shift the terminal `r`:
  - If `t` has higher priority, **reduce**.
  - If `r` has higher priority, **shift**.
  - If `t` and `r` have the same priority:
    - If `t` is left-associative, **reduce**.
    - If `t` is right-associative, **shift**.
    - If `t` is non-associative, **error**.

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E + E** (Left-assoc, pri. 0)
3. **E** → **E * E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		r2	s4		r2	r2	
6		r3	r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

SLR(1) Parsing with Ambiguity

1. **S** → **E**
2. **E** → **E** **+** **E** (Left-assoc, pri. 0)
3. **E** → **E** ***** **E** (Left-assoc, pri. 1)
4. **E** → **(E)**
5. **E** → **int**

1

S → · E
E → · E + E
E → · E * E
E → · int
E → · (E)

2

S → E ·
E → E ·+ E
E → E ·* E

3

E → E + ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

4

E → E * ·E
E → · E + E
E → · E * E
E → · int
E → · (E)

5

E → E + E ·
E → E · + E
E → E · * E

6

E → E * E ·
E → E · + E
E → E · * E

7

E → (·E)
E → · E + E
E → · E * E
E → · int
E → · (E)

8

E → (E ·)
E → E ·+ E
E → E ·* E

9

E → (E) ·

10

E → int ·

	int	+	*	(	)	$	E
1	s10			s7			s2
2		s3	s4			acc	
3	s10			s7			s5
4	s10			s7			s6
5		r2	s4		r2	r2	
6		r3	r3		r3	r3	
7	s10			s7			s8
8		s3	s4		s9		
9		r4	r4		r4	r4	
10		r5	r5		r5	r5	

Error Handling

- What should the parser do when it encounters an error?
- Could just say “syntax error,” but we'd like more detailed messages.
- How do we resume parsing after an error?

Error Productions

- One idea: add productions to the grammar that identify common mistakes.
- For example:

$$\mathbf{E} \rightarrow \mathbf{E} \; \texttt{+} \; \mathbf{E}$$

$$\mathbf{E} \rightarrow \mathbf{E} \; \texttt{*} \; \mathbf{E}$$

$$\mathbf{E} \rightarrow \texttt{int}$$

$$\mathbf{E} \rightarrow \texttt{(}\mathbf{E}\texttt{)}$$

$$\mathbf{E} \rightarrow \mathbf{E} \; \mathbf{E} \quad \textit{(error production)}$$

$$\mathbf{E} \rightarrow \mathbf{E} \; \texttt{+} \quad \textit{(error production)}$$

$$\mathbf{E} \rightarrow \mathbf{E} \; \texttt{*} \quad \textit{(error production)}$$

Analysis of Error Productions

- Useful for diagnosing common programmer mistakes.
  - For example, using **`implements`** instead of **`extends`** in Java.
- Increases risk of parsing problems.
  - More likelihood for ambiguity.
  - More likelihood grammar won't be accepted by parser generator (i.e. not LALR(1))
- Forces parser generator to anticipate errors.

Panic Mode

- Idea: Augment grammar by adding rules for resuming parsing when Bad Things happen.
- Example:

  E → **E** `+` **E**

  E → **E** `*` **E**

  E → `(`**E**`)`

  E → `int`

  E → **error** `int`

  E → `(`**error**`)`

- Tokens after errors are called **synchronizing tokens**.
- Technique employed by `bison` and many other parser generators.

Using Panic Mode

- When parser encounters an error in a configurating set, search for a production containing an error term.
    - Repeatedly shift tokens onto the stack until the synchronizing token is found.
    - Reduce using the error rule.
- Resume parsing as normal.