CS251: Introduction to Language Processing

Bottom-Up Parsing

Vishwesh Jatala

Department of CSE

Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

2023-24 M

Acknowledgement

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

$S \rightarrow E$

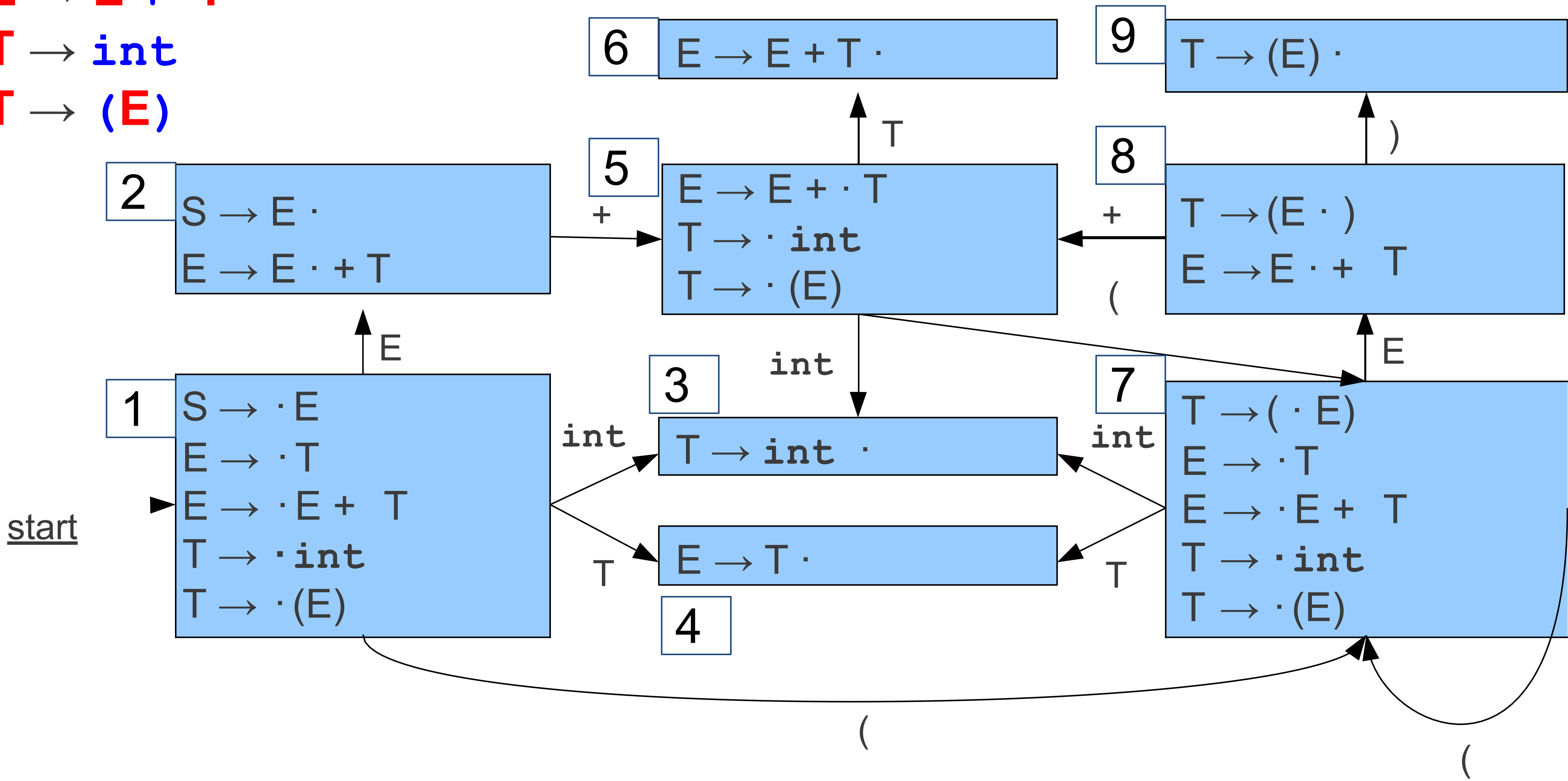
$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow$ `int`

$T \rightarrow (E)$

LR (0) Parsing



LR(0) Table

(1) $S \rightarrow E$
(2) $E \rightarrow T$
(3) $E \rightarrow E + T$
(4) $T \rightarrow$ `int`
(5) $T \rightarrow$ `(E)`

	Action					Goto	
	`int`	`+`	`(`	`)`	`$`	E	T
1	S3					2	4
2	r1	S5/r1	r1	r1	r1		
3	r4	r4	r4	r4	r4		
4	r2	r2	r2	r2	r2		
5	S3		S7				6
6	r3	r3	r3	r3	r3		
7	S3		S7			8	4
8		S5		S9			
9	r5	r5	r5	r5	r5		

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?

Example

(1) $S \longrightarrow E$

(2) $E \longrightarrow T$

(3) $E \longrightarrow E + T$

(4) $T \longrightarrow$ `int`

(5) $T \longrightarrow$ `(E)`

Try to parse int+int using LR(0)

SLR(1)

- **Simple LR(1)**
- Minor modification to LR(0) automaton that uses lookahead to avoid shift/reduce conflicts.

$S \rightarrow E$

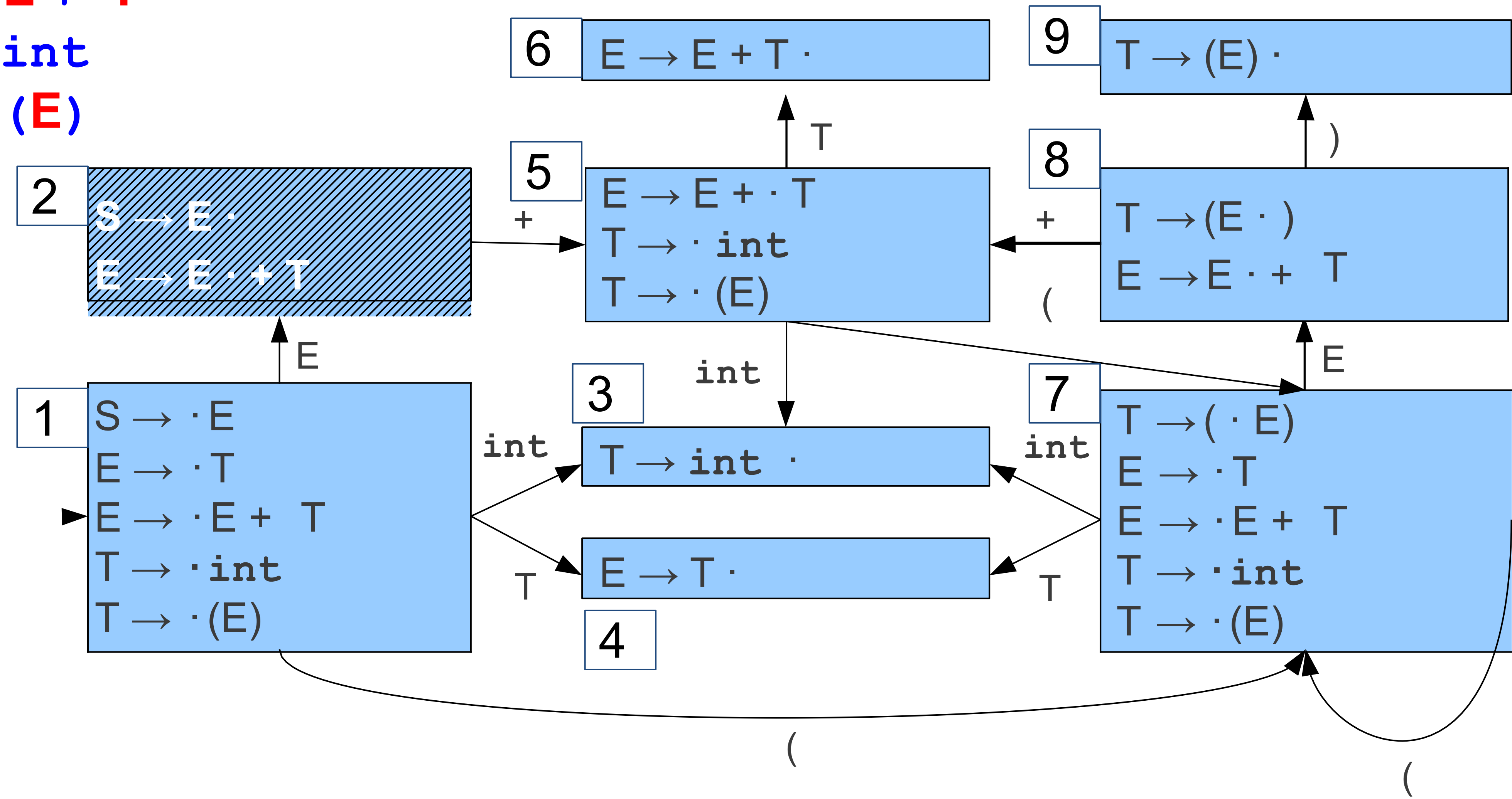
$E \rightarrow T$

$E \rightarrow E + T$

$T \rightarrow$ `int`

$T \rightarrow (E)$

SLR(1) Parsing



SLR(1) Table

(1) $S \to E$
(2) $E \to T$
(3) $E \to E + T$
(4) $T \to$ `int`
(5) $T \to$ `(E)`

	Action					Goto	
	int	+	(	)	$	E	T
1	S3					2	4
2		S5					
3							
4							
5	S3		S7				6
6							
7	S3		S7			8	4
8		S5		S9			
9							

SLR(1) Table

(1) $S \rightarrow E$

(2) $E \rightarrow T$

(3) $E \rightarrow E + T$

(4) $T \rightarrow$ `int`

(5) $T \rightarrow$ `(`E`)`

	Action					Goto	
	`int`	`+`	`(`	`)`	`$`	E	T
1	S3					2	4
2		S5			**r1**		
3		**r4**		**r4**	**r4**		
4		**r2**		**r2**	**r2**		
5	S3		S7				6
6		**r3**		**r3**	**r3**		
7	S3		S7			8	4
8		S5		S9			
9		**r5**		**r5**	**r5**		

SLR(1)

- **Simple LR(1)**
- Idea: Only reduce **A** $\rightarrow$ ***ω*** if the next token **`t`** is in FOLLOW(**A**).
- Automaton identical to LR(0) automaton; only change is when we choose to reduce.
-

Example

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

Try to parse int+int using SLR(1)

Analysis of SLR(1)

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

The Limits of SLR(1)

$\mathbf{S} \to \mathbf{E}$

$\mathbf{E} \to \mathbf{L}$ `=` $\mathbf{R}$

$\mathbf{E} \to \mathbf{R}$

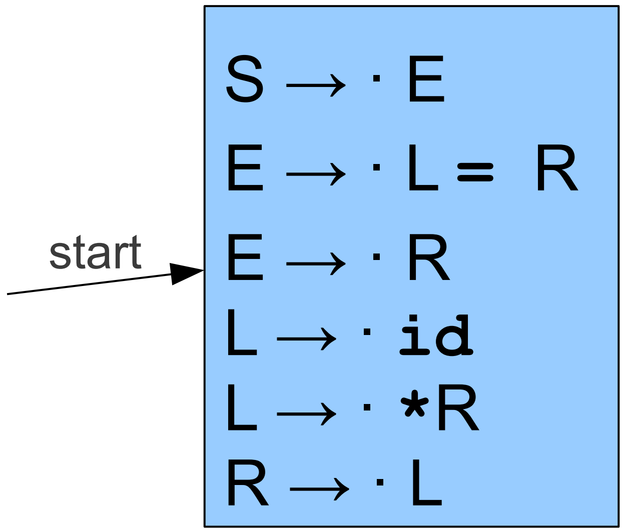
$\mathbf{L} \to$ `id`

$\mathbf{L} \to$ `*`$\mathbf{R}$

$\mathbf{R} \to \mathbf{L}$

The Limits of SLR(1)

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



The Limits of SLR(1)

$S \rightarrow E$

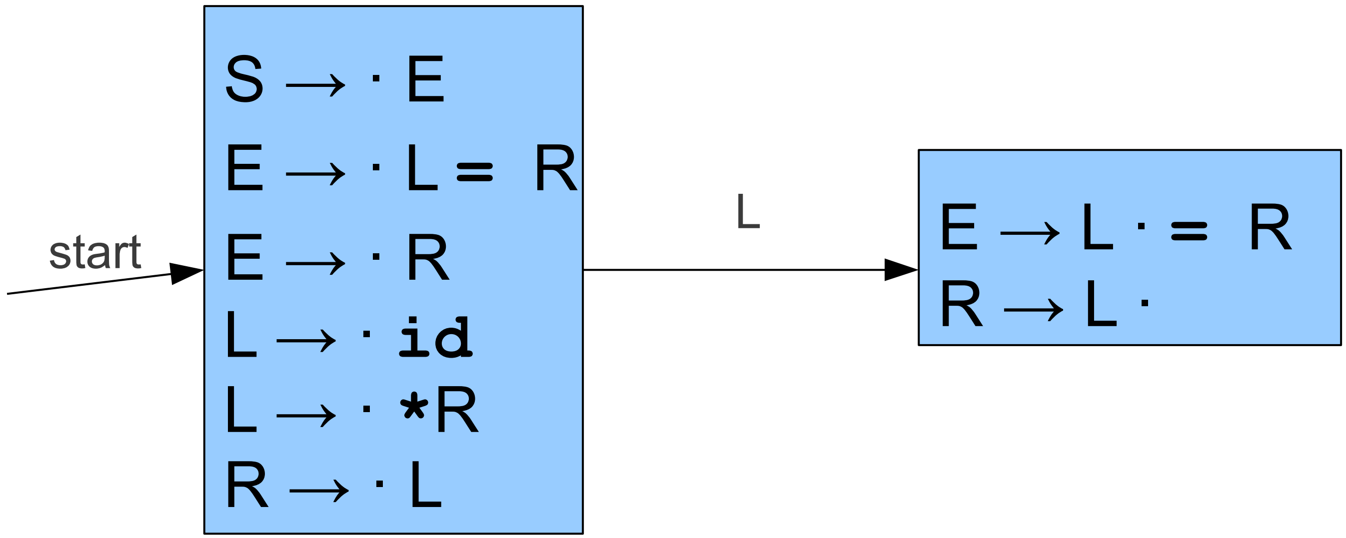
$E \rightarrow L = R$

$E \rightarrow R$

$L \rightarrow \mathtt{id}$

$L \rightarrow *R$

$R \rightarrow L$



The Limits of SLR(1)

S → E

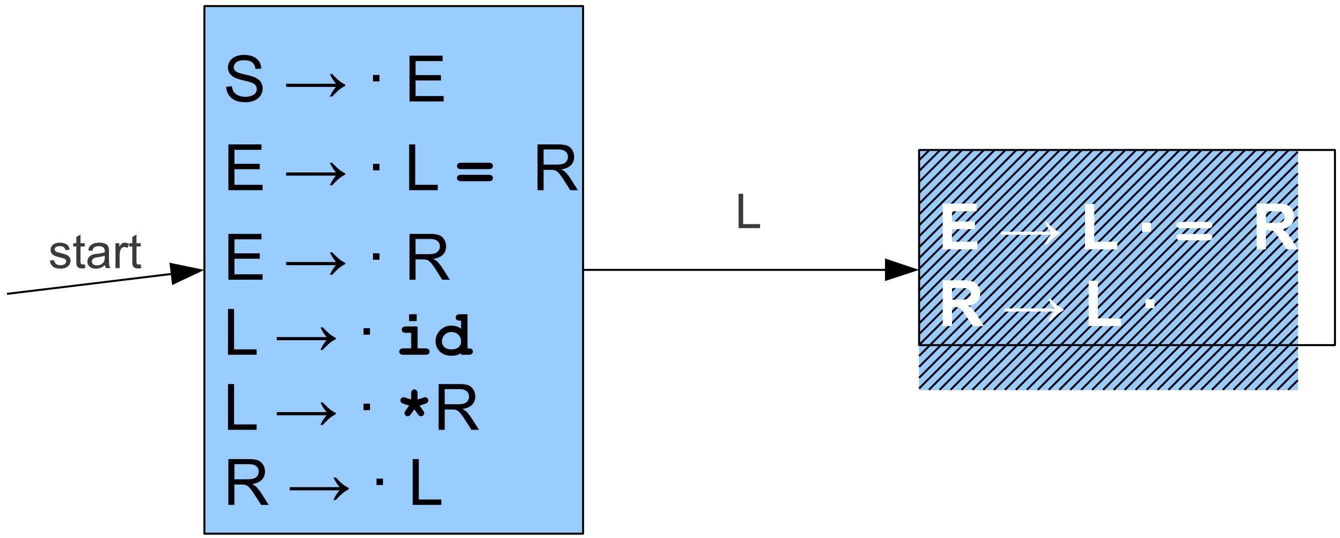
E → L = R

E → R

L → id

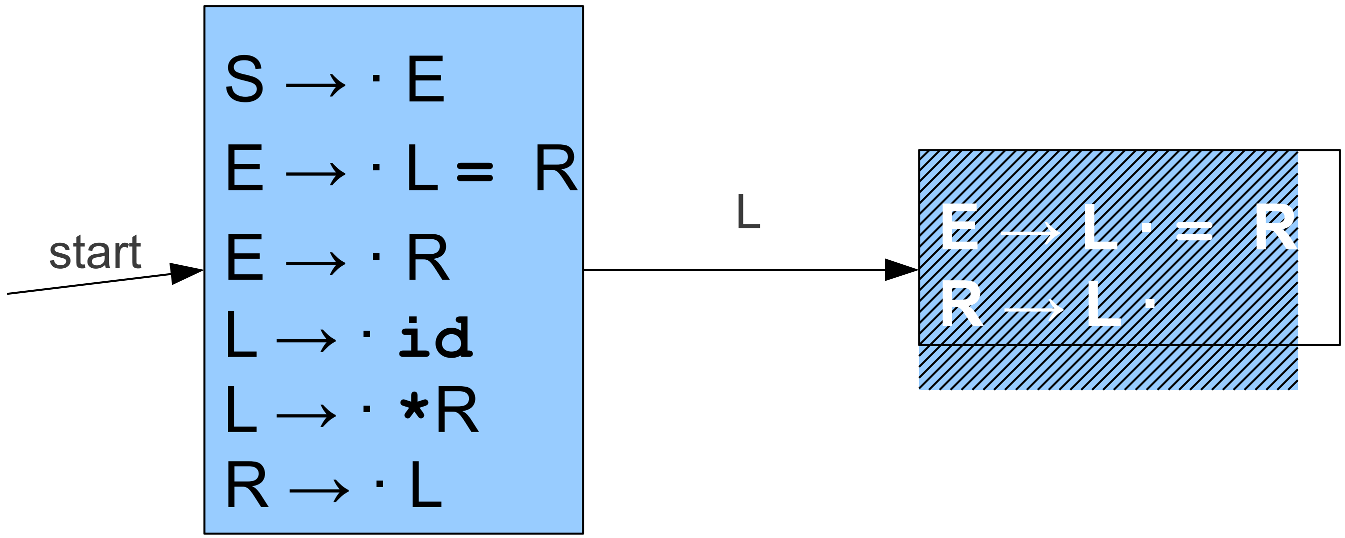
L → *R

R → L



The Limits of SLR(1)

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

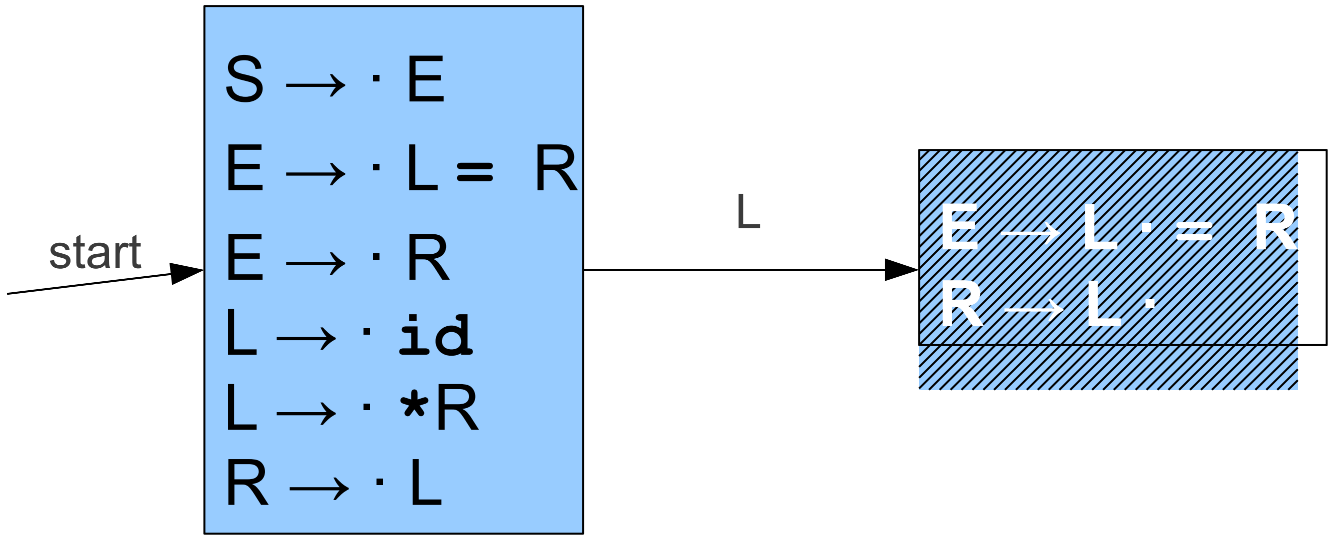


$E \rightarrow L \cdot = R$ tells us to shift on seeing $=$

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

The Limits of SLR(1)

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



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

The Limits of SLR(1)

$S \rightarrow E$

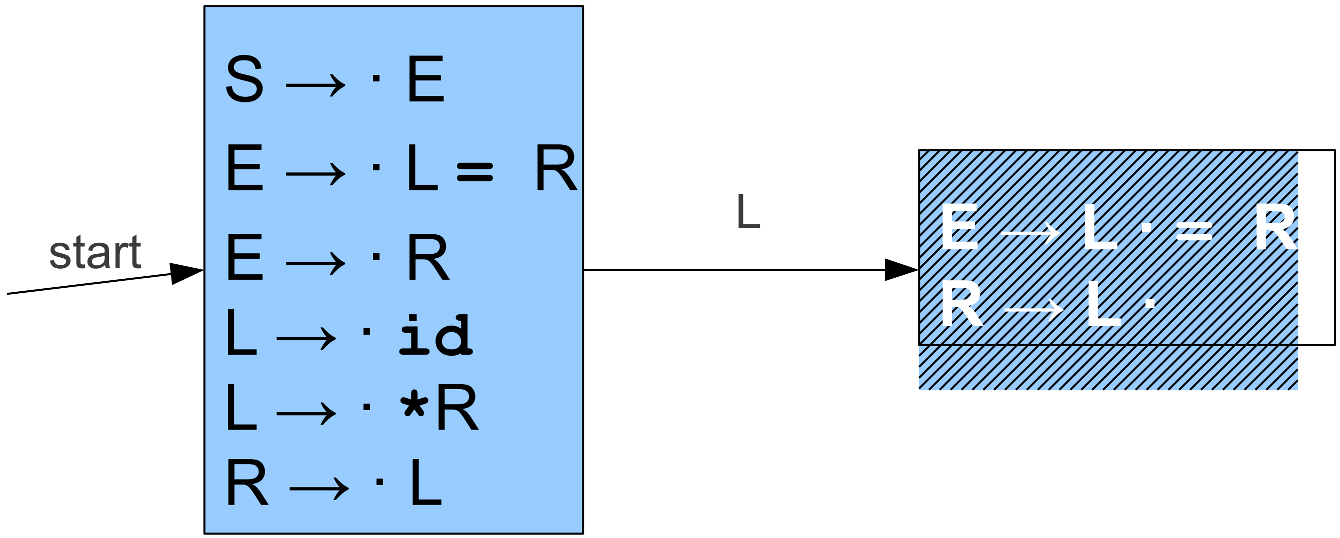
$E \rightarrow L = R$

$E \rightarrow R$

$L \rightarrow \texttt{id}$

$L \rightarrow *R$

$R \rightarrow L$



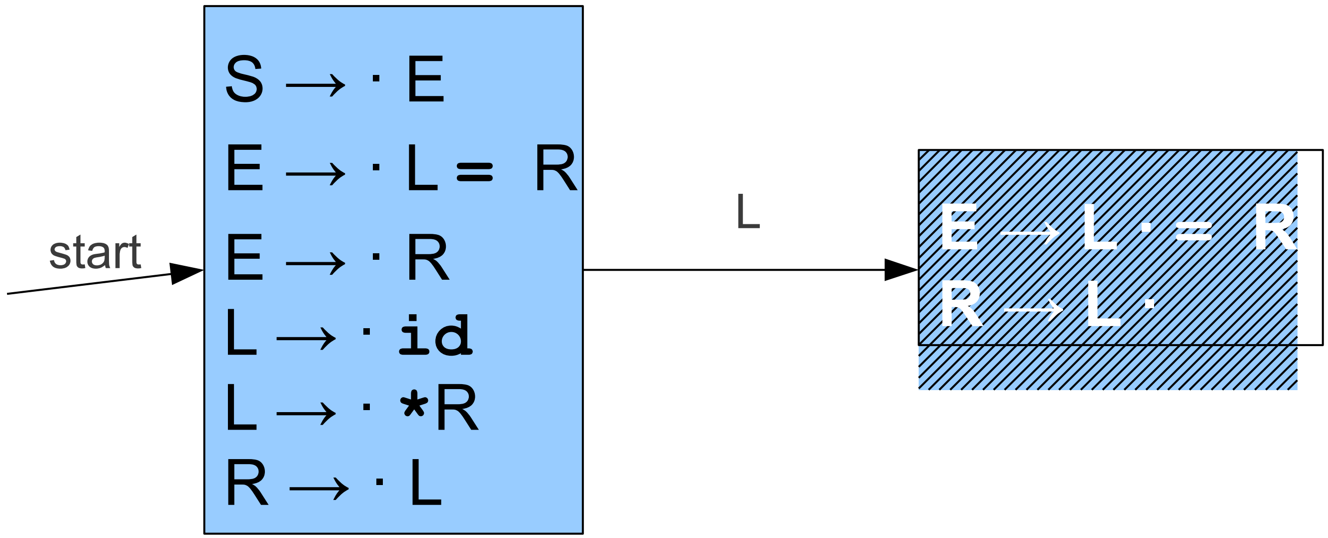
$E \rightarrow L \cdot = R$ tells us to shift on seeing $=$

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

$= \in$ FOLLOW(R).

The Limits of SLR(1)

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



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

$= \in$ FOLLOW(R).

We have a conflict!

A Lack of Context

$S \rightarrow E$

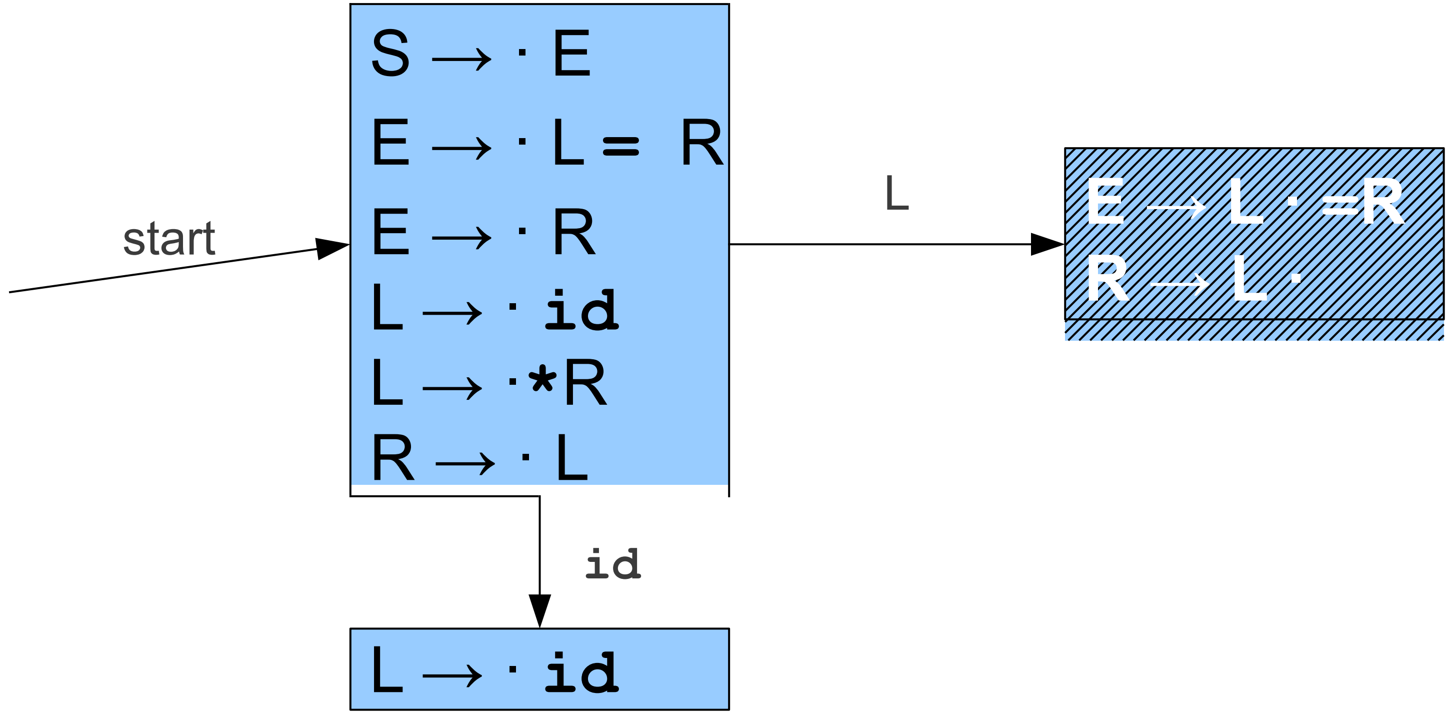
$E \rightarrow L$ `=` R

$E \rightarrow R$

$L \rightarrow$ `id`

$L \rightarrow$ `*`R

$R \rightarrow L$



A Lack of Context

$S \rightarrow E$

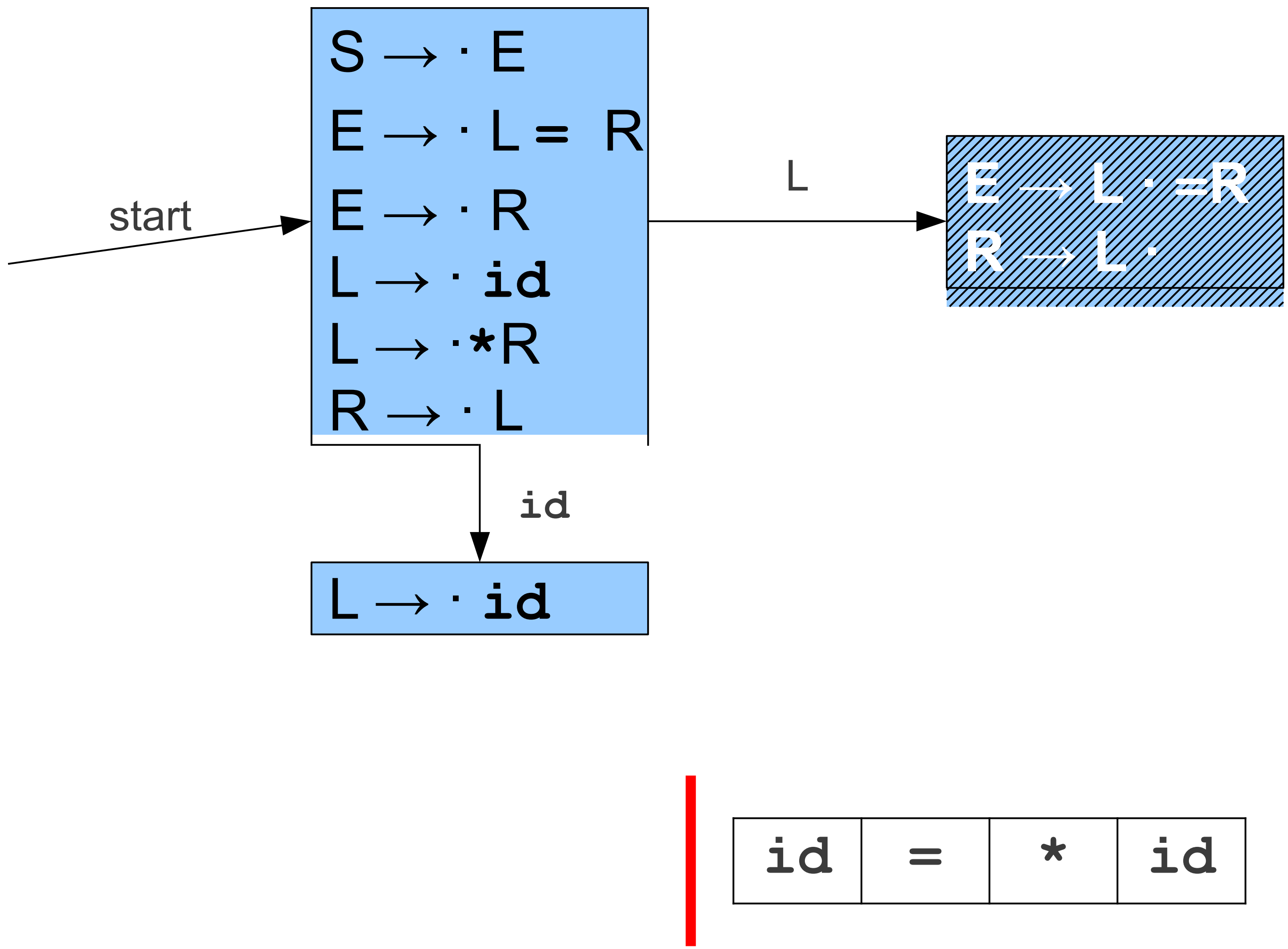
$E \rightarrow L = R$

$E \rightarrow R$

$L \rightarrow \texttt{id}$

$L \rightarrow *R$

$R \rightarrow L$



A Lack of Context

$S \rightarrow E$

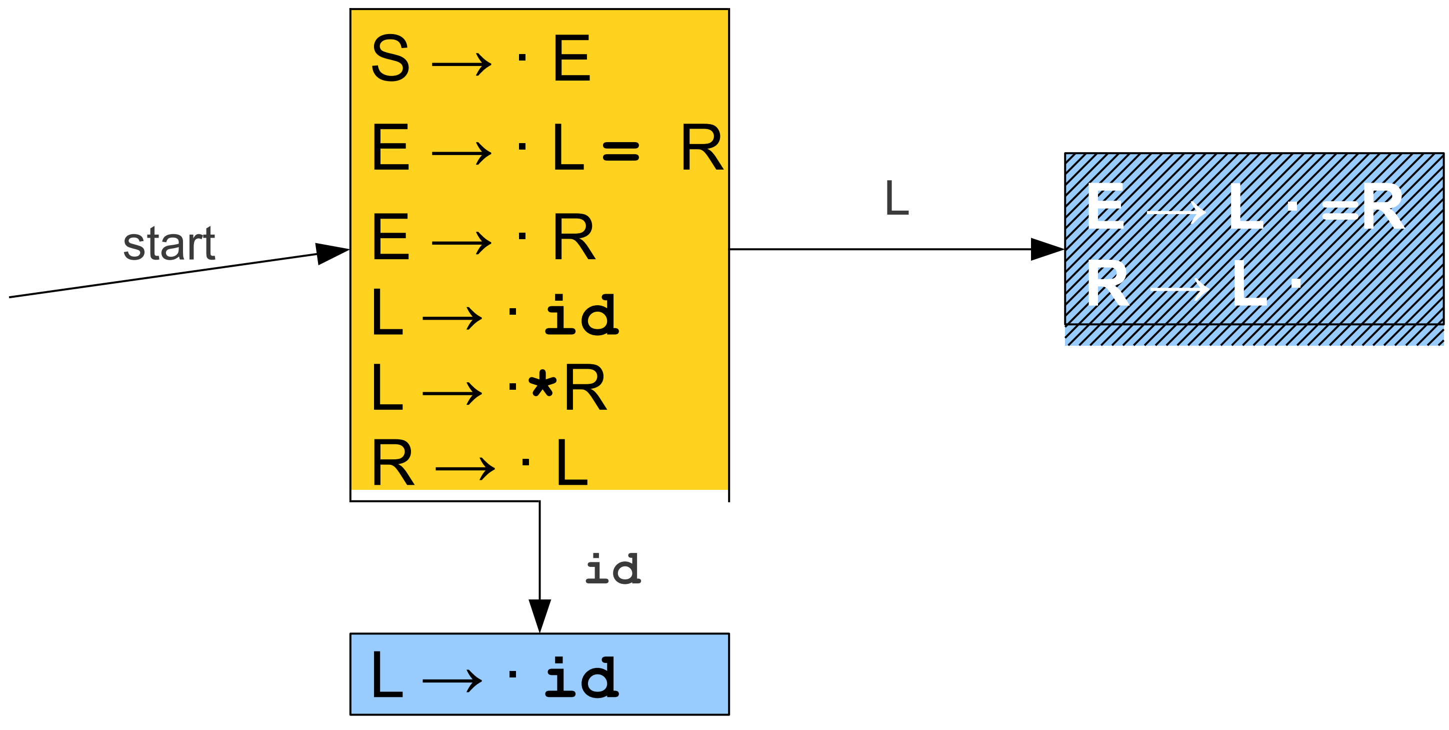
$E \rightarrow L = R$

$E \rightarrow R$

$L \rightarrow \texttt{id}$

$L \rightarrow *R$

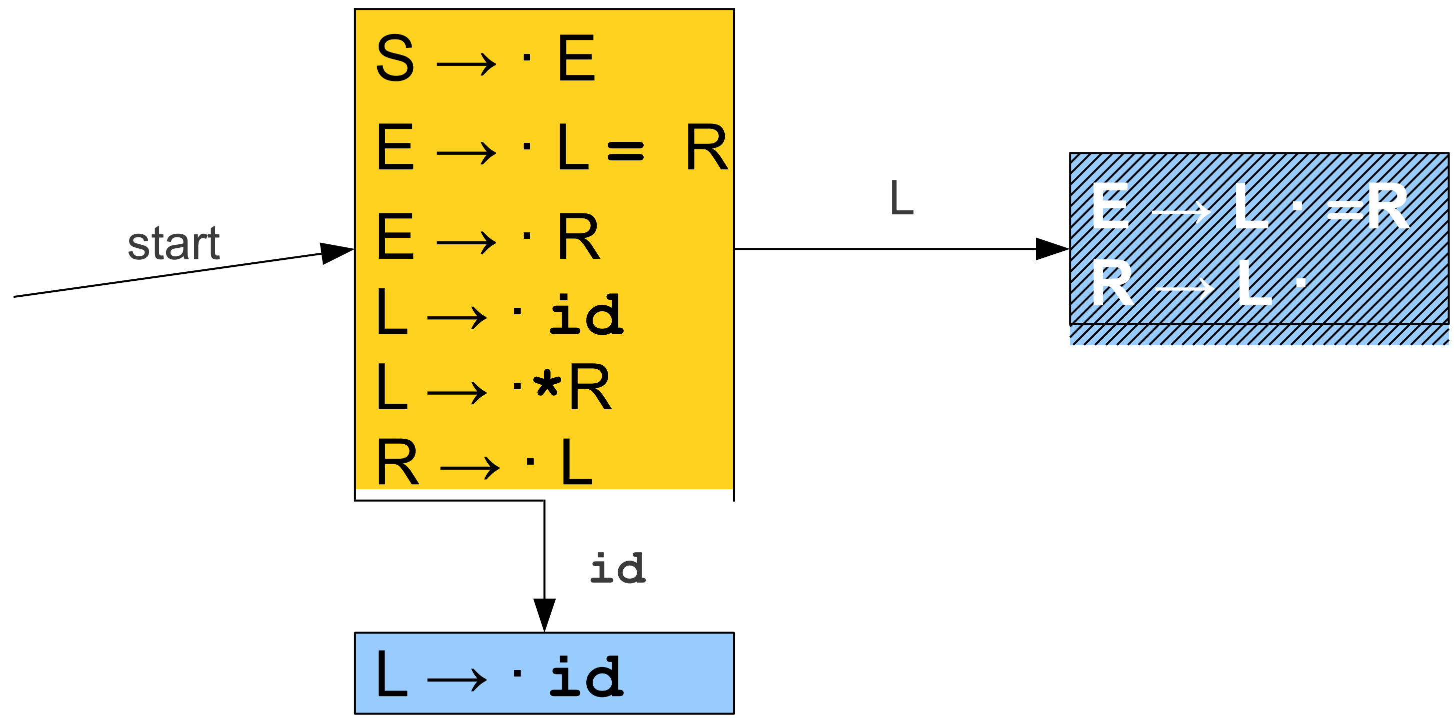
$R \rightarrow L$



id	=	*	id

A Lack of Context

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



id	ǀ	=	*	id

A Lack of Context

$S \rightarrow E$

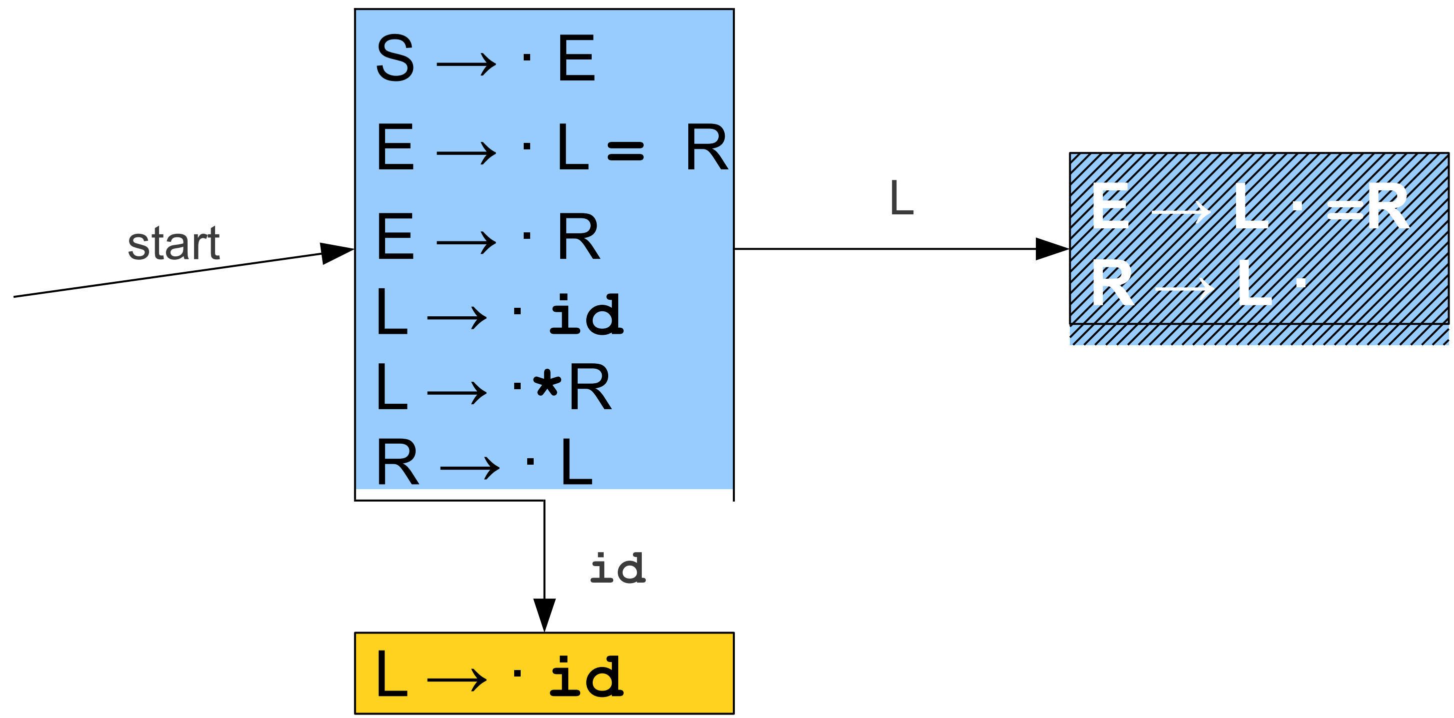
$E \rightarrow L = R$

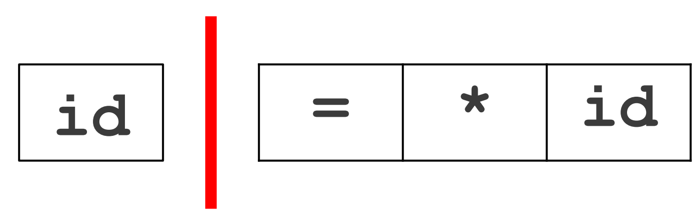
$E \rightarrow R$

$L \rightarrow id$

$L \rightarrow *R$

$R \rightarrow L$





A Lack of Context

S → E

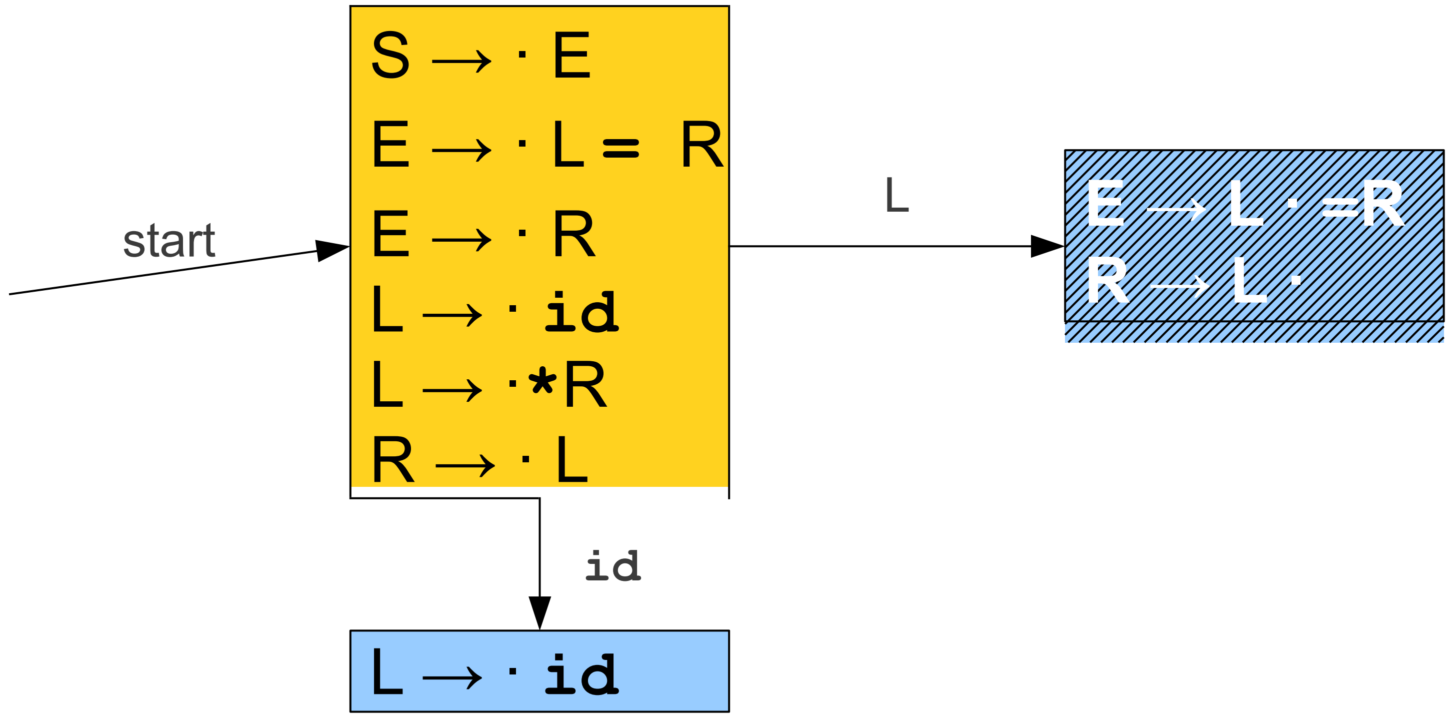
E → L = R

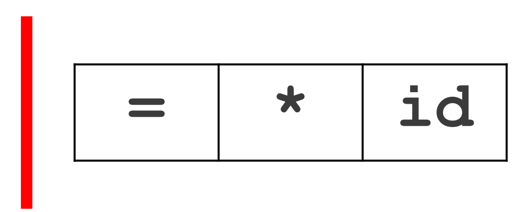
E → R

L → id

L → *R

R → L





A Lack of Context

$S \rightarrow E$

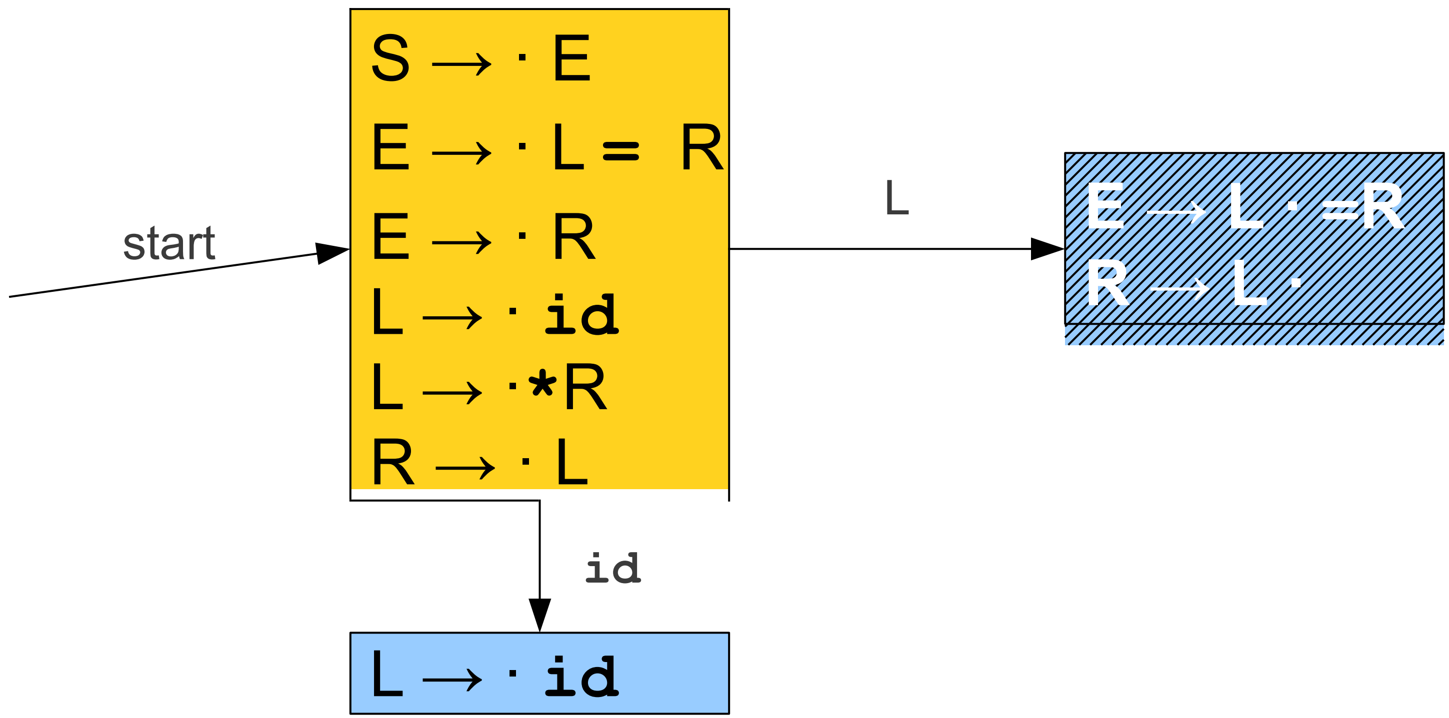
$E \rightarrow L = R$

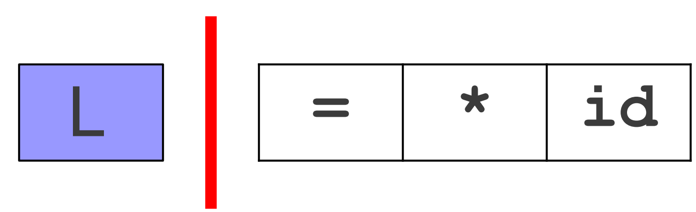
$E \rightarrow R$

$L \rightarrow id$

$L \rightarrow *R$

$R \rightarrow L$





A Lack of Context

$S \rightarrow E$

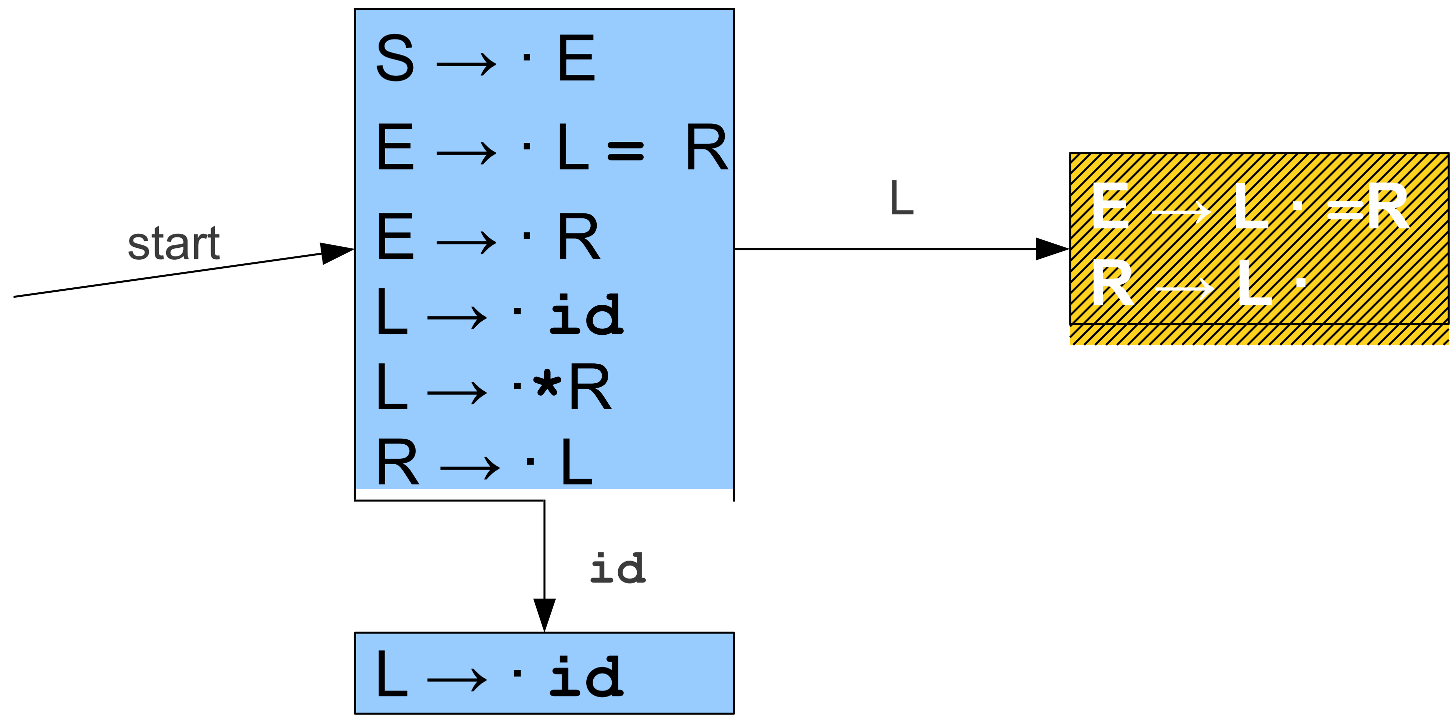
$E \rightarrow L = R$

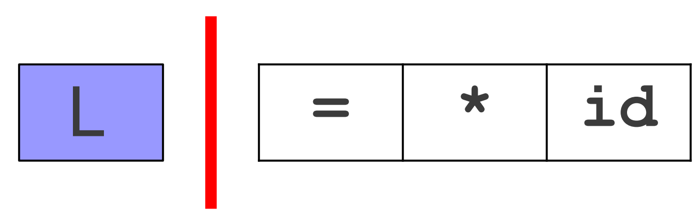
$E \rightarrow R$

$L \rightarrow$ `id`

$L \rightarrow$ `*`R

$R \rightarrow L$





A Lack of Context

$S \rightarrow E$

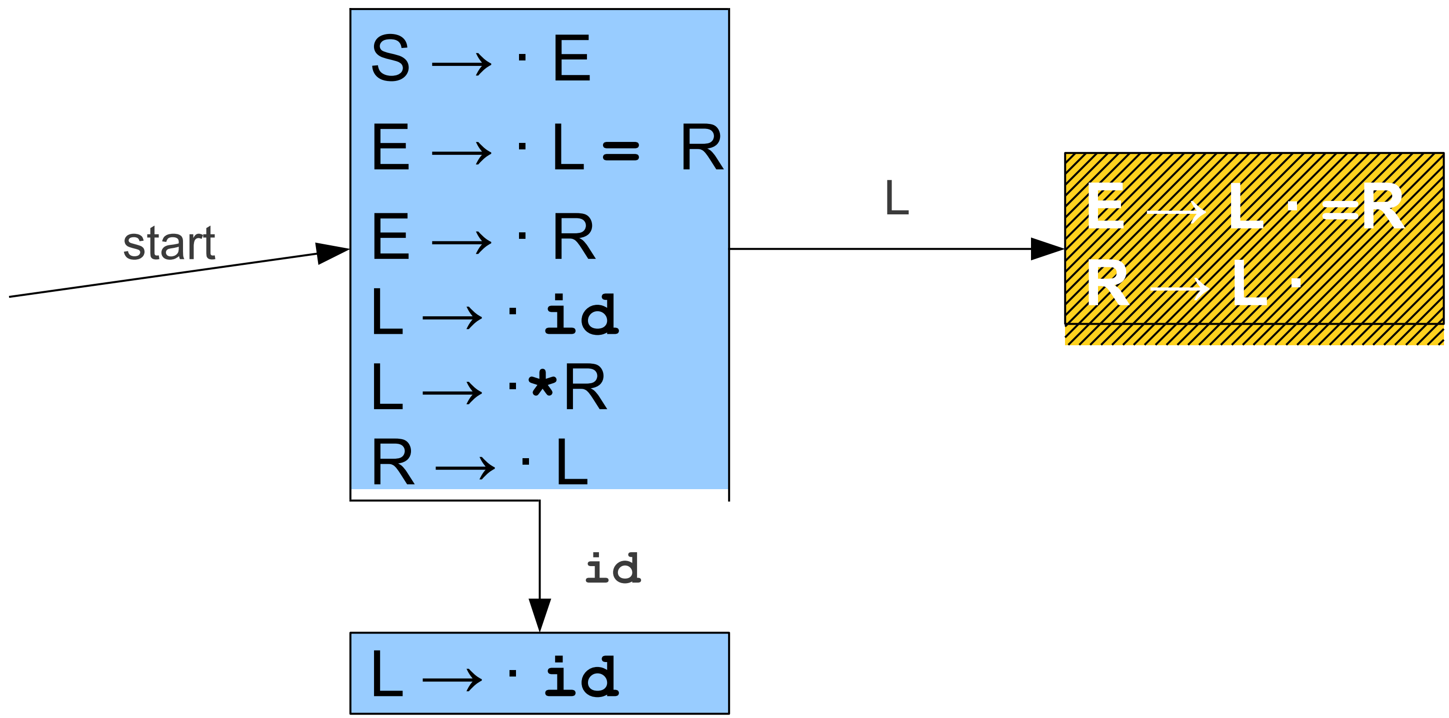
$E \rightarrow L = R$

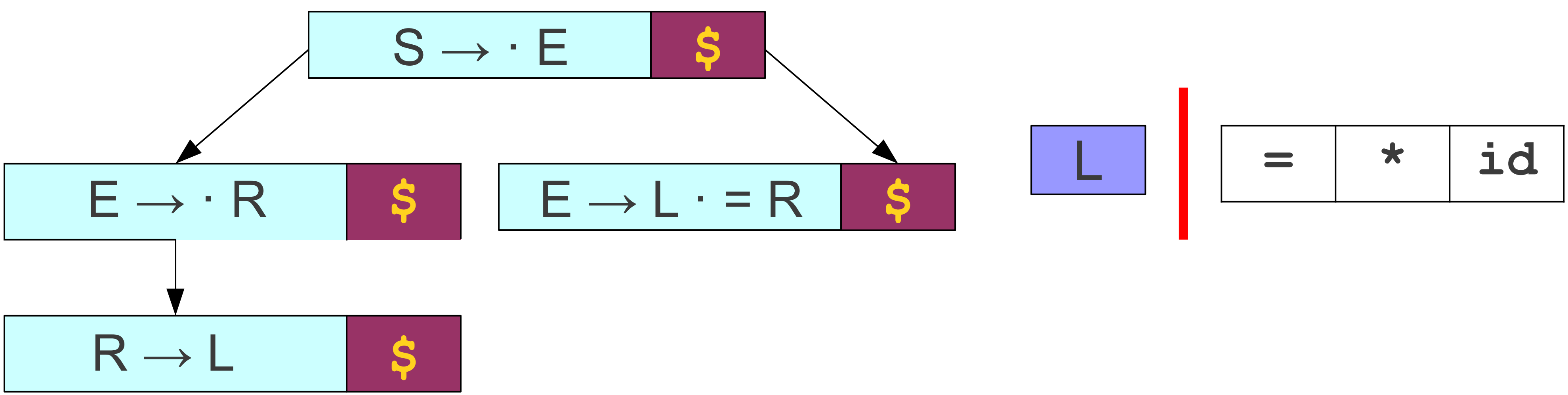
$E \rightarrow R$

$L \rightarrow \texttt{id}$

$L \rightarrow *R$

$R \rightarrow L$





A Lack of Context

$S \rightarrow E$

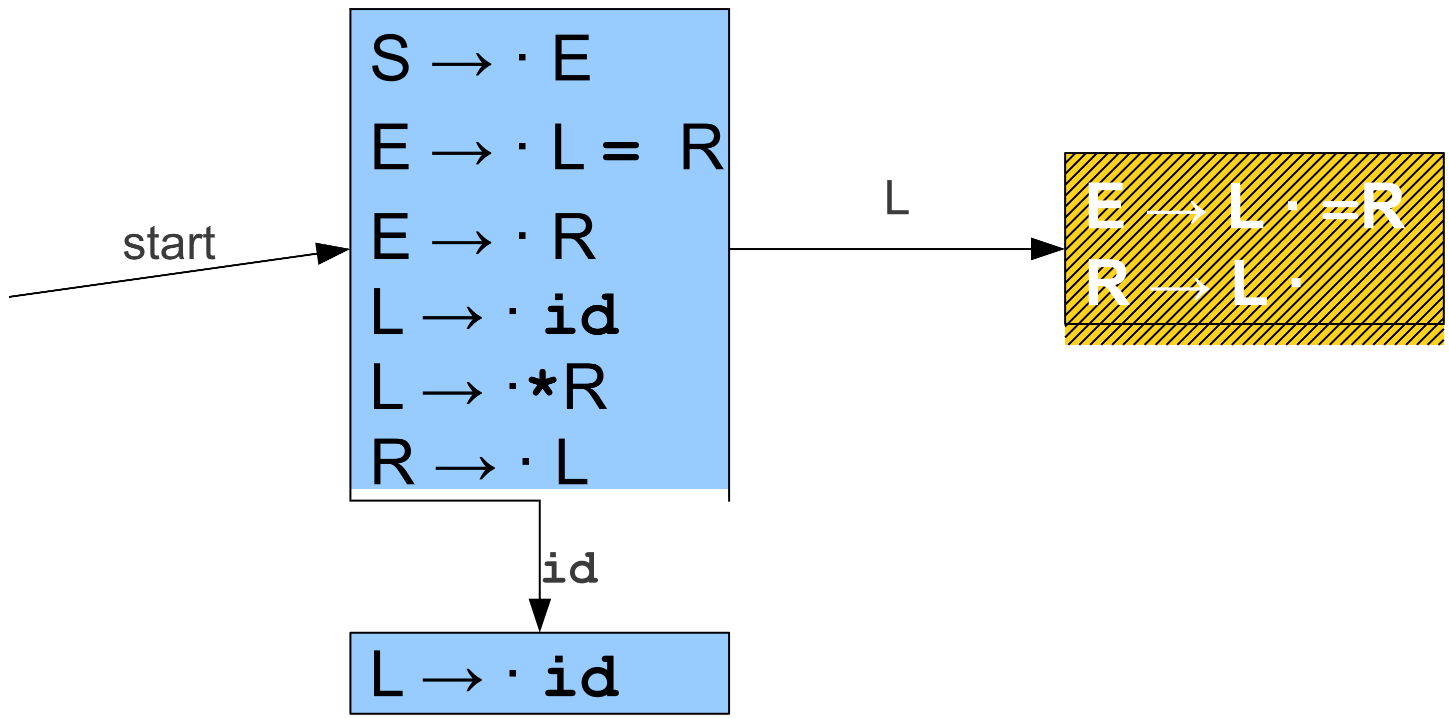
$E \rightarrow L = R$

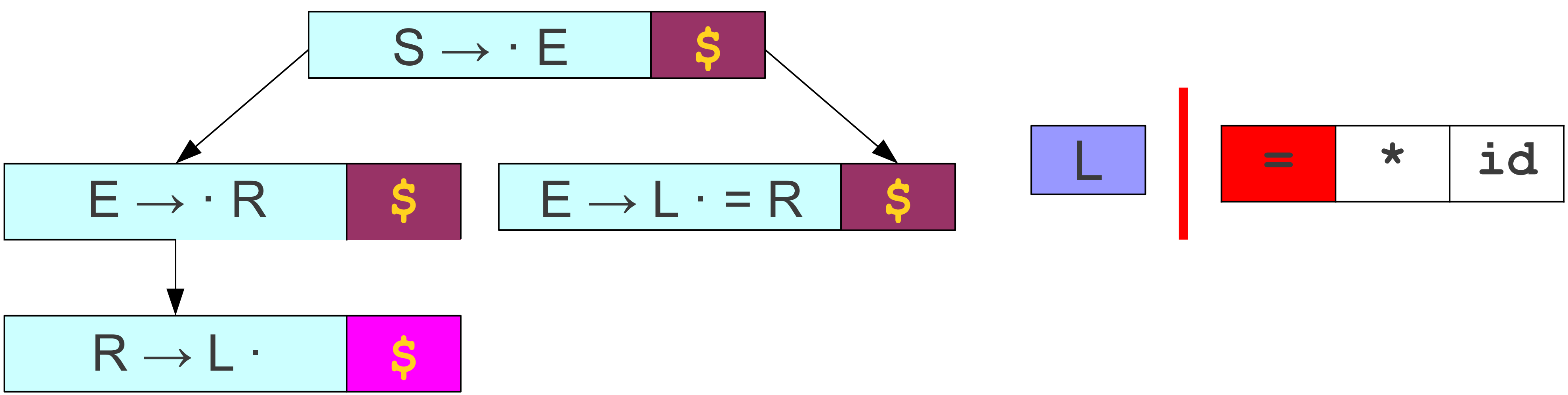
$E \rightarrow R$

$L \rightarrow \texttt{id}$

$L \rightarrow {*}R$

$R \rightarrow L$





Why is SLR(1) Weak?

- 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.
- With LR(1), we have contextual information.

Constructing LR(1) Items

- Begin in a state containing **S** $\rightarrow \cdot$ **E** , **\$** where **S** is the start symbol and **\$** `is lookahead`
- Compute the **closure** of the state:
  - If **A** $\rightarrow a \cdot$ **B**ω, l is in the state, add **B** $\rightarrow \cdot\, \gamma$, t to the state for each production **B** $\rightarrow \gamma$ and for each terminal `t` $\in$ FIRST*(ω`l`)

Deterministic LR(1) Automata

start

$S \to .E\ \$$

(1) $S \to E$

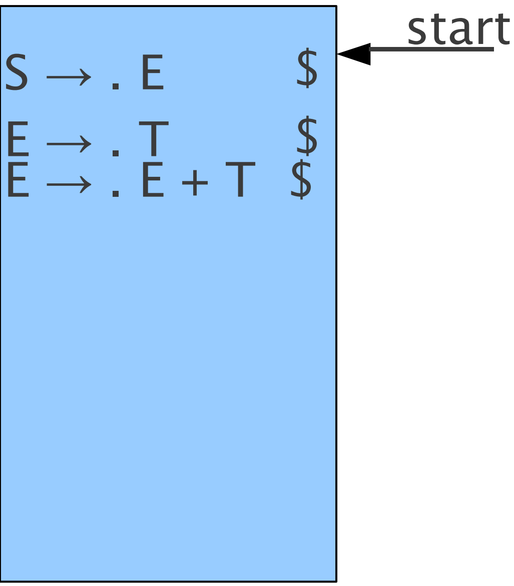
(2) $E \to T$

(3) $E \to E + T$

(4) $T \to \texttt{int}$

(5) $T \to (E)$

Deterministic LR(1) Automata



(1) S → E

(2) E → T

(3) E → E + T

(4) T → int

(5) T → (E)

Deterministic LR(1) Automata

start

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

Deterministic LR(1) Automata

start

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

Deterministic LR(1) Automata

start

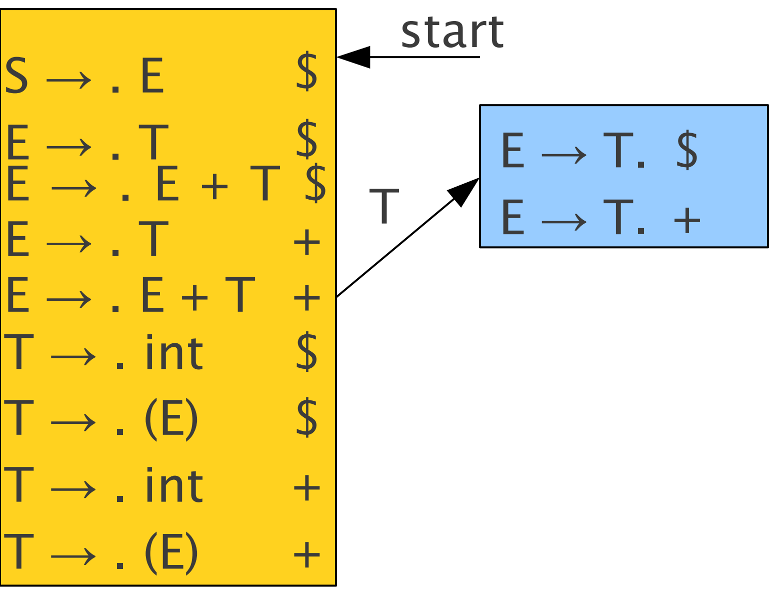
$S \rightarrow .\ E$	$
$E \rightarrow .\ T$	$
$E \rightarrow .\ E + T$	$
$E \rightarrow .\ T$	+
$E \rightarrow .\ E + T$	+
$T \rightarrow .\ int$	$
$T \rightarrow .\ (E)$	$
$T \rightarrow .\ int$	+
$T \rightarrow .\ (E)$	+

Deterministic LR(1) Automata

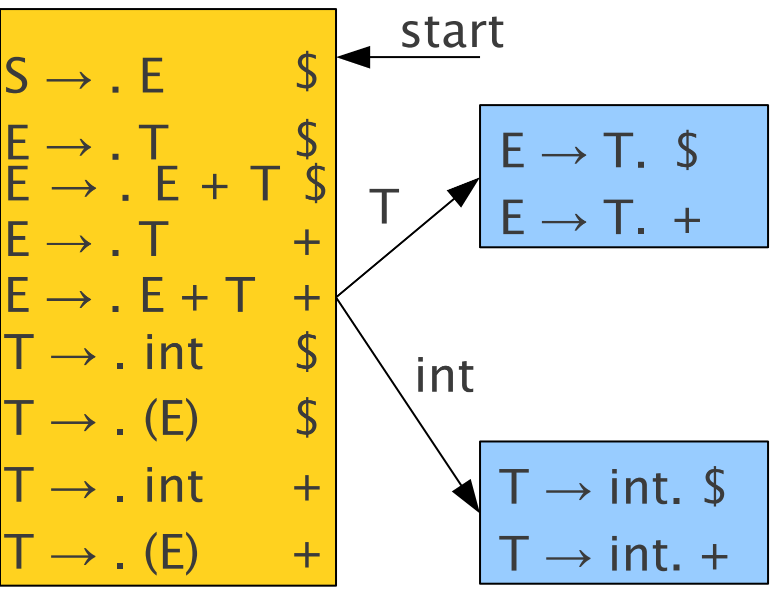
start

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

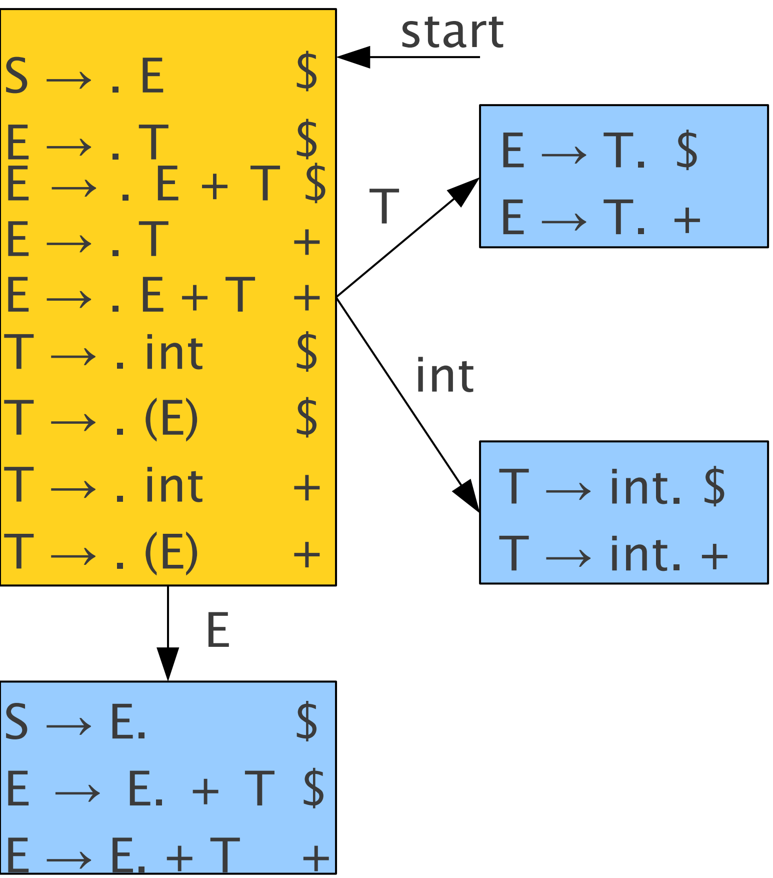
Deterministic LR(1) Automata



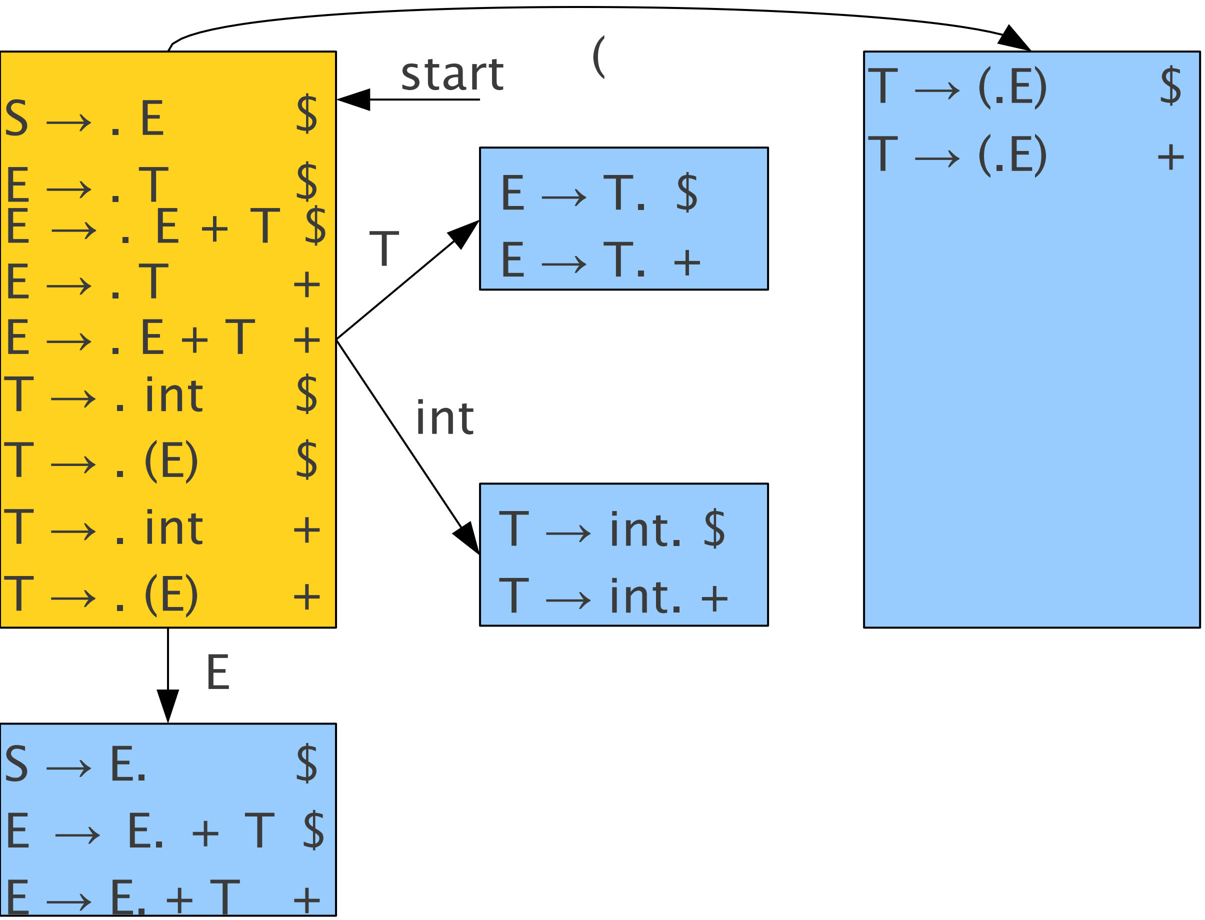
Deterministic LR(1) Automata



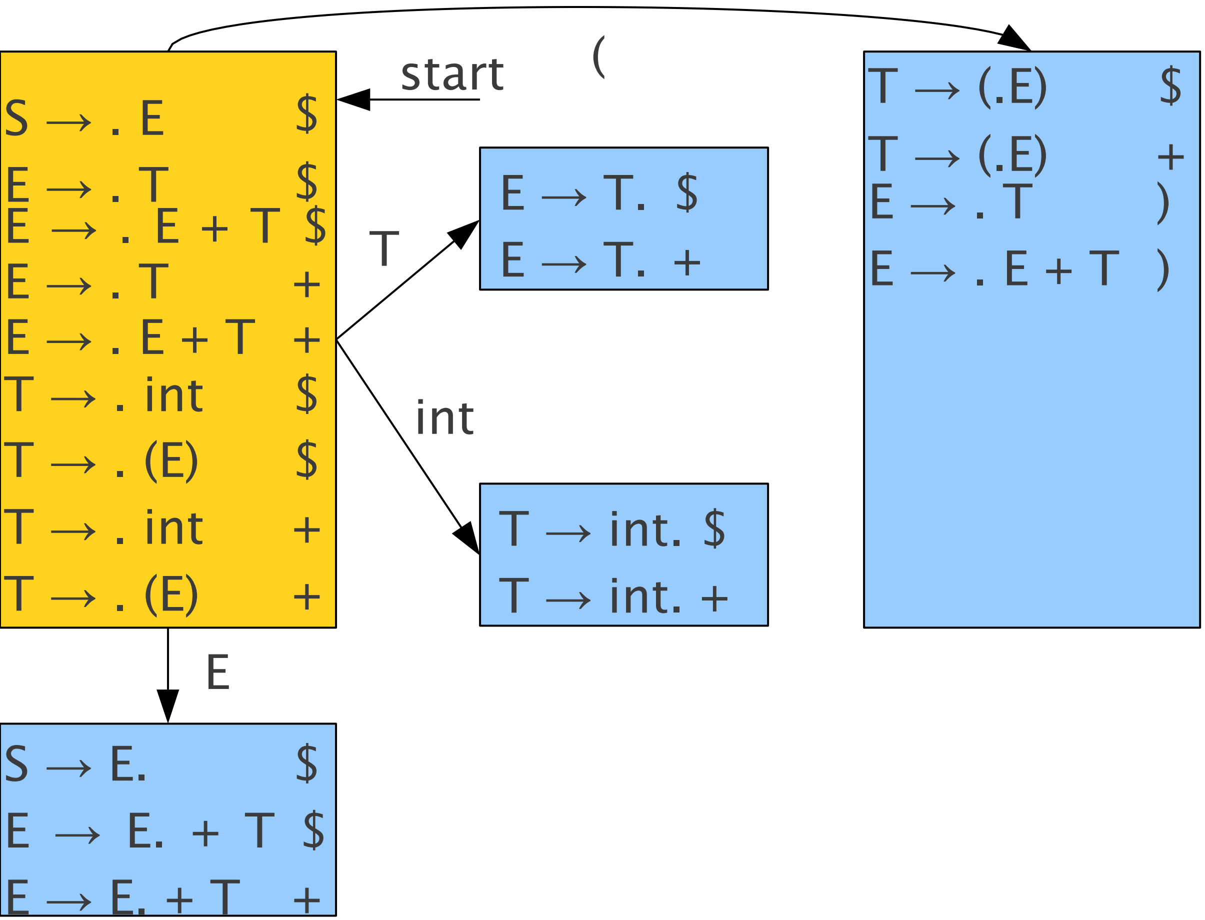
Deterministic LR(1) Automata



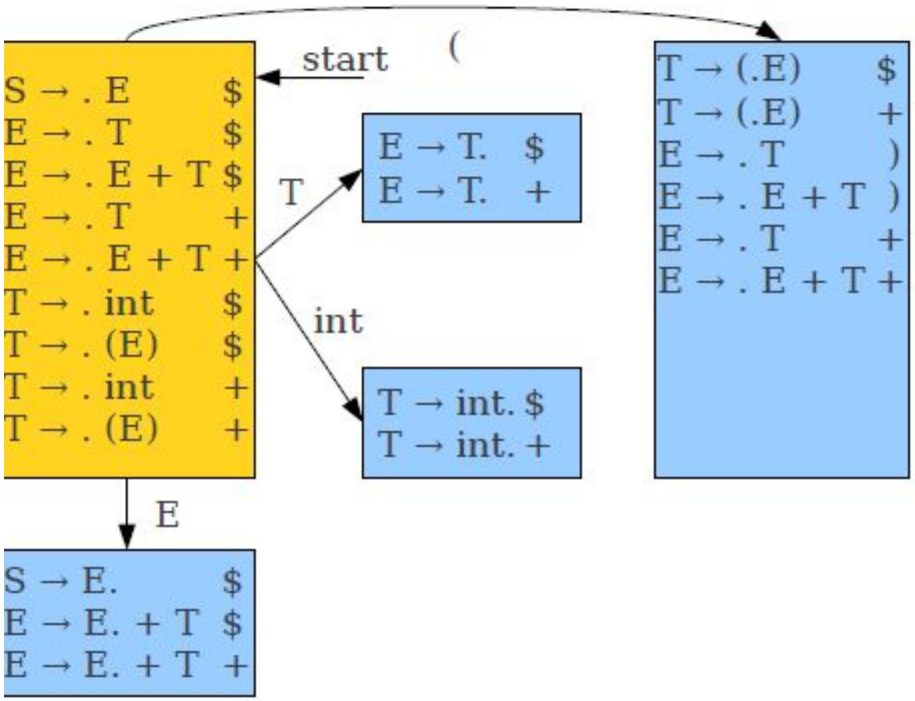
Deterministic LR(1) Automata



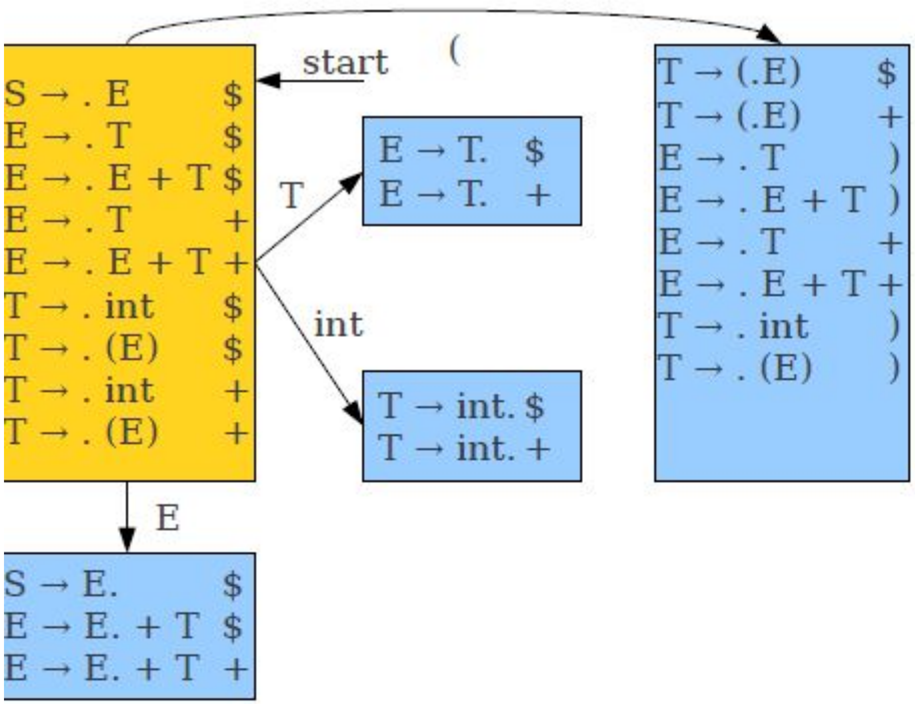
Deterministic LR(1) Automata



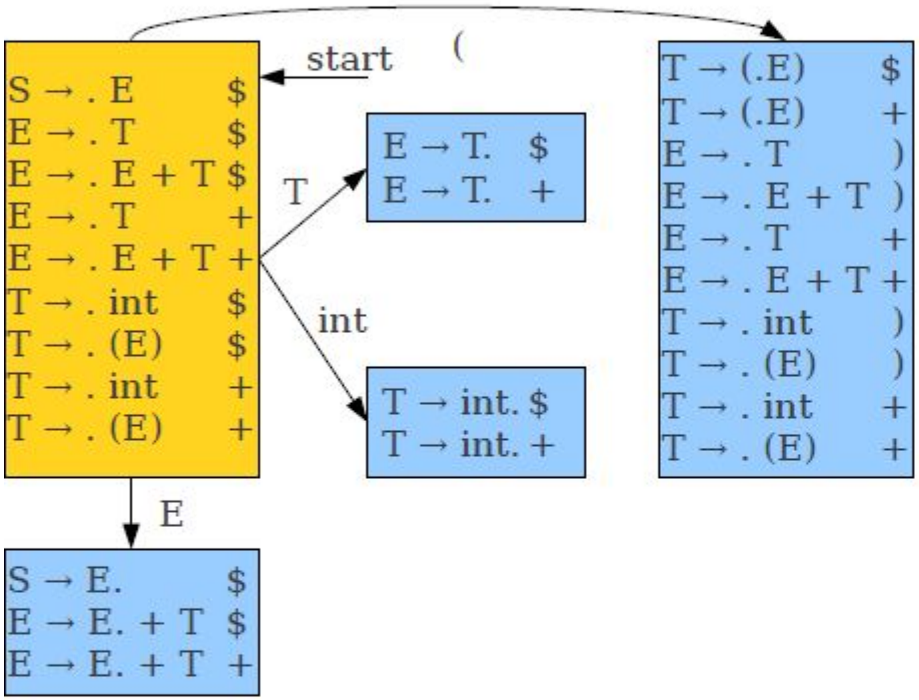
Deterministic LR(1) Automata



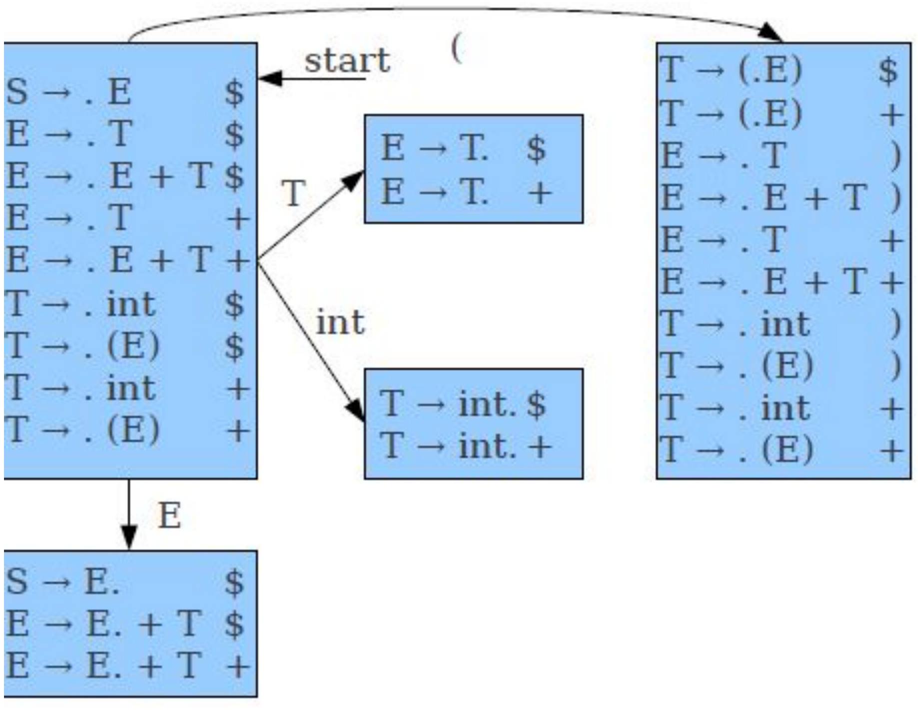
Deterministic LR(1) Automata



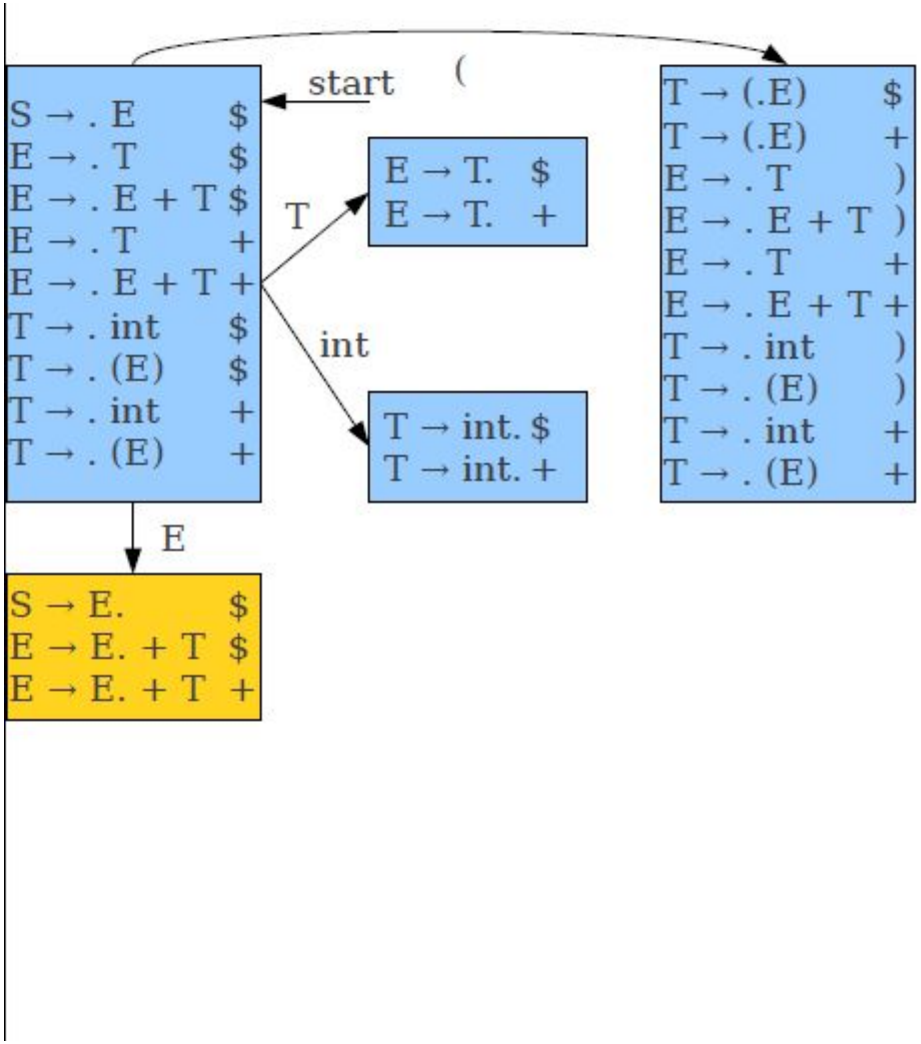
Deterministic LR(1) Automata



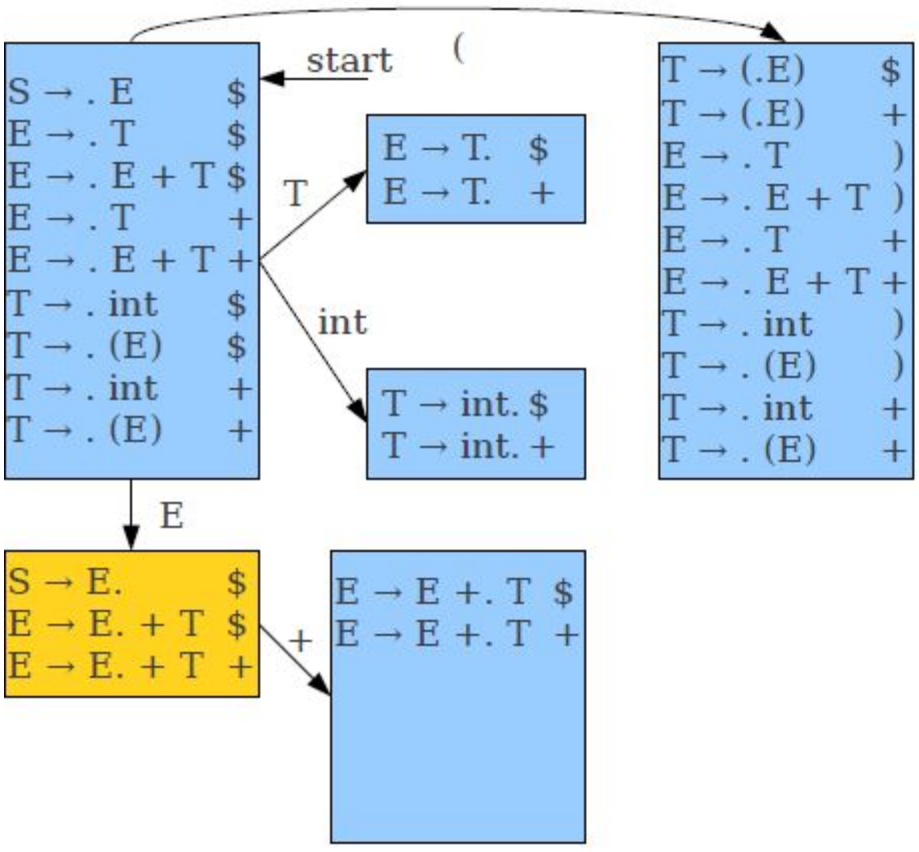
Deterministic LR(1) Automata



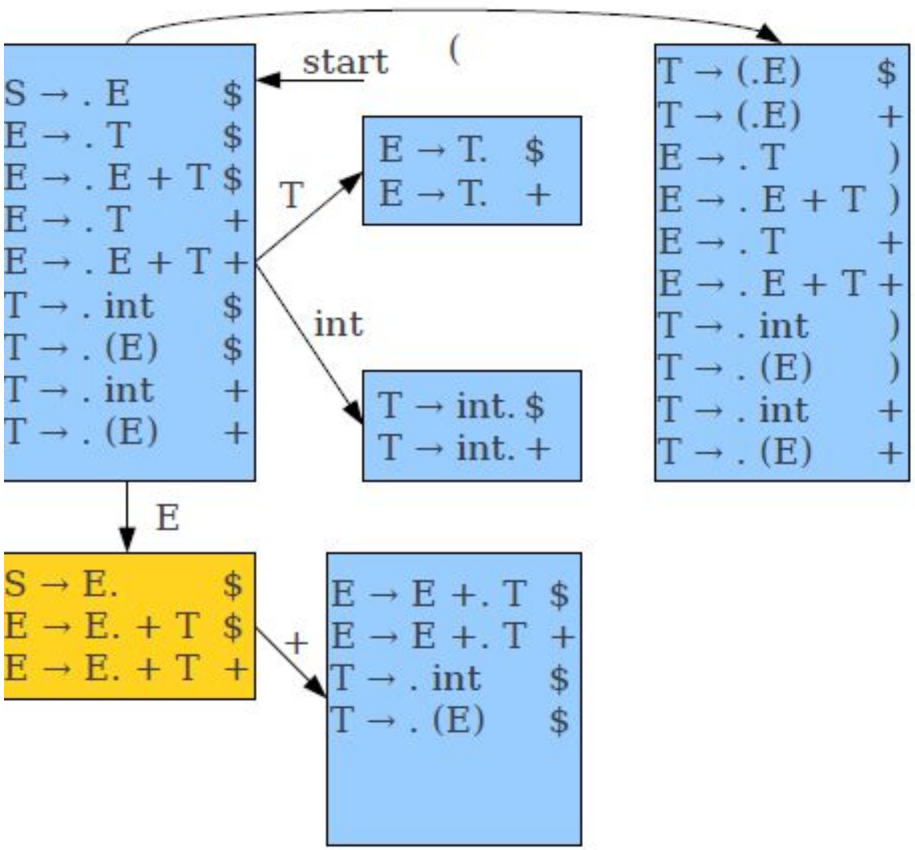
Deterministic LR(1) Automata



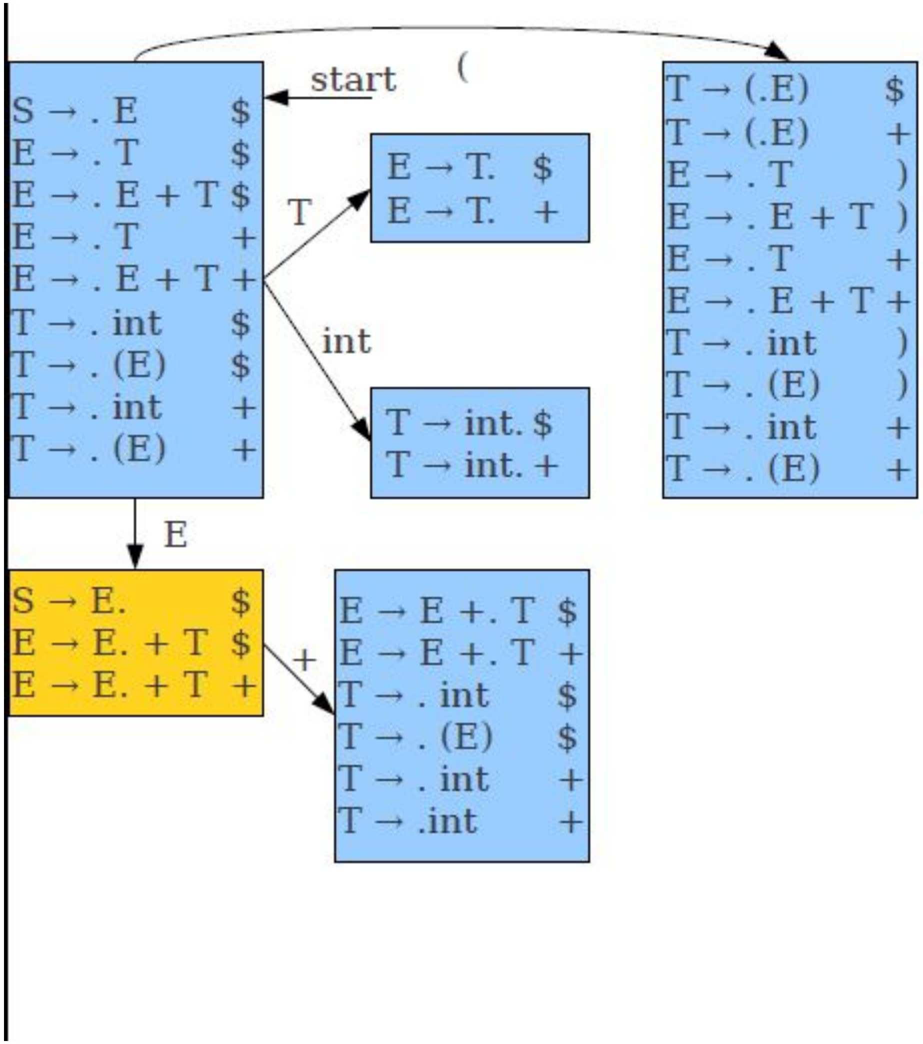
Deterministic LR(1) Automata



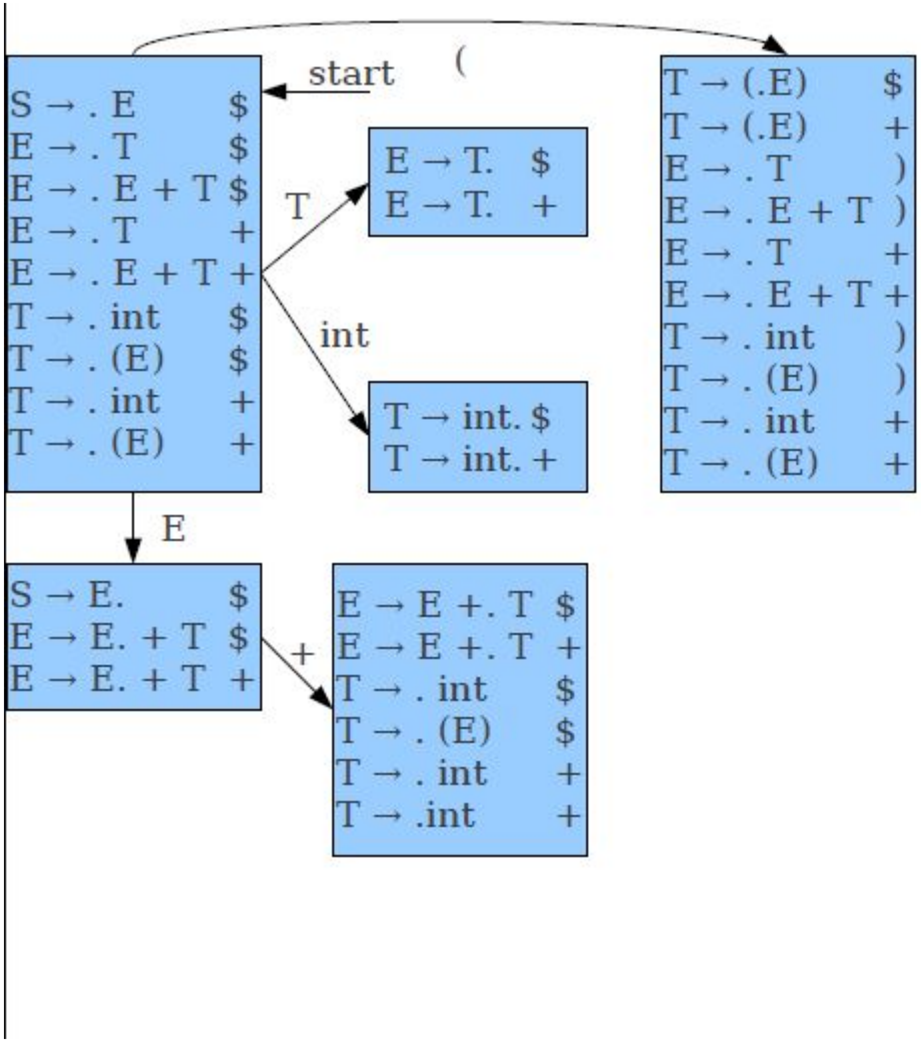
Deterministic LR(1) Automata



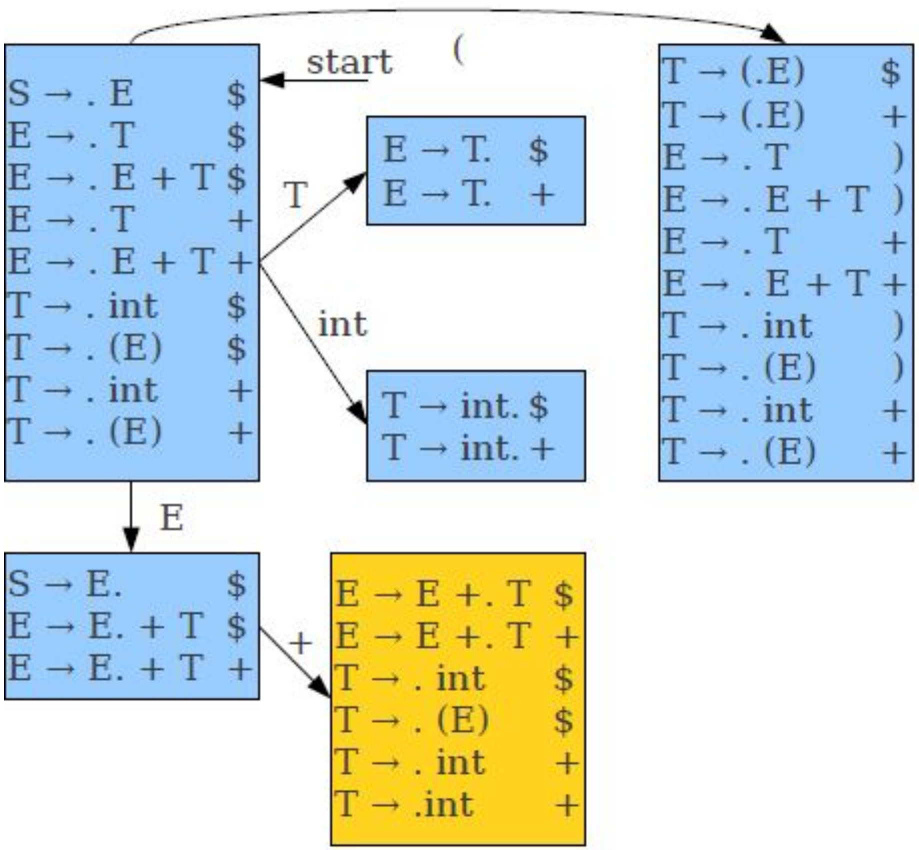
Deterministic LR(1) Automata



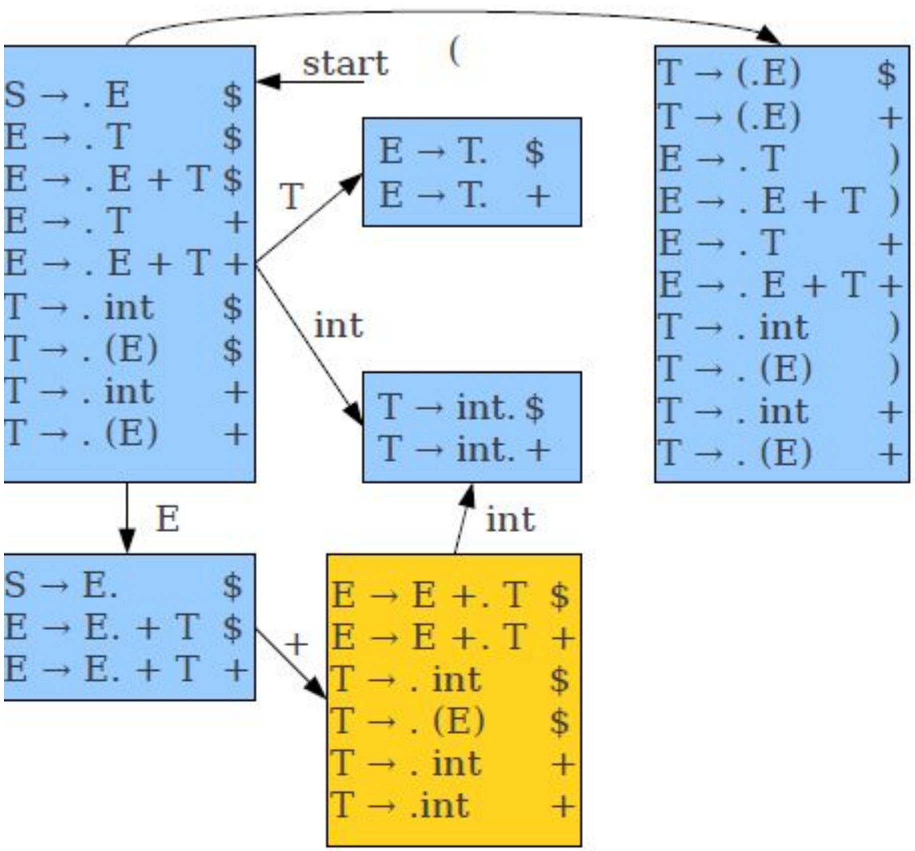
Deterministic LR(1) Automata



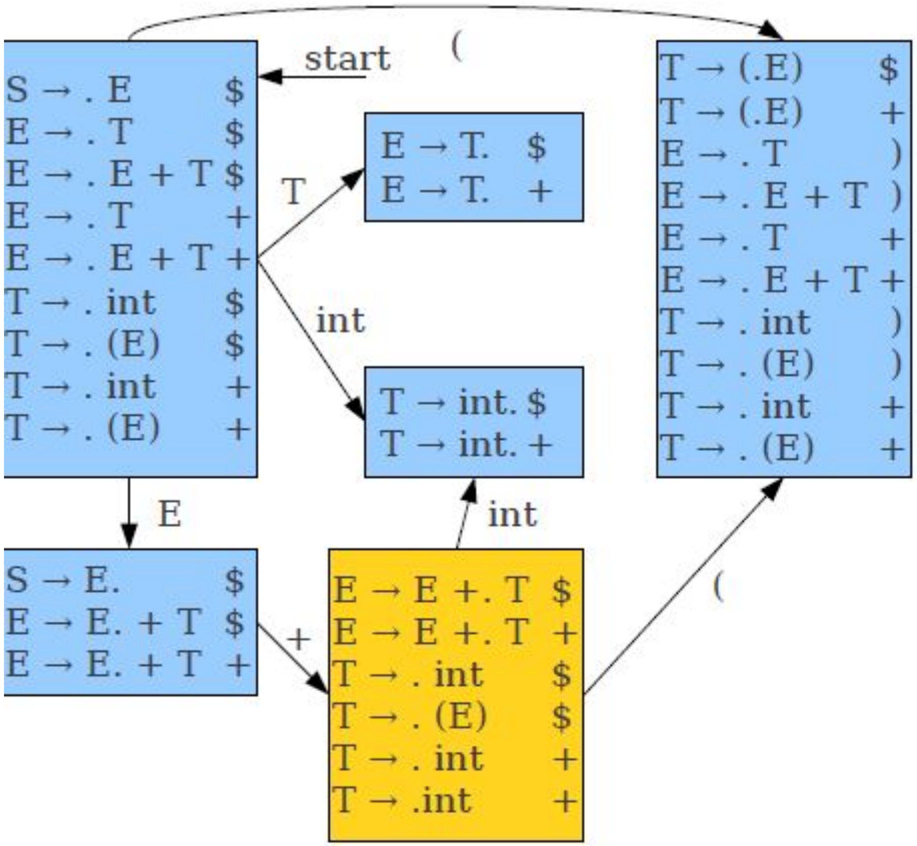
Deterministic LR(1) Automata



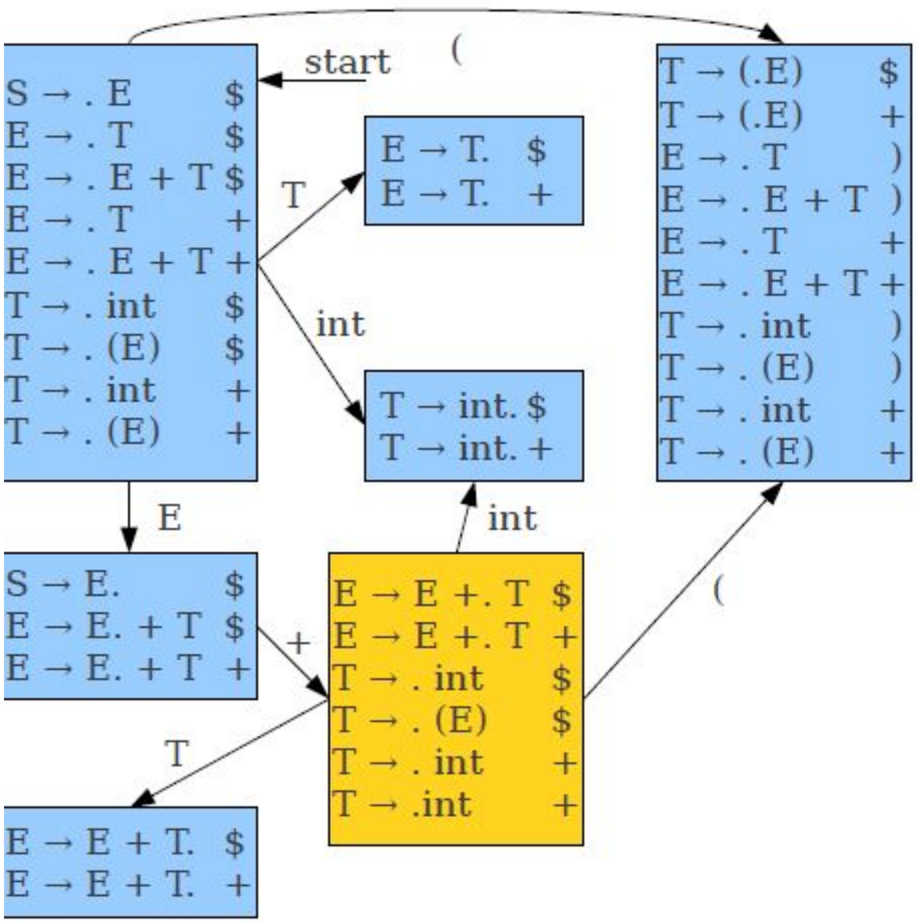
Deterministic LR(1) Automata



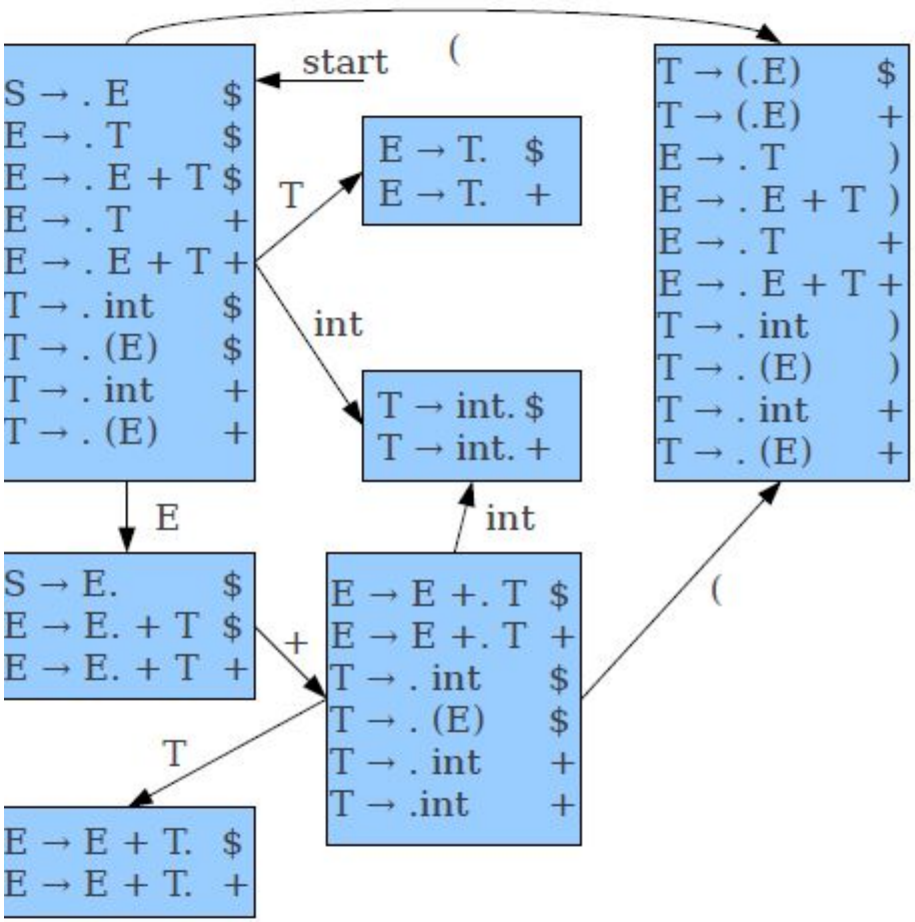
Deterministic LR(1) Automata



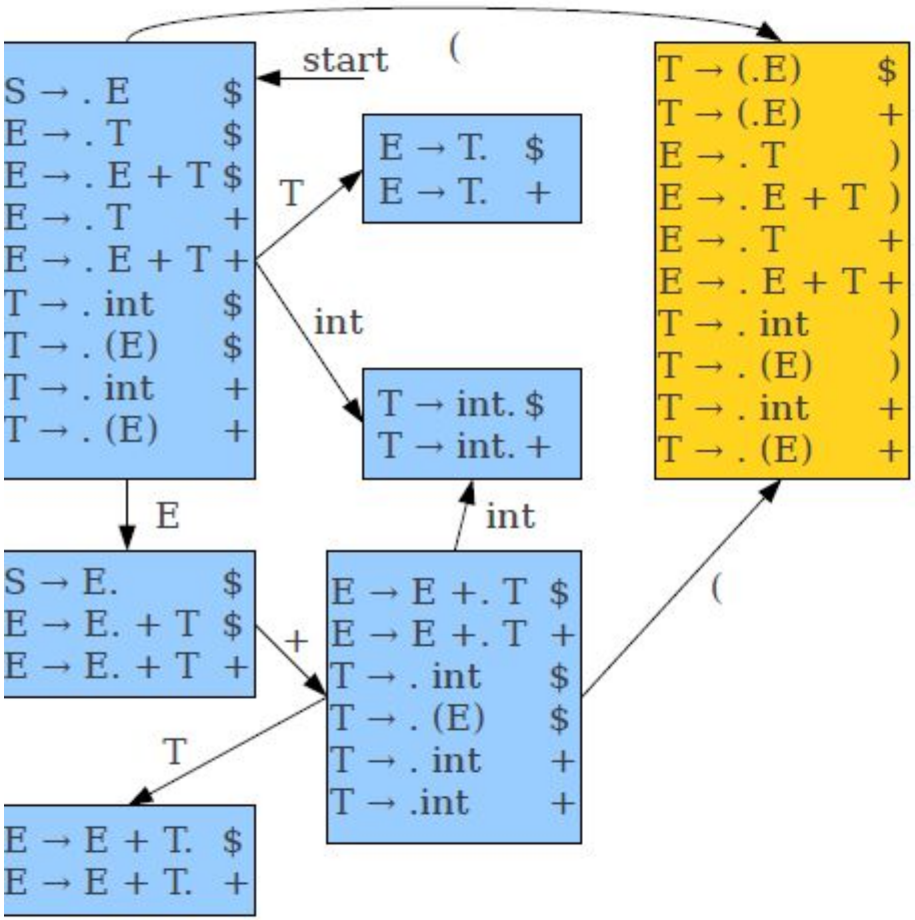
Deterministic LR(1) Automata



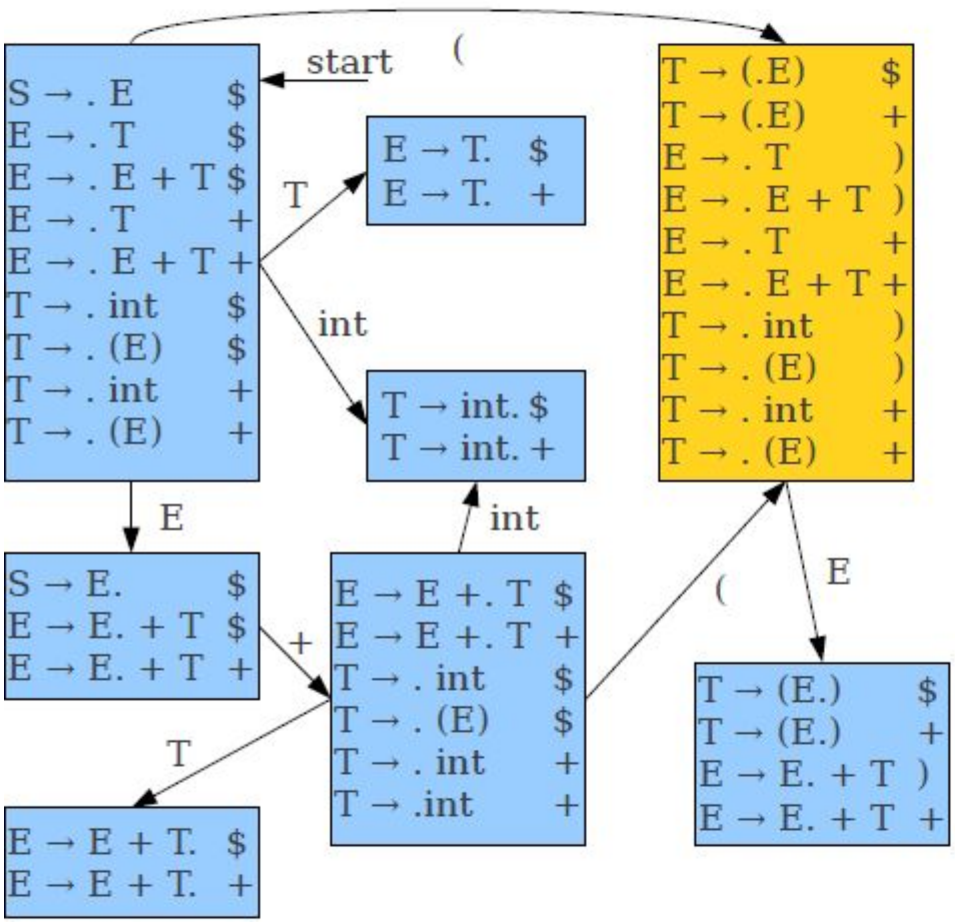
Deterministic LR(1) Automata



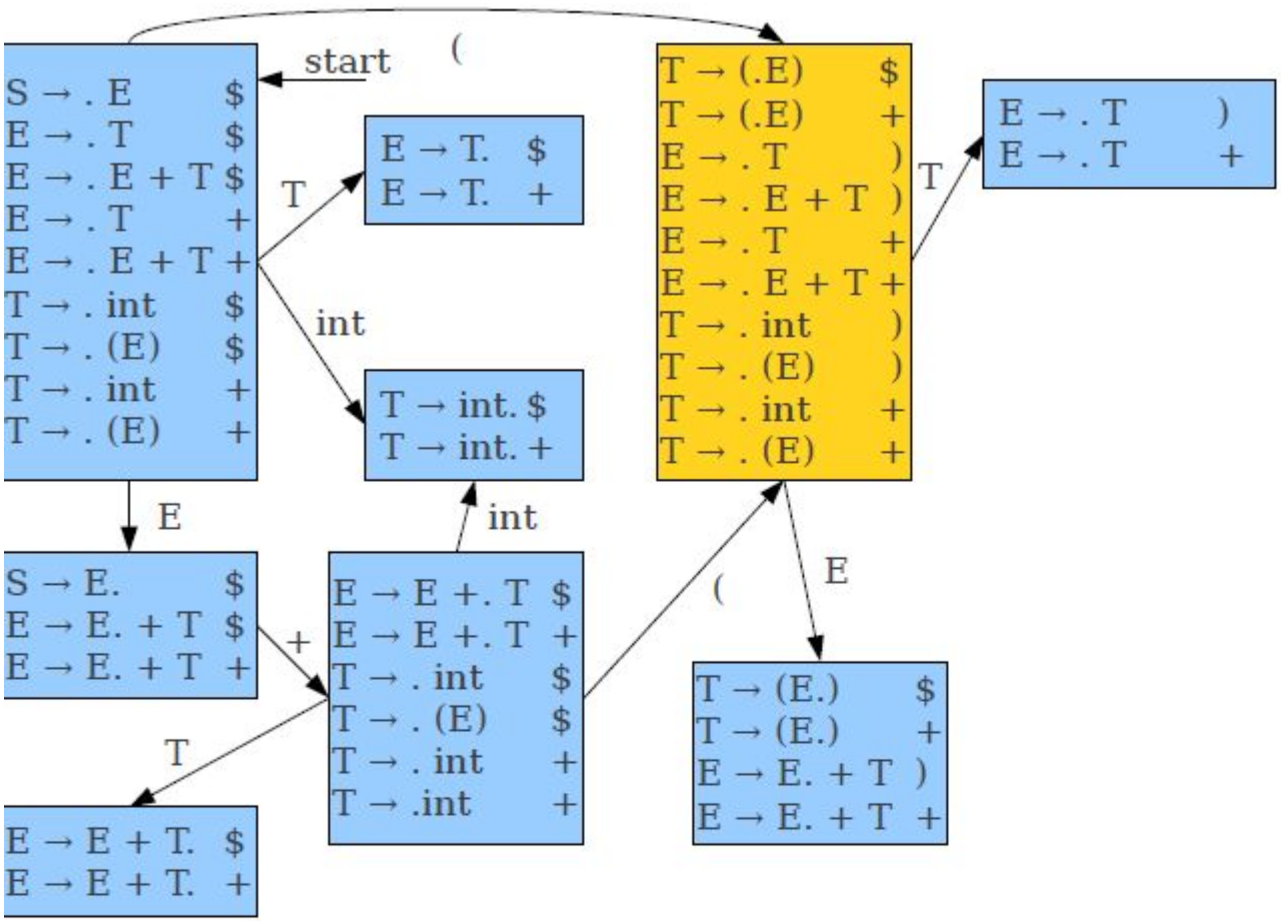
Deterministic LR(1) Automata



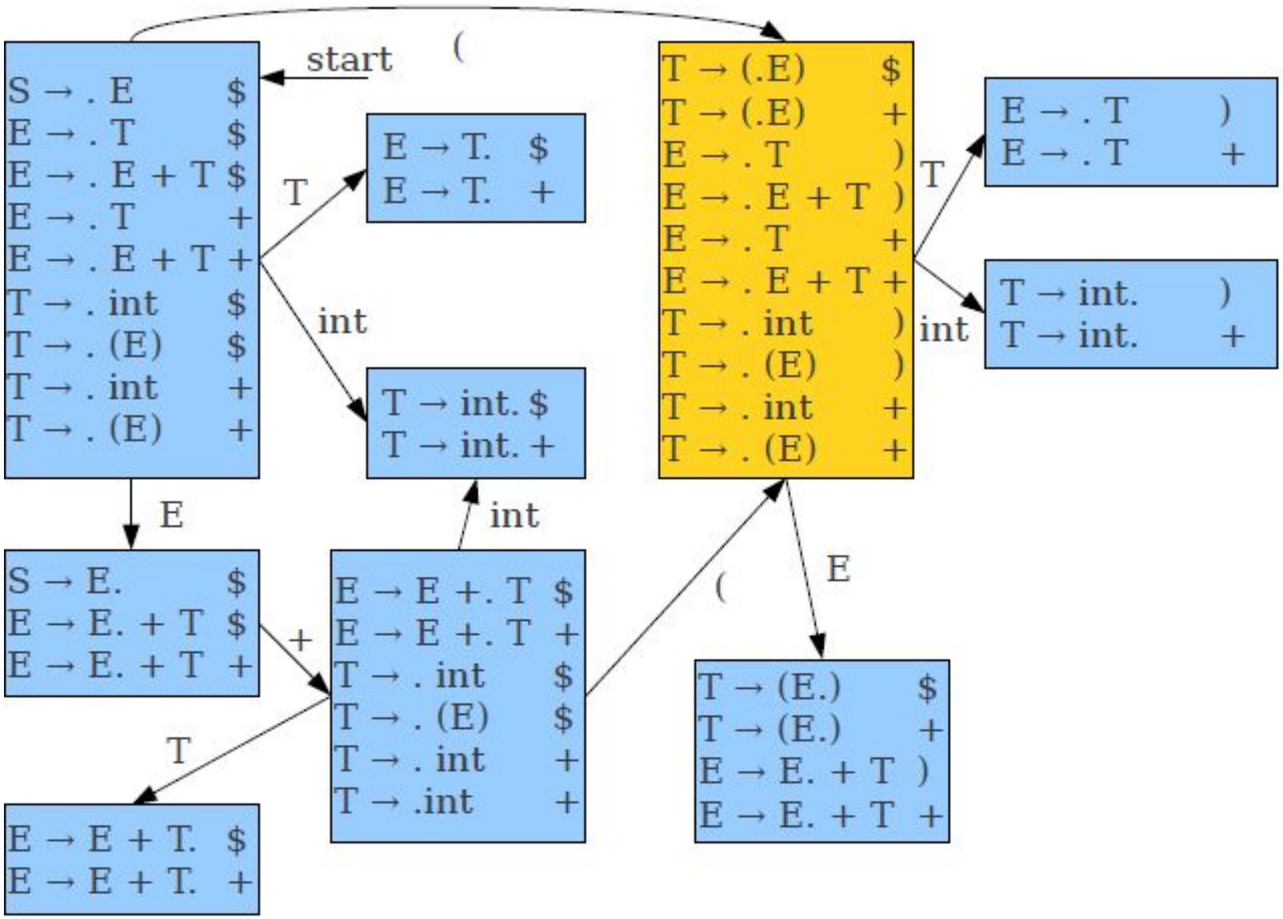
Deterministic LR(1) Automata



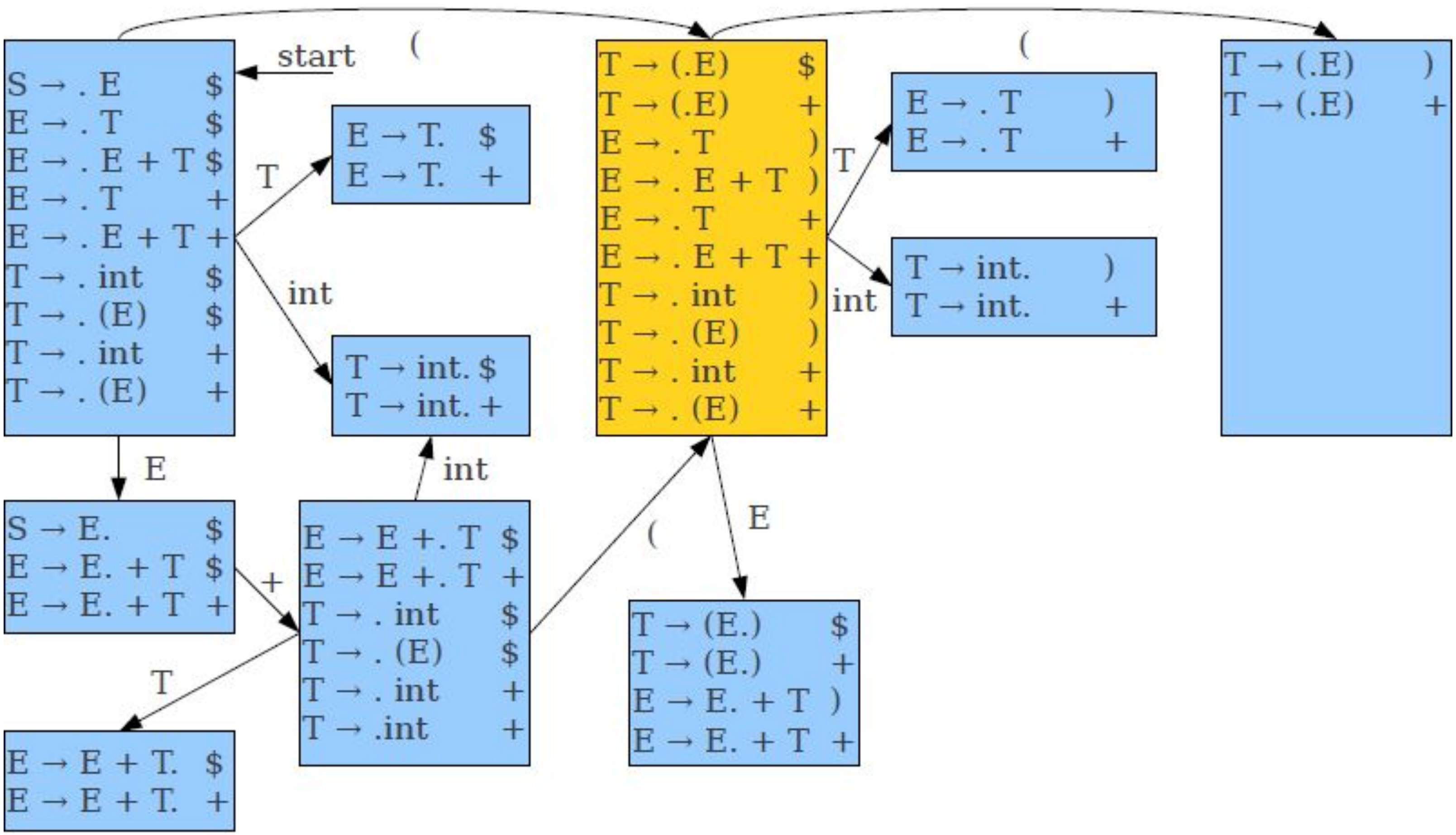
Deterministic LR(1) Automata



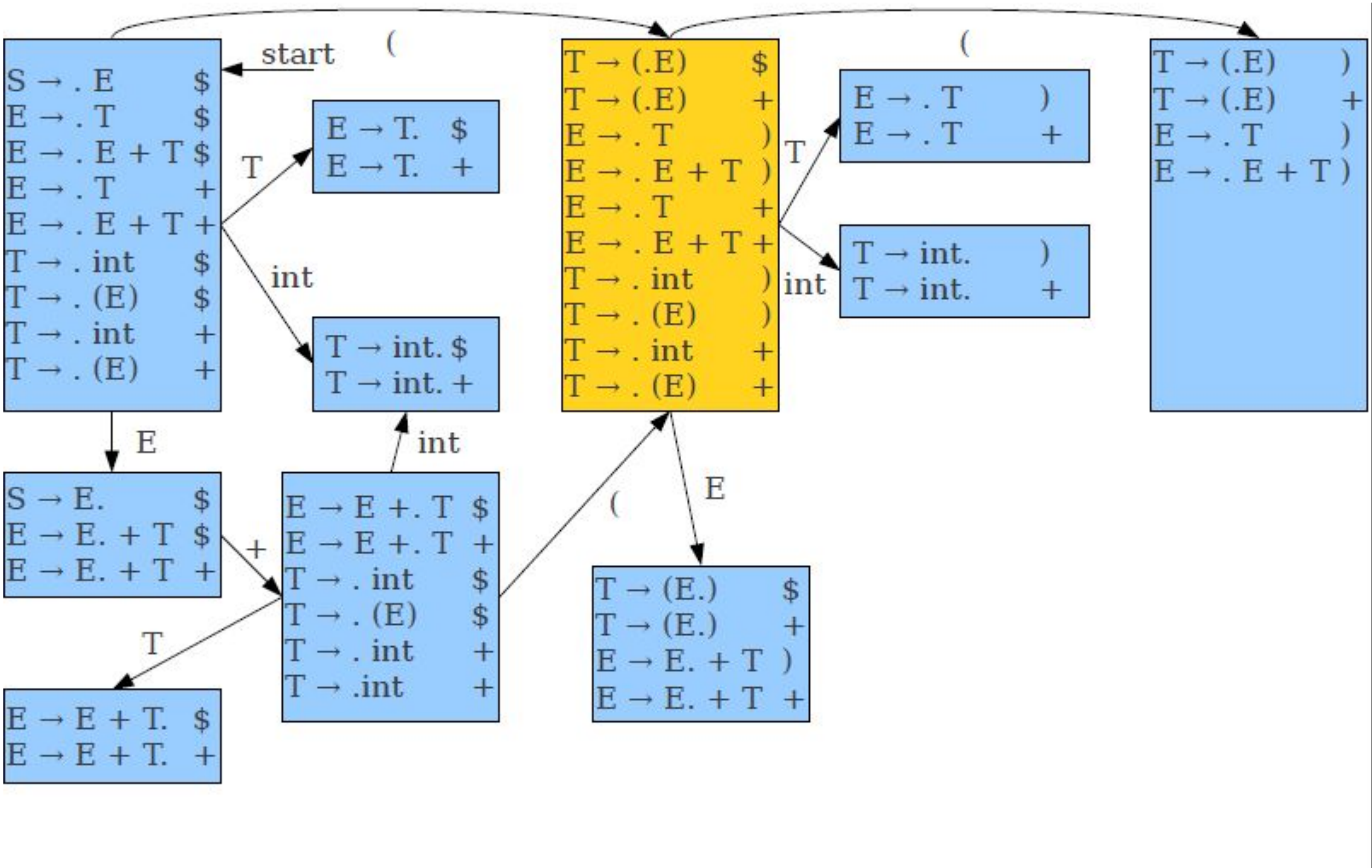
Deterministic LR(1) Automata



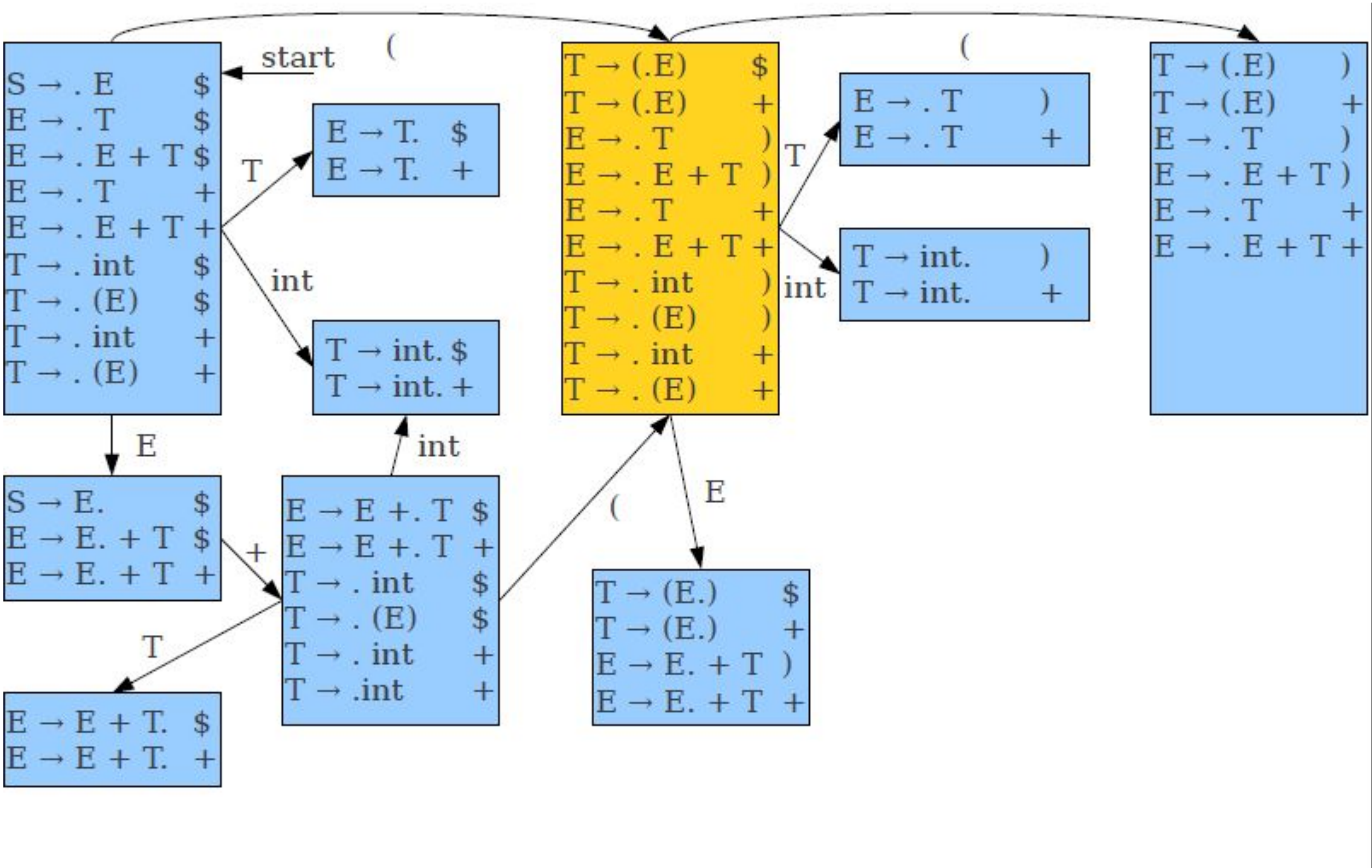
Deterministic LR(1) Automata



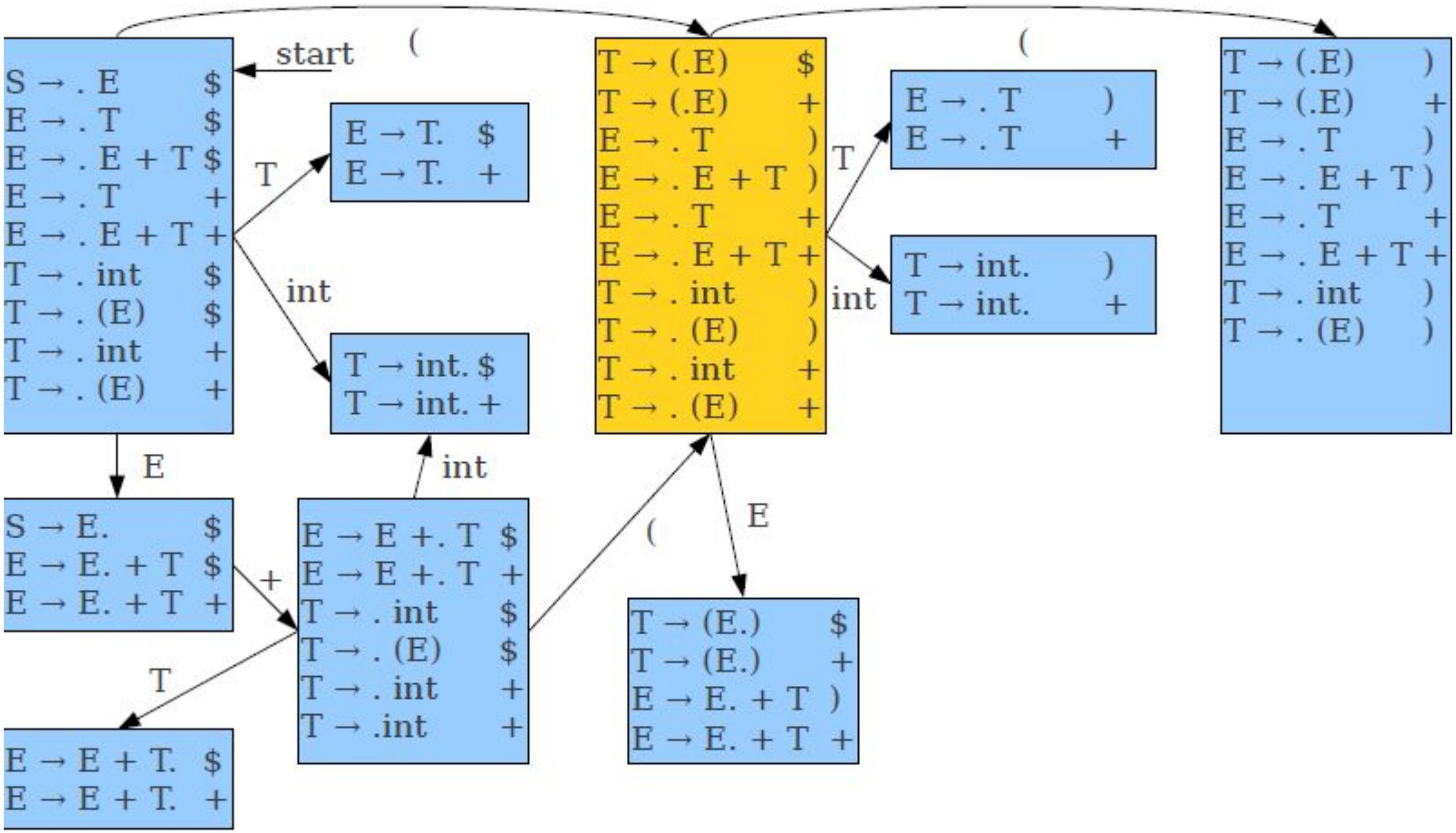
Deterministic LR(1) Automata



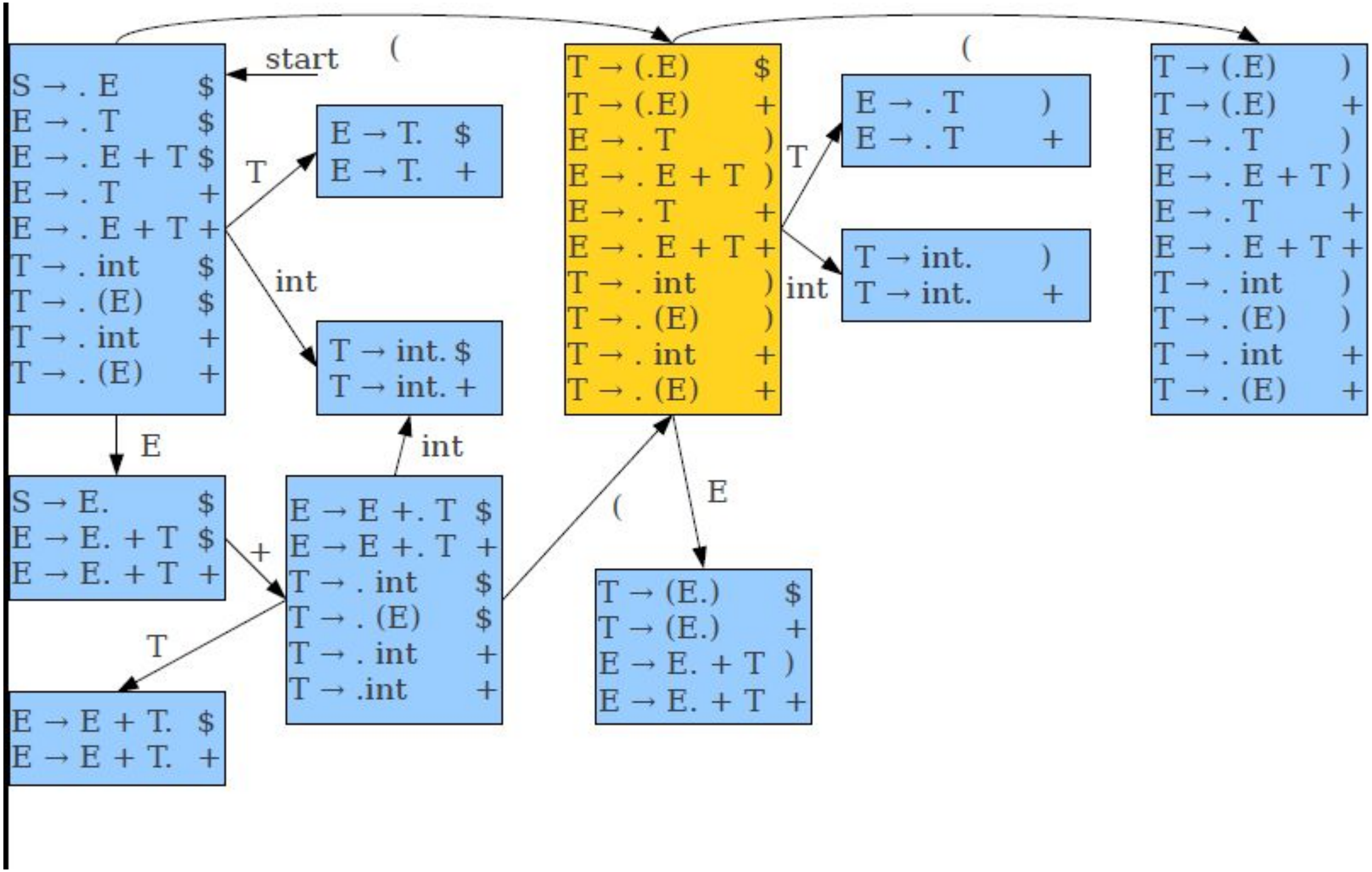
Deterministic LR(1) Automata



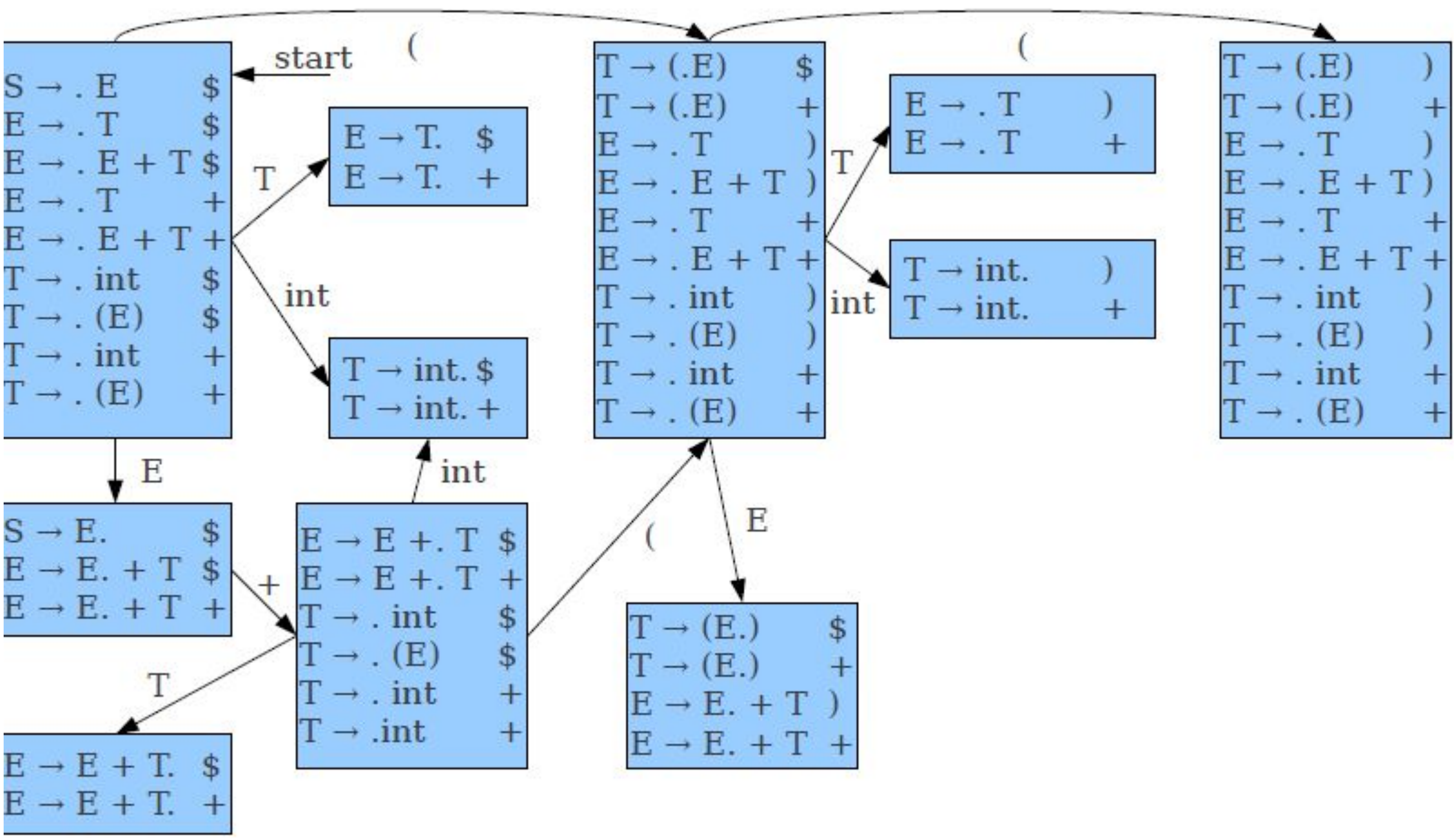
Deterministic LR(1) Automata



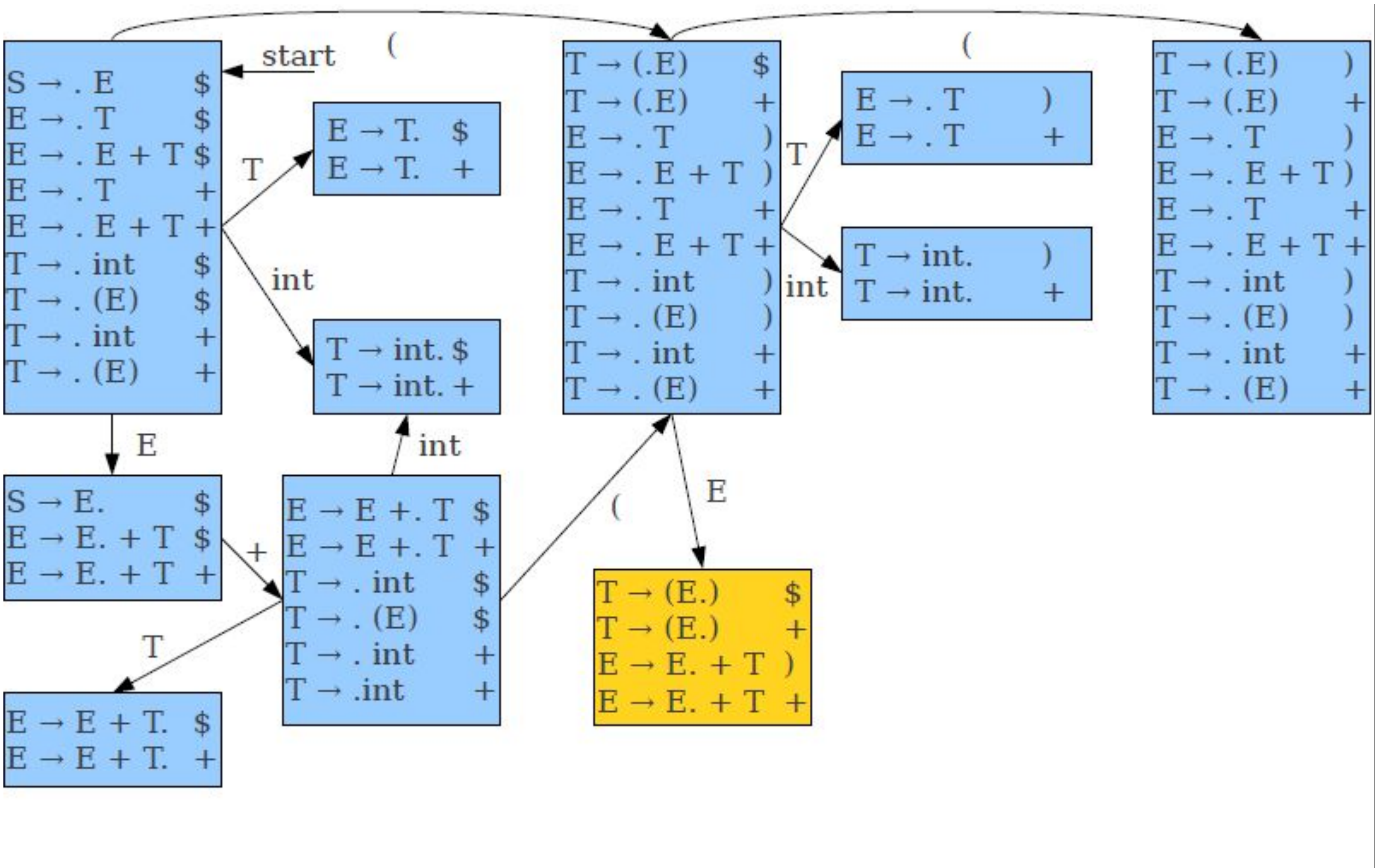
Deterministic LR(1) Automata



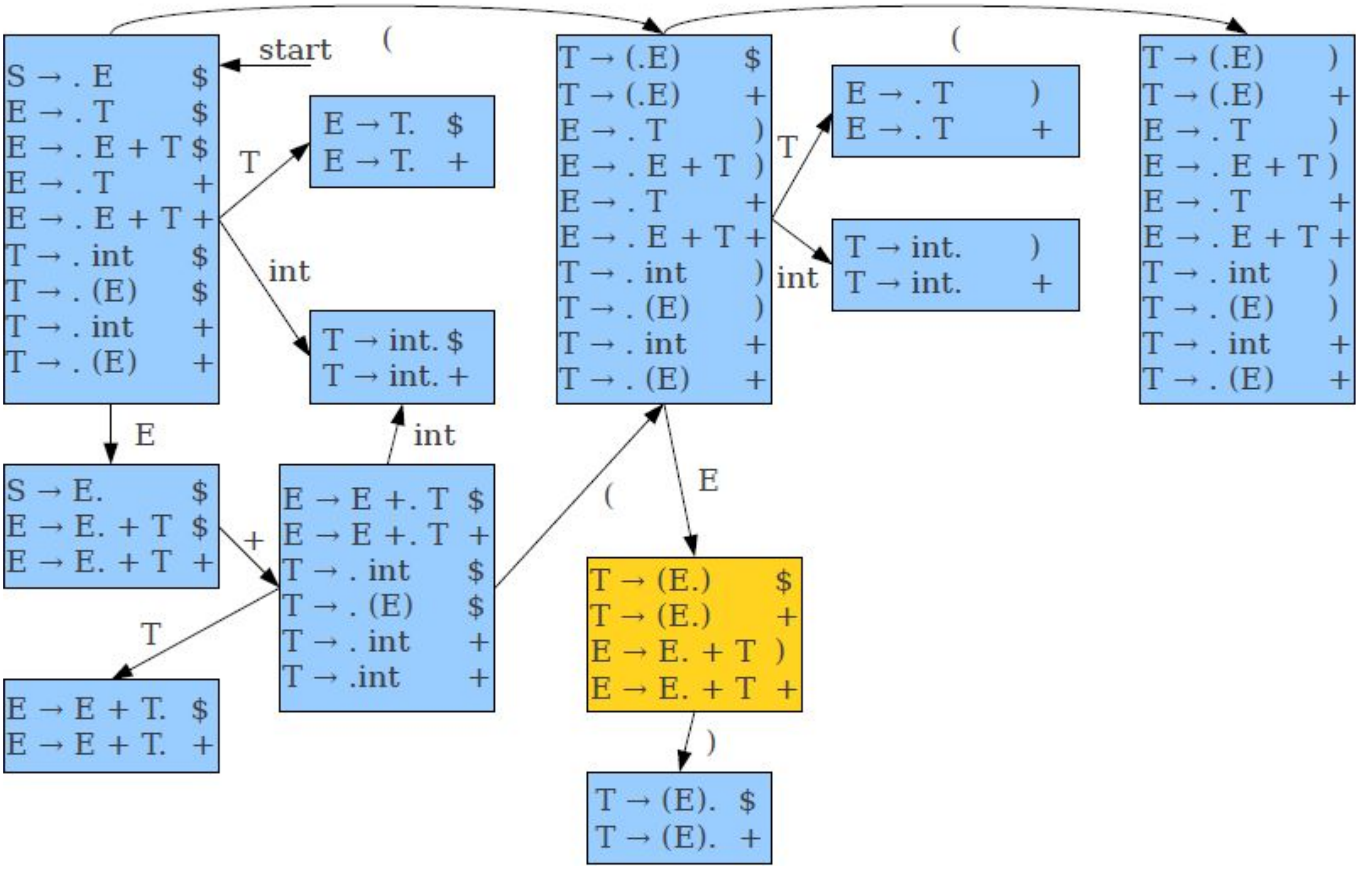
Deterministic LR(1) Automata



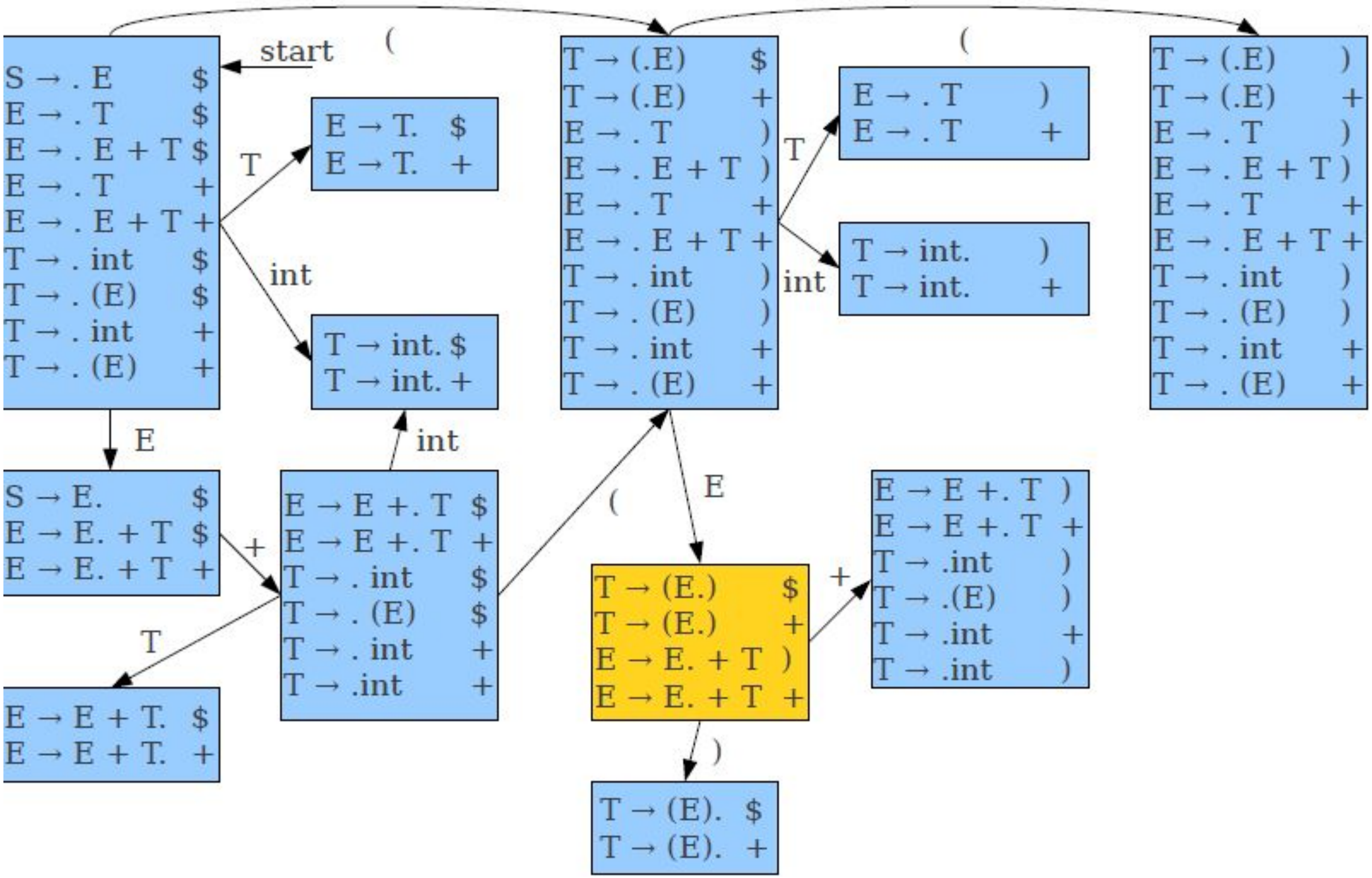
Deterministic LR(1) Automata



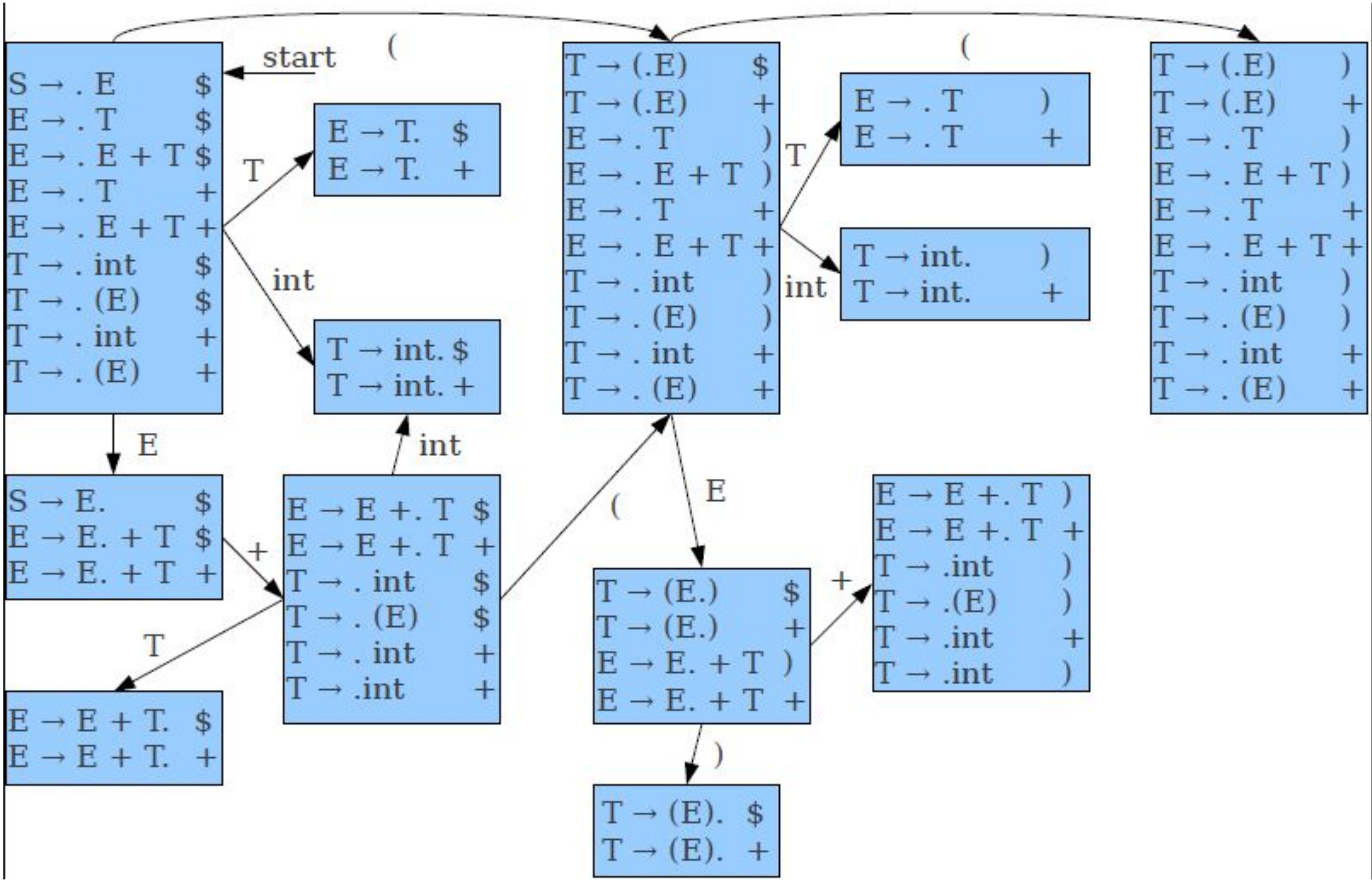
Deterministic LR(1) Automata



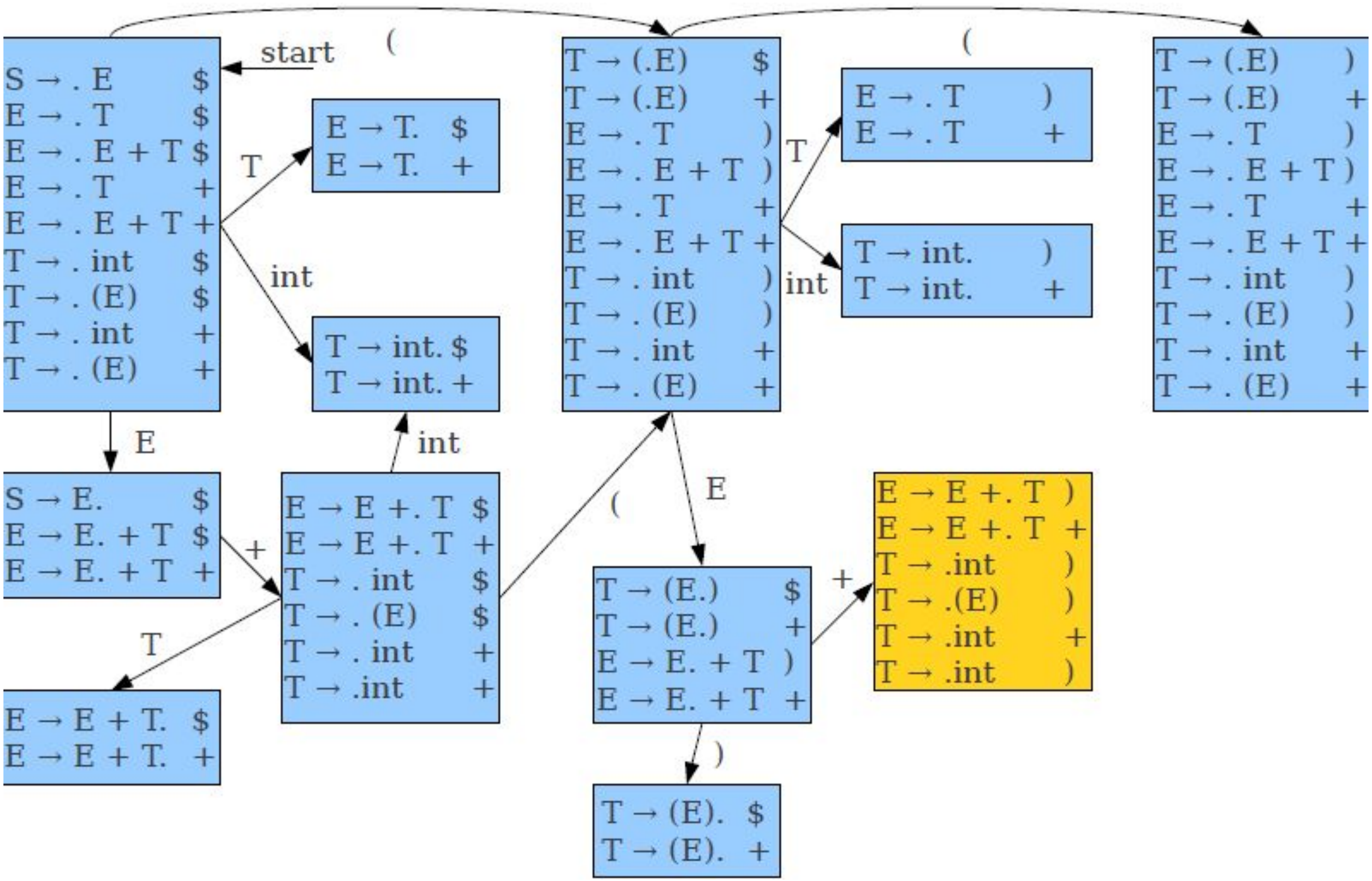
Deterministic LR(1) Automata



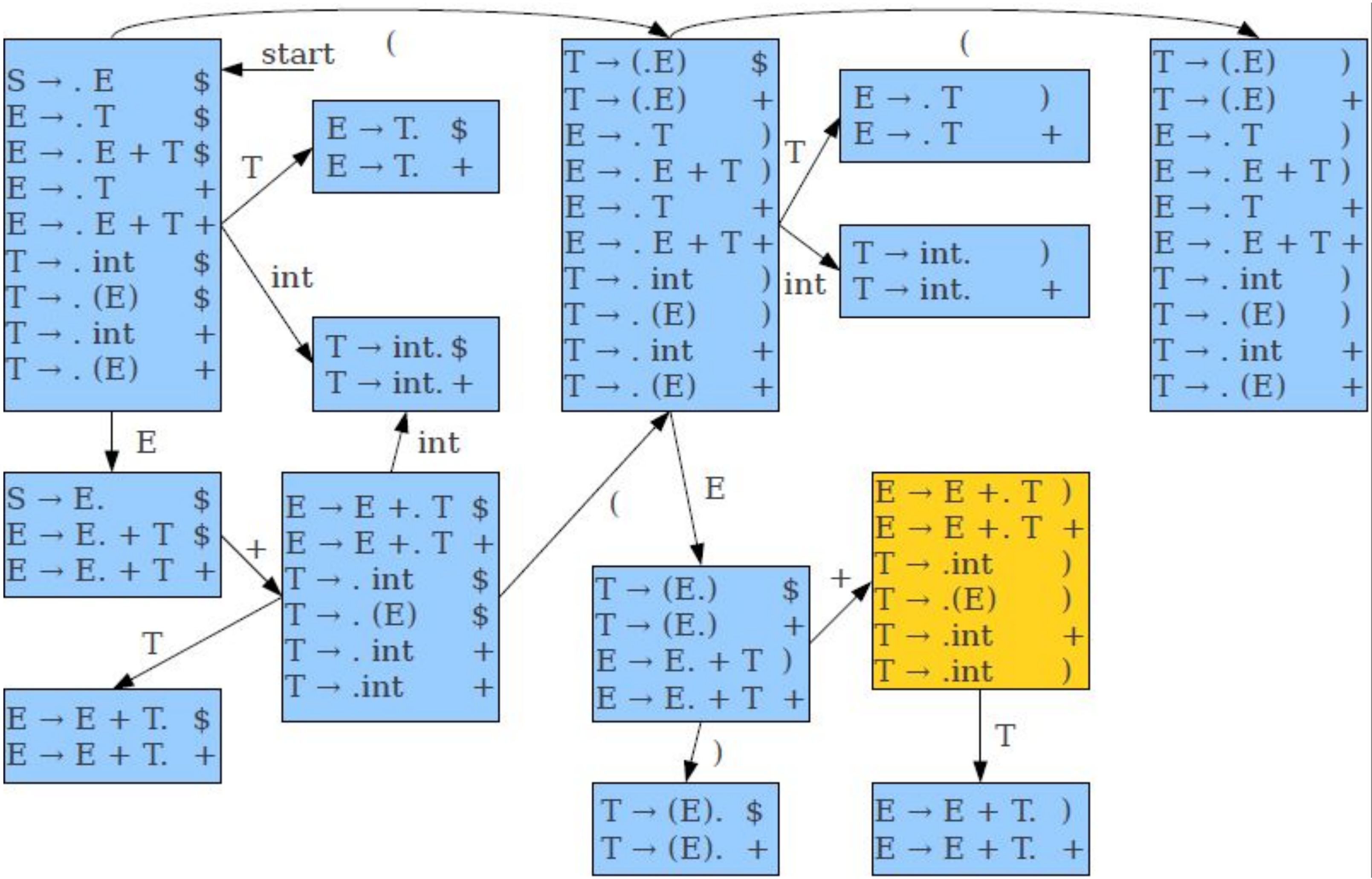
Deterministic LR(1) Automata



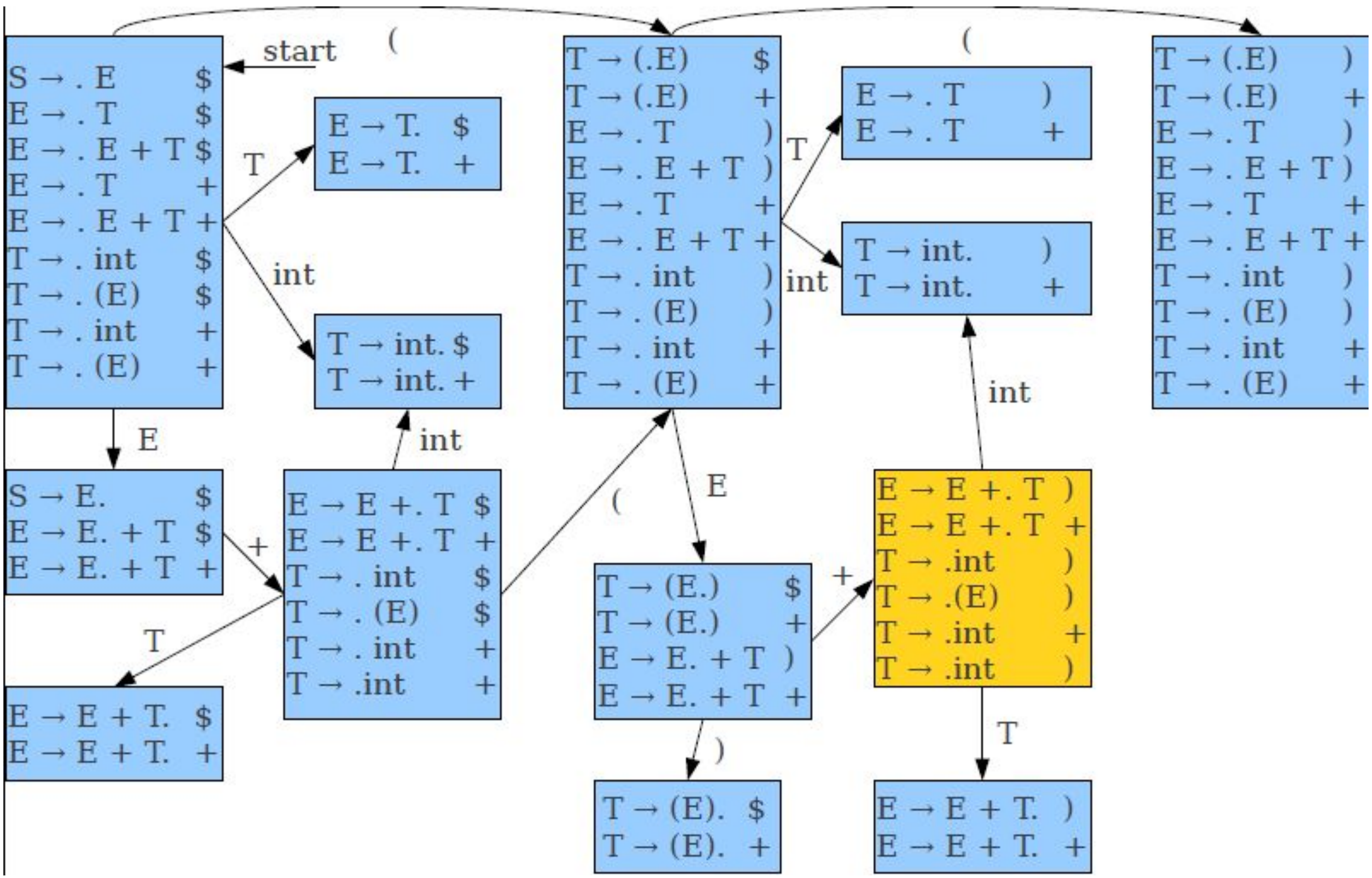
Deterministic LR(1) Automata



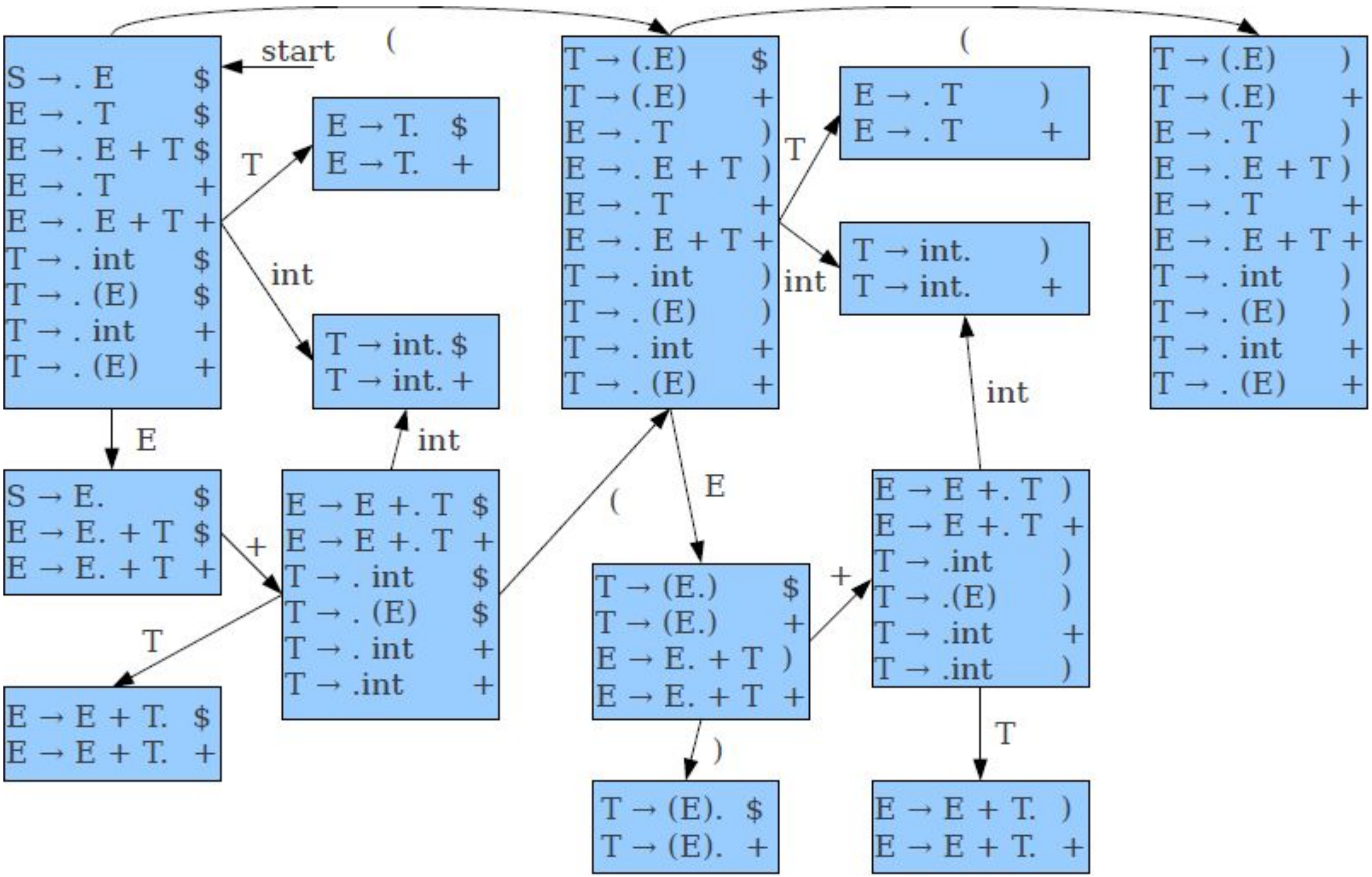
Deterministic LR(1) Automata



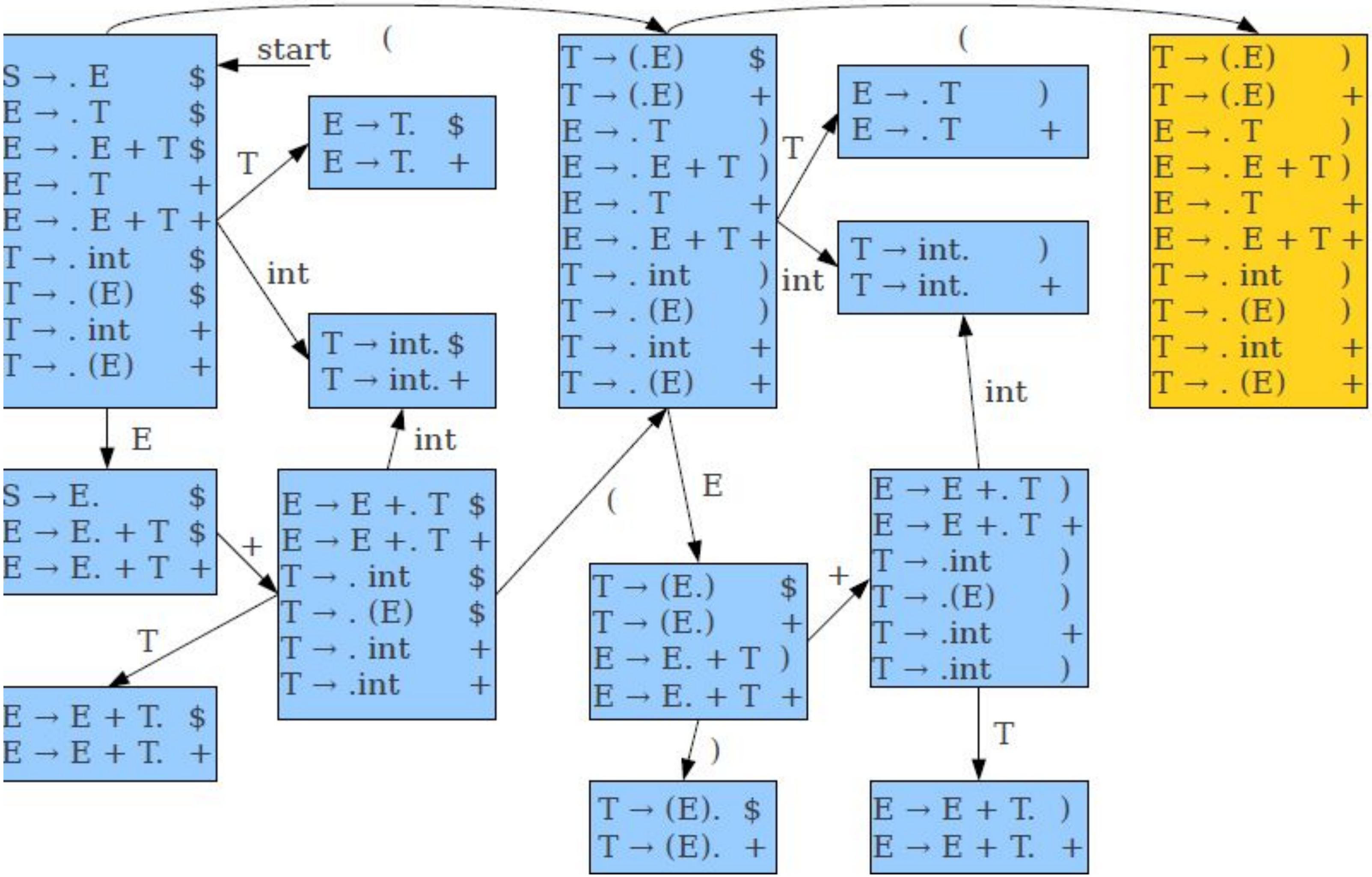
Deterministic LR(1) Automata



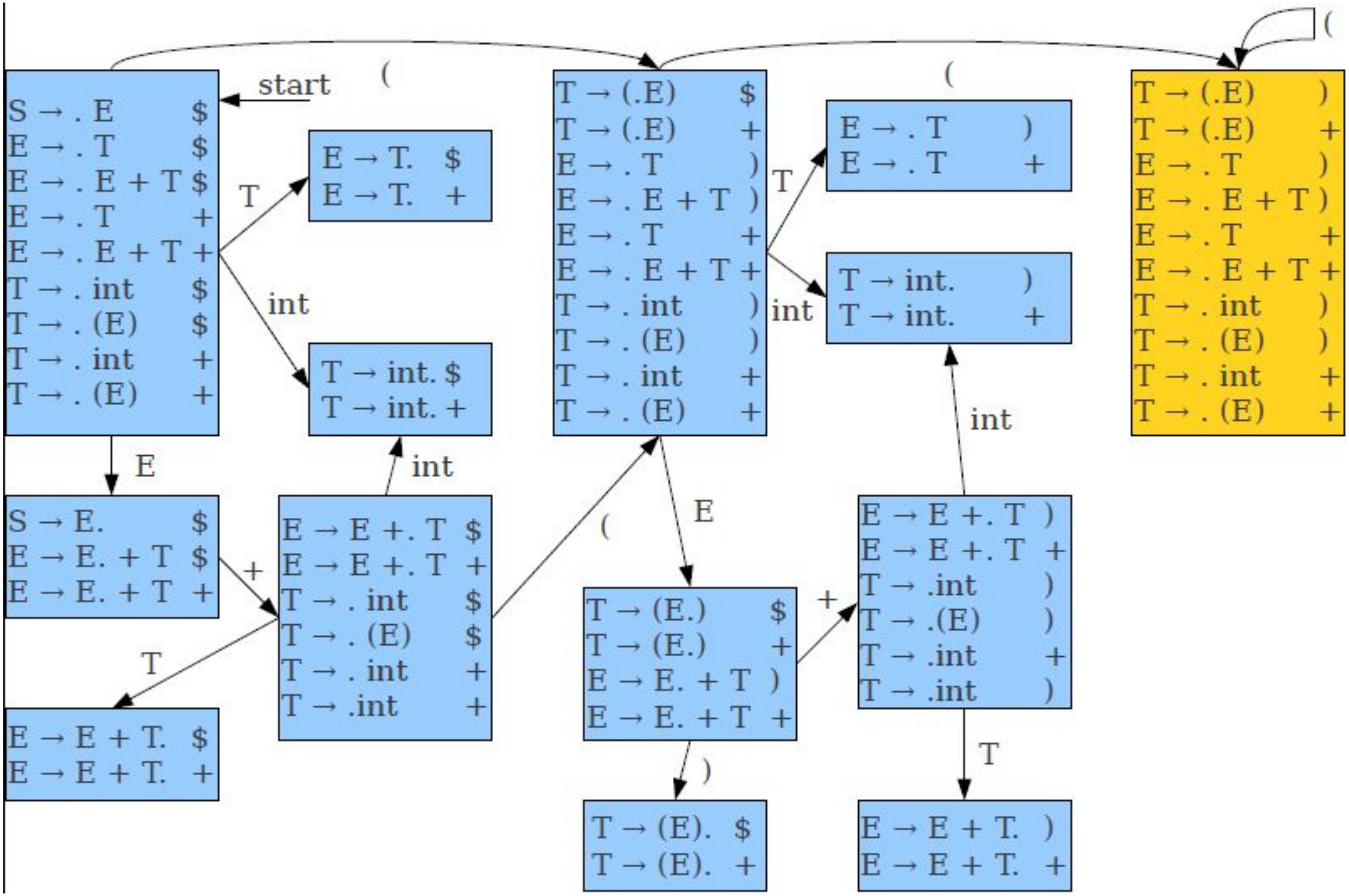
Deterministic LR(1) Automata



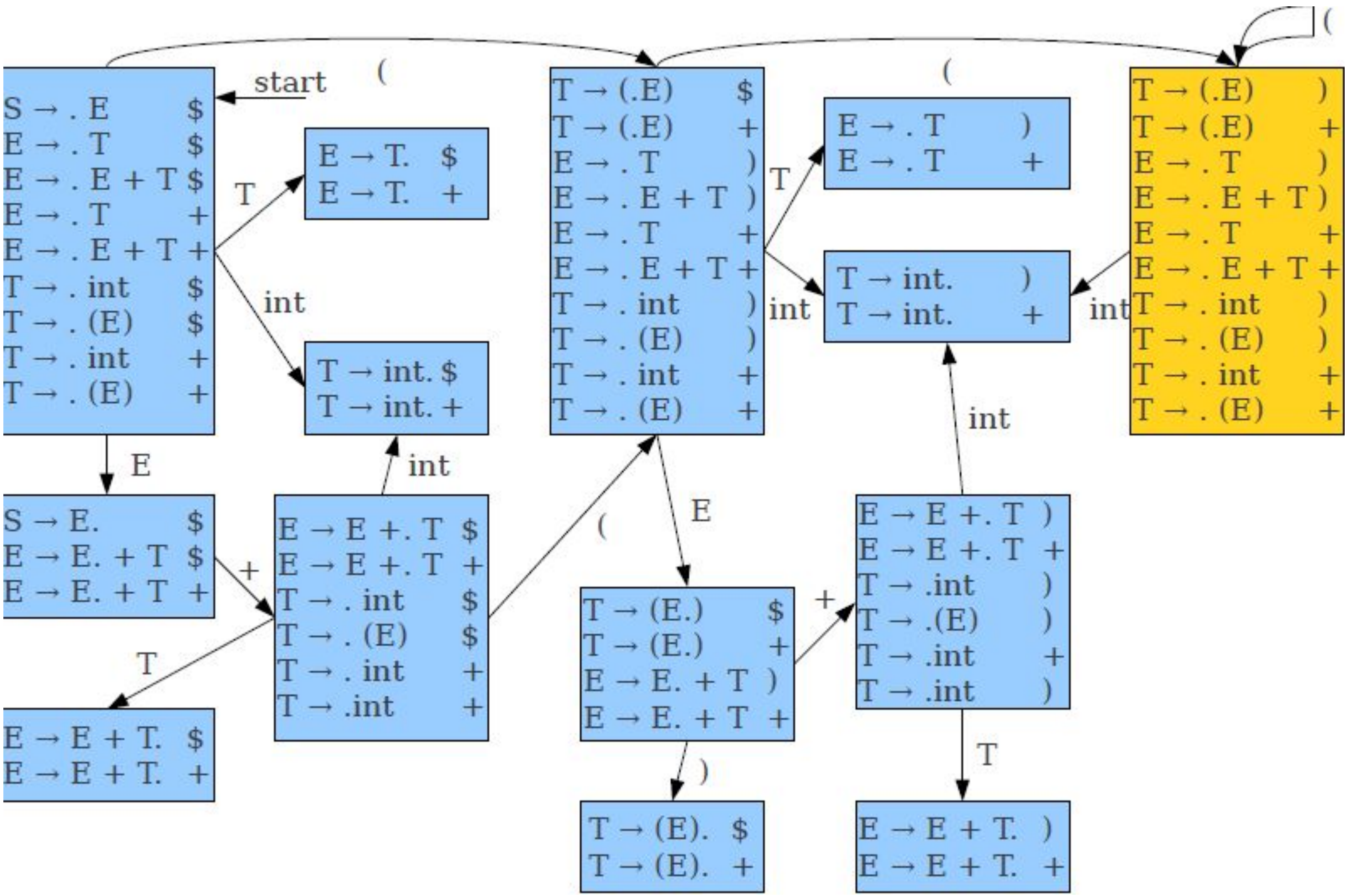
Deterministic LR(1) Automata



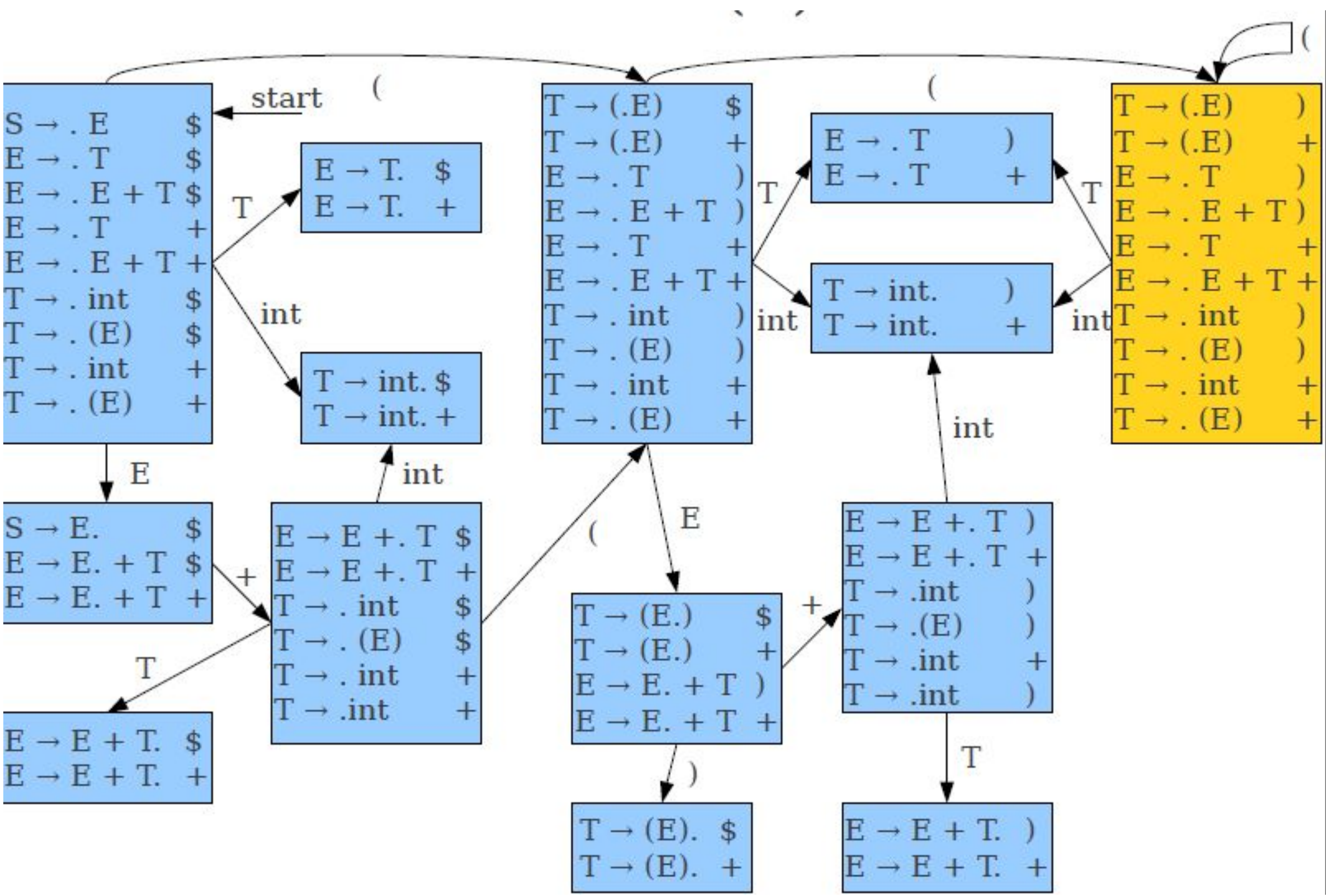
Deterministic LR(1) Automata



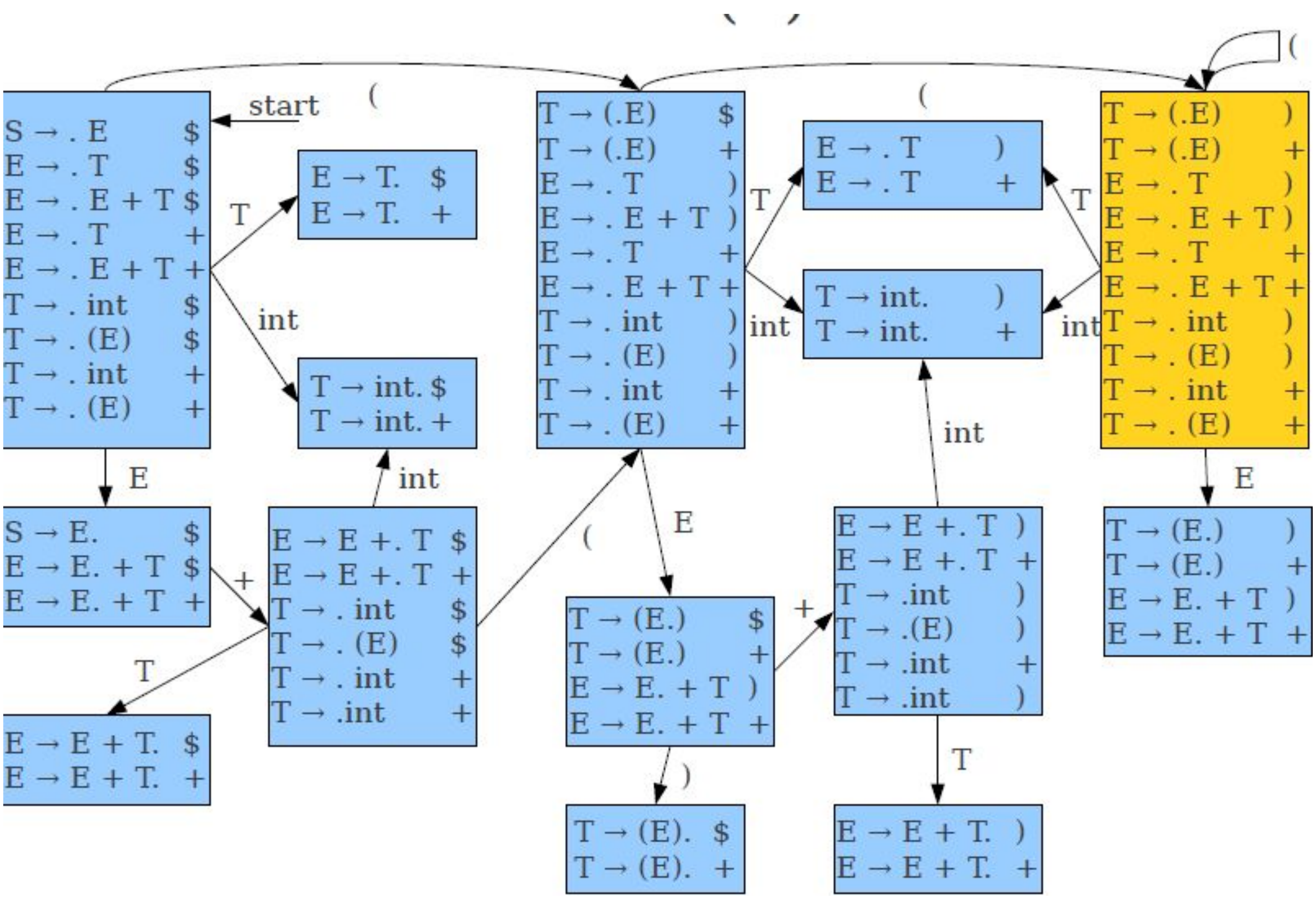
Deterministic LR(1) Automata



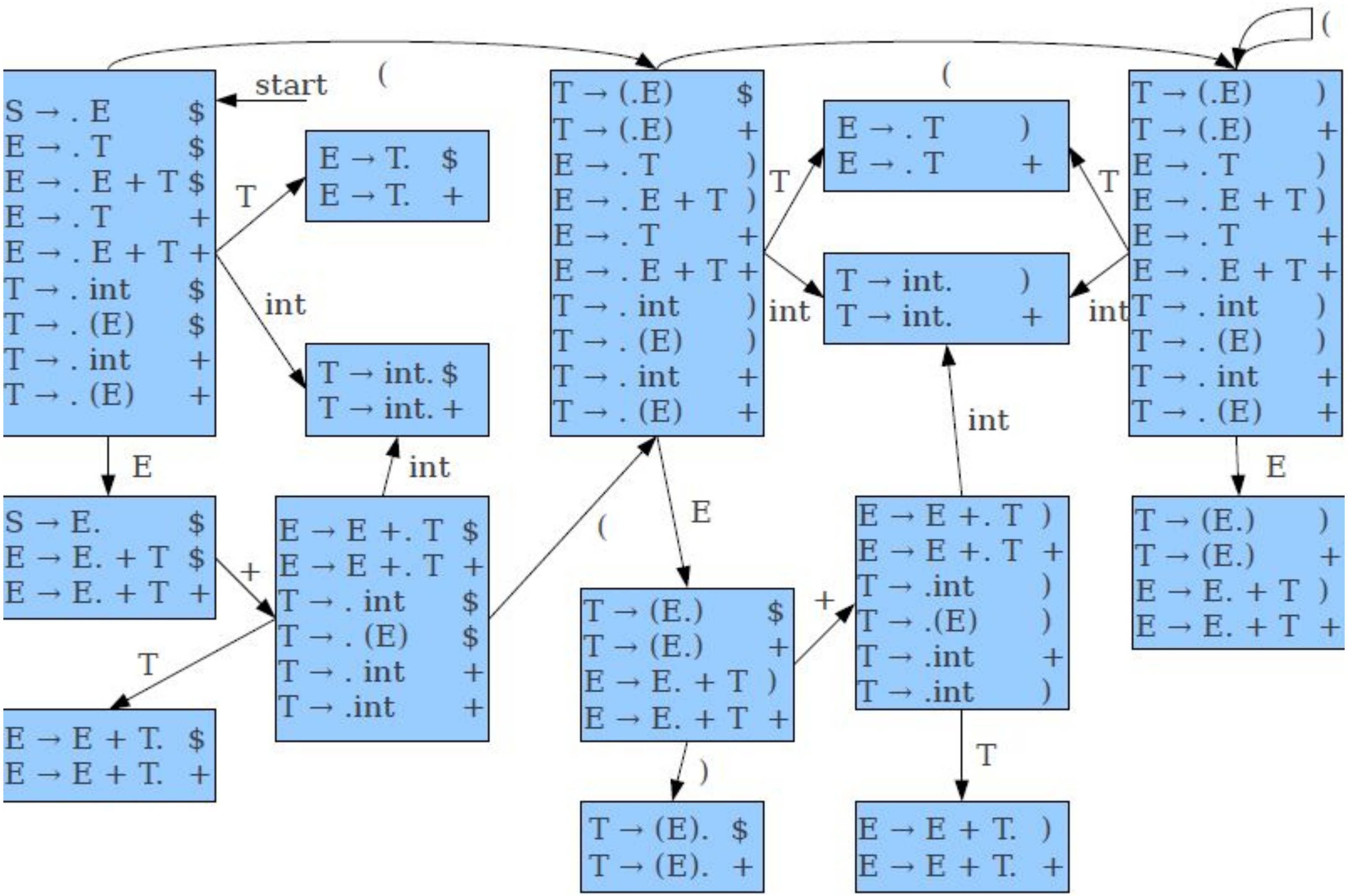
Deterministic LR(1) Automata



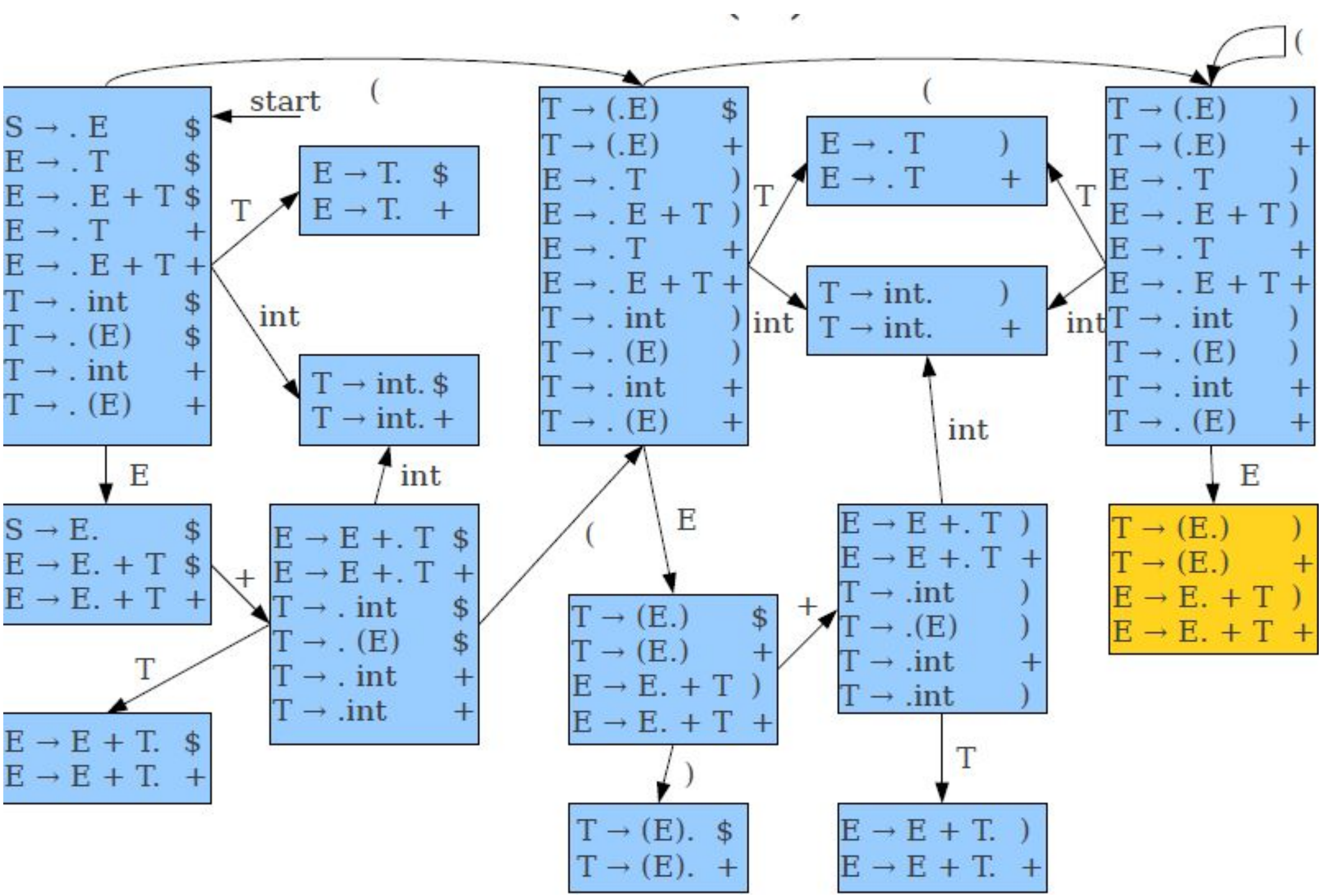
Deterministic LR(1) Automata



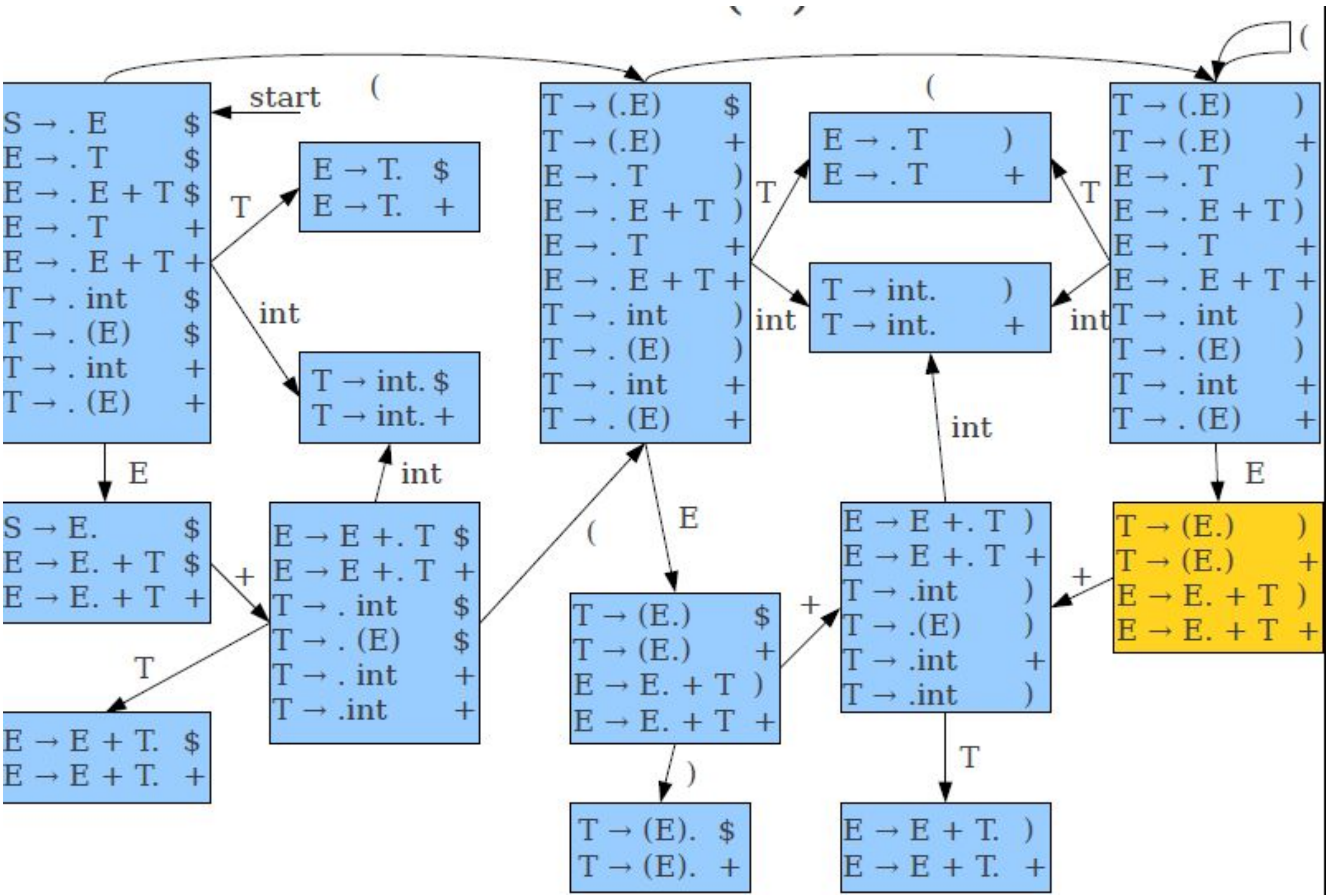
Deterministic LR(1) Automata



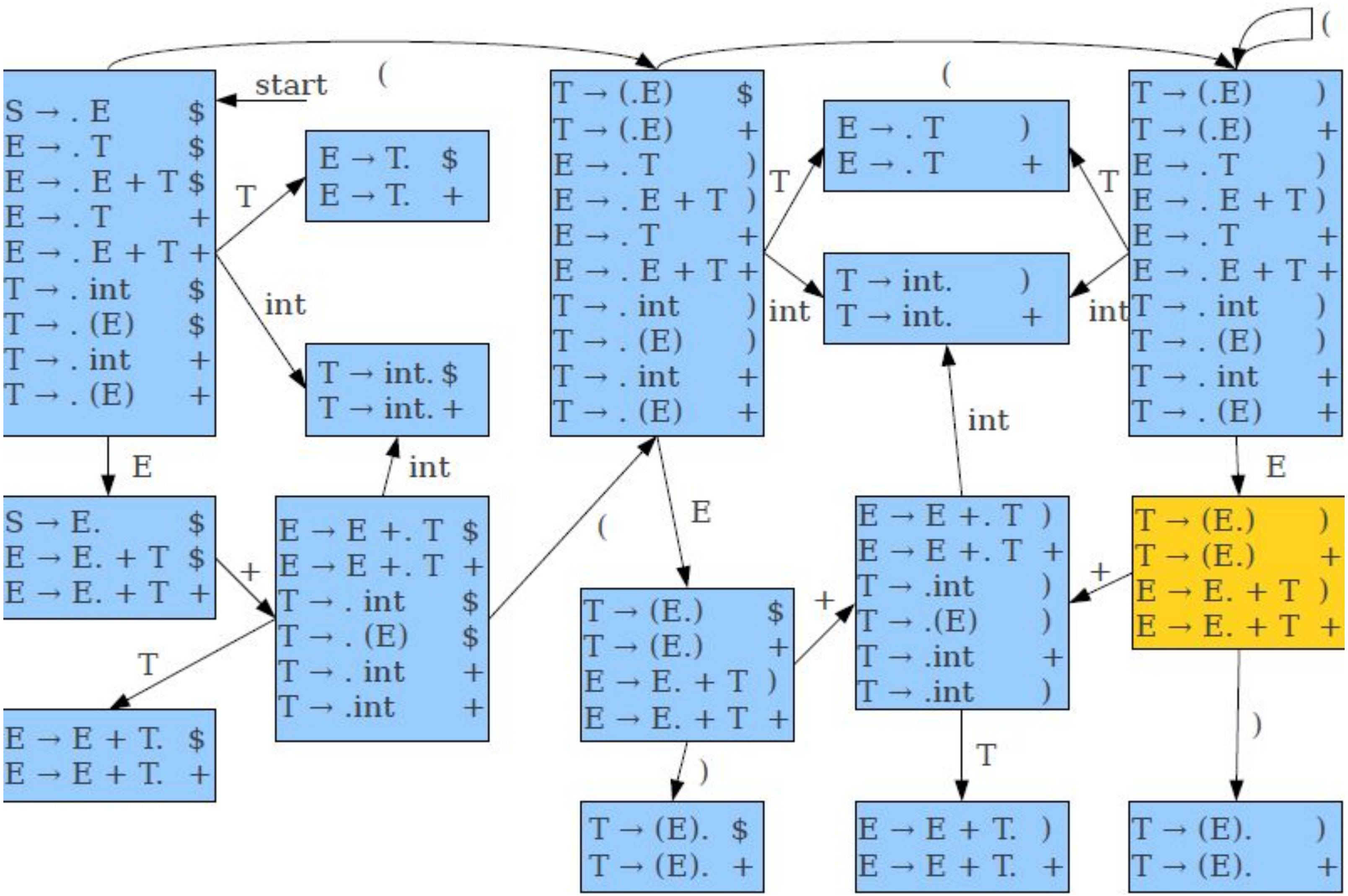
Deterministic LR(1) Automata



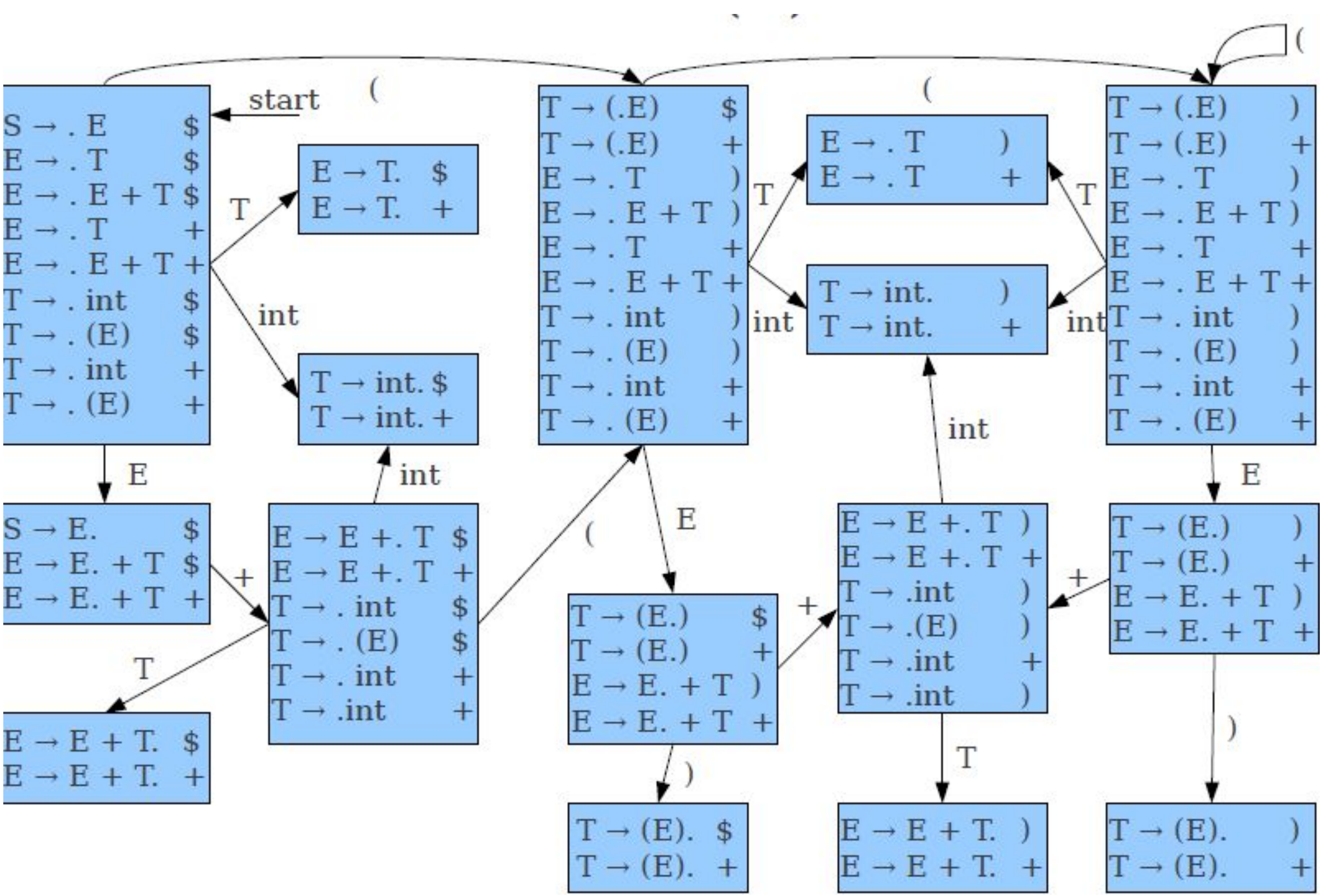
Deterministic LR(1) Automata



Deterministic LR(1) Automata



Deterministic LR(1) Automata



$S \longrightarrow E$ (1)
$E \longrightarrow T$ (2)
$E \longrightarrow E + T$ (3)
$T \longrightarrow$ int (4)
$T \longrightarrow (E)$ (5)

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

The LR(1) Parsing Algorithm

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

Constructing LR(1) Parse Tables

- For each state X:
  - If there is a production $\mathbf{A} \to \omega \cdot$ [**t**], set **action**[X, **t**] = reduce $\mathbf{A} \to \omega$.
  - If there is the special production $\mathbf{S} \to \mathbf{E} \cdot$ [**$**], 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).