# Bottom Up Parsing

TEACHING ASSISTANT: DAVID TRABISH

# LR(0) Parsing

- Build the derivation tree from the bottom
- First build the children, then connect to the parent
- Can handle left recursion
  - Which is common in real-world grammars

# LR(0) Item

An **LR(0) item** is of the form:

- $N \rightarrow \alpha . \beta$

The **dot** gives us the current location (a local view).

# LR(0) Item

An LR(0) item with the dot at the end is called **reduce** item:

- $N \rightarrow \alpha\beta.$

Otherwise, it's a **shift** item:

- $N \rightarrow .\alpha\beta$
- $N \rightarrow \alpha.\beta$

# LR(0) Item Closure Set

The **LR(0) closure set** of an LR(0) item $i$ is a set $S$ such that:

- $i \in S$
- If $A \to \alpha . N\beta \in S$ then for each rule $N \to \gamma$:
  - $N \to . \gamma \in S$

# LR(0) Item Closure Set

For example, given the following CFG:

- $S \rightarrow E\$$
- $E \rightarrow ID = X$
- $E \rightarrow \{ID\}$
- $X \rightarrow INT$

the closure set of the $S \rightarrow .E\$$ contains:

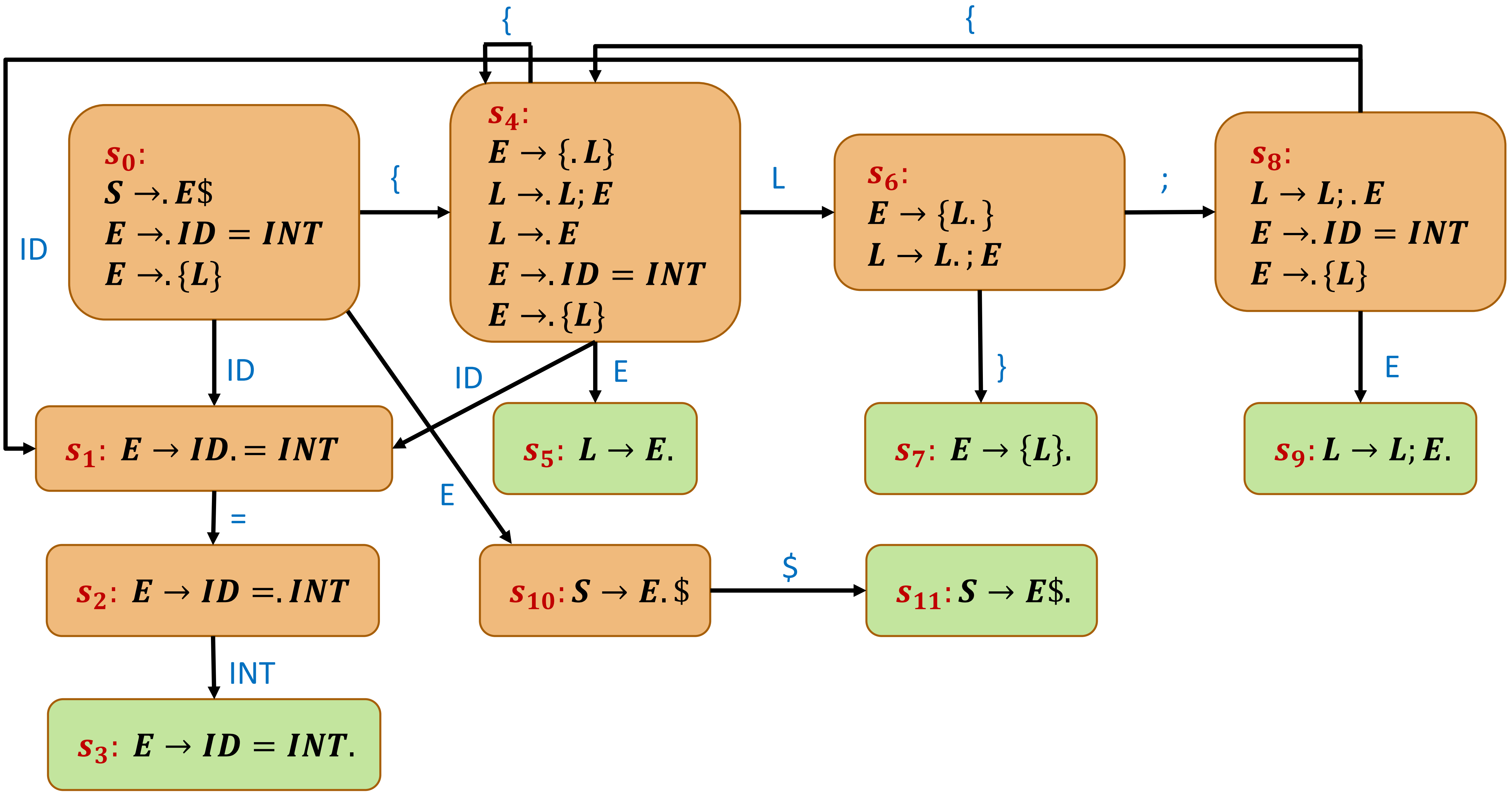
- $S \rightarrow .E\$$
- $E \rightarrow .ID = X$
- $E \rightarrow .\{ID\}$

# LR(0) Parsing

Consider the following CFG:

- $S \rightarrow E\$$
- $E \rightarrow ID = INT$
- $E \rightarrow \{L\}$
- $L \rightarrow E$
- $L \rightarrow L; E$

What will be the **transition system** of the LR(0) parser for this CFG?



$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =.INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L;E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.;E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L;.E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L;E.$
$s_{10}$: $S \to E.\$$
$s_{11}$: $S \to E\$.$
{
ID
L
;
=
INT
E
}
$

# LR(0) Parser

We start with the initial LR(0) item (that comes from the initial rule):

- $S \rightarrow .E\$$

The initial state is the $\epsilon$-**closure** of that item, which contains:

$$
\begin{aligned}
&S \rightarrow E\$ \\
&E \rightarrow ID = INT \\
&E \rightarrow \{L\} \\
&L \rightarrow E \\
&L \rightarrow L; E
\end{aligned}
$$

# LR(0) Parser

We start with the initial LR(0) item (that comes from the initial rule):

- $S \rightarrow .E\$$

The initial state is the $\epsilon$-closure of that item, which contains:

- $S \rightarrow .E\$$
- $E \rightarrow .ID = INT$
- $E \rightarrow .\{L\}$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$

$s_0$:

$S \to .E\$$

$E \to .ID = INT$

$E \to .\{L\}$

# LR(0) Parser

From $s_0$, if we recognized $ID$, then the next state will contain:

- $E \rightarrow ID. = INT$

So the next state (the $\epsilon$-closure) contains:

$$S \rightarrow E\$$$
$$E \rightarrow ID = INT$$
$$E \rightarrow \{L\}$$
$$L \rightarrow E$$
$$L \rightarrow L; E$$

# LR(0) Parser

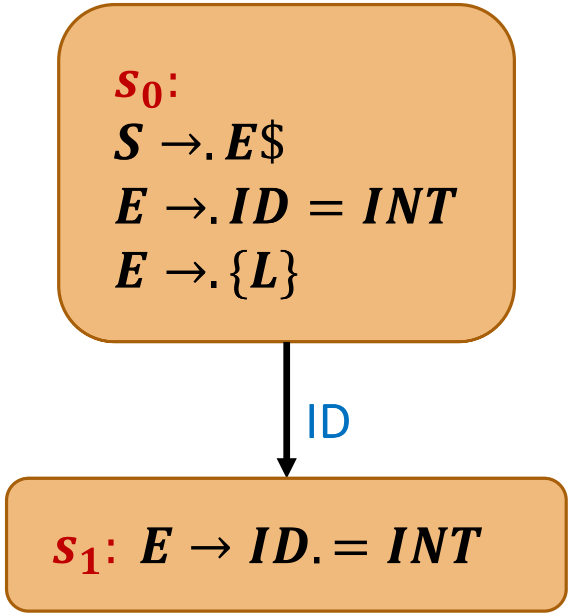
From $s_0$, if we recognized $ID$, then the next state will contain:

- $E \rightarrow ID. = INT$

So the next state (the $\epsilon$-closure) contains:

- $E \rightarrow ID. = INT$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$


$s_0$:
$S \rightarrow .E\$$
$E \rightarrow .ID = INT$
$E \rightarrow .\{L\}$
ID
$s_1$: $E \rightarrow ID. = INT$

# LR(0) Parser

From $s_1$, if we recognized $=$, then the next state will contain:

- $E \rightarrow ID =. INT$

So the next state (the $\epsilon$-closure) contains:

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$

# LR(0) Parser

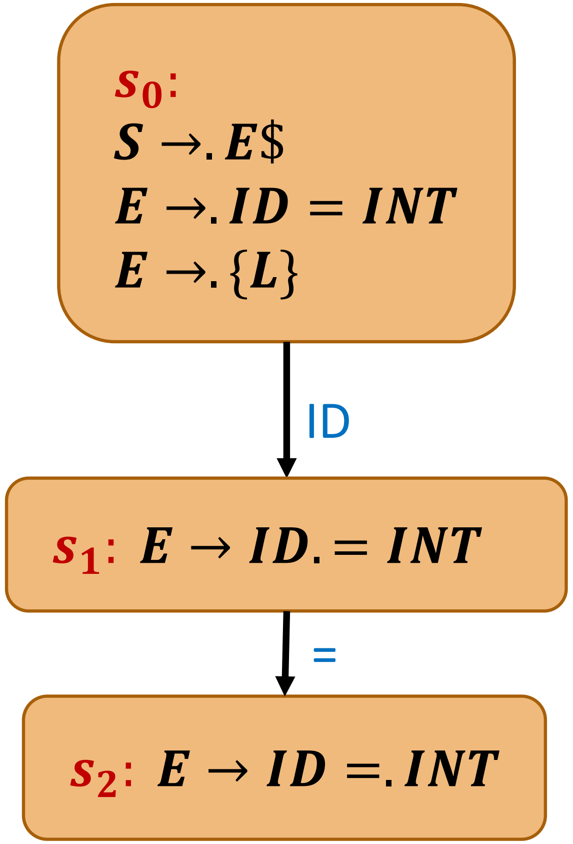
From $s_1$, if we recognized $=$, then the next state will contain:

- $E \rightarrow ID =. INT$

So the next state (the $\epsilon$-closure) contains:

- $E \rightarrow ID =. INT$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$


$s_0$:
$S \rightarrow .E\$$
$E \rightarrow .ID = INT$
$E \rightarrow .\{L\}$
ID
$s_1$: $E \rightarrow ID. = INT$
=
$s_2$: $E \rightarrow ID =. INT$

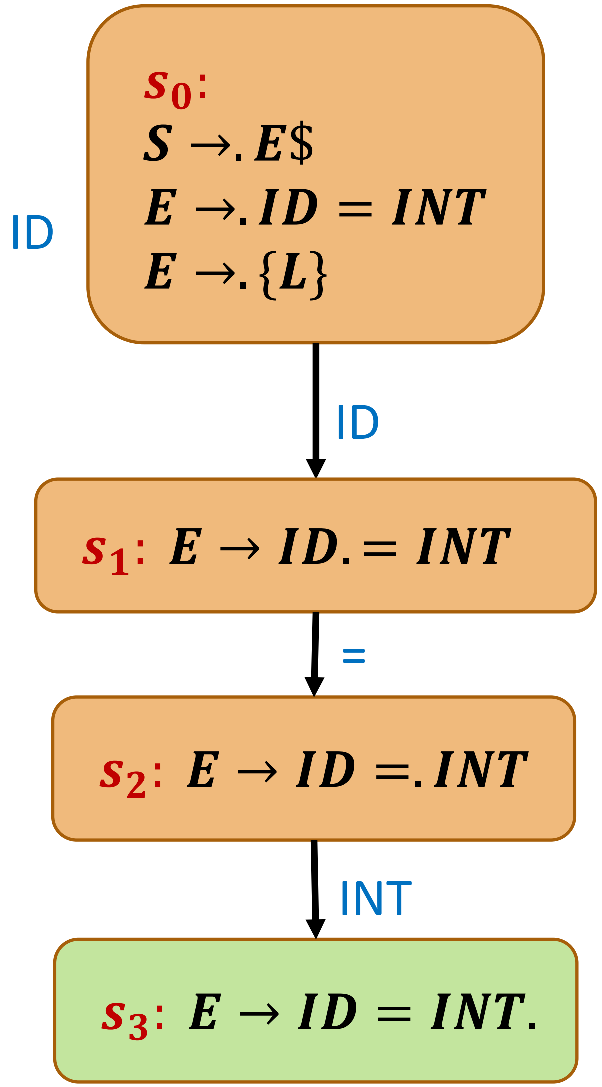
# LR(0) Parser

From $s_2$, if we recognized $INT$, then the next state will contain:

- $E \rightarrow ID = INT.$

Which is a reduce state.

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$


ID
$s_0$:
$S \rightarrow .E\$$
$E \rightarrow .ID = INT$
$E \rightarrow .\{L\}$
ID
$s_1$: $E \rightarrow ID. = INT$
=
$s_2$: $E \rightarrow ID =. INT$
INT
$s_3$: $E \rightarrow ID = INT.$

# LR(0) Parser

From $s_0$, if we recognized $\{$, then the next state will contain:

- $E \rightarrow \{.L\}$

So the next state (the $\epsilon$-closure) contains:

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$

# LR(0) Parser

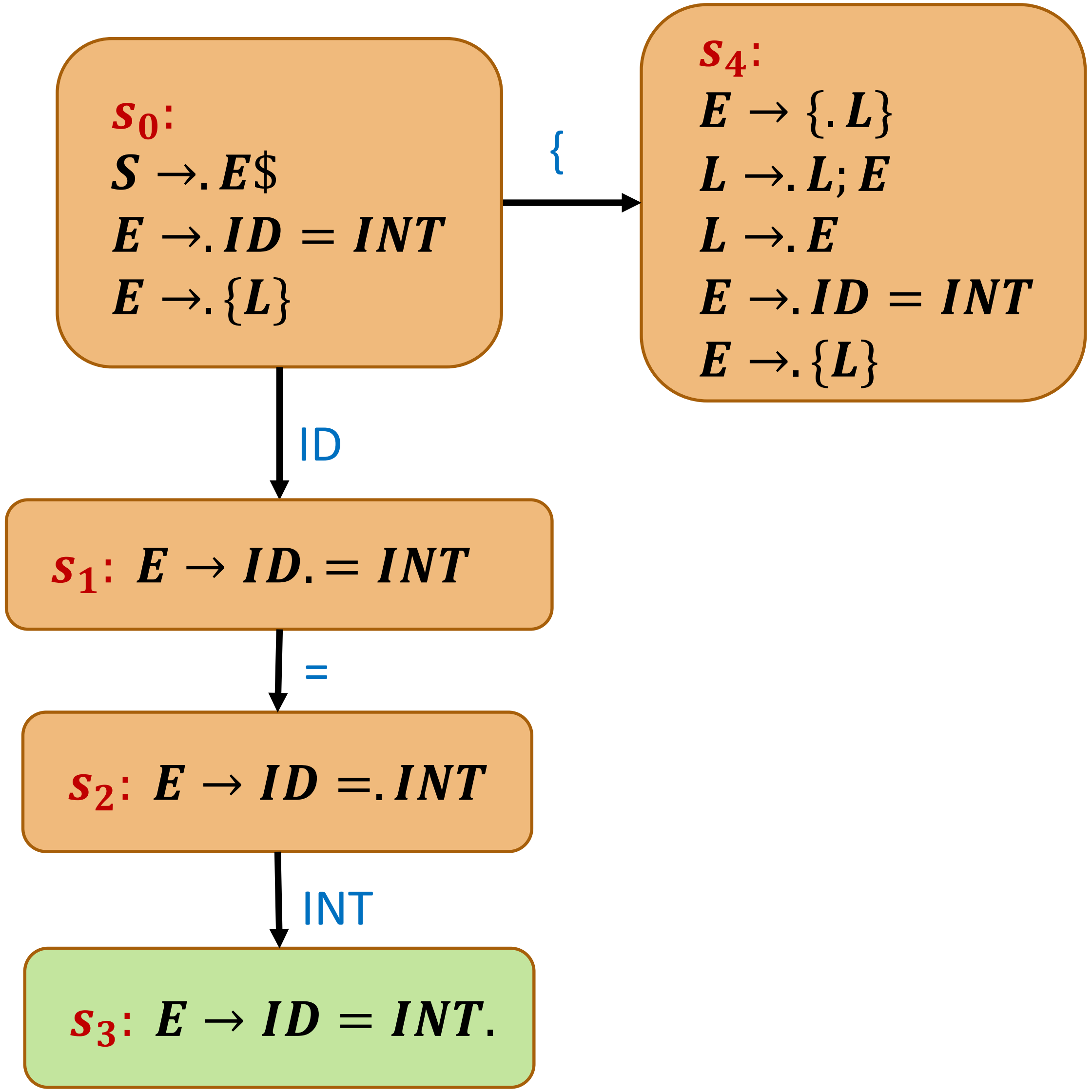
From $s_0$, if we recognized {, then the next state will contain:

- $E \rightarrow \{.L\}$

So the next state (the $\epsilon$-closure) contains:

- $E \rightarrow \{.L\}$
- $L \rightarrow .L;E$
- $L \rightarrow .E$
- $E \rightarrow .ID = INT$
- $E \rightarrow .\{L\}$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L;E$



$s_0$:
$S \to .E\$$
$E \to .ID = INT$
$E \to .\{L\}$
{
$s_4$:
$E \to \{.L\}$
$L \to .L; E$
$L \to .E$
$E \to .ID = INT$
$E \to .\{L\}$
ID
$s_1$: $E \to ID. = INT$
=
$s_2$: $E \to ID =. INT$
INT
$s_3$: $E \to ID = INT.$

# LR(0) Parser

From $s_4$, if we recognized $\{$, then the next state will contain:

- $E \rightarrow \{.L\}$

So the next state (the $\epsilon$-closure) contains:

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$

# LR(0) Parser

From $s_4$, if we recognized {, then the next state will contain:

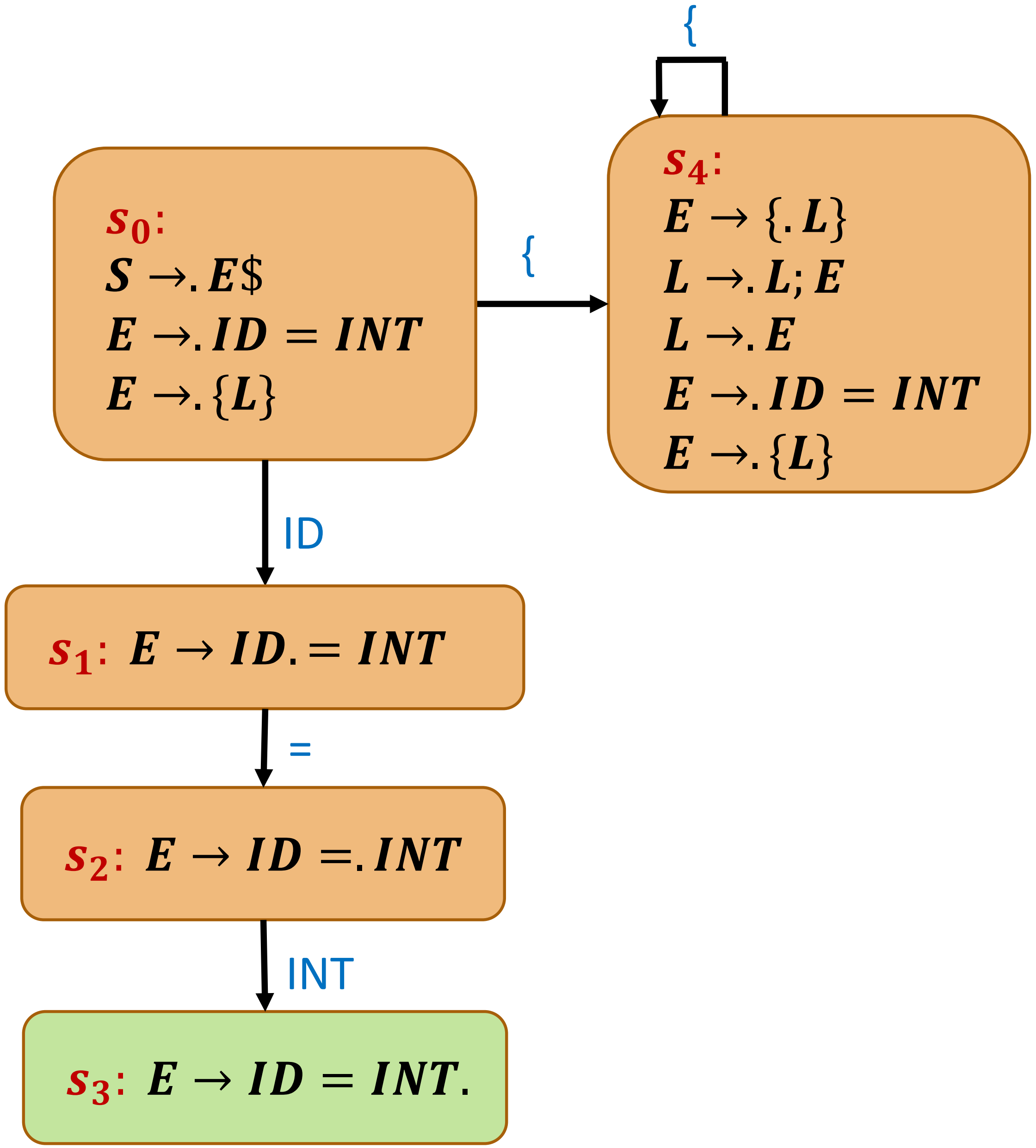
- $E \rightarrow \{.L\}$

So the next state (the $\epsilon$-closure) contains:

- $E \rightarrow \{.L\}$
- $L \rightarrow .L; E$
- $L \rightarrow .E$
- $E \rightarrow .ID = INT$
- $E \rightarrow .\{L\}$

which was already computed: $s_4$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$



$s_0$:
$S \rightarrow .E\$$
$E \rightarrow .ID = INT$
$E \rightarrow .\{L\}$
{
$s_4$:
$E \rightarrow \{.L\}$
$L \rightarrow .L; E$
$L \rightarrow .E$
$E \rightarrow .ID = INT$
$E \rightarrow .\{L\}$
{
ID
$s_1$: $E \rightarrow ID. = INT$
=
$s_2$: $E \rightarrow ID =. INT$
INT
$s_3$: $E \rightarrow ID = INT.$

# LR(0) Parser

From $s_4$, if we recognized $ID$, then the next state will contain:

- $E \rightarrow ID. = INT$

So the next state (the $\epsilon$-closure) contains:

$$S \rightarrow E\$$$
$$E \rightarrow ID = INT$$
$$E \rightarrow \{L\}$$
$$L \rightarrow E$$
$$L \rightarrow L; E$$

# LR(0) Parser

From $s_4$, if we recognized $ID$, then the next state will contain:

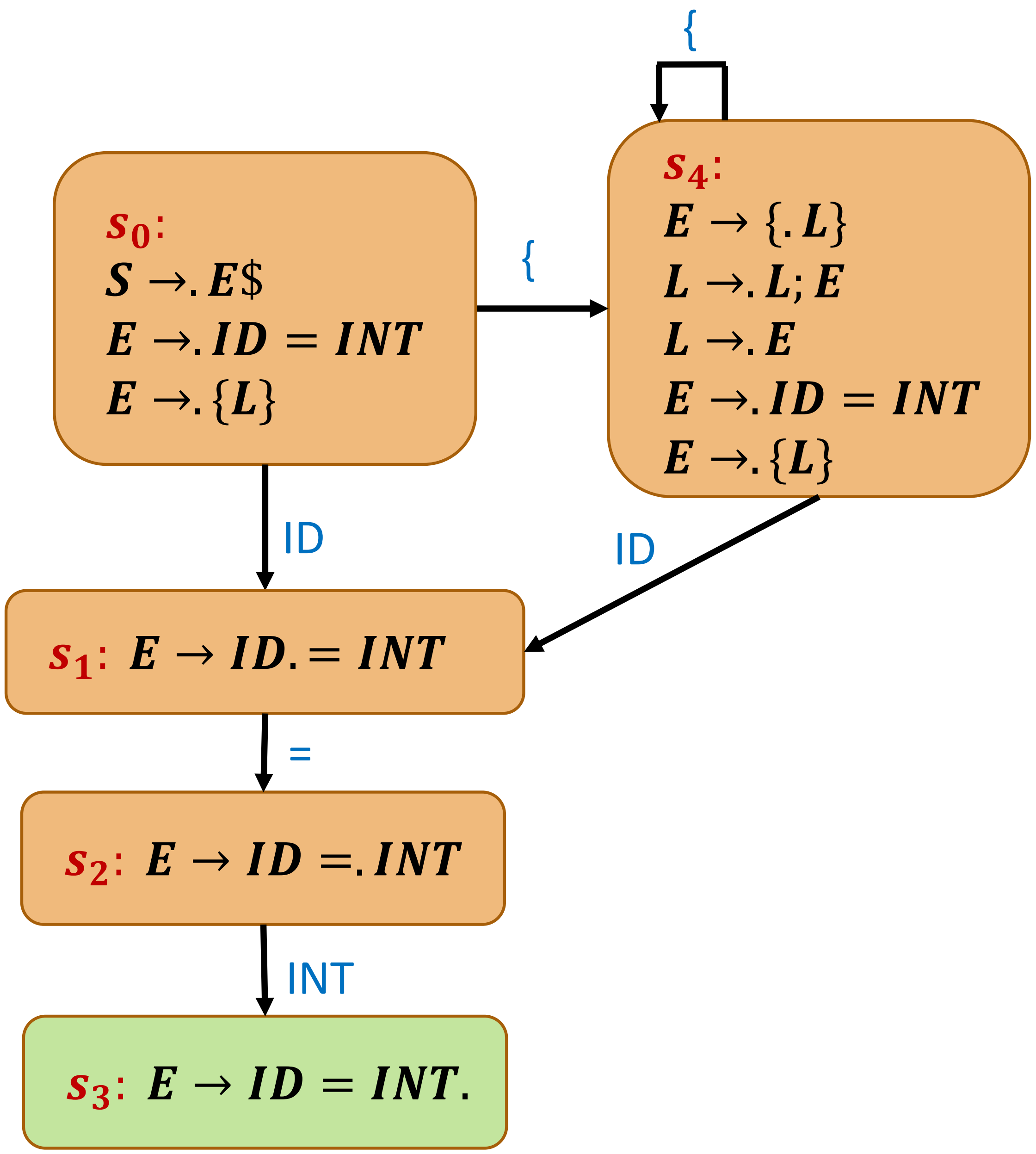
- $E \rightarrow ID. = INT$

So the next state (the $\epsilon$-closure) contains:

- $E \rightarrow ID. = INT$

which was already computed: $s_1$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$


$s_0$: $S \to .E\$$
$E \to .ID = INT$
$E \to .\{L\}$
{
$s_4$: $E \to \{.L\}$
$L \to .L; E$
$L \to .E$
$E \to .ID = INT$
$E \to .\{L\}$
{
ID
ID
$s_1$: $E \to ID. = INT$
=
$s_2$: $E \to ID =. INT$
INT
$s_3$: $E \to ID = INT.$

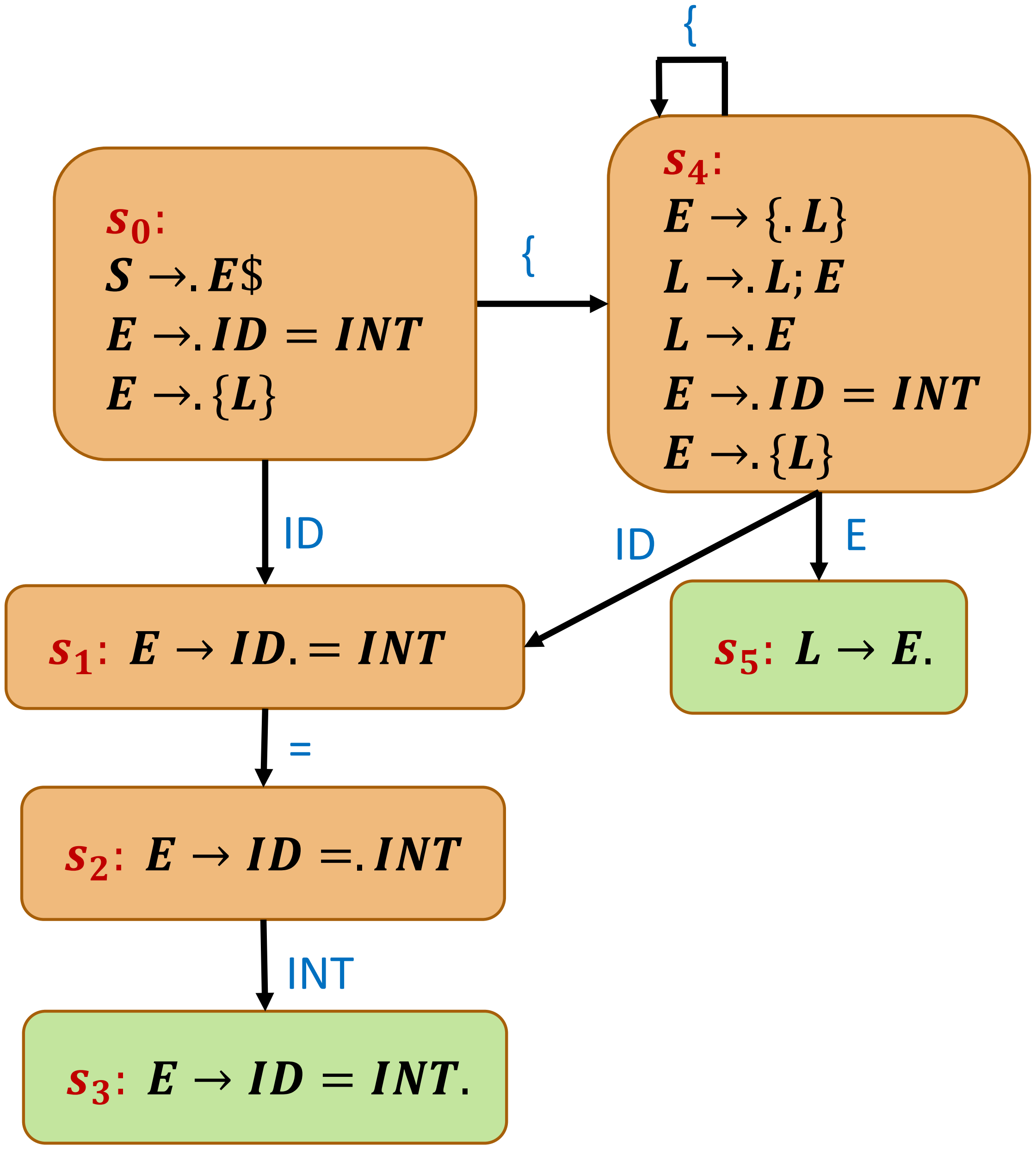
# LR(0) Parser

From $s_4$, if we recognized $E$, then the next state will contain:

- $L \rightarrow E.$

which is a reduce state.

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$


$s_0$:
$S \rightarrow .E\$$
$E \rightarrow .ID = INT$
$E \rightarrow .\{L\}$
{
$s_4$:
$E \rightarrow \{.L\}$
$L \rightarrow .L; E$
$L \rightarrow .E$
$E \rightarrow .ID = INT$
$E \rightarrow .\{L\}$
{
ID
ID
E
$s_1$: $E \rightarrow ID. = INT$
$s_5$: $L \rightarrow E.$
=
$s_2$: $E \rightarrow ID =. INT$
INT
$s_3$: $E \rightarrow ID = INT.$

# LR(0) Parser

From $s_4$, if we recognized $L$, then the next state will contain:

- $E \to \{L.\}$
- $L \to L.;E$

So the next state (the $\epsilon$-closure) contains:

$S \to E\$$
$E \to ID = INT$
$E \to \{L\}$
$L \to E$
$L \to L;E$

# LR(0) Parser

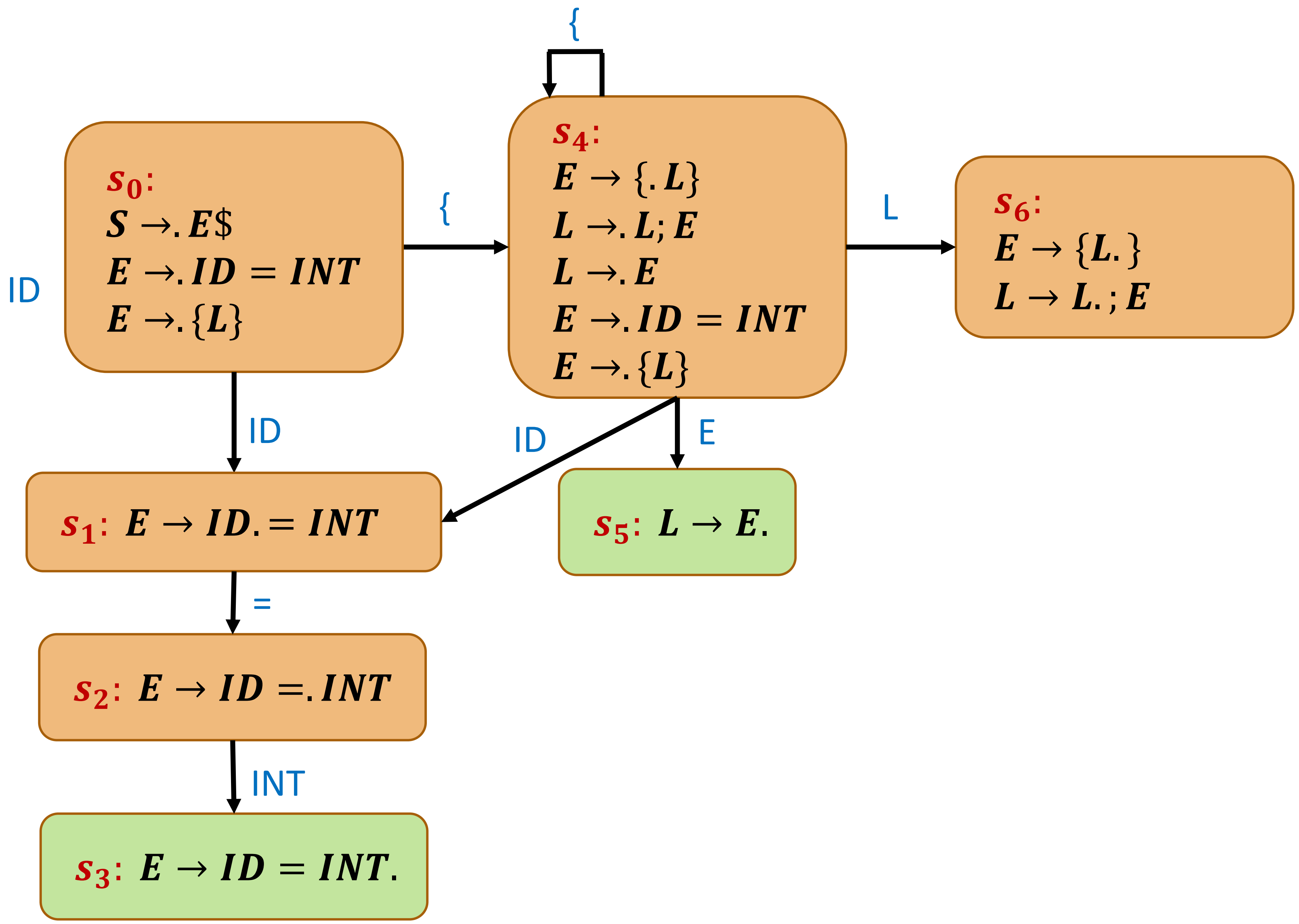
From $s_4$, if we recognized $L$, then the next state will contain:

- $E \rightarrow \{L.\}$
- $L \rightarrow L.;E$

So the next state (the $\epsilon$-closure) contains:

- $E \rightarrow \{L.\}$
- $L \rightarrow L.;E$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L;E$



$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =. INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L; E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.; E$
ID
{
L
E
=
INT

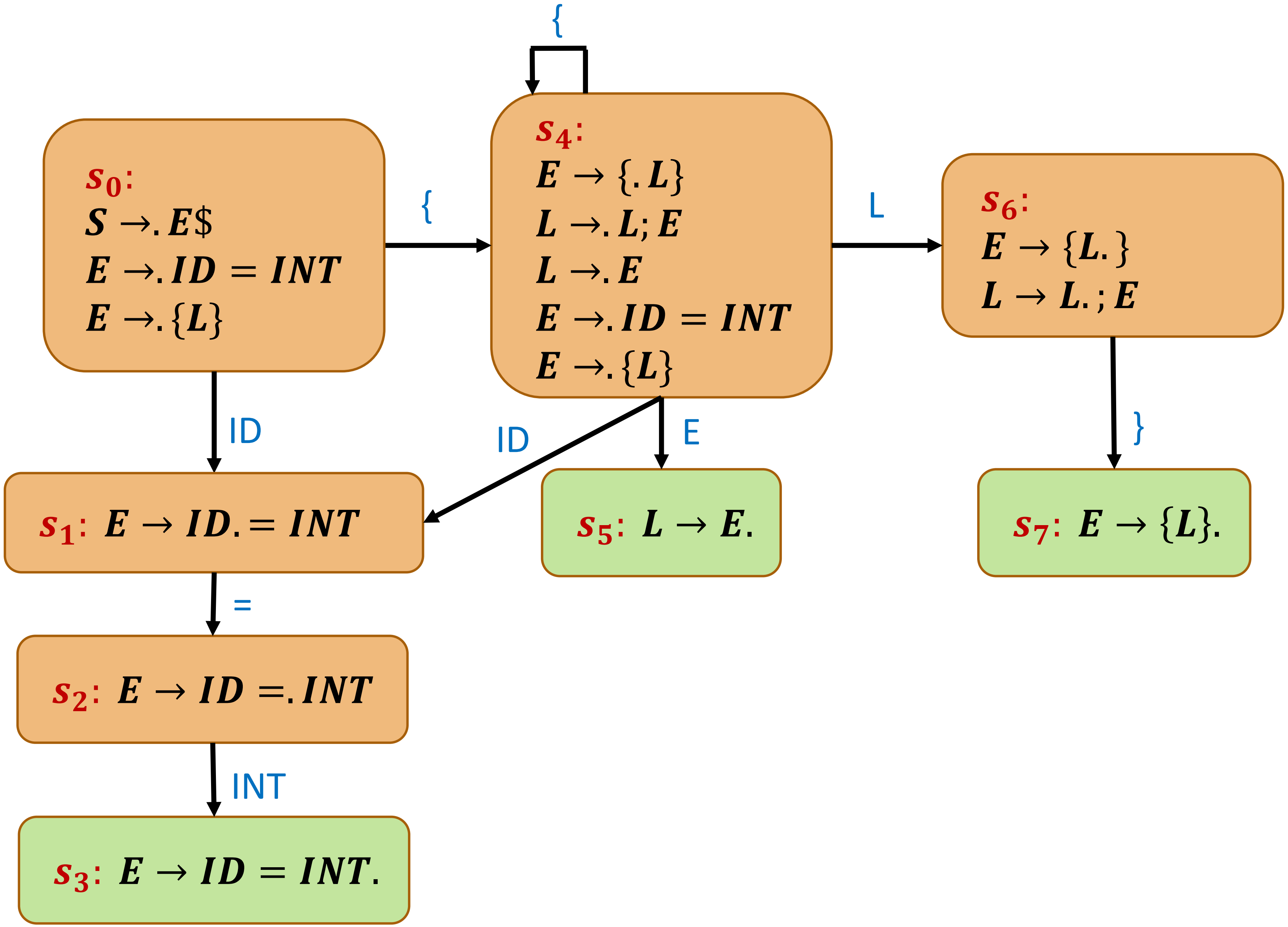
# LR(0) Parser

From $s_6$, if we recognized }, then the next state will contain:

- $E \rightarrow \{L\}.$

Which is a reduce state.

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$



$s_0$: $S \to .E\$$
$E \to .ID = INT$
$E \to .\{L\}$
{
$s_4$: $E \to \{.L\}$
$L \to .L; E$
$L \to .E$
$E \to .ID = INT$
$E \to .\{L\}$
{
L
$s_6$: $E \to \{L.\}$
$L \to L.; E$
ID
ID
E
}
$s_1$: $E \to ID. = INT$
$s_5$: $L \to E.$
$s_7$: $E \to \{L\}.$
=
$s_2$: $E \to ID = .INT$
INT
$s_3$: $E \to ID = INT.$

# LR(0) Parser

From $s_6$, if we recognized ;, then the next state will contain:

- $L \rightarrow L;.E$

So the next state (the $\epsilon$-closure) contains:

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$

# LR(0) Parser

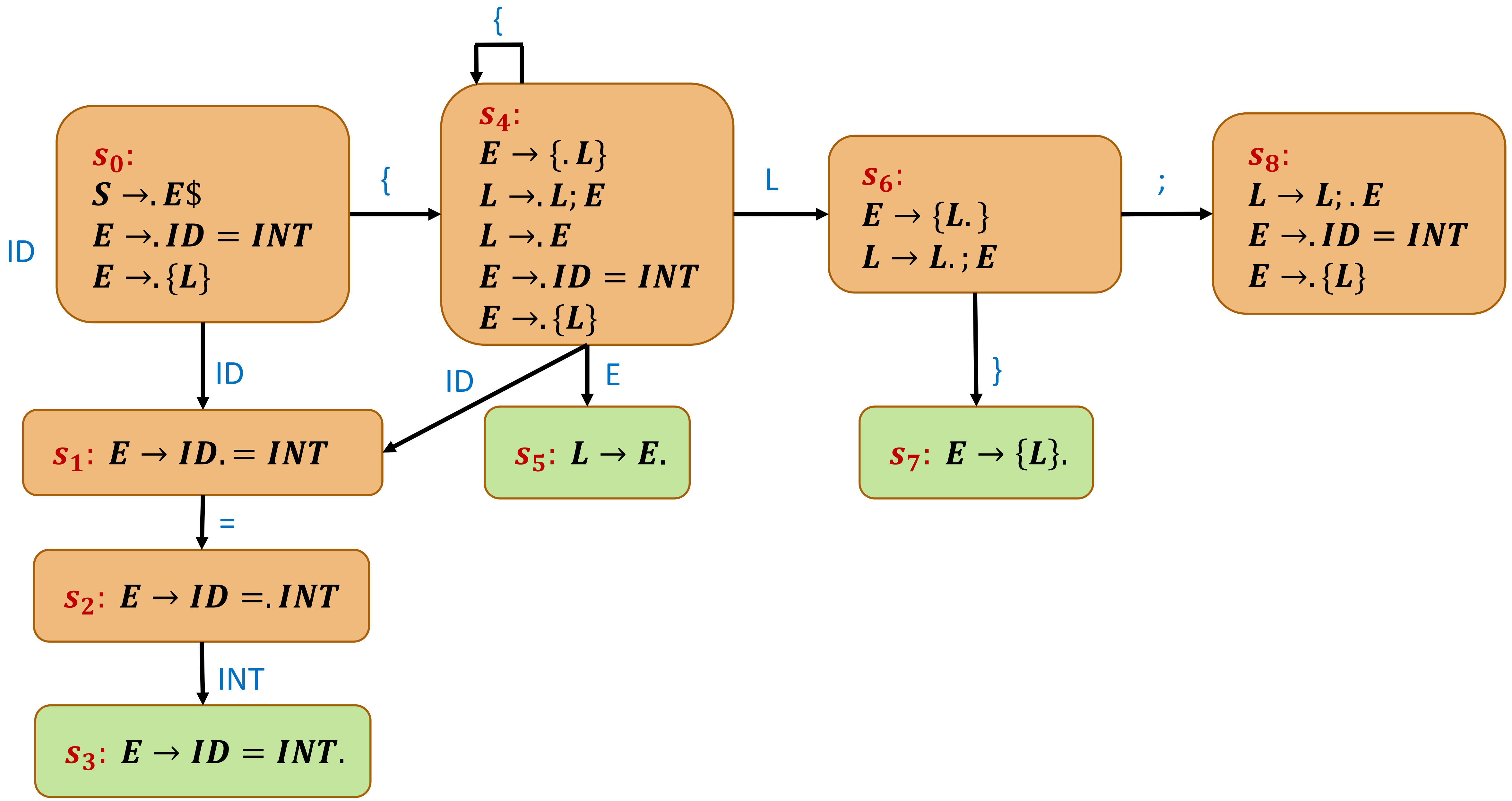
From $s_6$, if we recognized ;, then the next state will contain:

- $L \rightarrow L; .E$

So the next state (the $\epsilon$-closure) contains:

- $L \rightarrow L; .E$
- $E \rightarrow .ID = INT$
- $E \rightarrow .\{L\}$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$



$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
ID
{
$s_4$: $E \to \{.L\}$ $L \to .L;E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
{
L
$s_6$: $E \to \{L.\}$ $L \to L.;E$
;
$s_8$: $L \to L;.E$ $E \to .ID = INT$ $E \to .\{L\}$
ID
ID
E
}
$s_1$: $E \to ID. = INT$
$s_5$: $L \to E.$
$s_7$: $E \to \{L\}.$
=
$s_2$: $E \to ID =.INT$
INT
$s_3$: $E \to ID = INT.$

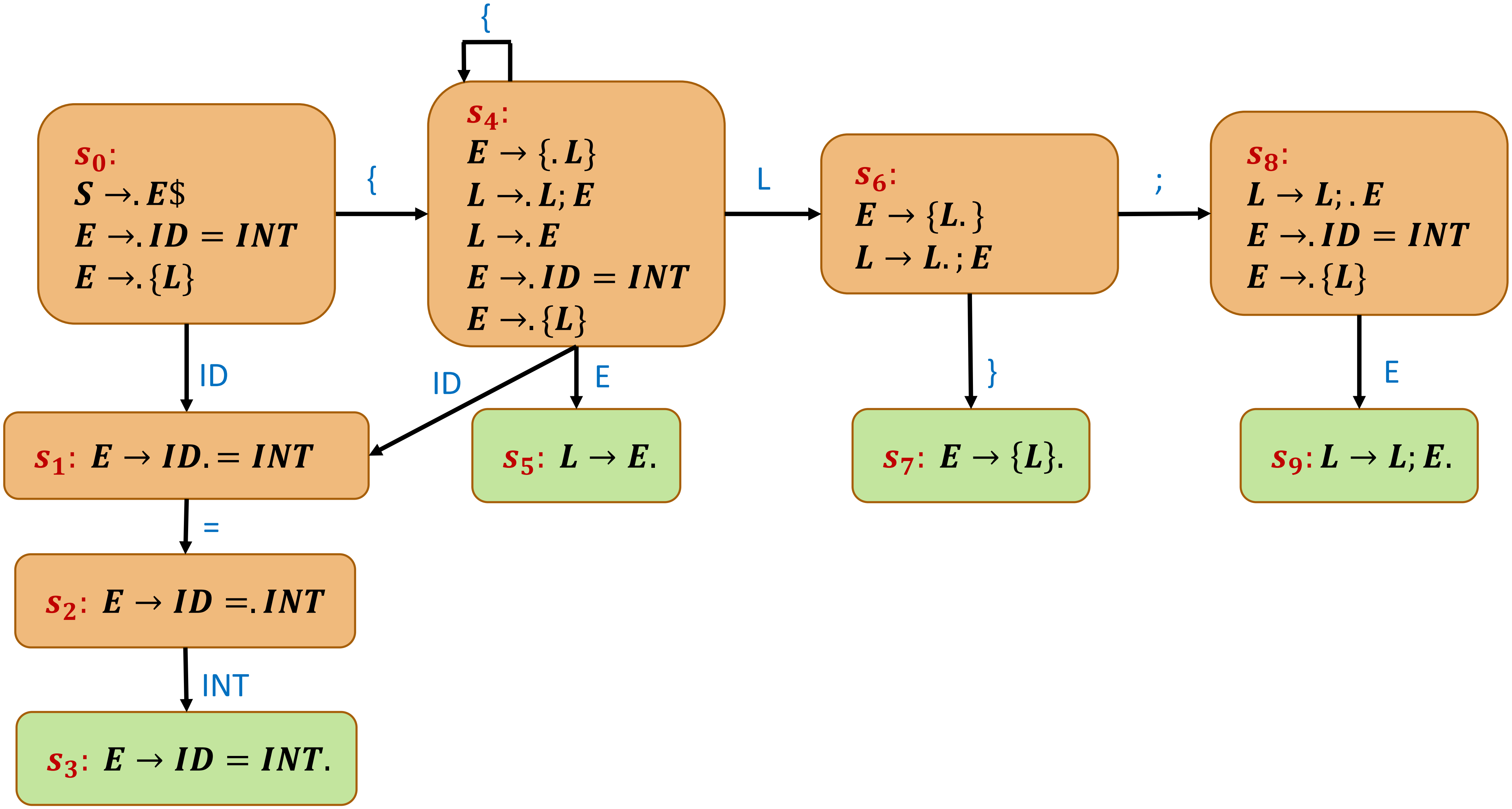
# LR(0) Parser

From $s_8$, if we recognized $E$, then the next state will contain:

- $E \rightarrow L; E.$

which is a reduce state.

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$



$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =. INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L; E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.; E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L;.E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L; E.$
{
{
ID
ID
E
=
INT
L
}
;
E

# LR(0) Parser

From $s_8$, if we recognized $\{$, then the next state will contain:

- $E \rightarrow \{.L\}$

So the next state (the $\epsilon$-closure) contains:

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$

# LR(0) Parser

From $s_8$, if we recognized {, then the next state will contain:

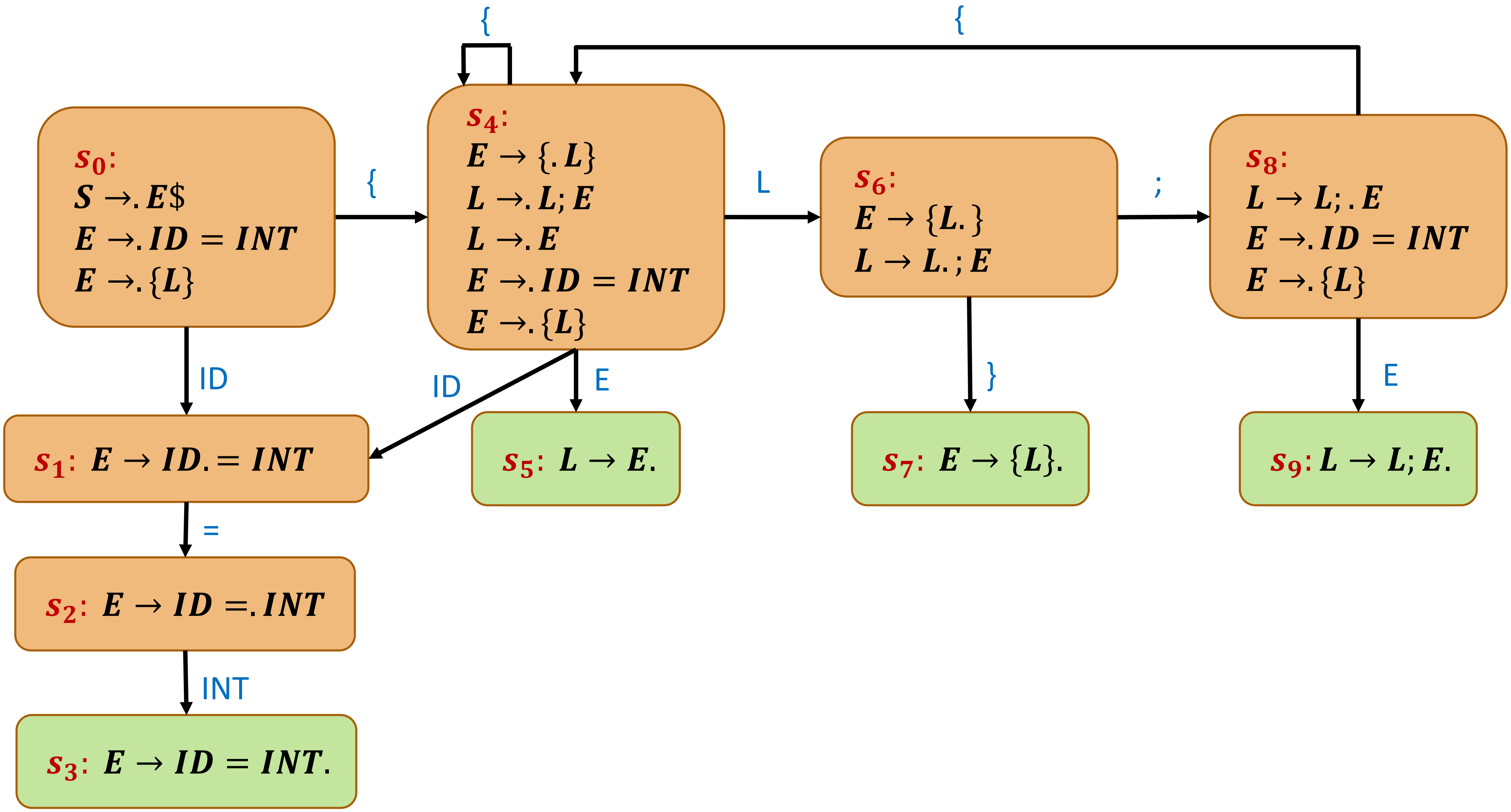
- $E \rightarrow \{.L\}$

So the next state (the $\epsilon$-closure) contains:

- $E \rightarrow \{.L\}$
- $L \rightarrow .L; E$
- $L \rightarrow .E$
- $E \rightarrow .ID = INT$
- $E \rightarrow .\{L\}$

which was already computed: $s_4$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$



$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =. INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L; E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.; E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L; .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L; E.$
{
{
{
L
;
ID
ID
E
}
E
=
INT

# LR(0) Parser

From $s_8$, if we recognized $ID$, then the next state will contain:

- $E \rightarrow ID. = INT$

So the next state (the $\epsilon$-closure) contains:

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$

# LR(0) Parser

From $s_8$, if we recognized $ID$, then the next state will contain:

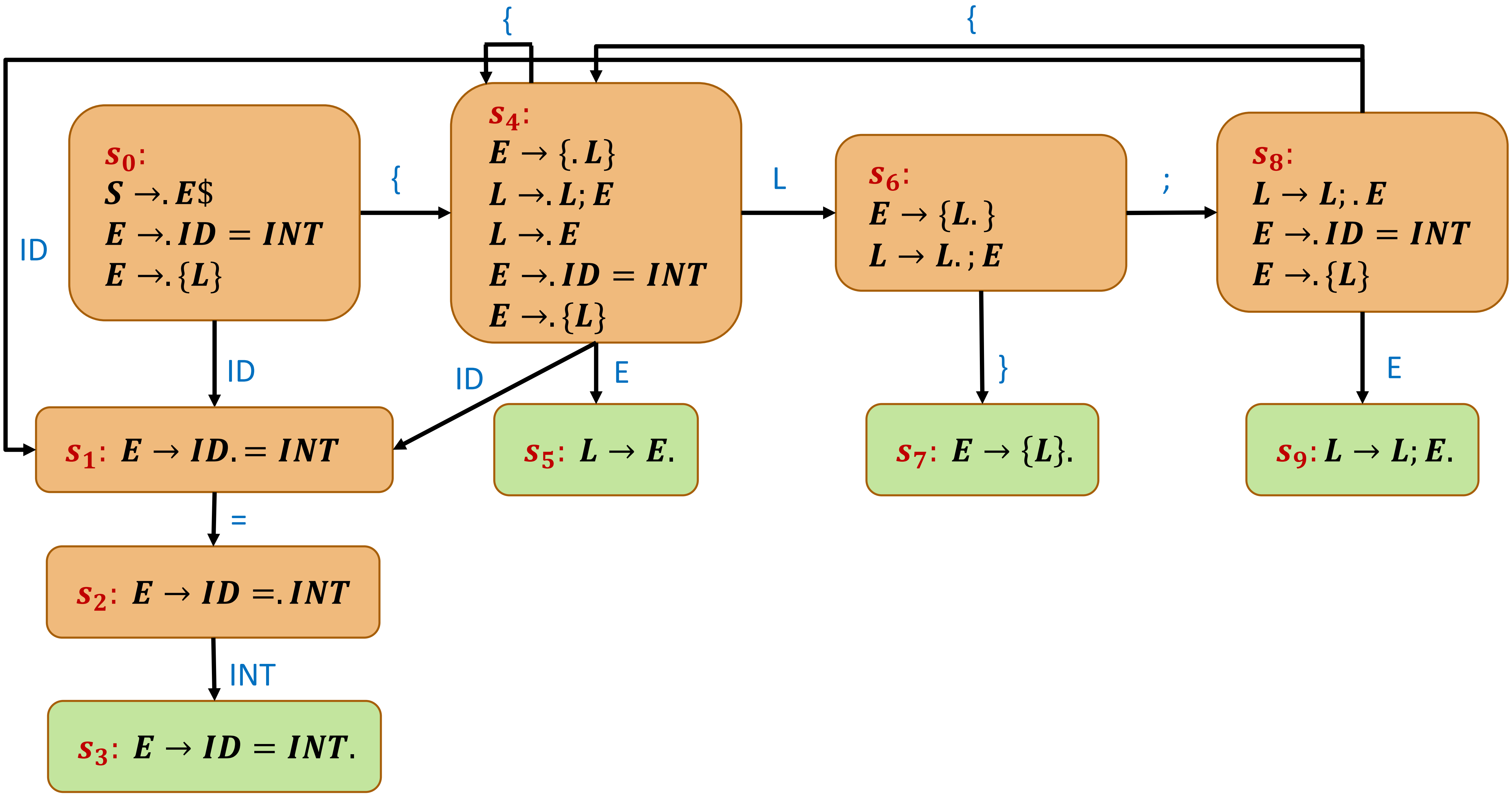
- $E \rightarrow ID. = INT$

So the next state (the $\epsilon$-closure) contains:

- $E \rightarrow ID. = INT$

which was already computed: $s_1$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$



$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID = .INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L;E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.;E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L;.E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L;E.$
{
{
{
L
;
ID
ID
ID
E
}
E
=
INT

# LR(0) Parser

From $s_0$, if we recognized $E$, then the next state will contain:

- $S \to E.\$$

So the next state (the $\epsilon$-closure) contains:

$S \to E\$$
$E \to ID = INT$
$E \to \{L\}$
$L \to E$
$L \to L; E$

# LR(0) Parser

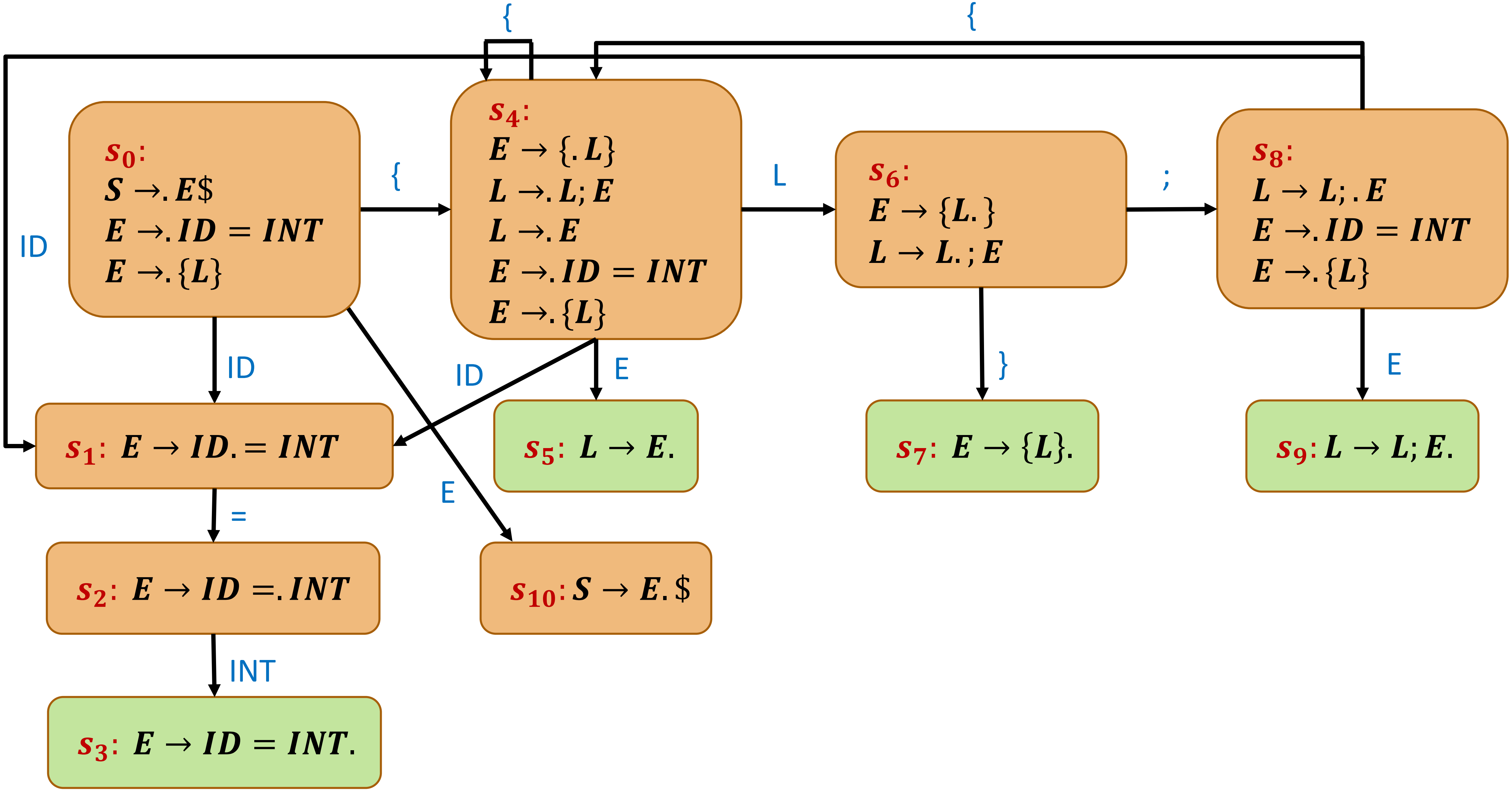
From $s_0$, if we recognized $E$, then the next state will contain:

- $S \rightarrow E.\$$

So the next state (the $\epsilon$-closure) contains:

- $S \rightarrow E.\$$

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$



$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =.INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L; E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.; E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L; .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L; E.$
$s_{10}$: $S \to E.\$$
{
{
{
L
;
ID
ID
ID
E
E
E
}
=
INT

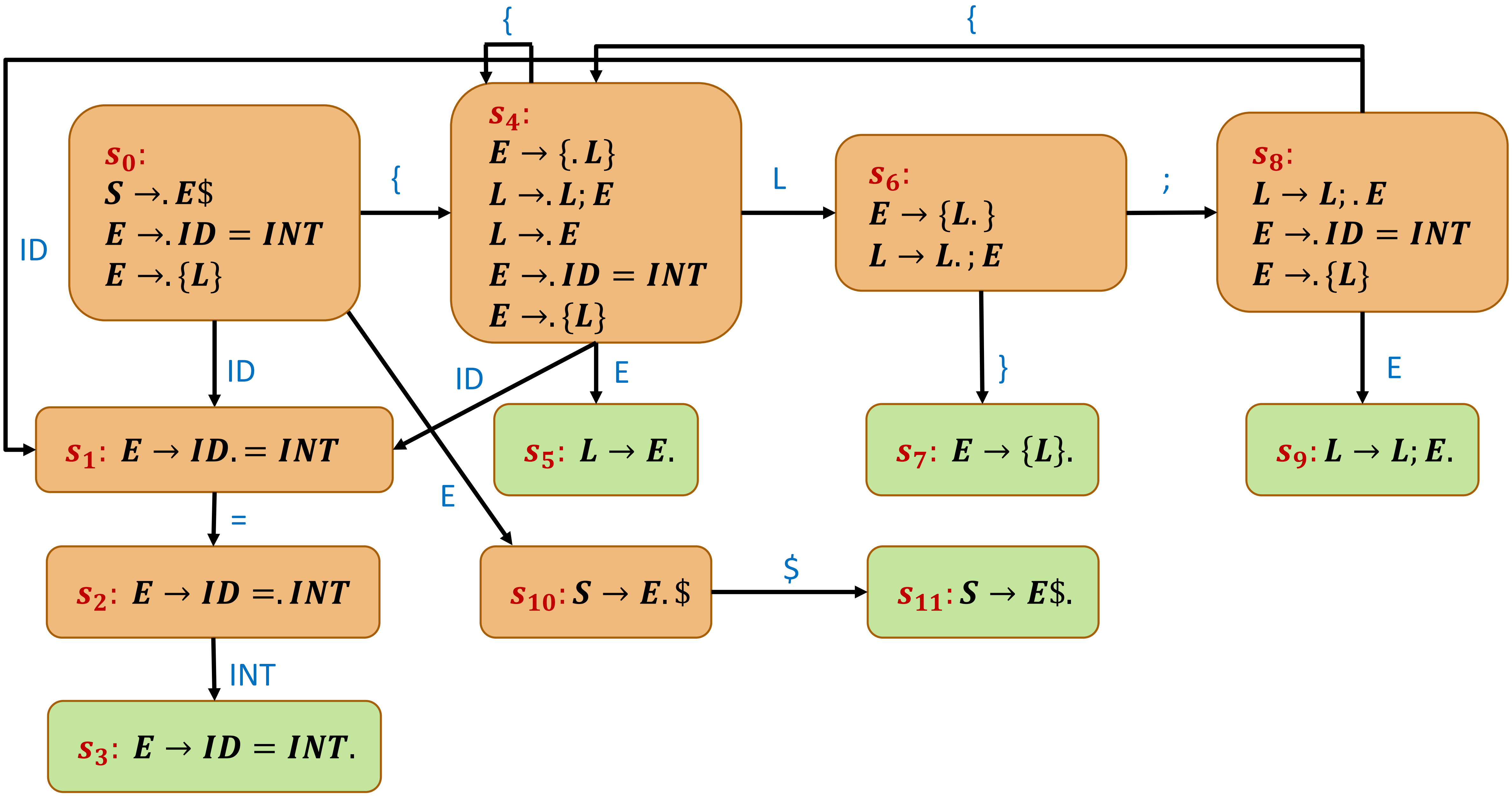
# LR(0) Parser

From $s_{10}$, if we recognized $E$, then the next state will contain:

- $S \rightarrow E\$.$

which is a reduce state.

$S \rightarrow E\$$
$E \rightarrow ID = INT$
$E \rightarrow \{L\}$
$L \rightarrow E$
$L \rightarrow L; E$

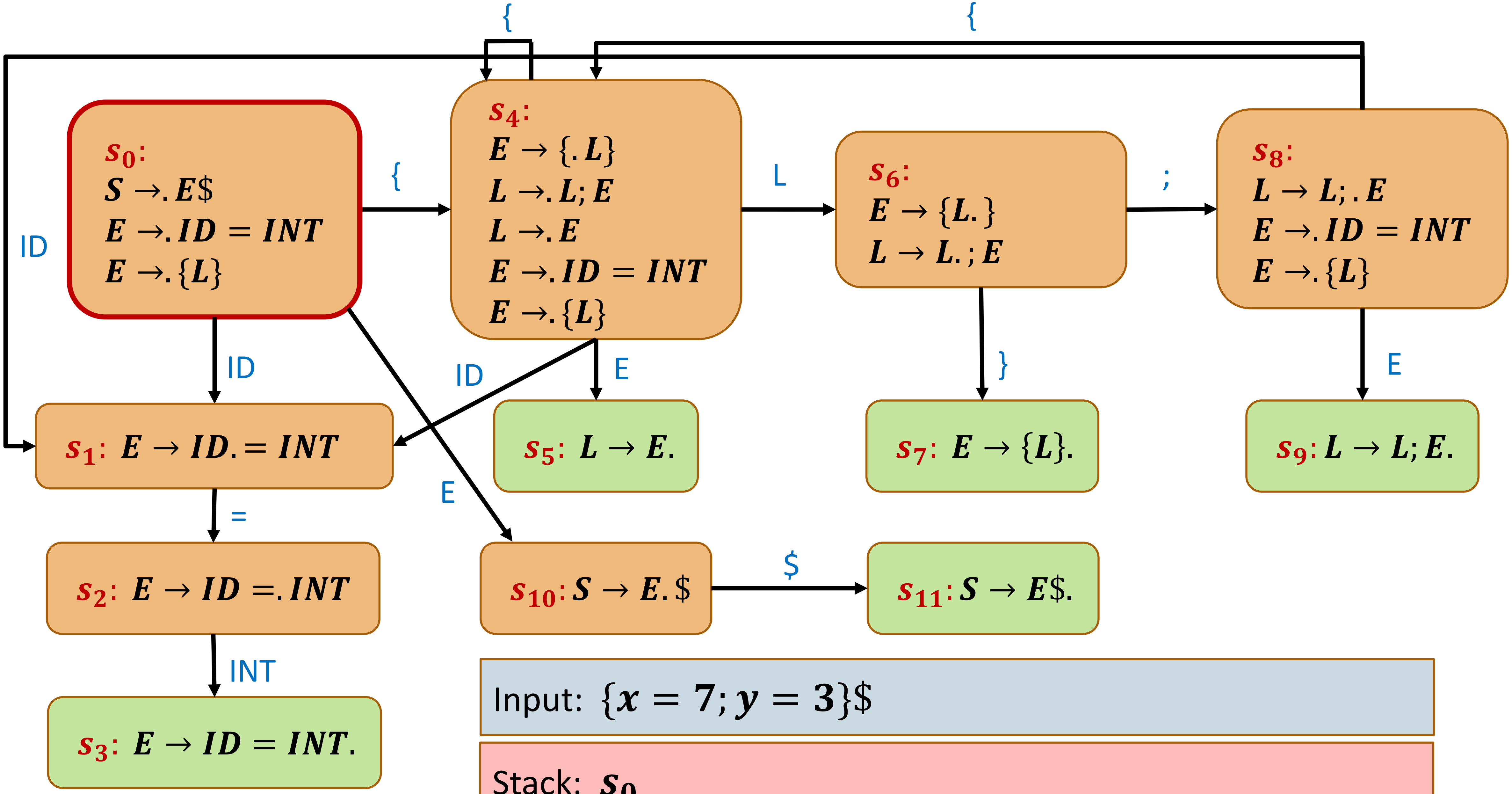


$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =. INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L; E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.; E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L;.E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L; E.$
$s_{10}$: $S \to E.\$$
$s_{11}$: $S \to E\$.$
{
ID
L
;
}
E
=
INT
$

# LR(0) Parser: Running Example

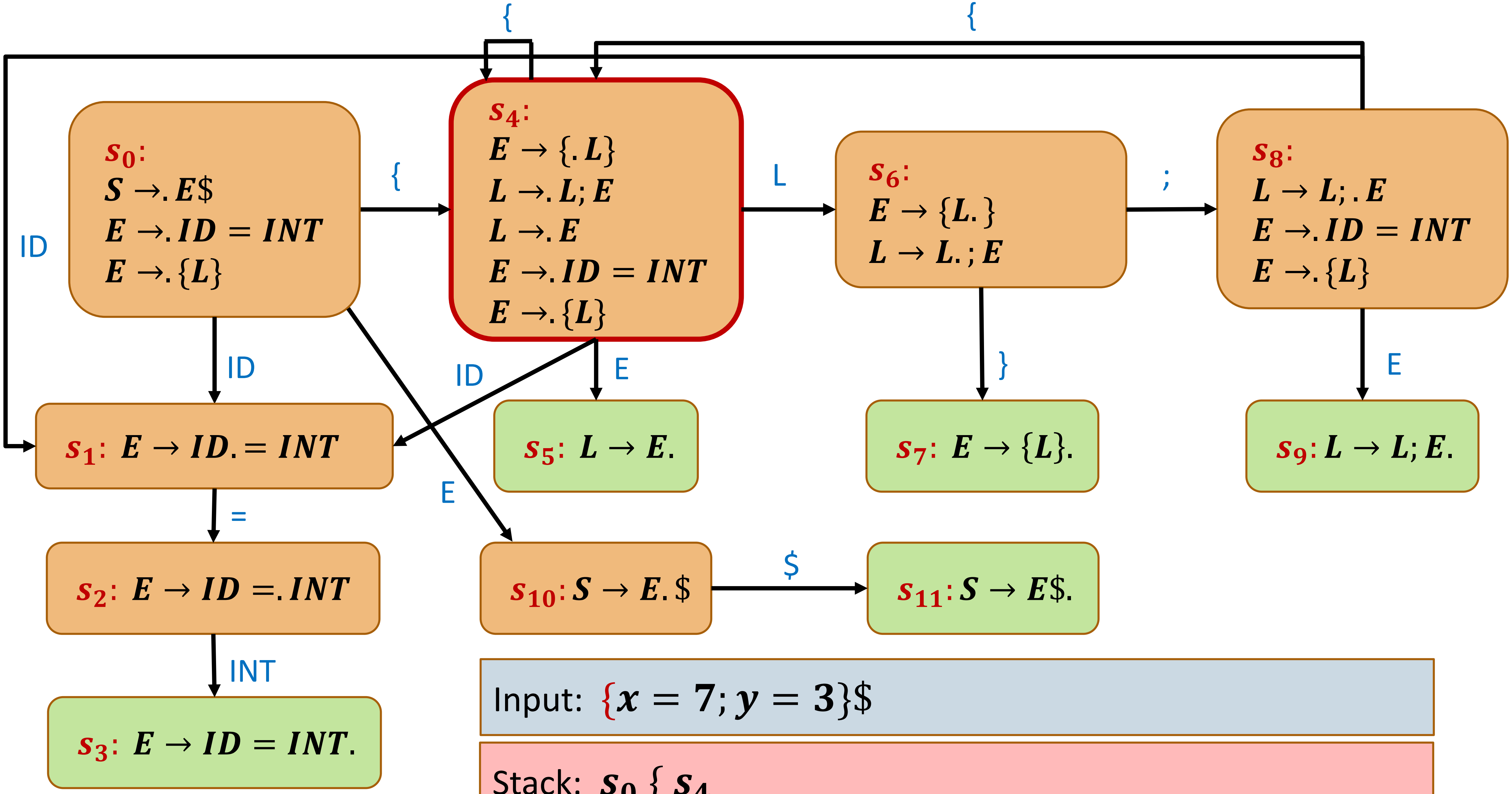
What will happen with the following input:

- $\{\mathrm{x} = 7; y = 3\}\$$



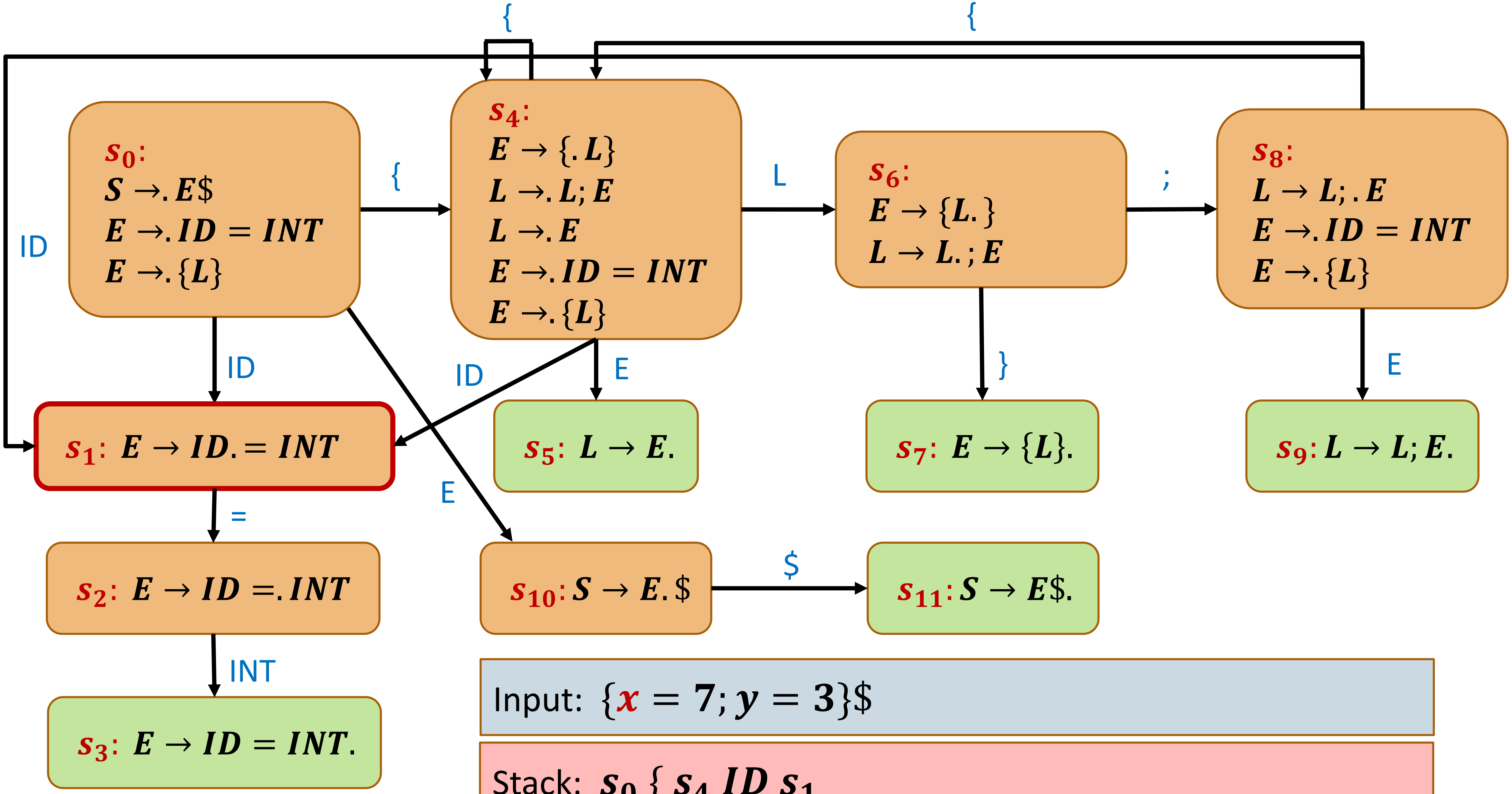

Input: $\{x = 7; y = 3\}\$$

Stack: $s_0$



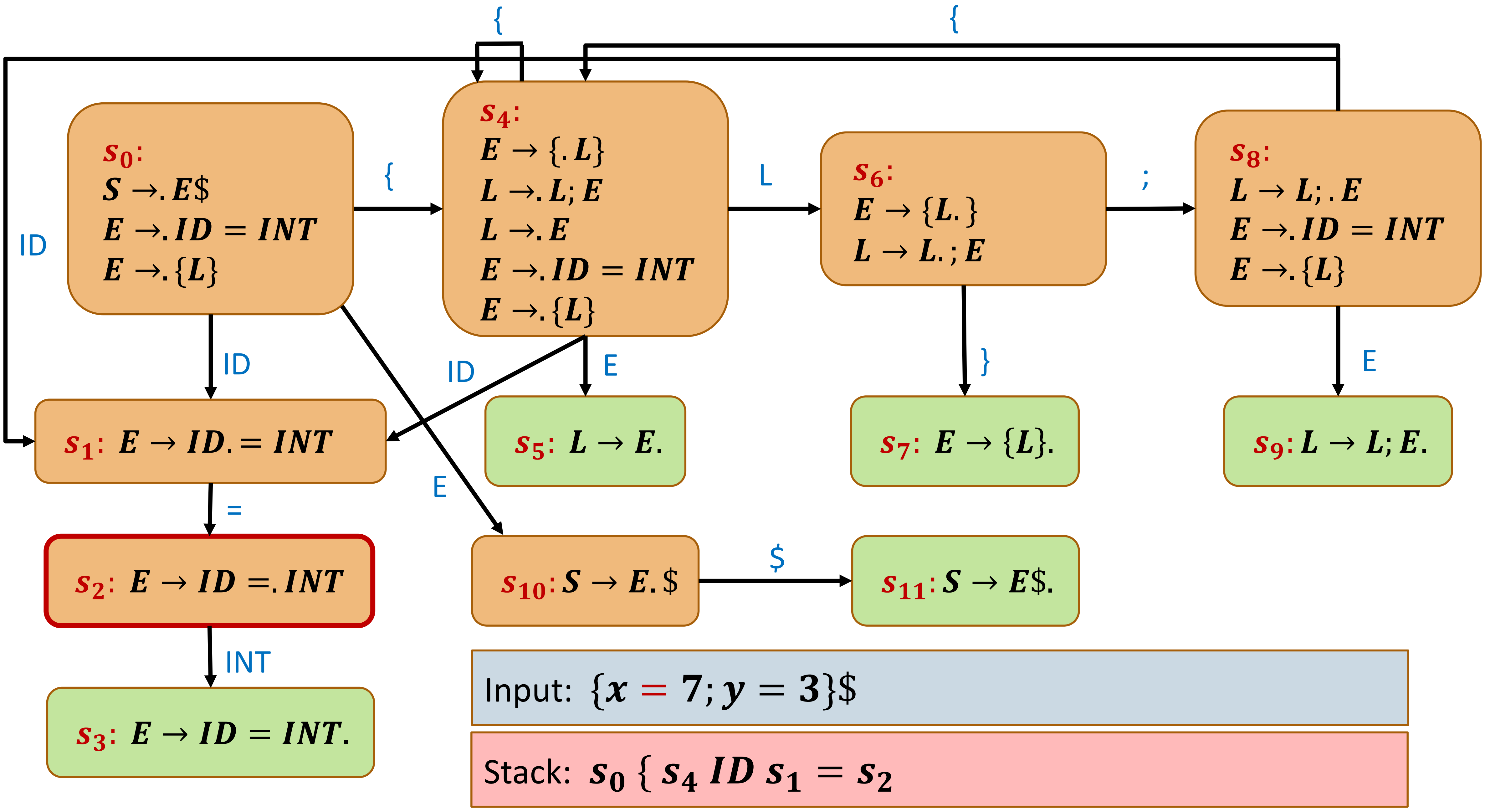

Input: $\{x = 7; y = 3\}\$$

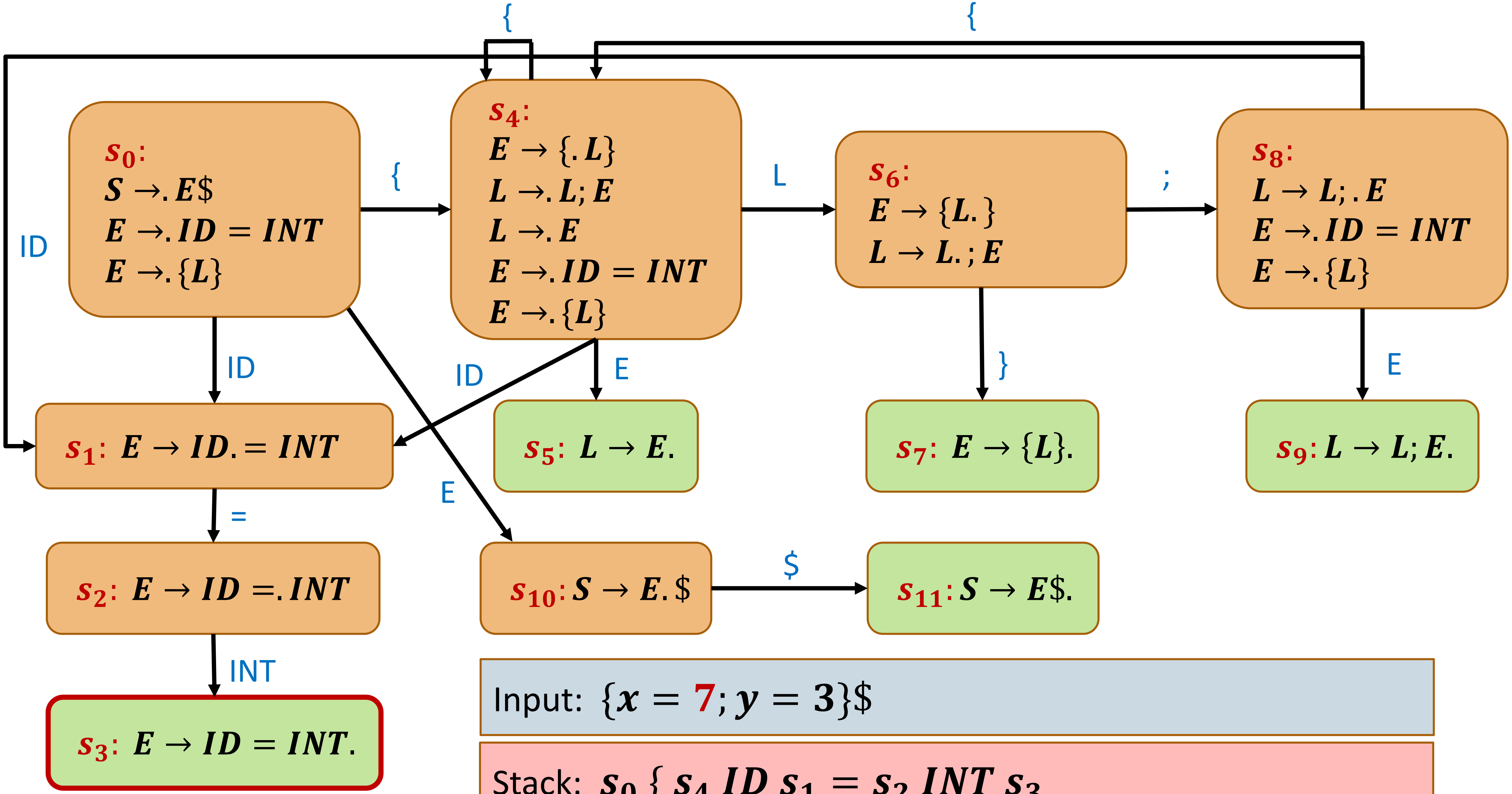
Stack: $s_0 \; \{ \; s_4$




Input: $\{x = 7; y = 3\}\$$

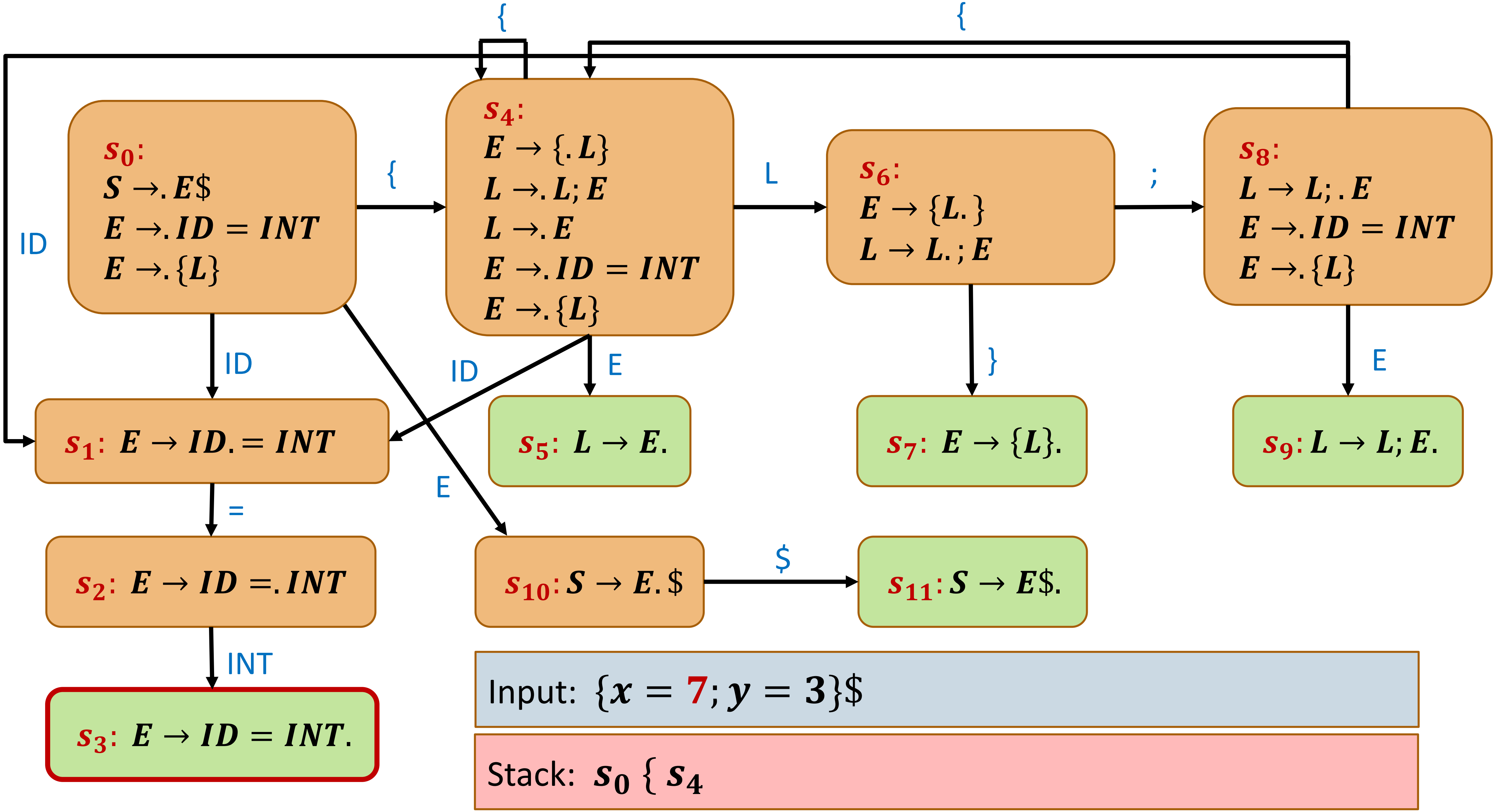
Stack: $s_0 \; \{ \; s_4 \; ID \; s_1$


$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =. INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L; E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.; E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L;.E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L; E.$
$s_{10}$: $S \to E.\$$
$s_{11}$: $S \to E\$.$
{
{
{
L
;
ID
ID
ID
E
}
E
=
E
INT
$
Input: $\{x = 7; y = 3\}\$$
Stack: $s_0 \, \{ \, s_4 \, ID \, s_1 = s_2$

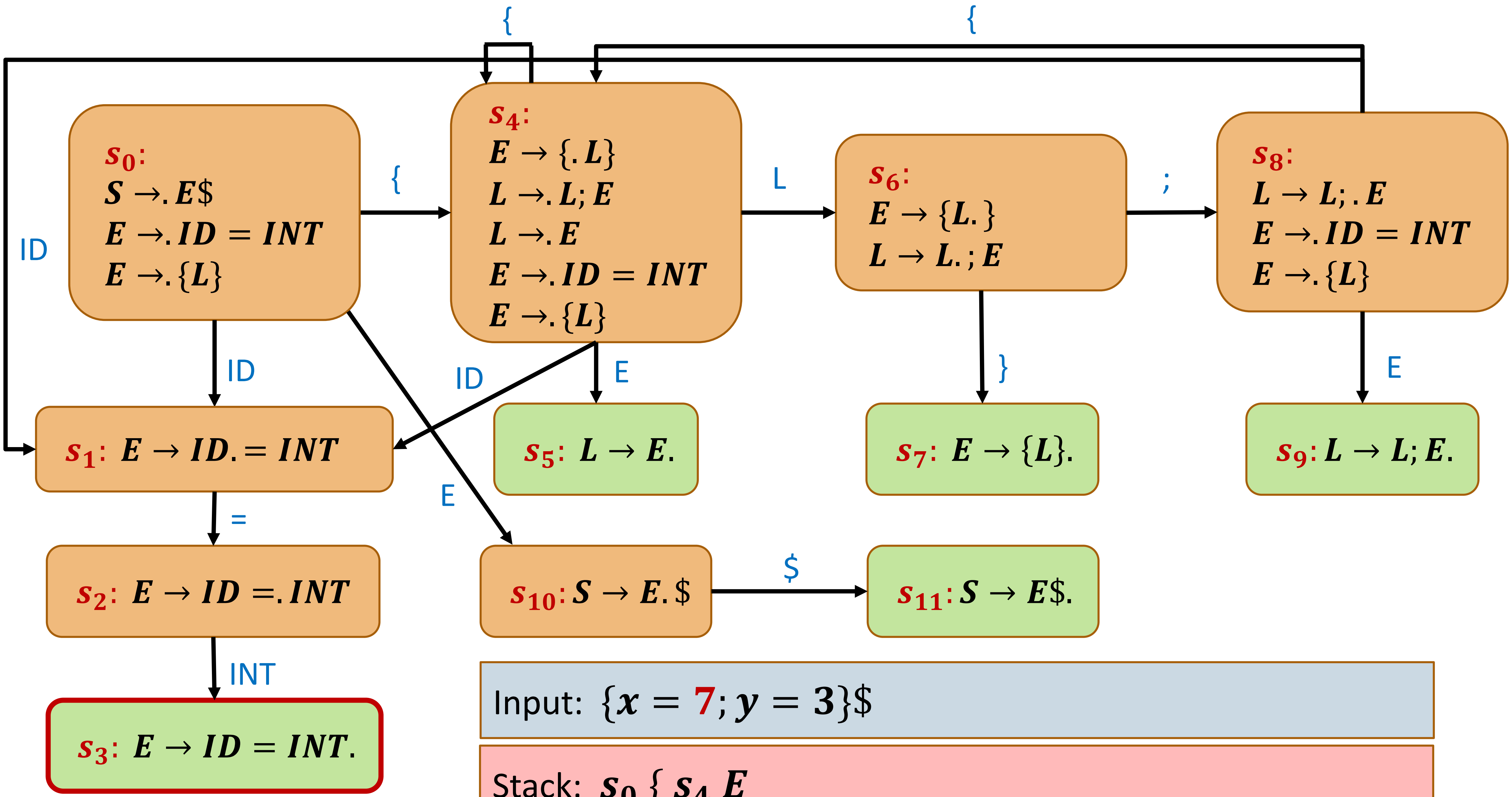



Input: $\{x = 7; y = 3\}\$$

Stack: $s_0 \,\{\, s_4 \; ID \; s_1 = s_2 \; INT \; s_3$

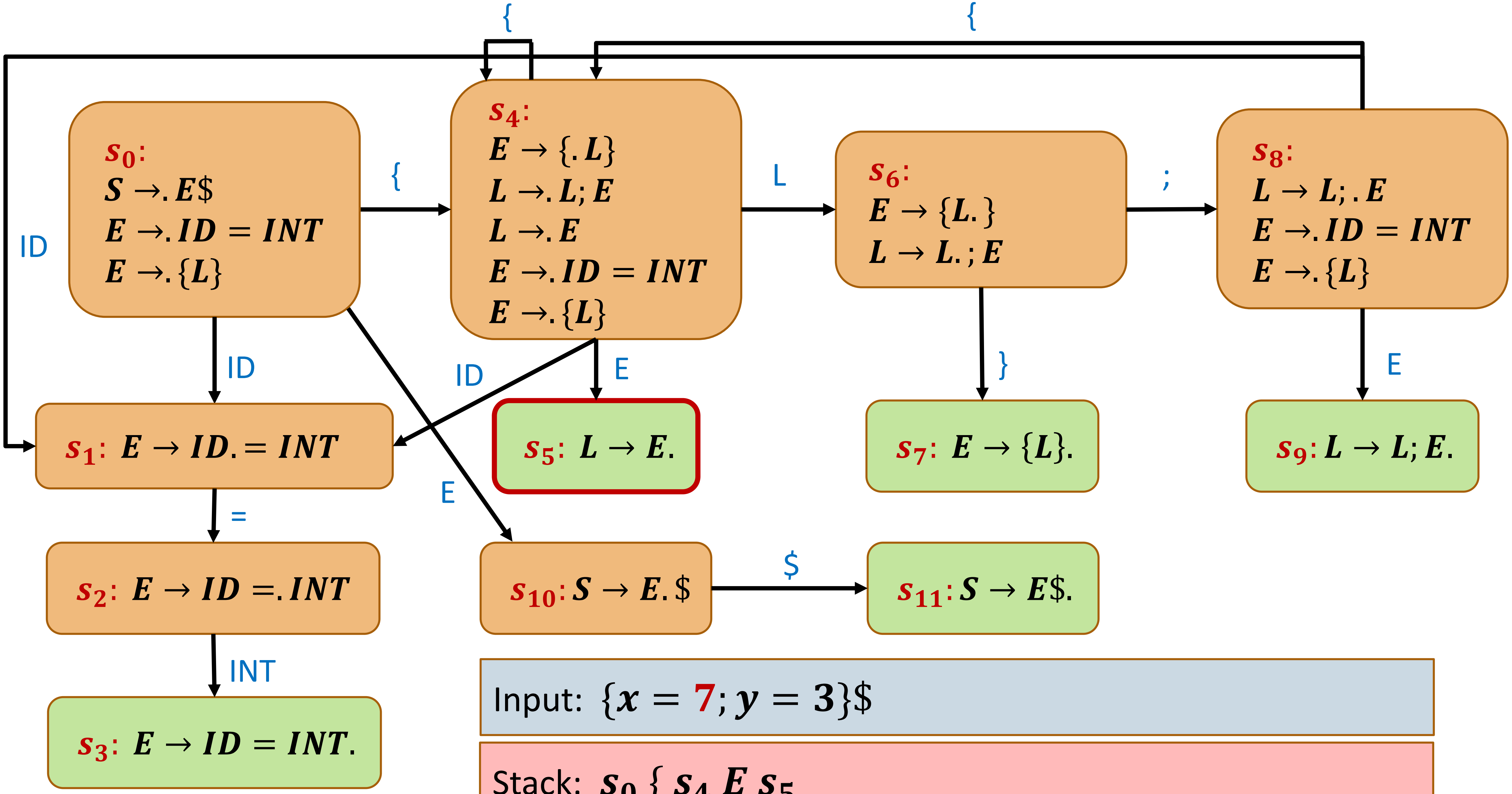


$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =. INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L;E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.;E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L;.E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L;E.$
$s_{10}$: $S \to E.\$$
$s_{11}$: $S \to E\$.$
{
{
{
L
;
ID
ID
ID
E
E
E
}
=
INT
$
Input: $\{x = 7; y = 3\}\$$
Stack: $s_0 \{ s_4$



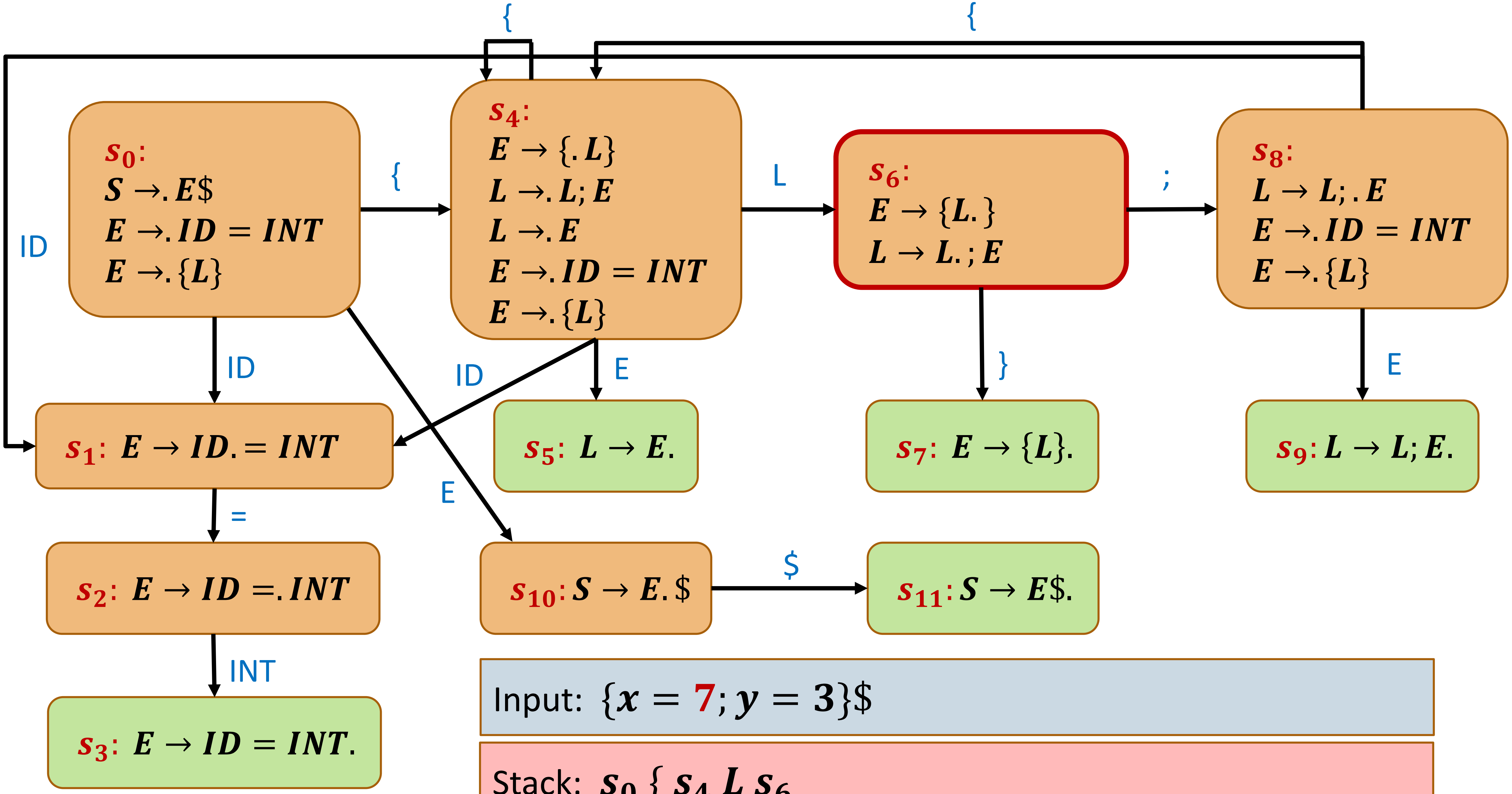

Input: $\{x = 7; y = 3\}\$$

Stack: $s_0 \; \{ \; s_4 \; E$



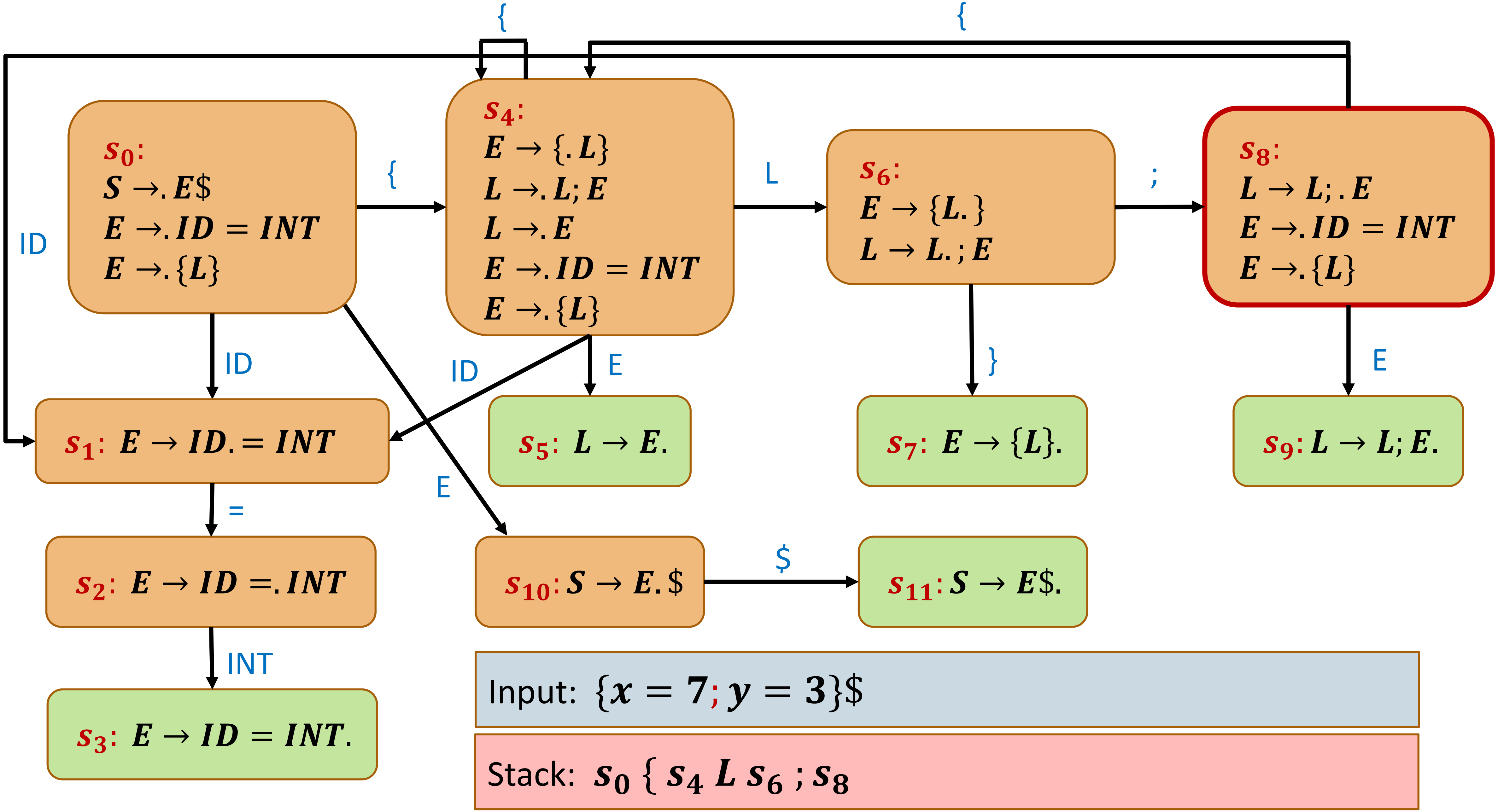

Input: $\{x = 7; y = 3\}\$$

Stack: $s_0 \, \{ \, s_4 \, E \, s_5$

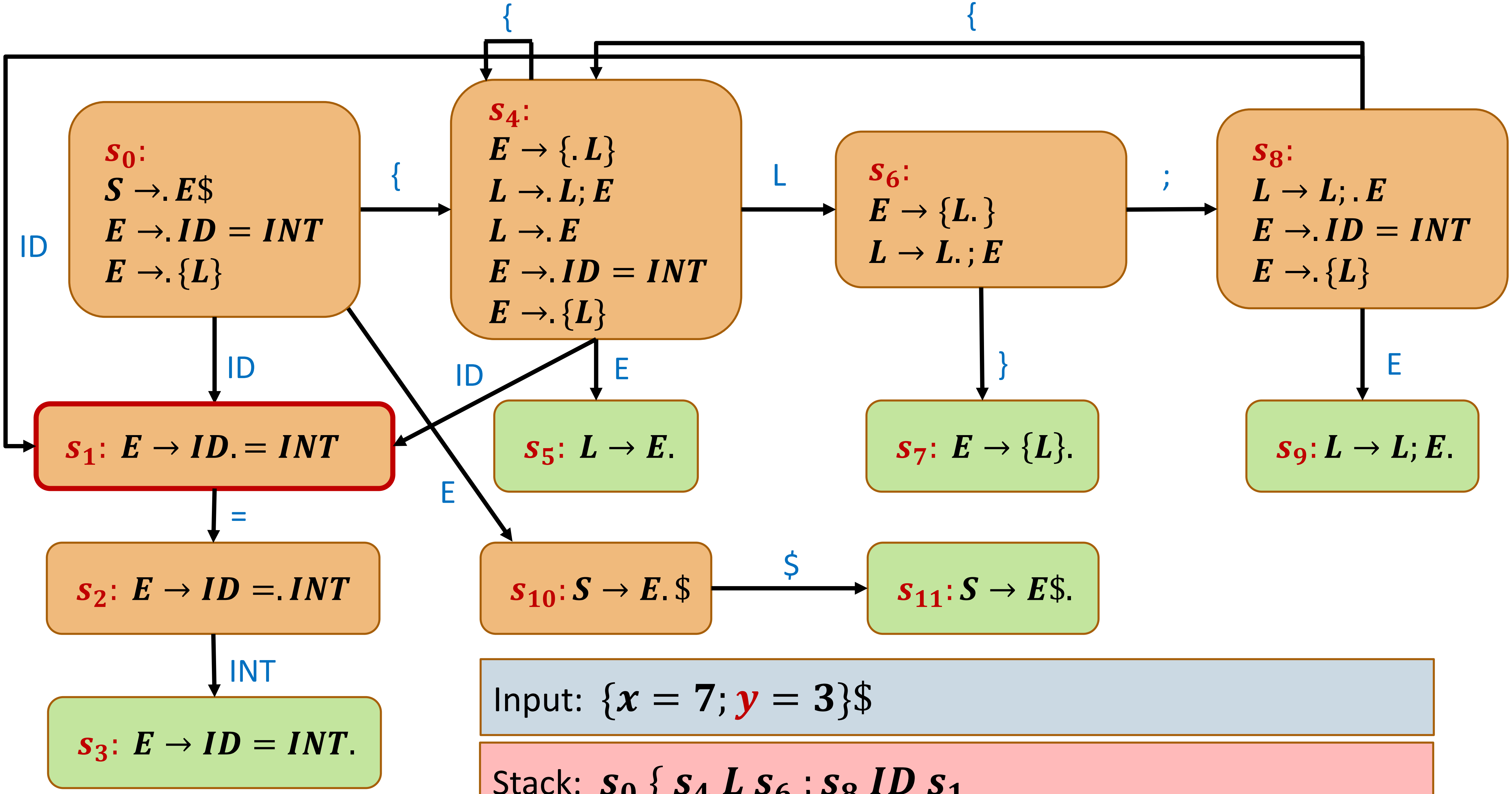



Input: $\{x = 7; y = 3\}\$$

Stack: $s_0 \; \{ \; s_4 \; L \; s_6$

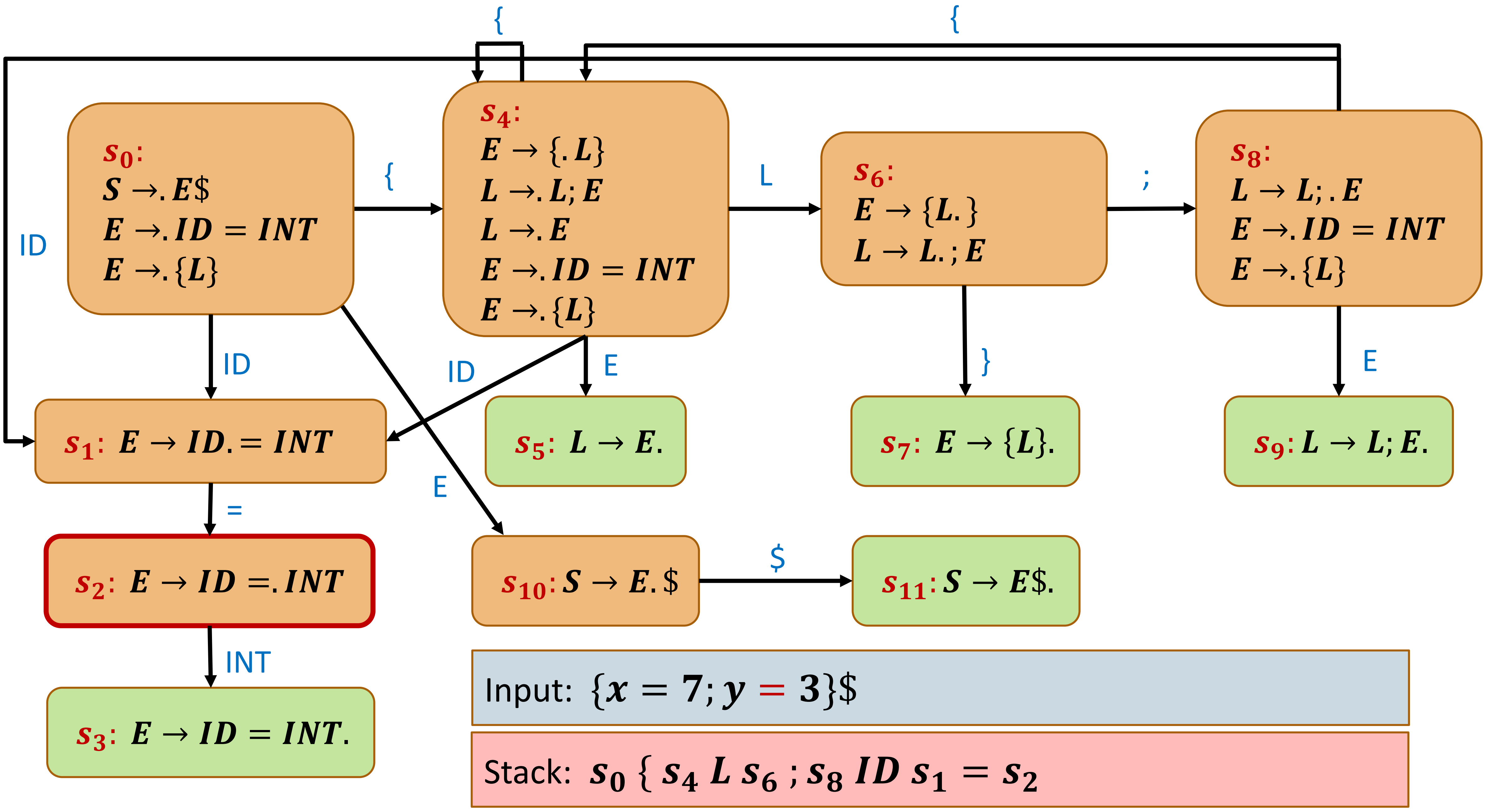


$s_0$: $S \rightarrow .E\$$ $E \rightarrow .ID = INT$ $E \rightarrow .\{L\}$
$s_1$: $E \rightarrow ID. = INT$
$s_2$: $E \rightarrow ID =. INT$
$s_3$: $E \rightarrow ID = INT.$
$s_4$: $E \rightarrow \{.L\}$ $L \rightarrow .L; E$ $L \rightarrow .E$ $E \rightarrow .ID = INT$ $E \rightarrow .\{L\}$
$s_5$: $L \rightarrow E.$
$s_6$: $E \rightarrow \{L.\}$ $L \rightarrow L.; E$
$s_7$: $E \rightarrow \{L\}.$
$s_8$: $L \rightarrow L; .E$ $E \rightarrow .ID = INT$ $E \rightarrow .\{L\}$
$s_9$: $L \rightarrow L; E.$
$s_{10}$: $S \rightarrow E.\$$
$s_{11}$: $S \rightarrow E\$.$
{
{
{
L
;
ID
ID
ID
E
E
}
E
=
INT
$
Input: $\{x = 7; y = 3\}\$$
Stack: $s_0 \{ s_4 \, L \, s_6 \, ; s_8$

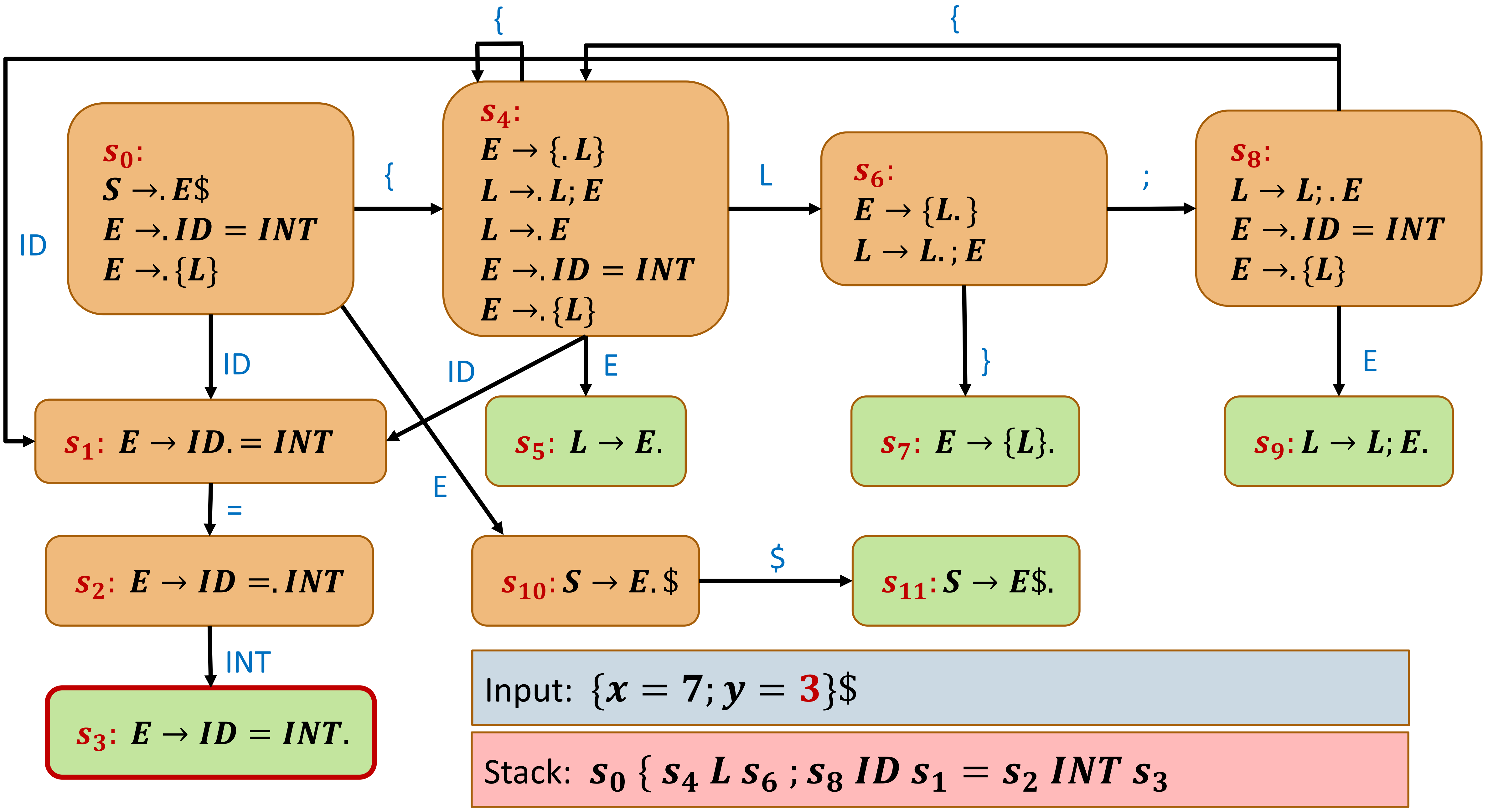



Input: $\{x = 7; y = 3\}\$$

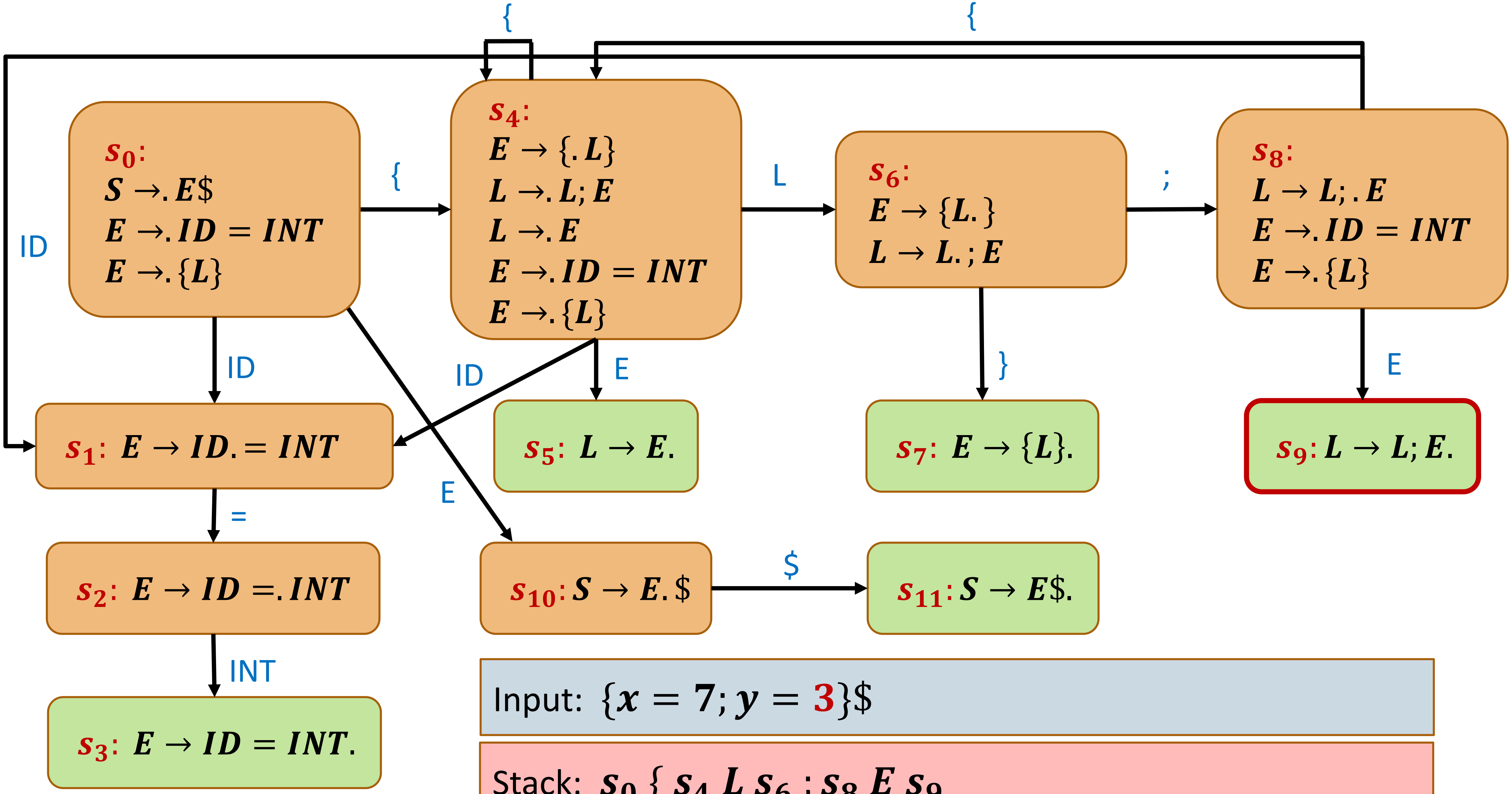
Stack: $s_0 \; \{ \; s_4 \; L \; s_6 \; ; \; s_8 \; ID \; s_1$



$s_0$: $S \rightarrow .E\$$ $E \rightarrow .ID = INT$ $E \rightarrow .\{L\}$
$s_1$: $E \rightarrow ID. = INT$
$s_2$: $E \rightarrow ID =. INT$
$s_3$: $E \rightarrow ID = INT.$
$s_4$: $E \rightarrow \{.L\}$ $L \rightarrow .L; E$ $L \rightarrow .E$ $E \rightarrow .ID = INT$ $E \rightarrow .\{L\}$
$s_5$: $L \rightarrow E.$
$s_6$: $E \rightarrow \{L.\}$ $L \rightarrow L.; E$
$s_7$: $E \rightarrow \{L\}.$
$s_8$: $L \rightarrow L; .E$ $E \rightarrow .ID = INT$ $E \rightarrow .\{L\}$
$s_9$: $L \rightarrow L; E.$
$s_{10}$: $S \rightarrow E.\$$
$s_{11}$: $S \rightarrow E\$.$
{
{
{
L
;
ID
ID
ID
E
E
E
}
=
INT
$
Input: $\{x = 7; y = 3\}\$$
Stack: $s_0 \{ s_4 L s_6 ; s_8 ID s_1 = s_2$

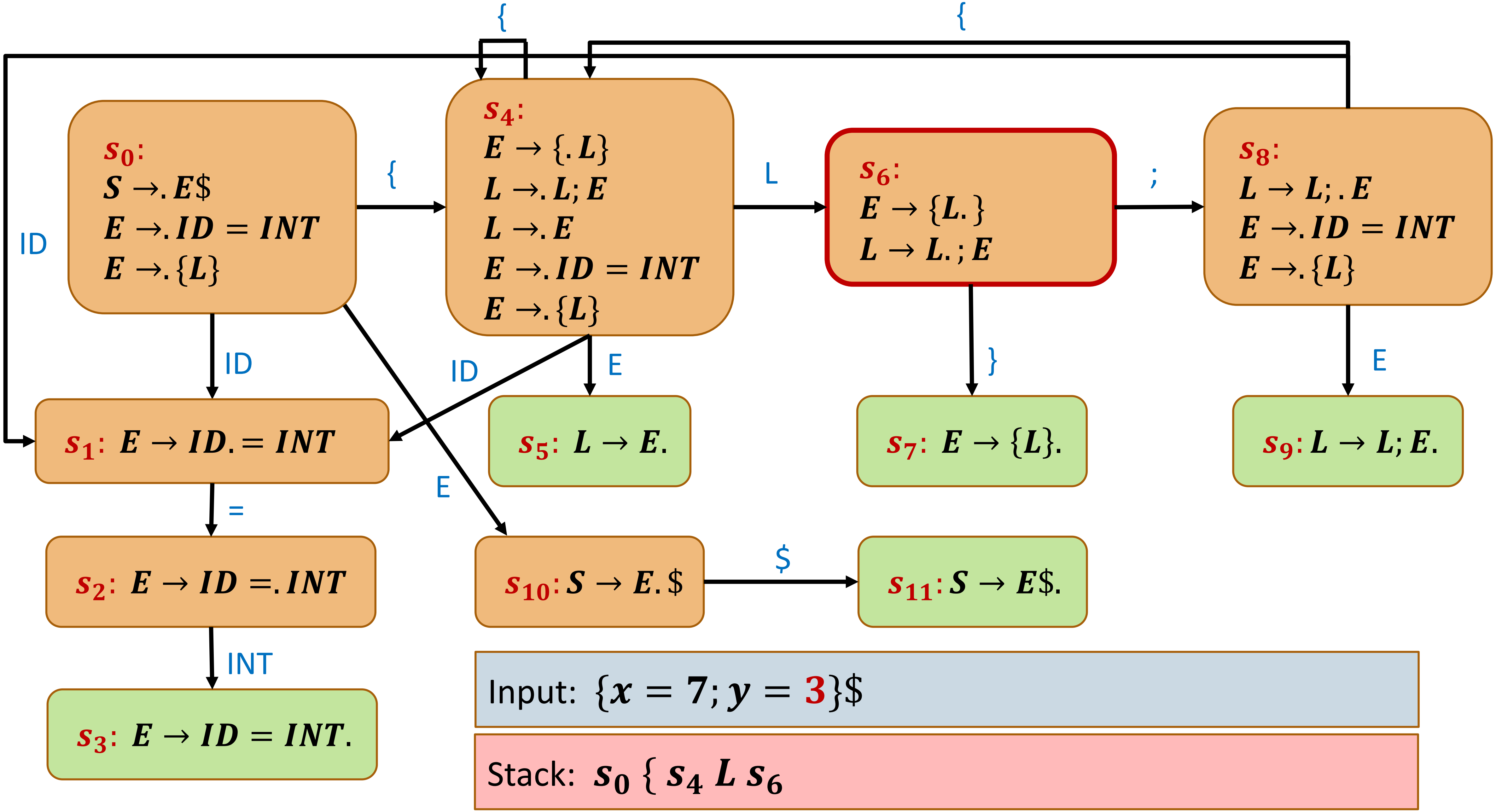


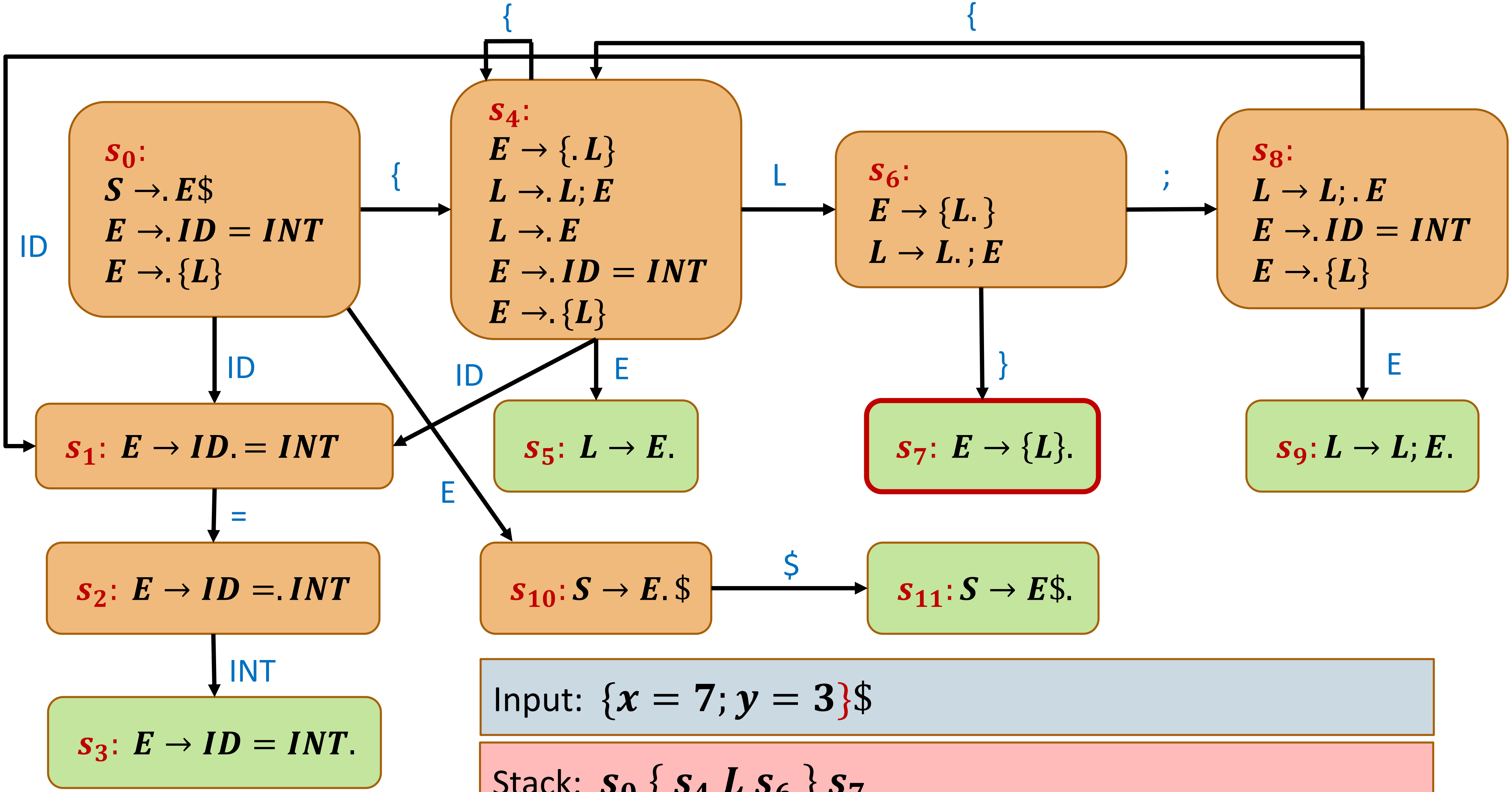
$s_0$: $S \to .E\$$ ; $E \to .ID = INT$ ; $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =. INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ ; $L \to .L;E$ ; $L \to .E$ ; $E \to .ID = INT$ ; $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ ; $L \to L.;E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L;.E$ ; $E \to .ID = INT$ ; $E \to .\{L\}$
$s_9$: $L \to L;E.$
$s_{10}$: $S \to E.\$$
$s_{11}$: $S \to E\$.$
{
L
;
ID
E
}
=
INT
$
Input: $\{x = 7; y = 3\}\$$
Stack: $s_0 \{ s_4\, L\, s_6 ; s_8\, ID\, s_1 = s_2\, INT\, s_3$




Input: $\{x = 7; y = 3\}\$$

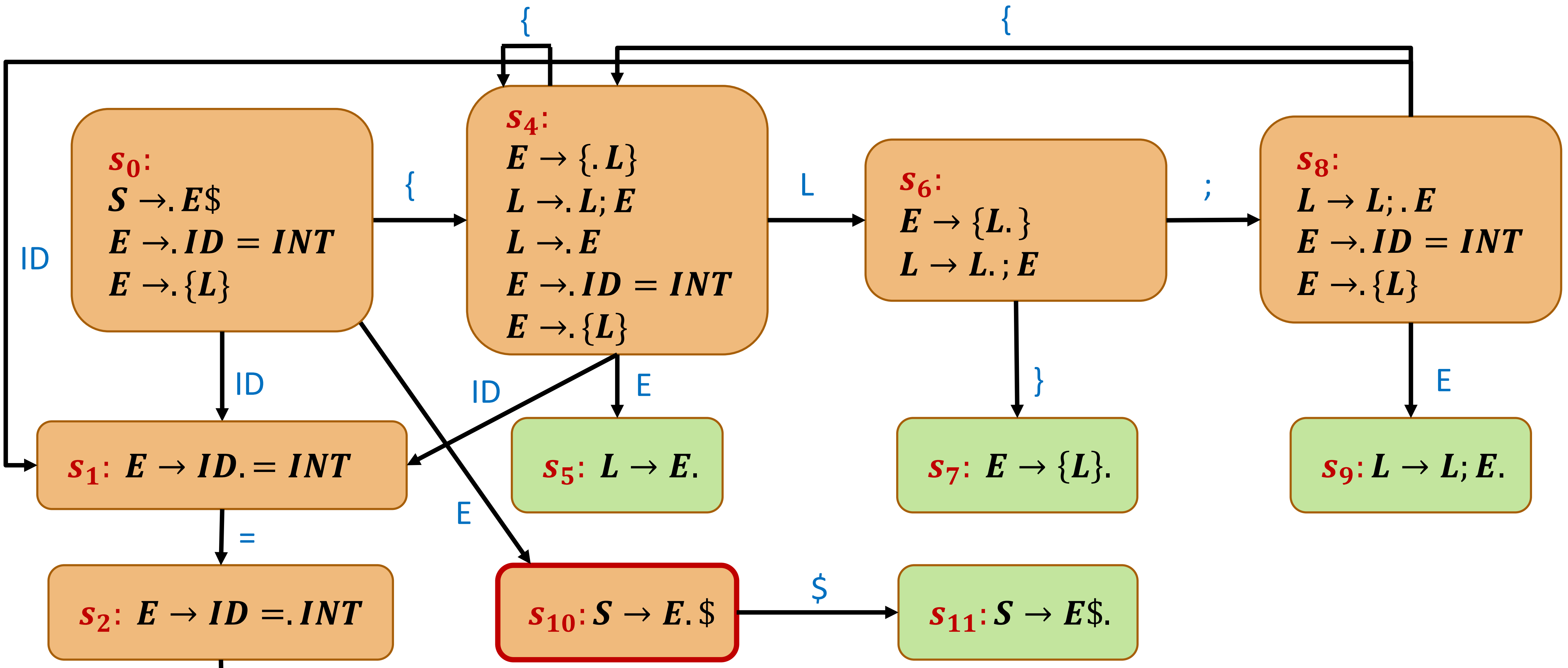
Stack: $s_0 \{ s_4\, L\, s_6 \,;\, s_8\, E\, s_9$


$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =.INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L;E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.;E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L;.E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L;E.$
$s_{10}$: $S \to E.\$$
$s_{11}$: $S \to E\$.$
{
{
{
L
;
ID
ID
ID
E
E
E
}
=
INT
$
Input: $\{x = 7; y = 3\}\$$
Stack: $s_0 \{ s_4 L s_6$



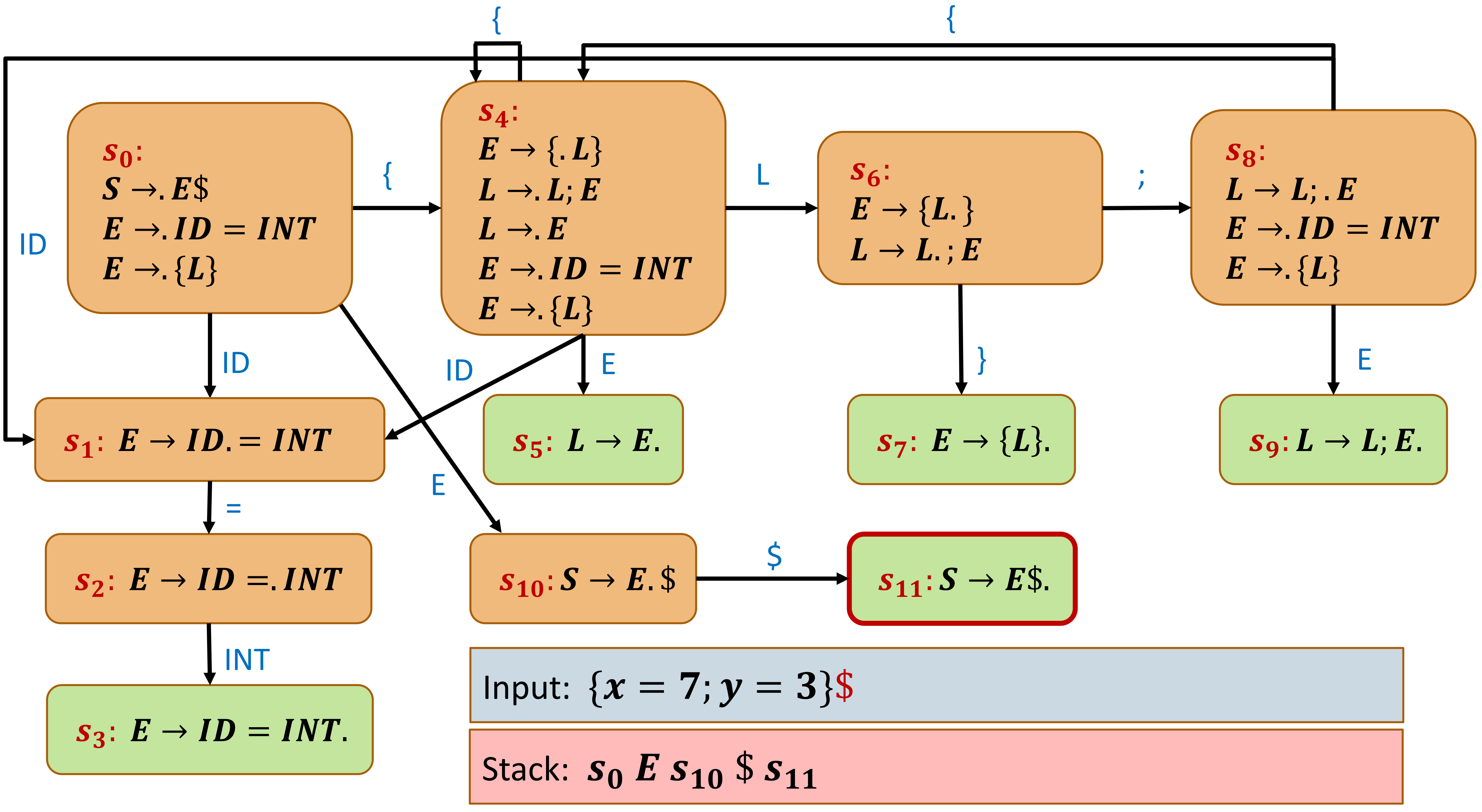

Input: $\{x = 7; y = 3\}\$$

Stack: $s_0 \; \{ \; s_4 \; L \; s_6 \; \} \; s_7$

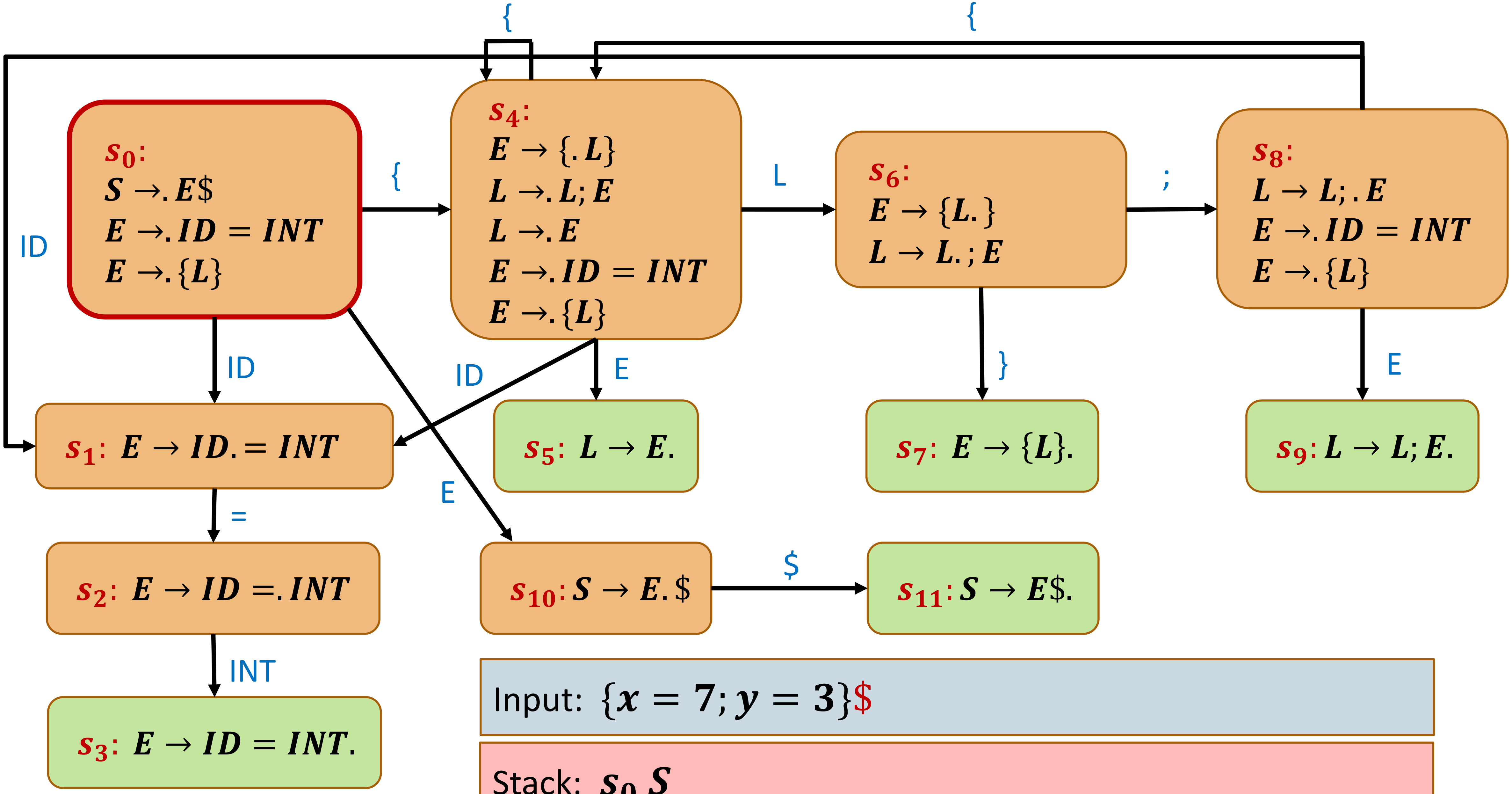



Input: $\{x = 7; y = 3\}\$$

Stack: $s_0\ E\ s_{10}$



$s_0$: $S \to .E\$$ $E \to .ID = INT$ $E \to .\{L\}$
$s_1$: $E \to ID. = INT$
$s_2$: $E \to ID =. INT$
$s_3$: $E \to ID = INT.$
$s_4$: $E \to \{.L\}$ $L \to .L; E$ $L \to .E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_5$: $L \to E.$
$s_6$: $E \to \{L.\}$ $L \to L.; E$
$s_7$: $E \to \{L\}.$
$s_8$: $L \to L;.E$ $E \to .ID = INT$ $E \to .\{L\}$
$s_9$: $L \to L; E.$
$s_{10}$: $S \to E.\$$
$s_{11}$: $S \to E\$.$
{
ID
L
;
}
E
=
INT
$
Input: $\{x = 7; y = 3\}\$$
Stack: $s_0\ E\ s_{10}\ \$\ s_{11}$




Input: $\{x = 7; y = 3\}\$$

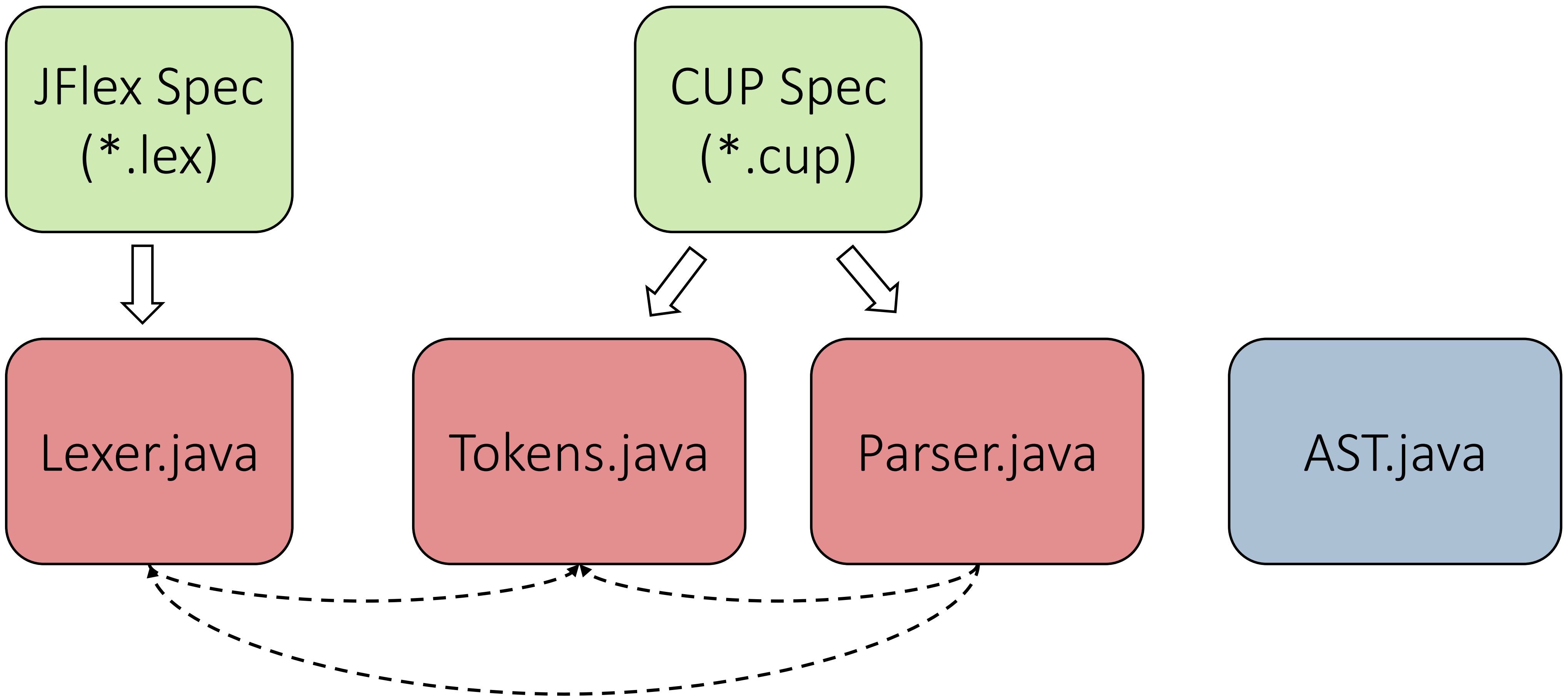
Stack: $s_0\ S$

# Parsing with CUP

# CUP

- Given a user-specified grammar, generates an LALR parser
- Works with JFlex, which provides the parsed tokens
- Other tools:
  - Bison (for C)

# CUP/JFlex Workflow



# CUP Format

**parser setup**

```
parser code {:
...
:}
```

**lexer setup**

```
scan with {:
...
:}
```

**grammar**

```
terminal …
non terminal …
start with …
<derivation rules…>
```

# CUP Spec: Parser Setup

```
parser code {:
    public Lexer lexer;
    public Parser(Lexer lexer) {
        super(lexer);
        this.lexer = lexer;
    }
    public void report_error(String message, Object info) {
        System.exit(0);
    }
:}
```

# CUP Spec: Lexer Setup

```
scan with {:
        Symbol s;
        s = lexer.next_token();
        // print token...
        return s;
:};
```

# CUP Spec: Terminals

terminal T1;
terminal T2;
terminal T3;
terminal T4;
…

# CUP Spec: Non-Terminals

non terminal AST_NODE_1 E1;
non terminal AST_NODE_2 E2;
non terminal AST_NODE_3 E3;
…

# CUP Spec: Operator Precedence

precedence left OP1;
precedence left OP2;
precedence left OP3;
precedence left OP4;
…

These are token names…

# CUP Spec: Grammar

```
start with S;
S ::=
  E1:v1 E2:v2 {: RESULT = new AST_NODE_1(v1, v2);  :} |
  E3:v3       {: RESULT = new AST_NODE_2(v3);  :} ;

E1 ::= ID:id    {: RESULT = new AST_NODE_3(id);   :}

E2 ::= INT:i    {: RESULT = new AST_NODE_4(i);  :}

E3 ::= ...
```

# CUP Spec: AST Nodes

- We need to **decide** which node types we have in our AST
- We need to **define** the classes for these AST nodes

# CUP Example

Consider the following CFG:

- $E \rightarrow INT$
- $E \rightarrow V$
- $E \rightarrow E + E$
- $E \rightarrow E \; - E$
- $V \rightarrow ID$
- $V \rightarrow V\,.\,ID$

# CUP Example: Terminals

terminal Integer INT;
terminal String ID;
terminal PLUS;
terminal MINUS;
terminal DOT;

# CUP Example: Non-Terminals

non terminal AST_EXP exp;
non terminal AST_VAR var;

# CUP Example: Operator Precedence

precedence left PLUS;
precedence left MINUS;

# CUP Example: Grammar

```
start with EXP;
EXP ::=
    INT:i {: RESULT = new AST_EXP_INT(i); :} |
    VAR:v {: RESULT = new AST_EXP_VAR(v); :} |
    EXP:e1 PLUS  EXP:e2 {: RESULT = new AST_EXP_BINOP(e1, e2, 0); :} |
    EXP:e1 MINUS EXP:e2 {: RESULT = new AST_EXP_BINOP(e1, e2, 1); :};
VAR ::=
    ID:name {: RESULT = new AST_VAR_SIMPLE(name); :} |
    VAR:v DOT ID:fieldName {: RESULT = new AST_VAR_FIELD(v, fieldName); :};
```

# CUP Example: AST Nodes

For the non-terminal ***VAR***:

```
public abstract class AST_VAR extends AST_Node {

}
```

# CUP Example: AST Nodes

For the rule ***VAR ::= ID:name***:

```
public class AST_VAR_SIMPLE extends AST_VAR {
	public String name;
	public AST_VAR_SIMPLE(String name) {
		this.name = name;
	}
}
```

# CUP Example: AST Nodes

For the rule ***VAR ::= VAR:v DOT ID:fieldName*** :

```
public class AST_VAR_FIELD extends AST_VAR {
        public AST_VAR var;
        public String fieldName;
        public AST_VAR_FIELD(AST_VAR var, String fieldName) {
                this.var = var;
                this.fieldName = fieldName;
        }
}
```

# CUP Example: AST Nodes

For the non-terminal ***EXP***:

public abstract class **AST_EXP** extends AST_Node {

}

# CUP Example: AST Nodes

For the rule ***EXP ::= INT:i***:

```
public class AST_EXP_INT extends AST_EXP {
	public int value;
	public AST_EXP_INT(int value)	{
		this.value = value;
	}
}
```

# CUP Example: AST Nodes

For the rule ***EXP ::= VAR:v***:

```
public class AST_EXP_VAR extends AST_EXP {
	public AST_VAR var;
	public AST_EXP_VAR(AST_VAR var) {
		this.var = var;
	}
}
```

# CUP Example: AST Nodes

For the rule ***EXP ::= EXP:e1 <OP>  EXP:e2*** :

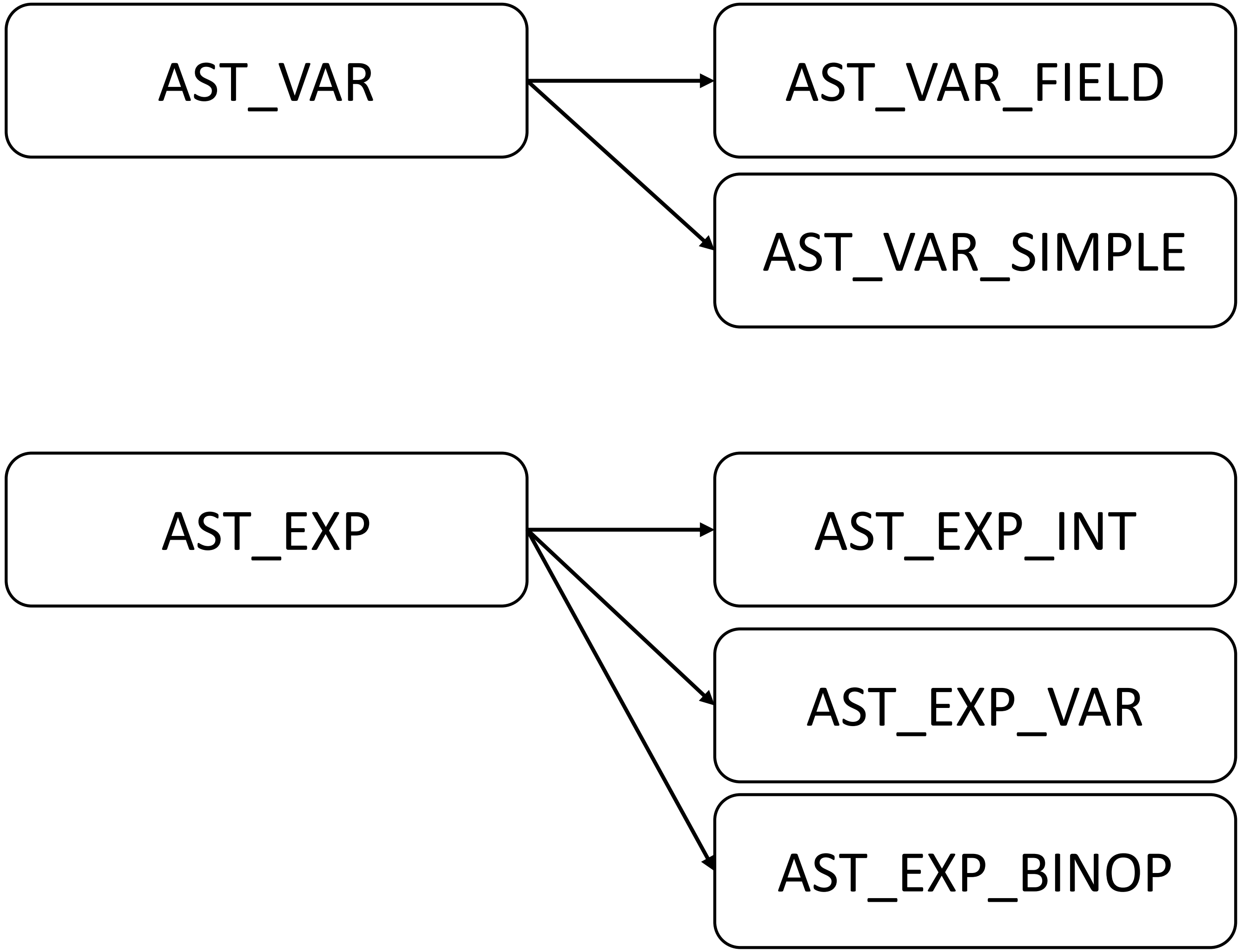
```
public class AST_EXP_BINOP extends AST_EXP {
	int OP;
	public AST_EXP left;
	public AST_EXP right;
	public AST_EXP_BINOP(AST_EXP left, AST_EXP right, int OP) {
		this.left = left;
		this.right = right;
		this.OP = OP;
	}
}
```

# CUP Example: AST Nodes

For the rule ***VAR:v DOT ID:fieldName***:

```
public class AST_VAR_FIELD extends AST_VAR {
	public AST_VAR var;
	public String fieldName;
	public AST_VAR_FIELD(AST_VAR var, String fieldName) {
		this.var = var;
		this.fieldName = fieldName;
	}
}
```

# Class Hierarchy (Inheritance)



# CUP Example: Debugging

We can generate an image of the AST (using the exercise template)

For the input foo + 3 + obj.field we have: