

# 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 item is its  $\epsilon$ -closure.

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:

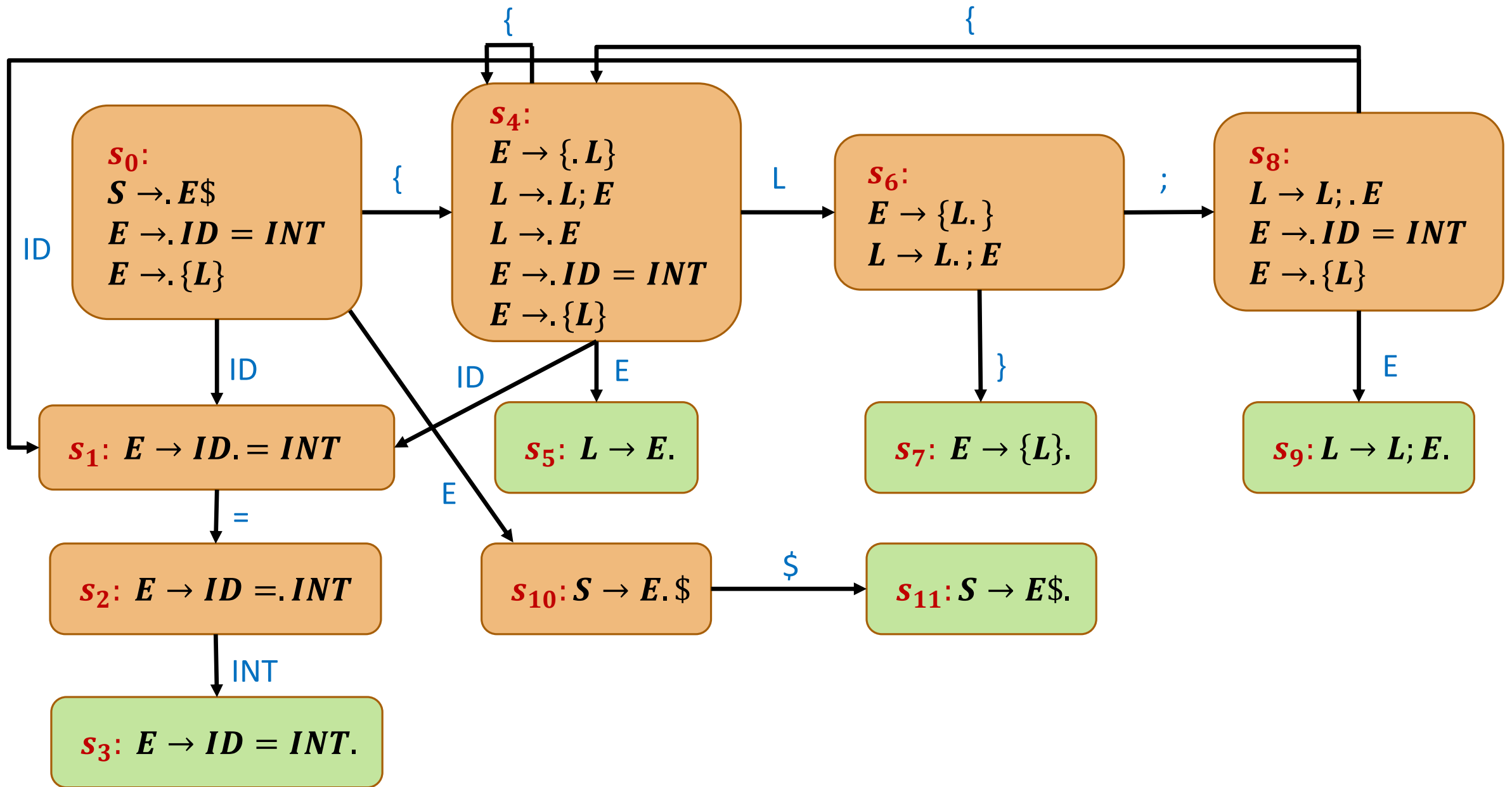
- $S \rightarrow .E\$$
- $E \rightarrow .ID = X$
- $E \rightarrow .\{L\}$

# 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?



# 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:

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

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



**$s_0$ :**

**$S \rightarrow .E\$$**

**$E \rightarrow .ID = INT$**

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

# 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

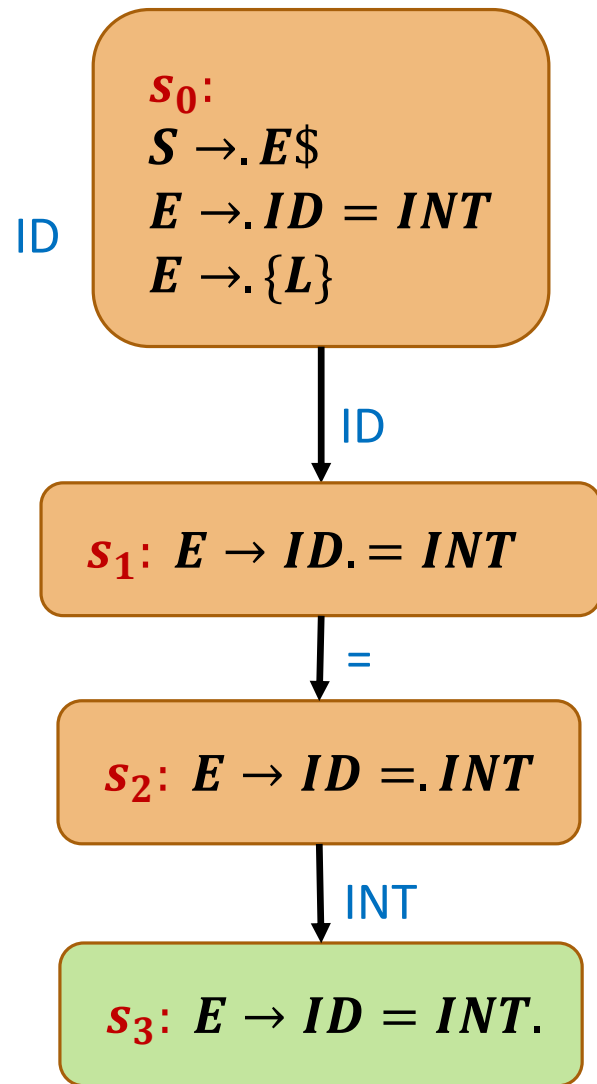
From  $s_2$ , if we recognized  $INT$ , then the next state will contain:

- $E \rightarrow ID = INT$ .

Which is a reduce state.

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





# 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

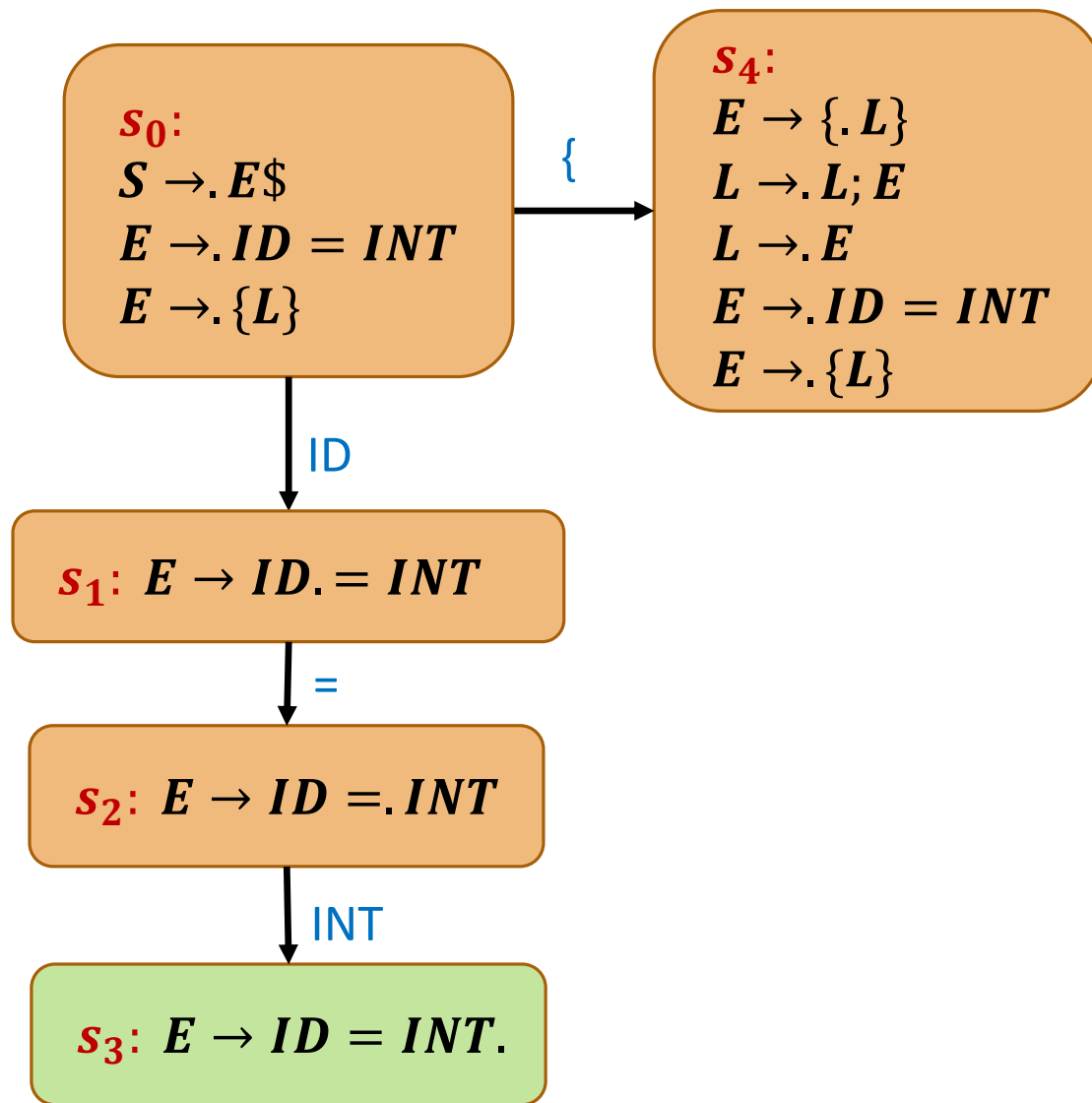
From  $s_0$ , if we recognized  $\{$ , then the next state will contain:

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

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

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

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



# LR(0) Parser

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

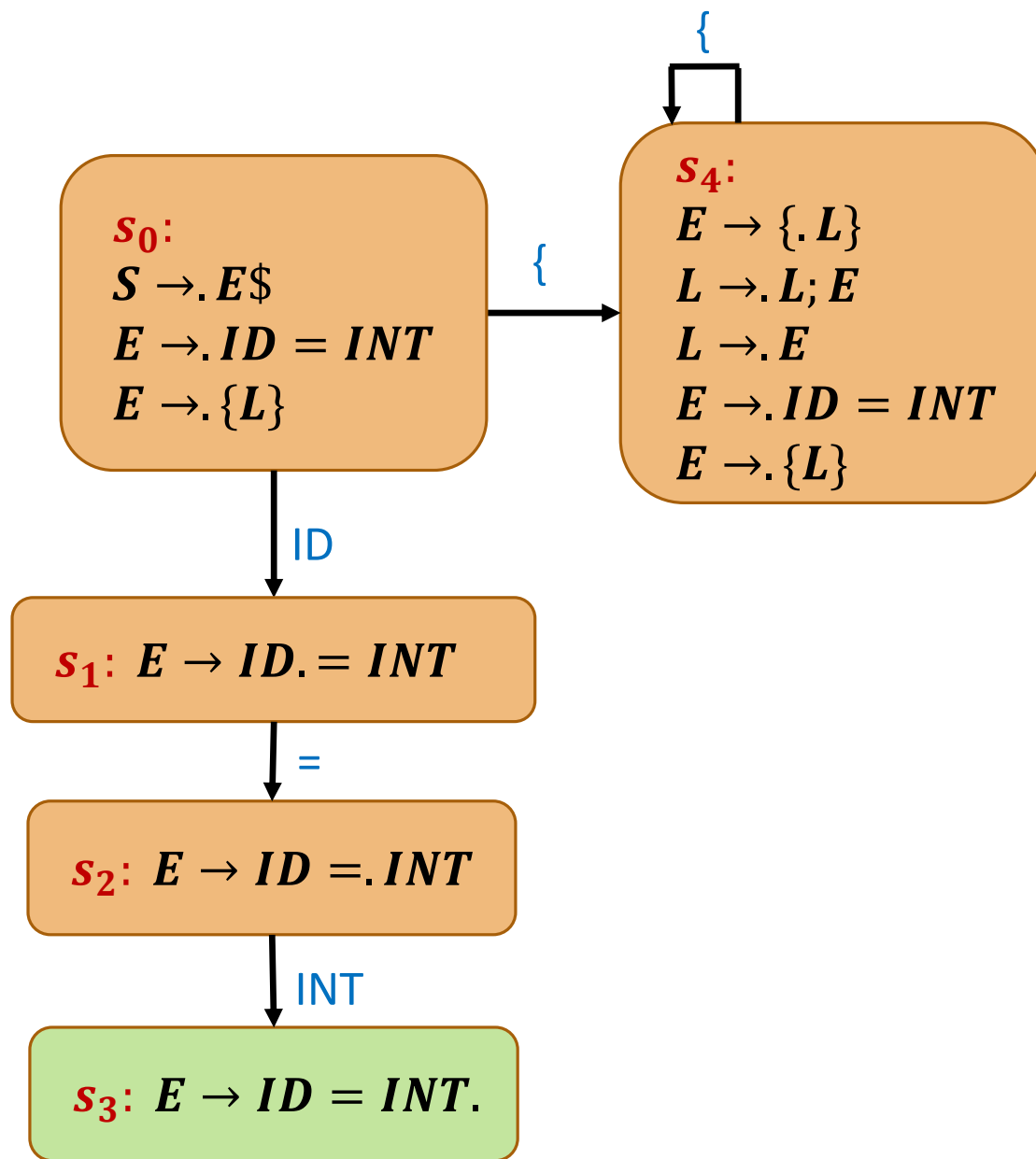
- $E \rightarrow \{.L\}$

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:  $s_4$

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



# LR(0) Parser

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

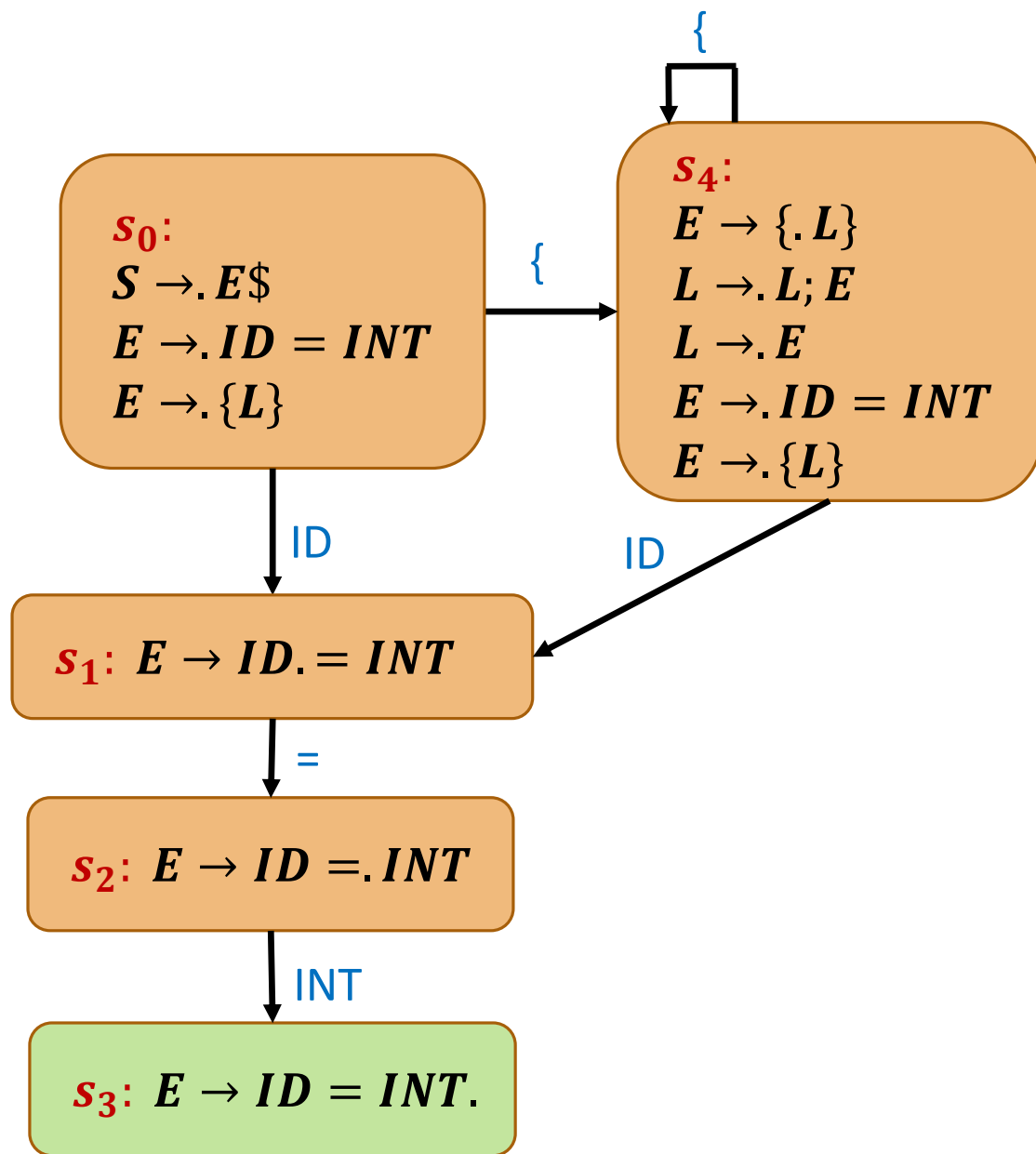
- $E \rightarrow ID. = INT$

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

- $E \rightarrow ID. = INT$

which was already computed:  $s_1$

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





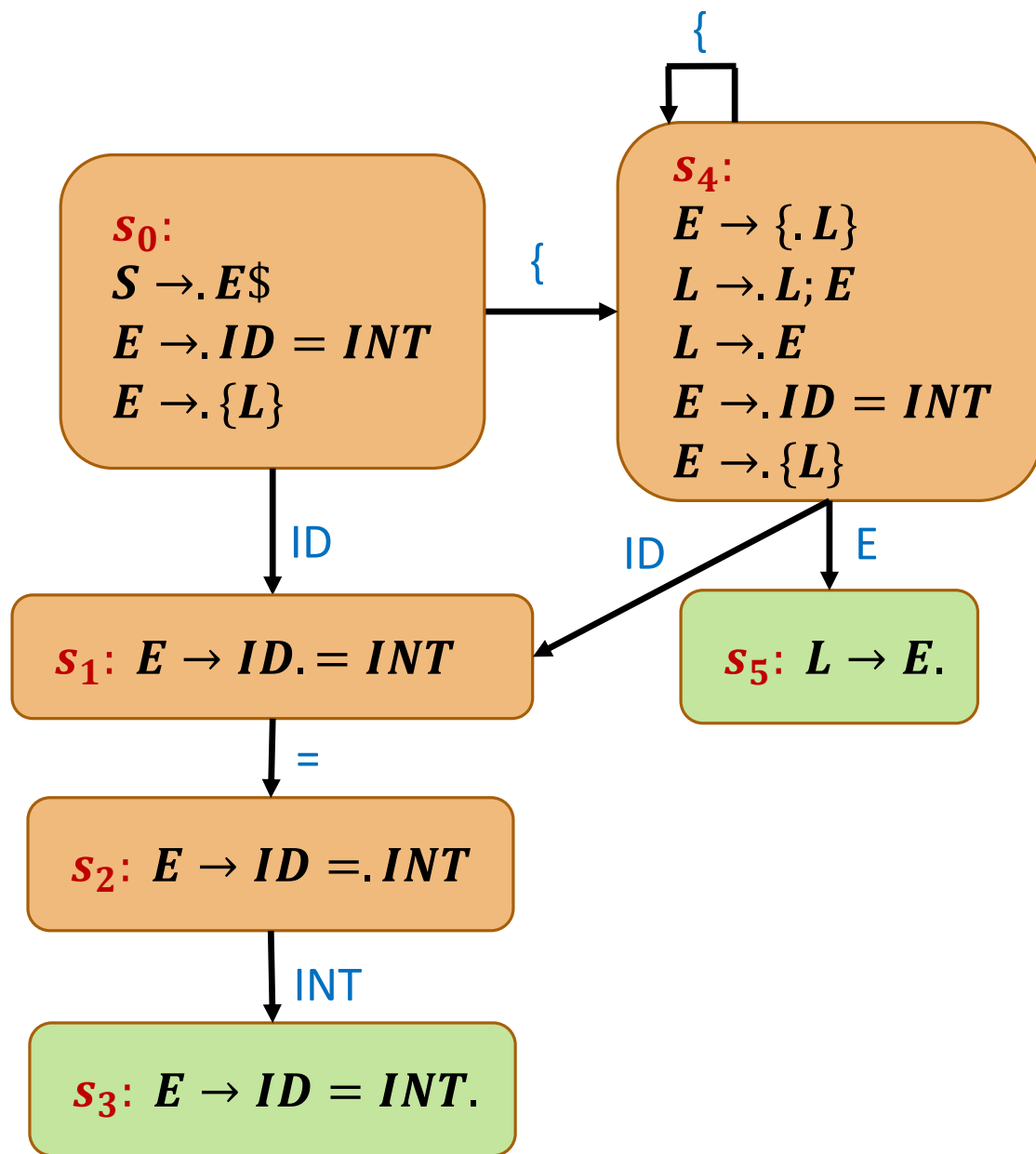
# LR(0) Parser

From  $s_5$ , if we recognized  $E$ , then the next state will contain:

- $L \rightarrow E$ .

which is a reduce state.

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



# 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:

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

# LR(0) Parser

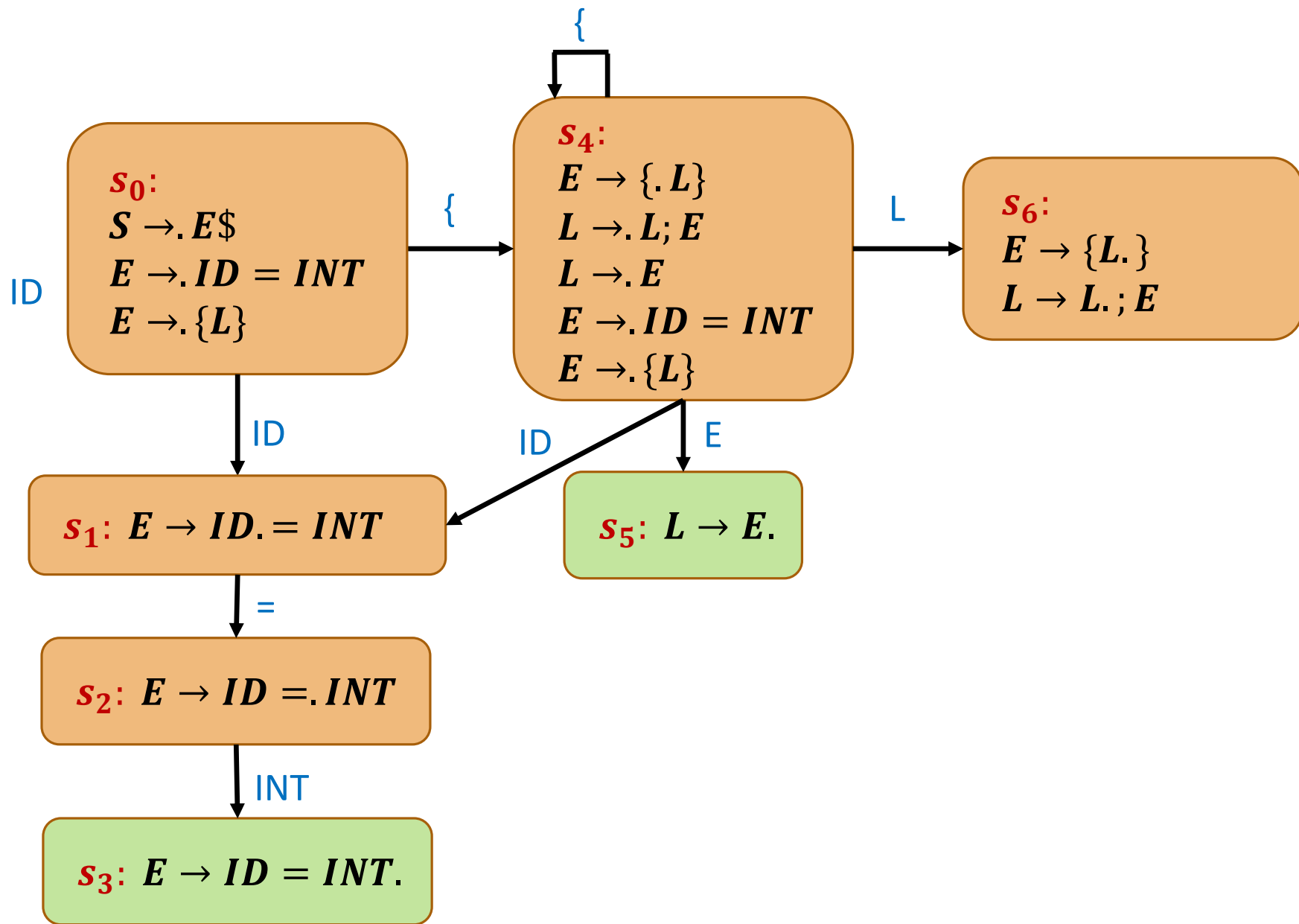
From  $s_4$ , if we recognized  $L$ , then the next state will contain:

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

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

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

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



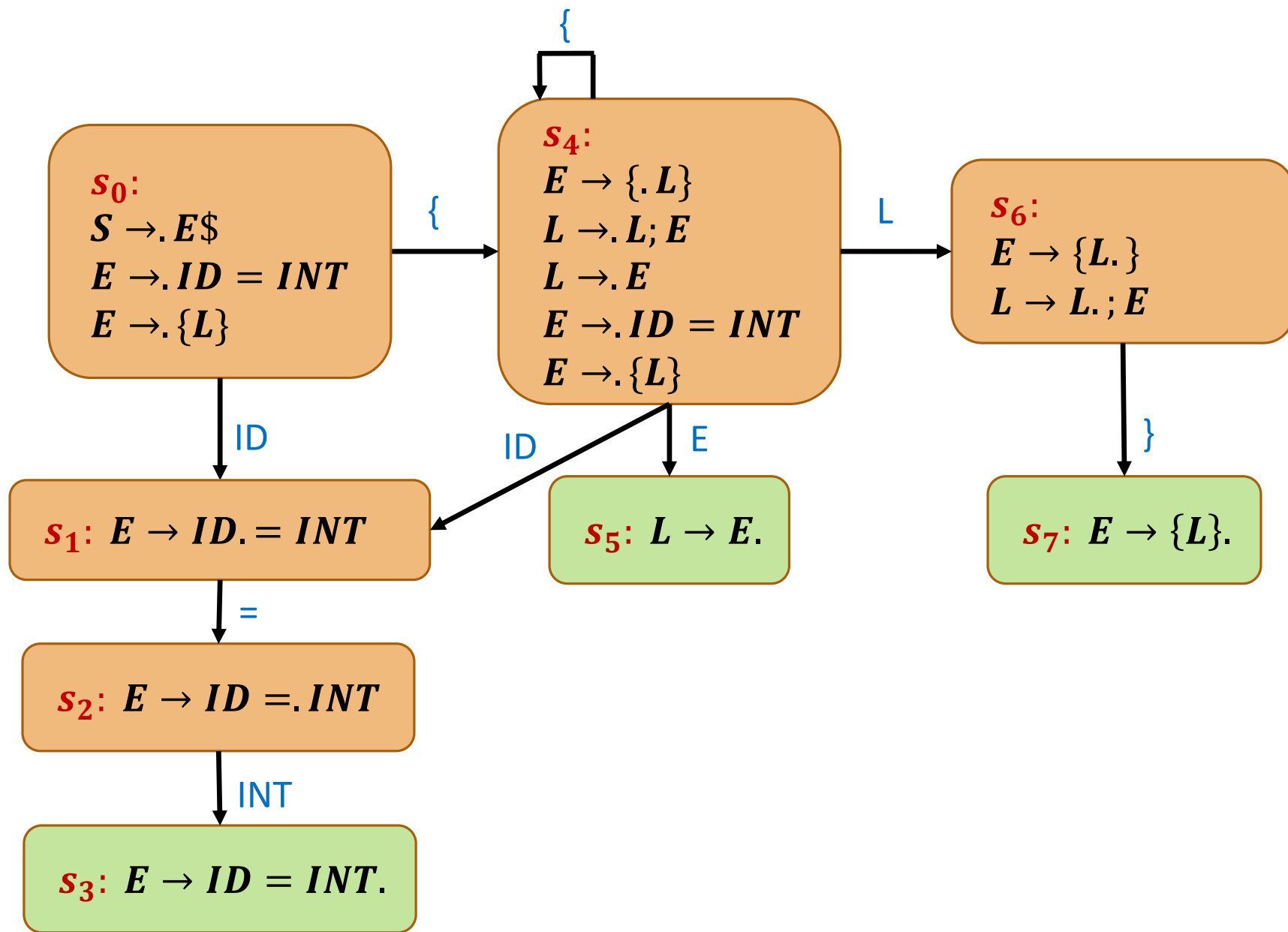
# LR(0) Parser

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

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

Which is a reduce state.

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



# 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:

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

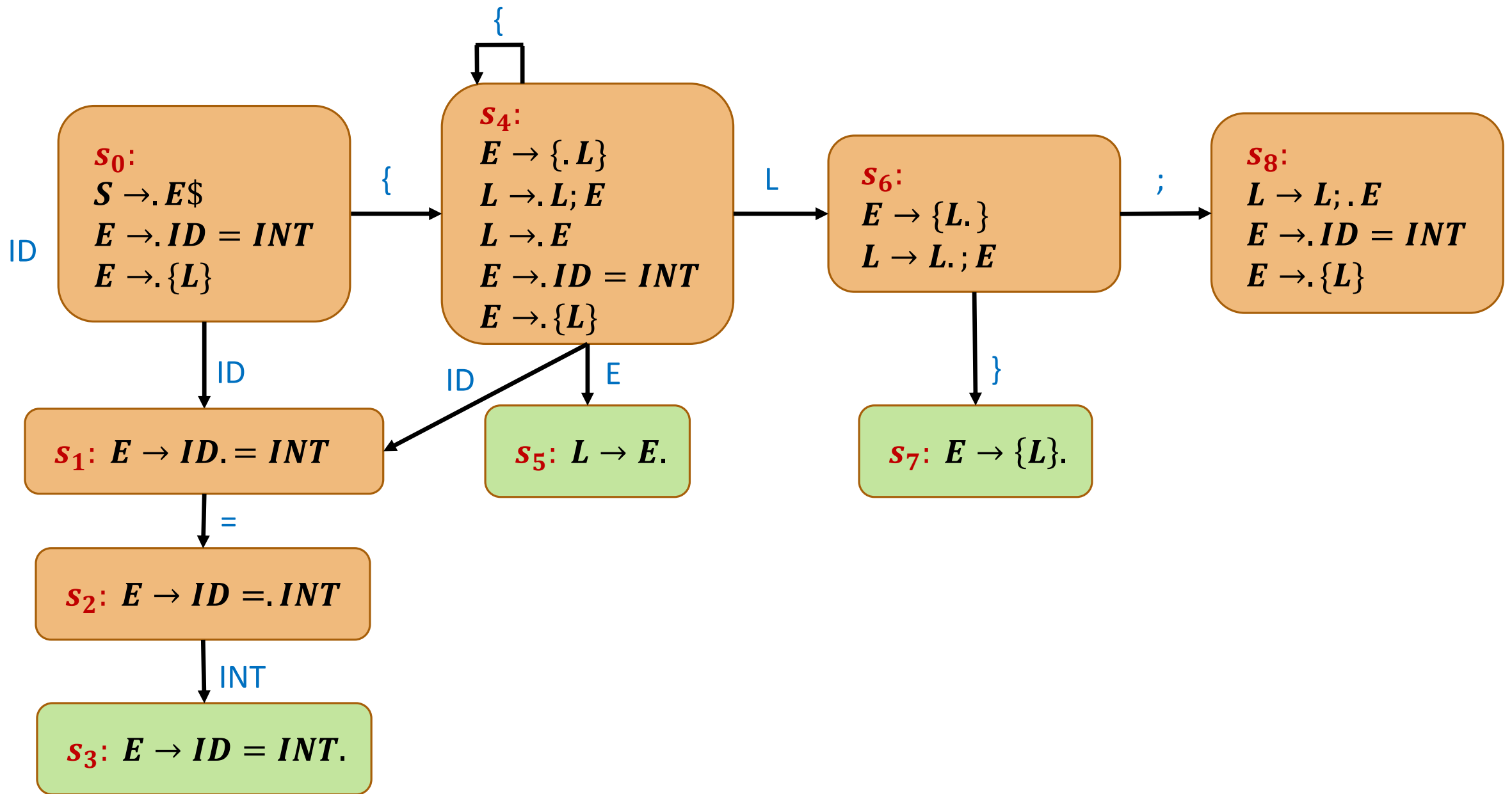
$S \rightarrow E\$$

$E \rightarrow ID = INT$

$E \rightarrow \{L\}$

$L \rightarrow E$

$L \rightarrow L; E$



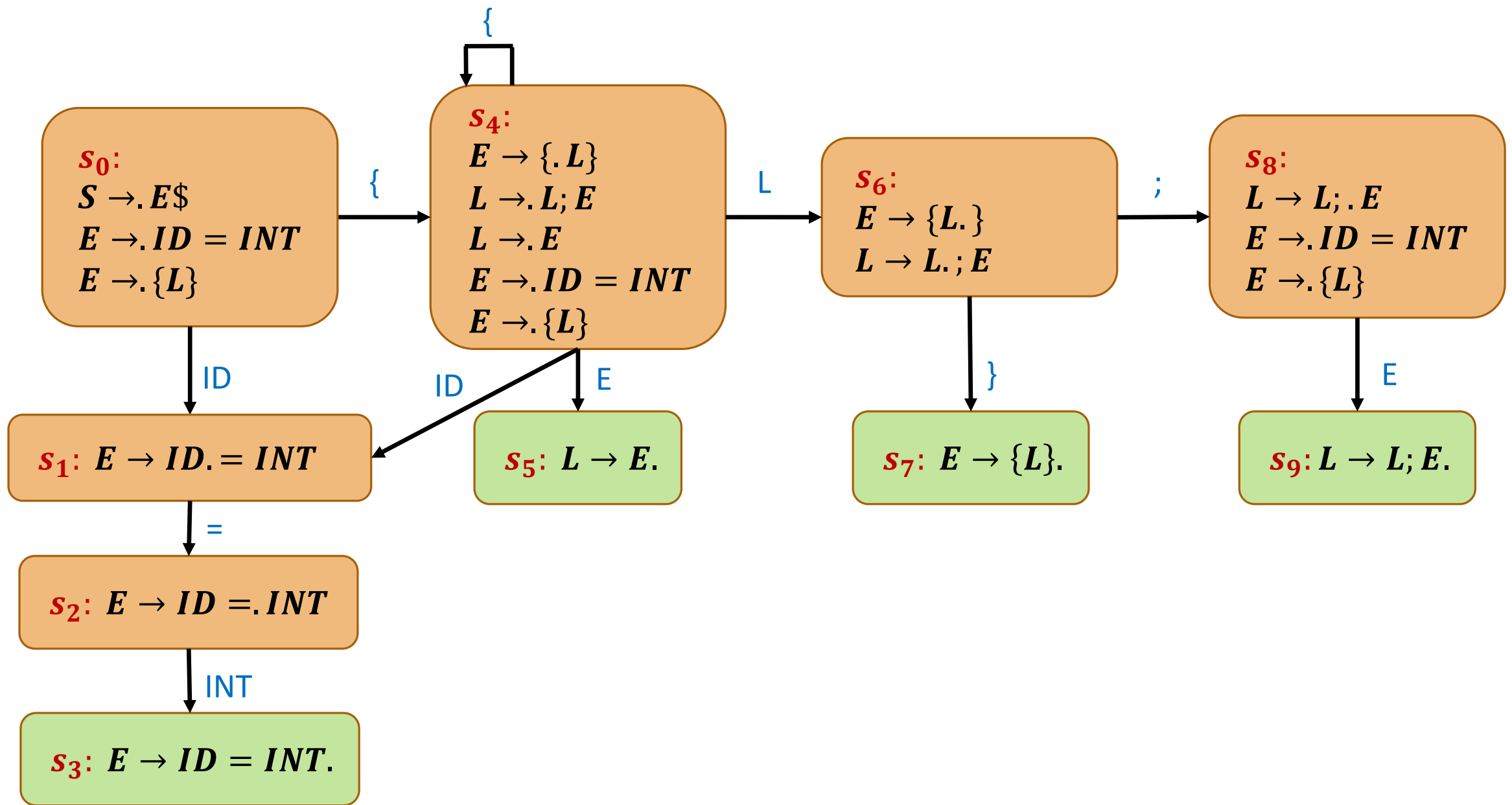
# LR(0) Parser

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

- $E \rightarrow L; E$ .

which is a reduce state.

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



# 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:

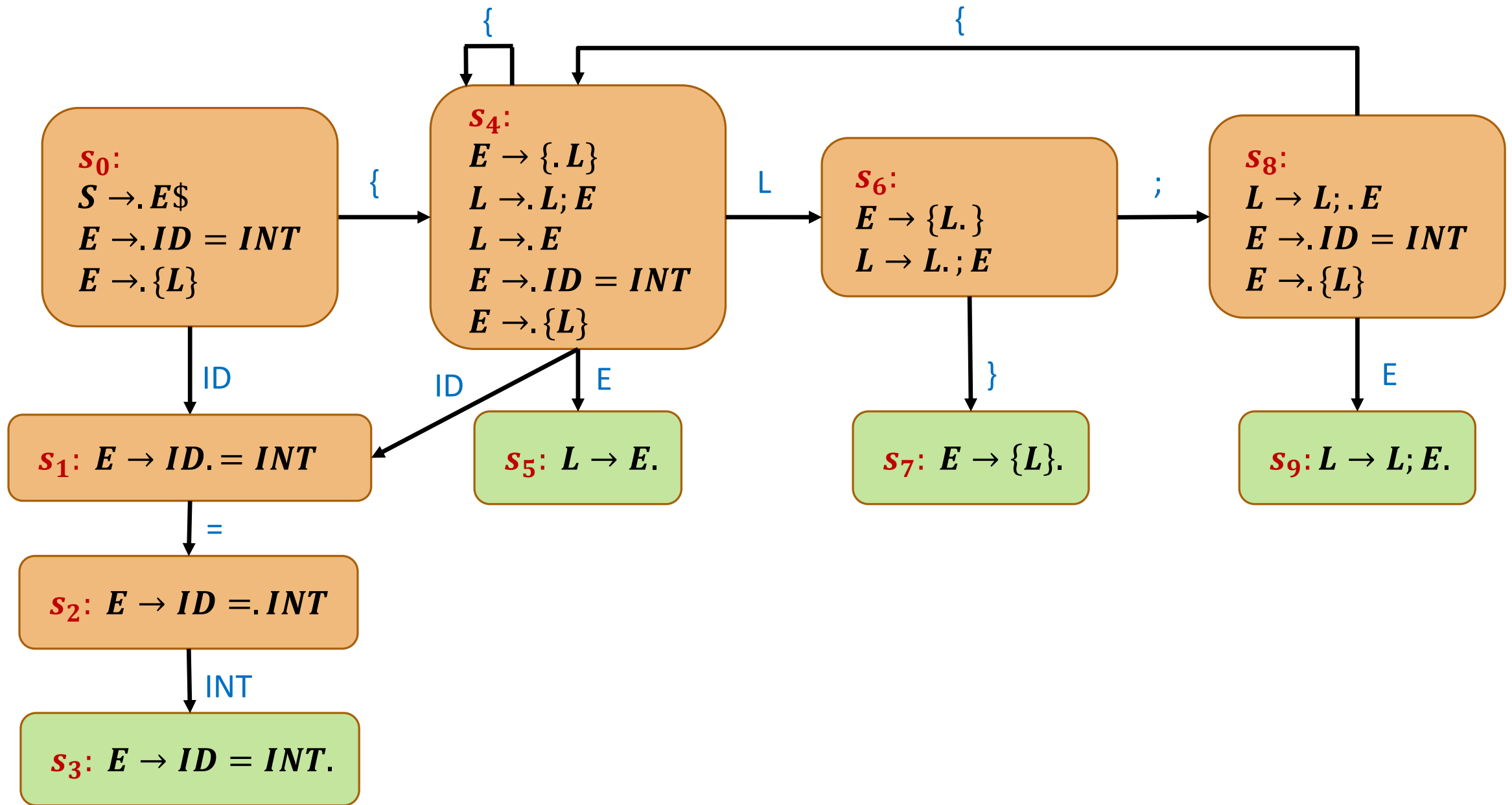
- $E \rightarrow \{.L\}$

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:  $s_4$

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



# 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:  $s_1$

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

# LR(0) Parser

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

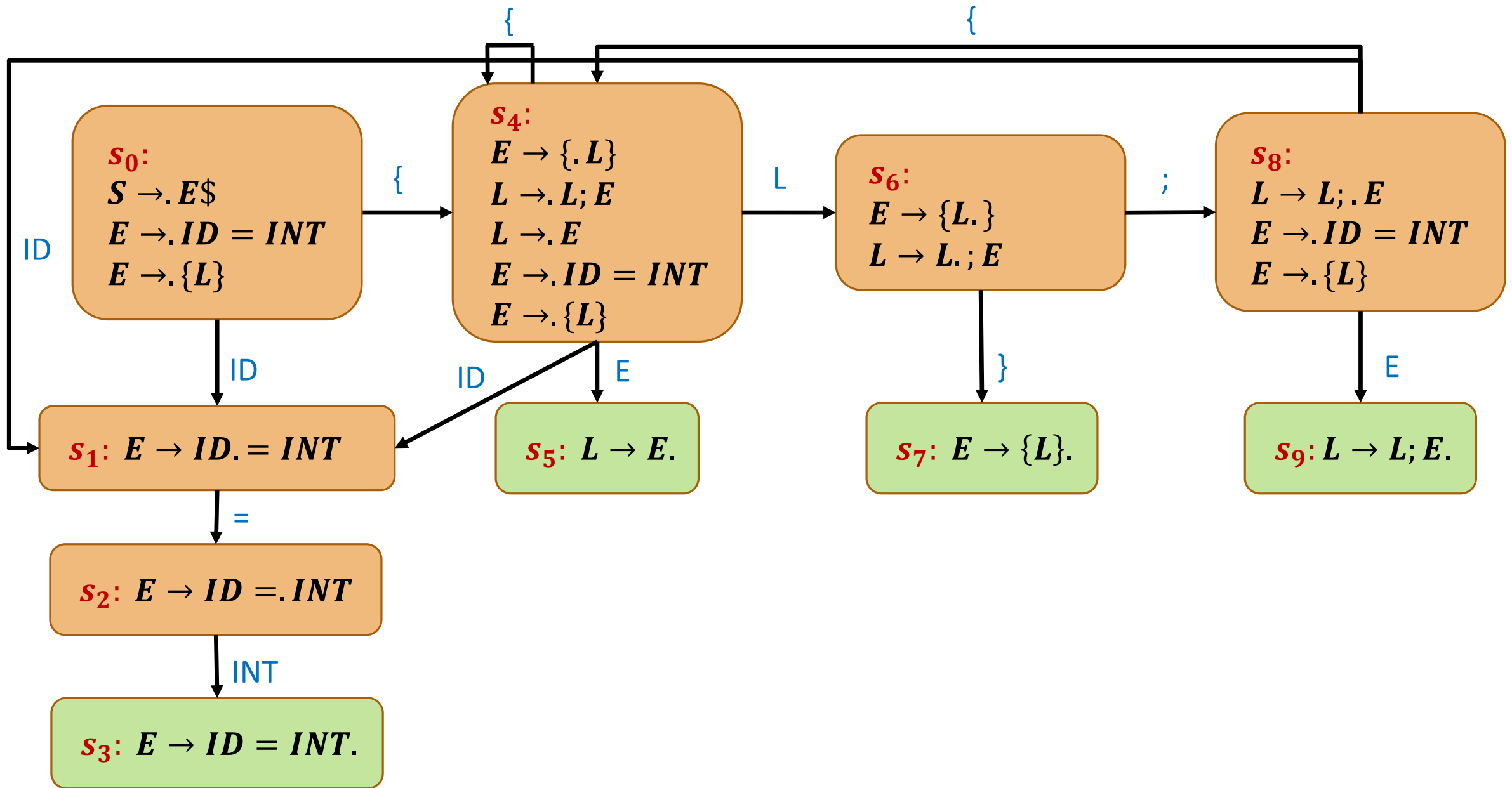
- $E \rightarrow ID. = INT$

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

- $E \rightarrow ID. = INT$

which was already computed:  $s_1$

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



# 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:

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

# LR(0) Parser

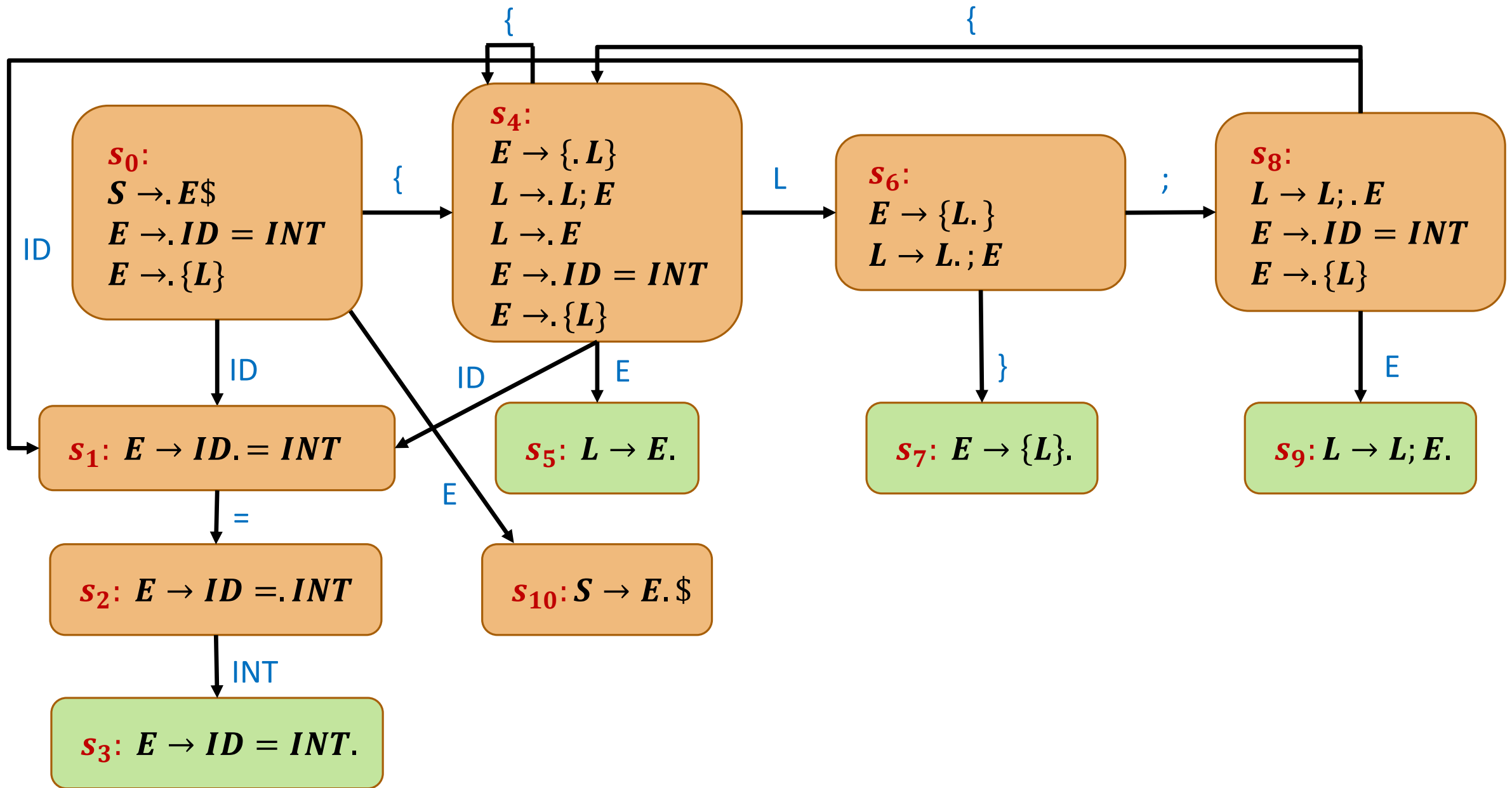
From  $s_0$ , if we recognized  $E$ , then the next state will contain:

- $S \rightarrow E.\$$

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

- $S \rightarrow E.\$$

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



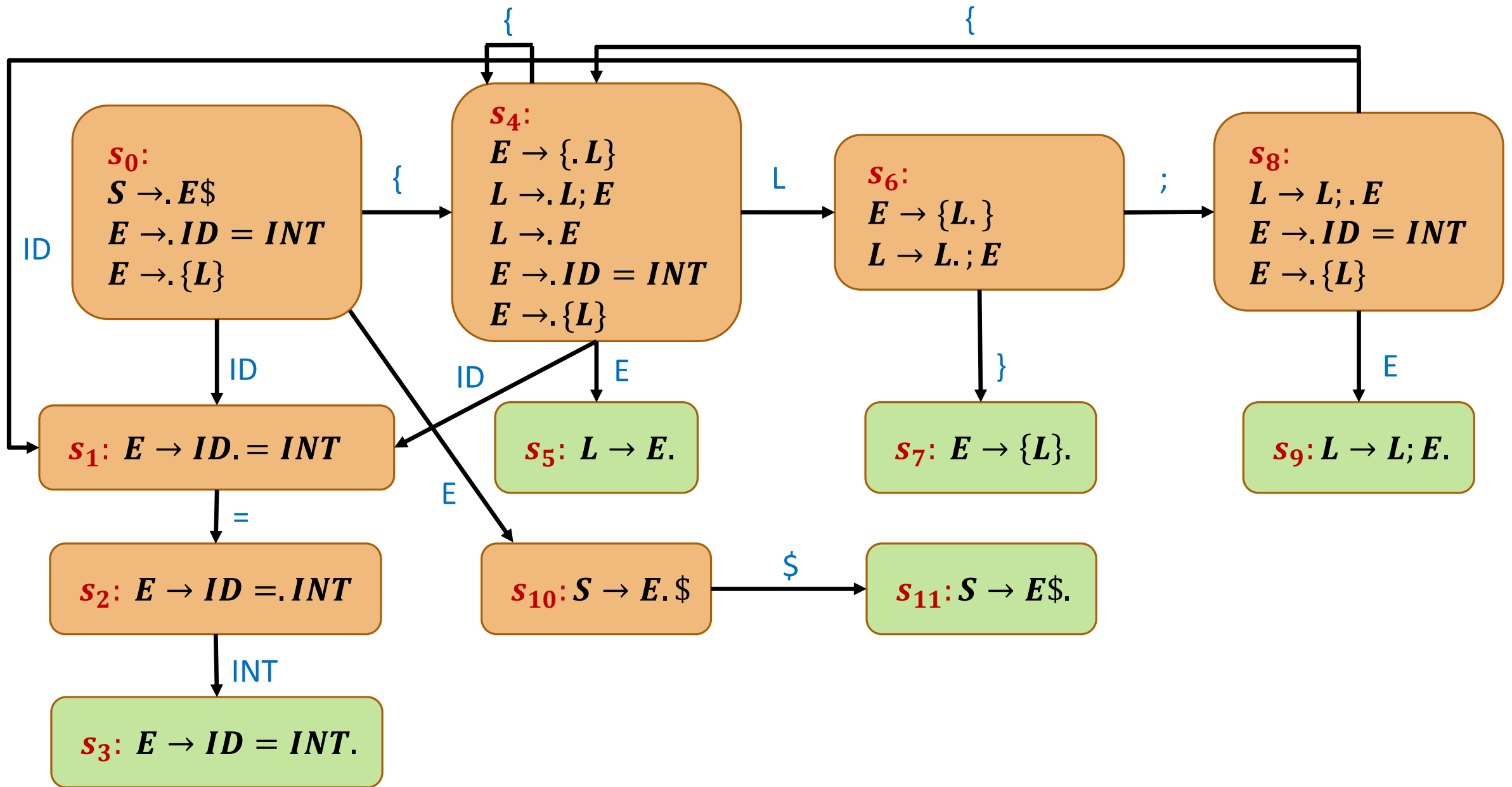
# LR(0) Parser

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

- $S \rightarrow E\$$ .

which is a reduce state.

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

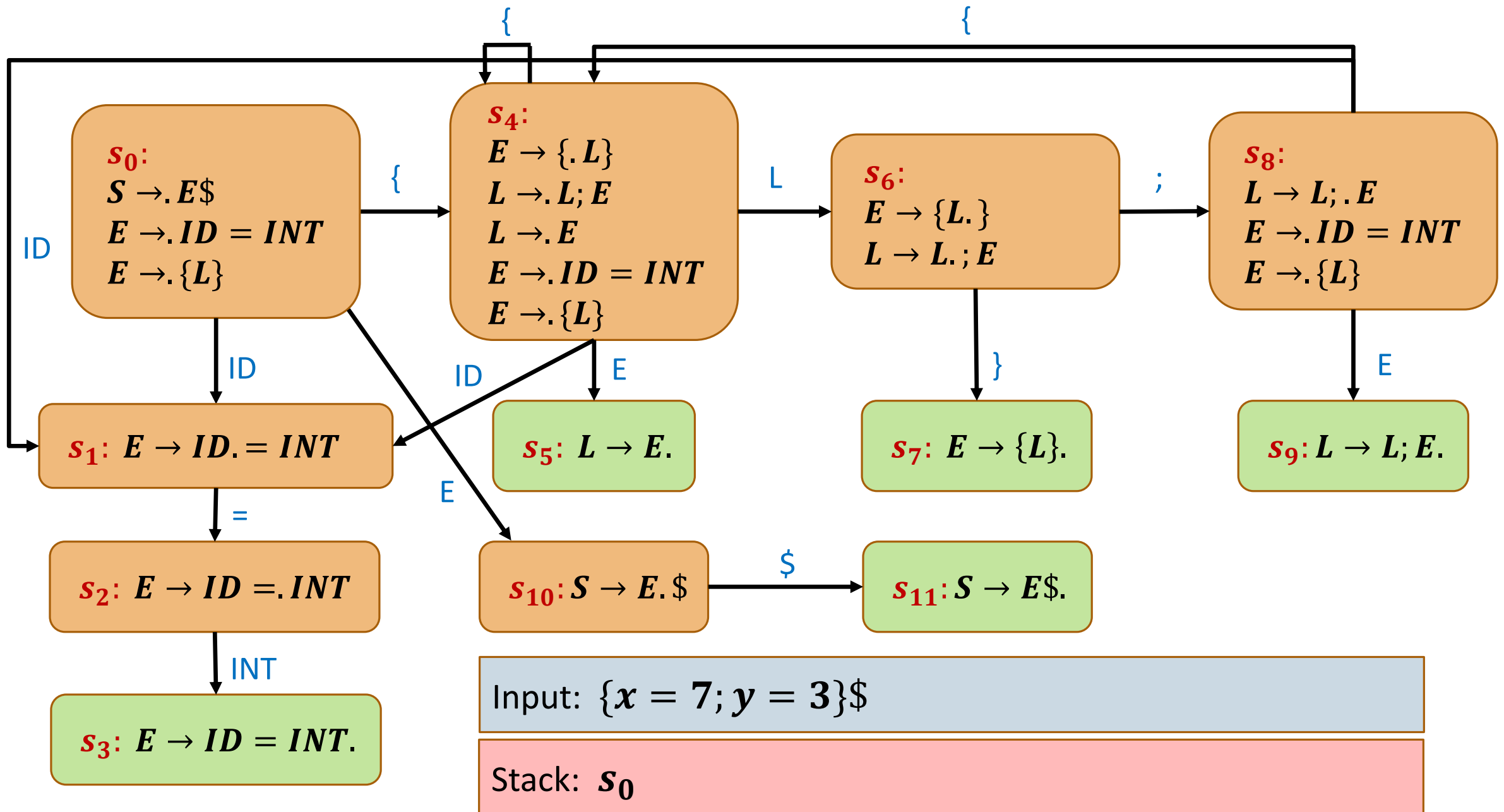


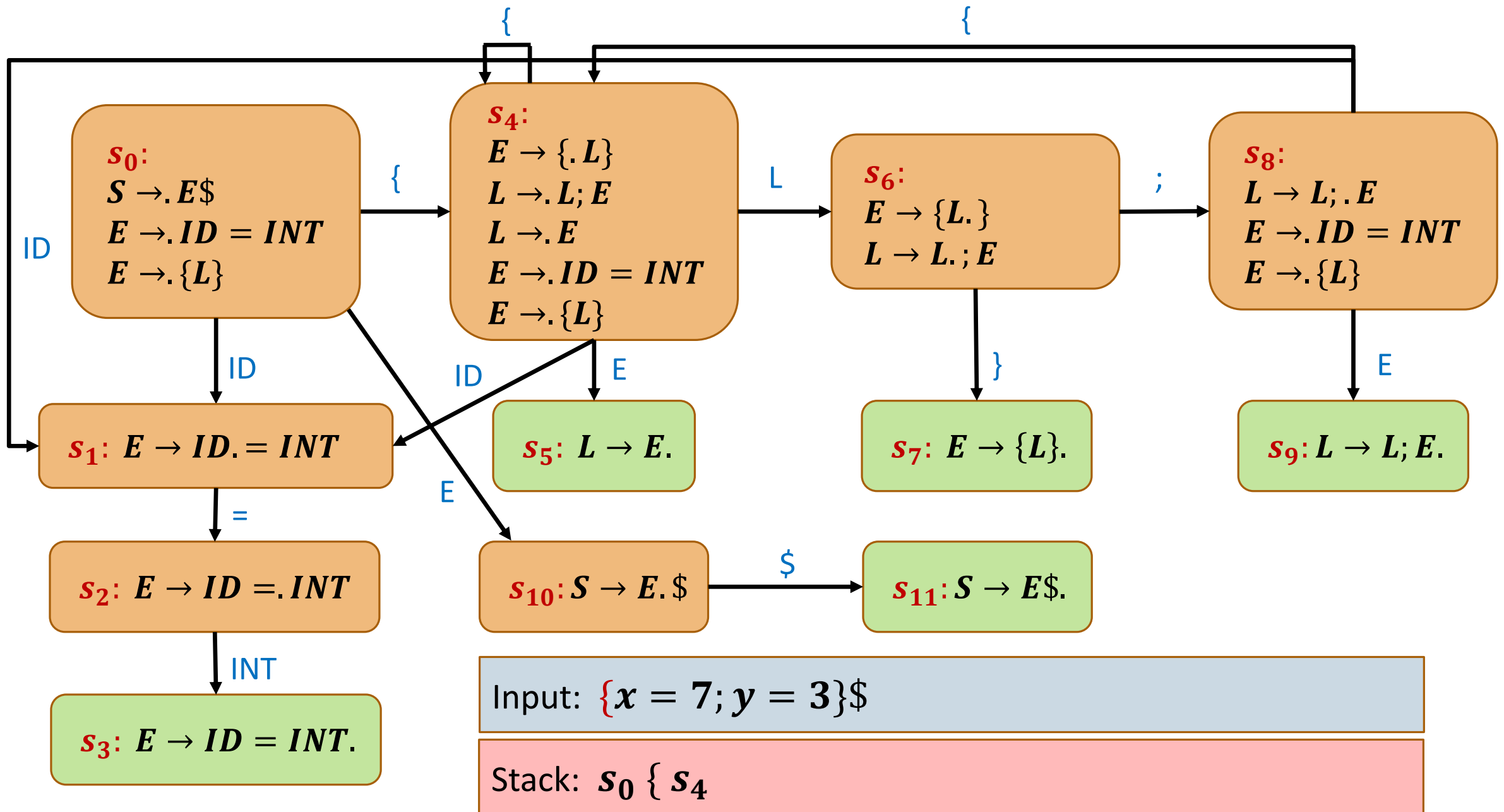


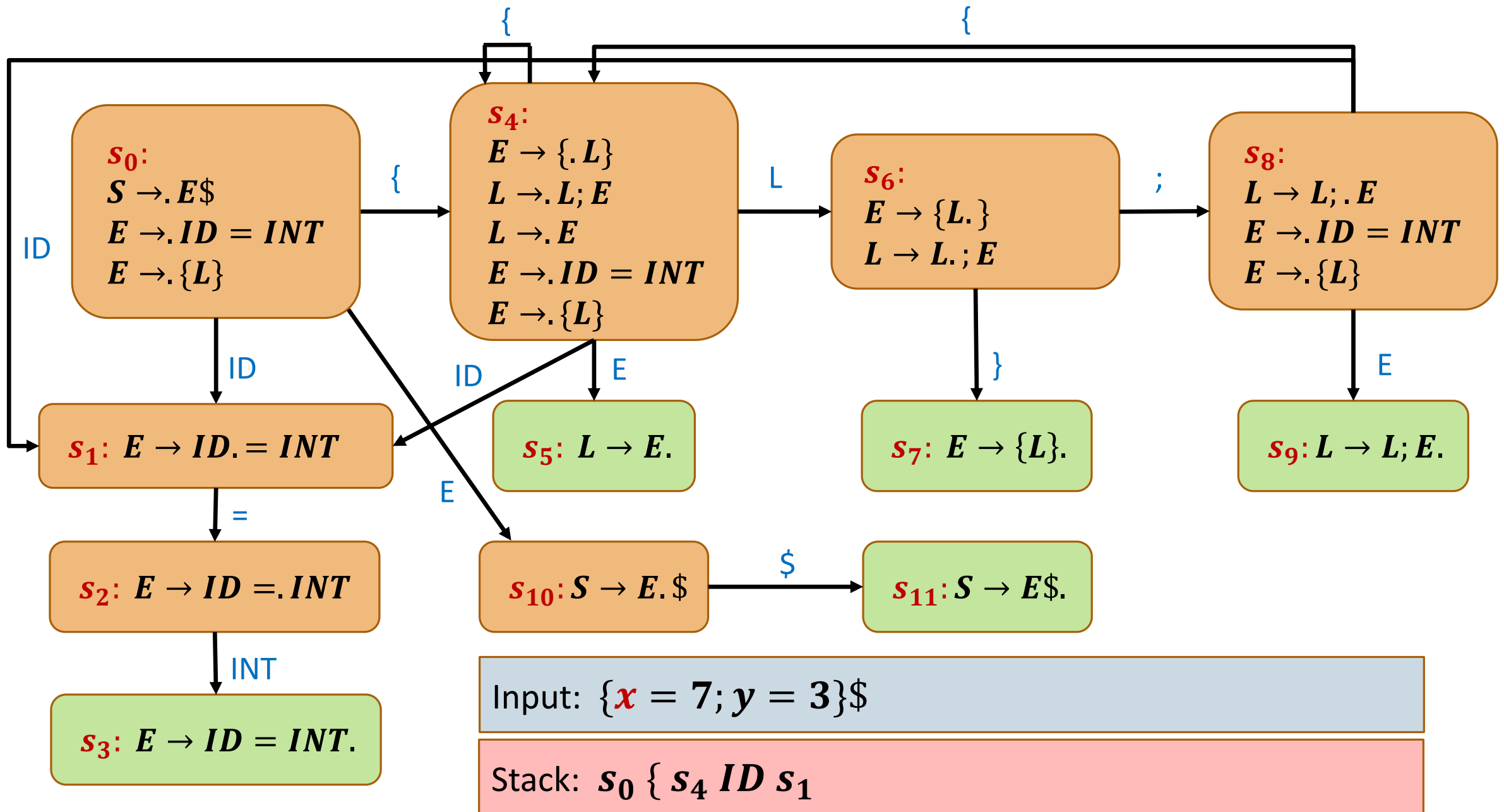
# LR(0) Parser: Running Example

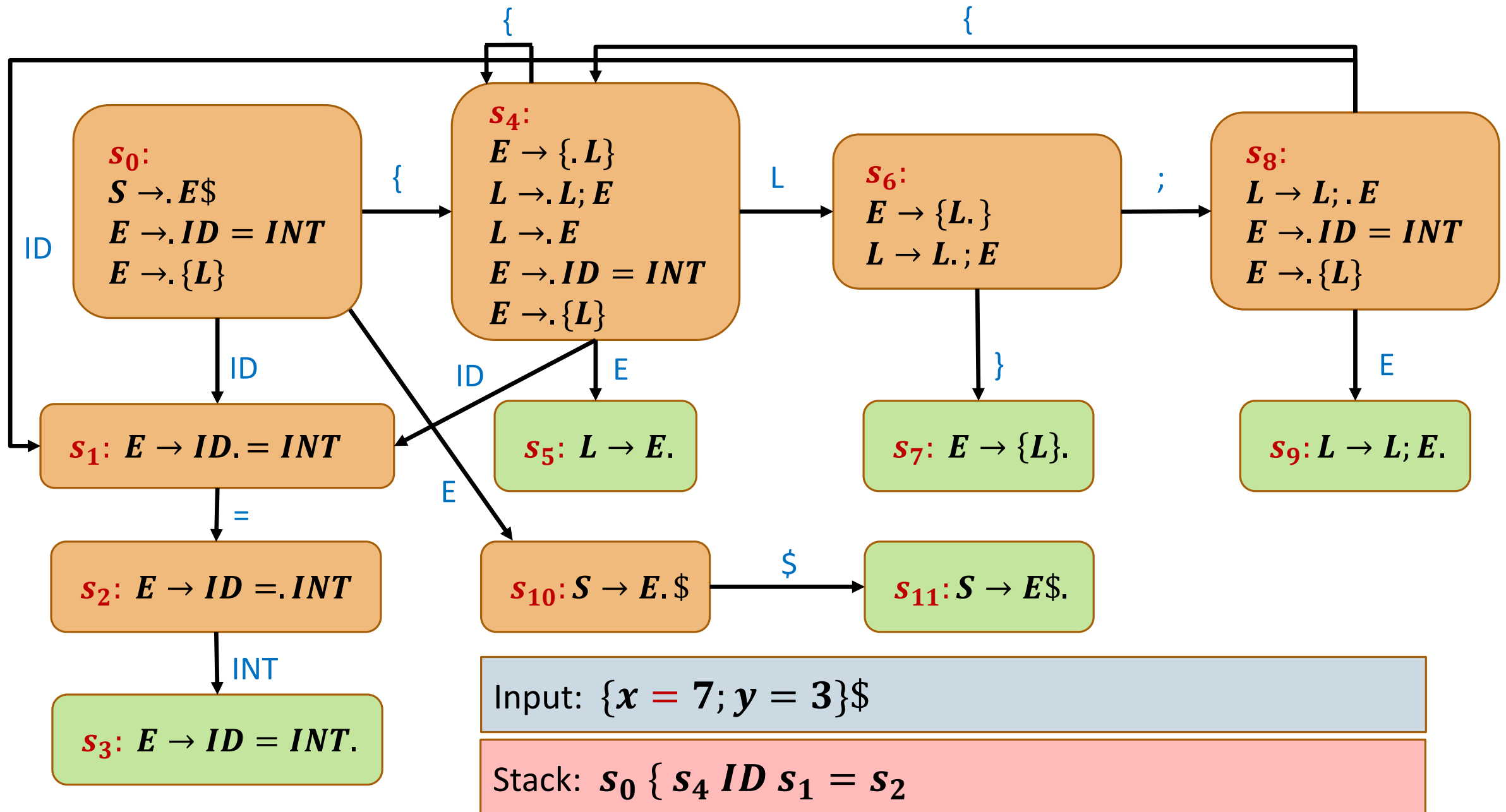
What will happen with the following input:

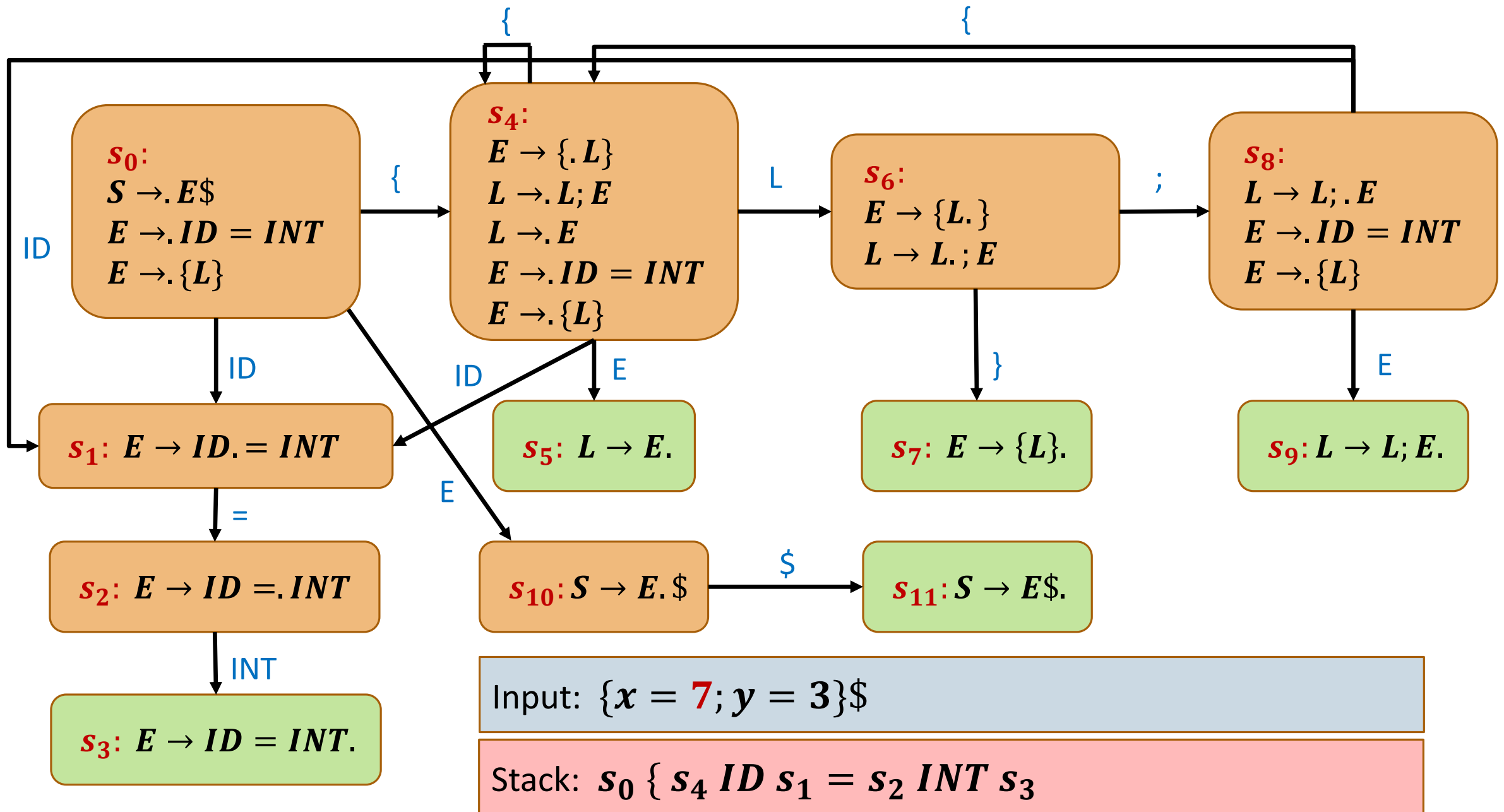
- $x = 7; y = 3\$$

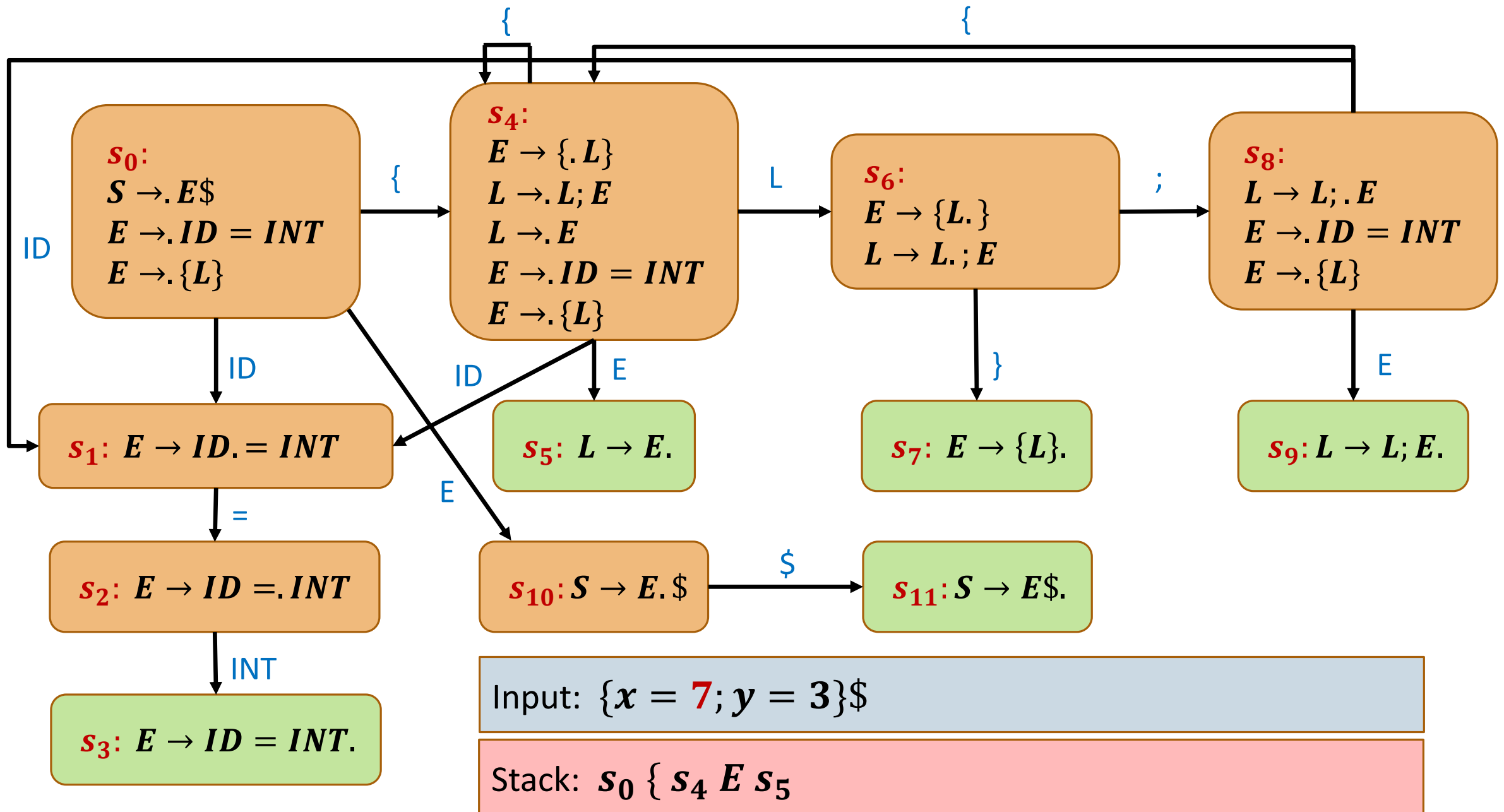


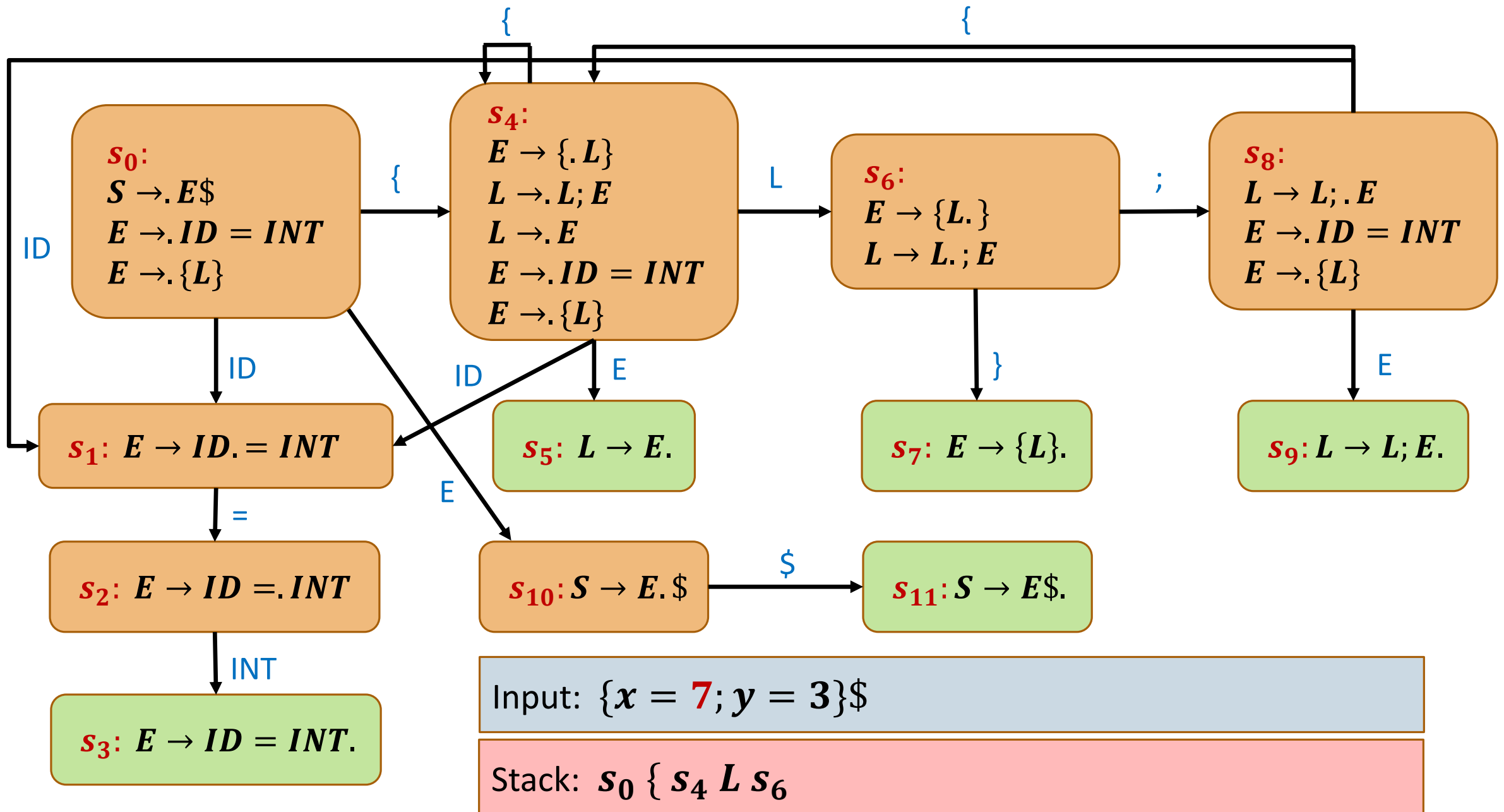




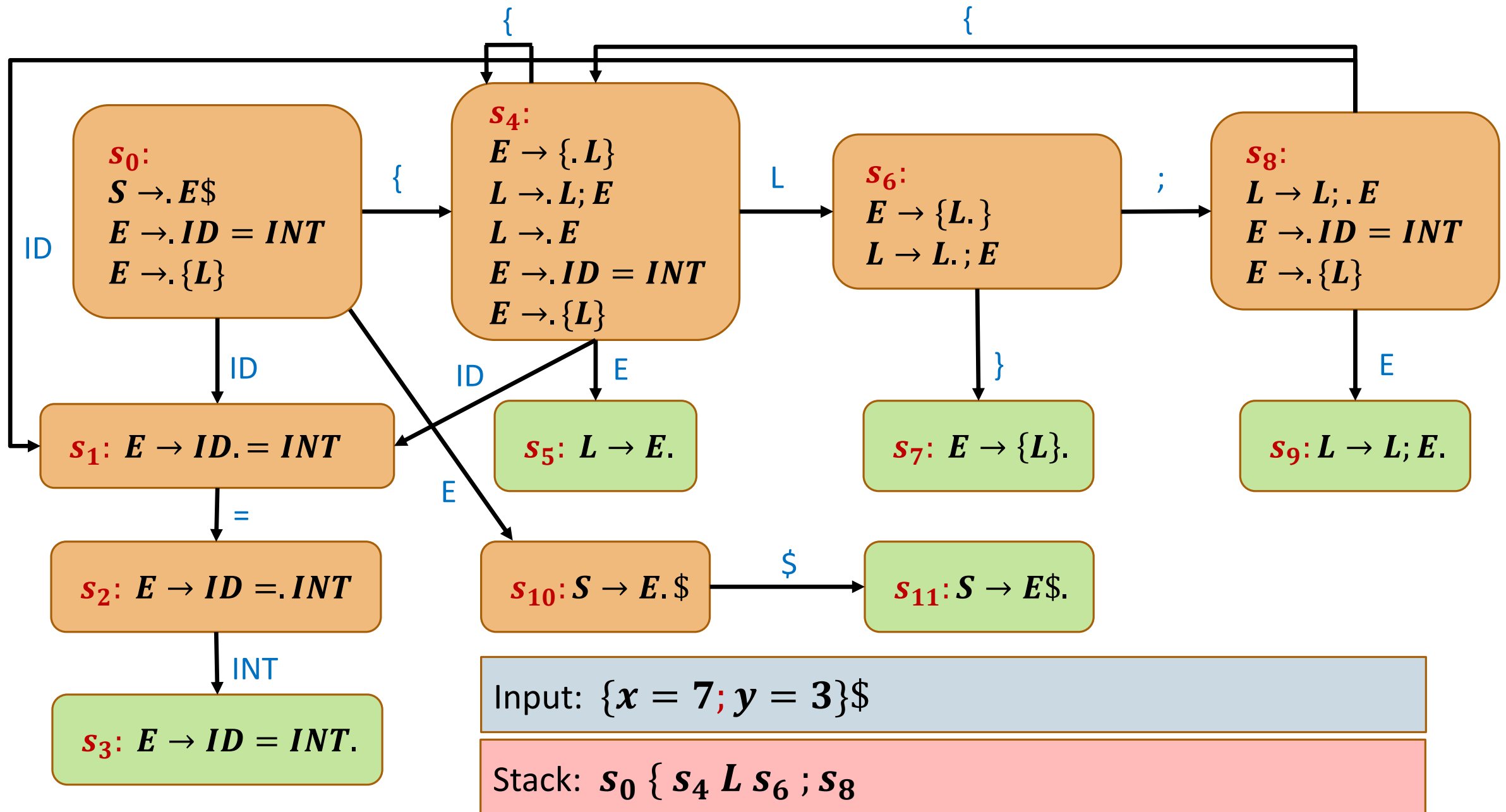


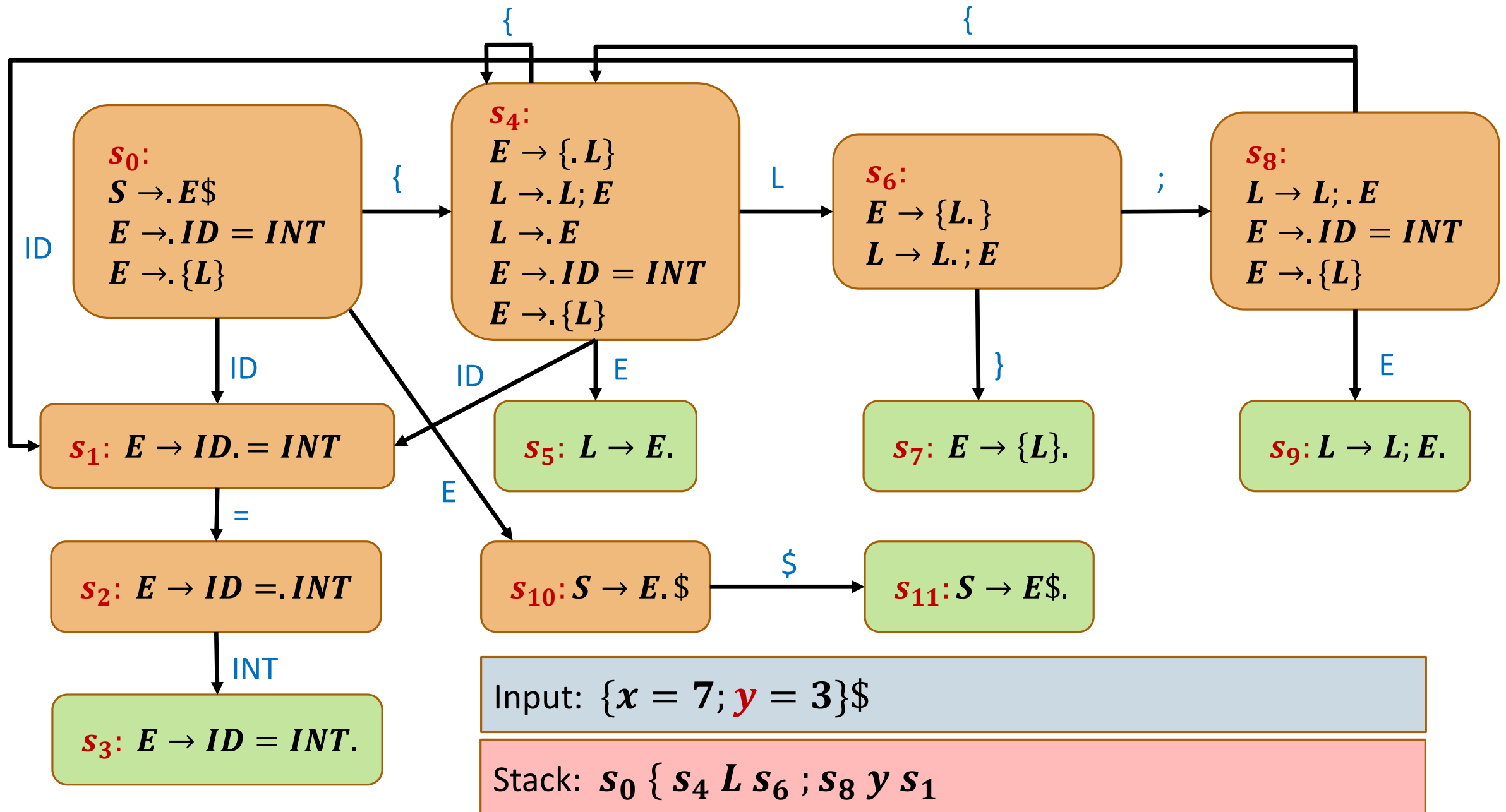


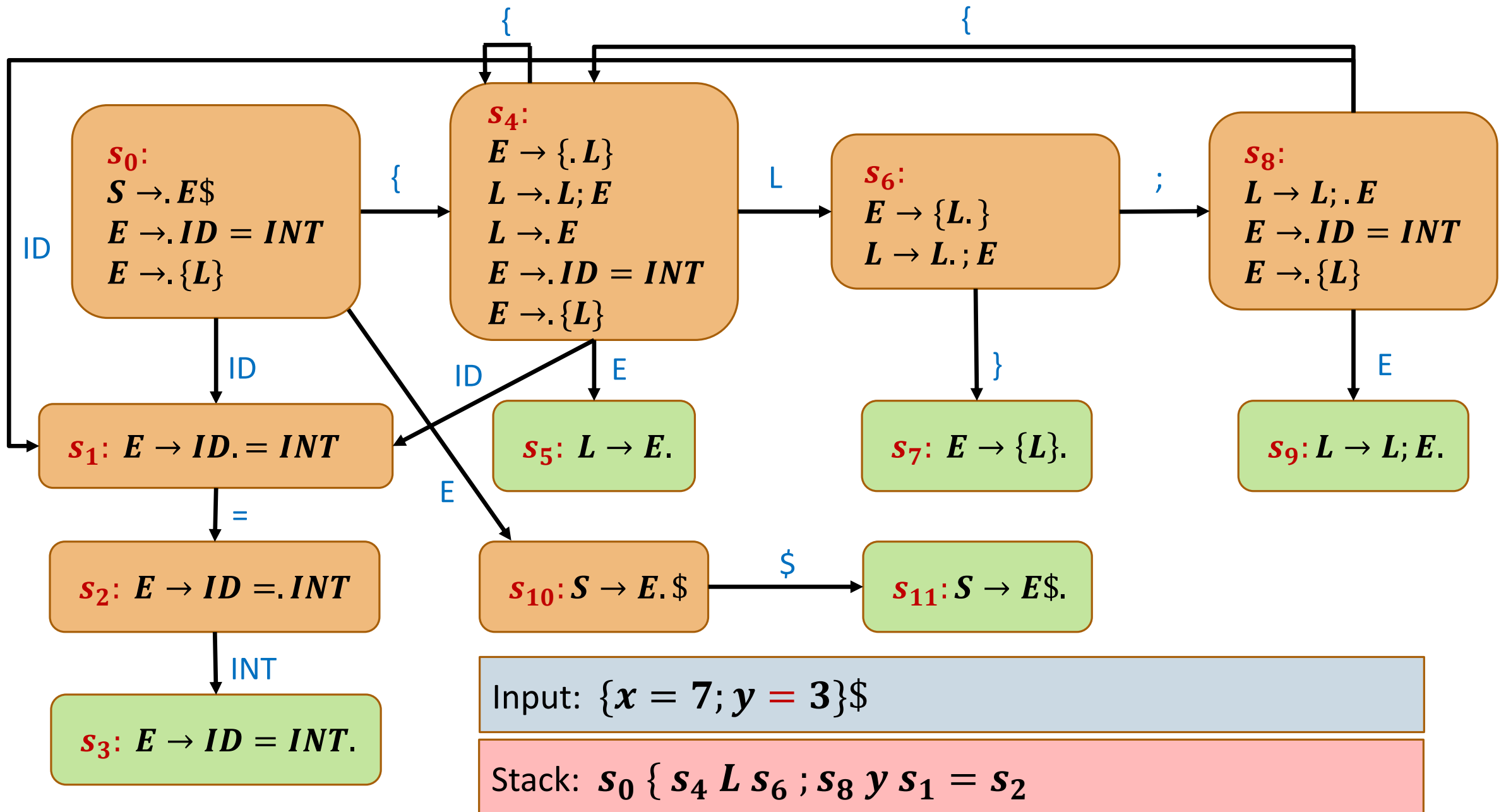


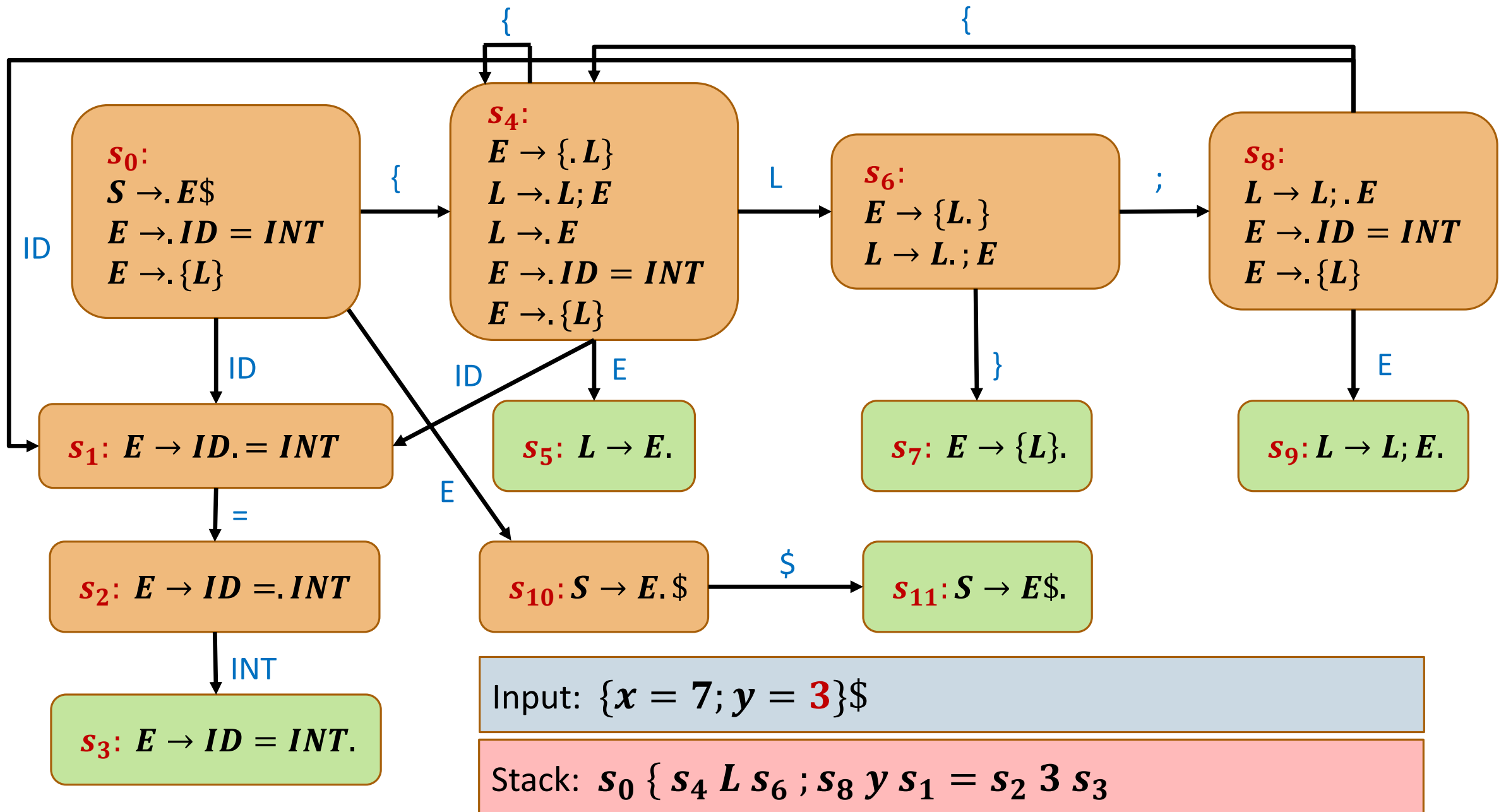


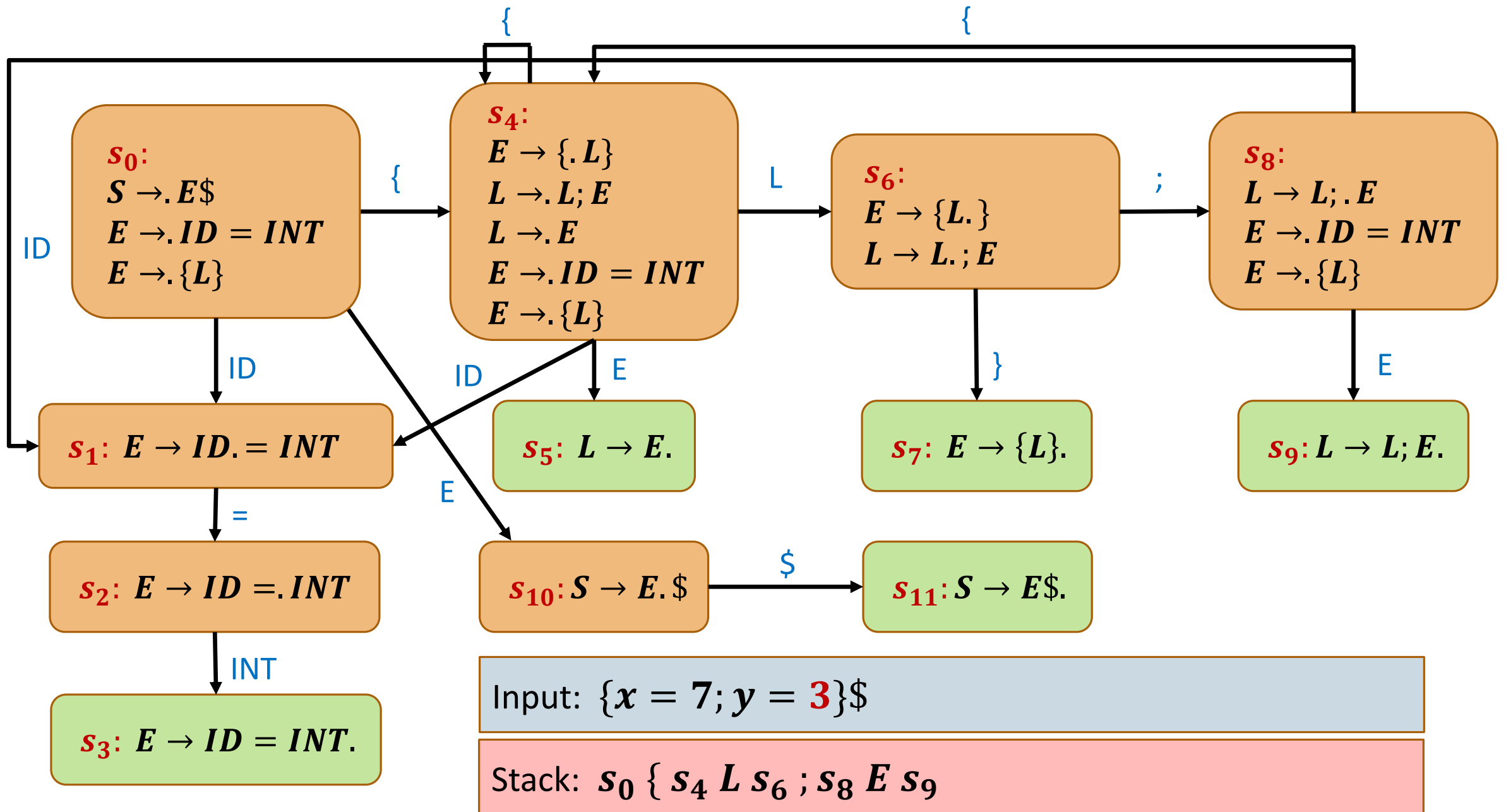


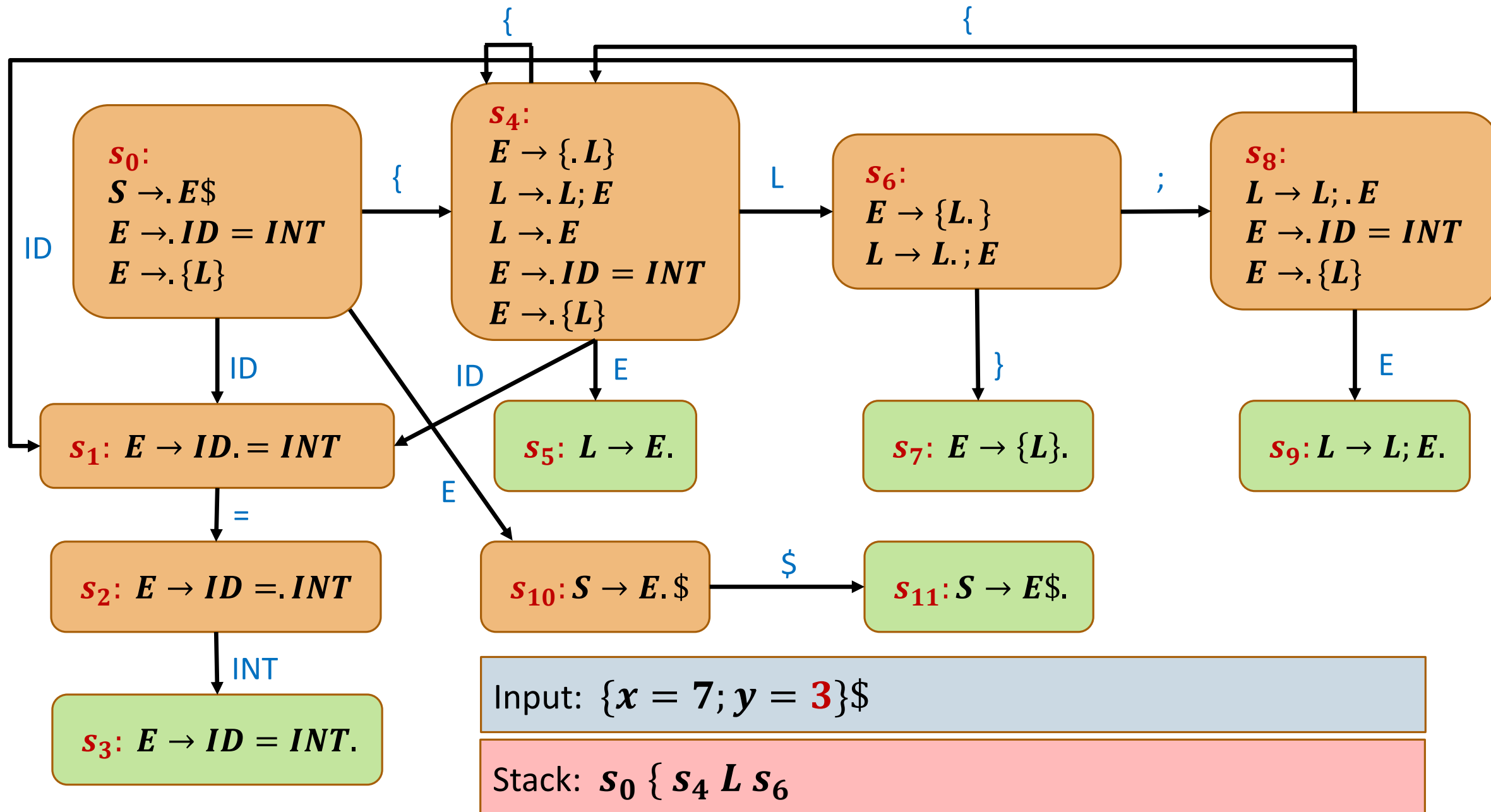


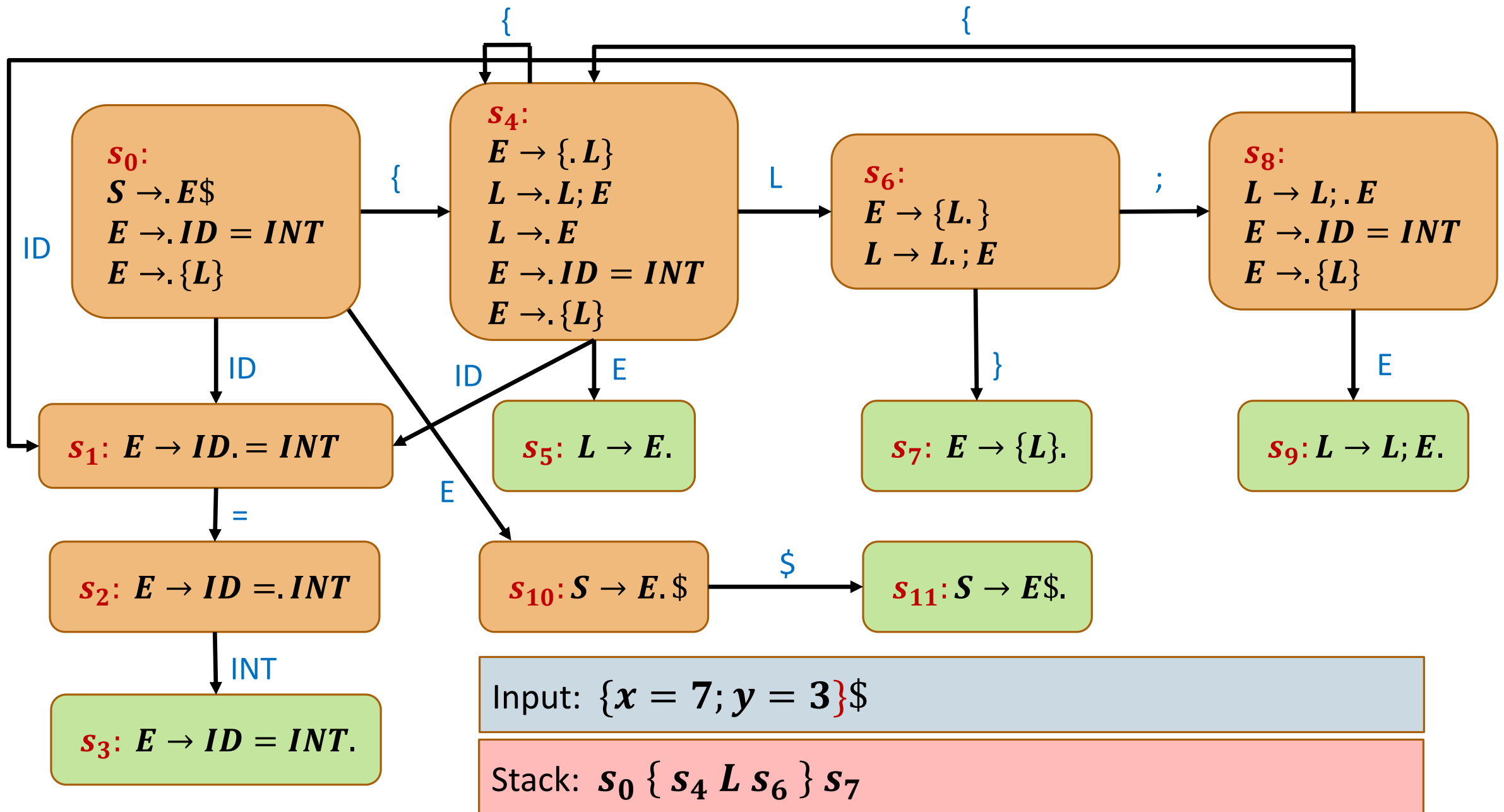


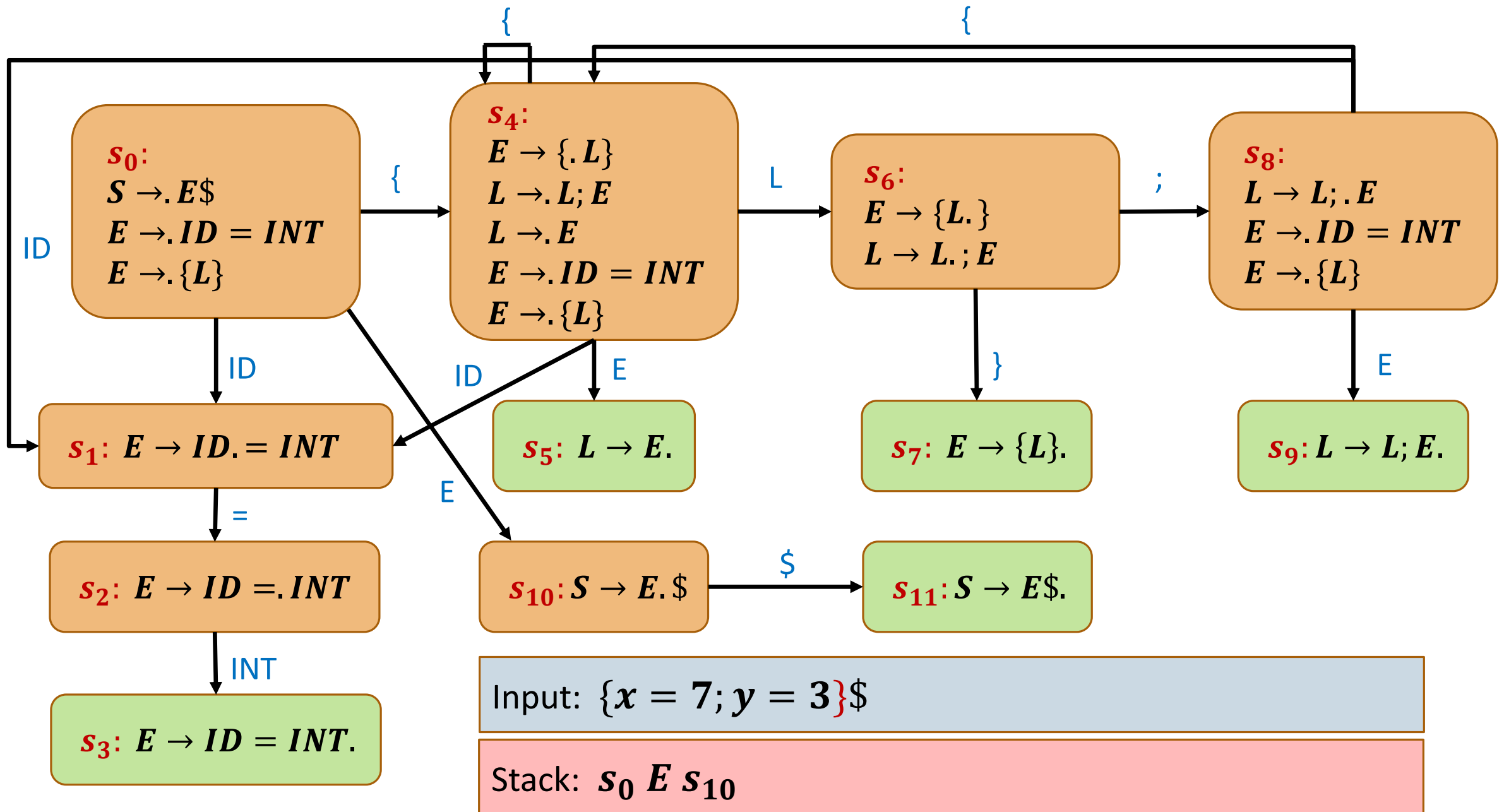




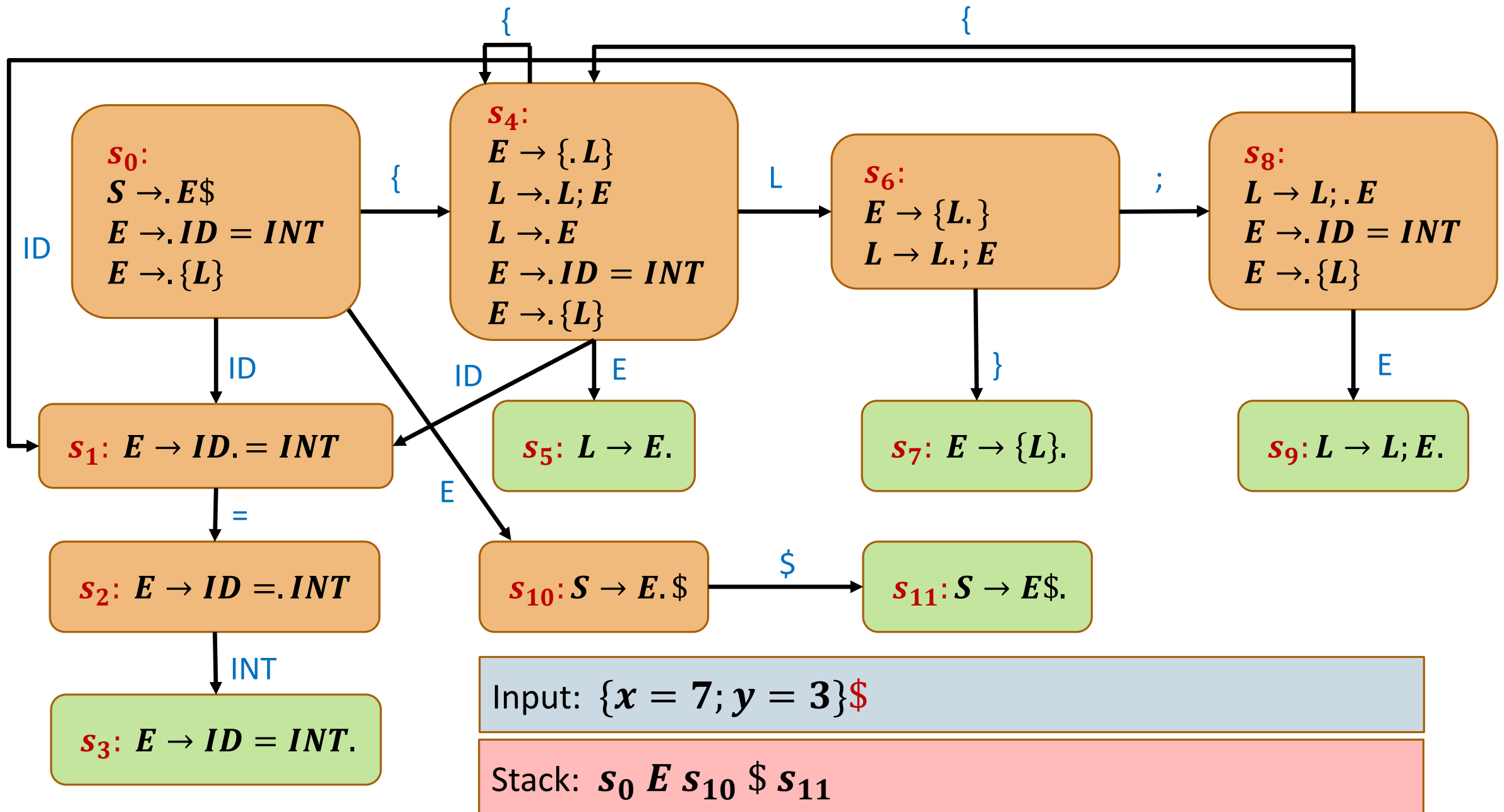


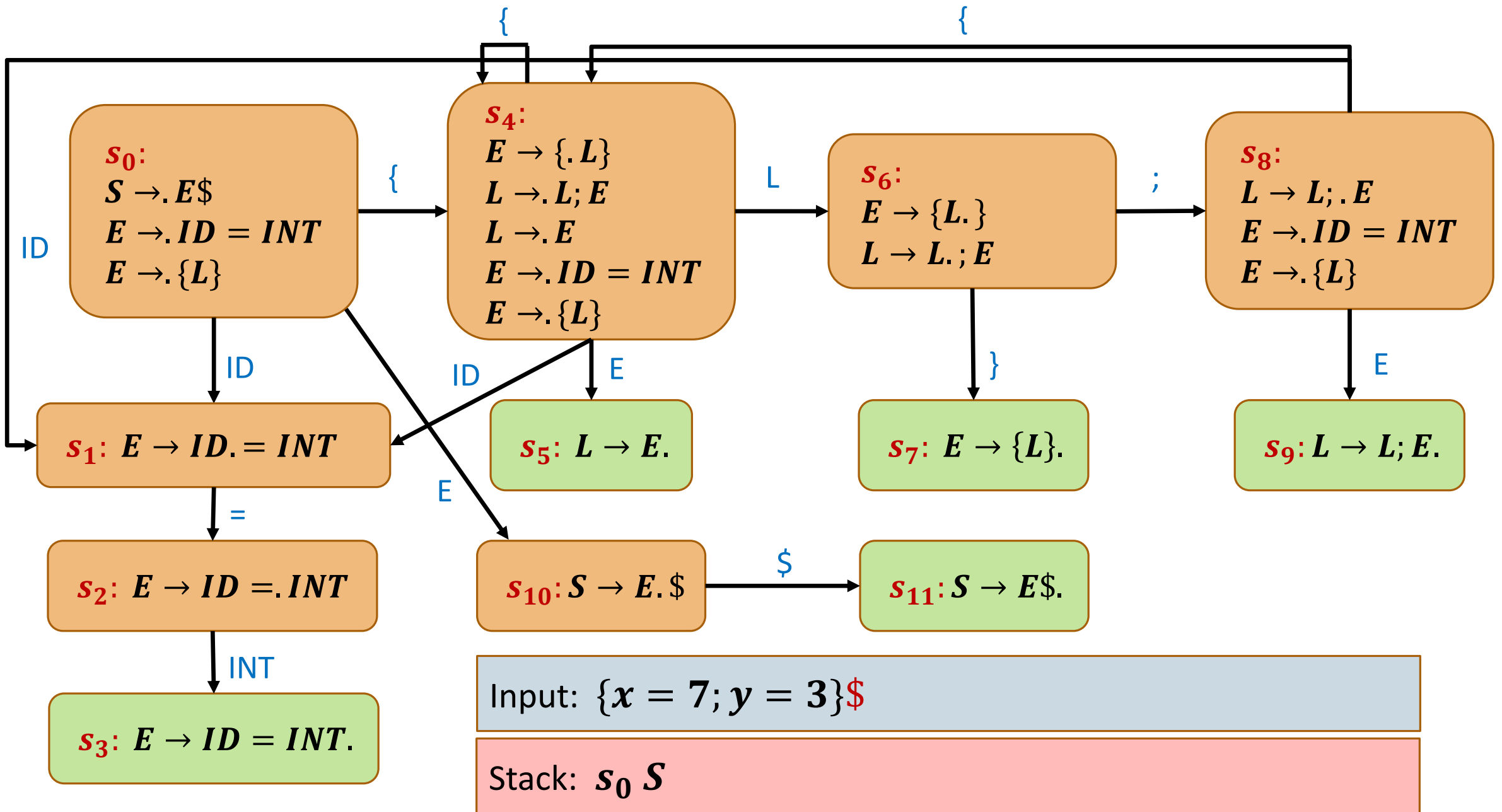












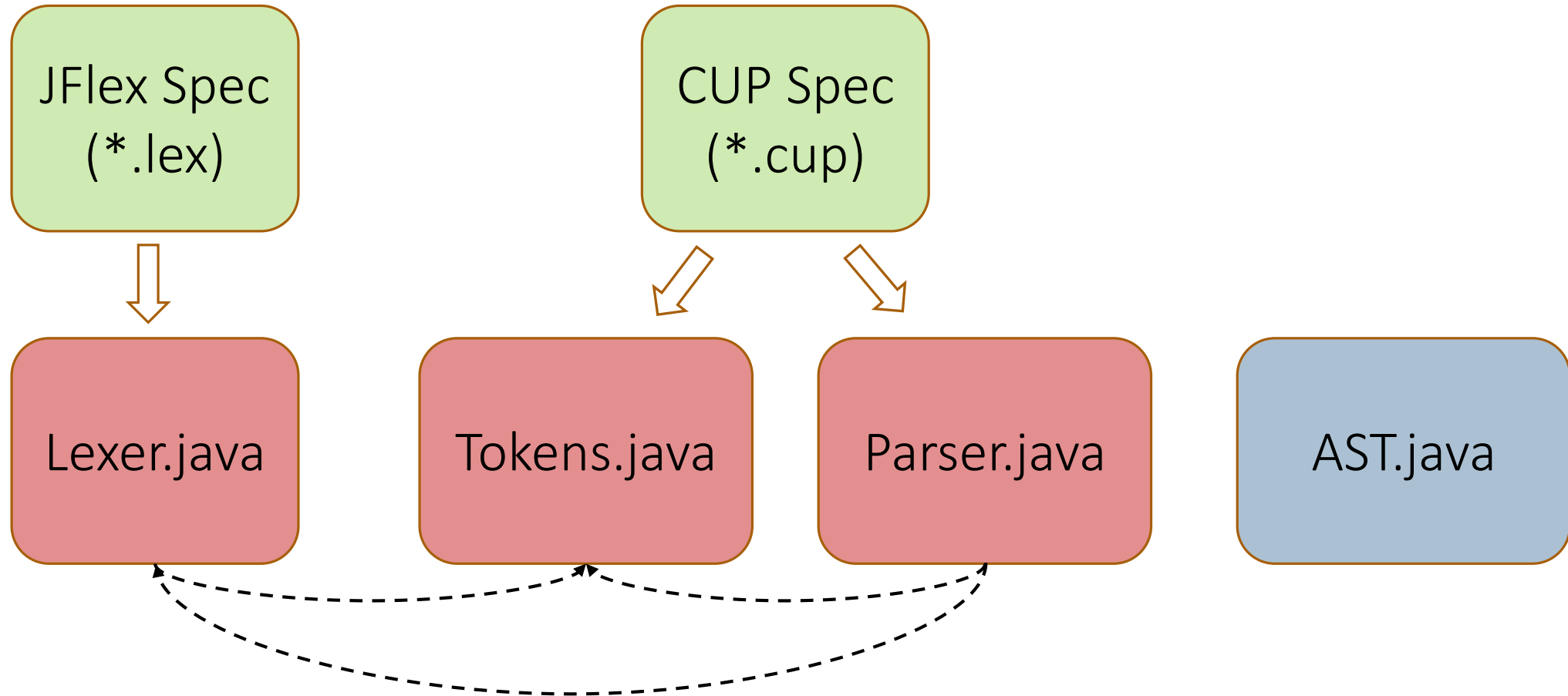
# Parsing with CUP

---

# CUP

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

# CUP/JFlex Workflow



# 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**:**e1** **E2**:**e2** { : RESULT = new **AST\_NODE\_1**(**e1**, **e2**); : } |  
**E3**:**e3** { : RESULT = new **AST\_NODE\_2**(**e3**); : } ;

**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
- For example, for integers (INT) we can do:

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

# 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 rule *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

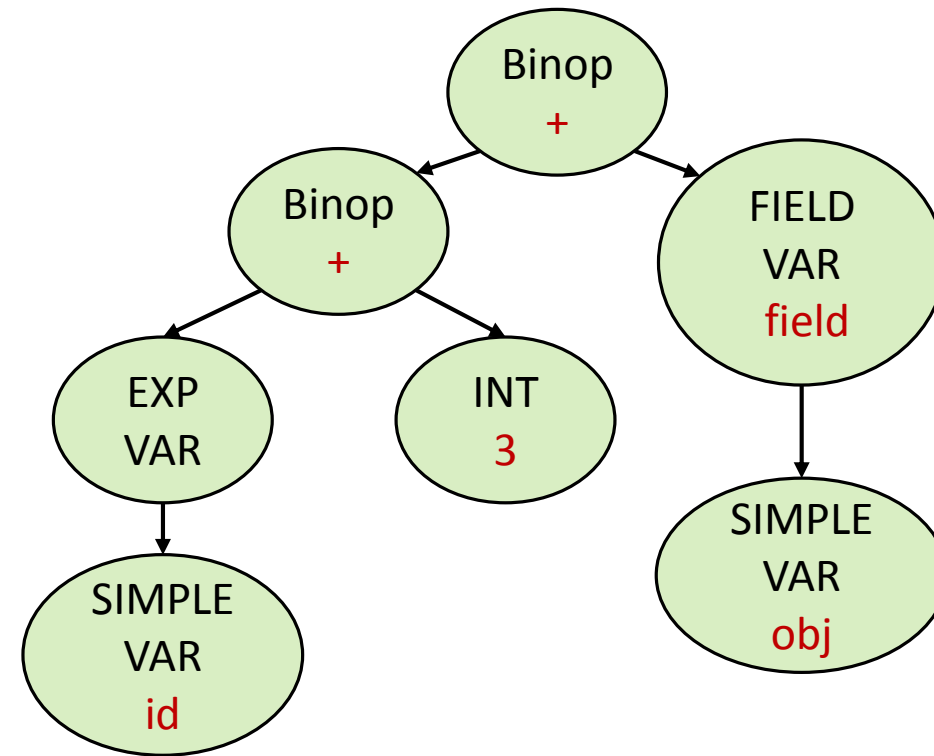
For the rule *var:v DOT ID:fieldname*:

```
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: Debugging

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

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



# CUP Example: Debugging

CUP enables to dump:

- The automaton
- The goto/action table