Compilation

0368-3133

Tutorial 4:

(More) Bottom-Up Parsing

LR(0) Parsing

S → E $

E → E + T

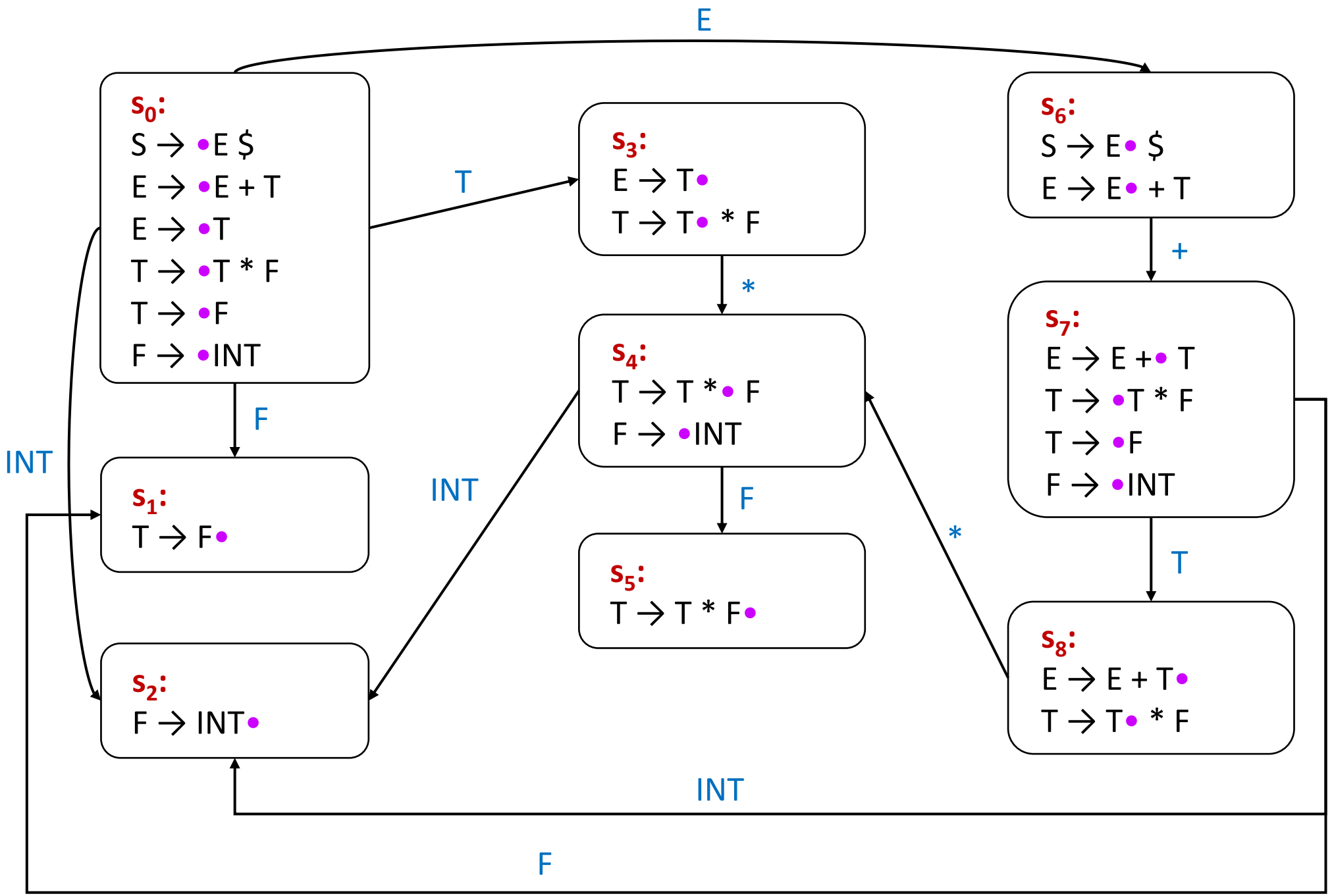
E → T

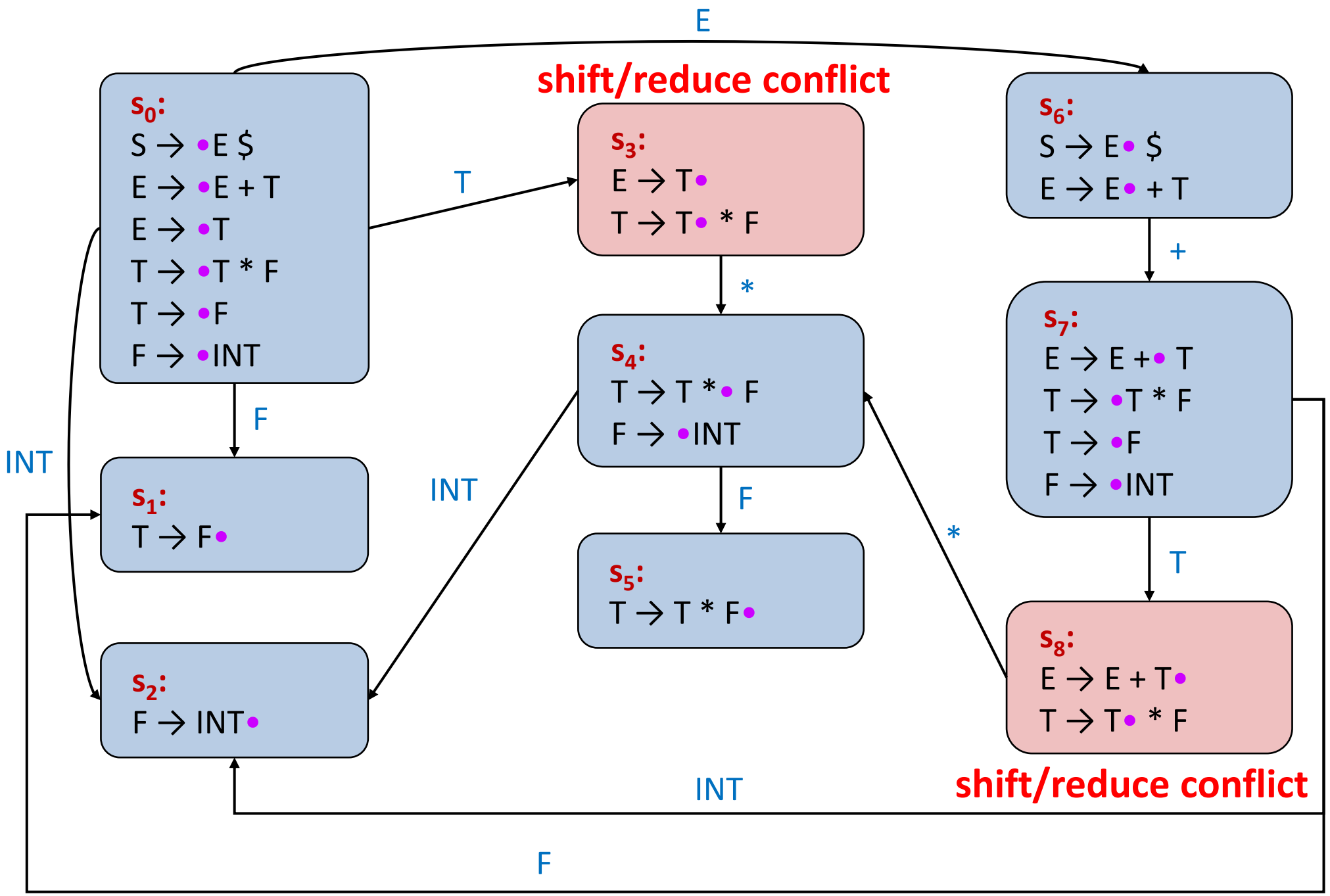
T → T * F

T → F

F → INT

- What will be the LR(0) automaton for this CFG?


E
S_0:
S → •E $
E → •E + T
E → •T
T → •T * F
T → •F
F → •INT
T
S_3:
E → T•
T → T• * F
S_6:
S → E• $
E → E• + T
+
*
S_4:
T → T *• F
F → •INT
S_7:
E → E +• T
T → •T * F
T → •F
F → •INT
F
INT
S_1:
T → F•
INT
F
S_5:
T → T * F•
*
T
S_2:
F → INT•
S_8:
E → E + T•
T → T• * F
INT
F


E
shift/reduce conflict
s_0:
S → •E $
E → •E + T
E → •T
T → •T * F
T → •F
F → •INT
T
s_3:
E → T•
T → T• * F
s_6:
S → E• $
E → E• + T
+
*
s_4:
T → T *• F
F → •INT
s_7:
E → E +• T
T → •T * F
T → •F
F → •INT
F
INT
s_1:
T → F•
INT
F
s_5:
T → T * F•
*
T
s_2:
F → INT•
s_8:
E → E + T•
T → T• * F
INT
shift/reduce conflict
F

LR(0) Conflict

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

- The conflict occurs when the next token is: *
  - Shift/reduce conflict
- E can **only** be followed by: +, $
- Taking this into account the next token can help…

s_3:
$E \rightarrow T\bullet$
$T \rightarrow T\bullet * F$

SLR(1)

- Same push-down automaton as in LR(0)
- But reduce items have a look-ahead set:
  - $A \rightarrow \alpha \bullet \{t_1, t_2, \ldots\}$
  - Where FOLLOW(A) = $\{t_1, t_2, \ldots\}$

Reminder:
Follow(A) = set of terminals (and $) that can immediately follow A in some derivation

SLR(1)

- Solves shift-reduce conflicts using the look-ahead token t
- If FOLLOW(Y) ∩ FIRST(β) = ∅ we can resolve the conflict:
  - If t ∋ FOLLOW(Y), apply the reduce
  - Otherwise, apply the shift

Y → γ• {…}
X → α •β

Reminder:
Follow(A) = set of terminals (and $) that can immediately follow A in some derivation
First(α) = all terminals that α can start with

SLR(1)

- FOLLOW(E) =

S → E $
E → E + T
E → T
T → T * F
T → F
F → INT

s_3:
E → T•
T → T• * F

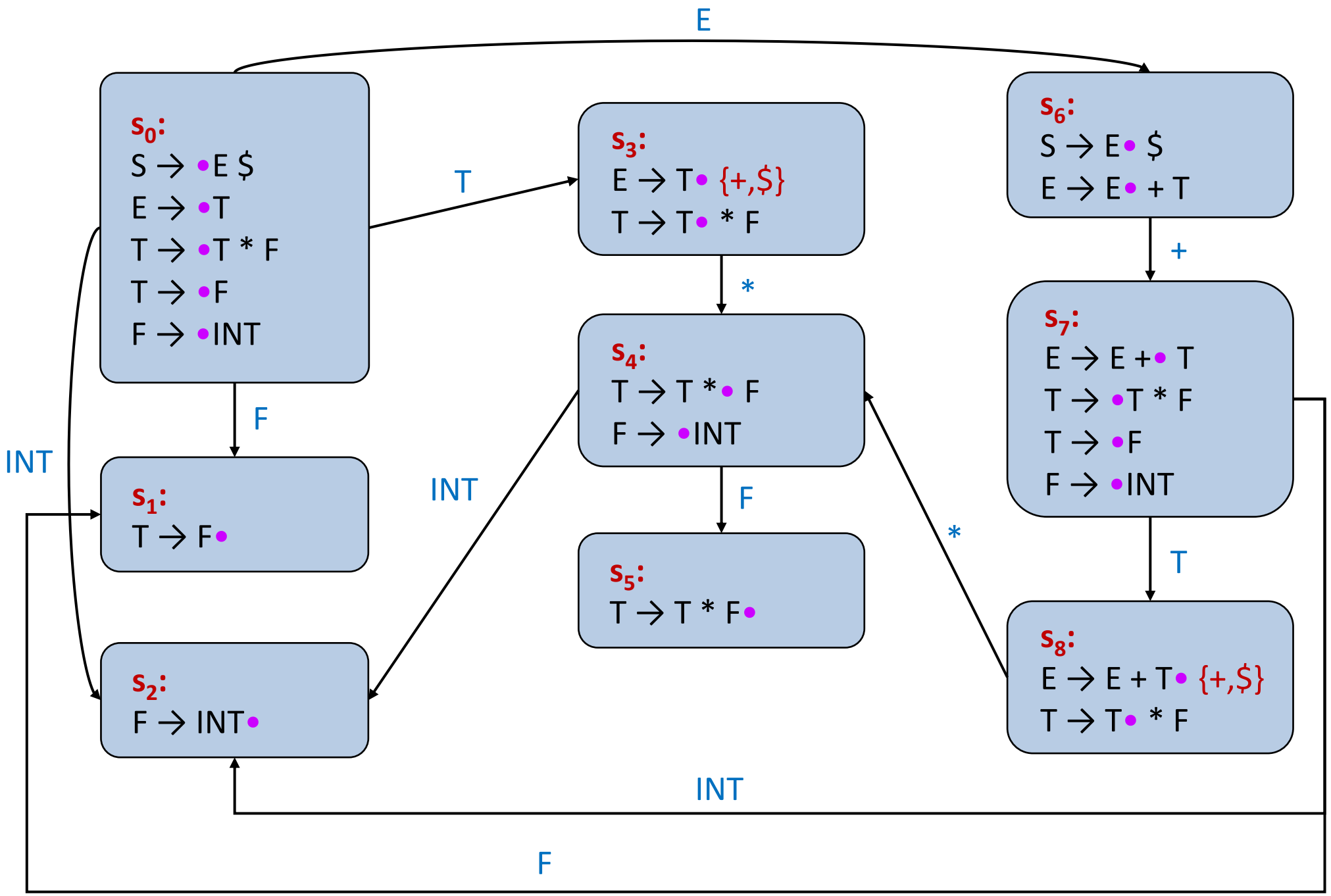
SLR(1)

- FOLLOW(E) = {+, $}
- FIRST(*) = {*}

- Can resolve the conflict ☺
  - If t ∋ {+, $} apply the **reduce**
  - Otherwise, apply the **shift**

S → E $
E → E + T
E → T
T → T * F
T → F
F → INT

s_3:
E → T•
T → T• * F


E
T
F
INT
*
+
INT
F
*
T
INT
F
s_0:
S → •E $
E → •T
T → •T * F
T → •F
F → •INT
s_1:
T → F•
s_2:
F → INT•
s_3:
E → T• {+,$}
T → T• * F
s_4:
T → T *• F
F → •INT
s_5:
T → T * F•
s_6:
S → E• $
E → E• + T
s_7:
E → E +• T
T → •T * F
T → •F
F → •INT
s_8:
E → E + T• {+,$}
T → T• * F

SLR(1) Parsing

S → E $

E → X

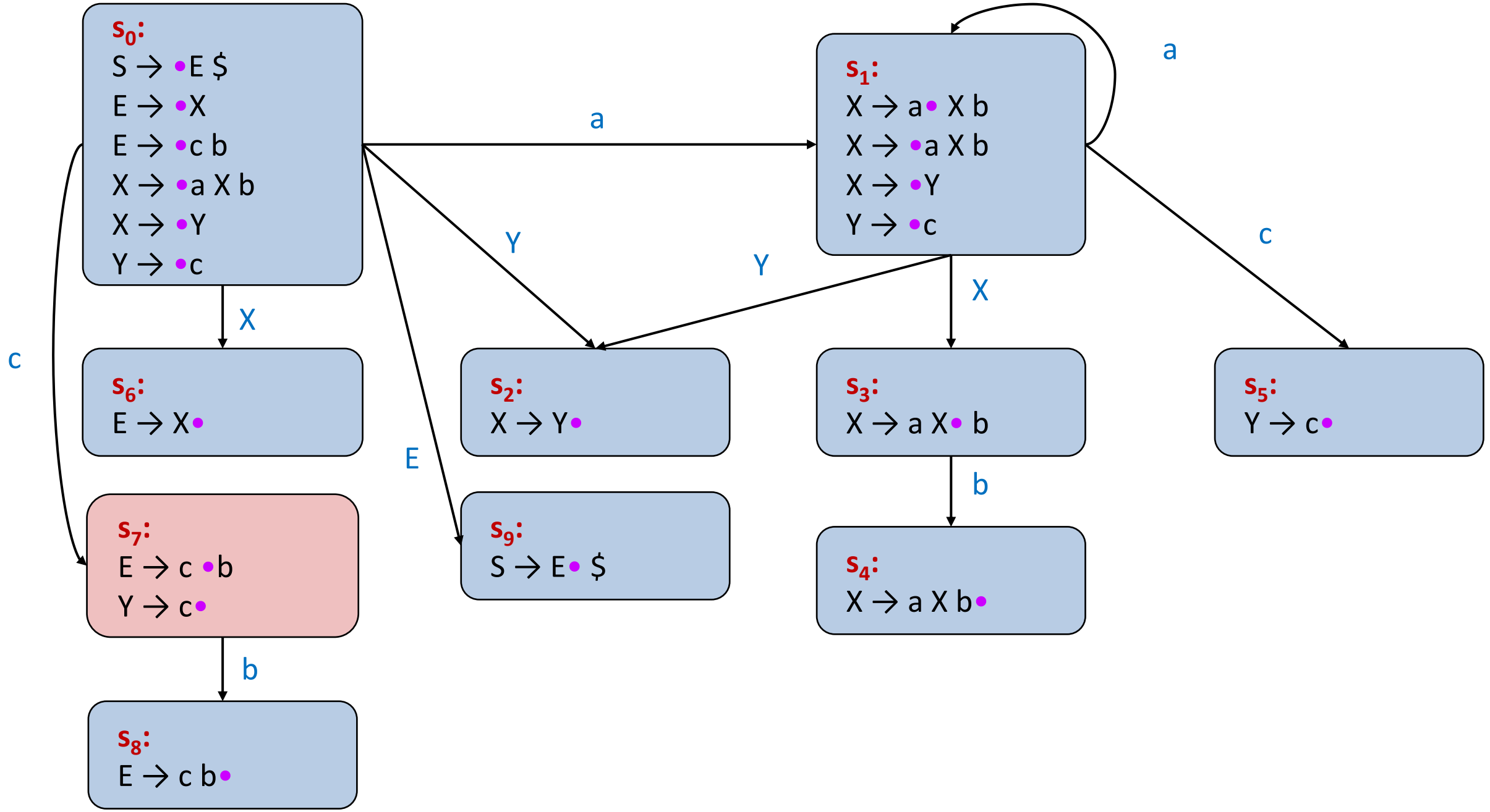
E → c b

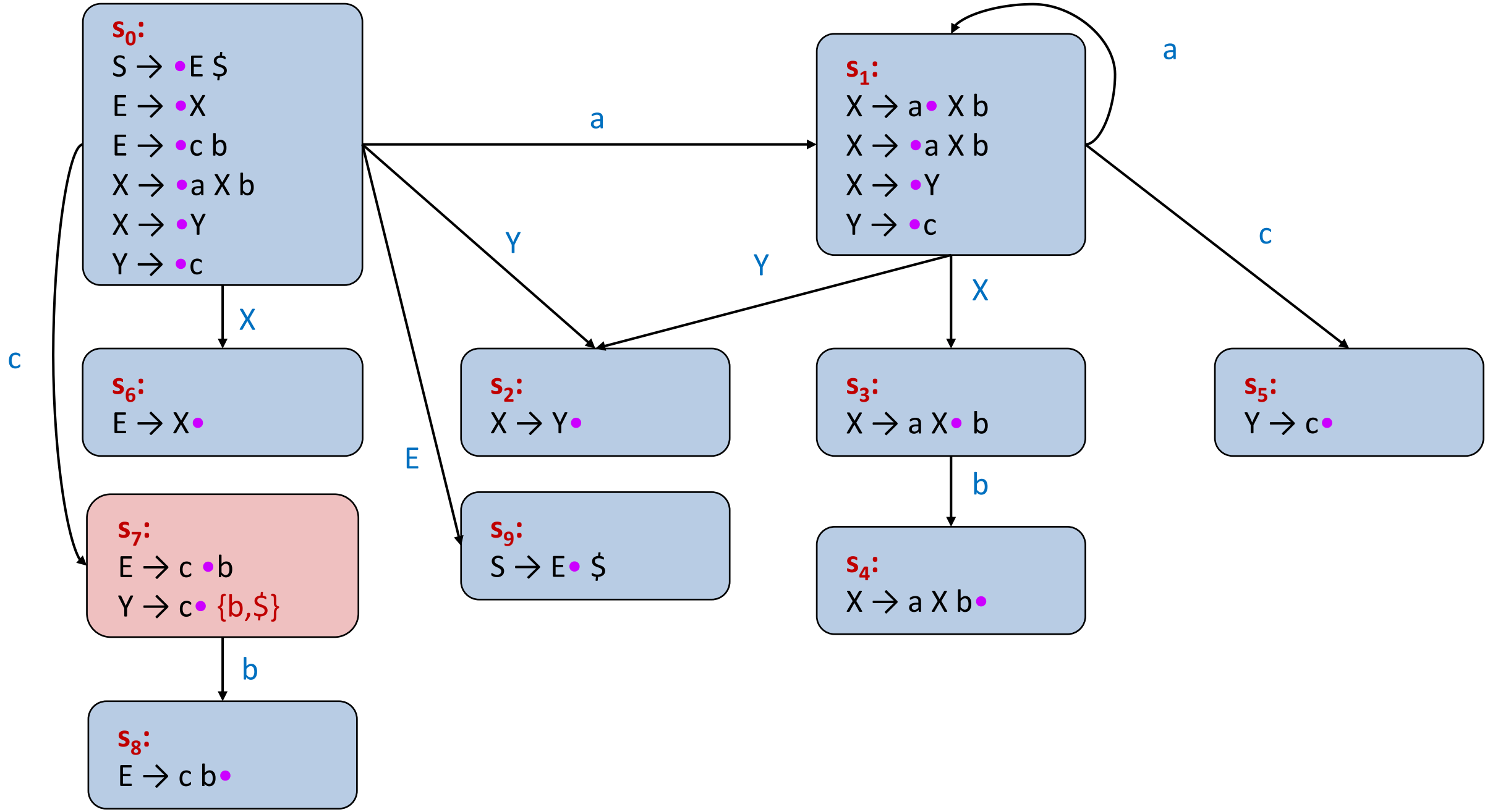
X → a X b

X → Y

Y → c

- What will be the SLR(1) automaton for this CFG?


s_0:
S → •E $
E → •X
E → •c b
X → •a X b
X → •Y
Y → •c
s_1:
X → a• X b
X → •a X b
X → •Y
Y → •c
s_2:
X → Y•
s_3:
X → a X• b
s_4:
X → a X b•
s_5:
Y → c•
s_6:
E → X•
s_7:
E → c •b
Y → c•
s_8:
E → c b•
s_9:
S → E• $
a
a
c
c
X
X
Y
Y
E
b
b


s_0:
S → •E $
E → •X
E → •c b
X → •a X b
X → •Y
Y → •c
s_1:
X → a• X b
X → •a X b
X → •Y
Y → •c
s_2:
X → Y•
s_3:
X → a X• b
s_4:
X → a X b•
s_5:
Y → c•
s_6:
E → X•
s_7:
E → c •b
Y → c• {b,$}
s_8:
E → c b•
s_9:
S → E• $
a
a
c
c
Y
Y
X
X
E
b
b

SLR(1) Conflict

- The conflict occurs when the next token is b
- Relying on FOLLOW(Y) considers all occurrences of Y in **all the states**
- **Three derivation options:**
  - S → E → cb
  - S → E → X → Y → c
  - S → E → X → a X b → …
    →a..a Y b…b → a..a c b…b

s_7:
E → c •b
Y → c• {b,$}

S → E $
E → X
E → c b
X → a X b
X → Y
Y → c

LR(1)

- Maintains items with more precise look-ahead sets
- An ***LR(1) item*** is of the form:
  - $A \rightarrow \alpha \bullet \beta \{\sigma\}$
  - Where $\sigma = t_1, t_2, \ldots$ (terminals)

LR(1) Item Closure Set

Given an LR(1) item I we define its *closure set* S inductively as follows:

- Base: $I \ni S$
- Inductive step: If $A \to \alpha \bullet B\beta \{\sigma\} \ni S$ then

  for each rule $B \to \gamma$ also $B \to \bullet\gamma \{\tau\} \ni S$

  where $\tau = \text{FIRST}(\beta,\sigma)$

$$\text{First}(\beta,\sigma) = \begin{cases} \text{First}(\beta) & \beta \text{ is not nullable} \\ (\text{First}(\beta) \cup \{\sigma\}) \setminus \{\epsilon\} & \text{otherwise} \end{cases}$$

LR(1) Parsing

S → E $

E → X

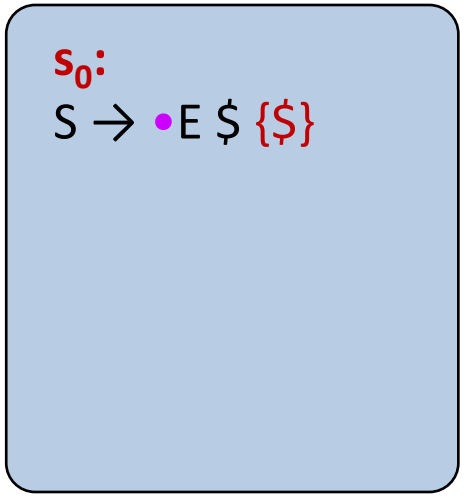
E → c b

X → a X b

X → Y

Y → c

- What will be the LR(1) automaton for this CFG?


s_0:
S → •E $ {$}

s_0:

S → •E $ {$}

E → •X

E → •c b

s_0:

S → •E $ {$}

E → •X {$}

E → •c b {$}

s_0:

S → •E $ {$}

E → •X {$}

E → •c b {$}

X → •a X b

X → •Y

s_0:

S → •E $ {$}

E → •X {$}

E → •c b {$}

X → •a X b {$}

X → •Y {$}

s_0:

S → •E $ {$}

E → •X {$}

E → •c b {$}

X → •a X b {$}

X → •Y {$}

Y → •c

s_0:

S → •E $ {$}

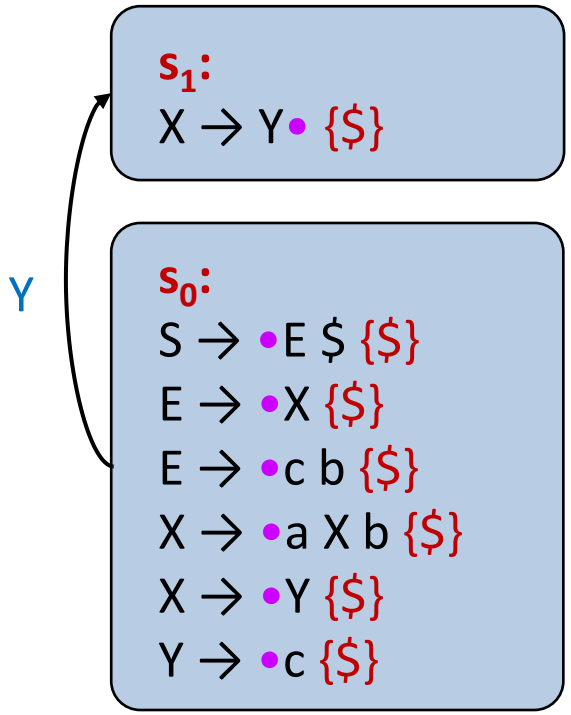
E → •X {$}

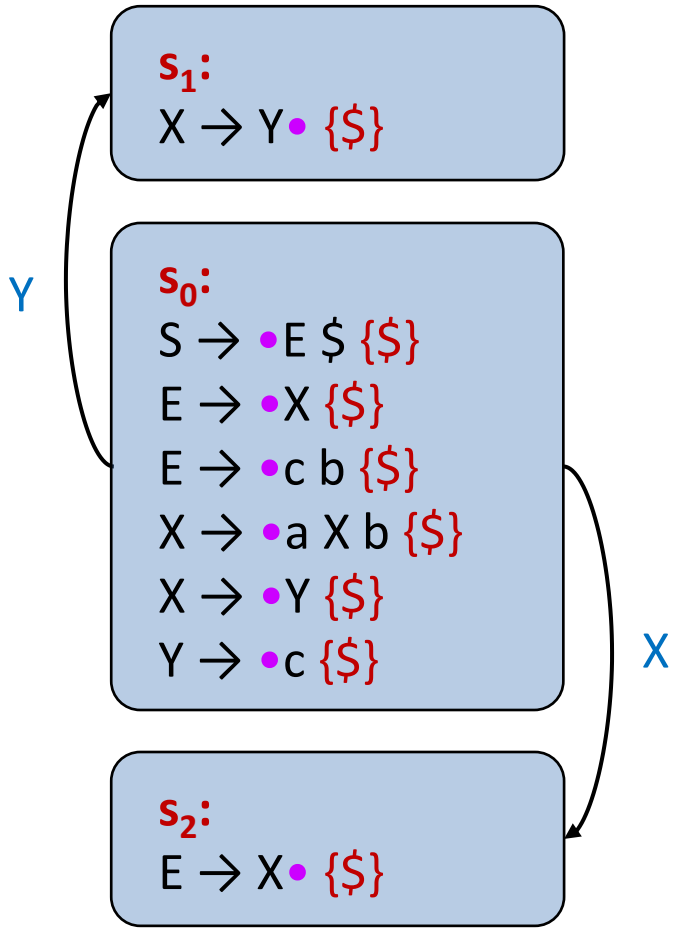
E → •c b {$}

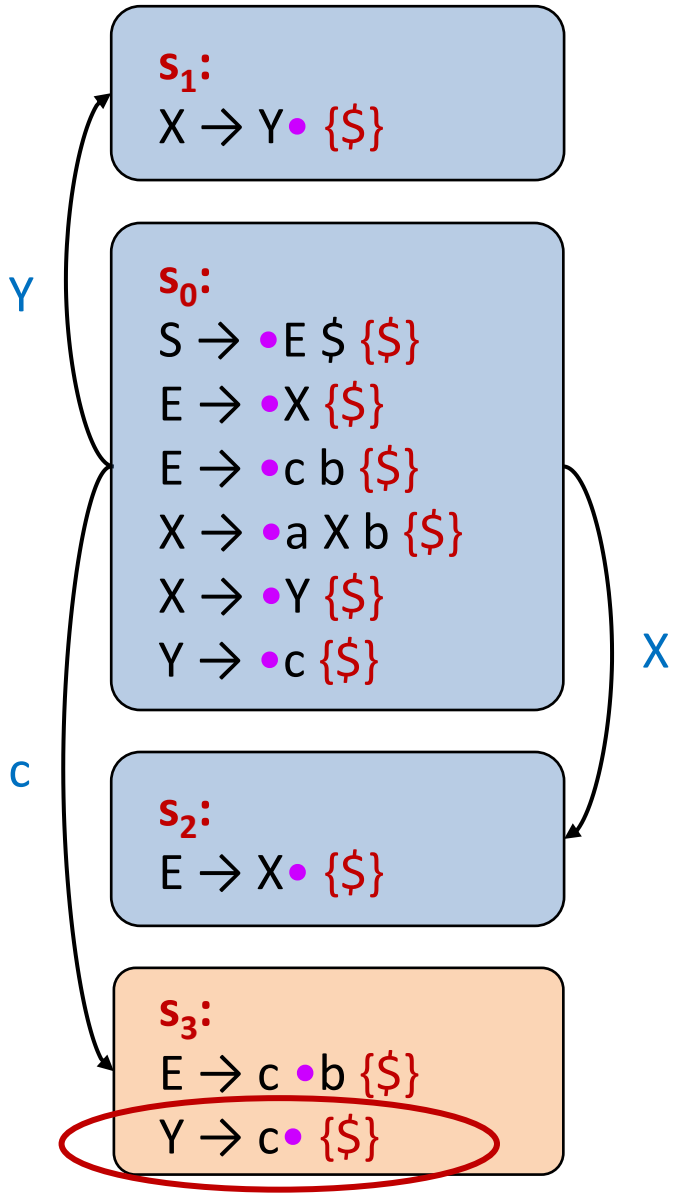
X → •a X b {$}

X → •Y {$}

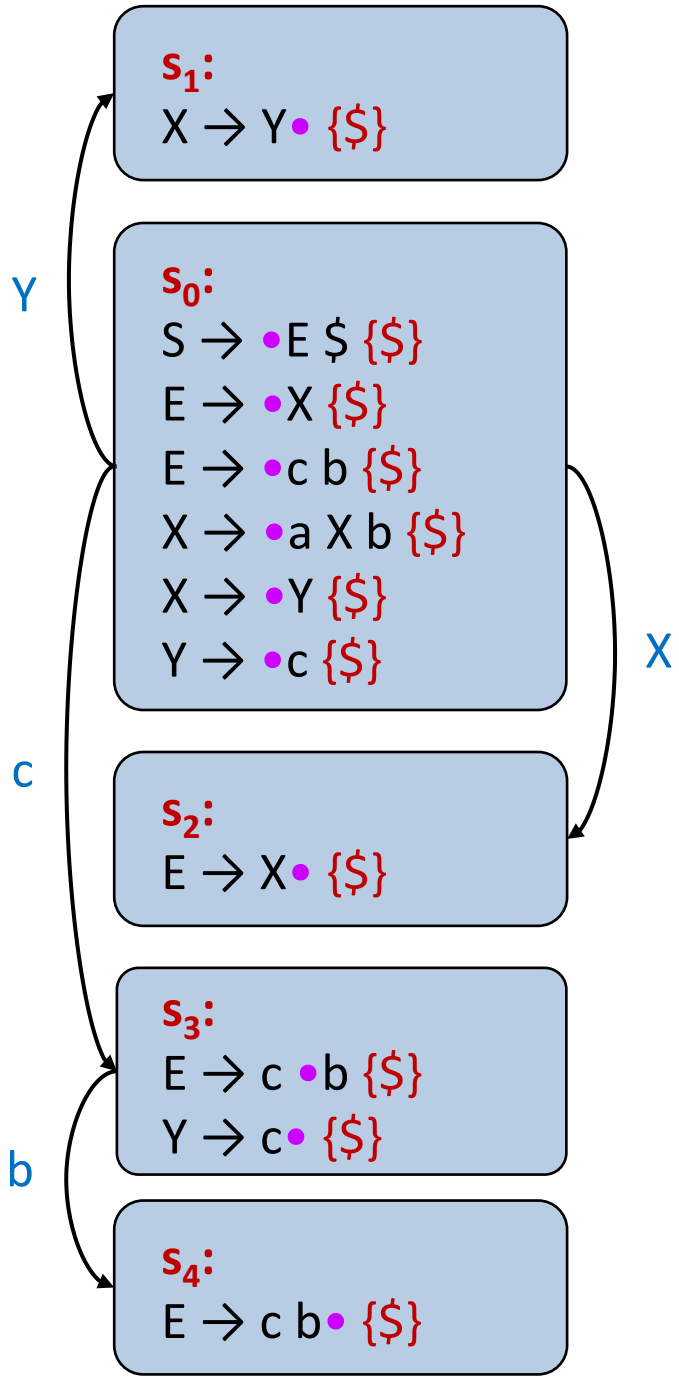
Y → •c {$}

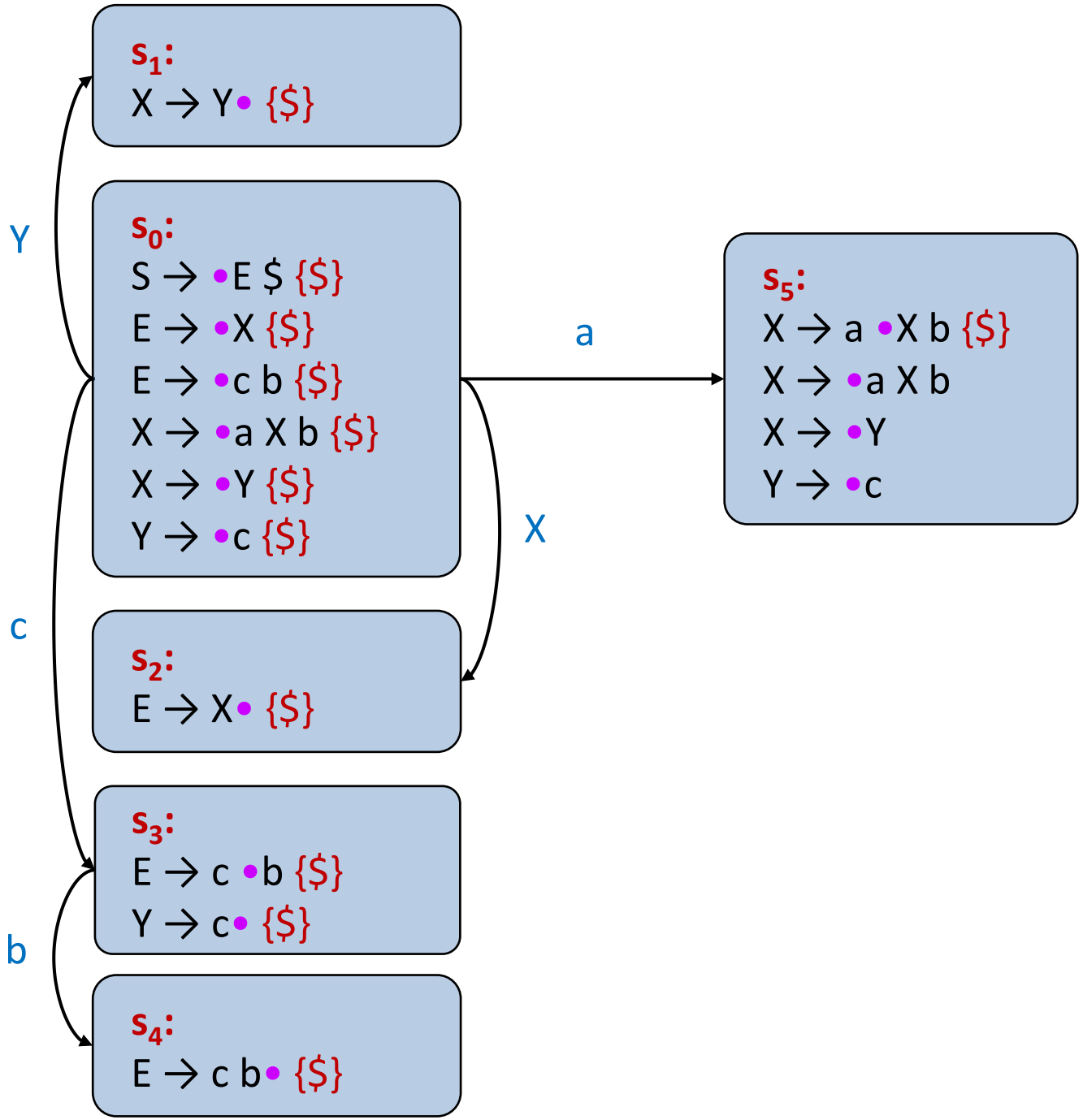

s_1:
X → Y• {$}
Y
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}

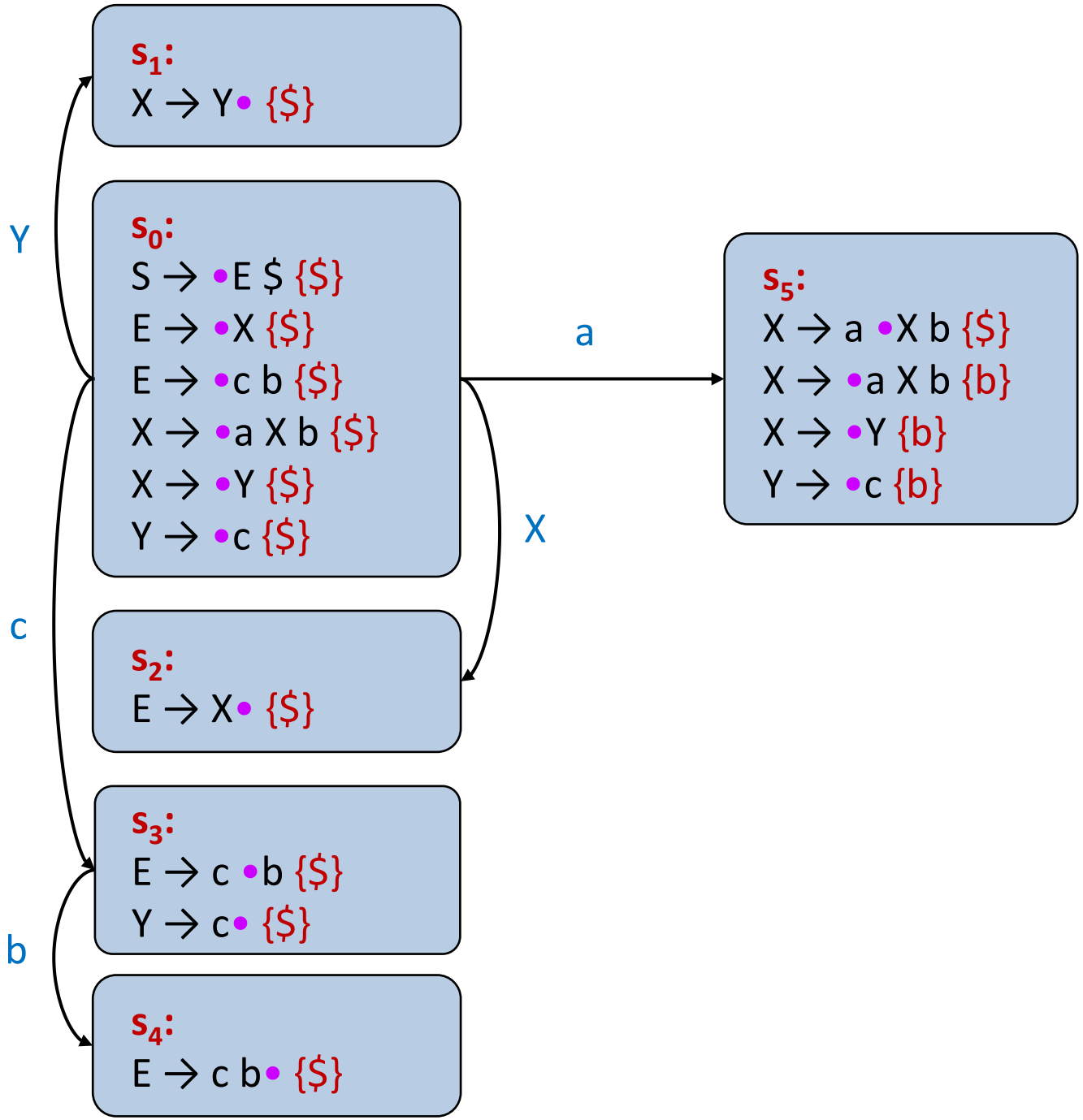

s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_2:
E → X• {$}
Y
X

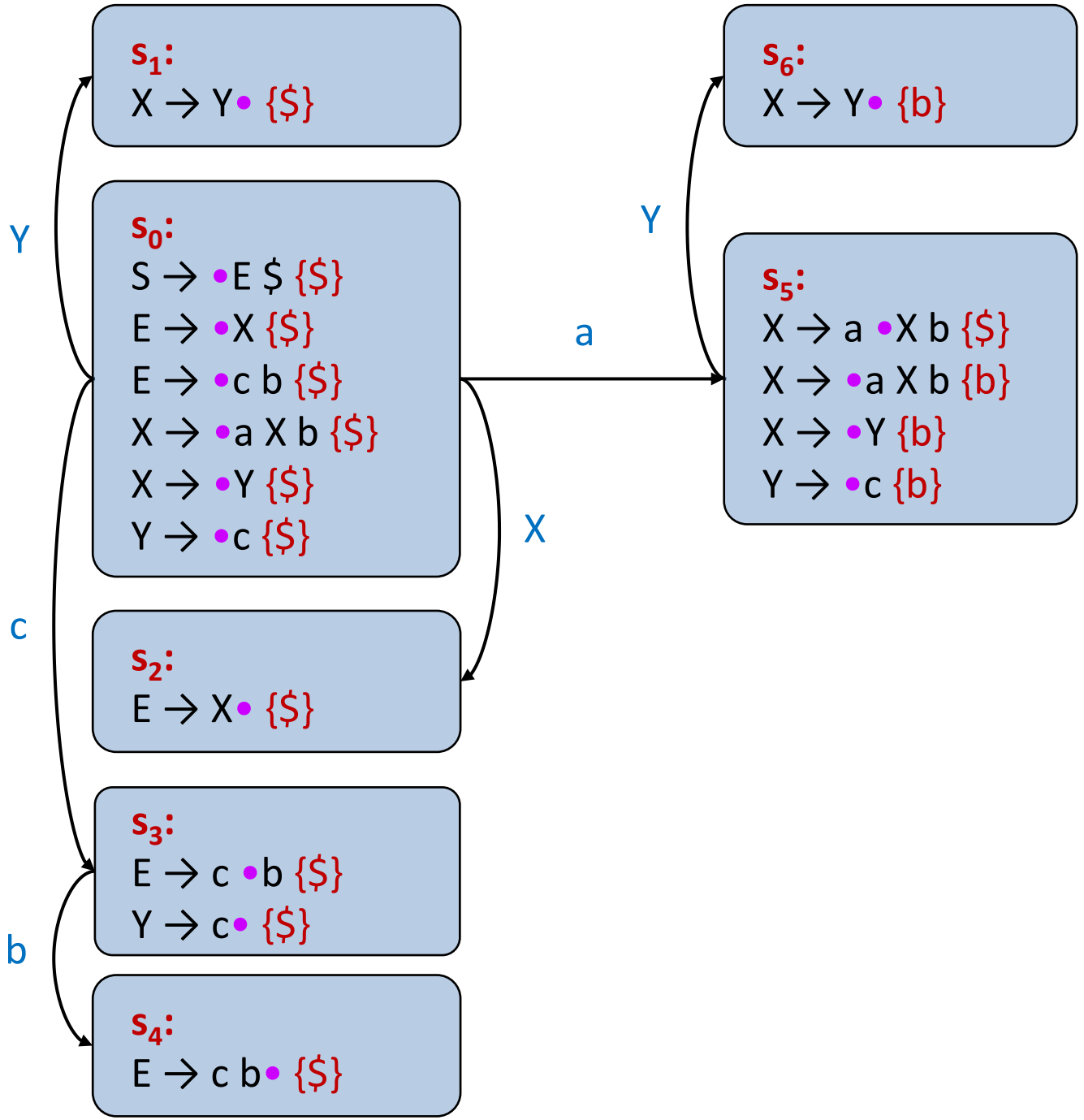


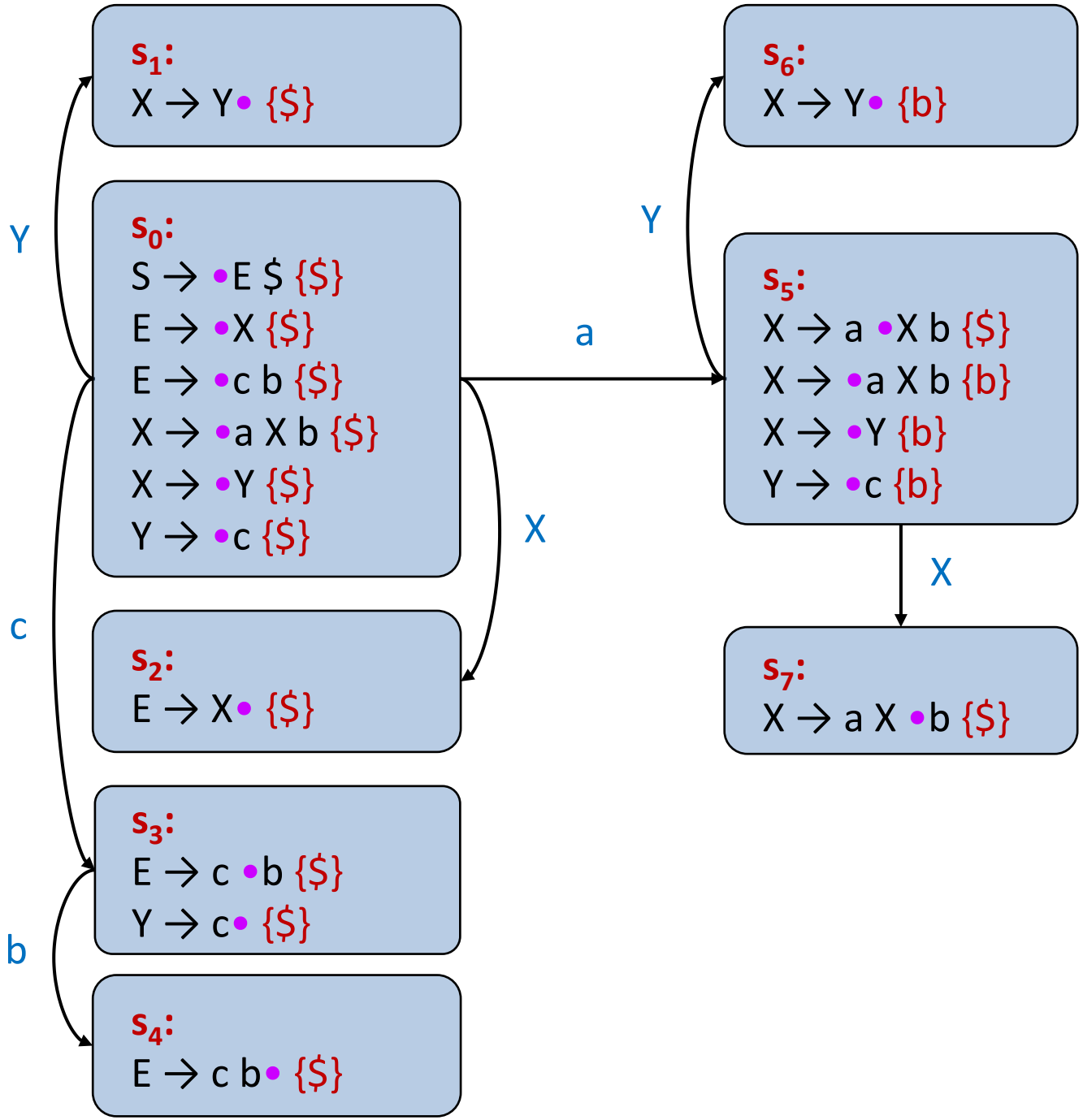
In SLR(1) we had: {b,$}

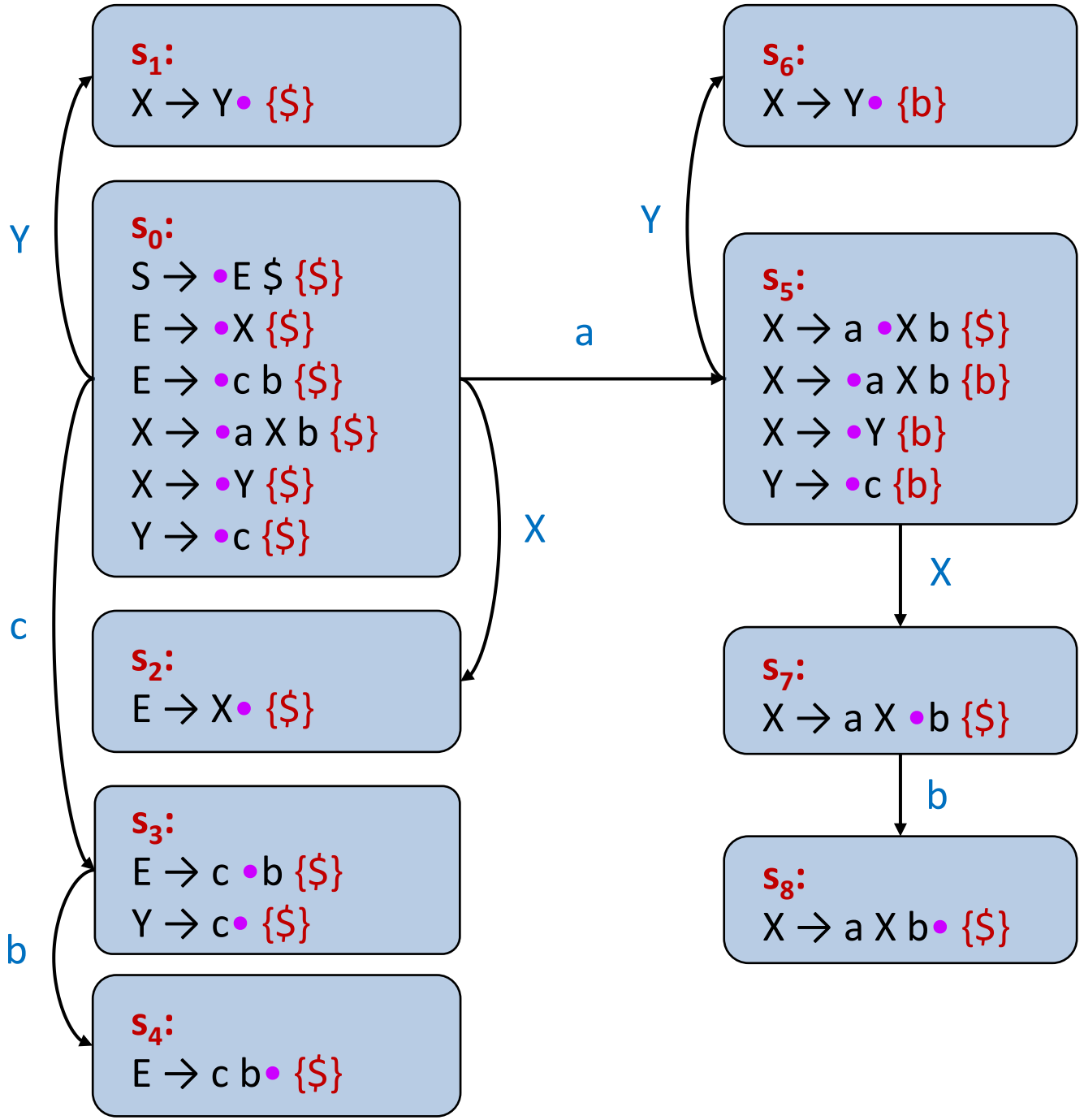

s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_2:
E → X• {$}
s_3:
E → c •b {$}
Y → c• {$}
s_4:
E → c b• {$}
Y
X
c
b

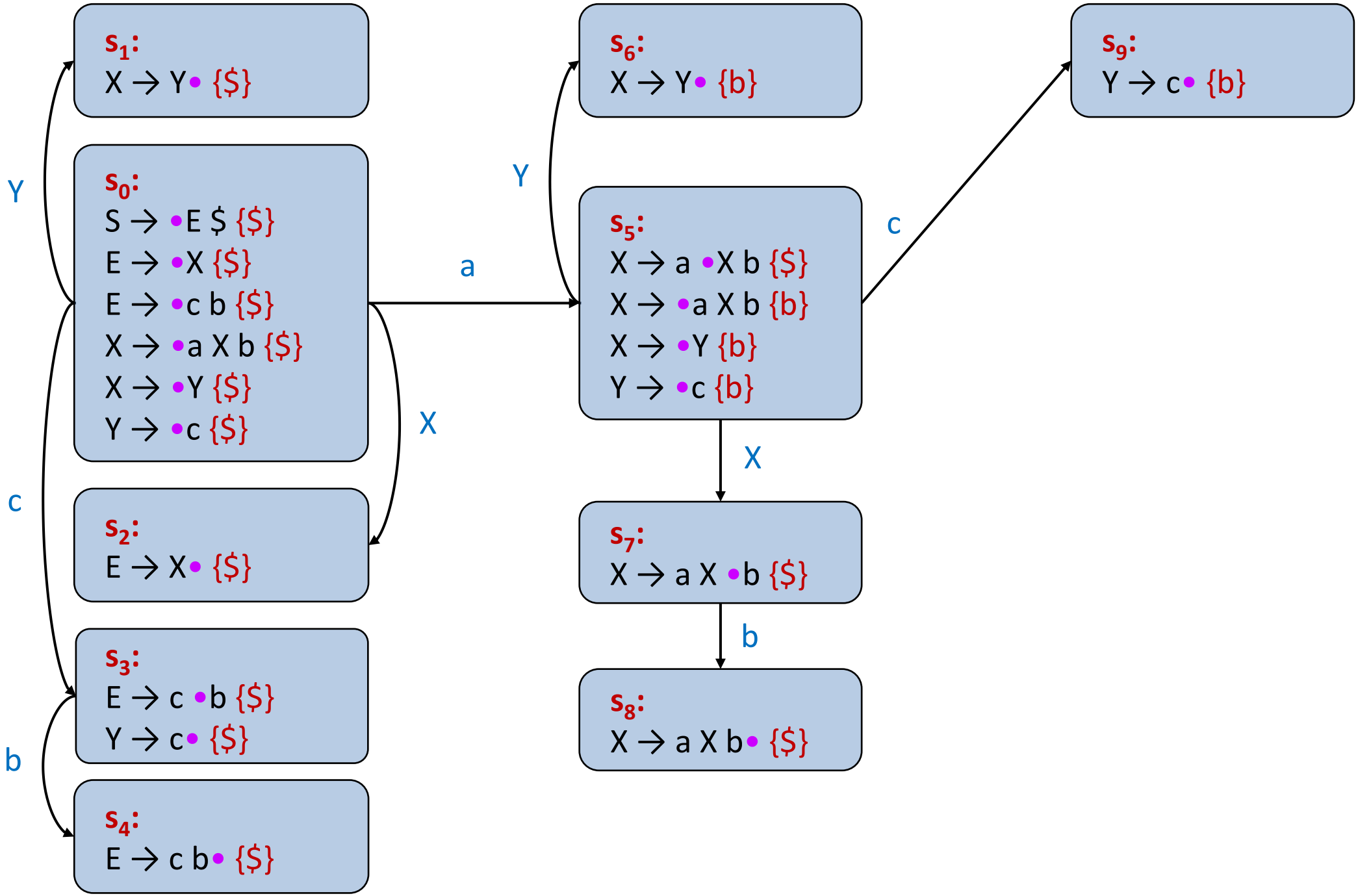

s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_5:
X → a •X b {$}
X → •a X b
X → •Y
Y → •c
s_2:
E → X• {$}
s_3:
E → c •b {$}
Y → c• {$}
s_4:
E → c b• {$}
Y
a
X
c
b

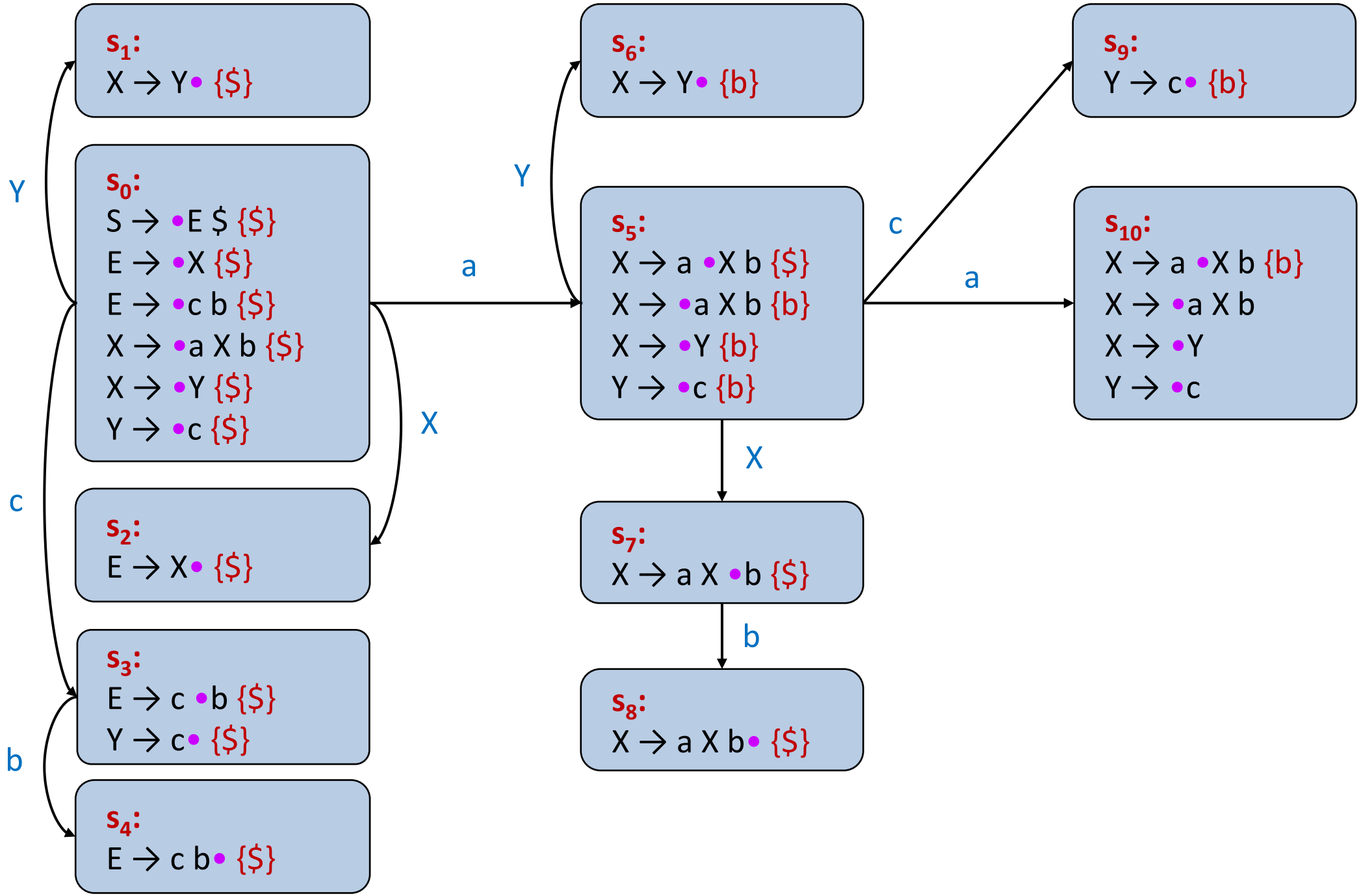

s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s_2:
E → X• {$}
s_3:
E → c •b {$}
Y → c• {$}
s_4:
E → c b• {$}
Y
a
X
c
b

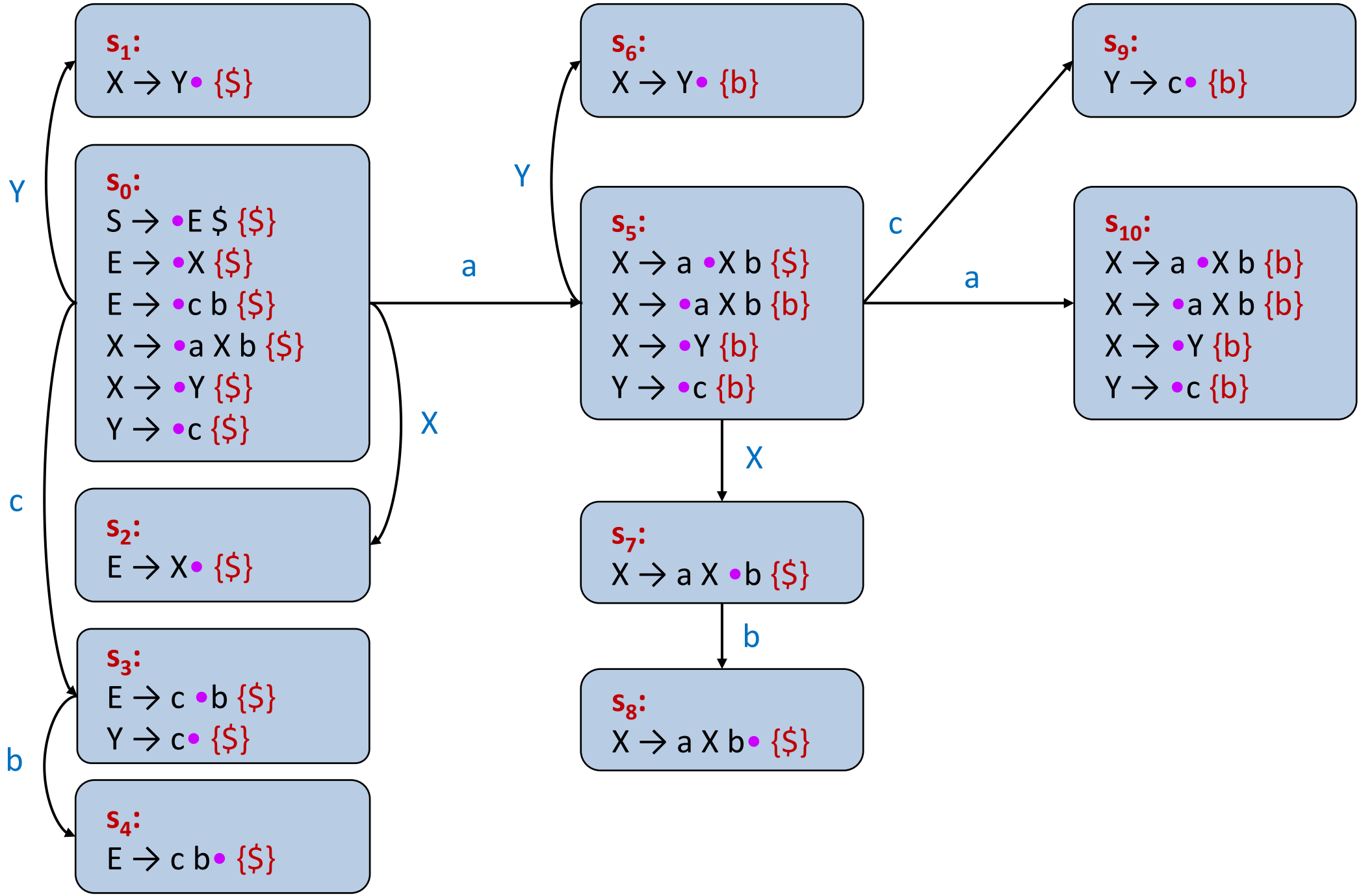

s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_6:
X → Y• {b}
s_5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s_2:
E → X• {$}
s_3:
E → c •b {$}
Y → c• {$}
s_4:
E → c b• {$}
Y
Y
a
X
c
b

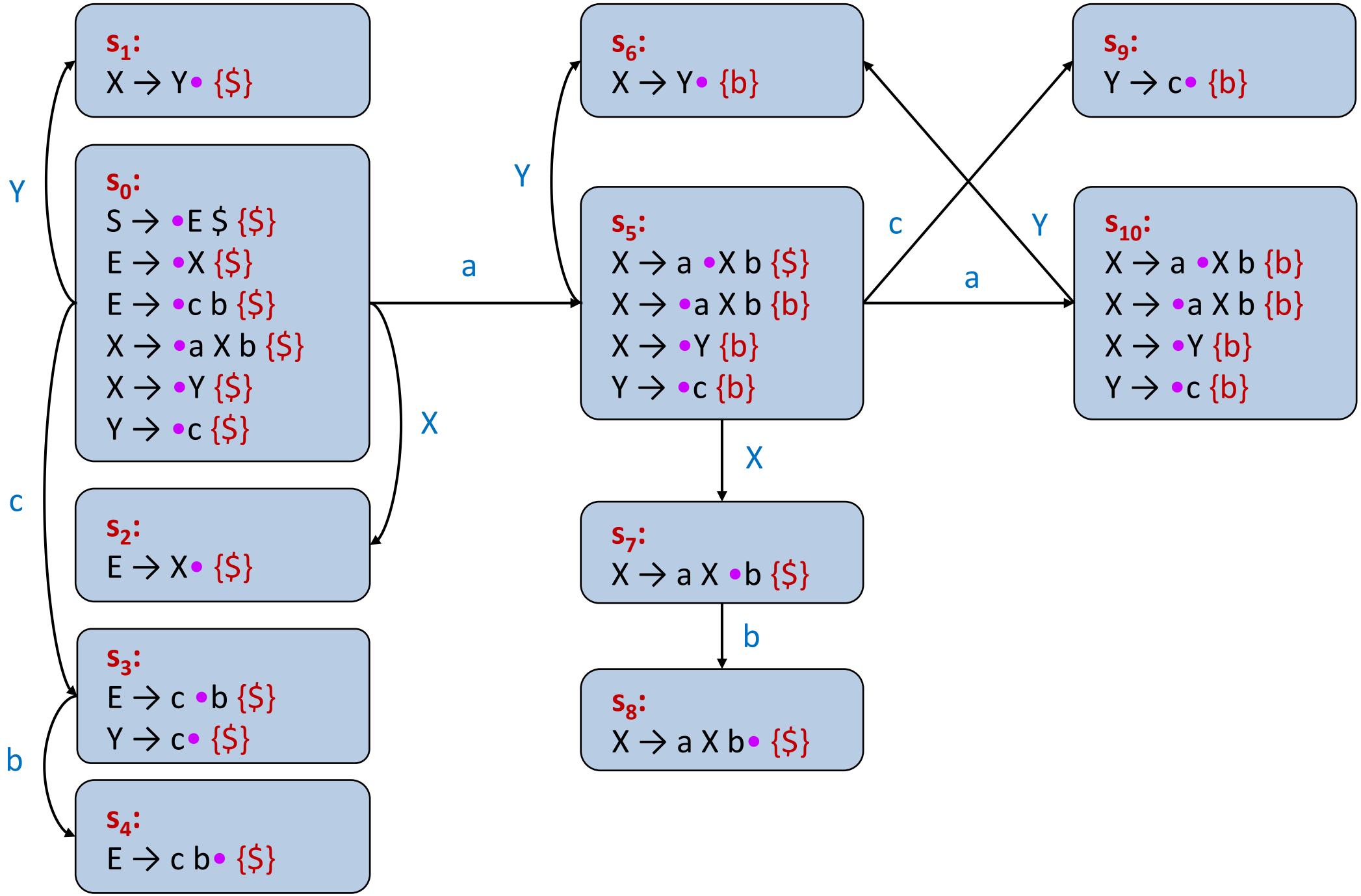

s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_2:
E → X• {$}
s_3:
E → c •b {$}
Y → c• {$}
s_4:
E → c b• {$}
s_6:
X → Y• {b}
s_5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s_7:
X → a X •b {$}
Y
Y
a
X
c
X
b

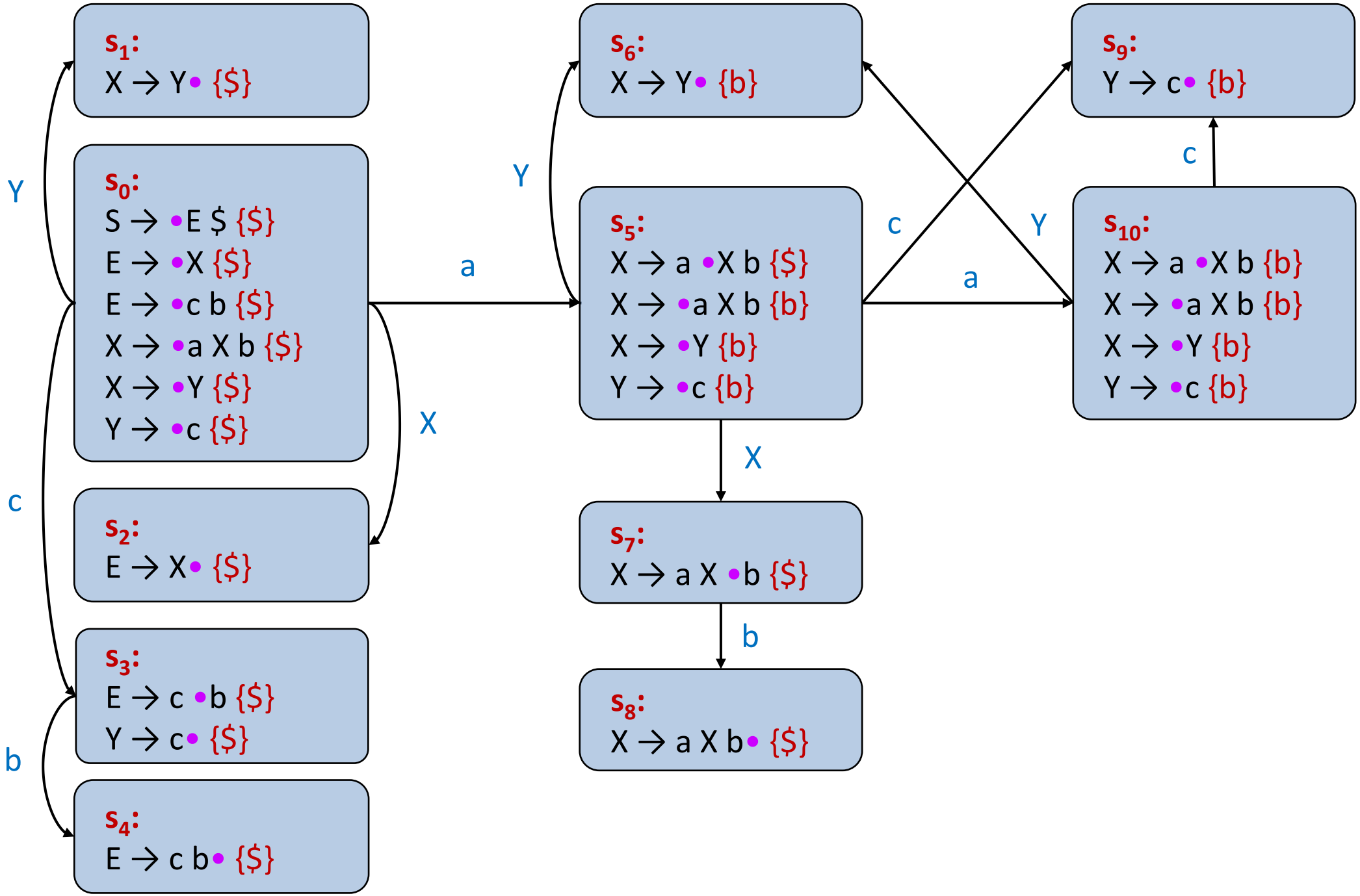

s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_2:
E → X• {$}
s_3:
E → c •b {$}
Y → c• {$}
s_4:
E → c b• {$}
s_6:
X → Y• {b}
s_5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s_7:
X → a X •b {$}
s_8:
X → a X b• {$}
Y
Y
a
X
c
b
X
b

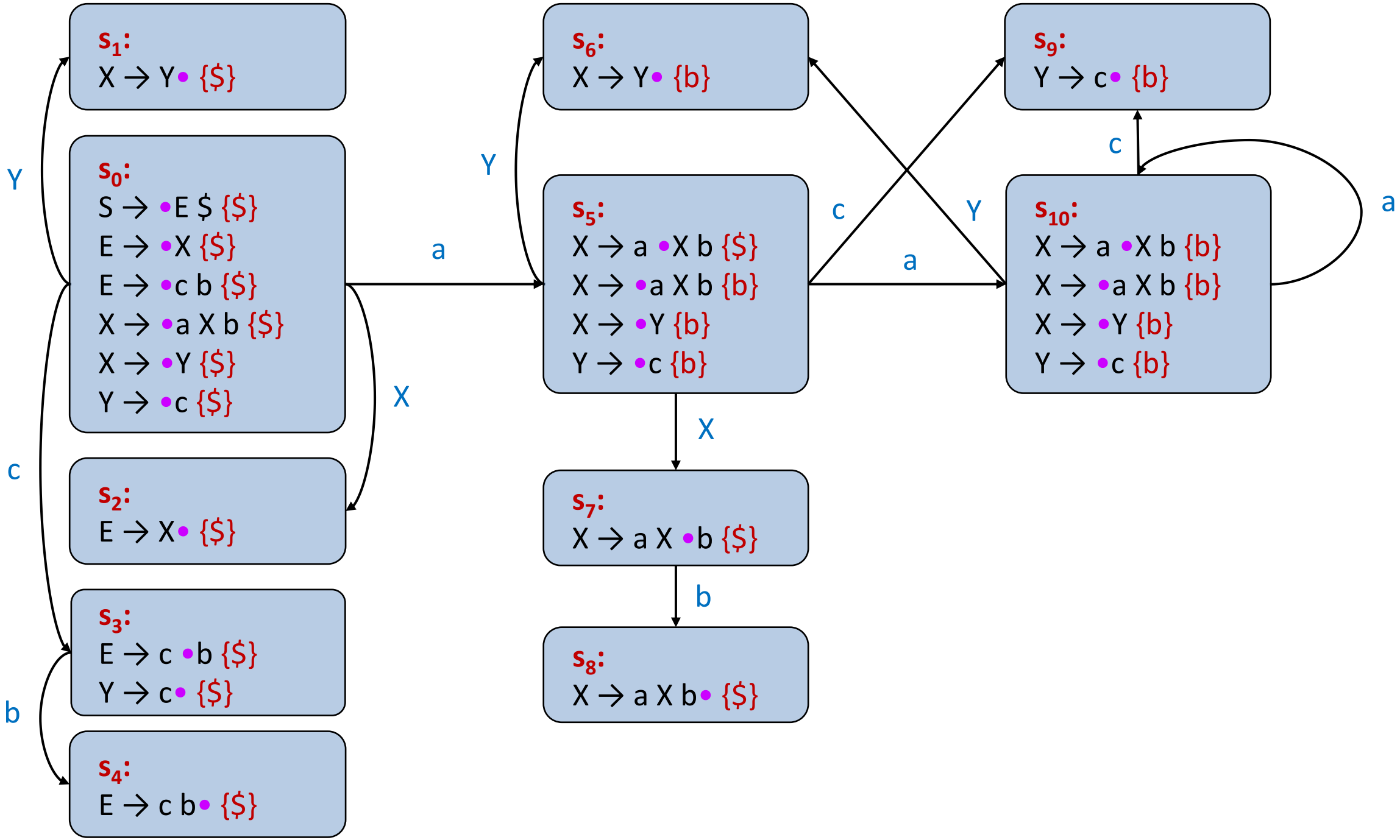

s1:
X → Y• {$}
s0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s2:
E → X• {$}
s3:
E → c •b {$}
Y → c• {$}
s4:
E → c b• {$}
s6:
X → Y• {b}
s5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s7:
X → a X •b {$}
s8:
X → a X b• {$}
s9:
Y → c• {b}
Y
c
b
a
X
Y
X
c
b

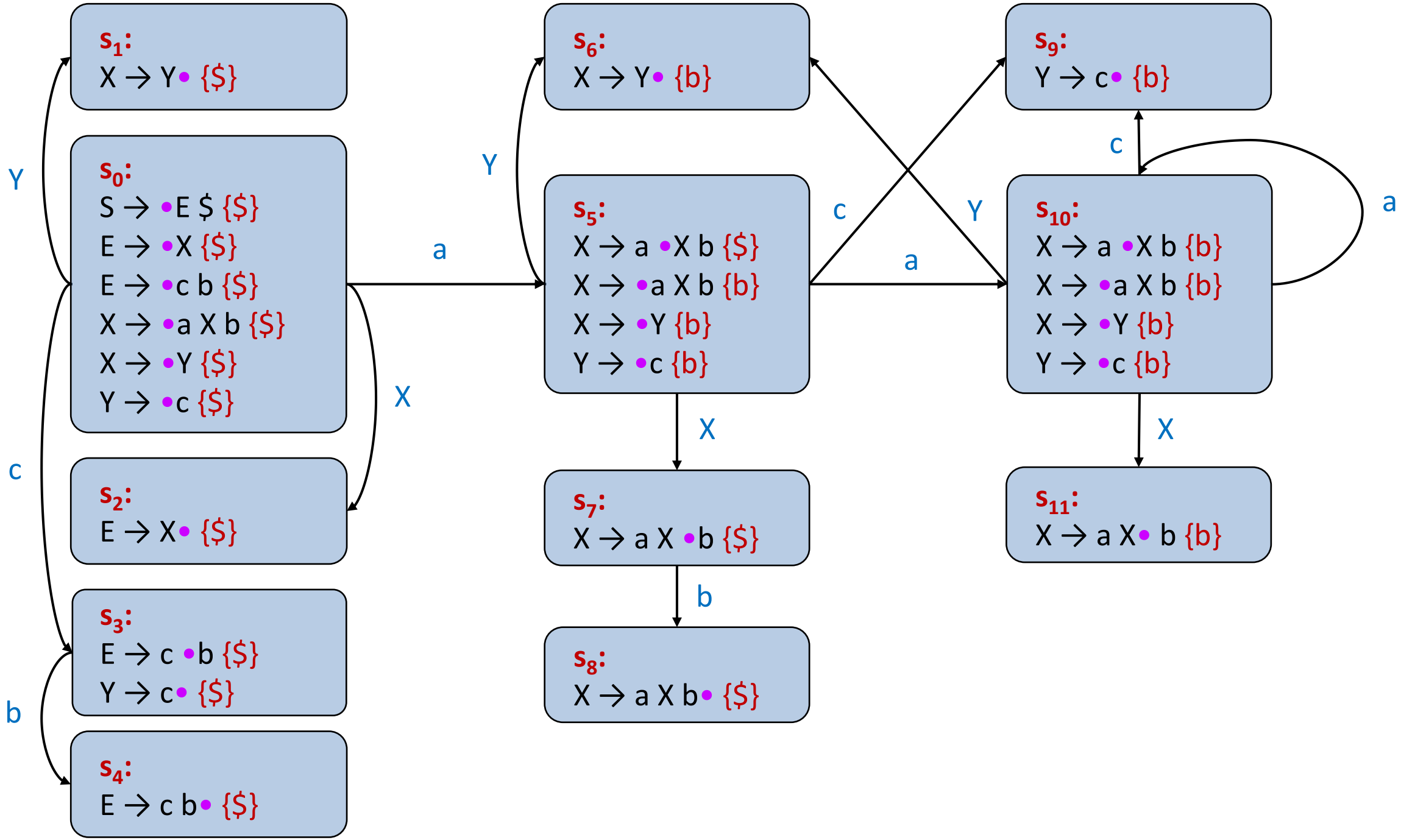

s1:
X → Y• {$}
s0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s2:
E → X• {$}
s3:
E → c •b {$}
Y → c• {$}
s4:
E → c b• {$}
s6:
X → Y• {b}
s5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s7:
X → a X •b {$}
s8:
X → a X b• {$}
s9:
Y → c• {b}
s10:
X → a •X b {b}
X → •a X b
X → •Y
Y → •c
Y
c
b
a
X
Y
a
c
X
b

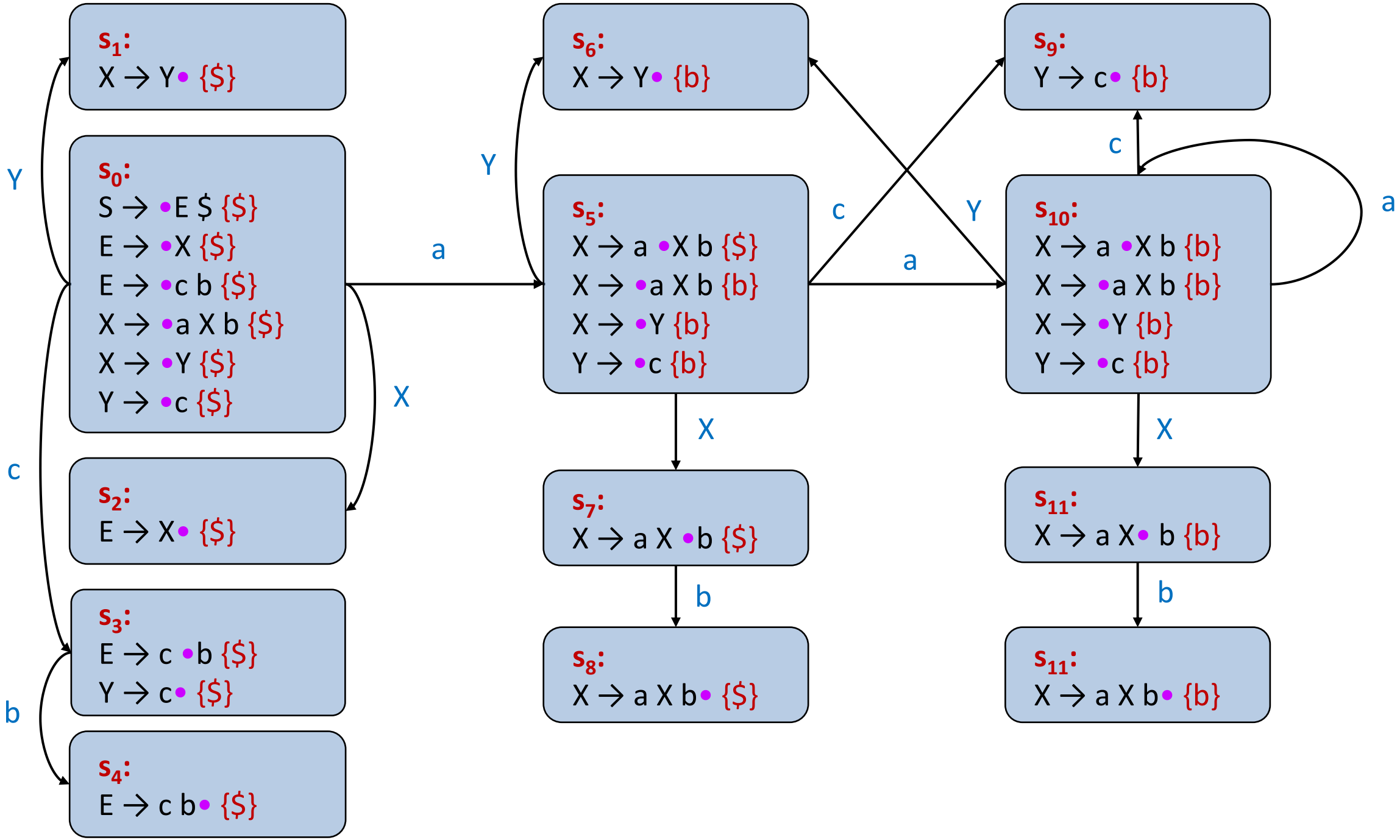

s1:
X → Y• {$}
s0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s2:
E → X• {$}
s3:
E → c •b {$}
Y → c• {$}
s4:
E → c b• {$}
s6:
X → Y• {b}
s5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s7:
X → a X •b {$}
s8:
X → a X b• {$}
s9:
Y → c• {b}
s10:
X → a •X b {b}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
Y
c
b
a
X
Y
X
b
c
a


s1:
X → Y• {$}
s0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s2:
E → X• {$}
s3:
E → c •b {$}
Y → c• {$}
s4:
E → c b• {$}
s6:
X → Y• {b}
s5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s7:
X → a X •b {$}
s8:
X → a X b• {$}
s9:
Y → c• {b}
s10:
X → a •X b {b}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
Y
a
X
c
b
Y
a
X
b
c
Y
a


s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_2:
E → X• {$}
s_3:
E → c •b {$}
Y → c• {$}
s_4:
E → c b• {$}
s_6:
X → Y• {b}
s_5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s_7:
X → a X •b {$}
s_8:
X → a X b• {$}
s_9:
Y → c• {b}
s_{10}:
X → a •X b {b}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
Y
c
b
a
X
Y
a
X
b
c
Y
a
c


s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_2:
E → X• {$}
s_3:
E → c •b {$}
Y → c• {$}
s_4:
E → c b• {$}
s_6:
X → Y• {b}
s_5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s_7:
X → a X •b {$}
s_8:
X → a X b• {$}
s_9:
Y → c• {b}
s_{10}:
X → a •X b {b}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
Y
c
b
a
X
Y
a
X
b
c
Y
a
c
a


s_1:
X → Y• {$}
s_0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s_2:
E → X• {$}
s_3:
E → c •b {$}
Y → c• {$}
s_4:
E → c b• {$}
s_6:
X → Y• {b}
s_5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s_7:
X → a X •b {$}
s_8:
X → a X b• {$}
s_9:
Y → c• {b}
s_{10}:
X → a •X b {b}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s_{11}:
X → a X• b {b}
Y
c
b
a
X
Y
a
X
c
Y
a
c
X
b
a


s1:
X → Y• {$}
s0:
S → •E $ {$}
E → •X {$}
E → •c b {$}
X → •a X b {$}
X → •Y {$}
Y → •c {$}
s2:
E → X• {$}
s3:
E → c •b {$}
Y → c• {$}
s4:
E → c b• {$}
s6:
X → Y• {b}
s5:
X → a •X b {$}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s7:
X → a X •b {$}
s8:
X → a X b• {$}
s9:
Y → c• {b}
s10:
X → a •X b {b}
X → •a X b {b}
X → •Y {b}
Y → •c {b}
s11:
X → a X• b {b}
s11:
X → a X b• {b}
Y
a
X
c
b
Y
a
X
c
Y
a
c
a
X
b

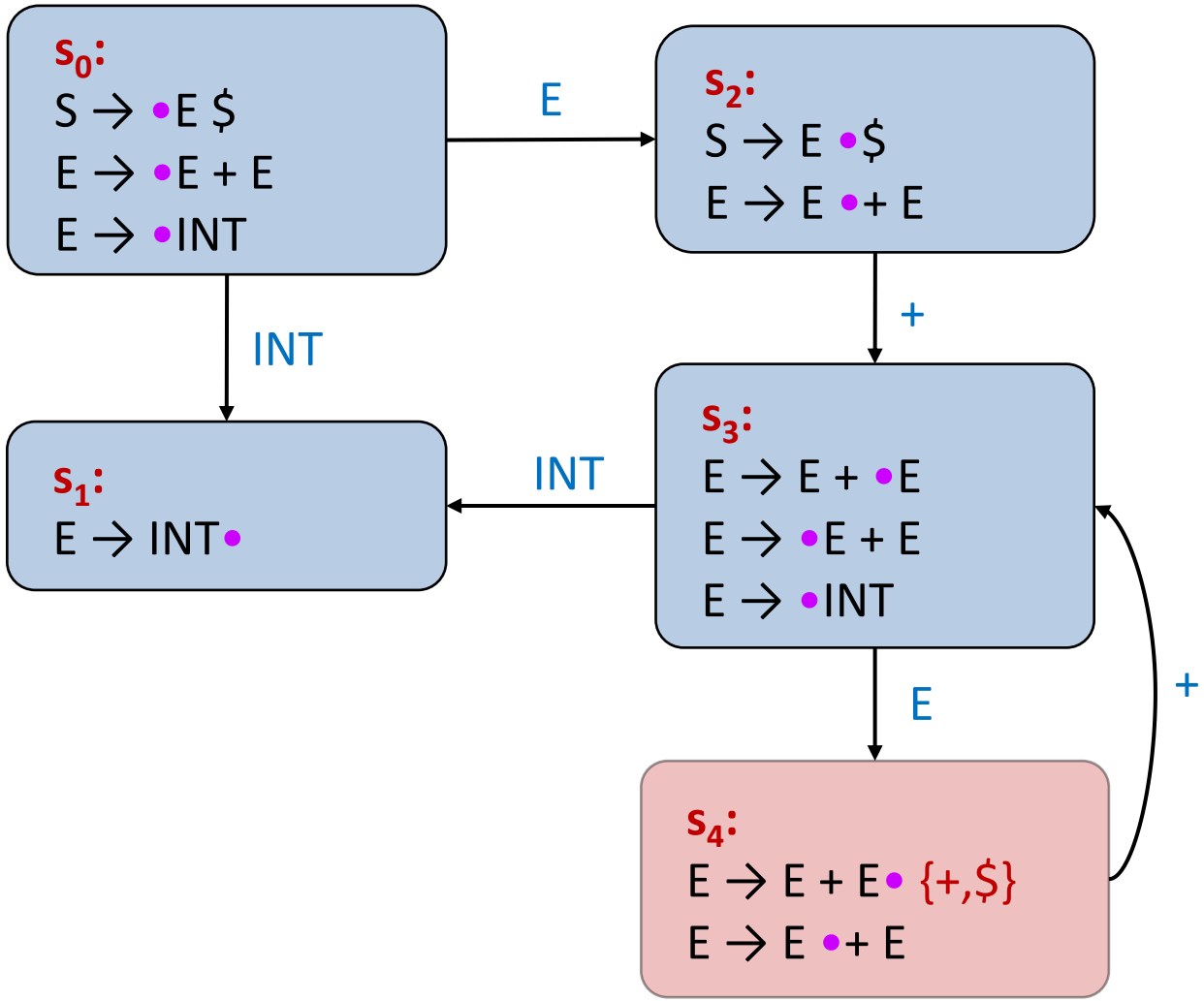
Resolving Conflicts

S → E $

E → E + E

F → INT

- What will be the SLR(1) automaton for this CFG?


s_0:
S → •E $
E → •E + E
E → •INT
E
s_2:
S → E •$
E → E •+ E
+
INT
s_3:
E → E + •E
E → •E + E
E → •INT
INT
s_1:
E → INT•
E
+
s_4:
E → E + E• {+,$}
E → E •+ E

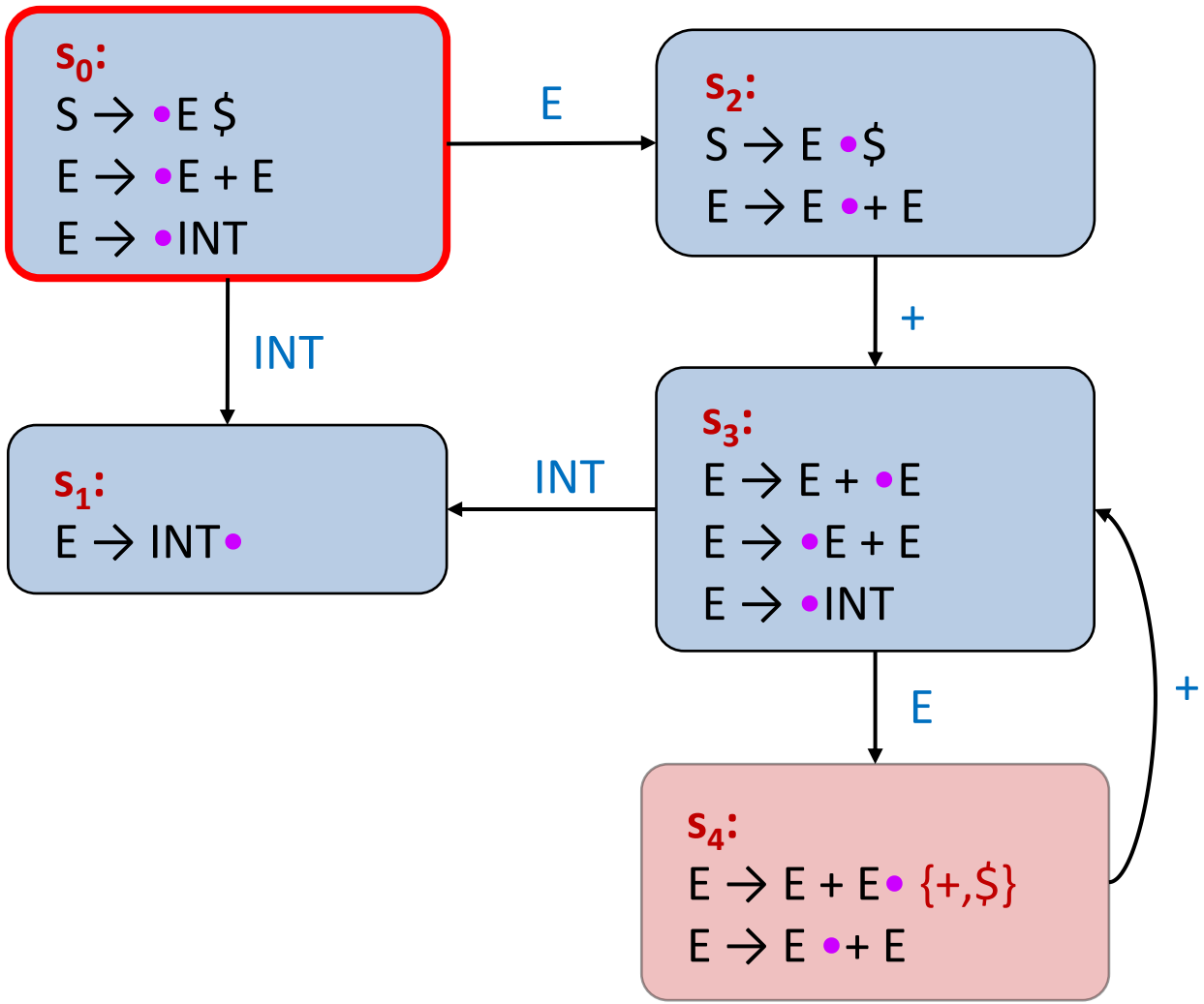
Resolving Conflicts – Associativity

- How will each operation affect associativity?
- Shift:
  - Right associative (x + y + z = (x + y) + z)
- Reduce:
  - Left associative (x + y + z = x + (y + z))

s_4:

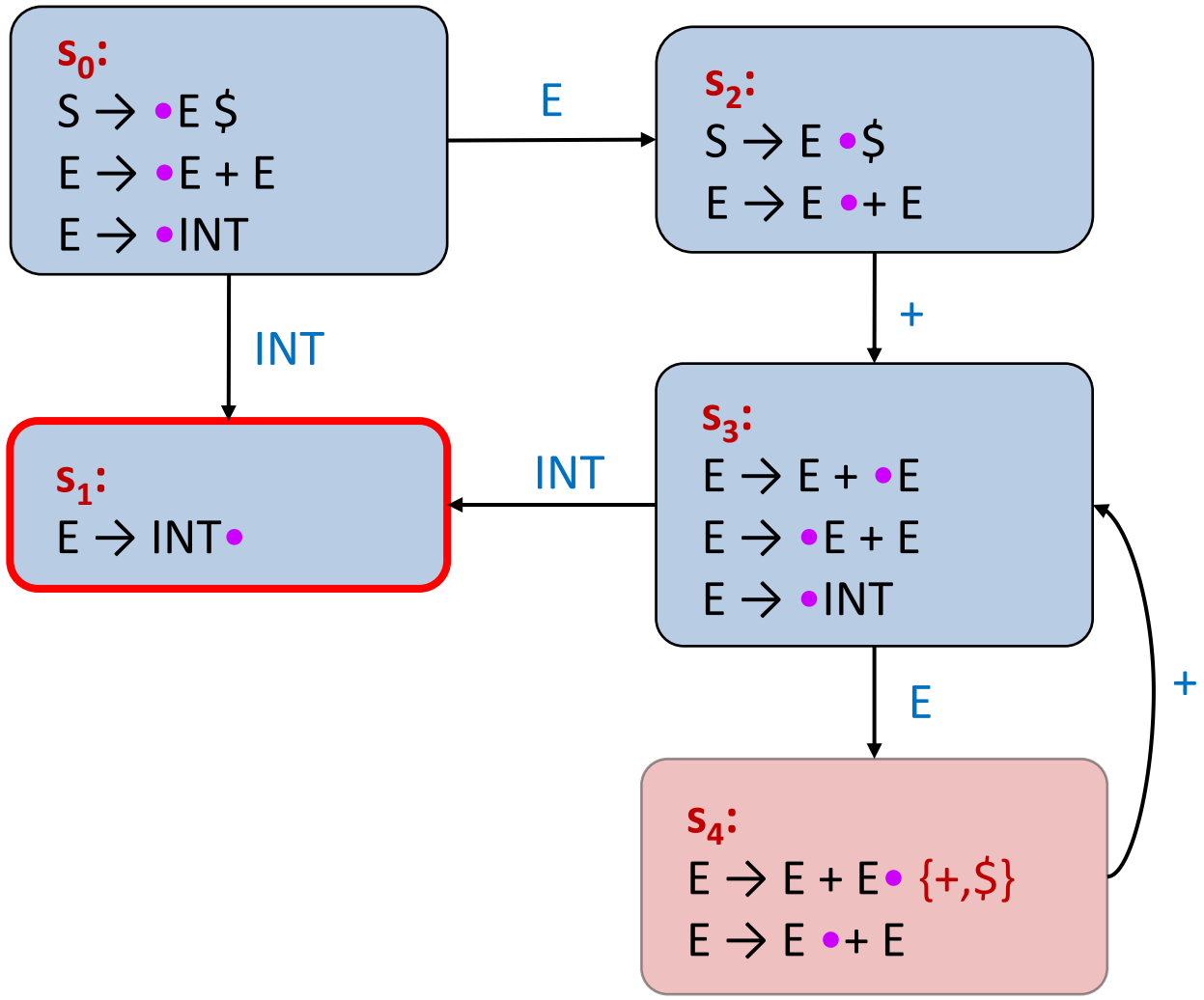
E → E + E• {+,$}

E → E •+ E



Input: 1 + 2 + 3$

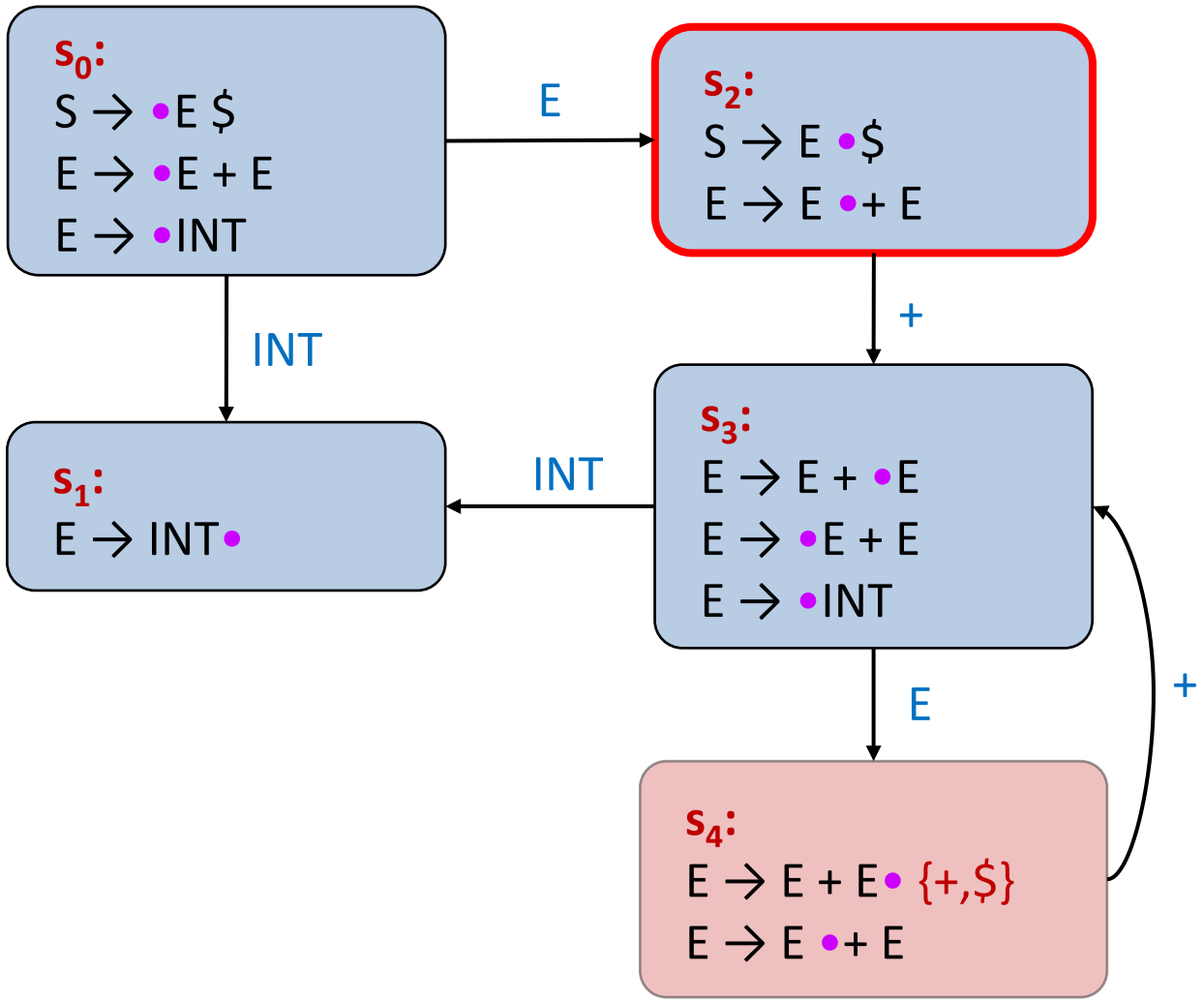
Stack: s_0

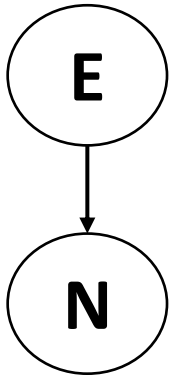


N

Input: 1 + 2 + 3$

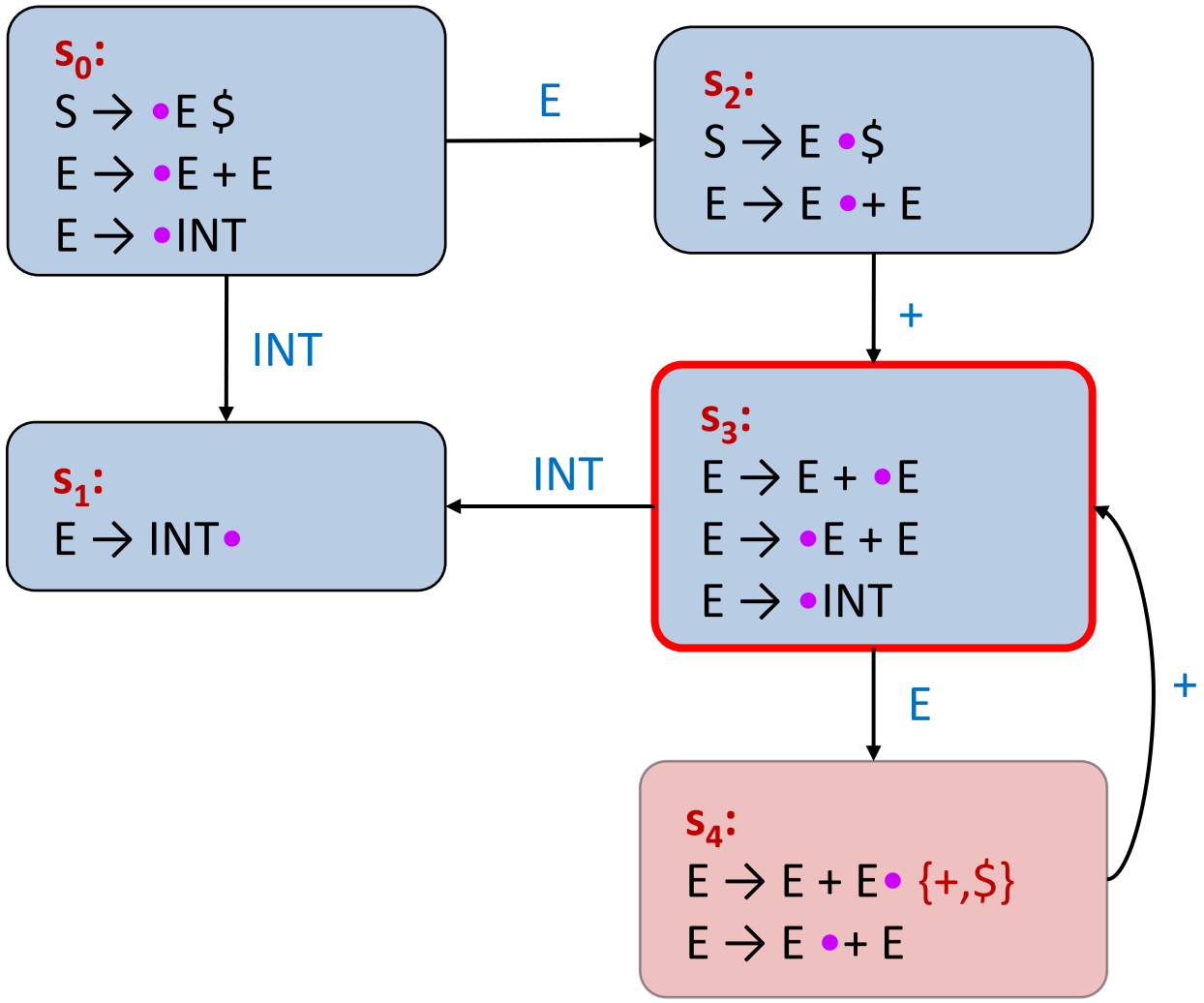
Stack: s_0 INT s_1

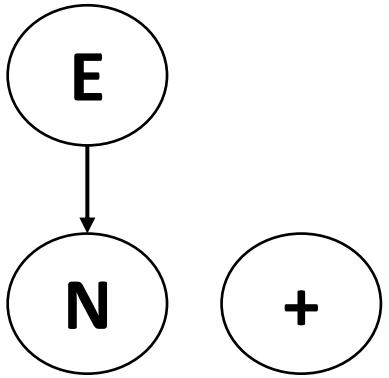




Input: 1 + 2 + 3$

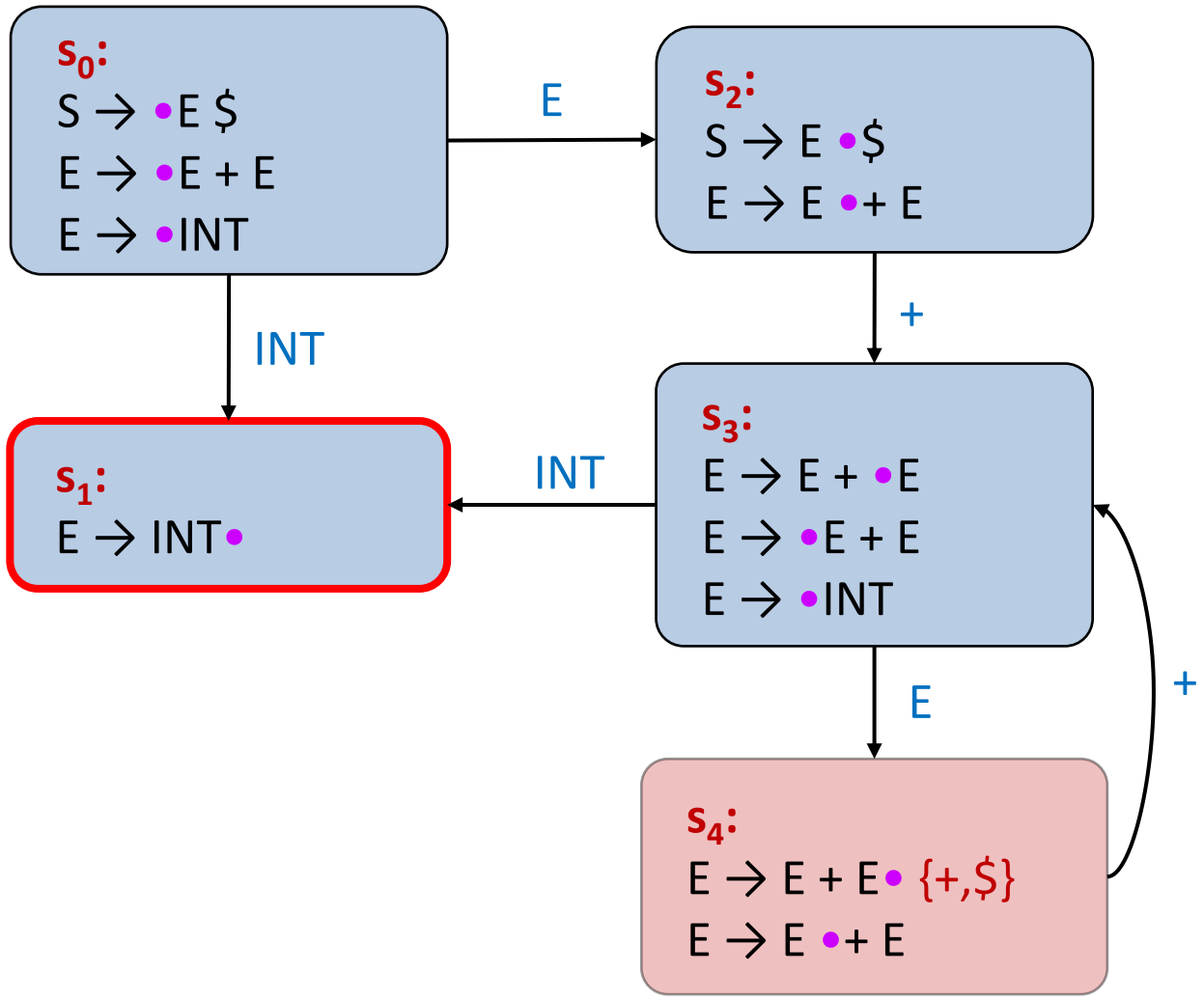
Stack: s_0 E s_2

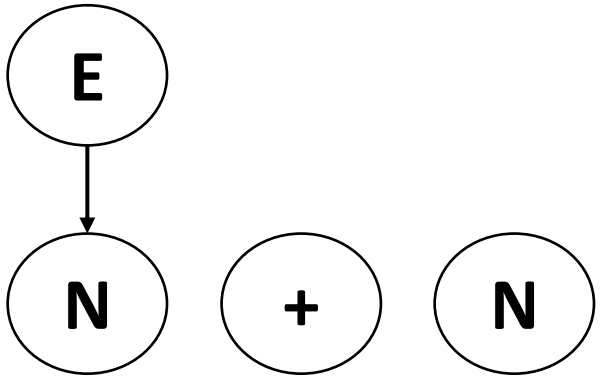




Input: 1 + 2 + 3$

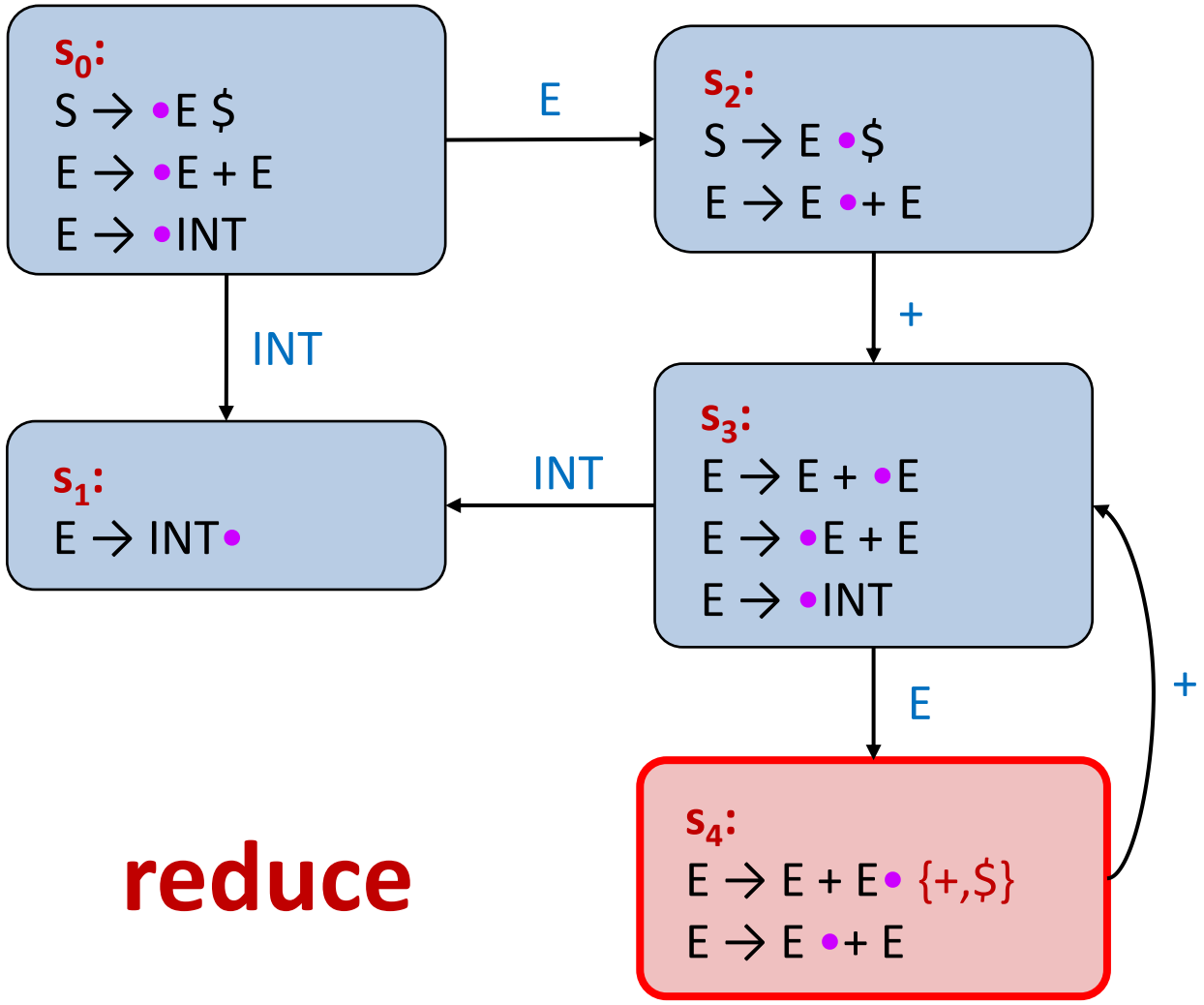
Stack: s_0 E s_2 + s_3



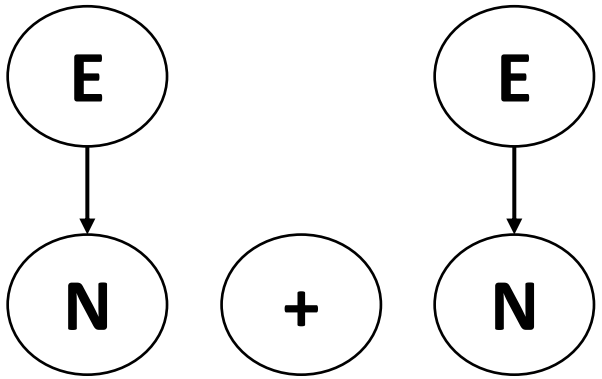


Input: 1 + 2 + 3$

Stack: s_0 E s_2 + s_3 INT s_1

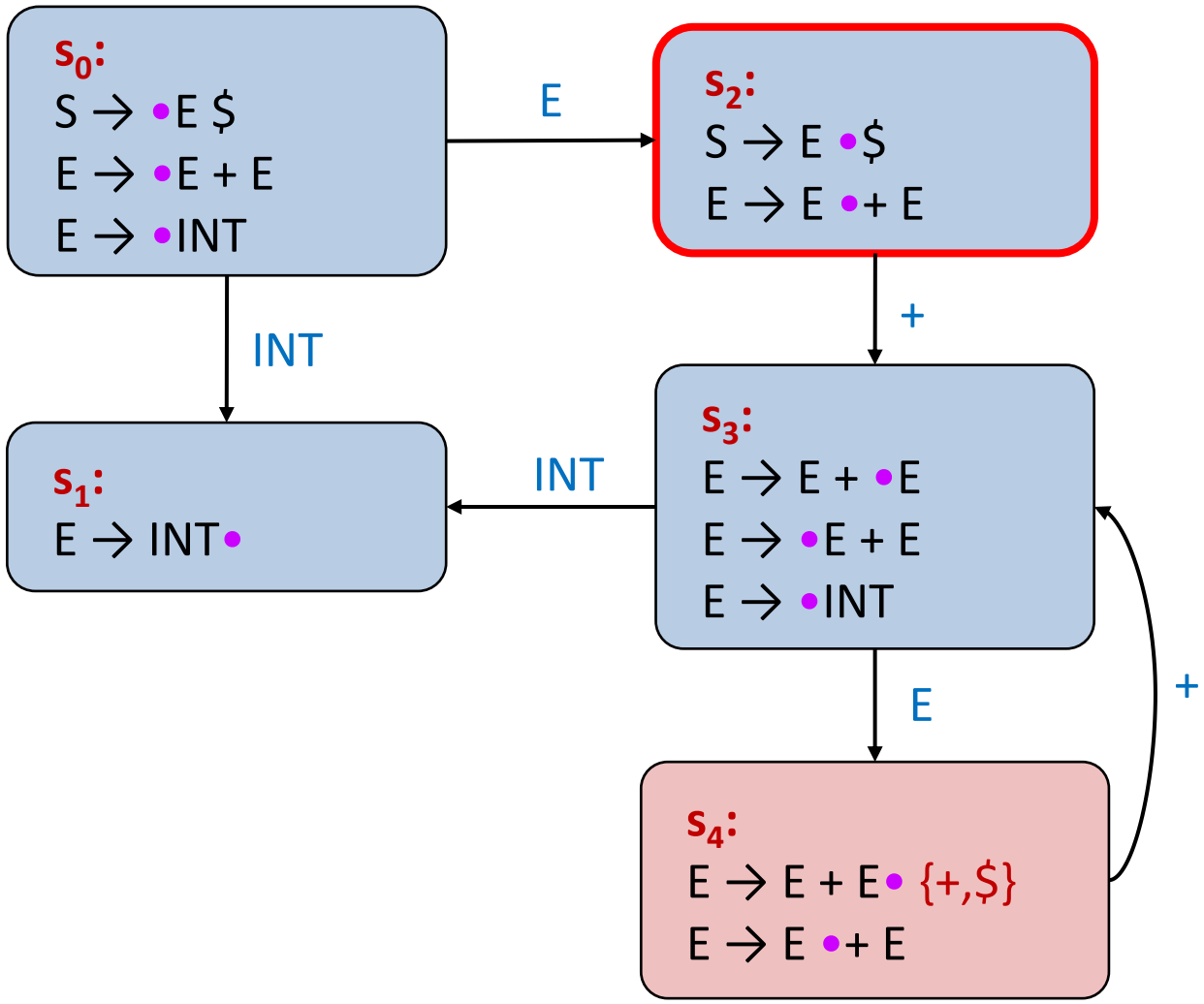


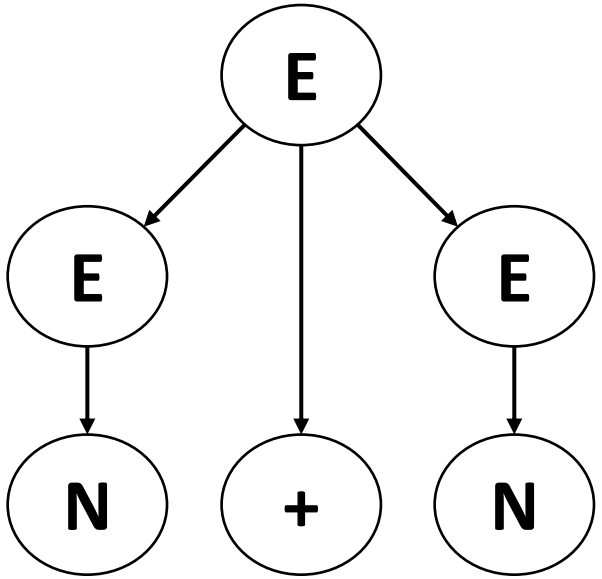
reduce



Input: 1 + 2 + 3$

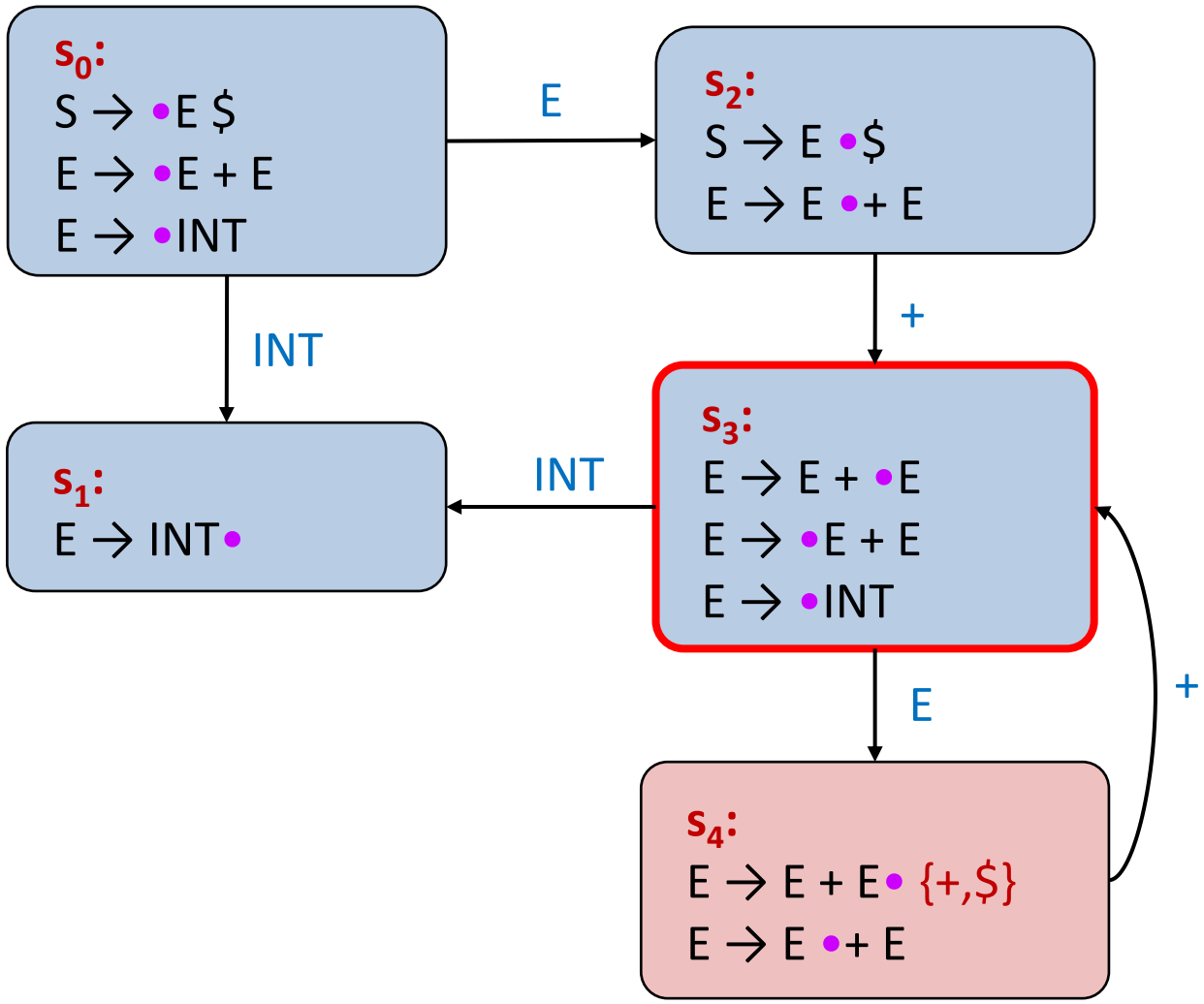
Stack: s_0 E s_2 + s_3 E s_4

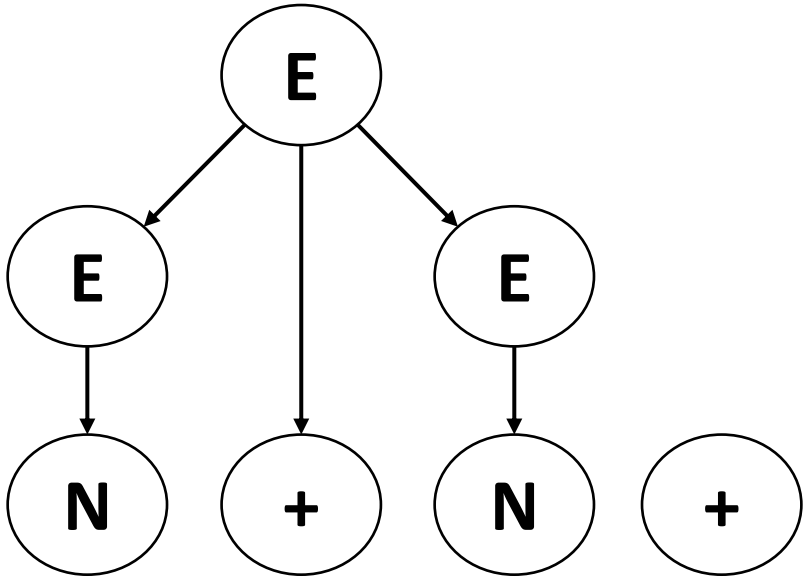




Input: 1 + 2 + 3$

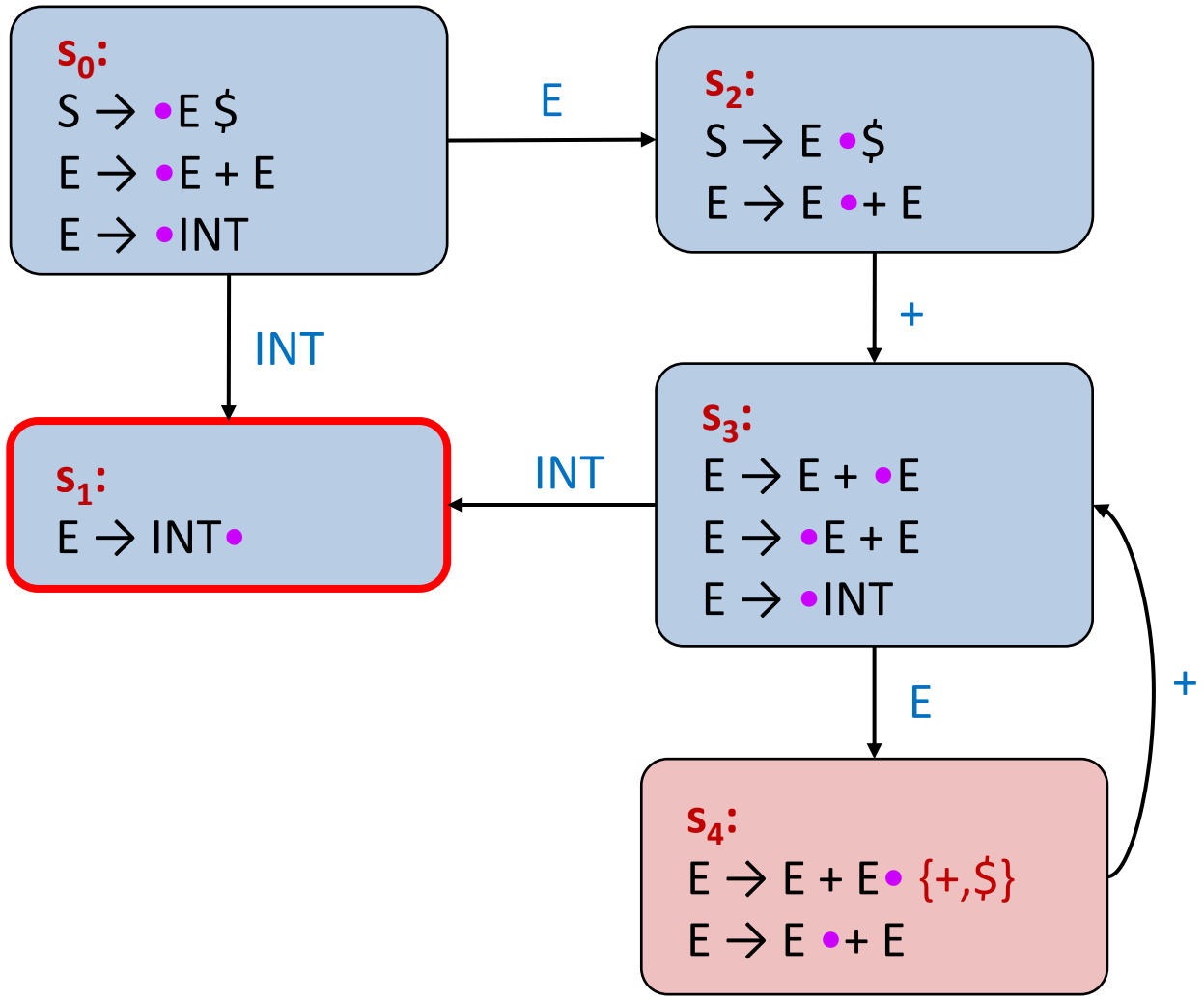
Stack: s_0 E s_2

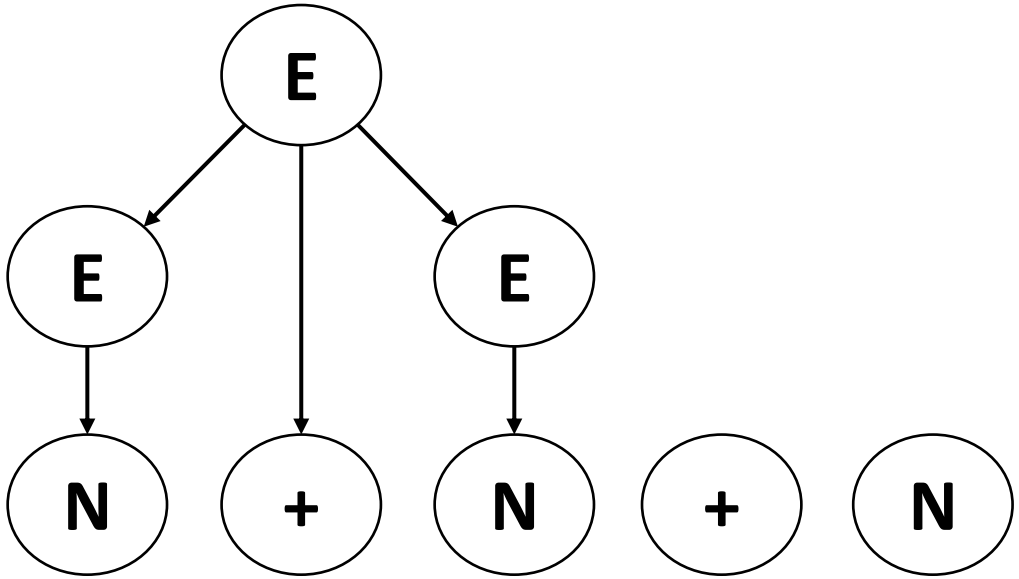




Input: 1 + 2 + 3$

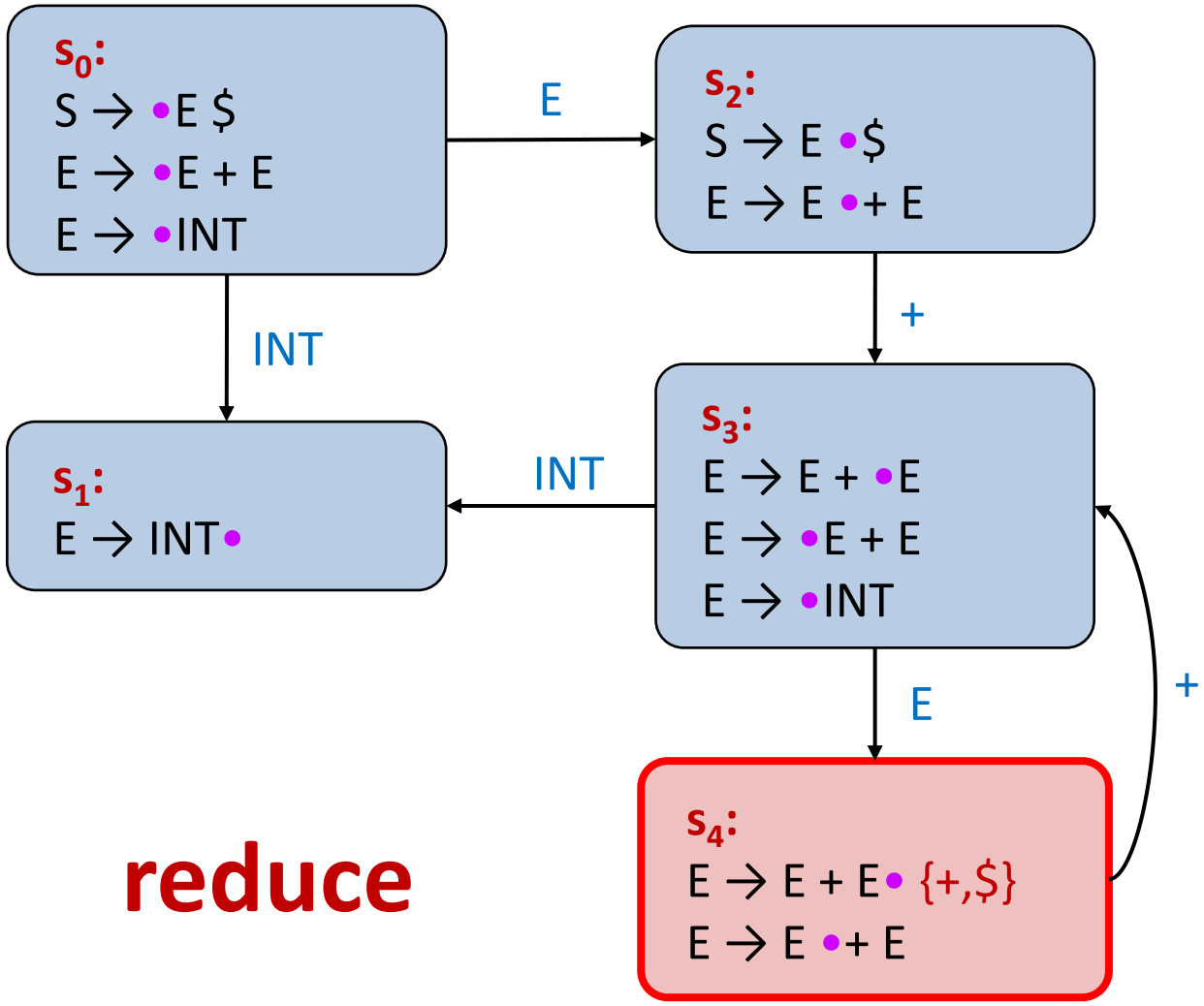
Stack: s_0 E s_2 + s_3

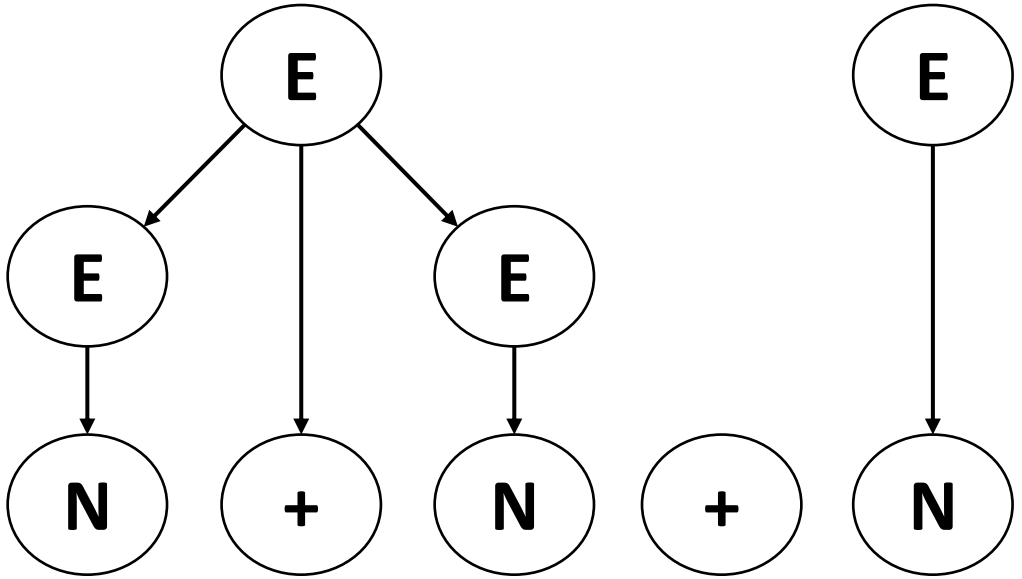




Input: 1 + 2 + 3$

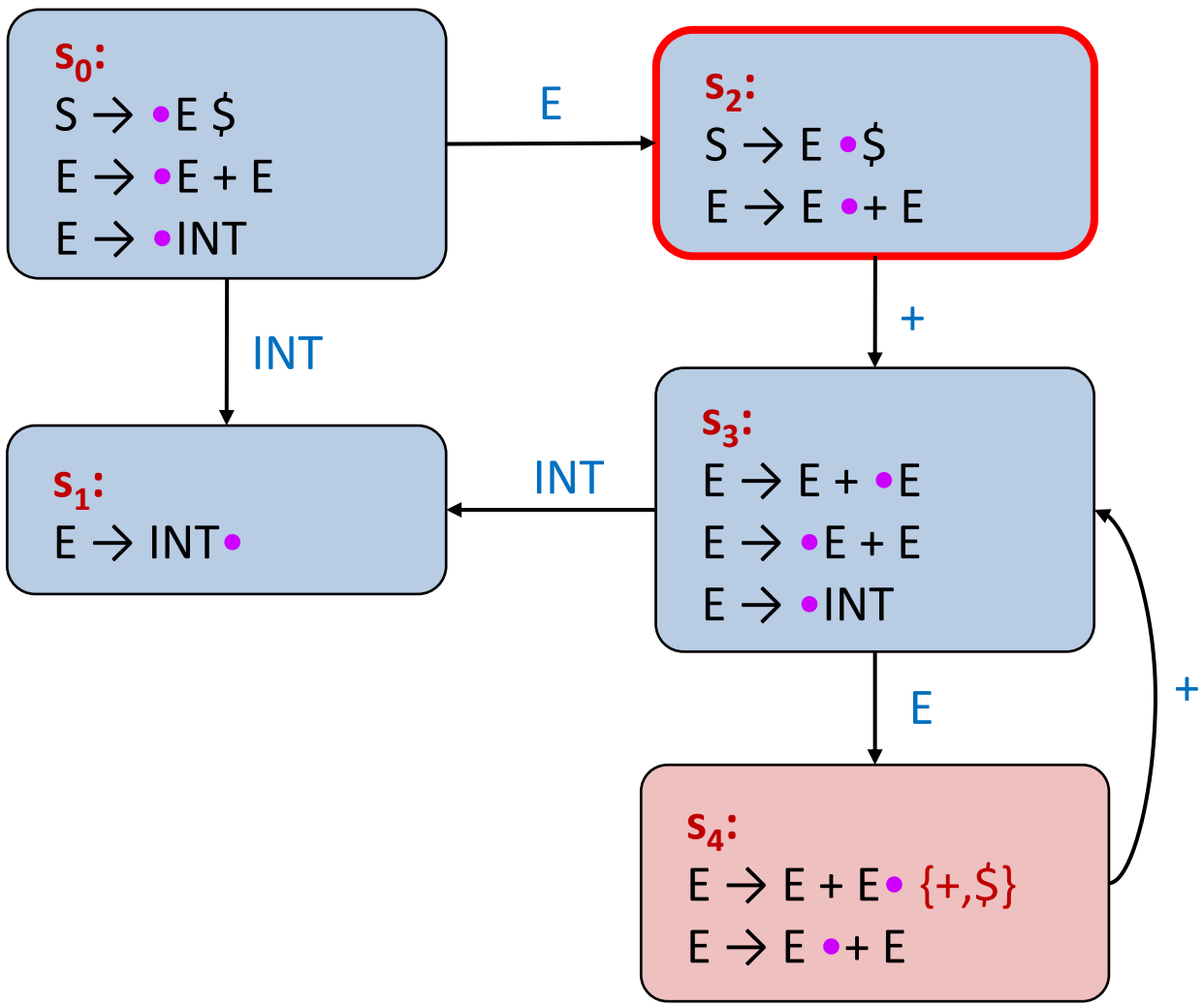
Stack: s_0 E s_2 + s_3 INT s_1

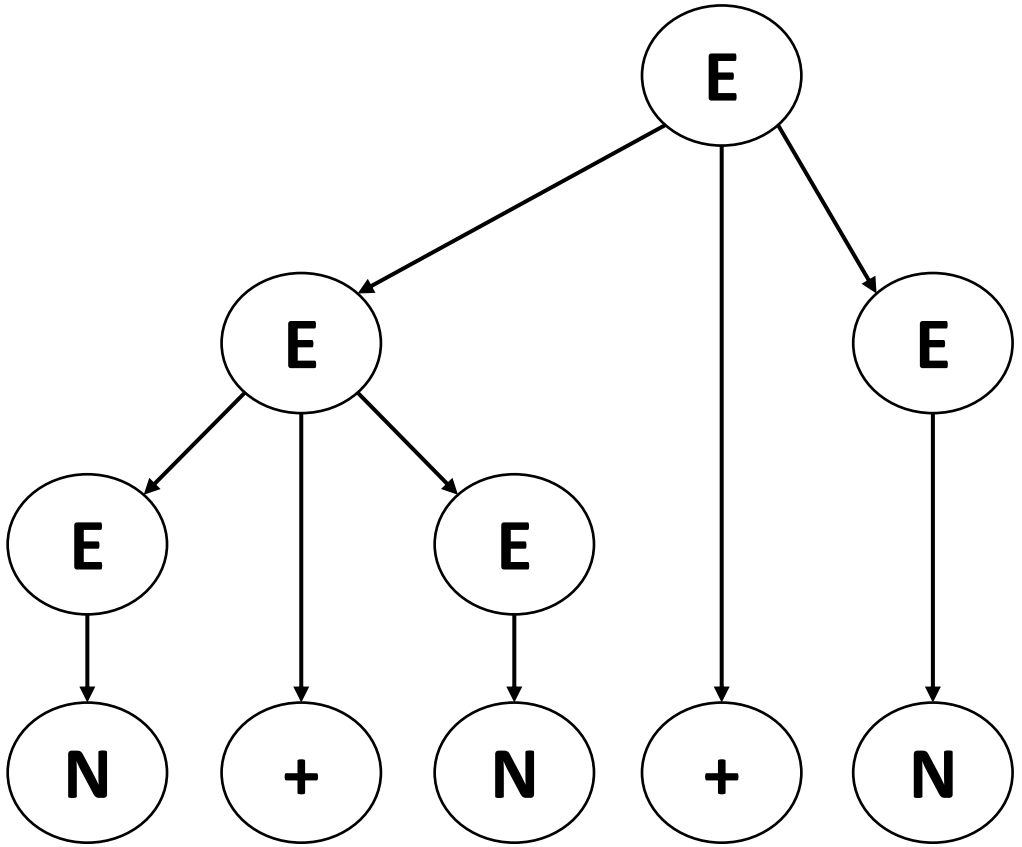




Input: 1 + 2 + 3$

Stack: s_0 E s_2 + s_3 E s_4





Input: 1 + 2 + 3$

Stack: s_0 E s_2

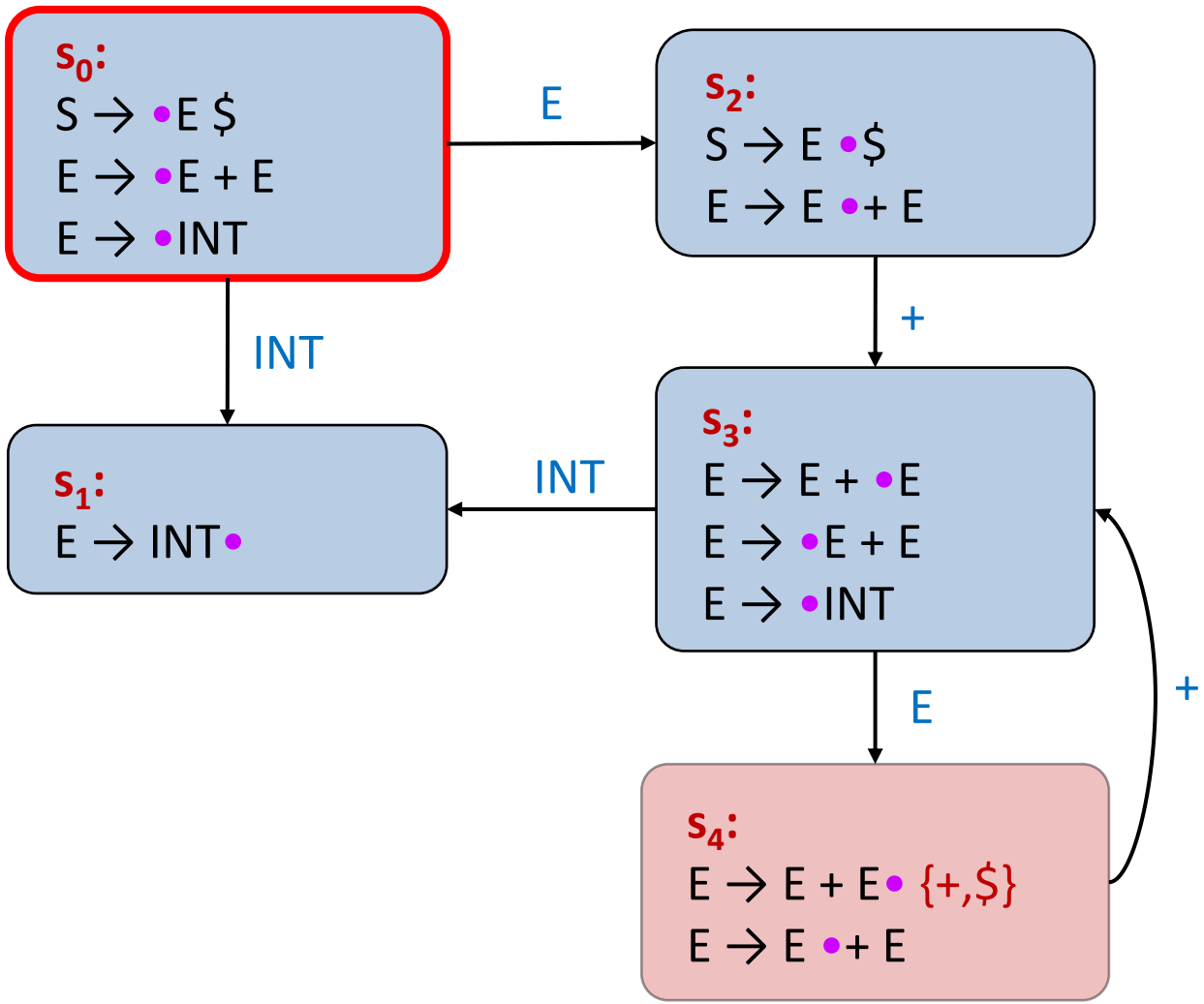
Resolving Conflicts – Associativity

- How will each operation affect associativity?
- Shift:
  - Right associative (x + y + z = (x + y) + z)
- Reduce:
  - Left associative (x + y + z = x + (y + z))

s_4:

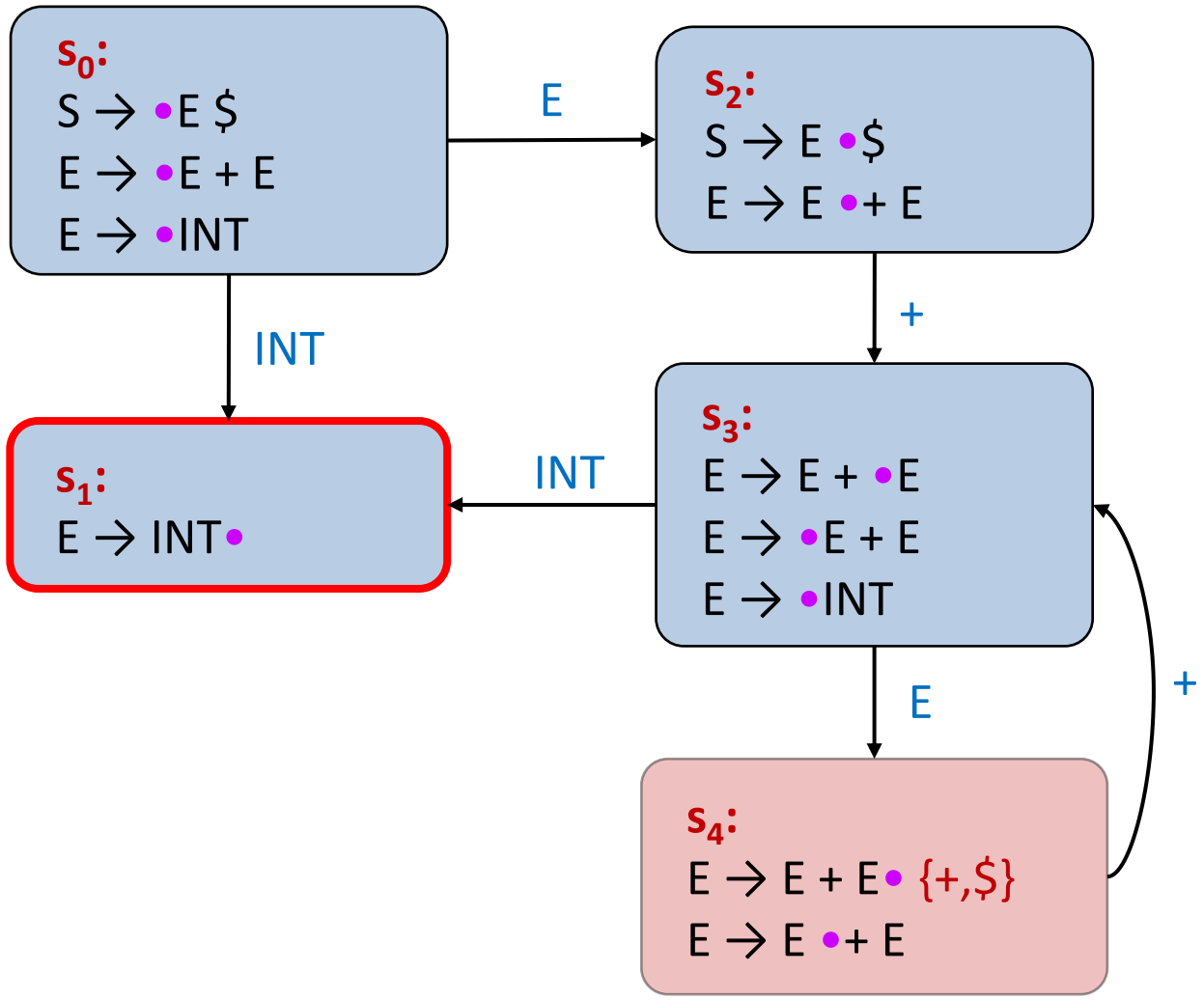
E → E + E• {+,$}

E → E •+ E



Input: 1 + 2 + 3$

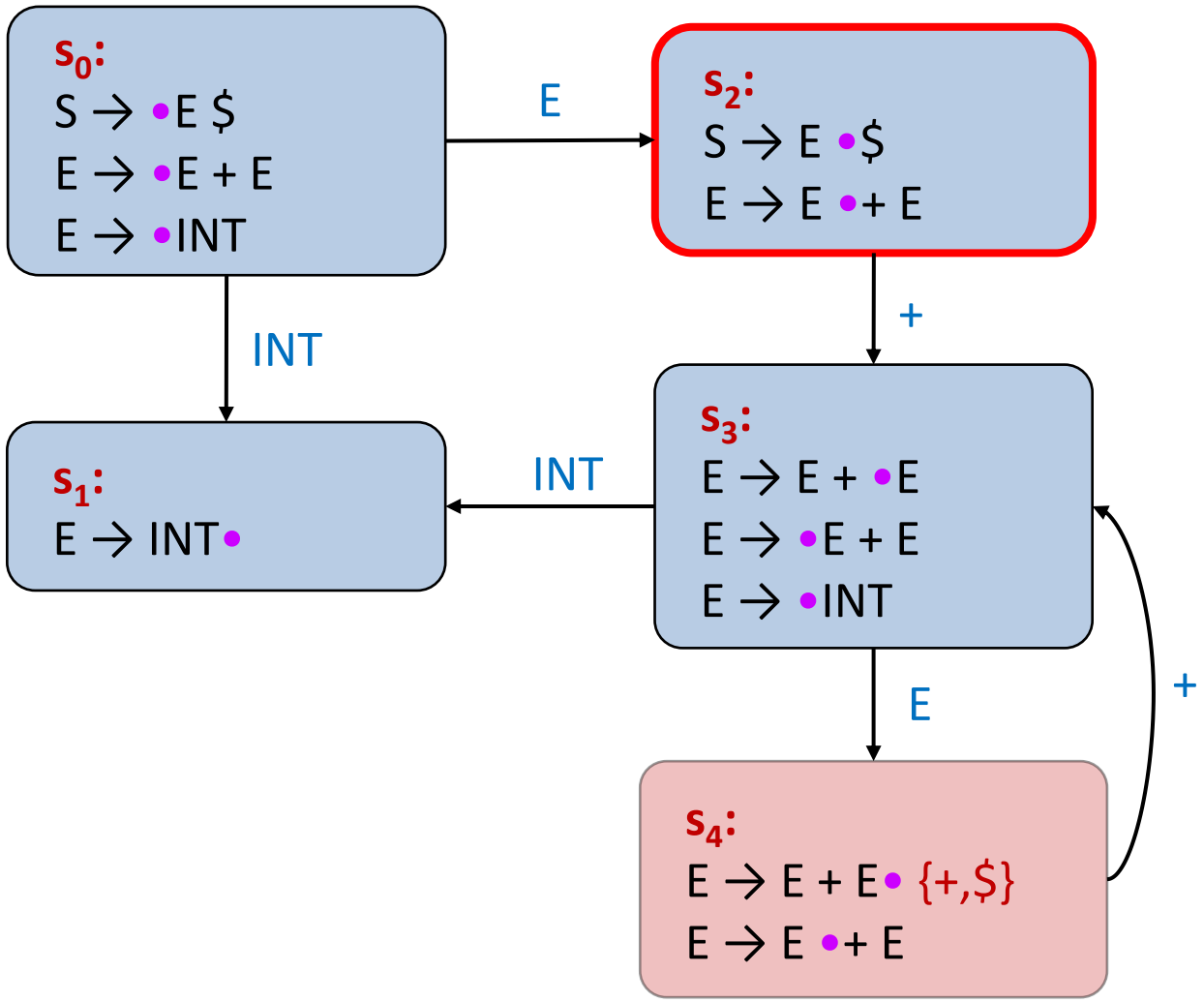
Stack: s_0

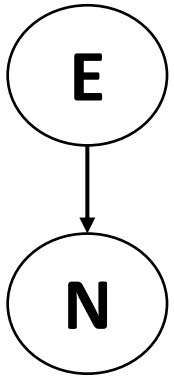


N

Input: 1 + 2 + 3$

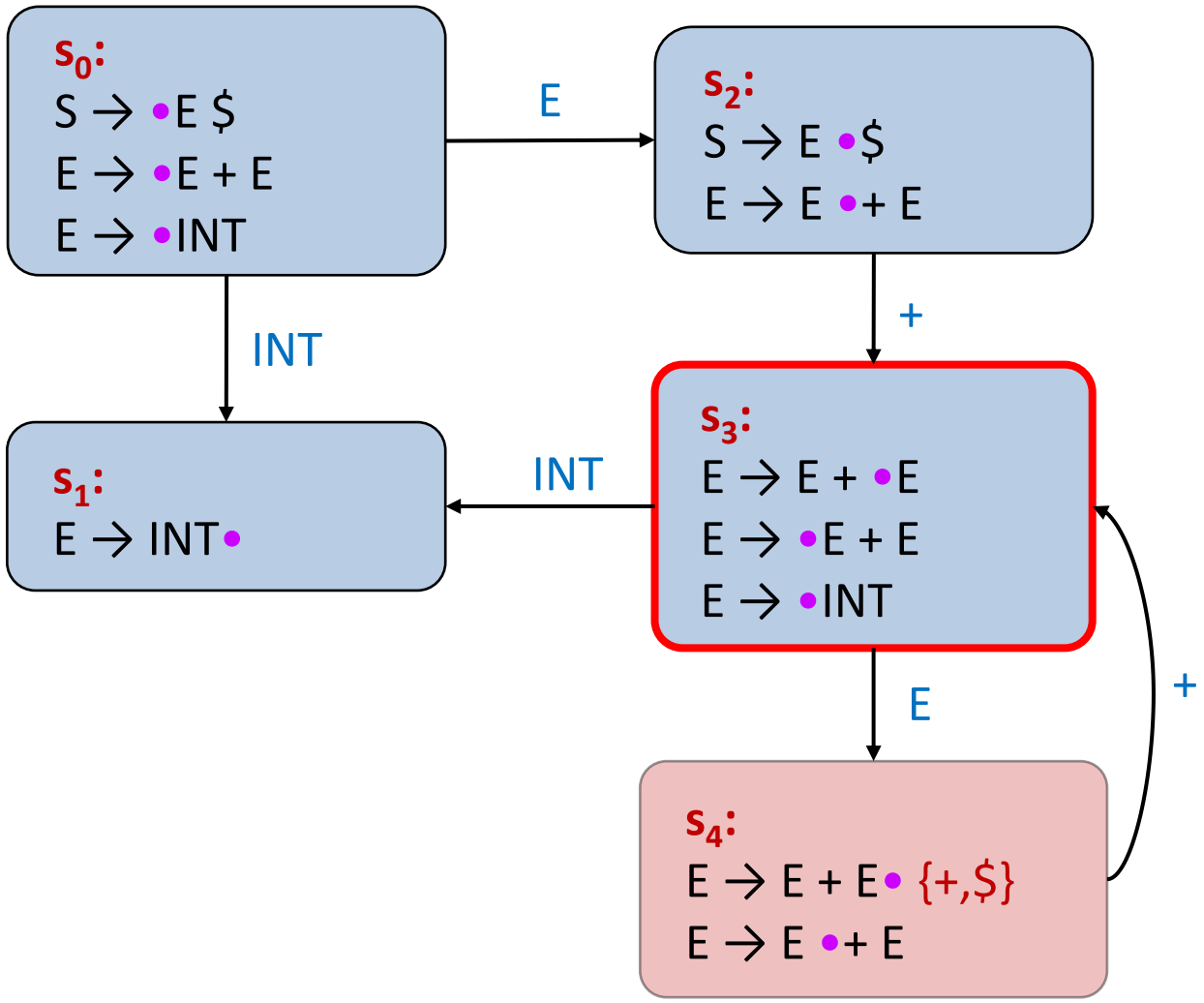
Stack: s_0 INT s_1

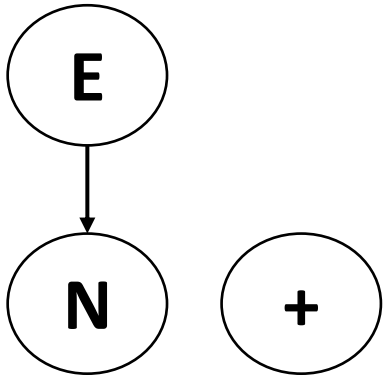




Input: 1 + 2 + 3$

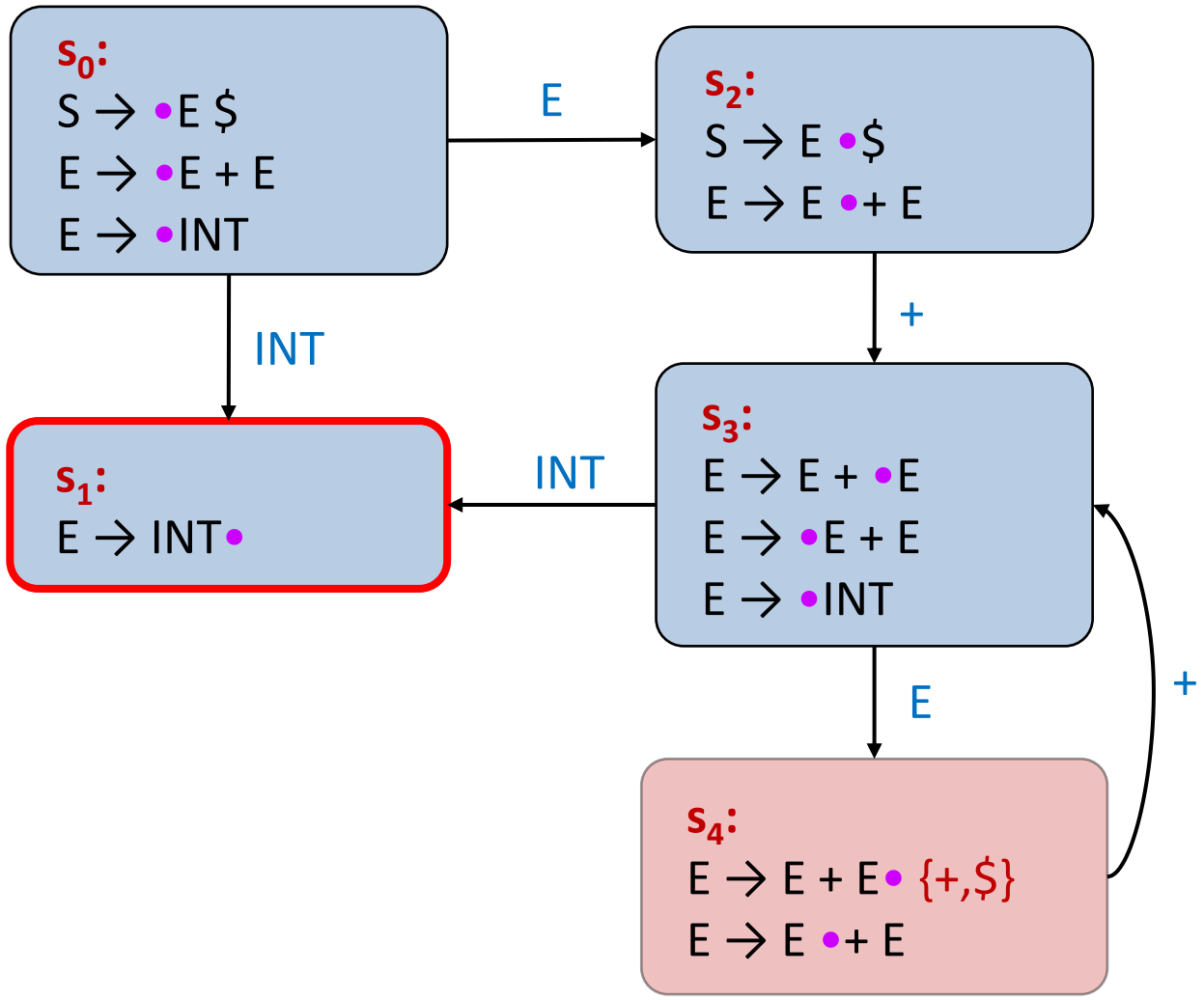
Stack: s_0 E s_2

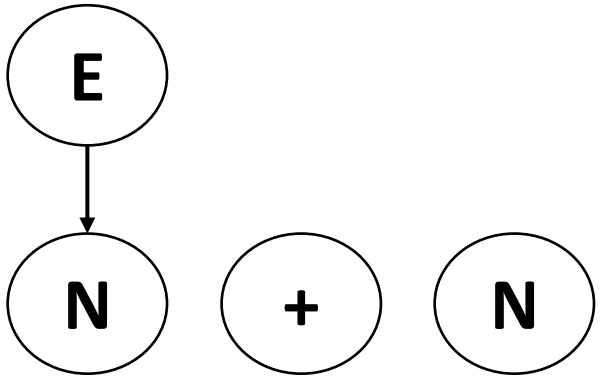




Input: 1 + 2 + 3$

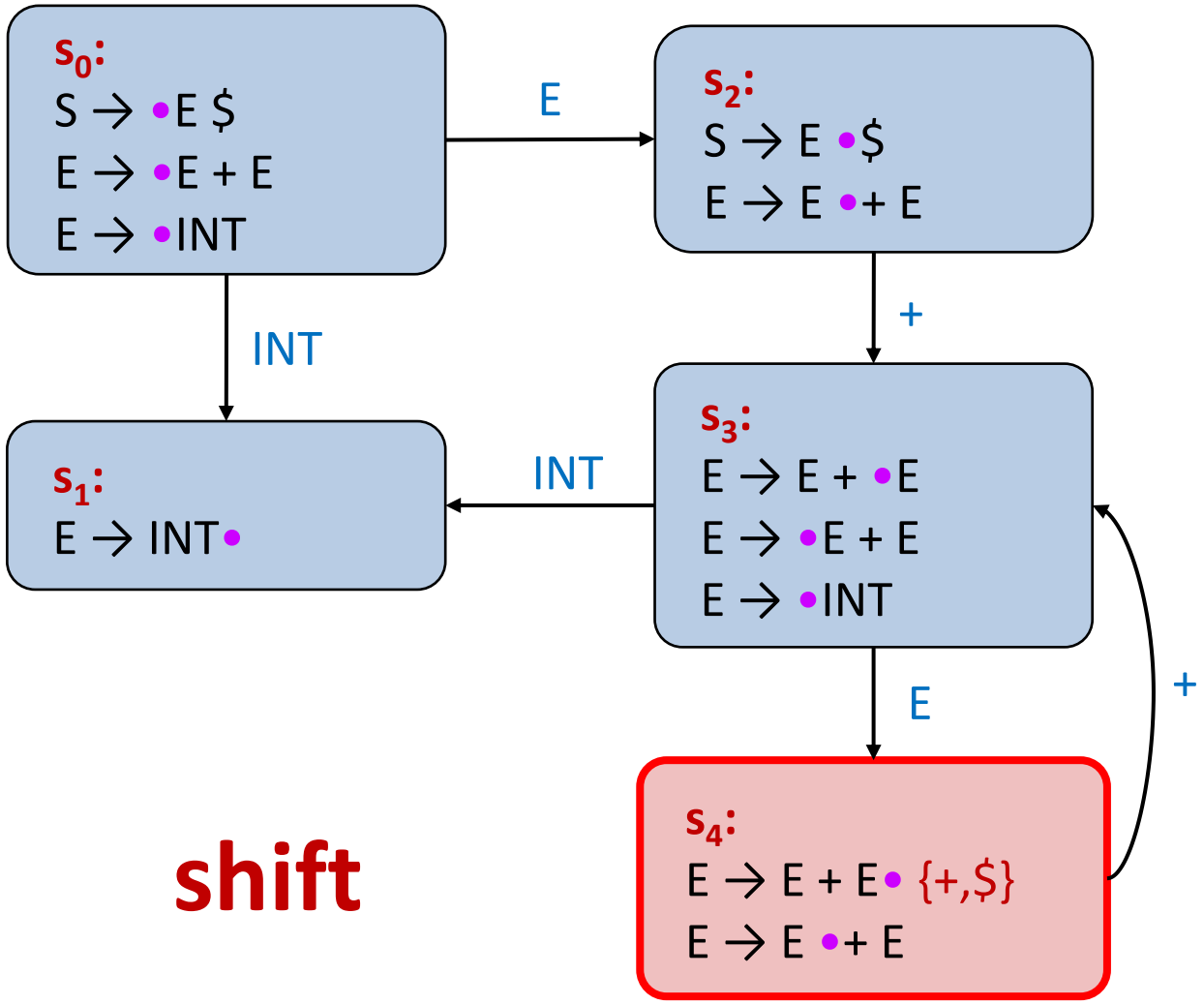
Stack: s_0 E s_2 + s_3



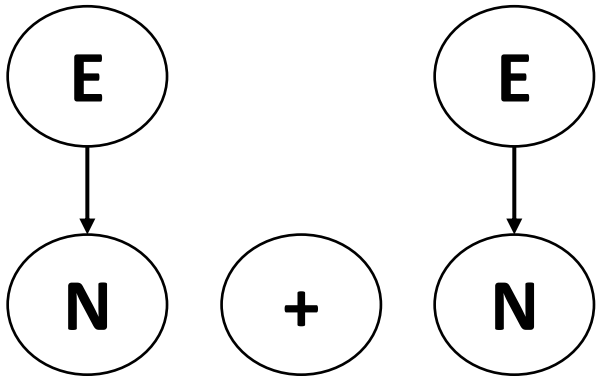


Input: 1 + 2 + 3$

Stack: s_0 E s_2 + s_3 INT s_1

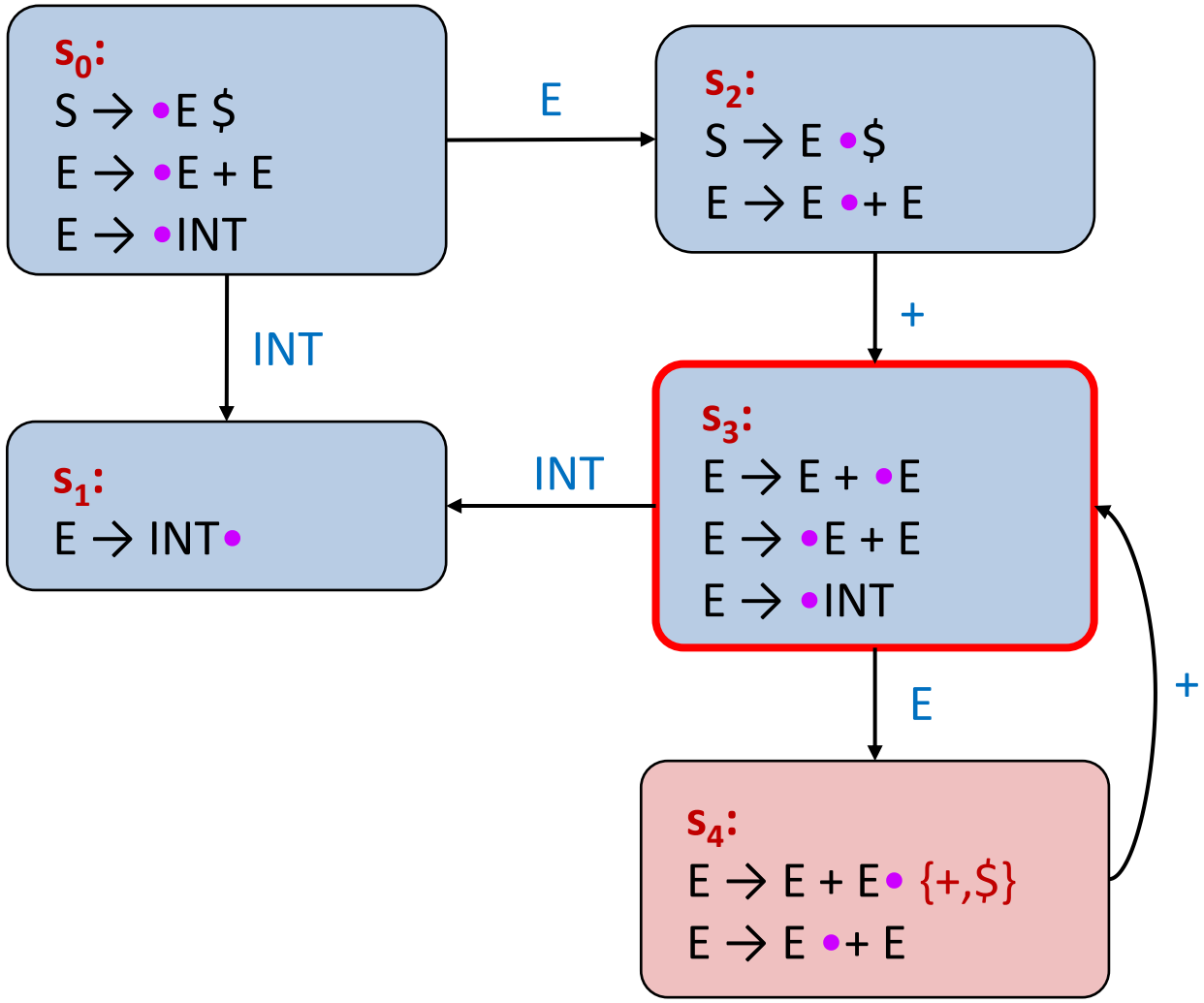


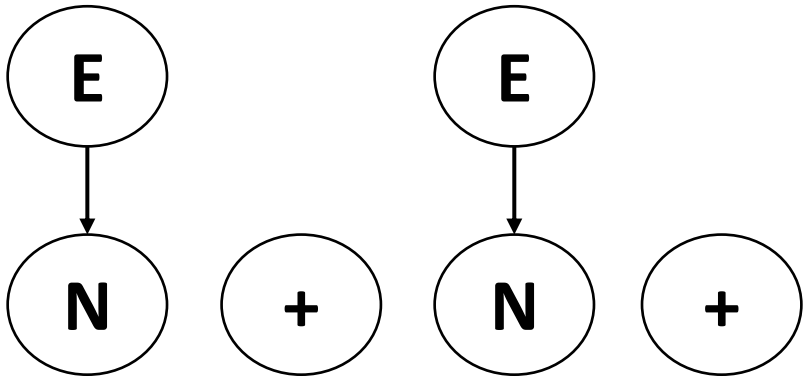
shift



Input: 1 + 2 + 3$

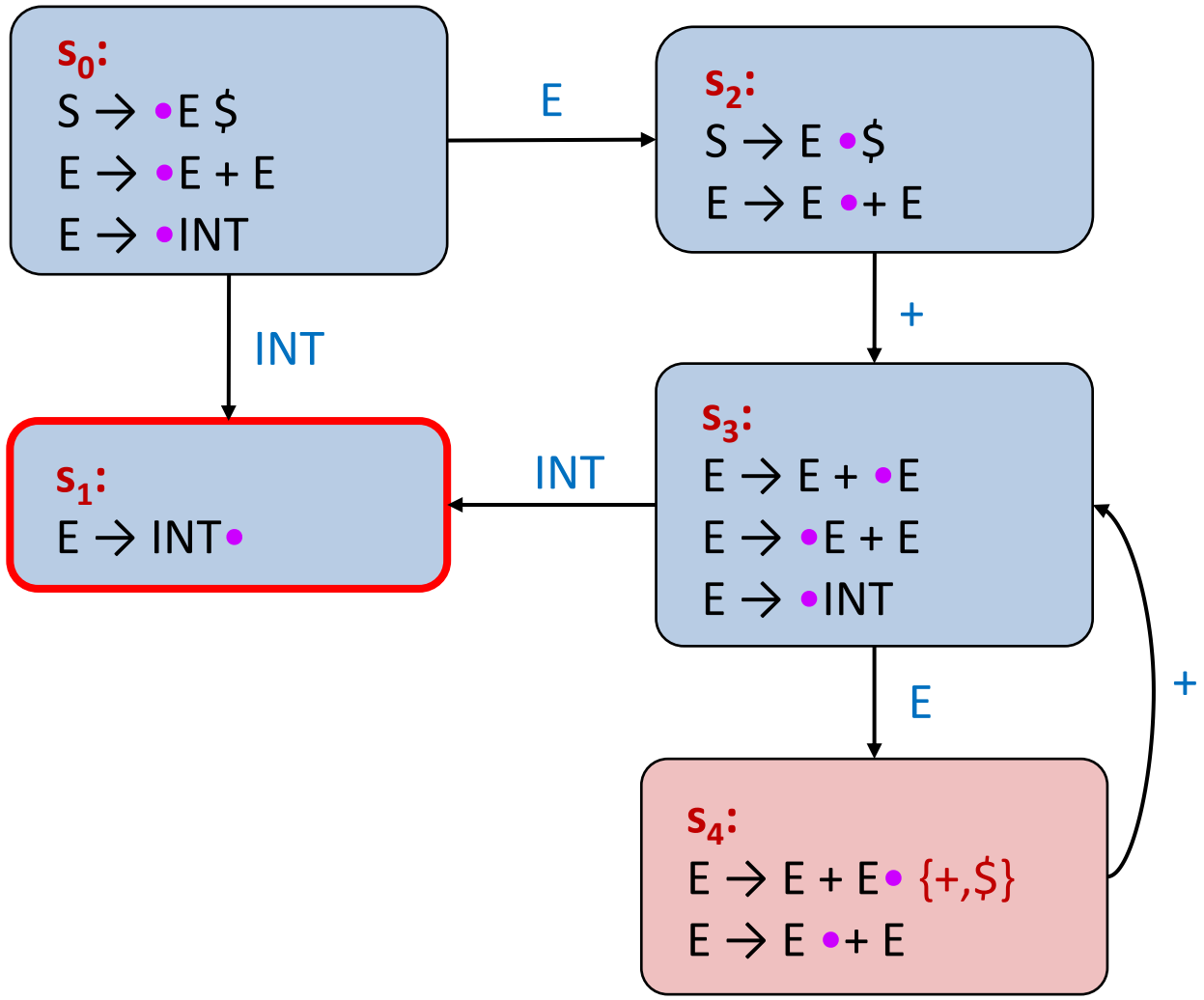
Stack: s_0 E s_2 + s_3 E s_4

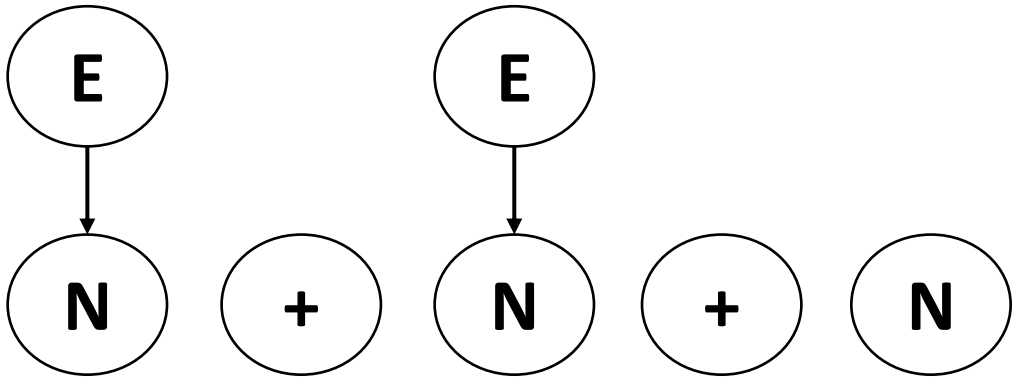




Input: 1 + 2 + 3$

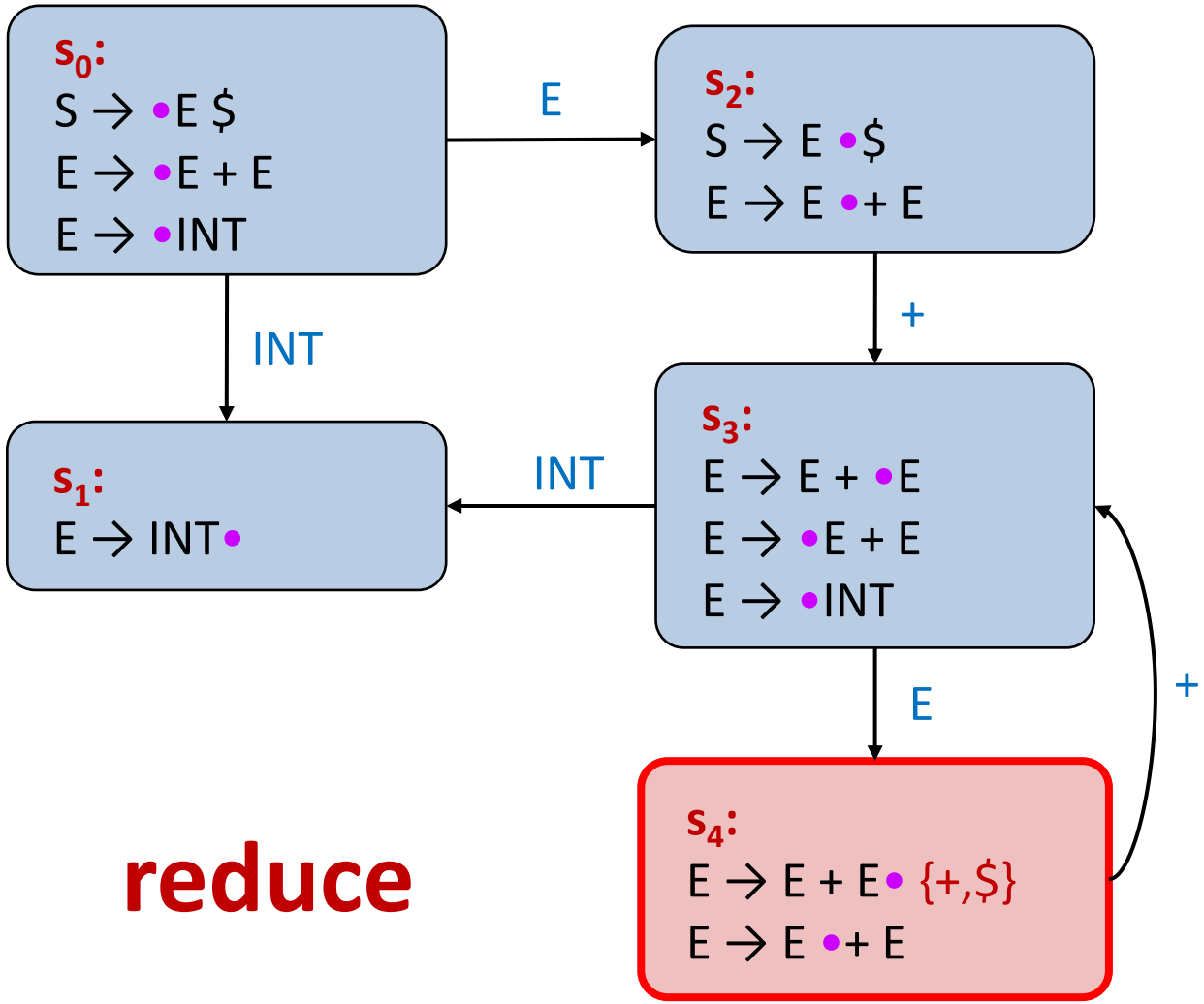
Stack: s_0 E s_2 + s_3 E s_4 + s_3

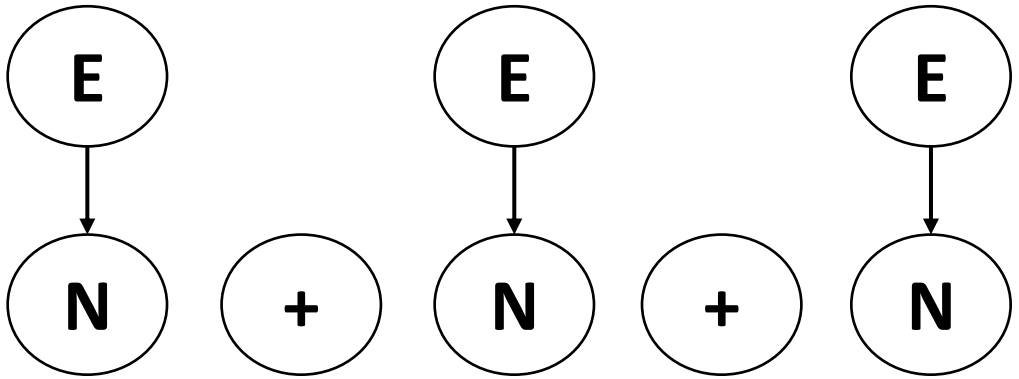




Input: 1 + 2 + 3$

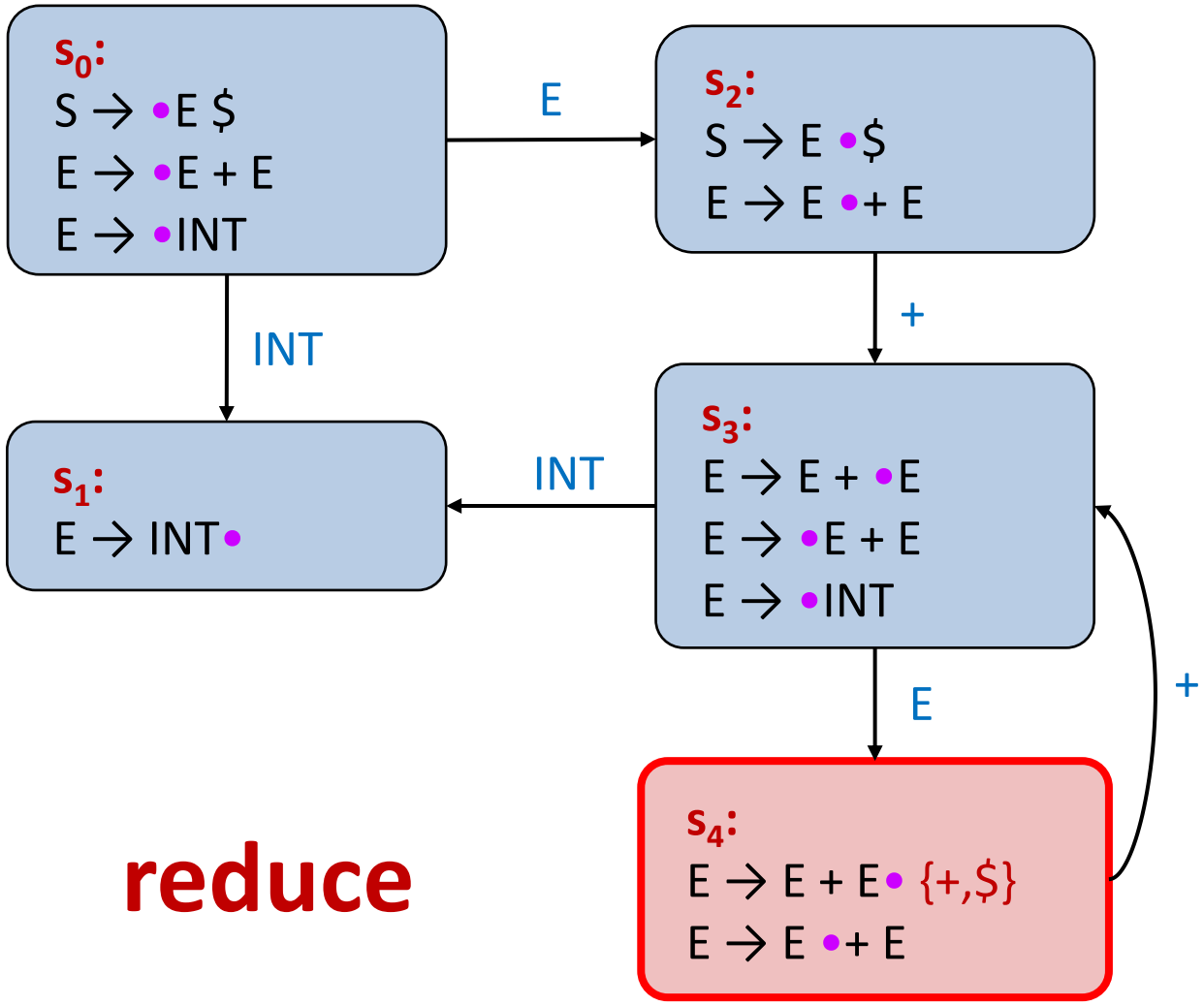
Stack: s_0 E s_2 + s_3 E s_4 + s_3 INT s_1

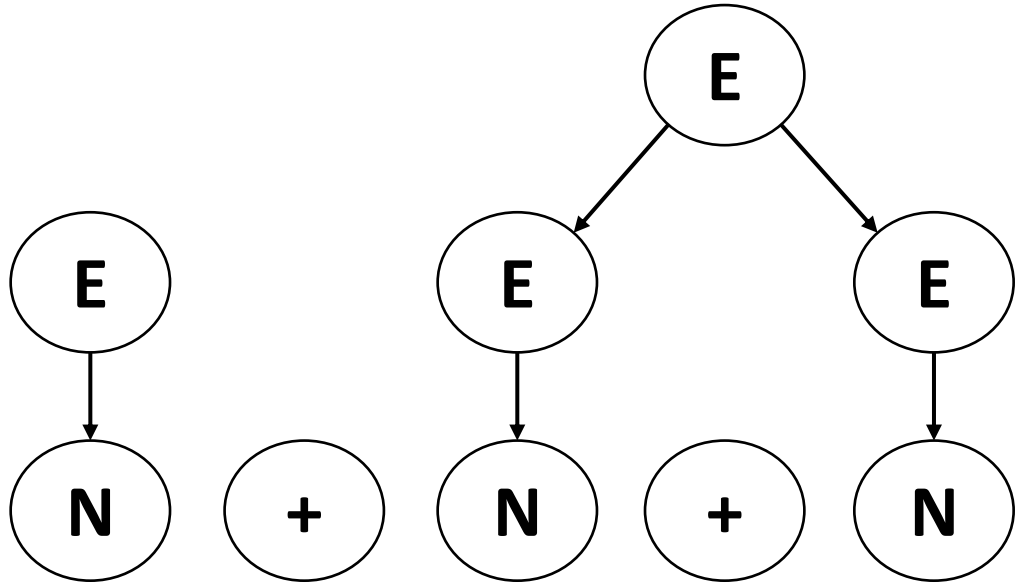




Input: 1 + 2 + 3$

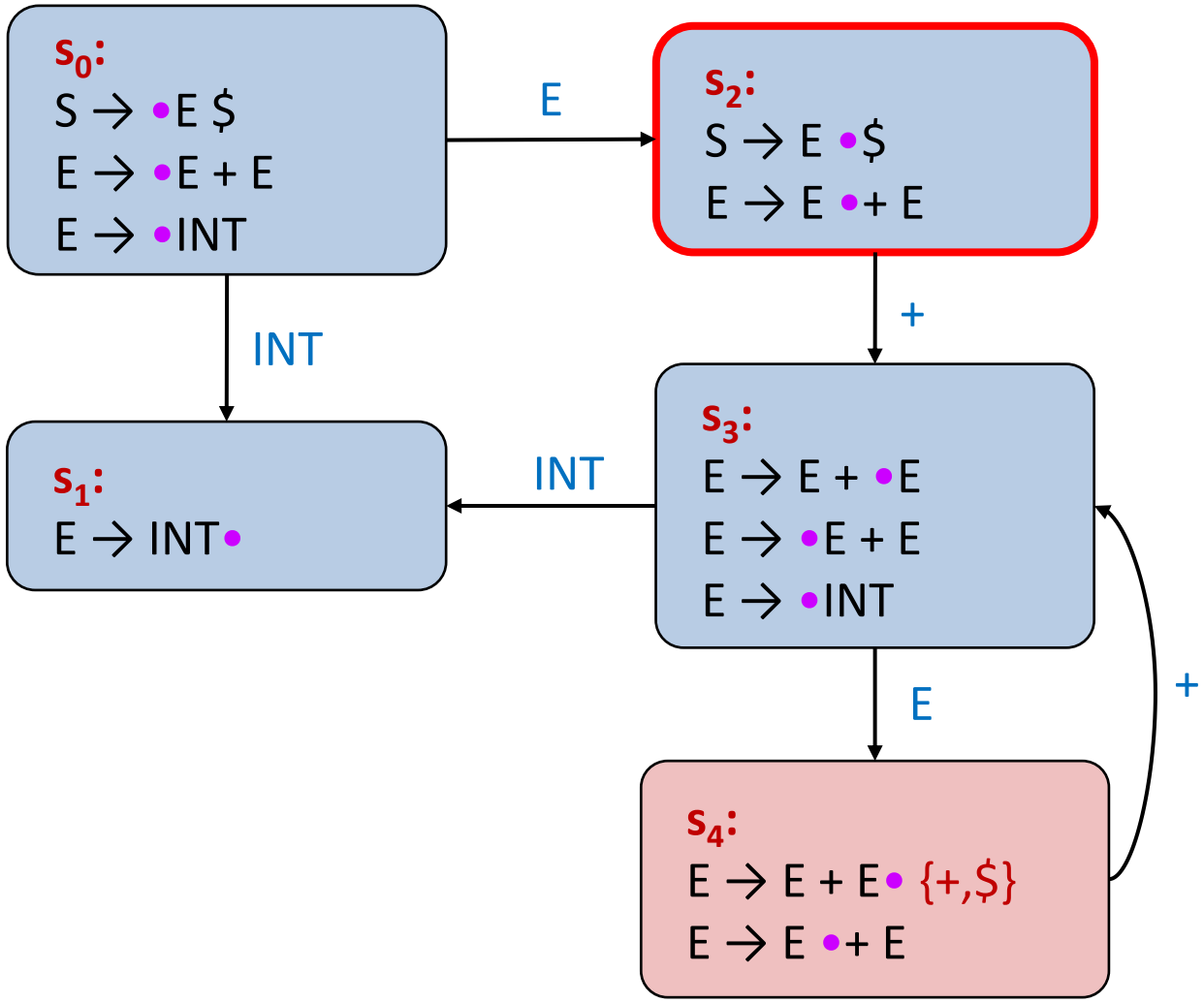
Stack: s_0 E s_2 + s_3 E s_4 + s_3 E s_4

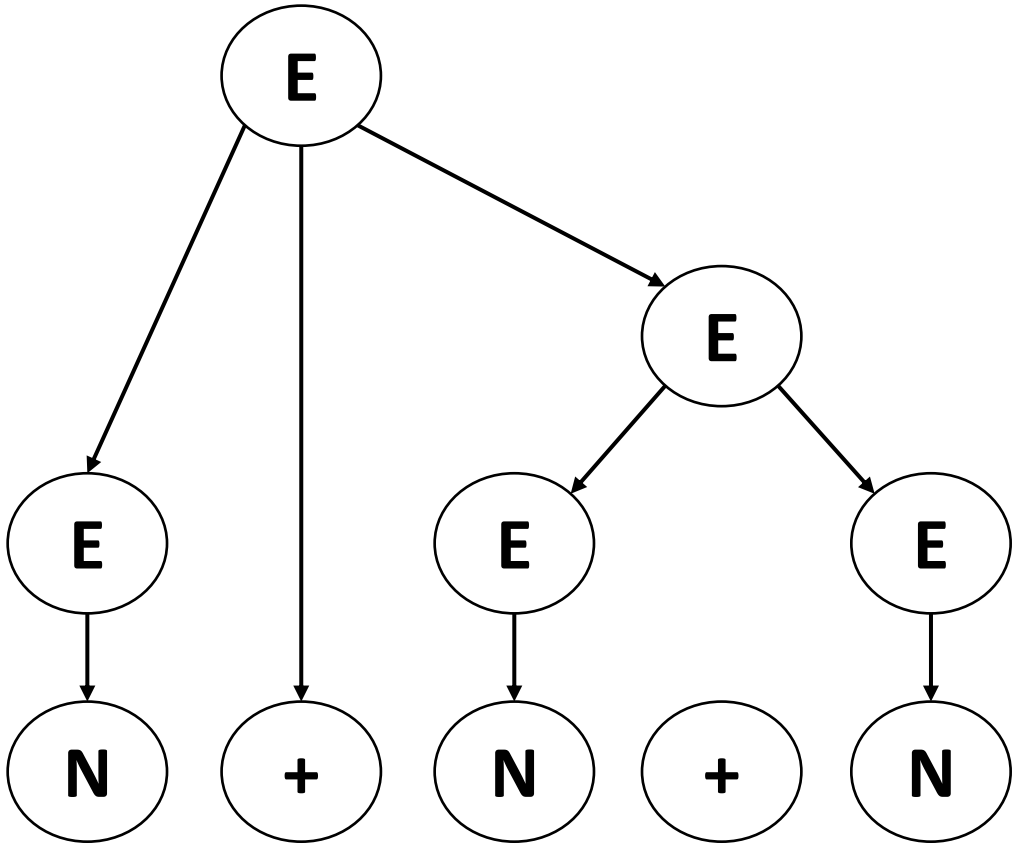




Input: 1 + 2 + 3$

Stack: s_0 E s_2 + s_3 E s_4





Input: 1 + 2 + 3$

Stack: s_0 E s_2

Resolving Conflicts – Precedence

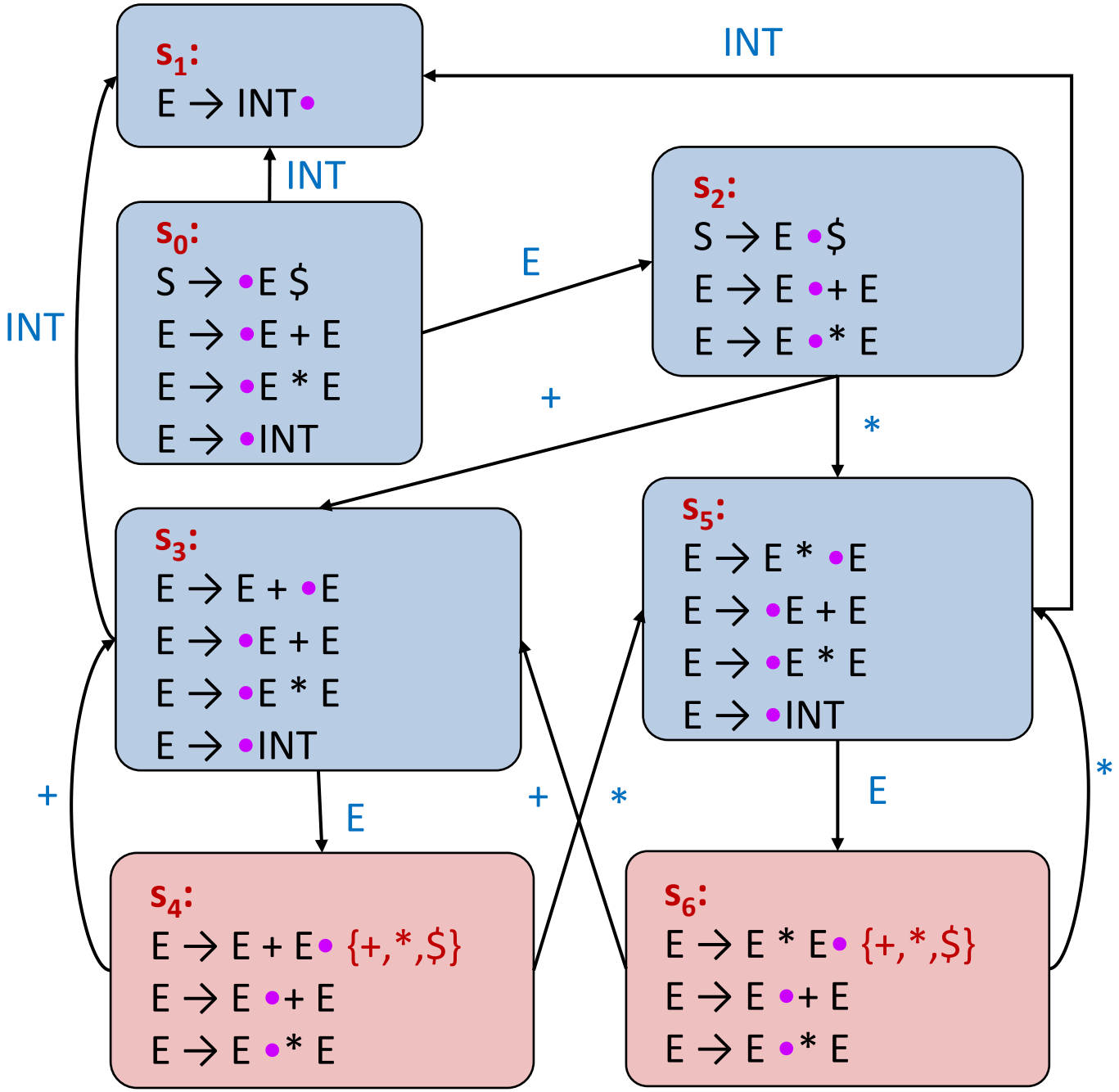
S → E $

E → E + E

E → E * E

F → INT

- What will be the SLR(1) automaton for this CFG?


s_1:
E → INT•
INT
INT
s_0:
S → •E $
E → •E + E
E → •E * E
E → •INT
E
s_2:
S → E •$
E → E •+ E
E → E •* E
INT
+
*
s_3:
E → E + •E
E → •E + E
E → •E * E
E → •INT
s_5:
E → E * •E
E → •E + E
E → •E * E
E → •INT
+
E
+
*
E
*
s_4:
E → E + E• {+,*,$}
E → E •+ E
E → E •* E
s_6:
E → E * E• {+,*,$}
E → E •+ E
E → E •* E

Resolving Conflicts – Precedence

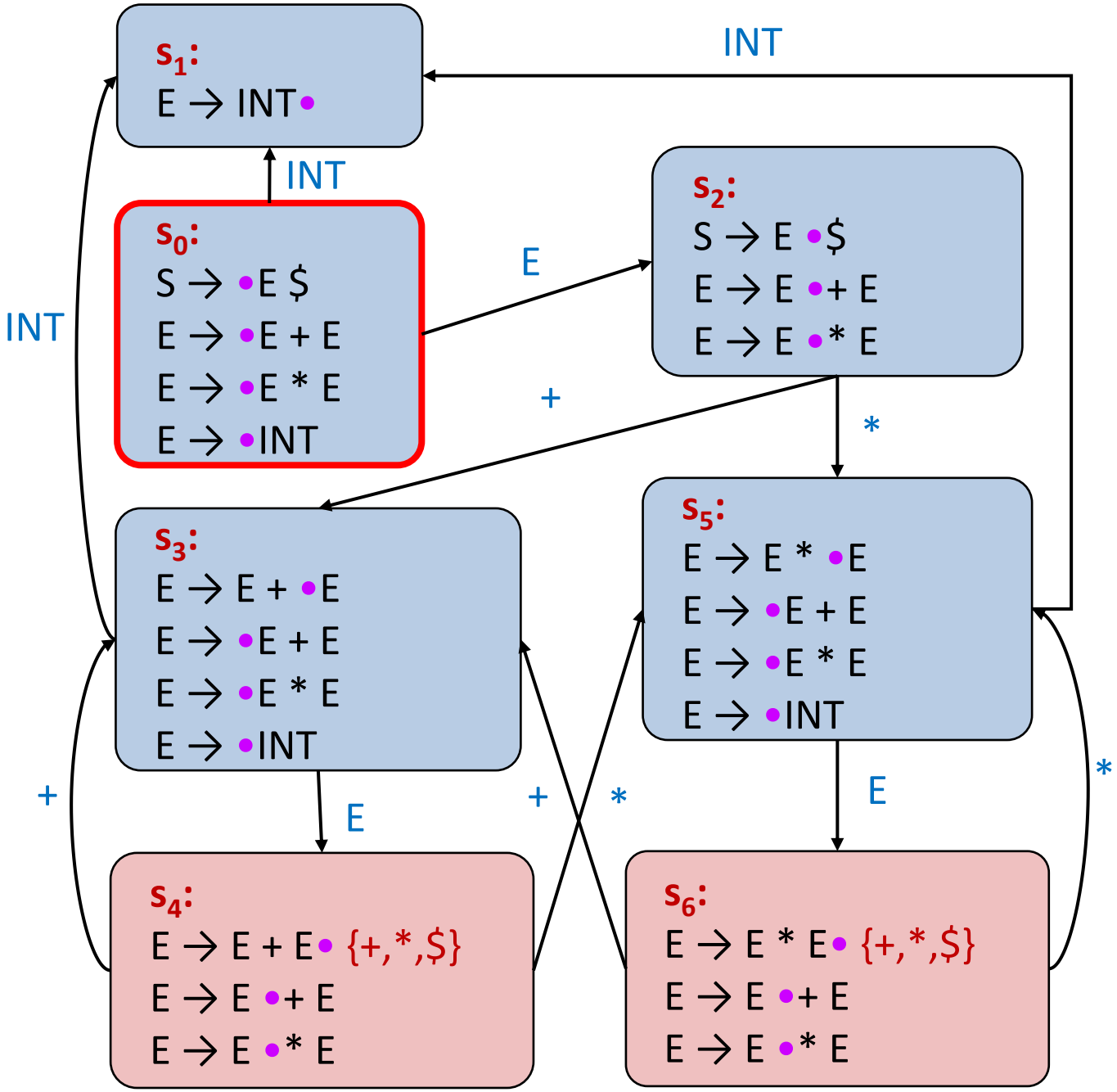
- When having a shift/reduce conflict:
  - $E_1 \rightarrow \alpha_1\ t_1\ \beta_1 \bullet$
  - $E_2 \rightarrow \alpha_2\ \bullet t_2\ \beta_2$

- If t_1 has higher precedence, **reduce**
- If t_2 has higher precedence, **shift**

Resolving Conflicts – Precedence

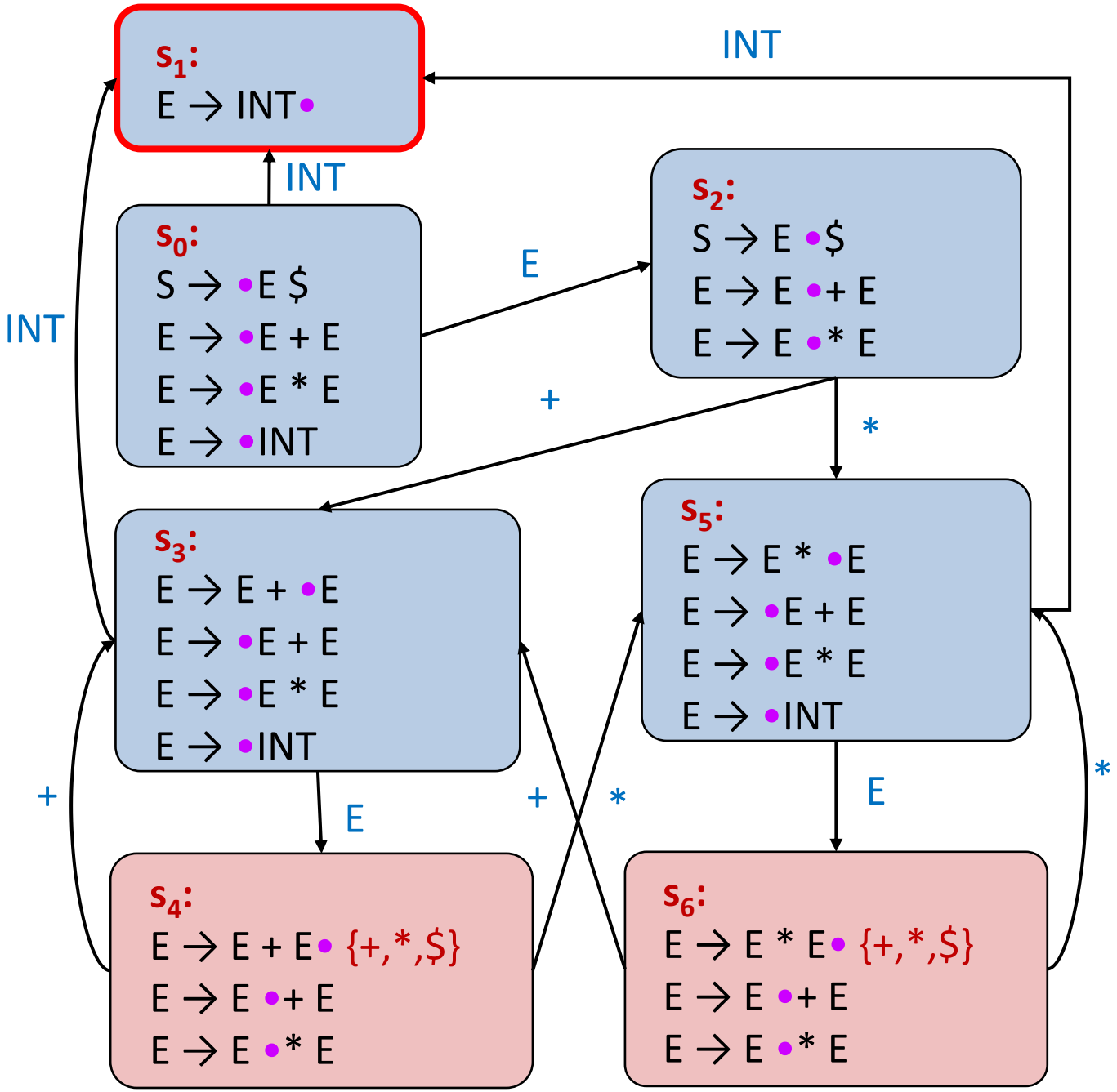
- In our case:
  - E → E + E•
  - E → E •* E

- Assuming multiplication has higher precedence:
  - Resolve by **shift**



Input: 3 + 4 * 8$

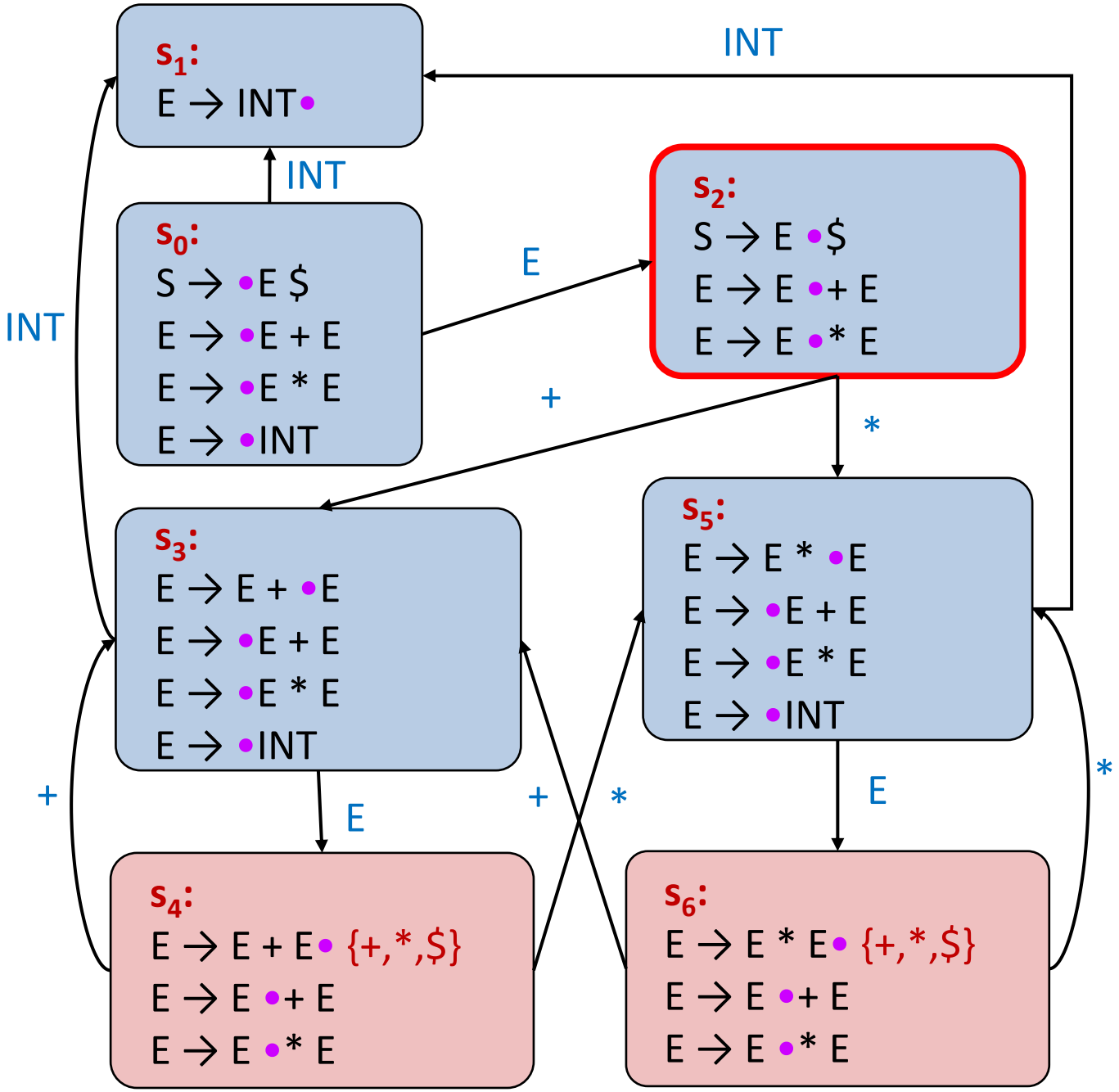
Stack: s_0

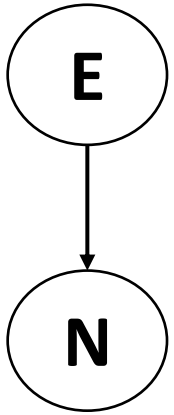


N

Input: 3 + 4 * 8$

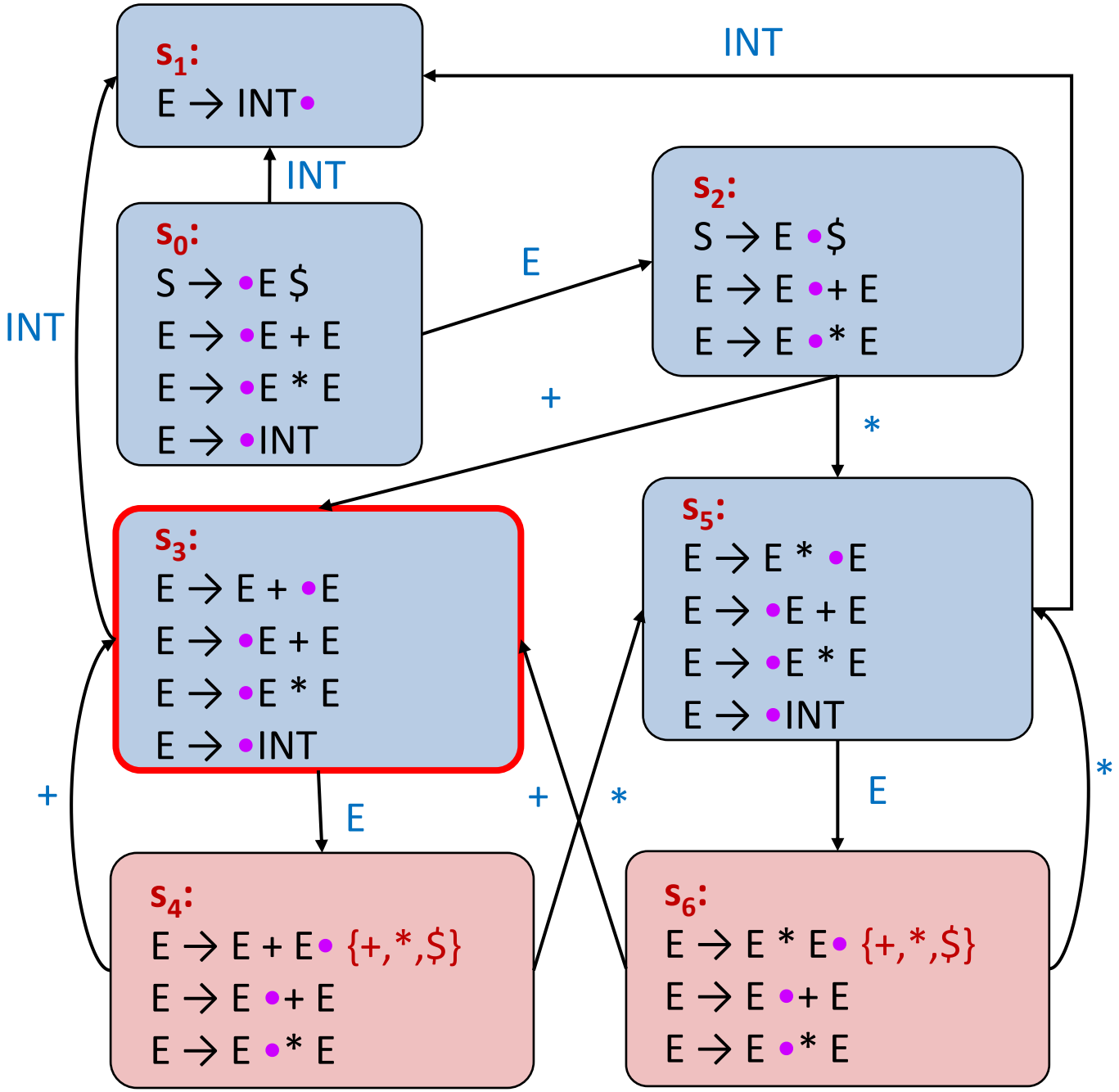
Stack: s_0 INT s_1

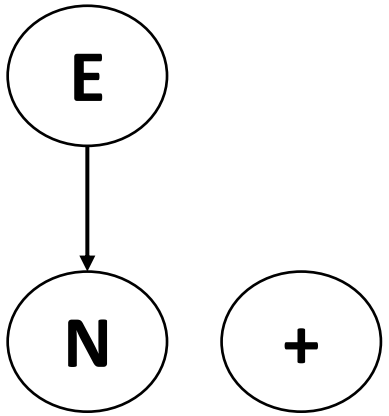




Input: 3 + 4 * 8$

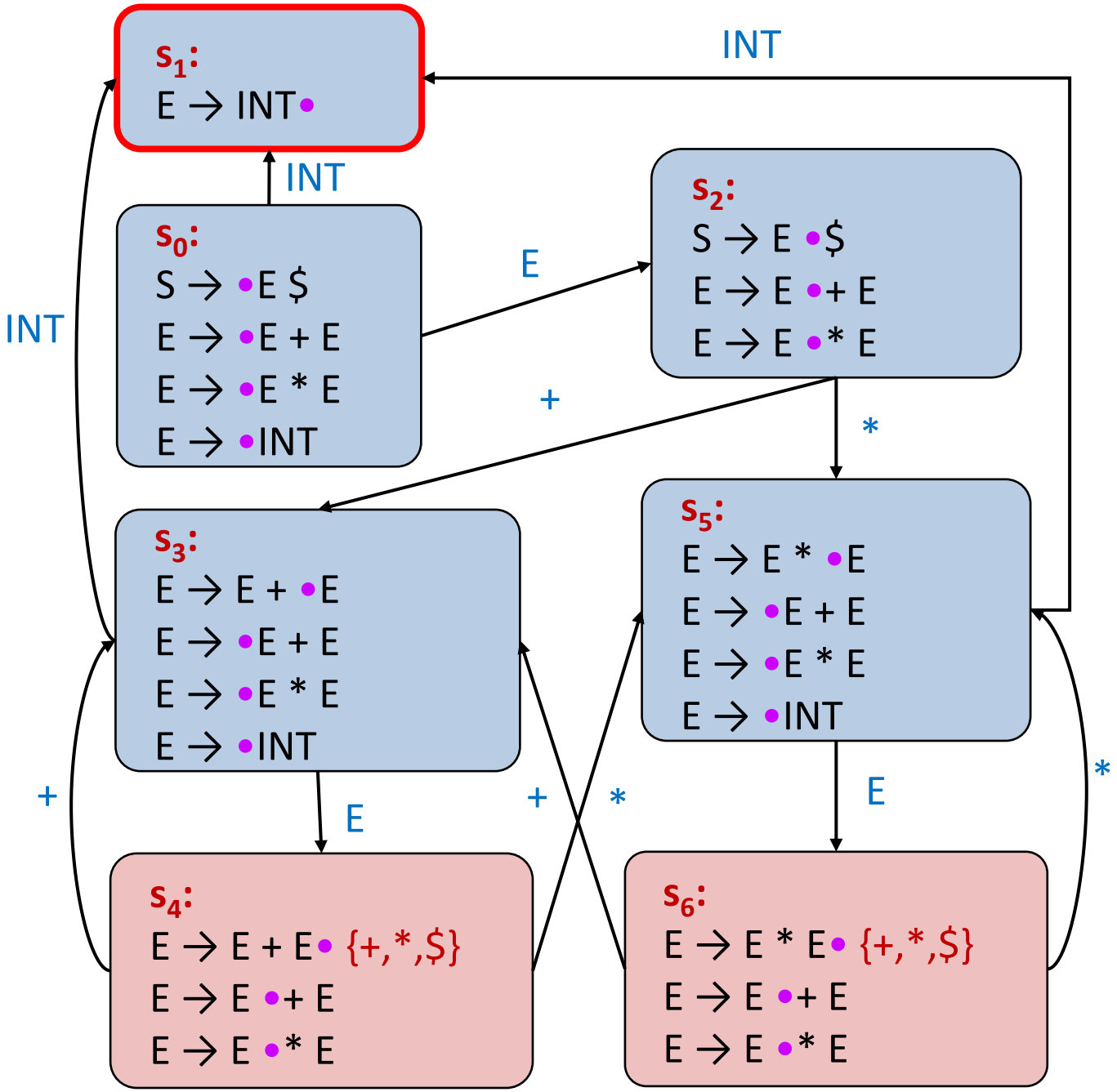
Stack: s_0 E s_2

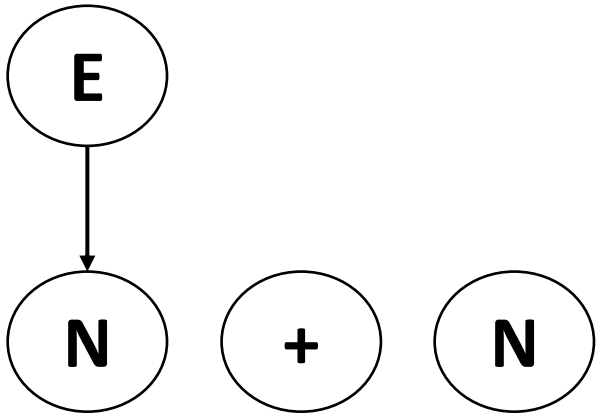




Input: 3 + 4 * 8$

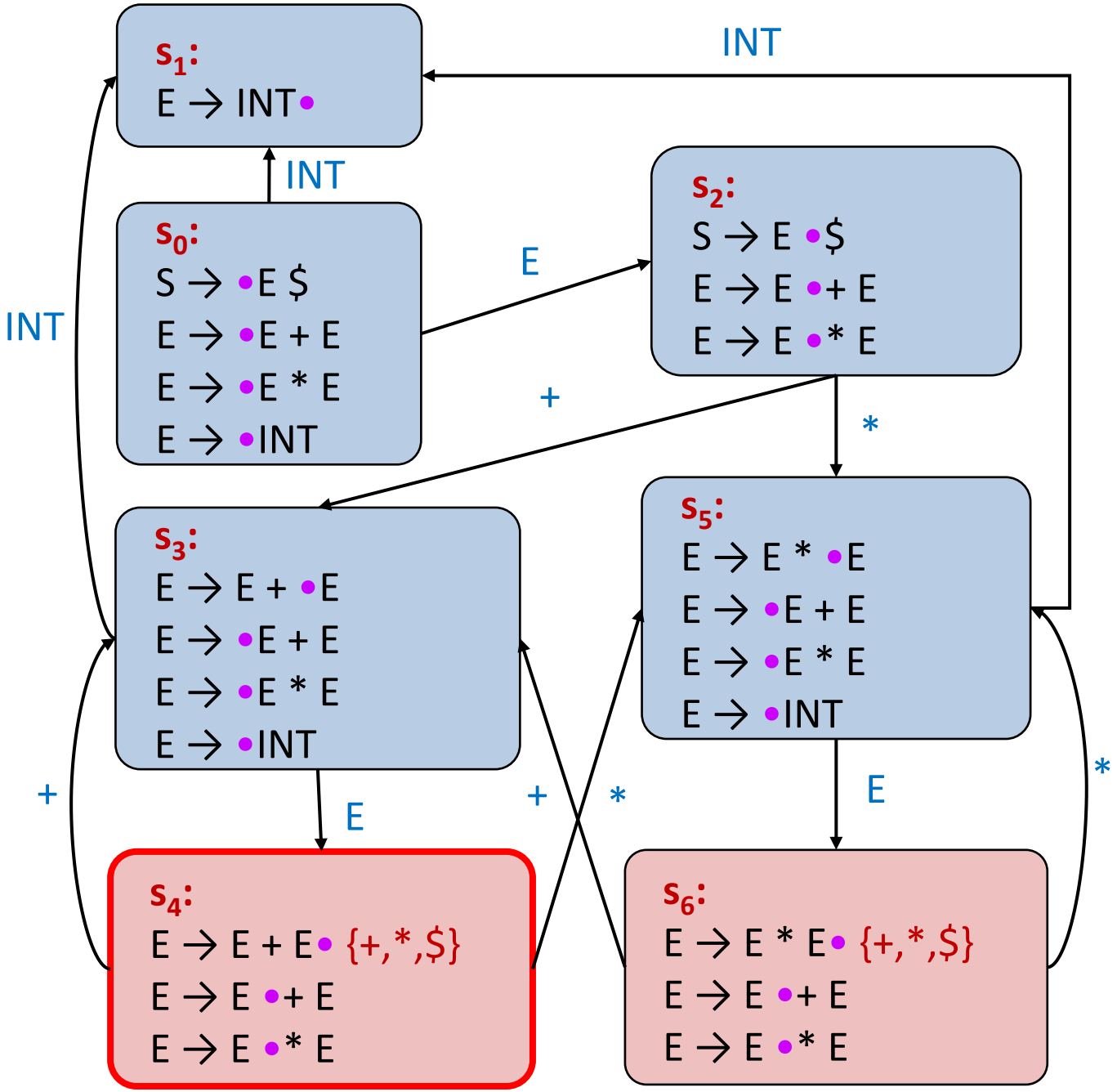
Stack: s_0 E s_2 + s_3



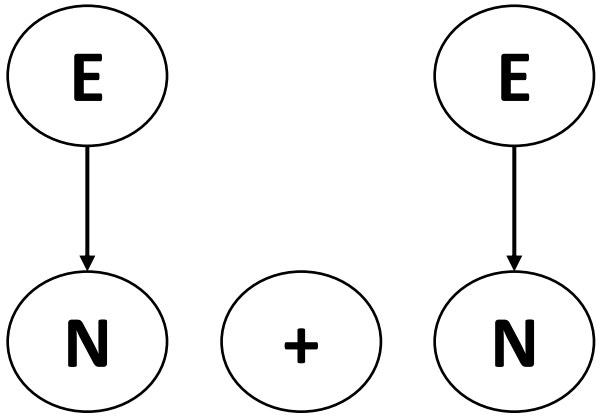


Input: 3 + 4 * 8$

Stack: s_0 E s_2 + s_3 INT s_1

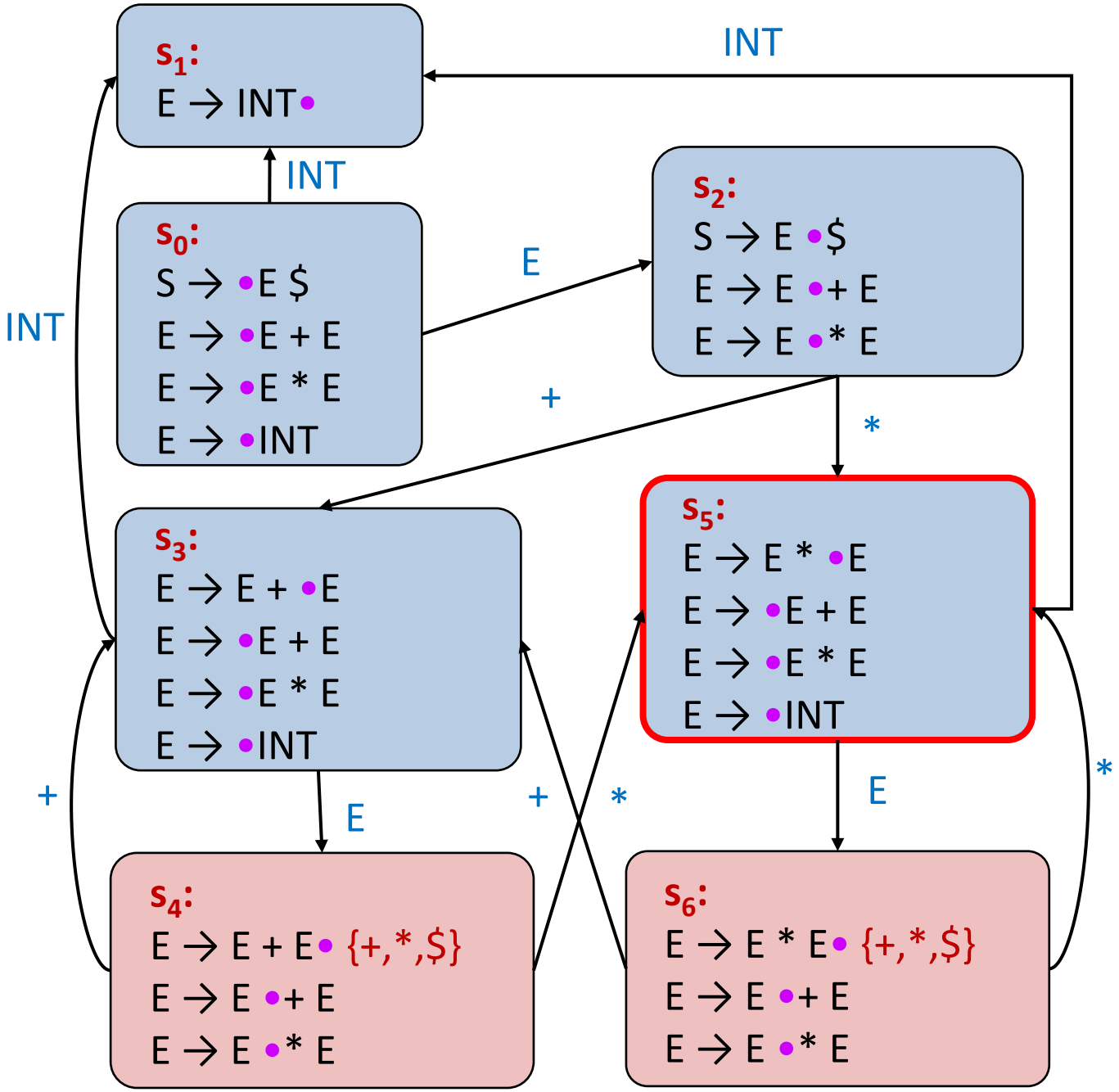


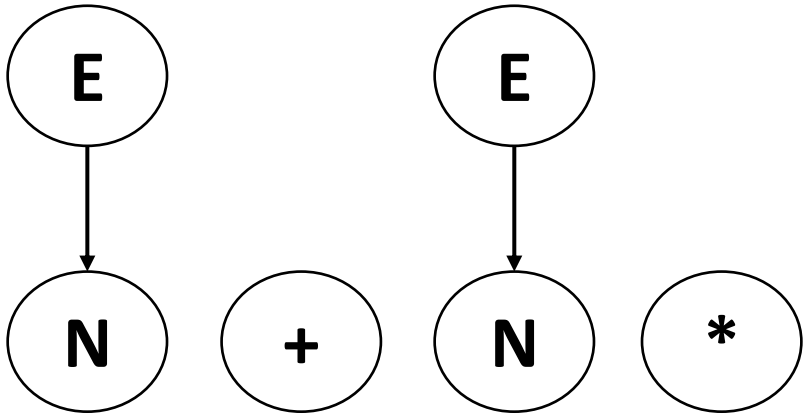
shift



Input: 3 + 4 * 8$

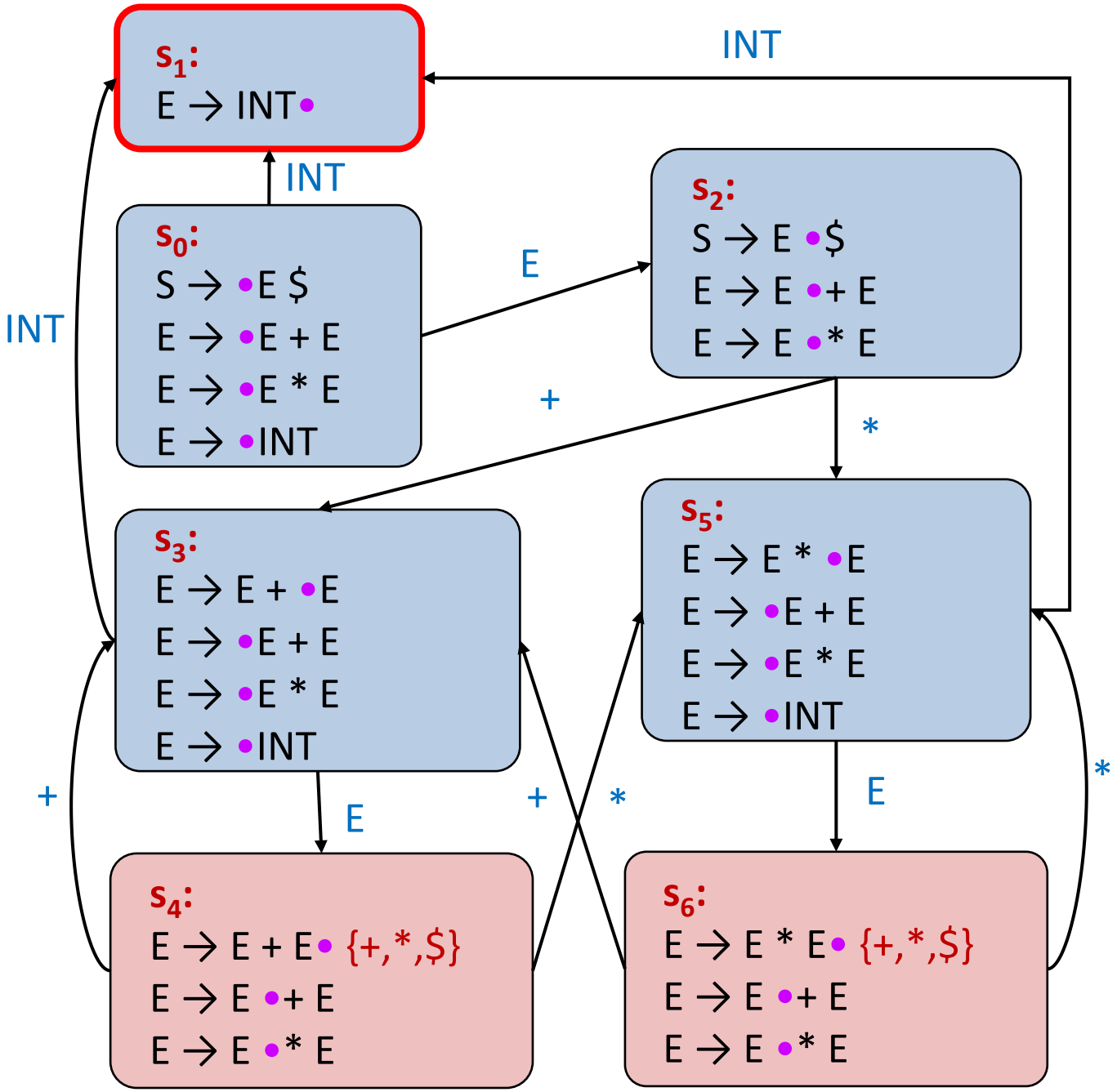
Stack: s_0 E s_2 + s_3 E s_4

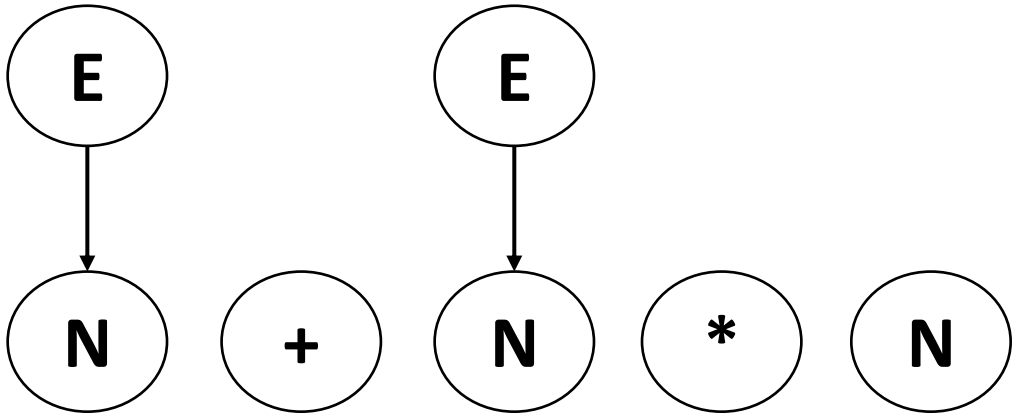




Input: 3 + 4 * 8$

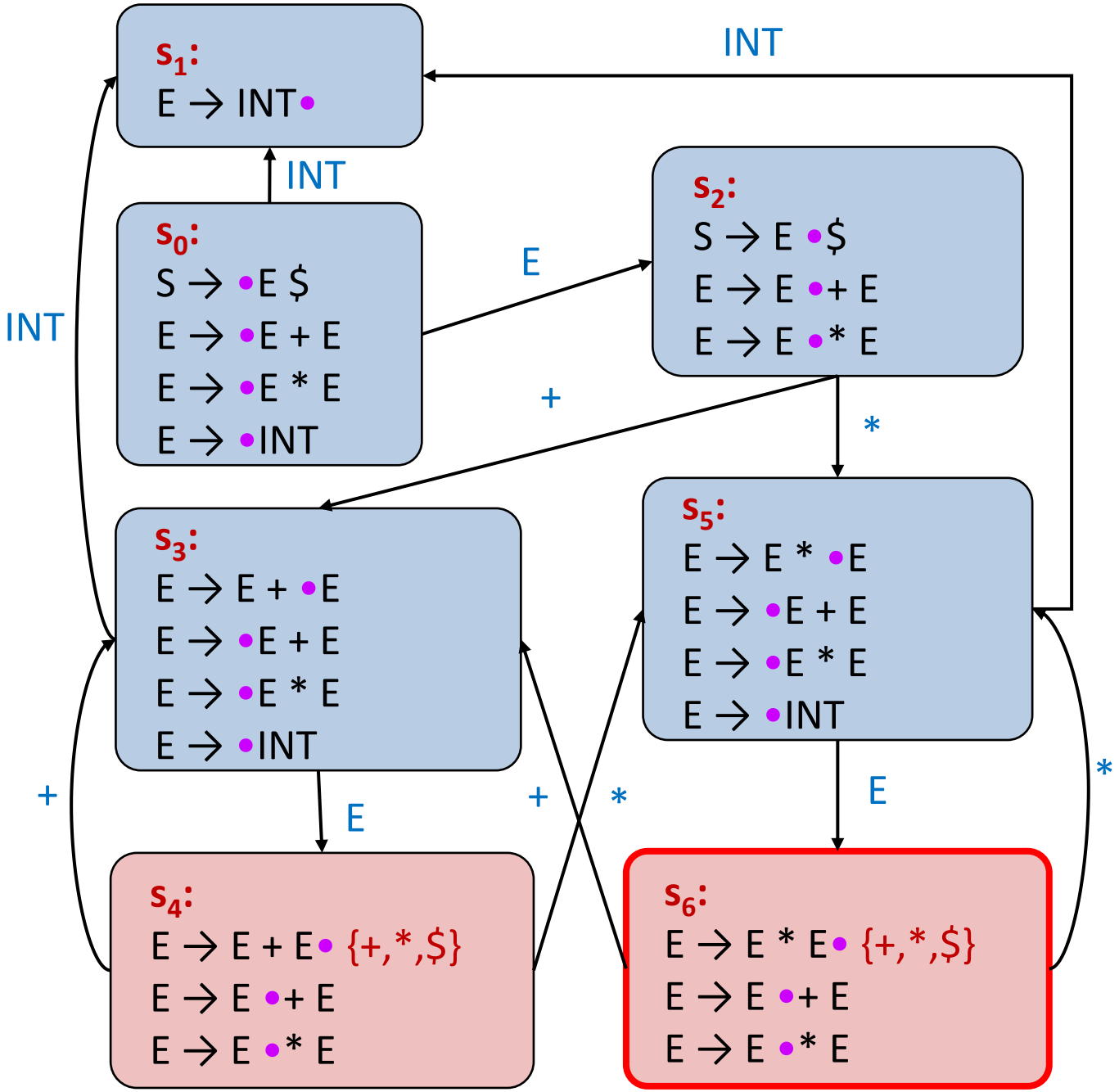
Stack: s_0 E s_2 + s_3 E s_4 * s_5



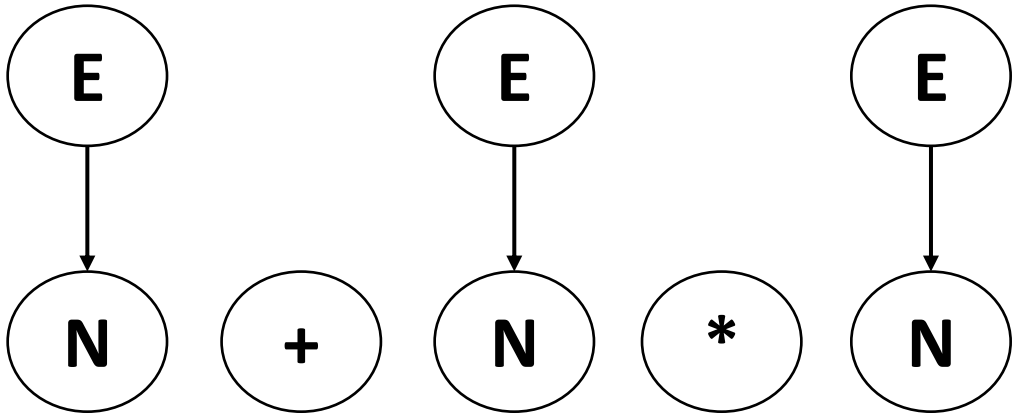


Input: 3 + 4 * 8$

Stack: s_0 E s_2 + s_3 E s_4 * s_5 INT s_1

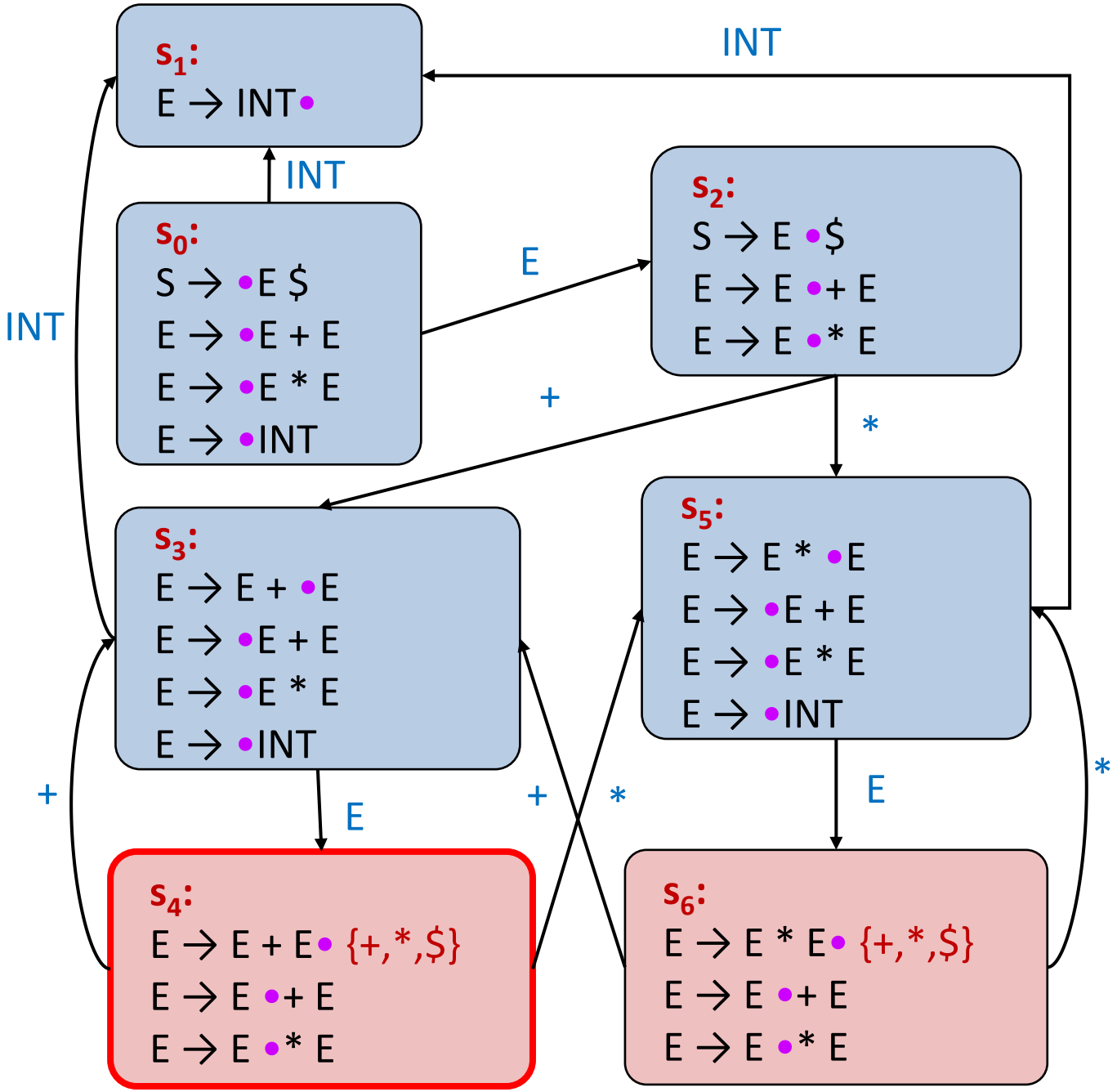


reduce

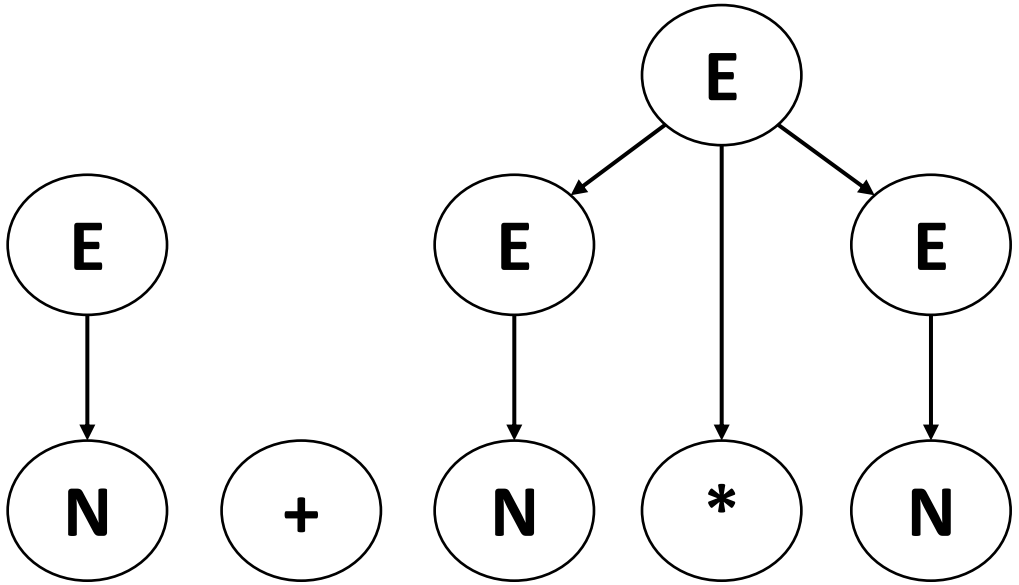


Input: 3 + 4 * 8$

Stack: s_0 E s_2 + s_3 E s_4 * s_5 E s_6

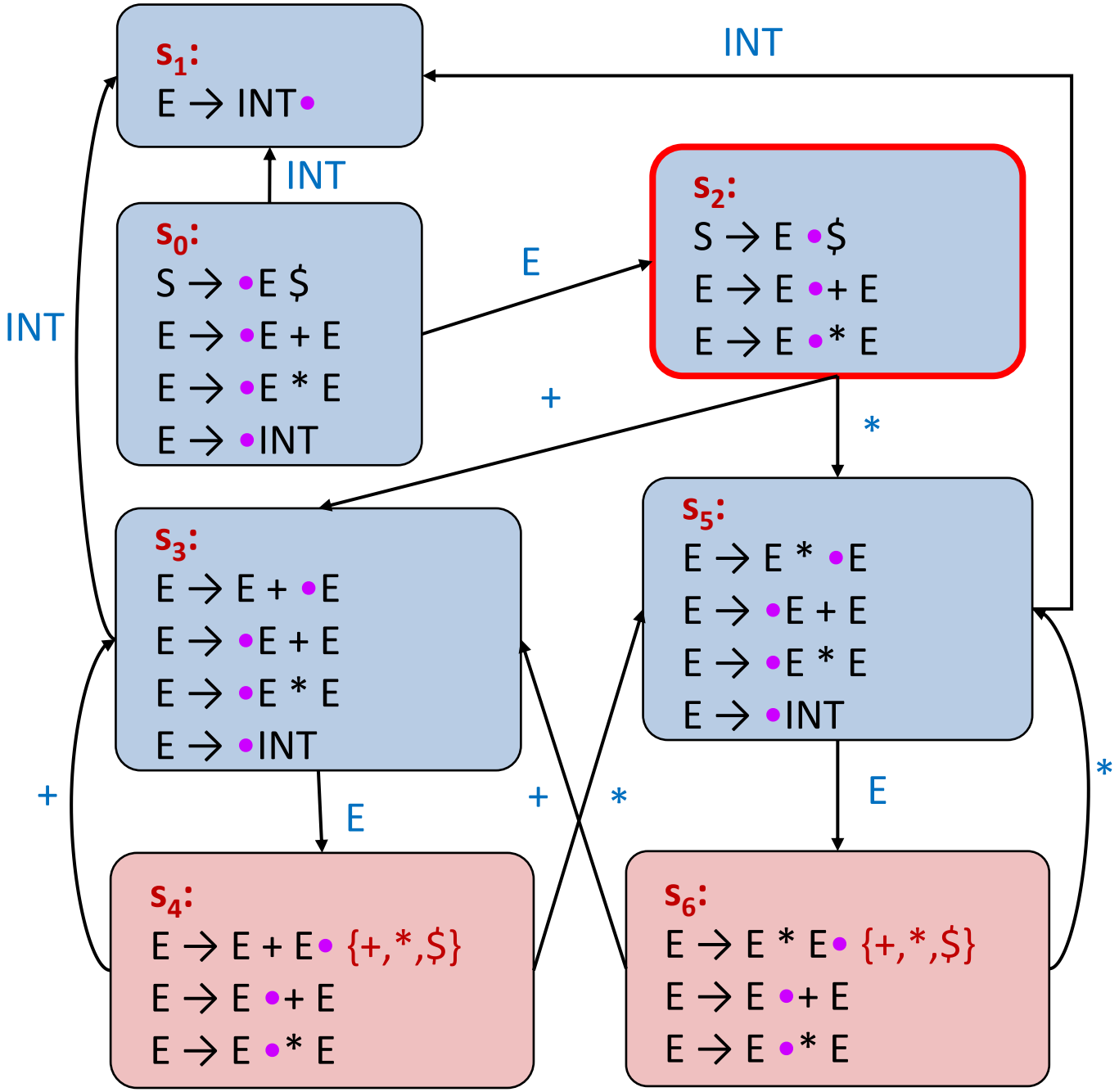


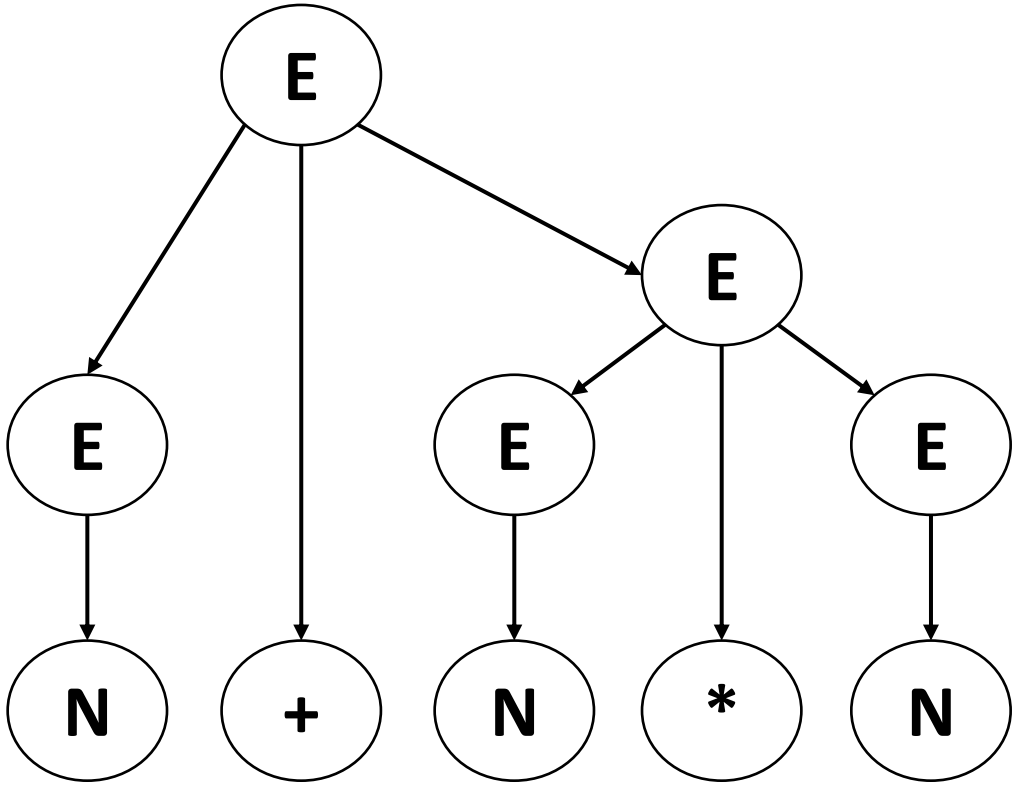
reduce



Input: 3 + 4 * 8$

Stack: s_0 E s_2 + s_3 E s_4





Input: 3 + 4 * 8$

Stack: s_0 E s_2

Precedence & Associativity in CUP

“When there is a shift/reduce conflict, the parser determines whether the terminal to be shifted has a higher precedence, or if the production to reduce by does. If the terminal has higher precedence, it is shifted

…

Associativities are also used to resolve shift/reduce conflicts, but only in the case of equal precedences.”

https://www2.cs.tum.edu/projects/cup/docs.php

CUP Precedence ≠ Operator Precedence

```
precedence left PLUS;
precedence left TIMES;

exp ::= INT
        | exp binop exp;

binop ::= PLUS | TIMES;
```

- **Operator precedence will not work correctly here**

"CUP also assigns each of its productions a precedence. That precedence is equal to the precedence of the last terminal in that production. If the production has no terminals, then it has lowest precedence."

- **We always shift!**