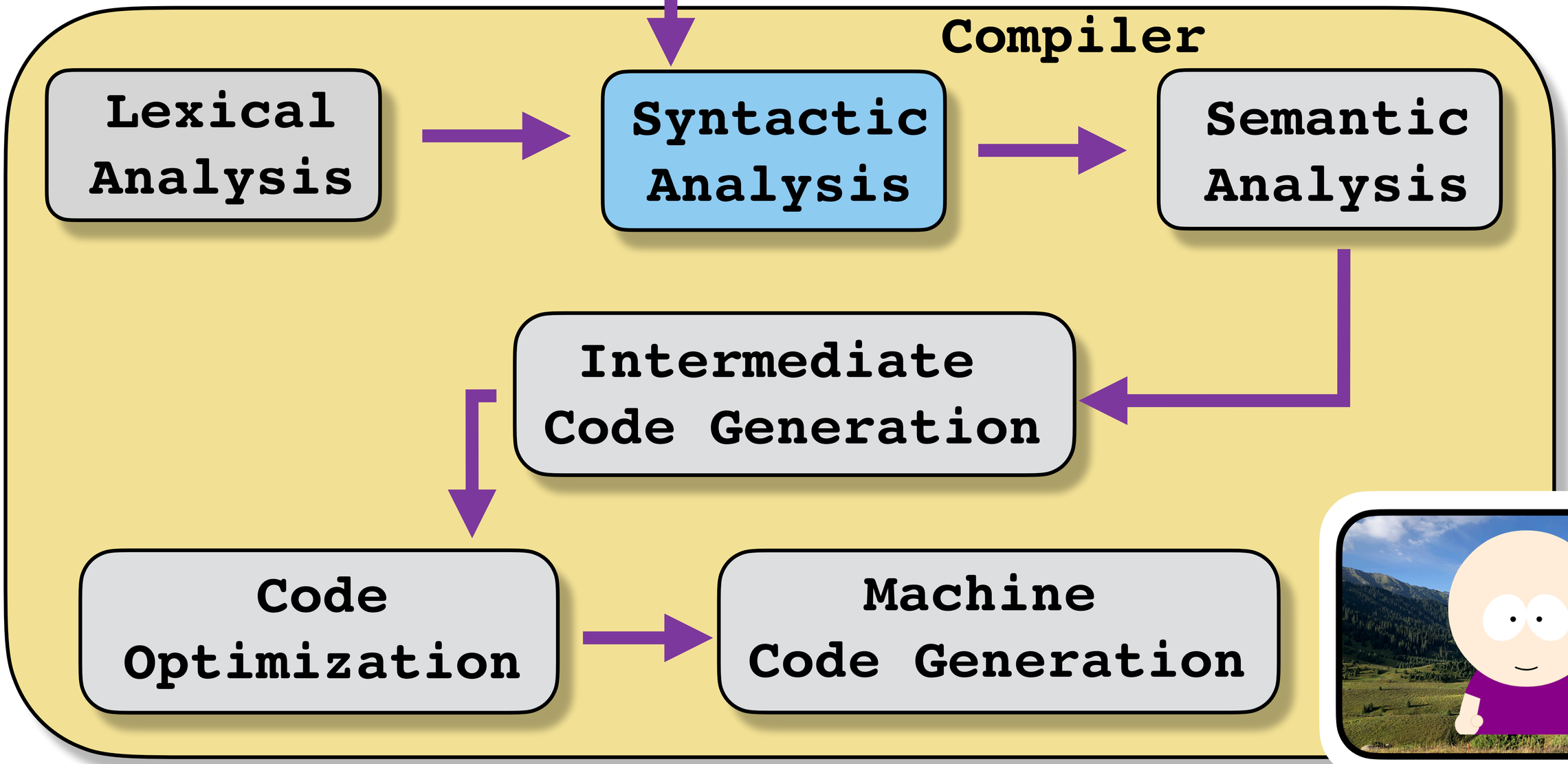
Parsing 2- Top-Down Parsing

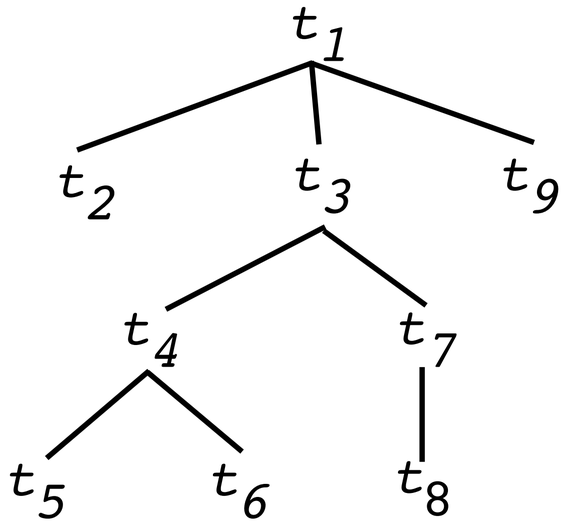
```
stmt→if
      ( expr )
      stmt
      [else stmt]
```



Top-down Parsing

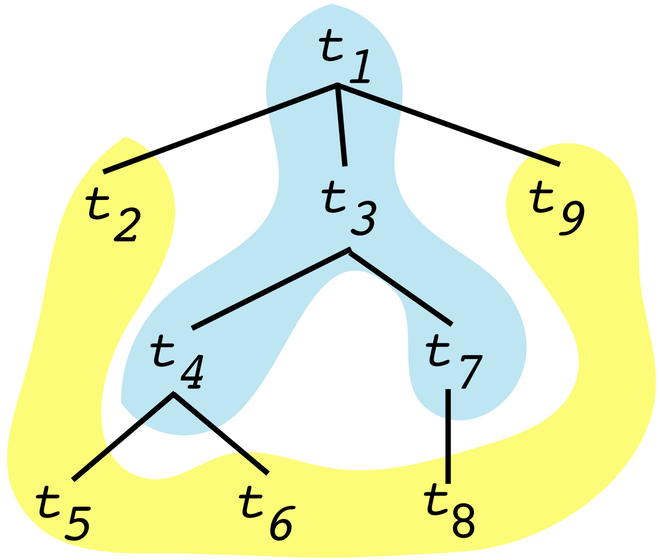
Top-Down Parsing

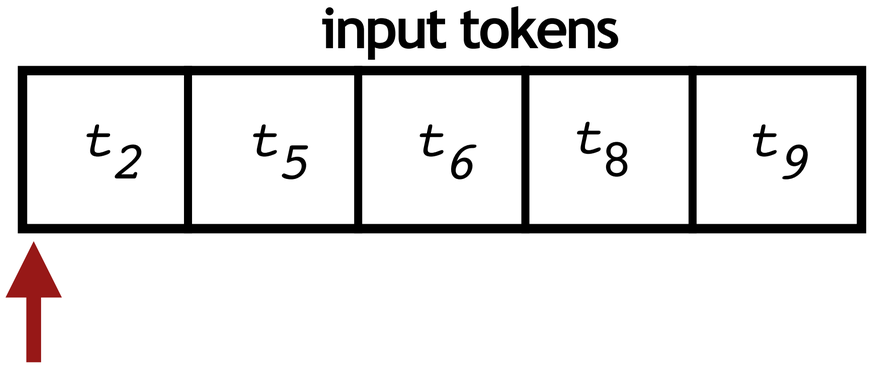
- The parse tree is constructed from the top from left to right.



Top-Down Parsing

- The terminals are seen in order of appearance in the token stream $(t_2, t_5, t_6, t_8, t_9)$.





```
PROCEDURE S ();
    IF curr_tok = if THEN
        match(if);
        E();
        match(then);
        S();
    ELSIF curr_tok = id THEN
        match(id); match(:=); E();
    ELSE syntax error ENDIF;
```

```
PROCEDURE match(t);
    IF curr_tok = t THEN
        curr_tok := next_token();
    ELSE
        syntax error
    ENDIF;
```

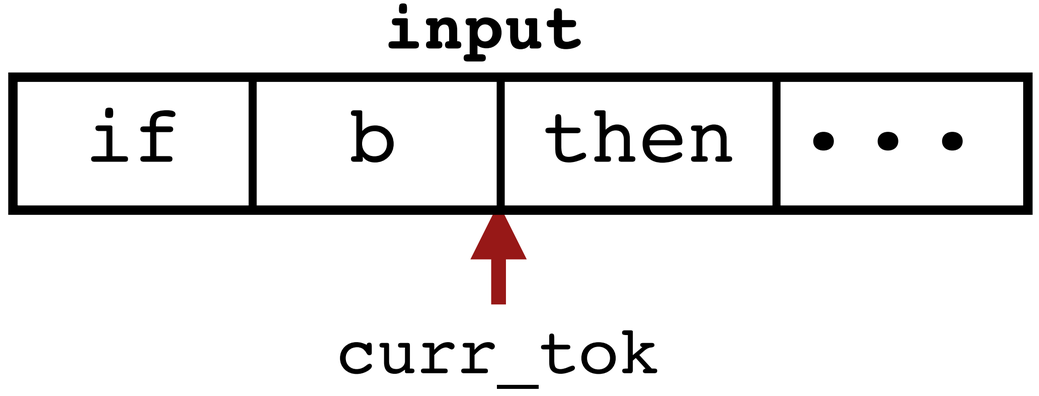
$S \rightarrow$ id := E |

if E then S

$E \rightarrow E + E$ |

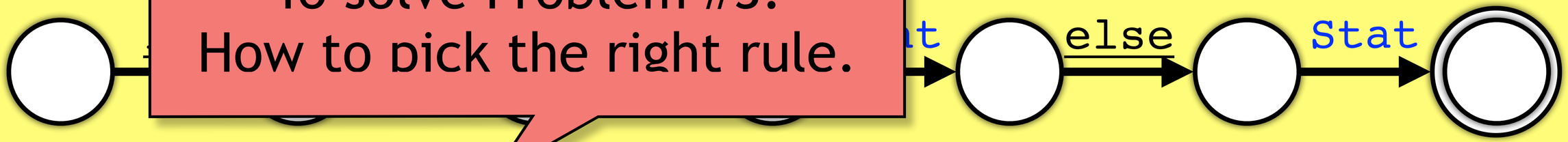
id |

num



Construct Recursive Descent Parser

1. Remove left recursion.
2. Left factor the grammar.
3. Const…ch production:



To solve Problem #3:
How to pick the right rule.

   1. Compute FIRST(A) for each grammar symbol A.
   2. Compute FOLLOW(A) for each nonterminal A.
4. Simplify the transition diagrams.
5. Construct the recursive procedures.

Problem #3: How to Pick the Right Rule?

```
PROCEDURE prog ();
    IF pick this rule? IN FIRST(stat) THEN
        stat();
    ELSIF pick this rule? IN FIRST(decl) THEN
        decl();
    ELSE syntax error ENDIF;
END;

PROCEDURE stat (); ... END;

PROCEDURE decl (); ... END;
```

```
prog → decl |
       stat

stat → if      ... |
       id      ( ) |
       while   ...

decl → int     id  |
       real    id
```

Computing FIRST Sets

FIRST Sets

FIRST(α) is the set of terminals that begin strings derived from α.

- FIRST(stat)= {if, id, while}
- FIRST(decl)= {int, real}

```
prog → decl |
       stat

stat → if    ... |
       id    ( ) |
       while ...

decl → int   id |
       real  id
```

Using FIRST Sets

- FIRST(stat)= {if, id, while}
- FIRST(decl)= {int, real}

```
PROCEDURE prog ();
    IF curr_tok ∈ {if, id, while} THEN
        stat();
    ELSIF curr_tok ∈ {int, real} THEN
        decl();
    ELSE syntax error ENDIF;
END;

PROCEDURE stat (); ... END;

PROCEDURE decl (); ... END;
```

FIRST Sets Definition

FIRST(α) is the set of terminals that begin strings derived from α:

FIRST(α) = {x | x is a terminal and α ⇒ xβ}

- FIRST(E) = {(, id}

E → T E′
E′→ + T E′
E′→ ϵ
T → F T′
T′→ * F T′
T′→ ϵ
F → (E)
F → id

What is FIRST(α)?

- If α is a terminal:

  FIRST(x) = {x}

- If α starts with a terminal:

  FIRST(xABC) = {x}

- If A derives ϵ, i.e. ϵ ∈ FIRST(A):

  FIRST(ABC) = (FIRST(A)-{ϵ}) ∪
  (FIRST(B)-{ϵ})

- If α derives ϵ,

  i.e. ϵ ∈ FIRST(A), ϵ ∈ FIRST(B), ϵ ∈ FIRST(C):

  FIRST(ABC) = FIRST(A)∪FIRST(B)∪FIRST(C)∪{ϵ}

Computing FIRST Sets

E ⇒ T E'

⇒ F T' E'

⇒ (E) T' E'

E ⇒ T E'

⇒ F T' E'

⇒ id T' E'

- FIRST(E) = {(, id}

E → T E′

E′→ + T E′

E′→ ∈

T → F T′

T′→ * F T′

T′→ ∈

F → (E)

F → id

Computing FIRST Sets

- FIRST(E) = {(, id}
- FIRST(T) = {(, id}
- FIRST(F) = {(, id}
- FIRST(E') = {+, ϵ}
- FIRST(T') = {*, ϵ}

- FIRST(+) = {+}
- FIRST(*) = {*}
- FIRST(() = {(}
- FIRST()) = {)}
- FIRST(id) = {id}

E → T E′
E′→ + T E′
E′→ ϵ
T → F T′
T′→ * F T′
T′→ ϵ
F → (E)
F → id

Computing FIRST Sets (no $A \rightarrow \epsilon$ rules)

```
1.FOR each non-terminal A DO
      FIRST(A) = {}
2.FOR each terminal t DO
      FIRST(t)
3.REPEAT until no more changes:
     FOR each production A → Y1 ... Yk DO
         FIRST(A) ∪= FIRST(Y1);
```

S ∪= T means S = S ∪ T

Example 1 (Intuition)

FIRST(S)={x,y,z}

FIRST(A)={x,y,z}

FIRST(B)={ ,y,z}

S → A B

S → y S

A → x B

A → B B

B → y

B → z B

- S ⇒ A B ⇒ x B ⇒ x.B ⇒ ...
- S ⇒ A B ⇒ B B ⇒ y B ⇒ y.B ⇒ ...
- S ⇒ A B ⇒ B B B ⇒ z B B ⇒ z.B B B ⇒ ...

Example 1 (no $A \rightarrow \epsilon$ rules)

1. FIRST(S)=FIRST(A)=FIRST(B)={}
2. FIRST(x)={x}; FIRST(y)={y}; FIRST(z)={z};
3. **REPEAT** until no more changes:
   FOR each rule $A \rightarrow Y_1 \ldots Y_k$ **DO**
   FIRST(A) ∪= FIRST(Y_1);

FIRST(S)={ ,y, }
FIRST(A)={x, , }
FIRST(B)={ ,y,z}

1) FIRST(S) ∪= FIRST(A) = {}
2) FIRST(S) ∪= FIRST(y) = {y}
3) FIRST(A) ∪= FIRST(x) = {x}
4) FIRST(A) ∪= FIRST(B) = {x}
5) FIRST(B) ∪= FIRST(y) = {y}
6) FIRST(B) ∪= FIRST(z) = {y,z}

S → A B
S → y S
A → x B
A → B B
B → y
B → z B

Example 1 (no A→ϵ rules)

FIRST(S)={x,y, }
FIRST(A)={x,y,z}
FIRST(B)={ ,y,z}

7) FIRST(S) U= FIRST(A) = {x,y}
8) FIRST(S) U= FIRST(y) = {x,y}
9) FIRST(A) U= FIRST(x) = {x}
10) FIRST(A) U= FIRST(B) = {x,y,z}
11) FIRST(B) U= FIRST(y) = {y,z}
12) FIRST(B) U= FIRST(z) = {y,z}

S → A B
S → y S
A → x B
A → B B
B → y
B → z B

Example 1 (no A→ϵ rules)

FIRST(S)={x,y,z}
FIRST(A)={x,y,z}
FIRST(B)={ ,y,z}

13)FIRST(S) U= FIRST(A) = {x,y,z}
14)FIRST(S) U= FIRST(y) = {x,y,z}
15)FIRST(A) U= FIRST(x) = {x,y,z}
16)FIRST(A) U= FIRST(B) = {x,y,z}
17)FIRST(B) U= FIRST(y) = {y,z}
18)FIRST(B) U= FIRST(z) = {y,z}

S → A B
S → y S
A → x B
A → B B
B → y
B → z B

Example 1 (no A→ϵ rules)

FIRST(S)={x,y,z}
FIRST(A)={x,y,z}
FIRST(B)={ ,y,z}

19)FIRST(S) U= FIRST(A) = {x,y,z}
20)FIRST(S) U= FIRST(y) = {x,y,z}
21)FIRST(A) U= FIRST(x) = {x,y,z}
22)FIRST(A) U= FIRST(B) = {x,y,z}
23)FIRST(B) U= FIRST(y) = {y,z}
24)FIRST(B) U= FIRST(z) = {y,z}

S → A B
S → y S
A → x B
A → B B
B → y
B → z B

Example 2
(Intuition)

FIRST(S)={x,y,z,w, }

FIRST(A)={x, , , ,ϵ}

FIRST(B)={x,y, , ,ϵ}

FIRST(C)={ , ,z, , }

S → A B C

S → w

A → x A

A → ϵ

B → y B

B → A y

B → ϵ

C → z

- S ⇒ A B C ⇒ x A B C
- S ⇒ A B C ⇒ y B C
- S ⇒ A B C ⇒ z

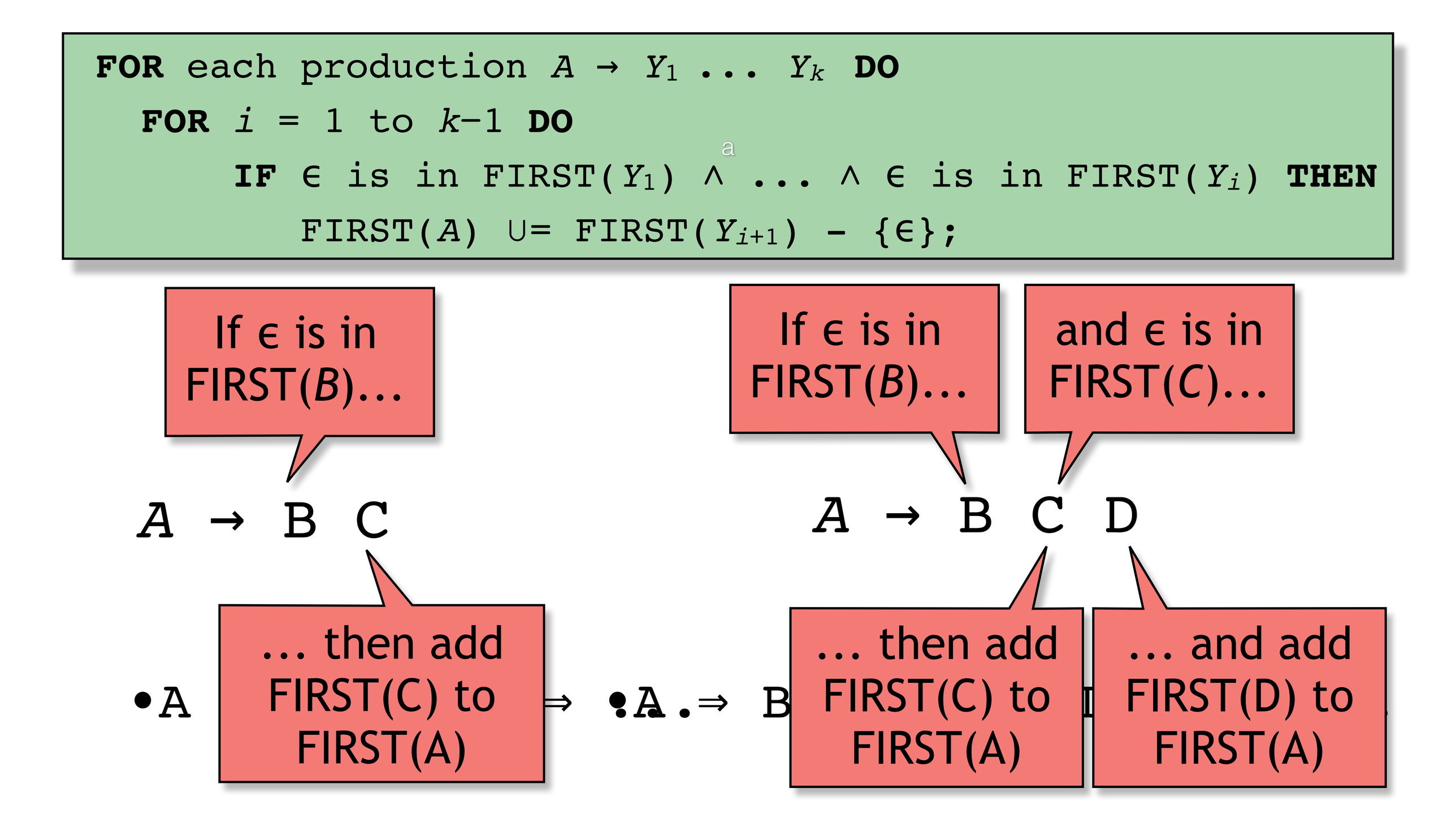
Computing FIRST Sets (with A→ϵ rules)

```
1.FOR each non-terminal A    DO FIRST(A) = {}
2.FOR each terminal t        DO FIRST(t) = {t}
3.FOR each produ                          }
4.REPEAT until n
    FOR each prod                    A→ϵ DO
                                  ∧ ϵ is in FIRST(Yi) THEN
        FIRST(A) ∪= FIRST(Yi+1) - {ϵ};
IF ϵ is in FIRST(Y1) ∧ ... ∧ ϵ is in FIR
    FIRST(A) ∪= {ϵ};
```

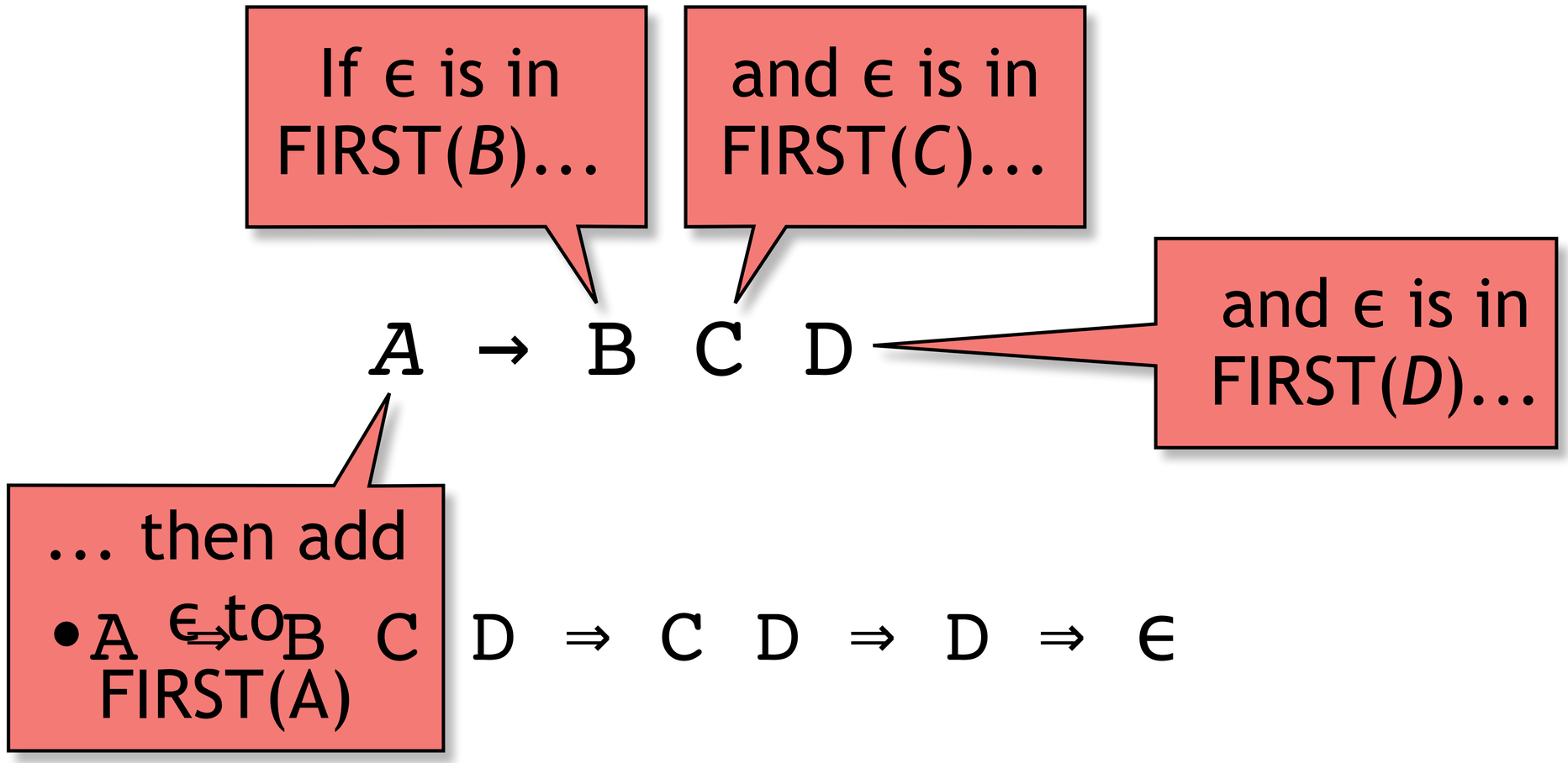
IF $Y_1 \ldots Y_i$ all derive ϵ in
$A \rightarrow Y_1\ Y_2 \ldots Y_i\ Y_{i+1} \ldots Y_k$
THEN
add FIRST(Y_{i+1}) to FIRST(A)

IF $Y_1 \ldots Y_k$ all derive ϵ THEN
add ϵ to FIRST(A)

```
FOR each production A → Y1 ... Yk DO
   FOR i = 1 to k–1 DO
        IF ϵ is in FIRST(Y1) ∧ ... ∧ ϵ is in FIRST(Yi) THEN
            FIRST(A) ∪= FIRST(Yi+1) – {ϵ};
```



```
FOR each production A → Y1 ... Yk DO
   IF ϵ is in FIRST(Y1) ∧ ... ∧ ϵ is in FIRST(Yk) THEN
        FIRST(A) ∪= {ϵ};
```



Example 2
(with A→ϵ rules)

FIRST(S)={ , , ,w, }

FIRST(A)={x, , , ,ϵ}

FIRST(B)={x,y, , ,ϵ}

FIRST(C)={ , ,z, , }

S → A B C

S → w

A → x A

A → ϵ

B → y B

B → A y

B → ϵ

C → z

1)FIRST(S)∪=FIRST(A)∪FIRST(B)∪FIRST(C)={}

2)FIRST(S)∪=FIRST(w)={w}

3)FIRST(A)∪=FIRST(x)={x,ϵ}

4)FIRST(B)∪=FIRST(y)={y,ϵ}

5)FIRST(B)∪=FIRST(A)∪{y}={x,y}

6)FIRST(C)∪=FIRST(z)={z}

Example 2
(with A→ϵ rules)

FIRST(S)={x,y,z,w, }
FIRST(A)={x, , , ,ϵ}
FIRST(B)={x,y, , ,ϵ}
FIRST(C)={ , ,z, , }

S → A B C
S → w
A → x A
A → ϵ
B → y B
B → A y
B → ϵ
C → z

7)FIRST(S)∪=FIRST(A)∪FIRST(B)∪FIRST(C)={x,y,z,w}

8)FIRST(S)∪=FIRST(w)={w}

9)FIRST(A)∪=FIRST(x)={x,ϵ}

10)FIRST(B)∪=FIRST(y)={y,ϵ}

11)FIRST(B)∪=FIRST(A)={x,y}

12)FIRST(C)∪=FIRST(z)={z}

Example 3
(with A→ϵ rules)

```
FIRST(S)={x,y,z,w,ϵ}
FIRST(A)={x, , , ,ϵ}
FIRST(B)={x,y, , ,ϵ}
FIRST(C)={ , ,z, ,ϵ}
```

```
S → A B C
S → w
A → x A
A → ϵ
B → y B
B → A y
B → ϵ
C → z
C → ϵ
```

```
FOR each production A → Y1 ... Yk except A→ϵ DO
    ... ... ...
  IF ϵ is in FIRST(Y1) ∧ ... ∧ ϵ is in FIRST(Yk) THEN
      FIRST(A) ∪= {ϵ};
```

Computing FOLLOW Sets

Problem #3: Starting the Same

```
PROCEDURE prog ();
   stat_set = {if,id,while};
   decl_set = {int,real};


   IF curr_tok ∈ stat_set THEN
       stat();
    ELSIF curr_tok ∈ decl_set THEN
       decl();
    ELSE syntax error ENDIF;
END;
```

prog → *decl* |

stat

stat →	if	... |
	id	( ) |
	while	...
decl →	int	id |
	real	id

Problem #3: Starting the Same

```
PROCEDURE prog ();
   stat_set = FIRST(stat);

  IF ∈ in FIRST(stat) THEN
     stat_set ∪= FOLLOW(stat);

  IF curr_tok ∈ stat_set THEN
      stat();
   ELSIF curr_tok ∈ decl_set THEN
      decl();
   ELSE syntax error ENDIF;
END;
```

prog → *stat* decl

stat → if ... |

while ... |

; stat |

∈

decl → int id |

real id

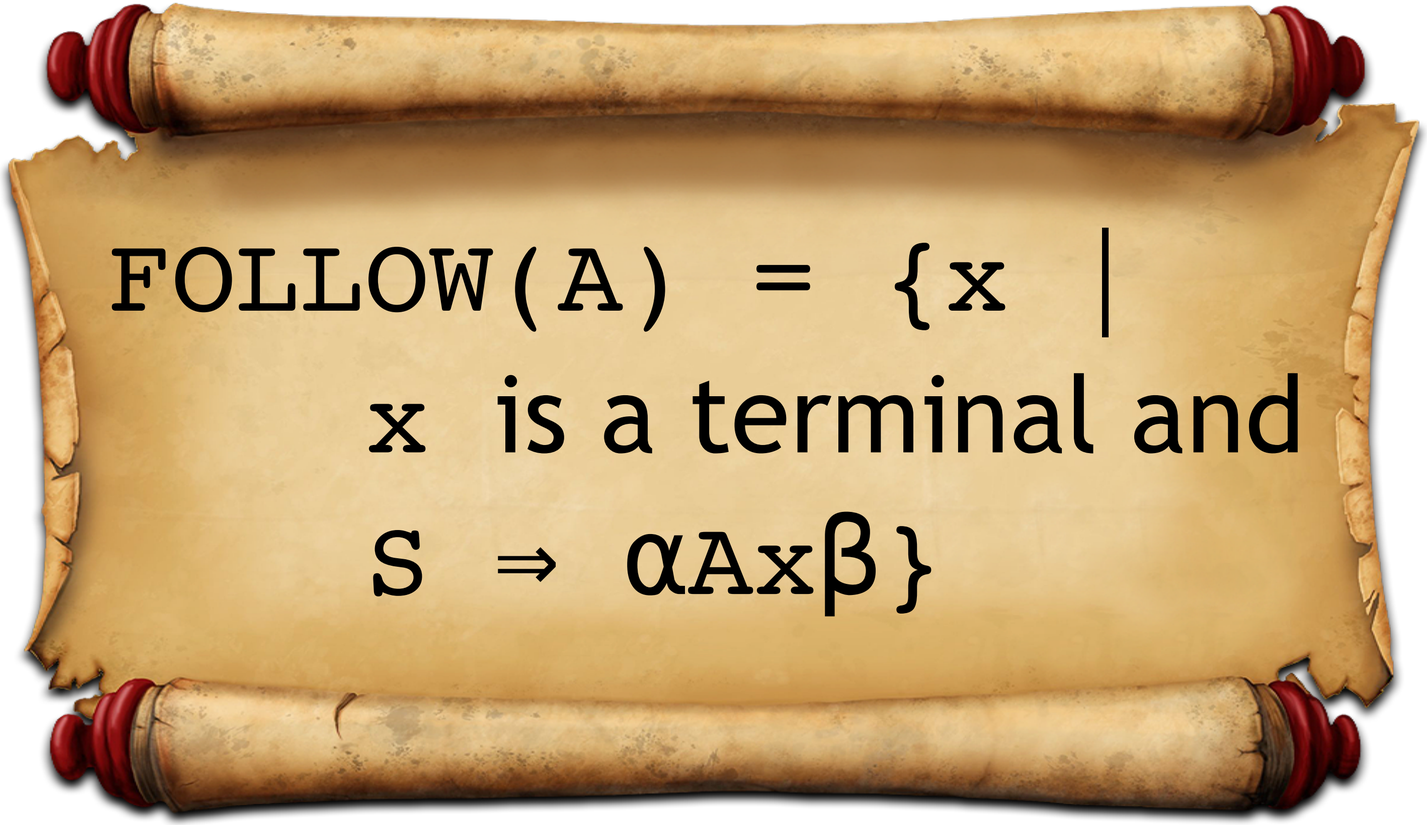
FOLLOW Sets

`FOLLOW(A)` is the set of terminals that can follow right after the nonterminal `A` in some sentential form.

$$S \Rightarrow \alpha A \beta$$

Terminals that can start β are in FOLLOW(A)

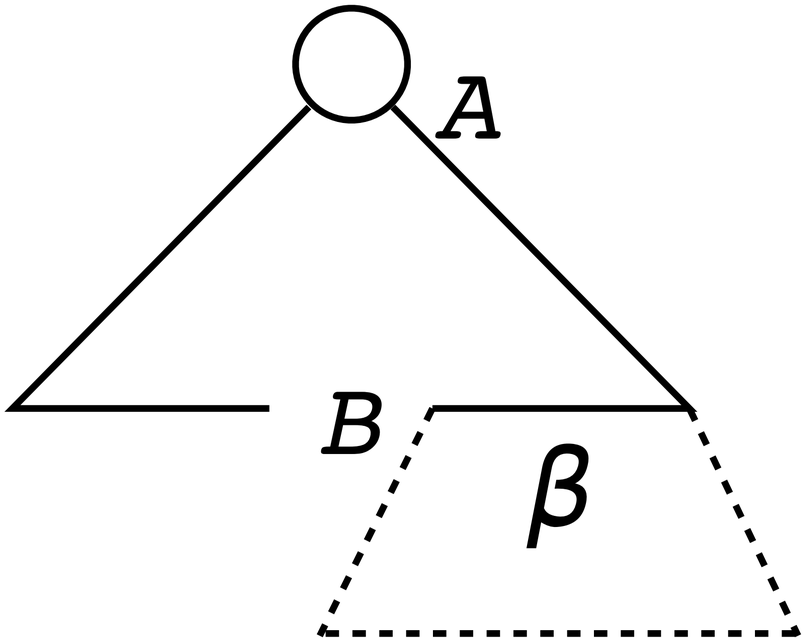
FOLLOW Sets



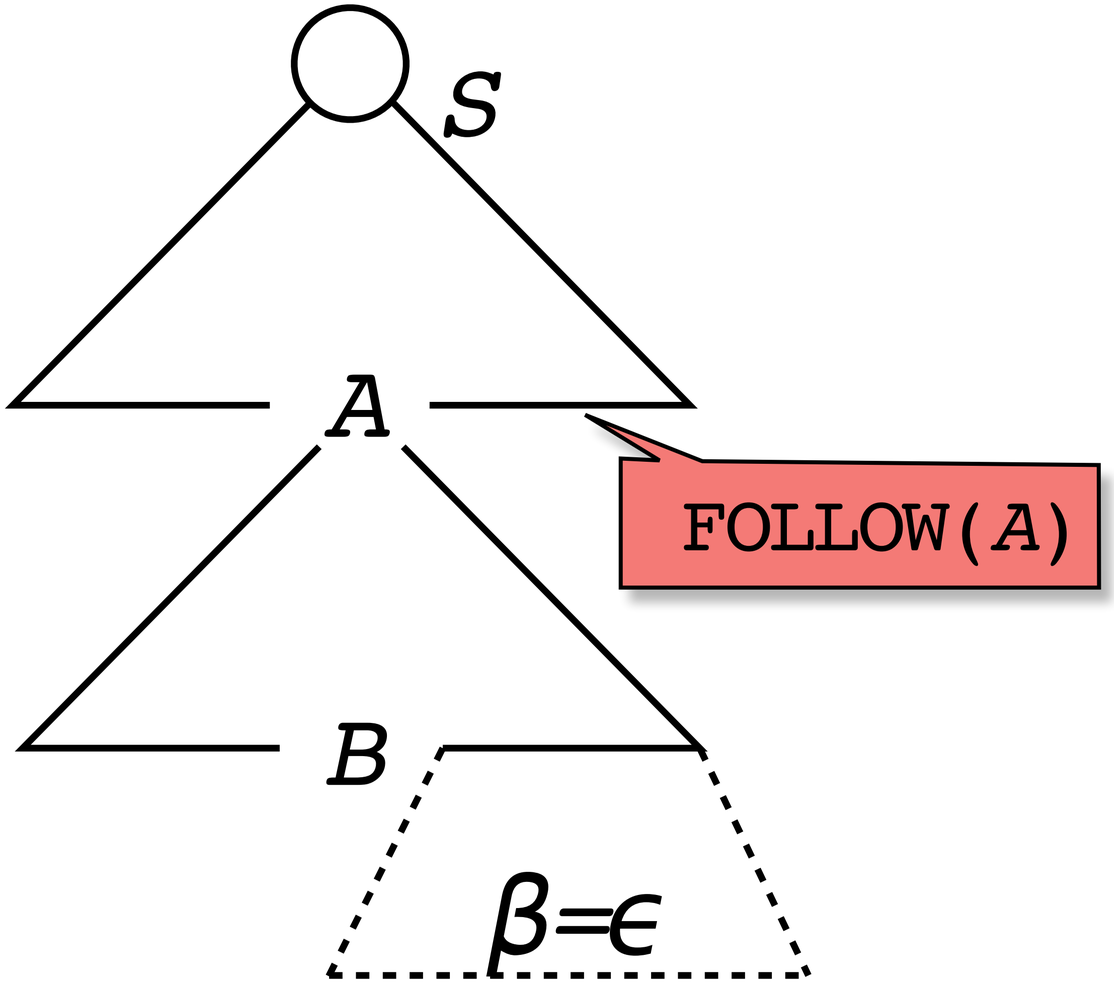
- Let $ symbolize end-of-input.
- $ ∈ FOLLOW(A) if A is the rightmost symbol in a sentential form, i.e. S ⇒ αA.

Visualizing FOLLOW Sets

S → ... A ...
A → B β



FOLLOW(*B*) ⊇ FIRST(β)



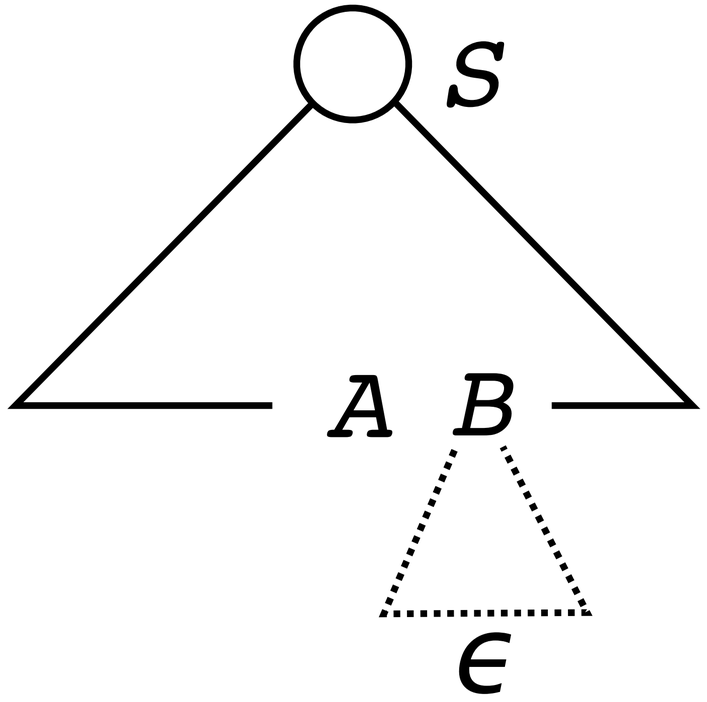
FOLLOW(*B*) ⊇ FOLLOW(*A*)

Visualizing FOLLOW Sets

S → x A B

B → β

B → ϵ



$$S \Rightarrow \underline{x}\ A\ B \Rightarrow \underline{x}\ A$$

At the end of a sentential form.

$ ∈ FOLLOW(A)

Computing FOLLOW Sets

1. **FOR** each non-terminal A **DO** FOLLOW(A) = {}
2. FOLLOW(S) = {$}

REPEAT until no more changes:

3. **FOR** each production $A \rightarrow \alpha B \beta$ **DO**

   FOLLOW(B) ∪= (FIRST(β) – {ϵ})

4. **FOR** each production $A \rightarrow \alpha B$ **DO**

   FOLLOW(B) ∪= FOLLOW(A)

5. **FOR** each production $A \rightarrow \alpha B \beta$ **WHERE** ϵ is in FIRST(β) **DO**

   FOLLOW(B) ∪= FOLLOW(A)

Example 4

3. **FOR** each $A \rightarrow \alpha B \beta$ **DO**
   FOLLOW(B) ∪= (FIRST(β) - {ϵ})
4. **FOR** each $A \rightarrow \alpha B$ **DO**
   FOLLOW(B) ∪= FOLLOW(A)
5. **FOR** each $A \rightarrow \alpha B \beta$ **IF** ϵ ∈ FIRST(β) **DO**
   FOLLOW(B) ∪= FOLLOW(A)

FIRST(S)={x}
FIRST(A)={x,y}
FIRST(B)={x,ϵ}

FOLLOW(S)={ , ,$}
FOLLOW(A)={x, ,$}
FOLLOW(B)={ , ,$}

1) FOLLOW(A) ∪= FIRST(B) - {ϵ} = {x}
2) FOLLOW(B) ∪= FOLLOW(S) = {$}
3) FOLLOW(A) ∪= FOLLOW(S) = {$}

S → x A B
A → x
A → y
B → x
B → ϵ

Example 5

3. **FOR** each $A \rightarrow \alpha B \beta$ **DO**
   FOLLOW(B) ∪= (FIRST(β) − {ϵ})
4. **FOR** each $A \rightarrow \alpha B$ **DO**
   FOLLOW(B) ∪= FOLLOW(A)
5. **FOR** each = $A \rightarrow \alpha B \beta$ **IF** ϵ ∈ FIRST(β) **DO**
   FOLLOW(B) ∪= FOLLOW(A)

FIRST(S)={x,y,z}
FIRST(A)={y,z}
FIRST(B)={w}

FOLLOW(S)={ ,y, , ,$}
FOLLOW(A)={ ,y, ,w,$}
FOLLOW(B)={ ,y, ,w,$}

1) FOLLOW(S) ∪= FIRST(y) = {y}
2) FOLLOW(A) ∪= FIRST(y) = {y}
3) FOLLOW(A) ∪= FIRST(B) = {w}
4) FOLLOW(A) ∪= FOLLOW(S) = {y,$}
5) FOLLOW(B) ∪= FOLLOW(A) = {y,w,$}

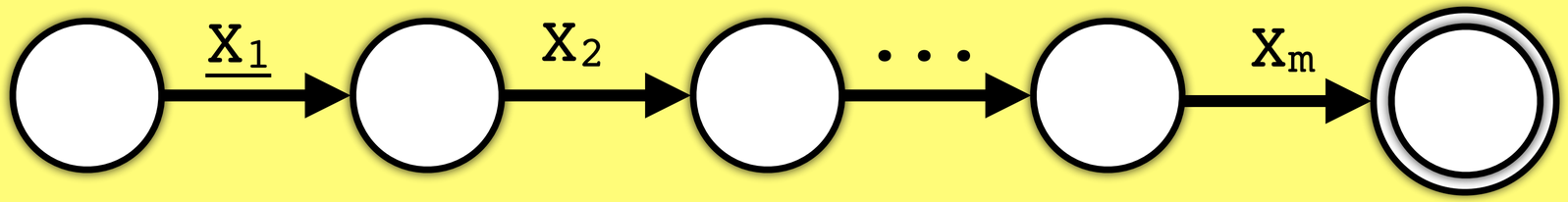
S → x S y
S → A
A → z A y
A → y
A → y A B
B → w

Construct Top-Down Parser

Construct Top-Down Parser

1. **FOR** each non-terminal A **DO**
   create initial ○ and final ◎ states.

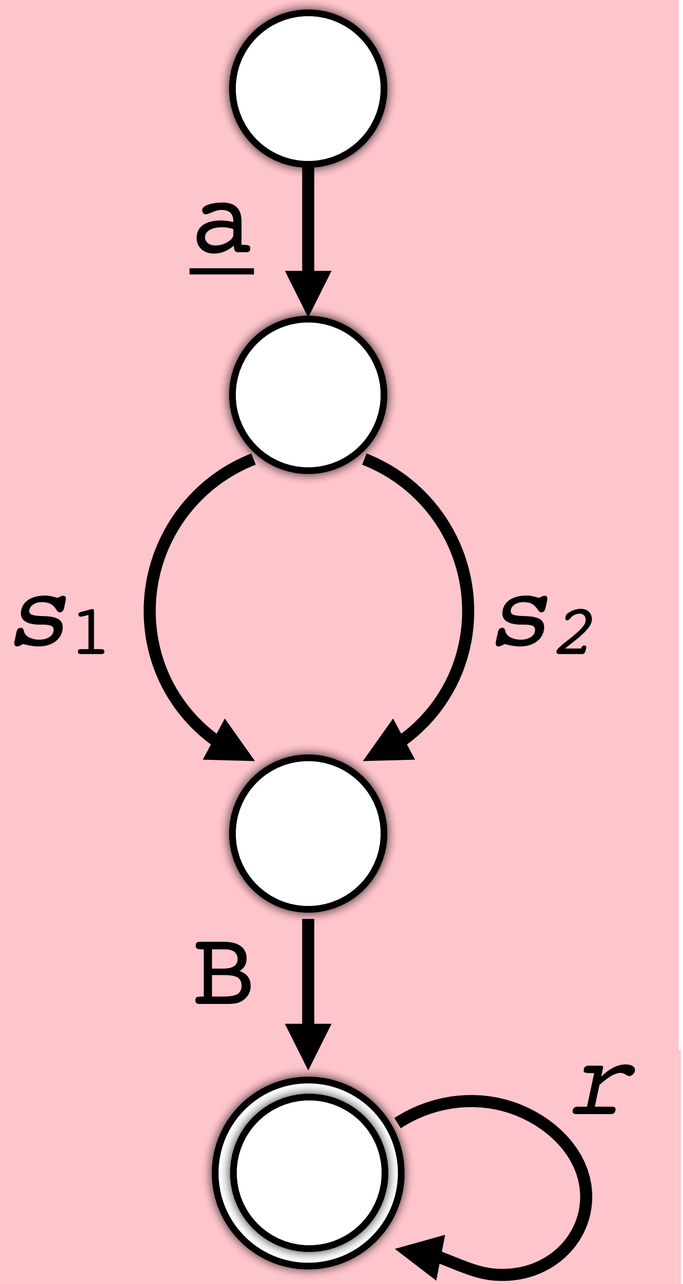
2. **FOR** each production A → $X_1, X_2, \ldots X_m$ **DO**
   create a path from A's initial to final node:



3. Simplify the transition diagrams.

4. **FOR** each transition diagram P **DO**
   Create a procedure P that ‘‘traverses’’
   the diagram guided by the input

```
PROCEDURE P();
   IF curr_tok = a THEN
      curr_tok := next_token()
   ELSE syntax error ENDIF;
   IF curr_tok ∈ FIRST(s1) THEN
      code for parsing s1
   ELSIF curr_tok ∈ FIRST(s2) THEN
      code for parsing s2
   ELSE syntax error; ENDIF
   B();
   WHILE curr_tok ∈ FIRST(r) DO
     code for parsing r
   ENDDO
END P;
```



Expression Grammar

```
FOR each production A → Y1 ... Yk except A→ϵ DO
    FIRST(A) ∪= FIRST(Y1) - {ϵ};
    FOR i = 1 to k-1 DO
        IF ϵ ∈ FIRST(Y1) ∧ ... ∧ ϵ ∈ FIRST(Yi) THEN
            FIRST(A) ∪= FIRST(Yi+1) - {ϵ};
    IF ϵ ∈ FIRST(Y1) ∧ ... ∧ ϵ ∈ FIRST(Yk) THEN
        FIRST(A) ∪= {ϵ};
```

```
FIRST(E)  = { , ,(, ,id, }
FIRST(E') = {+, , , ,  ,ϵ}
FIRST(T)  = { , ,(, ,id, }
FIRST(T') = { ,*, , ,  ,ϵ}
FIRST(F)  = { , ,(, ,id, }
```

```
E  → T E′
E′→ + T E′
E′→ ϵ
T  → F T′
T′→ * F T′
T′→ ϵ
F  → ( E )
F  → id
```

3.**FOR** each $A \rightarrow \alpha B \beta$ **DO**

FOLLOW(B) ∪= (FIRST(β) - {ϵ})

4.**FOR** each $A \rightarrow \alpha B$ **DO**

FOLLOW(B) ∪= FOLLOW(A)

5.**FOR** each $A \rightarrow \alpha B \beta$ **WHERE** ϵ ∈ FIRST(β) **DO**

FOLLOW(B) ∪= FOLLOW(A)

```
FIRST(E) ={(, id}
FIRST(E')={+, ϵ}
FIRST(T) ={(, id}
FIRST(T')={*, ϵ}
FIRST(F) ={(, id}
```

```
FOLLOW(E)  = { , , ,), ,$}
FOLLOW(E') = { , , ,), ,$}
FOLLOW(T)  = {+, , ,), ,$}
FOLLOW(T') = {+, , ,), ,$}
FOLLOW(F)  = {+,*, ,), ,$}
```

E → T E′

E′→ + T E′

E′→ ϵ

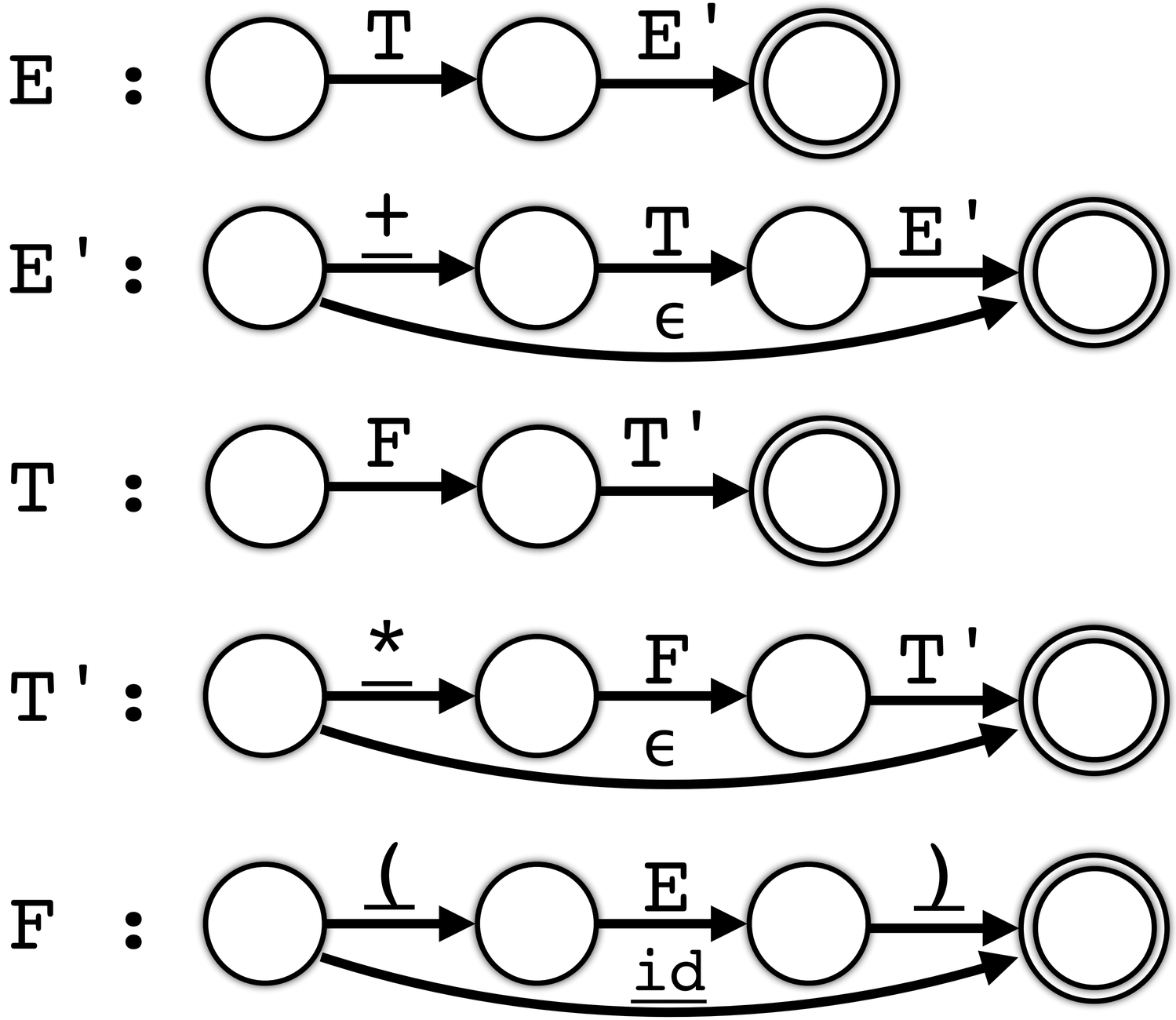
T → F T′

T′→ * F T′

T′→ ϵ

F → (E)

F → id



E → T E′

E′→ + T E′

E′→ ϵ

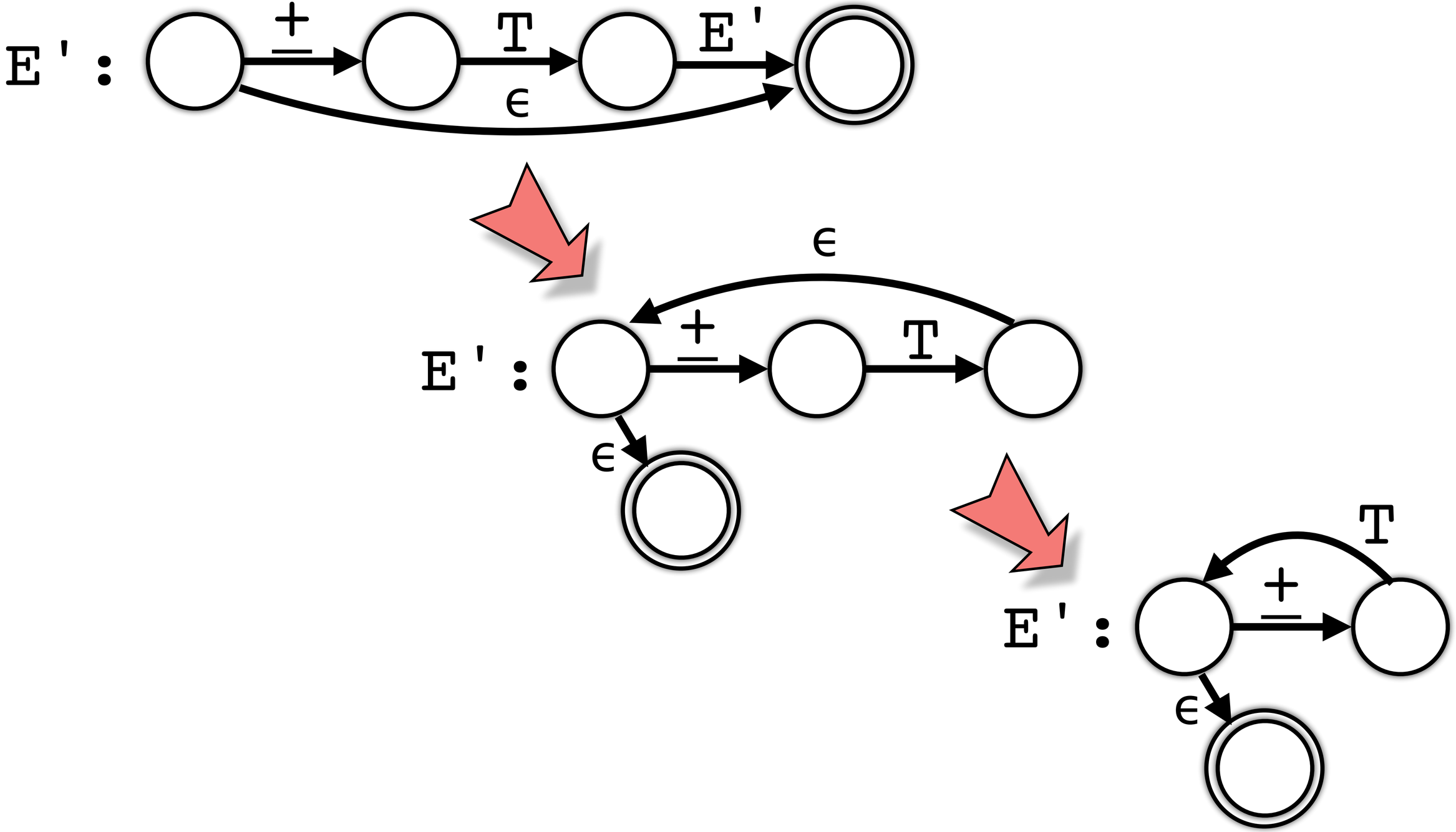
T → F T′

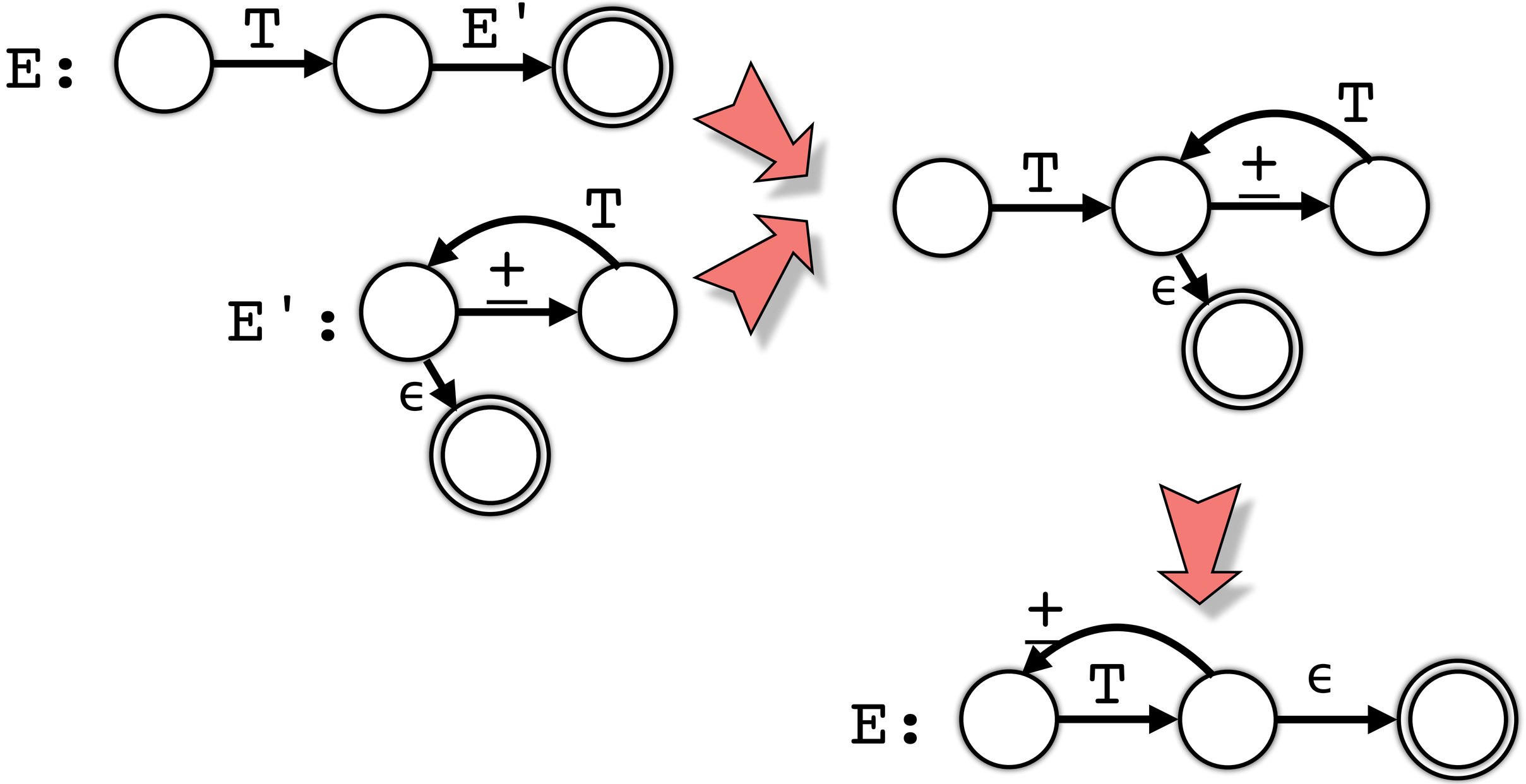
T′→ * F T′

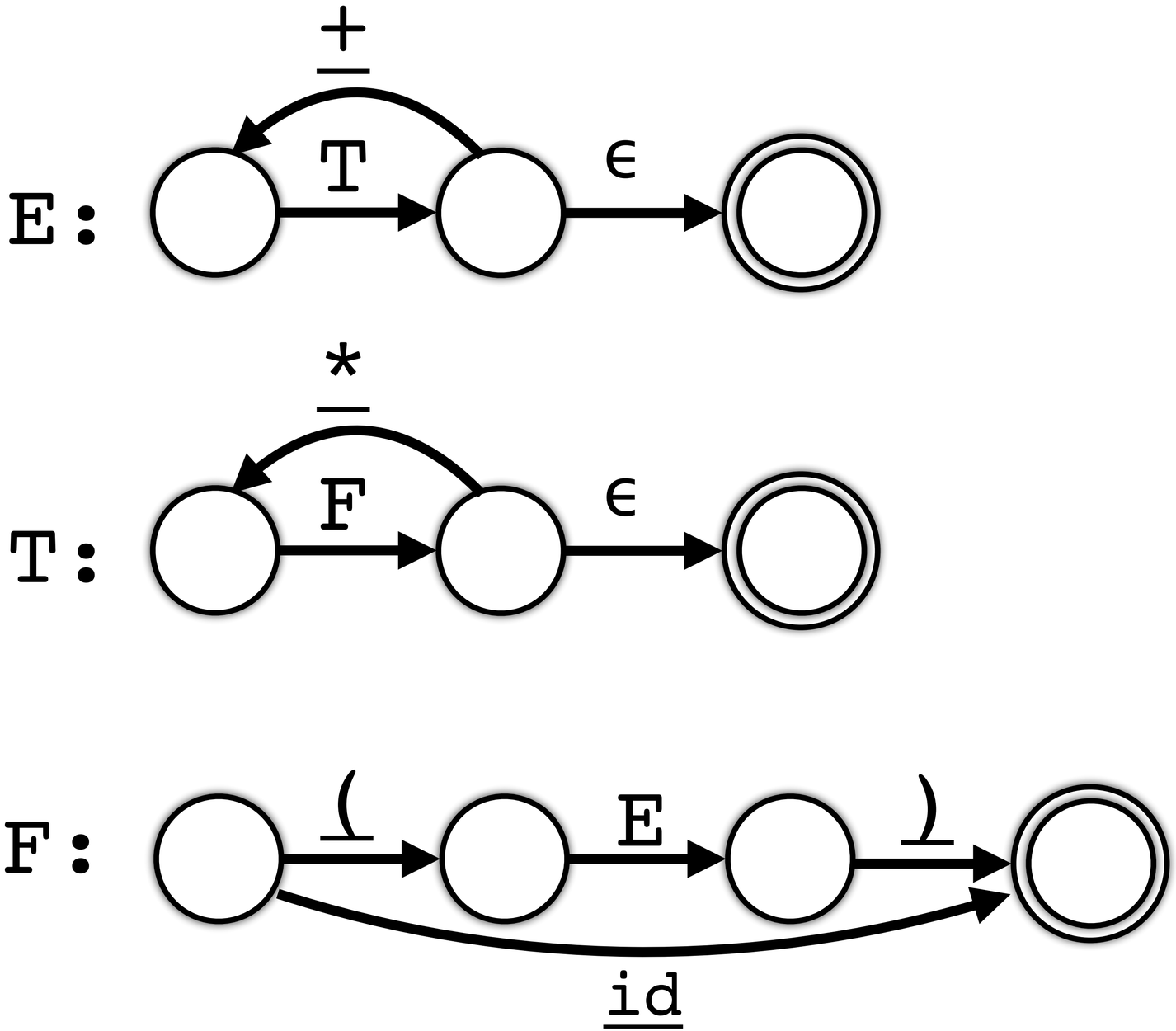
T′→ ϵ

F → (E)

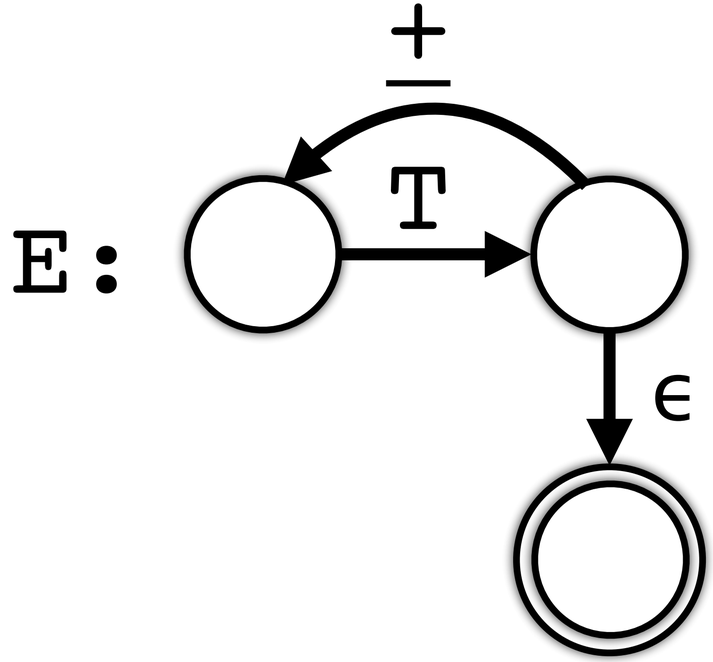
F → id


E':
+
T
E'
ϵ
E':
ϵ
+
T
ϵ
E':
T
+
ϵ

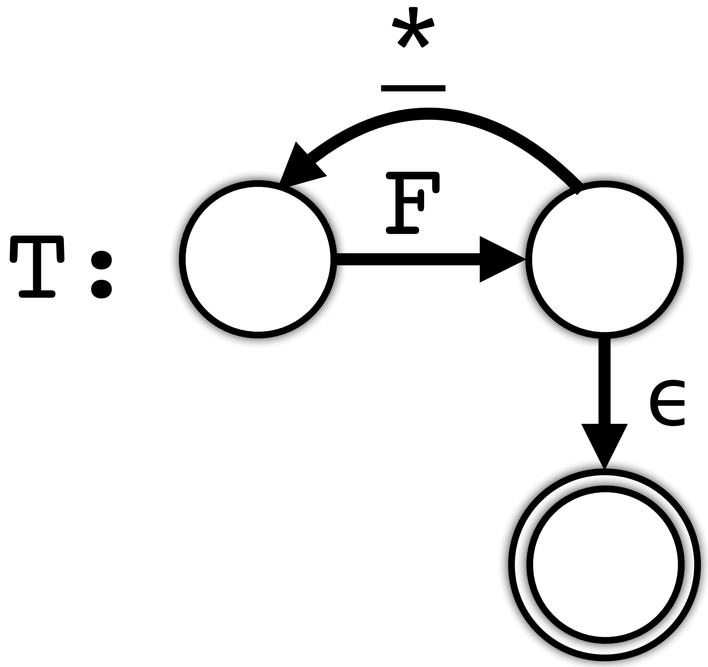

E:
T
E'
E':
T
+
ϵ
T
T
+
ϵ
+
T
ϵ
E:


E:
+
T
ϵ
T:
*
F
ϵ
F:
(
E
)
id

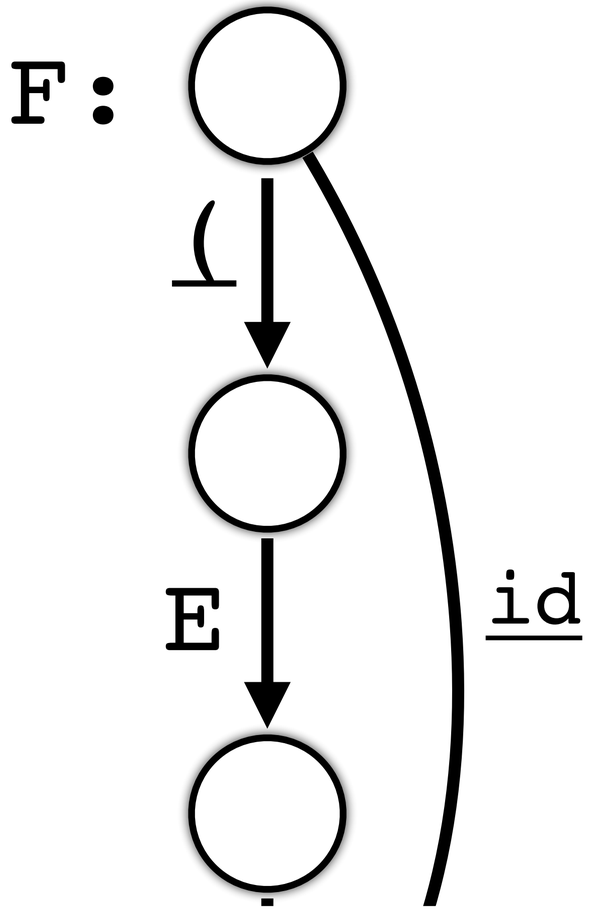
```
PROCEDURE E();
    WHILE TRUE DO
        T();
        IF curr_tok = + THEN
            curr_tok := next_token()
        ELSE
           EXIT
        ENDIF
    ENDDO;
END
```

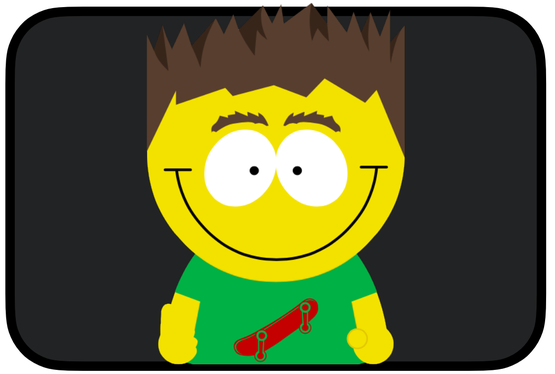


```
PROCEDURE T();
    WHILE TRUE DO
        F();
        IF curr_tok = * THEN
            curr_tok := next_token()
        ELSE
           EXIT
        ENDIF
    ENDDO;
END
```



```
PROCEDURE F();
   IF curr_tok = ( THEN
      curr_tok := next_token();
      E();
      IF curr_tok = ) THEN
         curr_tok := next_token();
      ELSE syntax error ENDIF
    ELSIF curr_tok = id THEN
       curr_tok := next_token()
    ELSE syntax error ENDIF
END
```





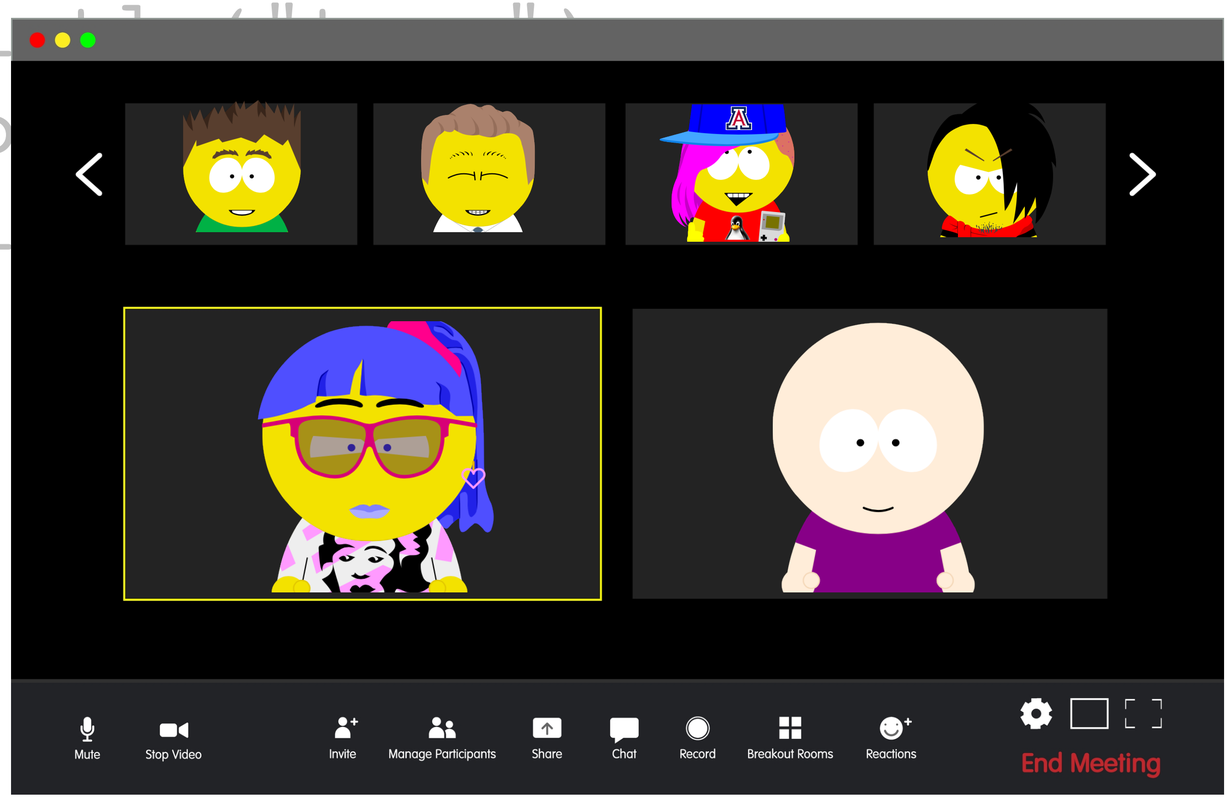
```
class ExprH {
    static String[] input = {"i","+","i","*","i","$"};
    static int current = 0;

    static boolean lookahead(String S) {
        return input[current].equals(S);
    }

    static void match(String S) throws Exception {
        if (input[current].equals(S))
            current++;
        else
            throw new Exception();
    }
```

```
static void E() throws Exception {
    T();
    while(lookahead("+")) {
       match("+");
       T();
    }
}
static void T() throws Exception {
    F();
    while(lookahead("*")) {
      match("*");
      F();
    }
}
```

```
static void F() throws Exception {
    if (lookahead("(")) {
       match("(");
       E();
       match(")");
    } else {
      match("i");
    }
}
```

```
    public static void main(String[] args) {
        try {
            E();
            System.out.println("correct!");
        } catch (Exception e) {
            System.out.println("syntax error!");
        }
    }
}
```

```
public static void main(String[] args) {
    try {
        E();
        System.out.pri
    } catch (Exceptio
        System.out.pri
    }
  }
}
```



If you liked this lecture, leave a tip!

15%

20%

25%

Custom Tip Amount

No Tip