- Content-Free Grammar (CFG) is defined by the following four components :

(1)A finite terminal vocabulary Vt ; this is the token set produced by the scanner.

(2)A finite set of different, interminate symbols, called the nonterminal vocabulary Vn.

(3)A start symbol $S \in Vn$ that starts all derivations. A start symbol is sometimes called a goal symbol.

(4)P, a finite set of productions (sometimes called rewriting rules) of the form $A \rightarrow X1 \ldots Xm$, where

$A \in V_n$, $X_i \in V_n \cup V_t$, $1 \le i \le m$, $m \ge 0$

Note that $A \to \lambda$ is a valid production.

a, b, c, … denote symbols in Vt

A, B, C, … denote symbols in Vn

U, V, W, … denote symbols in V

α,β,γ, … denote symbols in V*

u, v, w, … denote symbols in Vt*

- $A \rightarrow \alpha \mid \beta \mid \ldots \mid \zeta$
- This is an abbreviation for the sequence of productions :

$$A \rightarrow \alpha$$
$$A \rightarrow \beta$$
$$\ldots$$
$$A \rightarrow \zeta$$

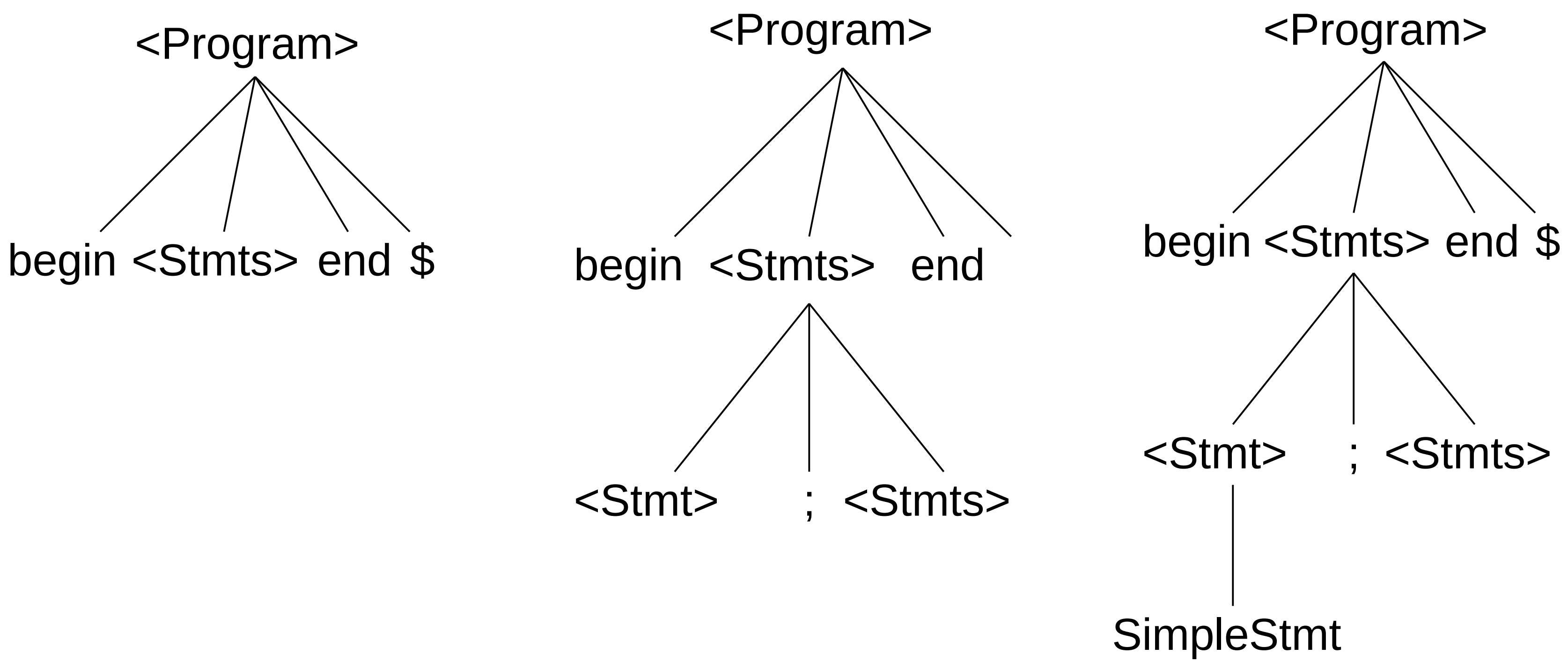
- If $A \rightarrow \gamma$ is a production, then $\alpha A \beta \Rightarrow \alpha\gamma\beta$, where $\Rightarrow$ denotes a one-step derivation (using production $A \rightarrow \gamma$). We extend $\Rightarrow$ to $\Rightarrow_+$, derived in one or more steps, and $\Rightarrow^*$, derived in zero or more steps. If $S \Rightarrow^* \beta$, then β is said to be a sentential form of the CFG. SF(G) is the set of sentential forms of grammar G. Similarly, $L(G) = \{ x \in V_t^* \mid S \Rightarrow^+ x \}$. Note that $L(G) = SF(G) \cap V_t^*$; that is, the language of G is simply those sentential forms of G that are terminal strings.

E	$\rightarrow$	Prefix (E)
E	$\rightarrow$	V Tail
Prefix	$\rightarrow$	F
Prefix	$\rightarrow$	λ
Tail	$\rightarrow$	+ E
Tail	$\rightarrow$	λ

A leftmost derivation of F(V+V) is :

E	$\Rightarrow_{lm}$	Prefix (E)
	$\Rightarrow_{lm}$	F(E)
	$\Rightarrow_{lm}$	F(V Tail)
	$\Rightarrow_{lm}$	F(V+E)
	$\Rightarrow_{lm}$	F(V+V Tail)
	$\Rightarrow_{lm}$	F(V+V)

<Program>


<Program>
begin <Stmts> end $
<Program>
begin <Stmts> end
<Stmt> ; <Stmts>
<Program>
begin <Stmts> end $
<Stmt> ; <Stmts>
SimpleStmt

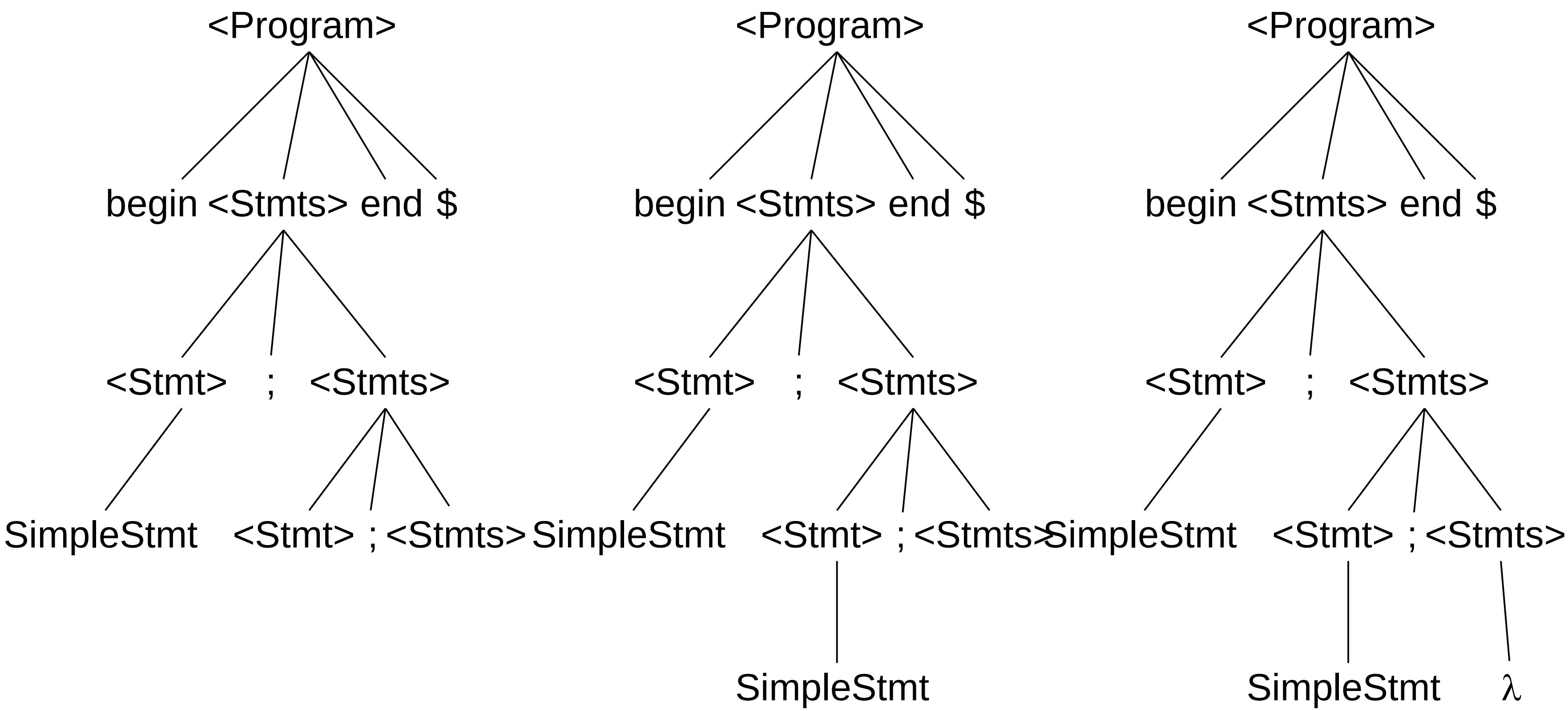
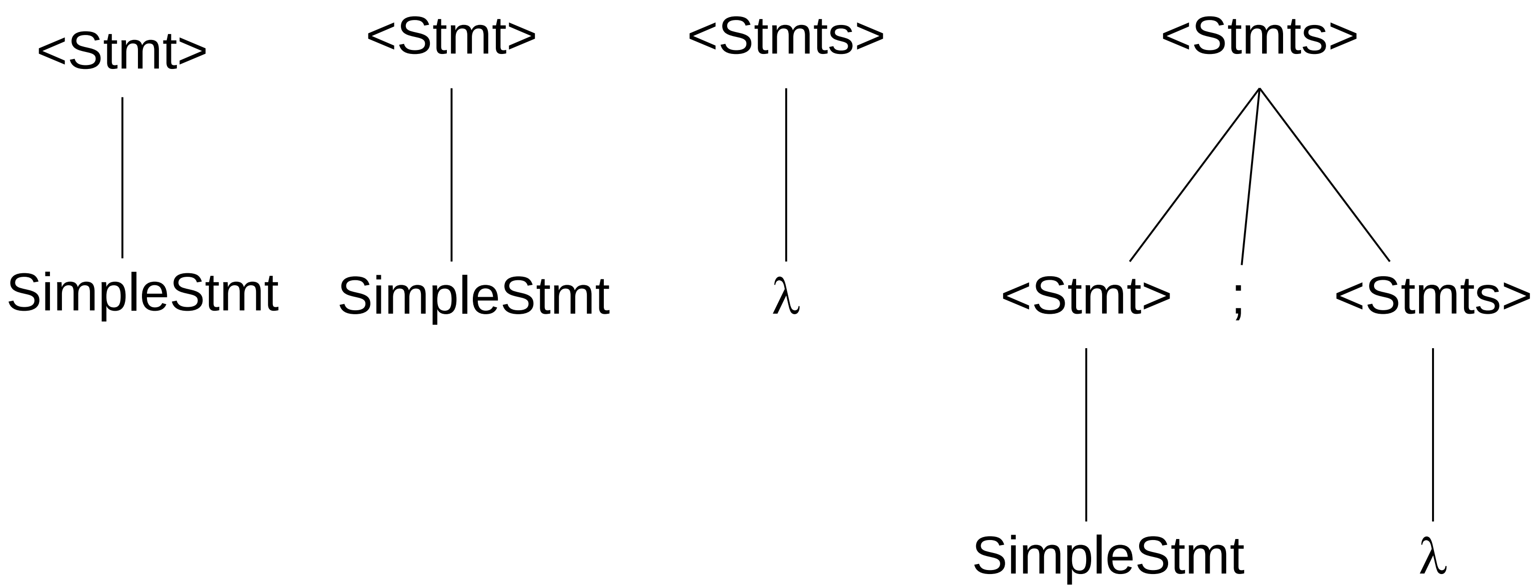


Figure 4.5 A Top-Down Parse


<Stmt>
SimpleStmt
<Stmt>
SimpleStmt
<Stmts>
λ
<Stmts>
<Stmt>
;
<Stmts>
SimpleStmt
λ

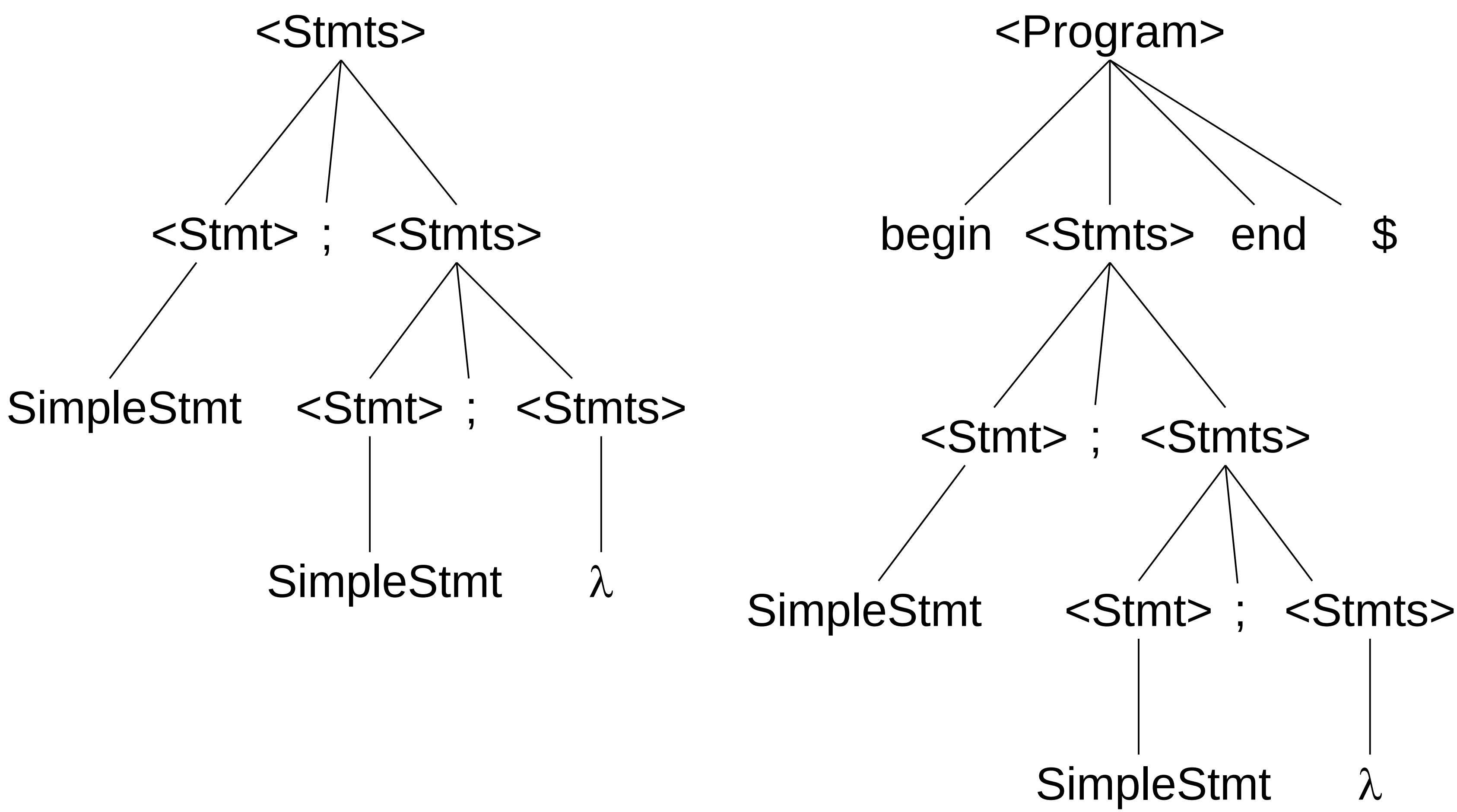


Figure 4.6 A Bottom-Up Parse

```
typedef set_of_terminal_or_lambda termset;
termset follow_set [NUM_NONTERMINAL];
termset first_set [SYMBOL];
marked_vocabulary derives_lambda = mark_lambda (g);
/* mark_lambda (g), p. 103*/
```

```
termset compute_first (string_of_symbols alpha)
{
 int I, k;
 termset result;
 k = length(alpha);
 if (k == 0)
   result = SET_OF(λ);
 else {
   result = first_set [ alpha[0] ];
   for ( i = 1 ; i < k && λ ∈ first_set[ alpha[i-1] ] ; i++)
     result = result ∪ (first_set [ alpha[i] ] – SET_OF(λ));
   if ( i == k && λ ∈ first_set [ alpha[k-1] ] )
     result = result ∪ SET_OF(λ);
     }
 return result;
}
```

Figure 4.8 Algorithm to Compute First(alpha)

```
extern grammar g;
void fill_first_set (void)
{
    nonterminal  A;
    terminal     a;
    production   p;
    boolean      changes;
    int          i, j;
    for ( i = 0 ; i < NUM_NONTERMINAL ; i++)
    {
        A = g.nonterminals[i];
        if (derives_lambda[A])
            first_set[A] = SET_OF(λ);
        else
            first_set[A] = ∅;
    }
```

```
for ( i = 0 ; i < NUM_TERMINAL ; i++)
{
    a = g.terminals[i];
    first_set[a] = SET_OF(a);
    for ( j = 0 ; j < NUM_NONTERMINAL ; j++)
    {
        A = g.nonterminals[j];
        if ( there exists a production A → aβ)
            first_set[A] = first_set[A] ∪ SET_OF(a);
    }
}
```

```
    do  {
          changes = FALSE;
          for ( i = 0 ; i < g.num_productions ; i++ )
          {
                   p = g.productions[i];
                  first_set[p.lhs] = first_set[p.lhs] ∪
                   compute_first(p.rhs);
                  if ( first_set changed )
                    changes = TRUE;
        }
    } while (changes);
}
```

Figure 4.9 Algorithm to Compute First Sets for V

- Follow (A) = { a $\in$ Vt | S $\Rightarrow$+ … Aa … } $\cup$

  (if S $\Rightarrow$+ αA then {λ} else $\varnothing$)

- Follow (A) is the set of terminals that may follow A in some sentential form.

- 用處：they define the right contest consistent with a

  given nonterminal. (lookahead)

$E \rightarrow Prefix\ (\ E\ )$

$E \rightarrow V\ Tail$

$Prefix \rightarrow F$

$Prefix \rightarrow \lambda$

$Tail \rightarrow +\ E$

$Tail \rightarrow \lambda$

Step	first_set							
	E	Prefix	Tail	(	)	V	F	+
(1)First loop	$\varnothing$	{ λ }	{ λ }					
(2)Second (nested) loop	{ V }	{ F, λ }	{ +, λ }	{ (}	{) }	{ V }	{ F }	{ + }
(3)Third loop, production 1	{ V,F,(}	{ F, λ }	{ +, λ }	{ (}	{) }	{ V }	{ F }	{ + }

For top-down

First(α) = { a $\in$ Vt | $\alpha \Rightarrow^* a\beta$ } $\cup$ (if $\alpha \Rightarrow^* \lambda$ then {λ} else $\emptyset$)

```
void fill_follow_set (void)
{
    nonterminal A, B;
    int i;
    boolean    changes;
    for ( i = 0 ; i < NUM_NONTERMINAL ; i++ )  {
        A = g.nonterminals[i];
        Follow_set[A] = ∅;
    }
    follow_set[g.start_symbol] = SET_OF( λ ) ;
```

```
do  {
   changes = FLASE;
   for (each production A → αBβ)  {
     / *
      * i.e. for each production and each occurrence
      * of a nonterminal in its right-hand side.
     */
     follow_set[B] = follow_set[B] ∪
                          (compute_first(β) – SET_OF(λ) );
     if ( λ ∈ compute_first(β) )
          follow_set[B] = follow_set[B] ∪ follow_set[A];
     if ( follow_set[B] changed )
          changes = TRUE;
   }
}  while (changes)
}
```

Figure 4.10 Algorithm to Compute Follow Sets for All Nonterminals

$$
\begin{aligned}
E &\rightarrow \text{Prefix} (E) \\
E &\rightarrow V\ \text{Tail} \\
\text{Prefix} &\rightarrow F \\
\text{Prefix} &\rightarrow \lambda \\
\text{Tail} &\rightarrow + E \\
\text{Tail} &\rightarrow \lambda
\end{aligned}
$$

Step	Follow_set		
	E	Prefix	Tail
(1)Initialization	{ λ }	$\varnothing$	$\varnothing$
(2)Process Prefix in production 1	{ λ }	{ (}	$\varnothing$
(3)Process E in production 1	{ λ,) }	{ (}	$\varnothing$
(4)Process Tail in production 2	{ λ,) }	{ (}	{ λ,) }

- To further illustrate the operation of the programs that compute First and Follow sets, we present two additional examples. For the following grammar

$$S \to aSe$$
$$S \to B$$
$$B \to bBe$$
$$B \to C$$
$$C \to cCe$$
$$C \to d$$

The execution of fill_first_set would proceed as follows :

Step	first_set							
	S	B	C	a	b	c	d	e
(1)First loop	∅	∅	∅					
(2)Second (nested) loop	{ a }	{ b }	{ c,d }	{ a }	{ b }	{ c }	{ d }	{ e }
(3)Third loop, production 2	{ a,b }	{ b }	{ c,d }	{ a }	{ b }	{ c }	{ d }	{ e }
(4)Third loop, production 4	{ a,b }	{b,c,d }	{ c,d }	{ a }	{ b }	{ c }	{ d }	{ e }
(5)Third loop, production 2	{ a,b,c,d }	{b,c,d }	{ c,d }	{ a }	{ b }	{ c }	{ d }	{ e }

- The execution of fill_follow_set is illustrated by

Step	follow_set		
	S	B	C
(1)Initialization	{ λ }	$\varnothing$	$\varnothing$
(2)Process S in production 1	{ e,λ }	$\varnothing$	$\varnothing$
(3)Process B in production 2	{ e,λ }	{ e,λ }	$\varnothing$
(4)Process B in production 3	No changes		
(5)Process C in production 4	{ e,λ }	{ e,λ }	{ e,λ }
(6)Process C in production 5	No changes		

- For the second example grammar

$$
\begin{array}{lll}
S & \rightarrow & ABc \\
A & \rightarrow & a \\
A & \rightarrow & \lambda \\
B & \rightarrow & b \\
B & \rightarrow & \lambda
\end{array}
$$

- The execution of fill_first_set would proceed as follows :

Step	first_set					
	S	A	B	a	b	c
(1)First loop	$\varnothing$	{ λ }	{ λ }			
(2)Second (nested) loop	$\varnothing$	{ a,λ }	{ b,λ }	{ a }	{ b }	{ c }
(3)Third loop, production 1	{ a,b,c }	{ a,λ }	{ b,λ }	{ a }	{ b }	{ c }

- The execution of fill_follow_set is illustrated by

Step	follow_set		
	S	A	B
(1)Initialization	$\{ \lambda \}$	$\varnothing$	$\varnothing$
(2)Process A in production 1	$\{ \lambda \}$	{ b,c }	$\varnothing$
(3)Process B in production 1	$\{ \lambda \}$	{ b,c }	{ c }