第三章：语法分析

递归下降法

1.1 递归下降法的基本原理

- 递归下降法(Recursive-Descent Parsing)

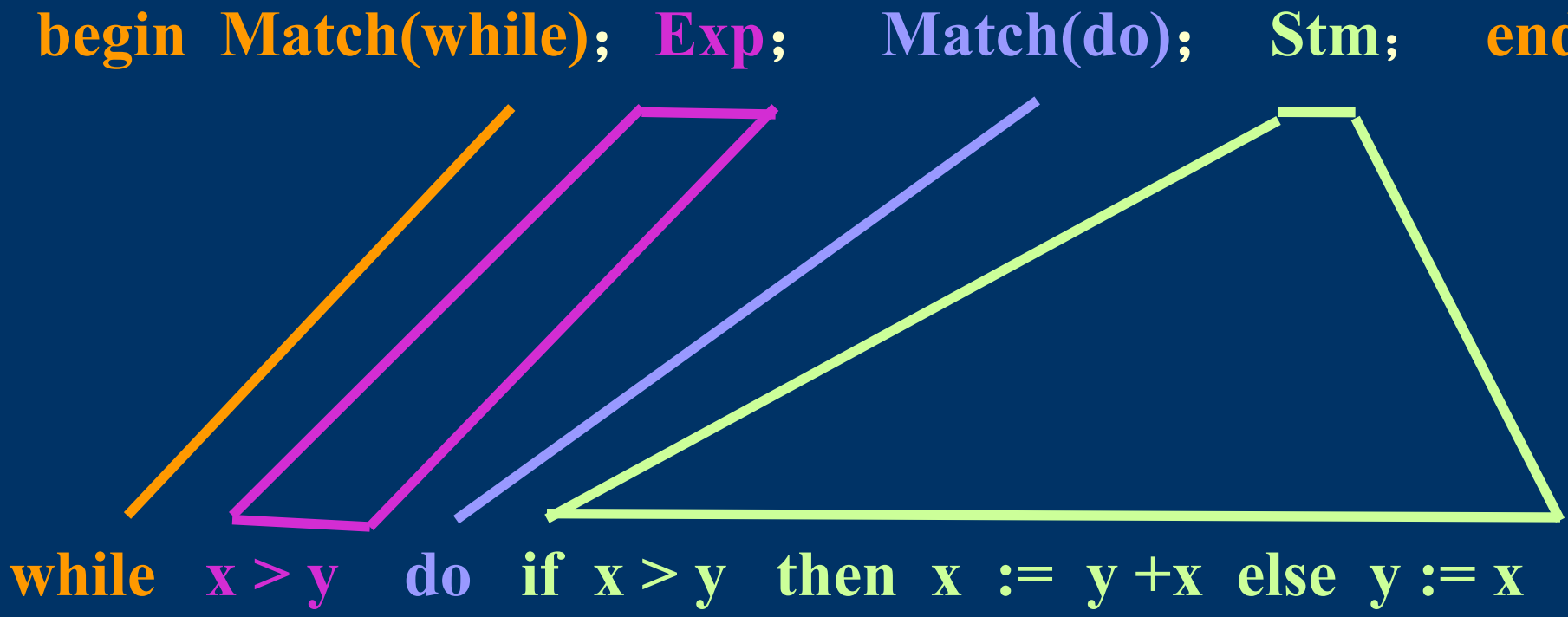
  对每个非终极符构造相应的一个子程序(称为语法分析子程序)，其功能是识别、分析该非终极符所能推导出的字符串.

例如：一条产生式：

While_Stm→while Exp do Stm

则对应产生式右部的语法分析程序部分如下：

```
begin
        Match(while);
        Exp;
        Match(do);
        Stm;
  end
```


begin Match(while); Exp; Match(do); Stm; end
while x > y do if x > y then x := y +x else y := x

递归下降分析示图

1.2 对文法的要求

- 为了保证推导的唯一性，对文法的要求与LL(1)文法相同。即对于文法G中任一非终极符A，其任意两个产生式$A\rightarrow\alpha$和$A\rightarrow\beta$，都要满足下面条件：

  $Predict(A\rightarrow\alpha)\cap Predict(A\rightarrow\beta)=\varnothing$

2. 语法分析程序的构造

- 两个标准函数

1. ReadToken：把输入流的头符读入变量token中
2. Match(a)： if token=a then ReadToken
   else 出错

2. 语法分析程序的构造

当产生式形如:$A\rightarrow\beta_1|\beta_2|\cdots|\beta_n$,则按下面的方法编写子程序A:

```
procedure A( )
begin if token∈Predict(A→β1) then θ(β1) else
      if token∈Predict(A→β2) then θ(β2) else
      ......
      if token∈Predict(A→βn) then θ(βn) else
      error( )
end
```

其中对$\beta_i=X_1X_2\cdots X_n$, $\theta(\beta_i)=\theta'(X_1);\theta'(X_2);\cdots;\theta'(X_n)$;

如果$X\in V_N$, $\theta'(X)=$ X();

如果$X\in V_T$, $\theta'(X)=$ Match(X); //即if(token==X)ReadToken();

如果X= ε, $\theta'(\varepsilon)$ = skip(空语句).

2. 语法分析程序的构造

- 主程序：

```
void main(){
    ReadToken(); S( );
  if (token=='#')  成功;
  else 失败
}
```

2. 语法分析程序的构造

- 具体构建流程

给定一个文法G

1. 求每条规则的Predict集
2. 写针对每个非终极符的函数
3. 写主函数

优点：
构造简单

缺点：
1. 频繁的函数调用影响效率
2. 程序比较长

终极符产生匹配命令，而非终极符则产生调用命令. 因为文法递归相应子程序也递归，所以称这种方法为递归子程序方法或递归下降法.

例：假设有文法

$Z \rightarrow a\ B\ a$

$B \rightarrow b\ B \mid c$

θ(aBa)

θ'(a); θ'(B); θ'(a)

Predict(Z→aBa)={a},

Predict(B→bB)={b}, Predict{B→c}={c}

则相应的递归子程序可如下：

```
procedure Z( )
begin
  if token=a then Match(a);
                  B;
  else  err( 1 )
end;
```

ReadToken

```
procedure B ( )
begin
  if token = b then Match(b);
                    B;
  else  if token = c
        then Match(c);
        else  err( 2 )
end;
```

ReadToken

语法分析主程序：Begin ReadToken; Z ； Match(#) End

例：

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E' \mid \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \varepsilon$

$F \rightarrow i \mid (\ E\ )$

Predict(E→TE') = first(TE') = { i , (}

Predict(E' →+TE') = first(+TE') = { + }

Predict(E' → ε) = follow(E') = {) , # }

Predict(T →FT') = first(FT') = { i , (}

Predict(T' →*FT') = first(*FT') = { * }

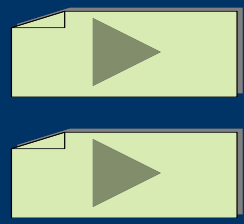
Predict(T' → ε) = follow(T') = { + ,) , # }

Predict(F →id) = first(id) = { i }

Predict(F →(E)) = first((E)) = { (}

语法分析主程序：Begin ReadToken; E ； Match(#) end

```
procedure E( )
begin
  if token∈{ i , ( } then T;
                          E' ;
                    else err( 1 )
end;
```



例：

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E' \mid \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \varepsilon$

$F \rightarrow i \mid (\ E\ )$

Predict($E \rightarrow TE'$) = first(TE') = { i , (}

Predict($E' \rightarrow +TE'$) = first($+TE'$) = { + }

Predict($E' \rightarrow \varepsilon$) = follow($E'$) = {) , # }

Predict($T \rightarrow FT'$) = first(FT') = { i , (}

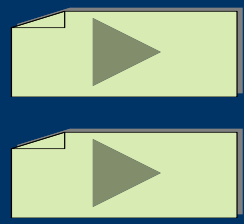
Predict($T' \rightarrow *FT'$) = first($*FT'$) = { * }

Predict($T' \rightarrow \varepsilon$) = follow(T') = { + ,) , # }

Predict($F \rightarrow id$) = first(id) = { i }

Predict($F \rightarrow (E)$) = first((E)) = { (}

```
procedure E' ( )
begin
  if token="+" then     Match(+);
                        T;
                        E'
               else  if  token∈ { ) , # }  then  skip
                                          else  err( 2 )
end;
```



例：

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E' \mid \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \varepsilon$

$F \rightarrow i \mid (\ E\ )$

Predict(E→TE') = first(TE') = { i , (}

Predict(E' →+TE') = first(+TE') = { + }

Predict(E' → ε) = follow(E') = {) , # }

Predict(T →FT') = first(FT') = { i , (}

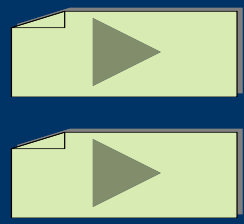
Predict(T' →*FT') = first(*FT') = { * }

Predict(T' → ε) = follow(T') = { + ,) , # }

Predict(F →id) = first(id) = { i }

Predict(F →(E)) = first((E)) = { (}

```
procedure T( )
begin
  if  token∈ { i , ( }  then F;
                              T ’
                        else  err( 3 )
end;
```



例：

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E' \mid \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \varepsilon$

$F \rightarrow i \mid (\ E\ )$

Predict(E→TE') = first(TE') = { i , (}

Predict(E' →+TE') = first(+TE') = { + }

Predict(E' → ε) = follow(E') = {) , # }

Predict(T →FT') = first(FT') = { i , (}

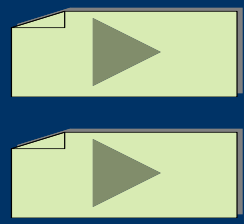
Predict(T' →*FT') = first(*FT') = { * }

Predict(T' → ε) = follow(T') = { + ,) , # }

Predict(F →id) = first(id) = { i }

Predict(F →(E)) = first((E)) = { (}

```
procedure T' ( )
begin
  if token=" * " then    Match( * );
                         F;
                         T'
               else  if  token∈ { +,) , # }  then skip
                                             else err( 4 )
end;
```



例：

$E \rightarrow T\ E'$

$E' \rightarrow +\ T\ E' \mid \varepsilon$

$T \rightarrow F\ T'$

$T' \rightarrow *\ F\ T' \mid \varepsilon$

$F \rightarrow i \mid (\ E\ )$

Predict(E→TE’) = first(TE’) = { i , (}

Predict(E’ →+TE’) = first(+TE’) = { + }

Predict(E’ → ε) = follow(E’) = {) , # }

Predict(T →FT’) = first(FT’) = { i , (}

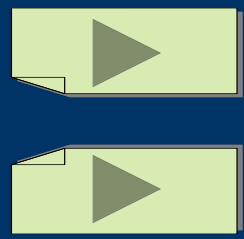
Predict(T’ →*FT’) = first(*FT’) = { * }

Predict(T’ → ε) = follow(T’) = { + ,) , # }

Predict(F →id) = first(id) = { i }

Predict(F →(E)) = first((E)) = { (}

```
procedure F ( )
begin
  if  token=“ i”  then      Match( i )
                  else   if   token=“(”   then   Match( ( );
                                                 E;
                                                 Match( ) )
                                               else  err( 5 )
end;
```



例：i+i*i # 递归下降分析过程

语法分析主程序：Begin ReadToken; E ； Match(#) end

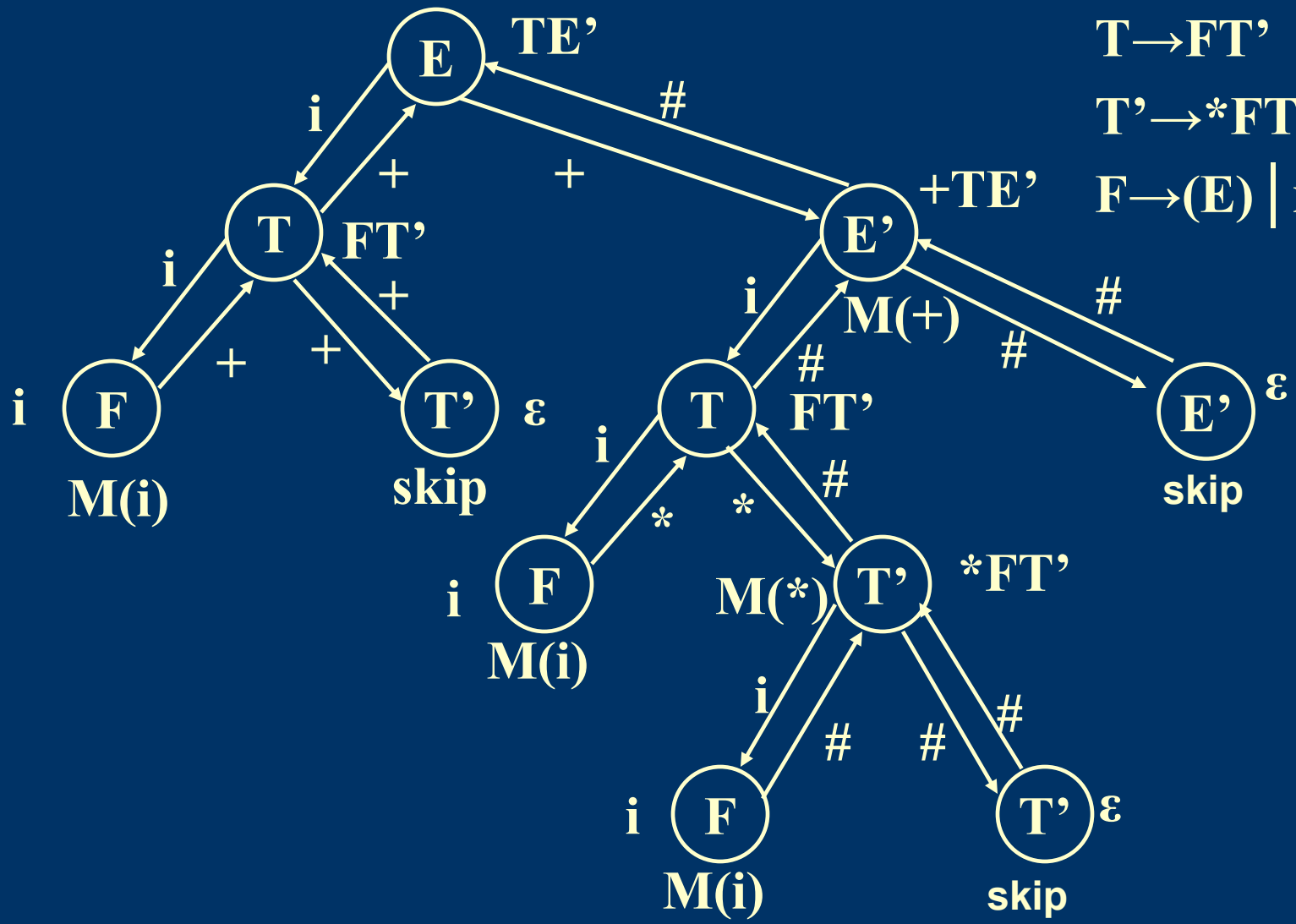
E→TE’

E’→+TE’ | ε

T→FT’

T’→*FT’ | ε

F→(E) | i



例： i+i*+i # 递归下降分析过程

语法分析主程序：Begin ReadToken; E ； Match(#) end

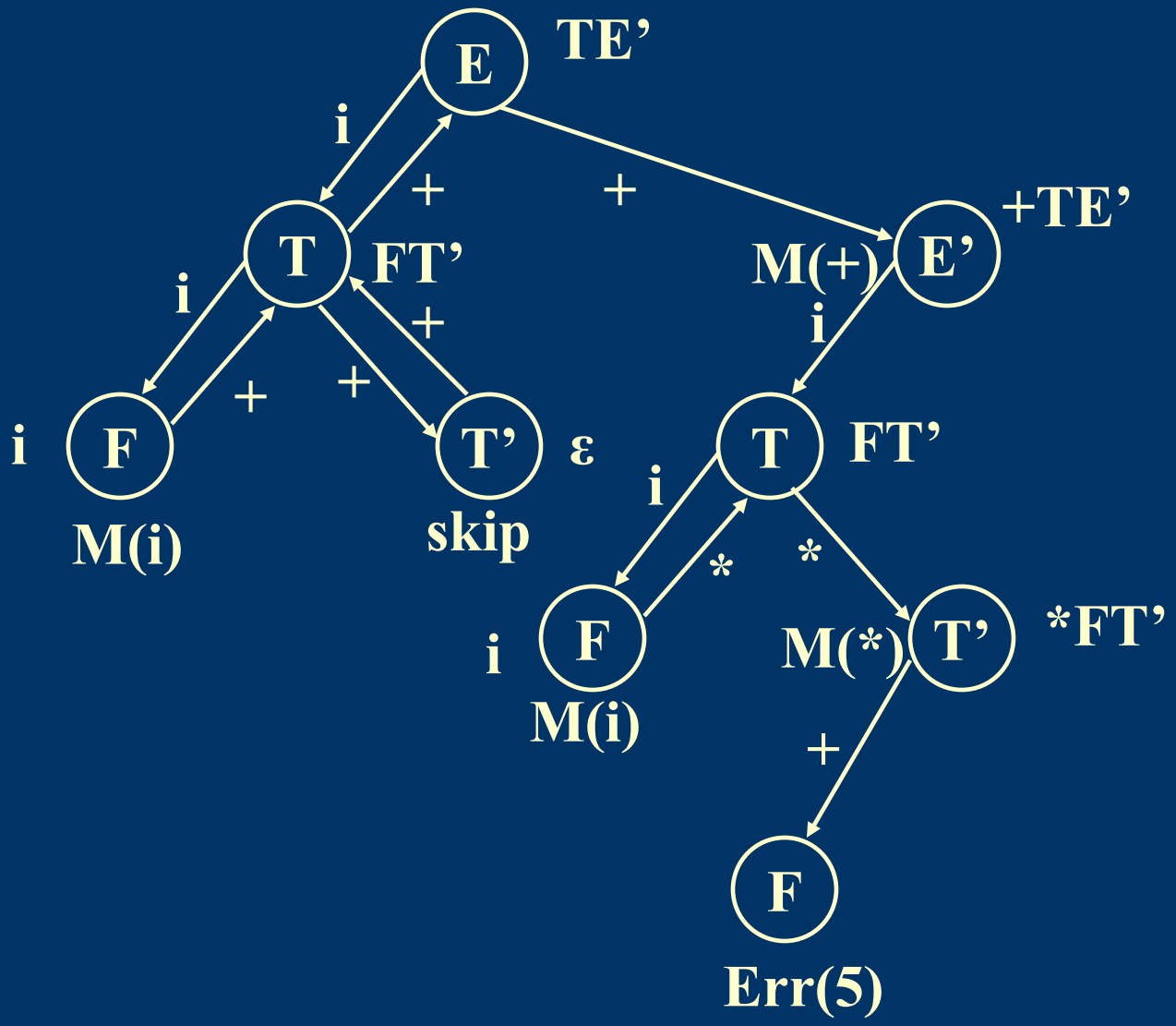
E→TE’

E’→+TE’ | ε

T→FT’

T’→*FT’ | ε

F→(E) | i



⌘ 练习题：

已知文法G[S]：

S→AB | bC

A →ε | b

B → ε | aD

C → AD | b

D → aS | c

1、计算每个非终极符的First集、Follow集.

2、计算每个产生式的Predict集合.

3、判断该文法是否为递归下降文法？

S→AB | bC
A →ε | b
B → ε | aD
C → AD | b
D → aS | c

文法符号	First集		
	$step_1$	$step_2$	$step_3$
S	{ }	{b }	{b, a,ε }
A	{ε}	{b, ε }	{b, ε}
B	{ε}	{a, ε}	{a, ε}
C	{ }	{b}	{b,a,c}
D	{ }	{a, c }	{a, c }

S→AB | bC
A →ε | b
B → ε | aD
C → AD | b
D → aS | c

文法符号	First集
S	{ a, b, ε }
A	{ b, ε}
B	{ a, ε}
C	{ a, b, c }
D	{ a, c }

文法符号				Follow集		
	step1	S→AB	S→bC	B → aD	C → AD	D → aS
S	{#}	{#}	{#}	{#}	{#}	
A	{ }	{a,# }	{a,# }	{a,# }	{a,#,c }	
B	{ }	{#}	{#}	{#}	{#}	
C	{ }	{ }	{#}	{#}	{#}	
D	{ }	{ }	{ }	{#}	{ #}	

Predict(S→AB) = { a,b,# }

Predict(S→ bC) = { b }

Predict(A → ε) = { a, c , # }

Predict(A→b) = { b }

Predict(B → ε) = { # }

Predict(B → aD) = {a }

Predict(C →AD) = {a,b,c }

Predict(C →b) = { b }

文法符号	First集	Follow集
S	{ a, b, ε }	{ # }
A	{ b, ε}	{ a, c, # }
B	{ a, ε}	{# }
C	{ a, b, c }	{# }
D	{ a, c }	{# }

由于：Predict(S→AB) ∩Predict(S→ bC) = { b }

Predict(C →AD) ∩Predict(C →b) = { b }

所以该文法不是递归下降文法.