第4章 自顶向下语法分析

- 4.1 自顶向下语法分析的关键问题
- 4.2 三个集合
- 4.3 递归下降语法分析
- 4.4 LL(1)语法分析

4.1 自顶向下语法分析的关键问题

- 自顶向下分析是从文法的开始符出发,尽可能找到一个合适的推导，使之生成源程序的过程. 关键问题:
  - 选择哪个非终极符进行推导
    - 为方便，选最左的非终极符
  - 对于选定的非终极符A,选择哪个产生式推导
    - 根据预测符集Predict(A→γ)选择
    - Predict(A→γ) = {a | S $\Rightarrow^{+}$ αAβ $\Rightarrow$ α γ β $\Rightarrow^{+}$ α a β'}，其中 α∈T*，S 是开始符.

4.2 三个集合

- 4.2.1 First(α)
- 4.2.2 Follow(A)
- 4.2.3 Predict(A→α)
- 4.2.4 确定的自顶向下语法分析条件

4.2.1 Fisrt (α)

- First(α)= {a | α⇒*aβ, a∈V_T},
- 如果 α⇒*ε 则 First(α)= First(α) ∪ {ε}
- 如何计算First(α)?
  - α = ε, First(α) = {ε}
  - α = a, a∈T, First(α) = {a}
  - α = A, A∈N, 且A→β_1, A→β_2, …, A→β_n,
    First(α) =∪ First(β_j) , j∈1,..,n
  - α = X_1 X_2 …… X_{i-1} X_i …… X_m X_i ∈ V
    若X_1⇒*ε, …, X_{i-1} ⇒* ε, First(α) = ∪ {First(X_k)-ε} ,
    k∈1,..,i-1.
    若 X_k⇒*ε, …, k∈1,..,m, First(α) = ∪ First(X_k)

4.2.1 Fisrt (α)

- 例4.1 文法G的产生式如下：
- $S \rightarrow AB \mid bC$
- $A \rightarrow \varepsilon \mid b$
- $B \rightarrow \varepsilon \mid aD$
- $C \rightarrow AD \mid b$
- $D \rightarrow aS \mid c$

First(AB) =
First(bC) = { b }
First(ε) = {ε}
First(b) = { b }
  First(A) = { ε, b}
First(ε) = {ε}
First(aD) = { a }
  First(B) = { ε, a}
    First(AB) = { ε, b, a}
First(AD) =
First(b) = { b }
First(aS) = {a}
First(c) = { c }
  First(D) = {a,c}
    First(AD) = { b, a, c}

4.2.2 Follow(A)

- Follow(A)={a|S⇒*αAβ,且a∈First(β), a∈T, α , β ∈ V*}. 规定#∈Follow(S).
- 若ε∈First(β), Follow(A)=Follow(A)∪Follow(S).
- 计算Follow(A):
- [1]. 对于每个非终极符A ∈ N,令Follow(A) = { };
- [2]. 对于开始符S,令Follow(S) = {#};
- [3].对于每个产生式A→αEβ (E∈N)：
  - 1). **Follow[E]=Follow[E]∪(First(β)-{ε});**
  - **2).如果ε∈First(β)，则Follow[E]=Follow[E]∪Follow[A]**
- [4].重复[3]，直至Follow(E)收敛.

4.2.2 Follow(A)

- 例4.2,文法G的产生式如下:
- $S \rightarrow AB \mid bC$
- $A \rightarrow \varepsilon \mid b$
- $B \rightarrow \varepsilon \mid aD$
- $C \rightarrow AD \mid b$
- $D \rightarrow aS \mid c$

初始： Follow(S) = {#},
Follow(A) =Follow(B)=Follow(C)=Follow(D) = { }

第一次迭代:

Follow(S) = Follow(S) ∪ Follow(D), Follow(D) = ?
Follow(A) = ?

First(B) = { ε, a}, ε∈First(B)

Follow(A) = Follow(A) ∪ {First(B)-{ε}} ∪ Follow(S) = {a,#}

First(D) = { a, c }, ε∉First(D)

Follow(A) = Follow(A) ∪ First(D) = {#, a,c}

Follow(B) = Follow(B) ∪ Follow(S) = {#}

Follow(C) = Follow(C) ∪ Follow(S) = {#}

Follow(D) = Follow(D) ∪ Follow(B) ∪ Follow(C)= {#}

Follow(D) = Follow(D) ∪ Follow(B) ∪ Follow(C)= {#}

4.2.2 Follow(A)

- 例4.2,文法G的产生式如下:
- $S \rightarrow AB \mid bC$
- $A \rightarrow \varepsilon \mid b$
- $B \rightarrow \varepsilon \mid aD$
- $C \rightarrow AD \mid b$
- $D \rightarrow aS \mid c$

将Follow(D)带入Follow(S), Follow(S) = {#},集合Follow(S)没有扩大,进而也不会引起Follow(A), Follow(B), Follow(C)和Follow(D)的变化,因此算法收敛.

最终结果:

Follow(S) = Follow(S) ∪ Follow(D)= {#}
Follow(A) = {#,a,c}
Follow(B) = {#}
Follow(C) = {#}
Follow(D) = {#}

4.2.3 Predict(A→α)

- Predict(A→α) = First(α), if ε ∉ First(α);
- (First(α)- ε) ∪ Follow(A), if ε ∈ First(α);
- 例4.3,文法G的产生式如下:
- S → AB | bC
- A → ε | b
- B → ε | aD
- C → AD | b
- D → aS | c
-

Predict(S → AB) = {First(AB)-{ε}} ∪Follow(S)
={ b, a, #}

Predict(S → bC) = First(bC) = { b}

Predict(A → ε) = {First(ε)-{ε}} ∪Follow(A)
= { a, c, #}

Predict(A → b) = First(b) = { b}

Predict(B → ε) = {First(ε)-{ε}} ∪Follow(B)
= {#}

Predict(B → aD) = First(aD) = { a }

Predict(C → AD) = First(AD)={ b, a, c}

Predict(C → b) = First(b) = { b}

Predict(D → aS) = First(aS) = { a }

Predict(D → c) = First(c) = { c}

4.2.3 Predict(A→α)

- 应用Predict集推导的例子
- 例4.4 文法G的产生式如下,判断句型abcd是否符合该文法.

- $S \rightarrow aBd$
- $B \rightarrow d$
- $B \rightarrow c$
- $B \rightarrow bB$

$S \Rightarrow aBd \Rightarrow ?$ B应选则哪个产生式呢?
根据Predict($B \rightarrow d$), Predict($B \rightarrow c$), Predict($B \rightarrow bB$),显然应该选择$B \rightarrow bB$

$S \Rightarrow aBd \Rightarrow abBd \Rightarrow ?$ B应选则哪个产生式呢?
显然选择$B \rightarrow c$.
$S \Rightarrow aBd \Rightarrow abBd \Rightarrow abcd$, 说明句型abcd符合该文法.

4.2.3 Predict(A→α)

- 应用Predict集推导的例子
- 例4.5 文法G的产生式如下,判断句型abcd是否符合该文法.

- S → aBd
- B → b
- B → c
- B → bB

S ⇒ aBd ⇒? B应选则哪个产生式呢?
根据Predict(B → b), Predict(B → c), Predict(B → bB),显然选择B → b或B → bB都可以. 若选择B → b，则推导变成:
S ⇒ aBd ⇒abd，与符号串abcd无法匹配,但不能说abcd不符合文法，因为还有一个选择没有尝试. 尝试选择B → bB, 则推导变成:
S ⇒ aBd ⇒abBd，是有可能成功的,继续选择产生式推导.

S ⇒ aBd ⇒abBd ⇒? B应选则哪个产生式呢?
显然选择B → c.

4.2.4 确定的自顶向下分析条件

- 对任意非终极符A,
  - 对A的任意两条产生式,
  - predict($A\rightarrow\beta_k$)$\cap$ predict($A\rightarrow\beta_j$)=$\varnothing$, 当k ≠ j
  - 即同一个非终极符的任意两个产生式的predict集合互不相交
- 这个条件保证:针对当前的符号和当前的非终极符,可以选择唯一的产生式来进行推导
- 当文法的产生式存在左递归或左公共前缀时,一定不会满足上述条件，需要对其进行变换,使之可以应用确定的自顶向下的语法分析方法进行分析

4.2.4 确定的自顶向下分析条件

- 左公共前缀
- $A \rightarrow \alpha\beta_1 \mid \cdots \mid \alpha\beta_n \mid \gamma_1 \mid \cdots \mid \gamma_m$
- 消除方法：提取左公共前缀(类似于数学上的提公因子)
  - $\mathbf{A \rightarrow \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_m}$
  - $\mathbf{A' \rightarrow \beta_1 \mid \ldots \mid \beta_n}$
- 注: A'是任意的新非终极符，$A' \notin N$(N是原文法的非终极符集合)
- 例4.6 文法产生式如下：
- $S \rightarrow a\,S \mid aA \mid c$
- $A \rightarrow b\,B \mid b$
- $B \rightarrow b \mid eB$

消除左公共前缀后的产生式如下：

$S \rightarrow a\,S' \mid c$　　$S' \rightarrow S \mid A$

$A \rightarrow b\,A'$　　$A' \rightarrow B \mid \varepsilon$

$B \rightarrow b \mid eB$

4.2.4 确定的自顶向下分析条件

- 左递归
- 直接左递归： $A \rightarrow A\alpha_1 | \ldots | A\alpha_n | \beta_1 | \ldots | \beta_m$
- 消除方法：左递归变成右递归

  $A \rightarrow \beta_1 A' | \ldots | \beta_m A'$

  $A' \rightarrow \alpha_1 A' | \ldots | \alpha_n A' | \varepsilon$
- 间接左递归：$S \rightarrow Ab$　$A \rightarrow Sa | b$
- 消除方法*: 非终极符排序消元法
- 例4.7: 文法产生式判断如下：
- $E \rightarrow E + T | T$
- 消除关于E的左递归, $E \rightarrow TE'$　$E' \rightarrow +TE' | \varepsilon$

4.3 递归下降语法分析

- 自顶向下语法分析方法的一种，需要满足确定的自顶向下语法分析方法的条件.
- 分析思想:
  - 对于每个非终极符N构造相应的语法分析子程序;
  - 整个语法分析程序由所有N的分析子程序构成;
  - 由于文法产生式可能存在递归，因此一些分析子程序也会存在递归，此为该方法名称的由来
- 特点:
  - 分析程序完全依赖于产生式结构，容易构造
  - 频繁调用子程序，分析效率低下，属于理论上的方法

4.3 递归下降语法分析

- 例4.8 产生式文法如下：
- $S \rightarrow aBd$
- $B \rightarrow d$
- $B \rightarrow c$
- $B \rightarrow bB$
-

构建递归下降语法分析程序步骤如下：

1.为每一条产生式计算predict集合，判断是否满足分析条件。

2.为每一个非终极符写分析子程序：

```
S( ) { match(a); B( ); match(d); }
B( ) { switch(token)
          {case d : match(d); break;
           case  c: match(c); break;
           case b: match(b); B(); break;
          default: error();}}
main() {read( );  S();}
match(char w)  {if (w!=token) error() else read();}
```

4.3 递归下降语法分析

- 例4.9 产生式文法如下：
- $S \rightarrow aBd$
- $B \rightarrow d$
- $B \rightarrow \varepsilon$
- $B \rightarrow bB$

构建递归下降语法分析程序步骤如下：

1.为每一条产生式计算predict集合，判断是否满足分析条件。

$Predict(B \rightarrow \varepsilon) = \{d\}$

$Predict(B \rightarrow d) = \{d\}$

$Predict(B \rightarrow bB) = \{b\}$

其中$Predict(B \rightarrow \varepsilon) \cap Predict(B \rightarrow d) = \{d\}$ 不满足递归下降分析条件，因此无法写出递归下降分析程序.

4.3 递归下降语法分析

- 例4.10 产生式文法如下：
- $S \rightarrow aBd$
- $B \rightarrow a$
- $B \rightarrow \varepsilon$
- $B \rightarrow bB$

构建递归下降语法分析程序步骤如下：

1.为每一条产生式计算predict集合，判断是否满足分析条件。

2.为每一个非终极符写分析子程序：

```
S( ) { match(a); B( ); match(d); }
B( ) { switch(token)
          {case a : match(a);break;
          case d: skip;  break; //case ε: match(ε);  NO!
           case b: match(b); B();break;
          default: error();}}
main() {read( );  S();}
match(char w)  {if (w!=token) error() else read();}
```

4.4 LL(1)语法分析

- LL(1)方法同递归下降方法一样，属于自顶向下的语法分析,两种方法的分析条件和分析能力相同
- 分析思想:
  - 采用LL(1)分析表记录每个产生式的预测符
  - 采用LL(1)分析驱动程序控制分析过程
  - 采用符号栈记录需要推导的句型
- 特点:
  - 不同的文法，LL(1)驱动程序通用
  - 分析能力有限

4.4.1 LL(1)分析过程

4.4.2 LL(1)分析表

- LL(1)分析表是一个二维表格：每一行对应文法的一个非终极符，每一列对应文法的一个终极符，还有一列对应输入串的结束符#.
  - $LL1(A_i, a_j) = A_i \rightarrow \alpha$, 如果$a_j \in predict(A_i \rightarrow \alpha)$
  - $LL1(A_i, a_j)$ = error或⊥, 如果a_j 不属于A_i的任何一条产生式的Predict集

	a_1	…	a_n	#
A_1				
…	….	….	…	
Am				

4.4.2 LL(1)分析表

- 例4.11 为下列文法构造LL(1)分析表

(1) E → TE’	{ i, n, (}
(2) E’ → + TE’	{+}
(3) E’ → ε	{#,)}
(4) T → FT’	{i,n,(}
(5) T’ → * F T’	{*}
(6) T’ → ε	{),+, # }
(7) F → (E)	{ (}
(8) F → i	{i}
(9) F → n	{n}

	+	*	(	)	i	n	#
E			1		1	1	
E’	2			3			3
T			4		4	4	
T’	6	5		6			6
F			7		8	9	

4.4.3 LL(1)驱动程序

- [1] 初始化： Stack :=empty； Push(S)；
- [2] 读下一个输入符： Read(a)；
- [3] 若当前格局是(,#)，则成功结束；否则转下；
- [4] 设当前格局为（..... X, a.....),
- //X是符号栈栈顶,a是输入流当前符号
- 若 X∈T & X=a，则{ Pop(1)； Read(a)； goto [3]}
- 若 X∈T & X≠a 则 error；
- 若 X∈N，则：
- if LL1(X,a)=X→$Y_1Y_2...Y_n$
- then {Pop(1)； Push(Yn ,.....,Y1)； goto[3]}
- else error

4.4.4 LL(1)分析实例

- 例4.12

P:
(1) S → aBd
(2) B → d
(3) B → c
(4) B → bB

产生式	Predict集
(1)	{a}
(2)	{d}
(3)	{c}
(4)	{b}

	a	b	c	d	#
S	(1)				
B		(4)	(3)	(2)	

LL1分析表

(S, abcd#) ⇒ (dBa, abcd#) ⇒ (dB, bcd#) ⇒ (dBb, bcd#) ⇒ (dB, cd#)
⇒ (dc, cd#) ⇒ (d, d#) ⇒(, #) 识别成功

(S, abdb#) ⇒ (dBa, abdb#) ⇒ (dB, bdb#) ⇒ (dBb, bdb#) ⇒ (dB, db#)
⇒ (dd, db#) ⇒ (d,b) 识别失败

作 业

- G = {T, N, S, P}
- T = {- , (,), id}
- N = {E, ET, V, VT}
- S = E
- P = { $E \rightarrow -E$ $ET \rightarrow \varepsilon$
  $E \rightarrow (E)$ $V \rightarrow id\ VT$
  $E \rightarrow V\ ET$ $VT \rightarrow (E)$
  $ET \rightarrow -E$ $VT \rightarrow \varepsilon$ }
- [1] 为文法G写一个递归下降程序
- [2] 为文法G构造LL1分析表