编译原理

北方工业大学信息学院
School of Information Science and Technology,
North China University of Technology
束劼
shujie@ncut.edu.cn
瀚学楼1122，88801615

第四章 语法分析
-自上而下分析

第四章 语法分析-自上而下分析

- 本章目录

4.1 语法分析器的功能
4.2 自上而下分析面临的问题
4.3 LL(1)分析法
4.4 递归下降分析程序构造
4.5 预测分析程序
4.6 LL(1)分析中的错误处理

第四章 语法分析-自上而下分析

- 大纲要求
- 掌握：LL(1)分析法的条件，消除左递归的算法，预测分析表的构造。
- 理解：预测分析程序、递归下降分析程序的设计方法。
- 了解：语法分析器的功能。

4.1 语法分析器的功能

4.1 语法分析器的功能

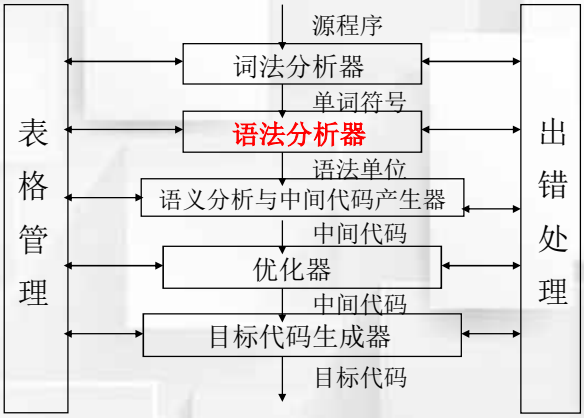


图1.1 编译程序总框

4.1 语法分析器的功能

- 语法分析器(Parser)

  语言的语法结构是上下文无关文法(context-free)。

  语法分析器的**本质**是按文法产生式，识别输入符号串是否为一个句子。建立一棵与输入符号串相匹配的语法分析树。

  语法分析方法可以分三类：

  1. 整体分析(universal)，Cocke-Younger-Kasami algorithm和Earleys algorithm;
  2. 自上而下分析法(top-down)
  3. 自下而上分析法(bottom-up)

  } 常用的两种方法

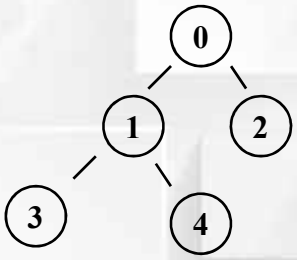
4.2 自上而下分析面临的问题

4.2 自上而下分析面临的问题

- 自上而下分析的方法

  深度优先的树访问方式

  从根结点开始，递归的访问各个结点的子结点。优先访问离根结点最远的未访问的结点，不一定要从左到右的访问，但可以设定先左后右的方式访问。



深度优先的访问顺序：
0，1，3，4，2
设定先左后右

9

4.2 自上而下分析面临的问题

- 自上而下分析面临的问题
- 问题一：文法存在左递归，将使自上而下的分析过程陷入无限循环。

$P \overset{+}{\Rightarrow} Pa$

P无法匹配任何输入串，回溯，重新要求P进行新的匹配

10

4.2 自上而下分析面临的问题

- 自上而下分析面临的问题
- 问题二：**回溯**是一项复杂而费时的工作，须废弃已做的许多工作，恢复到前面的某一情况，效率很低。

文法中非终结符A的产生式右部称为A的候选式，如果有多个候选式左端第一个符号相同，则语法分析程序无法根据当前输入符号选择产生式，只能试探。若不能匹配，则要**回溯**。

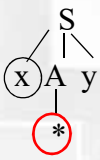
11

4.2 自上而下分析面临的问题

- 自上而下分析面临的问题
- 问题三：遇终结符匹配成功时，可能时暂时的成功。这就是**虚假匹配**。

输入串x*y

x**y



S⇒xAy

⇒x*y

试用A→*展开A

12

4.2 自上而下分析面临的问题

- 自上而下分析面临的问题
- 问题四：最终报告分析不成功时，难于知道输入串中出错的确切位置。出错位置未知。
- 问题五：带回溯的自上而下分析实际上采用了一种穷尽的试探法，效率很低，代价极高。这是一种理论上的方法，实践上价值不大。

4.3 LL(1)分析法

4.3 LL(1)分析法

- LL(1)分析法的目的

  构造不带回溯的自上而下分析算法

  要消除文法的左递归性

  克服回溯

4.3.1 左递归的消除

4.3.1 左递归的消除

- 左递归的消除 Elimination of Left Recursion

直接消除见诸于产生式中的左递归：假定关于非终结符P的产生式为

$P \rightarrow P\alpha \mid \beta$

其中β不以P开头， α不等于ε。

可以把P的产生式等价地改写为如下的非直接左递归形式：

$P \rightarrow \beta P'$

$P' \rightarrow \alpha P' \mid \varepsilon$

17

4.3.1 左递归的消除

- 左递归的消除 Elimination of Left Recursion

$P \rightarrow P\alpha_1 \mid P\alpha_2 \mid \ldots \mid P\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n \mid$

没有β_i以P开头

可以把P的产生式改写非直接左递归形式：

$P \rightarrow \beta_1 P' \mid \beta_2 P' \mid \ldots \mid \beta_n P' \mid$

$P' \rightarrow \alpha_1 P' \mid \alpha_2 P' \mid \ldots \mid \alpha_m P' \mid \varepsilon$

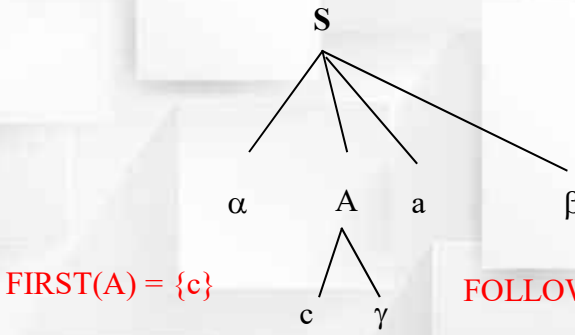
可以消除所有左递归吗?

18

4.3.2 消除回溯、提左因子

4.3.2 消除回溯、提左因子

- FIRST 和 FOLLOW



FIRST(A) = {c}

FOLLOW(A) = {a}

4.3.2 消除回溯、提左因子

- 提取左因子

  把条件句通用化为

  $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$ 难以选择

  $A \rightarrow \alpha A'$

  $A' \rightarrow \beta_1 \mid \beta_2$ 简单易选

4.3.3 LL(1)分析条件

4.3.3 LL(1)分析条件

- LL(1) Left-to-right Left-most

  第一个**L** Left-to-right 从左到右扫描输入字符串

  第二个**L** Left-most 最左推导

  (**1**) Lookahead 每次检查一个输入字符

4.3.3 LL(1)分析条件

- 语法G的LL(1)文法

  语法G是LL(1)文法，当且仅当，$A \rightarrow \alpha \mid \beta$ 是语法G的两个不同的产生式。并且，**满足下列条件**： 避免二义性

  1. α和β导出的字符串的首字符，不能是同一个终结符a；

  2. α和β中最多只有1个可以导出空字符串，即ε；

  3. 如果$\beta \overset{*}{\Rightarrow} \varepsilon$， 则α导出的字符串的首字符不能是FOLLOW(A)中的终结符。 如果$\alpha \overset{*}{\Rightarrow} \varepsilon$，则β也必须符合同样条件。

E’	{+, ε}	{) , # }

4.3.3 LL(1)分析条件

- LL(1)文法在语义分析中的作用

LL(1)文法用来构造二维语法分析表M[A, a]，其中A是非终结符，a是终结符或者输入结束符#。

输入字符中没有ε

语法分析表M

非终结符	输入字符					
	i	+	*	(	)	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
T	$T \rightarrow FT'$			$T \rightarrow FT'$		

4.3.3 LL(1)分析条件

- LL(1)文法在语义分析中的作用

① 如果终结符a在FIRST(α)中，则添加A → α到M[A,a]

E → TE’ FIRST(TE’) = FIRST(T) = {(, i}

M[E, (]和M[E, i]加入表格

语法分析表M

非终结符	输入字符					
	i	+	*	(	)	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
T	$T \rightarrow FT'$			$T \rightarrow FT'$		

4.3.3 LL(1)分析条件

- 语法G的LL(1)文法

② 如果ε在FIRST(α)中，则对FOLLOW(A)中的每个终结符b，添加A → α 到M[A, b]。如果ε在FIRST(α)，同时#在FOLLOW(A)，添加A → α 到M[A, #]。

E′→+TE′ | ε　FIRST(+TE’) = {+}，添加*E′→+TE′*到M[E’, +]

E’→ ε , FOLLOW(E’) = {), #}，添加*E’*→ ε到M[E’,)]和M[E’, #]

非终结符	输入字符					
	i	+	*	(	)	#
E’		*E′→+TE′*			*E’*→ ε	*E’*→ ε

4.3.3 LL(1)分析条件

- 语法G的LL(1)文法

③ 除了以上两条以外，如果M[A, a]没有填写任何产生式，则把M[A, a]设为error(通常表现为空白)。

非终结符	输入字符					
	i	+	*	(	)	#
E’		*E′→+TE′*			*E* → ε	*E* → ε

4.4 递归下降分析程序构造

4.4 递归下降分析程序构造

- 递归下降分析程序构造要求

  构造不带回溯的自上而下分析程序

  ① 要消除文法的左递归性

  ② 克服回溯

4.4 递归下降分析程序构造

- 递归下降分析程序构造

  当一个文法**满足LL(1)条件**时，我们就可以为它构造一个不带回溯的自上而下分析程序。

  这个分析程序是由**一组递归过程**组成的，**每个过程对应文法的一个非终结符**。

  这样的一个分析程序称为**递归下降分析器**。

4.4 递归下降分析程序构造

- 递归下降分析程序构造

  几个全局过程和变量：

ADVANCE	读入IP 指向的输入符号到SYM中，把输入串指示器IP指向下一个输入符号
SYM	IP当前所指的输入符号
ERROR	出错处理程序

4.4 递归下降分析程序构造

- 递归下降分析程序构造

每个非终结符都有对应的递归过程，在分析过程中，当需要从某个非终结符出发进行展开(推导)时，就调用这个非终结符对应的子程序。

非终结符号的分析子程序的**功能**是：用产生式右部符号串来匹配输入串。

假定在开始工作前，输入串指示器IP指向第一个输入符号。当每个子程序工作完毕之后，IP总是指向下一个未处理的符号。

33

4.4 递归下降分析程序构造

- 递归下降分析程序构造

例如 $E' \rightarrow +TE' \mid \varepsilon$

E′ 有两个候选；
当面临**输入符号**+时，**第一个候选**进入工作；
当面临**任何其它输入符号**时，**第二个候选**进入工作，由于第二个候选为ε，E′就自动认为获得了匹配；

如果使用LL(1)文法，则该判断该输入符号是否属于FIRST(E′)。

34

4.4 递归下降分析程序构造

- 递归下降分析程序构造

E → TE’　E’ → +TE’ | ε

T →FT’　T’ →*FT’ | ε

F →(E) | i

```
PROCEDURE T;
BEGIN
        F; T′
END;
```

```
PROCEDURE E;
BEGIN
        T; E′
END;
```

```
PROCEDURE E′;
  IF SYM= ‘+’ THEN
  BEGIN
     ADVANCE;
    T; E′
  END;
```

35

4.4 递归下降分析程序构造

- 递归下降分析程序构造

E → TE’　E’ → +TE’ | ε

T →FT’　T’ →*FT’ | ε

F →(E) | i

```
PROCEDURE T′;
IF SYM= ‘*’ THEN
BEGIN
  ADVANCE;
  F; T′
END;
```

```
PROCEDURE F;
IF SYM= ‘i’ THEN
   ADVANCE
ELSE
    IF SYM= ‘(’ THEN
    BEGIN
       ADVANCE;
       E;
       IF SYM= ‘)’ THEN
           ADVANCE
       ELSE ERROR
  END
  ELSE ERROR; ???
```

36

4.5 预测分析程序

4.5 预测分析程序

4.5.1 预测分析程序工作过程

Nonrecursive Predictive Parsing

4.5.2 预测分析表的构造

Construction of a Predictive Parsing Table

4.5.1 预测分析程序工作过程

4.5.1 预测分析程序工作过程

- 预测分析程序工作过程要点

① 分析时，显示的使用栈Stack，而不是隐示的使用递归，栈底是#(没有待判断符号时在顶部)

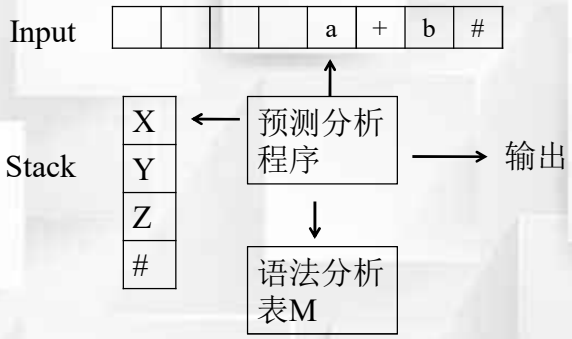
② 语法分析使用最左推导

③ 输入字符也用栈Input存储，栈底是#(没有输入符号时在顶部)

④ 使用语法分析表M[A, a]

4.5.1 预测分析程序工作过程

- 预测分析程序工作过程



语法分析程序X(非终结符)，也是Stack的栈顶元素

输入字符a，也是Input的栈顶元素

如果X是非终结符，并且M[X, a]产生式存在，则采用该产生式进行语法分析

如果X是终结符呢？

4.5.1 预测分析程序工作过程

- 预测分析程序工作过程中栈的主要操作

① 若X＝a＝‘#’，则宣布分析成功，停止分析过程。

② 若X＝a ≠‘#’，则宣布匹配成功，把X从STACK栈顶逐出，使a指向下一个输入符号。

③ X是一个非终结符，则查看分析表M。把M[X, a]对应的产生式的右部符号串按反序一一推进Stack栈。(若右部符号为ε，不推任何东西进栈)；若M[X，a]中存放着“出错标志”，则调用出错诊察程序ERROR。

4.5.1 预测分析程序工作过程

- 语法 E → TE’ E’ → +TE’ | ε
  T →FT’ T’ →*FT’ | ε
  F →(E) | i 语法分析表M

非终结符	输入字符					
	i	+	*	(	)	#
E	*E → TE’*			*E → TE’*		
E’		*E → TE’*			*E’→ ε*	*E’→ ε*
T	*T → FT’*			*T → FT’*		
T’		*T’→ ε*	*T’→ *FT’*		*T’→ ε*	*T’→ ε*
F	*F → i*			*F → (E)*		

43

E → TE’ E’ → +TE’ | ε T →FT’ T’ →*FT’ | ε F →(E) | i

非终结符	输入字符					
	i	+	*	(	)	#
E	*E → TE’*			*E → TE’*		
E’		*E → TE’*			*E’→ ε*	*E’→ ε*
T	*T → FT’*			*T → FT’*		
T’		*T’→ ε*	*T’→ *FT’*		*T’→ ε*	*T’→ ε*
F	*F → i*			*F → (E)*		

输入： i + i * i

尝试找到最左推导

E ⇒lm TE’ ⇒lm FT’E’ ⇒lm i T’E’ ⇒lm i E’ ⇒lm i + TE’ ⇒lm …

44

输入： i + i * i

语法分析表M

非终结符	输入字符					
	i	+	*	(	)	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E \rightarrow TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow i$			$F \rightarrow (E)$		

① M[E, i] ② M[T, i] ② M[F, i]

Matched	Stack	Input	Action
	E#	i+i*i#	
	TE'#	i+i*i#	output $E \rightarrow TE'$
	FT'E'#	i+i*i#	output $T \rightarrow FT'$

45

MATCHED	STACK	INPUT	ACTION
	$E\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	
	$TE'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $E \rightarrow TE'$
	$FT'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $T \rightarrow FT'$
	$\mathbf{id}\ T'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $F \rightarrow \mathbf{id}$
id	$T'E'\$$	$+ \mathbf{id} * \mathbf{id}\$$	match id
id	$E'\$$	$+ \mathbf{id} * \mathbf{id}\$$	output $T' \rightarrow \epsilon$
id	$+\ TE'\$$	$+ \mathbf{id} * \mathbf{id}\$$	output $E' \rightarrow + TE'$
id +	$TE'\$$	$\mathbf{id} * \mathbf{id}\$$	match +
id +	$FT'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $T \rightarrow FT'$
id +	$\mathbf{id}\ T'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $F \rightarrow \mathbf{id}$
id + id	$T'E'\$$	$* \mathbf{id}\$$	match id
id + id	$* FT'E'\$$	$* \mathbf{id}\$$	output $T' \rightarrow * FT'$
id + id *	$FT'E'\$$	$\mathbf{id}\$$	match *
id + id *	$\mathbf{id}\ T'E'\$$	$\mathbf{id}\$$	output $F \rightarrow \mathbf{id}$
id + id * id	$T'E'\$$	$\$$	match id
id + id * id	$E'\$$	$\$$	output $T' \rightarrow \epsilon$
id + id * id	$\$$	$\$$	output $E' \rightarrow \epsilon$

46

4.5.2 预测表的构造

4.5.2 预测分析表的构造

- 预测分析表的构造

需要FIRST(α)和FOLLOW(A)

有语法G
E→TE′
E′→+TE′ | ε
T→FT′
T′→*FT′ | ε
F→(E) | i

非终结符	FIRST	FOLLOW
E	{ (, i}	{) , # }
T	{ (, i}	{+,), #}
E’	{+, ε}	{) , # }
T’	{*, ε}	{+,), #}
F	{ (, i}	{ * , +,), # }

有语法G
E→TE′
E′→+TE′ | ε
T→FT′
T′→*FT′ | ε
F→(E) | i

非终结符	FIRST	FOLLOW
E	{ (, i}	{) , # }
T	{ (, i}	{+,), #}
E’	{+, ε}	{) , # }
T’	{*, ε}	{+,), #}
F	{ (, i}	{ * , +,), # }

第一步，对每个终结符a在FIRST(α)中，把产生式A → α 加入表M[A, a]

非终结符	输入字符					
	i	+	*	(	)	#
E	*E → TE’*			*E → TE’*		
E’		*E → TE’*				

49

有语法G
E→TE′
E′→+TE′ | ε
T→FT′
T′→*FT′ | ε
F→(E) | i

非终结符	FIRST	FOLLOW
E	{ (, i}	{) , # }
T	{ (, i}	{+,), #}
E’	{+, ε}	{) , # }
T’	{*, ε}	{+,), #}
F	{ (, i}	{ * , +,), # }

第二步，如果ε在FIRST(α)中，则FOLLOW(A)中的每个终结符b，把产生式A → α 加入表M[A, b]。如果ε在FIRST(α)中，同时#在FOLLOW(A)中，把产生式A → α 加入表M[A, #]。

非终结符	输入字符					
	i	+	*	(	)	#
E’		*E → TE’*			*E’→* ε	*E’→* ε
T’		*T’→* ε	*T’→ *FT’*		*T’→* ε	*T’→* ε

50

4.5.2 预测分析表的构造

- LL(1)文法的证明

如果G是左递归或二义的，那么，M至少含有一个多重定义入口。因此，消除左递归和提取左因子将有助于获得无多重定义的分析表M。

可以证明，一个文法G的预测分析表M不含多重定义入口，当且仅当该文法为LL(1)的。

4.5.2 预测分析表的构造

- LL(1)文法的证明

如果G是左递归或二义的，那么，M至少含有一个多重定义入口。因此，消除左递归和提取左因子将有助于获得无多重定义的分析表M。可以证明，一个文法G的预测分析表M不含多重定义入口，当且仅当该文法为LL(1)的。

证明条件：一个文法G，其分析表M不含多重定义入口(即分析表中无二条以上产生式)，则称它是一个LL(1)文法。

4.5.2 预测分析表的构造

- LL(1)文法的证明

例如 文法G(S):

S → iCtS | iCtSeS | a

C → b

提取左因子之后，改写成：

G′(S):

S → iCtSS′ | a

S′ →eS | ε

C → b

FIRST(S)={i, a}
FIRST(S′)={e, ε}
FIRST(C)={b}

FOLLOW(S)={e, #}
FOLLOW(S′)={e, #}
FOLLOW(C) ={t}

4.5.2 预测分析表的构造

- LL(1)文法的证明

文法G′(S) 不是LL(1)文法

例如 文法G′(S):

S → iCtSS′ | a

S′ →eS | ε

C → b

FIRST(S)={i, a}
FIRST(S′)={e, ε}
FIRST(C)={b}

FOLLOW(S)={e, #}
FOLLOW(S′)={e, #}
FOLLOW(C) ={t}

	a	b	e	i	t	#
S	S→a			S→iCtSS′		
S′			S′→ε S′→eS			S′→ε
C		C→b				

4.6 LL(1)中的错误处理

4.6 LL(1)分析中的错误处理

- LL(1)分析中的错误情况

① 栈顶的终结符与当前输入符号不匹配

② 非终结符A处于栈顶，面临的输入符号为a，但分析表M中的M[A, a]为空

问题：无法弹出终结符或非终结符A，分析程序不能继续

4.6 LL(1)分析中的错误处理

- LL(1)分析中的错误解决方法

思想：跳过输入串中的一些符号，直到遇到“同步符号”为止。

方法：建立非终结符A的同步符号集。

4.6 LL(1)分析中的错误处理

- LL(1)分析中的错误解决方法

考虑建立同步符号集的方法：

① 把FOLLOW(A)中的所有符号放入非终结符A的同步符号集。

② 只用FOLLOW(A)是不够的。例如C语言中的分号是语句的结束符，分号不在FOLLOW(A)，如果缺少分号，下一句的开始关键字右可能被跳过。

③ 把FIRST(A)中的符号也加入非终结符A的同步符号集，当FIRST(A)中的一个符号在输入中出现时，则可以根据A恢复语法分析。

4.6 LL(1)分析中的错误处理

- LL(1)分析中的错误解决方法

考虑建立同步符号集的方法：

④ 如果一个非终结符产生空串，推导ε的产生式可以作为缺省，这样可以推迟某些错误检查，从而减少在错误恢复期间的必须考虑的非终结符

⑤ 如果不能匹配栈顶的终结符号，简单的想法是弹出该终结符，并发出一条信息，说明已插入这个终结符，继续语法分析。但是，可能会让该单词符号的同步符号集包含所有其他的单词符号。

4.6 LL(1)分析中的错误处理

- LL(1)分析中的错误处理

当使用的语法是常见语法时，使用FIRST和FOLLOW中的符号组建的同步符号集，处理错误效果很好。

把语法分析表M中空缺的地方，并且属于FOLLOW的终结符，都填上“synch”，表明是同步符号集。

4.6 LL(1)分析中的错误处理

- LL(1)分析中的错误处理

语法分析表M

非终结符	输入字符					
	i	+	*	(	)	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E \rightarrow TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$T \rightarrow i$	synch	synch	$F \rightarrow (E)$	synch	synch

61

4.6 LL(1)分析中的错误处理

- LL(1)分析中的错误处理

三种情况的处理方式：

① 如果M[A, a]是空的，则忽略输入符号

② 如果M[A, a]是”synch”，则非终结符A被弹出并准备尝试恢复编译

③ 如果Stack栈顶的字符与输入字符不匹配，则弹出Stack栈顶的字符

62

4.6 LL(1)分析中的错误处理

- LL(1)分析中的错误处理

  如果输入的是带有错误的字符串 **) i * + i**

语法分析表M

非终结符	输入字符					
	i	+	*	(	)	#
E	$E \rightarrow TE'$			$E \rightarrow TE'$	synch	synch
E'		$E \rightarrow TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$	synch		$T \rightarrow FT'$	synch	synch
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$T \rightarrow i$	synch	synch	$F \rightarrow (E)$	synch	synch

63

STACK	INPUT	REMARK
$E\,\$$	$)\,\mathbf{id} * + \mathbf{id}\,\$$	error, skip)
$E\,\$$	$\mathbf{id} * + \mathbf{id}\,\$$	id is in FIRST(E)
$TE'\,\$$	$\mathbf{id} * + \mathbf{id}\,\$$	
$FT'E'\,\$$	$\mathbf{id} * + \mathbf{id}\,\$$	
$\mathbf{id}\,T'E'\,\$$	$\mathbf{id} * + \mathbf{id}\,\$$	
$T'E'\,\$$	$* + \mathbf{id}\,\$$	
$*\,FT'E'\,\$$	$* + \mathbf{id}\,\$$	
$FT'E'\,\$$	$+\,\mathbf{id}\,\$$	error, $M[F, +] =$ synch
$T'E'\,\$$	$+\,\mathbf{id}\,\$$	F has been popped
$E'\,\$$	$+\,\mathbf{id}\,\$$	
$+\,TE'\,\$$	$+\,\mathbf{id}\,\$$	
$TE'\,\$$	$\mathbf{id}\,\$$	
$FT'E'\,\$$	$\mathbf{id}\,\$$	
$\mathbf{id}\,T'E'\,\$$	$\mathbf{id}\,\$$	
$T'E'\,\$$	$\$$	
$E'\,\$$	$\$$	
$\$$	$\$$	

64

第四章 小结

第四章 小结

4.1 语法分析器的功能
4.2 自上而下分析面临的问题
4.3 LL(1)分析法
4.4 递归下降分析程序构造
4.5 预测分析程序
4.6 LL(1)分析中的错误处理

Coursework

4.1考虑下面文法G[S]

S→Aa

A→BB

B→Sb｜c

请消除该文法的左递归。

4.2 试消除下面文法G[A] 中的左递归，并提取公共左因子，判断改写后的文法是否为LL(1)文法?

A→aABe｜a

B→Bb｜d

67

Coursework

4.3 考虑下面文法G_1:

S→a｜∧｜(T)

T→T,S｜S

（1）消去G_1的左递归。然后对每个非终结符，写出不带回溯的递归子程序。

（2）经改写后的文法是否是LL(1)的？给出它的预测分析表。

68

Coursework

4.4 对下面的文法G：

E→TE′

E′→+E｜ε

T→FT′

T′→T｜ε

F→PF′

F′→*F′｜ε

P→(E)｜a｜b｜∧

（1）计算这个文法的每个非终结符的FIRST和FOLLOW。

（2）证明这个文法是LL(1)的。

（3）构造它的预测分析表。

（4）构造它的递归下降分析程序。

69