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

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

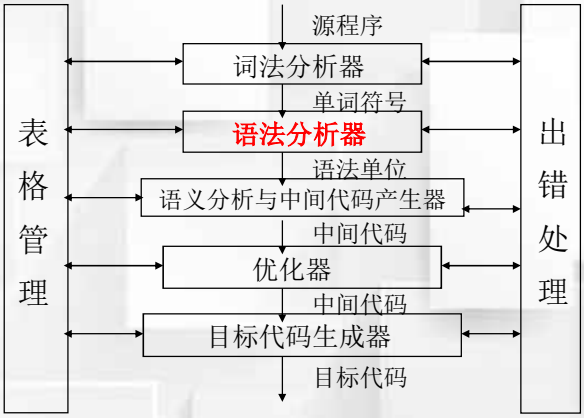


图1.1 编译程序总框

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

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。

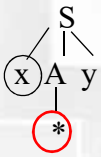
10

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

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

① 构造不带回溯的自上而下分析算法

② 要消除文法的左递归性

③ 克服回溯

4.3 LL(1)分析法

- 4.3.1 左递归的消除 Elimination of Left Recursion
- 4.3.2 消除回溯、提左因子 Left Factoring
- 4.3.3 LL(1)分析条件 LL(1) Grammars

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

4.3.1 左递归的消除

- 左递归的消除 Elimination of Left Recursion

S→Aα | b

A→Ac | Sd | ε

$S \Rightarrow A\alpha \Rightarrow Sd\alpha$

S→Aα |b　　P→Pα | β

S→βS′　　P→βP′

S′→αS′|ε　　P′→αP′|ε

4.3.1 左递归的消除

- 左递归的消除 Elimination of Left Recursion

S→Aα | b

A→Ac | Sd | ε

$A \rightarrow Ac \mid A\alpha d \mid bd \mid \varepsilon$

S→Aα | b

A→bdA′ | A′

A′→cA′| α dA′ | ε

输入：语法G，没有循环和ε

输出：等价语法G’，没有左递归

4.3.1 左递归的消除

- 左递归的消除 Elimination of Left Recursion

$S \to A\alpha \mid b$

$A \to Ac \mid Sd \mid \varepsilon$

=

$S \to A\alpha \mid b$

$A \to bdA' \mid A'$

$A' \to cA' \mid adA' \mid \varepsilon$

方法：第一步，把非终结点以任意方式排序 $A_1, A_2, \ldots, A_n$ (例如上述语法，排序为S, A)

4.3.1 左递归的消除

- 左递归的消除 Elimination of Left Recursion

$S \to A\alpha \mid b$

$A \to Ac \mid Sd \mid \varepsilon$

=

$S \to A\alpha \mid b$

$A \to bdA' \mid A'$

$A' \to cA' \mid adA' \mid \varepsilon$

方法：第二步，循环替换，把当前非终结点i之前的非终结点1~i的产生式带入当前非终结点i
(例如上述语法，排序(S, A)，S之前没有其他非终结点，A之前的终结点为S，则把S的产生式带入A)

$A \to Ac \mid Aad \mid bd \mid \varepsilon$

4.3.1 左递归的消除

- 左递归的消除 Elimination of Left Recursion

S→Aα | b

A→Ac | Sd | ε

S→Aα | b

A→bdA′ | A′

A′→cA′| adA′ | ε

方法：第三步，循环消除当前非终结点的左递归(例如上述语法，排序(S, A)，把S的产生式带入A以后，消除A的左递归)

A → Ac | Aad | bd | ε

α β γ

4.3.1 左递归的消除

- 左递归的消除 Elimination of Left Recursion

（1）把文法G的所有非终结符按任一种顺序排列成P_1，P_2，…，P_n；按此顺序执行；

（2）*FOR i:=1 TO n DO*

BEGIN

FOR j:=1 TO i-1 DO

把形如$P_i \to P_j\gamma$的产生式改写成

$P_i \to \delta_1\gamma|\delta_2\gamma|...|\delta_k\gamma$；

(其中$P_j \to \delta_1|\delta_2|...|\delta_k$是$P_j$的产生式)

消除P_i产生式的直接左递归性

END

（3）化简由（2）所得的文法。去除那些从开始符号出发永远无法到达的非终结符的产生式。

4.3.1 左递归的消除

- 左递归的消除 Elimination of Left Recursion
- 例 消除文法G(E)的左递归:

E→E＋T | T
T→T*F | F
F→(E) | i

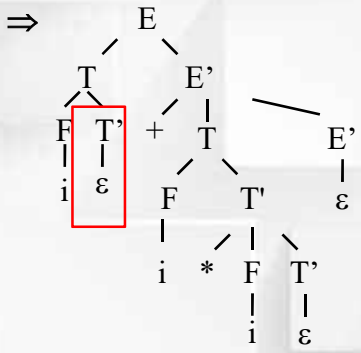
经消去直接左递归后变成：

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

25

例如：输入字符串为i + i * i, 文法

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



为什么要选T’ → ε，
而不是选T’ →*FT’ ？

26

4.3.2 消除回溯、提左因子

4.3.2 消除回溯、提左因子

- 什么是回溯?
  分析工作要部分地或全部地退回去重做叫回溯。

- 造成回溯的条件：
  文法中，对于某个非终结符号的产生式右部有多个选择，并根据所面临的输入符号不能准确地确定所要的选择时，就可能出现回溯。

4.3.2 消除回溯、提左因子

- 回溯带来的问题
  严重的低效率，只有在理论上的意义而无实际意义。

例 假定有文法G(S):
(1) S→xAy
(2) A→**|*　　分析输入串x*y

4.3.2 消除回溯、提左因子

- FIRST 和 FOLLOW
  实际中建立语法树是基于两个功能，FIRST和FOLLOW，并结合语法G。

  通过检验下一个输入字符，在FIRST 和 FOLLOW中的情况，可以帮助决定下一个产生式的选择，

4.3.2 消除回溯、提左因子

- FIRST 和 FOLLOW

  FIRST(α)是一组终结符， α是任意的语法字符串，
  FIRST(α)是α的产生式中的首个字符。

  例如 $\alpha \overset{*}{\Rightarrow} \varepsilon$ FIRST(α) = {ε}

  $A \overset{*}{\Rightarrow} c\gamma$ *(γ是任意语法字符串)* FIRST(A)={c}

31

4.3.2 消除回溯、提左因子

- FIRST 和 FOLLOW

  FIRST(α)的用法示例

  例如 $A \rightarrow \alpha \mid \beta$ FIRST(A) = {a}

  如果下一个输入字符是a，则选择$A \rightarrow \alpha \mid \beta$

32

4.3.2 消除回溯、提左因子

- FIRST 和 FOLLOW

FOLLOW(A)是一组终结符， A是非终结符，FOLLOW(A)是出现在A右边的一组终结符。

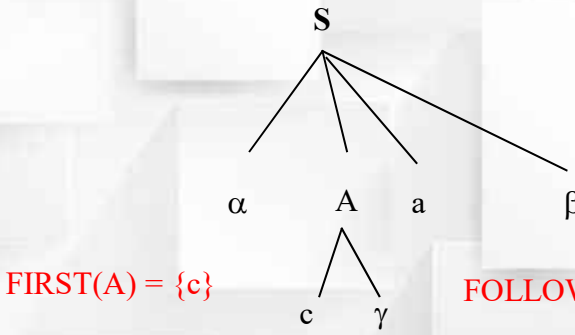
例如 $S \overset{*}{\Rightarrow} \alpha Aa\beta$ FOLLOW(A) = {a}

S如果是开始符号，把#放入FOLLOW(S)

如果A是语法G中某些句型最右边的字符串，则把{#}放入FOLLOW(A)中，#是读入字符时设定的**最后字符，但不是语法中的符号。**

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

- FIRST 和 FOLLOW

有语法G
E→TE′
E′→+TE′ | ε
T→FT′
T′→*FT′ | ε
F→(E) | i
求每个非终结符的FIRST()和FOLLOW()。

FIRST(F) = {(, i}
FIRST(T) = FIRST(F)

FIRST(E) = FIRST(T)

FIRST(E’) = {+, ε}
FIRST(T’) = {*, ε}

4.3.2 消除回溯、提左因子

- FIRST 和 FOLLOW

E是开始符号，FOLLOW(E)必须包含{#}

有语法G
E→TE′
E′→+TE′ | ε
T→FT′
T′→*FT′ | ε
F→(E) | i
求每个非终结符的FIRST()和FOLLOW()。

FOLLOW(E) = {) , # }
FOLLOW(E’) = FOLLOW(E)

FOLLOW(T) = FIRST(E’) ={+}
FOLLOW(T) = FOLLOW(E) = {) , # }
FOLLOW(T) = FOLLOW(E) + FIRST(E’)
= {+,), #}

FOLLOW(T’) = FOLLOW(T)

FOLLOW(F) = FIRST(T’) + FOLLOW(T)
= { * , +,), # }

4.3.2 消除回溯、提左因子

- FIRST 和 FOLLOW

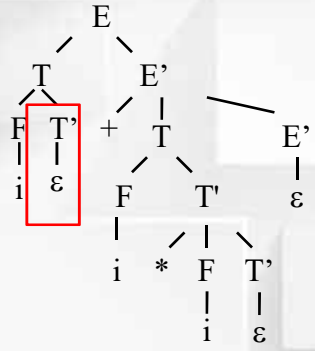
有语法G
E→TE′
E′→+TE′ | ε
T→FT′
T′→*FT′ | ε
F→(E) | i

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

例如：输入字符串为i + i * i, 文法

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}	{ * , +,), # }

FIRST(T’)没有+号，
FOLLOW(T’)有+号，所以选ε

4.3.2 消除回溯、提左因子

- 提取左因子

  假如有条件句

  stmt → if *expr* then *stmt* else *stmt*

  | if *expr* then *stmt*

  选择哪个产生式？

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

4.3.2 消除回溯、提左因子

- 提取左因子

有文法

$S \rightarrow i\ E\ t\ S \mid i\ E\ t\ S\ e\ S \mid a$

$E \rightarrow b$

i, t, e = if, then, else
E = *expr*
S = *stmt*

提取左因子

$S \rightarrow i\ E\ t\ S\ S' \mid a$

$S' \rightarrow e\ S \mid \varepsilon$

$E \rightarrow b$

41

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 每次向前搜索一个输入字符

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

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

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

51

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。

52

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。

53