编译原理

--自底向上的语法分析II

陈俊洁

天津大学智能与计算学部

Outline

- 自底向上的语法分析基本问题
  - 移动－归约分析法
  - 用栈实现移动归约分析
- 算符优先分析法
  - 算符优先分析法定义、优先分析表的确定
  - 使用算符优先关系进行分析
  - 算符优先分析中的错误恢复
- LR分析法
- 语法分析器的自动产生工具Yacc

算符优先分析法

- 算符优先分析法的关键是定义两个可能相继出现的终结符之间的优先关系，并根据这种优先关系寻找“可规约串”并进行规约（非规范规约）。
- 两个终结符之间的优先关系：
  - **a <b**：a的优先级低于b的优先级
  - **a =b**：a的优先级等于b的优先级
  - **a ·>b**：a的优先级高于b的优先级

注意：算符的优先关系是有序的
- 如果a ‘> b，不能推出b <’ a
- 如果a= ‘b，不能推出b= ‘a
- 如果a ‘> b，有可能b ‘> a
- 如果a ‘> b， b ‘> c，不一定a ‘> c

优先关系矩阵例子

- **E→E+E|E*E|E^E|(E)|i** 的终结符之间的优先关系可用矩阵表示**:**

	+	*	^	i	(	)	#
+	`>	<`	<`	<`	<`	`>	`>
*	`>	`>	<`	<`	<`	`>	`>
^	`>	`>	<`	<`	<`	`>	`>
i	`>	`>	`>			`>	`>
(	<`	<`	<`	<`	<`	=`	
)	`>	`>	`>			`>	`>
#	<`	<`	<`	<`	<`		=`

从表中可以看出**:**

1) 优先级 ^ * + 由高到底
2) * + 左结合, ^ 右结合.
3) 运算对象**i**要先处理.
4) 先括号内后括号外.
5) 对于**#,** 任何终结符**Q**都高于**#.**

算符优先文法定义

- 算符文法的定义：
  - 设有一个文法G，如果G中没有形如A→…BC…的产生式，其中B和C为非终结符，则称G为算符文法(Operator Grammar)也称OG文法.
- 算符优先文法的定义：设文法G是不含任何ε-产生式的算符文法，对任何一对终结符a, b, 我们说：
  - a = ‘b：文法中有形如A→…ab…或者A→…aBb…的产生式
  - a < ‘b：文法中有形如A→…aB...的产生式而$B \overset{+}{\Rightarrow} b\cdots$或$B \overset{+}{\Rightarrow} Cb\cdots$
  - a ’ > b：文法中有形如$A \rightarrow \cdots Bb\cdots$的产生式，而$B \overset{+}{\Rightarrow} \cdots a$或$B \overset{+}{\Rightarrow} \cdots aC$

如果G中的任何终结符对(a, b)至多只满足a= ‘b, a< ‘b, a’ >b 三种关系之一，则称G是一个算符优先文法

算符优先文法例子

- 考虑下述文法，
  其是否为算符优先文法？

(1) $E \longrightarrow T|E+T$

(2) $T \longrightarrow F|T*F$

(3) $F \longrightarrow P|P\hat{}F$

(4) $P \longrightarrow (E) \mid i$

由(4) 可得: （ = ‘ ）

由$E \longrightarrow E+T$及$T \overset{+}{=\!=>} T*F$ 可得: + <’ *

由$T \longrightarrow F|T*F$及$F=>P|P\hat{}F$ 可得: * <' ^

由$E \longrightarrow E+T$及 $E=>E+T$ 可得: + >' +

由$F \longrightarrow P\hat{}F$及$F=>P\hat{}F$ 可得: ^ <' ^

由$P \longrightarrow (E)$ 及

$E=>E+T=>T+T=>T*F+T=>F*F+T$

$=>P\hat{}F*F+T=>i\hat{}F*F+T$

可得: (<' + , (<' * , (<' ^ , (<' i.

同理可得: + >'), * >') , ^ >') , i >').

所以这个文法是算符优先文法.

算符优先文法例子

(1) E ⟶ T|E+T

(2) T ⟶ F|T*F

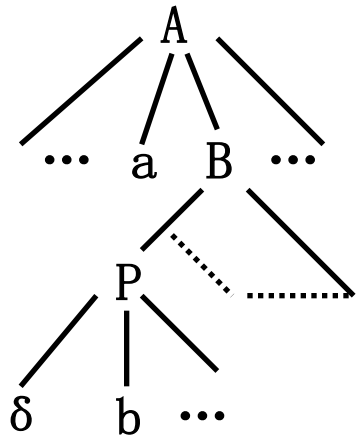
(3) F ⟶ P|P^F

(4) P ⟶ (E) | i

	+	*	^	i	(	)
+	`>	<`	<`	<`	<`	`>
*	`>	`>	<`	<`	<`	`>
^	`>	`>	<`	<`	<`	`>
i	`>	`>	`>			`>
(	<`	<`	<`	<`	<`	=`
)	`>	`>	`>			`>

算符优先关系解释I

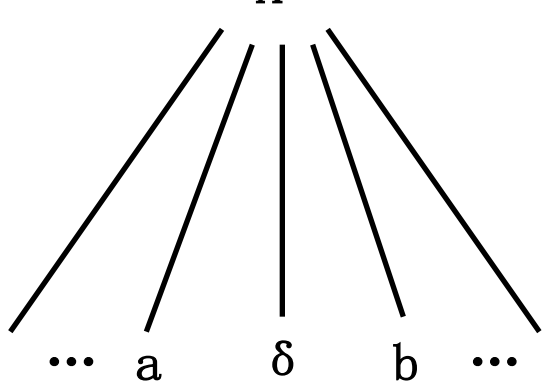
- a < ‘ b，则存在语法子树如下



其中，δ为ε或者C，
a和b不在同一个句柄中，b先被归约

算符优先关系解释II

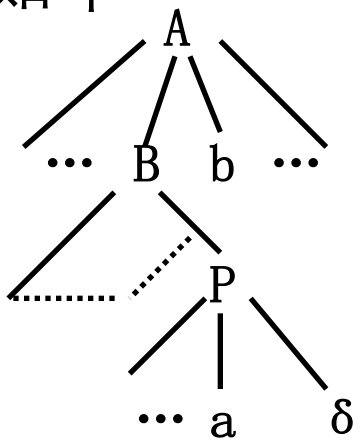
- a = ‘b，则存在语法子树如下



其中，δ为ε或者B，
a和b在同一个句柄中，同时被归约，

算符优先关系解释III

- a >’ b，则存在语法子树如下



其中，δ为ε或者C，
a和b不在同一个句柄中，a先被归约

FIRSTVT & LASTVT

- 构造文法的每个非终结符的首(尾)符号集合:
  - FIRSTVT(P)={a|$P \overset{+}{=}> a\cdots$或$P \overset{+}{=}> Qa\cdots$, $a \in V_T$, P, Q $\in V_N$}
  - LASTVT(P)={a|$P \overset{+}{=}> \cdots a$或$P \overset{+}{=}> \cdots aQ$, $a \in V_T$, P, Q $\in V_N$}
- 有了这两个集合就可以定义：’ > < ‘而= ‘可从文法直接找到.
- 如果形如A→…aB…的产生式,且任何b∈FIRSTVT(B), 满足a < ‘ b
  如果形如A→…Bb…的产生式,且任何a∈LASTVT(B), 满足a ‘> b

构造集合FIRSTVT(P)的算法I:

用下面两条规则构造集合FIRSTVT(P) :

a) **若有产生式P→a…或P→Qa…, 则a∈FIRSTVT(P).**

b) **若a∈FIRSTVT(Q), 且有产生式P→Q…, 则a∈FIRSTVT(P).**

建立一个布尔数组F[X, a], 使得F[X, a]为真的条件是，当且仅当a∈FIRSTVT(X). 按上述a）规则对每个数组元素F[X, a]赋初值。我们用一个stack把所有初值为真的数组元素F[X, a]的符号对（X, a）全部放到stack中，然后对stack进行如下运算：

- 如果stack不空，则将栈顶逐出，记此项为(Q, a)对于每个形如P→Q…的产生式，若F[P, a]为假，则变其值为真且将(P, a)推进stack.
- 重复上述过程，直到stack为空。

构造集合FIRSTVT(P)的算法II:

```
begin
    for 每个非终结符X和终结符a  do
            F[X,a]:=FALSE;
    for  每个形如P→a…或P→Qa…的产生式 do
            insert(P,a);
    while  stack 非空     do
     begin
      把stack的栈顶项,记为 (Q,a), pop出去;
      for  每条形如P→Q…的产生式  do
            insert(P,a);
     end of while;
end
```

下面是求F[P, a]的值的算法:

```
Proc insert(P, a);
   if not    F[P, a]    then
   begin
      F[P, a]:=TRUE;
      把(P,a)推进栈
   end;
```

FIRSTVT(P)={a | F[P,a] = TRUE }

构造优先表的算法

for　每条产生式 $P \longrightarrow X_1X_2\ldots X_n$ do
for i:=1 to n-1 do
begin
if Xi和Xi+1均为终结符 then 置 Xi=' Xi+1
if i<= n-2 且 Xi和Xi+2都为终结符
但Xi+1为非终结符 then 置 Xi=' Xi+2;
if Xi 为终结符而Xi+1 为非终结符 then
for FIRSTVT(Xi+1)中的每个a do
置Xi<' a
if Xi 为非终结符而X_{i+1} 为终结符 then
for LASTVT(X_i)中的每个a do
置 a '>X_{i+1}
end

算符优先分析—基本定义

- 素短语:是指这样的一个短语,它至少含有一个终结符,并且除它自身之外不再含任何更小的素短语。
- 最左素短语:处于句型最左边的那个素短语。
- 算符优先分析法的 “可归约串” 是最左素短语。

例如: 考虑下述文法的句型**P*P+i**的素短语.

(1) E⟶T|E+T

(2) T⟶F|T*F

(3) F⟶P|P^F

(4) P⟶(E) | i

短语是**P*P+i　P*P**和**i**

素短语是**P*P**和**i**

最左素短语是**P*P**

算符优先文法

- 一个算符优先文法的句型（包含在两个#之间）具有以下形式：
  $\#N_1a_1N_2a_2 \ldots N_na_n\ N_{n+1}\#$　　(*)
  - **其中a_i是终结符，N_i是可有可无的非终结符**
- 任何算符文法的句型都具有如上形式
- 一个算符优先文法G的任何句型（*）的最左素短语是满足如下条件的最左子串 $N_ja_j\cdots N_ia_iN_{i+1}$，
  - $a_{j-1}<\ `a_j$
  - $a_j=\ `a_{j+1},\cdots, a_{i-1}=`\ a_i$
  - $a_i\ `>a_{i+1}$

使用算符优先关系

- 算符优先关系主要用于界定句型的最左素短语
  - < ‘标记最左素短语的左端；=’ 出现在最左素短语的内部； ‘>标记最左素短语的右端。
  - 发现最左素短语的过程：
    - 从左端开始扫描串，直到遇到第一个 ‘>为止。
    - 向左扫描栈，跳过所有的= ‘，直到遇到一个< ‘为止。
    - 最左素短语包括从上一步遇到的< ‘右部到第一个 ‘>左部之间的所有符号，包括介于中间或者两边的非终结符
- 非终结符的处理
  - 因为非终结符不能影响语法分析，所以不需要区分它们，于是只用一个占位符来代替它们

算符优先分析算法

- 算法的主体思想
  - 用栈存储已经看到的输入符号，用优先关系指导移动-归约语法分析器的动作
    - 如果栈顶的终结符和下一个输入符之间的优先关系是< ‘或= ‘（即**当前输入符号**为最左素短语的左，中端），则进行移动；
    - 如果是’ >关系（即**当前输入符号**为最左素短语的最右端），则进行归约
- 算法描述
  - 输入：输入字符串w和优先关系表
  - 输出：如果w是语法产生的一个句子，则输出其用来归约的产生式；如果有错误，则转入错误处理

算符优先分析算法

- 最外层循环(2-17)：读入输入串w
- 内层While循环(5-13)：寻找最左素短语(6-9)进行规约，直到在当前栈内无法找到新的最左素短语为止

注意：第10行中，规约为某个N，这个N指的是这样一个产生式的左端：此产生式的右部和S[j+1]…S[k]构成如下一一对应关系：自左至右，终结符对终结符，非终结符对非终结符，而且对应的终结符相同。

```
1.    K:=1; S[k]:= '#' ;
2.    Repeat
3.       把下一输入符号读入a；
4.       If S[k]∈V_T Then j:=k ELSE j:=k-1;
5.       While S[j] ' > a Do
6.          Repeat
7.             Q:=S[j];
8.             If S[j-1] ∈V_T Then j:=j-1 Else j:=j-2;
9.          Until S[j] < ' Q;
10.         把 S[j+1]...S[k]归约为某个N；
11.         k:=j+1;
12.         S[k]=N;
13.      EndOfWhile
14.      If S[j] < ' a Or S[j]= ' a Then
15.         begin k:=k+1; S[k]:=a; end
16.      Else error
17.   Until a= '#'
```

规范归约和算符优先归约

- 句型i+i#的规范归约过程

E' => #E => #E+T => #E+F
=> #E+P => #E+i => #T+i
=> #F+i => #P+i => #i+i

步骤	符号栈	剩余输入串	句柄	归约用产生式
1	#	i+i#		
2	#i	+i#	i	P→i
3	#P	+i#	P	F→P
4	#F	+i#	F	T→F
5	#T	+i#	T	E→T
6	#E	+i#		
7	#E+	i#		
8	#E+i	#	i	P→i
9	#E+P	#	P	F→P
10	#E+F	#	F	T→F
11	#E+T	#	E+T	E→E+T
12	#E	#		接受

(1) $E \longrightarrow T|E+T$

(2) $T \longrightarrow F|T*F$

(3) $F \longrightarrow P|P\hat{}F$

(4) $P \longrightarrow (E) \mid i$

规范归约和算符优先归约

- 句型 i+i#的算符优先归约过程

	+	i	#
+	‘>’	‘<’	‘>’
i	‘>’		‘>’
#	‘<’	‘<’	‘=’

步骤	栈	优先关系	当前符号	剩余符号串	动作
1	#	‘<’	i	+i#	移进
2	#i	‘>’	+	i#	归约
3	#P	‘<’	+	i#	移进
3	#P+	‘<’	i	#	移进
4	#P+i	‘>’	#		归约
4	#P+P	‘>’	#		归约
4	#E	‘=’	#		接受

(1) E → T|E+T

(2) T → F|T*F

(3) F → P|P^F

(4) P → (E) | i

算符优先分析法优缺点

- 由于算符优先分析并未对文法的非终结符定义优先关系，所以就无法发现由单个非终结符组成的“可归约串”。
- 也就是说，在算符优先归约过程中，我们无法用那些右部仅含一个非终结符的产生式（称为单非产生式，如P→Q）进行归约。这样，算符优先分析比规范归约要快很多。这既是算符优先分析的优点，也是它的缺点。
- 忽略非终结符在归约过程中的作用，存在某种危险性，可能导致把本来不成句子的输入串误认为是句子。

自学

优先函数

- 优先函数:
  - 在实际实现算法分析时;用两个优先函 数**f, g,**把终结符**Q**与两个自然数**f(Q)**和 **g(Q)**相对应.
- 使得:
  - 若 Q1< ‘Q2 则 f(Q1) < g(Q2)
  - 若 Q1= ‘Q2 则 f(Q1) = g(Q2)
  - 若 Q1 ‘>Q2 则 f(Q1) > g(Q2)
- 函数**f**称为入栈优先函数. 函数**g**称为比较优先函数.

自学

例子

(1) E⟶T|E+T

(2) T⟶F|T*F

(3) F⟶P|P^F

(4) P⟶(E) | i

	+	*	^	i	(	)	#
+	`>	<`	<`	<`	<`	`>	`>
*	`>	`>	<`	<`	<`	`>	`>
^	`>	`>	<`	<`	<`	`>	`>
i	`>	`>	`>			`>	`>
(	<`	<`	<`	<`	<`	=`	
)	`>	`>	`>			`>	`>
#	<`	<`	<`	<`	<`		=`

	+	*	^	(	)	i	#
f	2	4	5	0	6	6	0
g	1	3	6	7	0	7	0

自学

优先函数

- 优先函数的问题
  - 优先关系表中的空白项是模糊的
  - 每个终结符都对应一对优先函数，而数是可以比较的，所以容易掩盖错误
    不便区分一目运算 “-” “+” 和二目运算 “-” “+”.

	a	b
a	= ‘	‘>
b	= ‘	= ‘

f(a)=g(a), f(a)>g(b), f(b)=g(a), f(b)=g(b)

f(a)>g(b)=f(b)=g(a)=f(a)

- 优先关系表的存储方式
  - 使得矩阵表示：准确直观；需要大量内存空间$(n+1)^2$
  - 优先函数：需要内存空间小$2(n+1)$；不利于出错处理

自学

优先函数的构造算法

- 输入：算符优先表
- 输出：表示输入矩阵的优先函数或指出它不存在
- 方法：
  - 为每个终结符a（包括#）创建f_a和g_a。并以其作为结点，画出2n个结点
  - 如果a ‘> b或者a =’ b，则画一条从f_a到g_b的箭弧
  - 如果a < ‘ b或者a =’ b，则画一条从g_b到f_a的箭弧
  - 如果构造的图中有环路，则不存在优先函数；如果没有环路，则将f(a)设为从f_a出发所能到达的结点个数（包括初始结点）；g(a)为从g_a出发所能到达的结点个数（包括初始结点）。
  - 检查与原算符优先表是否一致！

自学

构造优先函数-例子

	i	*	+	#
i		‘>	‘>	‘>
*	<‘	‘>	‘>	‘>
+	<‘	<‘	‘>	‘>
#	<‘	<‘	<‘	=‘

	i	*	+	#
f	6	6	4	2
g	7	5	3	2

检查是否一致！

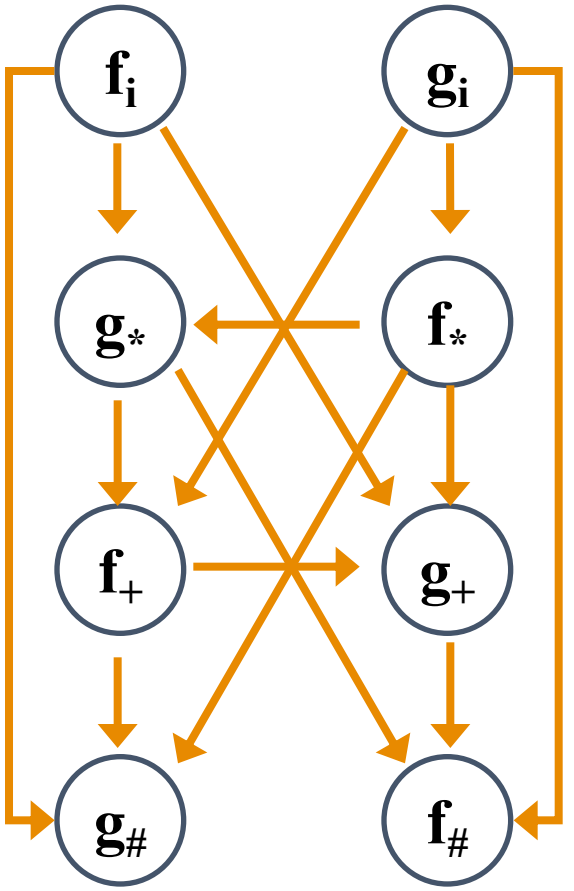
自学

构造优先函数-例子

	i	*	+	#
i		>	>	>
*	<	>	>	>
+	<	<	>	>
#	<	<	<	=

	i	*	+	#
f	4	4	2	0
g	5	3	1	0

检查是否一致！



算符优先分析中的错误恢复

- 算符优先分析器能发现的语法错误
  - 如果栈顶的终结符和当前输入之间没有优先关系（如果用优先函数存储，这个错误不能发现）
  - 如果发现最左素短语，但最左素短语不是任何产生式的右部
- 归约时的错误处理
  - 给出相应的具有描述性的出错信息
  - 试图通过插入、删除来获得句柄

一个加入了出错处理的优先矩阵

- E1：缺少整个表达式
  - 把id插入输入字符串中
  - 输出诊断信息
    - Missing operand
- E2：表达式以右括号开始
  - 从输入中删除)
  - 输出诊断信息
    - Unbalanced right parenthesis

E3：id/)后面跟id/(
把+插入输入字符串
输出诊断信息
Missing operator

E4：表达式以左括号结束
从栈中弹出(
输出诊断信息
Missing right parenthesis

	id	(	)	#
id	**E3**	**E3**	>	>
(	<	<	=	**E4**
)	**E3**	**E3**	>	>
#	<	<	**E2**	**E1**