第六章
自底向上的优先分析法

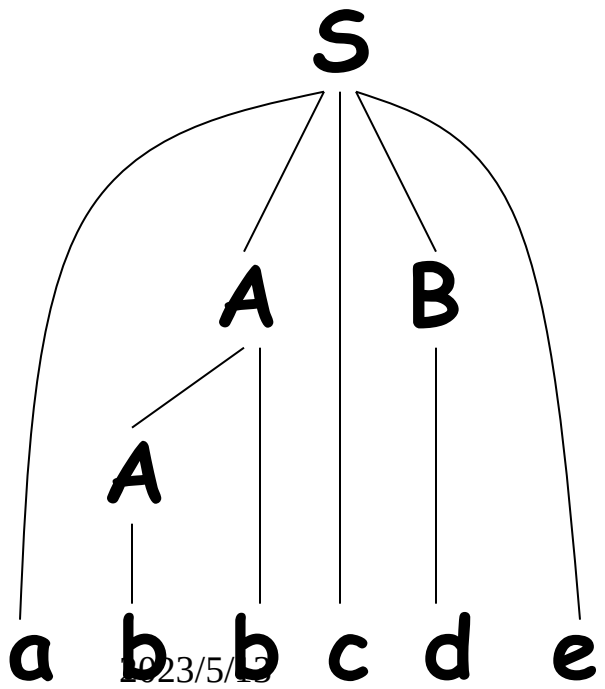
- 自底向上语法分析概述
- 简单优先分析
- 算符优先分析

6.1自底向上语法分析概述

- 自底向上语法分析试图将一个字符串归约至开始符号。
- 自下而上语法分析比自顶向下语法分析更有效率，对语法的限制更少
- “移进-归约”：从输入字符串开始，逐步进行归约直到归约到文法的开始符号。

分析符号串abbcde是否G[S]的句子

文法G[S]:
(1) S → aAcBe
(2) A → b
(3) A → Ab
(4) B → d



步骤	符号栈	输入符号串	动作
1）	#	abbcde#	移进
2）	#a	bbcde#	移进
3）	#ab	bcde#	归约(A→b)
4）	#aA	bcde#	移进
5）	#aAb	cde#	归约(A→Ab)
6）	#aA	cde#	移进
7）	#aAc	de#	移进
8）	# aAcd	e#	归约(B→d)
9）	#aAcB	e#	移进
10）	#aAcBe	#	归约(S→aAcBe)
11）	#S	#	接受

对输入串abbcde#的移进-规约分析过程

- 归约过程恰好是最右推导的逆过程：

S⇒aAcBe⇒aAcde⇒aAbcde⇒abbcde

- 规范归约定义：

句柄

- 假定α是文法G的一个句子，我们称序列$\alpha_n, \alpha_{n-1}, \ldots \alpha_0$是α的一个规范归约。如果此序列满足：

  1、 $\alpha_n = \alpha$

  2、 α_0为开始符号。

  3、对任何i,0<i<=n, α_{i-1}是从α_i经把句柄替换为相应产生式的左部符号而得到的。

- 规范归约也称最左归约，最右推导称为规范推导。规范推导得到的句型成为规范句型。
- 如果文法G无二义，则规范推导的逆过程一定是规范归约。

“移进-归约”法的栈实现

自顶向下：初始：分析栈：#S　输入串:a1a2…an#

结束：　　#　　#（成功）

分析过程：用产生式的右部替换左部。

自底向上：初始：分析栈：#　输入串:a1a2…an#

结束：　　#S　　#（成功）

分析过程：自左至右把输入符号串W的符号一一移进栈里，一旦发现栈顶的一部分符号形成一个可归约串，就把栈中这个子串用相应的归约符号替换。

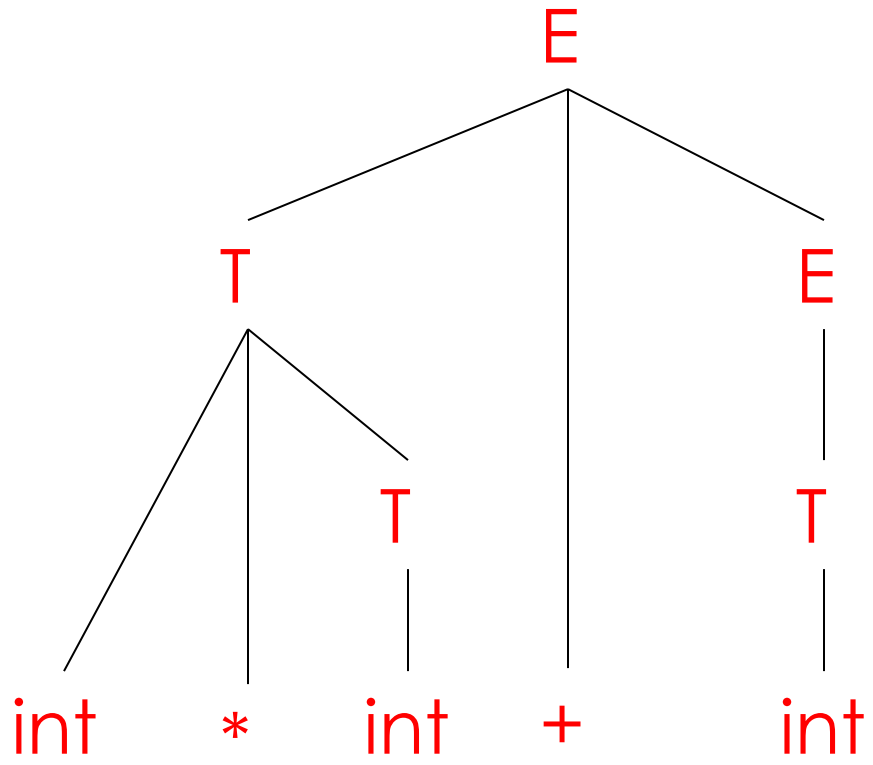
四类操作：移进，归约，接受，出错处理。

缺点

文法G[E]:

$E \rightarrow T + E \mid T$

$T \rightarrow int * T \mid int \mid (E)$



|int * int + int	移进
int | * int + int	移进
int * | int + int	移进
int * int | + int	规约 $T \rightarrow int$
int * T | + int	规约$T \rightarrow int * T$
T | + int	移进
T + | int	移进
T + int |	规约$T \rightarrow int$
T + T |	规约$E \rightarrow T$
T + E |	规约$E \rightarrow T + E$
E |	

A Shift-Reduce Parse in Detail (1)

|int * int + int

int * int + int

↑

A Shift-Reduce Parse in Detail (2)

|int * int + int

int | * int + int

A Shift-Reduce Parse in Detail (3)

|int * int + int

int | * int + int

int * | int + int

A Shift-Reduce Parse in Detail (4)

|int * int + int

int | * int + int

int * | int + int

int * int | + int

A Shift-Reduce Parse in Detail (5)

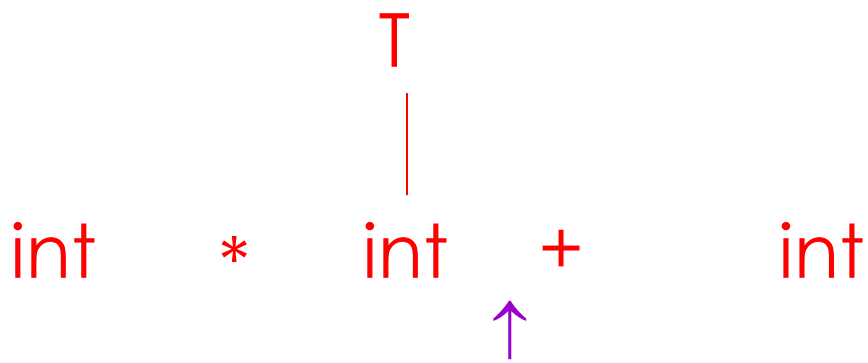
|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int



A Shift-Reduce Parse in Detail (6)

|int * int + int

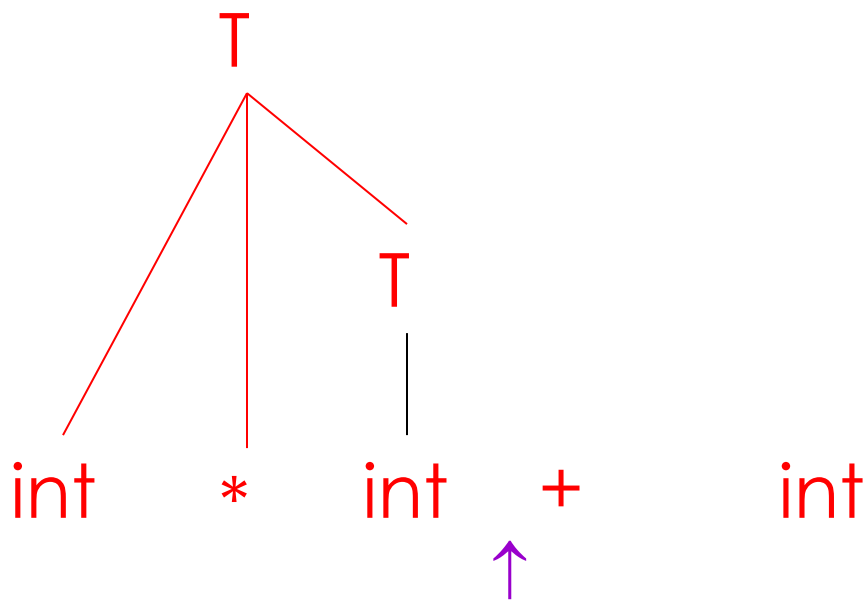
int | * int + int

int * | int + int

int * int | + int

int * T | + int

T | + int



A Shift-Reduce Parse in Detail (7)

|int * int + int

int | * int + int

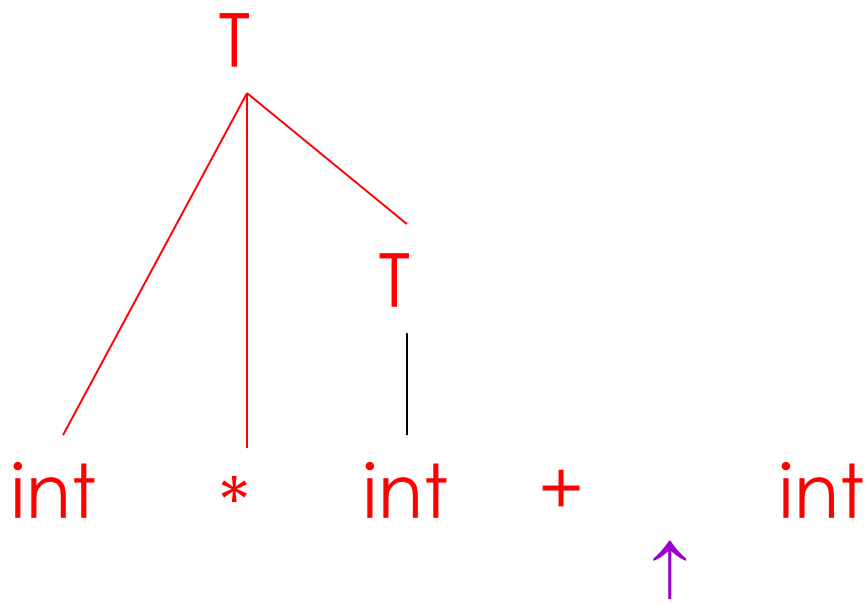
int * | int + int

int * int | + int

int * T | + int

T | + int

T + | int



A Shift-Reduce Parse in Detail (8)

|int * int + int

int | * int + int

int * | int + int

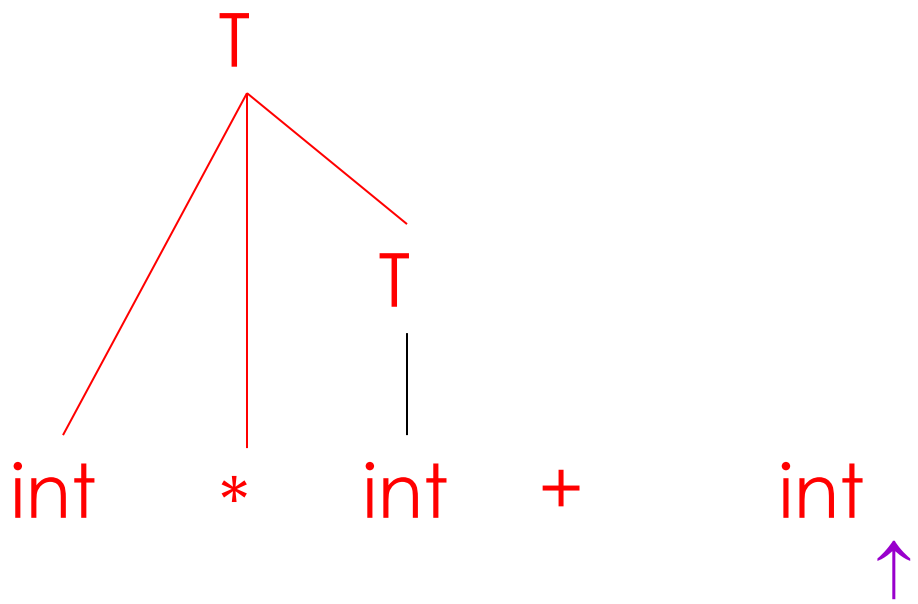
int * int | + int

int * T | + int

T | + int

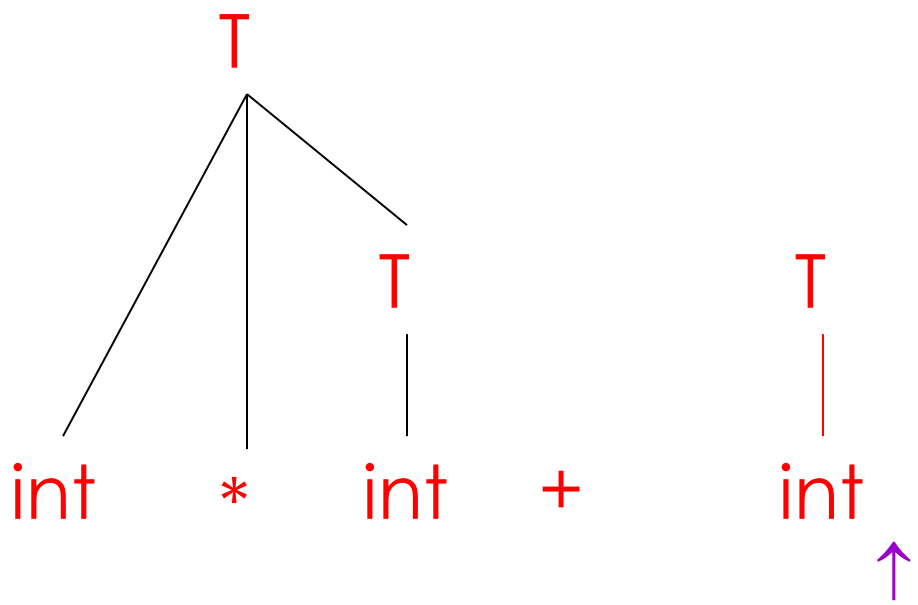
T + | int

T + int |



A Shift-Reduce Parse in Detail (9)

|int * int + int
int | * int + int
int * | int + int
int * int | + int
int * T | + int
T | + int
T + | int
T + int |
T + T |



A Shift-Reduce Parse in Detail (10)

|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int

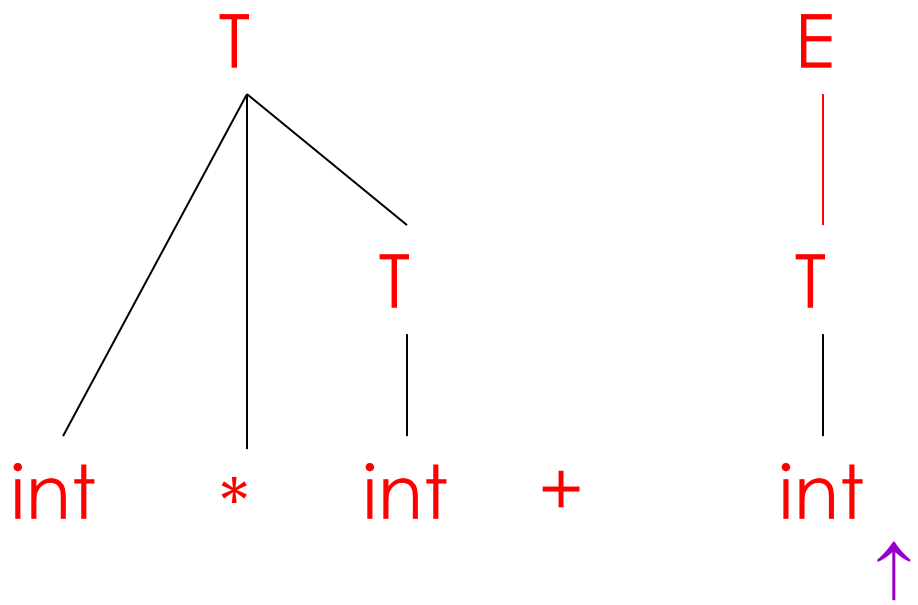
T | + int

T + | int

T + int |

T + T |

T + E |



A Shift-Reduce Parse in Detail (11)

|int * int + int

int | * int + int

int * | int + int

int * int | + int

int * T | + int

T | + int

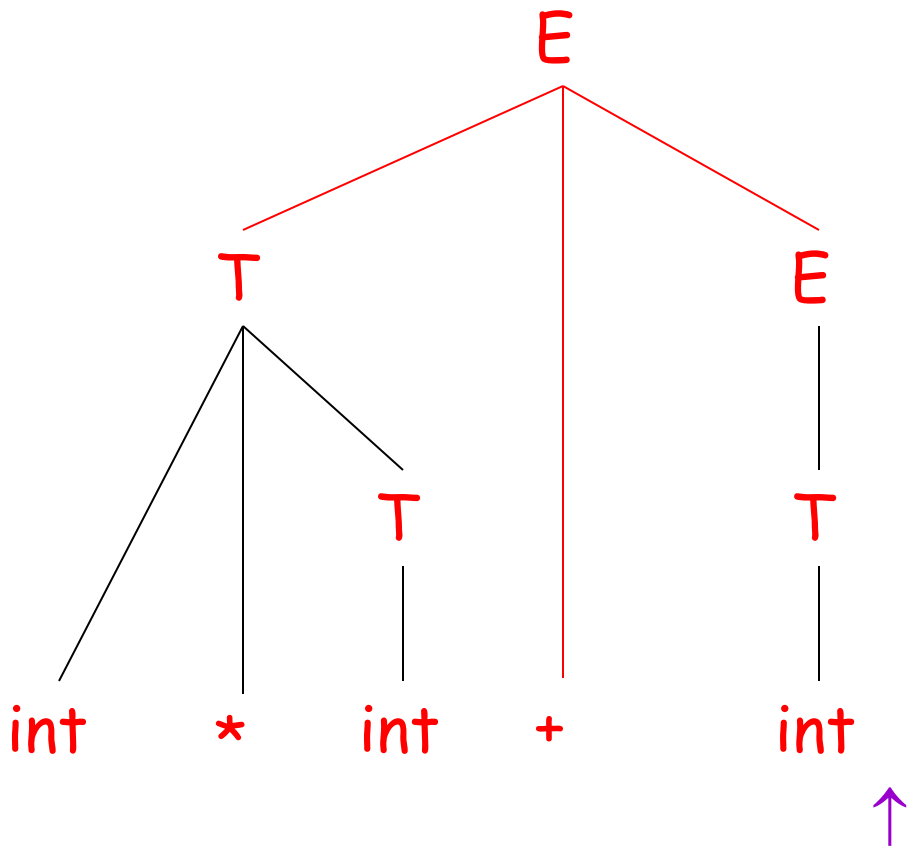
T + | int

T + int |

T + T |

T + E |

E |



6.2自底向上的优先分析算法

- 优先分析法
  - 简单优先分析法
  - 算符优先分析法

简单优先分析法

- 按照文法符号（包括终结符和非终结符）的优先关系确定句柄。
- 示例见下页

文法G[S]:

(1) S → bAb

(2) A → (B|a

(3) B → Aa)

	S	b	A	(	B	a	)	#
S								>
b			=	<		<		>
A		=				=		
(			<	<	=	<		
B		>				>		
a		>				>	=	
)		>				>		
#	<	<						

简单优先关系矩阵

步骤	符号栈	输入符号串	动作
1）	#	b(aa)b#	#<b,移进
2）	#b	(aa)b#	b<(,移进
3）	#b(	aa)b#	(<a,移进
4）	#b(a	a)b#	a>a,归约A→a
5）	#b(A	a)b#	A=a,移进
6）	#b(Aa	)b#	a=),移进
7）	#b(Aa)	b#	)>b,归约B→Aa)
8）	#b(B	b#	B>b,归约A→(B
9）	#bA	b#	A=b,移进
10）	#bAb	#	b>#,归约S→bAb
11）	#S	#	接受

对输入串b(aa)#的简单优先分析过程

优先关系

- 优先关系
  - X=Y ⇔ 文法G中存在产生式A→...XY...
  - X<Y⇔ 文法G中存在产生式A→...XB...，且B $\overset{+}{\Rightarrow}$Y...
  - X>Y⇔ 文法G中存在产生式A→...BD...，且B $\overset{+}{\Rightarrow}$...X,D $\overset{*}{\Rightarrow}$Y...
- #的优先级<所有符号，所有符号的优先级>#，这里仅对与#相邻的文法符号而言。

简单优先文法的定义

满足以下条件的文法是简单优先文法

（1）在文法符号集V中，任意两个符号之间最多只有一种优先关系成立。

（2）在文法中任意两个产生式没有相同的右部。

简单优先分析法的算法步骤

- 将输入符号串$a_1a_2\ldots a_n$#依次逐个存入符号栈S中，直到遇到栈顶符号a_i的优先性>下一个待输入符号a_j为止。
- 栈顶当前符号a_i为句柄尾，由此向左在栈中找句柄的头符号a_k，即找到$a_{k-1}<a_k$为止。
- 由句柄$a_k\ldots a_i$在文法的产生式中查找右部为$a_k\ldots a_i$的产生式，若找到则用相应左部代替句柄，若找不到则为出错。
- 重复1，2，3步，直到栈中只剩开始符。

算符优先分析

- 某些文法具有“算符”特性
  - 表达式运算符（优先级、结合性）
  - 人为地规定其算符的优先顺序，即给出优先级别和同一级别的结合性
- 只考虑算符之间的优先关系

文法G[E]：E→E+E|E-E|E*E|E/E|E↑E|(E)|i

	+	-	*	/	↑	(	)	i	#
+	>	>	<	<	<	<	>	<	>
-	>	>	<	<	<	<	>	<	>
*	>	>	>	>	<	<	>	<	>
/	>	>	>	>	<	<	>	<	>
↑	>	>	>	>	<	<	>	<	>
(	<	<	<	<	<	<	=	<	
)	>	>	>	>	>		>		>
i	>	>	>	>	>		>		>
#	<	<	<	<	<	<		<	=

算符优先关系表

步骤	符号栈	输入符号串	动作
1）	#	i+i*i#	#<i,移进
2）	#i	+i*i#	#<i>+,规约
3）	#E	+i*i#	#<+,移进
4）	#E+	i*i#	+<i,移进
5）	#E+i	*i#	+<i>*,规约
6）	#E+E	*i#	+<*,移进
7）	#E+E*	i#	*<i,移进
8）	#E+E*i	#	*<i>#,规约
9）	#E+E*E	#	+<*>#,规约
10）	#E+E	#	#<+>#,规约
11）	#E	#	接受

对输入串i+i*i的算符优先分析过程

归约成功标志：栈中只剩#N，输入符号串剩#

如何确定算符优先关系？

文法G[E]：E→E+E|E-E|E*E|E/E|E↑E|(E)|i

(1) i的优先级最高

(1) ↑优先级次于i，右结合

(2) *和/优先级次之，左结合

(3) +和-优先级最低，左结合

(4) 括号‘(’,‘)’的优先级大于括号外的运算符，小于括号内的运算符，内括号的优先性大于外括号

(5) #的优先性低于与其相邻的算符

	+	-	*	/	↑	(	)	i	#
+	>	>	<	<	<	<	>	<	>
-	>	>	<	<	<	<	>	<	>
*	>	>	>	>	<	<	>	<	>
/	>	>	>	>	<	<	>	<	>
↑	>	>	>	>	<	<	>	<	>
(	<	<	<	<	<	<	=	<	
)	>	>	>	>	>		>		>
i	>	>	>	>	>		>		>
#	<	<	<	<	<	<		<	=

算符优先关系表

算符文法的定义

- 定义　　如果不含空产生式的上下文无关文法 G 中没有形如 U→…VW…的产生式，其中V,W$\in V_N$则称G 为算符文法（OG）。
- 性质1：在算符文法中任何句型都不包含两个相邻的非终结符.(数学归纳法)
- 性质2：如 Vx 或 xV 出现在算符文法的句型 α 中，其中V$\in V_N$,x$\in V_T$，则 α 中任何含 x 的短语必含有V.（反证法）

算符优先关系的定义

- 在OG中 定义 （算符优先关系）
  - x = y　　G中有形如.U→…xy…或U → …xVy.. 的产生式。
  - x < y　　G中有形如.U →…xW…的产生式,而 $W \overset{+}{\Rightarrow} y\ldots$.或$W \overset{+}{\Rightarrow} Vy\ldots$
  - x>y　　G中有形如.U → …Wy…的产生式,而 $W \overset{+}{\Rightarrow} \ldots x$或$W \overset{+}{\Rightarrow} \ldots xV$

- 规定 若 $S \overset{+}{\Rightarrow} x\ldots$ 或 $S \overset{+}{\Rightarrow} Vx\ldots$ 则 # < x
  $S \overset{+}{\Rightarrow} \ldots x$ 或 $S \overset{+}{\Rightarrow} \ldots xV$ 则 x > #

算符优先文法的定义

- 在 OG文法 G 中，若任意两个终结符间至多有一种算符优先关系存在，则称G为算符优先文法(OPG)。

  - 结论 算符优先文法是无二义的。

算符优先关系表的构造

- 由定义直接构造
- 由关系图法构造算符优先关系表

- 首先引入两个概念
  - **FIRSTVT(B)={b|B$\overset{+}{\Rightarrow}$ b…或B$\overset{+}{\Rightarrow}$ Cb...}**
    - 对于非终结符**B**，其往下推导所可能出现的首个算符
  - **LASTVT(B)={a|B$\overset{+}{\Rightarrow}$… a或B $\overset{+}{\Rightarrow}$.. aC}**
    - 对于非终结符**B**，其往下推导所可能出现的最后一个算符

如何计算算符优先关系

1) '='关系
   - 直接看产生式的右部，若出现了 $A \rightarrow \ldots ab \ldots$ 或 $A \rightarrow \ldots aBb$，则 $a=b$

2) '<'关系
   - 求出每个非终结符B的FIRSTVT(B)
   - 若 $A \rightarrow \ldots aB \ldots$，则 $\forall b \in FIRSTVT(B), a<b$

3) '>'关系
   - 求出每个非终结符B的LASTVT(B)
   - 若 $A \rightarrow \ldots Bb \ldots$，则 $\forall a \in LASTVT(B), a>b$

文法G[E]:
(0) E'→#E#
(1) E→E+T
(2) E→T
(3) T→T*F
(4) T→F
(5) F→P↑F|P
(6) P→(E)
(7) P→i

1)'='关系
由产生式(0)和(6),得
#=#, (=)
2) '<'关系
找形如: A→…aB…的产生式
#E: 则#<FIRSTVT(E)
+T: 则+<FIRSTVT(T)
F: 则<FIRSTVT(F)
↑F:则 ↑<FIRSTVT(F)
(E: 则 (<FIRSTVT(E)

FIRSTVT(E')={#}
FIRSTVT(E)={+,*,↑,(,i}
FIRSTVT(T)={*,↑,(,i}
FIRSTVT(F)={↑,(,i}
FIRSTVT(P)={(,i}
LASTVT(E')={#}
LASTVT(E)={+,*,↑,),i}
LASTVT(T)={*,↑,),i}
LASTVT(F)={↑,),i}
LASTVT(P)={),i}

	+	-	*	/	↑	(	)	i	#
+	>	>	<	<	<	<	>	<	>
-	>	>	<	<	<	<	>	<	>
*	>	>	>	>	<	<	>	<	>
/	>	>	>	>	<	<	>	<	>
↑	>	>	>	>	<	<	>	<	>
(	<	<	<	<	<	<	=	<	
)	>	>	>	>	>		>		>
i	>	>	>	>	>		>		>
#	<	<	<	<	<	<		<	=

3)'>'关系
找形如: A→…Bb…的产生式
E# ,则 LASTVT(E)>#
E+ ,则 LASTVT(E)>+
T* ,则 LASTVT(T)>*
P↑ ,则 LASTVT(P)>↑
E) ,则 LASTVT(E)>)

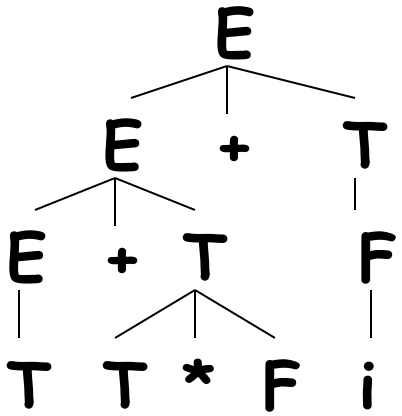
算符优先分析算法

- 归约过程中，只考虑终结符之间的优先关系来确定句柄，而与非终结符无关。这样去掉了对非终结符的归约，所以用算符优先分析法的规约过程与规范归约是不同的，P110.
- 为解决在算符优先分析过程中如何寻找可归约串，引进最左素短语的概念

- 算符文法的任一句型有如下形式：
  $\#N_1a_1N_2a_2\ldots\ldots N_na_nN_{n+1}\#$,若$N_ia_i\ldots\ldots N_ja_jN_{j+1}$为句柄，则有
  $a_{i-1}<a_i=a_{i+1}=\ldots=a_{j-1}=a_j>a_{i+1}$
- 对于算符优先文法，如果aNb(或ab)出现在句型r中
  - 若a<b，则在r中必含有b而不含a的短语存在
  - 若a>b，则在r中必含有a而不含b的短语存在
  - 若a=b，则在r中含有a的短语必含有b，反之亦然
- 定义 cfg G 的句型的素短语是一个短语，它至少包含一个终结符，且除自身外不再包含其他素短语。处于句型最左边的素短语为最左素短语。

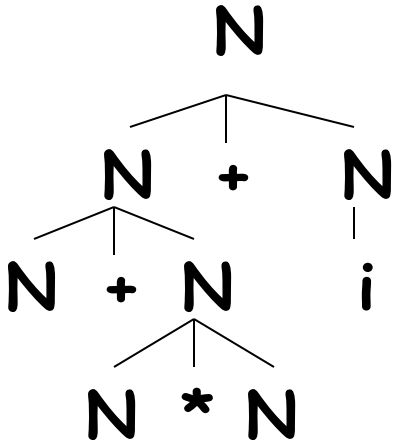
文法G[E]:
(1) E→E+T
(2) E→T
(3) T→T*F
(4) T→F
(5) F→P↑F|P
(6) P→(E)
(7) P→i

句型#T+T*F+i#
其短语有:
T+T*F+i
T+T*F
T
T*F
i



最左素短语为: T*F

句型#N+N*N+i#的归约过程



优先函数

- 优先函数比优先矩阵节省空间
- 优先函数:从终结符号映射到整数的函数。
- 若a<b,则f(a)<g(b)
- 若a=b,则f(a)=g(b)
- 若a>b,则f(a)>g(b)

例：

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

- f(*)<g(i)=>*<i,但f(i)>g(i)=>i>i是不存在的，所以错误检查能力损失。可通过检查栈顶和输入符号a来发现那些不可比较的情形。

构造优先函数

- 1、设a是一个终结符或#，对每一个a建立两个符号f_a和g_a.
- 2、将所有f_a和g_a组成的集合分为若干组，办法是若a=b则f_a和g_b在同一组。
- 3、画图，结点是2建立的组。对任何a和b，若a<b，则从g_b所在的组画一箭弧到f_a所在的组；若a>b，则从f_a 所在的组画一箭弧到g_b所在的组.
- 4、若第3步构造的图中有环路，则没有优先函数。若没有环路，令f(a)是从f_a 所在的组出发的沿箭弧前进的最长路径的长度。 g(a)是从g_a 所在的组出发的沿箭弧前进的最长路径的长度。
- 例

算符优先分析法的局限性

- 很难避免把错误的句子得到正确的归约.