中国科学技术大学
University of Science and Technology of China

语法制导的翻译 II

《编译原理和技术》

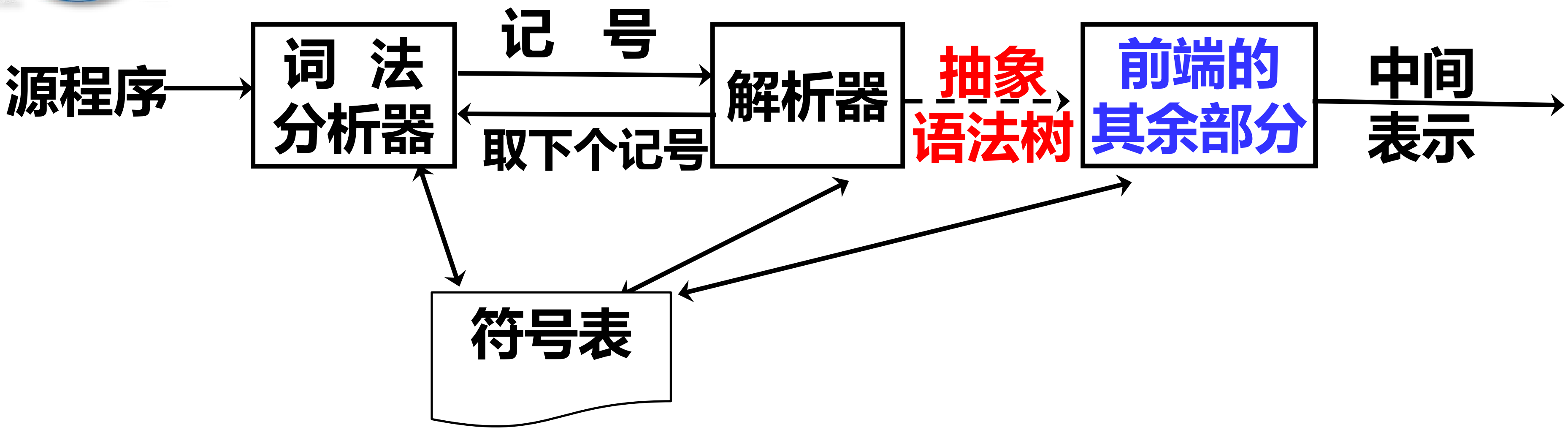
张昱

0551-63603804，yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

本章内容



- 语义的描述：语法制导的定义、翻译方案
  - 语法制导：syntax-directed
    按语法结构来指导语义的定义和计算
  - 抽象语法树、注释分析树等
- 语法制导翻译的实现方法：自上而下、自下而上
  - 边语法分析边翻译

中国科学技术大学
University of Science and Technology of China

4.3 自上而下计算

- 翻译方案
- 预测翻译器的设计
- 用综合属性代替继承属性

翻译方案—内嵌不传播的动作

例 把有加和减的中缀表达式翻译成后缀表达式

如果输入是8+5 –2，则输出是8 5 + 2 –

$E \rightarrow T\ R$

$R \rightarrow \textbf{addop}\ T\ \{print(\textbf{addop}.lexeme)\}\ R_1 \mid \varepsilon$

$T \rightarrow \textbf{num}\ \{print(\textbf{num}.val)\}$

$E \Rightarrow T\ R \Rightarrow \textbf{num}\ \{print\ (8)\}\ R$

$\Rightarrow \textbf{num}\ \{print(8)\}\textbf{addop}\ T\ \{print\ (+)\}R$

$\Rightarrow \textbf{num}\ \{print(8)\}\textbf{addop}\ \textbf{num}\{print(5)\}\{print\ (+)\}R$

$\ldots\ \{print(8)\}\{print(5)\}\{print(+)\}\textbf{addop}\ T\ \{print(-)\}R$

$\ldots\ \{print(8)\}\{print(5)\}\{print(+)\}\{print(2)\}\{print(-)\}$

翻译方案—内嵌有信息传播

$E \rightarrow T \quad \{R.i = T.nptr\}$ $\qquad T + T + T + \ldots$

$\quad R \quad \{E.nptr = R.s\}$

$R \rightarrow +$

$\quad T \quad \{R_1.i = mkNode\ (\text{‘+’}, R.i, T.nptr)\}$

$\quad R_1 \quad \{R.s = R_1.s\}$

$R \rightarrow \varepsilon \quad \{R.s = R.i\ \}$

$T \rightarrow F \quad \{W.i = F.nptr\}$

$\quad W \quad \{T.nptr = W.s\}$

$W \rightarrow *$

$\quad F \quad \{W_1.i = mkNode\ (\text{‘*’}, W.i, F.nptr)\}$

$\quad W_1 \quad \{W.s = W_1.s\}$

$W \rightarrow \varepsilon \quad \{W.s = W.i\ \}$

继承属性的计算嵌在产生式右部的某文法符号之前，表示在分析该文法符号之前计算

F 的产生式部分不再给出

预测翻译器的设计

方法： 将预测分析器的构造方法推广到翻译方案的实现（*LL*文法）

产生式$R \rightarrow +TR \mid \varepsilon$ 的分析过程

```
void R( ) {
    if (lookahead == '+' ) {
      match ( '+' ); T( ); R( );
    }
    else if (lookahead == ')' || lookahead == '$' ) ;
     else error( );
}
```

预测翻译器的设计

```
syntaxTreeNode * R (syntaxTreeNode * i) {
  //继承属性作为参数,综合属性为返回值
   syntaxTreeNode *nptr, *i1, *s1, *s;
   char addoplexeme;

   if (lookahead == '+' ) {
        addoplexeme = lexval;
        match('+' ); nptr = T( );
        i1 = mkNode(addoplexeme, i , np
        s1 = R (i1); s = s1;
   }
   else if (lookahead == ')' || lookahead == '$')  s = i;
   else error( );
   return s;
}
```

```
void R( ) {
    if (lookahead == '+' ) {
    match ( '+' ); T( ); R( );
    }
    else if (lookahead == ')' || lookahead == '$' )  ;
    else error( );
}
```

R : *i*, *s*

T : *nptr*

\+ : *addoplexeme*

非L属性定义

例 Pascal的声明，如m, n : integer

$D \to L : T$ $\quad\quad$ $L.in = T.type$

$T \to \text{integer} \mid \text{char}$ $\quad\quad$ $T.type = \ldots$

$L \to L_1, \text{id} \mid \text{id}$ $\quad\quad$ $L_1.in = L.in, \ldots$

该语法制导定义非*L*属性定义

信息从右向左流，归约从左向右，两者不一致

非L属性定义：改写文法

例 Pascal的声明，如m, n : integer

$D \to L : T$ $\qquad$ $L.in = T.type$ (非L属性定义)

$T \to$ integer | char $\qquad$ $T.type = \dots$

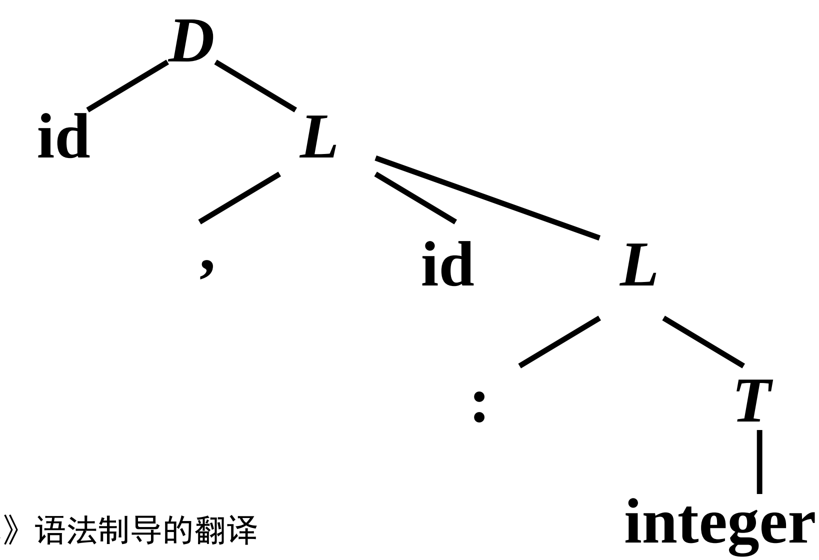
$L \to L_1$, id | id $\qquad$ $L_1.in = L.in, \dots$

等所需信息获得后再归约，改成从右向左归约

$D \to$ id L （S属性定义）

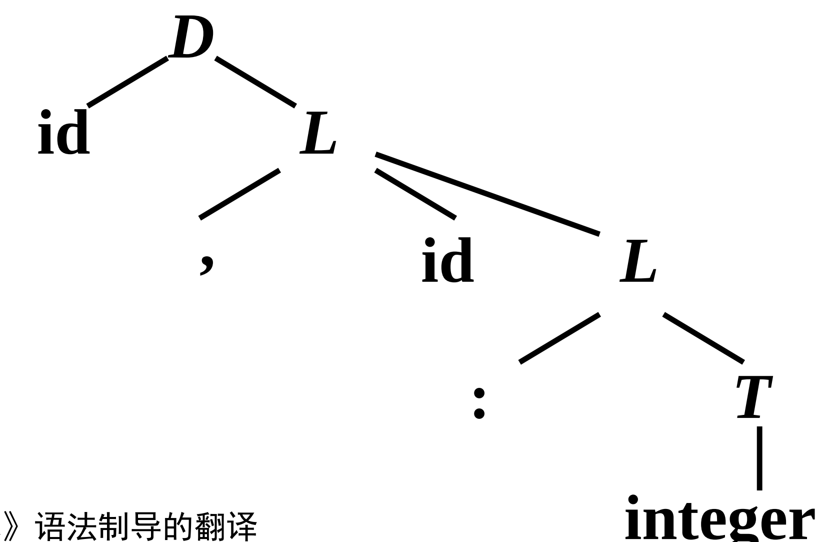
$L \to$, id L | : T

$T \to$ integer | char



用综合属性代替继承属性

$D \rightarrow \mathbf{id}\ L$	$\{\ addtype\ (\mathbf{id}.\ entry,\ L.\ type)\}$
$L \rightarrow ,\ \mathbf{id}\ L_1$	$\{\ L.\ type = L_1.\ Type;$ $addtype\ (\mathbf{id}.\ entry,\ L_1.\ type)\}$
$L \rightarrow :\ T$	$\{L.\ type = T.\ type\}$
$T \rightarrow \mathbf{integer}$	$\{T.\ type = integer\}$
$T \rightarrow \mathbf{real}$	$\{T.\ type = real\}$



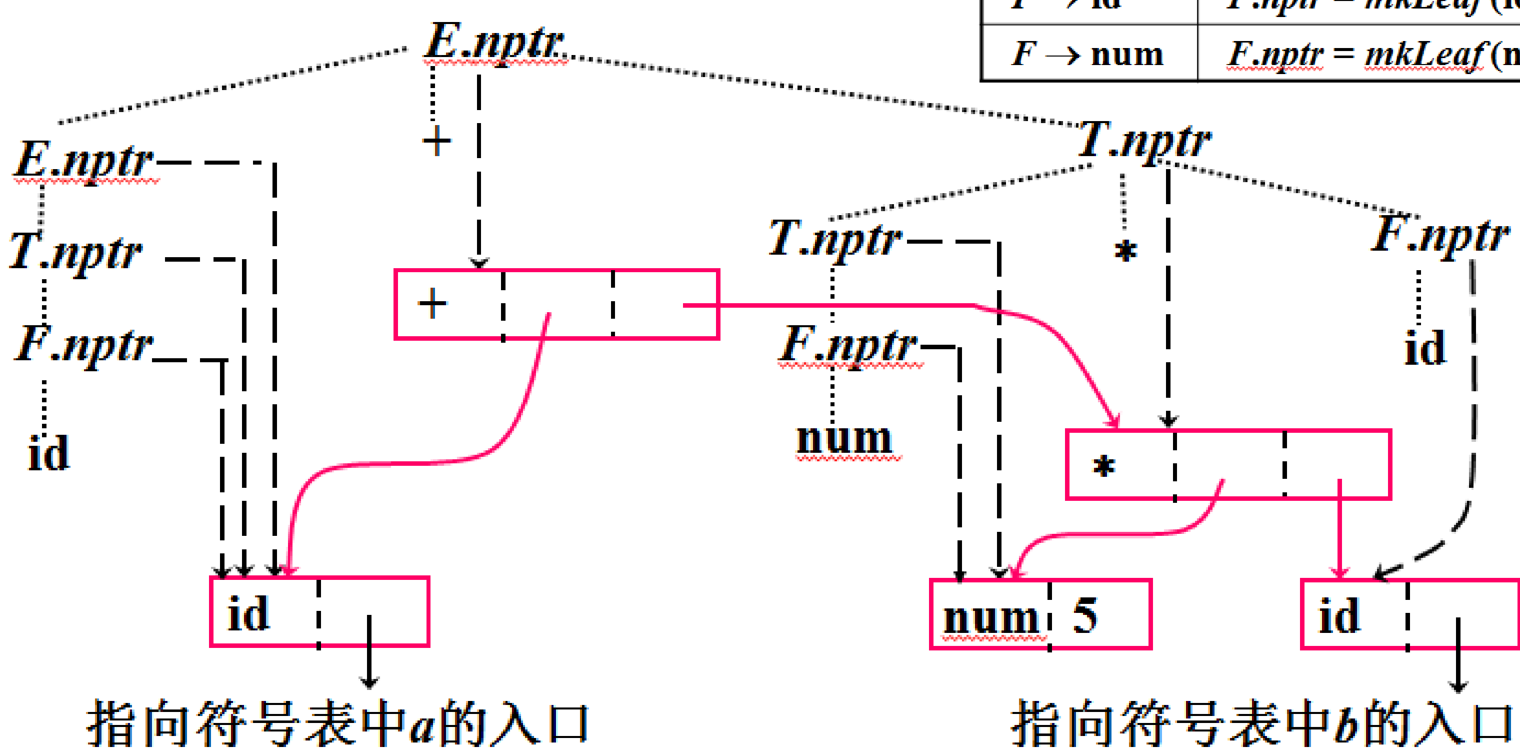
4.4 自下而上计算

- 综合属性的计算
- 删除翻译方案中嵌入的动作
- 继承属性的计算

S属性定义举例

产　生　式	语 义 规 则
$E \rightarrow E_1 + T$	$E.nptr = mkNode('+', E_1.nptr, T.nptr)$
$E \rightarrow T$	$E.nptr = T.nptr$
$T \rightarrow T_1 * F$	$T.nptr = mkNode('*', T_1.nptr, F.nptr)$
$T \rightarrow F$	$T.nptr = F.nptr$
$F \rightarrow (E)$	$F.nptr = E.nptr$
$F \rightarrow$ id	$F.nptr = mkLeaf$(id, id.$entry$)
$F \rightarrow$ num	$F.nptr = mkLeaf$(num, num.val)

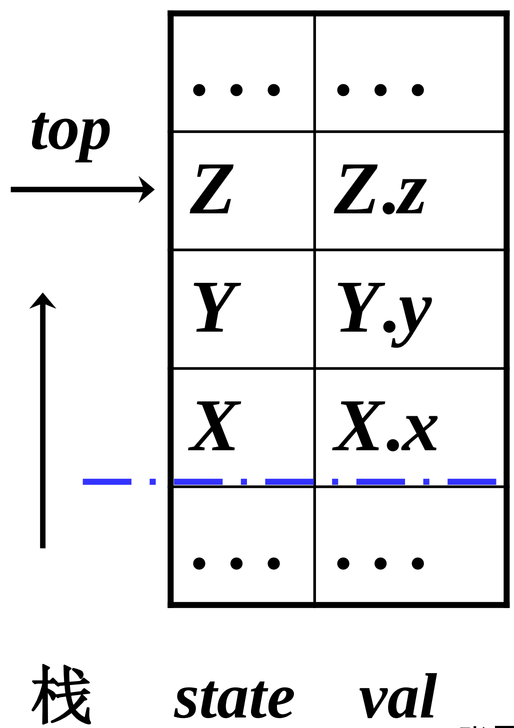
a+5∗*b*的语法树的构造



S属性的自下而上计算

□ 边分析边计算

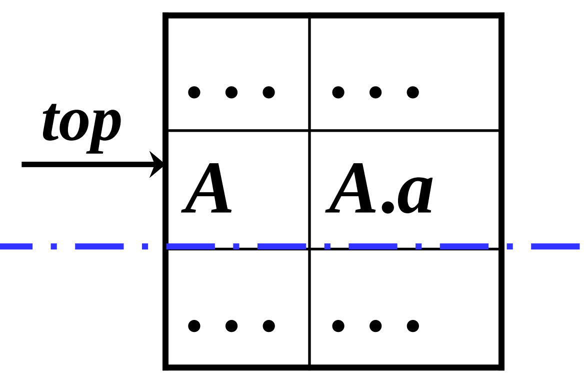
LR分析器的栈增加一个域来保存**综合属性值**



若产生式$A \rightarrow XYZ$的语义规则是

$$A.a = f\,(X.x,\ Y.y,\ Z.z),$$

那么归约后：



自下而上的翻译

例 简单计算器的语法制导定义改成栈操作代码

state	val
. . .	. . .
Z	$Z.z$
Y	$Y.y$
X	$X.x$
. . .	. . .

top →

栈 *state* *val*

产 生 式	语 义 规 则
$L \rightarrow E$ n	$print\ (E.val)$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val =$ digit.$lexval$

参见：bison-examples.tar.gz 中的config/expr1.y, expr.lex

自下而上的翻译

例　简单计算器的语法制导定义改成栈操作代码

top

state	val
. . .	. . .
Z	*Z.z*
Y	*Y.y*
X	*X.x*
. . .	. . .

栈　*state*　*val*

产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*E.val*)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	*F.val* = digit.*lexval*

自下而上的翻译

例 简单计算器的语法制导定义改成栈操作代码

top →

state	val
. . .	. . .
Z	$Z.z$
Y	$Y.y$
X	$X.x$
. . .	. . .

栈 *state* *val*

产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val*[*top*–1])
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val$ = digit.*lexval*

注：栈顶位置指示器top的修改由原来的分析程序在语义动作执行后去做

自下而上的翻译

例 简单计算器的语法制导定义改成栈操作代码

栈

state	val
. . .	. . .
Z	$Z.z$
Y	$Y.y$
X	$X.x$
. . .	. . .

top

产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val*[*top*–1]);
$E \rightarrow E_1 + T$	*val*[*top*–2] = *val* [*top*–2] +*val*[*top*];
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val$ = digit.*lexval*

注：栈顶位置指示器top的修改由原来的分析程序在语义动作执行后去做

自下而上的翻译

例 简单计算器的语法制导定义改成栈操作代码

top →

state	val
. . .	. . .
Z	$Z.z$
Y	$Y.y$
X	$X.x$
. . .	. . .

栈 *state* *val*

产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val*[*top*–1]);
$E \rightarrow E_1 + T$	*val*[*top*–2] = *val* [*top*–2] +*val*[*top*];
$E \rightarrow T$	
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val =$ digit.*lexval*

注：栈顶位置指示器**top**的修改由原来的分析程序在语义动作执行后去做

自下而上的翻译

例　简单计算器的语法制导定义改成栈操作代码

top

. . .	. . .
Z	*Z.z*
Y	*Y.y*
X	*X.x*
. . .	. . .

栈　*state*　*val*

产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val*[*top*–1]);
$E \rightarrow E_1 + T$	*val*[*top*–2] = *val* [*top*–2] +*val*[*top*];
$E \rightarrow T$	
$T \rightarrow T_1 * F$	*val*[*top*–2] = *val* [*top*–2] × *val*[*top*];
$T \rightarrow F$	*T.val* = *F.val*
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*

注：栈顶位置指示器top的修改由原来的分析程序在语义动作执行后去做

自下而上的翻译

例　简单计算器的语法制导定义改成栈操作代码

栈

state	val
. . .	. . .
Z	$Z.z$
Y	$Y.y$
X	$X.x$
. . .	. . .

top → Z

产 生 式	代 码 段
$L \to E$ n	*print* (*val*[*top*–1]);
$E \to E_1 + T$	*val*[*top*–2] = *val* [*top*–2] +*val*[*top*];
$E \to T$	
$T \to T_1 * F$	*val*[*top*–2] = *val* [*top*–2] × *val*[*top*];
$T \to F$	
$F \to (E)$	*F.val* = *E.val*
$F \to$ digit	*F.val* = digit.*lexval*

注：栈顶位置指示器top的修改由原来的分析程序在语义动作执行后去做

自下而上的翻译

例　简单计算器的语法制导定义改成栈操作代码

栈

state	val
…	…
Z	$Z.z$ ← top
Y	$Y.y$
X	$X.x$
…	…

产 生 式	代 码 段
$L \to E$ n	*print* (*val*[*top*–1]);
$E \to E_1 + T$	*val*[*top*–2] = *val* [*top*–2] +*val*[*top*];
$E \to T$	
$T \to T_1 * F$	*val*[*top*–2] = *val* [*top*–2] × *val*[*top*];
$T \to F$	
$F \to (E)$	*val*[*top*–2] = *val* [*top*–1] ;
$F \to$ digit	*F*.val = digit.*lexval*

注：栈顶位置指示器top的修改由原来的分析程序在语义动作执行后去做

自下而上的翻译

例　简单计算器的语法制导定义改成栈操作代码

top →

. . .	. . .
Z	$Z.z$
Y	$Y.y$
X	$X.x$
. . .	. . .

栈　*state*　*val*

产 生 式	代 码 段
$L \rightarrow E$ **n**	*print* (*val*[*top*–1]);
$E \rightarrow E_1 + T$	*val*[*top*–2] = *val* [*top*–2] +*val*[*top*];
$E \rightarrow T$	
$T \rightarrow T_1 * F$	*val*[*top*–2] = *val* [*top*–2] × *val*[*top*];
$T \rightarrow F$	
$F \rightarrow (E)$	*val*[*top*–2] = *val* [*top*–1] ;
$F \rightarrow$ **digit**	

注：栈顶位置指示器top的修改由原来的分析程序在语义动作执行后去做

Bison举例bison-examples: config/expr.y

```
%{
#include <stdio.h>
#include <math.h>
%}

%union {
  float val;
}
%token NUMBER
%token PLUS MINUS MULT DIV EXPON
…
%left  MINUS PLUS
%left  MULT DIV
%right EXPON

%type  <val> exp NUMBER
%%
```

各种语义值类型和域置于共用体中

声明exp、Number的语义值是在val域

```
input   :       | input line
            ;
…
exp     : NUMBER             { $$ = $1;       }
        | exp PLUS  exp      { $$ = $1 + $3;  }
        | exp MINUS exp      { $$ = $1 - $3;  }
        | exp MULT  exp      { $$ = $1 * $3;  }
        | exp DIV   exp      { $$ = $1 / $3;  }
        | MINUS  exp %prec MINUS { $$ = -$2; }
        | exp EXPON exp      { $$ = pow($1,$3);}
        | LB exp RB          { $$ = $2;       }
        ;
%%
yyerror(char *message)
{  printf("%s\n",message);}
int main(int argc, char *argv[])
{  yyparse();  return(0);}
```

*L*属性的自下而上计算

在自下而上分析的框架中实现*L*属性定义的方法

- **它能实现任何基于LL(1)文法的*L*属性定义**
- **也能实现许多（但不是所有的）基于LR(1) 的*L*属性定义**

删除翻译方案中嵌入的动作

□ 中缀表达式翻译成后缀表达式

$E \rightarrow T\ R$

$R \rightarrow +\ T\ \{print\ (\text{'+'})\}R_1 \mid -\ T\ \{print\ (\text{'–'})\}R_1 \mid \varepsilon$

$T \rightarrow \text{num}\ \{print(\text{num}.val)\}$

在文法中加入产生ε的标记非终结符，让每个嵌入动作由不同的标记非终结符*M*代表，并把该动作放在产生式$M \rightarrow \varepsilon$的右端　（继承属性=>综合属性）

$E \rightarrow T\ R$

$R \rightarrow +\ T\ M\ R_1 \mid -\ T\ N\ R_1 \mid \varepsilon$

$T \rightarrow \text{num}\ \{print\ (\text{num}.val)\}$

$M \rightarrow \varepsilon\ \{print\ (\text{'+'})\}$

$N \rightarrow \varepsilon\ \{print\ (\text{'–'})\}$

YACC会按这种方法来处理输入的文法，即为嵌入的语义动作引入ε产生式

L属性的自下而上计算

bison-examples: config/exprL.y

```
input  : …
   | input{ lineno ++; printf("Line %d:\t", lineno);}  line { printf("*"); } ;
```

$$表示LHS符号的语义值，$1, $2…依次为RHS中符号的语义值，本例中line的语义值通过$3 来引用

src/exprL.tab.c

```
case 4:
/* Line 1806 of yacc.c  */
#line 36 "config/exprL.y"
   { printf("*"); }
   break;
```

```
yyreduce:
  /* yyn is the number of a rule to reduce with.  */
 …
 YY_REDUCE_PRINT (yyn);
 switch (yyn) { …
   case 3:
   /* Line 1806 of yacc.c  */
   #line 32 "config/exprL.y"
            { lineno ++;
               printf("Line %d:\t", lineno);
             }    break;
```

产生式编号

4.4 自下而上计算

- 综合属性的计算
- 删除翻译方案中嵌入的动作
- 继承属性的计算

继承属性在分析栈中

情况1 属性位置可预测

例 int p, q, r

$D \rightarrow T\ \ \{L.in = T.type\}$

$\quad L$

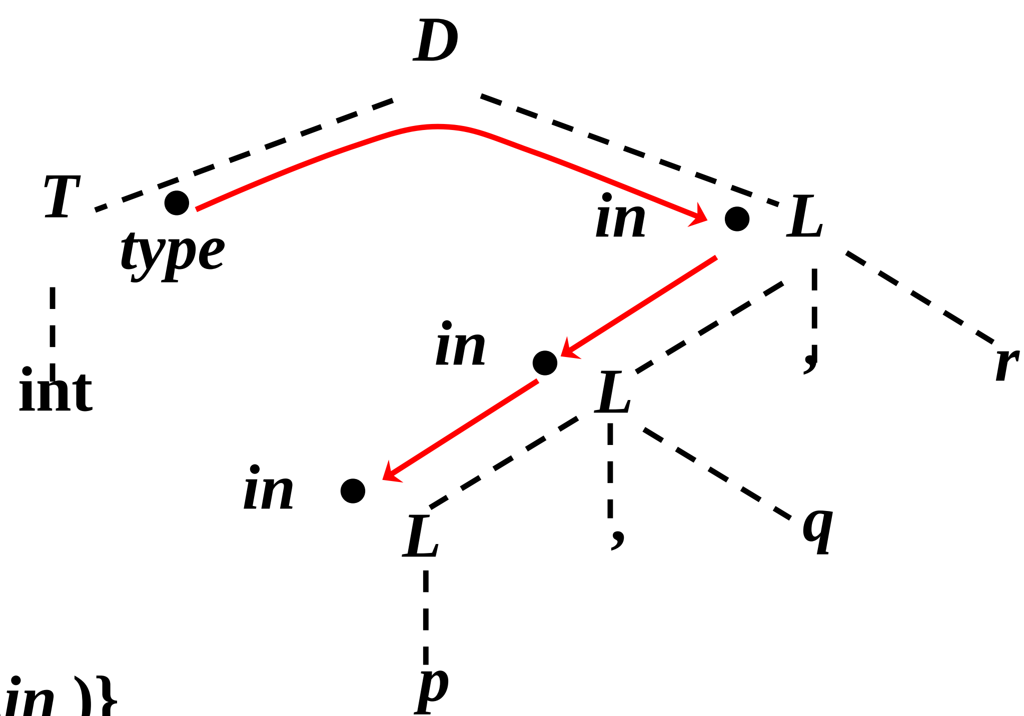
$T \rightarrow \text{int}\ \ \{T.type = integer\}$

$T \rightarrow \text{real}\ \{T.type = real\}$

$L \rightarrow \quad \{L_1.in = L.in\}$

$\quad L_1, \text{id}\ \{addtype\ (\text{id}.entry, L.in)\}$

$L \rightarrow \text{id}\quad \{addtype\ (\text{id}.entry, L.in)\}$



继承属性值已在分析栈中

继承属性在分析栈中

情况1 属性位置可预测

例 int p, q, r

$D \to T$ { ~~$L.in = T.type$~~ }

$\quad L$

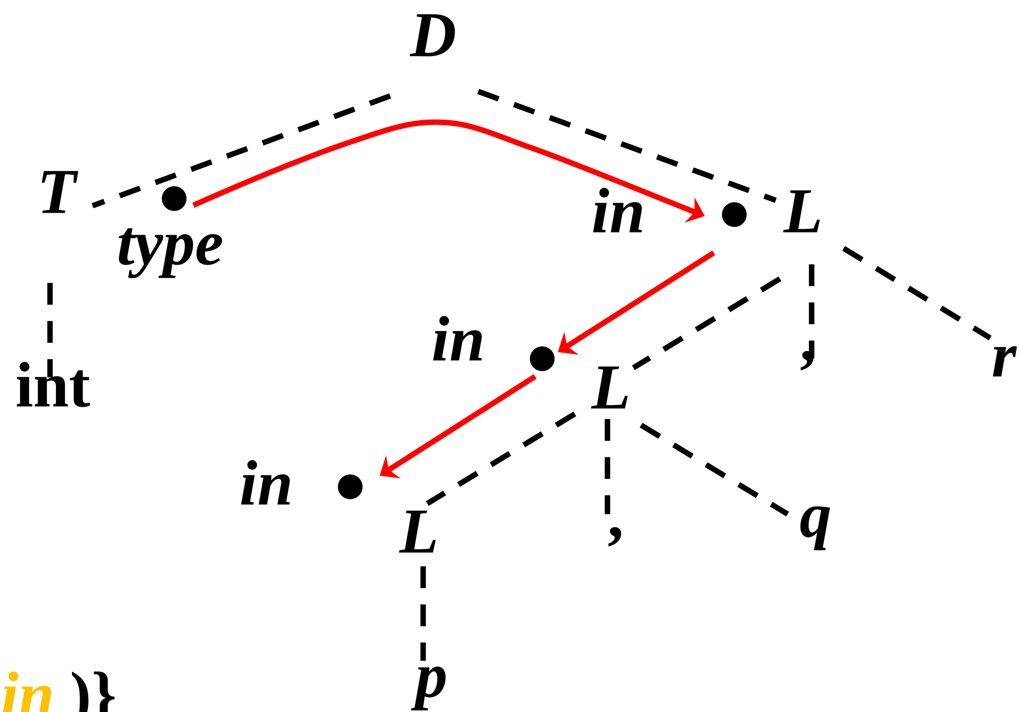
$T \to$ int {$T.type = integer$}

$T \to$ real {$T.type = real$}

$L \to$ { ~~$L_1.in = L.in$~~ }

$\quad L_1$, id {$addtype$ (id.$entry$, $L.in$)}

$L \to$ id {$addtype$ (id.$entry$, $L.in$)}



略去继承属性的计算

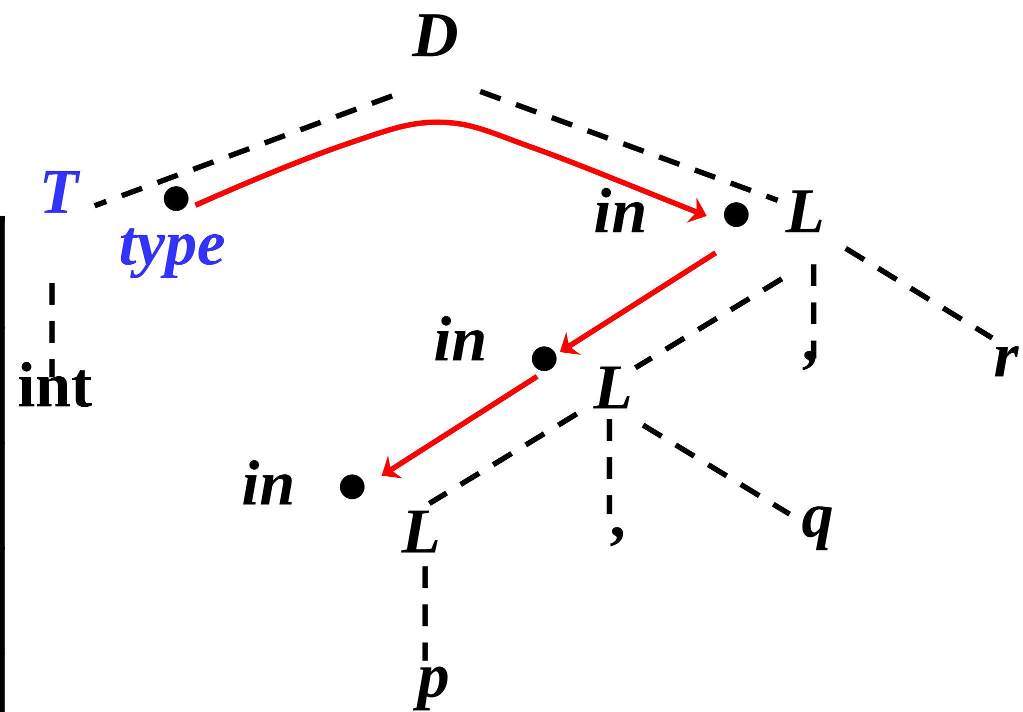
引用继承属性的地方改成

引用其他符号的综合属性

继承属性在分析栈中

情况1 属性位置可预测

产 生 式	代 码 段
$D \rightarrow TL$	
$T \rightarrow$ **int**	$val[top] = integer$
$T \rightarrow$ **real**	$val[top] = real$
$L \rightarrow L_1$, **id**	$addType(val[top], val[top-3])$;
$L \rightarrow$ **id**	$addType(val[top], val[top-1])$;



略去继承属性的计算
引用继承属性的地方改成
引用其他符号的综合属性

YACC中的继承属性定义

在内嵌动作代码中设置该文法符号的语义值

bison-examples: config/exprL1.y

```
line  : …
      | NUMBER { …
                $<val>lineno = $1;   // val是%union中声明的语义值类型
               // $<val>$ = $1;      // 该语义动作代码未指定名字时
               …
               } [lineno]
      exp EOL { …
               printf("Line %d: %g\n", (int) $<val>lineno, $3);
               …
      }
```

指明lineno的语义值所在的共用体的域

给内嵌语义动作对应的标记非终结符命名

YACC中的继承属性定义

在内嵌代码中使用存储在栈中任意固定相对位置的语义值

bison-examples: config/midrule.y

```
exp: a_1 a_2 { $<val>$ = 3; } { $<val>$ = $<val>3 + 1; } a_5
      sum_of_the_five_previous_values
    {
      USE (($1, $2, $<foo>3, $<foo>4, $5));
      printf ("%d\n", $6);
    }

sum_of_the_five_previous_values:
    {
      $$ = $<val>0 + $<val>-1 + $<val>-2 + $<val>-3 + $<val>-4;
    }
```

$<val>0、 $<val>-1、$<val>-2、$<val>-3、$<val>-4分别表示栈中a_5、{ $<val>$ = $<val>3 + 1; }、{ $<val>$ = 3; }、a_2、a_1文法符号的语义值

继承属性在分析栈中

情况2 属性位置不可预测

继承属性值已在分析栈中

$S \rightarrow aAC$	$C.i = A.s$
$S \rightarrow bABC$	$C.i = A.s$
$C \rightarrow c$	$C.s = g(C.i)$

A和C之间可能有B，也可能没有B，$C.i$ 的值有2种可能

- 增加标记非终结符，使得位置可以预测

$S \rightarrow aAC$	$C.i = A.s$
$S \rightarrow bABMC$	$M.i = A.s; C.i = M.s$
$C \rightarrow c$	$C.s = g(C.i)$
$M \rightarrow \varepsilon$	$M.s = M.i$

模拟继承属性的计算

- 继承属性是综合属性的函数

$S \to aAC \quad C.i = f(A.s)$

$C \to c \quad\quad C.s = g(C.i)$

继承属性不直接等于某个综合属性

- 增加标记非终结符，把$f(A.s)$的计算移到对标记非终结符归约时进行

$S \to aANC \quad\quad N.i = A.s;\ C.i = N.s$

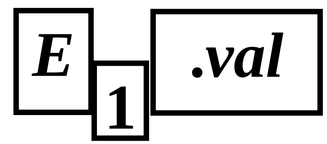
$N \to \varepsilon \quad\quad N.s = f(N.i)$

$C \to c \quad\quad C.s = g(C.i)$

L属性定义的自下而上计算

例 数学排版语言EQN

E sub 1 .val



$$S \to B$$

$$B \to B_1\ B_2$$

$$B \to B_1 \text{ sub } B_2$$

$$B \to \text{text}$$

数学排版语言EQN

语法制导定义　E sub 1 .val　　$\boxed{E}_{\boxed{1}}\boxed{.val}$

ps-point size (*L*属性); *ht*-height(*S*属性)

产 生 式	语 义 规 则
$S \to B$	$B.ps = 10$; $S.ht = B.ht$
$B \to B_1\ B_2$	$B_1.ps = B.ps$; $B_2.ps = B.ps$; $B.ht = max(B_1.ht, B_2.ht)$
$B \to B_1$ sub B_2	$B_1.ps = B.ps$; $B_2.ps = shrink(B.ps)$; $B.ht = disp(B_1.ht, B_2.ht)$
$B \to$ text	$B.ht = \text{text}.h \times B.ps$

数学排版语言EQN

产 生 式	语 义 规 则
$S \to B$	$B.ps = 10;\ S.ht = B.ht$
$B \to B_1 B_2$	$B_1.ps = B.ps;\ B_2.ps = B.ps;$ $B.ht = max(B_1.ht, B_2.ht)$
$B \to B_1$ sub B_2	$B_1.ps = B.ps;\ B_2.ps = shrink(B.ps);$ $B.ht = disp\ (B_1.ht, B_2.ht)$
$B \to$ text	$B.ht =$ text.$h \times B.ps$

$S \to$ $\{B.ps = 10\}$

B $\{S.ht = B.ht\}$

$B \to$ $\{B_1.ps = B.ps\}$

B_1 $\{B_2.ps = B.ps\}$

B_2 $\{B.ht = max(B_1.ht, B_2.ht)\}$

$B \to$ $\{B_1.ps = B.ps\}$

B_1

sub $\{B_2.ps = shrink(B.ps)\}$

B_2 $\{B.ht = disp\ (B_1.ht, B_2.ht)\}$

$B \to$ text $\{B.ht = \text{text}.h \times B.ps\}$

EQN： 自下而上计算的实现

产 生 式	语 义 规 则
$S \to LB$	$B.ps = L.s$; $S.ht = B.ht$
$L \to \varepsilon$	$L.s = 10$ 将$B.ps$存入栈中，便于引用
$B \to B_1\ MB_2$	$B_1.ps = B.ps$; $M.i = B.ps$; $B_2.ps = M.s; B.ht = max(B_1.ht, B_2.ht)$
$M \to \varepsilon$	$M.s = M.i$
$B \to B_1$ sub NB_2	$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp(B_1.ht, B_2.ht)$
$N \to \varepsilon$	$N.s = shrink(N.i)$
$B \to$ text	$B.ht = \text{text}.h \times B.ps$

EQN： 自下而上计算的实现

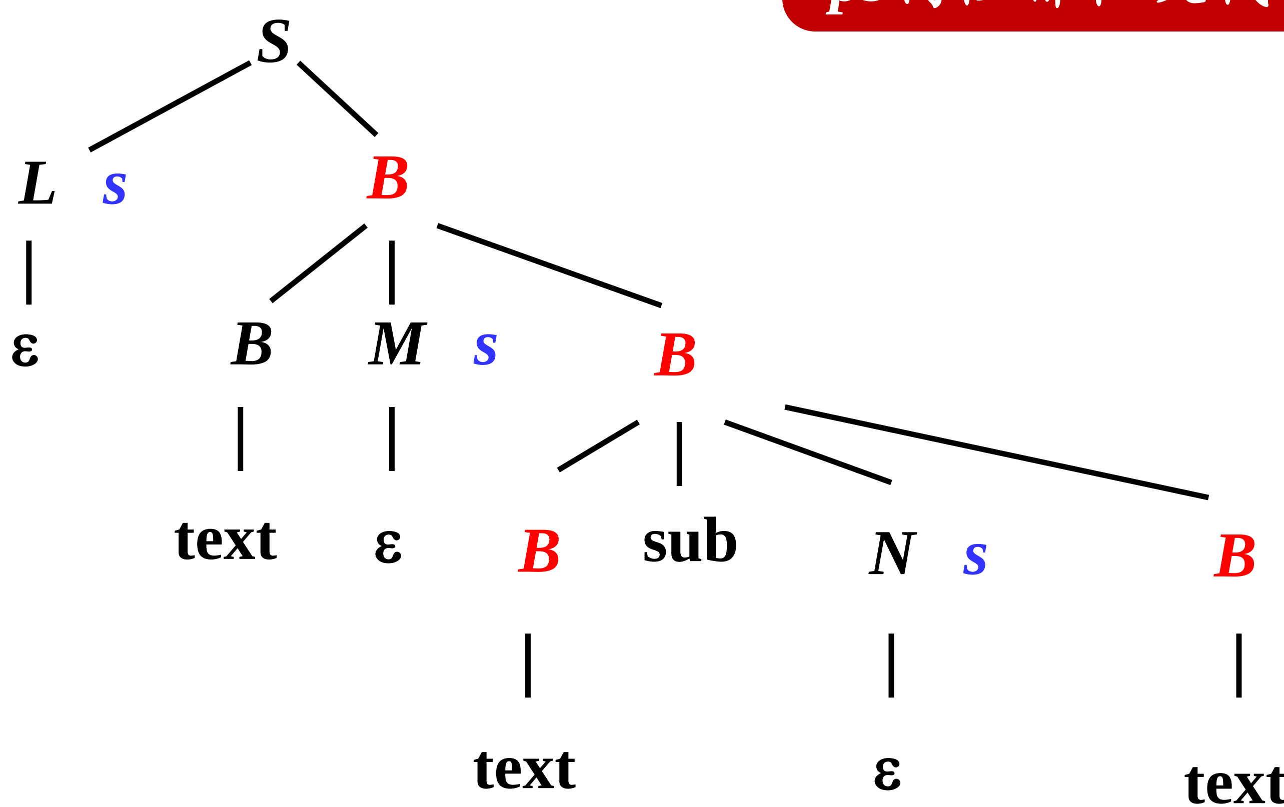
产 生 式	语 义 规 则
$S \rightarrow LB$	$B.ps = L.s$; $S.ht = B.ht$
$L \rightarrow \varepsilon$	$L.s = 10$ 将$B.ps$存入栈中，便于引用
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps$; $M.i = B.ps$; $B_2.ps = M.s$;$B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$ 单纯为了属性位置可预测
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp\ (B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht = \text{text}.h \times B.ps$

EQN： 自下而上计算的实现

产 生 式	语 义 规 则
$S \to LB$	$B.ps = L.s$; $S.ht = B.ht$
$L \to \varepsilon$	$L.s = 10$ 将$B.ps$存入栈中，便于引用
$B \to B_1\ MB_2$	$B_1.ps = B.ps$; $M.i = B.ps$; $B_2.ps = M.s$;$B.ht = max(B_1.ht, B_2.ht)$
$M \to \varepsilon$	$M.s = M.i$ 单纯为了属性位置可预测
$B \to B_1$ sub NB_2	$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp\ (B_1.ht, B_2.ht)$
$N \to \varepsilon$	$N.s = shrink(N.i)$ 兼有计算功能
$B \to$ text	$B.ht = \text{text}.h \times B.ps$

EQN：自下而上计算的实现

在text归约成*B*时，*B*的*ps*属性都在次栈顶位置



EQN：自下而上计算的实现

产 生 式	语 义 规 则
$S \to LB$	$B.ps = L.s;\ \ S.ht = B.ht$
$L \to \varepsilon$	$L.s = 10$
$B \to B_1\ MB_2$	$B_1.ps = B.ps;\ \ M.i = B.ps;$ $B_2.ps = M.s; B.ht = max(B_1.ht, B_2.ht)$
$M \to \varepsilon$	$M.s = M.i$
$B \to B_1$ sub NB_2	$B_1.ps = B.ps;\ \ N.i = B.ps;$ $B_2.ps = N.s;\ B.ht = disp\ (B_1.ht, B_2.ht)$
$N \to \varepsilon$	$N.s = shrink(N.i)$
$B \to$ text	$B.ht = \text{text}.h \times B.ps$

继承属性的值等于栈中某个综合属性的值，因此栈中只保存综合属性的值

EQN： 自下而上计算的实现

产 生 式	语 义 规 则
$S \to LB$	$val[top-1] = val[top]$
$L \to \varepsilon$	$L.s = 10$
$B \to B_1\ MB_2$	$B_1.ps = B.ps;\ \ M.i = B.ps;$ $B_2.ps = M.s; B.ht = max(B_1.ht, B_2.ht)$
$M \to \varepsilon$	$M.s = M.i$
$B \to B_1$ sub NB_2	$B_1.ps = B.ps;\ \ N.i = B.ps;$ $B_2.ps = N.s;\ B.ht = disp(B_1.ht, B_2.ht)$
$N \to \varepsilon$	$N.s = shrink(N.i)$
$B \to$ text	$B.ht = \text{text}.h \times B.ps$

$B.ps = L.s;\ \ S.ht = B.ht$

EQN： 自下而上计算的实现

产 生 式	语 义 规 则
$S \to LB$	$val[top-1] = val[top]$
$L \to \varepsilon$	$val[top+1] = 10$
$B \to B_1\ MB_2$	$B_1.ps = B.ps;\ \ M.i = B.ps;$ $B_2.ps = M.s; B.ht = max(B_1.ht,\ B_2.ht\ )$
$M \to \varepsilon$	$M.s = M.i$
$B \to B_1$ sub NB_2	$B_1.ps = B.ps;\ \ N.i = B.ps;$ $B_2.ps = N.s;\ B.ht = disp\ (B_1.ht,\ B_2.ht\ )$
$N \to \varepsilon$	$N.s = shrink(N.i)$
$B \to$ text	$B.ht = \text{text}.h \times B.ps$

$L.s = 10$

EQN： 自下而上计算的实现

产 生 式	语 义 规 则
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ sub NB_2	$B_1.ps = B.ps;\ \ N.i = B.ps;$ $B_2.ps = N.s;\ B.ht = disp\ (B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht = \text{text}.h \times B.ps$

$B_1.ps = B.ps; M.i = B.ps; B_2.ps = M.s; B.ht = max(B_1.ht, B_2.ht)$

EQN： 自下而上计算的实现

产 生 式	语 义 规 则
$S \to LB$	$val[top-1] = val[top]$
$L \to \varepsilon$	$val[top+1] = 10$
$B \to B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \to \varepsilon$	$val[top+1] = val[top-1]$
$B \to B_1$ sub NB_2	$B_1.ps = B.ps;\ \ N.i = B.ps;$ $B_2.ps = N.s;\ B.ht = disp\ (B_1.ht, B_2.ht)$
$N \to \varepsilon$	$N.s = shrink(N.i)$
$B \to \text{text}$	$B.ht = \text{text}.h \times B.ps$

$M.i = B.ps;\ \ M.s = M.i$

EQN：自下而上计算的实现

产 生 式	语 义 规 则
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1$ sub NB_2	$val[top-3] = disp\ (val[top-3], val[top])$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow$ text	$B.ht = \text{text}.h \times B.ps$

$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp\ (B_1.ht, B_2.ht)$

EQN： 自下而上计算的实现

产 生 式	语 义 规 则
$S \to LB$	$val[top-1] = val[top]$
$L \to \varepsilon$	$val[top+1] = 10$
$B \to B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \to \varepsilon$	$val[top+1] = val[top-1]$
$B \to B_1$ sub NB_2	$val[top-3] = disp(val[top-3], val[top])$
$N \to \varepsilon$	$val[top+1] = shrink(val[top-2])$
$B \to$ text	$B.ht = \text{text}.h \times B.ps$

$B.ht = \text{text}.h \times B.ps$

EQN： 自下而上计算的实现

产 生 式	语 义 规 则
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1 MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1$ sub NB_2	$val[top-3] = disp(val[top-3], val[top])$
$N \rightarrow \varepsilon$	$val[top+1] = shrink(val[top-2])$
$B \rightarrow$ text	$val[top] = val[top] \times val[top-1]$

$N.i = B.ps;\ \ N.s = shrink(N.i)$

下期预告：语义分析