



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》

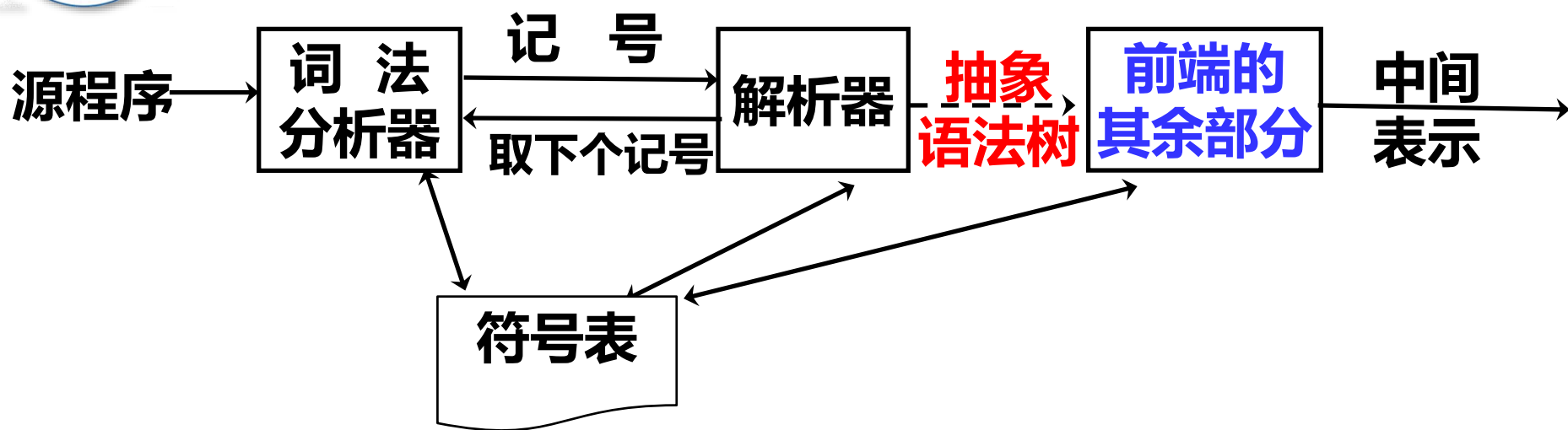
张昱

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



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 $8\ 5\ +\ 2\ -$

$$E \rightarrow T R$$
$$R \rightarrow \text{addop } T \{\textit{print}(\text{addop.lexeme})\} R_1 \mid \varepsilon$$
$$T \rightarrow \text{num } \{\textit{print}(\text{num.val})\}$$
$$E \Rightarrow T R \Rightarrow \text{num } \{\textit{print}(8)\} R$$
$$\Rightarrow \text{num } \{\textit{print}(8)\} \text{addop } T \{\textit{print}(+)\} R$$
$$\Rightarrow \text{num } \{\textit{print}(8)\} \text{addop num } \{\textit{print}(5)\} \{\textit{print}(+)\} R$$
$$\dots \{\textit{print}(8)\} \{\textit{print}(5)\} \{\textit{print}(+)\} \text{addop } T \{\textit{print}(-)\} R$$
$$\dots \{\textit{print}(8)\} \{\textit{print}(5)\} \{\textit{print}(+)\} \{\textit{print}(2)\} \{\textit{print}(-)\}$$



翻译方案—内嵌有信息传播

$$\begin{array}{lll} E \rightarrow T & \{R.i = T.nptr\} & T + T + T + \dots \\ & R & \\ & \{E.nptr = R.s\} & \\ R \rightarrow + & & \\ & T & \{R_1.i = mkNode ('+', R.i, T.nptr)\} \\ & R_1 & \{R.s = R_1.s\} \\ R \rightarrow \varepsilon & \{R.s = R.i\} & \\ T \rightarrow F & \{W.i = F.nptr\} & \\ & W & \{T.nptr = W.s\} \\ W \rightarrow * & & \\ & F & \{W_1.i = mkNode ('*', W.i, F.nptr)\} \\ & W_1 & \{W.s = W_1.s\} \\ W \rightarrow \varepsilon & \{W.s = W.i\} & \end{array}$$

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 , nptr);
```

```
    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 $m, n : \text{integer}$

$$D \rightarrow L : T \qquad L.in = T.type$$
$$T \rightarrow \text{integer} \mid \text{char} \qquad T.type = \dots$$
$$L \rightarrow L_1, \text{id} \mid \text{id} \qquad L_1.in = L.in, \dots$$

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 $m, n : \text{integer}$

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

$T \rightarrow \text{integer} \mid \text{char}$ $T.type = \dots$

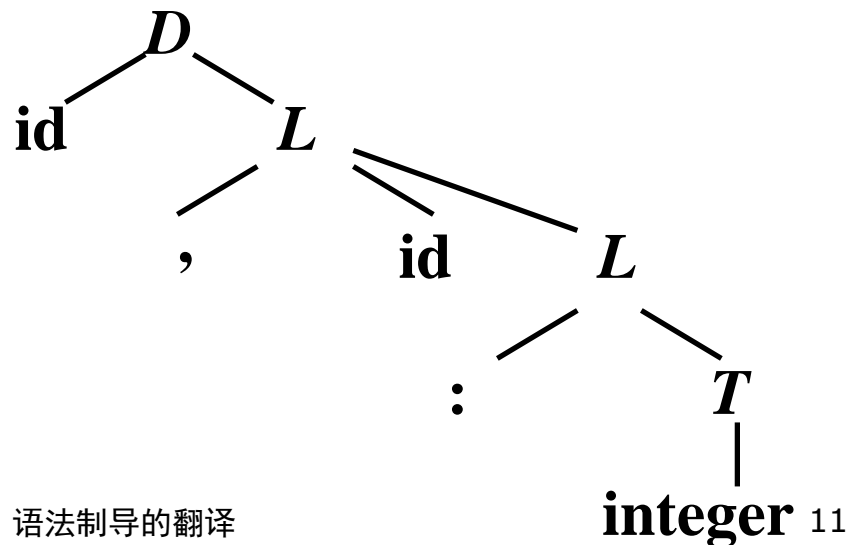
$L \rightarrow L_1, \text{id} \mid \text{id}$ $L_1.in = L.in, \dots$

等所需信息获得后再归约，改成从右向左归约

$D \rightarrow \text{id } L$ (S属性定义)

$L \rightarrow , \text{id } L \mid : T$

$T \rightarrow \text{integer} \mid \text{char}$





用综合属性代替继承属性

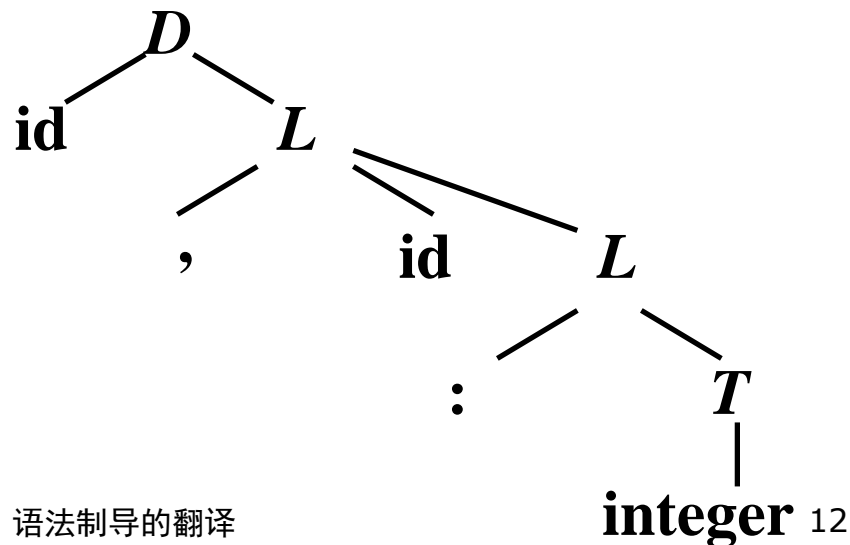
$D \rightarrow \text{id } L \quad \{ \text{addtype}(\text{id. entry}, L.\text{type}) \}$

$L \rightarrow , \text{id } L_1 \quad \{ L.\text{type} = L_1.\text{Type};$
 $\quad \text{addtype}(\text{id. entry}, L_1.\text{type}) \}$

$L \rightarrow : T \quad \{ L.\text{type} = T.\text{type} \}$

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

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





4.4 自下而上计算

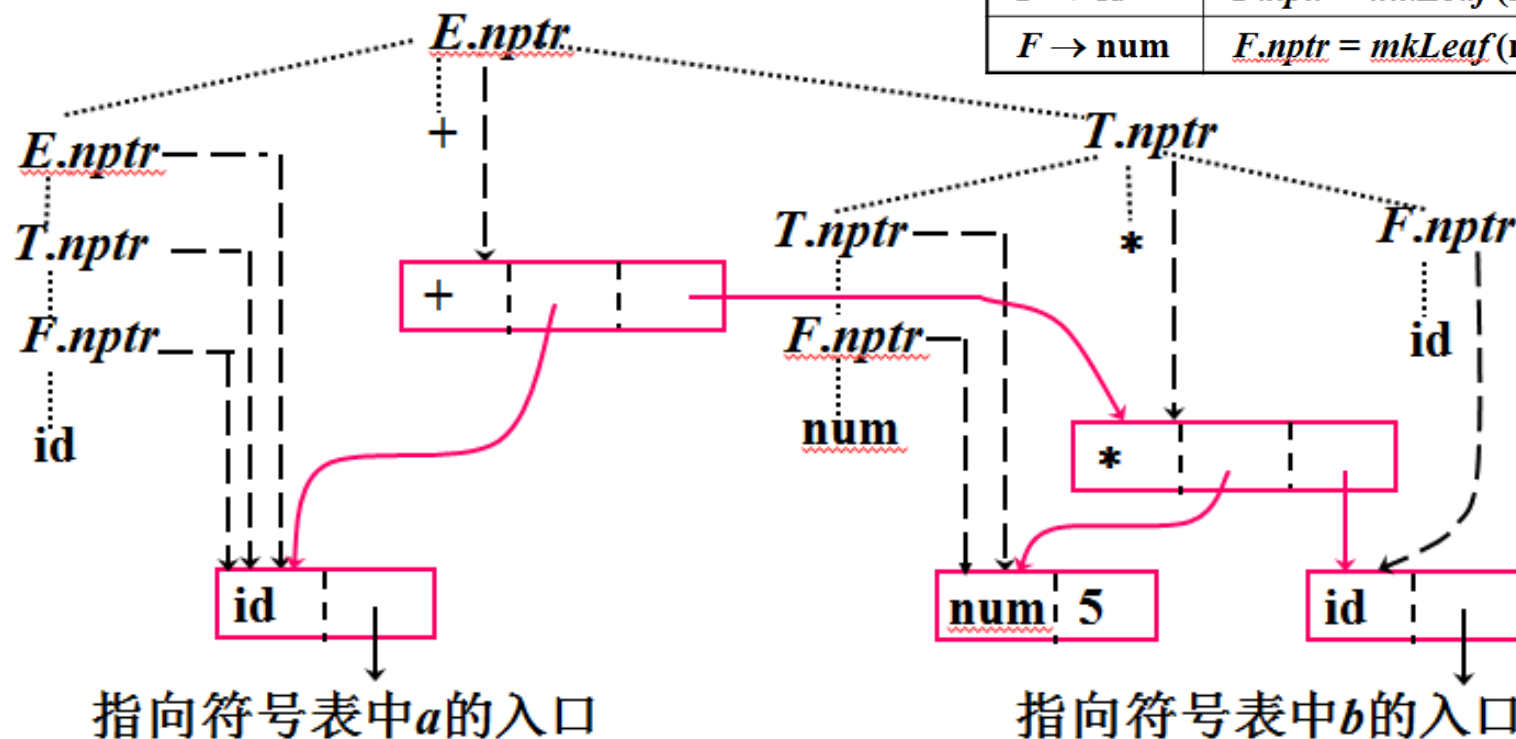
- 综合属性的计算
- 删除翻译方案中嵌入的动作
- 继承属性的计算



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

\xrightarrow{top}
	Z	$Z.z$
	Y	$Y.y$
	X	$X.x$

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:

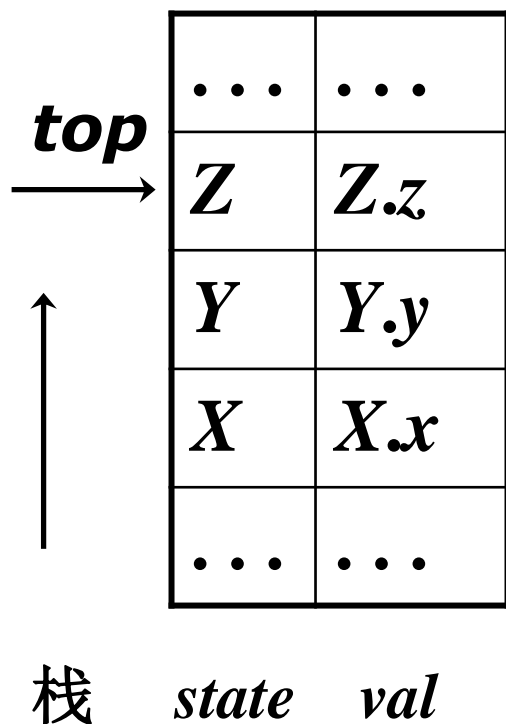
\xrightarrow{top}
	A	$A.a$

栈 *state* *val*



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



产生式	语义规则
$L \rightarrow E n$	$print(E.val)$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

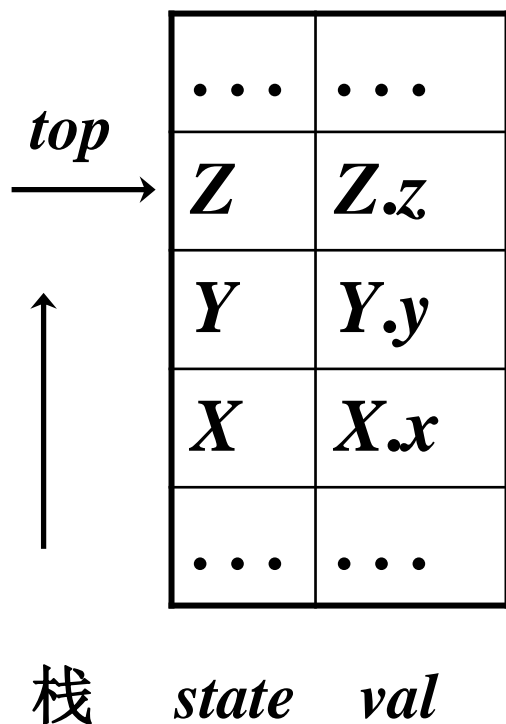
参见: bison-examples.tar.gz 中的 `config/expr1.y`, `expr.lex`

张昱: 《编译原理和技术》语法制导的翻译



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

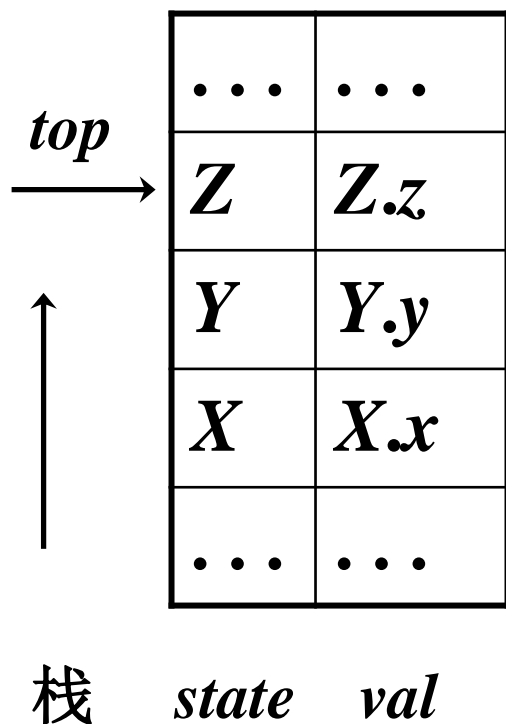


产生式	代码段
$L \rightarrow E n$	$print(E.val)$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$



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



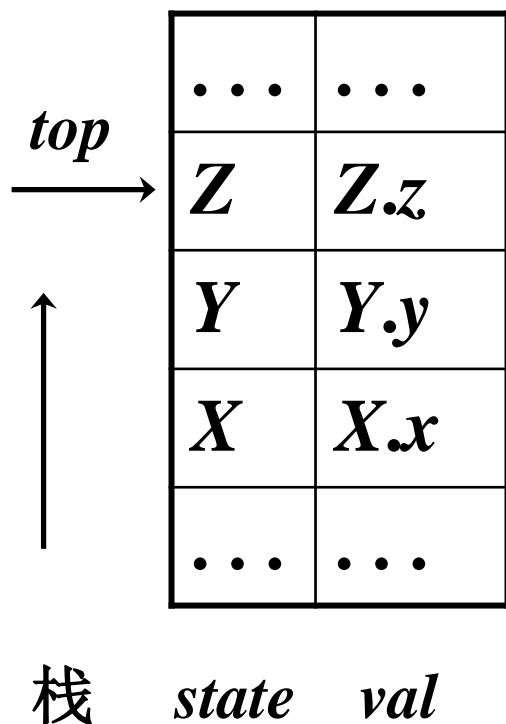
产生式	代码段
$L \rightarrow E n$	$print(val[top-1])$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

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



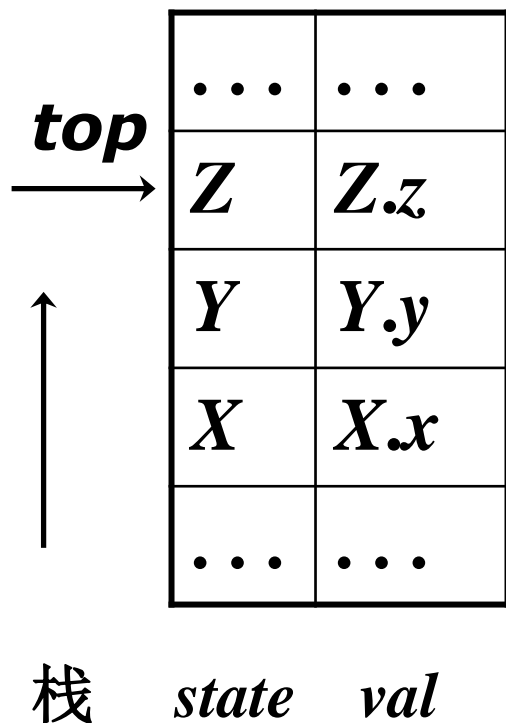
产生式	代码段
$L \rightarrow E n$	$print(val[top-1]);$
$E \rightarrow E_1 + T$	$val[top-2] = val[top-2] + val[top];$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

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



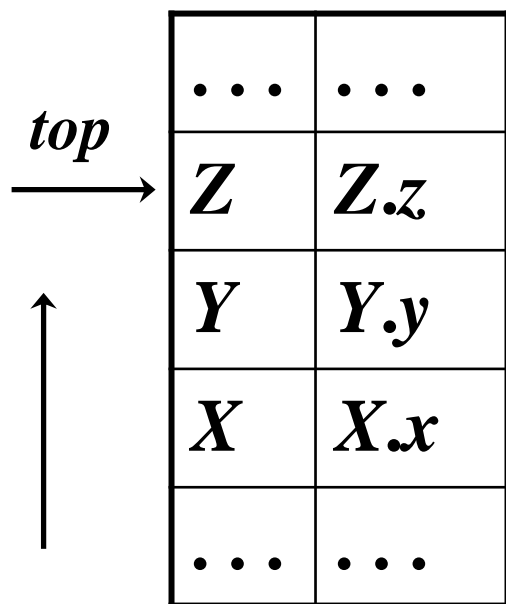
产生式	代码段
$L \rightarrow E n$	$print(val[top-1]);$
$E \rightarrow E_1 + T$	$val[top-2] = val[top-2] + val[top];$
$E \rightarrow T$	
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

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



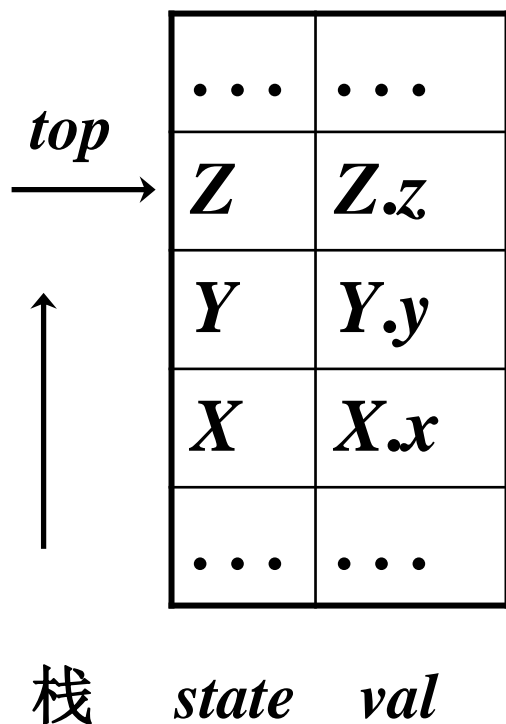
产生式	代码段
$L \rightarrow E n$	$print(val[top-1]);$
$E \rightarrow E_1 + T$	$val[top-2] = val[top-2] + val[top];$
$E \rightarrow T$	
$T \rightarrow T_1 * F$	$val[top-2] = val[top-2] \times val[top];$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

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



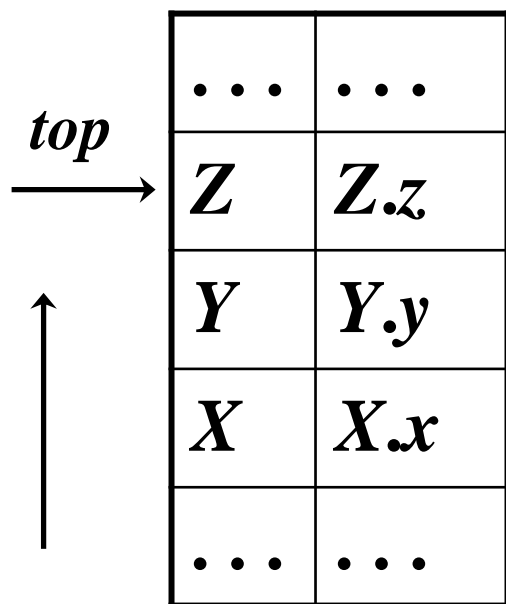
产生式	代码段
$L \rightarrow E n$	$print(val[top-1]);$
$E \rightarrow E_1 + T$	$val[top-2] = val[top-2] + val[top];$
$E \rightarrow T$	
$T \rightarrow T_1 * F$	$val[top-2] = val[top-2] \times val[top];$
$T \rightarrow F$	
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

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

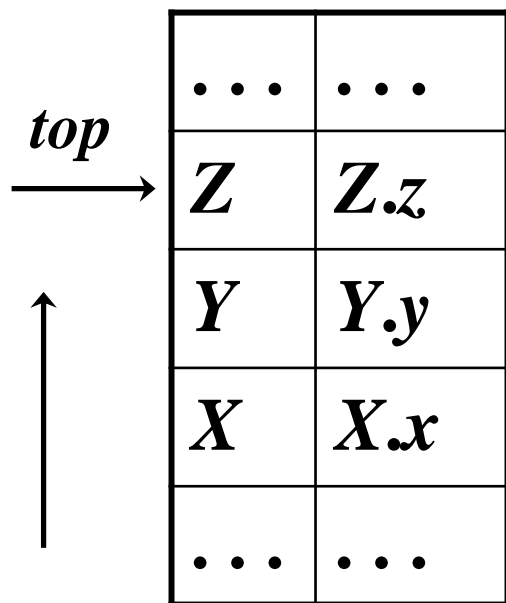
产生式	代码段
$L \rightarrow E n$	$print(val[top-1]);$
$E \rightarrow E_1 + T$	$val[top-2] = val[top-2] + val[top];$
$E \rightarrow T$	
$T \rightarrow T_1 * F$	$val[top-2] = val[top-2] \times val[top];$
$T \rightarrow F$	
$F \rightarrow (E)$	$val[top-2] = val[top-1];$
$F \rightarrow digit$	$F.val = digit.lexval$

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

产生式	代码段
$L \rightarrow E n$	$print(val[top-1]);$
$E \rightarrow E_1 + T$	$val[top-2] = val[top-2] + val[top];$
$E \rightarrow T$	
$T \rightarrow T_1 * F$	$val[top-2] = val[top-2] \times val[top];$
$T \rightarrow F$	
$F \rightarrow (E)$	$val[top-2] = val[top-1];$
$F \rightarrow digit$	

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

$$E \rightarrow T R$$

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

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

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 \Rightarrow 综合属性）

$$E \rightarrow T R$$

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

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

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

$$N \rightarrow \varepsilon \{print\ ('-')\}$$

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



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

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 $\text{int } p, q, r$

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

L

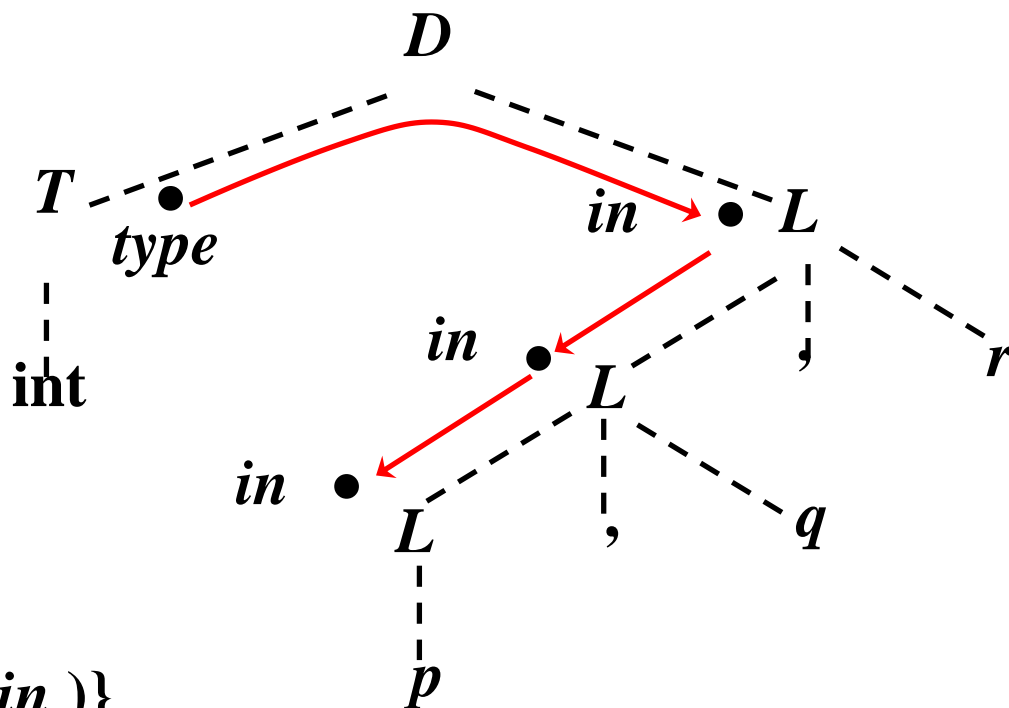
$T \rightarrow \text{int} \quad \{T.type = \text{integer}\}$

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

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

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

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



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`

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

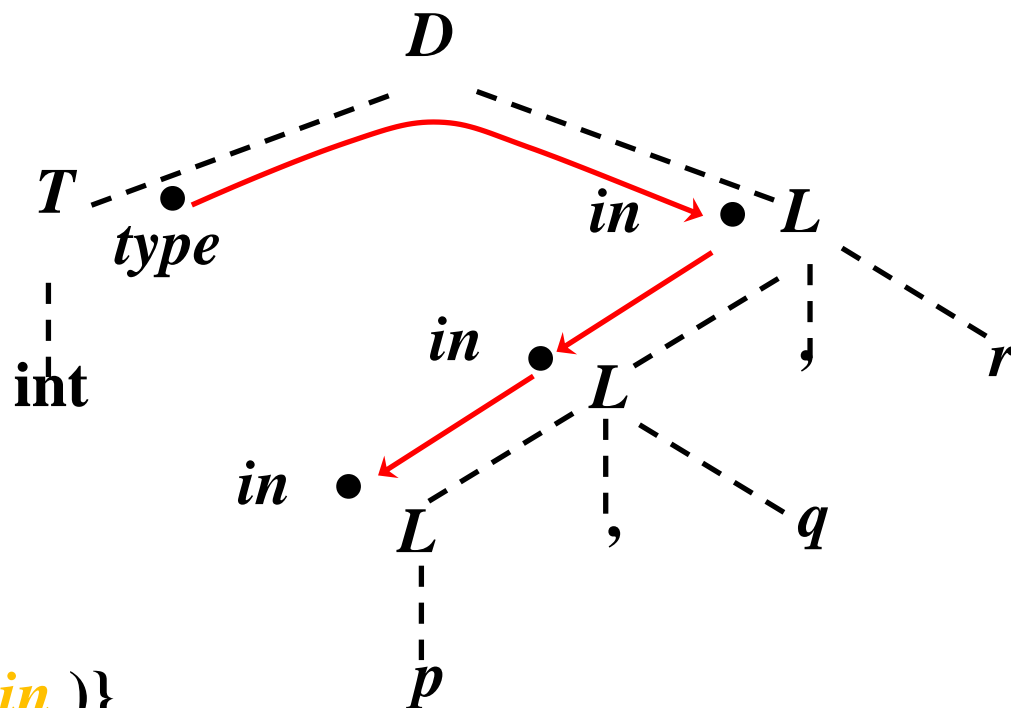
$T \rightarrow \text{int} \quad \{ T.type = \text{integer} \}$

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

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

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

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



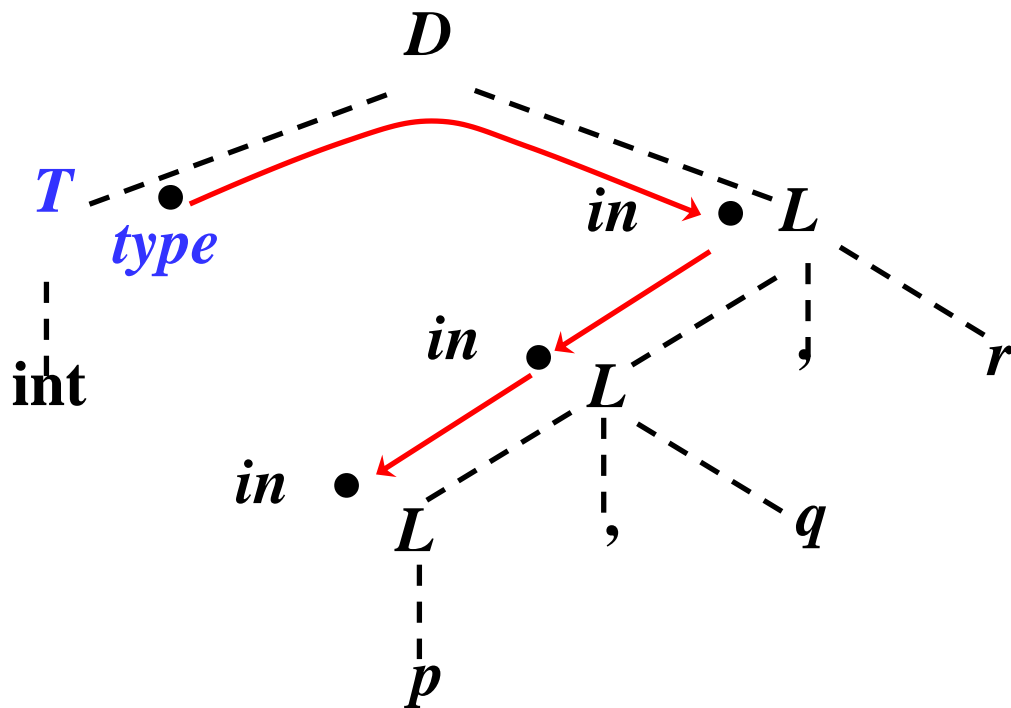
略去继承属性的计算
引用继承属性的地方改成
引用其他符号的综合属性



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
$D \rightarrow TL$	
$T \rightarrow \text{int}$	$val[top] = integer$
$T \rightarrow \text{real}$	$val[top] = real$
$L \rightarrow L_1, id$	$addType(val[top],$ $val[top-3]);$
$L \rightarrow id$	$addType(val[top],$ $val[top-1]);$



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 { ...

指明lineno的语义值
所在的共用体的域

$\$<val>lineno$ = \$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]**

给内嵌语义动作对应的
标记非终结符命名

exp EOL { ...

printf("Line %d: %g\n", (int) **$\$<val>lineno$** , \$3);

...

}



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

$$S \rightarrow aAC \quad C.i = A.s$$

$$S \rightarrow bA\textcolor{violet}{B}C \quad C.i = A.s$$

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

B可能在, 也可能不在A和C之间, 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

$$S \rightarrow aAC \quad C.i = A.s$$

$$S \rightarrow bAB\textcolor{violet}{M}C \quad \textcolor{violet}{M}.i = A.s; \textcolor{violet}{C}.i = \textcolor{violet}{M}.s$$

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

$$\textcolor{violet}{M} \rightarrow \varepsilon \quad \textcolor{violet}{M}.s = \textcolor{violet}{M}.i$$

继承属性值
已在分析栈中



模拟继承属性的计算

□ 继承属性是综合属性的函数

$$S \rightarrow aAC \quad \textcolor{red}{C.i} = \textcolor{blue}{f(A.s)}$$

$$C \rightarrow c \qquad C.s = g(C.i)$$

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

$$S \rightarrow aA\textcolor{red}{N}C$$

$$\textcolor{red}{N.i} = A.s; \textcolor{red}{C.i} = N.s$$

$$\textcolor{red}{N} \rightarrow \varepsilon$$

$$\textcolor{red}{N.s} = \textcolor{red}{f(N.i)}$$

$$C \rightarrow c$$

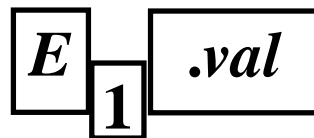
$$C.s = g(C.i)$$



L 属性定义的自下而上计算

例 数学排版语言EQN

$E \text{ sub } 1 \text{ .val}$



$S \rightarrow B$

$B \rightarrow B_1 B_2$

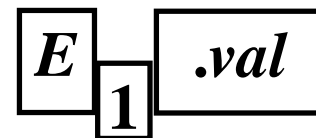
$B \rightarrow B_1 \text{ sub } B_2$

$B \rightarrow \text{text}$



数学排版语言EQN

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



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

产生式	语义规则
$S \rightarrow B$	$B.ps = 10; S.ht = B.ht$
$B \rightarrow B_1 B_2$	$B_1.ps = B.ps; B_2.ps = B.ps;$ $B.ht = \max(B_1.ht, B_2.ht)$
$B \rightarrow B_1 \text{ sub } B_2$	$B_1.ps = B.ps; B_2.ps = \text{shrink}(B.ps);$ $B.ht = \text{disp}(B_1.ht, B_2.ht)$
$B \rightarrow \text{text}$	$B.ht = \text{text.h} \times B.ps$



数学排版语言EQN

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

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

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

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

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

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

B_1

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

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

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

产生式	语义规则
$S \rightarrow B$	$B.ps = 10; S.ht = B.ht$
$B \rightarrow B_1 B_2$	$B_1.ps = B.ps; B_2.ps = B.ps;$ $B.ht = \max(B_1.ht, B_2.ht)$
$B \rightarrow B_1 \text{ sub } B_2$	$B_1.ps = B.ps; B_2.ps = \text{shrink}(B.ps);$ $B.ht = \text{disp}(B_1.ht, B_2.ht)$
$B \rightarrow \text{text}$	$B.ht = \text{text.h} \times B.ps$



EQN: 自下而上计算的实现

产生式	语义规则
$S \rightarrow LB$	$B.ps = L.s; S.ht = B.ht$
$L \rightarrow \epsilon$	$L.s = 10$ 将 $B.ps$ 存入栈中, 便于引用
$B \rightarrow B_1 MB_2$	$B_1.ps = B.ps; M.i = B.ps;$ $B_2.ps = M.s; B.ht = \max(B_1.ht, B_2.ht)$
$M \rightarrow \epsilon$	$M.s = M.i$
$B \rightarrow B_1 \text{ sub } NB_2$	$B_1.ps = B.ps; N.i = B.ps;$ $B_2.ps = N.s; B.ht = \text{disp}(B_1.ht, B_2.ht)$
$N \rightarrow \epsilon$	$N.s = \text{shrink}(N.i)$
$B \rightarrow \text{text}$	$B.ht = \text{text.h} \times B.ps$



EQN: 自下而上计算的实现

产生式	语义规则
$S \rightarrow LB$	$B.ps = L.s; S.ht = B.ht$
$L \rightarrow \epsilon$	$L.s = 10$ 将 $B.ps$ 存入栈中, 便于引用
$B \rightarrow B_1 MB_2$	$B_1.ps = B.ps; M.i = B.ps;$ $B_2.ps = M.s; B.ht = \max(B_1.ht, B_2.ht)$
$M \rightarrow \epsilon$	$M.s = M.i$ 单纯为了属性位置可预测
$B \rightarrow B_1 \text{ sub } NB_2$	$B_1.ps = B.ps; N.i = B.ps;$ $B_2.ps = N.s; B.ht = \text{disp}(B_1.ht, B_2.ht)$
$N \rightarrow \epsilon$	$N.s = \text{shrink}(N.i)$
$B \rightarrow \text{text}$	$B.ht = \text{text.h} \times B.ps$



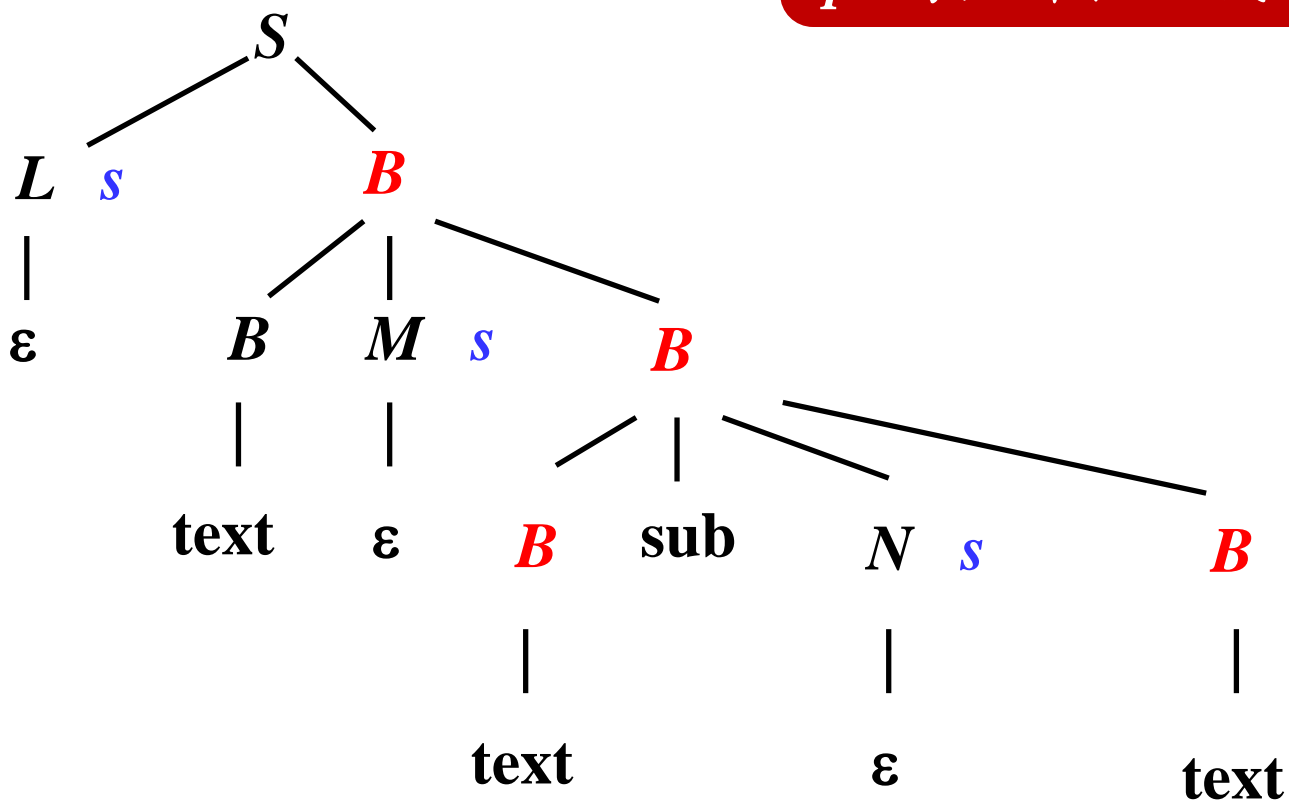
EQN: 自下而上计算的实现

产生式	语义规则
$S \rightarrow LB$	$B.ps = L.s; S.ht = B.ht$
$L \rightarrow \epsilon$	$L.s = 10$ 将 $B.ps$ 存入栈中, 便于引用
$B \rightarrow B_1 MB_2$	$B_1.ps = B.ps; M.i = B.ps;$ $B_2.ps = M.s; B.ht = \max(B_1.ht, B_2.ht)$
$M \rightarrow \epsilon$	$M.s = M.i$ 单纯为了属性位置可预测
$B \rightarrow B_1 \text{ sub } NB_2$	$B_1.ps = B.ps; N.i = B.ps;$ $B_2.ps = N.s; B.ht = \text{disp}(B_1.ht, B_2.ht)$
$N \rightarrow \epsilon$	$N.s = \text{shrink}(N.i)$ 兼有计算功能
$B \rightarrow \text{text}$	$B.ht = \text{text.h} \times B.ps$



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
$S \rightarrow LB$	$B.ps = L.s; S.ht = B.ht$
$L \rightarrow \varepsilon$	$L.s = 10$
$B \rightarrow B_1 MB_2$	$B_1.ps = B.ps; M.i = B.ps;$ $B_2.ps = M.s; B.ht = \max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1 \text{ sub } NB_2$	$B_1.ps = B.ps; N.i = B.ps;$ $B_2.ps = N.s; B.ht = \text{disp}(B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = \text{shrink}(N.i)$
$B \rightarrow \text{text}$	$B.ht = \text{text.h} \times B.ps$

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
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$L.s = 10$
$B \rightarrow B_1 MB_2$	$B_1.ps = B.ps; M.i = B.ps;$ $B_2.ps = M.s; B.ht = \max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1 \text{ sub } NB_2$	$B_1.ps = B.ps; N.i = B.ps;$ $B_2.ps = N.s; B.ht = \max(B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow \text{text}$	$B.ht = \text{text}.h \times B.ps$

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



EQN: 自下而上计算的实现

产生式	语义规则
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1 MB_2$	$B_1.ps = B.ps; M.i = B.ps;$ $B_2.ps = M.s; B.ht = \max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1 \text{ sub } NB_2$	$B_1.ps = B.ps; N.i = B.ps;$ $B_2.ps = N.s; B.ht = \text{disp}(B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = \text{shrink}(N.i)$
$B \rightarrow \text{text}$	$B.ht = \text{text}.h \times B.ps$

$L.s = 10$



EQN: 自下而上计算的实现

产生式	语义规则
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1 MB_2$	$val[top-2] = \max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1 \text{ sub } NB_2$	$B_1.ps = B.ps; N.i = B.ps;$ $B_2.ps = N.s; B.ht = \max(B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$
$B \rightarrow \text{text}$	$B.ht = \text{text}.h \times B.ps$

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



EQN: 自下而上计算的实现

产生式	语义规则
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1 MB_2$	$val[top-2] = \max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1 \text{ sub } NB_2$	$B_1.ps = B.ps; N.i = B.ps;$ $B_2.ps = N.s; B.ht = \text{disp}(B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = \text{shrink}(N.i)$
$B \rightarrow \text{text}$	$B.ht = \text{text}.h \times B.ps$

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



EQN: 自下而上计算的实现

产生式	语义规则
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1 MB_2$	$val[top-2] = \max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1 \text{ sub } NB_2$	$val[top-3] = \text{disp}(val[top-3], val[top])$
$N \rightarrow \varepsilon$	$N.s = \text{shrink}(N.i)$
$B \rightarrow \text{text}$	$B.ht = \text{text}.h \times B.ps$

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



EQN: 自下而上计算的实现

产生式	语义规则
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1 MB_2$	$val[top-2] = \max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1 \text{ sub } NB_2$	$val[top-3] = \text{disp}(val[top-3], val[top])$
$N \rightarrow \varepsilon$	$val[top+1] = \text{shrink}(val[top-2])$
$B \rightarrow \text{text}$	$B.ht = \text{text}.h \times B.ps$

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



EQN: 自下而上计算的实现

产生式	语义规则
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1 MB_2$	$val[top-2] = \max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1 \text{ sub } NB_2$	$val[top-3] = \text{disp}(val[top-3], val[top])$
$N \rightarrow \varepsilon$	$val[top+1] = \text{shrink}(val[top-2])$
$B \rightarrow \text{text}$	$val[top] = val[top] \times val[top-1]$

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



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

下期预告：语义分析