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

USTC, CHINA
ADSLAB
先进数据系统实验室

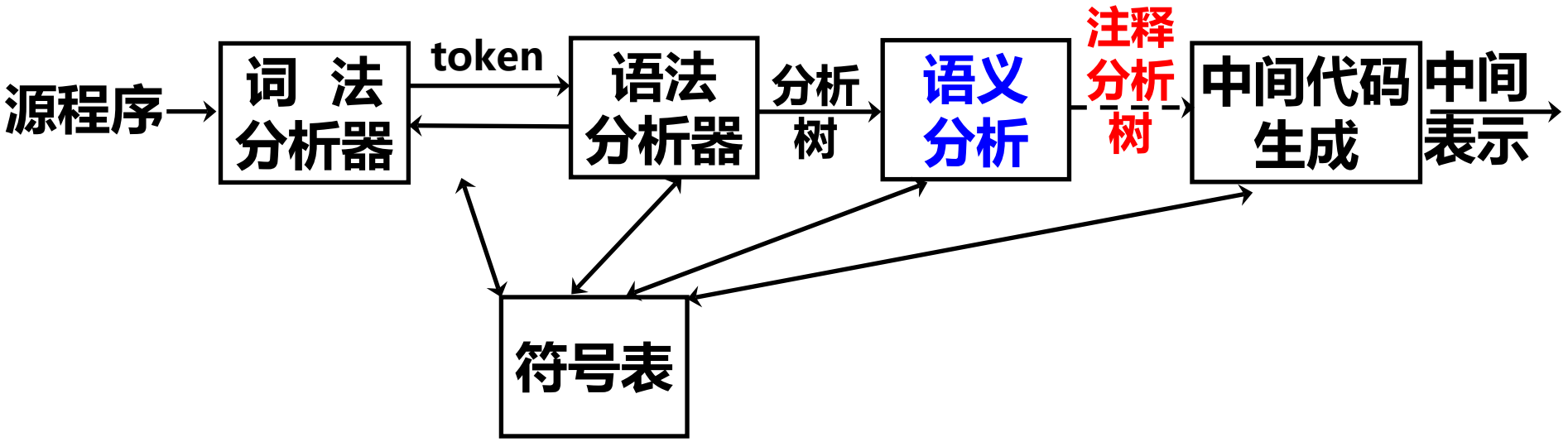
《编译原理与技术》
语法制导翻译 II

计算机科学与技术学院

李 诚

2021-10-20

主要内容



- 从语法制导定义到翻译方案
  - S属性定义的SDT
  - L属性定义的SDT

语法制导翻译方案

- 语法制导翻译方案(SDT)是在产生式右部中嵌入了程序片段(称为语义动作)的CFG
- SDT可以看作是SDD的具体实施方案
  - 通过建立语法分析树的方案
  - 在语法分析过程中，边分析边计算的方案
    - 与LR或者LL分析方法结合

将S-SDD转换为SDT

- 将一个S-SDD转换为SDT的方法：
  - 将每个语义动作都放在产生式的最后
  - 称为“后缀翻译方案”

S-SDD

产生式	语义规则
(1) $L \rightarrow E$ n	$L.val = E.val$
(2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
(3) $E \rightarrow T$	$E.val = T.val$
(4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
(5) $T \rightarrow F$	$T.val = F.val$
(6) $F \rightarrow (E)$	$F.val = E.val$
(7) $F \rightarrow$ digit	$F.val =$ digit.$lexval$

SDT

(1) $L \rightarrow E$ n { $L.val = E.val$}
(2) $E \rightarrow E_1 + T$ {$E.val = E_1.val + T.val$}
(3) $E \rightarrow T$ { $E.val = T.val$}
(4) $T \rightarrow T_1 * F$ { $T.val = T_1.val \times F.val$}
(5) $T \rightarrow F$ { $T.val = F.val$}
(6) $F \rightarrow (E)$ { $F.val = E.val$}
(7) $F \rightarrow$ digit { $F.val =$ digit.$lexval$}

S-属性定义的SDT实现-1

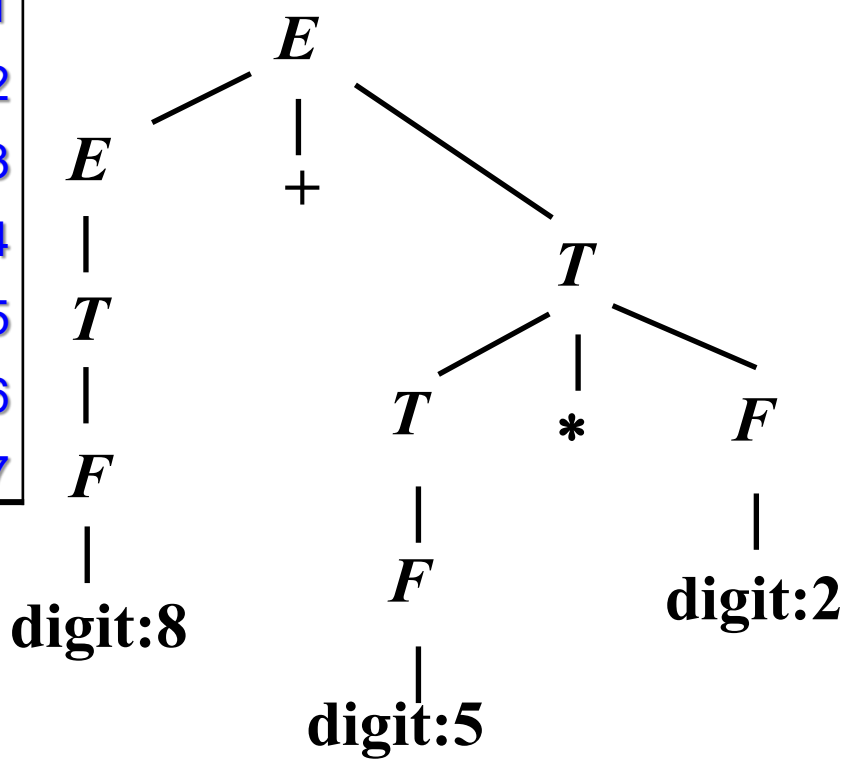
中国科学技术大学
University of Science and Technology of China

- 基于分析树的语法制导翻译方案
  - 建立语法分析树
  - 将语义动作看作是虚拟结点
  - 从左到右、深度优先地遍历分析树，在访问虚拟结点时执行相应的动作

S-属性定义的SDT实现-1

❑基于分析树的语法制导翻译方案

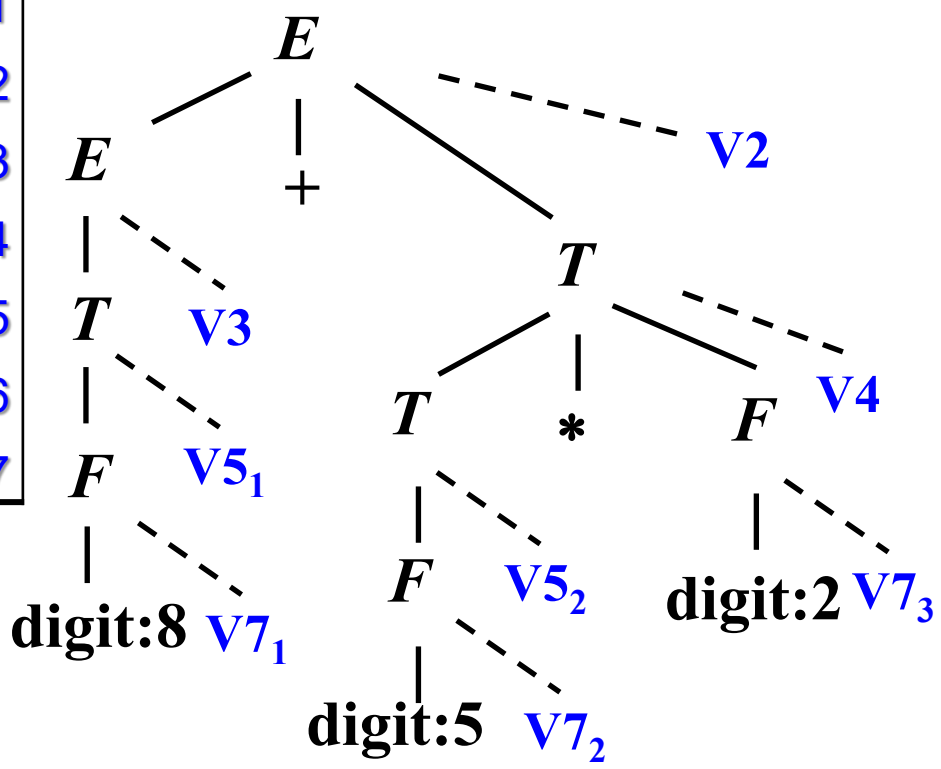
$L \rightarrow E$ n	{*print* (*E.val*) }	V1
$E \rightarrow E_1 + T$	{*E.val* = E_1 .*val* + *T.val* }	V2
$E \rightarrow T$	{*E.val* = *T.val* }	V3
$T \rightarrow T_1 * F$	{*T.val* = T_1.*val* * *F.val* }	V4
$T \rightarrow F$	{*T.val* = *F.val* }	V5
$F \rightarrow (E)$	{*F.val* = *E.val* }	V6
$F \rightarrow$ digit	{*F.val* = digit.*lexval*}	V7



S-属性定义的SDT实现-1

❑基于分析树的语法制导翻译方案

$L \rightarrow E$ n	{*print* (*E.val*) }	V1
$E \rightarrow E_1 + T$	{*E.val* = E_1 .*val* + *T.val* }	V2
$E \rightarrow T$	{*E.val* = *T.val* }	V3
$T \rightarrow T_1 * F$	{*T.val* = T_1.*val* * *F.val* }	V4
$T \rightarrow F$	{*T.val* = *F.val* }	V5
$F \rightarrow (E)$	{*F.val* = *E.val* }	V6
$F \rightarrow$ digit	{*F.val* = digit.*lexval*}	V7



- 语句8+5*2的分析树如右
- 深度优先可知动作执行顺序
  - $V7_1, V5_1, V3, V7_2, V5_2, V7_3, V4, V2$

S-属性定义的SDT实现-2

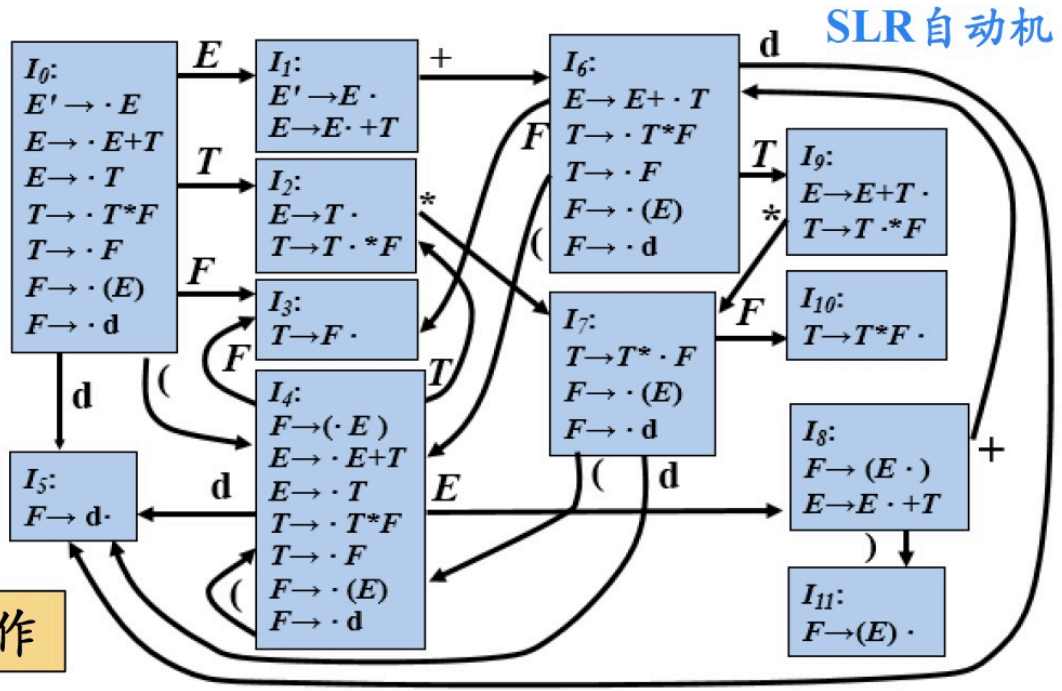
- 综合属性可通过自底向上的LR方法来计算
- 当归约发生时执行相应的语义动作

➢ 例

S-SDD

产生式	语义规则
(1) $L \to E$ n	$L.val = E.val$
(2) $E \to E_1 + T$	$E.val = E_1.val + T.val$
(3) $E \to T$	$E.val = T.val$
(4) $T \to T_1 * F$	$T.val = T_1.val \times F.val$
(5) $T \to F$	$T.val = F.val$
(6) $F \to (E)$	$F.val = E.val$
(7) $F \to$ digit	$F.val =$ digit.*lexval*

当归约发生时执行相应的语义动作



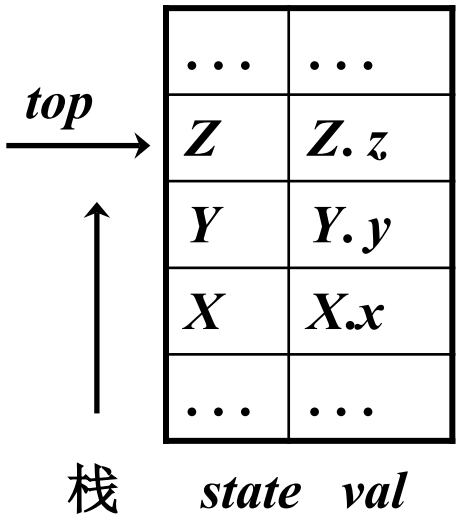
S-属性定义的SDT实现-2

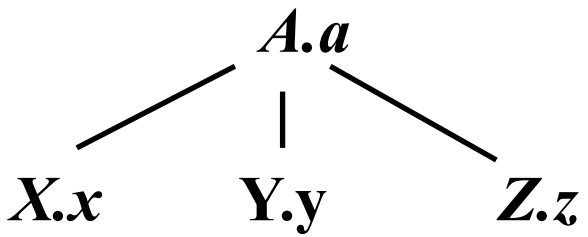
- 可以通过扩展的LR语法分析栈来实现
  - 在分析栈中使用一个附加的域来存放综合属性值。若支持多个属性，那么可以在栈中存放指针
  - 每一个栈元素包含状态、文法符号、综合属性三个域
    - 也可以将分析栈看成三个平行的栈，分别是状态栈、文法符号栈、综合属性栈，分开看的理由是，入栈出栈并不完全同步
  - 语义动作将修改为对栈中文法符号属性的计算

S-属性定义的SDT实现-2

❑可以通过扩展的LR语法分析栈来实现

❖考虑产生式 $A \rightarrow XYZ$



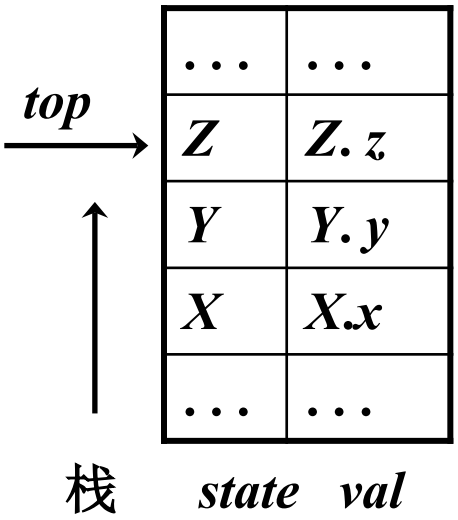


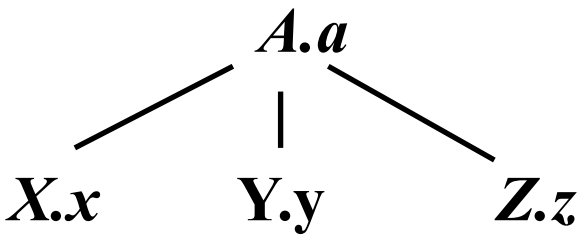
$A \rightarrow XYZ \{A.a = f(X.x, Y.y, Z.z)\}$

S-属性定义的SDT实现-2

❑可以通过扩展的LR语法分析栈来实现

❖考虑产生式 $A \rightarrow XYZ$





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

语义动作

state[top-2] = *A*

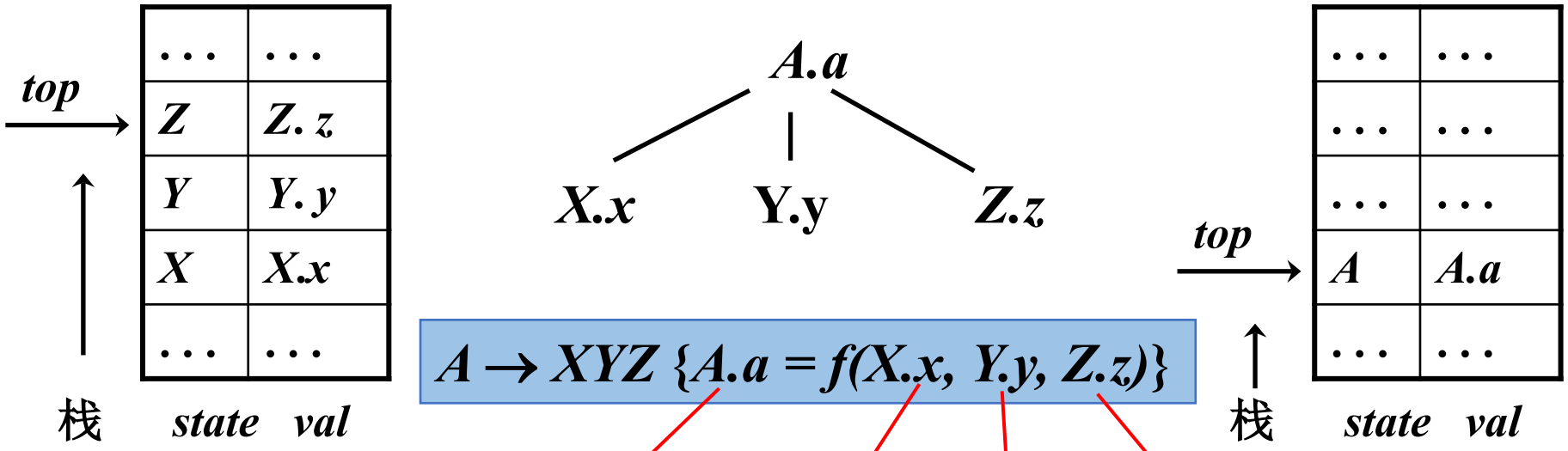
val[top-2] = *f*(*val*[top-2], *val*[top-1], *val*[top])

top = *top*-2

S-属性定义的SDT实现-2

❑可以通过扩展的LR语法分析栈来实现

❖考虑产生式 $A \rightarrow XYZ$



语义动作

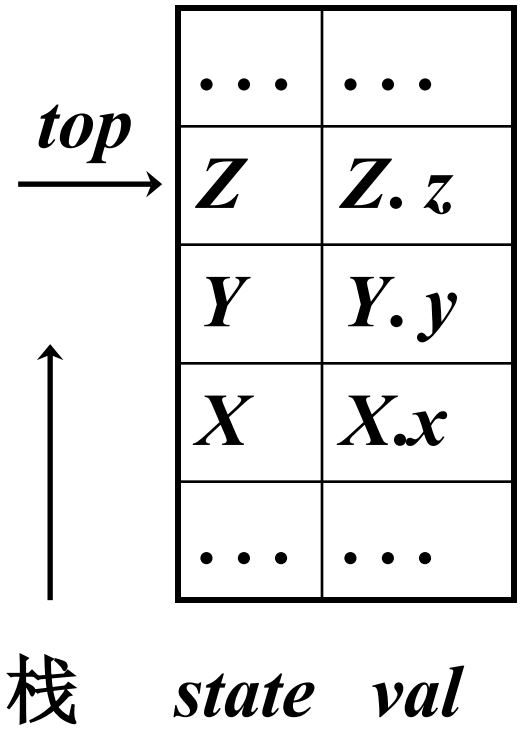
state[top-2] = *A*

val[top-2] = *f*(*val*[top-2], *val*[top-1], *val*[top])

top = *top*-2

SLR分析栈中实现计算器

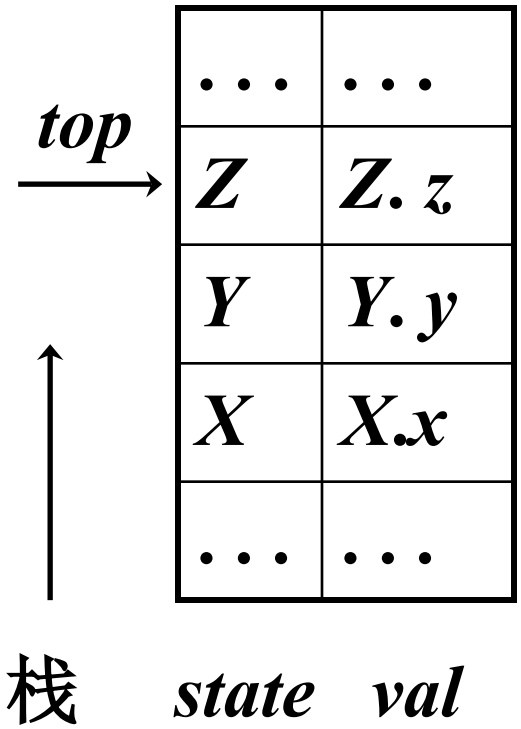
❑简单计算器的语法制导定义改成栈操作代码



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*

SLR分析栈中实现计算器

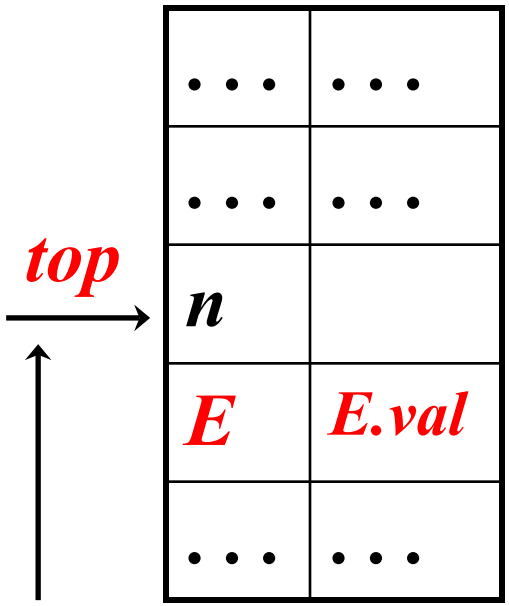
❑简单计算器的语法制导定义改成栈操作代码



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*

SLR分析栈中实现计算器

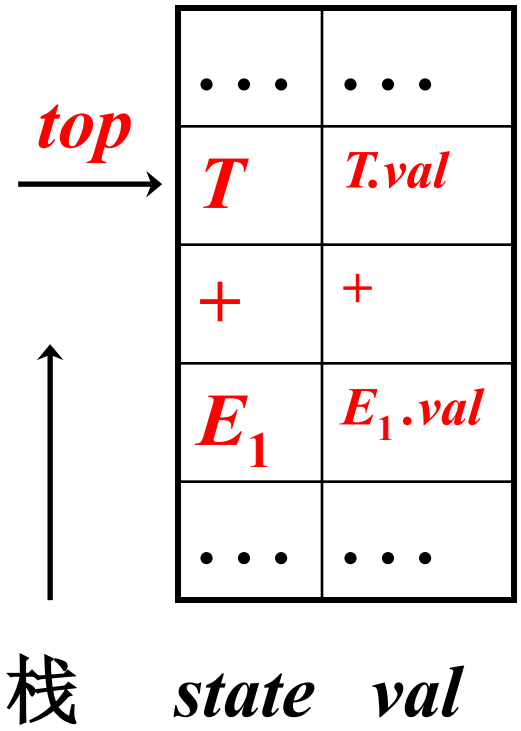
❑简单计算器的语法制导定义改成栈操作代码



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*

SLR分析栈中实现计算器

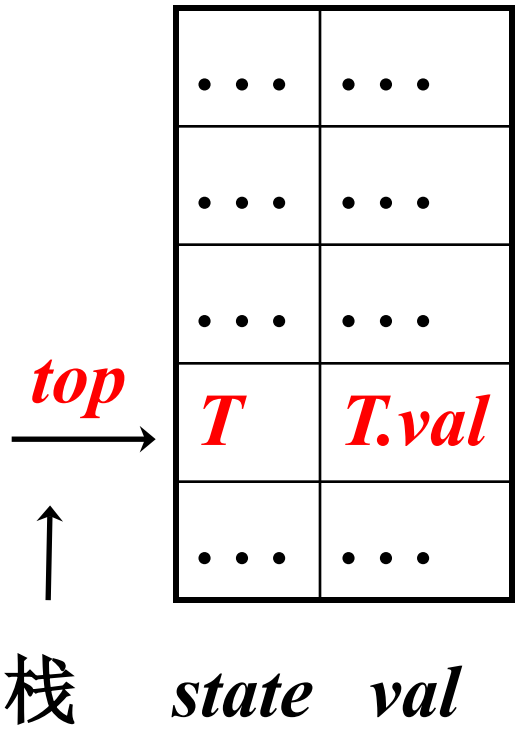
❑简单计算器的语法制导定义改成栈操作代码



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*

SLR分析栈中实现计算器

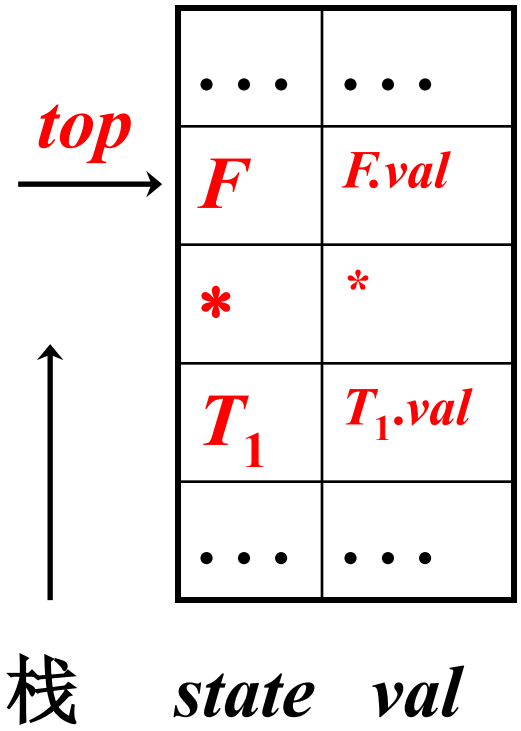
❑简单计算器的语法制导定义改成栈操作代码



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*]
$E \rightarrow T$	值不变，无动作
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow$ digit	$F.val =$ digit.*lexval*

SLR分析栈中实现计算器

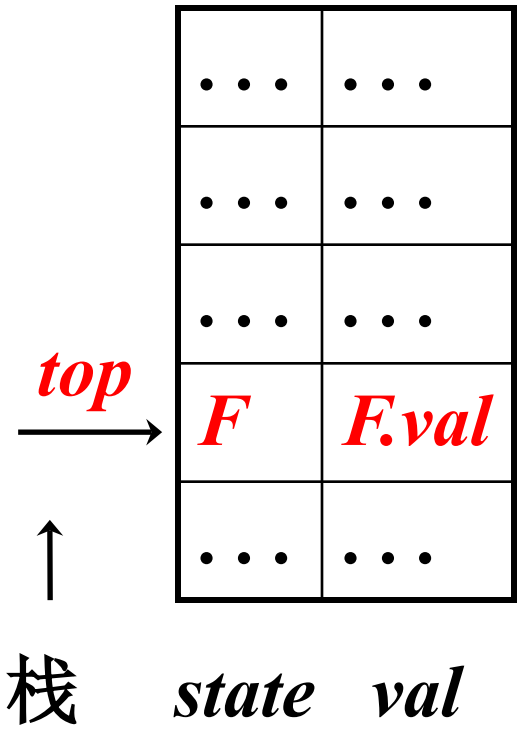
❑简单计算器的语法制导定义改成栈操作代码



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]×*val* [*top*]
$T \rightarrow F$	*T.val* = *F.val*
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*

SLR分析栈中实现计算器

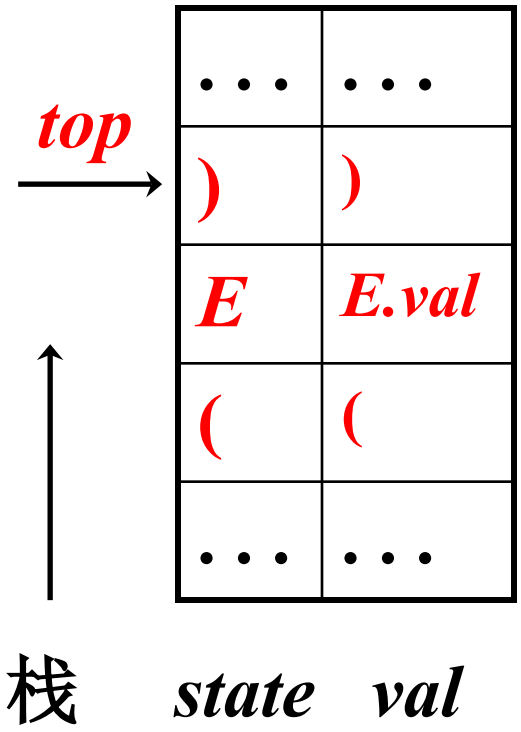
❑简单计算器的语法制导定义改成栈操作代码



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*]
$E \rightarrow T$	值不变，无动作
$T \rightarrow T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \rightarrow F$	值不变，无动作
$F \rightarrow (E)$	*F.val* = *E.val*
$F \rightarrow$ digit	*F.val* = digit.*lexval*

SLR分析栈中实现计算器

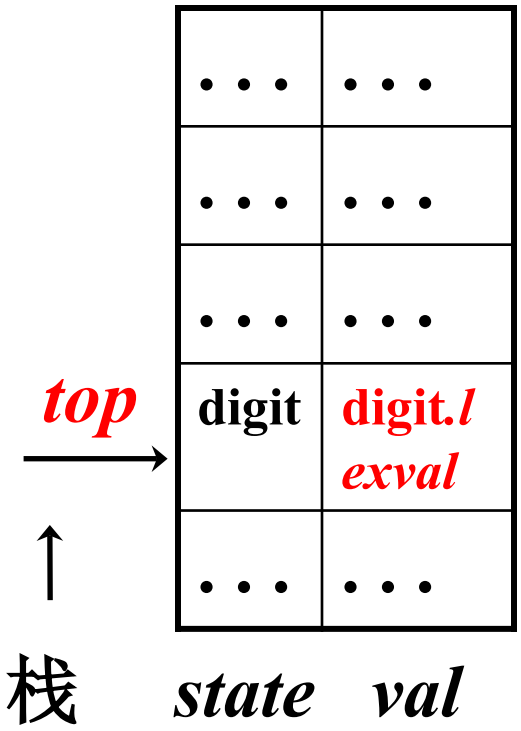
❑简单计算器的语法制导定义改成栈操作代码



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*]
$E \rightarrow T$	值不变，无动作
$T \rightarrow T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \rightarrow F$	值不变，无动作
$F \rightarrow (E)$	*val* [top –2] =*val* [top –1]
$F \rightarrow$ digit	*F.val* = digit.*lexval*

SLR分析栈中实现计算器

❑简单计算器的语法制导定义改成栈操作代码

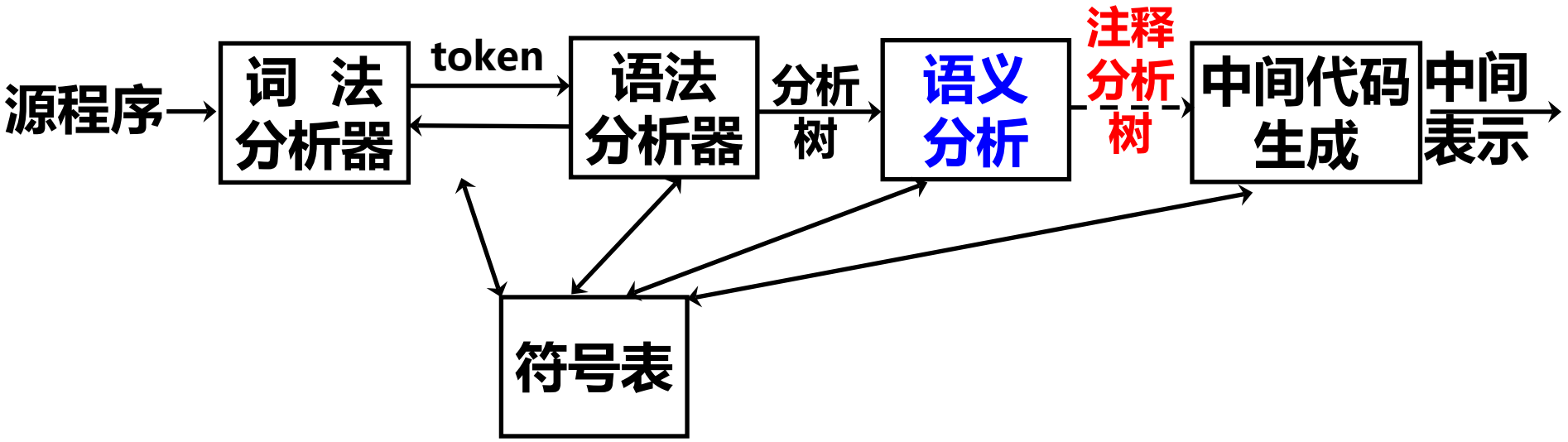


产 生 式	代 码 段
$L \rightarrow E$ n	*print* (*val* [*top*–1])
$E \rightarrow E_1 + T$	*val* [*top* –2] = *val* [*top* –2]+*val* [*top*]
$E \rightarrow T$	值不变，无动作
$T \rightarrow T_1 * F$	*val* [*top* –2] = *val* [*top* –2]×*val* [*top*]
$T \rightarrow F$	值不变，无动作
$F \rightarrow (E)$	*val* [top –2] =*val* [top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

总结

- 采用自底向上分析，例如LR分析，首先给出S-属性定义，然后，把S-属性定义变成可执行的代码段，放到产生式尾部，这就构成了翻译程序。
- 随着语法分析的进行，归约前调用相应的语义子程序，完成翻译的任务。

主要内容



- 从语法制导定义到翻译方案
  - S属性定义的SDT
  - L属性定义的SDT

L属性定义的SDT

- 边分析边翻译的方式能否用于继承属性?
  - 属性的计算次序一定受分析方法所限定的分析树结点建立次序的限制
  - 分析树的结点是自左向右生成
  - 如果属性信息是自左向右流动，那么就有可能在分析的同时完成属性计算

L属性定义的SDT

- 如果每个产生式$A \rightarrow X_1 \ldots X_{j-1} X_j \ldots X_n$的每条语义规则计算的属性是$A$的综合属性；或者是$X_j$的继承属性，但它仅依赖：
  - 该产生式中X_j左边符号$X_1, X_2, \ldots, X_{j-1}$的属性；
  - A的继承属性
  - X_j的其他属性，且不能成环

- S属性定义属于L属性定义

L属性定义的SDT

- 变量类型声明的语法制导定义是一个*L-SDD*

产 生 式	语 义 规 则
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow$ int	$T.type = integer$
$T \rightarrow$ real	$T.type = real$
$L \rightarrow L_1$, id	$L_1.in = L.in$; $addType$(id.$entry$, $L.in$)
$L \rightarrow$ id	$addType$(id.$entry$, $L.in$)

- 后缀SDT在这里并不适用

L属性定义的SDT

- 消除左递归的算术表达式语法制导定义是L-SDD

产 生 式	语 义 规 则
$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
$F \rightarrow$ digit	$F.val$ = digit.$lexval$

- 后缀SDT在这里并不适用

将L-SDD转换为SDT

❑将L-SDD转换为SDT的规则

- ❖将计算一个产生式左部符号的综合属性的动作放置在这个产生式右部的最右端
- ❖将计算某个非终结符号A的继承属性的动作插入到产生式右部中紧靠在A的本次出现之前的位置上
  - ➢多个继承属性，要考虑次序，防止形成环

将L-SDD转换为SDT-举例

L-SDD

产 生 式	语 义 规 则
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow$ int	$T.type = integer$
$T \rightarrow$ real	$T.type = real$
$L \rightarrow L_1$, id	$L_1.in = L.in$; $addType(\text{id}.entry, L.in)$
$L \rightarrow$ id	$addType(\text{id}.entry, L.in)$

将L-SDD转换为SDT-举例

L-SDD

产 生 式	语 义 规 则
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow$ int	$T.type = integer$
$T \rightarrow$ real	$T.type = real$
$L \rightarrow L_1$, id	$L_1.in = L.in$; $addType(\text{id}.entry, L.in)$
$L \rightarrow$ id	$addType(\text{id}.entry, L.in)$

L-SDT

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

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

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

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

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

将L-SDD转换为SDT-举例

L-SDD

产 生 式	语 义 规 则
$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
$F \rightarrow$ digit	$F.val =$ digit.$lexval$

将L-SDD转换为SDT-举例

L-SDD

产 生 式	语 义 规 则
$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow *FT_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
$F \rightarrow$ digit	$F.val$ = digit.$lexval$

L-SDT

$T \rightarrow F \{T'.inh = F.val\}\ T' \{T.val = T'.syn\}$

$T' \rightarrow *F \{T_1'.inh = T'.inh \times F.val\}\ T_1' \{T'.syn = T_1'.syn\}$

$T' \rightarrow \varepsilon \{T'.syn = T'.inh\}$

$F \rightarrow$ digit $\{F.val$ = digit.$lexval\}$

L属性定义的SDT-举例

❑例　把有加和减的中缀表达式翻译成后缀表达式输出

如果输入是8+5 –2，则输出是8 5 + 2 –，设计SDT

$E \rightarrow T\ R$

$R \rightarrow \text{addop}\ T\ R_1 \mid \varepsilon$

$T \rightarrow \text{num}$

L属性定义的SDT-举例

❑例 把有加和减的中缀表达式翻译成后缀表达式输出
如果输入是8+5 –2，则输出是8 5 + 2 –，设计SDT

$E \rightarrow T\ R$

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

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

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

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

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

… {*print*(8)}{*print*(5)}{*print*(+)}addop T{*print*(–)}R

… {*print*(8)}{*print*(5)}{*print*(+)}{*print*(2)}{*print*(–)}

L属性定义的SDT-举例

❑例　把有加和减的中缀表达式翻译成后缀表达式输出

如果输入是8+5 –2，则输出是8 5 + 2 –，设计SDT

$E \rightarrow T\ R$

$R \rightarrow \text{addop}\ T\ R_1$ {*print* (addop.*lexeme*)} | ε

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

语义动作不能随意放置

L属性定义的SDT-举例

❑例 把有加和减的中缀表达式翻译成后缀表达式输出
如果输入是8+5 –2，则输出是8 5 + 2 –，设计SDT

$E \rightarrow T\ R$

$R \rightarrow \text{addop}\ T\ R_1$ {*print* (addop.*lexeme*)} | ε

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

语义动作不能随意放置

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

$\Rightarrow \text{num}$ {*print* (8)} addop TR

$\Rightarrow \text{num}$ {*print*(8)} addop num {*print*(5)} R

… {*print*(8)} {*print*(5)} addop TR

… {*print*(8)} {*print*(5)} {*print*(2)} {*print*(–)} {*print*(+)}

L属性定义的SDT-举例

❑例 数学排版语言EQN，设计SDT，计算高度

E sub 1 .val

$S \rightarrow B$

$B \rightarrow B_1\ B_2$

$B \rightarrow B_1$ sub B_2

$B \rightarrow$ text

文字排版中的符号属性

S： *S.ht*，综合属性；待排公式的整体高度

B： *B.ps*，继承属性； 公式（文本）中字体的大小

B.ht，综合属性；公式排版高度

text： *text.h*，文本高度

文字排版中的符号属性

S： $S.ht$，综合属性；待排公式的整体高度

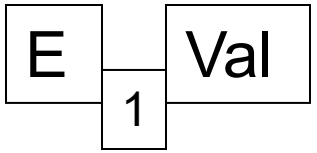
B： $B.ps$，继承属性； 公式（文本）中字体的大小
$B.ht$，综合属性；公式排版高度

$text$： $text.h$，文本高度

$max(B_1, B_2)$： 求两个排版公式的最大高度

$shrink(B)$： 将字体大小缩小为B的30%

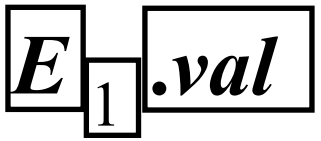
$disp(B_1.ht, B_2.ht)$： 向下调整B_2的位置



L属性定义的SDT-举例

❑例 数学排版语言EQN（语法制导定义L-SDD）

E sub 1 .val



ps-point size (继承属性); ht-height(综合属性)

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

L属性定义的SDT-举例

❑例 数学排版语言EQN（翻译方案SDT）

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

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

*B*继承属性的计算位于*B*的左边

L属性定义的SDT-举例

❑例 数学排版语言EQN（翻译方案SDT）

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

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

*B*综合属性的计算放在右部末端

L属性定义的SDT-举例

❑例 数学排版语言EQN（翻译方案SDT）

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

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

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

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

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

L属性定义的SDT-举例

❑例 数学排版语言EQN（翻译方案SDT）

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

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

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

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

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

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

B_1

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

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

L属性定义的SDT-举例

❑例 数学排版语言EQN（翻译方案SDT）

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

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

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

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

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

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

B_1

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

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

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

L属性定义的SDT-举例

❑例 翻译while循环语句，生成代码

产生式 $S \rightarrow \text{while } (C)\ S_1$

L-SDD

$S \rightarrow$ while (C) S_1	L1 = new(); L2 = new(); S_1.next = L1; C.false = S.next; C.true = L2; S.code = **label** ‖ L1 ‖ C.code ‖ **label** ‖ L2 ‖ S_1.code

- ***next:*** 继承属性，语句结束后应跳转到的标号
- ***true、false:*** C为真/假时应该跳转到的标号
- ***code:*** 综合属性，表示代码

L属性定义的SDT-举例

L-SDD

$S \rightarrow$ while (C) S_1	L1 = new();	临时变量
	L2 = new();	
S_1的继承属性 →	S_1.next = L1;	
	C.false = S.next;	C的继承属性
	C.true = L2;	
S的综合属性 →	S.code = **label** ‖ L1 ‖ C.Ccode ‖ **label** ‖ L2 ‖ S_1.code	

❑根据语义动作的放置规则得到如下SDT：

L-SDT

$S \rightarrow$ while (	{L1 = new(); L2 = new(); C.false = S.next; C.true = L2;}
C)	{S_1.next = L1;}
S_1	{S.code = **label** ‖ L1 ‖ C.Ccode ‖ **label** ‖ L2 ‖ S_1.code}

L属性定义的计算

- 结合SDT，考虑在语法分析过程中进行翻译
  - 自顶向下计算
    - 递归下降分析器
  - 自底向上计算，考虑与LR分析器的结合
    - 删除翻译方案中嵌入的动作
    - 继承属性在分析栈中的计算

L属性定义的自顶向下计算

- 递归下降翻译器的设计
  - 为每个非终结符A构造一个函数
    - A的每个继承属性对应该函数的一个形参
    - 函数的返回值是A的综合属性值

L属性定义的自顶向下计算

❑递归下降翻译器的设计

- ❖为每个非终结符A构造一个函数
  - ➢A的每个继承属性对应该函数的一个形参
  - ➢函数的返回值是A的综合属性值
- ❖在函数体中
  - ➢首先选择适当的A的产生式
  - ➢用局部变量保存产生式中文法符号的属性
  - ➢对产生式体中的终结符号，读入符号并获取其综合属性（由词法分析得到）
  - ➢对产生式体中的非终结符，调用相应函数，记录返回值

L属性定义计算-递归下降

$E \rightarrow T$	$\{R.i = T.nptr\}$	$T + T + T + \ldots$
R	$\{E.nptr = R.s\}$	
$R \rightarrow +$		
T	$\{R_1.i = mkNode(\text{'+'}, R.i, T.nptr)\}$	
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	$\{W_1.i = mkNode(\text{'*'}, W.i, F.nptr)\}$	
W_1	$\{W.s = W_1.s\}$	
$W \rightarrow \varepsilon$	$\{W.s = W.i\}$	

F 产生式部分不再给出

文法是LL(1)，所以适合自顶向下分析

L属性定义计算-递归下降

产生式$R \rightarrow +TR \mid \varepsilon$ 的递归下降分析过程

```
void R( ) {
  if (lookahead == '+' ) {
        match ( '+' );
        T( );
        R( );
  }
  else  /* 什么也不做 */
}
```

L属性定义计算-递归下降


```
syntaxTreeNode* R (syntaxTreeNode* i) {
  syntaxTreeNode *nptr, *i1, *s1, *s;
  char addoplexeme;

  if (lookahead == '+' ) { /* 产生式 R → +T R  */
        addoplexeme = lexval;
        match('+' );
        nptr = T();
        i1 = mkNode(addoplexeme, i , nptr);
        s1 = R (i1);
        s = s1;
  }
  else  s = i;  /* 产生式 R → ε  */
  return s;
}
```

R : i, s

T : $nptr$

$+$: $addoplexeme$

L属性定义的计算

- 结合SDT，考虑在语法分析过程中进行翻译
  - 自顶向下计算
    - 递归下降分析器
  - 自底向上计算，考虑与LR分析器的结合
    - 删除翻译方案中嵌入的动作
    - 继承属性在分析栈中的计算

L属性定义的自底向上计算

- 可否直接将带有继承属性的L-SDD直接与LR结合起来?
- 考虑$A \to BC$产生式，假设$B.i = A.i$
  - 当对B进行归约时，由于没有看到C，不能确定会使用该产生式，$A.i$无法得知，此时可以推迟计算
  - 等按照BC归约成为A之后，我们仍然不能确定包含了A的产生式，$A.i$仍无法得知，继续推迟计算
  - 最终退化为“先造分析树，后翻译”的策略
- 解决方案:
  - 可将产生式中嵌入的动作删除，挪到产生式最右端

L属性定义的自底向上计算

- 将L-SDD转换为SDT
- 对于产生式$A \to \alpha \{a\} \beta$，a是语义动作
  - 引入新的非终结符M，代替$\{a\}$，形成$A \to \alpha M \beta$
  - 引入新的产生式 $M \to \varepsilon$
  - 修改a得到a'：
    - 将a需要的A或者α中的属性作为M的继承属性进行复制
    - 按照a中的方法计算各属性，将这些属性作为M的综合属性保存起来
  - 将$\{a'\}$与$M \to \varepsilon$关联起来

L属性定义的自底向上计算

- 假设LL文法对应的SDT有如下片段

$$A \rightarrow \{B.i = f(A.i)\}BC$$

- 修改后的SDT为

$$A \rightarrow M\ B\ C$$

$$M \rightarrow \varepsilon\ \{M.i = A.i;\ M.s = f(M.i)\}$$

分析栈的设计可以保证M的语义规则中可以使用A.i，因为A.i在M下方紧邻的位置。

❑例：删除翻译方案中嵌入的动作

$E \rightarrow T\ R$

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

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

这些动作的一个重要特点：
没有引用原来产生式文法符号的属性，即只涉及虚拟属性

L属性定义的自底向上计算

❑例：删除翻译方案中嵌入的动作

$E \rightarrow T\ R$

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

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

加入产生ε的标记非终结符，让每个嵌入动作由不同标记非终结符M代表，并把该动作放在产生式$M \rightarrow \varepsilon$的右端

$E \rightarrow T\ R$

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

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

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

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

这些动作的一个重要特点：
没有引用原来产生式文法符号的属性，即只涉及虚拟属性

L属性定义的计算

- 结合SDT，考虑在语法分析过程中进行翻译
  - 自顶向下计算
    - 递归下降分析器
    - LL分析器
  - 自底向上计算，考虑与LR分析器的结合
    - 删除翻译方案中嵌入的动作
    - 继承属性在分析栈中的计算

L属性定义的自底向上计算

❑例 数学排版语言EQN

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

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

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

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

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

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

B_1

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

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

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

文字排版引入标记符号

为了自底向上的计算:

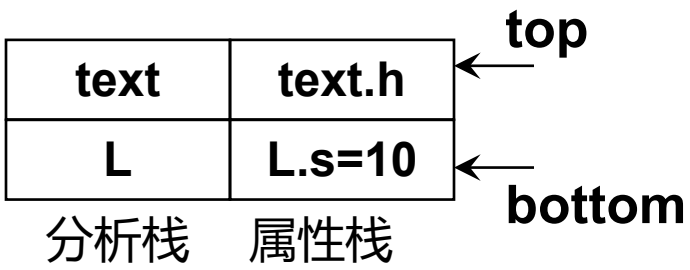
B→ text　　{ B.ht := text.h × B.ps }

必须确定继承属性B.ps的（“属性栈”）位置。为引入标记非终结符L、M和N及其属性，包括相应的空产生式和有关属性规则。这样B.ps即可在紧靠“句柄”text下方的位置上找到。(L的综合属性置为B.ps的初值)

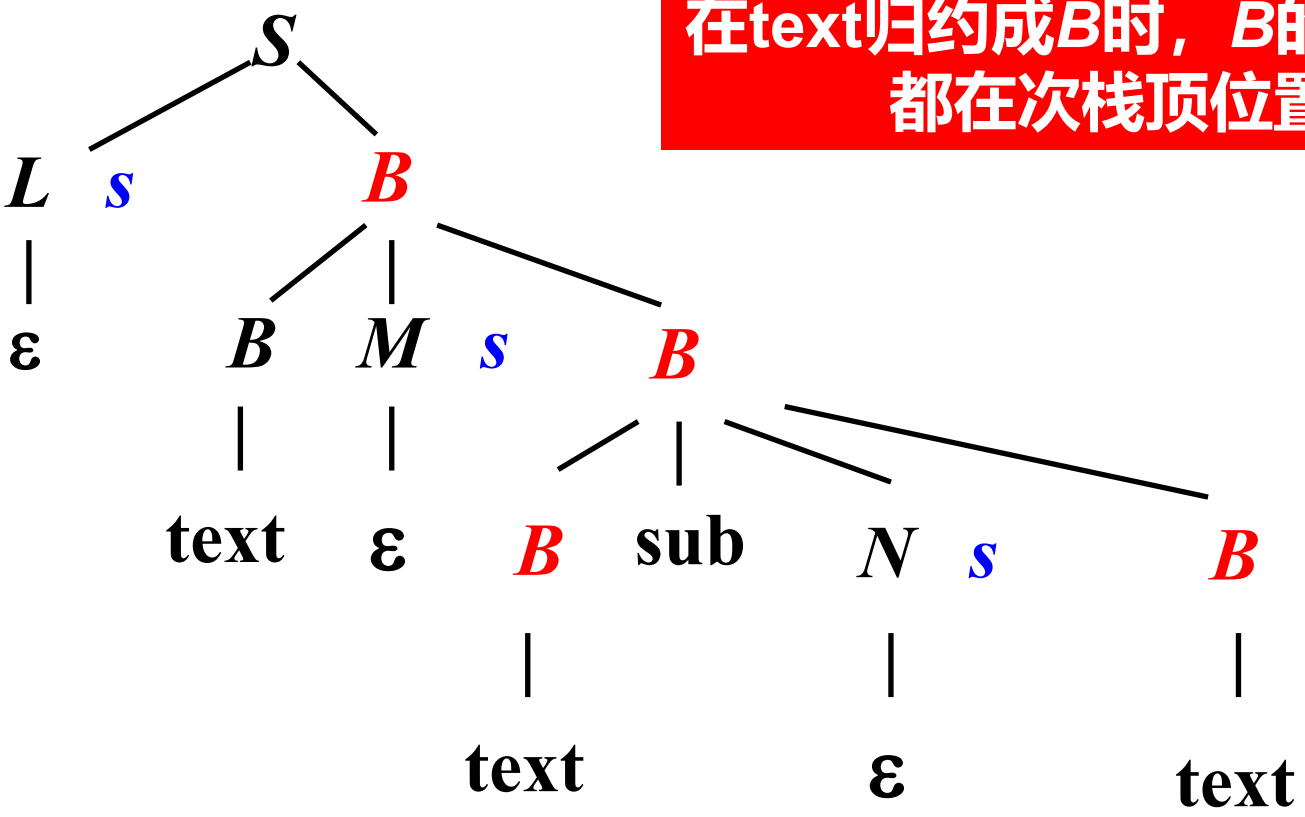
S→L B

B→B_1M B_2

B→B_1 sub N B_2



L属性定义的自底向上计算



在text归约成B时，B的ps属性都在次栈顶位置

L属性定义的自底向上计算

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
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps$; $M.i = B.ps$; $B_2.ps = M.s$;$B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ 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.$h \times B.ps$

L属性定义的自底向上计算

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.$h \times B.ps$

L属性定义的自底向上计算

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	$B_1.ps = B.ps$; $N.i = B.ps$; $B_2.ps = N.s$; $B.ht = disp\ (B_1.ht, B_2.ht)$
$N \rightarrow \varepsilon$	$N.s = shrink(N.i)$ 兼有计算功能
$B \rightarrow$ text	$B.ht =$ text.$h \times B.ps$

L属性定义的自底向上计算

产 生 式	语 义 规 则
$S \rightarrow LB$	$B.ps = L.s$; $S.ht = B.ht$
$L \rightarrow \varepsilon$	$L.s = 10$
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps$; $M.i = B.ps$; $B_2.ps = M.s$; $B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ 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.$h \times B.ps$

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

L属性定义的自底向上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$L.s = 10$
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps;\ \ M.i = B.ps;$ $B_2.ps = M.s; B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ 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.$h \times B.ps$

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

L属性定义的自底向上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps;\ \ M.i = B.ps;$ $B_2.ps = M.s; B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ 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	$B.ht = \text{text}.h \times B.ps$

L.s = 10

L属性定义的自底向上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$B_1.ps = B.ps;\ \ M.i = B.ps;$ $B_2.ps = M.s; B.ht = max(B_1.ht, B_2.ht)$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ 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.$h \times B.ps$

归约时，弹出0个元素，随后将L压栈，因此top为top+1

L.s = 10

L属性定义的自底向上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$M.s = M.i$
$B \rightarrow B_1$ 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	$B.ht = \text{text}.h \times B.ps$

B1.ps= B.ps; M.i = B.ps;B2.ps= M.s;B.ht= max(B1.ht, B2.ht)

L属性定义的自底向上计算

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

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

L属性定义的自底向上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1$ 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$.h \times B.ps$

B1.ps=B.ps; N.i = B.ps;B2.ps= N.s; B.ht= disp(B1.ht,B2.ht)

L属性定义的自底向上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1$ 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	$B.ht = \text{text}.h \times B.ps$

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

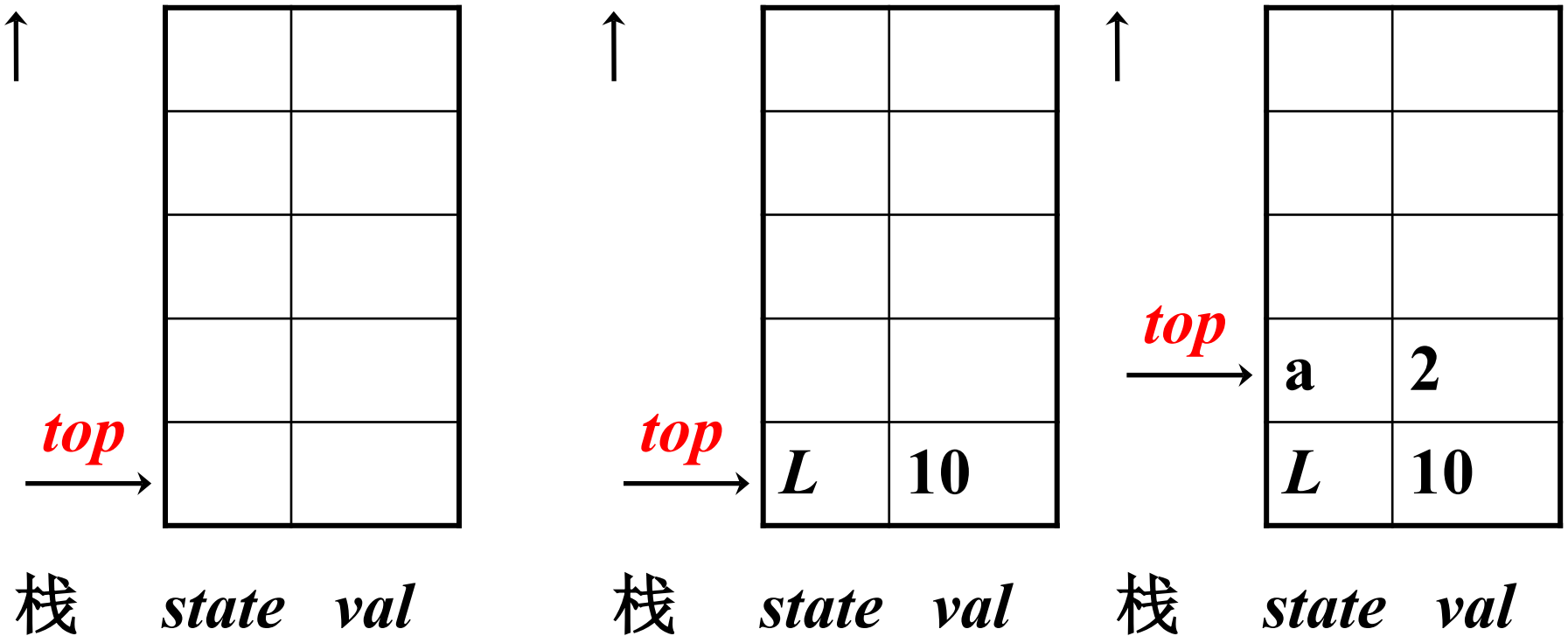
L属性定义的自底向上计算

产 生 式	代 码 段
$S \rightarrow LB$	$val[top-1] = val[top]$
$L \rightarrow \varepsilon$	$val[top+1] = 10$
$B \rightarrow B_1\ MB_2$	$val[top-2] = max(val[top-2], val[top])$
$M \rightarrow \varepsilon$	$val[top+1] = val[top-1]$
$B \rightarrow B_1$ 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	$val[top] = val[top] \times val[top-1]$

B.ht= text.h × B.ps

举例说明

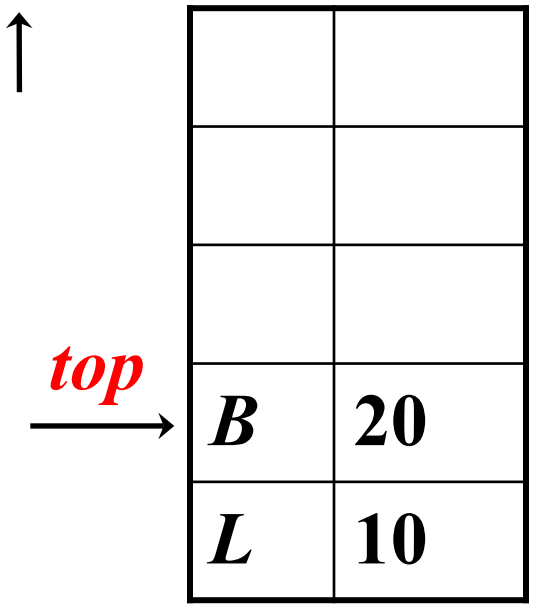
❑考虑字体大小10，基准高度2，输入串a sub 1



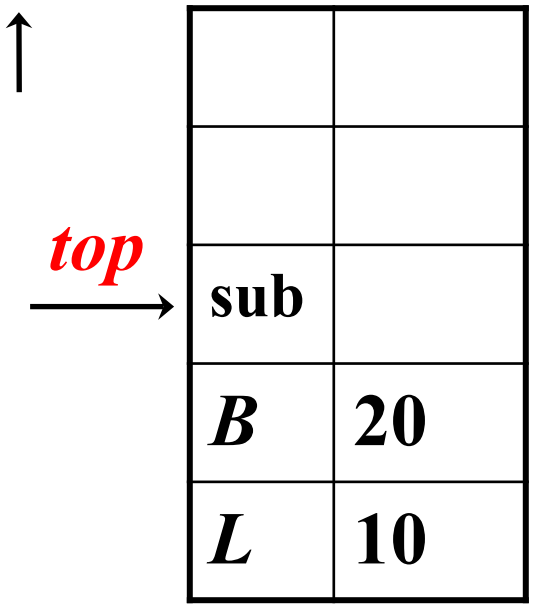
初始状态　　空归约　　移进

举例说明

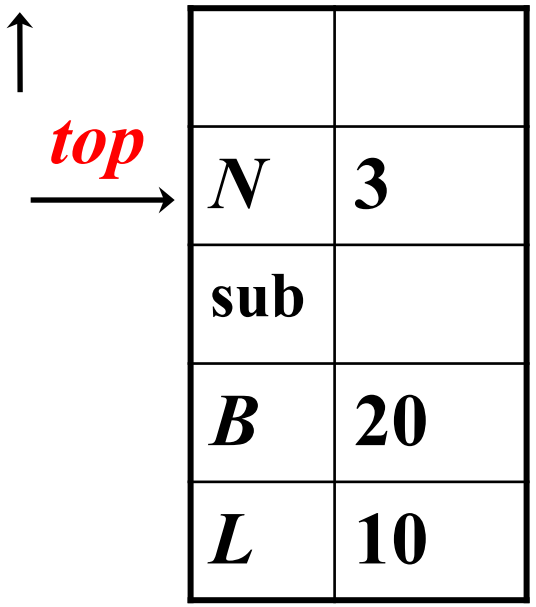
❑考虑字体大小10，基准高度2，输入串a sub 1



text归约



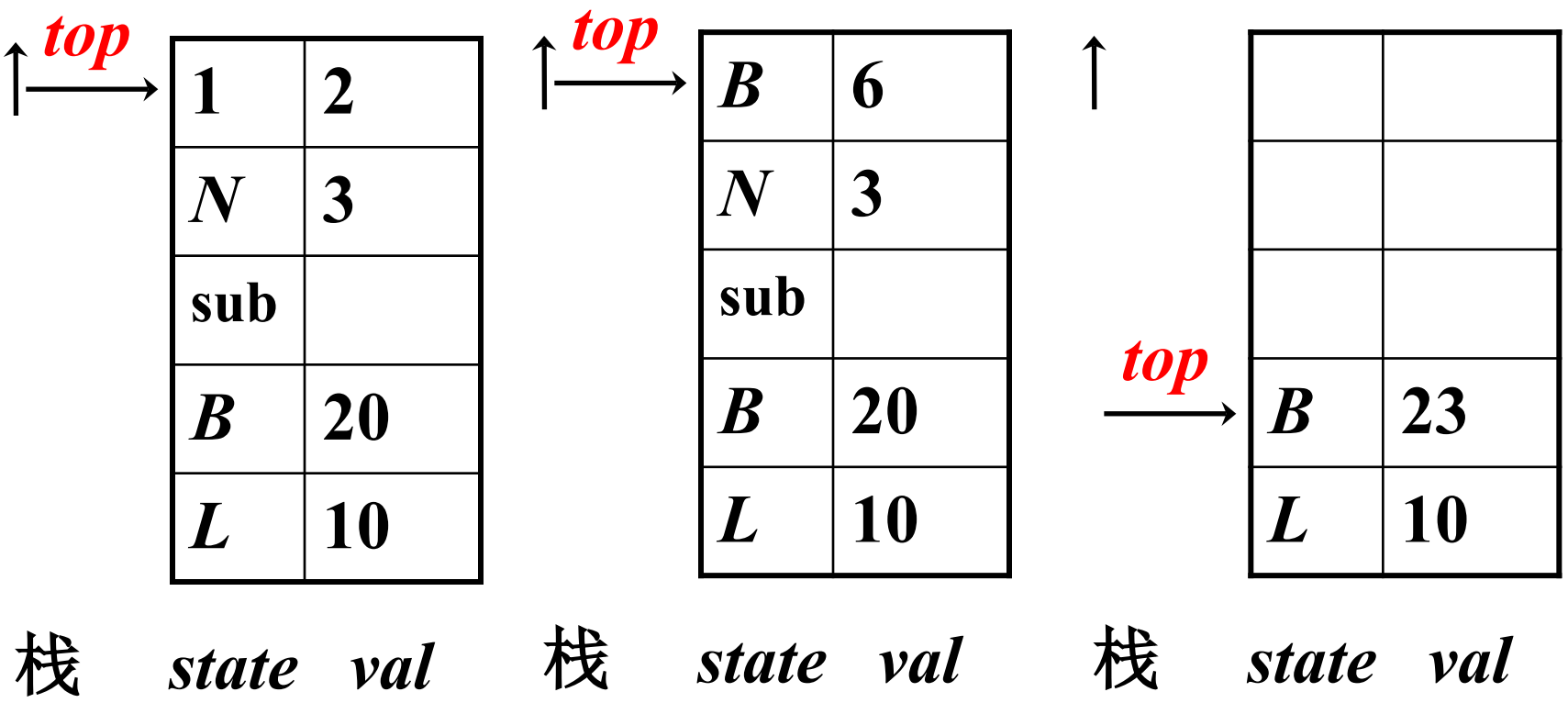
移进关键字



空归约，缩放字体

举例说明

❑考虑字体大小10，基准高度2，输入串a sub 1



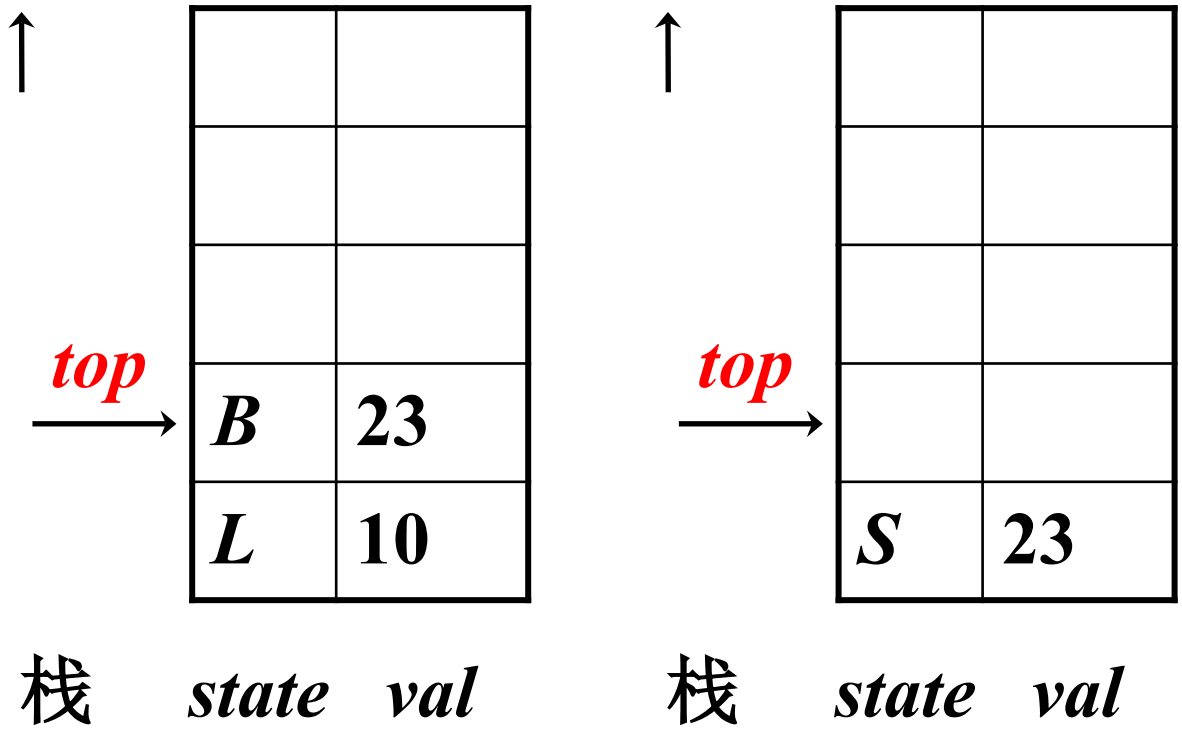
移进text

text归约，使用缩放后的字体大小

归约，重排高度
假设disp的结果为23

举例说明

❑考虑字体大小10，基准高度2，输入串a sub 1



本章要点

- 语义规则的两种描述方法：语法制导的定义和翻译方案
- 设计简单问题的语法制导定义和翻译方案，这是本章的重点和难点
- 语法制导定义和翻译方案的实现
  - 构造分析树
  - 与分析器结合

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

USTC, CHINA
ADSLAB
先进数据系统实验室

《编译原理与技术》
语法制导翻译 II

The End!