



语法制导翻译

Part3: 语法制导翻译方案

李 诚

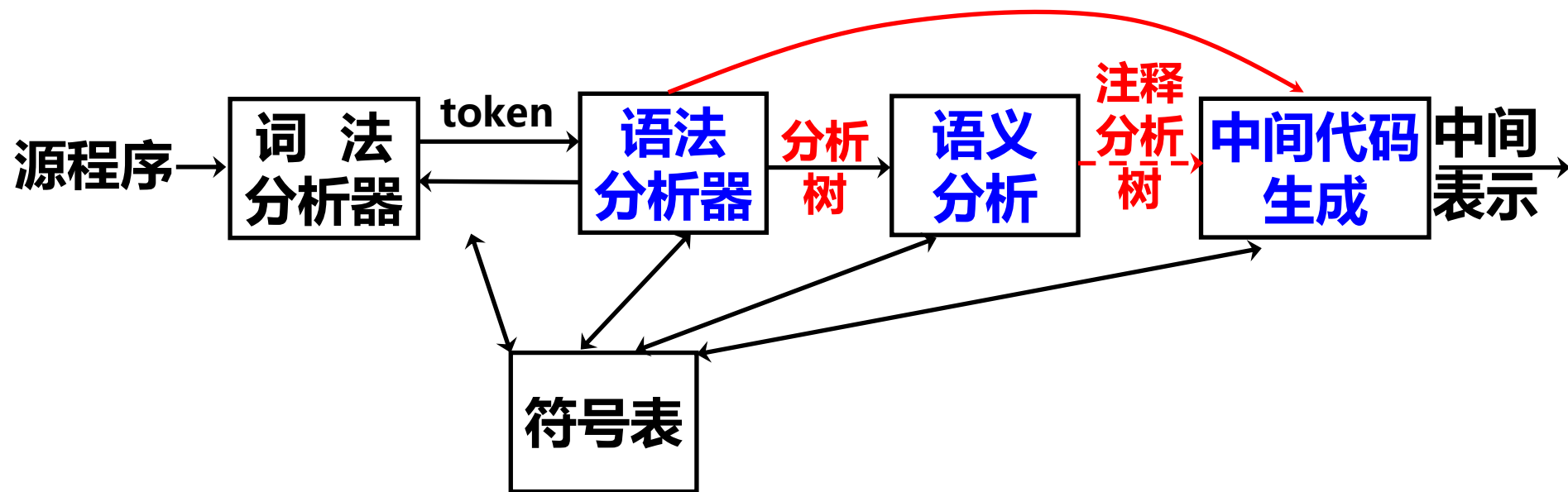
国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心

计算机科学与技术学院

2023年10月11日



本节提纲



- 从语法制导定义到翻译方案
- S属性定义的SDT
 - 实现方式1：先建树，后计算
 - 实现方式2：边分析，边计算



- **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

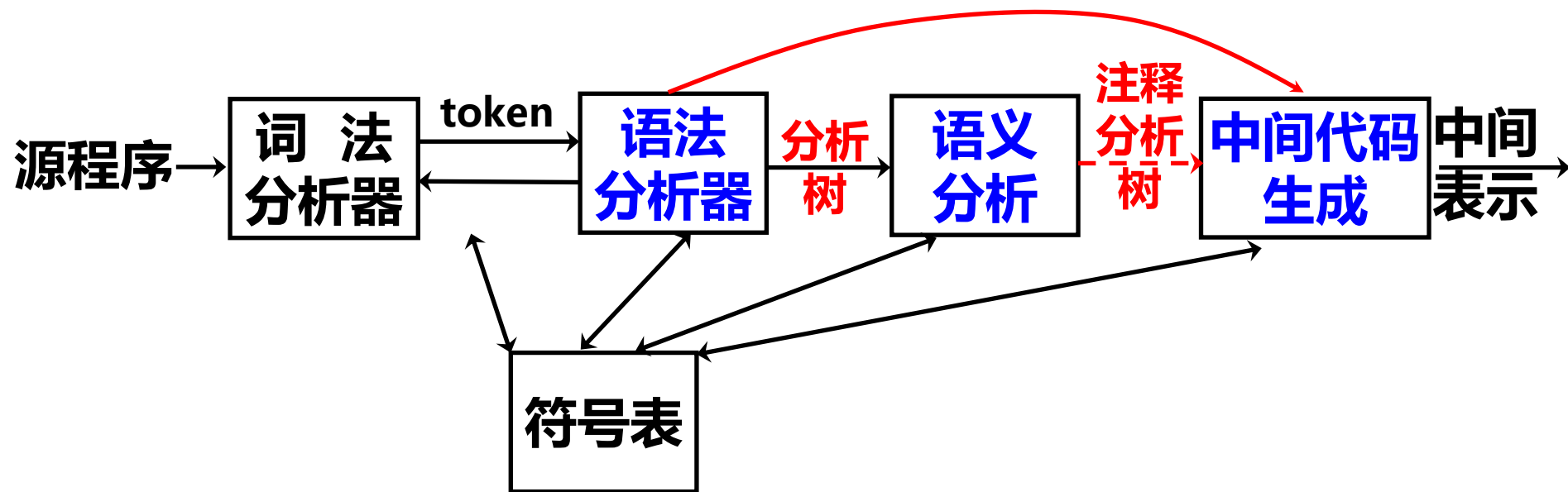
产生式	语义规则
(1) $L \rightarrow E \text{ n}$	$L.val = E.val$
(2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
(3) $E \rightarrow T$	$E.val = T.val$
(4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
(5) $T \rightarrow F$	$T.val = F.val$
(6) $F \rightarrow (E)$	$F.val = E.val$
(7) $F \rightarrow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

SDT

(1) $L \rightarrow E \text{ n} \{ L.val = E.val \}$
(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) $F \rightarrow \text{digit} \{ F.val = \text{digit.lexval} \}$



本节提纲



- 从语法制导定义到翻译方案
- S属性定义的SDT
 - 实现方式1：先建树，后计算
 - 实现方式2：边分析，边计算



- **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



• 基于分析树的语法制导翻译方案

$L \rightarrow E \mathbf{n}$ $\{ \text{print}(E.val) \}$ V1

$E \rightarrow E_1 + T$ $\{ E.val = \mathbf{E_1}.val + T.val \}$ V2

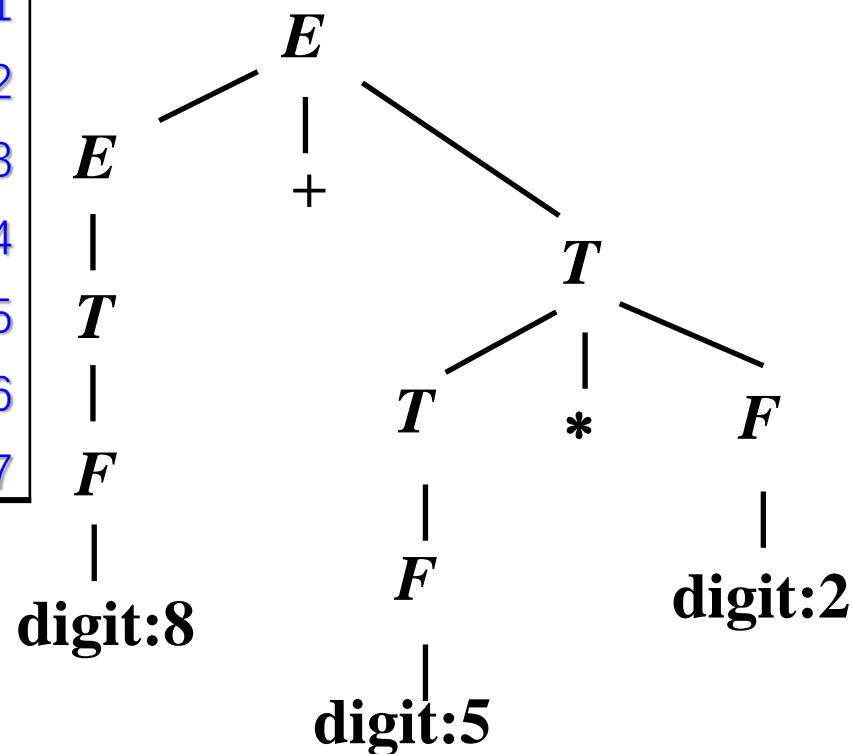
$E \rightarrow T$ $\{ E.val = T.val \}$ V3

$T \rightarrow T_1 * F$ $\{ T.val = T_1.val * F.val \}$ V4

$T \rightarrow F$ $\{ T.val = F.val \}$ V5

$F \rightarrow (E)$ $\{ F.val = E.val \}$ V6

$F \rightarrow \text{digit}$ $\{ F.val = \text{digit.lexval} \}$ V7





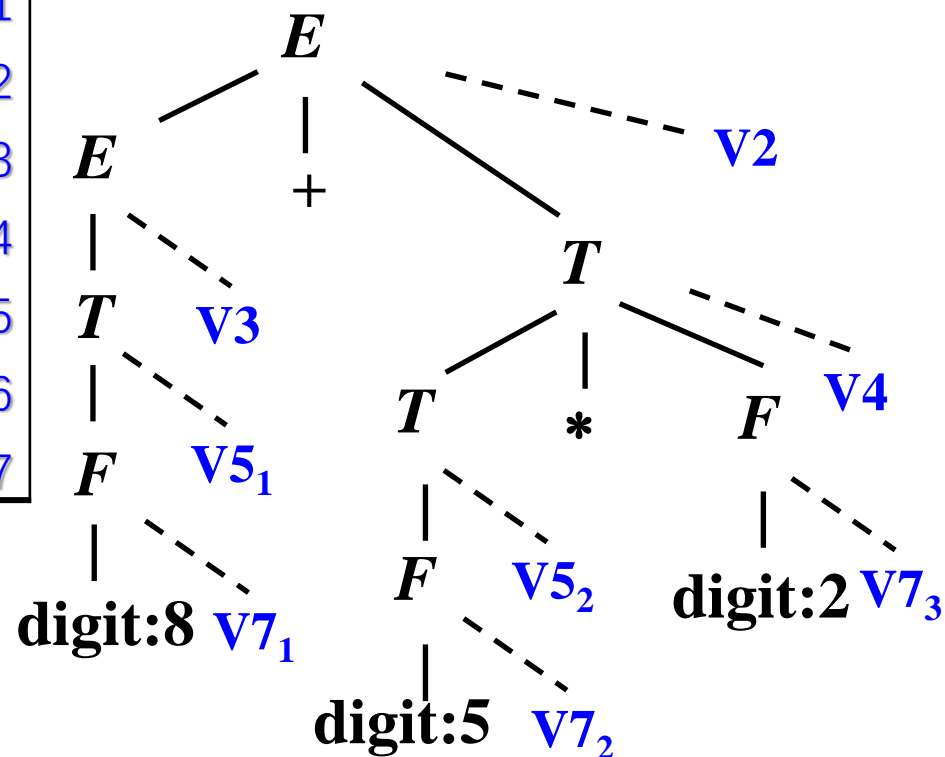
• 基于分析树的语法制导翻译方案

$L \rightarrow E \mathbf{n}$	$\{ \text{print}(E.val) \}$	V1
$E \rightarrow E_1 + T$	$\{ E.val = \mathbf{E_1}.val + T.val \}$	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
$F \rightarrow \text{digit}$	$\{ F.val = \text{digit.lexval} \}$	V7

• 语句 $8+5*2$ 的分析树如右

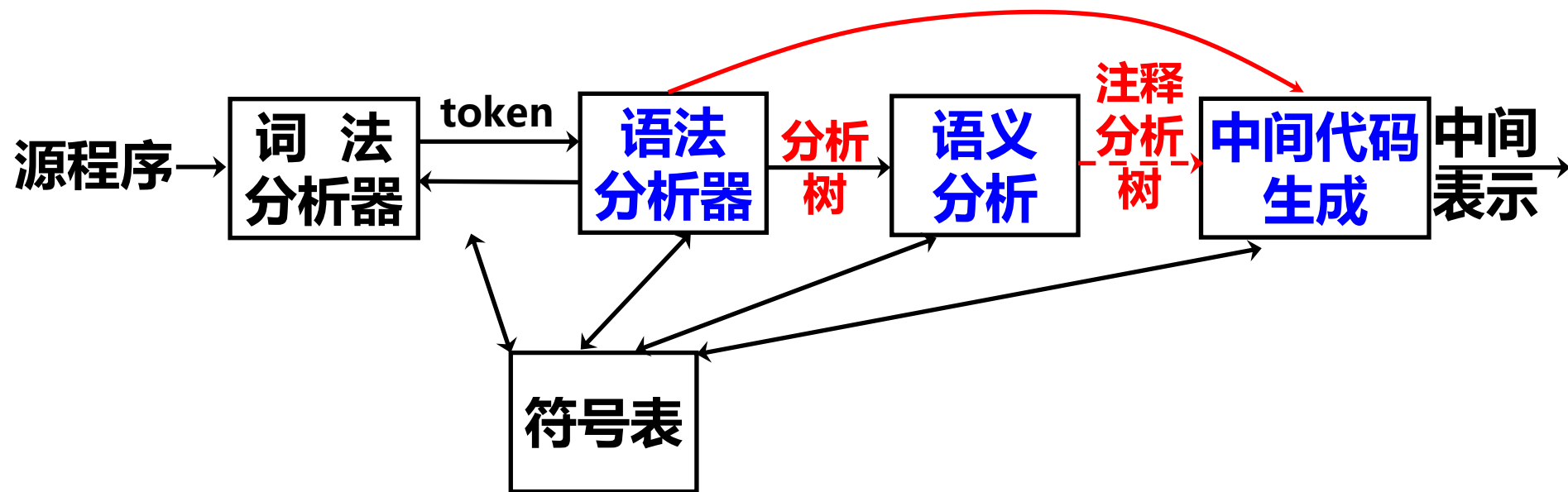
• 深度优先可知动作执行顺序

- $V7_1, V5_1, V3, V7_2, V5_2, V7_3, V4, V2$





本节提纲



- 从语法制导定义到翻译方案

- S属性定义的SDT

- 实现方式1：先建树，后计算
- 实现方式2：边分析，边计算



S-属性定义的SDT实现-2



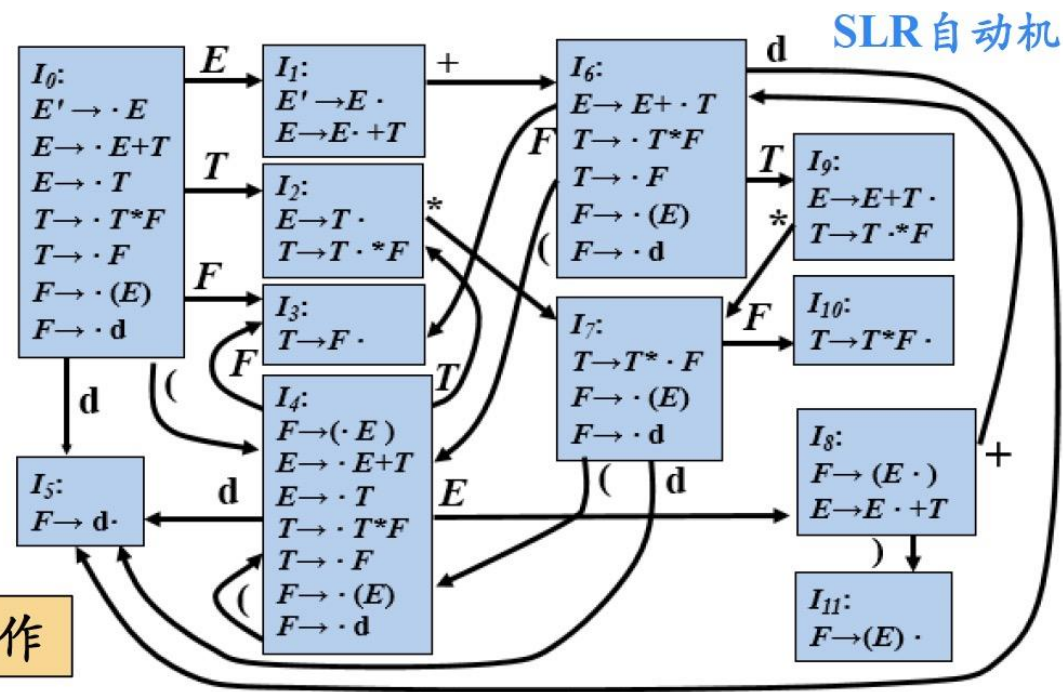
- 综合属性可通过自底向上的LR方法来计算
- 当归约发生时执行相应的语义动作

➤ 例

S-SDD

产生式	语义规则
(1) $L \rightarrow E \text{ n}$	$L.val = E.val$
(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) $F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

当归约发生时执行相应的语义动作





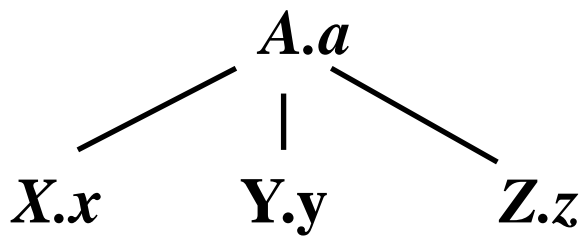
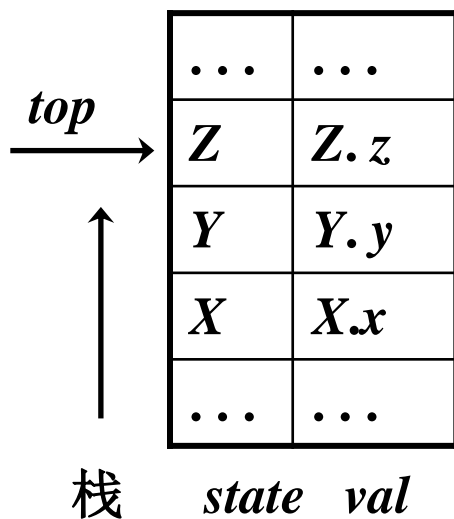
- **可以通过扩展的LR语法分析栈来实现**

- 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**



- 可以通过扩展的LR语法分析栈来实现

- 考虑产生式 $A \rightarrow XYZ$

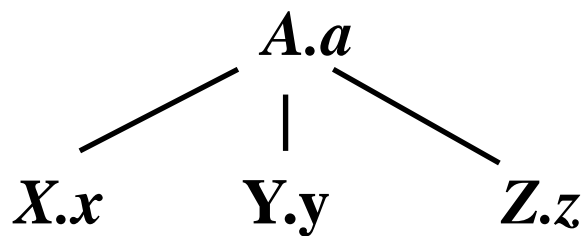
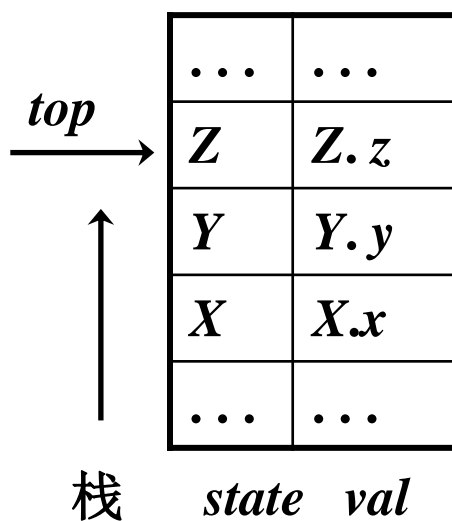


$$A \rightarrow XYZ \{A.a = f(X.x, Y.y, Z.z)\}$$



• 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

$state[top-2] = A$

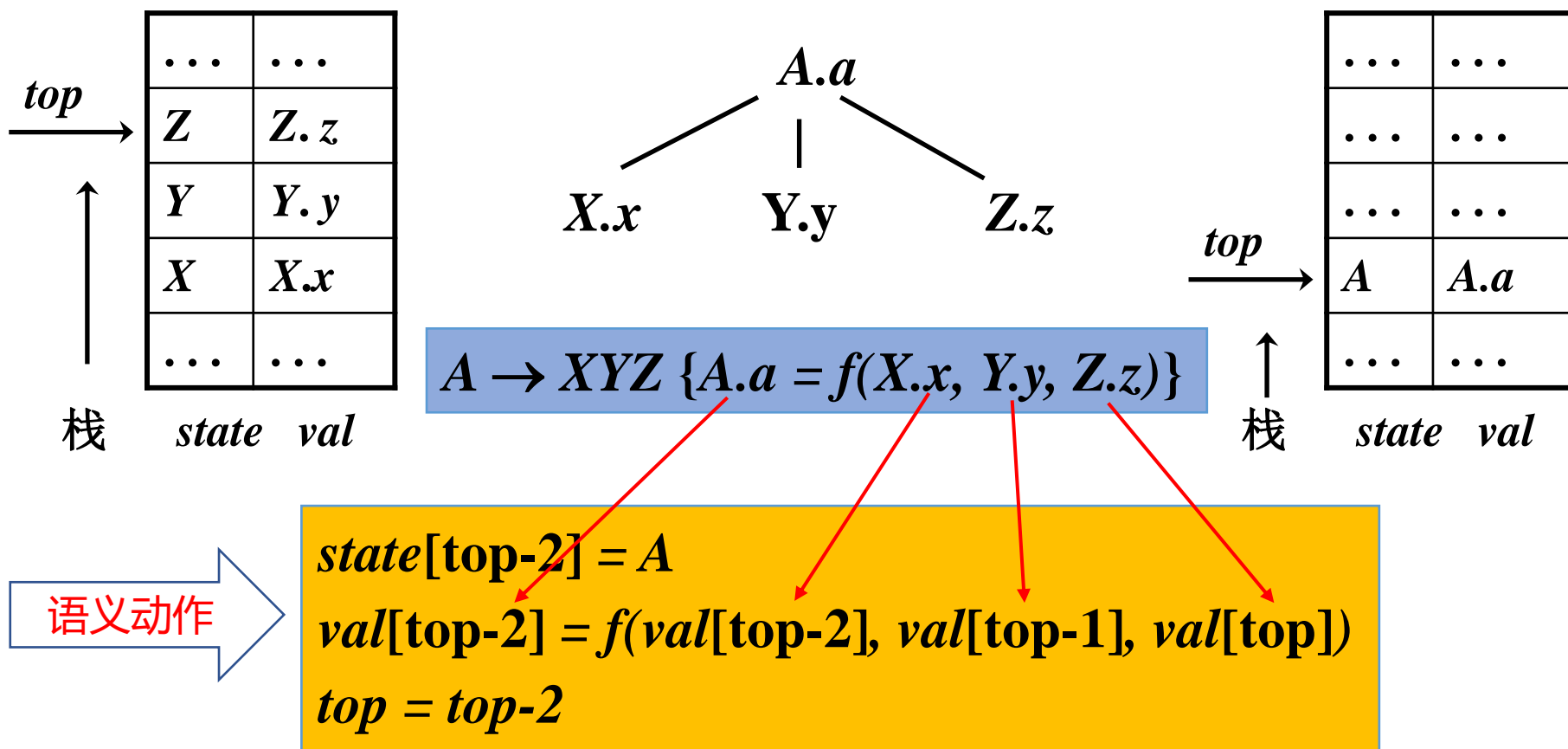
$val[top-2] = f(val[top-2], val[top-1], val[top])$

$top = top-2$



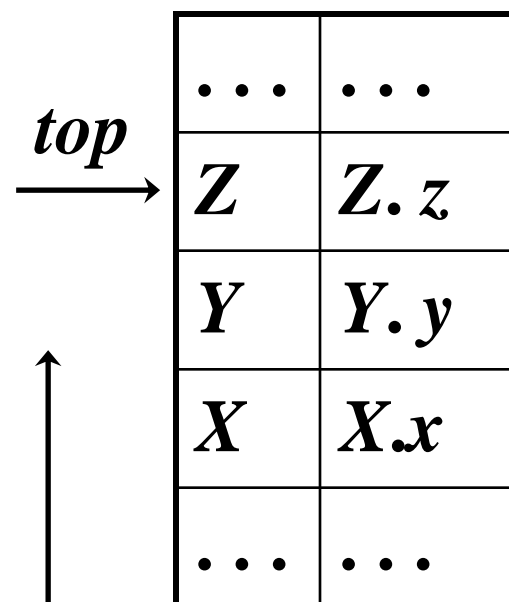
• 可以通过扩展的LR语法分析栈来实现

- 考虑产生式 $A \rightarrow XYZ$





- 简单计算器的语法制导定义改成栈操作代码

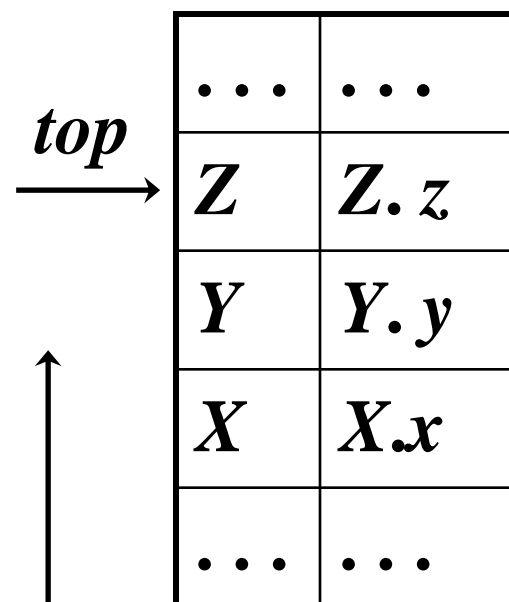


栈 *state* *val*

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



- 简单计算器的语法制导定义改成栈操作代码

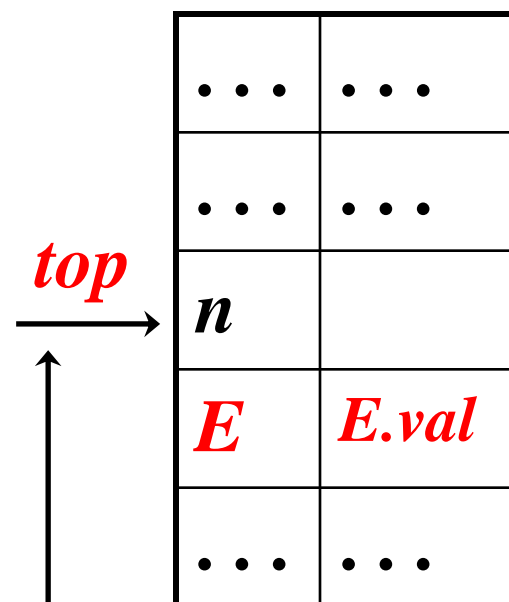


栈 *state* *val*

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



- 简单计算器的语法制导定义改成栈操作代码

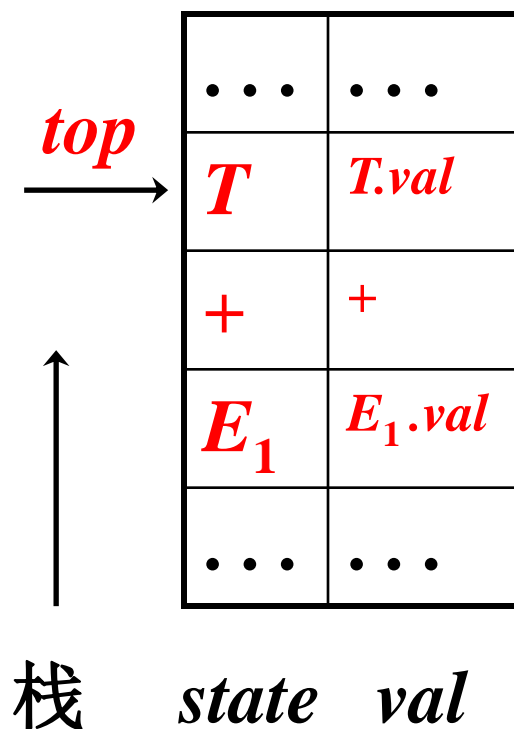


栈 *state* *val*

产生式	代码段
$L \rightarrow E n$	<code>print (<i>val</i> [<i>top</i>-1])</code>
$E \rightarrow E_1 + T$	<code><i>E.val</i> = <i>E</i>₁.<i>val</i> + <i>T.val</i></code>
$E \rightarrow T$	<code><i>E.val</i> = <i>T.val</i></code>
$T \rightarrow T_1 * F$	<code><i>T.val</i> = <i>T</i>₁.<i>val</i> * <i>F.val</i></code>
$T \rightarrow F$	<code><i>T.val</i> = <i>F.val</i></code>
$F \rightarrow (E)$	<code><i>F.val</i> = <i>E.val</i></code>
$F \rightarrow \text{digit}$	<code><i>F.val</i> = digit.<i>lexval</i></code>



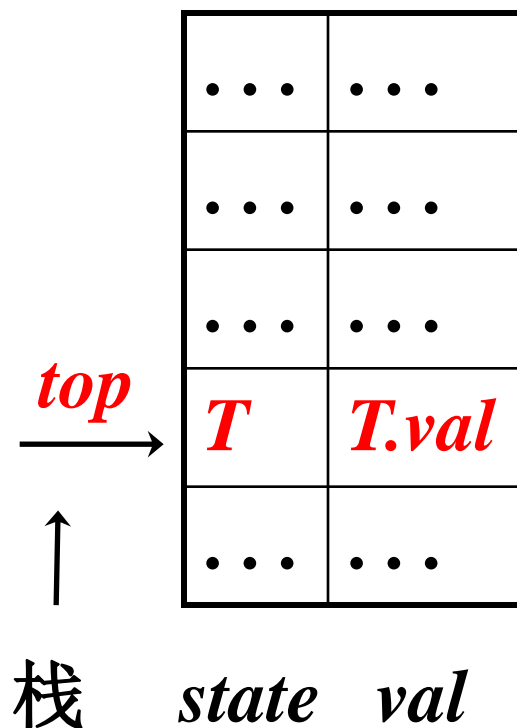
• 简单计算器的语法制导定义改成栈操作代码



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



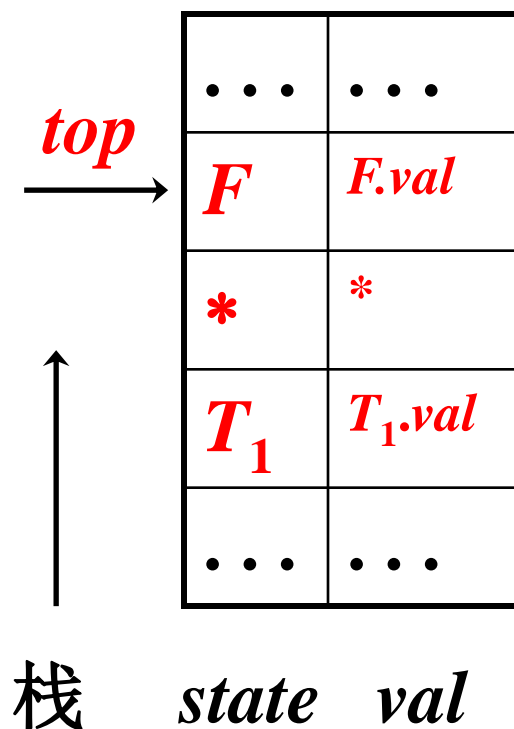
- 简单计算器的语法制导定义改成栈操作代码



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
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$



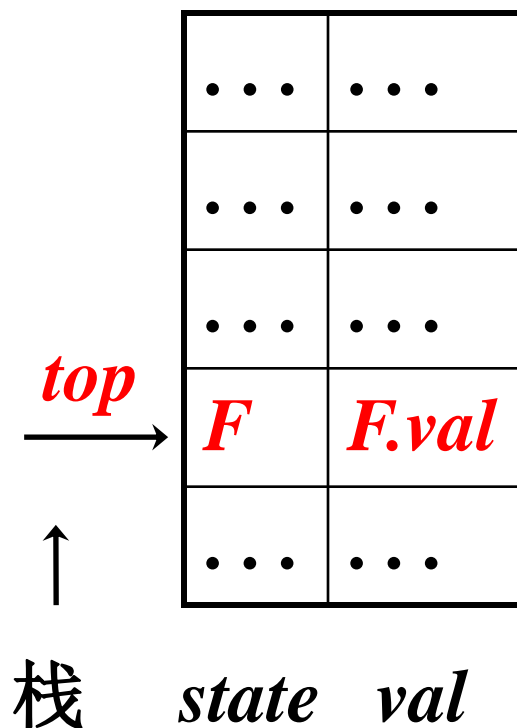
- 简单计算器的语法制导定义改成栈操作代码



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



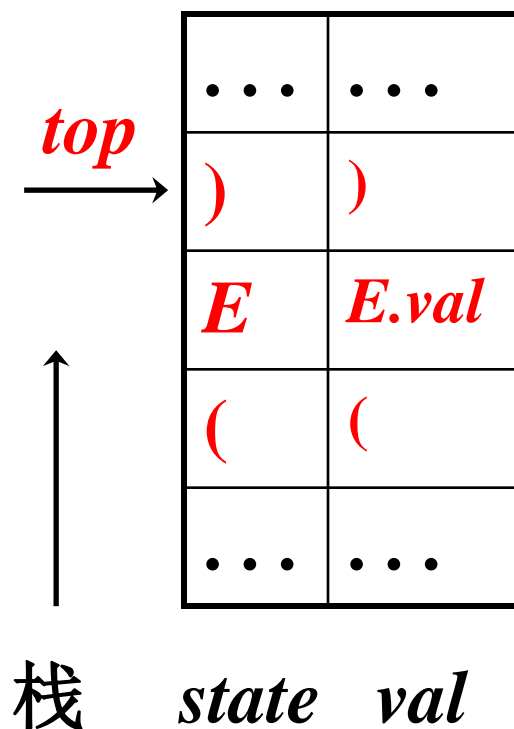
- 简单计算器的语法制导定义改成栈操作代码



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



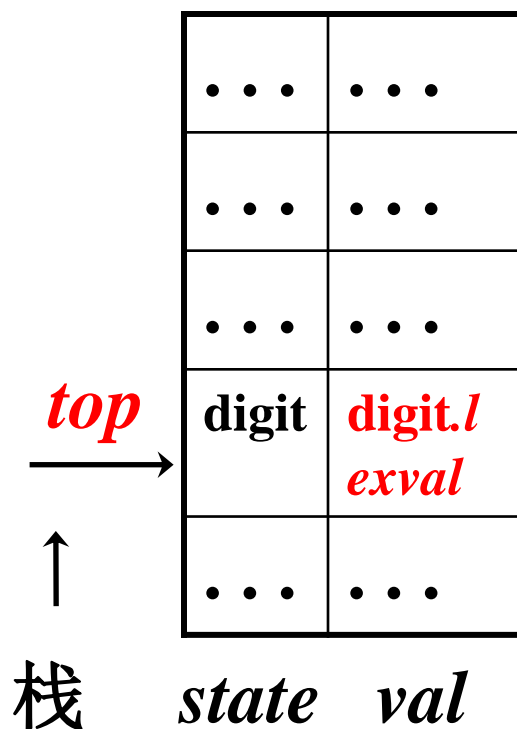
- 简单计算器的语法制导定义改成栈操作代码



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



• 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
$T \rightarrow T_1 * F$	$val[top-2] =$ $val[top-2] \times val[top]$
$T \rightarrow F$	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。

2023年秋季学期 《编译原理和技术》



一起努力 打造国产基础软硬件体系！

李 诚

国家高性能计算中心(合肥)、信息与计算机国家级实验教学示范中心
计算机科学与技术学院

2023年10月11日