语法制导翻译
Part2: S/L属性的定义

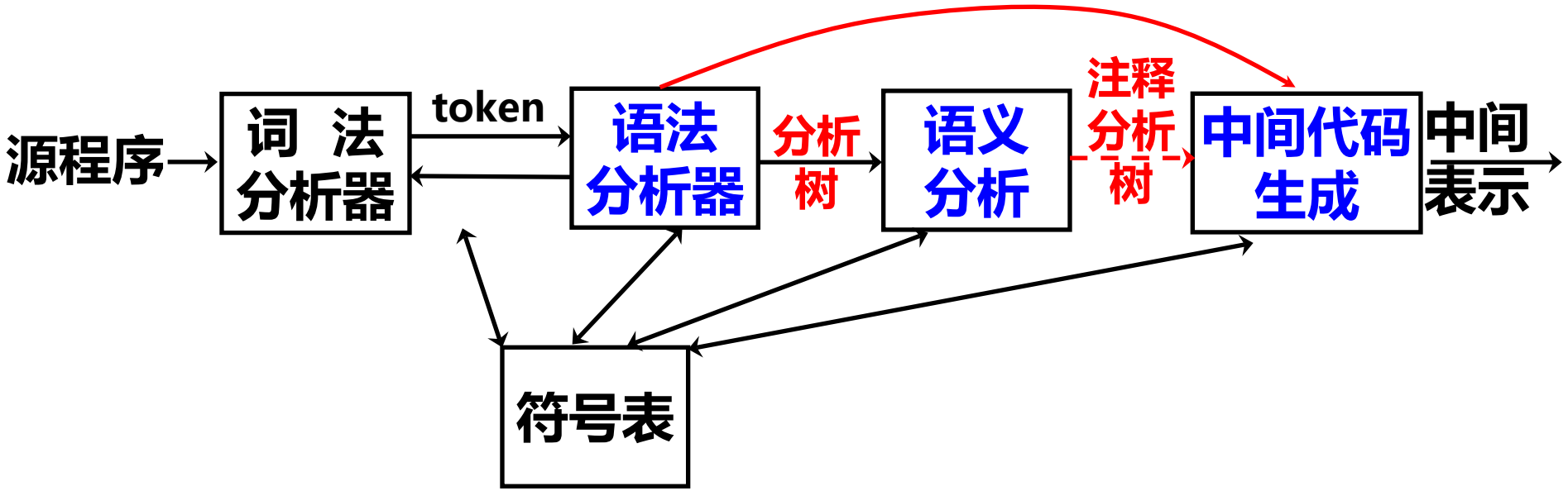
李 诚

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

2024年9月30日

本节提纲



- S属性的定义
- L属性的定义
- 语法制导定义的应用

S属性的定义(S-SDD)

- **仅仅使用综合属性的语法制导定义称为S属性的SDD，或S-属性定义、S-SDD**
  - 如果一个SDD是S属性的，可以按照语法分析树节点的任何自底向上顺序来计算它的各个属性值
  - S-属性定义可在自底向上的语法分析过程中实现
    - 例如：LR分析器

S属性的定义(S-SDD)

- 仅仅使用综合属性的语法制导定义称为S属性的SDD，或S-属性定义、S-SDD

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*

举例

- **下面是产生字母表Σ = {0, 1, 2}上数字串的一个文法：**

  S → D S D | 2

  D → 0 | 1

  - 写一个语法制导定义，判断它接受的句子是否为回文数

举例

- **下面是产生字母表Σ = {0, 1, 2}上数字串的一个文法：**

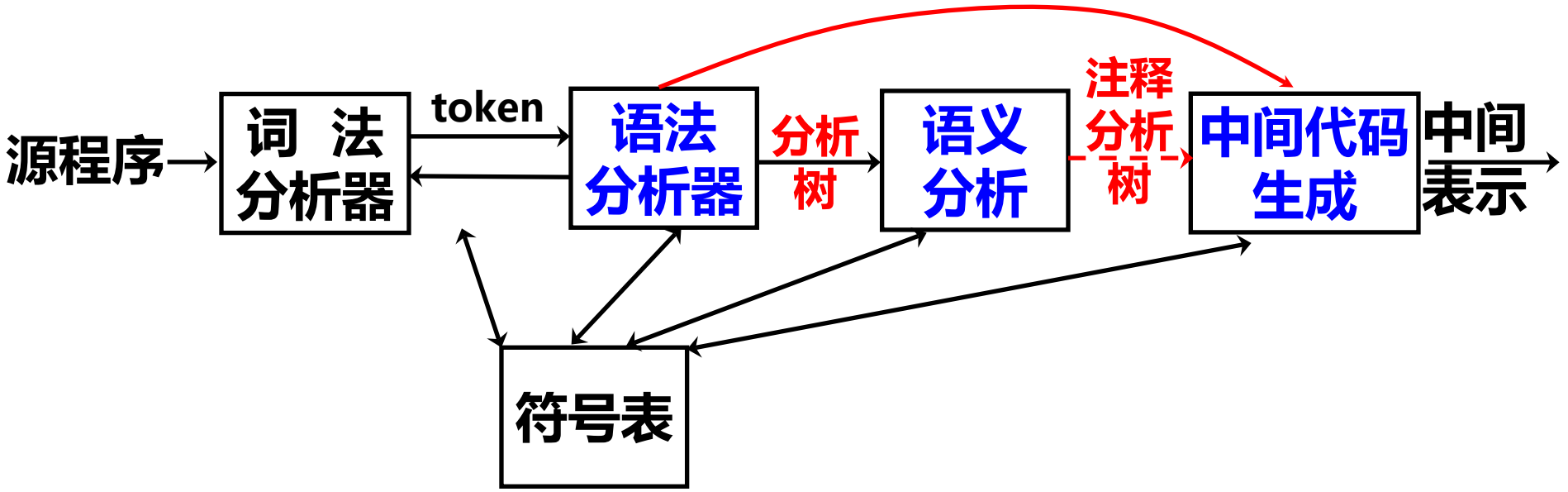
  S → D S D | 2

  D → 0 | 1

  - 写一个语法制导定义，判断它接受的句子是否为回文数

$S' \rightarrow S$	print(S.*val*)
$S \rightarrow D_1\ S_1\ D_2$	$S.val = (D_1.val == D_2.val)$ and $S_1.val$
$S \rightarrow 2$	$S.val = true$
$D \rightarrow 0$	$D.val = 0$
$D \rightarrow 1$	$D.val = 1$

本节提纲



- S属性的定义
- L属性的定义
- 语法制导定义的应用

L属性的定义

- **L-属性定义(也称为L属性的SDD或L-SDD)的直观含义：**
  - 在一个产生式所关联的各属性之间，依赖图的边可以从左到右，但不能从右到左(因此称为L属性的，L是Left的首字母)
  - 可以在LR分析器或者LL分析器中实现
  - 更加一般化

L属性的定义

- **任意产生式$A \rightarrow X_1X_2...X_n$，其右部符号$X_i(1 \leq i \leq n)$的继承属性仅依赖于下列属性：**
  - A的继承属性
    - 不能依赖A的综合属性的原因：由于A的综合属性可能依赖X_i的属性，包括X_i的综合属性和继承属性，因此可能形成环路
  - 产生式中X_i左边的符号$X_1,X_2,...,X_{i-1}$的属性
  - X_i本身的属性，但X_i的全部属性不能在依赖图中形成环路

L属性的定义

例：*L-SDD*

	产生式	语义规则
(1)	$T \rightarrow F\ T'$	$T'.inh = F.val$
		$T.val = T'.syn$
(2)	$T' \rightarrow *\ F\ T_1'$	$T_1'.inh = T'.inh \times F.val$
		$T'.syn = T_1'.syn$
(3)	$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
(4)	$F \rightarrow$ digit	$F.val =$ digit.*lexval*

继承属性

综合属性

L属性的定义

非L属性的SDD

➢例

产生式	语义规则
(1) $A \rightarrow LM$	$L.i = l(A.i)$ $M.i = m(L.s)$ $A.s = f(M.s)$
(2) $A \rightarrow QR$	$R.i = r(A.i)$ $Q.i = q(R.s)$ $A.s = f(Q.s)$

继承属性

综合属性

L属性的定义：举例

int id, id, id
标识符声明

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.$entry$, $L.in$)

- *type* – *T*的综合属性
- *in* – *L*的继承属性，把声明的类型传递给标识符列表
- *addType*– 把类型加到符号表中的标识符条目里(副作用)

属性文法vs.副作用

- **一个没有副作用的SDD称为属性文法**
  - 属性文法增加了语义规则描述的复杂度
  - 如：符号表必须作为属性传递
  - 为了简单起见，我们可以把符号表作为全局变量，通过副作用函数读取或者添加标识符

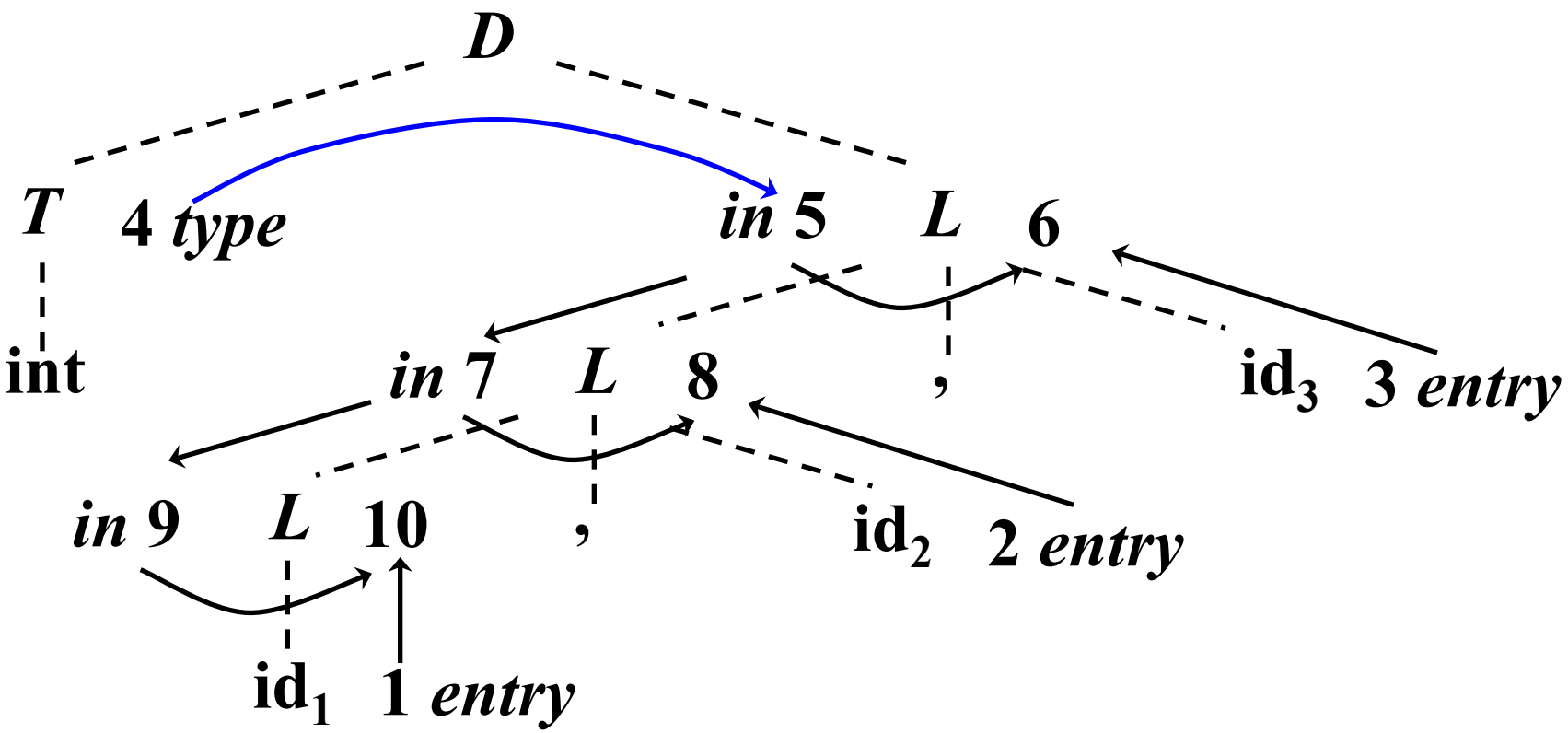
属性文法vs.副作用

- **一个没有副作用的SDD称为属性文法**
  - 属性文法增加了语义规则描述的复杂度
  - 如：符号表必须作为属性传递
  - 为了简单起见，我们可以把符号表作为全局变量，通过副作用函数读取或者添加标识符
- **受控的副作用**
  - 不会对属性求值产生约束，即可以按照任何拓扑顺序求值，不会影响最终结果
  - 或者对求值过程添加约束

L属性的定义：举例

- 例　int id_1, id_2, id_3的分析树（虚线）的依赖图（实线）

  $D \rightarrow TL$　　$L.in = T.type$

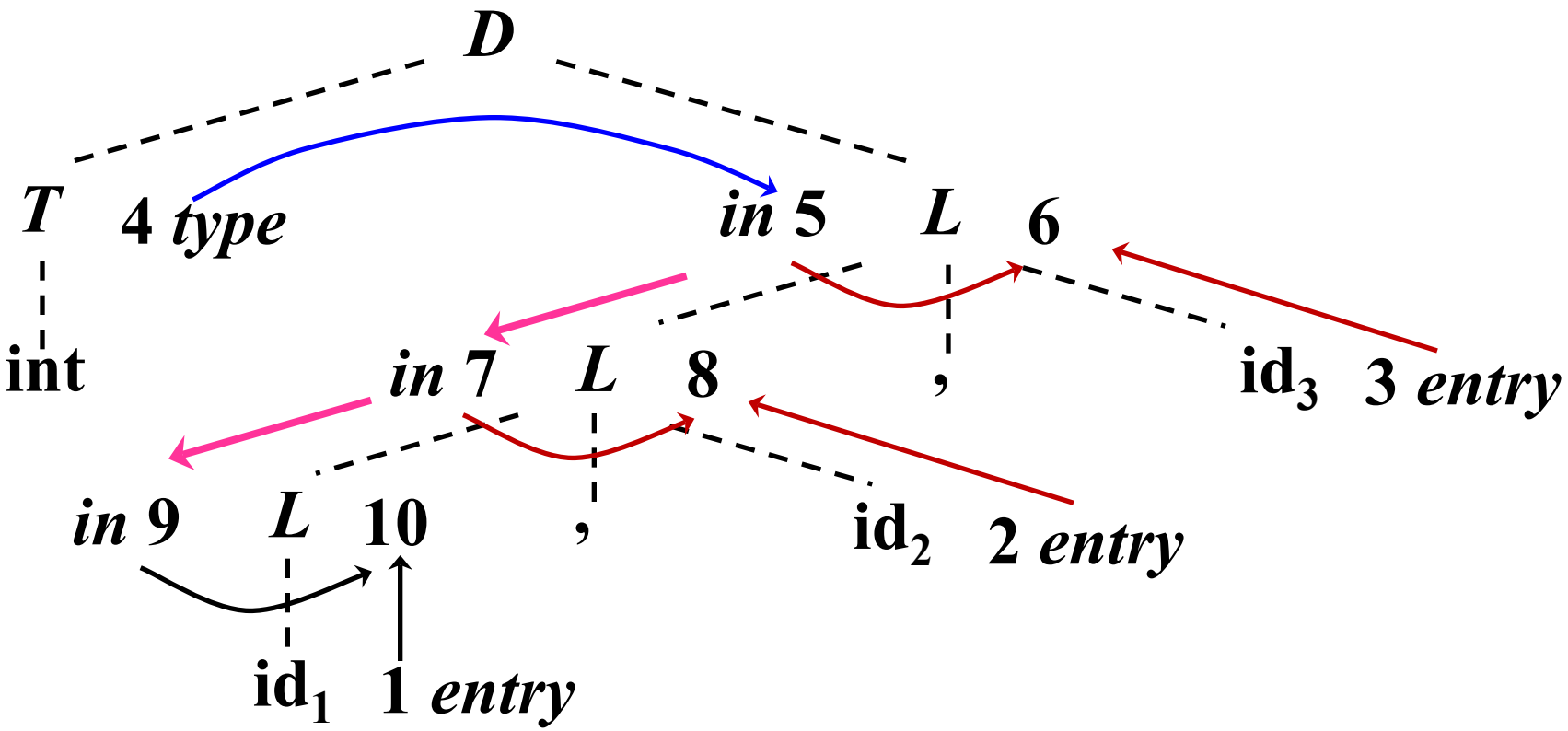


L属性的定义：举例

- 例　int id_1, id_2, id_3的分析树（虚线）的依赖图（实线）

  $L \rightarrow L_1$, id　$L_1.in = L.in$;

  addType (id.*entry*, *L.in*)

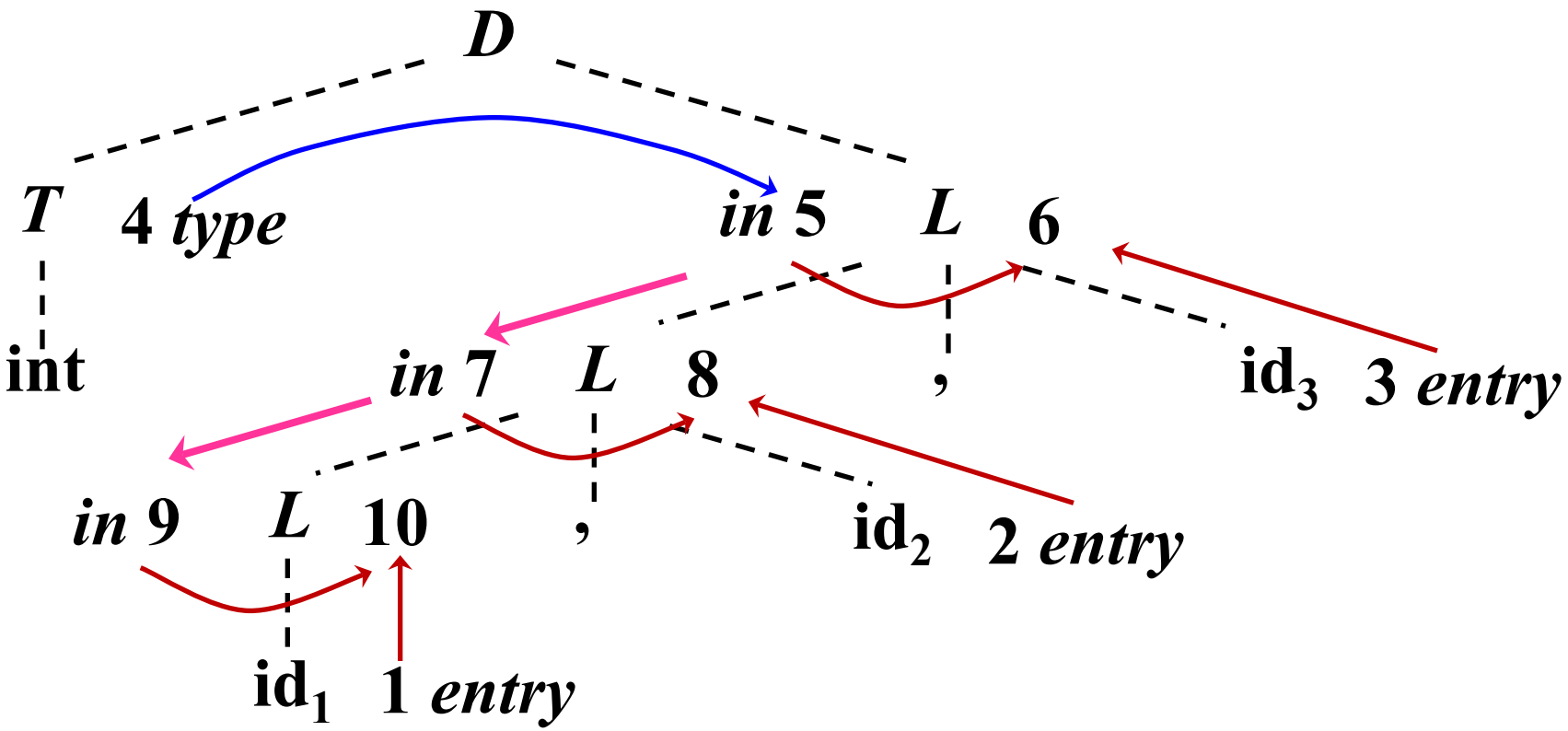


L属性的定义：举例

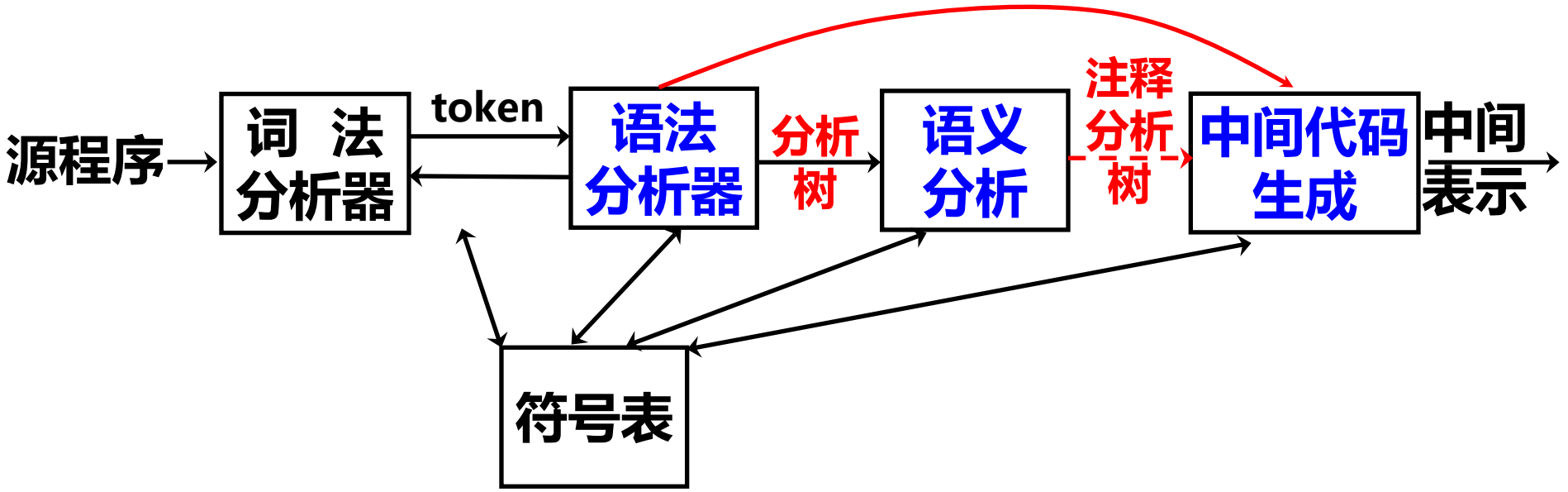
- 例　int id_1, id_2, id_3的分析树（虚线）的依赖图（实线）

  $L \rightarrow$ id

addType (id.*entry*, *L.in*)



本节提纲



- S属性的定义
- L属性的定义
- **语法制导定义的应用**

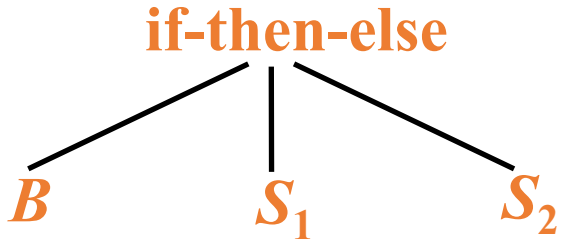
语法制导定义的应用

- **抽象语法树的构造**
  - S属性定义的方法
  - L属性定义的方法
- **类型检查(下一章)**
- **中间代码生成(下一章)**

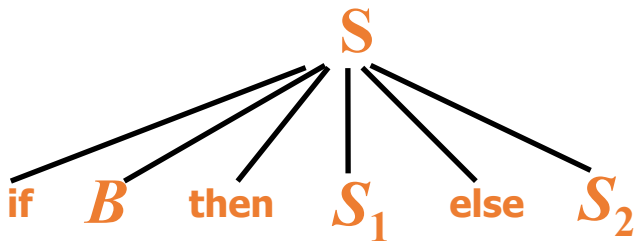
抽象语法树

- **抽象语法树(Abstract syntax tree, AST)**
  - 简称语法树，是分析树的浓缩表示：算符和关键字是作为内部结点。
  - 语法制导翻译可基于分析树，也可基于语法树

S→ if B then S1 else S2



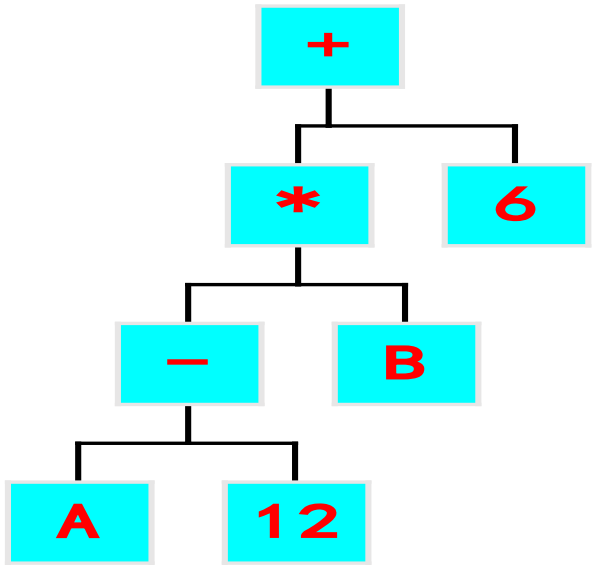
语法树



分析树

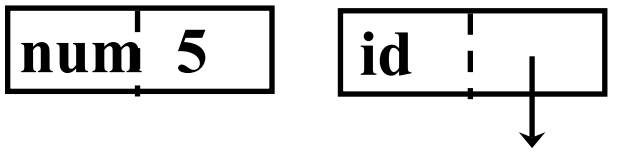
抽象语法树

- 例：表达式 (A - 12) * B + 6 的语法树。



建立算符表达式的语法树

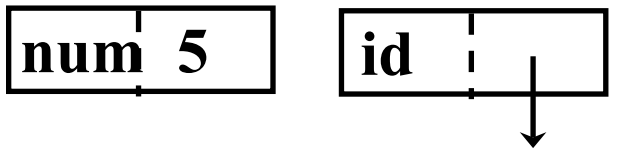
- 对**基本运算对象结点**
  - 一个域存放运算对象类别
  - 另一个域存放其值（也可用其他域保存其他属性或者指向该属性值的指针）



指向符号表中id的条目

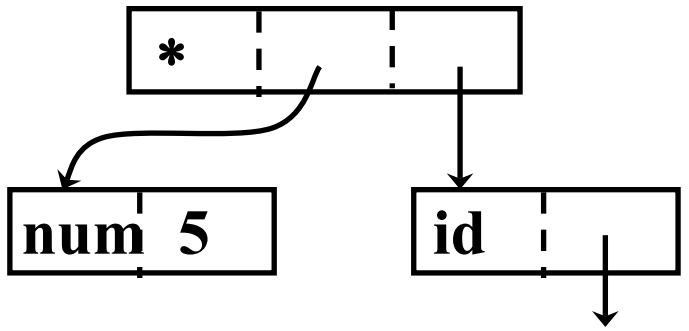
建立算符表达式的语法树

- 对**基本运算对象结点**
  - 一个域存放运算对象类别
  - 另一个域存放其值（也可用其他域保存其他属性或者指向该属性值的指针）



指向符号表中id的条目

- 对**算符结点**
  - 一个域存放算符并作为该结点的标记
  - 其余两个域存放指向运算对象的指针。



建立算符表达式的语法树

- **mknode (op, left, right)**
  - 建立一个运算符号结点，标号是op，两个域left和right分别指向左子树和右子树。
- **mkleaf (id, entry)**
  - 建立一个标识符结点，标号为id，一个域entry指向标识符在符号表中的入口。
- **mkleaf (num, val)**
  - 建立一个数结点，标号为num，一个域val用于存放数的值。

构造抽象语法树的S属性语法制导定义

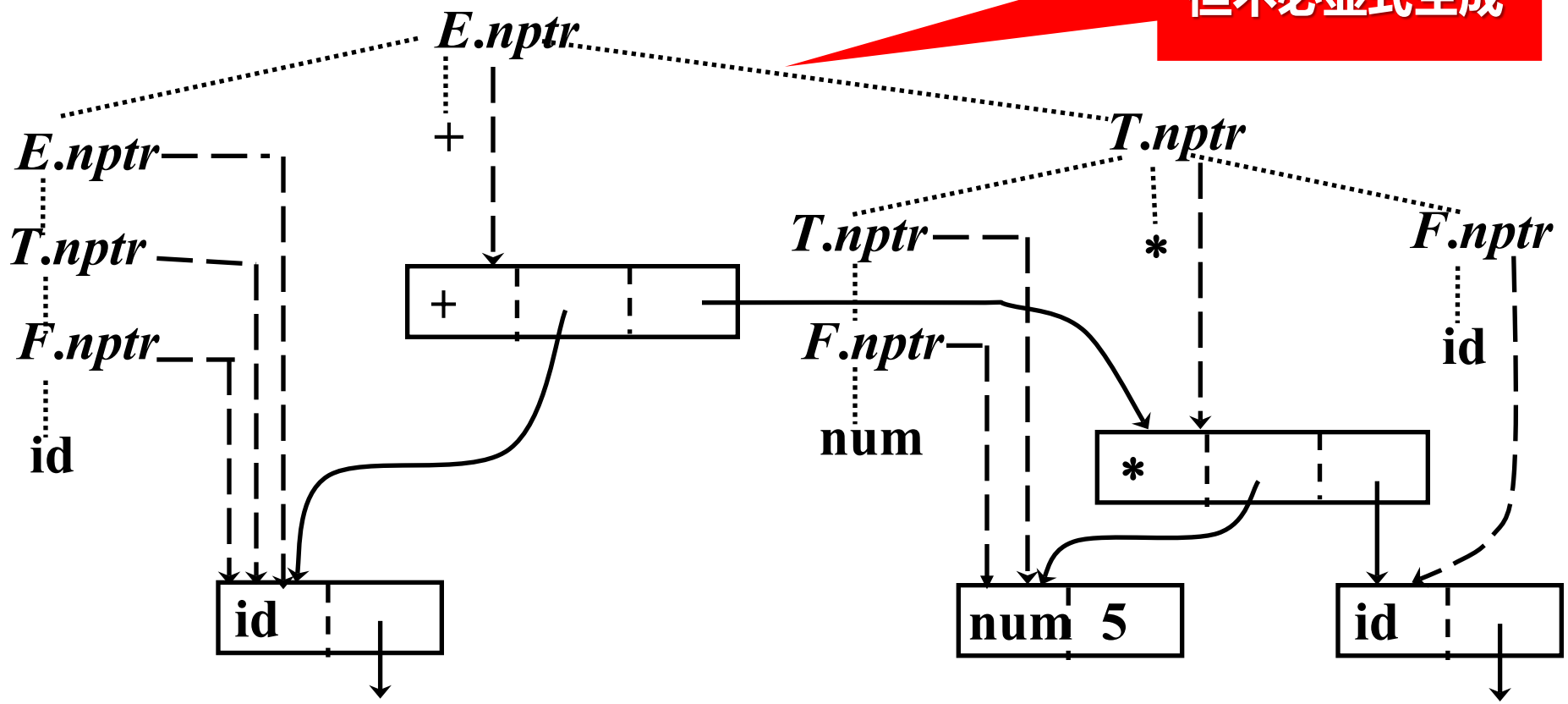
- **以算数表达式为例**
  - nptr综合属性：文法符号对应的抽象语法树结点

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

S属性定义的语法树构造

a+5*b的语法树的构造

不带箭头的虚线代表语法分析树，但不必显式生成

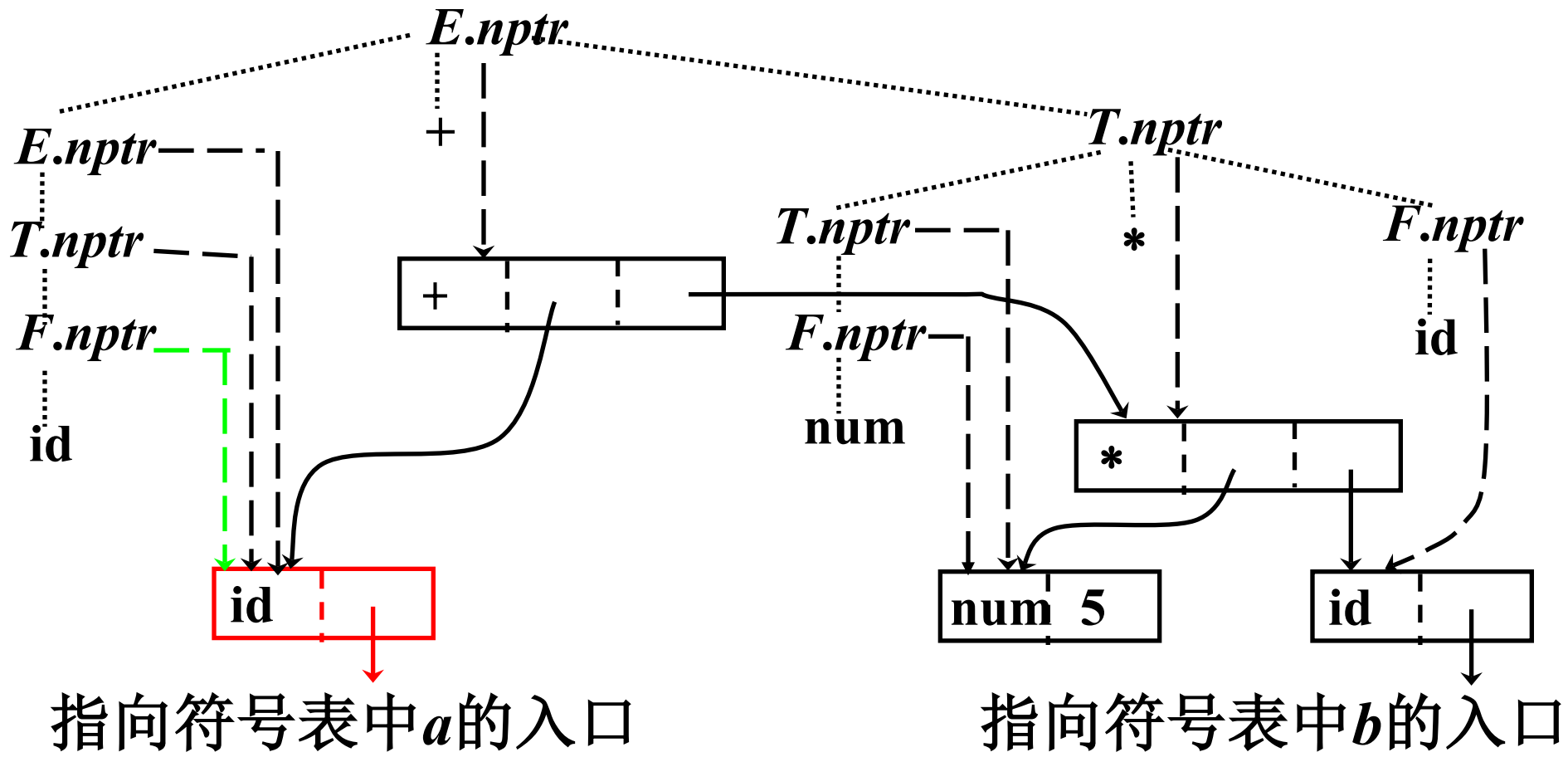


指向符号表中a的入口

指向符号表中b的入口

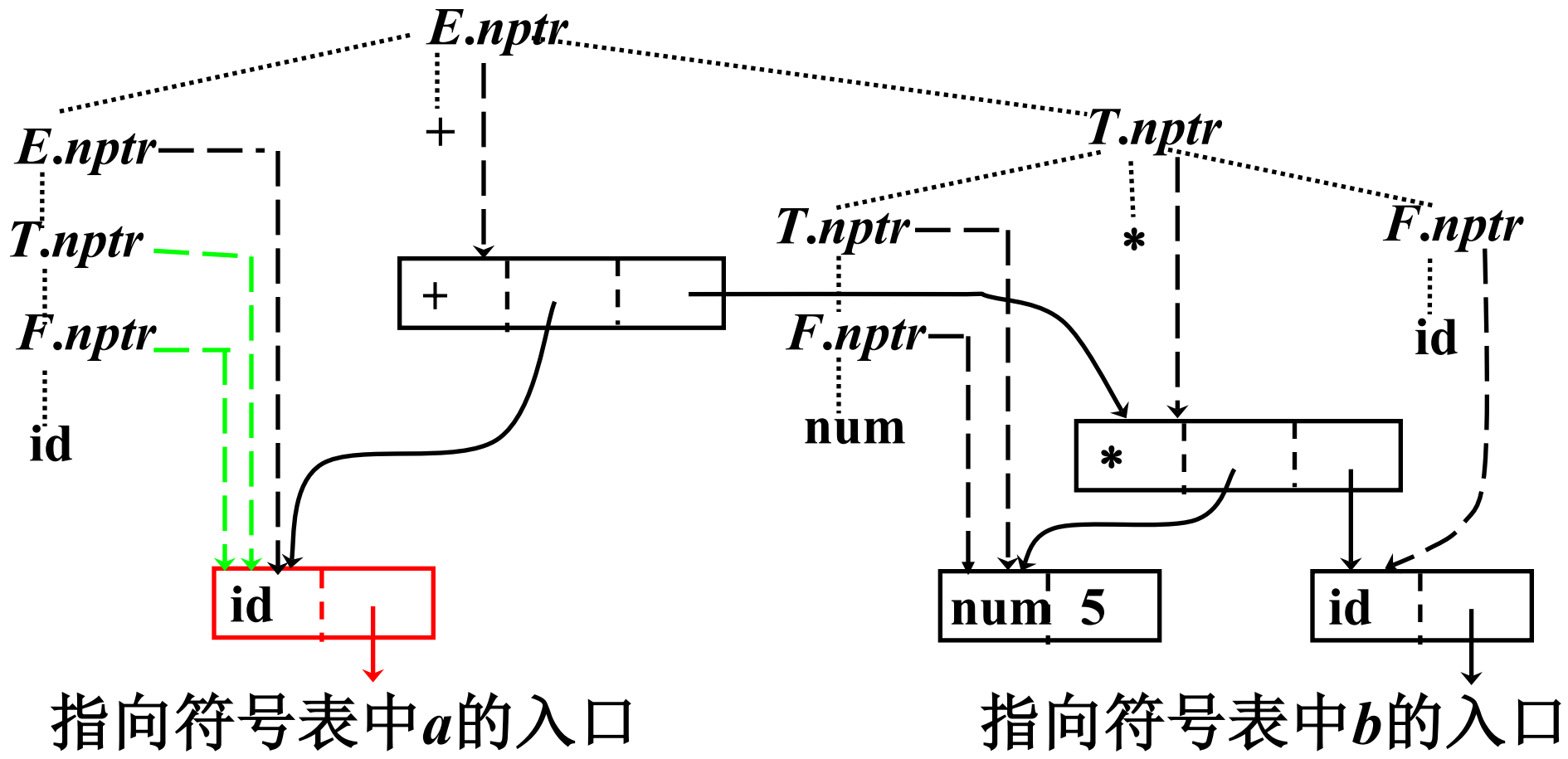
S属性定义的语法树构造

a+5*b的语法树的构造



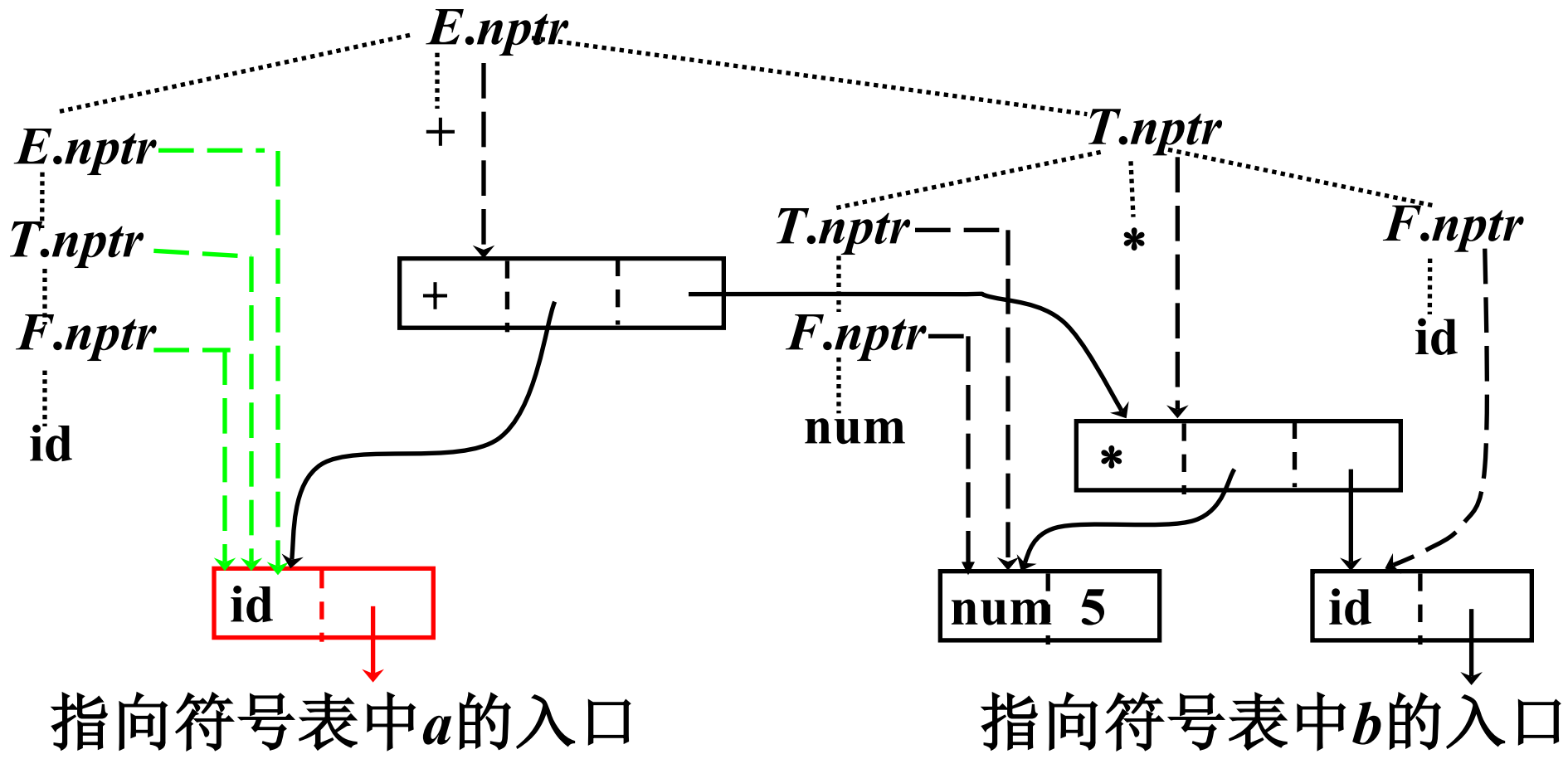
S属性定义的语法树构造

a+5*b的语法树的构造



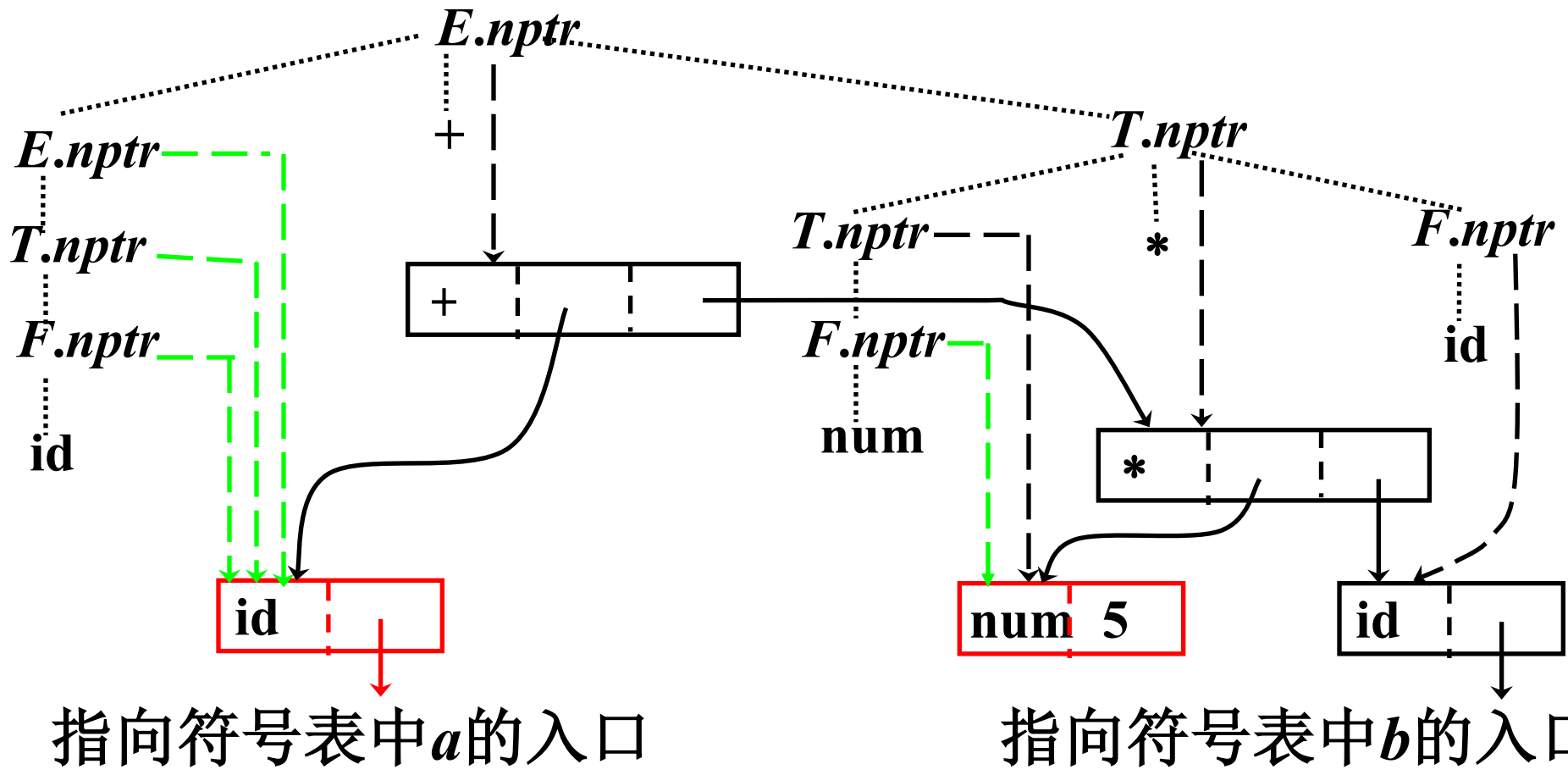
S属性定义的语法树构造

a+5*b的语法树的构造



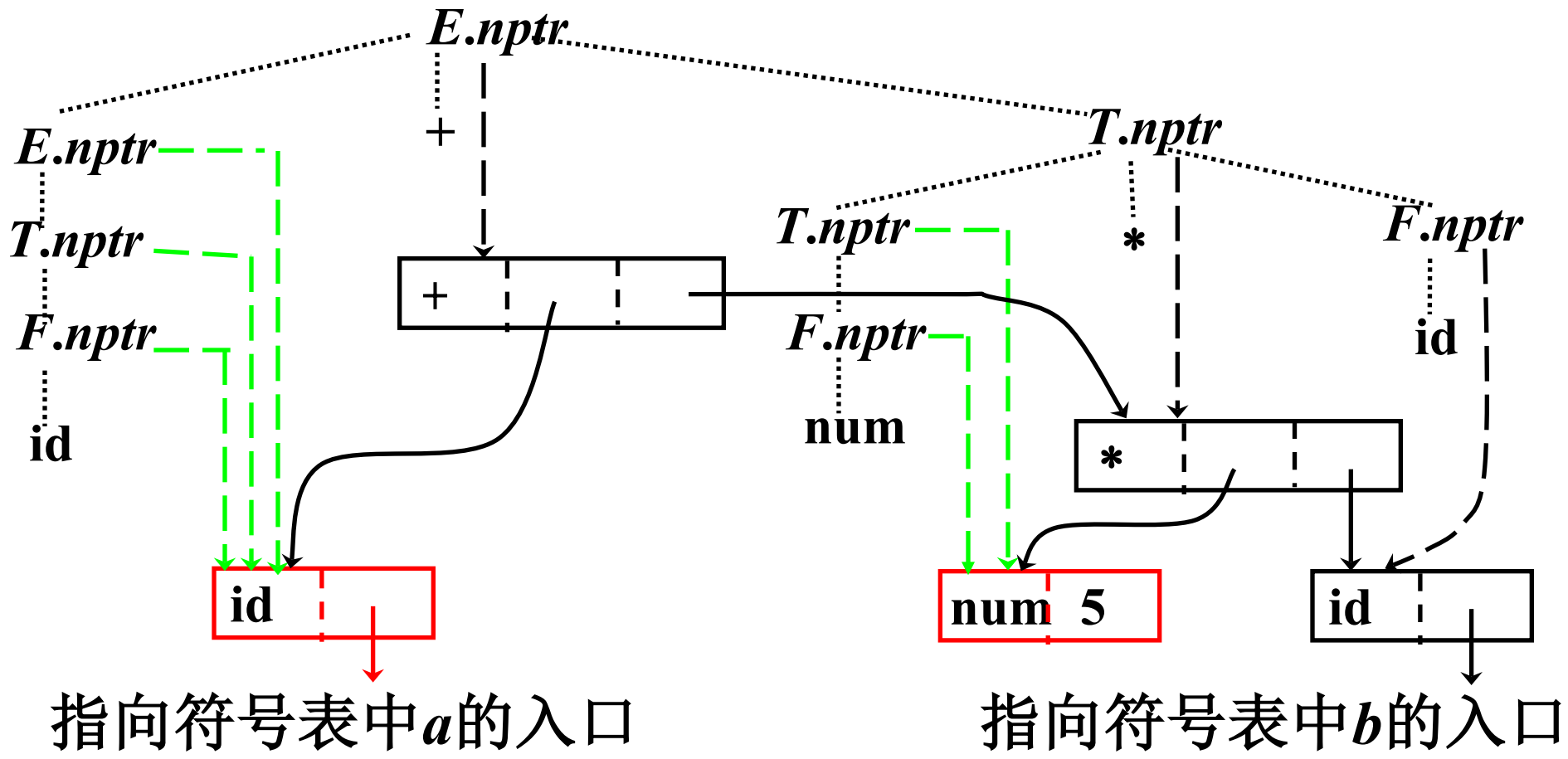
S属性定义的语法树构造

a+5*b的语法树的构造



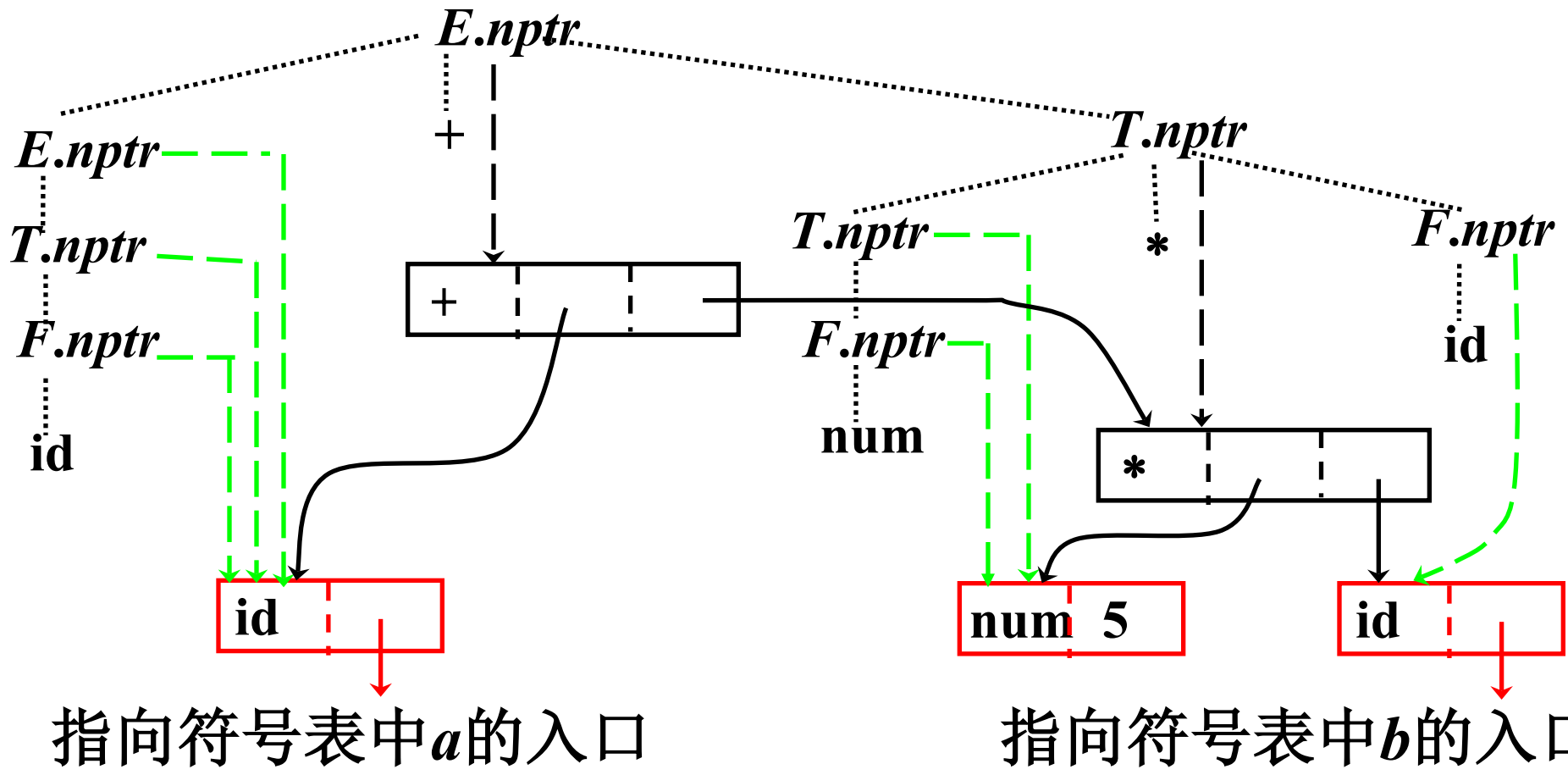
S属性定义的语法树构造

a+5*b的语法树的构造



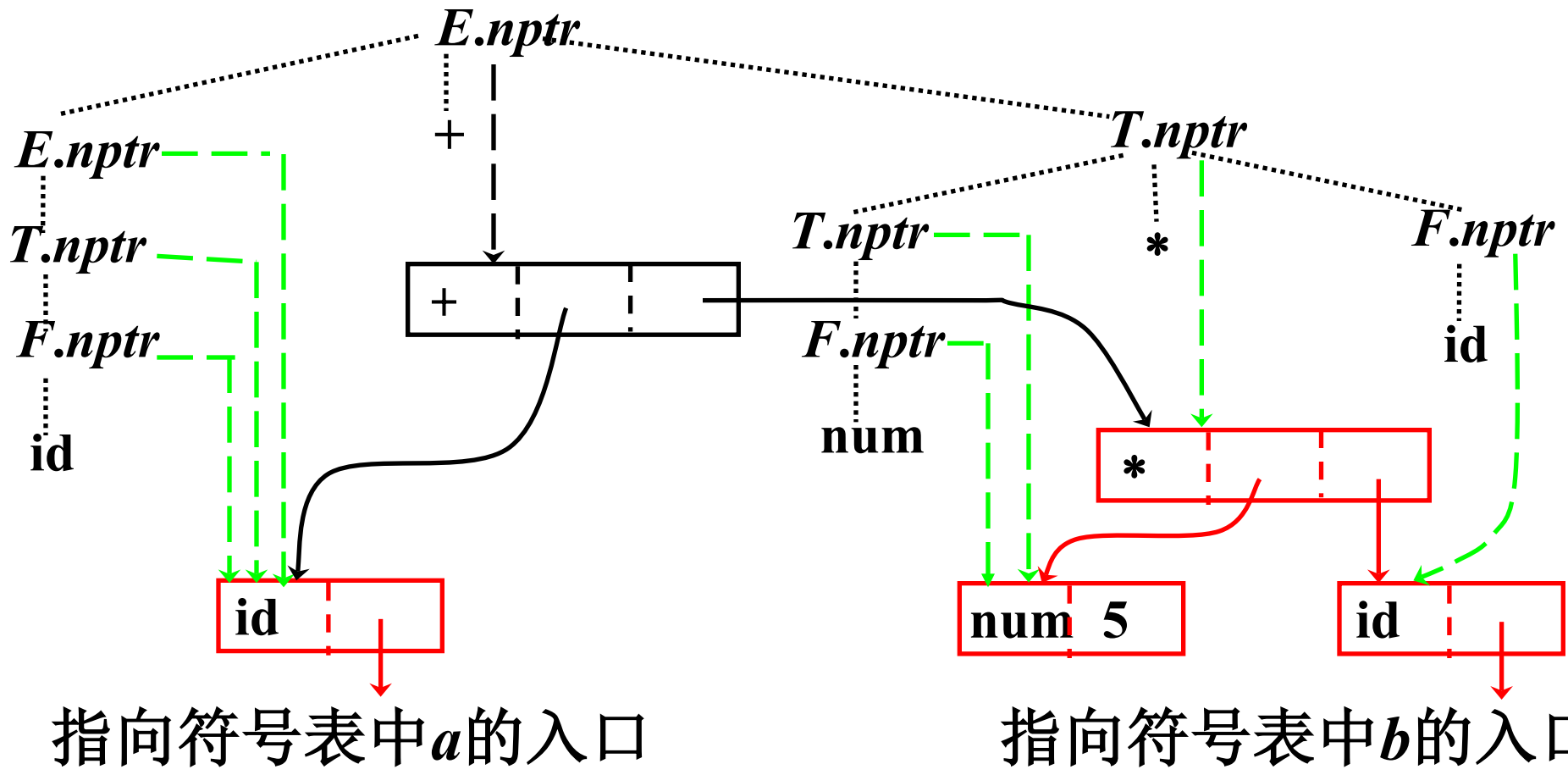
S属性定义的语法树构造

a+5*b的语法树的构造



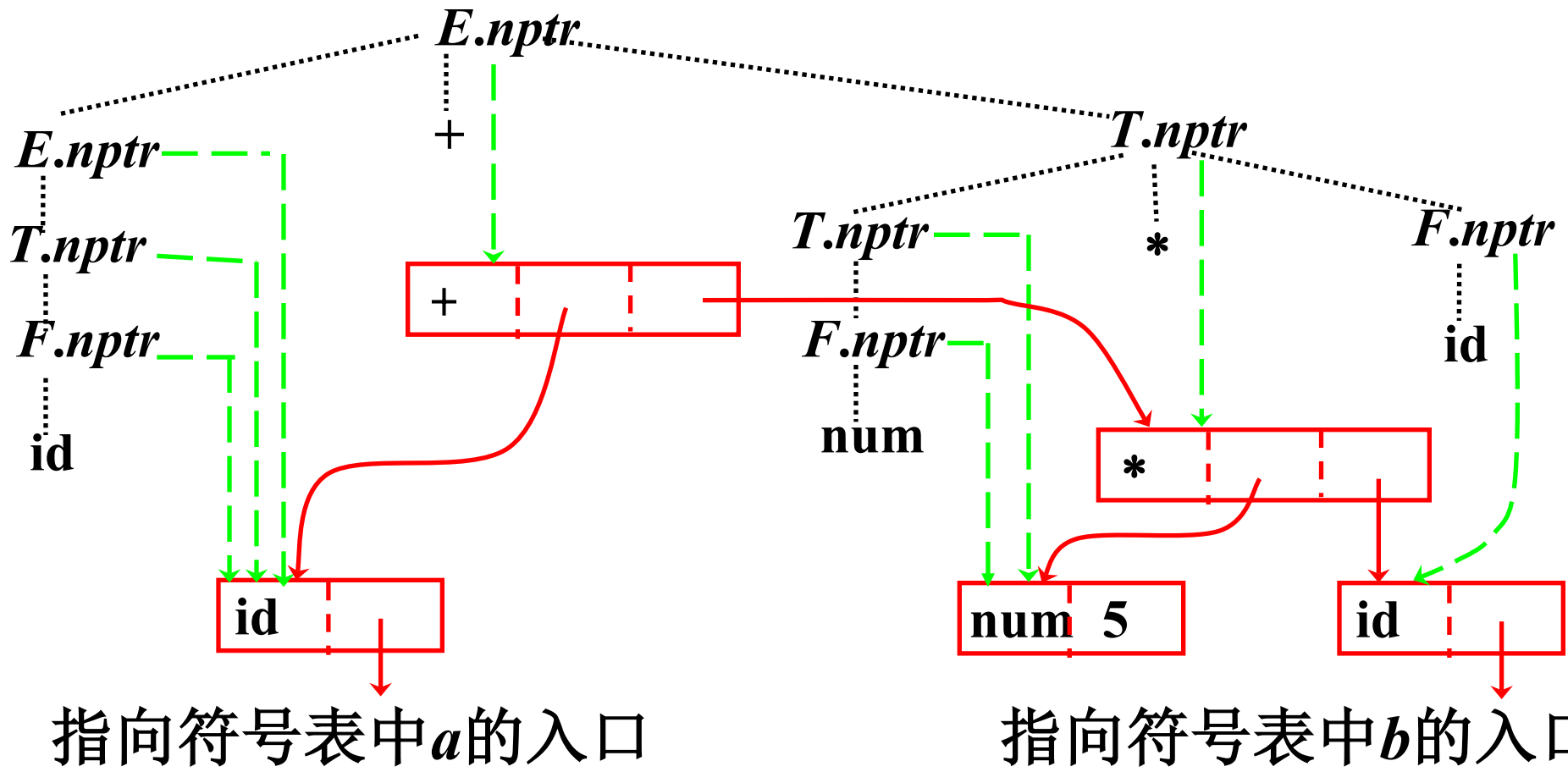
S属性定义的语法树构造

a+5*b的语法树的构造



S属性定义的语法树构造

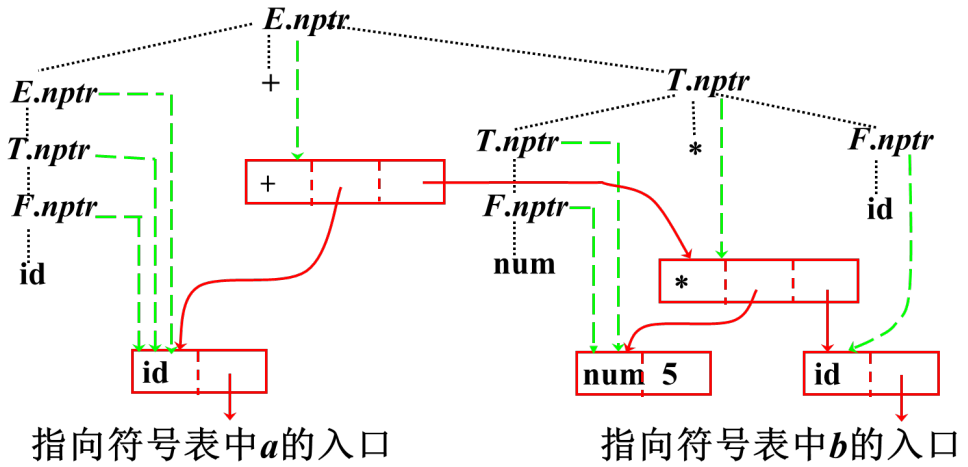
a+5*b的语法树的构造



S属性定义的语法树构造

a+5**b*的语法树构造步骤

1) p1:=mkleaf(id,entry a);
2) p2:=mkleaf(num, 5);
3) p3:=mkleaf(id, entry b)
4) p4:=mknode('*', p2, p3)
5) p5:=mknode('+', p1, p4)



$p_1, p_2, \ldots, p_5$是指向结点的指针，*entry a* 和 *entry c* 分别指向符号表中标识符*a*和*c*的指针。

S属性的SDD—小结

- 每个属性都是综合属性
- 在依赖图中，总是通过子结点的属性值来计算父结点的属性值。

S属性的SDD—小结

- **每个属性都是综合属性**
- **在依赖图中，总是通过子结点的属性值来计算父结点的属性值。**
  - 特别适合与自底向上的语法分析过程一起计算
    - 在用产生式归约时，即在构造分析树的中间节点时，计算相关属性（此时其子结点的属性必然已经计算完）

S属性的SDD—小结

- **每个属性都是综合属性**
- **在依赖图中，总是通过子结点的属性值来计算父结点的属性值。**
  - 特别适合与自底向上的语法分析过程一起计算
    - 在用产生式归约时，即在构造分析树的中间节点时，计算相关属性（此时其子结点的属性必然已经计算完）
  - 也可以与自顶向下的语法分析过程一起计算
    - 递归下降分析中，可以在过程A()的最后一步计算A的属性(此时，A调用的其他子结点过程已处理完)

语法制导定义的应用

- **抽象语法树的构造**
  - S属性定义的方法
  - L属性定义的方法
- **类型检查(下一章)**
- **中间代码生成(下一章)**

左递归的消除引起继承属性

- 考虑以下左递归文法

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

左递归的消除引起继承属性

- 首先消除左递归

$E \rightarrow E_1 + T$
$E \rightarrow T$
$T \rightarrow T_1 * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow \text{id}$
$F \rightarrow \text{num}$

$T + T + T + \ldots$

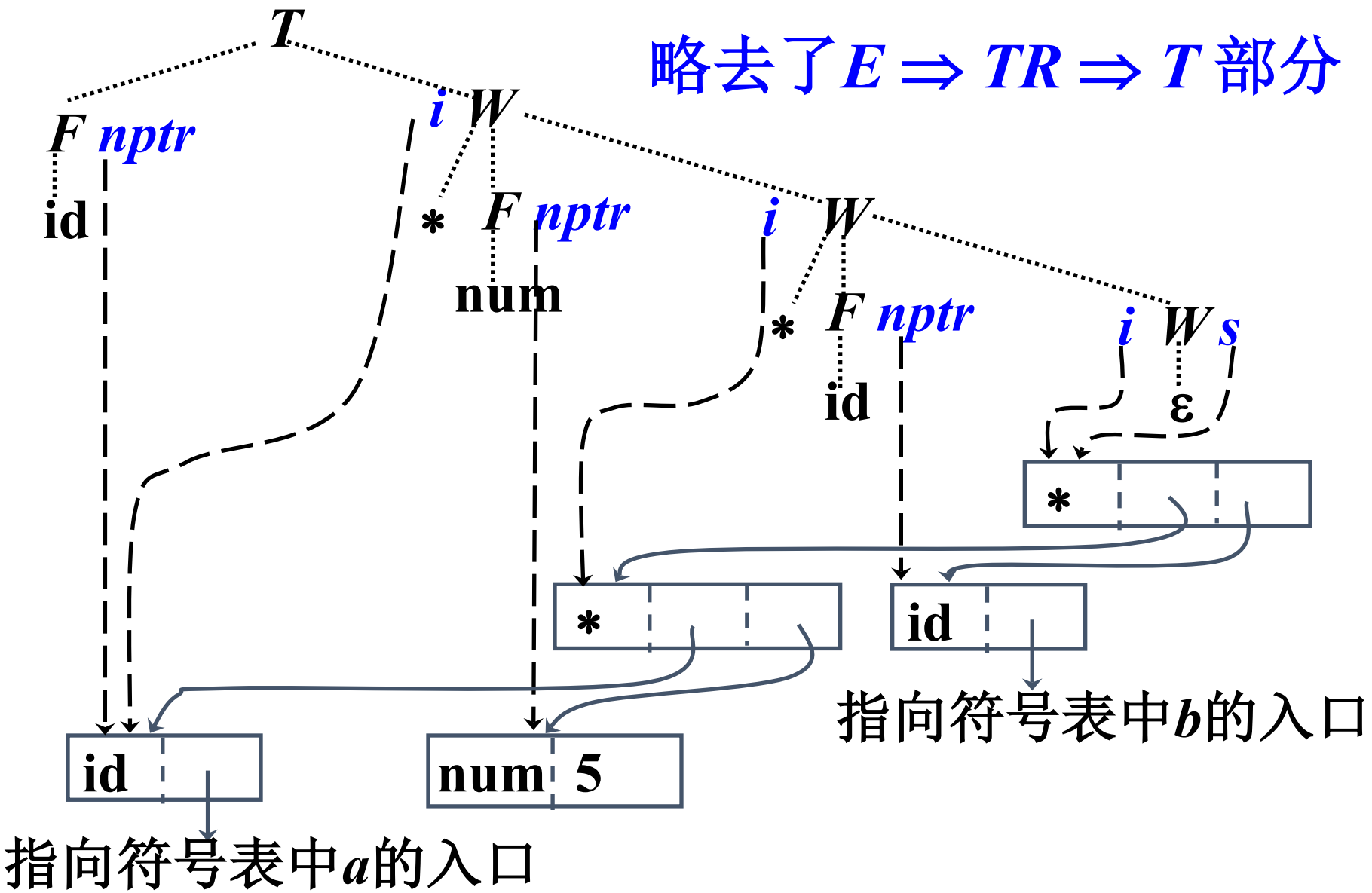
$E \rightarrow TR$
$R \rightarrow +TR_1$
$R \rightarrow \varepsilon$
$T \rightarrow FW$
$W \rightarrow *FW_1$
$W \rightarrow \varepsilon$

F产生式部分不再给出

L属性定义的语法树构造

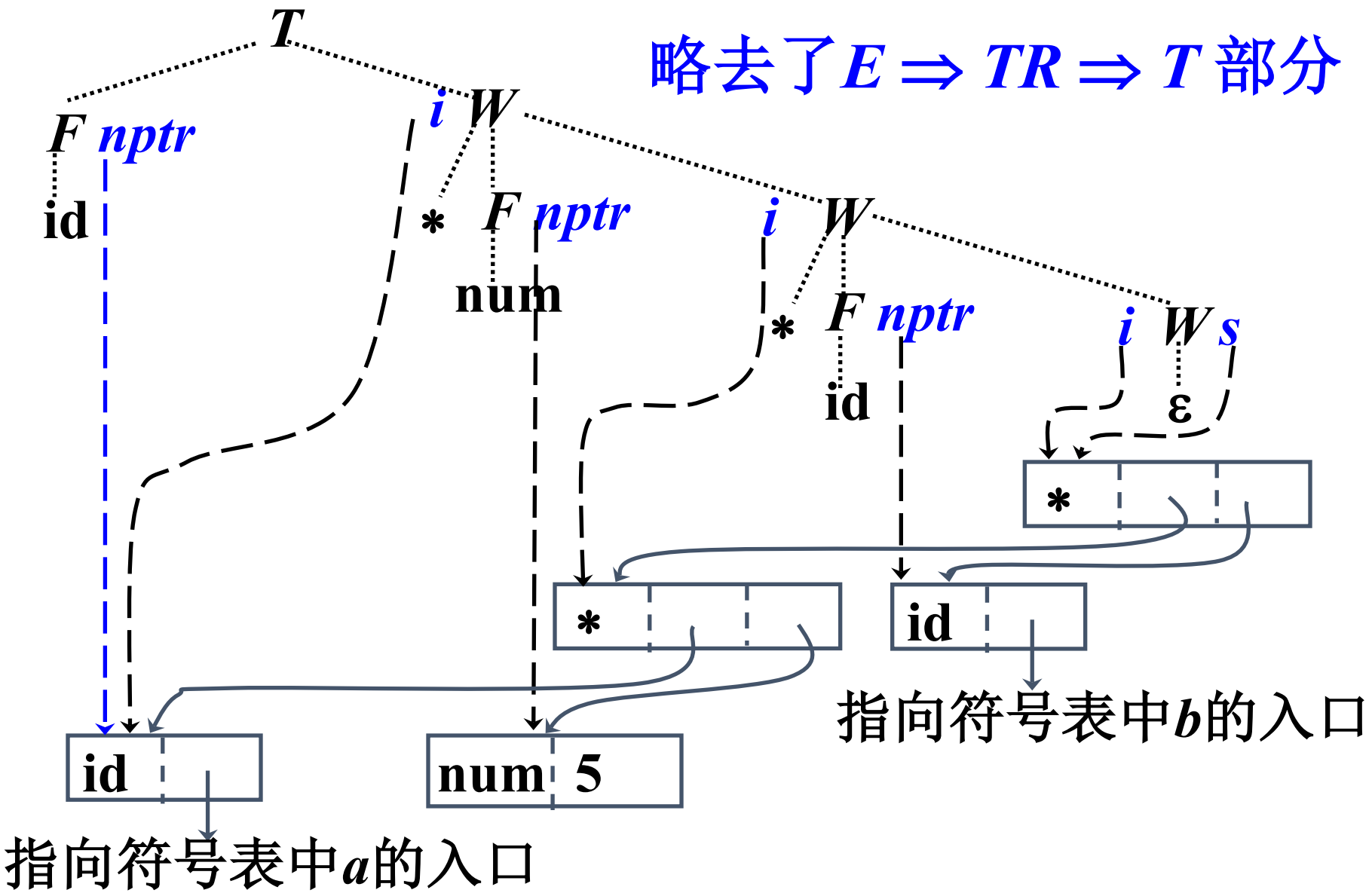
产　生　式	语　义　规　则
$T \rightarrow FW$	$W.i = F.nptr$ $T.nptr = W.s$
$W \rightarrow *FW_1$	$W_1.i = mkNode\ (\ '*',\ W.i,\ F.nptr)$ $W.s = W_1.s$
$W \rightarrow \varepsilon$	$W.s = W.i$
$F \rightarrow$ id	$F.nptr = mkLeaf$ (id, id.*entry*)
$F \rightarrow$ num	$F.nptr = mkLeaf$ (num, num.*val*)

L属性定义的语法树构造



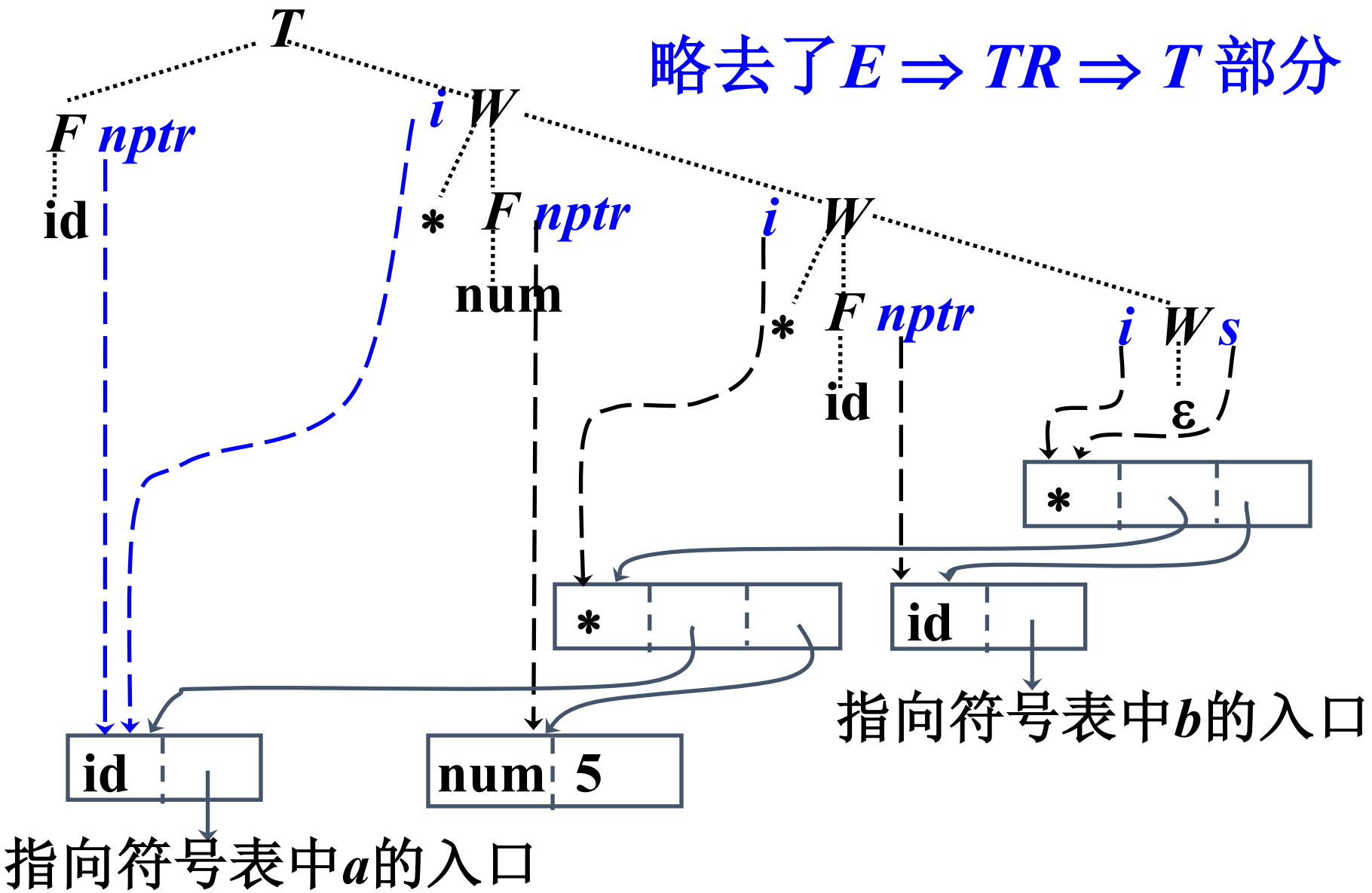
略去了$E \Rightarrow TR \Rightarrow T$部分

L属性定义的语法树构造

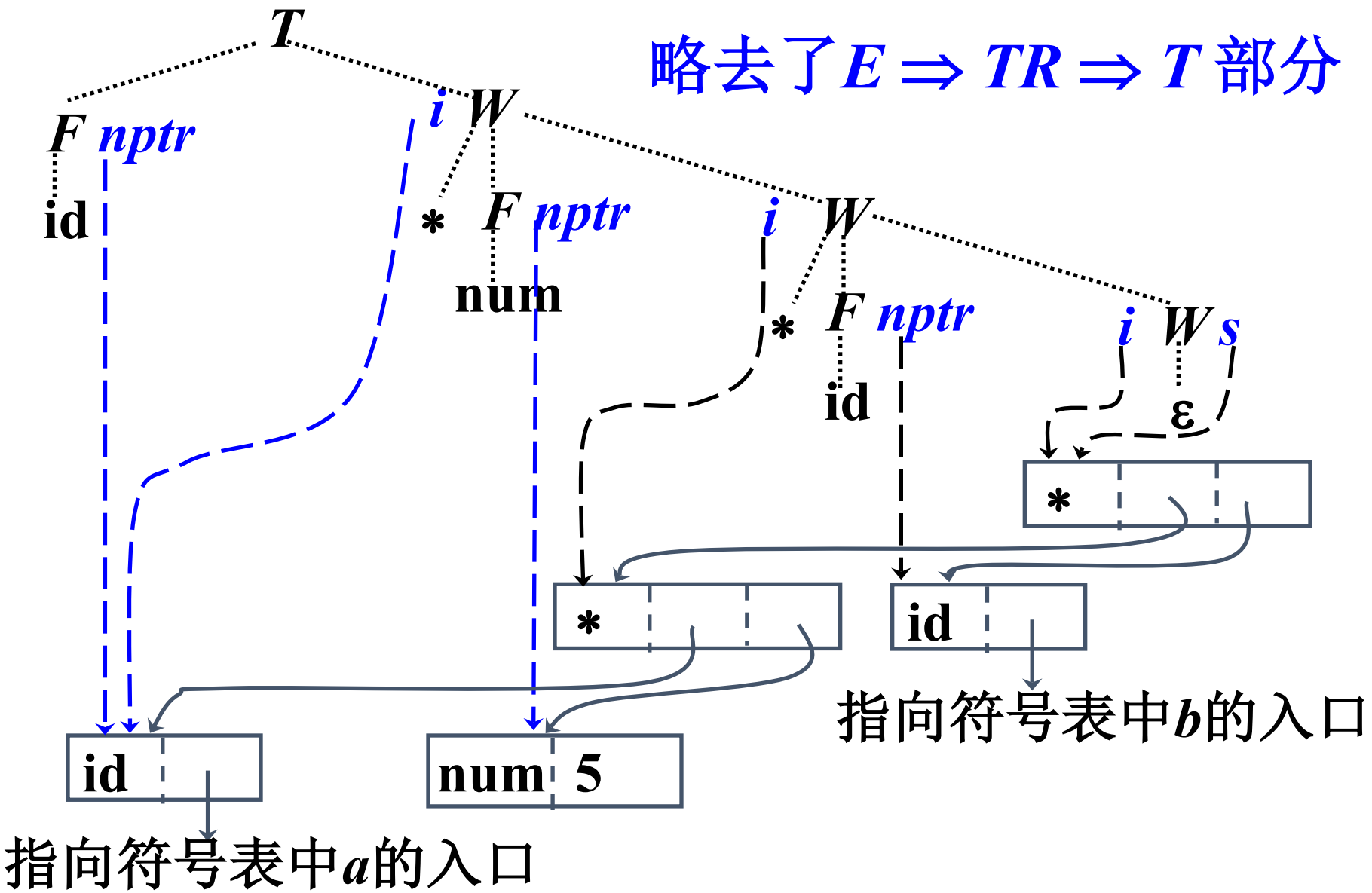


略去了$E \Rightarrow TR \Rightarrow T$部分

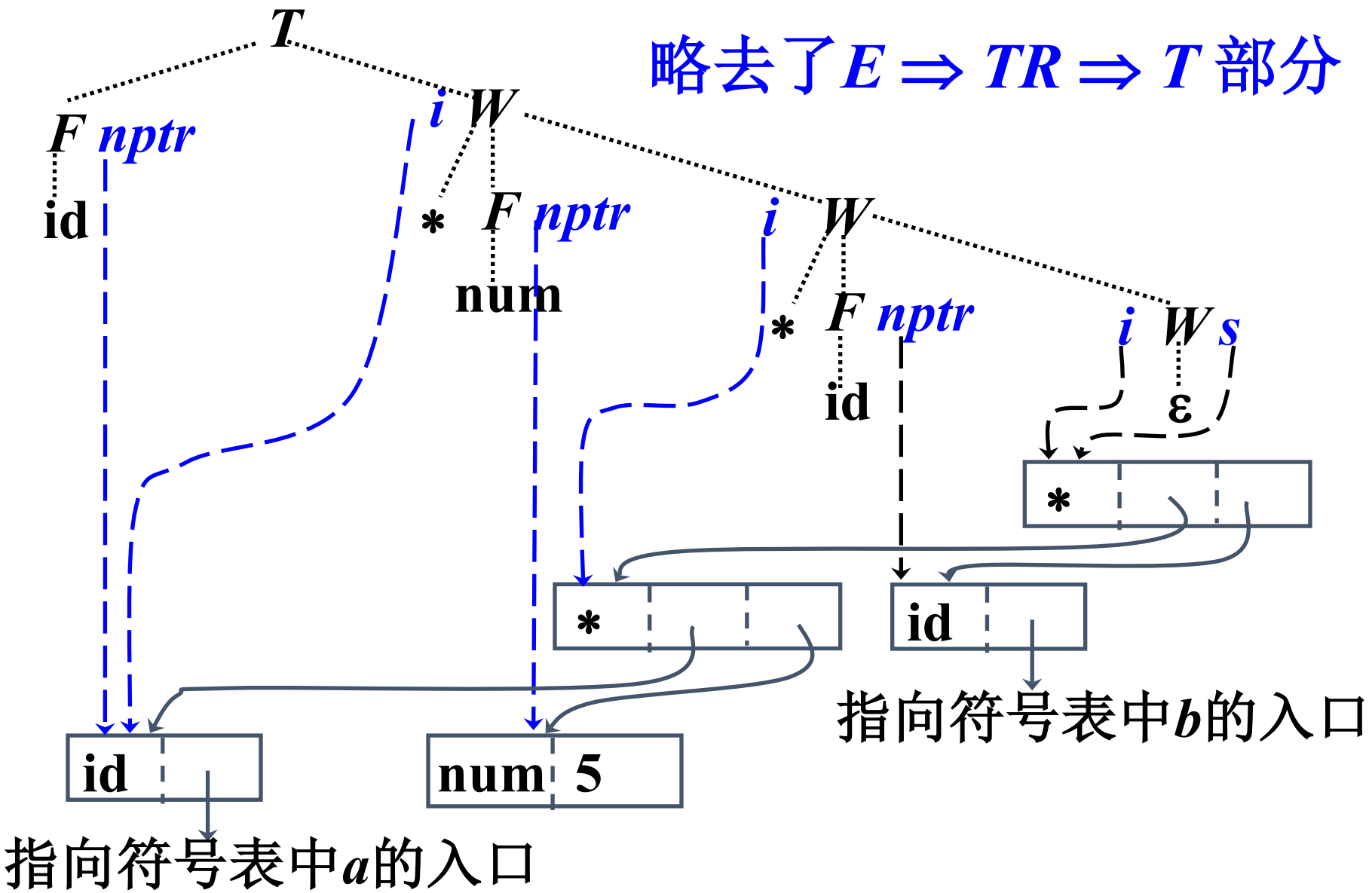
L属性定义的语法树构造



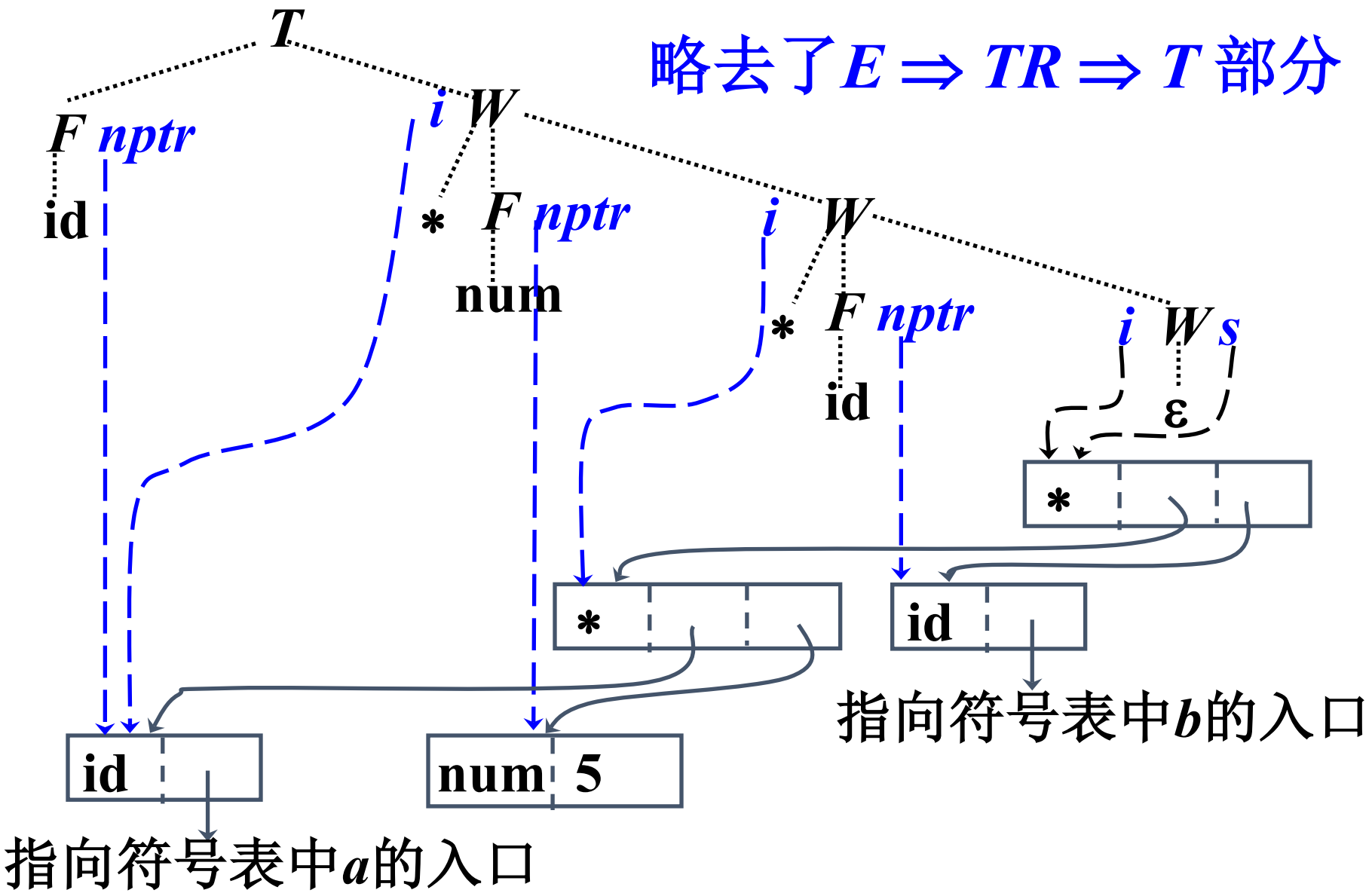
L属性定义的语法树构造



L属性定义的语法树构造

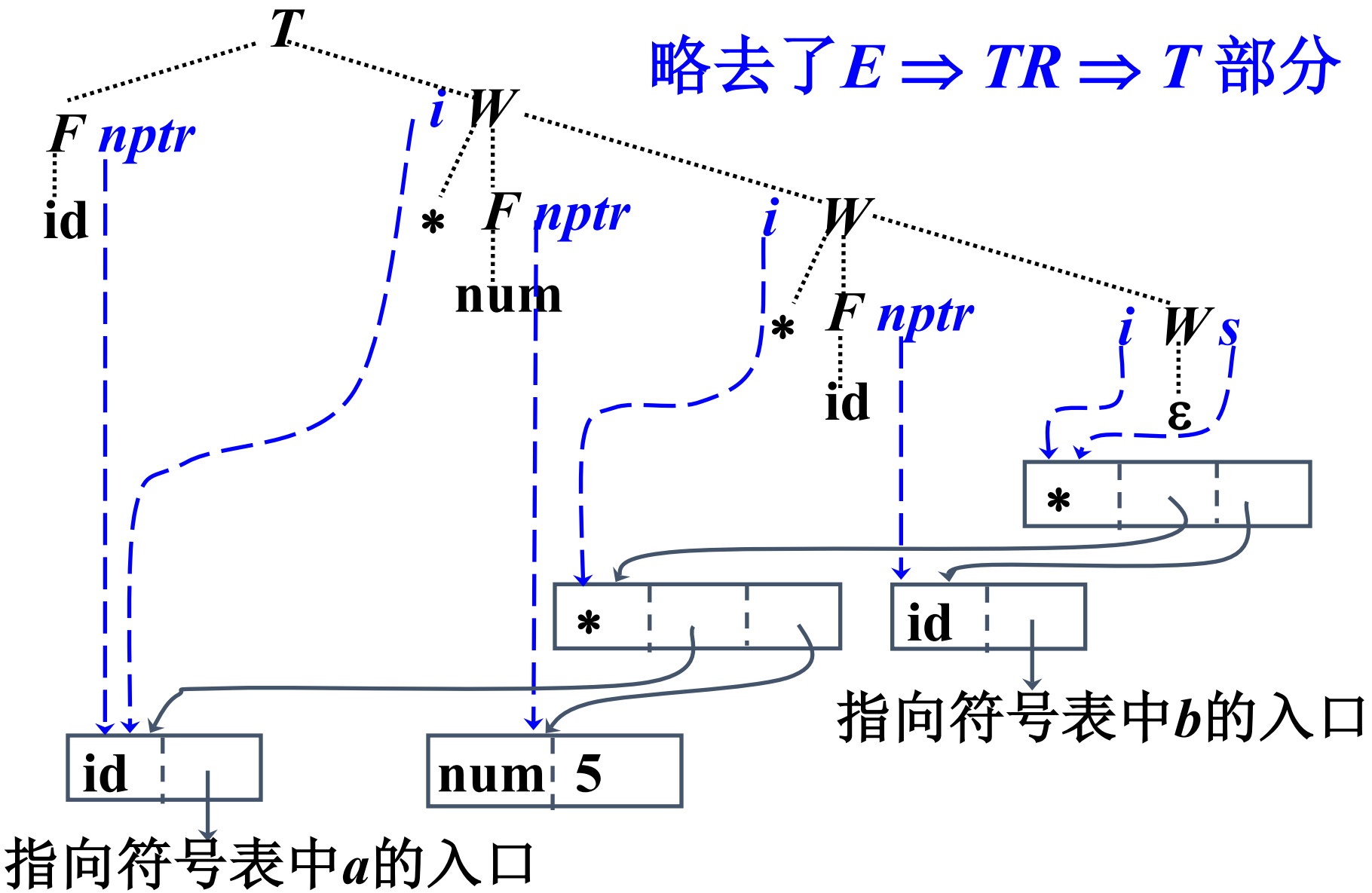


L属性定义的语法树构造



略去了$E \Rightarrow TR \Rightarrow T$部分

L属性定义的语法树构造



略去了$E \Rightarrow TR \Rightarrow T$部分

S-SDD和L-SDD对比

- **最后抽象语法树都是一样的**
- **但是，分析树却是不同的**
  - S-SDD的分析树与抽象语法树比较接近
  - L-SDD的分析树与抽象语法树结构不同

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

2024年9月30日