语义分析

魏恒峰

hfwei@nju.edu.cn

2020 年 12 月 10 日

Donald Knuth

Semantics of Context-Free Languages

by

DONALD E. KNUTH
California Institute of Technology

ABSTRACT

"Meaning" may be assigned to a string in a context-free language by defining "attributes" of the symbols in a derivation tree for that string. The attributes can be defined by functions associated with each production in the grammar. This paper examines the implications of this process when some of the attributes are "synthesized", i.e., defined solely in terms of attributes of the *descendants* of the corresponding nonterminal symbol, while other attributes are "inherited", i.e., defined in terms of attributes of the *ancestors* of the nonterminal symbol. An algorithm is given which detects when such semantic rules could possibly lead to circular definition of some attributes. An example is given of a simple programming language defined with both inherited and synthesized attributes, and the method of definition is compared to other techniques for formal specification of semantics which have appeared in the literature.

属性文法 (Attribute Grammar): 为上下文无关文法赋予**语义**

关键问题：如何基于上下文无关文法做上下文相关分析？

语法分析树上的**有序**信息流动

一对概念

两类属性定义

三种实现方式

四大应用

表达式求值

类型系统 (语义分析)

抽象语法树

后缀表达式 (中间代码生成)

Definition (语法制导定义 (Syntax-Directed Definition; SDD))

SDD 是一个上下文无关文法和**属性**及**规则**的结合。

每个文法符号都可以关联多个属性

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

每个产生式都可以关联一组规则

Definition (语法制导定义 (Syntax-Directed Definition; SDD))

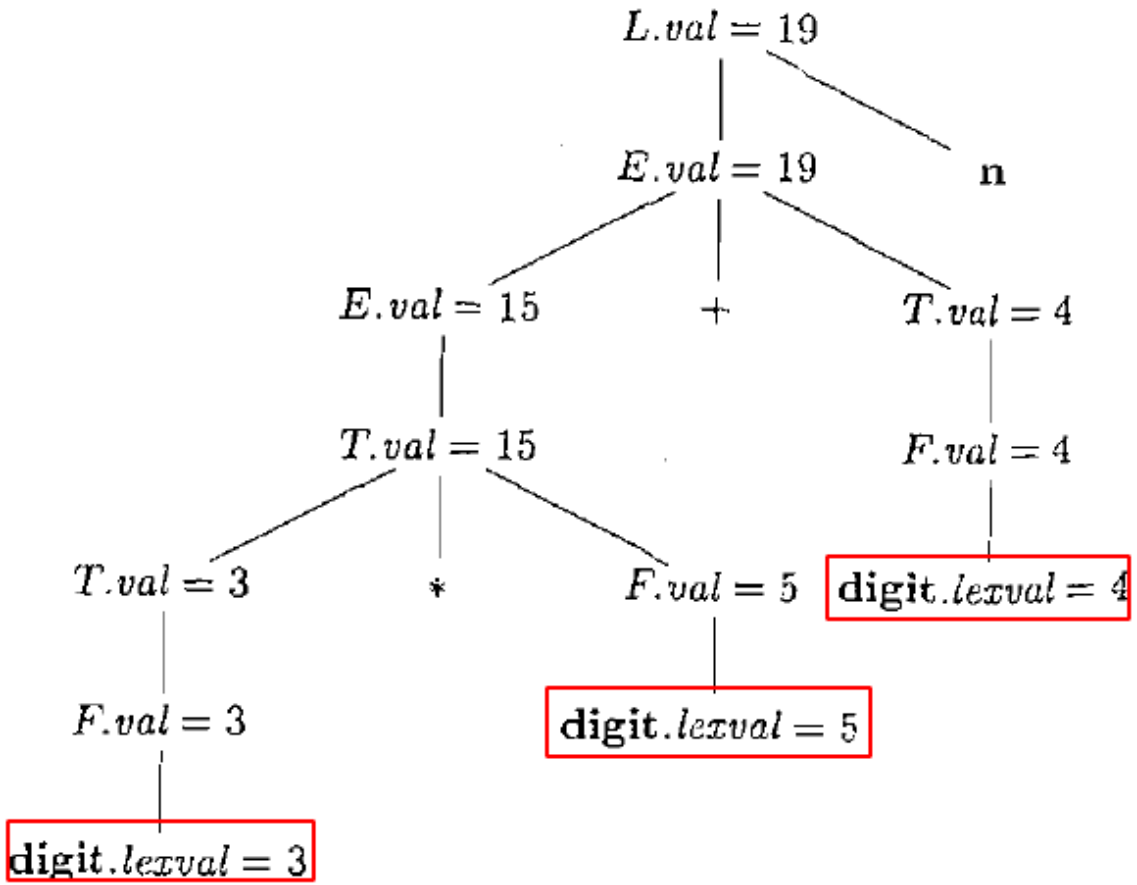
SDD 是一个上下文无关文法和**属性**及**规则**的结合。

SDD **唯一确定**了语法分析树上每个非终结符节点的属性值

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

SDD **没有**规定以什么方式、什么顺序计算这些属性值

注释（annotated）语法分析树：显示了各个属性值的语法分析树



$3 * 5 + 4$

Definition (综合属性 (Synthesized Attribute))

节点 N 上的**综合属性**只能通过 N 的子节点或 N 本身的属性来定义。

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

Definition (综合属性 (Synthesized Attribute))

节点 N 上的**综合属性**只能通过 N 的子节点或 N 本身的属性来定义。

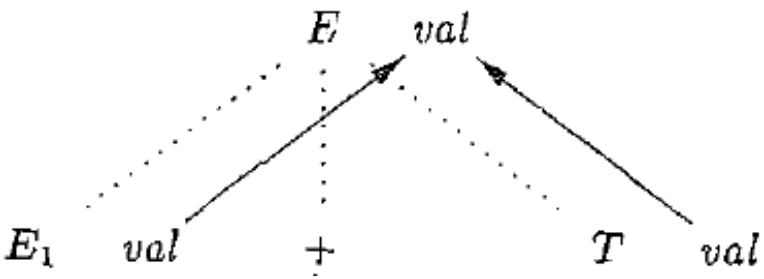
产生式	语义规则
1) $L \to E$ n	$L.val = E.val$
2) $E \to E_1 + T$	$E.val = E_1.val + T.val$
3) $E \to T$	$E.val = T.val$
4) $T \to T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \to F$	$T.val = F.val$
6) $F \to (E)$	$F.val = E.val$
7) $F \to$ **digit**	$F.val =$ **digit**.lexval

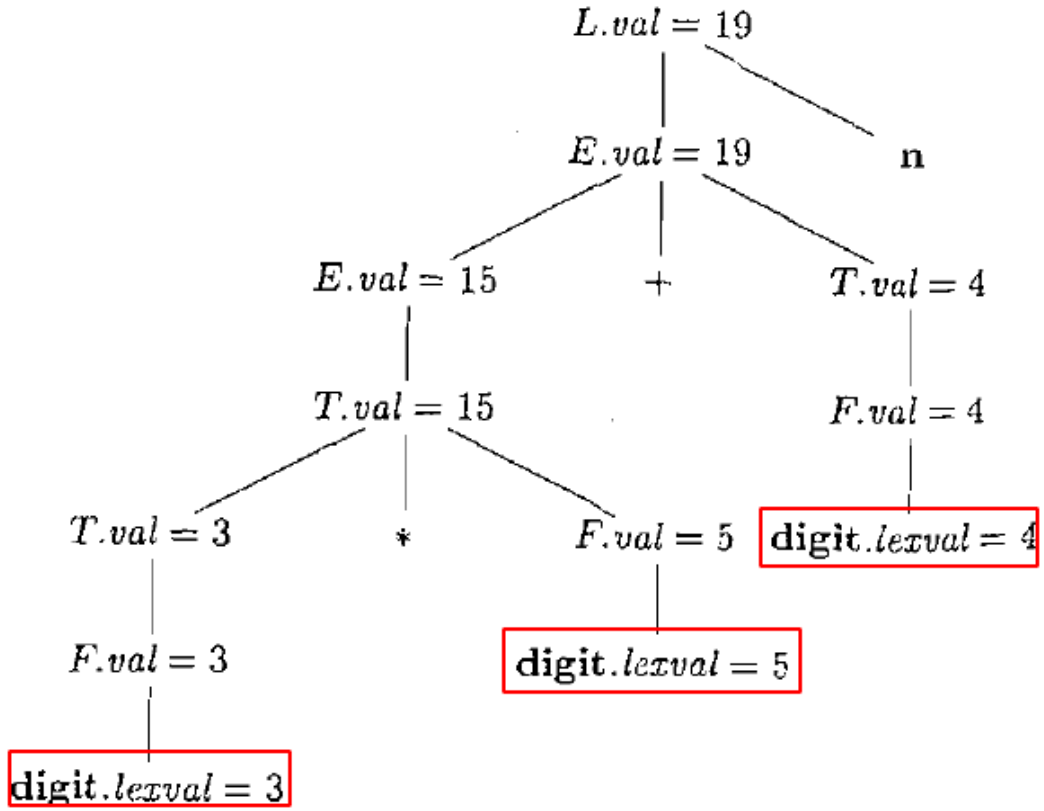
Definition (S 属性定义 (S-Attributed Definition))

如果一个 SDD 的每个属性都是综合属性, 则它是 S 属性定义。

依赖图用于确定一棵给定的语法分析树中各个属性实例之间的依赖关系

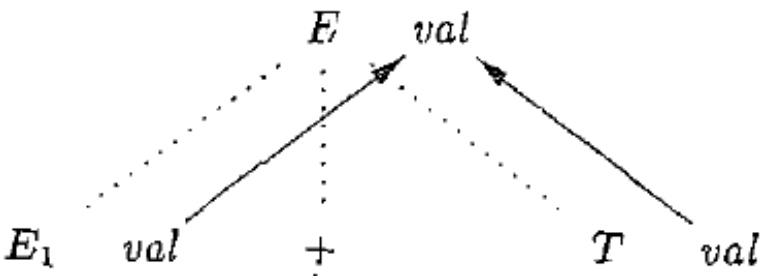
产生式	语义规则
1) $L \to E$ n	$L.val = E.val$
2) $E \to E_1 + T$	$E.val = E_1.val + T.val$
3) $E \to T$	$E.val = T.val$
4) $T \to T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \to F$	$T.val = F.val$
6) $F \to (E)$	$F.val = E.val$
7) $F \to$ digit	$F.val =$ digit.lexval

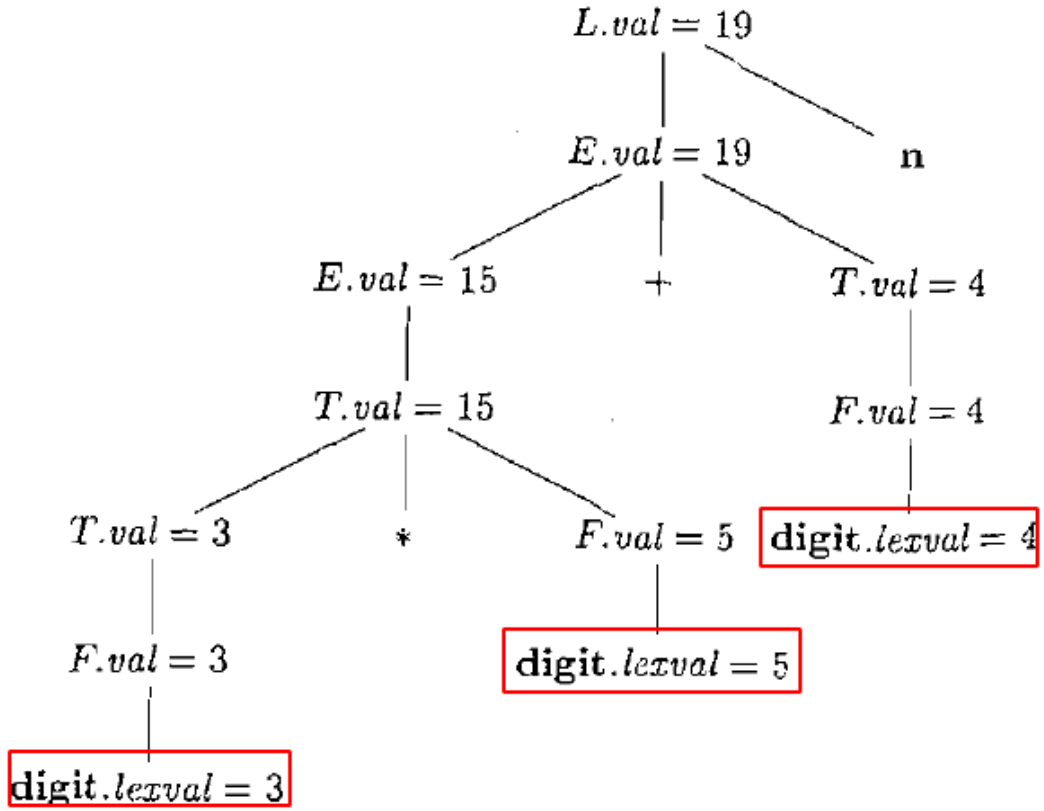




依赖图用于确定一棵给定的语法分析树中各个属性实例之间的依赖关系

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





S **属性定义**的依赖图描述了属性实例之间**自底向上**的信息流

S **属性定义**的依赖图描述了属性实例之间**自底向上**的信息流

因此, 此类属性值的计算可以自然地在自底向上的语法分析**过程中**实现

S **属性定义**的依赖图描述了属性实例之间**自底向上**的信息流

因此, 此类属性值的计算可以自然地在自底向上的语法分析**过程中**实现

当 LR 语法分析器进行**归约**时, 计算相应节点的综合属性值

S **属性定义**的依赖图描述了属性实例之间**自底向上**的信息流

此类属性值的计算也可以在自顶向下的语法分析**过程中**实现

S **属性定义**的依赖图描述了属性实例之间**自底向上**的信息流

此类属性值的计算也可以在自顶向下的语法分析**过程中**实现

在 LL 语法分析器中, 递归下降函数 A**返回**时,
计算相应节点 A 的综合属性值

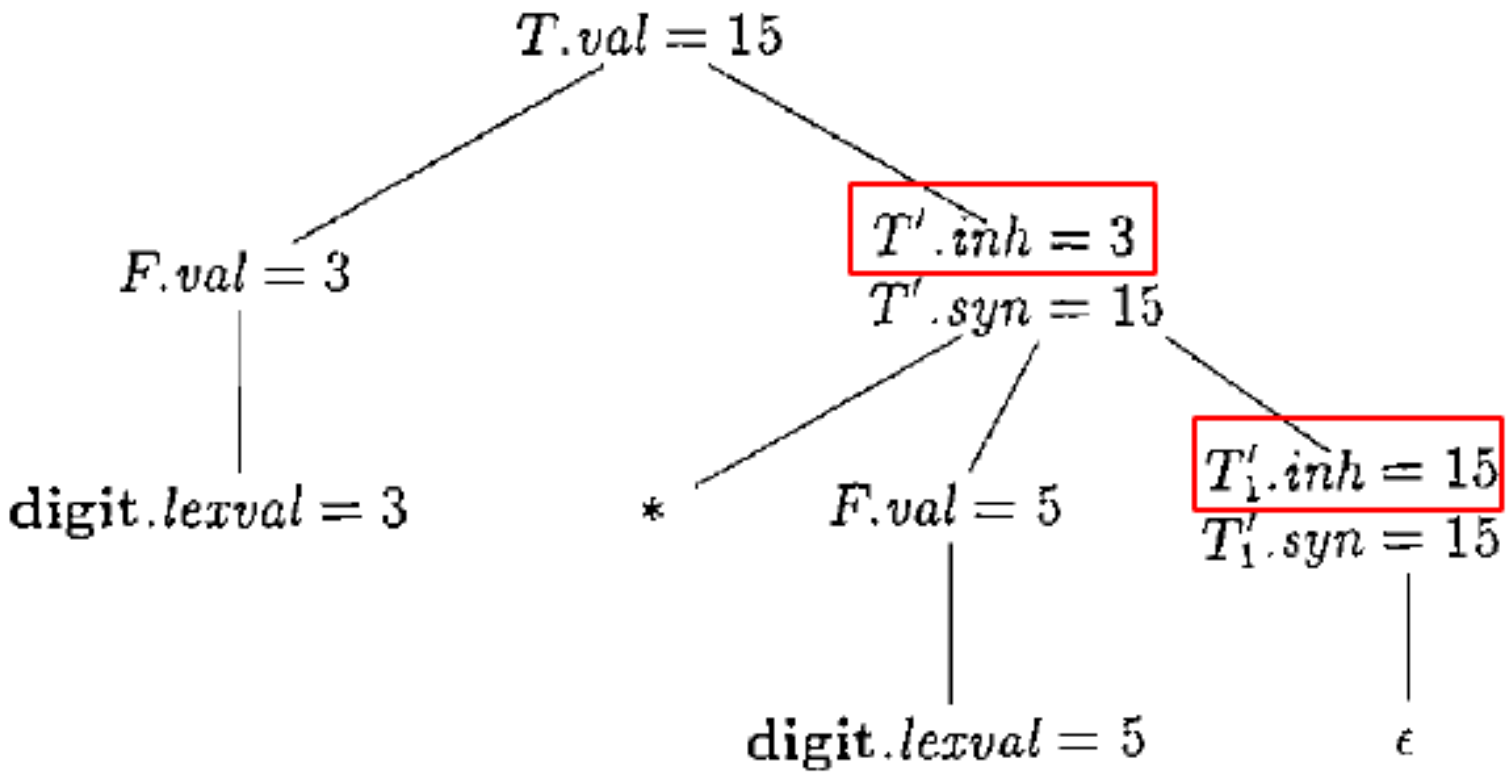
T' 有一个综合属性 syn 与一个继承属性 inh

产生式	语义规则
1) $T \to F\ T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \to *\ F\ T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3) $T' \to \epsilon$	$T'.syn = T'.inh$
4) $F \to \mathbf{digit}$	$F.val = \mathbf{digit}.lexval$

Definition (继承属性 (Inherited Attribute))

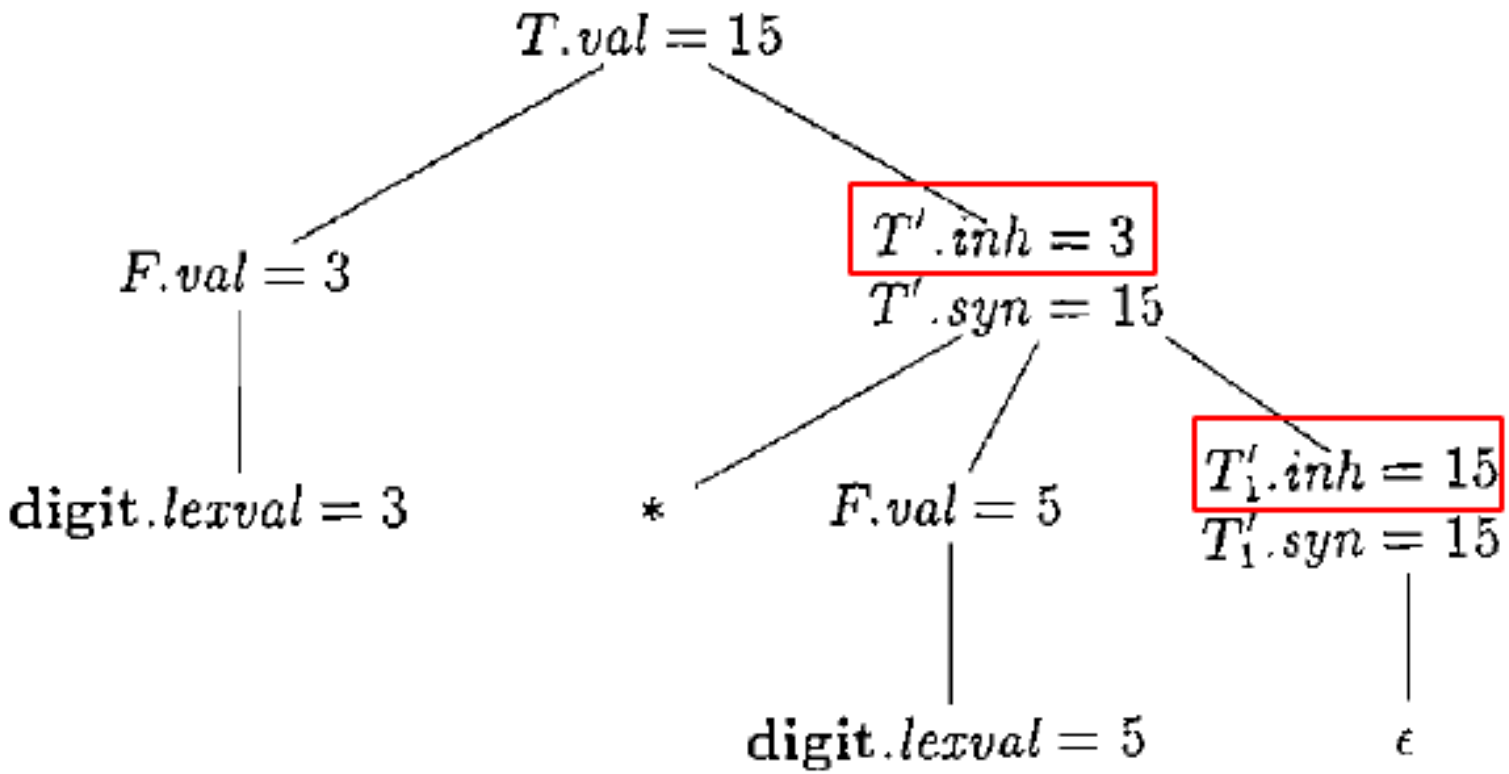
节点 N 上的**继承属性**只能通过N 的父节点、N 本身和 N 的兄弟节点上的属性来定义。

继承属性 $T'.inh$ 用于在表达式中从左向右传递中间计算结果



$3 * 5$

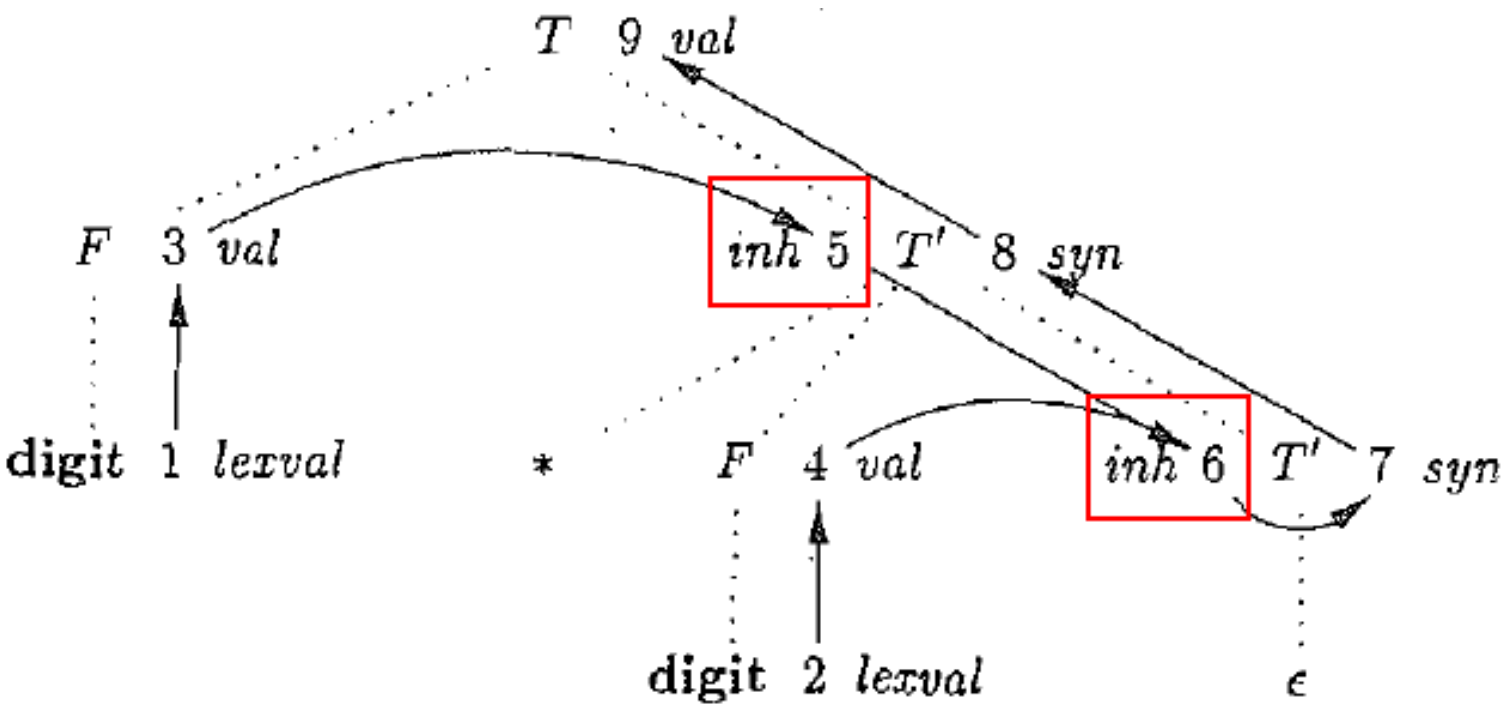
继承属性 $T'.inh$ 用于在表达式中从左向右传递中间计算结果



$3 * 5$

在右递归文法下实现了左结合

依赖图用于确定一棵给定的语法分析树中各个属性实例之间的依赖关系



信息流向: 先从左向右、从上到下传递信息, 再从下到上传递信息

Definition (L 属性定义 (L-Attributed Definition))

如果一个 SDD 的每个属性

(1) 要么是综合属性,

(2) 要么是继承属性, 但是它的规则满足如下限制:
对于产生式 $A \to X_1 X_2 \ldots X_n$ 及其对应规则定义的继承属性 $X_i.a$,
则这个规则只能使用

(a) 和**产生式头** A关联的**继承**属性;

(b) 位于X_i **左边**的文法符号实例 X_1、X_2、...、X_{i-1} 相关的**继承**属性或**综合**属性;

(c) 和**这个** X_i **的实例本身**相关的继承属性或综合属性, 但是在由这个 X_i 的全部属性组成的依赖图中**不存在环**。

则它是 L 属性定义。

非 L 属性定义

产生式	语义规则
$A \to B\ C$	$A.s = B.b;$ $B.i = f(C.c, A.s)$

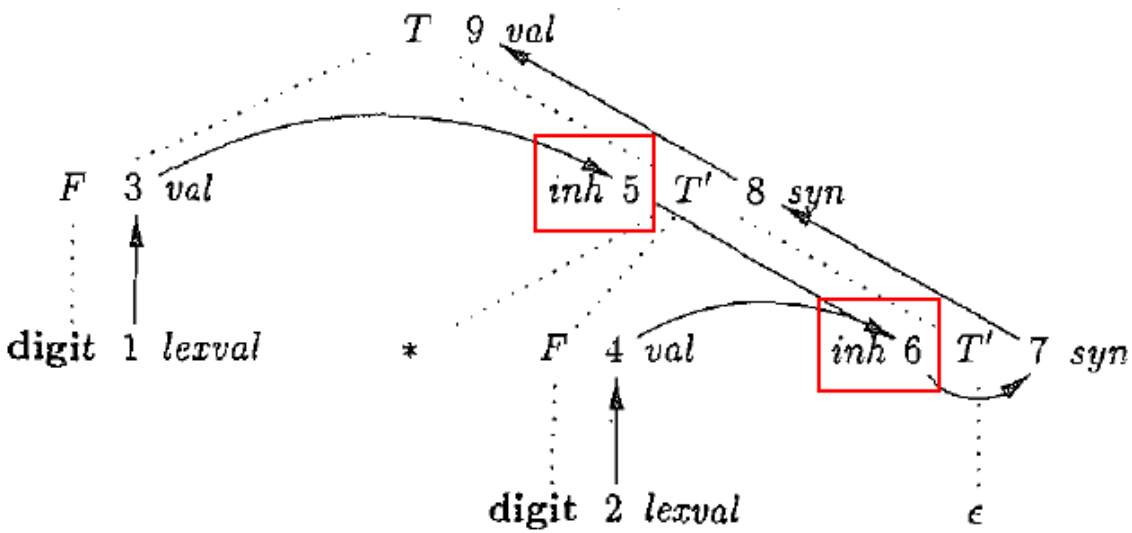
作为继承属性, $B.i$ 依赖了右边的 $C.c$ 属性与头部 $A.s$ 综合属性

L 属性文法：依赖图是无环的

产生式	语义规则
1) $T \to F\, T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \to * F\, T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3) $T' \to \epsilon$	$T'.syn = T'.inh$
4) $F \to \mathbf{digit}$	$F.val = \mathbf{digit}.lexval$

L 属性文法：依赖图是无环的

产生式	语义规则
1) $T \to F\ T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \to *\ F\ T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3) $T' \to \epsilon$	$T'.syn = T'.inh$
4) $F \to$ **digit**	$F.val =$ **digit**.*lexval*



(左递归) S 属性文法 $\Longrightarrow$ (右递归) L 属性文法

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

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

仍保持了操作的左结合性

(左递归) S 属性文法

$$A \to A_1 Y \qquad A.a = g(A_1.a, Y.y)$$

$$A \to X \qquad A.a = f(X.x)$$

(左递归) S 属性文法

$$A \to A_1 Y \qquad A.a = g(A_1.a, Y.y)$$

$$A \to X \qquad A.a = f(X.x)$$

(右递归) L 属性文法

$$A \to XR \qquad R.i = f(X.x); \quad A.a = R.s$$

$$R \to YR_1 \qquad R_1.i = g(R.i, Y.y); \quad R.s = R_1.s$$

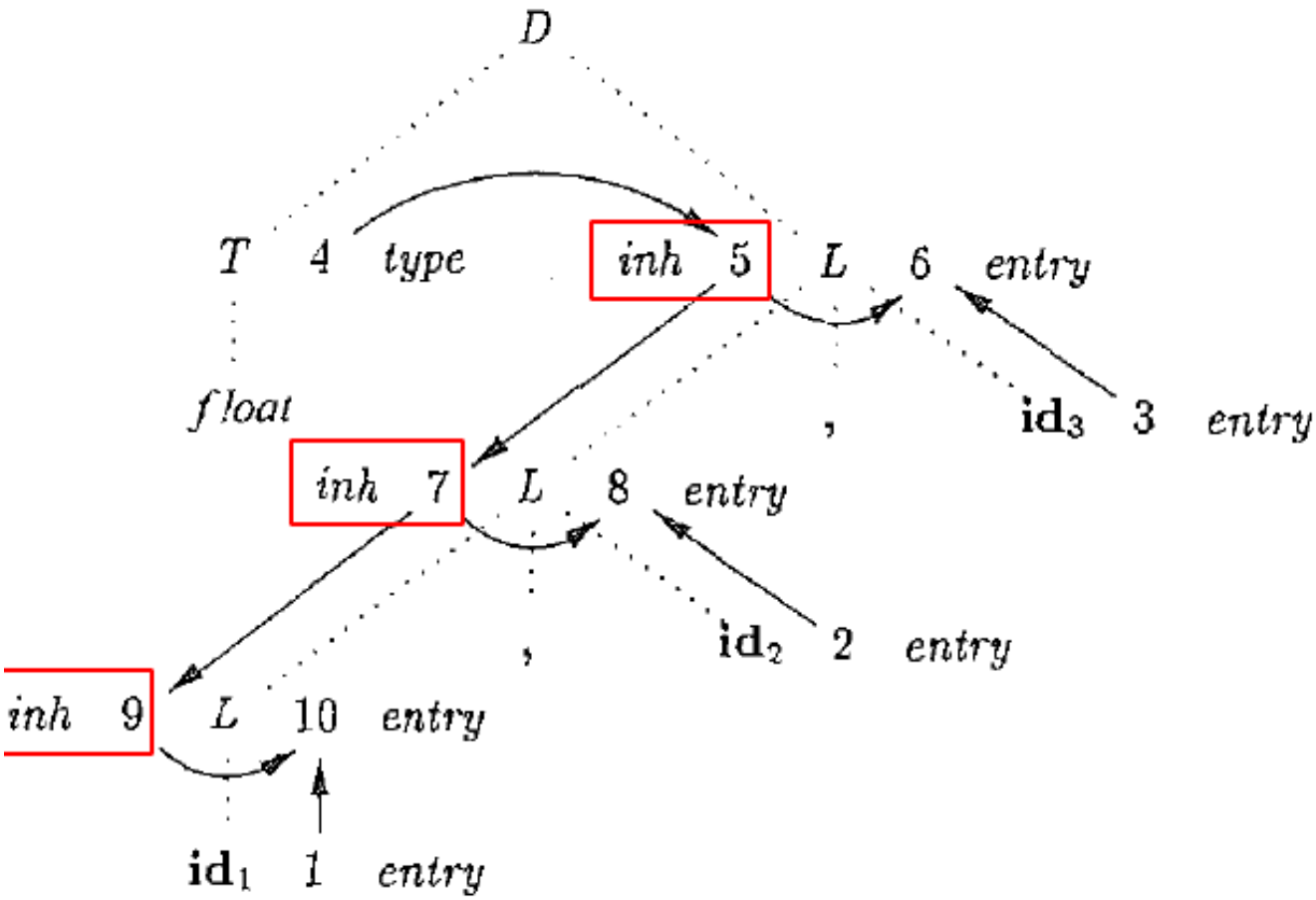
$$R \to \epsilon \qquad R.s = R.i$$

类型声明文法举例

	产生式	语义规则
1)	$D \rightarrow T\ L$	$L.inh = T.type$
2)	$T \rightarrow \mathbf{int}$	$T.type = \text{integer}$
3)	$T \rightarrow \mathbf{float}$	$T.type = \text{float}$
4)	$L \rightarrow L_1\ ,\ \mathbf{id}$	$L_1.inh = L.inh$ $addType(\mathbf{id}.entry, L.inh)$
5)	$L \rightarrow \mathbf{id}$	$addType(\mathbf{id}.entry, L.inh)$

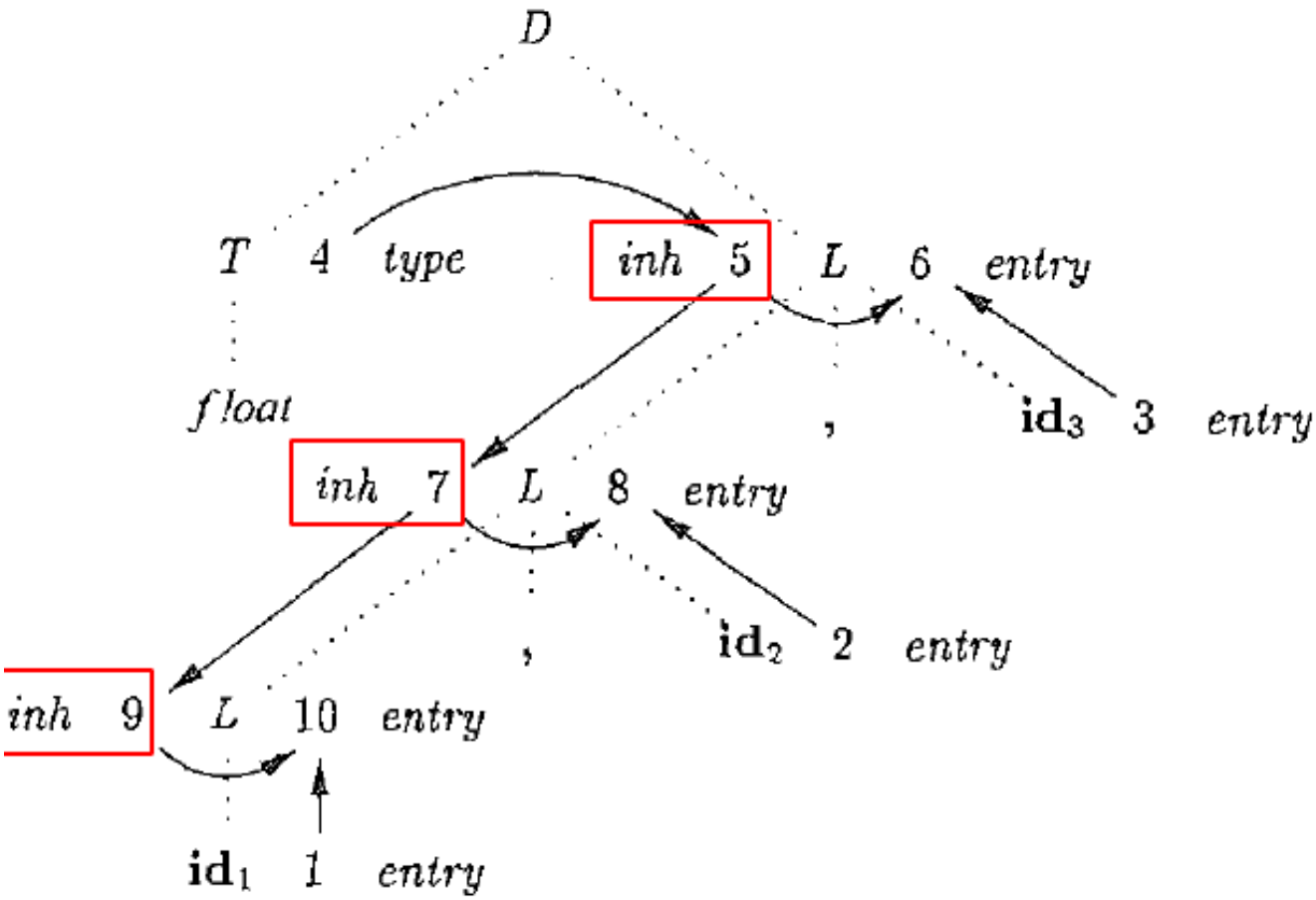
float $\mathbf{id}_1, \mathbf{id}_2, \mathbf{id}_3$

L.inh 将声明的类型沿着标识符列表向下传递, 并被加入到符号表中



float $\mathbf{id}_1, \mathbf{id}_2, \mathbf{id}_3$

L.inh 将声明的类型沿着标识符列表向下传递, 并被加入到符号表中



float $\mathbf{id}_1, \mathbf{id}_2, \mathbf{id}_3$

*addType()*是一种受控的副作用, 使用全局符号表, 更易实现

数组类型文法举例

产生式	语义规则
$T \rightarrow B\ C$	$T.t = C.t$
	$C.b = B.t$
$B \rightarrow \mathbf{int}$	$B.t = integer$
$B \rightarrow \mathbf{float}$	$B.t = float$
$C \rightarrow [\ \mathbf{num}\ ]\ C_1$	$C.t = array(\mathbf{num}.val,\ C_1.t)$
	$C_1.b = C.b$
$C \rightarrow \epsilon$	$C.t = C.b$

int[2][3]

数组类型文法举例

产生式	语义规则
$T \rightarrow B\ C$	$T.t = C.t$
	$C.b = B.t$
$B \rightarrow \mathbf{int}$	$B.t = integer$
$B \rightarrow \mathbf{float}$	$B.t = float$
$C \rightarrow [\ \mathbf{num}\ ]\ C_1$	$C.t = array(\mathbf{num}.val,\ C_1.t)$
	$C_1.b = C.b$
$C \rightarrow \epsilon$	$C.t = C.b$

int[2][3]

语义规则用以生成**类型表达式** $array(2, array(3, integer))$

数组类型文法举例

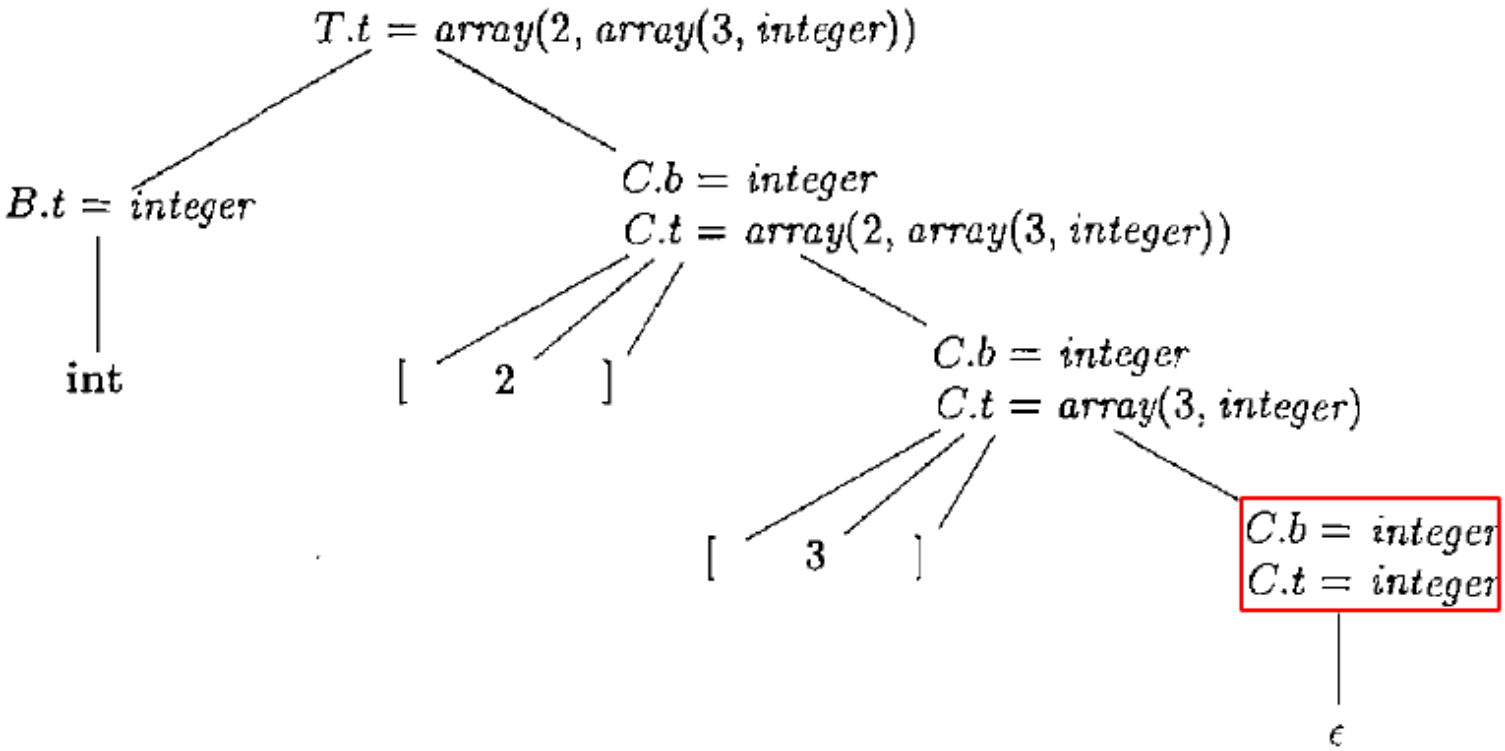
产生式	语义规则
$T \rightarrow B\ C$	$T.t = C.t$
	$C.b = B.t$
$B \rightarrow$ int	$B.t = integer$
$B \rightarrow$ float	$B.t = float$
$C \rightarrow [$ num $]\ C_1$	$C.t = array(\mathbf{num}.val,\ C_1.t)$
	$C_1.b = C.b$
$C \rightarrow \epsilon$	$C.t = C.b$

int[2][3]

语义规则用以生成**类型表达式** $array(2, array(3, integer))$

[ ]是右结合的

继承属性 $C.b$ 将一个基本类型沿着树向下传播



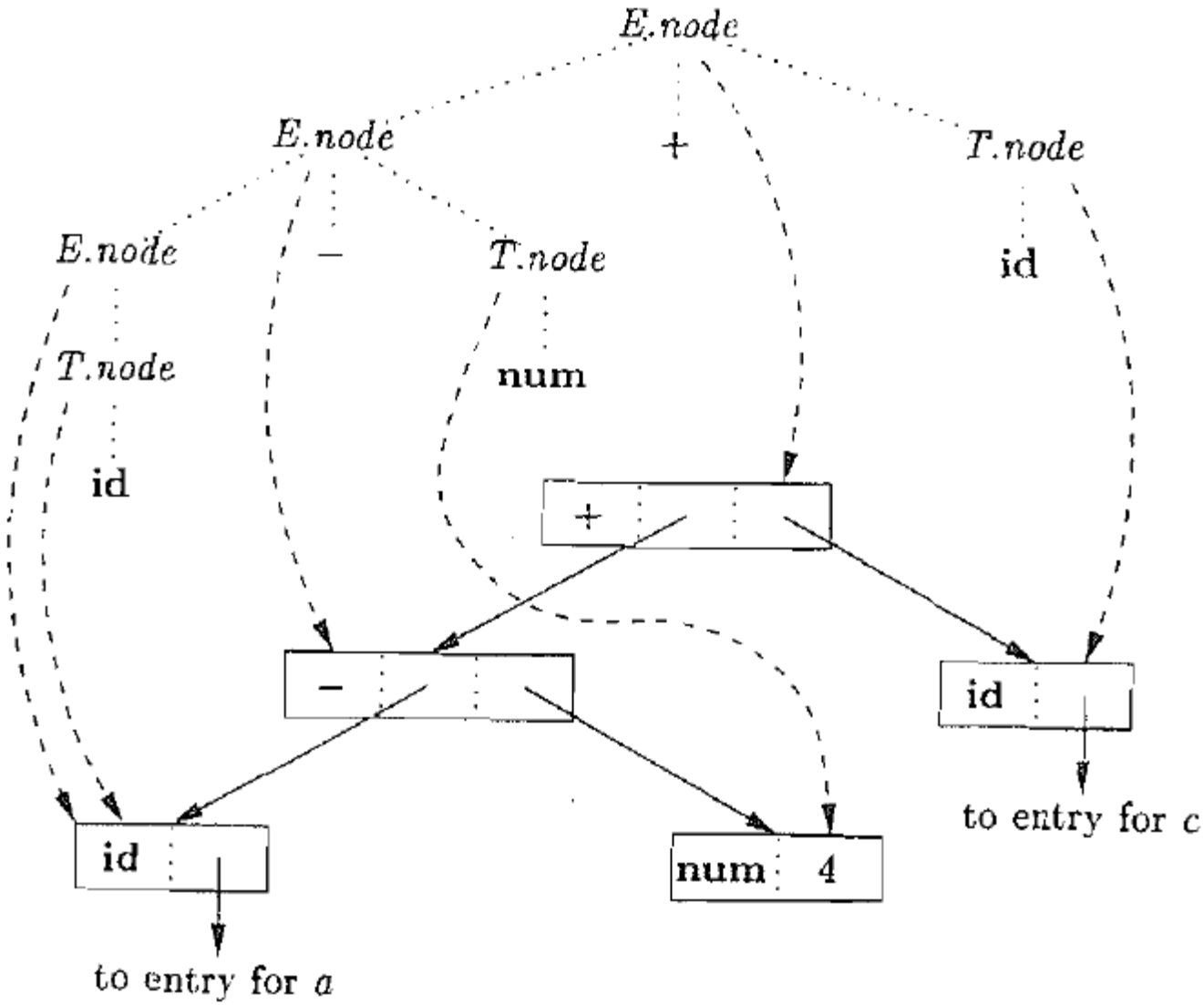
int[2][3]

综合属性 $C.t$ 收集最终得到的类型表达式

表达式的**抽象语法树** S 属性定义

	产生式	语义规则
1)	$E \to E_1 + T$	$E.node = \textbf{new}\ Node('+', E_1.node, T.node)$
2)	$E \to E_1 - T$	$E.node = \textbf{new}\ Node('-', E_1.node, T.node)$
3)	$E \to T$	$E.node = T.node$
4)	$T \to (\ E\ )$	$T.node = E.node$
5)	$T \to \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
6)	$T \to \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$

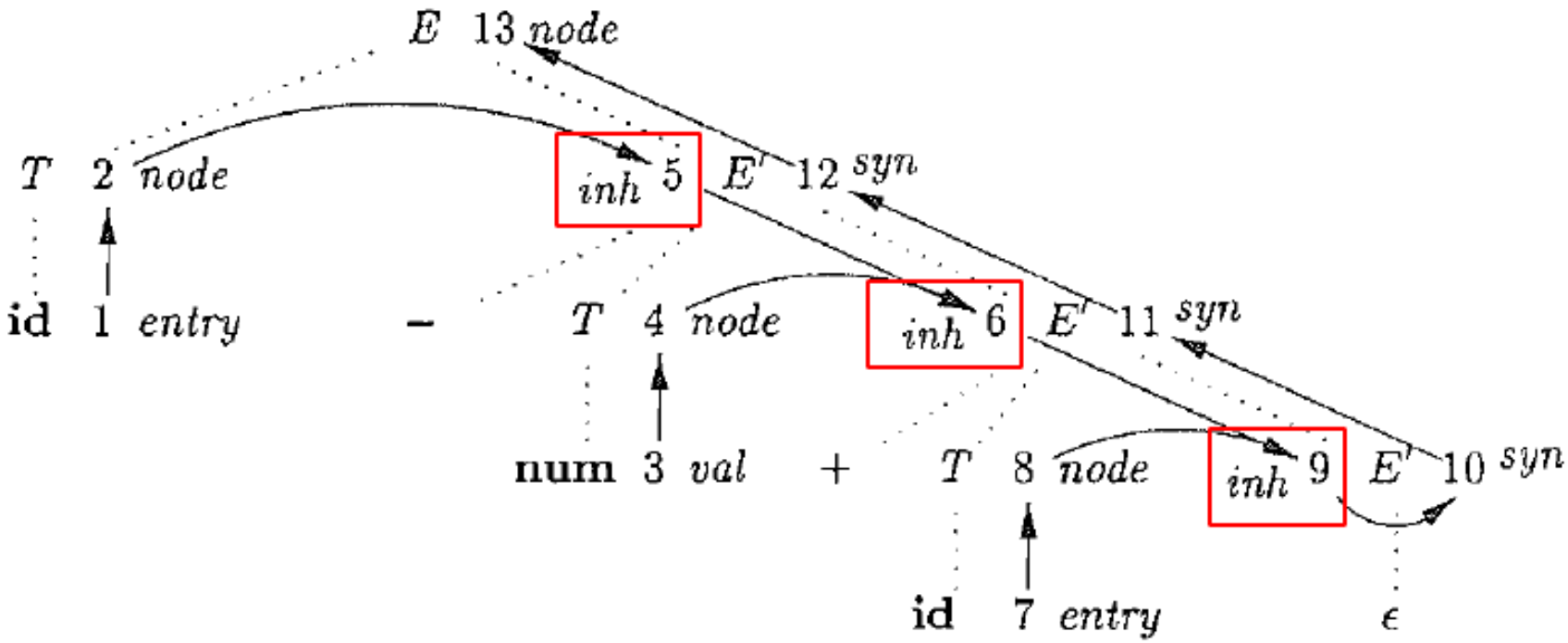
抽象语法树: 丢弃非本质的东西, 仅保留重要结构信息



$a - 4 + c$

表达式的**抽象语法树** L 属性定义

产生式	语义规则
1) $E \to T\ E'$	$E.node = E'.syn$ $E'.inh = T.node$
2) $E' \to +\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('+', E'.inh, T.node)$ $E'.syn = E'_1.syn$
3) $E' \to -\ T\ E'_1$	$E'_1.inh = \textbf{new}\ Node('-', E'.inh, T.node)$ $E'.syn = E'_1.syn$
4) $E' \to \epsilon$	$E'.syn = E'.inh$
5) $T \to (\ E\ )$	$T.node = E.node$
6) $T \to \textbf{id}$	$T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$
7) $T \to \textbf{num}$	$T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$


E 13 node
T 2 node
inh 5
E' 12 syn
id 1 entry
−
T 4 node
inh 6
E' 11 syn
num 3 val
+
T 8 node
inh 9
E' 10 syn
id 7 entry
ϵ

$$a - 4 + c$$

Definition (后缀表示 (Postfix Notation))

(1) 如果 E 是一个变量或常量, 则 E 的后缀表示是 E 本身;

(2) 如果 E 是形如 E_1 **op** E_2 的表达式, 则 E 的后缀表示是 $E_1' E_2'$**op**, 这里 E_1' 和 E_2' 分别是 E_1 与 E_2 的后缀表达式;

(3) 如果 E 是形如 (E_1) 的表达式, 则 E 的后缀表示是 E_1 的后缀表示。

Definition (后缀表示 (Postfix Notation))

(1) 如果 E 是一个变量或常量, 则 E 的后缀表示是 E 本身;

(2) 如果 E 是形如 E_1 **op** E_2 的表达式, 则 E 的后缀表示是 $E_1' E_2'$**op**, 这里 E_1' 和 E_2' 分别是 E_1 与 E_2 的后缀表达式;

(3) 如果 E 是形如 (E_1) 的表达式, 则 E 的后缀表示是 E_1 的后缀表示。

$$(9-5)+2 \implies 95-2+$$

$$9-(5+2) \implies 952+-$$

Definition (后缀表示 (Postfix Notation))

(1) 如果 E 是一个变量或常量, 则 E 的后缀表示是 E 本身;

(2) 如果 E 是形如 E_1 **op** E_2 的表达式, 则 E 的后缀表示是 $E_1' E_2'$**op**, 这里 E_1' 和 E_2' 分别是 E_1 与 E_2 的后缀表达式;

(3) 如果 E 是形如 (E_1) 的表达式, 则 E 的后缀表示是 E_1 的后缀表示。

$$(9-5)+2 \implies 95-2+$$

$$9-(5+2) \implies 952+-$$

$$952+-3* \implies$$

Definition (后缀表示 (Postfix Notation))

(1) 如果 E 是一个变量或常量, 则 E 的后缀表示是 E 本身;

(2) 如果 E 是形如 E_1 **op** E_2 的表达式, 则 E 的后缀表示是 $E_1' E_2'$**op**, 这里 E_1' 和 E_2' 分别是 E_1 与 E_2 的后缀表达式;

(3) 如果 E 是形如 (E_1) 的表达式, 则 E 的后缀表示是 E_1 的后缀表示。

$$(9-5)+2 \implies 95-2+$$

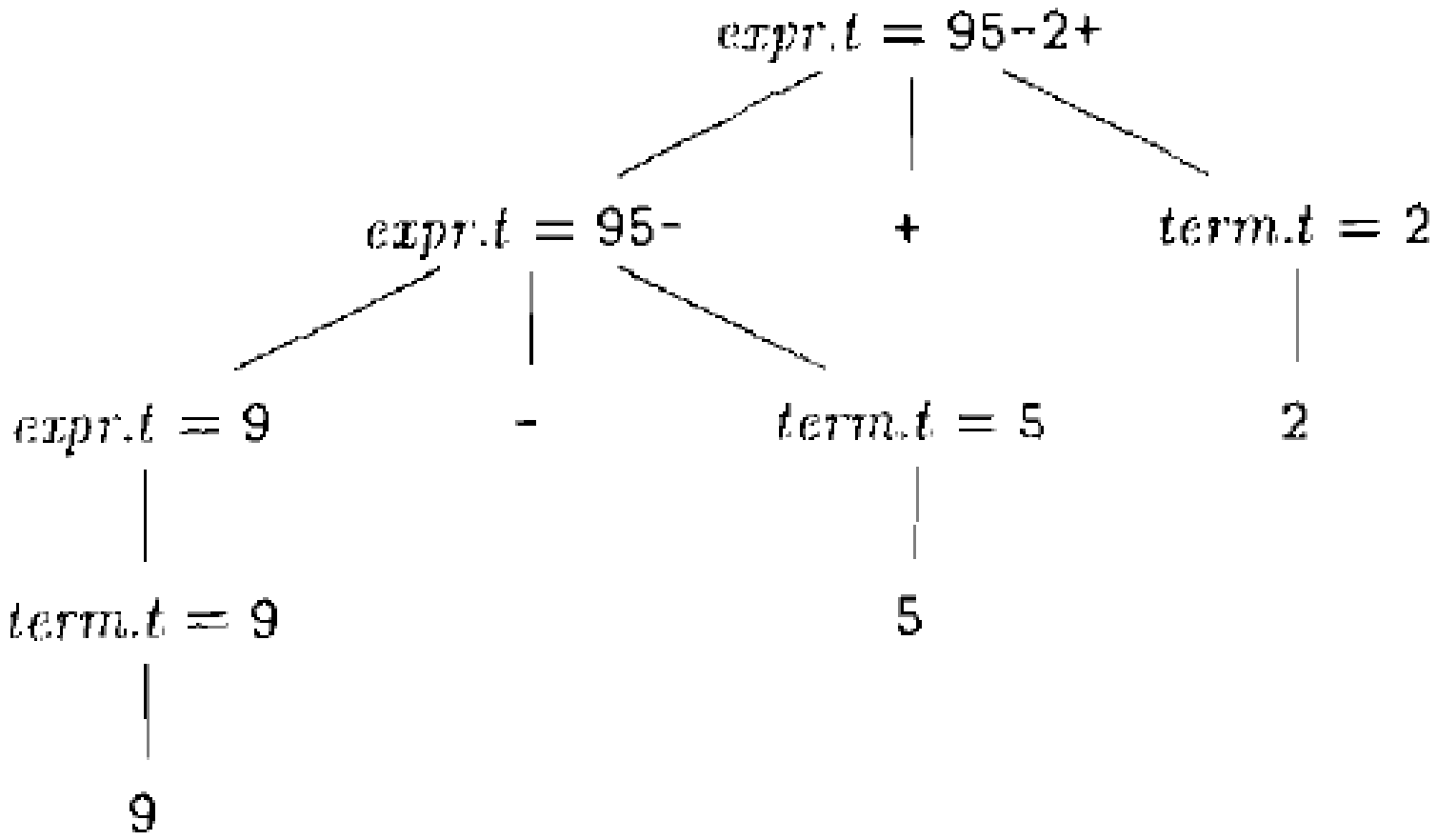
$$9-(5+2) \implies 952+-$$

$$952+-3* \implies (9-(5+2))*3$$

后缀表达式 S 属性文法

产生式	语义规则
$expr \rightarrow expr_1 + term$	$expr.t = expr_1.t \mid\mid term.t \mid\mid '+'$
$expr \rightarrow expr_1 - term$	$expr.t = expr_1.t \mid\mid term.t \mid\mid '-'$
$expr \rightarrow term$	$expr.t = term.t$
$term \rightarrow 0$	$term.t = '0'$
$term \rightarrow 1$	$term.t = '1'$
...	...
$term \rightarrow 9$	$term.t = '9'$

“||” 表示字符串的连接



$9 - 5 + 2$

有符号二进制数文法

$$
P = \left\{
\begin{array}{lcl}
\textit{Number} & \rightarrow & \textit{Sign List} \\
\textit{Sign} & \rightarrow & + \\
 & | & - \\
\textit{List} & \rightarrow & \textit{List Bit} \\
 & | & \textit{Bit} \\
\textit{Bit} & \rightarrow & 0 \\
 & | & 1
\end{array}
\right\}
$$

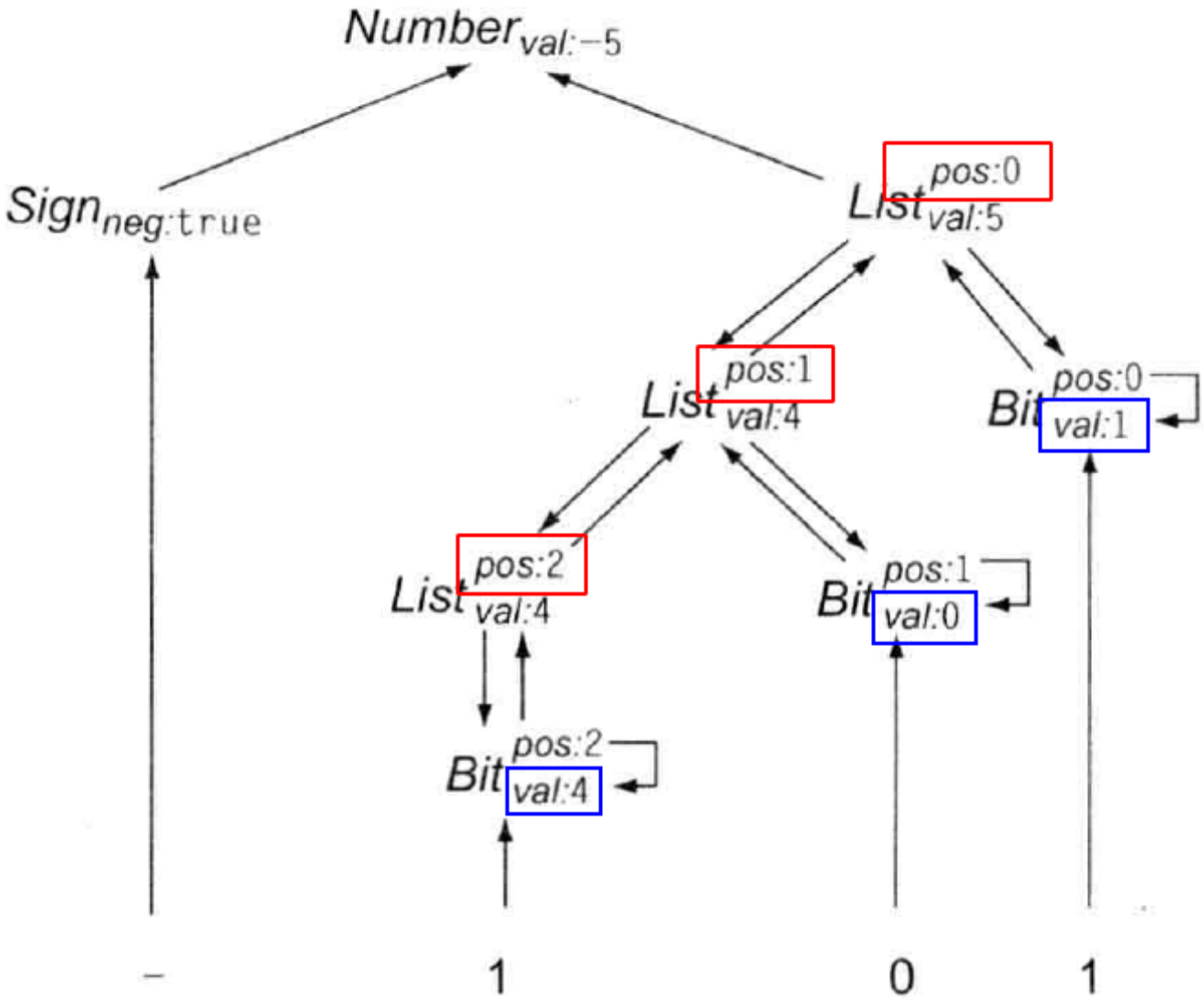
$$T = \{+, -, 0, 1\}$$

$$NT = \{\textit{Number}, \textit{Sign}, \textit{List}, \textit{Bit}\}$$

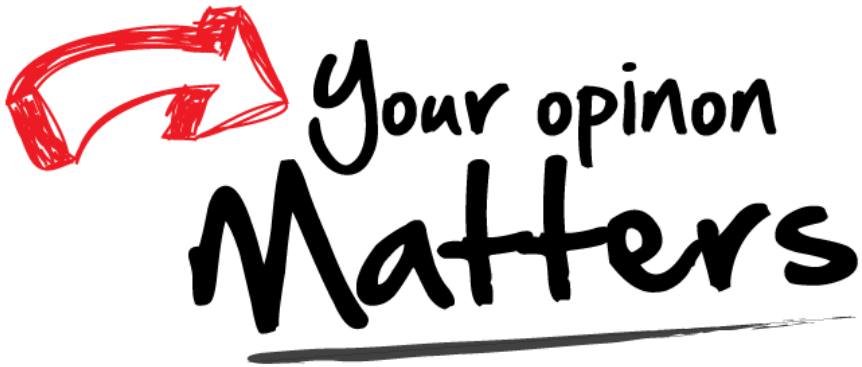
$$S = \{\textit{Number}\}$$

-101

有符号二进制数 L 属性定义




Thank
You!



Office 926

hfwei@nju.edu.cn