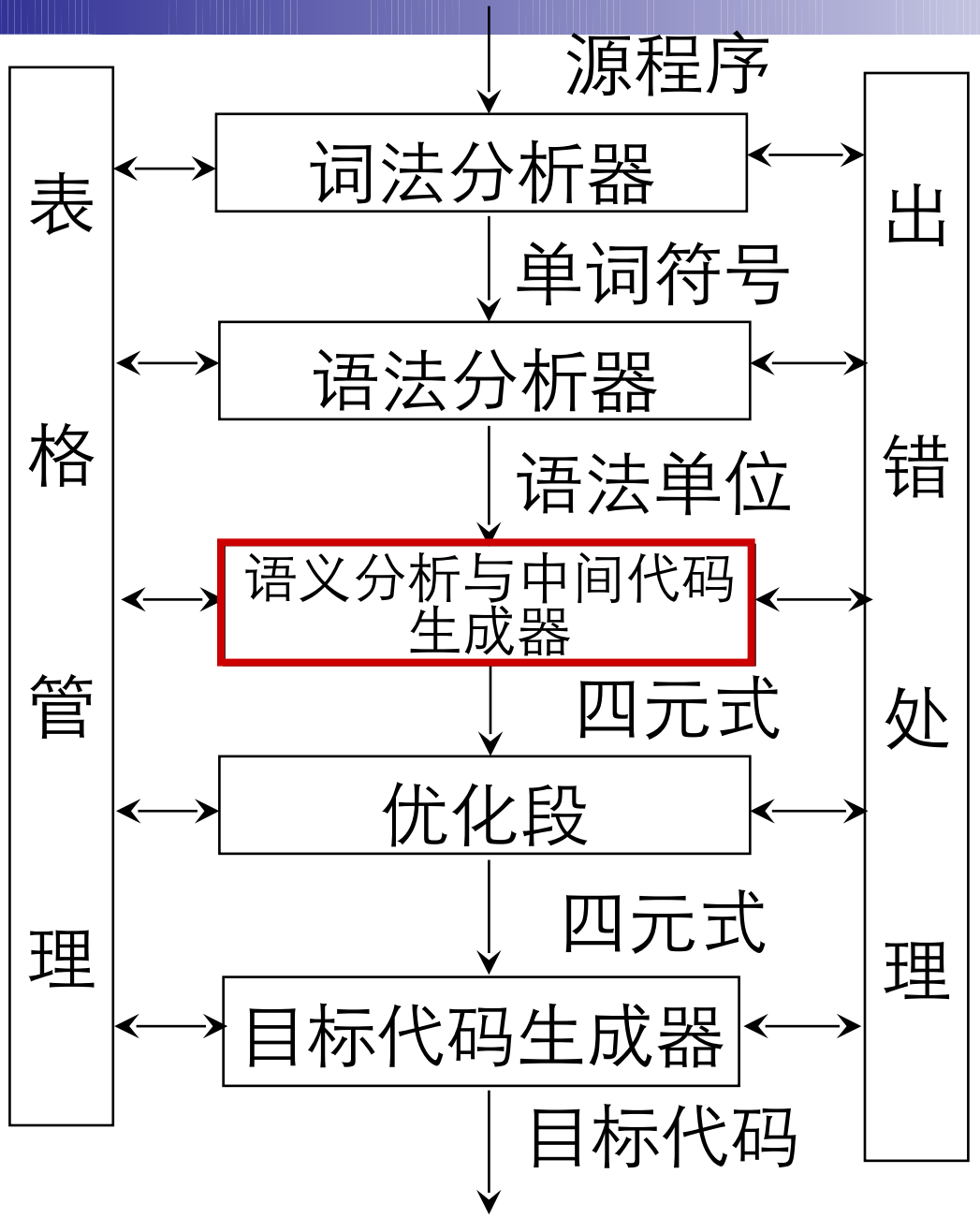
# 编译原理

## 第六章 属性文法和语法制导翻译

# 编译程序总框


源程序
词法分析器
单词符号
语法分析器
语法单位
语义分析与中间代码生成器
四元式
优化段
四元式
目标代码生成器
目标代码
表格管理
出错处理

# 第六章 属性文法和语法制导翻译

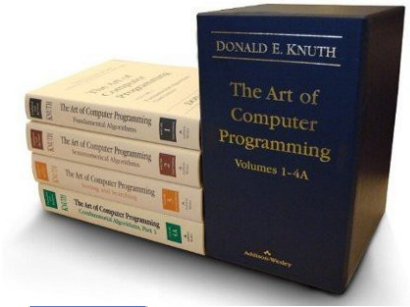
- 属性文法
- 基于属性文法的处理方法
- S- 属性文法的自下而上计算
- L- 属性文法和自顶向下翻译

# 第六章 属性文法和语法制导翻译

- 属性文法
- 基于属性文法的处理方法
- S- 属性文法的自下而上计算
- L- 属性文法和自顶向下翻译

# 6.1 属性文法

- 属性文法（也称属性翻译文法）
  - Knuth 在 1968 年提出




**The Art of Computer Programming**

...
Lexical scanning
Parsing techniques
Context-Free Languages
Compiler Techniques
...

**Donald Ervin Knuth**

TEX

# 属性文法

| 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 |

- 属性文法（也称属
  - Knuth 在 1968 年
  - 在上下文无关文法的基础上，为每个文法符号（终结符或非终结符）配备若干相关的“值”（称为属性）
    - 属性代表与文法符号相关信息，如类型、值、代码序列、符号表内容等
    - 属性可以进行计算和传递
    - 语义规则：对于文法的每个产生式都配备了一组属性的计算规则

- 属性
  - 综合属性：“自下而上”传递信息
  - 继承属性：“自上而下”传递信息
- 在一个属性文法中，对应于每个产生式 $A\rightarrow\alpha$ 都有一套与之相关联的语义规则，每条规则的形式为：

$$b:=f(c_1,c_2,\cdots,c_k)$$

这里，f 是一个函数，而且或者

1. b 是 A 的一个综合属性并且 $c_1,c_2,\cdots,c_k$ 是产生式右边文法符号的属性，或
2. b 是产生式右边某个文 $c_1,c_2,\cdots,c_k$ 是 A 或产生

- 这两种情况下：属性 b

| 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 |

# 属性文法

- 说明
  - 终结符只有综合属性，由词法分析器提供
    - F→digit
    - digit.lexval
  - 非终结符既可有综合属性也可有继承属性，文法开始符号的所有继承属性作为属性计算前的初始值
    - F→digit
    - F.val 、 digit.lexval

# 说明

- 对出现在产生式右边的继承属性和出现在产生式左边的综合属性都必须提供一个计算规则。属性计算规则中只能使用相应产生式中的文法符号的属性
  - F→digit
  - F.val :=digit.lexval
- 出现在产生式左边的继承属性和出现在产生式右边的综合属性不由所给的产生式的属性计算规则进行计算，由其它产生式的属性规则计算或者由属性计算器的参数提供
- 语义规则所描述的工作可以包括属性计算、静态语义检查、符号表操作、代码生成等等

- 例，考虑非终结符 A， B 和 C，其中，A 有一个继承属性 a 和一个综合属性 b，B 有综合属性 c，C 有继承属性 d。产生式 A→BC 可能有规则

  C.d:=B.c+1

  A.b:=A.a+B.c

而属性 A.a 和 B.c 在其它地方计算

| 产 生 式 | 语 义 规 则 |
| --- | --- |
| $L \rightarrow En$ | 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 |

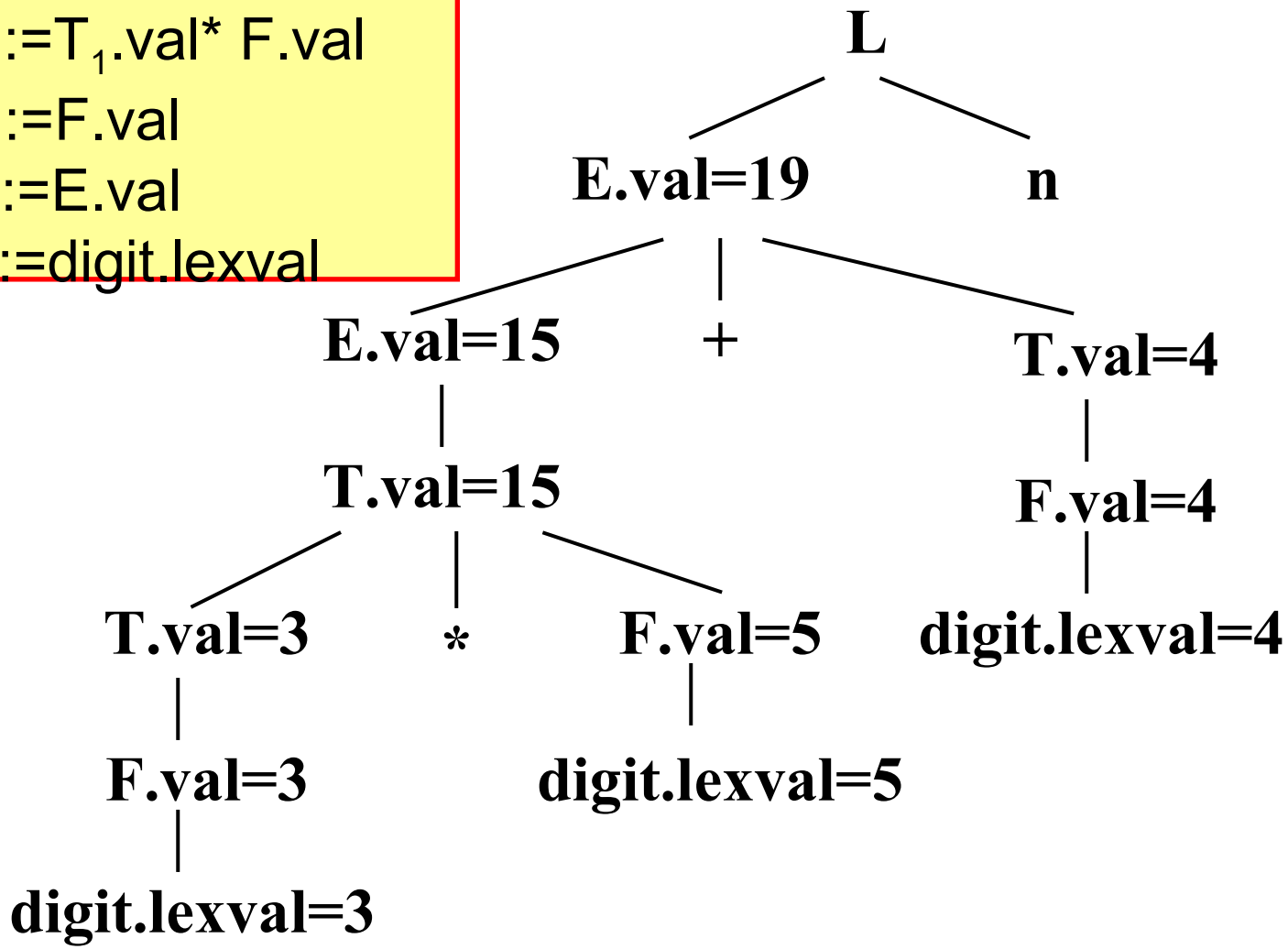
# 综合属性

- 综合属性
  - 在语法树中，一个结点的综合属性的值由其子结点和它本身的属性值确定
  - 使用自底向上的方法在每一个结点处使用语义规则计算综合属性的值
- 仅仅使用综合属性的属性文法称 S – 属性文法

| 产 生 式 | 语 义 规 则 |
| --- | --- |
| L→En | print(E.val) |
| $E \rightarrow E_1+T$ | E.val := $E_1$.val+T.val |
| E→T | E.val :=T.val |
| $T \rightarrow T_1*F$ | T.val :=$T_1$.val* F.val |
| T→F | T.val :=F.val |
| F→ (E) | F.val :=E.val |
| F→digit | F.val :=digit.lexval |

# 3*5+4n 的带注释的语法树

| 产生式 | 语义规则 |
|---|---|
| $L \rightarrow En$ | 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 |

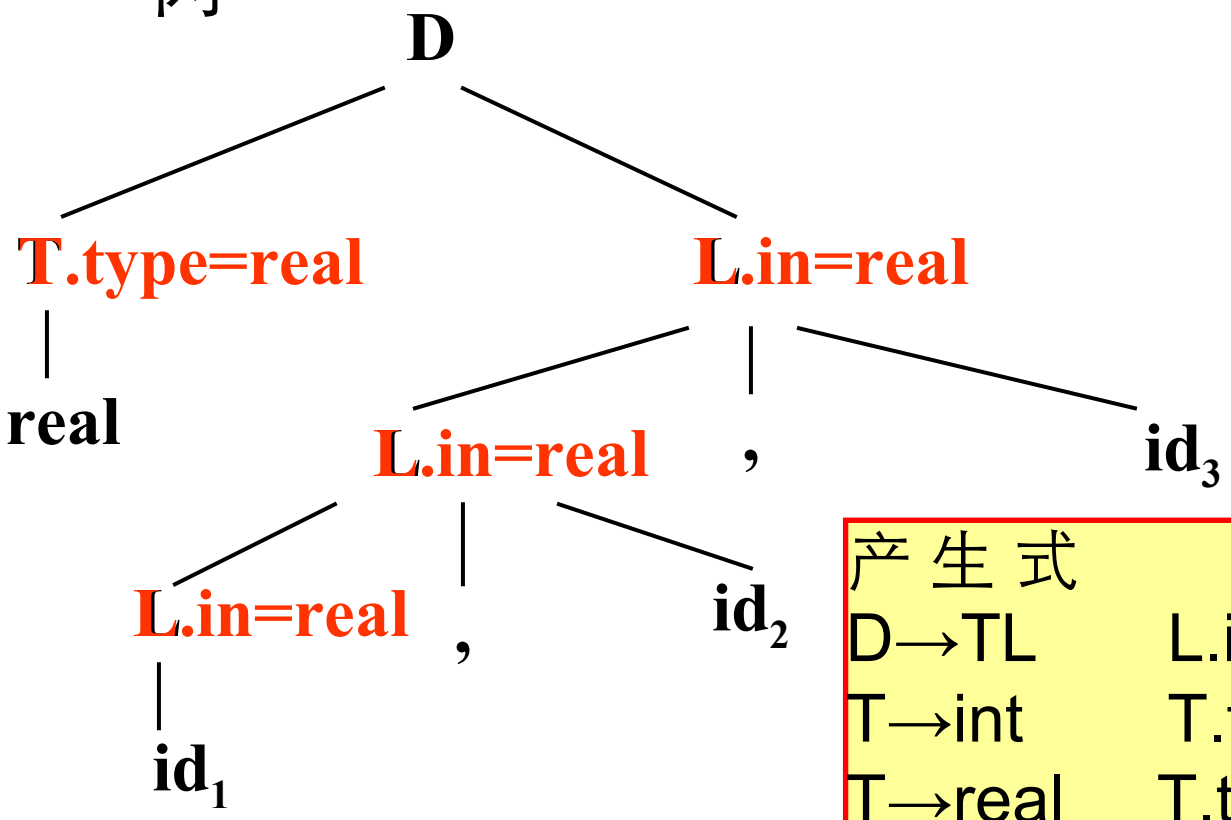


# 继承属性

- 继承属性
  - 在语法树中，一个结点的继承属性由其父结点、其兄弟结点和其本身的某些属性确定
  - 用继承属性来表示程序设计语言结构中的上下文依赖关系很方便

| 产 生 式 | 语 义 规 则 |
| --- | --- |
| D→TL | L.in := T.type |
| T→int | T.type := integer |
| T→real | T.type := real |
| $L \rightarrow L_1$, id | $L_1$.in :=L.in |
| | addtype(id.entry, L.in) |
| L→id | addtype(id.entry, L.in) |

# 句子 real $id_1$ , $id_2$ , $id_3$ 的带注释的语法树




| name | type |
|---|---|
| … | … |
| $id_3$ | real |
| $id_2$ | real |
| $id_1$ | real |

| 产生式 | 语义规则 |
|---|---|
| D→TL | L.in := T.type |
| T→int | T.type := integer |
| T→real | T.type := real |
| $L→L_1$, id | $L_1$.in :=L.in |
| | addtype(id.entry, L.in) |
| L→id | addtype(id.entry, L.in) |

# 小结

- 属性文法
- 综合属性
- 继承属性

# 作业

- P164 - 1