（续）编译知识点【5-10章】

Chapter 5 语法制导翻译

语法制导翻译在做什么

语法制导翻译把“语法分析时识别出的结构”直接用来做“翻译工作”，翻译可以是计算表达式的值、做类型检查、生成中间代码（如三地址码）、维护符号表、计算变量在栈帧里的偏移等。

做法是给文法符号（终结符和非终结符）挂上一些“属性”，属性就是你想在分析过程中携带的信息，例如 `val` 表示数值，`type` 表示类型，`place` 表示中间代码临时变量名，`offset` 表示内存偏移。**语法分析每用到一条产生式，就按这条产生式配套的“语义规则”去计算或传递这些属性。**

综合属性与继承属性

综合属性的信息流向是“从孩子到父亲”。当你在语法树上看一个非终结符节点时，它的综合属性只依赖它的子节点属性或词法单元本身的属性，所以常见于“自底向上算出来”的信息，例如表达式求值、类型推导后的结果类型、生成的代码片段等。

继承属性的信息流向是“从父亲到孩子，或从左兄弟到右兄弟”。它常见于“上下文信息往下传”的情况，例如变量声明时把“当前类型”传给后续的标识符列表，把“当前作用域的符号表指针”传给内部语句，把“当前栈帧偏移”传给后续声明来分配空间。

综合属性与继承属性经常一起用：继承属性把环境和约束传下去，综合属性把结果和产物（例如生成的代码）传上来。

一个很小的例子，用来立刻分清综合与继承

- 例子：**表达式求值通常只需要综合属性，因为值是从子表达式算到父表达式**
  - 产生式：`E → E + T | T`，`T → num`
  - `num` 的词法属性可以是 `num.lexval`（词法分析读到的数字值），然后 `T.val`、`E.val` 作为综合属性往上合成
  - 你会看到“父亲的 val 由孩子的 val 算出”，这就是综合属性的典型形态
  - 可以把语义规则写成这种形式（注意这里只是在展示规则形式）：

$$E.val = E_1.val + T.val$$

$$T.val = num.lexval$$

- 例子：**声明语句里经常需要继承属性，因为标识符列表要继承“正在声明的类型”**
  - 产生式：`D → T L`，`T → int | float`，`L → id , L | id`
  - `T.type` 可以综合得到 `int/float`，但 `L` 中的每个 `id` 需要知道自己应该被登记成什么类型，所以需要把 `T.type` 作为继承属性传给 `L.inType`，再传给每个 `id`
  - 你会看到“类型信息从父亲往下给孩子”，这就是继承属性的典型形态

S-属性定义是什么

S-属性定义指一种语法制导定义，它只允许使用综合属性，不允许使用继承属性。这样做的直接好处是：语义计算天然可以在归约时完成，因为归约就是“从右部孩子合成左部父亲”，正好符合综合属性的方向。

S-属性为什么和LR分析搭配很自然?

把它和 LR 分析结合时，最常见的实现就是“语义栈”：LR 的分析栈里不仅压状态和符号，也压与符号对应的属性值；当发生一次归约 `A → β`，就从栈顶取出 `β` 各符号的属性，按语义规则算出 `A` 的属性，再把 `A` 的属性压回去。

S-属性为什么和预测分析麻烦但也能搭配?

把它和预测分析（LL，自顶向下）结合也能做，只是 LL 在展开产生式时还没看见右部完整结果，综合属性往往要等子树完成后才能算，所以常见做法是“递归返回值”或者“在非终结符函数返回时带回综合属性”。只要不用继承属性，LL 也能做，但通常没有 LR 那么顺手。

L-属性定义是什么，它为什么强调“依赖父结点或左兄弟结点”

L-属性定义允许继承属性，但对继承属性的依赖做了限制：在产生式 `A → X1 X2 ... Xn` 中，任意 `Xi` 的继承属性只能依赖 `A` 的继承属性以及它左边符号 `X1...X(i-1)` 的属性，不能依赖右边还没处理到的符号 `X(i+1)...Xn` 的属性。

这个限制的意义是“能按从左到右的顺序计算”，因为预测分析和很多实际翻译过程都是从左到右处理输入的，右边的信息还没出现时就不能被依赖。

L-属性定义常见于需要上下文的任务：符号表与作用域传递、偏移分配、类型向下传播、参数个数与类型检查等。

L-属性定义如何与预测分析结合，为什么“最自然”?

- 在预测分析（递归下降）里，每个非终结符通常对应一个过程或函数。继承属性可以当作这个过程的参数传入，综合属性可以当作返回值传出。
- 因为 L-属性限制保证了“左到右就能算”，所以当你在过程里按产生式顺序调用子过程时，左边子过程已经算好的属性就能供右边子过程使用，这刚好匹配“先处理左边，再处理右边”的控制流。
- 例如声明列表里把 `inType` 传进去：处理第一个 `id` 后更新符号表，再处理后续 `, id`，所有 `id` 都能拿到同一个 `inType`。

L-属性定义如何与 LR 分析结合，为什么会更麻烦?

- LR 是自底向上，归约时右部已经在栈里，综合属性很好算；但继承属性需要“从上往下或从左到右传”，这和纯归约动作不完全一致。
- 常见解决方式是把语义动作嵌入到产生式中合适的位置，借助“标记非终结符（marker）”或“空产生式触发点”让 LR 在移进到某个位置时就执行一段动作，从而把需要的继承信息提前准备好。
- 结果上你仍然是在维护一个语义栈或若干辅助结构，只是语义动作不再只发生在归约点，还可能在“移进某些符号之后”发生，用来实现继承属性的传递与环境更新。

总结

- 它通常在考你是否理解“属性计算的时机”和“解析策略的控制流”必须匹配：
  - **LR（自底向上）最顺的是综合属性，因为归约就是合成父属性**
  - **预测分析（自顶向下、从左到右）最顺的是 L-属性**，因为继承属性能当参数往下传，左兄弟的信息在右兄弟之前就可得
- 如果题目让你选：**只用综合属性时，用 LR 实现往往最简单**；**需要继承属性且满足 L-属性限制时，用预测分析实现最直接**；需要在 LR 中做继承属性时，就要用嵌入语义动作/标记符号来安排执行时机。

PPT和试卷例子

综合属性计算的依赖关系是父节点依赖孩子节点，所以综合属性的计算_____。

A 容易与预测分析法相结合

B 容易与算符优先分析算法相结合

C 以上皆对

D 以上皆错

语法制导定义与翻译模式（翻译方案）在设计上有什么区别

- 语法制导定义（SDD）更像“规则集合”：告诉你每条产生式要计算哪些属性、属性之间怎么依赖，但不强制你把动作放在何时执行，属于“描述语义”。
- 翻译模式/翻译方案（常见叫语法制导翻译方案，SDT）更像“可执行流程”：把语义动作直接写在产生式右部的某些位置，明确规定在分析到这个位置时就执行动作，属于“安排执行时机”。
- 实际实现里，经常是先有 SDD（你知道要算什么），再把它改写成 SDT（你知道在预测分析或 LR 的哪个时刻执行）。
- 设计语法制导定义/翻译模式时最需要把握的要点
  - 先明确“翻译目标是什么”，例如只求值、要做类型检查、要生成三地址码、要构建抽象语法树、要维护符号表，这会决定你需要哪些属性（`val/type/place/code/env/offset` 等）。
  - 再确定“信息该往哪流”，如果只需要从子结构合成结果，尽量用综合属性；如果必须用上下文信息约束子结构，就引入继承属性，并尽量让它满足 L-属性的左到右可计算限制。
  - 最后把“规则”落到“执行时机”：
    - 走预测分析时，把继承属性当参数传递，把综合属性当返回值收集
    - 走 LR 时，把综合属性放到归约动作里算，把需要提前发生的继承动作用嵌入动作/标记点安排在移进到相应位置时执行

- 设计时要避免“循环依赖的属性”，也就是一个属性直接或间接依赖自己，导致无法确定计算顺序；考试里常用“依赖图是否有环”来判断属性定义是否可求值。