Syntax Directed Translation

基本思想：使用 CFG 来引导语言的翻译

语法制导定义

语法制导定义（Syntax Directed Definition, SDD）是 CFG 与**属性**和**规则**的结合，属性与文法符号结合，规则与产生式结合。语义规则用来描述如何计算属性的值

继承属性和综合属性

非终结符有两种属性

- 综合属性（synthesized attribute）：非终结符 A 的综合属性是由 A 的产生式关联的语义规则定义的，只能通过 A **本身和其子节点**的属性值定义
- 继承属性（inherited attribute）：非终结符 B 的继承属性是由其父节点的产生式关联的语义规则定义的，继承属性只能通过 B 的父节点，兄弟节点和 B 本身的属性值定义

终结符有综合属性而没有继承属性，终结符的属性是由词法分析器提供的

如果一个 SDD 求值的过程没有副作用，则称为属性文法（attribute grammar），其中的属性值定义仅依赖于其他属性和常量

如果一个 SDD 只含有综合属性，则称为 S 属性的 SDD，S 属性的 SDD 能和 LR 语法分析器一起实现

如果一个 SDD 的属性满足

- 是综合属性
- 是继承属性，且 $A \to X_1 X_2 \dots X_n$ 中计算 X_i 的属性的语义规则只使用
  - A 的继承属性
  - X_i 左边的文法符号的继承属性/综合属性
  - X_i 自身的继承属性/综合属性，且属性间的依赖不形成环

则称为 L 属性的 SDD

注释语法分析树（annotated parse tree）：显示了各个节点的属性的值的语法分析树

S 属性的 SDD 可以直接在语法分析树上自底向上求值

SDD 求值顺序

依赖图

在 SDD 求值中，求某个节点的属性，必须先求出其依赖的属性，这个依赖关系可以构成一个图结构：依赖图（dependency graph），如果依赖图中出现了环，则无法计算属性值

依赖图中从 a 到 b 的边表示计算 b 时需要 a 的值

一个 SDD 可行的求值顺序就是依赖图的拓扑排序，但是给定一个 SDD，很难判断是否存在一棵语法分析树，对应的依赖图中有环

上文中提到的两种 SDD（S 属性和 L 属性）一定不含有环，且有固定的排序模式，可以很好地跟已经研究过的语法分析结合起来实现

- S 属性的 SDD 中总是通过子节点的属性值计算父节点的属性值，可以在自顶向下和自底向上的过程中实现（分析树上后序遍历即可），若是在 LR 分析过程中计算，实际上不需要构造分析树，只需在归约的时候计算即可
- L 属性的 SDD 的依赖图中，继承属性边的方向总是从左到右，从上到下，而综合属性的边的方向总是从下到上，按照这个顺序扫描计算，即能保证在计算属性时，其依赖的属性都已计算完毕

具有受控副作用的语义规则

属性文法完全没有副作用，但是也增加了描述的复杂度

在 SDD 中某些副作用是可以被支持的

- 不会对属性求值产生约束的副作用，即按照任何依赖图中的拓扑序求得的属性值都能产生正确的翻译结果
- 或是约束求值顺序，使得任意允许的求值顺序都能得出正确的结果

SDD 的应用

构造 AST

抽象语法树（abstract syntax tree, AST）中每个节点代表一个语法结构，对应一个运算符，而其子节点代表运算分量，可以忽略掉标点之类的非本质的内容

AST 的叶节点仅存放词法值，而内部节点存放操作符和参数。不同的语法树可以对应同一颗 AST，这代表其语义相同，而在语法树结构与 AST 不同时，继承属性可以帮助求值

可以通过 SDD 为语法树构造 AST

类型结构

可以用 SDD 分析表达式的类型

如数组的基本类型和维数

语法制导的翻译方案

语法制导的翻译方案（Syntax Directed Translation Scheme, SDT）是在产生式体中嵌入程序片段的 CFG，这样的程序片段称为**语义动作**

SDT 的实现可以通过在语法分析树上进行从左到右的 DFS，将语义动作当作虚拟的节点，遍历到的时候执行。但通常 SDT 是在语法分析的过程中实现的，不用构造语法分析树。主要关注用 SDT 实现两类重要的 SDD

- 基本文法使用 LR 分析技术，且 SDD 为 S 属性
- 基本文法使用 LL 分析技术，且 SDD 为 L 属性

如何判定一个 SDT 能否在语法分析过程中实现：将每个内嵌的语义动作替换为一个独有的**标记非终结符**（marker nonterminal）M，其只有一个产生式 $M \to \epsilon$，如果这个文法能用某种方法进行语法分析，则这个 SDT 可以在该过程中实现

后缀翻译方案

后缀 SDT：所有语义动作都在产生式的最右端

如果一个文法可以自底向上分析，且其 SDD 是 S 属性的，则一定可以构造出一个后缀 SDT，只需将语义规则看作是赋值的语义动作，然后放在产生式最右端

后缀 SDT 可以在 LR 分析的过程中实现在归约时执行相应的语义动作即可，在归约时各个文法符号的属性值可以在对应的栈中找到

产生式内部带有语义动作的 SDT

考虑一个产生式 $B \to X\{a\}Y$

- 如果是自底向上分析，在 X 出现在栈顶时执行动作 a
- 如果是自顶向下分析，在输入中出现 Y 时执行 a

不是所有 SDT 都能在语法分析过程中完成的，对于这样的 SDT 只能按照建立语法分析树然后前序遍历的方式实现

SDT 中消除左递归

带有左递归的文法不能按照自顶向下的方法进行语法分析，故需要消除左递归，而当文法是 SDT 的一部分时，还需考虑到消除左递归时对语义动作的处理

当只需要关注语义动作执行的顺序时，如语义动作仅仅打印字符串，这时候将语义动作当作终结符处理即可，因为消除左递归的文法转换保证了生成的串中的终结符顺序不变

当一个语义动作是计算属性值时，如果 SDD 是 S 属性的，则总可以将计算属性的动作放在新产生式的适当位置得到一个 SDT，考虑

$$A \to A_1 Y \{A.a = g(A_1.a, Y.y)\}$$
$$A \to X \{A.a = f(X.x)\}$$

则可以构造新的文法

$$A \to XR$$
$$R \to YR \mid \epsilon$$

从而得到新的 SDT

$$A \to X \{R.i = f(X.x)\} R \{A.a = R.s\}$$
$$R \to Y \{R_1.i = g(R.i, Y.y)\} R_1 \{R.s = R_1.s\}$$
$$R \to \epsilon \{R.s = R.i\}$$

实现 L 属性的 SDD

L 属性的 SDT

将 L 属性的 SDD 转换为对应的 SDT 时，将每个语义规则看作是一个赋值语义动作

- 计算 A 的继承属性的语义动作插到产生式体中 A 的左边
- 计算产生式头的综合属性的语义动作在产生式最右边

在使用递归下降的语法分析器框架时，每个非终结符对应一个函数，参数为继承属性，而返回值为综合属性

边扫描边生成属性

在翻译工作时，属性体积很大会导致对属性的运算效率降低，此时可以采用边扫描边生成属性的方式进行运算

- 存在一个主属性为综合属性
- 各产生式中主属性是通过将产生式体中各个符号的主属性连接起来得到的，其中也可能包括非主属性的元素
- 各文法符号的主属性连接顺序和其在产生式中出现的顺序相同

这样可以增量式地修改主属性