自动生成前端编译程序的一种方法*

孙淑玲　郑启龙

（中国科学技术大学计算机系　合肥　230027）

摘要　本文构造的XYZ/NGAE系统是建立在属性文法基础上的前端编译程序自动生成系统. 本文在概述了属性文法及其描述语言之后，简要地介绍该系统的体结构以及为减少空间开销而采用的优化措施.

关键词　属性文法，前端编译程序，空间优化.

YACC是人们熟悉的编译程序生成器. 在YACC中语义子程序是用C语言写成的. 因此它的正确性难以验证，可靠性较差. Knuth在文献[1]中提出的属性文法为描述高级程序设计语言静态语义提供了形式化的框架. 属性文法的基本思想是对上下文无关文法进行定义扩充，使得每个文法符号具有一定的属性. 这些属性可通过一些计算规则在属性结构树上自上而下地继承传递，或者自下而上地综合传递，从而能方便的处理高级程序设计语言的上下文相关性.

XYZ/CCSS是中科院软件所唐稚松院士提出并研制的XYZ系统的子系统，它实现源源转换和编译的编译. 由该系统生成的编译程序明显地分成静态语义分析和动态语义分析2个阶段. 在静态语义分析阶段，利用属性文法描述高级语言静态语义. 在静态语义分析后将源程序转换为一种较标准的消除了上下文相关性的改写程序. 我们构造的XYZ/NGAE子系统是在原来蔡晓莉研制的XYZ/GPAE基础上经过剪裁、增补而形成的可实际运行的高级语言静态语义分析器的自动生成系统. 由XYZ/NGAE生成的系统称为前端编译程序，它完成对源程序的语法分析，建立带有属性的语法分析树（简称为属性结构树AST）；按照某种次序计算AST上的属性值实现静态语义分析.

1　属性文法及它的描述语言

属性文法AG以上下文无关文法为基础，扩充了能够反映具体程序设计语言中上下文相关性质的语义规则. 形式地，AG是一个五元组$AG=(G,A,VAL,SF,SC)$，其中G是上下文无关文法，A,VAL,SF,SC分别是属性集合，属性值集合，属性函数集合和属性条件集合. 用XYZ/DLAG语言写成的高级语言属性文法是由一些规则组成的. 每个规则包括一个

---

* 作者孙淑玲，女，1941年生，副教授，主要研究领域为形式语言与自动机，数据安全，智能计算机辅助教学. 郑启龙，1969年生，硕士，主要研究领域为计算机软件.

本文通讯联系人：孙淑玲，合肥 230027，中国科学技术大学计算机系

本文1995－04－18收到修改稿

上下文无关语法规则，一个或多个属性规则和上下文条件.

对于任何源程序所建立的属性结构树上，都应该能够计算出它们的属性值.这样就要求该语言的属性之间无循环依赖关系.这类 *AG* 称为良定义的属性文法 *WAG*.由于判定 *AG* 是否为 *WAG* 的算法复杂度是指数级的，故我们只考虑它的子类，即有序属性文法(简称为 *OAG*).对于 *OAG*，属性计算次序是在属性依赖关系的基础上自动构造出来的.对应每一条产生式 p，均有一个访问序列 VS_p.该序列是用产生式 p 进行派生而构造的某一属性结构子树上结点的局部遍历规则，它由计算、访问、判断和返回($VISIT_0$)4 种动作组成.[2,3]

描写 *AG* 的元语言是 XYZ/DLAG.由于属性刻画的是语言成分的静态性质，而且属性计算顺序是自动构造的.所以在 XYZ/DLAG 中不需要变量和描写控制流的语法成分.XYZ/DLAG 中可以从基本类型 boolean，integer，…等通过结构、联合等手段形成复杂类型，以适应对属性意义的抽象.我们在原来 XYZ/DLAG 中增加了远程属性访问 INCLUDING 和 CONSTITUENT(S) 2 种语义表达式.而在 XYZ/GPAE 中为传递子树某结点的属性值是通过为相关非终极符增添属性的方法解决的.有了远程属性访问就可以去掉这些冗余属性，使得 AG 简洁易读.XYZ/DLAG 语言编译程序是用 LEX，YACC 自动生成的.

远程属性访问的定义如下：

```
Remote_attribute_access::=
        'INCLUDING' '('attribute_name')'
      | 'INCLUDING' '('(attribute_name//,)')'
      |[symbol_name]'CONSTITUENTS' attribute_name
      | symbol_name 'CONSTITUENT' attribute_name
```

远程属性访问是在属性结构树上，显式地向上或向下远距离来传递相同属性值的一种缩写形式.相同属性指属性名相同、属性类型相同，但可为不同文法符号所拥有的属性.INCLUDING 的动作是沿结构树朝上追溯属性值，而 CONSTITUENT(S)相反.若 INCLUDING($Y_1.a_1,\cdots,Y_m.a_m$)，其中所有 $a_i,i\geqslant 1$均为同一类型 t，是产生式 r 中某条语义规则的一部分 $r:X_0::=X_1,X_2,\cdots,X_n$，那么该远程属性访问作用是沿着当前结构子树的根结点 X_0的父亲结点指针向上追溯.在此过程中，若某一祖先结点所对应的文法符号与 $Y_1\sim Y_m$ 之一相匹配，则返回该祖先实例结点的对应属性 $a_1\sim a_m$ 的值.它等同于调用函数 including(X_0.Parent).函数 including 返回的类型为 t，其实现框架如下(伪 C 语言)：

```
typedef struct xxNode {
    struct xxNode * Parent;
    struct xxNode * Child;
    struct xxNode * Brothers;
    ...
    /* 属性域 */
};
  t including(n)
    struct xxNode *n;
  {
    ...
    /* Symbol 是结点 n 所对应的文法符号 */
    return (!strcmp(Symbol,Y1) ? Y1.a1;
            ...
```

```
            !strcmp(Symbol,Ym) ? Ym.am:
            including(n->Parent) /* 继续向上追溯 */
          );
  }
```

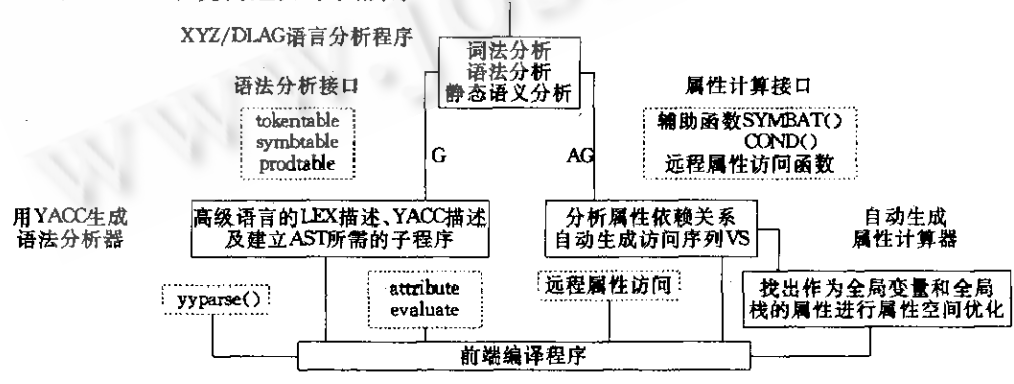
对于上述产生式 r,若存在远程属性访问 X_i CONSTITUENTS Z.a,则等同于调用函数 constituents(X_i).该函数类型为 t,其中 Z.a 的类型为 ta,类型 t 是以 ta 为基类型的表.其实现框架如下:

```
t constituents(n)
  struct xxNode *n;
{
  ...
  /* Symbol 是结点 n 所对应的文法符号 */
  return (!strcmp(Symbol,X0) ? t( ) :
          sub_constituents(n->Child)
          + /* 表的连接操作 */
          !strcmp(Symbol, Z) ? t(n.a) : t( )
         );
}
t sub_constituents(n1)
  struct xxNode *n1;
{t result;
 if (n1==NULL) return t( );
 else {
   result=constituents(n1)
          + /* 表的连接操作 */
          sub_constituents(n1->Brothers );
 }
 return result;
}
```

远程属性访问 CONSTITUENTS Z.a 等价于调用函数 sub_constituents(X_0.Child);而 Xi CONSTITUENT Z.a 则等价于函数调用 head(constituents(X_i)).

2　XYZ/NGAE 系统的实现

XYZ/NGAE 系统构造如下图所示.



XYZ/NGAE 系统接受由 XYZ/DLAG 语言写成的高级语言的属性文法.一方面对上下文无关文法进行分析,产生用于生成词法和语法分析器的 LEX 描述、YACC 描述以及各种数据表项.另一方面从属性文法分析属性间的相互依赖关系.对于每一产生式自动构造出

用于计算属性的属性访问序列.最后找出所有可以用全局变元或全局栈实现的属性,并把它们从属性结构树结点中抽出.从而大大减小属性结构树结点的规模,降低前端编译程序的空间开销.

3 属性空间优化

属性结构树作为前端编译程序的核心数据结构,它的存储空间占用量很大,并且随程序规模增大而迅速增长.因此,空间优化是使基于属性文法的前端编译程序实用化的重要举措.目前 XYZ/NGAE 系统中除一般性的优化措施(如尽量使用指针类型,避免复杂数据类型对象的拷贝)外,还把一些属性全局化.即将一些属性从结构树的结点中抽取出来,以全局变量或全局栈方式来实现,使结点变小,减少 AST 空间开销.

Saarinen 在文献[4]中首次提出,通过分析属性生命期把属性分成有效属性和临时属性.属性生命期是指,一个属性实例从它的第1次计算到它的最后1次引用之间的时间片段.粗略地说,临时属性仅在以该属性实例为根的子树中使用.故可能把它从结点中抽出作为全局量来实现.

XYZ/NGAE 系统分析每一个临时属性的生命期,以期确认该临时属性是否可以用全局变元或全局栈来实现.如果分析得知,在任何结构树上,属性 $X.a$ 的每一实例的生命期满足以下条件:

(1) 该实例生命期与其余 $X.a$ 的实例生命期不重叠,则将 $X.a$ 实现为全局变元.

(2) 若 $X.a$ 不能实现成全局变元,但进一步分析知在任何结构树上,$X.a$ 的任何2个实例的生命期不相交,或它们的生命期以适当形式相嵌套,则将 $X.a$ 实现为全局栈.设属性 X_1,X_2是一属性结构子树中非终极符 X 的2次出现.属性 $X_2.a$ 的生命期"适当"嵌套在属性 $X_1.a$ 的生命期中是指在属性计算时,如果属性 $X_2.a$ 的整个生命期存在于2个相继的属性 $X_1.a$ 的生命期之间.

这样我们就把确定一个临时属性 $X.a$ 是否可以用全局变元或全局栈来实现的问题转换成检查包含 $X.a$ 出现的每一个产生式是否满足某些条件.也就是说检查在这些产生式中 $X.a$ 是如何使用的.

下面是判定能否将临时属性实现为全局变元或全局栈的充分条件.[5]

(1) 作为全局变元实现的充分条件

若 $X.a$ 是临时继承属性,它在第 i 次访问 X 之前定义,并在此次访问之后不再被引用.假定产生式 P 以 X 为其左部,仅当下述条件之一满足:

(a) 产生式 P 的右部有 $X.a$ 的定义性出现;

(b) 访问产生式某个右部结点 Y,而且 Y 派生出 X 时 $X.a$ 不能用全局变元实现.

若 $X.a$ 为临时综合属性,它在对 X 的第 i 次访问中定义,且在执行 $VISIT_0$ 返回 X 的父结点后不再引用,那么,每一个右部有 X(如 X_j,$j>0$)出现的产生式 P,仅当在第 i 次对 X_j 结点访问后的第1个属性计算与最后1次引用 $X.a$ 之间以下条件之一满足:

(a) 产生式 P 左部有 $X.a$ 定义出现;

(b) 访问产生式 P 右部某结点 Y 且 $Y=X$ 或 Y 派生出 X,

则 $X.a$ 不能用全局变元实现.

(2)作为全局栈实现的充分条件

若 $X.a$ 是临时继承属性，在对 X 的第 i 次访问前定义且访问之后不再引用.同时我们假定产生式 P 左部为 X，P 的右部有 X 出现 $X_j(j>0)$且定义 $X_j.a$ 后在访问 X_j 之前 $X_0.a$ 又被访问，那么 $X.a$ 不能用全局栈实现.

若临时综合属性是 $X.s$ 的2个不同出现 $X_i.s$，$X_j.s$，它们同时满足

(a) $X_i.s$ 在 $X_j.s$ 之前定义；

(b) 对 $X_i.s$ 的最后1次引用在 $X_j.s$ 定义之后出现；

(c) $X_i.s$ 的最后引用在 $X_j.s$ 的最后引用前出现，

则 $X.s$ 不能用全局栈实现.

4 结束语

目前，我们已在SUN—4工作站上完成了XYZ/NGAE系统.整个系统源程序约10^4行.我们使用XYZ/NGAE系统生成了PASCAL语言的前端编译程序，它的可执行代码为500KB，比原XYZ/GPAE减少三分之一，并将它移植到微机上.由于实现了远程属性访问，PASCAL语言属性文法的规模得以减小.表1是在引入远程属性访问前后，对PASCAL语言属性文法的统计数据.

表1

	XYZ/GPAE系统	XYZ/NGAE系统
符号属性总数	353	187
属性赋值规则	235	224
属性传递	154	87
语义条件	68	67
远程访问	0	49
文法符号	75	74
符号平均属性数	4.7	2.5

表2

源文件行数	AST结点数	结点平均字节数		AST空间字节数	
		优化前	优化后	优化前	优化后
641	8451	37	34	317932	293146
974	13112	36	33	475872	437940
1458	19909	35	32	706156	649032
1949	26859	35	32	941380	864784
2465	33971	34	31	1182716	1085896
2932	40515	34	31	1404692	1289192
3436	47704	34	31	1648272	1512616
3936	54738	34	31	1886696	1731168
4436	61734	34	31	2124052	1948788
4860	67059	34	31	2590844	2376708
5872	81085	34	31	2779760	2549332

在引入“属性全局化”等空间优化措施后，我们又将近一半的属性从结构树结点中抽取出来. 以 PASCAL 的属性文法为例，仅有87个属性在结构树结点中分配空间，有14个属性实现为全局变元，87个属性实现为全局栈变元. 采用了优化措施以后，结点平均空间减少3个字节，PASCAL 的前端编译程序整体空间开销比优化前约降低10%，达到了基于属性文法的前端编译程序可实际运行的预期目的. 详细统计数据如表2所示.

参考文献

1 Knuth D E. Semantics of context—free language. Math. System Theory, 1968,2:127～145.

2 Kastens U. Ordered attributed grammars. Acta Information, 1980,13:229～256.

3 郑启龙. 基于属性文法的前端编译程序自动构造[硕士论文]. 中国科学技术大学,1994.

4 Saarinen M. On constructing efficient evaluators for attribute grammars. Automata, Languages, and Programming: 5th Colloquium, Berlin, Heidelberg, New York: Springer, 1978.

5 Farrow R, Yellin D. A comparison of storage optimizations in automatically—generated attribute evaluators. Acta Informatica, 1986,23:393～427.

A METHOD OF AUTOMATIC GENERATING FRONT—END COMPILER

Sun Shuling Zheng Qilong

(*Department of Computer Science University of Science and Technology of China Hefei* 230027)

Abstract This paper developed XYZ/NGAE system based on AG (attributed grammar). The system generates the FEC (front—end compiler) of a programming language automatically. In this paper, AG and the language for attribute definition are introduced at first, then the system structure of XYZ/NGAE and the attribute optimization technique are described.

Key words Attributed grammar, front—end compiler, attribute space optimization.