一个测试程序自动生成器的设计与实现

胡正国　朱志强

(西北工业大学计算机科学与工程系,西安 710069)

THE DESIGN AND IMPLEMENTATION OF AN AUTOMATIC GENERATOR FOR TESTING PROGRAM

Hu Zhengguo and Zhu Zhiqiang

(*Department of Computer Science and Engineering, Northwestern Polytechnical University, Xian* 710069)

Abstract This paper introduces an automatic program generator for testing compiler, which acceptes the context-free grammar of the compiled language as its input. According to the features of the language, we represent its basic elementes with some table-structures, and solve the context-sensitive problem of language effectively by operating on them. The generator can automatically produce not only the programmes that overlay all grammar conditiones, but also the programmes which have some special grammar conditions for users. The generator itself has some features in its generation-strategy, generation-algorithm and deravation—principle.

摘要 本文介绍了一个用于编译程序测试的程序自动生成器. 该生成器以编译程序对象语言的前后文无关文法作为输入,根据该语言的具体特点,对语言的数据对象采用了一种表结构的描述方式,并通过对表的操作,有效的解决了语言的前后文相关问题. 该生成器不但可按语言文法的描述自动生成覆盖该语言各种语法现象的程序,而且还可按用户要求生成具有某种语法现象的程序. 另外,该生成器在生成策略、生成算法及推导机制等方面也有自己显著的特色.

§0. 引　言

程序自动生成是对编译程序进行测试时生成测试用例的一种快速、有效的方法. 但是,构造一个生成器并不是一件容易的事. 其中,如何解决语言的前后文相关性是一个十分关键的问题. 近年来,人们在解决这一问题上已作了一些有益的工作. 例如,在文献[4]中 F. Bazzich 等构造了一前后文无关参数文法作为生成器的输入,这种文法在前后文无关文法中引入了一前后文相关参数作为沟通前后文信息的桥梁. 文献[5]中 A. Celentano 等采用了一

本文1991年2月25日收到,1991年6月3日定稿. 本课题受到国家自然科学基金项目6873039资助.作者**胡正国**,教授,主要研究领域为软件工程. **朱志强**,工程师,1990年硕士毕业于西北工业大学,主要研究领域为计算机应用.

种动作文法作为生成器的输入,对于自动生成的程序利用文法中的动作信息进行修正,从而达到前后文相关的目的.文献[6]中则把生成策略和解决相关性的动作信息全部列入产生式中,通过处理这些产生式来生成程序.显然,以上几种解决相关性的做法均是围绕作为生成器输入的语言的文法进行的.但是,构造一种动作文法或参数文法是一项很繁琐的工作,而且很难保证文法的正确性.因此,所生成程序的可靠性和有效性就受到很大限制.另外,对于不同的语言要构造不同的动作文法或参数文法,这就使所构造的生成器的通用性大大降低.

针对以上问题,我们提出了一种表结构方式,利用它对语言的各种数据对象及控制对象进行描述.在程序自动生成过程中,生成器以语言的前后文无关文法作为输入,以所建立的表结构为基础,一切操作均围绕"造表"与"查表"而进行.整个过程由一种有效的算法进行控制,且按照一定的生成策略自动进行.我们感到,这种表结构的方式简单、直观,而且稍加改造即可用于不同的语言,因而,其应用前景是相当广阔的.另外,由于自动生成的程序中使用的变量或常量是通过查表得到的,而表中的内容已全部经过定义或说明,这就很好的解决了语言的前后文相关问题.

基于以上的表结构思想,我们以 JOVIAL 语言为例构造了一个程序自动生成器.实践表明,我们提出的这一方法是完全可行和有效的.

§1. 表结构的设计

表结构的形式是我们自动生成程序,解决语言前后文相关性的基础.我们认为,表结构的构造可按以下方式进行.

(1) 分析语言的具体特点,确定出构成一个程序的基本数据对象.

(2) 对每一数据对象进行描述.这种描述包括数据对象的类型,属性标志,预置情况,与其它数据对象及程序结构的关系等.

(3) 根据语言的程序结构确定模块的表结构形式(如果有模块结构的话),以及程序的表结构形式.

概括地说,这些表结构都设计为链表的形式.例如,项(item)是 JOVIAL 语言的一个最基本的数据对象,对应于它的表结构是如图 1 所示的项表,它描述了一个模块中所用到的全部项,其中每一项的详细内容如图 2 所示.其中,域 CTV 取值为 c、t 或 v,分别表示该项为常数项、类型名或变量;分配属性指出该项是动态分配还是静态分配;预置情况指出在说明时是否预置过初值;SIZE 表示该项的长度;类型描述指出该项的类型;项类型名指出此项用另外一个项类型来定义,否则该域为空;通讯属性表示该项与其它模块的通讯关系;从属表名及从属块名表示该项从属于某一表或块.

图1 项表结构

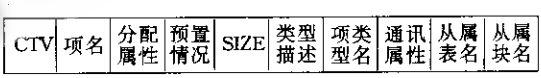

图2 项域内容

由于篇幅有限,模块表结构及程序表结构的形式不再举例说明了.需要指出的是,这些

表结构的每一个结点的域的设置是根据语言的特点确定的,但域的内容是在程序自动生成过程中产生的.即当生成一程序模块时,在生成算法的控制下首先进行造表,造表的顺序是:程序表→模块表→数据对象表,且在造表的同时生成程序说明部分的语句.造表结束后,在生成算法的控制下进行查表操作,生成程序的语句体部分.

§2. 生成策略和生成算法

2.1 生成策略

我们构造的程序自动生成器的输入为描述语言的前后文无关文法的产生式的集合,对于任意一个非终结符来说,可能存在着若干个推导该非终结符的产生式.因此,在推导过程中扫描到某一非终结符时需要从这些产生式中选择一个用于推导.为了使生成的程序能够覆盖语言的全部语法现象,而且满足交互式选择的要求,以及使推导尽量的短而精,我们制定了以下的生成策略(或称产生式选择策略).

(1) 选择交互式中用户指定的产生式.

(2) 选择递归次数未满足要求的产生式.

(3) 选择使用次数最少,且左端非终结符不在栈中的产生式.

(4) 选择按最短推导引入这样的非终结符的产生式:此非终结符不在栈中,且对于此非终结符不是重写它的所有产生式都使用了所要求的次数.

(5) 选择产生最短推导的产生式.

以上5条选择策略的优先级由上至下依次降低.而每次选择产生式时按优先顺序进行,且每次只能按其中的一条策略选择符合要求的产生式.其设计思想如下:

策略(1)是为系统的交互方式而设计的,其目的是为了首先使用户的要求得到满足,以生成具有某种特定语法现象的程序,若用户无要求,则按策略(2)继续考察;策略(2)是为解决产生式的递归推导而设计的,一个产生式递归多少次可决定产生某种语法现象的次数或程序的规模;策略(3)是为了使各产生式被均匀的使用,从而达到覆盖全部语法现象的目的;策略(4)、(5)的设计是为了在以上策略不满足的情况下,尽可能快地生成短而精的语句,这两条策略是Purdom算法的基本思想(详见文献[3]).

2.2 生成算法

我们构造的程序自动生成器以栈结构为基础,由生成算法进行控制,全部推导操作均在栈中进行.当栈为空时,表明数据对象处理完毕,即整个程序生成完毕.我们采用的生成算法可简单描述如下:

将要处理的文法的头符号放入一栈中,然后重复以下过程,直至栈空为止.

a. 若栈顶符号为终结符,则分别按以下方法处理:

a1. 若此终结符为语言的关键字、标识符或函数等,则从栈中弹出并输出到结果文件中.

a2. 若此终结符为说明部分的终结符,则进行造表、填表,并输出该终结符到结果文件中.

a3. 若此终结符为执行语句部分的终结符,则进行查表操作,找到表达该终结符的变量,并输出到结果文件中.

b. 否则(即栈顶为非终结符时),按照产生式选择策略,从重写栈顶符号的诸产生式中

选择满足要求的产生式,并使该产生式右端入栈.

§3. 其它问题

3.1 输入组织方式

程序自动生成器的输入为产生式的集合.这些产生式以文件的方式组织,文件的每一行只有一个产生式.其形式如下:

〈S〉=/a/〈A〉〈B〉/c/〈D〉#n1

〈A〉=/b/〈F〉……#n2

〈B〉=/Var/……#n3

……

其中,引号〈 〉中的符号为非终结符;斜杠/ /中的符号为终结符;#号为分隔符;ni 为第 i 个产生式的最大使用次数.以上形式的文件经预处理后,放入一产生式链表中供程序自动生成使用.

3.2 多栈推导方式

产生式的推导是在生成算法控制下,在栈中进行的.为了解决推导过程中过多的信息传送以及数据对象的识别,我们引入了多栈推导的概念.即在主程序中设立一主栈,每一独立的过程各自设立自己的分栈,组成一树结构的工作栈网.推导从主栈开始,然后依次在各分栈中进行.各栈工作过程均可表为:选择一产生式使其右端入栈,推导过程中每次从栈顶弹出一元素进行处理,直至栈空.

3.3 错误程序的生成及定位

为了对编译程序进行有效的测试,除了有语法上正确的程序,还应有带有各种语法错误的程序.我们采用构造错误产生式的方法来生成错误程序.另外,对于已生成程序中的错误必须进行适当标记以便于识别,这就是所谓的错误定位问题.我们采用的定位方法是在构造的错误产生式中加一些识别标志,并对识别标志以后的推导进行显式标记.例如,对于以下的产生式(其中 E 为错误标志)

〈assign_statement〉=/formula1/=/formula2/E〈statement〉#10

〈statement〉=FOR〈statement1〉#10

〈statement1〉=/varible1/=/valible2/#10

若从表结构中查得 formula1=A,formula2=200,varible1=B,varible2=C.通过推导可得以下错误程序(××为错误编号)

```
A=200
FOR B=C        "ERR ××"
```

3.4 错误耦合现象

对于两个或两个以上的错误产生式相互作用产生正确程序的耦合现象,我们采取以下预防措施:

(1) 生成若干个独立的程序,而每个程序的规模不要太大,只用来测试编译程序的某些语法现象.这样,在客观上就减少了耦合现象出现的机会.

(2) 错误程序中,错误出现的位置不要过分集中在一起,例如,可使每一语句不包含一

个以上的错误.其具体实现方法就是注意在推导时避免连续使用多个错误产生式.

3.5 失败的处理

在程序推导以及查表过程中,有可能发生诸如:在表中查不到所需的内容;或发现前面的推导工作与目前的推导相矛盾的现象.为此,我们采取以下一些方法来处理这些可能导致推导失败的问题.

(1) 回溯法:在推导过程中,当处理某一数据对象时,发现有上述问题,对推导栈进行搜索,找到并弹出表示该处理对象的所有符号.然后,按产生式选择策略选择其它满足要求的产生式,继续进行推导.

(2) 缓冲区输入法:设置一缓冲区,当处理到栈顶为语言的关键字时,把其记入该缓冲区中,随后对关键字后面的内容进行处理.若处理成功则记入缓冲区中,否则,缓冲区置空.当处理完该关键字的全部内容后,则输出该缓冲区内容至结果文件中.这样,若处理成功,输出的结果为被处理对象的全部内容,否则为一空行(在格式化时被删除).

(3) 标记输出法:在处理某一数据对象时,若数据对象的某项内容处理失败,则输出一标记,然后把未处理完的内容由回溯法弹出.在格式化时,若发现此类标记,则删除它以及它所指示的内容.

参考文献

1 W. Horner and R. Schovler, Independent Testing of Compiler Phases Using a Test Case Generator, Soft. Practice and Experience, Jan. 1989.

2 李友人,顾元祥,华庆一,编译程序的测试与实现,计算机技术,1985.6.

3 P. Purdom, A Sentence Generator for Testing Parsers, Bit 12, 1972.

4 F. Bazzich and I. Spadofora, An Automatic Generator for Compiler Testing, IEEE Trans. on Soft. Eng., July 1982.

5 A. Celentano, Compiler Testing Using a Sentence Generator, Soft. Practice and Experience, Nov. 1980.

6 P. A. Luker, Program Generator and Generator Software, The Computer Journal, 1988.4.