维普资讯 http://www.cqvip.com

软件开发 VB语言分析器 测试语义 设计

计算机科学 2000Vol.27№.11

基于测试语义的VB语言分析器的设计和实现*)

Design and Implementation of the Visual Basic Language Analyzer Based on Test Semantics

13-16,28 TP311.52

杨建军 陈卫东 叶澄清 潘云鹤

(浙江大学计算机科学系 杭州 310027)

Abstract Software testing, also known as an important part of Software Development Lifecycle, will continue to be a primary approach for ensuring correctness of software systems. It is necessary to develop a software testing tool to enhance testing efficiency and reduce testing cost. In this paper, on the basis of the software testing tool for the Visual Basic language we have developed, we firstly describe the implementation of the Engine—the kernal of this testing tool, then present the architecture of the VB language analyzer which is the front end of the Engine. Concerning the VB language analyzer, we stress on the design and implementation of lexical analyzer and parser that support the software testing. Finally, we show some results that proved the completeness of the testing semantics of our testing tool from a case study.

Keywords Software testing, Software testing tool, Structural testing, Lexical analyzer, Parser, Code instrumentation

1 引言

近几年来,随着信息技术的突飞猛进和计算理念的日益普及,软件开发作为一个产业得到了迅猛的发展,但在软件开发过程中,存在着诸多的问题。如:用户需求的频繁变更使得软件功能越来越复杂;软件功能越来越复杂又导致程序代码量增大,结构异常;而开发人员的频繁更替又使得程序维护甚是困难,等等,这些问题导致软件开发周期长,开发成本增加,产品稳定性差,质量低劣。为了能较好地解决上述问题,提高开发效率,保障软件的质量,我们开发了针对当前流行的面向对象的Visual Basic语言的软件测试工具。该工具以结构测试为主,服务的对象是了解软件内部编码逻辑的人员,或者说是软件的开发人员,能在单元测试和集成测试的级别上工作。我们的测试工具能实现程序结构图的自动生成、控制流图的自动生成和控制流分析自动化、覆盖分析自动化、质量度量自动化、动态跟踪自动化和测试执行自动化。

从逻辑上讲,我们开发的测试工具软件可以分为两大部分:引擎和自动化生成。其中引擎是整个系统的核心,是实现一系列自动化的基础,用以支撑整个软件测试工具。其工作流程如图1所示。

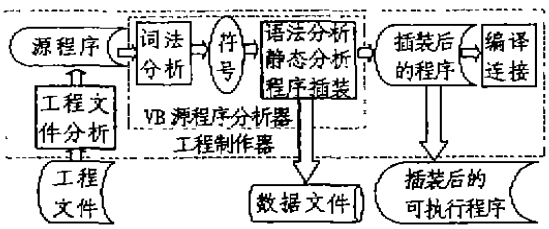


图1 引擎的工作流程

从上图可以看出,VB源程序分析器在引擎中起着举足轻重的作用。它主要对VB源程序进行词法、语法分析,相当于一个VB编译程序的前端。另外,在语法分析的同时对源程序作静态分析和程序插装,再把插装后的程序编译连接成可执行代码。静态分析是从源程序中提取必要的数据(比如:函数名、类名、行号等信息),同时对源程序的控制流进行分析,为源程序划分块,然后把得到的数据都保存到一定格式的专为测

*)本课题得到国家863高科技基金资助。杨建军 博士生,主要研究方向为软件测试、计算机网络、高性能及智能机系统。陈卫东 博士生,主要研究方向为计算机网络、智能CAD。叶澄清 教授,博士生导师,主要研究方向为软件测试、高性能及智能机系统、多机并行处理系统、多媒体计算机技术。潘云鹤 浙江大学校长,院士,博士生导师,主要研究方向为计算机图形学、人工智能、认知科学、CAD等。

维普资讯 http://www.cqvip.com

试设计的数据文件中，这些数据文件为实现图表自动生成和质量度量自动化作好了准备。程序插装就是要往源程序的特定位置中插入我们自己定义的代码，当插装后的可执行程序运行时就可以获取动态数据。插装后的可执行程序为实现覆盖分析自动化和动态跟踪自动化作好了准备。

我们架构的VB语言分析器已顺利通过许多VB源代码的测试，而且保证其测试语义的完好，即同时完成静态分析数据的提取和动态程序的插装。本文着重介绍分析器的设计和实现方法。

2 VB语言分析器的总体设计

2.1 分析器的逻辑结构

VB语言分析器与一般的编译程序有相似之处，同样需要完成语言的词法、语法分析工作，但在语义动作上、却有着本质的区别。VB语言分析器与一般的编译程序同样需要在词法分析的基础上，根据语言的文法规则，把单词符号串分解成各类语法单位，以确定整个输入串是否构成一个语法上正确的"程序"。但它的语义动作是为了实现测试的目的而执行的静态数据的获取和动态程序的插装，而不象一般高级语言编译程序的语义动作，是由计算机执行目标代码来完成算法所表述的功能。我们设计的分析器，其语义动作是服务于测试这个主题的。正因为如此，VB分析器在逻辑结构上有其自身的特点，如图2所示。

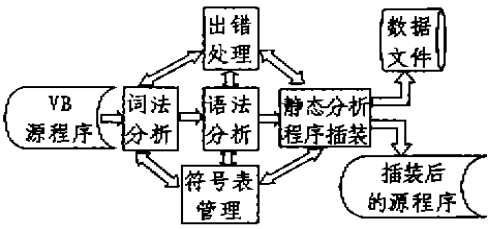


图2 分析器的逻辑结构

2.2 分析器的工作机制

分析器的词法分析程序本可以用Flex(类似于Unix下的Lex)产生，但考虑到用Flex产生的词法分析程序可读性较差且我们还要在词法分析程序中插入很多用于静态分析和程序插装的代码，另外还要让词法分析承担一部分语法分析的任务。所以，我们的词法分析程序是手工写的。

语法分析程序是我们用Bison根据源语言的语法产生式来生成的。Bison是当前比较流行的类似YACC的一个工具，要利用Bison分析一个语言，这个语言必须能用上下文无关文法描述。下面给出上下文无关文法的形式化定义：

一个上下文无关文法G是一个四元式(V_T，V_N，S，P)，其中：V_T是一个非空有限集，它的每个元素称为终结符；V_N是一个非空有限集，它的每个元素称为非终结符，$V_T \cap V_N = \varnothing$；S是一个非终结符，称为开始符号；P是一个产生式集合(有限)，每个产生式的形式如$P \rightarrow \alpha$，其中，$P \in V_N$，$\alpha \in (V_T \cup V_N)^*$。开始符号S至少必须在某个产生式的左部出现一次。

从以上定义可以看出，我们首先应根据测试任务的需求制定一个处理输入的说明。该说明的核心就是描述输入元素的一组规则即产生式。很多语言的产生式是公开的，但Visual Basic不是。而且Visual Basic严格讲是一种解释型语言，其语法不象纯编译语言那样规整，出现很多模棱两可的现象。所以，产生式的设计在本分析器的开发工作中占很大比重。

另一方面，利用Bison，只是把语法产生式转变为语法分析程序的框架，并不包含语义分析。用于静态分析和程序插装的代码就是镶嵌在这个框架中的。也就是说，测试的语义动作在框架中得以实现。

3 词法分析器的设计和实现

词法分析程序的输入是VB语言撰写的源程序，输出是终结符(Terminal Symbols，亦叫Token)所代表的数字代码。但是，仅有以上基本功能是不够的。为了正确分析VB代码和实现我们的测试语义，还必须加入许多控制和代码。在具体实现中，我们采用了以下技术：

1)*词法分析程序被一分为二* 对于象VB这样一种可视化的编程语言，系统自动在用户文件前端插入代码，如用以描述控件的属性的代码。在VB代码段，我们发现有些行为与用户代码段迥异。如变量可以以下划线起始，十六进制数据不需以"&H"开始等。另一方面，VB代码段的保留字表非常简单。为提高查找效率，同时为了处理VB代码段的特殊情况，我们将Lex分成了两部分：yylexInUserCode()和yylexInVBCode()。

2)*一遍扫描* 我们采取一遍扫描的策略，即对VB源程序从头到尾扫描一遍，并作有关的加工处理，直至生成最后结果。一遍扫描技术可以减少许多重复性的工作，大大提高工作效率。为了达到一遍扫描的目的，我们必须将VB的预编译指令也在词法分析程序中处理。分析预编译指令，关键是计算表达式的结果。为此，我们又构造了一个分析器。两个分析器在同一个系统中运作，会带来意想不到的后果。所以，我们必须

维普资讯 http://www.cqvip.com

修改 Bison 生成的语法分析文件。

3)保留字表的组织　保留字表的数据结构如下：

```
typedef struct
{
  char    acName[14];
  int     nLex Value;
  bool    bFlag;
}CReserveWord;
```

在分析 VB 代码时，我们发现某些字符串在特定的情况下是保留字，但在另外情况下却是标识符。如字符串 binary，在 Option 语句中它是保留字，除此之外，它却是标识符。为解决这一问题，我们在保留字表的每一项里设有一个标志位，为 TRUE 时表示该字符串是保留字；为 FALSE 时则表示是 IDENTIFIER。大多数保留字的该域初始为 TRUE，少部分为 FALSE。只是在需要时动态修改为 TRUE，之后仍旧置成 FALSE。

Hash 方法是一种在查表、填表两方面都能高速进行的技术，因此，我们采用 Hash 技术来组织保留字表。Hash 函数的构造采用直接定址法，做到一字一位。而且，由于直接定址所得地址集合和保留字集合的大小一致，所以，对于不同的保留字不会产生冲突。

4)超前搜索　象 VB 这样的语言，保留字不加特殊保护，一个字符串，既可以是保留字，也可以是标识符。虽然我们在保留字表中加了有关处理，但是如果在句首出现这种情况，保留字表里的标识位就无从翻转。为此，我们设置了一张句首表。如果句首是标识符，则检查句首表，若是句首表中的标识符之一，则要根据下一字符或下一个字符串来决定该标识符是否是保留字。

另外，词法分析器在从输入文件中识别出 Token 并传给语法分析器的同时，将有关字符串填入符号表。测试语义动作所需要的大部分信息，也是在词法分析程序中收集，如行号、Token 的偏移量、全注释行的统计和半注释行的统计等等。

4　语法分析器的设计和实现

Bison 提供了一个对计算机的输入进行结构化的通用工具，其实现必须依赖于为任务的需求而制定的处理输入的功能说明，该说明包括：描述输入元素的一组规则；识别出规则后调用的代码；和一个用来检测输入的底层扫描程序的定义和说明。

然后，由 Bison 将这样的说明转换成一个处理相应输入流的 C 语言函数（即语法分析函数），它通过调用用户提供的底层扫描程序（即词法分析程序），进而从输入流中找出基本项（即终结符）来。根据输入结构的规则（即产生式），对这些终结符加以组织，当某条规则被识别出来以后，就调用用户对应于这条规则所提供的代码（即动作）；动作具有回送值和利用别的动作的能力。

一个完整的 Bison 的规范说明文件的形式为：

```
%{
C declarations
%}
Bison declarations
%%
Grammar rules
%%
Additional C code
```

其中，C 声明部分包括在语义动作部分（action）要用到的类型和变量说明、C 预编译指令和文件包含等。Bison 声明部分则给出了终结符和非终结符的名字、操作符的优先级、语义值的数据类型等。语法规则部分是 Bison 程序的主体，其内容为文法的全部规则及与每一文法规则相关的语义动作（action）的描述。为测试 VB 代码而设计的语义动作正是在这儿实现的。附加部分也是由 C 语言程序构成的，可以在该段中放置任何你需要的代码，如在 action 中被调用的函数的函数体。当然，词法分析程序也可以放在这个部分，但为了结构清楚且易于维护，一般把词法分析程序放在一个单独的文件中。

从 VB 语言的具体特点出发，以 Bison 为整个词法分析器的基础，在具体实现中，我们采用了如下技术：

1)重新设置语义值　为了适应在处理 VB 源代码的过程中存取各种类型的符号，同时为了完成数据信息的静态统计和程序代码的动态插装，必须重新设计终结符的语义值。这首先可以在 Bison 的声明部分用 C 形式联合定义数值堆栈元素的数据类型，然后将相应的终结符定型。最后，将所有用符号 $ $ 和 $n 代表的非终结符定型。$ $ 代表每一个语法单位的语义值，而 $n 则代表产生式中每一个语法成分的语义值。

数值堆栈元素的数据类型如下：

```
%union{
  CCommon      Common;
  CIntType     IntType;
  CFloatType   FloatType;
  CStringType  StringType;
}
```

2)语法制导翻译　所谓语法制导翻译，是指在语法分析过程中，当一个产生式获得匹配（对于自上而下分析）或用于归约（对于自下而上分析）时，此产生式相应的语义子程序就进入工作，完成既定的翻译任务。语义子程序是给产生式赋予具体意义的手段。按语法制导翻译的方法来实现 VB 语言分析器，就是要根据 VB 的文法，分析各条产生式的语义，即静态数据的统计和

维普资讯 http://www.cqvip.com

动态程序的插装，分别写出完成这些操作的语义子程序，并把这些子程序插入到文法产生式右部合适的位置，从而形成翻译文法，即完成测试的语义。

3)Bison 的框架文件　Bison 所给出的 SIM 文件是语法分析程序的框架，它可以输出有关动作的冗长描述，包括读入了什么终结符，分析程序的动作(移进还是归约)是什么等，但这些信息是输出到标准输出的。为了提高分析程序的效率且符合 Windows 的编程规范，我们改写了 SIM 文件，使它的信息既可以输出到调试窗口，也可以输出到日志文件。

4)左递归　重复现象在 VB 语言中占有很大比例，在设计产生式的过程中，我们采用了大量的左递归方法来解决这个问题。如 VB 表达式的 BNF 形式的文法描述为：

```
expression_list::=
expression{,expression}
```

在 Bison 中利用左递归可以描述为：

```
expression_list:
  expression
| expression_list ','expression
;
```

5)BOGUS 的引入　我们设计了一个永远不会被词法分析程序返回的终结符 BOGUS。这个终结符最主要的用处是强制分析状态向我们定义的路径走，例如，以下两条产生式：

```
procedure:
    ...
| optional_procedure_specifier PROPERTY GET identifier
    ...
|...
;
optional_procedure_specifier:
    ...
|    switcher_keyword_property PUBLIC STATIC
    switcher_identifier_property
|switcher_keyword_property PUBLIC STATIC BOGUS
|...
;
```

其实，第二条产生式的第二个规则永远不会被分析到，因为 BOGUS 不是由词法分析程序产生的。当分析器读入 STATIC，强制其按第二条产生式的第一个规则分析下去。此时，下一个字符串是 property，而 switcher_keyword_property 已将 property 的标志位置为 TRUE，则词法分析程序将 property 作为保留字返回。随后，switcher_identifier_property 再将标志位置为 FALSE。这样，使得分析栈的状态与第一条产生式匹配。

5　实例分析

从第一版到现在，Visual Basic 经历了巨大的演变，它已成为一个重要的开发环境，包括编程的各个方面，从教育应用程序到数据库，从财务应用程序到开发 Internet 构件，我们依据 Visual Basic 6.0语法特征设计的产生式，经 Bison 所生成的语言分析器的转换状态共有将近1200个。在完成系统的设计并予以实现以后，我们对 VB 语言测试工具进行了实例测试。测试项目包括程序结构图及控制流图的自动生成、控制流分析自动化、覆盖分析自动化、质量度量自动化、动态跟踪自动化和测试执行自动化。本文给出与语言分析器相关的部分代码的测试结果。

5.1　实例源程序

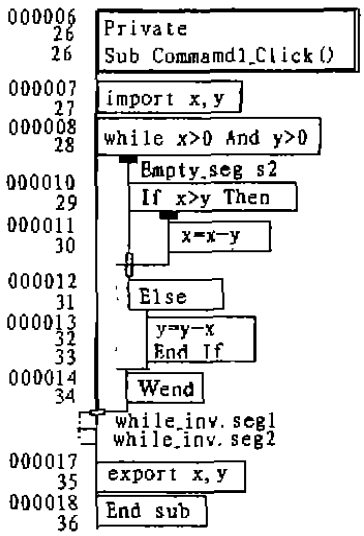
我们架构的 VB 语言分析器已通过众多的实例验证，以下是一个用欧几里德方法计算最大公约数的实例程序。

```
Private Sub Commandl_Click()
import x,y
While x>0 And y>0
    If x>y Then
      x=x-y
    Else
      y=y-x
    End If
Wend
export x,y
End Sub
```

将这段源程序作为 VB 语言分析器的输入，本文接着给出其分析结果，包括两个方面：一是根据数据文件生成的基于块(block)的划分图；二是将源程序插装后形成的文件。

5.2　基于块的划分图

我们把程序理解成块的序列，经静态分析之后，我们得到将程序划分成块的数据文件。下图是我们根据数据文件生成的程序的块的划分图。



5.3　插装后的源程序

动态分析的基础是代码插装。由于插装后的源代码很长，我们从中节选部分与实例代码对应的代码见 P28。

(下转28页)

维普资讯 http://www.cqvip.com

表2 算法求解成功率随网络节点变化的情况($\alpha/N=20\%$)

网络总节点数	10	20	30	40	50	60	70	80	90	100
成功率	100%	100%	100%	100%	100%	100%	100%	100%	99%	93.3%

结论 基于延迟敏感的最小费用问题是典型的优化问题。由于此类问题属于NP完全问题，目前多数采用非确定性的启发式方法来求解。其中最具代表性的是Parsa提出的延时限制的最小组播路由树启发式算法(BSMA)，它可计算出近似最佳的延时限制组播路由树，但它的运算时间开销巨大且未考虑延时抖动的因素。本文在CMST算法的基础上，分析了在高速网络中具有端-端延时和延时抖动的组播路由算法问题，提出了一种适合延迟与延迟抖动要求的组播路由遗传算法模型，并进行了实验分析，结果表明，本文提出的方法能求解适合延迟与延迟抖动要求的最小组播路由树，而且在相对大型的网络中，本文提出的MCMST算法与BSMA相比，具有搜索速度快、效率高，计算性能较稳定的特点，避免了采用启发式搜索技术对问题知识的要求，具有较强的通用性和鲁棒性，能够满足多媒体应用对时延和时延抖动的要求并可适用于大规模的网络中。

参考文献

1 Widyono R. The Design and Evaluation of Routing Algorithms for Real-Time Channels [Tech Rep. ICSI TR-94-024]. University of California at Berkeley, International Computer Science Institute, June 1994

2 Kompella V, et al. Multicast Routing for Multimedia Communication. IEEE/ACM Transactions on Networking, 1993, 1(3): 286～292

3 Kompella V, et al. Multicasting for Multimedia Applications. In: Proceedings of IEEE INFOCOM'92. 1992. 2078～2085

4 Parsa M, et al. An Iterative Algorithm for Delay-Constrained Minimum-Cost Multicasting. IEEE/ACM Transactions on Networking, 1998, 6(4): 461～474

5 Rouskas G N, Baldine I. Multicast routing with end-to-ent delay and delay variation constraints. IEEE Journal on Selected Areas in Comm, 1997, 15(3): 346～356

6 Garey M R, Johnson D S. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York: W H Freeman and CO., 1979

7 Zhu Q, et al. Garcia-Luna-Aceves. A source-based algorithm for near-optimum delay-constrained multicasting. In: Proceedings of IEEE Infocom'95. March 1995

8 Lawler E. Combinatorial Optimization: Networks and Matroids. Holt, Rinehart and Winston, 1976

9 陈国良，王熙法，庄镇泉，等. 遗传算法及其应用. 人民邮电出版社，1996

10 Waxman B M. Routing of Multipoint Connections. IEEE Journal of Selected Areas in Communications, 1988, 6(9): 1617～1622

(上接第16页)

```
...
Dim ISA_WHILE1 As Integer
    ISA_WHILE1=1
    Call CountRP(ISA_local,2)
While Cond(x>0 And y>0,-1)
    ISA_WHILE1=0
    Call CountRP(ISA_local,3)
Dim ISA_IF2 As Integer
    ISA_IF2=1
    Call CountRP(ISA_local,4)
If Cond(x<>y,0) Then
    ISA_IF2=0
    Call CountRP(ISA_local,5)
        x=x-y
    Call CountRP(ISA_local,7)
        y=y-x
    End If
    Call CountRP(ISA_local,8)
Wend
...
```

'Count WHILE_NODE

'Has been in WHILE_BODY
'Count WHILE_BODY

'Count IF_NODE

'Count IF_BODY

'Count ELSE_BODY

'Count IF_CONJ

结束语 产生式的设计与源语言是密切相关的，但是，完成测试语义动作的子程序与源语言并没有多大的联系。也就是说，只要修改产生式，使其面向另外的编程语言，我们可以方便地实现我们的测试语义，使我们开发的软件测试工具平滑地向其他编程语言过渡，如Delphi。

参考文献

1 陈火旺，钱家骅，孙永强. 程序设计语言编译原理. 北京：国防工业出版社，1984

2 郑人杰. 计算机软件测试技术. 北京：清华大学出版社，1992

3 孙霆. 软件测试工具的研究和实现.[硕士学位论文] 杭州：浙江大学，1999

4 Donnelly C, Stallman R. Bison Information. Cambridge, Free Software Foundation, 1992

5 Betzer B. Software Testing Techniques. New York: Van Nostrand Reinhold Company, 1990