软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(6):1155−1168 [doi: 10.3724/SP.J.1001.2011.04021]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

Object-Z 规格说明测试用例的自动生成器*

许庆国[1,2+], 缪淮扣[1], 曹晓夏[3], 胡晓波[1]

[1](上海大学 计算机工程与科学学院,上海 200072)

[2](武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

[3](上海第二工业大学 计算机与信息学院,上海 201209)

Automatic Test Case Generator for Object-Z Specification

XU Qing-Guo[1,2+], MIAO Huai-Kou[1], CAO Xiao-Xia[3], HU Xiao-Bo[1]

[1](School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

[2](State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)

[3](School of Computer and Information, Shanghai Second Polytechnic University, Shanghai 201209, China)

+ Corresponding author: E-mail: qgxu@shu.edu.cn

Xu QG, Miao HK, Cao XX, Hu XB. Automatic test case generator for Object-Z specification. *Journal of Software*, 2011,22(6):1155−1168. http://www.jos.org.cn/1000-9825/4021.htm

Abstract: Most research on test case generation from Object-Z specification focuses on theory. There is almost no tool to support generating test cases automatically. The Object-Z is a mathematics and logic based formal specification language. It uses schema composition and abbreviation format, which brings difficulty for extracting semantics and then generating test cases from specification automatically. This paper provides a solution in extracting semantics and generating test cases from Object-Z specification by unfolding the schema definition and improving its syntax in Object-Z. The process has three steps including parsing Object-Z language, extracting semantics, and generating test cases automatically.

Key words: specification-based testing; Object-Z; semantics extraction; test case generator

摘 要: 对Object-Z形式规格说明构造测试用例的研究,目前主要集中在理论研究阶段,测试用例的自动生成几乎没有相应的工具支持.Object-Z 是基于数学和逻辑的语言,并大量使用了模式复合和简写形式,这给计算机提取完整语义用以自动产生测试用例造成了困难.通过展开 Object-Z 规格说明中的模式定义,改进 Object-Z 的文法结构,给出了提取 Object-Z 规格说明语义的方法,研究了从 Object-Z 规格说明产生测试用例的自动化过程.这一过程主要包含 3 个阶段:Object-Z 语言的自动解析、语义自动抽取和测试用例自动产生.通过介绍的工具原型,可以很容易得到规格说明中的各种语义;基于某些测试准则,能够方便自动产生可视化的抽象测试用例.

关键词: 基于规格说明的测试;Object-Z;语义提取;测试用例生成器

中图法分类号: TP311 **文献标识码**: A

* 基金项目: 国家自然科学基金(60970007, 61073050); 国家重点基础研究发展计划(973)(2007CB310800); 上海市自然科学基金(09ZR1412100); 上海市科学技术委员会项目(10510704900); 上海市重点学科建设项目(J50103); 武汉大学软件工程国家重点实验室开放基金(SKLSE2010-08-26)

收稿时间: 2010-07-10; 定稿时间: 2011-03-29

由于形式规格说明语言提供了一种精确、一致的、易于被机器处理的符号来描述软件需求规格说明,所以基于规格说明的测试主要以形式规格说明作为产生测试用例的基础.根据所采用的形式规格说明语言的不同,基于规格说明的软件测试又可分为基于模型规格说明的测试、基于代数规格说明的测试和基于有限状态机的测试.基于模型的形式规格说明语言,如 VDM 和 Z 等,是通过一组逻辑谓词及其相关的状态变量来描述软件需求规格说明[1].本文研究的即是针对模型规格说明的测试.

Dick 和 Faivre 给出了一种根据 VDM 规格说明产生测试用例的方法[2].这种方法从规格说明中抽取出每个操作的前、后置条件,随后将这些条件谓词转化成析取范式,再对析取范式中的每个文字求解,所得到的结果就是最后的测试用例.Offutt 和 Liu[3]给出了从 SOFL 形式规格说明产生测试用例的详细步骤,并通过具体的实例详细说明测试用例的产生过程.Offutt 还给出了一系列基于状态规格说明的测试准则,用来指导基于状态规格说明的系统测试.

Z 语言是基于一阶谓词逻辑和集合论的形式规格说明语言,称 Z 语言为形式语言是由于它采用了严格的数学理论,这样可以产生简明、精确、无歧义且可证明的规格说明[1].Z 语言具有一种特殊的图形方式,称为模式(schema),模式主要由两部分组成:一是声明,负责定义各种变量的类型;另一部分则为谓词表达式.后者对于规格说明的功能描述是较重要的部分,通过这些谓词表达式可以推理出所描述软件的所有状态行为.基于 Z 规格说明语言的测试用例生成的研究工作主要有:

(1) Stocks[4]给出了一种从 Z 规格说明产生测试用例的框架,该框架主要用来形式测试规格说明和单元测试过程,整合了测试用例的推导过程和测试用例的形式描述. Paul Ammann 等人[5]给出了从 Z 产生测试用例的方法和测试准则;

(2) Hierons[6]给出一个从 Z 规格说明产生有限状态机的方法,得到的有限状态机可以用来控制测试过程;

(3) Burton[7]给出了一个基于规格说明的测试框架.该框架用 Z 描述测试用例,以定理的形式描述测试策略和测试充分性,并将测试用例的产生过程统一在原有的 CADiZ 开发环境中,充分体现了测试自动化的思想.

以上研究主要是针对一般规格说明语言的测试技术,而 Object-Z 语言在 Z 语言的基础上增加了面向对象的封装和继承的描述能力[8],因此需要针对 Object-Z 语言的特点提出测试准则,从而生成测试用例.

本文第 1 节简介 Object-Z 测试方法.第 2 节对 Object-Z 规格说明自动解析.第 3 节研究测试用例的自动生成技术.第 4 节介绍相应的测试用例生成器的工具实现.第 5 节给出用实现的工具考察的实例.最后是对本文的总结及将来工作的展望.

1 相关研究

1.1 Object-Z规格说明测试用例生成技术

类是面向对象中最重要的概念,也是面向对象测试中的单元测试对象.类是数据和对数据操作方法的集合.由于类的封装性使得一个对象的数据和状态信息对外部对象不可见,外部对象只能通过该对象提供的方法来访问该对象的内部状态.因此,要保证一个类正确性就是要保证类提供的方法的正确性.如果方法的实现没有正确地修改对象的内部状态,那么该类就可能存在问题.所以,封装性要求测试人员能够通过有效的测试用例来找出对对象内部状态进行了不正确地存取的方法.

Object-Z 是为描述面向对象规格说明而生的,因此也需要针对 Object-Z 语言的特点提出测试准则.Object-Z 语言是在 Z 语言的基础上发展而来,因此针对 Z 语言给出的测试方法和测试准则也可以运用到测试类内操作模式上来.

Carrington[9]介绍了从 Object-Z 规格说明的每一个操作模式产生测试模板,这些测试模板用来产生有限状态机,该有限状态机用来描述一个类的行为,并且把 ClassBench 测试执行框架作为执行测试用例的基础.

McDonald 等人给出了用手工把规格说明转换成测试场景的方法,其中包括程序框架、测试场景和执行驱动的方法,但是不包含测试用例自动生成[10].

此前,我们针对基于形式规格说明的单元测试和类测试方法进行了研究[11−13],针对目前基于形式规格说明在面向对象测试方面所面临的问题,研究了基于 Object-Z 产生类内测试用例以及类之间的测试用例的方法,给出一系列基于 Object-Z 的单元测试和集成测试的准则.通过这些准则产生的测试用例,探讨在测试由于类的封装、继承、多态、组合和使用等带来的问题上的有效性.这组准则利用方法内测试用例来构造类内部测试用例,避免了以往的研究中采用基于状态图的测试准则构造测试用例时需要的抽取类的状态图的工作,使得机械化地构造类测试用例成为可能.在类间测试层次,给出了一系列用于测试类间的多态关系的测试准则,改变了以往对多态关系的测试主要集中于基于程序代码的方法.应用这些用于测试类间多态关系的测试准则,测试人员可以从形式规格说明中推导出用于测试多态对象的测试用例,这些测试用例能够最终检测出程序代码中的多态关系的实现与规格说明中的定义是否一致.

1.2 Object-Z类内部测试准则

类的测试主要包括方法测试、方法间测试和类内部测试 3 个层次.关于前面两个层次的测试,已有不少研究工作报道,本课题组也有研究成果[11,12].本文主要研究 Object-Z 的类内部测试.Object-Z 语言从 Z 语言扩展而来,因此针对 Z 语言给出的测试方法和测试准则也可以运用到测试类内操作模式上来.组合谓词覆盖准则(combinational coverage)[12]是比较常用的方法内测试准则,也可用于 Object-Z 的类内部测试.下面给出其定义.

定义 1(组合覆盖准则(combinational coverage)). 一组文字的真值组合构成的集合 t 满足组合覆盖准则,当且仅当对每个 $p\in P$(其中,p 为文字,P 为谓词中包含的所有文字集合),t 包含所有可能的文字真值的组合.

例如,对于谓词 $A\vee B$,测试集{A=true∧B=true,A=true∧B=false,A=false∧B=true,A=false∧B=false}满足组合覆盖准则.

仅借用原有的测试准则是不够的.这主要是因为 Object-Z 语言为加强类继承和封装的描述能力,增加了新的表示方式[8],如:

OperationName≙OperationExpression

OperationExpression::=∧Declaration[|Predicate]•OperationExpression…

OperationExpression1::=[DeltaList[Declaration][|Predicate]]…

这些操作模式中使用了其他类的可视操作模式,相应地需要一些方法能够测试实现代码中的函数是否正确地调用了其他类的函数.下面给出产生这类操作模式测试用例的逻辑准则.

首先,给出调用模式(promotion schema)的测试覆盖准则,调用模式是指不含逻辑联结词的形如 Op≙instance.OperationName 的操作模式.

定义 2(调用模式覆盖准则). 测试用例集符合该测试准则,当且仅当调用模式包含至少一次取 true,一次取 false.

例如有操作模式 $Op1\triangleq a.Op$,其中,a 是类模式 A 的一个实例.$Op\triangleq\Delta(count)$,$count\leqslant 0\wedge count'=count+1$,其中,$count$ 为类模式 A 的状态变量,为整型类型.根据调用覆盖准则,可以得到测试用例 $a.count\leqslant 0$ 和 $a.count>0$,分别使得 Op 的前置条件的真值为 true 和 false.

其次,给出简单调用模式的测试覆盖准则,简单调用模式是指其定义部分包含多个调用模式的逻辑与(∧)联结.

定义 3(简单调用模式简单覆盖准则). 测试用例集符合该测试准则,当且仅当测试用例集使每个调用模式都符合定义 2 中的调用模式覆盖测试准则.

例如有简单调用模式 $Op2\triangleq a.Op\wedge a.Po$,其中,$Op$ 和 Po 均为类模式 A 的实例变量 a 的可视操作模式名.{$a.Op$=true∧$a.Po$=true,$a.Op$=false∧$a.Po$=false}即为符合该测试准则的测试用例集之一.

定义 4(简单调用模式全覆盖准则). 测试用例集包含两种测试用例:一种为所有调用模式取真值;另一种为 $\forall i=1\ldots n|instance_i.OperationName_i$ 取真值,$instance_j.OperationName_j$ 取假值,$j\neq i$.

{$a.Op$=true∧$a.Po$=true,$a.Op$=true∧$a.Po$=false,$a.Op$=false∧$a.Po$=true}为 $Op2$ 的符合简单调用模式全覆盖测的测试用例集.

若操作模式的定义部分包含多个由其他联结词(如∧,∨,□,⨟ ,•,||,|||等),则称为复杂调用模式,可以根据其语义将其转换为两个以上的简单调用模式[8],从而应用上面的覆盖准则来产生测试用例.

除上述调用模式、简单调用模式和复杂调用模式之外的其他操作模式称为组合调用模式,可用定义 5 中的覆盖准则来产生测试用例.

定义 5(组合调用模式覆盖准则).

- **分准则 1**:v 是任意一个组合调用模式中的变量,该变量为类联合类型或多态类型,则符合该测试准则的测试用例集合,当且仅当 v 覆盖所有有效的类类型至少一次.这里的"有效"是指当 v 为某一类类型时,谓词是一个合法的表达式,即谓词中包含的属性均存在于该类的可视列表中;
- **分准则 2**:v 是任意一个组合调用模式中的变量,该变量为集合类型中的一个元素或者子集,且该集合的元素为类类型,则符合该测试准则的测试用例集合,当且仅当 v 覆盖所有的元素.

分准则 1 和分准则 2 组成为组合调用模式覆盖准则.

例如有 $Op4 \triangleq a.count<-10 \wedge a.Op$,其中,$a$ 是类模式 A 的一个实例,类模式 B,C 从类模式 A 继承,A,B,C 均包含了属性 $count$.根据定义 5,$Op4$ 的测试用例集合将对类 A,B,C 均的实例分别产生测试用例.

显然,上述给出的测试准则针对的对象是不同的.针对组合调用模式中非复杂调用操作模式部分,使用组合调用模式覆盖准则可以产生测试用例;对于复杂调用部分,可以经过语义转换成若干简单调用模式,对每一个简单调用运用简单调用简单覆盖准则或简单调用全覆盖准则可以产生测试用例.从这两个准则的描述容易得出,这两个测试准则之间没有包含关系.对于调用模式,可以使用调用模式覆盖.显然,简单调用简单覆盖准则和简单调用全覆盖准则包含调用模式覆盖.

本文将在开发的工具(参见第 4 节)中应用其中的一些测试准则以自动产生相应模式的测试用例集合.

1.3 Object-Z相关工具

关于 Object-Z 的工具,目前文献报告并不多见.CZT(community Z tools)[14]项目是一个关于 Z 形式规格说明语言的集成化框架,它支持 Z 和 Z 的扩展(当然包括 Object-Z)[15],但其对 Object-Z 规格说明的检验,是通过将转换为其他语言(如 B machine)表示的状态转换系统再考虑验证或测试来实现的,并未提供对 Object-Z 规格说明本身的测试功能的支持.

2 Object-Z 规格说明的自动解析

2.1 Object-Z语法的文本化

Object-Z 是对基于模型的形式规格说明语言 Z 的扩展,使其包含了面向对象的特征.在 Object-Z 中,类由状态模式、用于定义类的属性和状态变量、与该状态模式相关的操作构成.其语法如图 1 所示.图 1 中 *FormalParameters* 表示类中变量的类型,类似于 C++中模板类的类型声明;*VisibilityList* 定义了类的接口,类似于 C++中的 public 定义;*InheritedClasses* 说明了被继承的类;*LocalDefinitions* 说明了在类中使用的类型和常量;*StateSchemas* 为状态模式,它定义了类的状态变量,该模式定义了类的可能状态空间;*InitialStateSchemas* 为初始状态模式,它定义了类的初始状态;*OperationSchemas* 为操作模式,定义了可以作用在该类的对象上的操作.

对于这里的每一语法元素,Object-Z 均有相应的文法定义.为使计算机系统方便地识别处理 Object-Z 语言,需要将图 1 所示的 Box 型图转换为文本形式,LaTeX 格式刚好满足这样的要求.根据 King 对 LaTeX 进行的扩展[16],使得 LaTeX 完全支持 Obejct-Z 规格说明的描述和显示,Object-Z 规格说明的完全文本化成为可能.下面是基本的转换约定:

- 整个类模式的定义包含在\begin{class},\end{class}对中;
- \visibility 表示下面一行为可视列表;
- 局部定义包含在\locpar,\endlocpar 对中;
- 状态模式定义包含在\begin{state},\end{state}对中;

- 初始状态模式定义包含在\begin{init},\end{init}对中;
- 操作模式定义包含在\begin{op},\end{op}对中,为对各个模式进行区分,在\begin{op}后面跟上模式名.

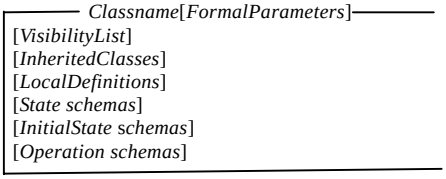


Fig.1 Box schema graph of class in Object-Z specification

图 1 Object-Z 中类语法的 Box 型式图

2.2 Object-Z规格说明文法的改进及自动解析

得到 Object-Z 规格说明 LaTeX 格式的文件后,可以对文件进行语法分析,生成整个语法树.我们采用 YACC 对文法进行分析,发现 Object-Z 文法存在 400 多处的冲突.把 Object-Z 转化成 LaTeX 格式后,可以消除大量的文法冲突,但还是不能完全消除所有的文法冲突.下面的片段是文法冲突的一个例子,Object-Z 的 BNF[8]:

Predicate1::=Expression Relation…Relation Expression

|SchemaReference…

Expression::=Expression1|…

Expression1::=Expression2|…

Expression2::=Expression3|…

Expression3::=Expression4|…

Expression4::=SchemaReference|…

可以看出,对于一个 Predicate1 产生 SchemaReference 的形式,有产生式冲突,如图 2 的语法树所示.这时,解析器无法判断下面的表达式应该使用第 1 个产生式还是使用第 2 个产生式.为了解决这类冲突,可以采用“向后看”的方法,如果发现后面的语法树结点是 Relation,则采用第 1 个产生式 Predicate1::=Expression Relation…Relation Expression;否则,采用 Predicate1::= SchemaReference 产生式.

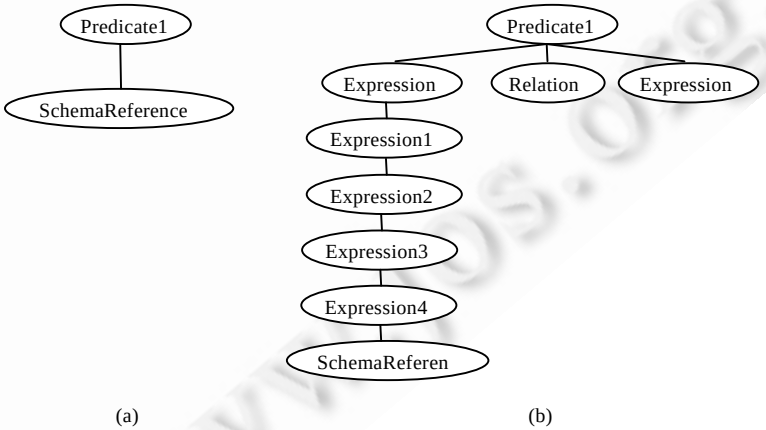


Fig.2 Syntax tree for Predicate1

图 2 Predicate1 的语法树

为了自动解析 Object-Z 转换过来的 LaTeX 文本,需要设计一种文法用于解析规格说明,而且能够解决上述

文法冲突问题.由于文法如果存在左递归,不是 LL(1)文法,需要将 Object-Z 文法进行改进,转换为相应的 LL(1)文法,再构造相应的 *FIRST*,*FOLLOW*,*LAST* 集合,就可以构造出 Object-Z 对应 LaTeX 文件的文法解析器.但考虑到 Object-Z 文法本身已经相当复杂了,把它转化成 LaTeX 格式后文法将更加复杂,人工计算构造 *FIRST*, *FOLLOW*,*LAST* 表将是一个非常容易出错的过程,所以本文不采取这种方法对 LaTeX 格式的 Object-Z 文法进行解析.

本文采用了类似自底向上的方式进行解析,转换为类 LR 文法再进行解析.转换时的主要规则为:

规则 1. 普通文本可以直接转换.

规则 2. 有可选项的文法要增加产生式.

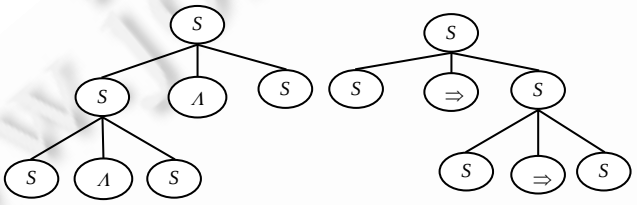
规则 3. 词法分析时要消除简单归约冲突.

规则 4. 解析包含递归产生式时,最后处理递归的表达式.

规则 5. 使用递归定义处理包含无限个元素的解析.

规则 6. 无法消除的归约冲突在实现时分别处理.

在 Object-Z 的表达式中,存在着多个函数运算符,而这些函数运算符的优先级是不同的.所以,在解析一个表达式的时候需要考虑不同运算符的结合顺序问题:(1) 在解析右结合联结词时放到语法树的更深一层,否则放到语法树的浅一层.如表达式 $S\wedge S\wedge S$,$S\Rightarrow S\Rightarrow S$,联结词$\wedge$是一个左结合联结词,而$\Rightarrow$是一个右结合联结词,那么这两个表达式按照结合规格应该为$(S\wedge S)\wedge S$,$S\Rightarrow(S\Rightarrow S)$,则形成的语法树分别如图 3(a)、图 3 (b)所示;(2) 函数关系符的优先级通过增加产生式使低优先级的函数关系符处于语法树较高层,高优先级的函数关系符处于语法树的较低层.



(a) $S\wedge S\wedge S$ (b) $S\Rightarrow S\Rightarrow S$

Fig.3 Syntax trees of $S\wedge S\wedge S$ and $S\Rightarrow S\Rightarrow S$

图 3 $S\wedge S\wedge S$ and $S\Rightarrow S\Rightarrow S$ 的语法树

如有 Object-Z 的关于中缀函数关系符文法:

InfixFunctionSymbol::=*div* 4 | ⊕ 5.

由于 *div* 中缀函数关系符的优先级 4 比⊕的优先级 5 低,表达式 *E div E* ⊕ *E* 产生的语法树如图 4 所示.

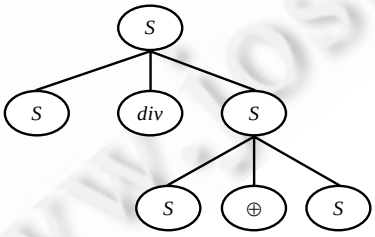


Fig.4 Syntax tree of *E div E* ⊕ *E*

图 4 *E div E* ⊕ *E* 的语法树

通过上述处理转换而得到的文法与 Object-Z 的原有文法并没有完全意义上严格的一一对应关系.因为这种对应关系导致的解析困难,本文工具中所使用的文法去除了一些不常用的规格说明文法(如 Stroke::=

Number).由于规则 1~规则 6 是基于 Object-Z 文法的语法规则进行处理的,尽管这种处理不能保证与原有文法完全等价,但为工具的实现提供了较大的便利,同时保证了可以处理大多数 Object-Z 规格说明.

3 测试用例的自动生成

经过 Object-Z 规格说明的解析,可以根据语法树产生测试用例.本节介绍自动产生符合组合覆盖准则(定义 1)和调用模式测试准则(定义 2)的生成测试用例的实现方法.

产生类中方法的测试用例的过程如下,每一步骤均由机器自动执行:

- 展开模式;
- 提取前置条件;
- 改写成文字集;
- 根据测试准则产生测试用例.

3.1 规格说明的扩展

Object-Z 语言为了编写和阅读的方便性,提供了多种模式包含的方法,使得最终的规格说明在表达上尽量简单.产生测试用例时需要提取出规格说明的语义,即需要还原这些模式,使其能展现最完整的信息.这个步骤称为模式展开(schema unfolding).其基本步骤为:

Step 1:从继承列表中找出所有被继承的父类,按照本步骤递归展开父类;

Step 2:根据继承列表中存在的重命名描述,进行重命名操作;

Step 3:把父类的所有模式按照操作模式复合的方法添加到子类中;添加的办法参照 Object-Z 中面向对象继承的语义,使用以下规则进行:

规则 a. 子类的操作模式名与父类的操作模式名相同时,父类的声明部分被包含到子类的声明部分,父类的谓词部分与子类的谓词部分一起组成合取多项式;

规则 b. 如果父类中的某个操作模式在子类中不存在,则直接复制父类的操作模式给子类;

规则 c. 如果子类中的操作模式不在父类中出现,则子类的操作模式就是完整的,无需添加;

Step 4:展开本类中存在的简写形式;

Step 5:展开本类中模式复合形式为完整模式;

Step 6:去掉模式中重复的谓词;

Step 7:改写模式中存在的如$\Rightarrow$,$\Leftrightarrow$等联结词.

3.2 测试用例的产生

得到了完整形式的规格说明后,可以提取规格说明的前置条件,随后就可产生测试用例了.首先,使用前置条件模式提取前置条件,具体步骤是:

(1) 从声明部分中删除变量和后状态变量,并在谓词部分对其用存在量词约束;

(2) 对谓词使用点规则(one-point-rule)替换后状态变量和输出变量,并化简获得简化后的前置条件.

将上述过程求得的谓词保存在一个 PredicateList 中,递归地从前置条件 PredicateList 中得到其文字集.

产生测试文字集合后,可根据组合覆盖准则(见定义 1)产生关于前置条件的测试用例.若该操作模式只包含本类的操作模式,则根据其前置条件得到文字集,而后采用组合覆盖准则产生测试用例.若该操作模式包含了其他类的操作模式,即调用模式,则使用调用模式覆盖准则(见定义 2~定义 5)产生相应的测试用例.

4 原型工具的设计与实现

此前,我们已经初步成功开发了可视化的 Object-Z 编辑器.在编辑器的基础上,可以对上述测试用例生成器的功能嵌入,同时要增加模式语法树存储模块.

4.1 总体结构

如图 5 所示,模式编辑框是 Object-Z 编辑器的功能,后续模式解析、语法树存储、测试用例生成即为上面所介绍技术的实现.

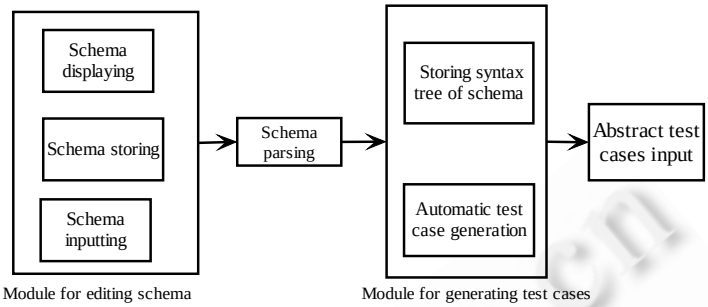


Fig.5 Overall structure

图 5 总体结构

具体来说,Object-Z 规格说明被序列化后可以保存在文件中;相反,也可以从文件中读入已被序列化的规格说明并显示成 Box 风格.该工具保留了转换成 LaTeX 格式文件的接口,可以把规格说明保存为 LaTeX 文件.为使用方便,本工具还可以把 Box 风格的规格说明输出为 JPG 格式的图片文件.

测试用例自动生成时,工具自动解析规格说明,规格说明被保存到语法树中.根据不同的数据类型存储到不同的类中,这些类都被存放在包 SyntaxTree 中,根据数据类型,确定了 76 个类来对应这些存储类型,这些类还包含了抽象测试输入数据的自动产生功能.

4.2 规格说明的可视化输入及显示

Fig.6 Symbol panel in the editor

图 6 编辑器中的符号面板

在 Object-Z 语言中存在许多数学符号,这些数学符号没有对应的 ASCII.为解决此问题,本工具创建了一种字体库(font),该字体库包含了所有需要用到的数学符号,并对这些数学符号进行了编码,范围从‘\uE11A’到‘\uE17C’,相关数学符号如图 6 所示.

模式编辑模块主要分布在 3 个包中,分别是数据存储包 m,数据显示包 v,数据输入包 c,遵循 MVC(model- view-controller)模型定义.根据规格说明的不同模式采用了不同的存储、输入和显示方式,在实现中主要包含的规格说明模式有:基本模式、泛型模式、类模式、公理模式、定义模式、自由模式和谓词模式.它们与相关实现类的对应关系见表 1.

Table 1 Corresponding relation between schema and class

表 1 模式与类的对应关系

Schema name	Stored classname	Displayed classname	Input classname
Basic schema	m_schema	v_schema	c_schema
Generic schema	m_generic	v_generic	c_generic
Class schema	m_class	v_class	c_class
Axiom schema	m_axiom	v_axiom	c_axiom
Definition schema, free schema & predicate schema	m_zed	v_zed	c_zed

表 1 中每一列的基本功能为:

- 存储类含有模式名称、声明和谓词等信息,实现了非普通字符的存储、判断模式是否处于编辑状态等功能;

- 显示模式类通过画图的方式把规格说明显示为 Box 风格,具有控制显示位置、输出数据等功能;
- 输入模式类负责数据的输入、编辑.需要创建不同的输入控件,把输入数据保存到存储类中,并驱动显示类显示数据.

每一种模式对应 3 个类,如一个类模式分别对应一个 m_class,v_class,c_class,这 3 个以元组的形式保存在 Vector 中,其结构如图 7 所示.其中,Type 用来区分模式类型.

m_class
v_class
c_class
Type

Fig.7 Structure of storing schema

图 7 模式保存在 Vector 中的结构

模式主要有如下几种操作:

- 创建模式:根据所要创建模式类型,调用相应的模式创建函数;
- 删除模式;
- 读取模式;
- 保存模式;
- 把模式保存为 LaTeX 格式文本.

整合以上功能模块,可实现如图 8 所示的 Object-Z 规格说明编辑器,所有的操作可能过图形界面输入,并实时可视地显示其效果.该界面由以下几部分组成:

- 菜单栏:包括文件操作、编辑处理(各种模式的输入及修改框架)、辅助工具(规格说明输出、语法分析等)、测试生成及简单的帮助系统等;
- 工具栏:菜单栏功能的快捷交互;
- 模式列表区:列出当前规格说明的所有模式,通过鼠标单击可在其间切换;
- 模式显示区:以 Box 型图显示规格说明的区域;
- 类基本信息编辑区:包括类名、形式参数列表、可见成员列表、继承类名、历史变量等信息的编辑;
- 状态模式相关信息编辑区:包括编辑状态声明及满足的谓词以及对系统初态及满足谓词的编辑;
- 局部定义和操作模式用户交互区:点击工具条可弹出相应的编辑框.

各个部分在界面上的位置如图 8 所示.

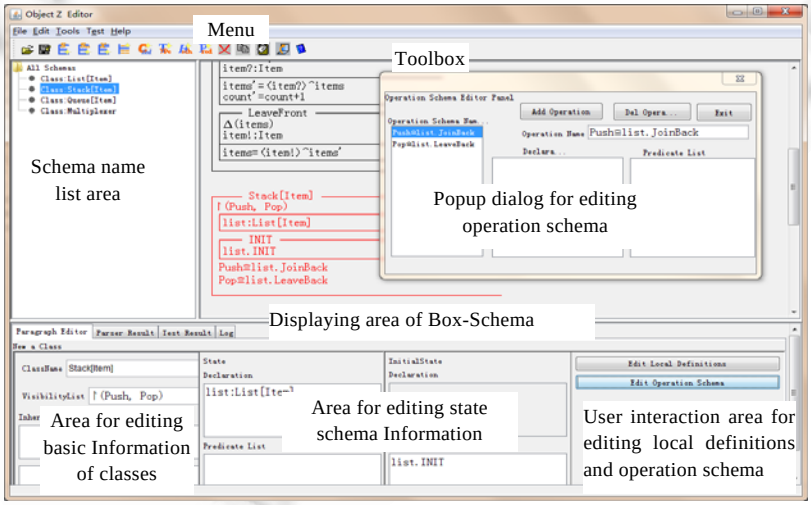


Fig.8 Editor of Object-Z specification

图 8 Object-Z 规格说明编辑界面

4.3 规格说明的解析及语法树存储

解析规格说明经历两个阶段完成,首先是由类 Lexer 负责词法分析,接着是类 Parser 负责语法分析.

类 Lexer 通过如图 9 所示的缓冲机制实现第 2.2 节向后“看”的功能.

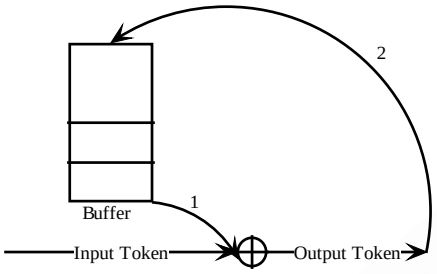


Fig.9 Buffering mechanism of inputting Token

图 9 输入 Token 缓冲机制

Parser 类提供了函数 parse,可循环调用 Lexer 类中读入 Token 的函数,直到处理完所有的 Token,算法如下:

(1) 根据读取的 Token 判断出后续的 Token 应该采用哪一条产生式进行归约;

(2) 调用该条产生式的解析函数,把解析的结果保存到对应的类中作为语法树的节点;

(3) 把生成的类作为其他类的一部分添加到其他类中,使之形成语法树.

例如表达式 *a*<*b*,经上述流程解析之后得到的语法树如图 10 所示.

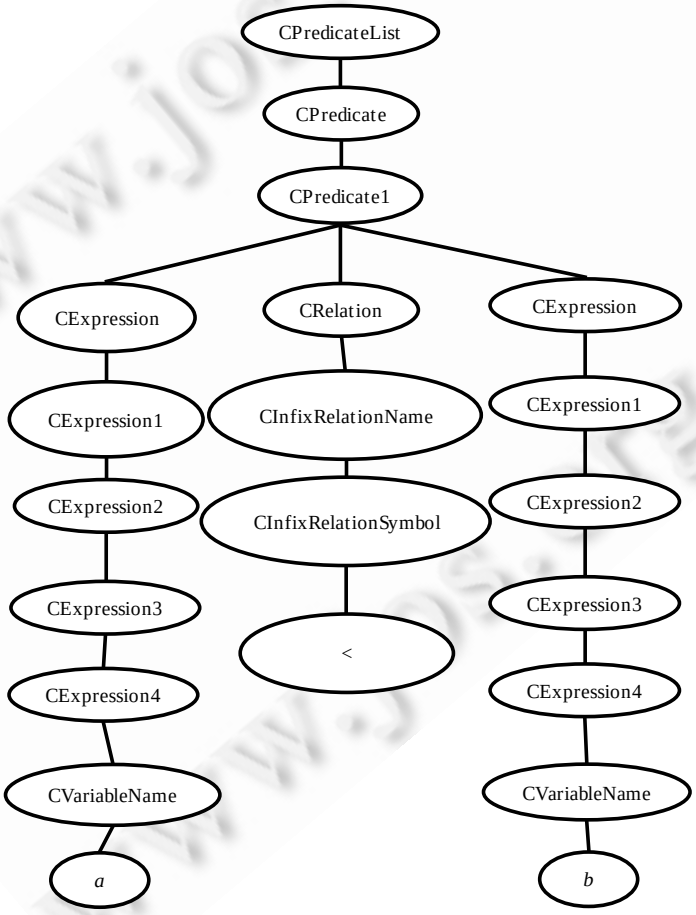


Fig.10 Syntax tree of *a*<*b*

图 10 表达式 *a*<*b* 的语法树

4.4 测试用例自动生成

为了产生测试用例,需要提取类模式中的操作模式,根据操作模式的不同表达形式和测试准则产生测试用例.如果该操作模式为普通的 DeltaList Declaration PredicateList[8]的书写形式,则使用组合覆盖测试准则,提取出所有的文字存储到一个数组 Vector 中.

例如,对于图 11 所示的 Object-Z 规格说明片段,某个类模式中的操作模式 Add 的的前置条件为 *incValue*?+*iValue*≤MAX∧*incValue*?=1,则存储在相应 Vector 的文字集为{*incValue*?+*iValue*≤MAX,*ncValue*?=1}.可对文字集使用组合覆盖测试准则,使文字分别取真值和假值,即产生了组合覆盖测试准则的测试用例.

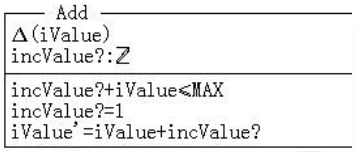


Fig.11 An Object-Z specification fragment

图 11 一个 Object-Z 规格说明片段

若操作模式为调用模式,并且当前操作模式使用了广义联结符,则需要对 Declaration 部分进行检查,如果某个变量为类模式类型或者集合类型,则使用组合模式覆盖准则产生测试用例,并对剩下的部分递归处理.

若未使用广义联结符且为非复杂调用模式,则使用组合模式覆盖准则产生测试用例;若为 Expression.Identifier 形式,则使用调用模式覆盖准则产生测试用例;如果是复杂调用模式,则根据语义转换成多个简单调用模式的形式.若为简单调用模式,则使用简单调用模式简单覆盖和简单模式全覆盖准则来产生测试用例.

5 实例研究及工具演示

本节以自动售货机为例,演示说明所实现工具的功能.该例中,自动售货机的软件规格说明是由 Object-Z 编写的,通过处理之后该工具产生测试用例.图 12 定义了模型 VendingMachine 的规格说明,并显示了其编辑界面.该售货机的 3 个操作模式 *Coin*,*GetChoc* 和 *AddChoc*,相应的功能分别为投币、购买巧克力和添加巧克力.该模型的具体功能见文献[12].

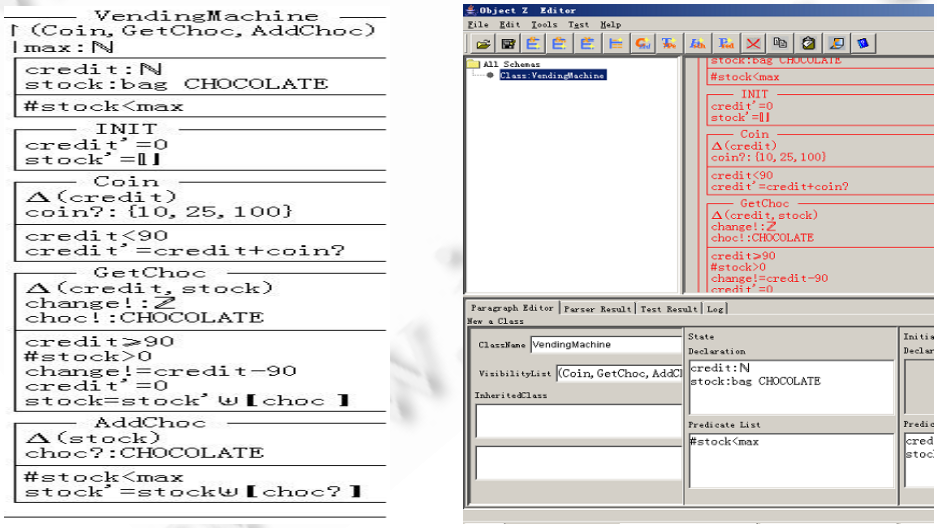


Fig.12 VendingMachine's specification and the associated editor interfaces

图 12 VendingMachine 规格说明及其编辑界面

选择菜单 Test->Test Cases,如图 13 所示的对话框选择需要产生哪个类的测试用例,从组合框中选择类名 VendingMachine 后,产生测试用例的结果如图 14 所示.

图 14 的第 1 行给出即将产生测试用例的操作模式名,接着给出该操作模式所包含的前置条件文字集,后面分别是该文字的抽象测试用例.对于 *GetChoc* 操作模式有两个文字,分别是 *credit*≥90 和#*stock*>0,在程序的结果中使用\geqslant 代表数学符号≥.文字后面跟着的字符“T”和“F”分别表示该文字在测试用例中的取值,如结果所示,第 1 个测试用例的各个文字分别取值为 true,true,第 2 个测试用例的取值分别为 true,false,依此类推.产生抽象的测试输入见表 2.

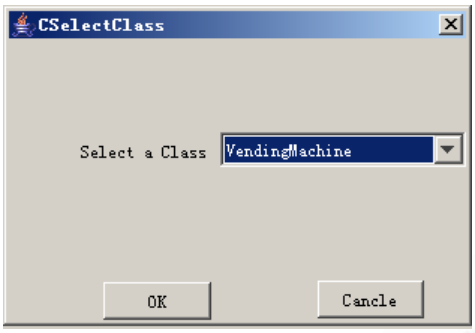


Fig.13 Selecting a class

图 13 选择待测试的类

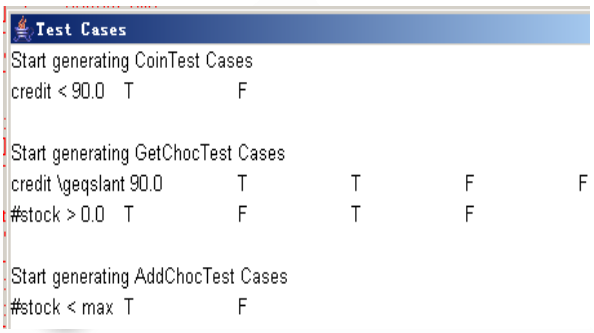


Fig.14 Generated test cases

图 14 测试用例生成效果图

Table 2 Abstract test cases input for class VendingMachine

表 2 类 VendingMachine 的抽象测试输入

No.	Operation schema name	Set of literals for precondition	Abstract Input
1	*Coin*	*credit*<90	T
2	*Coin*	*credit*<90	F
3	*GetChoc*	*credit*≥90, #*stock*>0	T,T
4	*GetChoc*	*credit*≥90, #*stock*>0	T, F
5	*GetChoc*	*credit*≥90, #*stock*>0	F, T
6	*GetChoc*	*credit*≥90, #*stock*>0	F, F
7	*AddChoc*	#*stock*<max	T
8	*AddChoc*	#*stock*<max	F

根据抽象的测试用例,可以手工或机器辅助方式进行测试用例实例化计算[17].即通过设计模式中各个变量(如 *Credit*,*Coin*?等)的具体数值,使其能够满足表 2 中“Abstract Input”列所示的谓词条件.同时,根据模式操作确定对应的输出变量值,即可得到具体的测试用例.更为详细的情况见文献[17].

从这个例子可以看出,该工具能够对类内方法有效地产生测试用例.由于模型 VendingMachine 规格说明中的 3 个操作模式均为简单操作模式,不涉及继承类等信息,因此,表 2 中的测试用例是使用定义 1 中的组合覆盖准则直接自动产生.如果规格说明涉及复杂调用模式或更为复杂的数据结构,则需要使用定义 2~定义 5 中的覆盖准则[17].

6 结 论

基于形式规格说明的测试用例生成逐渐成为研究热点,但目前还没有较好的工具支持自动产生抽象测试用例.

本文根据从 Object-Z 规格说明提取前置条件并自动产生测试用例的方法,在 Object-Z 规格说明编辑器的基础上设计实现了相应的自动测试工具.该工具通过对 Object-Z 规格说明的解析,根据自动生成的语法生成树,考虑函数操作语义,抽取相应的谓词文字集合,运用组合覆盖测试准则自动产生抽象测试用例.本文针对调用模式

设计并实现了产生测试用例的方法和若干测试准则,可以对类中成员函数调用进行测试.本项研究是基于规格说明的测试自动化方面的一次有效尝试.

随着形式方法研究的逐步深入和完善,形式规格说明技术将取得更为广泛的应用.本文所述的工具在应用中还存在一些问题,例如产生的测试用例只是输入数据,未包括输出结果等.因此,在本文工作的基础上,还可对完善自动生成测试用例的方法(比如扩展到类间测试等)、自动预期输出以及自动生成测试驱动程序等方面进一步研究.

References:

[1] Miao HK, Li G, Zhu GM. Software Engineering Language—Z. Shanghai: Shanghai Scientific and Technological Literature Press, 1999. 216−234 (in Chinese).

[2] Dick J, Faivre A. Automating the generation and sequencing of tset cases from model-based specification. In: Woodcock J, Larsen P, eds. Proc. of the Industrial-Strength Formal Methods (FME'93). Heidelberg: Springer-Verlag, 1993. 268−284. [doi: 10.1007/BFb0024651]

[3] Offut AJ, Liu S. Generating test data from SOFL specifications. Journal of Systems Software, 1999,49(1):49−62. [doi: 10.1016/s0164-1212(99)00066-7]

[4] Stocks P, Carrington D. A framework for specification-based testing. IEEE Trans. on Software Engineering, 1996,22(11):777−793. [doi: 10.1109/32.553698]

[5] Ammann P, Offutt AJ. Using formal methods to derive test frames in category-partition testing. In: Proc. of the 9th Annual Conf. on Computer Assurance. Gaithersburg: IEEE Computer Society Press, 1994. 69−79. [doi: 10.1109/CMPASS.1994.318466]

[6] Hierons RM. Testing from a Z specification. The Journal of Software Testing, Verification, and Reliability, 1997,7(1):19−33. [doi: 10.1002/(SICI)1099-1689(199703)7:1<19::AID-STVR124>3.0.CO;2-N]

[7] Burton S. Automated generation of high integrity test suites from graphical specification [Ph.D. Thesis]. York: University of York, 2002.

[8] Smith G. The Object-Z Specification Language. Massachusetts: Kluwer Academic Publishers, 2000. 1−142.

[9] Carrington D, MacColl I, McDonald J, Murray L, Strooper P. From Object-Z specifications to ClassBench test suites. Software Testing Verification Reliability, 2000,10(2):111−137. [doi: 10.1002/1099-1689(200006)10:2<111::AID-STVR204>3.0.CO;2-P]

[10] McDonald J, Strooper P. Translating Object-Z specifications to passive test oracles. In: Proc. of the 2nd IEEE Int'l Conf. on Formal Engineering Methods '98. Queensland: IEEE Computer Society, 1998. 165−174. [doi: 10.1109/ICFEM.1998.730580]

[11] Liu L, Miao HK, Zhan XD. A framework for specification-based class testing. In: Maryland G, ed. Proc. of the 8th IEEE Int'l Conf. on Engineering of Complex Computer Systems. Greenbelt: IEEE Computer Society Press, 2002. 164−171. [doi: 10.1109/ICECCS.2002.1181508]

[12] Liu L. Object-Oriented formal specification based test generation [Ph.D. Thesis]. Shanghai: Shanghai University, 2004 (in Chinese with English abstract).

[13] Liu L, Miao HK. Axiomatic assessment of logic coverage software testing criteria. Journal of Software, 2004,15(9):1301−1310 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1301.htm

[14] Malik P, Utting M. CZT: A framework for Z tools. In: Treharne H, King S, Henson M, Schneider S, eds. Proc. of the 4th Int'l Conf. of B and Z Users: Formal Specification and Development in Z and B (ZB 2005). Heidelberg: Springer-Verlag, 2005. 65−84. [doi: 10.1007/11415787_5]

[15] Miller T, Freitas L, Malik P, Utting M. CZT support for Z extensions. In: Romijn J, Smith G, van de Pol J, eds. Proc. of the Integrated Formal Methods. LNCS 3771, Heidelberg: Springer-Verlag, 2005. 227−245. [doi: 10.1007/11589976_14]

[16] King P. Printing Z and Object-Z LATEX Documents. Queensland: Department of Computer Science, University of Queensland, 1990.1−13.

[17] Hu XB. An approach to generating test case from Object-Z specification [MS. Thesis]. Shanghai: Shanghai University, 2008 (in Chinese with English abstract).

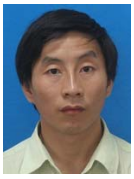
附中文参考文献:

[1] 缪淮扣,李刚,朱关铭.软件工程语言——Z.上海:上海科学技术文献出版社,1999.216−234.

[12] 刘玲.基于面向对象形式规格说明的测试用例生成技术[博士学位论文].上海:上海大学,2004.

[13] 刘玲,缪淮扣.对逻辑覆盖软件测试准则的公理化评估.软件学报,2004,15(9):1301−1310. http://www.jos.org.cn/1000-9825/15/1301.htm

[17] 胡晓波.基于 Object-Z 规格说明的测试用例生成方法研究[硕士学位论文].上海:上海大学,2008.



许庆国(1974－),男,山东聊城人,博士,讲师,CCF 会员,主要研究领域为软件的测试验证,形式化方法.

缪淮扣(1953－),男,教授,博士生导师,CCF 会员,主要研究领域为形式化方法,软件工程.

曹晓夏(1973－),女,博士,讲师,CCF 会员,主要研究领域为软件工程,形式化方法,模型转化.

胡晓波(1976－),男,硕士,主要研究领域为软件工程.