计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201000048

一种面向形式化表格需求模型的测试用例生成方法

汪文轩[1,2]　胡　军[1,2]　胡建成[1,2]　康介祥[3]　王　辉[3]　高忠杰[3]
1 南京航空航天大学计算机科学与技术学院　南京 211106
2 软件新技术与产业化协同创新中心　南京 211107
3 中国航空无线电电子研究所软件部　上海 200233
(wangwengx.1997@nuaa.edu.cn)

摘　要　现代安全关键性系统的软件规模和复杂性的快速增长给这类安全关键性软件系统的开发带来了很多挑战。传统文本文档的需求描述方法无法保证此类系统的开发进度和系统可靠性要求。为此文中提出了一种兼具可读性和可自动分析的形式化表格需求建模方法。文中介绍了一种针对这种表格模型测试用例的自动生成方法,工作包括对该形式化需求表格模型展开语义分析,建立需求模型的控制树结构,得到其测试等价类;为了减少不必要的测试,定义了不同安全级别的软件需求模型的测试覆盖标准,并针对不同覆盖率准则分别给出基于控制树结构的测试路径约束选择方法;对于每条路径约束测试等价类,提出了基于域错误的测试用例选择方法,能够自动生成所需的检测域错误的测试用例集。最后,通过一个需求模型实例展示了所提方法的有效性。

关键词:测试用例生成;表格需求模型;形式化需求模型;安全关键性系统
中图法分类号　TP311.5

Test Case Generation Method Oriented to Tabular Form Formal Requirement Model

WANG Wen-xuan[1,2], HU Jun[1,2], HU Jian-cheng[1,2], KANG Jie-xiang[3], WANG Hui[3] and GAO Zhong-jie[3]
1 College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China
2 Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 211107, China
3 Department of Software, China National Aeronautic Radio Electronics Research Institute, Shanghai 200233, China

Abstract　The rapid growth of the software size and complexity of modern safety-critical systems has brought many challenges to the development of such safety-critical software systems. Traditional text documents cannot guarantee the development progress and system reliability requirements. For this reason, this paper proposes a formal form requirement modeling method with both readability and automatic analysis. This paper introduces a method for automatically generating test cases for this tabular model. The work includes semantic analysis of the formal requirements tabular model, establishing the control tree structure of the requirements model, and obtaining its test equivalence classes. In order to reduce unnecessary testing, test path constraint selection methods are proposed based on those criteria. Through performing domain error test case selection, test cases are generated for each path constraint selected, which makes up a test case set for the requirement. At last, to demonstrate how we generate test cases form a requirement model, a case study is given.

Keywords　Test case generation, Tabular requirement model, Formal requirement model, Safety-critical system

1　引言

对于安全关键性系统如航空航天、汽车、医疗设备或核能发电来说,如何确保软件可靠性是首先需要解决的问题,在此类软件项目的研发中,软件验证(Verification)和软件确认(Validation)阶段(V&V)的工作占了大约50%～70%的软件开发资源[1]。常用的软件验证技术包括各类软件分析、评审以及测试技术。传统的开发流程中使用文字描述的设计文档作为媒介来进行上述过程,通过人工进行软件分析、评审和测试。然而,随着航电软件等安全关键性系统规模和复杂度的快速增长,传统的文字描述的设计文档已经难以保证开发进度和软件可靠性,一些能够自动测试和验证的建模方式如Z和VDM的可读性很差,学习门槛很高,不利于建模和评审,对于实现人员来说也不友好。UML等建模方法缺乏精确的定义,无法进行自动验证和测试[2],因而也不满足要求。

因此,在安全关键性系统的开发过程中,目前缺乏满足以

到稿日期:2020-10-10　返修日期:2021-03-20　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点基础研究发展计划(973计划)项目(2014CB744900)
This work was supported by the National Key Basic Research Development Plan(973)(2014CB744900).
通信作者:胡军(hujun@nuaa.edu.cn)

下要求的需求描述方法:对人员友好且易于阅读和理解,以便评审人员进行需求的非逻辑正确性的检查和开发人员据此进行开发实现;对描述的系统不做过多限制,即通用性很强;使用精确的数学语言描述,实现分析验证和测试过程的自动化,以自动地进行系统需求的逻辑正确性检查和基于需求的测试用例生成。

鉴于此,本文提出了一种基于四变量模型[3]的表格需求建模方法 VRM(Variable Relation Model),它使用二维表格的形式,通过关系演算逻辑系统和严格定义好的数学模型来描述系统需求,在保证可读性和易用性的前提下,精确地定义系统需求,该模型能够自动地进行需求模型验证,检查需求模型的逻辑正确性、完整性、无二义性。在目前工作的基础上,本文对此模型自动生成基于需求的测试用例的方法进行了研究。

本文在 VRM 准确定义的需求语义模型的基础上,分析了 VRM 需求模型中的路径结构,构建控制树结构,找出这些路径中最有可能发现软件典型错误的测试点(即测试敏感度较高的那些变量取值及组合),并考虑 DO-178C[4] 中的不同安全关键软件的测试准则要求,设计相应的算法来自动生成基于需求的测试用例集。

本文第 2 节概述了本文提出的 VRM 表格需求建模方法;第 3 节描述了基于 VRM 需求模式的测试用例自动生成方法的关键步骤和总体思路;第 4 节分析了 VRM 需求模型与需求实现代码结构之间的关系,并在此基础上提出了 VRM 模型控制树结构、VRM 模型测试覆盖率准则和记忆模型覆盖准则的测试路径选择策略;第 5 节给出了基于错误敏感度分析的测试用例选择策略;第 6 节用一个需求模型实例演示本文方法;最后给出了相关工作的比较,并总结全文。

2 VRM 需求模型

当前各种形式化方法和理论很多,但是来源于实际工程,并能真正应用于实际工程领域的需求描述很少;并且大多数形式化方法都采用各种很陌生的符号和逻辑公式的形式来表达模型,这使得绝大多数的形式化方法很难被现有的系统工程人员所理解和接受,学习应用的过程复杂,学习曲线非常陡峭,形式化建模过程和模型结果也容易出错。事实上,工程人员最了解的是工程领域知识,而且在安全关键性系统工程领域中最常使用的就是各种表格形式(如各种航电系统操作手册、飞行手册等),工程人员最熟悉的也是表格,因此本文采用的 VRM 模型就是同时具备表格化和形式化语义的需求模型[5]。

类似于在关系数据库中的数据存储处理方式,VRM 模型的直观形式是采用二维表格的形式来建立需求模型。关系数据库中的二维关系数据事实上是由关系演算逻辑系统来严格定义的数学模型(即形式化模型)。VRM 来源于四变量模型,其已在安全关键性领域成功应用近 40 年。VRM 模型使用定义清楚的变量和逻辑谓词来对变量和功能约束进行准确的描述;引入事件谓词对系统中的事件响应进行建模;对于需求中的模式转换关系提供了模式类和模式表的建模方法。其形式化定义如下。

VRM 形式化需求模型的六元组表达式形如$\{SV,C,E,F,TS,VR\}$;其中 SV 是所有状态变量的集合,它由一个四元组构成,四元组定义为 $SV=\{MV,CV,M,IV\}$,包含监督变量 MV、受控变量 CV、模式类 M、中间变量 IV。下面具体介绍六元组每个数据的功能。

MV:非空的不相交的监督变量集合,$MV=\{mv_1,mv_2,\cdots,mv_i\}$,$mv_1,mv_2,\cdots,mv_i$ 称为监督变量。

CV:非空的不相交的受控变量集合,$CV=\{cv_1,cv_2,\cdots,cv_j\}$,$cv_1,cv_2,\cdots,cv_j$ 称为受控变量。

M:非空的不相交的模式类集合 $M=\{mc_1,mc_2,\cdots,mc_m\}$,$mc_1,mc_2,\cdots,mc_m$ 称为模式,其中 mc_i 为某个模式类,它包含了该模式类下的所有模式,$mc_k=\{mc_{k_1},mc_{k_2},\cdots,mc_{k_n}\}$。

IV:非空的不相交的中间变量集合,$IV=\{iv_1,iv_2,\cdots,iv_k\}$,$iv_1,iv_2,\cdots,iv_k$ 称为中间变量。

TS:类型的并集,其中所有的类型都是值的非空集合。

VR:特殊的函数,用于将状态变量的名称映射到具体的值,表示状态变量的所有取值范围。对于 VR 中的所有 $r\in VR(r)$,r 是 SV 中的某个变量,TS 是 r 的值域类型,$VR(r)$ 是 r 可能取值的集合。

C:条件,表示单个状态变量上的谓词,如 $Altitude>500$ 表示当前的高度大于 500。条件是逻辑表达式,具有多种表达形式,可以为布尔变量 true,false,或布尔表达式 $c_i\odot c_j$ 等。$\odot\in\{AND,OR,NOT\}$ 表示逻辑操作符。$C=r\circ v$,其中 $\circ\in\{=,<,>,\neq,\geqslant,\leqslant\}$ 表示关系操作符。

E:事件,表示两个状态变量上的谓词,事件的通用表达式为:

$EVENT(S)GUARD\ D$

$EVENT\in\{@T,@F,@C\}$ 表示事件操作符;$GUARD\in\{WHEN,WHERE,WHILE\}$ 表示守卫操作符。

F:表格函数,所有的表格都是一个数学函数,都可以使用 F_i 表示。

在 VRM 模型中的表格函数包括三大类:条件表、事件表和模式转换表。这 3 类表都有相应的形式化语义定义,限于篇幅,下面给出了一个 VRM 需求模型实例。表 1 列出了一个条件表的例子,语义是基于状态依赖关系 $D_n=\{Pressure,Overriden\}$ 定义了受控变量 $SafetyInjection$ 的取值情况(即某个功能需求 F_6),其对应的 VRM 需求模型和对应的数学逻辑公式的模型如下。

表 1 条件表的例子和相应的逻辑公式

Table 1 Example of condition table

Mode Pressure	Condition	
High,Permitted	True	False
TooLow	Overridden	Not Overridden
SafetyInjection	Off	On

$$SafeInjection=F_6(Pressure,Overridden)$$
$$=\begin{cases}OFF, & \text{if } Pressure=High\vee Pressure=Permitted\vee(Pressure=TooLow\wedge Overridden=\text{true})\\ ON, & \text{if } Pressure=TooLow\wedge Overridden=\text{true}\end{cases}$$

3 测试用例生成的关键问题和方法框架

基于需求模型的测试是为了检验实现程序中不符合需求模型要求的错误[6]。从需求实现的角度看，VRM 需求模型中的监督变量对应于程序对于实现模块的输入变量，受控变量对应于程序的输出变量，在监督变量空间中任意选择一个可能的值 m，都可以通过模型检验的方法得到其对应的受控变量的取值 c。若需求实现程序是正确的，则程序输入 m，输出为 c。这样的一个(m，c)取值对即可作为一个测试用例。这是从 VRM 需求模型中获取基于需求的测试用例的基本方法。一个基于需求的测试用例测试的目标是一个实现程序对该输入的处理是否与需求模型所要求的一致[7]。

然而，我们不可能穷举所有可能的输入来组成测试用例集，可行的办法是，从所有可能的输入中寻找某个小的子集，这个子集应是能发现最多错误的子集。若要实现上述办法，就要对监督变量空间进行等价类划分，每一个等价类代表一条模型对一类监督变量的处理路径，这样如果一个等价类中的测试用例能够发现某个与需求模型规定的输入处理方式不一致的错误，则该等价类中的其他测试用例也能够发现同样的错误，因为它们执行的是相同的处理。

因此，生成测试用例集合的第一步即对需求模型中对监督变量的处理路径进行分析。为了便于后续的处理，我们将其路径结果以一个控制树的结构来表示。在实践过程中发现，尽管按照处理路径划分等价类，在规模较大的系统中路径数量还是很大，但是各个路径所代表的处理路径中有很多重复的子路径，因此本文还提出了一组需求模型控制树覆盖准则，以减小控制树的规模，去掉多余的等价类，减少多余的测试。

在划分完等价类后，在理想情况下，只需要从每个等价类中随机选择一个测试用例即可。然而，从大量的工程实践中发现，很多时候实现程序是错误的，但是由于某种巧合，在某个测试用例下，其输出是正确的。一个简单的例子是，在某个等价类下需求模型中输入 x 与 y，输出的映射关系为 $y=x^2$，实现人员漏掉了 2 次方而将此需求实现为了 $y=x$，这显然是错误的，但是当 x 为 0 时，该程序的输出为 0，与需求模型要求的结果是一致的。因此，对每个等价类仅仅随机选择一个测试用例是不够的(这也是一种可行的测试方案)，我们根据实际的开发经验提出了基于域错误的测试用例选择策略，为每个等价类选择对错误最敏感的测试集。

因此，从 VRM 需求模型中生成测试用例集合的关键步骤包括：模型处理路径划分及控制树生成、基于覆盖准则的处理路径精简、为每条处理路径等价类选择测试用例。

图 1 给出了本文方法的处理过程，它分为几个主要步骤。首先，在开始生成需求模型的测试用例之前，必须对其进行预处理，即路径划分和控制树结构的生成。然后，在控制树路径中为每个路径选择适当的分支，执行错误敏感性分析和域错误测试案例选择。最后，根据分析结果集成所有分支测试，从而形成基于需求模型的测试用例集合。

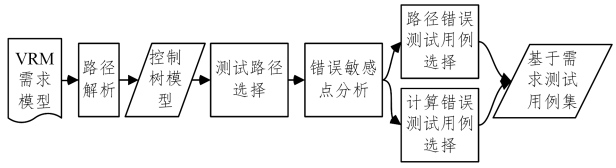


图 1 基于 VRM 需求模型的测试用例生成方法概述

Fig. 1 Overview of test case generation methods based on VRM requirements model

4 需求模型及其预处理

本节首先给出了一种航空电子系统形式化需求模型 VRM 的定义和示例，然后分析了 VRM 需求模型与需求实现代码结构之间的关系，最后在此基础上，提出了 VRM 模型控制树结构和 VRM 模型测试覆盖率准则。

4.1 将 VRM 需求模型转换为控制树模型

本文用前置条件、后置条件来表示需求模型[8]，如图 2 所示。一个前置条件即一个输入空间向量，同理后置条件即一个输出空间向量。关联谓词对先决条件(输入空间向量)进行分组归类。每个关联谓词描述输入空间对象上的数据和时序约束，功能关系将输出空间的对象描述为输入函数[9]。

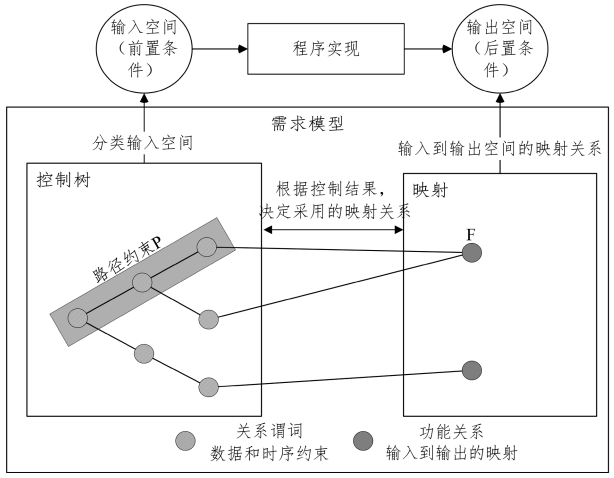


图 2 需求模型的前置条件、后置条件表示

Fig. 2 Pre/post-condition representation of requirement model

需求实现可能有很多种形式，然而所有实现正确的程序都应有对应的语句实现与需求模型等价的控制树结构。为了更好地描述需求模型的控制树结果，我们对 VRM 需求模型中的元素做出如下划分。

定义 1(模型判定) VRM 需求模型中的每一个输出变量表格函数中的每一个变量取值的条件称为一个判定。

定义 2(模型条件) 每一个判定中的单个逻辑谓词。

定义 3(模型语句) 输出变量、中间变量、模式表格函数的每一个取值。

表格函数 F_6 的模型元素划分如表 2 所列。事实上，对于正确的需求实现程序，模型判定实际上对应了程序中的一系列判定语句，模型条件对应了这些判定语句中的条件表达式，而模型语句对应了一系列的计算赋值语句，其执行结果应与模型语句等价[10]。

表 2 表格函数 F_6 的模型元素划分

Table 2 Model element division of tabular function F_6

No.	类型		
	语句	判定	条件
1	$S=ON$	$A\&B$	$A:Press=TooLow$ $B:Overwrite=false$
2	$S=OFF$	$C\|D\|(A\&\neg B)$	$C:Press=High$ $D:Press=permitted$ $A:Press=TooLow$ $\neg B$

本文给出了需求模型的控制树结构的定义。

定义 4（基本块） 对应一个模型语句。

定义 5（控制树） 可以用一个三元组表示，即 $T=(N,E,n_0)$，其中 N 表示控制树中所有的结点集，该结点集包含 3 个部分（模型条件、模型判定、模型语句）；E 表示控制树中所有有向边集合，E 的取值有 3 种情况，分别为逻辑与（A）、取反（F）、逻辑或（O）；n_0 表示模型的入口结点。

定义 6（路径约束） 由控制树中从入口结点到叶结点的一组不等式或等式组成，可以由元组 $PC_i=(p_i,f_i)$ 表示，其中 $p_i=\{c_{i1},c_{i2},\cdots,c_{ij}\}$，$f_i=\{f_{i1},f_{i2},\cdots,f_{ik}\}$。$c_{ij}$ 表示路径 p_i 上的模型条件 j，这些条件都是单个条件；满足 P 约束的所有变量的集合称为路径约束 P 的域 $D(P)$，域 $D(p)$ 由 p 的关系谓词 c 表示的域边界围成。$C(p)$ 表示路径 p 上的模型语句，p 对应于程序中的某些控制流语句，即执行路径，而 f 对应于这些控制流中的一系列计算赋值语句。

判定树的生成算法如算法 1 所示，判定树 T 初始化为一个判定结点 D，且右子树为 T'，依次遍历变革函数 F 中的每一条模型语句 s 和对应的判定 d，将 D 的值设置为 d，新建一个基本块 S，将其值设置为 s，将只包含 S 的判定树设置为 D 的左子树（即 True 分支）。分解判定结点 D，若只存在一个模型条件则分解结束，否则新建两个判定结点 D_1，D_2；若 D 为 $c\&d'$ 形式，则 D_1 设置为 c，D_2 设置为 d'，D_2 新建左子树且设置其值为 s，D_1 的左子树设置为 D_2，右子树为 T'；若 D 为 $c|d'$ 形式，则 D_1 设置为 c，D_2 设置为 d'，D_2 新建左子树且设置其值为 s，D_1 的右子树设置为 D_2，D_2 的右子树设置为 T'，继续拆分 D_2。用同样的方法继续遍历表格函数 F 的控制树。

算法 1 控制树生成算法 $BuildCT(F,T)$

输入：表格函数 F

输出：控制树 T

```
1. Foreach S in F
2.        T. lChild←S. s
3.    T. rChild←S. d
4.    Foreach complex decision d
5.      IF d. firstSymbol=& THEN
6.        T←s. d. firstCondition
7.        d. rest←T. lChild
8.        T. lChild←d. rest
9.          End
10.     IF s. d. firstSymbol=| THEN
11.       T←d. firstCondition
12.       d. rest←T. rChild
13.       T. rChild←d. rest
14.         End
15. BuildCT(F,T)
16. End
17. Return T
```

表格函数 F_6 的控制树结构如图 3 所示。

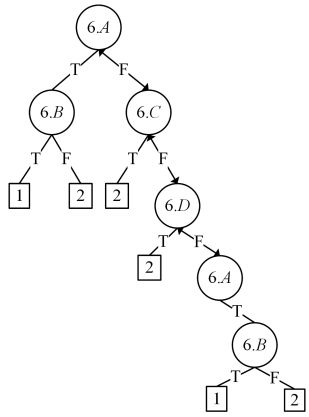


图 3 表格函数 F_6 的控制树结构

Fig. 3 Control tree structure for table function F_6

此控制树包含了 6 条路径约束，其中双向箭头连接线表示两个条件为互斥关系。

基于此控制树结构可以获取基于需求模型的测试用例[11]，我们将对测试控制树中的每一条路径约束选择满足该约束的测试点，并求解出每一个测试点的预期输出。

4.2 需求模型测试覆盖准则的定义

对于复杂的需求模型，控制树的分支数量很大，导致路径约束很多，尤其是模型条件很多的情况下路径数量会呈指数增长，而且控制树中会出现重复的子树结构，如图 3 所示。若对控制树中的所有路径约束都进行测试，则会导致测试路径约束过多，测试用例集过于庞大，存在很多冗余测试。

为了缩小测试规模，减少冗余测试，提高测试效率，我们定义了一个基于 VRM 需求模型的覆盖率准则[12-14]。

定义 7（对 VRM 模型的语句覆盖准则） 生成测试用例的控制树路径覆盖了所有的输出变量语句。

定义 8（对 VRM 模型的路径覆盖准则） 生成测试用例的控制树路径覆盖了所有的模型判定。

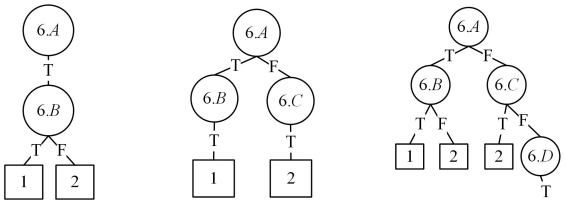
定义 9（对 VRM 模型的 MC/DC 覆盖准则） 生成测试用例的控制树路径，满足语句和路径覆盖，同时满足：1）模型中任意判定的所有可能结果都至少出现一次；2）任意判定的每一个条件的所有可能结果至少出现一次；3）每一个条件都能够独立地影响对应判定的结果，即存在两条路径，其中该条件对应的判定结果相反，该条件取值相反，该条件所在的判定中其他条件取值相同。

模型语句覆盖能够发现需求实现中输出函数映射的实现错误，如上溢出、下溢出、错误的计算映射实现。模型判定覆盖能发现需求实现中对需求表格函数的各种取值情况划分的覆盖情况。而需求 MC/DC 覆盖能够有效地降低由全部路径约束覆盖所生成的测试用例的数量，能够发现判定条件中逻辑操作符的实现错误，还能够发现将变量或表达式误实现为自身否定的错误，通过 MC/DC 覆盖准则能够实现通过测试较少的路径来检验模型中各个模型判定实现的正确性。

4.3 基于 VRM 模型测试覆盖率准则的测试路径约束选择

本节给出了基于 3 种需求模型测试覆盖率准则简化控制树结构的方法。

(1)模型语句覆盖测试路径选择。对于域语句覆盖,只需遍历需求模型控制树,为模型中的每一条语句找到一条包含该语句的路径。依次遍历该控制树,得到 6.A→6.B→1 和 6.A→6.B→2 两条路径即可。其简化后的待测试控制树如图4(a)所示。



(a)语句覆盖 (b)判断覆盖 (c)模型 MC/DC

图 4 应用 3 种覆盖率准则简化后的控制树结构

Fig. 4 Simplified control tree structure for F_6

(2)模型判定覆盖测试路径选择方法。对于每一个模型判定,首先寻找控制树中涉及到该判定的条件节点,然后选择左子树任意寻找一条路径,再在右子树中任意找到一条路径,对每个判定执行该操作,最后删除重复路径。满足判定覆盖的测试路径约束为:6.A→6.B→1 和 6.A→6.C→2 这两条路径。其简化后的待测控制树如图 4(b)所示。

(3)模型 MC/DC 覆盖测试路径选择方法。对于每一个判定,先列出其中的判定取值和条件取值情况,如表 3 所列。

表 3 表格函数 F_6 的模型判定取值

Table 3 Model decision value of table function F_6

No.	A	C	D	B	判定结果	判定
1	T	NG	NG	T	T	
2	F	NG	NG	NG	F	$A\&B$
3	NG	NG	NG	F	F	
4	NG	T	NG	NG	T	
5	NG	NG	T	NG	T	
6	T	NG	NG	F	T	$C\|D\|(A\&\neg B)$
7	NG	F	F	TF		

表 3 中,NG 表示判定结果与此条件无关,因此此变量无论取值为真还是假都不会影响整个判定的真假。得到此表后,遍历每一个条件,寻找只有该条件取值不同而其他条件相同,且判定结果相反的一对路径约束,其中 NG 既可以为 T 也可以为 F。合并所有路径,去掉重复路径。测试控制树如图 4(c)所示。

5 基于错误分析的测试用例选择策略

本节首先给出了安全关键性系统中有关代码实现错误的分类和定义,包括路径错误和计算错误;然后给出了测试用例集的错误敏感性的定义;最后给出了基于路径误差和计算误差的测试用例选择策略[15-17]。

5.1 两类代码的实现错误

第 4 节提出了需求模型的控制树结构和模型覆盖准则,并应用模型覆盖准则来获取要测试的路径约束集。对于每个路径约束 p,我们需要分析其模型条件和模型语句以选择适当的测试用例对其进行测试。

对于每个路径约束的测试用例选择,这里有两个极端的解决方案:

(1)在 P 的域 D 中随机选择一个测试点,以获得仅包含一个测试用例的测试用例集。

(2)选择 P 的域 D 中的所有测试点,以生成一组包含 D 中所有值的测试用例。

方案(1)几乎没有有效性可言,发现错误的可能性很低,但其可行性却很高,生成测试用例集及使用测试用例集花费的代价很低。方案(2)与之相反,只要程序在此功能点上发生错误都会被测试出来,然而其可行性极低,生成测试用例集及使用测试用例集花费的代价可能极高,甚至无法完成。

为特定路径约束选择测试点的基本原则是,在确保测试质量的同时,使用尽可能少的测试点。因此,针对每一条待测试路径,我们应该选择其中对错误最敏感的测试点进行测试。

为了找到对错误最敏感的测试点,我们对需求实现中可能发生的错误进行分类和定义。路径约束由(p,f)二进制表示,并且实现代码中可能存在两种类型的错误。p 错误是模型条件实现错误,f 错误是模型语句实现错误。

定义 10(路径错误) 假设 P 是需求模型 M 的需求实现。假设 P 的路径集 p 和 M 的路径约束集 p^* 之间存在同构,使得对于所有路径对(p_i,p_i^*),$C(p)=C(p^*)$,但是存在路径对(p_k,p_k^*),$D(p_k)\neq D(p_k^*)$,此时需求实现 P 中包含路径错误。

路径错误是由域边界的移动引起的,域边界偏移的 3 种情况如图 5 所示。

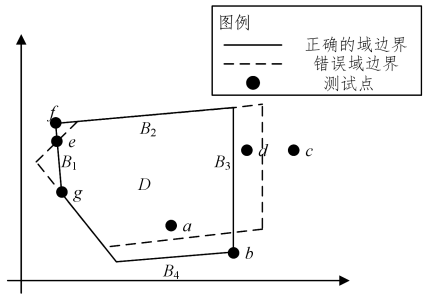


图 5 3 种域错误

Fig. 5 Three kinds of domain errors

图 5 中,实线表示需求模型中某路径约束的域范围,虚线表示错误的需求实现中对应该功能点上的域范围。其中,域边界 B_4 偏移减小了域 D。测试点 a 仍然能够产生正确的输出,因为尽管边界 B_4 发生了变化,它仍在正确的域 D 中。但是,边界 B_4 已移过测试点 b,导致其不在 D 中,而位于域 D 外,由于在测试测试点 b 时程序将遵循错误的路径,因此该测试点将产生错误的结果。由于边界 B_3 移动,域 D 扩大,此处的测试点 c 仍位于域 D 外,因此对 c 点的测试将被正确处理,但是 D 测试点能够检测到域边界的偏移,它本应位于域 D 外。最后,测试点 e,f,g 时仅测试点 f 能检测到错误,因为边界 B_1 偏移导致其位于域 D 外,其执行结果与预期输出不一致。

计算错误指发生在路径约束 P 上的赋值或计算语句序列组成的计算函数与需求模型对应功能点的 F 功能关系不

一致的情况，具体定义如下。

定义 11（计算错误） 假设 P 是需求模型 M 的需求实现。假设 P 的路径集 p 和 M 的路径约束集 p^* 之间存在同构，使得对于所有路径对 (p_i, p_i^*)，$D(p)=D(p^*)$，但是存在路径对 (p_k, p_k^*)，$C(p_k)\neq C(p_k^*)$，此时需求实现 P 中包含计算错误。

5.2 域错误测试用例选择策略

当程序发生域错误时，表现为域边界的偏移，因此我们用域边界的偏移程度来描述域错误的严重程度，域边界偏移越大，域错误就越严重。而越轻微的域错误意味着绝大多数测试点都不能发现这个错误。因此，我们提出错误敏感度这个概念来描述测试点能发现轻微域错误的能力，发现域错误能力越强意味着越有可能检测到轻微的域错误[15-21]。下面给出具体的定义。

定义 12（测试点域错误敏感度） 测试点 p 上能检测到的最小域边界偏移 d 越小，测试点的域错误敏感度 S 越高。即 $S(p)=1/d$，域错误敏感度越高，测试点检测到域错误的可能性就越大。

因此，越靠近域边界的点对域错误的敏感度越高。位于域边界上的点 b 对该边界上的域错误敏感度是最高的。应该注意到，在图 5 中，两条域边界的交点 b 对两条域边界 B_4，B_3 的偏移敏感度都很高，而位于域边界上的点 E 对域错误的敏感度高于域内点 A，因此最终的域错误敏感度排序为 $S(b)>S(e)>S(a)$。

我们可以得出以下结论：在域中，测试点对域误差的敏感性从高到低依次是域顶点、除域顶点之外的域边界上的点、域边界附近的测试点以及域内部的测试点[21]。

5.3 域错误测试用例选择

在域顶点上的测试点的价值显然是最高的，对于一个给定的域表达式 $D=\{L_1, L_2, \cdots, L_n\}$，即使是最简单的 n 维线性域空间，其域顶点数量也有 2^n 个，因此要得到其中的所有域顶点几乎是不可能的，故我们采用一种启发式的算法来获取部分域顶点，如算法 2 所示。

算法 2 获取域顶点测试点的启发式算法

输入：D=$\{L_1, L_2, \cdots, L_n\}$域表达式

输出：p 测试点

1. L_{item}←GetBaseItem(D) //分类逻辑表达式
2. L_{base}←GetBaseClause(D)
3. L_{clause}←GetComplexClause(D)
4. InitRange(D,L)
5. InputVars←GetInputVars(D)
6. Foreach var in InputVars//对所有变量执行
7. SubstitudeVar(var. Bound,D)
8. p. var←var. Bound
9. Propaga() //向其他变量做推导
10. End
11. Return p

(1)首先将域表达式中的逻辑表达式分为 3 类。

1)基本项：一个输入变量与一个常数的逻辑关系，如 $x<0$，$y>9$ 等。

2)基本子句：一组变量或函数与常数的逻辑关系，如 $x+y>10$，$\cos(x)>0.5$ 等。

3)复杂表达式：变量之间的关系，关系符号左右都不是常数，如 $x+y>z$ 等。

(2)依次使用基本项、基本子句、复杂表达式初始化域所涉及到的变量的取值范围。

(3)选择其中一个变量的上界，将所有非基本项逻辑表达式中的该变量替换为此值的上界，此步骤对于下界值的操作类似。

(4)重新约束其他变量范围，重复步骤(2)－步骤(4)的操作。

(5)当所有变量的取值都确定后，得到一个域顶点测试点。

根据所需的测试用例的规模，调整选择测试点时变量取值的先后顺序，即可得到不同的域顶点组成测试用例的集合。

在测试用例选择过程中，需要对测试路径的域表达式进行约束求解，并选择一个变量的最大值或最小值执行边界点选择启发式算法，当无法得出有效解时，表示输入变量之间存在冲突需要重新选择，这样可以避免选择出输入值存在冲突的测试用例。

6 实例研究

我们通过一个简单的示例演示本文方法，首先对给定的需求进行形式化建模，同时使用编程语言来实现需求；然后使用该方法生成测试用例集 T；最后修改编程实现，故意注入错误集 E，使用测试用例集测试注入了错误的程序，并分析测试结果。

6.1 需求实例

表 4 列出了一个航电显控系统中的一个文本形式要求。

表 4 一个文本形式的需求

Table 4 A text form requirement

编号	需求描述
1	在民航机载系统中的 Navigation Display 显示单元中有一个列示航路点信息的模块，用于为调用模块提高一定范围内的航路点信息。其调用格式如下： *getWayPoint* [*Loc*1] [*Loc*2|END|*Count*]
2	第一个参数规定了待显示的内容的起始编号，为一个 Short 型整数，若没有指定则默认为 0；该编号必须在实际航路点列表最大范围内，最大范围为 10000
3	第二个参数规定了要显示的航路点的数量。如果指定的是 *Loc*2，则确定了要显示的航路点范围最大编号，该编号为一个 Short 型整数，且必须大于或等于起始编号，同时 *Loc*2 也必须在实际航路点列表最大范围内。如果第二个参数为 END，那么从起始编号到列表末尾的所有航路点都将被显示出来；如果第二个参数为 *Count*，*Count* 必须为 Short 型整数，*Count* 和 *Loc*1 之和不能超过实际的列表长度＋1；若为空则默认类型为 Count，值为 1
4	在输出航路点结果时，输出方式按每 4 个航路点数据作为一个数据包： {WayPiont1，WayPoint2，WayPoint3，WayPoint4} 且无论 *Loc*1 为何值，或者要显示的航路点数量多大，每个数据包数据量总为 4，*Loc*1 包含在第一个数据包中
5	可能产生的错误如下： 1. E1 无效的调用格式 2. E2 所需显示的航路点数量超过了实际航路点总数范围 3. E3 请求的航路点编号非法(为 0 或负数)

我们对此需求进行建模，此需求对应的 VRM 模型包括两个监视变量：

$MV=\{ExistParm1, Loc1, ParmType, Loc2, Count\}$

监视变量的含义如表 5 所列。

表 5 监视变量表

Table 5 Monitor variables

名称	描述	类型	范围
ExistParm1	第一个参数存在？	Bool	{T,F}
Loc1	第一个参数	Short	Short
ParmType	第二个参数类型	ENUM	{END, *Loc2*, *Count*, None}
Loc2	结束位置	Short	Short
Count	数量	Short	Short

存在 3 个控制变量，具体如下：

$CV=\{errorType, isMultiPackage, isFirst\}$

受控变量的含义如表 6 所列。

表 6 受控变量表

Table 6 Controled variables

名称	描述	类型	范围
errorType	错误类型	ENUM	{NONE, E1, E2, E3}
isMultiPackage	返回的数据是否多行	Bool	{T,F}
isFirst	*Loc1* 所在的航路点是否在其所在数据包的第一个	Bool	{T,F}

模式为空，包括 8 个中间变量：

$IV=\{isLoc1Valid, isParam1Exist, isParam2Exist, IsLoc2Valid, isCountValid, isOverFlow, isBellow\text{-}Zero\}$

中间变量的含义如表 7 所列。

表 7 中间变量表

Table 7 Inner variables

名称	描述	类型	范围
isLoc1Valid	*Loc1* 合法	Bool	{T,F}
isParam1Exist	参数 1 存在	Bool	{T,F}
isParam1Valid	参数 1 合法	Bool	{T,F}
isParam2Valid	参数 2 合法	Bool	{T,F}
isParam1Exist	参数 2 存在	Bool	{T,F}
isLoc2Valid	*Loc2* 合法	Bool	{T,F}
isCountValid	*Count* 合法	Bool	{T,F}
isOverFlow	所需显示的航路点数量超过了实际航路点总数范围	Bool	{T,F}
isBellowZero	请求的航路点编号合法	Bool	{T,F}

对于每一个中间变量及输出变量，都存在一个对应的表格函数，限于篇幅，表 8－表 10 仅列出了受控变量的表格函数。

表 8 表格函数 F_1

Table 8 Table function F_1

Mode	Condition			
	isParam1Valid & *isParam2Valid* & NOT (*isOverFlow*) & NOT(*isBellowZero*)	NOT(*isParam1Valid* & *isParam2Valid*)	*isOverFlow*	*isBellowZero*
errorType	NONE	E1	E2	E3

表 9 表格函数 F_2

Table 9 Table function F_2

Mode	Condition	
	errorType=NONE & ((*ParmType* = END & (*ExistParm1*=F | *Loc1*< MAX−4) | *ParmType*= *Count* & *Count*>4) | (*ParmType* = *Loc2* & *Loc2*−*Loc1*>4))	*errorType* = *NONE* & NOT((*ParmType*=END & (*ExistParm1* = F | *Loc1* < MAX−4) | (*ParmType* = *Count* & *Count* > 4) | (*ParmType* = *Loc2* & *Loc2*−*Loc1*>4))
isMultiPackage	T	F

表 10 表格函数 F_3

Table 10 Table function F_3

Mode	Condition	
	errorType=NONE & ((*ExistParm1* = F | *ExistParm1*=F& *Loc1* % 4=0)	*errorType*=*NONE* & NOT((*ExistParm1* = F | *ExistParm1* = F& *Loc1*% 4=0)
isFirst	T	F

6.2 控制树结构生成和测试路径抽取

我们应用第 4 节的方法来生成及简化控制树结构，需求模型的控制树结构如图 6 所示。其中，C 节点表示模型需求模型中出现的模型条件，D 节点表示模型中出现的模型判定节点，S 节点表示模型中出现的所有输出可能的赋值情况。

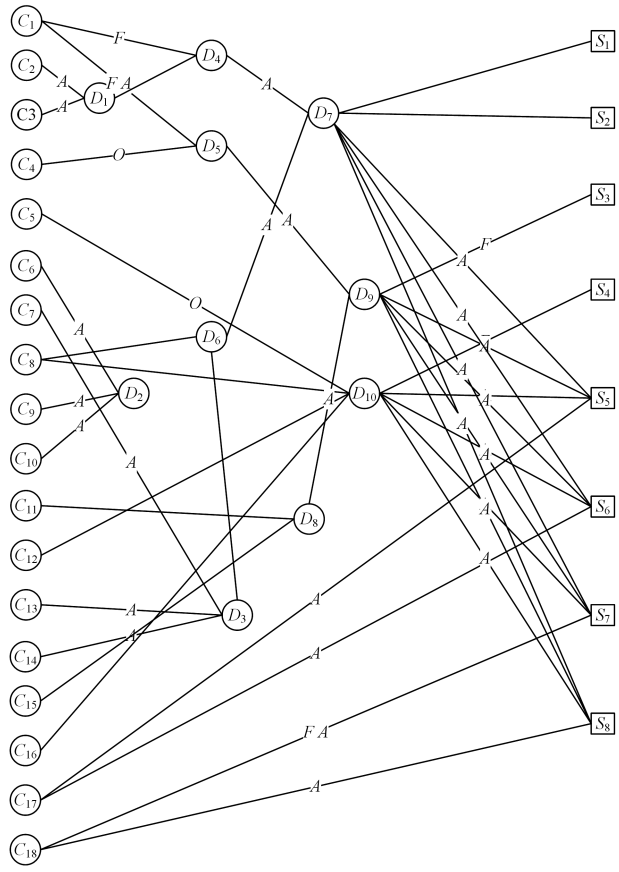


图 6 需求模型的控制树结构

Fig. 6 Control tree structure of requirement model

6.3 测试用例选择

从模型语句节点出发，为每条路径执行域错误测试用例选择，得到的部分测试用例如表 11 所列。通过简化控制树结

构和域错误测试用例选择可以得到在可接受的测试规模下能够覆盖到需求中所有可能的模型条件、模型语句以及模型判定的测试用例集合，该测试用例集合能够针对需求中的关键部分进行测试。

表 11 测试用例表

Table 11 Test cases set

No.	受控变量			监视变量		*ParmType*	*Loc2*	*Count*
	errorType	*isMultiPackage*	*isFirst*	*ExistParm1*	*Loc1*			
1	E1	—	—	T	10000	END	—	—
2	E1	—	—	T	10000	*Loc2*	10000	—
3	E1	—	—	T	0	*Loc2*	0	—
4	E2	—	—	F	—	*Count*	—	10002
5	E2	—	—	T	0	*Loc2*	—	10001
6	E2	—	—	T	0	*Loc2*	—	65535
7	E2	—	—	T	10000	NONE	—	—
8	E3	—	—	T	10000	*Loc2*	0	—
9	E3	—	—	F	—	*Count*	—	0
10	NONE	F	T	F	—	*Count*	—	1
11	NONE	F	T	F	—	*Count*	—	3
12	NONE	F	T	F	—	*Loc2*	3	—
13	NONE	F	T	T	10000	*Loc2*	10001	—
14	NONE	F	F	T	1	*Loc2*	4	—
15	NONE	F	F	T	9997	*Loc2*	10000	—
16	NONE	T	F	T	1	*Count*	9999	—
17	NONE	T	F	T	3	END	—	—
18	NONE	T	T	T	9992	*Count*	—	8
19	NONE	T	T	F	—	END	—	—
20	NONE	T	T	T	5	*Loc2*	10001	—

结束语 本文从方法论的角度介绍了一种在航空电子软件开发过程中通过分析形式需求模型生成测试用例的方法，并进行了理论分析和方法框架的设计。在后续的工作中，我们将设计实现一个软件工具来实施整个方法，考虑多层级的需求模型测试用例生成方法，并用较大规模的航电需求模型实例来进行验证。

工业界和学术界针对基于形式化需求模型的测试用例生成提出了很多方法，其中具有代表性的有：Carvalho 等[22]开发了 T-VEC 系统，该系统使用定理证明方法来生成测试用例。在证明公式的过程中，该工具会生成一个带有特定值的测试向量，用于输入和输出。然后，可以通过显式构造一个约束序列中所有状态的公式来获得将系统引导至感兴趣状态的输入序列。Carvalho 等[22]还讨论了定理证明方法在生成测试用例中的使用。Ammann 等提出了一种基于突变的新颖方法，其中 SMV 用于生成测试后代序列。通过将突变应用于规格和属性，可以获得大量的测试序列。Callahan 使用代表规范的过程来检查由仿真程序生成的过程追踪信息。通过这种方式，可以检测和分析软件实现和规范（或者说一组属性）之间的差异。Engels 等也描述了一种使用 Spin 生成测试序列的方法。与现有的工作相比，本文提出了从需求模型结构上进行分析的控制树模型，以及模型覆盖率的概念，可以有效地减少模型分析的规模，避免了现有工作中经常出现的组合爆炸问题，并且分析了安全关键性系统实现过程中的两类错误，为安全关键性系统软件实现中经常发生的错误给出了针对性的测试用例生成方法。

参 考 文 献

[1] TARHAN A, DEMIRORS O. Investigating the effect of variations in the test development process: A case from a safety-critical system[J]. Software Quality Journal, 2017, 19(4): 615-642.

[2] FRANZ T, LüDTKE, D, MAIBAUM O, et al. Model-based software engineering for an optical navigation system for spacecraft [J]. Ceas Space Journal, 2018, 10(2): 147-156.

[3] PATCAS L M, LAWFORD M, MAIBAUM T. implementability of Requirements in the Four-Variable Model [J]. Science of Computer Programming, 2015, 111(1PT. 2): 339-362.

[4] BROSGOL B. Do-178c [J]. ACM SIGAda Ada Letters, 2019, 11(4): 127-136.

[5] WANG R, KRISTENSEN L M, MELING H, et al. Automated Test Case Generation for the Paxos Single-decree Protocol using a Coloured Petri Net Model[J]. The Journal of Logic and Algebraic Programming, 2019, 104(APR.): 254-273.

[6] EISENMANN U, ALLEN J L. New Requirement-Definition and Verification Techniques According to DO-178C, DO-331, and DO-333[C]// AIAA Infotech @ Aerospace. 2016: 134-140.

[7] BRÜCKNER I, WEHRHEIM H. Slicing an Integrated Formal Method for Verification [M] // Formal Methods and Software Engineering. Springer Berlin Heidelberg, 2018.

[8] SAHOO R, RAY M. PSO based test case generation for critical path using improved combined fitness function [J]. Journal of King Saud University-Computer and Information Sciences, 2020, 32(4): 479-490.

[9] SANFORD F. A practical guide to SysML: the systems modeling language(2nd ed)[M]. Morgan Kaufmann, 2019.

[10] AITOR A, AGIRRE J A, SAGARDUI G. Seeding strategies for multi-objective test case selection: an application on simulation-based testing[C]// Genetic and Evolutionary Computation Conference 2020(GECCO'20). 2020: 45-54.

[11] CHAO T, JUNWEI F, LING X, et al. Application of MBSE

Method During Landing Gear System Design for Civil Aircraft [J]. Civil Aircraft Design & Research, 2015, 12(2): 345-361.

[12] LAILY H N, DAWOOD Y S. A Review on Test Case Generation Methods Using UML Statechart[C]// 2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE). IEEE, 2020: 38-46.

[13] ZHENG Y L, MA L H, ZHANG L Y, et al. On the convergence analysis and parameter selection in particle swarm optimization [C]// International Conference on Machine Learning & Cybernetics. IEEE, 2003: 61-72.

[14] YUE-HUA D, XIN F U, XIAO-NING Z. Convergence Analysis of FEM Model Based on Semi-circular Bending Test[J]. Western China Communications Science & Technology, 2019, 11(2): 145-167.

[15] WANG J M, WANG X H, MA Y Y, et al. Hierarchical Combination Design Method of Test Cases Based on Conditional Constraints[C]// IEEE International Conference on Software Quality. IEEE, 2017: 80-90.

[16] WU J L, H S F, D H N. Integration test case generating method based on UML[J]. Computer Engineering and Design, 2018, 23(2): 231-245.

[17] DASH S, PANDA N, ACHARYA A A. Model-based test case prioritization using UML activity diagram and design level attributes[J]. Advances in Intelligent Systems and Computing, 2018, 672(8): 380-390.

[18] WANG R, LI Z, JIANG S, et al. Regression Test Case Prioritization Based on Fixed Size Candidate Set ART Algorithm[J]. International Journal of Software Engineering and Knowledge Engineering, 2020, 345(2): 246-266.

[19] VERMA R P, GOPAL B, BEG M R. Data Structure & Algorithm for Combination Tree to Generate Test Case[J]. International Journal of Computer Science Issues, 2015, 8(3): 330-350.

[20] ANDRES N, FINGERHUT A K J, et al. p4pktgen: Automated Test Case Generation for P4 Programs[C]// Symposium. 2018: 88-96.

[21] NIKOLETA A, YANNIS S, FAUSTO B, et al. A DSM Test Case Applied on an End-to-End System, from Consumer to Energy Provider[J]. Sustainability, 2020, 10(4): 935-958.

[22] CARVALHO G, FALCAO D, BARROS F, et al. Test case generation from natural language requirements based on SCR specifications[J]. Science of Computer Programming, 2018, 95(3): 275-297.

WANG Wen-xuan, born in 1997, M. S candidate. His main research interests include software safety analysis and software test, etc.

HU Jun, born in 1973, Ph. D, associate professor, is a member of China Computer Federation. His main research interests include model-based safety analysis, software verification and embedded system design, etc.