基于 SPARDL 的模型和程序一致性测试*

陈玉祥[1]，蒲戈光[1]，綦艳霞[2]，陈铭松[1]，王　政[1]，陈朝晖[2]，顾　斌[2]
(1. 华东师范大学 软件学院，上海 200062；2. 北京控制工程研究所，北京 100080)

摘　要：针对周期控制系统的时序一致性进行研究，提出基于 SPARDL(space aircraft description language)的模型和程序一致性测试方法，通过模型抽取获取模式迁移图和控制流程图，通过程序插桩获取程序执行路径，实现了自动检测周期控制系统中的模式迁移和模块调用的一致性，给出了基于模式迁移图和控制流程图的覆盖检测并用于指导程序测试用例的生成。结合具体的周期控制系统，验证了一致性测试方法在实际工程中的有效性。

关键词：时序一致性；SPARDL；一致性测试；接受检测；覆盖检测
中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1001-3695(2013)03-0787-05
doi:10.3969/j.issn.1001-3695.2013.03.037

SPARDL based model-code conformance testing

CHEN Yu-xiang[1], PU Ge-guang[1], QI Yan-xia[2], CHEN Ming-song[1], WANG Zheng[1], CHEN Chao-hui[2], GU Bin[2]
(1. *School of Software Engineering, East China Normal University, Shanghai* 200062, *China*; 2. *Beijing Institute of Control Engineering, Beijing* 100080, *China*)

Abstract: Focusing on temporal conformance of periodic control systems, this paper proposed a testing method based on SPARDL for model-code conformance testing. It generated the mode diagram and control flow graph by mode extraction, and produced the program execution traces by program intrumentation. It also presented a algorithm to automatically check the consistency about mode transition and module calling inperiodic control systems. And it presented another algorithm to do coverage checking based on mode diagram and control flow graph and guide test case generation. Finally, it checked acasestudy about a specificperiodic control system by the proposed approch and it shows the effectiveness of this approach in industry.

Key words: temporal conformance; space aircraft description language; conformance testing; acceptance checking; coverage checking

在软件开发的生命周期中，测试是一个非常重要的阶段。一致性测试作为一种功能性测试被广泛应用于软件测试过程中。一致性测试是指检测软件产品的实现是否满足规范的要求[1,2]。在国内外研究中，协议一致性测试是一致性测试研究中最常见的一种，主要用于检测协议实现与协议规范是否一致[3]。对于实时系统的一致性测试也有相关研究，主要方法就是用时间自动机来描述实时系统，用输入/输出流来描述实时系统需要满足的规范，然后检测输入/输出在时间关系上是否满足一致性[4]。现有的模型检测工具在对大中型系统作一致性测试时存在状态空间爆炸的瓶颈[5,6]。近年来，基于模型的软件测试技术得到了快速的发展。基于模型的测试是一个轻量级的、形式化的验证软件系统的方法[7~9]。针对测试任务，通过对软件功能和结构进行抽象并用易于解的方式进行描述，获得测试模型，然后通过检测算法进行检测，从而在软件生命周期的早期检测出需求中可能存在的问题，但是基于模型的测试并不能保证模型和程序的一致性。文献[10]针对 Java 程序提出了基于 UML 活动图的测试用例生成，对 UML 活动图和 Java 程序的一致性也进行了测试。随着计算机技术的快速发展，计算机软件在嵌入式周期控制系统中发挥着越来越重要的作用。为了方便软件工程师建模与分析，本文采用形式化建模语言 SPARDL 为周期性控制系统建模。SPARDL 是一种用于控制系统的需求建模语言，尤其适用于基于模式的、有着混合的(连续的/离散的)状态、有限的周期行为和通信特征的控制系统。在每个周期中，系统都会并且只会精确地处在一个模式下，它可以停留在这个模式，或者根据它现在的状态转换到另一个模式。

周期控制系统是一种不会终止的反应式系统(reactive system)，这种系统按照给定的时间反复执行计算任务。周期控制系统的时序性至关重要，容易出错且不易检测，因此保证周期控制系统的时序性尤为重要。由于系统的非终止性，由工程师手动地去检查时序性费时费力且容易出错。对于周期控制系统的时序性，模式迁移的时序一致性关系和模块调用的时序一致性关系是控制工程师关注的两个主要时序关系，本文提出了一种基于 SPARDL 的模型和程序一致性测试用于对周期控制系统的时序性进行自动检测。

1　SPARDL

SPARDL 作为周期控制系统的建模语言分为三个层次：顶层展示各个模式及模式迁移关系；中间层描述每个模式的控制流程；底层是具体的控制算法。层次结构既反映了嵌入式周期控制系统本身的特征，也有利于软件工程师理解开发需求，分解开发任务[11~13]。

收稿日期：2012-07-07；**修回日期**：2012-09-04　　**基金项目**：国家自然科学基金资助项目(90818024)

作者简介：陈玉祥，男，硕士，主要研究方向为程序分析与验证(chyx06@hotmail.com)；蒲戈光，男，教授，博导，主要研究方向为程序分析与验证；綦艳霞，女，工程师，主要研究方向为嵌入式软件开发；陈铭松，男，副教授，主要研究方向为软件分析与测试、形式化验证；王政，男，博士，主要研究方向为模型检测、形式化验证；陈朝晖，男，工程师，主要研究方向为嵌入式软件开发；顾斌，男，研究员，主要研究方向为嵌入式软件开发.

SPARDL 描述的周期控制系统的模型是一个三元组：

Model: : = (Dictionary, Modules, ModeDiagram)

1) Dictionary 数据字典，定义了周期控制系统中使用的全局变量。每个变量包括变量名称、变量类型、变量单位、变量维度、变量默认值和变量物理含义。表 1 是数据字典的一个实例。表 1 中数据字典定义了 FailFlag、H_x 和 $\hat{q}_g$ 三个变量。第一个变量是标量变量，表示错误标记，无单位；第二个变量表示角速度，单位是 rad/s；第三个变量是一个长度为 4 的行向量，表示预估四元数。

表 1 数据字典示例

名称	类型	单位	维度	默认值	物理含义
FailFlag	int	Null	(0,0)	0	错误标志
H_x	Float	rad/s	(0,0)	0.0	MW 角速度
$\hat{q}_g$	Float	Null	(1,4)	(1.732,0.256,0141,0)	预估四元数

2) Modules 模块，定义了模式中会调用的各个控制模块。每个模块包括模块的名称、模块的输入变量集合和输出变量集合，以及一个用控制流表示的模式体。例 1 是模块的一个实例。

例 1 模块示例

模块名称：GDP
输入：$\Delta g_x, \Delta g_y, \Delta g_z, \hat{b}_x, \hat{b}_y, \hat{b}_z$
输出：$\hat{\omega}_x, \hat{\omega}_y, \hat{\omega}_z$
公式：

```
调用模块 FFT;
if( GKFlag = =0)
{GKFlag = 1;
}
```

$\hat{\omega}_x = \Delta g_x - \hat{b}_x * 60;$
$\hat{\omega}_y = \Delta g_y - \hat{b}_y * 60;$
$\hat{\omega}_z = \Delta g_z - \hat{b}_z * 60;$

GDP 模块由六个输入变量、三个输出变量和五条语句组成的模式体组成。

3) ModeDiagram 模式迁移图，阐明了组成周期控制系统的各个模式、模式的控制流，以及模式之间的迁移关系。每个模式包括模式名称、进入条件、模式控制流和模式转换。模式控制流描述了模式内的控制流程，包括模式初始化和过程调用。模式初始化用于完成模式开始时初始化行为。过程调用包含若干周期任务，每个周期任务描述了模式内部的计算任务以及完成计算任务的时间。模式转换描述了模式迁移关系，包括源模式、模式迁移的优先级、模式迁移条件和目标模式。一个源模式可以迁移到若干目标模式。模式迁移的优先级用于保障高优先级的目标模式得到优先迁移。例 2 是模式的一个实例。

例 2 模式示例

```
模式名称:SDM
进入条件: BZ1 = =1
初始化:
    调用模块 6;
调用过程:
    周期任务:16 ms
      调用模块 1;
      调用模块 4;
      调用模块 5;
    周期任务:32 ms
      调用模块 8;
    模式转换:
      优先级:1
      满足:t≥t_gi && t_gi >0
      目标:BZ1 =2;
      优先级:2
      满足:FailFlag = =1
      目标:BZ1 =8;
```

模式 SDM 在满足进入条件后调用模块 6 完成模式初始化，在第一个周期中调用模块 1、4、8 完成第一个周期任务，然后判断优先级为 1 的模式迁移条件是否成立。如果成立，模式迁移到模式 2；如果不成立，判断优先级为 2 的模式迁移条件是否成立。如果成立，模式迁移到模式 8；如果不成立，系统保持在模式 SDM 中，并开始下一个周期的运行。

2 问题描述

对周期控制系统的模型和程序一致性问题进行检查，可以检测程序的实现是否满足系统需求。目前大多数模型和程序一致性检测通过人工方式或者第三方工具进行检测，但这些方式都存在一定的不足，并且从哪些方面进行一致性检测也是一个值得探讨的问题。本文从周期控制系统的模式迁移和模块调用两个方面进行自动化的模型和程序一致性检测，最大限度地减少人力、资金以及时间上的开销。控制工程师在将周期控制系统实现为可执行程序后需要检测周期控制系统的实现是否满足需求定义。在这里周期控制系统的需求用 SPARDL 模型来描述。因此，检测周期控制系统的实现是否满足需求定义即是检测可执行程序是否满足 SPARDL 模型。为此，本文提出模型和程序一致性测试来解决这个问题。因为周期控制系统的功能性测试有很多，本文从周期控制系统的模式迁移和模块调用两个方面对模型和程序的一致性进行检测。图 1 描述了基于 SPARDL 的模型和程序一致性测试的框架。

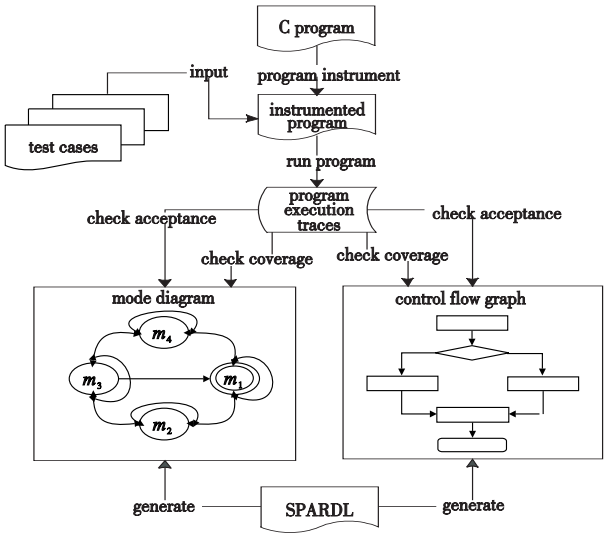

图1 基于SPARDL的模型和程序一致性测试的框架

本文随机生成若干测试用例，然后对程序进行插桩，通过运行随机生成的测试用例获取若干程序执行路径。按层次解析 SPARDL 描述的周期控制系统，分别获取模型的模式迁移图和控制流程图。将程序执行路径作为模式迁移图和控制流程图的输入，判断程序执行路径是否能被模式迁移图和控制流程图接收。如果能被接收，则模型和程序是一致的，再计算模式迁移图和控制流程图的顶点覆盖率以及分支覆盖率，作为生成测试用例的准则[14,15]。

3 一致性测试

3.1 程序执行路径

程序执行路径是一种用于记录程序执行信息的手段，它与要记录的信息有关，不同的信息描述不同的程序执行路径。例 3 给出了一个程序段，其中，f_1 f_2 f_3 f_4 是被 main 调用的函数。

以例 3 给出的程序段为例,关于函数调用的程序路径有 f_1、f_2、f_4 和 f_1、f_3、f_4。

例 3　程序段示例

```
int main(){
    int a = 0;
    f1();
    if(a>1){
        f2();
    }
    else{
        f3();
    }
    f4();
    return 1;
}
```

为了获取程序执行路径中的信息,需要对源程序进行相应的程序插桩。程序插桩是程序动态分析中的关键技术之一,目前已应用于程序覆盖分析、辅助测试用例设计、内存监控、提取不变式、程序调试、性能优化、软件安全性检查等多个方面。程序插桩的基本原理是在程序的特定位置插入采集信息的探针,通过运行插桩程序获得数据,帮助用户理解程序的行为和状态。为了获取模式迁移的程序路径,在为程序插桩时需要记录模式的名称;为了获取模块调用的程序路径,在为程序插桩时需要记录模块调用的名称。程序执行路径存储于队列中。在程序运行过程中,当遇到需要记录的信息时,首先为该信息创建一个节点,然后将该节点压入队列。对周期控制系统的源程序随机地生成若干测试用例,将这些测试用例作为周期控制系统插桩后程序的输入,运行插桩程序,分别获得一组关于模式迁移和模块调用的程序执行路径。考虑到周期控制系统的非终止性,在运行插桩程序的过程中需要限制程序执行路径的长度,使得程序执行路径是一个有限长度的序列。

3.2　模式迁移图和控制流程图

通过解析周期控制系统的设计文档获取系统的 SPARDL 模型表示,然后通过 SPARDL 语法解析器获取 SPARDL 模型的语法树,再解析语法树获取周期控制系统的模式迁移图和控制流程图。SPARDL 模型包含了丰富的信息,本文只关注模式迁移和模块调用的时序关系,因此,在实现 SPARDL 模型到模式迁移图和控制流程图的转换时只考虑周期控制系统的模式迁移和模块调用。模式迁移图是 SPARDL 模型的顶层表示,描述了模式及模式之间的迁移关系。模式迁移图可以用如下二元组形式化表示:ModeDiagram::=(Modes,Trans)。其中,Modes 是所有模式构成的集合,集合中的每个元素 $m \in$ Modes 表示一个模式,每个模式包括名称 modeName 和模式的进入条件 modeGuard。

Trans 是模式迁移关系的集合,集合中的每个元素(sourceMode,tranGuard,prio,targetMode) ∈ Trans 表示一个迁移关系。其中,sourceMode 是源模式,tranGuard 是模式迁移条件,prio 是模式迁移的优先级,targetMode 是模式迁移的目标模式。

模式迁移图是一个有向图,有向图的节点记录模式名称 modeName。有向图的边由如下两个规则定义:

a)对任意 $m \in$ Modes,存在一条模式 m 到 m 的有向边。

b)对任意 $m \in$ Modes,假定模式 m 中有 n 个模式迁移,记为 $Trans_m^n$。任意 $i \in [1,n]$,第 i 个模式迁移 $Trans_m^i$ 的目标模式记为 $targetMode_m^i$,存在一条模式 m 到模式 $targetMode_m^i$ 的有向边。

算法 1　SPARDL 模型到模式迁移图的转换算法

输入:模式集合 Modes。
输出:模式迁移图 ModeDiagram。

```
createModeDiagram:
begin
对每个模式 m∈Modes,作如下操作:
begin
    a)获取模式 m 的模式名称,记为 modeName_m;
    b)存在一条模式 m 指向模式 m 的有向边:
      modeName_m→modeName_m;
    c)获取模式 m 的 n 个模式迁移,记为 Trans_m^n;
    f)对每个 i∈[1,n],作如下操作:
begin
        (a)第 i 个模式迁移记为 Trans_m^i;
        (b)Trans_m^i 的目标模式记为 targetMode_m^i;
        (c)targetMode_m^i 的模式名称记为 targetModeName_m^i;
        (d)存在一条模式 m 指向模式 targetMode_m^i 的有向边:modeName_m→targetModeName_m^i;
end
end
end
```

控制流程图用于刻画模块调用的时序性。控制流程图由语句 stmts 构成。语句有基本语句 pStmt 和复合语句 cStmt 两种。因为这里只关注模块调用,所以基本语句中只包含模块调用语句。复合语句包括顺序语句、分支语句和循环语句。

控制流程图是一个有向图,有向图的节点用于记录模块调用语句,有向图的边由如下规则来定义:

假定模式 m 中包含 n 个周期任务,记为 $Tasks_m^n$。

a)任意 $i \in [1,n)$,存在一条 $Tasks_m^i$ 到 $Tasks_m^{i+1}$ 的有向边。

b)任意 $i \in [1,n)$,存在一条 $Tasks_m^i$ 到 $Tasks_m^1$ 的有向边。

c)每个周期任务的控制流程图由以下语句的类型来定义。

(a)基本语句 callmoduleName。创建一个节点记录模块名称 moduleName。

(b)顺序语句 stmt1、stmt2。存在一条 stmt1 指向 stmt2 的有向边。

(c)分支语句 if Bexpr then trueStmt else falseStmt。假定分支语句的前置语句为 preStmt,则存在一条 preStmt 指向 trueStmt 的有向边和一条 preStmt 指向 falseStmt 的有向边。若 preStmt 不存在,则分别创建 trueStmt 和 falseStmt 的控制流程图。

(d)循环语句 while Bexpr do stmts。存在一条 stmts 指向 stmts 的有向边。

算法 2　创建模式 m 的控制流程图的算法

输入:任意模式 m∈Modes。
输出:模式 m 的控制流程图 CFG。

```
modeToCFG:
begin
a)获取模式 m 中的 n 个周期任务,记为 Tasks_m^n;
b)对每个 i∈[1,n),创建周期任务 Tasks_m^i 的控制流程图;
c)对每个 i∈[1,n),存在一条 Tasks_m^i 到 Tasks_m^{i+1} 的有向边;
d)对每个 i∈[1,n),存在一条 Tasks_m^i 到 Tasks_m^1 的有向边;
end
```

3.3　接收检测

设 $G=\langle V,E\rangle$ 为一个有向图,V 是有向图的节点集合,E 是有向图的边集合;$Nodes=\langle node_1,node_2,\cdots,node_n\rangle$ 是一个长度为 n 的有限序列。有向图 G 能接收有限序列 Nodes 当且仅当在 G 中存在一条路径 $\langle v_1,v_2,\cdots,v_n\rangle$,使得对任意 $i \in [1,n]$,$v_i = node_i$。

模式迁移图和控制流程图都是有向图,程序执行路径是节点的有限序列,程序执行路径能被模式迁移图或控制流程图接收当且仅当程序执行路径对应的有限序列能被模式迁移图或者控制流程图对应的有向图接收。

算法 3　有向图接收有限序列的检测算法

输入:有向图 G,有限序列 Nodes。

输出:true,若有向图 G 能接收有限序列 Nodes;false,若有向图 G 不能接收有限序列 Nodes。

checkAcceptance:

begin

a)有限序列 Nodes 存储到队列 Queue;

b)获取队列的头节点,记为 $node_h$;

c)在有向图 G 中查找与 $node_h$ 对应的节点 v;

d)如果 v 不存在,则说明有向图 G 不能接收有限序列 Nodes,返回 False;

e)如果 v 存在,则

begin

(a)$node_h$ 从队列 queue 中出队;

(b)若队列 queue 为空,则说明有向图能接收有限序列 Nodes,返回 true;

(c)若队列 queue 不为空,则递归调用 checkAcceptance 检测有向图 G 能否接收有限序列 Nodes;

end

end

将模式迁移图 ModeDiagram 和模式迁移的程序执行路径 MadeTrace 作为接收检测算法的输入,检测模式迁移图能否接收模式迁移的程序执行路径。若模式迁移图能接收程序执行路径,则说明模型和程序在模式迁移的时序性上是一致的;若模式迁移不能接收程序执行路径,则说明模型和程序在模式迁移的时序性上是不一致的。导致这种不一致性有两个方面的原因:a)程序在实现周期控制系统的过程中出错;b)SPARDL 转换为模式迁移图的过程中出错。当出现不一致性时,需要根据需求来决定该模式迁移的时序性是否正确。如果模式迁移的时序性不正确,则将产生该程序执行路径的测试用例作为程序输入来检查程序在实现周期控制系统的过程中出现的错误;如果模式迁移的时序性是正确的,则修正模式迁移图。由于模块调用在具体实现过程中可能会引入辅助函数,所以模块调用的程序执行路径 ModuleTrace 可能包含设计文档中不曾定义的函数,因此在检测控制流程图 CFG 能否接收模块调用的程序路径之前需要对程序执行路径作预处理,将设计文档中不曾定义的函数隐藏,然后将控制流程图和程序执行路径作为接收检测算法的输入,检测控制流程图能否接收模块调用的程序执行路径。若控制流程图能接收程序执行路径,则说明模型和程序在模块调用的时序性上是一致的;若控制流程图不能接收程序执行路径,则说明模型和程序在模块调用的时序性上是不一致的,也需要从程序和控制流程图两个方面来检查不一致的原因。

3.4 覆盖检测

逻辑覆盖就是以程序内部的逻辑结构为基础,设计若干测试用例,使程序用这些测试用例运行时能覆盖程序中的每个语句、分支、路径或者子程序等。根据覆盖测试的目标不同,逻辑覆盖可分为语句覆盖、判定覆盖、判断—条件覆盖、条件组合覆盖、路径覆盖、子过程调用覆盖等。一般来说,语句覆盖是很弱的逻辑覆盖标准,判断覆盖比语句覆盖强,条件覆盖通常比判定覆盖强,判定—条件覆盖是判定覆盖和条件覆盖的综合,条件组合覆盖则比前面几种覆盖标准要强,但并不一定比路径覆盖强,路径覆盖是相当强的逻辑覆盖标准。路径覆盖要求程序的每条可能路径都至少执行一次,如果程序中有循环,则要求循环体至少经过一次。在工程实践中,由于程序量大,覆盖测试工具支持能力有限,往往只选择进行语句、判断、子过程调用覆盖测试。本文提出基于模式迁移图和控制流程图的顶点覆盖检测和边覆盖检测,用于指导测程序试用例的生成,使周期控制系统的模式和模块调用得到充分测试。

算法 4 有向图 G 中检测顶点覆盖和边覆盖的算法

输入:有向图 G,有限序列的集合 NodesSet。

输出:顶点覆盖率和边覆盖率。

checkCoverage:

begin

a)假设有限序列的集合 NodesSet 中包含 n 个有限序列,对每个有限序列 Nodes,作如下操作:

(a)调用 checkAccpetance 检测 Nodes 是否能被有向图 G 接收;

(b)如果不能被接收,则获取有限序列 Nodes 中的节点并加入节点集合 nodes 中;

(c)如果能被接收,则获取有限序列 Nodes 中的边并加入边集合 edges 中;

b)获取有向图 G 中所有节点的集合,记为 V;

c)获取有向图 G 中所有边的集合,记为 E;

d)顶点覆盖率 $\text{nodeCovRate} = \frac{\text{nodes}}{V}$;

e)边覆盖率 $\text{edgeCovRate} = \frac{\text{edges}}{E}$;

end

模式迁移图 ModeDiagram 和模式迁移的程序执行路径集合 ModeTrace 作为覆盖检测算法的输入,计算模式迁移图的顶点覆盖率和边覆盖率。如果模式迁移图的顶点覆盖率达到 100%,则说明用于产生模式迁移程序执行路径的测试用例是完备的,能够让周期控制系统的每个模式在程序运行过程中均被覆盖;如果顶点覆盖率没有达到 100%,则说明在周期控制系统中至少存在一个模式在程序运行过程中没有被覆盖,需要针对没有被覆盖的模式设计新的测试用例。如果模式迁移图的边覆盖率达到 100%,则说明周期控制系统中的模式迁移关系均被覆盖到,测试用例是完备的;如果边覆盖率没有达到 100%,则说明周期控制系统中至少还有一个模式迁移关系没有被测试到,还需要设计新的测试用例来覆盖没有被测试到的模式迁移关系。

同理,将控制流程图 CFG 和模块调用的程序执行路径集合 ModuleTrace 作为覆盖检测算法的输入,计算控制流程图的顶点覆盖率和边覆盖率。若覆盖率没有达到 100%,则需要对没有被覆盖的顶点和边设计新的测试用例,因此覆盖检测可以用于指导测试用例的生成。

4 应用实例

为了验证一致性测试方法在嵌入式周期控制系统中的有效性,本文对航天控制系统进行了一致性测试。图 2 给出了该周期控制系统的模式迁移图。

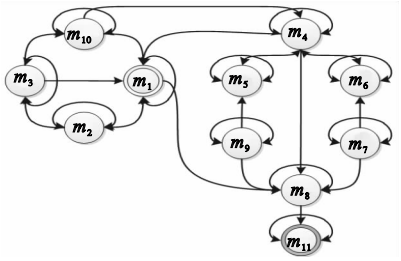

图2 周期控制系统的模式迁移图

该周期控制系统共有 11 个模式、其中 m_1 是初始模式,m_{11} 是结束模式,模式迁移关系如图 2 所示,每个模式都包含一条自回路。在周期控制系统源程序的特定位置插入采集模式名称的探针,随机生成测试用例,然后运行插桩程序获取程序执行路径。考虑到周期控制系统的非终止性,本文限定程序执行路径的最大程度为 200。将模式迁移图和程序执行路径作为接收检测算法的输入,判断程序执行路径能否被模式迁移图接收,最后统计模式迁移图的顶点覆盖率和边覆盖率。表 2 给出了模式迁移一致性测试数据。

模式迁移图包含 11 个顶点、35 条边。针对上述模式迁移

图设计了 1 350 个测试用例,对应 1 350 条程序执行路径均能被模式迁移图接收。航天控制系统经过长时间的运行维护,已经达到一个很稳定的状态,在模式迁移上表现出很强的一致性,因此所有的程序执行路径均能被模式迁移图接收。被覆盖的顶点有 9 个,有 2 个顶点未被覆盖,分别是 m_7 和 m_9,被覆盖的边有 25 条,有 6 条边未被覆盖,分别是 $m_7 \rightarrow m_7$,$m_7 \rightarrow m_8$,$m_7 \rightarrow m_6$,$m_9 \rightarrow m_9$,$m_9 \rightarrow m_8$,$m_9 \rightarrow m_5$。因为没有模式可以到达 m_7 和 m_9,因此需要设计新的测试用例来覆盖这两个模式以及与这两个模式相关的模式迁移。

表 2　模式迁移一致性测试

顶点数	边数	测试用例	接收检测	覆盖检测	
				顶点覆盖	边覆盖
11	31	1 350	1 350/1 350	9/11	25/31

周期控制系统中每个模式的控制流程图刻画了模块调用关系,以模式 m_4 为例,控制流程图如图 3 所示。

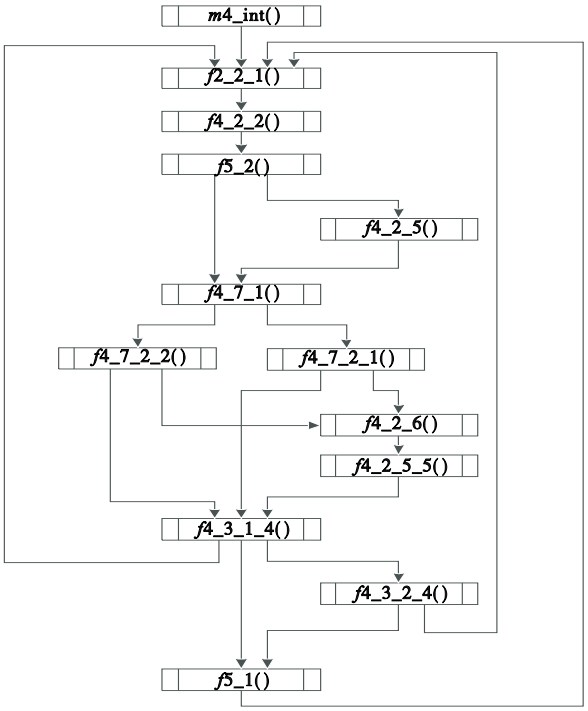

图3　模式m_4的控制流程图

同理,在周期控制系统源程序的特定位置插入采集模块调用名称的探针获取模块调用的程序执行路径,然后由接收检测算法检查程序执行路径能否被控制流程图接收,统计控制流程图的顶点覆盖率和边覆盖率。表 3 给出了模块调用一致性测试数据。

表 3　模块调用一致性测试

模式名称	顶点数	边数	测试用例	接收检测	覆盖检测	
					顶点覆盖	边覆盖
m_1	8	10	1 442	1 442/1 442	8/8	10/10
m_2	12	16	1 850	1 850/1 850	10/12	11/16
m_3	7	8	1 250	1 250/1 250	7/7	8/8
m_4	13	20	1 452	1 452/1 452	9/13	12/20
m_5	17	25	1 950	1 950/1 950	15/17	21/25
m_6	15	27	2 500	2 500/2 500	14/15	19/27
m_7	6	7	1 050	1 050/1 050	6/6	7/7
m_8	9	11	988	988/988	6/9	7/11
m_9	6	7	1 050	1 050/1 050	6/6	7/7
m_{10}	12	16	1 850	1 850/1 850	10/12	11/16
m_{11}	3	4	1350	1350/1350	3/3	4/4

表 3 描述了每个模式的控制流程图的顶点数、边数、针对该模式设计的测试用例数、能被控制流程图接收的程序执行路径数、控制流程图中被覆盖的顶点数和边数。以模式 m_4 为例,模式 m_4 的控制流程图包含 13 个顶点、20 条边,针对 m_4 随机生成 1 452 个测试用例,每个测试用例对应一条程序执行路径,1 452 条程序执行路径均能被控制流程图接收。由于系统非常稳定,模块调用表现出很强的一致性,因此所有的程序执行路径均能被控制流程图接收。控制流程图中有 9 个顶点被覆盖,4 个顶点未被覆盖,分别为模块 *f*4_2_5、*f*4_7_2_1、*f*4_2_6、*f*4_2_5_5。控制流程图中有 12 条边被覆盖,8 条边未被覆盖,分别为 *f*5_2→*f*4_2_5,*f*4_2_5→*f*4_7_1,*f*4_7_1→*f*4_7_2_1,*f*4_7_2_1→*f*4_2_6,*f*4_7_2_2→*f*4_2_6,*f*4_7_2_1→*f*4_3_1_4,*f*4_2_6→*f*4_2_5_5,*f*4_2_5_5→*f*4_3_1_4。因此,还需要针对控制流程图中尚未被覆盖的顶点和边设计测试用例。

5　结束语

本文将基于模型的测试与一致性测试相结合,提出了基于 SPARDL 的模型和程序一致性测试方法,实现了模式迁移和模块调用时序一致性的测试,并对测试用例的有效性进行了检测。本文将对周期控制系统中更多一致性需求进行测试,使得周期控制系统的可靠性得到保障。

参考文献:

[1] CALAME J R, IOUSTINOVA N, Van de POL J, *et al*. Data abstraction and constraint solving for conformance testing[C]//Proc of the 12th Asia Pacific Software Engineering Conference. 2005:541-548.

[2] BOZGA M, FERNANDEZ J C, GHIRVU L. Using static analysis to improve automatic test generation[C]//Proc of Conference on Tools and Algorithms for Construction and Analysis of Systems. 2000:235-250.

[3] TRETMANS G J. A formal approach to conformance testing[D]. Enschede, Netherlands: University of Twente, 1992.

[4] KRICHEN M, TRIPAKIS S. Black-box conformance testing for real-time systems[J]. Formal Methods in System Design, 2009, 34(3):238-304.

[5] GIANNAKOPOULOU D, PASAREANU C S, COBLEIGH J M. Assume-guarantee verification of source code with design-level assumptions[C]//Proc of International Conference on Software Engineering. 2004:211-220.

[6] De VRIES, RENÉ G, TRETMANS J. On-the-fly conformance testing using Spin[J]. International Journal on Software Tools for Technology Transfer, 2000, 2(4):382-393.

[7] FRANZ G, WOTAWA F, AMMANN P E. Testing with model checkers: a survey[J]. Software Testing, Verification & Reliability, 2009, 19(3):215-261.

[8] UTTING M, LEGEARD B. Practical model-based testing: a tools approach[M]. San Francisco: Morgan Kaufmann, 2007:1-433.

[9] 颜炯,王戟,陈火旺. 基于模型的软件测试综述[J]. 计算机科学, 2004, 31(2):184-187.

[10] CHEN Ming-song, QIU Xiao-kang, XU Wei, *et al*. UML activity diagram-based automatic test case generation for Java programs[J]. The Computer Journal, 2009, 52(5):545-556.

[11] LI Jian-wen, WANG Zheng, ZHAO Yong-xin, *et al*. An event-B interpretation for SPARDL model[C]//Proc of High-Assurance Systems Engineering. 2011:41-48.

[12] WANG Zheng, LI Jian-wen, ZHAO Yong-xin, *et al*. SPARDL: a requirement modeling language for periodic control system[C]//Proc of International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. 2010:594-608.

[13] 王政. 嵌入式周期控制系统的建模与分析[D]. 上海:华东师范大学, 2012.

[14] 王轶,蒋同海,董军,等. 基于路径覆盖插桩的可执行代码测试工具实现[J]. 计算机工程, 2012, 38(5):35-37.

[15] 张志华,牟永敏. 基于函数调用的路径覆盖生成技术研究[J]. 电子学报, 2010, 38(8):1808-1811.