基于接口自动机与符号执行的嵌入式软件测试用例生成

王 博[1),2)] 白晓颖[1),2)] 张 超[3)] 贺 飞[3)] SONG Xiao-Yu[4)]

[1)](清华大学计算机科学与技术系 北京 100084)

[2)](清华大学信息科学与技术国家实验室 北京 100084)

[3)](清华大学软件学院 北京 100084)

[4)](波特兰州立大学 Maseeh 电气与计算机工程学院 波特兰 97207 美国)

摘 要 随着嵌入式软件规模、复杂度的持续增长,基于构件的设计技术已在大规模嵌入式系统开发中得到广泛应用.嵌入式构件测试是保证构件质量以及构件间集成构造的重要手段.基于模型的测试是嵌入式软件测试的重要方法,通过基础模型描述系统预期的行为特性,以提供用例生成的基础.文中针对嵌入式软件构件,提出建立构件扩展语义接口自动机模型(Extended Semantic Interface Automata,ESIA),通过对接口自动机模型进行变量、约束条件等扩展,支持构件行为特性的描述与理解.以 ESIA 为基础模型,提出了基于符号执行的 ESIA-Symbolic 测试用例生成方法,通过搜索有效的事件/数据序列,设计相关测试用例与测试场景.实验以高速列车车载通信系统软件为例进行建模与测试生成,并在测试覆盖率、效率、有效性等方面与相关测试生成方法进行了对比.

关键词 扩展语义接口自动机;符号执行;测试用例生成

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2015.02125

Test Case Generation for Embedded Software Using Interface Automata and Symbolic Execution

WANG Bo[1),2)] BAI Xiao-Ying[1),2)] ZHANG Chao[3)] HE Fei[3)] SONG Xiao-Yu[4)]

[1)](*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)

[2)](*National Laboratory for Information Science and Technology, Tsinghua University, Beijing* 100084)

[3)](*School of Software, Tsinghua University, Beijing* 100084)

[4)](*Maseeh College of Electrical and Computer Engineering, Portland State University, Portland* 97207, *USA*)

Abstract With the increasing size and complexity of embedded systems, Component-Based Design (CBD) has been widely applied in large-scale embedded system development. Embedded component testing is important to ensure the quality of individual components and component integrations. Modeling is an effective technique in embedded system testing. It abstracts the expected system behavior and provides the basis for test case design. The paper proposes an ESIA (Extended Semantic Interface Automata) model which extends IA (Interface Automata) with variables and constraints for better understanding of component interface behavior. Based on ESIA, algorithms are designed following symbolic execution method to search for effective events/

收稿日期:2014-09-28;最终修改稿收到日期:2015-04-07.本课题得到国家自然科学基金(91218302,61472197)、国家"九七三"重点基础研究发展规划项目基金(2011CB302505)、国家"八六三"高技术研究发展计划项目基金(2013AA01A215)、北京市自然科学基金(4132062)资助.**王 博**,男,1979 年生,博士研究生,主要研究方向为软件测试、嵌入式系统.E-mail:harvicflyhigh@163.com.**白晓颖**,女,1973 年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为软件测试、服务计算.**张 超**,男,1991 年生,博士研究生,主要研究方向为软件测试和模型验证.**贺 飞**,男,1980 年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为形式化方法、自动机理论.**SONG Xiao-Yu**,男,1963 年生,博士,教授,主要研究领域为形式化方法、嵌入式计算系统.

data sequences as test cases and test scenarios. Experiments are exercised on a communication software of high-speed railway. It evaluates the test coverage and effectiveness of the proposed approach in comparison with others.

Keywords extended semantic interface automata; symbolic execution; test case generation

1 引 言

近年来,基于构件的设计(Component-Based Design,CBD)技术越来越广泛地应用于大规模、复杂嵌入式软件开发. CBD 将软件功能分解为高内聚、低耦合的功能模块,并集成构造嵌入式软件系统. CBD 可降低设计与开发的复杂度,提高开发效率,提高软件系统质量,改进系统可伸缩性.

构件可靠性是 CBD 系统可信性的基础,如何保证构件设计与实现的正确性成为重要的问题. 嵌入式软件构件测试是保证嵌入式软件质量的有效手段. 近年来,基于模型的测试(Model-Based Testing,MBT)技术在嵌入式软件测试中逐渐得到应用. 其中,常用的基础模型有扩展有限自动机、统一建模语言 UML、标号迁移系统等[1].

接口自动机(Interface Automat,IA)是一种轻量级的描述构件接口行为的状态机模型[2]. 构件接口是构件功能的外部表现,接口可屏蔽构件间差异,是嵌入式构件组装和重用的基础. IA 模型可描述构件的外部行为特性、接口交互行为、数据通信过程等,是嵌入式构件建模的有效工具.

本文对 IA 模型进行扩展,提出了 ESIA 模型,从以下 3 个方面提升模型的表达能力:

(1)增强了变量定义. IA 模型构建了基于接口事件的状态迁移序列,在 ESIA 模型中增加参数变量、内部变量的定义,以对行为参数、中间计算变量与计算结果等构件运行过程中的数据进行描述.

(2)增强了行为定义. IA 模型定义了输入行为、输出行为、内部行为,在 ESIA 模型中使用特定类型变量定义各类行为参数,以完整、准确描述构件行为与交互过程.

(3)增强了约束条件定义. IA 模型定义了构件的行为序列,在 ESIA 模型中增加变量取值约束定义,并通过各类运算符,将不同参数、中间计算变量的取值约束连接构成行为的前置、后置条件,以完整描述构件行为执行约束与构件运行规律.

本文以ESIA 为基础模型,提出了 ESIA-Symbolic 测试生成方法,支持正常、异常测试用例的自动生成,主要内容如下:

(1)制定模型覆盖准则. 针对 ESIA 模型,制定相应结构覆盖与数据覆盖准则,从模型结构元素覆盖、数据类型和等价类覆盖两个方面指导测试生成.

(2)测试路径生成. 在测试路径搜索过程中,引入符号执行(Symbolic Execution,SE)方法[3-4],生成满足模型覆盖准则的测试路径集合. 伴随测试路径的增长,实时分析行为参数取值约束的可满足性,分别记录生成的正常、异常测试路径. 针对嵌入式软件的特点,对与硬件紧密相关的参数或特殊环境参数进行插桩,采用确值/符号联合执行方法,进行确值/符号混合计算以及约束条件的混合求解.

(3)测试数据生成. 在每条正常测试路径之上,采用约束求解技术生成正常测试数据. 选取一个最小测试路径子集,并使用基于约束条件翻转、基于插桩、基于反例等方法生成异常测试数据.

本文以列车通信网络多功能车辆总线(Multi-function Vehicle Bus,MVB)网卡软件为例进行测试生成. 实验结果显示,通过 ESIA-Symbolic 方法的应用,可生成满足结构覆盖与数据覆盖准则的测试用例集合,包含正常、异常测试数据,可有效检测构件中存在的缺陷.

本文第 2 节介绍 ESIA 模型的相关内容;第 3 节介绍 ESIA 模型的结构与数据覆盖准则,叙述了基于符号执行的 ESIA-Symbolic 测试生成方法;第 4 节介绍内建 ESIA-Symbolic 方法的 ATES 测试平台,并依托该平台进行 MVB 网卡实时协议(Real Time Protocol,RTP)软件的建模与测试生成,并对实验结果进行分析与评估;第 5 节叙述相关领域的研究工作;第 6 节对全文进行总结.

2 扩展语义接口自动机

本节对 IA 模型进行变量扩展、行为扩展、约束条件扩展,给出 ESIA 模型定义,以其作为 ESIA-Symbolic 方法的基础模型.

2.1　语义扩展

2.1.1　变量扩展

嵌入式软件构件在运行过程中，维护各类内部变量、行为参数等. 在 ESIA 中进行变量扩展，分别定义行为参数集合 X^R、内部变量集合 X^H. 其中，行为参数表示各类行为的参数，而内部变量表示构件维护的中间计算变量与计算结果.

$\forall\chi\in X^R$ 或 $\forall\chi\in X^H$ 定义为一个四元组 $(type, \chi^{init}, D(\chi), Unit)$，$type$ 表示变量类型，χ^{init} 表示初始值，$D(\chi)$ 表示定义域，$Unit$ 表示度量单位. 其中，变量类型可以是整型、实型、布尔型等.

在 ESIA 中进一步定义状态变量 $\chi^V\in X^H$，以状态变量取值描述构件所处状态. χ^V 同样为一个四元组(整型，0，$0\leq\chi\leq N$，NULL)，其中状态变量无度量单位，NULL 表示度量单位未定义. χ^V 在定义域范围内的每项取值均代表 ESIA 模型的一个状态.

嵌入式软件常具有安全性设计策略，当环境条件异常时，构件执行将进入对应的异常状态，以对异常情况进行处理. 因此，在 ESIA 中将构件状态分为正常状态、异常状态，并对异常状态进行标记. 定义异常状态标识数组 $E\_tag[N]$，当 $E\_tag[i]$ 值为 1 时，表示 $\chi^V=i$ 对应的状态为异常状态，当 $E\_tag[i]$ 值为 0 时表示正常状态.

2.1.2　行为扩展

在嵌入式软件中，行为通常具有参数. 设 A^I、A^O、A^H 分别为构件的输入、输出及内部行为集合，在 ESIA 中进行行为扩展，分别定义各类行为的参数.

当 $a\in A^I$，定义 a 的输入参数集合为 $P_a=\{\chi_1^I,\cdots,\chi_n^I\}$，$\forall\chi_i^I\in P_a$，$\chi_i^I\in X^R$.

当 $a\in A^O$，定义 a 的输出参数集合为 $P_a=\{\chi_1^O,\cdots,\chi_m^O\}$，$\forall\chi_j^O\in P_a$，$\chi_j^O\in X^R\vee\chi_j^O\in X^H$.

当 $a\in A^H$，定义 a 的内部参数集合为 $P_a=\{\chi_1^H,\cdots,\chi_l^H\}$，$\forall\chi_k^H\in P_a$，$\chi_k^H\in X^H$.

构件运行通常需要一组中间计算变量的支持，以获取构件维护的数据或保存中间计算结果，通常以集合 X^H 中的内部变量表示.

在 ESIA 中，行为在特定状态下可被特定事件(Event)触发，由此定义 ESIA 的事件集合 E. 其中，输入行为需要外部事件触发执行，而内部行为、输出行为既可由事件触发执行，也可自发执行. 对于 $\forall e\in E$，$A[e]$ 表示由事件 e 触发执行的全部行为集合，$a\in A[e]$ 表示行为 a 由事件 e 触发.

2.1.3　约束条件扩展

约束条件描述了行为执行的限制条件. 当满足相关约束条件时，构件运行将表现出符合设计预期的行为特性. 其中，变量的取值约束、行为的前置与后置条件是重要的约束条件形式.

约束条件表达式由运算符连接不同的常量、变量等构成，为一阶逻辑表达式形式. 其中，常量与变量可为整型、实型、布尔型等类型，使用的运算符及其适用变量类型如表 1 所示.

表 1　约束条件运算符定义

符号	运算符名称	运算符类型	适用变量类型
+	加	算数运算符	整型、实型
−	减	算数运算符	整型、实型
*	乘	算数运算符	整型、实型
/	除	算数运算符	整型
=	赋值	算数运算符	整型、实型、布尔型
( )	优先级	算数运算符	整型、实型、布尔型
<	小于	关系运算符	整型、实型
≤	小于等于	关系运算符	整型、实型
==	等于	关系运算符	整型、实型、布尔型
!=	不等于	关系运算符	整型、实型、布尔型
∧	与	逻辑运算符	实型
∨	或	逻辑运算符	实型
¬	非	逻辑运算符	实型

本文定义以下 6 种基本约束表达式.

定义 1.　基本变量约束表达式. $\chi\in X$ 为 ESIA 中的一项变量，$v\in D(\chi)$ 为对应阈值，基本变量约束表达式定义如下：

(1) 小于最大值：$\chi<v$；

(2) 小于等于最大值：$\chi\leq v$；

(3) 大于最小值：$v<\chi$；

(4) 大于等于最小值：$v\leq\chi$；

(5) 等于确定值：$\chi==v$；

(6) 不等于确定值：$\chi!=v$；

以上 6 种基本约束表达式均适用于整型、实型变量，后两种还适用于布尔型变量. 此外，复杂约束可通过逻辑运算符连接基本约束表达式得到.

定义 2.　基本行为约束表达式. $a\in A$ 为 ESIA 中的一项行为，其约束表达式定义如下：

$$\psi(a)=(PreCon(a), PostCon(a));$$

其中，$PreCon(a)$ 为前置条件，描述一项行为可激活执行的前提条件；$PostCon(a)$ 为后置条件，描述一项行为执行效果是否符合预期的判断条件.

在 ESIA 中，行为前置、后置条件由状态变量、行为参数、内部变量取值约束等通过逻辑运算符连接构成，同样为一阶逻辑表达式. 其中，状态变量约束描述行为可被激活的状态以及迁移的目标状态；

行为参数、中间计算变量约束描述行为执行对相关数据的要求以及对执行结果的约束.通过在前置、后置条件中整合各类参数、变量取值约束,获得完整、简洁的行为约束描述.

在 ESIA 中,执行效果是对一组输入变量及中间变量进行特定计算,并根据计算结果更新一组输出变量或中间计算变量的值.行为 a 的执行效果表示为 $eff(a)$.一项执行效果包括更新变量与效果表达式两部分.其中,更新变量描述行为执行后、其值将被更新的变量;而效果表达式描述行为执行后、变量取值更新的方式.

如果后置条件中的一项变量为对应行为的一项更新变量,则可在后置条件中使用对应效果表达式替换该变量,从而将更新变量的取值约束转化为一组输入、中间变量的取值约束.

设 ESIA 的状态变量为 χ^V,输入行为 a 在状态 $(\chi^V==value_i)$ 下可被激活执行; P_a 为行为输入参数集合,$P_a\subseteq X^R$; X_1^H 为行为使用的内部变量集合,$X_1^H\subseteq X^H$; X^O 为行为输出参数集合,$X^O\subseteq X^R$; X_2^H 为行为更新的中间计算变量集合,$X_2^H\subseteq X^H$; 变量 χ 的取值约束记为 $constrain(\chi)$; $LF[X]$ 表示集合 X 中所有变量取值约束构成的约束条件表达式; 则 a 的前置条件、后置条件分别表示为

$$\begin{aligned}PreCon(a)&=LF(\chi^V)\wedge LF(P_a)\wedge LF(X_1^H)\\&=((\chi^V==value_i)\wedge(constrain(\chi_1^I)\wedge\cdots\wedge constrain(\chi_n^I))\wedge(constrain(\chi_1^H)\wedge\cdots\wedge constrain(\chi_l^H))),\ \chi^V\in X^H,\chi_j^I\in P_a,\chi_k^H\in X_1^H;\\PostCon(a)&=LF(\chi^V)\wedge LF(X^O)\wedge LF(X_2^H)\\&=((\chi^V==value_i)\wedge(constrain(\chi_1^O)\wedge\cdots\wedge constrain(\chi_m^O))\wedge(constrain(\chi_{l+1}^H)\wedge\cdots\wedge constrain(\chi_s^H))),\ \chi^V\in X^H,\chi_j^O\in X^O,\chi_k^H\in X_2^H.\end{aligned}$$

在 ESIA 中,一项行为可能具有不同的前置条件,描述了不同的参数取值范围,对应 ESIA 的不同执行分支.当行为被特定外部事件触发时,如果参数取值满足特定前置条件,ESIA 将进入对应的执行分支.因此,ESIA 的执行路径由外部事件序列与参数取值共同确定.

2.2 扩展语义接口自动机定义

扩展语义接口自动机 ESIA 模型定义如下.

定义 3. 扩展语义接口自动机.构件 P 的 ESIA 定义为一个七元组 $P=(V_P,V_P^{\text{init}},A_P,T_P,X_P,\Psi_P,E)$,其中:

V_P 是一个有限状态集合;

$V_P^{\text{init}}\subseteq V_P$ 表示初始状态集合,每一个自动机可包含一组初始状态,如果 $V_P^{\text{init}}=\varnothing$,则称 P 为空,如果 V_P^{init} 包含多项元素,表示 ESIA 具有多个初始状态;

A_P 表示行为集合,包含 A_P^I、A_P^O、A_P^H 三个互不相交的子集,分别表示输入行为、输出行为、内部行为集合;

T_P 表示状态间迁移关系集合,$T_P\subseteq V_P\times A_P\times V_P$,$\forall t\in T_P$ 表示一个状态迁移过程,具体描述如下:

$$t=(q)\xrightarrow[a]{PreCon(a),eff(a),PostCon(a)}(q');$$

当前置条件满足时,行为 a 被激活执行,引起状态迁移,并产生执行效果;当后置条件满足时,迁移过程完成;

X_P 表示变量集合,包含 X_P^R、X_P^H 两个互不相交的子集,分别表示行为参数集合、内部变量集合,状态变量同样为一项内部变量;

Ψ_P 表示行为约束集合,$\forall\psi(a)\in\Psi_P$ 定义为一个二元组,即 $\psi(a)=(PreCon(a),PostCon(a))$,其中 $PreCon(a)$ 表示行为 a 的前置条件,$PostCon(a)$ 表示后置条件;

E 是一个有限事件集合,触发特定行为执行.

以一个门禁软件构件为例,进一步说明 ESIA 模型相关定义.该构件验证人工输入的门禁编号与密码是否正确,如果正确则通过验证、门禁解锁,否则不予通过并返回错误信息.门禁构件的 ESIA 模型如图 1 所示.

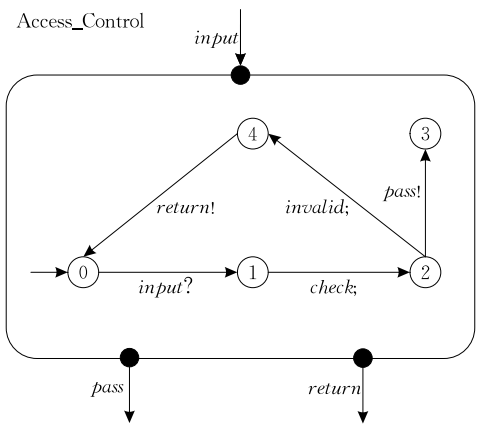


图 1 Access_Control 模型

模型中各项元素的定义如表 2 所示.

在 Access_Control 模型中,当人工输入的门禁编号或密码错误时,构件执行将进入异常状态“4”,并返回错误代码“90”; 当二者都正确时,构件执行进入状态“3”,并返回验证成功信息“0”.此外,人工输入的编号或密码应满足定义域要求,否则视为非法输入而不进行处理,输入行为 $input$ 不可执行.

表 2　Access_Control 模型元素

元素类型	Access_Control 模型定义
状态	状态“0、1、2、3、4”，其中状态 4 为异常状态；
初始状态	状态“0”；

行为	输入行为 *input*(*ID*, *password*)；
	内部行为 *check*(*ID_acc*, *password_acc*){
	if (*ID*==*ID_acc*)&&(*password*==*password_acc*)
	valid=TRUE; *error_code*=0; //效果*eff*(*check*)
	else
	valid=FALSE; *error_code*=90; //效果*eff*(*check*)}
	输出行为 *pass*(*error_code*)；
	内部行为 *invalid*( )；
	输出行为 *return*(*error_code*)；

变量与参数	状态变量 χ_{state}(整型, 0, $0\leq\chi_{state}\leq 4$, NULL)； 输入参数 *ID*(整型, 0, $0\leq ID\leq 65\,535$)； 输入参数 *password*(整型, 0, $0\leq password\leq 65\,535$)； 内部变量 *ID_acc*(整型, 0, $0\leq ID\_acc\leq 65\,535$)； 内部变量 *password_acc*(整型, 0, $0\leq password\_acc\leq 65\,535$)； 内部变量 *valid* (布尔型, FALSE, TRUE or FALSE)； 输出参数 *error_code*(整型, 0, $0\leq error\_code\leq 300$)；
行为约束	*PreCon*(*input*)=((0≤*ID*≤32 767)&&(0≤*password*≤65 535))； *PreCon*(*pass*)=(*valid*==TRUE)； *PreCon*(*invalid*)=(*valid*==FALSE)；
事件	*Invoke_input*：请求输入门禁编号与密码，触发 *input* 行为执行；

2.2.1　同步通信与异步通信

在嵌入式软件中，数据通信过程主要采用同步或异步模式. 在同步模式下，发送端在发送数据后阻塞运行并持续等待，直到收到接收端的响应后继续运行. 在异步模式下，发送端在发送数据后不阻塞运行，继续执行后续任务，而无需等待接收端的响应.

在 ESIA 中，对以上同步、异步通信过程采用统一的模型元素进行描述，但二者对应的行为序列不同. 在同步通信过程中，发送端输出行为 a^O 的后继行为中将包含一项输入行为 a^I，以接收对应的响应，当未收到接收端的响应时，将无法触发 a^I 执行，发送端在该行为处将阻塞运行；而在异步通信过程中，发送端无对应的响应接收行为，不阻塞运行. 由此，在 ESIA 模型之上，根据同步、异步通信过程的描述，采用统一方法生成相应测试用例.

2.2.2　构件间组合操作

在 ESIA 中，针对构件间同步或异步通信过程，利用共享行为描述通信过程中的数据发送与接收行为以及对应的响应发送与接收行为，其定义如下.

定义 4.　共享行为. 构件 P 和 Q 对应的 ESIA 分别为 A_P 和 A_Q，其共享行为集合定义如下：

$$shared(A_P,A_Q)=(A_P^I\cap A_Q^O)\cup(A_P^O\cap A_Q^I);$$

$$\forall a\in shared(A_P,A_Q),\ a=(a^O,a^I);$$

即共享行为 a 由构件间两项互为输入、输出行为的交互行为 a^O、a^I 构成. 在同步通信过程中，存在数据发送与接收、应答发送与确认两组共享行为；而在异步通信过程中，仅存在数据发送与接收一组共享行为.

嵌入式软件由不同构件组成，通过各构件间的协同运行，共同完成嵌入式软件各项功能，而可组合性描述了这种构件间协同运行的可能性. 其中，共享行为(a^O、a^I)的参数取值约束之间的协调性是构件间具有可组合性的重要因素. 本文从语法和语义层面给出 ESIA 间可组合性的定义，并以笛卡尔积运算为基础，定义 ESIA 间组合操作，支持嵌入式系统的组合构造.

定义 5.　ESIA 可组合性. 构件 P 和 Q 对应的 ESIA 分别为 A_P 和 A_Q，A_P 和 A_Q 之间是可组合的，当且仅当：

$$A_P^O\cap A_Q^O=A_P^H\cap A_Q=A_P\cap A_Q^H=\varnothing;$$

$$\forall a\in shared(A_P,A_Q),\ a=(a^O,a^I),$$

$$(a^I\in A_P^I\wedge a^O\in A_Q^O)\vee(a^O\in A_P^O\wedge a^I\in A_Q^I),$$

$$(P_{a^O}^O\subseteq P_{a^I}^I)\wedge(P_{a^O}^O\supseteq P_{a^I}^I),\ \text{and}\ \forall\chi_i\in P_{a^O}^O,$$

$$constrain_P(\chi_i)\wedge constrain_Q(\chi_i)!=\text{NULL};$$

其中，$P_{a^O}^O$ 为 a^O 的输出参数集合，$P_{a^I}^I$ 为 a^I 的输入参数集合.

由以上可组合性定义可知，A_P 与 A_Q 具有可组合性，首先要求 A_P 与 A_Q 之间除共享行为以及共同具有的输出行为之外，其他行为之间都是正交的；其次，在 A_P 与 A_Q 之间互为共享行为的输入行为与输出行为之间，输出参数与输入参数应匹配，且每项输出参数的取值约束与对应输入参数的取值约束之间应具有公共区间. ESIA 间的组合操作具体定义如下.

定义 6.　ESIA 组合乘积. 构件 P 和 Q 对应的 ESIA 分别为 A_P 和 A_Q，组合积运算 $A_P\otimes A_Q$ 定义如下：

$$V_{P\otimes Q}=V_P\times V_Q;$$

$$V_{P\otimes Q}^{\text{init}}=V_P^{\text{init}}\times V_Q^{\text{init}};$$

$$A_{P\otimes Q}^I=(A_P^I\cup A_Q^I)\backslash shared(A_P,A_Q);$$

$$A_{P\otimes Q}^O=(A_P^O\cup A_Q^O)\backslash shared(A_P,A_Q);$$

$$A_{P\otimes Q}^H=A_P^H\cup A_Q^H\cup shared(A_P,A_Q);$$

$$A_{P\otimes Q}=A_{P\otimes Q}^I\cup A_{P\otimes Q}^O\cup A_{P\otimes Q}^H;$$

$$\begin{aligned}T_{P\otimes Q}=&(\{((p,q),a,(p',q))\mid(p,a,p')\in T_P\wedge\\&a\notin shared(A_P,A_Q)\}\cup\\&\{((p,q),a,(p,q'))\mid(q,a,q')\in T_Q\wedge\\&a\notin shared(A_P,A_Q)\}\cup\\&\{((p,q),a,(p',q'))\mid(p,a,p')\in T_P\wedge\\&(q,a,q')\in T_Q\wedge a\in shared(A_P,A_Q)\});\end{aligned}$$

$X^R_{P\otimes Q}=(X^R_P\cup X^R_Q)\backslash\{x|a\in shared(A_P,A_Q)\wedge x\in P^O_{aO}\}$;

$X^H_{P\otimes Q}=(X^I_P\cup X^I_Q)\cup\{x|a\in shared(A_P,A_Q)\wedge x\in P^H_{aH}\}$;

$X^V_{P\otimes Q}=X^V_P\cup X^V_Q$;

$\forall a\in A^I_{P\otimes Q},\ \psi(a)=(PreCon(a),PostCon(a))$;

$\forall a\in A^O_{P\otimes Q},\ \psi(a)=(PreCon(a),PostCon(a))$;

$\forall a\in A^H_P\cup A^H_Q,\ \psi(a)=(PreCon(a),PostCon(a))$;

$\forall a\in shared(A_P,A_Q),\ a=(a^O,a^I)$,

$(a^I\in A^I_P\wedge a^O\in A^O_Q)\vee(a^O\in A^O_P\wedge a^I\in A^I_Q)$,

$\psi(a)=(PreCon(a^O)\wedge PreCon(a^I),$

$PostCon(a^O)\wedge PostCon(a^I))$;

$E_{P\otimes Q}=(E_P\cup E_Q)\backslash\{e|a\in shared(A_P,A_Q)\wedge a^I\in A[e]\}$.

由以上定义可见,在组合乘积运算中,组合 ESIA 的输入行为、输出行为集合应包括构件与环境之间的全部交互行为,而不包括构件间共享行为;互为共享行为的输出行为与输入行为合并为一项内部行为,其引起的迁移也被合并,共享行为对应的触发事件也转化为内部事件.其中,在同步模式下,将有数据发送与接收行为、响应发送与接收行为两组共享行为被合并;而在异步模式下,仅有数据发送与接收行为一组共享行为被合并.

3 基于符号执行的测试用例生成

嵌入式软件构件的执行路径由外部事件序列与参数取值共同确定.充分考虑路径之上各项约束条件,生成各类正常、异常测试数据,覆盖各类正常、异常执行路径,是构件测试的重要内容.

本节制定 ESIA 模型的结构覆盖与数据覆盖准则,在此基础上,提出 ESIA-Symbolic 方法,支持正常、异常测试用例的自动生成,其工作流程如图 2 所示.其中,各步骤分别以特定覆盖准则为指导、利用特定方法,配合生成完整的测试用例集.

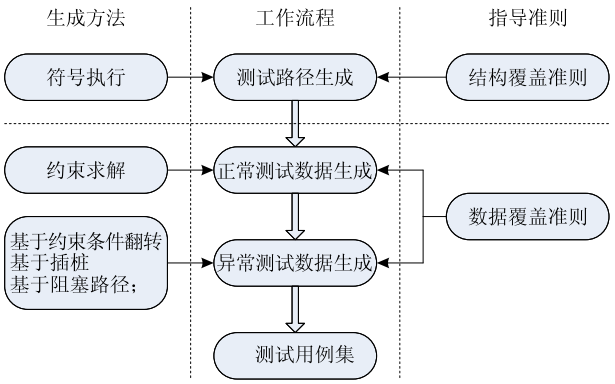


图 2 ESIA-Symbolic 方法工作流程

3.1 结构覆盖与数据覆盖准则

嵌入式软件构件在不同外部事件序列触发下,往往具有大量执行路径,难以进行完全测试.通常需要制定一组目标,当测试工作达到目标时,即认为测试工作已足够充分,以在保证测试充分性的前提下,控制测试用例集的规模.

MBT 中最常见的测试目标就是模型覆盖率准则,包括结构覆盖、数据覆盖、需求覆盖、基于缺陷、基于突变分析、随机准则等.结构覆盖关注底层模型的控制流,测试生成过程应充分覆盖模型结构元素;数据覆盖关注底层模型的数据流,测试生成过程应充分覆盖模型数据元素的取值范围、取值约束、等价类、边界值等.通过结构覆盖与数据覆盖准则,指导测试路径、数据生成过程,可对各模型元素进行充分地测试.

自动机模型常用的结构覆盖率准则有全状态覆盖、全迁移覆盖、全事件覆盖、深度-n 路径覆盖、长度-n 路径覆盖、全路径覆盖、MC/DC 覆盖等[1].全状态覆盖的要求太弱,而全路径覆盖、MC/DC 覆盖要求太强.通常全迁移覆盖/全事件覆盖被认为是最低要求.而采用深度-n 路径覆盖/长度-n 路径覆盖准则时,发现软件缺陷的概率将更高.

本文在全迁移覆盖/深度-n 路径覆盖准则基础上,制定 ESIA 模型之上的结构覆盖与数据覆盖准则,指导以 ESIA 模型为基础模型的 MBT 过程.其主要起到两方面作用:一是作为用例生成算法中的控制条件,指导测试用例生成;二是衡量测试用例集的质量,评估测试用例集是否满足充分性要求.

针对 ESIA 模型的结构覆盖准则如下:

(1)全迁移.测试用例集应覆盖 ESIA 中的所有迁移;

(2) All-Loop-n/All-m-Loop.在任意一条测试路径上,对 ESIA 中每一项循环结构的遍历不超过 n 次,且路径包含的循环结构数不超过 m;测试路径集合应覆盖 ESIA 中所有满足以上要求的路径.

针对 ESIA 模型的数据覆盖准则如下:

(1)数据类型覆盖.测试用例集应覆盖 ESIA 中所有数据类型;

(2)等价类覆盖.测试用例集应覆盖 ESIA 中行为参数的数据等价类.

3.2 测试路径与测试数据生成

SE 技术由 King[3] 和 Clarke[4] 提出,最初目标是找到一种介于测试与形式化验证之间的程序分析方法来验证程序正确性,该方法主要应用于对源代

码的测试. SE 技术使用符号输入代替实际数据输入，沿程序执行路径生成相应路径条件(Path Condition，PC)，并生成中间计算变量和执行结果的符号表达式. 其中，*PC* 描述引导程序沿指定路径运行的变量取值约束条件. 近年来，随着计算机处理能力的提升、约束求解技术、符号/确值联合执行方法的发展，SE 技术在软件测试中的应用又逐渐引起研究人员的关注[5-6].

SE 生成的测试路径具有可代换性. 对于一条路径，以一组确值作为输入执行该路径得到的结果，与在执行该路径生成的符号表达式中，将每个符号代入对应确值进行计算得到的结果相同. 可代换性是 SE 技术付诸应用的基础，也可用于用例执行结果的验证.

在 ESIA-Symbolic 方法中，将 SE 方法引入测试路径搜索过程. 在路径增长过程中，以 *PC* 的更新动态描述路径之上的变量取值约束变化情况，以 ESIA 结构覆盖准则为指导，生成一组测试路径以及对应的 *PC* 表达式，并得到输出参数的符号表达式. 随后对每一条测试路径对应的 *PC* 表达式进行约束求解，生成该路径之上的测试数据，控制被测软件沿预期路径执行.

在测试路径搜索过程中，依次抽取每项行为前置、后置条件中的参数、内部变量取值约束，并以“与”运算符连接，动态构建路径之上的 *PC* 表达式，其形式为一阶逻辑表达式. 其中，对于具有执行效果的行为，需要首先使用效果表达式替代其后置条件中的更新变量，再进行约束条件的抽取，以将后置条件中的输出参数取值约束转化为一组输入参数、内部变量取值约束. 因此，*PC* 表达式将包括测试路径上一系列输入、内部变量取值约束.

ESIA-Symbolic 方法中的 SE 语义描述如下：

(1) 建立符号与输入变量之间的映射关系，以符号代替数据确值代入模型；

(2) 在特定情况下，以一个常量代替数据确值，作为输入数据代入模型；

(3) 扩展基本运算符的适用范围，使其适用于符号对象，并通过计算产生符号表达式；

(4) 输出变量以符号或符号表达式描述；

(5) 在基于 SE 方法的测试路径搜索过程中建立 *PC* 表达式，并伴随测试路径搜索过程持续更新；

在测试路径生成过程中，从模型初始状态开始 SE 过程，并设置 *PC* 表达式初值为 TRUE. 当后继行为为 a 时，如果 a 具有输入参数 χ_1、χ_2，则创建符号 s_1、s_2，代替 χ_1、χ_2 作为输入. 如果 a 的前置条件中包含 χ_1、χ_2 的取值约束 $constrain(\chi_1)$ 和 $constrain(\chi_2)$，*PC* 表达式更新为 $PC = PC \wedge constrain(\langle s_1 \rangle) \wedge constrain(\langle s_2 \rangle)$. 如果 $PC \neq \varnothing$，则 a 可执行，SE 继续；如果 $PC == \varnothing$，表示变量不同的取值约束之间不一致，不存在公共区间，则 a 不可执行，SE 终止. 此外，如果 a 的后置条件中包含输出参数 χ_3 的取值约束 $constrain(\chi_3)$，且 a 具有效果“$\chi_3 = \chi_1 + \chi_2$”，则 *PC* 表达式更新为 $PC = PC \wedge constrain(\langle s_1 \rangle) \wedge constrain(\langle s_2 \rangle) \wedge constrain(\langle s_1 + s_2 \rangle)$，对 *PC* 表达式的判断过程同上.

当存在多项后继行为时，SE 将沿不同分支分别进行. 如果同时存在后继行为 b_1 和 b_2，对应输入参数取值约束构成的逻辑表达式分别为 pc_1 和 pc_2，当 $PC \wedge pc_1 != \varnothing$ 且 $PC \wedge pc_2 != \varnothing$ 时，b_1 和 b_2 均可执行，此时沿不同分支，*PC* 表达式分别更新为 $PC \wedge pc_1$ 和 $PC \wedge pc_2$，相应得到两个不同的 *PC* 表达式，SE 沿不同分支继续进行；当 $PC \wedge pc_1 != \varnothing$ 且 $PC \wedge pc_2 == \varnothing$ 时，b_1 可执行，b_2 不可执行，SE 沿 b_1 分支继续进行，而在 b_2 分支终止.

基于以上 SE 语义，提出 ESIA-Symbolic 方法的测试路径生成算法. 该算法是一种启发式的深度优先算法，以 *PC* 表达式作为启发条件，指导测试路径生成.

由于软件中存在循环结构，符号执行过程往往是无限的. 为了在有限时间内满足测试工作充分性要求，以全迁移覆盖、All-Loop-n/All-m-Loop 覆盖准则为指导，为算法制定合理终止条件，以生成相对充分、有限的测试路径集合. 从初始状态开始，在当前状态无后继行为或当前路径回到初始状态或对后继循环结构的遍历违反 All-Loop-n 或 All-m-Loop 准则时，测试路径生成过程终止并回溯.

由于 ESIA 包含正常状态与异常状态，算法在对模型元素进行遍历的过程中，将生成正常测试路径及异常测试路径.

ESIA 中的正常路径与异常路径定义如下：

(1) 正常路径. 当 ESIA 中的一条路径仅包含正常状态时，该路径为正常路径；

(2) 异常路径. 当 ESIA 中的一条路径包含异常状态时，该路径为异常路径.

算法还可实时检测模型中的执行路径是否可判定. 在测试路径生成过程中，如果出现 $PC == \varnothing$ 的情况，则意味着不存在一组数据，可使构件从初始状

态执行到当前状态,相关路径为部分可判定路径.对路径是否可判定的实时分析可辅助技术人员发现软件设计方案中的缺陷.同时,当 ESIA 模型规模持续增长且部分可判定路径的比例持续上升时,通过对路径是否可判定进行检测,在测试路径生成过程中对部分可判定路径进行剪枝,可避免生成无效的测试路径,提高路径生成效率.路径生成算法包含以下步骤:

(1)测试路径增长终止条件判断,如果满足终止条件,则终止路径增长,记录当前测试路径以及对应 PC 表达式并回溯;

(2)遍历当前状态的后继行为,以全迁移、All-Loop-n/All-m-Loop 准则为指导,遍历不满足覆盖准则要求的后继行为,并登记所遍历的循环结构;

(3)当前路径 PC 表达式判断,更新当前路径的 PC 表达式,进行约束求解,判断当前路径是否可判定,如果可判定,则更新当前状态并转至步(4);如果部分可判定,则记录当前测试路径以及对应的 PC 表达式并回溯;

(4)针对更新后的当前状态递归执行以上步骤.

算法具体定义如下.

算法 1. 测试路径生成.

```
输入: ESIA A
输出: A set of test paths
数据结构: visitedStates(VS); loopTransitions(LT);
MaxLoopCnt; MaxCntLoop; currentPath(CP); path-
Condition(PC); testPaths(TP); testConditions(TC);
invalidtestPaths(ITP); invalidtestConditions(ITC);

TPGen(A)
BEGIN
  Init VS, LT, MaxLoopCnt, MaxCntLoop, CP, PC,
       TP, TC, ITP, ITC
  SE(q∈V^init)
END

SE(q)
BEGIN
  Add q to VS
  IF (not exit t=(q,a,q') such that (t∈T and t∉LT))
    or (q'∈V^init and length of current path≠0)
    or (t∈T and t∈LT
      and Traversal times of t<MaxLoopCnt
      and Loop number of current path<MaxCntLoop)
      //测试路径增长终止条件判断
    Add CP to TP //可判定路径登记
    Add PC to TC
    Add PRC to TRC
  ELSE
    FOR each t=(q,a,q') such that (t∈T and t∉LT)
      or (t∈T and t∈LT
      and Traversal times of t in LT<MaxLoopCnt
      and Loop number of current path<MaxCntLoop)
      //遍历当前状态的后继行为
    IF (q'∈VS and t not exit in CP
        and Loopnumber of current path<MaxCntLoop)
      or (t∈LT
        and Traversal times of t in LT<MaxLoopCnt
        and Loop number of current path<MaxCntLoop)
      Add t to LT
      refresh Traversal times of t
      Refresh Loop number of current path
      Add t to temp_loop //循环结构登记
      temp_q=q
      temp_t=t
      WHILE(Out Branch Number of temp_q=1
        and temp_q∉V^init) //回溯登记至最近分支节点
        temp_q=Previous State of temp_q
        temp_t=Previous Transition of temp_t
        Add temp_t to LT
        Add t to temp_loop
        Add Precodition(a)/Postcondition(a) to PC
      IF(PC!=∅) //PC 表达式判断
        Add t to CP
        SE(q') //递归执行过程
      ELSE
        Add t to CP //部分可判定路径登记
        Add CP to ITP
        Add PC to ITC
        Add PRC to ITRC
      Remove Precodition(a)/Postcondition(a) from PC
      FOR each t in temp_loop
        Remove t from LT
        Refresh Traversal times of t
        Refresh Loop num of current path
      Remove t from CP
      Remove q from VS
END
```

在以上算法中,通过约束求解判断 PC 是否为 $\varnothing$.目前常见的约束求解工具有 Choco Solver[7]、MINION[8]等,本文利用 Choco Solver 约束求解器对 PC 进行约束求解.当解存在时,判定 $PC!=\varnothing$;当解不存在时,判定 $PC==\varnothing$.

测试数据生成过程分为正常测试数据生成与异常测试数据生成,后者在后续章节中介绍.在正常测试数据生成过程中,针对每一条正常路径,将对应 PC 表达式代入约束求解器,生成测试数据,驱动测试用例执行.在测试数据求解过程中,同样利用 Choco

Solver 约束求解器，创建 CPModel 约束模型，制定求解策略，生成满足特定约束条件的测试数据.

3.3　异常测试用例生成

嵌入式软件构件应对运行过程中出现的异常情况进行有效处理，避免产生非预期的结果，异常测试即对构件相关特性进行的测试. 在 ESIA-Symbolic 方法中，采用基于约束条件翻转、基于插桩、基于反例等方法生成异常测试数据.

3.3.1　基于约束条件翻转的异常测试

在 ESIA 中，一项行为可能正常执行或异常执行，触发 ESIA 中不同的迁移关系，导致构件运行进入正常状态或特定异常状态.

行为正常执行与异常执行时对应的前置条件不同. 如果一项或多项输入参数、变量的取值不满足正常执行的前置条件，将导致 ESIA 执行进入异常状态. 因此，通过对此类参数或变量的取值约束在定义域范围内求补(即进行翻转)，可获得导致行为异常执行的前置条件.

在进行异常测试时，如果同时翻转多项参数、变量的取值约束，虽然可生成异常测试数据，但无法具体体现某一项参数取值异常时对构件执行的影响. 因此，在 ESIA-Symbolic 方法中，在异常测试路径之上，针对具有异常执行分支的行为，依次翻转每一项独立影响行为前置条件表达式取值的输入参数或变量的取值约束.

在异常测试生成过程中，在翻转一项取值约束后，将生成的前置条件与对应行为所有正常执行分支的前置条件进行实时比对. 如果生成的前置条件与某项正常执行分支的前置条件相同，则剔除该生成结果，并依次翻转下一项约束条件；如果与所有正常执行分支的前置条件都不同，则该前置条件即为一项异常执行前置条件. 进而沿异常测试路径抽取相应参数、内部变量取值约束构建对应 PC 表达式，进行约束求解以求取异常测试数据.

不同运算符构成的约束表达式的翻转关系见表 3.

表 3　约束条件翻转关系

运算符	约束表达式	翻转后约束表达式
$<$	$\chi<v$	$v\leq\chi$
$\leq$	$\chi\leq v$	$v<\chi$
$==$	$\chi==v$	$\chi!=v$
$!=$	$\chi!=v$	$\chi==v$

如行为 a 的输入参数为 χ_1、χ_2，PC 表达式为 $PC=PC\wedge constrain(\langle s_1\rangle)\wedge constrain(\langle s_2\rangle)$，且 $PC!=\varnothing$. 通过翻转输入参数 χ_2 的取值约束，更新 PC 表达式为 $PC=PC\wedge constrain(\langle s_1\rangle)\wedge(\neg constrain(\langle s_2\rangle))$，重新代入约束求解器可生成相应异常测试数据.

由于在每条异常路径上都进行基于约束条件翻转的异常测试代价较高，本文在生成的异常路径集合中选取一个满足全迁移覆盖准则的最小异常路径子集，在该子集之上进行基于约束条件翻转的异常测试. 该子集中任意两条测试路径所包含的状态迁移集合不相同；包含相同状态迁移集合、但迁移序列不同的两条路径同样不允许同时出现在最小子集之中. 由此保证异常测试覆盖 ESIA 中所有的异常执行行为，在保证异常测试有效性的前提下提升工作效率.

ESIA-Symbolic 方法中基于约束条件翻转的异常测试用例生成方法定义如下.

算法 2.　基于约束条件翻转的异常测试用例生成.

```
输入：A set of test paths
输出：A set of exceptional test cases
数据结构：visitedTransitions(VT)；selectedPath(SP)；
allTransitons(AT)；exceptionalPath(EP)；exceptional-
Condition(EC)；exceptionaltestPaths(ETP)；

ECGen(TestPaths)
BEGIN
  Init VT,SP,AT
  WHILE(VT≠AT)
    FOR each p in TestPaths and each t on p
      IF (t∉VT)
        Add t to VT
        Add p to SP
    TCG(SP)
END

TCG(SP)
BEGIN
  FOR each p in SP
    Clear EP,EC
    FOR each t on p
      Add t to EP
      Add constraints of t to EC
      IF successive state of t is exceptional state
        FOR each independent c in constraints on t
          Convert c //翻转约束条件
          Solve EC
          IF (Solution existed)
            Add EP to ETP
            Generate Exceptional Test Data
          Recover c //恢复约束条件
END
```

3.3.2 基于插桩的异常测试

在测试用例设计过程中,对于与硬件紧密结合的参数或者由于测试环境条件限制无法完全控制其取值的参数,如通信端地址等,由于生成的测试数据通常不具有实际意义,故多采用插桩方法,在建模与测试用例生成过程中为其指定一组符合物理意义的特定值,以保证测试用例在测试环境中的适用性.

在 ESIA-Symbolic 方法中,同样采用插桩方法,为特定输入参数人工指定一组正常或异常值.由此,SE 的输入将由一组确值以及一组符号混合构成,此时纯粹的符号执行过程演化为一种确值/符号联合执行过程.如行为 a 具有输入参数 χ,当以常量 *Constant* 为 χ 赋值时,PC 表达式更新为 $PC=PC\wedge(\chi==Constant)$.

本文采用在 EXE[9] 与 KLEE[10] 等工具中应用的 EGT(Execution-Generated Testing)确值/符号联合执行方法[11],动态检查每项输入参数是否是确值,如果是则将相应确值代入程序执行,并在后续执行过程中持续使用该确值参与各类相关计算.由此,在 ESIA-Symbolic 方法中建立确值/符号联合执行机制,实现插桩数据与符号之间的混合运算,实现 PC 表达式、输出表达式的混合构建,支持对测试数据的插桩.

在 ESIA-Symbolic 方法中,在建立 ESIA 模型的同时,对于需要插桩的输入参数,在模型中为其指定特定确值.在生成测试路径时,对于包含插桩参数的前置、后置条件、效果表达式,均使用特定确值代替原参数参与各类运算.如 a 的输入参数为 χ_1、χ_2,效果表达式为 $\chi_3=\chi_1+\chi_2$.如果为输入参数 χ_2 设定确值 v_2,则效果表达式更新为 $\chi_3=\chi_1+v_2$.此时,沿特定路径生成的 PC 表达式中将不再包含插桩参数的取值约束.通过约束求解生成的测试数据与插桩值共同构成一条测试路径上的完整测试数据.

在基于插桩的异常测试生成过程中,在模型中为特定参数指定一组异常值,并通过确值/符号联合执行生成异常测试用例.

基于插桩的异常测试用例生成同样需要在一个最小测试路径子集上展开,以控制用例集的规模.在子集包含的每条路径之上,针对每一项具有异常插桩值的参数分别生成异常测试用例.对于具有多项异常值的参数,将针对每项异常值分别生成测试用例.

此外,当模型复杂度较高时,通过插桩以及确值/符号联合执行可解决测试数据生成过程中的复杂约束求解问题,提高测试数据生成效率.

3.3.3 基于反例的异常测试

前述测试路径生成算法可检测出构件 ESIA 模型的部分可判定路径,而这类路径常常隐含着构件设计方案中的缺陷,一条此类路径即为 ESIA 模型执行路径的一项反例.因此,根据反例有针对性地生成相应异常测试用例,测试构件的实现是否存在缺陷.

在部分可判定路径之上,提取从初始状态到异常状态的 PC 并求解,生成异常测试数据,引导构件从初始状态运行至异常状态,从而测试构件的异常处理机制是否符合预期.

4 实验与评估

4.1 ATES 测试平台

本文在 Eclipse 开放平台上设计开发 ATES(Automated Test Platform for Embeded Software)自动测试平台,内建 ESIA-Symbolic 方法,平台架构如图 3 所示.

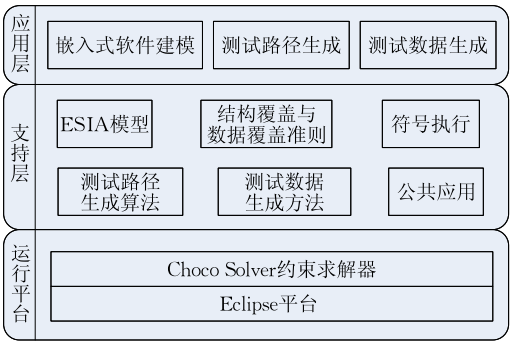


图 3 ATES 平台架构

ATES 平台具有以下功能:

(1)基于 ESIA 模型的嵌入式软件构件建模;

(2)测试路径生成,路径条件提取与解析;

(3)正常测试数据生成;

(4)基于约束条件翻转、基于插桩、基于反例的异常测试数据生成.

在实验过程中,利用 ATES 平台提供的基础类,人工编写被测软件构件的 ESIA 模型代码.随后在 ATES 平台上,以 ESIA 模型为基础模型,调用平台内建的测试路径生成算法生成测试路径集合,并生成相应正常测试数据;进一步调用基于约束条件翻转、基于插桩、基于反例等方法生成各类

异常测试数据.

4.2　MVB 网卡 RTP 软件建模

在国际电工协会(International Electrotechnical Commission,IEC)制定并颁布的列车通信网络标准 IEC61375-1(Electric railway equipment-Train bus)[12]中,现代列车通信网络(Train Communication Network,TCN)主要由列车总线、车辆总线以及总线上各设备节点构成,是典型的大规模、复杂嵌入式系统.典型的列车总线如绞式列车总线(Wire Train Bus,WTB),典型的车辆总线如多功能车辆总线即 MVB,由 WTB 和 MVB 共同构成两级 TCN,负责完成设备运行状态、控制指令在列车平台上的传输.

IEC61375-1 标准制定了列车、车辆总线实时通信协议,即 RTP 协议,所有列车通信网络设备的开发、测试均需遵循该协议.MVB 网卡是 TCN 中的关键设备,其同样遵循 RTP 协议,负责将列车上各传感器、设备挂接在 MVB 上,支持通过 MVB 进行设备间信息交换.MVB 网卡 RTP 软件是典型的嵌入式软件,其实现了 RTP 协议,并与底层硬件、嵌入式操作系统共同构成嵌入式构件,具有设置与读取过程数据、发送与接收消息数据等功能.RTP 软件向用户提供系列 API 函数,为车载设备间的数据发送与接收任务提供支持.

ATES 平台提供建立构件 ESIA 模型的基础类.测试人员在对被测软件进行充分功能需求分析的基础上,使用相关类编写被测软件的 ESIA 模型代码.在 RTP 软件建模过程中,以 IEC61375-1 标准为依据,进行测试需求分析,针对 RTP 协议描述的过程数据、消息数据通信过程,提取相关功能项,分别建立数据通信发送端和接收端模型.

在 TCN 中,发送端 MVB 网卡与接收端 MVB 网卡之间的通信过程如图 4 所示.

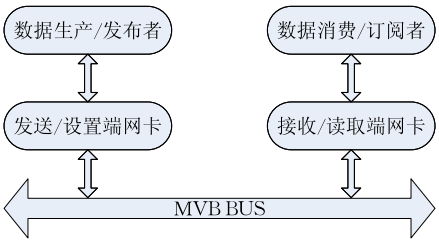


图 4　MVB 总线消息数据通信过程

在消息数据通信过程中,发送端负责读取数据生产者产生的数据,并通过 MVB 总线将数据向接收端发送;接收端负责从 MVB 总线接收数据,并将数据转发到数据消费者.在过程数据通信过程中,发送端负责读取数据源设备发布的数据,并在 MVB 总线的特定端口上设置数据;接收端负责从相应总线端口读取数据,并将数据转发到订阅设备.

针对发送端和接收端分别建立 ESIA 模型,得到 MVB_Caller、MVB_Respondent,如图 5、图 6 所示.

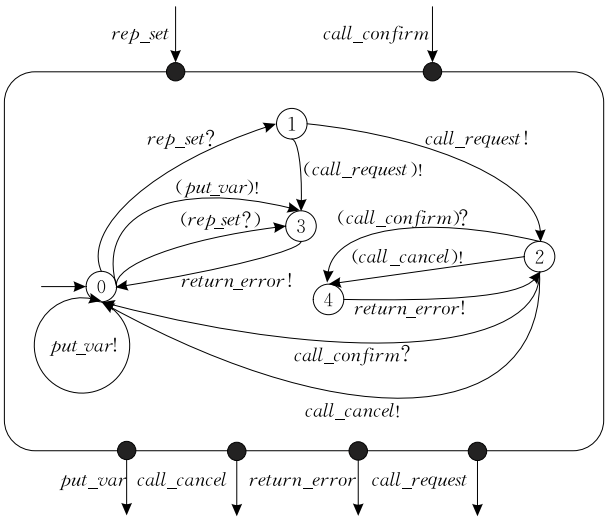


图 5　MVB_Caller 模型

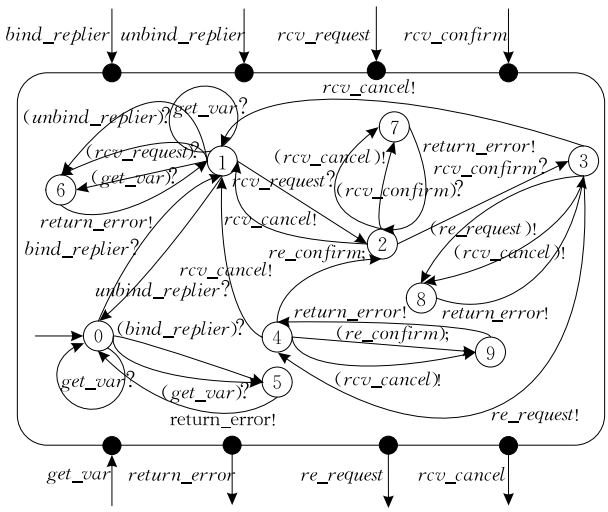


图 6　MVB_Respondent 模型

其中,消息数据发送与接收过程采用同步通信模式,即 MVB_Caller 发送消息数据后将阻塞运行,直到收到 MVB_Respondent 的响应后方继续运行.(*call_request*,*rcv_confirm*)、(*re_request*,*call_confirm*)为消息数据通信过程对应的两组共享行为,分别描述消息数据发送与接收过程、应答消息发送与确认过程.过程数据设置与读取过程采用异步通信模式,即 MVB_Caller 设置过程数据后继续运行,MVB_Respondent 自行读取相关过程数据.(*put_var*,*get_var*)为过程数据通信过程对应的一组共享行为,描述过程数据设置与读取过程.此外在

模型中,(*action*)表示行为 *action* 的异常执行分支,触发 ESIA 迁移至相应异常状态.

使用 ESIA 对嵌入式软件构件进行建模时,还需要根据目标系统、测试环境的特点,对不同类型的行为参数进行分类描述、处理,以建立完整的行为描述,并保证所生成测试数据的可操作性. RTP 软件是深嵌入式软件,其行为具有的参数类型以及对应变量类型如表 4 所示.

表 4 应用参数类型

参数类型	变量类型	说明
寻址地址	整型、指针等	通信呼叫者、应答者寻址
数据地址	整型、指针等	指向待处理的数据对象
数据变量	整型、实型、布尔型等	保存待处理的数据对象
回调函数入口	字符串、指针等	指向由本行为触发执行的后续行为
参考句柄	整型、字符串、指针等	甄别系列行为之间的配套关系
执行状态	枚举、整型等	描述行为执行状态
时间	整型、实型等	指定行为执行延时等性能特性

由表 4 可知,RTP 软件通过指针操作消息数据对象. 在 ESIA 模型中,将指针类型映射为布尔类型,当值为 TRUE 时,表示有效消息数据地址;当值为 FALSE 时,表示无效地址. 进一步将布尔类型变量映射为整型变量,其中 0 表示取值为 FALSE,1 表示 TRUE. 对以指针变量表示的通信端地址、回调函数入口地址、参考句柄等作同样处理. 此外,整型变量表示的通信端地址仍以整型变量描述.

过程数据包含整型、实型等数据类型,具有不同范围的定义域. 为简化讨论,选取布尔型、16 位整型、实型等类型进行建模和测试用例生成. 每项过程变量具有对应的检查变量,为一个 2 位变量,同样将其映射为整型变量,当值为 0 时,表示对应过程数据错误;当值为 1 时,表示正确;当值为 2 时,表示强制值;当值为 3 时,表示无定义.

嵌入式软件通常面向特定应用领域,与硬件平台紧密结合,测试环境建立困难. RTP 软件为清华大学研制,以实现高速列车核心装备国产化,其开发环境和测试环境均自主可控. 在研制工作中,通过应用 ESIA-Symbolic 方法对 RTP 软件进行基于 ESIA 模型的正常、异常测试,发现了一系列软件缺陷,有效保证了 RTP 软件设计与实现的正确性,保证了 RTP 软件产品的质量.

MVB_Caller 和 MVB_Respondent 是 RTP 软件最重要的核心底层模块,在嵌入式软件领域具有代表性. 因此,本文以 RTP 软件为例,对 ESIA 模型扩展与 ESIA-Symbolic 方法进行验证与评估.

4.3 用例生成结果

本节采用 ATES 平台内置的 ESIA-Symbolic 方法,在 MVB_Caller、MVB_Respondent 模型之上生成正常、异常测试用例集,并从用例集规模、模型覆盖率、执行效率、工作效率、缺陷检测等方面将 ESIA-Symbolic 方法与人工设计方法、Zhang 等人[13] 提出的 EIA-SATG 算法、随机生成算法进行综合比较.

EIA-SATG 算法是一种基于扩展接口自动机(Extended Interface Automata,EIA)的深度优先图遍历算法. 在 EIA 中,定义了构件变量集合,并定义了行为前置、后置条件. 以 EIA 为基础模型,EIA-SATG 算法将测试路径生成问题转化为图搜索问题. 在测试路径生成过程中,要求 EIA-SATG 算法使用与 ESIA-Symbolic 方法相同的结构覆盖率准则,即以全迁移覆盖、All-Loop-n/All-m-Loop 覆盖准则为指导,以便与 ESIA-Symbolic 方法生成结果进行比较. 在生成的每条测试路径之上,EIA-SATG 算法通过抽取行为前置、后置条件并进行约束求解,生成对应路径的测试数据.

EIA-SATG 算法使用的基础模型 EIA 无状态变量、异常状态定义,在测试路径生成过程中无法识别异常路径,因此 EIA-SATG 算法仅能生成正常测试路径与测试数据. 同时,EIA-SATG 算法不具有启发条件,在测试路径集合生成之后才进行测试数据求解. 而 ESIA-Symbolic 方法是一系列正常、异常测试用例生成方法与算法的集合. 通过采用 SE 技术,引入启发条件指导测试路径的生成,甄别各类正常、异常以及部分可判定路径,在此基础上通过各类方法求解正常、异常测试数据. 在实验过程中,由于 EIA-SATG 算法使用的 EIA 模型与 ESIA 模型不同,故参照 MVB_Caller、MVB_Respondent 模型语义,将其分别转化为对应的 EIA 模型,再进行测试生成. 在模型转化过程中,ESIA 模型的状态、行为、迁移关系集合分别转化为 EIA 模型的状态、行为、迁移关系集合,而由于 EIA 模型无异常状态定义,ESIA 模型的正常、异常状态均转化为 EIA 模型中的一般状态. 如 MVB_Caller 模型中的状态“3”、“4”,在 ESIA 模型中为异常状态,转化为 EIA 模型后即为一般状态.

随机生成算法则直接使用构件的 ESIA 模型作为基础模型进行测试生成,与 ESIA-Symbolic 方法

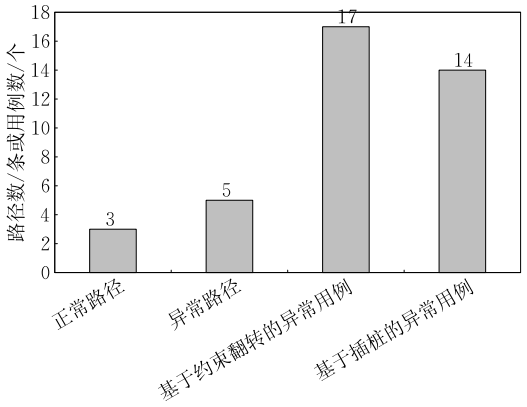
不同的是，在测试路径生成过程中，后继行为的选择是随机的.

由于 ESIA-Symbolic 方法与其他方法或算法的工作目标不完全相同，故在实验过程中，仅对相关方面的实验结果进行比较.

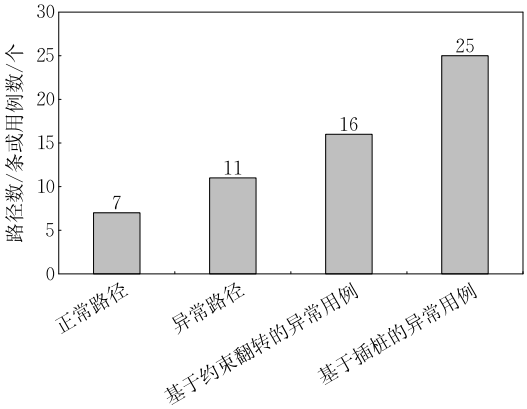
4.3.1　用例集规模

ESIA-Symbolic 方法在 MVB_Caller、MVB_Respondent 之上生成的测试路径集、测试用例集规模如图 7 所示. 由于 ESIA-Symbolic 方法在每条正常测试路径之上均求解一组对应的正常测试数据，因此正常测试路径数与正常测试用例数相同.

在基于约束条件翻转、基于插桩的异常测试数据生成过程中，需要针对不同类型的行为参数生成测试数据，生成结果需要覆盖对应等价类. MVB_Caller、MVB_Respondent 中不同参数类型对应的变量类型、等价类和测试数据生成方式如表 5 所示.



(a) MVB_Caller生成结果



(b) MVB_Respondent生成结果

图 7　生成测试路径与测试用例数

表 5　测试数据生成方式分析表

生成方式	参数类型	变量类型	等价类	测试数据取值	适用参数
约束求解	寻址地址	整型	约束条件范围内/外取值	正确值/超限值	拓扑计数器
	数据地址	指针	指定地址/null/未指定	1/0/未赋值	消息缓存/消息对象
	数据变量	整型	约束条件范围内/外取值	正确值/超限值	消息长度
			定义域内取值/未指定	正确值/未赋值	过程数据/检查变量
		实型	定义域内取值/未指定	正确值/未赋值	过程数据
		布尔型	定义域内取值/未指定	正确值/未赋值	过程数据
	回调函数入口	指针	指定地址/null/未指定	1/0/未赋值	回调函数入口
	参考句柄	指针	指定值/null/未指定	1/0/未赋值	行为间配套关系
插桩	寻址地址	整型	指定地址/错误地址/未指定	正确值/错误值/未赋值	功能标识/节点标识/站标识/通信存储器/端口标识
	数据变量	整型	0/小于真实消息长度/大于真实消息长度/未指定	0/过小值/过大值/未赋值	消息长度

由生成结果可见，ESIA-Symbolic 方法是一套系统方法，既可生成正常测试数据，也可生成异常测试数据. ESIA-Symbolic 方法实现了用例设计过程自动化，依据统一原则，采用通用方法生成用例.

4.3.2　模型覆盖率

ESIA-Symbolic 方法在 MVB_Caller、MVB_Respondent 之上分别生成 8 条与 18 条测试路径，对全迁移、All-Loop-n/All-m-Loop 准则的覆盖率均达到 100%. 相比之下，随机生成方法在 MVB_Caller、MVB_Respondent 之上生成的测试用例集合对全迁移、All-Loop-n/All-m-Loop 准则的覆盖率分别如图 8、图 9 所示.

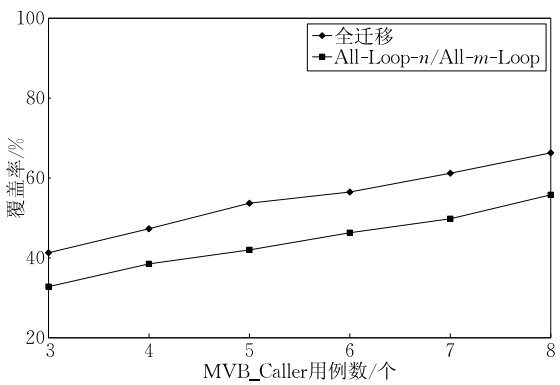


图 8　MVB_Caller 随机方法覆盖率

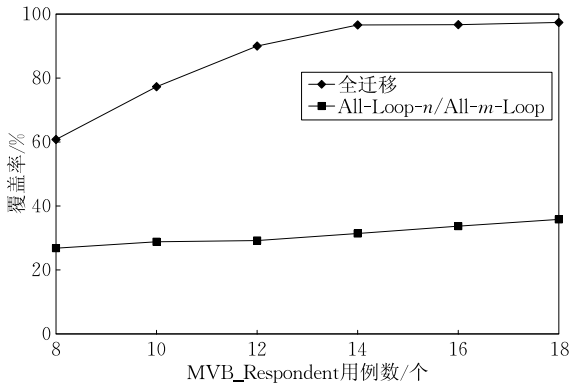


图 9 MVB_Respondent 随机方法覆盖率

由图 8、图 9 可见，随机方法生成的测试用例集对全迁移准则的覆盖率低于 ESIA-Symbolic 方法，而对 All-Loop-n/All-m-Loop 准则的覆盖率更明显低于 ESIA-Symbolic 方法. 随着用例集规模的增长，随机方法对全迁移准则的覆盖率逐渐逼近 ESIA-Symbolic 方法，而对 All-Loop-n/All-m-Loop 准则的覆盖率与 ESIA-Symbolic 方法仍存在很大差距.

按前述等价类要求，ESIA-Symbolic 方法使用基于约束条件翻转、基于插桩等方法生成正常、异常测试数据，对相关等价类的覆盖率达到 100%. 人工设计的测试用例集对数据类型、等价类的覆盖率也可达到 100%，但由于人工设计方法将被测对象视为黑盒，而未考虑软件内部运行机制，设计的测试用例集规模有限，对 All-Loop-n/All-m-Loop 准则的覆盖率很低.

4.3.3 算法执行效率

ESIA-Symbolic 方法可在短时间内生成测试用例集. 以 RTP 软件为被测软件，使用 ESIA-Symbolic 方法与 EIA-SATG 算法生成测试用例集的规模与所耗费的时间如表 6、表 7 所示. 其中，用例平均生成时间＝(测试路径生成时间＋正常用例生成时间＋异常用例生成时间)/(正常用例数＋异常用例数). ESIA-Symbolic 方法与 EIA-SATG 算法在 MVB_Caller、MVB_Respondent 之上的用例平均生成时间比分别为 31.1% 和 34.7%，其中，用例平均生成时间比＝(ESIA-Symbolic 方法用例平均生成时间/EIA-SATG 算法用例平均生成时间)×100%.

表 6 MVB_Caller 用例集生成时间

方法类型	正常测试路径数	异常测试路径数	测试路径生成时间/ms	正常测试用例数	正常用例生成时间/ms	异常测试用例数	异常用例生成时间/ms	用例平均生成时间/ms
ESIA-Symbolic	3	5	218	3	42	31	120	11.2
EIA-SATG	8	—	1	8	287	—	—	36.0

表 7 MVB_Respondent 用例集生成时间

方法类型	正常测试路径数	异常测试路径数	测试路径生成时间	正常测试用例数	正常用例生成时间/ms	异常测试用例数	异常用例生成时间/ms	用例平均生成时间/ms
ESIA-Symbolic	7	11	395	7	75	41	160	13.1
EIA-SATG	18	—	2	18	452	—	—	37.8

在用例生成过程中，EIA-SATG 算法使用与 ESIA-Symbolic 方法相同的全迁移、All-Loop-n/All-m-Loop 准则，仅生成正常测试数据. 由于 ESIA-Symbolic 方法与 EIA-SATG 算法在每条正常测试路径之上均求解一组对应的正常测试数据，因此正常测试路径数与正常测试用例数相同.

在 ESIA-Symbolic 方法中，伴随测试路径的生成实时进行符号执行，因此在生成测试路径集合的过程中，较 EIA-SATG 算法将耗费更多时间. 另一方面，ESIA-Symbolic 方法生成的用例集规模更大，且包含正常、异常用例. 当生成更多的异常测试用例时，“生成时间/用例集规模”的相对成本较 EIA-SATG 算法更低，即用例平均生成时间更短.

此外，由于 EIA-SATG 算法无法识别异常路径，故在测试路径生成过程中将 ESIA-Symbolic 方法生成的异常路径作为一般性正常路径生成，因此 ESIA-Symbolic 方法生成的正常路径/用例数少于 EIA-SATG 算法.

4.3.4 工作效率

以 RTP 软件为被测软件，在生成相同规模测试用例集的情况下，ESIA-Symbolic 方法与手工设计方法的工作量如表 8 所示，其中，每人天以 8 h 有效工作时间计.

表 8 用例集设计工作量

方法类型	需求分析	模型构建	测试用例设计
ESIA-Symbolic	5 人天	1 人天	自动生成
人工设计	5 人天	—	20 min/用例

在使用 ESIA-Symbolic 方法时，每个用例的平均设计时间估算如下：

$$C_c=\frac{C_d}{N\times(M+1)}+\frac{C_b}{N\times(M+1)},$$

其中，C_c表示每个用例的平均设计时间，C_d表示用例集设计时间，C_b表示熟悉背景知识的时间，N表示模型中的平均测试路径数，M表示测试路径上的平均约束条件数.

由软件测试工作经验可知，在使用人工设计方法时，每个用例的平均设计时间通常是一个与技术人员工作经验相关的常数，表示为C_m.

由此，ESIA-Symbolic 方法与人工设计方法对每个用例的平均设计时间之比如下：

$$\frac{C_c}{C_m}=\frac{C_d}{C_m\times N\times(M+1)}+\frac{C_b}{C_m\times N\times(M+1)},$$

其中，当C_c/C_m大于 1 时，表示 ESIA-Symbolic 方法对每个用例的平均设计时间更长；当C_c/C_m小于 1 时，表示 ESIA-Symbolic 方法对每个用例的平均设计时间更短，工作效率更高.

为了对 ESIA-Symbolic 方法与人工设计方法的工作效率进行比较，首先根据前述实验结果，推算C_c/C_m计算公式中的C_d/C_m、C_b/C_m等系数，以获得C_c/C_m与模型中的平均测试路径数N、测试路径上的平均约束条件数M之间的计算关系. 随后，对C_c/C_m随N、M的变化趋势进行仿真计算，在被测对象规模持续增长的条件下，评估两种方法相对工作效率的变化情况.

由于 ESIA-Symbolic 方法与人工设计方法在测试需求分析阶段的工作量基本一致，故在C_d中仅考虑建模、用例设计等阶段的工作量.

由表 8 可知，每个用例的平均人工设计时间C_m约为 20 min. 在使用 ESIA-Symbolic 方法时，建模时间约为 1 人天，而由于用例为自动生成，其所耗费时间可忽略不计，故C_d约为 1 人天. 通过在实验室内随机挑选数名技术人员参与实验可知，具有一定基础的技术人员通过 1.5 人天左右学习，可基本掌握 ESIA-Symbolic 方法的使用，即C_b约为 1.5 人天. C_d、C_b、C_d/C_m与C_b/C_m分别计算如下：

$C_d=1$ 人天$\times 8$ h/人天$\times 60$ min/h$=480$ min；

$C_b=1.5$ 人天$\times 8$ h/人天$\times 60$ min/h$=720$ min；

$$\frac{C_d}{C_m}=\frac{480\text{ min}}{20\text{ min}}=24;$$

$$\frac{C_b}{C_m}=\frac{720\text{ min}}{20\text{ min}}=36.$$

由此，C_c/C_m计算如下：

$$\frac{C_c}{C_m}=\frac{1}{N\times(M+1)}\times\left(\frac{C_d}{C_m}+\frac{C_b}{C_m}\right)=\frac{24+36}{N\times(M+1)}=\frac{60}{N\times(M+1)}.$$

根据以上结果，以N、M作为自变量，对C_c/C_m的值进行仿真计算，并根据仿真计算数据绘制图 10. 由仿真结果可见，当被测对象规模较小时，由于背景知识学习、建模等工作将耗费一定时间，人工设计方法表现更加灵活. 而随着被测对象规模的增长，ESIA-Symbolic 方法相对于人工设计方法的工作效率将大幅提升.

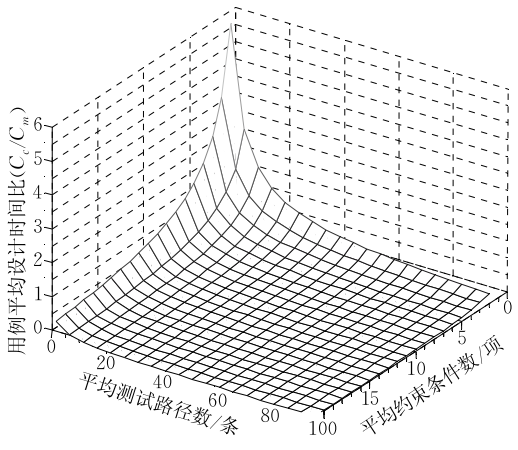


图 10　相对用例平均设计时间变化趋势

此外，ESIA-Symbolic 方法的应用也使测试目标调整以及回归测试更便捷. 测试目标变更时，仅需相应调整用例生成策略，即可针对不同目标迅速生成测试用例. 进行回归测试时，仅需根据软件修改情况，对涉及的模型结构、变量定义域、行为约束条件进行修订，并根据影响域分析结果划定回归测试用例生成范围，即可直接生成回归测试用例.

4.3.5　缺陷检测

在针对目标系统的测试工作中共发现 5 处软件缺陷，其中 4 处可通过 ESIA-Symbolic 方法自动生成的测试用例集暴露，具体如表 9 所示. ESIA-Symbolic 方法生成的测试用例集与人工设计的测试用例集具有基本一致的软件缺陷发现能力.

其中第 5 项问题，由于 ATES 平台未内建“创建具有相同参考句柄的会话”的特殊测试策略，故未针对该问题生成用例. 在人工设计测试用例时，由于人具有主观能动性，可针对特定功能点灵活设计特殊的测试策略. 此类测试策略不具有普遍性，难以在测试平台中进行一般化内建. 针对这类特殊测试策略，仍多采取人工分析、设计的方式进行处理.

表 9 缺陷发现能力分析

序号	缺陷类型	软件缺陷描述	ESIA-Symbolic 方法能力分析
1	超界数据处理错误	*call_request* 行为参数$\chi_{c_rep_topo_a}$取值为非法值*AM_MAX_TOPO*+1时，通信状态显示正常，未对$\chi_{c_rep_topo_a}$参数取值进行合法性检查；	可暴露，基于约束条件翻转方法使参数$\chi_{c_rep_topo_a}$的取值大于*AM_MAX_TOPO*(63)，生成异常测试用例，可发现此缺陷；
2	通信状态判别错误	执行 *re_request* 行为时，数据传输正常，而通信状态显示异常；	可暴露，包含 *re_request* 行为的正常测试用例可发现此缺陷；
3	寻址地址配置错误	*re_request* 行为参数$\chi_{r_rep_fun_d}$取值为错误值时，仍可正常返回响应消息；	可暴露，基于插桩方法指定参数$\chi_{r_rep_fun_d}$的取值为错误地址，生成异常测试用例，可发现此缺陷；
4	通信状态判别错误	*put_var* 行为参数$\chi_{c_traffic_store}$取值为错误值时，无法传输数据，但通信状态显示正常，未对端口号取值进行合法性检查；	可暴露，基于插桩方法指定参数$\chi_{c_traffic_store}$的取值为错误地址，生成异常测试用例，可发现此缺陷；
5	会话句柄配置错误	当发送端正在发送一条长消息的同时，在接收端再次执行一项与当前通信过程具有相同参考句柄的 *receive_request* 行为，原通信过程发生故障，在发送端将无法收到接收端的响应消息	无法暴漏，ATES 平台中未内建“创建具有相同参考句柄的会话”的测试策略，未生成相关测试用例

ESIA-Symbolic 方法可对软件设计方案中的部分可判定路径进行检测与识别，辅助技术人员发现软件设计方案中的缺陷. 实验过程中在 MVB_Caller 中建立输入行为 $long\_call\_confirm(\chi_{c\_rep\_long},\chi_{c\_rep\_len\_long},\chi_{c\_rep\_buf\_len\_long})$，并建立其前置条件 $PreCon(long\_call\_confirm)=(\chi_{c\_state}==2)\wedge(\chi_{c\_rep\_long}==\mathrm{TRUE})\wedge(4096\leq\chi_{c\_rep\_len\_long}\leq\chi_{c\_rep\_len\_ref})\wedge(\chi_{c\_rep\_len}\leq\chi_{c\_rep\_buf\_len})$，如图 11 所示.

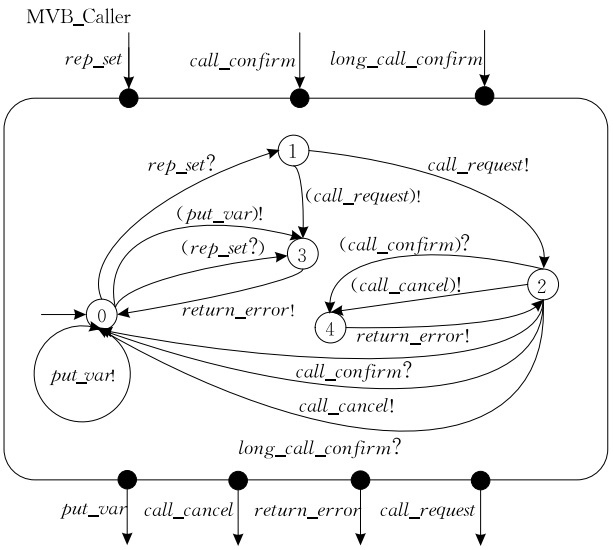


图 11 植入部分可判定路径的 MVB_Caller

由于在 *rep_set* 行为后置条件中存在约束条件($1\leq\chi_{c\_rep\_len\_ref}\leq4095$)，与约束条件($4096\leq\chi_{c\_rep\_len\_long}\leq\chi_{c\_rep\_len\_ref}$)不一致，故 *long_call_confirm* 行为的引入将在模型中植入部分可判定路径，即路径“0→*rep_set*→1→*call_request*→2→*long_call_confirm*→0”. ESIA-Symbolic 方法执行结果显示其可正确检测出所植入的部分可判定路径，并可根据该反例生成相应测试用例.

此外，ESIA-Symbolic 方法从构件外部行为特性和内部运行机制两方面着眼，可对在各类应用场景下的构件执行情况进行充分测试，特别对循环结构、内部变量可进行有针对性的测试，因而对构件内部运行机制中隐含的缺陷具有更高的发现率. 人工设计方法则未考虑软件内部运行机制，难以对各类应用场景进行充分测试.

4.3.6 局限性分析

(1) 模型构建的完整性、准确性

在利用 ESIA-Symbolic 方法生成测试用例时，首先需要依据被测软件构件的功能需求与设计方案建立相应 ESIA 模型，进行充分的测试需求分析，进而生成足够充分的测试用例集合. 这需要测试人员获取相关领域知识，并进行完整、准确的模型构建. 由于对目前领域知识、功能需求的存在形式难以进行自动抽象，因此对领域知识的获取与理解、对功能需求的抽象与建模等更多依赖于测试人员的技术水平与工程经验. 这不仅需要耗费测试人员大量的精力，同时模型的完整性、准确性等难以保证. 对于大规模、复杂嵌入式系统，由于其领域知识的广泛性、功能需求的复杂性等，使得此问题更加突出.

基于模型的设计方法(Model-Based Design，MBD)通过建立软件模型，提供对软件预期行为或结构的抽象定义，进行基于模型的设计方案验证及生成目标代码. MBT 是 MBD 过程中的一项重要内容. 本文针对嵌入式软件构件提出的 ESIA 模型既可作为 MBT 的基础模型，也可支持 MBD 的开发过程.

本文在进行测试生成之前，假定建立的被测构件 ESIA 模型能充分描述被测构件的功能特性. 在此基础上，通过 ESIA-Symbolic 方法，生成足够充分的测试用例集. 同时，针对嵌入式软件测试环境特点，通过符号执行、插桩等方法，避免无效测试路径、

测试数据的生成.模型构建的完整性与准确性是MBD领域内的重要研究内容,已有很多研究工作,立足于设计与开发层面解决该问题.经过扩展,ESIA模型元素可支持对嵌入式软件构件进行完整与准确的描述,在后续研究工作中,将进一步从设计与开发层面,研究利用ESIA模型对嵌入式软件进行完整、准确建模的方法.

(2)约束条件间的复杂依赖关系

通常情况下,构件的执行路径由外部事件序列以及多个参数的取值共同确定.如果这些参数的取值完全取决于外部输入行为,相互之间不存在依赖关系,则采用约束求解技术沿路径直接求解.如果某项参数的取值还受到前驱行为执行效果的影响,则其取值与前驱行为的参数或内部变量之间存在依赖关系.当这种依赖关系是复杂计算、数据处理过程时,将在测试路径之上引入复杂约束关系、甚至复杂非线性约束关系,不仅使 PC 表达式的构成更复杂,其求解也将耗费更多的计算资源,影响测试生成效率.在后续研究工作中,考虑通过优化运行配置、采用约束化简技术等措施,对算法进行进一步优化.

5　相关工作

其他学者对软件接口模型扩展、基于接口模型以及基于符号执行的测试生成技术也进行了研究,取得了一系列成果.

5.1　接口模型扩展

1998年至2003年间,Lynch、Tuttle、Segala和Vaandrager[14-15]提出并发展了I/O自动机(I/O Automat,IOA).IOA同样是一种轻量级的构件接口模型.与IA不同,IOA是输入使能的,即在所有状态上、所有输入行为均可被激活,IOA仅能限制输出行为和内部行为的执行时机.因此,IOA在描述反应式系统时形式更复杂.IOA上的可组合性定义是一种悲观定义,即只有组合构造的子系统,在任何环境中都能正常运行,而不会进入异常状态时,构件间才是可组合的.

2009年至2011年间,Mouelhi、Chouali和Mountassir[16-17]提出了语义接口自动机(Semantic Interface Automata,SIA).在SIA中,定义了行为参数、共享变量和局部变量.其中,行为参数与行为名称共同构成行为签名;局部变量描述构件所维护的内部变量;共享变量描述构件与其运行环境以及其他构件间共享的变量,主要用于对构件之间的一致性状态进行统一描述.共享变量的存在对构件的封装性造成不利影响.此外,在SIA中定义了行为执行的前置、后置条件、效果等,并以一阶逻辑表达式描述.

2012年,Cao和Wang[18]通过将Z语言与IA相结合,利用Z语言描述抽象数据结构的能力,对IA变量、数据特征的语义进行扩展,提出了Z标记接口自动机(Interface Automaton with Z notation,ZIA)模型.在ZIA中,状态与行为具有的数据特征均以一阶逻辑形式描述,简单数据特征之间可以运算符连接构成复杂数据特征,并可对行为执行之前、之后的数据特征进行区别描述.由此,在IA中建立了基于数据特征的行为约束,对数据取值约束与行为约束进行统一描述.

同样在2012年,Zhang[19]提出了一种扩展接口自动机(Extended Interface Automata,EIA).在EIA中,分别定义了输入、输出以及内部变量集合,并以输入、输出以及内部变量分别作为构件输入、输出以及内部行为的参数,但构件运行过程中所维护的中间计算变量并未包含在以上变量集合中.进一步选取OCL子集,定义了EIA的模型约束语言及相关语义,并定义了各类约束条件表达式.

在以上对IA模型的扩展中,多将构件行为分别纳入输入、输出与内部行为集合,分别以输入、输出与内部变量作为相应行为的参数,并定义行为的前置条件与后置条件.在EIA中,还定义了变量更新函数,以表示执行过程中的变量更新过程.本文提出的ESIA模型,不仅定义了行为参数,同时定义了构件运行过程中维护的中间计算变量,并将其纳入到内部变量集合之中,同时定义了变量约束的基本类型,给出由基本约束构造复杂约束的方法.本文还对行为约束与变量约束的描述方式进行统一,在前置、后置条件中,整合了各类状态变量、行为参数、中间计算变量的取值约束.

5.2　基于接口自动机的测试

一些学者以IA模型为基础模型,进行了测试技术的研究.2008年,Li等人[20]提出了一种基于IA的测试生成方法.通过在IA中定义可控运行片段、典型状态、特征序列等,提出可控运行片段的深度优先生成算法,并提出通过可控运行片段、特征序列构造测试序列的方法.在以上研究中,对IA模型的扩展与ESIA不同,并未对IA进行变量、行为参数扩展,也未定义行为的约束条件.在测试生成过程中,以上方法将生成一组测试序列,未在测试序列上进

行行为参数、变量的求解，而 ESIA-Symbolic 方法不仅生成测试路径，还在测试路径上进行约束求解以求取测试数据.

2014 年，Yang 等人[21]对 IA 进行时间特性扩展，以期提供详细的软件时序信息. 针对输入值受其他模块影响的软件输入行为，通过建立输入自动机，提供输入数据产生方式的模型描述. 在输入自动机上，进一步定义可控运行片段，提出可控运行片段的深度优先生成算法，并基于可控运行片段上的时间信息生成测试用例. 以上研究工作主要关注构件间交互行为的时间特性测试，而本文提出的 ESIA-Symbolic 方法则主要关注功能特性的测试，因此二者生成的测试数据是针对软件的不同方面特性.

5.3 基于符号执行的测试

King[3]在提出符号执行基本概念的同时，设计了一个交互式符号执行系统，即 EFFIGY 系统. 该系统具备基本调试功能，其内置测试管理器，可在生成的符号执行树中探索候选测试路径，并可根据使用者提供的断言，生成验证条件，检验用例执行结果.

Clarke[4]设计了一个基于符号执行的测试数据生成系统，以为 Fortran 语言程序生成测试数据. 该系统包括程序预处理、符号执行、约束化简、约束求解四部分. 其中，在符号执行部分，沿不同执行路径，使用计算表、变量演化图、符号表等数据结构生成输出变量的符号表达式，推导符号执行结果，并同步建立路径约束. 随后，将复杂变量约束化简为线性约束表达式，利用线性求解算法进行求解，以生成不同路径上的测试数据.

2010 年，Papadakis 和 Malevris[22]提出一种动态符号执行方法，以生成基于变异的测试数据. 在动态符号执行过程中，*PC* 中的每项约束条件被依次翻转，以实现约束条件的变异，在生成的 *PC* 引导下遍历不同路径，直到探索到所有可达路径为止. Papadakis 等人利用该技术实现了一个测试用例自动生成框架.

SE 技术作为一种有效的测试数据生成技术，在一系列测试工具中得到应用. 如 PathFinder 是一款 Java 字节码符号执行工具，通过符号执行方法求取变量约束，并使用货架约束求解器进行约束求解，可自动生成测试用例，并侦测软件缺陷[23]. CUTE 工具可对多线程程序进行测试，并使用指针操控动态数据结构，在此基础上推出了针对 Java 程序的 JCUTE 工具[5,24]. CREST 则是一款针对 C 语言程序的开源测试工具，采用启发式方法在众多候选路径中选择生成测试路径[5,25].

在以往符号执行方法的应用中，主要针对程序源代码开展测试，而未涉及符号执行在构件模型中的应用. 本文将符号执行方法引入 MBT 过程，提出了以上基于 ESIA 模型与符号执行的构件测试用例自动生成方法.

6 总 结

软件测试是保证嵌入式软件开发质量的有效手段. 由于 CBD 技术越来越广泛地应用于大规模、复杂嵌入式软件开发，通过嵌入式软件构件测试保证构件设计与实现的正确性成为重要的研究内容. 本文基于嵌入式软件构件特点，提出一种基于 ESIA 模型与 SE 技术的 ESIA-Symbolic 方法，自动生成满足结构覆盖与数据覆盖准则的测试用例集.

基于事件的反应式嵌入式软件在嵌入式系统中普遍存在，其具有以下特点：

(1) 反应式系统. 在运行过程中与外部环境进行交互，对各类外部事件作出反应，通过数据通信进行构件间交互与协同.

(2) 事件驱动. 构件的运行由外部事件序列驱动，执行路径由外部事件序列与参数、变量取值共同确定.

(3) 基于构件. 通过接口响应外部事件、提供构件服务，同时屏蔽构件内部结构，并支持组装和重用，构件通常具有行为参数、内部变量的取值约束.

(4) 异常处理. 异常处理机制是构件可靠性设计与验证的重要内容.

针对上述特点，本文提出了 ESIA 模型对该类嵌入式构件进行抽象建模. ESIA 模型是一种描述接口行为的状态机模型. 通过基于接口事件的状态迁移序列，描述构件的反应式运行过程. 通过各类参数变量和内部变量，描述构件的行为参数、中间计算变量与通信数据. 通过具有参数的接口行为，描述构件具有的外部行为特性与构件间交互过程. 通过各类约束条件，描述构件行为的执行约束与构件的运行规律. 通过异常状态以及相关联的异常处理行为，描述构件的异常处理机制. 因此，ESIA 模型适用于反应式嵌入式软件构件的建模.

本文提出的 ESIA-Symbolic 测试生成方法以 ESIA 模型为基础模型，支持正常、异常测试用例的自动生成. 通过结构覆盖与数据覆盖准则，从模型结

构元素覆盖、数据类型和等价类覆盖两个方面指导测试生成.通过基于符号执行的测试路径生成算法，生成满足模型覆盖准则的正常、异常测试路径集合，并在测试路径搜索过程中，实时分析行为参数取值约束的可满足性，检测执行路径是否可判定，识别异常状态与异常测试路径.在每条正常测试路径之上，采用约束求解技术生成正常测试数据，并使用基于约束条件翻转、基于插桩、基于反例等方法生成异常测试数据.

由于反应式嵌入式软件构件在嵌入式系统中普遍存在，且本文实验结果对以上构件均有一定的效果，因而可考虑进一步推广到较大规模的同类嵌入式软件系统中.

实验结果表明，以上方法针对嵌入式软件构件生成充分的正常、异常测试用例集，在提高测试效率的同时保证测试的充分性，并可在测试路径生成过程中实时判断路径是否可判定，是进行嵌入式软件构件测试的有效方法.

致　谢　在此，我们向对本文的工作给予支持和建议的同行，尤其是清华大学计算机科学与技术系陈文光教授以及实验室内各位老师和同学表示感谢！

参　考　文　献

[1] Zandet J, Schieferdecker I, Mosterman P J, et al. Model-based testing for embedded systems. Boca Raton, USA: CRC Press, 2011

[2] De Alfaro L, Henzinger T A. Interface automata//Proceedings of the 9th Annual Symposium on Foundations of Software Engineering. New York, USA, 2001: 109-120

[3] King J C. Symbolic execution and program testing. Communications of the ACM, 1976, 19(7): 385-394

[4] Clarke L A. A system to generate test data and symbolically execute programs. IEEE Transactions on Software Engineering, 1976, SE-2(3): 215-222

[5] Cadar C, Sen K. Symbolic execution for software testing: Three decades later. Communications of the ACM, 2013, 56(2): 82-90

[6] Pasareanu C S, Visser W. A survey of new trends in symbolic execution for software testing and analysis. International Journal on Software Tools for Technology Transfer, 2009, 11(4): 339-353

[7] Jussien N, Rochart G, Lorca X. Choco: An open source Java constraint programming library//Proceedings of the CPAIOR'08 Workshop on Open-Source Software for Integer and Constraint Programming. Paris, France, 2008: 1-10

[8] Gent I P, Jefferson C, Miguel I. Minion: A fast scalable constraint solver//Proceedings of the 17th European Conference on Artificial Intelligence. Riva del Garda, Italy, 2006: 98-102

[9] Cadar C, Ganesh V, Pawlowski P M, et al. EXE: Automatically generating inputs of death. ACM Transactions on Information and System Security, 2008, 12(2): 1-38

[10] Cadar C, Dunbar D, Engler D R. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs//Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation. San Diego, USA, 2008: 209-224

[11] Cadar C, Engler D. Execution generated test cases: How to make systems code crash itself//Proceedings of the 12th International SPIN Workshop. San Francisco, USA, 2005: 22-24

[12] International Electrotechnical Commission. Electric railway equipment-train bus-Part 1: Train communication network. Geneva, Switzerland: International Electrotechnical Commission, 1999

[13] Zhang C, Bai X Y, Li J L, Zhang R W. Automated test case generation for embedded software using extended interface automata//Proceedings of the 13th International Conference on Quality Software. Nanjing, China, 2013: 292-298

[14] Lynch N, Tuttle M R. An introduction to input/output automata. MIT Laboratory for Computer Science, Cambridge, MA, USA: Technical Report MIT/LCS/TM-373, 1988

[15] Lynch N, Segala R, Vaandrager F. Hybrid I/O automata. Information and Computation, 2003, 185(1): 105-157

[16] Mouelhi S, Chouali S, Mountassir H. Refinement of interface automata strengthened by action semantics. Electronic Notes in Theoretical Computer Science, 2009, 253(1): 111-126

[17] Mouelhi S, Chouali S, Mountassir H. Invariant preservation by component composition using semantical interface automata //Proceedings of the 6th International Conference on Software Engineering Advances. Barcelona, Spanish, 2011: 305-311

[18] Cao Z N, Wang H. Extending interface automata with Z notation//Proceedings of the 4th IPM International Conference on Fundamentals of Software Engineering. Tehran, Iran, 2011: 359-367

[19] Zhang Ren-Wei. Design and Implementation of Test Cases Generation System Based on Extended Interface Automata [M. S. dissertation]. Peking University, Beijing, 2012 (in Chinese)
(张任伟. 一种基于扩展接口自动机的测试用例生成子系统的设计与实现[硕士学位论文]. 北京大学，北京，2012)

[20] Li L M, Liu L, Wang Z J, Tang Y L. Research on interface automata testing//Proceedings of the 2008 International Conference on Computer Science and Software Engineering. Wuhan, China, 2008: 743-746

[21] Yang S K, Xu J Q, Man T L, Liu B. Real-time extended interface automata for software testing cases generation. The Scientific World Journal, 2014(2014)

[22] Papadakis M, Malevris N. Automatic mutation test case generation via dynamic symbolic execution//Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering. San Jose, USA, 2010: 121-130

[23] Pasareanu C S, Rungta N. Symbolic PathFinder: Symbolic execution of Java bytecode//Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering. Antwerp, Belgium, 2010: 179-180

[24] Sen K, Agha G. CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools//Proceedings of the 18th International Conference on Computer Aided Verification. Seattle, USA, 2006: 419-423

[25] Burnim J, Sen K. Heuristics for scalable dynamic test generation //Proceedings of the 23th IEEE/ACM International Conference on Automated Software Engineering. Washington, USA, 2008: 443-446

WANG Bo, born in 1979, Ph. D. candidate. His main research interests include software testing and embedded system.

BAI Xiao-Ying, born in 1973, Ph.D., associate professor. Her research interests include software testing and service computing.

ZHANG Chao, born in 1991, Ph. D. candidate. His research interests include software testing and model checking.

HE Fei, born in 1980, Ph. D., associate professor. Her research interests include formal methods and automata theory.

SONG Xiao-Yu, born in 1963, Ph. D., professor. His research interests include formal methods and embedded computing system.

Background

With the increasing size and complexity of embedded systems, Component-Based Development (CBD) has been widely applied in large-scale embedded system development. CBD builds system by composing reusable components. It aims to reduce development cost while enhance reliability and maintainability. The quality of components and composition are thus critical to system reliability.

Testing is an effective quality insurance method. The paper proposes a model-based automated testing technique for embedded component. A model called Extended Semantic Interface Automata (ESIA) is designed to capture component interface semantics including data, behavior, conditions, and constraints. It assumes that with sufficient domain knowledge, the model can provide a formal description of the expected behavior that the component exposes to others. To derive tests from ESIA, the paper investigates symbolic execution algorithms to generate test cases as sequences of state transitions. Both normal and abnormal test inputs are generated based on analysis of constraints along test paths.

The work is part of a project on high-speed railway communication systems. The experiments are carried on an in-house developed component of Multifunctional Vehicle Bus (MVB) with Real Time Protocol (RTP). The model is built based on an IEC (International Electrotechnical Commission) standard for railway communication network IEC 61375-1 (Electric railway equipment-Train bus). The generated test cases are compared with manually-developed test scripts during MVB product testing.

This research is supported by the National Basic Research Program(973 Program) of China (2011CB302505), the National High Technology Research and Development Program (863 Program) of China (2013AA01A215), the National Natural Science Foundation of China (91218302, 61472197), and the Beijing Natural Science Foundation (4132062).