基于执行序列的嵌入式软件时序异常检测

王 博[1),2)] 白晓颖[1),2)] 陈文光[1),2)] SONG Xiaoyu[3)]

[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)](波特兰州立大学电气与计算机工程系 波特兰 97207 美国)

摘 要 时序特性是嵌入式软件的重要特性,实时嵌入式软件运行的正确性不仅依赖于任务执行结果,更依赖于任务执行时间.执行任务往往具有多个时间约束,且相互之间可能存在制约关系,对时间约束进行验证和确认是嵌入式软件测试的一个重要问题.文中提出一种基于执行片段的嵌入式软件时序特性检测方法(Trace-Based Temporal Defect Detection,TBTDD).一方面,基于扩展语义接口自动机(Extended Semantic Interface Automata,ESIA)模型,刻画嵌入式软件时序特性需求,分析不同时间约束间存在的相关关系类型,并提出基于相关矩阵的相关时间约束识别算法;另一方面,在目标软件运行环境中提取包含时间信息的执行片段,通过执行片段与时间约束模型的匹配,依据预先制定的时序特性检测准则,检验执行序列是否满足模型中各项独立和相关时间约束的要求,进而发现被测软件中存在的时序缺陷.实验以卫星定位系统软件为例进行建模与缺陷检测,并在执行片段集和缺陷检测能力等方面进行了对比分析.实验表明,该方法可有效检测软件运行过程中存在的各类异常时序,提高了软件时序测试的有效性和充分性.

关键词 接口自动机;时间约束;执行片段;相关性分析;时序缺陷检测;软件测试;嵌入式软件

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2017.02635

Temporal Defect Detection of Embedded Software Using Timed Execution Trace

WANG Bo[1),2)] BAI Xiao-Ying[1),2)] CHEN Wen-Guang[1),2)] SONG Xiaoyu[3)]

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

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

[3)](*Department of Electrical and Computer Engineering, Portland State University, Portland*, 97207, *USA*)

Abstract Temporal requirements are critical to embedded software. The correctness of system operation depends on its satisfications to the timing constraints. In an embedded system, there usually exist many timing constraints which may also correlate with each other. It is important to ensure that the execution of the system conforms to its temporal requirements. The paper proposes an approach for detecting temporal defects based on execution traces analysis. The characteristics of timing constraints are analyzed, and the ESIA (Extended Semantic Interface Automata) is proposed to characterize temporal requirements of embedded software. Then the correlation patterns of timing constraints are analyzed, and an algorithm using timing matrix is proposed to identify the correlated timing constraints in ESIA model. Traces tagged with time stamps are gathered from the execution environment of the target software. According to the temporal checking rules predefined, the traces are matched to the temporal model to verify its conformance to timing constraints, both individual as well as correlated constraints. In case violations detected,

收稿日期:2016-05-30;在线出版日期:2016-11-17.本课题得到国家自然科学基金(91218302,61472197)、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.**陈文光**,男,1972 年生,博士,教授,中国计算机学会(CCF)会员,主要研究领域为并行与分布式计算.**SONG Xiaoyu**,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.

potential defects will be reported for further investigation. Experiments are exercised on a satellite positioning system for modeling and defect detection. The results are compared with other approaches in terms traces analysis and defect detection capabilities. It showed that the proposed approach can effectively detect temporal anomalies in system execution, which enhance the effectiveness and efficiency of temporal testing of embedded software.

Keywords interface automata; timing constraint; execution trace; correlation analysis; temporal defect detection; software testing; embedded software

1 引 言

时序特性是否满足设计要求将对实时嵌入式软件运行产生关键影响,时序异常可能造成任务执行失败,甚至对系统造成破坏性影响.特别在具有高实时和高安全性要求的应用领域内,例如航空机载任务系统、航天发射控制以及高速铁路运控等,各项任务的执行具有严格时间约束,对系统的安全和高效运行起着至关重要的作用.因此,保证嵌入式软件时序设计以及实现的正确性是嵌入式软件开发的重要任务.

嵌入式软件时序测试是检验嵌入式软件时序特性是否满足设计要求以及保证软件质量的重要手段.传统时序测试方法需要介入甚至控制软件运行过程,其核心是测试用例设计.在特定运行场景下,通过在目标软件运行环境中设置激励和预置数据,控制其按预期方式运行,随后观察、采集运行结果,并检查运行结果是否满足设计要求.

随着嵌入式软件规模和复杂度的不断上升,传统时序测试方法面临一些困难和局限性,时序测试与验证愈发成为一项困难的工作.首先,在某些情况下,测试人员无法直接获得目标软件控制接口,无法暂停、终止或介入软件运行过程,造成部分场景无法设置,部分用例无法执行;由于仪器性能限制以及数据观测误差等,测试人员无法从外部采集精确的运行时间信息;当嵌入式软件是由多个构件组成的分布式系统时,测试人员难以协调控制各构件间的运行时序.以上因素均可能造成嵌入式软件时序特性测试不充分和结果不准确.

其次,嵌入式软件中往往存在大量时间约束,限制各项任务的执行时间.不同任务的执行路径可能相互重叠,其执行时间之间可能存在特定依赖关系,从而造成相应时间约束之间间接相关.因此,在时序测试中,不仅需检验各项任务的执行时间是否满足相应时间约束,还需检验任务的执行时间是否满足此类依赖关系,以检测可能的时序异常,而传统测试方法难以针对此类相关时间约束进行测试.

再次,由于对目标软件理解以及测试环境条件的局限性,通常情况下,设计的测试用例集难以完全覆盖软件运行过程中的各类正常以及异常时序,特别是难以对偶发的异常时序进行测试.

针对以上问题,本文提出一种基于执行序列的时序缺陷检测方法(Trace-Based Temporal Defect Detection,TBTDD).相对于传统时序测试方法,TBTDD 无需控制目标软件运行,而通过持续监视过程,提取一组包含时间信息的执行片段,进而根据目标软件形式化描述,对收集的软件运行信息进行离线自动检测,以考察软件实现是否满足设计要求.TBTDD 方法可识别执行路径上不同时间约束间的相关关系,并对不同任务执行时间之间的依赖关系进行检测.TBTDD 方法可伴随目标系统交付运转过程执行,通过长期收集有效的执行片段,积累丰富的运行场景,实现对目标软件各类正常以及异常时序的充分检测.本文主要内容如下:

(1)时序特性分析与建模.对嵌入式软件时序特性进行分析,在扩展语义接口自动机(Extended Semantic Interface Automata,ESIA)模型中定义时间变量、置零行为以及时间约束等元素,建立 ESIA 模型时序语义,支持嵌入式软件时序特性的定量和精确描述.

(2)时间约束相关性分析.在 ESIA 模型的特定执行路径之上,分析不同时间变量的取值之间可能存在的依赖关系,据此定义不同时间约束间的包含相关和交叉相关关系.

(3)TBTDD 方法及算法.对嵌入式软件运行时序检测流程进行分析与讨论,在 ESIA 模型之上定义包含时间信息的执行片段,制定执行片段可接受性准则,提出执行片段匹配以及时序缺陷检测等算法,以检查嵌入式软件执行序列包含的时间信息是否满足设计要求.

实验结果显示，TBTDD 方法可有效检测嵌入式软件执行片段中存在的各类时序缺陷，特别是违反相关时间约束以及异常时序处理错误等缺陷，与传统时序测试方法配合使用时，可对目标软件时序特性进行更充分的测试. TBTDD 方法不需要预先设计测试用例与测试场景，降低了测试环境和人为因素对测试工作的限制与影响. TBTDD 方法不会干扰软件正常运行，可获得准确的运行时序信息与测试结果.

本文第 2 节描述研究背景；第 3 节介绍 ESIA 模型时序语义相关内容，讨论不同时间约束之间的相关关系；第 4 节介绍 TBTDD 方法的工作原理与系列算法等；第 5 节介绍实验设计，以卫星定位系统软件为例，进行时序特性建模与时序缺陷检测，以验证本文方法的有效性；第 6 节叙述与本文研究相关的工作；第 7 节总结全文内容.

2　研究背景

精确与完整的建模是理解目标系统时序特性和开展时序缺陷检测的基础. 近年来涌现出一系列时序特性定量描述模型，例如时间自动机(Timed Automata，TA)、时序描述逻辑(Temporal Description Logics，TDL)、统一建模语言(Unified Modeling Language，UML)、时间 Petri 网(Timed Petri Net，TPN)以及马尔科夫链(Markov Chain)等[1-3]，均可精确刻画嵌入式软件时序特性.

接口自动机(Interface Automat，IA)是一种轻量级的构件接口模型[4]. IA 模型通过状态机上的接口行为描述构件间的通信与交互过程，刻画系统外部行为特性，隐藏构件内部结构，是可组合嵌入式软件系统建模的有效工具.

定义 1.　接口自动机 IA[4]. 嵌入式软件 IA 定义为 $P=(V_P,V_P^{on},A_P^I,A_P^O,A_P^H,\Delta_P)$，其中：

V_P是自动机包含的状态集合；

V_P^{on}是自动机包含的初始状态集合，$V_P^{on}\subseteq V_P$，若 $V_P^{on}=\varnothing$，则称 P 为空；

A_P^I、A_P^O和 A_P^H分别为输入、输出和内部行为集合，$A_P^I\cap A_P^O=A_P^I\cap A_P^H=A_P^O\cap A_P^H=\varnothing$；$A_P=A_P^I\cup A_P^O\cup A_P^H$表示全部行为的集合；

$\Delta_P\subseteq V_P\times A_P\times V_P$是自动机包含的状态迁移集合.

ESIA 模型通过对 IA 进行变量以及执行约束等扩展，支持反应式嵌入式软件行为特性的建模与分析[5]. 在 ESIA 模型中，引入输入、输出行为参数、内部计算变量、参数约束条件以及行为前置/后置条件等，对软件运行过程中的数据信息进行描述.

定义 2.　扩展语义接口自动机 ESIA[5]. 嵌入式软件 ESIA 定义为 $P=(V_P,V_P^{on},X_P,A_P,\Psi_P,E,\Delta_P)$，其中：

V_P是自动机包含的状态集合；

V_P^{on}是自动机包含的初始状态集合，$V_P^{on}\subseteq V_P$，若 $V_P^{on}=\varnothing$，则称 P 为空；

X_P^R和 X_P^H分别为行为参数和内部变量集合，$X_P^R\cap X_P^H=\varnothing$；其中内部变量描述软件维护的中间计算变量与计算结果；$X_P=X_P^R\cup X_P^H$表示全部变量的集合；

A_P^I、A_P^O和 A_P^H分别为输入、输出和内部行为集合，$A_P^I\cap A_P^O=A_P^I\cap A_P^H=A_P^O\cap A_P^H=\varnothing$；$A_P=A_P^I\cup A_P^O\cup A_P^H$表示全部行为的集合；

Ψ_P是行为执行约束集合；$a\in A_P$是一项行为，$\forall\psi(a)\in\Psi_P$是行为 a 的一组执行约束，表示为 $\psi(a)=(PreCon(a),PostCon(a))$，其中，$PreCon(a)$和 $PostCon(a)$分别为行为 a 的前置和后置条件；

E 是触发事件集合，$\forall e\in E$ 触发 P 中特定行为的执行；

$\Delta_P\subseteq V_P\times A_P\times V_P$是自动机包含的状态迁移集合，$\forall\delta\in\Delta_P$表示一个状态迁移过程，具体描述如下：

$$\delta=(q)\xrightarrow[a]{e,PreCon(a),eff(a),PostCon(a)}(q')；$$

其中，$\delta.e$ 表示行为触发事件，$\delta.a$ 表示引起状态迁移的行为，$\delta.eff(a)$表示行为 a 产生的执行效果，$\delta.PreCon$ 表示行为 a 的前置条件，$\delta.PostCon$ 表示行为 a 的后置条件，$\delta.s=q$ 表示源状态，$\delta.t=q'$表示目标状态.

在 ESIA 模型基础上，设计了一种基于符号执行的测试用例生成算法，通过搜索有效的事件/数据序列，自动生成测试用例与测试场景[5].

3　嵌入式系统时序特性分析

本节以一个门禁控制系统为例，对嵌入式软件时序特性进行分析. 在此基础上，定义时间变量以及时间约束等时序元素，建立 ESIA 模型时序语义，支持嵌入式软件时序特性的定量描述与理解，并进一步分析不同时间约束之间存在的典型相关关系.

3.1　示　例

门禁控制软件是一项典型的嵌入式软件，在

特定事件触发下，按一定时序打开或关闭门禁，其ESIA模型如图1所示.

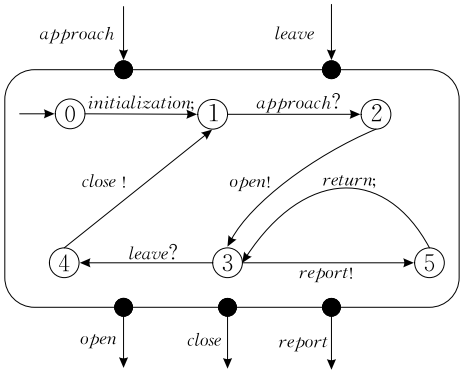


图1 门禁控制软件ESIA模型(其中，“?”表示输入行为，“!”表示输出行为，“;”表示内部行为)

门禁控制软件根据接收到的人员抵近(*approach*)或离开事件(*leave*)，按设计时序执行打开门禁(*open*)、关闭门禁(*close*)和向控制中心上报当前状态(*report*)等行为.其中：

(1)状态0是门禁系统上电初始状态；

(2)当控制软件完成初始化后，进入等待状态1；

(3)当检测到人员抵近后，进入状态2，准备打开自动门；

(4)当自动门打开后，进入门禁打开状态3；

(5)当检测到人员离开后，进入状态4，随即关闭自动门，返回等待状态1；

(6)在自动门打开时(状态3下)，门禁控制软件向监控中心周期上报状态信息并进入报告状态5，直到检测到人员离开后，退出周期上报过程.

在门禁控制软件中设定了一系列时间约束，具体包括：

DC-TR1. 门禁控制软件在系统上电后5 s时完成初始化；

DC-TR2. 当检测到人员抵近后，自动门应在时间区间[3 s,7 s]内打开；

DC-TR3. 当检测到人员离开后，自动门应在时间区间[3 s,5 s]内关闭；

DC-TR4. 当检测到人员抵近后，应在时间区间[3 s,30 s]内检测到相关人员离开；

DC-TR5. 在自动门打开后，自动门应在30 s内关闭，即自动门持续打开时间不应超过30 s；

DC-TR6. 在自动门打开时(状态3)，门禁控制软件每15 s向监控中心上报一次自动门打开状态.

3.2 时间约束分析与建模

在嵌入式软件中，时间约束限制特定任务的执行时间，例如任务开始、结束或持续时间等.当执行时间满足时间约束时，软件将表现出符合设计预期的时序特性.Dasarathy[6]将时间约束分为最小值约束、最大值约束以及持续约束.其中，最小值和最大值约束分别描述两个事件之间的最小和最大时间间隔，而持续约束描述一个事件从产生到结束的持续时间.在本文中，根据限制对象的不同，分别定义以下三类时间约束.

定义3. 时间点约束.限制事件产生的时间点.设τ为一项时间变量，t为一个时间点，一项时间点约束定义为$P_Cstr(\tau)=t$，即$\tau=t$.

定义4. 持续时间约束.限制任务执行持续时间，或者任务开始前等待时间.设τ为一项时间变量，一项持续时间约束$D_Cstr(\tau)$定义为以t_l为下边界，以t_u为上边界的时间区间，具体分为以下4种形式：

$D\_Cstr(\tau)=[t_l,t_u]$，即$t_l\leq\tau\leq t_u$；

$D\_Cstr(\tau)=(t_l,t_u]$，即$t_l<\tau\leq t_u$；

$D\_Cstr(\tau)=[t_l,t_u)$，即$t_l\leq\tau<t_u$；

$D\_Cstr(\tau)=(t_l,t_u)$，即$t_l<\tau<t_u$.

定义5. 周期时间约束.限制任务重复执行的时间间隔.设τ为一项时间变量，t为执行周期，一项周期时间约束定义为$C_Cstr(\tau)=t$，即$Mod(\tau,t)=0$.

在门禁控制软件中，DC-TR1为时间点约束，DC-TR2/DC-TR3等为持续时间约束，DC-TR6为周期时间约束.$\tau\vdash Cstr(\tau)$表示时间变量τ的取值满足时间约束$Cstr(\tau)$.此外，通过逻辑运算符连接不同类型时间约束，可得到各种复杂时间约束表达式.

对ESIA模型进行时间变量、置零行为以及时间约束等扩展如下.

定义6. ESIA时序语义.$P=(V_P,V_P^{on},X_P,A_P,\Psi_P,E,\Delta_P)$为一个ESIA，其时序语义描述了一个基于接口事件与时间约束的状态迁移系统：

X_P^T是一个有限时间变量集合，$X_P=X_P^R\cup X_P^H\cup X_P^T$表示全部变量的集合；其中，$\forall\tau\in X_P^T$为一项时间变量，在ESIA模型空间内可重复置零并启动计时，以描述对不同行为的时间约束；

$\forall\delta\in\Delta_P$表示一个状态迁移过程，具体描述如下：

$$\delta=(q)\xrightarrow[a]{e,PreCon(a),reset(),eff(a),PostCon(a),reset()}(q')；$$

其中，行为a的前置条件$PreCon(a)$包含行为a被激活时需要满足的时间约束，而后置条件

$PostCon(a)$包含行为 a 正常结束时需要满足的时间约束. $PreCon(a)$与 $PostCon(a)$均可包含多项时间约束,不同时间约束之间以逻辑运算符相连接,构成 $PreCon(a)$与 $PostCon(a)$的一阶逻辑表达式. 由此,在行为的前置和后置条件中整合各类时间约束,对其执行的时间条件进行统一描述. 设 X^T为被测软件 ESIA 模型的时间变量集合;X_1^T为行为 a 激活时完成计时的时间变量集合,$X_1^T \subseteq X^T$;X_2^T为行为 a 完成时完成计时的时间变量集合,$X_2^T \subseteq X^T$;行为 a 的前置和后置条件分别表示为:

$$PreCon(a)=$$
$$Cstr(\tau_{pre-1}) \wedge \cdots \wedge Cstr(\tau_{pre-n}), \tau_{pre-i} \in X_1^T;$$
$$PostCon(a)=$$
$$Cstr(\tau_{post-1}) \wedge \cdots \wedge Cstr(\tau_{post-m}), \tau_{post-j} \in X_2^T.$$

通常情况下,嵌入式软件具有异常处理机制,以对各类运行时序异常进行处理. 在 ESIA 模型中,对时序异常状态进行标记,以与正常状态区别描述. 设 P 为一个 ESIA 模型,定义异常状态标识 $v.E\_tag$,$v \in V_P$,当 $v.E\_tag$ 值为 1 时,表示 v 为异常状态;当值为 0 时,表示正常状态. 此外,$action\_E$ 表示行为 $action$ 在异常时序下的执行分支. 在 $action\_E$ 作用下,软件运行迁移至时序异常状态,执行相应异常处理任务. 在 ESIA 模型中,$action\_E$ 具有与 $action$ 相同的触发事件,但执行效果以及对应状态迁移的目标状态均不同. $action\_E$ 的前置和后置条件为对 $action$ 前置和后置条件中包含的变量取值约束以及时间约束取反(取反运算符为:¬),即若 $Cstr(\tau_1)$与 $Cstr(\tau_2)$分别为行为 $action$ 的前置与后置条件,则 $\neg Cstr(\tau_1)$与 $\neg Cstr(\tau_2)$分别为异常时序分支 $action\_E$ 的前置与后置条件.

具有时间约束的门禁控制软件 ESIA 模型如图 2 所示. 其中,针对不同定时器分别定义时间变量 τ_0、τ_1及 τ_2,其计时过程从置零开始,到时间约束作用点结束. 每项时间变量记录了特定状态迁移序列的执行时间. 如图 2 所示,当检测到人员抵近后,置零 τ_1与 τ_2,τ_1记录门禁控制软件沿状态迁移序列 $State2 \rightarrow State3$ 执行所耗费的时间,而 τ_2记录沿 $State2 \rightarrow State3 \rightarrow State4$ 执行所耗费的时间;当检测到人员离开后,再次置零 τ_1,并记录沿状态迁移序列 $State4 \rightarrow State1$ 执行所耗费的时间. 设定门禁控制软件的时间约束如下:

DC-TR1. 系统上电后置零τ_0,定义$P_Cstr(\tau_0)=5\ s(\tau_0=5\ s)$为行为 *initialization*;的后置条件;

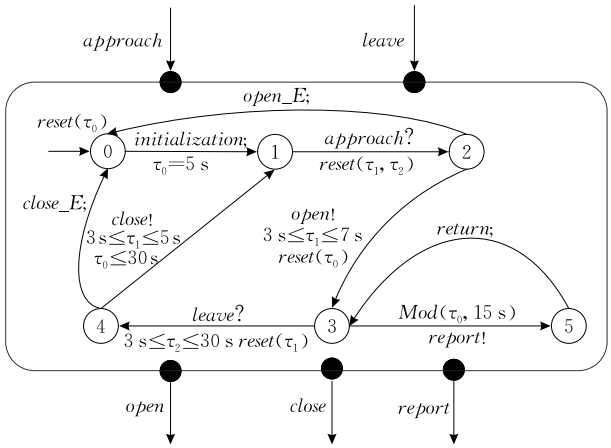


图 2　具有时间约束的门禁控制软件 ESIA 模型

DC-TR2. 当检测到人员抵近事件后置零 τ_1(状态 2),定义 $D_Cstr_1(\tau_1)=[3\ s, 7\ s]$为行为 *open*! 的后置条件;

DC-TR3. 当检测到人员离开事件后置零 τ_1(状态 4),定义 $D_Cstr_2(\tau_1)=[3\ s, 5\ s]$为行为 *close*! 的后置条件;

DC-TR4. 当检测到人员抵近事件后置零 τ_2(状态 2),定义 $D_Cstr(\tau_2)=[3\ s, 30\ s]$为行为 *leave*? 的后置条件;

DC-TR5. 在自动门打开后置零 τ_0(状态 3),定义 $D_Cstr(\tau_0)=[0\ s, 30\ s]$为行为 *close*! 的后置条件;

DC-TR6. 在自动门打开后置零 τ_0(状态 3),定义 $C_Cstr(\tau_0)=15(Mod(\tau_0, 15)=0)$为行为 *report*! 的前置条件.

此外,门禁系统具有一组异常时序处理机制. 在运行过程中,当检测到人员抵近事件后,若打开门禁的时间超出 DC-TR2 的要求(满足¬DC-TR2),则执行异常分支 *open_E*,门禁系统重新初始化;当检测到人员抵近事件后,若关闭门禁的时间超出 DC-TR3 的要求(满足¬DC-TR3),则执行异常分支 *close_E*,门禁系统同样重新初始化.

在本文中,认为目标软件的 ESIA 模型具有确定性与完整性. 其中,确定性表示在相同的源状态、触发事件与时序条件下,ESIA 具有唯一的后继行为;完整性表示目标系统具有的所有状态、行为和执行路径等,均在 ESIA 模型中予以描述,所有正确的执行片段均可映射至相应 ESIA 执行路径.

3.3　时间约束相关关系分析

在嵌入式软件中,一项任务的执行时间可能与其他任务相关. 当不同任务的执行路径相互重叠时,

例如一项任务是另一项任务的子任务或两项任务包含公共子任务时,不同任务的执行时间之间将存在特定依赖关系,造成相应时间约束之间间接相关.在时序检测中,不仅需检验各项任务的执行时间是否满足相应独立时间约束,还需检验任务的执行时间是否满足特定依赖关系,从而对软件运行时序进行充分检测.

如图 2 中,当检测到人员抵近事件后,时间变量 τ_1 与 τ_2 被同时置零并开始计时.由于与 τ_2 关联的执行路径包含与 τ_1 关联的执行路径,τ_1 与 τ_2 之间存在取值依赖关系:$\tau_1<\tau_2$.若任务的执行时间违反以上依赖关系,则表示软件运行时序存在异常.

在 ESIA 模型中,使用 ESIA 执行路径 $Path(\tau)$ 表示与时间变量 τ 相关联的状态迁移序列,并对不同时间约束间的相关关系进行描述.

定义 7. ESIA 执行路径. $P=(V_P, V_P^{on}, X_P, A_P, \Psi_P, E, \Delta_P)$ 为一个 ESIA,执行路径 $Path(\tau)$ 定义为 P 之上的一个有序状态迁移子集:

$Path(\tau)=\{\delta_i \mid i=1,2,\cdots,m\}$, $Path(\tau)\subseteq\Delta_P$, τ 在 δ_1 处置零,$Cstr(\tau)$ 作用于 δ_m;

$\forall\delta_i,\delta_{i+1}\in Path(\tau)$, $1\leq i\leq m-1$, δ_i 与 δ_{i+1} 为 ESIA 上两项相继的状态迁移,即 $\delta_i.t=\delta_{i+1}.s$.

如图 2 所示,$Path(\tau_1)=\{State2\rightarrow State3\}$ 与 $Path(\tau_2)=\{State2\rightarrow State3, State3\rightarrow State4\}$ 即为门禁控制软件 ESIA 模型上的两条执行路径.

定义 8. 时间约束相关关系. $\tau_1、\tau_2\in X^T$ 为 ESIA 上的两个时间变量,$Path(\tau_1)$ 与 $Path(\tau_2)$ 为与 $\tau_1、\tau_2$ 关联的执行路径,$Cstr(\tau_1)$ 与 $Cstr(\tau_2)$ 之间存在以下 3 类相关关系:

(1) 相互独立(Isolated, IR). 若 $Path(\tau_1)\cap Path(\tau_2)=\varnothing$,则 $Cstr(\tau_1)$ 与 $Cstr(\tau_2)$ 相互独立;

(2) 包含相关(Containment, CR). 若 $Path(\tau_1)\subseteq Path(\tau_2)$,则 τ_1 与 τ_2 之间存在取值依赖关系 $C\_Dep(\tau_1,\tau_2):\tau_1<\tau_2$, $Cstr(\tau_1)$ 与 $Cstr(\tau_2)$ 包含相关;

(3) 交叉相关(Overlapping, OR). 若 τ_1 与 τ_2 在不同时间点置零,$\exists Path(\tau_3)\subseteq\Delta_P$, $Path(\tau_3)=Path(\tau_1)\cap Path(\tau_2)\neq\varnothing$,则 $\tau_1、\tau_2$ 与 τ_3 之间存在取值依赖关系 $O\_Dep(\tau_1,\tau_2,\tau_3):(\tau_3<\tau_1)\wedge(\tau_3<\tau_2)$, $Cstr(\tau_1)$ 与 $Cstr(\tau_2)$ 交叉相关;

其中,若 $Cstr(\tau_1)$ 与 $Cstr(\tau_2)$ 相关,则称 τ_1 与 τ_2 为一组相关时间变量.

在门禁控制软件 ESIA 模型中,时间约束 $D\_Cstr(\tau_2)$ 与 $D\_Cstr_2(\tau_1)$ 相互独立(图 3(a));时间约束 $D\_Cstr_1(\tau_1)$ 与 $D\_Cstr(\tau_2)$ 包含相关(图 3(b));时间约束 $D\_Cstr(\tau_2)$ 与 $D\_Cstr(\tau_0)$ 交叉相关(图 3(c)).对相关关系的定义可扩展到多个时间变量.如图 3(d) 所示,若 $\exists\delta\in\Delta_P$,且 $\delta\in\bigcap_{i=0}^{k}Path(\tau_i)$,则 k 个时间变量 $\{\tau_i \mid i=1,2,\cdots,k\}$ 之间存在相关关系.本文主要对两项时间变量相关时的时序检测方法进行讨论.当多项时间约束之间存在多重相关关系时,不同时间变量之间的取值依赖关系将更加复杂.

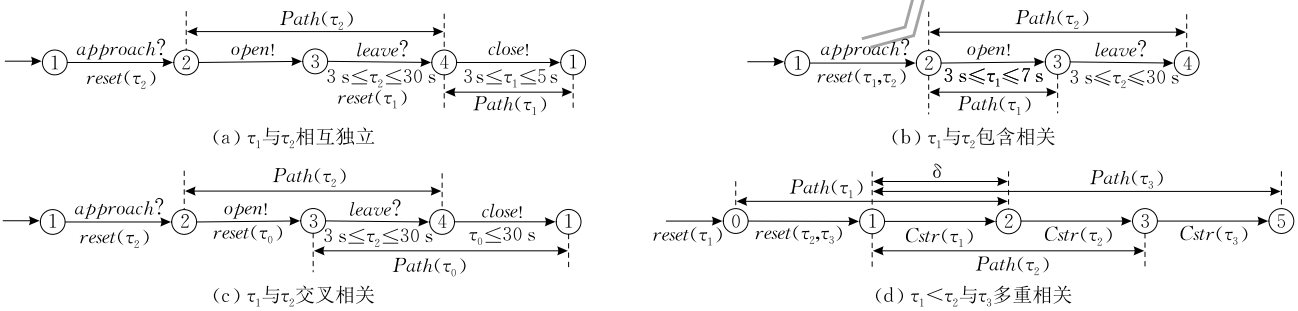


图 3 时间约束相关关系

4 基于相关性分析的时序缺陷检测

在时序缺陷检测过程中,根据目标软件 ESIA 模型描述与执行路径上的时间约束相关性分析结果,对预先提取的包含时间信息的执行片段进行离线自动检测,以检查软件在运行过程中是否存在时序异常,并检查软件的异常时序处理机制是否满足设计要求.本文将包含时间信息的执行片段定义为一个行为执行序列,以描述系列任务的执行过程.

定义 9. 包含时间信息的执行片段. 包含时间信息的执行片段 $Trace$ 定义为一个行为执行序列:

(1) $Trace=\{\lambda_j \mid j=1,2,\cdots,n\}$, λ_j 为第 j 项行为的执行过程;

(2) $\forall\lambda_j$ 表示为一个六元组 $\lambda_j=(e,\tau_{j1}=v_{j1},$

$reset(\tau_{j3}),a,\tau_{j2}=v_{j2},reset(\tau_{j4}))$；其中，$\lambda_j.e$ 为触发行为执行的事件，$\lambda_j.a$ 为相应行为；$\tau_{j1},\tau_{j2},\tau_{j3},\tau_{j4}\in X_P^T$ 为四项时间变量，τ_{j1} 在行为 $\lambda_j.a$ 开始时完成计时，计时时间为 v_{j1}；τ_{j2} 在行为 $\lambda_j.a$ 结束时完成计时，计时时间为 v_{j2}；τ_{j3} 在行为 $\lambda_j.a$ 开始执行时置零并启动计时；τ_{j4} 在行为 $\lambda_j.a$ 执行结束时置零并启动计时.

在执行片段提取过程中，若 λ_j 中的某项信息不存在，则相应元素空缺. 若一项执行片段不包含时间信息，则无需描述与其相关联的时间变量、任务执行时间与置零行为. 在 TBTDD 方法中，执行片段的记录应规范和完整，以便后续解析与检测.

异常时序处理机制是嵌入式软件可靠性设计与验证的重要内容. 在 TBTDD 方法中，不仅需要对软件正常运行时序进行检测，还需要对软件异常运行时序处理机制进行检测，以避免产生非预期的结果. 本文将执行片段分为正常时序片段与异常时序片段，定义如下：

(1) 正常时序片段：当一条执行片段仅包含正常时序与相应执行分支时，该片段为正常片段；

(2) 异常时序片段：当一条执行片段包含异常时序与相应异常处理分支时，该片段为异常片段.

执行片段的提取应尽可能覆盖各类正常和异常运行时序，以对目标软件时序特性进行充分检测.

TBTDD 方法工作流程如图 4 所示.

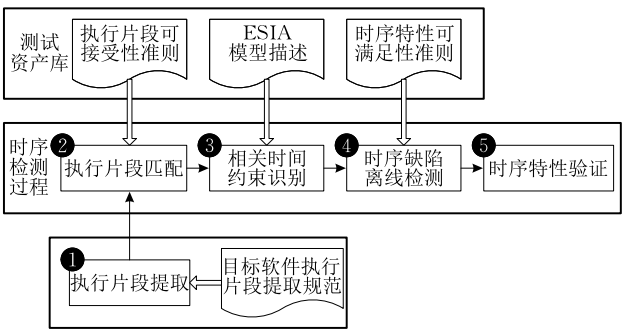


图 4　TBTDD 方法工作流程

(1) 执行片段提取与预处理. 在目标软件运行环境中，通过在目标机、周边设备以及仿真设备上设置观察点或探针，实时记录软件运行过程中的触发事件、任务执行序列、执行时间及启动计时等时间信息，按照记录规范对其进行处理后，组织构建执行片段并存入软件运行日志.

(2) 执行片段匹配. 依据执行片段可接受性准则，为每条执行片段选取匹配的 ESIA 执行路径；针对无法匹配的执行片段，报告软件运行异常，并判断缺陷类型.

(3) 相关时间约束识别. 在每条匹配的 ESIA 执行路径上，识别各项独立和相关时间约束，提取不同时间变量间的取值依赖关系.

(4) 时序缺陷检测. 依据时序特性可满足性准则，检查执行片段包含的时间信息是否满足匹配路径上各项独立和相关时间约束的要求，检查软件异常时序处理机制是否满足设计要求，检测软件缺陷并识别缺陷类型.

(5) 时序特性验证. 根据时序特性检测结果，验证目标软件时序特性是否满足设计要求.

4.1　执行片段匹配

若执行片段包含的行为序列与特定 ESIA 执行路径相匹配，则称该执行片段可被目标软件 ESIA 模型接受. 本节制定执行片段可接受性准则，依据该准则，提出一种广度优先候选状态算法，对特定执行片段上的每项行为进行逐项匹配，在目标软件 ESIA 模型之上搜索与其相匹配的 ESIA 执行路径，并滤除 ESIA 无法接受的执行片段. 由于目标软件 ESIA 模型包含各类正常和异常时序分支，以下执行片段可接受性准则与匹配算法同时适用于正常时序片段与异常时序片段的匹配过程.

执行片段可接受性准则定义如下.

定义 10.　执行片段可接受性. $P=(V_P,V_P^{on},X_P,A_P,\Psi_P,E,\Delta_P)$ 为一个 ESIA，执行片段 $Trace=\{\lambda_j\mid j=1,2,\cdots,n\}$ 可被 P 接受，当且仅当：

(1) $\forall\lambda_j=(e,\tau_{j1}=v_{j1},reset(\tau_{j3}),a,\tau_{j2}=v_{j2},reset(\tau_{j4}))$，$\exists\delta_i\in\Delta_P,\lambda_j.e=\delta_i.e$ 且 $\lambda_j.a=\delta_i.a$；当满足以上条件时，称 λ_j 与 δ_i 之间相匹配，记为 $\delta_i=map(\lambda_j)$；

(2) $\forall\lambda_j,\lambda_{j+1}\in Trace,1\le j\le n-1$，$\exists\delta_i,\delta_{i+1}\in\Delta_P,\delta_i=map(\lambda_j)$ 且 $\delta_{i+1}=map(\lambda_{j+1}),\delta_i.t=\delta_{i+1}.s$，即 δ_i 与 δ_{i+1} 为 ESIA 上两项相继的状态迁移.

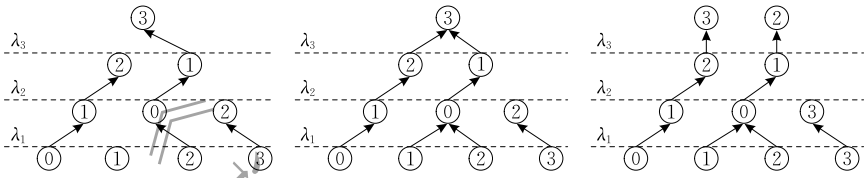
设 $Trace=\{\lambda_j\mid j=1,2,\cdots,n\}$ 为一条执行片段，$V_P=\{0,1,\cdots,m\}$ 为被测软件 ESIA 模型 P 的初始候选状态集合，基于可接受性准则的执行片段匹配过程包含以下步骤：

(1) 从 λ_1 开始，以 V_P 为初始候选状态集合，启动 $Trace$ 的匹配过程.

(2) 在对 λ_j 进行匹配的过程中，以一项当前候选状态为源状态，若在其上存在与 λ_j 相匹配的状态迁移，则记录该状态迁移与对应目标状态；在所有当前候选状态之上执行以上匹配过程，并利用记录的目标状态集合更新当前候选状态集合，λ_j 匹配过程完成；

(3)沿执行片段 $Trace$ 持续执行步骤“(2)”,若 $Trace$ 上的所有行为执行均正确匹配,则搜索到一组与 $Trace$ 相匹配的 ESIA 执行路径;若特定行为无法匹配,则匹配过程终止.

设 $Trace=\{\lambda_1,\lambda_2,\lambda_3\}$ 为一条执行片段,$V_P=\{0,1,2,3\}$ 为被测软件 ESIA 模型的初始候选状态集合,$Trace$ 的匹配过程如图 5(a)所示.其中,λ_j 对应的虚线表示对 $Trace$ 上的第 j 项行为进行匹配的过程,λ_j 对应的虚线下的一组状态表示当前候选状态集合,将在其上搜索与 λ_j 相匹配的状态迁移.首先,对 λ_1 进行匹配,初始候选状态集合为{0,1,2,3},若在候选状态 0、2、3 之上存在与 λ_1 相匹配的状态迁移,且对应目标状态为 1、0、2,则记录相应状态迁移“$State0\rightarrow State1$”、“$State2\rightarrow State0$”、“$State3\rightarrow State2$”,更新当前候选状态集合为{1,0,2};随后,对 λ_2 进行匹配,若在候选状态 1、0 之上存在与 λ_2 相匹配的状态迁移,且对应目标状态为 2、1,则记录相应状态迁移“$State1\rightarrow State2$”、“$State0\rightarrow State1$”,更新当前候选状态集合为{2,1};最后,对 λ_3 进行匹配,若在候选状态 1 之上存在与 λ_3 相匹配的状态迁移,且对应目标状态为 3,则记录相应状态迁移“$State1\rightarrow State3$”,执行片段匹配完成,搜索得到的匹配路径为{$State2\rightarrow State0$,$State0\rightarrow State1$,$State1\rightarrow State3$}.



(a)单一匹配路径 (b)多条匹配路径且终点相同 (c)多条匹配路径且终点不同

图 5 执行片段匹配过程

在一条执行片段的匹配过程中,由于一项行为可能与多个候选状态上的特定状态迁移相匹配,因而可能存在多条与其相匹配的 ESIA 执行路径.如图 5(b)、(c)所示,在被测软件 ESIA 模型之上均存在多条与 $Trace=\{\lambda_1,\lambda_2,\lambda_3\}$ 相匹配的执行路径.

例如在图 2 所示的门禁控制软件中,提取执行片段 $Trace=\{(man\ approach,,,approach?,,reset(\tau_1,\tau_2)),(,,,open!,D\_Cstr_1(\tau_1),reset(\tau_0)),(man\ leave,,,leave?,D\_Cstr(\tau_2),reset(\tau_1)),(,,,close!,D\_Cstr_2(\tau_1)\wedge D\_Cstr(\tau_0),)\}$.通过执行片段匹配,在门禁控制软件 ESIA 模型中搜索得到的匹配路径为 $Path=\{State1\rightarrow State2,State2\rightarrow State3,State3\rightarrow State4,State4\rightarrow State1\}$.

当存在多条与执行片段相匹配的 ESIA 执行路径时,若不同的匹配路径具有不同的时间约束,则无法唯一确定执行片段应满足的时序设计要求.在此情况下,针对所有匹配路径进行相关时间约束识别与时序缺陷检测,若执行片段包含的时间信息无法满足所有匹配路径的时序要求,则软件运行时序异常,需对其作进一步分析.此外,若不存在与执行片段相匹配的 ESIA 执行路径,则将该执行片段过滤,不再对其进行时序缺陷检测.在此情况下,软件实现可能具有 ESIA 模型未描述的运行场景,或可能存在功能性缺陷,需对提取的执行片段与相应功能设计要求进行进一步核查,但不能确定软件实现是否存在时序缺陷.

执行片段匹配算法具体定义如下.

算法 1. 执行片段匹配 $Trace\_Match(\ )$.

输入:A $Trace=\{\lambda_j\mid j=1,2,\cdots,n\}$ and ESIA P

输出:A set of matched execution path($SMEP$)

数据结构:Current candidate states(CCS)

```
Trace_Match(Trace,P)
BEGIN
  CCS=V_P
  FOR each q in CCS //初始化 SMEP
    Create an execution path for state q
  END FOR
  FOR each λ on Trace //检测每一项行为执行
    λ_pre is the precursor action of λ on Trace
    δ_pre=map(λ_pre),δ_pre∈Δ_P
    FOR each q in CCS //检测每一项候选状态
      IF(∃δ∈Δ_P,δ=map(λ) and δ.s=δ_pre.t)
        Add δ to corresponding path in SMEP
        Replace q by q' in CCS
      ELSE
        Remove corresponding path from SMEP
        Remove q from CCS
    END FOR
    Remove redundant states from CCS
    IF(CCS=∅) //执行片段不可接受
      Return No-Matching
    ELSE
```

```
        Null
    END FOR
    Return SMEP
END
```

在以上算法中，设 n 为执行片段长度，m 为候选状态集合包含的状态数，则算法复杂度为 $O(n\times m)$. 执行片段长度、ESIA 模型规模以及特定行为在 ESIA 模型中的出现频度等因素都将影响匹配结果与效率. 因此，通过制定执行片段提取规范，控制执行片段的起点、终点、结构与长度，可改善算法的执行效率.

4.1.1　模型覆盖率

嵌入式软件在不同外部事件触发下，往往具有大量执行序列，难以进行完全检测. 本文利用匹配路径集对 ESIA 模型的覆盖率衡量时序特性检测的充分性.

本文采用的 ESIA 模型覆盖准则如下：

(1) 状态覆盖. 执行路径集应覆盖 ESIA 中的所有状态；设 n_s 为 ESIA 总状态数，m_s 为执行路径集覆盖的状态数，状态覆盖率 $StatesCvr$ 计算如下：

$$StatesCvr=\frac{m_s}{n_s}\times 100\% \tag{1}$$

(2) 迁移覆盖. 执行路径集应覆盖 ESIA 中的所有状态迁移；设 n_t 为 ESIA 总状态迁移数，m_t 为执行路径集覆盖的状态迁移数，状态迁移覆盖率 $TransitionsCvr$ 计算如下：

$$TransitionsCvr=\frac{m_t}{n_t}\times 100\% \tag{2}$$

(3) 循环覆盖. 在任意一条执行路径之上，对 ESIA 中每一项循环结构的遍历不超过 n 次(All-Loop-n)，且路径包含的循环结构数不超过 m(All-m-Loop)；执行路径集应覆盖 ESIA 中所有满足以上要求的路径；设 n_l 为 ESIA 总循环结构数，m_l 为执行路径集覆盖的循环结构数，循环覆盖率 $LoopsCvr$ 计算如下：

$$LoopsCvr=\frac{m_l}{n_l}\times 100\% \tag{3}$$

(4) 独立时间约束覆盖. 执行路径集应覆盖 ESIA 中的所有独立时间约束；设 n_{itc} 为 ESIA 总独立时间约束数，m_{itc} 为执行路径集覆盖的独立时间约束数，独立时间约束覆盖率 $ITConstraintsCvr$ 计算如下：

$$ITConstraintsCvr=\frac{m_{itc}}{n_{itc}}\times 100\% \tag{4}$$

(5) 相关时间约束覆盖. 执行路径集应覆盖 ESIA 中的所有相关时间约束；设 n_{rtc} 为 ESIA 总相关时间约束数，m_{rtc} 为执行路径集覆盖的相关时间约束数，相关时间约束覆盖率 $RTConstraintsCvr$ 计算如下：

$$RTConstraintsCvr=\frac{m_{rtc}}{n_{rtc}}\times 100\% \tag{5}$$

当匹配路径集满足以上覆盖准则时，认为执行片段提取以及时序缺陷检测工作已足够充分.

4.2　相关时间约束识别

为检验执行片段包含的时间信息是否满足各项独立以及相关时间约束要求，需在每条匹配的 ESIA 执行路径之上，识别其包含的相关时间约束. 每项相关时间约束定义为一个三元组 $R\_Cstr=(X^T,\Psi^T,type)$，其中，X^T 表示一组时间变量；Ψ^T 表示 X^T 中的时间变量具有的一组时间约束；$type$ 表示 Ψ^T 中的时间约束具有的相关关系类型，可以是包含相关(CR)以及交叉相关(OR)等.

时间约束限制特定任务的执行时间. 在嵌入式软件中，通过时间变量的计时过程记录特定任务的执行时间，其计时过程从任务启动时开始，到任务完成或异常终止时结束. 因此，一项时间变量 τ 的计时区间与关联任务执行路径 $Path(\tau)$ 之间完全重合. 在相关时间约束识别过程中，通过分析不同时间变量计时区间的重叠情况，判断相应任务执行路径之间的重叠方式，进而根据前述相关关系定义，确定不同时间变量之间存在的取值依赖关系，识别相应时间约束之间存在的相关关系.

本文采用时间约束相关矩阵进行时间约束间的相关关系判断. 通过遍历特定匹配路径，搜索不同时间变量的计时起点和计时终点(时间约束作用点)，动态构建与更新相关矩阵. 在时间变量的计时起点，将其添加至相关矩阵；在时间变量的计时终点，将其从相关矩阵中移除，并对相应时间约束的相关性进行分析与判断，具体判断准则如下：

(1) 若一项时间变量所属列具有前列，或者同列中包含多项时间变量，则相应时间约束之间包含相关；

(2) 若一项时间变量所属列具有后列，则相应时间约束之间交叉相关；

(3) 若两项时间变量同时结束计时，则相应时间约束之间包含相关.

设 $Path=\{\delta_1,\delta_2,\delta_3,\delta_4\}$ 为一条 ESIA 执行路

径，τ_1、τ_2、τ_3和τ_4为与 *Path* 相关联的四项时间变量. 其中，τ_1在$\delta_1.a$开始时置零，τ_2、τ_3在$\delta_1.a$结束时置零，τ_4在$\delta_2.a$开始时置零；$Cstr(\tau_3)\in\delta_2.PostCon$，$Cstr(\tau_1)\in\delta_3.PreCon$，$Cstr(\tau_2)\in\delta_3.PostCon$，$Cstr(\tau_4)\in\delta_4.PostCon$. *Path* 之上的相关矩阵动态构建与相关时间约束识别过程如下所示：

$$(\tau_1) \tag{1.a}$$

$$\Rightarrow\begin{pmatrix}\tau_1 & \tau_2\\ — & \tau_3\end{pmatrix} \tag{1.b}$$

$$\Rightarrow\begin{pmatrix}\tau_1 & \tau_2 & \tau_4\\ — & \tau_3 & —\end{pmatrix} \tag{1.c}$$

$$\Rightarrow(\tau_1 \quad \tau_2 \quad \tau_4) \tag{1.d}$$

$$\Rightarrow(\tau_2 \quad \tau_4) \tag{1.e}$$

$$\Rightarrow(\tau_4) \tag{1.f}$$

当沿路径 *Path* 遍历至$\delta_1.a$开始位置时，时间变量τ_1开始计时，在时间约束相关矩阵中新建一列，将τ_1添加至该列(步骤 1.a)；在$\delta_1.a$结束位置，时间变量τ_2、τ_3开始计时，将τ_2、τ_3添加至相关矩阵新建列(步骤 1.b)；在$\delta_2.a$开始位置，时间变量τ_4开始计时，将τ_4添加至相关矩阵新建列(步骤 1.c)；在$\delta_2.a$结束位置，时间变量τ_3结束计时(即$Cstr(\tau_3)$作用点)，对$Cstr(\tau_3)$与其他时间约束之间的相关性进行判断，识别出$Cstr(\tau_3)$分别与$Cstr(\tau_1)$、$Cstr(\tau_2)$包含相关，与$Cstr(\tau_4)$交叉相关，并将τ_3从相关矩阵中移除(步骤 1.d)；继续执行后续步骤(步骤 1.e 与 1.f)，识别出$Cstr(\tau_1)$分别与$Cstr(\tau_2)$、$Cstr(\tau_4)$交叉相关，$Cstr(\tau_2)$与$Cstr(\tau_4)$交叉相关.

例如在图 2 所示的门禁控制软件 ESIA 模型中，$Path=\{State1\rightarrow State2, State2\rightarrow State3, State3\rightarrow State4, State4\rightarrow State1\}$为一条匹配路径，$\tau_0$、$\tau_1$、$\tau_2$为与 *Path* 相关联的三项时间变量. *Path* 之上的相关矩阵动态构建与相关时间约束识别过程如下所示：

$$\begin{pmatrix}\tau_1\\ \tau_2\end{pmatrix} \tag{2.a}$$

$$\Rightarrow(\tau_2 \quad \tau_0) \tag{2.b}$$

$$\Rightarrow(\tau_0 \quad \tau_1) \tag{2.c}$$

当沿路径 *Path* 遍历至 *approach* 结束位置时，时间变量τ_1、τ_2开始计时，在时间约束相关矩阵中新建一列，将τ_1、τ_2添加至该列(步骤 2.a)；在 *open* 结束位置，时间变量τ_1结束计时($D_Cstr_1(\tau_1)$作用点)，识别出$D_Cstr_1(\tau_1)$与$D_Cstr(\tau_2)$包含相关，并将τ_1从相关矩阵中移除，时间变量τ_0开始计时，将τ_0添加至相关矩阵新建列(步骤 2.b)；在 *leave* 结束位置，时间变量τ_2结束计时($D_Cstr(\tau_2)$作用点)，识别出$D_Cstr(\tau_2)$与$D_Cstr(\tau_0)$交叉相关，并将τ_2从相关矩阵中移除，时间变量τ_1重新开始计时，将τ_1添加至相关矩阵新建列(步骤 2.c)；在 *close* 结束位置，时间变量τ_0、τ_1结束计时($D\_Cstr(\tau_0)$、$D_Cstr_2(\tau_1)$作用点)，识别出$D_Cstr(\tau_0)$与$D_Cstr_2(\tau_1)$包含相关，并将τ_0、τ_1从相关矩阵中移除.

由此，在门禁控制软件的匹配路径 *Path* 之上，识别出的相关时间约束如下：

$R\_Cstr_1=(\{\tau_1,\tau_2\},\{D\_Cstr_1(\tau_1),D\_Cstr(\tau_2)\},\mathrm{CR})$；

$R\_Cstr_2=(\{\tau_2,\tau_0\},\{D\_Cstr(\tau_2),D\_Cstr(\tau_0)\},\mathrm{OR})$；

$R\_Cstr_3=(\{\tau_0,\tau_1\},\{D\_Cstr(\tau_0),D\_Cstr_2(\tau_1)\},\mathrm{CR})$.

相关时间约束识别算法具体定义如下.

算法 2. 相关时间约束识别 *Correlation_Identify*().

```
输入：An ESIA Path={δ_i | i=1,2,…,m}
输出：A set of correlated timing constraints(SCTC)
数据结构：Timing constraints matrix (TCM)

Correlation_Identify(Path)
BEGIN
  FOR each δ on Path
    IF(∃τ reset at the beginning of δ)
      Create a column c in TCM
      FOR each τ reset at the beginning of δ
        Add τ to c
    IF(∃Cstr(τ)∈δ.PreCon)
      Identify(δ.PreCon) //识别相关时间约束
    IF(∃τ reset at the end of δ)
      Create a column c in TCM
      FOR each τ reset at the end of δ_i
        Add τ to c
    IF(∃Cstr(τ)∈δ.PostCon)
      Identify(δ.PostCon) //识别相关时间约束
  END FOR
END

Identify(A set of timing constraints(STC))
BEGIN
  FOR each Cstr(τ) in STC
    Get the column number cn of τ in TCM
    Get the line number ln of τ in TCM
    IF(∃R_Cstr∈SCTC and τ∈R_Cstr.X^T)
                          //已识别的相关时间约束
      Add Cstr(τ) to R_Cstr.Ψ^T
    END IF
```

```
    IF(0<cn) //所属列具有前列
      FOR each τ_pre in pre-columns in TCM
        Create R_Cstr, R_Cstr.X^T={τ_pre,τ},
               R_Cstr.type=CR
        add R_Cstr to SCTC
    END IF
    IF(0<ln) //同列中包含多项时间变量
      FOR each τ_pre in same-column in TCM
        Create R_Cstr, R_Cstr.X^T={τ_pre,τ},
               R_Cstr.type=CR
        add R_Cstr to SCTC
    END IF
    IF(cn+1<the number of column in TCM)
                                  //所属列具有后列
      FOR each τ_post in post-columns in TCM
        Create R_Cstr, R_Cstr.X^T={τ,τ_post},
               R_Cstr.type=OR
        add R_Cstr to SCTC
    END IF
    Remove τ from TCM
  END FOR
END
```

在以上算法中，设 n 为执行路径长度，m 为特定行为前置和后置条件中包含的时间约束数，l_1、l_2 与 l_3 分别为相关矩阵中特定时间变量前列、同列与后列包含的时间变量数，则算法复杂度为 $O(n\times m\times(l_1+l_2+l_3))$.

4.3　时序缺陷分析

嵌入式软件故障管理主要包括五个方面，分别是故障检测、故障定位、故障隔离、系统重构以及故障修复等[7-8]. 本文主要关注对目标软件中存在的单一时序故障进行检测、识别与定位. 当目标软件中同时存在多重时序故障时，时序缺陷的检测和定位将更加复杂，且由于不同故障间的相互作用，软件运行过程中的时序异常可能被掩盖.

本节对嵌入式软件运行过程中可能出现的功能和时序异常进行分析，确定执行片段中可能存在的时序缺陷类型，作为制定时序特性检测准则的基础，有针对性的指导时序异常检测和缺陷识别工作.

在目标软件 ESIA 模型之上，若不存在与执行片段相匹配的 ESIA 执行路径，则软件实现存在功能性缺陷. Miller、Alcalde 等人[9-10]将执行序列检测过程中出现的功能异常分为输出故障与状态迁移故障. 其中，输出故障描述在特定源状态与行为作用下，软件运行产生不符合预期的输出；状态迁移故障描述在特定源状态与行为作用下，软件运行迁移至不符合预期的目标状态. 在本文中，根据执行片段可接受性准则，软件运行过程中可能存在的功能异常如下：

(1) 行为错配(违反准则定义 10.1). 针对执行片段上的特定行为，无法在 ESIA 模型之上搜索到匹配的状态迁移，具体包括触发事件匹配错误和行为执行匹配错误等.

(2) 行为错序(违反准则定义 10.2). 针对执行片段上的两项相继行为，无法在 ESIA 模型之上搜索到匹配的相继状态迁移，具体包括两项状态迁移的前驱/后继关系错误，两项状态迁移不是相继的状态迁移等.

在 TBTDD 方法中，若不存在与执行片段相匹配的 ESIA 执行路径，则识别并报告软件存在的功能性缺陷，并通过进一步核查确定该缺陷属于软件实现缺陷或设计缺陷.

嵌入式软件运行过程中可能存在的时序异常如下：

(1) 违反独立时间约束. 一项时间变量的取值违反特定时间约束，或在匹配的状态迁移之上未搜索到对应的时间约束.

(2) 违反相关时间约束. 一组时间变量的取值违反特定时间约束间相关关系.

(3) 违反计时起点要求. 特定时间变量的计时起点与设计要求不符.

(4) 异常时序处理错误. 针对特定异常时序的处理机制与设计要求不符.

其中，对于时间点约束，若预期事件在规定的时间点没有产生，则软件运行时序异常；对于持续时间约束，若任务执行时间低于下限阈值或高于上限阈值，则运行时序异常；而对于周期时间约束，若任务重复执行的时间间隔不满足周期时间要求，则运行时序异常.

时间约束间相关关系由执行路径上不同任务间的关联方式引起，是软件运行过程中必须遵循的物理规律. 在正确实现的目标软件中，不会存在违反相关时间约束的执行时间；而在错误实现的目标软件中，则可能存在此类异常时序. 在传统时序测试过程中，若针对此类异常时序设计测试用例，当软件实现正确时，其将为无效用例，无法在目标系统中配置与执行，或由于执行错误而产生错误的测试结果；若不设计此类用例，则难以发现违反相关关系的异常时序. 在 TBTDD 方法中，通过观察目标软件运行情况，对其中包含的时间信息进行检测，以验证软件运

行时序是否正确,从而避免了以上问题,为此类异常时序的检测提供了新的有效手段.

此外,不安全的异常时序处理方式同样会造成软件运行状态异常.在传统时序测试过程中,一般通过人工设置违反时间约束要求的运行时间,以观察软件时序异常处理机制是否满足设计要求.在此过程中,不仅难以完全覆盖软件运行过程中的各类异常时序,且难以模拟特定环境条件下的偶发异常时序.在 TBTDD 方法中,通过长期监控目标软件运行,提取充分的执行片段集合并进行检测,可有效发现此类影响目标软件运行可靠性的潜在缺陷,甚至侦测到多余的软件运行分支.

4.4 时序缺陷检测

在时序特性检测过程中,通常需要制定一组检测准则,当执行片段包含的时间信息满足检测准则时,即认为软件运行时序满足设计要求.在 TBTDD 方法中,根据时序缺陷类型分析,制定时序特性可满足性准则,并利用 ESIA 模型描述的时间需求作为检测度量,对执行片段包含的时间信息与异常时序处理机制进行检测.其中,通过对异常时序处理机制进行检测,可有效保证嵌入式软件运行的健壮性与可靠性.以下时序特性可满足性准则与时序缺陷检测算法同时适用于正常时序片段与异常时序片段的检测过程.

时序特性可满足性准则定义如下.

定义 11. 时序特性可满足性.P 为一个 ESIA,执行片段 $Trace=\{\lambda_j \mid j=1,2,\cdots,n\}$ 可被 P 接受,匹配路径为 $Path=\{\delta_i \mid i=1,2,\cdots,n\}$,执行片段包含的时间信息满足 ESIA 时序特性要求,当且仅当 $\forall\lambda_j=(e,\tau_{j1}=v_{j1},reset(\tau_{j3}),a,\tau_{j2}=v_{j2},reset(\tau_{j4}))$ 及 $\delta_i=map(\lambda_j)$:

(1) 对于 τ_{j1},$\exists Cstr(\tau_{j1})\in\delta_i.PreCon$,且 $v_{j1}\vdash Cstr(\tau_{i1})$;对 τ_{j2},$\exists Cstr(\tau_{j2})\in\delta_i.PostCon$,且 $v_{j2}\vdash Cstr(\tau_{j2})$;

(2) 若 $\exists R\_Cstr, R\_Cstr.X^T=\{\tau_{j1},\tau_x\},\tau_x\in X_P^T$,则 $v_{j1}\vdash Dep(\tau_{j1},\tau_x)$;若 $\exists R\_Cstr, R\_Cstr.X^T=\{\tau_{j2},\tau_y\},\tau_y\in X_P^T$,则 $v_{j2}\vdash Dep(\tau_{j2},\tau_y)$;若 $\exists R\_Cstr, R\_Cstr.X^T=\{\tau_{j1},\tau_{j2}\}$,则 $v_{j1}\vdash Dep(\tau_{j1},\tau_{j2})$ 且 $v_{j2}\vdash Dep(\tau_{j1},\tau_{j2})$;

(3) τ_{j3} 在行为 $\delta_i.a$ 开始时置零;τ_{j4} 在行为 $\delta_i.a$ 结束时置零.

在时序缺陷检测过程中,依据可满足性准则,对执行片段上的各项行为进行逐项检查,检测其包含的时间信息是否满足匹配路径上相应独立以及相关时间约束的要求,检测并识别不同类型的软件时序缺陷.其中,对于一组相关时间约束,在后结束计时的时间变量的计时终点,对两项相关时间变量间的取值依赖关系进行检测.

例如在图 2 所示的门禁控制软件中,执行片段为 $Trace=\{(man\ approach,,,approach?,,reset(\tau_1,\tau_2)),(,,,open!,D\_Cstr_1(\tau_1),reset(\tau_0)),(man\ leave,,,leave?,D\_Cstr(\tau_2),reset(\tau_1)),(,,,close!,D\_Cstr_2(\tau_1)\wedge D\_Cstr(\tau_0),)\}$,匹配的 ESIA 执行路径为 $Path=\{State1\rightarrow State2, State2\rightarrow State3, State3\rightarrow State4, State4\rightarrow State1\}$,识别的相关时间约束如 4.2 节所述($R\_Cstr_1$、$R\_Cstr_2$、$R\_Cstr_3$).该执行片段的时序缺陷检测过程为:针对行为 $approach?$,检测 τ_1、τ_2 是否在该行为结束时置零;针对行为 $open!$,检测 τ_1 的计时时间是否满足 $D\_Cstr_1(\tau_1)$ 的要求,检测 τ_0 是否在该行为结束时置零;针对行为 $leave?$,检测 τ_2 的计时时间是否满足 $D\_Cstr(\tau_2)$ 的要求,检测 τ_1、τ_2 的计时时间是否满足 R_Cstr_1 的要求,检测 τ_1 是否在该行为结束时置零;针对行为 $close!$,检测 τ_1 的计时时间是否满足 $D\_Cstr_2(\tau_1)$ 的要求,检测 τ_0 的计时时间是否满足 $D\_Cstr(\tau_0)$ 的要求,检测 τ_2、τ_0 的计时时间是否满足 R_Cstr_2 的要求,检测 τ_0、τ_1 的计时时间是否满足 R_Cstr_3 的要求.

时序缺陷检测算法具体定义如下.

算法 3. 时序缺陷检测 $Defect\_Detection(\ )$.

```
输入:A Trace={λ_j | j=1,2,…,n}
      A matched Path={δ_i | i=1,2,…,n} on ESIA P
      A set of correlated timing constraints(SCTC)
输出:The result of defect detection(RDD)
Defect_Detection(Trace,Path,SCTC)
RDD=FALSE
BEGIN
  FOR each λ on Trace
    Get δ=map(λ) on Path
    FOR each time value τ=v record before or after the execution of λ.a
      IF(( ∃Cstr(τ)∈δ.PreCon AND v⊢Cstr(τ))
      OR ( ∃Cstr(τ)∈δ.PostCon AND v⊢Cstr(τ)))
                                   //独立时间约束检测
        Null
      ELSE
        TDR=Constraint_voliation
        Return RDD
      IF( ∃R_Cstr∈SCTC,R_Cstr.X^T={τ,τ_x},τ_x∈
      X_P^T AND v⊢Dep(τ,τ_x)) //相关时间约束检测
```

```
        Null
      ELSE
        TDR=Correlation_voliation
        Return TDR
    END FOR
    FOR each τ resets at the beginning OR end of λ
                                    //计时起点检测
      IF(τ is reset at corresponding location of δ)
                                    //置零行为检测
        Null
      ELSE
        RDD=Reset_Error
        Return RDD
    END FOR
  END FOR
END
```

在以上算法中，设 n 为执行片段长度，m_1 为行为开始或结束时采集的执行时间的组数，m_2 为行为开始或结束时置零的时间变量数，则算法复杂度为 $O(n\times m_1+n\times m_2)$.

在时序缺陷检测过程中，针对每一条执行片段，依次调用以上执行片段匹配、相关时间约束识别以及时序缺陷检测算法，得到相应检测结果. 此外，在 TBTDD 方法中，可根据匹配的 ESIA 执行路径对时序缺陷进行定位. 当软件运行时序违反特定时间约束、相关关系或计时起点设置时，或者特定时序异常处理机制与设计要求不符时，软件实现的对应部分可能存在缺陷.

5　实验与评估

本文设计实现了 ATES(Automated Test Platform for Embeded Software)平台原型系统，内建 TBTDD 时序检测方法，支持嵌入式软件时序特性建模与离线自动检测，识别与管理各类时序故障，平台架构如图 6 所示.

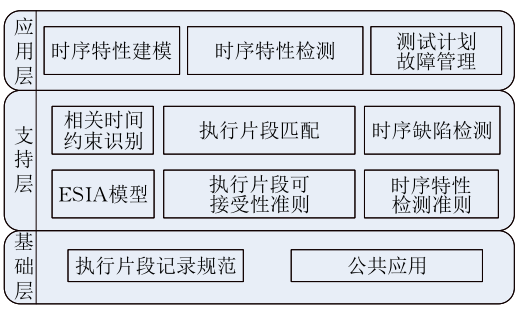


图 6　ATES 平台架构

ATES 平台具有以下功能：

(1) 基于 ESIA 模型的嵌入式软件时序特性建模；

(2) 基于可接受性准则的执行片段匹配；

(3) 基于相关矩阵的相关时间约束识别；

(4) 检测软件运行时序，侦测执行片段中的时序异常，验证异常时序处理机制的有效性，发现并识别各类软件时序缺陷；

(5) 针对执行片段集制定检测计划，管理并报告时序故障.

在实验过程中，利用 ATES 平台提供的基础类，人工编写被测软件的 ESIA 模型代码，对其时序特性进行定量描述；进而在 ATES 平台上，以 ESIA 模型为基础模型，调用平台内建的 TBTDD 方法，对预先提取的执行片段集执行时序检测. 在 ATES 平台之上，测试人员可灵活制定测试计划，组织调用各项平台功能，并动态监控检测过程.

5.1　实验设计

本节以一项卫星定位系统软件(Satellite Positioning System Software，SPS)为例，对 ESIA 时序语义与 TBTDD 方法进行验证与评估.

5.1.1　目标软件建模

卫星定位导航指利用导航卫星信号对地面、空中、空间以及海洋目标进行定位和导航的技术，通过测量平台与多颗导航卫星之间的距离，并综合卫星位置数据等信息，精确计算平台所处位置，引导其沿既定路线行进. 当前，卫星定位导航技术已在地面车辆、航空、航天、航海、地理数据采集以及高精度测量等领域得到广泛应用. 美、俄、欧洲以及中国等国家或地区均已建立、或正在建立各自主导的全球性卫星导航系统，如美国的全球定位系统(Navigation Satellite Timing And Ranging Global Positioning System，GPS)、俄罗斯的格洛纳斯系统(Global Navigation Satellite System，GLONASS)、欧洲的伽利略系统(Galileo Positioning System，Galileo)以及中国的北斗卫星导航系统(BeiDou Navigation Satellite System，BDS)等，目标均是实现全球范围内的连续、实时、高精度和全天候定位与导航. 为保证定位的精度与实时性，各类卫星定位系统均具有严格的时间性能要求.

本节中的卫星定位系统软件利用 BDS 卫星信号，实时完成定位解算，向用户提供平台位置、速度等信息. SPS 软件是典型的实时嵌入式软件，直接运行于板级支持包(Board Support Package，BSP)与嵌入式处理器平台之上，并通过各类总线与系统内的其他嵌入式设备连接.

SPS 软件既可利用 BDS 普通测距码(C 码)进行定位解算,也可利用精密测距码(P 码)进行定位解算.在利用 P 码进行定位解算时,其具有 M1、M2 两种 P 码接收方式.由此,SPS 软件具有 C、P-M1 以及 P-M2 三种定位解算模式.此外,SPS 软件具有 S1、S2 两个工作阶段,在不同工作阶段内,根据运行时间信息,依据特定切换策略,交替采用以上三种解算模式,以进行持续、可靠的定位与导航.

(1) S1 阶段.在此阶段内,SPS 软件首先进入 P-M1 模式,若在规定时间内未接收到 BDS 卫星信号,则切换至 P-M2 模式;

(2) S2 阶段.SPS 软件可能在 P-M1 或 P-M2 两种模式下由 S1 阶段转入 S2 阶段,在此情况下,其在 S2 阶段内具有不同的定位解算模式切换策略,导致以下不同的工作场景:

① S2_M1:SPS 软件首先进入 P-M1 模式,若在规定时间内未接收到 BDS 卫星信号或在规定时间内未完成定位解算,则在 P-M2 与 C 模式之间顺序切换;

② S2_M2:SPS 软件首先进入 P-M2 模式,若在规定时间内未接收到 BDS 卫星信号或在规定时间内未完成定位解算,则在 P-M1、C 与 P-M2 模式之间循环切换.

SPS 软件具有以下时间需求:

(1) SPS-TR1.在 P-M1 模式下,应在时间区间 $[t_{l-1},t_{u-1}]$ 内接收卫星信号;

(2) SPS-TR2.在 C、P-M1 与 P-M2 模式下,应在时间区间 $[t_{l-2},t_{u-2}]$ 内完成定位解算;

(3) SPS-TR3.在 C、P-M2 模式下,应在时间区间 $[t_{l-3},t_{u-3}]$ 内接收卫星信号.

SPS 软件在 S1、S2 阶段下的 ESIA 模型如图 7 所示.其中,*Rcv_M1Inf* 行为接收 P 码初始装订信息;*Ret_Inf* 行为返回卫星导航数据;*CapP_M1*、*CapP_M2* 与 *CapC* 行为分别采用 M1 模式接收 P 码、采用 M2 模式接收 P 码以及接收 C 码;*Exe_Pos* 行为在相应模式下执行定位解算;*Ret_PosInf* 行为返回定位结果;*Ret_ExcInf* 行为返回异常信息.其中,*action_E* 表示行为 *action* 的异常执行分支,引导软件进入相应异常处理.

在 SPS 软件运行的 S1 阶段(如图 7(a)所示),首先进入 P-M1 模式,*Rcv_M1Inf* 接收 P 码初始装订信息,引导 *CapP_M1* 接收 P 码,由 *Ret_Inf* 向用户返回接收到的导航信息.若在规定时间内未接收到 P 码,则转入异常分支 *CapP_M1_E*,由 *Ret_ExcInf*

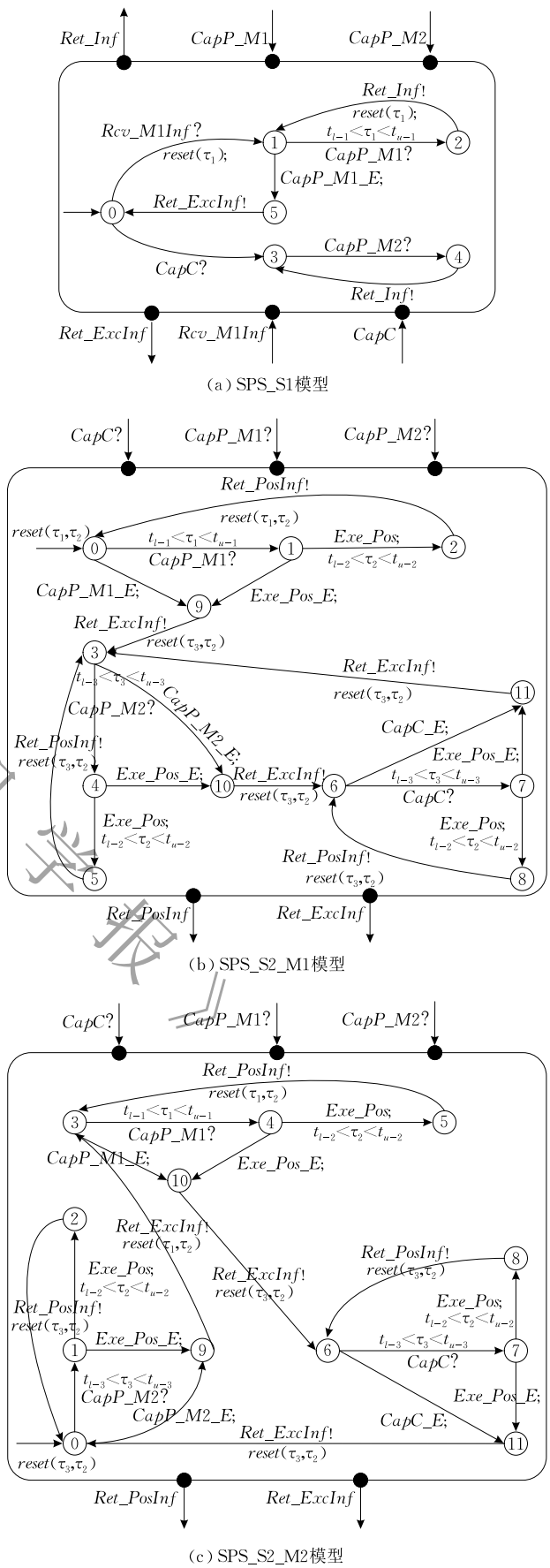


图 7 SPS_S1、SPS_S2_M1 和 SPS_S2_M2 模型

返回异常信息，SPS 软件切换至 P-M2 模式. 在 P-M2 模式下，*CapC* 接收 C 码，进而引导 *CapP_M2* 接收 P 码，并由 *Ret_Inf* 返回导航信息.

在 SPS 软件运行的 S2_M1 阶段（如图 7(b)所示），首先进入 P-M1 模式，*CapP_M1* 接收 P 码，*Exe_Pos* 执行定位解算，由 *Ret_PosInf* 返回定位结果. 若在规定时间内未接收到 P 码，或在规定时间内未完成定位解算，则分别转入异常分支 *CapP_M1_E* 和 *Exe_Pos_E*，由 *Ret_ExcInf* 返回异常信息，SPS 软件切换至 P-M2 模式. 在 P-M2 模式下，*CapP_M2* 接收 P 码，*Exe_Pos* 执行定位解算，由 *Ret_PosInf* 返回定位结果. 若在规定时间内未接收到 P 码，或在规定时间内未完成定位解算，则分别转入异常分支 *CapP_M2_E* 和 *Exe_Pos_E*，由 *Ret_ExcInf* 返回异常信息，SPS 软件切换至 C 模式. 在 C 模式下，*CapC* 接收 C 码，*Exe_Pos* 执行定位解算，由 *Ret_PosInf* 返回定位结果. 若在规定时间内未接收到 C 码，或在规定时间内未完成定位解算，则分别转入异常分支 *CapC_E* 和 *Exe_Pos_E*，由 *Ret_ExcInf* 返回异常信息，SPS 软件再次切换至 P-M2 模式.

SPS 软件运行的 S2_M2 阶段（如图 7(c)所示）与 S2_M1 阶段相似，区别在于 SPS 软件将首先进入 P-M2 模式，并在 P-M2 模式、P-M1 模式与 C 模式之间循环切换.

τ_1、τ_2与 τ_3为三项时间变量，在 SPS 软件不同运行阶段模型中，τ_1记录 P-M1 模式下的卫星信号接收时间，τ_2记录 C、P-M1 与 P-M2 模式下的定位解算时间，τ_3记录 C、P-M2 模式下的卫星信号接收时间.

根据 SPS 软件的时间需求，设置其时间约束如下：

SPS-TR1. $D\_Cstr(\tau_1)=[t_{l-1},t_{u-1}]$；

SPS-TR2. $D\_Cstr(\tau_2)=[t_{l-2},t_{u-2}]$；

SPS-TR3. $D\_Cstr(\tau_3)=[t_{l-3},t_{u-3}]$.

在以上时间约束之间，存在以下相关关系：

(1) $Path(\tau_1)\subseteq Path(\tau_2)$，$D_Cstr(\tau_1)$与$D_Cstr(\tau_2)$包含相关，$C\_Dep(\tau_1,\tau_2):\tau_1<\tau_2$；

(2) $Path(\tau_3)\subseteq Path(\tau_2)$，$D_Cstr(\tau_3)$与$D_Cstr(\tau_2)$包含相关，$C\_Dep(\tau_3,\tau_2):\tau_3<\tau_2$.

5.1.2　执行片段提取

执行片段提取工作在面向目标机的半实物仿真环境中进行. 在各类运行场景下，采用预先制定的信息标识，在软件运行日志中，按规定格式实时记录软件执行序列与相应时间信息，提取有限的执行片段集，作为离线时序检测的输入数据.

嵌入式软件运行日志保存一系列软件运行时信息，为后续的运行结果分析、故障管理、任务回放与系统维护等提供数据支持. 在软件运行过程中，将提取的执行片段实时保存在运行日志中，并可通过目标机调试接口进行调阅.

执行片段的提取需要遵循一定规范，以使获得的执行片段具有相同规格，为后续集中和统一处理提供便利. 在实验过程中，采用的执行片段提取规范如下：

(1) 起点要求. 在 S1 阶段，以上电初始状态为执行片段记录起点；在 S2 阶段，以转入 S2 阶段的初始状态为记录起点；

(2) 终点要求. 从记录起点开始，当软件运行终止（当前状态无后继行为）或软件运行回到相应阶段的初始状态时，则执行片段记录终止；

(3) 循环结构要求. 从记录起点开始，当软件中循环结构的执行违反前述循环覆盖 All-Loop-*n*/All-*m*-Loop 准则时，执行片段记录终止；

(4) 长度要求：当执行片段长度增长至 *len* 时，$len\in Z^+$，执行片段记录终止.

其中，*n*、*m* 与 *len* 的取值可根据软件规模、复杂度以及特定检测要求具体确定. 在本节中，设置 *n* 与 *m* 的取值为 1，根据 SPS_S1、SPS_S2_M1 与 SPS_S2_M2 模型描述的软件行为和时序特性，经分析得出满足规范(1)、(2)与(3)的执行路径最大长度为 9，故设置 *len* 的取值为 9. 在执行片段提取过程中，通过在目标软件中进行插桩，实时判断当前执行片段是否符合规范要求，及时启动和终止执行片段记录过程，并保存已提取的执行片段.

为对 TBTDD 方法的时序缺陷检测能力进行全面考察，在测试环境中，采用卫星信号仿真设备，设置 BDS 系统的各类星况，激活 SPS 软件的各项运行剖面. 在此情况下，持续运行 SPS 软件，按照以上规范在 S1、S2_M1 与 S2_M2 等阶段分别提取执行片段，从而使提取的执行片段集满足 4.1.1 节中所述的 ESIA 模型覆盖准则要求，以保证检测工作的充分性.

5.2　实验结果与分析

本节采用 ATES 平台内置的 TBTDD 方法，利用预先提取的执行片段集，在 SPS_S1、SPS_S2_M1 与 SPS_S2_M2 模型之上进行软件运行时序检测，并从执行片段集和缺陷检测等方面将 TBTDD 方法与 Andrés、Morales 和 Núñez 等人[11-12]提出的基于时间不变量的时序缺陷检测算法（Timed Invariant-

Based Defect Detection,TIBDD)进行比较.

Andrés、Morales 和 Núñez 等人在时间有限状态机模型(Timed Finite State Machine,TFSM)基础上,定义了时间不变量(Time Invariant),对特定事件/行为序列上的时序特性要求进行描述,并将其作为时序检测的准则.在时间不变量设计中,采用一组时间区间限制特定行为序列上各项任务的执行时间,其形式为"导言→结论".在此基础上,Andrés 和 Morales 等人提出 TIBDD 检测算法,对软件运行时序进行检测.当执行片段包含的行为序列和时间信息满足一项不变量导言部分的要求时,若软件后续运行过程中的行为序列和时间信息满足不变量结论部分的要求,则运行时序满足设计要求.

TBTDD 方法与 TIBDD 算法的主要区别如下:

(1)时序特性的描述方式不同.在 TBTDD 方法中,采用 ESIA 模型描述被测软件的时序特性,作为时序检测的基础模型.而在 TIBDD 算法中,以时间不变量描述被测软件的时序特性,需针对每一项时序设计要求分别定义相应的时间不变量.

(2)运行时序的检测过程不同.TBTDD 方法依据执行片段可接受性准则以及时序特性可满足性准则,在被测软件 ESIA 模型之上,检验执行序列包含的时间信息是否满足时序设计要求,进而发现相应时序故障.TIBDD 算法则依据预定义的时间不变量,检验执行片段是否满足其导言以及结论部分描述的时序设计要求.

(3)时序检测的故障类型不同.根据时间约束相关性分析和时序缺陷分析结果,TBTDD 方法可发现违反独立时间约束、违反相关时间约束、违反计时起点要求以及异常时序处理错误等典型时序故障.而在 TIBDD 算法研究中,未对时间约束相关性以及软件运行中可能存在的时序故障进行系统分析,其时序检测主要集中于对违反独立时间约束情况的检测.

(4)时序检测的覆盖范围不同.TBTDD 方法是一种通用检测方法,其检测范围覆盖被测软件 ESIA 模型所包含的全部时序设计信息,能否发现特定时序故障取决于提取的执行片段能否覆盖相应执行路径.TIBDD 算法则更多地针对特定时序设计要求开展检测,能否发现某项时序故障不仅取决于执行片段的提取,还取决于时间不变量定义的内容.

由于 TBTDD 方法与 TIBDD 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.

5.2.1 执行片段集

在实验过程中,在 SPS 软件运行的 S1、S2 阶段提取一系列执行片段.随后,在提取的执行片段集之上,通过聚类分析,合并同类执行片段,删除重复执行片段,共确定 S1 阶段执行片段 3 条,如表 1 所示;S2_M1 阶段执行片段 15 条,如表 2 所示;S2_M2 阶段执行片段 15 条.其中,S2_M2 阶段执行片段集与 S2_M1 阶段类似.

表 1 S1 阶段执行片段集

序号	执行片段包含的行为序列	类型
1	*Rcv_M1Inf*? /*CapP_M1*? /*Ret_Inf*!	正常
2	*Rcv_M1Inf*? /*CapP_M1_E*? /*Ret_ExcInf*!	异常
3	*Cap_C*? /*CapP_M2*? /*Ret_ExcInf*!	正常

表 2 S2_M1 阶段执行片段集

序号	执行片段包含的行为序列	类型
1	*CapP_M1*? /*Exe_Pos*; /*Ret_PosInf*!	正常
2	*CapP_M1*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*CapP_M2*? /*Exe_Pos*; /*Ret_PosInf*!	异常
3	*CapP_M1*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*CapP_M2*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*Cap_C*? /*Exe_Pos*; /*Ret_PosInf*!	异常
4	*CapP_M1*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*CapP_M2*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*Cap_C*? /*Exe_Pos_E*; /*Ret_ExcInf*!	异常
5	*CapP_M1*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*CapP_M2*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*Cap_C_E*? /*Ret_ExcInf*!	异常
6	*CapP_M1*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*CapP_M2_E*? /*Ret_ExcInf*! /*Cap_C*? /*Exe_Pos*; /*Ret_PosInf*!	异常
7	*CapP_M1*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*CapP_M2_E*? /*Ret_ExcInf*! /*Cap_C*? /*Exe_Pos_E*; /*Ret_ExcInf*!	异常
8	*CapP_M1*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*CapP_M2_E*? /*Ret_ExcInf*! /*Cap_C_E*? /*Ret_ExcInf*!	异常
9	*CapP_M1_E*? /*Ret_ExcInf*! /*CapP_M2*? /*Exe_Pos*; /*Ret_PosInf*!	异常
10	*CapP_M1_E*? /*Ret_ExcInf*! /*CapP_M2*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*Cap_C*? /*Exe_Pos*; /*Ret_PosInf*!	异常
11	*CapP_M1_E*? /*Ret_ExcInf*! /*CapP_M2*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*Cap_C*? /*Exe_Pos_E*; /*Ret_ExcInf*!	异常
12	*CapP_M1_E*? /*Ret_ExcInf*! /*CapP_M2*? /*Exe_Pos_E*; /*Ret_ExcInf*! /*Cap_C_E*? /*Ret_ExcInf*!	异常
13	*CapP_M1_E*? /*Ret_ExcInf*! /*CapP_M2_E*? /*Ret_ExcInf*! /*Cap_C*? /*Exe_Pos*; /*Ret_PosInf*!	异常
14	*CapP_M1_E*? /*Ret_ExcInf*! /*CapP_M2_E*? /*Ret_ExcInf*! /*Cap_C*? /*Exe_Pos_E*; /*Ret_ExcInf*!	异常
15	*CapP_M1_E*? /*Ret_ExcInf*! /*CapP_M2_E*? /*Ret_ExcInf*! /*Cap_C_E*? /*Ret_ExcInf*!	异常

以上执行片段集既包含正常时序片段，也包含异常时序片段，可利用其对目标软件各类运行时序以及异常时序处理方式进行检测. 在后续实验过程中，采用以上执行片段集，以 SPS_S1、SPS_S2_M1 与 SPS_S2_M2 模型为基础，开展执行片段匹配、相关时间约束分析以及时序缺陷检测等工作，以验证 TBTDD 方法的有效性.

5.2.2　缺陷检测

在实验中，通过在被测软件中植入一系列时序缺陷，对 TBTDD 方法的时序缺陷检测能力进行验证. 植入的时序缺陷包括违反独立时间约束、异常时序处理错误、违反相关时间约束以及违反计时起点要求等故障类型，如表 3 所示. 针对每种故障类型，分别植入了多项时序缺陷，检验 TBTDD 方法能否正确检测相应时序异常. 此外，针对执行片段匹配过程，植入行为错配和行为错序等功能性故障，检验 TBTDD 方法能否检测相应功能异常.

表 3　时序缺陷检测能力

缺陷类型	缺陷描述	TBTDD 检测能力	TIBDD 检测能力
行为错配	在 S1 阶段，提取执行片段 *Rcv_M1Inf*? /*CapP_M2*? /*Ret_Inf*!，行为 *CapP_M2* 错配；	发现	发现
行为错序	在 S2_M1 阶段，提取执行片段 *CapP_M1*? / *Ret_ExcInf*! /*Exe_Pos_E*; / *CapP_M2*? /*Exe_Pos*; /*Ret_PosInf*!，行为序列 *Ret_ExcInf*! /*Exe_Pos_E*; 错序；	发现	发现
违反独立时间约束	在 S1 阶段，*CapP_M1*? 的执行时间低于 $D\_Cstr(\tau_1)$ 的下限；	发现	发现
	在 S1 阶段，*CapP_M1*? 的执行时间高于 $D\_Cstr(\tau_1)$ 的上限；	发现	发现
	在 S2_M1 阶段，*CapP_M1*? 的执行时间低于 $D\_Cstr(\tau_1)$ 的下限，*CapP_M1*? 执行	发现	发现
	在 S2_M1 阶段，*CapP_M1*? 的执行时间高于 $D\_Cstr(\tau_1)$ 的上限；	发现	发现
	在 S2_M1 阶段，*CapP_M2*? 的执行时间低于 $D\_Cstr(\tau_3)$ 的下限；	发现	发现
	在 S2_M1 阶段，*CapP_M2*? 的执行时间高于 $D\_Cstr(\tau_3)$ 的上限；	发现	发现
	在 S2_M2 阶段，*CapC*? 的执行时间低于 $D\_Cstr(\tau_3)$ 的下限；	发现	发现
	在 S2_M2 阶段，*CapC*? 的执行时间高于 $D\_Cstr(\tau_3)$ 的上限；	发现	发现
	在 S2_M2 阶段，*Exe_Pos*; 的执行时间低于 $D\_Cstr(\tau_2)$ 的下限；	发现	发现
	在 S2_M2 阶段，*Exe_Pos*; 的执行时间高于 $D\_Cstr(\tau_2)$ 的下限；	发现	发现
违反相关时间约束	在 S2_M1 阶段，*CapP_M1*? 的执行时间满足 $D\_Cstr(\tau_1)$，*Exe_Pos*; 的执行时间满足 $D\_Cstr(\tau_2)$，同时二者违反 $C\_Dep(\tau_1,\tau_2)$；	发现	未发现
	在 S2_M1 阶段，*CapP_M2*? 的执行时间满足 $D\_Cstr(\tau_3)$，*Exe_Pos*; 的执行时间满足 $D\_Cstr(\tau_2)$，同时二者违反 $C\_Dep(\tau_3,\tau_2)$；	发现	未发现
	在 S2_M1 阶段，*CapC*? 的执行时间满足 $D\_Cstr(\tau_3)$，*Exe_Pos*; 的执行时间满足 $D\_Cstr(\tau_2)$，同时二者违反 $C\_Dep(\tau_3,\tau_2)$；	发现	未发现
	在 S2_M2 阶段，*CapP_M2*? 的执行时间满足 $D\_Cstr(\tau_3)$，*Exe_Pos*; 的执行时间满足 $D\_Cstr(\tau_2)$，同时二者违反 $C\_Dep(\tau_3,\tau_2)$；	发现	未发现
	在 S2_M2 阶段，*CapP_M1*? 的执行时间满足 $D\_Cstr(\tau_1)$，*Exe_Pos*; 的执行时间满足 $D\_Cstr(\tau_2)$，同时二者违反 $C\_Dep(\tau_1,\tau_2)$；	发现	未发现
	在 S2_M2 阶段，*CapC*? 的执行时间满足 $D\_Cstr(\tau_3)$，*Exe_Pos*; 的执行时间满足 $D\_Cstr(\tau_2)$，同时二者违反 $C\_Dep(\tau_3,\tau_2)$；	发现	未发现
违反计时起点要求	在 S1 阶段，在 *Rcv_M1Inf*? 执行结束时，τ_1 未置零；	发现	未发现
	在 S2_M1 阶段，在 *Ret_PosInf*! 执行结束时，τ_1 与 τ_2 未置零；	发现	未发现
	在 S2_M1 阶段，在 *Ret_ExcInf*! 执行结束时，τ_1 与 τ_2 未置零；	发现	未发现
	在 S2_M2 阶段，在 *Ret_PosInf*! 执行结束时，τ_3 与 τ_2 未置零；	发现	未发现
	在 S2_M2 阶段，在 *Ret_ExcInf*! 执行结束时，τ_3 与 τ_2 未置零；	发现	未发现
异常时序处理错误	在 S2_M1 阶段，*CapP_M1*? 的执行时间满足 $D\_Cstr(\tau_1)$，定位解算时间违反 $D\_Cstr(\tau_2)$，时序异常处理未被激活，*Exe_Pos_E*; 未执行；	发现	发现
	在 S2_M1 阶段，M1 模式下的 P 码接收时间违反 $D\_Cstr(\tau_1)$，时序异常处理未被激活，*CapP_M1_E*; 未执行；	发现	发现
	在 S2_M2 阶段，*CapP_M2*? 的执行时间满足 $D\_Cstr(\tau_3)$，定位解算时间违反 $D\_Cstr(\tau_2)$，时序异常处理未被激活，*Exe_Pos_E*; 未执行；	发现	发现
	在 S2_M2 阶段，M2 模式下的 P 码接收时间违反 $D\_Cstr(\tau_3)$，时序异常处理未被激活，*CapP_M2_E*; 未执行；	发现	发现
	在 S2_M2 阶段，*CapC*? 的执行时间满足 $D\_Cstr(\tau_3)$，定位解算时间违反 $D\_Cstr(\tau_2)$，时序异常处理未被激活，*Exe_Pos_E*; 未执行；	发现	发现
	S2_M2 阶段，C 模式下的 C 码接收时间违反 $D\_Cstr(\tau_3)$，时序异常处理未被激活，*CapC_E*; 未执行；	发现	发现

以上植入的各类时序缺陷分布在 SPS 软件运行的 S1、S2_M1 以及 S2_M2 等阶段内，针对各类独立和相关时间约束，覆盖各阶段的正常时序以及异常时序处理分支. 在缺陷植入时，通过设置特定行为

的执行时间,使其违反相应独立和相关时间约束;或针对特定任务的计时区间,设置错误的计时起点,使任务执行时间的记录出现偏差,由此造成软件运行时序异常.

实验结果显示,TBTDD 方法可正确发现以上 29 项时序缺陷,表现出良好的缺陷检测能力. TIBDD 方法可发现 18 项植入的时序缺陷,未发现违反相关时间约束以及违反计时起点要求的时序异常.

TBTDD 方法的时序缺陷检测能力主要受以下方面因素影响:

(1)时间需求的描述. 完整、精确的被测软件 ESIA 模型是时序缺陷检测的基础. EISA 模型应完整且精确地描述被测软件的时序特性,以支持对时序设计要求的正确分析与理解.

(2)时间约束相关性分析. 在时序检测中,不仅需检验各项任务的执行时间是否满足相应时间约束,还需检验任务的执行时间是否满足与其间接相关的时间约束.

(3)执行片段的提取. 提取的执行片段集合应满足 ESIA 模型覆盖准则要求,以保证时序特性检测的充分性.

(4)执行序列的匹配与检测. 时序检测准则、执行片段匹配算法以及时序缺陷检测算法的精确性与执行效率均将直接影响 TBTDD 方法的时序缺陷检测能力.

TIBDD 算法的时序缺陷检测能力主要受以下方面因素影响:

(1)时间不变量的定义. 时间不变量定义的充分性、正确性将直接影响 TIBDD 方法的缺陷检测能力. 在时序检测中,需要预估可能出现的时序异常,设计相应时间不变量. 不充分的时序缺陷分析可能造成时间不变量缺失,使 TIBDD 方法无法发现相应时序异常. 此外,若时间不变量的定义不符合规定格式或不满足软件设计要求,则可能造成时序故障的检测或定位不准确.

(2)TIBDD 算法的时序缺陷检测能力同样受执行片段提取、算法的精确性与执行效率等方面因素的影响.

由以上分析可见,时间需求的描述以及时间约束相关性分析等因素是造成 TBTDD 方法与 TIBDD 算法缺陷检测能力差异的主要原因. 在 TIBDD 算法研究中,未对时间约束相关性以及软件运行中可能存在的时序故障进行系统分析,其时间不变量的定义未考虑违反相关时间约束以及违反计时起点要求造成的时序异常,因而难以发现相应类型的时序故障. 由此,理论上 TBTDD 方法具有比 TIBDD 算法更强的时序缺陷检测能力,其可检测的时序异常包含 TIBDD 方法能够检测的时序异常. 若对时间不变量的定义和描述方式进行扩展,补充考虑时间约束相关性以及计时区间设置等因素,则可进一步增强其发现相应时序故障的能力.

5.2.3 执行效率分析

TIBDD 算法的执行需要预先定义一系列时间不变量,增加了额外的形式化工作;在对一组关联任务或在复杂运行场景下进行检测时,时间不变量的形式将更复杂. 针对不同任务剖面制定的时间不变量可能包含相同的独立时间约束,将造成时间约束的重复定义;当时序设计要求变更时,需对全部受影响的时间不变量进行调整,回归测试工作量大. 此外,在时序检测过程中,TIBDD 方法需要将每条执行片段与所有时间不变量的引语部分进行逐次匹配,当执行片段集合和时间不变量集合的规模不断增长或不变量导言部分较长时,算法执行效率将受到显著影响.

TBTDD 方法则直接利用 ESIA 模型描述的时间需求作为检测度量,无需定义时间不变量,避免了额外的形式化工作. 通用的执行片段匹配与检测准则以及完整的被测软件 ESIA 模型适用于各类运行场景下的时序特性检测,具有良好的适用性. 若时序设计要求发生变更,仅需更新 ESIA 模型相关部分描述,支持便捷的回归测试.

5.2.4 局限性分析

在时序检测工作中,对领域知识的获取与理解以及对时间需求的抽象与建模等更多依赖于测试人员的技术能力与工程经验,因此,ESIA 模型构建可能存在完整性和准确性等问题,TBTDD 方法可能无法检测到特定时序缺陷. 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 模型对嵌入式软件时序特性进行完整、准确建模的方法.

(1)对时间约束间复杂依赖关系的处理

通常情况下,嵌入式软件中的时间约束相关关系主要存在于二项时间约束之间. 当多项时间约束间存在多重相关关系时,不同时间变量之间的取值依赖关系将更加复杂. 以上情况将导致时序缺陷检测过程更为复杂,缺陷的识别与定位也将更加困难.

在后续研究工作中，针对时间约束间多重相关的情况，研究不同时间变量取值间复杂依赖关系的提取、描述与化简方法，考虑制定相关关系可满足性的一般准则，降低其检测复杂度，以期在较小的代价下对违反相关关系的异常时序进行充分测试.

(2) 与行为特性相结合的混合特性检测

嵌入式软件执行路径由外部事件序列、触发事件产生时间、行为执行时间与变量以及参数取值共同确定. 在 TBTDD 方法中，仅对特定执行片段上的时序特性进行检测，而在执行片段之上，还存在一系列行为参数和内部变量等，其取值同样将影响软件运行状态与结果. 此外，任务执行时间与变量以及参数取值之间可能存在特定相互作用，参数的规模和取值等因素均可能影响软件运行时序. 当前，对行为特性与时序特性的分离检测无法适应以上混合特性检测的要求. 在后续工作中，考虑将嵌入式软件行为特性检测与时序特性检测相结合，研究建立通用的混合特性检测方法.

6　相关工作

在形式化测试方法研究中，基于执行序列的检测技术广泛应用于各类网络通信协议的测试工作. 近年来，时序特性检测和基于状态机的行为特性检测等领域的研究工作取得了一系列成果，提出了各种不同的检测方法.

6.1　时序特性检测

学术界针对基于执行序列的时序特性检测技术开展了一系列研究. 作为软件测试和形式化验证技术的有益补充，运行时序检测可在系统的设计、开发、确认、交付与维护等各个阶段实施，持续收集各类正常和异常时序信息，为系统时序特性设计方案的改进提供支持. 运行时序检测技术兼顾软件测试与形式化验证的优点，利用真实的执行序列作为检测输入，利用形式化的功能以及时间需求作为自动检测的依据，既可探查软件测试难以覆盖的特殊场景和特殊缺陷，也可避免模型检测带来的状态空间爆炸等问题. 在系统交付之后，运行时序检测更成为状态监控、异常处理以及维护保障的有效手段.

基于执行序列的时序特性检测技术主要可分为两类：一是基于时间模型的检测；二是基于时间规则的检测. 在基于时间模型的检测中，采用包含时间信息的行为模型对被测软件时序特性进行完整描述，进而在被测软件模型之上通过执行序列匹配和时间信息检测等过程实施时序特性检测，例如本文提出的 TBTDD 方法等. 在基于时间规则的检测中，采用时序逻辑对特定任务和运行场景的时间需求进行具体描述，制定一组形式化的时间规则，软件在运行的任何时刻均应遵循以上规则，进而利用该规则对提取的执行序列进行匹配与检测，例如前述 TIBDD 检测算法等.

2009 年，Trinh、Do 和 Truong 等人[13] 提出一种基于 UML 时序图的实时软件系统时间约束检测方法. 在检测过程中，利用面向方面的编程技术(Aspect Oriented Programming，AOP)在源代码中织入监控及检测代码，在软件运行过程中，实时提取任务执行信息和检验任务执行是否满足相应时间约束，并指出潜在的违反时间约束的时序故障.

2015 年，Chen 和 Kumar[14] 提出一种基于线性时间逻辑(Linear-Time Temporal Logic，LTL)的离散时间随机系统时序特性检测方法. 在该方法中，使用离散差分公式描述物理系统在离散时间点上的动态特性，并以 LTL 描述其时间需求. 通过精化过程，将物理系统的形式化描述及其时序特性的 LTL 描述转化为 I/O 随机混合自动机(Input-Output Stochastic Hybrid Automaton，I/O-SHA)模型. 由此，物理系统的时序故障检测问题转化为 I/O-SHA 模型之上的状态可达性分析问题，即，若特定故障状态可达，则系统在运行时可能产生相应故障. 该方法利用随机过滤方程(Stochastic Filtering Equation)估算系统在运行时抵达特定故障状态的概率，从而对故障发生的可能性进行评估.

以上方法采用的时序特性描述模型和故障检测方式与 TBTDD 方法均存在不同. 在基于 UML 时序图的检测方法中，利用 AOP 技术在被测软件中织入监控及检测代码，将检测过程与软件运行过程交织在一起，而 TBTDD 方法则根据被测软件时间模型在后台实施检测过程. 在基于线性时间逻辑(Linear-Time Temporal Logic，LTL)的检测方法中，通过可达性分析搜索故障状态，而 TBTDD 方法则根据预定义的检测准则判断软件运行是否产生时序故障. 学术界对基于时间规则的时序特性检测技术进行了一系列研究，取得了更多的成果.

2008 年至 2009 年间，Andrés、Merayo 和 Núñez[15-16] 提出一种基于时间不变量的时序特性检测方法. 在时间有限状态机(Timed Finite State

Machine,TFSM)形式化描述基础上,采用时间不变量描述目标软件的时间特性需求.一项时间不变量包括导言部分与结论部分,其中,导言部分表示为一个输入/输出行为序列,以及一系列约束行为执行的时间区间;结论部分表示为一个输入/输出行为对,以及相应时间区间.当提取的软件执行片段与导言中的输入/输出行为序列相匹配,且执行片段包含的时间信息满足相应时间区间的要求时,若软件在后续运行过程中表现出的行为与时序特性满足结论部分的要求,则软件运行时序满足设计要求,否则存在时序缺陷.在此基础上,其提出了运行日志的正确性检测算法,自动验证执行片段包含的时间信息是否满足各类时间不变量的要求,从而建立了完整的实时系统时序特性检测框架.

随后,Andrés、Merayo 和 Núnez[17] 对以上时序特性检测方法进行扩展,以时间概率分布函数的形式,描述时间不变量中各项输入/输出行为的时间条件,从而对软件行为执行概率相对于运行时间区间的变化情况进行描述,并对运行日志检测算法进行适应性改进.针对软件运行过程中在各个状态上的驻留时间,Merayo[18] 在时间不变量中增加一系列时间区间,以约束软件在特定状态上的驻留时间,进一步支持输入行为等待时间的验证.此外,Andrés 和 Merayo 等人[19] 还讨论了从目标软件 TFSM 描述中自动抽取时间不变量的方法,以期提高以上方法的执行效率.

Andrés、Merayo 与 Molinero[20] 进一步针对目标软件执行片段,提出改变目标状态和改变行为输出等变异操作,通过对正常提取的执行片段执行变异操作,获得一组仿真执行片段,通过仿真片段集合检验时间不变量发现时序缺陷的能力.

2010 年,Bessayah 和 Cavalli[21] 提出一种基于精确时间逻辑(eXplicit Clock Temporal Logic,XCTL)的时间约束检测方法.在此方法中,将各类复杂的系统时间需求描述为一系列 XCTL 公式,提出了时间需求一致性检测算法,并验证执行片段包含的时间信息是否满足 XCTL 公式描述的设计要求.

2012 年,Wei、Huang 与 Cao 等人[22] 提出一种基于三值计算树逻辑(3-Valued Computation Tree Logic,CTL_3)的普适计算时序特性检测方法.在普适计算环境中,使用 CTL_3 描述特定应用的时序特性;利用其三值语义(3-Valued Semantics)描述有限前驱行为序列下和后继行为序列的非决定性;利用分支时间(Branching Time)概念,描述普适计算环境异步性造成的预期行为的不确定性.在普适计算的运行时序检测中,以一组 CTL_3 形式化描述为准则,检测特定应用的时序特性是否满足设计要求.

2014 年,Zhao、Chai 和 Liu[23] 提出一种基于监测的列车控制系统时序特性检测方法.针对 LTL 无法对时间约束进行定量描述的问题,作者提出一种度量区间时序逻辑(Metric Interval Temporal Logic,MITL),用以对列车控制系统的时序特性进行定量描述.在此基础上,采用区间标记技术,提出一种增量式的在线检测算法,对运行过程中采集的执行片段进行持续检测,以判断列车控制系统的工作时序是否满足时间约束要求.

6.2 基于状态机的行为特性检测

1997 年,Lee、Netravali 与 Sabnani 等人[24] 提出一种基于 FSM 的行为特性检测算法,分为行为追踪(homing)与故障检测(fault detection)两个阶段.在行为追踪阶段,针对提取的执行片段,在目标软件 FSM 模型之上搜索与其相匹配的执行路径,直到确定唯一的当前状态为止,转入下一阶段执行;在故障检测阶段,针对软件后续运行过程,持续检测当前行为的源状态是否为当前状态.在算法执行过程中,若当前行为的源状态不满足预期,则软件实现存在缺陷.2007 年,Ural、Xu 和 Zhang[25] 针对以上算法提出了三类改进算法,通过避免冗余检测,使用特定数据结构缓存中间结果等措施,进一步提高算法执行效率.

2001 年,Zhao、Yin 和 Wu[26] 实现了一个路由协议在线测试系统(OnLine Test System,OLTS),支持分布式环境中多重实例的在线检测.OLTS 通过一系列模块仿真路由信息的交换与控制过程,并通过状态同步算法,识别软件运行当前状态;通过拓扑分析,检测网络拓扑与设计要求的一致性,以检测路由协议与配置缺陷.2002 年,他们进一步讨论了路由协议实时检测所面临的一系列问题,如计算任务量大、需要开放多个系统接口、路由配置复杂以及路由信息获取困难等,并提出了相应改进措施[27].

2002 年至2006年间,Lee、Chen 和 Hao 等人[28-29] 在事件驱动的 EFSM 基础上,综合考虑行为前置条件中包含的变量约束条件以及不同变量间的依赖关系,提出一种数据特性检测方法.在行为追踪与故障检测过程中,利用路径上的各项变量约束条件,持续

精化变量取值区间，并判断变量在相应执行路径之上是否存在合理取值，若不存在则存在缺陷. 2003年，Chen、Wu 和 Chu[30] 同样在事件驱动的 EFSM 基础上，针对具有复杂相关关系的变量取值范围难以确定等问题，提出采用整数线性编程技术(Inter Liner Programming，ILP)进行变量约束条件一致性检测，持续精化变量取值区间，以支持数据特性检测.

2007 年，Benharref、Dssouli 和 Serhani 等人[31] 提出一种加速行为追踪过程的方法，以提高在线检测方法的执行效率. 在在线检测过程中，当观察到一个事件或行为时，在 EFSM 之上执行一步正向行为匹配；在等待下一事件或行为的间隙，在 EFSM 之上执行一步逆向行为追踪，以推导可能导致当前状态的执行路径；在以上正向/逆向相结合的检测过程中，构建执行树，加速行为追踪过程.

7　总　结

时序测试是检验嵌入式软件时序特性是否满足设计要求以及对各类异常时序能否进行正确处理的有效手段. 本文提出一种基于执行序列的嵌入式软件时序缺陷检测方法 TBTDD. 在目标系统 ESIA 模型之上，制定了一组时序特性检测准则，通过收集目标软件执行片段集，依据匹配的 ESIA 执行路径上的各项独立、相关时间约束，对其包含的时间信息进行离线检测，从而发现软件实现中隐藏的时序缺陷. 本文对不同时间约束间相关关系类型进行讨论，在时序检测过程中，针对时间约束间相互作用的情况，分类检测违反特定时间约束、违反相关时间约束以及计时起点错误等时序缺陷.

实验结果表明，TBTDD 方法可有效检测软件运行过程中的各类时序异常，避免了环境因素对时序测试工作的影响，与传统时序测试方法配合使用时，可对目标软件时序特性进行更充分的测试，提高了测试的有效性和充分性. 在后续工作中，将对多项时间约束之间存在多重相关关系的情况进行研究，并对检测流程与算法进行进一步优化，研究运行时序异常的在线检测与故障定位方法.

致　谢　在此，我们向对本文工作给予支持和建议的同行，特别是实验室内各位老师和同学表示感谢！

参　考　文　献

[1] Alur R, Dill D L. A theory of timed automata. Theoretical Computer Science, 1994, 126(2): 183-235

[2] Lutz C, Wolter F, Zakharyaschev M. Temporal description logics: A Survey//Proceedings of the 15th International Symposium on Temporal Representation and Reasoning. Montreal, Canada, 2008: 3-14

[3] Yin Y F, Liu B, Ni H Y. A survey on the formal testing techniques for real-time embedded software//Proceedings of the 2nd International Conference on Information Science and Engineering. Hangzhou, China, 2010: 6426-6429

[4] 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

[5] Wang Bo, Bai Xiao-Ying, Zhang Chao, et al. Test case generation for embedded software using interface automata and symbolic execution. Chinese Journal of Computers, 2015, 38(11): 2125-2144(in Chinese)
(王博，白晓颖，张超等. 基于接口自动机与符号执行的嵌入式软件测试用例生成. 计算机学报，2015，38(11)：2125-2144)

[6] Dasarathy B. Timing constraints of real-time systems: Constructs for expressing them, methods of validating them. IEEE Transactions on Software Engineering, 1985, 11(1): 80-86

[7] Lee D, Netravali A N, Sabnani K K, et al. Passive testing and applications to network management//Proceedings of the IEEE International Conference on Network Protocols. Atlanta, USA, 1997: 113-122

[8] Miller R E, Arisha K A. Fault identification in networks by passive testing//Proceedings of the 34th Annual Simulation Symposium. Washington, USA, 2001: 277-284

[9] Miller R E. Passive testing of networks using a CFSM specification//Proceedings of the 17th International Performance Computing and Communications Conference. Phoenix/Tempe, USA, 1998: 111-116

[10] Alcalde B, Cavalli A, Chen D, et al. Network protocol system passive testing for fault management: A backward checking approach//de Frutos-Escrig D, Núñez M eds. Formal Techniques for Networked and Distributed Systems—FORTE. Berlin Heidelberg: Springer, 2004: 150-166

[11] Andrés C, Merayo M G, Núñez M. Passive testing of timed systems//Cha S(S.), Choi J-Y, Kim M eds. Automated Technology for Verification and Analysis. Berlin Heidelberg: Springer, 2008: 418-427

[12] Morales G, Maag S, Cavalli A, et al. Timed extended invariants for the passive testing of web services//Proceedings of the IEEE International Conference on Web Services. Miami, USA, 2010: 592-599

[13] Trinh T B, Do T A, Truong N T, Nguyen V H. Checking the compliance of timing constraints in software applications //Proceedings of the International Conference on Knowledge and Systems Engineering. Hanoi, Vietnam, 2009: 220-225

[14] Chen J, Kumar R. Fault detection of discrete-time stochastic systems subject to temporal logic correctness requirements. IEEE Transactions on Automation Science and Engineering, 2015, 12(4): 1369-1379

[15] Andrés C, Merayo M G, Núñez M. Formal correctness of a passive testing approach for timed systems//Proceedings of the 2nd IEEE International Conference on Software Testing, Verification and Validation Workshops. Denver, USA, 2009: 67-76

[16] Andrés C, Merayo M G, Núñez M. Applying formal passive testing to study temporal properties of the stream control transmission protocol//Proceedings of the 7th IEEE International Conference on Software Engineering and Formal Methods. Pisa, Italy, 2009: 73-82

[17] Andrés C, Merayo M G, Núñez M. Passive testing of stochastic timed systems//Proceedings of the 2nd IEEE International Conference on Software Testing Verification and Validation. Denver, USA, 2009: 71-80

[18] Merayo M G. Passive testing of timed systems with timeouts //Proceedings of the 12th IEEE International Conference on Quality Software. Xi'an, China, 2012: 69-78

[19] Andrés C, Merayo M G, Nuez M. Supporting the extraction of timed properties for passive testing by using probabilistic user models//Proceedings of the 9th IEEE International Conference on Quality Software. Jeju, Korea, 2009: 145-154

[20] Andrés C, Merayo M G, Molinero C. Advantages of mutation in passive testing: An empirical study//Proceedings of the 2nd IEEE International Conference on Software Testing, Verification and Validation Workshops. Denver, USA 2009: 230-239

[21] Bessayah F, Cavalli A. A formal passive testing approach for checking real time constraints//Proceedings of the 7th IEEE International Conference on the Quality of Information and Communications Technology. Porto, Portugal, 2010: 274-279

[22] Wei H, Huang Y, Cao J, et al. Formal specification and runtime detection of temporal properties for asynchronous context//Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications. Lugano, Switzerland, 2012: 30-38

[23] Zhao L, Chai M, Liu Y. Monitor-based temporal properties checking of train control systems with quantitative constraints //Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems. Qingdao, China, 2014: 2846-2851

[24] Lee D, Netravali A N, Sabnani K K, et al. Passive testing and applications to network management//Proceedings of the IEEE International Conference on Network Protocols. Atlanta, USA, 1997: 113-122

[25] Ural H, Xu Z, Zhang F. An improved approach to passive testing of FSM-based systems//Proceedings of the 2nd IEEE International Workshop on Automation of Software Test. Minneapolis, USA, 2007: 6

[26] Zhao Y X, Yin X, Wu J P. Online test system, an application of passive testing in routing protocols test//Proceedings of the IEEE 9th International Conference on Networks. Washington, USA, 2001: 190-195

[27] Wu J P, Zhao Y X, Yin X. From active to passive: Progress in testing of internet routing protocols//Proceedings of the 22nd International Conference on Formal Techniques for Networked and Distributed Systems. Houston, USA, 2002: 101-116

[28] Lee D, Chen D, Hao R, et al. A formal approach for passive testing of protocol data portions//Proceedings of the 10th IEEE International Conference on Network Protocols. Paris, France, 2002: 122-131

[29] Lee D, Chen D, Hao R, et al. Network protocol system monitoring: A formal approach with passive testing. IEEE/ACM Transactions on Networking, 2006, 14(2): 424-437

[30] Chen D, Wu J, Chu T I. An enhanced passive testing tool for network protocols//Proceedings of the International Conference on Computer Networks and Mobile Computing. Shanghai, China, 2003: 513-516

[31] Benharref A, Dssouli R, Serhani M A, et al. New approach for EFSM-based passive testing of web services//Petrenko A et al., eds. Testing of Software and Communicating Systems. Berlin Heidelberg: Springer, 2007: 13-27

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.

CHEN Wen-Guang, born in 1972, Ph.D., professor. His research interests include parallel and distributed computing.

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

Background

Timing constraints are critical to real-time embedded software. Mission-critical and safety-critical systems, such as avionics electronics and high-speed railway operation, usually have hard timing requirements. Temporal defects may cause runtime abnormities, even disasters. However, it is hard to verify and validate system temporal correctness when there exist multiple timing constraints with complex inter-dependencies.

Temporal defect detection using timed execution trace has been applied in verification of timing characteristics. It aims to reduce the impact of test environment and test engineer while enhance effectiveness and maintainability of temporal testing through formal method. The paper proposes a temporal defect detecting method using execution trace of embedded software. A model called Extended Semantic Interface Automata (ESIA) with temporal semantics is designed to capture timing characteristics including time variable, timing constraint, and reset action. It assumes that with sufficient domain knowledge, the model can provide a formal description of the expected timing characteristics that the software exposes. Execution traces with time information are extracted in running environment of target system, and correlations between timing constraints on execution trace are analyzed. Both normal temporal and temporal exception are checked based on analysis of constraints along test paths.

Experiments are exercised on a satellite positioning system using BeiDou Navigation Satellite System (BDS) signals. The execution traces are extracted on an in-house developed environment with target software. The ESIA model is built based on timing requirements of target software. The detecting effectiveness and efficiency are compared with other formal testing methods such as temporal defect detecting using timed invariants.

This research is supported by the National Natural Science Foundation of China (91218302, 61472197), and the Beijing Natural Science Foundation (4132062).