基于形式化测试的实时系统变更后安全性验证

孙海英^①, 刘静^{①*}, 陈小红^①, 杜德慧^①, 周庭梁^②

① 华东师范大学, 上海市高可信重点实验室, 上海 200062

② 同济大学道路与交通工程教育部重点实验室, 上海 200092

* 通信作者. E-mail: jliu@sei.ecnu.edu.cn

收稿日期: 2013-08-15; 接受日期: 2013-12-02

国家重点基础研究发展计划 (批准号: 2009CB320702)、国家自然科学基金 (批准号: 91318301, 61332008) 和国家自然科学基金青年基金 (批准号: 61202104) 资助项目

摘要 变更后系统实现的安全性验证是安全攸关系统维护过程中必不可少的环节, 也是其面临的主要挑战之一. 软件模型检测和程序验证是目前常用的作用于代码层面的自动化安全性验证技术. 本文站在系统行为角度, 基于形式化方法, 提出了一种将变更后系统实现的安全性验证问题归结为一致性测试的方法, 尝试通过自动生成的一致性测试用例在系统行为级别上判定系统实现是否安全. 为此, 首先以时间输入输出自动机及其语义模型为基础, 构建了该方法的证明体系, 证明了该方法的正确性; 其次, 建立了变更后系统实现安全性验证的回归测试生成框架. 相对于其它实时系统测试方法, 这种测试方法不仅可以发现实时系统中常规的 inconsistency 缺陷, 而且为变更后系统实现在运行时是否满足指定的安全性属性提供了依据. 最后, 以轨道交通系统中的列车自动防护功能的变更情景为案例研究, 说明了方法的具体应用.

关键词 软件工程 软件演化 实时系统 形式化方法 安全性验证 软件测试 安全性测试

1 引言

随着安全攸关系统需求数量和应用数量的不断增加以及系统规模和复杂度的不断提高, 高效和有效地确保变更后系统实现的正确性成为了该类系统必须面临的挑战之一. 一般情况下, 安全攸关系统的正确性意味着系统实现必须满足期望的相关属性, 例如安全性、活性等. 有研究表明, 为了无限接近系统实现正确性的目标, 传统上, 在安全攸关系统研发过程中约 50%~70% 的工作量是花费在对系统的各种验证和确认活动上^[1]. 由于通常直接关系到生命安全, 安全攸关系统的变更需要保证一个已经验证为安全的系统在变更后仍然安全, 即除了需要确保产生变化的系统部分的正确性, 还必须验证系统没有变化的部分没有受到影响. 因此, 如果没有合理有效的解决方法, 花费在安全性验证上的工作量会随着安全攸关系统的大规模应用以及随之而来的各种变更而变得更加突出.

软件模型检测和程序验证是目前常用的针对软件代码的自动化安全性验证方法. 但是目前这些方法在实际应用中存在以下几个问题:

- 在面对实际的系统代码时, 软件模型检测方法的通用性和可用性有限. Engler 等^[2] 根据对 3 个不同类型的项目长达几年的跟踪研究指出, 在不是为被验证代码设计的模型检测工具上验证代码是一

引用格式: 孙海英, 刘静, 陈小红, 等. 基于形式化测试的实时系统变更后安全性验证. 中国科学: 信息科学, 2014, 44: 70-90, doi: 10.1360/N112013-00140

件极其困难的事情, 这种困难主要来源于需要手工从代码中构造相关模型. 另外, 与一般理解相违背, 软件模型检测工具存在误报情况, 并且较之以其它静态方法, 其效率也不高^[2].

- 软件模型检测和程序验证本质上都是基于逻辑的方法, 因此, 包含了任何逻辑方法本身所固有的“前件失效”^[3]问题, 使得其无法精准地给出系统实现是正确的证明.

- 由于复用式开发的广泛存在, 一些开发团队在维护阶段只有部分甚至没有系统源代码, 限制了基于源码的各类验证方法的使用. 实际上, 如何保证商用现成 (commercial-off-the-shelf, COTS) 软件的安全性重用一直是安全攸关领域面临的挑战之一^[1,4].

除此以外, 重要的一点是, 安全性是一个系统层面的问题^[4,5]. 实际上, 来自航空^[6]、汽车^[7]、医疗^[1]等领域的一些数据表明, 只有极少比例的灾难性事故 (hazard accident) 是直接由软件本身的错误引起的. 导致灾难性事故发生的原因通常是, 在软件运行的系统上下文环境中产生了引起软件失效、继而导致灾难性事故发生的场景^[1,4]. 因此, 在目前理论和技术尚不完备的前提下, 仅通过单一的静态推理验证软件部分的安全性并不足以保证整个系统运行时的安全性, 重要的是要确保软件在其系统运行环境中的安全性.

与上述基于逻辑演算的方法不同, 一致性测试是一种被广泛采用的, 直接以系统实现为考查对象、以系统动态执行为主要手段的验证技术. 它将被测系统看成“黑盒”, 仅通过可见的输入输出行为判断被测系统是否满足给定规约. 由于是一种黑盒方法, 一致性测试方法不会受到代码有无的影响. 又由于测试的动态执行特性, 使得测试可以很容易解决所有以逻辑为基础的验证方法的“前件失效”问题. 更重要的是, 一致性测试是在系统运行上下文环境中执行的, 这有利于各种运行场景下系统实现的安全性验证. 然而, 传统的一致性测试方法大多建立在以寻找功能性缺陷为主要目标的命题上, 因此, 无法直接给出系统实现是否满足期望属性的说明.

本文以形式化方法为基础, 通过建立针对系统规约的安全性验证和一致性测试之间的形式化关系, 将变更后系统实现的安全性回归验证问题归结为一致性测试, 并且构造了可以判定变更后系统实现是否安全的测试框架. 该框架在使用自动化测试达到易化软件变更过程中测试活动的同时, 试图通过对系统规约的验证和相关的一致性测试结果直接判定变更后系统实现是否满足期望的安全性属性, 以省去在代码上进行形式化验证的环节. 基于形式化的测试是一种以严密数学定义和推导为理论基础, 以形式化规约为测试依据的自动化测试技术, 是形式化方法和测试相结合的典型代表. 两者的结合能够为系统实现的正确性提供强有力的保证^[8]. 由于本质上是一种自动化测试技术, 基于形式化的测试可以有效降低因劳动密集型本质带来的高测试成本和避免人为错误, 提高测试资产的可维护性和可演化性, 易化在软件变更过程中不断增长的包括回归测试在内的各种测试活动, 适应现代系统高频发布的研发需求. 除此以外, 这种方法突出的优势在于能够推理测试方法本身的正确性, 因而适用于苛求正确的安全攸关系统.

本文第1节描述了研究背景和动机; 第2节比较了本文所作工作与其它相关工作的异同; 第3节定义了相关的形式化建模方法, 算子和关键关系; 第4节建立了基于一致性测试的变更后安全性回归验证的理论基础; 第5节基于第4节中的结论构建了可用于系统安全性回归验证的测试生成框架; 第6节以列车自动防护功能的规约变更为研究案例, 说明了本文所提出方法的具体应用; 最后, 对全文进行了总结并指出了下一步的研究工作.

2 相关工作

基于形式化的测试根植于严密的数学定义和推理, 因此适合于苛求代码正确性的各种测试活动.

考虑到在维护阶段代码可能缺失的情况, 我们在研究中采用了不依赖于代码实现的“黑盒”方法. 具体地说, 本文的主要研究工作是基于形式化方法, 将变更后实时系统实现的安全性回归验证归结为基于系统规约变更的一致性测试. 从理论上, 为了说明方法本身的正确性, 需要包含下面 2 个工作: 1) 定义与研究内容相关的实时系统一致性关系; 2) 建立验证为正确的实时系统规约与一致性回归测试之间的关系. 从方法应用角度说, 需要建立与理论相应的回归测试生成方法. 本节从这几个方面入手, 陈述并比较相关工作.

基于形式化的测试研究包含基于形式化规约模型的测试 (testing from model-based formal specification)、基于有限状态的测试 (finite-state-based testing)、基于标号迁移系统的测试 (testing based on labeled transition system) 等分支 [8]. 本文提出的测试方法本质上归属于基于标号迁移系统的测试, 该类测试的核心是定义系统规约与相关系统实现集合之间的一致性关系. Tretmans [9] 在这方面作了开创性工作, 提出了用于构造非时间系统测试生成方法的一致性关系集合. 这些一致性关系依据测试用例在被测系统上可观测输出的不同而有所区别 [8]. 在这些一致性关系集合中被后续研究人员广泛研究和采用的是输入输出一致性关系 (input output conformance relation, ioco), 其核心是将系统在给定时间内的无输出行为定义为一种可观测行为, 即静止 (quiescence). 不同于非时间系统, 由于安全攸关系统的强实时特性, 以超时为本质特征的静止行为在实时系统一致性关系定义中的合理性和可用性受到了质疑. 目前存在两种意见: 一种认为所有延迟都应都是可观测的, 而另一种则认为只有小于给定标准的延迟是可观测的 [10]. 比较有代表性的工作包括 Briones 等 [11] 提出的参数化实时输入输出一致性关系 $tioco_M$, Larson 等 [12] 提出的相对时间输入输出一致性关系 $rtioco$, Krichen 等 [10,13] 提出的基于部分观测和非确定性系统规约的实时输入输出一致性关系 $tioco_{gre}$, 以及 Bohnenkamp 等 [14] 提出的实时输入输出一致性关系 $tioco$. 在这些一致性关系中, 除了 $tioco_M$ 属于延迟受限的关系外, 其余三者都属于延迟不受限的关系. 虽然这些关系定义的出发点和方式不同, 但是在满足相关条件下, 这些关系之间可以进行等价转换 [10]. 由于考虑到一方面我们所面临实际的研究对象中不存在非确定性问题, 另一方面实际的测试活动会受到人力物力等资源的客观限制, 在测试过程中长时间的等待对于实时系统的测试没有任何意义, 因此, 在本文中采用了延迟受限的实时输入输出一致性关系, 其本质是一种特殊的参数化实时输入输出一致性关系.

系统设计规约是测试生成的依据之一, 因此, 在测试生成前需要首先保证设计规约的正确性. 这个客观事实为我们提出了一个问题: 基于已证明为安全的系统设计规约生成的测试, 是否可以用于说明系统实现也是正确的? Gargantini 等 [15] 通过对被验证属性取反作为验证属性, 使用模型检测工具, 在验证为正确的设计规约上计算反例作为违背系统属性的测试用例. 与此相反, Ammann 等 [16] 通过对验证为正确的设计规约进行变异, 使用模型检测工具在变异体上找到违反属性的执行序列, 并将其作为可验证系统实现是否为正确的测试用例. 由于在测试生成过程中需要分析变量的值, Ammann 的方法并不完全是一个黑盒方法. Rusu 等 [17,18] 通过迹包含和非实时系统的输入输出一致性关系, 在系统规约安全性验证和一致性测试之间建立了相应的关系, 讨论了如何在非实时系统中解答这个问题, 其方法基于部分输入完备的系统规约并且支持不确定性. 更进一步, 为了提高该方法的可用性和通用性, Constant 等 [19] 扩展了 Rusu 的方法, 使得这种方法不仅支持安全性验证同样也能够支持活性验证. 另外, Constant 等采用了输入输出符号转移系统 (input/output symbolic transition system, IOSYS) 作为系统行为的形式化语言, 加入了变量和参数等建模元素, 较之以 Rusu 方法中的基于标号迁移系统的形式化语言而言, 该方法降低了建模的难度和复杂度. 上述研究工作都有一个统一的问题就是不能处理系统的实时特性, 而实时特性对于安全攸关系统而言非常重要. 本文基于时间受限的输入输出一致性关系, 以时间输入输出自动机 (timed input/output automata, TIOA) 为系统行为建模语言, 构

建了一个基于一致性测试的系统实现安全性验证的通用理论框架, 并依据代码变更和规约变更等不同的变更类型将该框架扩展到系统实现的安全性回归验证活动上. 但是, 我们的方法不支持活性验证并且要求系统规约必须是输入完备且确定的, 为此, 在方法中增加了系统规约的可测化计算.

测试用例选择 (test case selection)、测试用例修复 (test case repairing) 和测试用例扩增 (test case augmentation) 是目前回归测试研究领域的主要方向. 其中, 前两者重点关注基线测试用例的有效再利用问题, 也是过去多年间, 研究人员主要关注的方面^[20,21]. 然而, 在系统变更过程中还存在着相当数量的新增类变更, 为了验证这些新增类变更的正确性, 只局限于基线测试用例集合显然是不适宜的, 新的测试用例必须被扩充进来. 在这种背景下, 以全面关注演化软件的测试需求 (未变的、修改的和新增的) 为主要目标的测试用例扩增技术正在逐渐成为研究人员的关注点^[22], 其技术核心是如何识别针对变更并可将变更影响传播至输出的测试用例集合. 这包括两方面任务: 1) 识别变更元素; 2) 生成或指导可以执行这些变更元素的测试用例^[23]. Santelices 等^[24,25] 通过构造基于变更前后系统实现的依赖图 (program dependence graph) 识别长度从 1 到指定最大长度 k 的变更传播链 (chain) 作为变更测试需求 (change-testing requirements), 并以此作为生成测试用例的依据, 获取能够将变更产生影响传播至输出的测试用例集. 与 Santelices 等的方法相对, Taneja 等^[26] 通过定义与变更无关的路径类型, 提出了可用于避免遍历变更不相关路径的动态符号执行方法, 达到生成有效回归测试用例集合的目的. Qi 等^[27] 则通过识别两种无法将变更传播至输出的变更实现模式, 将基于动态符号执行获取的变更可达输入 (change-reaching input) 转换成能够将变更产生影响传播至输出的测试用例. Xu 等^[28] 从研究影响测试用例扩增的性能因子入手, 并根据相关结论, 提出了通过混合动态符号执行和遗传算法, 利用基线测试用例引导生成满足分支覆盖的扩增用例生成方法. 与上述依据变更前后系统实现工作不同的是, 还有一些研究工作是基于更高层次的设计规约. 相对于作用于系统实现层面上的方法而言, 基于设计规约的方法并不要求变更被实现, 有利于缺陷的早期发现. Jiang 等^[29] 基于基线测试用例图 (original test case graph) 和变更后模型程序 (model program) 规约图, 通过判定变更后实现被基线测试用例覆盖的情况确定可重用的基线测试用例和需要新增的测试用例. Korel 等^[30] 利用扩展的有限状态机 (extended finite state machine, EFSM) 构造系统变更行为依赖图 (dependence graph), 构造了迁移增加和迁移删除两种情况下, 利用受限路径选择策略生成和约减回归测试用例的方法. Beydeda 等^[31] 首先通过构造类规约实现图 (class specification implementation graph, CSIG) 建模对象行为的控制依赖和数据依赖, 然后根据变更前后对象行为 CSIG, 在基线测试用例中选择可用部分作为回归测试用例. Farooq 等^[32] 通过在元模型 (meta model) 上定义类变更 (class change) 和状态变更 (state change), 并在迁移树上搜索被影响的测试路径, 获取相应的回归测试用例集合. 后两者的工作本质上仍旧局限于在基线测试用例中选择相关可用测试用例子集, 并没有涉及测试新增变更的讨论. 在本文中, 通过将 TIOA 转换为时间序列树判定受变更影响的系统行为, 不仅包括修改的部分同时也包含新增的内容.

3 实时系统实现的安全性验证

为了建立基于形式化测试的系统变更后安全性验证方法的理论框架, 需要首先定义能够在各抽象层次上描述系统行为的建模语言及其语义, 并在此基础上构造一般情况下系统实现的安全性验证方法. 为此, 在本节中, 基于我们前期的工作^[33], 首先说明了定义系统实时行为的建模语言及其相关算子; 接着利用迹包含形式化定义了安全性验证和时间受限的输入输出一致性关系; 最后, 构造了一般

情况下时间受限的一致性关系与安全性验证之间的关联, 并由此得到指导系统实现安全性验证的相关定理.

3.1 实时系统建模

时间自动机 (timed automata)^[34] 是一种被广泛采用的实时系统行为建模方法. 时间输入输出自动机 (TIOA)^[35] 是经典时间自动机的一个变体. 它与经典时间自动机相区别的方面在于, 为了反映实时系统与环境交互时, 系统对环境输入的非阻塞性和环境对系统输出的非阻塞性, 在 TIOA 中, 动作集合被划分为输入动作集合和输出动作集合.

定义 1 时间输入输出自动机. 时间输入输出自动机可形式化的定义为一个六元组 $(L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$, 其中

- L 是有限的位位置集合, $l_0 \in L$ 是初始位置;
- C 是时钟变量集合, $\Phi(C)$ 是定义在 C 上的时钟约束集合, 时钟约束的定义须符合下列 BNF 范式:

$$\varphi := \text{true} \mid x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2, x \in C, c \in \mathbb{Q};$$

- $\Sigma = \Sigma_I \cup \Sigma_O \cup \Sigma_\tau$ 且 $\Sigma_I \cap \Sigma_O \cap \Sigma_\tau = \emptyset$ 是动作集合, Σ_I 是输入动作集合, Σ_O 是输出动作集合, Σ_τ 是内部动作集合, Σ_I 和 Σ_O 是从外部观测到的行为集合;
- $\text{Inv} : L \rightarrow \Phi(C)$ 是不变式集合, 定义了位置上可流逝的时间;
- $\rightarrow \subseteq L \times \Phi(C) \times \Sigma \times R(2^C) \times L$ 是迁移关系, $l \xrightarrow{\varphi, a, \gamma} l'$ 表示当时钟约束 φ 为真时, 发生动作 a , 时钟 $c \in \gamma$ 被重置, 系统从 l 迁移到 l' .

时间输入输出自动机的语义由时间输入输出标记迁移系统 (timed input output labeled transition system) 定义^[12]. TIOTS 是经典标签迁移系统的一个扩展, 将非负实数加入标签集合中并且将活动集合划分为输入和输出, 其形式化定义如下描述.

定义 2 一个时间输入输出标记迁移系统 TIOTS $\mathcal{S}$ 是一个四元组 (S, s_0, Σ, E) , 其中

- S 是状态的集合, $s_0 \in S$ 是初始状态;
- $\Sigma = \Sigma_I \cup \Sigma_O \cup \Sigma_\tau$ 是动作集合并且满足 $\Sigma_I \cap \Sigma_O \cap \Sigma_\tau = \emptyset$, Σ_I , Σ_O 和 Σ_τ 分别是输入动作, 输出动作和内部动作集合;
- $E : S \times \Sigma \cup R_{\geq 0} \times S$ 定义了状态之间的迁移关系, 包含离散迁移 $E_a = \{s \xrightarrow{a} s' \mid a \in \Sigma\}$ 和延迟迁移 $E_d = \{s \xrightarrow{d} s' \mid d \in R_{\geq 0}\}$, 并且延迟迁移需满足下列约束
 - 1) 时间确定性: $d \in R_{\geq 0}$, if $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$ then $s' = s''$;
 - 2) 时间可加性: $d_1, d_2 \in R_{\geq 0}$, if $s \xrightarrow{d_1} s'$ and $s' \xrightarrow{d_2} s''$ then $s \xrightarrow{d_1+d_2} s''$;
 - 3) 零延迟: $\forall s \in S, s \xrightarrow{0} s$.

一个时间输入输出自动机 $\mathcal{T} : (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$ 的 TIOTS 语义可定义为 $\mathcal{S}_{\mathcal{T}} = (S, s_0, \Sigma, E)$, 其中

- $S = \{(l, \bar{v}) \mid l \in L, \bar{v} : C \rightarrow R_{\geq 0}^{|C|} \wedge \text{Inv}(l)(\bar{v})\}$, $\bar{v}$ 是满足 l 不变式的一个时钟解释, $s_0 = (l_0, \bar{v}_0)$, $\bar{v}_0$ 可解释为所有时钟变量取值为 0.
- 迁移关系 $E = E_d \cup E_a$, 其中, 延迟迁移 E_d 和离散迁移 E_a 分别为

$$\frac{d' \in R_{\geq 0}, \forall d' \leq d, \text{Inv}(l)(\bar{v} + d')}{(l, \bar{v}) \xrightarrow{d} (l, \bar{v} + d)}, \quad \frac{l \xrightarrow{\varphi, a, \gamma} l' \wedge \varphi(\bar{v}) \wedge \text{Inv}(l')(\bar{v}'), \bar{v}' = \bar{v}[\gamma := 0]}{(l, \bar{v}) \xrightarrow{a} (l', \bar{v}')}.$$

由于时间的客观存在, 因此合理的时间输入输出自动机必须具备不能阻碍时间流逝的特性, 这被称为时间非阻塞性. 本文讨论的时间输入输出自动机都是时间非阻塞的. 若时间输入输出自动机的每个状态对于每一个可见输入都有迁移定义, 则称该时间输入输出自动机是弱输入完备的. 相对于弱输入完备性, 在实时嵌入式领域中还存在强输入完备性. 两者之间的区别在于, 强输入完备性不仅需要保证系统行为对可见输入的动作响应定义还考虑了系统的内部行为结果. 由于一致性测试仅考虑可见行为, 因此, 在本文后续的论述中所指的输入完备性均指弱输入完备性. 若时间输入输出自动机的每个迁移都有唯一的迁入状态, 则称其是确定的. 一个时间输入输出自动机 $\mathcal{T}$ 满足时间非阻塞性、弱输入完备性和确定性当且仅当其 TIOTS 语义模型 $\mathcal{S}_{\mathcal{T}}$ 满足时间非阻塞性, 弱输入完备性和确定性.

系统是否正确取决于系统运行时的行为表现是否满足要求. 执行路径是系统运行时行为的形式化描述, 是判断系统运行时行为正确与否的基础. 对于时间输入输出自动机 $\mathcal{T}$ 而言, 从初始状态开始, 由状态与延迟或者状态与可见行为构成的交替对序列 $(l_0, \bar{v}_0) \cdot ((\Sigma - \Sigma_{\mathcal{T}}) \cup R_{\geq 0} \cdot (l, \bar{v}))^*$, 叫做 $\mathcal{T}$ 的执行路径. 约定用 $\Pi(\mathcal{T})$ 表示 $\mathcal{T}$ 的执行路径集合. 对于 $\pi \in \Pi(\mathcal{T})$, 其在可见行为和非负实数集上的映射 $\rho = \pi|_{(\Sigma - \Sigma_{\mathcal{T}}) \cup R_{\geq 0}}$ 称为 $\mathcal{T}$ 的一条时间迹, $\text{tTr}(\mathcal{T})$ 是 $\mathcal{T}$ 的时间迹集合^[11]. 形如 $(R_{\geq 0} \cdot \Sigma - \Sigma_{\mathcal{T}})^* \cdot (\epsilon \cup R_{\geq 0})$ 的时间迹被称为规范时间迹. 根据时间输入输出标号迁移系统的可加性, 可以证明任何一条时间迹都可被转换成一条与之等价的规范时间迹^[36]. 因此, 在本文中所指的时间迹均为规范时间迹.

3.2 时间输入输出自动机的同步组合

同步组合是用于计算系统各个部件之间同步交互行为的形式化算子.

定义 3 兼容性. 若时间输入输出自动机 $\mathcal{T}_1 : (L_1, l_{10}, C_1, \Sigma_1, \text{Inv}_1, \rightarrow_1)$ 和 $\mathcal{T}_2 : (L_2, l_{20}, C_2, \Sigma_2, \text{Inv}_2, \rightarrow_2)$ 满足 $C_1 \cap C_2 = \emptyset$, $\Sigma_1 - \Sigma_{1\tau} = \Sigma_2 - \Sigma_{2\tau}$ 并且 $\Sigma_{1\tau} \cap \Sigma_{2\tau} = \emptyset$, 则说 $\mathcal{T}_1$ 和 $\mathcal{T}_2$ 是互相兼容的.

定义 4 同步组合 $\parallel$. 若 $\mathcal{T}_1 : (L_1, l_{10}, C_1, \Sigma_1, \text{Inv}_1, \rightarrow_1)$ 和 $\mathcal{T}_2 : (L_2, l_{20}, C_2, \Sigma_2, \text{Inv}_2, \rightarrow_2)$ 是两个兼容的时间输入输出自动机, 则其同步组合 $\mathcal{T}_1 \parallel \mathcal{T}_2 = (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$, 其中, $L = L_1 \times L_2$, $l_0 = (l_{10}, l_{20})$, $C = C_1 \cup C_2$, $\Sigma = \Sigma_1 \cup \Sigma_{2\tau} = \Sigma_2 \cup \Sigma_{1\tau}$, $\text{Inv}(l_i, l_j) = \text{Inv}_1(l_i) \wedge \text{Inv}_2(l_j)$, 其中, $l_i \in L_1$, $l_j \in L_2$, 迁移关系 $\rightarrow$ 由下列规则定义:

- $\frac{(l_1, \varphi_1, a, \gamma_1, l'_1) \in \rightarrow_1, a \in \Sigma_{1\tau}, l_2 \in L_2}{((l_1, l_2), \varphi_1, a, \gamma_1 \cup \{c := c\}_{(c \in C_2)}, (l'_1, l_2)) \in \rightarrow}$ 或者 $\frac{(l_2, \varphi_2, a, \gamma_2, l'_2) \in \rightarrow_2, a \in \Sigma_{2\tau}, l_1 \in L_1}{((l_1, l_2), \varphi_2, a, \gamma_2 \cup \{c := c\}_{(c \in C_1)}, (l_1, l'_2)) \in \rightarrow};$
- $\frac{(l_1, \varphi_1, a, \gamma_1, l'_1) \in \rightarrow_1, (l_2, \varphi_2, a, \gamma_2, l'_2) \in \rightarrow_2}{((l_1, l_2), \varphi_1 \wedge \varphi_2, a, \gamma_1 \cup \gamma_2, (l'_1, l'_2)) \in \rightarrow}.$

引理 1 若时间输入输出自动机 $\mathcal{T}_1 : (L_1, l_{10}, C_1, \Sigma_1, \text{Inv}_1, \rightarrow_1)$ 和 $\mathcal{T}_2 : (L_2, l_{20}, C_2, \Sigma_2, \text{Inv}_2, \rightarrow_2)$ 是兼容的, 则有 $\text{tTr}(\mathcal{T}_1 \parallel \mathcal{T}_2) = \text{tTr}(\mathcal{T}_1) \cap \text{tTr}(\mathcal{T}_2)$.

证明 $(\Rightarrow)$. $\forall d_1 a_1 d_2 a_2 d_3 a_3 \cdots \in \text{tTr}(\mathcal{T}_1 \parallel \mathcal{T}_2)$, $d_i \in R_{\geq 0}$, $a_i \in \Sigma_1 - \Sigma_{1\tau}$, 假设 $d_1 a_1 d_2 a_2 d_3 a_3 \cdots$ 的执行路径是 $\pi: (l_{10}, l_{20}, \bar{v}_0) \xrightarrow{d_1} (l_{10}, l_{20}, \bar{v}_1) \xrightarrow{a_1} (l_{11}, l_{21}, \bar{v}_2) \xrightarrow{d_2} (l_{11}, l_{21}, \bar{v}_3) \xrightarrow{a_2} (l_{12}, l_{22}, \bar{v}_4) \xrightarrow{d_3} (l_{12}, l_{22}, \bar{v}_5) \xrightarrow{a_3} (l_{13}, l_{23}, \bar{v}_6) \xrightarrow{\cdots} \cdots$, 其中, $(l_{1i}, l_{2j}) \in L_1 \times L_2$. 假设 $\bar{v}$ 是时钟集合 C 的解释, 由于 $C = C_1 \cup C_2$, 因此, 可以将 $\bar{v}$ 分解为 $\bar{v}_1$ 和 $\bar{v}_2$.

1) 对于 π 中的任一离散迁移: $(l_{1i}, l_{2j}, \bar{v}) \xrightarrow{a_k} (l_{1i'}, l_{2j'}, \bar{v}')$, 根据同步组合迁移计算规则, 可得必然存在 $l_{1i} \xrightarrow{\varphi_1, a_k, \gamma_1} l_{1i'} \in \rightarrow_1$ 和 $l_{2j} \xrightarrow{\varphi_2, a_k, \gamma_2} l_{2j'} \in \rightarrow_2$, 分别满足 $\frac{l_{1i} \xrightarrow{\varphi_1, a_k, \gamma_1} l_{1i'} \wedge \varphi(\bar{v}_1) \wedge \text{Inv}(l_{1i'}) (\bar{v}_1), \bar{v}_1 = \bar{v}_1 [\gamma_1 := 0]}{(l_{1i}, \bar{v}_1) \xrightarrow{a_k} (l_{1i'}, \bar{v}_1')}$ 和 $\frac{l_{2j} \xrightarrow{\varphi_2, a_k, \gamma_2} l_{2j'} \wedge \varphi(\bar{v}_2) \wedge \text{Inv}(l_{2j'}) (\bar{v}_2), \bar{v}_2 = \bar{v}_2 [\gamma_2 := 0]}{(l_{2j}, \bar{v}_2) \xrightarrow{a_k} (l_{2j'}, \bar{v}_2')}$, 即有 $(l_{1i}, \bar{v}_1) \xrightarrow{a_k} (l_{1i'}, \bar{v}_1')$ 和 $(l_{2j}, \bar{v}_2) \xrightarrow{a_k} (l_{2j'}, \bar{v}_2')$ 成立.

2) 对于 π 中的任一延迟迁移: $(l_{1i}, l_{2j}, \bar{v}) \xrightarrow{d_k} (l_{1i}, l_{2j}, \bar{v}')$, 根据 TIOTS 的语义定义, 该迁移的 $d' \in R_{\geq 0}, \forall d' \leq d_k, \text{Inv}(l_{1i}, l_{2j}) (\bar{v} + d')$. 注意到 $\text{Inv}(l_{1i}, l_{2j}) = \text{Inv}(l_{1i}) \wedge \text{Inv}(l_{2j})$, 因此, 有 $\frac{d' \in R_{\geq 0}, \forall d' \leq d_k, \text{Inv}(l_{1i}) (\bar{v}_1 + d')}{(l_{1i}, \bar{v}_1) \xrightarrow{d_k} (l_{1i}, \bar{v}_1 + d)}$ 和 $\frac{d' \in R_{\geq 0}, \forall d' \leq d_k, \text{Inv}(l_{2j}) (\bar{v}_2 + d')}{(l_{2j}, \bar{v}_2) \xrightarrow{d_k} (l_{2j}, \bar{v}_2 + d)}$.

和 $\frac{d' \in R_{\geq 0}, \forall d' \leq d_k, \text{Inv}(l_{2j})(\overline{v_2} + d')}{(l_{2j}, \overline{v_2}) \xrightarrow{d_k} (l_{2j}, \overline{v_2} + d)}$ 成立. 即迁移 $(l_{1i}, \overline{v_1}) \xrightarrow{d_k} (l_{1i'}, \overline{v_1'})$ 和 $(l_{2j}, \overline{v_2}) \xrightarrow{d_k} (l_{2j'}, \overline{v_2'})$ 存在.

综合以上两点可得执行路径 $\pi_1: (l_{10}, \overline{v_{10}}) \xrightarrow{d_1} (l_{10}, \overline{v_{11}}) \xrightarrow{a_1} (l_{11}, \overline{v_{12}}) \xrightarrow{d_2} (l_{11}, \overline{v_{13}}) \xrightarrow{a_2} (l_{12}, \overline{v_{14}}) \xrightarrow{d_3} (l_{12}, \overline{v_{15}}) \xrightarrow{a_3} (l_{13}, \overline{v_{16}}) \xrightarrow{\dots} \dots$, 和执行路径 $\pi_2: (l_{20}, \overline{v_{20}}) \xrightarrow{d_1} (l_{20}, \overline{v_{21}}) \xrightarrow{a_1} (l_{21}, \overline{v_{22}}) \xrightarrow{d_2} (l_{21}, \overline{v_{23}}) \xrightarrow{a_2} (l_{22}, \overline{v_{24}}) \xrightarrow{d_3} (l_{22}, \overline{v_{25}}) \xrightarrow{a_3} (l_{23}, \overline{v_{26}}) \xrightarrow{\dots} \dots$ 存在. 因此, $d_1 a_1 d_2 a_2 d_3 a_3 \dots \in \text{tTr}(\mathcal{T}_1) \cap \text{tTr}(\mathcal{T}_2)$. 从而可得, $\text{tTr}(\mathcal{T}_1 \parallel \mathcal{T}_2) \subseteq \text{tTr}(\mathcal{T}_1) \cap \text{tTr}(\mathcal{T}_2)$.

($\Leftarrow$) 的证明过程与 ($\Rightarrow$) 类似, 只是先使用语义定义再使用同步组合的迁移计算规则, 考虑到版面问题, 在此不再赘述.

3.3 安全性观察者

在形式化验证领域, 安全性属性通常采用时序逻辑的形式进行描述. 在本文中, 为了建立安全性验证和一致性测试之间的关联, 我们采用一种特殊类型的时间输入输出自动机, 即安全性观察者, 描述期望系统满足的安全性属性.

定义 5 安全性观察者. 时间输入输出自动机 $\mathcal{T} : (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$ 的一个安全性观察者是—个确定的, 输入完备的时间输入输出自动机 $\varphi = (L^o \cup \{\text{Violated}\}, l_0^o, C^o, \Sigma - \Sigma_\tau, \text{Inv}^o, \rightarrow^o)$, 其中 $\{\text{Violated}\}$ 是一个没有时间含义的决断位置, 表示被验证系统行为不满足该属性, $C^o \wedge C = \emptyset$. $\Omega(\mathcal{T})$ 表示 $\mathcal{T}$ 的安全属性集合.

定义 6 可识别时间迹. 对于时间输入输出自动机 $\mathcal{T}$ 和一个安全性观察者 φ , 若 L_v 表示决断位置集合, 并且 $L \cap L_v = \emptyset$, 对于 $l_v \in L_v$, l_v 的可识别时间迹定义为 $\text{tTr}(\mathcal{T}, l_v) =_{\text{def}} \{\sigma \mid \sigma \in \text{tTr}(\mathcal{T}) \wedge (l_0, \overline{v_0}) \xrightarrow{\sigma} l_v\}$.

安全性属性用于描述包含有“坏事情永远不会发生”含义的系统属性. 从系统行为的角度理解, 即系统时间迹集合中不存在可以被安全性观察者决断位置 Violated 识别的时间迹. 由此, 可得安全性验证的形式化定义.

定义 7 安全性验证. 对于时间输入输出自动机 $\mathcal{T}$ 和安全性属性 φ , 符号 $\mathcal{T} \models \varphi$ 表示 $\mathcal{T}$ 满足 φ , 其定义为

$$\mathcal{T} \models \varphi =_{\text{def}} \text{tTr}(\mathcal{T}) \cap \text{tTr}(\varphi, \text{Violate}) = \emptyset,$$

由引理 1 可知 $\text{tTr}(\mathcal{T}) \cap \text{tTr}(\varphi, \text{Violate}) = \text{tTr}(\mathcal{T} \parallel \varphi, \text{Violated})$. 这个结论是后续安全性测试生成的基础之一.

3.4 时间受限的输入输出一致性关系

在由形式化支持的测试生成中, 一致性关系是判定实现相对于规约是否正确的数学基础. 因此, 为了确定实时系统实现是否正确, 需要首先定义可用于描述实时系统的一致性关系. 输入输出一致性关系 (input output conformance relation, ioco) 是非时间系统的经典代表. 考虑到实际的测试活动都会受到进度和成本的约束, 同时, 当系统长时间没有输出时, 测试人员的无休止等待没有任何意义, 因此, 在我们的测试生成方法中选择了时间受限的输入输出一致性关系. 时间受限的输入输出一致性关系 (time-bounded input output conformance relation, tioco) 是 ioco 在实时系统上的一种扩展, 其本质是采用了时间受限的静止状态. 静止状态是指系统不输出任何可见行为时的情况, 是定义一致性关系的基础和核心. 实时系统中的静止是指在给定允许的最大延迟后, 系统没有输出任何行为的行为. 这种行为由系统中的静止状态引起. 在实时系统中, 没有输出迁移的状态、等待输入行为的状态、进入内

部循环的状态和不受时间约束的状态都是潜在的静止状态. 若使用 λ 表示给定实时系统所允许的最大延迟, 则时间输入输出标号系统 $\mathcal{S}$ 的静止状态集合 $\delta^\lambda(\mathcal{S})$ 可以定义为 $\delta^\lambda(\mathcal{S}) = \{s \in S \mid \forall a \in \Sigma_I, \forall d \in R_{\geq 0} \wedge d > \lambda : s \not\xrightarrow{da}\}$.

非实时系统的一致性关系 *ioco*, 以在任何情况下 (包括系统静止), 系统实现是否总是输出系统规约所指定的输出行为为其核心基础. 因此, 与 *ioco* 类似, 实时系统的输入输出一致性关系意味着对于任意在系统规约中存在的时间迹, 包含存在静止行为的时间迹, 被评估系统实现总是在规定时间内输出系统规约期望的行为或者系统规约中允许的静止行为. 若 $\mathcal{S}_S$ 表示系统规约的时间输入输出标号系统, $\mathcal{S}_I$ 表示某满足弱输入完备性的系统实现的时间输入输出标号系统, 并且 $\Sigma_{\mathcal{S}_S} = \Sigma_{\mathcal{S}_I}$, 假设系统允许的最大延迟为 λ , 那么 λ 受限的输入输出一致性关系可以定义为

$$\mathcal{S}_I \text{ tioco}^\lambda \mathcal{S}_S \Leftrightarrow \forall \sigma \in \text{tTr}(\mathcal{S}_S^{\delta^\lambda}), \text{Out}_t^\lambda(\mathcal{S}_I^{\delta^\lambda} \text{ After}_t \sigma) \subseteq \text{Out}_t^\lambda(\mathcal{S}_S^{\delta^\lambda} \text{ After}_t \sigma).$$

用自然语言可以解释为, 系统实现 $\mathcal{S}_I$ 相对于系统规约 $\mathcal{S}_S$ 而言是一致的, 当且仅当, 对任意包含静止行为的系统规约的迹 σ 而言, 在包含静止行为的系统实现执行过路径 σ 后所达状态的输出总是在包含静止行为的系统规约定义的输出范围内. 其中, $\mathcal{S}_S^{\delta^\lambda}$ 和 $\mathcal{S}_I^{\delta^\lambda}$ 分别表示系统规约和系统实现的 λ 受限的输入输出标号迁移系统 (λ -bounded TIOTS). 对于 TIOTS $\mathcal{S}$, 其 λ 受限的时间输入输出标号迁移系统 $\mathcal{S}^{\delta^\lambda}$ 是通过在原系统的静止状态上增加在时间流逝 λ 时输出静止行为的自迁移而获得, 即增加迁移关系 $E^\delta = \{s \xrightarrow{\lambda\delta!} s \mid s \in \delta^\lambda(\mathcal{S})\}$. 算子 $S\text{After}_t \sigma$ 用于计算对于任意 $s \in S$, 从状态 s 开始沿时间迹 σ 执行可达的所有状态. 算子 $\text{Out}_t^\lambda(S)$ 用于计算 $s \in S$ 的所有输出行为以及这些行为发生前的时间流逝. 值得一提的是, 由于现实情况下, 设计人员所定义模型通常是部分输入完备的, 因此, 在 tioco^λ 的定义中并没有要求系统规约是输入完备的模型. 基于时间输入输出标号迁移系统的一致性关系, 接下来我们构造用于描述时间输入输出自动机的一致性关系.

定义 8 λ 受限的静止位置. 给定实时系统所允许的最大延迟 λ , 时间输入输出自动机 $\mathcal{T} = (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$, 对于 $l \in L$, 若 $(l, \bar{v}) \in \delta^\lambda(\mathcal{T})$, 则说 l 是 $\mathcal{T}$ 的 λ 受限的静止位置, 表示为 l^δ . $\mathcal{T}$ 的 λ 受限的静止位置集合表示为 $\delta^\lambda(\mathcal{T})$.

定义 9 λ 受限的时间输入输出自动机 (λ -bounded TIOA). 时间输入输出自动机 $\mathcal{T} = (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$ 的 λ 受限时间输入输出自动机为 $\mathcal{T}^{\delta^\lambda} = (L, l_0, C^{\delta^\lambda}, \Sigma \cup \delta, \text{Inv}, \rightarrow^{\delta^\lambda})$, 其中

- $C^{\delta^\lambda} = C \cup \{x^\delta\}$, $x^\delta \notin C$ 是用于约束静止行为 δ 的时钟;
- $\rightarrow^{\delta^\lambda} = \rightarrow' \cup \rightarrow^\delta$:
- 1) $\rightarrow' = \{l \xrightarrow{\varphi, a, \gamma \cup x^\delta := 0} l^\delta \mid l^\delta \in \delta^\lambda(\mathcal{T})\} \cup \{l \xrightarrow{\varphi, a, \gamma} l' \mid l' \notin \delta^\lambda(\mathcal{T})\}$;
- 2) $\rightarrow^\delta = \{l^\delta \xrightarrow{x^\delta > \lambda, \delta!, x^\delta := 0} l^\delta \mid l^\delta \in \delta^\lambda(\mathcal{T})\}$.

λ 受限的时间输入输出自动机旨在建模任意两个连续行为之间的时间延迟不能超过 λ . 由定义可知, 基于 $\mathcal{T}$ 构造其 λ 受限的输入输出自动机包含两个部分, 第一部分是在原有时钟集合 C 上增加用于约束静止行为 δ 的逻辑时钟 x^δ ; 第二部分是修改迁移关系, 对于静止位置 $l^\delta \in \delta^\lambda(\mathcal{T})$ 需要增加受卫士条件 $x^\delta > \lambda$ 约束的静止行为的自迁移并设置相关的时钟重置.

定义 10 若 $\mathcal{T}_S^{\delta^\lambda}$ 表示系统规约 $\mathcal{T}_S$ 的 λ 受限输入输出自动机, $\mathcal{T}_I^{\delta^\lambda}$ 表示系统实现的 $\mathcal{T}_I$ 的 λ 受限输入输出自动机, 那么, $\mathcal{T}_I \text{ tioco}^\lambda \mathcal{T}_S =_{\text{def}} \text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cdot ((R_{\geq 0} \cdot \Sigma_I) \cup \{\lambda \cdot \delta\}) \cap \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$.

从直觉的角度理解, 某个系统实现与系统规约一致意味着实现的输出总是被规约所允许. 同样, 在该定义中系统规约可以是部分完备的. 然而, 作为设计模型的系统规约是可以部分完备的, 但是作为生成测试用例的测试模型必须是输入完备的, 这是模型具备可测性的要求之一.

引理 2 若 $\mathcal{T}_S$ 是输入完备的系统规约, $\mathcal{T}_I$ 是系统实现, $\mathcal{T}_S$ 和 $\mathcal{T}_I$ 具有相同的时间不变式, 则

$$\mathcal{T}_I \text{ tioco}^\lambda \mathcal{T}_S \Leftrightarrow \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda}).$$

证明 ($\Rightarrow$). 用反证法. 假设 $\exists \sigma \in \text{tTr}(\mathcal{T}_I^{\delta^\lambda})$ 并且 $\sigma \notin \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 由此可得, 存在 $\sigma_1, \sigma_2, a \in \Sigma - \Sigma_\tau$ 和 a 的延迟 $d \in R_{\geq 0}$ 且 $0 \leq d < \lambda$, 使得 $\sigma = \sigma_1 \widehat{da} \sigma_2$. $\sigma_1 \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$ 并且 $\sigma_1 \widehat{da} \notin \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$ 成立:

1) 考虑当 a 是输入行为时, 即 $a \in \Sigma_\tau$. 由于系统规约 $\mathcal{T}_S^{\delta^\lambda}$ 是输入完备的, 因此, $\exists 0 \leq d' < \lambda, \sigma_1 d' a \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 又考虑到 $\mathcal{T}_I$ 和 $\mathcal{T}_S$ 有相同的时间不变式, 因此, 一定存在一个时间延迟 d' 满足 $d' = d$, 由此得到, $\sigma_1 \widehat{da} \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 与假设矛盾.

2) 考虑 a 是输出行为或者是静止行为时, 即 $a \in \Sigma_I \cup \{\delta\}$. 由假设 $\sigma = \sigma_1 \widehat{da} \sigma_2 \in \text{tTr}(\mathcal{T}_I^{\delta^\lambda})$ 可得 $\sigma_1 \widehat{da} \in \text{tTr}(\mathcal{T}_I^{\delta^\lambda})$, 注意到在假设中成立的 $\sigma_1 \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 并且 $a \in \Sigma_I \cup \{\delta\}$, 根据 $\mathcal{T}_I \text{ tioco}^\lambda \mathcal{T}_S$ 的定义 $\text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cdot ((R_{\geq 0} \cdot \Sigma_I) \cup \{\lambda \cdot \delta\}) \cap \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 可得 $\{\sigma_1 \cdot \widehat{da}\} \cap \{\sigma_1 \widehat{da}\} = \sigma_1 \widehat{da} \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 与假设矛盾.

($\Leftarrow$). 因为 $\text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cdot ((R_{\geq 0} \cdot \Sigma_I) \cup \{\lambda \cdot \delta\}) \cap \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_I^{\delta^\lambda})$, 并且根据已知条件 $\text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 可得 $\text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cdot ((R_{\geq 0} \cdot \Sigma_I) \cup \{\lambda \cdot \delta\}) \cap \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$. 根据 tioco^λ 的定义, 有 $\mathcal{T}_I \text{ tioco}^\lambda \mathcal{T}_S$ 成立.

在基于形式化的测试生成中, 一致性关系被定义为迹包含 (trace inclusion). 通常情况下, 模型检测是通过可达性分析检测模型是否满足期望属性, 但是, 这个问题可以同样使用迹包含说明. 因此, 通过迹包含我们可以在一致性测试和系统代码验证之间建立起连接. 给定输入完备的系统规约 $\mathcal{T}_S$ 和具有相同时间约束的某个系统实现 $\mathcal{T}_I$, φ 是系统期望满足的属性, 则有定理 1.

定理 1 $\mathcal{T}_I \text{ tioco}^\lambda \mathcal{T}_S \Leftrightarrow \forall \varphi \in \Omega(\mathcal{T}_S), \mathcal{T}_S^{\delta^\lambda} \models \varphi \Rightarrow \mathcal{T}_I^{\delta^\lambda} \models \varphi$.

证明 ($\Rightarrow$). 根据引理 2, 可得 $\mathcal{T}_I \text{ tioco}^\lambda \mathcal{T}_S \Rightarrow \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$. $\mathcal{T}_S \models \varphi$ 意味着 $\text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$. 因此, $\text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$, 根据前文给出的安全性验证的定义 (定义 7) 得 $\mathcal{T}_I^{\delta^\lambda} \models \varphi$.

($\Leftarrow$). 假设 $\mathcal{T}_I \not\text{tioco}^\lambda \mathcal{T}_S$, 根据输入输出时间自动机的 λ 受限一致性关系定义可得, $\exists \sigma \in (\Sigma - \Sigma_\tau \cup R_{\geq 0})^*$, $\sigma \in \text{tTr}(\mathcal{T}_I^{\delta^\lambda})$, 但是 $\sigma \notin \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 构造一个将 φ 作为其 Violated 可识别迹之一的观察者. 因此, 有 $\text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$, 即, $\mathcal{T}_S^{\delta^\lambda} \models \varphi$, 可是 $\text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\varphi) \neq \emptyset$, 即, $\mathcal{T}_I^{\delta^\lambda} \not\models \varphi$ 与已知条件矛盾.

通俗地讲, 定理 1 指出, 在系统允许的最大延迟 λ 下, 给定一个输入完备的系统规约及其相关的某个系统实现, 如果该系统实现与规约一致的话并且该规约满足指定的安全性属性, 那么系统实现也满足相同的安全性属性. 反之, 如果系统规约和系统实现都满足系统任意的安全属性, 那么, 系统实现与规约一致. 这个结论和一致性测试的本质相吻合. 因为, 一致性测试的本质含义为永远不出现不符合系统规约的行为, 而这本身就是一个安全性属性.

4 关联变更后测试和安全性验证

发布后变更是每个系统在其生命周期中必然存在的活动之一. 变更有多种分类方法, 本文站在变更输出产物的角度, 将变更分为两种: 代码变更和系统规约变更. 代码变更是指变更活动仅涉及系统实现代码的修改, 例如已发布系统缺陷修改, 被变更的代码称为基线代码. 系统规约变更是指系统功能性要求发生增, 删, 改等变动, 被变更的系统规约称为基线规约. 系统规约变更包含代码变更, 反之, 不成立. 在本文中, 如果代码变更需要修改系统规约, 则认为这种变更为系统规约变更.

定义 11 变更迹. 若时间输入输出自动机 $\mathcal{T}$ 的变更后模型为 $\mathcal{T}'$, 则 $\Delta tTr(\mathcal{T}) =_{\text{def}} tTr(\mathcal{T}') \setminus tTr(\mathcal{T})$, 称为 $\mathcal{T}$ 的变更迹集合. $\forall \sigma \in \Delta tTr(\mathcal{T})$ 称为 $\mathcal{T}$ 的变更迹.

变更迹的计算策略与变更操作相关. 考虑到嵌入式系统的交互具有序列特性, 因此, 本文除了识别增加、删除、修改等 3 个作用于时间输入输出自动机的建模成员上的变更算子外, 还考虑了针对交互序列变更的序变算子. 对于时间输入输出自动机 $\mathcal{T} = (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$, 定义各算子的含义:

- 增加表示在 $\mathcal{T}$ 的位置集合、时钟集合、动作集合、不变式和迁移集合等集合中添加新元素;
- 删除表示在 $\mathcal{T}$ 的位置集合、时钟集合、动作集合、不变式和迁移集合中去除已有的相关元素;
- 修改则指改变 $\mathcal{T}$ 的位置集合、初始状态、时钟集合、动作集合、不变式和迁移集合中原成员的相关属性, 例如重命名、迁移守护条件加强或减弱等;
- 序变是指基线的事件顺序发生了变化.

除上述算子外, 还可能存在拆分和合并等变更操作, 不过这些变更操作可以转换为增加和删除的叠加操作, 因此在本文中并没有将其识别为独立的变更算子. 由于序变可能不涉及到建模元素的变更, 不能通过单一的模型结构比较确定变更迹. 本文采用了将时间输入输出自动机转换成时间序列树的方式计算变更迹.

定义 12 时间输入输出自动机 $\mathcal{T} = (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$ 的时间序列树表示为 $\mathcal{T}_{\bar{\lambda}} = (N, \rho)$, 其中, $N = \{(l, \text{Inv}(l)) | l \in L\}$ 是节点集合, $\rho = \{((l, \text{Inv}(l)), \varphi, a, \gamma, (l', \text{Inv}(l')) | (l, \text{Inv}(l)), (l', \text{Inv}(l')) \in N \wedge l \xrightarrow{\varphi, a, \gamma} l' \in \rightarrow\}$ 是边的集合且 $\mathcal{T}_{\bar{\lambda}}$ 需满足下列约束条件:

- 1) $(l_0, \text{Inv}(l_0)) \in N$ 是 $\mathcal{T}_{\bar{\lambda}}$ 的唯一根节点;
- 2) $\forall (l, \text{Inv}(l)) \in N \setminus (l_0, \text{Inv}(l_0))$, 如果 $\exists (l_1, \text{Inv}(l_1)), (l_2, \text{Inv}(l_2)) \in N, (l_1, \text{Inv}(l_1)) \xrightarrow{\varphi, a, \gamma} (l, \text{Inv}(l))$ 和 $(l_2, \text{Inv}(l_2)) \xrightarrow{\varphi, a, \gamma} (l, \text{Inv}(l))$, 那么 $(l_1, \text{Inv}(l_1)) = (l_2, \text{Inv}(l_2))$, 即每个节点在 (φ, a, γ) 下存在唯一的前驱节点;
- 3) 若 $\forall n \in N, \text{Des}(n)$ 是 n 的所有子孙节点集合, 则 $\nexists n' \in \text{Des}(n), n' \xrightarrow{\varphi, a, \gamma} n$.

时间输入输出自动机 $\mathcal{T} = (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$ 转换成对应的时间序列树 $\mathcal{T}_{\bar{\lambda}} = (N, \rho)$ 的步骤如下.

- 1) $\mathcal{T}_{\bar{\lambda}}$ 的根节点 $n_0 = (l_0, \text{Inv}(l_0))$;
- 2) 假设 $\mathcal{T}_{\bar{\lambda}}$ 的第 $k-1$ 层已经构造完毕, 则第 k 层的构造方式为: 在第 $k-1$ 层任选一个节点 $n = (l, \text{Inv}(l))$, 若该节点已经在第 1 层至第 $k-1$ 层出现过, 则将该节点标记为叶节点; 否则, 若 $\exists l \xrightarrow{\varphi, a, \gamma} l' \in \rightarrow$, 在 ρ 中增加 $(l, \text{Inv}(l)) \xrightarrow{\varphi, a, \gamma} (l', \text{Inv}(l'))$, 在 N 中增加节点 $n' = (l', \text{Inv}(l'))$; 若 $\nexists l \xrightarrow{\varphi, a, \gamma} l' \in \rightarrow$, 则将 n 标记为叶节点; 对第 $k-1$ 层的所有节点重复此步骤.

若使用 $\text{Path}(\mathcal{T}_{\bar{\lambda}})$ 表示时间序列树从根节点到叶节点的所有路径, 则对于时间输入输出自动机 $\mathcal{T}$ 及其变更 $\mathcal{T}'$ 可计算 $\Delta P = \text{Path}(\mathcal{T}'_{\bar{\lambda}}) - \text{Path}(\mathcal{T}_{\bar{\lambda}})$ 是被识别的系统变更, 由这部分系统迁移序列产生的行为迹即为系统的变更迹 $\Delta tTr(\mathcal{T})$.

然而, 由于定义于实数集合上的时间标签的加入导致变更迹 $\Delta tTr(\mathcal{T})$ 是一个无穷集, 必须对其进行等价有限化约减后才可能作为测试生成的基础. 区域等价 (region equivalence)^[33] 是定义在时间自动机状态上的一种等价关系. 本文通过构造时间序列树 $\mathcal{T}_{\bar{\lambda}}$ 的区域序列树 (region sequence tree) 达到约减变更迹集合的目的.

定义 13 区域序列树. 若时间输入输出自动机 $\mathcal{T} = (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$ 的时间序列树为 $\mathcal{T}_{\bar{\lambda}} = (N, \rho)$, 则 $\mathcal{T}_{\bar{\lambda}}$ 的区域序列树表示为 $\mathcal{T}_{\bar{\lambda}r} = (N_r, \rho_r)$, 其中, $N_r = \{(l, [\bar{v}]) | l \in L \wedge \bar{v} \in V(C)\}$, $V(C)$ 表示时钟集合 C 上的所有解释, $[\bar{v}]$ 是 $\bar{v}$ 所在时钟区域的等价类; $\rho_r = \rho_{ra} \cup \rho_{rd}$, $\rho_{ra} = \{(l, [\bar{v}]) \xrightarrow{a} (l', [\bar{v}']) | (l, \text{Inv}(l)) \xrightarrow{\varphi, a, \gamma} (l', \text{Inv}(l')) \in \rho \wedge v \models \varphi \wedge [\bar{v}'] = [\gamma := 0]\bar{v}\}$, $\rho_{rd} = \{(l, [\bar{v}]) \xrightarrow{d} (l', [\bar{v}']) | d >$

$0, [\bar{v}] = [\bar{v} + d]\}$.

引理 3 对于时间输入输出自动机 $\mathcal{T}$ 而言, 若其变更后的系统模型为 $\mathcal{T}'$, 有

$$\text{tTr}(\mathcal{T}') = \Delta\text{tTr}(\mathcal{T}) \cup (\text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T})).$$

用自然语言解释该命题的含义为变更后系统的时间迹集合由变更迹 $\Delta\text{tTr}(\mathcal{T})$ 和未变更迹 $\text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T})$ 两部分构成. 这个结论将在下面证明安全性验证和测试之间的关系时用到.

证明 先证 $\text{tTr}(\mathcal{T}') \subseteq \Delta\text{tTr}(\mathcal{T}) \cup (\text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T}))$. $\forall \sigma \in \text{tTr}(\mathcal{T}')$, 若 $\sigma \in \text{tTr}(\mathcal{T})$, 则有 $\sigma \in \text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T})$; 若 $\sigma \notin \text{tTr}(\mathcal{T})$, 则有 $\sigma \in \text{tTr}(\mathcal{T}') \setminus \text{tTr}(\mathcal{T}) = \Delta\text{tTr}(\mathcal{T})$. 因此可得, $\sigma \in \Delta\text{tTr}(\mathcal{T}) \cup (\text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T}))$. 进而, $\text{tTr}(\mathcal{T}') \subseteq \Delta\text{tTr}(\mathcal{T}) \cup (\text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T}))$.

再证 $\Delta\text{tTr}(\mathcal{T}) \cup (\text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T})) \subseteq \text{tTr}(\mathcal{T}')$. $\forall \sigma \in \Delta\text{tTr}(\mathcal{T}) \cup (\text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T}))$, 有 $\sigma \in \Delta\text{tTr}(\mathcal{T})$ 或者 $\sigma \in \text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T})$. 很明显, $\sigma \in \text{tTr}(\mathcal{T}')$, 故有 $\Delta\text{tTr}(\mathcal{T}) \cup (\text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T})) \subseteq \text{tTr}(\mathcal{T}')$ 成立.

综合以上两点, 可得 $\text{tTr}(\mathcal{T}') = \Delta\text{tTr}(\mathcal{T}) \cup (\text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T}))$ 成立.

推论 1 若 $\mathcal{T}_I$ 是与系统规约 $\mathcal{T}_S$ 一致的系统实现, 其代码变更后的模型为 $\mathcal{T}'_I$, 则有

$$\mathcal{T}'_I \text{ tioco}^\lambda \mathcal{T}_S \Leftrightarrow \forall \varphi \in \Omega(\mathcal{T}_S), \mathcal{T}_S^{\delta^\lambda} \models \varphi \Rightarrow \Delta\text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset.$$

证明 $(\Rightarrow)$. 根据引理 1, 由 $\mathcal{T}'_I \text{ tioco}^\lambda \mathcal{T}_S$ 可得 $\text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 又有已知条件 $\mathcal{T}_S^{\delta^\lambda} \models \varphi$ 得 $\text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$, 因此可得, $\text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$, 又 $\Delta\text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}'_I^{\delta^\lambda})$, 由此可得 $\Delta\text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$ 成立.

$(\Leftarrow)$. 假设 $\mathcal{T}'_I \not\text{tioco}^\lambda \mathcal{T}_S$, 则存在时间迹 $\sigma \in \text{tTr}(\mathcal{T}'_I^{\delta^\lambda})$. 但 $\sigma \notin \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 分两种情况讨论:

1) 若 σ 是一条变更迹, 即 $\sigma \in \text{tTr}(\mathcal{T}'_I^{\delta^\lambda})$, 但 $\sigma \notin \text{tTr}(\mathcal{T}_I^{\delta^\lambda})$. 以 σ 为依据, 构造一个安全观察者 φ , 使得 σ 成为 φ 的一条 Violated 可识别时间迹. 根据已知 $\sigma \notin \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 可得 $\mathcal{T}_S^{\delta^\lambda} \models \varphi$, 因此有 $\Delta\text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$, 然而 $\sigma \in \Delta\text{tTr}(\mathcal{T}'_I^{\delta^\lambda})$, 因此, $\Delta\text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) \neq \emptyset$, 由此产生矛盾, 故假设不成立.

2) 若 σ 不是一条变更迹, 即 $\sigma \in \text{tTr}(\mathcal{T}'_I^{\delta^\lambda})$, 并且 $\sigma \in \text{tTr}(\mathcal{T}_I^{\delta^\lambda})$. 同样, 以 σ 为依据, 构造一个安全观察者 φ , 使得 σ 成为 φ 的一条 Violated 可识别时间迹. 根据已知 $\sigma \notin \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 可得 $\mathcal{T}_S^{\delta^\lambda} \models \varphi$, 根据定理 1 可得 $\mathcal{T}_I^{\delta^\lambda} \models \varphi$, 即 $\text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$. 但是 $\sigma \in \text{tTr}(\mathcal{T}'_I^{\delta^\lambda})$, 因此, $\text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) \neq \emptyset$, 由此产生矛盾, 故假设不成立.

综合以上两点, 原定理成立.

通俗地说, 推论 1 指出, 在系统允许的最大延迟 λ 下, 对于仅实施代码变更的系统实现而言, 若变更后实现与系统规约一致且系统规约满足期望属性, 那么变更后系统实现也满足期望属性. 同时, 在系统规约满足期望属性的前提下, 若要确保变更后代码满足期望的系统属性, 只要保证变更后系统实现与系统规约一致. 换句话说, 变更后代码相对于指定属性是否正确可以通过一致性测试判定.

引理 4 对于时间输入输出自动机 $\mathcal{T}$ 而言, 若其变更后的系统模型为 $\mathcal{T}'$, 有

$$\text{若 } \forall \varphi \in \Omega(\mathcal{T}), \mathcal{T} \models \varphi, \text{ 则 } \mathcal{T}' \models \varphi \Leftrightarrow \Delta\text{tTr}(\mathcal{T}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset.$$

证明 $(\Rightarrow)$. 根据 $\mathcal{T}' \models \varphi$, 由安全性验证的定义可得 $\text{tTr}(\mathcal{T}') \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$, 又 $\Delta\text{tTr}(\mathcal{T}) \subseteq \text{tTr}(\mathcal{T}')$, 因此可得, $\Delta\text{tTr}(\mathcal{T}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$.

$(\Leftarrow)$. 假设当 $\Delta\text{tTr}(\mathcal{T}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$ 时, $\mathcal{T}' \not\models \varphi$, 则存在一条时间迹 σ , $\sigma \in \text{tTr}(\mathcal{T}') \cap \text{tTr}(\varphi, \text{Violated})$, 分 σ 是变更迹和不是变更迹两种情况讨论:

1) 若 $\sigma \in \Delta \text{tTr}(\mathcal{T})$, 则与已知产生矛盾, 故假设不成立.

2) 若 $\sigma \in \text{tTr}(\mathcal{T}') \cap \text{tTr}(\mathcal{T})$, 则 $\sigma \in \text{tTr}(\mathcal{T})$, 又 $\sigma \in \text{tTr}(\varphi, \text{Violated})$, 可得 $\text{tTr}(\mathcal{T}) \cap \text{tTr}(\varphi, \text{Violated}) \neq \emptyset$, 进而有 $\mathcal{T} \models \varphi$ 成立, 产生矛盾, 故假设不成立.

综上所述, 原命题成立.

推论 2 若 $\mathcal{T}_S$ 和 $\mathcal{T}_I$ 分别为变更前系统规约和系统实现, 且 $\mathcal{T}_S$ 和 $\mathcal{T}_I$ 一致, $\mathcal{T}'_S$ 和 $\mathcal{T}'_I$ 是实施规格变更后的系统规约和系统实现, 则有

$$\mathcal{T}'_I \text{ tioco}^\lambda \mathcal{T}'_S \Leftrightarrow \forall \varphi \in \Omega(\mathcal{T}'_S), \Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset \Rightarrow \Delta \text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset.$$

证明 ($\Rightarrow$). 根据引理 4 可知 $\Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset \Rightarrow \mathcal{T}_S^{\delta^\lambda} \models \varphi$, 由此可得 $\text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$. 根据引理 2, 由 $\mathcal{T}'_I \text{ tioco}^\lambda \mathcal{T}'_S$, 可得 $\text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$. 因此有, $\text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$, 又 $\Delta \text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}'_I^{\delta^\lambda})$, 所以有结论 $\Delta \text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$ 成立.

($\Leftarrow$). 根据定理 1 可得 $\mathcal{T}'_I \text{ tioco}^\lambda \mathcal{T}'_S \Leftarrow \forall \varphi \in \Omega(\mathcal{T}'_S), \mathcal{T}_S^{\delta^\lambda} \models \varphi \Rightarrow \mathcal{T}'_I^{\delta^\lambda} \models \varphi$, 又根据引理 4 可知, $\mathcal{T}_S^{\delta^\lambda} \models \varphi \Leftrightarrow \Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$ 和 $\mathcal{T}'_I^{\delta^\lambda} \models \varphi \Leftrightarrow \Delta \text{tTr}(\mathcal{T}'_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) = \emptyset$ 成立.

由此原命题得证.

推论 2 指出, 在系统允许的最大延迟 λ 下, 对于规格变更而言, 如果需要验证变更后代码是否满足期望的安全属性, 需要两步完成. 第一步需要验证变更部分的系统规约是否满足期望属性, 第二步通过一致性测试, 确定变更后系统实现是否与变更后的系统规约保持一致. 如果两个步骤的结果都是肯定的, 那么变更后代码必然满足期望的属性. 其实, 从推论 2 中可以得出代码变更是规约变更的一个特例, 即系统规约变更迹为空集而代码变更迹不为空时的规约变更.

5 变更后安全性测试生成

一个高质量的测试是指在有限资源前提下可以发现尽可能多缺陷的测试. 测试所发现缺陷应集中在语义层面, 然而, 大多数基于模型的测试生成方法以满足被测对象的结构指标, 例如状态覆盖、迁移覆盖、迁移对覆盖, 分支覆盖, 条件覆盖等作为测试生成的结束条件. 虽然, 这些测试方法在提高测试的揭错能力和生产率方面都有积极的影响, 但是, 对于安全攸关的系统而言, 由于这类测试生成方法本身以结构性覆盖为测试设计的出发点, 使得它们很难给出系统实现是否满足期望的安全性属性的结论. 在本节中, 基于第 4 节建立的安全性验证和一致性测试之间的关联, 构建可用于变更后系统实现安全性验证的测试框架. 与其它测试方法相比, 该框架不仅可以发现常规的一致性缺陷, 同时也可以用于确定系统实现是否满足期望的安全性属性.

5.1 可测的输入输出自动机

可测性是指对于任何输入, 被测对象都能够提供唯一用于判定相关行为是否正确的标准. 由于测试是通过在给定输入下, 系统实际行为结果与期望结果之间的一致性比较来判定系统的正误, 因此, 从这个角度上讲, 我们认为可测性是测试生成所基于模型必须满足的属性之一.

定义 14 时间输入输出自动机 $\mathcal{T} = (L, l_0, C, \Sigma, \text{Inv}, \rightarrow)$ 的可测化模型表示为 $\hat{\mathcal{T}} = (L \cup \{\text{Fail}\}, l_0, C, \Sigma - \Sigma_\tau \cup \{\xi\}, \text{Inv}, \overset{\circ}{\rightarrow})$, 其中

- $\text{Fail} \notin L$ 是一个没有时间概念的决断位置, 表示系统产生了与期望行为违背的错误行为.
- $\xi \notin \Sigma$ 是新增的超时行为, 其在系统行为的时间约束被破坏时输出.

$$\bullet \rightarrow = \rightarrow \cup \{l \xrightarrow{\neg \text{Inv}(l), \xi^!} \text{Fail} \mid l \in L \wedge \text{Inv}(l) \neq \text{true}\} \cup \{l \xrightarrow{g, a} \text{Fail} \mid l \in L \wedge \text{Inv}(l) \neq \text{true} \wedge a \in \Sigma_l \wedge \frac{g}{a}\}.$$

从上面的定义可以看出, 一个时间输入输出自动机的可测化模型, 是通过完备化原模型中各种行为结果而得, 包括增加当位置不变式被破坏时, 输出超时行为的异常迁移 (迁移的目标位置为 Fail) 和在各种条件都满足的情况下, 系统没有输出期望行为的异常迁移. 基于可测化模型的定义, 我们可以得到如下定理.

定理 2 若 $\exists \sigma \in \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$ 并且 $\sigma \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda}, \text{Fail}) \Rightarrow \mathcal{T}_I \text{ tioco}^\lambda \mathcal{T}_S$.

通俗地说, 该定理指出, 如果系统实现的行为中存在可测化后系统规约定义中的异常行为, 那么, 系统实现中存在一致性缺陷. 因为, 可测化模型中的 Fail 可识别时间迹本身就是不期望发生的异常行为. 更进一步说, 时间迹 $\text{tTr}(\mathcal{T}_S^{\delta^\lambda}, \text{Fail})$ 是诱导系统实现可能发生故障的行为路径, 可以用于检测系统实现中是否存在一致性缺陷.

定理 3 若系统实现为 $\mathcal{T}_I$, 其代码变更后的模型为 $\mathcal{T}_I'$, 则有下列结论成立:

$$\exists \sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \text{ 并且 } \sigma \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda}, \text{Fail}) \Rightarrow \mathcal{T}_I' \text{ tioco}^\lambda \mathcal{T}_S.$$

定理 3 指出, 在系统允许的最大延迟 λ 下, 如果系统实现的变更迹中存在系统规约中不期望的异常行为, 那么代码变更后的系统实现存在一致性缺陷. 对于规约变更而言, 有类似的结论, 即如果系统实现的变更迹中包含系统规约变更迹所不期望的行为, 则变更后系统实现与变更后系统规约不一致.

定理 4 若 $\mathcal{T}_S$ 和 $\mathcal{T}_I$ 分别为变更前系统规约和系统实现, $\mathcal{T}_S'$ 和 $\mathcal{T}_I'$ 是实施规格变更后的系统规约和系统实现, 则有:

$$\exists \sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \text{ 并且 } \sigma \in \Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda}, \text{Fail}) \Rightarrow \mathcal{T}_I' \text{ tioco}^\lambda \mathcal{T}_S'.$$

5.2 安全性测试生成

定理 2, 3 和 4 虽然指明了如何诱导系统实现产生不一致性缺陷, 但是并没有说明如何验证系统实现是否满足期望的安全性属性. 这是因为在可测化模型中缺少了对安全性属性进行验证的信息. 为了达到这个目标, 最直接的做法就是在可测化模型中加入安全性属性的描述, 即在两者之间进行同步组合运算. 给定系统规约 $\mathcal{T}_S$ 和系统期望满足的安全性属性 φ , $\mathcal{T}_S^{\delta^\lambda} \parallel \varphi$ 是可用于安全性测试生成的可测化系统规约, 同时, 该模型也可用于检测一致性缺陷.

定理 5 对于代码变更而言, 若 $\exists \sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$ 并且 $\sigma \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda} \parallel \varphi, \text{Violated}) \Rightarrow \mathcal{T}_I^{\delta^\lambda} \not\models \varphi$.

证明 根据已知条件, $\sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 又 $\Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_I^{\delta^\lambda})$, 可得 $\sigma \in \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$. 又 $\sigma \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda} \parallel \varphi, \text{Violated})$, 因此可得, $\sigma \in \text{tTr}(\mathcal{T}_I^{\delta^\lambda} \parallel \varphi, \text{Violated})$, 即 $\text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) \neq \emptyset$. 因此, $\mathcal{T}_I^{\delta^\lambda} \not\models \varphi$ 成立.

定理 6 对于规约变更而言, 若 $\exists \sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$ 并且 $\sigma \in \Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda}, \text{Violated}) \Rightarrow \mathcal{T}_I^{\delta^\lambda} \not\models \varphi$.

证明 根据已知条件, $\sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 且 $\Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$, 因此有 $\sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$. 同时注意到 $\sigma \in \Delta \text{tTr}(\mathcal{T}_S^{\delta^\lambda}, \text{Violated}) \subseteq \text{tTr}(\mathcal{T}_S^{\delta^\lambda} \parallel \varphi, \text{Violated})$, 由此可得, $\sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) \subseteq \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\varphi, \text{Violated}) \neq \emptyset$. 由此原命题得证.

基于上述几个定理和同步组合算子的定义, 我们可以很自然地得到下面两个推论.

推论 3 对于代码变更而言, 若 $\exists \sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta^\lambda}) \cap \text{tTr}(\mathcal{T}_S^{\delta^\lambda})$ 并且 $\sigma \in \text{tTr}(\mathcal{T}_S^{\delta^\lambda}, (\text{Failed}, \text{Violated})) \Rightarrow \mathcal{T}_I' \text{ tioco}^\lambda \mathcal{T}_S$ 并且 $\mathcal{T}_I^{\delta^\lambda} \not\models \varphi$.

推论 4 对于规约变更而言, 若 $\exists \sigma \in \Delta \text{tTr}(\mathcal{T}_I^{\delta\lambda}) \cap \Delta \text{tTr}(\mathcal{T}_S^{\delta\lambda})$ 并且 $\sigma \in \Delta \text{tTr}(\mathcal{T}_S^{\delta\lambda}, (\text{Failed}, \text{Violated})) \Rightarrow \mathcal{T}_I^{\delta\lambda} \not\sim \mathcal{T}_S^{\delta\lambda}$ 并且 $\mathcal{T}_I^{\delta\lambda} \not\models \varphi$.

上述各个定理或者推论的含义可以通俗地解释为, 在系统允许的最大延迟 λ 下, 如果系统实现的变更迹中存在系统规约中定义的异常行为或者破坏安全性属性的行为, 那么, 以系统允许的最大延迟 λ 为限的实时系统实现存在不一致性缺陷或者不满足期望的安全性属性. 由此, 下列各种时间迹是可以诱导系统实现出现不一致或者违反安全性属性的行为序列.

1) $\text{tTr}(\mathcal{T}_S^{\delta\lambda} \parallel \varphi, \text{Fail})$ 可用于检测代码变更的一致性缺陷; $\Delta \text{tTr}(\mathcal{T}_S^{\delta\lambda} \parallel \varphi, \text{Fail})$ 可以用于检测规约变更的一致性缺陷.

2) $\text{tTr}(\mathcal{T}_S^{\delta\lambda} \parallel \varphi, \text{Violated})$ 可用于检测实施代码变更后, 系统实现是否满足安全性属性; $\Delta \text{tTr}(\mathcal{T}_S^{\delta\lambda} \parallel \varphi, \text{Violated})$ 可以用于检测实施规约变更后, 系统实现是否满足期望的安全性属性.

3) $\text{tTr}(\mathcal{T}_S^{\delta\lambda} \parallel \varphi, (\text{Failed}, \text{Violated}))$ 对于代码变更而言, 即可以检测系统实现的一致性缺陷又能够验证安全性属性; $\Delta \text{tTr}(\mathcal{T}_S^{\delta\lambda} \parallel \varphi, (\text{Failed}, \text{Violated}))$ 对于规约变更而言, 即可以检测系统实现的一致性缺陷又能够验证安全性属性.

从测试用例有效性的角度而言, 上述行为序列是一致性测试和安全性验证的最佳代表. 最后, 由于测试的输入输出与系统行为具有反向关系, 因此, 在经过各种计算得到上述行为迹之后, 还需要进行镜面运算, 即输入转换为输出, 输出转换为输入, 才能使之变成真正的测试步骤序列.

6 案例研究

列车自动控制系统 (automatic train control, ATC) 是一种控制列车安全运行的实时系统. 列车自动防护功能 (automatic train protection, ATP) 是 ATC 系统必须提供的安全攸关功能之一. ATP 功能是在地面 ATP 设备和车载 ATP 设备 (vehicle onboard controller, VOBC) 的相互配合下完成的. 其中, VOBC 通过实施超速防护直接负责避免列车出现超速、追尾等各类灾难性事故. 在本节中, 以 VOBC 的列车自动防护功能为案例研究, 说明我们提出的基于形式化测试的变更后安全性验证方法.

6.1 车载设备 ATP 功能说明和行为建模

车载 ATP 的超速防护功能在轨旁信号保护模式下 (可以理解为列车运行时的主要驾驶模式, 还有其它诸如限制人工向前/后、关闭、洗车等模式, 这些模式在本文中不作考虑) 的规格说明可如下描述: 在每个应用程序周期, VOBC 实时计算列车的当前速度并将其与列车在当前模式下的允许速度进行比较. 在轨旁信号保护模式下, 列车需要根据一些条件实时计算允许速度. 当列车速度接近允许速度时, VOBC 必须发出超速警报; 如果在 50 个时间单位后, 列车速度仍然大于允许速度的话, 那么 VOBC 需要在 50 个时间单位内启动紧急制动. 在速度超过允许速度时但在减速的过程中 VOBC 必须保持紧急制动状态直到速度减到低于允许速度为止. 根据上述说明, 利用时间输入输出自动我们可以构造出 VOBC 的超速防护行为模型, 如图 1.

为了保证列车行驶的安全, 期望列车满足的安全性属性之一是, 系统在被要求启动紧急刹车时, 必须在 150 个时间单位内成功启动刹车. 该属性从描述结构上说应该属于时间受限活性. 时间受限活性 (time-bounded liveness) 是安全攸关系统在实时性维度上所特有的一类系统属性, 用于表达在给定输

图 1 VOBC 的时间输入输出自动机

Figure 1 TIOA for VOBC

图 2 VOBC 的安全性观察者

Figure 2 Safety observer for VOBC

出的时限产生期望的系统行为,但其本质是一种安全性属性,因此可以使用安全性观察者进行描述.图2给出了该属性的时间输入输出自动机模型.其中, l_0 是初始位置,安全性观察者在 l_0 处只负责监听系统是否产生刹车请求,其余都不考虑,因此有一个标号为 $*/EB!$ 的自迁移.在检测到系统产生刹车请求时,安全性观察者开始计时,并迁移到位置 l_1 .根据期望的安全属性的描述,在位置 l_1 处停留的时间不能超过150个时间单位,因此, l_1 上有位置不变式 $x^o \leq 150$.同时,一旦延迟超过150个时间单位,系统即破坏了期望的安全性属性,不论是否成功启动刹车与否.

为了进一步提高超速防护的可靠性,并且减少紧急刹车给乘客带来的不良乘车体验,提高乘车舒适度,决定在列车接近允许速度与超速之间再增加一级迫近报警,用以及时调整列车速度.修改后的需求如下描述:在每个应用程序周期,VOBC实时计算列车的当前速度并将其与列车允许速度进行比较.在轨旁信号保护模式下,列车需要根据一些条件实时计算允许速度.当列车速度接近允许速度时,VOBC必须发出超速警报;如果在50个时间单位后,列车速度仍然大于允许速度的话,VOBC必须把超速警报设为迫近警报并且需要在100个时间单位启动刹车以使速度恢复正常.若当前速度大于允许速度,那么VOBC需要在50个时间单位内启动紧急制动.在速度超过允许速度时但在减速的过程中VOBC必须保持紧急制动状态直到速度减到低于允许速度为止.图3是变更的系统规约,增加了新位置Alarming,以及Approaching和Alarming,Alarming和Braking,Approaching和Braking之间的迁移.

6.2 车载ATP功能的变更后安全性测试生成

本文提出的测试方法的一个优势是,不需要依赖于实际代码进行测试用例设计,所有工作都是通过对系统规约的各种计算完成的.这不仅有利于精确确定受变更影响的被测范围,而且支持没有源代

图 3 实施变更后的车载 ATP 时间输入输出自动机

Figure 3 TIOA for changed VOBC

图 4 变更后 VOBC 的 λ 受限时间输入输出自动机Figure 4 λ -bounded TIOA for changed VOBC

图 5 变更后 VOBC 的可测模型

Figure 5 Testable model for changed VOBC

码或者代码不完整的系统验证. 同时, 根据我们的理论证明, 得到的测试结果不仅可以发现被测系统中的一致性缺陷, 而且能够用于检测系统实现是否满足期望的安全性属性. 为了验证变更后的车载 ATP 功能实现是否满足期望的安全性属性, 需要包含下列计算步骤.

1) 计算车载 ATP 功能规约的 λ 受限的时间输入输出自动机. 由于一般情况下车载 ATP 的一个工作周期为 300 个或 500 个时间单位, 因此, 在这里我们取 λ 为 300. 根据 λ 受限的时间输入输出自动机的定义 (定义 9), 可求得变更后车载 ATP 的 λ 受限的时间输入输出自动机, 如图 4 所示.

2) 在车载 ATP 功能规约的 λ 受限的时间输入输出自动机的基础上, 计算其可测化模型 (定义 14). 这个模型在 λ 受限的时间输入输出自动机模型的基础上, 增加了不期望系统输出的行为, 图 5 展示了相关结果.

图 6 VOB 的测试模型

Figure 6 Test model for VOB

3) 计算车载 ATP 可测模型与系统期望属性之间的同步组合 (定义 4), 使其包含可用于安全性属性验证的信息, 我们称其为 ATP 的测试模型, 如图 6 所示。

4) 将 ATP 的测试模型转换为时间序列树 (定义 12), 并根据基线行为模型和变更后行为模型计算 ATP 模型的变更路径。ATP 的时间序列树如图 7 所示, 图中圈出的部分即为变更的行为序列。根据定理 4, 6 和推论 4, 在这部分序列中, 包含 Fail 决断位置的行为序列是可用于检测不一致性的行为序列, 包含 Violate 决断位置的行为序列是可用于安全性检测的行为序列。这些序列再进行将输入动作映射为输出动作, 输出动作映射为输入动作之后, 即成为真正的测试序列。

7 结论

发布后变更是实时系统在其生命周期中必然经历的研发活动。在人类社会对安全实时系统需求量增加, 需求多样化以及技术演变等因素的共同作用下, 安全攸关系统发布后变更的频率和复杂度已经大幅提高。由于直接关系到生命安全, 对安全攸关系统实施变更后必须保证系统实现仍然安全。本文从方法的可用性和可靠性的角度出发, 提出了一种基于形式化的可用于实时系统变更后安全性回归验证的测试框架。与常用的软件模型检测和程序验证技术相比, 该框架将对系统实现安全性验证归结为自动化的一致性测试, 因而容易被研发工程师学习和接受。与其它测试方法相比, 该框架建立在严密的数学证明的基础之上, 可以保证方法本身的正确性。同时, 这种方法还可以直接判定变更后系统行为是否满足期望的安全性属性, 弥补了大多数测试生成方法以发现被测代码缺陷为主要目标, 不能直接判定其是否满足安全性属性的不足。该框架是建立在一致性测试本身就是一个安全性属性的认知之上。一致性测试的含义其实可以被描述为系统实现不会违背相关系统规约, 因此, 本质上一致性测试就是一个安全性属性。这为建立基于测试验证系统是否满足期望的安全性属性提供了理论上的可能性。但是, 也需注意到, 该测试方法对行为设计模型依赖较大, 所以必须确保设计模型的实效性。

本文的方法以时间输入输出自动机为建模语言, 通过使用时间输入输出自动机描述期望的安全性

图 7 VOBC 的时间序列树

Figure 7 Timed sequence tree for VOBC

属性, 并计算其与可测化的系统规约模型之间的同步组合作为变更后系统的一致性测试和安全性验证的基础. 与以结构覆盖为测试生成目标的其它方法相比, 该测试方法不是覆盖指定的建模元素, 而是意图引导被测系统实现进入错误状态, 包括违反规约和期望的安全属性. 不过, 由于时间输入输出自动机只支持时钟变量, 因此, 无法方便地处理其他类型或结构更为复杂的变量信息, 导致在建模系统行为粒度方面受到了一定的限制. 同时, 又由于时间输入输出自动机不存在层次信息, 限制了其在建模复杂系统的行为方面的应用. UML Statechart 是被广为使用的复杂系统行为建模工具. MARTE (modeling and analysis for real-time embedded system) 是 UML 在强实时嵌入式系统建模领域的一个扩展. 在后续的研究工作中, 我们将会借助 UML Statechart 和 MARTE 丰富的行为和时间建模能力进一步增加该方法在面对复杂系统行为建模方面的能力和应用. 另外, 方法中所定义的各种算子的实现还处于初始阶段, 还需要进行高效地算法设计以及相关工具的开发, 并需要开展广泛的案例研究.

参考文献

- 1 Heimdahl M P E. Safety and software intensive systems: challenges old and new. In: Proceedings of Future of Software Engineering 2007. Piscataway: IEEE, 2007. 137–152
- 2 Engler D, Musuvathi M. Static analysis versus software model checking for bug finding. In: Proceedings of LNCS2937, Verification, Model Checking, and Abstract Interpretation. Berlin: Springer, 2004. 191–210
- 3 Beatty D L, Bryant R E. Formally verifying a microprocessor using a simulation methodology. In: Proceedings of 31st

- Conference on Design Automation. Piscataway: IEEE, 1994. 596–602
- 4 Lutz R R. Software engineering for safety: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. New York: ACM, 2000. 213–226
 - 5 Gardiner S N. Testing Safety-Related Software: A Practical Handbook. London: Springer, 1999
 - 6 McDermid J, Kelly T. Software in safety critical systems-achievement and prediction. Nuclear Future, 2006, 2: 140–146
 - 7 Ellims M. On wheels, nuts and software. In: Proceedings of the 9th Australian Workshop on Safety Critical Systems and Software. New York: ACM, 2004. 67–76
 - 8 Hierons R M, Bogdanov K, Bowen J P, et al. Using formal specifications to support testing. ACM Comput Surv, 2009, 41: 9
 - 9 Tretmans J. Test generation with inputs, outputs and repetitive quiescence. Softw Concepts Tools, 1996, 17: 103–120
 - 10 Schmaltz J, Tretmans J. On conformance testing for timed systems. In: Cassez F, Jard C, eds. Formal Modeling and Analysis of Timed Systems. Berlin: Springer, 2008. 250–264
 - 11 Briones L B, Brinksma E. A test generation framework for quiescent real-time systems. In: Grabowski J, Nielsen B, eds. Formal Approaches to Software Testing. Berlin: Springer, 2005. 64–78
 - 12 Hessel A, Larsen K G, Mikucionis M, et al. Testing real-time systems using UPPAAL. In: Hierons R M, Bowen J P, Harman M, eds. Formal Methods and Testing. Berlin: Springer, 2008. 77–117
 - 13 Krichen M, Tripakis S. Conformance testing for real-time systems. Formal Methods Syst Des, 2009, 34: 238–304
 - 14 Bohnenkamp H, Belinfante A. Timed testing with TorX. In: Fitzgerald J, Hayes I J, Tarlecki A, eds. FM 2005: Formal Methods. Berlin: Springer, 2005. 173–188
 - 15 Gargantini A, Heitmeyer C. Using model checking to generate tests from requirements specifications. In: Nierstrasz O, Lemoine M, eds. Software Engineering ESEC/FSE'99. Berlin: Springer, 1999. 146–162
 - 16 Ammann P, Ding W, Xu D. Using a model checker to test safety properties. In: Proceedings of 7th IEEE International Conference on Engineering of Complex Computer Systems. Piscataway: IEEE, 2001. 212–221
 - 17 Rusu V, Marchand H, Tschaen V, et al. From safety verification to safety testing. In: Processings of 16th IFIP International Conference of Testing of Communicating System. Berlin: Springer, 2004. 160–176
 - 18 Rusu V. Combining formal verification and conformance testing for validating reactive systems. Softw Test Verif Rel, 2003, 13: 157–180
 - 19 Constant C, Jeron T, Marchand H, et al. Integrating formal verification and conformance testing for reactive systems. IEEE Trans Softw Eng, 2007, 33: 558–574
 - 20 Yoo S, Harman M. Regression testing minimization, selection and prioritization: a survey. Softw Test Verif Rel, 2012, 22: 67–120
 - 21 Engstrom E, Runeson P, Skoglund M. A systematic review on regression test selection techniques. Inf Softw Technol, 2010, 52: 14–30
 - 22 Zhang Z Y, Chen Z Y, Xu B W, et al. Research progress on test case evolution. J Softw, 2013, 24: 663–674 [张智轶, 陈振宇, 徐宝文, 等. 测试用例演化研究进展. 软件学报, 2013, 24: 663–674]
 - 23 Xu Z H. Directed test suite augmentation. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE2011). New York: ACM, 2011. 1110–1113
 - 24 Santelices R, Chittimalli P K, Apiwattanapong T, et al. Test-suite augmentation for evolving software. In: Proceedings of 23rd IEEE/ACM International Conference on Automated Software Engineering. Piscataway: IEEE, 2008. 218–227
 - 25 Santelices R, Harrold M J. Applying aggressive propagation-based strategies for testing changes. In: Proceedings of 2011 IEEE 4th International Conference on Software Testing, Verification and Validation. Piscataway: IEEE, 2011. 11–20
 - 26 Taneja K, Xie T, Tillmann N, et al. Guided path exploration for regression test generation. In: Proceedings of 31st International Conference on Software Engineering. Piscataway: IEEE, 2009. 311–314
 - 27 Qi D, Roychoudhury A, Liang Z. Test generation to expose changes in evolving programs. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2010. 397–406
 - 28 Xu Z, Kim Y, Kim M, et al. A hybrid directed test suite augmentation technique. In: Proceedings of 2011 IEEE 22nd International Symposium on Software Reliability Engineering. Piscataway: IEEE, 2011. 150–159
 - 29 Jiang B, Tse T H, Grieskamp W, et al. Assuring the model evolution of protocol software specifications by regression testing process improvement. Softw Pract Exper, 2011, 41: 1073–1103

- 30 Korel B, Tahat L H, Vaysburg B. Model based regression test reduction using dependence analysis. In: Proceedings of International Conference on Software Maintenance. Piscataway: IEEE, 2002. 214–223
- 31 Beydeda S, Gruhn V. Integrating white- and black-box techniques for class-level regression testing. In: Proceedings of 25th Annual International Computer Software and Applications Conference. Piscataway: IEEE, 2001. 357–362
- 32 Farooq Q, Iqbal M, Malik Z I, et al. A model-based regression testing approach for evolving software systems with flexible tool support. In: Proceedings of 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems. Piscataway: IEEE, 2010. 41–49
- 33 Sun H Y, Liu J, Du D H. Integration of safety verification with conformance testing in real-time reactive system. In: Proceedings of 19th Asia-Pacific Software Engineering Conference. Piscataway: IEEE, 2012. 10–19
- 34 Alur R, Dill D L. A theory of timed automata. Theor Comput Sci, 1994, 126: 183–235
- 35 En-Nouaary A, Dssouli R, Khendek F. Timed Wp-method: testing real-time systems. IEEE Trans Softw Eng, 2002, 28: 1023–1038
- 36 Briones B L. Theories for Model-based Testing: Real-time and Coverage. Dissertation for Ph.D. Degree. Enschede: University of Twente, 2007

A safety verification approach for changed real time implementation based on formal testing

SUN HaiYing¹, LIU Jing^{1*}, CHEN XiaoHong¹, DU DeHui¹ & ZHOU TingLiang²

1 *Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 200062, China;*
 2 *Key Laboratory of Road and Traffic Engineering of Ministry of Education, Tongji University, Shanghai 200092, China*

*E-mail: jliu@sei.ecnu.edu.cn

Abstract It is necessary and competitive to verify the desired safety properties on the changed implementation during the maintenance phase of a safety critical system. Software model checking and program verification are the popular applied automatic safety verification techniques on the program level. In the paper, we propose a formal testing method for the system behaviors which tries to verify the desired safety properties for the changed implementation by means of conformance testing. Firstly, we prove the correctness of our method based on timed input/output automata and its semantics model. Secondly, we present a regression testing generation framework of safety verification for the changed implementation. Compared with other real time testing methods, our method can not only detect the common non-conformance defects but also conclude the satisfaction with the desired safety properties. The method is illustrated with a changing requirement of the automatic train protection function in the train control system.

Keywords software engineering, software evolution, real time systems, formal methods, safety verification, software testing, safety testing

model-driven engineering, etc.

SUN HaiYing was born in 1976. She received the master's degree in computer science and technology from Nanjing University of Science and Technology in 2001. She is now a lecturer and also a Ph.D. student at East China Normal University. Her research interest includes formal method, automatic test generation, system simulation, model-

and verification in hybrid system; model driven architecture, unifying theories of programming, MARTE/UML and time constraints, formal theory in object-oriented programming, etc. She was a fellow of UNU/IIST during 2003–2004.

LIU Jing received her B.S. and M.S. degrees in Computer Science from Hefei University of Technology in 1985 and 1993, respectively. She earned her Ph.D. degree in formal methods in 2005. She is a professor of computer science and vice director of Theoretical Computing Institute, East China Normal University. Her research interest includes formal design, specification

CHEN XiaoHong was born in 1982. She received the Ph.D. degree in computer software and theory from Academy of Mathematics and Systems Science, Chinese Academy of Sciences in 2010. Currently, she is a lecturer at East China Normal University. Her research interest includes formal requirements engineering, software engineering, embedded systems, etc.

DU DeHui was born in 1979. She received the Ph.D. degree in computer science from Wuhan University in 2007. Currently, she is an associate professor at SEI of East China Normal University. Her research interest includes model checking, modeling and verifying real-time stochastic hybrid system, formal software engineering, etc.