软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(10):4729−4750 [doi: 10.13328/j.cnki.jos.006997]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

多时间无干扰性验证方法*

刘乔森, 孙 聪, 魏晓敏, 曾荟铭, 马建峰

(西安电子科技大学 网络与信息安全学院, 陕西 西安 710071)

通信作者: 孙聪, E-mail: suncong@xidian.edu.cn

摘 要: 安全关键嵌入式软件的运行时行为通常具有严格时间约束, 对安全属性的执行提出额外要求. 针对嵌入式软件的信息流安全保护要求, 以及现有安全性验证方法面向单一属性且存在假阳性等问题, 首先从现实场景的安全需求出发, 提出一种新的时间无干扰属性 timed SIR-NNI; 然后提出一种兼容多种时间无干扰属性 (timed BNNI, timed BSNNI 及 timed SIR-NNI) 统一验证的信息流安全验证方法, 该验证方法依据不同的时间无干扰性要求, 从待验证时间自动机自动构造测试自动机和精化自动机, 通过 UPPAAL 的可达性分析实现精化关系检查和安全性验证. 实现的验证工具 TINIVER 从 SysML 顺序图模型或 C++源码提取时间自动机实施验证流程. 使用 TINIVER 对现有时间自动机模型和安全属性的验证说明方法的可用性, 对无人机飞行控制系统 ArduPilot 和 PX4 的典型飞行模式切换模型的安全验证说明方法的实用性和可扩展性. 此外, 方法能避免现有典型验证方法的假阳性缺陷.

关键词: 无干扰性; 时间自动机; 精化关系; 可达性分析; 信息流安全

中图法分类号: TP311

中文引用格式: 刘乔森, 孙聪, 魏晓敏, 曾荟铭, 马建峰. 多时间无干扰性验证方法. 软件学报, 2024, 35(10): 4729–4750. http://www.jos.org.cn/1000-9825/6997.htm

英文引用格式: Liu QS, Sun C, Wei XM, Zeng HM, Ma JF. Multi-timed Noninterference Verification. Ruan Jian Xue Bao/Journal of Software, 2024, 35(10): 4729–4750 (in Chinese). http://www.jos.org.cn/1000-9825/6997.htm

Multi-timed Noninterference Verification

LIU Qiao-Sen, SUN Cong, WEI Xiao-Min, ZENG Hui-Ming, MA Jian-Feng

(School of Cyber Engineering, Xidian University, Xi'an 710071, China)

Abstract: Safety-critical embedded software usually has rigorous time constraints over the runtime behaviors, raising additional requirements for enforcing security properties. To protect the information flow security of embedded software and mitigate the limitations of the existing simplex verification approaches and their potential false positives, this study first proposes a new timed noninterference property, i.e., timed SIR-NNI, based on the security requirement of a realistic scenario. Then the study presents an information flow security verification approach that unifies the verification of multiple timed noninterference properties, i.e., timed BNNI, timed BSNNI, and timed SIR-NNI. Based on the different timed noninterference requirements, the approach constructs the refined automata and test automata from the timed automata under verification. The study uses UPPAAL's reachability analysis to implement the refinement relation check and the security verification. The verification tool, i.e., TINIVER, extracts timed automata from SysML's sequential diagrams or C++ source code to conduct the verification procedure. The verification results of TINIVER on existing timed automata models and security properties justify the usability of the proposed approach. The security verifications on the typical flight-mode switch models of the UAV flight control systems ArduPilot and PX4 demonstrate the practicability and scalability of the proposed approach. Besides, the approach is effective in mitigating the false positives of a state-of-the-art verification approach.

Key words: noninterference; timed automata; refinement; reachability analysis; information flow security

* 基金项目: 国家自然科学基金 (62272366, 61872279); 陕西省重点研发计划 (2023-YBGY-371); 陕西省自然科学基础研究计划 (2021JQ-207)
收稿时间: 2022-09-29; 修改时间: 2023-03-28, 2023-06-05; 采用时间: 2023-07-14; jos 在线出版时间: 2023-11-15
CNKI 网络首发时间: 2023-11-16

安全关键的嵌入式软件常具有实时性要求. 以无人机飞行控制系统为例, 飞控系统通常可设置多种调度器, 以不同周期通过回调方式轮询执行各控制任务, 对控制任务施以明确的时间约束. 另一方面, 嵌入式软件从工作环境接收不同完整性等级的数据进行处理并作用于反馈控制过程, 数据的完整性变化和流向是衡量嵌入式软件信息流安全的关键因素. 信息流安全作为系统信息安全的核心特性之一, 对嵌入式软件的功能安全具有直接影响. 因此, 在实时性约束下实现对嵌入式软件信息流安全属性的验证能力对嵌入式软件功能安全至关重要.

经典无干扰性理论[1]给出了无干扰属性 (noninterference) 的一般定义, 定义了确定性有限自动机的信息流安全性, 要求低安全级的组件不应推断出任何有关高安全级组件活动的信息, 从而使公开可观察的动作不依赖于敏感信息. 无干扰属性的验证在抽象模型[2,3]和程序[4]上已有长期发展, 但早期工作虽有关注模型和程序的非确定性, 但对时间约束缺乏考虑. 近年来, 一些研究愈加关注与时间约束相关的信息流安全属性验证问题, 例如在同步系统模型上验证系统不存在时间隐蔽信道[5], 但其系统模型并未对时间进行量化. 张帆等人[6]提出了使用无干扰模型进行软件可信性度量的方法, 基于系统调用的预期行为与真实行为的差异对程序的可信性进行在线度量, 然而其所述实时性并未考虑具体迁移动作所需的时间约束, 且本文针对的反馈控制系统场景受环境不确定性影响, 难以建立该方法所需的预期行为模型, 因而该方法不适用于本文场景. 通过引入特定的轨迹量词或显式同步量化关系提出的多种超时态逻辑, 如超线性时态逻辑 (HyperLTL)[7]、超度量时态逻辑 (HyperMTL)[8]、超度量间隔时态逻辑 (HyperMITL)[9]等, 能够用于定义具有同步和异步语义的时间超属性 (timed hyperproperty), 时间超属性可看作时间无干扰属性的超集, 但其验证尚缺乏工具支持.

由于确定性模型的局限性, 经典无干扰属性不能作用于非确定性模型, 因而不适用于行为多源的控制系统软件. 为适应并发软件特点, Focardi 等人[10]扩展了经典无干扰定义, 提出了非确定无干扰性 (non-deterministic noninterference, NNI) 和强非确定无干扰性 (strong non-deterministic noninterference, SNNI), 并通过使用互模拟等价条件替换了 NNI 和 SNNI 定义中的 trace 等价条件, 推广出了互模拟非确定无干扰性 (bisimulation non-deterministic noninterference, BNNI) 和互模拟强非确定无干扰性 (bisimulation strong non-deterministic noninterference, BSNNI). Lee 等人在接口自动机上提出严格输入精化非确定无干扰性 (strict input refinement-based non-deterministic noninterference, SIR-NNI)[11], 从待验证自动机构造出无干扰性定义涉及的两个自动机, 并对它们执行 (半) 同步积操作并在 (半) 同步积上引入向失败状态的迁移, 如果同步执行过程无法避免到达失败状态, 则自动机不满足无干扰性. 李沁等人[12]指出并发程序时间信道可能使得敏感信息通过线程状态变量传递从而引起的非法信息流, 提出了一种针对线程信息流的程序逻辑. 基于变量依赖关系表达无干扰性, 利用依赖逻辑进行属性验证, 但未给出自动化验证手段. Sun 等人[13]提出了一种抽象不可区分关系, 用基于不可区分关系的无干扰性抽象 (R-非确定无干扰性) 统一了 BSNNI 和 SIR-NNI 的概念及工具化验证方法[14]. 为泛化安全格表达能力并提高组合验证效率, 后续工作[15]将安全接口结构扩展到适用于任意有限安全格, 提出了两种适用于复杂安全格的无干扰属性及工具化验证方法[16]. 此外, 谢钧等人[17]提出了适用于非确定系统的无干扰模型, 并验证了一个动态访问控制模型满足无干扰模型规定的安全性. 缓存测信道是另一种可以使用无干扰性推理进行验证的系统脆弱性, Jiang 等人[18]提出的基于熵的无干扰性推理架构能够支持在非确定性事件模型上验证解除条件是否违反, 从而推断是否存在信息泄露, 方法应用于对 8 种现有缓存设计的边信道分析.

以上针对非确定无干扰性的验证均未考虑嵌入式软件可能面临的实时性约束问题. 在此类软件中, 要求攻击者观察到的系统动作在时钟约束下不依赖于敏感信息和高安全级活动. 而现有时间模型上的非确定无干扰性存在各种局限. 早期用时钟周期刻画的信息流行为仅限于描述动作的序关系[19]. 在后续研究中提出的时间无干扰属性要求可公开观察动作及其执行时间不会受到高安全级动作执行与否的影响[20]. 时间互模拟强非确定无干扰性 (timed bisimulation strong non-deterministic noninterference, timed BSNNI)[21−23]、广义时间组合不可推断性 (timed generalized non deducibility on compositions, tGNDC) 和时间组合不可推断性 (timed non deducibility on compositions, tNDC)[24]由于对输入动作的时间约束要求不够严格, 无法用于检测利用输入动作反应来进行推断的攻击. Valilikos 等人[25]提出比时间无干扰性更宽松的信息泄露策略, 容许系统在特定条件下对外发布机密信息, 但由于未考虑输入输出动作对系统的影响, 因此也无法用于检测利用输入动作反应来进行推断的攻击. 此外, 以上工作均

未考虑时间无干扰性的验证问题. 时间互模拟非确定无干扰性 (timed bisimulation non-deterministic noninterference, timed BNNI)[26]和广义时间无干扰性 (timed generalized noninterference, timed GNI)[20]是两种有验证方法支撑的时间无干扰属性, 它们都沿用了精化检查[27]方法思路, 利用由待验证自动机生成的测试自动机和精化自动机进行并行组合, 对组合结果进行可达性分析, 验证模型是否满足这些时间无干扰性. 然而, timed GNI 属性基于时间模拟这一拟序关系, 该关系仅能保证对系统输入动作的单向限制, 亦无法用于检测利用系统对输入动作反应的差异进行推断的攻击; 而文献 [26] 方法在验证 timed BNNI 时存在假阳性问题.

针对以上时间无干扰属性无法刻画系统对输入动作反应差异的缺点以及 timed BNNI 的验证缺陷, 本文提出了 timed SIR-NNI 属性, 该属性利用 timed SIR 精化关系避免了现有时间无干扰属性对输入动作的时间约束要求不严格的问题; 进一步提出了一种新的时间无干扰性自动验证方法及工具 (timed noninterference verifier, TINIVER). TINIVER 从 SysML 模型顺序图或飞控系统 C++源码中提取时间约束标记的模型信息, 自动生成待验证时间自动机和相应的精化自动机及测试自动机, 调用 UPPAAL[28]的可达性分析实现精化关系检查和信息流安全性判定. TINIVER 具有灵活性和高精确性特点, 其灵活性体现在能够统一验证 timed BNNI, timed BSNNI 及 timed SIR-NNI 安全属性; 其高精确性体现在对 timed BNNI 属性的验证方法能有效避免文献 [26] 验证方法的假阳性. 相反, Gerking 等人[26]对 timed BNNI 的验证方法无法验证 timed BSNNI 属性和本文提出的 timed SIR-NNI 属性. 本文的主要贡献如下.

(1) 从现实场景的信息流安全需求出发, 提出了 timed SIR-NNI 属性, 该属性弥补了 SIR-NNI 属性无法归约系统动作时间约束的缺陷, 并克服了现有时间无干扰性无法用于检测基于输入动作反应的推断攻击的缺陷, 可更准确地描述由时间自动机抽象的实时系统的信息流安全行为.

(2) 提出了一种面向多种时间无干扰属性 (timed BNNI、timed BSNNI、timed SIR-NNI) 验证的信息流安全验证算法, 基于该算法开发了面向实时系统软件模型的信息流安全验证工具 TINIVER. TINIVER 的模型转换组件能将 SysML 顺序图模型及飞控系统 C++源码自动转换为待验证的时间自动机; 其验证组件依据不同的时间无干扰性要求, 将待验证时间自动机自动转换为精化关系检查所需的测试自动机和精化自动机, 并调用 UPPAAL 的可达性分析功能进行时间无干扰属性验证.

(3) 实验说明了 TINIVER 工具在验证 3 种时间无干扰性时的可用性, 并说明了 TINIVER 在验证 timed BNNI 时相比文献 [26] 方法的精确性. 使用 TINIVER 抽象了 ArduPilot 4.2 飞控系统的飞行模式切换模型, 并验证了从返航模式切换至定高模式, 以及从定高模式切换至自稳模式的模式切换过程满足 timed BSNNI、timed BNNI 和 timed SIR-NNI 属性. 同时通过对 PX4 1.13.2 的 C++代码进行自动建模基础上的时间无干扰性验证, 验证了 PX4 定高模式向自稳模式的切换满足时间无干扰性, 并说明了 TINIVER 的验证器后端能支持代码级和更大规模的时间无干扰性验证.

1 相关概念

本节简述与本文相关的时间自动机概念. 时间自动机[29,30]是一种用于建模系统实时行为的形式化模型, 通过在标记迁移系统上扩展引入时钟变量及其约束进行定义. 假定时域为非负实数集 $\mathbb{R}_{\geqslant 0}$. 时钟取值 (clock valuation) 函数 $v: C \to \mathbb{R}_{\geqslant 0}$ 将时钟变量集合 C 中的每个时钟变量 x 映射到 $\mathbb{R}_{\geqslant 0}$ 上的一个时刻值 $v(x)$. 定义 C 上所有时钟取值函数的集合为 $\mathbb{R}_{\geqslant 0}^{C}$, 对于时钟取值函数 $v \in \mathbb{R}_{\geqslant 0}^{C}$ 及时间 $t \in \mathbb{R}_{\geqslant 0}$, 时钟取值函数 v^{+t} 定义为对任意时钟变量 $x \in C$ 有 $v^{+t}(x) = v(x) + t$.

C 上的时钟约束 (clock constraint) 的集合由 $\Phi(C)$ 表示, 其中的任一时钟约束 φ 的语法如下:

$$\varphi ::= x \sim t | \varphi_1 \wedge \varphi_2 \ (x \in C, t \in \mathbb{R}_{\geqslant 0}, \sim \in \{<, \leqslant, =, >, \geqslant\}, \varphi_1, \varphi_2 \in \Phi(C)).$$

对于时钟取值函数 $v \in \mathbb{R}_{\geqslant 0}^{C}$, v 满足时间约束 φ (记作 $v \vDash \varphi$) 当且仅当对任意 $x \in C$, $v(x) \vDash \varphi$.

令 $r \subseteq C$ 表示重置时钟集合 (reset clocks), 使用 $v[r]$ 表示重置 r 中的时钟变量后的赋值, 即:

$$v[r](x) = \begin{cases} 0, & x \in r \\ v(x), & x \in C \backslash r \end{cases}.$$

定义 1 (时间自动机). 时间自动机 A 是一个六元组 (L,ℓ_0,C,Σ,I,E), 其中,

- L 表示位置 (location) 的有限集合, 初始位置 $\ell_0\in L$.
- C 为时钟变量的有限集合.
- Σ 为动作 (action) 的有限集合, 特别地, 内部动作 $\tau\in\Sigma$ 为不可观察动作, 其他动作 $a\in\Sigma\backslash\{\tau\}$ 为可观察动作.
- $I:L\to\Phi(C)$ 为不变量映射, 将 L 中的位置映射到时钟约束.
- $E\subseteq L\times\Phi(C)\times\Sigma\times 2^C\times L$ 表示边的集合, 对于每一条边 $(\ell,\varphi,a,r,\ell')\in E$, 可记作 $\ell\xrightarrow{\varphi,a,r}\ell'$; 其中, 时钟约束 φ 表示迁移条件 (guard), r 表示在进行迁移后需要重置的时钟变量集合.

从语义上讲, 时间自动机的状态表示为二元组 (位置, 时钟取值函数), 定义时间自动机的状态集合 $S=\{(\ell,v)\in L\times\mathbb{R}^C_{\geqslant 0}|v\vDash I(\ell)\}$, 特别地, 初始状态 $s_0=(\ell_0,0_C)$ 满足 $0_C\vDash I(\ell_0)$. 定义时间自动机的两种状态迁移如下.

(1) 离散动作迁移 $(\ell_i,v_i)\xrightarrow{a}(\ell_j,v_j)$: 当存在边 $(\ell_i,\varphi,a,r,\ell_j)$ 使得 $v_i\vDash\varphi\wedge I(\ell_i)$ 且 $v_j=v_i[r]$ 时, 从状态 (ℓ_i,v_i) 到 (ℓ_j,v_j) 的迁移.

(2) 连续时间迁移 $(\ell_i,v_i)\xrightarrow{t}(\ell_i,v_j)$: 如果 $v_j=v_i^{+t}$ 且 $\forall t'\in[0,t]\ v_i^{+t'}\vDash I(\ell_i)$ 时, 从状态 (ℓ_i,v_i) 到 (ℓ_i,v_j) 的迁移.

时间自动机 A 从状态 (l_0,v_0) 开始到 (l_m,v_n) 的运行 ρ 由连续时间迁移和离散动作迁移交替产生, 例如 $\rho=(l_0,v_0)\xrightarrow{t_1}(l_0,v_1)\xrightarrow{t_2}(l_0,v_2)\xrightarrow{a_1}(l_1,v_3)\xrightarrow{a_2}(l_2,v_4)\xrightarrow{t_3}(l_2,v_5)\ldots(l_m,v_n)$, 特别地, 使用 $s\overset{\tau}{\Rightarrow}s'$ 表示运行 $(l,v)(\xrightarrow{\tau})^*(l',v')$. 给定 $\hat{a}\in\Sigma\cup\mathbb{R}_{\geqslant 0}$ 表示动作 a 或时间 t, 但不为 τ 动作, 使用 $s\overset{\hat{a}}{\Rightarrow}s'$ 表示运行 $s\overset{\tau}{\Rightarrow}s''\xrightarrow{\hat{a}}s'''\overset{\tau}{\Rightarrow}s'$.

为描述精化关系和系统动作的敏感性, 对系统动作集合 Σ 定义两种划分. 一方面, 将 Σ 划分为互不相交的输入动作集合 Σ_I、输出动作集合 Σ_O 和内部动作 τ, 即有 $\Sigma=\Sigma_I\cup\Sigma_O\cup\{\tau\}$. 另一方面, 将 Σ 划分为互不相交的高安全级可见动作集合 Σ_H, 低安全级可见动作集合 Σ_L 和内部动作 τ, 即有 $\Sigma=\Sigma_H\cup\Sigma_L\cup\{\tau\}$. 定义 $A\backslash\Sigma$ 表示从 A 的动作集合中删除 Σ 中所有动作; $A\backslash_{\text{in}}\Sigma$ 表示从 A 的动作集合中删除 Σ 中所有输入动作; A/Σ 表示将 A 的动作集合中所有存在于 Σ 中的动作隐藏为 τ. 对于区分输入输出动作的系统, 假定 $a?$ 和 $a!$ 分别表示输入和输出动作. 特别地, 当迁移和状态的时间不受限制, 即时间约束为 $x<+\infty$ 时, 在自动机图形表示中省略迁移和状态的时间约束.

2 严格输入的时间无干扰属性

2.1 现有时间无干扰性的缺陷

当前用于定义时间无干扰性的精化关系包括时间模拟和时间互模拟. 时间模拟 (timed simulation) 是时间自动机状态集合上的偏序关系. 它要求精化时间自动机 (refined timed automata) 仅包含抽象时间自动机 (abstract timed automata) 规定的动作序列, 且精化时间自动机只在不长于抽象时间自动机所需时间间隔内执行这些动作[27]. 以下介绍现有时间模拟关系的定义.

定义 2 (时间模拟). 给定时间自动机 P 和 Q, 时间模拟关系 $\preccurlyeq_R\subseteq S_P\times S_Q$ 为 P 和 Q 状态集合 S_P 和 S_Q 上的二元关系. 假定 $((\ell_p^0,0_p),(\ell_q^0,0_q))\in\preccurlyeq_R$, $(\ell_p^0,0_p)$ 和 $(\ell_q^0,0_q)$ 分别为 P 和 Q 的初始状态, 且对于所有的 $((\ell_p,v_p),(\ell_q,v_q))\in\preccurlyeq_R$ 满足:

(a) 若 $(\ell_p,v_p)\overset{\hat{a}}{\Rightarrow}(\ell_p',v_p')$, 则存在 (ℓ_q',v_q') 使得 $(\ell_q,v_q)\overset{\hat{a}}{\Rightarrow}(\ell_q',v_q')$ 并且 $((\ell_p',v_p'),(\ell_q',v_q'))\in\preccurlyeq_R$;

(b) 若 $(\ell_p,v_p)\overset{\tau}{\Rightarrow}(\ell_p',v_p')$, 则存在 (ℓ_q',v_q') 使得 $(\ell_q,v_q)\overset{\tau}{\Rightarrow}(\ell_q',v_q')$ 并且 $((\ell_p',v_p'),(\ell_q',v_q'))\in\preccurlyeq_R$.

此时, 称 Q 时间模拟 P, 记作 $P\preccurlyeq_R Q$. 在这一关系中, P 是精化时间自动机, Q 是抽象时间自动机.

时间互模拟作为刻画实时系统间时序特征关系的手段, 其定义基于时间模拟关系给出.

定义 3 (时间互模拟). 两个时间自动机 P 和 Q 之间存在时间互模拟关系 (表示为 $P\approx Q$), 当且仅当:

(a) 存在时间模拟关系 $\preccurlyeq_R\subseteq S_P\times S_Q$ 满足 $P\preccurlyeq_R Q$;

(b) $\preccurlyeq_R$ 的逆关系 $\preccurlyeq_{R^{-1}}\subseteq S_Q\times S_P$ 满足, $\preccurlyeq_{R^{-1}}$ 为时间模拟关系且 $Q\preccurlyeq_{R^{-1}}P$.

如果两个系统在相同时间间隔内执行相同的可观察动作序列, 则它们满足时间互模拟关系[27]. 基于以上时间模拟和时间互模拟定义, 以下介绍现有的时间约束非确定无干扰属性 timed BNNI 和 timed BSNNI.

定义 4 (timed BNNI)[26]. timed BNNI 对时间自动机 A 成立, 当且仅当 $(A\backslash_{\text{in}}\Sigma_H)/\Sigma_H\approx A/\Sigma_H$ 成立.

定义 5 (timed BSNNI)[22]. timed BSNNI 对时间自动机 A 成立, 当且仅当 $A\backslash\Sigma_H \approx A/\Sigma_H$ 成立.

直观地说, timed BSNNI 相较于 timed BNNI 而言不仅要求高安全级动作的输入对低安全级用户的可观察结果没有影响, 还要求高安全级输出动作的执行情况不会影响低安全级用户的可观察结果.

由于时间互模拟关系相较于时间 trace 等价关系而言更加的严格[31], 因此基于时间互模拟的时间无干扰属性 (timed BSNNI[22]和 timed BNNI[26]) 能够描述比 tGNDC 和 tNDC 属性[24]更加严格的安全需求. 然而在一些场景下, 现有的基于时间互模拟的 timed BSNNI 和 timed BNNI 属性不足以描述更强的安全需求, 以下两个不安全用例可以佐证.

(1) 图 1 所示时间自动机满足 timed BSNNI 和 timed BNNI, 但是高安全级输入动作的执行会导致后续不一样的输入情况. 低安全级的用户可以发现, 在执行输入动作 $H1$ 之前, 可以执行两个低安全级输入动作 a 和 b, 而在执行完输入动作 $H1$ 之后仅能够执行一个低安全级输入动作 a. 攻击者可以通过观察自己在特定时间能够向系统交互的低安全级输入动作判定高安全级输入 $H1$ 是否被执行. 这种利用输入动作反应来进行推断的攻击方式, 上述基于时间互模拟的时间无干扰性 (timed BNNI 和 timed BSNNI) 无法有效检测.

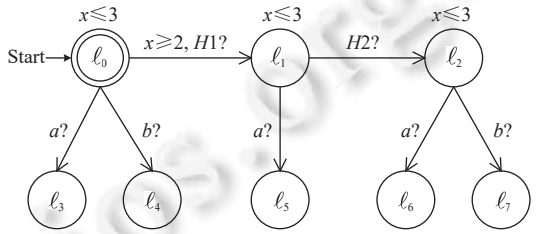


图 1 一个满足 timed BNNI/BSNNI 但存在不安全信息流的时间自动机

(2) 针对 RSA 加密的时间侧信道, Kocher[32]提出的侧信道攻击利用 RSA 参考实现代码中条件分支执行时间的差异来对 RSA 密钥进行推断. 当攻击者掌握大量明文密文对, 且对目标实现有详细了解时, 即可以大量密文作为输入并测量 RSA 实现完成私钥操作的运行时间来推测密钥. 检测这种侧信道攻击的关键即识别出这种时间信息泄露的存在. Kocher 的方法利用了 RSAREF 2.0[33]中关于模指数运算实现的缺陷, 即在进行模指数运算的迭代过程中, 每次迭代会判断当前处理的比特位的值是否为 1, 当比特位为 1 时, 因多执行一次模乘, 故运算时间较长. 首先构造 RSA 参考实现的模指数运算时间自动机. 具体地, 读取模指数运算函数的 256 次执行中比特位为 1 和为 0 的最大执行时间作为时间自动机模型上的时间约束 (分别为 85 μs 和 44 μs). 根据上述时间约束, 结合 RSAREF 2.0 的源码, 逆向构造出 RSAREF 2.0 中对应于模指数运算单次迭代执行的时间自动机, 如图 2 所示.

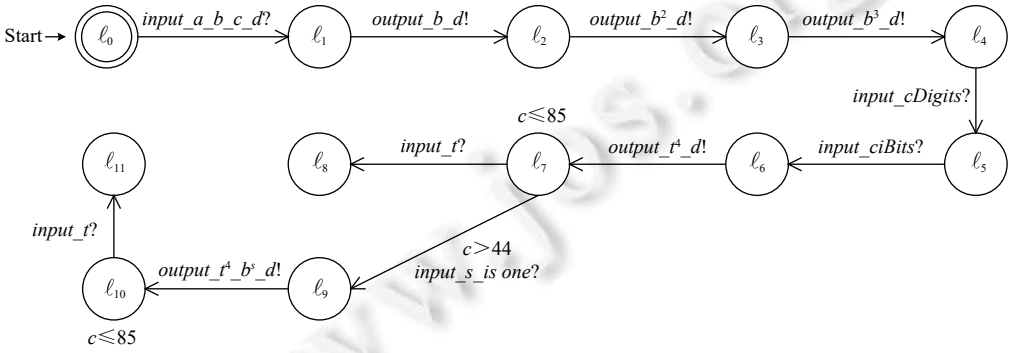


图 2 RSAREF 2.0 模指数运算时间自动机

图 2 中, 将 RSAREF 2.0 模指数运算函数中对关键变量的赋值操作作为自动机模型的输入动作, 将通过传入关键变量调用其他函数的操作作为自动机模型的输出动作. 在该自动机的 ℓ_7 位置删除高安全级输入动作 *input_s_is_one* 后在 $c \leqslant 85$ 的时间约束内可以观察到输入动作 *input_t* 执行, 这与隐藏所有的高安全级动作后观察到的结果一致,

因此该模型满足 timed BNNI 属性. 同理, 在该自动机的 ℓ_7 位置处删除所有的高安全级动作 (输入动作 *input_s_is_one* 和输出动作 *output_t*^{4}*_b*^{s}*_d*) 后的观察也与隐藏所有的高安全级动作后观察到的结果一致, 因此该模型也满足 timed BSNNI 属性. 可见, 这一被攻击的系统动作模型满足 timed BNNI 和 timed BSNNI 的定义, 但实际上是不安全的.

2.2 timed SIR-NNI 属性定义

为解决以上现实的信息流安全问题, 本文提出一种新的时间模拟关系, 即时间严格输入精化关系 (timed strict input refinement, timed SIR). 相比于 timed BNNI 和 timed BSNNI 使用的时间互模拟关系, 时间严格输入精化关系会区别定义在时间约束下系统的输入动作和输出动作的单步迁移关系, 从而对系统动作进行更细粒度区分.

定义 6 (timed SIR). 给定时间自动机 P 和 Q, 关系 $R\subseteq S_P\times S_Q$ 为 P 和 Q 状态空间集合 S_P 和 S_Q 上的二元关系, 假定 $((\ell_p^0,0_p),(\ell_q^0,0_q))\in R$, $(\ell_p^0,0_p)$ 和 $(\ell_q^0,0_q)$ 分别为 P 和 Q 的初始状态, 且对于所有的 $((\ell_p,v_p),(\ell_q,v_q))\in R$ 满足:

(a) $\forall a\in\Sigma_I$, 若有 $(\ell_q,v_q)\xrightarrow{\hat{a}}(\ell'_q,v'_q)$, 则存在 (ℓ'_p,v'_p) 使得 $(\ell_p,v_p)\xrightarrow{\hat{a}}(\ell'_p,v'_p)$ 并且 $((\ell'_p,v'_p),(\ell'_q,v'_q))\in R$;

(b) $\forall a\in\Sigma_I$, 若有 $(\ell_p,v_p)\xrightarrow{\hat{a}}(\ell'_p,v'_p)$, 则存在 (ℓ'_q,v'_q) 使得 $(\ell_q,v_q)\xrightarrow{\hat{a}}(\ell'_q,v'_q)$ 并且 $((\ell'_p,v'_p),(\ell'_q,v'_q))\in R$;

(c) $\forall a\in\Sigma_O$, 若有 $(\ell_p,v_p)\xrightarrow{\hat{a}}(\ell'_p,v'_p)$, 则存在 (ℓ'_q,v'_q) 使得 $(\ell_q,v_q)\overset{\tau}{\Rightarrow}\xrightarrow{\hat{a}}(\ell'_q,v'_q)$ 并且 $((\ell'_p,v'_p),(\ell'_q,v'_q))\in R$;

(d) 若有 $(\ell_p,v_p)\xrightarrow{\tau}(\ell'_p,v'_p)$, 则存在 (ℓ'_q,v'_q) 使得 $(\ell_q,v_q)\overset{\tau}{\Rightarrow}(\ell'_q,v'_q)$ 并且 $((\ell'_p,v'_p),(\ell'_q,v'_q))\in R$;

则称 P 时间严格输入精化 Q, 记作 $P\preccurlyeq_{TS}Q$.

时间互模拟关系对系统输入动作的限制不够严格, 允许两个系统并发执行相同输入动作迁移的前序和后序参杂 τ 动作. 相对地, timed SIR 关系对系统输入动作有严格限制, 要求两个系统之间任一方的输入动作在时间约束下总可以被另一方严格模拟 (即执行相同输入动作迁移的前序和后序不能参杂 τ 动作), 抽象系统与精化系统在执行输入动作时应表现一致, 而精化系统在执行其他动作时遵循抽象系统所规定的单向一致性. 此外, timed SIR 定义中的 $\hat{a}$ 具备时间属性约束, 要求系统间的离散动作迁移一致且连续时间迁移一致.

进一步地, 本文扩展 Lee 等人[11]基于严格输入精化 (SIR) 关系的安全属性定义, 提出了支持时间约束的时间严格输入精化非确定无干扰属性 (timed SIR-NNI).

定义 7 (timed SIR-NNI). 时间严格输入精化非确定无干扰性 timed SIR-NNI 对时间自动机 A 成立, 当且仅当 $(A\backslash_{\text{in}}\Sigma_H)/\Sigma_H\succcurlyeq_{TS}A/\Sigma_H$ 成立.

timed SIR-NNI 属性与基于时间互模拟关系的 timed BNNI 和 timed BSNNI 的主要差异在于定义使用的精化关系不同. 以下针对第 2.1 节的两个不安全系统示例, 说明 timed SIR-NNI 在刻画利用输入动作反应的推断攻击方面, 所具有的独特性.

(1) 图 1 所示自动机不满足 timed SIR-NNI 属性. 根据 timed SIR, 低安全级用户在满足时间约束 $x\leqslant 3$ 的时间内, 于图 1 自动机的 ℓ_1 位置仅会看到该系统执行输入动作 *a* 而看不到执行输入动作 *b*, 这与图 1 自动机在删除高安全级输入动作 *H*1 后形成的自动机在 ℓ_0 观察到的情况不同 (删除输入动作 *H*1 后, 低安全级用户在满足时间约束 $x\leqslant 3$ 的时间内, 于 ℓ_0 位置可见输入动作 *a* 或 *b* 均可被执行). 因此, 图 1 示例不满足 timed SIR-NNI. 对于满足 timed BNNI 和 timed BSNNI 属性的系统, 攻击者可能利用系统对输入动作的反应推断机密动作, timed SIR-NNI 可以规约这一安全需求, 识别出此类攻击的发生. 说明本文提出的 timed SIR-NNI 属性的不可替代性.

(2) Kocher 的 RSA 时间侧信道攻击[32]对应于 RSA 参考实现的模指数运算时间自动机 (图 2) 亦不满足 timed SIR-NNI. 在模指数运算函数中, 变量 *s* 是否取值为 1 作为安全关键信息不希望披露给攻击者, 同理攻击者也不能观察到变量 *s* 取值为 1 时额外执行的模运算操作, 因此将输入动作 *input_s_is_one* 和输出动作 *output_t*^{4}*_b*^{s}*_d* 作为高安全级动作. 在图 2 中输入动作 *input_s_is_one* 执行与否会导致执行时间的差别. 当输入动作 *input_s_is_one* 被删除时, 观测到输入动作 *input_t* 的执行时间始终不超过 44 μs; 而在输入动作 *input_s_is_one* 可执行的情况下, 观察到输入动作 *input_t* 的执行时间会超过 44 μs. 高安全级动作 *input_s_is_one* 执行与否会影响到攻击者的观察时间. 攻击者通过这种执行时间的变化可判定当前比特位是否为 1, 因此 RSA 参考实现不满足 timed SIR-NNI 蕴含 RSA 时间侧信道攻击的存在.

3 多时间无干扰属性验证方法

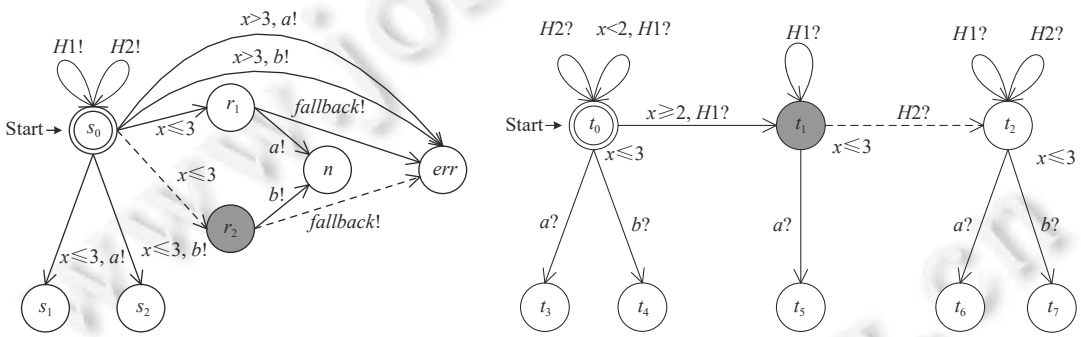
有效的自动验证方法, 对于安全属性应用于实际系统模型至关重要. 本节提出一种兼容包括本文 timed SIR-NNI 在内的多种时间无干扰属性的自动验证方法, 并克服了现有 timed BNNI 验证方法[26]的缺陷.

3.1 现有 timed BNNI 验证方法缺陷

Gerking 等人[26]对 timed BNNI 属性的验证方法存在以下两方面明显缺陷.

(1) 无法验证 timed BSNNI 属性[22]和本文提出的 timed SIR-NNI 属性. 主要原因为: ① 该验证方法仅考虑了对高安全级输入动作的限制, 而 timed BSNNI 属性要求限制所有高安全级动作. ② 该验证方法是基于 Heinzemann 等人[27]的时间互模拟精化关系验证方法展开的, 时间互模拟精化关系验证方法不能用于验证 timed SIR 精化关系, 原因是 timed SIR 定义要求满足 timed SIR 关系的两个自动机之间的每个低安全级输入动作迁移都应严格一致 (不能包含 τ 迁移), 而时间互模拟精化关系定义要求满足时间互模拟关系的两个自动机所执行的每个低安全级动作迁移都在容忍前序和后序 τ 迁移的情况下一致, 该验证方法未针对二者定义差异提供时间自动机的不同构造和变换方法. 故该验证方法不能验证基于 timed SIR 关系的 timed SIR-NNI 属性.

(2) 对 timed BNNI 属性的验证存在假阳性问题. 具体地, 文献 [26] 验证方法将对无干扰性的检查限制在单个位置上, 在某个位置不允许的所有可见事件都将被视为不安全的信息流, 即使自动机通过不可观察动作迁移到允许该事件发生的未来某位置是可能的. 例如, 对于图 1 自动机, 图 3(a) 和图 3(b) 分别为其测试自动机和精化自动机. 将这两个自动机输入文献 [26] 的 timed BNNI 判定过程.



(a) 图 1 自动机的测试自动机 (b) 图 1 自动机的精化自动机

图 3 现有 timed BNNI 验证方法的判定方式的缺陷

① 测试自动机的 s_0 位置会检查精化自动机的 t_0 位置是否有违反 timed BNNI 的情况. 根据并行组合的规则, 当测试自动机处于 s_0 且精化自动机处于 t_0 时, 无论是否进行状态迁移, 测试自动机的 *err* 位置始终不可达.

② 随时间演进, 当 $0 \leqslant x < 2$, t_0 可能被 r_1 或 r_2 测试, 因为 r_1 和 r_2 均有适当的输出动作与 t_0 的输入动作并发执行, 这两种情况均不会到达 *err* 位置.

③ 当精化自动机迁移至位置 t_1 (时间约束 $2 \leqslant x \leqslant 3$), t_1 可能被测试自动机的位置 s_0、r_1 或 r_2 测试. 当 t_1 被 s_0 或 r_1 测试时, 测试自动机可以通过输出动作 *a* 并发执行 t_1 位置的可执行输入动作 *a*, 不会到达 *err* 位置; 而当 t_1 被灰色位置 r_2 测试时, 由于测试自动机在 r_2 位置无法执行精化自动机 t_1 位置的可执行输入动作 (*a* 和 *H*2), 因此测试自动机会执行输出动作 *fallback*. *fallback* 动作用于为测试自动机无法匹配的精化自动机低安全级动作提供匹配 (文献 [26] 要求当测试自动机当前状态下没有任何可与精化自动机并行执行的动作时, 执行此动作到达 *err* 位置), *fallback* 动作并不会导致精化自动机随着时间演进从而使得 t_1 位置不执行任何动作. 故测试自动机的 *err* 位置在 $x \leqslant 3$ 的时间内可达. 根据文献 [26] 方法, 判定图 1 自动机违反 timed BNNI.

以上判定结果与 timed BNNI 定义不符. 根据 timed BNNI 定义, 在测试 t_1 位置时是否会被测试为不安全时, 不应急于根据 r_2 位置可能发生的迁移判定位置 t_1 导致系统不安全 (图 3(a) 中的虚线迁移是为了强调文献 [26] 的判

定过程输出假阳性结果时所采取的迁移). 由于低安全级的用户无法区分位置 t_1 和 t_2 ($H2$ 为不可观察输入动作), 因此只要位置 t_1 和 t_2 中的任意一个通过测试自动机的测试, 即可认为图 1 自动机满足 timed BNNI. 所以应继续检测精化自动机通过从 t_1 执行不可观察迁移 (图 3(b) 中虚线迁移) 到达的 t_2 是否会使得 err 位置可达. 易见, 精化自动机的 t_2 位置仍可能被测试自动机的位置 s_0 、 r_1 或 r_2 测试. 当 t_2 被测试自动机的 s_0 位置测试时, 由于 t_2 的时间约束 $x \leqslant 3$, 测试自动机无法通过 $x > 3$ 约束的两个动作并发执行到 err 位置; 当 t_2 被测试自动机的 r_1 或 r_2 位置测试时, 测试自动机总会优先执行输出动作 a 或 b (因为 t_2 位置总有匹配执行的输入动作 a 或 b). 由于 t_2 不会导致 err 位置可达, 因此实际上图 1 自动机满足 timed BNNI. 文献 [26] 验证方法的精化自动机由于不会执行图 3(b) 中的虚线迁移, 因而导致文献 [26] 方法验证结果为假阳性.

3.2 多种无干扰性的兼容验证

时间无干扰性的基本验证方法[26]是将时间无干扰性的验证问题转化为可达性分析问题. 通过构造测试自动机, 规定原自动机在满足时间无干扰属性时所允许和禁止执行的动作. 通过构造精化自动机以保留原自动机的执行行为. 在对测试自动机和精化自动机并行执行时, 若精化自动机执行了时间无干扰属性不允许的动作, 则导致测试自动机中的 err 位置可达, 反之则不可达. 当不考虑时间约束时, 原始的 SIR-NNI 的验证可参见文献 [11]. 第 3.1 节已说明了 Gerking 等人[26]的方法无法验证 timed BSNNI 属性[22]和本文提出的 timed SIR-NNI 属性. 这种针对单一属性的验证在对复杂系统进行验证时缺乏灵活性. 本节在分析 timed BSNNI, timed BNNI 和 timed SIR-NNI 属性间差异的基础上, 给出兼容这 3 种属性验证的自动验证算法.

时间约束的非确定无干扰属性 timed BNNI 和 timed BSNNI 均使用时间互模拟关系定义, 且都要求指定的高安全级动作的执行与否对系统的攻击者视角 (将系统的所有高安全级可见动作均看作内部动作) 没有影响, 即无论系统 A 是否执行指定的高安全级动作, 从攻击者视角, 系统的执行均能符合相同的时间属性 (这些系统执行之间满足时间互模拟关系), 这一时间属性保证了系统的低安全级时序动作的一致性. 但二者之间亦存在明显差异: 在所指定的高安全级动作的范围上, timed BNNI 仅要求高安全级输入动作的执行与否对系统的攻击者视角没有影响; 而 timed BSNNI 相比 timed BNNI 更加严格, 不仅局限于高安全级输入动作执行与否对系统的攻击者视角的影响, 而是要求所有高安全级动作执行与否都不能干扰到该系统的攻击者视角. 以图 4 所示时间自动机为例, 该自动机满足 timed BNNI, 但不满足 timed BSNNI. 该自动机在执行完高安全级输出动作 H 的情况下可以执行输出动作 b 但不能执行输出动作 a, 而在执行该高安全级输出动作 H 之前可以执行输出动作 a 而不能执行输出动作 b, 这不满足时间互模拟的定义, 因此不满足 timed BSNNI 属性. 但该自动机中并不包含任何高安全级输入动作, 因此安全属性对于系统高安全级输入动作的删除要求不会影响系统的攻击者视角, 因此该系统满足 timed BNNI.

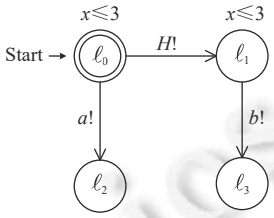


图 4 一个满足 timed BNNI 但不满足 timed BSNNI 的自动机

timed BNNI 和 timed BSNNI 均基于时间互模拟. 时间互模拟扩充了时间模拟的要求, 反向要求抽象系统的时间行为能够被精化系统模拟, 以达到抽象系统和精化系统的行为表现的基本一致. 时间互模拟所作的限制与 timed SIR 相比, 对输入动作的要求不够严格 (满足时间互模拟关系的系统间, 在执行相同输入动作迁移的前序和后序可参杂 τ 迁移). timed SIR 在执行输入动作时, 要求抽象系统与精化系统应表现一致, 即一致的输入动作迁移的前序和后序不参杂 τ 迁移; 而在执行其他动作时, timed SIR 仅要求精化系统遵循抽象系统所规定的单向一致性.

针对上述属性差异和特点, 以及 Gerking 等人[26]验证方法的缺陷, 以下提出一种通用的时间无干扰性验证方

法 (TINIVER), 能够统一验证现有时间无干扰性 timed BNNI、timed BSNNI 和本文提出的 timed SIR-NNI, 且能解决上述假阳性问题. TINIVER 验证算法首先为待检查无干扰性的自动机 $A=(L,\ell_0,C,\Sigma,I,E)$ 构造两个副本: $A_1=(L_1,\ell_0^1,C_1,\Sigma_1,I_1,E_1)$ 和 $A_2=(L_2,\ell_0^2,C_2,\Sigma_2,I_2,E_2)$, 以这两个副本分别作为输入, 构建测试自动机 (第 3.2.1 节) 和精化自动机 (第 3.2.2 节), 然后以构建的测试自动机和精化自动机作为输入验证 3 种无干扰性 (第 3.2.3 节).

3.2.1 构造测试自动机

算法 1 描述了构造测试自动机的过程, 输入为原自动机的副本 $A_1=(L_1,\ell_0^1,C_1,\Sigma_1,I_1,E_1)$ 以及待验证的属性 P. 输出为测试自动机 $A_{\text{test}}=(L_t,\ell_0^t,C_t,\Sigma_t,I_t,E_t)$. 为使测试自动机 A_{test} 能够用于检查 timed BNNI, timed BSNNI 和 timed SIR-NNI, 本文使用了与文献 [26] 不同的构造过程. 主要区别在于以下方面: TINIVER 验证 timed BSNNI 时, 会将所有高安全级动作均删除, 而不仅删除高安全级输入动作 (如算法 1 第 5–9 行); TINIVER 验证 timed SIR-NNI 时, 仅要求 A_1 执行的每个低安全级输入动作均能被 A_2 执行, 而非要求所有的低安全级动作均能被 A_2 执行. 具体的测试自动机构造过程如下.

1) 构造算法须确保 A_{test} 能够容许 A_1 的动作, 即保证当测试自动机检测到这些动作执行时不会切换到 *err* 位置. 如算法 1 第 1–4 行所示, 保留 A_1 的每个迁移, 但是改变了每个迁移上的 guard, 将原始 guard 和起始位置的不变量合取作为新的 guard (这样保证了在 A_1 的迁移被激活的时间内 A_{test} 亦可被激活, 且便于后续检测违反时间的迁移的发生).

2) 为便于后续与精化自动机的并行组合, 将 A_{test} 的动作集合 Σ_t 中的每个动作的后缀取反 (即将后缀!变为?, 将后缀?变为!), 如算法 1 第 11 行.

3) 由于 timed BNNI, timed BSNNI 和 timed SIR-NNI 都要求 A_2 的每个低安全级动作也被 A_1 执行, 因此, 要求 A_{test} 能够检测出原自动机违反这些无干扰属性. 可进一步分两种情况: ① A_2 执行了 A_1 中没有定义的低安全级动作. ② A_2 虽执行了 A_1 中定义的低安全级动作, 但未在允许的时间内执行. 如算法 1 第 12–20 行及第 22–28 行所示, 对于情况①, 使用 $\Theta(\ell)$ 表示每个位置 ℓ 外向边上的动作的集合 (即处于位置 ℓ 时可执行的动作) 并保证当检测到 A_2 在该位置执行了会违反安全性的低安全级动作时, 测试自动机到达 *err* 位置. 而对于那些会导致安全性不满足的高安全级动作, 在测试自动机中创建一个 ℓ 到自身的循环, 过滤对这种情况的检查的同时, 达到隐藏高安全级动作的目的. 对于情况②, 为所有 A_1 允许执行的低安全级动作在测试自动机中创建一条到 *err* 位置的迁移, 该迁移的 guard 为 A_1 中对应迁移的 guard 取反 (即检查 A_2 的执行时间是否在不允许的时间范围内). 类似地, 创建循环过滤高安全级的动作.

4) 需要检测出 A_1 执行的所有低安全级动作亦会被 A_2 执行. timed BNNI 和 timed BSNNI 均要求 A_1 执行的每一个低安全级动作都能被 A_2 执行, 而 timed SIR-NNI 仅要求 A_1 执行的每一个低安全级输入动作都能被 A_2 执行. 如算法 1 第 29–34 行所示, 判断输入的无干扰属性是否为 timed SIR-NNI, 若是, 则只为 A_1 中每个包含低安全级输入动作的迁移创建位置 r. 如果 A_1 执行的低安全级输入动作在正确的时间内被 A_2 执行, 那么测试自动机的位置 n (在算法 1 的第 10 行和 *err* 位置一起创建) 可达; 否则 *err* 位置可达. 若输入的属性不为 timed SIR-NNI, 则为 A_1 中每个包含低安全级动作的迁移创建位置 r_i.

算法 1. 构造测试自动机.

输入: 原自动机的副本 $A_1=(L_1,\ell_0^1,C_1,\Sigma_1,I_1,E_1)$, 待验证属性 P;
输出: 测试自动机 $A_{\text{test}}=(L_t,\ell_0^t,C_t,\Sigma_t,I_t,E_t)$.

1. **foreach** $(\ell,\varphi,a,r,\ell')\in A_1$ **do**
2. $\quad\varphi\leftarrow\varphi\wedge I(\ell)$
3. **done**
4. $A_{\text{test}}\leftarrow A_1$ /*将测试自动机初始化为 A_1 */
5. **if** (P == BSNNI) **then**

6. $A_{\text{test}} \leftarrow A_{\text{test}} \backslash \Sigma_H$
7. **else**
8. $A_{\text{test}} \leftarrow A_{\text{test}} \backslash_{\text{in}} \Sigma_H$
9. **endif**
10. $L_t \leftarrow L_t \cup \{err, n\}$
11. $\Sigma_t \leftarrow$ switch_suffix(Σ_t) /*将测试自动机中所有的后缀取反, 即将!变为?, 将?变为!*/
12. **foreach** $\ell \in A_1$ **do**
13. $\Theta(\ell) \leftarrow$ outgoing_edge_actions(ℓ) /*定义 $\Theta(\ell)$ 为从位置 ℓ 开始的所有外向边上的动作的集合*/
14. **foreach** $a \in \Sigma_H \backslash \Theta(\ell)$ **do**
15. $E_t \leftarrow E_t \cup \{(\ell, \text{null}, a, \text{null}, \ell)\}$
16. **done**
17. **foreach** $a \in \Sigma_L \backslash \Theta(\ell)$ **do**
18. $E_t \leftarrow E_t \cup \{(\ell, I_t(\ell), a, \text{null}, err)\}$
19. **done**
20. **done**
21. **foreach** $(\ell, \varphi, a, r, \ell') \in A_1$ **do**
22. **if** $\varphi \neq \text{null}$ **then**
23. **if** $a \in \Sigma_L$ **then**
24. $E_t \leftarrow E_t \cup \{(\ell, \neg\varphi, a, \text{null}, err)\}$
25. **else**
26. $E_t \leftarrow E_t \cup \{(\ell, \neg\varphi, a, \text{null}, \ell)\}$
27. **endif**
28. **endif**
29. **if** $(P == \text{SIR-NNI} \wedge a \in \Sigma_L \wedge a \in \Sigma_t) \vee (P \neq \text{SIR-NNI} \wedge a \in \Sigma_L)$ **then**
30. $L_t \leftarrow L_t \cup \{r_i\}$
31. $E_t \leftarrow E_t \cup \{(\ell, \varphi, \text{null}, \text{null}, r_i)\}$
32. $E_t \leftarrow E_t \cup \{(r_i, \text{null}, a, \text{null}, n)\}$
33. $E_t \leftarrow E_t \cup \{(r_i, \text{null}, fallback!, \text{null}, err)\}$
34. **endif**
35. **done**
36. $I_t \leftarrow \text{null}$
37. **return** $(L_t, \ell'_0, C_t, \Sigma_t, I_t, E_t)$

根据算法 1, 对于原始时间自动机 $A = (L, \ell_0, C, \Sigma, I, E)$ 的任一迁移 $(\ell, \varphi, a, \text{null}, \ell')$, 该迁移在不同安全属性所需测试自动机中将转化成不同的形式, 如后文表 1 所示. 注意本文针对的安全属性与时钟重置无关, 因而所有迁移的需重置时钟变量集合 r 总为 null . 表 1 中及后文中"_"表示"无 guard"或"无动作". 结合第 3.2.2 节表 2 中的构造规则, 使得 UPPAAL 对测试自动机和精化自动机进行并发执行过程中的 *err* 位置可达性分析分别等价于验证 timed BNNI、timed BSNNI 和 timed SIR-NNI 定义 (即定义 4、5、7) 中的时间互模拟关系或时间严格输入精化关系是否成立 (见第 3.2.3 节).

3.2.2 构造精化自动机

为了隐藏 A_2 的高安全级动作并调整 A_2 以保证 A_2 被正确地用于并行组合, 需要在 A_2 的基础上构造精化自动机 A_{ref} . timed BNNI, timed BSNNI 和 timed SIR-NNI 这 3 种属性要求原自动机在删除了高安全级 (输入) 动作的副

本与原自动机隐藏高安全级动作的副本之间满足不同的精化关系. 由于在测试自动机构造算法中, 已经考虑了删除不同的高安全级动作以及对不同精化关系的处理方法, 因此在构造精化自动机时仅需要考虑隐藏原自动机高安全级动作. 而上述 3 种属性都仅要求隐藏高安全级动作, 因此针对上述 3 种属性以 A_2 为基础构造精化自动机的方法相同, 具体构造规则如表 2 所示. 和测试自动机构造算法中隐藏高安全级动作的方法一样, 我们检查 A_2 的每一个位置 ℓ, 如果存在从该位置发出的高安全级动作迁移, 则为每一个未从该位置发出的高安全级动作创建一个从该位置到自身的迁移 (迁移动作为该高安全级动作), 并为每一个从该位置发出的 guard 为 φ 的高安全级动作创建一个从该位置到自身、guard 为 $\neg\varphi$ 且以该高安全级动作为迁移动作的迁移. 根据表 1 与表 2 中针对高安全级动作的构造规则, 测试自动机和精化自动机对于高安全级动作的删除或隐藏处理体现了 timed BNNI、timed BSNNI 和 timed SIR-NNI 定义 (即定义 4、定义 5、定义 7) 中的关系两端的时间自动机变换构造. 删除高安全级动作边, 即让当前位置原本被允许的高安全级动作不能向下演进; 隐藏高安全级动作边, 即描述在任一位置上, 不被允许的高安全级动作不会使得时间自动机向下演进. 这样, 对于被允许执行的高安全级动作边, 在测试自动机和精化自动机上构造出了无干扰性所要求的差异.

表 1　安全属性所需测试自动机的构造规则

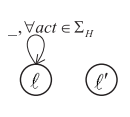
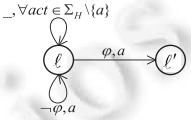
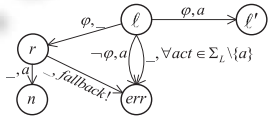
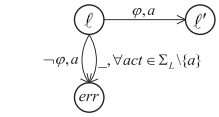
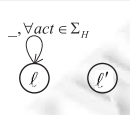
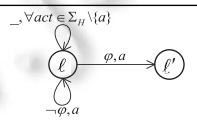
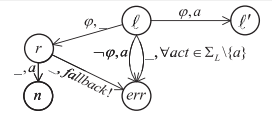
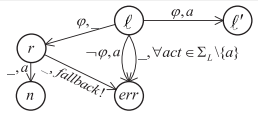
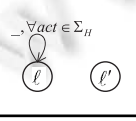
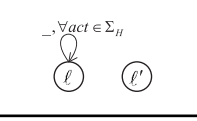
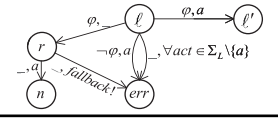
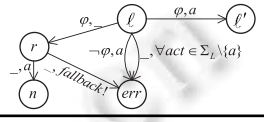
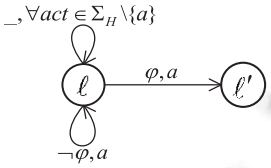
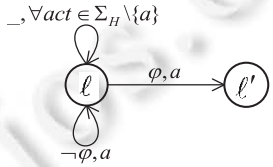
属性	边 $(\ell,\varphi,a,\text{null},\ell')$ 的 a 动作的类型			
	高安全级输入	高安全级输出	低安全级输入	低安全级输出
timed SIR-NNI				
timed BNNI				
timed BSNNI				

表 2　安全属性所需精化自动机的构造规则

属性	边 $(\ell,\varphi,a,\text{null},\ell')$ 的 a 动作的类型			
	高安全级输入	高安全级输出	低安全级输入	低安全级输出
timed SIR-NNI/BNNI/BSNNI			保持不变	保持不变

3.2.3　TINIVER 验证算法

TINIVER 验证算法如算法 2 所示. 以第 3.2.1 节和第 3.2.2 节生成的测试自动机和精化自动机作为输入, 输出对原始时间自动机是否满足无干扰性的判定. TINIVER 会调用 UPPAAL 的可达性分析, 如算法 2 第 3 行所示, 利用该分析方法检查 *err* 位置的可达性并返回一个 trace: 若 *err* 位置不可达, 则 trace 为空, 利用布尔型全局变量 result 记录输出的值 true; 若 *err* 位置可达, 则返回一个到达 *err* 位置的 trace. 具体分两种情况: ① 直接到达 *err* 位置, 即 $\text{trace}=\ldots\to(\ell_{\text{test}},\ell_{\text{ref}})\to(err,\_)$. 这种情况说明 A_2 执行了 A_1 中没有定义的低安全级动作或者未在正确的时间内执行可执行的动作. ② 通过额外添加的位置 (r_i 位置) 到达 *err* 位置, 即 $\text{trace}=\ldots\to(\ell_{\text{test}},\ell_{\text{ref}})\to(r_i,\_)\to(err,\_)$.

这种情况说明存在 A_1 可执行的低安全级动作未被 A_2 执行. trace 反映出精化自动机的 ℓ_{ref} 位置没有通过测试自动机 ℓ_{test} 位置的测试, 有可能存在违反无干扰性的情况, 需要进行进一步的处理.

1) 对于 trace = … → $(\ell_{\text{test}}, \ell_{\text{ref}})$ → $(err, \_)$ 的情况, 需检查以 ℓ_{ref} 为起始位置的迁移上是否有高安全级动作, 仅考虑高安全级动作是因为高安全级动作对于低安全级用户不可观察, 使得 ℓ_{ref} 和 $next_i(\ell_{\text{ref}})$ 在低安全级用户看来表现得一样. 如果不存在此类迁移, 则说明测试自动机确实检测出了违反无干扰性的情况, 使用 result 记录结果 false (算法 2 第 9–11 行). 如果存在此类迁移, 则需进行进一步的检查(算法 2 第 12–18 行): 首先, 需要将测试自动机的初始位置更新为 ℓ_{test} 得到新的测试自动机. 然后, 需要判断 ℓ_{ref} 位置通过高安全级迁移到达的下一位置及后续位置是否可能通过 ℓ_{test} 的测试. 因此, 为每个从 ℓ_{ref} 到 $next_i(\ell_{\text{ref}})$ 的高安全级迁移创建一个精化自动机副本, 并将该副本的初始位置更新为 $next_i(\ell_{\text{ref}})$, 得到若干新的精化自动机. 将新测试自动机与每个新精化自动机递归使用本算法继续测试.

2) 对于 trace = … → $(\ell_{\text{test}}, \ell_{\text{ref}})$ → $(r_i, \_)$ → $(err, \_)$ 的情况, 仅对 timed BSNNI 和 timed BNNI 进行上述 1) 的验证步骤. 对于 timed SIR-NNI, 如算法 2 的第 19–22 行所示, 直接使用 result 记录结果 false. 这样做是由于 timed SIR-NNI 要求 A_2 的每一个低安全级输入迁移都必须被 A_1 严格执行 (不能参杂 τ 迁移), 即 A_2 中执行了输入迁移 $\xrightarrow{a}$ 时, A_1 也必须执行 $\xrightarrow{a}$ 而不能执行 $\overset{\tau}{\Rightarrow}\xrightarrow{a}\overset{\tau}{\Rightarrow}$.

算法 2. TINIVER 验证算法.

输入: 测试自动机 $A_{\text{test}} = (L_t, \ell_0^t, C_t, \Sigma_t, I_t, E_t)$, 无干扰属性 P, 精化自动机 $A_{\text{ref}} = (L_r, \ell_0^r, C_r, \Sigma_r, I_r, E_r)$;
输出: NI satisfied/not satisfied.

```
1. bool result ← false
2. FUNC Rec_Uppaal_reachability( A_test , A_ref )
3.    trace ← Uppaal_reachability( A_test , A_ref )
4.    if trace == ∅ then /* err 位置不可达*/
5.       result ← result ∨ true
6.       return
7.    else if P !=SIR-NNI or trace = … → (ℓ_test, ℓ_ref) → (err, _) then /*trace 不为空有两种情况, 即 ℓ_test 经过 r_i
8.                                                                  位置到达 err 位置或直接到达 err 位置*/
9.       if ∃(ℓ_ref, _, a ∈ Σ_H, _, _) ∉ A_ref then /* A_ref 中不存在从 ℓ_ref 开始的高安全级迁移*/
10.         result ← result ∨ false
11.         return
12.      else /* A_ref 中存在从 ℓ_ref 开始的高安全级迁移*/
13.         A_test ← (L_t, ℓ_test, C_t, Σ_t, I_t, E_t)
14.         foreach (ℓ_ref, _, a_i ∈ Σ_H, _, next_i(ℓ_ref)) ∈ A_ref do
15.            A_ref ← (L_r, next_i(ℓ_ref), C_r, Σ_r, I_r, E_r)
16.            Rec_Uppaal_reachability( A_test , A_ref )
17.         done
18.      endif
19.   else /* trace = … → (ℓ_test, ℓ_ref) → (r_i, _) → (err, _) 并且 P =SIR-NNI*/
20.      result ← result ∨ false
21.      return
22.   endif
23. endFUNC
```

24. **if** result **then**
25. *P* satisfied
26. **else**
27. *P* not satisfied
28. **endif**

算法 2 需要为 timed BNNI 和 timed BSNNI 属性验证时间互模拟关系 (定义 3) 成立, 并为 timed SIR-NNI 属性验证时间严格输入精化关系 (定义 6) 成立. 以下结合表 1 和表 2 的自动机迁移构造规则, 分情况说明算法 2 利用 UPPAAL 进行的并发执行 *err* 位置可达性分析能够验证定义 4、定义 5、定义 7 中的时间互模拟关系和时间严格输入精化关系.

1) 时间互模拟关系 (定义 3) 的规则 (a) 的实现. 对于低安全级动作, 时间互模拟关系的规则 (a) 要求测试自动机执行的低安全级动作均能够在正确的时间约束内被精化自动机执行. 根据表 1 和表 2 中对 timed BNNI 和 timed BSNNI 所要求的低安全级动作迁移的构造, 当测试自动机和精化自动机在当前位置均存在满足时间约束 φ 的低安全级动作迁移 a, 在表 1 测试自动机规则下, 测试自动机会同时演进到位置 ℓ' 和 n. 当测试自动机在 ℓ 位置存在 φ 时间约束的低安全级动作迁移 a, 而精化自动机在 ℓ 的对应位置存在不满足 φ 的低安全级动作迁移 a 时, 测试自动机进入 *err* 位置. 当测试自动机在 ℓ 位置存在 φ 时间约束的低安全级动作迁移 a, 而精化自动机在 ℓ 的对应位置存在与 a 不同的低安全级动作迁移 b, 若 b 迁移不满足 φ 时间约束, 则测试自动机进入 *err* 位置; 若 b 迁移满足 φ 时间约束, 则 b 迁移经由 r 位置进入 *err* 位置. 上述进入 *err* 位置的条件与违反定义 3 的规则 (a) 的情况一致.

2) 时间互模拟关系 (定义 3) 的规则 (b) 的实现. 时间互模拟关系的规则 (b) 要求精化自动机执行的低安全级动作均能够在正确的时间约束内被测试自动机执行. 根据表 1 和表 2 中对 timed BNNI 和 timed BSNNI 所要求的低安全级动作迁移的构造, 当精化自动机在 ℓ 位置存在 φ 时间约束的低安全级动作迁移 a, 而测试自动机在 ℓ 的对应位置存在不满足 φ 的低安全级动作迁移 a 时, 测试自动机进入 *err* 位置. 当精化自动机在 ℓ 位置存在 φ 时间约束的低安全级动作迁移 a, 而测试自动机在 ℓ 的对应位置存在与 a 不同的低安全级动作迁移 b 时, 由于精化自动机的 a 迁移满足 φ 时间约束, 因而测试自动机可以直接进入 *err* 位置或经由 r 位置进入 *err* 位置. 上述进入 *err* 位置的条件与违反定义 3 的规则 (b) 的情况一致.

3) 时间严格输入精化关系 (定义 6) 的规则 (a) 的实现. 对于低安全级输入动作, 时间严格输入精化关系的规则 (a) 要求测试自动机执行的低安全级输入动作均能够在正确的时间约束内被精化自动机执行. 根据表 1 和表 2 中对 timed SIR-NNI 所要求的低安全级输入动作的构造, 测试自动机与精化自动机的并发执行情况完全类似于上述对时间互模拟关系 (定义 3) 的规则 (a) 的实现, 区别仅在于 timed BNNI 和 timed BSNNI 要求在所有低安全级动作上实现该可达性分析, 而 timed SIR-NNI 仅要求在低安全级输入动作上实现. 进入 *err* 位置的各种条件与违反定义 6 的规则 (a) 的情况一致.

4) 时间严格输入精化关系 (定义 6) 的规则 (b) 的实现. 对于低安全级输入动作, 时间严格输入精化关系的规则 (b) 要求精化自动机执行的低安全级输入动作均能够在正确的时间约束内被测试自动机执行. 根据表 1 和表 2 中对 timed SIR-NNI 所要求的低安全级输入动作的构造, 测试自动机与精化自动机的并发执行情况完全类似于上述情况 2 (即对时间互模拟关系的规则 (b) 的实现), 区别仅在于 timed BNNI 和 timed BSNNI 要求在所有低安全级动作上实现该可达性分析, 而 timed SIR-NNI 仅要求在低安全级输入动作上实现. 进入 *err* 位置的各种条件与违反定义 6 的规则 (b) 的情况一致.

5) 时间严格输入精化关系 (定义 6) 的规则 (c) 的实现. 对于低安全级输出动作, 时间严格输入精化关系的规则 (c) 要求精化自动机执行的低安全级输出动作均能够在正确的时间约束内被测试自动机执行. 根据表 1 和表 2 中对 timed SIR-NNI 所要求的低安全级输出动作的构造, 当测试自动机和精化自动机在当前位置均存在满足时间约束 φ 的低安全级输出动作迁移 a, 在表 1 测试自动机规则下, 测试自动机会演进到位置 ℓ'. 当精化自动机在 ℓ 位置存在 φ 时间约束的低安全级输出动作迁移 a, 而测试自动机在 ℓ 的对应位置存在与 a 不同的低安全级动作迁移

b 或存在不满足 φ 的低安全级输出动作迁移 a 时, 则测试自动机会进入 err 位置. 进入 err 位置的各种条件与违反定义 6 的规则 (c) 的情况一致. 对于低安全级输出, 不需要引入位置 r 和 n 的构造的原因是由于定义 6 对于低安全级输出的规定具有单向性, 上述情况 2) 和 4) 所使用的直接进入 err 位置的路径已足够支持对违反定义 6 规则 (c) 的情况进行验证.

6) 对于 τ 动作迁移 (包括时间严格输入精化关系 (定义 6) 的规则 (d)) 的实现. 表 1 和表 2 中隐藏和删除高安全级动作的方法不是引入显式的 τ 动作迁移, 而是引入从位置 ℓ 到自身的迁移, 而引入此类迁移的目的是构造删除和隐藏高安全级动作边的差异. 对于精化自动机当前位置不被允许的高安全级动作, 总能在测试自动机的对应位置找到不被允许的高安全级操作, 从而保证验证结果的差异性反映的是被允许的高安全级动作在两个自动机上的构造差异, 与无干扰属性定义一致.

本节验证算法时间复杂度主要依赖于自动机有向图节点数量. Rec_Uppaal_reachability(A_{test}, A_{ref}) 的执行复杂度为 UPPAAL 的可达性分析算法时间复杂度 O(V), 其中 V 表示图节点数量. 另一方面, 在最坏情况下, 需要以图中每个节点为起始点, 构成子图递归输入到算法中, 共需执行|V|次; 在最优情况下, 一次执行即可结束算法. 故 Rec_Uppaal_reachability 执行次数的复杂度也为 O(V). 综上, 验证算法的时间复杂度为 O(V^2) .

与现有使用 UPPAAL 对测试自动机与精化自动机的并行组合进行可达性分析的方法[27]相比, 本文验证算法将精化关系联系到无干扰安全属性, 而文献 [27] 则仅能将针对不同精化关系的测试自动机构造应用于验证可靠安全 (safety) 和活性 (liveness) 属性. 而无干扰性并非 safety 或 liveness 属性[34], 因此文献 [27] 的精化检查过程不适用于本文验证场景. 此外, 本文验证方法并未考虑对 timed GNI 属性[20]的验证. 将本文方法扩展到对更宽松的时间无干扰属性 (如 tGNDC、tNDC[24]和 timed GNI[20]) 存在挑战: tGNDC 和 tNDC 属性[24]要求系统与任意威胁系统组合后形成的系统和原系统之间满足特定等价关系. 由于威胁系统的不确定性, 使得系统与威胁系统的组合存在无限可能性, 若要使用本文验证方法, 则需要对每一个可能的组合结果进行验证, 难以在有效时间内完成验证. timed GNI 属性要求时间自动机在删除任意数量的高安全级输入动作后形成的自动机都要与原自动机之间满足时间模拟关系. 为了实现对高安全级输入动作的任意删除, 文献 [20] 的验证方法通过向自动机中高安全级输入动作所在边添加额外的迁移条件, 并在后续的验证过程中随机地满足这些迁移条件, 实现对高安全级输入动作所在边的随机激活和禁用. 而本文的验证方法并不支持上述过程, 因此难以验证 timed GNI 属性.

3.3 TINIVER 信息流安全验证演示示例

第 3.1 节已说明了 Gerking 等人[26]的验证方法在验证图 1 时间自动机的 timed BNNI 属性时存在假阳性. 本节首先说明 TINIVER 算法如何针对文献 [26] 验证方法缺陷, 正确验证图 1 自动机的 timed BNNI 属性. 将图 1 自动机的两个副本及待验证的属性 timed BNNI 输入到 TINIVER 的测试自动机与精化自动机构造方法中, 得到了和图 3(a)、图 3(b) 所示一样的测试自动机和精化自动机结果. 将这两个自动机输入本文验证算法.

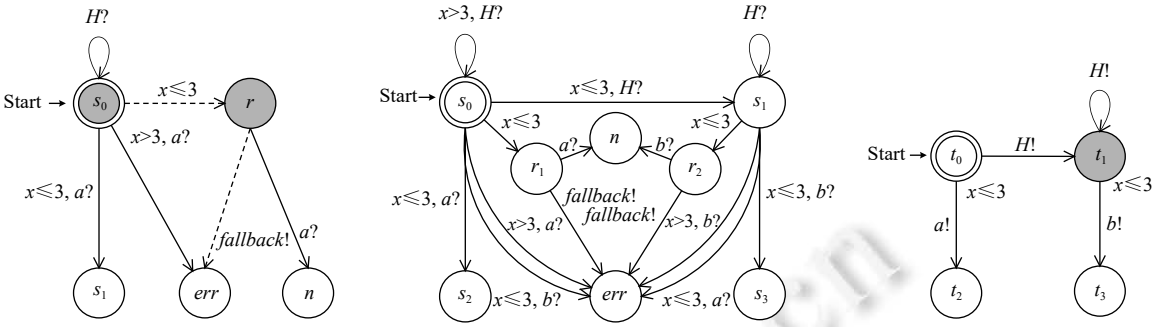
1) 得到一个 trace = … → $(\ell_{\text{test}}, \ell_{\text{ref}})$ → $(r_2, \_)$ → $(err, \_)$, 其中 $\ell_{\text{test}} = s_0$, $\ell_{\text{ref}} = t_1$. 该 trace 表明 err 位置可达, t_1 未通过 s_0 的测试, 因此将全局变量 result 与 false 析取, 由于 result 初始为 false, 此时 result 值为 false. 后续检查从 t_1 经高安全级迁移到达的未来某位置是否能够通过当前 ℓ_{test} 的测试.

2) TINIVER 算法会更改图 3(a) 测试自动机和图 3(b) 精化自动机的初始位置, 以便后续的并行组合可达性分析能跳过先前已检查的位置. 由于当前 t_1 位置未通过 s_0 的测试, 因此需判断 t_1 的后续位置是否可以通过 s_0 的测试. 精化自动机初始位置更改为 $next_t(t_1)$. 由于 t_1 通过高安全级迁移仅能到达 t_2 , 故仅得到一个初始位置为 t_2 的新精化自动机.

3) 将新的测试自动机和精化自动机迭代输入 TINIVER 验证算法, 可达性分析过程总是从测试自动机和精化自动机的初始位置开始. 以 t_2 作为初始位置的迭代中, err 位置始终不可达, 说明 t_1 的后续位置能通过测试自动机的测试, 故将 result 与 true 进行析取得到 result = true 并结束递归过程. 递归过程对全局 result 进行析取是出于对无干扰性基于的精化关系存在性的考量. 最后通过判断 result 的值为 true 判定 timed BNNI 满足.

然后, 利用图 4 自动机进一步证明 TINIVER 算法对 timed BSNNI 和 timed BNNI 的验证能力. 图 4 自动机使

用算法 1 构造测试自动机, 使用第 3.2.2 节方法构造精化自动机. 验证 timed BSNNI 和 timed BNNI 的测试自动机分别如图 5(a) 和图 5(b). 验证各属性的精化自动机如图 5(c).



(a) 用于验证 timed BSNNI 的测试自动机　(b) 用于验证 timed BNNI 的测试自动机　(c) 用于验证 timed BSNNI/BNNI 的精化自动机

图 5　TINIVER 验证图 4 自动机的 timed BSNNI 和 timed BNNI

1) TINIVER 验证出图 4 自动机不满足 timed BSNNI. ① 利用 UPPAAL 可达性分析, 得到一个 trace = … → $(\ell_{\text{test}}, \ell_{\text{ref}}) \to (r, \_) \to (err, \_)$ (其中 $\ell_{\text{test}} = s_0$, $\ell_{\text{ref}} = t_1$), 如图 5 中虚线及灰色位置所示, 该 trace 说明当前的 t_1 未通过 s_0 的测试, 存在违反无干扰性的可能. ② 检查位置 t_1 是否有可以通过高安全级动作迁移达到且能被当前 s_0 检查的不同位置. 由于不存在这种低安全级用户看来与 t_1 表现一致且可被当前 s_0 检查的位置, 因此 result 为 false 并结束函数 Rec_Uppaal_reachability(s_0 , t_0). 最终得到 result 为 false, 即不满足 timed BSNNI 属性.

2) TINIVER 验证出图 4 自动机满足 timed BNNI. 将图 5(b) 测试自动机和图 5(c) 精化自动机输入验证算法, 利用 UPPAAL 可达性分析, 发现 *err* 位置始终不可达, UPPAAL 无任何 trace 输出, 说明精化自动机通过了测试自动机的测试, 不存在违反无干扰性的情况. 根据 result 的值 true 判定图 4 自动机满足 timed BNNI 属性.

4　工具实现与评价

4.1　工具实现

TINIVER 工具使用 Java 实现. 其核心功能是从系统的时间自动机模型自动构造出测试时间自动机和精化时间自动机, 再由 TINIVER 验证算法调用 UPPAAL 的可达性分析功能验证时间自动机的信息流安全性. 为了能在实际系统的设计模型上使用 TINIVER 的验证方法, 本文还通过扩展文献 [14] 工具的模型转换模块实现了一个前端模型转换组件, 该组件从 Papyrus 建模工具构建的 SysML 顺序图模型出发, 自动将带有时间约束的 SysML 顺序图转换为行为等价的时间自动机模型. 相比文献 [14] 工具, 本文工具提取的抽象模型不再是安全相关的接口自动机, 而是安全相关的时间自动机, 且本文工具能够支持 Papyrus 4.23 和 SysML 1.6 等新的模型和规范版本. 此外, 本文还利用 LLVM 编译器的中间表示 (LLVM-IR[35]) 开发了将飞控系统 C++源码自动构造为时间自动机的前端转换模块. 该模块首先从飞控系统源码生成 LLVM-IR 表示的过程间控制流图 (interprocedural control-flow graph, ICFG); 然后, 由 ICFG 自动生成所有时间约束为空的时间自动机模板, 该时间自动机模板以 ICFG 中基本块的唯一 ID 作为动作迁移, 以 ICFG 中的控制流跳转边作为位置; 此后, 针对 ICFG 进行计时日志插桩并编译为飞控固件二进制, 使用该固件仿真飞行, 通过运行时采样的方式得到各动作迁移 (ICFG 基本块) 的有效执行时间约束并添加到前述时间自动机模板中, 形成验证用的 UPPAAL 时间自动机.

本文工具的组成和工作流程如图 6 所示. 前端模型转换组件将由 Papyrus[36]构建的 SysML 顺序图模型自动转换为 UPPAAL 可接受的时间自动机模型 XML 文本, 前端代码转换组件则从 LLVM-IR 表示的 ICFG 自动生成 UPPAAL 可接受的时间自动机模型 XML 文本. 然后, 自动机构造变换过程 (含算法 1 和第 3.2.2 节实现) 依据待验证的无干扰属性, 用时间自动机 XML 文本分别构造出测试自动机和精化自动机. 随后, 执行无干扰性验证算法 (算法 2), 调用 UPPAAL 的可达性分析 API, 分析 *err* 位置是否可达, 验证算法根据 UPPAAL 的返回结果确定验证

流程是否结束, 结束时输出无干扰性验证结果. 若 UPPAAL 的返回结果显示还需进一步迭代验证, 则调用自动机构造变换模块修改测试自动机和精化自动机的初始位置, 并对新生成的测试自动机和精化自动机执行后续验证.

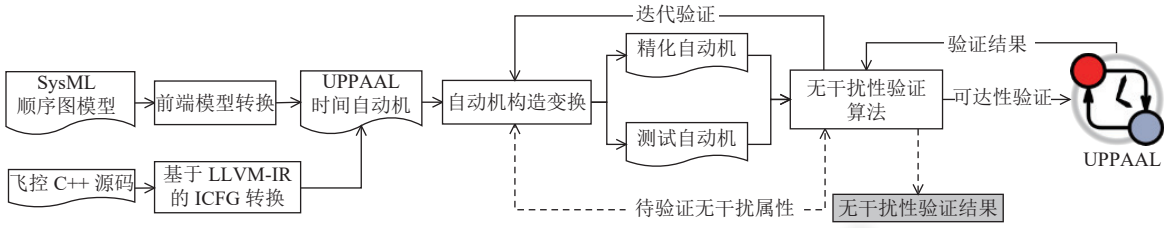


图 6 TINIVER 工具的组成和工作流程

4.2 验证有效性评价

为证明 TINIVER 工具的有效性, 使用 TINIVER 工具验证了相关工作 Lee 等人[11]、Benattar 等人[23]、Gerking 等人[20,26]和 Gardey 等人[22]中给出的自动机示例的无干扰性结论 (其中 Lee 等人[11]的示例不是时间自动机, 本文默认其示例上的所有时间约束均为小于∞). 如表 3–表 5 所示, 验证结果与人工检查确认 (Ground truth) 的无干扰性判断真实值一致. 在验证 Gerking 等人[20]中用例时, 并不验证文献 [20] 所针对的 timed GNI 属性, 而是验证该用例的 timed BSNNI, timed BNNI 和 timed SIR-NNI 属性. 本文还验证了 Kocher[32]发现的 RSAREF 2.0 源码中的漏洞, 证明了 RSAREF 2.0 源码 (漏洞相关源码的规模约 160 行) 关于模指数运算的实现不满足 timed SIR-NNI 属性. 此外, 还验证了本文示例 (fig4.xml 和 fig1.xml 分别对应于本文图 4 和图 1 所示的时间自动机) 的信息流安全结果. 这些模型的真实信息流安全性结果与 TINIVER 工具输出的一致. 对每个测试用例, 我们还度量 5 次验证的平均时间开销.

表 3 timed BSNNI 验证结果及验证时间

用例来源	时间自动机模型	Ground truth	TINIVER验证结果	5次验证平均时间 (ms)
Lee等人[11]	case1.xml	×	×	82.8
	case2.xml	√	√	82.0
Benattar等人[23]	BSNNI_3.xml	×	×	75.8
	BSNNI_4.xml	√	√	64.6
Gardey等人[22]	BSNNI_1.xml	×	×	85.0
	BSNNI_2.xml	√	√	69.4
Gerking等人[20]	application.xml	×	×	575.4
Kocher[32]	RSA.xml	√	√	86.8
本文	fig4.xml	×	×	75.6
	fig1.xml	√	√	73.2

注: √: 属性满足; ×: 属性不满足

表 4 timed BNNI 验证结果及验证时间

用例来源	时间自动机模型	Ground truth	Gerking等人方法[26]验证结果	TINIVER验证结果	5次验证平均时间 (ms)
Lee等人[11]	case1.xml	×	×	×	71.0
	case2.xml	√	×	√	81.0
Gerking等人[26]	BNNI_1.xml	√	√	√	72.8
	BNNI_2.xml	×	×	×	84.2
Gerking等人[20]	application.xml	×	×	×	605.0
Kocher[32]	RSA.xml	√	√	√	84.4
本文	fig4.xml	√	√	√	65.4
	fig1.xml	√	×	√	79.0

注: √: 属性满足; ×: 属性不满足

表 5 timed SIR-NNI 验证结果及验证时间

用例来源	时间自动机模型	Ground truth	TINIVER验证结果	5次验证平均时间 (ms)
Lee等人[11]	case1.xml	√	√	72.0
	case2.xml	×	×	75.0
	SIRNNI_1.xml	√	√	66.0
Gerking等人[20]	application.xml	×	×	490.8
Kocher[32]	RSA.xml	×	×	79.0
本文	fig4.xml	√	√	66.4
	fig1.xml	×	×	141.6

注: √: 属性满足; ×: 属性不满足

4.3 无人系统飞控模式切换模块的时间无干扰性分析

本节使用 TINIVER 工具分析 ArduPilot 4.2 飞控系统和 PX4 1.13.2 飞控系统上的飞行模式切换功能模块的时间无干扰属性. 由于不同飞行模式所关联的传感器不同, 使得不同飞行模式的完整性存在差异. 分析不同飞行模式之间切换的信息流安全性, 即要分析在有时间约束的情况下, 从低完整性飞行模式切换至高完整性飞行模式的过程中, 时间无干扰性是否得到满足. 易见, 与现有基于定理证明的飞控可靠性证明[37]相比, 本文从验证方法和目标属性上均完全不同.

(1) ArduPilot 4.2 飞控系统上基于 SysML 模型的时间无干扰性分析

ArduPilot 飞控系统基于的编译工具 waf 不支持生成 LLVM 中间表示, 因此 ArduPilot 不适用于 TINIVER 的飞控 C++源码转换前端, 故本文对 ArduPilot 的特定模块进行 SysML 建模, 借助 TINIVER 的 SysML 顺序图转换前端生成待验证的时间自动机.

首先, 通过分析 ArduPilot 飞控系统中飞行模式切换模块的源码 (规模约为 1 180行), 对 ArduPilot 飞行模式切换进行逆向建模, 将源码中用于控制模式切换和执行具体模式的对象作为顺序图生命线, 将从源码中抽象出的模式切换所涉及的具体动作作为消息, 得出一组包含时间约束的 SysML 顺序图模型. 静态分析源码无法得出约束每个具体动作的时间约束的上限阈值, 为了获得这些具体阈值以形成验证可用的具体时间约束, 进行运行时的动作时间开销采样. 具体地, 向 ArduPilot 源码中插入记录模式切换相关动作执行时间的日志代码, 在真实四旋翼无人机上多次飞行并执行多次飞行模式切换动作, 获取每个模式切换相关动作的执行时间集合 (飞行次数和飞行模式切换次数保证对于每个具体动作的执行时间集合中有不少于 7 个执行时间值), 从中选择每个动作的最大执行时间近似代表该动作的时间约束上限阈值. 这一过程还需考虑一个动作与其子动作运行时间的包含关系, 确保日志记录的该动作执行时间上限阈值大于等于其所有子动作的执行时间上限阈值之和. 经过以上过程, SysML 模型的时间约束阈值均为具体值.

然后, 为飞行模式切换模块的 SysML 顺序图的每个动作划分安全级. 为来自传感器以及遥控器的消息分配低完整性级别, 为模式切换的内部动作消息分配高完整性级别. 将对外部低完整性数据依赖较少的飞行模式定义为具有较高完整性, 据此得出以下的飞行模式间潜在完整性关系.

1) 自稳模式≥定高模式. 自稳模式的执行过程仅读取来自遥控器的滚动输入 (roll)、俯仰输入 (pitch) 和偏航输入 (yaw) 以及用于维持无人机稳定的基本传感器数据. 定高模式的执行不仅要读取来自遥控器的控制信号和用于维持无人机稳定的基本传感器数据, 还需要从惯导传感器获取数据来更新飞控的当前高度信息. 因此, 自稳模式相对于定高模式具有更高完整性.

2) 定高模式≥返航模式. 当从定高模式切换至返航模式时, 相比于定高模式所需的传感器数据, 返航模式还额外需要来自姿态航向参考系统以及 GPS 传感器的信息, 因此, 定高模式相对于返航模式具有更高完整性.

如果能通过时间无干扰性验证, 确定从较低完整性飞行模式切换到较高完整性飞行模式的过程中, 不会产生低完整性模式独有的传感器输入对高完整性模式的内部动作的干扰, 则可证明较低完整性飞行模式在这一模式切换过程中, 实际上具有与较高完整性飞行模式相同的完整性, 亦即证明了模式切换的安全性. 据此, 对从返航模式切换至定高模式, 以及从定高模式切换至自稳模式的时间约束 SysML 顺序图, 进行消息安全级标注. 具体动作的安全级划

分以及时间约束上限阈值如表 6 和表 7 所示. 其中, 定高模式的动作 get_position_z_up_cm, get_velocity_z_up_cms 和 get_z_accel_cmss 相对于自稳模式而言是低完整性的额外外部输入, 因为定高模式相较于自稳模式, 除了读取用于控制姿态的传感器数据外, 还需调用这 3 个动作读取用于垂直方向位置控制的传感器数据. 返航模式的动作 get_position_xy_cm 和 get_velocity_z_up_cms 相较于定高模式来说是低完整性的额外外部输入, 因为返航模式相较于定高模式, 除了需要读取垂直方向位置控制的传感器数据外, 还需读取用于水平方向位置控制的传感器数据.

表 6 ArduPilot 定高切换至自稳模式所用动作的安全级划分及时间约束

消息名称	安全级划分	时间约束阈值 (μs)
set_max_speed_accel_z	H	215
get_pilot_desired_lean_angles	H	203
get_pilot_desired_yaw_rate	H	114
get_pilot_desired_climb_rate	H	215
rc_loop	H	749
set_mode	H	487
AltHold_Flying	H	262
input_to_euler	H	312
get_position_z_up_cm	L	222
get_velocity_z_up_cms	L	225
get_z_accel_cmss	L	187
set_throttle_out	H	139

注: H: 高完整性; L: 低完整性

表 7 ArduPilot 返航切换至定高模式所用动作的安全级划分及时间约束

消息名称	安全级划分	时间约束阈值 (μs)
get_stopping_point	H	47
compute_return_target	H	101
get_pilot_desired_yaw_rate	H	237
set_desired_spool_state	H	200
update_wpnav	H	407
get_position_z_up_cm	H	246
get_velocity_z_up_cms	H	238
get_z_accel_cmss	H	265
set_throttle_out	H	290
set_yaw	H	315
reached_wp_destination	H	175
get_position_xy_cm	L	16
get_velocity_xy_cms	L	33
rc_loop	H	749
set_mode	H	487
AltHold_Flying	H	262
input_to_euler	H	312
set_throttle_out	H	139

注: H: 高完整性; L: 低完整性

在完成了上述分析后, 得到定高模式切换至自稳模式, 以及返航模式切换至定高模式的 SysML 顺序图模型. 将这两个模型分别输入 TINIVER 的模型转换组件, 分别得到上述两种模式切换过程的 UPPAAL 时间自动机. 两个 UPPAAL 时间自动机的位置集合分别包含 17 个和 25 个位置. 利用 TINIVER 工具验证以上两种飞行模式的切换过程不存在低完整性模式独有的传感器输入对高完整性模式的内部动作的干扰, 从而证明以上两种飞行模式切换过程的信息流安全性. 具体验证结果如表 8 所示. 表 8 中 Ground truth 为人工对 ArduPilot 4.2 的飞行模式切换源码的分析结果, 可见, TINIVER 成功验证出了定高模式切换至自稳模式以及返航模式切换至定高模式的过程满足 timed BNNI、timed BSNNI 和 timed SIR-NNI. 因此, 上述两种飞行模式切换过程不存在低完整性模式独有的传感器输入对高完整性模式的内部动作的干扰, 是信息流安全的.

表 8 ArduPilot 的两种飞行模式切换过程的安全性验证结果及验证时间开销

时间无干扰性	飞行模式切换实例	Ground truth	Gerking方法[26]验证结果	TINIVER验证结果	5次验证平均时间 (ms)
timed BSNNI	定高切换自稳	√	N/A	√	124.2
	返航切换定高	√	N/A	√	184.4
timed BNNI	定高切换自稳	√	×	√	122.6
	返航切换定高	√	×	√	185.4
timed SIR-NNI	定高切换自稳	√	N/A	√	96.6
	返航切换定高	√	N/A	√	158.4

注: √: 模式切换安全; ×: 模式切换不安全

需要注意的是, Gerking 等人[26]的验证方法对这两个模式切换过程亦存在误报. 原因是在定高切换至自稳模式的时间自动机中, 当测试自动机上可执行的低完整性动作 get_position_z_up_cm 被限制导致精化自动机的高完

整性动作 input_to_euler 无法正常执行时, Gerking 等人方法仅简单地判定不满足 timed BNNI 属性, 而不会将通过包含 input_to_euler 在内的高完整性动作序列可到达的后续位置送入迭代过程继续验证. 在返航切换至定高模式的时间自动机中, 当测试自动机上可执行的低完整性动作 get_position_xy_cm 被限制, 同样会导致精化自动机的高完整性动作 AltHold_Flying 无法执行, 同理, Gerking 等人方法依然不会继续选择通过包含 AltHold_Flying 在内的高完整性动作序列到达的后续位置进行迭代验证.

此外还需注意, 本文定义的飞行模式间切换的安全性不可传递, 即从返航切换至定高模式安全和从定高切换至自稳模式安全, 无法直接推出返航切换至自稳模式安全, 原因是定高模式在两种切换过程中, 其部分传感器输入仅在一种切换的安全性判定中作为低完整性的额外外部输入, 而在另一种切换的安全性判定中作为无关量. 如需验证返航切换至自稳模式的安全性, 应直接为这两种模式间切换建立 SysML 模型并实测时间约束阈值. 不失一般性地, TINIVER 还可用于验证飞行控制系统的其他飞行模式间切换是否满足时间无干扰属性. 此外, 虽然对 ArduPilot 的验证要求对飞控系统源码进行逆向 SysML 建模, 但 TINIVER 本身的功能并不局限于时间约束的 SysML 模型, 可以支持用 LLVM-IR 表示的飞控系统 C++源码.

本文在用 SysML 对飞控系统的飞行模式切换功能模块代码进行建模的过程中, 省略了内部函数中与模式切换无关的对其他飞行指令处理函数的调用. 当在定高模式切换至自稳模式以及返航模式切换至定高模式的 SysML 模型中保留这些信息流无关的消息动作迁移时, 两个时间自动机的位置集合分别扩大为 53 个和 72 个位置. 由于这些新引入的消息动作迁移与信息流无关, 因而无需对它们分配安全级. 重新测定这些动作迁移上的时间约束后进行时间无干扰性的重新验证, TINIVER 对两个时间自动机进行 timed BNNI、timed BSNNI 和 timed SIR-NNI 属性验证的结果与表 8 结果一致, 但平均验证时间增长到了 147.6–247.4 ms. 易见, 保留未分配安全级的信息流无关动作对验证时间开销的影响有限, 原因是在飞行模式切换功能模块中, 未分配安全级的动作迁移的关系较简单, TINIVER 构造的测试和精化自动机的复杂度以及验证的复杂度均主要取决于高 (低) 安全级动作迁移的数量.

(2) PX4 1.13.2 飞控系统代码的时间无干扰性分析

PX4 飞控系统的编译链适用于基于 LLVM 的分析, 因此使用 TINIVER 的 C++代码转换前端直接自动生成时间自动机. 由于单个基本块在仿真环境下执行时间过短难以有效记录, 因此使用记录函数执行时间上限阈值、并按控制流路径长度和基本块中指令数将函数的执行时间上限阈值分配到各基本块的近似方式获得单个基本块的执行时间约束, 构造时间自动机模型.

进一步获得 PX4 飞控系统的典型飞行模式间的潜在完整性关系. 与 ArduPilot 类似地, PX4 的自稳和定高模式之间的完整性关系为: 自稳模式≥定高模式. 但与 ArduPilot 不同的是, PX4 的定高模式和返航模式的完整性关系不可比. 其原因是 PX4 的定高模式相较于返航模式要求来自遥控器的输入, 而返航模式相较于定高模式要求来自 GPS 等位置传感器的数据, 二者不存在传感器输入数据的完整性偏序关系. 因此, 本文仅验证 PX4 从定高模式切换到自稳模式的过程是否违反时间无干扰性.

本文利用保存在 LLVM-IR 中的调试信息获取基本块与源码的对应关系. 通过分析源码, 结合上述对应关系, 将包含获取无人机当前高度数据的基本块作为定高模式相较于自稳模式额外的低完整性输入, 其他基本块作为内部动作分配高完整性级别. 对于 PX4 飞控系统 FlightModeManager 模块下的 run 函数、start_flight_task 函数、switchTask 函数、handleCommand 函数以及 generateTrajectorySetpoint 函数, 将它们的代码 (共计约 420 行) 共同传入 TINIVER 的飞控 C++代码转换前端, 得到定高模式切换至自稳模式的 UPPAAL 时间自动机模型. 该模型包含 179 个位置和 259 个动作迁移. 利用 TINIVER 工具对该时间自动机模型进行验证, 分析 PX4 从定高模式切换到自稳模式的过程中是否存在定高模式独有的无人机当前高度输入对自稳模式的内部动作的干扰. 具体验证结果如表 9 所示. 说明 PX4 从定高模式切换到自稳模式的过程是信息流安全的. 表 9 中, Ground truth 为对 PX4 1.13.2 飞行模式切换源码进行人工分析的结果. 可见, TINIVER 的验证结果与 Ground truth 一致. 表 9 所示验证时间开销相比在 ArduPilot 上基于 SysML 模型的验证时间开销, 增长到约 99–105 s. 这一显著增长不仅是由于代码自动生成的自动机的位置和迁移数量相较由 SysML 模型生成的自动机而言均有所增长, 更重要的是, 代码级验证将信息

流安全相关的函数分解到基本块时, 分解出的基本块都需要分配安全级, 使得实际分配高 (低) 安全级的动作迁移的数量较多, 增大了 TINIVER 构造测试和精化自动机的复杂度以及验证的复杂度.

表 9 PX4 从定高模式切换至自稳模式的安全性验证结果及验证时间开销

时间无干扰性	Ground truth	Gerking等人方法验证结果	TINIVER验证结果	5次验证平均时间 (ms)
timed BSNNI	√	N/A	√	103 067.4
timed BNNI	√	√	√	104 808.2
timed SIR-NNI	√	N/A	√	99 220.0

注: √: 模式切换安全; ×: 模式切换不安全

5 总 结

本文提出了 TINIVER, 一种兼容多种时间无干扰属性验证的软件信息流安全验证方法和工具. 方法能够统一验证本文提出的 timed SIR-NNI 属性以及现有的 timed BNNI, timed BSNNI 属性. 在本文实现的前端模型转换工具的辅助下, 能够验证 SysML 顺序图模型的时间无干扰属性. 实验说明本文方法避免了现有典型验证方法的假阳性缺陷并能应用于无人机飞控系统典型组件的信息流安全性验证, 相比现有验证方法具有更强的适用性和可扩展性. 未来将进一步把现有基于 LLVM-IR 的飞控系统 C++代码验证从当前针对个别飞行模式切换过程推广应用到整个飞控系统. 未来还考虑将更多时间无干扰属性 (如 tGNDC、tNDC、timed GNI) 的验证方法纳入 TINIVER 的验证框架, 并拓展 TINIVER 前端模型转换能力, 以支持 SysML 顺序图的完整语义, 包括对 Par 组合片段的处理.

TINIVER 的实现已开源: https://github.com/suncongxd/TINIVER.

References:

[1] Goguen JA, Meseguer J. Security policies and security models. In: Proc. of the 1982 IEEE Symp. on Security and Privacy. Oakland: IEEE, 1982. 11–20. [doi: 10.1109/SP.1982.10014]

[2] Focardi R, Gorrieri R. The compositional security checker: A tool for the verification of information flow security properties. IEEE Trans. on Software Engineering, 1997, 23(9): 550–571. [doi: 10.1109/32.629493]

[3] van der Meyden R, Zhang CY. Algorithmic verification of noninterference properties. Electronic Notes in Theoretical Computer Science, 2007, 168: 61–75. [doi: 10.1016/j.entcs.2006.11.002]

[4] Sabelfeld A, Myers AC. Language-based information-flow security. IEEE Journal on Selected Areas in Communications, 2003, 21(1): 5–19. [doi: 10.1109/JSAC.2002.806121]

[5] Cassez F, van der Meyden R, Zhang CY. The complexity of synchronous notions of information flow security. In: Proc. of the 13th Int'l Conf. on Foundations of Software Science and Computational Structures. Paphos: Springer, 2010. 282–296. [doi: 10.1007/978-3-642-12032-9_20]

[6] Zhang F, Xu MD, Zhao HC, Zhang C, Liu XL, Hu FN. Real-time trust measurement of software: Behavior trust analysis approach based on noninterference. Ruan Jian Xue Bao/Journal of Software, 2019, 30(8): 2268−2286 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5768.htm [doi: 10.13328/j.cnki.jos.005768]

[7] Clarkson MR, Finkbeiner B, Koleini M, Micinski KK, Rabe MN, Sánchez C. Temporal logics for hyperproperties. In: Proc. of the 3rd Int'l Conf. on Principles of Security and Trust. Grenoble: Springer, 2014. 265–284. [doi: 10.1007/978-3-642-54792-8_15]

[8] Bonakdarpour B, Prabhakar P, Sánchez C. Model checking timed hyperproperties in discrete-time systems. In: Proc. of the 12th NASA Formal Methods Symp. Moffett Field: Springer, 2020. 311–328. [doi: 10.1007/978-3-030-55754-6_18]

[9] Ho HM, Zhou RY, Jones TM. Timed hyperproperties. Information and Computation, 2021, 280: 104639. [doi: 10.1016/j.ic.2020.104639]

[10] Focardi R, Gorrieri R. Classification of security properties (Part I: Information flow). In: Proc. of the 2000 Int'l School on Foundations of Security Analysis and Design. Berlin: Springer, 2000. 331–396. [doi: 10.1007/3-540-45608-2_6]

[11] Lee M, D'Argenio PR. A refinement based notion of non-interference for interface automata: Compositionality, decidability and synthesis. In: Proc. of the 2010 XXIX Int'l Conf. of the Chilean Computer Science Society. Antofagasta: IEEE, 2010. 280–289. [doi: 10.1109/SCCC.2010.14]

[12] Li Q, Zeng QK, Yuan ZX. Logic of multi-threaded programs for non-interference. Ruan Jian Xue Bao/Journal of Software, 2014, 25(6): 1143−1153 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4429.htm [doi: 10.13328/j.cnki.jos.004429]

[13] Sun C, Xi N, Li JK, Yao QS, Ma JF. Verifying secure interface composition for component-based system designs. In: Proc. of the 21st Asia-Pacific Software Engineering Conf. Jeju: IEEE, 2014. 359–366. [doi: 10.1109/APSEC.2014.60]

[14] Sun C. SysML2ia. 2020. https://bitbucket.org/suncong_xdu/sysml2ia/src/master/

[15] Sun C, Xi N, Ma JF. Enforcing generalized refinement-based noninterference for secure interface Composition. In: Proc. of the 41st IEEE Annual Computer Software and Applications Conf. Turin: IEEE, 2017. 586–595. [doi: 10.1109/COMPSAC.2017.118]

[16] Sun C. CertIA: A coq library for certifying interface automata. 2020. https://github.com/suncongxd/certia

[17] Xie J, Huang H. A noninterference model for nondeterministic systems. Ruan Jian Xue Bao/Journal of Software, 2006, 17(7): 1601−1608 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1601.htm [doi: 10.1360/jos171601]

[18] Jiang K, Zhang TW, Sanán D, Zhao YW, Liu Y. A formal methodology for verifying side-channel vulnerabilities in cache architectures. In: Proc. of the 23rd Int'l Conf. on Formal Engineering Methods, Formal Methods and Software Engineering (ICFEM 2022). Madrid: Springer, 2022. 190–208. [doi: 10.1007/978-3-031-17244-1_12]

[19] Gray III JW. Toward a mathematical foundation for information flow security. Journal of Computer Security, 1992, 1(3–4): 255–294. [doi: 10.3233/JCS-1992-13-405]

[20] Gerking C, Schubert D. Component-based refinement and verification of information-flow security policies for cyber-physical microservice architectures. In: Proc. of the 2019 IEEE Int'l Conf. on Software Architecture. Hamburg: IEEE, 2019. 61–70. [doi: 10.1109/ICSA.2019.00015]

[21] Focardi R, Gorrieri R, Martinelli F. Real-time information flow analysis. IEEE Journal on Selected Areas in Communications, 2003, 21(1): 20–35. [doi: 10.1109/JSAC.2002.806122]

[22] Gardey G, Mullins J, Roux OH. Non-interference control synthesis for security timed automata. Electronic Notes in Theoretical Computer Science, 2007, 180(1): 35–53. [doi: 10.1016/j.entcs.2005.05.046]

[23] Benattar G, Cassez F, Lime D, Roux OH. Control and synthesis of non-interferent timed systems. Int'l Journal of Control, 2015, 88(2): 217–236. [doi: 10.1080/00207179.2014.944356]

[24] Gorrieri R, Locatelli E, Martinelli F. A simple language for real-time cryptographic protocol analysis. In: Proc. of the 12th European Symp. on Programming. Warsaw: Springer, 2003. 114–128. [doi: 10.1007/3-540-36575-3_9]

[25] Vasilikos P, Nielson F, Nielson HR. Secure information release in timed automata. In: Proc. of the 7th Int'l Conf. on Principles of Security and Trust. Thessaloniki: Springer, 2018. 28–52. [doi: 10.1007/978-3-319-89722-6_2]

[26] Gerking C, Schubert D, Bodden E. Model checking the information flow security of real-time systems. In: Proc. of the 10th Int'l Symp. on Engineering Secure Software and Systems. Paris: Springer, 2018. 27–43. [doi: 10.1007/978-3-319-94496-8_3]

[27] Heinzemann C, Brenner C, Dziwok S, Schäfer W. Automata-based refinement checking for real-time systems. Computer Science-research and Development, 2015, 30(3): 255–283. [doi: 10.1007/s00450-014-0257-9]

[28] UPPAAL. 2022. https://uppaal.org/

[29] Alur R, Dill DL. A theory of timed automata. Theoretical Computer Science, 1994, 126(2): 183–235. [doi: 10.1016/0304-3975(94)90010-8]

[30] Clarke EM, Henzinger TA, Veith H, Bloem R. Handbook of Model Checking. Berlin: Springer, 2018. 1001–1046. [doi: 10.1007/978-3-319-10575-8]

[31] Alur R, Courcoubetis C, Henzinger TA. The observational power of clocks. In: Proc. of the 5th Int'l Conf. on Concurrency Theory. Uppsala: Springer, 1994. 162–177. [doi: 10.1007/978-3-540-48654-1_16]

[32] Kocher PC. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Proc. of the 16th Annual Int'l Cryptology Conf. Santa Barbara: Springer, 1996. 104–113. [doi: 10.1007/3-540-68697-5_9]

[33] RSA Laboratories. RSAREF. 2016. https://sourceforge.net/projects/rsaref/

[34] McLean J. A general theory of composition for trace sets closed under selective interleaving functions. In: Proc. of the 1994 IEEE Computer Society Symp. on Research in Security and Privacy. Oakland: IEEE, 1994. 79–93. [doi: 10.1109/RISP.1994.296590]

[35] Lattner C, Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. In: Proc. of the 2008 Int'l Symp. on Code Generation and Optimization. Piscataway: IEEE, 2008. 1–20. [doi: 10.1109/CGO.2004.1281665]

[36] Eclipse papyrus™ modeling environment. 2023. https://www.eclipse.org/papyrus/

[37] Shi ZP, Cui M, Xie GJ, Chen G. Coq formalization of propulsion subsystem of flight control system for multicopter. Ruan Jian Xue Bao/Journal of Software, 2022, 33(6): 2150–2171 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6575.htm [doi: 10.13328/j.cnki.jos.006575]

附中文参考文献:

[6] 张帆, 徐明迪, 赵涵捷, 张聪, 刘小丽, 胡方宁. 软件实时可信度量: 一种无干扰行为可信性分析方法. 软件学报, 2019, 30(8): 2268−2286. http://www.jos.org.cn/1000-9825/5768.htm [doi: 10.13328/j.cnki.jos.005768]

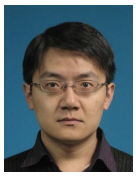
[12] 李沁, 曾庆凯, 袁志祥. 一种面向非干扰的线程程序逻辑. 软件学报, 2014, 25(6): 1143−1153. http://www.jos.org.cn/1000-9825/4429.htm [doi: 10.13328/j.cnki.jos.004429]

[17] 谢钧, 黄皓. 一个非确定系统的不干扰模型. 软件学报, 2006, 17(7): 1601−1608. http://www.jos.org.cn/jos/article/abstract/20060714?st=search [doi: 10.1360/jos171601]

[37] 石正璞, 崔敏, 谢果君, 陈钢. 多旋翼飞控推进子系统的Coq形式化验证. 软件学报, 2022, 33(6): 2150−2171. http://www.jos.org.cn/1000-9825/6575.htm [doi: 10.13328/j.cnki.jos.006575]

刘乔森(1998−), 男, 博士生, 主要研究领域为无人系统安全, 信息流安全.

曾荟铭(1997−), 男, 工程师, 主要研究领域为无人系统安全.



孙聪(1982−), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为软件安全, 程序分析, 无人系统安全.

马建峰(1963−), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为信息安全, 密码学, 网络安全.

魏晓敏(1990−), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为系统安全, 可靠安全, 无人机, 嵌入式软件.