基于观察者模式的实时系统验证方法

赵 鹤 洪 玫 杨秋辉 高婉玲
(四川大学计算机学院 成都 610065)

摘 要 复杂实时系统的验证问题一直备受关注。验证过程中,验证特性可以用时序逻辑来描述,但时序逻辑对于非专业人员而言较为复杂,难度较大。观察者模式是一个额外的子系统,可以将复杂的验证特性转换为简单的可达性问题,同时也可以避免使用复杂的验证算法。将Etienne和Nouha Abid等人提出的抽象的观察者模式应用到实时系统实例——Train-Gate系统中,采用UPPAAL工具对Train-Gate系统中的某些场景建立观察者模型,并采用对比实验将验证结果与无观察者模式状态下的验证结果进行对比。对比结果表明,使用观察者模式和验证特性都可以得到正确的验证结果,但观察者更节省时间,对于非专业人员而言更简单且更容易接受。因此,使用观察者模式对如Train-Gate的实时系统进行验证是可行的。

关键词 观察者模式,实时系统,UPPAAL,Train-Gate,模型检测
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.12.030

Real-time System Verification Approach Based on Observer Patterns

ZHAO He HONG Mei YANG Qiu-hui GAO Wan-ling
(College of Computer Science, Sichuan University, Chengdu 610065, China)

Abstract The verification of complex real-time system is always high-profile. A common way to describe the verification properties is using temporal logic, and the way is complex and difficult for laypeople. Observer patterns is an additional subsystem. It can transform the complex verification properties into simple reachability problem. The use of observer patterns can avoid using complex verification algorithm. The observer patterns proposed by Etienne and Nouha Abid were applied to the real-time system, Train-Gate system. UPPAAL was used to construct the observer models according to some scenarios in Train-Gate. Comparative experiment was used to compare the verification result with and without observer patterns. The experiment result shows that the use of observer patterns and verification properties both can get correct results. And the use of observer patterns can save time and it's easier to accept for laypeople. Therefore, the use of observer patterns is feasible in the real-time systems like Train-Gate verification.

Keywords Observer patterns, Real-time system, UPPAAL, Train-Gate, Model checking

1 引言

复杂实时系统正确性的验证问题一直备受关注,针对此问题有许多形式化的方法被提出,但是这些方法受限于复杂的公式,在某种程度上难以广泛应用。时序逻辑可用于形式化验证,描述系统验证过程中的验证特性,但对于非专业人员而言,其实现过程较为复杂,且需要使用复杂的验证算法来进行验证。观察者模式是一个额外的子系统,具有较强的表现能力,通过观察者可以将复杂的验证特性转化为简单的可达性问题,进而观察系统中的某些行为是否发生。通过观察者可以降低编写验证特性的复杂度,避免使用复杂的验证算法,从而缓解复杂实时系统中的验证复杂性问题。

目前,许多形式体系被用来构建复杂的实时系统,特别是时间自动机[1-2]、CSP的定时扩展[3-4]、Z[5]及Petri网的不同的定时扩展[6]。近期的一些工作[7-10]包含可以用来构建组合或分层系统的形式体系。在文献[8]中,时间自动机模式可以被用来构建常用的实时系统行为,如截止时间、超时以及时间中断。这些模式是组合的,并且以分层的方式来辅助构建一个系统。本文采用观察者的目的并不是构建系统模型,而是表现系统的正确性。此外,本文所采用的模式不是组合的。

文献[11]定义和使用了针对动态系统的逻辑-代数规约语言——CASL-LTL。在文献[12]中,它的表现力很强,其类似自然语言的语法使得它可以被非专业人员所接受。CASL-LTL虽然可以表现时序行为,但是无法表示定时的行为。

一些研究已提出了将某些特性转换为可达性检测的想法如在文献[13]中,安全性和有限的活性属性被转化为测试自

到稿日期:2016-10-17 返修日期:2016-12-31 本文受四川省应用基础研究项目:嵌入式系统软件形式化验证技术研究(2014JY0112)资助。
赵 鹤(1992-),女,硕士,CCF会员,主要研究方向为软件质量保障与测试,E-mail:1085593477@qq.com;洪 玫(1963-),女,教授,主要研究方向为软件质量保障与测试;杨秋辉(1970-),女,博士,副教授,主要研究方向为软件测试、软件工程;高婉玲(1992-),女,硕士,主要研究方向为软件质量保障与测试。

动机,这与本文所采用的观察者的概念等价;差异在于本文表现的是通常情况下所使用的模式,而文献[13]的目的在于完整性(这样的可达性检测的表现形式在文献[14]中有所体现),且本文仅考虑可达性的属性,不考虑非可达性的属性。文献[15]将典型的时序约束用于调度问题的建模,然后将其转化为时间自动机。文献[16]用 UML 状态图的形式来定义用于指定系统正确性的模式,然后将其转换为时间自动机。在观察者中,则是将正确性转化为可达性检测问题。

文献[17]提出了一组实时的规约模式,用于验证具有强实施约束的反应系统;设计了一个模式来表示实时系统分析过程中发现的一般的定时约束;同时,给出了一个可以被用来检测时间需求的集成的模型检测工具链。文献[17]中的模式可以被视为是 Dwyer[18] 的规约模式的一个实时扩展。Dwyer 通过一个 500 个规约案例的研究表明,80%的时序需求可以通过少数的模式公式被覆盖。文献[19]对 Dwyer 等人的工作进行了扩展,加入了实时概念,对观察者重新分类并采用 MTL,TGIL 和 FOTT 3 种方式进行定义和描述。

时序逻辑的使用虽然较为广泛,但是时序逻辑比观察者更加昂贵,且对于非专业人员而言更难操控,许多工具只能表现时序逻辑的一部分,无法支持完整的表达。观察者模式中的"模式"与软件工程中的设计模式有相似之处,关注于描绘一般的正确性属性,且不是组合的。软件工程中的设计模式可以被轻松地插入到写好的代码中,而观察者则是独立的,且具有定义定量的时间行为的能力,观察者还可以将特性验证转化为可达性检测,因此本文采用观察者模式来对实时系统进行验证。

本文将 Etienne[3] 和 Nouha Abid 等人[20] 提出的观察者模式应用到实时系统的验证中,采用模型检测工具 UPPAAL 对 Train-Gate 实例中几种场景下的观察者模式进行建模,最后通过实例验证的方法来分析观察者的可行性及应用效果。结果表明,观察者可以有效缓解验证过程中的复杂性问题,将复杂的验证特性转化为可达性问题,从而减少编写复杂验证特性的工作量,也可以避免复杂验证算法的使用,即将观察者模式应用到如 Train-Gate 的实时系统的验证过程中是可行且有效的。

2　研究背景

本节对研究过程中所使用到的基础知识(包括时间自动机、TCTL、观察者模式和 UPPAAL 工具的形式化定义及相关内容)进行介绍。

2.1　时间自动机(Timed Automata)

时间自动机 A 是一个元组 $(L, l_0, E, Label, C, clocks, guard, inv)$,其中:

1) L,一个非空的有限的状态集合,初始状态 $l_0 \in L$;

2) $E \subseteq L \times L$,边的一组集合;

3) $Label: L \to 2^{AP}$,一个函数,为每个状态 $l \in L$ 分配一组原子命题 $Label(l)$;

4) C,时钟的一个有限集合;

5) $clocks: E \to 2^C$,一个函数,为每一条边 $e \in E$ 分配一组时钟 $clocks(e)$;

6) $guard: E \to \Psi(C)$,一个函数,为每一条边 $e \in E$ 标记一个 C 内的时钟约束 $guard(e)$;

7) $inv: L \to \Psi(C)$,一个函数,为每个状态分配一个变量。

2.2　TCTL(Timed Computation Tree Logic)

TCTL 是定时的计算树逻辑。计算树逻辑 CTL 是一种分支时间逻辑,即它的时间模型是一个树状结构。CTL 公式,Backus-Naur 范式的定义如下[21]:

$$\Phi: \bot \mid T \mid p \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \to \Phi \mid AX\Phi \mid EX\Phi \mid AF\Phi \mid EF\Phi \mid AG\Phi \mid EG\Phi \mid A[\Phi U \Phi] \mid E[\Phi U \Phi]$$

其中,p 为去边原子公式的集合,$\bot$ 表示永假式,T 表示永真式;A 和 E 为路径量词,A 的含义是"沿所有路径",E 的含义是"沿至少一条路径";X, F, G, U 为时态算子,X 表示"下一个状态",F 表示"某个未来状态",G 表示"所有未来状态",U 表示"直到"。在 CTL 中,每个 A 或 E 必须伴随着 X, F, G 或 U 之一出现[21]。

2.3　观察者模式(Observer Patterns)

观察者是标准的子系统,对系统不能有任何影响,不能阻止任何系统行为的发生。在时间自动机的形式下,观察者是一个带有某些约束的标准时间自动机。一个全局模型被定义为 $A_1 \parallel \cdots \parallel A_n \parallel A_{obs}$,其中 $A_1, \cdots, A_n$ 是时间自动机构建的原始模型,A_{obs} 是观察者自动机。并行组合确保了行为可以被观察者和原始模型同步共享。

2.4　模型检测工具 UPPAAL

UPPAAL[22] 是一个用于实时系统建模、模拟和验证的工具,由 Uppsala 大学和 Aalborg 大学共同开发。在 UPPAAL 中,一个系统被表示为由一组时间自动机、全局数据、变量及 CCS(Calculus of Communicating Systems)风格的同步管道组成的模型。通过定义信号 c,时间自动机之间可以完成同步,其中,发布消息使用 $c!$,接收消息使用 $c?$。UPPAAL 支持 C 语言风格的语法,能够实现自定义的函数[23]。

本文所涉及的 UPPAAL 模型包含状态(location)、状态变量(invariant)、边(edge)、选择(select)语句、条件语句(guard)、同步(Synchronisation)语句、更新(Update)语句。Guard 语句用来设定前置条件,同步语句用来同步原始模型和观察者模型,更新语句用来更新某些变量的值。UPPAAL 模型的图例表示如图 1 所示。

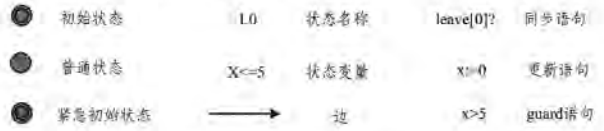

图 1　UPPAAL 模型图例

3　观察者模式及对应的 UPPAAL 模型

本节主要以文献[2]中的无环版本的有限时间响应模式为例,对 UPPAAL 模型的构建过程进行描述,并给出部分典型的观察者模式对应的 UPPAAL 模型以及用途。

3.1　UPPAAL 模型的构建

文献[2]中的无环版本的有限时间相应模式如图 2 所示,其表现形式为时间自动机。

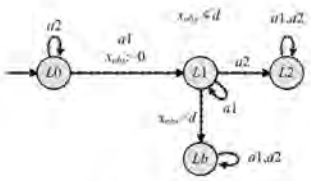

图 2 时间自动机表示的无环的有限时间响应的观察者模式

其对应的 UPPAAL 模型的构建过程及每个参数的意义如下。

1)在 UPPAAL 中添加时间自动机中的 4 个状态 $L0$,$L1$,$L2$,Lb,其中 $L0$ 是初始状态;

2)定义两个全局变量 $a1$,$a2$ 和一个全局时钟变量 x(对应 x_{obs}),其中 $a1$,$a2$ 主要负责原始模型与观察者模型之间的同步,原始模型通过 a1!和 a2!发布消息,观察者模型通过 a1?和 a2?接收消息;

3)添加 $L0$ 到 $L1$ 状态的转换边,添加更新语句 $x:=0$,在执行过程中将时钟初始化为 0,并添加同步语句 a1?,接收同步消息;

4)添加状态 $L1$ 的变量,令 $x\leqslant d$,如果时钟 x 小于或等于给定的时间 d,则可以进入到 $L1$ 状态,其中 d 为可变量,使用时需替换为具体的时间长度;

5)添加 $L1$ 到 Lb 状态的转换边,添加 Guard 语句(前置条件)$x==d$,满足条件时,系统从状态 $L1$ 进入状态 Lb;

6)添加 $L1$ 到 $L2$ 状态的转换边,添加同步语句 a2?;

7)分别在 4 个状态上添加相应的自环边,并添加同步语句。

构建好的模型如图 2 所示,其他模型的构建过程及参数意义基本同上。

无环的有限时间响应的观察者模式对应的 UPPAAL 模式如图 3 所示。

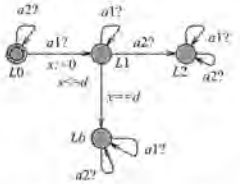

图 3 无环的有限时间响应的观察者模式对应的 UPPAAL 模型

3.2 部分典型的观察者模式对应的 UPPAAL 模型及用途

观察者模式包括存在模式、不存在模式、响应模式、优先级模式以及顺序模式等。本节将给出部分典型的观察者模式的用途以及对应的 UPPAAL 模型。

3.2.1 存在模式(Existence Patterns)

存在模式通常用来表示“在系统的每条路径中,某些事件一定会发生”。

(1)事件发生后的有限时间内的存在模式用于描述系统中一个事件第一次出现之后,另一个事件在一个时间段($d1-d2$)内都存在的情况,其 UPPAAL 模型如图 4 所示。

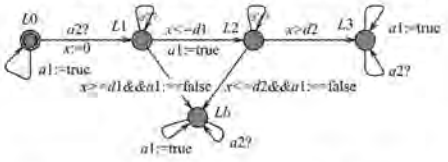

图 4 事件发生后的有限时间内的存在模式 UPPAAL 模型

(2)事件发生前的有限时间内的存在模式用于描述系统中一个事件第一次发生之前,另一个事件在一个时间段($d1-d2$)内都存在的情况,其 UPPAAL 模型如图 5 所示。

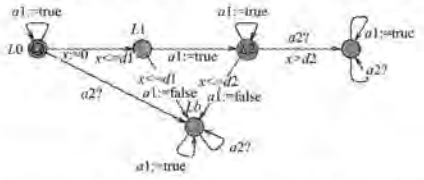

图 5 事件发生前的有限时间内的存在模式 UPPAAL 模型

3.2.2 不存在模式(Absence Patterns)

不存在模式通常用来表示“某些情况不应该发生”。

(1)事件发生后的有限时间内的不存在模式用于描述系统中一个事件第一次出现后的一个时间段 d 内,另一个事件永远不成立的情况,其 UPPAAL 模型如图 6 所示。

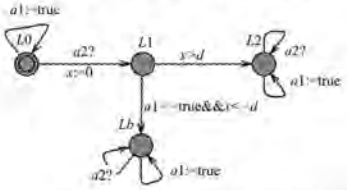

图 6 事件发生后的有限时间内的不存在模式 UPPAAL 模型

(2)事件发生前的有限时间内的不存在模式用于描述系统中一个事件出现之前的一个时间段 d 内,另一个事件不能出现的情况,其 UPPAAL 模型如图 7 所示。

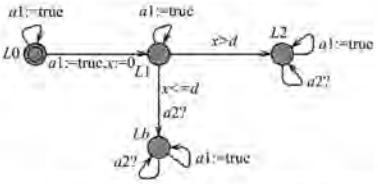

图 7 事件发生前的有限时间内的不存在模式 UPPAAL 模型

3.2.3 响应模式(Response Patterns)

响应模式通常用来表示“一个触发事件发生之后必须总是跟随一个响应事件的发生”。

(1)严格循环版本的响应模式用于描述系统中每次一个事件 $a1$ 发生后都会有另一个事件 $a2$ 发生且只发生一次,且在 $a1$ 事件下一次发生之前,其 UPPAAL 模型如图 8 所示。

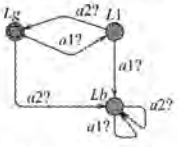

图 8 严格循环版本的响应模式 UPPAAL 模型

(2)无环的有限时间内的响应模式用于描述系统中一个事件发生,另一个事件在 d 个时间单元内最终都会发生的情况,其 UPPAAL 模型如图 9 所示。

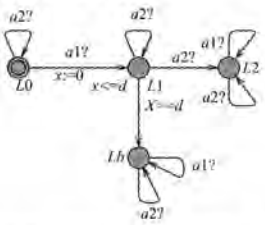

图 9 无环版本的有限时间内的响应模式 UPPAAL 模型

3.2.4　优先级模式(Precedence Patterns)

优先级模式通常用来表示“在每一个信号事件发生之前必须有一个给定的触发事件发生”。

(1)严格循环版本的优先级模式用于描述系统中一个事件 $a2$ 每次发生,另一个事件 $a1$ 在之前已经发生且只发生了一次,直到 $a2$ 最后一次发生的情况,其 UPPAAL 模型如图 10 所示。

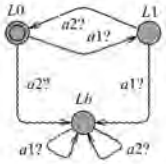

图 10　严格循环版本的优先级模式 UPPAAL 模型

(2)无环版本的有限时间内的优先级模式用于描述系统中一个事件至少发生 1 次,则该事件在另一个事件第一次发生的一个时间段内已经发生的情况,其 UPPAAL 模型如图 11 所示。

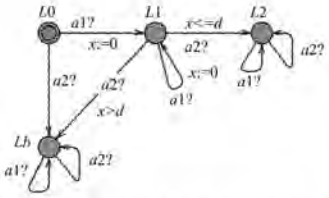

图 11　无环版本的有限时间内的优先级模式 UPPAAL 模型

3.2.5　顺序模式(Sequence Patterns)

顺序模式通常用来表示“n 个事件以一个预先定义的顺序一个接一个地执行”。

(1)无环版本的顺序模式的抽象语义为:sequence a_1,…,a_n,用于描述系统中事件的发生情况必须按照一定顺序的情况,其 UPPAAL 模型如图 12 所示。

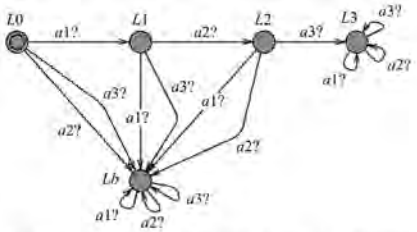

图 12　无环版本的顺序模式 UPPAAL 模型

4　基于观察者模式的 Train-Gate 系统的模型检测

本节以第 3 节为基础,将 Train-Gate 系统作为案例,构建 Train-Gate 不同场景下的 UPPAAL 观察者模型,以描述其验证特性;对 Train-Gate 系统进行模型检测,从而得到最终的结果。

4.1　Train-Gate 系统描述

本文采用 UPPAAL 中较为经典的轨道控制(Train-Gate)系统作为实例,并增加一个布尔变量 *light*,对该系统进行扩展,如图 13 所示。火车过桥问题在现实生活中经常遇到,为了提高火车过桥的效率和安全性,通常采用一个实时系统对其进行控制。Train-Gate 是控制通过一座桥的多个火车的系统,其中桥是一个共享资源,同一时间只能有一辆火车通过。系统由多列火车和一个控制器组成,火车不能立即停止,且火车启动也需要一定的时间。因此,火车在过桥之前具有一定的时间限制。当火车靠近桥时,发送一个 appr!信号,然后它有 10 个时间单元来接收一个停止信号,使得火车可以在桥前安全停止。如果火车停止了,当桥上的上一列火车经过后发送一个 leave!信号,控制器对火车发送一个 go!信号;若 10 个时间单元内火车没有停止,则火车可以通过。火车过桥时,信号灯亮(true);通过后,信号灯熄灭(false)。

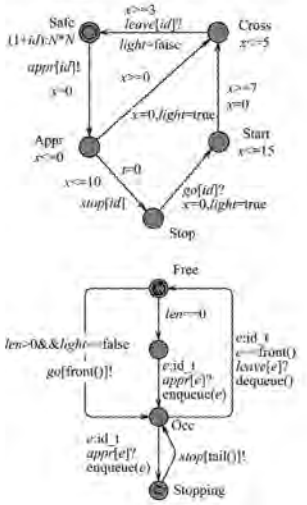

图 13　Train-Gate 模型

4.2　Train-Gate 系统观察者建模和验证特性描述

本节根据第 3 节给出的观察者模式对应的 UPPAAL 模型,以 Train-Gate 系统作为实例,构建 Train-Gate 系统不同场景下的观察者模型,并给出相应的验证特性。

(1)场景 1

火车 0 接收到 go[0]信号,信号灯 *light* 在时间段 0－15 内都是亮着的。根据场景描述,选择事件发生后的有限时间的存在模式,根据图 4,将 $a1$ 和 $a2$ 分别替换为 *light* 和 *go*[0],将 $d1$ 和 $d2$ 分别替换为 0,15,得到如图 14 所示的模型。验证特性为:A[] Observer.L3。

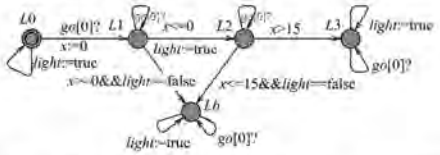

图 14　Train-Gate 场景 1 的观察者 UPPAAL 模型

(2)场景 2

事件 *leave*[0]第一次发生之前,信号灯 *light* 在时间单元 0－5 内都是亮着的。选择事件发生前的有限时间的存在模式,根据图 5,将 $a1$ 和 $a2$ 分别替换为 *light* 和 *leave*[0],将 $d1$ 和 $d2$ 分别替换为 0,5,得到如图 15 所示的模型。验证特性为:A[] Observer.L3。

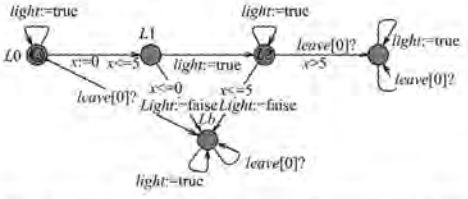

图 15　Train-Gate 场景 2 的观察者 UPPAAL 模型

(3)场景 3

事件 *stop*[0]第一次出现后的 15 个时间单元内，*light* 为真永不成立。选择事件发生后的有限时间内的不存在模式，根据图 6，将 *a*1 和 *a*2 分别替换为 *light* 和 *stop*[0]，将 *d* 替换为 15，得到如图 16 所示的模型。验证特性为：*A*[] Observer. *L*2。

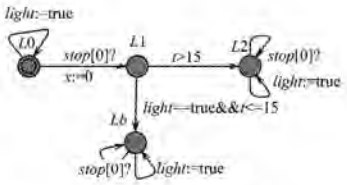

图 16 Train-Gate 场景 3 的观察者 UPPAAL 模型

(4)场景 4

在事件 *leave*[0]发生之前的 30 个时间单元内，信号灯 *light* 不能是亮的。选择事件发生前的有限时间内的不存在模式，根据图 7，将 *a*1 和 *a*2 分别替换为 *light* 和 *leave*[0]，将 *d* 替换为 30，得到如图 17 所示的模型。验证特性为：*A*[] Observer. *L*2。

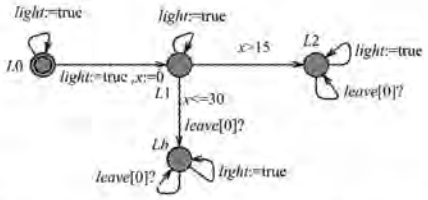

图 17 Train-Gate 场景 4 的观察者 UPPAAL 模型

(5)场景 5

如果火车 0 在靠近桥时接收到 *stop*[0]信号，那么 *go*[0]最终会发生。选择无环版本的响应模式，将 *a*1 和 *a*2 替换为 *go*[0]和 *stop*[0]，得到如图 18 所示的模型。验证特性为：*A*[] Observer. *L*2。

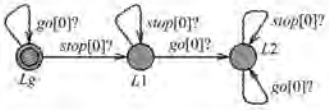

图 18 Train-Gate 场景 5 的观察者 UPPAAL 模型

(6)场景 6

每次接收到 *appr*[0]信号，在下一次 *appr*[0]信号发出之前事件 *leave*[0]最终都会发生且只发生一次。选择严格循环版本的响应模式，根据图 8，将 *a*1 和 *a*2 替换为 *appr*[0]和 *leave*[0]，得到如图 19 所示的模型。验证特性为：*E*〈〉 Observer. *Lb*。

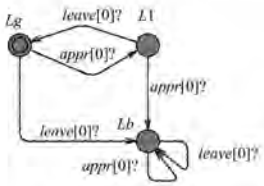

图 19 Train-Gate 场景 6 的观察者 UPPAAL 模型

(7)场景 7

如果火车 0 在靠近桥的过程中接收到 *stop*[0]信号，那么事件 *go*[0]在 20 个时间单元内最终会发生。选择有限时间的无环版本响应模式，根据图 9，将 *a*1 和 *a*2 替换为 *stop*[0]和 *go*[0]，将 *d* 替换为 20，得到如图 20 所示的模型。验证特性为：*A*[] Observer. *L*2。

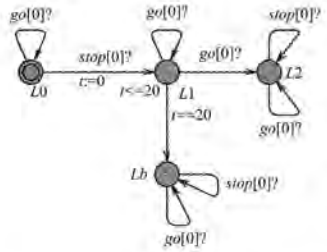

图 20 Train-Gate 场景 7 的观察者 UPPAAL 模型

(8)场景 8

事件 *appr*[0]每次发生，下一次 *appr*[0]发生之前，事件 *leave*[0]在 100 个时间单元内最终都会发生且只发生一次。选择有限时间的严格循环的响应模式，得到如图 21 所示的模型。验证特性为：*A*[] Observer. *L*0。

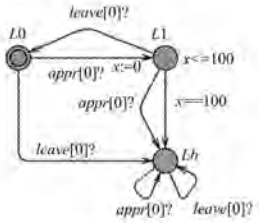

图 21 Train-Gate 场景 8 的观察者 UPPAAL 模型

(9)场景 9

事件 *appr*[0]将会在系统启动后的 20 个时间单元内发生。选择截止时间前的响应模式，得到如图 22 所示的模型。验证特性为：*A*[] Observer. *L*2。

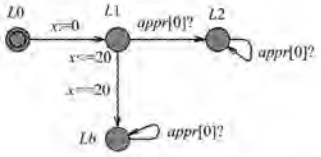

图 22 Train-Gate 场景 9 的观察者 UPPAAL 模型

(10)场景 10

若火车 0 接收到 *stop*[0]信号，那么在 *stop*[0]发生之前的 10 个时间单元内 *appr*[0]已经发生。选择无环的有限时间的优先级模式，根据图 11，将 *a*1 和 *a*2 替换为 *appr*[0]和 *stop*[0]，将 *d* 替换为 10，得到如图 23 所示的模型。验证特性为：*A*[] Observer. *L*2。

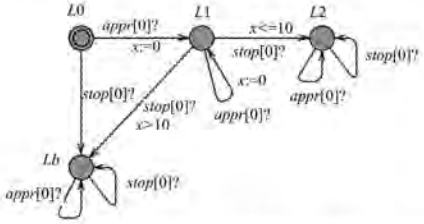

图 23 Train-Gate 场景 10 的观察者 UPPAAL 模型

(11)场景 11

火车按照第 0 辆、第 1 辆的顺序靠近桥。观察者建模:选择无环的顺序模式,根据图 12,得到如图 24 所示的模型。验证特性为:$E\langle\rangle$ Observer. $L2$。

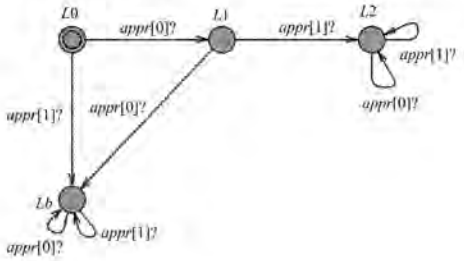

图 24　Train-Gate 场景 11 的观察者 UPPAAL 模型

4.3　Train-Gate 系统模型检测

根据上述的应用场景和对应的观察者模型,以应用场景"如果火车 0 在靠近桥的过程中接收到 $stop[0]$信号,那么事件 $go[0]$在 20 个时间单元内最终会发生"为例,对验证过程进行详细描述,具体过程如下。

1)根据选定好的场景,向 Train-Gate 模型中添加相应的观察者模型;

2)编写验证性质:$A[]$ Observer. $L2$;

3)在 UPPAAL 中添加上述验证性质,进行验证;

4)验证结果:不满足该性质。

其他场景下具体的模型检测过程同上,最终的验证结果如表 1 所列。

表 1　Train-Gate 系统模型检测结果

场景编号	验证性质	验证结果
1	$A[]$ Observer. $L3$	不满足
2	$A[]$ Observer. $L3$	不满足
3	$A[]$ Observer. $L2$	不满足
4	$A[]$ Observer. $L2$	不满足
5	$A[]$ Observer. $L2$	不满足
6	$E\langle\rangle$ Observer. Lb	不满足
7	$A[]$ Observer. $L2$	不满足
8	$A[]$ Observer. $L0$	不满足
9	$A[]$ Observer. $L2$	不满足
10	$A[]$ Observer. $L2$	不满足
11	$E\langle\rangle$ Observer. $L2$	满足

5　实验验证

本节将通过实验得出不使用观察者时 Train-Gate 系统的模型检测结果,将该结果与第 4 节使用观察者模式的结果进行对比,得出两种情况下模型检测结果及验证特性复杂度之间的差异,并进行分析。

5.1　不采用观察者模式的 Train-Gate 系统的模型检测

以场景 7 为例,对不采用观察者模式的验证过程进行详细描述,具体过程如下。

1)根据选定的场景编写响应的验证性质:$Train(0)$. Appr$->Train(0)$. Start and $t<=20$;

2)在 UPPAAL 中添加上述验证性质,进行验证;

3)验证结果:不满足该性质。

Train-Gate 案例中其他应用场景下的验证过程同上。表 2 列出了具体的验证场景、验证性质以及验证结果。

表 2　不采用观察者模式的 Train-Gate 验证特性及验证结果

场景编号	验证性质	验证结果
1	$Train(0)$. Stop $->Train(0)$. Start and ($a1==$ true and $x>=0$ and $x<=15$)	不满足
2	$Train(0)$. Appr $->$ $Train(0)$. Cross and ($light==$ true and $x>=0$ and $x<=5$)	不满足
3	$Train(0)$. Stop $->$ $Train(0)$. Start and $light==$ true and t<=15	不满足
4	$Train(0)$. Stop $->$ $Train(0)$. Cross and ($a1==$ false and $x<=30$)	不满足
5	$Train(0)$. Appr $->Train(0)$. Start	不满足
6	$Train(0)$. Appr $->$ $Train(0)$. Cross	满足
7	$Train(0)$. Appr $->Train(0)$. Start and $t<=20$	不满足
8	$Train(0)$. Appr $->Train(0)$. Cross and $x<=100$	不满足
9	$Train(0)$. Safe $->Train(0)$. Appr and $x<=20$	不满足
10	$Train(0)$. Appr and $x<=10->$ $Train(0)$. Stop	不满足
11	$E\langle\rangle$ $Train(0)$. Appr and $Train(1)$. Appr	满足

5.2　验证结果对比及分析

本节主要从验证特性的复杂度、验证结果的一致性两个方面对采用观察者前后所得到的结果进行对比分析,并对结果进行分析。

表 3　验证结果及验证特性复杂度的对比

场景编号	验证效率				验证结果
	验证费时(s)/Kernel 费时(s)/总费时(s)		常驻内存(kB)/虚拟内存的使用峰值(kB)		
	使用观察者	无观察者	使用观察者	无观察者	
1	0/0/0	0/0.016/0.015	7616/27364	7816/27664	一致
2	0/0/0.001	0.016/0/0.005	8180/28412	8240/28572	一致
3	0/0/0.001	0.016/0/0.006	8680/29452	8780/29656	一致
4	0/0/0.001	0.016/0/0.007	9072/30248	9132/30356	一致
5	0/0/0.001	0.078/0.015/0.1	9448/31044	9560/31172	一致
6	0.062/0/0.076	0624/0.11/0.875	16944/46004	56680/125248	一致
7	0/0/0.001	0/0/0.007	56456/125048	56460/125056	一致
8	0/0/0.001	1.139/0.093/1.309	7696/27460	9136/30132	一致
9	0/0/0.001	0/0/0.003	56456/125048	56460/125056	一致
10	0/0/0.001	0.016/0/0.006	56456/125048	56460/125056	一致
11	0/0/0.001	0/0/0.004	56456/125048	56456/125048	一致

从表 3 可以看出,采用观察者得到的验证结果与不采用观察者的相同。从结果的正确性而言,观察者模式可以应用到实时系统的验证中,这个方案是可行的。从验证特性而言,通过比较表 1 与表 2 中的验证性质的复杂度,如场景 1 的验证特性:"$A[]$ Observer. $L3$"与"$Train(0)$. Stop $->Train(0)$. Start and ($a1==$true and $x>=0$ and $x<=15$)",可以发现采用观察者的验证特性比无观察者的更简单,仅为单纯的状态可达性问题,不需要使用复杂的验证算法,编写也较为简单。从验证的时间上来看,不采用观察者进行验证时的"验证费时/Kernel 费时/总费时"比采用观察者进行验证时的高,说明采用观察者的验证更节省时间。内存方面,不采用观察

者的验证所占用的内存大于或等于采用观察者的情况，说明采用观察者的验证在某些情况下更节省空间。

6 讨论

根据第 5 节的对比结果可以看出，采用观察者模式对系统进行验证，可以得到与采用无观察者验证特性相同的验证结果，可以得到正确的验证结果，即观察者模式对于实时系统的验证是可行的。

采用观察者模式对系统进行验证可以将复杂的验证特性转换为某一个状态的可达性问题，如可以将“*Train*(0). Appr－＞*Train*(0). Start and $t<=20$”转换为“$A[]$ Observer. $L2$”，避免使用复杂的验证算法，只需判断 $L2$ 状态是否可达，降低对验证算法的要求。

编写验证特性要求验证人员对－＞，imply，A[]，E〈〉等相关基础知识有一定的理解，需要考虑每个状态及时间之间的关系，要求较高且编写过程难度较大；而使用观察者模式进行验证，只需根据自己的验证需要对模式中的变量进行替换即可，如有限时间响应模式中只需将事件 $a1$，$a2$，时钟 x 和 d 替换为 $stop[0]$，$go[0]$，t 以及 20 即可，相比于编写复杂的验证特性，观察者对非专业人员而言更简单也更容易接受。

另外，采用观察者前后的“验证费时/Kernel 费时/总费时”以及“常驻内存/虚拟内存的使用峰值”也存在一定的差异，采用观察者的方式可以节省一定的时间和空间。

结束语 采用模型检测工具 UPPAAL 对观察者模式进行建模，将复杂的验证特性转换为某一个状态的可达性问题，给出了具体的模式实现和解释。以 Train-Gate 系统模型为例，对给出的观察者模式进行了验证，并列举了部分验证结果。结果表明，观察者模式可以用于如 Train-Gate 的实时系统的验证，同时可以将复杂的验证特性转化为可达性问题，避免使用复杂的验证算法，降低了对验证算法的要求；同时，观察者模式不需要掌握熟练的专业知识，相比于编写负责验证特性，模式的套用也可以节约时间，对于非专业的验证人员而言，更简单，也更容易接受。

本文主要关注观察者模式在实时系统验证过程中的可行性，以及观察者能否缓解实时系统验证过程中的复杂性问题，因此选取了 UPPAAL 中比较经典的案例进行验证，忽略了系统的规模问题。在今后的工作中，会选取更大规模的实例对观察者模式在实时系统验证中的性能问题进行进一步的验证和研究。

参 考 文 献

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

[2] ANDRE E. Observer patterns for real-time systems[C]// 18th International Conference on Engineering of Complex Computer Systems(ICECCS 2013). Singapore, 2013: 125-134.

[3] SUN J L, YANG D, DONG J S, et al. Modeling and verifying hierarchical real-time systems using Stateful Timed CSP[J]. ACM Transactions on Software Engineering and Methodology, 2013, 22(1): 1-29.

[4] HOENICKE J. Combining specification techniques for processes, data and time[C]// International Conference on Integrated Formal Methods. Springer-Verlag, 2002: 245-266.

[5] MAHONY B P, DONG J S. Overview of the semantics of TCOZ[C]// International Conference on Integrated Formal Methods. 1999: 66-85.

[6] MERLIN P M. A study of the recoverability of computing systems[OL]. http://www.informatik/vni-hamburg.ed/TGI/pnbib/m/merlin_p_m8.html.

[7] BOUYER P, PETIT A. Decomposition and composition of timed automata[J]. Lecture Notes in Computer Science, 1999, 39(6): 210-219.

[8] DONG J S, HAO P, QIN S, et al. Timed automata patterns[J]. IEEE Transactions on Software Engineering, 2008, 34(6): 844-859.

[9] JENSEN K, KRISTENSEN L M. Coloured Petri Nets-Modelling and Validation of Concurrent Systems[M]. Springer, 2009: 19-34.

[10] DAVID A, LARSEN K G, LEGAY A, et al. ECDAR: An environment for compositional design and analysis of real time systems[C]// Internation Conference on Automated Technology for Verification and Analaysis. 2010: 365-370.

[11] REGGIO G, ASTESIANO E, CHOPPY C. CASL-LTL: A CASL extension for dynamic reactive systems version 1.0- summary [OL]. http://www-lipn.univ-paris13.fr/～choppy/PAPERS/N-CASLTL.pdf.

[12] CHOPPY C, REGGIO G. A formally grounded software specification method[J]. Journal of Logic and Algebraic Programming, 2006, 67(1): 52-86.

[13] ACETO L, LARSEN K G. Model checking via reachability testing for timed automata[C]// Internation Conference on Tools and Algorithms for the Construction and Analysis of Systems. 1998: 263-280.

[14] AECTO L, BOUYER P, BURGUEÑO A, et al. The power of reachability testing for timed automata[C]// Foundation of Software Technology and Theoretical Computer Science. India, 1998: 245-256.

[15] KHATIB L, MUSCETTOLA N, HAVELUND K. Mapping temporal planning constraints into timed automata[C]// TIME'01. 2001: 21-27.

[16] MEKKI A, GHAZEL M, TOGUYENI A. Validating time-constrained systems using UML statecharts patterns and timed automata observers[C]// VECoS'09. 2009: 112-124.

[17] ABID N, ZILIO S D, BOTLAN D L. Real-Time Specification Patterns and Tools[M]// Formal Methods for Industrial Criticla Systems. Springer Berlin Heidelberg, 2012: 1-15.

[18] DWYER M B, AVRUNIN G S, CORBETT J C. Patterns in property specifications for finite-state verification[C]// International Conerence on Software Engineering. 1999: 411-420.

[19] ABID N, ZILLIO S D, BOTLAN D L. A Real-Time Specification Patterns Language[OL]. http://hal.archives-uvertes.fr/hal-00593965.

（下转第 174 页）

间的触发类型事件的时序关系，获得系统内部构件间的所有不可达状态对，删除构件间不可同时到达状态组，对构件间的可同时到达状态建立笛卡尔积，通过连接表和最小割序集建立系统失效生成图。实验结果验证了方法的可行性与稳定性。所提方法有效解决了作战系统进行可靠性[13-16]分析时存在的状态空间爆炸问题，同时也为状态事件故障树生成系统可达图提供了一种新的方法。

参 考 文 献

[1] KAISER B. State Event trees: A safety and reliability analysis techniqure for software controlled systems[D]. Kaiser-slautern: University Kaiserslautern, 2007.

[2] KAISER B, GRAMLICH C. State-Event-Fault-Trees-A Safety Analysis Model for Software Controlled Systems[J]. Reliability Engineering and System Safety, 2007, 92(11): 1521-1537.

[3] GUCK D, HAN T, KATOEN J P, et al. Quantitative timed analysis of interactive markov chains [M]// NASA Formal Methods. Springer Berlin Heidelberg, 2012: 8-23.

[4] XU B F. Research on security analysis method of component based embedded software [D]. Nanjing: Nanjing University of Aeronautics and Astronautics, 2014. (in Chinese)
徐丙凤. 构件化嵌入式软件安全性分析方法研究[D]. 南京：南京航空航天大学，2014.

[5] ROTH M, LIGGESMEYER P. Qualitative analysis of state/event fault trees for supporting the certification process of software-intensive systems[C]// IEEE International Symposium on Software Reliability Engineering Workshops. 2013: 353-358.

[6] XU B F, HUANG Z Q, HU J, et al. A state event fault tree quantitative analysis method [J]. Chinese Journal of Electronics, 2013, 41(8): 1480-1486. (in Chinese)
徐丙凤，黄志球，胡军，等. 一种状态事件故障树的定量分析方法[J]. 电子学报，2013，41(8)：1480-1486.

[7] LIU W B. Study on the dynamic fault tree analysis method based on modular idea [D]. Nanjing: Nanjing University of Science and Technology, 2009. (in Chinese)
刘文彬. 基于模块化思想的动态故障树分析方法研究[D]. 南京：南京理工大学，2009.

[8] ROTH M, HARTOYO A, LIGGESMEYER P. Efficient reachability graph development for qualitive analysis of state/event fault trees[C]// IEEE International Symposium on Software Reliability Engineering Workshops. 2015: 144-151.

[9] ROTH M, LIGGESMEYER P. Sequential Logic for State/Event Fault Trees: A Methodology to Support the Failure Modeling of Cyber Physical Systems[M]// Computer Safety, Reliability, and Security. 2015: 121-132.

[10] ROTH M, LIGGESMEYER P. Modeling and Analysis of Safety-Critical Cyber Physical Systems using State/Event Fault Trees[OL]. http://hal.archives-ouvertes.fr/hal-00848640.

[11] WALKER M D. Pandora: A Logic for the Qualitative Analysis of Temporal Fault Trees[D]. University of Hall, UK, 2009.

[12] TANG Z, DUGAN J B. Minimal cut set/sequence generation for dynamic fault trees[C]// Proccedings of the Annual Reliability and Maintainability Symposium (RAMS). Charlottesville, USA, 2004: 207-213.

[13] LIU D. Key technology research on reliability design and analysis of spatial information processing system [D]. Changsha: National Defense Science and Technology University, 2008. (in Chinese)
刘东. 空间信息处理系统可靠性设计与分析关键技术研究[D]. 长沙：国防科学技术大学，2008.

[14] LI Y F. New method of dynamic fault tree analysis of complex system and its application [D]. Chengdu: Electronic Science and Technology University, 2013. (in Chinese)
李彦锋. 复杂系统动态故障树分析的新方法及其应用研究[D]. 成都：电子科技大学，2013.

[15] QIN Q N. The complex system reliability modeling, analysis and comprehensive evaluation method of [D]. Beijing: Beijing Jiaotong University, 2012. (in Chinese)
覃庆努. 复杂系统可靠性建模、分析和综合评价方法研究[D]. 北京：北京交通大学，2012.

[16] GUO Y. Research on reliability evaluation method of software system based on component [D]. Harbin: Harbin Institute of Technology, 2013. (in Chinese)
郭勇. 基于构件的软件系统的可靠性评估方法研究[D]. 哈尔滨：哈尔滨工业大学，2013.

(上接第 162 页)

[20] ABID N, ZILLIO S D, BOTLAN D L. Real-time specification patterns and tools[C]// 17th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2012). Paris, 2012: 1-15.

[21] CHEN Z Y, HUANG S B, BAI Y, et al. CTL formalized specification templates in model checking[J]. Journal of Harbin Engineering University, 2013, 34(4): 483-487. (in Chinese)
陈志远，黄少滨，白玉，等. 模型检测中的 CTL 形式化描述模板[J]. 哈尔滨工程大学学报，2013，34(4)：483-487.

[22] BEHRMANN G, DAVID A, LARSEN K. A tutorial on UPPAAL[J]. Formal Methods For The Design of Real-time Systems, 2004, 4(12): 200-236.

[23] DAI S X, HONG M, GUO B, et al. Schedulability Analysis Model for Multiprocessor Real-Time Systems Using UPPAAL[J]. Journal of Software, 2015(2): 279-296. (in Chinese)
代声馨，洪玫，郭兵，等. 多处理器实时系统可调度性分析的 UPPAAL 模型 [J]. 软件学报，2015(2)：279-296.