一种面向性质的实时系统测试方法

李书浩[1],王 戟[1],齐治昌[1],董 威[1,2]
(1. 国防科技大学计算机学院,湖南长沙 410073;2. 武汉大学软件工程国家重点实验室,湖北武汉 430072)

摘 要: 尽管 Statecharts 在反应式实时系统建模领域获得了广泛应用,基于 Statecharts 开发的实时软件的测试仍然十分困难. 由于引入了时间维,待测系统的行为空间变得非常庞大,使得难以对其进行全面深入测试. 本文提出了一种面向性质的实时系统测试方法. 首先对 UML Statecharts 作适当实时扩展,使得扩展后能描述 non-trivial 时间约束;然后用一种受限实时逻辑描述待测系统的功能特性;在此基础上根据待测性质从系统模型生成有针对性的测试序列. 实验表明,在相同测试深度下,面向性质测试比非面向性质测试需要少得多的测试序列.

关键词: 软件测试; 实时系统; UML Statecharts; 时序逻辑; 测试序列
中图分类号: TP311 **文献标识码**: A **文章编号**: 0372-2112 (2005) 05-0827-08

An Approach to Property-Oriented Testing of Real-Time Systems

LI Shu-hao[1], WANG Ji[1], QI Zhi-Chang[1], DONG Wei[1,2]
(1. *School of Computing, National University of Defense Technology, Changsha, Hunan* 410073, *China*;
2. *State Key Laboratory for Software Engineering, Wuhan University, Wuhan, Hubei* 430072, *China*)

Abstract: Although Statecharts have gained widespread use as a formalism for modeling reactive real-time systems, testing these systems still confronts some difficulties, of which a major one is the existence of numerous system behaviors. It is extremely difficult to conduct comprehensive and in-depth testing of such real-time systems. This paper presents an approach to property-oriented real-time testing. Necessary real-time extensions are proposed such that the time-enriched Statecharts can describe non-trivial timing constraints. The properties to be tested are characterized by a restricted real-time logic. Then the targeted test sequences are derived from the real-time models according to the user-specified properties. Using this approach, testing efforts can be focused on particular properties of the real-time systems and usually only a small portion of the total behaviors needs to be tested.

Key words: software testing; real-time system; UML Statecharts; temporal logic; test sequence

1 引言

由于 UML Statecharts[1] 描述问题直观、描述机制丰富,因此它常常用于刻画对象在其生命周期中的实时行为. 随着越来越多的反应式实时系统基于 Statecharts 进行开发,这些系统的测试变得越来越重要.

反应式实时系统测试面临的主要困难是行为空间极其庞大. 导致此困难的原因包括:(1) 系统本身控制结构复杂,(2) 系统中含有数据变量,(3) 时间维的引入. 三者中后者影响尤其显著. 对于这类系统,即便是中等规模系统,对其进行全面深入测试也十分困难. 然而我们注意到,在许多场合人们仅对待测实时系统的某些特定性质感兴趣,希望对其进行专门深入测试,或者在资源有限情况下,只能对系统的关键性质进行重点测试. 在这些场合,对系统进行针对性测试就显得非常必要.

本文提出了一种面向性质的从实时扩展后的 UML Statecharts 生成测试序列的方法,它适合对基于 UML 模型设计开发/生成的实时软件的测试. 令系统模型为 M,待测性质为 P. 假设 M 满足 P. 面向性质测试的原理是:只对模型 M 中使得性质 P 的前件为真的行为进行测试,如图 1 所示. 一般来说,使性质前件为真的模型行为(待测行为)远远少于不加限制的模型行为,因此在相同资源条件下,面向性质测试比非面向性质测试进行得更深入.

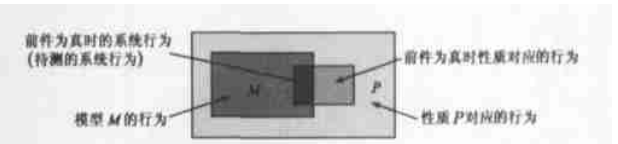

图 1 面向性质测试的原理

第 2 节对 UML 作了必要实时扩展,使得扩展后 Statecharts 能自然地描述 non-trivial 时间约束. 然后用扩展有限状态机(EFSM)表示扩展后 Statecharts 的所有行为. 第 3 节定义了一种

收稿日期:2004-04-12;修回日期:2004-12-08
基金项目:国家自然科学基金(No. 60233020, No. 90104007, No. 60303013);国家 863 项目(No. 2001AA113202, No. 2001AA113190);武汉大学软件工程国家重点实验室开放基金(No. SKLSE03-08);国防科技大学计算机学院基金"基于构件的软件开发与测试"

用于描述反应式实时系统功能特性的受限实时逻辑. 第 4 节提出了面向该类性质的测试序列生成方法. 第 5 节对所给测试方法进行了案例分析. 第 6 节将本文方法与相关方法作了比较. 第 7 节给出了结论和下一步研究方向.

2 实时系统规约

2.1 Real-Time Statecharts 及其语义

经典 Statecharts 和 UML Statecharts 可以直接描述"迁移在指定时间之后发生",但不能直接描述"迁移在指定时间之前发生". 为了便于描述时间区间上限,我们在 UML 元模型层次作实时扩展,使得扩展后 Statecharts 中,外部输入事件可以与时间区间联合构成带时间的激励,例如 $[lower, upper]\{e_1, e_2, \cdots, e_n\}$. 称扩展后的 Statecharts 为 Real-Time Statecharts.

为了简化问题,我们对扩展前 UML Statecharts 作适当限定:暂不考虑各种伪状态;假定事件不带参数;不考虑迁移优先级;不考虑复合迁移;不考虑与状态相关的动作和活动. 此外,假设所有变量都是离散的,且都在有穷集合内取值.

定义 1 (Real-Time Statecharts, RTS) Real-Time Statecharts 是一个 7 元组 $Z = (S, C_0, \Pi, V, \Theta, R, T)$,其中 S 是状态集,C_0 是 RTS 所处的一组初始状态,Π 是事件集,V 是变量集,Θ 是 V 上的一个解释,它为 V 中每个变量赋一个初始值,R 是由非负实数构成的时间域,T 是迁移集.

RTS 事件集 Π 分成三个不相交子集 Π_I, Π_L 和 Π_O,分别表示(外部)输入事件、局部事件和(外部)输出事件. 输入事件由外部环境产生. 在 RTS 中,我们采用类似 UML Statecharts 的单事件处理机制,即任意时刻最多处理一个输入事件. 局部事件由 RTS 自身产生,用于内部通讯,外部不可见. 输出事件是 RTS 对激励作出的外部可观察响应.

迁移集 T 中每一个迁移 t 是一个 6 元组 $(source(t), time(t), events(t), guard(t), action(t), target(t))$,其中 $source(t)$, $target(t) \in S$; $time(t) = [l, u]$ 是一段时间延迟,或者是输入事件发生的时间区间; $events(t)$ 要么是空集,要么是 Π_I 的子集,要么是 Π_L 的子集; $guard(t)$ 是 V 上的一个谓词; $action(t)$ 包括两部分:一部分是对 V 中变量的一组赋值 $assignments(t)$,另一部分是 $\Pi_L \cup \Pi_O$ 中的一组事件 $generated(t)$.

输入事件集合可以与时间区间相关联,构成带时间激励,但局部事件不能这样关联. 带时间激励的迁移标记的一般形式是 $[l, u]\{e_1, e_2, \cdots, e_n\}[g]/a$,其中 $e_i \in \Pi_I, 1 \le i \le n$. 其直观含义是,从进入迁移源状态开始计时,如果在时间 l 之后、时间 u 之前有某个输入事件 $e_i (1 \le i \le n)$ 发生,并且 e_i 发生时刻条件 g 满足,那么迁移被激发. 上述迁移标记中,时间区间既可以是闭区间,也可以是半开区间或者开区间,如 (l, u), $(l, u]$, $[l, u)$. 此外,若 $l = u$,则 $[l, u]$ 可写为 $after(l)$;若 $l = 0$ 且 $u = \infty$,则 $[l, u]$ 可略去不写.

图 2 是用 RTS 描述一个简单自助咖啡机 (Coffee Vending Machine, CVM) 的例子. CVM 原理是:投币 (*inc*) 后方能要求咖啡 (*coffee*);咖啡机"吃掉"硬币 (*dec*) 后,才开始煮咖啡;开始加热后,若太早接咖啡 ($[0, 2)$ *accept*),则尚未煮好,若太晚接咖啡 ($[6, \infty)$ *accept*),则咖啡已凉,若适时接咖啡 ($[2, 6]$ *accept*),则 1 个时间单位后 (*after*(1)),杯子接满.

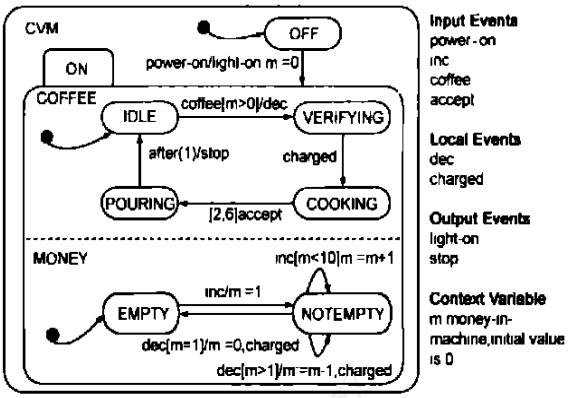

图 2 自助咖啡机的 RTS 规约

我们用异步时间模型[2,3]定义 RTS 语义. RTS 的执行以 step 为语义单位. 一个 step 是被一个外部激励 (即一个纯时间延迟 $[l, u]$,或者一个纯输入事件 e_I,或者一个延迟输入事件 $([l, u] e_I)$ 或者一个局部激励 (即一个局部事件 e_L) 所激发的最大无冲突使能迁移集. 最初,一个外部激励可能激发一组 RTS 迁移,这组迁移中的若干迁移 (一个 step) 被选择执行后,可能产生一些局部事件,而这些局部事件中的某一个又可能激发另一组迁移 (从而进入另一 step) 继续执行. 如此进行"链式反应",直到系统内不再产生局部事件即系统达到稳定状态为止. 系统只有在到达稳定状态后才能接收下一个外部激励.

对于 RTS $Z = (S, C_0, \Pi, V, \Theta, R, T)$,假设 *Config* 是 Z 的所有格局的集合 (我们用 RTS 所处的一组状态表示一个格局),Σ 是 V 的所有解释的集合. 下面给出我们对 RTS 中 step 的定义.

定义 2 (step) 令 C、$C' \in Config$, σ、$\sigma' \in \Sigma$. $\tau \subseteq T$ 是一组迁移; i 称为输入,它是一个单元素集合,其元素要么是一个外部激励,要么是一个内部激励; $o \subseteq \Pi_L \cup \Pi_O$ 称为输出,它是一组局部事件和/或输出事件. 如果下列条件得到满足,则称 3 元组 (i, τ, o) 是从 (C, σ) 到 (C', σ') 的一个 step,记作 $(C, \sigma) \xrightarrow{(i, \tau, o)} (C', \sigma')$,

® τ 中每个迁移被 i 激发,具体地:

-若 i 中元素是一个纯时间延迟 $[l, u]$,则 $\exists v \in [l, u]. \forall t \in \tau. ((v \in time(t)) \wedge (events(t) = \emptyset))$;

-若 i 中元素是一个纯输入事件 e_I,则 $\exists v \in [0, \infty). \forall t \in \tau. ((v \in time(t)) \wedge (e_I \in events(t)))$;

-若 i 中元素是一个延迟输入事件 $[l, u] e_I$,则 $\exists v \in [l, u]. \forall t \in \tau. ((v \in time(t)) \wedge (e_I \in events(t)))$;

-若 i 中元素是一个局部事件 e_L,则 $\forall t \in \tau. ((time(t) = null) \wedge (e_L \in events(t)))$;

® τ 中每个迁移在 (C, σ) 下被使能,即 $\forall t \in \tau. ((source(t) \in C) \wedge (\sigma \models guard(t)))$;

® τ 中迁移无冲突,即 τ 中各迁移离开状态集的交集为空;

® 集合 τ 是最大的,即那些被 i 激发、在 (C, σ) 下被使能的迁移如果不在 τ 中,那么它肯定与 τ 中某个迁移冲突;

® $C' = (C - \cup_{t\in\tau} Exits(t)) \cup (\cup_{t\in\tau} Enters(t))$,其中 $Exits(t)$ 和 $Enters(t)$ 分别表示迁移 t 的离开状态集和进入状态集;

® $o = \cup_{t\in\tau} generated(t)$;

® $\sigma' = \sigma \circ a$,其中 $a = \cup_{t\in\tau} assignments(t)$,"$\circ$"表示函数合成.

我们用可达图(reachability graph)描述 RTS 行为. RTS 可达图是一个状态机,其每一节点(C, E, σ)对应 RTS 的一个状况(status),其中 C 是 RTS 当前格局, E 是上一 step 产生的局部事件的集合,σ是系统变量的当前解释. RTS 可达图中每一条边对应 RTS 中一组迁移.如果 RTS 某个状况下 $E = \emptyset$成立,则表明'链式反应'已经终止,称这种状况为稳定状况.

我们用 2^{Π} 表示 Π 的幂集,用 $[[R]]$ 表示 RTS 中所出现的所有时间区间的集合. RTS 可达图定义如下.

定义 3(RTS 可达图)　RTS $Z = (S, C_0, \Pi, V, \Theta, R, T)$ 在异步时间模型下的可达图 $G(Z)$ 是一个状态机

$$G(Z) = (Config \times 2^{\Pi} \times \Sigma, (C_0, \emptyset, \Theta), [[R]] \times 2^{\Pi}, 2^{\Pi}, \delta)$$

它满足:$((C, E, \sigma), i, o, (C', E', \sigma')) \in \delta$ 当且仅当 Z 中存在从(C, σ)到(C', σ')的一个 step (i, τ, o) 满足:(1) 若 $E \neq \emptyset$,则 i 是一个内部激励,否则, i 是一个在 RTS 中出现的外部激励,(2) $E' = o \cap \Pi_L$.

RTS Z 的行为用其可达图 $G(Z)$ 中的路径来刻画. $G(Z)$ 中从初始格局出发的每条路径是 Z 在异步时间模型下的一个(抽象)运行. Z 的所有运行(即行为)表示为 $B(Z)$.

2.2　从 RTS 到 EFSM

尽管 RTS 有较丰富的实时建模设施,但它不便于直接处理以生成测试用例.扩展有限状态机(EFSM)能同时建模反应式系统的控制行为和数据结构,且相对简单,因此本文用 EFSM 描述 RTS 行为,然后在 EFSM 基础上进行测试生成.

定义 4(扩展有限状态机,EFSM)　扩展有限状态机 M 是一个 7 元组, $M = (Q, q_0, I, O, V, \Theta, \delta)$,其中 Q 是有穷状态集; q_0 是初态; I, O, V 分别是输入符号、输出符号、变量的有穷集; Θ 是 V 的一个解释,它为 V 中每个变量赋初值; δ 是有穷迁移集. δ 中每个迁移是一个 6 元组 (q, i, o, g, a, q'),其中 $q, q' \in Q$, $i \in I$, $o \in O$, g 是 V 中变元上的谓词, a 是对 V 中变元的一组赋值.

EFSM 的形式语义用可达图表示如下.

定义 5(EFSM 可达图)　令 $M = (Q, q_0, I, O, V, \Theta, \delta)$ 是一个 EFSM, Σ 为 V 的所有解释的集合. M 的可达图 $G(M)$ 是这样一个状态机

$$G(M) = (Q \times \Sigma, (q_0, \Theta), I, O, \delta')$$

它满足:如果存在迁移 $(q, i, o, g, a, q') \in \delta$ 使得 $\sigma \models g$ 且 $\sigma' = \sigma \circ a$,那么 $((q, \sigma), i, o, (q', \sigma')) \in \delta'$.

EFSM M 的行为用 $G(M)$ 中的路径来刻画. $G(M)$ 中每条路径称为 M 的一个运行. M 的所有运行(行为)表示为 $B(M)$.

本文采用如下方法将 RTS 转换成对应的 EFSM:(1) 将 RTS 格局和抵达此格局的 step 产生的局部事件作为 EFSM 的状态;(2) 将 RTS 的可能 steps 作为 EFSM 的迁移.称这样得到的 EFSM 是该 RTS 的一个范式规约(NFS).

定义 6(范式规约)　RTS $Z = (S, C_0, \Pi, V, \Theta, R, T)$ 的范式规约 NFS(Z) 是如下 EFSM:

$$NFS(Z) = (Config \times 2^{\Pi}, (C_0, \emptyset), [[R]] \times 2^{\Pi}, 2^{\Pi}, V, \Theta, \delta)$$

其中 $((C, E), i, o, g, a, (C', E')) \in \delta$ 当且仅当

® 如果 $E \neq \emptyset$,则 i 是一个内部激励;否则, i 是在 RTS 中出现的一个外部激励;

® $E' = o \cap \Pi_L$;

® 存在一组迁移 $\tau \subseteq T$ 满足:

- τ 中每个迁移被 i 激发;
- τ 中每个迁移的源状态包含在 C 中,即 $\forall t \in \tau.\ (source(t) \in C)$;
- τ 中迁移无冲突;
- $g = \wedge_{t\in\tau} guard(t)$;
- $a = \cup_{t\in\tau} assignments(t)$;
- $C' = (C - \cup_{t\in\tau} Exits(t)) \cup (\cup_{t\in\tau} Enters(t))$;
- $o = \cup_{t\in\tau} generated(t)$.

图 3 是按照上述转换方法得到的 CVM 例子的 EFSM,其中阴影状态对应 RTS 稳态.为了简洁起见,图 3 的 EFSM 状态中只标出了 RTS 基本状态.为了增强可读性,图 3 中标出了迁移 guards 和赋值语句.

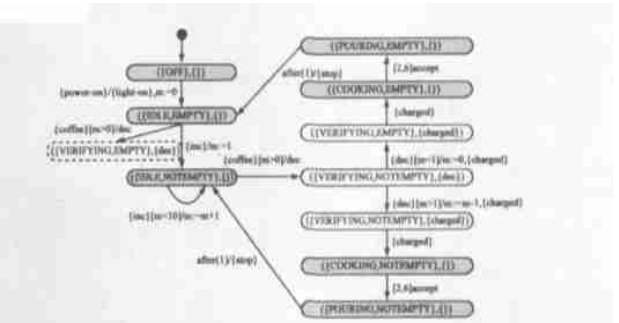

图 3 自助咖啡机 RTS 规约对应的 EFSM

定理 1　令 $Z = (S, C_0, \Pi, V, \Theta, R, T)$ 是一个 RTS,NFS(Z)是其范式规约,则 B(Z) ⊆B(NFS(Z)).

定理 1 说明,作为范式规约的 *EFSM* 保留了 *RTS* 的所有行为.

2.3　EFSM 修剪

我们将 RTS 转换为 EFSM 时,EFSM 迁移对应无冲突(部分)使能迁移集,而不是最大无冲突(完全)使能迁移集,因此, EFSM 不仅包含了 RTS 全部行为,而且还包含了额外行为.这些额外行为使 EFSM 更复杂,更难以测试,而且会导致虚假错误(例如,若对额外行为进行测试但未获通过,那么并不表明系统中有错误),因此必须将它们剔除.称这个过程为 EFSM 修剪.

为了进行 EFSM 修剪,必须知道迁移 guard 的可满足性.然而,EFSM 状态只包含格局信息和局部事件信息,不包含变量取值信息.为此,我们对 EFSM 进行变量取值集合标注,标出 EFSM 每个状态下各变量的所有可能值.

假设 EFSM 状态集合是 Q,变量集合是 V.根据前面的假

设,在 EFSM 每一个状态 $q=(C,E)\in Q$ 下,系统中每一变量 $v\in V$ 都可以对应一个有穷的允许取值集合 *Allowable*(q,v). 在作标注之前,先建立一个栈 *stack* 用来存放 EFSM 状态. 图 4 是变量取值集合标注算法. 算法的基本思想是:用栈对 EFSM (视为有向图)进行深度优先搜索. 如果某个迁移使得其目标状态下某个变量的取值范围扩大,则将这种变化一直"传播"下去(第 19 行);否则,不再考虑该目标状态的次态(第 17 行). 一个状态处理完毕之后,接下来先处理那些迁移 *guard* 能被满足的次态,再处理其他次态(第 7 行,第 19 行). 由于每个变量的取值范围都是有穷的,因此算法必然终止.

输入: 由 RTS Z 得到的范式规约 EFSM.
输出: 带标注的 EFSM.
算法:

1. ForEach $q_i\in Q$ and $v_j\in V$ DO
2. Allowable $(q_i,v_j):=\varnothing$;
3. ENDFOR
4. ForEach $v_j\in V$ DO
5. *Allowable*$(q_0,v_j):=\{\Theta(v_j)\}$;/ * *Initialization* */
6. ENDFOR
7. 对 q_0 诸次态,依迁移 guard 目前不满足、可满足的顺序进栈;
8. IF EMPTY(*stack*) THEN
9. 标记过程结束;
10. ELSE
11. $q:=$POP(*stack*);
12. IF 从前一状态 q 到此状态 q 的迁移 $e[g]/a$ 中,g 可被 q' 处的变量取值集合满足 THEN
13. ForEach $v\in V$ DO
14. 根据 $e[g]/a$,修改状态 q 的变量取值集合;
15. ENDFOR
16. IF 状态 q 曾被标记过,且从 q 到 q 的迁移未能扩大集合 *Allowable*(q',v), $v\in V$ THEN
17. 转 8;
18. ENDIF
19. 对 q 诸次态,依迁移 *guard* 不满足、可满足顺序进栈. 转 8;
20. ELSE
21. 将 q 标记为"*unreachable*",转 8.
22. ENDIF
23. ENDIF

图 4 变量取值集合标注算法

令 $n=\max_{q\in Q,v\in V}|Allowable(q,v)|$. 图 4 算法复杂性为 $O(|Q|^2\times|V|\cdot n^2)$. 复杂性分析见[4]. 标注后 CVM 例子的 EFSM 如图 5,图中各状态右下角集合即 *Allowable*(q,v).

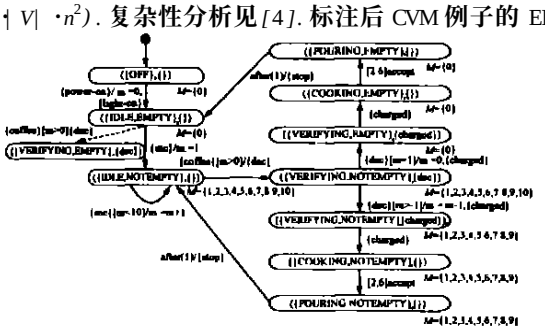

图 5 标注后的自助咖啡机 EFSM

EFSM 修剪的基本思想是:(1)将所有未对应 *RTS* 最大无冲突使能迁移集的 *EFSM* 迁移剔除. 如果在某个迁移的源状态,系统变量的所有可能取值都无法满足该迁移 *guard*,则称该迁移不可达. 例如,图 3 中从$(\{IDLE,EMPTY\},\varnothing)$出发的迁移$\{coffee\}[m>0]/\{dec\}$是不可达迁移. 不可达迁移的目标状态是不可达状态. 例如,图 3 中$(\{VERIFYING,EMPTY\},\{dec\})$是不可达状态. *EFSM* 中所有不可达迁移和不可达状态都应剔除. 具体而言,如果 *EFSM* 的某个状态未进行变量取值集合标注,那么该状态是不可达状态. 所有进入、离开不可达状态的迁移都是不可达迁移. 经过这种修剪后,图 5 中虚线框状态和虚线迁移将被去掉. (2)将所有未对应 *RTS* 最大无冲突使能迁移集的 *EFSM* 迁移剔除. 相应的状态和迁移也应剔除.

经过 *EFSM* 修剪后,得到的 *EFSM* 恰好包含 *RTS* 全部行为. 因此,如果通过测试能断定 *EFSM* 满足某性质(即 *EFSM* 中任何一个运行都满足某性质),那么相应 *RTS* 也满足该性质.

3 实时性质描述

反应式实时系统的许多性质可以用命题线性时间逻辑(PLTL)[5]公式描述. 本文允许 PLTL 公式包含三类原子命题:(1)状态命题 $@s_i$,其中 s_i 是 RTS 状态名,当 RTS 当前活跃格局包含 s_i 时命题为真;(2)关系命题 $v_i\ rop\ c_i$,其中 *rop* 是关系运算符(例如 =, >, ≤等),当系统变量 v_i 和常量 c_i 满足 *rop* 时命题为真;(3)事件命题$[l,u]e_I$,$[l,u]e_I,e_O$,其中$[l,u]$是时间区间,e_I、e_O 分别是输入事件、输出事件,当系统以它们为外部激励或者输出事件时命题为真. PLTL 公式归纳定义如下:

$$\Phi:=p\mid\neg\Phi_1\mid\Phi_1\wedge\Phi_2\mid\Phi_1\ U_{[l,u]}\Phi_2$$

其中 p 是原子命题,$[l,u]$是时间区间,其左右端点皆为非负有理数,$U_{[l,u]}$是受限时态算子"*Until*".

PLTL 公式语义可用时间观察序列[5]解释. 公式 $\Phi_1\ U_{[l,u]}\Phi_2$ 的直观含义为:从当前时刻算起,命题 Φ_2 在时间 l 之后、时间 u 之前必然成立;在 Φ_2 成立之前(不含该时刻),命题 Φ_1 一直成立.

上述 PLTL 逻辑如果不予以适当限制的话,会出现许多缺乏实际意义的公式,例如$(m=1)\mid\rightarrow coffee$,其中"$\mid\rightarrow$"表示如果在当前稳态下$(m=1)$成立,则下一稳态下产生 *coffee* 事件. 公式缺乏实际意义的原因是因为它没有遵循一定的描述模式. 一般来说,实时系统的反应式行为可以用"场景→外部激励→观察"模式加以刻画. 场景是 RTS 当前稳定状况的快照,包括当前活跃状态、系统变量取值、刚刚产生的输出事件等信息;外部激励要么是一个纯时间延迟,要么是一个纯输入事件,要么是一个延迟输入事件;观察(observation)描述了系统(从某一场景出发)(在某些外部激励作用下)在一定时间范围内所能到达的场景. 我们对场景、外部激励、观察分别定义如下:

定义 7(场景) 若 p 是状态命题、关系命题、输出事件,则 p 是场景;

若 p,q 是场景,则 $\neg p,p\wedge q$ 是场景.

定义 8(外部激励) 若 p 是事件命题且 p 不是输出事

件,则 p 是外部激励.

定义 9(观察) 若 p, q 是场景,则 $p\ U_{[l,u]}\ q$ 是观察;

若 p 是外部激励,q 是场景或观察,则 $(p \rightarrow q)$ 是观察;

若 p, q 是观察,则 $\neg p$, $p \wedge q$, $p\ U^{[l,u]}\ q$ 是观察.

定义 10(受限 PLTL 公式) 受限 PLTL 公式递归定义如下:

$f ::= [$[场景 $\rightarrow$]外部激励 $\rightarrow$]观察$\mid \neg f \mid f_1 \wedge f_2 \mid f_1 U_{[l,u]} f_2$.

在命题连接词 $\neg$ 和 $\wedge$ 基础上,可定义 $\vee$;在受限时态算子 $U_{[l,u]}$ 基础上,可以定义 $F_{[l,u]}$ 和 $G_{[l,u]}$,即"*Future*"和"*Globally*".引入这些额外命题连接词和受限时态算子后,将能很方便地描述许多实时性质.例如, $F($ @EMPTY $\wedge$ @COOKING$)$, $([3,4]\ accept \mid \neg F_{[1,2]}($ @IDLE$))$, $G($ @NOTEMPTY $\wedge (m = 1) \mid \rightarrow (coffee \mid \neg F_{[3,4]}\ accept \mid \neg F_{[1,1]}($ @IDLE$))))$ 都是受限 PLTL 公式.

4 面向性质的实时测试生成

进行测试之前,我们假设系统模型 M 满足待测性质 P.此假设是必要的,因为如果 M 不满足 P,那么要么模型 M 中存在设计错误,要么 P 并不是我们真正希望测试的性质.此假设也是可行的,因为目前已存在诸如 UPPAAL[6] 等实时模型检验工具,我们可以利用它们来检查 M 是否满足 P.

将 RTS 转换成 EFSM 并修剪后,面向性质的测试生成按如下步骤进行:

Step1　根据待测受限 PLTL 性质,构造 EFSM 路径表达式;

Step2　由路径表达式得到有穷条、有穷长实时测试序列;

Step3　判断生成的测试序列的可行性.

4.1 路径表达式构造

为了测试 EFSM 是否满足指定的受限 PLTL 性质,必须列出使性质场景部分(如果存在的话)为真的所有路径.我们用正规表达式(或其集合)表示这些路径.

先考察状态命题 $@s_i$,其中 s_i 是 RTS 状态.为命题 $@s_i$ 构造路径表达式,实质上就是寻找从 EFSM 初始状态到所有与包含 s_i 的 RTS 格局相对应的 EFSM 稳定状态的所有路径.如果将 EFSM 视为一个 FSM,将求得的 EFSM 稳态视为 FSM 的接收状态,那么上述问题转化为寻找该 FSM 所接收的语言的问题.给定一个 FSM,寻找与该 FSM 所接收语言相对应的正规表达式的算法是已知的[7].求得的正规表达式长度之和最大为 $3^{|K|}$,其中 K 是 FSM 状态集.图 6 是状态命题路径表达式构造算法.虽然该算法呈指数复杂性,但可以在很大程度上予以改进.如果一条 EFSM 路径中,任何状态最多出现一次,则称该路径为基本路径.针对图 6 算法的一种优化方法是:将 EFSM 中所有非稳态和基本路径中间的稳态隐藏起来,将被隐藏状态和相关迁移合并成子路径.精简后的 EFSM 含有较少状态,因此该算法所能处理的问题规模将明显提高.

对形如 $v_i\ rop\ c_i$ 的关系命题,先寻找 EFSM 中那些其变量取值集合能够满足 $v_i\ rop\ c_i$ 的稳态,然后用图 6 算法求出其路径表达式.如果路径表达式中某个迁移执行一次后会修改变量 v_i 值,那么称该迁移为变量 v_i 的影响迁移.目前我们只考虑赋值语句部分包含常数赋值(如 $v := 3$)和简单算术运算(如 $v := v + 5$)的影响迁移.假设 v_i 的路径表达式中有 m 个影响迁移,其影响值分别为 $k_1, k_2, \cdots, k_m$,那么构造线性方程或者不等式 $(k_1 \cdot n_1 + k_2 \cdot n_2 + \cdots + k_m \cdot n_m)\ rop\ c_i$,寻找其整数解 $(n_1, n_2, \cdots, n_m)$ 的集合,按照这些整数解构造关系命题的路径,使得各影响迁移分别出现 $n_1, n_2, \cdots, n_m$ 次.

输入:修剪后的 EFSM(其状态集为 Q);RTS 状态命题 $@s_i$.
输出:满足所给状态命题的 EFSM 路径表达式的集合 PathExp.
算法:

```
1.  PathExp := Ø; /* initialization */
2.  ForEach qi ∈Q DO /* qi is a tuple of (Config, (local) Event) */
3.    IF (si ∈qi.Config) ∧(qi.Event = Ø) THEN
4.      将 qi 标记为"接收状态"; /* view EFSM as an FSM */
5.    ENDIF
6.  ENDFOR
7.  求出所有与该 FSM 相对应的正规表达式,将其放入 PathExp 中.
```

图 6　状态命题的路径表达式构造算法

第 3 节中,关系命题被定义为变量与常量相比较的形式.实际上,它也可以是多个变量(和/或常量)之间的比较,例如 $v_1 >= v_2 + 3$.为它们构造路径表达式的方法与上述方法类似.

对于输出事件命题,我们先找出产生该输出事件的迁移(可能有多个),然后找出从迁移目标状态向后数的第一个稳态,接着生成从 EFSM 初态到这些稳态的路径表达式.

以上是原子命题的路径表达式构造算法.对于用户按照定义 10 构造出的一般形式的受限 PLTL 公式 ϕ,我们先对 ϕ 进行扫描,按照受限 PLTL 公式构成规则识别出公式的场景、外部激励、观察部分.识别完毕之后,对 ϕ 进行处理,如图 7 所示.

- 若 ϕ 是原子命题,则按照上述方法生成 ϕ 的路径表达式;
- 若 ϕ 形如 $(\phi_1 \rightarrow \phi_2)$,则
  - 若 ϕ_1 是外部激励,则
    - 若 ϕ_1 是公式的第一个原子命题(即尚未生成任何路径表达式),则生成 EFSM 初态至所有稳态的路径表达式,将 ϕ_1 添加到表达式末尾,继续分析 ϕ_2;
    - 否则
      - 将 ϕ_1 添加到已生成的路径表达式末尾,继续分析 ϕ_2;
  - 否则 /* ϕ_1 一定是场景,ϕ_2 一定是带外部激励的观察 */
    - 寻找满足 ϕ_1 的路径,继续分析 ϕ_2;
- 若 ϕ 形如 $\neg \phi_1$,则寻找不满足 ϕ_1 的路径;
- 若 ϕ 形如 $\phi_1 \wedge \phi_2$,则寻找满足 ϕ_1 且满足 ϕ_2 的路径;
- 若 ϕ 形如 $\phi_1\ U_{[l,u]}\ \phi_2$,则寻找从进入当前稳态时刻算起,$l$ 在时间 l 之后、时间 u 之前有某个时刻的稳态能满足 ϕ_2,且从当前时刻(含)到该时刻(不含)之间的任一时刻都满足 ϕ_1 的路径.

图 7　性质指导的测试生成方法

如果性质的场景部分无法满足,那么该性质就是平凡性质,例如 $G((\ @\text{NOTEMPTY}) \wedge (m = 13) \rightarrow (coffee \rightarrow (\ @\text{COOKING})))$.平凡性质显然能被满足.但是面向平凡性质生成测试序列是没有意义的.因此在进行面向性质测试之前,我们先用模型检验工具对性质前件进行验证.如果不成立,则报告该性

质是平凡性质.

4.2 测试序列选择

反应式实时系统测试选择面临的两个主要问题是:(1)如何根据路径表达式选择有穷多个有穷长的测试序列?(2)如何为测试序列中的时间区间选取代表性的时间点?

为了生成带时间的测试序列,我们提出了两阶段的测试选择方法.在第一阶段,将 EFSM 路径中的纯时间延迟和延迟输入事件视为普通输入事件,采用一定的覆盖准则,所得到的测试集里每个测试序列中的时间延迟都以区间形式存在,称该测试集为抽象测试集.在第二阶段,对抽象测试集里每个测试序列运用时间覆盖准则,为时间区间选择代表性的时间点,所得到的测试集称为实时测试集.

在测试选择的第一阶段,传统的覆盖准则(例如状态覆盖/迁移覆盖/迁移对覆盖/定义-使用覆盖等)不再合适,因为它们都是用来度量待测实时系统全部行为的测试充分性的,而不是度量用户感兴趣的系统行为的测试深入程度的.本节提出一组度量面向性质测试深入程度的覆盖准则.

路径表达式之所以对应无穷条无穷长的测试序列,其原因是路径中存在回路.由于回路可能嵌套,所以我们只考虑不出现重复状态的回路,即基本回路.基本回路对变量有 3 类效应:(1)使变量值增大,如图 5 中状态({IDLE,NOTEMPTY},Ø)的 $inc[m<10]$ 自圈;(2)使变量值减小,如图 5 中状态({IDLE,NOTEMPTY},Ø)的回路 $\{coffee[m>0]\},\{dec[m>1]\},\{charged\},\{[2,6]accept\},\{after(1)\}$;(3)为变量赋值,如图 5 中状态({IDLE,NOTEMPTY},Ø)的回路 $\{coffee[m>0]\},\{dec[m=1]\},\{charged\},\{[2,6]accept\},\{after(1)\},\{inc\}$.如果基本回路的执行会改变某个变量的值,则称之为基本影响回路.沿回路执行次数越多,表明测试越深入.

定义 11(基本路径覆盖) 如果路径表达式中每个基本路径都在某个测试序列中出现,那么称该测试集满足基本路径覆盖.

定义 12(k-度基本影响回路覆盖) 如果路径表达式中每个基本影响回路都在某个测试序列中分别出现 i 次($i=0,1,2,\cdots,k$),那么称该测试集满足 k-度基本影响回路覆盖.

定义 13(k-度基本回路覆盖) 如果路径表达式中每个基本回路都在某个测试序列中分别出现 i 次($i=0,1,2,\cdots,k$),那么称该测试集满足 k-度基本回路覆盖.

定义 12 的准则包含定义 11 的准则,而定义 13 的准则包含定义 12 的准则.定义 12 考察基本影响回路.定义 13 进一步考虑一般的基本回路.用户可以根据资源状况和测试深度要求来选择适当的 k 值.

在测试选择的第二阶段,我们定义以下时间覆盖准则:

定义 14(网格覆盖) 假设选定的网格粒度为 δ.如果对于抽象测试集里每一个包含 $[l,u]$ 的测试序列,实时测试集里都有若干个分别包含 $after(l)$,$after(l+\delta)$,$after(l+2\delta)$,$\cdots$,$after(l+k*\delta)$ 的测试序列与之对应,其中 $l+k*\delta \le u < l+(k+1)*\delta$,则称实时测试集满足粒度为 δ 的网格覆盖.

定义 15(时间边界覆盖) 如果对于抽象测试集里每一个包含 $[l,u]$ 的测试序列,实时测试集里都有分别包含 $after(l)$,$after(u)$ 的测试序列与之对应,则称实时测试集满足时间边界覆盖.

4.3 测试序列可行性分析

如果一个测试序列在 EFSM 上执行时,某个迁移 guard 未能满足,则称该测试序列不可行.经过测试选择所得到的测试集里可能存在不可行测试序列.例如,图 5 中测试序列 $\{power\text{-}on\},\{inc\},\{coffee[m>0]\},\{dec[m>1]\}$ 是不可行的,因为执行迁移 $\{dec[m>1]\}$ 之前 m 值为 1,迁移 guard 不满足.

基于模型生成的测试序列需精化后方能用于 SUT.此精化过程需要一定代价.既然上述序列对于 EFSM 模型不可行,那么它们对于 SUT 也无意义,因此不必浪费时间和精力来精化这些测试序列.

为了去掉测试集里的不可行测试序列,我们进行测试序列可行性分析.判断某个测试序列对于 EFSM 是否可行,只需将该测试序列应用于 EFSM,随着 EFSM 的执行来判断.

测试序列集里可能有某些测试序列是另一些测试序列的前缀.如果某个测试序列不可行,那么所有以该序列为前缀的测试序列都不可行.

5 举例

已知自助咖啡机 RTS 规约如图 2 所示,要求针对 PLTL 性质 $G(@NOTEMPTY \wedge (m=1) \rightarrow (coffee \rightarrow ([3,5]accept \rightarrow F_{[1,2]}(@IDLE))))$ 生成测试序列.

图 2 中自助咖啡机 RTS 转换成 EFSM 后,结果如图 3 所示,修剪后的 EFSM 如图 5 所示.

待测性质中,$@NOTEMPTY \wedge (m=1)$ 是"场景",$coffee$ 和 $[3,5]accept$ 是"外部激励",$F_{[1,2]}(@IDLE)$ 是"观察".

满足场景"$@NOTEMPTY \wedge (m=1)$"的 EFSM 稳态是({IDLE,NOTEMPTY},Ø).因此,该场景所对应的路径就是从 EFSM 初态到上述稳态的路径,并且要求路径中所出现的 m 的影响迁移的影响值之和为 1.为此稳态构造路径表达式之前,我们采用 4.1 节的"状态隐藏"技术,得到的精简后 EFSM 如图 8 所示.

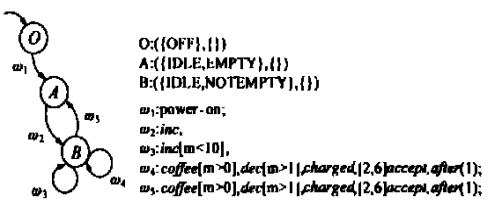

图 8 CVM 例子精简后的 EFSM

根据 4.1 节方法,图 8 中状态 B(即稳态({IDLE,NOTEMPTY},Ø))的路径表达式为:

$$\omega_1(\omega_2(\omega_3|\omega_4)^*\omega_5)^*\omega_2(\omega_3|\omega_4|\omega_5\omega_2)^*.$$

该路径表达式中变量 m 的基本影响回路是 $c_1=\omega_3=\{inc[m<10]\}$($m$ 增 1),$c_2=\omega_4=\{coffee[m>0]\},\{dec[m>1]\},\{charged\},\{[2,6]accept\},\{after(1)\}$($m$ 减 1).因此,构造线性方程 $1*n_1+(-1)^*n_2=1$,其中 n_1、n_2 是 c_1、c_2 的出现次数.

下一步,假定选取"2-度影响回路覆盖",则满足场景 $@NOTEMPTY \wedge (m=1)$ 的路径共有 9 条,例如

® $\omega_1\omega_2$;

® $\omega_1\omega_2\omega_3\omega_4$;

® $\omega_1\omega_2\omega_4\omega_3$;等等.

测试序列可行性分析有助于剔除一些不可行路径,例如 $\omega_1\omega_2\omega_4\omega_3$. 剔除这些不可行路径,最终得到的抽象测试集包含以下 4 条测试序列:

® $\omega_1\omega_2$;

® $\omega_1\omega_2\omega_3\omega_4$;

® $\omega_1\omega_2\omega_3\omega_3\omega_4\omega_4$;

® $\omega_1\omega_2\omega_3\omega_4\omega_3\omega_4$.

在测试选择的第二阶段,假定选取粒度为 1 个时间单位的网格覆盖,则抽象测试集里各抽象测试序列所对应的实时测试序列个数为 3 + 5 ×3 + 5 ×5 ×3 + 5 ×5 ×3 = 168. 例如,第一个抽象测试序列所对应的 3 个实时测试序列为:

®{ power-on} ,{ inc} ,{ coffee} ,{ after(3) accept};

®{ power-on} ,{ inc} ,{ coffee} ,{ after(4) accept};

®{ power-on} ,{ inc} ,{ coffee} ,{ after(5) accept}.

在同样覆盖准则下,针对其它若干性质得到的测试结果如表 1 所示. 可以看出,待测性质越强(越具体),得到的测试集越小. 从表中还可以看出,在相同测试深度下,面向性质测试比非面向性质测试(即性质为"None")需要少得多的开销.

表 1　实时系统面向性质测试所需的测试集大小

待测性质	抽象测试集	实时测试集
$F($ @NOTEMPTY $\wedge(m=1)\mid\rightarrow($ coffee$\mid\rightarrow$ ([3,5] accept$\mid\rightarrow F_{[1,2]}($ @IDLE$))))$	1	3
$G($ @NOTEMPTY $\wedge(m=1)\mid\rightarrow($ coffee$\mid\rightarrow$ ([3,5] accept$\mid\rightarrow F_{[1,2]}($ @IDLE$))))$	4	168
$G($ @NOTEMPTY $\wedge(m\leq 2)\mid\rightarrow($ coffee$\mid\rightarrow$ ([3,5] accept$\mid\rightarrow F_{[1,2]}($ @IDLE$))))$	7	201
$G($ @NOTEMPTY$\mid\rightarrow$ (coffee$\mid\rightarrow$ ([3,5] accept$\mid\rightarrow F_{[1,2]}($ @IDLE$))))$	8	204
(None)	40	935

6　相关工作

为了描述 non-trivial 时间约束,Kesten 等[8]提出了 Timed Statecharts 概念. 在他们的模型中,迁移分为两类:立即迁移和时间迁移. 前者形如 $e[g]/a$,需要有激发事件 e,迁移瞬间发生;后者形如 $([g]\ for\ [l,u])/a$,不需要激发事件,该迁移与一个最小等待时间 l 和一个最大等待时间 u 关联. 文献[9]用未作实时扩展的 UML Statecharts 建模,它采用额外状态、额外迁移、额外判断来辅助描述时间区间上限. 该方法在建模自然性和模型复杂性方面需要改进. 文献[10]对 UML 作了实时扩展,它允许单个迁移由多个并发外部事件共同激发(形如 $e_1\wedge e_2\wedge\cdots\wedge e_n$),允许外部事件和时间区间共同构成带时间激励. 该方法的后一种扩展与本文类似,但前一种扩展(即多个并发外部事件共同激发迁移)与 UML Statecharts 单事件处理假设不一致,导致了语义冲突. 本文模型将迁移标记统一为 $[l,u]e_1\vee e_2\vee\cdots\vee e_n[g]/a$ 形式,在模型描述能力足够的情况下使得对问题建模更加自然.

已有的面向性质测试方法大多是基于程序测试,例如[11]. 这些方法难以用于基于规约的测试,并且待测性质的描述受到诸多限制. 虽然[12]的方法可以对系统规约进行测试,但它所采用的是较低层次系统规约,不宜描述较大规模的、有并发行为的、带数据变量的反应式系统. [4]面向高层规约进行测试,但它未考虑实时性质. 在面向性质的实时测试方面,Clarke 等[13]用 ACSR 进程代数描述待测系统,用输入/输出事件间的最小/最大允许延迟描述待测时间约束,采用分域测试技术根据给定约束生成测试用例. 此外,Fouchal 等[14]用 test purpose 描述待测实时性质. 但 test purpose 只是一个无环图,其描述能力有限,不能描述含有循环的系统行为.

实时模型检验技术也能用来判断系统模型 M 是否满足指定性质 P,但它不能判断 SUT 是否满足 P. 将模型检验工具作为引擎的实时测试研究也已展开[15],但它们还不是针对性质的测试.

7　结论

本文提出了一种面向性质的实时系统测试方法,该方法能针对用户给出的实时性质,从基于 UML Statecharts 实时扩展的模型生成测试序列. 由于明确指定了 RTS 模型语义,给出了从 RTS 到 EFSM 的转换规则,因此本文方法可以自动化. 由于进行了 EFSM 测试序列可行性分析,因此本文方法只生成可执行测试序列.

下一步研究方向包括:(1)放松 RTS 模型限制;(2)考虑含有参数化事件的开放系统;(3)提供性质模式,辅助非专家用户构造待测实时性质;(4)混成系统和多对象实时系统面向性质的测试;(5)进一步的经验研究与方法性能优化.

致谢　论文撰写过程中与文艳军同学进行过深入讨论,在此表示感谢. 也感谢李仁见同学为工具实现所作的工作.

参考文献:

[1] OMG. Unified Modeling Language: Superstructure[S]. version 2.0. Object Management Group, 2003.

[2] H S Hong, Y G Kim, et al. A test sequence selection method for statecharts[J]. Journal of Software Testing, Verification, and Reliability, 2000, 10(4): 203 - 227.

[3] D Harel, A Naamad. The STATEMATE semantics of statecharts[J]. ACM Transactions on Software Engineering and Methodology, 1996, 5(4): 293 - 333.

[4] S Li, J Wang, Z C Qi. Property-oriented test generation from UML statecharts[A]. Proc. 19th IEEE Int Conf On Automated Software Engineering[C]. Linz, Austria, 2004, 9: 122 - 131.

[5] R Alur, T A Henzinger. Logics and models of real time: a survey[A]. In real time: theory in practice (LNCS600)[C]. Springer-Verlag, 1992. 74 - 106.

[6] K G Larsen, P Pettersson, W Yi. UPPAAL in a nutshell[J]. Int J Software Tools for Technology Transfer, 1997, 1(1/2): 134 - 152.

[7] H R Lewis, C H Papadimitriou. Elements of the Theory of Computation (2nd ed.)[M]. Prentice Hall, Upper Saddle River, NJ, 1998.

[8] Y Kesten, A Pnueli. Timed and hybrid statecharts and their textual representation[A]. J Vytopil, eds., Formal Techniques in Real-Time and Fault-Tolerant Systems (LNCS571) [C]. Springer-Verlag, 1992. 591 - 619.

[9] L Lavazza, G Quaroni, M Venturelli. Combining UML and formal notations for modeling real-time systems[A]. Proc 9th ACM SIGSOFT Int Symp on the Foundations of Software Engineering[C]. Vienna, Austria: September 2001. 9. 196 - 206.

[10] V D Bianco, L Lavazza, M Mauri. A formalization of UML statecharts for real-time software modeling[A]. Proc. 6th Biennial World Conference on Integrated Design Process Technology[C]. Pasadena, California, 2002.

[11] G Fink, M Bishop. Property based testing: A new approach to testing for assurance[J]. ACM SIGSOFT Software Engineering Notes, 1997, 22 (4): 74 - 80.

[12] J C Fernandez, L Mounier, C Pachon. Property oriented test case generation[A]. Proc. 3rd Int. Workshop on Formal Approaches to Software Testing[C]. LNCS 2931, Montreal, Canada: Springer-Verlag, 2003. 147 - 163.

[13] D Clarke, I Lee. Testing real-time constraints in a process algebraic setting[A]. Proc. 17th Int Conf on Software Engineering [C]. Seattle, Washington, 1995. 51 - 60.

[14] H Fouchal, E Petitjean, S Salva. An user-oriented testing of real time systems[A]. Proc. IEEE/ IEE Real-Time Embedded Systems Workshop [C]. London, UK, December 2001.

[15] M Mikucionis, B Nielsen, K G Larsen. Real-time system testing on-the-fly[A]. Proc. 15th Nordic Workshop on Programming Theory [C]. Turku, Finland, 2003. 10. 36 - 38.

作者简介:

李书浩 男, 1976 年出生于湖北汉川, 博士生, 主要从事软件质量保障与测试、软件开发方法学等方面的研究. E-mail: shuhao_li@yahoo.com.

王 戟 男, 1969 年出生于上海, 博士, 教授, 博士生导师, 主要从事高可信软件技术、软件工程、语义 Web 等方面的研究. E-mail: ji.wang@263.net.