一种基于活性顺序图的运行时验证研究

叶俊民[1] 张 坤[1] 叶竹君[2] 陈 盼[1] 陈 曙[1]
(华中师范大学计算机学院 武汉 430079)[1]
(华中师范大学国家物理学人才培养基地 武汉 430079)[2]

摘 要 运行时验证是一种轻量级的形式化验证方法,使用可视化的需求规约描述语言建模需求规约场景是运行时验证领域的研究热点。针对目前基于活性顺序图的运行时验证方法中容易产生冗余性质、二值语义的验证结果不准确、基于Maude工具引擎的重写逻辑验证算法效率较低等问题,提出一种基于活性顺序图的运行时验证的改进方法,以支持现有的运行时验证技术。实验表明,改进方法验证结果准确,且验证过程开销较小。

关键词 活性顺序图,线性时序逻辑,重写逻辑,运行时验证
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.8.029

Research of Runtime Verification Based on Live Sequence Chart

YE Jun-min[1] ZHANG Kun[1] YE Zhu-jun[2] CHEN Pan[1] CHEN Shu[1]
(School of Computer Science, Central China Normal University, Wuhan 430079, China)[1]
(Personnel Training Station of National Physics, Central China Normal University, Wuhan 430079, China)[2]

Abstract Runtime verification is a lightweight formal verification method. Using visual requirements description language to model requirements specification scene is a hotspot in runtime verification. For the problems that existing verification methods based on live sequence chart easily generate redundant properties, verification overhead is quite large, two true value semantics verification result is not accurate and the efficiency of existing verification algorithm of rewriting logic based on Maude is low, this paper proposed an improved runtime verification method based on live sequence chart to support existing runtime verification technologies. Experiments show that the result of improved method in this paper is accurate and verification overhead is low.

Keywords Live sequence chart, Linear temporal logic, Rewriting logic, Runtime verification

1 引言

某些运行时验证系统可支持软件工程师易懂的规约语言[1,2],如UML顺序图[3]是一种基于场景的语言,能够图形化地表示系统实例间的交互关系,基于场景的形式化验证可使用UML顺序图描述需求规约。UML顺序图包括消息顺序图(Message Sequence Chart, MSC)[4]、UML2.0顺序图(UML2.0 Sequence Diagram)[5]、活性顺序图(Live Sequence Chart)[6]及其它变种。但是UML顺序图缺乏形式化语义[7],活性顺序图则对顺序图进行了扩展以支持形式化语义,其应用领域为模型检测领域[8,9]。在运行时验证领域,应用较多的工具是活性顺序图[10],但已有的验证方法效率较低且因二值语义之故,其验证结果是不准确的。为此,本文研究基于活性顺序图描述的需求规约场景的运行时验证方法,验证系统的执行轨迹是否满足需求规约,且在发现性质违约时,能否尽早地报告可能的违约情况。

2 研究基础

2.1 线性时态逻辑

线性时序逻辑LTL由Pnueli等人提出,运行时验证常常使用LTL形式的性质来实现对需求规约的验证。

定义1(LTL语法) 令AP表示原子命题集合,$\Sigma=2^{AP}$表示有穷字母表。有穷轨迹上的LTL语法定义为:

$\psi::=\text{true}|a|\varphi_1\wedge\varphi_2|\neg\varphi|X\varphi|\varphi_1U\varphi_2$,其中$a\in AP$。

为了方便表述,本文将$\neg(\neg\varphi_1\wedge\neg\varphi_2)$,$\text{true}\,U\varphi$,$\neg F\neg\varphi$和$\neg\varphi\vee\psi$分别简写为$\varphi_1\vee\varphi_2$,$F\varphi$,$G\varphi$及$\varphi\rightarrow\psi$。

LTL公式中的运算符分为布尔运算符和时序运算符,布尔运算符包括:逻辑非$\neg$、逻辑与$\wedge$、逻辑或$\vee$及逻辑蕴含$\rightarrow$。时序运算符包括:G(Globally)表示总是为真,即公式在轨迹上的每个状态都为真;F(Future)表示将来为真,即公式在轨迹上的将来某个状态为真;$\varphi_1U\varphi_2$(Until)表示直到,即如果轨迹上某个状态S满足φ_2,且S之前的所有状态都满足φ_1,则

到稿日期:2015-06-13 返修日期:2015-10-23 本文受国家科技支撑计划项目(2015BAK33B00),中央高校基本科研业务费专项资金科研项目(CCNU15GF003),教育部人文社会科学研究规划基金(15YJA880095)资助。

叶俊民(1965-),男,博士,教授,CCF高级会员,主要研究方向为软件可靠性;张 坤(1988-),男,硕士生,主要研究方向为软件工程,E-mail:245101751@qq.com(通信作者);叶竹君(1995-),女,主要研究方向为形式化方法;陈 盼(1990-),女,硕士生,主要研究方向为形式化方法;陈 曙(1981-),男,博士,讲师,主要研究方向为软件工程。

该公式为真；X(Next)表示下一个，即轨迹上某一状态的下一个状态满足公式。对于所有原子命题 $a \in AP$，有：1)a 表示 LTL 公式；2)如果 φ、ψ 是 LTL 公式，则 $\varphi_1 \wedge \varphi_2$，$\varphi_1 \vee \varphi_2$，$\neg\varphi$，$X\varphi$，$\varphi_1 U \varphi_2$，$F\varphi$，$G\varphi$ 及 $\varphi \rightarrow \psi$ 都是 LTL 公式；3)所有 LTL 公式都可以由上述规则构造获得。

2.2 活动顺序图

采用活性顺序图描述的需求规约场景有两种性质，即图中消息时间的顺序性性质(ϕ 性质)和唯一性性质(χ 性质)。ϕ 性质是指对于任意两个消息 x_i 与 x_j，其顺序性性质 $x_i \prec x_j$ 用 LTL 公式表示为 $\phi_{x_i,x_j} = \neg x_j U x_i$，即 x_j 在 x_i 发生后才发生。χ 性质指对于任意两个消息 x_i 与 x_j，且 $x_i \not\prec x_j$，LTL 公式 $\chi_{x_i,x_j} = (\neg x_i \wedge \neg x_j) U (x_i \wedge X((\neg x_i \wedge \neg x_j) U x_i))$ 表示 x_i 在 x_j 发生之前发生了两次，其否定形式 $\neg\chi_{x_i,x_j}$ 则表示 x_i 在 x_j 发生之前没有发生两次。基于这两种 LTL 公式表示的性质，可以将活性顺序图描述的需求规约转换为 LTL 公式。

Kugler 等[11]提出将活性顺序图描述的规约转换为 LTL 公式，即对于一个 universal 模式的活性顺序图，对应的 LTL 公式 φ_c 如式(1)所示，其中 e，p 与 m 分别表示整个图、前置图和主图中对应于消息的事件。符号"⇒"将式(1)分为两部分，表示如果前置图的场景成功执行，则主图中的场景就一定会执行，其描述的是活性场景。

$$\varphi_c = G\left(\begin{array}{c}\bigwedge\limits_{p_i \prec p_j} \phi_{p_i,p_j} \\ \bigwedge\limits_{\forall p_i,m_j} \phi_{p_i,m_j} \\ \bigwedge\limits_{p_i \not\prec p_j} \neg\chi_{p_j,p_i}\end{array}\right) \Rightarrow \left(\begin{array}{c}\bigwedge\limits_{m_i \prec m_j} \phi_{m_i,m_j} \\ \bigwedge\limits_{m_j \text{ is max}} Fm_j \\ \bigwedge\limits_{\forall e_i,m_j} \neg\chi_{e_j,m_i}\end{array}\right) \tag{1}$$

式(1)的 $\bigwedge\limits_{p_i \prec p_j} \phi_{p_i,p_j}$ 性质描述了前置图内消息事件之间的偏序关系，$\bigwedge\limits_{m_i \prec m_j} \phi_{m_i,m_j}$ 性质描述了主图内消息事件之间的偏序关系；$\bigwedge\limits_{\forall p_i,m_j} \phi_{p_i,m_j}$ 性质保证了只有前置图的消息都发生后，主图中的消息才会发生，且 $\bigwedge\limits_{m_j \text{ is max}} Fm_j$ 性质保证了主图中的消息一定会发生；$\bigwedge\limits_{p_i \neg\prec p_j} \neg\chi_{p_j,p_i}$ 性质与 $\bigwedge\limits_{\forall e_i,m_j} \neg\chi_{e_j,m_i}$ 性质则描述了图中的唯一性性质。基于式(1)可以将活性顺序图描述的需求规约转换为 LTL 公式。

3 基于活性顺序图研究运行时验证

3.1 基于活性顺序图的运行时验证的框架

首先定义如图 1 所示的基于活性顺序图的运行时验证框架，该框架首先需要获取目标系统的执行轨迹，然后再获取活性顺序图描述的需求规约，基于此将系统执行轨迹信息和需求规约作为 Maude 工具引擎[12,13]的输入，以实现基于重写逻辑原理的运行时验证。

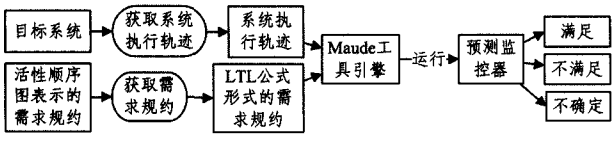

图 1 基于活性顺序图的运行时验证框架图

3.2 基于活性顺序图的运行时验证方法

由图 1 可知验证方法的步骤有：1)插装并获取系统的执行轨迹；2)将获取活性顺序图描述的需求规约转换为 LTL 公式；3)为了利用 LTL 公式进行运行时验证，需给出基于 LTL 三值可执行语义的公式重写运行时验证算法。

3.2.1 插装获取系统执行轨迹

为了通过代码插装的方式获取系统执行轨迹，可使用 AOP 的横切技术，即在目标系统源代码中设置若干横切关注点以实现代码插装。由用户定义观测点(插装点)，观测点就是横切关注点，观测点中的通知(Advice)代码可以捕获并存储系统执行轨迹信息。在系统运行期间，这些观测点可以获取目标系统执行参数、执行时间、工作状态等信息。本文设计的用户获取目标系统执行轨迹的切面类形式如图 2 所示。

```
@Aspect
public class AspectName {
@after(Pointcut)
public void afterAdvice() {
//通知代码
}
…
}
```

图 2 获取目标系统执行轨迹的切面类

图 2 中，@Aspect 注解定义了当前类是切面类，使用@after(Pointcut)后置通知获取系统执行轨迹，Pointcut 形式是"execution(＊…Method …＊(…))"，Pointcut 定义了切点，Pointcut 对系统中某种命名规范开头的方法进行拦截，在系统目标函数执行后执行通知代码，通过通知代码实现的具体功能来获取目标函数的执行轨迹信息。在完成关注点分离设计、切面设计和定义通知后，使用 Spring AOP 框架中的 applicationContext. xml 配置文件指定切面中的代理与作用范围等信息，就可以使用切面获取系统的执行轨迹。

3.2.2 活性顺序图到 LTL 的优化转换

式(1)描述的转换过程为活性顺序图 c 的每一个可能的执行入口都生成 LTL 公式，这种转换方法产生的 LTL 公式的规模是图 c 中消息事件规模的多项式复杂度，这里，LTL 公式的规模指 LTL 公式中子式的个数，因此直接使用式(1)转换所得的 LTL 公式进行验证的开销很大。由于活性顺序图描述的性质间具有传递性，因此可使用性质的传递性对式(1)转换后的结果进行化简。

具体做法是：1)化简顺序性性质对应的 LTL 公式的规模，首先可使用 U 模态算子描述顺序性性质，并通过性质间的传递性消除使用该算子描述时产生的冗余性质；其次对活性顺序图中部分消息所产生的多个并发后继消息进行单个顺序性性质化简；2)化简唯一性性质对应的 LTL 公式的规模。

针对 1)的化简要求，首先定义函数 $next(c)(e_i) = \{e_j \mid (e_i \lhd e_j) \wedge (e_j \notin \{T, -|, \perp\})\}$，其含义是：若给定消息 e_i，则可根据 e_i 的消息会话中由实例线引出的偏序关系返回 e_i 的直接后继消息的集合。引入函数 $\phi'_{x_i,N} = (\bigwedge\limits_{o \in N} \neg o) U x_i$，$\phi'$ 函数是式(1)中 ϕ 函数的修订版，当 N 满足集合内的消息都是并发消息时，ϕ' 函数可以将同一消息所引发的多重顺序性性质化简为单个顺序性性质。为此，可利用 $next(c)$ 函数与 ϕ' 函数化简顺序性性质，给定活性顺序图 c，式(2)成立：

$$\pi \vDash \bigwedge_{p_i \prec p_j} \phi_{p_i,p_j} \Leftrightarrow \pi \vDash \bigwedge_{p \in msg(c)} \phi'_{p,next(c)(p)} \tag{2}$$

针对 2)的化简需求，化简唯一性性质对应的 LTL 公式

的规模，将采用如下结论：给定 k 个消息组成的消息集合 N，其顺序是 $e_1 \prec e_2 \prec \cdots \prec e_k$，如果消息 e_i 与消息 $e_k(i<k)$ 之间没有重复发生，则任一中间消息 $e_j(i<j<k)$ 都不是重复消息，即：

$$\pi \vdash \bigwedge_{e_i \prec e_j} \neg\chi_{e_i,e_j} \Leftrightarrow \pi \vdash \bigwedge_{e_i \in N} \neg\chi_{e_i,e_k} \tag{3}$$

现说明活性顺序图到 LTL 转换过程的化简：利用式(2)与式(3)可化简通过式(1)转换所得的 LTL 公式的规模。化简后的 LTL 公式 φ_c' 如式(4)所示，化简后的式(4)与原等式(1)之间的主要不同是：式(4)使用 $next(c)$ 函数表示顺序性性质，使用 $\max_m(c)$ 函数表示唯一性性质。

$$\varphi_c' = G\left[\left(\begin{array}{c}\bigwedge\limits_{e\in msg_p(c)} \phi'_{e,next(c)(e)} \\ \bigwedge\limits_{e\in \max_p(c)} \phi'_{e,msg_m(c)} \\ \bigwedge\limits_{(e,p)\in msg_p(c)\times \max_p(c)} \neg\chi_{e,p}\end{array}\right) \Rightarrow \left(\begin{array}{c}\bigwedge\limits_{e\in msg_m(c)} \phi'_{e,next(c)(e)} \\ \bigwedge\limits_{(e,m)\in msg(c)\times \max_m(c)} \neg\chi_{e,m}\end{array}\right)\right] \tag{4}$$

分析可知，使用式(4)转换产生的 LTL 公式的规模是图中最大消息的规模的平方复杂度。式(4)中的转换过程描述顺序性性质事件的复杂度上界是 $|msg_p(c)|+|\max_p(c)|+|msg_m(c)|$，这与图中消息的规模呈线性关系，而式(1)描述该性质的复杂度是图中所有消息规模的平方复杂度。式(4)描述唯一性性质事件的复杂度上界是 $|msg_p(c)|\times|\max_p(c)|+|msg_m(c)|\times|\max(c)|$，对于前置图和主图，唯一性性质对应的事件规模依赖于图中最大消息的规模，而式(1)描述该性质的复杂度也是图中所有消息规模的平方复杂度。因此在平均情况下，相对式(1)将活性顺序图转换为 LTL 公式的转换过程而言，使用式(4)的转换过程将简化转换所得的 LTL 公式的规模。

3.2.3 基于重写逻辑的运行时验证算法

获取系统执行轨迹与 LTL 公式形式的需求规约后，即可进行基于重写逻辑的运行时验证，本文采用文献[17]中定义的 LTL 三值可执行语义进行运行时验证。

根据 LTL 三值可执行语义，满足关系“$\vdash$”的情况有 3 种可能取值：true，false，maybe。如果有穷轨迹 t 已经满足或违反给定的 LTL 公式，那么 t 的所有后继轨迹都满足或者违反该 LTL 公式，否则满足关系“$\vdash$”为 *maybe*，这表示 t 目前还无法提供足够的信息判定 LTL 公式成立与否。上述 LTL 三值语义是按照可执行的方式定义的，该语义可应用于基于重写逻辑的运行时验证。这里首先需要定义重写过程中的数据类型与基本操作，重写逻辑过程中的数据类型与基本操作如图 3 所示。

图 3 定义的数据类型包括：*Atom*，*state*，*state**，*Trace* 及 *Formula*。其中 *Atom*，*state* 和 *Trace* 分别表示原子命题、状态及轨迹。重写操作符 $\underline{A}\{\underline{B}\}$ *Formula*×*State*→*Formula* 是重写逻辑算法中的一个重要操作，其输入 A 表示一个 LTL 公式，B 表示一个状态，输出为一个新的 LTL 公式。以 $A\{A'S\}$ 为例，设“_”代表占位符，当操作符_{_}读入公式 A 及状态 $\{A'S\}$ 时，判定 $A==A'$ 是否成立，若 $A=A'$ 成立，则重写结果为 true，表示状态 $\{A'S\}$ 满足公式 A；否则去掉当前状态中的 A'，利用后续的 S 重写公式 A，即返回 $A\{S\}$。对于 $A\{S\}$，有 $A\{S\}$ 为真，当且仅当 S 满足 A。属性[prec]表示操作符_{_}的优先级，prec 数字越大其优先级越低。引入优先级是为了避免在重写逻辑过程中使用过多的括号，提高重写逻辑的简洁性。

```
Variable
    A,A′: Atom
E:State
E*:State*
φ,ψ: Formula
Operation
    _⊢_
Case
    ture{E*}:return true
    false{E*}:return false
    maybe{E*}:return maybe
A{nil}:return false
A{A′}:if A==A′ return true else return false
A{A′E}:if A==A′ return true else return A{E}
A{E*}:return A{E}*
```

图 3 重写逻辑过程中的数据类型和基本操作

利用重写操作符_{_}可以定义满足操作符“$\vdash$”，即令 E，T 为有穷轨迹序列，轨迹的起始状态为 E，后续轨迹为 T，如果 E,T 满足 φ，当且仅当后续轨迹 T 满足 $\varphi\{E\}$，即 $E,T\vdash\varphi \Leftrightarrow T\vdash\varphi\{E\}$。

“_{_}”操作符的计算过程是按照轨迹中的状态逐步进行的，基于文献[12]中轨迹状态消耗的思想，可构造基于 LTL 三值可执行语义的公式重写算法。该算法的输入是有穷长度轨迹 t 和性质公式 φ，输出是 t 是否满足 φ 的判定，其具体执行过程是：给定有穷长度轨迹 t 和性质公式 φ，在重写过程中，φ 与 t 的第一个状态结合，得到新的公式 φ' 和除去第一个状态的新轨迹 t'，然后 φ' 再与 t' 的第一状态结合得到新的公式和新的轨迹，根据 t 上的状态迭代进行该重写过程，直到 t 的最后一个状态与公式结合，最终给出验证结果。本文基于 LTL 三值可执行语义的公式重写改进算法如图 4 所示。

在图 4 中，第 1—7 行是算法的输入、输出与变量声明，第 8，9 行根据轨迹 t 上的状态迭代进行重写过程，第 11—16 行是操作符的声明，其中第 16 行根据 Maude 工具引擎的特点，引入“_{_}”操作符的缓存机制来提高公式重写效率。第 18，19 行在上文已经介绍，第 20—27 行是时序操作符的重写过程，其中 G,F,X 操作符重写过程是根据 G,F,X 的语义定义实现的。上述算法已经在 Maude 中实现，其时间复杂度是 $O(lenght(t))$，即算法的时间复杂度与执行轨迹的长度是线性相关的。

```
1. Function:LogicRewritingFun(t,φ)
2. Input: 有穷轨迹 t 和转换所得的公式 φ
3. T:Trace is Variable
4. E:State is Variable
5. E*:State* is Variable
6. φ,ψ:Formula is Variable
7. Output:有穷轨迹 t 是否满足性质公式 φ 的判定
8. While(length(t)>0){
9.        E←first(t)
10. Operation
11. _⊢_:Trance×Formula→ Formula[prec4]
12. G_:Formula→ Formula[prec2]
13. F_:Formula→ Formula[prec2]
```

14. X_:Formula→ Formula[prec2]
15. _U_: Formula×Formula→ Formula[prec3]
16. _{_} :Formula State* → Formula[memo prec 10]
17. Case
18. E,T⊨ φ:return T⊨ φ{E}
19. E⊨ φ:return φ{E}
20. Gφ{E}:if E is not terminal return Gφ∧φ{E}
21. else return φ{E* }∧maybe
22. Fφ{E}:if E is not terminal return Fφ∨φ{E}
23. else return φ{E* }∨maybe
24. φUψ(E):if E is not terminal return ψ{E}∨φ{E}∧φUψ
25. else return ψ{E* }∨φ{E* }∧maybe
26. Xφ{E}:if E is not terminal return φ
27. else return maybe
28. t←t—first(t)
29. }

图 4　基于 LTL 三值可执行语义的公式重写运行时验证算法

4　实验与分析

4.1　实验数据集选择

以欧洲列车控制系统 ETCS-2 中 RBC(无线闭塞中心)间的切换场景[15,16]为例,进行运行时验证实验研究。

4.2　实验样例设计

在 ETCS 系统的 2 级别 ETCS-2 中,无线阻塞中心(Radio Block Center,RBC)向列车提供移动授权。列车移动路线中包括若干个 RBC 监管区域,当列车到达两个 RBC 监管区域的边界时,会发生 RBC/RBC 切换。将当前的 RBC 称为切换 RBC(Handing over RBC,HOVRBC),将紧邻的 RBC 称为接受 RBC(Accepting RBC,ACCRBC)。RBC/RBC 切换过程主要通过两个 RBC 之间交换消息序列 NRBC 实现。NRBC 消息包含"提前通告"(preAnn)、"路由相关信息请求"(RRIReq)、"路由相关信息"(RRI)与"应答"(Ackn)等消息序列。NRBC 消息通过 GSM-R 通信系统实现交互。

在 RBC/RBC 切换过程中,NRBC 中的消息交互需要满足一定要求,只有 HOVRBC 检测到切换条件后,才能向 ACCRBC 发送 preaAnn 消息。在 HOVRBC 检测到切换条件并发送了 preaAnn 消息后,HOVRBC 向 ACCRBC 发送 RRIReq 消息,ACCRBC 在收到 RRIReq 消息后即向 HOVRBC 发送 RRI 消息,HOVRBC 在收到 RRI 消息后向 ACCRBC 发送一个 Ackn 消息,由此两个 RBC 之间就正确交换了 NRBC 消息。上述需求规约相应的活性顺序图如图 5 所示。

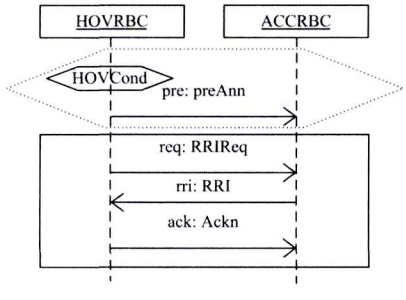

图 5　活性顺序图表示的 RBC/RBC 切换需求规约

在图 5 中,HOVCond 条件表示切换条件,pre 表示 preaAnn 消息,req 表示 RRIReq 消息,rri 表示 RRI 消息,ack 表示 Ackn 消息。该图描述了如下场景规约:对于每个成功的 RBC/RBC 切换过程,当列车到达两个 RBC 的边界时,两个 RBC 之间应该按正确的顺序交换 NRBC 消息。

4.3　验证过程

第一,插装获取系统执行轨迹。首先,工具将导入目标系统,当目标系统运行结束后,就获取了执行轨迹序列,同时将获取的执行轨迹序列显示在工具面板中,然后工具生成原子命题声明文件 verification. maude 并将其存储在本地,原子命题声明文件 verification. maude 用于声明轨迹中的状态变量,同时将行轨迹序列存储在本地文件 traceInput. txt 中并作为 Maude 工具引擎的输入,以便后续的运行时验证。插装获取系统执行轨迹过程如图 6 所示。

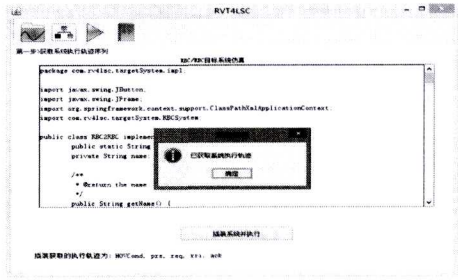

图 6　插装获取系统执行轨迹序列

第二,导入待验证的需求规约。根据活性顺序图描述的需求规约场景,该待验证的需求规约为使用式(3)转换所得的 LTL 公式。

第三,进行运行时验证。为此系统首先生成重写规则文件,再调用 Maude 工具引擎,该引擎根据重写规则产生预测监控器,进行运行时验证,一种验证结果如图 7 所示。

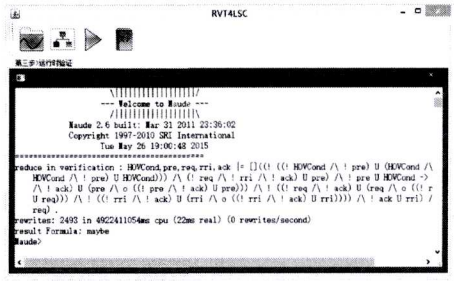

图 7　运行时验证

图 7 中的实验结果为"maybe",这是由于由式(3)转换所得的 LTL 公式含有"总是"时序逻辑运算符,验证工具分析出就目前的执行轨迹而言,没有发生需求违约,但对以后的执行情况还无法确定,判定结果为可能为真,因此验证结果为 maybe,这也体现出 LTL 三值语义相对于 LTL 二值语义的优势。

4.4　对比性实验

本文提出的方法是对传统验证方法的改进,相关的对比实验过程如下。

首先,选择对比对象,选择文献[10]中使用的活性顺序图转换为 LTL 公式,并将使用 LTL 公式进行验证的方法作为对比对象。然后,选取同一需求规约场景下不同执行轨迹作为案例。在不同执行轨迹下,首先使用式(1)描述的转换过程(化简前)所得的 LTL 公式进行验证,然后使用式(3)描述的转换过程(化简后)所得的 LTL 公式进行验证,对比化简前后验证过程中的重写次数、验证时间的差异。最后,进行对比实验。使用传统的方法与本文的转换方法进行不同案例的实验,利用 Maude 工具引擎进行验证,Maude 工具引擎在输出

验证结果的同时也给出了重写次数与验证时间等，通过验证过程所使用的重写次数、验证时间的差异可以分析转换的差异。

对比文献[11]中传统的转换方法和本文优化后转换方法所生成的 LTL 公式在验证过程所使用的重写次数、验证时间的差异，其中重写次数以千次为单位，重写时间以毫秒为单位。在同一场景中，分别选择不同长度的系统执行轨迹进行对比实验，对比结果分别如图 8 和图 9 所示。

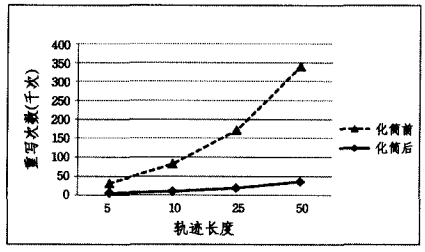

图 8　验证重写次数对比

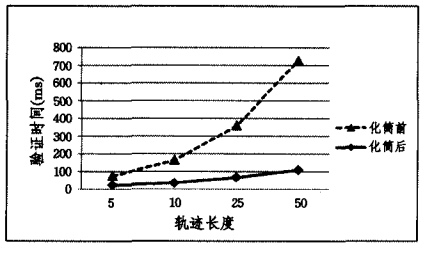

图 9　验证时间对比

图 8 和图 9 的实验结果表明，在同一需求规约场景的运行时验证中，使用式(3)生成的 LTL 公式进行运行时验证时，在重写次数、验证时间上均有比较明显的改善。多次对比性实验结果表明，对于消息并发度较小的实验场景而言，这两种转换方法的验证开销基本相当；对于消息并发度较大的实验场景而言，优化的转换方法能够明显减小转换的 LTL 公式的规模。本文改进方法验证的开销较小的原因是：首先，式(3)的转换过程将生成更小规模的 LTL 公式；其次，在公式重写过程中引入重写操作符的缓存机制将显著提高公式重写效率。

对比性实验表明：利用本文的转换方法验证的开销较小，传统的方法将活性顺序图转换为 LTL 公式的规模是图中消息规模的多项式复杂度，因此验证开销较大。利用本文的优化转换方法转换后的 LTL 公式的规模是图中最大消息的平方复杂度，因此对于一般场景而言，本方法降低了转换后的 LTL 公式的规模，减小了实际的运行时验证的开销。

5　相关工作对比

下面将从化简活性顺序图转换 LTL 公式的过程、图中性质的传递性转换和提高公式的重写效率等方面与相关文献工作进行对比。

5.1　化简活性顺序图转换成 LTL 公式的过程

文献[9]使用活性顺序图表示需求规约场景，并将之转换为 LTL 公式，然后化简了转换所得的 LTL 公式的规模，通过相关实例[7]验证其提出的活性顺序图到 LTL 的转换方法的可行性。而本文先化简了由活性顺序图转换所得的 LTL 公式的规模，并在保证化简前后轨迹对需求规约的满足关系的等价性前提下，研究运行时验证问题。

5.2　图中性质的传递性优化转换以降低验证开销

文献[10]研究了在欧洲列车控制系统 ETCS 的应用场景中使用活性顺序图进行运行时验证，提出了标准活性顺序图描述其研究特定场景中的正确性性质不具有充分性，因此引入充分前置图的概念，提出扩展的活性顺序图(eLSC)。但是文献[10]中转换所得的 LTL 公式的规模是活性顺序图中消息规模的多项式复杂度，且验证结果是二值语义的。本文针对该问题，使用活性顺序图中性质的传递性优化转换过程，并利用转换所得的 LTL 公式实现运行时验证，验证过程开销相对较小。

5.3　引入缓存机制提高公式重写效率

文献[14]实现了 LTL 三值语义的公式重写算法，在此基础上实现了系统某些 LTL 性质公式的运行时验证。与之相比，本文从活性顺序图转换的 LTL 公式相对复杂，为此基于轨迹状态消耗的思想，根据 Maude 引擎的特点，引入重写操作符的缓存机制来提高公式重写效率，由此可以更快地发现系统故障，从而最大限度地减少系统故障所带来的损失。

结束语　本文提出一种基于活性顺序图的运行时验证的改进方法。实验表明，提出的改进方法是可行的，且验证过程开销较小。同时，该方法能及时检测到系统故障，这为进一步的故障恢复提供了可能，验证结果有助于提高软件系统的可靠性。如何支持复杂活性顺序图描述的需求规约场景的运行时验证，是值得进一步研究的问题。

参 考 文 献

[1] Chen F, Rosu G. Java-MOP: A Monitoring Oriented Programming Environment for Java[M]// Tools and Algorithms for the Construction and Analysis of Systems. Springer, 2005: 546-550

[2] ONeil P M, Jin D Y, et al. Efficient Monitoring of Parametric Context-free Patterns[J]. Automated Software Engineering, 2010, 17(2): 149-180

[3] Bruegge B, Dutoit A H. Object-Oriented Software Engineering. Using UML, Patterns, and Java (Third Edition)[M]. Ye Junmin, et al. Beijing: Tsinghua University Press, 2011(in Chinese)
Bruegge B, Dutoit A H. 面向对象软件工程(第 3 版)[M]. 叶俊民，等译. 北京：清华大学出版社，2011

[4] Harel D, Thiagarajan P S. Message Sequence Charts[M]// UML for Real. Springer, US, 2003: 537-558

[5] Wang H Y, Wang L, Zhang J C, et al. Control Flow Analysis of UML2.0 Sequence Diagram Based on Message Semantic[J]. Journal of Jilin University, 2007, 45(4): 595-600

[6] Damm W, Harel D. LSCs: Breathing Life into Message Sequence Charts[J]. Formal Methods in System Design, 2001, 19(1): 45-80

[7] Li Wen-rui, Wang Zhi-jian, Zhang Peng-cheng. Formal Semantics of Universal Modal Sequence Diagram[J]. Journal of Software, 2011, 22(4): 659-675(in Chinese)
李雯睿，王志坚，张鹏程. 模态顺序图 uMSD 的形式化语义[J]. 软件学报，2011，22(4)：659-675

[8] Larsen K G, Li S. Scenario-based analysis and synthesis of real-time systems using Uppaal[C]// Proc 13th Conference on Design, Automation, and Test in Europe. 2010: 447-452

[9] Fu Ming-hui, Zhou Qing-lei, Zhang Bing. Transfer methods from live sequence charts to temporal logic[J]. Computer Engineering and Design, 2012, 33(9): 3437-3441(in Chinese)
付明慧，周清雷，张兵. 从活性顺序图到时态逻辑的转化方法[J]. 计算机工程与设计，2012，33(9)：3437-3441

(下转第 164 页)

[9] Hu Q P, Xie M, Ng S H. Software Reliability Predictions using Artificial Neural Networks[M]// Computational Intelligence in Reliability Engineering. Springer Berlin Heidelberg, 2007: 197-222

[10] Moura M C, Zio E, Lins I D, et al. Failure and reliability prediction by support vector machines regression of time series data [J]. Reliability Engineering & System Safety, 2011, 96(11): 1527-1534

[11] Jelinski Z, Moranda P. Software reliability research[J]. Statistical Computer Performance Evaluation. Freiberger W, Ed. Academic Press: New York, 1972: 465-484

[12] Goel A L, Okumoto K. Time dependent error detection rate model for software reliability and other performance measures [J]. IEEE Transactions on Reliability, 1979, 28(3): 206-211

[13] Zhao M. Change-point problems in software and hardware reliability[J]. Communications in Statistics-Theory and Methods, 1993, 22(3): 757-768

[14] Chang Y P. Estimation of parameters for nonhomogeneous Poisson process: Software reliability with change-point model[J]. Communications in Statistics-Simulation and Computation, 2001, 30(3): 623-635

[15] Jeong K M. An adaptive failure rate change-point model for software reliability[J]. International Journal of Reliability and Applications, 2001, 2(3): 199-207

[16] Zou F Z. A change-point perspective on the software failure process[J]. Software Testing, Verification and Reliability, 2003, 13(2): 85-93

[17] Shyur H J. A stochastic software reliability model with imperfect debugging and change-point[J]. Journal of Systems and Software, 2003, 66(2): 135-141

[18] Zhao J, Liu H W, Cui G, et al. Software reliability growth model with change-point and environmental function[J]. Journal of Systems and Software, 2006, 79(11): 1578-1587

[19] Huang C Y. Performance analysis of software reliability growth models with testing-effort and change-point[J]. Journal of Systems and Software, 2005, 76(2): 181-194

[20] Wang Z, Wang J. Parameter estimation of some NHPP software reliability models with change-point[J]. Communications in Statistics-Simulation and Computation, 2005, 34(1): 121-134

[21] Kapur P K, Singh V B, Anand S, et al. Software reliability growth model with change-point and effort control using a power function of testing time[J]. International Journal of Production Research, 2008, 46(3): 771-787

[22] Lin C T, Huang C Y. Enhancing and measuring the predictive capabilities of the testing-effort dependent software reliability models[J]. Journal of Systems and Software, 2008, 81(6): 1025-1038

[23] Li X, Xie M, Ng S H. Sensitivity analysis of release time of software reliability models incorporating testing effort with multiple change-points[J]. Applied Mathematical Modelling, 2010, 34(11): 3560-3570

[24] Huang C Y, Hung T Y. Software reliability analysis and assessment using queueing models with multiple change-points[J]. Computers and Mathematics with Applications, 2010, 60(7): 2015-2030

[25] Huang C Y, Lyu M R. Estimation and analysis of some generalized multiple change-point software reliability models[J]. IEEE Transactions on Reliability, 2011, 60(2): 498-514

[26] Singh O, Anand A, Singh J, et al. Assessment of distribution based SRGM with the effect of change-point and imperfect debugging incorporating irregular fluctuations[J]. Journal of Pure and Applied Science & Technology, 2012, 2(1): 37-49

[27] Ravishanker N, Liu Z H, Ray B K. NHPP models with Markov switching for software reliability[J]. Computational Statistics and Data Analysis, 2008, 52(8): 3988-3999

[28] Nam S, Cha J H, Cho S. A Bayesian Change-Point Analysis for Software Reliability Models[J]. Communications in Statistics-Simulation and Computation, 2008, 37(9): 1855-1869

[29] Durand J B, Gaudoin O. Software reliability modelling and prediction with hidden Markov chain[J]. Statistical Modelling, 2005, 5(1): 75-93

[30] Forney Jr G D. The Viterbi algorithm[J]. Proceedings of the IEEE, 1973, 61(3): 268-278

[31] Inoue S, Hayashida S, Yamada S. Toward Practical Software Reliability Assessment with Change-Point Based on Hazard Rate Models[C]// 2013 IEEE 37th Annual Computer Software and Applications Conference (COMPSAC). IEEE, 2013: 268-273

[32] Wang Shuai. Software Reliability Forecasting Method Based on Decomposition and Reconstruction of Series[D]. Nanjing: Nanjing University of Aeronautics and Astronautics, 2014 (in Chinese)
汪帅. 基于序列分解与重构的软件可靠性预测方法[D]. 南京: 南京航空航天大学, 2014

(上接第 141 页)

[10] Chai M, Schlingloff B H. Monitoring Systems with Extended Live Sequence Charts[C]// 14th International Conference on Runtime Verification. Springer, 2014: 48-63

[11] Kugler H, Harel D, Pnueli A, et al. Temporal logic for scenario-based specifications[C]// Proc. of the 11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS05). 2005, 3440: 445-460

[12] Clavel, Duran, Marti-Oliet, et al. Maude Manual (version 2.6) [M]. University of Illinois, Urbana-Champaign 1, 2011: 1-491

[13] Martí—Oliet N. An Introduction to Maude and Some of Its Applications[M]// Practical Aspects of Declarative Languages. Springer Berlin Heidelberg, 2010: 4-9

[14] Zhao Lin, Tang Tao, Xu Tian-hua, et al. Runtime Verification and its Applications in Train Control System[J]. Railway Society, 2011, 33(12): 65-71 (in Chinese)
赵琳, 唐涛, 徐田华, 等. 运行时验证及其在列车运行控制系统中的应用[J]. 铁道学报, 2011, 33(12): 65-71

[15] Niu Ru, Cao Yuan, Tang Tao. Formal Modelling and Analysis of RBC Handover protocol for ETCS Level 2 Using Stochastic Petri Nets[J]. Railway Society, 2009, 31(4): 52-58 (in Chinese)
牛儒, 曹源, 唐涛. ETCS-2 级列控系统 RBC 交接协议的形式化分析[J]. 铁道学报, 2009, 31(4): 52-58

[16] ERTMS/ETCS Subset-098: ERTMS/ETCs Class 1 RBC-RBC safe communication interface[EB/OL]. http://www.aeif.org/db/docs/ccm/SUBSET-052 v222. 2005

[17] Zhao Lin, Tang Tao, Xu Tian-hui, et al. Runtime Verification and its Applications in Train Control Systems[J]. Journal of The China Railway Society, 2011, 33(12): 65-71 (in Chinese)
赵琳, 唐涛, 徐田华, 等. 运行时验证及其在列车运行控制系统中的应用[J]. 铁道学报, 2011, 33(12): 65-71