嵌入偏序约简的状态事件线性时序逻辑验证

谢 健[1)] 阚双龙[1)] 黄志球[1)] 王 飞[1)] 杨志斌[1)] 李伟津[2)]

[1)](南京航空航天大学计算机科学与技术学院 南京 210016)

[2)](南京航空航天大学航天学院 南京 210016)

摘 要 模型检验是硬件和软件形式化验证最成功的技术之一.目前大部分的模型检验技术是基于状态的而不考虑迁移上的操作和事件.这导致模型检验在验证使用事件进行交互的组件系统中面临新的困难,因此需要新的规约技术对状态事件系统进行规约.状态事件线性时序逻辑(State/Event Linear Temporal Logic,SE-LTL)给出了一种简洁和直接的方式表达包含状态和事件的系统属性.在 SE-LTL 中,状态和事件都可以作为原子命题.基于自动机理论的线性时序逻辑(Linear Temporal Logic,LTL)模型检验可以被用来对 SE-LTL 属性进行验证.然而 SE-LTL 属性在经典的 stutter 等价(stutter-equivalent)下无法保持,所以最有效的并发程序状态约简技术:偏序约简技术(Partial Order Reduction,POR)不能直接应用于 SE-LTL 的验证.该文提出一种新的方法利用已有的偏序约简技术对 SE-LTL 验证过程的状态空间进行约简.该方法分为两个部分:第一个部分是针对 SE-LTL 不带 NEXT 算子的约简方法;第二部分则是带 NEXT 算子的约简方法.第一部分的主要思想是从一个 Büchi 自动机(Automata,BA)中抽取出"状态部分","状态部分"的含义是该部分只与系统的状态相关.基于"状态部分",给出关于 BA 和标签 Kripke 结构(Labeled Kripke Structure)的同步乘,并在同步乘的构造过程中嵌入偏序约简技术,从而约简同步乘的状态空间,即该文的约简技术是 on-the-fly 的.嵌入的偏序约简在已有的偏序约简基础上,面向 SE-LTL 公式中的事件引入新的可见操作的识别方法.为了能够将偏序约简技术应用到所有的 SE-LTL 公式,该文同时给出验证 SE-LTL 带 NEXT 算子的偏序约简算法.NEXT 算子是偏序约简的另一个主要障碍.该部分是文中的第二部分工作.该部分的技术依然是 on-the-fly 的,并且需要与状态部分的识别相结合.通过将该文技术实现到 SPIN 模型检验器中对已有的模型进行验证.Spin 是针对 LTL 的并发程序模型检验器.实现部分包括 SE-LTL 到 BA 的转化,以及 on-the-fly 的模型验证过程.实验的过程主要针对三个模型集:生产消费者模型,哲学家就餐问题以及公共对象请求代理体系结构中的 GIOP 协议.验证结果表明,对比完全基于状态的模型检验和不带偏序约简的状态事件模型检验,该文的方法具有更好的效率,并且能够被应用于状态事件系统,特别是安全有关嵌入式系统的验证.

关键词 偏序约简;状态事件线性时序逻辑;模型检验;同步乘;标签 Kripke 结构

中图法分类号 TP302 **DOI 号** 10.11897/SP.J.1016.2019.02145

State/Event Linear Temporal Logic Verification with the Integration of Partial Order Reduction

XIE Jian[1)] KAN Shuang-Long[1)] HUANG Zhi-Qiu[1)] WANG Fei[1)] YANG Zhi-Bin[1)] LI Wei-Wei[2)]

[1)](*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016)

[2)](*College of Astronautics, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016)

Abstract Model checking is one of the most successful formal techniques for verifying hardware and software systems. Most of the model checking techniques are state-based, that is, events and

收稿日期:2016-11-02;在线出版日期:2018-02-08.本课题得到国家高技术研究发展计划项目(2015AA105303)、国家重点研发计划项目(2016YFB1000802)、江苏省自然科学基金青年基金(BK20170809)和中国博士后基金面上基金(2018M632304)资助.**谢 健**,博士研究生,主要研究方向为安全性分析、模型检验、定理证明.E-mail:xiejian_5@nuaa.edu.cn.**阚双龙**(通信作者),博士,主要研究方向为模型检验、定理证明.E-mail:kanshuanglong@nuaa.edu.cn.**黄志球**,博士,教授,主要研究领域为安全关键软件开发、形式化方法.**王 飞**,博士研究生,主要研究方向为软件可追踪性.**杨志斌**,博士,副教授,主要研究方向为安全关键嵌入式系统、形式化验证、AADL 和同步语言.**李伟津**,博士,助理研究员,主要研究方向为软件仓库挖掘、机器学习.

actions on the transitions are not under consideration. However, problems arise when verifying component based systems with model checking, in which the communication among components are data dependent events. Therefore, the formal specification techniques for state/event systems are required. State/Event Linear Temporal Logic (SE-LTL) is a natural and concise way to formal specify a system property, which include both states and events. In SE-LTL, both states and events are viewed as atomic propositions. Model Checking based on automata-theoretic for Linear Temporal Logic (LTL) can be used to verify SE-LTL properties automatically. However, most SE-LTL formulas are not preserved under classic stutter-equivalent relations. Therefore, they cannot be verified with classic Partial Order Reduction (POR), which is one of the most efficiency state space reduction techniques for concurrent systems. In this paper, we propose a novel technique, which is to exploit classical POR to reduce the number of states and transitions of systems during the verification of SE-LTL formulas. It consists of two parts: the first part concentrates on the state space reduction with respect to SE-LTL formulas without NEXT operators and the second part concentrates on the state space reduction with respect to SE-LTL formulas with NEXT operators. The main idea of the first part lies at detecting state parts from Büchi Automata (BA) translated from SE-LTL formulas. State parts means that this part is only related to the states of the systems. Based on the state parts, we propose a Synchronous Product (SP) of a BA and a Labeled Kripke Structure (LKS), which is used to model the behavior of the system. During the construction of the SP, POR is integrated in the construction to reduce the state space of the SP. In other words, POR reduces the state spaces in on-the-fly manner. The integrated POR is modified by introducing a visible action detection with respect to the events appearing in SE-LTL formulas. In order to apply the technique to the full class of SE-LTL, we extend the technique for the formulas with NEXT operators, which are another obstacle for POR. This is the second part of our work. The technique detects POR-components in a BA, which can be verified by POR. This technique is also in on-the-fly manner and mixed with the detection of state-transitions. In order to evaluate the efficiency of the technique, we have implemented our techniques into SPIN model checker, which is a tool for verifying concurrent system with respect to classic LTL. The implementation includes the translation from SE-LTL formulas and the on-the-fly state space reductions. The experiments are conducted on three benchmarks: the producer-consumer models, the dinning philosopher problem, and the GIOP protocol in the common object request broker architecture (CORBA). The experimental results show that compared with the pure state-based model checking with POR and SE-LTL model checking without POR, our technique is more efficient and promising and can be used for the verification of state/event systems, especially safety-critical embedded systems.

Keywords partial order reduction; state/event linear temporal logic; model checking; synchronous products; Labeled Kripke Structure

1 引 言

将已有的模型检验技术[1-3]应用到基于模块或组件的软件系统中面临两个重要的困难:(1)如何有效地规约软件的行为以及软件应该满足的属性;(2)难以解决的软件状态空间爆炸问题.这类软件组件间的交互主要通过数据依赖的事件完成.因此需要一种规约方法,能够同时规约事件和状态.基于标记的有限状态自动机(annotated finite state automata)建模方法可以是完全基于状态的或者是完全基于事件的.尽管这两种方法是可以相互转化的,但是从一种规约转化到另外一种规约通常会导致状态空间的剧烈增加.除此之外,两种方法很难被应用到

组件化软件建模中，因为组件间的交互是通过具有数据依赖的事件完成的. 为了解决这个问题，Chaki等人[4-5]提出一种规约框架可以同时规约事件和状态. 在这个框架中，软件行为使用标记 Kripke 结构进行规约(Labeled Kripke Structure，LKS). LKS是传统的 Kripke 结构(Kripke Structure，KS)[6]的扩展. LKS 的状态被标记一个状态的原子命题集合，迁移被标记一个操作. 属性的规约采用状态事件线性时序逻辑(State/Event Linear Temporal Logic，SE-LTL). SE-LTL 是经典的线性时序逻辑(Linear Temporal Logic，LTL)[7]的扩展. SE-LTL 与 LTL 的不同点在于 SE-LTL 中允许事件作为原子命题. Chaki 等人[5]证明了已有的面向 LTL 的自动机理论模型检验技术可以被应用于 SE-LTL 的模型检验过程，并且不会增加任何其他代价.

然而，模型检验中最有效的约简技术：偏序约简技术(Partial Order Reduction，POR)[8-10]，不能够被直接用来验证 SE-LTL. 原因是大部分的 SE-LTL 属性无法在 stutter-等价(stutter-equivalence)下保持，即对于两条 stutter-等价的路径，无法保证对于任意 SE-LTL 公式，这两条路径同时满足或同时不满足. 可以更进一步的解释为：一个 LKS 中的事件可以表示为 LKS 中状态变量的变化，这种变化需要使用 LTL 中的 NEXT 算子来表示. 例如对于一条从状态 s 到状态 s' 的迁移，它对应的事件可以表示为 $s \wedge Xs'$，即当前状态为 s，下一个状态为 s'，其中 $\wedge$ 为合取符号，X 为 LTL 中 NEXT 算子. 带有 NEXT 算子的 LTL 在 stutter-等价关系下是不保持的. 因此无法使用偏序约简技术. Chaki 等人[4]没有在文章中介绍偏序约简技术，但是提出了面向 SE-LTL 偏序约简的必要性. Benes 等人[11-12]提出了一种面向 SE-LTL的偏序约简技术. 这种约简技术基于状态事件 stutter-等价(State/Event Stutter-Equivalence). 这种方法的缺点在于大部分 SE-LTL 公式在状态事件 stutter-等价下依然不能保持，所以他们基于 SE-LTL 定义了一种新的逻辑：弱状态事件线性时序逻辑(weak State/Event Linear Temporal Logic，wSE-LTL). 这种逻辑在状态事件 stutter-等价下是保持的. 实际上 wSE-LTL 是 SE-LTL 的一个子集，所以文献[11-12]的方法只适用于 SE-LTL 的一个子集. 文献[13]给出了本团队已有工作与文献[3-4]中工作的详细比较. 本团队之前工作文献[14]介绍了一种面向 LTL 带 NEXT 算子的偏序约简方法. 因为每一个 SE-LTL 公式都可以转化为带 NEXT 算子的 LTL 公式，所以文献[14]的方法可以被用到 SE-LTL 公式的检验中. 但是该方法没有本文介绍的方法的效率高. 还有，例如文献[15]将状态事件系统转化为完全的状态系统，然后使用已有的偏序约简技术对系统进行约简，但是文献[5]表明这种转化首先就会大幅度地增加系统的状态空间. 也有其它的规约方法可以在异步交互过程中同时规约事件和状态. 例如系统 Verilog 断言(System Verilog Assertion，SVA)[16]. SVA 支持对系统的立即断言、并发断言、序列等规约. 实际上本文的偏序约简技术也可以被应用到 SVA 中. 例如一个断言 $assert(\neg p)$ 也可以规约为 Gp，其中 G 是 LTL 中的算子，它表示对于所有的未来状态，p 都被满足. 如果系统行为能够被描述为一个 LKS，SVA 断言可以被描述为 SE-LTL公式，那么本文方法也可以被使用. Event-B[17]是一种事件驱动的离散系统规约语言. 通过将 Event-B 模型转化到 LKS，可以对该系统的 SE-LTL 属性进行验证.

本文提出一种新的方法将偏序约简应用于 SE-LTL 模型检验过程中. 首先将一个 SE-LTL 公式转化为一个 Büchi 自动机(Büchi Automata，BA)，然后将偏序约简嵌入到 BA 和 LKS 的同步乘(Synchronous Products，SP)的构造过程中. 一个从 SE-LTL 公式转化得到的 BA 在下文称为 SE-BA. 本文的方法是从一个 SE-BA 中识别“状态部分”，状态部分表示该部分的验证可以使用偏序约简，即当验证到 SE-BA 的状态部分时，偏序约简可以被使用对 LKS 状态空间进行约简. 除此之外，经典的偏序约简技术需要被修改来适应面向 SE-LTL 的状态约简. 需要重新考虑偏序约简中面向事件的可见操作的识别. 本文的核心是定义一个嵌入偏序约简的 BA 和 LKS 的同步乘. 在同步乘的构造过程中，状态部分指导状态约简. 为了能够将偏序约简方法应用于所有的 SE-LTL 公式，包括带 NEXT 算子的公式，本文提出了一种面向 SE-LTL 带 NEXT 算子(SE-LTL 带 NEXT 算子简写为 SE-LTLX)的偏序约简方法. 该方法通过识别 SE-BA 中的一类组件，这类组件的验证过程可以使用偏序约简. 本文方法已经实现到模型检验器 SPIN[18-19]中. 实验结果表明本文方法比不带偏序约简的状态事件模型检验算法以及完全基于状态的偏序约简算法效率更高.

本文第 2 节给出本文方法的理论基础；第 3 节给出从 SE-LTL 到 SE-BA 转化过程中约简信息的

提取;第 4 节给出面向 SE-LTL 的偏序约简技术;第 5 节给出面向 SE-LTLX 的偏序约简技术;第 6 节给出工具实现和实验的结果;第 7 节是本文的总结和未来工作.

2 理论基础

本节介绍 SE-LTL 模型检验的理论基础[5],包括 LKS,SE-LTL,SE-BA,SE-BA 和 LKS 的同步乘以及经典偏序约简的基本概念.

标签 Kripke 结构(LKS). 一个标签 Kripke 结构定义为一个 6 元组$(S, Init, P, Act, T, \mathcal{L}_s)$,其中:(1)$S$ 定义为一个有限状态的集合;(2)$Init \subseteq S$ 是初始状态的集合;(3)P 是一个有限的状态原子命题的集合;(4)Act 是一个操作的有限集合;(5)$T \subseteq S \times Act \times S$ 是一组迁移关系的集合;(6)$\mathcal{L}_s: S \to 2^P$ 是一个状态标签函数,对于任意一个状态 s,$\mathcal{L}_s(s)$ 表示所有在该状态满足的原子命题的集合.

LKS 的行为可以用 LKS 的路径表示,LKS 的一条无穷路径 $\sigma = \langle s_0, a_0, s_1, a_1, s_2, \cdots \rangle$ 是一条状态和操作交替出现的序列,并且该序列满足对于任意的 $i \geq 0$,$(s_i, a_i, s_{i+1}) \in T$. 本文使用符号 $L(s)$ 表示 LKS 中所有从状态 s 出发的无穷路径的集合. 一条从初始状态出发的无穷路径称为初始无穷路径. 一个 LKS $\mathcal{M}$ 的所有初始无穷路径集合表示为 $L(\mathcal{M})$.

KS 结构是经典 LTL 模型检验的行为描述,LKS 通过添加迁移上的操作扩展 KS. 本文继承 KS 中迹的概念来定义 LKS 中一条路径的迹. LKS 中一条路径 $\sigma = \langle s_0, a_0, s_1, a_1, s_2, \cdots \rangle$ 的迹定义为 $\pi(\sigma) = \mathcal{L}_s(s_0), \mathcal{L}_s(s_1), \cdots$,即 σ 的迹为将标签函数 $\mathcal{L}_s$ 按顺序应用到 σ 的所有状态上. 两条迹是 stutter-等价的当且仅当存在一个对两条迹的划分,满足对于任意自然数的 $k \geq 0$,两条迹的第 k 个块的所有原子命题集合相同. 偏序约简技术应用的前提是验证属性在 stutter-等价下是保持的,但是 SE-LTL 和 LTL 带 NEXT 算子(表示为 LTL+X)是不满足该要求的. 所以在这两类属性的模型检验过程无法使用偏序约简. LTL 不带 NEXT 算子(表示为 LTL-X)是满足该要求的,所以可以使用偏序约简.

状态事件线性时序逻辑. SE-LTL 通过将事件加入到原子命题中来扩展经典的 LTL,即 SE-LTL 的原子命题既包括系统状态的原子命题也包括系统的事件. SE-LTL 的语法定义如下:

$$\phi = p \mid e \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg \phi \mid \mathrm{X}\phi \mid \phi \mathrm{U} \phi,$$

其中 p 为状态原子命题,状态原子命题集合表示为 SP,e 为一个事件,事件的集合定义为 EP. 符号 $\wedge$,$\vee$,$\neg$ 分别表示合取、析取、取反. 符号 X,U 为时序算子.

需要注意的是本文中名词"操作"和名词"事件"具有不同的含义. 这和文献[5,11]不同,它们认为事件和操作是等价的. 本文"操作"在 LKS 中表示可以引起状态变化的动作,事件是为了验证而设定的. 事件比操作具有更高的抽象程度. 从操作到事件存在一个映射:$\mathcal{L}_e: Act \to EP$,表示当一个操作被执行,它映射的事件发生.

SE-LTL 的语义需要在 LKS 模型下进行解释. LKS 的一条无穷路径表示为 $\sigma = \langle s_0, a_0, s_1, a_1, \cdots \rangle$. 它的第 i 个后缀定义为 $\sigma^i = \langle s_i, a_i, s_{i+1}, \cdots \rangle$. 对于一个 SE-LTL 公式 ϕ,它在 LKS 的一条路径 $\sigma = \langle s_0, a_0, s_1, a_1, \cdots \rangle$ 的语义解释为:

(1)$\sigma \models p$,当且仅当 $p \in \mathcal{L}_s(s_0)$;

(2)$\sigma \models e$,当且仅当 $\mathcal{L}_e(a_0) = e$;

(3)$\sigma \models \phi_1 \wedge \phi_2$,当且仅当 $\sigma \models \phi_1$ 且 $\sigma \models \phi_2$;

(4)$\sigma \models \phi_1 \vee \phi_2$,当且仅当 $\sigma \models \phi_1$ 或 $\sigma \models \phi_2$;

(5)$\sigma \models \mathrm{X}\phi_1$,当且仅当 $\sigma^1 \models \phi_1$,

(6)$\sigma \models \phi_1 \mathrm{U} \phi_2$,当且仅当存在一个 $i \geq 0$ 满足 $\sigma^i \models \phi_2$ 并且对于所有的 $0 \leq j \leq i-1$,$\sigma^j \models \phi_1$.

在 SE-LTL 中存在一些常用的扩展算子,可以由已有的算子进行定义,例如 $\phi_1 \mathrm{R} \phi_2 \equiv \neg(\neg \phi_1 \mathrm{U} \neg \phi_2)$,$\mathrm{G}\phi \equiv \bot \mathrm{R}\phi$,$F\phi \equiv \top \mathrm{U}\phi$. 对于一个 LKS $\mathcal{M}$ 和一个 SE-LTL 公式 ϕ,$\mathcal{M} \models \phi$ 当且仅当对于 $\mathcal{M}$ 所有初始无穷路径 σ,$\sigma \models \phi$.

Büchi 自动机. 在 LTL 模型检验过程中,通常需要将 LTL 公式转化为一个 Büchi 自动机. SE-LTL 模型检验也需要将 SE-LTL 转化为 Büchi 自动机,这里称为状态事件 Büchi 自动机(State/Event-Büchi Automata,SE-BA). 首先给出 SE-BA 的定义:

一个 SE-BA 定义为一个 5 元组 $\mathcal{B} = (Q, \Sigma, \mathcal{T}, I, F)$,其中 Q 为一个状态的有限集合;Σ 为有限的字母表,它由所有的状态原子命题和事件的集合并的幂集组成,即 $\Sigma = 2^{SP \cup Ep}$,$\Sigma' = 2^{\Sigma}$;$\mathcal{T} \subseteq Q \times \Sigma' \times Q$ 为迁移函数,一个迁移表示为 (q, α, q');$I \subseteq Q$ 初始状态集合;$F \subseteq Q$ 为接受状态集合.

对于 SE-BA 的定义是基于迁移的(transition-based),即在迁移上标 Σ' 的元素. 一条 SE-BA 的无穷路径定义为 $\langle q_0, \alpha_0, q_1, \alpha_1, q_2, \cdots \rangle$,满足对于任意的 $i \geq 0$,$(q_i, \alpha_i, q_{i+1}) \in \mathcal{T}$. 该路径为接受路径当且仅当在路径上存在无穷多个状态在 F 中. 对于任意一

条 LKS 无穷路径 $\sigma=\langle s_0,a_0,s_1,a_1,\cdots\rangle$ 和一条 SE-BA $\mathcal{B}$ 的接受路径 $\sigma'=\langle q_0,\alpha_0,q_1,\alpha_1,\cdots\rangle$，如果满足对于任意的 $i\geq 0$，$\mathcal{L}_s(s_i)\wedge\mathcal{L}_e(a_i)\models\alpha_i$，则称为 σ' 接受 σ. 一条 LKS 路径被一个 SE-BA 接受当且仅当在该 SE-BA 中存在一条接受路径接受该 LKS 路径. 对于 SE-BA 的任意一个状态 q，$L(q)$ 表示一个 LKS 路径的集合，该集合满足对于其中任意一条 LKS 路径 σ，存在一条从 q 出发的 SE-BA 接受路径接受 σ.

模型检验的过程是建立一个 BA 和一个 LKS 的同步乘，同步乘是这两者行为的交集. 当同步乘中存在一条接受路径，表明存在一条反例路径.

SE-BA 和 LKS 的同步乘. 对于一个 LKS $\mathcal{M}=(S,Init,P,Act,T,\mathcal{L}_s)$ 和一个 SE-BA $\mathcal{B}=(Q,\Sigma,\mathcal{T},I,F)$. $\mathcal{M}$ 和 $\mathcal{B}$ 的同步乘定义为：$\mathcal{M}\otimes\mathcal{B}=(Q_p,\Sigma,\mathcal{T}_p,I_p,F_p)$，其中 $Q_p\subseteq S\times Q$ 为有限状态集合，一个状态表示为一个二元组 (s,q)，其中 $s\in S$，$q\in Q$；$I_p\subseteq Init\times I$ 为初始状态集合；$F_p\subseteq Q_p$ 为接受状态集合；对于任意一个状态 $(s,q)\in F_p$，$q\in F$；$\mathcal{T}_p\subseteq Q_p\times Q_p$ 为迁移的集合，$((s,q),(s',q'))\in Q_p\times Q_p$ 为 $\mathcal{M}\otimes\mathcal{B}$ 的一个迁移当且仅当：$\exists(s,a,s')\in T$，$\exists(q,\alpha,q')\in\mathcal{T}$，$\mathcal{L}_s(s)\wedge\mathcal{L}_e(a)\models\alpha$.

LKS 和 SE-BA 的同步乘也是一个 SE-BA. 模型检验方法是在同步乘中搜索一条接受路径. 如果存在一条接受路径，则在系统中存在一条违反属性的反例路径，否则系统设计是正确的.

定理 1.　对于一个从 SE-LTL 公式 $\neg\phi$ 转化而来的 SE-BA $\mathcal{B}$ 和一个 LKS $\mathcal{M}$. $\mathcal{M}\models\mathcal{B}$ 当且仅当 $L(\mathcal{M}\otimes\mathcal{B})=\varnothing$.

定理 1 表明同步乘$\otimes$定义的正确性. 它的证明参见文献[5].

偏序约简. 本节给出经典的基于状态的偏序约简的基本思想. 因为 LKS 可以通过对 KS 的迁移标记操作得到，这里使用 LKS 来描述经典的偏序约简而不是 KS. 实际上经典的偏序约简也需要考虑迁移上的操作.

对于一个 LKS $\mathcal{M}$ 的状态 s，它的迁移关系为 T，符号 $enable(s)=\{(s,a,s')\mid(s,a,s')\in T\}$ 表示所有可以从状态 s 出发的迁移的集合. 经典的偏序约简技术通过选择 $enable(s)$ 的一个子集来扩展状态 s. 这个子集称为 ample 集，表示为 $ample(s)$. 因为 $ample(s)$ 是 $enable(s)$ 的一个子集，所以只有 $\mathcal{M}$ 的部分行为被检验，从而可以对状态进行约简. 经典偏序约简算法的核心就是计算 ample 集. 首先需要介绍两个重要的概念：(1) 对于两个迁移，如果它们可以并发运行那么它们称为相互独立的，否则称为相互依赖的. 相互依赖和相互独立的形式化定义见文献[6]. (2) 一个操作 a 是不可见操作当且仅当对于所有的迁移 (s,a,s')，有 $\mathcal{L}_s(s)=\mathcal{L}_s(s')$，否则该操作称为可见迁移. 对于一个 LKS 状态 s，$ample(s)$ 的计算方法是选择 $enable(s)$ 中一个子集作为 $ample(s)$，该子集满足如下 4 个条件：

C1. $ample(s)=\varnothing$ 当且仅当 $enable(s)=\varnothing$；

C2. 在 LKS 中，从 s 出发的所有路径满足：对于该路径上的一条迁移，如果它依赖某一个 $ample(s)$ 中的迁移，则在该迁移执行前，路径上已经有 $ample(s)$ 中的一个迁移在它之前执行过；

C3. 如果 s 没有完全扩展，即 $ample(s)\subset enable(s)$，那么 $ample(s)$ 中的迁移都被标记不可见操作；

C4. 在 LKS 中的任意一个环，该环上至少存在一个状态 s，满足 $ample(s)=enable(s)$.

所有满足以上 4 个条件的 $enable(s)$ 的子集都可以作为 $ample(s)$. 本文面向 SE-LTL 的偏序约简技术也需要计算 ample 集，但是需要对条件 C3 进行修改来适用 SE-LTL.

3　SE-LTL 到 SE-BA 转化过程中可约简信息提取

经典的偏序约简技术只需要考虑一个 LTL 公式中的状态原子命题. 而本文面向 SE-LTL 的偏序约简需要考虑从 SE-LTL 转化的 SE-BA 的自动机结构. 面向 SE-LTL 偏序约简在下文简称为 SE-POR. 本节通过识别 SE-BA 中的“状态部分”来识别 SE-LTL 中的状态部分.

本文的工作主要基于迁移. 已存在部分工作讨论了从 SE-LTL 到 SE-BA 的转化，例如 LTL2BA[20]，LTL3BA[13] 和 SPIN 的转化部分[18]. 所有的这些算法在不做修改的状态下都可以被用来将 SE-LTL 公式转化为 SE-BA 并且通过修改可以被用来识别 SE-BA 中的状态部分. 本文给出一种新的转化过程描述，在转化过程中，识别 SE-BA 中状态部分. 本节的转化没有考虑 SE-BA 的简化方法，这样可以更清楚地描述偏序约简算法.

首先，介绍文献[20]中的两个重要概念. (1) 取反范式(Negative Normal Form，NNF). 一个 SE-LTL 公式是 NNF 当且仅当没有其他的算子在取反算子的作用域内. 每一个 SE-LTL 公式都可以被转化为

对应 NNF. 但是对于算子 U 的转化需要使用算子 R. 所以 SE-LTL 到 SE-BA 转化过程需要考虑算子 R. 在下文中,所有的 SE-LTL 公式都假设为 NNF. 第二个重要概念是时序公式(temporal formula). 一个 SE-LTL 公式为时序公式当且仅当该公式的最大作用域算子(即公式解析树的根)既不是合取算子又不是析取算子. 例如 $p\mathrm{U}q$ 是一个时序公式,但是 $p\wedge(p\mathrm{U}q)$不是一个时序公式. 基于时序公式,我们定义 SE-LTL 的析取范式(Disjunctive Normal Form, DNF). 对于一个 SE-LTL 公式 ϕ,如果 ϕ 是析取范式当且仅当 $\phi=\phi_1\vee\cdots\vee\phi_n$,其中 $\phi_i(1\leq i\leq n)$是一个由时序公式组成的合取(原子命题按定义也是时序公式). 例如对于 SE-LTL 公式 $((p\mathrm{U}q)\vee t)\wedge r$,它的析取范式为 $((p\mathrm{U}q)\wedge r)\vee(t\wedge r)$.

本节从 SE-LTL 公式转化到 SE-BA 包含两个步骤:(1)将一个 SE-LTL 公式转化为一个泛化的 BA(Generalized BA, GBA);(2)将 GBA 转化为一个 BA. 首先给出 GBA 的基本定义. 一个 GBA 是一个 5 元组:$\mathcal{G}=(Q_g,\Sigma,\mathcal{T}_g,I_g,F_g)$,其中 Q_g 为一个状态的有限集合;Σ 为有限的字母表,$\Sigma'=2^{\Sigma}$;$\mathcal{T}_g\subseteq Q_g\times\Sigma'\times Q_g$ 为迁移函数,一个迁移表示为 (q,α,q');$I_g\subseteq Q_g$ 为初始状态集合;$F=\{T_1,\cdots,T_r\}$,其中 $T_i(1\leq i\leq r)$为接受迁移的集合. GBA 的一条无穷路径是一条状态迁移交错序列$\langle q_0,\alpha_0,q_1,\cdots\rangle$满足:$q_0\in I_g$ 且对于 $i\geq 0$,$(q_i,\alpha_i,q_{i+1})\in\mathcal{T}_g$. GBA 的一条无穷路径是接受路径当且仅当对于任意的 $1\leq j\leq r$,该路径上出现无穷多条迁移存在于 T_j 中.

3.1 SE-LTL 到 GBA 的转化

对于一个 SE-LTL 公式 ψ,为了能够将其转化为一个 GBA$(Q_g,\Sigma,\mathcal{T}_g,I_g,F_g)$,首先将 ψ 转化为其 NNF 然后转化为其 DNF. 假设它的 DNF 为 $\psi=\psi_1\vee\cdots\vee\psi_n$,其中 ψ_i 是一组时序公式的合取. GBA 的初始状态集合 I_g 中包含了 n 个初始状态,分别对应 ψ_1 到 ψ_n. 为了能够构造状态集合 Q_g 和迁移函数 $\mathcal{T}_g$. 首先设置 $Q_g=I_g$,然后扩展 Q_g 中的状态,即对每一个状态,它所有迁移的下一个状态可以被加入到 Q_g 中.

下面描述如何构造一个状态的所有迁移. 对于一个状态 q,假设它对应的 SE-LTL 公式为 ψ_i 且 $\psi_i=\phi_1\wedge\cdots\wedge\phi_m$. 一个合取也可以表示成一个集合,即 $\psi_i=\{\phi_1,\cdots,\phi_m\}$. 计算从一个状态 q 出发的所有状态分为两步:

第一步为对每一个 ϕ_i 计算一个集合,集合中的元素为一个对(P,N),其中 P 是一个文字(literal)的集合,N 为时序公式的集合. 一个文字是一个原子命题或是该原子命题的取反. 原子命题可以为状态原子命题或者事件. 对于一个时序公式 ϕ,ϕ 对应的集合表示为 $trans(\phi)$. 我们介绍一个针对该集合的操作$\otimes$. 对于两个对的集合 S_1,S_2,那么

$$S_1\otimes S_2=\{(P,N)\mid\exists(P_1,N_1)\in S_1,\exists(P_2,N_2)\in S_2.\ P=P_1\cup P_2\text{且}N=N_1\cup N_2\}.$$

集合 $trans(\phi)$的递归计算如下:

(1)如果 $\phi=a$ 或者 $\phi=\neg a$,其中 a 是一个原子命题(状态原子命题或者事件),那么 $trans(\phi)$分别为$\{(\{a\},\varnothing)\}$或者$\{(\{\neg a\},\varnothing)\}$.

(2)如果$\phi=\mathrm{X}\varphi$,且$\varphi=\varphi_1\vee\cdots\vee\varphi_k$,那么$trans(\phi)=\{(\varnothing,\varphi_1),\cdots,(\varnothing,\varphi_k)\}$. 这个集合中所有的对称为 X-对. X-对将在 SE-LTL 带 NEXT 算子的偏序约简中使用.

(3)如果$\phi=\varphi_1\mathrm{U}\varphi_2$,那么 $trans(\phi)=trans(\varphi_2)\cup(trans(\varphi_1)\otimes\{(\varnothing,\{\phi\})\})$,即首先需要计算$trans(\varphi_2)$和 $trans(\varphi_1)$.

(4)如果 $\phi=\varphi_1\mathrm{R}\varphi_2$,那么 $trans(\phi)=trans(\varphi_2)\otimes(trans(\varphi_1)\cup\{(\varnothing,\{\phi\})\})$.

(5)如果 $\phi=\varphi_1\wedge\varphi_2$,那么 $trans(\phi)=trans(\varphi_1)\otimes trans(\varphi_2)$.

(6)如果 $\phi=\varphi_1\vee\varphi_2$,那么 $trans(\phi)=trans(\varphi_1)\cup trans(\varphi_2)$.

第二步构造所有从状态 q 出发的迁移. 从 q 出发的所有迁移集合为 $trans(\phi_1)\times\cdots\times trans(\phi_m)$. 更精确地说,对于所有的 $1\leq l\leq m$,$trans(\phi_l)$中的一个元素为(P_t,N_t),那么从 q 出发的一个迁移为$(q,P_1\cup\cdots\cup P_m,q')$,其中 q'对应公式集合 $N'=N_1\cup\cdots\cup N_m$. N'也是一个时序公式的集合. 实际上 q 对应一个公式的集合为$\{\phi_1,\cdots,\phi_m\}$. 这个迁移可以被解释为:为了满足 q 对应的公式$\phi_1\wedge\cdots\wedge\phi_m$,该迁移选择在当前状态满足 $P_1\cup\cdots\cup P_m$中的所有原子命题,并且在下一状态满足 N'中的所有时序公式的合取. 使用符号$\mathcal{S}(q)$来表示一个状态 q 对应的时序公式集合. 例如$\mathcal{S}(q')=N'$. 对于 Q_g 中的任意一个状态 q_t,首先构造所有从它出发的迁移的集合. 对于其中的一条迁移(q_t,α,q_t'),如果 q_t'不在 Q_g 中,那么将 q_t'加入到 Q_g 中. 将该迁移加入到 $\mathcal{T}_g$ 中.

最后构造 GBA 的接受迁移条件 F_g. 接受条件定义为 $F_g=\{T_f\mid f\in U\}$,其中:

$$T_f=\{(q,\alpha,q')\mid f\notin\mathcal{S}(q)\vee\exists(P,N)\in trans(f).\ P\sqsubset\alpha\wedge f\notin N\subseteq\mathcal{S}(q')\}.$$

定义 T_f比较复杂,该定义来自文献[20]. 读者不必过

多关注该定义，因为它与本文方法关系不紧密.

下面定义 GBA 中三种类型的迁移. 一个文字是一个事件文字当且仅当该文字是一个事件或是一个事件的取反.

定义 1. 对于一个 GBA 中的迁移(q,α,q')且$\mathcal{S}(q)=\{\phi_1,\cdots,\phi_m\}$. 如果存在至少一个事件文字在$\alpha$中则该迁移称为事件迁移，否则称为状态迁移. 该迁移称为 X-迁移当且仅当存在一个$\phi_i\in\mathcal{S}(q)$，且(q,α,q')是选择$trans(\phi_i)$的一个 X-对构造的.

3.2 GBA 到 BA 的转化

对于一个 GBA $\mathcal{G}=(Q_g,\Sigma,\mathcal{T}_g,I_g,F_g)$，其中$F_g=\{T_1,\cdots,T_r\}$. 它对应的 BA 为$\mathcal{B}=(Q,\Sigma,\mathcal{T},I,F)$，其中：(1) $Q=Q_g\times\{0,\cdots,r\}$为状态的集合；(2) $I=I_g\times\{0\}$为初始状态集合；(3) $F=Q_g\times\{r\}$是接受状态的集合；(4) $\mathcal{T}=\{((q,j),\alpha,(q',j'))\mid(q,\alpha,q')\in\mathcal{T}_g$且$j'=next(j,(q,\alpha,q'))\}$，其中：

$$next(j,t)=\begin{cases}\max\{j\le i\le r\mid\forall j<k\le i,t\in T_k\}, & j\neq r\\ \max\{0\le i\le r\mid\forall 0<k\le i,t\in T_k\}, & j=r\end{cases}.$$

该转化方法来自文献[20]. 现在定义 SE-BA 中的状态迁移、事件迁移和 X-迁移.

定义 2. 对于一个 BA 中的迁移$((q,i),\alpha,(q',j))$，如果(q,α,q')为事件迁移则$((q,i),\alpha,(q',j))$称为事件迁移，否则称为状态迁移. 该迁移称为 X-迁移当且仅当(q,α,q')为 X-迁移.

图 1 给出一个从 $p\,\mathrm{U}(\mathrm{X}e)$转化的 SE-BA，其中$p$为一个状态原子命题，$e$为一个事件. 初始状态为$q_0$，接受状态集合为$\{q_2\}$. 只有一个 X-迁移$(q_0,\varnothing,q_1)$，它是由$trans(p\,\mathrm{U}\,\mathrm{X}e)$的 X-对构造的. 只有一个事件迁移$(q_1,\{e\},q_2)$. 其它迁移为状态迁移. $\mathcal{S}(q_0)$，$\mathcal{S}(q_1)$，$\mathcal{S}(q_2)$在图中展示. 图中迁移上空集表示该迁移不需要满足任何原子命题.

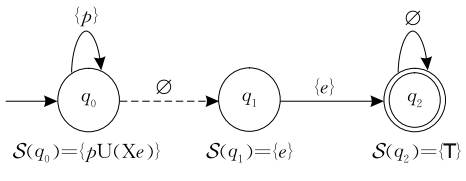

图 1　从 $p\mathrm{U}(\mathrm{X}e)$转化的 SE-BA

现在我们来定义一个 SE-BA 的状态部分. 假设ST为所有状态迁移的集合，XT为所有 X-迁移的集合. 那么 SE-BA 的状态部分为$ST-XT$，即所有不是 X-迁移的状态迁移. 事件迁移要求某些事件必须在系统中出现，状态迁移只要求系统处在某些状态. 下一节给出如何利用状态迁移对系统状态空间进行约简.

4　面向 SE-LTL 的偏序约简

经典的偏序约简技术需要计算 ample 集，本文提出 SE-POR 同样需要计算 ample 集. 但是在 SE-POR 中如何计算 ample 集以及如何使用 ample 集和经典的偏序约简不同. 本节只考虑不带 NEXT 算子的 SE-LTL 约简技术. 在下一节中，将介绍带 NEXT 算子的 SE-LTL 约简技术.

在经典的偏序约简技术中，总是选择 ample 集来扩展一个状态，但是在 SE-POR，选择 ample 集或者 enable 集来扩展一个状态需要考虑当前状态的验证是否是针对 SE-BA 的状态部分. 除此之外，SE-LTL 包含带事件的原子命题，在计算 ample 集的时候，需要考虑面向事件的可见操作选择. 在 4.1 节中给出嵌入偏序约简的 SE-BA 和 LKS 同步乘，同步乘展示了如何在其构造过程中使用状态部分来引导偏序约简的使用. 4.2 节给出了面向 SE-LTL 的 ample 集计算.

4.1 带偏序约简的同步乘

带偏序约简的同步乘定义如下：

定义 3(带偏序约简的同步乘). 对于一个 LKS $\mathcal{M}=(S,Init,P,Act,T,\mathcal{L}_s)$和一个 SE-BA $\mathcal{B}=\{Q,\Sigma,\mathcal{T},I,F\}$. $\mathcal{M}$和$\mathcal{B}$的带偏序约简的同步乘定义为：$\mathcal{M}\otimes_s\mathcal{B}=\{Q_p,\Sigma,\mathcal{T}_p,I_p,F_p\}$，其中$Q_p,I_p,F_p$的定义同$\mathcal{M}\otimes\mathcal{B}$；$\mathcal{T}_p\subseteq Q_p\times Q_p$为迁移的集合，$((s,q),(s',q'))\in Q_p\times Q_p$为$\mathcal{M}\otimes_s\mathcal{B}$的一个迁移当且仅当下面两个条件之一满足：

(1) $\exists(q,\alpha,q')\in\mathcal{T}$满足$(q,\alpha,q')$为事件迁移且$\exists(s,a,s')\in enable(s)$，$\mathcal{L}_s(s)\wedge\mathcal{L}_e(a)\models\alpha$；

(2) $\exists(q,\alpha,q')\in\mathcal{T}$满足$(q,\alpha,q')$为状态迁移且$\exists(s,a,s')\in emple(s)$，$\mathcal{L}_s(s)\models\alpha$.

同步乘$\otimes_s$与$\otimes$的主要区别在于当遇到状态迁移的时候，$\otimes_s$选择$ample(s)$来扩展之后的状态，当遇到事件迁移的时候，$\otimes_s$选择$enable(s)$扩展之后的状态. 而$\otimes$总是选择$enable(s)$扩展之后的状态.

下面我们通过图 2 说明带偏序约简的同步乘在状态约简中的作用. 图 2(a)为一个 LKS 的一部分，它有三条迁移，即$enable(s_0)=\{(s_0,a,s_1),(s_0,b,s_2),(s_0,b,s_3)\}$，这里假设$ample(s_0)=\{(s_0,a,s_1)\}$. 图 2(b)为一个 SE-BA 的一部分，它有两条迁移(q_0,α,q_1)，(q_0,β,q_2). 假设同步乘包含状态(s_0,q_0)，$\otimes_s$将扩展该状态. 首先考虑 SE-BA 的迁移(q_0,α,q_1)，假设其为状态迁移，并且$\mathcal{L}_s(s_0)\models\alpha$. 根据$\otimes_s$的定义，只要

考虑 LKS 部分在 $ample(s)$ 中的迁移 (s_0,a,s_1)，生成迁移 $((s_0,q_0),(s_1,q_1))$. 其次考虑 SE-BA 的迁移 (q_0,β,q_2)，假设该迁移为事件迁移，且满足 $\mathcal{L}_s(s_0)\wedge\mathcal{L}_e(a)\not\models\beta$，$\mathcal{L}_s(s_0)\wedge\mathcal{L}_e(b)\models\beta$. 根据$\otimes_s$的定义，LKS 部分的三条迁移都要参与生成下一个状态，但是由于 $\mathcal{L}_s(s_0)\wedge\mathcal{L}_e(a)\not\models\beta$，所有只有两条迁移可以参与，即生成迁移 $((s_0,q_0),(s_2,q_2))$，$((s_0,q_0),(s_3,q_2))$. 综上所述，$\otimes_s$生成 3 条迁移，而如果是$\otimes$将会生成 5 条迁移. 所以$\otimes_s$将会减少系统的状态空间以及搜索路径的数量.

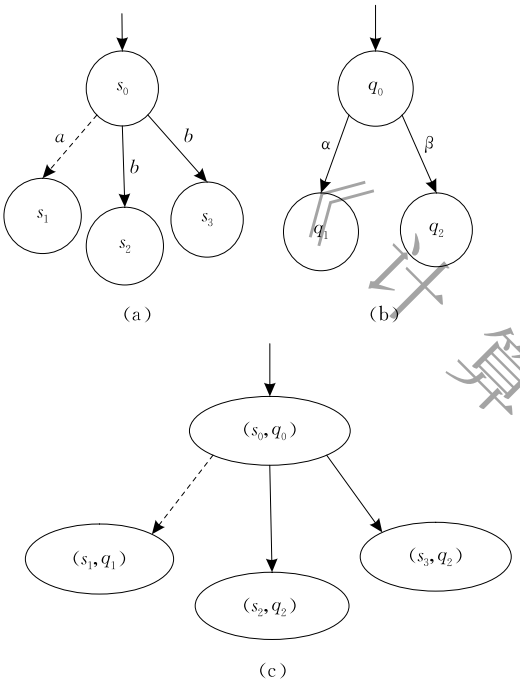

图 2 带偏序约简的同步乘示意图

4.2 Ample 集的计算

在上一节当中，给出带偏序约简的 LKS 和 SE-BA 的同步乘，但是没有给出 $ample(s)$ 的计算方法. 本节在面向 LTL 的 ample 集计算基础上，给出面向 SE-LTL 的 ample 集计算方法.

经典的偏序约简技术计算 ample 集的方法是选择 enable 集的一个子集作为 ample 集，该子集满足 C1，C2，C3，C4 四个条件（详见第 2 节）. 针对 SE-LTL 的 ample 集计算需要对 C3 条件进行更改，其它条件保持不变. 更精确地说，面向 LTL 的 ample 集计算只需考虑状态原子命题，而面向 SE-LTL 的 ample 计算需要同时考虑状态原子命题和事件. 实际上，只需要重新考虑上面的条件 C3，即需要重新针对事件来识别 LKS 中的可见操作和不可见操作. 这也是本文方法的一个重要方面. 条件 C1，C2，C4 保持不变.

在 SE-LTL 中，针对状态原子命题的可见操作的识别与 LTL 中可见操作的识别相同，即对于一个状态原子命题 p，所有不能改变该原子命题的真值的操作都是该原子命题的不可见操作，否则为可见操作.

下面介绍 SE-LTL 中针对事件的不可见操作的识别，首先定义一系列的符号，方便接下来对识别方法的描述. 对于一个事件 e，e 和 $\neg e$ 称为事件文字. 对于一个 SE-LTL 公式 ϕ，该公式的所有事件文字记为 $elit(\phi)$，该公式的所有事件记为 $evt(\phi)$. 例如 SE-LTL 公式 $\phi_1=F(e_1\rightarrow G(p\wedge\neg e_2))$，其中 e_1，e_2 为两个事件，p 为状态原子命题，那么有 $elit(\phi_1)=\{e_1,\neg e_2\}$，$evt(\phi_1)=\{e_1,e_2\}$.

本文面向事件的可见操作识别主要是面向事件文字的，已有工作[11]的可见操作识别是面向事件的，但是该方法只能应用于 SE-LTL 的一个子集，本文的方法扩大了偏序约简可以应用的 SE-LTL 属性集合，因此本文方法更具有通用性.

最简单的针对 $elit(\phi)$ 的可见操作识别是对每一个 $l\in elit(\phi)$，如果 $l=e$，其中 e 为一个事件，那么它对应的可见操作集合为 $vib(e)=\{a\in Act\mid\mathcal{L}_e(a)=e\}$，如果 $l=\neg e$，其中 e 为一个事件，那么它对应的可见操作集合为 $vib(\neg e)=Act-vib(e)$. 所以针对 SE-LTL 公式 ϕ 的可见操作集合表示为 $vib(\phi)=\bigcup_{l\in elit(\phi)}vib(l)$. 将 vib 符号扩展到集合. 对于一个事件文字集合 S，$vib(S)=\bigcup_{l\in S}vib(l)$. 所以 $vib(\phi)=vib(elit(\phi))$.

但是如果系统中映射到一个事件 e 的操作很少，那么 $vib(\neg e)$包含的操作很多. 如果在公式中包含事件文字 $\neg e$，那么该方法识别的可见操作集合可能会很大，即系统中可见操作较多，从而导致偏序约简的效果较差. 针对上面方法的缺陷，下面我们给出一个判定规则，当一个文字事件 l 满足该判定规则，可以选择 $vib(l)$或者 $vib(\neg l)$作为 l 的可见操作.

首先介绍事件文字在一个 SE-LTL 公式中覆盖的定义.

定义 4（覆盖）. 对于一个事件文字 l 和一个 SE-LTL 公式 ϕ. ϕ 的析取范式为 $\phi_1\vee\cdots\vee\phi_n$. 事件文字 l 在 ϕ 中没有被覆盖当且仅当存在一个 $i(1\leq i\leq n)$满足 $\phi_i=\phi_1\wedge\cdots\wedge\phi_k$ 并且存在一个 $j(1\leq j\leq k)$，$\phi_j=l$，否则 l 在 ϕ 被覆盖.

针对覆盖的概念，定义一个针对事件文字和 SE-LTL 公式的谓词：$inst$. 首先介绍针对时序算子

的一个符号.对一个 SE-LTL 公式 ϕ，UR(ϕ)表示 ϕ 的一个子公式的集合，集合中的子公式都是形式如 $\phi_1 \mathrm{U} \phi_2$ 或者 $\phi_1 \mathrm{R} \phi_2$ 的子公式.

定义 5.　对于一个 SE-LTL 公式 ϕ 和一个事件文字 l.谓词 $inst(\phi,l)=$false 当且仅当存在一个 ϕ 的子公式 $f \in \mathrm{UR}(\phi)$ 满足：(1) 如果 $f=\phi_1 \mathrm{U} \phi_2$，那么 l 在 ϕ_2 没有被覆盖；(2) 如果 $f=\phi_1 \mathrm{R} \phi_2$，那么 l 在 ϕ_1 中没有被覆盖，否则，$inst(\phi,l)=$true.

对于一个 SE-LTL 公式的所有的事件文字 l，如果满足 $inst(\phi,l)=$true，那么可以选择 $vib(l)$ 或者 $vib(\neg l)$作为 l 的可见操作.因为可见操作有两种选择，本文给出一种特定的选择可见操作的方法.对于一个事件 e 和一个事件文字 l，并且 $l=e$ 或 $l=\neg e$，定义如下关于 l 的符号：

$$c(l)=\begin{cases} e, & \text{如果 } l=\neg e \wedge inst(\phi,l)=\text{true} \\ l, & \text{其它} \end{cases}.$$

可见，$c(l)$表示当 l 为一个事件 e 的取反并且谓词 $inst(\phi,l)=$true，那么总是选择 e 来代替 l.对于一个事件文字集合，定义 $elit'(\phi)=\{c(l) \mid l \in elit(\phi)\}$.对于一个 SE-LTL 公式 ϕ，新的可见操作定义为 $vib(elit'(\phi))$.下面通过一个例子来说明该方法.考虑一个属性：对于系统的所有状态，如果一个进程发送一条消息到其他进程，那么该进程最终会收的一个回复消息.这个属性可以表示为 $\phi=G(send \rightarrow F\ receive)$.根据模型检验算法，首先将 ϕ 取反，得到 $\neg\phi=F(send \wedge G \neg receive)$.因为 $\neg\phi=\top U(send \wedge (\bot R \neg receive))$，可知 $inst(\neg\phi,\neg receive)=$true，所以对于 $\neg\phi$，它的可见操作集合为 $vib(\neg\phi)=\{send,receive\}$，即所有映射到事件 $send$ 和 $receive$ 的操作.

下面更新一下状态迁移和事件迁移的定义，在 3.2 节中已经给出了状态迁移和事件迁移的定义，新的定义会使状态迁移更多，从而减少事件迁移，这样可以使状态约简的效果更好.

定义 6.　对于一个从 SE-LTL 公式 ϕ 转化的 SE-BA，它的一条迁移为(q,α,q')，如果存在一个事件文字 l，满足 $l \in elit'(\phi) \bigcap \alpha$，那么该迁移称为事件迁移，否则称为状态迁移.

在这个新的定义中，事件迁移要求存在一个事件文字 l，既在 α 中又在 $elit'(\phi)$，而定义 1 中，事件迁移只要求该事件文字在 α 中.即事件迁移的要求变高了，从而事件迁移的数量就变少了.

对于一个 LKS 的任一状态 s，它的 ample 集的计算，依然是选择 $enable(s)$ 的一个子集满足条件 C1，C2，C3，C4.但是对于 C3 条件，针对事件的可见操作识别使用 $vib(elit'(\phi))$.下面给出本文 SE-LTL 方法的正确性定理.

定理 2.　对于一个从 SE-LTL 公式 $\neg\phi$ 转化而来的 SE-BA $\mathcal{B}$，和一个 LKS $\mathcal{M}$.$\mathcal{B}$的状态迁移和事件迁移由定义 6 定义，可见操作由 $vib(elit'(\neg\phi))$ 定义，那么 $\mathcal{M} \models \phi$ 当且仅当 $L(\mathcal{M} \otimes_s \mathcal{B})=\varnothing$.

定理的证明见附录 1.该定理表明带偏序约简的同步乘 $\otimes_s$ 对于验证 SE-LTL 公式是正确的.

5　SE-LTLX 的偏序约简

本节给出面向 SE-LTL 带 NEXT 算子偏序约简算法.首先刻画出一类 SE-LTLX 公式，这类公式可以直接使用第 4 节给出偏序约简方法.对于不属于该类的 SE-LTLX 公式，本节提出一种方法从 SE-BA 中抽取出一个组件，该组件可以被 SE-POR 检测.该方法基于我们之前的工作[21].

这里需要根据 NEXT 算子重新考虑谓词 $inst$ 的定义.增加一个新的条件，当满足该条件，谓词 $inst$ 为假.对于一个SE-LTL 公式 ϕ 和一个事件文字 l，如果 l 满足存在一个 ϕ 的子公式 $X\varphi$，且 l 在 φ 没有被覆盖，那么 $inst(\phi,l)=$false.刻画了能够使用 SE-POR 的 SE-LTLX 公式是基于 X-迁移的.对于一个 LKS $\mathcal{M}$ 和一个 SE-BA $\mathcal{B}$，如果$\mathcal{B}$满足对于任意一条迁移，如果该迁移是 X-迁移，那么该迁移也是事件迁移，则有 $L(\mathcal{M} \otimes \mathcal{B})=\varnothing$ 当且仅当 $L(\mathcal{M} \otimes_s \mathcal{B})=\varnothing$，即该 SE-BA 可以使用 SE-POR.对于 SE-BA 中的迁移，如果该迁移是一个 X-迁移，那么它表示该迁移要求系统的路径满足一个形如 Xφ 的公式.又因为该迁移为一个事件迁移，那么只有 LKS 中的可见操作迁移才能和该迁移同步.所以公式 Xφ 的 φ 应该在一个可见操作迁移之后被满足.下面的定理表明该方法刻画的 SE-LTLX 公式可以使用 SE-POR 进行约简.

定理 3.　对于一个 LKS $\mathcal{M}$ 和一个 SE-BA $\mathcal{B}$，该 SE-BA 是从一个 SE-LTLX 公式 $\neg\psi$ 转化而来的.如果$\mathcal{B}$满足：对于任意一条迁移，如果该迁移是 X-迁移，那么该迁移也是事件迁移，则有 $\mathcal{M} \models \psi$ 当且仅当 $L(\mathcal{M} \otimes_s \mathcal{B})=\varnothing$.

该定理的证明见附录 1.该刻画表明一部分的 SE-LTLX 公式的验证可以直接使用 SE-POR 进行约简.下面的方法适用于所有的 SE-LTLX 公式.

首先通过一个简单的例子来说明本节从 SE-BA 中抽取一个可以使用 SE-POR 进行约简的组件的动机.图 3 给出了一个从 SE-LTL 公式 X(pUq)转

化得到的 SE-BA. 其中 p,q 可以为一个状态原子命题或者一个事件. 在虚线框内的组件对应子公式 $p\mathrm{U}q$. 该组件由两个状态组成:q_1,q_2. 状态集合$\{q_1,q_2\}$在该 SE-BA 的迁移下闭的. 该组件接受所有满足 $p\mathrm{U}q$ 的路径的集合. 该组件的初始状态为 q_1. 唯一的接受状态为 q_2. 因为 $p\mathrm{U}q$ 不含有 X 算子,SE-POR 可以被用来检测所有的从状态(s_i,q_1)出发的路径的集合. 下面定义可以使用 SE-POR 约简的 SE-BA 组件.

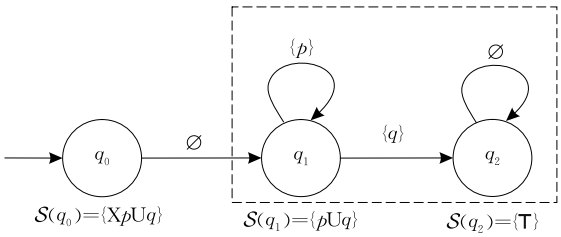

图 3 一个 BA 实例

定义 7. 对于一个 SE-BA $\mathcal{B}=(Q,\Sigma,\mathcal{T},I,F)$,它的一个 POR-组件定义为该 SE-BA 的一个子 SE-BA $\mathcal{B}_s=(Q_s,\Sigma,\mathcal{T}_s,I_s,F_s)$,满足如下条件:

(1) $Q_s\subseteq Q$,为 $\mathcal{B}_s$ 的状态集合,它是 $\mathcal{B}$ 的状态集合的一个子集;

(2) Q_s 满足在 $\mathcal{B}$ 的转移关系 $\mathcal{T}$ 下是闭的(closed),即对于任意的状态 $q\in Q_s$,对于所有从 q 出发的迁移 $(q,\alpha,q')\in\mathcal{T}$,满足 $q'\in Q_s$,即 $\forall q\in Q_s,\forall(q,\alpha,q')\in\mathcal{T},q'\in Q_s$;

(3) 迁移函数 $\mathcal{T}_s$ 满足 $\forall q\in Q_s$,如果 $(q,\alpha,q')\in\mathcal{T}$,那么 $(q,\alpha,q')\in\mathcal{T}_s$,实际上 $\mathcal{T}_s$ 是 $\mathcal{T}$ 的一个子集,它等价于 $\mathcal{T}$ 在 Q_s 上的关系;

(4) $I_s\subseteq Q_s$ 为 $\mathcal{B}_s$ 的初始状态集合,$I_s=\{q\in Q_s\mid q\in I\vee\exists q'\in Q-Q_s,\exists(q',\alpha,q),(q',\alpha,q)\in\mathcal{T}\}$;

(5) $F_s\subseteq F$ 为 $\mathcal{B}_s$ 的接受状态集合,$F_s=F\cap Q_s$;

(6) $\mathcal{B}_s$ 的验证过程可以使用 SE-POR 进行约简.

从定义中可以看出如果状态集合 Q_s 是确定的,那么 $\mathcal{T}_s,I_s,F_s$ 可以根据定义 7 的(3)、(4)、(5)确定. 所以从一个 SE-BA 中抽取 POR-组件的关键在于确定状态集合 Q_s. 该集合的确定包含两个步骤:

(1) 第一步,对于 SE-BA 中的一个状态 q,如果该状态满足对于所有的 $\psi\in\mathcal{S}(q)$,ψ 不包含 X 算子,那么将该状态加入到集合 Q_s 中.

(2) 第二步,对于所有 $Q-Q_s$ 中的状态,如果从 q 出发的所有迁移(q,α,q')满足 ① $q'\in Q_s$ 且② 如果该迁移为 X-迁移那么该迁移为事件迁移,那么将 q' 加入到状态集合 Q_s 中. 重复第二步直到没有新状态加入到 Q_s 中.

使用以上两步构造的状态集合记为 $NX(Q)$,即 $Q_s=NX(Q)$. 下面解释一下 $NX(Q)$满足定义 7 的条件(2). 在第一步的计算过程中,如果一个状态 q 在 Q_s 中,那么 $\mathcal{S}(q)$ 中不含有带 X 算子的子公式. 因为对于所有从 q 出发的迁移(q,α,q'),$\mathcal{S}(q')$中所有的公式都是 $\mathcal{S}(q)$ 中的一个子公式,所以 q' 也在 Q_s 中. 对于第二步,其中的条件①可以保证定义 7 的条件(2)成立.

引理 1. 对于一个 SE-BA $\mathcal{B}$,它的状态集合为 Q. 它的一个子 SE-BA 为 $\mathcal{B}_s=(Q_s,\Sigma,\mathcal{T}_s,I_s,F_s)$,其中 $Q_s=NX(Q)$ 且 $\mathcal{T}_s,I_s,F_s$ 通过定义 7 的(3)(4)(5)和 Q_s 确定,那么 $\mathcal{B}_s$ 是一个 POR-组件.

证明. 有 $NX(Q)$ 的构造过程可以得到 $\mathcal{B}_s$ 的所有迁移满足如果该迁移为 X-迁移,那么该迁移为事件迁移. $\mathcal{B}_s$ 是由 SE-LTL 公式 $\bigcup_{q\in I_s}\mathcal{S}(q)$ 转化而来,由定理 3 可知 $\mathcal{B}_s$ 为 POR-组件.

对于一个 LKS $\mathcal{M}=(S,Init,P,Act,T,\mathcal{L}_s)$ 和一个 SE-BA $\mathcal{B}=(Q,\Sigma,\mathcal{T},I,F)$. 图 4 给出了如何使用 POR-component 对状态空间进行约简. 图 4 中的算法对 LTL 模型检验的带嵌套深度优先搜索进行

```
Procedure emptiness(M,B)
  for each s in Init and each q in I do
    dfs1(s,q)
  terminate false
Procedure dfs1(s,q)
  hash(s,q)
  for each (q,α,q')∈T do
    work_set=∅
    if q is in POR-Component and
          (q,α,q') is a state-transition then
      work_set=ample(s)
    else
      work_set=enable(s)
    for each (s,a,s')∈work_set do
      if (s',q') is not hashed
        and L_s(s)∧E(a)⊨α then
        dfs1(s',q')
  if q is an accepting state then
    dfs2(s,q)
Procedure dfs2(s,q)
  flag(s,q)
  for each (q,α,q')∈T do
    work_set=∅
    if q is in POR-Component and
          (q,α,q') is a state-transition then
      work_set=ample(s)
    else
      work_set=enable(s)
    for each (s,a,s')∈work_set do
      if (s',q') is on the first search stack and
          L_s(s)∧E(a)⊨α then
        terminate true
      if (s',q') is not Hagged
        and L_s(s)∧E(a)⊨α then
        dfs2(s',q')
```

图 4 面向 POR-组件的偏序约简算法

修改. 关键的步骤是如何选择 *work_set*. 对于任意一个状态 q，如果该状态在 Q_s 中且 (q,α,q') 是一个状态迁移那么选择 $ample(s)$ 来作为 *work_set*，否则选择 $enable(s)$ 作为 *work_set*. 算法有两个深度优先搜索：dfs1 和 dfs2. 算法 dfs1 可以启动 dfs2，dfs2 可以终止检测过程. 在 dfs1 中被访问过的状态将会被 hashed，在 dfs2 中被访问过的状态将会被 flagged. 算法 dfs1 有一个搜索栈，该栈存储着从同步乘的初始状态到当前状态的路径. 当 dfs2 在的搜索过程中遇到一个状态在 dfs1 搜索栈中，那么表示找到一个反例路径，即一个接受环被识别. 对于每一个初始状态，dfs1(s_0,q_0) 将会被调用启动模型进行检验.

定理 4.　对于一个 LKS $\mathcal{M}$ 和一个 SE-BA $\mathcal{B}$，有 $L(\mathcal{M}\otimes\mathcal{B})\neq\varnothing$ 当且仅当图 4 中 emptiness 最终 terminate true.

定理 4 的证明类似于我们之前的工作[21] 中的定理 4. 这里不作详细介绍.

下面介绍一下本文工作与本团队已有工作的[14,21] 的区别. 图 5 给出工作对比图. 经典的偏序约简是针对 LTL 不带 NEXT 算子(LTL-X)，文献[21]的偏序约简是针对 LTL 带 NEXT 算子(LTL+X)，文献[14]的偏序约简是针对 SE-LTL 不带 NEXT 算子(SE-LTL-X)，而本文工作则是针对 SE-LTL 带 NEXT 算子(SE-LTL+X).

LTL-X (经典 POR)	LTL+X[21]
SE-LTL-X[13]	SE-LTL+X (本文工作)

图 5　本文工作与已有工作

6　工具实现和实验分析

为了评估本文方法的有效性，已实现一个基于 SPIN 模型检验器的原型系统，该系统支持 SE-LTL 模型检验以及本文的偏序约简方法 SE-POR. 原型工具称为 SE-SPIN. SPIN 使用 PROMELA[22] 语言对系统行为进行建模，使用 LTL[19] 对系统属性进行规约. SE-SPIN 扩展 SPIN，使其可以在 PROMELA 模型中定义事件. 定义事件的语法格式如下：

[(statement)]@event_name,

其中 statement 是一条 PROMELA 执行语句，event_name 是一个事件的名称. 这个事件出现当且仅当这条语句被执行. 事件只能在 PROMELA 模型中定义，在 LTL 中被引用. 除此之外在 SPIN 的 ON-THE-FLY 模型检验过程，嵌入 SE-LTL 模型检验以及 SE-POR. 图 6 给出了 SPIN 模型检验器的运行结果示例. 其中 Full statespace search for 下面列举了本次实验需要验证的属性. State vector 列举了搜索过程的信息，包括搜索深度(depth)、状态数量、迁移数量. 最后的 elapsed time 表示的是本次验证的运行时间.

```
(Spin Version 6.4.3 -- 16 December 2014)

Full statespace search for:
        never claim             + (never_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 132 byte, depth reached 9999, errors: 0
    57618 states, stored (87141 visited)
   338190 states, matched
   425331 transitions (= visited+matched)
        0 atomic steps
hash conflicts:      1097 (resolved)

pan: elapsed time 0.45 seconds
```

图 6　SPIN 运行结果示例

实验针对三个模型，所有的实验在 Intel core i5，4 GB RAM 以及 64-bit 版本的 MAC OS 系统上执行. SE-SPIN 的实现以及所有的实验数据可以在以下链接下载：https://sourceforge.net/p/se-spin/code/ci/master/tree/.

第一个模型是经典的生产消费者模型，在这个模型中有一个队列，生产者向队列的尾部插入一个产品，消费者从队列头部拿走一个产品. 这里在队尾插入一个产品以及从队列头部拿走一个产品都是事件.

实验的结果见表 1. 其中 NC，NP，SQ 分别表示消费者数量、生产者数量以及队列的长度. States NOPOR 表示不使用 SE-POR 的实验搜索状态数，States POR 表示使用 SE-POR 的实验搜索状态数. 符号'—'表示在一个小时之内没有得到实验结果. 实验结果表明使用 SE-POR 对比不使用 SE-POR 模型检验效率高出很多.

表 1　生产者消费者实验分析结果

NC	NP	SQ	States NOPOR	times	States POR	times
1	1	5	8962	0.03	8798	0.02
2	2	5	1022333	2.56	585341	0.55
2	3	5	7984123	74.30	2858123	17.50
3	3	5	—		29184813	276.00

第二个模型是哲学家就餐问题，该问题是典型的并发访问资源问题. 一群哲学家围坐在桌子周围，哲学家有两种状态，一种是思考状态，一种是就餐状态. 就餐时，哲学家需要使用他两边的筷子. 每两个

哲学家公用他们中间的筷子.为了防止死锁的发生,本文参考了文献[23]给出的一种方法对哲学家就餐问题进行建模,并使用本文方法验证无死锁属性.这里无死锁属性表示为每一个哲学家在任何时刻的未来时刻都能够执行拿起第2根筷子的操作,表示为 GFp,其中 p 表示拿起第2根筷子的操作.实验对哲学家的人数从2递增到6.实验的结果如表2所示.Number of Philosophers 是哲学家人数,States NOPOR 和 States POR 与第一个实验含义相同.从实验结果可以看出随着哲学数量的增加,无约简和有约简的方法检验的状态数量递增,但是有约简方法递增速度慢,访问的状态数量和用时都少于无约简的方法.

表2 哲学家就餐问题实验结果

Number of Philosophers	States NOPOR	times	States POR	times
2	322	0.01	210	0
3	4662	0.04	2397	0.01
4	57618	0.45	28922	0.17
5	673387	7.57	141306	1.01
6	9096561	114.00	1061221	7.67

第三个实验模型来自文献[24],该模型对 Common Object Request Broker Architecture(CORBA)中的 GIOP 协议进行检测.本文针对两个事件属性进行验证.第一个属性表达为:当一个用户向其他进程发送一条消息,这个用户最终会受到一条回复消息.这里称该属性为 S-R 属性,可以使用 SE-LTL 表达为 $G(send \rightarrow (F\ receive))$,其中 *send* 表示发送一条消息,*receive* 表示接收一条消息.第二个属性表达 GIOPCLient 永远接收不到反应信息在之前的请求被取消的情况下,用 SE-LTL 表达为:$G(\neg t \rightarrow X((\neg p U t) \vee G \neg p))$,其中 t 为一个状态原子命题,表示请求被取消,p 为一个事件,表示接受到反应信息.在文献[24]中,采用在模型中加入标签来表示一个事件的执行完毕.这种方法是完全基于状态的.表3给出了 GIOP 的实验结果.其中 State-based POR,SE-based, SE-based POR 分别表示完全基于状态的模型检验并使用偏序约简(文献[24]的方法),状态事件模型检验不带偏序约简,带偏序约简的模型检验方法.另外 *** 表示该处实验无法做,因为完全基于状态的模型检验的偏序约简无法应用于带 NEXT 算子的公式,而属性2是带 NEXT 算子的公式.本文选择了其中3个模型进行分析:Giop1, Giop2, Giop3.实验的结果使用一个对(pair)进行表示,即(状态数,运行时间/s).可以看出 SE-POR 的效率最高,比带偏序约简的完全基于状态的模型检验效率也高出很多.不带偏序约简的状态事件模型检验效率最低.

表3 GIOP 实验结果分析

Models	Property	State-based POR	SE-based	SE-based POR
Giop1	属性1	(450006, 5.2)	(1195940, 26.4)	(196472, 1.93)
Giop2	属性1	(161858, 2.74)	(651231, 26.4)	(62054, 1.09)
Giop3	属性1	(261200, 3.76)	(1271334, 46.5)	(49473, 0.82)
Giop3	属性2	***	(2836713, 107)	(2594527, 57.2)

需要说明的是本文方法的约简效率和系统模型的结构相关性非常大,这也是经典偏序约简方法的一个特性.当系统中相互依赖的操作较多时,偏序约简方法的效率较低,当相互依赖操作较少时则偏序约简方法的效率较高.所以当系统中不同进程之间下相互依赖操作较少,即使系统规模较大,偏序约简也能够将其约简到可接受的状态空间范围内.因为偏序约简在并发程序验证中的有效性,国际形式化验证顶级会议(Computer-Aided Verification,CAV)在2014年授予偏序约简发明人 CAV award.

7 结论和未来研究方向

本文结合 SE-LTL 模型检验和偏序约简技术提高状态事件系统的验证效率.SE-POR 的核心是将偏序约简技术嵌入到 SE-BA 和 LKS 同步乘的构造过程中,以及面向事件的可见操作识别中.本文通过考虑从 SE-LTLX 转化得到的 SE-BA 的结构,将 SE-POR 技术扩展到 SE-LTL 的所有公式的验证过程中.初步的验证结果表明本文的偏序约简方法比完全基于状态的偏序约简以及不带偏序约简的状态事件模型检验效率都高.如何选择可见操作以提高约简效率是本文的未来研究工作.除此之外,我们考虑将本文技术实现到符号模型检验(Symbolic Model Checking)中以及形式化建模语言 Event-B[25-26] 规约的模型验证过程中.

参 考 文 献

[1] Clarke E M, Emerson E A. Design and synthesis of synchro-

nization skeletons using branching-time temporal logic//Proceedings of the Conference on the Logic of Programs. New York, USA, 1981: 52-71

[2] Jhala R, Majumdar R. Software model checking. ACM Computing Surveys, 2009, 41(4): 1-21

[3] Queille J-P, Sifakis J. Specification and verification of concurrent systems in CESAR//Proceedings of the International Symposium on Programming. Torino, Italy, 1982: 337-351

[4] Chaki S, Clarke E, Grumberg O, et al. An expressive verification framework for state/event systems//Proceedings of the 5th International Conference on Integrated Formal Methods (IFM). Eindhoben, The Netherlands, 2004: 1-17

[5] Chaki S, Clarke E, Ouaknine J, et al. Concurrent software verification with states, events, and deadlocks. Formal Aspects of Computing, 2005, 17(4): 461-483

[6] Clarke E M, Grumberg O, Peled D A. Model Checking. Cambridge, USA: MIT Press, 1999

[7] Huth M, Ryan M. Logic in Computer Science. Modelling and Reasoning About Systems. New York, USA: Cambridge University Press, 2004

[8] Godefroid P, Wolper P. A partial approach to model checking. Information and Computation, 1994, 110(2): 305-326

[9] Peled D. All from one, one for all: On model checking using representatives//Proceedings of the International Conference on Computer Aided Verification (CAV1993). Elounda, Greece, 1993: 409-423

[10] Valmari A. Stubborn sets for reduced state space generation//Proceedings of the International Conference on Applications and Theory of Petri Nets and Concurrency. Bonn, Germany, 1991: 491-515

[11] Benes N, Brim L, Buhnova B, et al. Partial order reduction for state/event LTL with application to component-interaction automata. Science of Computer Programming, 2011, 76(10): 877-890

[12] Benes N, Brim L, Cerná I, et al. Partial order reduction for state/event LTL//Proceedings of the 7th International Conference on Integrated Formal Methods (IFM 2009). Düsseldorf, Germany, 2009: 307-321

[13] Babiak T, Kretınsky M, Rehak V, Strejcek J. LTL to Büchi automata translation: Fast and more deterministic//Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS2012). Tallinn, Estonia, 2012: 95-109

[14] Kan S, Huang Z, Chen Z. Partial order reduction for state/event systems//Proceedings of the 18th International Conference on Formal Engineering Methods (ICFEM 2016). Tokyo, Japan, 2016: 329-345

[15] Sun J, Liu Y, Dong J S, Wang H H. Specifying and verifying event-based fairness enhanced systems//Proceedings of the 10th International Conference on Formal Engineering Methods (ICFEM 2008). Kitakyushu-City, Japan, 2008: 5-24

[16] Das S, Mohanty R, Dasgupta P, Chakrabarti P P. Synthesis of system verilog assertions//Proceedings of the Conference on Design, Automation and Test in Europe: Designers' Forum. Munich, Germany, 2006: 70-75

[17] Fathabadi A S, Butler M J, Rezazadeh A. Language and tool support for event refinement structures in Event-B. Formal Aspects of Computing, 2015, 27(3): 499-523

[18] Holzmann G K. The model checker SPIN. IEEE Transactions on Software Engineering, 1997, 23(5): 279-295

[19] Holzmann G J. The SPIN model checker: Primer and reference manual. Boston, USA: Addison-Wesley, 2003

[20] Gastin P, Oddoux D. Fast LTL to Büchi automata translation //Proceedings of the International Conference Computer Aided Verification (CAV2001). Paris, France, 2001: 53-65

[21] Kan S, Huang Z, Chen Z, et al. Partial order reduction for checking LTL formulae with the next-time operator. Journal of Logic and Computation, 2016, 27(4): 1095-1011

[22] Holzmann G J. Design and Validation of Computer Protocols. Upper Saddle River, USA: Prentice Hall, 1990

[23] Baier C, Katoen J-P. Principles of Model Checking. Cambridge, USA: MIT Press, 2008

[24] Kamel M, Leue S. Formalization and validation of the General Inter-ORB Protocol (GIOP) using PROMELA and SPIN. International Journal on Software Tools for Technology Transfer, 2000, 2(4): 394-409

[25] Abrial J-R. Modeling in Event-B: System and Software Engineering. New York, USA: Cambridge University Press, 2010

[26] Fathabadi A S, Butler M J, Rezazadeh A. Language and tool support for event refinement structures in Event-B. Formal Aspects of Computing, 2015, 27(3): 499-523

附录 1.

(1)定理 2 的证明

证明.　对于一个状态 s,如果存在一条迁移$(s,a,s')\in ample(s)$,那么同样表示为 $a\in ample(s)$,表示在状态 s,操作 a 可以执行.首先我们解释一下定义 2 的证明思路.根据定理 1, $\mathcal{M}\models\phi$ 当且仅当 $L(\mathcal{M}\otimes\mathcal{B})=\varnothing$,所以为了证明定理 2,我们证明在 $\mathcal{M}\otimes\mathcal{B}$ 中存在一条接受路径当且仅当在 $\mathcal{M}\otimes_s\mathcal{B}$ 中存在一条接受路径,该证明包含两个方向.方向 1:如果在 $\mathcal{M}\otimes_s\mathcal{B}$ 中存在一条接受路径那么在 $\mathcal{M}\otimes\mathcal{B}$ 也存在一条接受路径,该方向的证明很明显,因为 $\mathcal{M}\otimes_s\mathcal{B}$ 的行为是 $\mathcal{M}\otimes\mathcal{B}$ 的一个子集,即 $L(\mathcal{M}\otimes_s\mathcal{B})\subseteq L(\mathcal{M}\otimes\mathcal{B})$.这里我们主要证明另外一个方向.

假设 $\sigma_p=\langle(s_0,q_0),(s_1,q_1),\cdots\rangle$ 为 $\mathcal{M}\otimes\mathcal{B}$ 中的一条接受路径,这条路径在 $\mathcal{M}$ 上的映射记为 $Pjm(\sigma_p)=\langle s_0,a_0,s_1,\cdots\rangle$.

假设 $\sigma=Pjm(\sigma_p)$，所有的证明包含两个步骤：

① 基于 σ，在 $\mathcal{M}$ 中找到一条路径 σ'；

② 证明在 $\mathcal{M}\otimes_s\mathcal{B}$ 中存在一条接受路径 σ'_p 满足它在 $\mathcal{M}$ 上的映射为 σ'，即 $Pjm(\sigma'_p)=\sigma'$.

对于同步乘的一个迁移 $((s_i,q_i),(s_{i+1},q_{i+1}))$，该迁移在 $\mathcal{M}$ 和 $\mathcal{B}$ 上映射分别为 (s_i,a_i,s_{i+1}) 和 (q_i,α_i,q_{i+1}). 如果 (q_i,α_i,q_{i+1}) 为一条事件迁移，那么称 (s_i,a_i,s_{i+1}) 与一条事件迁移同步，否则称 (s_i,a_i,s_{i+1}) 与一条状态迁移同步. 一条与事件迁移同步的迁移称为事件敏感迁移. 如果一条 LKS 迁移和一个状态迁移 (q_i,α_i,q_{i+1}) 同步，但是 α_i 包含一个事件文字，那么称该 LKS 迁移为事件相关迁移(定义 6 表明即使一个迁移为状态迁移，它也有可能包含一个事件文字).

下面给出 σ' 的构造过程. 假设 $\sigma^i=\langle s_i,a_i,s_{i+1},\cdots\rangle$ 是 σ 的第 i 个后缀. 考虑如下对 σ^i 的更新操作：

① 如果迁移 (s_i,a_i,s_{i+1}) 与 $\mathcal{B}$ 中的一条事件迁移同步那么保持 σ_i 不变.

② 如果迁移 (s_i,a_i,s_{i+1}) 没有与一条事件迁移同步但是该迁移在 $ample(s_i)$ 中，那么保持 σ^i 不变.

③ 如果迁移 (s_i,a_i,s_{i+1}) 没有与一条事件迁移同步且该迁移不在 $ample(s_i)$ 中，那么在后缀 σ^i 中找到第一个操作 a_j 满足 $a_j\in ample(s_i)$.

(a) 如果存在这样的一个操作 a_j，那么有 $\sigma^i=s_i,a_i,s_{i+1},\cdots,s_j,a_j,s_{j+1},\cdots$. 根据偏序约简的 C2 条件可知，操作 a_j 独立于所有的操作 $a_i,a_{i+1},\cdots,a_{j-1}$. 我们可以更新路径 σ^i，将操作 a_j 移到 s_i 之后，即 $\sigma^i=\langle s_i,a_j,s'_i,a_i,s'_{i+1},a_{i+1},s'_{i+2}\rangle$；

(b) 如果不存在这样一个操作 a_j，即 $\sigma^i=\langle s_i,a_i,s_{i+1},\cdots\rangle$，其中对于 $k\geq i$，$a_k\notin ample(s_i)$. 根据偏序约简的 C2 条件可知，$ample(s_i)$ 的操作独立于所有的操作 $a_i,a_{i+1},\cdots$. 我们可以对 σ^i 作如下更新：在 s_i 首先执行 $ample(s_i)$ 中的任一操作 a，然后执行操作序列 $a_i,a_{i+1},\cdots$，即更新后的 σ^i 为

$$\sigma^i=s_i,a,s'_i,a_i,s'_{i+1},a_{i+1},s'_{i+2},\cdots.$$

偏序约简的条件 C2，C3 保证更新后的 σ^i 与原路径是 stutter-等价的. 偏序约简的 C4 保证在(b)中操作 a_i 最终一定会被执行.

路径 σ' 可以通过如下方法构造：① 初始时设置 $\sigma'=\sigma$，$i=0$；② 用以上的方法更新 σ'^i；③ 将 i 的值加 1. 然后跳到② 继续执行. 最后更新完毕之后可以得到 σ' 和 σ 是 stutter-等价的，且它们具有相同的可见操作序列. 假设它们的可见操作序列为 $vs=a_{v0},a_{v1},a_{v2},\cdots$. 我们定义一个映射 f 从 σ 中的迁移到 σ' 中的迁移. 假设 t 为 σ 的一条可见迁移，该迁移标记操作 $a_{vi}\in vs$. 映射 $f(t)$ 为 σ' 中的同样标记 a_{vi} 的迁移.

下面介绍如何构造 σ'_p. 路径 σ 满足 $\mathcal{S}(q_0)$ 中所有时序公式的合取. 为了能够证明 σ'_p 在 $\mathcal{M}\otimes_s\mathcal{B}$ 中存在，需要证明 ① σ' 也满足 $\mathcal{S}(q_0)$，且 σ 中的任意一条迁移为事件敏感迁移当且仅当 σ' 中的 $f(t)$ 在 σ'_p 中也为事件敏感迁移. 假设 $\mathcal{S}(q_0)$ 的合取为 ϕ.

σ 的迹表示为 $\pi(\sigma)=A_0^{n0}A_1^{n1}\cdots$，其中 $ni>0$. 因为 σ' 与 σ 的迹为 stutter-等价，所以 $\pi(\sigma)=A_0^{n0\prime}A_1^{n1\prime}\cdots$，其中 $ni'>0$. 这里我们使用一个技巧来表示迹. 对于任意一个标记了可见操作的迁移 (s,a,s')，即使 $\mathcal{L}_s(s)=\mathcal{L}_s(s')$，我们仍然使用不同的符号来表示它们. 例如，如果 $\mathcal{L}_s(s)=A_k$，那么即使 $\mathcal{L}_s(s)=\mathcal{L}_s(s')$，我们仍然表示 $\mathcal{L}_s(s')=A_{k+1}$. 我们通过对 ϕ 的结构化归纳来证明 σ' 满足 ϕ. 归纳假设针对两条路径 σ_1,σ_2 和一个 SE-LTL 公式 φ，如果 σ_1,σ_2 满足如下三个条件，那么如果 $\sigma_1\models\varphi$，我们有 $\sigma_2\models\varphi$.

K1. σ_1,σ_2 是 stutter-等价的，且它们有相同的可见操作序列.

K2. 如果 l 为 φ 中一个没有被覆盖的事件文字，且 σ_1 中的第一条迁移 t 是满足事件文字 l 的事件敏感迁移，那么 σ_2 中的第一条迁移为 $f(t)$.

K3. 如果 l 为 φ 中一个被覆盖的事件文字，且 σ_1 中的第一条迁移 t 是满足事件文字 l 的事件相关迁移，那么：① 如果 t 被标记为不可见操作，那么 σ_2 的第一条迁移为不可见操作；② 如果 t 被标记可见操作，那么 σ_2 的第一条迁移为 $f(t)$ 或者是不可见操作.

条件 K2，K3 是为了证明可见操作识别的正确性. 初始时，两条路径 σ,σ' 以及 SE-LTL 公式 ϕ 明显满足以上归纳假设. 假设 $\sigma'=\langle s'_0,a'_0,s'_1,a'_1,s'_2,\cdots\rangle$.

基础：如果 ϕ 是一个无事件文字的 SE-LTL 公式，即它为一个 LTL 公式，因为 σ,σ' 为两个 stutter-等价路径，所以 σ' 满足 ϕ. 如果 $\phi=e$ 或者为 $\neg e$，其中 e 为一个事件，如果 σ 的第一个迁移为事件敏感迁移，那么根据条件 K2 可知，σ' 的第一个迁移标记了同样的可见操作，所以 σ' 满足 ϕ. 如果 σ 的第一个迁移不是事件敏感迁移，假设与该迁移同步的 SE-BA 迁移为 (q,α,q')，有 $\alpha\cap elit'(\phi)=\varnothing$. 根据 K3，$\sigma'$ 的第一个迁移 t' 为不可见迁移或者 $f(t)=t'$. 这两种情况，我们都有 $\sigma'\models\phi$.

归纳：对于 $\phi=\phi_1\wedge\phi_2$ 或者 $\phi=\phi_1\vee\phi_2$ 这两种情况，很容易根据 ϕ_1 或 ϕ_2 的归纳假设得到. 下面主要考虑两种情况：$\phi=\phi_1\mathrm{U}\phi_2$ 和 $\phi=\phi_1\mathrm{R}\phi_2$.

当 $\phi=\phi_1\mathrm{U}\phi_2$. 因为 σ 满足 ϕ，假设后缀 σ^k 满足 ϕ_2 并且对于 $0\leq i\leq k-1$，σ^i 满足 ϕ_1. 假设 $\mathcal{L}_s(s_k)=A_t$. 我们将会找到一个自然数 m，使 σ^m 满足 ϕ_2 并且对于 $0\leq i\leq m-1$，σ^i 满足 ϕ_1. 有两种情况需要考虑：

(a) 如果 (s_k,a_k,s_{k+1}) 是一个事件敏感迁移并且 $f(s_k,a_k,s_{k+1})=(s'_j,a'_j,s'_{j+1})$. 那么 $m=j$. 明显可知 σ^k,σ'^m 和 ϕ_2 满足 K1，K2，K3，所以 σ'^m 满足 ϕ_2.

(b) 如果 (s_k,a_k,s_{k+1}) 不是一个事件敏感迁移，设 $m=n'_0+\cdots+n'_{t-1}+1$. 根据可见操作识别规则，不存在一个事件文字，(s_k,a_k,s_{k+1}) 满足该文字并且该文字在 ϕ_2 没有被覆盖. 所以 σ_k,σ'^m 以及 ϕ_2 满足 K1，K2，K3，所以 σ'^m 满足 ϕ_2.

下面我们来考虑对于 $0\leq i\leq m-1$，如何证明 $\sigma'^i\models\phi_1$. 我们需要找到 σ 的一个后缀，该后缀满足 ϕ_1 并且该后缀和 σ'^i 满足 K1，K2，K3. 有两种情况需要考虑：

(a) 如果 (s'_i,a'_i,s'_{i+1}) 标记可见迁移并且 $f(s_j,a_j,s_{j+1})=(s'_i,a'_i,s'_{i+1})$，那么 σ'^i,σ^j 和 ϕ_1 满足 K1，K2，K3. 所以 $\sigma'^i\models\phi_1$.

(b) 如果 (s'_i,a'_i,s'_{i+1}) 标记不可见迁移，假设 $j=\min(\{o\geq$

$k \wedge (s'_o, a'_o, s'_{o+1})$ is visible$\}$)，其中 min 求一个集合中最小元素. 假设 $t'=(s_b, a_b, s_{b+1})$ 为 σ 中的一条迁移且 $f(t')=(s'_j, a'_j, s'_{j+1})$. 如果 t' 不是一个事件敏感迁移那么 σ'^i, σ^b 和 ϕ_1 满足 K1,K2,K3. 所以 $\sigma'^i \models \phi_1$. 如果 t' 是一个事件敏感迁移，那么根据 σ' 构造规则可知 (s_{b-1}, a_{b-1}, s_b) 必然标记不可见迁移，所以 σ'^i, σ^{b-1} 和 ϕ_1 满足 K1,K2,K3. 所以 $\sigma'^i \models \phi_1$.

当 $\phi=\phi_1 \mathrm{R} \phi_2$，这种情况的证明与 $\phi=\phi_1 \mathrm{U} \phi_2$ 的证明相似.

从构造过程中我们可以看出 σ' 满足 ϕ 并且对于 σ 中每一条事件敏感迁移 t，$f(t)$ 在 σ' 也是敏感迁移. 所以如果在 $\mathcal{M} \otimes \mathcal{B}$ 中有一条接受路径，那么在 $\mathcal{M} \otimes_s \mathcal{B}$ 中也存在一条接受路径.

(2) 定理 3 的证明

证明.　这个定义的证明是定理 2 证明的继续，即归纳步骤中在考虑 $\phi=\phi_1 \mathrm{U} \phi_2$ 和 $\phi=\phi_1 \mathrm{R} \phi_2$ 之后，我们考虑 $\phi=\mathrm{X}\psi$.

如果 $\phi=\mathrm{X}\psi$，那么 σ 的第一个迁移 $t=(s_0, a_0, s_1)$ 是一条事件敏感迁移. 所以 σ' 的第一条迁移为 $f(t)$. 除此之外 σ_1 和 σ'_1 是 stutter-等价的并且它们有相同的可见操作序列. 如果 (s_1, a_1, s_2) 也是一个事件敏感迁移那么 σ' 的构造方法可知 σ' 的第 2 个迁移为 $f((s_1, a_1, s_2))$，所以 σ^1, σ'^1 和 ψ 满足 K1,K2,K3. 因此 $\sigma'^1 \models \psi$. 如果 (s_1, a_1, s_2) 不是一个事件敏感迁移，那么不存在一个事件文字，该事件文字被 (s_1, a_1, s_2) 并且它在 ψ 没有被覆盖. 所以 σ^1, σ'^1 和 ψ 满足 K1,K2,K3. 因此 $\sigma'^1 \models \psi$.

XIE Jian, Ph. D. candidate. His research interests include safety analysis, model checking and theorem proving.

KAN Shuang-Long, Ph. D. His research interests include model checking and theorem proving.

HUANG Zhi-Qiu, professor, Ph. D. His research interests include safety-critical software development and formal methods.

WANG Fei, Ph. D. candidate. His research interests focus on software traceability.

YANG Zhi-Bin, associate professor, Ph. D. His research interests include safety-critical real-time system, formal verification, AADL, and synchronous languages.

LI Wei-Wei, associate researcher, Ph. D. Her research interests include software data mining, machine Learning.

Background

Applying existing model checking techniques in modular or component-based software is complicated by two important factors: (1) the difficulties in modeling the behavior of software and specifying meaning properties, and (2) the perennial state-space explosion problem. Modeling techniques based on annotated finite state automata might be either state-based or event-based. Although these two frameworks are interchangeable converting from one representation to the other often results in a significant enlargement of the state space. In addition, both approaches are not practical when it comes to modular software.

Existing work provided a framework in which both states and events can be expressed. In the framework, the behavior of a system is modeled by Labeled Kripke Structures. The property specification logic is State/Event Linear Temporal Logic. But there is no efficient partial order reduction for this framework.

This paper presents a novel approach to integrate partial order reduction into the verification of state/event systems. The experimental results show that our approach is more efficient than other approaches.