模型检验软件体系结构研究与进展*)

张鹏程[1] 李必信[1,2] 周 宇[1]
(东南大学计算机科学与工程学院 南京 210096)[1]
(南京大学计算机软件新技术国家重点实验室 南京 210093)[2]

摘 要 软件体系结构经过10多年的发展,在体系结构的基础理论、体系结构风格、体系结构描述语言和体系结构建模等方面的研究取得了一系列可喜的成就。目前,就软件体系结构的分析、评价、测试和验证的研究也在如火如荼地进行中。模型检验是一种基于自动机理论的形式验证方法,采用穷举状态空间的方式来证明系统的模型是否满足要验证的属性。同传统的测试和验证的手段相比,模型检验有其自身的优势。为此,越来越多的研究人员正在将模型检验技术应用到软件体系结构的分析和验证中。本文调查了模型检验技术在软件体系结构中的应用现状,剖析了影响软件体系结构模型检验的因素,给出了软件体系结构模型检验的未来主题。

关键词 软件体系结构,模型检验,应有属性,形式验证

Research and Development of Model Checking Software Architecture

ZHANG Peng-Cheng[1] LI Bi-Xin[1,2] ZHOU Yu[1]
(Department of Computer Science and Engineering, Southeast University, Nanjing 210096)[1]
(State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210093)[2]

Abstract In the past ten years, software architecture has made a great step in the research of foundation theory, SA style, architecture description language (ADL), SA model etc. Currently, many people are doing the research of analysis, evaluation, testing and verification. As a formal verification technique based on automata theory, model checking verifies system's attributes by enumerating all its state spaces. Compared with the traditional testing and verification techniques, it has its own advantages. Therefore, more and more researches are applying model checking to analyze and verify quality attributes of SA. In this paper the states of model checking SA are first investigated, the factors affecting to it are analyzed in the second, and the future research topics are debated in the final.

Keywords Software architecture, Model checking, Desired properties, Formal verification

1 引言

软件体系结构(SA,Software Architecture)是20世纪90年代兴起的一种软件开发技术[1]。SA是现代软件开发过程中的一个至关重要的部分,一方面它是连接软件需求和实现的桥梁,另一方面它是实现软件开发在设计级重用的必然产物。基于SA的家族式产品开发正越来越受到软件从业人员的推崇,故SA的设计必须既完全正确又完全精确地满足软件需求规约,即要实现软件需求规约提到的功能性属性和非功能性属性,同时要考虑如何保证软件实现是符合SA的描述的。而且,由于SA是家族式软件产品的共同设计基础,SA中任何微小的错误都有可能传播到家族的所有成员中,这就可能给软件开发的后期维护带来更大的麻烦。可见,如何减少SA中包含的错误,显得特别重要。

模型检验(Model Checking)技术是最早由美国卡耐基梅隆大学的Clarke等人提出的一种基于自动机理论的形式验证方法[2]。其基本思想是将系统表示成为自动机模型;将系统要验证的属性用某种逻辑公式来表示;采用穷举状态空间的方式来证明系统的模型是否满足要验证的属性。同传统的测试和验证的手段相比,模型检验有其自身的优势,采用穷举状态的方式比传统的测试方式更能保证软件的质量;而其完全自动的方式,比基于传统验证的定理证明方式去除了人工的参与,减少了出错的可能性,也提高了软件的质量。早期模型检验技术在硬件、通讯系统、协议以及临界安全系统中已经取得了令人瞩目的成功[2],而将之用于软件和SA中也是新近形式工作者研究的热点之一[3~8]。目前,已有越来越多的人[9~11,19~22]采用模型检验的方法来验证软件体系结构的质量属性,来保证软件体系结构的质量。

2 基本术语定义

2.1 基本术语

SA定义:当前,关于SA的定义很多,不同的研究组织对SA的理解和侧重点不同,很难给出一个统一的定义。如Perry和Wolf在1992年给的定义是最早也是最简洁的,他们把SA看成是由处理元素、数据元素和连接元素组成的集合[23];1996年,由Shaw和Garlan给出的SA定义是最受关注的,它说明SA是软件设计过程的一个层次,既要刻画SA结构,又要刻画未来软件系统的功能和性能需求[1];由

*)本研究受到国家自然科学基金(批准号:60473065)和计算机软件新技术国家重点实验室(南京大学)课题资助。**张鹏程** 博士研究生,研究方向是软件体系结构的分析、测试和验证;**李必信** 博士,教授,研究方向是软件分析与测试、程序切片技术和经验软件工程等;**周 宇** 硕士研究生,研究方向是软件体系结构的分析、测试和验证。

Booch, Rumbaugh 和 Jacobson 在 1999 年结合 UML 的相关知识,并引入 SA 风格来指导 SA 中构件元素及其接口的组合问题[24];2003 年,Bass,Clements,和 Kazman 给出了强调软件元素、外部可见属性和它们之间的关系的一种新的定义。由于 SA 定义很多,它们的侧重点各不相同,这给关于 SA 的研究(如建模、分析、测试和验证)带来了很多困难。

体系结构描述语言(ADL,Architecture Description Language):ADL 试图采用基于数学和逻辑的形式化方式来描述体系结构模型,目的是为了提供一种规范化的体系结构描述,从而方便软件开发人员在体系结构层进行建模、分析、测试和验证等工作。但由于当前的 ADL 种类很多,风格不一样,强调的重点和适应的领域也不同,很难形成统一的标准,所以在 SA 层进行建模、分析、测试和验证还存在很多问题。

模型检验基本思想:模型检验是一种自动的、基于模型的、验证属性的方式[2]。其中待验证的属性是基于时态逻辑。时态逻辑的基本思想是模型中表示的公式的值不是如命题和谓词逻辑中静态不变的。相反,时态逻辑中的模型包括几种状态,即公式在某些状态是真,在某些状态是假,也就是说公式随着系统状态的演化能够改变其值。在模型检验中,模型 M 是转换系统,属性是时态逻辑中的公式。为了验证系统满足某种属性,必须按以下 3 个步骤进行:(1)使用某种模型检验器的描述语言来模拟系统,从而得到系统的一个模型 M;(2)使用同样的模型检验器的规约语言来形式化系统的属性,最终得到时态逻辑公式 ϕ;(3)根据输入 M 和 ϕ 来运行系统。模型检验器会得到正确或错误两种结论。前者表示系统模型 M 是满足相应的属性 ϕ 的,即 $M\models\phi$;后者则会产生系统行为违背这种属性的轨迹。这种自动生成的反例为系统在设计和调试阶段改正错误提供了非常重要的信息。

属性的表示逻辑:系统属性有 3 种主流的表示逻辑,包括计算树逻辑(CTL,Computation Tree Logic)、线性时态逻辑(LTL,Linear Temporal Logic)以及模态命题 u-演算(u-Calculus)[2]。

主流模型检验工具:当前的主流模型检验工具有卡耐基梅隆大学的用于符合模型检验的 SMV[16]、Bell 实验室开发针对并发进程的 Spin[15]、英国伦敦帝国学院的针对并发程序的 LTSA[17](Labeled Transition System Analyzer)和 Illinois 大学的基于重写逻辑语义的工具 Maude[26]。

安全性(Safety):指系统不应该达到的危险状态,即坏的事情是从来不会发生的。如无死锁即是系统的一种安全属性。

活性(Liveness):指系统应该达到的正确状态,即好的事情最终是会发生的。如系统中某进程进行了一个请求,该请求总能够得到回应。

公平性(Fairness):指如何保证系统的资源能够公平地得到各个任务的使用,不会导致某些任务长期不能得到响应,即好的事情能否无限重复地发生。

一致性(Consistency):指对于同一个 SA,不同的人员有着不同的看法,就会产生不同的软件体系结构视图。如何采用形式化的方法来验证这些不同视图之间的一致性问题。

SA 模型检验:指采用模型检验的方法来验证 SA 的相关质量属性,如安全性、活性和公平性以及一致性等。由于早期的设计阶段是软件生命周期中错误发生频率最多的时期,而软件开发的基本原则是错误越早被发现,相应的改正错误所花的代价就会越少[13]。故本文对 SA 进行的验证实际上就是对 SA 规约的验证。

2.2 比较框架概述

本节提出一种比较现有的在 SA 层应用模型检验技术来验证规约的框架,旨在总结当前在 SA 层应用模型检验技术存在的问题和发展趋势,试图能够找到一种系统化和可重用的方式在 SA 层应用模型检验技术对相关属性进行验证。表 1 给出了从 SA 的表示方式、关注验证的属性、属性的表示逻辑、支持工具、缩减状态空间的方式以及转换的自动化程度等方面对现有的主流的在体系结构层进行模型检验技术的分析和比较框架。

表 1 SA 模型检验技术的比较框架

框架元素	描述
体系结构模型的描述方式	用何种方式如具体哪一种 ADL 或是用 UML 来描述系统
待验证的属性	系统验证时关注的是何种类型的属性,如安全性、活性等
属性的表示逻辑	具体用 LTL、CTL 还是 u-演算来表示系统的属性
支持工具	相关的模型检验工具支持,如 Spin、SMV,还是自研的工具
缩减状态空间的方式	具体是哪种方式来解决状态空间爆炸的问题
转换的自动化程度	从系统和属性的描述到模型检验的输入是否是自动化的过程

3 SA 模型检验基本方法概述

3.1 Tsai 方法

早在 1997 年,Tsai 等人就提出了一种增量式验证实时系统软件体系结构规约的方法[7]。针对当前没有形式体系结构规约语言用来模拟和分析复杂实时系统,试图在软件体系结构层应用模型检验的方法验证实时系统的时间属性。

(1)体系结构模型的描述方式:该方法用基于 Horn 子句逻辑的面向对象的规约语言作为体系结构的规约语言,其语法包括对象和活动帧的集合。对象的层次通过继承关系来表示,活动帧模拟为过程。由于 Horn 子句逻辑中缺少属性的继承和异常处理机制,采用变量来补充说明这两种机制。该语言支持模拟并发系统的大部分机制,如 AND 和 OR 并行、不确定性、异步和同步通讯。

(2)待验证的属性:实时系统关心的一个主要方面是验证系统满足的时间限制,故主要在软件体系结构层针对实时系统与时间属性相关的安全性、活性和公平性等属性进行了验证。

(3)属性的表示逻辑:该方法用模态 u-演算来表示实时系统时间属性。已经证明模态 u-演算可以用来形式化大部分分布式实时系统的时间属性,同时证明存在一个较小的多项式时间复杂度的模型检验算法。

(4)支持工具:该方法还没有相关工具的支持。

(5)缩减状态空间的方式:该方法将传统的模型检验算法改进为增量式算法。将整个系统的状态空间工作映射为与仅与待验证属性相关的一个子集,从而简化了状态空间的搜索。这种算法具有较低的复杂度,可推广到大的更加复杂的系统中。

(6)转换的过程及自动化程度:首先将系统规约语言转化为转换系统(Transition System),然后验证该转换系统是否满足用来表示系统时间属性的模态 u-演算公式。其中的转化过程是采取手工的方式进行的。

(7)评价:该方法是最早提出在软件体系结构层应用模型检验方法的成功应用,当中提出的增量验证方法是一种用来处理大而复杂系统的状态空间爆炸问题的有效方式。然而,由于缺乏自动化的转化过程,很难在工业界得到广泛应用。

3.2 TRACTA 方法

Giannakopoulou 等人在 1999 年提出了一种基于 TRACTA 的模型检验方法在体系结构层分析并发系统的行为[8,17]。

(1)体系结构模型的描述方式:开发了一种特定的 ADL 语言 Darwin,Darwin 可广泛用来描述分布式系统的结构以及它们的直接构建过程。Darwin 有本文和图形语法的工具支持,它用构件来描述系统,而构件是用来管理服务的执行。一般而言,系统被描述为层次构件,多个构件可以被定义成为组合构件类型的子结构。Darwin 支持多个视图,其中有行为视图(用来对 SA 进行行为分析)和服务视图(用来构建 SA)两种。每个视图都是基本结构视图的一个细化视图。

(2)待验证的属性:以安全性和活性为例进行了实践验证,还提出了一种简单而高效的方式来解决公平性,即动作优先模式(Action Priority Scheme),允许用户在分析中为系统强加不利序列条件(Adverse Scheduling Condition),从而能够进行部分搜索,而取代了对整个大系统的穷举搜索。

(3)属性的表示逻辑:本方法用 ALTL(Action LTL)来描述系统的质量属性,ALTL 是基于 LTL 并针对并发系统模型 LTS 的扩充,其中增加了描述系统活动的动作(Action),将系统动作分为 3 种类型:发送消息、接受消息和系统内部的动作。由于引入了系统内部的动作,缩减了整个系统状态空间。

(4)支持工具:TRACTA 是一个完全自动的方法,由现有的模型检验工具 LTSA(Labeled Transition System Analysis)来支持。

(5)缩减状态空间的方式:提出了一种叫 CRA(Compositional Reachability Analysis)方法来检验 SA 质量属性的通用框架。使用 CRA(Compositional Reachability Analysis)方法来对系统的行为进行增量的分析。CRA 是指以自底向上的方式由其子系统的行为逐层向上组合来得到整个系统的行为。在分析的每个中间阶段,子系统行为的内部细节被隐藏了,子系统被最小化。由于内部细节被隐藏,可观察的子系统的行为将被较少的状态空间代替。这种技术的成功之处在于尽可能地隐藏每个系统内在行为。尽管 CRA 在缩减全局系统的状态空间时发挥了很大的作用,但还受到中间状态空间爆炸的影响,即当系统快速增长的构件超过了系统自身,将会出现中间状态空间爆炸问题。在活动的上下文中加入限制,这些构件将有相对较少的状态空间。TRACTA 的优点在于它能够在 CRA 中自然地引入模型检验机制,即在上下文环境中需要时引入相应的机制时。

(6)转换的过程及自动化程度:该方法借助软件体系结构助理(SAA、Software Architecture's Assistant)将用 Darwin 描述的系统模拟为交互的有限状态机模型一带标签的转换系统(LTS,Labeled Transition System)的集合,再将 ALTL 公式转换为 Büchi 自动机,最后根据标准自动机理论的验证方法在支持工具 LTAS 中进行验证。

(7)评价:该方法和 SAA 结合起来可用于对并发和分布式系统的软件体系结构进行设计和分析。由于支持对软件体系结构的多视图细化,可用来从不同角度描述软件体系结构的质量属性。同时,改进传统的 CAR 方法在解决状态空间爆炸问题上已具备了很好的效果。

3.3 PoliS 方法

Ciancarini 和 Mascolo 于 1999 年提出了一种形式化描述软件体系结构的协调(Coordination)模型 PoliS[21]。

(1)体系结构模型的描述方式:PoliS 是一个基于嵌套的元组空间(tuple spaces)协调语言,包括元组及其空间。用 PoliS 将系统描述为层次结构,即表示为一个能够随着时间动态演化的嵌套的树形结构。

(2)待验证的属性:该方法以 C/S 软件体系结构风格为例,检验系统的活性。

(3)属性的表示逻辑:引入了一种简单的时态逻辑(PTL,PoliS Temporal Logic),它是 CTL 的一种方言,与 CTL 的主要区别在于其基于多空间的,所以时态逻辑公式的求值是在特定空间进行的,提供了一种缩减系统状态空间的方式。

(4)支持工具:针对 PoliS 开发了模型检验工具 PoliMC。

(5)缩减状态空间的方式:该方法采取对时态逻辑公式在特定空间中进行求值的方法来进行状态空间的缩减,这和 TRACTA 方法在上下文中进行验证的思想是相似的。

(6)转换的过程及自动化程度:在进行模型检验前,首先将 SA 中的构件和连接件映射为 PoliS 中的空间和元组。PoliMC 有两个输入端:其一接受用 PoliS 描述的系统规约;其二接受用 PTL 表示的系统待验证的属性。PoliMC 首先解析 PoliS 规约,生成图形来表示系统模型。其中结点显示了空间如何演化,边的变化表示元组在父空间中增加或删除。其次,PoliMC 递归地从嵌套最深的空间开始建立图形直到根空间(StartContext),同时进行信息的收集。当建立好图形以后,检验器解析 PTL 公式并建立语法树,结果仅包括了 CTL 操作符。最后,PoliMC 开始进行模型检验,其算法和标准 CTL 模型检验的思想是一致的,不同之处在于它的每个公式都是在特定的空间中进行的。

(7)评价:由于 PoliS 本质上是重写的多集合系统,所以所描述的体系结构将包括大量的不同构件,这将导致系统被表示为大而复杂的图形系统,很难建立和检验。但由于调和语言具有动态性,用来描述移动环境下的体系结构是很有前景的。

3.4 Arcade 方法

Barber 等人于 2001 年提出了一种评估 SA 的工具 Arcade(Architecture Analysis Dynamic Environment)[9],该工具将模型检验技术应用到 DRA(Domain Reference Architecture)来提供给系统分析师和软件开发者早期的关于系统安全性和活性评估的反馈。

(1)体系结构模型的描述方式:采用 DRA 来描述整个软件系统体系结构。由于软件需求包括领域需求、应用和系统范围内的非功能需求以及系统在安装时的软硬件环境需求,DRA 能显式地将领域需求从其它需求中分离开来,强调检测和改进功能性方面的需求缺陷。

(2)待验证的属性:该方法验证了软件体系结构中的安全性和活性。

(3)属性的表示逻辑:从 DRA 模型中抽取出 LTL 来表示

质量属性。

(4)支持工具:该方法以现有的模型检验工具 Spin 为支持工具。

(5)缩减状态空间的方式:主要采用以反例指导的迭代过程,不断地精化和改正原来的领域模型,而不需要对整个系统进行完全状态空间的搜索。

(6)转换的过程及自动化程度:评估体系结构的属性有两个目标:(1)能够尽量早地检测到嵌入在 SA 中的需求缺陷;(2)尽早准确地保证 SA 是系统执行的蓝图。Arcade 完全自动化模型检验过程分为以下 4 个步骤:①将 DRA 转换为 Spin 模型检验器接受的 Promela 模型;②调用 Spin 模型检验器来建立一个验证器;③调用该验证器来搜索属性背离;④如果发现了属性背离,将每个属性背离创建和表示为 ATD(Architecture Trace Diagram),从而便于系统架构师和领域专家来解释。以上过程循环进行,直到不再发现属性背离为止。由于重复使用反例来指导错误的发现和改正,故称上面的过程为反例指导的迭代方法。将模型检验工具应用到 SA 中存在两个方面障碍,即转换障碍和解释障碍。前者指模型检验需要特殊的技能,而且用手工做,消耗时间和容易出错,后者指解释模型检验结果时需要特殊的知识和技术。经过以上的自动过程和图形化的解释工具,Arcade 解决了这两方面的障碍。

(7)评价:该方法需要领域工程师、系统分析员等多方面的人参与,从而便于他们之间进行交流,更多地减少系统设计时的错误,增加了该方法的权威性。

3.5 LfP 方法

LfP(Language for Prototyping)是一个用来描述分布式嵌入式系统控制结构的形式化语言,它具有 ADL 和协调语言的双重特性,同时还是基于 UML,由 RM－ODP(Reference Model for Open Distributed Processing)提倡[22]。

(1)体系结构模型的描述方式:LfP 规约是一个层次图的集合,用来代表构件的行为。每个 LfP 模型包括两个类型的图,即 LfP－AD(LfP—Architecture Diagram)和 LfP－BD(LfP—Behavior Diagram)。前者代表系统不同的构件的组成和它们之间的关系,后者代表每个构件之间的行为契约。

(2)待验证的属性:该方法以一个实例来检验系统无死锁状态。

(3)属性的表示逻辑:用 Maude 工具支持的 LTL 来表示软件体系结构的属性。

(4)支持工具:现有基于重写逻辑语义的模型检验工具 Maude。

(5)缩减状态空间的方式:Maude 工具自身支持的。

(6)转换的过程及自动化程度:Jerad 等人于 2005 年提出了将 LfP 语言映射为重写逻辑,并用基于重写逻辑的高性能解释器 Maude 来在体系结构层验证分布式对象之间进行交互时的属性。映射方式如下:1)首先在 Maude 中为映射创建一个模块 LfP;2)将 LfP－AD 映射成重写理论中的面向对象重写理论。即将构件中的类、调停者映射成重写理论中的对象模块,将绑定映射成操作;3)将 LfP—BD 映射成模块中的条件或非条件重写规则,用来描述分布式系统的动态特性。映射完成以后,由 Maude 通过 MODEL－CHECKER 模块使用 LTL 来提供模型检验技术。当属性不满足时,它会生成反例轨迹,来帮助定位和改正错误。

(7)评价:该方法缺陷在于映射过程还不是以自动的方式完成。

3.6 Bose 方法

Bose 于 1999 年提出了一种把 SA 的 UML 模型自动转换为 Spin 模型检验器的输入[4]。针对面向调停者(Mediated)模式 SA,根据 UML 的可扩展性,将 UML 扩展适合用来验证。

表 2 软件体系结构层的模型检验方法比较

名称	体系结构模型的描述方式	待验证的属性	属性的表示逻辑	支持工具	缩减状态空间的方式	转换的自动化程度	其它优点
Tsai	拓展 Horn 逻辑子句	与时间相关的安全和活性	Computation Tree Logic	无	增量式	低	适合实时分布式
Tractar	Darwin	安全性、活性和公平性	Action Linear Temporal Logic	Labeled Transition System Analyzer	Compositional Reachability Analysis	高	适合并发分布式系统
PoliS	PoliS 调和语言	活性	PoliS Temporal Logic	PoliMC	工具自身	低	支持动态性,适合移动环境
Arcade	Domain Reference Architecture	安全性、活性	Linear Temporal Logic	Spin	反例指导的迭代过程	高	多方面人员的参与
LfP	LfP	安全性	Linear Temporal Logic	Maude	工具自身	低	适合分布式
Bose	UML	安全性	Linear Temporal Logic	Spin	工具自身	高	面向调停者模式体系结构
CHARMY 及其拓展	状态图和场景	一致性	Linear Temporal Logic	Spin	Compositional Reason	高	能隐含模型检验的复杂性,应用广泛

(1)体系结构模型的描述方式:用 UML 描述的面向调停者 SA 包括一类计算和资源构件的集合和调停者构件的集合。其中计算和资源构件的端口(Port)属性用来描述某种计算或提供的某种服务;调停者构件则包括角色和协调策略,用来协调构件在扮演角色时能满足相应的全局属性。

(2)待验证的属性:该方法以检测无死锁状态的安全性进行了验证。

(3)属性的表示逻辑:选择用 UML 活动图来模拟系统的属性。由于活动图描述了单个构件接受和发送事件的顺序,不描述构件之间的直接交换,所以比时序图更加适合用来描

述面向调停者模式体系结构的属性。

(4)支持工具:该方法以现有的模型检验工具 Spin 为其支持工具。

(5)缩减状态空间的方式:仅以 Spin 工具支持的方法。

(6)转换的过程及自动化程度:给出了 UML 模型转换为 Spin 模型检验接受的语言 Promela 的过程:1)构件之间的非直接互连的空间映射成支持异步通讯的通道;2)构件端口类和它们的行为模拟的有限状态机模型映射成 Promela 的进程;3)中介类和它们的行为模拟的有限状态机映射成 Promela 的进程;4)活动图模拟的需求的动作序列映射成 Spin 产生的轨迹。根据以上转换方法要在模型中搜索相关类型的元素并把它们转换成 Promela 声明。文中给出了一种简单的搜索算法来实现以上过程,并应用 Rational 的工具 SODA 将以上转换过程自动化。

(7)评价:该方法是针对用 UML 及其扩展机制描述的 SA 的验证的一个成功范例,尽管仅适合以非直接互连方式连接的 SA,但由于 UML 在工业界的广泛应用,所以为统一验证 SA 提供了一种切实可行性的思想。

3.7 CHARMY 方法

Inveradi 和 Muccini 等人在 2001 年开始开发出的 CHARMY(CHecking ARchitectural Model consistencY) SA 分析验证框架[3,19],在软件开发阶段的早期引入,目的是在设计和确认阶段辅助软件架构师,是迄今为止学术界和工业界最著名和成功的基于 UML 的软件系统结构验证工具。

(1)体系结构模型的描述方式:假设 SA 描述包括构件、连接件和通道(Channels)。构件是计算和存贮的单元;连接件是一种特殊的构件,用来协调构件之间的交互;通道是用来进行构件和连接件之间的交互。构件的动态性通过 UML 状态机进行描述;连接件可由场景(Scenarios)显式描述;而一组场景的集合可以用来描述构件和连接之间的交互。UML 状态机和场景包含了由 ADL 的所有能够在当前用来进行分析和验证的特性。

(2)待验证的属性:主要验证软件体系结构多视图(系统的拓扑视图和行为视图)之间的一致性问题。也就是说,要检验场景描述的事件的时态序列同时是由 Spin 产生的体系结构模型的执行路径。

(3)属性的表示逻辑:从场景中以 LTL 的形式抽象出来软件体系结构的相关质量属性。

(4)支持工具:以现有的模型检验工具 Spin 为其支持工具。

(5)缩减状态空间的方式:仅以 Spin 工具支持的方法,但在实际应用时可以组合其它方式。

(6)转换的过程及自动化程度:验证体系结构时,分为 3 个步骤进行:1)UML 状态机表示的 SA 拓扑结构转换成模型检验器 Spin 能够接受的 Promela 模型,其中要分 4 个步骤:ⅰ)构件状态图映射为 Promela 的进程 proctypes;ⅱ)用 Promela 模型来表达从场景中攫取的连接件;ⅲ)体系结构通道映射为 Promela 中的 Input 和 Output 通道;ⅳ)定义变量。2)场景描述的体系结构的行为转换成 LTL 公式或 Büchi 自动机;3)最后交给 Spin 模型检验器检验全局模型是否满足相应的 LTL 公式表示的时态属性。

(7)评价:该方法隐藏了模型检验过程的复杂性,将 Spin 作为插件嵌入 CHARMY 工具中。这样,软件工程师可以直接以体系结构模型作为输入,产生一个原型来自动分析它,将人的参与最小化地缩减。由于该工具是可嵌入的,作为插件可方便地整合到其它开发工具中,如 Sun 公司的 Eclipse 中,故可在工具界广泛应用。

表 2 给出了从体系结构模型的描述方式、待验证的属性、属性的表示逻辑、支持工具、缩减状态空间的方式以及转换的自动化程度等 7 个方面对上述 7 种基于 SA 的模型检验技术进行了详细比较。

4 总结和思考

通过上节的现状调查评估和比较,我们已经明确了 SA 模型检验的基本思想,得到了一个模型检验 SA 的通用方式,如图 1 所示。其中(1)表示获取 SA 属性的过程;(2)表示将 SA 属性转化为某种逻辑公式的过程;(3)表示逻辑公式作为某种模型检验器的输入;(4)表示获取 SA 系统规约的过程;(5)表示将 SA 规约转化为某种模型检验器的输入;(6)表示作为某种模型检验器的输入;(7)将模型检验的结果反馈给 SA,帮助发现和改正错误。以上过程可以重复迭代地完成,直到 SA 的模型满足 SA 的属性规约为止。但在实际的研究工作中,仍然会遇到很多问题和阻碍,同时还有很多因素将制约着我们的手脚。通过本文的分析和研究,我们指出了制约 SA 模型检验的因素主要来源于以下几个方面。

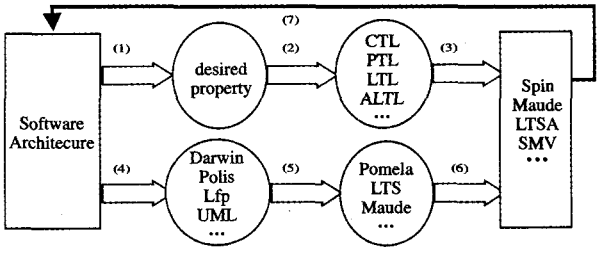

图 1 模型检验 SA 的一般过程

(1)模型检验的基本问题:从模型检验概念的提出到如今的应用,一直受到所谓的"状态空间爆炸"问题的困扰。Holzmann 指出,没有一个实践的模型检验技术不受状态空间的影响[15],构件之间的交互或数据结构的多值都会导致全局的状态空间是难以计数和不可处理的。目前,可将已有的方法分为两类[8]:一种是内部方法,即在模型检验器如 Spin、SMV 和 Maude 等内部使用算法和技术来更加有效地代表并发进程之间的转换关系,如针对同步进程的二元决策图(BDD)和异步进程的偏序缩减技术(Partial Order Reduction)。另一种是外部方法,即改进模型检验器输入技术,同时可以和内部方法组合使用。这些方法包括抽象、对称技术和组合推理技术等。改进传统的方法,使模型检验技术能够适应在 SA 环境下,是解决体系结构模型检验中状态空间爆炸问题的一种可适用方法。

(2)转换的自动程度:限制模型检验技术广泛应用的另一个原因是需要特定的理论和技术基础来掌握它。一般的软件工程师和系统分析者很难花费大量的精力来学习和掌握它,所以做成成型的工具,自动化这种转换过程,隐藏模型检验过程本身的复杂性,是一种有效和可行的办法。如 CHARMY 工具就是该方法成功的典范。

(3)SA 定义和风格:根据 CMU 软件工程研究所(SEI)收集的资料显示[13],当前关于 SA 的定义有 200 多种,其中有的定义很详细,具有很多共性,但也有一些定义很特别。我们总结出的基本上是集中在将软件体系结构定义成为构件、连接

件和限制这3个方面，将集中验证构件、连接件以及它们之间的交互时应满足的相关属性。SA风格对基于SA的模型检验也有很大的影响。不同的体系结构风格，所验证的属性的侧重点也不同。

(4)ADL：已经有大量的ADL被用来模拟体系结构，有的针对特殊的领域，有的针对通用的体系结构。根据Medvidovice的定义[14]，ADL的开发有两个方面的趋向：一种是指ADL主要是用来辅助理解和交流软件系统的，即ADL必须是简单、易理解和图形化的语法，但是缺乏形式化的定义，难以用来分析和验证；另一种是指提供ADL的形式语义和语法，能够用来分析、模型检验、解析和运行时的工具支持。显然，如果采用完全具有形式化的语法和语义的ADL，能够很好地支持对SA进行模型检验，这其中代表性的ADL有Wright、Darwin等，如果非形式化或半形式化的图形或文字描述时，就需要将其描述转换为一个形式化的方式或直接转换为一种适合的形式化输入，如将UML状态机转换为Spin输入可接受的语言Promela。

结论 至此，本文调查分析了基于SA模型检验的几个问题。从现有的文献来看，与SA有关的模型检验问题的研究还处于初始阶段，很多问题有待探讨和解决。在未来的研究中，以下几个方面必将成为需要重点研究的主题：

(1)验证属性的拓展。当前的针对SA模型检验的方式大都是集中在验证SA安全性、活性、公平性和一致性等属性。而SA的其它方面属性，如系统的安全性(Security)、性能、可用性和容错等属性，也是下一步值得验证的问题[5]，实践显示可以进一步用来验证。

(2)解决模型检验中状态空间的方法。状态空间问题一直是制约模型检验技术在工业界广泛应用的一个关键问题。在软体体系结构的环境下，这个问题仍然存在。而解决的主要途径有两种：一是改变现有的方法，如On-the-Fly技术、抽象、对称、组合推理和模型检验、增量法等，使之与SA相适应；二是能够针对SA模型检验提出新的缩减状态空间的方法。

(3)切片技术的使用。在SA的模型检验中，特别是针对大规模的系统而言，运用软件体系结构的动态和静态切片技术来抽象出待验证系统模型的建立。一般的模型检验器所提供的反例信息并不能容易地定位和改正错误。因为反例的轨迹可能是很长且难懂的，经常包括和错误不相关的成百上千的无用代码。故在发现反例时，可以应用切片技术辅助发现和改正错误[18]。我们正在试图做这方面的探索。初步的研究表明这种做法是可行的，将会得到很多益处。

(4)当软件需求发生改变，或是软件遇到了新的环境，SA的演化总是不可避免。故考虑在演化环境下的SA模型检验技术仍然是未来研究的热点之一[20]。考虑支持动态演化的体系结构描述语言将是其中的研究重点之一。

参考文献

1 Shaw M, Garlan D. Software architecture: perspectives on an emerging discipline. Prentice Hall, 1996

2 Clarke E, Grumberg O, Peled D. Model checking. MIT Press, 2000

3 Inverardi P, Muccini H, Pelliccione P. Automated check of architectural models consistency using SPIN. In: Proc. of 16th IEEE International Conference on Automated Software Engineering, 2001. 346 ~ 349

4 Bose P. Automated translation of UML models of architectures for verification and simulation using SPIN. In: Proc. of 14th IEEE International Conference on Automated Software Engineering, 1999. 102 ~109

5 Deng Y, Wang J C, Tsai J J P, et al. An approach for modeling and analysis of security system architectures. IEEE Trans Knowledge and Data Engineering, 2003, 15(5):1099~1119

6 Caporuscio M, Inverardi P, Pelliccione P. Compositional verification of middleware-based software architecture descriptions. In: Proc. of 26th International Conference on Software Engineering, 2004. 221~ 230

7 Tsai J J P, Sistla A P, et al. Incremental verification of architecture specification language for real-time systems. In: Proc. of 3th International Workshop on Object-Oriented Real-Time Dependable Systems, 1997. 215~ 222

8 Magee J, Kramer J, Giannakopoulou D. Behavior analysis of software architectures. In: Proc. 1th Working IFIP Conference on Software Architecture, 1999

9 Barber K S, Graser T, Holt J. Providing early feedback in the development cycle through automated application of model checking to software architectures. In: Proc. of 16th IEEE International Conference on Automated Software Engineering, 2001. 341 ~345

10 Barber K S, Holt J. Software architecture correctness. IEEE Software, 2001, 18(6):64 ~ 65

11 Yu H Q, He X D, Deng Y, et al. A Formal Approach to Designing Secure Software Architectures. In: Proceedings of 8th IEEE International Symposium on High Assurance Systems Engineering, 2004. 289~ 290

12 Holzmann G J. The logic of bugs. In: FSE Foundations of Software Engineering, 2002

13 Bass L, Kazman R. Architecture-based development: [Tech Rep]. CMU/SEI-99-TR-007. 1999

14 Medvidovice N, Taylor R N. A classification and comparison framework for software architecture description language. IEEE Tran Software Engineering, 2000, 26(1):70~ 93

15 Holzmann G J. The model checker SPIN. IEEE Tran Software Engineering, 1997, 23(5):279~295

16 McMillan K L. Symbolic model checking. Boston: Kluwer Academic Publisher, 1993

17 Giannakopoulou D. Model checking for concurrent software architectures: [PhD dissertation]. London: Imperial College of London, 1999

18 Groce A D. Error explanation and fault localization with distance metrics: [PhD dissertation]. Pittsburgh: Carnegie Mellon University, 2005

19 Muccini H. Software architecture for testing, coordination and views model checking: [PhD dissertation]. Roma: Universita degli Studi di Roma La Sapienza Dottorato di Ricerca in Informatica, 2002

20 Mateescu R. Model checking for software architectures. In: Proc. of the 1st European Workshop on Software Architecture Springer Verlag, 2004. 219~224

21 Ciancarini P, Mascolo C. Model checking a software architecture. In: Proc. of ROSATEA: International Workshop on the Role of Software Architecture in Analysis and Testing. Software Engineering Notes, 1999, 24(4)

22 Jerad C, Barkaoui K. On the use of rewriting logic for verification of distributed software architecture description based LfP. In: Proc. of 16th IEEE International Workshop on Rapid System Prototyping, 2005. 202 ~208

23 Perry D E, Wolf A L. Foundation for the research of software architecture. SIGSOFT Software Enginerring Notes, 1992, 17(2): 40~52

24 Booch G J, Jacobson R V. The unified modeling language use guide. Addison Wesley, 1999

25 Bass L, Clements P, Kazman R. Software Architecture in Practice. Addison Wesley, 2003

26 CLavel M, et al. Maude manual (version 2.1), March 2004. URL=http://maude.cs.uiuc.edu/cn