基于线性时序逻辑的动态软件体系结构模型验证*

唐姗 彭鑫 赵文耘 刘奕明

Email: tangshan@fudan.edu.cn

(复旦大学计算机科学与工程系软件工程实验室，上海 200433)

摘要：如何有效地保证软件体系结构能够正确地进行动态演化是目前软件工程领域中一个亟待解决的问题。模型验证是一种关于系统性质验证的算法方法,它通常采用状态空间搜索的方法来检查一个给定的计算模型是否满足用某个时序逻辑公式描述的特定性质。模型验证的复杂性主要依赖于系统状态空间大小，所以模型验证方法所面临的最大问题是状态空间爆炸和内存不足。针对大规模动态软件体系结构模型验证方法的不足，本文提出了一种基于线性时序逻辑的模块化模型验证方法，从系统组成模块和整体系统两个不同层次探讨了验证动态软件体系结构的方法，给出了相应的实现算法，并结合实例说明了本方法的实用性。

关键词：动态软件体系结构 模型验证 线性时序逻辑 自动机

Model Checking of Dynamic Software Architecture Based on Linear Temporal Logic

Tang Shan, Peng Xin, Zhao Wen-yun, Liu Yi-ming

(Computer Science and Engineering Department, Software Engineering Lab of Fudan University, Shanghai 200433)

Abstract: One of the urgent problems in software engineering is how to guarantee the correctness of software architecture evolution. Model checking is an approach which is used to verify properties of systems. Generally, it checks whether a given model satisfies some special property which are expressed by linear temporal logic through checking all of states of the target model. Since the complexity of the model checking mainly depends on the amount of number of the states, the most intractable problems of the approach are lack of memory and the state space explosion. There are few effective model checking approaches for large scale dynamic software architecture, this paper proposes a modular model checking strategy based on linear temporal logic. From two different levels: the component composing the system and the whole system, this paper discusses how to verify dynamic software architecture in detail, gives the corresponding algorithms and introduces an e-business case to illustrate the availability of our approach.

Keyword: Dynamic Software Architecture, Model Checking, Linear Temporal Logic, Automaton

1. 引言

基于“构件+软件体系结构”的软件开发方法已成为目前软件工程领域的主流开发模式。近年来随着计算机软件系统的规模不断扩大，软件复杂度越来越高，软件系统的结构变得日益复杂，软件体系结构更是成为了决定软件系统质量的重要因素。特别是在Internet成为主流软件运行环境之后，网络的开放性和动态性使得用户需求与系统运行环境更加频繁地变化，这进一步增强了软件系统开发与维护的复杂性。如何有效地保证软件体系结构能够正确地进行动态演化，以指导运行时软件系统的动态变更，使其可以不断满足用户及环境的变更需求、具有持续可用的特点，是当前软件工程领域中一个亟待解决的问题。

关于如何对动态软件体系结构进行建模与验证，业界已做了大量的研究工作。文献[1]总结了近年来一些经典的描述动态软件体系结构的形式化方法。基于软件体

*基金资助说明：本课题得到国家自然科学基金(60473061)、国家 863 计划(2005AA113120)、973 项目(2006CB3003002)的资助。

作者简介：唐姗(1982-)，女，湖南益阳人，博士生，研究方向：软件体系结构、软件构件技术。彭鑫(1979-)，男，博士，讲师，研究方向：软件体系结构、领域工程、软件维护与再工程。赵文耘(1964-)，男，教授，博士生导师，CCF 高级会员，研究方向：软件复用、领域工程、电子商务。刘奕明(1978-)，男，博士生，研究方向：软件体系结构，软件构件技术。

系结构描述语言(Architecture Description Language,简称ADL)的方法[2]通过描述构件与连接件之间的连接、重连接等关系来刻画软件体系结构的动态演化。基于图文法的方法[3]通过动态改变图的结构来描述软件体系结构的动态变更。但是上面这些方法只关注体系结构的拓扑结构，不支持对体系结构行为的分析。Allen等人采用基于进程代数的方法[4]来对软件体系结构的行为进行建模，虽然他们提出的Dynamic Wright可以对软件体系结构进行死锁与一致性检测，但对体系结构动态演化的表达能力不足。Dowling等人提出了K-Component框架元模型[5]，该模型提供了一种专门的机制来显示地表达演化逻辑，将演化逻辑与计算逻辑相分离，实现了动态地修改演化逻辑，但是该模型缺乏严格的行为语义基础和结构语义基础，从而不能对模型推理和验证。C2[2]是一种基于分层结构、事件驱动的体系结构风格，它提供了一种体系结构变更语言AML，该语言支持在运行时添加、删除、替换构件以及更改构件之间的连接关系，但也是因为缺乏严格的形式化基础，不能对体系结构的动态行为进行推理和验证。文献[6]提出了一个验证动态自适应程序的“单片电路”(monolithic)方法，但它将整个程序作为一个整体来进行验证，面临状态空间爆炸的问题，且不支持增量式的开发。文献[7]提出一种叫做“变迁不变量格子”的方法，运用理论证明技术来证明系统在演化过程中及演化后都满足指定的变迁不变量，以此来断定系统一直处于正确的状态中，但是要从数学上计算出这样的变迁不变量格子的开销是非常大的，同时一旦系统的并发操作增多，验证所需的格子数目急速增多，极大地限制了系统的并发规模。[8]提出了一种比较高效地模块化模型验证方法，但是该方法只适应于各个被验证的子模块中状态个数不多的情况，如果状态个数太多就难以体现出该方法的优越性。本文在此基础上做了改进，本文一方面采用模块化的验证方法以支持增量式的软件开发与验证，另一方面对各个被验证的子模块运用动态构建状态空间的方法，有效地解决了状态空间爆炸的问题。

动态软件体系结构基于其本身高度的复杂性，通常很难对其进行描述和验证。所幸的是，形式化方法能够对大规模复杂软件系统的体系结构进行精确的、无二义性的描述，并能利用数学方法对软件系统的关键属性进行分析、验证和推理，从而有效地降低了动态软件体系结构难以描述和验证的难度。目前人们主要是从模型验证这一形式化方法入手来研究如何保证软件体系结构动态演化的正确性和可靠性。模型验证是一种关于系统性质验证的算法方法,它通常采用状态空间搜索的方法来检查一个给定的计算模型(程序)是否满足用某个时序逻辑公式描述的特定的性质[9]。具体来说，在模型检验中，系统的行为用Kripke结构(可看作是一种w自动机)来描述，系统的性质用时序逻辑公式(因为时序逻辑能够断言随时间变化的系统行为)描述，目前这一方法已在硬件设计和通信协议等领域得到了成功的应用。本文重点研究如何应用这些技术来对动态软件体系结构进行分析验证。

接下来本文着重阐述如何结合线性时序逻辑(LTL)和自动机两种形式化描述手段，同时融合多种模型验证技术来模块化地验证一个动态软件体系结构的行为演化模型是否满足用LTL公式描述的系统规范(亦即系统属性)。本文余下的内容结构组织如下：第2节简要阐述了结合运用LTL和Büchi自动机的原因，并给出了LTL和Büchi自动机的基础知识；第3节介绍了动态软件体系结构模型验证的方法；第4节对全文进行了总结和展望。

2．背景知识

线性时序逻辑(LTL)作为一种系统规范描述语言，有一些十分显著并不可替代的特征：直观、具有良好的兼容性。LTL (Linear Temporal Logic) 与自动机之间有着紧密的联系, 结合LTL语义和语法, 可以将LTL公式转换成Büchi自动机。由LTL生成的Büchi自动机所接受的语言能准确地表达了LTL公式所描述的系统属性。从而，我们可以把对LTL公式描述的系统属性的验证问题转换成检验Büchi自动机的包含问题。

2.1 线性时序逻辑（Linear Temporal Logic ,LTL）

线性时序逻辑是在命题逻辑的基础上加上时序操作而得来的。基于线性时序逻辑的分析方法是一种重要的描述和验证软件体系结构特性的形式化分析方法。它由Manna和Pnueli 首次开发出来用以描述系统的并发特性[10]。它以路径(状态序列)作为命题的论断对象，在状态序列上解释其真值。线性时序逻辑可以方便准确地描述并发系统的重要性质，如安全性(Safety)和活性(Liveness)。安全性用于说明“坏事情永远都不会发生”；活性用于说明“好事情最终会发生”。下面基于Manna-Pnueli 时序逻辑框架[10]给出线性时序逻辑的语法和语义。

2.1.1 LTL的语法

定义1 LTL的语法可以递归定义如下：

- 原子命题变元p, q, r, …是线性时序逻辑公式；
- 如果p，q是线性时序逻辑公式，则由布尔运算符∨(或)、∧(与)、¬(非)或时序操作符∪(until)、◇(finally)、□(always)、X(next)等连接而成的公式如：p∨q，p∧q，¬p，p∪q，p◇q，p□q，pXq也是线性时序逻辑公式；

2.1.2 LTL的语义

定义2 设F为线性时序逻辑公式集，LTL的语义模型可以用Kripke三元组结构M=(S,P,R)解释。其中$S = (S_0, S_1, \dots)$是非空状态集; $P = (P_0, P_1, P_2, \dots)$是非空路径集(状态序列集)，$P_i = S_i S_{i+1} \dots$称为一条路径；关系R: F×P→{true, False}是时序逻辑公式集F和非空路径集P的积到{true，false}上的映射，即有 ∀ f∈F, i≥0, P_i∈P, R(f, P_i)∈{true, False}。因此，根据Kripke三元组的语义模型，线性时序逻辑的语义可解释成一个无穷路径序列$P= S_0 S_1 S_2 \dots \dots$，其中每个$S_i$都是对命题变元的一个赋值。我们将否定只出现在原子命题前的线性时序逻辑公式称之为线性时序逻辑公式的NNF形式。若用M(R, P_i)|= f 表示R(f, P_i) = true，则各时序算子的语义可解释如下：

- M(R, P_i)|= f　　iff R(¬f, P_i) = false
- M(R, P_i)|= f∨q　iff M(R, P_i)|= f∨M(R, P_i)|= q
- M(R, P_i)|= f∧q　iff M(R, P_i)|=f∧M(R, P_i)|= q
- M(R, P_i)|=□f　　iff ∀ k≥i,M(R, P_k)|= f
- M(R, P_i)|=◇f　　iff ∃ k≥i, M(R, P_k)|= f
- M(R, P_i)|=Xf　　iff M(R, P_{i+1}) |= f
- M(R, P_i)|= f∪q　iff (∃ k≥i, M(R, P_k)|= q)∧(∀ j, i≤j<k, M(R, P_j)|= f)

2.2 Büchi自动机

Büchi自动机是可接受无限字(例如: $\omega = a_0a_1...a_i...$)的自动机，它由Büchi在1962年首次提出。当时用于构造S1S的判定过程。后来Vardi和Wopher把Büchi自动机用于系统的验证和模态逻辑的判定过程。Büchi自动机包含若干初始状态和若干接受状态，它可以生成一个给定的时序逻辑公式的所有无穷序列。

定义3 Büchi**自动机**：是一个五元组：BA= (S,Σ,△,I, F)，其中：S是有限状态集，Σ是有限字母集，△⊆S×Σ×S是变迁关系，I⊆S是初始状态集，F⊆S是可接受状态集，$F=\{F_1,F_2,\dots,F_n\}$，值得注意的是，状态集F可为空。BA的一个执行是一个无穷序列 $\delta = q_0 q_1 q_2 \dots$，其中：$q_0$∈I, 并且对任意的i≥0, 都有$q_i \rightarrow q_{i+1}$。若对每一个可接受状态集$F_i$∈F，至少有一个状态q∈$F_i$ 在序列δ中出现无穷次，则称序列δ为BA的一个可接受的执行δ。

3. 模型验证

目前业界对动态软件体系结构的研究主要集中在对软件体系结构拓扑结构演化的分析，很少有人分析体系结构的行为演化。针对这一方面的不足，本文通过深入研究，给出了一个有效的分析、验证动态软件体系结构行为的方法。软件体系结构的演化可以看作是由系统的源模型经过一组变迁演化到系统的目标模型[11]。本文采用模块化验证的思想，将动态软件体系结构看成是由若干个稳态程序(steady-state program)组成的整体，各个稳态程序之间可以按照一定的规则进行动态演化。只要符合演化规则，这些稳态程序在不同的演化情景下可充当源模型或者目标模型。基于这一基本思想，本文主要实现两个验证目标：(1)验证组成系统的各个稳态程序(模块)，因为它们是构成系统整体模型的基本要素，所以它们的正确性是确保整体系统模型的正确性的首要问题；(2)验证由所有稳态程序组装成的整体演化模型，这是保证最终整体模型正确性的关键。

鉴于动态软件体系结构的演化特性，很难直接应用传统的模型验证方法[12][13][14]来验证大规模动态软件体系结构。本文在传统模型验证方法的基础上，采取了如下两个方面的改进措施：第一，采用分而治之的办法，将大规模系统分解成若干个小的验证模块，即模块化模型验证，该方法一方面可以支持增量式的软件体系结构开发与验证，提高验证效率；另一方面可以简化问题的复杂度。具体来说，我们将系统结构看作由若干个稳态程序组成的整体。对于一个由n个稳态程序组成的动态软件体系结构，当我们开发和验证完了前n-1个稳态程序后，在开发第n个稳态程序时，我们只需要对该第n个稳态程序进行模型验证，而不需要从头至尾地再对系统的所有稳态程序进行验证了；第二，实际上,系统的许多性质的验证无须遍历模型的整个状态空间。针对每个模块(稳态程序)的验证，我们通过采用一组启发式策略来动态构造自动机，该方法不需事先构造出自动机的整个状态空间，而是在构造自动机的过程中按需动态地生成部分状态结点，若有错误出现，根据这些结点就能找到一个违反系统规范的反例来提前完成验证。该方法实现了对自动机进行瘦身的效果，从而有效地解决了内存不足和状态空

间爆炸等问题。

3.1 验证系统组成模块

这里是基于自动机来实现模型验证的。采用自动机做模型验证的主要优点是系统的行为抽象模型与要验证的系统属性都可以用自动机来表示。下面我们以对某个稳态程序的验证为例，详细介绍验证的实现过程， 验证算法如下：

1) 对被验证的稳态程序建模，得到用 Büchi 自动机表示的实现自动机；
2) 将需要验证的系统性质用 LTL 公式来描述；
3) 将 LTL 公式取反后转化为 Büchi 自动机，称为属性自动机；
4) 构造实现自动机与属性自动机的乘积自动机；
5) 验证该乘积自动机所接受的语言是否为空，如果为空，则说明系统实现满足系统属性。反之，则说明不满足，并可以由所接受的语言给出违反反例。反例详细描述了系统实现模型无法满足系统属性的原因，通过研究反例，可以精确地定位和纠正模型中的缺陷。

根据第 2 节背景知识中给出的 LTL 的语法和语义、Büchi 自动机的定义，不难直接构造出描述系统实现的 Büchi 自动机和描述系统属性的 LTL 公式，本节的重点在于阐述如何缩减自动机的状态空间，所以在此不对算法的 1、2 步展开讨论，接下来着重讨论算法中的第三个步骤。

3.1.1 构造属性自动机

将LTL公式Φ转化成为Büchi自动机的过程分为2步: (1)首先构造LTL公式的标记迁移图(Labeled Transition Graph) ; (2)将标记迁移图转换成Büchi自动机。

(1)构造LTL公式的标记迁移图

本文基于表(tableau)方法[15]并以深度优先顺序来构造标记迁移图。设标记迁移图中的结点的数据结构为一个5元组：node = <ID, Incoming, New, Old, Next>，其中，ID是当前结点的惟一标识符；Incoming是一个结点列表，该列表当中的每个结点都有一条指向当前结点的输出边；New表示一组在当前结点处必须满足，但还未被处理的时序属性；Old表示一组在当前结点处必须满足且已被处理的时序属性；Next表示那些满足Old中属性的状态结点的所有直接后继状态节点必须满足的时序属性；结点n的Incoming、New、Old、Next字段分别用Incoming(n)、New(n)、Old(n)、Next(n)表示。具体构造算法的伪代码描述见图1。(注：$\Box\mu$等价于$\mu \bigcup$ False; $\Diamond\mu$等价于True$\bigcup\mu$，从而这两种形式的公式，可以视为是$\mu\bigcup\varphi$的特殊情形。)

算法说明：算法中的递归函数expand(Node, Node_Set)的功能是构造组成图的结点集，其中参数Node为当前处理结点，Node_Set为由已创建的结点组成的结点集。算法最终返回的结点集中的结点的New字段均为null。

- 第42-45行表示：从要验证的公式集Φ开始，先产生一个初始结点，该结点仅有一条输入边init，New字段为{ Φ },Old字段为空，Next字段为空，Node_Set为空。
- 第3-8行表示：递归调用expand函数，先判断当前处理结点的New字段是否为空，
  - 如果是，则继续判断该结点是否能被添加到Node_Set中：
    - 如果Node_Set中已存在与该结点在Old字段和Next字段相同的结点n，那么将当前结点的输入边加入到n的Incoming列表中，并抛弃该结点，然后返回结点集；
    - 否则，根据当前结点产生一个后继结点q(q的Incoming为当前结点的ID，New字段为当前结点的Next字段，Old和Next字段都为空)，然后将当前结点加入到Node_Set中并将q作为新的当前结点。
  - 第9-41行表示：如果当前处理结点的New字段不为空，那么就从它的New字段中挑取一个公式γ，并将γ从New字段中删除，然后根据不同的情况进行相应处理。若发现错误公式或冲突公式，则也作抛弃处理。

(2)将标记迁移图转换成Büchi自动机

由上述算法所产生的结点集生成属性自动机BA= (S, Σ,$\triangle$,I, F)。其中状态集S就是上述算法生成的结点集Node_Set；Σ由2^{AP}组成(AP为原子命题集合)；变迁关系$\triangle$由p→q来描述，p与q满足p∈In**comi**ng(q)；初始状态集I由这么一些结点组成，它们的Incoming列表中包含init边。

3.1.2 构造并验证乘积自动机

设属性自动机$BA_P=(S_1,\Sigma,\triangle_1,I_1,F_1)$，实现自动机$BA_I=(S_2,\Sigma,\triangle_2,I_2,F_2)$，则它们的乘积自动机$BA_M=BA_P\times BA_I=(S,\Sigma,\triangle,I,F)$，其中：$S=S_1\times S_2$； $I=I_1\times I_2$； $F=F_1$

$\times F_2$；　△: $S_1\times S_2\times\Sigma\rightarrow 2^{S1\times S2}$定义如下：(s2,t2)∈△((s1,t1),a) iff s2∈$\triangle_1$(s1,a1)∧ t2∈$\triangle_2$(t1,a2)∧(a1∧a2)；

若把乘积自动机看成是一个有向图，则验证乘积自动机是否为空的问题可以通过检查是否存在至少一个从初态可达的环路。所以本文将验证问题转化为判断是否有从乘积自动机的初态集出发到达接受状态的环路径，找到环路则表明系统行为模型不满足系统属性，并返回一个反例，结束遍历。反之则表明系统行为模型满足系统属性。

```
1 record node= [ID:string, Incoming:set of string, New:set of formula,
2        Old: set of formula, Next: set of formula];
3 function expand (Node, Node_Set)
4 If New(Node)=null then
5   If ND∈Node_Set with Old(ND)= Old(Node) and Next(ND)=Next(Node)
6   Then Incoming(ND)= Incoming(ND)∪Incoming(Node);
7     Return (Node_Set);
8   Else return (expand([ID←new_id(),Incoming←{ID(Node)},New←Next(Node),Old←null,
          Next←null],{Node}∪Node_Set)); //new_id()函数为每一个后续调用产生一个新的字符串
9 Else
10  Let γ ∈New;
11  New(Node) :=New(Node)\{ γ };
12  Case γ of:
13   γ =Pn or ¬Pn or γ =True or γ =False =>
14    If =False or Neg(γ )∈Old(Node) // Neg()函数的功能是产生公式的否定式
15      Then return(Node_Set);
16      Else Old(Node):= Old(Node)∪{γ };
17      Return(expand(Node, Node_Set));
18   γ = μ ∪ φ ⇒
19   Node1:=[ID←new_id(),Incoming←Incoming(Node),
20        New←New(Node)∪({ μ }\Old(Node)),Old←Old(Node)∪{γ },
21        Next←Next(Node)∪{ γ }];
22   Node2:=[ID←new_id(), Incoming←Incoming(Node),
23        New←New(Node)∪({ φ }\Old(Node)),Old←Old(Node)∪{ γ },
24        Next←Next(Node)];
25   Return(expand(Node2, expand(Node1,Node_Set)));
26   γ = μ ∨ φ ⇒
27   Node1:=[ID←new_id(),Incoming←Incoming(Node),
28        New←New(Node)∪({ μ }\Old(Node)),Old←Old(Node)∪{ γ },
29        Next←Next(Node)];
30   Node2:=[ID←new_id(), Incoming←Incoming(Node),
31        New←New(Node)∪({ φ }\Old(Node)),Old←Old(Node)∪{ γ },
32        Next←Next(Node)];
33   Return(expand(Node2, expand(Node1,Node_Set)));
34   γ =  μ ∧ φ ⇒
35   Return(expand([ID←ID(Node),Incoming←Incoming(Node),
36          New←New(Node)∪({ μ, φ }\Old(Node)),
37          Old←Old(Node)∪{ γ }, Next= Next(Node)],Node_Set ));
38  γ = Xμ ⇒
    Return(expand([ID←ID(Node),Incoming←Incoming(Node),
39          New←New(Node), Old←Old(Node)∪{ γ },
40          Next= Next(Node)∪{ μ }],Node_Set ));
41 End expand;

42 Function create_graph(Φ )
43 Return(expand([ID←new_id(), Incoming ←{init}, New←{Φ}, Old←null,
44     Next←null], null));
45 End create_graph;
```

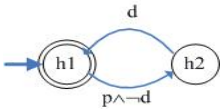
图1 属性自动机构造算法的伪代码描述

3.1.3 实例介绍

以网上购物系统为例，为了简单起见，我们这里考虑系统行为仅由支付处理和发货处理组成，行为图用Büchi自动机表示如下图2(图中同心圆代表自动机的接受状态，下同)所示。我们规定系统必须满足这样一个属性 K :系统成功处理完客户的支付手续(pay，简写为p)后，必会进行发货(deliver，简写为d)处理。用LTL公式描述如下：

$$K=\Box(p\Rightarrow\Diamond d)$$

将属性公式取反，得到 $\neg K=\Diamond(p\wedge\neg d)$。根据 3.1.1 节中的算法，我们可以最终得到两个结点：N1=[ID:N1, Incoming: init, New: null, Old: null, Next: null]、N2=[ID:N2, Incoming: N1, New: null, Old:{ ¬d , p∧¬d }, Next: null]。将由两结点所构成的标记迁移图转换成 Büchi 自动机，如下图 3 所示。



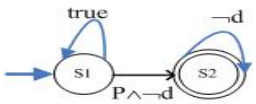


图2 系统行为自动机A1　　**图3 系统属性自动机A2**

由图4可以看出乘积自动机A1×A2没有从初态(h1,s1)出发到达接受状态(h1,s2)的环路，则可得知乘积自动机接受的语言为空。所以，我们可以确认系统行为模型满足系统的属性 K 。

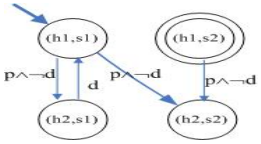


图 4 乘积自动机 A1×A2

3.2 验证系统整体演化模型

本文采用假设/保证（Assume/Guarantee，简称A/G）的推理方法[16][17]来验证系统整体演化模型。A/G推理方法最初用于渐进式地开发并发程序，而现在越来越多地用于分解对大规模复杂软件系统的验证。对于一个由多个进程组成的并发系统，基于模块化以及组合模型验证的思想，从子系统的性质推导出整体系统模型的性质：我们将系统的整体行为性质分解为组成系统的各个构件的行为性质。如果通过验证每个构件是否满足其局部属性，然后再加以综合就可以推断出整个系统是否满足其全局属性的话，复杂问题的处理就变得容易多了。但是由于系统中各个构件之间存在着各种依赖关系，亦即每个构件的状态会受到与其有关系的周围构件(下文简称为环境)的影响，为了不失正确性，我们在对构件进行验证时，还必需将环境因素考虑进去，也就是说我们必须基于构件的环境假设条件来对构件进行验证。在A/G范式中，我们可以把问题描述抽象为如下的表达式：

$$\langle A\rangle\ M\ \langle G\rangle$$

其中，***M***为一个构件，***A***为对构件***M***所处的环境设置的假设条件，***G***为一个属性表达式。如果当构件***M***的环境满足A，在此环境下，构件M满足属性***G***，那么我们称表达式<***A***> ***M*** <***G***>为真。在系统验证中，为了简单起见，我们先假设一个系统由两个构件***M1***、***M2***组成。要验证系

统是否满足属性***G***，我们可以这样来应用A/G推理方法：如果有<***A***> ***M1*** <***G***> 和 <***True***> ***M2*** <***A***>都为真，那么<***True***> ***M1***|| ***M2*** <***G***>也为真，这种证明策略也可以用如下的推理规则来表达：

(Premise 1) <***A***> ***M1*** <***G***>

(Premise 2) <***True***> ***M2*** <***A***>

<***True***> ***M1*** || ***M2*** <***G***>

将上述推理规则映射到动态软件体系结构的模型验证中，我们可以解释为：设M1是系统演化后的目标模型，M2是系统演化前的源模型，如果M2在任何条件下都满足A，M1在环境满足A的条件下满足属性G(这里，M1的环境指M2)，那么我们可以推断出由源模型M2和目标模型M1所构成的整个系统模型在任何情况下满足属性G。由此，我们发现，要验证动态软件体系结构是否满足它的变迁不变量属性G，只需验证在目标模型的环境满足条件A的前提下，它的目标模型是否满足G。从而实现了将对大规模系统的验证任务分解成对组成它的子系统的验证，这样缩小了问题规模，降低了处理难度。

如果我们将A视为构件M1的前置条件，将G视为构件M1的后置条件的话，从软件工程的角度来看，只有每个构件尽可能少的向环境提出要求，并尽可能多的向外提供服务，才能使构件发挥出其自身最大的功效。本文所研究的体系结构的动态演化，是指对系统的在线升级，即目标模型要比源模型的功能更多。综上所述，我们不难得出，一个设计良好的系统，必定满足G蕴含A这个条件。最后我们把问题归结到判断M1的G是否蕴含A，如果是，则表明由源模型M2和目标模型M1所组成的演化系统M1||M2满足属性G，亦即我们能够从源模型正确演化到目标模型；反之，则表明不能进行正确的演化。

下面我们以两个稳态程序为例，描述如何验证从源模型(S)演化到目标模型(T)。验证算法如下：

1) 从初始状态开始，分别计算出各个稳态程序中每个状态在不发生演化的情况下必须满足的条件(guarantee)。每个状态上的guarantee是确保整个稳态程序满足其局部属性的必要条件。开始时，初态的guarantee是该稳态程序的局部属性，然后运用[8]中的标记算法依次对当前状态的后继状态计算并标记出相应的guarantee；
2) 计算并标记出各个稳态程序中每个状态的上对环境的假设条件(assumption)。每个assumption是为了保证当前状态的正确性而对当前处理状态所处环境做出的假设,即指该环境必须满足用assumption描述的性质。具体来说依次对S中各个带有变迁边的状态进行处理：将其上的guarantee通过变迁边传递给T中的目标状态，并标记为相应目标状态的assumption。假设S中状态i的guarantee为 ϕ，则我们沿着从状态i出发的变迁边将 $\phi \rightarrow$LPt传递给T中相应的目标状态(LPt为T的局部属性)，作为目标状态的assumption；
3) 验证稳态程序中的状态在满足assumption的情况下是否满足guarantee。即通过比较分析各个状态上的guarantee和assumption，判断guarantee是否蕴含assumption。如果是，则表明系统可以在该状态上从源模型正确演化到目标模型；反之，则表明在该处的演化违反了系统属性，并找到一个反例。

3.2.1 实例介绍

我们举个简单的例子，以网上购物为实例来描述我们的验证方法。假定有两种购物处理模式：普通用户模式和混合用户模式。在普通用户处理模式中(如下图5中稳态程序SP1所示)，处理流程依次为：处理用户注册/登录(login)、填写订单(fill)、支付(pay)、确认订单(confirm)、发货(deliver)。后来商家为了刺激顾客消费，引入了会员制，会员可以享受打折优惠。该模式(如下图5中稳态程序SP2所示)下的处理流程和SP1中的流程很相似，只是在填写完订单后，系统需要对用户身份进行验证(verify)，如果为vip用户则先进入打折状态(discount)对订单上的商品进行打折，再进入支付状态，如果为一般用户，则直接进入支付状态。

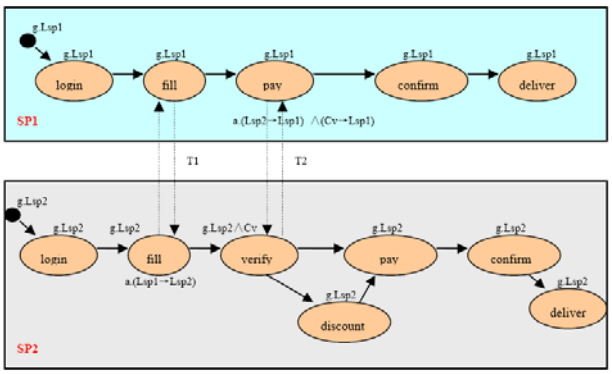


图5 网上购物实例

验证过程：

1) 计算每个稳态程序中各个状态上的guarantee。我们先计算SP1、SP2的局部属性，然后将计算出的局部属性分别赋给它们各自的初始状态(初态

在图中用实心黑圆点表示)。我们设SP1和SP2的局部属性是一样的，即一旦确认订单，就一定会发货。用LTL公式描述如下：

$$Lsp1 = Lsp2 = \Box(confirmed \Rightarrow \Diamond delivered)$$

在混合模式SP2中，进行身份验证后，必须对vip客户进行打折处理才能进入支付状态，所以在验证状态verify上加上条件Cv，用LTL公式描述如下：

$$Cv = \Box(vip \Rightarrow (\neg paid \cup discounted))$$

接下来我们采用[8]中的标记算法，将各个状态上的guarantee标上，并以g.作为前缀。如上图5所示。

2) 假设有两个变迁活动，一个是T1，从SP1中的fill状态演化到SP2中的fill状态，按照算法中的思想，为SP2中的fill生成assumption：Lsp1→Lsp2,并冠以a.作为前缀。另一个变迁是T2，从SP2中的verfiy状态演化到SP1中的pay状态，同样地，为目标状态生成相应的assumption：(Lsp2→Lsp1)∧(Cv→Lsp1)，并以a.作为前缀。

3) 验证目标状态的guarantee是否蕴含相应的assumption。验证结果为：g.Lsp2蕴含a.(Lsp1→Lsp2)，则可推断T1为合法变迁，而g.Lsp1不蕴含a.(Lsp2→Lsp1)∧(Cv→Lsp1)，则可推断T2为非法变迁。

4．总结与进一步的工作

模型验证的优势在于能自动穷尽地搜索状态空间,并且可以为用户精确定位系统的错误行为。但是这种方法的最大局限性是系统所面临的状态空间爆炸问题。为此，业界提出了各种解决方案，如组合方法(Compositional Methods),符号化方法(Symbolic Methods),偏序缩减方法(Partial Order Reduction Methods)和抽象方法等。本文通过研究比较，综合各种合适的技术，将它们引入到动态软件体系结构的模型验证工作中。从系统组成模块、系统整体模型两个不同层次探讨了如何进行模型验证的方法。与现有方法相比，本文的优点如下：一方面在对系统组成模块进行验证时，通过动态地构造自动机，有效地解决了状态空间爆炸的问题；另一方面，通过采用模块化的方法来开发验证系统，支持增量式的开发验证模式，大大提高了工作效率。但是由于某些需要验证的属性很难用线性时序逻辑公式进行描述，有待我们进一步发现或研究出更好的描述方法。其次，考虑到系统的日益复杂性，我们还应该对缓解状态空间爆炸的方法作进一步的研究。

参考文献

[1] J. Bradbury, J. Cordy, J. Dingel, and M. Wermelinger. A survey of self management in dynamic software architecture specifications. in Proc. of the ACM SIGSOFT International Workshop on Self-Managed Systems (WOSS'04), 2004.

[2] P. Oreizy, N. Medvidovic, and R. N. Taylor. Architecture-based runtime software evolution. In Proceedings of the 20th international conference on Software engineering(ICSE'98). IEEE Computer Society, 1998.

[3] G. Taentzer, M. Goedicke, and T. Meyer. Dynamic change management by distributed graph transformation: Towards configurable distributed systems. The 6th International Workshop on Theory and Application of Graph Transformations, pp. 179 – 193, Springer-Verlag, 2000.

[4] R. Allen, R. Douence, and D. Garlan. Specifying and analyzing dynamic software architectures. In Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE'98), 1998.

[5] Dowling J, Cahill V, Clarke S. Dynamic software evolution and the k-component model. In: Northrop L, Vlissides J, eds. Workshop on Software Evolution, Conf. on Object-Oriented Programming Systems, Languages, and Applications 2001. New York: ACM Press, 2001.

[6] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: an investigation of decomposition for assume-guarantee reasoning. In ISSTA'06: Proceedings of the 2006 International Symposium on Software Testing and Analysis,pp. 97.108, ACM Press, 2006.

[7] S. Kulkarni and K. Biyani. Correctness of component-based adaptation. In Proceedings of International Symposium on Component-based Software Engineering, 2004.

[8] J.Zhang, B.H.C Cheng. Modular model checking of

dynamically adaptive programs. Technical Report MSU-CSE-06-18, computer science and engineering, Michigan State University, East Lansing, Michigan,2006

[9] 李广元,唐稚松. 基于线性时序逻辑的实时系统模型检查.软件学报,Vol.13,No.2. 2002

[10] Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, 1992

[11] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In : Proceedings of International Conference on Software Engineering (ICSE'06), 2006.

[12] E. M. Clarke, O. Grumberg, D. Peled. Model Checking. Cambridge [M], MA: MIT Press, 2001, 35-49.

[13] Vardi M Y, Wolper P. An automata-theoretic approach to automatic program verification. In proc of 1st IEEE Symp on Logic in Computer Science. Cambridge,1986.322~331.

[14] Campos SV. A quantitative approach to the formal verification of real-time system[PhD dissertation]. Carnegie Mellon University,Pittsburgh,1996

[15] P. Wolper, The tableau method for temporal logic: an overview, Logique et Analyse,110–111(1985), 119–136.

[16] S. Chaki, E. Clarke, D. Giannakopoulou, and C. Pasareanu. Abstraction and assume-guarantee reasoning for automated software verification. RIACS TR 05.02,October 2004.

[17] Dimitra Giannakopoulou, Corina S. P˘as˘areanu, and Jamieson M. Cobleigh. Assumeguarantee verification of source code with design-level assumptions. In International Conference on Software Engineering, pages 211–220, May 2004.

姓名(中文)	唐　　姗	姓名(英文)	Tang Shan	性别	女	出生年月	1982.03
职称(职务)	无	电话	02155074192	Email	tangshan@fudan.edu.cn		
手机	13661862018	传真		通信地址	复旦大学北区学生公寓 117 号楼 602 室		
研究方向	中文	软件体系结构、软件构件技术					
	英文	Software Architecture、Software Component Technology					

基金资助	中文	国家自然科学基金: 动态软件体系结构建模技术研究(60473061)、国家 863 计划: 特征驱动的领域分析、体系结构建模技术及支持工具研究(2005AA113120)、国家 973 项目: 无线传感网络的自主组网模型与方法研究（2006CB3003002）
	英文	