软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(10):2564−2582 [doi: 10.13328/j.cnki.jos.005145]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

CPN 建模与 on-the-fly 方法相结合的测试用例生成*

张玉荣[1], 李 华[1,2], 邢 熠[1], 王显荣[1], 阮宏玮[1,2], 张素梅[1]

[1](内蒙古大学 计算机学院,内蒙古 呼和浩特 010021)

[2](内蒙古大学 网络信息中心,内蒙古 呼和浩特 010021)

通讯作者: 李华, E-mail: cslihua@imu.edu.cn

摘 要: 在对复杂的软件系统进行测试时,生成的系统状态空间可能会非常庞大.为了避免对整个状态空间进行遍历,提出将 on-the-fly 方法与 CPN 形式化建模方法结合起来,用于生成测试例.在这种方法中,无需对整个状态空间进行遍历,只是仅对测试人员感兴趣的部分状态空间进行针对性的测试.首先,给出 CPN 和扩展可达图的定义,介绍了 on-the-fly 测试方法中涉及的相关概念,包括系统规约、测试目的、同步乘积和测试例等.然后,实现了同步乘积算法,并设计相关测试例对其进行了测试.最后,选定一个被测系统示例 CPN 建模与 on-the-fly 结合的方法,并通过适配器实现与被测系统的交互,生成和执行测试例,由此验证了方法的可行性和有效性.

关键词: on-the-fly 测试;CPN 层次模型;扩展可达图;同步乘积;测试例

中图法分类号: TP311

中文引用格式: 张玉荣,李华,邢熠,王显荣,阮宏玮,张素梅.CPN 建模与 on-the-fly 方法相结合的测试用例生成.软件学报,2017,28(10):2564−2582. http://www.jos.org.cn/1000-9825/5145.htm

英文引用格式: Zhang YR, Li H, Xing Y, Wang XR, Ruan HW, Zhang SM. Test case generation based on the combination of CPN modeling and on-the-fly method. Ruan Jian Xue Bao/Journal of Software, 2017,28(10):2564−2582 (in Chinese). http://www.jos.org.cn/1000-9825/5145.htm

Test Case Generation Based on the Combination of CPN Modeling and On-the-Fly Method

ZHANG Yu-Rong[1], LI Hua[1,2], XING Yi[1], WANG Xian-Rong[1], RUAN Hong-Wei[1,2], ZHANG Su-Mei[1]

[1](College of Computer Science, Inner Mongolia University, Hohhot 010021, China)

[2](Centre of Network and Information, Inner Mongolia University, Hohhot 010021, China)

Abstract: The generated system state space can be very large when a complex software system is tested. In order to avoid the unnecessary traversing of the entire state space, a new method is presented based on the combination of CPN modeling and on-the-fly method to generate test cases. During such a process, only part of the state space is traversed according to the tester's personnel interest. Firstly, both the definitions of CPN and the extended reachability graph are introduced, and the related concepts relating to the on-the-fly testing method, including system specification, test purpose, synchronous product and test cases, are introduced. Secondly, a synchronous product algorithm is implemented, and the test cases are designed to test the algorithm as well. Finally, an implementation under test is selected to sample the combination method of CPN modeling and on-the-fly method. The interactions between the tester and the implementation under test are realized through an adapter, and the test cases are generated and executed simultaneously. Thus the feasibility and the effectiveness of the proposed method are verified.

Key words: on-the-fly testing; CPN hierarchy modeling; extended reachability graph; synchronous product; test case

* 基金项目: 国家自然科学基金(61163011, 61262082); 内蒙古自治区自然科学基金(2015MS0612); 内蒙古自治区高校科学技术研究项目(NJZY010); 赛尔创新项目(NGII20150112)

Foundation item: National Natural Science Foundation of China (61163011, 61262082); Natural Science Foundation of Inner Mongolia Autonomous Region of China (2015MS0612); Scientific Research Foundation of the Higher Education Institutions of Inner Mongolia Autonomous Region of China (NJZY010); Cernet Innovation Project (NGII20150112).

收稿时间: 2015-10-25; 修改时间: 2016-03-03, 2016-05-31, 2016-09-07; 采用时间: 2016-09-29

随着 Internet 的迅速发展,互联网研究领域中出现了各种各样的软件系统,且系统的规模、复杂性、分布性和并发性等不断增长.如何保障系统的正确性、安全性和可靠性成为目前需要高度关注和亟需解决的问题,特别是一些安全关键系统,若在其运行期间出现错误,可能会造成巨大的经济或人力损失,甚至产生灾难性的后果.因此,需要通过测试来保证系统的正确性.其中,基于形式化方法进行测试是当前比较流行的一种确认技术.形式化方法是指建立在严格数学基础上的软件系统建模方法,其目的是刻画系统的行为,以便在基于形式化模型对复杂系统进行测试生成时,不至于产生太多冗余的测试例,以提高测试效率.

在对软件系统进行测试时,尤其是复杂的软件系统,存在的大量测试数据或诸多并发行为等因素都可能会使软件系统形式化模型的状态空间非常庞大,从而导致测试执行效率较低甚至产生状态空间爆炸等问题.并且,当基于互联网运行软件时,环境的不可预测也可能会导致系统执行的动态性,进而导致对于系统的测试不能像有些系统可以预先推导出测试并逐步执行,而是需要依据现实环境中当前的执行结果进行测试的执行.因此,如何应对被测动态的输出以进行高效的测试执行就成为问题.

一般软件测试过程至少包括测试生成、测试执行和测试判定 3 个阶段,其中,测试生成和测试执行是独立进行的,这使得测试耗时长、效率低、代价大以及测试例出现冗余等.基于形式化方法进行测试生成可以使测试生成借助一些成熟的数学方法进行推导,能够保证一定的测试覆盖;缺点是建模需要时间,并且需要有将推出的测试序列向具体的测试例转换的时间.有限状态机、进程代数及 Petri 网等是常用的形式化建模方法,着色 Petri 网(coloured Petri net,简称 CPN)是一种高级 Petri 网,它能够更加直观、准确地刻画并发、同步或异步等动态的系统行为,非常适合对大型复杂的软件系统进行建模和仿真.CPN Tools[1]是一种成熟的 Petri 网建模和仿真工具,它的出现为基于 CPN 的研究提供了可靠的、易使用的仿真平台.而且在采用 CPN 形式化建模方法建模时,CPN Tools 工具会自动进行语法和类型的检查,用户可根据其提供的错误信息快速建立正确的模型.on-the-fly 方法是一种局部模型检测,是根据待验证模型性质的需要,有一定针对性地以需求驱动的方式构建状态空间,在找到反例之前只产生部分或必要的系统状态,避免对整个状态空间进行穷举搜索[2].该方法不仅能够解决内存和时间不足的问题,而且还可以减少或避免出现状态空间爆炸问题.根据 on-the-fly 模型检测的思想,将 on-the-fly 方法用于软件测试中,即通过测试目的从系统规约状态空间中构建待搜索的状态空间,并对生成的状态空间进行遍历得到测试例,进而完成对待测软件系统的测试.在实际应用中,尤其是复杂的并发系统,可以采用层次化 CPN 建模方法.所谓层次化 CPN 建模是指用一些子网来代替需要进一步细化的变迁,这样做一方面能够使模型的结构更加清晰,另一方面也便于对模型进行形式化分析.此外,on-the-fly 测试方法将测试生成和测试执行同时进行,不仅可以实时应对并发过程中的不确定性,而且可以减少测试例冗余,提高测试效率.因此,本文提出将 on-the-fly 测试方法与 CPN 形式化建模方法相结合,生成并执行测试例.

本文主要贡献在于:在测试用例生成过程中,采用 CPN 建模与 on-the-fly 测试相结合的方法.利用 CPN 中的层次概念,对被测系统实现从整体到局部、由粗到精的逐步细化,使建模更加清晰;利用 CPN Tools 工具建立和执行模型,并对模型进行安全性、活性等属性的验证;on-the-fly 测试方法的引入避免了对整个状态空间的搜索,在很大程度上可以减少时间和内存的使用,同时提高测试例生成和执行的效率.

本文第 1 节介绍一些基本概念.第 2 节概述 CPN 建模方法与 on-the-fly 测试方法的相关研究工作.第 3 节给出基于 CPN 的 on-the-fly 测试方法.第 4 节以学籍管理系统为例,说明本文方法的可用性和有效性.第 5 节进行总结并展望未来工作.

1 前导知识

Petri 网(Petri net)是一种对离散并行系统的数学表示方法[3,4],是描述具有分布、并发、异步特征系统的有效工具,但其在描述复杂系统时容易面临状态空间爆炸问题,因而只适用于简单系统的建模.CPN 是基于 Petri 网提出的一种高级 Petri 网,它保留了基本 Petri 网的分析验证方法,同时对其语义进行了扩展,增加了颜色集.颜色集的引入可在一定程度上缓解状态空间爆炸等问题[4].下面是着色 Petri 网的定义[5,6].

定义 1. 着色 Petri 网是一个九元组,$CPN=(\Sigma,P,T,A,N,C,G,E,I)$,满足以下条件.

(1) Σ 是一个非空有限类型集,也称为颜色集,它决定用于 CPN 中的 token 的类型、运算和函数,例如初始化表达式、弧表达式、防卫函数等;

(2) P 是库所的有限集合;

(3) T 是变迁的有限集合;

(4) A 是有向弧的集合,满足 $A\subseteq P\times T\cup T\times P$,且 $P\cap T=P\cap A=T\cap A=\varnothing$;

(5) N 是节点函数,定义从 A 到 $P\times T\cup T\times P$ 的映射关系,且弧连接的必须是不同类型的节点,即(库所,变迁)或(变迁,库所);

(6) C 是颜色函数,定义从 P 到 Σ 的映射关系,即 C 把每个库所 p 都映射到一个颜色集 $C(p)$.换言之,库所中的每个标识都必须属于某个颜色集;

(7) G 是防卫函数,定义从 T 到表达式的映射关系,且满足:$\forall t\in T:[Type(G(t))=B\wedge Type(Var(G(t)))\in\Sigma]$,其中,$B$ 表示布尔值,$G(t)$中所有变量的类型必须包含于 Σ 中.$Type(expr)$表示表达式 $expr$ 的类型,$Type(v)$表示变量 v 的类型,$Var(expr)$表示表达式 $expr$ 中变量的集合.当且仅当变迁的防卫函数为真时变迁才可能被触发.若防卫函数为空,则默认为真;

(8) E 是弧表达式函数,定义从 A 到表达式的映射关系,且满足:$\forall a\in A:[Type(E(a))=C(p(a))_{MS}\wedge Type(Var(E(a)))\subseteq\Sigma]$,其中,$p(a)$是 $N(a)$的库所,表达式的类型是 $C(p(a))_{\mathrm{MS}}$.$C(p(a))_{\mathrm{MS}}$ 表示颜色集 $C(p(a))$上的多重集[7],$E(a)$的每个求值都生成一个与相邻库所颜色集类型相同的多重集.此外,弧表达式也可以不出现,默认为空;

(9) I 是初始化函数,定义从 P 到闭表达式的映射关系,且满足:$\forall p\in P:[Type(I(p))=C(p)_{\mathrm{MS}}]$,即 I 把每一个库所 $p\in P$ 都映射到一个闭表达式,且闭表达式的类型为 $C(p)_{\mathrm{MS}}$.所谓闭表达式是不含任何变量的表达式.

为说明 CPN 的形式化定义,以一个简单协议传输为例[8].图 1 为简单协议的 CPN 模型,其多元组表示形式如图 2 所示.

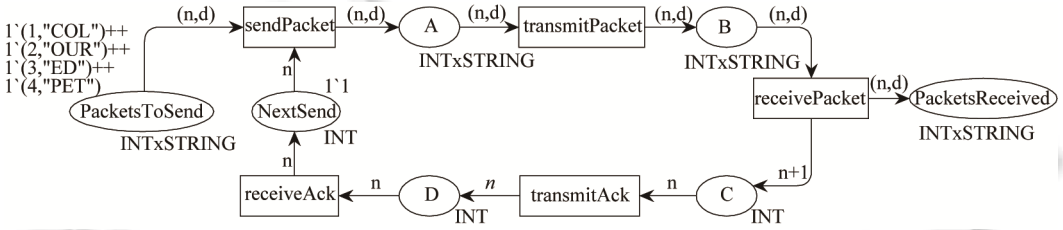

Fig.1 An example of CPN model of simple protocol

图 1 简单协议的 CPN 模型示例

(1) Σ = { INT, STRING, INTxSTRING }.

(2) P = { A, B, C, D, PacketsToSend, PacketsReceived, NextSend }.

(3) T = { sendPacket, transmitPacket, receivePacket, transmitAck, receiveAck }.

(4) A = { PacketsToSend to sendPacket, sendPacket to A,A to transmitPacket, transmitPacket to B, B to receivePacket, receivePacket to PacketsReceived, receivePacket to C,C to transmitAck, transmitAck to D, D to receiveAck, receiveAck to NextSend, NextSend to sendPacket }.

(5) N(a) = (SOURCE,DEST) 若a的形式为SOURCE to DEST

(6) $C(p)=\begin{cases}\text{INT} & \text{若p}\in\{\text{C ,D, NextSend}\}\\ \text{INTxSTRING} & \text{其他}\end{cases}$

(7) G(t) = true.

(8) $E(a)=\begin{cases}\text{n} & \text{若a}\in\{\text{C to transmitAck, transmitAck to D, D to receiveAck, receiveAck to NextSend, NextSend to sendPacket}\}\\ \text{n+1} & \text{若a= receivePacket to C}\\ \text{(n,d)} & \text{其他}\end{cases}$

(9) $I(p)=\begin{cases}\text{1`(1,"COL" )++1`(2,"OUR")++1`(3,"ED" )++1`(4,"PET")} & \text{若p=PacketsToSend}\\ \varnothing & \text{其他}\end{cases}$

Fig.2 The many-tuple representation of CPN model in Fig.1

图 2 图 1 中 CPN 模型的多元组表示

在系统模型构建完成后,通过执行模型并生成系统模型的状态空间,即可达图(reachability graph).CPN 中的可达图与基本 Petri 网中的可达树类似,其核心思想是构造一个包含系统模型执行过程中的所有状态和触发序列的图.在可达图中,节点表示 CPN 模型中描述系统的一个可能的状态,弧表示 CPN 模型中可能发生的变迁,该变迁能够触发被测系统状态的改变.图 3 为图 1 中 CPN 模型仿真运行后得到的可达图.通过可达图可以对 CPN 的有界性、可达性和活性等属性进行分析.

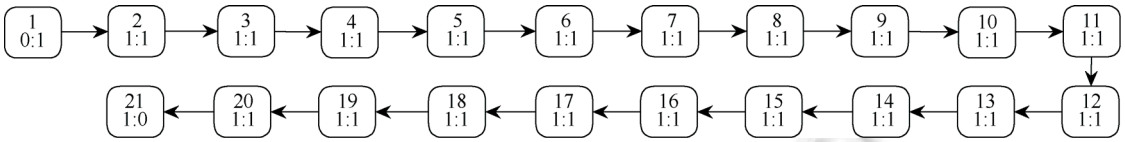

Fig.3 The reachability graph generated from Fig.1

图 3 图 1 的可达图

本文在基本 Petri 网可达图的基础上,依据需求对可达图概念进行扩展.

定义 2. CPN 的扩展可达图是一个有向图 $G=(V,E,R,v_0)$,其中,$V=\{v_0,v_1,\ldots,v_n\}$是可达节点的集合,每个节点 v_i 对应 CPN 模型可达图中的一个状态;$E=\{e_0,e_1,\ldots,e_m\}$是有向弧集,e_i 表示由一个可达节点到另一个可达节点的有向弧;$R=\{t_0,t_1,\ldots,t_m\}$是转换关系集,$R:E\rightarrow V\times V$,t_i 是弧上的旁标,且每个 t_i 对应 CPN 模型中的一个变迁;v_0 是扩展可达图的初始节点,扩展可达图的终点为 G 中没有出弧的节点,可能不唯一.

针对图 3 所示的可达图,根据定义 2 对其进行扩展,可得该模型的扩展可达图的多元组表示,如图 4 所示.

(1) V = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21 }.

(2) E = { 1→2, 2→3, 3→4, 4→5, 5→6, 6→7, 7→8, 8→9, 9→10, 10→11, 11→12, 12→13, 13→14, 14→15, 15→16, 16→17, 17→18, 18→19, 19→20, 20→21 }.

(3) R(e) =

sendPacket	若e∈{ 1→2, 6→7, 11→12, 16→17 }
transmitPacket	若e∈{ 2→3, 7→8, 12→13, 17→18 }
receivePacket	若e∈{ 3→4, 8→9, 13→14, 18→19 }
transmitAck	若e∈{ 4→5, 9→10, 14→15, 19→20 }
receiveAck	其他

Fig.4 The extended reachability graph of Fig.3

图 4 图 3 的扩展可达图

2 相关工作

CPN 形式化建模方法可用于网络协议、软件系统的建模与测试等方面.文献[9]对于软件定义网络的核心协议 OpenFlow 的交互属性进行层次 CPN 建模,并基于模型给出两种测试生成方法进行测试例的推导.文献[10]分析、研究了 Minix3 操作系统的源码,并采用 CPN 的形式化建模方法,基于属性,对操作系统调用进行了建模和测试研究.文献[11]中对并行软件系统采用 CPN 形式化方法建模,针对模型状态空间规模较大的问题,提出基于迹等价的化简模型的算法,在不影响测试结果的情况下缩小模型状态空间并提高测试效率.文献[12]中将 CPN 用于软件系统性能测试方面,使用 LoadRunner 性能测试工具获取 Web 应用软件中的相关性能参数,利用得到的参数表及参数公式,评价系统的性能是否符合最初的开发需求.文献[13]中提出了一种基于 CPN 的语义 Web 服务组合的验证与测试方法,达到了较好的效果,但是针对复杂的服务组合流程,可能会产生状态爆炸等问题.本文研究工作与文献[9–13]相比,虽然都是用 CPN 进行建模,但是被测不同,建模侧重点不同.就测试生成方法而言,上述文献并没有考虑依据被测执行过程中可能产生的反馈进行测试生成.

基于形式化方法进行测试的常规方法是建模、基于模型进行测试生成,可能会遇到状态空间爆炸问题和测试例生成过程的复杂性等问题,为了减少存储并避免出现状态空间爆炸问题,Jean-Claude Fernandez 等人首次提出 on-the-fly 方法的原型[14],将验证中的 on-the-fly 算法应用到软件测试中,即根据测试目的来选择测试例,进

而得到测试套;并且利用 on-the-fly 的思想实现了 TGV 工具[15].最后,通过实例证明 TGV 工具不仅能够实现 on-the-fly 测试算法,与手工生成测试例相比,该工具能够高效地生成更加完整和正确的测试例[16].

在 on-the-fly 测试中,进行测试例生成和执行时,根据逐步的测试执行结果指导状态集的生成[17];遍历可达图时,需要考虑被测实现(implementation under test,简称 IUT)的实际响应,进行路径选择并动态生成测试例[18].因此,on-the-fly 测试也可称为基于形式化模型的在线测试,以区别于测试生成和测试执行分开的离线测试.在测试运行时,也可以通过基于一个动态变化的概率分布选择动作,并提供用户指导的测试场景控制[19],测试时只需考虑实际执行过程的动作序列.on-the-fly 测试中通过直接观察具体的输出来指导相应的测试例生成,而不是预想系统的可能输出提前编写测试例[20].图 5 是一种 on-the-fly 测试机的架构,在测试过程中,通过观察被测系统的实际响应来减少需要搜索的状态空间,即测试驱动部分能够将测试模型部分产生的触发刺激输入给被测系统,并能够接受被测系统的反馈,使测试工作依据被测系统的具体反馈进行下一步操作[21,22].on-the-fly 测试的目标是根据被测系统的实际响应减少需要考虑的状态数和变迁数.

Fig.5 The architecture of on-the-fly tester

图 5 On-the-Fly 测试机架构

On-the-Fly 测试最初应用于简单系统的测试[23],随着对 on-the-fly 测试算法的不断改进,现已应用到含有实时和并发的系统中.在测试实时系统时,微软研究所将 on-the-fly 测试算法集成在基于模型的测试工具 Spec Explorer 中,使用的建模语言是 AdmL 或 Spec#,on-the-fly 测试的实现是根据动作的权重和动态改变的策略,选择实时系统的控制动作.使用 Spec Explorer 对聊天室系统进行测试时,发现该工具得出的部分结果与预期结果不符,其原因是模型的错误与 IUT 的代码错误,说明该工具设计的模型有待完善[19].此外,UPPAAL TRON[18]支持对实时系统的 on-the-fly 测试,它是对 on-the-fly 验证工具 UPPAAL 进行扩展得到的.文献[18]通过使用 UPPAAL TRON 工具对简单实时系统鼠标单击系统执行 on-the-fly 测试;又通过对复杂实时系统火车门控制器的测试,说明 UPPAAL TRON 能够解决更多实际问题.同时发现,在测试实时系统时,IUT 和环境的时钟匹配困难,说明对于 IUT 和环境的时间同步还存在问题.Lars Frantzen 等人实现了基于符号变迁系统形式化建模方法的 on-the-fly 测试,开发了 Jambition 工具,对 Alarm Dispatcher eHealth 服务系统进行验证和测试,并将自己方法的测试结果与传统测试结果进行比较,发现使用 on-the-fly 测试方法可以极大地减少测试例的设计和生成成本.但是由于测试数据选择的随机性,使得测试覆盖度较低,不能遍历模型中的每一个状态节点和变迁[20].在对 Web 应用系统或并发系统的测试方面,如何确保与系统规约的一致性以及如何自动生成测试例都是测试的研究重点.文献[24]基于自动机理论为 Web 应用提出一种 on-the-fly 测试方法.其中,Web 应用和测试目的使用有限状态机(finite state machine,简称 FSM)建模,用于选择测试用例.实验结果表明,on-the-fly 测试方法中的同步乘积算法在一定程度上减少了发生状态空间爆炸的可能性,但在执行两个 FSM 模型的同步乘积运算时复杂度较大.

文献[25]通过使用 NModel 工具对分布式 Web 应用系统进行基于模型的 on-the-fly 测试,实验结果表明,基于模型的 on-the-fly 测试可以发现传统测试中不能发现的错误.但是,其缺点是对 Web 应用系统的建模比较耗时,而且建立测试框架的过程比较复杂.文献[26]的作者 van Weelden 等人使用自己开发的 GAST 工具对 Java 小程序(smart card applet)进行 on-the-fly 测试,结果表明,该工具可以实现 on-the-fly 测试,并可自动发现错误.

通过以上分析可以发现,在对 on-the-fly 的测试研究中,已有与自动机、标记变迁系统等形式化方法结合的应用,但与 CPN 结合进行 on-the-fly 测试的研究还很少.与其他形式化建模方法相比,CPN 形式化建模方法能够很好地支持对复杂和并发系统的建模,并支持图形化;而且 CPN Tools 工具能够实现对模型的检查和仿真模

拟,并可自动生成和分析完整或部分状态空间.利用 CPN 形式化建模方法能够保证模型在安全性、活性方面正确性的优势,将其与 on-the-fly 测试方法相结合,不仅可以保障形式化建模时所要求的属性,也可以利用 on-the-fly 测试方法简化 CPN 模型的搜索状态空间,并可以对动态、实时甚至不确定的系统进行测试生成及执行.

3 基于 CPN 的 on-the-fly 测试方法

3.1 相关概念

3.1.1 系统规约

系统规约是基于需求规约描述被测系统的抽象模型,该模型能够预测每个特定状态下被测系统对将要发生的事件的可能响应[27].系统规约中规定了系统应该做什么和不应该做什么,同时它也是验证被测系统是否正确的依据[28].根据定义 2 可以得到系统规约扩展可达图的定义.

定义 3. 系统规约的扩展可达图是一个有向图 $S=(V^S,E^S,R^S,v_0^S)$,其中,V^S 是有限节点集,E^S 是弧集,R^S 是关系集,v_0^S 是初始节点,各个集合的解释与定义 2 类似.

3.1.2 测试目的

测试目的是指测试人员欲测试的被测系统的某一具体的功能,是使用形式化方法对被测系统中的部分行为的描述,这些行为一般是系统规约动作集中不完整的动作序列.测试目的的使用可以帮助测试人员在保证错误覆盖可接受的情况下减少生成测试例的数量,从而在一定程度上降低发生状态空间爆炸问题的可能性.测试目的模型可以根据设计的测试目的建模得到,也可以从完整的被测系统规约模型中获取.本文中需要构建被测系统的系统规约模型,因此,测试目的扩展可达图是在已建好的系统规约扩展可达图的基础上保留需要的节点及与之相关的弧而得到的.

定义 4. 测试目的的扩展可达图是一个有向图 $TP=(V^{TP},E^{TP},R^{TP},v_0^{TP})$,其中,$V^{TP}$ 是有限节点集,E^{TP} 是弧集,R^{TP} 是关系集,v_0^{TP} 是初始节点,各个集合的解释与定义 2 类似.

3.1.3 同步乘积

同步乘积是指同时遍历系统规约和测试目的的状态空间,并从系统规约中选取与测试目的相关的部分状态空间,进而实现仅对测试者所关心的部分状态空间进行测试例的生成和执行.系统规约扩展可达图与测试目的扩展可达图经同步乘积后,得到的同步乘积扩展可达图即为后续测试过程中需要搜索的状态空间.为描述本文的方法,下面给出同步乘积扩展可达图的概念.

定义 5. 同步乘积的扩展可达图是一个有向图 $SP=(V^{SP},E^{SP},R^{SP},v_0^{SP})$,其中,$V^{SP}$ 是有限节点集,E^{SP} 是弧集,R^{SP} 是关系集,v_0^{SP} 是初始节点,各个集合的解释与定义 2 类似.且满足如下条件.

(1) $R^{SP}\subseteq R^S$ 或 $R^{SP}\subseteq R^{TP}$,即同步乘积的关系集包含于系统规约(或测试目的)的关系集;

(2) 初始节点 $v_0^{SP}=(v_0^S,v_0^{TP})$,其他节点为 $v^{SP}=(v^S,v^{TP})$.在同步乘积扩展可达图中,每个节点都是由系统规约和测试目的中的节点组合构成的二元组;

(3) 假设 v_n^S 为 S 的终点,v_m^{TP} 为 TP 的终点,则 SP 的终点 $v_l^{SP}=(v_n^S,v_m^{TP})$.

3.1.4 测试例

在 on-the-fly 的测试方法中,测试例的生成和执行是同时进行的.首先对系统规约扩展可达图和测试目的扩展可达图进行同步乘积得到同步乘积的扩展可达图;然后根据被测系统的实际响应遍历同步乘积的扩展可达图,并最终得到测试例.如果被测系统实际运行的行为与遍历同步乘积扩展可达图得到的路径一致,则测试判定结果为“通过”,否则为“失败”.若测试执行以“通过”状态结束,则被测系统通过了该测试例的测试执行.下面给出测试例的扩展可达图的概念.

定义 6. 测试例的扩展可达图是一个有向图 $TC=(V^{TC},E^{TC},R^{TC},v_0^{TC})$,其中,$V^{TC}$ 是有限节点集,E^{TC} 是弧集,R^{TC} 是关系集,v_0^{TC} 是初始节点,各个集合的解释与定义 2 类似.

3.2 基于 CPN 的 on-the-fly 测试方法

On-the-Fly 测试方法首先是依据某个测试目的选出测试者关心的那部分系统功能模型,然后根据被测系统的实际响应来生成测试例,并且测试例的生成和执行是同步进行的.on-the-fly 测试方法的本质是对生成的同步乘积可达图进行遍历,可以避免传统测试方法中需对整个系统规约状态空间进行遍历以及生成测试例数目多且存在冗余的问题.本文中的 on-the-fly 测试方法实现过程可以分为 3 个阶段.

(1) 建模阶段:若系统比较复杂,可采用层次化的 CPN 建模.然后使用仿真工具和状态空间工具对系统规约模型进行执行,生成系统规约的状态空间可达图,根据定义 3 对其扩展即可得到系统规约的扩展可达图,从中选取测试目的得到测试目的的扩展可达图;

(2) 生成同步乘积阶段:运行同步乘积算法,深度优先遍历系统规约和测试目的的扩展可达图得到同步乘积扩展可达图,即后续测试过程需要搜索的状态空间;

(3) 生成和执行测试例阶段:以被测系统的实际响应作为遍历同步乘积扩展可达图过程中选择路径的条件,系统实际执行的路径即为得到的测试例[20].与被测系统的交互过程通过适配器实现.

On-the-Fly 测试方法实现过程如图 6 所示.

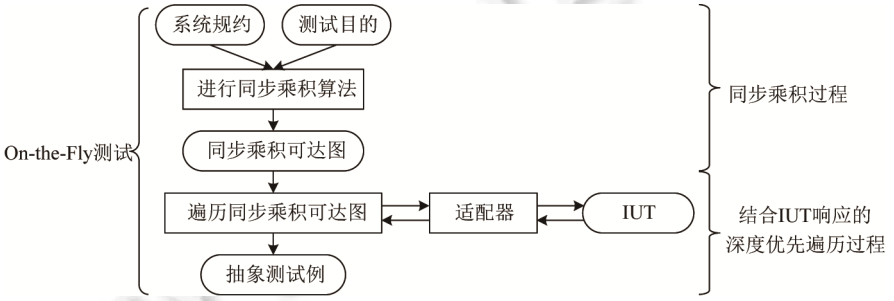

Fig.6 The process of on-the-fly testing method

图 6 On-the-Fly 测试方法实现过程

3.3 同步乘积算法

3.3.1 同步乘积算法描述

从图 6 中可以看出,同步乘积算法是 on-the-fly 测试中的核心算法.根据定义 3~定义 5,同步乘积算法的工作原理如下.

(1) 若 $t_a^S:e_k^S\rightarrow v_i^S\times v_j^S,t_b^{TP}:e_l^{TP}\rightarrow v_m^{TP}\times v_n^{TP}$,其中,$e_k^S$ 表示系统规约 S 中的弧,v_i^S 表示 e_k^S 的起点,v_j^S 表示 e_k^S 的终点,t_a^S 表示 e_k^S 上的变迁.其余符号定义与此类似,且 $t_a^S=t_b^{TP},v_n^{TP}$ 不是测试目的的终点,则有 $t_c^{SP}=t_a^S,t_c^{SP}:e_h^{SP}\rightarrow(v_i^S,v_m^{TP})\times(v_j^S,v_n^{TP})$;

(2) 若 $t_a^S:e_k^S\rightarrow v_i^S\times v_j^S,t_b^{TP}:e_l^{TP}\rightarrow v_m^{TP}\times v_n^{TP}$,且 $t_a^S\neq t_b^{TP},v_n^{TP}$ 不是测试目的的终点,则有 $t_c^{SP}=t_a^S,t_c^{SP}:e_h^{SP}\rightarrow(v_i^S,v_m^{TP})\times(v_j^S,v_m^{TP})$.其中,$(v_j^S,v_m^{TP})$为同步乘积 SP 中的当前节点;

(3) 若 v_n^{TP} 为测试目的的终点,则在系统规约 S 中搜索从 v_j^S 到终点的一条最短路径,最短路径中的每个节点分别与 v_n^{TP} 组合构成同步乘积 SP 中的节点,SP 中弧上的变迁与系统规约 S 中相同.

为了更加清晰地理解同步乘积算法,给出其流程图,如图 7 所示.

同步乘积算法的伪代码如图 8 所示.其中,

算法输入:系统规约 $S=(V^S,E^S,R^S,v_0^S)$和测试目的 $TP=(V^{TP},E^{TP},R^{TP},v_0^{TP})$.

算法输出:同步乘积 $SP=(V^{SP},E^{SP},R^{SP},v_0^{SP})$.

初始化:(1) 同步乘积 SP 初始节点 $v_0^{SP}=(v_0^S,v_0^{TP})$;(2) 将 S 和 TP 中所有的弧置为未访问,且弧的访问次数置为 0;(3) $v_i^S=v_0^S,v_m^{TP}=v_0^{TP}$.

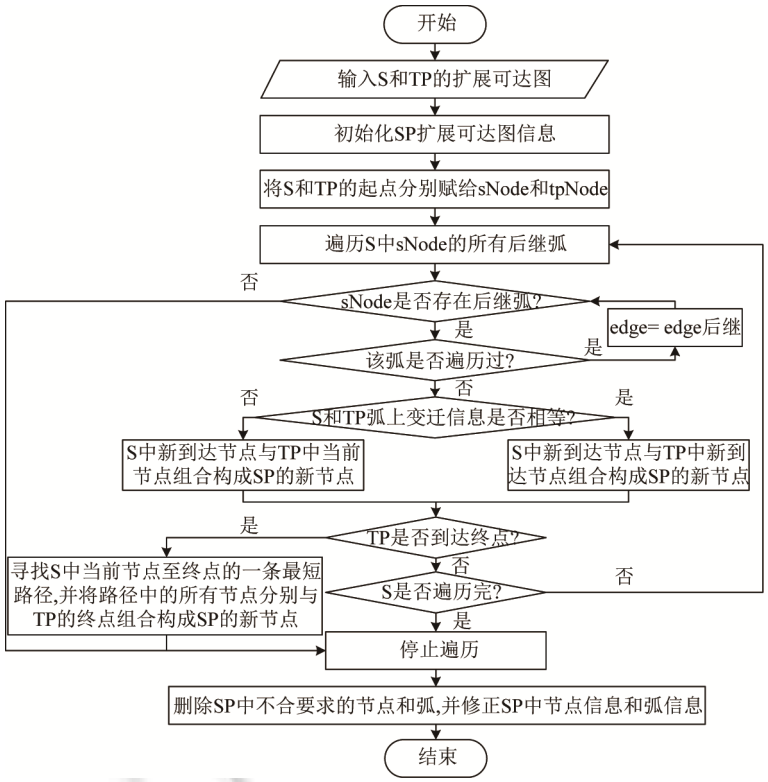

Fig.7 The flow chart of the synchronous product algorithm

图 7 同步乘积算法实现的流程图

```
while v_i^S存在后继
    e^S = v_i^S的邻接弧;
    while e^S已被访问过
        if e^S后继不为空
            e^S=e^S的后继;
        else
            break;
        end if
    end while
    if e^S已被访问过
        return;
    else
        if v_m^TP不是TP的终点
            e^TP = v_m^TP的邻接弧;
            if t_a^S: e_k^S→v_i^S×v_j^S, t_b^TP: e_i^TP→v_m^TP×v_n^TP, 且t_a^S = t_b^TP
                SP中增加节点(v_j^S, v_n^TP);
                SP中增加弧t_a^S: e_c^SP→(v_i^S, v_m^TP)×(v_j^S, v_n^TP); v_p^SP = (v_i^S, v_m^TP);
                if v_p^SP后继为空
                    v_p^SP后继弧 = e_c^SP;
                end if
                e_k^S访问次数加1;
                if e_k^S访问次数≥v_i^S的前驱数
                    将e_k^S置为已访问;
                end if
                v_i^S = v_j^S; v_m^TP = v_n^TP; v_p^SP =  (v_j^S, v_n^TP);
                递归调用同步乘积算法;
                if v_i^S为S的终点
                    return;
                end if
            else if t_a^S: e_k^S→v_i^S×v_j^S, t_b^TP: e_i^TP→v_m^TP×v_n^TP, 且t_a^S ≠ t_b^TP
                SP中增加节点(v_j^S, v_m^TP);
                SP中增加弧t_a^S: e_c^SP→(v_i^S, v_m^TP)×(v_j^S, v_m^TP); v_p^SP = (v_i^S, v_m^TP);
                if v_p^SP后继为空
                    v_p^SP后继弧 = e_c^SP;
                end if
                e_k^S访问次数加1;
                if e_k^S访问次数≥v_i^S的前驱数
                    将e_k^S置为已访问;
                end if
                v_i^S = v_j^S; v_p^SP = (v_j^S, v_n^TP);
                递归调用同步乘积算法;
                if v_i^S为S的终点
                    return;
                end if
            end if
        else if v_m^TP为TP的终点
            if e^S终点的后继弧为空
                SP中增加节点(e^S终点, v_m^TP);
                SP中增加弧t_a^S: e_c^SP→(v_i^S, v_m^TP)×(e^S终点, v_m^TP);
                e^S访问次数加1;
                if e^S访问次数≥v_i^S的前驱数
                    e^S置为已访问;
                end if
                return;
            else
                spath = ShortPath (S, e^S);
                for 遍历最短路径
                    构建SP后续节点和弧;
                end for
            end if
        end if
    end if
    if 当前访问弧sEdge无后继
        return;
    else
        sEdge赋值为其后继; 递归调用同步乘积算法;
    end if
end while
```

Fig.8 Pseudo code of the synchronous product algorithm

图 8 同步乘积算法的伪代码

根据图 8 所示的同步乘积算法的伪代码,本文使用 Java 语言编程实现了同步乘积算法.经分析,该算法在最坏情况下的时间复杂度为 $O(n^3)$,空间复杂度为 $O(n)$.

3.3.2 同步乘积算法测试

为确保算法的正确性,本节构造两个 CPN 模型,实现对该算法的测试,如图 9 和图 10 所示.其中,图 9 为含有两个并发变迁的系统规约 S1 的 CPN 模型,图 10 为含有 3 个并发变迁的系统规约 S2 的 CPN 模型.模型中涉及的颜色集有整型 INT,变量有整型 a.初始库所 a1 中含有一个 token 为 1`2.

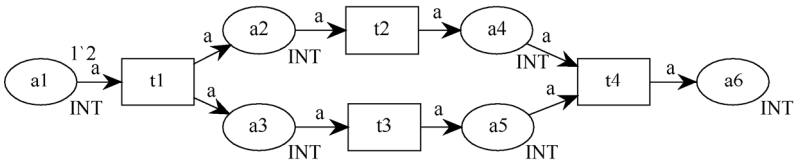

Fig.9 The CPN model of system specification S1 with two concurrencies

图 9 含有两个并发变迁的系统规约 S1 的 CPN 模型

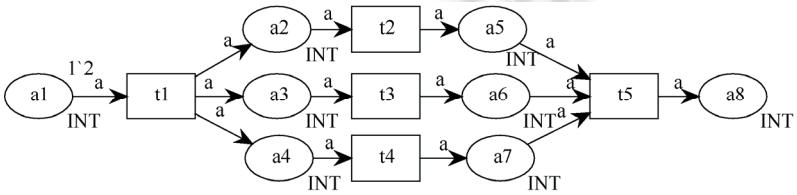

Fig.10 The CPN model of system specification S2 with three concurrencies

图 10 含有 3 个并发变迁的系统规约 S2 的 CPN 模型

根据并发变迁覆盖原则和边界覆盖原则设计了如表 1 所示的测试例,其中涉及的系统规约与测试目的的符号描述见表 2.

Table 1 The test cases design of synchronous product algorithm

表 1 同步乘积算法的测试例设计

测试目的	系统规约	
	含 2 个并发变迁	含 3 个并发变迁
含 1 个并发变迁	S1×TP12, S1×TP13	S2×TP23
含 2 个并发变迁	S1×TP11	S2×TP22
含 3 个并发变迁	–	S2×TP21
部分变迁不属于系统规约	S1×TP14	S2×TP24
所有变迁都不属于系统规约	S1×TP15	S2×TP25

Table 2 The symbolic description of system specifications and test purposes

表 2 系统规约和测试目的的符号描述

名称		描述
系统规约	S1	含 t2、t3 这 2 个并发变迁的系统
	S2	含 t2、t3 和 t4 这 3 个并发变迁的系统
测试目的	TP11	t2→t3
	TP12	t3→t4
	TP13	t1→t3
	TP14	t1→t5
	TP15	t5→t6
	TP21	t3→t4→t2
	TP22	t2→t4
	TP23	t3→t5
	TP24	t1→t6
	TP25	t6→t7

利用 CPN Tools 工具分别执行系统规约 S1 和 S2 的 CPN 模型并生成模型的可达图,然后扩展系统规约的可达图,得到其扩展可达图,分别如图 11 和图 12 所示.

同步乘积算法中的测试目的扩展可达图是从已得到的系统规约扩展可达图中选取的,保留了所需的节点及相关的弧.在测试过程中,从系统规约 S1 和 S2 中分别提取 3 条测试目的,见表 2.以 TP22 为例,其扩展可达图如图 13 所示.

以系统规约 S2 与测试目的 TP22 进行同步乘积运算为例,算法运行结果如图 14 所示,得到的同步乘积 SP22 的扩展可达图如图 15 所示.

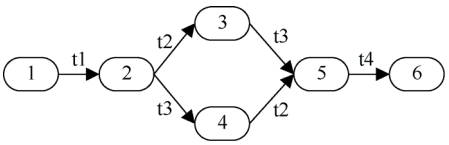

Fig.11 The extended reachability graph of S1
图 11 系统规约 S1 的扩展可达图

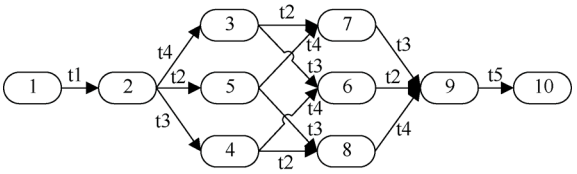

Fig.12 The extended reachability graph of S2
图 12 系统规约 S2 的扩展可达图

同步乘积可达图节点信息:

ID	前驱节点数	邻接弧
1,1	0	ID1412375042
2,1	1	ID1412375182
5,2	1	ID1412375329
8,2	2	ID1412375468
9,3	2	ID1412375605
10,3	1	空
7,3	1	ID1412375875
4,1	1	ID1412376327

同步乘积可达图弧信息:(共有 3 条路径)

ID	始点	终点	变迁	邻接弧
ID1412375042	1,1	2,1	t1	空
ID1412375182	2,1	5,2	t2	ID1412375187
ID1412375329	5,2	8,2	t3	ID1412375334
ID1412375468	8,2	9,3	t4	空
ID1412375605	9,3	10,3	t5	空
ID1412375334	5,2	7,3	t4	空
ID1412375875	7,3	9,3	t3	空
ID1412375187	2,1	4,1	t3	空
ID1412376327	4,1	8,2	t2	空

Fig.14 The execution result of S2×TP22
图 14 同步乘积 S2×TP22 的运行结果

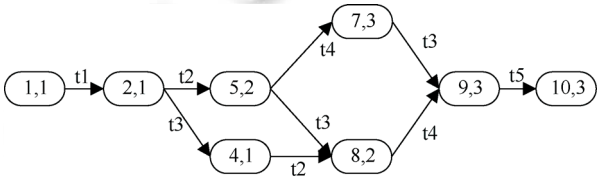

Fig.15 The extended reachability graph of SP22
图 15 同步乘积 SP22 的扩展可达图

根据表 1 中设计的测试例,分别将系统规约与测试目的进行同步乘积运算,得到所有测试例的运行结果,见表 3.

从表 3 中可以看出,同一系统规约与不同的测试目的进行同步乘积运算后,得到的同步乘积扩展可达图不同.系统规约扩展可达图中与测试目的相关的路径都在同步乘积扩展可达图中保留,不相关的路径均已被删除.

此外,通过分析表 3 也可得出结论:同步乘积算法可以实现在系统规约扩展可达图中选择与测试目的相关的部分可达图.而且,测试目的中是否包含并发变迁以及含有的可并发变迁的不同数量,都会在一定程度上对生成的同步乘积扩展可达图的规模产生影响.

Table 3 The test results of the synchronous product algorithm

表 3 同步乘积算法的测试结果

编号	测试过程	预期结果	实际结果	测试判定
1	输入:S1 与 TP11 (S1 有 2 条路径)	输出 1 条可达路径 1,1→2,1→3,2→5,3→6,3	输出 1 条可达路径 1,1→2,1→3,2→5,3→6,3	通过
2	输入:S1 与 TP12 (S1 有 2 条路径)	输出 2 条可达路径 1,1→2,1→3,1→5,2→6,3 1,1→2,1→4,2→5,2→6,3	输出 2 条可达路径 1,1→2,1→3,1→5,2→6,3 1,1→2,1→4,2→5,2→6,3	通过
3	输入:S1 与 TP13 (S1 有 2 条路径)	输出 2 条可达路径 1,1→2,2→3,2→5,3→6,3 1,1→2,2→4,3→5,3→6,3	输出 2 条可达路径 1,1→2,2→3,2→5,3→6,3 1,1→2,2→4,3→5,3→6,3	通过
4	输入:S1 与 TP14 (S1 有 2 条路径)	输出 0 条可达路径	输出 0 条可达路径	通过
5	输入:S1 与 TP15 (S1 有 2 条路径)	输出 0 条可达路径	输出 0 条可达路径	通过

Table 3 The test results of the synchronous product algorithm (Continued)
表 3 同步乘积算法的测试结果(续)

编号	测试过程	预期结果	实际结果	测试判定
6	输入:S2 与 TP21 (S2 有 6 条路径)	输出 1 条可达路径 1,1→2,1→4,2→6,3→9,4→10,4	输出 1 条可达路径 1,1→2,1→4,2→6,3→9,4→10,4	通过
7	输入:S2 与 TP22 (S2 有 6 条路径)	输出 3 条可达路径 1,1→2,1→5,2→8,2→9,3→10,3 1,1→2,1→5,2→7,3→9,3→10,3 1,1→2,1→4,1→8,2→9,3→10,3	输出 3 条可达路径 1,1→2,1→5,2→8,2→9,3→10,3 1,1→2,1→5,2→7,3→9,3→10,3 1,1→2,1→4,1→8,2→9,3→10,3	通过
8	输入:S2 与 TP23 (S2 有 6 条路径)	输出 6 条可达路径 1,1→2,1→5,1→8,2→9,2→10,3 1,1→2,1→5,1→7,1→9,2→10,3 1,1→2,1→4,2→8,2→9,2→10,3 1,1→2,1→4,2→6,2→9,2→10,3 1,1→2,1→3,1→7,1→9,2→10,3 1,1→2,1→3,1→6,2→9,2→10,3	输出 6 条可达路径 1,1→2,1→5,1→8,2→9,2→10,3 1,1→2,1→5,1→7,1→9,2→10,3 1,1→2,1→4,2→8,2→9,2→10,3 1,1→2,1→4,2→6,2→9,2→10,3 1,1→2,1→3,1→7,1→9,2→10,3 1,1→2,1→3,1→6,2→9,2→10,3	通过
9	输入:S2 与 TP24 (S2 有 6 条路径)	输出 0 条可达路径	输出 0 条可达路径	通过
10	输入:S2 与 TP25 (S2 有 6 条路径)	输出 0 条可达路径	输出 0 条可达路径	通过

3.4 结合 IUT 实际响应的遍历

On-the-Fly 测试方法实现的第 3 阶段是生成和执行测试例.其实现过程是在遍历同步乘积扩展可达图的同时,与 IUT 进行实时交互,并根据 IUT 的实际响应从同步乘积扩展可达图中选择路径,生成和执行测试例.该过程中,交互主要包括两个方面.

(1) 将遍历同步乘积的输入,即触发变迁执行的输入转化为 IUT 可以接受的输入,使 IUT 执行下一步操作;

(2) 将 IUT 的实际响应转化为 CPN 模型中的变迁行为,并将其作为遍历同步积扩展可达图中的选择路径的条件,驱动遍历过程进一步执行以得到表示被测系统实际行为的路径.

这两种数据传递可以使用一个连接同步乘积遍历过程和 IUT 的适配器来实现.

4 示例及测试分析

学籍管理系统是一种常见的 Web 应用系统,具有并发性、实时性和交互性等特点,本文以其为被测,示例层次 CPN 结合 on-the-fly 的测试方法,进行测试例的生成和执行.本节首先对被测系统进行建模,设计不同的场景执行模型并得到各场景的可达图;然后依据 on-the-fly 方法的实现过程,通过适配器实现与被测系统的交互,生成和执行测试例;最后,将得到的测试结果与使用传统测试方法得到的测试结果进行比较和分析.

4.1 学籍管理系统的 CPN 模型

学籍管理系统主要包括两类用户:管理员和普通用户.管理员具有添加、修改和删除的功能,而普通用户只能查看系统中的信息.系统中的用户均需要登录系统后才能实现相应的功能,未注册用户不能进入系统.为便于分析问题,本文对学籍管理系统采用分角色分层次的建模方法,对系统中添加学生信息模块、查询学生列表模块和按关键字查询学生信息模块等代表性功能模块进行建模.其中,涉及的库所和变迁的含义见表 4 和表 5.模型中涉及的颜色集、变量和函数声明如图 16 所示.

Table 4 The meaning of places in the CPN model
表 4 CPN 模型中库所的含义

库所名称	类型	含义	库所名称	类型	含义
start	USER	初始状态	hasUser	USER	已注册用户
noUser	user	未注册用户	customer	user	已登录用户
admin	user	管理员	user	user	普通用户
nStuInfo	STUINFO	待添加学生信息	info	KEYWORD	待搜索关键字
finish	user	操作完成	end	user	结束
nnStuInfo	user	成功添加的学生信息	hasRecord	user	成功查询的记录

Table 5 The meaning of transitions in the CPN model

表 5 CPN 模型中变迁的含义

变迁名称	变迁含义	变迁名称	变迁含义
judge	判断用户注册信息	login	登录
isAdmin	判断是否是管理员	search	转向搜索
newStuInfo	添加学生信息	searchStuList	查询学生信息列表
searchStuInfo	按关键字查询学生信息	logout	退出系统
addStuInfo	添加学生信息	succAdd	成功添加学生信息
queryStuInfo	查询学生信息	succQuery	成功查询到学生信息
exit	结束		

```
Standard declarations
------------------------------------------------------------------------------------------------------------------------------------------------
colset BOOL= bool;                                                    var uname, upwd: STRING;
colset INT= int;                                                      var exist, login: BOOL;
colset STRING= string;                                                var role: INT;
colset USER = product STRING * STRING * BOOL * INT * BOOL;            var stuID, stuName, stuGender, stuNative: STRING;
colset user = product STRING * INT * BOOL;                            var stateAdd, stateQuery: BOOL;
colset STUINFO = product STRING * STRING * STRING * STRING * BOOL;    var keyword: STRING;

fun JudgeYUser ( uname, upwd, exist, role, login ) = if ( exist = true ) then 1`( uname, upwd, exist, role, true) else empty;
fun JudgeNUser ( uname, upwd, exist, role, login ) = if ( exist = false ) then 1`( uname, role, login ) else empty;
fun JudgeLoginInfo ( uname, upwd, exist, role, login ) = if ( login = true ) then 1`( uname, role, login ) else empty;
fun JudgeYAdmin ( uname, role, login ) = if ( role = 1 ) then 1`( uname, role, login ) else empty;
fun JudgeNAdmin ( uname, role, login ) = if ( role <> 1) then 1`( uname, role, login ) else empty;
fun Logout ( uname, role, login ) = 1`(uname, role, false)
fun AddStuSucc ( uname, role, login, stateAdd ) = if ( stateAdd = true ) then 1`( uname, role, login ) else empty;
fun AddStuFail ( uname, role, login, stateAdd ) = if ( stateAdd = false ) then 1`( uname, role, login ) else empty;
fun QueryStuSucc ( uname, role, login, stateQuery ) = if ( stateQuery = true ) then 1`( uname, role, login ) else empty;
fun QueryStuFail ( uname, role, login, stateQuery ) = if ( stateQuery = false ) then 1`( uname, role, login ) else empty;
------------------------------------------------------------------------------------------------------------------------------------------------
```

Fig.16 The statements of the colour sets, variables and functions in the CPN model

图 16 CPN 模型中颜色集、变量和函数声明

在系统模型的建立过程中,首先对系统的模型进行整体设计,建立包含系统各功能模块的顶层模型,用替代变迁表示各功能,并在子页中具体实现该功能的详细流程,进而形成系统的层次模型.图 17 为学籍管理系统的 top 顶层模型,该页中展示了系统中新添加学生、查询学生列表和按关键字查询学生信息等功能的实现流程,其中包含两个替代变迁 newStuInfo 和 searchStuInfo.

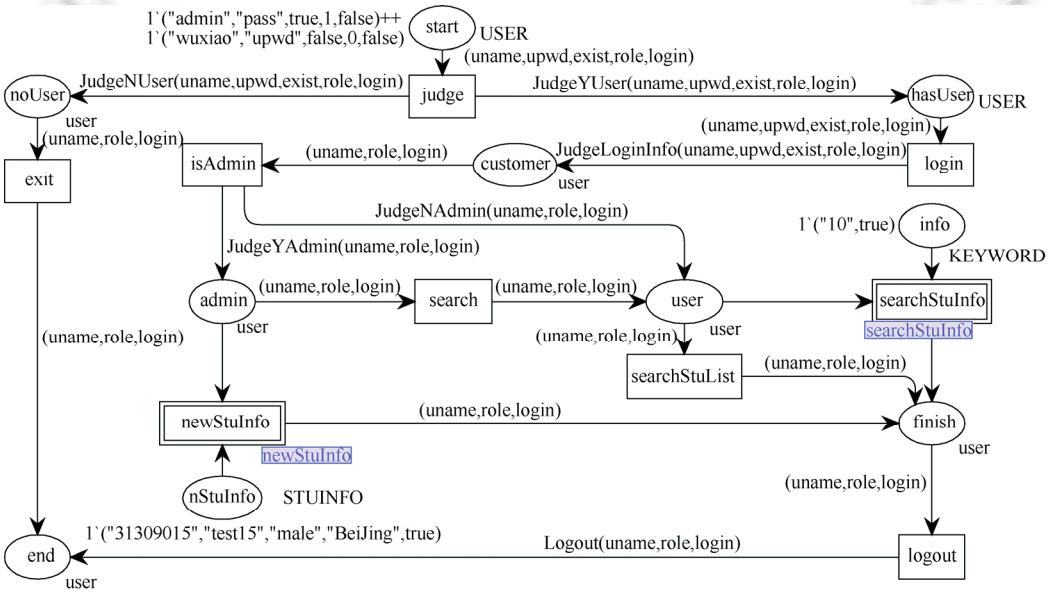

Fig.17 The top level model of student roll management system

图 17 学籍管理系统的顶层模型

图17中的替代变迁newStuInfo和searchStuInfo对应的子页面分别如图18和图19所示.图18中,newStuInfo子页面主要描述管理员添加学生信息操作的过程.图19中,searchStuInfo子页面主要描述管理员或普通用户根据关键字查询学生信息的操作过程.

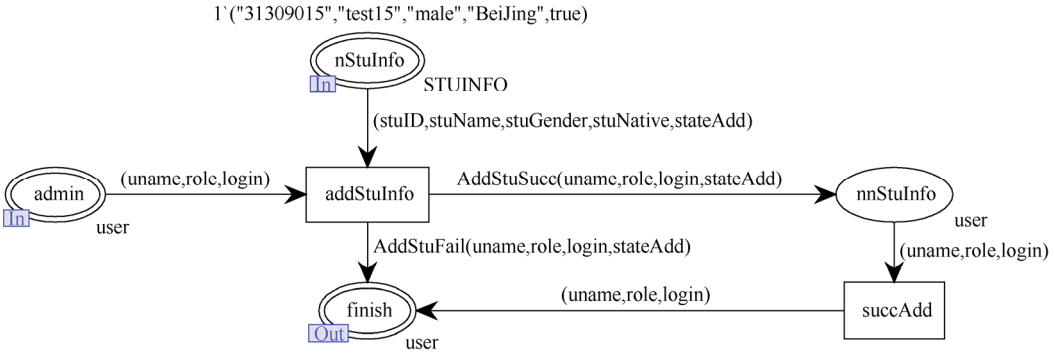

Fig.18 The sub-page of substitution transition newStuInfo

图18 替代变迁newStuInfo子页

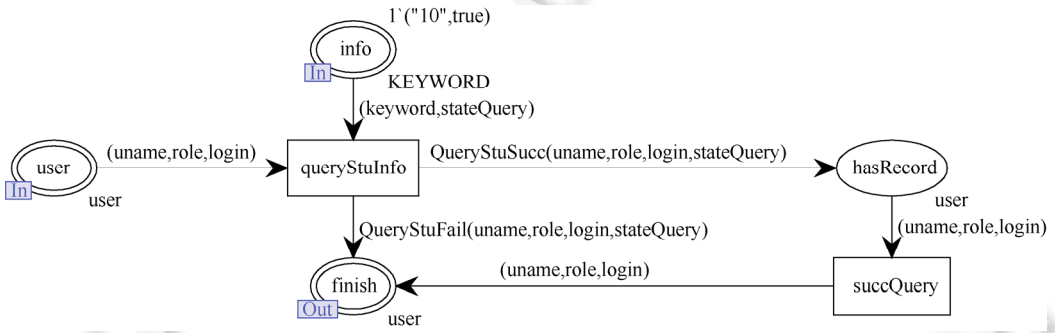

Fig.19 The sub-page of substitution transition searchStuInfo

图19 替代变迁searchStuInfo子页

4.2 测试生成过程

在完成对学籍管理系统的建模后,设置不同的场景运行模型,从而得到模型的可达图.然后使用 on-the-fly 测试方法对其进行测试生成和测试执行.

一般情况下,m 个用户登录系统时,系统规约的状态可达图是包含顺序和多并发的.本文根据 token 数量和变迁覆盖原则设计了表6所示的测试场景.

Table 6 The design of test scenario of student roll management system

表6 学籍管理系统的测试场景设计

测试场景	TP1:新添加学生信息	TP2:按关键字查询学生信息
场景1.S1:1个管理员用户,1条学生信息,1条关键字信息	S1×TP1	S1×TP2
场景2.S2:1个管理员用户,1个未注册用户,1条学生信息,1条关键字信息	S2×TP1	S2×TP2
场景3.S3:1个管理员用户,1个普通用户,1条学生信息,1条关键字信息	S3×TP1	S3×TP2

假设已建立系统的CPN模型,需要在系统中添加一条学生信息,或根据一条关键字信息查询所有符合要求的学生,下面分别以场景1和场景2为例,展示本文的方法.其中,两种测试场景中均包含了管理员用户添加一条学生信息的行为过程和根据关键字“10”查看符合条件的学生信息的行为过程,因此,本文以这两种行为作为测试目的进行测试,如表6中TP1和TP2所示.TP1表示管理员用户登录系统后添加学生信息的行为序列,TP2表

示管理员或普通用户根据关键字查看学生信息的行为序列.本文中测试目的的扩展可达图是从系统规约扩展可达图中选取的.

场景 1:当系统模型中的 token 为一个管理员用户、一条学生信息和一条关键字信息时,即模型的“start”库所中有一个 token("admin","pass",true,1,false)表示有一个管理员用户进入系统,用户名为“admin”,密码为“pass”,其当前状态是未登录;“nStuInfo”库所中有一个 token("31309015","test15","male","BeiJing",true)表示待添加学生的相关信息;“info”库所中有一个 token("10", true)表示待查询的关键字“10”.利用 CPN Tools 工具执行 CPN 层次模型得到该模型的可达图,并对其进行扩展转化为扩展可达图,如图 20 所示,其弧上信息仅以变迁名称表示,受篇幅所限,其详细信息省略,下同.

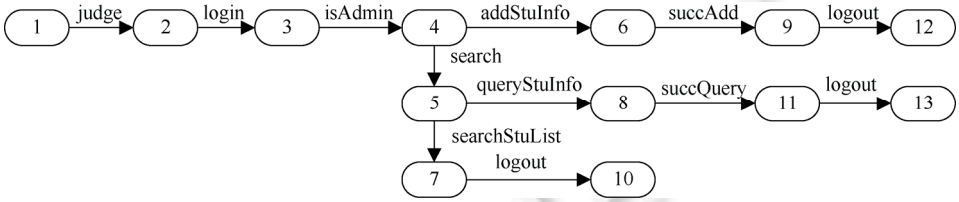

Fig.20 The extended reachability graph of S1

图 20 系统规约 S1 的扩展可达图

从系统规约 S1 的扩展可达图中选取测试目的得到测试目的的扩展可达图 TP1 和 TP2,如图 21 和图 22 所示.

Fig.21 The extended reachability graph of TP1

图 21 测试目的 TP1 的扩展可达图

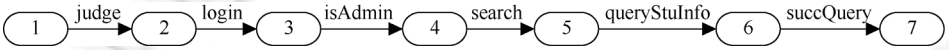

Fig.22 The extended reachability graph of TP2

图 22 测试目的 TP2 的扩展可达图

将系统规约 S1 分别与测试目的 TP1 和 TP2 的扩展可达图作为同步乘积算法的输入,即通过运行同步乘积算法将系统规约扩展可达图中覆盖测试目的的部分图形选择出来,并得到同步乘积的扩展可达图 S1×TP1 和 S1×TP2,如图 23 和图 24 所示.

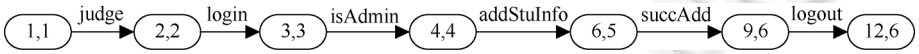

Fig.23 The extended reachability graph of S1×TP1

图 23 同步乘积 S1×TP1 的扩展可达图

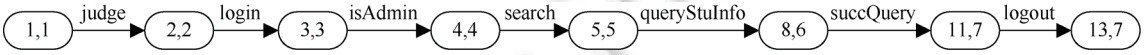

Fig.24 The extended reachability graph of S1×TP2

图 24 同步乘积 S1×TP2 的扩展可达图

以遍历 S1×TP1 为例,需要模拟用户运行学籍管理系统,并根据其实时响应来选择测试生成和执行的路径.在此过程中,将会进一步削减同步乘积扩展可达图中关于测试目的的冗余路径和无关路径,得到满足系统实时响应的测试例.通过适配器驱动学籍管理系统运行的程序截图如图 25 和图 26 所示.其中,图 25 所示为成功添加学生信息的情况,“同步乘积可达图信息”域中给出了生成的同步乘积可达图的节点信息和弧信息,以及可达图中的路径数信息等;“程序运行过程”域中给出了测试机与学籍管理系统交互过程中的执行情况;“生成的路径信

息”域中给出了生成的路径,最终得到的路径为“1,1→2,2→3,3→4,4→6,5→9,6→12,6”.该路径中“12,6”为同步乘积扩展可达图的终点,因此,该路径表示的测试例的判定结果为“通过”.图 26 所示为添加学生信息失败的情况,新添加的学生信息为“31309015,test15,male,Beijing”,而数据库中已存在学号为“31309015”的记录,因而添加失败,最终得到的路径为“1,1→2,2→3,3→4,4”.由于“4,4”不是同步乘积可达图的终点,因此该路径表示的测试例的判定结果为“失败”.

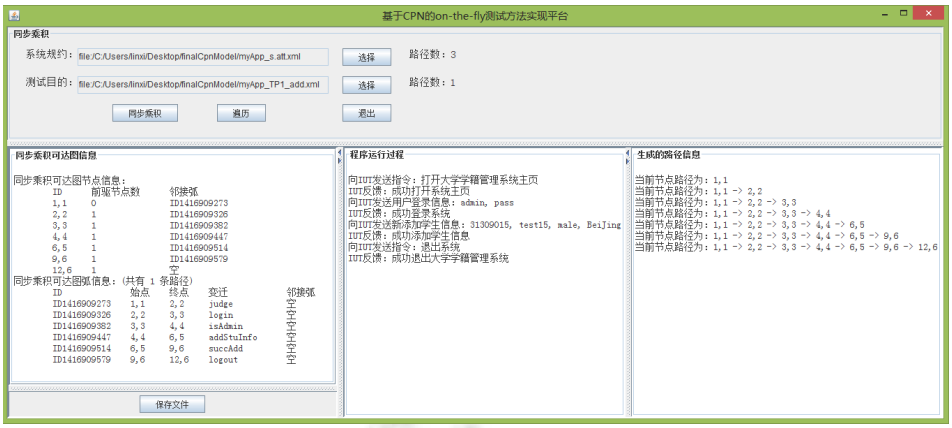

Fig.25 Succeed to add student information

图 25 成功添加学生信息

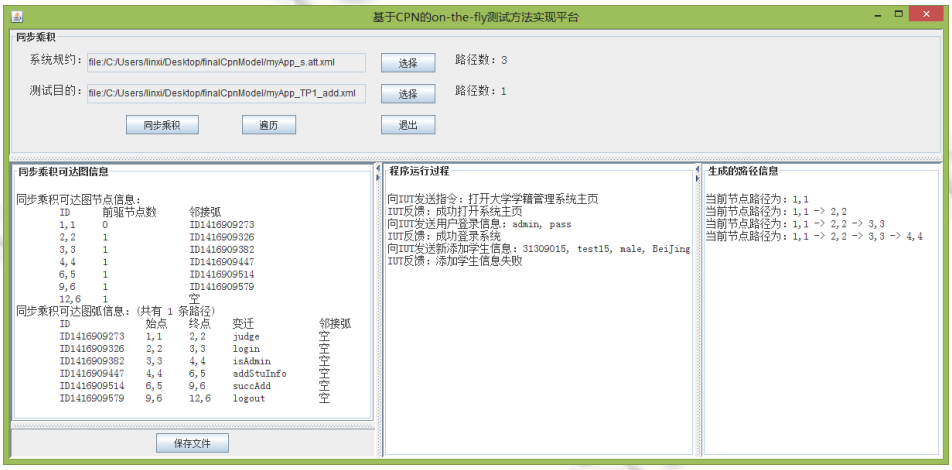

Fig.26 Failed to add student information

图 26 添加学生信息失败

场景 2:当系统模型中的 token 为一个管理员用户、一个未注册用户、一条学生信息和一条关键字信息时,即模型的“start”库所中有两个 token("admin","pass",true,1,false)和("wuxiao","upwd",false,0,false),“nStuInfo”库所中有一个 token("31309015","test15","male","BeiJing",true),“info”库所中有一个 token("10",true).其中,token ("wuxiao","upwd",false,0,false)表示未注册用户,不能进入学籍管理系统,其他各 token 的含义同场景 1.利用 CPN Tools 工具执行 CPN 层次模型得到该模型的可达图,并对其进行扩展转化为扩展可达图,如图 27 所示.

此场景中的测试目的与场景 1 中的测试目的相同,这里不再赘述.将系统规约 S2 分别与测试目的 TP1 和 TP2 的扩展可达图作为同步乘积算法的输入,即通过运行同步乘积算法将系统规约扩展可达图中覆盖测试目的的部分图形选择出来,并得到同步乘积的扩展可达图 S2×TP1 和 S2×TP2,如图 28、图 29 所示.

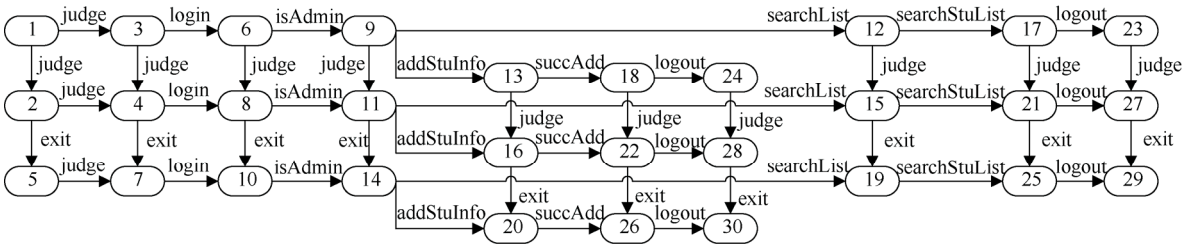

Fig.27 The extended reachability graph of S2

图 27 系统规约 S2 的扩展可达图

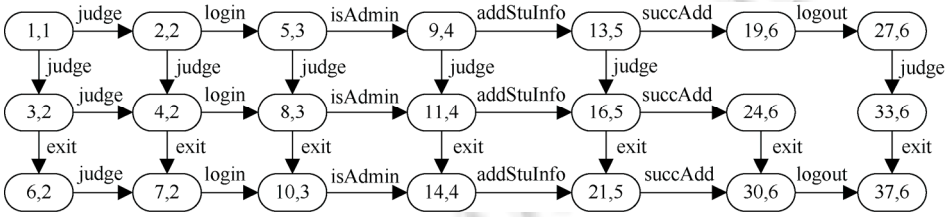

Fig.28 The extended reachability graph of S2×TP1

图 28 同步乘积 S2×TP1 的扩展可达图

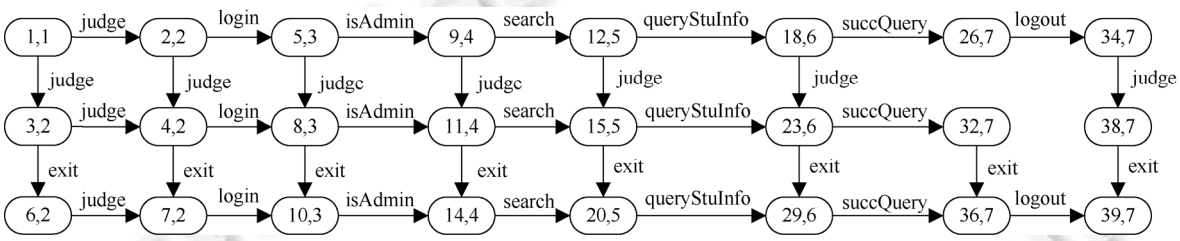

Fig.29 The extended reachability graph of S2×TP2

图 29 同步乘积 S2×TP2 的扩展可达图

以遍历 S2×TP2 为例,需要模拟用户运行学籍管理系统,并根据其实时响应来选择测试生成和执行的路径.通过适配器驱动学籍管理系统运行的程序截图如图 30 所示.图中各文本域的含义如场景 1,这里不再赘述.场景 1 中最终得到的路径为“1,1→2,2→5,3→9,4→12,5→18,6→26,7→34,7→38,7→39,7”,且“39,7”为同步乘积扩展可达图的终点,因此,该路径表示的测试例的判定结果为“通过”.

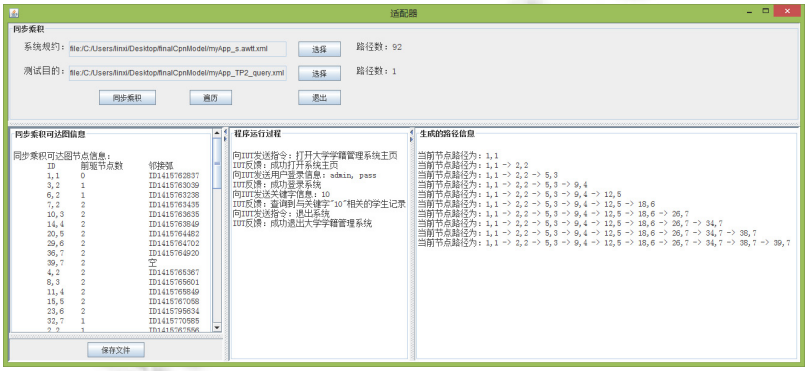

Fig.30 Succeed to search student information

图 30 成功查询到学生信息

4.3 测试结果分析

从图 23、图 24、图 28 和图 29 中可以看出,当系统中用户数增加时,生成的状态空间明显增大.执行同步乘积后,可达图的规模有所减小.在此过程中,削减了系统规约扩展可达图中与测试目的无关的、冗余的路径.在对同步乘积 S×TP 的扩展可达图进行遍历时,根据系统的实时响应来选择测试生成和执行路径.最终可以得到满足系统实时响应的测试例.使用本文方法对表 6 中设计的测试场景进行测试,测试结果见表 7.

Table 7 The test result of on-the-fly testing method

表 7 On-the-Fly 测试方法的测试结果

测试场景	系统规约 S 路径数	测试目的 TP	同步乘积 S×TP 路径数	实际执行路径
S1	3	TP1	1	1,1→2,2→3,3→4,4→6,5→9,6→12,6
		TP2	1	1,1→2,2→3,3→4,4→5,5→8,6→11,7→13,7
S2	92	TP1	21	1,1→2,2→5,3→9,4→13,5→19,6→27,6→33,6→37,6
		TP2	28	1,1→2,2→5,3→9,4→12,5→18,6→26,7→34,7→38,7→39,7
S3	3 564	TP1	658	1,1→3,2→6,3→10,4→17,5→28,6→42,6→55,6→69,6→83,6→96,6→102,6→104,6
		TP2	798	1,1→3,2→6,3→10,4→16,5→27,6→41,7→56,7→70,7→84,7→93,7→100,7→103,7

使用传统测试方法进行测试时需要遍历系统规约的完整状态空间,而本文方法只需遍历与测试目的同步乘积后得到的同步乘积可达图即可.针对表 6 中的测试场景,分别采用传统测试方法和 on-the-fly 测试方法进行测试,对两种测试方法中需要遍历的状态空间进行比较,见表 8.

Table 8 The comparison of the state space to be traversed in traditional testing and on-the-fly testing

表 8 传统测试与 on-the-fly 测试中的待遍历状态空间比较

测试场景	测试目的	传统测试方法遍历路径数	On-the-Fly 测试方法遍历路径数
S1	TP1	3	1
	TP2		1
S2	TP1	92	21
	TP2		28
S3	TP1	3 564	658
	TP2		798

从表 8 中可以看出,与传统测试方法相比,on-the-fly 测试方法可以明显减小待搜索的状态空间的规模.此外,通过适配器驱动被测系统的执行可以实现测试例的生成和执行同时进行.而文献[9–13]中提到的方法虽然也是基于 CPN 进行测试生成的,但这几种方法无论选择哪种测试覆盖标准,都需要先生成所有的测试序列,再进行测试执行,在这一点上与传统测试方法是一致的,即测试生成过程与测试执行过程是分开的.如果被测系统比较复杂并且具有动态性,则生成的测试序列可能并不能反映系统实际运行时执行的路径,因此造成用于实际执行的测试工作效率低.而本文方法中测试生成和测试执行同时进行,能够在生成测试例的同时运行被测系统,得到的测试例直接反映了系统运行时的行为过程,提高了测试效率.

5 结 论

在 on-the-fly 测试方法中,不需要遍历整个状态空间,而是根据需要搜索部分状态空间.此外,测试例的生成和执行是同时进行的,可以减少测试例冗余,提高测试效率.CPN 是一种常用的形式化建模方法,可以直观地描述系统并发、同步或异步的动态行为,非常适合对大型复杂的软件系统进行建模和仿真.本文通过 on-the-fly 测试方法与 CPN 形式化建模方法的结合使用,不但充分利用 CPN 模型的优点,并使用 CPN Tools 工具实现对模型的建立、正确性验证和状态空间分析等功能.而且,引入 on-the-fly 测试方法后,通过运行同步乘积算法实现在系统规约扩展可达图中选择含有测试目的的部分可达图,因而避免了测试生成时对整个状态空间的穷举遍历,可以减少时间和内存的使用,进而提高测试例生成和执行的效率.最后,以学籍管理系统为例,说明本文方法的可行性和有效性.

自动化执行测试一直是测试追求的目标,目前在本方法的实现中,被测系统的建模过程需要采用人工方式,

文中 on-the-fly 测试方法与 IUT 实际响应过程的结合部分虽然已可通过适配器实现,但与具体被测系统相关的部分还需要测试人员编写.而且这两个部分由于和被测密切相关,采用统一的方式实现非常困难,因此在未来工作中将对这两部分作进一步的分析研究,以逐步实现 CPN 与 on-the-fly 相结合过程的全自动化.

References:

[1] CPN Tools (version 4.0). http://cpntools.org/

[2] Wu LJ, Su KL, Chen QL, Yang ZH. Algorithm research on “On the Fly” model checking temporal logics of knowledge in multi-agent systems. Journal of Computer Research and Development, 2006,43(8):1417−1424 (in Chinese with English abstract).

[3] Petri CA. Kommunikation mit automaten [Ph.D. Thesis]. Bonn: Institut für Instrumentelle Mathematik, 1962.

[4] Murata T. Petri nets: Properties, analysis and applications. Proc. of the IEEE, 1989,77(4):541−580. [doi: 10.1109/5.24143]

[5] Jensen K. Coloured Petri nets: Basic concepts, analysis methods and practical use. In: EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1992. 65−87.

[6] Jensen K. An introduction to the theoretical aspects of colored Petri nets. Lecture Notes in Computer Science, 1994,803:230−272. [doi: 10.1007/3-540-58043-3_21]

[7] Jense K. Coloured Petri nets: A high level language for system design and analysis. In: Advances in Petri Nets 1990. Berlin, Heidelberg: Springer-Verlag, 1989. 342−416. [doi: 10.1007/3-540-53863-1_31]

[8] Jensen K, Kristensen LM. Coloured Petri nets: Modeling and Validation of Concurrent Systems. Berlin, Heidelberg: Springer-Verlag, 2009. [doi: 10.1007/b95112]

[9] Dong LL, Li H, He N, Xing Y. Testing OpenFlow interaction property based on hierarchy CPN. In: Proc. of the 21st IEEE Int’l Conf. on Network Protocols (2013ICNP). 2013. 1−2. [doi: 10.1109/ICNP.2013.6733620]

[10] Li H, Yue JY, Pang B, He N. CPN modeling and testing of system call for Minix3 access control. Journal of Chinese Computer Systems, 2013,34(12):2828−2832 (in Chinese with English abstract).

[11] Sun T. Research on testing method for parallel software based on colored Petri nets [Ph.D. Thesis]. Hohhot: Inner Mongolia University, 2012 (in Chinese with English abstract).

[12] Li F. The research on performance parameters in modeling for Web application [Ph.D. Thesis]. China University of Petroleum, 2011 (in Chinese with English abstract).

[13] Su M. Verification and testing method of the sematic Web service composition based on colored Petri net [Ph.D. Thesis]. Hohhot: Inner Mongolia Agricultural University, 2014 (in Chinese with English abstract).

[14] Fernandez JC, Mounier L. Verifying bisimulations “On the Fly”. In: Proc. of the IFIP TC6/WG6.1 the 3rd Int’l Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols: Formal Description Techniques. Amsterdam: North-Holland Publishing Co., 1990,90:95−110.

[15] Fernandez JC, Garavel H, Kerbrat A, Mounier L, Mateescu R, Sighireanu M. CADP: A protocol validation and verification toolbox. In: Proc. of the 8th Int’l Conf. on Computer Aided Verification. Springer-Verlag, 1996. 437−440. [doi: 10.1007/3-540-61474-5_97]

[16] Jard C, Jéron T. TGV: Theory, principles and algorithms. Int’l Journal on Software Tools for Technology Transfer, 2005,7(4): 297−315. [doi: 10.1007/s10009-004-0153-x]

[17] Mikucionis M, Larsen KG, Nielsen B. Online On-the-fly testing of real-time systems. In: Proc. of the BRICS. 2003. https://www.researchgate.net/publication/250285411_Online_On-the-Fly_Testing_of_Real-time_Systems

[18] Mikucionis M, Sasnauskaite E. On-the-Fly testing using UPPAAL [Ph.D. Thesis]. The Kingdom of Denmark: Aalborg University, Department of Computer Science, 2003.

[19] Veanes M, Campbell C, Schulte W, Kohli P. On-the-Fly testing of reactive systems. Microsoft Research, 2005. https://www.researchgate.net/publication/228910668_On-the-fly_testing_of_reactive_systems

[20] Frantzen L, Huerta MDLN, Kiss ZG, Wallet T. On-the-Fly model-based testing of Web services with jambition. In: Web Services and Formal Methods. Berlin, Heidelberg: Springer-Verlag, 2009. 143−157. [doi: 10.1007/978-3-642-01364-5_9]

[21] Peled D. Ten years of partial order reduction. In: Computer Aided Verification. Berlin, Heidelberg: Springer-Verlag, 1998. 17−28. [doi: 10.1007/BFb0028727]

[22] Vries RGD, Tretmans J. On-the-Fly conformance testing using SPIN. Int’l Journal on Software Tools for Technology Transfer, 2000,2(4):382−393. [doi: 10.1007/s100090050044]

[23] Fernandez JC, Jard C, Jéron T, Viho C. Using on-the-fly verification techniques for the generation of test suites. In: Computer Aided Verification. Berlin, Heidelberg: Springer-Verlag, 1996. 348–359. [doi: 10.1007/3-540-61474-5_82]

[24] Li L, Qian Z, He T. An approach to testing Web applications on-the-fly. In: Proc. of the Int'l Conf. on Management of e-Commerce and e-Government (ICMECG 2009). IEEE, 2009. 428–431. [doi: 10.1109/ICMeCG.2009.128]

[25] Ernits J, Roo R, Jacky J, Veanes M. Model-Based testing of Web applications using NModel. In: Testing of Software and Communication Systems. Berlin, Heidelberg: Springer-Verlag, 2009. 211–216. [doi: 10.1007/978-3-642-05031-2_14]

[26] Weelden AV, Oostdijk M, Frantzen L, Koopman P, Tretmans J. On-the-Fly formal testing of a smart card Applet. In: Security and Privacy in the Age of Ubiquitous Computing. Springer US, 2005. 565–576. [doi: 10.1007/0-387-25660-1_37]

[27] Jia GP, Zheng GL. Classification of specification method based on software engineering. Software, 1995,7:4–10 (in Chinese with English abstract).

[28] Tretmans J. Testing techniques. In: Formal Methods & Tools Group. Faculty of Computer Science. The Netherlands: University of Twente, 2002. https://www.researchgate.net/publication/266038408_Testing_Techniques

附中文参考文献:

[2] 吴立军,苏开乐,陈清亮,杨志华.多主体系统时态认知规范的"On the Fly"模型检测算法研究.计算机研究与发展,2006,43(8):1417–1424.

[10] 李华,岳婧媛,庞滨,贺楠.Minix3 访问控制的系统调用的 CPN 建模与测试.小型微型计算机,2013,34(12):2828–2832.

[11] 孙涛.基于 CP-nets 模型的并行软件测试方法研究[博士学位论文].呼和浩特:内蒙古大学,2012.

[12] 李峰.Web 应用软件建模过程中性能参数研究[博士学位论文].北京:中国石油大学,2011.

[13] 苏萌.基于有色 Petri 网的语义 Web 服务组合的验证与测试方法[博士学位论文].呼和浩特:内蒙古农业大学,2014.

[27] 贾国平,郑国梁.基于软件工程的规约方法分类.软件,1995,7:4–10.

张玉荣(1988－),女,山东菏泽人,硕士,主要研究领域为分布式计算,测试工程.

王显荣(1964－),男,副教授,主要研究领域为软件建模,测试工程.

李华(1964－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布式计算,测试工程.

阮宏玮(1981－),男,讲师,CCF 专业会员,主要研究领域为软件定义网络,软件测试.

邢熠(1971－),男,博士,讲师,主要研究领域为形式化验证与测试.

张素梅(1989－),女,硕士,主要研究领域为分布式计算,测试工程.