具体反例生成与图形化显示系统①

信贤卫 [1,2]

[1](中国科学院软件研究所, 北京 100190)
[2](中国科学院大学, 北京 100190)

摘 要: 模型检测是用来验证系统模型是否满足所期望性质的一种形式化方法, 模型检测相对于其它的模型检验方法有两个显著的特点, 一个是它对模型进行检测的过程是自动化的, 另一个是当系统不满足所验证的性质时, 它会给出一条反例路径, 这条反例路径可以为系统修正提供帮助. 本文研究的重点就是如何使这条反例路径的生成在高效的同时其反例信息又直观易懂, 为系统修正带来更方便快捷的帮助. 本文中实现了具体反例生成与图形化显示系统(简称 CCGS), 它能快速生成离散语义下具体反例并图形化显示时间自动机沿着该具体反例的运行过程. 实验结果表明 CCGS 能够快速生成具体反例路径信息, 并且能够图形化显示具体反例信息, 为系统修正提供更直观的信息, 提高系统的正确性和安全性.

关键词: 时间自动机; 模型检测; LTL 性质; 反例生成; 模拟器

CCGS System

XIN Xian-Wei[1,2]

[1](Institute of Software, Chinese Academy of Science, Beijing 100190, China)
[2](University of Chinese Academy of Science, Beijing 100190, China)

Abstract: Model checking is a formal method to verify the system satisfies an expected property or not. Trere are two significant advantages of it, one is that it is fully automatic and the other is that if the system doesn't satisfy the checked property, it will generate an counterexamples which can help to fix errors in the system. The main purpose of this paper is to generate this counterexamples efficiently and intuitively. In order to generate the counterexamples efficiently and graphically display the operating processes of the system running alongside the concrete counterexamples, a system CCGS has been developed. Experimental results have shown that CCGS delivers an expected performance, and can help to improve the correctness and safety of the checked systems.

Key words: timed automata; model checking; LTL properties; counterexample generation; simulator

实时系统[1]是能及时响应外部发生的事件, 并以足够快的速度完成对事件处理的计算机应用系统, 譬如支付系统, 导弹控制系统, 火车票售票系统等. 实时系统要求对外部的响应必须在特定时限内完成, 由于涉及并发、不确定性以及时间约束等因素, 实时系统的正确性很难把握. 为了保证实时系统的正确性和安全性, 更改系统设计中可能存在的缺陷, 利用模型检测[2]技术和工具对其进行分析检测是十分有必要的.

模型检测是验证系统模型是否满足所期望性质的一种形式化方法, 可以用来验证各种各样的软件、硬件系统. 模型检测在对系统模型进行检测的过程中用户只需输入特定的系统模型以及所验证的性质描述, 其验证过程是自动化的, 并且如果模型检测的结果是系统不满足所验证的性质, 则会产生一条反例, 在该反例路径上, 系统不满足所验证的性质. 正是这两个显著优点使得模型检测的方法受到越来越多的青睐. 而在实时系统的模型检测中, 广泛使用时间自动机来

① 基金项目:国家科技重大专项(2012ZX01039-004)
收稿时间:2013-04-12;收到修改稿时间:2013-04-22

对系统建模, LTL[3]公式也是一种描述实时系统性质非常常用的规范语言, 本文研究的方向就是关于 LTL 性质的时间自动机模型检测中如何快速生成具体反例, 为系统修正带来更直观高效的帮助.

时间自动机中使用的时间域主要有两种: 一种使用非负整数, 另一种使用非负实数. 在本文中我们称前者对应的时间自动机为离散语义下的时间自动机, 后者对应的时间自动机为连续语义下的时间自动机. 离散语义下时间自动机的模型检测生成的反例路径对应时间自动机的一次运行, 直观且易于理解, 但是由于在离散模型检测中, 状态空间(指系统的全部可能状态的集合)的展开过程会产生大量的后继状态(指系统在某个状态下可以迁移到的下一个状态), 效率较低且能解决问题的规模较小, 所以为了提高模型检测的效率, 时间自动机的模型检测后来主要采用了在连续语义下的基于 zone[4]的符号化模型检测技术, 使得模型检测能用于较大规模的实时系统的验证. 由于 zone 是一些由时钟约束表示的时间状态的集合, 在这种符号化技术下得到的反例路径是关于 zone 的一个序列, 相较于离散反例路径失去了直观性, 在此种反例路径的指导下完成系统的修正往往比较麻烦, 因此需要一种能综合基于 zone 的符号化模型检测过程的高效性以及离散模型检测具体反例的直观性的工具, 使得反例路径能快速生成并且能提供直观易懂的信息便于系统修正.

闭时间自动机[5]是指时钟约束只含有 x≤c 或者 x≥c(x∈X, c∈N*)的形式的时间自动机. 类似于论文[5]中的证明, 我们可以证明每一个闭时间自动机若在连续语义下存在关于某个 LTL 公式的反例, 则在离散语义下存在关于此公式的一个相应的反例. 利用此结果我们实现了以 CTAV[6,7]反例信息为指导, 快速生成闭时间自动机离散语义下具体反例并能图形化显示时间自动机沿着具体反例运行过程的系统 CCGS(concrete counterexamples generation and simulation), CCGS 主要包含快速生成具体反例的 DCFG(discrete counterexamples fast generation)部分与图形化显示具体反例的 simulator 部分. 实验结果表明, CCGS 提高了离散语义下闭时间自动机具体反例生成的速度, 充分利用了具体反例路径的直观性, 能为系统修正提供快捷有效、更加直观的反例信息, 帮助提高系统的正确性及安全性.

1　基本概念

1.1 时间自动机, 闭时间自动机

时间自动机是由 Alur 和 Dill 提出的一种自动机模型, 它是对一般的有限状态自动机的扩充, 可以用来对系统建模. 如图 1 所示即为一个简单的时间自动机.

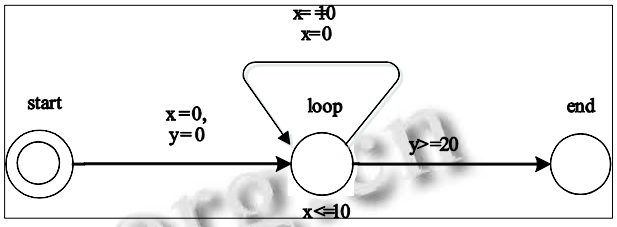

图 1　一个简单的时间自动机

此时间自动机有三个节点, 分别为 start、loop、和 end, start 是初始节点. x, y 是时钟变量, 用于描述时间约束, x 时钟用于保持 loop 循环, y 时钟用于控制 loop 循环结束最小的时间, 自动机到达 loop 循环后至少经过 20 个单位时间后才能迁移到 end 节点. 自动机一开始处于节点 start, 此时 x, y 的值都是 0, 即系统的初始状态为(start,0,0). 在节点 start 处可以于任何时刻发生从 start 到 loop 的离散迁移(延迟迁移以外的所有迁移), 迁移发生时, 由于 start 到 loop 边上有赋值语句“x=0, y=0”, 所以时钟 x, y 都将被重置为 0, 此时系统由初始状态到达下一个状态(loop,0,0). loop 节点的不变式约束“x<=10”表示它可以停留一段时间, 只要时钟 x 不超过 10, 譬如系统可以停留在此状态 5 个单位时间, 也就是经过延迟迁移(延迟迁移中要求系统节点状态不变, 同时所有时钟增加相同的延迟)到达下一个状态(loop,5,5). 在 loop 节点下, 当 x==10 时可以发生从 loop 到 loop 的离散迁移, 同时将 x 重置为 0, 此时系统到达下一个状态(loop,0,10); 当 y>=20 时可以发生从 loop 到 end 的迁移, 譬如在 y=20 时发生状态迁移, 则系统到达下一个状态(end,10,20). 节点中的时间约束(Invariant)限制了可以停留在该节点的时间, 如图 1 中 loop 节点的“x<=10”. 有向边上标有允许迁移发生的约束条件(Guard), 如图 1 中节点 loop 到节点 end 的迁移上的“y>=20”. 在发生迁移时时钟可能被重置(update), 如节点 start 到节点 loop 时的“x=0”, “y=0”.

时间自动机显式的状态是(l,u), $l\in L$ 是节点, 如图 1 中的 start, loop, end. u 是从时钟集合 X 到时间域 T 的一个映射, 称为时钟赋值, 如图 1 中对 x, y 时钟的赋值

“x=0”, “y=0”等. 若 u 中的时钟赋值可以取非负实数, 则该时间自动机是连续语义下的, 若 u 中的时钟赋值只能取非负整数, 则该时间自动机是离散语义下的时间自动机.

闭时间自动机是指时间域为非负整数, 时间约束 Guard 及 Invariant 的表达式中只含有“≤”或者“>=”的时间自动机.

1.2 zone 以及 zone 的表示

连续语义下时间自动机模型检测主要采用基于 zone 的符号化方法, CTAV 是中科院软件所实现的连续语义下基于 zone 的符号化模型检测工具.

zone 是满足扩展的时间约束的时钟赋值的集合, 如满足扩展的时间约束“x≥3∧0≤y≤2∧y≤x-3”的时钟赋值的集合, 描述的是关于时钟 x, y 的一个时钟区域, 相较于离散模型检测中使用的是离散的整数值, 连续语义下的符号化模型检测可以利用 zone 实现状态的聚合, 减少模型检测中生成的状态空间, 因此有着更好的效率.

DBM[4](difference bound matrices)是表示 zone 的数据结构,是符号化语义下反例路径的重要组成部分. 如我们要表示时钟区域 x≥3∧0≤y≤2∧y≤x-3, 可以将其改写为只含‘<’或者‘≤’的表达式: 0-x≤-3 ∧ 0-y≤0∧y-0≤2∧y-x≤-3, 这些表达式可以用关于 0, x, y 三个时钟的 3×3 的矩阵表示, 如表 1 所示即为表示这些表达式的 DBM 矩阵, 其中 INF 表示无穷大.

表 1 DBM 矩阵

0-0 ≤ 0	0-x ≤ -3	0-y ≤ 0
x-0 < INF	x-x ≤ 0	x-y < INF
y-0 ≤ 2	y-x ≤ -3	y-y ≤ 0

该 DBM 矩阵的压缩表示如表 2 所示.

表 2 DBM 矩阵的压缩表示

≤ 0	≤ -3	≤ 0
< INF	≤ 0	< INF
≤ 2	≤ -3	≤ 0

1.3 连续语义符号化反例及离散语义具体反例

CTAV 工具在系统模型不满足所检测的性质时所给出的连续语义下符号化反例是基于 zone(示例中用框标记的内容)组成的状态序列构成的. 如下所述取自 CTAV 对 Fischer 互斥协议[8]的 LTL 性质: !(([]<>P(1).req)->([]<>P(1).cs))进行检测后得到的反例路径上的某个状态:

GF(“p(1).req”|“p(1).cs”)*location: (A,A,A) var[0,0] stateType: 1

```
<=0   <=0   <=0   <=0                \
<INF <=0   <INF <INF                 \
<=INF <INF <=0 <INF                  \
<INF <INF <INF <=0                   \
```

|!“p(1).req” & ! “p(1).cs”

而对应的离散语义下具体反例的状态信息是下面这种表示方式:

GF(“p(1).req” | “p(1).cs”)*location: (A,A,A))
clocks:[0,3,3] var[0,0] stateType:1

两种反例形式的区别主要有两点: 一个是相关时钟的描述, 另一个是迁移的表示. 在基于 zone 的连续语义反例中, zone 的时钟信息表现为两两时钟的差值, 描述的是一个时钟区域, 并且状态之间的迁移没有延迟迁移, 它的符号化反例路径对应的有可能是系统的多次或无穷多次运行过程, 而离散语义下具体反例中的时钟描述对应的都是具体的时钟值, 状态之间的迁移可以是离散迁移也可以是延迟迁移, 因此离散语义下的具体反例对应系统的一次运行过程, 通过图形化显示工具, 可以非常直观准确地提示系统是如何沿着具体反例路径一步一步进入不满足性质的状态的, 从而为系统的修正带来快捷有效的信息, 有助于提高系统的正确性与安全性.

1.4 CTAV 简介

CTAV 是中国科学院软件研究所近年来实现的一个 LTL 性质符号化模型检测工具, 采用时间自动机描述系统模型, 采用 LTL 描述系统性质, 使用基于 zone 的最大上下界 LU(lower/upper bound)抽象技术[7]削减状态空间, 提高了模型检测的效率. 若某个特定的实时系统验证失败, CTAV 工具会产生基于 zone 的符号化反例路径信息. 本文研究的重点就是根据 CTAV 产生的符号化反例信息来生成离散语义下具体反例, 以此来综合 CTAV 的高效性以及具体反例的直观性, CCGS 系统中的 DCFG 部分实现了以 CTAV 符号化反例生成具体反例的功能, simulator 通过图形化模拟具体反例的运行, 增加反例信息的直观性, 为系统修正提供更快捷有效的信息.

2　系统总体设计和分析

为了快速生成系统模型的具体反例及图形化显示这条反例路径在时间自动机上的运行过程，在已有的 CTAV 工具基础上，还需要增加快速生成具体反例的 DCFG 部分及图形化显示具体反例的 simulator 部分，CCGS 系统设计的总体结构图如图 2 所示:

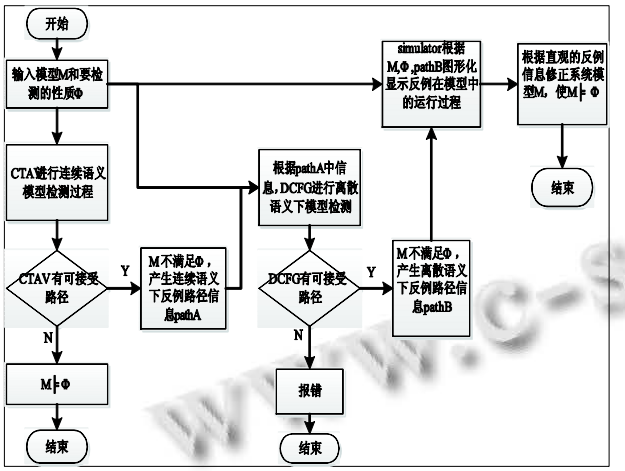

图 2　CCGS 系统体结构图

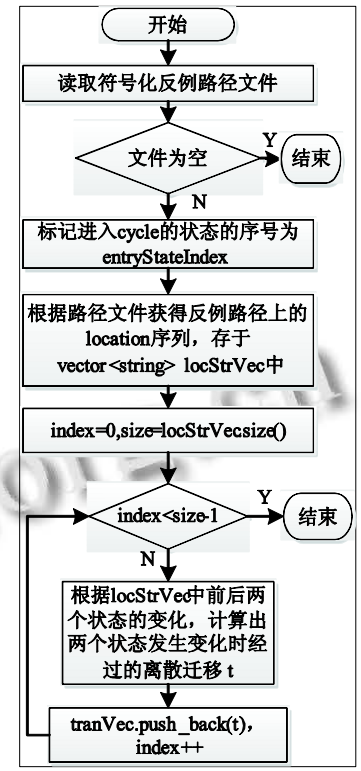

图 3　提取符号化反例路径信息模块流程图

2.1 DCFG 部分

在 DCFG 的实现中，要解决的问题就是从基于 zone 的符号化反例路径得到相应的离散语义下具体反例路径，解决问题的关键就在于以符号化反例路径指导离散语义下具体反例上的离散迁移，而离散语义下具体反例的延迟迁移则根据离散模型检测过程来决定. DCFG 主要包含两个部分，一个是符号化反例路径信息的提取，另一是离散语义下具体反例的生成.

2.1.1 符号化反例路径信息提取

此部分功能的程序流程图如图 3 所示. 其中，tranVec 是 transition 类型的 vector 容器，结构体 transition 在 C++语言中定义，表示闭时间自动机上的迁移，其中有三个 int 类型数据成员: l, p, e. l 表示当前迁移所处的起始 location 信息，p 表示迁移所处的进程号，e 表示当前迁移是 location 的第几个迁移. 所以获得了符号化反例路径上的 transition 序列也就获得了相应的离散迁移序列.

2.1.2 离散语义下具体反例的生成

离散语义下具体反例的生成过程中采用 UPPAAL[4]模型描述语言刻画时间自动机模型，利用 LIBUTAP 库解析 UPPAAL 模型文件，用 BDD[9]表示节点信息和离散化的时钟信息，使用 SPOT[10]库(SPOT 是一个为第三方工具构建模型检测工具提供相应的基础支持的 C++库文件)中的 cou99[10]空性检测算法. 在离散语义下反例路径的生成过程中，先解析 UPPAAL 模型文件和 LTL 性质，然后获得可用于检测的模型表示形式，求出初始状态，再在符号化反例信息的指导下，计算后继状态和性质中原子命题的值来深度优先展开状态空间，在展开状态空间的过程中进行空性检测. 具体反例生成过程的核心和关键就在模型状态的表示以及状态空间的展开.

在 DCFG 中，模型的状态用 taState 类表示，taState 类继承自 SPOT 库的 state 类，在 taState 类里重载了 state 类里的 compare 函数，clone 函数，以及 hash 函数，其中 compare 函数用来判断状态展开过程中生成的状态是否已经访问过，hash 函数用于将 taState 类映射到 BDD，压缩存储空间，使得在状态展开过程中进行状态比较时更方便，clone 函数用于产生一个同样的状态类的对象. 另外，增加了 stateIndex 成员，用于标记此状态在状态展开过程中的序号，以便在状态空间展开中根据基于 zone 的符号化路径得到的迁移序列寻找当前状态沿着反例路径的离散迁移.

在状态空间的展开中, SPOT 库的使用者主要需要提供计算某个状态下的所有后继的算法. 计算某个状态下的所有后继的模块如图 4 所示, 其中 src 是时间自动机当前所处的状态, dst 用来返回时间自动机将要到达的目的状态, tranVec 记录了符号化反例路径的离散迁移信息, entryStateIndex 记录了符号化反例路径进入 cycle 时的状态位置信息, 另外每个状态都有一个 stateIndex 信息标记, 初始化为 0, 状态的 stateIndex 对应此状态在基于 zone 的符号化反例路径中何时产生, 以便计算出此状态沿着符号化反例将经过的离散迁移 t, 有了离散迁移 t 就可以根据当前状态和迁移 t 确定模型的下一个状态, 当前状态的离散迁移的第一个目的状态用 sdsHeadState 表示, 它对应当前状态满足迁移条件的最小的延迟后经过离散迁移 t 到达的目的状态, stateIndex 只在获得 sdsHeadState 时加 1, sdsHeadState 经过延迟得到的后继集合的 stateIndex 保持不变, 这样就可以保证所有新产生的状态的信息标记 stateIndex 与其在基于 zone 的符号化反例路径中的序号保持一致.

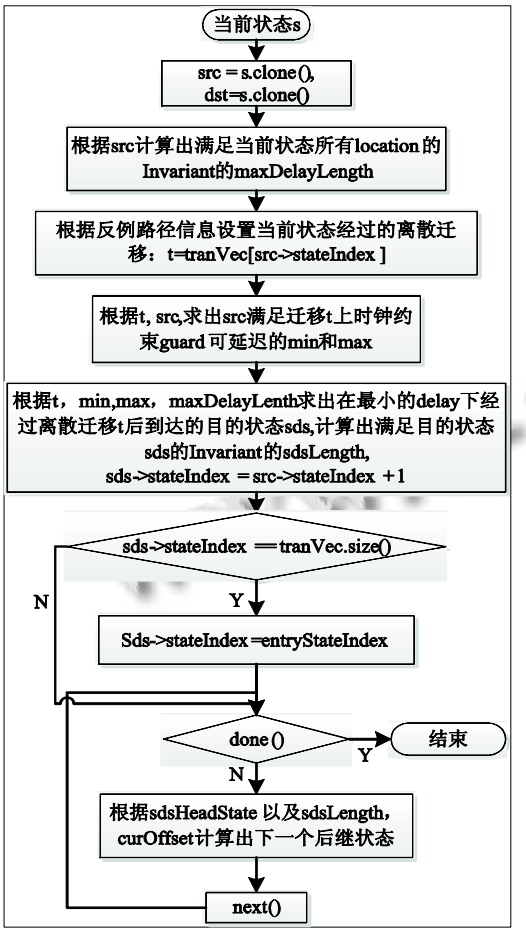

图 4 状态空间的展开模块流程图

2.2 simulator 工具部分

simulator 可以图形化模拟时间自动机沿着具体反例的运行过程, 使得具体反例的信息表示更加直观, 在发现问题时也更容易理解, 能为系统的修正带来不小的帮助.

simulator 用 java 语言实现, 主要含有四个包: figureOfConnection, figureOfNodes, operationBoard, timedAutomataCreated. 其中的 figureOfConnection, figureOfNodes 用来定义时间自动机的迁移和结点信息. figureOfConnection 中的 MyConnection 扩展了 PolylineConnection 并增加了选择改变监听器, 为迁移的表示以及迁移状态改变时需进行的事件处理奠定了基础, 迁移包含一个源结点, 一个目标结点, figureOfNodes 定义结点相关信息. operatingBoard 中的 MainBoardWindowInit 为 simulator 的主进程, 用来定制图形界面用于显示的主界面所使用的画布, 绘制系统的时间自动机模型的图形, 其中的 DataOfEnableTransition 类用来为操作面板上的 EnabledTranstion 文本框提供内容, 其中的 DataOfTrace 类用来为 Transition trace 文本框提供迁移的轨迹相关的内容, 两个文本框都是 Jface 中的 ListViewer 类型的组件, 支持各种类型的事件处理, 因此可以支持生成新的内容触发相关事件, 调用事件处理机制来操作图形界面以图形化显示时间自动机的运行过程. timedAutomataCreated 包中的 XMLReader 类对 xml 文本文件进行读取, 获得系统模型相关的信息, InstanceAnalysis 类用来确定相应的系统模型拥有的实例类型和个数, 也就是图形界面中出现的时间自动机的类型和个数, 而 TimedAutomataFigure 类可以在某画布上根据模板相应的结点和边的信息画出时间自动机模型.

simulator 的图形界面如图 5 所示, 左半部分为操作面板, 用于控制图形显示, 右半部分根据系统模型对应的所有时间自动机来图形化显示, 用绿色来表示当前处于活动状态的节点.

simulator 图形化显示时间自动机沿着具体反例运行的过程大致如下:

① 读入 UPPAAL 模型描述语言支持的 xml 文本文件.

② 调用 InstanceAnalysis 类分析 xml 文本文件内容, 得到系统模型定义的实例进程个数以及各模板相

关信息.

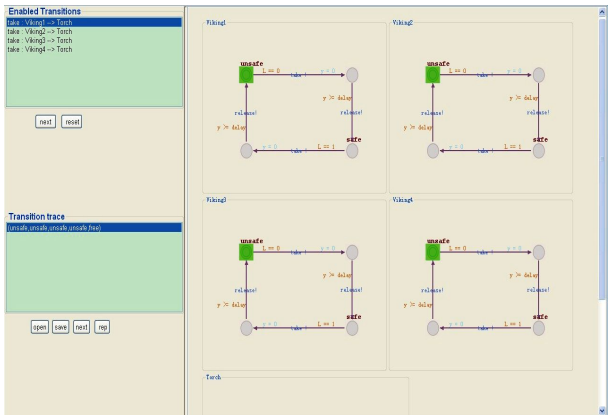

图 5　simulator 工具图形界面

③ 设置好应用程序的主界面，根据第二步得到的进程信息将系统模型各实例进程对应的所有时间自动机图形化显示于图形界面中.

④ 点击操作面板下方的 open 按钮，读入 DCFG 部分产生的具体反例信息，根据具体反例的初始状态设置各时间自动机处于活动状态的节点.

⑤ 通过事件处理机制响应用户点击操作面板上的 next 按钮，此时 TransitionTrace 文本框的内容下移，图形界面的各组件相应文本框内容选择改变事件，调整当前处于活动状态的边及结点，图形化显示时间自动机沿着具体反例运行过程.

3　实验研究

为了验证以连续语义下符号化反例路径信息为指导的离散语义下具体反例路径的生成效率要优于一般的离散模型检测，本文做了以下试验.

实验是 DCTAV，CTAV，CCGS 三个工具之间的比较，其中 DCTAV 是作者实现的离散模型检测工具，相较于一般的离散模型检测工具有着不错的效率，CTAV 是中科院软件所实现的连续语义下符号化模型检测工具，CCGS 为本文中实现的快速生成离散语义下具体反例及图形化显示具体反例的工具，所使用的检测模型是 Fischer 互斥协议模型，所检测的 LTL 性质为：(([]<>“p(i).req”)—>([]<>“p(i).cs”))，其表示的性质是：进程 p(i)每次发出请求后都能进入临界区. 其中 p(i)中的 i 每次取不同的值，表示不同的进程 p(i). 模型的参数取 a=7，b=4，进程数 N=20，试验结果如表 3 所示，‘—’表示时间超过半个小时.

表 3　DCTAV、CTAV、CCGS 的比较

i 值	DCTAV		CTAV		CCGS		
	时间(s)	状态数	时间(s)	状态数	时间(s)	状态数	路径长度
1	0.250	34	0.180	23	0.531	21	7
4	0.313	98	0.301	103	0.578	50	28
7	0.688	3666	0.598	1476	0.641	386	196
9	9.078	50108	2.892	6452	1.094	1538	772
10	37.438	161361	7.675	13300	1.719	3074	1540
11	1094.89	465063	28.622	27316	4.047	6148	3077
12	--	--	93.75	55988	6.078	12290	6140

由表 3 可以看出，有基于 zone 的符号化反例信息指导的离散语义下具体反例生成工具 CCGS 的状态空间要远远小于 DCTAV 工具所产生的状态空间，在时间上空间上都有更高的效率，并且在模型反例路径长度增加，系统规模较大时，CTAV 与 CCGS 的时间之和以及生成的状态空间之和都远低于 DCTAV，表现出了极好的优化效果. 因此 CCGS 有效地利用了符号化模型检测过程的高效性和具体反例的直观性.

CCGS 中 simulator 图形化显示 Fischer 互斥协议模型对性质(([]<>“p(1).req”)—>([]<>“p(1).cs”))检测后产生的具体反例的过程见图 6.

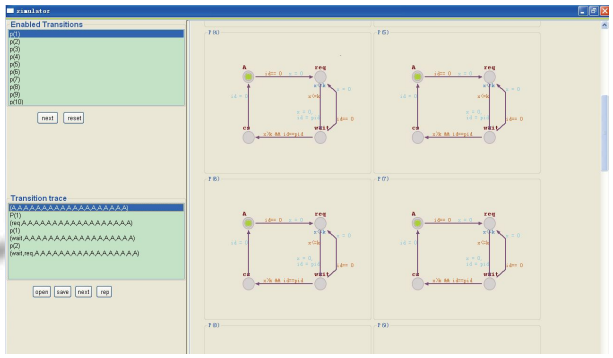

图 6　Fischer 互斥协议图形界面

因为所验证的系统模型有 20 个进程，对应有 20 个时间自动机图形，同时反例路径较长，因此完整显示整个过程几乎不可能. 图 6 取自图形化显示时间自动机沿着具体反例运行过程的某个状态，此时各时间自动机的 A 节点处于活动状态，它表示当前系统处于状态(A,A,A…)，不断点击操作面板下方的 next 按钮，图形界面的时间自动机处于活动状态的节点和边将发生相应的变化变为绿色，由活动状态转为非活动状态的边或者节点变为灰色，以此来图形化显示时间自动

机模型沿着具体反例的运行过程, 这个过程直观易理解, 并且可以以此来研究系统是如何一步步进入不满足所检测性质的状态的, 然后修正系统, 使得系统能满足所需的性质.

4 结语

在本文中, 为了解决离散语义下模型检测过程检测效率较低的问题以及克服连续语义下基于 zone 的符号化反例路径给系统修正带来的麻烦, 实现了以连续语义下基于 zone 的符号化反例为指导, 快速生成闭时间自动机离散语义下具体反例并图形化显示时间自动机沿着具体反例运行过程的工具 CCGS, 有效地利用了符号化模型检测过程的高效性以及具体反例路径信息的直观性及确定性, 使得反例路径的生成在高效的同时其反例信息又直观易懂, 这种方法可以用于对系统模型进行检验的过程中, 为系统修正提供快捷有效、更加直观的反例信息, 帮助提高系统的正确性及安全性.

接下来的主要工作是扩充及完善 CCGS 工具, 增加能验证的 LTL 性质的范围, 增加 simulator 在模拟过程中提示模型相关信息的功能, 使其功能更加全面.

参考文献

1 Alur R, Henzinger TA. Real-time system=discrete system + clock variables. Software Tools for Technology Transfer, 1997, 1(1/2): 86–89.

2 Clarke EM, Grumberg O, Peled DA. Model Checking. The MIT Press, 1999.

3 李广元,唐稚松.带有时钟变量的线性时序逻辑与实时系统验证.软件学报,2002,13(1):33–41.

4 Bengtsson J, Wang Y.Timed automata: semantics, algori- thms and tools. LNCS 3098, Springer-Verlag, 2004: 87–124.

5 Beyer D, Noack A. Efficient verification of timed automata using BDDs. Proc. of the 6th International ERCIM Workshop on Formal Methods for Industrial Critical Systems, 2001. 95–113.

6 魏绪凯.时间自动机关于 LTL 性质的符号化模型检测工具及其改进[博士学位论文].北京:中国科学院软件研究所,2009.

7 Li GY. Checking timed Büchi automata emptiness using LU-abstractions. Proc. of the 7th International Conference on Formal Modeling and Analysis of Timed Systems, 2009. 228–442.

8 Yu F, Wang BY. Sat-based model checking for region automata. International Journal of Foundations of Computer Science, 2006, 17(4): 775–796.

9 Duret-Lutz A, Poitrenatud D. Spot: An extensive model checking library using transition-based generalized Büchi automata. Volendam, Netherlands. IEEE Computer Society Process. 2004. 76–83.

(上接第 50 页)

参考文献

1 Fowler M, Continuous Integration, www.martinfowler.com/articles/continuousIntegration.html, May2006.

2 Martin RC.敏捷软件开发:原则、模式与实践.邓辉译.北京:清华大学出版社,2003:3–16,23–56.

3 成奋华;金敏.基于敏捷过程的IT项目范围管理的研究与应用.计算机技术与发展,2010(10):232–236.

4 石军霞.高校学生满意度调查研究.苏州:苏州大学,2008.

5 王炜,刘西涛.基于学生满意度视角的高校教学质量的困境与对策.继续教育研究,2011(10):157–159.

6 王延玲.如何有效地开展网上调查.市场研究,2005,(4):59–60.

7 刘永亮,闫丽丽.基于web的跨平台信息系统敏捷开发架构.电脑编程技巧与维护,2013(2):52–55.

8 Kevin V, Jansen S,Brinkkemper S.The agile requirements refinery: Applying SCRUM principles to soft- ware product management. Information and Software Tech- nology, 2011, 53(1): 58.

9 李珂.高校大学生求学满意度调查.煤炭高等教育,2007,25(3):89–91.

10 李俊杰.maven在企业java软件产品中的应用.电脑知识与技术,2011,07(7):120–113.

11 张宇,王映辉,张翔南.基于Spring的MVC框架设计与实现.计算机工程,2010,36(4):59–62.

12 戚琦,廖建新,王纯,武家春.基于敏捷方法的轻量级 J2EE 架构的应用.计算机系统应用,2007(2):53–56.

13 刘慧,路正南.基于 PLS 路径建模技术的中国高等教育学生满意度测评研究.高教探索,2012(2):30–36.

14 谢治国,胡化凯.运用 BCG 矩阵对高新技术产业的划分定位研究.科学学与科学技术管理.2004,25(8):140–144.