多Agent交互策略模型检测方法

张 涛[1,2]，谢 红[2]，黄少滨[1]

(1. 东北农业大学电气与信息学院 哈尔滨 150030；2. 哈尔滨工程大学信息与通信工程学院 哈尔滨 150001)

【摘要】提出一种基于模型检测的多Agent交互策略验证方法，首先通过责任政策语言建模多Agent的交互策略，基于责任政策语言的操作语义将政策模型转换为模型检测器NuSMV的输入，利用时态逻辑声明表征策略冲突的系统性质，然后利用模型检测器NuSMV自动验证政策模型对性质的可满足性，并根据模型检测器产生的反例分析交互策略中的各种错误。该方法可提高交互策略的验证效率，确保多Agent系统设计的正确性。

关 键 词 形式化方法； 模型检测； 多Agent系统； NuSMV； 政策建模

中图分类号 TP399 **文献标志码** A doi:10.3969/j.issn.1001-0548.2016.05.016

The Method of Model Checking Policy of Multi-Agent Interaction

ZHANG Tao[1,2], XIE Hong[2], and HUANG Shao-bin[1]

(1. School of Electrical and Information, Northeast Agricultural University Harbin 150030;
2. College of Information and Communication Engineering, Harbin Engineering University Harbin 150001)

Abstract A verification method of multi-agent interaction policy is proposed based on model checking. The model of system is specified with obligation policy language and it is converted to the input model of model checker NuSMV based on its operational semantics, the properties of system depending on different types of policy conflicts are represented with temporal logic, and the violations of properties are detected by using NuSMV model checker, which can provide the counterexample and trace it back to the errors in interaction policy. The result shows that the method can improve the efficiency of verifying interaction policy, and it ensures the correctness of the design of Multi-agent systems.

Key words formal method; model checking; multi-agent system; NuSMV; policy model

多Agent系统中，责任一般是Agent为了满足某些要求而执行的动作[1]，如“社会保障管理条例规定企业在成立之日起的20个工作日内，需持营业执照到当地社会保险经办机构办理社会保险登记”。责任或是Agent被要求保持的某种状态[2]，如“养老保险缴费人员必须处于参保状态”。责任政策约束其所属领域内Agent间的交互行为，可被视为系统设计中的高层“需求规格”或“交互策略”[3]。在多Agent系统中，建模和验证多Agent间的交互策略有助于正确设计Agent间的交互行为，提高系统的正确性和可靠性[4]。文献[5]在计算机科学领域提出研究规律政策的观点，文献[6]开创了关于分布式系统中的政策研究方向，并提出了“规律控制的交互(law-governed interaction, LGI)”，由此激发了该领域的深远研究与大量实践工作。尽管LGI模型曾被成功用于多Agent系统的各种研究与应用领域，但该模型使用低层抽象信息描述系统中的交互行为，使其逐渐不能满足复杂系统的设计需求。为此，多种责任政策的形式化语言和方法被相继提出，在开发人工制度[7]、验证程序通信[8]、建模Agent交互[9]等研究领域得到了广泛的应用，这些方法可被分为三类：1) 设计政策执行语言，如文献[10]。这类语言可对政策进行简单的描述和解释，但由于缺乏形式语义，所以不能对其进行形式分析或性质验证。2) 设计政策分析语言，如文献[11-12]。它们允许对政策进行形式化定义与分析，但是这类方法没有定义政策语言执行与管理的动态操作语义，不能作为自动验证技术的形式化基础。3) 文献[13]直接给出了策略模型的验证方法，但是没有研究策略模型的形式语义，使策略模型不易被理解和解释，无法建模更为复杂的交互

收稿日期：2015－01－12；修回日期：2016－04－20

基金项目：国家科技支撑计划(2012BAH08B02)；中央高校基本科研业务费专项基金(HEUCF100603, HEUCF041204)；黑龙江省博士后资助项目(3236310148)

作者简介：张涛(1981－)，男，博士，主要从事分布式计算与仿真、形式化方法等方面的研究.

策略。针对上述问题，本文提出一种多Agent交互策略模型检测方法，旨在弥补上述两类方法的不足。该方法中的责任政策框架语言既可以对政策进行描述和解释，又允许对政策进行形式化定义与分析，最终支持使用自动化工具执行验证分析。

1　责任政策语言

责任政策描述了多Agent之间的交互策略，即Agent被允许或禁止保持某种状态或执行某些活动，并关联着一些条件集合，这些条件以环境的形式存在于责任描述中，描述系统状态以及政策的应用规则。因此，在定义责任政策语言前，先定义责任政策环境。与责任相关的环境主要有基于状态的环境和基于事件的环境。基于状态的环境声明政策中责任的状态环境，基于事件的环境声明责任被激活、失效、违反或履行的时刻。基于状态的责任环境表达式为：$Cs(\mathrm{Ag}_1,\mathrm{Ag}_2,\varphi)\leftarrow p_1,p_2,\cdots,p_n$，其形式语义定义为当命题公式 $p_1,p_2,\cdots,p_n$ 为真，Ag_1 向 Ag_2 承诺 φ 时所在的环境 Cs 有效。基于事件的环境则描述了基于状态的环境开始有效和终止有效的时刻，其被声明为 start(Cs) 和 end(Cs) 。责任政策的环境表达式如下：

$$Cs ::= \mathrm{true}\,|\,\mathrm{false}\,|\,c\,|\,Cs\wedge Cs\,|\,Cs\vee Cs\,|\,\neg Cs\,|\,[Cs_1,Cs_2]$$

$$Ce ::= \mathrm{start}(Cs)\,|\,\mathrm{end}(Cs)\,|\,Cs\wedge Ce\,|\,Ce\wedge Cs$$

$$C ::= Cs\,|\,Ce$$

式中，C 表示责任政策的环境表达式；Cs 是基于状态的环境表达式；Ce 是基于事件的环境表达式；c 是用户定义的基于状态的环境标识符；$[Cs_1,Cs_2]$ 表示一个区间，其声明基于状态的环境在 Cs_1 为真时开始有效，在 Cs_2 为真时终止有效。

定义 1　责任(Obligation)。责任是一个六元组 Obligation=(N,debtor,creditor,φ,Ca,Cv)，其中，N 是责任标识符，debtor 是责任的发起方，creditor 是责任的接收方，φ 是声明责任内容的逻辑公式，Ca 是责任的激活环境，Cv 是责任的违反环境。责任被声明后，debtor 将向 creditor 传递由 φ 表示的事实或需要执行的动作，Ca 和 Cv 分别声明了责任被激活和违反的环境。

定义 2　责任政策语言(obligation policy language, OPL)。$\mathrm{OPL}=(A,O,C,\mathrm{Action},R,\mathrm{sta})$ 是一个六元组，其中，A 是政策中所有Agent的集合，$\forall Ag_i\in A$，$i\in N$，Ag_i 既可以是责任的 debtor 也可以是 creditor；O是政策中所有责任的集合；C是政策环境的集合；$\mathrm{Action}=\mathrm{Act}_o\cup\mathrm{Act}_c$ 是政策动作的集合，$\mathrm{Act}_o=\{\mathrm{create},\mathrm{active},\mathrm{fulfil},\mathrm{violate},\mathrm{deactivate}\}$ 是责任操作动作的集合，Act_c 是责任内容中动作的集合，即责任内容中 debtor 承诺执行的动作；R 是责任政策执行需满足的时序性质集合。$\mathrm{sta}:O\rightarrow A$ 是一个责任分配函数，给出每个责任所归属的Agent。

2　责任政策语言的表达能力

责任政策语言通过责任状态模型描述交互策略的动态演化，基于操作语义可将交互策略的OPL模型转换为NuSMV的输入模型。

2.1　责任状态模型

OPL模型中主要包含连续型责任和非连续型责任。连续型责任声明系统中有维持事物状态需求的责任，在系统中这种责任从环境起始直至环境结束都要求其保持激活状态，如考虑如下责任 Obligation=(n_0,Ag_1,Ag_2,φ_3,insuranced, below_limit_payment)，责任 n_0 声明参加养老保险的职工 Ag_1 向管理机构 Ag_2 缴费不能低于缴费下限。其中，Ag_1 是养老保险参保人员，Ag_2 是养老保险管理机构，φ_3= pay_fees(Ag_1,Ag_2) 声明 Ag_1 向 Ag_2 执行缴费行为，激活环境声明责任在 insuranced 为真的情况下有效，违反环境低于费款下限 below_ limit_ fees 被定义为 below_limit_payment (Ag_1,_,_)$\leftarrow$ limit_payment (Ag_1,x_1), current_payment(Ag_1,x_2),$x_2\prec x_1$，其声明参保人员缴纳保险费低于政策规定的缴费下限的情况。

非连续型责任是被激活后可能发生失效的责任，其定义条件可表示为 end(Ca)$\neq$ false 。

责任的状态模型描述责任政策的动态演化过程，根据责任类型的定义，责任状态模型被分为两种：连续型责任状态模型和非连续型责任状态模型，分别如图1和图2所示。

在责任状态模型中，责任 Obligation=(N,debtor,creditor,φ,Ca,Cv) 的状态主要包括：

1) 初始状态 •，该状态表示责任未被创建。

2) 非活跃状态 inactive，其可被形式化声明为 inactive(obligation)$\leftarrow$(create(debtor,obligation)$\wedge$($\neg$start(Ca))$\vee$end(Ca)))。责任处于非活跃状态主要有两种可能，一种是责任创建后未被要求履行，即 create(debtor,obligation)$\wedge\neg$start(Ca)；另一种是责任在创建后曾被激活过，但现已不被要求履行，即环境 end(Ca) 有效，此时责任同样处于非活跃状态。

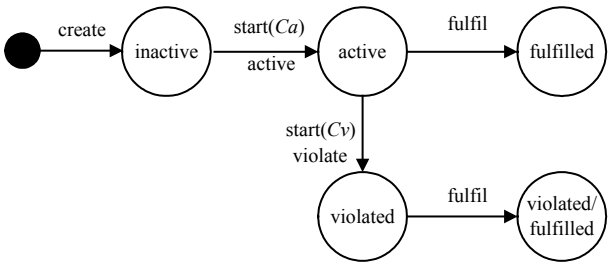

图1 连续型责任的状态模型

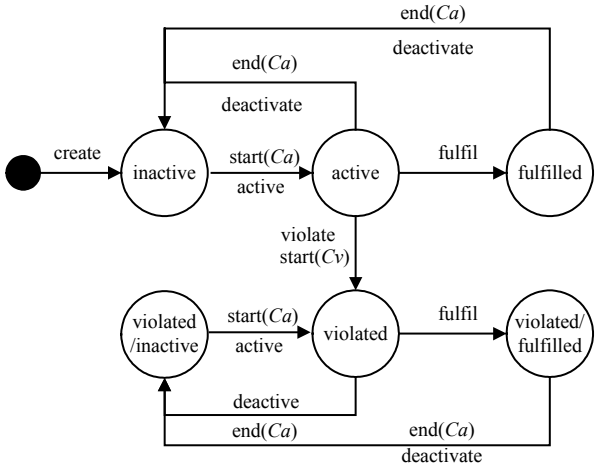

图2 非连续型责任的状态模型

3) 活跃状态 active，其可被形式化声明为active(obligation) ← active(debtor,obligation) ∧ start(*Ca*)。当责任被要求履行时，即环境 start(*Ca*) 有效，由 debtor 激活责任使其处于活跃状态。

4) 履行状态 fulfilled，其可被形式化声明为 fulfilled(obligation) ← fulfil(debtor,obligation) ∧ start(*Ca*)。责任在激活环境下被 debtor 履行后，责任处于履行状态。

5) 违反状态 violated，其可被形式化声明为 violated(obligation) ← violate(debtor,obligation) ∧ start(*Ca*) ∧ start(*Cv*)。责任在激活环境和违反环境有效的条件下被 debtor 违反，则其处于违反状态。

6) 违反/非活跃状态 violated/inactive，violated/inactive(obligation) ← (violated(obligation) ∧end(*Ca*) ∧ deactivate(debtor,obligation)。责任被 debtor 违反后，在激活环境终止有效的条件下，debtor 执行 deactivate 行为，使责任处于违反/非活跃状态。

7) 违反/履行状态 violated/fulfilled，可被声明为 violated/fulfilled(obligation) ← (violated(obligation) ∧ start(*Ca*) ∧ fulfil(debtor,obligation)。责任被违反后，且在激活环境有效的条件下被 debtor 履行，则责任处于违反/履行状态。

责任状态模型中的违反状态被定义为持久状态，即一旦责任的违反环境为真且 debtor 执行违反操作，则责任处于违反状态并在后续的状态演化中保持违反状态。

2.2 OPL的操作语义

为了将OPL模型转换为NUSMV的输入模型，本文将OPL的操作语义定义为具有Kripke结构的状态迁移系统。

定义 3 迁移系统(transition system, TS)[14]。迁移系统是一个六元组：$\mathrm{TS}=(S,\mathrm{Act},\rightarrow,I,\mathrm{AP},L)$，其中，$S$是系统状态的有穷集合，Act 是系统动作的有穷集合，$\rightarrow\subseteq S\times\mathrm{Act}\times S$ 表示状态迁移关系集合，$I\subseteq S$ 是系统初始状态的有穷集合，AP 是原子命题的有穷集合，$L:S\rightarrow 2^{\mathrm{AP}}$ 是一个标签函数。

迁移系统中，系统状态由一组取值为真的原子命题集合表示，标签函数可以将一个原子命题集合 $L(s)\in 2^{\mathrm{AP}}$ 关联到任意状态 $s\in S$。对于一个给定的逻辑公式Φ，如果 $L(s)$ 包含的原子命题使公式Φ成立，则称状态s满足公式Φ，即$s\models\Phi$ iff $L(s)\models\Phi$。

定义 4 责任政策语言OPL的操作语义。$\mathrm{OPL}=(A,O,C,\mathrm{Action},R,\mathrm{sta})$ 的操作语义是一个六元组：$\mathrm{TS}_{\mathrm{OPL}}=(S,\mathrm{Act},\rightarrow,I,\mathrm{AP},L)$，其中，$S\subseteq(\mathrm{sta}(o_1)\times\cdots\times\mathrm{sta}(o_n))\times C$ 是系统状态的有穷集合，Act=Action 是动作的有穷集合，$\rightarrow\subseteq S\times\mathrm{Act}\times S$ 表示状态迁移关系集合，$I\subseteq(\bullet_1\times\cdots\times\bullet_n)\times C$ 是系统初始状态的有穷集合，AP 是原子命题的有穷集合，$L:S\rightarrow 2^{\mathrm{AP}}$ 是标签函数。

在状态迁移系统 $\mathrm{TS}_{\mathrm{OPL}}$ 中，系统状态集合 S 中的状态由当前时刻每个责任所处状态和责任政策的当前环境组成。系统初始状态由每个责任的初始状态和系统初始时刻的责任环境组成。$\mathrm{TS}_{\mathrm{OPL}}$ 的动作集合 Act 与 OPL 的动作集合 Action 定义相同，即 $\mathrm{Act}=\mathrm{Act}_o\cup\mathrm{Act}_c$，$\mathrm{Act}_o$ 是责任操作动作的集合，Act_c 是责任内容中所执行动作的集合。在OPL中，Act_c 中动作的发生导致责任政策环境发生变化，而 Act_o 中动作的发生导致责任状态发生变化，所以基于 Act 中的状态、动作执行序列可描述责任政策规则的执行以及责任状态的演变。AP 集合中的元素是政策环境表达式与责任内容公式 φ 中的命题。

3 模型检测交互策略模型

模型检测(model checking)[14] 是一种自动验证有穷状态并发系统的形式化技术。在验证过程中，系统被建模为具有Kripke语义结构的状态迁移系

统，系统规范被声明为模态/时序逻辑公式，当模型满足规范时验证算法给出肯定答案，否则算法会将导致错误结果的事件序列作为反例返回给用户，为系统漏洞定位和改进提供帮助。基于模型检测技术验证多Agent交互策略，首先要将交互策略模型转换为模型检测器的输入模型，本文选用基于Kripke结构作为输入的NuSMV作为模型检测工具验证交互策略。

3.1　转换交互策略模型到NuSMV模型

NuSMV以模型描述程序和时态逻辑规范文本作为输入。若规范成立，则输出为真，否则显示一个轨迹，该轨迹解释规范为假的原因。NuSMV输入模型由一个或多个模块组成，其中一个模块必须被声明为主模块，每个模块都可由三部分组成：变量的声明、变量赋值以及性质声明。模块内部声明变量并对变量赋值，赋值操作通常给出变量的初始值，而变量下一个值是关于变量当前值的表达式。NuSMV建模语言的具体语法可参照文献[14]。

本文基于OPL的操作语义，将其转换为NuSMV的输入模型，转换过程中每个Agent被转换为输入模型的一个独立模块，并在主模块中初始化。各Agent的责任被建模为对应模块中的枚举变量，枚举值为责任的各个状态。各Agent的责任所关联的环境变量被转换为模块参数变量及其内部中的枚举变量和布尔变量，与责任状态相关的责任操作则被转换为状态演化的推理规则，这种迁移关系定义在各模块的ASSIGN部分，责任的内容动作被转换为迁移条件定义在模块的NEXT语句部分。

3.2　被验证性质描述

本文采用计算树逻辑CTL声明系统应满足的关键性质，CTL的语法定义如下：

定义 5　令p为原子命题，通过Backs-Naur范式定义CTL公式ϕ为：

$$\phi ::= \bot \mid \top \mid p \mid (\neg\phi) \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \to \phi \mid \mathrm{AX}\phi \mid \mathrm{EX}\phi \mid \mathrm{AF}\phi \mid \mathrm{EF}\phi \mid \mathrm{AG}\phi \mid \mathrm{EG}\phi \mid \mathrm{A}[\phi \mathrm{U} \phi] \mid \mathrm{E}[\phi \mathrm{U} \phi]$$

式中，路径量词A表示沿着所有路径(无一例外)；E表示沿着至少(存在)一条路径(可能)；时态算子X、F、G、U分别为下一个状态、未来某个状态、所有未来状态和直到。

系统被期望满足的性质主要包括安全性和活性性质，系统的安全性被描述为坏事ϕ一定不会发生，其CTL公式形式为$\mathrm{AG}\neg\phi$，系统的活性被描述为前提ψ成立时好事φ会经常发生，其CTL公式的形式为$\mathrm{AG}(\psi \to \mathrm{AF}\varphi)$。

4　实例研究

4.1　系统建模

本文以《城镇企业职工基本养老保险关系转移接续暂行办法》为例(简称《暂行办法》)，作为多Agent交互策略，创建该策略的OPL模型。《暂行办法》对指导养老保险领域多Agent交互行为建模具有重要意义，限于篇幅，本文不对《暂行办法》的内容一一介绍，仅介绍与验证有关的重要内容，具体如下：

第三条给出了跨省流动参保人员达到基本养老保险待遇领取条件和未达到待遇领取年龄前，基本养老保险关系和个人账户的具体处理办法，特别强调了不得办理退保手续。

第四条规定了如何计算转移基金，增加了按照实际缴费工资的12%转移统筹基金，更好地平衡地方利益。

第六条规定了参保人保险关系不在户籍所在地，则累计缴费年限必须满10年，才可在保险关系所在地享受基本养老保险待遇。

第七条规定了参保人员转移接续后，符合待遇领取条件的按照国发2005年38号文的规定享受基本养老金，确保转移人员的基本养老金计算办法与其他参保人员的一致性。

《暂行办法》的OPL模型中，Agent集合A主要包括：社保经办机构PB和参保人GR。Agent GR的责任主要包括：参加保险n_1，向管理机构登记n_2，缴纳保险费n_3，申请参保关系转移n_4，领取保险待遇n_5，退出保险n_6。Agent PB的责任主要包括：同意转移参保关系n_7，支付保险待遇n_8，同意退保n_9。各责任的形式化定义如下：

Obligation = $(n_1, \mathrm{GR}, \mathrm{PB}, \varphi_1, \mathrm{want\_insured}, \mathrm{over\_age\_limit})$ 声明50周岁以下的参保人GR可以参加城镇基本养老保险，其中 $\varphi_1 = \mathrm{participating(GR, endowment\_insurance)}$ 声明参保人参加养老保险的行为，布尔型的环境 want_insured 声明参保人是否要参保，环境 $\mathrm{over\_age\_limit} \leftarrow \mathrm{age(GR)} > 50$ 声明参保人年龄下限为50周岁。

Obligation = $(n_2, \mathrm{GR}, \mathrm{PB}, \varphi_2, \mathrm{insuranced}, \mathrm{not\_alive})$ 声明参保人GR在社保经办机构PB登记注册，由PB管理GR，其中 $\varphi_2 = \mathrm{register(PB, GR)}$，环境 $\mathrm{insuranced(GR, \_, \_)} \leftarrow \mathrm{participating(GR, endowment\_insurance)}$ 声明GR处于参保状态，简写为 insuranced 。布尔型的环

境 not_alive 声明GR已死亡。

Obligation = $(n_3, \mathrm{GR}, \mathrm{PB}, \varphi_3, \mathrm{insuranced}, \mathrm{retired})$ 声明未退休参保人GR在退休前向PB缴纳养老金，其中 φ_3=pay_fees(GR,PB)，布尔型环境 retired 声明GR是否处于退休状态。

Obligation = $(n_4, \mathrm{GR}, \mathrm{PB}, \varphi_4, \mathrm{insuranced} \wedge \mathrm{want\_transfer}, \mathrm{retired})$ 声明未退休参保人GR向PB申请转移参保关系，其中 φ_4=apply_transfer(GR,PB)，布尔型环境 want_transfer 声明GR是否要转移参保关系。

Obligation = $(n_5, \mathrm{GR}, \mathrm{PB}, \varphi_5, \mathrm{insuranced} \wedge \mathrm{retired} \wedge \mathrm{acc\_payment\_year}, \mathrm{not\_alive})$ 声明活着的已退休参保人达到缴费年限可领取社保机构支付的基本养老保险待遇，其中 φ_5=get_money(GR,PB)，布尔型环境变量 acc_payment_year 声明GR是否达到缴纳养老保险的累计年限。

Obligation = $(n_6, \mathrm{GR}, \mathrm{PB}, \varphi_6, \mathrm{insuranced} \wedge \neg\mathrm{retired}, \mathrm{not\_alive})$ 声明未退休参保人退出养老保险，φ_6=apply_quit(GR,endowment_insurance)。

Obligation = $(n_7, \mathrm{PB}, \mathrm{GR}, \varphi_7, \mathrm{insuranced} \wedge \mathrm{want\_transfer} \wedge \neg\mathrm{retired}, \mathrm{not\_alive})$ 声明PB同意未退休参保人GR转移参保关系，其中 φ_7=agree_transfer(PB,GR)。

Obligation = $(n_8, \mathrm{PB}, \mathrm{GR}, \varphi_8, \mathrm{insuranced} \wedge \mathrm{retired} \wedge \mathrm{achieve\_payment\_year}, \mathrm{not\_alive})$ 声明PB支付已退休参保人GR养老保险待遇，其中 φ_8=pay(PB,GR)，布尔型环境变量 achieve_payment_year 声明参保人的累计缴费年限是否达到10年。

Obligation = $(n_9, \mathrm{PB}, \mathrm{GR}, \varphi_9, \mathrm{insuranced} \wedge \mathrm{achieve\_payment\_year}, \mathrm{retired})$ 声明PB同意撤销未退休但是已达到累计缴费年限的GR的参保关系，其中 φ_9=agree_quit(PB,GR)。

4.2 实验结果及分析

根据OPL模型的操作语义及前文定义的转换关系，与《暂行办法》的OPL模型对应的NUSMV部分代码如图3所示。

该模型被期望满足性质的CTL公式为：AG((g.n_1=fufilled)->AF(g.n_5=fufilled))，其含义为如果Agent g履行参保责任，则其退休一定可以享受养老保险待遇。将该性质输入NuSMV执行验证，检测结果如图4所示。

图中，CTL公式的断言为False，表示《暂行办法》的NuSMV模型不满足该公式，即在《暂行办法》中存在参保人正常参保，退休后却不能享受养老保险待遇的情况，通过模型检测器的反例展示功能可获得验证反例的轨迹，如图5所示。

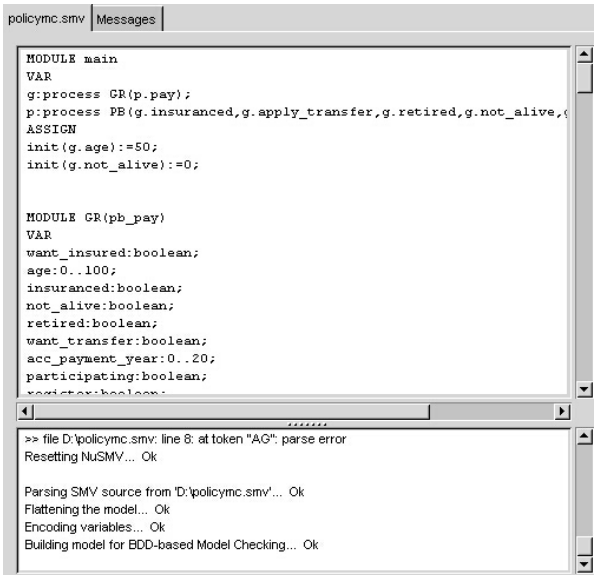

图3 《暂行办法》的部分NUSMV代码型

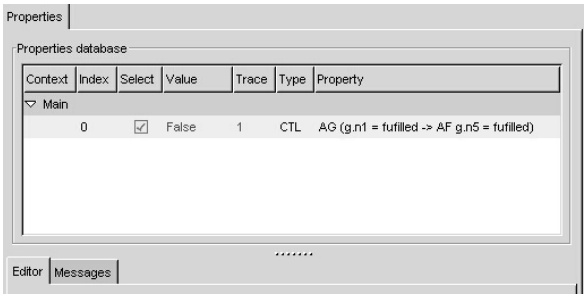

图4 模型检测结果

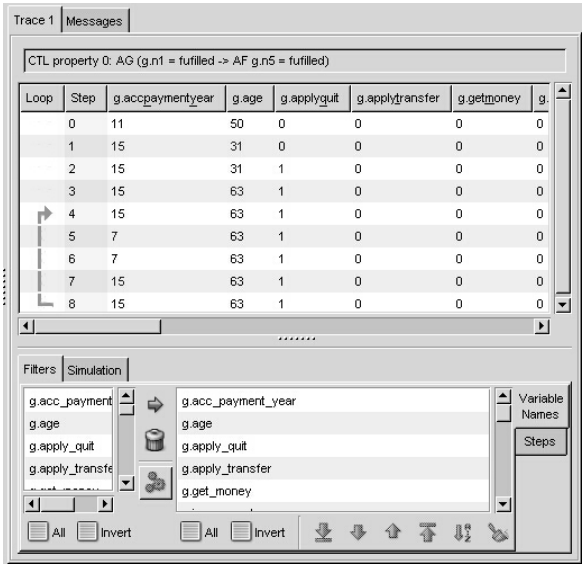

图5 验证反例的轨迹

分析验证反例可发现错误的原因在于《暂行办法》第六条规定参保人累计缴费年限必须满10年才可享受基本养老保险待遇。如果当前年龄在50岁以上的参保人，此前由于种种原因未能积累缴费年限，则不能享受保险待遇。《暂行办法》第三条又规定参

保人员不能提前退保，因此这类参保人员面临着退休后既不能享受待遇又不能提前退保的尴尬境遇。

文献[15]给出了一种基于RAISE建模语言和模型检测技术的领域政策形式化分析方法，该方法首先用RAISE建模语言构建领域政策的形式化描述，再将这种形式化描述转换为模型检测器SPIN的输入模型，同时将被验证性质声明为线性时序逻辑公式，最后使用SPIN自动验证政策模型对被验证性质的可满足性。文献[15]同样以《暂行办法》为研究实例，并验证了政策的RAISE模型对线性时序逻辑公式声明性质的可满足性。本文的研究方法与文献[15]对比，主要有以下不同：1) RAISE建模语言是一种软件工程领域通用的形式化语言，而本文设计的责任政策语言OPL是一种面向领域政策的专用建模语言，OPL相比RAISE建模语言更具备政策建模特质，且易于理解和使用。2) 文献[15]将RAISE政策模型转换为模型检测器SPIN的输入模型。本文将OPL政策转换为模型检测器NuSMV的输入模型，文献[15]只能验证线性时序逻辑公式声明的性质，而本文既能够验证线性时序逻辑公式声明的性质，又能验证计算树逻辑声明的政策性质，所以本文的方法可以验证比文献[15]的方法更为广泛的政策性质。

5 结 束 语

基于模型检测技术，本文提出一种多Agent交互策略形式化验证方法，用于验证多Agent系统中交互行为设计的正确性，该方法既使交互策略模型具备易于理解和使用形式语义，又能够自动验证设计模型的正确性。实验结果表明，该方法可有效地发现并解释策略模型中存在的错误，降低验证过程中的人工参与程度，提高验证效率。

参 考 文 献

[1] LOIZOS M, DAVID P C, PFEFFER A. Specifying and monitoring economic environments using rights and obligations[J]. Autonomous Agents and Multi-Agent Systems, 2010, 20(2): 158-197.

[2] XU Dian-xiang, SANFORD M, LIU Zhao-liang. Testing access control and obligation policies[C]//International Conference on Computing, Networking and Communications. San Diego, CA , USA: IEEE Computer Society, 2013: 540-544

[3] BALDONI M, BAROGLIO C. Constitutive and regulative specifications of commitment protocols: a decoupled approach[J]. Acm Transactions on Intelligent Systems and Technology, 2013, 4(2): 1-25.

[4] 董孟高, 毛新军, 常志明, 等. 自适应多Agent系统的运行机制和策略描述语言SADL[J]. 软件学报, 2011, 22(4): 609-624.
DONG Meng-gao, MAO Xin-jun, CHANG Zhi-ming, et al. Running mechanism and strategy description language SADL for self-adaptive MAS[J]. Journal of Software, 2011, 22(4): 609-624.

[5] MINSKY N H, ROZENSHTEIN D. A law-based approach to object-oriented programming[C]//Proceedings on Object-Oriented Programming Systems, Languages and Applications. New York, NY, USA: Applied Intelligence, 1987: 482-493.

[6]MINSKY N H, UNGUREANU V. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems[J]. ACM Transactions on Software Engineering Methodology, 2000, 9(3): 273-305.

[7] FORNARA N, COLOMBETTI M. Specifying and enforcing norms in artificial institutions: a retrospective review[C]// Proceedings of the 9th International Workshop on Declarative Agent Languages and Technologies. Taipei, Taiwan, China: Springer, 2012: 117-119.

[8] EI-MENSHAWY M, BENTAHAR J. Reducing model checking commitments for agent communication to model checking ARCTL and GCTL[J]. Autonomous Agents and Multi-Agent Systems, 2013, 27(3): 375-418.

[9] YOLUM A P . Constraint satisfaction as a tool for modeling and checking feasibility of multiagent commitments[J]. Applied Intelligence, 2013, 39(3): 489-509.

[10] DOUGHERTY D J, FISLER K, KRISHNAMURTHI S. Obligations and their interaction with programs[C]// Proceedings of 12th European Symposium on Research in Computer Security. Dresden, Germany: Springer, 2007: 375-389.

[11] CRAVEN R, LOBO J, MA J, et al. Expressive policy analysis with enhanced system dynamicity[C]// Proceedings of the 4th International Symposium on Information, Computer, and Communications Security. New York, USA: Association for Computing Machinery, 2009: 239-250.

[12] ELRAKAIBY Y, CUPPENS F, CUPPENS-Boulahia N. Formalization and management of group obligations[C]// IEEE International Symposium on Policies for Distributed Systems and Networks. London, England: Springer, 2009: 158-165.

[13] 吴丹, 危胜军. 基于模型检测的策略冲突检测方法[J].电子科技大学学报, 2013, 42(5): 745-748.
WU Dan, WEI Sheng-jun. Policy conflict detection method based on model checking[J]. Journal of University of Electronic Science and Technology of China, 2013, 42(5): 745-748.

[14] CHRISTEL B, JOOST-PIETER K. Principles of model checking[M]. [S.l.]: MIT Press, 2008.

[15] ZHANG Tao, HUANG Shao-bi, HUANG Hong-tao. Specification and verification of policy using raise and model checking[C]//International Conference on International Conference on BioMedical Engineering and Informatics. Shanghai, China: [s.n.], 2011.

编 辑 黄 莘