

# 基于可能性方法的认知逻辑与 模型检测的研究与实现

(申请清华大学工程硕士专业学位论文)

培 养 单 位：    软件学院

工 程 领 域：    软件工程

申 请 人：    刘        媛

指 导 教 师：    罗 贵 明 教 授

二〇一五年六月

基于可能性方法的认知逻辑与模型检测的研究与实现

刘媛

# **Epistemic Logic and Model Checking Based on Possibility Method and Implementation**

Thesis Submitted to

**Tsinghua University**

in partial fulfillment of the requirement

for the professional degree of

**Master of Software Engineering**

by

**Liu Yuan**

**(Software Engineering)**

Thesis Supervisor: Professor Luo Guiming

**May, 2015**

# 关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容。

本人保证遵守上述规定。

**（保密的论文在解密后遵守此规定）**

作者签名：

导师签名：

日期：

日期：

## 摘要

模型检测是目前软件工程领域的研究热点，其中虽然概率模型检测已经在定量属性方面被广泛研究与应用，然而现实系统中的有关非确定性的定量性质不是全部都可以用概率描述的，有一部分非确定性其本质不是随机的而是模糊的，这部分系统的描述需要将可能性与模型检测相结合，才可以建模。可能性及其基础模糊数学在人工智能和工业控制领域的大量应用，这种系统的安全性越来越重要。但是可能性模型检测相关研究却非常少。同样可能性模型的属性的相关研究也很少，可能性理论与模态逻辑中的可能性概念有诸多联系，将可能性与模态逻辑结合是一个研究方向，将认知逻辑与可能性结合就可以作为可能性与模态逻辑结合的一个尝试。因此本文将可能性理论与认知逻辑、模型检测相结合，提出了一种新的模型检测方法，解决了不能用概率模型建模的系统的建模与检测问题，主要工作内容如下：

- (1) 本文提出了基于可能性的计算树认知逻辑 **PoCTLK** (Possibilistic Computation Tree Logic of Knowledge) 的语义。通过分析了多智能体系统对不确定性和定量的属性需求，将认知逻辑与可能性方法相结合，用基于分布知识的可能性解释系统为可能性多智能体系统建模，并在该模型下，定义了 **PoCTLK** 的两种语义，来表示多智能体系统中量化的不确定性属性。
- (2) 本文提出了基于可能性方法的模型检测过程。分析了 **PoCTL** 在可能性模型上的语义，提出了 **PoCTL** 可能性算子  $PO_{vb}(X\phi)$ 、 $PO_{vb}(\Phi_1 U^{\leq n} \Phi_2)$  和  $PO_{vb}(\Phi_1 U \Phi_2)$  的计算过程，找到了  $PO_{vb}(X\phi)$ 、 $PO_{vb}(\Phi_1 U^{\leq n} \Phi_2)$  的计算中的迭代规律，为了实现该迭代，自定义了两个矩阵运算符。
- (3) 对本文中提出的模型检测算法给出了实现。通过分析对比 **PoTL** 在 **DTMC** 上的模型检测实现与 **PoCTL** 在可能性模型上的模型检测实现，提出了可能性模型检测算法；通过修改概率模型检测 **PRISM** 工具中的具体检测代码，实现了可能性模型检测算法，从而借助 **PRISM** 工具完成可能性模型检测过程。并选择同步领导选择协议实例，分别给出了其概率版本和可能性版本，通过其证明了可能性模型检测与概率模型检测的区别。

关键词：PoCTLK；可能性模型检测；非确定性；矩阵运算符

## Abstract

Model checking is a hot spot in software engineering recently. And probability model checking on reachability, reliability, and other property has been widely researched for years. However, not all the uncertainties of a system can be described by probability, we can take the fuzzy part of a system as an example. So we need to combine possibility and model checking to solve it. In the meantime, possibility theory got mass application in the area of fuzzy control and artificial intelligence and these applications take safety as their first consideration. Under such conditions, possibility based model is becoming a fertilized land to study while it has just gained few attentions right now. Same thing happens about possibility based temporal property. Possibility is a concept which plays an important role both in possibility theory and temporal property. Possibility based epistemic logic (kind of modal logic) can be a try. In this paper, possibility, epistemic logic and model checking is combined to propose a new model checking method, which can cover the shortage of probability model checking. Main work as follows:

- (1) Proposed a possibility based computation tree logic of knowledge in this paper. We analysis the need of properties about uncertain and quantitative in multi-agent system and combine the epistemic logic and possibility method to propose PoCTLK. Distributed knowledge based interpreted system is used to model the multi-agent system and two kinds of semantics of PoCTLK is defined under it which represent the quantitative properties of multi-agent system.
- (2) The model checking process based on possibility method is given in paper. According to the research into semantics of PoCTL defined on possibility model, we propose a computing process for possibility operator  $PO_{\nabla b}(X\phi)$ 、 $PO_{\nabla b}(\Phi_1 U^{\leq n} \Phi_2)$  and  $PO_{\nabla b}(\Phi_1 U \Phi_2)$ . A discover about the iterative rules with in the computing process about  $PO_{\nabla b}(X\phi)$ 、 $PO_{\nabla b}(\Phi_1 U^{\leq n} \Phi_2)$  is made in the paper. And two matrix operator to accomplish the iterative rules is defined.
- (3) The algorithm about the model checking based on possibility process is implemented in paper. Comparing the algorithm of model checking on DTMC about PCTL properties and on possibility model checking about PoCTL. I propose the algorithm to implement model checking on possibility model about PoCTL. And

modifying the source code of Prism to accomplish the possibility model checking. An experiment is conducted to prove the difference between the probability model checking and possibility model checking, which is the two edition of synchronous leader election protocol, and one of edition is probability, the other is possibility.

**Keywords:** PoCTLK; possibility model checking; uncertain; matrix operator

# 目录

第 1 章 引言 .....	1
1.1 研究背景及意义 .....	1
1.2 相关研究工作 .....	3
1.2.1 可能性模型检测 .....	3
1.2.2 多智能体系统的概率认知逻辑 .....	4
1.3 论文主要工作 .....	5
1.4 论文组织结构 .....	6
第 2 章 相关理论介绍 .....	7
2.1 可能性模型检测理论 .....	7
2.1.1 可能性模型构造 .....	7
2.1.2 可能性计算树逻辑语法 .....	8
2.1.3 可能性计算树逻辑在可能性模型上的语义 .....	9
2.2 概率模型检测在 prism 中的实现 .....	10
2.2.1 概率模型（连续马尔科夫链）构造及概率表示 .....	11
2.2.2 prism 实现概率模型检测的基本集合操作在 BDD 上的实现 .....	13
2.3 多智能体系统的认知逻辑 .....	19
2.4 本章小结 .....	21
第 3 章 可能性模型检测的算法与实现 .....	23
3.1 问题分析 .....	23
3.2 分析可能性模型检测过程 .....	24
3.2.1 $X\phi$ 的可能性计算过程 .....	25
3.2.2 $\Phi1 \cup \Phi2$ 的可能性计算过程 .....	26
3.2.3 $\Phi1 \wedge \Phi2$ 的可能性计算过程 .....	27
3.3 可能性运算在矩阵上的实现 .....	29
3.4 与 MDP 对比 .....	33
3.5 本章小结 .....	37
第 4 章 可能性认知逻辑 .....	38
4.1 普通可能性认知逻辑 .....	38
4.1.1 可能性多智能体系统 .....	38



4.1.2 可能性认知逻辑语法 .....	39
4.1.3 基于可能性解释系统的可能性认知逻辑语义 .....	40
4.1.4 实例说明 .....	40
4.2 深层可能性认知逻辑 .....	41
4.2.1 深层可能性认知逻辑语法 .....	41
4.2.2 基于可能性解释系统的深层可能性认知逻辑语义 .....	42
4.2.3 实例说明 .....	42
4.3 本章小结 .....	44
第 5 章 案例分析 .....	45
5.1 实验环境 .....	45
5.2 可能性模型与概率模型实例对比 .....	45
5.2.1 DTMC 概率模型检测实例 .....	47
5.2.2 可能性模型检测实例 .....	49
5.3 实验结果 .....	51
5.4 本章小结 .....	52
第 6 章 结束语 .....	53
6.1 工作总结 .....	53
6.2 后续研究展望 .....	53
参考文献 .....	55
致谢 .....	58
声明 .....	59
个人简历、在学期间发表的学术论文与研究成果 .....	60

## 第 1 章 引言

### 1.1 研究背景及意义

自从 20 世纪 60 年代软件危机发生后，软件工程师们就开始探索软件工程化的道路，而一部分人则独辟蹊径开始研究形式化方法在软件开发中的应用，试图通过数学的方法来解决软件危机。软件形式化方法通常包括模型检测和定理证明这两个方向。其中模型检测由 Clarke, Emerson 等人<sup>[1][2][3]</sup>首次提出以后，该方法就被广泛的应用在大规模数字电路、通信协议、控制系统、安全认证协议等方面的分析与验证中，许多公司，如 Intel、HP、Phillip、IBM 等都成立了专门的模型检测技术小组负责应用模型检测技术辅助产品的设计与测试。

模型检测作为一种自动的形式化的验证方法之所以受到高校和企业的亲赖，是有其深刻的历史原因的，随着高科技的发展，事故的后果越来越严重，比如一次飞机或者高铁的失事造成的人员伤亡与经济损失之严重程度是社会不可接受的后果。现实有很多实例：1996 年，阿丽亚娜 5 型火箭，耗时数年时间，耗资 80 亿美元，几乎是一升空便爆炸，震惊世界，在随后的调查中发现，事故原因是软件系统中的数值类型转换异常，一个小错误造成了巨大的损失。2010 年 2 月，丰田的第一款量产的混合动力车普锐斯，其防抱死系统软件“glitch”出问题招来了无数事故与投诉，不得不通过花费巨资，召回 185000 辆车升级软件。事故的风险是后果与概率的乘积，后果不可改变的情况下，降低事故发生的概率很必要。在现在和未来，自动驾驶的汽车等等各种无人操作的交通工具必将登上人类历史，但是这些的安全性必须得到保障。而模型检测正是为检测各种协议、硬件、软件中的逻辑错误而生，因此模型检测会越来越受到重视。

模型检测的基本做法<sup>[1]</sup>是把要检测的系统表示为状态变迁系统，把要检测的系统属性用逻辑表述出来，然后对逻辑取反，查看系统与取反的属性是否有交集，如果有交集则表示有系统不满足属性，否则则表示系统满足属性。模型检测的基本方向有很多：其中一大支是研究的目的是减少状态爆炸的影响，由于模型检测的基本思想是遍历系统的状态，而现实系统的状态非常庞大，因此状态爆炸是阻止模型检测应用的一大原因，尽量减少模型的状态有多种技术，比如 on the fly, 偏序归约，组合模型检测，有界模型检测，以及模型检测的另外一个方向符号化模型检测，以及各种工具优化措施都是在减少状态的指导思想下出现的；另一个大支研究的是模型检测理论本身，这一支包括研究建模过程与模型本身，如概

率模型、可能性模型、模型抽象与规约等，研究表示描述系统属性的逻辑，如各种模态逻辑的结合，比如时序逻辑、认知逻辑、道义逻辑及其组合，概率与模态逻辑的结合，可能性与模态时序的结合等。这两个大支的研究不是泾渭分明的，而是紧密结合的。应用催生理论研究，理论又反作用与应用。

On-the-fly<sup>[4]</sup>技术是针对不需要一次性求解全部系统状态的模型检测的状态减少技术，其基本想法是将属性取反后与部分从初始状态可达的状态集合进行与操作，看其结果是否有交集；这里的重点是，不是整个系统模型在完成模型检测，而是从初始状态可达的一个状态集合在完成模型检测，这样做的目的是减少了每次模型检测的状态数量。

偏序归约技术<sup>[5][6]</sup>是针对系统中的路径进行分析，进而发现等价的有限路径，所谓的等价的路径在包含多个的进程的系统中可能出现，这些进程不同的排列顺序形成不同的状态，但是有些进程不同的排列顺序对结果并不产生影响，这些进程的不同排列顺序除了增加状态外没有用处，因此可以给这些进程规定一个特定的顺序即偏序归约技术，从而减少了系统状态。

组合模型检测是把要建模的系统分解成几个部分，分别在各个部分上进行模型检测后，再经过一定的方式组合，从而得到整个系统的模型检测的结果。

符号化模型检测<sup>[2]</sup>是针对显式状态模型检测而言的，初期模型检测工具，如 spin，为单个状态设计了特定的数据结构，采用这种数据结构存储是非常直观的想法，基于这种数据结构模型检测就是显式状态模型检测，后来的模型检测工具，如 NuSmv<sup>[7][8]</sup>，基本数据结构不再存储单独的状态，而是将一个状态集合存储成 BDD<sup>[9][10]</sup> (binary decision diagram) 的形式，BDD 是已知的一种数据结构，采用这种数据结构表示状态变迁系统，并在此基础上进行模型检测的技术就是符号化模型检测。但是实际应用中，并不是所有的符号化模型检测在时间和效率上都胜于显示状态模型检测。

有界模型检测<sup>[11][12][13][14]</sup>也是从表示状态的数据结构入手，将状态集合表示成布尔表达式，从而将模型检测问题转化成布尔公式求解问题，NuSmv 中就实现了这种方法；但是由于有界模型检测的概念，从一开始就不是把整个模型全部转化成布尔公式，而是从一个转移开始，逐渐转化，因此有界模型检测有个重要的概念叫阈值<sup>[15][16]</sup>，当达到阈值之后，如果模型检测结果没有反例才能说明系统满足属性。受限于布尔表达式表达图形的无力与不直接，阈值的计算是基于原图形化的状态表示上的，而且目前为止，阈值的计算与模型检测本身的时间复杂度是一个级别。

不同的属性描述方式要求不同的模型，因为逻辑语义是定义在特定的模型上

的,如时序逻辑语义定义在 kripke 结构上<sup>[1]</sup>,因此时序逻辑描述的属性描述的是用 kripke 结构构建的系统模型的特征;而概率时序逻辑对应的是马尔科夫链<sup>[17]</sup>;认知逻辑<sup>[18][19][20]</sup>对应的是交互系统模型。不同的属性及其对应的模型描述了系统的不同方面,选择哪种取决于现实的需求。描述确定性质的时序逻辑,无论是 LTL 还是 CTL 已经被研究了很长时间,随着模型检测的应用愈来愈广泛,现实中的场景与非真即假的属性相差不小,通常都是一个模糊范围内,满足一个属性,这就需要能够描述不确定性特点的逻辑,概率模型检测和可能性模型检测均可完成这个任务。概率模型检测虽能表示不确定性,但是受限于概率与可能性的不同,无法表达可能性模型检测的含义,尤其是可能性与认知逻辑的天生关系,更是概率无法比拟的。

## 1.2 相关研究工作

本文研究的是可能性时序逻辑在可能性模型上的实现及可能性时序逻辑与认知逻辑的结合及在可能性交互模型上的语义。

### 1.2.1 可能性模型检测

可能性模型检测的属性描述采用的是可能性概念与时序逻辑的结合,模型用的是可能性模型,模型检测过程是基于可能性 CTL 在可能性模型上的语义。

可能性是 Zahedi 于 1978 年<sup>[22]</sup>首次提出的,是对模糊集合和模糊逻辑的一种扩展,自从可能性理论提出后,世人对其与概率论的区别一直抱着质疑态度,后来有多篇文章包括 Zahedi 自己<sup>[22][23][25]</sup>来解释两者的区别。尽管两者都是研究不确定性的概念,但是从第一次提出开始,Zahedi 就给出了两者区别,并用一个吃鸡蛋的例子做了解释。可能性测度的定义有各种具体定义,只要满足可能性的基本性质,便可以根据实际给出具体定义,已知的有基于可能性分布的<sup>[22]</sup>定义,基于一致性的定义<sup>[25]</sup>。

可能性与模型检测的结合,由陕西师范大学数学系教授李永明和数名学生<sup>[26][27][28]</sup>首先研究,他们于 2012 年<sup>[26]</sup>首次定义了可能性 kripke 结构、其上的可能性测度和 LTL 属性在可能性测度下的语法及在可能性 kripke 结构上的语义,并对可能性测度和概率测度进行了对比。2014 年<sup>[27]</sup>定义了 CTL 属性在可能性测度下的语义,从数学角度证明了可能性测度下的 CTL 属性(PoCTL)与经典 CTL 属性的等价性,并通过实例对比可能性测度与概率测度下 CTL 属性的异同。同年<sup>[28]</sup>他们研究了基于广义可能性测度的 CTL (GPoCTL) 模型检测,定义了广义可能性 kripke 结构,及在此之上的广义可能性测度的 CTL 语

中英文统一吧?!

义，并给出了具体算法和复杂度，用实例展示了模型检测的过程，最后比较了 GPoCTL 和 PoCTL。

2014 年 Zahed<sup>[29]</sup>为了探究可能性理论本身，比较了研究可能性理论两大理论体系，一个是模态逻辑中的可能性，一个是可能性理论中的可能性，并利用状态变迁模型做为中介，分别表示了两者的韦恩图；模态逻辑中的可能性表示的是到达某个目标集合的可能性；可能性理论中的可能性指的是事实与目标之间的关系；但是两者的韦恩图是相同的。

### 1.2.2 多智能体系统的概率认知逻辑

模型检测一开始验证的属性都是定性的，直到开始研究概率模型，模型检测才可以检测定量的属性。目前实用性最高的就是概率模型检测工具 PRISM，经过多年发展<sup>[30][31][32]</sup>PRISM 成为成熟的工具，被广泛应用于科研和工作中。2001 年<sup>[30]</sup>首次将概率模型检测过程，用 MTBDD 数据结构作为基础实现，并做了实验，文中模型是概率系统属性是 PBTL。04 年<sup>[31]</sup>采用多终端二元决策图（MTBDD）和 BDD 作为基本数据结构和稀疏矩阵或者两者结合三种方式作为基础，实现了 PCTL、CSL 在 DTMC、CTMC、MDP 上的模型检测过程。12 年<sup>[32]</sup>介绍了概率时间自动机的模型检测，概率时间模型应用在汽车和航空系统的嵌入式控制器中、无线通讯协议如蓝牙等协议中，PRISM 也增加了相应的独立模块来构建、验证、精化概率模型，以及一个显示状态实现概率模型检测的应用程序库，用于统计模型检测的离散事件模拟引擎，并提供了一个标准检查程序库。PRISM 应用广泛<sup>[33]</sup>，包括对设计和编程必须达到安全性和可靠性的最高水平要求的心脏起搏器。近年来，嵌入式软件错误导致安全警报的大幅增加，造成了昂贵的设备召回、患者死亡等后果。为了解决这些问题，该文提出了一种基于模型的框架来定量自动验证起搏器软件。文章采用了克利福德等人的心电图模型，生成正常和异常的心跳的行为，以及一定的概率转换，以产生一个时间序列的动作电位信号作为起搏器输入。使用江等人的心脏起搏器的时间自动机模型，开发了一种基于离散的方法获得的心脏和起搏器的组成。考虑的主要属性是正确性，包括检查起搏器纠正心动过缓（心脏跳动缓慢）和不诱导性心动过速（心跳快），一系列的现实的心的行为。文章也通过对传感器的读数噪声和能源使用分析不足。该文文利用概率模型验证器 PRISM 和 MATLAB 实现的框架展示了令人鼓舞的结果。

认知系统的主流模型有两个：部分可观察马尔可夫决策过程（POMDPs）和解释系统。POMDPs 是一个概括的马尔可夫决策过程（MDP），20 世纪 90 年代<sup>[34][35]</sup>以来被用于对随机代理的知识的不确定性。近年来 POMDPs 被广泛应用于机器学习



习<sup>[36]</sup>，代理决策，和机器人应用。基于框架的 POMDPs，代理只部分观察基础状态并维护可能状态集合上的概率分布，这些状态称为信念状态，是基于观测值来计算的。使用解释系统<sup>[37]</sup>，认知逻辑可以推断知识和时间，广泛用于验证的是多智能体系统<sup>[38]</sup>。认知模态既能表示单个代理的认知，也能表示一组代理的认知，如不同代理在不同的时序逻辑下的常识。使用解释系统表示代理的不确定性知识仍然是一个前景广阔的研究课题。表示和推理随机认知系统的两种方法：POMDPs 和扩展的解释系统。前者已被广泛研究，后者有广阔研究前景。

研究认知逻辑与概率的结合可以追溯到 1999 年，IBM 的研究人员<sup>[39]</sup>提出了适合认知逻辑和概率结合的模型，并给出了模型上的概率公式的语义，通过一个例子来展示了检测过程。Lawry 和 Tang<sup>[40]</sup>利用估值对，估值设置了三值命题：真值，边界，假。命题的值，既不是绝对真实的和绝对错误时，即是非真非假的边界。命题的值不是绝对正确或完全错误，这允许代理考虑不确定性和模糊命题。但是这个逻辑是有限的因为它不能表达命题的概率值。此外，三值逻辑模型检测是一个复杂的过程，这种模型检测的算法是尚未被提出。Cao<sup>[41]</sup>提出 PETL，概率认知时态逻辑。PETL 是一个时态逻辑和概率逻辑组合逻辑的一种尝试，PETL 支持概率知识的表示，如“在具体概率下每个代理的认知”和“概率常识”。PETL 是基于 LTL 的。Wen<sup>[42]</sup>提出了 PCTLK 和基于多智能体知识的概率模型，及模型检测实现。其模型检测过程是把 PCTLK 及其模型的检测过程转化成等价的 PCTL 和 MDP 的模型检测过程，调用 PRISM 来实现。并给了实例，及于 MCMAS 的在时间空间方面的对比。

### 1.3 论文主要工作

本论文完成的工作主要包括三部分：

第一部分是提出了可能性算子的模型检测过程：通过分析可能性模型检测中可能性算子在可能性模型上的语义，给出了可能性算子的具体检测过程，发现了  $PO_{\nabla b}(X\phi)$ 、 $PO_{\nabla b}(\Phi_1 U^{\leq n} \Phi_2)$  检测过程中的迭代性，并为此自定义了两个矩阵运算符；

第二部分实现完成可能性模型检测过程中算法，验证了可能性模型检测实例：通过分析对比概率模型检测过程与可能性模型检测过程，参考 PRISM 的具体实现，修改 PRISM 的源码实现可能性模型检测过程。并应用修改的 PRISM 完成了实例，实例结果证明了可能性模型检测与概率模型检测的不同。

第三部分是分析了多智能体系统对不确定性和定量这样的属性需求，提出了认知逻辑与可能性相结合的属性 PoCTLK。本文用基于分布知识的可能性解释系统

为可能性多智能体系统建模，并定义了在该系统下，定义了PoCTLK的两种语义，来表示多智能体可能性系统中量化的不确定性属性。

## 1.4 论文组织结构

第一章是对课题背景的研究，通过阅读国内外的概率模型检测、可能性测度下的时序逻辑和认知逻辑等论文，调研分析了国内外研究现状，总结本文的主要研究内容的研究意义。

第二章介绍了基本的可能性测度下的模型检测的理论，以及概率模型检测工具 PRISM 的实现过程。

第三章具体介绍了可能性模型检测过程，为了实现该过程自定义的矩阵运算符，及其在 PRISM 上实现的伪码。

第四章提出了认知逻辑与可能性相结合的属性 PoCTL 及基于分布知识的可能性解释系统为多智能体概率系统建模。

第五章我们通过一个实例来应用第三章种实现的可能性模型检测过程，同时证明了可能性模型检测与概率模型检测的本质区别。

第六章总结了本文并提出下一步的工作。

## 第2章 相关理论介绍

### 2.1 可能性模型检测理论

可能性模型检测理论同经典模型检测理论一样，需要研究以下三方面的内容：一是描述系统的模型的构建，二是描述系统属性的语法的定义及语义的定义，三是模型是否满足属性的检测过程的实现。本节就是根据这个思想从以下三个方面介绍可能性模型检测理论。

#### 2.1.1 可能性模型构造

可能性模型是对 Kripke 结构的一种扩展，与概率模型（连续马尔科夫链）的结构极为相似，除去连续马尔科夫链变迁上的数值代表的是状态转移的概率，而可能性模型变迁上的数值表示的是状态转移的可能性，其定义如下所示：

定义 2.1：可能性模型是定义在一组原子命题上的五元组  $M_p = (S, P, I, AP, L)$ ，其中：

$S$ ：非空的可数的状态集合；

$P$ ：变迁可能性函数， $S \times S \rightarrow [0,1]$ ，对于  $S$  中的每个状态， $\bigvee_{t \in S} P(s,t) = 1$ ；

$I$ ：初始状态可能性分布， $S \rightarrow [0,1]$ ， $\bigvee_{s \in S} I(s) = 1$ ；

$AP$ ：原子命题集合；

$L$ ： $S \rightarrow 2^{AP}$  是标签函数。

上述定义中用到  $\bigvee$  表示的是集合上确界运算或者说集合。变迁可能性函数  $P$  用矩阵表示， $S \times S \rightarrow [0,1]$ ， $S \times S$  中的每个元素代表的是一个状态到另一个状态的变迁，它的值代表的是这个变迁的可能性； $\bigvee_{t \in S} P(s,t) = 1$  表示从  $s$  出发的变迁可能性上确界为 1。初始状态可能性分布  $I$  是用列向量来表示的，该列向量的行数等于非空的可数的状态集合的大小，每行的数值代表的是对应状态是初始状态的可能性； $\bigvee_{s \in S} I(s) = 1$  表示某个状态是全集中初始状态的可能性的上确界为 1。其余的字符的含义同经典 Kripke 结构相同。

图 2-1 是一个简单的可能性模型的例子。其中， $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ ， $I = \{1, 0, 0, 0, 0, 0\}$ ， $L(s_1) = \{\text{感冒}\}$ ， $L(s_2) = \{\text{风寒感冒}\}$ ， $L(s_3) = \{\text{细菌感染感冒}\}$ ， $L(s_4) = \{\text{抗生素}\}$ ， $L(s_5) = \{\text{银翘解毒片}\}$ ， $L(s_6) = \{\text{治愈}\}$



$$P = \begin{bmatrix} 0 & 1 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0.5 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.3 \\ 1 & 0 & 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}。$$

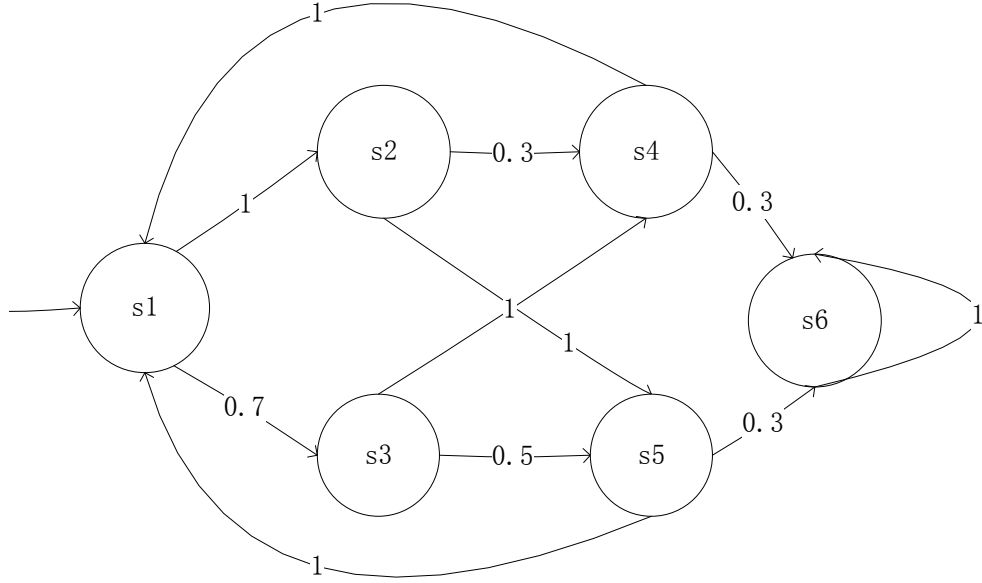


图 2-1 病情状态转移图

### 2.1.2 可能性计算树逻辑语法

可能性计算树逻辑（Possibilistic Computation Tree Logic，简称为 PoCTL）是对经典 CTL 的一种扩展，可以定性的描述系统属性的逻辑。PoCTL 公式包含两种，状态公式，路径公式。

定义 2.2: PoCTL 状态公式是定义在 AP 集合上的，

$$\Phi ::= \text{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid \text{PO}_{\nabla b}(\varphi)$$

其中  $\varphi$  是路径公式， $a \in \text{AP}$ 。  $\nabla \in \{<, \leq, >, \geq\}$ ，  $b \in [0, 1]$ 。

定义 2.3: PoCTL 路径公式，

$$\varphi ::= X\Phi \mid \Phi_1 U \Phi_2 \mid \Phi_1 U^{\leq n} \Phi_2$$

其中  $\Phi$ 、 $\Phi_1$ 、 $\Phi_2$  是状态公式，  $n \in \mathbb{N}$ 。

$\text{PO}_{<0.5}(\varphi)$  表示路径公式  $\varphi$  的可能性低于 0.5 的路径集合。

PoCTL 与经典 CTL 公式不同的有如下几点：(1)经典 CTL 状态公式中不存在定量表示的属性，而 PoCTL 存在定量描述路径的可能性的公式，  $\text{PO}_{\nabla b}(\varphi)$ ；(2)经典

CTL 路径公式前要用 A, E 两个算子来修饰, 从而表达了所有路径满足还是某条路径满足该公式, 而 PoCTL 路径公式前用可能性算子来修饰, 表达的是满足特定可能性的路径集合, CTL 的 A, E 算子可以用特殊的可能性算子来实现; 3. 经典 CTL 公式不包含  $U^{\leq n}$  算子, 而 PoCTL 中加入了  $U^{\leq n}$  算子, 因为在 PoCTL 中在至多  $n$  步之内到达  $\Phi_2$  的可能性和在至多  $n+1$  步之内到达  $\Phi_2$  的可能性不同。

### 2.1.3 可能性计算树逻辑在可能性模型上的语义

PoCTL 的公式中与经典 CTL 公式中重合的部分, 其语义没有发生变化, 因此本文不再赘述, 此处只给出  $PO_{\forall b}(\varphi)$  的语义。在给出 PoCTL 的语义之前, 先了解可能性模型的测度定义。只有定义了模型的测度, 才能定义 PoCTL 的语义。测度是将一个集合中的子集与特定的数字进行映射的函数, 测度定义需要三个要素, 集合  $X$ , 集合  $X$  上的  $\sigma$  代数  $A$ , 及  $A$  上定义的一个满足可数可加性的及空集映射为 0 的函数  $\mu$ ,  $\{X, A, \mu\}$  三元组构成了测度空间。可能性模型的测度是定义在可能性模型路径幂集上的。可能性模型  $M$  的全部无穷路径集合记为  $Paths(M)$ , 全部有穷路径集合记为  $Paths_{fin}(M)$ , 从  $s$  出发的全部无穷路径集合记为  $Paths(s)$ , 其中以  $s_0 \dots s_m$  为前缀的无穷路径集合由  $Cyl(\hat{\pi})$  表示,  $Cyl(\hat{\pi}) = s_0 \dots s_m \dots$ 。

定义 2.4 : 设  $M$  是可能性模型, 令  $X = \{Cyl(\hat{\pi}) | \hat{\pi} \in Paths_{fin}(M)\}$ ;  $A = 2^{Paths(M)}$ , 是定义在  $X$  上的  $\sigma$  代数, 且满足  $X \subseteq A$ , 如果  $\varepsilon \subseteq A$ , 那么  $\bar{\varepsilon} = A - \varepsilon \subseteq A$ , 如果  $B_i \subseteq A$ ,  $i=1, 2, \dots$ , 那么  $\bigcup_{i=1}^{\infty} B_i \subseteq A$ ;  $Po$  是定义在  $A$  上的非负集合函数,  $Po: Paths(M) \rightarrow [0, 1]$ , 且满足  $\forall \varepsilon \subseteq A$ ,  $Po(\varepsilon) \geq 0$ ,  $Po(X) = 1$ ,  $\forall \varepsilon, \mu \subseteq A$   $Po(\varepsilon \cup \mu) = \max(Po(\varepsilon), Po(\mu))$ ;  $\{X, A, Po\}$  构成了可能性测度空间。

在给出了可能性模型上关于无穷路径集合的测度空间后, 定义了以  $s_0 \dots s_m$  为前缀的无穷路径集合  $Cyl(\hat{\pi})$  的可能性计算公式, 如下所示:

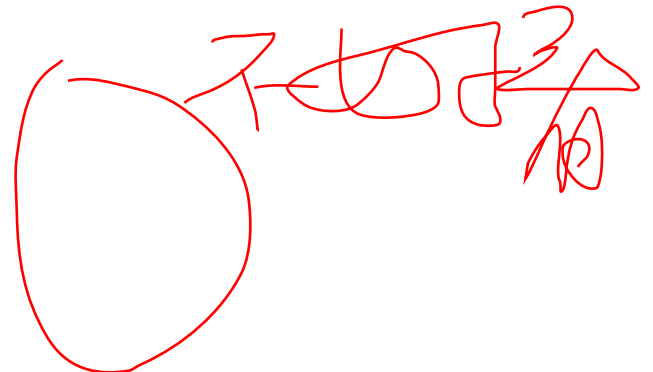
$$\begin{aligned} \{ Po(Cyl(\hat{\pi})) = Po(Cyl(s_0 \dots s_m)) = Po(s_0 \dots s_m \dots) = I_{init}(s_0) \wedge Po(s_0 \dots s_m) \\ Po(s_0 \dots s_m) = \bigwedge_{i=0}^{m-1} P(s_i, s_{i+1}) \end{aligned}$$

接下来给出  $PO_{\forall b}(\varphi)$  语义:

$s \models PO_{\forall b}(\bigcirc \phi)$ , 当  $Po(s, Paths(s), \bigcirc \phi) \nabla b$ ,

$Po(s, Paths(s), \bigcirc \phi) = \bigvee_{\pi \in Paths(s), \pi(1) \models \phi} Po(s, \pi(1))$ ;

$s \models PO_{\forall b}(\phi_1 U^{\leq n} \phi_2)$ , 当  $Po(s, Paths(s), \phi_1 U^{\leq n} \phi_2) \nabla b$ ,



$$Po(s, Paths(s), \phi_1 U^{\leq n} \phi_2) = \begin{cases} 1 & , \text{当 } s \models \phi_2 \\ \bigvee_{\forall \pi \in Paths(s) \pi \models \phi_1 U^{\leq n} \phi_2} Po(Cyl(\hat{\pi})) & , \text{当 } s_m \models \phi_2, 0 < m \leq n, \\ & , \forall 0 \leq i < m, s_i \models \phi_1 \\ 0 & , \text{以上都不是} \end{cases}$$

$s \models PO_{\forall b}(\phi_1 U \phi_2)$ , 当  $Po(s, Paths(s), \phi_1 U \phi_2) \nabla b$ ,

$$Po(s, Paths(s), \phi_1 U \phi_2) = \begin{cases} 1 & , \text{当 } s \models \phi_2 \\ \bigvee_{\forall \pi \in Paths(s) \pi \models \phi_1 U \phi_2} Po(Cyl(\hat{\pi})) & , \text{当 } s_m \models \phi_2, \forall 0 \leq i < m, s_i \models \phi_1 ; \\ 0 & , \text{以上都不是} \end{cases}$$

## 2.2 概率模型检测在 prism 中的实现

概率模型检测工具 prism 的系统框架图如下图所示，主要包括两部分：模型构建和模型检测。

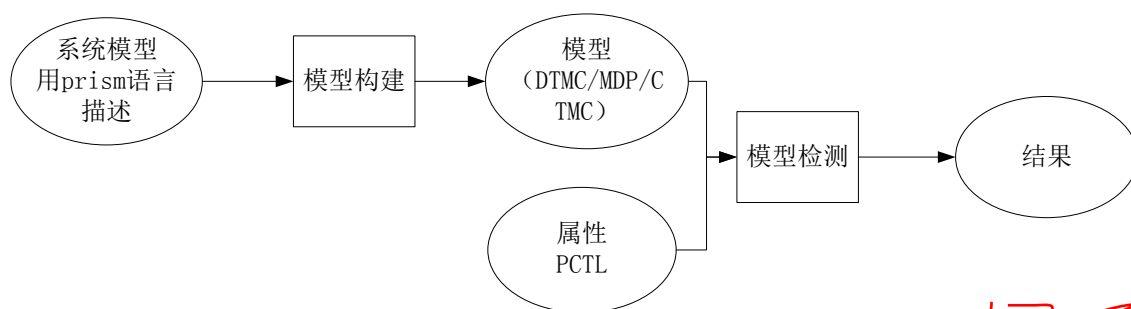


图 2-2 prism 模型检测具体步骤

prism 工具读取用 prism 语言描述的系统模型（概率状态变迁模型），并将概率状态变迁模型存成 DTMC、MDP、CTMC 中的一种，然后根据 PCTL 在模型上的语义生成的算法进行模型检测，得到结果。

其中的模型构建过程和模型检测的具体实现分别如图 2-3、图 2-4 所示。

概率模型有两种基本数据结构，一种是显式状态：使用数学概念来表示状态，另一种是符号化表示法：使用 BDD 及其变体作为基本数据结构。显式状态的作为基础时，使用位向量来表示状态集合，使用实数集合来表示实数向量，使用稀疏矩阵来表示实数矩阵。符号化表示法中使用 BDD 来表示状态集合，使用 MTBDD 来表示实数向量和实数矩阵。

概率模型检测中常用的算法有两种，一种是基于图算法的，此处的图指的是以状态为结点，以 DTMC/MDP/CTMC 的变迁为边的图，在模型检测过程中需要

计算可达状态等运算，可以将可达状态转换成在图中找可达点集，即把模型检测过程中运算转换成通过现成的图算法（如最大连通子图）等算法来完成的运算；另一种是数学迭代计算，概率模型检测涉及到了概率的计算，其中由于 DTMC/MDP/CTMC 的概率变迁使用矩阵进行表示，因此进行矩阵乘积等运算可以完成模型检测过程。

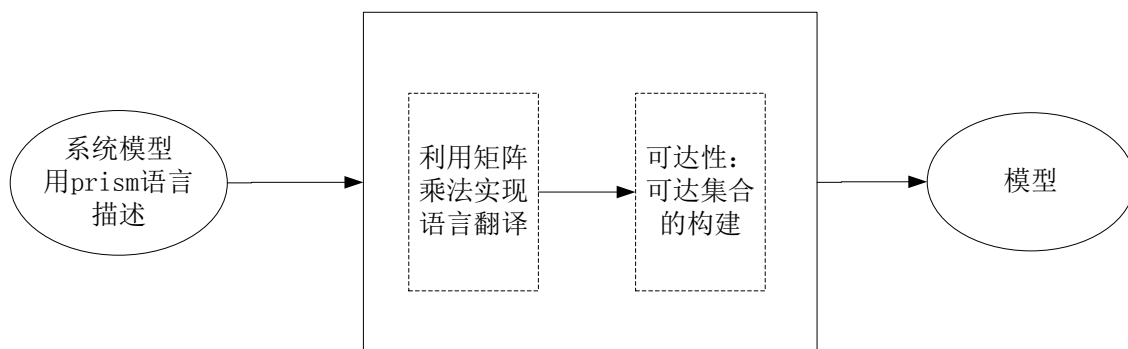


图 2-3 模型构造具体步骤

模型检测算法针对两种数据结构，分别采用不同的计算方法，采用显示状态表示的模型使用迭代数值计算法来实现模型检测，主要操作是矩阵乘法的数值计算；而采用 BDD 表示的模型使用图算法来实现模型检测，主要操作是矩阵乘法在 MTBDD 上的实现。



图 2-4 模型检测具体步骤

### 2.2.1 概率模型（连续马尔科夫链）构造及概率表示

本文主要关注的是 Prism 采用 BDD 作为基本数据结构存储概率模型和利用图算法进行集合操作。因为显示状态表示法比符号化表示法理论上更容易出现状态

爆炸。本节以下图为例，来表示 BDD 是如何表示状态集、变迁集。

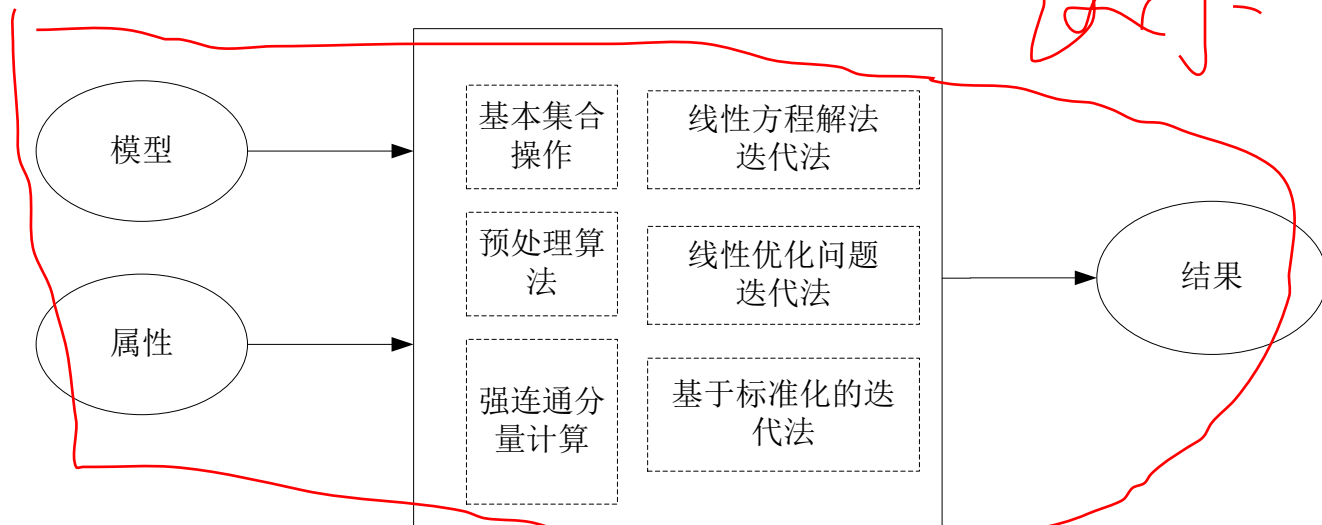


图 2-5 概率自动机示例

初始状态分布  $\{s_1 \rightarrow 1, s_2 \rightarrow 0, s_3 \rightarrow 0, s_4 \rightarrow 0\}$  可以用两个布尔变量  $x_0, x_1$  来表示成列向量，变迁集  $\{s_1s_2 \rightarrow 0.5, s_1s_3 \rightarrow 0.5, s_2s_4 \rightarrow 0.3, s_2s_3 \rightarrow 0.7, s_3s_4 \rightarrow 0.7, s_3s_1 \rightarrow 0.3, s_4s_1 \rightarrow 1\}$  用矩阵表示，可以用  $x_0, x_1$  表示该矩阵的行向量，可以用  $y_0, y_1$  表示该矩阵的列向量，状态集和变迁集的真值表及 BDD 表示下所示：

表 2-1 状态集的布尔变量表示

	$x_0$	$x_1$	$f_V$
$s_1$	0	0	1
$s_2$	0	1	0
$s_3$	1	0	0
$s_4$	1	1	0

表 2-2 变迁集的布尔变量表示

变迁	$x_0$	$x_1$	$y_0$	$y_1$	$x_0y_0x_1y_1$	$f_M$
$s_1s_2$	0	0	0	1	0001	0.5
$s_1s_3$	0	0	1	0	0100	0.5
$s_2s_4$	0	1	1	1	0111	0.3
$s_2s_3$	0	1	1	0	0110	0.7
$s_3s_4$	1	0	1	1	1101	0.7
$s_3s_1$	1	0	0	0	1000	0.3

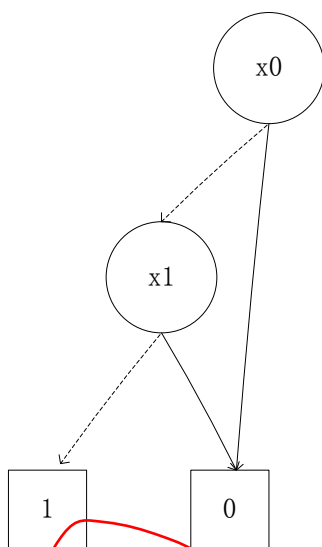


图 2-6 状态集 BDD 表示

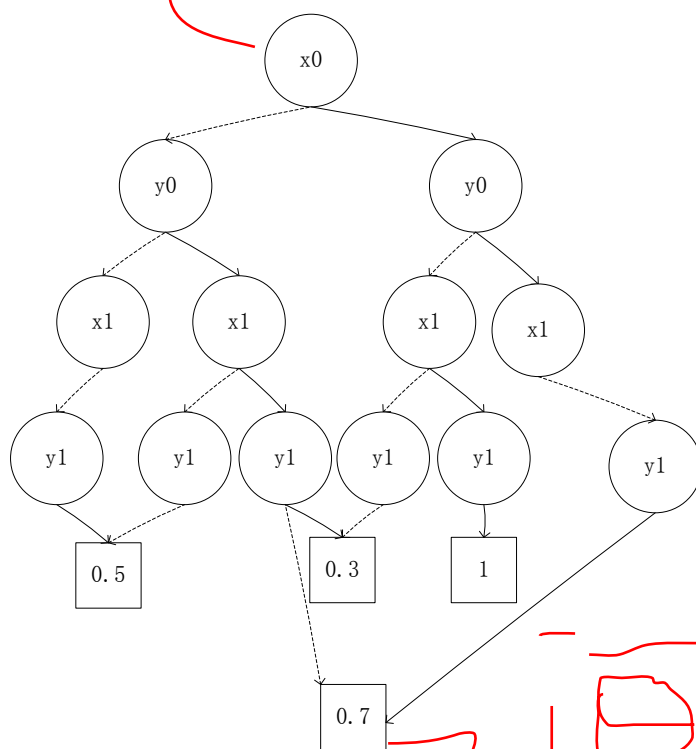


图 2-7 变迁集 BDD 表示

问题

### 2.2.2 prism 实现概率模型检测的基本集合操作在 BDD 上的实现

在 2.2.2 节中我们给出了状态集合和变迁集合的 BDD 表示形式，接下来要考虑的就是在 BDD 图上的集合操作。主要研究向量乘法、向量乘以矩阵、矩阵乘法。

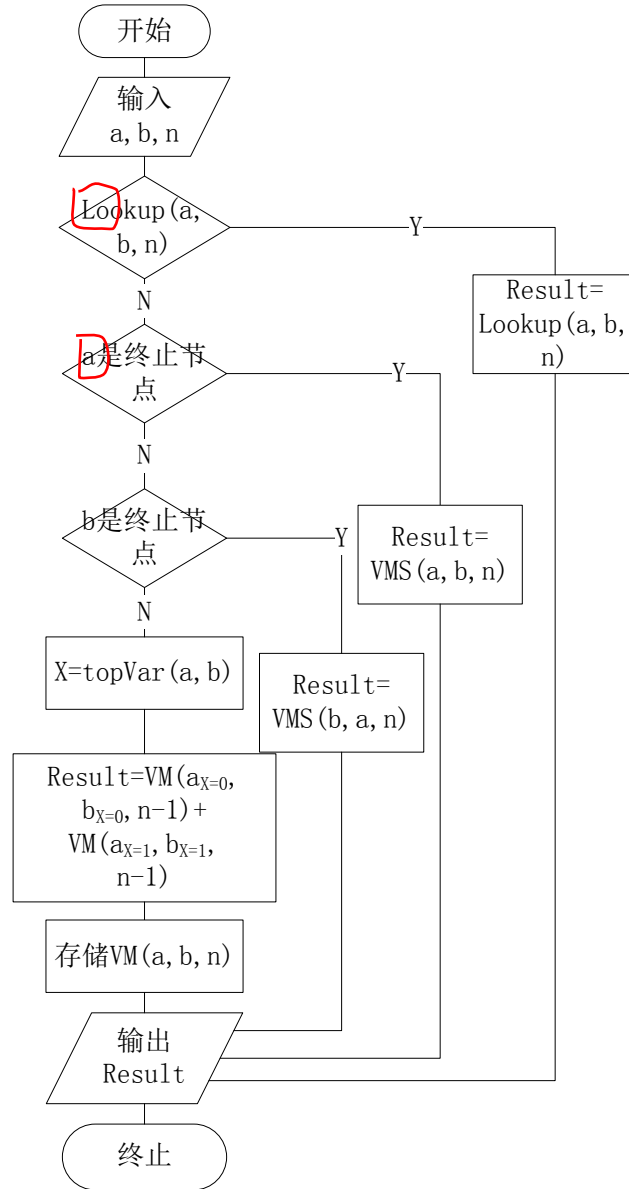
其实现基础是香农展开定理： $f(x_1, x_2, \dots, x_n) = x_1 f(1, x_2, \dots, x_n) + \bar{x}_1 f(0, x_2, \dots, x_n)$ 。

向量乘法，令  $a, b$  是长度为  $n$  的向量， $a \cdot b = \sum_{i=0}^{n-1} a_i b_i$ ，用布尔函数表示可以

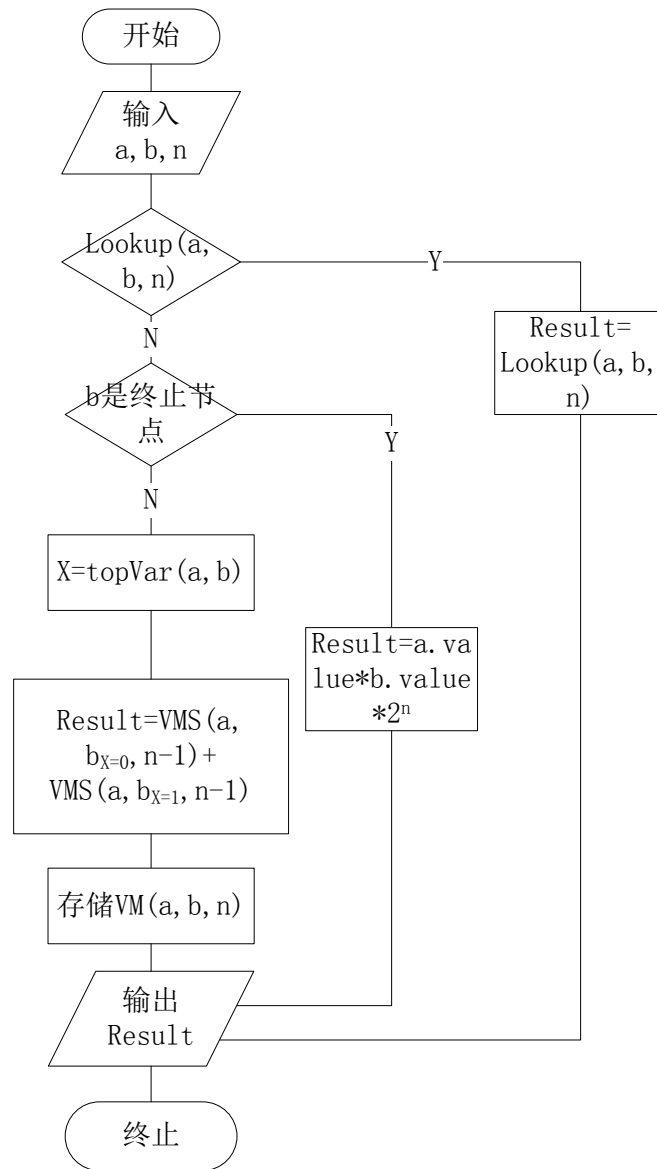
写成:

$$r = [a_{x_1} \dots a_{x_l}] \begin{bmatrix} b_{x_1} \\ \vdots \\ b_{x_l} \end{bmatrix}$$

因此  $a \cdot b = a_{x_1} \cdot b_{x_1} + a_{x_2} \cdot b_{x_2} + \dots + a_{x_l} \cdot b_{x_l}$ 。接下来我们用流程图来表示向量乘法的具体步骤。



(a) VM(a, b, n) 的流程图



(b) VMS (a,b,n) 流程图

图 2-7 向量相乘流程图

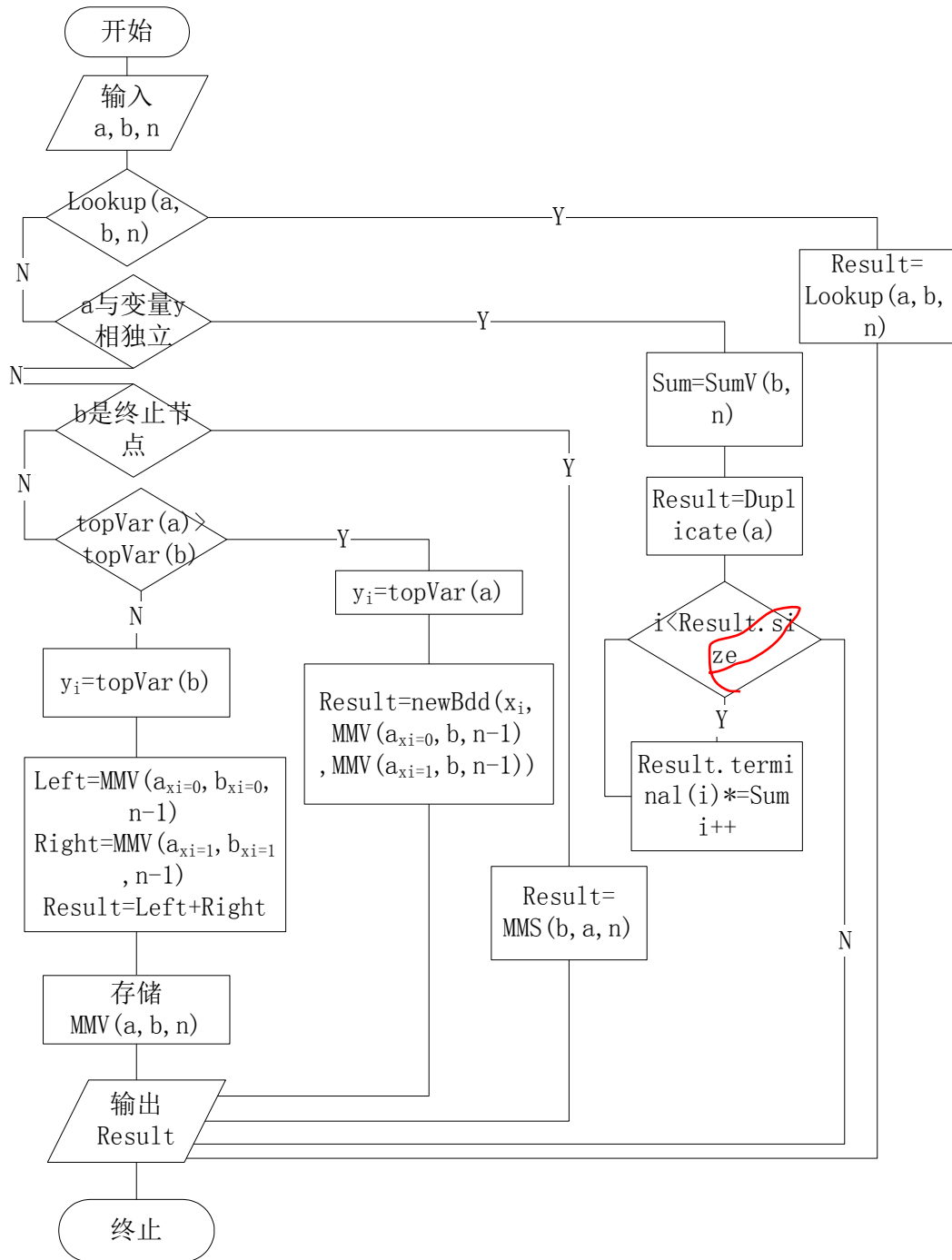
*Times new?*

向量乘以矩阵，令  $\mathbf{a}$  是  $m \times n$  阶矩阵， $\mathbf{b}$  是长度为  $n$  的向量， $\mathbf{r}(x_1, x_2 \dots x_n) = \mathbf{a}(x_1 y_1, x_2 y_1 \dots x_n y_1, \dots x_m y_1, x_m y_1 \dots x_m y_n) \cdot \mathbf{b}(y_1 \dots y_n)$ 。

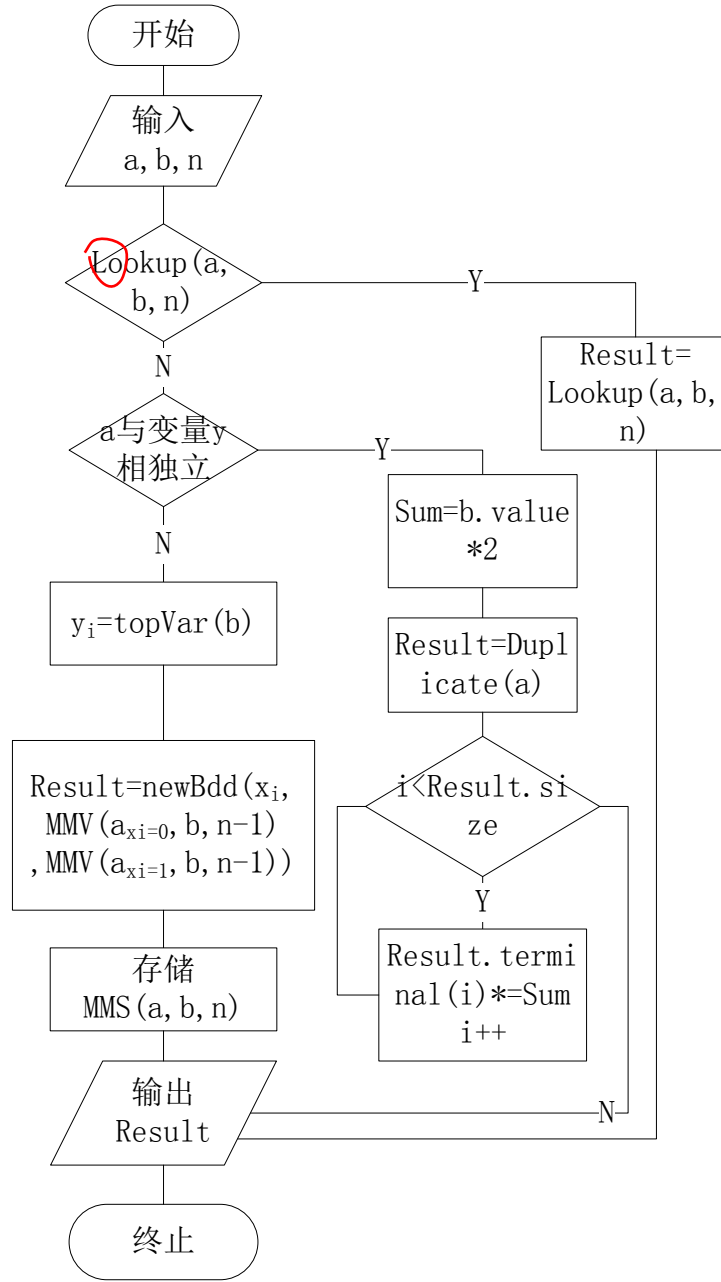
$$\begin{bmatrix} r_{x_1} \\ r_{x_1} \end{bmatrix} = \begin{bmatrix} a_{x_1 y_1} & a_{x_1 y_1} \\ a_{x_1 y_1} & a_{x_1 y_1} \end{bmatrix} \begin{bmatrix} b_{x_1} \\ b_{x_1} \end{bmatrix}$$

$r_{x_1} = a_{x_1 y_1} * b_{x_1} + a_{x_1 y_1} b_{x_1}$ ,  $r_{x_1} = a_{x_1 y_1} * b_{x_1} + a_{x_1 y_1} b_{x_1}$  接下来我们用流程图来表示向量乘以矩阵的具体步骤。





(a) MMV (a,b,n) 的流程图



(b) MMS (a,b,n) 的流程图

图 2-8 矩阵乘向量流程图

矩阵乘法，两个矩阵相令  $a$ ， $b$  分别是  $m*n$  和  $n*p$  矩阵，  
 $r(x_1y_1, x_1y_2 \dots x_1y_n, \dots, x_my_1, x_my_2 \dots x_my_n) = a(x_1z_1, x_1z_2 \dots x_1z_p, \dots, x_nz_1, x_nz_2 \dots x_nz_p) \cdot b(z_1y_1, z_1y_2 \dots z_1y_n, \dots, z_py_1, z_py_2 \dots z_py_n)$

$$\begin{bmatrix} \overline{r_{x_1y_1}} & \overline{r_{x_1y_1}} \\ \overline{r_{x_1y_1}} & \overline{r_{x_1y_1}} \end{bmatrix} = \begin{bmatrix} \overline{a_{x_1z_1}} & \overline{a_{x_1z_1}} \\ \overline{a_{x_1z_1}} & \overline{a_{x_1z_1}} \end{bmatrix} \begin{bmatrix} \overline{b_{z_1y_1}} & \overline{b_{z_1y_1}} \\ \overline{b_{z_1y_1}} & \overline{b_{z_1y_1}} \end{bmatrix}$$

接下来我们用流程图来表示向量乘以矩阵的具体步骤。

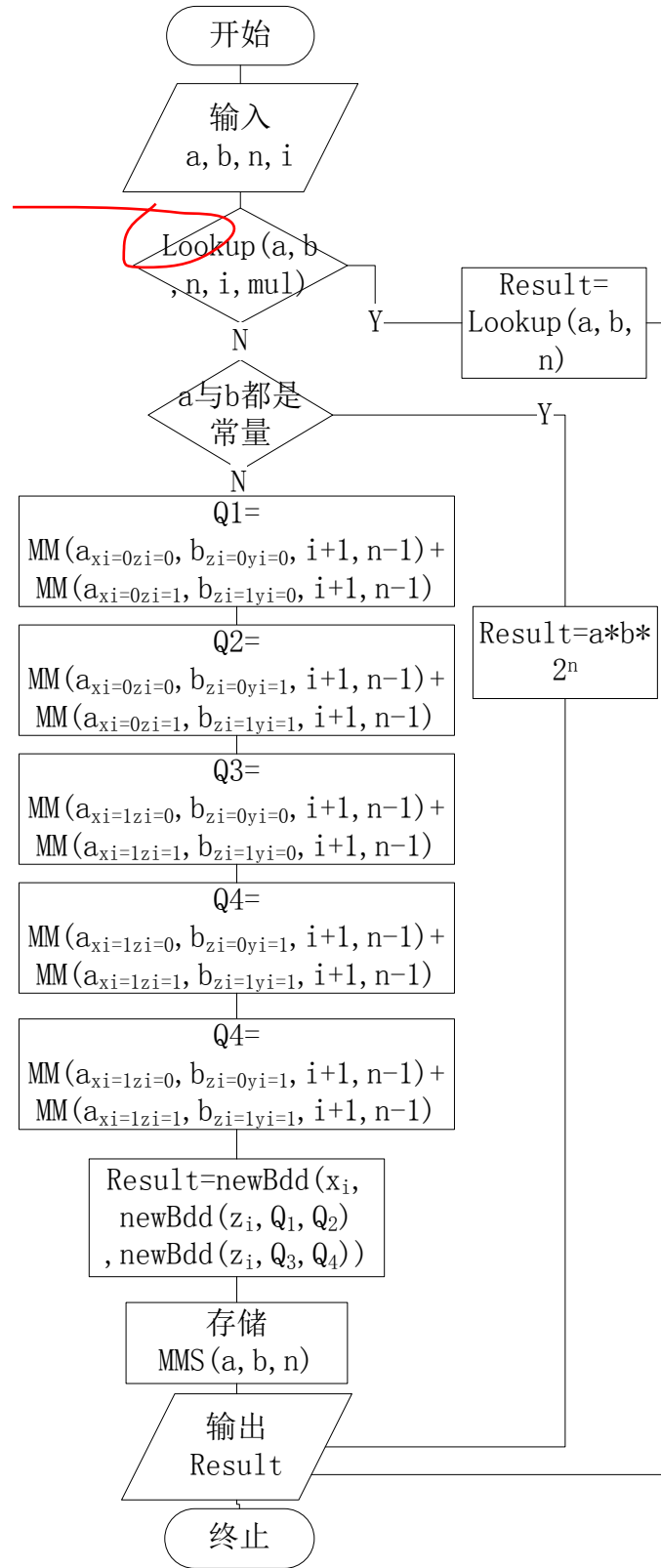
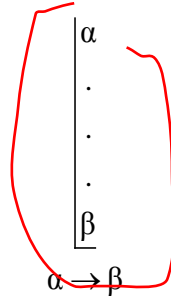


图 2-9 矩阵乘法流程图

### 2.3 多智能体系统的认知逻辑

认知逻辑来源于推理系统。假设我们的代理使用一步一步的自然推理规则做推理，如： $\alpha_1, \dots, \alpha_n \triangleright \beta$ ，从 $\alpha_1, \dots, \alpha_n$ 推断出 $\beta$ 。这里最重要的是每一步推理规则，每次推理都需要应用代理的一组资源。一个典型的例子就是 $\alpha, \beta \triangleright \alpha \wedge \beta$ 。



上述公式中，垂直线标志的假设范围即 $\alpha$ 。这条规则要求做如下事情：首先，假设 $\alpha$ ；然后，在假设的情况下正常推理；最后，假设终止和推断出公式，推导出公式 $\beta$ 。

将这样的推理规则在一步规则系统中编码，并将假设作为上下文。不需要假设的规则被允许内嵌到任意上下文中。输入上下文（一步操作）对应于制定相应的假设，离开上下文（又一个单步操作）对应于假设终止和在假设外推断出相应的公式。给出了这样一组一步规则，一个使用这些推断规则可能的方法推断模型构建如下：模型是变迁系统，其中每个变迁即为应用了一个一步推理规则。这个想法是可以用下图说明。假设代理使用“与引入”推断。然后，从 $p, q$ 推理的可能途径，如下：

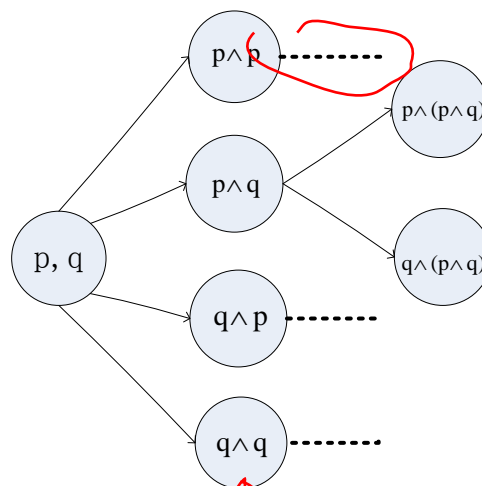


图 2-10 “与引入”模型图

变迁系统只是一个关系结构：灰色方块是它的状态，箭头表明持状态之间的

转换关系。在这个例子中，每个箭头对应应用“与引入”一步推理规则，一般来说，变迁对应一步规则。相应的，每一个可能的规则应用对应着从一个状态变迁到下一个。

方括号“[]”指的是写在灰色框内的公式即状态的标签。括号区分了“ $\wedge$ ”出现的位置，变迁图中根状态用 $[p] \wedge [q]$ 表示，而不是 $[p \wedge q]$ 。一个模型系统是一个三元组 $\langle S, T, V \rangle$ ，其中  $S$  是状态集， $T \subseteq S \times S$  是状态间的变迁关系， $V(S \times L \rightarrow \{\text{真}, \text{假}\})$  是打标函数，为每种状态公式对分配一个值。

这个模型的特殊性在于  $T$  和  $V$  的互动方式。在代理推断新公式时，变迁 $T_{su}$ 捕捉了一步法规则中的可能应用。 $T_{su}$ 中， $u$  被标记在根据变迁对应规则到达的状态  $s$  上。代理资源限制是由模型中分支路径的长度表示的。代理只能推断  $n$  步对应模型中相应分支路径中至多包含  $n + 1$  状态。

认知可能性即，如果代理根据其所有的资源，在已知  $\alpha_1.. \alpha_N$  的情况下，不能从集合 $\{1, \alpha, \dots, \alpha_N, \beta\}$ 推断出一个明确的矛盾，那么代理认为 $\beta$ 是可能的。一个代理的认知逻辑中的可能性是指这个代理根据已知信息和可用资源不能判断真假的命题，则该命题对于该代理是认知可能的。

举例来说，假设一个较弱的棋手将要下一步糟糕的棋  $m$ 。对于这个代理， $m$  看起来像一个好棋（或至少不是一个特别糟糕的一步）。对于这个代理这步棋是好棋是认知可能的，尽管它实际上是一个糟糕一步。事实上，鉴于国际象棋的规则和游戏状态（假设代理知道）， $m$  不可能是一步好棋。一个好的棋手可以看出  $m$  是一个糟糕棋，也许通过使用她的推理能力的发现  $m$  为对手提供了一个制胜战略。好的棋手，可以排除  $m$  是好棋，所以  $m$  是一步好棋对于好的棋手来说在认知逻辑上不是可能的。

一个代理的推理认知逻辑上的可能性的能力和他拥有的资源有关。假设一个代理  $a$  已知如下信息： $p_1 \vee \dots \vee p_n \vee p_{n+1}$  和  $\neg p_1 \wedge \dots \wedge \neg p_n$ ，可以利用“与消去”， $\alpha \vee \beta, \neg \alpha \supset \beta$  进行推理。那么对于这个代理， $p_{n+1}$  是认知可能的吗？明显不是，如果代理  $a$  没有足够的时间去推断，如果  $a$  有足够时间就可以在  $2n$  次应用规则后推出明显的矛盾 $p_{n+1}, \neg p_{n+1}$ 。假设代理  $b$  和  $a$  一样有相同的初始化信息，相同的推理规则，但是只有  $n$  步推理资源。 $b$  不能在  $n$  步之内根据这些已知内容推出矛盾。因此对于代理  $b$  来说 $\neg p_{n+1}$ 是认知可能的，尽管  $b$  和  $a$  有相同的已知信息。下图显示了  $n=2$  时推理的可能的路径。

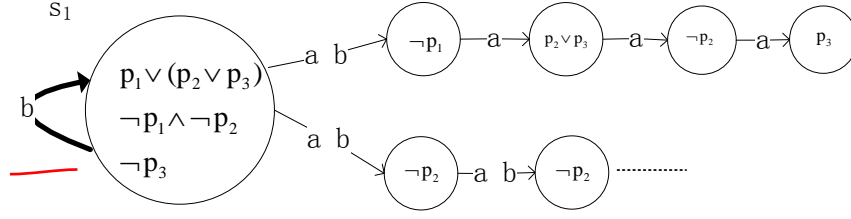


图 2-11 “与消去”例子模型图

直观地，最左边状态标签是对于  $b$  是认知可能的，但是对于  $a$  是认知不可能的。即对于代理  $b$  根状态中的命题是认知可达的，而对于代理  $a$  根状态中的命题是认知不可达的。我们不需要代理  $a$  真实的发现的矛盾，考虑代理已知信息和拥有的资源，只要知道代理是能够发现矛盾就足够了。

设  $i$  是任意一个代理， $\delta_i \in \mathbb{N}$  是代理  $i$  的时间限制，代表了在变迁系统中代理能够走得变迁数目。从状态  $s$  可以推出一个明显的矛盾，如果对于公式  $\alpha \in \mathcal{L}$ ， $\exists s', T^{\delta_i} s s'$  以及  $s' \models [\alpha] \wedge [\neg \alpha]$ 。在上述例子中，设  $\delta_a = 4$ ， $\delta_b = 2$ 。 $s_1$  上的标签是不一致的，在四步之内使用推理规则可以推出矛盾。因此，代理  $a$  从  $s_1$  出发可以发现明显的矛盾，而  $b$  只能推理两步，因此  $b$  推不出矛盾。因此， $s_1$  对于  $b$  来说是认知可能的，而对于  $a$  不是。图中浅色的变迁是状态间的变迁，粗黑色的变迁展示的是认知可能关系。

代理  $i$  的认知可能变迁关系表示成  $\sim_i$ ，其形式化定义如下：

$s \sim_i u$  当且仅当  $\forall v (T^{\delta_i} u v \text{ 意味着 } v \models [\alpha] \wedge [\neg \alpha])$

$T^{\delta_i} u v$  指的是从  $u$  出发在  $\delta_i$  个变迁之内  $v$  是可达的。令  $\mathcal{L}$  是代理语言，是基础命题  $p_1 p_2$  在布尔连接词下的闭包。而关于代理的信息描述组成了模态语言  $\mathcal{ML}$ ， $\mathcal{ML}$  的基础是  $[\alpha]$ ， $\alpha \in \mathcal{L}$ 。如果  $\mathcal{ML}$  的一个布尔自由公式  $\phi$ ，那么  $K_i \phi$  是  $\mathcal{ML}$  的公式之一。

$\alpha \in \mathcal{L}$  对于代理  $i$  来说是认知可能的，记作  $E_i[\alpha]$ ，如果对于代理  $i$  有个标签为  $\alpha$  的状态是认知可达的。在上面的例子中，考虑到  $b$  只能推理两步， $s_1$  对于  $b$  是认知可达的。 $\neg p_1$ ， $\neg p_2$ ， $p_2 \vee p_3$  对于代理  $a$  是认知可能的。从可达状态中推断出的公式对于代理来说是认知可能的。用公式表示如下： $M, s \models K_i \phi$  当且仅当  $\exists u, v (s \sim_i u, T^{\delta_i} u v \text{ and } M, v \models \phi)$ 。 $K_i K_j [\alpha]$  是良好定义的公式，其含义是对于代理  $i$  来说代理  $j$  满足  $\alpha$  是认知可能的。

## 2.4 本章小结

第2章主要介绍的是可能性模型检测理论(2.1节)和概率模型检测工具 Prism 的实现过程(2.2节)。其中可能性模型检测理论，又分为三个部分介绍，分别是

可能性模型的构造（2.1.1 节），可能性计算树逻辑的语法（2.1.2 节）及其在可能性模型上的语义（2.1.3）；而概率模型检测工具 **Prism** 的实现过程，则从 **Prism** 的大框架开始介绍，然后是 **Prism** 的基本数据结构及其用法（2.2.1 节）和基本数据结构上的操作（2.2.2 节）；多智能体系统的认知逻辑的来源，及其基础公式含义的介绍（2.3 节）。

### 第 3 章 可能性模型检测的算法与实现

理论分析

#### 3.1 问题分析

[ ]

可能性模型检测理论研究始于 2011 年，但是其软件实现一直没有人做过，本文准备实现可能性模型检测软件。

首先本文对可能性模型检测与概率模型检测做对比。经对比，可以发现可能性模型与概率模型的定义有诸多相似之处，首先这两个模型都是五元组，五元组中也包括状态集、原子命题集和标签函数，不同的是可能性模型中另外两个元组分别是初始状态可能性分布和变迁可能性函数，而概率模型的另外两个元组分别是初始状态概率分布和概率变迁函数。其中初始状态可能性分布满足的是  $\bigvee_{s \in S} I(s) = 1$ ，而初始状态概率分布满足的是  $\sum_{s \in S} I_{init}(s) = 1$ ；变迁可能性函数需要满足对于  $S$  中的每个状态， $\bigvee_{t \in S} P(s, t) = 1$ ，概率变迁函数需要满足的是对于  $S$  中的每个状态  $\sum_{s' \in S} P(s, s') = 1$ 。这些区别来自于概率与可能性的概念区别。

可能性计算树逻辑 (PoCTL) 与概率计算树逻辑 (PCTL) 的语法在与经典 CTL 重合的部分是相同的，而且两者都去掉了经典 CTL 中描述路径的量词  $\Box$  (代表全部路径)、 $E$  (代表有一条路径)，不同的是 PoCTL 的状态公式中新加入了描述路径的可能性的公式  $PO_{vb}(\varphi)$ ，而概率计算树逻辑中与经典 CTL 比多的是描述路径的概率的公式  $P_{vb}(\varphi)$ 。

PoCTL 在可能性模型上的语义与 PCTL 在概率模型上的语义在与经典 CTL 重合部分是相似的，在  $PO_{vb}(\varphi)$  和  $P_{vb}(\varphi)$  两个公式的语义是不同的 (参考 2.1.3 节中  $PO_{vb}(\varphi)$  在可能性模型上的语义)，因为这两个公式的语义基础分别是可能性模型上的可能性测度和概率模型上的概率测度，它们计算依赖的共享前缀无穷路径集合的计算不同，可能性测度公式如下：

$$\begin{aligned} Po(Cyl(\hat{\pi})) &= Po(Cyl(s_0 \dots s_m)) = Po(s_0 \dots s_m \dots) = I(s_0) \wedge Po(s_0 \dots s_m), \\ Po(s_0 \dots s_m) &= \bigwedge_{i=0}^{m-1} Po(s_i, s_{i+1}); \end{aligned}$$

与此同时，概率测度公式如下：

$$\begin{aligned} Prob(Cyl(\hat{\pi})) &= Prob(Cyl(s_0 \dots s_m)) = I_{init}(s_0) \cdot P(s_0 \dots s_m), \\ P(s_0 \dots s_m) &= \prod_{0 \leq i < m} P(s_i, s_{i+1}). \end{aligned}$$



不难看出，一条有限路径的可能性等于这条路径上所有变迁的可能性的最小值，而一条有限路径的概率等于这条路径上所有变迁的概率的乘积。

综上，可以看出可能性模型检测过程与概率模型检测过程框架有很大程度似性。但是在具体检测过程中上由于可能性与概率概念不同导致的两种逻辑语义不同，导致了其模型检测过程不同。因此可以在成熟的概率模型检测工具 **Prism** 上作修改来实现可能性模型检测过程。本文实现的可能性模型检测工具是基于 **Prism** 的，第一个难点是分析 **MDP** 模型检测与可能性模型检测的区别；第二个难点是概率模型检测的具体实现流程和在 **Prism** 中的代码实现要分析清楚（参见 2.2 节）；第三个难点是分析比较可能性模型检测和概率模型检测相同不同之处，在具体实现流程和在代码实现方面实现上的体现；第四个难点是将可能性运算定义在矩阵上。

### 3.2 分析可能性模型检测过程

分析可能性模型检测过程，还是要从了解概率模型检测过程开始。参考 2.2 节，可以知道计算  $P_{vb}(\varphi)$  时，要先计算每个状态的概率。所以计算  $PO_{vb}(\varphi)$  时，也应该先计算每个状态的可能性。因此首先研究的是概率和可能性的异同。可能性和概率虽然都是表示不确定性的概念，但是不论从数学角度还是从现实意义都有明显区别。从意义方面来看，一般来说，概率与统计不可分割，概率的定义基础是无穷多次重复实验基础上，计算出某个事件发生的频率即为概率；而可能性与统计是完全无关的，是掌握一定现实情况下，对情况的估计。可能性测度自提出以来在不同的具体场景下有各种具体定义，有基于可能性分布的<sup>[22]</sup>定义，有基于一致性的定义<sup>[25]</sup>，等等具体定义，zadeh<sup>[22][25]</sup>分别给出了在基于上述两种定义下概率测度与可能性测度的异同。

本文分析的是 2.1.3 节中定义的基于可能性模型的路径集合可能性测度与 3.1 节提到的基于概率模型的路径集合的概率测度，可以看到在计算一条路径的可能性测度为  $Po(s_0 \dots s_m) = \bigwedge_{i=0}^{m-1} Po(s_i, s_{i+1})$ ，即一条路径的可能性值为这条路径上所有变迁的可能性的最小值；而概率测度的基本运算为  $P(s_0 \dots s_m) = \prod_{0 \leq i < m} P(s_i, s_{i+1})$ ，其含义是一条路径的概率为这条路径上所有状态的概率的乘积。在计算路径集合的可能性测度时，本文参考的是 zadeh<sup>[3]</sup>在关于可能性操作符的定义：令  $A$  和  $B$  是  $U$  的任意模糊子集，那么， $Po(A \cup B) = Po(A) \vee Po(B)$ ， $Po(A \cap B) \leq Po(A) \wedge Po(B)$ ；当  $A$  与  $B$  没有交集时， $\pi(A \cap B) = \pi(A) \wedge \pi(B)$ 。对比  $A$ ， $B$  的概率测度满足  $P(A \cup B) \leq P(A) + P(B)$ ， $P(A \cap B) \leq P(A)P(B)$ ，当  $A$  与  $B$  没有交集时， $P(A \cup B) = P(A) + P(B)$ ， $P(A \cap B) = P(A)P(B)$ 。

本文在接下来的三小节中分别介绍 **Prism** 中  $X\varphi$ ， $\Phi_1 \cup^{\leq n} \Phi_2$ ， $\Phi_1 \cup \Phi_2$  的概率

计算过程，同时给出这些公式的可能性计算过程。

### 3.2.1 $X\varphi$ 的可能性计算过程

$X\varphi$ :  $\text{Prob}(s, X\varphi) = \sum_{s' \in \text{Sat}(\varphi)} P(s, s')$ ，由于符号化模型检测的核心就是把状态集合表示成列向量，概率变迁是用矩阵表示，单独计算某个状态的概率是显式模型检测的特点，符号化模型检测需要同时计算所有的状态的概率。 $\text{Prob}(X\varphi) = P s_\varphi$ ，其中  $s_\varphi$  是 0—1 向量，且当  $s \models \varphi$  时， $s_\varphi = 1$ ， $P$  是概率变迁函数。 $P \cdot s_\varphi$  计算所得结果是一列向量，结果列向量中每个元素的值是概率变迁矩阵中的一行中的元素分别乘以对应状态集合列向量中的元素的加和，概率变迁矩阵中的操作的那行记录的是结果列向量中对应的元素表示的状态到系统中所有状态的变迁概率，状态集合列向量的赋值是，满足  $\varphi$  的状态为 1，不满足  $\varphi$  的为 0，所以结果列向量中每个元素的值都是存在本状态到满足  $\varphi$  的状态的变迁的概率的加和。假设概率变迁矩阵和列向量如下列公式所示：即  $[x_{i1} x_{i2} \dots x_{in}]$  表示的是状态  $i$  到状态 1 到  $n$  变迁概率，列向量  $[y_1 y_2 \dots y_n]$  中  $y_i = 1$  表示的是状态  $i$  满足  $\varphi$ ，因此结果列向量中第  $i$  个状态的值为  $x_{i1}y_1 + x_{i2}y_2 + \dots + x_{in}y_n = \sum_{s' \in \text{Sat}(\varphi)} P(s_i, s')$ 。

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} x_{11}y_1 + x_{12}y_2 + \dots + x_{1n}y_n \\ x_{21}y_1 + x_{22}y_2 + \dots + x_{2n}y_n \\ \dots \\ x_{m1}y_1 + x_{m2}y_2 + \dots + x_{mn}y_n \end{bmatrix}$$

计算所有状态满足  $X\varphi$  的可能性：每个状态的  $X\varphi$  的可能性公式为  $\text{Po}(s, X\varphi) = \vee_{s' \in \text{Sat}(\varphi)} \text{Po}(s, s')$ ，同时计算所有的状态的可能性时，结果列向量中每个元素的值等于存在到本状态的变迁且满足  $\varphi$  的状态的可能性的最大值，鉴于  $s_\varphi$  的定义，则凡是满足  $\varphi$  的状态对应  $s_\varphi$  中的值为 1，否则值为 0，所以在计算所有状态满足  $X\varphi$  的可能性时，所以结果列向量中每个元素的值都是存在本状态到满足  $\varphi$  的状态的变迁的可能性与对应  $s_\varphi$  的乘积中的最大值，即为每个元素的值都是存在本状态到满足  $\varphi$  的状态的变迁的可能性的最大值，但是没有现成的矩阵运算可以实现这一运算。因此本文为解决这一问题，自定义了一种矩阵向量最大值运算，记为  $\boxplus$ ，其定义由如下表示：

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} \boxplus \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} \text{Max}(x_{11}y_1, x_{12}y_2, \dots, x_{1n}y_n) \\ \text{Max}(x_{21}y_1, x_{22}y_2, \dots, x_{2n}y_n) \\ \dots \\ \text{Max}(x_{m1}y_1, x_{m2}y_2, \dots, x_{mn}y_n) \end{bmatrix}$$

则  $Po(X\varphi) = P \boxtimes s_\varphi$ 。

以图 2-1 为例，计算  $Po(X\varphi)$ ， $\varphi = \{\text{风寒感冒}\}$ ， $s_\varphi = [0, 1, 0, 0, 0, 0]$ ， $P$  是变迁可能性矩阵， $Po(X\varphi)$  的计算如下：

$$Po(X\varphi) = P \boxtimes s_\varphi = \begin{bmatrix} 0 & 1 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0.5 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.3 \\ 1 & 0 & 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boxtimes \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

### 3.2.2 $\Phi_1 U^{\leq n} \Phi_2$ 的可能性计算过程

$\Phi_1 U^{\leq n} \Phi_2$ ：计算一个状态满足  $\Phi_1 U^{\leq n} \Phi_2$  的概率，要分区间，在完全不满足  $\Phi_1 U^{\leq n} \Phi_2$  状态集合  $s^{no}$  中， $s^{no}$  中的所有状态的满足  $\Phi_1 U^{\leq n} \Phi_2$  概率为 0，而在完全满足  $\Phi_1 U^{\leq n} \Phi_2$  的状态集合  $s^{yes}$  中， $s^{yes}$  中的所有状态的满足  $\Phi_1 U^{\leq n} \Phi_2$  概率为 1。因此首先计算状态集合  $s^{no}$  和状态集合  $s^{yes}$ ， $s^{yes} = \text{Sat}(\Phi_2)$ ， $s^{no} = S \setminus (\text{Sat}(\Phi_1) \cup \text{Sat}(\Phi_2))$ 。剩余的状态集合记为  $s^{\text{unknown}} = S \setminus (s^{no} \cup s^{yes})$ ，当  $n=0$  时， $s^{\text{unknown}}$  中的状态满足  $\Phi_1 U^{\leq 0} \Phi_2$  概率为 0，因为本状态不满足  $\Phi_2$ ，而 0 步内只能到达本状态；当  $n>0$  时， $s^{\text{unknown}}$  中的某个状态的概率为  $\sum_{s' \in S} P(s, s') \text{Prob}(s', \Phi_1 U^{\leq n-1} \Phi_2)$ 。要同时计算所有状态满足  $\Phi_1 U^{\leq n} \Phi_2$  的概率需要迭代，其中概率变迁矩阵中的元素赋值规则如下：当  $s \subseteq s^{\text{unknown}}$  时， $P'(s, s') = P(s, s')$ ，当  $s \subseteq s^{yes}$  且  $s = s'$  时， $P'(s, s') = 1$ ，除去以上两种情况  $P'(s, s') = 0$ 。  $\text{Prob}(\Phi_1 U^{\leq 0} \Phi_2) = \text{Sat}(\Phi_2)$ ， $\text{Prob}(\Phi_1 U^{\leq n} \Phi_2) = P' \text{Prob}(\Phi_1 U^{\leq n-1} \Phi_2)$ 。

计算一个状态满足  $\Phi_1 U^{\leq n} \Phi_2$  的可能性时也要分区间，在完全不满足  $\Phi_1 U^{\leq n} \Phi_2$  状态集合  $s^{no}$  中， $s^{no}$  中的所有状态的满足  $\Phi_1 U^{\leq n} \Phi_2$  可能性为 0，而在完全满足  $\Phi_1 U^{\leq n} \Phi_2$  的状态集合  $s^{yes}$  中， $s^{yes}$  中的所有状态的满足  $\Phi_1 U^{\leq n} \Phi_2$  可能性为 1。当  $n=0$  时，定义  $s^{\text{unknown}}$  中的状态的可能性为 0，因为本状态不满足  $\Phi_2$ ，而 0 步内只能到达本状态；当  $n>0$  时， $s^{\text{unknown}}$  中的某个状态的可能性为  $\bigvee_{s' \in S} (\wedge (P(s, s'), Po(s', \Phi_1 U^{\leq n-1} \Phi_2)))$ ，即

$$Po(s, \Phi_1 U^{\leq n} \Phi_2) = \begin{cases} 1 & , \text{当 } s \subseteq s^{yes} \\ \bigvee_{s' \in S} (\wedge (P(s, s'), Po(s', \Phi_1 U^{\leq n-1} \Phi_2))) & , \text{当 } n > 0 \text{ 且 } s \subseteq s^{\text{unknown}} \\ 0 & , \text{当 } n = 0 \text{ 且 } s \subseteq s^{\text{unknown}} \\ 0 & , \text{当 } s \subseteq s^{no} \end{cases}$$

计算所有状态满足  $\Phi_1 U^{\leq n} \Phi_2$  的可能性也是使用迭代计算，其中概率变迁矩阵中的元素赋值规则如下：当  $s \subseteq s^{\text{unknown}}$  时， $P'(s, s') = P(s, s')$ ，当  $s \subseteq s^{yes}$  且  $s = s'$  时，

$P'(s, s') = 1$ , 除去以上两种情况  $P'(s, s') = 0$ 。  $Po(\Phi_1 U^{\leq 0} \Phi_2) = \text{Sat}(\Phi_2)$ 。 计算  $s^{\text{unknown}}$  中所有状态满足  $\Phi_1 U^{\leq n} \Phi_2$  的可能性时, 结果列向量中每个元素的值都是存在本状态到满足  $\varphi$  的状态的变迁的可能性与对应  $Po(s', \Phi_1 U^{\leq n-1} \Phi_2)$  中的最小值中的最大值, 但是没有现成的矩阵运算可以实现这一运算。 因此本文为解决这一问题, 自定义了一种矩阵向量元素最小值的最大值运算, 记为  $\boxplus$ , 其定义由如下表示:

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} \boxplus \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} \text{Max}(\text{Min}(x_{11}, y_1), \text{Min}(x_{12}, y_2), \dots, \text{Min}(x_{1n}, y_n)) \\ \text{Max}(\text{Min}(x_{21}, y_1), \text{Min}(x_{22}, y_2), \dots, \text{Min}(x_{2n}, y_n)) \\ \dots \\ \text{Max}(\text{Min}(x_{m1}, y_1), \text{Min}(x_{m2}, y_2), \dots, \text{Min}(x_{mn}, y_n)) \end{bmatrix}$$

则  $Po(\Phi_1 U^{\leq n} \Phi_2) = P' \boxplus Po(\Phi_1 U^{\leq n-1} \Phi_2)$ 。

其实 3.2.1 节中定义的  $\boxtimes$  在计算  $Po(X\varphi)$  是  $\boxplus$  的特例, 因为  $Po(X\varphi) = P \boxtimes s_\varphi$ , 而  $s_\varphi$  是个 01 列向量, 结果向量的值为  $\text{Max}(x_{i1}y_1, x_{i2}y_2, \dots, x_{in}y_n)$ , 其中  $y_i \in s_\varphi$ ,  $x_{ji} \in [0,1]$ , 当  $y_i = 0$  时,  $x_{ji}y_i = 0$ , 相当于  $x_{ij}y_j = \text{Min}(x_{ji}, 0) = 0$ , 当  $y_i = 1$  时,  $x_{ji}y_i = x_j$ , 相当于  $x_{ij}y_j = \text{Min}(x_{ji}, 1) = x_{ji}$ 。

以图 3-1 为例, 计算  $Po(\Phi_1 U^{\leq 1} \Phi_2)$ ,  $P$  是变迁可能性矩阵,  $Po(\Phi_1 U^{\leq 1} \Phi_2)$  的计算如下:

$$\begin{aligned} \Phi_1 &= \{\text{风寒感冒}, \text{细菌感染感冒}, \text{感冒}\}, \quad \Phi_2 = \{\text{治愈}\}, \\ s^{\text{yes}} &= \text{Sat}(\Phi_2) = \{s_6\}, \quad s^{\text{no}} = S \setminus (\text{Sat}(\Phi_1) \cup \text{Sat}(\Phi_2)) = \{s_4, s_5\}, \\ s^{\text{unknown}} &= S \setminus (s^{\text{no}} \cup s^{\text{yes}}) = \{s_1, s_2, s_3\}, \\ Po(\Phi_1 U^{\leq 0} \Phi_2) &= [0, 0, 0, 0, 0, 1] \end{aligned}$$

$$Po(\Phi_1 U^{\leq 1} \Phi_2) = P' \boxplus Po(\Phi_1 U^{\leq 0} \Phi_2) = \begin{bmatrix} 0 & 1 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0.5 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0.3 \\ 1 & 0 & 0 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boxplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

### 3.2.2 $\Phi_1 \cup \Phi_2$ 的可能性计算过程

$\Phi_1 \cup \Phi_2$ : 计算某个状态满足  $\Phi_1 \cup \Phi_2$  的概率, 首先计算出完全不满足  $\Phi_1 \cup \Phi_2$  状态集合  $s^{\text{no}}$  和完全满足  $\Phi_1 \cup \Phi_2$  的状态集合  $s^{\text{yes}}$ ,  $s^{\text{yes}} = \text{Sat}(P_{\geq 1}[\Phi_1 \cup \Phi_2])$ ,  $s^{\text{no}} = \text{Sat}(P_{\leq 0}[\Phi_1 \cup \Phi_2])$ , 容易看出  $s^{\text{yes}}$  中的所有状态的概率为 1,  $s^{\text{no}}$  中的所有状态的概率为 0。 除去  $s^{\text{yes}}$  和  $s^{\text{no}}$  的剩余的状态集合记为  $s^{\text{unknown}} = S \setminus (s^{\text{no}} \cup s^{\text{yes}})$ , 概率为

$\sum_{s' \in S} P(s, s') \text{Prob}(s', \Phi_1 \cup \Phi_2)$ 。PRISM 中并没有给出同时计算所有状态的  $\Phi_1 \cup \Phi_2$  的概率的迭代过程，相反采用了线性方程，然后利用雅可比迭代法或者高斯-赛德尔迭代去解线性方程，从而计算  $s^{\text{unknown}}$  中满足公式的概率。

计算所有状态满足  $\Phi_1 \cup \Phi_2$  的可能性，同计算概率时一样的步骤有计算  $s^{\text{no}}$  和  $s^{\text{yes}}$  及  $s^{\text{unknown}}$ ，并定义  $s^{\text{yes}}$  中的所有状态的可能性为 1， $s^{\text{no}}$  中的所有状态的可能性为 0。 $s^{\text{unknown}}$  中的某个状态的可能性为  $\bigvee_{s' \in S} (\wedge (P(s, s'), \text{Po}(s', \Phi_1 \cup \Phi_2)))$ 。

$$\text{Po}(s, \text{Paths}(s), \Phi_1 \cup \Phi_2) = \begin{cases} 1 & , \text{当 } s \in s^{\text{yes}} \\ 0 & , \text{当 } s \in s^{\text{no}} \\ \bigvee_{s' \in S} (\wedge (P(s, s'), \text{Po}(s', \Phi_1 \cup \Phi_2))) & , \text{否则} \end{cases}$$

经过研究发现  $\text{Po}(\Phi_1 \cup \Phi_2)$  的计算过程没有办法用矩阵迭代表示，只能用线性计算方法计算。

以图 3-1 为例，计算  $\text{Po}(\Phi_1 \cup \Phi_2)$ ， $P$  是变迁可能性矩阵， $\text{Po}(\neg a \cup b)$  的计算如下：

$$s^{\text{no}} = \text{Sat}(P_{\leq 0}[\Phi_1 \cup \Phi_2]) = S \setminus \text{Sat}(P_{> 0}[\Phi_1 \cup \Phi_2]) = \{s_1, s_3\},$$

$$s^{\text{yes}} = \text{Sat}(P_{\geq 1}[\Phi_1 \cup \Phi_2]) = S \setminus (P_{< 1}[\Phi_1 \cup \Phi_2]) = \{s_4, s_5\},$$

$$s^{\text{unknown}} = S \setminus (s^{\text{no}} \cup s^{\text{yes}}) = \{s_0, s_2\},$$

$$\text{Po}(s_1 \models \Phi_1 \cup \Phi_2) = \text{Po}(s_3 \models \Phi_1 \cup \Phi_2) = 0,$$

$$\text{Po}(s_4 \models \Phi_1 \cup \Phi_2) = \text{Po}(s_5 \models \Phi_1 \cup \Phi_2) = 1,$$

$$\begin{aligned} \text{Po}(s_2 \models \Phi_1 \cup \Phi_2) &= \text{Max}(\text{Min}(0.4, \text{Po}(s_2 \models \Phi_1 \cup \Phi_2)), \text{Min}(0.5, \text{Po}(s_3 \models \Phi_1 \cup \Phi_2))), \\ &\quad \text{Min}(0.5, \text{Po}(s_4 \models \Phi_1 \cup \Phi_2)), \text{Min}(1, \text{Po}(s_5 \models \Phi_1 \cup \Phi_2))) = \\ &= \text{Max}(\text{Min}(0.4, \text{Po}(s_2 \models \Phi_1 \cup \Phi_2)), 0, 0.5, 1) = 1, \end{aligned}$$

$$\begin{aligned} \text{Po}(s_0 \models \Phi_1 \cup \Phi_2) &= \text{Max}(\text{Min}(1, \text{Po}(s_1 \models \Phi_1 \cup \Phi_2)), \text{Min}(0.7, \text{Po}(s_2 \models \Phi_1 \cup \Phi_2))), \\ &= \text{Max}(0, 0.7) = 0.7 \end{aligned}$$

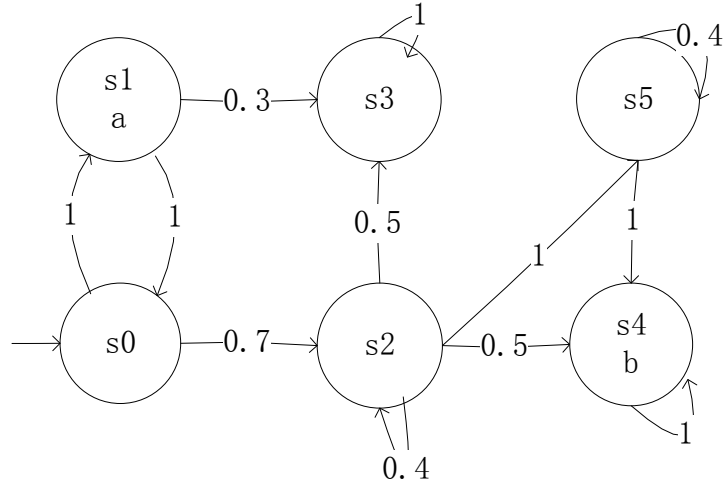


图 3-1 可能性模型实例

### 3.3 可能性运算在矩阵上的实现

由 3.2 节可知，如果要想实现每个状态 $X\varphi$ 可能性的计算，需要在 BDD 表示矩阵和向量下实现矩阵与向量的 $\boxtimes$ 运算。

给出 $\boxtimes$ 运算的具体步骤之前，先定义一个矩阵上的 Max 运算，下边提到的  $\max(m,n)$  的值为实数  $m,n$  中大的那个。令  $a(x_1y_1, \dots, x_1y_n, \dots, x_my_1, \dots, x_my_n)$  与  $b(x_1y_1, \dots, x_1y_n, \dots, x_my_1, \dots, x_my_n)$  同为  $m \times n$  阶矩阵， $\text{Max}(a,b) = (\max(a[x_1y_1], b[x_1y_1]), \dots, \max(a[x_1y_n], b[x_1y_n]), \dots, \max(a[x_my_1], b[x_my_1]), \dots, \max(a[x_my_n], b[x_my_n]))$ 。

$$\text{例: } a = \begin{bmatrix} 4 & 2 \\ 6 & 0 \end{bmatrix}, b = \begin{bmatrix} 4 & 6 \\ 3 & 6 \end{bmatrix}, \text{Max}(a,b) = \begin{bmatrix} 4 & 6 \\ 6 & 6 \end{bmatrix}$$

矩阵与向量的 $\boxtimes$ 运算，令  $a$  是  $m \times n$  阶矩阵， $b$  是长度为  $n$  的向量， $r(x_1, x_2 \dots x_n) = a(x_1y_1, x_1y_2 \dots x_1y_n, \dots, x_my_1, x_my_2 \dots x_my_n) \boxtimes b(y_1, y_2 \dots y_n)$ 。

$$\begin{bmatrix} r_{\bar{x}_1} \\ r_{x_1} \end{bmatrix} = \begin{bmatrix} a_{\bar{x}_1\bar{y}_1} & a_{\bar{x}_1y_1} \\ a_{x_1\bar{y}_1} & a_{x_1y_1} \end{bmatrix} \boxtimes \begin{bmatrix} b_{\bar{x}_1} \\ b_{x_1} \end{bmatrix}$$

$r_{\bar{x}_1} = \text{Max}(a_{\bar{x}_1\bar{y}_1} \boxtimes b_{\bar{x}_1}, a_{\bar{x}_1y_1} \boxtimes b_{x_1})$ ,  $r_{x_1} = \text{Max}(a_{x_1\bar{y}_1} \boxtimes b_{\bar{x}_1}, a_{x_1y_1} \boxtimes b_{x_1})$ 。首先看个例子。

$$\begin{bmatrix} 0 & 2 & 0 & 5 \\ 3 & 0 & 0 & 5 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 3 & 0 \end{bmatrix} \boxtimes \begin{bmatrix} 1 \\ 2 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ 10 \\ 3 \end{bmatrix}$$

$$a_{\overline{x_1}y_1} = \begin{bmatrix} 0 & 2 \\ 3 & 0 \end{bmatrix}, a_{\overline{x_1}y_1} = \begin{bmatrix} 0 & 5 \\ 0 & 5 \end{bmatrix}, a_{x_1\overline{y_1}} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, a_{x_1y_1} = \begin{bmatrix} 0 & 5 \\ 3 & 0 \end{bmatrix},$$

$$b_{\overline{x_1}} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, b_{x_1} = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

$$a_{\overline{x_1}y_1} \boxtimes b_{\overline{x_1}} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}, a_{\overline{x_1}y_1} \boxtimes b_{x_1} = \begin{bmatrix} 10 \\ 10 \end{bmatrix}, a_{x_1\overline{y_1}} \boxtimes b_{\overline{x_1}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, a_{x_1y_1} \boxtimes b_{x_1} = \begin{bmatrix} 10 \\ 3 \end{bmatrix},$$

$$r_{\overline{x_1}} = \text{Max}(a_{\overline{x_1}y_1} \boxtimes b_{\overline{x_1}}, a_{\overline{x_1}y_1} \boxtimes b_{x_1}) = \begin{bmatrix} 10 \\ 10 \end{bmatrix},$$

$$r_{x_1} = \text{Max}(a_{x_1\overline{y_1}} \boxtimes b_{\overline{x_1}}, a_{x_1y_1} \boxtimes b_{x_1}) = \begin{bmatrix} 10 \\ 3 \end{bmatrix},$$

$$r = \begin{bmatrix} r_{\overline{x_1}} \\ r_{x_1} \end{bmatrix} = \begin{bmatrix} 10 \\ 10 \\ 10 \\ 3 \end{bmatrix}.$$

在 prism 中矩阵和向量以 BDD 的形式存放，本文将在 BDD 上定义  $\text{Max}(a,b)$  和  $a \boxtimes b$  的实现伪码。

---

**算法 3.1** 矩阵 Max 运算伪码

输入：a,b 是用 BDD 表示的矩阵，n 表示的 a, b 矩阵用 BDD 表示使用的变量数

输出：result 是对矩阵 a,b 做 Max 操作后的结果

---

**function** Max(a,b,n) {

1. **if**((result = Lookup(a,b,n,Max))  $\neq$  NULL)
  2.     **return** result;//Max(a,b,n)的操作已经执行过一次，其结果保存在 cache 表中
  3. **if**(a is terminal && b is terminal){//a,b 在 BDD 表示下只有一个终止节点
  4.     **if**(a > b)
  5.         **return** a;
  6.     **else**
  7.         **return** b;
  8.     }
  9.     x = topVariable(a,b);
  10. **if**(a is terminal){
  11.     t = Max(a,bx,n-1);
  12.     e = Max(a,bx',n-1);
  13.     **if**(t equals e)
  14.         result = t;
  15.     **else**
  16.         result = findOrAddUniqueTable(x,t,e);//唯一表中有找出，没有就计算插入
  17.     **insert** ComputedTable({a,b},n);//cache 中插入 a,b,max 的条目
-

续

```

18.     return result;
19. }
20. if(b is terminal){
21.     t = Max(ax,b);
22.     e = Max(ax',b);
23.     if(t equals e)
24.         result =t;
25.     else
26.         result = findOrAddUniqueTable(x,t,e);
27.     insert ComputedTable({a,b},result);
28.     return result;
29. }
30. t = Max(ax,bx);
31. e = Max(ax',bx');
32. if(t equals e)
33.     result =t;
34. else
35.     result = findOrAddUniqueTable(x,t,e);
36. insert ComputedTable({a,b},result);
37. return result;
38. }

```

拷贝？

**算法 3.2** 矩阵向量 $\boxtimes$ 运算伪码

输入：a,b 是用 BDD 表示的矩阵，n 表示的 a, b 矩阵用 BDD 表示使用的变量数

输出：result 是对矩阵 a,b 做 $\boxtimes$ 操作后的结果

```

function Max_matrix_vector (a,b,n) {
1.  if((result = Lookup(a,b,n, Max_matrix) )  $\neq$  NULL)
2.      return result;// Max_matrix_vector (a,b,n)的操作已经执行过一次
3.  if(a is terminal && b is terminal){//a,b 在 BDD 表示下只有一个终止节点
4.      return a*b;
5.  }
6.  x = topVariable(a,b);
7.  y = topVariable(ax,ax');
8.  Q1 = Max_matrix_vector(ax'y',bx');
9.  Q2 = Max_matrix_vector(ax'y,bx);
10. Q3 = Max_matrix_vector(axy',bx');
11. Q4 = Max_matrix_vector(axy,bx);
12. R1 = Max(Q1,Q2);
13. R2 = Max(Q3,Q4);
14. Result = newMTBDD(xi,R1,R2);
15. Store(a,b,n,I,MVM,Result);
16. return Result;
17. }

```



由 3.3 节可知, 如果要想实现每个状态  $\Phi_1 U^{\leq n} \Phi_2$  可能性的计算, 需要在 BDD 表示矩阵和向量下实现矩阵与向量的  $\boxtimes$  运算。

矩阵与向量的  $\boxtimes$  运算, 令  $a$  是  $m \times n$  阶矩阵,  $b$  是长度为  $n$  的向量,  $r(x_1, x_2 \dots x_n) = a(x_1 y_1, x_1 y_2 \dots x_1 y_n, \dots x_m y_1, x_m y_2 \dots x_m y_n) \boxtimes b(y_1, y_2 \dots y_n)$ 。

$$\begin{bmatrix} r_{\bar{x}_1} \\ r_{x_1} \end{bmatrix} = \begin{bmatrix} a_{\bar{x}_1 \bar{y}_1} & a_{\bar{x}_1 y_1} \\ a_{x_1 \bar{y}_1} & a_{x_1 y_1} \end{bmatrix} \boxtimes \begin{bmatrix} b_{\bar{y}_1} \\ b_{y_1} \end{bmatrix}$$

$r_{\bar{x}_1} = \text{Max}(a_{\bar{x}_1 \bar{y}_1} \boxtimes b_{\bar{y}_1}, a_{\bar{x}_1 y_1} \boxtimes b_{y_1})$ ,  $r_{x_1} = \text{Max}(a_{x_1 \bar{y}_1} \boxtimes b_{\bar{y}_1}, a_{x_1 y_1} \boxtimes b_{y_1})$ 。首先看个例子。

$$\begin{bmatrix} 0 & 2 & 0 & 5 \\ 3 & 0 & 0 & 5 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 3 & 0 \end{bmatrix} \boxtimes \begin{bmatrix} 1 \\ 2 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 1 \end{bmatrix}$$

$$a_{\bar{x}_1 \bar{y}_1} = \begin{bmatrix} 0 & 2 \\ 3 & 0 \end{bmatrix}, a_{\bar{x}_1 y_1} = \begin{bmatrix} 0 & 5 \\ 0 & 5 \end{bmatrix}, a_{x_1 \bar{y}_1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, a_{x_1 y_1} = \begin{bmatrix} 0 & 5 \\ 3 & 0 \end{bmatrix},$$

$$b_{\bar{y}_1} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, b_{y_1} = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

$$a_{\bar{x}_1 \bar{y}_1} \boxtimes b_{\bar{y}_1} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, a_{\bar{x}_1 y_1} \boxtimes b_{y_1} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, a_{x_1 \bar{y}_1} \boxtimes b_{\bar{y}_1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, a_{x_1 y_1} \boxtimes b_{y_1} = \begin{bmatrix} 2 \\ 1 \end{bmatrix},$$

$$r_{\bar{x}_1} = \text{Max}(a_{\bar{x}_1 \bar{y}_1} \boxtimes b_{\bar{y}_1}, a_{\bar{x}_1 y_1} \boxtimes b_{y_1}) = \begin{bmatrix} 2 \\ 2 \end{bmatrix},$$

$$r_{x_1} = \text{Max}(a_{x_1 \bar{y}_1} \boxtimes b_{\bar{y}_1}, a_{x_1 y_1} \boxtimes b_{y_1}) = \begin{bmatrix} 2 \\ 1 \end{bmatrix},$$

$$r = \begin{bmatrix} r_{\bar{x}_1} \\ r_{x_1} \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 1 \end{bmatrix}.$$

### 算法 3.2 矩阵向量 $\boxtimes$ 运算伪码

输入:  $a, b$  是用 BDD 表示的矩阵,  $n$  表示的  $a, b$  矩阵用 BDD 表示使用的变量数

输出: result 是对矩阵  $a, b$  做  $\boxtimes$  操作后的结果

**function** M\_matrix\_vector ( $a, b, n$ ) {

1. **if**((result = Lookup( $a, b, n$ , M\_matrix))  $\neq$  NULL)
2.     **return** result; // M\_matrix\_vector ( $a, b, n$ ) 的操作已经执行过一次
3. **if**( $a$  is terminal &&  $b$  is terminal) { //  $a, b$  在 BDD 表示下只有一个终止节点
4.     **if**( $a > b$ )
5.         **return**  $b$ ;
6.     **else**

---

```

7.   return a;
8. }
9.  x = topVariable(a,b);
10. y = topVariable(ax,ax');
11. Q1 = M_matrix_vector(ax'y',bx');
12. Q2 = M_matrix_vector(ax'y,bx);
13. Q3 = M_matrix_vector(axy',bx');
14. Q4 = M_matrix_vector(axy,bx);
15. R1 = Max(Q1,Q2);
16. R2 = Max(Q3,Q4);
17. Result = newMTBDD(xi,R1,R2);
18. Store(a,b,n,I,MMV,Result);
19. return Result;
20. }

```

---

### 3.4 与 MDP 对比

PRISM 中实现了 MDP 的模型检测，MDP 是 DTMC 外加上考虑非确定性的一种模型。因为现实中的系统的有些方面无法用概率描述的，因此不该被建模成概率模型，如下四种情形中系统的属性就不是完全可以用概率刻画的，而包含了不确定的因素：在平等组件调度时，每个组件被调度的可能性是相同的，即并发的，实例有随机分布算法中异步多概率进程操作；在未知环境中，将要发生的事件是不可用概率刻画的，实例有概率安全协议的未知调度算法；在模型参数未知时，传入的参数是区间等可能的，非确定的，实例有为表示信息传播延迟设计的概率通信协议。为了将概率与非确定性结合在一起来表示系统，PRISM 选择了 MDP 模型。

MDP (Markov decision processes) 是 DTMC 的在非确定性上的一种扩展，与 DTMC 类似，MDP 中用离散的状态集合表示建模的可能的配置，状态之间的变迁在离散时间内发生；同时它也包括非确定性，在每个状态都要从后续状态的离散的概率分布中作出一非确定性的选择。

MDP 形式化的表示成一个四元组， $(S, s_{init}, Steps, L)$ ，其中：

$S$  是有限状态集合

$s_{init} \in S$  是初始状态

$Steps: S \rightarrow 2^{Act \times Dist(S)}$  是状态概率函数，其中  $Act$  是动作集合， $Dist(S)$  是状态集合  $S$  上的离散概率分布

$L: S \rightarrow 2^{AP}$  是状态的用原子命题打标函数

接下来给出一个 MDP 实例，通过实例来解释 MDP 是如何将概率与非确定性

融合在一个模型中的。

图 3-2 中的实例，进程从第一步通过概率为 1 的变迁进入第二步，进程在第二步需要做一个非确定性选择：a.以 0.3 的概率在本状态上等待，以 0.7 的概率回到初始状态；b.以 0.5 的概率到状态 3，然后在状态 3 上不停自环，以 0.5 的概率到状态 2，然后在状态 2 上不停自环。

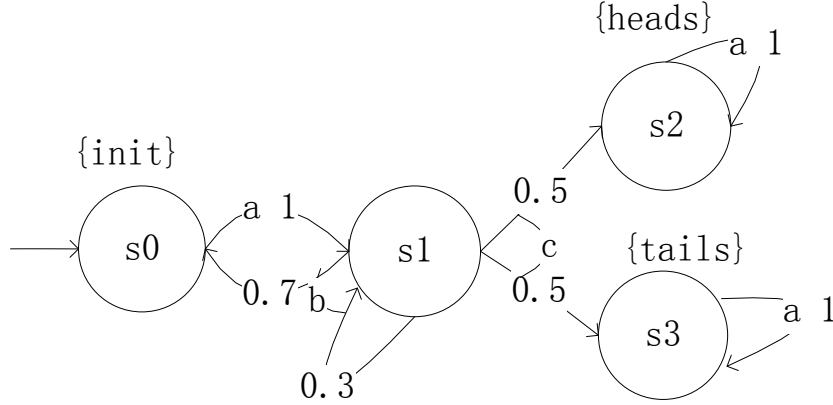


图 3-2 MDP 实例

其中图 3-2 中的实例的 MDP 中的四元组的具体值如下：

$M = \{S, s_{init}, Steps, L\}$ ,  $S = \{s_0, s_1, s_2, s_3\}$ ,  $s_{init} = s_0$ ,  $AP = \{init, heads, tails\}$ ,  $L(s_0) = \{init\}$ ,  $L(s_1) = \emptyset$ ,  $L(s_2) = \{heads\}$ ,  $L(s_3) = \{tails\}$ ,  $Steps(s_0) = \{(a, [s_1 \rightarrow 1])\}$ ,  $Steps(s_1) = \{(b, [s_0 \rightarrow 0.7, s_1 \rightarrow 0.3]), (c, [s_2 \rightarrow 0.5, s_3 \rightarrow 0.5])\}$ ,  $Steps(s_2) = \{(a, [s_2 \rightarrow 1])\}$ ,  $Steps(s_3) = \{(a, [s_3 \rightarrow 1])\}$

$$Steps = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.7 & 0.3 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

MDP 中一条有限或者无限的路径：是状态和动作/概率分布组，eg.  $s_0(a_0, \mu_0)s_1(a_1, \mu_1)s_2 \dots$ ，其中  $(a_i, \mu_i) \in Steps(s_i)$ ，代表了 MDP 建模的系统的一个执行。MDP 中的概率行为是通过调度来实现的，首先需要解决非确定性的选择，确定了之后就生成一个 DTMC。一个调度形式化的定义是将每条有限路径  $\omega = s_0(a_0, \mu_0)s_1(a_1, \mu_1) \dots s_n$  映射到  $Steps(s_n)$  的元素  $\sigma(\omega)$  上。这样便基于执行历史解决了非确定性。

接下来我们看看图 3-2 的两个调度实例：调度  $\sigma_1(s_0s_1) = (c, \mu_c)$ ，代表第一次选取的是动作 c，见图 3-3；调度  $\sigma_2(s_0s_1) = (b, \mu_b)$ ， $\sigma_2(s_0s_1s_1) = (c, \mu_c)$ ， $\sigma_2(s_0s_1s_0s_1) = (c, \mu_c)$  代表第一次选取的是动作 b 然后是 c，见图 3-4。

$\text{Path}^\sigma(s) \subseteq \text{Path}(s)$ , 从  $s$  出发的非确定性由  $\sigma$  确定的路径,  $\text{Path}(s) = s_0(a_0, \mu_0)s_1(a_1, \mu_1)s_2 \dots$ ,  $\sigma(s_0(a_0, \mu_0)s_1 \dots s_n) = (a_n, \mu_n)$ ; 如图 3-3 中的  $\text{Path}^{\sigma_1}(s_0)$  和  $\text{Path}^{\sigma_2}(s_0)$ ,  $\text{Path}^{\sigma_1}(s_0) = \{s_0s_1s_2^\omega, s_0s_1s_3^\omega\}$ ,  $\text{Path}^{\sigma_2}(s_0) = \{s_0s_1s_0s_1s_2^\omega, s_0s_1s_0s_1s_3^\omega, s_0s_1s_1s_2^\omega, s_0s_1s_1s_3^\omega\}$ 。

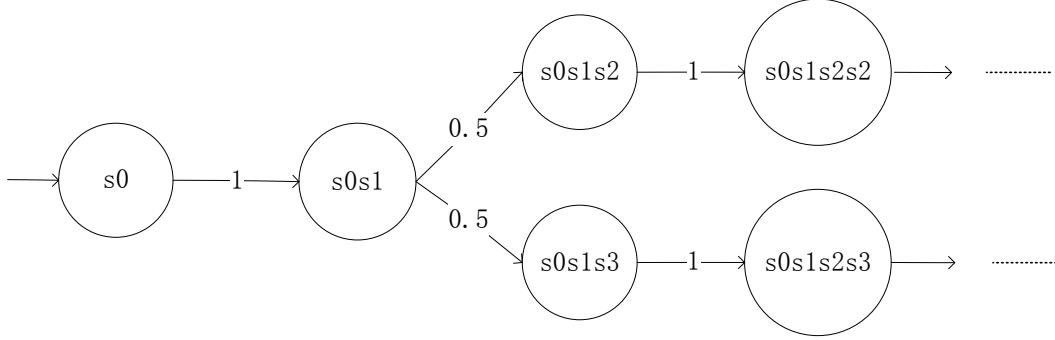


图 3-3 MDP 实例在  $\sigma_1$  下的 DTMC  $D^{\sigma_1}$

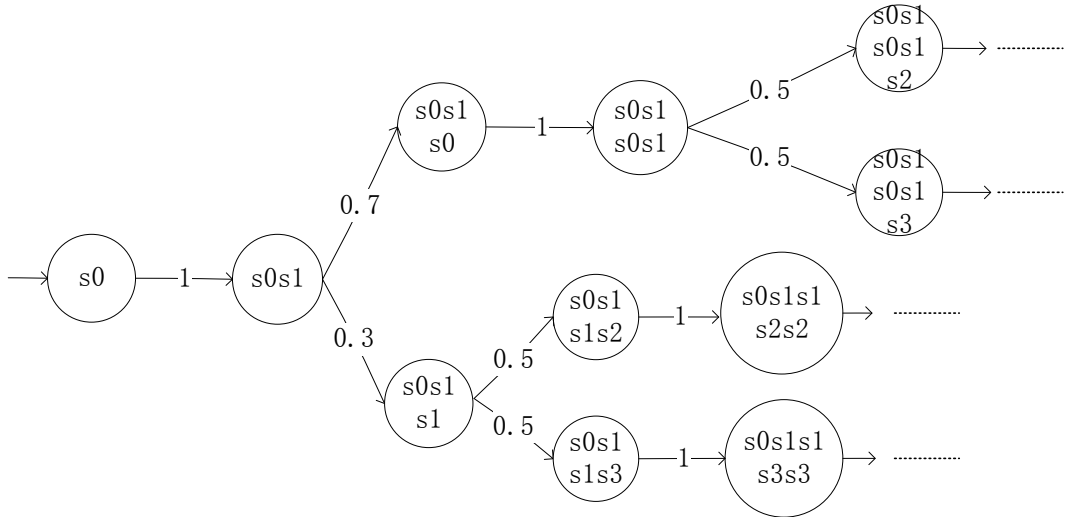


图 3-4 MDP 实例在  $\sigma_2$  下的 DTMC  $D^{\sigma_2}$

MDP 的调度可以推导出一个无限状态的 DTMC  $D^\sigma$ ,  $D^\sigma = (\text{Path}^\sigma_{\text{fin}}(s), s, P_s^\sigma)$ , 其中: DTMC 的状态是从状态  $s$  出发的有限路径; 初始状态为  $s$  (从  $s$  出发的路径长度为 0 的路径);  $P_s^\sigma(\omega, \omega') = \mu(s')$ , 如果  $\omega' = \omega(a, \mu)s'$  和  $\sigma(\omega) = (a, \mu)$ ; 否则,  $P_s^\sigma(\omega, \omega') = 0$ 。

数学中的非确定性有两种含意, 一种是随机性, 另一种是模糊性。其中随机性指的是, 某件事发生的后果不只一种, 但是事件没发生时, 结果是哪个是未知的, 尽管不同结果的概率是已知的, 可以用概率刻画的非确定性的前提是同一事件在严格的相同的环境下进行重复的实验时, 而这个设定与现实系统的不确定性不是同一种不确定; 现实中的不确定性还有一部分是人类语言的模糊性造成的;

而模型集合及其相关理论，如模糊集合、模糊逻辑、可能性等概念正是为了描述这种类型的不确定性而提出的；模糊性的含义是指事件发生本身定义具有模糊性，如经典集合与模型集合的区别，在模糊集合中，每个元素以一定的隶属度属于某个集合，因此每个模糊集合对应一个隶属函数用来表示集合中的元素的隶属度，如温度适宜这个集合，其对应的温度从 20 度到 30 度，对应的隶属度是  $\{0.3, 0.5, 0.5, 0.7, 1, 1, 0.7, 0.5, 0.5, 0.3\}$ ，即 20 度属于温度适宜的隶属度是 0.3。可能性可以定义在模糊集合上，其与模糊集合的关联实例描述如下：比如  $u$  代表年纪， $u = 28$  属于年轻集合的隶属度是 0.7，可以把隶属度 0.7 解读成 28 岁与年轻这个集合的匹配度，同时“28 岁与年轻的匹配度是 0.7”也可解读为“一个年轻人是 28 岁的可能性是 0.7”，简短来说  $u$  的具体值与年轻的匹配度被理解成了已知年轻人的基础上， $u$  是某个值的可能性，即某个变量是特定具体值的可能性与该具体值属于的对应的模糊集合的隶属度是相同的。因此对于模糊集合或者对应的可能性来说，隶属函数的计算是最重要的，隶属函数通常有三种确定方法，分别是模糊统计法、指派法和借用已有尺度（如恩格尔系数）。其中模糊统计法适合涉及不同人有不同定义的模糊集合，如年轻、温度适宜等人类的模糊描述性集合，指派法主要考虑的人们已有的工作经验。

但是 MDP 中的非确定性即不是随机性，所谓的随机性是可以概率刻画的，也不是模糊性的，所谓的模糊性是可以可能性刻画的，同时 MDP 中的非确定性既可以是概率的，也可以是可能性的，却不能用这两种来刻画。MDP 中的非确定性与 NFA 中的非确定性是同一种，如图 3-3 的例子，不到达状态  $s_1$  时，就不知道哪个动作会被选择，只有到达了才能知道，而且动作  $b$  和动作  $c$  的变迁不在同一个评判级别上。MDP 的模型检测过程与 NFA 有异曲同工之处，NFA 是先转换成相应的 DFA，再进行检测；MDP 是通过调度确定每次非确定性时的选择后，转换成相应的 DTMC，再进行检测，一个 MDP 模型通过不同的调度可以转换成多个相应的 DTMC。但是 MDP 转换成 DTMC 会增加很多状态，是个非常复杂的过程。

可能性模型、DTMC 与 MDP 分别表示了现实中不同的非确定性，可能性模型与 DTMC 分别描述的是在每个状态在同一个评判标准下系统的模糊性和随机性，其变迁上的值是一个估计，可能性模型检测上的变迁的值是对模糊性的描述，DTMC 上的变迁的值是对随机性的描述，而 MDP 中增加的非确定性要表示的是在某个状态过后有不同评判标准下的几种情况，但是每次不到该状态就不知道下一个变迁是在哪个评判标准下。

可能性模型上也可以加上 MDP 中考虑的这种非确定性，MDP 中考虑的这种非确定性在现实生活中也是很常见的。比如：并发系统中，不到运行时，就不知

道并发系统中哪一个进程先走，而且不论运行该并发系统多少次，都无法准确预测下次是哪个进程先运行，尽管可以统计哪个进程先走的概率，却无法根据概率，准确预测下次具体哪个进程先运行。

### 3.5 本章小结

第 3 章主要描述的是可能性模型检测的具体实现过程，可能性模型检测的实现是在 PRISM 的基础上实现的，鉴于其与概率模型检测的相似性，因此需要与概率模型检测作对比分析，从而挖掘出其具体实现细节（3.1 节）；因此概率模型检测工具 Prism 的实现细节及两者的对比成为研究重点及难点（3.2 节）；随后在 3.3 节中，分析并给出了  $X\varphi$ ,  $\Phi_1 U^{\leq n} \Phi_2$ ,  $\Phi_1 \cup \Phi_2$  三个时序逻辑的可能性计算过程；在 3.4 节中，根据 3.3 节中计算的需要，定义了一些矩阵运算。3.5 节则给出了可能性模型与同是描述不确定性的模型 MDP 的对比。

## 第4章 可能性认知逻辑

本文的一个主要工作是研究可能性与模态逻辑中的一种认知逻辑的结合。认知逻辑表示的是解释系统的属性。解释系统使用了可能世界和可达世界概念，代理的本地状态和系统全局状态能够捕获知识逻辑哲学的基础。简单的说，解释系统中代理的知识是存储在当前状态的本地等价状态中。因此，代理知道  $u$  在一个给定的状态中，当且仅当  $u$  在所有的等价状态都为真（这些状态时可能的或可达到的）。为了检查解释系表示不确定性的定量认知性质有两个必须回答的问题：，如何指定可测量的认知性质和如何表示捕捉可衡量的认知特征的模型。本文结合可能性模型和解释系统来表达可能性多智能体系统，指定这些系统的定量性质，提出了可能性-认知逻辑。

### 4.1 普通可能性认知逻辑

#### 4.1.1 可能性多智能体系统

解释系统定义如下：假设  $A = \{1, \dots, n\}$  是系统中的  $n$  个代理，每个代理  $i \in A$  都有自己的局部状态集合  $L_i$  和可能的动作集合  $Act_i$ 。全局状态集  $\Gamma \in L_1 \times \dots \times L_n$  是  $n$  元组  $(l_1, \dots, l_n)$  的集合， $(l_1, \dots, l_n)$  代表的是整个系统的状态。假设所有的动作的机会平等，那么可以把动作映射到系统可能变迁函数  $T$  上。 $T: \Gamma \times Act \times \Gamma \rightarrow [0,1]$ ， $Act \in Act_1 \times \dots \times Act_n$ ，指的是系统中各个代理共同协作完成的动作，对于一个全局状态  $\gamma \in \Gamma$ ， $\forall \gamma' \in \Gamma T(\gamma, \alpha^{\gamma\gamma'}, \gamma') = 1$ ， $\alpha^{\gamma\gamma'} \in Act$ ，是状态  $\gamma$  到  $\gamma'$  的变迁上标记的动作。每个代理的本地可能性变迁函数  $T_i: L_i \times Act_i \times L_i \rightarrow [0,1]$ ，对于每个代理  $i \in A$ ，该代理的每个本地状态  $l_i \in L_i$ ，有  $\forall l_i' \in L_i T_i(l_i, \alpha^{l_i l_i'}, l_i') = 1$ ， $\alpha^{l_i l_i'} \in Act_i$ ，是代理  $i$  的本地状态  $l_i$  到  $l_i'$  的变迁上标记的动作。对于  $l_i \in \gamma$ ， $l_i' \in \gamma'$ ，系统的可能性变迁函数  $T$  定义如下：

$$T(\gamma, \alpha^{\gamma\gamma'}, \gamma') = \eta \bigwedge_{i \in A \wedge l_i \in \gamma \wedge l_i' \in \gamma'} T_i(l_i, \alpha^{l_i l_i'}, l_i')$$

其中  $\eta$  是归一化因子，对于每一个全局状态  $\gamma$ ，使得  $\gamma$  出发的可能性变迁满足可能性分布  $\sum_{\gamma' \in \Gamma} t(\gamma, \alpha^{\gamma\gamma'}, \gamma') = 1$ 。一个全局初始分布  $I_{init}$  表示了系统的开始状态，满足  $\sum_{\gamma \in \Gamma} I_{init}(\gamma) = 1$ 。定义 4.1 给出了可能性多智能体系统形式化定义：

定义 4.1: ( $M_{PoIS}$  —— 可能性多智能体系统形式化表示) 在原子命题  $AP$  集上，

$M_{\text{PoIS}} = (W, R, I_{\text{init}}, \sim_1, \dots, \sim_n, V)$ ,  $W \subseteq \Gamma$  是系统的可达集合。

一个状态  $w$  是可达的当且仅当存在从初始状态出发的到达  $w$  的可能性变迁路径, 且这条路径上的变迁上的可能性大于 0。

$I_{\text{init}}: W \rightarrow [0,1]$ , 是模型的初始分布,  $\forall w \in W, I_{\text{init}}(w) = 1$ 。

$R: W \times W \rightarrow [0,1]$  是由  $R(w, w') = j (j \in [0,1])$  定义的模糊变迁函数当且仅当存在一个联动  $(a_1, \dots, a_n) \in \text{Act}$ , 使得  $\forall i \in A, T_i(w, a_i, w') > 0$  和  $R = t(w, \alpha^{ww'}, w')$ 。对于所有的  $w \in W$ , 有  $\sum_{w' \in W} R(w, w') = 1$ 。

$\sim_i: \sim_i \subseteq W \times W$  是代理  $i$  的认知可达关系, 对于两个全局状态  $(l_1, \dots, l_n)$  和  $(l'_1, \dots, l'_n)$ ,  $(l_1, \dots, l_n) \sim_i (l'_1, \dots, l'_n)$  当且仅当  $l_i = l'_i$ 。

$V$ : 是全局的状态打标函数  $V: W \rightarrow 2^{AP}$ 。

初始状态分布  $I_{\text{init}}$  可以看成有  $|W|$  行的列向量,  $|W|$  是集合  $W(I_{\text{init}}(w))_{w \in W}$  的元素个数, 列向量中的每行的值表示的是对应状态的初始的可能性。

变迁模糊函数  $R: W \times W \rightarrow [0,1]$  可以表示成矩阵  $(R(s, t))_{s, t \in W}$ 。从一个状态到它的后继状态的可能性在  $R(s, \cdot)$  中表示, 而到达一个状态的可能性在  $R(\cdot, s)$  中表示, 同。

#### 4.1.2 可能性认知逻辑语法

本文定义了 PoCTLK (Possibilistic Computation Tree Logic of Knowledge), PoCTLK 把基于可能性测度的 CTL 与认知逻辑进行了组合。PoCTLK 可以用来描述可能性 CTL 和多智能体系统的认知逻辑。PoCTLK 由三种公式组成: 状态公式  $\phi$ , 路径公式  $\psi$ , 认知公式  $k$ 。PoCTLK 路径公式与 PoCTL 完全相同, 状态公式比其多了一个, 认知公式包括知识和组知识操作符。其语法如下定义:

定义 4.2: (PoCTLK 语法) 设  $p, p_1, p_2 \dots$  是原子命题的集合  $\Phi_P$  中元素,  $A = \{1, \dots, n\}$  是代理集合,  $G \subseteq A$  是一组代理, PoCTLK 公式 BNF 语法定义如下:

- $\phi ::= \text{true} | p | \neg \phi | \phi \wedge \phi | k | PO_{\nabla J}(\psi)$
- $\psi ::= \circ \phi | \phi_1 U^{\leq n} \phi_2 | \phi_1 U \phi_2$
- $k ::= K_i \phi | E_G \phi | C_G \phi | D_G \phi$

其中  $\nabla \in \{<, \leq, >, \geq\}$ ,  $J \in [0,1]$ ,  $n \in \mathbb{N}$ 。认知公式  $k$  是 PoCTLK 中特殊的状态公式, 它可以描述认知属性。有四个认知模态:  $K_i, E_G, C_G, D_G$ , 分别代表的是“代理  $i$  知道”, “ $G$  组中的每个代理知道”, “ $G$  组中所有代理的常识”, “ $G$  组中代理的分布式知识”。假设  $C_G \emptyset$ , 则对于  $G$  中的任意代理  $a \in G$ ,  $K_a \emptyset$ , 且对于  $G$  中的任意代理  $b \in G$ , 有  $K_a K_b \emptyset, K_a \dots K_b \emptyset$ 。 $K_i \text{Pr}_{\nabla J} \psi$  表示的是代理  $i$  知道路径公式  $\psi$  的可能性是  $\nabla b$ 。 $\nabla b$  表示的是可能性的上限或下限 (e.g.  $< b, \leq b, \geq b, > b$ )。例如



$K_1Pr_{\geq 0.9}\phi$ 意思是代理 1 知道公式 $\phi$ 的可能性是至少 0.9。

同 PoCTL 一样, PoCTLK 中没有路径量词。但是其中的线性时序操作符 $\circ(\text{next})$ ,  $U(\text{until})$ , 和 $U^{\leq n}$  (bounded until) 必须跟在可能性操作符 $PO_{vj}$ 之后。PoCTLK 中的时序部分同 CTL 中一样, 如: 公式 $\phi$ 意味着“下一个状态满足 $\phi$ ”,  $\phi_1 U \phi_2$ 即“ $\phi_1$ 在路径上一直满足, 直到 $\phi_2$ 出现”,  $\phi_1 U^{\leq n} \phi_2$ 的含义是“ $\phi_2$ 满足之前至多有  $n$  步 $\phi_1$ 是满足的”。

#### 4.1.3 基于可能性解释系统的可能性认知逻辑语义

PoCTLK 的公式中与 PoCTL 公式中重合的部分, 其语义没有发生变化, 其重合部分包括 PoCTL 的路径公式和状态公式中除去认知公式 $k$ 之外的部分, 因此本文不再赘述, 此处只给出认知逻辑 $M_{PoIS}$ 的语义。

令  $G \subseteq A$  是一组代理, 认知操作符的语义 $E_G, C_G, D_G$ , 在认知可达变迁 $\sim_i$ 的基础上给出了组认知可达变迁定义:

定义 4.4:  $\sim_i$ : (组认知可达变迁)

- $\sim_G^E$  小组  $G$  中各个代理可达变迁的并:  $\sim_G^E = \bigcup_{i \in G} \sim_i$
- $\sim_G^C$  是  $\sim_G^E$  的传递闭包
- $\sim_G^D$  小组  $G$  中各个代理可达变迁的交:  $\sim_G^D = \bigcap_{i \in G} \sim_i$

定义 3.5: (认知公式在 $M_{PoIS}$ 上的语义) 假设有一个 $M_{PoIS}$ ,  $\phi$ 为 $M_{PoIS}$ 上的状态公式, 则认知公式 $k$ 在 $M_{PoIS}$ 上的语义如下:

- $s \models K_i \phi$  当且仅当  $\forall s' \in W$ , 如果  $s \sim_i s'$  那么  $s' \models \phi$
- $s \models E_G \phi$  当且仅当  $\forall s' \in W$ , 如果  $s \sim_G^E s'$  那么  $s' \models \phi$
- $s \models C_G \phi$  当且仅当  $\forall s' \in W$ , 如果  $s \sim_G^C s'$  那么  $s' \models \phi$
- $s \models D_G \phi$  当且仅当  $\forall s' \in W$ , 如果  $s \sim_G^D s'$  那么  $s' \models \phi$

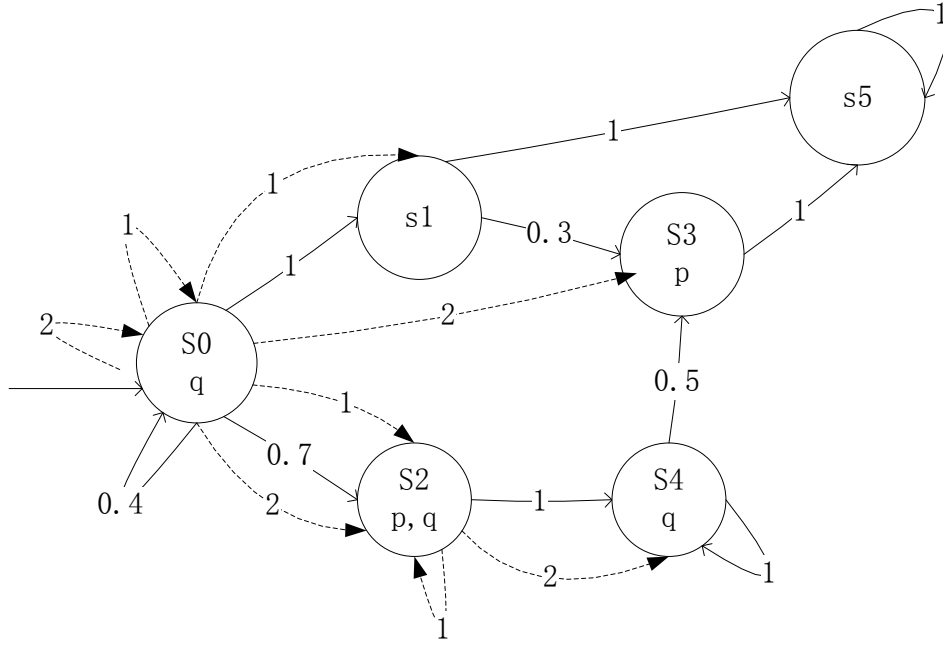
#### 4.1.4 实例说明

图 3-1 是一个简单的 $M_{PoIS}$ 的例子。其中,  $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$ ,  $I =$

$$\{1, 0, 0, 0, 0, 0\}, \quad P = \begin{bmatrix} 0.4 & 1 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\sim_1 = \{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_2, s_2)\}, \sim_2 = \{(s_0, s_0), (s_0, s_3), (s_0, s_2), (s_2, s_4)\}$$

$$V(s_0) = \{q\}, V(s_1) = V(s_5) = \{\}, V(s_2) = \{p, q\}, V(s_3) = \{p\}, V(s_4) = \{q\}。$$


 图 3-1  $M_{PoIS}$ 模型

根据定义 4.3 计算, 得到  $\sim_G^E$ ,  $\sim_G^C$ ,  $\sim_G^D$  的集合如下:

$$\begin{aligned} \{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_0, s_3)\} &\subseteq \sim_G^E, \\ \{(s_0, s_0), (s_0, s_1), (s_0, s_2), (s_0, s_3), (s_0, s_4)\} &\subseteq \sim_G^C, \\ \{(s_0, s_0), (s_0, s_2)\} &\subseteq \sim_G^D. \end{aligned}$$

## 4.2 深层可能性认知逻辑

### 4.2.1 深层可能性认知逻辑语法

本文定义了两种 DPoCTLK (Deep Possibilistic Computation Tree Logic of Knowledge), 第一种如 4.1.2 节定义, 本节定义了深层可能性认知逻辑, 其语法如下定义:

定义 4.6: (DPoCTLK 语法) 设  $p, p_1, p_2 \dots$  是原子命题的集合  $\Phi_P$  中元素,  $A = \{1, \dots, n\}$  是代理集合,  $G \subseteq A$  是一组代理, PoCTLK 公式 BNF 语法定义如下:

- $\phi ::= true | p | \neg \phi | \phi \wedge \phi | k | PO_{\nabla J}(\psi) | PO_{\nabla J}(k)$
- $\psi ::= \circ \phi | \phi_1 U^{\leq n} \phi_2 | \phi_1 U \phi_2$
- $k ::= K_i \phi | E_G \phi | C_G \phi | D_G \phi$

同 4.1.2 节定义的 PoCTLK 相比, DPoCTLK 的状态公式多了一个  $PO_{\nabla J}(k)$  操作符, 即认知公式可以嵌套在可能性测度里面,  $Pr_{\nabla J}(K_i \phi)$  代表的是代理  $i$  知道公式  $\phi$  的可能性是  $\nabla J$ , 其中  $\phi$  是状态公式。  $K_i Pr_{\nabla J} \psi$  表示的是代理  $i$  知道路径公式  $\psi$  的可能性

是 $\nabla b$ 。 $\nabla b$ 表示的是可能性的上限或下限(e.g.  $< b, \leq b, \geq b, > b$ )。这两个公式的区别是 $\text{Pr}_{\nabla j}(K_i \phi)$ 表示的是代理  $i$  知道某个事情的可能性的程度，而 $K_i \text{Pr}_{\nabla j} \psi$ 指的是代理  $i$  知道一件不确定的事。例如 $\text{PO}_{\geq 0.9}(K_1 \phi)$ 表示代理知道公式 $\phi$ 这个事情的可能性至少达到 0.9， $K_1 \text{Pr}_{\geq 0.9} \phi$ 意思是代理 1 知道公式 $\phi$ 的可能性是至少 0.9。在认知公式上的可能性操作符 $\text{PO}_{\nabla j}(k)$ 表示了认知公式的可能性测度：代理对于该认知公式有多少信心认为其是认知可能的。比如： $\text{Pr}_{\leq 0.8}(K_1(\text{agent\_2\_has\_resource\_A}))$ 描述了代理 1 最多有 0.8 的可能性知道代理 2 拥有资源 A。

#### 4.2.2 基于可能性解释系统的深层可能性认知逻辑语义

DPoCTLK 的公式中与 PoCTLK 公式中重合的部分，其语义没有发生变化，此处只给出 $\text{PO}_{\nabla j}(k)$ 在 $M_{\text{PoIS}}$ 的语义。

定义 4.7: (认知公式在 $M_{\text{PoIS}}$ 上的语义) 假设有一个 $M_{\text{PoIS}}$ ， $\phi$ 为 $M_{\text{PoIS}}$ 上的状态公式，则认知公式 $k$ 在 $M_{\text{PoIS}}$ 上的语义如下：

- $s \models \text{PO}_{\nabla b}(K_i \phi)$ , 当且仅当  $\text{Po}(s \models K_i \phi) \nabla b$ , 其中:  
 $\text{Po}(s \models K_i \phi) = \bigvee_{s \downarrow_i s'} P(s' \models \phi)$ ;
- $s \models \text{PO}_{\nabla b}(E_G \phi)$ , 当且仅当  $\text{Po}(s \models E_G \phi) \nabla b$ , 其中:  
 $\text{Po}(s \models E_G \phi) = \bigvee_{s \sim_G^E s'} P(s' \models \phi)$ ;
- $s \models \text{PO}_{\nabla b}(C_G \phi)$ , 当且仅当  $\text{Po}(s \models C_G \phi) \nabla b$ , 其中:  
 $\text{Po}(s \models C_G \phi) = \bigvee_{s \sim_G^C s'} P(s' \models \phi)$ ;
- $s \models \text{PO}_{\nabla b}(D_G \phi)$ , 当且仅当  $\text{Po}(s \models D_G \phi) \nabla b$ , 其中:  
 $\text{Po}(s \models D_G \phi) = \bigvee_{s \sim_G^D s'} P(s' \models \phi)$ ;

其中：

$$P(s \models \phi) = \begin{cases} \text{Po}(s, \text{Paths}(s), \circ \phi_1), & \text{当且仅当 } \phi = \circ \phi_1 \\ \text{Po}(s, \text{Paths}(s), \phi_1 U^{\leq n} \phi_2), & \text{当且仅当 } \phi = \phi_1 U^{\leq n} \phi_2 \\ \text{Po}(s, \text{Paths}(s), \phi_1 U \phi_2), & \text{当且仅当 } \phi = \phi_1 U \phi_2 \\ 1, & \text{当且仅当 } \phi = p \wedge p \in V(s) \\ 0, & \text{当且仅当 } \phi = p \wedge p \notin V(s) \end{cases}$$

#### 4.2.3 实例说明

根据 4.2.2 中对 $\text{PO}_{\nabla j}(k)$ 的定义，利用 4.1.4 中的实例来展示 $\text{PO}_{\nabla j}(k)$ 的计算：

当  $\phi = q$  时， $P(s_0 \models q) = 1$ ， $P(s_1 \models q) = 0$ ， $P(s_2 \models q) = 1$ ， $P(s_3 \models q) = 0$ ， $P(s_4 \models q) = 1$ 。

- $\text{Po}(s_0 \models K_1 q) = \bigvee_{s_0 \sim_1 s'} P(s' \models q) = \bigvee (P(s_0 \models q), P(s_1 \models q), P(s_2 \models q)) = 1$
- $\text{Po}(s_0 \models K_2 q) = \bigvee_{s_0 \sim_2 s'} P(s' \models q) = \bigvee (P(s_0 \models q), P(s_2 \models q), P(s_3 \models q)) = 1$

- $Po(s_0 \models E_G q) = V_{s_0 \sim_G^E s'} P(s' \models q) = V(P(s_0 \models q), P(s_1 \models q), P(s_2 \models q), P(s_3 \models q)) = 1$
- $Po(s_0 \models D_G q) = V_{s_0 \sim_G^D s'} P(s' \models q) = V(P(s_0 \models q), P(s_2 \models q)) = 1$
- $Po(s_0 \models C_G q) = V_{s_0 \sim_G^C s'} P(s' \models q) = V(P(s_0 \models q), P(s_1 \models q), P(s_2 \models q), P(s_3 \models q), P(s_4 \models q)) = 1$

当  $\phi = qU^3p$  时,  $s^{yes} = \text{Sat}(p) = \{s_2, s_3\}$ ,  $s^{no} = S \setminus (\text{Sat}(\phi_1) \cup \text{Sat}(\phi_2)) = \{s_1, s_5\}$ ,  $s^{\text{unknown}} = S \setminus (s^{no} \cup s^{yes}) = \{s_0, s_4\}$ ,  $Po(\phi_1 U^{\leq 0} \phi_2) = [0, 0, 1, 1, 0, 0]$

$$Po(\phi_1 U^{\leq 1} \phi_2) = P' \boxtimes Po(\phi_1 U^{\leq 0} \phi_2) = \begin{bmatrix} 0.4 & 1 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boxtimes \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.3 \\ 0 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}$$

$$Po(\phi_1 U^{\leq 2} \phi_2) = P' \boxtimes Po(\phi_1 U^{\leq 0} \phi_2) = \begin{bmatrix} 0.4 & 1 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boxtimes \begin{bmatrix} 0.7 \\ 0.3 \\ 0 \\ 0 \\ 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0 \\ 0.5 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}$$

$$Po(\phi_1 U^{\leq 3} \phi_2) = P' \boxtimes Po(\phi_1 U^{\leq 0} \phi_2) = \begin{bmatrix} 0.4 & 1 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.3 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \boxtimes \begin{bmatrix} 0.3 \\ 0 \\ 0.5 \\ 0 \\ 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.35 \\ 0 \\ 0.5 \\ 0 \\ 0.5 \\ 0 \end{bmatrix}$$

- $P(s_0 \models \phi) = 0.35, P(s_1 \models \phi) = 0, P(s_2 \models \phi) = 0.5, P(s_3 \models \phi) = 0, P(s_4 \models \phi) = 0.5$
- $Po(s_0 \models K_1 \phi) = V_{s_0 \sim_1 s'} P(s' \models \phi) = V(P(s_0 \models \phi), P(s_1 \models \phi), P(s_2 \models \phi)) = 0.5$
- $Po(s_0 \models K_2 \phi) = V_{s_0 \sim_2 s'} P(s' \models \phi) = V(P(s_0 \models \phi), P(s_2 \models \phi), P(s_3 \models \phi)) = 0.5$
- $Po(s_0 \models E_G \phi) = V_{s_0 \sim_G^E s'} P(s' \models \phi) = V(P(s_0 \models \phi), P(s_1 \models \phi), P(s_2 \models \phi), P(s_3 \models \phi)) = 0.5$
- $Po(s_0 \models D_G \phi) = V_{s_0 \sim_G^D s'} P(s' \models \phi) = V(P(s_0 \models \phi), P(s_2 \models \phi)) = 0.5$
- $Po(s_0 \models C_G \phi) = V_{s_0 \sim_G^C s'} P(s' \models \phi) = V(P(s_0 \models \phi), P(s_1 \models \phi), P(s_2 \models \phi), P(s_3 \models \phi), P(s_4 \models \phi)) = 0.5$

### 4.3 本章小结

本章分析了多智能体系统对不确定性和定量这样的属性需求，4.1节定义了基于分布知识的可能性解释系统为多智能体概率系统建模，PoCTLK的语法，并定义了在该系统下，PoCTLK的直接语义，最后给出了一个实例来描述；5.2节给出了基于分布知识的可能性解释系统为下PoCTLK的深层语义，最后给出了一个实例来描述。

## 第 5 章 案例分析

本章将分析对比 DTMC 概率模型检测实例，可能性模型检测实例及含不确定性的 MDP 概率模型检测实例。

### 5.1 实验环境

操作系统：ubuntu 10.10

集成开发环境：eclipse

PRISM 的版本号：4.2.1

相关软件版本号：java1.7.6

⇒ 验证？

### 5.2 可能性模型与概率模型实例对比

首先通过一个实例来验证本算法的正确性。

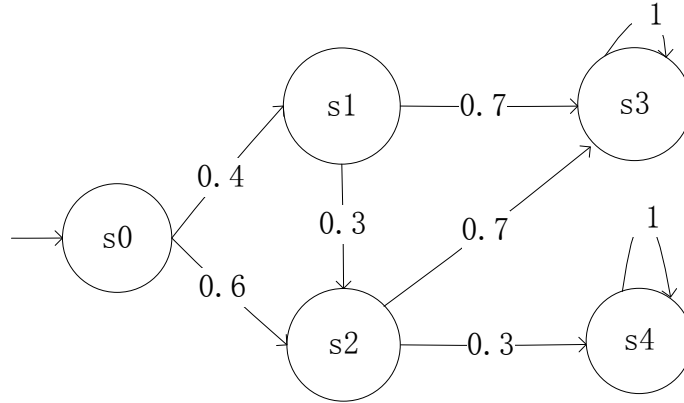


图 5-1 DTMC/可能性实例

图 5-1 验证 DTMC 上的  $P = ? [X(X s3)]$  属性实验结果为 0.7，在验证可能性模型检测下的属性  $Po = ? [X(X s3)] = 0.6$ 。当 5-3 是 DTMC 时：

$$P(s1, X s3) = \sum_{\pi \in \text{Paths}(s1), \pi(1) \models s3} P(s1, \pi(1)) = 0.7;$$

$$P(s2, X s3) = \sum_{\pi \in \text{Paths}(s2), \pi(1) \models s3} P(s2, \pi(1)) = 0.7;$$

$$P(X(X s3)) = \sum_{\pi \in \text{Paths}(s0), \pi(1) \models X s3} P(s0, \pi(1)) = (0.4 * P(s1, X s3)) + (0.6 * P(s2, X s3)) = 0.4 * 0.7 + 0.6 * 0.7 = 0.7;$$

当 5-1 是可能性模型时：

$$Po(s1, X s3) = V_{\pi \in \text{Paths}(s1), \pi(1) \models s3} P(s1, \pi(1)) = 0.7;$$

$$Po(s2, X s3) = V_{\pi \in \text{Paths}(s2), \pi(1) \models s3} P(s2, \pi(1)) = 0.7;$$

$$Po(X(X s3)) = V_{\pi \in \text{Paths}(s0), \pi(1) \models X s3} P(s0, \pi(1)) = V(\wedge(0.4, P(s1, X s3)),$$

$$\Lambda(0.6, P(s2, X s3))) = V(\Lambda(0.4, 0.7), \Lambda(0.6, 0.7)) = 0.6;$$

接着通过一个简单信道传输实例来对比概率模型、可能性模型和 MDP 模型的不同意义。图 5-3 中的 DTMC/可能性信道传输模型实例，进程从状态  $s_0$  开始，通过概率/可能性为 1 动作为 **start** 的变迁进入状态  $s_1$ ，进程在状态  $s_1$  开始试图发信息，状态  $s_1$  分别以概率/可能性为 0.01 在状态  $s_1$  等待，或者以概率/可能性为 0.01 发送失败进入状态  $s_2$ ，状态  $s_2$  上将以概率为 1 的变迁回到  $s_0$ ，协议终止重新开始整个过程；或者以概率/可能性为 0.98 发送成功进入状态  $s_3$ ，状态  $s_3$  上将以概率为 1 的变迁进行自环，协议终止。

DTMC 在  $s_1$  状态下， $s_1$  开始试图发信息，经过多次观察、实验发现有 0.01 的概率，根本没发信息，回到本状态，等待；有 0.01 的概率，发送失败，到  $s_2$ ；有 0.98 的概率，发送成功，到  $s_3$ 。

可能性模型在  $s_1$  状态下， $s_1$  处可能是一个人在做决定，他决定在心情好的时候开始试图发信息，心情好的可能性是 0.98，发送成功，到  $s_3$ ；在心情糟糕的时候在本状态等待，心情糟糕的可能性是 0.01；心情激动的时候发送的信息会因为程序不完整而失败，心情激动的可能性是 0.01。概率无法描述一个人的心情好、坏、激动，这是个模糊概念。

与图 5-2 中的实例略有不同，图 5-3 给出的是带非确定性的 MDP 信道传输实例。当到达状态  $s_1$  后，需要做一个非确定性选择：a.当信道还没准备好时，状态  $s_1$  将选择概率为 1 动作为 **wait** 的变迁在本状态上等待；b.当信道准备好发信息时，状态  $s_1$  将选择动作 **send** 的变迁，动作 **send** 的变迁有两条，分别是概率为 0.99 变迁，将到达发送成功的状态  $s_3$ 。

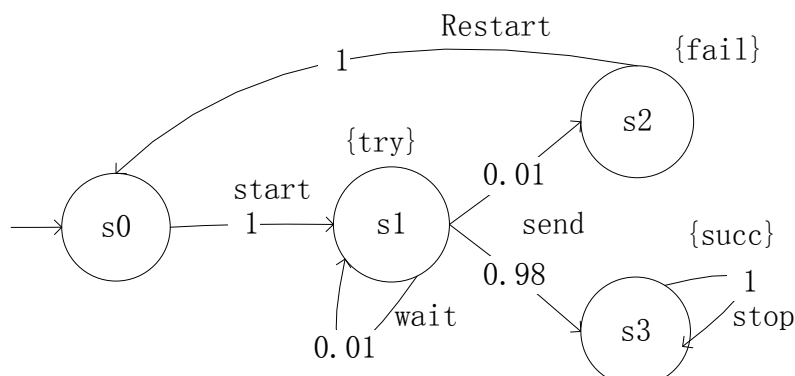


图 5-2 DTMC/可能性信道传输模型实例

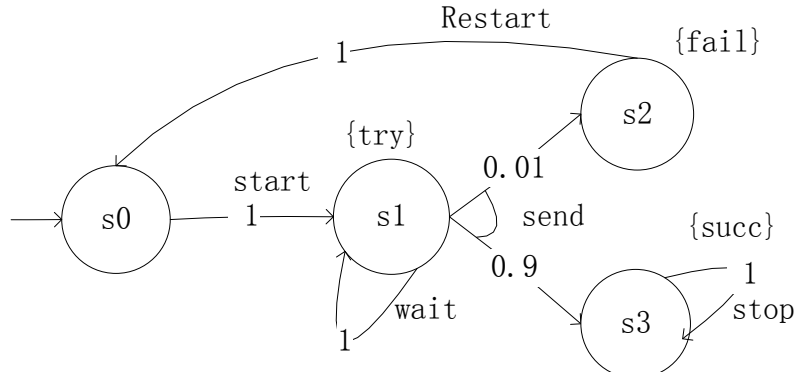


图 5-3 MDP 信道传输实例

### 5.2.1 DTMC 概率模型检测实例

DTMC 概率模型检测实例，本文选择了一个简单的信道传输协议，一个同步领导协议。

同步领导选择协议，在这个协议中， $N$  个进程在一个单向环中选择一个主。为了这个目的，每个进程在  $\{1, \dots, K\}$  上选取一个数字作为标识。通过同步信息传送，进程沿着环发送他们的标识，直到每个进程看到其他进程的标识，这时所有进程就可以决定是否选出领导（领导是那个标识最大的进程）。如果可以选出，那么本协议终止，如果选不出，那么开始下一轮选领导。

每个进程都有三个过程，选标识，查看别的进程的标识，选领导；选标识，就是从  $\{1, \dots, K\}$  中选取一个数字；查看别的进程的标识，即每个进程有了自己的标识后，将自己的标识沿着环传递，所有进程在环上同时沿一个方向传递，这样每过一个时间槽，每个进程就会多知道一个别的进程的标识，直到每个进程知道所有进程的标识，如果在查看别的进程标识过程中，发现自己的进程标识和别的进程标识相同时，则本进程的标识不能够被选为领导；选领导，其实就是在有效的候选进程中，从中选取进程标识最大的；如果没有有效的候选进程，则重复本过程。详细流程见图 5-1 进程  $i$  选取领导流程。

除去每个进程对应的进程外，还需要一个额外的计数进程来管理选举过程，计数进程记录当前进程查看了几个别的进程，当所有进程被查看完时，实际由计数进程来完成选领导的过程。详细流程见图 5-2 计数进程选取领导流程。

要检测的属性是在  $L$  次选择之中选取了领导的概率，每个进程对应的自动机如图所示，其中第  $n$  个进程的状态在对应的 PRISM 语言中被描述成了有四个值的变量  $sn$ ，以第 1 个进程的状态变量  $s1$  为例，变量  $s1=0$  表示的是一个状态，这个状态的任务是选取  $id$ ，变量  $s1=1$  代表的状态负责查看别的进程的  $id$ ， $s1=2$  的状态完成了选择领导， $s1=3$  对应的状态代表的是选出领导，结束，或者没有合适的候选人，结束。假设有 4 个进程，那么对应的 PCTL 属性被描述成： $P = ? [true \ U \leq (N + 1)(s1 = 3 \ \& \ s2 = 3 \ \& \ s3 = 3 \ \& \ s4 = 3)]$ 。



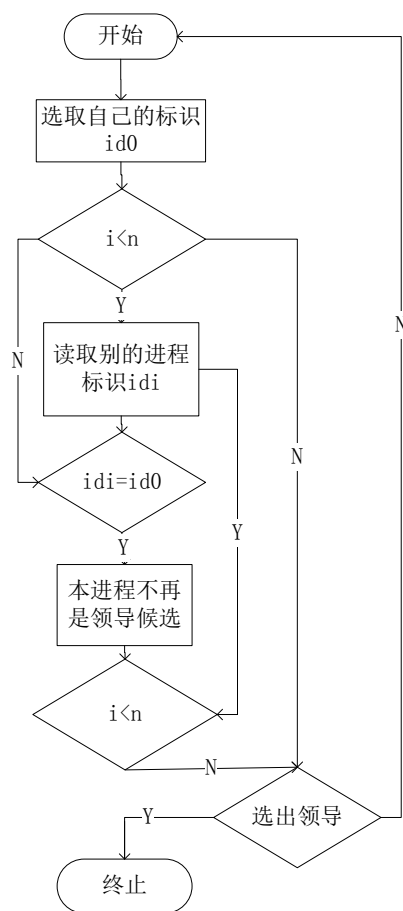


图 5-1 进程  $i$  选取领导流程

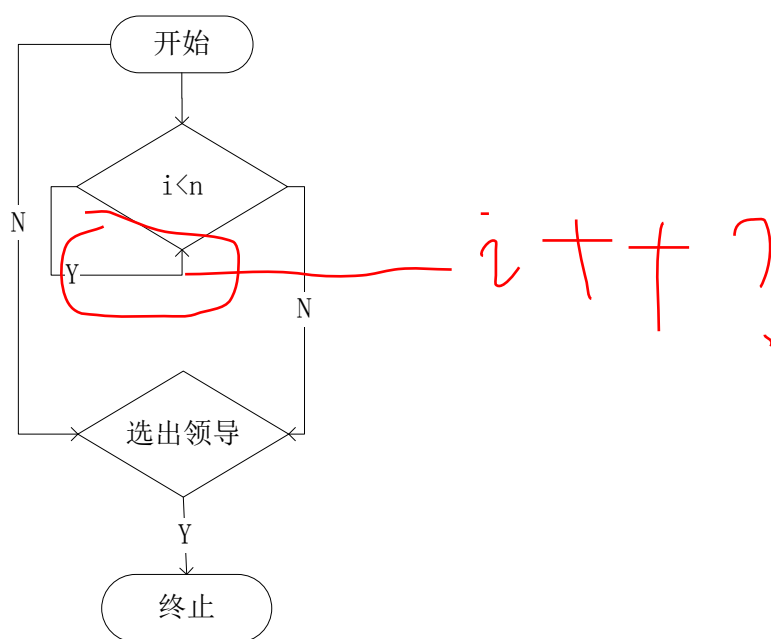


图 5-2 计数进程选取领导流程

### 5.2.2 可能性模型检测实例

可能性模型检测实例，本文选择了两个，一个是室温模糊控制系统；一个是把同步领导选择协议中涉及到概率的部分，用可能性来表示，从而可以看出可能性和概率的区别。模糊控制系统包括三个阶段，输入，处理，输出。输入通常会把实时数据监控的传感器等其他设备的输入映射成隶属函数，处理阶段调用了合理的模糊推断规则，最后输出阶段把组合结果映射成特定的控制输出值。其中处理阶段是基于一组 IF-THEN 命题的逻辑集合，IF 部分是前提，THEN 部分是结果。典型的模糊控制系统有一堆规则。接下来给出本文的实例，室温模糊控制系统，两个输入：室内温度，温度变化；一个输出：温度调节量；处理规则：

- (1) IF 室内温度低，温度变化属于微微下降，THEN 全力加热；
  - (2) IF 室内温度低，温度变化属于几乎不变，THEN 中等加温；
  - IF 室内温度适中，温度变化属于几乎不变，THEN 不加热不降温；
  - IF 室内温度适中，温度变化属于微微上升，THEN 中等降温；
  - IF 室内温度高，温度变化属于几乎不变，THEN 微微降温；
  - IF 室内温度高，温度变化属于微微上升，THEN 中等降温；
- 处理规则可以用可能性模型构建如下图所示。

要检测的属性是输出最终可以将温度调节到温度适中，一个进程即可描述整个室温模糊控制系统，此实例中的两个输入和一个输出均是模糊集合，其中具体的温度值和温度变化值本来均对应有自己的隶属函数，此处为了简化建模模型，将两者的结合对应一个隶属函数，该隶属函数对应的模糊集合是对应规则中的前提，比如 10 度和每分钟 1 度下降属于室内温度低，温度变化属于微微下降这个模糊集合中的隶属度定义为 0.8，属于室内温度低，温度变化属于几乎不变这个模糊集合中的隶属度定义为 0.2。其对应的 prism 代码见下表所示。其要验证的属性是室温调节器是将温度向适宜的温度调节中，在此处为简化属性，使得属性中不出现模糊逻辑，将适宜温度指定为 25 度。本实验验证了如下属性：1.  $Po = ? [X(X \text{ tem} = 20)]$ ；2.  $Po = ? [\text{tem} = 20 \wedge \text{output} \leq 5]$ ；3.  $Po = ? [\text{output} > 4 \wedge \text{tem} \geq 30]$ 。

**算法 5.1** 简化室温控制系统的可能性模型类 prism 语言描述

**module** thermometer

1. **tem** : [10..45] init 10; // value of tempreature in room
2. **change**: [1..5] init 2;  
//1 decrease rapidly,2 decrease,3 unchanged,4 increase,5 increase rapidly
3. **output** : [1..7] init 1;  
//1cooling rapidly,2cooling,3cooling slightly,4do nothing,5warming slightly,6warming,  
//7warming rapidly  
// set of rule

- 
4.  $\square \text{ tem}=10 \ \& \ \text{change}=2 \rightarrow 0.8 : (\text{output}'=7) \ \& \ (\text{tem}'=20) + 0.2 : (\text{output}'=6) \ \& \ (\text{tem}'=15);$
  5.  $\square \text{ tem}=10 \ \& \ \text{change}=3 \rightarrow 0.6 : (\text{output}'=7) \ \& \ (\text{tem}'=25) + 0.4 : (\text{output}'=6) \ \& \ (\text{tem}'=20);$
  6.  $\square \text{ tem}=15 \ \& \ \text{change}=2 \rightarrow 0.3 : (\text{output}'=7) \ \& \ (\text{tem}'=25) + 0.7 : (\text{output}'=6) \ \& \ (\text{tem}'=20);$
  7.  $\square \text{ tem}=15 \ \& \ \text{change}=3 \rightarrow 0.1 : (\text{output}'=7) \ \& \ (\text{tem}'=30) + 0.5 : (\text{output}'=6) \ \& \ (\text{tem}'=25) + 0.4 : (\text{output}'=5) \ \& \ (\text{tem}'=20);$
  8.  $\square \text{ tem}=20 \ \& \ \text{change}=1 \rightarrow 0.5 : (\text{output}'=7) \ \& \ (\text{tem}'=25) + 0.5 : (\text{output}'=6) \ \& \ (\text{tem}'=20);$
  9.  $\square \text{ tem}=20 \ \& \ \text{change}=2 \rightarrow 0.5 : (\text{output}'=6) \ \& \ (\text{tem}'=25) + 0.5 : (\text{output}'=7) \ \& \ (\text{tem}'=30);$
  10.  $\square \text{ tem}=20 \ \& \ \text{change}=3 \rightarrow 0.3 : (\text{output}'=5) \ \& \ (\text{tem}'=25) + 0.7 : (\text{output}'=4);$
  11.  $\square \text{ tem}=20 \ \& \ \text{change}=4 \rightarrow 0.7 : (\text{output}'=4) \ \& \ (\text{tem}'=25) + 0.3 : (\text{output}'=5) \ \& \ (\text{tem}'=30);$
  12.  $\square \text{ tem}=20 \ \& \ \text{change}=5 \rightarrow 0.5 : (\text{output}'=4) \ \& \ (\text{tem}'=30) + 0.5 : (\text{output}'=3) \ \& \ (\text{tem}'=25);$
  13.  $\square \text{ tem}=25 \ \& \ \text{change}=1 \rightarrow 0.3 : (\text{output}'=5) \ \& \ (\text{tem}'=20) + 0.4 : (\text{output}'=6) + 0.3 : (\text{output}'=7) \ \& \ (\text{tem}'=30);$
  14.  $\square \text{ tem}=25 \ \& \ \text{change}=2 \rightarrow 0.9 : (\text{output}'=5) + 0.1 : (\text{output}'=4) \ \& \ (\text{tem}'=20);$
  15.  $\square \text{ tem}=25 \ \& \ \text{change}=3 \rightarrow 1 : (\text{output}'=4);$
  16.  $\square \text{ tem}=25 \ \& \ \text{change}=4 \rightarrow 0.7 : (\text{output}'=4) \ \& \ (\text{tem}'=30) + 0.3 : (\text{output}'=3);$
  17.  $\square \text{ tem}=25 \ \& \ \text{change}=5 \rightarrow 0.3 : (\text{output}'=1) \ \& \ (\text{tem}'=20) + 0.4 : (\text{output}'=2) + 0.3 : (\text{output}'=3) \ \& \ (\text{tem}'=30);$
  18.  $\square \text{ tem}=30 \ \& \ \text{change}=1 \rightarrow 0.7 : (\text{output}'=4) \ \& \ (\text{tem}'=20) + 0.3 : (\text{output}'=5) \ \& \ (\text{tem}'=25);$
  19.  $\square \text{ tem}=30 \ \& \ \text{change}=2 \rightarrow 1 : (\text{output}'=4) \ \& \ (\text{tem}'=25);$
  20.  $\square \text{ tem}=30 \ \& \ \text{change}=3 \rightarrow 0.3 : (\text{output}'=2) \ \& \ (\text{tem}'=20) + 0.4 : (\text{output}'=3) \ \& \ (\text{tem}'=25) + 0.3 : (\text{output}'=4);$
  21.  $\square \text{ tem}=30 \ \& \ \text{change}=4 \rightarrow 0.8 : (\text{output}'=2) \ \& \ (\text{tem}'=25) + 0.2 : (\text{output}'=1) \ \& \ (\text{tem}'=20);$
  22.  $\square \text{ tem}=30 \ \& \ \text{change}=5 \rightarrow 0.8 : (\text{output}'=1) \ \& \ (\text{tem}'=25) + 0.2 : (\text{output}'=2);$
  23.  $\square \text{ tem}=35 \ \& \ \text{change}=2 \rightarrow 0.7 : (\text{output}'=3) \ \& \ (\text{tem}'=25) + 0.3 : (\text{output}'=4) \ \& \ (\text{tem}'=30);$
  24.  $\square \text{ tem}=35 \ \& \ \text{change}=3 \rightarrow 0.3 : (\text{output}'=3) \ \& \ (\text{tem}'=30) + 0.7 : (\text{output}'=2) \ \& \ (\text{tem}'=25);$
  25.  $\square \text{ tem}=35 \ \& \ \text{change}=5 \rightarrow 0.9 : (\text{output}'=1) \ \& \ (\text{tem}'=25) + 0.1 : (\text{output}'=2) \ \& \ (\text{tem}'=30);$
  26.  $\square \text{ tem}=40 \ \& \ \text{change}=1 \rightarrow 0.8 : (\text{output}'=4) \ \& \ (\text{tem}'=30) + 0.2 : (\text{output}'=3) \ \& \ (\text{tem}'=25);$
  27.  $\square \text{ tem}=40 \ \& \ \text{change}=4 \rightarrow 1 : (\text{output}'=1) \ \& \ (\text{tem}'=30);$
  28.  $\square \text{ tem}=45 \ \& \ \text{change}=2 \rightarrow 0.6 : (\text{output}'=1) \ \& \ (\text{tem}'=25) + 0.4 : (\text{output}'=2) \ \& \ (\text{tem}'=30);$
  29.  $\square \text{ tem}=45 \ \& \ \text{change}=3 \rightarrow 0.8 : (\text{output}'=1) \ \& \ (\text{tem}'=30) + 0.2 : (\text{output}'=2) \ \& \ (\text{tem}'=35);$
  30. **endmodule**
- 

第二个示例即将同步领导协议中涉及到概率的，用可能性来描述，同步领导协议中涉及到概率的部分指的是每个进程选择自己的 id 时，是等概率的选择从 1 到 K 中选取一个数作为自己的 id，此处如果每个进程选择自己的 id 时，是从模糊

集合中按照一定的隶属函数取, 比如每个进程选择自己的 id 时, 会选取与自己的进程号相近的 id 值, 或者会选取与右边进程号相近的 id 值, 相近是一个模糊集合, 因此进程选择 id 时, 不再是等概率从一个整数区间中选取一个值, 而是遵照相近这个模糊集合对应的隶属函数从这个模糊集合中选取一个值。这样就把一个概率模型检测实例, 转化成了可能性模型检测实例。

要检测的属性几乎同 4.1.1 中相同, 除了将概率换成可能性外, 即在  $L$  次选择之中选取了领导的可能性, 每个进程对应的自动机如图所示, 其中第  $n$  个进程的状态在对应的 PRISM 语言中被描述成了有四个值的变量  $sn$ , 以  $s1$  为例, 变量  $s1=0$  表示的是一个状态, 这个状态的任务是选取 id, 变量  $s1=1$  代表的状态负责查看别的进程的 id,  $s1=2$  的状态完成了选择领导,  $s1=3$  对应的状态代表的是选出领导, 结束, 或者是没有合适的候选人, 结束。假设有 4 个进程, 那么对应的 PoCTL 属性被描述成:  $Po = ? [true \ U \leq L * (N + 1)(s1 = 3 \ \& \ s2 = 3 \ \dots \ sn = 3)]$ 。

### 5.3 实验结果

5.2.1 节中的实例与 5.2.2 节中的实例二分别检测的是同步领导协议的概率版本和可能性版本的关于  $P = ? [true \ U \leq L * (N + 1)(s1 = 3 \ \& \ s2 = 3 \ \dots \ \& \ sN = 3)]$  和  $Po = ? [true \ U \leq L * (N + 1)(s1 = 3 \ \& \ s2 = 3 \ \dots \ \& \ sN = 3)]$  模型检测结果, 其结果在  $L=1$ ,  $N$  与  $K$  取不同值时结果见表 5-2:

表 5-2 同步领导协议的概率结果和可能性结果

N	K	模型		构建时间	概率结果	可能性结果
		状态	变迁			
3	2	26	33	0.014	0.7500	1.250E-1
	4	147	210	0.042	0.9375	1.563E-2
	8	1059	1570	0.135	0.9840	1.953E-3
4	2	61	76	0.008	0.5000	6.250E-2
	4	812	1067	0.072	0.8438	3.906E-3
	8	12400	16495	1.403	0.9570	2.441E-4
5	2	141	172	0.067	0.3125	3.125E-2
	4	4244	5267	110.092s	0.8789	9.766E-4
	8	131521	164288	110.092s	0.9827	3.052E-5

表中  $N$  代表的是协议中投票的进程数,  $K$  代表的是每个进程可以选择的 id 的最大值, 第三、四、五列分别是生成的 DTMC/可能性模型中的状态数、变迁数和构建时间, 最后两列分别是  $P = ? [true \ U \leq (N + 1)(s1 = 3 \ \& \ s2 = 3 \ \& \ s3 = \ \& \ s4 =$

3)]和 $Po = ? [true \ U \leq (N + 1)(s1 = 3 \ \& \ s2 = 3 \ \dots \ sn = 3)]$ 的概率值和可能性值。可以看到对于相同的模型，把其中的概率变迁改成可能性变迁后，其对应的含义的情况下，其对应的结果值是不同的，用值说明了概率模型检测与可能性模型检测不同。

5.1.2 节中的实例一显示了可能性模型检测的实用范围，即可以应用在模糊控制领域，对如下几个属性进行了检测，其结果如下：

$Po = ? [true \ U < 2 \ \text{tem} = 20]$ : 0.139

$Po = ? [\text{tem} = 20 \ U < 3 \ \text{output} \leq 5]$ : 1

$Po = ? [\text{output} > 4 \ U < 3 \ \text{tem} \geq 30]$ : 0

5.1.2 节中的实例结果展示了模糊控制的几个属性值，概率模型无法对模糊控制建模，只有可能性才能描述模糊性，本实例展示的就是模糊控制的简化建模过程，其建模还有待完善。检测的三个属性分别表示的是在 2 步之内，温度变成 20 度的可能性，温度从 20 度在 3 步之内输出的温度控制是小于等于 5 的（温度控制小于等于 5 即温度控制的范围是缓慢加热、不作为、缓慢制冷、制冷、快速制冷）的可能性，及输出小于 4 在 3 步之内温度大于等于 30 度的可能性。

本节中的实验利用本文中提出的可能性模型检测算法，计算出了可能性算子，这是 PRISM 没有实现的。

## 5.4 本章小结

本章先是通过一个简单的实例，证明了可能性模型与概率模型检测结果的不同；然后再通过另一个信道传输实例，说明了可能性模型、DTMC、MDP 中的非确定性的不同含义是什么意思；最后给出了同步领导协议和室温控制两个现实的例子，展示了模糊控制这个领域如何使用模型检测方法来检测自己的属性。

## 第6章 结束语

### 6.1 工作总结

可能性模型用来描述概率模型无法描述的不确定性问题，本文对可能性模型检测进行了调查与实现，提出了可能性与认知逻辑的结合。

模型检测的理论研究可以按照内容来分，模型，属性，检测过程，实现四个方面；

模型与属性：本文研究了可能性多智体系统的模型——基于分布式知识的可能性解释系统；将可能性与认知逻辑和计算树逻辑统一起来，提出了新的逻辑 PoCTLK；在该系统下定义了两种 PoCTLK 语义，普通语义即在可能性多智能体系统下即能表示 CTL 属性的定量和定性部分，又能表示认知逻辑的定性部分；深层语义还加了认知逻辑的定量表示。

检测过程与实现：研究了可能性模型检测的具体实现。可能性模型检测的具体实现涉及到可能性模型描述语言的设计，语法分析器的实现、及数据结构的确和检测过程的实现，经过分析对比发现可能性模型检测与概率模型检测的异同点，发现可以通过修改概率模型检测工具来实现可能性模型检测工具。于是本文在分析可能性模型检测过程后，发现只需要重新定义两种矩阵运算，即可实现可能性模型检测中关于  $X\phi$ ,  $\Phi_1 \cup \Phi_2$  的实现过程。基于上述实现，本文通过实例，通过实例证明了可能性模型检测与概率模型检测的区别。

### 6.2 后续研究展望

可能性与模型检测的目前应用范围不广，同可能性理论的实际应用一样，没有得到足够重视。本文后续可做工作还有很多：

(1) 本文提到的可能性模型检测中检测的属性考虑的 PCTL 和 PCTLK，没有把模糊逻辑考虑进来，而实际上可能性模型与模糊逻辑的结合更适合实际应用。比如第五章的可能性模型检测中涉及到的模糊控制的例子，其属性就是由模糊逻辑表示更贴近实际。

(2) 可能性模型检测的  $\Phi_1 \cup \Phi_2$  的可能性计算具体过程中需要一步一步的计算，没有实现迭代计算过程，后续要仔细分析 Gauss-Seidel 和 Jacobi 解线性议程的计算方法，来实现  $\Phi_1 \cup \Phi_2$  的可能性计算。

(3) 本文中实现可能性模型检测的状态的可能性计算时，只实现了基于 BDD

数据结构的计算方式，而 Prism 中计算模型的概率时采用了三种办法，分别是基于 BDD 数据结构的计算，基于稀疏矩阵的数据结构的计算，以及两种方法的结合；在后续工作中可以实现可能性模型检测的三种方法，这是因为基于 BDD 的计算和基于稀疏矩阵的计算各有各的优势。基于稀疏矩阵的计算的优势是，存储更紧凑（对于非零项的比例相当低的情况下），访问矩阵元素更快（可以直接通过简单计算进行访问）；其劣势是矩阵元素操作比起 BDD 操作来说要慢很多。所以后续工作中希望可以实现两种方法，然后根据模型的结构选择合适的方法。

(4) 本文中 PoCTLK 中关于认知逻辑的可能性定义，可以随着实际应用的不同，给出不同的定义，正如可能性本身的可能性分布随着不同应用，有不同的定义一样，需要更深入的研究。

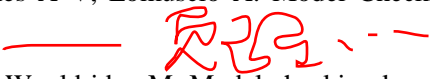
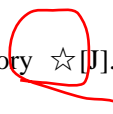
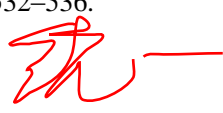
(5) 本文提到的模糊控制建模实例，限于只能计算  $X\varphi$ ,  $\Phi_1 U^{\leq n} \Phi_2$  两种属性，没有表示出其安全性属性和实际关注的属性。



## 参考文献

- [1] Clarke E M, Grumberg O, Peled D A. Model checking[M]. MIT press, 1999.
- [2] A.Biere, A. Cimatti, E. Clarke, and Y.Zhu, Symbolic model checking without BDDs, Proc. of the 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems(FACAS'99), LNCS, vol. 1579, Springer-Verlag, 1999.
- [3] A.Biere, A.Cimatti, E.Clare, O.strichman, and Y.Zhu, Bounded model checking, Highly Dependable Software, Advances in Computers, vol. 58, Academic Press, 2003.
- [4] Gerth R, Peled D, Y. Vardi M, et al. Simple On-the-fly Automatic Verification of Linear Temporal Logic[J]. Simple On-the-fly Automatic Verification of Linear Temporal Logic - ResearchGate, 1995.
- [5] Katz S, Peled D. Verification of distributed programs using representative interleaving sequences[C]. //Distributed Computing. 1992:107-120.
- [6] Holzmann G J, Peled. D.: An improvement in formal verification[J]. Proc Formal Description Techniques Forte, 1994, 80(2):p ágs. 62-65.
- [7] Cimatti A, Clarke E, Giunchiglia E, et al. NuSMV 2: An OpenSource Tool for Symbolic Model Checking[J]. Lecture Notes in Computer Science, 2002:359-364
- [8] 张军林. NuSMV 模型验证器实现分析[D] 中山大学, 2010.
- [9] Fujita M, McGeer P C, Yang J C Y. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation[J]. Formal methods in system design, 1997, 10(2-3): 149-169. Bahar R I, Frohm E A, Gaona C M, et al. Algebraic decision diagrams and their applications[J]. Formal methods in system design, 1997, 10(2-3): 171-206.
- [10] Somenzi F. CUDD: CU decision diagram package release 2.3. 0[J]. University of Colorado at Boulder, 1998.
- [11] L. Mcmillan K. Applying SAT Methods in Unbounded Symbolic Model Checking[J]. Lecture Notes in Computer Science, 2002:250-264.
- [12] Clarke E, Biere A, Raimi R, et al. Bounded Model Checking Using Satisfiability Solving[J]. Formal Methods in System Design, 2001, 19(1):7-34.
- [13] Shtrichman O. Tuning SAT checkers for bounded model checking[C]//Computer Aided Verification. Springer Berlin Heidelberg, 2000: 480-494.
- [14] Penczek W, Wozna B, Zbrzezny A. Bounded Model Checking for the Universal Fragment of CTL[J]. Fundamenta Informaticae, 2002, 51:135-156.
- [15] Kroening D, Ouaknine J, Strichman O, et al. Linear Completeness Thresholds for Bounded Model Checking[J]. Lecture Notes in Computer Science, 2011, 19(1):177 - 186.
- [16] Bundala D, Ouaknine J, Worrell J. On the Magnitude of Completeness Thresholds in Bounded Model Checking[C]. //IEEE Symposium on Logic in Computer Science. IEEE, 2012:155 - 164.
- [17] Adnan Aziz Kumud Sanwal, Vigyan Singhal and Robert Brayton. Verifying continuous time



- Markov chains [M]. Springer Berlin Heidelberg, 1996:269--276.
- [18] Halpern J Y, Vardi M Y. The complexity of reasoning about knowledge and time. I. Lower bounds[J]. Journal of Computer and System Sciences, 1989, 38(1): 195-237.
- [19] Belardinelli F, Jones A V, Lomuscio A. Model Checking Temporal-Epistemic Logic Using Tree Automata[J].  ---
- [20] Van Der Hoek W, Wooldridge M. Model checking knowledge and time[M]//Model Checking Software. Springer Berlin Heidelberg, 2002: 95-111.
- [21] Raimondi F, Lomuscio A. Automatic verification of deontic interpreted systems by model checking via OBDD's[C]//ECAI. 2004, 16: 53.
- [22] Zadeh L A. Fuzzy sets as a basis for a theory of possibility[J]. Fuzzy sets and systems, 1999, 100: 9-34.
- [23] Dubois D, Prade H. Possibility Theory, Probability Theory and Multiple-Valued Logics: A Clarification[J]. Annals of Mathematics & Artificial Intelligence, 2001, 32(1-4):35-66.
- [24] Lomuscio A, Raimondi F. Model checking knowledge, strategies, and games in multi-agent systems[C]//Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. ACM, 2006: 161-168.
- [25] Zadeh L A. A note on similarity-based definitions of possibility and probability[J]. Information Sciences, 2014, 267: 334-336.
- [26] Li Y, Li L. Model-Checking of Linear-Time Properties Based on Possibility Measure[J]. Fuzzy Systems IEEE Transactions on, 2012, 21(5):842 - 854.
- [27] Li Y, Li Y, Ma Z. Computation Tree Logic Model Checking Based on Possibility Measures[J]. Fuzzy Sets & Systems, 2014. ---
- [28] Li Y, Ma Z. Quantitative Computation Tree Logic Model Checking Based on Generalized Possibility Measures[J]. EprintArxiv, 2014. ---
- [29] Zadeh L A. A note on modal logic and possibility theory ☆[J]. Information Sciences, 2014:908 - 913. 
- [30] Kwiatkowska M, Norman G, Parker D. PRISM 2.0: A tool for probabilistic model checking[J]. International Conference on Quantitative Evaluation of Systems, 2004:322 - 323.
- [31] Alfaro L D, Kwiatkowska M, Norman G, et al. Symbolic model checking for probabilistic processes using MTBDDs and the Kroneckerrepresentation[J]. Tools & Algorithms for the Analysis & Construction of Systems Lncs, 2000:395--410.
- [32] M. Kwiatkowska and G. Norman and D. Parker. PRISM4.0: Verification of Probabilistic Real-time Systems[J].Proc. 23rd International Conference on Computer Aided Verification CAV'11, 2011585--591.
- [33] Chen T, Diciolla M, Kwiatkowska M, et al. Quantitative Verification of Implantable Cardiac Pacemakers[C]. //IEEE Real-time Systems Symposium. IEEE, 2012:263 - 272.
- [34] H. Geffner, J. Wainer, Modeling action, knowledge and control, in: Proceedings of the European Conference on Artificial Intelligence (ECAI), 1998, pp 532--536. 

- [35] L.P. Kaelbling, M.L. Littman, A.R. Cassandra, Planning and acting in partially observable stochastic domains, *Artificial Intelligence* 101 (1–2) (1998) 99–134.
- [36] I. Anciutti, A learning classifier system for emergent team behavior in real-time POMDP in: *Proceedings of the IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS)*, 2009, pp. 733–738.
- [37] R. Fagin, J.Y. Halpern, Y. Moses, Y.M. Vardi, *Reasoning About Knowledge*, MIT Press, Cambridge, 1995.
- [38] A. Lomuscio, W. Penczek, Symbolic model checking for temporal-epistemic logics, *SIGACT News* 38 (3) (2007) 76–100.
- [39] Fagin R, Halpern J Y. Reasoning about knowledge and probability[J]. *Journal of the Acm*, 1999, 41(2):340–367.
- [40] J. Lawry, Y. Tang, On truth-gaps, bipolar belief and the assertability of vague propositions, *Artificial Intelligence* (2012) 20–41.
- [41] Z. Cao, Model checking for epistemic and temporal properties of uncertain agents, in: *Agent Computing and Multi-Agent Systems*, LNAI, vol. 4088, Springer, 2006, pp. 46–58.
- [42] Wan W, Bentahar J, Hamza A B. Model checking epistemic-probabilistic logic using probabilistic interpreted systems[J]. *Knowledge-Based Systems*, 2013, 50: 279–295.

## 生 致谢 生

研究生三年的生活，罗贵明教授言传身教，每天朝八点晚十点，节假日几乎不休息，每次找罗老师讨论学术，老师都认真，提纲挈领的提建议，我愚钝，有时要一个星期后才能反映过来，老师也从不责怪。除了关于具体的学术讨论，老师还会就学术态度给出自己的见解，这些都是我从老师身上学到的东西，将指导我日后的生活。

实验室的师兄师姐师弟师妹们，都在不同地方得到了大家的帮助，心存感激。尤其是张宇来师兄，罗建师兄和夏默师兄。张宇来师兄从学术和心理上都给予我一定的指导，用自己的亲身经历教我如何做学术；罗建师兄对概念的精确把握对完美的追求，每次和他讨论问题都可以清晰自己的思维；夏默师兄的软件编程能力特别厉害，遇到编程方面的问题找他一般都能解决。

同时还要感谢我的朋友，研究生期间和他们一起上课，成长，他们是我一起成长的伙伴，他们身上的优点也帮助我变成更好的自己。最后要感谢的是父母，从物质、精神上父母都给予了我最大的支持。

## 声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签名：\_\_\_\_\_日期：\_\_\_\_\_

## 个人简历、在学期间发表的学术论文与研究成果

### 个人简历

1988 年 4 月 2 日出生于山西省太原市清徐县。

2007 年 9 月考入大连交通大学，学习安全工程+软件工程专业，2012 年 7 月本科毕业并获得双专业工学学士学位。

2012 年 9 月免试进入清华大学，学习软件工程专业攻读工程硕士学位至今。