密级★保密期限：（涉密论文须标注）

浙江理工大學

Zhejiang Sci-Tech University

硕 士 学 位 论 文

Master's Thesis

中文论文题目：自适应软件系统的建模与验证

英文论文题目：**Modeling and Verification of Self-Adaptive Software Systems**

学科专业：计算数学

作者姓名：周远

指导教师：丁佐华

完成日期：**2014-12-24**

浙江理工大学学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得浙江理工大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：

签字日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解浙江理工大学有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权浙江理工大学可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：

签字日期：　　年　　月　　日

导师签名：

签字日期：　　年　　月　　日

摘 要

自适应软件系统能够根据自身需求和环境的变化自动改变自己的行为。此时，在软件运行时，运行环境的变化将产生新的需求，而传统软件模型是对固定的需求进行建模，因此它无法适应这些新的需求。我们必须对自适应软件建立新的模型，它们能够适应实时产生的新需求。此外，由于环境的连续性和不确定性，这类模型刻画的是不定的、无限的状态，因此在建立自适应软件模型后，还需要对模型进行准确性和性质的验证。

本文主要研究了两类自适应软件系统的建模与验证。第一类是基于控制理论的自适应软件系统，主要研究了两类行为的切换系统：切换模糊系统和切换随机系统。切换模糊系统由全局切换规则、局部模糊规则和常微分方程组构成；首先对模糊系统去模糊化，然后用混合自动机对其建模，最后通过 PHAVer 计算可达状态空间来分析系统的稳定性。切换随机系统包括由 Markov 链描述的切换规则和由随机微分方程描述的子系统；本文提出了一种新的 Petri 网——随机微分 Petri 网对系统的离散切换行为、连续和随机行为进行建模；然后通过等价类的划分，化无限可达状态为有限 Markov 行为，最后利用 PRISM 进行验证。第二类是环境感知的自适应软件系统。对此，本文提出了一种新的建模语言——自适应 Petri 网来对其进行建模，并分析了模型的自适应性。自适应 Petri 网是混合 Petri 网的扩展，在其中嵌入了神经网络，用来对环境的变化做出决策。自适应 Petri 网具有如下一些优点：对运行环境进行建模、不同组件通过合作来完成决策过程、通过神经网络的局部计算达到全局的自适应行为。最后，通过一个制造系统的例子给出了自适应 Petri 网的应用。

本文主要进行了如下四个方面的工作：

1）提出了对具有模糊行为的自适应系统——切换模糊系统的形式化建模方法和系统稳定性的验证技术，从而避免传统方法中寻找 Lyapunov 函数的困难；

2）提出了一种能同时描述离散、连续和随机行为的形式化建模语言，它可以对具有随机行为的自适应系统——切换随机系统进行建模和模型检查；

3）提出了一种对环境感知的自适应系统的建模语言——自适应 Petri 网。它既能描述系统的行为，又能对运行环境进行建模，同时具有良好的可扩展性；

4）提出了对系统的可达状态构造等价类，通过等价类划分，化无限状态为有限行为，从而在一定程度上解决自适应模型在验证时产生的状态爆炸问题。

关键字：自适应软件系统，软件建模，模型验证，Petri 网，神经网络，混合自动机，随机微分 Petri 网，自适应 Petri 网

Abstract

A self-adaptive software system can modify its own behavior in response to the changes of its requirements and operational environment at runtime. During runtime, the changes of operational environment will produce new requirements, but traditional models can only deal with fixed requirements and cannot adapt themselves to the new requirements. Thus new models which have the ability to describe the new requirements automatically must to be built to model the self-adaptive software systems. Besides, because of the continuity and uncertainty of the environment, the new models have infinite and uncertain states, so we must verify the correctness and properties of the new models.

This thesis focuses on the modeling and verification of two kinds of self-adaptive systems. The first kind is the control theory based self-adaptive software systems. For this one, the thesis mainly investigates the formal modeling methods and verification techniques for the switched fuzzy systems and switched stochastic systems. A switched fuzzy system consists of global switching rules, local fuzzy rules and ordinary differential equations. The system is firstly deblurred, and then transformed to a hybrid automaton. The reachable space of the gotten automaton is then computed by the model checker PHAVer. Finally the stability is obtained by analyzing the reachable space. A switched stochastic system is usually described by a Markov chain and stochastic differential equations. A new Petri net called stochastic-differential Petri net is proposed to model the discrete behavior, continuous and stochastic behavior of the switched stochastic systems. The proposed model has infinite reachable states, so an equivalence relation between reachable states is constructed and a Markov chain is obtained from the state space based on the corresponding equivalence partitioning. Thus the probabilistic model checker PRISM can be used to do model checking. The second kind is the environment-awareness self-adaptive software systems, and a new modeling language called adaptive Petri net is proposed to model this kind of systems. An adaptive Petri net is an extension of the hybrid Petri net by embedding a neural network in some special transitions, which is used to make decisions according the changes of environment. The proposed net has the following advantages: It can model a runtime environment, the components in the model can collaborate to make adaption decisions; and the computing is done at the local while the adaption is for the whole system. At last a manufacturing system is used to illustrate the proposed adaptive Petri net.

This thesis has done the following work:

1) It is the first work that gives a formal method for modeling the switched fuzzy systems and is able to use a model checking technique to analyze the systems' stability, which can avoid finding the Lyapunov functions in the traditional methods.

2) A new Petri net is proposed to model discrete behavior, continuous behavior and stochastic behavior. It can be used to describe the switched stochastic systems and do model checking.

3) A new formal language is built to describe the environment-awareness self-adaptive software systems. It can not only model the software systems, but also can model the runtime environment. It has a good scalability.

4) A new technology is proposed to solve the problem of state explosion during the model checking of self-adaptive software systems.

Keywords: Self-Adaptive software systems, software modeling, model checking, petri ent, neural network, hybrid automata, stochastic-differential Petri net, adaptive Petri net.

目 录

图 清 单

1 绪论

1.1 研究意义

随着因特网向社会各个角落的渗透式扩张、各种技术的整合，以及普适计算、网格计算、物联网、云计算、CPS 等新型应用模式不断涌现，系统从软件密集型系统向超大规模系统的发展，使得软件的规模和复杂性不断增加；而同时我们又希望软件系统具有多功能、可靠性、健壮性、可修复、可配置、自身优化等诸多特点，这些给软件在开发、管理、维护方面带来很大的困难。例如无人驾驶汽车[1][2]中的软件需要实时观测周围环境的变化，并根据环境的不同来调整系统的行为，以保证系统能够正常运行，例如当前面有障碍时，汽车需改变其方向来避开障碍物；当前面有急转弯时，汽车需减小速度以避免转弯时翻车，等等。又如，移动 Ad-Hoc 网络[3]中的移动主机需要实时检测其位置，并实时搜索覆盖范围内的其他移动主机，确定合适的通信方式。再如，服务组合需根据运行环境（用户数量的变化）、服务质量的变化，实时选择合适的服务来完成服务。我们发现这些例子都有一个共性，即软件系统必须根据运行环境和自身需求的变化而改变系统的行为(结构、模式等)来完成目标任务。此时传统软件的使用将使系统的更新成本和维护成本变得十分昂贵,因为传统的软件系统要求软件工程人员在描述软件过程时预期所有可能发生的情况，并显式地定义对应的解决方案；这种软件过程模型大多是静态的、机械的、被动的，难以自适应地对环境或系统的变化作出合理的反应。因此，人们受生物学和自然界中的一些自适应系统启发，提出了软件的自适应性和自适应软件，以期望其能够满足上述软件需求和应对软件设计的新挑战。

目前有很多被广泛接受的自适应的定义，例如，Ravindranathan 等[4]认为，自适应性是指在多模型系统中不同模型间切换的过程和方法；Salehie 等[5]认为自适应系统能够对系统自身的行为进行评估，当评估发现软件无法完成所预期的行为或者有更好的功能、性能时，软件就自动改变其行为；Oreizy 等[6]定义自适应软件为能够修改其自身行为以便对操作环境的变化做出响应，这里的操作环境是指任何可被软件系统观测到的事件，例如终端用户的输入、外部硬件设备和传感器、程序插桩等。总而言之，自适应系统是指系统在运行时能够根据自身需求和运行环境的变化，自动地改变系统行为。

目前人们在各个研究领域对自适应开展研究，例如从软件工程领域，包括需求工程[7][8][9][10][11][12]、软件架构[6][13][14][15][16][17][18][19][20][21]、中间体[21][22] [23][24][25][26]、基于组件的开发[3][27][28][29][30][31][32][33]、程序语言[34][35]等等，但是这些研究是相对独立的；研究者还从系统理论、人工智能、计算机科学等理论基础研究领域和实际应用领域开展自适应研究，例

如控制工程、移动自主机器人、多 agent 系统、容错计算、可靠性计算、自主通信、可调整用户接口、机器学习、传感器网络、移动 Ad Hoc 网络、嵌入式系统等等。软件已成为这些系统具有自适应性的关键和普遍的要素。因此对自适应软件的研究具有重要的理论和应用意义。

1.2 国内外研究现状

20 世纪 90 年代前后，学术界和工业界开始兴起对自适应软件的研究。导弹、机器人等中嵌入的软件可认为是早期阶段的自适应软件。真正从软件适应性角度对自适应进行研究的一个标志是 IEEE 杂志“智能系统”(Intelligent System)在 1999 年第 3 期上出版的专集。随后软件自适应的研究很快成为软件界的研究热点，举办了许多关于自适应的系列性国际会议，例如：SEAMS（International Symposium on Software Engineering for Adaptive and Self-Managing Systems，已举办了 9 界），SASO（International Conference on Self-Adaptive and Self-Organizing Systems，已举办 8 界），ICAC（International Conference on Autonomic Computing，已举办 11 界），SAAES（PECCS Special Session on Self-Aware and Adaptive Embedded Systems，已举办 4 届），WOSS（ACM Workshop on Self-healing Systems，后来演变为 SEAMS），IWSOS（International Workshop on Self-Organizing Systems，已举办 8 界），IWSAS（International Workshop Self-Adaptive Software，举办了 2 界）；同时美国计算机协会创办了相关的专门期刊：ACM Transactions on Autonomous and Adaptive Systems。目前也有一些文献对自适应软件的研究现状、研究困难、研究方向进行了总结和归纳[5][36][37][38]。本文将从自适应软件的需求、模型和验证简单地介绍一些相关工作。

1.2.1 自适应软件需求的研究

随着自适应软件的广泛应用，对自适应软件的需求的描述变得至关重要。Sawyer 等[11]指出自适应系统的需求是实时的，能够处理因意料之外的环境所引起的新的需求。因此，他们提出了需求反射（requirement reflection），即根据运行时的需求模型进行需求自省和推理。需求反射允许自适应系统在运行时修改和重新评估设计阶段的决策行为。Dardenne 等[8]提出的 KAOS 方法提供了用图形化方式来表示自适应语义，Brown 等[7]对每一个自适应语义定义了一个高层的目标，并把它们表示为对应的 KAOS 全局实体，Nakagawa 等[10]利用 KAOS 对面向全局的需求进行分析和建模，并构建了相应的自适应性系统。Whittle 等[9]提出了一种对自适应需求进行建模的语言——RELAX，通过 *SHALL*、*MAY*、*AS CLOSE AS POSSIBLE*、 *AS EARLY AS POSSIBLE AFTER*、*AS {MANY|FEW} <subject> AS POSSIBLE* 等表达式来描述自适应需求中的不确定性。

1.2.2 自适应软件模型的研究

研究者对自适应模型已经展开了较多的研究， Chen 等在 2008 年的自适应系统的软件工程讨论班上总结了自适应软件模型应具备的特征或维（Dimensions），分别从目标（Goal）、变化（Change）、机制（Mechanism）、效果（Effects）四个方面进行了讨论[36]；在 2010 年的自适应系统的软件工程讨论班上又从自适应解决方案的设计空间（design space for adaptive solutions）、过程（processes）、从集中到分散的控制（from centralized to decentralized control）、实际运行时的验证和确认（practical run-time verication and validation）等四个方面进行了总结[37]。

到目前为止，人们提出了许多的自适应软件系统模型，有从软件架构方面来建模的[6][13-21]，有基于 Agent 的自适应模型[21-26]，有基于组件的自适应模型[3][27-33]，也有从控制论角度来建模的[5][39-41]。

- 基于软件架构的自适应模型。例如，Garlan 等[14]通过描述架构类型和修复策略来实现动态变化，同时他们设计了一个叫做 Rainbow 的框架[15]，它具备在运行时监视、检测、决定、实施的自适应能力； Gomaa 等[17]描述了自适应软件的架构的演变模型和开发方法。高晖、张莉等[19]结合经验数据和专家知识，基于贝叶斯网建立了软件体系结构层次的软件适应性预测模型。黄罡、王千祥等[21]介绍了一个反射式的 J2EE 应用服务器 PKUAS，引入软件体系结构作为全局视图以实现反射体系对系统整体的表示和控制。
- 基于 Agent 的自适应模型。例如，Fock 等[22]提出了一个叫做 Agilla 的移动 Agent 中间体来描述无限传感网络的自适应性质；吕建等[23]提出了一种建立基于 Agent 的网构软件模型的技术途径。赵欣培、李明树等[24]提出了一种基于 Agent 的自适应软件过程模型，这些软件过程 Agent 能够对软件过程环境的变化主动地、自治地做出反应，动态地确定和变更其行为以实现软件开放的目标；
- 基于组件的自适应模型。Bencomo 等[27]提出了一个称为 Genic 的建模工具，它能够支持对可重构的、基于组件的系统的建模、生产和操作；Mishra 等[30]设计了一个基于组件的自适应模型，它通过访问先验可用存储库中的等价组件来实现运行时的组件集成；Yoem 等[31]提出了一个可扩展模型来描述自适应的、自治的、高度分布的、基于移动 Agent 的网络应用，他们在模型中定义了基于组件的 agent 的一些关键特性。孙熙、梅宏等[32]将软件 Agent 技术和组件技术结合起来，使得组件能够根据环境的状态调整自己的行为。

- 基于控制理论的自适应软件。Salehie 等[5]将自适应机制分解成四个过程：监督（monitoring）、分析（analyzing）、规划（planning）和执行（executing），简称 MAPE 模式，并构成一个闭回路来实现系统的自适应机制；Weyns 等在文献[39]中归纳总结了如何对具有多个 MAPE 回路的复杂自适应系统进行回路的分散化；Kokar 等[40]探讨了自适应软件的五种控制模式：开回路、闭回路、闭回路带有 QoS 子系统、间接适应、可重配置，其中闭回路带有 QoS 子系统是最常用的模型；Karsai[41]等运用控制论设计了一个两层系统，把主要的应用功能与监控分开，把与适应性相关的活动拿来设计架构，同时还阐述了如何把复杂的决策逻辑嵌入到设计中。

1.2.3 自适应模型验证的研究

目前对自适应模型验证的研究相对集中在静态研究，人们尝试用传统的静态模型验证方法来对自适应软件的一些关键性质进行验证，如 Kramer 和 Magee[42]提出了一个动态配置模型，该模型只允许变化出现在系统中当前处于静止的组件，他们利用标签变迁系统（Labelled Transition Systems）来描述系统的行为，合成可达性分析（Compositional Reachability Analysis）来验证系统模型； Zhang 和 Cheng[43]提出了利用全局不变式来描述软件自适应时应满足的性质，其中全局不变式由线序时序逻辑来描述；通过对全局不变式的验证来进行模型的验证。但随着自适应软件的规模和复杂度的增加，静态模型检查会遇到状态爆炸问题。此时，动态模型验证可作为对静态模型检查的补充。这是因为在软件运行时，模型检查工具仅产生一条轨迹，而不是全部的状态；然后通过分析该轨迹确定是否与形式化规约一致[44]。Meredith 在他的博士论文中[45]研究了实时验证过程中的一个重要方面，即实时监督；提出来多种可以被有效监督的逻辑形式，同时开发一个实施监督软件，其中嵌入了面向监督的编程（MOP）框架（目前包含了两个实例：JavaMOP, BusMOP）。Iftikhar 等[46]研究了分散控制自适应系统（decentralized self-adaptive systems）的行为的形式化验证技术，他们用时间自动机描述系统过程，用时间计算逻辑树描述需要验证的性质，然后利用 Uppaal 来验证系统的灵活性和鲁棒性。Rouff[47]介绍了一种称为 *AdaptiV* 的自适应验证技术，该方法结合了稳定性科学、高性能计算模拟、组合验证和传统验证技术，以及操作监督。Goldsby 等[48]介绍了一种实时验证模型的方法——AMOEBA-RT，其中自适应性质由 A-LTL 来描述。Wang 等[49]提出一种基于数据流的测试方法来帮助开发人员检查模型一致性问题的方法。Calinescu 等[50]提出了对自适应系统在运行时的定量验证，该方法用来分析具有随机行为的系统的正确性、性能和可靠性，其思路为：用数学模型描述软件系统，用嵌入概率和消费/回报的时序逻辑来描述需求，然后详细分析系统是否满足需求，

最后利用概率检查模型 PRISM 来验证上述提出的方法。Bencomo[51]研究了运行时模型的使用情况，并指出了其中问题。

1.3 本文研究内容

自适应软件系统是指系统在实时运行时能够根据环境变化和系统自身变化而自动调整系统行为。国内外对自适应系统进行了很多研究，其研究重点在于自适应模型的提出和对自适应模型的检查。但是这些工作有如下一些不足：

- 在自适应模型上，人们提出了很多自适应模型，但是这些模型都是在系统框架上进行阐述，而很少涉及自适应软件系统的形式化建模，而这对软件的实施和验证是十分必要的；
- 环境对自适应系统的影响是巨大的，但是很少有模型能同时对环境进行建模的，而是将环境和系统分离开来；
- 在模型验证上面，国内外学者试图将静态验证模型技术应用到自适应系统的模型检查，但是自适应系统是实时动态变化的，故其状态是无限的，因此传统的基于状态空间的模型检查技术有可能导致状态爆炸问题而无法起作用；而动态的模型检查仅是对执行时的一条轨迹进行分析、验证，无法体现整个系统的全部行为。

因此，本文对自适应软件系统的形式化建模方法开展研究，建立形式化建模语言，为软件系统的底层编程实现提供高层设计模型；其次，提出自适应模型的验证技术，该技术能够将无限状态变为有限行为，然后进行模型验证，避免状态爆炸。

本文主要研究两类自适应系统的形式化建模和模型验证。第一类自适应系统是基于控制理论的自适应切换系统，它通过一个独立的监督控制组件来监测控制输入和系统变量的变化并实时进行系统行为的选择。第二类自适应系统是环境感知的自适应系统，它不仅需要根据系统本身的变化来改变行为，同时还需要对环境进行感知，根据环境的变化来调整行为以适应外部环境。这类自适应系统也是当前自适应软件研究者的关注重心。两类自适应系统的一般框架如图 1.1。

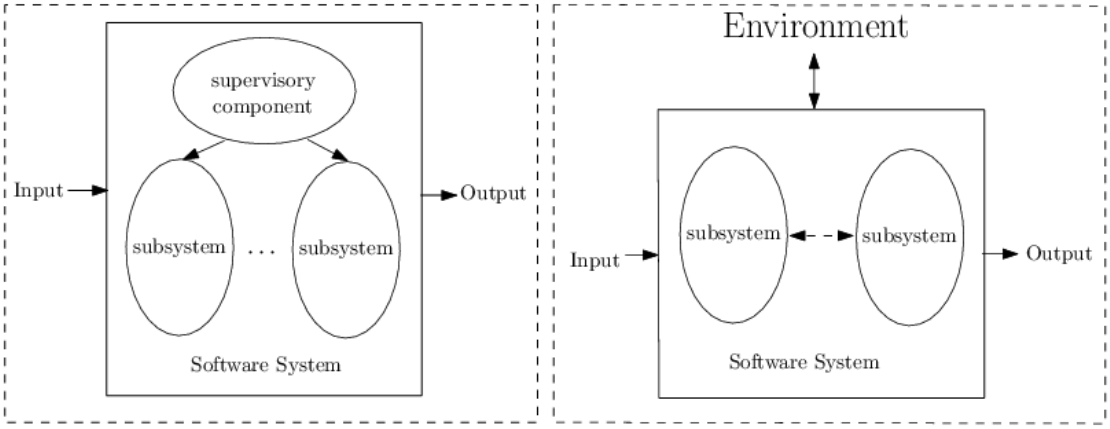


图 1.1 本文研究的两类自适应系统

对于第一类系统，本文主要研究了子系统由模糊规则和常微分方程描述、监督组件由确定有限状态机描述的切换模糊系统和子系统由随机微分方程描述、监督组件由连续时间 Markov 链描述的切换随机系统。对于前者，本文用混合自动机进行建模，并利用模型检查工具 PHAVer 进行验证；对于后者，本文提出了一种新的称为随机微分 Petri 网的形式化模型对其进行建模，并利用概率模型检查工具 PRISM 进行验证。

对于第二类系统，其形式化建模相对复杂。由于环境是高度开放、不确定的，因此本文通过神经网络进行环境的训练和学习，然后将其嵌入到 Petri 网中，利用神经网络对环境的变化做出决策，提出了一种新的形式化建模语言——自适应 Petri 网，并进行了自适应性的分析。最后通过一个制造系统例子来讲述该模型的应用和实验模拟。

本文剩余部分的结构如下：第二章是预备知识介绍，主要介绍了 Petri 网、混合自动机和神经网络的一些基本知识；第三章具体研究了对第一类基于控制理论的自适应系统的形式化建模和验证方法；第四章对第二类环境感知的自适应系统的形式化建模和验证方法进行研究分析；第五章给出两类自适应系统的实例演示和分析；最后，第六章对本论文进行总结和展望。

2 预备知识

本章将首先简单介绍一些基本知识，包括 Petri 网、混合自动机以及神经网络。

2.1 Petri 网

Petri 网是一个可对复杂系统进行建模的图形化、数学化的建模语言，尤其适用于对并发系统、分布系统的建模。可用 Petri 网来建模的典型场景有：同步/异步过程、序列化过程、并发过程以及冲突选择。Petri 网理论最早由 Carl Adam Petri 在他的博士论文里提出[52]。此后，Petri 网理论进一步发展，并逐渐在很多理论领域和应用领域得到应用，例如通信系统、ATM 网络、计算机系统、柔性制造系统、Web 服务集群、工作流等等。Petri 网中包含两种类型的节点：库所（place）和变迁（transition），节点之间用有向弧连接，且有向弧只能连接不同类型的节点，而不能连接同类的节点。Petri 网的基本思想是通过 Petri 网来描述系统的状态变化。因此，在 Petri 网中，库所表示系统的局部状态（状况或条件），每一个库所在不同时刻包含或者不包含标记（token），通过标记的变化来表示系统状态的变化，当一个库所包含标记时，则说明系统处于这个状态下或者代表的条件为真；变迁表示系统的动作，这些动作的发生需要具备不同的条件，只有当变迁的所有输入库所都包含标记时，变迁才能实施，从而引起库所中的标记的变化，体现出系统状态的变化。Petri 网既有直观的图形表示来描述系统的行为，又有严格的数学表达式用来分析系统的性质，其基本形式化定义如下：

定义 2-1（Petri 网）[53]：Petri 网是一个 5 元组$PN = \langle P, T, F, W, M_0 \rangle$，满足：

1）$P = \{p_1, p_2, \ldots, p_m\}$是有限库所集合；

2）$T = \{t_1, t_2, \ldots, t_n\}$是有限变迁集合；

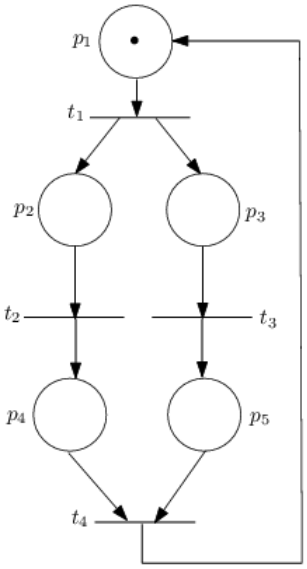
3）$F \subseteq (P \times T) \bigcup (T \times P)$是有向弧集合；

4）$W : F \to Z^+$是每条弧的权重；

5）M_0是初始标识（initial making）。

其中P, T需满足$P \bigcap T = \emptyset$和$P \bigcup T \neq \emptyset$。

图 2.1 是一个简单的 Petri 网模型，右边给出了库所集合、变迁集合、弧集合、权重和初始标记。



1) $P=\{p_1,p_2,p_3,p_4,p_5\}$

2) $T=\{t_1,t_2,t_3,t_4\}$

3) $F=\{(p_1,t_1),(t_1,p_2),(t_1,p_3),$
$(p_2,t_2),(p_3,t_3),(t_2,p_4),$
$(t_3,p_5),(p_4,t_4),(p_5,t_4),(t_4,p_1)\}$

4) $W(arc)=1, arc\in F$

5) $M_0=(1,0,0,0,0)$

图 2.1 一个简单的 Petri 网模型

2.2 混合自动机

混合自动机是精确描述混合系统的一个图形化、数学化工具。混合自动机实际上是一类具有连续变量的有限状态机。其中离散状态用图的顶点表示（即控制模式），离散动态变化用图中的有向弧描述（即控制切换）；连续状态用欧式空间中的点描述（连续变量），连续动态变化用常微分方程组来描述（流条件）。连续状态的变化（流条件）依赖于控制模式，不同模式用不同的常微分方程组来控制连续变量的变化；反过来，模式的切换又依赖于连续状态（连续变量的值），不同连接状态导致不同的模式切换。混合自动机基本的形式化定义如下：

定义 2-2[54]：混合自动机(Hybrid Automata)是一个七元组：$(V, X, \Lambda, flow, jump, inv, init)$，其中：

1) $V=(v_1,v_2,\cdots,v_m)$是控制模式集合。

2) $X=(x_1,x_2,\cdots,x_n)$是实值连续变量集合， n 称为自动机的维数。

3) Λ是切换集合。每一个切换$e=(v,v')\in\Lambda$表示一条由源模式 v 指向目标模式 v'的有向线段；

4) $flow$是V 上的流条件函数。它为每一模式分配一个流条件来描述系统在该模式内的状态变化，是混合自动机的连续部分。因此 $flow(v)$是关于 $X\cup\dot{X}$ 的微分方程组，其中 $\dot{X}=\{\dot{x}_1,\dot{x}_2,\cdots,\dot{x}_n\}$， $\dot{x}_i(i=1,\cdots,n)$表示系统变量x_i的一阶导数。

5) *jump* 是 Λ 上的跳跃条件函数。它为每一个切换 e 设置一个切换条件来控制状态源模式和目标模式间的切换，是混合自动机的离散部分。因此 $jump(e)$ 是关于 $X \cup \tilde{X}$ 的表达式，其中 $\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_n\}$ 表示离散切换结束后系统的状态值。

6) *inv* 为 V 上的不变条件函数。它为每一个模式设定了对应的不变区域(定义域)。$inv(v)$ 是关于 X 的表达式。当系统位于模式 v 时，系统的状态值必须满足 $inv(v)$。

7) *init* 为初始条件函数。它给出了系统的初始状态。

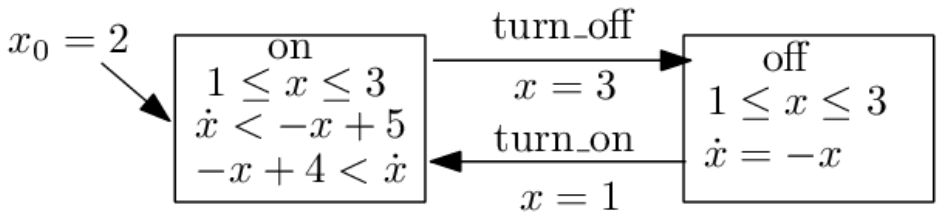


图 2.2 一个简单的混合自动机模型

图 2.2 是一个简单的混合自动机，对应的九个元素分别是：

1) $V = \{v_1, v_2\}$。系统共有两个模式，其中 v_1 表示系统的 *on* 模式，v_2 表示系统的 *off* 模式。

2) $X = \{x\}$。

3) $\Lambda = \{e_1, e_2\}$。其中 $e_1 = (v_1, v_2)$ 表示系统从 v_1 切换到 v_2，$e_2 = (v_2, v_1)$ 表示系统从 v_2 切换到 v_1。

4) $flow(v_1) = \{\dot{x} = -x + 5\}$，$flow(v_2) = \{\dot{x} = -x\}$。

5) $jump(e_1) = \{x = 3 \wedge \tilde{x} = x\}$，$jump(e_2) = \{x = 1 \wedge \tilde{x} = x\}$。

6) $inv(v_1) = \{1 \le x \le 3\}$，$inv(v_2) = \{1 \le x \le 3\}$。

7) $init = (v_1, x_0 = 2)$。系统初始处于 on 模式，初始状态值为 2。

2.3 神经网络

神经网络是人工智能领域一个广泛使用的工具。它具有如下一些性质[55]：非线性、输入输出映射、适应性等。目前学术界已经提出了多种神经网络模型，比较经典的有 BP 神经网络、竞争神经网络等；神经网络在工业界也得到了广泛的应用，如洗衣机、冰箱、啤酒鉴定、市场预测等等。

一个神经元模型如图 2.3，包含 3 个基本元素：突触权重、加法器、激活函数。图中的

$x_1, \cdots, x_l$为输入信号；b_k为偏置，用来调节激活函数的输入，它是可选项，根据实际需要确定；y_k为输出信号。此神经元的数学术语表示即为：

$$u_k = \sum_{i=1}^{l} x_i w_{ki}, \quad y_k = \varphi(u_k + b_k) = f_k(x_1, x_2, \cdots, x_l)$$

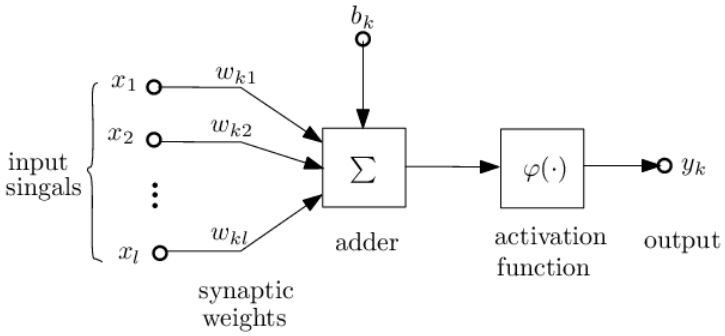


图 2.3 神经元模型

由许多不同的神经元按层次组合构成一个网络，即为神经网络。在神经网络中，一般采用一个实心圆表示一个神经元，例如图 2.4 是一个三层神经网络模型，第一层为输入层，提供两个输入信号；第二层为隐藏层，有 5 个神经元；第三层为输出层，有一个神经元。根据样本数据，神经网络通过不断学习和训练，调整突触权重和偏置值，最终确定神经网络结构，并用来对随机输入的计算和预测。

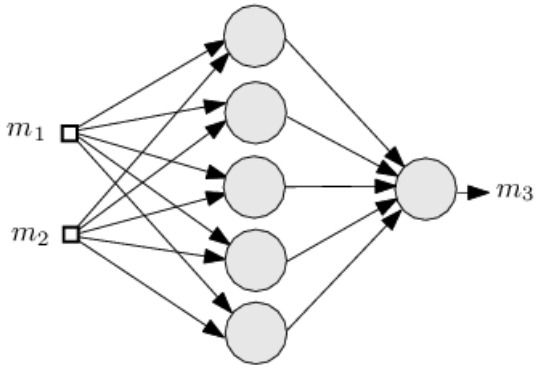


图 2.4 一个 2-5-1 结构的三层神经网络模型

3 第一类自适应软件系统的建模与验证

本文研究的第一类自适应系统是利用控制理论对自适应软件系统进行研究和分析。此时，自适应软件系统由一系列连续的或离散的子系统（或者行为组件）以及协调这些子系统之间切换的规则（控制组件）构成，系统能够检测控制输入变量的变化来选择不同的执行子系统，从而调整系统的行为。这类系统从控制论来说实际上是一种切换系统，在软件实施中专门设计了一个控制组件来实现切换规则。一般来说，两种常用来描述连续子系统的数学工具是常微分方程和随机微分方程。因此本章研究子系统由模糊规则和常微分方程描述的切换模糊系统和子系统由随机微分方程描述的切换随机系统。

3.1 基于模糊行为的自适应系统——切换模糊系统的建模与分析

切换模糊系统是一个多模型系统，它利用一系列模糊子模型和切换规则来描述非线性系统的整体行为。其中一类典型的切换模糊系统是模糊子模型为 Takagi-Sugeno (T–S) 模糊模型，切换规则为有限状态机。T-S 模型由一系列 IF-THEN 模糊规则和常微分方程组组成。研究者对基于 T-S 模型的切换系统的研究主要集中在稳定性的分析和模糊控制的设计，而常用的方法是 Lyapunov 方法。但是如果我们找不到 Lyapunov 函数，我们并不能断定系统不稳定。事实上，根据逆 Lyapunov 定理[57]，一定存在一个 Lyapunov 函数可用来判断系统的稳定性，但是有时我们根本找不到合适的 Lyapunov 函数。因此，本文从软件工程角度出发，首先用混合自动机对切换模型系统进行建模，然后通过 PHAVer 工具进行模型验证来分析其稳定性。下面我们介绍具体的过程。

3.1.1 切换模糊系统

一个 T-S 模型可描述为：

For $l=1,2,\cdots,N$

$$R^l:\ \text{If}\ \ \xi_1\ \ is\ \ M_1^l\wedge\xi_2\ \ is\ \ M_2^l\wedge\cdots\wedge\xi_p\ \ is\ \ M_p^l \qquad 3\text{-}(1)$$

$$\text{Then}\ \ \dot{x}(t)=A_l x(t)+B_l u_{\sigma(t)}(t)$$

其中 N 表示模糊推理规则数，R^l 表示第 l 个推理规则，$u=(u_1,\cdots,u_r)^T$ 是控制输入变量，$x(t)=(x_1(t),\cdots,x_n(t))^T$ 表示状态变量，$\xi(t)=(\xi_1(t),\cdots,\xi_p(t))^T$ 表示规则的前提变量，$M_1^l-M_p^l$ 是前提变量对应的模糊集，$A_l\in R^{n\times n}$，$B_l\in R^{n\times r}$。

设 $\sigma:R^+\to M=\{1,2,\cdots,m\}$ 表示切换信号，它是一个分段常函数；$N_{\sigma(t)}$ 表示 $\sigma(t)$ 子系统的推理规则数，则每一个子系统可描述为：

$\left[\text{Local Plant Rule}\right]$

For $l=1,2,\cdots,N_{\sigma(t)}$

$R_{\sigma(t)}^{l}$: If ξ_1 *is* $M_{\sigma(t)1}^{l}\wedge\xi_2$ *is* $M_{\sigma(t)2}^{l}\wedge\cdots\wedge\xi_p$ *is* $M_{\sigma(t)p}^{l}$

Then $\dot{x}(t)=A_{\sigma(t)l}x(t)+B_{\sigma(t)l}u_{\sigma(t)}(t)$

其中 $N_{\sigma(t)}$、$R_{\sigma(t)}^{l}$ 分别表示第 $\sigma(t)$ 个子系统的总推理规则数和第 l 个推理规则，$u_{\sigma(t)}=\left(u_{\sigma(t)1},\cdots,u_{\sigma(t)r}\right)^T$ 是第 $\sigma(t)$ 个子系统的控制输入变量，$x(t)=\left(x_1(t),\cdots,x_n(t)\right)^T$ 表示状态变量，$\xi(t)=\left(\xi_1(t),\cdots,\xi_p(t)\right)^T$ 表示规则的前提变量，$M_{\sigma(t)1}^{l}-M_{\sigma(t)p}^{l}$ 是前提变量对应的模糊集，$A_{\sigma(t)l}\in R^{n\times n}$，$B_{\sigma(t)l}\in R^{n\times r}$。具体的来说切换模糊系统的第 *i* 个子系统可以表述为：

$$\text{子系统 } i:\begin{cases} R_i^1:\text{if}\quad \xi_1(t)\ \ is\ \ M_{i1}^{1}\wedge\cdots\wedge\xi_p(t)\ \ is\ \ M_{ip}^{1} \\ \qquad \text{then}\quad \dot{x}(t)=A_{i1}x(t)+B_{i1}u_i(t) \\ R_i^2:\text{if}\quad \xi_1(t)\ \ is\ \ M_{i1}^{2}\wedge\cdots\wedge\xi_p(t)\ \ is\ \ M_{ip}^{2} \\ \qquad \text{then}\quad \dot{x}(t)=A_{i2}x(t)+B_{i2}u_i(t) \\ \qquad\qquad \vdots \\ R_i^{N_i}:\text{if}\quad \xi_1(t)\ \ is\ \ M_{i1}^{N_i}\wedge\cdots\wedge\xi_p(t)\ \ is\ \ M_{ip}^{N_i} \\ \qquad \text{then}\quad \dot{x}(t)=A_{iN_i}x(t)+B_{iN_i}u_i(t) \end{cases} \qquad 3\text{-}(2)$$

利用重心法对该子系统进行去模糊化，则子系统 *i* 可以用下式描述：

$$\dot{x}(t)=\sum_{l=1}^{N_i}h_{il}\left(\xi(t)\right)\left[A_{il}x(t)+B_{il}u_i(t)\right] \qquad 3\text{-}(3)$$

其中 $h_{il}(t)\triangleq h_{il}\left(\xi(t)\right)=\dfrac{\prod_{\rho=1}^{p}\mu_{M_{i\rho}^{l}}(t)}{\sum_{l=1}^{N_i}\prod_{\rho=1}^{p}\mu_{M_{i\rho}^{l}}(t)}$，$l=1,2,\cdots,N_i$，$\mu_{M_{i\rho}^{l}}(t)$ 表示 $\xi_\rho(t)$ 隶属于模糊集 $M_{i\rho}^{l}$ 的隶属函数。显然 $0\le h_{il}(t)\le 1,\sum_{l=1}^{N_i}h_{il}(t)=1$。

控制输入变量控制 $u_i(t)$ 可由 PDC 规则确定，即对 *l*=1,2,…,N_i，有

R_i^l:if $\xi_1(t)$ *is* $M_{i1}^{l}\wedge\cdots\wedge\xi_p(t)$ *is* M_{ip}^{l}

then $u_i(t)=-F_{il}x(t)$

其中 $F_{il}\in R^{r\times n}$。因此，子系统 *i* 的控制输入可表示为

$$u(t)=-\sum_{l=1}^{N_i}h_{il}(t)F_{il}x(t)。 \qquad 3-(4)$$

接下来，本文将考虑子系统间的切换规则，我们用确定有限状态机来刻画。令Ω是整个切换模糊系统的输入空间，将输入空间划分为 m 部分，记为$\Omega_1,\Omega_2,\cdots,\Omega_m$，满足$\Omega_i\subset R^n$且$\bigcup_{i=1}^{m}\Omega_i=\Omega,\ \Omega_i\bigcap\Omega_j=\varnothing,i\neq j$。每一个子空间$\Omega_i$对应子系统 i 的输入空间。于是我们可以得到一个有限状态机，每个子空间对应状态机的一个模式。

因此全局切换规则为：

[Region Rule]

if $\xi_1(t)\in\Omega_{i1}\wedge\cdots\wedge\xi_p(t)\in\Omega_{ip}$, then

[Local Plant Rule]

其中$\Omega_i=\Omega_{i1}\times\cdots\times\Omega_{ip}\times\cdots\times\Omega_{in}$，$\Omega_{ij}$是一个确定集。于是，当状态变量$x(t)$满足$\xi_1(t)\in\Omega_{i1}\wedge\cdots\wedge\xi_p(t)\in\Omega_{ip}$时，则有$\sigma(t)=i$。不同子系统间的切换由“切换面”确定。不同系统可以定义不同的“切换面”。当系统状态轨迹$x(t)$到达切换面S_{ij}时，系统将从Ω_i子系统切换到Ω_j子系统，并通过函数$reset(x)$来确定进入Ω_j子系统的初始状态。

通过去模糊化，我们将一个切换模糊系统转化成一个一般的切换系统。此时系统还是抽象的数学表达式，因此我们将该切换系统转化成混合自动机，然后在进行模型检查。

3.1.2 切换模糊系统的混合自动机建模

在 3.1.1 节，我们通过去模糊化将一个模糊切换系统化为一个一般的切换系统。本节将讨论将获得的切换系统转化为混合自动机。具体规则如下：

***Rule* 1**：设$\Omega_1,\Omega_2,\cdots\Omega_n$是切换系统所划分的 n 个区域，每个区域对于一个子系统。因此，我们将每一个子系统映射为自动机的一个模式，其中Ω_i子系统对应模式v_i。

***Rule* 2**： 切换系统的状态变量$x=(x_1,x_2,\cdots,x_n)$即为自动机的状态变量。为与自动机的定义相一致，本文将这些变量重新用 $X=\{x_1,x_2,\cdots,x_n\}$ 表示；在不引起歧义的条件下，本文并不区分 x 和 X 。

***Rule* 3**： 如果在切换系统中存在切换面S_{ij}，则在混合自动机中存在切换(e_i,e_j)。

***Rule* 4**：切换系统中Ω_i区域内的状态方程确定混合自动机中 v_i 的流条件 *flow*。例如Ω_i区域内，状态的控制方程为$\dot{x}(t) = Ax(t) + Bu(t)$，则对应的模式 v_i 的流条件为$\dot{X}(t) = AX(t) + Bu(t)$，即 $flow(v_i) = \{\dot{X}(t) = AX(t) + Bu(t)\}$。

***Rule* 5**：切换系统中的切换面确定自动机中的跳跃条件 *jump*。设S_{ij}是从Ω_i到Ω_j的切换面，且切换后的新状态由$reset(x)$确定，则$jump(v_i, v_j) = \{S_{ij} \wedge \tilde{X} = reset(X)\}$。

***Rule* 6**：切换系统中空间划分Ω_i确定自动机的不变条件*inv*，即$inv(v_i) = \Omega_i$

***Rule* 7**：切换系统中初始条件确定自动机中的初始条件*init*。如果切换系统中的初始条件为(Ω_i, x_0)，则对应的自动机的初始条件为$init = (v_i, X_0 = x_0)$。

通过上述的转化规则，我们将一个切换系统转化为了一个混合自动机。于是有以下定理：

定理 3-1：通过 Rule 1-Rule 7 获得的混合自动机与原切换系统是一致的，即它们具有相同的状态数、相同的状态迁移结构、相同的状态迁移条件以及相同的初始状态。

3.1.3 混合自动机的模型验证——稳定性检验

本节首先简单介绍一下本文使用的模型检查工具 PHAVer[1]，全称为 Polyhedral Hybrid Automaton Verifyer。PHAVer 具有如下性质[56]：基于 Parma Polyhedra Library 的准确、稳健的算法；利用 on-the-fly 技术对按段仿射行为进行过近似；多面体计算的约束；支持组合推理和假设推理。此外，该研究团队同时提供了对 2 维输出的多边形的可视化描述的 matlab 脚本。PHAVer 可以进行可达分析和模拟验证。其输入是一个线性混合自动机或者仿射混合自动机和初始状态集合，根据不同命令函数可以获得不同输出，例如命令 aut_ident.reachable 输出自动机 aut_ident 的可达状态，get_sim(aut_ident1,aut_ident2)可以获得自动机 aut_ident1 和 aut_ident2 的模拟关系。本文使用 PHAVer 进行可达状态分析，输出的可达状态由多面体表示。

由于 PHAVer 的输入是仿射混合自动机，而本文需验证的是系统的稳定性。因此，本文首先需要将系统的稳定性描述转为混合自动机的可达性描述。

定义 3-1：切换模型系统的一个状态x^*称为稳定的，如果$\forall\varepsilon > 0$，$\exists\delta(\varepsilon)$，只要初值条件$x_0$满足$|x_0 - x^*| \le \delta(\varepsilon)$，则由初值条件$x_0$确定的解$x(t)$满足$|x(t) - x^*| < \varepsilon$，其中$t \ge 0$；$x^*$称为渐进稳定的，如果$x^*$是稳定的，且存在$\delta_0$，使得当$|x_0 - x^*| < \delta_0$时，满足初始条件的解

[1] 下载地址：http://www-verimag.imag.fr/~frehse/phaver_web/

$x(t)$均有 $\lim_{t\to+\infty} x(t)=x^*$。

定义 3-2：一个状态(v,p)称为混合自动机的**可接受状态**，如果有 $p\in inv(v)$；状态(v,p)称为混合自动机的初始状态，如果$(v,p)=init$。混合自动机的一条**轨迹**是有限个或者无限个可接受状态序列$\{q_j\}(j=0,1,\cdots)$满足如下两个条件：1) q_0是初始状态，2）序列中前后两个相继状态(q_j,q_{j+1})或者是由流条件引起的变迁或者是由跳跃条件引起的变迁。一个状态称为可达的，如果存在某条轨迹经过该状态。

定义 3-3：一个混合自动机是稳定的，如果存在一个确定状态(v_i,p^*)，其中$p^*=(x_1^*,\ x_2^*,\cdots,x_n^*)\in inv(v_i)$,满足 1) $\forall\varepsilon>0$，$T=\{X\,\|\,\|X-p^*\|<\varepsilon\}\subseteq inv(v_i)$，存在$\delta(\varepsilon)$，使得对于任意初始状态在$\{X\,\|\,\|X-p^*\|\le\delta\}$内的轨迹都在 T 内；2) 存在$\delta_0$，使得对任意初始状态在$\{X\,\|\,\|X-p^*\|\le\delta_0\}$内的轨迹$X(t)$满足$\lim_{t\to+\infty}X(t)=p^*$。

设 SFS 表示某个切换模糊系统，*HA*表示由 SFS 转化得到的混合自动机。于是有如下定理：

定理 3-2：HA 的稳定性与 SFS 的渐进稳定性等价。

由于现有的模型检查工具只适用于线性自动机或者仿射线性自动机，因为本文通过近似，用一个仿射混合自动机来近似非线性混合自动机。然后利用 PHAVer 进行验证。所谓仿射混合自动机是自动机的流条件是关于 X 和 $\dot{X}$ 的线性组合或者线性不等式，例如图 3.1 所表示的自动机就是一个仿射混合自动机。

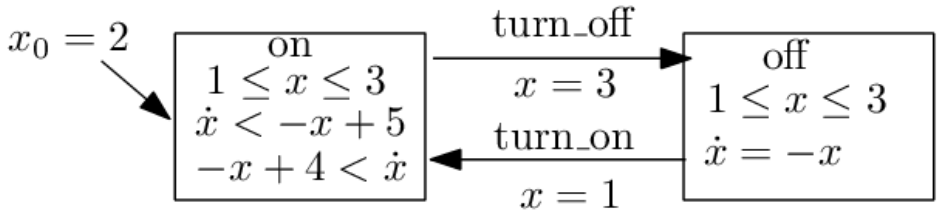


图 3.1 一个简单的混合自动机模型

类似于用直线代替曲线，对于非线性混合自动机，本文通过对输入变量的划分，将原先的非线性方程转化 X 和 $\dot{X}$ 的线性不等式，此时得到的状态空间将大于原来的状态空间，即仿射自动机的可达状态包含原先的混合自动机的所有可达状态，但是当分割越细（以导数的距离衡量），两者的可达状态就越接近[56]。令 P_n 为输入空间第 n 次分割后所有子空间中的最大导数距离值，其对应的仿射混合自动机为 HAA_n，则有当$|P_n|\to 0$时，有 $HAA_n\to HA$，

记作 $HAA_n \xrightarrow{P_n} HA$。

定义 3-4：称 HAA_n 是稳定的，如果能找到一个可达状态的真子集 D，使得任何一个轨迹都能到达 D 并最终终止于 D。称区域 D 为稳定区域。

定理 3-3：设 HAA_n 是一个仿射混合自动机序列，满足 $HAA_n \xrightarrow{P_n} HA$。如果 $\forall n$，HAA_n 是稳定的，则 HA 是稳定的。

因此，切换模糊系统的稳定性验证问题，就转化为求稳定区域。设 g 是初始状态到可达状态的映射，D 是稳定区域， 则有 $g(D)=D$。其中映射 g 就是根据初始状态求可达状态，这恰好是 PHAVer 的功能，可以用 PHAVer 实现。

根据上述，本文可归纳得到如何利用 PHAVer 进行稳定性验证。具体步骤如下：

Step 1：给定误差 ε，用仿射混合自动机对非线性自动机近似，得到一个 HAA；

Step 2：利用 PHAVer 计算可达状态。PHAVer 所计算输出的可达状态是有限的多面体（多边形）。

Step 3：确定稳定区域。对 Step 2 产生的多面体进行如下操作：

Step 3.1：选择一个多面体作为自动机的初始状态，运行 PHAVer 计算得到新的可达状态 D，判断新产生的可达状态是否满足 $g(D)=D$，若是，结束；否则转 Step 3.2.

Step 3.2：重新选取一个多面体，转 Step 3.1.

3.2 基于随机行为的自适应系统——切换随机系统的建模与分析

切换随机系统已经广泛地应用于工业界了，它是一类混合系统。一般来说，主要有两类切换随机系统：随机性位于系统的连续变化部分或者随机性加在系统的离散切换部分。本节介绍的是前者，即系统的连续行为由 Itô 随机微分方程描述。人们对这类系统的建模主要使用 Markov 链或者随机微分方程，并根据数值解对模型进行分析。但是这些研究都是基于分析模型，而忽略了对这类系统的软件实现，因而没能提出形式化模型来对其进行建模。因此，本节将从软件理论角度提出了一种新的 Petri 网—随机微分 Petri 网来对这类切换随机系统进行形式化建模，该模型既能对切换随机系统进行形式化建模，又能进行模型检查。

3.2.1 切换随机系统的数学描述

对于切换随机系统，其子系统(连续部分)由随机微分方程描述，子系统间的切换由 Markov 链控制。下面将对它们分别进行描述。

考虑 n 维非线性 Itô 随机微分方程

$$\begin{cases} \mathrm{d}x(\tau)=b\big(x(\tau)\big)\mathrm{d}\tau+\sigma\big(x(\tau)\big)\mathrm{d}B(\tau),\tau\geq 0 \\ x(0)=x_0 \end{cases} \qquad 3\text{-}(5)$$

其中 $B(\tau)$ 是 m 维标准布朗运动；$b:R^n\to R^n$，$\sigma:R^n\to R^{n\times m}$ 满足随机微分方程的解的存在唯一性定理，即对任意的 $\tau>0,\tau_1>0,\tau_2>0$，存在常数 C,D，使得

$$\big|b\big(x(\tau)\big)\big|+\big|\sigma\big(x(\tau)\big)\big|\leq C\big(1+\big|x(\tau)\big|\big) \qquad 3\text{-}(6)$$

和

$$\big|b\big(x(\tau_1)\big)-b\big(x(\tau_2)\big)\big|+\big|\sigma\big(x(\tau_1)\big)-\sigma\big(x(\tau_2)\big)\big|\leq D\big|x(\tau_1)-x(\tau_2)\big| \qquad 3\text{-}(7)$$

成立，其中 $|\bullet|$ 表示欧几里得范数。

由于随机微分方程很难求解，因此一般进行数值求解。因此，本文首先利用 Euler-Maruyama 方法对随机微分方程进行离散化[59]。给定时间间隔 $\Delta\tau(>0)$，令 $\tau_k=k\Delta\tau$，$X_k=x(\tau_k)$，其中 $X_0=x_0$，于是有

$$X_{k+1}=X_k+b(X_k)\Delta\tau+\sigma(X_k)\Delta B_k \qquad 3\text{-}(8)$$

其中 $\Delta B_k=B(\tau_{k+1})-B(\tau_k)$。

现在考虑切换随机系统。设 $r(\tau),\tau\geq 0$ 是一个右连续 Markov 链，其状态空间 $S=\{1,2,\cdots,N\}$。则切换随机系统可以描述为：

$$\begin{cases} \mathrm{d}x(\tau)=b\big(x(\tau),r(\tau)\big)\mathrm{d}\tau+\sigma\big(x(\tau),r(\tau)\big)\mathrm{d}B(\tau),\tau\geq 0 \\ x(0)=x_0,r(0)=i_0 \end{cases} \qquad 3\text{-}(9)$$

其中

1) $r:R\to S$ 是一个 Markov 链，每个状态代表一个子系统，其成生子矩阵为 $\Gamma=(\gamma_{ij})_{N\times N}$，满足 $\gamma_{ij}>0(i\neq j),\gamma_{ii}=-\sum_{j\neq i}\gamma_{ij}$，其中 γ_{ij} 称为状态 i 到状态 j 的转移速率，转移概率

$$P\big(r(t+\delta)=j\,|\,r(t)=i\big)=\begin{cases}\gamma_{ij}\delta+o(\delta),i\neq j\\ 1+\gamma_{ij}\delta+o(\delta),i=j\end{cases}$$

2) $b:R^n\times S\to R^n$，$\sigma:R^n\times S\to R^{n\times m}$。

类似于一般的随机微分方程离散，接来下对切换随机微分方程进行离散化。首先对连续时间 Markov 链进行离散。假设时间间隔为 $\Delta\tau(>0)$，令 $\tau_k=k\Delta\tau$，$r_k^{\Delta\tau}=r(\tau_k)$，则

$\left\{r_k^{\Delta\tau}, k=0,1,\cdots\right\}$ 是一个离散 Markov 链，其一步迁移概率矩阵为

$$P(\Delta\tau)=\left(P_{ij}(\Delta\tau)\right)_{N\times N},\quad P_{ij}(\Delta\tau)=\begin{cases}\gamma_{ij}\Delta\tau+o(\Delta\tau), i\neq j\\ 1+\gamma_{ij}\Delta\tau+o(\Delta\tau), i=j\end{cases}。$$

下面计算该 Markov 链的状态序列[58]：令 $r_0^{\Delta\tau}=i_0$，产生一个[0,1]上均匀分布的随机数 ξ_1。

定义

$$r_1^{\Delta\tau}=\begin{cases}i_1, 如果存在 i_1\in S-\{N\} 且 \sum_{j=1}^{i_1-1}P_{i_0 j}(\Delta\tau)\le\xi_1<\sum_{j=1}^{i_1}P_{i_0 j}(\Delta\tau)\\ N, 如果 \sum_{j=1}^{N-1}P_{i_0 j}(\Delta\tau)\le\xi_1\end{cases}$$

然后重新产生一个独立的新的[0,1]上均匀分布的随机数 ξ_2，并定义

$$r_2^{\Delta\tau}=\begin{cases}i_2, 如果存在 i_2\in S-\{N\} 且 \sum_{j=1}^{i_2-1}P_{i_1 j}(\Delta\tau)\le\xi_2<\sum_{j=1}^{i_2}P_{i_1 j}(\Delta\tau)\\ N, 如果 \sum_{j=1}^{N-1}P_{i_1 j}(\Delta\tau)\le\xi_2\end{cases}$$，如此重复，假设已经产生

$r_0^{\Delta\tau}, r_1^{\Delta\tau}, \cdots r_k^{\Delta\tau}$，产生一个独立的新的[0,1]上均匀分布的随机数 ξ_{k+1}，并定义

$$r_{k+1}^{\Delta\tau}=\begin{cases}i_{k+1}, 如果存在 i_{k+1}\in S-\{N\} 且 \sum_{j=1}^{i_{k+1}-1}P_{r_k^{\Delta\tau} j}(\Delta\tau)\le\xi_{k+1}<\sum_{j=1}^{i_{k+1}}P_{r_k^{\Delta\tau} j}(\Delta\tau)\\ N, 如果 \sum_{j=1}^{N-1}P_{r_k^{\Delta\tau} j}(\Delta\tau)\le\xi_{k+1}\end{cases}$$。于是可以得到

$\left\{r_k^{\Delta\tau}, k=0,1,\cdots\right\}$。

其次，对切换随机微分方程进行离散。令 $X_k=x(\tau_k)$，其中 $X_0=x_0, r_0^{\Delta\tau}=i_0$，于是有

$$X_{k+1}=X_k+b\left(X_k, r_k^{\Delta\tau}\right)\Delta\tau+\sigma\left(X_k, r_k^{\Delta\tau}\right)\Delta B_k \qquad 3-(10)$$

其中 $\Delta B_k=B(\tau_{k+1})-B(\tau_k)$。

注[59]：EM 方法强收敛于 3-（9）的解，其收敛阶为 0.5.

3.2.2 切换随机系统的 Petri 网建模

首先，本文用 Petri 网对切换逻辑——连续 Markov 链进行建模。令 $\Gamma=\left(\gamma_{ij}\right)_{N\times N}$ 为连续 Markov 链 $r(t)$ 的生成子矩阵。将状态空间中的每一个状态映射 s_i 为 Petri 网中的一个库所 p_i；如果存在从状态 s_i 到 s_j 的迁移速度 $\gamma_{ij}>0$，则在 Petri 网中存在变迁 t_{ij} 和弧(p_i，t_{ij})和弧

(t_{ij}, p_j)，其中变迁的平均点火速度为γ_{ij}。因此有图 3.2 中的 5 种元结构变换：

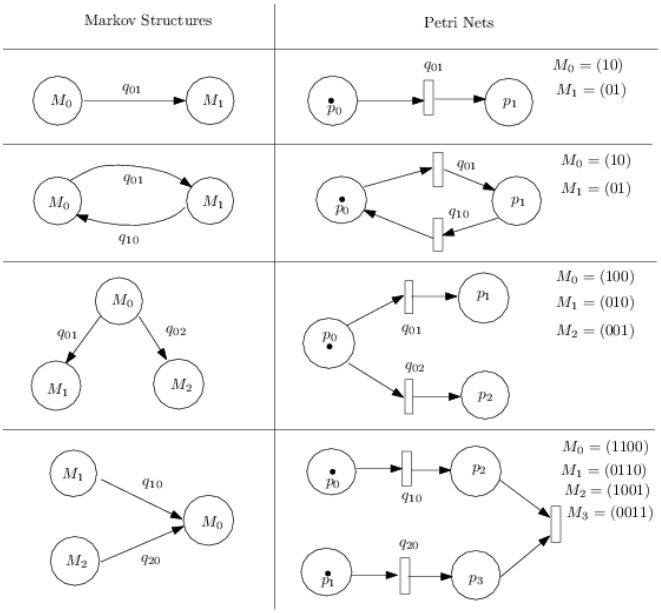


图 3.2 Markov 链中 5 种元结构到 Petri 网映射

如此，我们可以将任何一个 Markov 链转化为随机 Petri 网。

接下来本文将对每个子系统进行形式化建模。由于每个子系统是有随机微分方程描述的，因此为方便起见，在下面的讨论中，我们仅考虑方程 3-（5）的形式，而忽略切换逻辑信号$r(t)$。不失一般性，考虑方程

$$\begin{cases} \mathrm{d}x(\tau) = b\big(x(\tau)\big)\mathrm{d}\tau + \sigma\big(x(\tau)\big)\mathrm{d}B(\tau), \tau \geq 0 \\ x(0) = x_0 \end{cases} \qquad 3\text{-（11）}$$

其中布朗运动也是一维的，其离散形式为：

$$X_{k+1} = X_k + b(X_k)\Delta\tau + \sigma(X_k)\Delta B_k \qquad 3\text{-（12）}$$

为此，可构造图 3.3 所示 Petri 网

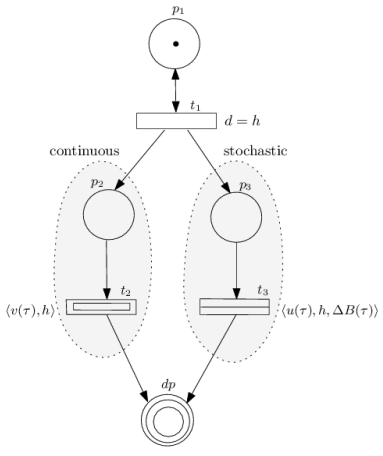


图 3.3 对随机微分方程$\mathrm{d}x(\tau) = b\big(x(\tau)\big)\mathrm{d}\tau + \sigma\big(x(\tau)\big)\mathrm{d}B(\tau)$的 Petri 建模

图中所示的左半部分是连续变化部分，右半部分是随机变化部分。

- 库所 dp 是一个随机微分库所，dp 中的标识值 m_{dp} 表示随机微分方程的变量值，即 X 的值；dp 的初始标识为方程的初始值，即 $m_{dp}(0)=x_0$。库所 p_1, p_2, p_3 是离散库所，初始标识为 $m_d(0)=(1,0,0)$。因此上述 Petri 网的初始标识为 $m_0=(1,0,0,x_0)$
- 变迁 t_1 是一个离散变迁，同时关联了一个时间延迟 h，其中 h 为随机微分方程的离散时间步长，$h=\Delta\tau$。如此，t_1 每隔时间 h 将点火实施一次。
- 变迁 t_2 是微分变迁，其在时刻 τ 的实施速度为 $v(\tau)=b\left(m_{dp}(\tau)\right)$，同时还关联了一个时间延迟 h。实际上该时间延迟通过一个隐含的离散变迁实现，如图 3.4，其中隐含了一个离散库所 p 和一个离散变迁 t，该离散变迁一直处于激活（enabling）状态，但是每经过时间 h 实施（firing）一次。因此实际上，一旦微分变迁符合激活条件，就处于激活状态，但需要经过时间 h 才能实施一次，这恰好体现了微分部分的连续性和离散后的跳跃性。当微分变迁实施后，dp 的标识变化方程为：$m_{dp}(\tau+h)=m_{dp}(\tau)+v(\tau)h$

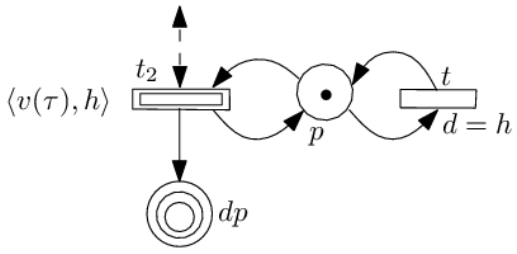


图 3.4 微分变迁的实现

- 变迁 t_3 是一个随机变迁，它关联了一个三元组 $\langle u(\tau), h, \Delta B(\tau)\rangle$，其中 $u(\tau)$ 是 t_3 在时刻 τ 的实施速度，且 $u(\tau)=\sigma\left(m_{dp}(\tau)\right)$；$h$ 为时间延迟，类似于微分变迁，它实际关联在一个隐含的离散变迁上，如图 3.5。

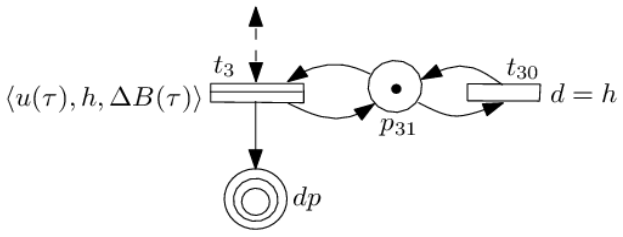


图 3.5 随机变迁的实现

t_{30} 关联了一个时间延迟 h，因此 t_{30} 一直处于激活状态，但是只能每隔时间 h 实施一次，于是实现随机部分的连续增加和离散跳跃；$\Delta B(\tau)$ 为布朗运动在时间 h 内的随机增量，且服从标准正态分布，即 $\Delta B(\tau)\sim N(0,h)$。当 t_3 实施后，dp 的标识变化方程

为：

$$m_{dp}(\tau+h)=m_{dp}(\tau)+u(\tau)\Delta B(\tau)$$

接下来再来看一下图 3.3 中 Petri 网的实施过程。当 t_1 实施后，连续部分和随机部分同时激活并同步实施，然后同时引起随机微分库所的标识变化，因此，可得到随机微分库所 *dp* 的 marking 变化方程为：

$$\begin{aligned} m_{dp}(\tau+h) &= m_{dp}(\tau)+v(\tau)h+u(\tau)\Delta B(\tau) \\ &= m_{dp}(\tau)+b\left(m_{dp}(\tau)\right)h+\sigma\left(m_{dp}(\tau)\right)\Delta B(\tau) \end{aligned} \qquad 3\text{-（13）}$$

将 τ 换成 τ_i，并记 $m(\tau_i)=m(i)$，则有

$$\begin{aligned} m_{dp}(i+1) &= m_{dp}(i)+v(\tau_i)h+u(\tau_i)\Delta B(\tau_i) \\ &= m_{dp}(i)+b\left(m_{dp}(i)\right)h+\sigma\left(m_{dp}(i)\right)\Delta B(i) \end{aligned} \qquad 3\text{-（14）}$$

因为随机微分库所的标识表示的是状态变量，即 $X_i=X(\tau_i)=m(\tau_i)=m(i)$，$h=\Delta\tau$，因此方程 3-（14）即为

$$X_{k+1}=X_k+b(X_k)\Delta\tau+\sigma(X_k)\Delta B_k \qquad 3\text{-（15）}$$

这就是随机微分方程的离散形式 3-（8）。因此，该 Petri 网能够用来描述随机微分方程。

整个的切换随机系统是控制逻辑和有限子系统的合成。如前所述，控制逻辑由连续 Markov 链描述，通过离散 Petri 网进行建模；子系统由混合 Petri 网进行建模。合成模型如图 3.6。

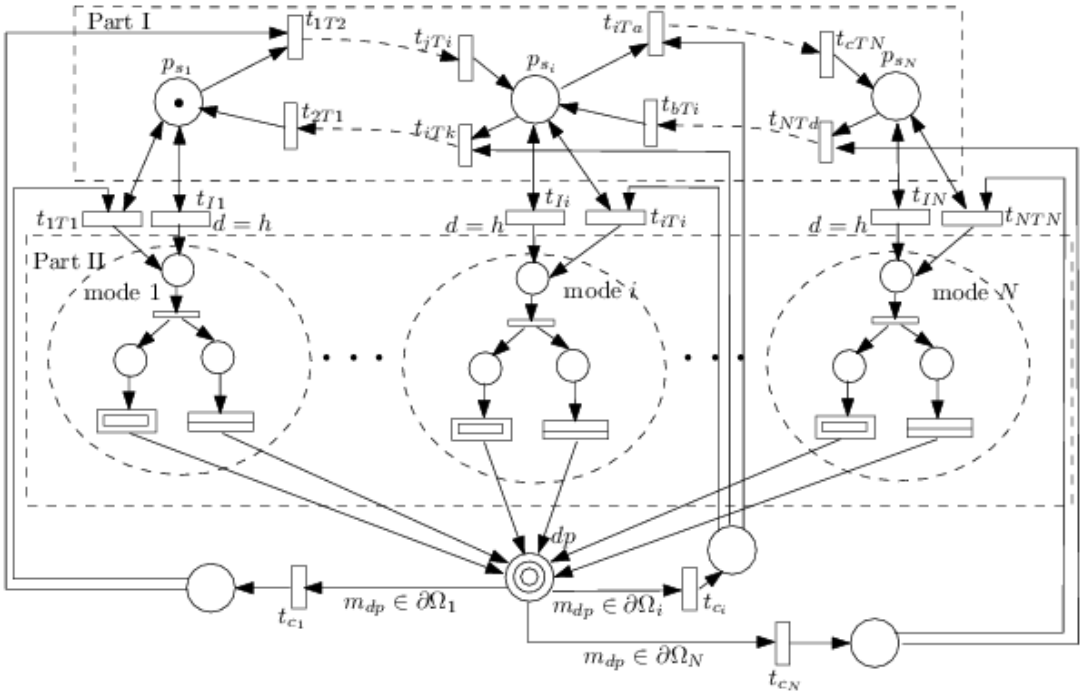


图 3.6 切换随机系统的 Petri 网模型

图中的 Part I 控制逻辑，Part II 是各个子系统，每个虚线椭圆表示一个子系统，*dp* 是随机微分库所；变迁 t_{c1}-t_{cN}，t_{1T1}-t_{NTN} 均为瞬时变迁，其实施优先级高于有时间延迟的延时变迁（有时也称为时间变迁）。变迁 t_{c1}-t_{cN} 只有当 m_{dp} 达到各自子空间的边界时才可激活，否则 Petri 网只在各自子系统内发生状态变化。

3.2.3 随机微分 Petri 网及其语义

本节将根据定义一种新的 Petri 网，使得该网能够用来对切换随机系统进行建模，而又不局限于对切换随机系统建模。

定义 3-5：随机微分 Petri 网是一个六元组 $SDPN=(P,T,F,f,w,J)$，其中

1） $P=P_d\cup P_\wp$ 是有限库所集合， P_d 是离散库所集， $P_\wp$ 是随机微分库所集。

2）$T=T_d\cup T_D\cup T_\wp$ 是有限变迁集合，T_d 是离散变迁集，T_D 是微分变迁集，$T_\wp$ 是随机微分变迁集。

3）$F\subseteq(P\times T)\cup(T\times P)$ 是连接不同节点的有向弧。

4） $f:P\cup T\to\{d,D,\wp\}$ 是类型函数，它指定了每一个库所和变迁的类型。

5） $w:F\to R$ 为权重函数，为每一条弧分配一个权重值。 $Pre(p_i,t_j)=w(p_i,t_j)$ 是表示从库所 p_i 指向变迁 t_j 的弧的权重， $Post(p_i,t_j)=w(t_j,p_i)$ 表示从变迁 t_j 指向库所 p_i 的弧的权重。

当 $f(p_i)=d$ 时， $Pre(p_i,t_j)\in\mathbb{Z}^+$， $Post(p_i,t_j)\in\mathbb{Z}^+$；当 $f(p_i)=\wp$ 时， $Pre(p_i,t_j)\in\mathbb{R}$，$Post(p_i,t_j)\in\mathbb{R}$。$W=[w_{ij}], w_{ij}=Post(p_i,t_j)-Pre(p_i,t_j)$。、

6） J 是时间映射函数，定义为：

$$J(t_j,\tau)=\begin{cases} d_j, & f(t_j)=d \\ \langle v_j(\tau),h\rangle, & f(t_j)=D \\ \langle u_j(\tau),h,\Delta B_j(\tau)\rangle, & f(t_j)=\wp \end{cases}$$

这里，$d_j=0$或者h，表示离散变迁的时间延迟；h 是随机微分方程离散化的时间间隔；$v_j(\tau)$ 表示微分变迁 t_j 在时刻 τ 的实施速度，它是关于随机微分库所的标识的函数；$u_j(\tau)$ 表示随机微分变迁 t_j 在时刻 τ 的实施速度，它是关于随机微分库所的标识的函数；$\Delta B_j(\tau)\sim N(0,h)$ 表示布朗运动在时间间隔 h 内的随机增量。

定义 3-6：标识随机微分 Petri 网是一个二元组 $(SDPN,m_0)$，其中 $SDPN$ 是一个随机微分 Petri 网， m_0 是初始标识。

令 $m(\tau)$ 为随机微分 Petri 网在时刻 τ 的标识向量，$m_d(\tau)$ 为离散库所所组成的标识向量，$m_\wp(\tau)$ 为随机微分库所所组成的标识向量，则 $m(\tau)=\begin{pmatrix} m_d(\tau) \\ m_\wp(\tau) \end{pmatrix}$。

下面首先给出各类变迁的激活条件和一些优先级。记 ${}^\bullet t_j$ 为 t_j 的所有输入库所集合，$t_j^\bullet$ 为 t_j 的所有输出库所集合，则有

1）d-变迁 t_j 的激活条件为 $\forall p_i\in {}^\bullet t_j$， $m_i(\tau)\geq Pre(p_i,t_j)$

2）D-变迁或者 $\wp$-变迁 t_j 的激活条件为 $\forall p_i\in P_d\cap {}^\bullet t_j$， $m_i(\tau)\geq Pre(p_i,t_j)$；

3）D-变迁和 $\wp$-变迁的优先级高于 d-变迁，瞬时变迁的优先级高于有延时的离散变迁。

有了激活条件后，不同的变迁将被激活并由优先级从高到低并发实施，从而产生该网标识的变化。Petri 网的标识的具体变化按下述方程进行：

1）当变迁 t_j 在时刻 τ 实施时，其中 $f(t_j)=d, J(t_j)=d_j$，有

$$m_i(\tau+d_j)=\begin{cases} m_i(\tau)-w(p_i,t_j), & \forall p_i\in {}^\bullet t_j \\ m_i(\tau)+w(t_j,p_i), & \forall p_i\in t_j^\bullet \\ m_i(\tau), & \text{其他} \end{cases} \quad 3\text{-}(16_1)$$

2）当变迁 t_j 在时刻 τ 实施时，其中 $f(t_j)=D, J(t_j)=\langle v_j,h\rangle$，有

$$m_i(\tau+h)=\begin{cases} m_i(\tau)-v_j(\tau)w(p_i,t_j)h, & \forall p_i\in {}^\bullet t_j \\ m_i(\tau)+v_j(\tau)w(t_j,p_i)h, & \forall p_i\in t_j^\bullet \\ m_i(\tau), & \text{其他} \end{cases} \quad 3\text{-}(16_2)$$

3）当变迁 t_j 在时刻 τ 实施时，其中 $f(t_j)=\wp, J(t_j)=\langle u_j,h,\Delta B_j\rangle$，有

$$m_i(\tau+h)=\begin{cases} m_i(\tau)-u_j(\tau)w(p_i,t_j)\Delta B_j(\tau), & \forall p_i\in {}^\bullet t_j \\ m_i(\tau)+u_j(\tau)w(t_j,p_i)\Delta B_j(\tau), & \forall p_i\in t_j^\bullet \\ m_i(\tau), & \text{其他} \end{cases} \quad 3\text{-}(16_3)$$

因此，对任意一个随机微分变迁，有如下的状态转移方程：

$$\begin{aligned} m_i(\tau+h) &= m_i(\tau)+\sum_{t_j\in T_D}(w(t_j,p_i)-w(p_i,t_j))v_j(\tau)h \\ &+ \sum_{t_j\in T_\wp}(w(t_j,p_i)-w(p_i,t_j))u_j(\tau)\Delta B_j \end{aligned} \quad 3\text{-}(17)$$

当 $h\to 0$ 时，方程 3-（17）的积分表达式为

$$\begin{aligned} m_i(\tau+h)=m_i(\tau) &+ \sum_{t_j\in T_D}(w(t_j,p_i)-w(p_i,t_j))\int_\tau^{\tau+h} v_j(s) \\ &+ \sum_{t_j\in T}(w(t_j,p_i)-w(p_i,t_j))\int_\tau^{\tau+h} u_j(s)dB(s) \end{aligned} \quad 3\text{-}(18)$$

于是可以得到 S-DPN 的基本方程：

$$\begin{aligned} m(\tau_k) &= m(\tau_i)+\tilde{W}\cdot[(\delta_d(\tau_k),0,0)^T \\ &+ (0,\int_{\tau_i}^{\tau_k} v(s)ds,\int_{\tau_i}^{\tau_k} u(s)dB(s))^T\otimes\delta(\tau_k)] \end{aligned} \quad 3\text{-}(19)$$

其中，$\tilde{W}=\begin{pmatrix} W_{dd} & W_{dD} & W_d \\ W_d & W_D & W \end{pmatrix}$，$\delta(\tau_k)=(\delta_d(\tau_k),\delta_D(\tau_k),\delta(\tau_k))^T$，$\delta_d(\tau_k)$, $\delta_D(\tau_k)$, $\delta(\tau_k)$分别表示离散变迁、微分变迁和随机微分变迁的实施序列，$\otimes$表示两个向量的对应元素的乘积。

根据上述标识的变化方程，我们就可以获得随机微分 Petri 网的可达图，具体可由下面算法得到：

算法 1：随机微分 Petri 网的可达图计算算法

输入：随机微分 Petri 网 SDPN；输出：可达图 EG

Step 1：根据初始标识初始化各库所，设定时间间隔

Step 2：将初始标识作为根节点

Step 3：任意选取 EG 中无叶子节点的标识，计算所有可实施的变迁；

Step 4：根据标识变化方程，计算所有可实施的变迁实施后的新标识；

Step 5：将这些新标识作为该节点的直接叶节点加入到 EG 中，并返回 Step 3

此外有：

定理 3-4：随机微分方程 S-DPN 是切换随机微分方程的离散近似，且 S-DPN 的可达状态空间收敛于切换随机微分方程的解空间，且其收敛阶为 0.5。

3.2.4 随机微分 Petri 网的验证

本节将利用 PRISM[2]进行模型验证。PRISM 是一个概率模型检查器，可以用来对具有随机或概率行为的系统进行分析。PRISM 可以分析的概率模型有离散时间 Markov 链 (DTMCs)，连续时间 Markov 链 (CTMCs)，Markov 决策过程 (MDPs)，概率自动机 (PAs)，概率时间自动机 (PTAs)，以及这些模型的扩展结构，可以用来验证由时序逻辑 PCTL, CSL, LTL 和 PCTL*等描述的定量性质。

为了能够使用 PRISM 进行分析，本文首先介绍如何获得与随机微分 Petri 同构的 Markov 链，并利用该 Markov 链进行性质的验证，其流程图如图 3.7 所示。

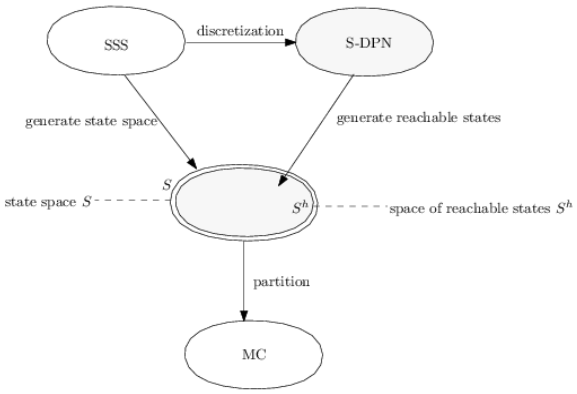


图 3.7 获得同构 Markov 链的过程

本文将通过切换随机系统的状态空间的划分来构造一个 Markov 链，然后证明该 Markov 链与随机微分 Petri 网同构，即随机微分 Petri 网的每个标识映射到该 Markov 链的一个状态；Petri 网可达图同构（通过对原始可达图进行等价类划分后的可达图）与该 Markov 链的状态空间。接下来将介绍具体过程。

[2] PRISM 官网地址：http://www.prismmodelchecker.org/

首先由切换随机微分方程（SSS）的状态空间的划分获得一个 Markov 链。该过程已在文献[60]中详细叙述了，这里就不再具体描述，本文只给出结果。首先对状态空间进行均匀网格划分，网格长度为δ，因此可以得到状态空间$S_\delta=\left\{\eta:\eta=\delta\sum_{i=1}^{n}e_iu_i\right\}$，其中$e_i$是一个单位向量满足第 i 个元素为 1，其他元素为 0，$u_i=0,\pm1,\pm2,\cdots(i=1,2,\cdots,n)$。于是可以构造一个 Markov 链，其状态空间为S_δ，状态转移概率为

$$Q^\delta(x)=\sum_{i=1}^{n}\left[a_{ii}(x)-\sum_{j:j\neq i}^{n}\frac{|a_{ij}(x)|}{2}\right]+\delta\sum_{i=1}^{n}|b_i(x)|,\quad \Delta\tau^\delta(x)=\frac{\delta^2}{Q^\delta(x)}$$

$$p^\delta(x,x+\delta e_i)=\frac{a_{ii}(x)-\sum_{j:j\neq i}^{n}\frac{|a_{ij}(x)|}{2}+\delta b_i^+(x)}{Q^\delta(x)},$$

$$p^\delta(x,x-\delta e_i)=\frac{a_{ii}(x)-\sum_{j:j\neq i}^{n}\frac{|a_{ij}(x)|}{2}+\delta b_i^-(x)}{Q^\delta(x)},$$

$$p^\delta(x,x+\delta e_i+\delta e_j)=p^\delta(x,x-\delta e_i-\delta e_j)=\frac{a_{ij}^+(x)}{Q^\delta(x)},$$

$$p^\delta(x,x-\delta e_i+\delta e_j)=p^\delta(x,x+\delta e_i-\delta e_j)=\frac{a_{ij}^-(x)}{Q^\delta(x)},$$

这里$p^\delta(x,y)$表示状态 x 到状态 y 的转移概率，其中$x,y\in S_\delta$，当 y 取上述表达式中的值时，其对应的概率即由该式计算得到，当 y 取其他值时，转移概率为 0，这里$\Delta\tau^\delta(x)$表示系统在状态 x 的停时。其中$a(x)=\sigma(x)\sigma^T(x)=(a_{ij}(x))_{n\times n}$，$a_{ij}^+(x)=\max\{a_{ij}(x),0\}$，$a_{ij}^+(x)=\max\{-a_{ij}(x),0\}$，$b_i^+(x)=\max\{b_i(x),0\}$，$b_i^-(x)=\max\{-b_i(x),0\}$。由此我们就从状态空间得到了一个 Markov 链——*MC*，*MC* 的状态传播和原系统的状态传播是局部一致的，这已在文献[60]获得了证明。

注：这里本文需要指出的是随机微分方程离散的步长 h 和 *MC* 的状态停时$\Delta\tau^\delta$需满足一定的条件：对于每一个状态η，其停时$\Delta\tau^\delta$必须是 h 的倍数。这样当 Markov 链中的状态从η_1转移到η_2时，S-DPN 的可达状态也同时发生变迁，从而保证两者的一致性，而不会发生

S-DPN 的一个状态属于 *MC* 中不同的状态。此时 h 的必须是 $\Delta\tau^{\delta}$ 的“公约数”，即对任意的 $\Delta\tau^{\delta}$，存在一个正整数 z，使得 $\Delta\tau^{\delta}=zh$。具体做法为：选取一个足够大的正整数 l，使得所有 $10^{l}\times\Delta\tau^{\delta}$ 为整数，然后任意选取 $10^{l}\times\Delta\tau^{\delta}$ 的一个公约数 h_1，则 $h=\frac{h_1}{l_1}$，其中 l_1 为不小于 l 的一个正整数。

定义 3-7[63]：令 $X:\Omega\to R^n$ 的一个随机变量， X 的 L^2 范数，$\|X\|$，定义为

$$||X||=\{E[|X|^2]\}^{\frac{1}{2}} \quad = \quad [\int_{\Omega}|X(\omega)|^2dP(\omega)]^{\frac{1}{2}}=[\int_{R^n}|x|^2d\mu_X(x)]^{\frac{1}{2}}$$

其中 $\mu_X(x)$ 表示随机变量 X 的分布函数。对应的 L^2 空间定义为：

$$L^2(\Omega)=\{X:\Omega\to R^n|||X||<\infty\}$$

于是 L^2 空间中任意两个元素 X,Y 的距离可定义为：

$$\rho(X,Y)=||X-Y||=\sqrt{E[|X-Y|^2]}。$$

定义 3-8：称 S-DPN 的一个可达状态 $rs=(m_d,m_{\wp})$ 属于 *MC* 的某个状态 η，如果 $\rho(rs,\eta)<\frac{\delta}{2}$，其中 $\rho(rs,\eta)=\rho(m_{\wp},\eta)$。

定理 3-5：由状态空间分割得到的 Markov 链 *MC* 可以描述 S-DPN 的行为，即 *MC* 为由 S-DPN 的可达状态生成的 Markov 链。

证明：本文需要证明下述几点：1）S-DPN 的可达状态空间能够逼近原始系统的状态空间；2）S-DPN 的每一个可达状态属于 *MC* 的某个状态；3）S-DPN 的任何两个相继的可达状态要么在 *MC* 的同一个状态里，要么属于 *MC* 两个相邻的状态里。

1）S-DPN 的可达状态空间能够逼近原始系统的状态空间。

设 $RS(h)$ 表示以步长 h 获得的 S-DPN 的可达状态空间， *SP* 是切换随机系统的实际状态空间，$rs(\tau)=(m_d(\tau),m_{\mathscr{D}}(\tau))\in RS(h)$，$sp(\tau)=x(\tau)\in SP$。为此本文需要证明：$\forall\varepsilon>0,\exists h>0$，使得

$$\forall\tau_0,\ sp(\tau_0)\in SP,\ \exists rs(\tau_0)\in RS，\text{有}\rho(m_{\mathscr{D}}(\tau_0),\ sp(\tau_0))<\varepsilon。$$

令 $x(\tau)$ 为切换随机系统的解，$X(\tau)$ 为离散后的数值解，显然 $\forall\tau\in[\tau_k,\tau_{k+1})$ 有 $X(\tau)=X(\tau_k)\doteq X_k$。

$\forall\tau_0, sp(\tau_0)\in SP, \exists rs(\tau_0)\in RS$，存在$k$，使得$\tau_0\in[\tau_k,\tau_{k+1})$，$X(\tau_0)=X_k$。根据Euler-Maruyama法的强收敛性，有

$$E[sup_\tau|x(\tau)-X(\tau)|^2]\le C_0h \qquad 3\text{-}(20)$$

其中$h=\Delta\tau=\tau_{k+1}-\tau_k$，$C_0$是一个常数。

因此，有

$$\sqrt{E[|x(\tau_0)-X(\tau_0)|^2]}\le\sqrt{E[sup_\tau|x(\tau)-X(\tau)|^2]}\le\sqrt{C_0h}. \qquad 3\text{-}(21)$$

另一方面，由于$X(\tau_0)=X_k$，而X_k是切换随机系统的离散集，根据 S-DPN 的构造，存在一个可达状态$rs(\tau_0)=(m_d(\tau_0),m_{\mathscr{D}}(\tau_0))$，使得$m_{\mathscr{D}}(\tau_0)=X_k=X(\tau_0)$，于是根据 3-（21）有

$$\begin{aligned}\rho(sp(\tau_0),rs(\tau_0)) &= \sqrt{E[|sp(\tau_0)-m_{\mathscr{D}}(\tau_0)|^2]}\\ &= \sqrt{E[|x(\tau_0)-X(\tau_0)|]}\le\sqrt{C_0h}.\end{aligned} \qquad 3\text{-}(22)$$

因此，对$\forall\varepsilon>0$，取$0<h<\frac{\varepsilon^2}{2C_0}$，则有

$\rho(sp(\tau_0),rs(\tau_0))<\frac{\varepsilon}{2}<\varepsilon$。第一点得证。

注：对任意的$\eta=sp(\tau_0)(\in S_\delta)$，存在 S-DPN 的一个可达状态$rs(\tau_0)$，使得$rs(\tau_0)$ 属于η。这就是说 *MC* 的每一个状态至少包含 S-DPN 的可达图中的一个可达状态。

2）S-DPN 的每一个可达状态属于 *MC* 的某个状态。

这个性质是显然的。这是因为对任意的一个可达状态rs，$rs=(m_d,m_{\mathscr{D}})\in RS(h)$，$m_{\mathscr{D}}\in SP$，而对 *SP* 中的任何一个状态，它必然属于某个网格划分内。故存在η_0，使得$\rho(\eta_0,m_{\mathscr{D}})\le\delta$，即状态$rs$在状态$\eta_0$内。

3）S-DPN 的任何两个相继的可达状态要么在 *MC* 的同一个状态里，要么属于 *MC* 两个相邻的状态里。

为此本文还需要证明$\forall\tau_i=ih$，令$rs_1=m(\tau_i),rs_2=m(\tau_i+h)$，有

$$\rho(rs_1,rs_2)=\rho(m_{\mathscr{D}}(\tau_i+h),m_{\mathscr{D}}(\tau_i))\le\delta。 \qquad 3\text{-}(23)$$

事实上，根据 3-（18）随机微分库所的标识变化方程，有

$$m_{\mathscr{D}}(\tau_i+h)=m_{\mathscr{D}}(\tau_i)+\sum_{t_j\in T_D}W_{ij}\int_{\tau_i}^{\tau_i+h}v_j(s)ds+\sum_{t_j\in T_{\mathscr{D}}}W_{ij}\int_{\tau_i}^{\tau_i+h}u_j(s)dB(s)$$

于是有

$$
\begin{aligned}
& E[|m_{\mathscr{D}}(\tau_i+h)-m_{\mathscr{D}}(\tau_i)|^2] \\
= \;& E[|\sum_{t_j\in T_D} W_{ij}\int_{\tau_i}^{\tau_i+h} v_j(s)ds+\sum_{t_j\in T_{\mathscr{D}}} W_{ij}\int_{\tau_i}^{\tau_i+h} u_j(s)dB(s)|^2] \\
= \;& E[|\int_{\tau_i}^{\tau_i+h}\sum_{t_j\in T_D} W_{ij}v_j(s)ds+\int_{\tau_i}^{\tau_i+h}\sum_{t_j\in T_{\mathscr{D}}} W_{ij}u_j(s)dB(s)|^2] \\
= \;& E[|\int_{\tau_i}^{\tau_i+h} b(x(s))ds+\int_{\tau_i}^{\tau_i+h}\sigma(x(s))dB(s)|^2] \\
\le \;& 2E[|\int_{\tau_i}^{\tau_i+h} b(x(s))ds|^2+|\int_{\tau_i}^{\tau_i+h}\sigma(x(s))dB(s)|^2] \\
= \;& 2E[|\int_{\tau_i}^{\tau_i+h} b(x(s))ds|^2]+2E[|\int_{\tau_i}^{\tau_i+h}\sigma(x(s))dB(s)|^2] \\
= \;& 2E[|\int_{\tau_i}^{\tau_i+h} b(x(s))ds|^2]+2\int_{\tau_i}^{\tau_i+h} E(|\sigma(x(s))|^2)ds \\
\le \;& 2E[|\max_{s\in[\tau_i,\tau_i+h)} b(x(s))|^2|\int_{\tau_i}^{\tau_i+h} ds|^2]+2|\max_{s\in[\tau_i,\tau_i+h)}\sigma(x(s))|^2|\int_{\tau_i}^{\tau_i+h} ds| \\
= \;& 2h^2|\max_{s\in[\tau_i,\tau_i+h)} b(x(s))|^2+2h|\max_{s\in[\tau_i,\tau_i+h)}\sigma(x(s))|^2 \qquad \text{3-（24）}
\end{aligned}
$$

根据微分方程解的存在条件 3-（6），有：

$|b(x(\tau))|\le C(1+|x(\tau)|)$和$|\sigma(x(\tau))|\le C(1+|x(\tau)|)$。

因此

$|\max_{s\in[\tau_i,\tau_i+h)} b(x(s))|^2\le C^2(1+|x_{max}|)^2$以及$|\max_{s\in[\tau_i,\tau_i+h)}\sigma(x(s))|^2\le C^2(1+|x_{max}|)^2$。

当$h\le 1$时，取$h<min\{1,\frac{\delta^2}{4C^2(1+|x_{max}|)^2},\frac{\varepsilon^2}{C_0}\}$，则

$$3-(24)\le(2h^2+2h)C^2(1+|x_{max}|)^2\le 4hC^2(1+|x_{max}|)^2\le\delta^2;$$

当$h>1$时，取h,δ满足$\frac{\delta}{2C(1+|m_{max}|)}>1$和$h\le\frac{\delta}{2C(1+|m_{max}|)}$，则有

$$3-(24)\le(2h^2+2h)C^2(1+|x_{max}|)^2\le 4h^2C^2(1+|x_{max}|)^2\le\delta^2.$$

因此，有$\rho(rs_1,rs_2)\le\delta$，得证。 ■

由定理 3-5，假设对可达图进行划分，$\tilde{rs}_\eta=\left\{rs:\rho(rs,\eta)<\frac{\delta}{2}\right\}$，则这是一个等价类划分，等价关系 R 为 $rs_1Rrs_2\Leftrightarrow\rho(rs_1,\eta)<\frac{\delta}{2}\wedge\rho(rs_2,\eta)<\frac{\delta}{2}$，所有等价类构成的空间为 $\tilde{RS}(h)$。于是我们可以构造一个从 $\tilde{RS}(h)$ 到 S_δ 的映射： $f:\tilde{RS}(h)\to S_\delta$ 满足： $f\left(\tilde{rs}_\eta\right)=\eta$。由定理以及 h 和 $\Delta\tau^\delta$ 的关系，我们知道 f 是一个同构映射。事实上，根据定理 3.5 和等价类划分，f

是双射；其次 $\tilde{rs}_{\eta_1} \to \tilde{rs}_{\eta_2}$ 时，必存在两个相继的状态 $rs(\tau)$ 和 $rs(\tau+h)$，使得 $rs(\tau) \in \tilde{rs}_{\eta_1}$、$rs(\tau+h) \in \tilde{rs}_{\eta_2}$，由定理 3-5 证明中的 3）,还可以知道此时有 $\eta_1 \to \eta_2$；而 $f\left(\tilde{rs}_{\eta_1}\right) = \eta_1$，$f\left(\tilde{rs}_{\eta_2}\right) = \eta_2$，因此有 $f\left(\tilde{rs}_{\eta_1} \to \tilde{rs}_{\eta_2}\right) = f\left(\tilde{rs}_{\eta_1}\right) \to f\left(\tilde{rs}_{\eta_2}\right)$。故 *MC* 即可以作为与 S-DPN 同构的 Markov 链。将其作为 PRISM 的输入，我们就可以进行模型验证，例如计算某个状态的概率、验证某个状态的概率是否达到预期值等。

3.3 本章小结

本章主要对基于控制理论的自适应系统的建模和验证技术。这类自适应系统的一般框架图如图 3.8。它通过一个单独的控制组件来控制子系统的切换，而子系统间不存在通信过程。本文主要对两类系统——切换模糊系统和切换随机系统提出了形式化建模方法和验证技术，从而为底层实现提供顶层可验证的设计框架，提高软件的可靠性和鲁棒性。

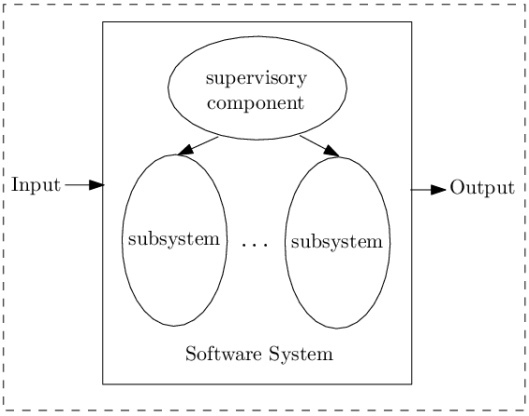


图 3.8 基于控制理论的自适应软件系统框架

4 第二类自适应软件系统的建模与验证

本章将研究第二类自适应系统，它不但能够根据系统本身来改变系统行为，而且还能通过感知环境变化而自动调整自身行为。这类自适应系统是目前学术界研究的重点，研究者提出了不同的结构模型，以期获得良好的自适应能力，但是很少有专门的一种形式化语言对这类自适应系统进行建模和验证。本章将通过一个工业制造系统的例子来提出一种形式化的自适应建模语言——自适应 Petri 网。它既能够描述系统的行为，同时还能够对环境建模和学习，并通过对环境的感知来实时改变系统的行为，具有良好的自适应性。此外，该模型是基于组件的形式化模型，具有良好的扩展性。

4.1 引例

首先来看一个制造企业的供应链模型[64]，如图 4.1。它包括客户下订单、产品生产过程和客户接收产品。

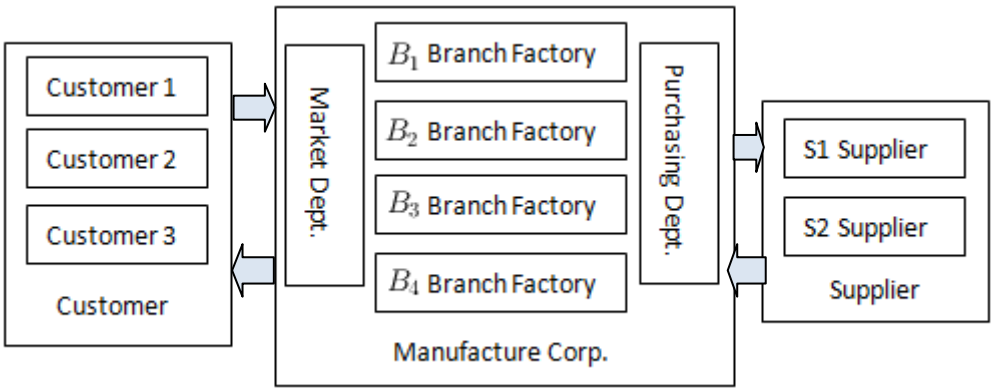


图 4.1 制造系统的生产过程

供应链模型可以看作是不同生产过程中不同参与者的有机组合，包括订单、产品设计、制造、交易等等。产品和交易的复杂性决定了制造企业对子公司和供货商的选择是一个动态的、实时的决策过程。跨企业的生产过程具有如下一些性质：多个离散生产过程的结合、管理规模的扩张、交易越来越快速灵活、对市场的反应越来越快速智能。制造系统的生产子公司和供货商的选择分析如图 4.2.

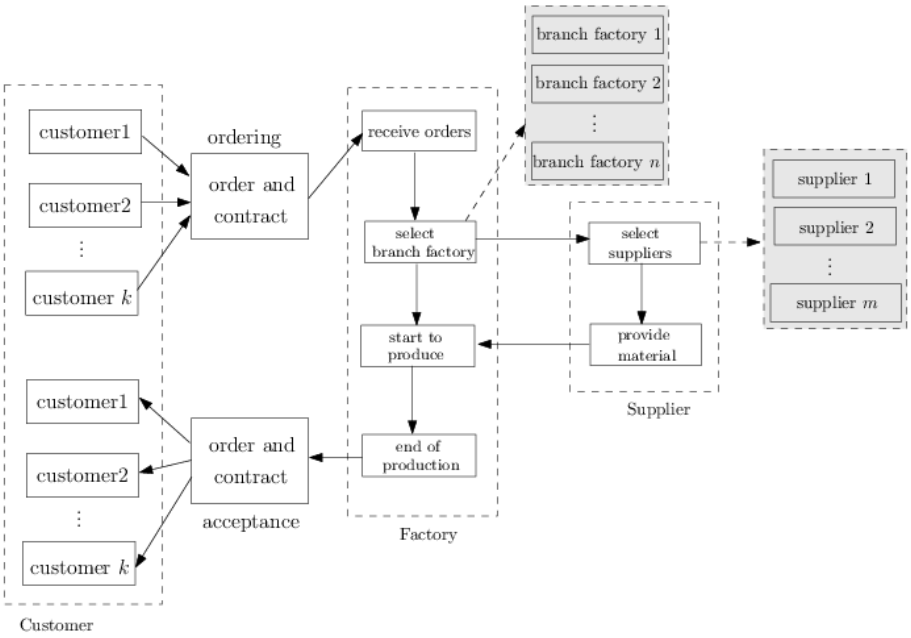


图 4.2 制造系统生产和选择的过程

为方便分析，本文假设市场部门负责收集汇总所有订单信息并下发生产计划，生成部门和采购部门合作完成订单，其中生产部门负责子公司的选择，采购部门负责供货商的选择。假设有 B_1、B_2、B_3、B_4 四个可供选择的子公司分散分布在四个不同的区域，S_1、S_2 两个供选择的原材料供货商。一般来说，影响子公司选择的因素有：生产能力、生产率、产品价格、运输条件、运输时间、季节因素、订单数量、其他因子；影响供货商选择的因素有：材料价格、订单数量、运输费用和其他因子。这 12 个因素构成了整个制造企业的外部市场环境，也即制造系统的运行环境。制造系统需要根据这 12 个因素选择一个合适的组合来完成生产过程。当系统环境变化时，这 12 个变量就发生了变化，于是系统需要重新安排生产过程。因此该系统具有如下性能需求：

R_1：（合作性）选择合适的生产子公司和供货商，使产品的生产成本最低；

R_2：（学习性）选择过程能够识别市场（环境）变化并对此做出快速反应；

R_3：（自适应性）系统能够根据市场变化，实时动态的确定的生产过程，即选择子公司和供货商。本节对制造企业进行建分析。

4.2 引例分析——需要对哪些方面进行建模

从这些需求中我们发现，该系统有三个过程：用户、制造企业、供货商。用户过程包含正常的下订单、等候订单、接收订单等行为，可以用正规的有限状态机或者一般离散 Petri 网来建模。而后两个过程则不然，它们需要识别环境的变化，从环境变化中学习应如何改变自身行为，最后两者合作完成生产过程，具体过程如图 4.3。因此我们的模型需要具有如下功能：（1）描述系统离散行为；（2）描述环境的连续变化；（3）具有学习功能；（4）

能够动态改变自身行为。

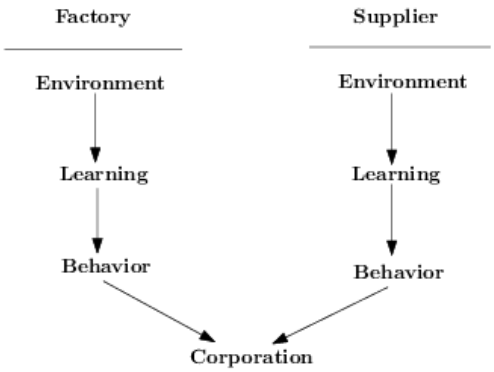


图 4.3 生产系统和供货商的需求分析

根据上述需求发现，我们建立的模型必须能够描述一个混合系统。目前已有很多可以对混合系统进行描述的语言，例如变迁系统（transition systems），过程代数（process algebras）等。然而，Petri 网是一个很好的选择。一方面，Petri 网被广泛的用作对离散事件系统进行建模、分析和合成的工具；尤其是考虑复杂模型和并发过程时，Petri 网相对自动机来说更具有优势。另一方面，一个连续系统可以用 Petri 网来近似，而连续变量系统的离散事件描述也可以用 Petri 网来实现。但是 Petri 网没有学习功能。因此，本文提出了一类新的 Petri 网来对制造企业系统进行建模。该网具有自适应性，其中包含了具有学习功能的自适应变迁（adaptive transition）和描述环境变化的连续变迁（continue transition）。

4.2.1 离散状态和连续状态建模

传统的 Petri 网可以用来描述离散状态。实际上，传统的 Petri 网是有限状态机的扩展，增加了描述并发行为的功能。对于连续状态，我们需增加连续变迁和连续库所来进行建模。因此，本文利用离散库所表示系统的离散状态，离散库所中的标记数（标识数）是正整数；连续库所表示环境，连续库所中的标识是实数。此外，本文用一个空心圆表示离散库所，用两个同心圆表示连续库所，离散变迁用直线表示，连续变迁用空心矩形表示，如图 4.4。

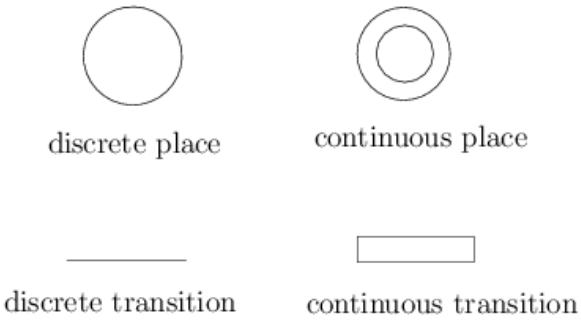


图 4.4 不同库所和变迁的表示

4.2.2 对环境建模

环境用一类专门的连续变迁进行建模。该连续变迁只有一个连续输入库所和一个连续

输出库所，环境的变化体现在从输入弧（从输入库所到变迁的弧）的权重的变化。例如，如果要对环境因子 X 进行建模，可设置输入弧的权重为 $\frac{1}{\alpha(\tau)}$，满足

$$\alpha(\tau)=\frac{x-\min}{\max-\min}$$

其中 x 表示 X 当前的观测值，min 和 max 表示环境的最小值和最大值。这样当连续变迁实施后，就可得到了归一化后的环境值。如图 4.5 所示，连续库所 p_1 的标识为 1，表示环境值的初始化；连续库所 p_2 的标识为经归一化处理后的当前环境变量 X 的值；权重 $w(p_1,t_e)=\frac{1}{\alpha(\tau)}$；$t_e$ 是连续变迁，它通过实施度(firing degree，计算公式为 $\frac{m_{p_1}(\tau)}{w(p_1,t_e)}$)的变化来反应环境的变化，这里连续变迁的实施度是一个实数，本文将在后面给出具体定义和说明。

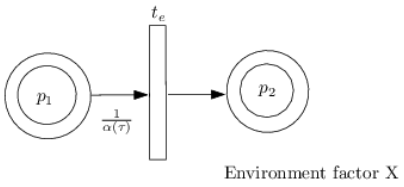


图 4.5 对环境变量 X 的建模

注：这里本文需要说明的一点是当 x=min 时，定义 $\frac{1}{\alpha(\tau)}=\omega$，其中 ω 是一个无穷大量，满足 $\forall r\in R, r<\omega$ 且 $\omega+\omega=\omega+r=\omega-r=\omega$。

4.2.3 对学习功能建模

我们知道，神经网络具有很好的学习功能，已经广泛的应用于智能系统中了。因此，在本文的模型中，我们将神经网络嵌入到 Petri 网。本文采用包含一个隐藏层的三层 BP 神经网络。当前有很多种方法来确定隐藏层的神经元数目，本文利用 Kolmogorov 理论[61]。利用 Petri 网对神经网络进行模拟的具体过程如下：

1）从神经网络转化得到的 Petri 网的所有库所都是连续库所，库所中的标识表示神经网络的信号。

2）为了表示神经网络的变化，我们需引入两类不同的变迁：$\alpha-$变迁和 $\beta-$变迁。其中 $\alpha-$变迁是一类连续变迁，用来对神经网络中的加法器进行建模，而 $\beta-$变迁中关联了神经网络中的激活函数，用来对神经网络中的激活函数的行为进行建模。在图形表示中，

本文用虚边矩形表示$\alpha-$变迁，用画有对角线的矩形表示$\beta-$变迁，如图 4.6。

3）由于神经网络中具有并发性，但是同一层中不同神经元的加法器计算和激活函数计算是同步的，因此在用 Petri 网进行建模时，我们需要考虑其同步性，因此我们还需引入控制库所。类似于同步通信，本文通过控制库所来描述神经网络中计算的同步性。此外，对于控制库所，它本质是一个离散库所，其标识为 0 或者 1，它只起控制作用而不参与神经网络的计算。本文采用虚线圆表示控制库所，如图 4.6

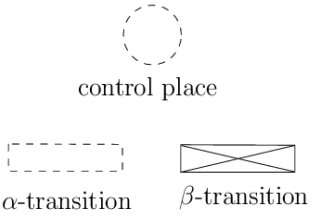
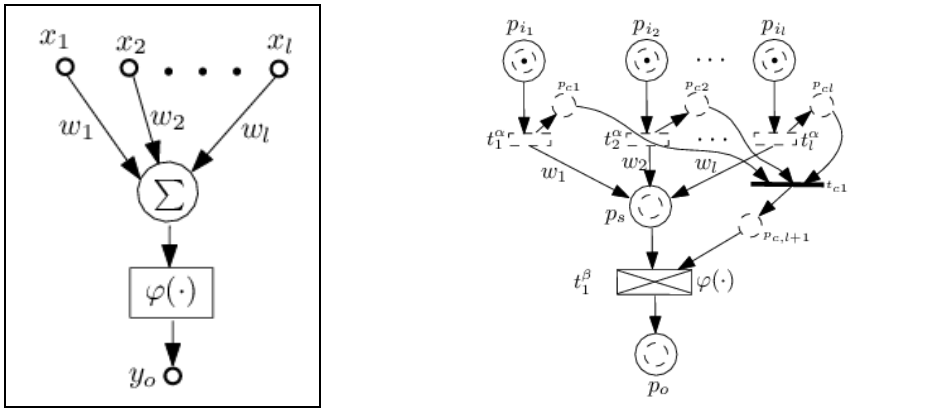


图 4.6 描述神经网络行为的变迁表示和库所表示

下面我们来具体说明如何用 Petri 网对神经网络进行建模。首先对单个的神经元进行建模。如图 4.7（a）是一个神经元模型，其中$x_1,x_2,\cdots,x_l$是神经元的输入变量，$w_1,w_2,\cdots,w_l$是输入权重，Σ是加法器，φ是激活函数；为方便起见，本文省略神经元的偏置。图 4.7（b）是对应的 Petri 网模型，其中$p_{i_1}-p_{i_l}$是 l 个连续库所，它们的标识表示神经元的输入变量，即$m\left(p_{i_j}\right)=x_j, j=1,2,\cdots,l$； p_s表示加法器处理后的输出，其中加法器行为由 l 个$\alpha-$变迁描述，即$t_1^\alpha-t_l^\alpha$；激活函数行为由$\beta-$变迁t_1^β描述，p_0表示神经网络的输出。此外，$p_1^c-p_l^c$为控制库所，用来描述同步行为。下面我们来证明该 Petri 网能获得神经元的输出：首先在变迁$\left\{t_1^\alpha,\cdots,t_l^\alpha\right\}$中随机选择一个变迁激活并实施（假设为$t_{j_1}^\alpha$），于是$p_s$获得标识$w_{j_1}m\left(p_{i_{j_1}}\right)$，同时$p_{cj_1}$获得一个 token；接着在$\left\{t_1^\alpha,\cdots,t_l^\alpha\right\}$中剩余的变迁中随机选择一个变迁实施后（假设为$t_{j_2}^\alpha$），$p_s$的标识变为$w_{j_1}m\left(p_{i_{j_1}}\right)+w_{j_2}m\left(p_{i_{j_2}}\right)$，$p_{cj_2}$获得一个 token；然后随机选择$t_{j_3}^\alpha$激活并实施，如此下去，一直到所有变迁被实施后，$p_s$的标识变为$m\left(p_s\right)=\sum_{j=1}^{l}w_j m\left(p_{i_j}\right)$，此记为神经网络中加法器的输出；此后$t_{c1}$被激活并实施，控制库所$p_{c,l+1}$同时获得了 token，故此后$t_1^\beta$被激活，当其实施后，$p_0$获得的标识为$\varphi\left(m\left(p_s\right)\right)$，于是有

$$m(p_0)=\varphi(m(p_s))=\varphi\left(w_1 m(p_{i_1})+w_2 m(p_{i_2})+\cdots+w_l m(p_{i_l})\right)$$
$$=\varphi(w_1x_1+w_2x_2+\cdots+w_lx_l)=y_0$$

因此，我们证明了 4.7（b）的 Petri 网确实能表示神经元。



（a）神经元　　（b）对神经元建模得到的 Petri 网

图 4.7 神经元的 Petri 网建模

神经网络是有许多神经元组成的，根据上述建模过程，可以很容易地对任何一个神经网络进行建模。例如图 4.8 所示，（a）是一个 2-5-1 的三层神经网络，输入层有两个神经元，中间隐藏层有 5 个神经元，输出层为一个神经元。(b)是其对应的 Petri 网模型，输入为 p_1, p_2；$p_{h1}-p_{h5}$ 为隐藏层 5 个神经元的输出； p_3 为输出层神经元的输出。

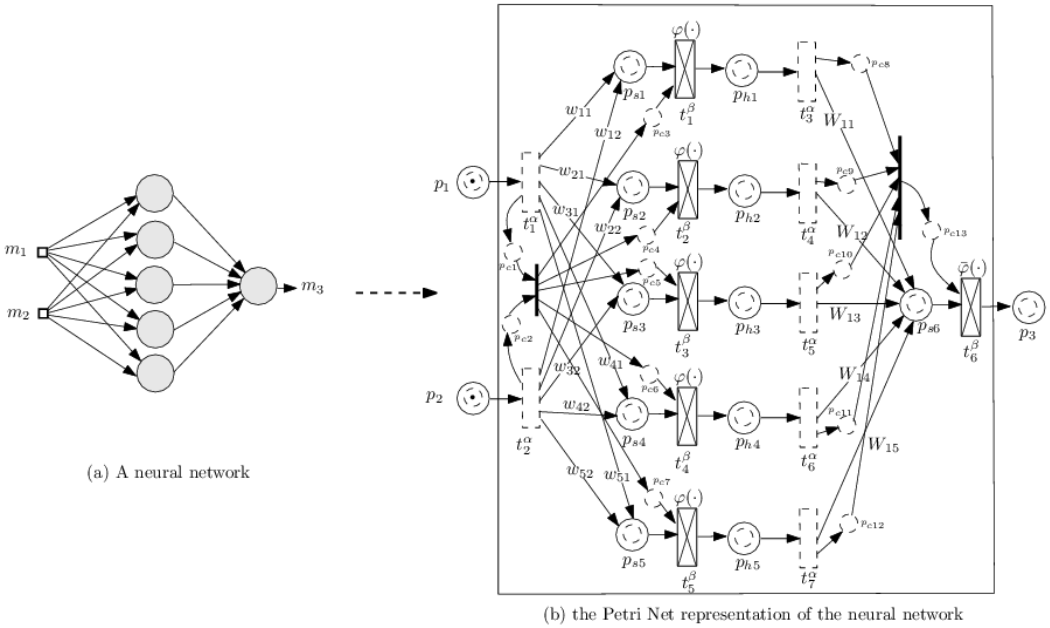


图 4.8 一个 2-5-1 结构的神经网络及其 Petri 网描述

因为在实际建模中，我们并不关心神经网络的具体训练过程，我们关注的是神经网络能够根据输入（环境）的变化，通过自动学习输出相应的学习结果，即我们需要的是神经网络的输出。因此有时为方便起见，我们并不是把整个神经网络的具体 Petri 网表示画出来，

而是通过一个变迁进行抽象，本文将该变迁命名为自适应变迁(Adaptive transition)，简记为A-变迁，并用一个实心矩形表示。例如，图 4.8（b）的 Petri 网可以简化为图 4.9 的形式，其中 t^A 为 A-变迁，它关联的函数 $f(\bullet)$ 为神经网络所描述的函数关系，在实际中我们并不知道具体表达式，而是通过神经网络进行计算的。

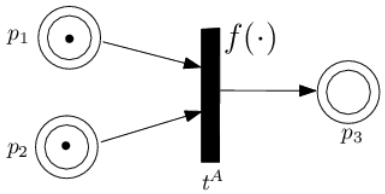
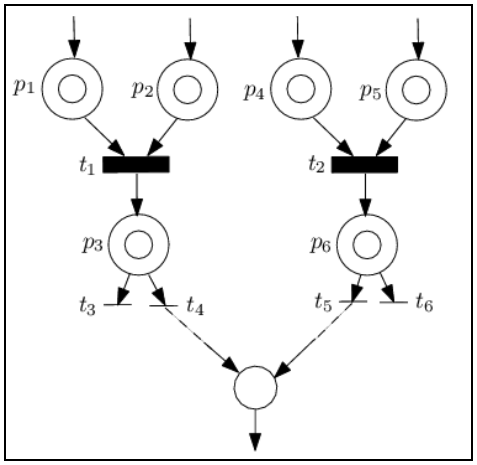


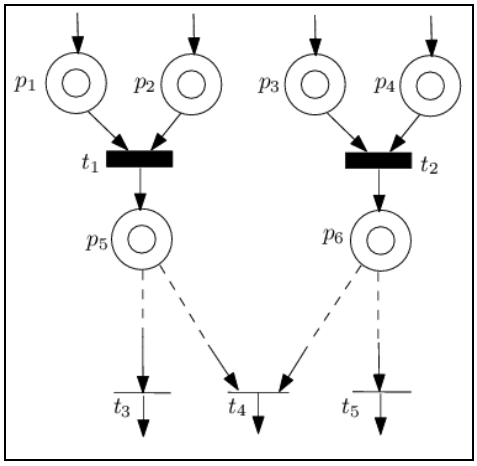
图 4.9 由神经网络建模得到的 Petri 网的简化形式

4.2.4 对合作性质建模

为方便讨论，本文仅考虑两个自适应变迁的合作，对多个自适应变迁可类似进行扩展。如果两个 A-变迁的输出最终可到达同一个库所，则称它们是独立的，如图 4.10（a）；如果两个 A-变迁的输出最终达到同一个变迁，则称它们是相互依赖的，如图 4.10（b）。



（a）两个独立的 A-变迁



（b）两个依赖的 A-变迁

图 4.10 多个变迁的两类关系

下面我们分别介绍对这类变迁所表示的神经网络的训练。首先，对两个独立的 A-变迁的训练是明显的，我们只需用各自的样本数据分别对这个神经网络进行单独的训练即可。而对于两个依赖的 A-变迁，我们需对这两个神经网络同时进行训练，具体过程如下：

1）样本数据的获取。对于新版本的软件系统，神经网络的输入样本数据可以根据历史获得；对于第一个版本的软件系统，神经网络的输入样本数据可以根据类似软件在类似环境中的运行数据获得，也可以从领域专家那里获得。而对于神经网络的输出，我们需要对两个组件进行综合考虑，分别计算不同组合所能得到的目标值，然后选择最优的组合作为神经网络的输出。例如，在制造企业中，根据 12 个环境值，分别计算不同组合（共有 8 种）下的产品成本，然后选取最小成本的组合作为综合输出。当然，因为神经网络的输出是实

数，而我们的不同选择的组合是离散整数，故我们需对选择进行连续化处理，样本输出设置为对应连续区间的中心点所表示的值。

2）算法设计。首先给出需要用到的一些符号的说明。$\dot{x}$ 表示 x（x 为某个库所或变迁）的输入集合，$x^{\cdot}$ 表示 x 的输出集合；t_1,t_2 表示两个相互依赖的 A-变迁；$k(=1,2)$ 表示两个 A-变迁所关联的神经网络，其中 $k=1$ 表示 t_1 所关联的神经网络，$k=2$ 表示 t_2 所关联的神经网络；q_k,l_k,s_k,S_k 分别表示神经网络 k 的输入神经元个数、隐藏层神经元个数、输出神经元个数和训练样本数据集；w_{hi}^k 表示神经网 k 中的输入神经元 i 到隐藏层神经元 h 的权重；W_{oh}^k 表示神经网 k 中的隐藏层神经元 h 到输出神经元 o 的权重；φ_1^k,φ_2^k 分别表示输入层到隐藏层的激活函数和隐藏层到输出层的激活函数；$x^{[k]}=\left(x_1^{[k]},x_2^{[k]},\cdots,x_{q_k}^{[k]}\right)^T$ 表示神经 k 的输入样本数据，$y^{[k]}=\left(y_1^{[k]},y_2^{[k]},\cdots,y_{s_k}^{[k]}\right)^T$ 表示神经 k 的输出样本数据，$\tilde{y}^{[k]}=\left(\tilde{y}_1^{[k]},\tilde{y}_2^{[k]},\cdots,\tilde{y}_{s_k}^{[k]}\right)^T$ 表示神经 k 的实际输出。具体训练算法分如下几步：

Step 1：给定精度 ε，学习率 η 和最大迭代次数 N；

Step 2：初始化各神经网络的权重。本文采用 Nguyen-Widrow 算法进行初始化[67]。

Step 3：分别从 S_1 和 S_2 中选取一组样本数据 $\left(x^{[1]},y^{[1]}\right)$ 和 $\left(x^{[2]},y^{[2]}\right)$；

Step 4：计算神经网络的输出，以及局部误差；

$$zz_{h_k}^{[k]}=\sum_{i=1}^{q_k}w_{h_ki}^{[k]}x_i^k,\quad z_{h_k}^{[k]}=\varphi_1^{[k]}(zz_{h_k}^{[k]}) \tag{4-(1)}$$

$$\tilde{yy}_{o_k}^{[k]}=\sum_{h=1}^{l_k}W_{o_kh}^{[k]}z_h^{[k]},\quad \tilde{y}_{o_k}^{[k]}=\varphi_2^{[k]}(\tilde{yy}_{o_k}^{[k]}) \tag{4-(2)}$$

$$e_k=\sum_{o=1}^{s_k}(y_o^{[k]}-\tilde{y}_o^{[k]})^2,\quad e=\tfrac{1}{2}(e_1+e_2) \tag{4-(3)}$$

Step 5：计算 e 关于 $W_{o_kh_k}^{[k]}$ $(o_k=1,\cdots,s_k;\ h_k=1,\cdots,l_k)$；

$$\frac{\partial e}{\partial W_{o_kh_k}^{[k]}}=\frac{\partial e}{\partial\tilde{y}_{o_k}^{[k]}}\frac{d\tilde{y}_{o_k}^{[k]}}{d\tilde{yy}_{o_k}^{[k]}}\frac{\partial\tilde{yy}_{o_k}^{[k]}}{\partial W_{o_kh_k}^{[k]}}=-(y_{o_k}^{[k]}-\tilde{y}_{o_k}^{[k]})\varphi_2^{[k]\prime}(\tilde{yy}_{o_k}^{[k]})z_{h_k}^{[k]}\triangleq-\delta_{o_k}^{[k]}z_{h_k}^{[k]} \tag{4-(4)}$$

其中 $\varphi_2^{[k]\prime}(\cdot)$ 表示函数 $\varphi_2^{[k]}(\cdot)$ 的导数；

Step 6：计算 e 关于 $w_{h_ki_k}^{[k]}$ $i_k=1,\cdots,q_k;\ h_k=1,\cdots,l_k)$；

$$\frac{\partial e}{\partial w_{h_k i_k}^{[k]}} = \sum_{o=1}^{s_k}(\frac{\partial e}{\partial \tilde{y}_o^{[k]}} \cdot \frac{d\tilde{y}_o^{[k]}}{d\tilde{y}y_o^{[k]}} \cdot \frac{\partial \tilde{y}y_o^{[k]}}{\partial z_{h_k}^{[k]}}) \cdot \frac{dz_{h_k}^{[k]}}{dzz_{h_k}^{[k]}} \cdot \frac{\partial zz_{h_k}^{[k]}}{\partial w_{h_k i_k}^{[k]}}$$

$$= -\sum_{o=1}^{s_k}[(y_0^{[k]} - \tilde{y}_o^{[k]}) \cdot \varphi_2^{[k]'}(\tilde{y}y_o^{[k]}) \cdot w_{oh_k}^{[k]}]\varphi_1^{[k]'}(zz_{h_k}^{[k]})x_{i_k}^{[k]}$$

$$= -\sum_{o=1}^{s_k}(\delta_o^{[k]} \cdot w_{oh_k}^{[k]}) \cdot \varphi_1^{[k]'}(zz_{h_k}^{[k]}) \cdot x_{i_k}^{[k]} \triangleq -\delta_{h_k}^{[k]} \cdot x_{i_k}^{[k]} \quad \text{4-(5)}$$

Step 7：更新神经网络的所有权重；

$$\Delta w_{o_k h_k}^{[k]} = -\eta \frac{\partial e}{\partial w_{o_k h_k}^{[k]}} = \eta \delta_{o_k}^{[k]} z_{h_k}^{[k]}, \quad (w_{o_k h_k}^{[k]})^{n+1} = (w_{o_k h_k}^{[k]})^n + \Delta w_{o_k h_k}^{[k]} \quad \text{4-（6）}$$

$$\Delta w_{h_k i_k}^{[k]} = -\eta \frac{\partial e}{\partial w_{h_k i_k}^{[k]}} = \eta \delta_{h_k}^{[k]} x_{i_k}^{[k]}, \quad (w_{h_k i_k}^{[k]})^{n+1} = (w_{h_k i_k}^{[k]})^n + \Delta w_{h_k i_k}^{[k]} \quad \text{4-（7）}$$

Step 8：根据样本数据S_1和S_2，计算全局误差。

$$E = \frac{1}{2|\mathcal{S}_1| + 2|\mathcal{S}_2|}[\sum_{(x^{[1]},y^{[1]})\in\mathcal{S}_1} ||y^{[1]} - \tilde{y}^{[1]}|| + \sum_{(x^{[2]},y^{[2]})\in\mathcal{S}_2} ||y^{[2]} - \tilde{y}^{[2]}||]$$

$$= \frac{1}{2|\mathcal{S}_1| + 2|\mathcal{S}_2|}[\sum_{(x^{[1]},y^{[1]})\in\mathcal{S}_1}\sum_{o=1}^{s_1}(y_o^{[1]} - \tilde{y}_o^{[1]})^2 + \sum_{(x^{[2]},y^{[2]})\in\mathcal{S}_2}\sum_{o=1}^{s_2}(y_o^{[2]} - \tilde{y}_o^{[2]})^2] \quad \text{4-（8）}$$

Step 9：迭代次数加 1，并判断：如果$n > N$或者$E < \varepsilon$，停止；否则，转 Step 3.

注 1：文本考虑的是线性合作，故神经网络的局部误差和全局误差为单个神经网络误差的算术平均，即 4-（3）和 4-（8）；对于其他类型合成，我们需根据实际重新定义误差算法。

注 2：考虑线性合作，对于多个神经网络，我们可类似扩展，其中需要修改 4-（3）和 4-（8），并对每个权重求梯度和更新。假设有 Z 个神经网络，则有

$$e = \frac{1}{Z}\sum_{i=1}^{Z} e_i \quad \text{4-（3’）}$$

$$E = \frac{1}{2\sum_{j=1}^{Z}|\mathcal{S}_j|}\sum_{i=1}^{Z}\sum_{(x^i,y^i)\in\mathcal{S}_i}\sum_{o=1}^{s_i}(y_o^i - \tilde{y}_o^i)^2 \quad \text{4-（8’）}$$

4.3 自适应 Petri 网的定义和语义

本节给出自适应 Petri 网的定义，它既能对环境进行建模，同时还能够描述系统的行为变化。

4.3.1 自适应 Petri 网的定义

定义 4-1：**学习网**是一个多元组$PN^L = \langle P_c \cup \{p_s^l, p_e^l\}, \{t_d, t_a\} \cup T_c, F, w\rangle$满足

1）P_c是一个连续库所集合，它可以分为两类子集：P_c^d和P_c^a，其中$|P_c^d| = |P_c^a|$，且$P_c^d = t_d^\bullet$，$P_c^a = {}^\bullet t_a$。

2）p_s^l是一个离散库所，${}^\bullet p_s^l = \varnothing$，$p_s^{l\bullet} = t_d$；$p_e^l$是一个连续库所，${}^\bullet p_e^l = t_a$，$p_e^{l\bullet} = \varnothing$。

3）F 是有向弧的集合，$F \subset (P \times T) \cup (T \times P)$。

4）w 是有向弧的权重集合，其中 $\forall p \in P_c^d, w(p,t) \in R^+ \cup \{\omega\}$，其他弧的权重为 1.

图 4.11 所示即为一个学习网。

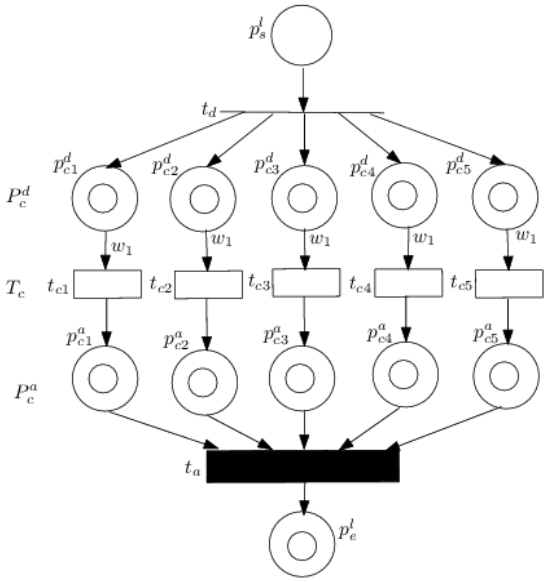


图 4.11 学习网的例子

定义 4-2：分支网是一个三元组 $PN^B = \langle P,T,F \rangle$ 满足

1）P 是有限库所集，满足对任意的 $p \in P$，有 $|{}^\bullet p| \geq 1, |p^\bullet| \geq 1$；

2）T 是有限变迁集，满足对任意的 $t \in T$，有 $|{}^\bullet t| = 1, |t^\bullet| = 1$；

3）F 是有向弧集合，$F \subset (P \times T) \cup (T \times P)$。

定义 4-3： 学习分子网是分支网 $PN^B = \langle P,T,F \rangle$ 和学习网 $PN^L = \langle P_c \cup \{p_s^l, p_e^l\}, \{t_d, t_a\} \cup T_c, F, w, \rangle$ 的合成，其中存在两个变迁 $t_1, t_2 \in T$，使得 ${}^\bullet p_s^l = t_1$，$p_e^{l\bullet} = t_2$。

定义 4-4：闭合过程网是一个多元组 $PN^C = \langle P \cup \{p_s, p_e\}, T \cup \{t_e\}, F \rangle$，满足：

1）P 是有限内部库所，且对任意的库所 $p \in P$，有 $|{}^\bullet p| \geq 1$、$|p^\bullet| \geq 1$；

2）T 是有限内部变迁，且对任意的变迁 $t \in T$，有 $|{}^\bullet t| = 1$、$|t^\bullet| = 1$；

3）p_s 为起始库所，满足 $|{}^\bullet p_s| = 1$，$|p_s^\bullet| \geq 1$；

4）p_e 是终止库所，满足 $|{}^\bullet p_e| \geq 1$，$|p_e^\bullet| = 1$；

5）${}^\bullet t_e = p_e$，$t_e^\bullet = p_s$

6）F 是有向弧集合，$F \subset (P \times T) \cup (T \times P)$。

定义 4-5：闭合学习过程网PN^{CL}是闭合过程网$PN^{C} = \langle P \cup \{p_s, p_e\}, T \cup \{t_e\}, F\rangle$和学习网 $PN^{L} = \left\langle P_c \cup \left\{p_s^l, p_e^l\right\}, \{t_d, t_a\} \cup T_c, F, w\right\rangle$ 的合成，其中存在两个变迁$t_1, t_2 \in T$，满足 ${}^{\bullet}p_s^l = t_1$，$p_e^{l\bullet} = t_2$。

根据定义，我们可以知道，闭合（学习）过程网由起始库所p_s引导产生一个或者多个（学习）分支网，并最后在库所p_e会合。如图 4.12 所示，（a）是闭合过程网的一般框架，（b）是一个由三个分支网——其中最左边的是一个学习分支网，剩余两个是分支网——构成的闭合学习过程网实例。

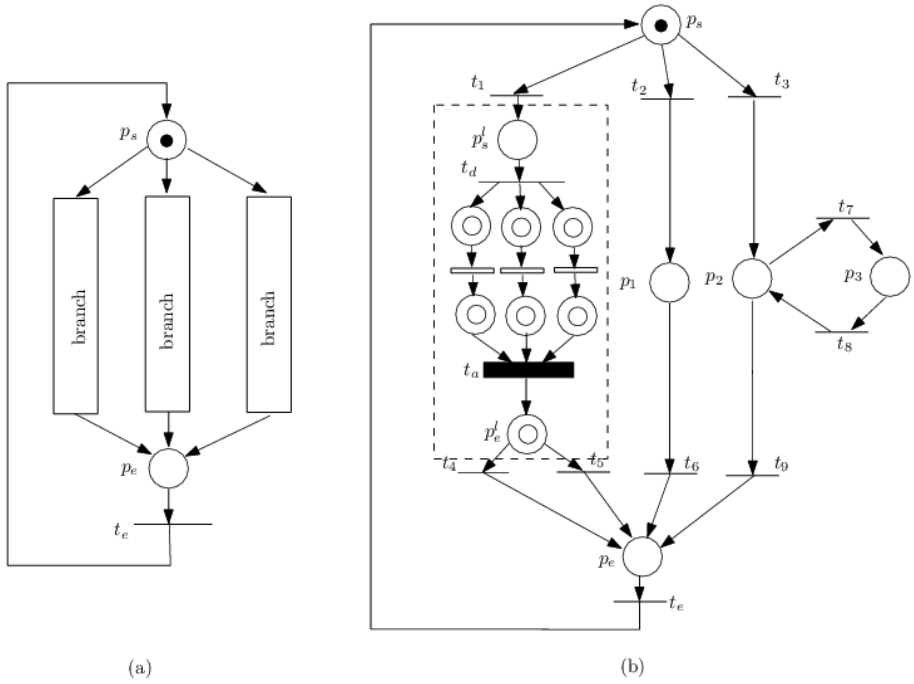


图 4.12 (*a*) 闭合（学习）过程网；(*b*) 一个闭合学习过程网的实例

接下来描述不同过程之间的通信行为。不同过程通过交会通信（rendezvous communication）进行通信，如图 4.13。交会通信过程依赖于互斥的库所p_{mcp}[68]。p_{mcp}确保了在任何时刻，有且只有一个闭合过程网能发送一个标识（token）给某个过程网，直到发送标识的过程网接受到来自p_{ack}的标识后，才可能有其他过程网给这个接受标识的过程网发送信息。p_{send}和p_{ack}称为**缓冲库所**（buffer place），连接变迁称为**通信变迁**（communication transition）。

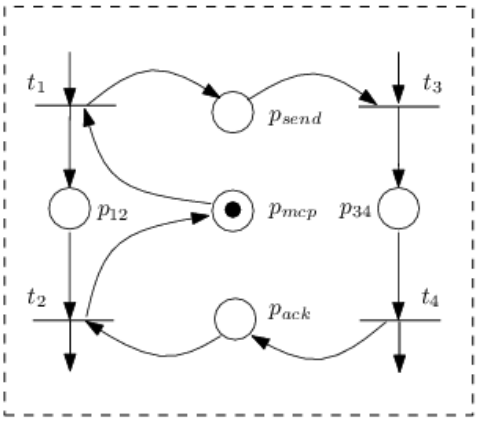


图 4.13 两个过程的会合通信机制

定义 4-6：交会通信机制是一个多元组$CM^{②} = \langle t_1, t_2, t_3, t_4, p_{12}, p_{34}, p_{send}, p_{ack}, p_{mcp}\rangle$，其中

1）库所p_{12}, p_{34}, p_{send}和p_{ack}满足$|{}^\bullet p_{12}| = |p_{12}^\bullet| = 1$，$|{}^\bullet p_{34}| = |p_{34}^\bullet| = 1$，$|{}^\bullet p_{send}| = |p_{send}^\bullet| = 1$，以及$|{}^\bullet p_{ack}| = |p_{ack}^\bullet| = 1$；$p_{mcp}$满足$|{}^\bullet p_{mcp}| = |p_{mcp}^\bullet|$；

2）变迁t_1满足$|{}^\bullet t_1| = |t_1^\bullet| = 2$，${}^\bullet p_{12} = t_1$，${}^\bullet p_{send} = t_1$，以及$t_1 \in p_{mcp}^\bullet$；

3）变迁t_2满足$|{}^\bullet t_2| = |t_2^\bullet| = 2$，$p_{12}^\bullet = t_2$，$p_{ack}^\bullet = t_2$，以及$t_2 \in {}^\bullet p_{mcp}$

4）变迁t_3满足$|{}^\bullet t_3| = 2$，$|t_3^\bullet| = 1$，${}^\bullet p_{34} = t_3$，以及$p_{send}^\bullet = t_3$；

5）变迁t_4满足$|{}^\bullet t_4| = 1$，$|t_4^\bullet| = 2$，$p_{34}^\bullet = t_4$，以及${}^\bullet p_{ack} = t_4$

我们称p_{12}, p_{34}为中介状态，p_{send}, p_{ack}称为缓冲库所，p_{mcp}称为通信控制库所，t_1, t_4称为输出通信变迁，t_2, t_3称为输入通信变迁。

定义 4-7：令PN_1^C, PN_2^C连个闭合（学习）过程网。如果存在一个交会通信机制$CM^{②} = \langle t_1, t_2, t_3, t_4, p_{12}, p_{34}, p_{send}, p_{ack}, p_{mcp}\rangle$使得$PN_1^C$和$CM^{②}$有公共部分$\{t_1, t_2, p_{12}\}$，而$PN_2^C$和$CM^{②}$有公共部分$\{t_3, t_4, p_{34}\}$，则称$PN_1^C$通过$CM^{②}$与$PN_2^C$通信，简称通信。

定义 4-8：令 *A,B,C* 为三个闭合（学习）过程网，如果 *A,B* 同时与 *C* 通信，则它们的交会通信机制有相同的$\{p_{send}, p_{ack}, p_{mcp}\}$，即 *A,B* 都可以发送消息给 *C*，但是不能同时发送。此时我们也称有两对过程网（$(A,C),(B,C)$）共用一个通信机制。

例如图 4.14 描述了两个闭合过程网交替的和同一个闭合过程网进行通信。

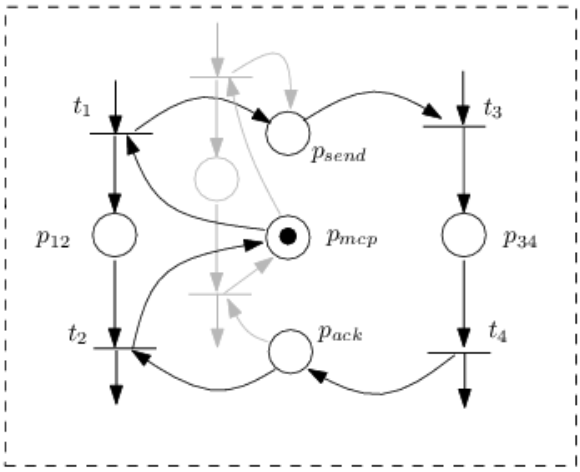


图 4.14 两个闭合过程网与同一个闭合过程网进行通信

上述过程还可以扩展到一个闭合过程网给不同闭合过程发生通信请求，同时等待至少一个的回复，如图 4.15。这里不再详细叙述。

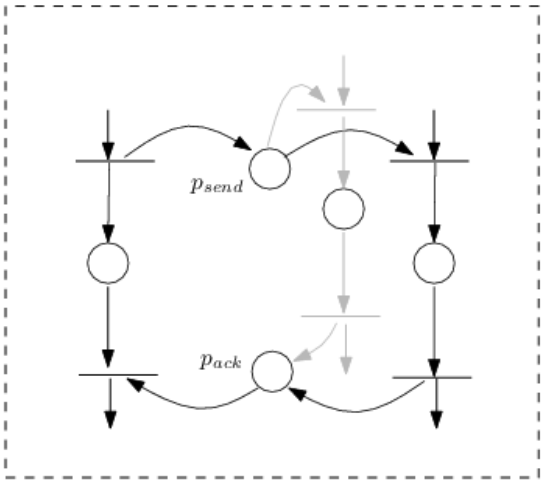


图 4.15 一个闭合过程网给两个其他闭合过程网发送通信请求

下面将给出自适应 Petri 网的定义。

定义 4-9：自适应 Petri 网（APN）是一类扩展的 Petri 网，它由闭合过程网集合$\{PN_o^C, o=1,2,\ldots,g\}$、闭合学习过程网集合$\{PN_i^{CL}, i=1,2,\ldots,k\}$以及交会通信机制集合$\{CM_j^Q, j=1,2,\ldots,l\}$组成，同时满足

1）每个闭合（学习）过程网至少通过一个交会通信机制和其他闭合（学习）过程网进行通信；

2）每一个交会通信机制CM_j^Q仅被一对闭合过程网使用。

我们将自适应 Petri 网记为PN^N。

在自适应 Petri 网中，所有起始库所 p_s 的集合记为P^S，所有缓冲库所p_{send}和p_{ack}记为P^B，

所有通信控制库所p_{mcp}记为P^C。

定义 4-10：标识自适应 Petri 网是一个多元组$\langle PN^N, M_0\rangle$，满足

1）$PN^N = \langle P, T, F\rangle$为自适应 Petri 网；

2）$M_0 : P \to \{0,1\}$为初始标识：$\forall p \in P^S, M_0(p) = 1$，$\forall p \in P^C, M_0(p) = 1$，其余均为 0。

根据自适应 Petri 网的定义，我们可以知道该模型可以对具有多个过程且过程间通过交会通信机制进行通信的系统进行建模。

此外，我们根据每个库所的标识的类型，可以将自适应 Petri 网中的库所分为离散库所集合P_D，其标识为正整数；连续库所P_C，其标识为实数；变迁可以分为离散变迁T_D，连续变迁T_C和自适应变迁T_A。P_C，T_C和T_A只出现在学习网中。若将 A-变迁展开，这自适应 Petri 网还有$\alpha-$变迁 集合T_α，$\beta-$变迁 集合T_β。下面，我们根据这种分类给出不同变迁的实施条件和库所标识的变化方程。

4.3.2 自适应 Petri 网的语义

本节定义自适应 Petri 网的语义。首先，我们根据自适应 Petri 网中变迁的实施条件。

1）离散变迁t可被标识M_i激活的条件为：

$\forall p \in {}^\bullet t \cap (P_D \cup P_C), M_i(p) \geq W(p,t)$

2）连续变迁t可被标识M_i激活的条件为：

$\forall p \in {}^\bullet t \cap (P_D \cup P_C), M_i(p) > 0$

3）A-变迁t可被标识M_i激活的条件为：

$\forall p \in {}^\bullet t \cap P_C, M_i(p) > 0$

当我们考虑神经网络的 Petri 网展开时，还有如下变迁激活条件：

4）$\alpha-$变迁t可被标识M_i激活的条件为：

$\forall p \in {}^\bullet t \cap P_C, M_i(p) > 0$且$\forall p \in {}^\bullet t \cap P_{Ctl}, M_i(p) = 1$

5）$\beta-$变迁t可被标识M_i激活的条件为：

$\forall p \in {}^\bullet t,\ M_i(p) > 0$且$\forall p \in {}^\bullet t \cap P_{Ctl}, M_i(p) = 1$

其中P_{Ctl}为 A-变迁展开中的控制库所集合。

在自适应 Petri 网中，实施度描述了变迁实施后的最大标识传输量。不通过变迁有不同的实施度，具体为：

1）离散变迁的实施度。对任意的$t \in T_D$，其在标识 M 下的实施度，记为$q(t,M)$或简单记为q，是一个正整数，满足

$$q \le \min_{p_i:p_i \in {}^\bullet t \cap (P_D \cup P_C)} \frac{M(p_i)}{W(p_i,t)} < q+1 \qquad 4\text{-}(9)$$

2）连续变迁的实施度。对任意的$t \in T_C$，其在标识 M 下的实施度，记为$\delta(t,M)$或简单记为δ，是一个正实数：

$$\delta = \min_{p_i:p_i \in {}^\bullet t \cap (P_D \cup P_C)} \frac{M(p_i)}{W(p_i,t)} \qquad 4\text{-}(10)$$

2）α–变迁 的实施度。对任意的$t \in T_\alpha$，其在标识 M 下的实施度，记为δ_α，是一个正实数：

$$\delta_\alpha = \min_{p_i:p_i \in {}^\bullet t \cap P_C} \frac{M(p_i)}{W(p_i,t)} \qquad 4\text{-}(11)$$

因此，根据实施度，本文可定义自适应 Petri 网的标识变化。假设变迁 t 可被标识 M_i 激活并实施，则实施后的标识 M_{i+1} 的变化为：

1）如果$t \in T_D$，实施度为q，则有

$$M_{i+1}(p) = \begin{cases} M_i(p) + qW(t,p), & p \in t^\bullet \cap (P_D \cup P_C) \\ M_i(p) - qW(p,t), & p \in {}^\bullet t \cap (P_D \cup P_C) \\ M_i(p), & \text{otherwise} \end{cases} \qquad 4\text{-}(12)$$

2）如果$t \in T_C$，实施度为δ，则有

$$M_{i+1}(p) = \begin{cases} M_i(p) + \delta(t,M_i)W(t,p), & p \in t^\bullet \cap (P_D \cup P_C) \\ M_i(p) - \delta(t,M_i)W(p,t), & p \in {}^\bullet t \cap (P_D \cup P_C) \\ M_i(p), & \text{otherwise} \end{cases} \qquad 4\text{-}(13)$$

3）如果$t \in T_\Lambda$，则有

$$M_{i+1}(p) = \begin{cases} 0, & p \in {}^\bullet t \\ f_p({}^\bullet t), & p \in t^\bullet \end{cases} \qquad 4\text{-}(14)$$

其中f_p为 t 所嵌入的神经网络所描述的函数。

4）如果$t \in T_\alpha$，则有

$$M_{i+1}(p) = \begin{cases} M_i(p) + \delta_\alpha W(t,p), & p \in t^\bullet \cap P_C \\ 1, & p \in t^\bullet \cap P_{Ctl} \\ M_i(p) - \delta_\alpha W(p,t), & p \in {}^\bullet t \cap P_C \\ 0, & p \in {}^\bullet t \cap P_{Ctl} \\ M_i(p), & \text{otherwise} \end{cases} \qquad 4\text{-}(15)$$

5）如果$t \in T_\beta$，则有

$$M_{i+1}(p) = \begin{cases} 0, & p \in {}^\bullet t \\ \varphi(M_i(p)), & p \in t^\bullet \cap P_C \\ 1, & p \in t^\bullet \cap P_{Ctl} \\ M_i(p), & \text{otherwise} \end{cases} \qquad 4\text{-}(16)$$

根据方程 4-（12），4-（13），4-（14）（有时此方程将由 4-（15），4-（16）替换），我们可以计算自适应 Petri 网的可达图，具体算法如下：

表 4.1 自适应 Petri 网的可达图计算

输入：自适应 Petri 网(P, T, F, w, M_0);

输出:可达状态 RS, 可达图 RG;

初始化: RS = RG:= {M_0}, new_states = {M_0}, i:=0;

```
1. begin
2.   if(new_states !=NULL)
3.       T_enabled = the transitions enabled by new_states;
4.       old_states=new_states,   new_states=NULL;
5.       while(T_enabled != NULL)
6.          selecting a transition t from T_enabled;
7.          M' = the marking after firing t;
8.          if(M' is not belong to old_states)
9.                RS := RS+{M'} and RG:=RG+{M_i-(t)->M'};
10.                new_states=new_states+{M'};
11.          end
12.          T_enabled := T_enabled-{t};
13.       end
14.    end
15. end
```

4.3.3 制造系统的自适应 petri 网建模

根据引例的分析和自适应 Petri 网的定义，我们可以很容易地给出制造系统的自适应 Petri 网建模，如图 4.16。在该模型中，$P^S = \{p_1, p_6, p_{22}\}$，$P^B = \{p_4, p_5, p_{19}, p_{20}, p_{21}\}$，$P^C = \emptyset$。从另一角度来说，$P_C = \{p_i \mid i = 26, \cdots, 51\}$，其余均为离散库所，$T_C = \{t_i^C \mid i = 1, \cdots, 12\}$，$T_A = \{t_1^A, t_2^A\}$。初始标识为$M_0(p_1) = M_0(p_6) = M_0(p_{22}) = 1$，其余为 0；连续库所$p_{34} - p_{41}$和$p_{47} - p_{50}$表示实际环境值；A-变迁$t_1^A$和$t_2^A$分别嵌入了一个三层 BP 神经网络，记为$NN_1$和$NN_2$，其中间层的神经元个数分别为 17 和 9，它们可以根据 4.2.2 节中提出的方法展开为 Petri 网。

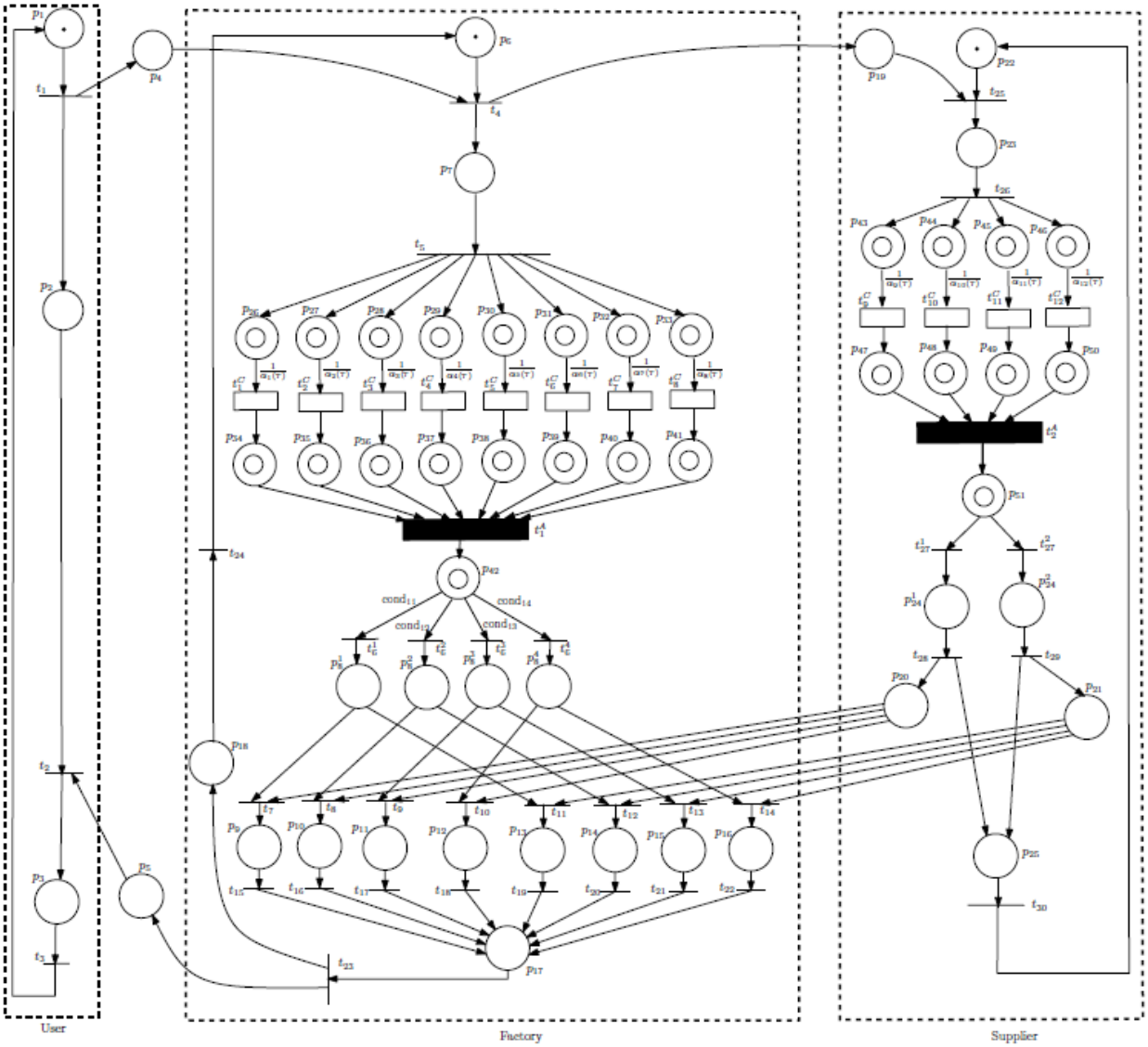


图 4.16 制造系统的自适应 Petri 网模型

4.4 自适应 Petri 网的自适应性分析

这一节将从理论上对 APN 的行为和自适应性进行分析和证明。

4.4.1 自适应 Petri 网的可达图的建立和分析

由第四章可以得到其可达图。这一小节我们将对可达图的结构进行分析。我们将可达图中由相同变迁序列实施获得的节点重叠在一起，则可以得到一个新的图---演化图。此时演化图中的每个节点有无限个标识：在不同标识中，离散库所的标识值是一样的，不同的在于连续库所的标识。例如图 4.16 描述的制造企业的自适应 Petri 网的 *EG* 如图 4.17.

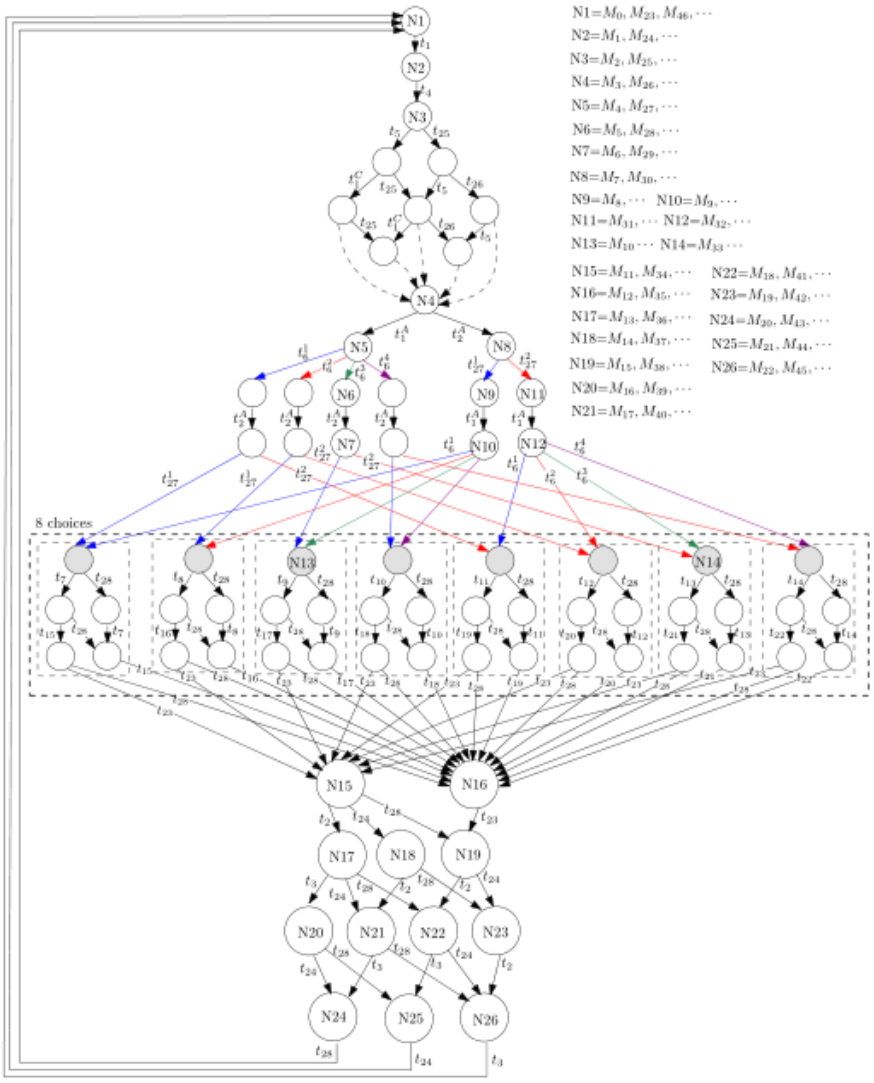


图 4.17 制造系统的演化图

和传统的 Petri 网（将自适应 Petri 网中的连续变迁和 A-变迁换成离散变迁，连续库所替换为离散库所）的可达图比较，我们有如下结论：

定理 4-1：假设自适应 Petri 网的演化图为 EG。将自适应 Petri 网的连续变迁和 A-变迁替换成离散变迁，相关联的弧权重变为 1，则得到一个传统的 Petri 网，它的可达图为 RG。则在忽略节点中的标识时，EG 和 RG 具有相同的结构。

证明：首先考虑传统 Petri 网计算可达图的算法[62]。本文将简单地复述基于广度优先的生成算法：Step 1，确定所有激活变迁；Step 2，依次实施可激活变迁，产生后继标识集合；Step 3，判断，对每一个后继标识，如果它是已有的标识或者死锁标识，则将其从后继集合中移除；Step 4，对后继标识集合重复前面三个步骤，直到集合为空。由于传统 Petri 网的有界性，因此该过程是有限的。

将上述过程与本文的算法相比较，第 3 行就是上述的 Step1；5-13 行是 Step 2 和 Step 3

中的过程，其中 8-11 行是 Step3 的过程，而 Step 2 的过程由 while 循环实现；Step 4 的过程由第 2 行开始的 if 循环实现。但是在本文的算法中，第 2 行的循环是死循环，因为我们产生的标识是连续的、无限的。当研究 EG 时，本文不再关心标识的变化，而只考虑 EG 的结构。根据 EG 的获得可知，EG 的结构是由变迁的实施序列决定的。当我们将连续变迁和 A-变化替换为离散变迁时，新获得的传统 Petri 网和自适应 Petri 网具有相同的结构。因此，在自适应 Petri 网中如果存在一个变迁序列，则在传统 Petri 网也对应存在一样的变迁序列；反之亦然。它们的不同点在于在自适应 Petri 网前后两次实施同一变迁序列将得到不同的标识，但是在传统 Petri 网中，前后两次实施同一变迁序列将得到同一标识集。因此，EG 和 RG 具有相同的结构。证毕。 ■

对自适应系统来说，它需要根据环境的变化和系统本身的变化而实时改变系统的行为。其难点在于系统运行环境的开放性和系统的复杂性导致我们不知道环境具体是如何变化的，因此我们需要对环境进行学习来确定系统如何改变自身的行为。一旦我们能够预知环境的变化，那么系统的行为就是确定的，这就是一个一般的系统。在自适应 Petri 网的演化图中，环境的变化体现在节点的连续库所的标识上，而系统的行为体现在演化图的结构上。根据定理 4-1，我们知道自适应 Petri 网的演化图的结构和传统 Petri 网的可达图具有相同的结构，这说明自适应系统和一般传统系统具有相同的行为集合。但是自适应系统能够根据环境的变化自动地改变自身行为以适应环境的变化。这也是研究者提出自适应的原因：使软件具有一定的智能性、适应性，从而减少软件系统的人工维护成本。故接下来我们将给出自适应 Petri 网的自适应性分析。

4.4.2 自适应 Petri 网的自适应性验证

在 4.4.1 节，我们通过定理 4-1 证明了自适应 Petri 网能够描述自适应系统的切换行为，本节将证明自适应 Petri 网的适应性。首先我们定义自适应 Petri 网的环境：

定义 4-11（环境向量）：对自适应 Petri 网 PN^N，令 $P_e=\left\{p_1^e,p_2^e,\cdots,p_r^e\right\}$ 为所有 A-变迁的输入库所，则向量 $e=\left(m\left(p_1^e\right),m\left(p_2^e\right),\cdots,m\left(p_r^e\right)\right)$ 称为 PN^N 的一个环境向量。

定义 4-12（环境空间）：环境向量中的标识变化范围称为环境空间，也就是说所有的环境向量组成环境空间。环境空间记为 E。

由于自适应系统要根据环境的变化来选择合适的行为，这里的选择是离散。另一方面，在本文的模型中，我们利用神经网络来做出选择，但是神经网络的输出是连续的，因此我们需要对自适应选择连续化，用一个区间表示一个选择，称为选择区间。于是自适应系统

的所有行为对应于选择区间集合。显然任何两个选择区间I_1, I_2只有两种关系：$I_1 = I_2$或者$I_1 \bigcap I_2 = \Phi$。实际上，如果I_1, I_2有交集，不妨设$c \in I_1 \bigcap I_2$，则当神经网络的输出为 c 时，系统将会因为有两个不同的选择而产生故障乃至死机。

定义 4-13（环境空间中的一个二元关系）：设*nn*表示自适应 Petri 网中 A-变迁的个数，f_i表示第 *i* 个 A-变迁嵌入的神经网络所描述的函数，$f = (f_1, \cdots, f_{mn})$。*R* 是环境空间的一个二元关系满足：对 E 的任意两个向量e_1, e_2，$(e_1, e_2) \in R$当且仅当$f(e_1)$和$f(e_2)$位于同一个选择区间内。

根据二元关系 *R* 的定义，有引理 4.1：

引理 4-1：*R* 是一个等价关系。

证明：我们需要证明 *R* 满足自反性、对称性和传递性。设e_1, e_2, e_3是 *E* 中的任意三个环境向量，则

1）自反性：显然$f(e_1) = f(e_1)$，故$(e_1, e_1) \in R$，自反性成立；

2）对称性：如果$(e_1, e_2) \in R$，则$f(e_1)$和$f(e_2)$位于相同的区间，设为 *I*，则$f(e_1) \in I$且$f(e_2) \in I$，故$(e_2, e_1) \in R$，对称性成立；

3）传递性：如果$(e_1, e_2) \in R$，$(e_2, e_3) \in R$，则存在两个区间I_1, I_2使得$f(e_1) \in I_1$，$f(e_2) \in I_1$，且$f(e_2) \in I_2$，$f(e_3) \in I_2$，则$\varnothing \neq f(e_2) \in I_1 \bigcap I_2$；由于$I_1, I_2$满足$I_1 = I_2$和$I_1 \bigcap I_2 = \varnothing$二者之一，因此有$I_1 = I_2$，故$(e_1, e_3) \in R$。

因此我们可以根据等价关系 R 将环境空间进行等价类划分，等价类集合记为$\{E_i\}$。根据等价类划分的性质，我们有 1）$E = \bigcup E_i$，2）对任意的$i \neq j$，$E_i \bigcap E_j = \varnothing$。我们称每个$E_i$为一个**环境**。

为了使环境空间是完全的，本文做出如下假设：

假设 1：自适应系统的每一个动作必在某个环境值之下发生；每一个环境值将产生一个动作。

定义 4-14（行为）：PN^N中库所的标识表示PN^N的一个局部状态。如果PN^N的一个执行能够重复遍历一些局部状态，即重复遍历PN^N某个子图，则该执行称为PN^N的一个行

为。

设 *EG* 是 PN^N 的演化图。则根据行为的定义，它每遍历一个局部状态，就产生一个全局状态，而这个全局状态必包含在 *EG* 的一个节点中，因此每个状态必对应于 *EG* 中的一个子图。其次，由于该行为可以重复执行，因此对应的子图必是一个圈。其次，在考虑行为时，多个变迁可同时实施，但其顺序是随机的。因此生成 *EG* 时，我们将会产生多个分支，但是这些分支最后必然汇集到同一个节点。因此，一个行为对应于 *EG* 中的一个包含一个或者多个圈的子图，但是不同的圈具有相同的实施变迁集合，不同点在于变迁的实施次序不一样。因此我们有引理 4-2：

引理 4-2：每一个行为对应于 *EG* 中的一个子图，该子图包含一个或者多个圈。不同圈具有相同的实施变迁集合，但具有不同的实施次序。

每个行为对应的子图我们称为行为子图，记作 G_i。

引理 4-3：1）每一个环境 E_i 产生且仅产生一个行为子图 G_i；2）不同环境产生不同行为子图；3）所有环境对应的行为子图构造系统的全部行为，即令 *G* 为系统的行为集合，则$G = \cup G_i,\ G_i \cap G_j = \emptyset$。

证明：

1）令 E_i 是一个环境，$e_i \in E_i$。根据 PN^N 的结构和语义，e_i是 PN^N 的一个局部状态，因此它是某个行为的一部分。根据引理 4-2，存在 *EG* 中一个子图G_i，其中至少有一个节点 *q* 包含e_i。其次，设$\forall e_j \in E_i$且$e_i \neq e_j$，据前面所述，假设包含e_j的行为子图为G_i'；因为E_i是一个等价类，故$f(e_i)$和$f(e_j)$位于同一个选择区间内，于是在e_i, e_j环境值下，对每一个 A-变迁t_A，PN^N 将执行$(t_a^\bullet)^\bullet$中的同一个变迁，而e_i, e_j不参与其他变迁的选择与执行，故在e_j条件下执行的变迁与在e_j条件下执行的变迁相同，根据引理 4-2，$G_i = G_i'$。因此每一个环境产生且仅产生一个行为子图。

2）设E_i, E_j是两个不同的行为，其产生的行为子图分别为G_i, G_j。根据环境空间的等价类定理，对任意的$e_i \in E_i$，$e_j \in E_j$，存在一个 A-变迁t_a满足$f_i(e_i)$和$f_i(e_j)$位于不同的选择区间内，其中f_i是t_a所表示的神经网络描述的函数。因此在e_i条件下 PN^N 执行的$(t_a^\bullet)^\bullet$中的变迁不同于在e_j条件下 PN^N 执行的$(t_a^\bullet)^\bullet$中的变迁，因此$G_i \neq G_j$。

3）根据假设 1 和定理的 1）2），3）成立。

根据 1）2）3），该引理证毕。 ■

定理 4-2：令 E_i 和 E_j 为自适应 Petri 网 PN^N 的两个不同的环境，G_i 和 G_j 为对应的两个行为。如果 $E_i \to E_j$，则 $G_i \to G_j$。

证明：根据引理 4-3，G_i 和 G_j 是 E_i，E_j 产生的唯一的行为子图。当环境从 E_i 变化到 E_j 时，所有 A-变迁所表示的神经网络将重新计算输出，并执行每个 $(t_a^\bullet)^\bullet$ 中的相应变迁。根据引理 4-3，这些变迁在 G_j 中，因此系统的行为将由 G_i 变化到 G_j。证毕。 ■

最后，假设一个系统可用 PN^N 进行建模，本文有如下一些定理和结论。

定义 4-15（自适应性）：系统的自适应性是指系统具有动态改变行为以适应环境变化的性质。

定义 4-16：一个系统称为自适应系统，如果它具有自适应性。

根据定义 4-15 和定理 5-2，有：

定理 4-3：由 PN^N 描述的系统是一个自适应系统。

4.5 本章小结

本章以一个制造企业为例，提出了一种新的能够对自适应软件系统进行形式化建模的形式化语言——自适应 Petri 网。该语言能够对环境建模，具有学习功能和合作功能，通过对环境的学习，促使不同子系统间合作来完成预期的系统行为。同时，本文通过理论证明了自适应 Petri 网的自适应性。此外，该模型还适合对基于组件的系统进行建模，每一个组件可以用嵌入一个或者零个神经网络的闭合 process 网来建模，不同 process 网可以通过交会通信机制来对组件间的合作进行建模。故自适应 Petri 网可以扩展到由多个组件组成的软件系统，具有良好的扩展性。

5 实例分析

本章通过三个实例来介绍本文前面提到的建模方法和验证技术，它们分别是两轮移动机器人例子、室内自动控温器例子和制造企业系统例子。

5.1 第一类自适应系统的实例分析

本节我们根据第三章提到的对不同切换系统的建模方法对两个实例分别进行建模和分析，这两个例子分别是两轮移动机器人的控制设计和室温的调节控制。

5.1.1 模糊自适应系统的实例分析

首先，我们以两轮移动机器人为例介绍本文在第 3.1 节介绍的对切换模糊系统的建模和验证。

两轮移动机器人能够根据控制设计，通过改变后轴的高度和前进角度来确保其能在基准轨道上正确行驶，如图 5.1，其中 y 表示后轴高度，θ 表示偏离基准航道的角度。移动机器人的动态行为是一个非线性系统。为保证机器人能稳定行走，则需保证

$$\lim_{t\to\infty} y(t) = 0，\lim_{t\to\infty} \theta(t) = 0 \qquad 5\text{-}(1)$$

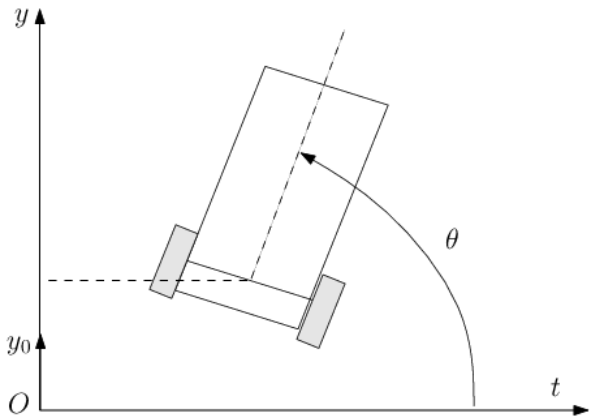


图 5.1 两轮移动机器人

为了能够简单有效的对移动机器人进行控制，Tanaka 等[65]引入了切换模糊模型来描述移动机器人的非线性行为：

Region Rule 1: 如果$d < \theta(t) < 3.131$，则

Local Plant Rule 1：如果$\theta(t)$为h_{11}，则

$$\dot{x}(t) = A_{11}x(t) + B_{11}u(t)$$

Local Plant Rule 2：如果$\theta(t)$为h_{12}，则

$$\dot{x}(t) = A_{12}x(t) + B_{12}u(t)$$

Region Rule 2: 如果$-d < \theta(t) < d$，则

Local Plant Rule 1：如果$\theta(t)$为h_{21}，则

$$\dot{x}(t) = A_{21}x(t) + B_{21}u(t)$$

Local Plant Rule 2：如果$\theta(t)$为h_{22}，则

$$\dot{x}(t) = A_{22}x(t) + B_{22}u(t)$$

Region Rule 3: 如果$-3.131 < \theta(t) < -d$，则

Local Plant Rule 1：如果$\theta(t)$为h_{31}，则

$$\dot{x}(t) = A_{31}x(t) + B_{31}u(t)$$

Local Plant Rule 2：如果$\theta(t)$为h_{32}，则

$$\dot{x}(t) = A_{32}x(t) + B_{32}u(t)$$

其中，$x=\begin{pmatrix} y(t) \\ \theta(t) \end{pmatrix}$，$u=\begin{pmatrix} u_1(t) \\ u_2(t) \end{pmatrix}$，$d=\frac{\pi}{50}$，$C=0.5$，$A_{11}=A_{12}=A_{31}=A_{32}=0$，$A_{21}=\begin{pmatrix} 0 & C \\ 0 & 0 \end{pmatrix}$，

$A_{22}=\begin{pmatrix} 0 & C\frac{\sin d}{d} \\ 0 & 0 \end{pmatrix}$，$B_{11}=\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$，$B_{12}=\begin{pmatrix} 0.01 & 0 \\ 0 & 1 \end{pmatrix}$，$B_{21}==B_{22}=\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$，$B_{31}=\begin{pmatrix} -0.01 & 0 \\ 0 & 1 \end{pmatrix}$，$B_{32}=\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$；

$h_{ij}(i=1,2,3;j=1,2)$ 为模糊数，其隶属度分别为 $h_{11}(\theta)=\frac{\sin\theta-0.01}{0.99}$，$h_{12}(\theta)=\frac{1-\sin\theta}{0.99}$，

$h_{21}(\theta)=\frac{\frac{\sin\theta}{\theta}-\frac{\sin d}{d}}{1-\frac{\sin d}{d}}$，$h_{22}(\theta)=\frac{1-\frac{\sin d}{d}}{1-\frac{\sin d}{d}}$，$h_{31}(\theta)=\frac{\sin\theta+0.01}{-0.99}$，$h_{31}(\theta)=\frac{-1-\sin\theta}{-0.99}$。初始条件为

$x_0=\begin{pmatrix} y_0 \\ \theta_0 \end{pmatrix}=\begin{pmatrix} 2 \\ 1 \end{pmatrix}$。

系统的输入控制为：

$$u(t) = -\sum_{i=1}^{3}\sum_{j=1}^{2}\nu_i(\theta(t))h_{ij}(\theta(t))F_{ij}x(t)$$

其中$\nu_i(\theta(t)) = \begin{cases} 1 & \theta \in \text{Region } i \\ 0 & \text{otherwise} \end{cases}$ $i = 1, 2, 3.$

我们的目的是选择合适的 F_{ij} 使系统的动态行为满足 5-（1）。本文试验下面这组数据：

$F_{11}=F_{12}=\begin{pmatrix} 0.8233 & 0.3908 \\ 0.1711 & 0.3604 \end{pmatrix}$，$F_{21}=F_{22}=\begin{pmatrix} 0 & 0 \\ 0.8163 & 0.6667 \end{pmatrix}$，$F_{11}=F_{12}=\begin{pmatrix} -0.8233 & -0.3908 \\ 0.1711 & 0.3604 \end{pmatrix}$

首先，对每个子系统去模糊化，得到：

Region Rule 1: 如果$d < \theta(t) < 3.131$，则

$$\begin{cases}\dot{y} = (-0.8233y - 0.3908\theta)\sin\theta \\ \dot{\theta} = -0.1711y - 0.3604\theta\end{cases}$$

Region Rule 2: 如果$-d < \theta(t) < d$，则

$$\begin{cases}\dot{y} = \frac{1}{2}\sin\theta \\ \dot{\theta} = -0.8163y - 0.6667\theta\end{cases}$$

Region Rule 3: 如果$-3.131 < \theta(t) < -d$，则

$$\begin{cases}\dot{y} = (-0.8233y - 0.3908\theta)\sin\theta - (0.8315y + 0.3947\theta) \\ \dot{\theta} = -0.1711y - 0.3604\theta\end{cases}$$

由此我们得到的原始的混合自动机如图 5.2.

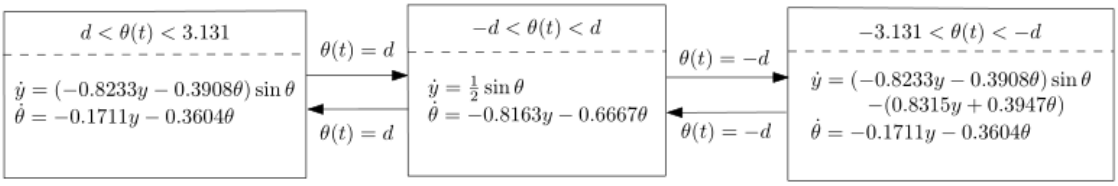


图 5.2 两轮移动机器人的混合自动机模型

然后对其进行状态分割，得到仿射混合自动机，如图 5.3。

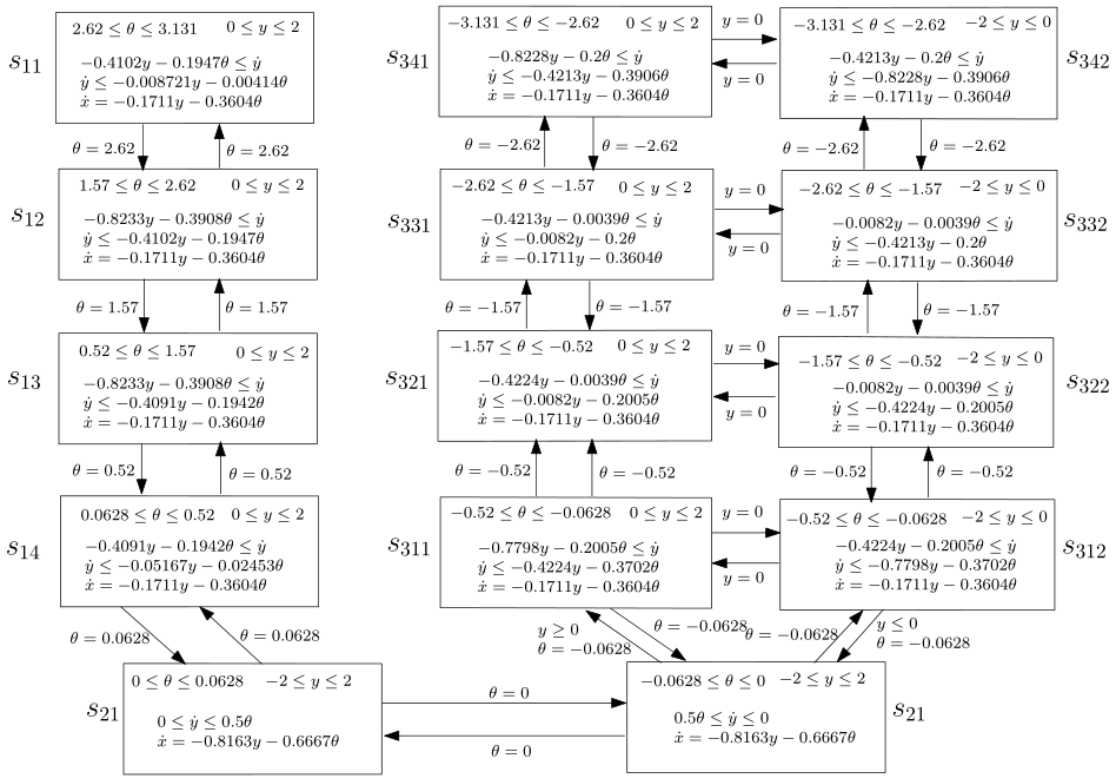


图 5.3 两轮移动机器人的仿射自动机模型

将图 5.3 的仿射自动机作为 PHAVer 的输入，计算可达图，并利用 matlab 将二维数据

可视化，本文分别得到 y 和 θ 随时间的可达状态变化，如图 5.4 和 5.5，其中绿色区域是由 PHAVer 计算的可达状态，红色曲线是移动机器人的实际运行轨迹。

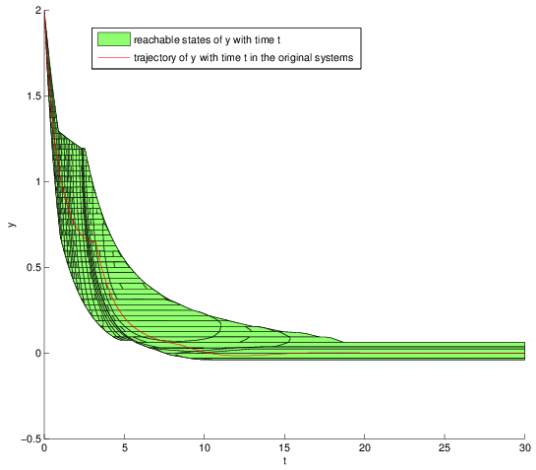


图 5.4 (*t*, *y*)的可达状态和实际轨迹曲线

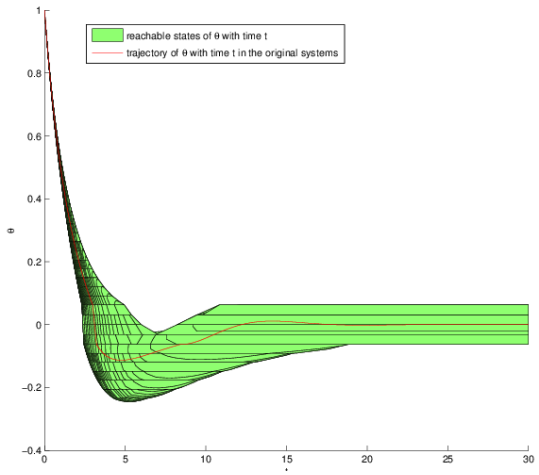
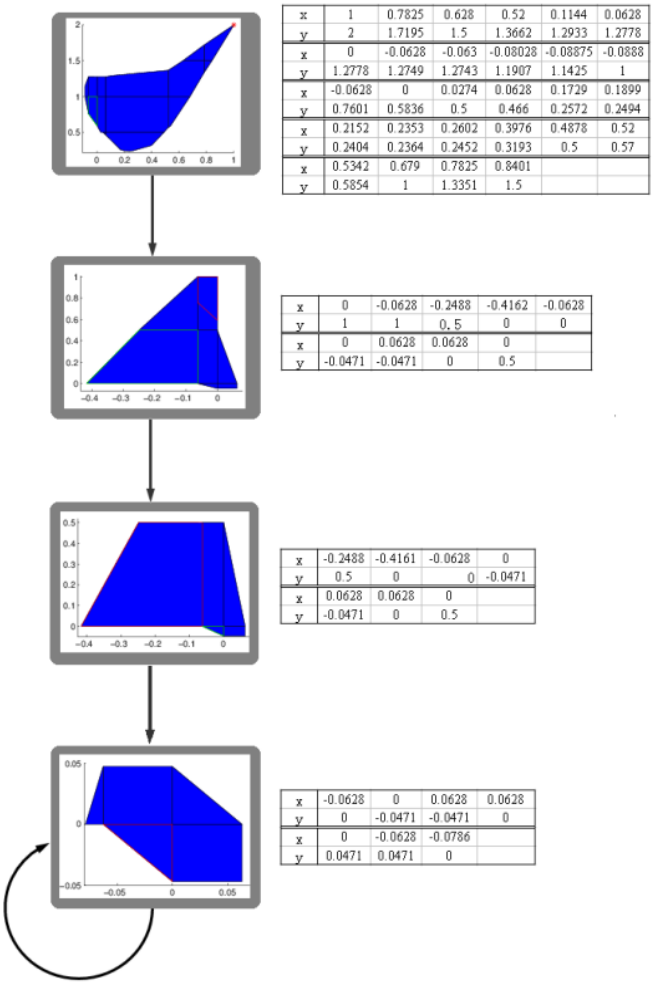


图 5.5 (*t*, *θ*)的可达状态和实际轨迹曲线

由图可以初步断定系统满足 5-（1）的稳定性要求，接下来我们将查找稳定区域。首先我们计算第一个 2 分钟的可达状态，然后从中选取一个可达多边形作为下一个 2 分钟的初始状态，继续计算可达状态，如此循环下去，直到我们找到一个小空间 D，使得以 D 为初始状态的可达状态还是 D。我们得到可达状态变化如图 5.6，其中图中蓝色区域是 PHAVer 计算的可达区域；右边是对应的边界的顶点坐标。

根据实验结果，我们发现当计算得到第四轮可达区域时，以其中的任意一个多边形为初始状态，得到的可达图相同，均为第四轮得到的可达区域。由此，我们得到了一个可达区域 D，其顶点坐标为 $\{(-0.0628,0),(0,-0.0471),(0.0628,-0.0471),(0.0628,0),(0,0.0471),(-0.0628,0.0471),(-0.0786,0)\}$，显然这是一个包含（0,0）的一个小空间。

当对图 5.2 的混合自动机划分越细时，我们得到更小的包含（0,0）的可达区域，即随着分割的增多，D 最终将会退化为（0,0）这个点。当我们继续选取其他不同的系统初值，重复上述过程，我们同样可以得到一个在（0,0）附近且包含（0,0）的稳定区域，因此该系统是稳定的。



x	1	0.7825	0.628	0.52	0.1144	0.0628
y	2	1.7195	1.5	1.3662	1.2933	1.2778
x	0	-0.0628	-0.063	-0.08028	-0.08875	-0.0888
y	1.2778	1.2749	1.2743	1.1907	1.1425	1
x	-0.0628	0	0.0274	0.0628	0.1729	0.1899
y	0.7601	0.5836	0.5	0.466	0.2572	0.2494
x	0.2152	0.2353	0.2602	0.3976	0.4878	0.52
y	0.2404	0.2364	0.2452	0.3193	0.5	0.57
x	0.5342	0.679	0.7825	0.8401		
y	0.5854	1	1.3351	1.5		

x	0	-0.0628	-0.2488	-0.4162	-0.0628
y	1	1	0. 5	0	0
x	0	0.0628	0.0628	0	
y	-0.0471	-0.0471	0	0.5	

x	-0.2488	-0.4161	-0.0628	0
y	0.5	0	0	-0.0471
x	0.0628	0.0628	0	
y	-0.0471	0	0.5	

x	-0.0628	0	0.0628	0.0628
y	0	-0.0471	-0.0471	0
x	0	-0.0628	-0.0786	
y	0.0471	0.0471	0	

图 5.6 可达区域的搜索过程图

5.1.2 随机自适应系统分实例分析

在本节，我们考虑装有自动调温装置的房间的温度控制[66]。自动调温装置有 ON 和 OFF 两个状态，它通过观测室内温度的变化自动在这两个状态之间进行切换，从而来维持房间的温度。具体切换过程如图 5.7。

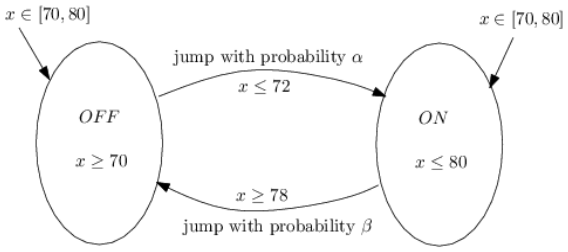


图 5.7 自动调温装置的切换示意图

假设当室温一旦低于 72°F 时，调温装置将切换到 ON 状态提高室内温度；当室温高于 78°F 时，调温装置将切换到 OFF 状态降低室温。但是在实际过程中，切换需要一定的时间，因此室内温度可能继续升高或者降低，因此我们设定的温度范围为[70°F,80°F]。假设不同

状态间切换所需的时间是一个随机数，因此该切换过程是一个离散时间 Markov 链（因此，在离散化过程中，我们无需对切换控制 Markov 链进行离散），假设其一步转移概率为：

$$P=\begin{pmatrix}1-\alpha & \alpha\\ \beta & 1-\beta\end{pmatrix}$$

每个状态的连续变化过程为：

$$dX(\tau)=\begin{cases}-\frac{a}{C}(X(\tau)-x_a)d\tau+\frac{1}{C}dB(\tau), & s=0\\ -\frac{a}{C}(X(\tau)-x_a)d\tau+\frac{r}{C}d\tau+\frac{1}{C}dB(\tau), & s=1\end{cases}$$

其中 a 为平均热量损失率，C 为室内的平均热容量，x_a 为环境温度（假定是一个常数），r 为热量获得率，$B(\tau)$ 为标识布朗运动，用来描述影响温度变化的噪声，s=0 表示 OFF 状态，s=1 表示 ON 状态。

在本文的例子中，$a=0.1$，$C=1$，$x_a=10.5$，$\alpha=\beta=0.8$，$r=10$。状态空间网格分割的网格长度 $\delta=0.5$，随机微分方程离散化的步长 $h=0.01$。因此，本文得到的 S-DPN 模型如图 5.8。

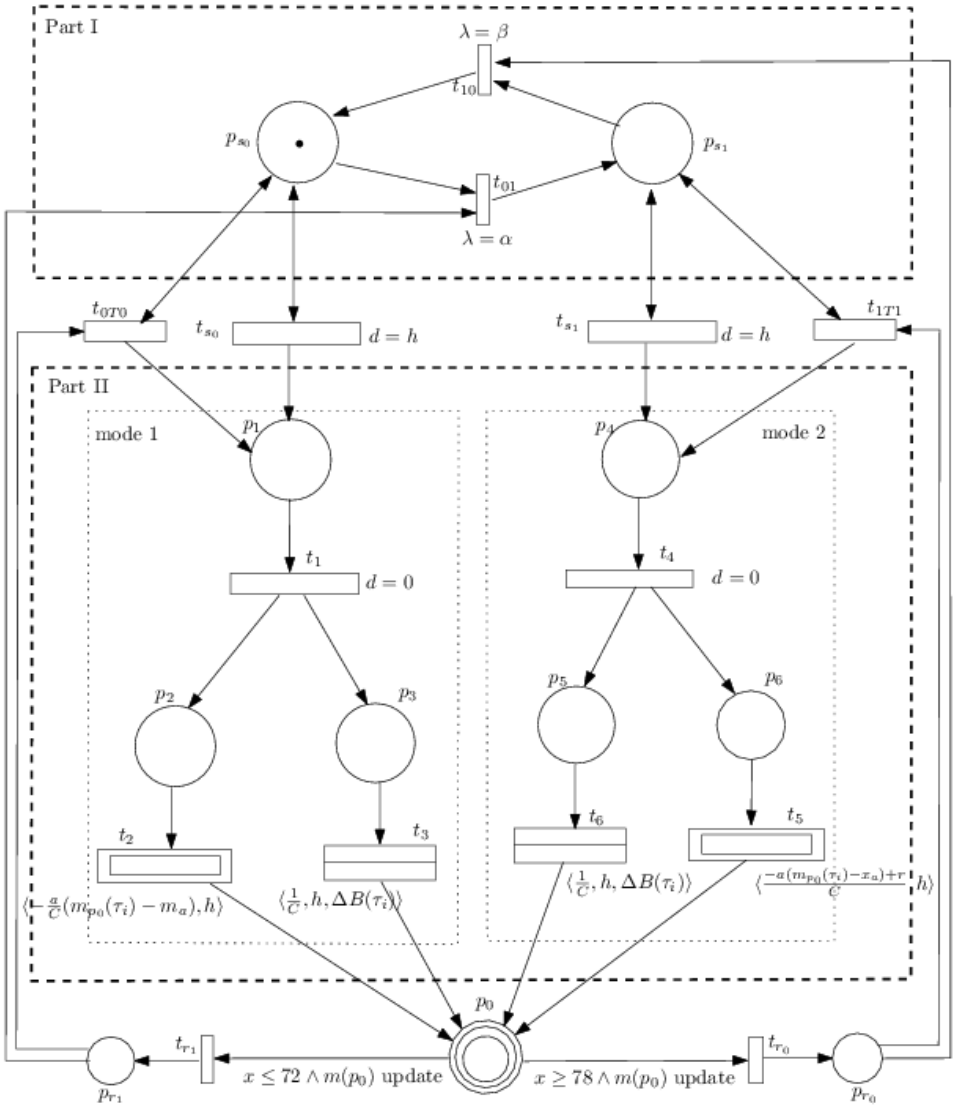


图 5.8 室温调节系统的随机微分 Petri 网模型

图中 p_0 为随机微分库所，其标识 $m(p_0)$ 表示室内温度。当 $m(p_0)=72$ 时， t_{r1} 激活并实施，于是 t_{oTo}, t_{01} 和 t_{s0} 均被激活，但是 t_{s0} 为时间变迁，无法实施。于是 t_{oTo}, t_{01} 有不同的概率被实施，从而实现不同模式间的切换。当 $m(p_0)=78$ 时，同理可分析。当 $72<m(p_0)<78$ 时，该 Petri 网只能在某个模式中进行演化，且因 t_{s0} 或者 t_{s1} 的时间延迟为 h，故 $m(p_0)$ 每隔 h 时间更新一次，记为随机微分方程的离散部分。

下面进行模型验证。假定给定的时间为[0,200]，我们需计算或验证的性质有：

1）计算：在任意时刻 τ， $X(\tau)=75^oF$ 的概率；

2）验证：R. 对任意初值， $X(200)>78^oF$ 的概率不大于 0.2。

首先我们得到图 5.8 的演化图和对应的 Markov 链，如图 5.9 和图 5.10。

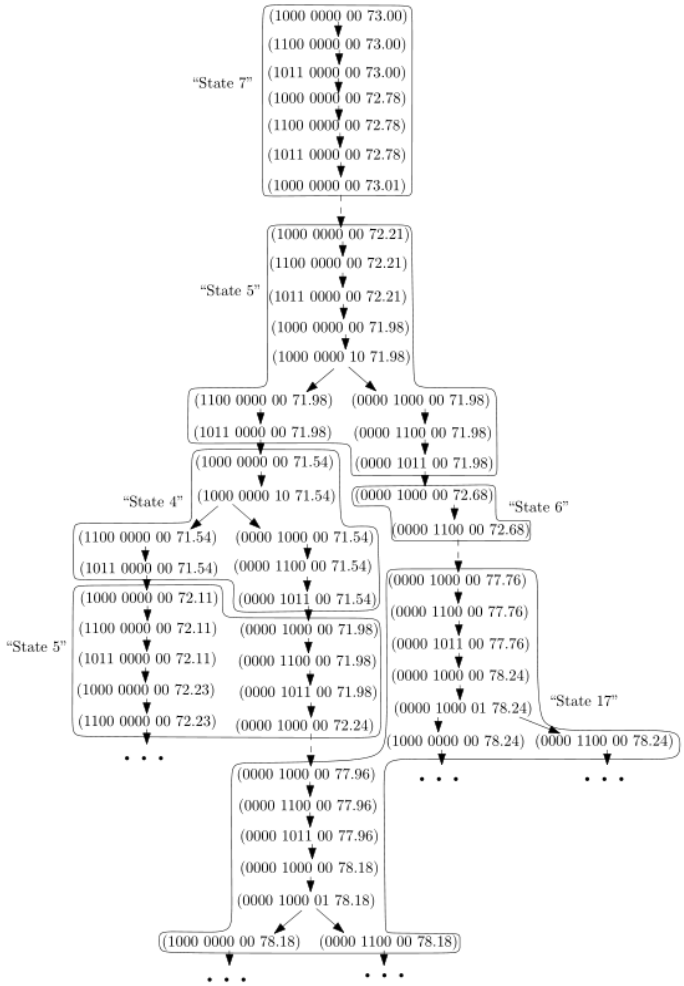


图 5.9 温度控制的随机微分 Petri 网的演化图

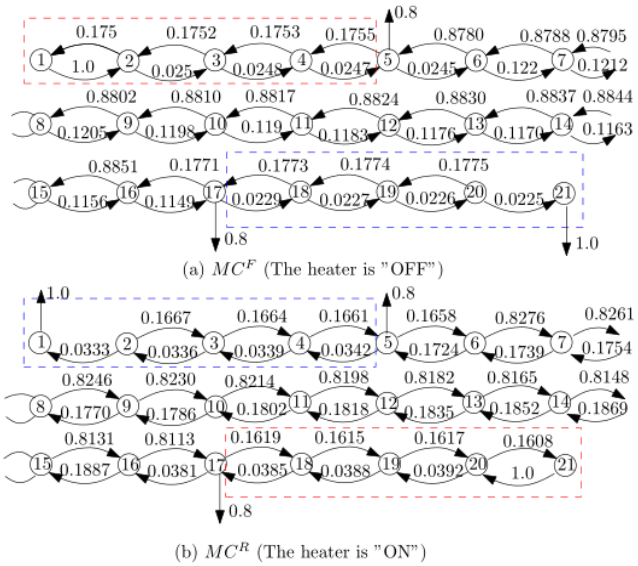


图 5.10 对应的 Markov 链模型

图 5.11 给出了在初值温度为 73°F 时，不同时刻温度为 75°F 的概率。在 PRISM 中即可导出对应时刻的概率，也可直接导出时间-概率图。本文的图形是根据导数的数据，利用 matlab 画出来的。

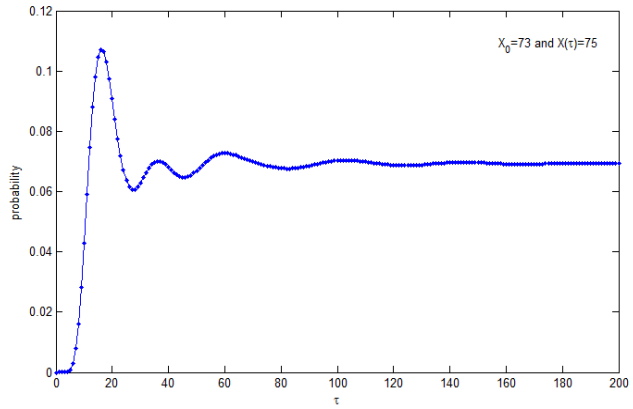


图 5.11 当 X_0=73 °F 时，系统在不同时刻 $X(\tau)=75$ °F 的概率

将每个状态作为初始，分别计算室温处于[72°F,78°F]的概率。表 5.1 给出了部分初值下，系统处于[72°F,78°F]内的各个状态的概率以及在[72°F,78°F]内的总概率。根据实验结果，我们得出性质 R 成立。

表 5.1 不同初值条件下，t=200 时，室温处于不同值的概率

初值(°F) / 状态	72.5	73	74	75.5	76.5	77.5
s_5(72°F)	0.100326	0.099934	0.099298	0.098877	0.099018	0.099482
s_6	0.064371	0.06417	0.063856	0.063710	0.063850	0.064162
s_7	0.067758	0.06758	0.067308	0.067195	0.067333	0.067622
s_8	0.068403	0.06827	0.068070	0.068007	0.068132	0.068371

s_9	0.068701	0.06861	0.068500	0.068487	0.068589	0.068763
s_{10}	0.068969	0.06894	0.068905	0.068937	0.069007	0.069106
s_{11}	0.069249	0.06927	0.069320	0.069388	0.069420	0.069434
s_{12}	0.069538	0.06961	0.069735	0.069829	0.069816	0.069743
s_{13}	0.069784	0.06990	0.070091	0.070199	0.070138	0.069978
s_{14}	0.069771	0.06993	0.070165	0.070277	0.070168	0.069928
s_{15}	0.068633	0.06881	0.069085	0.069189	0.069040	0.068737
s_{16}	0.062725	0.06291	0.063171	0.063250	0.063078	0.062753
$s_{17}(78^oF)$	0.097660	0.09798	0.098470	0.098662	0.098401	0.097867
概率和	0.945889	0.945914	0.945976	0.946006	0.945990	0.945947

5.2 第二类自适应系统的实例分析

在第四章中，我们用自适应 Petri 网对制造企业进行了建模，并从理论上证明了自适应 Petri 网的自适应性。本节将对该实例进行实验模拟。

第一步，制造系统的 APN 建模。由于神经网络的输出是连续值，因此需对子工厂和供货商的选择进行连续化处理。对子工厂有：$[0,1)$ 表示对 B_1 的选择，$[1,2)$ 表示对 B_2 的选择，$[2,3)$ 表示对 B_3 的选择，$[3,4)$ 表示对 B_4 的选择；对供货商有：$[0,1)$ 表示对 S_1 的选择，$[1,2)$ 表示对 S_2 的选择。于是有下述选择规则：

Rule 1：如果 NN_1 的输出在区间 $[0,1)$ 内，则选择 B_1 进行生产；

Rule 2：如果 NN_1 的输出在区间 $[1,2)$ 内，则选择 B_2 进行生产；

Rule 3：如果 NN_1 的输出在区间 $[2,3)$ 内，则选择 B_3 进行生产；

Rule 4：如果 NN_1 的输出在区间 $[3,4)$ 内，则选择 B_4 进行生产；

Rule 5：如果 NN_2 的输出在区间 $[0,1)$ 内，则选择 S_1 提供原材料；

Rule 6：如果 NN_2 的输出在区间 $[1,2)$ 内，则选择 S_2 提供原材料。

根据上述自适应规则，我们可以得到制造系统的 APN 模型中两个 A-变迁的输出库所的选择条件为：

cond_{11}：$0 \le m(p_{42}) < 1$； cond_{12}：$1 \le m(p_{42}) < 2$； cond_{13}：$2 \le m(p_{42}) < 3$

cond_{14}：$3 \le m(p_{42}) < 4$； cond_{21}：$0 \le m(p_{51}) < 1$；cond_{12}：$1 \le m(p_{51}) < 2$。

因此我们易得制造系统的 APN 模型，如图 4.16。其中$p_{34} \sim p_{41}$和$p_{47} \sim p_{50}$为连续库所，表示环境因子，则环境向量为$e = (m_{34}, \cdots, m_{41}, m_{47}, \cdots, m_{50})$；连续变迁$t_1^C \sim t_{12}^C$对环境变量赋值。

根据行为选择规则，我们可以将整个环境空间划分为 8 个环境，虽然我们并不知道每个环境的边界是多少，但是我们可以进行形式化描述，具体为：

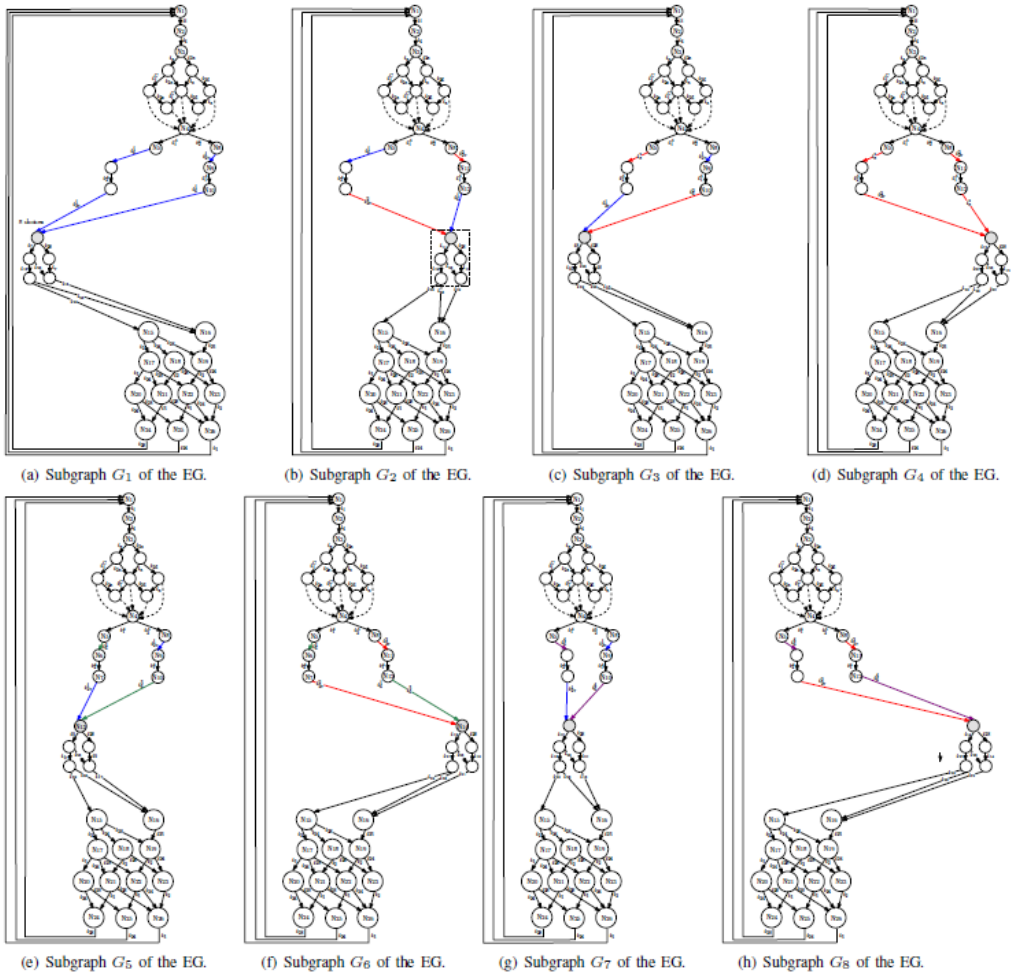
$E_{11} = \{e \mid 0 \le f_1(e) < 1 \wedge 0 \le f_2(e) < 1\}$， $E_{12} = \{e \mid 0 \le f_1(e) < 1 \wedge 1 \le f_2(e) < 2\}$，

$E_{21} = \{e \mid 1 \le f_1(e) < 2 \wedge 0 \le f_2(e) < 1\}$， $E_{22} = \{e \mid 1 \le f_1(e) < 2 \wedge 1 \le f_2(e) < 2\}$，

$E_{31} = \{e \mid 2 \le f_1(e) < 3 \wedge 0 \le f_2(e) < 1\}$， $E_{32} = \{e \mid 2 \le f_1(e) < 3 \wedge 1 \le f_2(e) < 2\}$，

$E_{41} = \{e \mid 3 \le f_1(e) < 4 \wedge 0 \le f_2(e) < 1\}$， $E_{42} = \{e \mid 3 \le f_1(e) < 4 \wedge 1 \le f_2(e) < 2\}$。

根据引理 5-3， 8 个行为子图为：



(a) Subgraph G_1 of the EG. (b) Subgraph G_2 of the EG. (c) Subgraph G_3 of the EG. (d) Subgraph G_4 of the EG.

(e) Subgraph G_5 of the EG. (f) Subgraph G_6 of the EG. (g) Subgraph G_7 of the EG. (h) Subgraph G_8 of the EG.

图 5.12 制造系统的 8 个行为子图

第二步，根据样本对自适应 Petri 网中所嵌入的神经网络进行训练，确定网络结构。样本数据和训练结果见附录。这里我们根据第一步的连续化操作，事先对训练样本数据进行

如下预处理：样本中作的输入数据是环境变量归一化后的数据，输出数据我们设置为离散选择所对应的连续化区间的中心，例如，如果某个样本数据中，输入所对应的选择为B_1和S_2，则我们预处理后对应的神经网络的输出样本为 0.5 和 1.5（这是因为B_1所对应的连续化区间为$[0,1)$，S_2所对应的连续化区间为$[1,2)$）。

第三步，对训练好的自适应 Petri 网进行模拟。本文选择三种数据进行模拟，分别为：

第一组数据：环境值 e_1=(0.5918, 0.3776, 0.9516, 0.7424, 0.8871, 0.5989, 0.6896, 0.3911, 0.8087, 0.0667, 0.1958, 0.9748)； 需选择目标：B_3和S_2。

第二组数据：环境值 e_2=(0.3087, 0.0851, 0.8768, 0.5907, 0.3031, 0.2681, 0.3399, 0.3674, 0.4819, 0.1605, 0.7543, 0.7935)；需选择目标：B_2和S_2。

第三组数据：环境值 e_3=(0.2514, 0.3257, 0.2196, 0.0265, 0.3099, 0.4572, 0.1782, 0.2457, 0.2855, 0.243, 0.6425, 0.9569)；需选择目标：B_1和S_1。

假设当前环境值为 e_1，此时神经网络 NN1 和 NN2 的输出为 2.9775 和 1.2786，由于$2.9775\in[2,3)$，$1.2786\in[1,2)$，故第三个生成工厂B_3和第二个供货商S_2被选择用来完成生产过程，也即 APN 处于G_6行为子图中（图 5.12(*f*)）；当环境向量有 e_1 变化到 e_2，此时，系统将重新计算 NN1 和 NN2 的输出，分别为 1.754 和 1.316，于是第二个生成工厂B_2和第二个供货商S_2被选择用来完成生产过程，也即 APN 处于G_4行为子图中（图 5.12(*d*)）；同理，当环境向量由 e_2 变化到 e_3 时，由于神经计算得到的输出为 0.8798 和 0.5662，于是系统选择B_1和S_1来完成生成过程。因此，当环境发生变化时，系统能够根据环境的变化，通过局部计算而进行全局行为的自适应。图 5.13 给出了制造系统的 APN 模型的模拟实验结果：初始时刻，环境值为 e_1，此时系统选择点位于(B_3,S_2)；当 *t*=19 时，环境由 e_1 变化到 e_2，对应的系统重新进行选择，位于(B_2,S_2)；接着，在 *t*=44 时，环境继续发生变化，从 e_2 变化到 e_3，此时系统对应的选择为(B_1,S_1)。

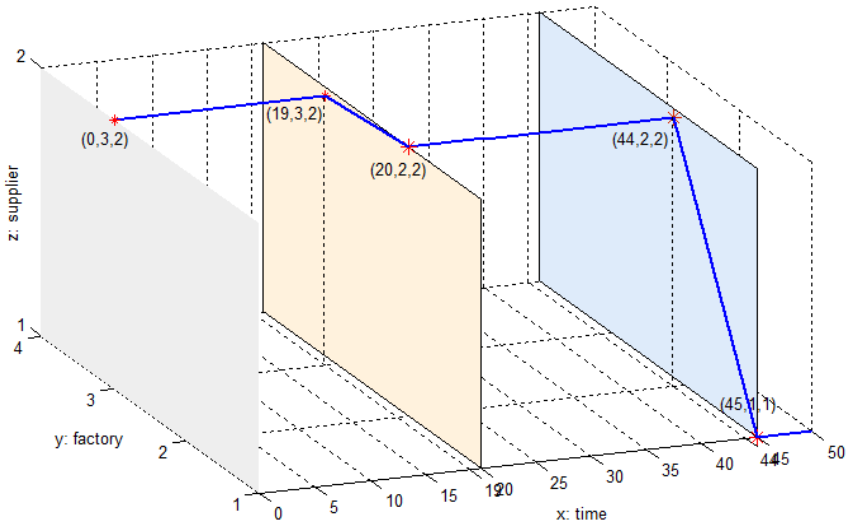


图 5.13 制造系统的模拟结果

5.3 本章小结

本章为实验部分，本文采用不同实例来实践前面提出的对不同自适应系统的建模方法和验证技术。本文用混合自动机对两轮移动机器人进行建模，并用 PHAVer 通过可达状态来分析系统的稳定性；用随机微分 Petri 网来对室内自动调温装置进行建模，并用 PRISM 来计算装置在不同时刻处于不同状态的概率以及验证系统的不同时刻的性质；最后用自适应 Petri 网来对制造系统进行建模和分析，并通过实验来模拟自适应 Petri 网的执行结果。

6 总结和展望

当前研究者对自适应系统进行了很多的研究，提出了很多的框架模型，也实现了一些自适应软件系统和相关平台。但是，在自适应软件的理论研究，尤其是自适应软件的形式化理论研究相对较少。本文从软件建模和验证方面研究了自适应软件的形式化建模语言和验证方法。主要进行了如下几方面的工作：

1. 提出了对切换模糊系统的形式化建模和稳定性验证。切换模糊系统是一类基于控制理论的自适应系统，它根据控制变量和状态变量，结合模糊规则来进行子系统间的切换。本文通过混合自动机对其建模，并用模型检查工具 PHAVer 计算可达状态，通过可达状态的演化来进行稳定性的验证。
2. 提出了一种新的形式化建模语言——随机微分 Petri 网对切换随机系统进行建模和验证。随机微分 Petri 网中包含两部分：第一部分是一般的随机 Petri 网，用来对切换规则——Markov 链建模；第二部分是对随机微分方程建模，引入了两类新的变迁：微分变迁和随机微分变迁，其中微分变迁用来描述连续变量的变化，随机微分变迁用来描述随机变量的变化。最后通过随机微分 Petri 网的可达状态进行等价类划分，构造 Markov 链，并用概率检查工具 PRISM 继续性质的验证。
3. 对环境影响的自适应系统进行形式化建模。此类自适应系统因环境的不确定性而变得更为复杂。本文引入人工智能算法，将神经网络和 Petri 网有机结合，建立了自适应 Petri 网模型，并对其自适应性质进行了分析。该模型既能够描述系统行为的变化，同时能够对环境进行建模。其次，该模型是一个基于过程的模型，具有良好的扩展性。

将来，本文还可以进行一些更深入的研究。例如，对于切换模糊系统，本文仅给出了如何对其进行稳定性验证，将来还可以进行如下研究：例如对于不稳定的系统，如何通过模型检查来确定合理的参数，从而使系统达到稳定。对于切换随机系统，还可以进行更多的实验来说明所提方法的有效性，以及如何确定合理的离散时间步长和状态分割参数以达到给定性质的验证。对自适应系统，能否建立不同模型之间的联系和转化，或者能否构造更为一般的形式化模型和验证理论，以期建立一个统一的形式化模型。如何对已有的验证工具进行扩展或者开发新的验证工具，实现对自适应软件系统的模型验证；如何建立基于自适应模型的自适应需求的描述；开发从高层设计模型到底层程序实现的转化工具等等。

参考文献

[1] Seetharaman G., Lakhotia A, Blasch E P. Unmanned Vehicles Come of Age: The DARPA Grand Challenge. Computer, 2006, 39(12): 26-29.

[2] Georgas J C, Taylor R N. Policy-based self-adaptive architectures: a feasibility study in the robotics domain[C]. Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems. ACM, 2008: 105-112.

[3] Peper C, Schneider D. Component engineering for adaptive ad-hoc systems[C]. Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems. ACM, 2008: 49-56.

[4] Ravindranathan M, Leitch R. Heterogeneous intelligent control systems[C]. IEE Proceedings- Control Theory and Applications. IET, 1998, 145(6): 551-558.

[5] Salehie M, Tahvildari L. Self-adaptive software: Landscape and research challenges[J]. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2009, 4(2): 14.

[6] Oreizy P, Gorlick M M, Taylor R N, et al. An architecture-based approach to self-adaptive software[J]. IEEE Intelligent systems, 1999, 14(3): 54-62.

[7] Brown G, Cheng B H C, Goldsby H, *et al*. Goal-oriented specification of adaptation requirements engineering in adaptive systems[C]. Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems. ACM, 2006: 23-29.

[8] Dardenne A, Van Lamsweerde A, Fickas S. Goal-directed requirements acquisition[J]. Science of computer programming, 1993, 20(1): 3-50.

[9] Whittle J, Sawyer P, Bencomo N, *et al*. A language for self-adaptive system requirements[C]. International Workshop on Service-Oriented Computing: Consequences for Engineering Requirements, 2008. IEEE, 2008: 24-29.

[10] Nakagawa H, Ohsuga A, Honiden S. Constructing self-adaptive systems using a KAOS model[C]. Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, 2008. IEEE, 2008: 132-137.

[11] Sawyer P, Bencomo N, Whittle J, *et al*. Requirements-aware systems: A research agenda for re for self-adaptive systems[C]. 18th IEEE International Requirements Engineering Conference (RE), 2010. IEEE, 2010: 95-103.

[12] Qureshi N A, Perini A, Ernst N A, *et al*. Towards a continuous requirements engineering

framework for self-adaptive systems[C]. 2010 First International Workshop on Requirements@ Run. Time. IEEE, 2010: 9-16.

[13] den Hamer P, Skramstad T. Autonomic service-oriented architecture for resilient complex systems[C]. 30th IEEE Symposium on Reliable Distributed Systems Workshops, 2011. IEEE, 2011: 62-66.

[14] Garlan D, Cheng S W, Schmerl B. Increasing system dependability through architecture-based self-repair[M]. Architecting dependable systems. Springer Berlin Heidelberg, 2003: 61-89.

[15] Garlan D, Cheng S W, Huang A C, *et al*. Rainbow: Architecture-based self-adaptation with reusable infrastructure[J]. Computer, 2004, 37(10): 46-54.

[16] Richter U, Mnif M, Branke J, *et al*. Towards a generic observer/controller architecture for Organic Computing[J]. GI Jahrestagung (1), 2006, 93: 112-119.

[17] Gomaa H, Hussein M. Model-based software design and adaptation[C]. Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems. IEEE Computer Society, 2007: 7.

[18] Tajalli H. A reference architecture for integrated self-adaptive software environments[D]. UNIVERSITY OF SOUTHERN CALIFORNIA, 2014.

[19] 高晖，张莉，李琳.软件体系结构层次的软件适应性预测模型[J]. 软件学报，2010, 21(9): 2118-2134.

[20] 李长云，李赣生，何频捷．一种形式化的动态体系结构描述语言[J]. 软件学报，2006，17(6): 1349-1359．

[21] 黄罡，王千祥，梅宏，杨芙清．基于软件体系结构的反射式中间件研究[J]. 软件学报，2003,14(11): 1819-1826.

[22] Fok C L, Roman G C, Lu C. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks[J]. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2009, 4(3): 16.

[23] 吕建，陶先平，马晓星，胡昊，徐锋，曹春．基于 Agent 的网构软件模型研究[J]. 中国科学(E 辑)，2005,35(12):1233-1253.

[24] 赵欣培，李明树，王青，陈振冲，梁金能.一种基于 Agent 的自适应软件过程模型[J]. 软件学报, 2004,15(3):348-359.

[25] Schmitt J, Roth M, Kiefhaber R, *et al*. Realizing self-x properties by an automated planner[C]. Proceedings of the 8th ACM international conference on Autonomic computing. ACM, 2011: 185-186.

[26] 赵祺,刘譞哲,王旭东,黄罡,梅宏.一种面向富客户端应用的运行时自适应中间件[J]. 软件学报, 2013, 24(7): 1419-1435.

[27] Bencomo N, Grace P, Flores C, *et al*. Genie: Supporting the model driven development of reflective, component-based adaptive systems[C]. Proceedings of the 30th international conference on Software engineering. ACM, 2008: 811-814.

[28] Castañeda B L. A Reference Architecture for Component-Based Self-Adaptive Software Systems: [硕士学位论文][D]. Colombia: Universidad ICESI, 2012.

[29] Dragone M. Building Self-adaptive Software Systems with Component, Services & Agents Technologies: Self-OSGi[M]. Agents and Artificial Intelligence. Springer Berlin Heidelberg, 2013: 300-316.

[30] Mishra A, Misra A K. Component assessment and proactive model for support of dynamic integration in self-adaptive system[J]. SIGSOFT Software Engineering Notes, 2009, 34: 1–9.

[31] Yeom K, Park J H. Morphological approach for autonomous and adaptive systems based on self-reconfigurable modular agents[J]. Future Generation Computer Systems, 2012, 28: 533–543.

[32] 孙熙，庄磊，刘文，焦文品，梅宏．一种可定制的自主构件运行支撑框架[J]. 软件学报，2008，19(6):1340-1348．

[33] 高俊，李长云，文志华，饶居华. 面向网构软件的构件自适应机制研究[J]. 计算机应用研究, 2009, 26(5): 1749-1753.

[34] Palsberg J, Xiao C, Lieberherr K. Efficient implementation of adaptive software[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1995, 17(2): 264-292.

[35] Salvaneschi G, Ghezzi C, Pradella M. An analysis of language-level support for self-adaptive software[J]. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2013, 8(2): 7.

[36] Cheng B H C, De Lemos R, Giese H, *et al*. Software engineering for self-adaptive systems: A research roadmap[M]. Software engineering for self-adaptive systems. Springer Berlin

Heidelberg, 2009: 1-26.

[37] De Lemos R, Giese H, Müller H A, *et al*. Software engineering for self-adaptive systems: A second research roadmap[M]. Software Engineering for Self-Adaptive Systems II. Springer Berlin Heidelberg, 2013: 1-32.

[38] Macías-Escrivá F D, Haber R, del Toro R, *et al*. Self-adaptive systems: A survey of current approaches, research challenges and applications[J]. Expert Systems with Applications, 2013, 40(18): 7267-7279.

[39] Weyns D, Schmerl B, Grassi V, *et al*. On patterns for decentralized control in self-adaptive systems[M]. Software Engineering for Self-Adaptive Systems II. Springer Berlin Heidelberg, 2013: 76-107.

[40] Kokar M M, Baclawski K, Eracar Y A. Control theory-based foundations of self-controlling software[J]. Intelligent Systems and their Applications, IEEE, 1999, 14(3): 37-45.

[41] Karsai G, Ledeczi A, Sztipanovits J, *et al*. An approach to self-adaptive software based on supervisory control[M]. Self-adaptive software: applications. Springer Berlin Heidelberg, 2003: 24-38.

[42] Kramer J, Magee J. Analysing dynamic change in software architectures: a case study[C]. Proceedings Fourth International Conference on Configurable Distributed Systems, 1998. IEEE, 1998: 91-100.

[43] Zhang J, Cheng B H C. Model-based development of dynamically adaptive software[C]. Proceedings of the 28th international conference on Software engineering. ACM, 2006: 371-380.

[44] Lee I, Kannan S, Kim M, *et al*. Runtime assurance based on formal specifications[J]. Departmental Papers (CIS), 1999: 294.

[45] Meredith P O N. Efficient, expressive, and effective runtime verification:[博士论文][D]. Urbana: University of Illinois, 2012.

[46] Iftikhar MU, Weyns D. A case study on formal verification of self-adaptive behaviors in a decentralized system[J]. arXiv preprint arXiv:1208.4635, 2012.

[47] Rouff C, Buskens R, Pullum L, *et al*. The *AdaptiV* approach to verification of adaptive systems[C]. Proceedings of the Fifth International C* Conference on Computer Science and

Software Engineering. ACM, 2012: 118-122.

[48] Goldsby H J, Cheng B H C, Zhang J. AMOEBA-RT: Run-time verification of adaptive software[M]. Models in Software Engineering. Springer Berlin Heidelberg, 2008: 212-224.

[49] Wang CW, Cavarra A. Checking model consistency using data-flow testing[C]. Asia-Pacific Software Engineering Conference, 2009. IEEE, 2009: 414-421.

[50] Calinescu R, Ghezzi C, Kwiatkowska M, *et al*. Self-adaptive software needs quantitative verification at runtime[J]. Communications of the ACM, 2012, 55(9): 69-77.

[51] Bencomo N. On the use of software models during software execution[C]. Proceedings of the 2009 ICSE workshop on modeling in software engineering. IEEE Computer Society, 2009: 62-67.

[52] Petri C A. Kommunikation mit automaten[D]. Technical report, Doctoral Thesis, University of Bonn, 1962. (Available in English as: Communication with automata, Technical Report RADC-TR-65-377, Rome Air Development Center, Griffiss NY, 1966).

[53] Murata T. Petri nets: Properties, analysis and applications[J]. Proceedings of the IEEE, 1989, 77(4): 541-580.

[54] Henzinger T A. The theory of hybrid automata[C]. Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science. IEEE Computer Society, 1996: 278.

[55] Haykin S. Neural networks: A Comprehensive Foundation, 2nd Edition. London: Prentice Hall PTR, 1999.

[56] Frehse G. PHAVer: algorithmic verification of hybrid systems past HyTech[J]. International Journal on Software Tools for Technology Transfer, 2008, 10(3): 263-279.

[57] Khalil H K, Grizzle J W. Nonlinear systems[M]. Upper Saddle River: Prentice hall, 2002.

[58] Mao X, Truman A, Yuan C. Euler-Maruyama approximations in mean-reverting stochastic volatility model under regime-switching [J]. Journal of Applied Mathematics and Stochastic Analysis, 2006, 2006: Article ID 80967, 1-20.

[59] Yuan C, Mao X. Convergence of the Euler–Maruyama method for stochastic differential equations with Markovian switching[J]. Mathematics and Computers in Simulation, 2004, 64(2): 223-235.

[60] Kushner H J, Dupuis P. Numerical methods for stochastic control problems in continuous time[M]. Springer, 2001.

[61] Kůrková V. Kolmogorov's theorem and multilayer neural networks[J]. Neural networks, 1992, 5(3): 501-506.

[62] Zhou M C. Modeling, analysis, simulation, scheduling, and control of semiconductor manufacturing systems: A Petri net approach[J]. IEEE Transactions on Semiconductor Manufacturing, 1998, 11(3): 333-357.

[63] Øksendal B. Stochastic differential equations[M]. Springer Berlin Heidelberg, 2003.

[64] Wu X Q. An intelligent Petri nets model based on competitive neural network[M]. Computer Supported Cooperative Work in Design I. Springer Berlin Heidelberg, 2005: 388-397.

[65] Tanaka K, Iwasaki M, Wang H O. Stable switching fuzzy control and its application to a hovercraft type vehicle[C]. The Ninth IEEE International Conference on Fuzzy Systems, 2000. IEEE, 2000, 2: 804-809.

[66] Amin S, Abate A, Prandini M, *et al*. Reachability analysis for controlled discrete time stochastic hybrid systems[M]. Hybrid Systems: Computation and Control. Springer Berlin Heidelberg, 2006: 49-63.

[67] Nguyen D, Widrow B. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights[C]. International Joint Conference on Neural Networks, 1990. IEEE, 1990: 21-26.

[68] Shatz S M, Mai K, Black C, *et al*. Design and implementation of a Petri net based toolkit for ada tasking analysis[J]. IEEE Transactions on Parallel and Distributed Systems, 1990, 1(4): 424-441.

附录 A 文中编写的部分代码和数据

A1 文 5.1.1 节中模型验证时编写的主要 PHAVer 代码

A1.1 图 5.3 中所示仿射自动机的 PHAVer 描述

```
automaton test
contr_var:  t,x,y;//t11,t12,t13,t14,t21,t22,t311,t312,t321,t322,t331,t332,t341,t342
synclabs:   a;

loc s11: while 2.62<=x & x<=3.131 & 0<=y & y<=2 & t<=30
      wait{-0.4102*y-0.1947*x<=y' & y'<=-0.008721*y-0.00414*x & x'==-0.1711*y-0.3604*x & t'==1};
      when x==2.62 sync a do {x'==x & y'==y & t'==t} goto s12;

loc s12: while 1.57<=x & x<=2.62 & 0<=y & y<=2 & t<=30
      wait{-0.8233*y-0.3908*x<=y' & y'<=-0.4102*y-0.1947*x & x'==-0.1711*y-0.3604*x & t'==1};
      when x==2.62 sync a do {x'==x & y'==y & t'==t} goto s11;
      when x==1.57 sync a do {x'==x & y'==y & t'==t} goto s13;

loc s13: while 0.52<=x & x<=1.57 & 0<=y & y<=2 & t<=30
      wait{-0.8233*y-0.3908*x<=y' & y'<=-0.4091*y-0.1942*x & x'==-0.1711*y-0.3604*x & t'==1};
      when x==1.57 sync a do {x'==x & y'==y & t'==t} goto s12;
      when x==0.52 sync a do {x'==x & y'==y & t'==t} goto s14;

loc s14: while 0.0628<=x & x<=0.52 & 0<=y & y<=2 & t<=30
      wait{-0.4091*y-0.1942*x<=y' & y'<=-0.05167*y-0.02453*x & x'==-0.1711*y-0.3604*x & t'==1};
      when x==0.52 sync a do {x'==x & y'==y & t'==t} goto s13;
      when x==0.0628 sync a do {x'==x & y'==y & t'==t} goto s21;

loc s21: while 0<= x & x <= 0.0628 & -2<=y & y<=2 & t<=30
          wait { 0<=y' & y'<=0.5*x & x'==-0.8163*y-0.6667*x & t'==1};
   when x==0.0628 sync a do {x'==x & y'==y & t'==t} goto s14;
   when x==0  sync a do {x'==x & y'==y & t'==t} goto s22;
loc s22: while -0.0628 <= x & x <=0 & -2<=y & y<=2 & t<=30
          wait {0.5*x<=y' & y'<=0 & x'==-0.8163*y-0.6667*x & t'==1};
   when x==0 sync a do {x'==x & y'==y & t'==t} goto s21;
   when x==-0.0628 & 0<=y sync a do {x'==x & y'==y & t'==t} goto s311;
   when x==-0.0628 & y<=0 sync a do {x'==x & y'==y & t'==t} goto s312;

loc s311: while -0.52<=x & x<=-0.0628 & 0<=y & y<=2 & t<=30
      wait{-0.7798*y-0.2005*x<=y' & y'<=-0.4224*y-0.3702*x & x'==-0.1711*y-0.3604*x& t'==1};
    when -0.0628==x sync a do {x'==x & y'==y & t'==t} goto s22;
    when -0.52<=x & x<=-0.0628 & y<=0 sync a do {x'==x & y'==y & t'==t} goto s312;
    when x<=-0.52 sync a do {x'==x & y'==y & t'==t} goto s321;

loc s312: while -0.52<=x & x<=-0.0628 & -2<=y & y<=0 & t<=30
      wait{-0.4224*y-0.2005*x<=y' & y'<=-0.7798*y-0.3702*x & x'==-0.1711*y-0.3604*x& t'==1};
    when -0.0628==x sync a do {x'==x & y'==y & t'==t} goto s22;
    when -0.52<=x & x<=-0.0628 & 0<=y sync a do {x'==x & y'==y & t'==t} goto s311;
    when x==-0.52 sync a do {x'==x & y'==y & t'==t} goto s322;

loc s321: while -1.57<=x & x<=-0.52 & 0<=y & y<=2 & t<=30
      wait{-0.4224*y-0.0039*x<=y' & y'<=-0.0082*y-0.2005*x & x'==-0.1711*y-0.3604*x& t'==1};
    when -1.57<=x & x<=-0.52 & y<=0 sync a do {x'==x & y'==y & t'==t} goto s322;
    when -0.52==x sync a do {x'==x & y'==y & t'==t} goto s311;
    when x==-1.57 sync a do {x'==x & y'==y & t'==t} goto s331;

loc s322: while -1.57<=x & x<=-0.52 & -2<=y & y<=0 & t<=30
      wait{-0.0082*y-0.0039*x<=y' & y'<=-0.4224*y-0.2005*x & x'==-0.1711*y-0.3604*x& t'==1};
    when -1.57<=x & x<=-0.52 & 0<=y sync a do {x'==x & y'==y & t'==t} goto s321;
    when -0.52==x  sync a do {x'==x & y'==y & t'==t} goto s312;
    when x==-1.57  sync a do {x'==x & y'==y & t'==t} goto s332;

loc s331: while -2.62<=x & x<=-1.57 & 0<=y & y<=2 & t<=30
      wait{-0.4213*y-0.0039*x<=y' & y'<=-0.0082*y-0.2*x & x'==-0.1711*y-0.3604*x& t'==1};
    when -2.62<=x & x<=-1.57 & y<=0 sync a do {x'==x & y'==y & t'==t} goto s332;
    when x==-2.62 sync a do {x'==x & y'==y & t'==t} goto s341;
    when -1.57==x sync a do {x'==x & y'==y & t'==t} goto s321;

loc s332: while -2.62<=x & x<=-1.57 & -2<=y & y<=0 & t<=30
      wait{-0.0082*y-0.0039*x<=y' & y'<=-0.4213*y-0.2*x & x'==-0.1711*y-0.3604*x& t'==1};
    when -2.62<=x & x<=-1.57 & 0<=y sync a do {x'==x & y'==y & t'==t} goto s331;
    when x==-2.62 sync a do {x'==x & y'==y & t'==t} goto s342;
    when -1.57==x sync a do {x'==x & y'==y & t'==t} goto s322;

loc s341: while -3.131<=x & x<=-2.62 & 0<=y & y<=2 & t<=30
      wait{-0.8228*y-0.2*x<=y' & y'<=-0.4213*y-0.3906*x & x'==-0.1711*y-0.3604*x& t'==1};
    when -3.131<=x & x<=-2.62 & y<=0 sync a do {x'==x & y'==y & t'==t} goto s342;
    when -2.62==x sync a do {x'==x & y'==y & t'==t} goto s331;

loc s342: while -3.131<=x & x<=-2.62 & -2<=y & y<=0 & t<=30
      wait{-0.4213*y-0.2*x<=y' & y'<=-0.8228*y-0.3906*x & x'==-0.1711*y-0.3604*x& t'==1};
    when -3.131<=x & x<=-2.62 & 0<=y sync a do {x'==x & y'==y & t'==t} goto s341;
    when -2.62==x sync a do {x'==x & y'==y & t'==t} goto s332;

initially:  s13 & x==2 &  y==1 & t==0;
end
```

A1.2 计算图 5.3 中仿射自动机的可达状态的 PHAVer 代码

```
REACH_USE_CONVEX_HULL = true;
test.set_partition_constraints((x,0.01),(y,0.01),a);

// ------------------------------------------------------
// Optional analysis parameters to speed up the computations
// ------------------------------------------------------
REFINE_DERIVATIVE_METHOD = 2;
SEARCH_METHOD = 6;
REFINE_CHECK_TIME_RELEVANCE_DURING = true;
REFINE_CHECK_TIME_RELEVANCE_FINAL = true;
CHEAP_CONTAIN_RETURN_OTHERS=false;

reg=test.reachable;
//reg.remove(t);
reg.project_to(x,y);
reg.print("reach1",2);
reg.print("reach2",0);
```

由 A1.1 得到一个名为“test”的自动机，该程序计算 test 的可达状态，其中 reg.print(“reach1”,2)表示在文件 reach1 中输出可达状态，这里的可达状态用每个多边形的顶点坐标表示；reg.print(“reach2”,0)表示在文件 reach2 中输出可达状态，这里的可达状态用线性不等式表示。

A1.3 由上述代码获得的部分可达图的表示

```
0.972121321131475091270602 0.96875
0.979374999999999995559108 0.96875
0.979374999999999995559108 0.976951646421583941304334
0.972121321131475091270602 0.96875

0.979374999999999995559108 0.976951646421583941304334
0.979374999999999995559108 0.96875
0.986420774325638904933555 0.96875
1 1
0.979374999999999995559108 0.976951646421583941304334

0.979374999999999995559108 0.96875
0.979374999999999995559108 0.952717822047153717335277
0.986420774325638904933555 0.96875
0.979374999999999995559108 0.96875
```

reach1 中的可达多边形表示

```
reg = test.{s13@14-@15-@16+@17+@18+@19+@20+@21-@35+@36+@37+ &
32*y >= 31 & -1600*x >= -1567 & 283234189167616*x - 250497247406592*y >= 32668785738089,
s13@14-@15-@16+@17+@18+@19+@20+@21+@22+@23-@24+ &
1600*x >= 1567 & 9486119*x - 8488728*y >= 997391 & 32*y >= 31 & -4824159*x + 2096267*y >= -2727892,
s13@14-@15-@16+@17+@18+@19+@20+@21+@22+@23-@24- &
1600*x >= 1567 & -728833401414192*x + 320305555642896*y >= -408640401148321 & -32*y >= -31,
```

reach2 中的可达多边形表示

A2 文 5.1.2 节中模型验证时编写的主要 PRISM 代码

A2.1 Markov 链切换控制的 PRISM 描述

```
ctmc
//Patameter
//T=70F and T=79F are represented by tp=1 and tp=19
//The init temperature is 73F;
global tp : [1..20] init 7;

module control
s : [0..1] init 1;  // 1--switch, 0--no switch
direction: [1 ..2 ] init 1;// 1-decrease ,   2-increase
[o2t] direction=1 -> (direction'=2) & (s'=1);//jump from s1 to s2
[t2o] direction=2 -> (direction'=1) & (s'=1); //jump from s2 to s1
// init-jump to decrease
[d1] tp=1 & direction=1 & p1!=0 & s=1-> (s'=0);
[d2] tp=2 & direction=1 & p1!=0 & s=1-> (s'=0);
[d3] tp=3 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d4] tp=4 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d5] tp=5 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d6] tp=6 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d7] tp=7 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d8] tp=8 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d9] tp=9 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d10] tp=10 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d11] tp=11 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d12] tp=12 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d13] tp=13 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d14] tp=14 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d15] tp=15 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d16] tp=16 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d17] tp=17 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d18] tp=18 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d19] tp=19 & direction=1 & p1!=0 & s=1 -> (s'=0);
[d20] tp=20 & direction=1 & p1!=0 & s=1 -> (s'=0);
// init-jump to increase
[i1] tp=1 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i2] tp=2 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i3] tp=3 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i4] tp=4 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i5] tp=5 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i6] tp=6 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i7] tp=7 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i8] tp=8 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i9] tp=9 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i10] tp=10 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i11] tp=11 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i12] tp=12 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i13] tp=13 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i14] tp=14 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i15] tp=15 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i16] tp=16 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i17] tp=17 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i18] tp=18 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i19] tp=19 & direction=2 & p2!=0 & s=1 -> (s'=0);
[i20] tp=20 & direction=2 & p2!=0 & s=1 -> (s'=0);
endmodule
```

A2.2 off 子系统的 PRISM 描述

```
module s1       //decrease  , direction=1
p1 : [0..22] init 22;   // 0-jump, 22-idle  state
[d1] true-> (p1'=1);
[d2] true-> (p1'=2);
[d3] true-> (p1'=3);
[d4] true-> (p1'=4);
[d5] true-> (p1'=5);
[d6] true-> (p1'=6);
[d7] true-> (p1'=7);
[d8] true-> (p1'=8);
[d9] true-> (p1'=9);
[d10] true-> (p1'=10);
[d11] true-> (p1'=11);
[d12] true-> (p1'=12);
[d13] true-> (p1'=13);
[d14] true-> (p1'=14);
[d15] true-> (p1'=15);
[d16] true-> (p1'=16);
[d17] true-> (p1'=17);
[d18] true-> (p1'=18);
[d19] true-> (p1'=19);
[d20] true-> (p1'=20);

[o2t] p1=0 -> (p1'=22);   //jump

[] p1=1 -> (p1'=0) & (tp'=1);
[] p1=2 -> 0.025: (p1'=3)&(tp'=3) + 0.175 : (p1'=1)&(tp'=1) + 0.8 : (p1'=0);
[] p1=3 -> 0.0248 :(p1'=4)&(tp'=4) + 0.1752 : (p1'=2)&(tp'=2) + 0.8 : (p1'=0);
[] p1=4 -> 0.0247 : (p1'=5)&(tp'=5) + 0.1753 : (p1'=3)&(tp'=3) + 0.8 : (p1'=0);
[] p1=5 -> 0.0245 : (p1'=6)&(tp'=6) + 0.1755 : (p1'=4)&(tp'=4) + 0.8 : (p1'=0);
[] p1=6 -> 0.122 : (p1'=7)&(tp'=7) + 0.878 : (p1'=5)&(tp'=5);
[] p1=7 -> 0.1212 : (p1'=8)&(tp'=8) + 0.8788 : (p1'=6)&(tp'=6);
[] p1=8 -> 0.1205 : (p1'=9)&(tp'=9) + 0.8795 : (p1'=7)&(tp'=7);
[] p1=9 -> 0.1198 : (p1'=10)&(tp'=10) + 0.8802 : (p1'=8)&(tp'=8);
[] p1=10 -> 0.119 : (p1'=11)&(tp'=11) + 0.881 : (p1'=9)&(tp'=9);
[] p1=11 -> 0.1183 : (p1'=12)&(tp'=12) + 0.8817 : (p1'=10)&(tp'=10);
[] p1=12 -> 0.1176 : (p1'=13)&(tp'=13) + 0.8824 : (p1'=11)&(tp'=11);
[] p1=13 -> 0.117 : (p1'=14)&(tp'=14) + 0.883 :   (p1'=12)&(tp'=12);
[] p1=14 -> 0.1163 : (p1'=15)&(tp'=15) + 0.8837 : (p1'=13)&(tp'=13);
[] p1=15 -> 0.1156 : (p1'=16)&(tp'=16) + 0.8844 : (p1'=14)&(tp'=14);
[] p1=16 -> 0.1149 : (p1'=17)&(tp'=17) + 0.8851 : (p1'=15)&(tp'=15);
[] p1=17 -> 0.1143 : (p1'=18)&(tp'=18) + 0.8857 : (p1'=16)&(tp'=16);
[] p1=18 -> 0.1136 : (p1'=19)&(tp'=19) + 0.8864 : (p1'=17)&(tp'=17);
[] p1=19 -> 0.113 : (p1'=20)&(tp'=20) + 0.887 :   (p1'=18)&(tp'=18);
[] p1=20 -> (p1'=19) & (tp'=19);
endmodule
```

A2.3 on 模式的 PRISM 描述

```
module s2    //increase  ,direction=2
p2 : [0..22] init 22;

[i1] true->(p2'=1);
[i2] true->(p2'=2);
[i3] true->(p2'=3);
[i4] true->(p2'=4);
[i5] true->(p2'=5);
[i6] true->(p2'=6);
[i7] true->(p2'=7);
[i8] true->(p2'=8);
[i9] true->(p2'=9);
[i10] true->(p2'=10);
[i11] true->(p2'=11);
[i12] true->(p2'=12);
[i13] true->(p2'=13);
[i14] true->(p2'=14);
[i15] true->(p2'=15);
[i16] true->(p2'=16);
[i17] true->(p2'=17);
[i18] true->(p2'=18);
[i19] true->(p2'=19);
[i20] true->(p2'=20);

[t2o] p2=0 -> (p2'=22);  //jump

[] p2=1 -> (p2'=2)&(tp'=2);
[] p2=2 -> 0.8333: (p2'=3)&(tp'=3) + 0.1667 : (p2'=1)&(tp'=1);
[] p2=3 -> 0.8319 :(p2'=4)&(tp'=4) + 0.1681 : (p2'=2)&(tp'=2);
[] p2=4 -> 0.8305 : (p2'=5)&(tp'=5) + 0.1695 : (p2'=3)&(tp'=3);
[] p2=5 -> 0.8291 : (p2'=6)&(tp'=6) + 0.1709 : (p2'=4)&(tp'=4);
[] p2=6 -> 0.8276 : (p2'=7)&(tp'=7) + 0.1724 : (p2'=5)&(tp'=5);
[] p2=7 -> 0.8261 : (p2'=8)&(tp'=8) + 0.1739 : (p2'=6)&(tp'=6);
[] p2=8 -> 0.8246 : (p2'=9) &(tp'=9)+ 0.1754 : (p2'=7)&(tp'=7);
[] p2=9 -> 0.8230 : (p2'=10)&(tp'=10) + 0.1770 : (p2'=8)&(tp'=8);
[] p2=10 -> 0.8214 : (p2'=11)&(tp'=11)+ 0.1786 : (p2'=9)&(tp'=9);
[] p2=11 -> 0.8198 : (p2'=12)&(tp'=12) + 0.1802 : (p2'=10)&(tp'=10);
[] p2=12 -> 0.8182 : (p2'=13)&(tp'=13) + 0.1818 : (p2'=11)&(tp'=11);
[] p2=13 -> 0.8165 : (p2'=14)&(tp'=14) + 0.1835 : (p2'=12)&(tp'=12);
[] p2=14 -> 0.8148 : (p2'=15)&(tp'=15) + 0.1852 : (p2'=13)&(tp'=13);
[] p2=15 -> 0.8131 : (p2'=16)&(tp'=16) + 0.1869 : (p2'=14)&(tp'=14);
[] p2=16 -> 0.8113 : (p2'=17)&(tp'=17) + 0.1887 : (p2'=15)&(tp'=15);
[] p2=17 -> 0.1619 : (p2'=18)&(tp'=18) + 0.0381 : (p2'=16)&(tp'=16) + 0.8 : (p2'=0);
[] p2=18 -> 0.1615 : (p2'=19)&(tp'=19) + 0.0385 : (p2'=17)&(tp'=17) + 0.8 : (p2'=0);
[] p2=19 -> 0.1617 : (p2'=20)&(tp'=20) + 0.0388 : (p2'=18)&(tp'=18) + 0.8 : (p2'=0);
[] p2=20 -> (p2'=0) & (tp'=20);
endmodule
```

A2.4 几个验证形式

命令 1：P<=pp[tp=x]，该命令用来验证系统处于状态 x 的概率是否小于等于 pp。

命令 2：P=? [F[t1,t2]tp = x]，该命令计算系统在[t1,t2]时间内处于状态 x 的概率。

A3 文 5.2 节中自适应 Petri 网中的神经网络的训练数据和结果

A3.1 神经网络 NN_1 和 NN_2 的训练样本数据

本节附录列出了 72 组预处理后的训练样本数据，见表 A1 和表 A2。

表 A1:神经网络 NN1 的训练样本数据

序号	输入								输出
	变量 1	变量 2	变量 3	变量 4	变量 5	变量 6	变量 7	变量 8	
1	0	0	0	0	0	0	0	0	0.5
2	0	0.01	0.012	0	0.1	0.08	0.01	0	0.5
3	0.01	0.11	0.01	0.05	0.01	0.03	0.06	0.07	0.5
4	0.121	0.101	0.211	0.113	0.142	0.213	0.167	0.152	0.5
5	0.231	0.421	0.234	0	0.157	0.201	0.01	0.127	0.5
6	0.236	0.253	0.15	0.233	0.056	0.153	0	0.293	0.5
7	0.231	0	0.178	0.203	0.111	0.168	0.217	0.222	0.5
8	0.111	0.2	0.27	0.29	0.25	0.1	0.27	0.1	0.5
9	0.128	0.261	0.1	0.2	0.121	0.287	0.3	0	0.5
10	0	0.1	0.08	0.132	0.241	0.032	0.101	0.05	0.5
11	0.1	0.08	0.287	0.087	0.2	0.14	0.257	0.123	0.5
12	0.223	0.245	0.251	0.234	0.287	0.212	0.231	0.2	0.5
13	0.087	0.141	0.241	0.145	0.1	0.121	0.152	0.143	0.5
14	0.78	0.124	0.154	0.021	0.15	0.157	0.1	0.04	0.5
15	0.13	0.6	0.145	0.124	0.25	0.21	0.047	0.1	0.5
16	0.17	0.21	0.64	0.14	0.12	0.25	0.21	0.12	0.5
17	0.21	0.174	0.128	0.598	0.147	0.213	0.142	0.261	0.5
18	0.102	0.201	0.14	0.175	0.624	0.131	0.182	0.24	0.5
19	0.3	0.3	0.321	0.301	0.341	0.321	0.333	0.312	1.5
20	0.312	0.421	0.233	0.223	0.337	0.368	0.237	0.512	1.5
21	0.356	0.547	0.25	0.433	0.428	0.533	0.417	0.2470	1.5
22	0.452	0.323	0.35	0	0.88	0.578	0.34	0.4730	1.5
23	0.272	0.327	0.325	0.443	0.356	0.34	0.457	0.2600	1.5
24	0.176	0.473	0.55	0.468	0.192	0.08	0.583	0.3930	1.5
25	0.192	0.373	0.45	0	0.312	0.04	0.767	0.5670	1.5
26	0.504	0.587	0.65	0.567	0.316	0.66	0.317	0.1270	1.5
27	0.345	0.421	0.412	0.321	0.289	0.364	0.458	0.587	1.5
28	0.123	0.567	0.487	0.654	0.587	0.349	0.4	0.5	1.5
29	0.489	0.214	0.124	0.547	0.457	0.687	0.5	0.47	1.5
30	0.345	0.389	0.401	0.01	0.374	0.421	0.5	0.378	1.5
31	0.412	0.451	0.386	0.451	0.12	0.389	0.421	0.341	1.5
32	0.33	0.4	0.47	0.5	0.47	0.145	0.44	0.412	1.5
33	0.43	0.464	0.389	0.378	0.45	0.461	0.12	0.45	1.5
34	0.389	0.34	0.43	0.354	0.298	0.43	0.471	0.201	1.5
35	0.287	0.371	0.298	0.275	0.314	0.378	0.328	0.289	1.5
36	0.299	0.356	0.461	0.381	0.356	0.312	0.378	0.345	1.5

37	0.652	0.523	0.95	0.533	0.788	0.78	0.234	0.473	2.5
38	0.64	0.687	0.45	0.7	0.652	0.42	0.567	0.58	2.5
39	0.547	0	0.821	0.612	0.457	0.368	0.571	0.123	2.5
40	0.308	0.727	0.35	0.563	0.456	0.293	0.233	0.68	2.5
41	0.547	0.654	0.475	0.874	0	0.471	0.354	0.667	2.5
42	0.512	0.8	0.75	0.45	0.736	0.527	0.167	0.387	2.5
43	0.468	0.927	0.45	0.233	0.46	0.713	0.817	0.347	2.5
44	1	0.578	0.674	0.687	0.669	0.725	0.731	0.725	2.5
45	0.67	0.58	0.54	0.57	0.55	0.54	0.687	0.544	2.5
46	0.64	0.57	0.61	0.72	0.53	0.612	0.552	0.678	2.5
47	0.54	1	0.21	0.651	0.587	0.578	0.529	0.617	2.5
48	0.57	0.61	0.987	0.51	0.616	0.645	0.589	0.571	2.5
49	0.612	0.541	0.71	0.99	0.52	0.74	0.75	0.64	2.5
50	0.71	0.624	0.654	0.75	1	0.65	0.63	0.655	2.5
51	0.621	0.666	0.671	0.634	0.671	1	0.678	0.631	2.5
52	0.578	0.651	0.589	0.547	0.567	0.681	1	0.5	2.5
53	0.551	0.621	0.544	0.671	0.71	0.697	0.654	1	2.5
54	0.75	0.324	0.741	0.699	0.75	0.711	0.6	0.657	2.5
55	0.12	0.814	0.881	0.911	0.789	0.854	0.874	0.901	3.5
56	0.814	0.201	0.812	0.854	0.841	0.789	0.891	0.831	3.5
57	0.821	0.831	0.121	0.9	0.861	0.811	0.845	0.851	3.5
58	0.789	0.8	0.831	0.811	0.11	0.823	0.789	0.8	3.5
59	0.85	0.98	0.841	0.96	0.921	0.123	0.98	0.93	3.5
60	0.83	0.961	0.912	0.8	0.947	0.912	0.124	0.31	3.5
61	0.87	0.89	0.944	0.987	0.912	0.874	0.971	0.854	3.5
62	0.91	0.91	0.931	0.87	0.79	0.867	0.801	0.93	3.5
63	0.95	0.931	0.889	0.79	0.967	0.931	0.821	0.958	3.5
64	0.712	0.68	0.78	0.867	0.896	0.827	0.717	0.7200	3.5
65	0.76	0.781	0.821	0.841	0.811	0.774	0.951	0.765	3.5
66	0.943	0.906	0.828	0.932	0.673	0.728	0.964	0.910	3.5
67	0.981	0.845	0.721	0.91	0.568	0.671	0.924	0.874	3.5
68	0.872	0.957	0.887	0.754	0.891	0.971	1	0.869	3.5
69	0.987	0.834	0.8474	0.824	0.85	0.92	0.874	0.82	3.5
70	0.971	0.945	0.857	0.914	0.991	0.947	0.943	0.897	3.5
71	0.574	0.897	0.987	0.527	0.811	0.7	0.621	0.679	3.5
72	1	1	1	1	1	1	1	1	3.5

表 A2 神经网络 NN2 的训练样本数据

序号	输入				输出
	变量 9	变量 10	变量 11	变量 12	
1	0	0	0	0	0.5
2	0.101	0.087	0.123	1	0.5
3	0.211	1	0.121	0.014	0.5
4	0.121	0.241	1	0.178	0.5

5	0.157	0.012	0.214	0.314	0.5
6	0.211	0.421	0.312	0.151	0.5
7	0.234	0.345	0.265	0.421	0.0
8	0.345	0.114	0.487	0.387	0.5
9	0.321	0.287	0.112	0.404	0.5
10	0.5	0.458	0.154	0.211	0.5
11	0.498	0.214	0.289	0.444	0.5
12	0.41	0.26	0.41	0.23	0.5
13	1	0.024	0.324	0.112	0.5
14	0.211	0.024	0.768	0.001	0.5
15	0.245	0.321	0.217	0.211	0.5
16	0.456	0.324	0.121	0.125	0.5
17	0.5	0.437	0.478	0.432	1.5
18	0.654	0.741	0.811	0.789	1.5
19	0.75	0.687	0.91	0.88	1.5
20	0.547	0.634	0.687	0.911	1.5
21	0.784	0.754	0.654	0.824	1.5
22	0.691	0.912	0.753	0.723	1.5
23	0.723	0.672	0.666	0.589	1.5
24	0.891	0.75	0.828	0.742	1.5
25	0.851	0.571	0.582	0.643	1.5
26	0.831	0.843	0.94	0.761	1.5
27	0.8	0.89	0.71	0.67	1.5
28	0.841	0.951	0.811	0	1.5
29	0.64	0.56	0.82	0.84	1.5
30	0.85	0.73	0.71	0.63	1.5
31	1	1	1	1	1.5
32	0.9	0.817	0	0.957	1.5
33	0	0.798	1	0.9	1.5
34	0.978	0	0.79	0.867	1.5
35	0.12	0.201	0.101	0.005	0.5
36	0.012	0.231	0.241	0.01	0.5
37	0.321	0.124	0.781	0.012	0.5
38	0.412	0.321	0.04	0.12	0.5
39	0.214	0.123	0.245	0.12	0.5
40	0.354	0.02	0.147	0.258	0.5
41	0.123	0.169	0.247	0.358	0.5
42	0.2194	0.138	0.3756	0.4204	0.5
43	0.1908	0.3399	0.1275	0.1271	0.5
44	0.3828	0.3275	0.253	0.4071	0.5
45	0.3976	0.0813	0.5495	0.1218	0.5
46	0.0934	0.0595	0.4455	0.4646	0.5
47	0.2449	0.2492	0.4796	0.175	0.5
48	0.2228	0.4799	0.2736	0.0983	0.5
49	0.3232	0.1702	0.0693	0.1255	0.5

50	0.3547	0.2926	0.0746	0.308	0.5
51	0.3773	0.1119	0.1288	0.2366	0.5
52	0.3517	0.7943	0.646	0.5132	1.5
53	0.8308	0.3112	0.7749	0.4018	1.5
54	0.5853	0.5285	0.8173	0.76	1.5
55	0.5497	0.656	0.8687	0.2399	1.5
56	0.9172	0.602	0.844	0.1233	1.5
57	0.2858	0.63	0.998	0.839	1.5
58	0.7572	0.6541	0.599	0.624	1.5
59	0.7537	0.6892	0.8001	0.4173	1.5
60	0.3804	0.7482	0.4314	0.797	1.5
61	0.5678	0.4505	0.9106	0.9027	1.5
62	0.0759	0.838	0.818	0.9448	1.5
63	0.054	0.749	0.638	0.909	1.5
64	0.5308	0.9133	0.1455	0.6893	1.5
65	0.7792	0.524	0.61	0.3377	1.5
66	0.934	0.258	0.8693	0.9001	1.5
67	0.7299	0.5383	0.5797	0.692	1.5
68	0.5688	0.9961	0.5499	0.1112	1.5
69	0.694	0.782	0.145	0.7803	1.5
70	0.0119	0.4427	0.853	0.897	1.5
71	0.6371	0.1067	0.6221	0.8417	1.5
72	0.1622	0.9619	0.7351	0.911	1.5

A3.2 神经网络的训练结果

本节附录列出了两个神经网络的训练结果，包括权重和偏置。

A3.2.1 NN_1 输入层到隐藏层的权重和偏置

表 A3 给出了输入层到隐藏层的权重，其中行表示隐藏层神经元，列表示输入层神经元。

表 A3 输入层到隐藏层的权重

-0.17868	-0.36488	1.131685	-0.96615	0.171362	-0.33229	-1.00185	-2.03299
-1.36656	0.077257	0.238678	1.998846	-0.90496	0.630474	-0.13365	-0.44524
-0.39446	0.843985	0.592927	0.442009	0.960547	-0.94767	0.799905	0.289066
-0.03414	-0.67751	-0.36712	0.887104	-0.06042	1.082169	-0.63021	0.672486
-0.72641	-0.53278	0.783551	-0.29339	-0.96667	0.999618	-0.67666	0.528579
-0.51915	0.315905	-1.76877	-1.49476	0.215217	0.646587	0.160749	-0.41179
0.776663	0.714143	0.496529	-0.15067	0.225104	0.287401	0.447989	0.541478
0.005129	1.149636	-0.62519	-0.99635	-0.33209	1.147629	-0.91055	0.582729
0.575935	0.213364	-0.09645	-0.77484	0.899626	0.501243	0.941984	-0.64967
0.319041	-0.13137	0.215646	-1.23479	0.474595	-1.19462	0.70488	0.086018

-0.811	1.218993	-1.1037	-0.24254	-0.28597	-1.79775	-0.95267	-0.84989
-1.78445	0.471038	0.623625	0.477763	-0.13094	-0.65316	0.798999	0.432939
0.485453	1.005764	-0.56898	0.528841	-0.89703	0.162235	-2.08396	1.026748
-0.48161	0.683205	0.449839	-0.64961	-0.16127	-1.11275	0.596016	-1.15246
1.135126	-0.37495	0.02203	-0.80207	-0.23163	0.889539	0.511558	1.482287
0.258361	-1.63496	-0.51115	-1.0107	-1.13394	-0.85925	0.389263	-0.41749
-0.80413	-0.73368	-0.76502	-1.01019	-0.28286	-0.61831	-0.85919	-1.28113

隐藏层神经元的偏置为：[2.793265888, 1.703982665, 1.684253602, 1.147455352,1.011358714, 1.427702685, -0.641248031,-0.323311271, 0.919984763, 0.584885813, 0.169100223, -0.510525302, 1.56400955, -1.553397752, 2.309249869, 3.045680335, -3.136768503]

A3.2.2 NN_1 **隐藏层到输出成的权重和偏置**

权重为：[-1.703046874, 1.463538027, 0.495903649,0.464737225, 0.341809769, -1.694518523, -0.168343701, 0.837934083, 0.83374859, 0.860848735, 1.572849081, -0.677327311, -1.571211876, -0.535175702, 1.693350763, -1.796123841, -1.483905882],

偏置为：0.232729244。

A3.2.3 NN_2 **输入层到隐藏层的权重和偏置**

表 A4 NN_2 **神经元的输入层到隐藏层的权重和隐藏层神经元的偏置**

输入层 / 隐藏层	神经元 1	神经元 2	神经元 3	神经元 4	偏置
神经元 1	-2.75331	0.788449	3.467816	-2.22097	4.705856
神经元 2	-3.71889	-2.67885	-3.534	-4.46651	1.809408
神经元 3	4.453207	1.063085	2.21821	2.141043	-2.02398
神经元 4	0.514866	-2.77201	-3.42879	0.256851	-0.52492
神经元 5	-0.21119	3.285368	1.95635	-0.39289	1.626516
神经元 6	3.128407	0.473741	3.892892	1.599779	0.094327
神经元 7	0.126996	-1.45618	0.765469	4.003938	1.911132
神经元 8	1.307872	-4.83859	-2.25669	0.296076	3.515234
神经元 9	2.508939	0.379773	3.758138	0.616657	5.100065

A3.2.4. NN_2 **隐藏层到输出层的权重和偏置**

权 重 为 ： [-1.421042324, -7.490295591, 4.474324218, -1.237791552, 1.141732062, 1.883259232, 1.538479165, -4.43889514, 1.835970758]；

偏置为：2.2783

攻读硕士期间的研究成果

1 以第一作者或导师外第一作者完成的学术论文

[1] Zuohua Ding, **Yuan Zhou**, Mengchu Zhou. Modeling Self-Adaptive Software Systems with Learning Petri Nets. ICSE Companion 2014: 464-467. (EI 会议)

[2] Zuohua Ding, **Yuan Zhou**, Mengchu Zhou. Stability Analysis of Switched Fuzzy Systems via Model Checking. IEEE Transactions on Fuzzy Systems, 22(6): 1503-1514.(SCI 一档)

[3] Zuohua Ding, **Yuan Zhou**, Mengchu Zhou. A Polynomial Algorithm to Performance Analysis of Concurrent Systems via Petri Nets and Ordinary Differential Equations. IEEE Transactions on Automation Science and Engineering, 12(1): 295-308. (SCI 三档)

[4] Zuohua Ding, **Yuan Zhou**, Mingyue Jiang, Mengchu Zhou. A New Class of Petri Nets for Modeling and Property Verification of Switched Stochastic Systems. IEEE Transactions on Systems, Man and Cybernetics: Systems, DOI: 10.1109/TSMC.2014.2379654. (SCI 二档)

[5] **周远**, 丁佐华. 能用程序不变量计算软件可靠性吗?. 软件学报, 录用. (EI 期刊)

[6] Zuohua Ding, **Yuan Zhou**, Mengchu Zhou. Modeling Self-Adaptive Software Systems with Learning Petri Nets. IEEE Transactions on Systems, Man and Cybernetics: Systems, 审稿中. (SCI 二档)

[7] **周远**, 徐映红, 徐定华. 结合粒子群算法的一类双层纺织材料厚度设计反问题. 纺织学报, 2013, 34(6): 40-45. (核心期刊)

2 参与完成的学术论文

[8] 杨晓燕, **周远**, 丁佐华. 基于在线故障定位及自主适应提高软件可靠性. 软件学报,录用. (EI 期刊)

[9] Zuohua Ding, Ting Xu, Tiantian Ye, **Yuan Zhou**. Online Prediction and Improvement of Reliability for Service Oriented Systems. IEEE Transactions on Reliability, Minor Revision. (SCI 二档)

3 学科竞赛

[10] 第十届“华为杯”全国研究生数学建模竞赛　全国二等奖。

致 谢

时光荏苒，光阴飞逝。自此，我的毕业论文也即将完成，心中有分喜悦，又有分失落。喜悦的是两年半的努力终于汇成了这篇近百页的硕士论文，不管过程如何的艰辛，最终的结果还是好的；失落的是两年半的研究生生涯即将结束，同学、同门也将各奔东西，踏入新的人生征程。两年半的研究生生涯使我在各方面都有了较大的进步：眼界开阔了、知识增长了、能量力提高了。然而所有这些的成功和收获，都离不开父母、导师、朋友无私的帮助和关怀。纵有千言万语，最终汇成两个字：感谢！

感谢我的父母！是他们的无私奉献，是他们的艰辛劳动使我能够安心的完成学业；是他们的殷殷期盼给了我前进的功力，促使我奋发前进！

感谢我的导师丁佐华教授！是他的敦敦教导、细心指导开启了我的学术之路，使我能够畅游在学术的海洋中！是他对研究方向的精确把握使我能够朝着目的地前进而不迷失方向！他对我的无私关怀、帮助让我能够专心科研！他孜孜不倦的探索精神、严谨治学和精益求精的科研态度为我树立了良好的榜样！

感谢伍凤华同学，是她的宽容、理解使我能够做我喜欢的事情；是她的默默支持使我能够按照自己的理想去奋斗！

感谢实验室的所有老师和同门，不管何时何事，他们都能在我需要帮助的时候来帮助我，感谢你们！

最后，感谢所有抽出宝贵时间仔细审阅了本论文的专家、学者，感谢你们中肯的修改意见。