普通高等学校计算机精品课程教材

人工智能方法与应用

主编　尹朝庆

编著　尹朝庆　尹　皓　彭德巍

华中科技大学出版社

中国·武汉

前　言

人工智能是一门综合性学科，它旨在研究如何利用计算机等现代工具设计模拟人类智能行为的系统。随着计算机科学与技术的发展和计算机应用的日益普及，人工智能技术也随之广泛地渗透到各学科领域和各行各业。因此，人工智能不仅是高等学校计算机科学与技术专业和软件工程专业等计算机类专业的主干课程，也是其他信息类和管理类等学科领域提高计算机应用水平的重要基础。

人工智能及其应用领域的研究发展很快，涉及的学科领域越来越多，新的理论、方法与技术不断涌现，已形成多个研究方向。我们认为本科生的人工智能课程的教学目标应定位为：培养学生掌握人工智能的基本理论、方法和技术，具有开发应用系统的基本能力。教学内容不应过于宽泛，而应突出重点，强调应用，深入浅出地介绍人工智能的方法与实现技术。本着这一指导思想，作者根据多年教学实践的积累编撰了本书。

本书共分 7 章，各章内容如下：

第 1 章绪论　介绍人工智能的发展史、主要学派及研究应用领域。

第 2 章知识表示方法　介绍一阶谓词逻辑表示方法、产生式表示方法和框架表示方法等知识表示方法。

第 3 章搜索方法　在给出问题求解过程的两种形式表示的基础上，分别讨论状态空间和与/或图的多种盲目搜索算法和启发式搜索算法，以及典型应用实例。

第 4 章经典逻辑推理　主要讨论归结演绎推理和与/或形演绎推理，及其在定理证明和问题求解中的应用。

第 5 章专家系统　主要介绍产生式专家系统的知识库、推理机与解释器及其实现技术，讨论知识检测与求精的方法。

第 6 章不确定推理　在讨论知识不确定性的基础上，介绍可信度的不确定推理和模糊推理两种不确定推理方法。

第 7 章机器学习　主要讨论归纳学习方法(CLS 算法和 ID3 算法)、遗传算法和人工神经网络方法(BP 算法)等机器学习方法与算法，以及典型应用实例。

本书内容详实，层次清晰，详略适当，重点突出，语言严谨，例题丰富，便于教学。可作为高等学校计算机等信息类和管理类相关专业的本科生教材，也可供有关科技人员参考。

本书第 2、3、5 章由尹皓编写，第 1、4 章由彭德巍编写，第 6、7 章由尹朝庆编写。尹朝庆任主编，负责全书内容的审定与统稿。

作者十分感谢华中科技大学出版社对本书出版的指导与支持，感谢乔迪为书稿编辑和

PPT 电子课件制作所做的大量工作。

为了便于教师使用本书进行教学，作者制作了本书的 PPT 电子课件，该课件可从华中科技大学出版社网站免费下载，网址是：

www. hustp. com

由于作者水平有限，书中难免有不妥之处，请广大读者批评指正。

E-mail：yinchaoqing@yahoo. com. cn

作　者

2007 年 6 月

内 容 简 介

本书介绍人工智能的基本理论、方法及实现技术。全书共7章，分为两部分。第一部分包括第1章至第4章，主要介绍人工智能的基本概念、方法和技术，包括知识表示方法和搜索、逻辑推理等问题求解的基本方法。第二部分包括第5章至第7章，讨论产生式专家系统及其实现技术、不确定推理方法和机器学习方法及其应用实例。

本书内容详实，层次清晰，详略适当，重点突出，语言严谨，例题丰富。可作为高等学校计算机等信息类和管理类相关专业的本科生教材，也可供有关科技人员参考。

目　录

第 1 章

绪　　论

人工智能(Artificial Intelligence,AI)是一门综合性学科,它旨在研究如何利用计算机等现代工具设计模拟人类智能行为的系统。随着计算机科学与技术的发展和计算机应用的日益普及,人工智能技术也随之渗透到各学科领域和各行各业。

1.1　人工智能及其发展

人工智能是计算机科学、控制论、信息论、神经生理学、语言学等多种学科互相渗透而发展起来的一门学科。人工智能的发展虽然已走过了半个世纪的历程,但是,对人工智能至今尚无统一的定义。尽管学术界有各种各样的说法和定义,但就其本质而言,人工智能是研究、设计和应用智能机器或智能系统,来模拟人类智能活动的能力,以延伸人类智能的科学。人类智能活动的能力是指人类在认识世界和改造世界的活动中,经过脑力劳动表现出来的能力。一般地说,人类智能主要表现在以下几个方面。

(1) 感知能力

通过视觉、听觉、触觉等感官活动,接受并理解文字、图像、声音、语言等各种外界信息,认识和理解外界环境的能力。

(2) 推理与决策能力

通过人脑的生理与心理活动及有关的信息处理过程,将感性知识抽象为理性知识,并能对事物运行的规律进行分析、判断和推理,这就是提出概念、建立方法、进行演绎和归纳推理、作出决策的能力。

(3) 学习能力

通过教育、训练和学习过程,更新和丰富拥有的知识和技能,这就是学习的能力。

(4) 适应能力

对不断变化的外界环境(如干扰、刺激等)能灵活地作出正确的反应,这就是自适应能力。

不论从什么角度来研究人工智能,都是通过计算机等现代工具来实现的。计算机科学与技术的飞速发展和计算机应用的日益普及,为人工智能的研究和应用奠定了良好的物质基础。人工智能的发展使计算机更聪明、更有效,与人更接近。

1. 人工智能的起源

自古以来,人类对人工智能就有持久的、狂热的追求,并凭借当时的认识水平和技术条件,设法用机器来代替人的部分脑力劳动,用机器来延伸和扩展人类的某种智能行为。例如,公元

前 900 多年，我国就有歌舞机器人传说的记载。12 世纪末至 13 世纪初，西班牙的一位神学家和逻辑学家曾试图制造能解决各种问题的通用逻辑机。17 世纪，法国物理学家和数学家巴斯卡(B. Pascal)制成了世界第一台会演算的机械加法器并获得实际应用。随后，德国数学家和哲学家莱布尼兹(G. W. Leibniz)在这台加法器的基础上研发了可进行全部四则运算的计算器，他还提出了逻辑机的设计思想，即通过形式逻辑符号化，对思维进行推理计算。这种“万能符号”和“基于符号的推理计算”的思想是“智能机器”的萌芽，因而莱布尼兹被誉为数理逻辑的第一个奠基人。进入 20 世纪后，人工智能相继出现若干开创性的工作。1936 年，年仅 24 岁的英国数学家图灵(A. M. Turing)在他的一篇“理想计算机”的论文中，提出了著名的图灵机模型；1945 年，他进一步论述了电子数字计算机的设计思想；1950 年，他又在“计算机能思维吗？”一文中提出了机器能够思维的论述。1938 年，德国工程师苏斯(Zuse)研制成第一台累计数字计算机 Z-1。1946 年，在美国诞生了世界上第一台电子数字计算机 ENIAC。在同一时代，控制论和信息论创立，生物学家设计了脑模型等，都为人工智能学科的诞生作出了理论和实验工具的巨大贡献。

1956 年的一次历史性聚会被认为是人工智能学科诞生的标志。1956 年夏季，在美国达特莫斯(Dartmouth)大学，由当时的年青数学助教、后任斯坦福大学教授的麦卡锡(J. McCarthy)联合他的三位朋友：哈佛大学年青数学和神经学家、后任麻省理工学院教授的明斯基(M. L. Minsky)，IBM 公司信息研究中心负责人洛切斯特(N. Lochester)和贝尔实验室信息部数学研究员香农(C. E. Shannoon)共同发起，邀请 IBM 公司的莫尔(T. Moore)和塞缪尔(A. L. Samuel)、麻省理工学院的塞尔夫利奇(O. Selfridge)和索罗莫夫(R. Solomonff)、兰德(RAND)公司和卡内基工科大学的纽厄尔(A. Newell)和西蒙(H. A. Simon)等 10 名年青学者，举办了为期 2 个月的学术讨论会，讨论机器智能问题。经麦卡锡提议，在会上正式决定使用“人工智能”(Artificial Intelligence)这一术语，从而开创了人工智能作为一门独立学科的研究方向。麦卡锡因而被称为人工智能之父。从此，在美国开始形成了以人工智能为研究目标的几个研究组，如纽厄尔和西蒙的 Carnegie-RAND 协作组，明斯基和麦卡锡的 MIT 研究组，塞缪尔的 IBM 工程研究组等。

1956 年，人工智能的研究取得了两项重大突破。第一项是纽厄尔、肖(J. Shaw)和西蒙的研究组编制了一个逻辑理论程序 LT(The Logic Theory Machine)，模拟人们用数理逻辑证明定理的思想，采用分解、代入、替换等规则，证明了怀特赫德(A. N. Whitehead)和罗素(B. A. W. Russell)合著的《数学原理》第二章中的 38 条定理。1963 年，修订的程序在大机器上终于完成了该章中全部 52 条定理的证明。一般认为，这是用计算机模拟人的高级思维活动的一个重大成果，是人工智能研究的真正开端。第二项是 IBM 工程研究组的塞缪尔研制的西洋跳棋程序。这个程序可以像一个优秀棋手那样，向前看几步来下棋。尤其是它具有自学习、自组织、自适应的能力，能在下棋过程中积累经验，不断提高棋艺。它能学习棋谱，在学习了 175000 多个棋局后，可以根据棋局猜测棋谱所有推荐的走步，准确度达 48%，这是机器模拟人类学习过程的一次极有意义的探索。1959 年，这个程序战胜了设计者本人，1962 年，它又击败了美国一个州的跳棋冠军。

1957 年，纽厄尔、肖和西蒙通过心理学实验，发现了人在问题求解过程中思维的一般规律：

① 先思考出大致的解题计划；

② 根据记忆中的公理、定理和推理规则组织解题过程；

③ 进行方法和目的分析，不断修正解题计划。

基于这一规律，他们于 1960 年合作编制成功一种不依赖于具体领域的通用问题求解程序 GPS(General Problem Solver)，该程序能求解 11 种不同类型的问题。

1959 年，麻省理工学院研究组的麦卡锡发表了表处理语言 LISP。由于 LISP 可以方便地处理符号，所以很快成为人工智能程序设计的主要语言。LISP 武装了一代人工智能科学家，至今仍然是研究人工智能的重要工具。

一连串的研究成果使醉心于人工智能远景的学者们作出了过于乐观的预言。1958 年，纽厄尔和西蒙曾充满自信地认为：在 10 年内，计算机将成为世界的象棋冠军；计算机将要发现和证明重要的数学定理；计算机将能谱写具有优秀作曲家水平的乐曲；大多数心理学理论将在计算机上形成。有人甚至断言：20 世纪 80 年代将是全面实现人工智能的年代，到了 2000 年，机器的智能可以超过人的智能。

但是，事情的发展远非如此理想。塞缪尔的下棋程序在获得州冠军之后并没有获得全国冠军。自然语言的机器翻译是人工智能研究最早并取得实验性成果的研究方向之一。人们以为只要用一部双向互译字典和某些语法知识即可很快地解决自然语言之间的互译问题，实际上，由机器翻译出来的文字有时会出现十分荒谬的错误。例如，英语句子“The spirit is willing but the flesh is weak”(心有余而力不足)，翻译成俄语后再翻译成英语，竟然成为“The wine is good but the meat is spoiled”(酒是好的但肉变质了)。

自从工人智能形成一个学科之后，许多学者遵循的指导思想是：研究和总结人类思维的普遍规律，并用计算机来模拟人类的思维活动。他们认为，实现这种计算机智能模拟的关键是建立一种通用的符号逻辑运算体系。但是，由于人类的认知和思维过程是一种非常复杂的行为，故至今仍未能被完全解释；也由于现实世界的复杂性和问题的多样性，故老一辈人工智能科学家为之奋斗的通用逻辑推理体系至今也没有创造出来。其早期的代表作通用问题求解程序 GPS 的通用性受到严格的限制，只能对具有相当小的状态集和良定义的形式规则的问题有效。人工智能的早期研究只能停留在实验室里，作为研究的实验系统或演示系统，不能解决实际问题。科学家们开始对工人智能探索人类思维普遍规律的研究战略思想进行反思。

2. 人工智能的发展

20 世纪 60 年代中期以后，人工智能由追求万能、通用的一般研究转入特定的具体研究，通用的解题策略与特定领域的专业知识及实际经验结合，产生了以专家系统(Expert System，ES)为代表的基于知识的各种人工智能系统，使人工智能走向社会、走向实际应用研究。斯坦福大学当时的年轻教授费根鲍姆(E. A. Feigenbaum)重新举起了英国 16 世纪的哲学家和自然科学家培根(Francis Bacon)的旗帜“知识就是力量”，于 1965 年开创了基于知识的专家系统这一人工智能研究的新领域。与通用问题求解程序 GPS 那样的系统不同，专家系统并不试图发现强有力的和通用的问题求解方法，而是把研究范围缩小在一个特定的相对狭小的专业领域中。人类专家之所以成为专家，是因为他拥有解决自己专业领域问题的大量专门知识，包括各

种有用的经验。

在费根鲍姆的主持下，第一个专家系统课题 DENDRAL 化学分子结构分析系统于 1965 年在斯坦福大学开始研究，1968 年研制成功。该系统能根据质谱仪数据推断未知有机化合物的分子结构。它是一个启发式系统，把化学专家关于分子结构质谱测定法的知识结合到控制搜索的规则中，从而能迅速消去不可能为真的分子结构，避免了搜索空间以指数级膨胀。通过产生全部可能为真的分子结构，它甚至可以找出那些人类专家可能漏掉的结构。DENDRAL 及附属的 CONGEN 系统商品化后，每天为上百个国际用户提供化学结构的解释。这一研究成果使人们看到，在某个专门领域里，以知识为基础的计算机系统完全可能起到相当于这个领域里的人类专家的作用。

MACSYMA 系统是麻省理工学院于 1968 年开始研制的大型符号数学专家系统。该系统从应用数学家那儿获得了几百条关于一个表达式与另一等价表达式之间转换的规则，擅长于易引起组合爆炸的符号表达式的化简，能执行微分、积分、解方程、台劳级数展开、矩阵运算、向量代数等 600 多种不同的数学运算。1971 年研制成功后，由于它具有很强的与应用分析相结合的符号运算能力，很多数学和物理学的研究人员及各类工程师争相使用 MACSYMA 系统，遍及美国各地的很多用户每天都通过 ARPA 网与它联机工作达数小时。

在 DENDRAL 和 MACSYMA 的影响下，在化学、数学、医学、生物工程、地质探矿、石油勘探、气象预报、地震分析、过程控制、计算机配置、集成电路测试、电子线路分析、情报处理、法律咨询和军事决策等方面出现了一大批专家系统。著名的 MYCIN 系统就是斯坦福大学人工智能研究所于 1973 年开始研制的一个诊断和治疗细菌感染性血液病的专家咨询系统。该系统可以看成是 DENDRAL 系统的直接后继者，并且具有更广泛的影响。MYCIN 拥有的知识包括约 450 条“前提-结论”型的关于细菌性血液感染的诊疗规则，系统根据提供的数据和主动向医生询问获得的数据，运用相应的规则进行推理，最终给出对患者诊断和治疗方面的咨询性建议。经专家小组对医学专家、实习医生及 MYCIN 系统的行为进行正式测试评价，认为 MYCIN 的行为能力超过了临床医生助手，尤其在诊断和治疗败血症和脑膜炎方面有相当高的准确率。在 MYCIN 系统框架基础上建立的肺功能专家系统 PUFF 曾在旧金山太平洋医疗中心使用过相当长的一段时间。

由拉特格尔斯(Rutgers)大学于 20 世纪 70 年代中期开发的 CASNET 是一个诊断和治疗青光眼的专家咨询系统。CASNET 的知识不使用 MYCIN 的静态规则的表示方式，而是按因果关系把疾病表示为与病理生理状态相连接的网络。系统共有 150 个状态、350 个测试、50 个分类表。系统利用因果网络推断疾病类型，预测治疗效果。此外，系统还包括广泛的医学参考资料，能对疾病作出相当完善的解释。经评价，该系统接近专家水平，曾被美国和日本一些学术团体用于疾病研究。

CADUCEUS(原名 INTERNIST)系统也是 20 世纪 70 年代研制的医疗咨询专家系统，由匹兹堡大学的计算机专家和内科专家合作研制，用于内科疾病诊断。它的知识表示方式与 CASNET 类似，它的知识库是当前专家系统中最大的知识库之一，据称拥有比人类任何个人都多的内科知识。它用疾病树表达疾病分类知识，知识库中包含 500 多种内科疾病及 10 万条疾病与症状的相关关系。它还采用了一些复杂的策略来区别和组合多种疾病。因此，它能正

确地对人类专家感到棘手的复杂病案作出诊断。CADUCEUS 已用于医学教学和某些由美国国家健康研究机构赞助的医疗服务点。

HEARSAY Ⅰ和Ⅱ是卡内基-梅隆大学于 20 世纪 70 年代先后研制的两个语音理解系统。HEARSAY Ⅱ通过称为“黑板”的全局数据库来组合专家的知识。它的语音理解能力可以和一个 10 岁的孩子接近，但离专家水平甚远。这是因为语音理解是一个更为困难的研究方向。

PROSPECTOR 是一个著名的地质勘探专家系统，由斯坦福大学人工智能研究所于 1976 年开始研制。它能帮助地质学家解释地质矿藏数据，提供硬岩石矿物勘探方面的咨询，包括勘探评价、区域资源估值、钻井井位选择等。该系统的知识来自地质学家的矿床模型，用似然推理网络表示知识，网络的节点为有关探测证据和重要地质假设的各种断言，有向弧表示所连接的节点之间的推理规则，用贝叶斯(Bayes)概率推理处理不确定的数据和知识。该系统拥有 12 种矿藏知识库，共有 1100 多条规则，另有 400 种岩石和地质术语。它的性能足以与地质学家相比较，并已在实际应用中取得巨大的经济效益，例如，它发现了一个钼矿，据说该系统在这个钼矿勘探与开采中的使用价值可能超过 1 亿美元。

XCON(原名 R1)是由卡内基-梅隆大学从 1978 年开始研制的一个 VAX 计算机配置专家系统，它能根据 DEC 公司的 VAX 机的用户对每一部件的需求订单，以及关于部件结合方面的上千条规则型约束知识，形成一个功能上可以接受的 VAX 系统配置。经评价小组的正式性能测试，认为该系统具有足够的专家水平。后来，系统又进行了多次改进和发展，最后成为一个成熟的系统，在 DEC 公司担负了日常的 VAX 计算机系统配置任务，每年为 DEC 公司可节省 1500 万美元的设计费用。

专家系统的一系列研究成果展示了人工智能应用的广阔前景，社会各界对人工智能的兴趣与应用需求与日俱增。卫生界因某些医学专家系统的成功而受到极大鼓舞；军界期望它能开辟军事决策和指挥自动化的新局面；工商企业界出自对新技术的厚望，对人工智能的热情迅速高涨起来，过去对人工智能研究的态度十分保守的 IBM 公司也很快改变了态度。20 世纪 70 年代后期，随着专家系统技术的逐渐成熟和应用领域的不断开拓，人们从各种不同类型的专家系统和知识处理系统中抽取共性，人工智能又从具体系统的研究逐渐回到一般研究。围绕知识这一核心问题，人们重新对人工智能的原理和方法进行探索，并在知识获取、知识表示和知识推理等方面开始出现新的原理、方法、技术和工具。在这一背景下，在国际人工智能联合会(International Joint Conference on Artificial Intelligence，IJCAI)于 1977 年召开的 IJCAI-77 会议上，费根鲍姆提出了“知识工程”(Knowledge Engineering，KE)的概念。费根鲍姆因而被称为知识工程之父。知识工程综合了科学、技术和方法论三方面的因素，研究专门知识的获取、形式化和计算机实现，为研制以知识为基础的各类人工智能应用系统提供一般方法和基本工具。知识工程的研究有利于缩短专家系统的研制周期，促进了专家系统从单学科专用型向多学科通用型的发展，出现了一批通用程度不等、类型不同的专家系统工具，包括骨架型工具、有更大通用性的语言型工具和知识处理系统环境。应该说，知识工程和专家系统是人工智能研究中最有成就的分支领域之一，为推进人工智能研究起到了重要作用。

20 世纪 90 年代，计算机发展的主要趋势是小型化、并行化、网络化和智能化。人工智能技术逐渐与数据库技术、多媒体技术、网络技术相结合，旨在使计算机系统更聪明、更有效、与

人更接近。计算机智能化技术的主要研究方向大体上可分为三个方面：一是并行与分布处理技术，包括大规模并行计算机和机群系统的体系结构，并行操作系统，并行编译系统与并行数据库系统，分布式 Client/Server 计算模型及其处理技术，多专家协同工作与知识共享技术等；二是知识的获取、表示、更新和推理的新方法与新技术，大型知识库的组织与维护，新一代逻辑处理机制等；三是多功能的感知技术，包括对语音、文字、图形和图像等多媒体信息的获取、压缩、识别与转化，以及虚拟现实技术等。

我国从 1978 年才开始设立人工智能的研究课题，主要在定理证明、汉语自然语言理解、机器人及专家系统方面设立课题，并取得一批研究成果。在我国的“863”高技术研究开发计划中，智能计算机系统和智能机器人被列入我国高技术的重点发展主题。自该计划实施以来，我国在人工智能与智能计算机领域已取得许多可喜成绩。根据智能计算机系统的高技术研究开发计划，我国将研制可扩充的、大规模并行的、智能化的先进计算机系统，它们主要用于智能化的可视计算、事务处理、信息管理、信息分析与服务，它们将具有高速的和可视化计算的能力，具有对大量复杂信息进行存取与分析的能力，具有有效灵活的推理及机器学习的能力，具有以语音、文字、图形、图像为界面的和谐的人机交互能力，具有方便有效的智能化的软件开发能力。我们相信，21 世纪初期，这种计算机系统将成为促进我国社会经济发展的重要基础。

3. 人工智能的主要学派

随着人工智能的发展，围绕人工智能的基本理论和方法，诸如人工智能的定义、基础、核心、要素、认识过程、学科体系及人工智能与人类智能的关系等，形成几个学派。

1987 年，在美国波士顿由麻省理工学院人工智能研究所、美国国家科学基金（NSF）和美国人工智能学会（AAAI）联合主办了人工智能基础国际研讨会。许多在人工智能发展历史上作过重要贡献的各种学派的人工智能科学家应邀出席会议，并在会上报告了自己的论文。在报告中，他们阐明了各自的学术思想，讨论作为其方法基础的各种原理，叙述使用其方法取得成功的例子，解释这些方法的效用，探讨这些方法的适用性和局限性。这些报告已在国际杂志《人工智能》第 47 卷（1991）1～3 期（合刊）上发表。

目前，人工智能的主要学派有下述 3 家。

① 符号主义（Symbolicism）学派，又称为逻辑主义（Logicism）学派、心理学派（Psychologism）或计算机学派（Computerism）。

② 联结主义（Connectionism）学派，又称为仿生学派（Bionicsism）或生理学派（Physiologism）。

③ 行为主义（Actionism）学派，又称为进化主义（Evolutionism）学派或控制论学派（Cyberneticsism）。

人工智能各学派的起源及进行人工智能研究所采用的基本理论和方法各不相同。

(1) 人工智能各学派的起源

符号主义认为人工智能源于数理逻辑。形式逻辑从 19 世纪末开始迅速发展，到 20 世纪 30 年代开始用于描述智能行为。计算机出现后，又在计算机上实现了逻辑演绎系统，其代表性的成果是 1956 年研制的逻辑理论程序 LT，可证明 38 条数学定理，从而表明可以应用计算

机研究人的思维过程，模拟人类智能活动。正是这些符号主义者在 1956 年首先采用“人工智能”这个术语；后来又发展了启发式算法、专家系统、知识工程理论与技术，并在 20 世纪 80 年代取得很大的发展。符号主义曾长期一枝独秀，为人工智能发展作出了重要贡献，尤其是专家系统的成功开发与应用，对人工智能走向工程应用具有特别重要的意义。在人工智能的其他学派出现之后，符号主义学派仍然是人工智能的主流。这个学派的代表人物有纽厄尔、肖、西蒙、尼尔逊(N. J. Nilsson)等。

联结主义认为人工智能源于仿生学，特别是人脑模型的研究。1943 年，由生理学专家麦卡洛克(McCulloch)和数理逻辑学家皮茨(Pitts)创立的脑模型，即 MP 模型，开创了用电子装置模仿人脑结构和功能的途径。它从神经元开始进而研究神经网络模型和脑模型，开辟了人工智能研究的新途径。20 世纪 60 年代至 70 年代，联结主义以感知器为代表的脑模型的研究曾出现过热潮，受当时的理论模型、生物原型和技术条件的限制，脑模型研究在 20 世纪 70 年代后期至 80 年代初期进入低潮。1982 年，美国加州工学院物理学家荷伯维尔德(Hopfield)提出了人工神经网络(Artificial Neural Network，ANN)的 HNN 模型，有力地推动了人工神经网络的研究。他引入了“计算能量函数”的概念，给出了神经网络稳定性判据。HNN 模型的电子电路的实现为神经计算机的研究奠定了基础，同时开拓了神经网络用于联想记忆和优化计算的新途径。1985 年，赫顿(Hinton)和塞罗斯基(Sejnowski)借用统计物理学的概念和方法，提出了 Boltzman 机模型，首次采用了多层神经网络的学习算法，即在学习过程中用模拟退火技术保证神经网络趋于全局稳定。1986 年，鲁梅尔哈特(Rumelhart)和麦克勒兰德(McClelland)提出多层神经网络的反向传播(BP)学习算法；通过实例的学习改变多层神经网络的邻接权矩阵，从而达到学习的目的。这是至今为止使用最广泛的人工神经网络。荷尔兰德(Holland)提出的分类系统类似于以规则为基础的专家系统，他提出的发现和改进规则的学习算法是对专家系统的重要发展。至此，人工神经网络的研究再次出现热潮。神经网络的发展为认知科学、计算机科学及人工智能的发展开辟了一条崭新的途径。虽然有不少人对神经网络研究的前景存在疑虑，但最悲观的估计仍然认为它会带来重大的科学研究成果和广泛的应用，而最乐观的估计则称之为一种新主义——联结主义，它是一种能解决知识表示、推理、学习、联想记忆等复杂系统的统一模型。

行为主义是人工智能的一个新的学派。行为主义源于控制论。在 20 世纪 50 年代，控制论的思想已对早期的人工智能工作者产生影响。早期的研究工作重点是模拟人在控制过程中的智能行为和作用，如对自寻优、自适应、自组织和自学习等控制系统的研究，并进行“控制论动物”的研制。在 20 世纪 60 年代至 70 年代，上述控制论系统的研究取得一定的进展，产生了智能控制和智能机器人的萌芽，并在 20 世纪 80 年代诞生了智能控制和智能机器人系统。行为主义学派的代表作首推布鲁克斯(Brooks)的六足行走机器人(机器虫)，它被看作新一代的“控制论动物”，是一个基于感知-动作模式的模拟昆虫行为的智能控制系统。

(2) 人工智能学派的基本理论框架

人类的认知过程是非常复杂的行为，至今仍未能完全被解释，人工智能各学派从不同的角度对人的智能行为进行研究，从而形成各学派研究人工智能的不同的基本理论。

符号主义认为可以用一个符号系统在计算机上形式化地描述和模拟人的思维活动过程。

符号主义还认为知识是智能的基础，人工智能的核心问题是知识表示与知识推理，知识是可以用一种符号系统来表示的，也可以用符号的操作进行知识推理。因此，有可能建立起基于知识的人类智能和机器智能的统一理论体系。

联结主义利用人工神经网络模仿人类智能，认为人的智能的基本单元是神经元，由许多人工神经元连接起来的人工神经网络可以具有自学习和自适应功能，能更好地模仿人类智能。人工神经网络并不是人脑的真实描写，只是人脑的某种抽象、简化与模拟。人工神经网络实际上是一种非线性自适应信息处理系统，信息处理由神经元之间的相互作用来实现，知识与信息的存储表现为网络各神经元之间的联系，学习则表现为网络各神经元连接权的动态变化过程。

行为主义认为智能取决于感知、表现为行动，智能行为只能在现实世界中与周围环境交互作用时表现出来，从而提出智能行为的“感知-动作”模式。

由于各学派对人工智能研究的基本理论框架不同，因此，对人工智能的研究方法也各不相同。

符号主义认为人工智能的研究方法应是功能模拟方法：分析人类认知系统所具备的功能和机理，然后用计算机来模拟这些功能，从而实现人工智能。符号主义力图用数理逻辑方法来建立人工智能的统一理论体系。

联结主义认为人工智能应着重于结构模拟，即模拟人的生理神经网络结构，并认为功能与结构是密切相关的，不同的结构表现出不同的功能和智能行为。至今，已经提出多种人工神经网络结构和多种学习算法。

行为主义认为人工智能的研究方法应采用行为模拟方法，也认为功能、结构和智能行为是不可分开的，不同的行为需要不同的控制结构并表现出不同的功能。行为主义的研究方法受到其他学派的怀疑：认为行为主义最多只能创造出低智能的昆虫行为，无法创造出人的高智能行为。

现在，在人工智能的基本理论、研究方法和技术路线等方面存在几种不同的学派，说明人工智能已经从“一枝独秀”的符号主义发展到多学派“百花争艳”，必将促进人工智能的进一步发展。

人工智能的近期研究目标是建造智能计算机，即：使现有的计算机更聪明、更有用。正是根据这一近期研究目标，才把人工智能理解为计算机科学的一个分支。人工智能的远期研究目标是研究人类智能和机器智能的基本原理，用智能机器来模拟人类的思维过程和智能行为。这个远期目标几乎涉及所有的自然科学学科和社会学科。

人类智能是人脑系统的整体效应，有着极为丰富的层次和侧面。符号主义、联结主义和行为主义都只抽象出人脑思维的部分特征来模仿人的智能和功能，人的认知过程和智能行为要比人们想象的复杂得多，因此，人工智能需要多学科长期协作研究，才能达到一个更高的水平。

1.2 人工智能的研究与应用领域

目前，人工智能的研究更多的是结合具体应用领域来进行的。这里，介绍几个主要的应用

研究领域。

1. 定理证明

数学领域中对臆测的定理寻求一个证明或反证，一直被认为是一项需要智能才能完成的工作。证明定理时，不仅需要具有根据假设进行演绎的能力，而且需要具有某些直觉的技巧。例如，为了求证一个定理，数学家会熟练地运用专业知识，设想需要先证明哪几个引理，精确判断出已有的哪些定理会在这个定理的证明过程中起作用，并把这个定理证明分解为若干子问题，分别独立地对子问题进行求解。

定理证明的研究在人工智能方法的发展中曾经产生过重要的影响。例如，采用谓词逻辑的演绎过程的形式化，可以帮助人们更清楚地理解演绎推理过程。许多其他领域的问题，如医疗诊断、信息检索等也可以应用定理证明的方法，因此，机器定理证明的研究在人工智能研究中具有普遍意义。

2. 专家系统

一般地说，专家系统是一个具有大量专门知识与经验的程序系统。专家系统存储有某个专门领域中经过事先总结分析并按某种模式表示的专家知识（组成知识库），以及拥有类似于领域专家解决实际问题的推理机制（构成推理机）。系统能对输入信息进行处理，并运用知识进行推理，做出决策和判断，其解决问题的水平达到或接近专家的水平，因此能起到专家或专家助手的作用。专家系统是人工智能中最活跃的一个应用研究领域，涉及社会各个方面，可以说，需要有专家工作的场合，就可以开发专家系统。

开发专家系统的关键是表示和运用专家知识，即：来自专家的已被证明对解决有关领域内的典型问题有用的事实和过程。目前，专家系统主要采用基于规则的知识表示和推理技术。由于领域的知识更多是不精确或不确定的，因此，不确定的知识表示与知识推理是专家系统开发与研究的重要课题。此外，专家系统开发工具的研制与发展也很迅速，这对扩大专家系统的应用范围，加快专家系统的开发过程，将起到积极促进的作用。随着计算机科学技术整体水平的提高，分布式专家系统、协同式专家系统等新一代专家系统的研究也发展很快。在新一代专家系统中，不但采用基于规则的推理方法，而且采用诸如人工神经网络的方法。

3. 机器学习

学习是人类智能的主要标志，也是人类获得知识的基本手段，学习能力无疑是人工智能研究的一个最重要的方面。学习是一个有特定目的的知识获取过程，其内部表现为新知识的不断建立和知识的更新，而外部则表现为系统的性能得到改善。机器学习过程本质上是把导师或专家提供的学习实例或信息转换成能被学习系统理解并应用的形式存储起来的过程。

传统的机器学习倾向于使用符号表示知识而不是使用数值表示知识，使用归纳方法进行学习而不是使用演绎方法进行学习。近几年来，又发展了下述各种学习方法：基于解释的学习方法、基于事例的学习方法、基于人工神经网络的学习方法、基于遗传算法的学习方法等。

4. 自然语言理解

自然语言是人类之间信息交流的主要媒介，由于人类有很强的理解自然语言的能力，因此，人们相互间的信息交流轻松自如。但是，目前计算机系统和人类之间的交互几乎还只能使用严格限制的各种非自然语言，因此，解决计算机系统理解自然语言的问题就是人工智能研究的一个十分重要的课题。在智能计算机的研究中，自然语言理解就是其中的重点研究课题之一。

一个独立的简单句子是比较容易理解它的含义的，但是，对于一段用自然语言表示的较长的文章或对话，要准确地理解其中每个句子的含义就不局限于这个句子本身，而与这个句子的上下文甚至背景知识有关。因此，自然语言理解涉及对上下文知识结构的表示和根据上下文知识进行推理的方法和技术。目前，在理解有限范围的自然语言对话和小段文章方面的程序系统研制已有一些进展，但是，实现功能较强的自然语言理解系统还是一个比较艰巨的任务。显然，在实现机器翻译时，如果机器系统能准确地理解每一个句子的含义，那么就能翻译出更准确、更通顺的译文。

5. 智能检索

数据库系统是存储某学科大量事实的计算机系统，随着应用的发展，存储的信息量愈来愈庞大。因此，研究智能检索系统具有重要的实际意义。

智能信息检索系统应具有下述功能。

① 理解自然语言，允许用户使用自然语言提出检索要求和询问。

② 具有推理能力，能根据数据库存储的事实，推理产生用户要求和询问的答案。

③ 系统拥有一定的常识性知识，以补充数据库中学科范围的专业知识。系统根据这些常识性知识和专业知识能演绎推理出专业知识中没有包含的答案。例如，某单位的人事档案数据库中有下列事实："张强是采购部工作人员"、"李明是采购部经理"。如果系统具有"部门经理是该部门工作人员的领导"这一常识性知识，就可以对询问"谁是张强的领导"演绎推理出答案"李明"。

6. 机器人学

随着工业自动化和计算机技术的发展，到 20 世纪 60 年代，机器人开始进入大量生产和实际应用阶段。后来，由于自动装配、海洋开发、空间探索等领域的需要，对机器人的智能水平提出了更高的要求。特别是危险环境和恶劣环境更迫切需要机器人代替人来工作，从而推动了智能机器人的研究。

智能机器人的运动规划分为高层规划和低层规划两个层次。先由高层规划根据感知的环境信息和要求实现的目标规划出机器人执行动作的命令序列，然后由低层规划将每一个动作命令转换成驱动机器人各关节运动的驱动电机的角速度或角位移，各关节驱动电机的协调运动将保证实现相应的动作命令。

智能机器人是多学科交叉的综合课题，它涉及精密机械，视觉、触觉、力觉等信息传感技

术，自动控制，人工智能的规划方法等。这一课题的研究有利于促进各学科的相互结合和渗透，并将大大推动人工智能技术的发展。

7. 自动程序设计

自动程序设计是指：设计一个能自动生成程序的程序系统，这个程序系统只需要对其输入要求生成的程序的实现目标的高级的描述，就能自动生成能完成这个目标的程序。

从某种意义上来说，编译程序实际上就是做“自动程序设计”的工作，编译程序接受做某一件工作的源代码（源程序），然后生成目标代码（目标程序）去执行这件工作。这里所说的自动程序设计相当于“超级编译程序”，它要求不是给出完整的源代码来详细说明要做的工作，而只需要对要做的工作给出目标性的高级描述就可以生成完成这个工作的程序。

自动程序设计所涉及的基本问题与定理证明和机器人学涉及的问题有关，要求对高级的目标描述通过规划过程生成所需的程序。

8. 组合调度问题

有许多实际问题是属于最佳调度或最佳组合问题。例如，推销员旅行问题就属于这一类问题。推销员旅行问题是：推销员从某个城市出发，遍访他所要访问的城市一次（仅一次），回到他出发的城市，求推销员最短的旅行路线。该问题的一般化为：对若干节点组成的一个图，寻找一条最小耗费的路径，使这条路径对每个节点穿行一次。

在大多数组合调度问题中，随着求解问题规模的增大，求解程序都面临着组合爆炸问题。在推销员旅行问题中，问题规模可用需要穿行的城市数目来表示。随着求解问题规模的增大，问题求解程序的复杂性（用于求解程序运行所需的时间和空间或求解步数）可随问题规模按线性关系、多项式关系或指数关系增长。

组合调度问题中有一类问题称为 NP 完全问题，NP 完全问题是指：用目前知道的最好的方法求解，问题求解需要花费的时间（或称为问题求解的复杂性）随问题规模增大以指数关系增长。推销员旅行问题就是一个 NP 完全问题，我们至今还不知道对 NP 完全问题是否有花费时间较少的求解方法。例如，可使求解时间随问题规模按多项式关系增长。

组合调度问题的求解方法已经应用于交通运输调度、列车编组、空中交通管制和军事指挥自动化等系统。

9. 模式识别

“模式”（Pattern）一词的本意是指完整无缺的供模仿的标本或标识。模式识别就是识别出给定物体所模仿的标本或标识。计算机模式识别系统使一个计算机系统具有模拟人类通过感官接受外界信息、识别和理解周围环境的感知能力。

模式识别是一个不断发展的学科分支，它的理论基础和研究范围也在不断发展。在二维的文字、图形和图像的识别方面，已取得许多成果。三维景物和活动目标的识别和分析是目前研究的热点。语音的识别和合成技术也有很大的发展。基于人工神经网络的模式识别技术在手写字符的识别、汽车牌照的识别、指纹识别、语音识别等方面已经有许多成功的应用。模式

识别技术是智能计算机和智能机器人研究的十分重要的基础。

10. 机器视觉

实验表明，人类接受外界信息的80%以上来自视觉，10%左右来自听觉，其余来自嗅觉、味觉及触觉。在机器视觉方面，只要给计算机系统装上电视摄像输入装置就可以“看见”周围的东西。但是，视觉是一种感知，机器视觉的感知过程包含一系列的处理过程，例如，一个可见的景物由传感器编码输入，表示成一个灰度数值矩阵；图像的灰度数值由图像检测器进行处理，检测器检测出图像的主要成分，如组成景物的线段、简单曲线和角度等；这些成分又被处理，以便根据景物的表面特征和形状特征来推断有关景物的特征信息；最终目标是利用某个适当的模型来表示该景物。

视觉感知问题的要点是形成一个精练的表示来取代极其庞大的未经加工的输入信息，把庞大的视觉输入信息转化为一种易于处理和有感知意义的描述。

机器视觉可分为低层视觉和高层视觉两个层次。低层视觉主要是对视觉图像执行预处理，例如，边缘检测、运动目标检测、纹理分析等，另外还有立体造型、曲面色彩等，其目的是使对象凸现出来，这时还谈不上对它的理解。高层视觉主要是理解对象，显然，实现高层视觉需要掌握与对象相关的知识。

机器视觉的前沿研究课题包括：实时图像的并行处理，实时图像的压缩、传输与复原，三维景物的建模识别，动态和时变视觉等。

习 题 一

1.1 何谓人工智能？人类智能主要包括哪些？

1.2 “知识工程”是在什么背景下提出的？知识工程对人工智能的发展有何重要作用？

1.3 当前计算机发展的主要趋势是什么？智能计算机的主要研究方向有哪些？

1.4 人工智能有哪几个主要学派？各学派的基本理论框架和研究方法有何不同？

1.5 人工智能的近期研究目标与远期研究目标分别是什么？

1.6 人工智能主要的应用研究领域有哪些？

第 2 章

知识表示方法

人类的智能活动过程主要是一个获得知识并运用知识的过程，知识是智能的基础。为了使计算机具有智能，能模拟人类的智能行为，就必须使它具有知识。把人类拥有的知识采用适当的模式表示出来以便存储到计算机中，这就是知识表示要解决的问题。知识表示是对知识的一种描述，或者说是一组约定，是一种计算机可以接受的用于描述知识的数据结构，对知识进行表示就是把知识表示成便于计算机存储和利用的某种数据结构。知识表示方法给出的知识表示形式称为知识表示模式，知识表示模式分为外部表示模式和内部表示模式两个层次。知识外部表示模式是与软件开发的工具、运行的软件平台无关的知识表示的形式化描述。知识内部表示模式是与软件开发工具及平台有关的知识表示的存储结构。对某一种知识外部表示模式采用不同的软件开发工具与平台实现时，其内部表示模式不同。

目前使用较多的知识表示方法主要有：一阶谓词逻辑表示方法、产生式表示方法、框架表示方法、语义网络表示方法、面向对象表示方法和基于人工神经网络的知识表示方法等。根据是否可以表示不确定性知识，知识表示方法分为确定性知识表示和不确定性知识表示两类。本章讨论确定性知识表示中的一阶谓词逻辑表示方法、产生式表示方法和框架表示方法。

2.1 一阶谓词逻辑表示方法

谓词逻辑是一种形式语言，也是目前能够表达人类思维活动的一种最精确的语言，它与人类的自然语言比较接近，可以方便地存储到计算机中并被计算机处理，因此成为最早应用于人工智能中表示知识的一种逻辑表示方法。

2.1.1 一阶谓词逻辑表示

谓词逻辑是在命题逻辑的基础上发展起来的、对于知识的一种形式化表示，它在定理的自动证明中发挥了重要作用，在人工智能发展史中占有重要地位。

1. 命题

命题是具有真假意义的语句。命题代表人们进行思维时的一种判断，或者是肯定，或者是否定，只有这两种情况。若命题的意义为真，则称它的真值为真，记为 T；若命题的意义为假，则称它的真值为假，记为 F。一个命题不能同时既为真又为假，但可以在一定条件下为真，在另一种条件下为假。没有真假意义的语句（如感叹句、疑问句等）不是命题。例如，“中华人民

共和国的首都是北京”,“3<4”都是真值为 T 的命题;“太阳从西边升起”是真值为 F 的命题;“1+1=10”在二进制情况下是真值为 T 的命题,但在十进制情况下是真值为 F 的命题。

命题逻辑的这种表示有较大的局限性,它无法描述客观事物的结构及逻辑特征,也不能把不同事物间的共同特征表述出来。例如,对于“杨青是教师”和“李文是教师”这两个命题,用命题逻辑表示时,无法把两人都是教师这一共同特征表示出来。

2. 谓词

在谓词逻辑中,一个谓词可分为谓词名与个体两个部分,个体表示某个独立存在的事物或者某个抽象的概念,谓词名用于刻画个体的性质、状态或个体间的关系。例如,对于“老张是教师”这个命题,用谓词可表示为 Teacher (Zhang),其中,Teacher 是谓词名,Zhang 是个体,Teacher 刻画了 Zhang 是教师的职业特征;又如,“5>3”这个命题可用谓词表示为 Greater (5, 3),Greater 刻画了两个个体“5”与“3”之间的“大于”关系。

一阶谓词的一般形式为

$$P(x_1, x_2, \cdots, x_n)$$

其中,P 是谓词名,$x_1, x_2, \cdots, x_n$ 是个体。谓词名通常用大写英文字母表示,个体通常用小写英文字母表示。谓词中包含的个体数目称为谓词的元数,例如,$P(x)$ 是一元谓词,$P(x\ y)$ 是二元谓词,$P(x_1, x_2, \cdots, x_n)$ 是 n 元谓词。

一阶谓词中的个体可以是常量,也可以是变元,还可以是一个函数。例如,“小王的父亲是教师”可以表示为 Teacher(father (Wang)),其中,father(Wang)是一个函数,表示“小王的父亲”,它是谓词 Teacher 的个体。又如,“$x<5$”可以表示为 Less(x,5),其中,x 是变元,5 是常量。显然,当谓词中的变元都用特定的个体取代时,谓词就成为一个命题并具有一个确定的真值:T 或 F。

个体变元的取值范围称为个体域。个体域可以是有限的,也可以是无限的。例如,若用 $I(x)$ 表示“x 是整数”,则变元 x 的个体域 X 是所有整数,它是无限的。

谓词与函数表面上很相似,容易混淆,其实这是两个完全不同的概念。谓词的真值是“真”或“假”,而函数的值是某个个体域中的某个个体,函数只是从一个个体到另一个个体的映射,函数无真值可言。例如,father(Wang)是把个体“小王”映射到另一个个体“小王的父亲”。

个体常量与个体变元、函数统称为“项”。

在谓词 $P(x_1, x_2, \cdots, x_n)$ 中,若 $x_i(i=1,2,\cdots,n)$ 都是个体常量、变元或函数,则称它为一阶谓词。若某个 x_i 本身又是一个一阶谓词,则称 P 为二阶谓词,余者类推。为了区别谓词名与项,规定:谓词名或谓词名的第一个字符用大写字符表示,项中的常量用大写字符表示,项中的变元和函数名(或函数名的第一个字符)及函数的变元都用小写字符表示。

3. 谓词公式

在谓词逻辑中,可以用连词来连接若干个谓词组成一个谓词公式,以表示一个比较复杂的含义。在谓词公式中还可以引入量词来刻画谓词与个体间的关系。

(1) 连词

在谓词逻辑中,有下述连词。

① 非连词¬。其作用是否定位于后面的命题。当命题 P 为真时，$\neg P$ 为假；当 P 为假时，$\neg P$ 为真。

② 或连词 $\vee$。它表示被它连接的两个命题有“或”关系。用 $\vee$ 连接两个命题称为析取。

③ 与连词 $\wedge$。它表示被它连接的两个命题有“与”关系。用 $\wedge$ 连接两个命题称为合取。

④ 蕴含连词→。它表示被它连接的两个命题的“蕴含”关系。$P \rightarrow Q$ 表示“P 蕴含 Q”，即“如果 P，则 Q”，其中 P 称为前件，Q 称为后件。

以上连词的逻辑真值表由表 2.1 给出。

表 2.1　谓词逻辑真值表

P　Q	$\neg P$	$P \vee Q$	$P \wedge Q$	$P \rightarrow Q$
T　T	F	T	T	T
T　F	F	T	F	F
F　T	T	T	F	T
F　F	T	F	F	T

(2) 量词

在谓词逻辑中，有下述两个量词。

① 全称量词($\forall x$)。它表示对个体域 X 中的所有(或任一个)个体 x。

② 存在量词($\exists x$)。它表示在个体域 X 中存在个体 x。

例如，若谓词 $P(x)$表示 x 是正数，$F(x,y)$表示 x 与 y 是朋友，则

$(\forall x)P(x)$表示个体域 X 中的所有个体 x 都是正数；

$(\forall x)(\exists y)F(x,y)$表示对于个体域 X 中的任何个体 x，在个体域 Y 中都存在个体 y，x 与 y 是朋友；

$(\exists x)(\forall y)F(x,y)$表示在个体域 X 中存在个体 x，与个体域 Y 中的任何个体 y 都是朋友；

$(\exists x)(\exists y)F(x,y)$表示在个体域 X 中存在个体 x 和在个体域 Y 中存在个体 y，x 与 y 是朋友。

(3) 谓词公式

由下述规则得到的谓词公式称为合式公式：

① 单个谓词和单个谓词的否定称为原子谓词公式，原子谓词公式是合式公式。

② 若 A 是合式公式，则 $\neg A$ 也是合式公式。

③ 若 A、B 都是合式公式，则 $A \vee B$、$A \wedge B$、$A \rightarrow B$ 也都是合式公式。

④ 若 A 是合式公式，x 是任一个体变元，则 $(\forall x)A$ 和 $(\exists x)A$ 也都是合式公式。

在合式公式中，连词的优先级别依序为：¬，∧，∨，→。

位于量词后面的单个谓词或者用括弧括起来的合式公式称为该量词的辖域，辖域内与量词变元同名的变元称为约束变元，不受约束的变元称为自由变元。例如

$$(\exists x)(P(x,y) \rightarrow Q(x,y)) \vee R(x,y)$$

其中，$(P(x,y) \rightarrow Q(x,y))$是量词 $(\exists x)$ 的辖域，辖域内的变元 x 是受 $(\exists x)$ 约束的变元，而 $R(x,y)$中的 x 是自由变元，公式中所有的 y 都是自由变元。

在谓词公式中，可以把一个变元的名字换成另一个名字。但必须注意，当对量词辖域内的约束变元更名时，必须把辖域内同名的约束变元都统一改成相同的名字，且不能与辖域内的自由变元同名；当对辖域内的自由变元改名时，不能将其改成与约束变元相同的名字。例如，对于公式$(\forall x)P(x,y)$，可以改名为$(\forall z)P(z,t)$，这里，把约束变元 x 更名为 z，把自由变元 y 更名为 t。

4. 谓词公式的解释

在命题逻辑中，对命题公式中各个命题的一次真值指派称为命题公式的一个解释。一旦解释确定后，根据各连词的定义就可求出命题公式的真值(T 或 F)。在谓词逻辑中，由于公式中可能有个体常量、个体变元和函数，因此不能像命题公式那样直接通过真值指派给出解释，必须首先考虑个体变元和函数在个体域中的取值，然后才能针对变元与函数的具体取值为谓词分别指派真值。由于存在多种组合情况，所以一个谓词公式的解释可能有很多个。对于每一个解释，谓词公式都可求出一个真值(T 或 F)。

首先给出谓词公式的解释定义，然后用例子说明如何构造一个解释及如何根据解释求出谓词公式的真值。

定义 2.1 设 D 为谓词公式 P 的个体域，若对 P 中的个体常量、函数和谓词按如下规定赋值：

① 为每个个体变元指派 D 中的一个元素。

② 为每个 n 元函数指派一个从 D^n 到 D 的映射，其中，

$$D^n=\{(x_1,x_2,\cdots,x_n)\mid x_1,x_2,\cdots,x_n\in D\}$$

③ 为每个 n 元谓词指派一个从 D^n 到$\{F,T\}$的映射。

则称这些指派为公式 P 在域 D 上的一个解释。

例 2.1 设变元 x 和 y 的个体域是 $D=\{1,2\}$，谓词 $P(x,y)$表示 x 大于等于 y，请给出公式 $A=(\forall x)(\exists y)P(x,y)$在 D 上的解释，并指出在每一种解释下公式 A 的真值。

解 由于在公式 A 中没有包括个体常量和函数，所以可由谓词 $P(x,y)$的定义得出谓词的真值指派。设对谓词 $P(x,y)$在个体域 D 上的真值指派为

$$P(1,1)=T,\quad P(1,2)=F,\quad P(2,1)=T,\quad P(2,2)=T$$

这就是公式 A 在 D 上的一个解释。在此解释下，因为 $x=1$ 时有 $y=1$ 使 $P(x,y)$的真值为 T，$x=2$ 时也有 $y=1$ 使 $P(x,y)$的真值为 T，即 x 对于 D 中的所有取值，都存在 $y=1$，使 $P(x,y)$的真值为 T，所以在此解释下公式 A 的真值为 T。

例 2.2 设个体域 $D=\{1,2\}$，试给出公式 $R=(\forall x)(P(x)\rightarrow Q(f(x),B))$在 D 上的一个解释，并指出公式 R 在此解释下的真值。

解 设对个体常量 B 指派 D 中的一个元素为 $B=1$，对函数 $f(x)$指派到 D 的映射为 $f(1)=2$，$f(2)=1$；对谓词指派的真值为

$$P(1)=F,\quad P(2)=T,\quad Q(1,1)=T,\quad Q(2,1)=F$$

这里，由于已对个体常量 B 指派 $B=1$，所以 $Q(1,2)$与 $Q(2,2)$不可能出现，因此没有给它们指派真值。

上述指派就是对公式 R 的一个解释。在此解释下，由于当 $x=1$ 时，有

$$P(1)=\mathrm{F},\quad Q(f(1),1)=Q(2,1)=\mathrm{F}$$

所以 $P(1)\rightarrow Q(f(1),1)$ 的真值为 T。当 $x=2$ 时，有

$$P(2)=\mathrm{T},\quad Q(f(2),1)=Q(1,1)=\mathrm{T}$$

所以 $P(2)\rightarrow Q(f(2),1)$ 的真值也为 T，即：对个体域 D 中的所有 x 都有

$$P(x)\rightarrow Q(f(x),B)$$

的真值为 T。所以，公式 R 在此解释下的真值为 T。

由上述的例子可见，谓词公式的真值是针对某一个解释而言的，它可能在某一个解释下的真值为 T，在另一个解释下的真值为 F。

5. 谓词公式的永真性、可满足性、不可满足性

定义 2.2　如果谓词公式 P 对个体域 D 上的任何一个解释都取得真值 T，则称公式 P 在域 D 上是永真的。如果 P 在每个非空个体域上均永真，则称 P 是永真的。

由此定义可以看出，为了判定某个公式永真，必须对每个个体域上的每一个解释逐一判定公式的真值。当解释的个数为有限时，尽管工作量较大，总还是可以判定的，但当解释的个数为无限时，公式的永真性就难以判定了。

定义 2.3　对于谓词公式 P，如果至少存在一个解释使得公式 P 在此解释下的真值为 T，则称公式 P 是可满足的。

定义 2.4　如果谓词公式 P 对于个体域 D 上的任何一个解释都取得真值 F，则称公式 P 在域 D 上是永假的。如果 P 在每个非空个体域上均永假，则称 P 是永假的。

谓词公式的永假性又称为不可满足性。

6. 谓词公式的等价性

定义 2.5　设 P 与 Q 是两个谓词公式，D 是它们共同的个体域，若对 D 上的任何一个解释，P 与 Q 都有相同的真值，则称公式 P 和 Q 在 D 上是等价的。如果 D 是任意的个体域，则称 P 和 Q 是等价的，记为 $P\Leftrightarrow Q$。

下面列出今后要用到一些主要的等价式。

- **交换律**

$$P\vee Q\Leftrightarrow Q\vee P$$

$$P\wedge Q\Leftrightarrow Q\wedge P$$

- **结合律**

$$(P\vee Q)\vee R\Leftrightarrow P\vee(Q\vee R)$$

$$(P\wedge Q)\wedge R\Leftrightarrow P\wedge(Q\wedge R)$$

- **分配律**

$$P\vee(Q\wedge R)\Leftrightarrow(P\vee Q)\wedge(P\vee R)$$

$$P\wedge(Q\vee R)\Leftrightarrow(P\wedge Q)\vee(P\wedge R)$$

- **狄·摩根律**

$$\neg(P\vee Q)\Leftrightarrow\neg P\wedge\neg Q$$

$$\neg(P\wedge Q)\Leftrightarrow\neg P\vee\neg Q$$

- **双重否定律**

$$\neg\ \neg P\Leftrightarrow P$$

- **吸收律**

$$P\vee(P\wedge Q)\Leftrightarrow P$$

$$P \wedge (P \vee Q) \Leftrightarrow P$$

- **补余律** $P \vee \neg P \Leftrightarrow \mathrm{T}$

$$P \wedge \neg P \Leftrightarrow \mathrm{F}$$

- **连词化归律** $P \rightarrow Q \Leftrightarrow \neg P \vee Q$
- **量词转换律** $\neg(\exists x)P \Leftrightarrow (\forall x)(\neg P)$

$$\neg(\forall x)P \Leftrightarrow (\exists x)(\neg P)$$

- **量词分配律** $(\forall x)(P \wedge Q) \Leftrightarrow (\forall x)P \wedge (\forall x)Q$

$$(\exists x)(P \vee Q) \Leftrightarrow (\exists x)P \vee (\exists x)Q$$

7. 谓词公式的永真蕴含

定义 2.6 对于谓词公式 P 和 Q，如果 $P \rightarrow Q$ 永真，则称 P 永真蕴含 Q，且称 Q 为 P 的逻辑结论，称 P 为 Q 的前提，记为 $P \Rightarrow Q$。

8. 推理规则、定理与证明

上面列出的等价式和永真蕴含式是进行演绎推理的重要依据，应用这些公式可保证推理的有效性，因此这些公式又称为推理规则。推理规则用来由已知的合式公式推导出新的合式公式。在谓词逻辑中导出的合式公式称为定理，而所使用的推理规则的序列则构成该定理的一个证明。在人工智能中，可以把某些问题求解的任务表示为某个定理求证的任务，在该证明中所应用的推理规则序列给出了该问题的一个解答。

除上述推理规则之外，谓词逻辑中还有如下的一些推理规则。

- P 规则　在推理的任何步骤上都可引入前提。
- T 规则　在推理时，如果前面步骤中有一个或多个公式永真蕴含公式 S，则可把 S 引入推理过程中。
- CP 规则　如果能从 R 和前提集合中推出 S 来，则可从前提集合推出 $R \rightarrow S$。
- **反证法规则**　$P \Rightarrow Q$，当且仅当 $P \wedge \neg Q \Leftrightarrow \mathrm{F}$，即：$Q$ 为 P 的逻辑结论，当且仅当 $P \wedge \neg Q$ 是不可满足的。

把反证法推广到谓词公式集，可得到以下反证法定理。

定理 2.1　Q 为 $P_1, P_2, \cdots, P_n$ 的逻辑结论，当且仅当

$$(P_1 \wedge P_2 \wedge \cdots \wedge P_n) \wedge \neg Q$$

是不可满足的。

该定理将在归结反演中得到应用，它是归结反演的理论根据。

2.1.2 一阶谓词逻辑表示方法的特点

谓词逻辑适合于表示事物的状态、属性、概念等事实性的知识，也可以用来表示事物间确定的因果关系，即规则。事实通常用谓词公式的与/或形表示，所谓与/或形是指用合取符号（$\wedge$）及析取符号（$\vee$）连接起来的公式。规则通常用蕴含式表示。

用谓词公式表示知识时，首先需要定义谓词，给出每个谓词的确切含义，然后用连词把有关谓词连接起来表示一个更复杂的含义。对谓词公式中的变元，根据知识表示的需要，把需要约束的变元用相应的量词予以约束。

例 2.3　用谓词公式表示下列知识：

王林是计算机系的学生，但他不喜欢编程序。

人人爱劳动。

解　首先定义下列谓词：

COMPUTER(x)	表示 x 是计算机系的学生
LIKE(x,y)	表示 x 喜欢 y
LOVE(x,y)	表示 x 爱 y
MAN(x)	表示 x 是人

然后用谓词公式把上述知识表示为

$$\text{COMPUTER(WangLin)} \wedge \neg \text{LIKE(Wang Lin, Programing)}$$

$$(\forall x)(\text{MAN}(x) \rightarrow \text{LOVE}(x, \text{Labour}))$$

例 2.4　用谓词公式表示下列知识：

自然数是大于零的整数。

所有整数不是偶数就是奇数。

偶数除以 2 是整数。

解　首先定义下列谓词：

$N(x)$	表示 x 是自然数
$I(x)$	表示 x 是整数
$E(x)$	表示 x 是偶数
$O(x)$	表示 x 是奇数
$GZ(x)$	表示 x 大于零

然后用谓词公式分别表示上述知识：

$$(\forall x)(N(x) \rightarrow GZ(x) \wedge I(x))$$

$$(\forall x)(I(x) \rightarrow E(x) \vee O(x))$$

$$(\forall x)(E(x) \rightarrow I(f(x)))$$

其中，函数 $f(x)=x/2$。

1. 一阶谓词逻辑表示方法的优点

一阶谓词逻辑是一种形式语言，它用逻辑方法研究推理的规律，即条件与结论之间的蕴含关系。一阶谓词逻辑表示知识的方法有如下优点。

(1) 自然性

谓词逻辑是一种接近于自然语言的形式语言，人们比较容易接受，用它表示知识比较容易理解。

(2) 精确性

谓词逻辑是二值逻辑，谓词公式的真值只有“真”与“假”，因此可用它表示精确知识，并可

保证经演绎推理所得出的结论的精确性。

(3) 严密性

谓词逻辑具有严格的形式定义及推理规则,利用这些推理规则及有关定理证明的方法和技术可从已知事实推出新的事实,或证明提出的假设。

(4) 容易实现

用谓词逻辑表示的知识可以比较容易地转换为计算机易于存储与处理的内部表示模式,便于实现对知识的增加、删除与修改。基于谓词逻辑知识表示的归结演绎推理易于在计算机上实现。

2. 一阶谓词逻辑表示方法的局限性

(1) 不能表示不确定的知识

现实世界中的许多知识并不总是只有"真"与"假"两种状态,在"真"与"假"之间还存在许多中间状态,即存在为"真"的程度问题,知识的这一特性称为知识的不确定性。例如,"王林可能是计算机系的学生"就是一个不确定的事实;"如果头痛且流涕,则可能患了感冒"是一个不确定的规则。

谓词逻辑只能表示确定的知识,不能表示不确定的知识,但是,由于人类的知识大多都不同程度地具有不确定性,这就使得它表示知识的范围受到限制。另外,谓词逻辑难以表示启发性知识,启发性知识是与被求解的问题的特性有关的知识。

(2) 易导致组合爆炸

在谓词逻辑的推理过程中,随着事实数目的逐步增大及盲目地使用推理规则,有可能导致组合爆炸。目前,在这一方面已做了大量的研究工作,出现了一些比较有效的方法,例如,定义一种控制策略来选取合适的规则等。

(3) 效率低

用谓词逻辑表示知识时,其推理是根据形式逻辑进行的,把推理与知识的语义割裂开来,这就使得推理过程冗长,降低了系统的效率。

2.2 产生式表示方法

产生式表示方法又称为产生式规则表示方法。"产生式"这一术语是由美国数学家波斯特(E. Post)在 1943 年首先提出来的,他根据串替代规则提出了一种称为波斯特机的计算模型,模型中的每一条规则称为一个产生式。在此之后,经过不断修改和充实,如今已被用到许多领域。例如,用产生式来描述形式语言的语法,表示人类心理活动的认知过程等。1972 年,纽厄尔和西蒙在研究人类的认识模型中开发了基于规则的产生式系统。目前它已成为人工智能中应用最多的一种知识表示模式,许多成功的专家系统都是用产生式来表示知识的。例如,费根鲍姆等人研制的化学分子结构专家系统 DENDRAL、肖特里菲等人研制的诊断和治疗细菌感染性疾病的专家系统 MYCIN 等。

2.2.1　产生式与产生式系统

1. 产生式的基本形式

产生式通常用于表示具有因果关系的知识，其基本形式是

$$P \rightarrow Q \qquad \text{或者} \qquad \text{if } P \text{ then } Q$$

其中，P 是产生式的前提，亦可称为前件、条件、前提条件，用于指出该产生式是否可用的条件；Q 是产生式的结论或操作，亦可称为后件，用于指出当前提 P 所指示的条件被满足时，应该得出的结论或应该执行的操作。产生式的含义是：如果前提 P 被满足，则可推出结论 Q 或执行 Q 所规定的操作。

谓词逻辑中的蕴含式与产生式的有相同的基本形式，但是，蕴含式与产生式是有差别的，实际上，可以把蕴含式看成是产生式的一种特殊情况，这是因为

① 蕴含式只能表示精确知识，其真值或者为真、或者为假；而产生式不仅可以表示精确知识，也可以表示不精确知识。例如，在专家系统 MYCIN 中有这样一条产生式：

if　微生物的染色斑是革兰氏阴性
　　微生物的形状呈杆状
　　病人是中间宿主
then　该微生物是绿脓杆菌，可信度为 0.6

它表示当前提中列出的 3 个条件都满足时，结论“该微生物是绿脓杆菌”可以相信的程度为 0.6，这里，用 0.6 指出了这一知识的知识强度。但对于谓词逻辑中的蕴含式是不可以这样做的。

② 在用产生式表示知识的系统中，决定一条知识是否可用的方法是检查当前是否有已知事实可与前提中规定的条件匹配，且匹配可以是精确的，也可以是不精确的，只要按某种算法求出的相似度落在某个预先指定的范围内就认为是可匹配的。但对谓词逻辑的蕴含式来说，要求匹配是精确的。

由于产生式与蕴含式的这些区别，它们在处理方法及应用等方面都有较大的差别。

为了严格地描述产生式，下面用巴科斯范式 BNF(Backus Normal Form)给出产生式的形式描述及语义：

<产生式>::=<前提>→<结论>
<前　提>::=<简单条件>|<复合条件>
<结　论>::=<事实>|<操作>
<复合条件>::=<简单条件>AND<简单条件>[(AND<简单条件>)…]
　　|<简单条件>OR<简单条件>[(OR<简单条件>)…]
<操　作>::=<操作名>[(<变元>,…)]

2. 产生式系统

把一组产生式放在一起，并让它们互相配合，协同作用，一个产生式生成的结论可以供另

一个产生式作为已知事实使用，以求得问题的解决，这样的系统称为产生式系统。一个产生式系统由以下 3 个基本部分组成：规则库、综合数据库和控制机构。

(1) 规则库

用于描述相应领域内知识的产生式集合称为规则库。规则库是产生式系统进行问题求解的基础，其中的知识是否完整、一致，表达是否准确，对知识的组织是否合理等，不仅直接影响系统的性能，而且影响系统的运行效率，因此对规则库的设计与组织应予以足够的重视。一般来说，在建立规则库时应注意以下问题。

① 有效地表达领域内的过程性知识。

规则库中存放的主要是过程性知识，用于实现对问题的求解。为了使系统具有较强的问题求解能力，除了需要获得足够的知识外，还需要对知识进行有效的表达。为此，需要解决如下一些问题：如何把领域中的知识表达出来，即为了求解领域内的各种问题需要建立哪些产生式规则？对知识中的不确定性如何表示？规则库建成后能否对领域内的不同求解问题分别形成相应的推理链，即规则库中的知识是否具有完整性？对于第一个问题可以从下面给出的一个典型例子中得到启发，其他的问题将在后面的章节中予以讨论。

例 2.5　建立一个动物识别系统的规则库，用以识别虎、豹、斑马、长颈鹿、企鹅、鸵鸟、信天翁等 7 种动物。

解　为了识别这些动物，可以根据动物识别的特征，建立包含下述规则的规则库。

R_1：if 动物有毛发 then 动物是哺乳动物

R_2：if 动物有奶 then 动物是哺乳动物

R_3：if 动物有羽毛 then 动物是鸟

R_4：if 动物会飞 and 会生蛋 then 动物是鸟

R_5：if 动物吃肉 then 动物是食肉动物

R_6：if 动物有犀利牙齿 and 有爪 and 眼向前方 then 动物是食肉动物

R_7：if 动物是哺乳动物 and 有蹄 then 动物是有蹄类动物

R_8：if 动物是哺乳动物 and 反刍 then 动物是有蹄类动物

R_9：if 动物是哺乳动物 and 是食肉动物 and 有黄褐色 and 有暗斑点 then 动物是豹

R_{10}：if 动物是哺乳动物 and 是食肉动物 and 有黄褐色 and 有黑色条纹 then 动物是虎

R_{11}：if 动物是有蹄类动物 and 有长脖子 and 有长腿 and 有暗斑点 then 动物是长颈鹿

R_{12}：if 动物是有蹄类动物 and 有黑色条纹 then 动物是斑马

R_{13}：if 动物是鸟 and 不会飞 and 有长脖子 and 有长腿 and 有黑白二色 then 动物是鸵鸟

R_{14}：if 动物是鸟 and 不会飞 and 会游泳 and 有黑白二色 then 动物是企鹅

R_{15}：if 动物是鸟 and 善飞 then 动物是信天翁

由上述产生式规则可以看出，虽然该系统是用来识别 7 种动物的，但并不是简单地只设计 7 条规则分别直接用于识别 7 种动物。规则设计的基本思想是：首先把动物划分为若干类，如“哺乳动物”、“鸟”、“食肉动物”、“有蹄类动物”等，根据“类”的识别特征建立若干条规则，如规则 $R_1 \sim R_8$；然后对各类的各个动物根据其个性的识别特征建立各自相应的规则，如规则 $R_9 \sim R_{15}$。这样至少有两个好处，一是当给出的已知事实不完全时，虽然不能推出最终结论，但可能

会给出分类结果；二是当需要增加对其他动物（如牛、马等）的识别要求时，规则库中只需增加关于这些动物个性方面的知识，对于规则库中已有的分类知识（如规则 $R_1 \sim R_8$）就可以直接使用。

② 对知识进行合理的组织与管理。

对规则库中的知识进行适当的组织，采用合理的结构形式，可避免访问那些与当前问题求解无关的知识，从而提高问题求解的效率。

例如，对例 2.5 的规则库，可根据“哺乳动物”和“鸟”这两类动物的识别规则将 15 条规则分为两个子集：

$$\{R_1、R_2、R_5、R_6、R_7、R_8、R_9、R_{10}、R_{11}、R_{12}\}$$

$$\{R_3、R_4、R_{13}、R_{14}、R_{15}\}$$

当待识别动物的识别过程一旦开始使用其中某一个子集的规则时，就可以只使用该子集中的规则完成推理过程，而无需在另一个子集中去查找所需要的规则，从而减少查找规则的时间。当然，这种划分还可以逐级进行下去，使得相关的知识构成一个子集或子子集，组成一个层次型的规则库。

(2) 综合数据库

综合数据库又称为全局数据库，或称为事实库、黑板。综合数据库用于存放问题求解过程中各种当前信息，例如，问题的初始事实、原始证据、推理中得到的中间结论（如上例中的“动物是哺乳动物”、“动物是鸟”等）及最终结论（如上例中的“动物是虎”等）。当规则库中某一条产生式的前提可与综合数据库中的某些已知事实匹配时，该产生式就被激活，并把用它推出的结论放入综合数据库中，作为其后推理的已知事实。可见，综合数据库的内容随着推理的进行是在不断动态变化的。

(3) 控制机构

控制机构又称为推理机构或推理机，由一组程序组成，实现对问题的推理求解。正向推理的推理机主要功能如下。

① 按某种策略从规则库中选择规则与综合数据库中的已知事实进行匹配。所谓匹配是指把规则的前提条件与综合数据库中的已知事实进行比较，如果两者一致或者近似一致且满足预先规定的近似程度，则称匹配成功，其相应的规则称为可用规则；否则，称为匹配不成功，其相应的规则不可用于当前的推理。

② 若匹配成功的可用规则有多条，则称为发生冲突。此时，推理机必须调用执行某种冲突消解策略的程序，从中选出一条可用规则来执行。

③ 在执行某一条规则时，如果该规则的后件是一个或多个结论，则把这些结论添加到综合数据库中；如果规则的后件是一个或多个操作，则依序执行这些操作。

④ 对于不确定性知识，在执行每一条规则时还要按一定的算法来计算结论的不确定性。

⑤ 随时检查结束推理机运行的条件，在满足结束条件时停止推理机的运行。

可见，问题的求解过程是一个不断地从规则库中选取可用规则与综合数据库中的已知事实进行匹配的过程，规则的每一次成功匹配与执行都使综合数据库增加了新的事实，并向着问题的解前进了一步，这一过程称为推理。

2.2.2 产生式系统的分类及其特点

从不同角度对产生式系统进行分类，可得出不同的分类结果。按产生式所表示的知识是否具有确定性可分为确定性产生式系统和不确定性产生式系统；按推理机的推理方向可分为正向、反向和双向推理产生式系统。这里，仅讨论按规则库及综合数据库的性质与结构特征进行的分类，产生式系统可分为三类：可交换的产生式系统、可分解的产生式系统和可恢复的产生式系统。

1. 可交换的产生式系统

产生式系统求解问题的推理过程是一个反复从规则库中选择合适规则并执行规则的过程。在求解问题的过程中，采用不同的控制策略将会得到不同的规则执行次序，从而反映出不同的求解效率。如果一个产生式系统对规则的使用次序是可交换的，无论先使用哪一条规则都可达到目的，即规则的使用次序对问题的求解是无关紧要的，则称为可交换的产生式系统。

首先来看一个简单的例子。设综合数据库 DB 的初始状态是$\{A, B, C\}$，并设规则库 RB 中有下述规则：

$$\mathrm{R}_1: \text{if } \{A, B, C\} \quad \text{then } \{A, B, C, A \times B\}$$

$$\mathrm{R}_2: \text{if } \{A, B, C\} \quad \text{then } \{A, B, C, B \times C\}$$

$$\mathrm{R}_3: \text{if } \{A, B, C\} \quad \text{then } \{A, B, C, A \times C\}$$

现在希望通过推理使综合数据库 DB 中的内容变为

$$\{A, B, C, A \times B, B \times C, A \times C\}$$

显然，这三条规则各被使用一次后就可达到目的，且与规则使用的次序无关。所以，由上述 RB 和 DB 构成的产生式系统是一个可交换的产生式系统。

若一个产生式系统是可交换的，则指定的 RB 和 DB 都具有如下性质：

① 设 RS 为可应用于 DB 的状态 DB_i 的规则集合，当使用 RS 中任何一条规则 R 使 DB 的状态由 DB_i 改变为 DB_{i+1} 后，该 RS 仍然是 DB_{i+1} 的可用规则集。

② 如果 DB_i 满足目标条件，则当应用 RS 中任何一条规则所生成的 DB 的新状态 DB_{i+1} 仍然满足目标条件。

③ 若对当前状态 DB_i 使用某一规则序列 $\mathrm{R}_1, \mathrm{R}_2, \cdots, \mathrm{R}_k$ 得到 DB 的一个新的状态 DB_{i+k}，即

$$\mathrm{DB}_i \xrightarrow{\mathrm{R}_1} \mathrm{DB}_{i+1} \xrightarrow{\mathrm{R}_2} \cdots \xrightarrow{\mathrm{R}_k} \mathrm{DB}_{i+k}$$

则当改变规则的使用次序后，仍然可得到 DB_{i+k}。

由以上性质可以看出，在可交换产生式系统中，综合数据库 DB 中的内容是递增的，即对任意一个规则执行序列 $\mathrm{R}_1, \mathrm{R}_2, \cdots, \mathrm{R}_k$ 都有

$$\mathrm{DB}_i \subseteq \mathrm{DB}_{i+1} \subseteq \cdots \subseteq \mathrm{DB}_{i+k}$$

由此可见，在可交换产生式系统中，被执行的产生式规则后件总是包含着 DB 当前状态未包含的新的结论，且不包含删除 DB 中已有事实的操作。还可看出，用可交换产生式系统求解

问题时，推理过程不必回溯。所谓回溯是指：当执行一条规则使 DB 的状态由 DB_i 变为 DB_{i+1} 时，如果发现由 DB_{i+1} 不可能得到问题的解，则立即撤消由刚才执行规则所产生的结果，使 DB 恢复到先前的状态 DB_i，然后选用别的规则继续求解。由于求解问题时只需选用任意一个规则序列，而不必搜索多个规则序列，从而节省了问题求解的时空开销，提高了求解的效率。

2. 可分解的产生式系统

把一个规模较大且较复杂的问题分解为若干个规模较小且较简单的子问题，然后对每个子问题分别进行求解，这是人们求解问题时常用到的方法。可分解的产生式系统就是基于这一思想提出来的。

一个产生式系统可分解的要求是，综合数据库 DB 的当前状态 DB_i 可被分解为若干个独立的部分 $DB_i^1, DB_i^2, \cdots, DB_i^m$，且根据 DB 的状态确定的推理过程的终止条件也可被分解为对这些独立部分进行推理的终止条件。例如，设综合数据库 DB 的初始内容为 $DB_0=\{D, B, Z\}$，规则库 RB 中有如下规则：

$$R_1: \text{if } C \text{ then } \{D, L\}$$
$$R_2: \text{if } C \text{ then } \{B, M\}$$
$$R_3: \text{if } B \text{ then } \{M, M\}$$
$$R_4: \text{if } Z \text{ then } \{B, B, M\}$$

终止条件是生成只包含 M 的综合数据库，即使综合数据库的内容变为

$$\{M, M, \cdots, M\}$$

由于规则库 RB 中的每条规则的前件都只含有单一条件 C 或 B 或 Z，因此可把综合数据库 DB 的初始状态 $DB_0=\{C, B, Z\}$ 分解为独立的三个部分：$DB_0^1=\{C\}$，$DB_0^2=\{B\}$，$DB_0^3=\{Z\}$。而且每次对 DB 中的一部分执行一条规则使其状态为 DB_i 后，若 DB_i 中有 m 个事实，则把 DB_i 分成 m 个独立部分：$DB_i^1, DB_i^2, \cdots, DB_i^m$，每一部分都只含有单一事实。若含有的单一事实是 M，则对这一部分状态不再使用规则；否则，继续使用合适规则。其求解过程如图 2.1 所示。

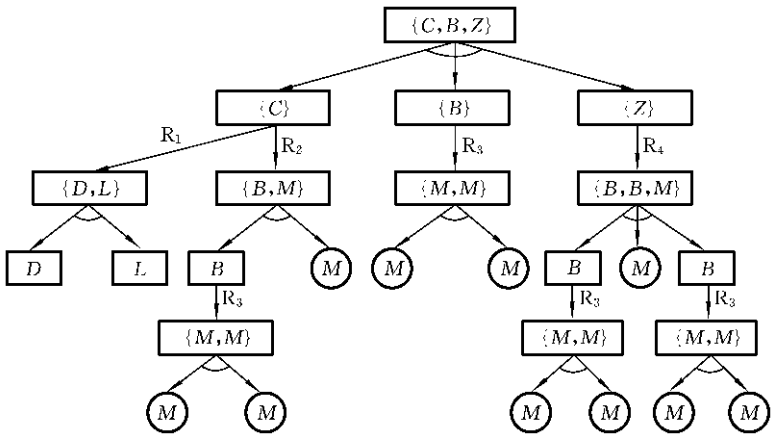

图 2.1　可分解的产生式系统示例

在图 2.1 中，用圆弧连接起来的子节点是“与”关系，不用圆弧连接的子节点是“或”关系。用图表示可分解产生式系统求解问题的过程时，得到的是一棵与/或树。

在可分解产生式系统中，由于综合数据库被分解成若干子库，故随着推理过程的进行，有关的子库又可分解成若干子子库，以此类推，从而减小了在规则库搜索规则和与数据库当前状态进行匹配的工作量，提高了问题求解的效率。

3. 可恢复的产生式系统

在可交换产生式系统中，要求每条规则的执行只能为综合数据库增添新的内容，且不能删除和修改综合数据库中已有的内容。这一要求是很高的，在许多规则的设计中难以达到。因此就需要产生式系统具有回溯功能，一旦问题求解到某一步发现无法继续下去时，就撤消在此之前得到的某些结果，恢复到先前的某个状态，然后选用别的规则继续求解。在问题求解过程中既可以对综合数据库添加新内容，又可删除或修改旧内容的产生式系统称为可恢复的产生式系统。有关可回溯的搜索策略将在后面的章节中详细讨论。

4. 产生式表示方法的优缺点

产生式表示方法主要有以下优点。

(1) 自然性

产生式表示方法用“如果……则……”的形式表示知识，这是人们常用的一种表达事物因果关系的知识表示形式，既直观、自然，又便于进行推理。正是这一原因，使得产生式表示方法成为人工智能中最重要且应用最多的一种知识表示模式。

(2) 模块性

产生式是规则库中最基本的知识单元，规则库与推理机相对独立，而且每条规则都具有相同的形式，这就便于对规则库进行模块化处理，为知识的增、删、改带来方便。

(3) 有效性

产生式表示方法既可表示确定性知识，又可表示不确定性知识；既有利于表示启发式知识，又可方便地表示过程性知识；既可表示领域知识，又可表示元知识。因此，产生式表示方法可以把专家系统中需要的多方面的知识用统一的知识表示模式有效地表示出来。这也是目前已建造成功的专家系统大多采用产生式表示方法的一个重要原因。

(4) 清晰性

产生式有固定的格式，每一条产生式规则都由前提与结论(操作)两部分组成。产生式规则具有所谓自含性，即一条产生式规则仅仅描述该规则的前提与结论之间的静态的因果关系，且所含的知识量都比较少。这就便于对规则进行设计和保证规则的正确性，同时，也便于在知识获取时对规则库进行知识的一致性和完整性检测。

产生式表示方法也有以下不足之处。

(1) 效率不高

在产生式系统问题求解过程中，首先要从规则库中选出可与综合数据库当前状态匹配的可用规则，若可用规则不止一个，就需要按某种冲突消解策略从中选出一条规则来执行。因

此，产生式系统求解问题的过程是一个“匹配—冲突消解—执行”反复进行的过程。由于规则库一般规模较大，拥有的规则条数较多，而规则匹配又是一件十分费时的工作，因此产生式系统的工作效率是不高的。

(2) 不能表达具有结构性的知识

产生式适合于表达具有因果关系的过程性知识，但对具有结构关系的知识却无能为力，它不能把具有结构关系的事物之间的结构联系表示出来。因此，产生式除了可以独立作为一种知识表示模式外，还常与其他可表示知识的结构关系的表示方法结合起来使用。例如，把产生式与语义网络两种表示方法相结合，把产生式与框架表示两种表示方法相结合等。

2.3　框架表示方法

1975 年，美国的人工智能学者明斯基首先提出框架理论，该理论认为人们对现实世界中各种事物的认识都是以一种类似于框架的结构存储在记忆中的，当面临一个新事物时，就从记忆中找出一个合适的框架，并根据实际情况对其细节加以修改、补充，从而形成对当前事物的认识。世界上的各类事物都各自具有不同的属性，不同事物的属性之间往往具有一定的规律性的联系。这种规律性的知识经过提炼，就可以形成人们认识某一类事物的一种固定的框架。框架表示方法就是用来表示这种经验性知识的一种知识表示方法。

2.3.1　框架与框架网络

框架是描述对象(一个事物、一个事件或一个概念)属性的一种数据结构，在框架表示方法中，框架被看成是知识表示的基本单位。不同的框架之间可以通过属性之间关系建立联系，从而构成一个框架网络，充分表达相关对象间的各种关系。

1. 框架

一个框架由若干个被称为“槽”的结构组成，每一个槽又可根据实际需要分为若干个“侧面”。一个槽用于描述对象的某一方面的属性，一个侧面用于描述相应属性的一个方面。槽和侧面所具有的属性值分别称为槽值和侧面值。在一个用框架表示知识的系统中都含有多个框架，需要给它们赋予不同的框架名。同样，对一个框架内的不同槽和不同侧面也需要分别赋予不同的槽名和侧面名。

一个框架可以形式化地表示如下。

＜框架名＞

槽名 1：侧面名 11：侧面值 11
　　侧面名 12：侧面值 12
　　……
　　侧面名 1n：侧面值 1n

⋮

槽名 k：侧面名 $k1$：侧面值 $k1$

侧面名 $k2$：侧面值 $k2$

……

侧面名 km：侧面值 km

下面给出框架的 BNF 描述：

＜框架＞::＝＜框架头＞＜槽部分＞[＜约束部分＞]

＜框架头＞::＝框架名＜框架名的值＞

＜槽部分＞::＝＜槽＞,[＜槽＞]

＜约束部分＞::＝约束＜约束条件＞,[＜约束条件＞]

＜框架名的值＞::＝＜符号名＞|＜符号名＞(＜参数＞,[＜参数＞])

＜槽＞::＝＜槽名＞＜槽值＞|＜侧面部分＞

＜槽名＞::＝＜系统预定义槽名＞|＜用户自定义槽名＞

＜槽值＞::＝＜静态描述＞|＜过程＞|＜谓词＞|＜框架名的值＞|＜空＞

＜侧面部分＞::＝＜侧面＞,[＜侧面＞]

＜侧面＞::＝＜侧面名＞＜侧面值＞

＜侧面名＞::＝＜系统预定义侧面名＞|＜用户自定义侧面名＞

＜侧面值＞::＝＜静态描述＞|＜过程＞|＜谓词＞|＜框架名的值＞|＜空＞

＜静态描述＞::＝＜数值＞|＜字符串＞|＜布尔值＞|＜其他值＞

＜过程＞::＝＜动作＞|＜动作＞,[＜动作＞]

＜参数＞::＝＜符号名＞

对框架的 BNF 有如下几点说明：

① 框架的值允许带有用符号名表示的参数(形参)。当一个框架 A 调用另一个带有形参的框架 B 时，框架 A 需要为框架 B 提供相应形参的实参。

② 当槽值或侧面值是一个过程时，它既可以是一个明确表示出来的＜动作＞串，也可以是对程序语言编制的某个过程的调用，从而可将过程性知识表示出来。

③ 当槽值或侧面值是谓词时，谓词的真值由谓词变元的取值确定。

④ 当槽值或侧面值为＜空＞时，表示该值还不能确定，等待以后填入。

⑤ ＜约束条件＞是任选的，约束条件可以为框架附加上一些说明性的信息，也可以用于指出什么样的值才能填入到槽或侧面中去。当不指出约束条件时，表示没有约束。

2. 框架网络

由于框架中的槽值或侧面值都可以是另一个框架的框架名，这就可在框架之间建立联系，通过一个框架可以找到另一个框架。共处于某种环境中的若干对象必然会有某些共同的属性，在对这些对象进行描述时，可以把它们具有的共同属性抽取出来，构成一个上层框架，然后对各类对象独有的属性分别构成若干个下层框架。为了指明框架之间的这种上下关系，可在下层框架中设立一个专用的槽(一般称之为“继承”槽)，用以指出它的上层框架是哪一个。下

层框架可以继承上层框架的属性及值，这样就可以把一组有上下关系的框架组成具有层次结构的框架网络。通过框架网络，可以避免对相关对象的重复描述，节约了时间和空间的开销。

继承性是框架表示方法的一个重要特性，它不仅可以在相邻的上、下两层框架之间实现继承，而且可以从最低层追溯到最高层，使高层框架的描述信息逐层向低层框架传递。

用下述一个例子来说明框架表示方法的继承性。

例 2.6　建立一个分层的框架网络，框架网络从最高层框架至最低层框架的框架名依序为＜师生员工＞，＜教职工＞，＜教师＞，＜教师 1＞，并为相应框架设置继承槽来避免重复描述。

解　师生员工框架为

框架名：＜师生员工＞
　　姓名：单位（姓，名）
　　年龄：单位（岁）
　　性别：范围（男，女）
　　缺省：男
　　健康状况：范围（健康，一般，差）
　　缺省：一般
　　住房：＜住房＞

教职工框架为

框架名：＜教职工＞
　　继承：＜师生员工＞
　　工作类别：范围（教师，干部，工人）
　　缺省：教师
　　学历：范围（中专，大专，本科，硕士，博士）
　　缺省：本科
　　参加工作时间：单位（年，月）

教师框架为

框架名：＜教师＞
　　继承：＜教职工＞
　　部门：单位（系，教研室）
　　语种：范围（英语，法语，德语，日语，俄语）
　　缺省：英语
　　职称：范围（教授，副教授，讲师，助教）
　　缺省：讲师

某个教师的实例框架为

框架名：＜教师 1＞
　　继承：＜教师＞
　　姓名：王林

年龄:36
健康状况:健康
参加工作时间:1982.9
部门:计算机系软件教研室
职称:副教授

由上述框架描述可以看出如下几点。

① 在框架网络中,既有用“继承”槽指出上、下层框架之间的纵向联系,也有以框架名作为槽值指出的框架之间的横向联系。例如,在<师生员工>框架中,对槽名为“住房”的槽填入的槽值是一个框架名<住房>,从而在<师生员工>框架与<住房>框架之间建立了横向联系的框架体系结构。

② 实例框架中的每一个槽都应给出槽值,并可以继承上层框架槽的槽值,从而获得实例框架中没有直接给出的知识。例如,在实例框架<教师 1>中,虽然没有给出“性别”、“工作类别”、“学历”、“语种”槽及其槽值,但由继承性可得知:王林的性别是“男”,工作类别是“教师”,学历是“本科”,语种是“英语”。

③ 以框架作为知识表示模式时,知识是通过事物的属性表示的。为满足问题求解的需要,就要求框架中有足够的槽把事物有关方面的属性充分表达出来。但是,一个事物的属性通常都是多方面的,在选择把哪些属性作为一个框架的槽的描述对象时,要根据系统的设计目标和属性之间的联系对事物的属性进行筛选和分类,仅需要对有关的属性设立槽,不可面面俱到,以免浪费空间和降低系统的运行效率。

在框架系统中,事物之间的联系是通过在槽中填入相应的框架名来实现的,至于它们之间的关系可以由槽名来指明。在框架表示方法中,对于一些常用且可公用的槽名给出了标准槽名及其定义,称这些槽名为系统预定义槽名。

2.3.2 框架推理及其特点

用框架表示知识的系统主要由两大部分组成:一是由框架网络构成的知识库,二是由一组程序组成的框架推理机。前者的作用是提供求解问题所需要的知识,后者的作用是针对用户提的问题,运用知识库中的知识完成问题求解。

1. 框架推理的基本过程

在用框架表示知识的系统中,推理主要是通过框架匹配与填槽来实现的。首先把要求解的问题用一个称为问题框架的框架表示出来,然后把初始问题框架与知识库中已有的框架进行匹配。框架的匹配是把两个框架相应的槽名及槽值逐个进行比较,如果两个框架的各对应槽没有矛盾或满足预先规定的某些条件,就认为这两个框架可以匹配。由于框架之间存在继承关系,一个框架所描述的某些属性及值可能是从它的上层框架继承过来的,因此两个框架的比较往往要牵涉到它们的上层、上上层框架。

我们来看一个例子。假设例 2.6 提出的关于师生员工的框架网络已建立在知识库中,当

前要求解的问题是从知识库中找出一个满足如下条件的教师：

男性，年龄在 40 岁以下的副教授，身体健康，会英语

则可把求解问题表示成如下初始问题框架。

框架名：＜教师 x＞

姓名：

年龄：＜40

性别：男

健康状况：健康

职称：副教授

语种：英语

在初始问题框架中，姓名槽的槽值是空的，反映了需要求解的问题，有待推理过程中填入合适的槽值，从而求出问题的解。

用此问题框架与知识库中的框架匹配，显然＜教师 1＞框架可以与之匹配，因为“年龄”槽、“健康状况”槽与“职称”槽的槽值都符合要求，＜教师 1＞框架虽然没有给出“性别”槽与“语种”槽的槽值，但由继承性可知＜教师 1＞框架继承有“性别”槽，且槽值是“男”，还继承有“语种”槽，且槽值是“英语”。由框架匹配可知要找的教师可能就是王林。这里之所以说“可能”，是因为知识库中可与问题框架＜教师 x＞匹配成功的框架可能不止一个，因而目前匹配成功的框架还只能作为预选框架，需要进一步收集信息，以便从中选出一个，或者根据框架中其他槽的内容及框架间的关系明确下一步查找的方向和线索。

2. 框架推理

在用框架表示知识的系统中，构成的框架网络是一个层次结构。框架推理是以此层次结构为基础，按照一定的搜索策略，不断寻找可匹配的框架并进行填槽的过程。在此过程中，有可能找到了合适的框架，得到了问题的解而成功结束，也可能因找不到合适框架而被迫终止。这里，按深度优先搜索策略讨论自顶向下进行搜索的框架推理过程。框架推理的步骤如下。

① 把用户要求解的问题形成一个初始问题框架，并将已知的知识（一般为该问题中一些已知的事实或数据）填入相应槽中，在此框架中，有些槽是空的，它反映了需要解决的问题，有待推理过程填入合适的值。槽的排列顺序可以与知识库中相应框架中的槽的顺序不相同，但一般要求问题框架中的槽名（特别是关键性的槽名）应包含在知识库的相应框架中。

② 把框架网络的根框架作为当前框架，即以根框架作为搜索推理的起点。

③ 把问题框架与当前框架进行匹配，即对两个框架除空槽以外的相应槽逐个地进行确定性匹配或不确定性匹配。如果两个框架能够满足确定性匹配或不确定性匹配的条件，则转步骤④进行填槽；否则转步骤⑤搜索下一个框架。在对两个框架进行匹配的过程中，允许当前框架的某些槽及槽值是从其上层框架继承的。

④ 把当前框架中相应槽的槽值填入问题框架的对应空槽中，并可把当前框架从其上层框架继承的槽值或者通过与用户交互得到的槽值填入问题框架的对应空槽中（填槽的过程类似

于程序设计语言中对变量进行赋值的过程）。在填槽后，检查问题框架是否已经包含了问题的解，即反映求解问题的空槽是否已填入槽值。若已包含，则转步骤⑦；否则转步骤⑤。

⑤ 按当前框架的 Instance 槽的槽值找一个尚未进行过匹配操作的子框架。若有这样的子框架，则把该子框架作为当前框架，转步骤③进行匹配及填槽操作，此时，原已填入到问题框架中的槽值有可能被当前新的填槽操作所修改。若没有这样的子框架，则转步骤⑥进行回溯。Instance 是框架中常用的系统预定义槽名，其槽值是该框架的子框架的框架名。

⑥ 由当前框架的 AKO 槽的槽值找到它的父框架。如果该父框架不是根框架，则把该父框架作为当前框架，并转步骤⑤。如果该父框架是根框架且有未进行匹配操作的子框架，则把该根框架作为当前框架，并转步骤⑤；否则，表示由该父框架为根框架的框架网络已不可能求得问题的解，需要另外再选一个根框架重复进行上述推理过程，如果已没有合适的根框架可选，则表示问题无解，结束推理过程。AKO 是框架中常用的系统预定义槽名，其槽值是该框架的父框架的框架名。

⑦ 如果问题的解具有不确定性，则根据采用的不确定性知识表示方法计算解的不确定性度量，成功地结束推理过程。

上述过程实际上是沿 Instance 槽的自顶向下搜索和沿 AKO 槽的回溯组成的深度优先搜索，这只是框架推理的搜索策略之一。有关搜索策略的问题将在后面的章节中详细讨论。

由于槽值可以是满足指定条件时可触发执行的一个动作或过程，因此，在推理过程中，由于执行了某些动作或过程可能会增加或修改框架中已有的知识。

3. 框架表示方法的特点

框架表示方法有以下主要特点。

(1) 结构性

框架表示方法最突出的特点是它善于表示结构性知识，能够把知识的内部结构关系及知识之间的联系表示出来，因此它是一种经过组织的结构化的知识表示方法。这一特点是产生式表示方法所不具备的，产生式系统中的知识的组织单位是产生式规则，这种知识组织单位由于太小而难于处理复杂问题，也不能把知识之间的结构关系显式地表示出来。框架表示方法的知识组织单位是框架，而框架由若干槽组成，槽又由若干侧面组成，这就可把知识的内部结构显式地表示出来。另外，产生式规则只能表示事物间的因果关系，而框架表示方法不仅可以通过 Infer 槽或 Possible-Reason 槽表示事物间的因果关系，而且可以通过其他槽表示出事物之间更复杂的联系。

(2) 继承性

继承在知识表示方法中支持概念抽象和信息共享等思想，在框架系统中起到极其重要的作用。继承性不仅减少了框架网络表示知识的冗余，而且较好地保证了知识的一致性。需要指出的是：由于子框架可以继承父框架的槽值，也可以补充和修改，因此多重继承有可能产生属性描述的多义性。如何解决继承过程中属性的歧义，目前还没有一种统一的方法。

框架表示方法的主要不足之处是不善于表示过程性的知识，因此，可以把框架表示方法与产生式表示方法结合起来使用，以取得互补的效果。另外，对于给定的问题领域，如果有关的

领域知识具有比较明显的结构性特点，就比较容易建立框架表示的知识库；否则，用框架网络来形式化领域知识就不是一件简单的事情。

习　题　二

2.1　名词解释：

命题	谓词
个体	个体域
谓词公式的解释	谓词公式的永真性
谓词公式的可满足性	谓词公式的不可满足性
谓词公式的永真蕴含	谓词公式的等价性

2.2　简述谓词逻辑中的下述推理规则：

(1) P 规则；

(2) T 规则；

(3) CP 规则；

(4) 反证法规则。

2.3　一阶谓词逻辑表示方法适合于表示哪种类型的知识？它有哪些主要特点？

2.4　请用相应的谓词公式表示下述语句：

(1) 有的人喜欢足球，有的人喜欢排球，有的人既喜欢足球又喜欢排球。

(2) 不是每一个人都喜欢游泳。

(3) 如果没有利息，就没有人去储蓄。

(4) 对于所有的 x 和 y，如果 x 是 y 的父亲，y 是 z 的父亲，那么 x 是 z 的祖父。

(5) 对于所有的 x 和 y，如果 x 是 y 的孩子，那么 y 是 x 的父母。

(6) 如果 $b>a>0$ 和 $c>d>0$，则有 $(b\times(a+c)/d)>b$。

2.5　产生式与谓词逻辑中的蕴含式有何异同点？

2.6　简述产生式系统问题求解的一般步骤。

2.7　何谓可交换产生式系统？可交换产生式系统的规则库与数据库应具有什么性质？

2.8　何谓可恢复产生式系统？为什么可恢复产生式系统应具有回溯功能？

2.9　简述产生式表示方法的主要优缺点。

2.10　有下述猴子摘香蕉问题：在一个房间的地板上有一只猴子和一个箱子，天花板下挂有一串香蕉，猴子的水平位置为 a，箱子的水平位置为 b，香蕉的水平位置为 c，猴子只有爬到箱子上才能摘到香蕉。猴子被允许有以下 4 种行为：在地板上行走，在地板上推动箱子，爬到箱子上，摘香蕉。为了规划出猴子摘香蕉的行为方案，请把猴子的 4 种行为表示成 4 条产生式规则。

2.11　有下述传教士与野人问题：有 3 名传教士和 3 名野人来到一条河的右岸，欲乘一条船渡河到左岸，该船的最大负载能力为 2 人，传教士或野人皆可撑船。在任何时候，不论是在右岸还是在左岸，如果野人人数超过传教士人数，那么，野人就会吃掉传教士。为了规划出一个渡河方案，把 6 个人都安全地渡过河去，请应用产生式表示方法表示求解该问题的所有规则。

2.12　有下述农夫过河问题：农夫、狐狸、山羊和白菜都在一条河的左岸，只有农夫可撑船，要将狐狸、山羊和白菜都渡河到右岸去。但是，农夫每次撑船过河时，船上至多只能载狐狸，或者载山羊，或者载白菜。若农夫不在时，则狐狸会吃掉山羊、山羊会吃掉白菜。为了规划出一个渡河方案，使农夫能把狐狸、山羊和白菜

都带过河去，请应用产生式表示方法表示求解该问题的所有规则。

2.13 何谓框架知识表示？给出框架的一般表示形式。

2.14 简述框架系统的推理过程。

2.15 简述框架表示方法的主要特点。

2.16 试建立一个“学生”框架网络，其中，至少有“学生基本情况”、“学生课程学习情况”和“学生奖惩情况”三个框架描述。

第 3 章

搜 索 方 法

研究人工智能技术的一个主要目的是求解非平凡问题，即难以用常规技术（数值计算、数据库应用等）直接求解的问题。这些问题的求解依赖于问题本身的描述和特定领域相应知识的表示与应用。根据问题求解可使用的领域知识的多寡，问题求解系统可以划分为两大类：知识贫乏系统和知识丰富系统。前者必须使用搜索技术求解，后者则依靠推理技术求解。

搜索技术适合于可设计一个操作算子（算符）集的问题的求解，每个操作算子的执行均能更接近问题的解。问题的解将由搜索过程中实际选用的操作算子的序列组成。

搜索方法分为盲目搜索方法和启发式搜索方法。盲目搜索方法是按预定的搜索方向进行搜索。由于盲目搜索总是按预先规定的方向进行，没有考虑到问题本身的特性，所以这种搜索方法效率不高。启发式搜索方法是在搜索中加入了与问题有关的启发性知识，用以指导搜索朝着最有希望的方向前进，加快问题的求解速度。显然，启发式搜索优于盲目搜索，但由于启发式搜索需要具有与问题本身特性有关的知识，并非对每一类问题都可方便地抽取出来，因此盲目搜索仍不失为一种应用较多的搜索方法。

3.1 问题求解过程的形式表示

应用搜索技术求解问题的过程实际上是一个搜索过程。在应用各种搜索方法进行搜索时，首先必须考虑问题及其求解过程的形式表示是否适当，因为这将直接影响在计算机上编程求解的方便程度以及求解问题的效率。

3.1.1 状态空间表示方法

状态空间表示方法是表示问题及其搜索过程的一种形式表示方法。状态空间表示方法用“状态”和“算符”来表示问题及求解问题过程中可使用的知识。

状态是求解过程中用以描述问题在任一时刻状况的数据结构，需要根据求解问题的特征来定义问题的状态。常用于定义状态的数据结构有：字符串、一维数组、二维数组等。

算符表示对状态的操作，算符的每一次使用都使问题由一种状态变换为另一种状态。当到达目标状态时，由初始状态到目标状态所用算符的序列就是问题的一个解。在产生式系统中，每一条产生式规则就是一个算符。

由问题的全部状态及一切可用算符所构成的集合称为问题的状态空间，一般用一个三元组表示：

$$(S,F,G)$$

其中，S 是问题的所有初始状态构成的集合；F 是算符的集合；G 是目标状态的集合。

状态空间的图示形式称为状态空间图。其中，节点表示状态；有向边(弧)表示算符。

例 3.1 二阶梵塔问题。设有三根柱子，在 1 号柱子上穿有 A、B 两个盘片，盘 A 小于盘 B，盘 A 位于盘 B 的上面。要求把这两个盘片全部移到另一根柱子上，而且规定每次只能移动一片，任何时刻都不能使盘 B 位于盘 A 的上面。画出二阶梵塔问题的状态空间有向图，并给出问题的解。

解 设用 $S=(x_A,x_B)$ 表示问题的状态，x_A 表示盘 A 所在的柱号，x_B 表示盘 B 所在的柱号。全部可能的状态有以下 9 种：

$$S_0=(1,1),\quad S_1=(1,2),\quad S_2=(1,3)$$
$$S_3=(2,1),\quad S_4=(2,2),\quad S_5=(2,3)$$
$$S_6=(3,1),\quad S_7=(3,2),\quad S_8=(3,3)$$

问题的初始状态集合 $S=\{S_0\}$，目标状态集合 $G=\{S_4,S_8\}$。

算符分别用 $A(i,j)$ 及 $B(i,j)$ 表示。$A(i,j)$ 表示把盘 A 从柱 i 移到柱 j 上；$B(i,j)$ 表示把盘 B 从柱 i 移到柱 j 上。共有 12 个算符，它们分别是

$$A(1,2),\quad A(1,3),\quad A(2,1),\quad A(2,3),\quad A(3,1),\quad A(3,2)$$
$$B(1,2),\quad B(1,3),\quad B(2,1),\quad B(2,3),\quad B(3,1),\quad B(3,2)$$

根据 9 种可能的状态和 12 种算符，可构成二阶梵塔问题的状态空间图，如图 3.1 所示。

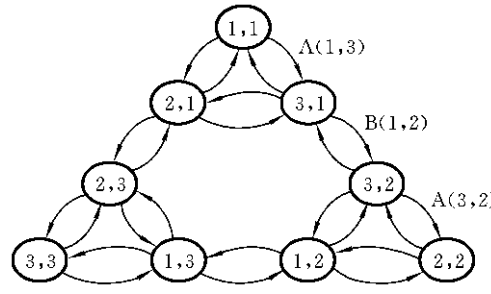

图 3.1 二阶梵塔的状态空间

在图 3.1 所示的状态空间中，从初始节点(1,1)到目标节点(2,2)或者(3,3)的任何一条路径都是问题的一个解，其中最短的路径长度是 3，它由 3 个算符组成，例如 A(1,3)，B(1,2)，A(3,2)。

由此例可以看出：

① 在用状态空间表示方法表示问题时，首先必须定义状态的描述形式，通过使用这种描述形式可把问题的一切状态都表示出来；其次还要定义一组算符，通过使用算符可把问题由一种状态转变为另一种状态。

② 问题的求解过程是一个不断把算符作用于状态的过程。如果在使用某个算符后得到的新状态是目标状态，就得到了问题的一个解。这个解是从初始状态到目标状态所用算符构成的序列。

③ 算符的一次使用，就使问题由一种状态转变为另一种状态。可能有多个算符序列都可使问题从初始状态变到目标状态，也就是问题可以有多个解。有的解使用算符较少，有的较多，称使用算符最少的解为最优解。例如，在上例中使用 3 个算符的解是最优解。这只是从使用算符个数来评价解的优劣，更一般地说是使用算符时所付出的代价，只有总代价最小的解才是最优解。

④ 对任何一个状态，可使用的算符可能不止一个，若有两个或两个以上的可用算符，则称

为发生可用算符冲突，在编程实现问题求解程序时，需要考虑采用何种冲突消解策略来选用可用算符。常用的一种冲突消解策略是按算符在算符集中的排序选用可用算符。

⑤ 由一个状态使用多个可用算符所生成的后继状态有多个。当对这些后继状态使用算符时，首先应对哪一个后继状态进行操作则取决于问题求解采用的搜索策略。搜索策略将影响问题搜索求解过程的效率。

3.1.2　与/或图表示方法

与/或图是用于表示问题及其求解过程的又一种形式化方法，也称为问题归约方法。它把初始问题通过一系列变换最终变为由若干子问题组成的集合，而这些子问题的解可以直接得到，从而解答了初始问题。

1. 分解变换

把一个复杂问题分解为若干个较为简单的子问题，每个子问题又可继续分解为若干个更为简单的子问题，重复此过程，直到不需要再分解或者不能再分解为止；然后对每个子问题分别进行求解；最后把各子问题的解复合起来就得到了原问题的解。

问题的分解过程可用一个图表示出来。例如，把问题 P 分解为三个子问题 P_1、P_2、P_3，可用图 3.2 表示。P_1、P_2、P_3 是问题 P 的三个子问题，只有当这三个子问题都可解时，问题 P 才可解，称 P_1、P_2、P_3 之间存在“与关系”；称节点 P 为“与节点”；由 P、P_1、P_2、P_3 所构成的图称为“与树”。在图中，为了标明某个节点是与节点，通常用一条弧把与节点连接其与关系的子节点的各条边连接起来。

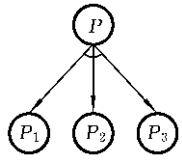

图 3.2　与树

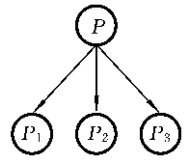

图 3.3　或树

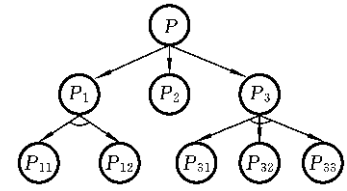

图 3.4　与/或树

2. 等价变换

对于一个复杂问题，除了可用分解变换进行求解外，还可利用等价变换把它变换为若干个较容易求解的新问题，若新问题中有一个可求解，则得到了原问题的解。

问题的等价变换过程也可用一个图表示出来，称为“或”树。例如，可用图 3.3 表示问题 P 被等价变换为新问题 P_1、P_2、P_3。其中，新问题 P_1、P_2、P_3 中只要有一个可解，则原问题就可解，称 P_1、P_2、P_3 之间存在“或”关系；节点 P 称为“或”节点，由 P、P_1、P_2、P_3 所构成的图是一个“或”树。

上述两种方法也可结合起来使用，此时的图称为与/或图。其中，既有与节点，也有或节点，如图 3.4 所示。

3. 相关概念

(1) 本原问题

不能再分解变换或等价变换，而且直接可解的子问题称为本原问题。

(2) 端节点与终叶节点

在与/或图中，没有子节点的节点称为端节点；本原问题所对应的节点称为终叶节点。显然，终叶节点一定是端节点，但端节点不一定是终叶节点。

(3) 可解节点

在与/或图中，满足下列条件之一者，称为可解节点：

① 它是一个终叶节点。

② 它是一个或节点，且其子节点至少有一个是可解节点。

③ 它是一个与节点，且其子节点全部是可解节点。

(4) 不可解节点

在与/或图中满足下列条件之一者，称为不可解节点：

① 它是一个非终叶节点的端节点，即无法变换的非终叶节点。

② 它是一个或节点，且其子节点全部是不可解节点。

③ 它是一个与节点，且其子节点至少有一个是不可解节点。

(5) 解树

由可解节点构成的、且由这些可解节点可推出初始节点(它对应于原始问题)为可解节点的与/或树称为解树。在解树中一定包含初始节点。

例 3.2　三阶梵塔问题。设有 A、B、C 三个盘片以及三根柱子，三个盘片按从小到大的顺序穿在 1 号柱上，要求把它们全部移到 3 号柱上，而且每次只能移动一个盘片，任何时刻都不能把大的盘片压在小的盘片上面，如图 3.5 所示。应用分解变换求解三阶梵塔问题。

图 3.5　三阶梵塔问题

(a) 初始状态；(b) 目标状态

解　为了求解该问题，可进行下述分析：

① 为了把三个盘片全部移到 3 号柱上，必须先把盘片 C 移到 3 号柱上。

② 为了移盘片 C，必须先把盘片 A 及 B 移到 2 号柱上。

③ 当把盘片 C 移到 3 号柱上后，就可把盘片 A、B 从 2 号柱移到 3 号柱上，这样就可完成问题的求解。

由以上分析，可把原问题分解为下述三个子问题：

① 把盘片 A 及 B 移到 2 号柱的双盘片问题。

② 把盘片 C 移到 3 号柱的单盘片问题。

③ 把盘片 A 及 B 移到 3 号柱的双盘片问题。

其中，子问题①与子问题③又分别可分解为三个子问题。

为了用与/或树把问题的分解过程表示出来，需要先定义问题的状态表示：

$$S=(x_C,x_B,x_A)$$

其中，x_C表示盘片 C 所在的柱号；x_B表示盘片 B 所在的柱号；x_A表示盘片 A 所在的柱号。这样初始问题就可表示为

$$(1,1,1)\Rightarrow(3,3,3)$$

算符分别用 A(i,j)，B(i,j)和 C(i,j)表示，分别表示把盘片 A，B，C 从柱 i 移到柱 j 上。

可用与/或树把分解过程表示出来，如图 3.6 所示。

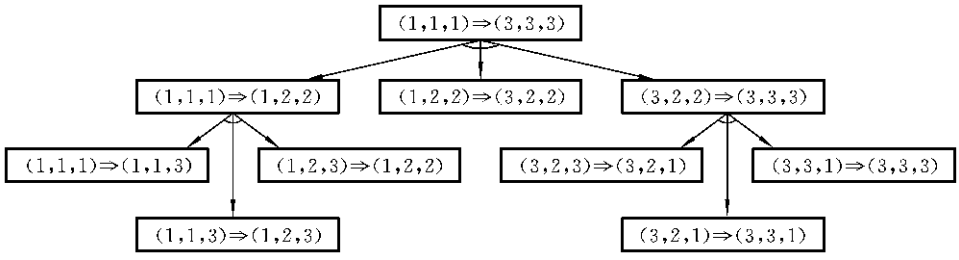

图 3.6　三阶梵塔问题的与/或树

在与/或树中，通过“分解”得到 7 个终叶节点，对应于 7 个本原问题。本原问题从左至右排列分别是

$$(1,1,1)\Rightarrow(1,1,3),(1,1,3)\Rightarrow(1,2,3),(1,2,3)\Rightarrow(1,2,2)$$
$$(1,2,2)\Rightarrow(3,2,2),(3,2,2)\Rightarrow(3,2,1),(3,2,1)\Rightarrow(3,3,1)$$
$$(3,3,1)\Rightarrow(3,3,3)$$

把这些本原问题的解从左至右排列，就得到了初始问题的解：

$$A(1,3),B(1,2),A(3,2),C(1,3),A(2,1),B(2,3),A(1,3)$$

实际上，可以把状态空间图看成是与/或图的特例，仅由等价变换算符形成的状态空间图的所有节点都是或节点，因此，状态空间图也可称为或图。如果既有等价变换算符又有分解算符，那么将形成与/或图，图中既有与节点，也有或节点。

3.2　状态空间的搜索方法

状态空间搜索算法的任务是：从初始节点出发，使用提供的可用算符（规则），在状态空间中搜索出一条解路径，从而得出问题的解。搜索的结果是获得一棵搜索树。如果搜索算法成功终止，那么，搜索树一定包含从初始节点到目标节点的解路径。搜索树是由称为 open 表和 closed 表的两个表共同记载的。open 表与 closed 表的数据结构是链表，它们的作用是

① open 表记载搜索过程中尚未考查的节点和新生成的节点；

② open 表支持按指定要求对表中节点排序；

③ 搜索算法每次循环时都将 open 表中的第一个节点移送到 closed 表的表尾；

④ closed 表记载搜索过程中已被考查过的节点。考查是指：将节点记录的状态与目标状态进行比较，以确定该节点是否是目标节点。如果不是目标节点，则在算符集中搜索可用算符，作用于该节点状态，生成其子节点状态；如果在算符集中找不到可用算符，则该节点称为不可扩展。

状态空间的搜索算法可以分为盲目搜索算法和启发式搜索算法两类。

3.2.1 盲目搜索算法

状态空间盲目搜索算法是指：按算法预定的搜索方向，在状态空间里搜索目标节点，生成搜索树。根据算法预定的搜索方向，盲目搜索算法可分为宽度优先搜索算法和深度优先搜索算法两种。根据每次搜索所使用的算符的代价是否相同，盲目搜索算法可分为无代价的盲目搜索算法和有代价的盲目搜索算法两种。

1. 无代价的宽度优先搜索算法

如果问题求解定义的所有算符的使用代价都相同或都定义为单位 1，那么，属于无代价盲目搜索。宽度优先搜索是指：在搜索树的生成过程中，只有对搜索树中同一层的所有节点都考查完之后，才会对下一层的节点进行考查。或者说，宽度优先搜索预定的搜索方向是沿搜索树的宽度方向展开的。

无代价的宽度优先搜索的 open 表和 closed 表的节点的域可如下定义：

j	S_j	F_k	i

其中，j 为节点 j 的序号（通常按节点生成的顺序编号），S_j 为节点 j 的状态，F_k 为生成节点 j 所使用的算符编号（算符名称），i 为节点 j 的父节点序号（或是节点 j 指向父节点 i 的指针）。

无代价宽度优先搜索算法

（1）初始化 open 表与 closed 表，把初始节点（序号 1）放入 open 表中。

（2）若 open＝( )，则算法失败终止；否则，转步骤（3）。

（3）把 open 表的第 1 个节点 i 移出，放入 closed 表的表尾。

（4）若节点 i 的状态 S_i 是目标状态，则算法成功终止；否则，转步骤（5）。

（5）若节点 i 不可扩展（即在算符集中找不到可用算符），则转步骤（2）；否则，转步骤（6）。

（6）逐一用可用算符扩展节点 i，生成 i 的所有子节点。若子节点 j 的状态 S_j 既不在 open 表中，也不在 closed 表中，则节点 j 是一个新节点。将所有的新子节点逐一放入 open 表的表尾，并记载子节点各个域的值，转步骤（2）；否则，不把新生成的节点 j 放入 open 表中，放弃节点 j，转步骤（2）。

状态空间是一个有向图，任何一个节点都可能有 2 个或 2 个以上的父节点。搜索算法找到问题的解路径是搜索树的一条路径，搜索树中的任何节点都只有唯一的父节点。因此，在步

骤(6)中，需要对可用算符生成的子节点确认是否是一个新节点，只有是新节点才能放入到 open 表中。

2. 无代价的深度优先搜索算法

深度优先搜索是指：在搜索树的生成过程中，对 open 表中同一层的节点每次只选择表中一个节点进行考查和扩展，只有当这个节点是不可扩展的才选择同层的兄弟节点进行考查和扩展。或者说，深度优先搜索预定的搜索方向是沿搜索树的深度方向展开的。

无代价的深度优先搜索的 open 表和 closed 表的节点域可为如下定义：

j	S_j	F_k	d_j	i

其中，d_j 是节点 j 的深度，$d_j=d_i+1$，其他域的定义与宽度优先搜索的节点域定义相同。

无代价深度优先搜索算法

(1) 初始化 open 表与 closed 表，把初始节点(序号 1)放入 open 表中。设置搜索最大深度 $d_{\max}$ 的值。

(2) 若 open=( )，则算法失败终止；否则，转步骤(3)。

(3) 把 open 表的第 1 个节点 i 移出，放入 closed 表的表尾。

(4) 若节点 i 的状态 S_i 是目标状态，则算法成功终止；否则，转步骤(5)。

(5) 若节点 i 不可扩展(即在算符集中找不到可用算符)，则转步骤(2)；否则，转步骤(6)。

(6) 逐一用可用算符扩展节点 i，生成 i 的所有子节点。若子节点 j 的状态 S_j 既不在 open 表中，又不在 closed 表中，且 $d_j \leqslant d_{\max}$，则节点 j 是一个允许的新节点。将所有的新子节点按节点序号从小到大依序放入 open 表的表首，并记载子节点各域的值。否则，不把新节点 j 放入 open 表中(放弃节点 j)，转步骤(2)。

比较宽度优先搜索与深度优先搜索，两者不同之处在于：

① 宽度优先搜索生成的子节点放入 open 表的表尾，深度优先搜索生成的子节点放入 open 表的表首。或者说，宽度优先搜索中的 open 表是先进先出的表，深度优先搜索中的 open 表是先进后出的表。由此可见，对 open 表的节点采取不同的排序方法，搜索树的扩展方向就不同。

② 如果问题有解，那么宽度优先搜索总能找到路径最短的解路径，组成这条解路径的算符序列称为最优解。能找到最优解的搜索称为完备性搜索。深度优先搜索是非完备的。

③ 如果搜索最大深度 $d_{\max}$ 设置合理，即目标节点的深度 $d_g \leqslant d_{\max}$，那么深度优先搜索能找到一条解路径，但是不一定是最优解的解路径。此时，一般而言，深度优先搜索成功终止时，生成的搜索树的规模(节点数量)小于宽度优先搜索生成的搜索树的规模。或者说，深度优先搜索的时空开销小于宽度优先搜索。

3. 有代价的宽度优先搜索算法

如果问题求解定义的算符的代价是不相同的，那么，属于有代价的搜索。有代价的搜索算法可以分为有代价的盲目搜索和启发式搜索两类。有代价的盲目搜索包括有代价的宽度优先搜索和有代价的深度优先搜索两种。

如果节点 i 使用算符 F_k 生成其子节点 j，算符 F_k 的使用代价记为 $c(i,j)$，那么，定义节点 j 的代价函数为

$$g_j = g_i + c(i,j)$$

有代价的宽度优先搜索是在无代价的宽度优先搜索基础上改进得到的。open 表和 closed 表的节点域可定义如下：

j	S_j	F_k	g_j	i

其中，g_j 为节点 j 的代价函数值，其他域的定义与前面的节点域定义相同。

有代价宽度优先搜索算法

(1) 初始化 open 表与 closed 表，把初始节点(序号 1)放入 open 表中，且令 $g_1=0$。

(2) 若 open=( )，则算法失败终止；否则，转步骤(3)。

(3) 把 open 表的第 1 个节点 i 移出，放入 closed 表的表尾。

(4) 若节点 i 的状态 S_i 是目标状态，则算法成功终止；否则，转步骤(5)。

(5) 若节点 i 不可扩展(即在算符集中找不到可用算符)，则转步骤(2)；否则，转步骤(6)。

(6) 逐一用可用算符扩展节点 i，生成 i 的所有子节点，并计算所有子节点的代价值。

① 若子节点 j 的状态 S_j 既不在 open 表中，也不在 closed 表中，则节点 j 是一个新节点。将所有新子节点放入 open 表中，记载子节点各域的值，转步骤(7)。

② 若子节点 j 的状态 S_j 已在 open 表中，即节点 j 与 open 表中的一个老节点 j' 的状态相同，则比较 g_j 与 $g_{j'}$。若 $g_j \geqslant g_{j'}$，则放弃新节点 j；若 $g_j < g_{j'}$，则将新节点 j 放入到 open 表中，记载节点各域的值，并删除 open 表中的老节点 j'。转步骤(7)。

③ 若子节点 j 的状态 S_j 已在 closed 表中，即节点 j 与 closed 表中的一个老节点 j' 的状态相同，则比较 g_j 与 $g_{j'}$。若 $g_j \geqslant g_{j'}$，则放弃新节点 j；若 $g_j < g_{j'}$，则将新节点 j 放入到 open 表中，记载节点各域的值，并删除 closed 表中的老节点 j' 以及节点 j' 在 open 表和 closed 表中的所有后裔节点。转步骤(7)。

(7) 对 open 表中的所有节点按 g 值从小到大的顺序重新排序，若有 2 个或 2 个以上的节点有相同的 g 值，则对这些节点再按节点序号排序。转步骤(2)。

有代价宽度优先搜索是在无代价宽度优先搜索的基础上改进得出的，两者的异同点主要如下。

① 无代价宽度优先搜索只能用于无代价问题的求解，有代价宽度优先搜索既可以用于有代价问题的求解，也可以用于无代价问题的求解，此时，由于所有的算符代价为 $c(i,j)=1$，那么代价函数

$$g_j = g_i + c(i,j) = g_i + 1$$

由于初始节点有 $g_1=d_1=0$，故

$$g_j = d_i + 1$$

即任何一个节点的代价值都是该节点的深度。应用有代价宽度优先搜索的结果与无代价宽度优先搜索的结果相同。

② 有代价宽度优先搜索与无代价宽度优先搜索都是完备性搜索。有代价宽度优先搜索能找到路径代价最小的解路径，从而得到最优解。

③ 无代价宽度优先搜索的 open 表按节点序号从小到大对所有节点排序，有代价宽度优先搜索的 open 表按节点代价值从小到大对所有节点排序。

④ 在有代价宽度优先搜索步骤(6)中，若扩展生成的子节点 j 已在 open 表中，且 $g_j<g_{j'}$，则删除 open 表中的老节点 j'；若子节点 j 已在 closed 表中，且 $g_j<g_{j'}$，则不仅删除 closed 表中的老节点 j'，还要删除老节点 j' 可能在 open 表和 closed 表中的所有后裔节点。其原因是：一是保证成功终止时生成一棵搜索树；二是终止沿代价较高的可能解路径继续搜索，以保证搜索的完备性和节省搜索的时空开销。

4. 有代价的深度优先搜索算法

有代价的深度优先搜索是在无代价的深度优先搜索基础上改进得到的。open 表和 closed 表的节点域可定义如下：

j	S_j	F_k	g_j	i

域的定义与前面的节点域定义相同。

有代价深度优先搜索算法

(1) 初始化 open 表与 closed 表，把初始节点(序号 1)放入 open 表中，且令 $g_1=0$。

(2) 若 open=( )，则算法失败终止；否则，转步骤(3)。

(3) 把 open 表的第 1 个节点 i 移出，放入 closed 表的表尾。

(4) 若节点 i 的状态 S_i 是目标状态，则算法成功终止；否则，转步骤(5)。

(5) 若节点 i 不可扩展(即在算符集中找不到可用算符)，则转步骤(2)；否则，转步骤(6)。

(6) 逐一用可用算符扩展节点 i，生成 i 的所有子节点，并计算所有子节点的代价值。

① 若子节点 j 的状态 S_j 既不在 open 表中，也不在 closed 表中，则节点 j 是一个新节点。将所有新子节点放入 open 表中，记载子节点各域的值。转步骤(7)。

② 若子节点 j 的状态 S_j 已在 open 表中，即节点 j 与 open 表中的一个老节点 j' 的状态相同，则比较 g_j 与 $g_{j'}$。若 $g_j\geqslant g_{j'}$，则放弃新节点 j；若 $g_j<g_{j'}$，则将新节点 j 放入到 open 表中，记载节点各域的值，并删除 open 表中的老节点 j'。转步骤(7)。

③ 若子节点 j 的状态 S_j 已在 closed 表中的一个，即节点 j 与 closed 表中的一个老节点 j' 的状态相同，则比较 g_j 与 $g_{j'}$。若 $g_j\geqslant g_{j'}$，则放弃新节点 j；若 $g_j<g_{j'}$，则将新节点 j 放入到 open 表中，记载节点各域的值，并删除 closed 表中的老节点 j' 以及节点 j' 在 open 表和 closed 表中的所有后裔节点。转步骤(7)。

(7) 对 open 表中的新子节点按 g 值从小到大排序后放入 open 表中的表首，若有 2 个或 2 个以上的节点有相同的 g 值，则对这些节点再按节点序号排序。转步骤(2)。

比较有代价深度优先搜索与有代价宽度优先搜索及无代价深度优先搜索，它们的异同点主要如下。

① 无代价的深度优先搜索只能用于无代价问题的求解。有代价深度优先搜索可以用于有代价问题的求解，也可以用于无代价问题的求解。此时，由于所有的算符代价 $c(i,j)=1$，那么任何一个节点 j 的代价值 g_j 是该节点的深度 d_j，即有 $g_j=d_j$。应用有代价深度优先搜索的结果与无代价深度优先搜索的结果相同。

② 有代价深度优先搜索是在无代价深度优先搜索的基础上改进得出的，也是非完备性搜索。

③ 无代价深度优先搜索的 open 表是将新扩展生成的子节点按节点序号从小到大排序后放入 open 表表首；有代价深度优先搜索的 open 表是将新扩展生成的子节点按代价值从小到大排序后放入 open 表表首；有代价宽度优先搜索的 open 表是将表中所有节点按代价值从小到大排序。

④ 有代价深度优先搜索的步骤(6)对老节点的处理方式与有代价宽度优先搜索相同。

例 3.3 应用状态空间的搜索策略求解最短路径问题。五城市的交通路线图如图 3.7 所示，其中的数字为城市之间的交通费用(代价)。试分别应用有代价宽度优先搜索和有代价深度优先搜索，求从城市 A 经过每个城市最多一次到达城市 E 的最小交通费用：

(1) 给出问题求解的状态定义；

(2) 给出问题求解的算符定义；

(3) 应用有代价宽度优先搜索求解；

(4) 应用有代价深度优先搜索求解。

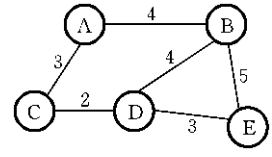

图 3.7 例 3.3 的城市交通路线图

解 (1) 状态定义

状态 S 定义为当前走过的城市名字符串，刚走过的城市名在串尾。

初态 $S_0=\mathrm{A}$

终态 $S_g=\mathrm{A}\ \$\ \mathrm{E}$

其中，$ 为城市名子串，$\$\in\{\mathrm{B},\mathrm{C},\mathrm{D}\}$。

(2) 算符定义

给出算符的一般形式定义：

$\mathrm{go}(x,y)$ 表示从城市 x 走到城市 y

使用条件：当前状态 S_i 的城市名字符串串尾城市名是 x，且 S_i 不含城市名 y。

算符操作：将城市名 y 添加到当前状态 S_i 的串尾。

由全部 12 个算符组成的算符集如表 3.1 所示。算符按出发城市名和到达城市名从 A 到 E 在算符集中排序。在搜索过程中，搜索算法每次使用可用算符的冲突消解策略是：按可用算符在算符集中的排序优先使用。

表 3.1 例 3.3 的算符集

F_k	算 符	算符代价	F_k	算 符	算符代价
F_1	go(A,B)	4	F_7	go(C,D)	2
F_2	go(A,C)	3	F_8	go(D,B)	4
F_3	go(B,A)	4	F_9	go(D,C)	2
F_4	go(B,D)	4	F_{10}	go(D,E)	3
F_5	go(B,E)	5	F_{11}	go(E,B)	5
F_6	go(C,A)	3	F_{12}	go(E,D)	3

(3) 应用有代价宽度优先搜索生成搜索树

应用有代价宽度优先搜索生成搜索树，给出搜索算法每次循环后 open 表和 closed 表中记

载的节点，表中每个节点仅给出节点序号和用括号括起来的该节点的代价值。

open=（1（0））

Loop1：closed=（1（0））

open=（3（3），2（4））

Loop2：closed=（1（0），3（3））

open=（2（4），4（5））

Loop3：closed=（1（0），3（3），2（4））

open=（4（5），5（8），6（9））

Loop4：closed=（1（0），3（3），2（4），4（5））

open=（5（8），8（8），6（9），7(9)）

Loop5：closed=（1（0），3（3），2（4），4（5），5（8））

open=（8（8），6（9），7(9)，9(10)，10(11)）

Loop6：closed=（1（0），3（3），2（4），4（5），5（8），8（8））

S_8= ACDE 是目标状态，故算法成功终止。

open=（6（9），7(9)，9(10)，10(11)）

算法终止时，生成的搜索树如图 3.8 所示。

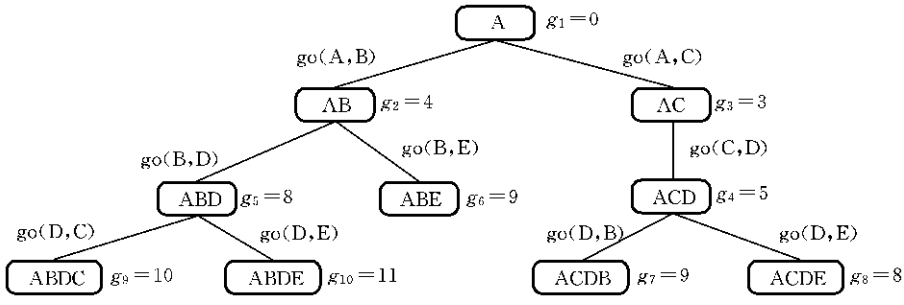

图 3.8　例 3.3 的有代价宽度优先搜索生成的搜索树

由搜索树得出问题的解　go(A,C),go(C,D),go(D,E)

解路径的费用（代价）　$g_8=8$

（4）应用有代价深度优先搜索生成搜索树

open=（1（0））

Loop1：closed=（1（0））

open=（3（3），2（4））

Loop2：closed=（1（0），3（3））

open=（4（5），2（4））

Loop3：closed=（1（0），3（3），4（5））

open=（6（8），5（9），2（4））

Loop4：closed=（1（0），3（3），4（5），6（8））

S_6= ACDE 是目标状态，故算法成功终止。

open=（5（9），2（4））

算法终止时，生成的搜索树如图 3.9 所示。

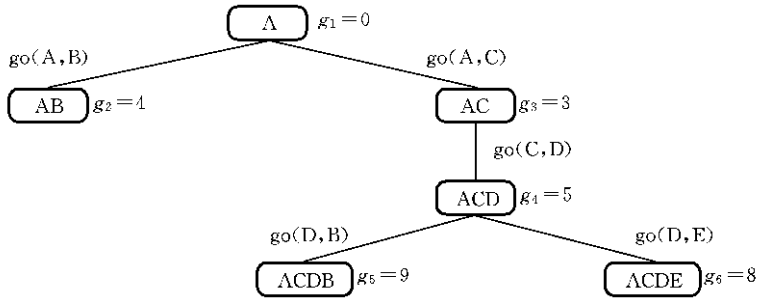

图 3.9　例 3.3 的有代价深度优先搜索生成的搜索树

由搜索树得出问题的解　　go(A,C),go(C,D),go(D,E)

解路径的费用(代价)　　$g_6=8$

3.2.2　启发式搜索算法

盲目搜索策略或者是按事先规定的路线进行搜索，或者是按已经付出的代价决定下一步要搜索的节点。例如，无代价宽度优先搜索是按“层”进行搜索的，先进入 open 表的节点先被考察；无代价深度优先搜索是沿着纵深方向进行搜索的，后进入 open 表的节点先被考察；有代价宽度优先搜索是根据 open 表中全体节点各自已付出的代价(即初始节点到该节点路径的代价)来决定哪一个节点先被考察的；而有代价深度优先搜索是在当前节点的子节点中选代价最小的节点作为被考察的节点的。它们的一个共同特点是都没有利用问题本身的特征信息，在决定被扩展的节点时，都没有考察该节点在解路径上的可能性有多大，它是否有利于问题求解及求出的解是否为最优解等。因此，这些搜索方法都具有较大的盲目性，产生的无用节点较多，搜索空间较大，效率不高。

启发式搜索要用到问题自身的某些特征信息，以指导搜索朝着最有希望的方向前进。由于这种搜索求解的针对性较强，因而效率较高。

在搜索过程中，关键的一步是如何确定下一个要考察的节点，确定的方法不同就形成了不同的搜索方法。如果在确定节点时能充分利用与问题求解有关的特征信息，估计出节点的重要性，就能在搜索时选择重要性较高的节点，以利于尽快求得最优解。这种可用于指导搜索过程，且与具体问题求解有关的控制性信息称为启发性信息。

用于估价节点重要性的函数称为估价函数。其一般形式为

$$f(x)=g(x)+h(x)$$

其中，代价函数 $g(x)$为从初始节点到节点 x 已经实际付出的代价，$h(x)$是从节点 x 到目标节点的最优路径的估计代价，它体现了问题的启发性信息，其形式要根据问题的特征确定。例如，它可以是节点 x 到目标节点的距离，也可以是节点 x 处于最优路径上的概率，等等。$h(x)$称为启发函数。

启发式搜索的 open 表和 closed 表的节点域可定义如下：

j	S_j	F_k	g_j	h_j	f_j	i

其中，h_j为节点 j 的启发函数值，f_j为节点 j 的估计函数值，其他域的定义与前面的节点域的定义相同。

1. 局部择优搜索算法

局部择优搜索是对有代价深度优先搜索的改进，有代价深度优先搜索是对新扩展生成的子节点按节点代价值 g 从小到大排序放入 open 表的表首；局部择优搜索是对新扩展生成的子节点按节点估价值 f 从小到大排序放入 open 表的表首。

局部择优搜索算法

(1) 初始化 open 表与 closed 表，把初始节点(序号 1)放入 open 表中；且令 $g_1=0$，计算h_1和 $f_1=g_1+h_1$。

(2) 若 open=( )，则算法失败终止；否则，转步骤(3)。

(3) 把 open 表的第 1 个节点 i 移出，放入 closed 表的表尾。

(4) 若节点 i 的状态 S_i是目标状态，则算法成功终止；否则，转步骤(5)。

(5) 若节点 i 不可扩展(即在算符集中找不到可用算符)，则转步骤(2)；否则，转步骤(6)。

(6) 逐一用可用算符扩展节点 i，生成 i 的所有子节点，并计算所有子节点的估价值 f。

① 若子节点 j 的状态 S_j既不在 open 表中，也不在 closed 表中，则节点 j 是一个新节点。将所有新子节点放入 open 表中，记载子节点各域的值，转步骤(7)。

② 若子节点 j 的状态 S_j已在 open 表中，即节点 j 与 open 表中的一个老节点 j'的状态相同，则比较 f_j与 $f_{j'}$。若 $f_j \geqslant f_{j'}$，则放弃新节点 j；若 $f_j < f_{j'}$，则将新节点 j 放入到 open 表中，记载节点各域的值，并删除 open 表中的老节点 j'。转步骤(7)。

③ 若子节点 j 的状态 S_j已在 closed 表中，即节点 j 与 closed 表中的一个老节点 j'的状态相同，则比较f_j与 $f_{j'}$。若 $f_j \geqslant f_{j'}$，则放弃新节点 j；若 $f_j < f_{j'}$，则将新节点 j 放入到 open 表中，记载节点各域的值，并删除 closed 表中的老节点 j'以及节点 j'在 open 表和 closed 表中的所有后裔节点。转步骤(7)。

(7) 对 open 表中的新子节点按 f 值从小到大排序后放入 open 表中的表首，若有 2 个或 2 个以上的节点有相同的 f 值，则对这些节点再按节点序号排序。转步骤(2)。

比较无代价深度优先搜索、有代价深度优先搜索与局部择优搜索，它们的异同点主要是

① 三种搜索算法都是基于深度优先的，也就是说，搜索树的扩展方向基本上是沿搜索树的纵深方向进行的。为此，三种搜索算法都是将新生成的子节点放在 open 表表首，以使得下一次循环时，被考查的节点是刚生成的子节点。

② 在局部择优搜索算法中，若所有节点 j 的启发函数值 h_j都是 0 或为一个常量，那么，局部择优搜索的结果与有代价深度优先搜索的结果相同；若又有所有的算符代价都是单位 1，即有所有节点的代价函数值 $g_j=d_j$，那么，局部择优搜索的结果与无代价深度优先搜索的结果相同。

③ 三种搜索算法都是非完备的。

④ 如果三种搜索算法都能找到问题的解路径而成功终止，那么，一般而言，无代价深度优

先搜索生成的搜索树规模较大，局部择优搜索生成的搜索树规模较小。但是，局部择优搜索的效率取决于启发函数 h 的设计，如果启发函数 h 设计合理，能较准确地估计从节点 x 到目标节点的代价，那么，将有效地减小生成的搜索树的规模，降低搜索的时空开销。

2. 全局择优搜索算法

全局择优搜索是对有代价宽度优先搜索的改进。有代价宽度优先搜索是对 open 表中全部节点按节点代价函数值 g 从小到大排序；全局择优搜索是对 open 表中全部节点按节点估计函数值 f 从小到大排序。

全局择优搜索算法

(1) 初始化 open 表与 closed 表，把初始节点(序号 1)放入 open 表中；且令 $g_1=0$，计算 h_1 和 $f_1=g_1+h_1$。

(2) 若 open=( )，则算法失败终止；否则，转步骤(3)。

(3) 把 open 表的第 1 个节点 i 移出，放入 closed 表的表尾。

(4) 若节点 i 的状态 S_i 是目标状态，则算法成功终止；否则，转步骤(5)。

(5) 若节点 i 不可扩展(即在算符集中找不到可用算符)，则转步骤(2)；否则，转步骤(6)。

(6) 逐一用可用算符扩展节点 i，生成 i 的所有子节点，并计算所有子节点的估价值。

① 若子节点 j 的状态 S_j 既不在 open 表中，也不在 closed 表中，则节点 j 是一个新节点。将所有新子节点放入 open 表中，记载子节点各域的值，转步骤(7)。

② 若子节点 j 的状态 S_j 已在 open 表中，即节点 j 与 open 表中的一个老节点 j' 的状态相同，则比较 f_j 与 $f_{j'}$。若 $f_j \geqslant f_{j'}$，则放弃新节点 j；若 $f_j < f_{j'}$，则将新节点 j 放入到 open 表中，记载节点各域的值，并删除 open 表中的老节点 j'。转步骤(7)。

③ 若子节点 j 的状态 S_j 已在 closed 表中，即节点 j 与 closed 表中的一个老节点 j' 的状态相同，则比较 f_j 与 $f_{j'}$。若 $f_j \geqslant f_{j'}$，则放弃新节点 j；若 $f_j < f_{j'}$，则将新节点 j 放入到 open 表中，记载节点各域的值，并删除 closed 表中的老节点 j' 以及节点 j' 在 open 表和 closed 表中的所有后裔节点。转步骤(7)。

(7) 对 open 表中的所有节点按 f 值从小到大的顺序重新排序，若有 2 个或 2 个以上的节点有相同的 f 值，则对这些节点再按节点序号排序。转步骤(2)。

比较无代价的宽度优先搜索、有代价宽度优先搜索与全局择优搜索，其异同点主要是

① 三种搜索算法都是基于宽度优先搜索的，但是无代价宽度优先搜索的搜索方向是严格按搜索树的“层”的方向扩展的，有代价宽度优先搜索与全局择优搜索的搜索方向不是按层扩展的。

② 在全局择优搜索算法中，若所有节点 j 的启发函数值 h_j 都是 0 或一个常量，那么，全局择优搜索的结果与有代价宽度优先搜索的结果相同；若又有所有的算符代价都是单位 1，即有所有节点的代价函数值 $g_j=d_j$，那么，全局择优搜索的结果与无代价宽度优先搜索的结果相同。

③ 无代价宽度优先搜索与有代价宽度优先搜索是完备的，但是，全局择优搜索是非完备的。

④ 一般而言，无代价宽度优先搜索生成的搜索树规模较大，全局择优搜索生成的搜索树规模较小。但是，全局择优搜索的效率取决于启发函数 h 的设计，如果启发函数 h 设计合理，能较准确地估计从节点 x 到目标节点的代价，那么，将有效地减小生成的搜索树的规模，降低搜索的时空开销。

例 3.4　应用状态空间的全局择优搜索求解例 3.3 的最短路径问题。

解　(1) 状态定义(同例 3.3)

(2) 算符定义(同例 3.3)

(3) 应用全局择优搜索生成搜索树

估计函数　　$f_j = g_j + h_j$

其中，代价函数　　$g_j = g_i + c(i,j)$

启发函数　　$h_j = a(b - d_j)$

其中，参数 a 为两个城市之间的平均费用(代价)，有

$$a = (3+4+4+2+3+5)/6 = 21 / 6 = 3.5$$

为便于计算，可取整数 $a=3$；参数 b 为从初始节点到目标节点需要到达的城市数的估计，由图 3.7 可见，可以取 $b=2$ 或取 $b=3$ 或取 $b=4$；参数 d_j 为节点 d 的深度，即表示从初始节点到达节点 j 时已到达的城市数。$(b-d_j)$ 为从节点 j 到目标节点还需到达的城市数的估计。则 h_j 表示从节点 j 到目标节点的费用(代价)的估计。

① 取 $b=2$，则 $h_j = 3(2-d_j)$。

open = (1 (6))

Loop1: closed = (1 (6))

open = (3(6) , 2(7))

Loop2: closed = (1(6) , 3(6))

open = (4(5) , 2(7))

Loop3: closed = (1(6) , 3(6) , 4(5))

open = (6(5) , 5(6) , 2(7))

Loop4: closed = (1(6) , 3(6) , 4(5) , 6(5))

S_6 = ACDE 是目标状态，故算法成功终止。

open = (5(6) , 2(7))

算法终止时，生成的搜索树如图 3.10 所示。

② 取 $b=3$，则 $h_j = 3(3-d_j)$。

open = (1(9))

Loop1: closed = (1(9))

open = (3(9) , 2(10))

Loop2: closed = (1(9) , 3(9))

open = (4(8) , 2(10))

Loop3: closed = (1(9) , 3(9) ,4(8))

open = (6(8) , 5(9) , 2(10))

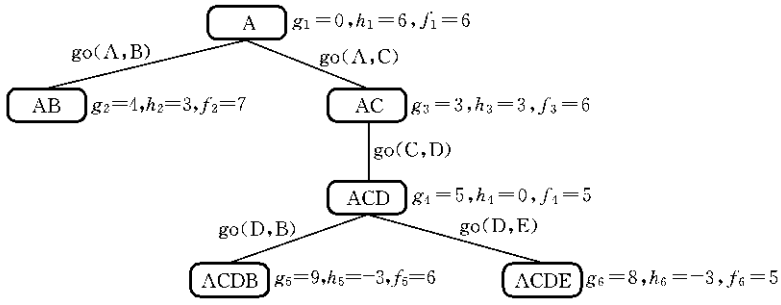

图 3.10 例 3.4 生成的搜索树(1)

Loop4:closed=(1(9),3(9),4(8),6(8))

S_6=ACDE **是目标状态,故算法成功终止。**

open=(5(9),2(10))

算法终止时,生成的搜索树如图 3.11 **所示。**

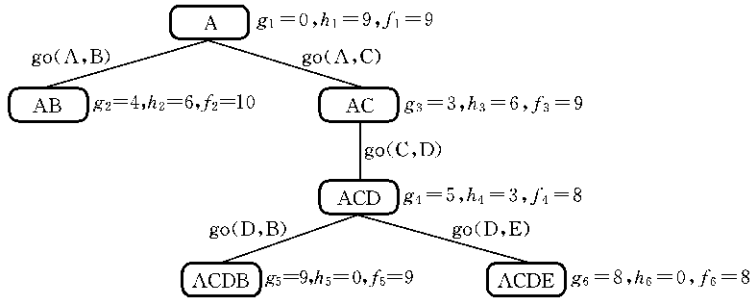

图 3.11 例 3.4 生成的搜索树(2)

③ **取** $b=4$ **,则** $h_j=3(4-d_j)$。

open=(1(12))

Loop1:closed=(1(12))

open=(3(12),2(13))

Loop2:closed=(1(12),3(12))

open=(4(11),2(13))

Loop3:closed=(1(12),3(12),4(11))

open=(6(11),5(12),2(13))

Loop4:closed=(1(12),3(12),4(11),6(11))

S_6=ACDE **是目标状态,故算法成功终止。**

open=(5(12),2(13))

算法终止时,生成的搜索树如图 3.12 **所示。**

由三种搜索树得出问题的解相同,为

go(A,C),go(C,D),go(D,E)

解路径的费用(代价) $g_6=8$

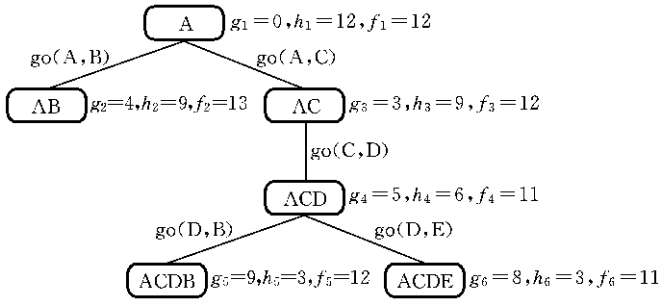

图 3.12　例 3.4 生成的搜索树(3)

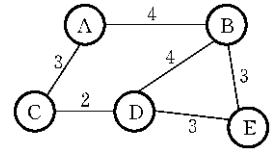

图 3.13　例 3.5 的城市交通路线图

例 3.5　五个城市之间的交通路线图如图 3.13 所示，其中的数字为城市之间的交通费用（代价）。从城市 A 出发到城市 E，且经过每个城市最多一次，求从 A 到 E 的最小费用的解：

（1）应用有代价宽度优先搜索策略，算法循环多少次终止？画出搜索树，给出问题的解。

（2）应用全局择优搜索策略，算法循环多少次终止？画出搜索树，给出问题的解。

解　（1）有代价宽度优先搜索

应用有代价宽度优先搜索算法生成的搜索树如图 3.14 所示。

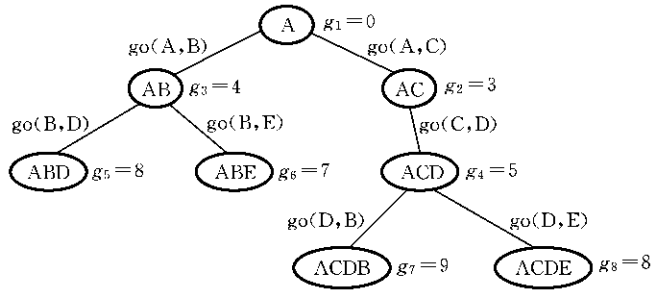

图 3.14　例 3.5 的代价树宽度优先搜索的搜索树

算法循环执行 5 次。

问题的解　　go(A,B),go(B,E)

解路径的代价　　$g_6=7$

（2）全局择优搜索

估价函数　　$f_j=g_j+h_j$

其中，代价函数　　$h_j=a(3-d_j)$

平均代价　　$a=(3+2+3+4+4+5)/6=|\ 21/6\ |=3$

应用全局择优搜索算法生成的搜索树如图 3.15 所示。

算法循环执行 4 次。

问题的解　　go(A,C),go(C,D),go(D,E)

解路径的代价　　$g_6=8$

可见，由有代价宽度优先搜索求得的解路径的代价小于由全局择优搜索求得的解路径的代价，故而前者求得的解是最优解，后者求得的解不是最优解。这是因为前者是完备的，后者是非完备的。

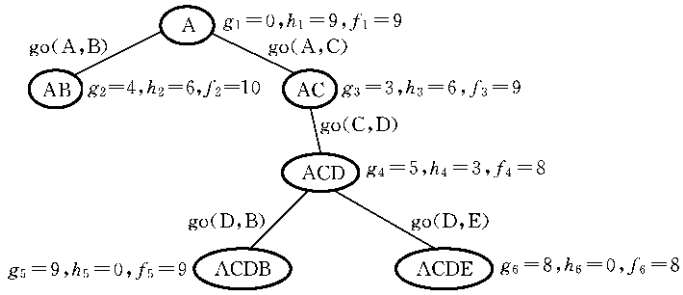

图 3.15 例 3.5 的全局择优搜索的搜索树

3.2.3 状态空间搜索算法的应用

本节以几个典型问题：最短路径问题、推销员旅行问题、九宫重排问题和三阶梵塔问题等问题的求解过程为例，说明状态空间搜索算法的应用。

例 3.6 应用状态空间的搜索策略求解最短路径问题。六城市的交通路线图如图 3.16 所示，其中的数字为城市之间的交通费用（代价）。应用下述搜索策略，求从城市 A 经过每个城市最多一次到达城市 F 的最小交通费用的解：

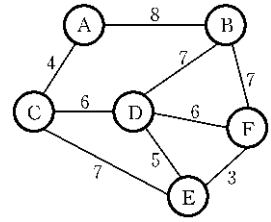

图 3.16 例 3.6 的交通路线图

（1）应用有代价宽度优先搜索求解；

（2）应用有代价深度优先搜索求解；

（3）应用全局择优搜索求解。

解 （1）状态定义

状态 S 定义为当前走过的城市名字符串，刚走过的城市名在串尾。

初态 $S_0=\mathrm{A}$

终态 $S_g=\mathrm{A}\ \$\ \mathrm{F}$

其中，$ 为城市名子串，$\in\{\mathrm{B},\mathrm{C},\mathrm{D},\mathrm{E}\}$。

（2）算符定义

给出算符的一般形式定义：

$\mathrm{go}(x,y)$ 表示从城市 x 走到城市 y

使用条件：当前状态 S_i 城市名字符串串尾城市名是 x，且 S_i 不含城市名 y。

算符操作：将城市名 y 添加到当前状态 S_i 的串尾。

由 18 个算符组成算符集，算符按出发城市名和到达城市名从 A 到 F 在算符集中排序，类似例 3.3 中的表 3.1。搜索算法每次使用可用算符时，将优先使用排序在前的可用算符。

（3）应用有代价宽度优先搜索生成搜索树

open=（1（0））

Loop1：closed=（1（0））

open=（3（4），2（8））

Loop2：closed=（1（0），3（4））

open＝（2（8），4（10），5(11））

Loop3：closed＝（1（0），3（4），2（8））

open＝（4（10），5（11），6（15），7(15））

Loop4：closed＝（1（0），3（4），2（8），4（10））

open＝（5（11），6（15），7（15），9(15），10(16），8(17））

Loop5：closed＝（1（0），3（4），2（8），4（10），5（11））

open＝（12(14），6（15），7(15），9(15），10(16），11(16），8(17））

Loop6：closed＝（1（0），3（4），2（8），4（10），5（11），12（14））

S_{12}＝ACEF **是目标状态，故算法成功终止。**

open＝（6（15），7(15），9(15），10(16），11(16），8(17））

算法终止时生成的搜索树如图 3.17 **所示。**

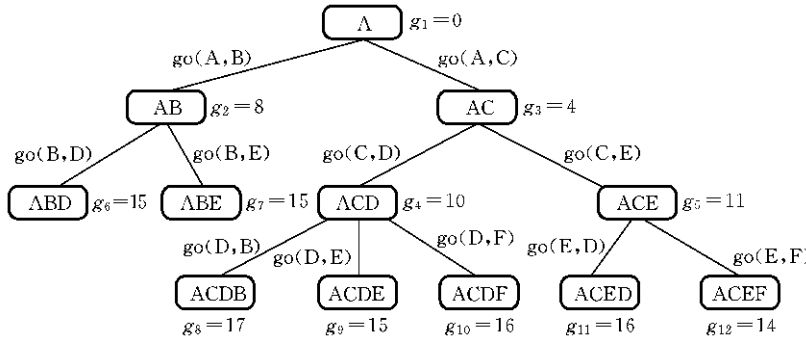

图 3.17　例 3.6 的有代价宽度优先搜索生成的搜索树

由搜索树得出问题的解　go(A,C),go(C,E),go(E,F)

解路径的费用（代价）　$g_{12}=14$

（4）**应用有代价深度优先搜索生成搜索树**

open＝（1（0））

Loop1：closed＝（1（0））

open＝（3（4），2（8））

Loop2：closed＝（1（0），3（4））

open＝（4（10），5（11），2(8））

Loop3：closed＝（1（0），3（4），4（10））

open＝（7（15），8（16），6（17），5(11），2(8））

Loop4：closed＝（1（0），3（4），4（10），7(15））

open＝（9(18），8（16），6（17），5(11），2(8））

Loop5：closed＝（1（0），3（4），4（10），7（15），9(18））

S_9＝ACDEF **是目标状态，故算法成功终止。**

open＝（8(16），6(17），5(11），2(8））

算法终止时生成的搜索树如图 3.18 **所示。**

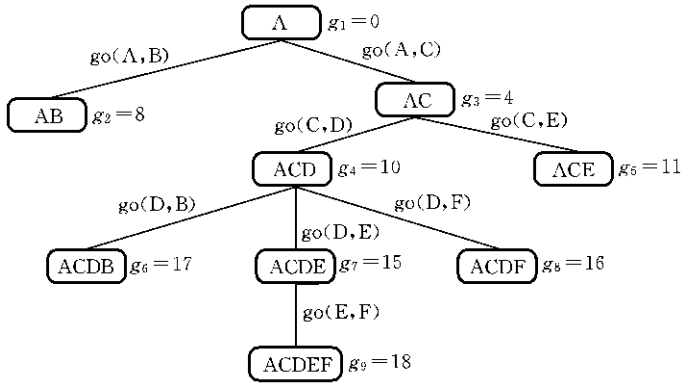

图 3.18　例 3.6 的有代价深度优先搜索的搜索树

由搜索树得出问题的解　go(A,C),go(C,D),go(D,E),go(E,F)

解路径的费用(代价)　$g_9=18$

比较有代价宽度优先搜索和有代价深度优先搜索的结果,可见:深度优先搜索即使找到问题的解,也不一定是最优解。

(5) **应用全局择优搜索生成搜索树**

估计函数　$f_j=g_j+h_j$

其中,代价函数　$g_j=g_i+c(i,j)$

启发函数　$h_j=a\ (b-d_j)$

其中,参数 a **为两个城市之间的平均费用(代价),有**

$$a=(4+6+8+7+7+7+5+6+3)/\ 9=53/9\approx 6$$

参数 b **为从初始节点到目标节点需要到达的城市数的估计,由图 3.16 可见,可以取** $b=2\sim5$**。参数** d_j **为节点** j **的深度,即表示从初始节点到节点** j **时已到达的城市数;**$(b-d_j)$**为从节点** j **到目标节点还需到达的城市数的估计;**h_j**表示从节点** j **到目标节点的费用(代价)的估计,若取** $b=3$**,则** $h_j=6(3-d_j)$**。**

open= (1 (0))

Loop1:closed= (1 (0))

open= (3 (12) , 2 (20))

Loop2:closed= (1 (0) , 3 (12))

open= (4 (16) , 5 (17) , 2(20))

Loop3:closed= (1 (0) , 3 (12) , 4 (16))

open= (7 (15) , 8 (16) , 5 (17) , 6(17) , 2(20))

Loop4:closed= (1 (0) , 3 (12) , 4 (10), 7(15))

open= (9(12) , 8 (16) , 5 (17) , 6(17) , 2(20))

Loop5:closed= (1 (0) , 3 (12) , 4 (16) , 7 (15) , 9(12))

S_9= ACDEF **是目标状态,故算法成功终止。**

open= (8(16) , 5(17) , 6(17) , 2(20))

算法终止时,生成的搜索树如图 3.19 所示。

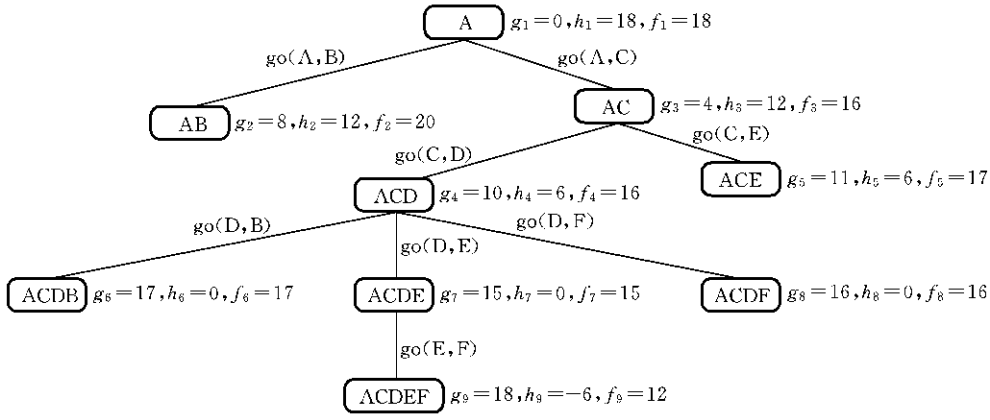

图 3.19　例 3.6 的全局择优搜索的搜索树

由搜索树得出问题的解　$\mathrm{go(A,C),go(C,D),go(D,E),go(E,F)}$

解路径的费用(代价)　$g_9=18$

比较代价树宽度优先搜索与全局择优搜索的结果，可见：全局择优搜索可以找到问题的解，但是否能找到最优解与启发函数的定义有关。

例 3.7　应用全局择优搜索求解推销员旅行问题。 A～E **五城市的交通路线图如图** 3.20 **所示，其中的数字为城市之间的交通费用(代价)。推销员从城市** A **出发经过图中每个城市一次且仅一次，回到城市** A**。求最小交通费用的解。**

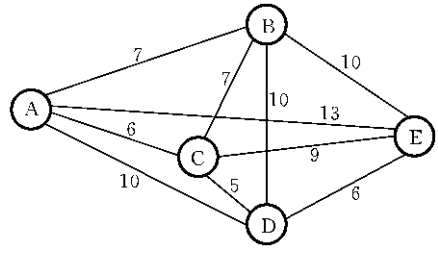

图 3.20　例 3.7 的交通路线图

解　(1) **状态定义**

状态 S **定义为当前走过的城市名字符串，刚走过的城市名在串尾。**

初态　$S_0=\mathrm{A}$

终态　$S_g=\mathrm{A}x_1x_2x_3x_4\mathrm{A}$

其中，$x_i\in\{\mathrm{B,C,D,E}\}$**，且** $x_i\neq x_j$；$i,\ j=1,2,3,4$。

(2) **算符定义**

给出算符的一般形式定义：

$\mathrm{go}(x,y)$　**表示从城市** x **走到城市** y

使用条件：当前状态 S_i**城市名字符串串尾城市名是** x**；且若字符串长度小于** 5**，则** S_i**不含城市名** y**；若字符串长度等于** 5**，则只能选取** $y=\mathrm{A}$ **的算符。**

算符操作：将城市名 y **添加到当前状态** S_i**的串尾。**

由 20 **个算符组成算符集，算符按出发城市名和到达城市名从** A **到** E **在算符集中排序，类同例** 3.3 **中的表** 3.1。

(3) **应用全局择优搜索策略生成搜索树**

估计函数　$f_j=g_j+h_j$

其中，代价函数　$g_j=g_i+c(i,j)$

启发函数 $h_j = a\ (b - d_j)$

其中，参数 a **为两个城市之间的平均费用(代价)，有**

$$a = (7+7+10+10+13+6+9+10+5+6)/10 = 83/10 \approx 8$$

参数 b **为从初始节点到目标节点需要到达的城市数，由题意可知，**$b=5$**；参数** d_j **为节点** j **的深度，即表示从初始节点到达节点** j **时已到达的城市数；**$(b-d_j)$**为从节点** j **到目标节点还需到达的城市数；**h_j**表示从节点** j **到目标节点的费用(代价)的估计，**$h_j = 8(5-d_j)$**。**

open= (1 (40))

Loop1：closed= (1 (40))

open= (3 (38) , 2 (39) , 4(42) , 5(45))

Loop2：closed= (1 (40) , 3 (38))

open= (7 (35) , 6 (37) , 2(39) , 8(39) , 4(42) , 5(45))

Loop3：closed= (1 (40) , 3 (38) , 7 (35))

open= (10(33) , 6(37) , 9(37) , 2(39) , 8(39) , 4(42) , 5(45))

Loop4：closed= (1(40) , 3(38) , 7(35), 10(33))

open= (11(35) , 6(37) , 9(37) , 2(39) , 8(39) , 4(42) , 5(45))

Loop5：closed= (1(40) , 3(38) , 7(35) , 10(33) , 11(35))

open= (12(34) , 6(37) , 9(37) , 2(39) , 8(39) , 4(42) , 5(45))

Loop6：closed= (1(40) , 3(38) , 7(35) , 10(33) , 11(35) , 12(34))

S_{12}= ACDEBA **是目标状态，故算法成功终止。**

open= (6(37) , 9(37) , 2(39) , 8(39) , 4(42) , 5(45))

算法终止时，生成的搜索树如图 3.21 **所示。**

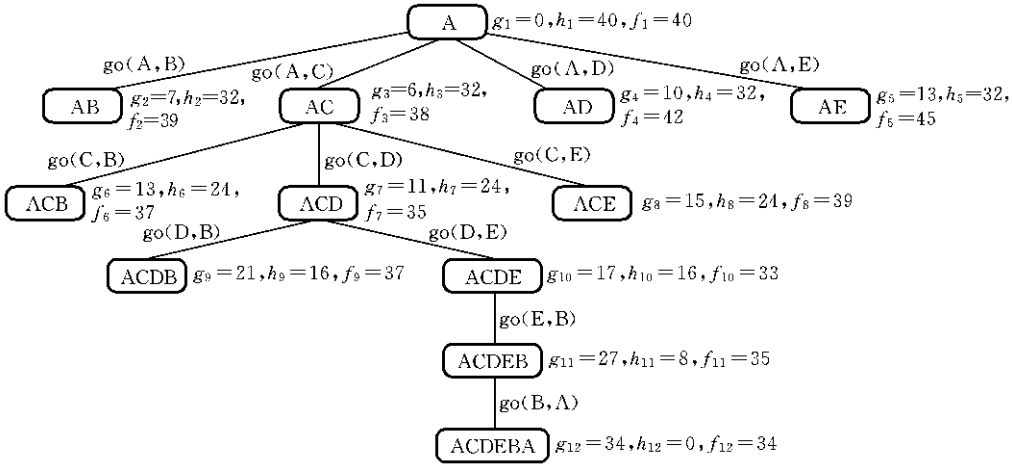

图 3.21　例 3.7 的全局择优搜索生成的搜索树

由搜索树得出问题的解　go(A,C),go(C,D),go(D,E),go(E,B),go(B,A)

解路径的费用(代价)　$g_{12}=34$

例 3.8　应用全局择优搜索求解九宫重排问题。在 3×3 **九宫棋盘上有编号为** 1～8 **共** 8 **个棋子，初始棋局如图** 3.22(a)**所示，目标棋局如图** 3.22(b)**所示。每次走子将一个棋子移动**

图 3.22　例 3.8 的初始棋局与目标棋局

(a) 初始棋局；(b) 目标棋局

到与其相邻的空格位置上。求最少移动次数的解。

解　(1) 状态定义

状态 S 定义为一个 3×3 的二维数组，表示当前棋局：

$$S=[a_{ij}]_{3\times3}\qquad a_{ij}\in\{1,2,3,4,5,6,7,8\}$$

初态

$$S_0=\begin{pmatrix}2&8&3\\1&6&4\\7&&5\end{pmatrix}$$

终态

$$S_g=\begin{pmatrix}1&2&3\\8&&4\\7&6&5\end{pmatrix}$$

(2) 算符定义

算符集　$F=\{F_1,F_2,F_3,F_4\}$

空格左移 F_1：$(a_{ij}=\text{" "})\wedge(a_{i,j-1}\neq\text{" "})\rightarrow(a_{ij}=a_{i,j-1})\wedge(a_{i,j-1}=\text{" "})$

空格上移 F_2：$(a_{ij}=\text{" "})\wedge(a_{i-1,j}\neq\text{" "})\rightarrow(a_{ij}=a_{i-1,j})\wedge(a_{i-1,j}=\text{" "})$

空格右移 F_3：$(a_{ij}=\text{" "})\wedge(a_{i,j+1}\neq\text{" "})\rightarrow(a_{ij}=a_{i,j+1})\wedge(a_{i,j+1}=\text{" "})$

空格下移 F_4：$(a_{ij}=\text{" "})\wedge(a_{i+1,j}\neq\text{" "})\rightarrow(a_{ij}=a_{i+1,j})\wedge(a_{i+1,j}=\text{" "})$

(3) 应用全局择优搜索生成搜索树

估计函数　$f_j=g_j+h_j$

其中，代价函数　$g_j=g_i+c(i,j)$

由于所有算符的代价 $c(i,j)=1$，故 $g_j=g_i+1=d_j$。

启发函数 h_j 定义为：将节点 j 的状态 S_j 与目标节点状态 S_g 比较，其错位棋子的个数。将 S_j 中错位棋子移动到 S_g 指定的位置以实现目标棋局，至少要走 h_j 次，每次移动一个棋子的代价是 1，因此，h_j 表示从节点 j 到目标节点的代价估计的下限值。

open＝(1(4))

Loop1：closed＝(1(4))

open＝(3(4)，2(6)，4(6))

Loop2：closed＝(1(4)，3(4))

open＝(5(5)，6(5)，2(6)，4(6)，7(6))

Loop3：closed＝(1(4)，3(4)，5(5))

open＝(6(5)，2(6)，4(6)，7(6)，8(6)，9(7))

Loop4：closed＝(1(4)，3(4)，5(5)，6(5))

open=（10(5)，2(6)，4(6)，7(6)，8(6)，9(7)，11(7)）

Loop5：closed=（1(4)，3(4)，5(5)，6(5)，10(5)）

open=（12(5)，2(6)，4(6)，7(6)，8(6)，9(7)，11(7)）

Loop6：closed=（1(4)，3(4)，5(5)，6(5)，10(5)，12(5)）

open=（13(5)，2(6)，4(6)，7(6)，8(6)，9(7)，11(7)，4(7)）

Loop7：closed=（1(4)，3(4)，5(5)，6(5)，10(5)，12(5)，13(5)）

$S_{13}=\begin{pmatrix}1&2&3\\8&&4\\7&6&5\end{pmatrix}$ 是目标状态，故算法成功终止。

open=（2(6)，4(6)，7(6)，8(6)，9(7)，11(7)，14(7)）

算法终止时，生成的搜索树如图 3.23 所示。图中，用黑体字表示的数字是错位棋子。

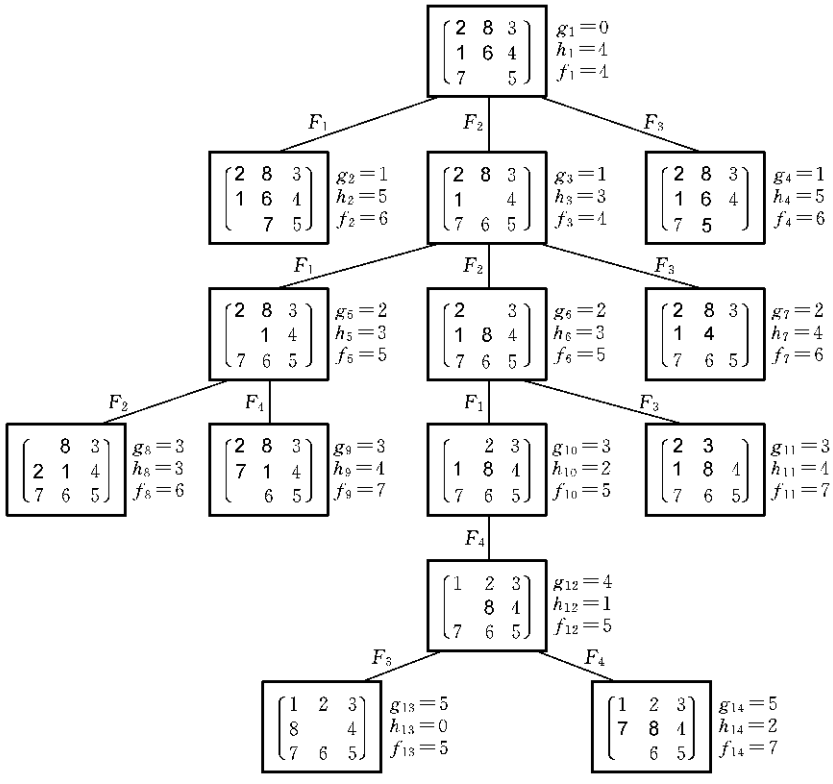

图 3.23 例 3.8 的全局择优搜索生成的搜索树

由搜索树得出问题的解

$$F_2,F_2,F_1,F_4,F_3$$

即空格上移，空格上移，空格左移，空格下移，空格右移。

解路径的费用（代价）　　$g_{13}=5$

例 3.9 应用全局择优搜索求解三阶梵塔问题。在编号为 1、2、3 的 1 号柱上有 A、B、C 三个盘，大盘 C 在下层，小盘 A 在上层，如图 3.24(a)所示。每次在柱间移动一个盘，任何一根柱上若有 2 个或 2 个以上的盘，都必须大盘在下，小盘在上。要求将 3 个盘都移动到 3 号柱

图 3.24　例 3.9 的初始状态与目标状态

(a) 初始状态；(b) 目标状态

上，如图 3.24(b)所示。求移动次数最少的解。

解　(1) 状态定义

状态 S 定义为三元组：

$$S=(x_1,x_2,x_3)$$

其中，x_n 为柱 n 上的盘名序列。由于柱上各盘只能大盘在下，小盘在上，因此，盘名序列遵守 C>B>A，即 C 在 B 之前，B 在 A 之前。

初态　　$S_0=(\mathrm{CBA},\varnothing,\varnothing)$

终态　　$S_g=(\varnothing,\varnothing,\mathrm{CBA})$

(2) 算符定义

给出算符的一般形式定义：

算符 $M(k,n,m)$ 表示将盘 k 从柱 n 移至柱 m；$k\in\{\mathrm{A},\mathrm{B},\mathrm{C}\}$，$n,m\in\{1,2,3\}$

使用条件：当前状态 S_i 中的 x_n 盘名序列末尾的盘名是 k，且 x_m 盘名序列末尾的盘名与盘 k 之间遵守 C>B>A。

算符操作：将当前状态 S_i 中的 x_n 盘名序列末尾的盘名 k 删除，并在 x_m 盘名序列的末尾增添盘名 k。

每个盘在三个柱间移动的算符有 6 个，三个盘共有 18 个算符组成算符集。算符在算符集中的排序如表 3.2 所示，所有算符的代价是 1。

表 3.2　例 3.9 的算符集

F_K	算　符	F_K	算　符	F_K	算　符
F_1	$M(\mathrm{A},1,2)$	F_7	$M(\mathrm{B},1,2)$	F_{13}	$M(\mathrm{C},1,2)$
F_2	$M(\mathrm{A},1,3)$	F_8	$M(\mathrm{B},1,3)$	F_{14}	$M(\mathrm{C},1,3)$
F_3	$M(\mathrm{A},2,1)$	F_9	$M(\mathrm{B},2,1)$	F_{15}	$M(\mathrm{C},2,1)$
F_4	$M(\mathrm{A},2,3)$	F_{10}	$M(\mathrm{B},2,3)$	F_{16}	$M(\mathrm{C},2,3)$
F_5	$M(\mathrm{A},3,1)$	F_{11}	$M(\mathrm{B},3,1)$	F_{17}	$M(\mathrm{C},3,1)$
F_6	$M(\mathrm{A},3,2)$	F_{12}	$M(\mathrm{B},3,2)$	F_{18}	$M(\mathrm{C},3,2)$

(3) 应用全局择优搜索生成搜索树

估计函数　　$f_j=g_j+h_j$

其中，代价函数 $g_j=g_i+c(i,j)$，由于所有算符的代价 $c(i,j)=1$，故 $g_j=g_i+1=d_j$。

启发函数　　$h_j=a_{\mathrm{A}}+a_{\mathrm{B}}+a_{\mathrm{C}}=\sum a_k$

其中，a_k为在当前状态 S_j 中，盘 k 移动到目标状态 S_g 指定位置上至少需要移动次数的估计。

$$a_k=\begin{cases}0 & \text{若盘 } k \text{ 在 } S_g \text{ 指定的位置上，即 } k\in x_3\text{，且在 } x_3=\text{CBA 指定位置上}\\ 1 & \text{若盘 } k \text{ 在柱 1 或柱 2 上，即 } k\in x_1 \text{ 或 } k\in x_2\\ 2 & \text{若盘 } k \text{ 在柱 3 上，且不在 } S_g \text{ 指定位置上}\end{cases}$$

open=（1(3)）

Loop1：closed=（1(3)）

open=（2(4)，3(5)）

Loop2：closed=（1(3)，2(4)）

open=（3(5)，4(6)）

Loop3：closed=（1(3)，2(4)，3(5)）

open=（4(6)，5(6)）

Loop4：closed=（1(3)，2(4)，3(5)，4(6)）

open=（5(6)，7(7)，6(8)）

Loop5：closed=（1(3)，2(4)，3(5)，4(6)，5(6)）

open=（8(6)，9(6)，7(7)，6(8)）

Loop6：closed=（1(3)，2(4)，3(5)，4(6)，5(6)，8(6)）

open=（9(6)，10(6)，7(7)，6(8)）

Loop7：closed=（1(3)，2(4)，3(5)，4(6)，5(6)，8(6)，9(6)）

节点9 的三个子节点的状态都是老状态，分别是 S_8，S_5，S_7。

open=（10(6)，7(7)，6(8)）

Loop8：closed=（1(3)，2(4)，3(5)，4(6)，5(6)，8(6)，9(6)，10(6)）

open=（7(7)，12(7)，6(8)，11(8)）

Loop9：closed=（1(3)，2(4)，3(5)，4(6)，5(6)，8(6)，9(6)，
10(6)，7(7)）

节点7 的三个子节点的状态都是老状态，分别是 S_4，S_6，S_9。

open=（12(7)，6(8)，11(8)）

Loop10：closed=（1(3)，2(4)，3(5)，4(6)，5(6)，8(6)，9(6)，
10(6)，7(7)，12(7)）

open=（13(7)，6(8)，11(8)）

Loop11：closed=（1(3)，2(4)，3(5)，4(6)，5(6)，8(6)，9(6)，
10(6)，7(7)，12(7)，13(7)）

open=（15(7)，6(8)，11(8)，14(8)）

Loop12：closed=（1(3)，2(4)，3(5)，4(6)，5(6)，8(6)，9(6)，
10(6)，7(7)，12(7)，13(7)，15(7)）

$S_{15}=(\varnothing,\varnothing,\text{CBA})$ 是目标状态，故算法成功终止。

open=（6(8)，11(8)，14(8)）

算法终止时，生成的搜索树如图 3.25 所示。

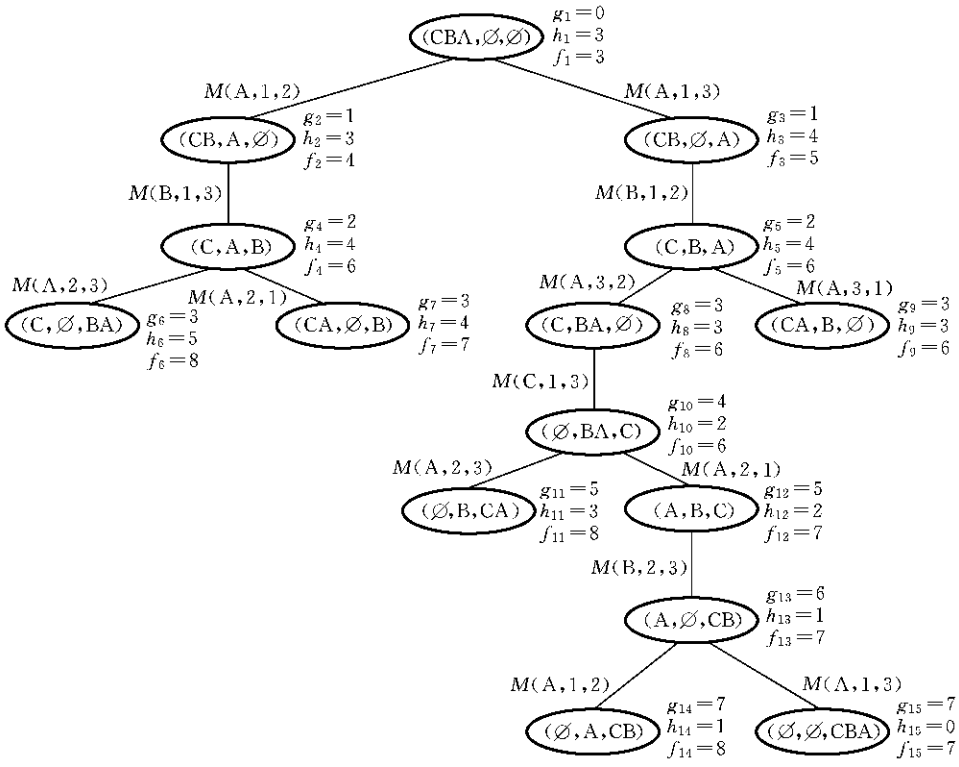

图 3.25　例 3.9 的全局择优搜索生成的搜索树

由搜索树得出问题的解

$M(\mathrm{A},1,3),M(\mathrm{B},1,2),M(\mathrm{A},3,2),M(\mathrm{C},1,3),M(\mathrm{A},2,1),M(\mathrm{B},2,3),M(\mathrm{A},1,3)$

解路径的代价　　$g_{15}=7$

例 3.10　应用状态空间的全局择优搜索求解三阶梵塔问题：在编号为 1,2,3 的 1 号柱上有 A,B,C 三个盘，大盘 C 在下层，小盘 A 在上层，如图 3.24(a)所示。每次在柱间移动一个盘，任何一根柱上若有 2 个或 2 个以上的盘，都必须大盘在下，小盘在上。移动小盘 A 一次的代价是 1，移动中盘 B 一次的代价是 2，移动大盘 C 一次的代价是 3。要求将 3 个盘都移动到 3 号柱上，如图 3.24(b)所示。求移动代价最小的解。

解　(1) **状态定义**

状态 S 定义为三元组：

$$S=(x_1,x_2,x_3)$$

其中，x_n 为柱 n 上的盘名序列。由于柱上各盘只能大盘在下，小盘在上，因此，盘名序列遵守 C>B>A，即 C 在 B 之前，B 在 A 之前。

初态　　$S_0=(\mathrm{CBA},\varnothing,\varnothing)$

终态　　$S_g=(\varnothing,\varnothing,\mathrm{CBA})$

(2) **算符定义**

给出算符的一般形式定义：

算符 $M(k,n,m)$ 表示将盘 k 从柱 n 移至柱 m，$k\in\{\mathrm{A},\mathrm{B},\mathrm{C}\},n,m\in\{1,2,3\}$

使用条件：当前状态 S_j 中的 x_n 盘名序列末尾的盘名是 k，且 x_m 盘名序列末尾的盘名与盘 k 之间遵守 C>B>A。

算符操作：将 x_n 盘名序列末尾的盘名 k 删除，并在 x_m 盘名序列的末尾增添盘名 k。

算符 $M(\mathrm{A},n,m)$ 的代价为 $c(\mathrm{A},n,m)=1$

算符 $M(\mathrm{B},n,m)$ 的代价为 $c(\mathrm{B},n,m)=2$

算符 $M(\mathrm{C},n,m)$ 的代价为 $c(\mathrm{C},n,m)=3$

(3) 应用全局择优搜索生成搜索树

估计函数 $$f_j=g_j+h_j$$

其中，代价函数 $$g_j=g_i+c(k,i,j)$$

$c(k,i,j)$ 为在状态 S_i 使用算符 $M(k,n,m)$ 将盘 k 移动后生成状态 S_j 时的算符代价。

启发函数 $$h_j=a_{\mathrm{A}}c(\mathrm{A},i,j)+a_{\mathrm{B}}c(\mathrm{B},i,j)+a_{\mathrm{C}}c(\mathrm{C},i,j)=\sum a_k c(k,i,j)$$

由于 a_k 为在当前状态 S_j 中，盘 k 移动到目标状态 S_g 指定位置上至少需要移动次数的估计，故有

$$a_k=\begin{cases}0 & \text{若盘 } k \text{ 在 } S_g \text{ 指定的位置上，即 } k\in x_3\text{，且在 } x_3=\mathrm{CBA} \text{ 指定位置上}\\ 1 & \text{若盘 } k \text{ 在柱 1 或柱 2 上，即 } k\in x_1 \text{ 或 } k\in x_2\\ 2 & \text{若盘 } k \text{ 在柱 3 上，且不在 } S_g \text{ 指定位置上}\end{cases}$$

open=(1(6))

Loop1: closed=(1(6))

open=(2(7),3(8))

Loop2: closed=(1(6),2(7))

open=(3(8),4(10))

Loop3: closed=(1(6),2(7),3(8))

open=(5(9),4(10))

Loop4: closed=(1(6),2(7),3(8),5(9))

open=(6(9),7(9),4(10))

Loop5: closed=(1(6),2(7),3(8),5(9),6(9))

open=(8(7),7(9),4(10))

Loop6: closed=(1(6),2(7),3(8),5(9),6(9),8(7))

open=(10(8),7(9),9(9),4(10))

Loop7: closed=(1(6),2(7),3(8),5(9),6(9),8(7),10(8))

open=(11(7),7(9),9(9),4(10))

Loop8: closed=(1(6),2(7),3(8),5(9),6(9),8(7),10(8),11(7))

open=(12(7),13(8),7(9),9(9),4(10))

Loop9: closed=(1(6),2(7),3(8),5(9),6(9),8(7),10(8),
11(7),12(7))

$S_{12}=(\varnothing,\varnothing,\mathrm{CBA})$ 是目标状态，故算法成功终止。

open=(13(8),7(9),9(9),4(10))

搜索树如图 3.26 所示。

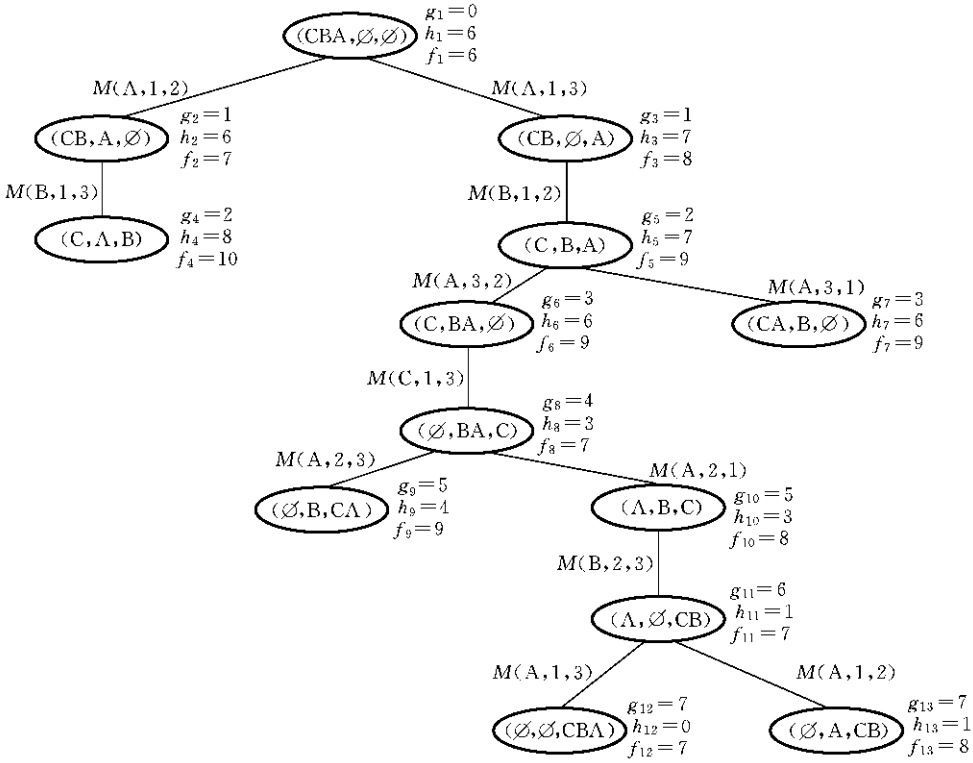

图 3.26　例 3.10 的全局择优搜索生成的搜索树

由搜索树得出问题的解

$M(\mathrm{A},1,3),M(\mathrm{B},1,2),M(\mathrm{A},3,2),M(\mathrm{C},1,3),M(\mathrm{A},2,1),M(\mathrm{B},2,3),M(\mathrm{A},1,3)$

解路径的代价　　$g_{12}=7$

3.2.4　A* 算法及其特性

由前述的多个例题可见，全局择优搜索是非完备的，也就是说，全局择优搜索求得的解可能不是最优解，这与设计的启发函数有关。为此，给出完备的 A* 算法。

1. A* 算法

如果全局择优搜索算法的估价函数 f 满足如下限制，则成为 A* 算法，节点的估价函数

$$f(x)=g(x)+h(x)$$

其中：

① 代价函数 $g(x)$ 是对 $g^*(x)$ 的估计，$g(x)>0$。$g^*(x)$ 是从初始节点 S_0 到节点 x 的最小代价。

② 启发函数 $h(x)$ 是 $h^*(x)$ 的下界，即对所有的 x 均有 $h(x)\leqslant h^*(x)$。$h^*(x)$ 是从节点 x 到目标节点的最小代价，若有多个目标节点，则为其中最小的代价。

在 A* 算法中，$g(x)$比较容易得到，它实际上就是从初始节点 S_0 到节点 x 的路径代价，恒有 $g(x)\geqslant g^*(x)$。

$h(x)$的确定依赖于具体问题中的启发性知识，其中，$h(x)\leqslant h^*(x)$的限制是十分重要的，它可保证 A* 算法能找到最优解。

2. A* 算法的特性

(1) A* 算法的可纳性

对于可解状态空间图(即从初始节点到目标节点有路径存在)来说，如果一个搜索算法能在有限步内终止，并且能找到最优解，则称该搜索算法是可纳的。

A* 算法是可纳的，即它能在有限步内终止并找到最优解。下面分三步证明这一结论。

① 对于有限图，A* 算法一定会在有限步内终止。

证明　对于有限图，其节点个数是有限的。因此，A* 算法在经过若干次循环之后只可能出现两种情况：由于搜索到了目标节点而在第(4)步终止；由于 open 表中的节点被取完而在第(2)步终止。不管发生哪种情况，A* 算法都在有限步内终止。

② 对于无限图，只要从初始节点到目标节点有路径存在，则 A* 算法也必然会终止。

证明　该证明分两步进行。第一步先证明在 A* 算法结束之前，open 表中总存在节点 x'，它是最优路径的一个节点，且满足

$$f(x')\leqslant f^*(S_0)$$

设最优路径是　$S_0,x_1,x_2,\cdots,x_m,S_g^*$

由于 A* 算法中的 $h(x)$满足　$h(x)\leqslant h^*(x)$

所以，$f(S_0),f(x_1),f(x_2),\cdots,f(x_m)$均不大于 $f(S_g^*)$，$f(S_g^*)=f^*(S_0)$。

又因为 A* 算法是全局择优的，所以在它结束之前，open 表中一定含有 $S_0,x_1,x_2,\cdots,x_m,S_g^*$ 中的一些节点，设 x'是其中最前面的一个，则必然满足

$$f(x')\leqslant f^*(S_0)$$

至此，第一步证明结束。

现在来进行第二步的证明。这一步用反证法，即假设 A* 算法不终止，则会得出与上一步矛盾的结论，从而说明 A* 算法一定会终止。

假设 A* 算法不终止，并设 e 是图中各条边的最小代价，$d^*(x_n)$是从 S_0 到节点 x_n 的最短路径长度，则显然有

$$g^*(x_n)\geqslant d^*(x_n)\times e$$

因为 $g(x_n)\geqslant g^*(x_n)$，所以有

$$g(x_n)\geqslant d^*(x_n)\times e$$

又因为 $h(x_n)\geqslant 0$，$f(x_n)\geqslant g(x_n)$，故得到

$$f(x_n)\geqslant d^*(x_n)\times e$$

由于 A* 算法不终止，随着搜索的进行，$d^*(x_n)$会无限增大，从而使 $f(x_n)$也无限增大。这就与上一步证明得出的结论矛盾，因为对可解状态空间来说，$f^*(S_0)$一定是有限值。所以，只要从初始节点到目标节点有路径存在，即使对于无限图，A* 算法也一定会终止。

③ A^* 算法一定终止在最优路径上。

证明　假设 A^* 算法不是在最优路径上终止，而是在某个目标节点 t 处终止，即 A^* 算法未能找到一条最优路径，则

$$f(t)=g(t)>f^*(S_0)$$

但由②的证明可知，在 A^* 算法结束之前，open 表中存在节点 x'，它在最优路径上，且满足

$$f(x')\leqslant f^*(S_0)$$

此时，A^* 算法一定会选择 x' 来扩展而不会选择 t，这就与假设矛盾。所以，A^* 算法一定终止在最优路径上。

根据可纳性的定义及以上证明可知 A^* 算法是可纳的。同时，由上面的证明还可得知 A^* 算法选择扩展的任何一个节点 x 都满足如下性质：

$$f(x)\leqslant f^*(S_0)$$

(2) A^* 算法的最优性

A^* 算法的搜索效率在很大程度上取决于 $h(x)$，在满足 $h(x)\leqslant h^*(x)$ 的前提下，$h(x)$ 的值越大越好。$h(x)$ 的值越大，表明它携带的启发性信息越多，搜索时扩展的节点数越少，搜索的效率越高。

设 $f_1(x)$ 与 $f_2(x)$ 是对同一问题的两个估价函数：

$$f_1(x)=g_1(x)+h_1(x)$$

$$f_2(x)=g_2(x)+h_2(x)$$

A_1^* 与 A_2^* 分别是以 $f_1(x)$ 及 $f_2(x)$ 为估价函数的 A^* 算法，且设对所有非目标节点 x 均有

$$h_1(x)<h_2(x)$$

在此情况下，我们将证明 A_1^* 扩展的节点数不会比 A_2^* 扩展的节点数少，即 A_2^* 扩展的节点集是 A_1^* 扩展的节点集的子集。用归纳法证明如下。

设 k 表示搜索树的深度。当 $k=0$ 时，结论显然成立。因为若初始状态就是目标状态，则 A_1^* 与 A_2^* 都无须扩展任何节点；若初始状态不是目标状态，它们都要对初始节点进行扩展，此时 A_1^* 与 A_2^* 扩展的节点数是相同的。

设当搜索树的深度为 $k-1$ 时结论成立，即凡 A_2^* 扩展了的前 $k-1$ 代节点，A_1^* 也都扩展了。此时，只要证明 A_2^* 扩展的第 k 代的任一节点被 A_1^* 扩展就可以了。

由假设可知，A_2^* 扩展的前 $k-1$ 代节点 A_1^* 也都扩展了，因此在 A_1^* 搜索树中有一条从初始节点 S_0 到 x_k 的路径，其费用不会比 A_2^* 搜索树中从 S_0 到 x_k 的费用更大，即

$$g_1(x_k)\leqslant g_2(x_k)$$

假设 A_1^* 不扩展节点 x_k，这表示 A_1^* 能找到另一个具有更小估价值的节点进行扩展并找到最优解，此时有

$$f_1(x_k)\geqslant f^*(S_0)$$

即

$$g_1(x_k)+h_1(x_k)\geqslant f^*(S_0)$$

应用关系式 $g_1(x_k)\leqslant g_2(x_k)$ 到上述不等式中，得到

$$h_1(x_k)\geqslant f^*(S_0)-g_2(x_k)$$

由于 $h_2(x_k)=f^*(S_0)-g_2(x_k)$，所以得到

$$h_1(x_k) \geqslant h_2(x_k)$$

这与最初的假设 $h_1(x) < h_2(x)$ 是矛盾的。

由此可得出“A_1^* 所扩展的节点数不会比 A_2^* 扩展的节点数少”这一结论是正确的，即：启发函数所携带的启发性信息越多，搜索时扩展的节点数越少，搜索效率越高。

(3) $h(x)$的单调性限制

在 A^* 算法中，每当要扩展一个节点时都要先检查其子节点是否已在 open 表或 closed 表中，有时还需要对老节点及其后裔节点进行处理，这就增加了搜索的代价。如果对启发函数 $h(x)$ 加上单调性限制，就可减少检查及调整的工作量，从而减少搜索代价。

单调性限制是指 $h(x)$ 满足如下两个条件：

① $h(S_g)=0$；

② 设 x_j 是节点 x_i 的任意子节点，则有

$$h(x_i)-h(x_j) \leqslant c(x_i, x_j)$$

其中，S_g 是目标节点；$c(x_i, x_j)$ 是节点 x_i 到其子节点 x_j 的边代价。

若把上述不等式改写为如下形式：

$$h(x_i) \leqslant h(x_j)+c(x_i, x_j)$$

就可看出节点 x_i 到目标节点最优费用的估价不会超过从 x_i 到其子节点 x_j 的边代价加上从 x_j 到目标节点最优费用的估价。

可以证明，当 A^* 算法的启发函数 $h(x)$ 满足单调限制时，可得到如下两个结论：

① 若 A^* 算法选择节点 x_n 进行扩展，则

$$g(x_n)=g^*(x_n)$$

② 由 A^* 算法所扩展的节点序列其 f 值是非递减的。

这两个结论都是当 $h(x)$ 满足单调限制时才成立；否则，它们不一定成立。例如，对第②个结论，当 $h(x)$ 不满足单调限制时，有可能某个要扩展的节点比以前扩展的节点具有较小的 f 值。

3.3 与/或图的搜索方法

如果问题求解定义的算符除了有等价变换算符，还有分解变换算符，那么，搜索生成的搜索树中，除了有或节点，还有与节点，则得到的解也不是解路径，而是解树。因此，需要根据这种新情况改进状态空间的搜索方法，得出与/或图的搜索方法，搜索生成的搜索树是一棵与/或树。

与/或图的搜索方法也分为盲目搜索与启发式搜索两类。

3.3.1 与/或图的盲目搜索算法

与/或图的盲目搜索方法用于求解无代价问题，分为宽度优先搜索与深度优先搜索两种。

1. 可解标示过程与不可解标示过程

与/或图上的一个节点是否为可解节点是由它的子节点确定的。对于一个与节点，只有当其子节点全部为可解节点时，它才为可解节点；只要子节点中有一个为不可解节点，它就是不可解节点。对于一个或节点，只要子节点中有一个是可解节点，它就是可解节点；只有当全部子节点都是不可解节点时，它才是不可解节点。解树是由可解的初始节点与其搜索的可解的后裔节点组成。因此，在与/或图的搜索方法中需要专门编写两个过程，由子节点的可解或不可解来确定父节点的可解或不可解。由可解子节点来确定父节点、祖父节点等为可解节点的回溯向上过程称为可解标示过程；由不可解子节点来确定其父节点、祖父节点等为不可解节点的回溯向上过程称为不可解标示过程。在与/或树的搜索过程中将反复使用这两个过程，直到初始节点（即原始问题）被标示为可解或不可解节点为止。

可解标示过程和不可解标示过程都是自下而上进行的，即由子节点的可解性确定父节点的可解性。由于与/或图搜索的目标是寻找解树，因此，如果已确定某个节点是不可解节点，则其全部后裔节点都不再有用，可从搜索树中删去，但当前这个不可解节点还不能删去，因为在判断其先辈节点的可解性时还要用到它。

与/或图搜索的目标是：寻找解树，从而求得初始问题的解。如果在搜索的某一时刻，通过可解标示过程可确定初始节点是可解的，则由此初始节点及其可解的后裔节点就构成了解树。

2. 与/或图的宽度优先搜索

与/或图的宽度优先搜索的 open 表和 closed 表的节点域可定义如下：

j	S_j	F_k	and/or	y/n	i

其中，j 为节点 j 的序号（通常按节点生成的顺序编号）。S_j 为节点 j 的问题表示。F_k 为生成节点 j 所使用的算符编号（算符名称）。and/or 是与/或标志，若 F_k 是分解算符，则与/或标志为 and，表示节点 j 是与关系节点；若 F_k 是等价变换算符，则与/或标志为 or，表示节点 j 是或关系节点。y/n 是可解/不可解标志，若节点 j 是可解节点，则可解/不可解标志为 y；若节点 j 是不可解节点，则可解/不可解标志为 n。如果节点 j 是终叶节点，则节点 j 的可解/不可解标志为 y；如果节点 j 是非终叶节点的端节点，则节点 j 的可解/不可解标志为 n。其他节点的可解/不可解标志由可解标示过程或不可解标示过程设置。i 为节点 j 的父节点序号（或者是节点 j 指向父节点 i 的指针）。

与/或图宽度优先搜索算法

（1）初始化 open 表与 closed 表，把初始节点（序号 1）放入 open 表中。

（2）若 open＝（ ），则算法失败终止；否则，转步骤（3）。

（3）把 open 表中尚未标示可解/不可解标志的第一个节点 i 移出，放入 closed 表表尾。

（4）若节点 i 可扩展（即在算符集中找到可用算符），则用可用算符作用于节点 i 的问题表示 S_i，生成其子问题节点；否则，转步骤（6）。

若子节点 j 的问题表示 S_j，既不在 open 表中，又不在 closed 表中，则节点 j 是一个新节点，并将节点 j 放入 open 表的表尾，并记载节点 j 各域的值，转步骤（5）；否则，放弃节点 j，转

步骤(2)。

(5) 若子节点 j 是终叶节点(即 S_j 是本原问题),则标示节点 j 可解,并调用可解标示过程。若可解标示过程将初始节点标示为可解,则算法成功终止;否则,转步骤(2)。

(6) 若节点 i 不可扩展(即在算符集中找不到可用算符),则标示节点 i 不可解,并调用不可解标示过程。若不可解标示过程将初始节点标示为不可解,则问题无解,算法终止;否则,删除节点 i 在 open 表和 closed 表中的所有后裔节点,转步骤(2)。

3. 与/或图的深度优先搜索

与/或图的深度优先搜索的 open 表和 closed 表的节点域可定义如下:

j	S_j	F_k	d_j	and/or	y/n	i

其中,d_j 是节点 j 的深度,$d_j=d_i+1$。其他域的定义与宽度优先搜索的节点域定义相同。

与/或图深度优先搜索算法

(1) 初始化 open 表与 closed 表,把初始节点(序号 1)放入 open 表中,设置搜索最大深度 $d_{\max}$ 的值。

(2) 若 open=( ),则算法失败终止;否则,转步骤(3)。

(3) 把 open 表中尚未标示可解/不可解标志的第一个节点 i 移出,放入 closed 表表尾。

(4) 若节点 i 的深度 $d_i<d_{\max}$ 且可扩展(即在算符集中找到可用算符),则用可用算符作用于节点 i 的问题表示 S_i,生成其子问题节点;否则,转步骤(6)。

若子节点 j 的问题表示 S_j,既不在 open 表中,又不在 closed 表中,则节点 j 是一个新节点。将节点 j 放入 open 表的表首,并记载节点 j 各域的值,转步骤(5);否则,放弃节点 j,转步骤(2)。

(5) 若子节点 j 是终叶节点(即 S_j 是本原问题),则标示节点 j 可解,并调用可解标示过程。若可解标示过程将初始节点标示为可解,则算法成功终止;否则,转步骤(2)。

(6) 若节点 i 不可扩展(即在算符集中找不到可用算符),则标示节点 i 不可解,并调用不可解标示过程。若不可解标示过程将初始节点标示为不可解,则问题无解,算法终止;否则,删除节点 i 在 open 表和 closed 表中的所有后裔节点,转步骤(2)。

3.3.2 与/或图的启发式搜索算法

如果求解问题定义的算符的代价不等,则不能使用与/或图的盲目搜索方法求解。使用与/或图的启发式搜索方法,一是可求解有代价的问题,二是可利用节点的估价值进行启发式搜索。

1. 解树的代价

为了进行启发式搜索,需要计算解树的代价。解树的代价可通过计算解树中节点的代价得到。

设 $c(x,y)$ 表示节点 x 到其子节点 y 的算符代价,$g(x)$ 表示节点 x 的代价,计算节点 x 代

价的方法如下。

① 如果 x 是终叶节点，则定义节点 x 的代价 $g(x)=0$。

② 如果 x 是或节点，y_1，y_2，…，y_n是它的子节点，则节点 x 的代价为

$$h(x)=\min_{1\leqslant i\leqslant n}\{c(x,y_i)+h(y_i)\}$$

③ 如果 x 是与节点，则节点 x 的代价有两种计算方法：和代价法与最大代价法。若按和代价法计算，则有

$$h(x)=\sum_{i=1}^{n}(c(x,y_i)+h(y_i))$$

若按最大代价法计算，则有

$$h(x)=\max_{1\leqslant i\leqslant n}\{c(x,y_i)+h(y_i)\}$$

④ 如果 x 既不可扩展，又不是终叶节点，则定义 $g(x)=\infty$。

由上述计算节点的代价可以看出，如果问题是可解的，则由子节点的代价就可推算出父节点代价，这样逐层上推，最终就可求出初始节点的代价。初始节点的代价就是解树的代价。

例 3.11　图 3.27 所示为一棵与/或树，包括两棵解树，已知节点的问题 S_4，S_5，S_9，S_{12}，S_{13}是本原问题，即所在节点是终叶节点，边上的数字是算符的代价。

（1）请按和代价法求出最优解树；

（2）请按最大代价法求出最优解树。

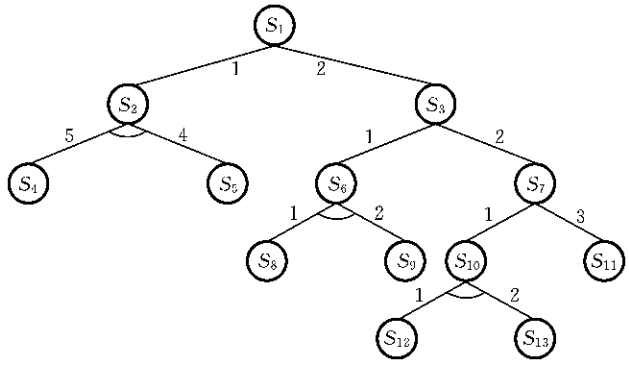

图 3.27　与/或树的代价计算示例

解　图 3.27 所示的与/ 或树有 2 棵解树，分别是

左解树　　S_1,S_2,S_4,S_5

右解树　　$S_1,S_3,S_7,S_{10},S_{12},S_{13}$

（1）按和代价法分别计算 2 棵解树的代价

对左解树由底向上分别得出各节点的代价为

$$h(S_4)=0,\ h(S_5)=0,\ h(S_2)=9,\ h(S_1)=10$$

对右解树由底向上分别得出各节点的代价为

$$h(S_{12})=0,\ h(S_{13})=0,\ h(S_{10})=3,\ h(S_7)=4,\ h(S_3)=6,\ h(S_1)=8$$

可见，右解树是最优解树。

(2) 按最大代价法分别计算 2 棵树的代价

对左解树由底向上分别得出各节点的代价为

$$g(S_4)=0,\ g(S_5)=0,\ g(S_2)=5,\ g(S_1)=6$$

对右解树由底向上分别得出各节点的代价为

$$g(S_{12})=0,\ g(S_{13})=0,\ g(S_{10})=2,\ g(S_7)=3,\ g(S_3)=5,\ g(S_1)=7$$

可见,左解树是最优解树。

2. 希望树

启发式搜索希望求出最优解树,即代价最小的解树。这就要求搜索过程中每次求出的部分解树的代价都应是最小的,如同状态空间的启发式搜索一样,每次循环求得的部分解路径的代价都是最小值。为此,每次选择欲扩展的节点时都应挑选有希望成为最优解树的一部分的那些节点进行扩展。由这些节点及其先辈节点(包括初始节点)所构成的与/或树称为"希望树"。在搜索过程中,随着新节点的不断生成,节点的代价值是在不断变化的,因此希望树也是在不断变化的。

下面给出希望树的定义:

① 初始节点 S_0 在希望树 T 中。

② 如果节点 x 在希望树 T 中,则一定有

(i) 如果 x 是具有子节点 y_1, y_2, …, y_n 的或节点,则具有

$$\min_{1\leqslant i\leqslant n}\{c(x,y_i)+h(y_i)\}$$

值的那个子节点 y_i 也应在 T 中。

(ii) 如果 x 是与节点,则它的全部子节点都应在 T 中。

3. 与/或图的启发式搜索

与/或图启发式搜索的 open 表和 closed 表的节点域可定义如下:

j	S_j	F_k	and/or	y/n	h_j	P_j	i

其中,h_j 为节点 j 的启发函数值,P 为希望节点标志,如果节点 j 在希望树上,则有 $P_j=1$。其他域的定义与宽度优先搜索的节点域定义相同。

与/或图启发式搜索算法

(1) 初始化 open 表与 closed 表,把初始节点(序号 1)放入 open 表中,并设初始节点的希望标志 $P=1$。

(2) 若 open=( ),则算法失败终止;否则,转步骤(3)。

(3) 修改以初始节点为根的希望树,置希望树上的节点的 $P=1$;其他节点的 $P=0$。

(4) 把 open 表中尚未标示可解/不可解标志且希望标志 $P=1$ 的第一个节点 i 移出,放入 closed 表表尾。

(5) 若节点 i 可扩展(即在算符集中找到可用算符),则用可用算符作用于节点 i 的问题表示 S_i,生成其子问题节点;否则,转步骤(7)。

① 若子节点 j 的问题表示 S_j 既不在 open 表中，又不在 closed 表中，则节点 j 是一个新节点，将新节点 j 放入 open 表中，记载节点各域的值，转步骤(6)。

② 若子节点 j 的问题表示 S_j 已在 open 表中，即节点 j 与 open 表中的一个老节点 j' 的问题表示相同，则比较 h_j 和 $h_{j'}$。若 $h_j \geqslant h_{j'}$，则放弃新节点 j；若 $h_j < h_{j'}$，则将新节点 j 放入 open 表中，记载节点各域的值，并删除 open 表中的老节点 j'。转步骤(8)。

③ 若子节点 j 的问题表示 S_j 已在 closed 表中，即节点 j 与 closed 表中的一个老节点 j' 的问题表示相同，则比较 h_j 和 $h_{j'}$。若 $h_j \geqslant h_{j'}$，则放弃新节点 j；若 $h_j < h_{j'}$，则将新节点 j 放入 open 表中，记载节点各域的值，并删除 closed 表中的老节点 j' 以及 j' 在 open 表和 closed 表中的所有后裔节点。转步骤(8)。

(6) 若子节点 j 是终叶节点(即 S_j 是本原问题)，则标示节点 j 可解，并调用可解标示过程。若可解标示过程将初始节点标示为可解，则算法成功终止；否则，转步骤(8)。

(7) 若节点 i 不可扩展(即在算符集中找不到可用算符)，则标示节点 i 不可解，并调用不可解标示过程。若不可解标示过程将初始节点标示为不可解，则问题无解，算法终止；否则，删除节点 i 在 open 表中和 closed 表中的所有后裔节点，转步骤(8)。

(8) 对 open 表中的所有节点按 h 值从小到大的顺序重新排序，若有 2 个或 2 个以上的节点有相同的 h 值，则对这些节点再按节点序号排序。转步骤(2)。

3.3.3　与/或图搜索算法的应用

诸如下棋、打牌、战争等一类竞争性智能活动称为博弈。其中最简单的一种称为双方完备博弈。双方完备博弈是指：

① 对垒的 A、B 双方轮流采取行动，博弈的结果只有三种情况：A 方胜，B 方败；B 方胜，A 方败；双方战成平局。

② 在对垒过程中，任何一方都了解当前的格局及过去的历史。

③ 任何一方在采取行动前都要根据当前的实际情况进行得失分析，选取对自己最为有利而对对方最为不利的对策。

在博弈过程中，任何一方都希望自己取得胜利。因此，在某一方当前有多个行动方案可供选择时，他总是挑选对自己最有利而对对方最不利的那个行动方案。此时，如果站在 A 方的立场上，则可供 A 方选择的若干行动方案之间是或关系。但是，若 B 方也有若干个可供选择的行动方案，则对 A 方来说这些行动方案之间是与关系，因为这时主动权操在 B 方手里，A 方必须考虑任何一个可能被 B 方选中的行动方案。若把上述博弈过程用图表示出来，得到的是一棵与/或树。这里要特别指出，该与/或树是始终站在某一方(例如，A 方)的立场上得出的。称描述博弈过程的与/或树为博弈树，它有如下特点：

① 博弈的初始格局是初始节点。

② 在博弈树中，或节点和与节点是逐层交替出现的。自己一方扩展的节点之间是或关系，对方扩展的节点之间是与关系。双方轮流地扩展节点。

③ 所有能使自己一方获胜的终局都是本原问题，相应的节点是可解节点，所有使对方获

胜的终局都是不可解节点。

1. 极大极小分析法

在二人博弈问题中，为了从众多可供选择的行动方案中选出一个对自己有利的行动方案，就需要对当前情况以及将要发生的情况进行分析，从中选出最优者。最常使用的分析方法是极大极小分析法。其基本思想是

① 设博弈的双方中一方为 A，另一方为 B。极大极小分析法是为其中的一方（例如 A 方）寻找一个最优行动方案的方法。

② 为了找到当前的最优行动方案，需要对各个方案可能产生的后果进行比较。具体地说，就是要考虑每一方案实施后对方可能采取的所有行动，并计算可能的得分。

③ 为了计算得分，需要根据问题的特性信息定义一个估价函数，用来估算当前博弈树端节点的得分。此时，估算出来的得分称为静态估值。

④ 当端节点的估值计算出来后，再推算出父节点的得分。推算的方法是：对或节点，选其子节点中一个最大的得分作为父节点的得分，这是为了使自己在可供选择的方案中选一个对自己最有利的方案；对与节点，选其子节点中一个最小的得分作为父节点的得分，这是为了立足于最坏的情况。这样计算出的父节点的得分称为倒推值。

⑤ 如果一个行动方案能获得较大的倒推值，则它就是当前最好的行动方案。

博弈树的启发式搜索算法

(1) $k=1$，初始棋局 $S_k=S_1$。

(2) 如果棋局 S_k 是终叶节点棋局，则算法成功终止；否则，由棋局 S_k 生成 A 方所有可能的或关系子节点 S_i，$i=1,2,\cdots,n$。

(3) 对每一个或关系子节点 S_i，生成其 B 方所有可能的与关系子节点 S_j，$j=1,2,\cdots,m$。生成节点数为 $n\times m+1$ 的部分博弈树。

(4) 计算每个与关系子节点 S_j 的启发函数值 h_j。

(5) 分别由 m 个与关系子节点倒推计算其父节点（与节点）的启发函数值：

$$h_i=\min\{h_j \mid j=1,2,\cdots,m\}, \qquad i=1,2,\cdots,n$$

(6) 由 n 个或关系子节点倒推计算其父节点 S_k（或节点）的启发函数值：

$$h_k=\max\{h_i \mid i=1,2,\cdots,n\}$$

(7) A 方从 S_k 的 n 个或关系子节点中选择节点 i 作为最优行动方案，获得棋局 S_i。

(8) B 方从 S_i 的 m 个与关系子节点中选择节点 j 作为最优行动方案，获得棋局 S_j。

(9) 若节点 j 是端节点，则算法终止；否则，令 $k=j$，转步骤(2)。

在上述博弈树的启发式搜索算法中，步骤(2)～(6)生成 2 层节点组成的部分博弈树，并计算各节点的倒推值。步骤(7)从生成的 A 方所有可能的第一层 n 个或关系子节点中，自主选择倒推值最大的或关系子节点作为最优行动方案，从而获得最有利自己的实际棋局 S_i。步骤(8)表明 B 方只能面对棋局 S_i，从 S_i 的 m 个子节点中自主选择一个子节点作为 B 方的最优行动方案，由于在 A 方的博弈树中，已在步骤(5)考虑到 B 方所有可能的行动方案，其中包括对 A 方最不利的棋局（即对 B 方最有利的棋局），因此，无论 B 方如何选择自己的行动方案，都将

保证在这一回合(A、B 双方各走一步)中,A 方获得最大收益。

2. 博弈树启发式搜索求解示例

例 3.12　应用博弈树的启发式搜索算法求解一字棋游戏。棋盘如图 3.28 所示有 9 个空格,由 A、B 两人对弈,轮流由一方在棋盘上放上一枚自己的棋子,谁先使自己的棋子在棋盘上构成三子一线,谁就取得胜利。设 A 方先走棋。

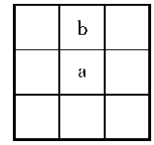

图 3.28　一字棋游戏

解　(1) 启发函数定义

A 方的启发函数定义为

$$h_i(S_i)=e_a(S_i)-e_b(S_i)$$

B 方的启发函数定义为

$$h_i(S_i)=e_b(S_i)-e_a(S_i)$$

其中,当 $e_a(S_i)$ 为形成棋局 S_i 时,A 方棋子 a 可能占满行、列和对角线的数目;$e_b(S_i)$ 为形成棋局 S_i 时,B 方棋子 b 可能占满行、列和对角线的数目。例如,如果棋局 S_i 如图 3.28 所示,则 $e_a(S_i)=6$,$e_b(S_i)=4$,那么,A 方的启发函数值为

$$h_i(S_i)=e_a(S_i)-e_b(S_i)=6-4=2$$

B 方的启发函数值为

$$h_i(S_i)=e_b(S_i)-e_a(S_i)=4-6=-2$$

(2) A 方博弈树

A 方先走棋,面对 $3\times3=9$ 个空格的棋盘,A 方可先下一子 a 于任一空格位置上,因此,初始棋局有 $n=9$ 个或关系子节点。每个或关系子节点的棋局 $S_i(i=1,2,\cdots,9)$ 都有 8 个空格,因此,每个 S_i 又都有 $m=8$ 个与关系子节点。也就是说,仅一个回合生成的节点数为 $n\times m+1=9\times8+1=73$ 个节点。

为了便于表述,仅在图 3.29 中给出其中部分节点组成的第一回合生成的博弈树。并约定图中的棋局 S_2、S_3 和 S_4 是初始棋局 S_1 的全部可能的或关系子节点棋局,S_5、S_6 和 S_7 是棋局 S_2 的全部可能的与关系子节点棋局,S_8、S_9 和 S_{10} 是棋局 S_3 的全部可能的与关系子节点棋局,

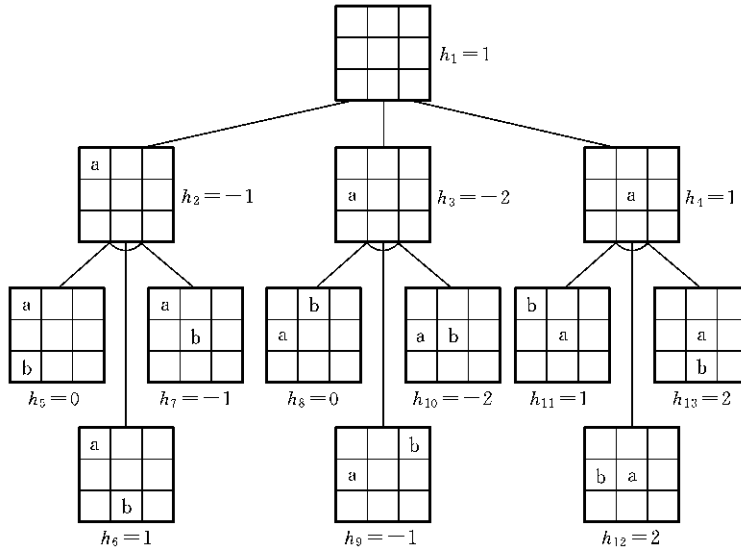

图 3.29　A 方第一回合的博弈树

S_{11}、S_{12}和 S_{13}是棋局 S_4的全部可能的与关系子节点棋局。

其中，与关系子节点的启发函数值分别是：$h_5(S_5)=e_a(S_5)-e_b(S_5)=5-5=0$，$h_6=6-5=1$，$h_7=4-5=-1$，$h_8=6-6=0$，$h_9=5-6=-1$，$h_{10}=4-6=-2$，$h_{11}=5-4=1$，$h_{12}=6-4=2$，$h_{13}=6-4=2$。

倒推计算或关系子节点的倒推值分别是

$$h_2=\min\{h_5,h_6,h_7\}=\min\{0,1,-1\}=-1$$

$$h_3=\min\{h_8,h_9,h_{10}\}=\min\{0,-1,-2\}=-2$$

$$h_4=\min\{h_{11},h_{12},h_{13}\}=\min\{1,1,2\}=1$$

倒推计算初始节点的倒推值是

$$h_1=\max\{h_2,h_3,h_4\}=\max\{-1,-2,1\}=h_4=1$$

因此，A 方由博弈树从初始棋局 S_1选择的最优行动方案是 S_4。

(3) B 方博弈树

B 方面对的棋局是 S_4，根据约定，S_{11}、S_{12}和 S_{13}是 S_4的全部可能的子节点棋局(实际上S_4的全部子节点棋局应有 8 个)，因此 B 方从 S_4只能生成这 3 个子节点，生成 B 方的第一回合博弈树如图 3.30 所示。

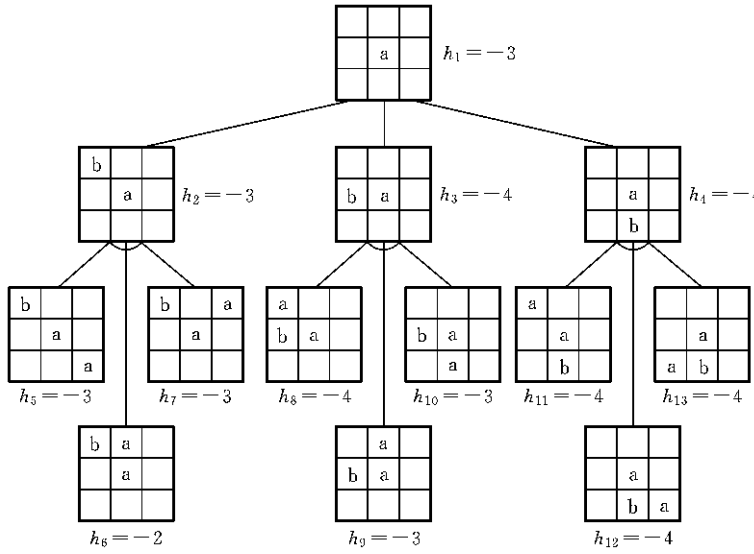

图 3.30 B 方第一回合的博弈树

其中，与关系子节点的启发函数值分别是：$h_5(S_5)=e_b(S_5)-e_a(S_5)=2-5=-3$，$h_6=3-5=-2$，$h_7=2-5=-3$，$h_8=2-6=-4$，$h_9=3-6=-3$，$h_{10}=3-6=-3$，$h_{11}=2-6=-4$，$h_{12}=2-6=-4$，$h_{13}=2-6=-4$。

倒推计算或关系子节点的倒推值分别是

$$h_2=\min\{h_5,h_6,h_7\}=\min\{-3,-2,-3\}=-3$$

$$h_3=\min\{h_8,h_9,h_{10}\}=\min\{-4,-3,-3\}=-4$$

$$h_4=\min\{h_{11},h_{12},h_{13}\}=\min\{-4,-4,-4\}=-4$$

倒推计算初始节点的倒推值是

$$h_1=\max\{h_2,h_3,h_4\}=\max\{-3,-4,-4\}=h_2=-3$$

因此，B 方由博弈树从棋局 S_1 选择的最优行动方案是 S_2。

在第二回合中，A 方面对的棋局是 B 方博弈树中的棋局 S_2，也就是 A 方博弈树中的棋局 S_7。A 方将以 S_7 作为 S_k 继续扩展它自己的博弈树；然后，B 方再扩展自己的博弈树，扩展过程类同第一回合，直至任何一方扩展生成终叶节点，算法成功终止。

习　题　三

3.1　何谓搜索？盲目搜索与启发式搜索的根本区别是什么？

3.2　用状态空间法表示问题时，什么是问题的解？什么是最优解？问题的最优解是唯一的吗？

3.3　在与/或树中，何谓端节点？何谓终叶节点？何谓可解节点？何谓不可解节点？

3.4　简述宽度优先搜索与深度优先搜索两者的差异。

3.5　简述宽度优先搜索与有代价的宽度优先搜索两者的差异。

3.6　试比较深度优先搜索、有代价的深度优先搜索与局部择优搜索三者之间的差异。

3.7　试比较宽度优先搜索、有代价的宽度优先搜索与全局择优搜索三者之间的差异。

3.8　请说明 A^* 算法对估价函数的限制。A^* 算法相对全局择优搜索的主要优点是什么？

3.9　何为搜索算法的可纳性？

3.10　简述与/或树中节点代价的计算方法。

3.11　与/或树启发式搜索算法为何需要生成希望树？希望树由哪些节点组成？

3.12　求解重排九宫问题。问题的初始状态 S_0 和目标状态 S_g 分别为

$$S_0=\begin{pmatrix}2&8&3\\1&&4\\7&6&5\end{pmatrix},\quad S_g=\begin{pmatrix}1&2&3\\8&&4\\7&6&5\end{pmatrix}$$

可使用的算符有：空格左移、空格上移、空格右移、空格下移。

(1) 应用局部择优搜索，画出搜索树，并给出问题的解。

(2) 应用全局择优搜索，画出搜索树，并给出问题的解。

3.13　应用全局择优搜索求解重排九宫问题。问题的初始状态 S_0 和目标状态 S_g 分别为

$$S_0=\begin{pmatrix}2&&3\\1&8&4\\7&6&5\end{pmatrix},\quad S_g=\begin{pmatrix}1&2&3\\8&&4\\7&6&5\end{pmatrix}$$

试画出搜索树，并给出问题的解。

3.14　设有下述火柴倒置问题：桌面上有 3 根并列的火柴，中间的一根火柴头向上，其余 2 根火柴头向下。规定每次操作为倒置其中一根火柴，最多可连续操作 3 次，使 3 根火柴都头向上或者都头向下。请画出火柴倒置问题的状态空间图，并给出所有可能的解和最优解。

提示：用三元组 $S=(x_1,x_2,x_3)$ 表示状态，其中，

$$x_i=\begin{cases}0 & \text{表示火柴 } i \text{ 头向下}\\1 & \text{表示火柴 } i \text{ 头向上}\end{cases}$$

3.15　若在图 3.24 所示的三阶梵塔问题中，移动 A 盘一次的代价是 1，移动 B 盘一次的代价是 3，移动 C 盘一次的代价是 4。试应用全局择优搜索求解，画出搜索树，并给出问题的解。

3.16　应用全局择优搜索求解推销员旅行问题：一个推销员从城市 A 出发，访问城市 B、C、D、E 和 F 一次且一次，最后回到城市 A，求最小交通费用的解。城市之间的交通图如图 3.31 所示，图中数字为城市之间的交通费用。试画出搜索树，并给出问题的解。

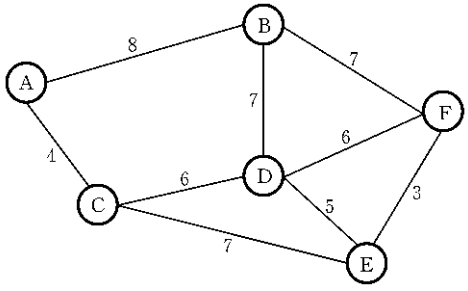

图 3.31 推销员旅行问题

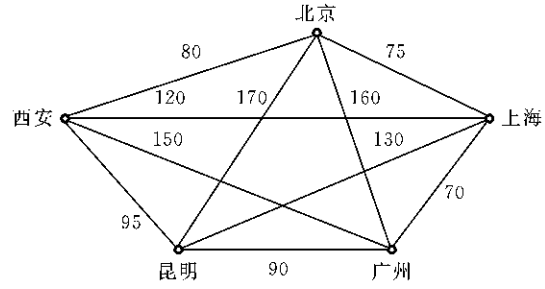

图 3.32 交通费用问题

3.17 五个城市之间的交通费用图如图 3.32 所示，边上的数字是两城市之间的交通费用。若从西安出发，经过每个城市一次且一次，最后到达广州，请找出一条交通费用最少的路线；选择状态空间方法中的一个适合的搜索方法求解，画出搜索树，并给出问题的解。

3.18 设有下述传教士与野人问题：有 3 名传教士和 3 名野人来到一条河的右岸，欲乘一条船渡河到左岸，该船的最大负载能力为 2 人，传教士或野人皆可撑船。在任何时候，不论是在右岸还是在左岸，如果野人人数超过传教士人数，那么，野人就会吃掉传教士。请选择状态空间方法中的一个合适的搜索方法，规划出一个渡河方案，把 6 个人都安全地渡过河去；画出搜索树，并给出问题的解。

3.19 设有下述农夫过河问题：农夫、狐狸、山羊和白菜都在一条河的左岸，只有农夫可撑船，要将狐狸、山羊和白菜都渡河到右岸去。但是，农夫每次撑船过河时，船上至多只能载狐狸，或者载山羊，或者载白菜。若农夫不在时，则狐狸会吃掉山羊、山羊会吃掉白菜。请选择状态空间方法中的一个合适的搜索方法规划出一个渡河方案，使农夫能把狐狸、山羊和白菜都带过河去；画出搜索树，并给出问题的解。

提示：用 4 元组 $S=(x_1, x_2, x_3, x_4)$ 表示状态，其中，变元 x_1，x_2，x_3 和 x_4 分别表示农夫、狐狸、山羊和白菜在状态 S 下所在的河岸。若其值为 0，则表示在左岸；若其值为 1，则表示在右岸。

3.20 设有如图 3.33 所示的与/或树，其中，$t_1 \sim t_5$ 是终叶节点。请分别应用与/或树的宽度优先搜索和与/或树的深度优先搜索求出解树。

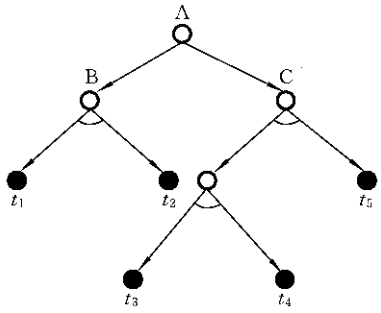

图 3.33 3.20 题与/或树

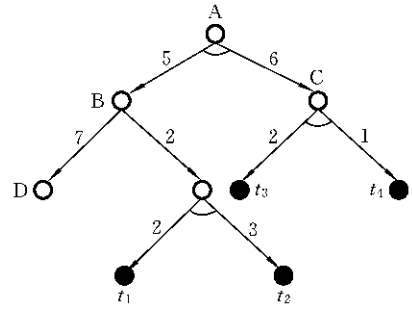

图 3.34 3.21 题与/或树

3.21 设有如图 3.34 所示的与/或树，其中，$t_1 \sim t_4$ 是终叶节点，边上的数字是该边的代价。请分别按和代价法与最大代价法求解树的代价。

第 4 章

经典逻辑推理

经典逻辑推理是根据经典逻辑(命题逻辑及一阶谓词逻辑)的逻辑规则进行的一种推理,主要推理方法有自然演绎推理、归结演绎推理及与/或形演绎推理等。由于这种推理是基于经典逻辑的,其真值只有“真”和“假”两种,因此它是一种精确推理,或称为确定性推理。

4.1 推理的基本概念

从已知的事实出发,通过运用已掌握的知识,找出其中蕴含的事实,或归纳出新的事实,这一过程通常称为推理。或者说,推理是按某种策略由已知判断推出另一判断的思维过程。实现推理的程序称为推理机。

4.1.1 推理方式及其分类

人类的智能活动有多种思维方式,人工智能作为对人类智能的模拟,相应也有多种推理方式,下面分别从不同的角度对它们进行讨论。

1. 演绎推理、归纳推理、默认推理

推理的基本任务是从一种判断推出另一种判断,从新判断推出的途径来分类,推理可分为演绎推理、归纳推理及默认推理。

演绎推理是从全称判断推导出特称判断或单称判断的过程,即由一般性知识推出适合于某一具体情况的结论。这是一种从一般到个别的推理。演绎推理有多种形式,经常用的是三段论式,它包括:

① 大前提,这是已知的一般性知识或假设;

② 小前提,这是关于所研究的具体情况或个别事实的判断;

③ 结论,这是由大前提推出的适合于小前提的新判断。

例如:

① 足球运动员的身体都是强壮的;

② 李波是一名足球运动员;

③ 所以,李波的身体是强壮的。

这就是一个三段论推理。其中:①是大前提;②是小前提;③是经演绎推出的结论。结论“李波的身体是强壮的”是蕴含于“足球运动员的身体都是强壮的”这一大前提之中的,它没有超出大

前提所断定的范围。在任何情况下，由演绎推理导出的结论都是蕴含在大前提的一般性知识之中的。只要大前提和小前提是正确的，由它们推出的结论也必然是正确的。演绎推理是人工智能中的一种重要推理方式，在目前研制成功的各类智能系统中，大多是用演绎推理实现的。

归纳推理是从足够多的事例中归纳出一般性结论的推理过程，是一种从个别到一般的推理。若从归纳时所选事例的广泛性来划分，归纳推理又可分为完全归纳推理与不完全归纳推理。完全归纳推理是指在进行归纳时考察了相应事物的全部对象，并根据这些对象是否都具有某种属性，从而推出这个事物是否具有这个属性。例如，某厂进行产品质量检查，如果对每一件产品都进行了严格检查，并且都是合格的，则推导出结论“该厂生产的产品是合格的”，这就是一个完全归纳推理。不完全归纳推理是指只考察了相应事物的部分对象，就得出了结论。例如，检查产品质量时，只是随机地抽查了部分产品，只要它们都合格，就得出“该厂生产的产品是合格的”结论，这就是一个不完全归纳推理。不完全归纳推理推出的结论不具有必然性，属于非必然性推理，而完全归纳推理是必然性推理。但由于要考察事物的所有对象通常比较困难，因而大多数归纳推理都是不完全归纳推理。归纳推理是人类思维活动中最基本、最常用的一种推理形式，人们在由个别到一般的思维过程中经常要用到它。

默认推理又称为缺省推理，它是在知识不完全的情况下假设某些条件已经具备所进行的推理。例如，在条件 A 已成立的情况下，如果没有足够的证据能证明条件 B 不成立，则默认 B 是成立的，并在此默认的前提下进行推理，推导出某个结论。由于这种推理允许默认某些条件是成立的，这就摆脱了需要知道全部有关事实才能进行推理的要求，使得在知识不完全的情况下也能进行推理。在默认推理过程中，如果某一时刻发现原先所作的默认不正确，就要撤消所作的默认及由此默认推出的结论，重新按新情况进行推理。

2. 确定性推理、不确定性推理

按推理时所用知识的确定性来分类，推理可分为确定性推理与不确定性推理。

确定性推理是指推理时所用的知识都是精确的，推出的结论也是确定的，其真值或者为真，或者为假，没有第三种情况出现。本章将要讨论的经典逻辑推理就属于这一类。

不确定性推理是指推理时所用的知识不都是精确的，推出的结论也不完全是肯定的，其真值位于真与假之间。现实世界中的事物和现象大都是不严格、不精确的，许多概念是模糊的，没有明确的类属界限。在此情况下，若仍用经典逻辑做精确处理，势必要人为地在本来没有明确界限的事物间划定界限，从而舍弃了事物固有的模糊性，失去了真实性。这就是为什么近年来各种非经典逻辑迅速崛起，人工智能亦把不精确知识的表示与处理作为重要研究课题的原因。

3. 单调推理、非单调推理

按推理过程中推出的结论是否单调地增加，或者说推出的结论是否越来越接近最终目标来分类，推理可分为单调推理与非单调推理。

单调推理是指在推理过程中随着推理过程向前推进及新知识的进入，推出的结论呈单调

增加的趋势，并且越来越接近最终目标，在推理过程中不会出现反复的情况，即不会由于新知识的加入否定了前面推出的结论，从而使推理又退回到前面的某一步。本章将要讨论的基于经典逻辑的演绎推理属于单调推理。

非单调推理是指在推理过程中由于新知识的加入，不仅没有加强已推出的结论，反而要否定它，使得推理退回到前面的某一步，重新开始。非单调推理多是在知识不完全的情况下发生的。由于知识不完全，为使推理进行下去，就要先做某些假设，并在此假设的基础上进行推理，当以后由于新知识的加入发现原先的假设不正确时，就需要推翻该假设及以此假设为基础推出的一切结论，再用新知识重新进行推理。显然，前面所说的默认推理是非单调推理。

4. 启发式推理、非启发式推理

按推理中是否运用与问题有关的启发性知识，推理可分为启发式推理与非启发式推理。启发性知识是指与问题有关且能加快推理进程、求得问题最优解的知识。

5. 基于知识的推理、直觉推理

从方法论的角度分类，推理可分为基于知识的推理与直觉推理。

基于知识的推理是根据已掌握的事实，通过运用知识进行推理。我们所讨论的推理都属于这一类。

直觉推理又称为常识性推理，是根据常识进行的推理。例如，当你从某建筑物下面走过时，突然发现有一物体从建筑物上掉下来，你立即会意识到“有危险”，并立即躲开，这就是使用了直觉推理。目前，在计算机上实现直觉推理还是一件很困难的工作，有待进行深入的研究。

除了上述分类方法外，推理还有一些其他分类方法，如根据推理的繁简不同，分为简单推理与复合推理；根据结论是否具有必然性，分为必然性推理与或然性推理；在不确定性推理中，又分为似然推理与近似推理或模糊推理，前者是基于概率论的推理，后者是基于模糊逻辑的推理。

4.1.2　推理的控制策略

推理过程是一个求解问题的过程。问题求解的质量与效率依赖于求解问题的策略，即推理的控制策略。推理的控制策略主要包括推理方向、搜索策略、冲突消解策略、求解策略及限制策略等。

1. 推理方向

推理方向用于确定推理的驱动方式，分为正向推理、逆向推理、混合推理及双向推理四种。无论按哪个方向进行推理，一般都要求系统具有一个存放知识的知识库、一个存放初始已知事实及问题状态的数据库和一个用于推理的推理机。

(1) 正向推理

正向推理是以已知事实作为出发点的推理，又称为数据驱动推理、前向链推理、模式制导

推理及前件推理等。

正向推理的基本思想是：从用户提供的初始已知事实出发，在知识库 KB 中找出当前可用的知识，构成可用知识集 KS，然后按某种冲突消解策略从 KS 中选出一条知识进行推理，并将推出的新事实加入到数据库 DB 中作为下一步推理的已知事实，在此之后再在知识库中选取可用知识进行推理，如此重复进行这一过程，直到求得了所要求的解或者知识库中再无可用的知识为止。其推理过程可描述如下：

① 将用户提供的初始已知事实送入数据库 DB。

② 检查数据库 DB 中是否已经包含了问题的解，若有，则求解成功终止；否则，执行下一步。

③ 根据数据库 DB 中的已知事实，扫描知识库 KB，检查 KB 中是否有可用（即可与 DB 中已知事实匹配）的知识，若有，则转步骤④；否则，转步骤⑥。

④ 把 KB 中所有的可用知识都选出来，构成可用知识集 KS。

⑤ 若 KS 不空，则按某种冲突消解策略从中选出一条知识进行推理，并将推出的新事实加入 DB 中，然后转步骤②；若 KS 空，则转步骤⑥。

⑥ 询问用户是否可进一步补充新的事实，若可补充，则将补充的新事实加入 DB 中，然后转步骤③；否则，（表示求不出解）失败退出。

上述推理过程在具体实现时还有许多工作要做。例如，在推理过程中要从知识库 KB 中选出可用知识，就要用知识库中的知识与数据库中的已知事实进行匹配，为此就需要确定匹配的方法。而且，匹配通常难以做到完全一致匹配，这就需要解决怎样才算是匹配成功的问题。其次，为了进行匹配，就要查找知识，这就牵涉到按什么路线进行查找的问题，即按什么策略搜索知识库。再如，如果可用的知识只有一条，系统立即就可用它进行推理，并将推出的新事实送入数据库 DB 中；但是，如果当前可用的知识有多条，应该先用哪一条？这是推理的一个重要问题，称为冲突消解策略。总之，为了实现正向推理，有许多具体问题需要解决，将分别对它们进行讨论。

(2) 逆向推理

逆向推理是以某个假设目标作为出发点的推理，又称为目标驱动推理、逆向链推理、目标制导推理及后件推理等。

逆向推理的基本思想是：首先选定一个假设目标，然后寻找支持该假设的证据，若所需的证据都能找到，则说明原假设是成立的；若无论如何都找不到所需要的证据，则说明原假设不成立，此时需要另外选定新的假设。其推理过程可描述如下：

① 提出要求证的目标（假设）。

② 检查该目标是否已在数据库 DB 中，若在，则该目标成立，成功退出推理或者对下一个假设目标进行验证；否则，转下一步。

③ 判断该目标是否是证据，即它是否为应由用户证实的原始事实，若是，则询问用户；否则，转下一步。

④ 在知识库 KB 中找出所有能导出该目标的知识，形成可用知识集 KS，然后转下一步。

⑤ 从 KS 中选出一条知识，并将该知识的运用条件作为新的假设目标，然后转步骤②。

与正向推理相比，逆向推理更复杂一些，上述过程只是描述了它的大致过程，许多细节也没有反映出来。

逆向推理的主要优点是不必使用与目标无关的知识，目的性强。其主要缺点是初始目标的选择有盲目性，若不符合实际，就要多次提出假设，也会影响到系统的效率。

(3) 混合推理

正向推理可能会推出许多与问题求解无关的子目标；逆向推理中，若提出的假设目标不符合实际，也会降低系统的效率。可把正向推理与逆向推理结合起来，使其各自发挥自己的优势，取长补短，既有正向又有逆向的推理称为混合推理。在下述几种情况下，通常需要进行混合推理：

① 已知的事实不充分。当数据库中的已知事实不够充分时，若用这些事实与知识的运用条件匹配进行正向推理，则有可能一条适用知识也选不出来，这就使推理无法进行下去。此时，可通过正向推理先把其运用条件不能完全匹配的知识都找出来，并把这些知识可导出的结论作为假设，然后分别对这些假设进行逆向推理。由于在逆向推理中可以向用户询问有关证据，故有可能使推理进行下去。

② 由正向推理推出的结论可信度不高。用正向推理进行推理时，虽然推出了结论，但可信度可能不高，达不到预定的要求。此时，为了得到一个可信度符合要求的结论，可用这些结论作为假设，然后进行逆向推理，通过向用户询问进一步的信息，有可能会得到可信度较高的结论。

③ 希望得到更多的结论。在逆向推理过程中，由于可与用户进行对话，有针对性地向用户提问，这就有可能获得一些原来未曾掌握的有用信息，这些信息不仅可用于证实要证明的假设，同时还有助于推出一些其他结论。因此，在用逆向推理证实了某个假设之后，可以再用正向推理推出另外一些结论。例如，在医疗诊断系统中，先用逆向推理证实某病人患有某种病，然后利用逆向推理过程中获得的信息进行正向推理，就有可能推出该病人还患有别的什么病。

由以上讨论可以看出，混合推理分为两种情况：一种是先进行正向推理，帮助选择某个目标，即从已知事实演绎出部分结果，然后用逆向推理证实该目标或提高其可信度；另一种是先假设一个目标进行逆向推理，然后利用逆向推理中得到的信息进行正向推理，以推出更多的结论。

(4) 双向推理

双向推理是指正向推理与逆向推理同时进行，且在推理过程中的某一步骤上“碰头”的一种推理。其基本思想是：一方面根据已知事实进行正向推理，但并不推到最终目标；另一方面从某假设目标出发进行逆向推理，但并不推至原始事实，而是让它们在中途相遇，即由正向推理所得的中间结论恰好是逆向推理此时所要求的证据，这时推理就可结束，逆向推理的假设就是推理的最终结论。

双向推理的困难在于“碰头”的判断。另外，如何权衡正向推理与逆向推理的比重，即如何确定“碰头”的时机也是一个困难问题。

2. 求解策略

推理的求解策略是指，推理是只求一个解，还是求所有解或最优解等。例如，前述的正向

推理只用于求一个解，只要略加修改就可用来求所有解。

3. 限制策略

为了防止无穷的推理过程，或者由于推理过程太长从而增加时间及空间的复杂性，可在控制策略中指定推理的限制条件，以对推理的深度、宽度、时间、空间等进行限制。

4. 冲突消解策略

在推理过程中，系统要不断地用当前已知的事实与知识库中的知识进行匹配，此时可能发生如下三种情况：

① 已知事实不能与知识库中的任何知识匹配成功。

② 已知事实恰好与知识库中的一个知识匹配成功。

③ 已知事实可与知识库中的多个知识匹配成功，或者多个已知事实都可与知识库中某一个知识匹配成功，或者多个已知事实可与知识库中的多个知识匹配成功。

当第一种情况发生时，由于找不到与当前已知事实匹配成功的知识，故使推理无法继续进行下去，这或者是由于知识库中缺少某些必要的知识，或者是由于欲求解的问题超出了系统的功能范围等，此时可根据当时的实际情况做相应的处理。对于第二种情况，由于匹配成功的知识只有一个，所以它就是可用知识，可直接把它用于当前的推理。第三种情况不仅有知识匹配成功，而且有多个知识匹配成功，称这种情况为发生了可用知识冲突。此时需要按一定策略解决冲突，以便从中挑选一个可用知识用于当前的推理，称这一解决冲突的过程为冲突消解。解决冲突所用的方法称为冲突消解策略。

例如，在产生式系统中，若出现如下情况就认为发生了冲突：

① 对正向推理而言，如果有多条产生式规则的前件都和已知事实匹配成功，或者有多组不同的已知事实与同一条产生式规则的前件匹配成功，或者以上两种情况同时出现。

② 对逆向推理而言，如果有多条产生式规则的后件都和同一个假设匹配成功，或者有多条产生式规则的后件可与多个假设匹配成功。

冲突消解的任务是解决冲突。对正向推理来说，它将决定选择哪一组已知事实来激活哪一条产生式规则，产生其后件指出的结论或执行后件指定的操作。对逆向推理来说，它将决定用哪一个假设与哪一个产生式规则的后件进行匹配，推出相应的前件作为新的假设。

目前已有多种消解冲突的策略，其基本思想都是对知识进行排序，常用的有以下几种。

(1) 按针对性排序

设有如下两条产生式规则：

$$R_1: \text{if}\quad A_1\quad \text{and}\quad A_2\quad \text{and}\quad \cdots\quad \text{and}\quad A_n\ \text{then}\quad H_1$$

$$R_2: \text{if}\quad B_1\quad \text{and}\quad B_2\quad \text{and}\quad \cdots\quad \text{and}\quad B_m\ \text{then}\quad H_2$$

如果 R_2 中除了包含 R_1 的全部条件 $A_1, A_2, \cdots, A_n$ 外，还包含其他条件，则称 R_2 比 R_1 有更大的针对性，R_1 比 R_2 有更大的通用性。

本策略是优先选用针对性较强的产生式规则。因为它要求的条件较多，其结论一般更接近于目标，一旦得到满足，可缩短推理过程。

(2) 按匹配度排序

在不确定性匹配中，为了确定两个知识模式是否可以匹配，需要计算这两个模式的相似程度，当其相似度达到某个预先规定的值时，就认为它们是可匹配的。相似度又称为匹配度，它除了可用来确定两个知识模式是否可匹配外，还可用于冲突消解。若产生式规则 R_1 与 R_2 都可匹配成功，则可根据它们的匹配度来决定哪一个产生式规则可优先被使用。

(3) 根据领域问题的特点排序

如果事先可知道领域问题的某些特点，则可根据这些特点事先确定知识库中知识的使用顺序。例如：

① 当领域问题有固定的求解次序时，可按该次序对知识库中的知识排序，排在前面的知识优先被使用。

② 当已知某些产生式规则被使用后会明显地有利于问题的求解时，可对这些产生式规则指定较高的优先级，使这些产生式规则优先被使用。

4.1.3　模式匹配及其变量代换

模式匹配是指两个知识模式(如两个谓词公式、两个框架片断等)的比较，检查这两个知识模式是否完全一致或近似一致。如果两者完全一致，或者虽不完全一致但其相似程度落在指定的限度内，就称它们是可匹配的，否则为不可匹配。

模式匹配是推理中必须进行的一项重要工作，因为只有经过模式匹配才能从知识库中选出当前适用的知识，才能进行推理。例如，在产生式系统中，为了由已知的初始事实推出相应的结论，首先必须从知识库中选出可与已知事实匹配的产生式规则，然后才能应用这些产生式规则进行推理，逐步推出结论。框架推理与此类似，也需要先通过匹配选出相应的框架片断，然后再进行推理。

按匹配时两个知识模式的相似程度划分，模式匹配可分为确定性匹配与不确定性匹配。

确定性匹配是指两个知识模式完全一致，或者经过变量代换后变得完全一致。例如，设有如下两个知识模式：

P_1：Father(李四，李小四)and Man (李小四)

P_2：Father(x，y) and Man (y)

若用“李四”代换变量 x，用“李小四”代换变量 y，则 P_1 与 P_2 就变得完全一致。若用这两个知识模式进行匹配，则它们是确定性匹配。确定性匹配又称为完全匹配或精确匹配。

不确定性匹配是指两个知识模式不完全一致，但从总体上看，它们的相似程度又落在规定的限度内。

无论是确定性匹配还是不确定性匹配，在匹配时一般都需要进行变量的代换。

定义 4.1　代换是形如

$$\{t_1/x_1, t_2/x_2, \cdots, t_n/x_n\}$$

的有限集合。其中，$t_1, t_2, \cdots, t_n$ 是项；$x_1, x_2, \cdots, x_n$ 是互不相同的变元；t_i/x_i 表示用 t_i 代换 x_i，不允许 t_i 与 x_i 相同，也不允许变元 x_i 循环地出现在另一个 t_j 中。

例如，

$$\{g(y)/x, f(x)/y\}$$

不是一个代换，因为代换的目的是使某些变元被另外的变元、常量或函数取代，使之不再在公式中出现，而$\{g(y)/x, f(x)/y\}$在 x 与 y 之间出现了循环代换的情况，它既没有消去 x，也没有消去 y。如果将它改为

$$\{g(A)/x, f(x)/y\}$$

就可以，它将把公式中的 x 用 $g(A)$代换，y 用 $f(g(A))$代换，从而消去了变元 x 和 y。

定义 4.2 设

$$\theta=\{t_1/x_1, t_2/x_2, \cdots, t_n/x_n\}$$
$$\lambda=\{u_1/y_1, u_2/y_2, \cdots, u_m/y_m\}$$

是两个代换，则此两个代换的复合也是一个代换，它是从

$$\{t_1\lambda/x_1, t_2\lambda/x_2, \cdots, t_n\lambda/x_n, u_1/y_1, u_2/y_2, \cdots, u_m/y_m\}$$

中删去如下两种元素：

$t_i\lambda/x_i$　　当 $t_i\lambda = x_i$

u_i/y_i　　当 $y_i \in \{x_1, x_2, \cdots, x_n\}$

后剩下的元素所构成的集合，记为 $\theta \circ \lambda$。

例如，设有代换

$$\theta = \{f(y)/x, z/y\}$$
$$\lambda = \{A/x, B/y, y/z\}$$

则

$$\theta \circ \lambda = \{f(B)/x, y/z\}$$

定义 4.3 设有公式集 $F=\{F_1, F_2, \cdots, F_n\}$，若存在一个代换 λ 使得

$$F_1\lambda = F_2\lambda = \cdots = F_n\lambda$$

则称 λ 为公式集 F 的一个合一，且称 $F_1, F_2, \cdots, F_n$ 是可合一的。

例如，设有公式集

$$F=\{\ P(x, y, f(y)), P(A, g(x), z)\ \}$$

则下式是它的一个合一：

$$\lambda=\{A/x, g(A)/y, f(g(A))/z\}$$

一个公式集的合一通常是不唯一的。

定义 4.4 设 σ 是公式集 F 的一个合一，如果对任一个合一 θ 都存在一个代换 λ，使得

$$\theta = \sigma \circ \lambda$$

则称 σ 是公式集 F 的最一般合一(mgu)。

最一般合一是唯一的。若用最一般合一去代换那些可合一的谓词公式，则可使它们变成完全一致的谓词公式。由此可知，为了使两个知识模式匹配，可用其最一般合一对它们进行代换。

在给出求取最一般合一的算法之前，先引入差异集的概念。设有如下两个谓词公式：

$$F_1:\quad P(x, y, z)$$
$$F_2:\quad P\ (x, f\ (A), h(B)\ )$$

分别从 F_1 与 F_2 的第一个符号开始，逐个向右比较，此时发现 F_1 中 y 与 F_2 中的 $f(A)$ 不同，它们构成了一个差异集：

$$D_1=\{y,f(A)\}$$

当继续向右边比较时，又发现 F_1 中的 z 与 F_2 中的 $h(B)$ 不同，则又得到一个差异集：

$$D_2=\{z,h(B)\}$$

下面，对任意公式集 F，给出求最一般合一的算法。

求最一般合一算法

(1) 初始化，令 $k=0$，$F_k=F$，$\sigma_k=\varnothing$。其中，$\varnothing$ 是代换空集。

(2) 若 F_k 只含一个表达式，则算法成功终止，σ_k 就是最一般合一；否则，转步骤(3)。

(3) 找出 F_k 的差异集 D_k。

(4) 若 D_k 中存在变元 x_k 和项 t_k，且 x_k 不在 t_k 中出现，则

$$\sigma_{k+1}=\sigma\circ\{t_k/x_k\}$$
$$F_{k+1}=F_k\{t_k/x_k\}$$
$$k=k+1$$

转步骤(2)；否则，算法终止，F 的最一般合一不存在。

例 4.1　设有公式集

$$F=\{P(A,x,f(g(y))),P(z,f(z),f(u))\}$$

求其最一般合一。

解　初始化，令 $k=0$，$\sigma_0=\varnothing$，$F_0=F=\{P(A,x,f(g(y))),P(z,f(z),f(u))\}$

Loop 1：F_0 含有 2 个表达式，故 σ_0 不是最一般合一，

F_0 的差异集 $D_0=\{A,z\}$，可有代换 A/z，

$\sigma_1=\sigma_0\circ\{A/z\}=\{A/z\}$

$F_1=F_0\{A/z\}=\{P(A,x,f(g(y))),P(A,f(A),f(u))\}$

Loop 2：F_1 含有 2 个表达式，故 σ_1 不是最一般合一，

F_1 的差异集 $D_1=\{x,f(A)\}$，可有代换 $\{f(A)/x\}$，

$\sigma_2=\sigma_1\circ\{f(A)/x\}=\{A/z\}\circ\{f(A)/x\}=\{A/z,f(A)/x\}$

$F_2=F_1\{f(A)/x\}=\{P(A,f(A),f(g(y))),P(A,f(A),f(u))\}$

Loop 3：F_2 含有 2 个表达式，故 σ_2 不是最一般合一；

F_2 的差异集 $D_2=\{g(y),u\}$，可有代换 $\{g(y)/u\}$，

$\sigma_3=\sigma_2\circ\{g(y)/u\}=\{A/z,f(A)/x\}\circ\{g(y)/u\}=\{A/z,f(A)/x,g(y)/u\}$

$F_3=F_2\{g(y)/u\}=\{P(A,f(A),f(g(y))),P(A,f(A),f(g(y)))\}$

$=\{P(A,f(A),f(g(y)))\}$

Loop 4：F_3 中只含有一个表达式，故算法成功终止

$$\sigma_3=\{A/z,f(A)/x,g(y)/u\}$$

为公式集 F 的最一般合一。

4.2 归结演绎推理

自动定理证明是人工智能的一个重要研究领域，定理证明的实质是对已知前提 P 和待证结论 Q，证明 $P \rightarrow Q$ 的永真性。但是，正如在第 2 章所讨论的那样，要证明一个谓词公式的永真性是相当困难的，甚至在某些情况下是不可能的。通过研究发现，应用反证法的思想可把关于永真性的证明转化为不可满足性的证明，即欲证明 $P \rightarrow Q$ 永真，只要证明 $P \wedge \neg Q$ 是不可满足的就可以了。关于不可满足性的证明，海伯伦(Herbrand)及鲁宾逊(Robinson)先后进行了卓有成效的研究，提出了相应的理论和方法。海伯伦的理论和鲁宾逊的归结原理，都是以子句集为背景开展研究的。

4.2.1 谓词公式化为子句集的方法

定义 4.5 在谓词逻辑中，把原子谓词公式及其否定统称为文字。任何文字的析取式称为子句。

例如，$P(x) \vee Q(x)$，$\neg P(x, f(x)) \vee Q(x, g(x))$ 都是子句。

定义 4.6 不包含任何文字的子句称为空子句。

由于空子句不含有文字，它不能被任何解释满足，所以空子句是永假的、不可满足的。

由子句构成的集合称为子句集。在谓词逻辑中，任何一个谓词公式都可通过应用等价关系及推理规则化成相应的子句集。

把谓词公式化成子句集的步骤如下。

① 消去蕴含连词。利用下述等价关系消去谓词公式中的“$\rightarrow$”

$$P \rightarrow Q \Leftrightarrow \neg P \vee Q$$

② 减小否定连词的辖域。利用下述等价关系把“$\neg$”移到紧靠谓词的位置上：

$$\neg(\neg P) \Leftrightarrow P$$

$$\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q$$

$$\neg(P \vee Q) \Leftrightarrow \neg P \wedge \neg Q$$

$$\neg(\forall x) P \Leftrightarrow (\exists x) \neg P$$

$$\neg(\exists x) P \Leftrightarrow (\forall x) \neg P$$

③ 约束变元标准化。重新命名约束变元名，使不同量词约束的变元有不同的名字。

④ 消去存在量词。若存在量词不在全称量词的辖域内，则用一个新的个体常量替换受该存在量词约束的变元就可消去存在量词(因为若原公式为真，则总能找到一个个体常量替换，替换后仍使公式为真)；若存在量词位于一个或多个全称量词的辖域内，例如，

$$(\forall x_1)(\forall x_2)\cdots(\forall x_n)(\exists y) P(x_1, x_2, \cdots, x_n, y)$$

则需要用 Skolem 函数 $f(x_1, x_2, \cdots, x_n)$ 替换受该存在量词约束的变元 y，然后才能消去存在量词。

⑤ 组成全称量词前缀。把全称量词全部移到公式的左边。

⑥ 把母式化为 Skolem 标准形。利用下述等价关系：

$$P \vee (Q \wedge R) \Leftrightarrow (P \vee Q) \wedge (P \vee R)$$

把公式化为 Skolem 标准形。

Skolem 标准形的一般形式是

$$(\forall x_1)(\forall x_2)\cdots(\forall x_n)M$$

其中，M 是子句的合取式，称为 Skolem 标准形母式。

⑦ 消去全称量词。

⑧ 对变元更名，使不同子句中的变元不同名。

⑨ 消去合取连词，得到子句集。

例 4.2　请将下述谓词公式化为子句集：

$$(\forall x)((\forall y)P(x,y)) \rightarrow \neg(\forall y)(Q(x,y) \rightarrow R(x,y))$$

解

$$(\forall x)((\forall y)P(x,y)) \rightarrow \neg(\forall y)(Q(x,y) \rightarrow R(x,y))$$

$$\Leftrightarrow (\forall x)(\neg(\forall y)P(x,y) \vee \neg(\forall y)(\neg Q(x,y) \vee R(x,y)))$$

$$\Leftrightarrow (\forall x)((\exists y)\neg P(x,y) \vee (\exists y)(Q(x,y) \wedge \neg R(x,y)))$$

$$\Leftrightarrow (\forall x)((\exists y)\neg P(x,y) \vee (\exists z)(Q(x,z) \wedge \neg R(x,z)))$$

$$\Leftrightarrow (\forall x)(\neg P(x,f(x)) \vee (Q(x,g(x)) \wedge \neg R(x,g(x))))$$

$$\Leftrightarrow (\forall x)((\neg P(x,f(x)) \vee Q(x,g(x))) \wedge (\neg P(x,f(x)) \vee \neg R(x,g(x))))$$

$$\Leftrightarrow (\neg P(x,f(x)) \vee Q(x,g(x))) \wedge (\neg P(x,f(x)) \vee \neg R(x,g(x)))$$

$$\Leftrightarrow (\neg P(x,f(x)) \vee Q(x,g(x))) \wedge (\neg P(y,f(y)) \vee \neg R(y,g(y)))$$

消去合取词后，上式就变为下述子句集：

$$\neg P(x,f(x)) \vee Q(x,g(x))$$

$$\neg P(y,f(y)) \vee R(y,g(y))$$

如果谓词公式是不可满足的，则等价变换得到的子句集也一定是不可满足的，反之亦然。因此，在不可满足的意义上两者是等价的，由此给出下述定理。

定理 4.1　设有谓词公式 F，其标准形的子句集为 S，则 F 不可满足的充要条件是 S 不可满足。

由此定理可知，要证明一个谓词公式是不可满足的，只要证明相应的子句集是不可满足的就可以了。

4.2.2　归结原理

由谓词公式转化子句集的过程可以看出，子句集中的子句之间是合取关系，其中只要有一个子句不可满足，则子句集就不可满足。另外，已经指出空子句是不可满足的。因此，若一个子句集中包含空子句，则这个子句集一定是不可满足的。鲁宾逊归结原理就是基于这一认识提出来的。其基本思想是：检查子句集 S 中是否包含空子句，若包含，则 S 不可满足；若不包含，就在子句集中选择合适的子句进行归结，一旦通过归结能推出空子句，就说明子句集 S 是

不可满足的。

定义 4.7 若 P 是原子谓词公式，则称 P 与 $\neg P$ 为互补文字。

1. 命题逻辑中的归结原理

定义 4.8 设 C_1 与 C_2 是子句集中的任意两个子句，如果 C_1 中的文字 L_1 与 C_2 中的文字 L_2 互补，那么从 C_1 和 C_2 中分别消去 L_1 和 L_2，并将两个子句中余下的部分析取，构成一个新子句 C_{12}，则称这一过程为归结，称 C_{12} 为 C_1 和 C_2 的归结式，称 C_1 和 C_2 为 C_{12} 的亲本子句。

定理 4.2 归结式 C_{12} 是其亲本子句 C_1 与 C_2 的逻辑结论。

证明 设

$$C_1 = L \vee C_1', \quad C_2 = \neg L \vee C_2'$$

通过归结可以得到

$$C_{12} = C_1' \vee C_2'$$

C_1 和 C_2 是 C_{12} 的亲本子句。因为

$$C_1' \vee L \Leftrightarrow C_1' \rightarrow L$$

$$\neg L \vee C_2' \Leftrightarrow L \rightarrow C_2'$$

所以

$$C_1 \wedge C_2 = (\neg C_1' \rightarrow L) \wedge (L \rightarrow C_2')$$

根据假言三段论得到

$$(\neg C_1' \rightarrow L) \wedge (L \rightarrow C_2') \Leftrightarrow \neg C_1' \rightarrow C_2'$$

因为

$$\neg C_1' \rightarrow C_2' \Leftrightarrow C_1' \vee C_2' = C_{12}$$

所以

$$C_1 \wedge C_2 \Leftrightarrow C_{12}$$

由第 2 章关于逻辑结论的定义 2.6，可知 C_{12} 是其亲本子句 C_1 和 C_2 的逻辑结论。

这个定理是归结原理中的一个很重要的定理，由它可得到如下两个推论。

推论 4.1 设 C_1 与 C_2 是子句集 S 中的两个子句，C_{12} 是它们的归结式，若用 C_{12} 代替 C_1 和 C_2 后得到新子句集 S_1，则由 S_1 的不可满足性可推出原子句集 S 的不可满足性，即

$$S_1 \text{的不可满足性} \Rightarrow S \text{的不可满足性}$$

推论 4.2 设 C_1 与 C_2 是子句集 S 中的两个子句，C_{12} 是它们的归结式，若把 C_{12} 加入 S 中，得到新子句集 S_2，则 S 与 S_2 在不可满足的意义上是等价的，即

$$S_2 \text{的不可满足性} \Leftrightarrow S \text{的不可满足性}$$

这两个推论给出了用归结原理证明子句集不可满足性的基本思想：为了要证明子句集 S 的不可满足性，只要对其中可进行归结的子句进行归结，并把归结式加入子句集 S，或者用归结式替换它的亲本子句，然后对新子句集（S_1 或 S_2）证明不可满足性就可以了。如果经过归结能得到空子句，根据空子句的不可满足性，可得出原子句集 S 是不可满足的。

在命题逻辑中，对不可满足的子句集 S，归结原理是完备的，即：若子句集不可满足，则必然存在一个从 S 到空子句的归结演绎；若存在一个从 S 到空子句的归结演绎，则 S 一定是不可满足的。但是，对于可满足的子句集 S，用归结原理得不到任何结果。

2. 谓词逻辑中的归结原理

在谓词逻辑中，由于子句中含有变元，所以不可直接消去互补文字，而需要先用最一般合

一对变元进行代换，然后才能进行归结。例如，设有如下两个子句：

$$C_1 = P(x) \vee Q(x)$$

$$C_2 = \neg P(A) \vee R(y)$$

由于 $P(x)$与 $P(A)$不同，所以 C_1与 C_2不能直接进行归结，但若用最一般合一

$$\sigma = \{A/x\}$$

对两个子句分别进行代换：

$$C_1\sigma = P(A) \vee Q(A), \quad C_2\sigma = \neg P(A) \vee R(y)$$

就可对它们进行归结，消去 $P(A)$与 $\neg P(A)$，得到如下归结式：

$$Q(A) \vee R(y)$$

下面给出谓词逻辑中关于归结的定义。

定义 4.9　设 C_1与 C_2是两个没有相同变元的子句，L_1和 L_2分别是 C_1和 C_2中的文字，若 σ是 L_1和 $\neg L_2$的最一般合一，则称

$$C_{12} = (C_1\sigma - \{L_1\sigma\}) \bigcup (C_2\sigma - \{L_2\sigma\})$$

为 C_1和 C_2的二元归结式，L_1和 L_2称为归结式的文字。

例 4.3　设

$$C_1 = P(A) \vee \neg Q(x) \vee R(x)$$

$$C_2 = \neg P(y) \vee Q(B)$$

给出 C_1和 C_2的归结式。

解　若选 $L_1 = P(A)$，$L_2 = \neg P(y)$，则 $\sigma = \{A/y\}$是 L_1与 $\neg L_2$的最一般合一，根据定义 4.9，可得

$$\begin{aligned} C_{12} &= (C_1\sigma - \{L_1\sigma\}) \bigcup (C_2\sigma - \{L_2\sigma\}) \\ &= (\{P(A), \neg Q(x), R(x)\} - \{P(A)\}) \bigcup (\{\neg P(A), Q(B)\} - \{\neg P(A)\}) \\ &= (\{\neg Q(x), R(x)\}) \bigcup (\{Q(B)\}) \\ &= \{\neg Q(x), R(x), Q(B)\} \\ &= \neg Q(x) \vee R(x) \vee Q(B) \end{aligned}$$

上述归结过程可以用归结树表示如图 4.1 所示。

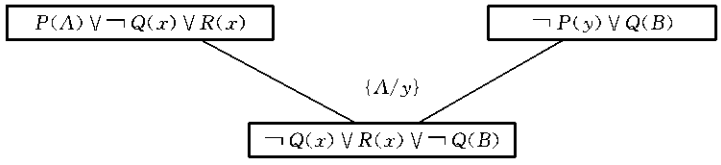

图 4.1　例 4.3 的一种归结树

若选 $L_1 = \neg Q(x)$，$L_2 = Q(B)$，$\sigma = \{B/x\}$，则可得

$$\begin{aligned} C_{12} &= (\{P(A), \neg Q(B), R(B)\} - \{\neg Q(B)\}) \bigcup (\{\neg P(y), Q(B)\} - \{Q(B)\}) \\ &= (\{P(A), R(B)\}) \bigcup (\{\neg P(y)\}) \\ &= \{P(A), R(B), \neg P(y)\} \\ &= P(A) \vee R(B) \vee \neg P(y) \end{aligned}$$

上述归结过程的归结树如图 4.2 所示。

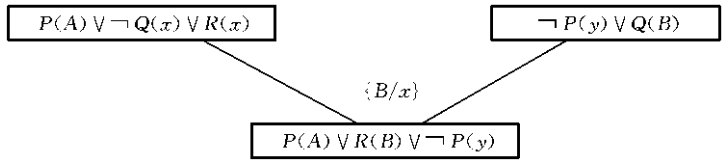

图 4.2 例 4.3 的另一种归结树

由此可见，谓词逻辑中含有变元，对 2 个子句中的变元选择不同的代换，得出的归结式是不同的。

例 4.4 设

$$C_1=P(x)\vee Q(A),\quad C_2=\neg P(B)\vee R(x)$$

给出 C_1 和 C_2 的归结式。

解 由于 C_1 与 C_2 有相同的变元，不符合定义 4.9 的要求。为了进行归结，需修改 C_2 中变元的名字，令 $C_2=\neg P(B)\vee R(y)$。此时，对 C_1 和 C_2 有

$$L_1=P(x),\quad L_2=\neg P(B)$$

L_1 与 $\neg L_2$ 的最一般合一 $\sigma=\{B/x\}$，则

$$\begin{aligned}C_{12}&=(\{P(B),Q(A)\}-\{P(B)\})\cup(\{\neg P(B),R(y)\}-\{\neg P(B)\})\\&=\{Q(A),R(y)\}\\&=Q(A)\vee R(y)\end{aligned}$$

可以用归结树直接表示 C_1 和 C_2 的归结，如图 4.3 所示。

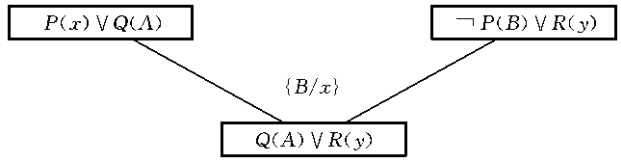

图 4.3 例 4.4 的归结树

如果在参加归结的子句内部含有可合一的文字，则在进行归结之前应对这些文字先进行合一。

例 4.5 设有如下两个子句：

$$C_1=P(x)\vee P(f(A))\vee Q(x)$$
$$C_2=\neg P(y)\vee R(B)$$

在 C_1 中有可合一的文字 $P(x)$ 与 $P(f(A))$，若用它们的最一般合一 $\theta=\{f(A)/x\}$ 进行代换，得到

$$C_1\theta=P(f(A))\vee Q(f(A))$$

此时可对 $C_1\theta$ 和 C_2 进行归结，从而得到 C_1 与 C_2 的二元归结式。

对 $C_1\theta$ 和 C_2 分别选 $L_1=P(f(A))$，$L_2=\neg P(y)$。L_1 和 $\neg L_2$ 的最一般合一是 $\sigma=\{f(A)/y\}$，则

$$C_{12}=R(B)\vee Q(f(A))$$

在上例中，把 $C_1\theta$ 称为 C_1 的因子。一般来说，若子句 C 中有两个或两个以上的文字具有最一般合一 σ，则称为 $C\sigma$ 为子句 C 的因子。如果 $C\sigma$ 是一个单文字，则称它为 C 的单元因子。

应用因子的概念，可对谓词逻辑中的归结原理给出如下定义。

定义 4.10　子句 C_1 和 C_2 的归结式是下列二元归结式之一：

① C_1 与 C_2 的二元归结式；

② C_1 与 C_2 的因子 $C_2\sigma_2$ 的二元归结式；

③ C_1 的因子 $C_1\sigma_1$ 与 C_2 的二元归结式；

④ C_1 的因子 $C_1\sigma_1$ 与 C_2 的因子 $C_2\sigma_2$ 的二元归结式。

对于谓词逻辑，定理 4.2 仍然适用，即归结式是它的亲本子句的逻辑结论。用归结式取代它在子句集 S 中的亲本子句所得到的新子句仍然保持着原子句集 S 的不可满足性。

另外，对于一阶谓词逻辑，从不可满足的意义上说，归结原理也是完备的，即：若子句集是不可满足的，则必存在一个从该子句集到空子句的归结演绎；若从子句集存在一个到空子句的演绎，则该子句集是不可满足的。

4.2.3　归结反演

为了证明 Q 为 $P_1, P_2, \cdots, P_n$ 的逻辑结论，根据定理 2.1 只需证明谓词公式

$$(P_1 \wedge P_2 \wedge \cdots \wedge P_n) \wedge \neg Q$$

是不可满足的；根据定理 4.1，在不可满足的意义上，该谓词公式与其子句集是等价的。可用归结原理来进行子句集的不可满足性的证明，从而证明 Q 是 $P_1, P_2, \cdots, P_n$ 的逻辑结论。

应用归结原理证明结论为真的过程称为归结反演。

设 F 为已知前提的公式集，Q 为目标公式（结论），用归结反演证明 Q 为真的步骤是

① 否定 Q，得到 $\neg Q$；

② 把 $\neg Q$ 并入到公式集 F 中，得到 $\{F, \neg Q\}$；

③ 把公式集 $\{F, \neg Q\}$ 化为子句集 S；

④ 应用归结原理对子句集 S 中的子句进行归结，并把每次归结得到的归结式都并入 S 中。如此反复进行，若出现了空子句，则停止归结，此时就证明了 Q 为真。

例 4.6　已知

$$F: (\forall x)(\exists y)(A(x,y) \wedge B(y)) \rightarrow (\exists y)(C(y) \wedge D(x,y)))$$

$$G: \neg(\exists x)C(x) \rightarrow (\forall x)(\forall y)(A(x,y) \rightarrow \neg B(y))$$

求证：G 是 F 的逻辑结论。

证明　首先把 F 和 $\neg G$ 化为子句集：

$$\left.\begin{array}{l}(1)\ \neg A(x,y) \vee \neg B(y) \vee C(f(x)) \\ (2)\ \neg A(x,y) \vee \neg B(y) \vee D(x,f(x))\end{array}\right\} F$$

$$\left.\begin{array}{l}(3)\ \neg C(z) \\ (4)\ A(A,B) \\ (5)\ B(B)\end{array}\right\} \neg G$$

归结反演过程用归结树表示如图 4.4 所示。

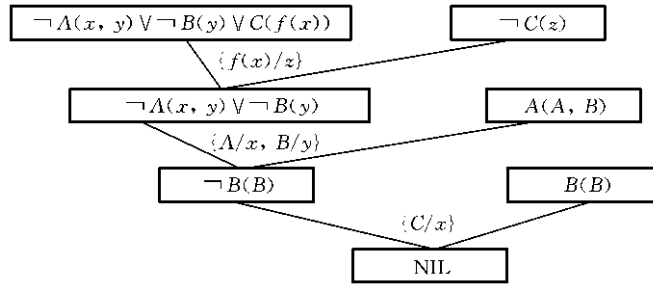

图 4.4 例 4.6 的归结树

例 4.7 在例 2.4 中曾经得到如下公式。

F_1:$(\forall x)(N(x)\rightarrow GZ(x)\wedge I(x))$ 自然数都是大于零的整数。

F_2:$(\forall x)(I(x)\rightarrow E(x)\vee O(x))$ 所有整数不是偶数就是奇数。

F_3:$(\forall x)(E(x)\rightarrow I(s(x)))$ 偶数除以 2 是整数。

求证:所有自然数不是奇数就是其一半为整数的数。

证明 首先把求证的问题用谓词公式表示出来:

$$G:(\forall x)(N(x)\rightarrow(O(x)\vee I(s(x))))$$

再把 F_1,F_2,F_3 及 $\neg G$ 化成子句集:

(1) $\neg N(x)\vee GZ(x)$

(2) $\neg N(u)\vee I(u)$

(3) $\neg I(y)\vee E(y)\vee O(y)$

(4) $\neg E(z)\vee I(s(z))$

(5) $N(t)$

(6) $\neg O(t)$

(7) $\neg I(s(t))$

对上述子句进行归结反演的过程用归结树表示如图 4.5 所示。

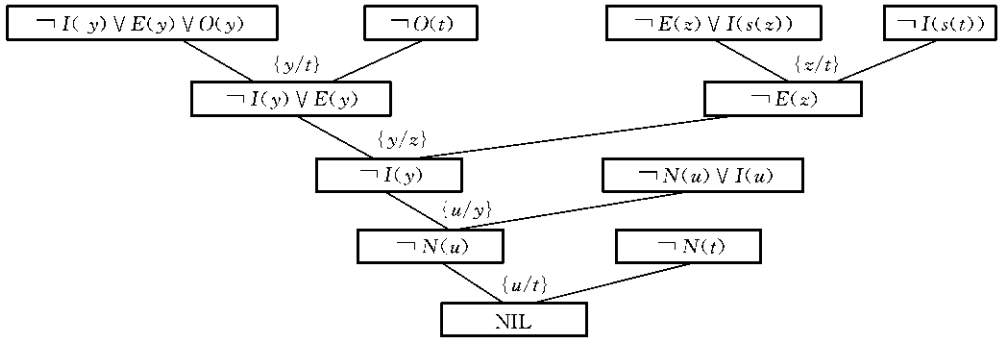

图 4.5 例 4.7 的归结树

4.3　基于归结反演的问题求解

归结反演除了可用于定理证明外，还可用来求取问题的答案，问题求解的方法与定理证明类似。问题求解的步骤如下：

① 把已知前提用谓词公式表示出来，并且化为相应的子句集 S。

② 把待求解的问题也用谓词公式表示出来，然后把它的否定式与谓词 ANSWER 构成一个析取式，ANSWER 是一个为了求解问题而专设的谓词，其变元数量和变元名必须与问题公式的变元完全一致。

③ 把此析取式化为子句集，并且把该子句集并入到子句集 S 中，得到子句集 S'。

④ 对 S' 应用归结原理进行归结。

⑤ 若在归结树的根节点中仅得到归结式 ANSWER，则答案就在 ANSWER 中。

例 4.8　已知

F_1：王(Wang)先生是小李(Li)的老师。

F_2：小李与小张(Zhang)是同班同学。

F_3：如果 x 与 y 是同班同学，则 x 的老师也是 y 的老师。

问：小张的老师是谁？

解　首先定义谓词：

$T(x, y)$　x 是 y 的老师。

$C(x, y)$　x 与 y 是同班同学。

再把已知前提及待求解的问题表示成谓词公式：

F_1：$T(\text{Wang},\text{Li})$

F_2：$C(\text{Li},\text{Zhang})$

F_3：$(\forall x)(\forall y)(\forall z)(C(x,y) \wedge T(z,x) \rightarrow T(z,y))$

G：$(\exists x)T(x,\text{Zhang})$

目标公式 G 的否定式与 ANSWER 的析取式为

$$\neg(\exists x)T(x,\text{Zhang}) \vee \text{ANSWER}(x)$$

把上述公式化为子句集：

(1) $T(\text{Wang},\text{Li})$

(2) $C(\text{Li},\text{Zhang})$

(3) $\neg C(x,y) \vee \neg T(z,x) \vee T(z,y)$

(4) $\neg T(u,\text{Zhang}) \vee \text{ANSWER}(u)$

应用归结原理进行归结，归结过程可用图 4.6 所示的归结树表示。

由 ANSWER(Wang)得出小张的老师是王老师。

例 4.9　设 A,B,C 三人中有人从不说真话，也有人从不说假话，某人向这三人分别提出同一个问题：谁是说谎者？A 答“B 和 C 都是说谎者”；B 答“A 和 C 都是说谎者”；C 答“A 和 B

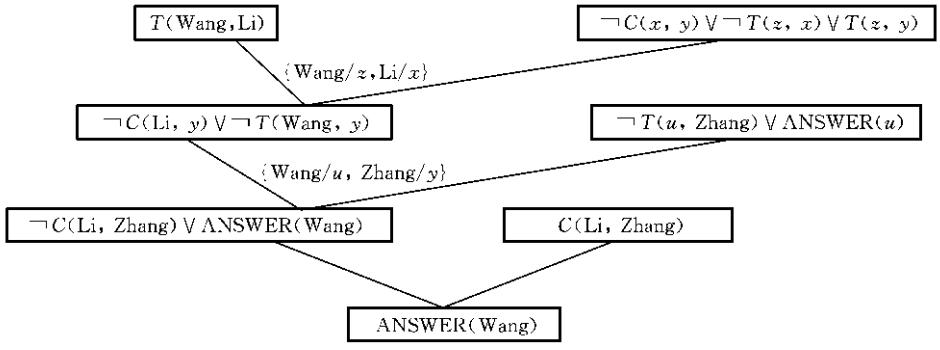

图 4.6　例 4.8 的归结树

中至少有一个是说谎者”。求谁是老实人，谁是说谎者？

解　首先，定义谓词 $T(x)$ **表示** x **说真话。如果** A **说的是真话，则有**

$$T(\mathrm{A}) \rightarrow \neg T(\mathrm{B}) \wedge \neg T(\mathrm{C})$$

如果 A **说的是假话，则有**

$$\neg T(\mathrm{A}) \rightarrow T(\mathrm{B}) \vee T(\mathrm{C})$$

对 B **和** C **说的话作相同的处理，可得**

$$T(\mathrm{B}) \rightarrow \neg T(\mathrm{A}) \wedge \neg T(\mathrm{C})$$
$$\neg T(\mathrm{B}) \rightarrow T(\mathrm{A}) \vee T(\mathrm{C})$$
$$T(\mathrm{C}) \rightarrow \neg T(\mathrm{A}) \vee \neg T(\mathrm{B})$$
$$\neg T(\mathrm{C}) \rightarrow T(\mathrm{A}) \wedge T(\mathrm{B})$$

再把上面这些公式化成子句集 S：

(1) $\neg T(\mathrm{A}) \vee \neg T(\mathrm{B})$

(2) $\neg T(\mathrm{A}) \vee \neg T(\mathrm{C})$

(3) $T(\mathrm{A}) \vee T(\mathrm{B}) \vee T(\mathrm{C})$

(4) $\neg T(\mathrm{B}) \vee \neg T(\mathrm{C})$

(5) $\neg T(\mathrm{C}) \vee \neg T(\mathrm{A}) \vee \neg T(\mathrm{B})$

(6) $T(\mathrm{C}) \vee T(\mathrm{A})$

(7) $T(\mathrm{C}) \vee T(\mathrm{B})$

下面首先求谁是老实人。把目标公式$(\exists x)T(x)$**的否定式**$\neg(\exists x)T(x)$**与** ANSWER(x)**组成的析取式化为子句并入** S **得到** S_1，**即** S_1 **比** S **多如下一个子句**：

(8) $\neg T(x) \vee \mathrm{ANSWER}(x)$

应用归结原理进行归结，归结过程如图 4.7 **所示。**

由 ANSWER(C)**可得** C **是老实人，即** C **从不说假话。**

除此之外，无论如何对 S_1 **进行归结，都推不出** ANSWER(B)**和** ANSWER(A)。

下面来证明 A **和** B **不是老实人。**

设 A **不是老实人，则有**$\neg T(\mathrm{A})$，**把它的否定式** $T(\mathrm{A})$**并入到** S **中，得到子句集** S_2，**即** S_2 **比** S **多一个子句**：$T(\mathrm{A})$

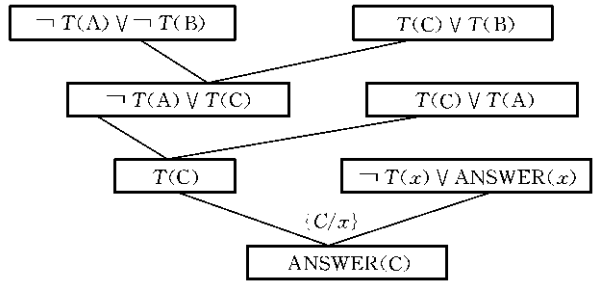

图 4.7　求谁是老实人的归结树

应用归结原理对 S_2 进行归结，归结过程如图 4.8 所示。

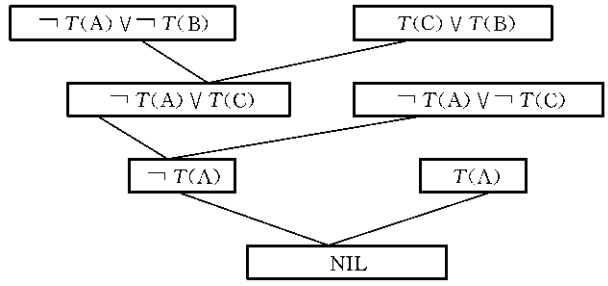

图 4.8　例 4.9 证明 A 不是老实人的归结树

从而证明了 A 不是老实人。同理，可证明 B 也不是老实人。

由上面的例子可以看出，在归结时并不要求把子句集中对归结全部的子句都用到，只要在定理证明时能归结出空子句，在求取问题答案时能归结出 ANSWER 就可以了。另外，在归结过程中，一个子句还可以多次被用来进行归结。

4.4　归结反演的改进策略

对子句集进行归结反演时，由于事先不知道哪两个子句可以进行归结，更不知道通过对哪些子句对的归结可以尽快地得到空子句，因而必须对子句集中的所有子句逐对地进行比较，对任何一个可归结的子句对都进行归结。也就是说，在子句集中采用了类似宽度优先搜索方法来搜索亲本子句，因此归结的效率很低。为此，研究了多种归结反演的改进策略。这些归结反演策略大致可分为两大类：一类是删除策略，另一类是限制策略。前一类通过删除某些无用的子句来缩小归结的范围，后一类通过对参加归结的子句进行种种限制，尽可能地减小归结的盲目性，使其尽快地归结出空子句。

4.4.1　删除策略

归结过程是一个不断寻找可归结子句的过程，子句越多，时空花费就越大。如果在归结时

先把子句集中对归结无用的子句删除掉，就会提高归结的效率。删除策略正是出于这一考虑提出来的，有以下几种删除策略。

1. 纯文字删除

如果某文字 L 在子句集中不存在可与之互补的文字 $\neg L$，则称该文字为纯文字。显然，在归结时纯文字不可能被消去，因而用包含它的子句进行归结时不可能得到空子句，即这样的子句对归结是无意义的，所以可以把它所在的子句从子句集中删去而不会影响子句集的不可满足性。例如，设有子句集：

$$S=\{P\vee Q\vee R,\neg Q\vee R,Q,\neg R\}$$

其中，P 是纯文字，因此可将子句 $P\vee Q\vee R$ 从 S 中删去。

2. 重言式删除

如果一个子句中同时包含互补文字时，则称该子句为重言式。例如，$P(x)\vee\neg P(x)$，$P(x)\vee Q(x)\vee\neg P(x)$都是重言式。重言式是真值为真的子句。不管 $P(x)$为真还是为假，$P(x)\vee\neg P(x)$以及 $P(x)\vee Q(x)\vee\neg P(x)$都均为真。对于一个子句集来说，增加或者删去一个真值为真的子句都不会影响它的不可满足性，因而可从子句集中删去重言式。

3. 包孕删除

设有子句 C_1和 C_2，如果存在一个代换 σ，使得 $C_1\sigma\subseteq C_2$，则称 C_1包孕于 C_2。例如：

$P(x)$	包孕于	$P(y)\vee Q(z)$	$\sigma=\{y/x\}$
$P(x)\vee Q(A)$	包孕于	$P(f(A))\vee Q(A)\vee R(y)$	$\sigma=\{f(A)/x\}$
$P(x)\vee Q(y)$	包孕于	$P(A)\vee Q(u)\vee R(w)$	$\sigma=\{A/x,u/y\}$

把子句集中被包孕的子句删去后，不会影响子句集的不可满足性，因而可从子句集中删去被其他子句包孕的子句。

4.4.2 限制策略

首先讨论类似宽度优先搜索的归结的一般过程，然后讨论几种限制策略。

1. 归结的一般过程

设有初始子句集

$$S=\{C_1,C_2,C_3,C_4\}$$

其中，C_1,C_2,C_3,C_4是 S 中的子句。对此子句集进行归结的一般过程是

① 从子句 C_1开始，逐个与 C_2、C_3、C_4进行比较，看哪两个子句可进行归结。若能找到，就求出归结式。然后，用 C_2与 C_3、C_4进行比较，凡可归结的都进行归结。最后，用 C_3与 C_4比较，若能归结也对它们进行归结。经过这一轮的比较及归结后，就会得到一组归结式，称为第一级归结式。

② 再从 C_1 开始，用 S 中的所有子句分别与第一级归结式中的子句逐个地进行比较、归结，这样又会得到一组归结式，称为第二级归结式。

③ 仍然从 C_1 开始，用 S 中的所有子句及第一级归结式中的所有子句逐个地与第二级归结式中的子句进行比较，得到第三级归结式。

如此继续，只要子句集是不可满足的，上述归结过程直至归结出空子句而终止。

2. 支持集策略

支持集策略对参加归结的子句提出了如下限制：每一次归结时，亲本子句中至少应有一个是由目标公式的否定所得到的子句，或者是它们的后裔。可以证明，支持集策略是完备的，即：若子句集是不可满足的，则由支持集策略一定可以归结出空子句。

例 4.10　设有初始子句集：

$$S=\{\neg I(x)\vee R(x),I(A),\neg R(y)\vee\neg L(y),L(A)\}$$

其中，$\neg I(x)\vee R(x)$ 是目标公式否定得到的子句。

解　用支持集策略进行归结的归结树如图 4.9 所示。

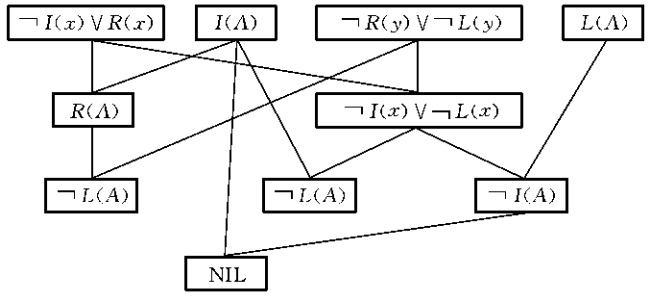

图 4.9　支持集策略归结树

3. 线性输入策略

线性输入策略对参加归结的子句提出了如下限制：参加归结的两个子句中必须至少有一个是初始子句集中的子句。初始子句集是由已知前提与结论的否定组成的子句集。

例 4.11　应用线性输入策略对例 4.10 的初始子句集进行归结。

解　用线性输入策略进行归结的归结树如图 4.10 所示。

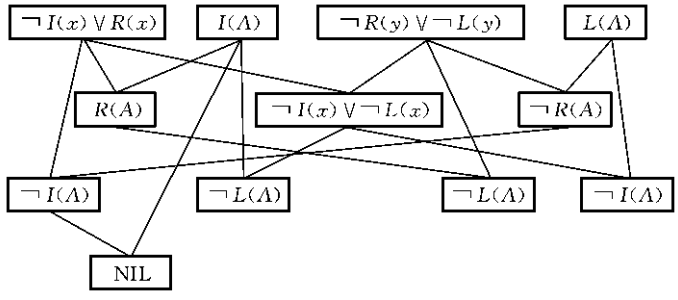

图 4.10　线性输入策略归结树

线性输入策略可限制生成归结式的数量，具有简单、高效的优点，但它是不完备的。也就是说，即使子句集是不可满足的，用线性输入策略进行归结也不一定能归结出空子句。例如，对于子句集

$$S=\{P(x)\vee Q(x),\quad \neg P(y)\vee Q(y),\quad P(u)\vee \neg Q(u),\quad \neg P(t)\vee \neg Q(t)\}$$

可以证明它是不可满足的，但用线性输入策略归结却得不出空子句。

4. 单文字子句策略

如果一个子句只包含一个文字，则称它为单文字子句。单文字子句策略要求参加归结的两个子句中必须有一个是单文字子句。

例 4.12 对例 4.10 给出的初始子句集按单文字子句策略进行归结。

解 用单文字子句策略进行归结的归结树如图 4.11 所示。

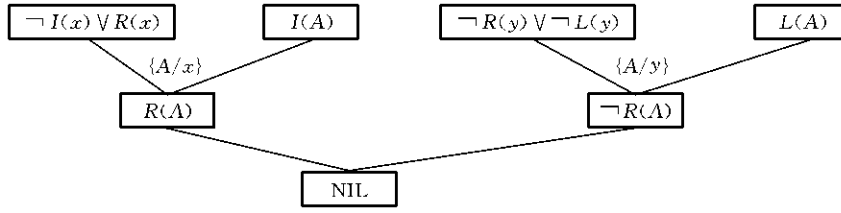

图 4.11 单文字子句策略的归结树

用单文字子句策略归结时，归结式将比亲本子句含有较少的文字，这有利于朝着空子句的方向前进，因此它有较高的归结效率。但是，这种归结策略是不完备的。另外，当初始子句集中不包含单文字子句时，归结就无法进行。

5. 祖先过滤型策略

祖先过滤型策略与线性输入策略比较相似，但放宽了限制。当对两个子句 C_1 和 C_2 进行归结时，要求它们满足下述两个条件中的任意一个条件：

① C_1 与 C_2 中至少有一个是初始子句集中的子句。

② 如果两个子句都不是初始子句集中的子句，则一个子句应是另一个子句的祖先。所谓一个子句（例如，C_1）是另一个子句（例如，C_2）的祖先是指 C_2 是由 C_1 与别的子句归结后得到的归结式。

例 4.13 有子句集 $S=\{\neg P(x)\vee Q(x),\ \neg P(y)\vee\neg Q(y),P(u)\vee Q(u),P(t)\vee\neg Q(t)\}$，用祖先过滤型策略进行归结时，归结树如图 4.12 所示。

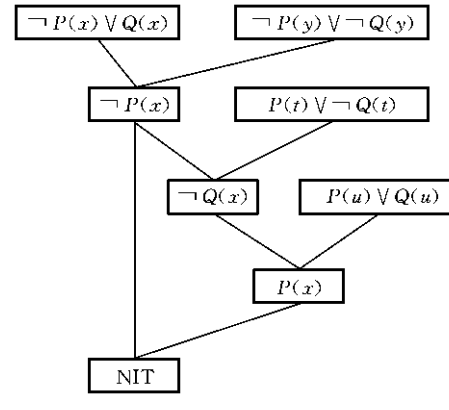

图 4.12 祖先过滤型策略归结树

在此例中，归结出空子句的两个子句是：$C_1=\neg P(x)$，$C_2=P(x)$，其中，C_1 是 C_2 的祖先。

可以证明，祖先过滤型策略是完备的。

以上讨论的几种基本的归结策略，在具体应用时可把几种策略组合在一起使用。

归结演绎推理是在自动定理证明领域影响较大的一种推理方法，它比较简单且又便于在计算机上实现。但由于它要求把逻辑公式转化成子句集，就可能丢失蕴含式含有的逻辑控制信息。例如，下列逻辑公式：

$$(\neg A \wedge \neg B) \rightarrow C$$
$$(\neg A \wedge \neg C) \rightarrow B$$
$$(\neg B \wedge \neg C) \rightarrow A$$
$$\neg A \rightarrow (B \vee C)$$
$$\neg B \rightarrow (A \vee C)$$
$$\neg C \rightarrow (A \vee B)$$

它们分别具有不同的逻辑控制信息，但若把它们分别化为子句，则得到的子句却是相同的，即都是：$A \vee B \vee C$。

针对归结演绎推理存在的上述问题，人们提出了多种非子句定理证明方法，例如，基于与/或形的演绎推理就是其中的一种。

4.5　与/或形演绎推理

与/或形演绎推理不需要把有关知识化为子句集，而是把领域知识和已知事实分别用蕴含式和与/或形表示，然后运用蕴含式进行演绎推理，从而证明某个目标公式。与/或形演绎推理分为正向演绎与逆向演绎两种基本推理方式。

4.5.1　与/或形正向演绎推理

与/或形正向演绎推理从已知事实出发，正向地使用蕴含式(F 规则)进行演绎推理，直至得到目标公式为止。

1. 事实表达式的与/或形变换及其与/或树表示

与/或形正向演绎推理要求已知事实用不含蕴含符号“→”的与/或形表示。把表示已知事实的谓词公式化为与/或形的步骤与化为子句集类似，只是不必把公式化为子句的合取形式，即不能消去公式中的合取连词。把一个事实表达式变换为与/或形的步骤如下：

① 利用 $P \rightarrow Q \Leftrightarrow \neg P \vee Q$ 消去公式中的“→”。

② 利用德·摩根律及量词转换律把“¬”移到紧靠谓词的位置上。

③ 重新命名变元名，使不同量词约束的变元有不同的名字。

④ 引入 Skolem 函数消去存在量词。

⑤ 消去全称量词，且使各主要合取式中的变元不同名。

例如，对如下事实表达式：

$$(\exists x)(\forall y)\{Q(y,x) \wedge \neg[(R(y) \vee P(y)) \wedge S(x, y)]\}$$

按上述步骤进行转化后得到与/或形表达式：

$$Q(z,A)\wedge\{[\neg R(y)\wedge\neg P(y)]\vee S(A,y)\}$$

事实表达式的与/或形可用一棵与/或树表示出来，上例的与/或树如图 4.13 所示。

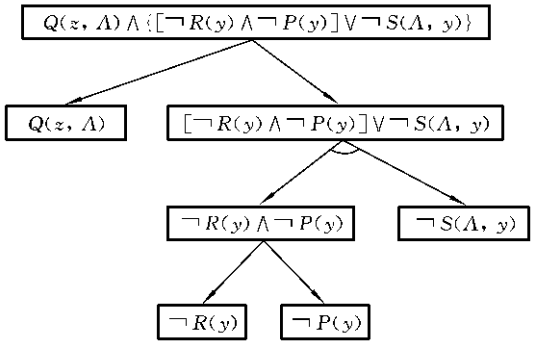

图 4.13 事实表达式的与/或树表示

在图 4.13 中，每个节点表示相应事实表达式的一个子表达式，叶节点为谓词公式中的文字。对于用析取符号“∨”连接而成的表达式，用一个连接符（即图中的半圆弧）把它们连接起来。如果把与/或树中用连接符连接的节点视为具有“与”关系，把不用连接符连接的节点视为具有“或”关系，那么由叶节点所组成的公式：

$$Q(z,A)$$
$$\neg R(y)\vee S(A,y)$$
$$\neg P(y)\vee\neg S(A,y)$$

恰好是原表达式化成的子句集。

2. F 规则的表示形式

在与/或形正向演绎推理中，要求 F 规则具有如下形式：

$$L\rightarrow W$$

其中，L 为单文字，W 为与/或形。之所以限制 F 规则的左部为单文字，是因为在进行演绎推理时，要用 F 规则作用于表示事实的与/或树，而该与/或树的叶节点都是单文字，这样就可用 F 规则的左部与叶节点进行简单匹配（合一）。

如果领域知识的表示形式不是所要求的形式，则需通过变换将它变成规定的形式，变换步骤如下。

① 暂时消去蕴含符号“→”。例如，对公式

$$(\forall x)\{[(\exists y)(\forall z)P(x,y,z)]\rightarrow(\forall u)Q(x,u)\}$$

通过运用等价关系 $P\rightarrow Q\Leftrightarrow\neg P\vee Q$ 可变为

$$(\forall x)\{\neg[(\exists y)(\forall z)P(x,y,z)]\vee(\forall u)Q(x,u)\}$$

② 把“¬”移到紧靠谓词的位置上。通过运用德·摩根律及量词转换律可把“¬”移到括弧中，经移动“¬”，上式变为

$$(\forall x)\{(\forall y)(\exists z)[\neg P(x,y,z)]\vee(\forall u)Q(x,u)\}$$

③ 引入 Skolem 函数消去存在量词。消去存在量词后，上式变为

$$(\forall x)\{(\forall y)[\neg P(x,y,f(x,y))]\vee(\forall u)Q(x,u)\}$$

④ 消去全称量词。消去全称量词后，上式变为

$$\neg P(x,y,f(x,y))\vee Q(x,u)$$

⑤ 恢复为蕴含式。利用等价关系 $\neg P\vee Q\Leftrightarrow P\rightarrow Q$ 将上式变为

$$P(x,y,f(x,y))\rightarrow Q(x,u)$$

3. 目标公式的表示形式

在与/或形正向演绎推理中，要求目标公式用子句表示，否则就需要化成子句形式。

4. 推理过程

应用 F 规则进行推理的目的在于证明某个目标公式。如果从已知事实的与/或树出发，通过运用 F 规则最终推出了欲证明的目标公式，则推理就可成功结束。其推理过程为

① 首先用与/或树把已知事实表示出来；

② 用 F 规则的左部和与/或树的叶节点进行匹配，将匹配成功的 F 规则加入到与/或树中；

③ 重复进行步骤②，直到产生一个含有以目标节点作为终止节点的解图为止。

例 4.14　设已知事实为

$$A\vee B$$

F 规则为

$$r_1:A\rightarrow C\wedge D$$

$$r_2:B\rightarrow E\wedge G$$

待证的目标公式为

$$C\vee G$$

其证明过程如图 4.14 所示。其中，为把这里所说的与/或树与一般意义下的与/或树区别开来，将它按倒置的形式存放，双箭头表示匹配。

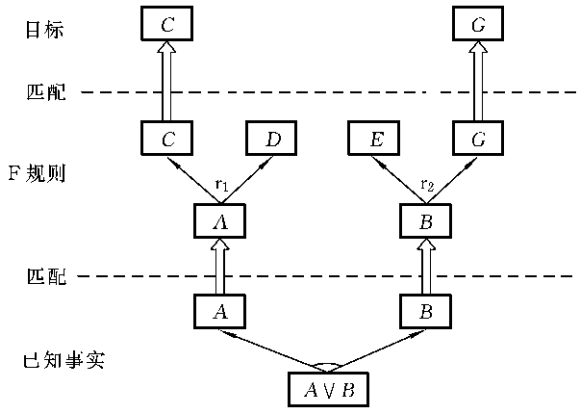

图 4.14　例 4.14 推理过程

对于用谓词公式表示已知事实及 F 规则的情形，推理中需要用最一般合一进行变元的代换，下面用例子说明。

例 4.15 设已知事实为

$$\neg P(A) \vee \{Q(A) \wedge R(A)\}$$

F 规则为

$$r_1: \neg P(x) \rightarrow \neg S(x)$$

$$r_2: Q(y) \rightarrow N(y)$$

待证的目标公式为

$$\neg S(z) \vee N(z)$$

推理过程如图 4.15 所示。

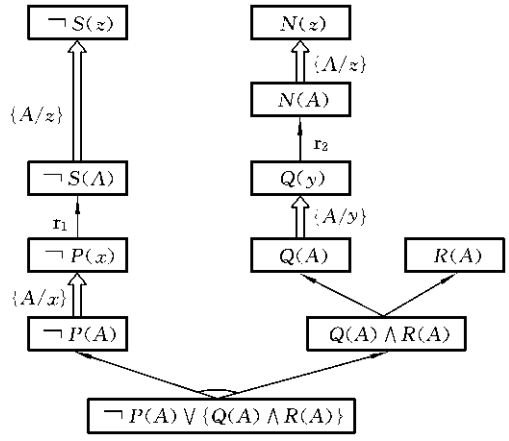

图 4.15 例 4.15 的推理过程

4.5.2 与/或形逆向演绎推理

与/或形逆向演绎推理是从待证明的问题(目标)出发,逆向使用蕴含式(B 规则)进行演绎推理,直至得到包含已知事实的终止条件为止。

1. 目标公式的与/或形变换及其与/或树表示

在与/或形逆向演绎推理过程中,要求目标公式用与/或形表示,其变换过程与正向演绎推理中对已知事实的变换相似,只是要用存在量词约束的变元的 Skolem 函数替换由全称量词约束的相应变元,并且消去全称量词,然后消去存在量词,这是与正向演绎推理中对已知事实进行变换的不同之处。例如,对如下目标公式:

$$(\exists y)(\forall x)\{P(x) \rightarrow [Q(x,y) \vee \neg(R(x) \wedge S(y))]\}$$

经变换后得到

$$\neg P(f(z)) \vee \{Q(f(y),y) \wedge [\neg R(f(y)) \vee \neg S(y)]\}$$

变换时应注意使各个主要的析取式具有不同的变元名。

目标公式的与/或形可用与/或树表示出来,但其表示方式与正向演绎推理中对已知事实的与/或树表示略有不同,它的连接符用来把具有合取关系的子表达式连接起来,而在正向演

绎推理中是把已知事实中具有析取关系的子表达式连接起来。例 4.15 的与/或树如图 4.16 所示。

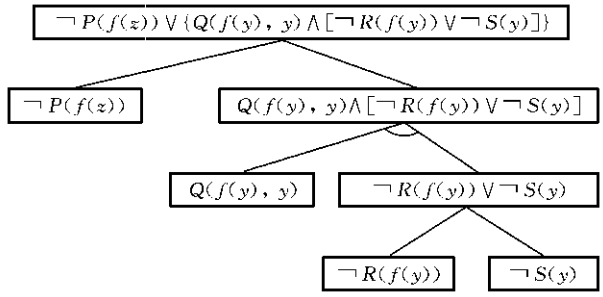

图 4.16　目标公式的与/或树表示

在图 4.16 中，若把叶节点用它们之间的合取及析取关系连接起来，就可得到原目标公式的三个子目标：

$$\neg P(f(z))$$
$$Q(f(y),y) \wedge \neg R(f(y))$$
$$Q(f(y),y) \wedge \neg S(y)$$

可见子目标是文字的合取式。

2. B 规则的表示形式

B 规则的表示形式为

$$W \rightarrow L$$

其中，W 为任一与/或形公式，L 为文字。之所以限制规则的右部为文字，是因为推理时要用它与目标与/或树中的叶节点进行匹配(合一)，而目标与/或树中的叶节点是文字。

如果已知的 B 规则不是所要求的形式，可用与转化 F 规则类似的方法把它化成规定的形式，特别是对于像

$$W \rightarrow (L_1 \wedge L_2)$$

这样的蕴含式可化为两个 B 规则：

$$W \rightarrow L_1, \quad W \rightarrow L_2$$

3. 已知事实的表示形式

在逆向演绎推理中，要求已知事实是文字的合取式，即形如

$$F_1 \wedge F_2 \wedge \cdots \wedge F_n$$

在问题求解中，由于每个 $F_i(i=1,2,\cdots,n)$ 都可单独起作用，因此可把上面公式表示为事实的集合：

$$\{F_1, F_2, \cdots, F_n\}$$

4. 推理过程

应用 B 规则进行逆向演绎推理的目的是求解问题，当从目标公式的与/或树出发，通过运

用 B 规则最终得到某个终止在事实节点上的一致解图时，推理成功结束。其推理过程如下：

① 先用与/或树把目标公式表示出来。

② 用 B 规则的右部和与/或树的叶节点进行匹配，将匹配成功的 B 规则加入到与/或树中。

③ 重复进行步骤②，直到产生某个终止在事实节点上的一致解图为止。这里所说的“一致解图”是指在推理过程中所用到的代换应该是一致的。

例 4.16 设有如下事实及规则。

(1) 事实

F_1：DOG(Fido)　　Fido 是一只狗

F_2：$\neg$BARKS(Fido)　　Fido 不叫

F_3：WAGS-TAIL(Fido)　　Fido 摇尾巴

F_4：MEOWS(Myrtle)　　Myrtle 咪咪叫

(2) 规则

r_1：(WAGS-TAIL(x_1) $\wedge$ DOG(x_1)) $\rightarrow$ FRIENDLY(x_1)　　狗摇尾巴表示友好

r_2：(FRIENLY(x_2) $\wedge$ $\neg$BARKS(x_2)) $\rightarrow$ $\neg$AFRAID(y_2, x_2)　　友好且不叫的狗不可怕

r_3：DOG (x_3) $\rightarrow$ ANIMAL(x_3)　　狗是动物

r_4：CAT(x_4) $\rightarrow$ ANIMAL(x_4)　　猫是动物

r_5：MEOWS(x_5) $\rightarrow$ CAT(x_5)　　咪咪叫者是猫

需要求解的问题是：是否有这样的一只猫和一条狗，而且这只猫不怕这条狗？

解 该问题的目标公式为

$$(\exists x)(\exists y)[\mathrm{CAT}(x) \wedge \mathrm{DOG}(y) \wedge \neg \mathrm{AFRAID}(x, y)]$$

求解该问题的过程如图 4.17 所示。

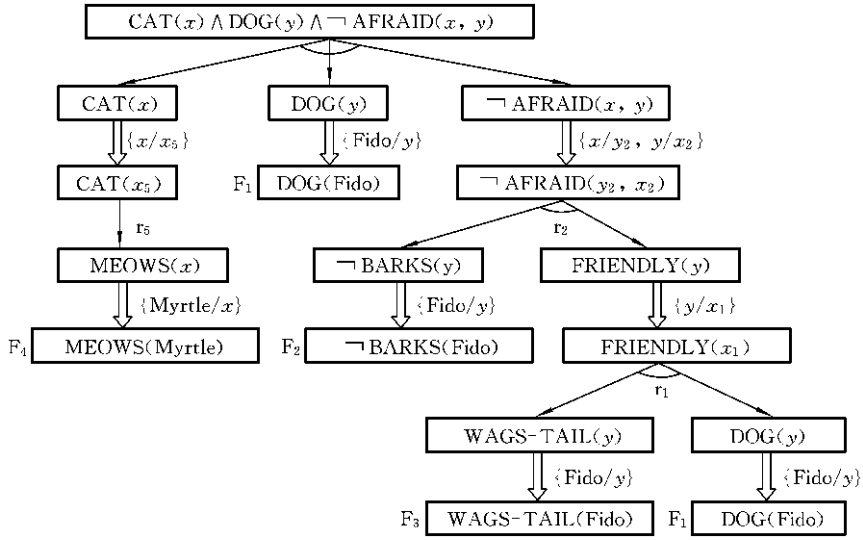

图 4.17　例 4.16 的推理过程

该推理过程得到的解图是一致解图。图中有 8 条匹配弧，每条弧上都有一个代换，而且这些代换是一致的。终止在事实节点上的代换为{Myrtle/x}和{Fido/y}，把它们应用到目标公式，就得到该问题的解：

$$\mathrm{CAT(Myrtle)} \wedge \mathrm{DOT(Fido)} \wedge \neg \mathrm{AFRAID(Myrtle,Fido)}$$

它表示：一只名叫 Myrtle 的猫和一条名叫 Fido 的狗，而且这只猫不怕那条狗。

4.5.3　代换的一致性与剪枝策略

1. 代换的一致性

无论对与/或形的正向演绎推理还是逆向演绎推理，都要求推理过程中所用的代换集合具有一致性，下面给出代换集合一致性的定义。

定义 4.11　设代换集合

$$\theta=\{\theta_1,\theta_2,\cdots,\theta_n\}$$

中第 i 个代换 $\theta_i(i=1,2,\cdots,n)$ 为

$$\theta_i=\{t_{i1}/x_{i1},t_{i2}/x_{i2},\cdots,t_{im(i)}/x_{im(i)}\}$$

其中，t_{ij} 为项，x_{ij} 为变元($j=1,2,\cdots,m(i)$)，则代换集 θ 是一致的充要条件是如下两个元组

$$T=\{t_{11},t_{12},\cdots,t_{1m(1)},t_{21},\cdots,t_{2m(2)},\cdots,t_{nm(n)}\}$$

$$X=\{x_{11},x_{12},\cdots,x_{1m(1)}x_{21},\cdots,x_{2m(2)},\cdots,x_{nm(m)}\}$$

可合一。

例如：

设 $\theta_1=\{x/y\}$，$\theta_2=\{y/z\}$，则 $\theta=\{\theta_1,\theta_2\}$ 是一致的。

设 $\theta_1=\{f(g(x_1))/x_3,f(x_2)/x_4\}$，$\theta_2=\{x_4/x_3,g(x_1)/x_2\}$，则 $\theta=\{\theta_1,\theta_2\}$ 是一致的。

设 $\theta_1=\{A/x\}$，$\theta_2=\{B/x\}$，则 $\theta=\{\theta_1,\theta_2\}$ 是不一致的。

设 $\theta_1=\{g(y)/x\}$，$\theta_2=\{f(x)/y\}$，则 $\theta=\{\theta_1,\theta_2\}$ 是不一致的。

2. 剪枝策略

在与/或形演绎推理中，要不断地用 F 规则的左部(正向演绎推理)或 B 规则的右部(逆向演绎推理)和与/或树的叶节点进行匹配(合一)，在每一步上可匹配的规则都可能不止一条，为了得到一致解图，应选哪一条规则？这里以逆向演绎推理为例讨论一种方法，即剪枝策略。

剪枝策略的基本思想是：每当选用一条规则时，就进行一次一致性检查，如果当前的部分解图是一致的，则继续向下扩展，否则就放弃该规则而选用其他候选规则。由于应用该方法在使用每一条规则时都进行了一致性检查，所以最终得到的解图必然是一致的。另外，由于这种方法可尽量地发现并剪除不一致的分枝，从而避免了大的返工。

例如，设当前的与/或树如图 4.18(a)所示，可用的 B 规则有

$$\mathrm{r}_1: S(z) \rightarrow P(B)$$

$$\mathrm{r}_2: R(y) \rightarrow P(y)$$

若首先选用 r_1，则得到代换集合为

$$\{A/x, B/x\}$$

显然它是不一致的，因而必须放弃这一选择。接着选用 r_2，得到的代换集合为

$$\{A/x, x/y\}$$

这是一致的，所以可把 r_2 用于推理，并把它加入到目标与/或树中，如图 4.18(b)所示。

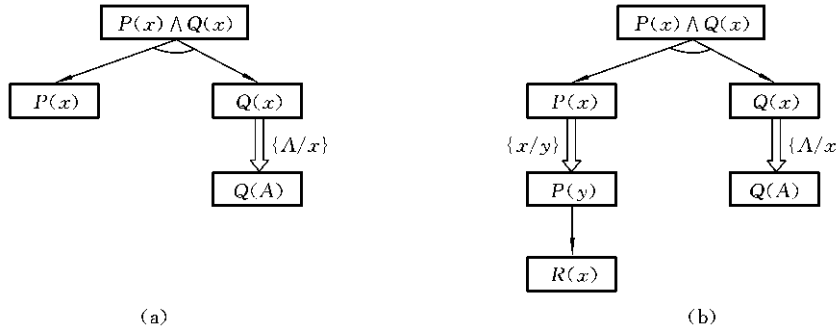

图 4.18 剪枝策略

习 题 四

4.1 名词解释：

演绎推理　　归纳推理

确定性推理　　不确定性推理

单调推理　　非单调推理

基于知识的推理　　常识性推理

4.2 何谓正向推理？请根据正向推理的基本过程画出正向推理的流程示意图。

4.3 何谓逆向推理？请根据逆向推理的基本过程画出逆向推理的流程示意图。

4.4 何谓模式匹配？什么是确定性匹配？什么是不确定性匹配？

4.5 何谓冲突消解？在产生式系统中，什么情况下会发生冲突？常用的冲突消解策略有哪些？请分别简要说明。

4.6 公式集的合一是什么含义？何谓最一般合一？

4.7 何谓归结式？为什么说归结式是其亲本子句的逻辑结论？

4.8 应用归结原理来证明子句集的不可满足性的依据是什么？如何进行证明？

4.9 哪些归结反演的限制策略是完备的？

4.10 下述公式集 F 是否可合一，若可合一，则求出 F 的最一般合一。

(1) $F=\{P(A,B), P(x,y)\}$

(2) $F=\{P(f(x),B), P(y,z)\}$

(3) $F=\{P(f(x),y), P(y,f(B))\}$

(4) $F=\{P(f(y),y,x), P(x,f(A),f(B))\}$

4.11 把下列谓词公式分别化为相应的子句集：

(1) $(\forall x)(\forall y)(P(x,y) \wedge Q(x,y))$

(2) $(\forall x)(\forall y)(P(x,y) \rightarrow Q(x,y))$

(3) $(\forall x)(\exists y)(P(x,y)\vee(Q(x,y)\rightarrow R(x,y)))$

(4) $(\forall x)(\exists y)(\exists z)(P(x,y)\rightarrow Q(x,y)\vee R(x,z))$

(5) $(\exists x)(\exists y)(\forall z)(\exists u)(\exists v)(\exists w)(P(x,y,z,u,v,w)\wedge(Q(x,y,z,u,v,w)\vee\neg R(x,z,w)))$

4.12　**判断下列子句集中哪些是不可满足的：**

(1) $S=\{\neg P\vee Q,\neg Q,P,\neg P\}$

(2) $S=\{P\vee Q,\neg P\vee Q,P\vee\neg Q,\neg P\vee\neg Q\}$

(3) $S=\{P(y)\vee Q(y),\neg P(f(x))\vee R(A)\}$

(4) $S=\{\neg P(x)\vee Q(x),\neg P(y)\vee R(y),P(A),S(A),\neg S(z)\vee R(z)\}$

(5) $S=\{\neg P(x)\vee\neg Q(y)\vee\neg L(x,y),P(A),\neg R(z)\vee L(A,z),R(B),Q(B)\}$

(6) $S=\{\neg P(x)\vee Q(f(x),A),\neg P(h(y))\vee Q(f(h(y)),A)\vee\neg P(z)\}$

(7) $S=\{P(x)\vee Q(x)\vee R(x),\neg P(y)\vee R(y),\neg Q(A),\neg R(B)\}$

(8) $S=\{P(x)\vee Q(x),\neg Q(y)\vee R(y),\neg P(z)\vee Q(z),\neg R(u)\}$

4.13　**对下列各题分别证明 G 是否为前提公式集 F 的逻辑结论：**

(1) $F_1:(\exists x)(\exists y)P(x,y)$

$G:(\forall y)(\exists x)P(x,y)$

(2) $F_1:(\forall x)(P(x)\wedge(Q(A)\vee Q(B)))$

$G:(\exists x)(P(x)\wedge Q(x))$

(3) $F_1:(\exists x)(\exists y)(P(f(x))\wedge Q(f(B)))$

$G:P(f(A))\wedge P(y)\wedge Q(y)$

(4) $F_1:(\forall x)(P(x)\rightarrow(\forall y)(Q(y)\rightarrow\neg L(x,y)))$

$F_2:(\exists x)(P(x)\wedge(\forall y)(R(y)\rightarrow L(x,y)))$

$G:(\forall x)(R(x)\rightarrow\neg Q(x))$

(5) $F_1:(\forall x)(P(x)\rightarrow(Q(x)\wedge R(x)))$

$F_2:(\exists x)(P(x)\wedge S(x))$

$G:(\exists x)(S(x)\wedge R(x))$

4.14　**某公司招聘工作人员，A，B，C 三人应试。经面试后，公司表示如下意见：**

① **三人中至少录用一人；**

② **如果录用 A 而不录用 B，则一定录用 C；**

③ **如果录用 B，则一定录用 C。**

求证：公司一定录用 C。

4.15　**已知前提：每个储蓄钱的人都获得利息。求证结论：如果没有利息，那么就没有人去储蓄钱。**

4.16　**已知前提：**

① **某些病人喜欢所有的医生；**

② **没有一个病人喜欢任何一个骗子。**

求证结论：任何一个医生都不是骗子。

4.17　**应用归结反演方法证明理发师悖论：若每个理发师都为不能给自己理发的人理发，且每个理发师都不为能给自己理发的人理发，那么不存在任何理发师。**

4.18　**设已知**

① **如果 x 是 y 的父亲，y 是 z 的父亲，则 x 是 z 的祖父；**

② **每个人都有一个父亲。**

试用归结演绎推理证明：对于某人 u，一定存在一个人 v，且 v 是 u 的祖父。

4.19 张某被盗，公安局派出五个侦察员去调查。研究案情时，侦察员 A 说“赵与钱中至少有一人作案”；侦察员 B 说“钱与孙中至少有一人作案”；侦察员 C 说“孙与李中至少有一人作案”；侦察员 D 说“赵与孙中至少有一人与此案无关”；侦察员 E 说“钱与李中至少有一人与此案无关”。如果这五个侦察员的话都是可信的，试用归结演绎推理求出谁是盗窃犯。

4.20 已知下述事实：

① 小李只喜欢较容易的课程；

② 工程类课程是较难的；

③ PR 系的所有课程都是较容易的；

④ PR150 是 PR 系的一门课程。

应用归结演绎推理回答问题：小李喜欢什么课程？

4.21 设已知

① 能阅读者都能识字；

② 海豚不识字；

③ 有些海豚是聪明的。

分别用线性输入策略、祖先过滤型策略证明：有些聪明者不能阅读。

4.22 已知下述事实：

① 小杨、小刘和小林是高山俱乐部成员；

② 高山俱乐部的每个成员都是滑雪者或登山者，或者既滑雪又登山；

③ 没有一个登山者喜欢下雨；

④ 所有滑雪者都喜欢下雪；

⑤ 凡是小杨喜欢的，小刘就不喜欢；

⑥ 凡是小杨不喜欢的，小刘就喜欢；

⑦ 小杨喜欢下雨和下雪。

试证明：俱乐部是否有是登山者而不是滑雪者的成员？如果有，他是谁？

4.23 设已知事实为

$$((P \vee Q) \wedge R) \vee (S \wedge (T \vee U))$$

F 规则为

$$S \rightarrow (X \wedge Y) \vee Z$$

用正向演绎推理推出所有可能的目标子句。

4.24 已知下述事实和规则：

事实 Fido 又叫(BARKS)又咬(BITS)，否则 Fido 就不是狗(DOG)。

规则 所有的小猎犬(TERRIES)都是狗。

任何一种叫声都是噪声(NOISY)。

应用与/或形正向演绎推理证明：存在某些不是小猎犬而是噪声的东西。

第 5 章

专 家 系 统

专家系统(Expert System,ES)是一种模拟专家解决领域问题的计算机程序系统。专家系统内部含有大量的某个领域的专家的知识与经验,能够运用专家的知识和解决问题的方法进行推理和判断,模拟专家的决策过程来解决该领域的复杂问题。

5.1 专家系统概述

专家系统是人工智能应用研究领域中最活跃和最广泛的应用领域。自从 1965 年第一个专家系统 DENDRAL 在美国斯坦福大学问世以来,各种专家系统已遍布各个专业领域,取得了很大的成功。

5.1.1 专家系统的类型与特点

1. 专家系统的类型

1983 年,Hayes-Roth 等根据专家系统处理的问题的类型,把专家系统分为以下 10 种类型。

(1) 解释型专家系统

解释型专家系统的任务是:通过对已知信息和数据的分析,解释这些信息和数据的实际含义。典型的解释型专家系统有信号理解和化学结构解释等专家系统。例如,由质谱仪数据解释化合物分子结构的 DENDRAL 系统、语言理解系统 HEARSAY,由声呐信号识别舰船的 HASP/SIAP 系统等都是对于给定数据,找出与之相一致的、符合客观规律的解释。

(2) 诊断型专家系统

诊断型专家系统的任务是根据输入信息找出处理对象中存在的故障。主要有医疗、机械和电子等领域中的各种诊断专家系统。例如,血液凝结疾病诊断系统 CLOT、计算机硬件故障诊断系统 DART、化学处理工厂故障诊断系统 FALCON 等,都是通过处理对象内部各部件的功能及其相互之间的关系,来检测和查找可能的故障。

(3) 调试型专家系统

调试型专家系统的任务是给出已确认故障的排除方案。主要有电子设备和机械设备的计算机辅助调试专家系统。例如,VAX/VMS 计算机系统的辅助调试系统 TIMM/TUNER、石油钻探机械故障的诊断与排除系统 Drilling Advisor 等,都是根据处理对象和故障的特点,从

多种纠错方案中选择最佳方案。

(4) 维修型专家系统

维修型专家系统的任务是制定并实施纠正某类故障的规划。例如,计算机网络的维护专家系统、电话电缆维护专家系统 ACE、诊断和排除内燃机故障的 DELTA 系统等,都是根据对象系统的故障和纠错方法的特点,制定出合理的故障维修规划。

(5) 教育型专家系统

教育型专家系统的任务是:根据学生的学习特点,把知识以适当的方法组织起来,对学生进行教学和辅导,及时诊断学习过程中出现的错误和处理学习过程中遇到的问题。例如,GUIDON 和 STEAMER 等专家系统、可进行逻辑学与集合论教学的 EXCHECK 教学系统及一些计算机辅助教学(CAI)系统和聋哑人语言训练系统等。

(6) 预测型专家系统

预测型专家系统的任务是根据处理对象过去和现在的情况分析,推测未来的演变和发展。典型的应用有天气预报、财政预测、经济发展预测、人口预测、交通预测等。例如,各种产品市场预测专家系统、气象预报系统、军事冲突预测系统 I&W 等,都是进行与时间顺序有关的推理,处理随时间变化的数据和按时间顺序发生的事件。

(7) 规划型专家系统

规划型专家系统的任务是寻找出某个能够达到给定目标的动作序列或步骤。规划型专家系统可用于机器人动作规划、交通运输调度、工程项目论证与规划、通信与军事指挥及生产作业规划等。比较典型的规划型专家系统有 ROPES 机器人运动规划专家系统、制定最佳行车路线的 CARG 系统、安排宇航员在空间站中活动的 KNEECAP 系统、分子遗传学实验设计专家系统 MOLGEN 等,都是在一定的约束条件下,以较小的代价达到给定目标。

(8) 设计型专家系统

设计型专家系统的任务是根据给定的要求形成所需要的方案或图形描述。典型的应用有电路设计和机械设计。例如,VAX 计算机的总体结构和配置系统 XCON、超大规模集成电路辅助设计系统 KBVLSI、自动程序设计系统 PSI 等,都是在给定要求的限制下,提供最佳或较佳的设计方案。另外,花布图案设计和花布印染专家系统、各种机械零件设计及加工工艺设计专家系统等,都属于设计型专家系统。

(9) 监测型专家系统

监测型专家系统主要用于完成实时监测任务,对系统、对对象或过程的行为进行不间断监测,把监测到的行为与其应当具有的行为进行比较,以发现异常情况,发出警报。典型的应用有空中交通管制监测和核电站安全监测等。例如,航空母舰周围空中交通管制系统 AIRPLAN、核反应堆事故诊断与处理系统 REACTOR、高危病人监护系统 VM 等,都是随时收集有关处理对象的各种数据,并把这些数据与预期的数据比较,一旦发现异常就立即发出报警信号。这类系统通常是诊断型、预测型和调试型的合成。

(10) 控制型专家系统

控制型专家系统的任务是自适应地管理一个受控对象或客体的全部行为,使之满足预期要求,通常用于实时控制型任务。典型的应用有战场辅助作战指挥系统、汽车变速箱控制系

统、生产过程控制和空中交通管制等。例如，维持钻机最佳钻探流特征的 MUD 系统、MVS 操作系统的监督控制系统 YES/MVS 等。这类系统通常是监测型和维修型的合成。

实际上，上述 10 种任务类型之间往往互相关联，有些专家系统通常能够完成几种类型的任务。例如，MYCIN 就是一个诊断型和调试型的专家系统。

1985 年，Clancy 指出，无论专家系统完成什么类型的任务，就领域问题的基本操作来说，专家系统求解的问题可分为分类问题和构造问题两类。求解分类问题的专家系统称为分析型专家系统，广泛用于解释、诊断和调试等类型的任务；求解构造问题的专家系统称为设计型专家系统，广泛用于规划、设计等类型的任务。

2. 专家系统的一般特点

各种类型的专家系统都有各自的特点，在总体上，专家系统还具有以下一些共同的特点。

(1) 知识的汇集

一个专家系统汇集了某个领域多位专家的经验和知识及他们协作解决重大问题的能力。因此，专家系统应表现出更渊博的知识、更丰富的经验和更强的工作能力，而且能够高效、准确、迅速和不知疲倦地工作。

(2) 启发性推理

专家系统运用专家的经验和知识进行启发式推理，对问题作出判断和决策。

(3) 推理和解释的透明性

用户无需了解推理过程，就能从专家系统获得问题的结论，而且推理过程对用户是透明的。专家系统的解释器可以回答用户关于“系统是怎样得出这一结论的”和“为什么会提出这样的问题”之类的询问，专家系统是如何实现这些问题的解释对用户也是透明的。

(4) 知识获取与知识更新

专家系统能够不断地获取知识，增加新的知识，修改原有知识。机器学习就是专家系统知识获取与知识更新的重要方法。

5.1.2　专家系统的结构与开发方法

1. 专家系统的结构

专家系统的结构是指专家系统各组成部分的构造和组织形式。

专家系统一般的系统结构框图如图 5.1 所示，其组成部分及其主要功能说明如下。

(1) 知识库

知识库(Knowledge Base)以某种存储结构存储领域专家的知识，例如，求解领域问题所需的操作与规则等。为了建立知识库，首先要解决知识表示问题，即要确定知识表示的外部模式和内部模式。

(2) 全局数据库

全局数据库(Global Database)亦称为“黑板”，它用于存储求解问题的初始数据和推理过

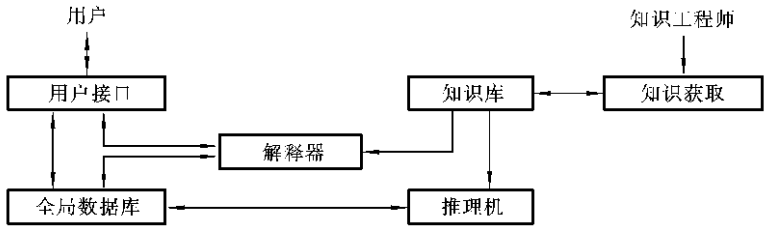

图 5.1 专家系统结构框图

程中得到的中间数据，以及最终的推理结论。

(3) 推理机

推理机(Reasoning Machine)根据全局数据库的当前内容，从知识库中选择匹配成功的可用规则，并通过执行可用规则来修改数据库中的内容，直至推理出问题的结论。推理机中包含如何从知识库中选择可用规则的策略和当有多个可用规则时如何消解规则冲突的策略。

(4) 解释器

解释器(Expositor)用于向用户解释专家系统的行为，包括解释“系统是怎样得出这一结论的”、“系统为什么要提出这样的问题来询问用户”等用户需要解释的问题。

(5) 用户接口

用户接口(Interface)是系统与用户进行对话的界面。用户通过接口输入必要的数据、提出问题和输出推理获得的结果及系统向用户作出的解释；系统通过人机接口要求用户回答系统的询问，回答用户的问题并作出解释。

(6) 知识获取

知识获取模块把知识工程师提供的知识转换为知识内部表示模式存入知识库中，在知识存储的过程中，对知识进行一致性、完整性检测。

由于每个专家系统所需要完成的任务不同，因此其系统结构也不尽相同。知识库和推理机是专家系统中最基本的模块。知识表示的方法不同，知识库的结构也就不同。推理机是对知识库中的知识进行操作的，推理机程序与知识表示的方法及知识库结构是紧密相关的，不同的知识表示需要有不同的推理机程序。

2. 专家系统的开发方法

专家系统的开发是一项综合技术，一个成功的专家系统的开发需要知识工程师和领域专家的密切配合和坚持不懈的努力。

(1) 建造专家系统的步骤

根据软件工程的生命周期方法，一个实用专家系统的开发过程可类同一般软件系统的开发过程，分为认识、概念化、形式化、实现和测试等阶段。

① 认识阶段。知识工程师与领域专家合作，对领域问题进行需求分析，包括认识系统需要处理的问题范围、类型和各种重要特征、预期的效益等，并确定系统开发所需的资源、人员、经费和进度等。

② 概念化阶段。把问题求解所需要的专门知识概念化，确定概念之间的关系，并对任务

进行划分，确定求解问题的控制流程和约束条件。

③ 形式化阶段。把已整理的概念、概念之间的关系和领域专门知识用适合于计算机表示和处理的形式化方法进行描述和表示，并选择合适的系统结构，确定数据结构、推理规则和有关控制策略，建立问题求解模型。

④ 实现阶段。选择适当的程序设计语言或专家系统工具建立可执行的原型系统。

⑤ 测试阶段。通过运行大量的实例，检测原型系统的正确性及系统性能。通过测试原型系统，对反馈信息进行分析，进而进行必要的修改，包括重新认识问题，建立新的概念或修改概念之间的联系、完善知识表示与组织形式、丰富知识库的内容、改进推理方法等。

专家系统的这一开发过程，类似一般软件系统开发过程的瀑布模型，各阶段目标明确，逐级深化。开发过程的瀑布模型如图 5.2 所示。

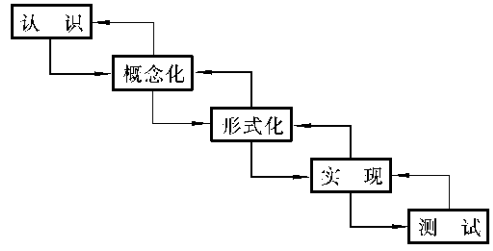

图 5.2　专家系统开发过程的瀑布模型

(2) 原型系统与快速原型法

由于领域专家的知识是长期积累的经验和专门知识，因此，知识工程师不可能在短时间内获得所需要的全部专家知识，并把它们按知识表示方式和知识库的结构要求存入知识库中。也就是说，决定专家系统性能的专门知识是逐步增加和不断完善的，这就需要采用增量式开发方法，即通过对基本功能的逐步扩大来完善系统。专家系统具有将需要经常修改和完善的知识库同相对稳定的推理机相分离的结构特点，从而适应了这种增量式开发方法。增量式开发可保证对基本功能的有效验证，有利于在整个开发过程中得到一系列功能日趋完善的原型系统。

根据系统的复杂程度和实用性，原型系统一般可分成以下 5 种。

① 演示原型。大多数专家系统都开始于一个演示原型，它是仅能解决少量问题的一个演示型系统。演示原型主要有两个作用，一是确信人工智能和专家系统技术能有效地用于所要解决的问题；二是测定问题的定义和范围及领域知识的表示是否正确。一个典型的基于规则的大型专家系统，其演示原型一般仅有 50～100 条规则，能充分地执行 2～3 个测试实例。

② 研究原型。研究原型是能运行多个测试实例的原型系统，这些测试实例能显示领域问题的重要特点。大型专家系统的研究原型一般具有 200～500 条规则。

③ 领域原型。通过改进研究原型而获得领域原型，领域原型系统运行可靠，具有比较流畅和友善的用户接口，能基本满足用户的需要。大型专家系统的领域原型一般具有 500～1000 条规则，能很好地执行许多测试实例。

④ 产品原型。产品原型是已经过广泛的领域问题测试的原型系统，并往往用一种效率更高的语言或专家系统工具来实现，以增加推理的速度和减少存储空间。大型专家系统的产品

原型一般具有 500～1500 条规则，求解领域问题准确快速，工作可靠。

⑤ 商品化系统。商品化系统是已投入商品市场实际销售和运行的系统，并能适应用户市场的需要。

利用专家系统技术和专家系统的开发工具尽快地建立专家系统的演示原型，然后进行修改、充实和完善，就是专家系统开发的快速原型法。虽然演示原形比较简单，只能解决少量的领域问题，也不具备许多辅助功能，但是，通过演示原型的运行和测试可以实际验证系统方案的可行性和有效性，检验应用问题的定义范围，从而在系统设计的最初阶段就能避免较大的原则性错误；而且可以提高领域专家的兴趣和信心，增强同领域专家的合作。

5.2 LISP 语言

LISP 语言诞生 30 多年来，最重要的应用领域是人工智能，尤其是专家系统领域。20 世纪 80 年代，LISP 语言及其应用迅速发展，大约有十多种 LISP 语言。1984 年，推出的 Common LISP包括了各种重要的 LISP 语言的优点，成为 LISP 语言的事实上的标准语言。在 PC 机上使用的 GCLISP 是 Common LISP 的子集。为学习本书提供的产生式系统实例的需要，本节讲述 Common LISP 中有关的基本内容。

5.2.1 LISP 语言的特点与表达式

大多数传统的程序设计语言是过程型的，要求程序员在程序中把问题求解过程的繁琐细节都要描述出来，而 LISP 语言是一种基于表结构的函数型语言。

1. LISP 语言的特点

(1) 函数性

函数型语言的基本特点是用函数定义和函数调用构成程序。程序员用函数定义和函数调用组成的表达式来描述求解问题的算法，表达式的值就是问题的解。用 FORTRAN、PASCAL 和 C 等传统程序设计语言编写的程序是按一定顺序执行的命令序列，执行结果就是问题的解。用这些语言编程时，程序员要规定求解的顺序，即要描述控制流。用 LISP 语言编程只需要确定函数之间的调用，把函数执行的细节交给 LISP 系统来解决。因此，LISP 语言是更加面向用户的语言。

传统的程序设计语言是适应冯·诺依曼型计算机系统结构而发展起来的，LISP 在诺依曼型计算机上运行的效率要低一些。计算机系统结构的发展，使得函数型语言有着广阔的前途。为了适应当前微型机发展水平和程序员使用传统语言编程的习惯，LISP 语言增加了许多非函数型的语言成分，例如，prog、go 等函数，所以，LISP 已不是纯函数型语言，它既具有函数语言的功能，又具有传统语言的功能。

(2) 递归性

递归函数是指在函数的定义中调用了这个函数本身。所有的可计算函数已被证明都可以

用递归函数的形式来定义。

由于 LISP 的主要数据结构是表，而且表是用递归方法定义的，即表中的一个元素也可以定义为一个表，因此，程序员用 LISP 提供的自定义函数来定义用户自己的函数时，可以用递归函数的形式来定义自己的函数。自定义的递归函数能够很方便地对递归定义的表进行操作。递归定义的方法使程序简明、优美，程序员应充分利用递归程序设计方法。

(3) 数据与程序的一致性

LISP 的一段程序是用户的一个自定义函数，这个函数可被其他函数调用，或者说，一段程序可被其他程序调用。函数执行后的输出数据称为这个函数的返回值。一个函数被其他函数调用，就是调用了这个函数的返回值。在 LISP 中，函数与这个函数的返回值是一致的。这一特点使得 LISP 的编程就是定义一个宏函数，也使得 LISP 语言的扩充比较容易。可以根据应用领域的需要，使用 LISP 提供的基本函数扩充若干面向专门应用领域的宏函数。

(4) 自动进行存储分配

用 LISP 语言编程时，程序员完全可以不考虑存储分配问题。程序中定义的函数、数据和表等都能在程序运行时，由 LISP 自动提供。对不再需要的数据，LISP 自动释放其占用的存储区。

(5) 语法简单

LISP 的语法极其简单，对变量和数据不需要事先定义和说明类型。LISP 语言的基本语法就是函数定义和函数调用。因此，LISP 语言的程序便于修改、调试和纠错，可以边实验边设计，通过不断修改和增加用户自定义函数来构成复杂的系统。

LISP 语言不仅在专家系统和 CAD 领域有广泛的应用，在符号代数、定理证明、机器人规划等领域也有广泛的应用。影响 LISP 语言使用的主要原因有：一是 LISP 是非可视化语言；二是 LISP 在通用计算机上的运行效率较低；三是 LISP 的数值计算能力较差；四是人们对函数型语言的编程风格不习惯。

2. LISP 的符号表达式

LISP 的符号类似其他程序设计语言中的变量，一个符号名是以字母开头不含规定的特殊字符的字符串，可以把计算或处理后的结果赋给一个符号。数和符号都称为原子，它们是 LISP 中不能再分割的对象。若干个数或符号用括号括起来就构成一个表，表中的元素用空格分开。没有元素的表称为空表，空表可用( )表示，也可表示为 nil，空表也是原子。表和原子是 LISP 的主要对象。

表是可递归定义的，即可用若干个表来定义另一个表，或者说，表中的元素可以是其他的表。表中元素的个数称为表的长度。

原子和表组成 LISP 的符号表达式。LISP 的符号表达式采用前缀表示形式，即表中第一个元素是函数符号名，其余的元素是这个函数要求的运算或处理的元素。例如，符号表达式

(setq　y　(＊　2　3　4))

是一个表，表中第一个元素是符号名为 setq 的赋值函数名，setq 对 y 的赋值是符号表达式(＊　2　3　4)的值。(＊　2　3　4)也是一个表，表中第一个元素是乘法函数名＊，其后的

元素是参加乘法运算的数，因此，(* 2 3 4)的返回值是 24，(setq y (* 2 3 4))的返回值是 $y=24$。

LISP 中的串是用双引号引起来的字符串，串中可以包含空格。LISP 中的文件名用串表示。例如，符号表达式

(load "my.l sp")

也是一个表，表中第一个元素是装入文件的函数 load，装入的文件名用一个字符串表示。

5.2.2 LISP 语言的基本函数

LISP 提供近 400 个基本函数，我们只能根据本书给出的推理机程序和解释器程序的需要，分类说明部分基本函数的功能和使用方法。

1. 数值运算函数

数值运算函数对指定的数进行运算，函数返回值是数值运算的结果。对函数功能进行说明时，用一个箭头"→"表示其后是该函数的返回值。

(1) 算术运算函数

算术运算函数有：加函数"+"、减函数"-"、乘函数"*"、除函数"/"、加 1 函数"1+"、减 1 函数"1-"等。+、-、*、/等函数可对多个数或已经赋值的符号进行数值运算，例如

(setq x 10)
(/ x 2 2)

的返回值为 2.5。

1+和 1-函数对指定的一个变量加 1 或减 1。

(2) 超越函数

主要的超越函数有

① (exp n)→e^n

② (expt y n)→y^n

③ (log x y)→$\log_y x$

若缺省 y，则返回值为 $\log_{10} x$。

④ (sqrt x)→$\sqrt{x}$

⑤ (xys n)→$|n|$

⑥ (signum n)→ -1,0,1

若 $n<0$，则返回值为 -1；若 $n=0$，则返回值为 0；若 $n>0$，则返回值为 1。

另外，还有一组三角函数。

(3) 数的逻辑运算函数

数的逻辑运算函数可把指定的多个十进制整数先转换成二进制数，然后把这多个二进制数的对应位进行逻辑运算，把运算结果再转换成十进制整数作为函数返回值。这样的逻辑运算函数有：逻辑或运算函数 logior、逻辑异或运算函数 logxor、逻辑与运算函数 logand、逻辑非

运算函数 lognot 等。例如

(logior　25　30)→31

(logxor　25　15)→22

(logand　30　42)→10

另外，还有可将二进制数左移或右移若干位后再转换成十进制数的移位逻辑运算函数等。

2. 求值与赋值函数

(1) 禁止求值函数

LISP 用符号表达式进行求值。符号表达式是一个表，LISP 把表中第一个元素解释为一个函数，把其他元素解释这个函数的运算数，符号表达式的返回值是这个函数的运算结果。但是，表还有另一种情况，表中的元素都是数，即不希望把表中第一个元素解释为一个函数。为此，LISP 提供一个禁止求值函数 quote 对指定的表说明表中元素都是数。例如

(quote　(a　b　c))

的返回值是表(a　b　c)。quote 函数的缩写为“'”。

LISP 执行符号表达式(setq　x　'(a　b　c))的结果是对 x 赋值为一个表(a　b　c)。但是，LISP 对符号表达式(setq　x　(a　b　c))会发出出错信息，因为 LISP 会把 a 解释为一个函数的符号名，由于 a 不是一个已定义的函数，因此，LISP 认为这是一个非法的符号表达式。

(2) 赋值函数

赋值函数 setq 用于对变元赋值，对一个变元赋的值可以是一个数、一个符号表达式、一个表或者另一个变元。setq 函数可以对多个变元依序赋值，例如

(setq　x　'(1　2)　y　x)

setq 先对变元 x 赋值为(1　2)，再对变元 y 赋值为 x，使变元 y 的值也为(1　2)。

(3) 求值函数

LISP 有多个求值函数。函数 values 返回其后各变元的值，函数 values-list 返回其后一个表的各元素。例如

(values　(*　2　3) (+　4　5))→6　9

(values　'(a　b)　'(c　d))→(a　b)(c　d)

(values-list　'(a　b　c))→a　b　c

3. 表处理函数

LISP 的基本数据结构是表，LISP 语言应最擅长于表的处理，为此，LISP 提供了若干表处理函数，包括提取表中部分内容的函数、构造表的函数、表结构变换的函数和其他表处理函数。

(1) 取表部分内容的函数

取表部分内容的函数取出一个表中由函数指定的部分内容。

① car 函数取表的第一个元素，例如

(car　'(a　b　c))→a

② cdr 函数取表中除掉第一个元素的余下表，例如

(cdr '(a b c))→(b c)

③ 函数 car 和 cdr 可对一个表连续作用，例如

(car(cdr(cdr(cdr '(a b c d e f)))))

可表示为

(cadddr '(a b c d e f))

cadddr 也是 LISP 函数，该例的返回值为 d。LISP 提供 car 和 cdr 连续作用函数，但连续作用的次数不超过 4 次，即这一组函数中，任何一个函数名中的 a 和 d 的个数总共不超过 4 个。

④ first 函数等同于 car 函数。

⑤ second 函数等同于 cadr 函数。

⑥ third 函数等同于 caddr 函数。

⑦ rest 函数等同于 cdr 函数。

⑧ nth 函数返回表中指定的那一个元素，例如

(nth 3 '(a b c d))→d

注意：a 为第 0 个元素。

(2) 构造表的函数

① cons 函数把指定的两个元素构造成一个表，如果第 2 个元素是一个表，则把第 1 个元素加到第 2 个元素的表头。例如

(cons a '(b c))→(a b c)

(cons '(a b) '(c d))→((a b) c d)

② list 函数把指定的多个元素按顺序构造成一个表。例如

(list a b c d)→(a b c d)

(list '(a b) '(c d))→((a b) (c d))

③ append 函数把指定的多个表拼接成一个表。例如

(append '(a b) '(c d) '(e f))→(a b c d e f)

(3) 其他表函数

① list-length 函数返回指定的一个表的元素个数。例如

(list-length '(a '(b c)))→2

② member 函数表达式为

(member item list)

如果 item 是表 list 中的一个元素，则 member 返回 list 中从元素 item 开始的余下表；否则，返回空表( )，也即是返回 nil。例如

(member b '(a b c d))→(b c d)

(member '(a b) '(a b c d))→( )

(member '(b c) '(a '(b c) d))→((b c) d)

4. 逻辑函数

逻辑函数的返回值只能是真(t)或假(nil)。逻辑函数用于判断某种情况是否发生或用于

判断函数各变元的逻辑值。

(1) 数据类型判断函数

① atom 函数判断其后的对象是否是原子，若是一个原子，则返回 t；否则，返回 nil。例如，

(atom　( ))→ t

空表( )是一个原子。

② listp 函数判断其后的对象是否是一个表，若是一个表(包括空表)，则返回 t；否则，返回 nil。

③ null 函数判断其后的对象是否是一个空表，若是一个空表，则返回 t；否则，返回 nil。

④ stringp 函数判断其后的对象是否是一个串，若是一个串，则返回 t；否则，返回 nil。

⑤ characterp 函数判断其后的对象是否是字符，若是字符，则返回 t；否则，返回 nil。

⑥ symbolp 函数判断其后的对象是否是符号，若是符号，则返回 t；否则，返回 nil。例如

(symbolp　(a　b))→t

(symbolp　′(a　b))→nil

其中，(a　b)是一个符号表达式，a 表示一个已定义的函数。′(a　b)则是一个表。

⑦ numberp 函数判断其后的对象是否是一个数，若是一个数，则返回 t；否则，返回 nil。

另外，还有正数判断函数 minusp、负数判断函数 plusp、零判断函数 zerop、整数判断函数 integerp、偶数判断函数 evenp、奇数判断函数 oddp 等。

(2) 数的比较函数

数的比较函数用于比较两个数的大小，有大于比较函数＞、小于比较函数＜、大于等于比较函数＞＝、小于等于比较函数＜＝、等于比较函数＝、不等于比较函数/＝。若指定的两个数满足函数的比较关系，则返回 t；否则，返回 nil。

(3) 等值函数

数的等于比较函数“＝”用于比较两个数是否相等，若要判别两个表或符号是否相等，则要用等值函数 equal。例如

(equal　′(a　b　c)　′(a　b　c))→t

(equal　( )　nil)　→t

(equal　′(a　b　c)　′(a　(b　c)))→nil

(4) 逻辑运算函数

逻辑运算函数有逻辑与运算函数 and、逻辑或运算函数 or、逻辑非运算函数 not。

① and 函数当且仅当其各元素的值均为非 nil，则返回值为 t；否则，返回 nil。

② or 函数当且仅当其各元素中只要有一个元素的值为非 nil，则返回值为 t；否则，返回 nil。

③ not 函数当且仅当其元素的值为 nil，则返回值为 t；否则，返回 nil。

5. 条件函数

LISP 程序的条件分支控制是用条件函数来实现的。

(1) if 函数

if 函数的表达式为

(if test then [else])

若 test 的值为非 nil，则对 then 求值，且 if 函数的返回值就是 then 的值；否则，对 else 求值并作为 if 函数的返回值，如果没有 else，则 if 函数返回 nil。if 函数表达式中，else 部分是可缺省的。表达式中用方括号[ ]括起来的部分表示是可缺省的。

(2) when 函数

when 函数的表达式为

(when test {form}*)

其中，test 为测试条件，form 为符号表达式，{form}* 表示可有多个符号表达式。若 test 的值为非 nil，则顺序对多个 form 求值，且以最后一个 form 的值作为 when 函数的返回值；否则，when 函数返回 nil。

(3) unless 函数

unless 函数的表达式为

(unless test {form}*)

若 test 的值为 nil，则顺序对多个 form 求值，且以最后一个 form 的值作为 unless 函数的返回值；否则，unless 函数返回 nil。

(4) cond 函数

cond 函数的表达式为

$$\begin{aligned}(\text{cond}\quad & (\exp_{11}\quad \exp_{12}\quad \cdots)\\ & (\exp_{21}\quad \exp_{22}\quad \cdots)\\ & \quad\vdots\\ & (\exp_{n1}\quad \exp_{n2}\quad \cdots))\end{aligned}$$

cond 函数顺序对 n 个表进行处理，一个表$(\exp_{i1}\quad \exp_{i2}\quad \cdots)$称为一个 cond 分句。每个分句中的第一个元素 $\exp_{i1}$ 是这个分句的测试条件，其后各元素是符号表达式。

LISP 顺序计算各分句的测试条件 $\exp_{i1}$，若 $\exp_{i1}$ 的值为非 nil，则计算这个分句各表达式的值，且以最后一个表达式的值作为 cond 函数的返回值，不再计算其后的分句；若所有分句的测试条件均为 nil，则 cond 函数返回 nil。一种特殊情况是：若某个 cond 分句中只有测试条件 $\exp_{i1}$，且 $\exp_{i1}$ 的值为非 nil，则 cond 函数返回这个 $\exp_{i1}$ 的非 nil 值。

6. 自定义函数与无名函数

(1) 自定义函数 defun

用户可以用自定义的函数 defun 来自行定义所需要的新函数。自定义函数的表达式为

(defun name lambda-list {form}*)

其中，name 是新函数的函数名，lambda-list 是新函数 name 的变量表，{form}* 是由多个表达式组成的新函数 name 的定义体。

例 5.1 用自定义函数 defun 定义比较函数 compare，它比较两个数 x 和 y，若 $x=y$，则返回串“numbers are same”；若 $x>y$，则返回串“first is bigger”；若 $x<y$，则返回串“first is smaller”。

可如下定义函数 compare：

```
(defun  compare  (x  y)
    (cond  ((=  x  y) (print  "numbers are same"))
           ((> x  y) (print  "fist is bigger"))
           (t  (print  "first is smaller"))))
```

一个新的函数被定义后就可如同 LISP 的基本函数一样被直接调用。例如

```
(compare  (3  5))→"first is smaller"
```

(2) 无名函数 lambda

使用 defun 函数来定义一个函数时，要给出函数名、变量表和定义体，LISP 要将其保存起来，以便可用函数名来调用这个函数。若某个新函数定义后只使用一次，那么，用 defun 来定义这个新函数就是一种浪费。可以使用无名函数 lambda 来定义并同时执行这个新函数，定义时无需给出新函数的函数名，执行后不被保存，也不能被调用。

lambda 函数的表达式为

```
(lambda  lambda-list  {form}*  list)
```

其中，实参表 list 中元素的个数应与变量表 lambda-list 中的形参个数一致。lambda 函数的返回值为用实参表中的实参代替定义体{form}* 中相应形参后的函数计算值。例如

```
(lambda  (x)  (*  x  x  x)  3)→27
(lambda  (x  y)  (+  x  y)  '(1  3))→4
```

7. 高阶函数

高阶函数是以函数为变量而定义的一种函数。若 $g(f(x,y))$ 中 $f(x,y)$ 是函数，那么，g 称为高阶函数。例如，可以定义广义求和函数 $g=\sum_{i=m}^{n}f(i)=f(m)+f(m+1)+\cdots+f(n)$，$f$ 是一个任意函数。在求和函数 g 被定义后，可对任意指定函数 f 调用求和函数 g，从而得出对 f 的求和。或者说，指定一个函数名实参，就可得到这个函数的高阶函数 g 的值。

例 5.2　定义平方和函数 sum-sqares 与立方和函数 sum-cubes，它们分别计算从整数 m 到 n 的平方和与立方和。

可如下定义：

```
(defun  sqare  (m)
        (*  m  m))
(defun  cube  (m)
        (*  m  m  m))
(defun  sum-sqares  (m  n)
  (if  (<  m  n)
       (+  (sqare  m) (sum-sqares  (1+  m)  n))))
(defun  sum-cubes  (m  n)
  (if  (<  m  n)
       (+  (cube  m) (sum-cubes  (1+  m)  n))))
```

例 5.2 中求和函数的定义基本相同，只是 sum-squares 调用 square 函数来计算 m^2，sum-cubes 调用 cube 函数来计算 m^3。因此，可以采用高阶函数的形式来定义一个广义求和函数 sum-item，调用 sum-item 时，若指定其函数名形参为 square 就得出平方和；若指定函数名形参为 cube 就得出立方和。另外，注意到上述两个求和函数的定义中，都使用了递归调用的方式，即一个函数调用这个函数本身。

高阶函数可使用函数 apply 来定义。apply 函数的表达式为

(apply function arg)

apply 的第一个变量 function 是一个函数名或者是一个广义的函数名符号，apply 的第二个变量 arg 是函数 function 的变量。如果 function 有多个变量{arg}*，那么最后一个变量应是一个表。

例 5.3 使用 apply 函数来定义广义求和函数 sum-item。

解 可如下定义：

```
(defun sum-item (func m n)
  (if (< m n)
      (+ (apply func (list m))
         (sum-item (func (1+ m) n))))))
```

定义了广义求和函数 sum-item 后，调用 sum-item 时，用已定义的函数的函数名作为实参替换 sum-item 中广义函数名符号 func，就可得出求和值。例如

(sum-item (square 2 3))→$2^2+3^2=13$

(sum-item (cube 2 3))→$2^3+3^3=35$

可以直接使用 apply 函数表达式来计算一个高阶函数的返回值，在 apply 函数表达式中，function 就是一个已定义的函数名，且在 function 前要加上“'”，表示 apply 不直接对 function 求值，而 function 只是 apply 的一个变量。例如

(apply 'cons '(a (b c)))→(a b c)

(apply '+ '(1 2))→3

5.2.3 迭代与递归

循环程序是一种重要的程序结构，循环程序可采用迭代和递归两种方法来实现。LISP 提供若干迭代函数来定义迭代方式的函数，也可以按递归的方法用自定义函数 defun 来定义递归方式的函数。

1. 迭代函数

LISP 提供的迭代函数可分为结构迭代函数、非结构迭代函数和映射函数三类。

(1) 结构迭代函数

结构迭代是指迭代过程中没有 goto 之类的无条件转移，程序的结构性好，是结构化程序设计方法所提倡的。LISP 的结构迭代函数以函数 do 为代表。

do 函数的表达式为

```
(do  ((var1  init1  upd1)
      (var2  init2  upd2)
             ⋮
                          )
     (condition  action1  action2  …)
     {form}*  )
```

其中，var1，var2，…是变量名，init1，init2，…是各变量赋给的初值，若某个变量的初值没有给出，则该变量的初值为 nil。upd1，upd2，…是每次循环后对相应变量的值进行更新的表达式，若某个更新表达式没有给出，则相应变量的值由循环体来更新。(condition action1 action2…)是循环是否结束的判断子句，若 condition 的值为非 nil，则循环结束，并对 action1，action2，…依序求值，把最后一个表达式 action 的值作为 do 函数的返回值；否则，执行循环体 {form}* 。

例 5.4　用结构迭代方式定义计算阶乘 $n!$ 的函数 factorial (n)。

解　可如下定义：

```
(defun  factorial  (n)
 (do  ((i  1  (1+i))
       (result  1))
      ((>  i  n)  result)
      (setq  result  (*  i  result))))
```

调用时，用实参代替函数 factorial 的形参 n。例如

```
(factorial  4)→24
```

(2) 非结构迭代函数

LISP 提供函数 prog、函数 go 和函数 return 用于定义非结构迭代函数。

① prog 函数的表达式为

```
(prog  ((var1  init1) (var2  init2)…)
   exp1  exp2  …)
```

其中，var1，var2，…是变量名，init1，init2，…是各变量赋给的初值，若某个变量的初值没有给出，则该变量的初值为 nil。在表达式序列 exp1，exp2，…中可以有多个表达式含有 go 函数表达式，至少有一个表达式含有 return 函数表达式。若没有 go 和 return，则对各表达式顺序求值后退出 prog，且函数 prog 的返回值为 nil。

② go 函数的表达式为

```
(go  symbol)
```

执行 go 函数时，LISP 在 prog 中找名为 symbol 的表达式(原子)，若未找到，则 prog 函数返回出错信息；若找到，即 prog 函数转到名为 symbol 的原子之后的表达式继续执行。

③ return 函数的表达式为

```
(return  form)
```

执行 return 函数时，对 form 求值后退出 prog，且函数 prog 的返回值为 form 的值。

在 prog 函数中，可以出现多个 go 和 return 的函数表达式，prog 函数的返回值将依赖于具体执行情况。

(3) 映射函数

映射函数是一种隐式的迭代控制方式，LISP 提供一组映射函数的函数名以 map 开头，它们都以某种方式处理表的每一个元素。

① mapcar 函数的表达式为

(mapcar function {list}*)

其中，function 是一个函数名，{list}* 是一个或多个表，表的个数是函数 function 要求的变量个数。mapcar 把由函数名 function 指定的函数功能映射到表的各个元素上，映射的结果为一个表，并作为 mapcar 函数的返回值。在函数名 function 前需要加上“'”，使 LISP 对 function 不直接求值，只是把函数名 function 作为一个元素送给映射函数 mapcar。例如

(mapcar 'car '(a b) '(c d) '(e f))→(a c e)

(mapcar 'factorial '(1 2 3 4))→(1 2 6 24)

② maplist 函数的表达式为

(maplist function {list}*)

maplist 把由函数名 function 指定的函数功能映射到表{list}*，产生的映射结果表作为 maplist 返回表的第 1 个元素，对{list}* 先用 cdr 作用后再映射作为返回表的第 2 个元素，对{list}* 先用 cddr 作用后再映射作为返回表的第 3 个元素，……直至{list}* 为空表为止。例如

(maplist 'append '(a b c) '(1 2 3))

→((a b c 1 2 3) (b c 2 3) (c 3))

2. 递归

递归是一种重要的程序设计方法和技巧，采用递归方法设计的递归程序给人以简洁优美的感觉。同样，在 LISP 中可按递归方法定义递归函数。LISP 没有提供特殊的基本函数来支持递归函数的定义和调用。LISP 对递归函数调用和非递归函数的调用过程的处理是相同的，LISP 用栈的数据结构来保存每层调用的实参和中间结果，并维护着调用的层次关系。由内层到外层返回的过程按栈中各层内容返回。正是依靠 LISP 对程序员透明的这种调用和返回机制，使程序员无需关心调用和返回的过程细节。

递归函数的表现能力要强于迭代函数，但这并不意味着在任何情况下都使用递归而不使用迭代。一般而言，递归应用于对上下文无关或块结构语言的句法分析、语义检查及代码生成，用于求解递归定义的数学问题，或者更一般地讲，它应用于自定义嵌套很深的问题的求解。重复地对某些事物做同样事情则应选用迭代，比如对表中的每个元素都做某种相同的操作。另外，迭代比递归的效率一般来讲要高一些，效率包括时间和空间两个方面。如果更看重效率，那么应尽可能设计一个迭代的解法。

5.3　知识库与推理机

领域专家解决领域问题的能力主要体现在两个方面：一是专家拥有大量的知识，二是专家具有选择知识来解决问题的能力。这种选择知识和应用知识求解问题的过程就称为基于知识的推理。知识库与推理机是专家系统中必不可少的组成部分，是基于知识的推理的基础和核心。

5.3.1　产生式规则与规则库的存储结构

1. 产生式规则的存储结构

一般产生式规则的前件或后件可能是有限个事实或结论的合取式的析取，例如，规则 R 为

$$(F_1 \wedge F_2 \wedge F_3) \vee (F_4 \wedge F_5) \rightarrow H_1 \vee H_2$$

可等价变换为下述 4 条规则：

$$R_{11}: \quad F_1 \wedge F_2 \wedge F_3 \rightarrow H_1$$

$$R_{12}: \quad F_4 \wedge F_5 \rightarrow H_1$$

$$R_{21}: \quad F_1 \wedge F_2 \wedge F_3 \rightarrow H_2$$

$$R_{22}: \quad F_4 \wedge F_5 \rightarrow H_2$$

在规则库中，允许有前件不同但后件相同的规则，例如：R_{11} 和 R_{12} 的后件均为 H_1，R_{21} 和 R_{22} 的后件均为 H_2。但是，产生式规则的一致性要求规则库中的规则之间满足：凡是后件相同的规则，它们的前件没有包含关系，即一条规则的前件不是另一条规则的前件的子集。例如，R_{11} 与 R_{12} 的前件之间没有包含关系，R_{21} 与 R_{22} 的前件之间也没有包含关系。

可以用一个与/或图表示产生式规则的事实和结论之间的与或关系。例如，规则 R 的与/或图如图 5.3 所示。

从产生式规则的形式上看，它与传统程序设计语言中的条件语句很相似，但是，两者之间存在以下根本的区别：

① 产生式规则具有自含性。所谓自含性有两个方面的含义：一是一条产生式规则仅仅描述了规则前件的条件与后件的结论之间的静态关系，因此，规则的正确性可以独立地得到保证；二是一条产生式规则不能直接调用另一条规则，一条规则是否被使用是由综合数据库中的当前内容来决定的。正是由于产生式规则具有自含性，才使得产生式系统的知识表示与知识推理能够分离。

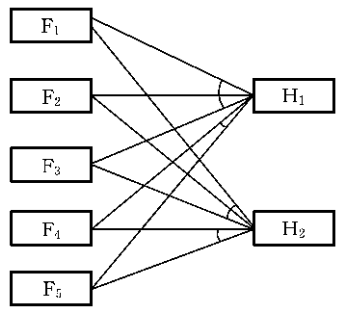

图 5.3　产生式规则与/或图

② 规则之间的控制流不是像传统语言程序那样从一条语

句向其下一条语句传递，而且满足条件的规则虽然可用但不一定立即执行，这将取决于产生式系统的冲突消解策略。这是传统语言程序与产生式系统行为特征的主要区别。

在 LISP 中，一条产生式规则的存储结构是一个表，通常一条规则存储形式是

(规则名
  (if (条件 1)(条件 2)…(条件 n))
  (then (结论 1)(结论 2)…(结论 m)))

一条规则的表有 3 个顶层元素，第一个元素是规则名，第二个元素是包括 if 在内的规则前件，第三个元素是包括 then 在内的规则后件。第二个元素和第三个元素又分别是一个表，它们的元素个数分别是 $n+1$ 个和 $m+1$ 个。

2. 规则库的存储结构

在 LISP 中，一个规则库的存储结构是一个分层结构的表。若规则库有 N 条规则，则规则库表就有 N 个顶层元素，每个顶层元素是一个规则子表，每个规则子表有 3 个元素，分别是规则名、包含 if 在内的规则前件和包含 then 在内的规则后件。一条规则的前件和后件是规则库表的第 3 层子表，前件子表和后件子表的第 1 个元素分别是 if 和 then，其余元素分别是规则前件中的多个条件和规则后件中的多个结论。一个规则库的各条规则的条件和结论之间的关系可以用一个与/或图来描述。

把若干条规则写入一个规则库的最简单的方法是使用 setq 函数，直接把若干条规则组成一个表赋值给一个规则库符号名。下面以动物识别专家系统为例，说明如何用 setq 函数直接建立规则库，并用与/或图来描述这个规则库。

例 5.5 建立动物识别专家系统的规则库，并用与/或图来描述这个规则库。规则库由 15 条规则组成，规则名分别是 $rule_1$，$rule_2$，…，$rule_{15}$，规则库的符号名为 rules。

用 setq 函数把 15 条规则组成一个表直接赋值给规则库 rules：

```
(setq  rules
  ((rule1
     (if  (animal  has  hair))                    若动物有毛发(F1)
     (then  (animal  is  mammal)))                则动物是哺乳动物(M1)
   (rule2
     (if  (animal  gives  milk))                  若动物有奶(F2)
     (then  (animal  is  mammal)))                则动物是哺乳动物(M1)
   (rule3
     (if  (animal  has  feathers))                若动物有羽毛(F9)
     (then  (animal  is  bird)))                  则动物是鸟(M4)
   (rule4
     (if  (animal  flies)                         若动物会飞(F10)
       (animal  lays  eggs))                      且生蛋(F11)
     (then  (animal  is  bird)))                  则动物是鸟(M4)
```

```
(rule5
  (if (animal eats meat))                    若动物吃肉(F3)
  (then (animal is carnivore)))              则动物是食肉动物(M2)
(rule6
  (if (animal has pointed teeth)             若动物有犀利牙齿(F4)
    (animal has claws)                       且有爪(F5)
    (animal has forward eyes))               且眼向前方(F6)
  (then (animal is carnivore)))              则动物是食肉动物(M2)
(rule7
  (if (animal is mammal)                     若动物是哺乳动物(M1)
    (animal has hoofs))                      且有蹄(F7)
  (then (animal is ungulate)))               则动物是有蹄类动物(M3)
(rule8
  (if (animal is mammal)                     若动物是哺乳动物(M1)
    (animal chews cud))                      且反刍(F8)
  (then (animal is ungulate)))               则动物是有蹄类动物(M3)
(rule9
  (if (animal is mammal)                     若动物是哺乳动物(M1)
    (animal is carnivore)                    且是食肉动物(M2)
    (animal has tawny color)                 且有黄褐色(F12)
    (animal has dark sports))                且有暗斑点(F13)
  (then (animal is cheetah)))                则动物是豹(H1)
(rule10
  (if (animal is mammal)                     若动物是哺乳动物(M1)
    (animal is carnivore)                    且是食肉动物(M2)
    (animal has tawny color)                 且有黄褐色(F12)
    (animal has black stripes))              且有黑色条纹(F15)
  (then (animal is tiger)))                  则动物是虎(H2)
(rule11
  (if (animal is ungulate)                   若动物是有蹄类动物(M3)
    (animal has long neck)                   且有长脖子(F16)
    (animal has long legs)                   且有长腿(F14)
    (animal has dark spots))                 且有暗斑点(F13)
  (then (animal is giraffe)))                则动物是长颈鹿(H3)
(rule12
  (if (animal is ungulate)                   若动物是有蹄类动物(M3)
    (animal has black stripes))              且有黑色条纹(F15)
```

```
  (then (animal is zebra)))                    则动物是斑马(H4)
(rule13
  (if (animal is bird)                         若动物是鸟(M4)
    (animal does not fly)                      且不会飞(F17)
    (animal has long neck)                     且有长脖子(F16)
    (animal has long legs)                     且有长腿(F14)
    (animal is black and white))               且有黑白二色(F18)
  (then (animal is ostrich)))                  则动物是鸵鸟(H5)
(rule14
  (if (animal is bird)                         若动物是鸟(M4)
    (animal does not fly)                      且不会飞(F17)
    (animal swims)                             且会游泳(F19)
    (animal is black and white))               且有黑白二色(F18)
  (then (animal is penguin)))                  则动物是企鹅(H6)
(rule15
  (if (animal is bird)                         若动物是鸟(M4)
    (animal flies well))                       且善飞(F20)
  (then (animal is albatross))))               则动物是信天翁(H7)
```

上述 setq 函数由 LISP 执行后，把一个表赋给变量 rules，这个表有 15 个顶层元素，每一个顶层元素是一条规则，每条规则都是有 3 个元素的一个表。

在上述规则的说明中，用 $F_1 \sim F_{20}$ 标记的是初始事实或证据，用 $M_1 \sim M_4$ 标记的是中间结论，用 $H_1 \sim H_7$ 标记的是最终结论。用标记表示 15 条规则为

R_1： $F_1 \rightarrow M_1$

R_2： $F_2 \rightarrow M_1$

R_3： $F_9 \rightarrow M_4$

R_4： $F_{10} \wedge F_{11} \rightarrow M_4$

R_5： $F_3 \rightarrow M_2$

R_6： $F_4 \wedge F_5 \wedge F_6 \rightarrow M_2$

R_7： $F_7 \wedge M_1 \rightarrow M_3$

R_8： $F_8 \wedge M_1 \rightarrow M_3$

R_9： $F_{12} \wedge F_{13} \wedge M_1 \wedge M_2 \rightarrow H_1$

R_{10}： $F_{12} \wedge F_{15} \wedge M_1 \wedge M_2 \rightarrow H_2$

R_{11}： $F_{13} \wedge F_{14} \wedge F_{16} \wedge M_3 \rightarrow H_3$

R_{12}： $F_{15} \wedge M_3 \rightarrow H_4$

R_{13}： $F_{14} \wedge F_{16} \wedge F_{17} \wedge F_{18} \wedge M_4 \rightarrow H_5$

R_{14}： $F_{17} \wedge F_{18} \wedge F_{19} \wedge M_4 \rightarrow H_6$

R_{15}： $F_{20} \wedge M_4 \rightarrow H_7$

这个动物识别专家系统的规则库的与/或图如图 5.4 所示。

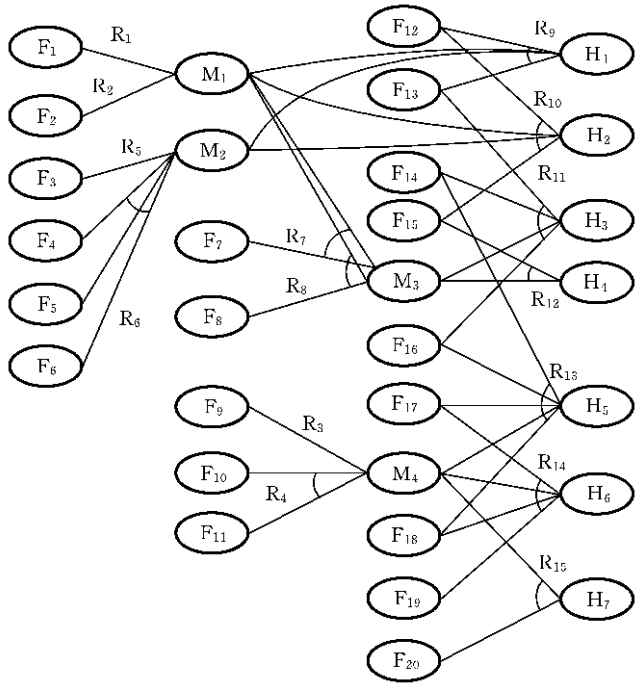

图 5.4　动物识别专家系统规则库与/或图

5.3.2　推理机及其实现

产生式系统有正向推理和反向推理两种基本推理方式。首先给出正向推理过程和反向推理过程的基本描述，然后给出用 LISP 语言编写的正向推理机和反向推理机。

1. 正向推理过程与反向推理过程

(1) 正向推理过程

正向推理是：根据在综合数据库中给出的已知事实，正向使用规则，即把规则的前件同当前数据库的内容进行匹配来选取可用规则；若有多条规则可用，则按冲突消解策略从中选择一条规则执行，将执行规则的结论添加到综合数据库中，……直至问题求解或没有可用规则为止。

(2) 反向推理过程

反向推理是：根据在综合数据库中给出的假设，反向使用规则，即把规则的后件同当前数据库的内容进行匹配来选取可用规则；若有多条规则可用，则按冲突消解策略从中选择一条规则，将该规则的前件添加到综合数据库中，……直至问题求解（假设成立所需要的全部证据和事实在数据库中）或没有可用规则为止。

2. 正向推理机

用 LISP 语言编制的产生式系统中的综合数据库的存储结构也是一个表，用 facts 作为综合数据库表的表名。

LISP 语言是函数型语言，因此，用 LISP 语言实现的正向推理机也是一个函数。首先给出正向推理机需要调用的几个函数的定义，最后给出实现的正向推理机。

(1) 函数 recall

函数表达式 (recall fact)

函数功能 判断变量 fact 中的一个事实是否在表 facts 中，若是，recall 返回值是 fact 中的事实；否则，返回 nil。

函数定义
```
( defun (recall fact)
    ( cond ((member fact facts) fact)
           (t nil)))
```

(2) 函数 test-if

函数表达式 (test-if rule)

函数功能 判断变量 rule 中的一条规则的前件包含的全部事实是否在表 facts 中，若是，则 test-if 返回 t；否则，返回 nil。

函数定义
```
(defun (test-if rule)
    (prog (ifs)
    (setq ifs (cdadr rule))
    loop
    (cond ((null ifs)(return t))
          (recall (car ifs))
          (t (return nil)))
    (setq ifs (cdr ifs))
    (go loop)))
```

(3) 函数 remember

函数表达式 (remember new)

函数功能 判断变量 new 中的一个事实是否在表 facts 中，若是，remember 返回 nil；否则，将 new 中的事实添加到表 facts 的表头，且 remember 返回 new 中的事实。

函数定义
```
(defun (remember new)
    (cond ((member new facts) nil)
          (t (setq facts (cons new facts)) new)))
```

(4) 函数 use-then

函数表达式 (use-then rule)

函数功能 判断变量 rule 中的一条规则的后件包含的全部结论是否在表 facts 中，若全部结论都在 facts 中，则 use-then 返回 nil；否则，将不在 facts 中的结论逐一添加到表 facts 中，

且 use-then 返回 t。

函数定义　(defun　(use-then　rule)
　　(prog　(thens　success)
　　(setq　thens　(cdddr　rule))
　　loop
　　(cond　((null　thens)(return　success))
　　((remember　(car　thens))
　　(print　(car　rule))
　　(print|DEDUCES|　(car　thens))
　　(　setq　success　t)))
　　(setq　thens　(cdr　thens))
　　(go　loop)))

在 use-then 的函数定义中两次调用了 LISP 输出函数 print，把输出流定向输出到显示终端。第 2 次调用的 print 函数的输出流使用了转义符“|”，两个“|”之间的所有字符都作为普通字符而失去原定义的语法含义。屏幕显示 rule 中规则名，换行显示 DEDUCES 和 thens 中的第一个元素。

(5) 函数 try-rule

函数表达式　(try-rule　rule)

函数功能　判断规则变量 rule 中的一条规则的前件包含的全部事实是否在表 facts 中，若全部事实都在 facts 中，且规则后件有不在 facts 中的结论，则把不在 facts 中的结论逐一添加到表 facts 中，try-rule 返回 t；否则，try-rule 返回 nil。

函数定义　(defun　(try-rule　rule)
　　(and　(test-if　rule)(use-then　rule))

(6) 函数 step-forward

函数表达式　(step-forward　rules)

函数功能　逐次扫描规则库 rules 中的规则，若发现 rules 中有一条可用规则，即该规则的前件包含的全部事实在表 facts 中，则把该规则的后件中不在 facts 中的所有结论添加 facts 中，且 step-forward 返回 t；若 rules 中没有一条可用规则，则 step-forward 返回 nil。

函数定义　(defun　(step-forward　rules)
　　(prog　(rule-list)
　　(setq　rule-list　rules)
　　loop
　　(cond　((null　rule-list)(return　nil))
　　((try-rule　(car　rule-list))(return　t)))
　　(setq　rule-list　(cdr　rule-list))
　　(go　loop)))

(7) 正向推理机函数 deduce

step-forward 函数只从 rules 中选择一条可用规则，并使用该规则对 facts 作一次扩充。

推理机要完成推理，通常要对 facts 进行多次扩充，也就是说，按扩充了的 facts 继续从 rules 中选择可用规则，对 facts 再扩充，直至没有可用规则可选，不能对 facts 再扩充为止。正向推理机 deduce 可用 step-forward 函数来定义。

函数表达式 (deduce facts)

函数功能 连续不断地从规则库 rules 中选择可用规则，每选择一条可用规则，就把该规则的后件中不在 facts 中的所有结论添加到 facts 中，对 facts 扩充，由扩充了的 facts 来选择下一条可用规则对 facts 再次扩充，直至没有可用规则可选为止。若曾找到一条可用规则对 facts 进行过一次扩充，则 deduce 返回 t；否则，deduce 返回 nil。

函数定义
```
(defun (deduce facts)
       (prog (progress)
         loop
         (cond ((step-forward rules)(setq progress t))
                       (t (return progress)))
         (go loop)))
```

例 5.6 对于动物识别专家系统，若已知的初始事实是 F_{13}、F_{12}、F_3 和 F_1，应用正向推理机 deduce，说明推理过程和得出的推理结论。

使用 setq 函数把已知的初始事实赋值给事实表 facts：
```
(setq facts
  ((animal has dark spots)
    (animal has tawny color)
    (animal eats meat)
    (animal has hair)))
```
即有 facts=(F_{13} F_{12} F_3 F_1)。

调用正向推理机(deduce facts)，使用的规则库 rules 是例 5.5 给出的规则库。推理过程说明如下。

① 在 rules 中查找规则前件的全部条件在当前 facts=(F_{13} F_{12} F_3 F_1)中的可用规则，首先找到规则 R_1，并把 R_1 后件中不在 facts 中的结论 M_1 添加到 facts 中，扩充 facts 为 facts=(M_1 F_{13} F_{12} F_3 F_1)。

实际上，对 facts=(F_{13} F_{12} F_3 F_1)，还有另一条可用规则 R_5，因为 R_5 的前件 F_3 也在当前 facts 中。但是，由 step-forward 函数定义的冲突消解策略是：若有多条可用规则，则按可用规则在规则库表 rules 中的顺序选择第一条可用规则。因此，首先选择 R_1 把 facts 扩充为 facts=(M_1 F_{13} F_{12} F_3 F_1)。

② 对当前 facts 在 rules 中查找可用规则，仍然找到规则 R_1，但 R_1 的后件结论 M_1 已在 facts 中，因此不会执行规则 R_1。继续查找可用规则，找到规则 R_5，因为 R_5 的后件结论 M_2 不在当前 facts 中，故执行 R_5，把 R_5 不在 facts 中的结论 M_2 添加到 facts 中，扩充 facts 为 facts=(M_2 M_1 F_{13} F_{12} F_3 F_1)。

③ 对当前 facts 在 rules 中继续查找可用规则，只有规则 R_9 的前件包含的全部条件 F_{13}、

F_{12}、M_1和 M_2在 facts 中，因此，R_9是可用规则；且 R_9的后件结论 H_1不在 facts 中，故而执行 R_9，把 R_9的结论 H_1扩充到 facts 中，使得 facts=(H_1　M_2　M_1　F_{13}　F_{12}　F_3　F_1)。

④ 对当前 facts，在 rules 中找不到规则的前件包含的全部条件在 facts 中且后件有不在 facts 中的结论的任何规则，经过 4 步推理，正向推理终止，推理机函数 deduce 返回变量 progress 的值为 t。

实际上，这个推理过程是按图 5.4 所示的规则库与/或图的一个子图的正向进行推理的。

推理过程终止后，解释器取当前 facts 中的第一个元素得出推理的结论是 H_1，即(animal is　cheetah)。

3. 反向推理机

反向推理是对给定的一个假设 fact，判断其是否为真。确定一个假设 fact 是否为真的函数为 verify，verify 函数的定义由下述 4 部分组成：

① 若 fact 已在 facts 中，则 fact 为真，函数 verify 终止。

② 若 fact 不在 facts 中，则从 rules 中选出所有后件含有 fact 的可用规则组成可用规则表 relevant1。若 relevant1 为空，则直接向用户询问 fact 的真假，若用户确认 fact 为真，则把 fact 添加到 facts 中，函数 verify 终止。

③ 若 relevant1 不为空，则调用函数 try-rule 逐条作用于 relevant1 中的可用规则，试看能否直接推出 fact，若推出 fact，则 fact 为真，把 fact 添加到 facts 中，函数 verify 终止。

④ 若不能直接推出 fact，则调用函数 try-rule＋逐条作用于 relevant1 中的可用规则，试看能否间接推出 fact，若推出 fact，则 fact 为真，把 fact 添加到 facts 中，函数 verify 终止。

函数 try-rule＋ 的间接推出与函数 try-rule 的直接推出的不同之处在于：try-rule 调用函数 test-if 来判断一条规则前件包含的全部事实是否在 facts 中，test-if 调用函数 recall 来判断前件中的一个事实是否为真。try-rule＋调用函数 test-if＋来判断一条规则前件包含的全部事实是否在 facts 中，而 test-if＋是调用函数 verify 来判断前件中的一个事实是否为真。因此，verify 是一个递归函数。

首先给出反向推理机需要调用的几个函数的定义，最后给出实现的反向推理机。

(1) 函数 thenp

函数表达式　(thenp　fact　rule)

函数功能　判断 fact 是否在规则变量 rule 中的一条规则的后件中，若是，则函数 thenp 返回规则后件从 fact 开始的余下表；否则，thenp 返回 nil。

函数定义
```
(defun (thenp fact rule)
       (member fact (cdddr rule)))
```

(2) 函数 inthen

函数表达式　(inthen　fact)

函数功能　从规则库表 rules 中送出所有的可用规则组成一个可用规则表，表中每个元素是一条可用规则。若规则的后件含有 fact，则这条规则是一条可用规则，且函数 inthen 返回可用规则表。若没有一条可用规则，则 inthen 返回空表。

函数定义 (defun (inthen fact)
(apply 'append
(mapcar '(lambda (rule)
(cond ((thenp fact rule)(list rule))
(t nil))
(rules)))))

在函数 inthen 的定义中，调用了映射函数 mapcar，mapcar 把无名函数 lambda 定义的功能映射到规则库表 rules 中的各条规则上去。lambda 函数定义的功能是：对 rules 中的一条规则 rule 执行 cond 函数定义的功能，若 fact 在规则 rule 的后件中，则由 list 函数把规则 rule 形成一个表，由 lambda 函数返回；若 fact 不在规则 rule 的后件中，则 lambda 函数返回一个空表。因此，mapcar 函数从规则库表 rules 中选出多个可用规则，每个可用规则是一个非空表。函数 inthen 用函数 apply 来定义高阶函数 append。因为 append 的变量是函数 mapcar，所以定义的函数 append 是一个高阶函数。函数 append 把 mapcar 返回的多个表(包括返回的空表)拼接成一个表，这个表就是可用规则表。当然，若找不到一条可用规则，可用规则表就是一个空表。

(3) 函数 test-if+

函数表达式 (test-if+ rule)

函数功能 直接或间接地反向推理确认规则变量 rule 中的一条规则的前件包含的所有条件是否都为真，若是，则函数 test-if+返回 t；否则，test-if+返回 nil 。

函数定义 (defun (test-if+ rule)
(prog (ifs)
(setq ifs (cdadr rule))
loop
(cond ((null ifs)(return t))
(verify (car ifs))
(t (return nil)))
(setq ifs (cdr ifs))
(go loop)))

在函数 test-if+的定义中，调用了函数 verify。函数 verify 定义的功能是：直接或间接地反向推理来确认 ifs 中的第一个元素(规则 rule 中的前件包含的一个条件)是否为真。

(4) 函数 try-rule+

函数表达式 (try-rule+ rule)

函数功能 若规则变量 rule 中的一条规则的前件包含的所有条件都能直接或间接地反向推理确认为真，且规则的后件中有不在 facts 中的结论，则把规则后件中不在 facts 中的结论添加到 facts 中，函数 try-rule+返回 t；否则，try-rule+返回 nil。

函数定义 (defun (try-rule+ rule)
(and (test-if+ rule)(use-then rule)))

(5) 函数 verify

函数表达式　(verify　fact)

函数功能　直接或间接地反向推理确认变量 fact 中的一个假设是否为真，若为真，则函数 verify 返回 t；否则，verify 返回 nil。

函数定义

```
(defun (verify fact)
  (prog (relevant1 relevant2)
    (cond ((recall fact)(return t)))
    (setq relevant1 (inthen fact))
    (setq relevant2 relevant1)
    (cond ((null relevant1)
           (cond ((member fact askes)(return nil))
                 ((and (print |IS THIS TRUE:|fact)(read))
                  (remember fact)
                  (return t))
                 (t (setq asked (cons fact asked))
                    (return nil))))
    loop1
    (cond ((null relevant1)(go loop2))
          ((try-rule(car relevant1))(return t)))
    (setq relevant1 (cdr relevant1))
    (go loop1)
    loop2
    (cond ((null relevant2)(go exit))
          ((try-rule+(car relevant2))(return t)))
    (setq relevant2 (cdr relevant2))
    (go loop2)
    exit
    (return nil)))
```

在函数 verify 的定义中，可分为以下 4 部分。

① 首先调用 recall 函数，直接确认假设 fact 是否在 facts 中，若是，则 fact 为真，函数 verify 终止并返回 t；若不是，则由(inthen　fact)从 rules 中选出所有的可用规则组成一个表 relevant1，可用规则是指规则的后件含有 fact，并生成 relevant1 的一个拷贝 relevant2 。

② 若可用规则表 relevant1 是一个空表，则由用户直接确认假设 fact 是真或假。用户在屏幕提示“IS　THIS　TRUE:fact ”的询问下，通过键盘直接确认 fact 是真或假。若用户通过 LISP 输入函数 read 用键盘回答为 t，则 and 函数返回 t，执行(remember　fact)。若 fact 不在 facts 中，则 remember 把 fact 加到 facts 的表头，且函数 verify 终止并返回 t；若 fact 在 facts 中，则 remember 不把 fact 加到 facts 中，且函数 verify 也终止并返回 t。

若用户通过键盘直接回答为假，则 and 函数返回 nil，执行(setq asked (cons fact asked))，把 fact 加到表 asked 的表头，且函数 verify 终止并返回 nil 。

③ 若可用规则表 relevant1 不是空表，则不由用户来直接确认 fact 是真或假，而由 verify 函数从标号 loop1 开始进行直接反向推理来确认假设 fact 是真或假。

逐条使用 relevant1 中的可用规则，执行(try-rule ((car relevant1))，若规则(car relevant1)的前件包含的全部条件在 facts 中，则把该规则不在 facts 中的后件结论添加到 facts 中，且函数 verify 终止并返回 t；否则，对 relevant1 中的下一条规则执行 try-rule 函数功能。只要使用了一条可用规则直接反向推理确认了 fact 为真，verify 就终止并返回 t。

若逐条使用了 relevant1 中所有可用规则都不能直接反向推理确认 fact 为真，则转到从标号 loop2 开始的间接反向推理。

④ 间接反向推理是通过逐条对 relevant2 中所有可用规则调用函数 try-rule＋来进行的。如前所述，try-rule＋调用函数 test-if＋，而 test-if＋又调用函数 verify。只要使用了一条可用规则间接反向推理确认了假设 fact 为真，函数 verify 就终止并返回 t；若逐条使用了所有的可用规则都不能确认 fact 为真，则 verify 也终止并返回 nil。

(6) 反向推理机函数 diagnose

verify 函数只能直接和间接地确认一个假设是真或假，反向推理机需要对用户提出的多个假设确定其真或假。对用户在假设表 hypo 中设置的多个假设，反向推理机将使用规则库 rules 对多个假设逐一确定其真或假。

函数表达式 (diagnose hypo)

函数功能 逐一确定假设表 hypo 中多个假设的真或假，对确定为真的假设给出屏幕输出说明。

函数定义
```
(defun (diagnose hypo)
  (prog (poss)
    (setq poss hypo)
    loop
    ( cond ((null poss)
       (print |NO HYPOTHESES CAN BE CONFIRMED|)
       (return))
       ((verify (car poss))
       (print |HYPOTHESES|(car poss))
       (print |IS TRUE|)))
    (setq poss (cdr poss))
    (go loop)))
```

函数 diagnose 首先把用户的假设表 hypo 生成一个拷贝 poss，若 poss 是一个空表，则屏幕提示“NO HYPOTHESES CAN BE CONFIRMED”，且 diagnose 终止；若 poss 不是空表，则调用 verify 函数逐一确认 poss 表中的假设，对确定为真的假设，屏幕输出说明：“HYPTHESES×××IS TRUE”，其中，××××是这个假设的内容，直至对 poss 中的所

有假设都确认完后，diagnose 终止。

例 5.7　对于动物识别专家系统，若用户要求确定的假设是 H_1，应用反向推理机 diagnose 确定假设 H_1 是否成立，说明推理过程。

使用 setq 函数把假设 H_1 赋给假设表 hypo：

```
(setq  hypo
    ((animal  is  cheetah)))
```

即有 hypo=(H_1)。

调用反向推理机(diagnose hypo)，使用的规则库 rules 是例 5.1 给出的规则库。推理过程说明如下。

① 由(diagnose hypo)先复制 poss=(H_1)，然后调用(verify H_1)，此时，facts=( )。

首先调用(recall H_1)，由于 fact=H_1，不在 facts 中，调用(inthen H_1)，从 rules 中选出后件含有 H_1 的可用规则只有 R_9，故 relevant1=relevant2=(R_9)。

由于 relevant1=(R_9)≠( )，故调用(try-rule R_9)；因为 R_9 的前件的全部条件不在 facts 中，故对 H_1 进行间接反向推理。

由于 relevant2=(R_9)≠( )，故调用(try-rule+ R_9)。它首先调用(test-if+ R_9)，把 R_9 的前件包含的全部条件赋给 ifs，即有 ifs=(F_{12} F_{13} M_1 M_2)。在(test-if+ R_9)中，将顺序调用(verify F_{12})、(verify F_{13})、(verify M_1) 和(verify M_2)。

② 调用(verify F_{12})，此时，facts=( )。

首先调用(recall F_{12})，由于 fact=F_{12} 不在 facts 中，调用(inthen F_{12})，从 rules 中找不到后件含有 F_{12} 的可用规则，故 relevant1=relevant2=( )。

由于 relevant1=( )，屏幕出现提示："IS THIS TRUE:F_{12}"。若用户确认 F_{12} 为真，则把 F_{12} 加到 facts 的表头，使 facts=(F_{12})。

③ 调用(verify F_{13})，此时，facts=(F_{12})。

类似(verify F_{12})，若用户在屏幕提示："IS THIS TRUE:F_{13}"的询问下，确认 F_{13} 为真，则把 F_{13} 加到 facts 的表头，使 facts=(F_{13} F_{12})。

④ 调用(verify M_1)，此时，facts=(F_{13} F_{12})。

首先调用(recall M_1)，由于 facts=M_1 不在 facts 中，调用(inthen M_1)，从 rules 中选出后件含有 M_1 的可用规则有 R_1 和 R_2，故 relevant1=relevant2=(R_1 R_2)。

由于 relevant1=(R_1 R_2)，顺序调用(try-rule R_1)和(try-rule R_2)，因为 R_1 的前件条件 F_1 和 R_2 的前件条件 F_2 都不在 facts 中，故对 M_1 进行间接反向推理。

由于 relevant2=(R_1 R_2)，顺序调用(try-rule+ R_1)和(try-rule+ R_2)。前者首先调用(test-if+ R_1)，把 R_1 的前件条件赋给 ifs，即有 ifs=(F_1) 。在(test-if+ R_1)中将调用(verify F_1)。同样，后者将产生调用(verify F_2)。

调用(verify F_1)，类似(verify F_{12})，若用户在屏幕提示"IS THIS TRUE:F_1"的询问下，确认 F_1 为真，则把 F_1 加到 facts 的表头，使 facts=(F_1 F_{13} F_{12})。

调用(verify F_2)，若用户在屏幕提示"IS THIS TRUE:F_2"的询问下，确认 F_2 为真，则把 F_2 加到 facts 的表头，使 facts=(F_2 F_1 F_{13} F_{12})。

⑤调用(verify M_2),此时,facts=(F_2 F_1 F_{13} F_{12})。

首先调用(recall M_2),由于 fact=M_2不在 facts 中,调用(inthen M_2),从 rules 中选出后件含有 M_2的可用规则 R_5和 R_6,故 relevant1=relevant2=(R_5 R_6)。

由于 relevant1=(R_5 R_6),顺序调用(try-rule R_5)和(try-rule R_6),因为 R_5的前件条件 F_3与 R_6的前件包含的全部条件 F_4、F_5和 F_6都不在 facts 中,故对 M_2进行间接反向推理。

由于 relevant2=(R_5 R_6),顺序调用(try-rule+ R_5)和(try-rule+ R_6),前者产生调用(verify F_3),后者产生调用(verify F_4)、(verify F_5)和(verify F_6)。在执行这 4 个调用时,若用户在相应屏幕提示下分别确认 F_3、F_4、F_5和 F_6为真,则使 facts=(F_6 F_5 F_4 F_3 F_2 F_1 F_{13} F_{12})。

至此,调用(verify H_1)执行完毕,屏幕输出:"HYPOTHESES H_1 IS TRUE"。由(setq poss (cdr poss))使 poss 为空表,反向推理机(diagnose hypo)终止。

实际上,这个推理过程是按图 5.4 所示的规则库与/或图的一个子图的反向进行推理的。

由例 5.7 所见,为了确认用户提出的假设是真或假,反向推理机用规则的后件匹配假设来选择可用规则,然后,对可用规则前件包含的全部条件,逐一向用户询问。对用户不能确认的条件,以该条件作为中间假设再选择可用规则。直至支持所有中间假设为真的所有条件都被用户确认为真,才能得出由用户提出的假设是真的判断。反向推理机为用户推断假设是否为真提供了一个以假设为根节点的与/或树,树的所有的叶节点是需要用户确认的初始条件。反向推理机并不能随意确认条件的真或假,它只是引导用户进行反向推理,减少了推理的盲目性。

5.3.3 元知识与元规则

产生式规则是完全自含的,一条规则不能直接调用另一条规则,一条规则是否可用由综合数据库(事实表)的当前内容来决定。这种规则激活机制使产生式系统推理机制具有单纯明确的结构,但是,对于大型产生式系统来说,推理的效率受到严重影响。因此,希望能直接地、显式地表示出激活规则的规则控制知识,用于选择可用规则和激活规则,提高推理效率。这种控制性知识称为元知识,或者更一般地说,元知识是关于知识的知识。

在产生式系统中使用的元知识由元事实(Metafact)和元规则(Metarule)构成。元事实用于描述领域知识表示的结构、规则之间的控制约束关系、知识的适用范围等。元规则用来说明目标规则如何使用的规则,其作用是指导问题求解的推理过程。为了区别这两种不同的推理层次,称使用目标规则进行的推理为目标级推理,它是问题求解实际执行的推理;称使用元规则进行的推理为元级推理,元级推理在目标级推理过程中决定下一步需要执行的规则,从而完成冲突消解,或者改进已有的冲突消解策略,提高目标级推理的效率。

1. 元知识的用途

在专家系统中,知识可以分为以下 4 种主要类型。

① 启发性知识,即经验知识。例如,在动物识别专家系统中,根据经验知识确定的动物外

形、能力等特征与动物类别(或动物名称)之间的联系规则。

② 策略性知识,即控制性知识,是如何使用知识的知识。例如,在动物识别专家系统的正向推理机 deduce 中,从规则库 rules 中选择可用规则的顺序是按可用规则在 rules 中的排列次序进行的。这是如何使用目标规则最简单的控制性知识。

③ 结构性知识,即描述领域对象知识结构的知识。例如,可以用结构性知识来描述一个领域对象目标规则结构及规则之间关系的结构。

④ 支撑性知识,即用于论证规则的知识。它给出规则成立的理由、提供者的权威性,甚至包括一些用于参考的引文。

除第一类知识外,其他三类知识均属于元知识的范畴。

元知识在产生式系统中有以下主要用途。

(1) 描述规则的控制约束

目标规则的自含性使目标规则之间没有调用关系,规则库与推理机是相互独立的,由数据库的当前内容和推理机制决定规则执行的次序。有时需要显式地控制规则执行的次序,就可以利用元知识来描述规则之间的执行次序。以医疗诊断专家系统 MYCIN 中的元规则为例来说明。

if　① 感染是骨盆脓肿
　　② 存在前件含有“肠杆菌”的可用规则
　　③ 存在前件含有“革兰氏阳性杆菌”的可用规则
then　先使用满足②的规则,后使用满足③的规则

这是一条对目标规则排序所使用的元规则。由于肠杆菌感染和骨盆脓肿关系密切,因此,当发现患者有骨盆脓肿时,应首先考虑肠杆菌感染的可能性,其次考虑革兰氏阳性杆菌感染的可能性。

如果元规则的形式与目标规则的形式完全相同,那么元级推理的基本模式也就可以与目标级推理相同。不同的是元规则对目标规则进行推理,得出目标规则使用的排序关系;目标规则对领域问题进行推理,得出领域问题的解。

(2) 描述知识的适用范围

对一个专家系统需要使用的知识可以按某些知识特征将其划分为若干适用范围,利用元知识来描述知识的适用范围,可以避免知识的盲目使用,提高推理的效率。例如,陆地植物生态知识可按海拔高度特征划分若干适用范围。设有一组知识适用范围是海拔高度 1000～1200m,将这一描述知识适用范围的知识用元规则形式描述,在目标级推理过程中,用这个元规则检查知识的海拔高度特征,一旦发现推理结果的海拔高度不属于(1000,1200),就中止推理过程。

(3) 描述通用问题求解策略

有关通用问题求解策略的元规则可以是域相关的,也可以是域无关的。一条域相关元规则可以是关于领域的特定控制策略性知识,例如,在医疗诊断专家系统中,可以用元规则表示酗酒者或烧伤患者易于受某种特定的细菌感染,以指出在推理中遇到这种情况时首先应考虑的问题。一条相对域无关的元规则可以在产生较大的搜索空间之前建议系统采用产生相对较

小搜索空间的规则。元规则与领域的相关程度与所表示知识的抽象程度有关。例如，下述是一条领域相关的元规则：

if x 是酗酒者

then 先使用与酗酒有关的疾病规则，再使用确定其他疾病的规则

可以抽象成下述相对域无关的元规则：

if x 有属性 y，y 是 z 的某种原因，存在与 z 相关的规则

then 先使用与 z 相关的规则，再使用其他可用规则

相对域无关的元规则可以应用于较多的领域，域相关的元规则只能在某个特定相关的领域中应用。域相关元规则可以看成是相对域无关元规则的一个特例。

元规则是表示推理控制策略的一种强有力的手段，合理地使用元规则可以大大提高产生式系统的推理速度。但是，对元规则也应该有选择地使用，如果系统需要花太多时间进行元级推理来选择用哪一条目标规则进行目标级推理，反而可能会降低系统推理的效率。

2. 元知识的表示与使用模式

在产生式系统中，元知识一般采用与目标级知识相同的表示形式，并作为一个知识实体与目标级知识共存于知识库中。元规则与目标规则都具有相同产生式规则形式，共存于一个规则库中，这种表示与使用模式的主要优点在于：元级推理与目标级推理可共享一个推理机。

为了保证元规则优先执行，可以为规则增加一个优先数说明模式，在元规则的优先数说明模式中设定较大的优先数，在目标规则的优先数说明模式中设定较小的优先数。这样，当元规则与目标规则由当前数据库内容确定为可用规则时，将优先执行元规则。在专家系统通用型语言工具 OPS(Official Production System)和 CLIPS (C Language Integrated Production System)中，就使元规则与目标规则共存于规则库中，并为规则增加了一个优先数说明模式。

当元级知识与目标级知识明显分离时，即系统分设目标级规则库和元级规则库时，系统将增加一个调度程序。当有多个目标规则可用时，由调度程序根据元级规则与可用目标规则冲突集的匹配情况，从中选择一条可用规则执行。

5.4 解释机制与解释器

在专家系统中，由于知识库与推理机是分离的，系统运行的每一步及系统激活哪一条规则都是由当时的环境信息(综合数据库当前内容、推理控制信息等)和推理机共同决定的，即使采用简单的推理控制策略，动态变化的综合数据库当前内容和多条可用规则也使得程序员难以了解系统的执行行为，因此希望专家系统能够用人们易于理解的方式来解释自身的推理过程。专家系统的解释就是对系统设计者或用户提出的问题给出解释或说明，专家系统的解释器就是专家系统中为完成解释而设置的程序模块。

5.4.1　解释的方法

系统设计者和不同的用户对解释器提出的问题各有侧重，提问题的风格也不相同。一般用户经常询问的是系统得出某个结论的表层原因；一个领域专家更注重系统求解问题的质量；领域的学生更希望了解问题求解的详细过程；知识工程师关心的是问题求解的结论及解释是否与专家的思路吻合，因而更注意审查系统内部执行的流程。

此外，专家系统本身的知识结构与解释机制也有很大的关系，采用适当的知识表示方法可以得到更易表达和更易接受的解释。显然，知识的描述性表示比过程性表示对解释机制实现有效的解释更有利。知识库的层次结构模型不仅有利于提高解释文本的质量，而且便于实现解释的组织和修改。

解释机制可以有以下几种不同的实现方法。

1. 预置文本与路径跟踪法

最简单的解释方法是预置文本，即把问题的解释预先用自然语言或其他易于理解的形式写好，插入程序段或相应的数据库中。在推理过程中或推理之后，一旦用户询问到已有预置解释文本的问题，就只需把相应的解释文本填入解释框架，组织成对这个问题的解释，提交给用户。对于模糊推理，还需要把模糊语言变量转化为相应的模糊修饰词。

预置文本的解释方法简单直观，知识工程师在编制解释的预置文本时，可以针对不同用户的要求编制不同的解释文本。其缺点是：对每一个可能的问题都要编制解释预置文本，甚至对一个问题要编制几个解释预置文本，大大增加了系统开发的工作量。

路径跟踪法是对推理过程进行跟踪，将问题求解所使用的知识自动记录下来。当用户提出需要解释时，解释器向用户显示问题求解路径。路径跟踪法向用户提供 Why 解释和 How 解释，Why 解释用于回答用户关于“系统为什么需要提出这样的问题”的询问，How 解释用于回答用户关于“系统是怎样得出这样的结论”的询问。

实现 Why 解释比较简单，系统从推理过程中确定导致这个问题的规则，将该规则向用户解释即可。实现 How 解释需要从这个结论出发，把导致这个结论的推理链中涉及的有关规则或知识组织成解释文本，告诉用户是怎样推理得出这个结论的。

2. 策略解释法

在专家系统的许多实际应用中，用户往往不满足于系统简单地告诉他们是怎样一步一步得到问题的结论的，而是要求系统给出求解问题所采用的基本方法和手段。基于认识论和心理学的考虑，为了让用户在更高的层次上把握系统对问题进行求解的行为特征，Clancey 等人在他们开发的 NEOMYCIN 系统中提出了一种新的解释方法，这就是策略解释法。

策略解释法向用户解释的是与问题求解策略有关的策略和方法，包括策略的抽象表示及其策略使用过程中产生的关于问题求解的策略解释。为实现策略解释，NEOMYCIN 在推理控制和知识库的构造上提供了有力的支持。它使用元规则表示策略知识，并与领域知识分离；

按分类学的观点对疾病进行分类组织，使得假设的推理空间能够用分类知识显式地表示出来；策略知识、分类知识和把数据与假设联系起来的经验知识被分开编码存储。

可以说，策略解释法的实质是对表示推理策略的元规则进行的一种路径跟踪法。

3. 自动程序员方法

通过跟踪推理过程中规则的执行而给出的解释仅仅描述了系统的行为过程，没有论证其行为的合理性。行为合理性的解释与行为的背景知识有关，而这些知识一般没有组织到系统中去。行为合理性已不是 Why 解释所能回答的问题，它是要求系统在比领域知识和控制知识更高、更抽象的层次上将设计思想向用户说明的问题。

为了解决这一问题，Swartout 在 XPLAIN 系统的设计中提出了自动程序员解释方法。其基本思想是：在设计一个专家咨询系统中，对领域知识进行描述的同时，把自动程序员程序模块嵌入其中，通过自动程序员把描述性领域知识转化成可执行的程序，并产生有关程序行为的合理性说明，从而向用户提供一个非常有力的解释机制。

4. 解释型专家系统

1985 年，Neches 等人在 XPLAIN 系统研究的基础上，提出了一种新的专家系统设计模式，即解释型专家系统(Explanable Expert System)。其基本思想是把专家系统的设计与解释机制的设计进行全盘考虑，从而给出更合理的解释方案。

5.4.2 解释器及其实现

本节将给出 LISP 语言实现的产生式系统的一个解释器，这个解释器的解释机制采用路径跟踪方法，可向用户提供 Why 解释和 How 解释。路径跟踪法需要把推理过程中使用过的规则自动记录下来，才能向用户提供解释，为此，需要重新定义推理机中的有关函数，对其增加记录功能。

1. 有关函数的定义

(1) 正向推理机中函数 try-rule 的重新定义

函数表达式 (try-rule rule)

函数功能 判断规则变量 rule 中的一条规则的前件包含的全部事实是否在表 facts 中，若全部事实都在 facts 中，且规则后件有不在 facts 中的结论，则把不在 facts 中的结论逐一添加到表 facts 中，并把 rule 中的这条规则添加到表 rules-used 的表头，try-rule 返回 t；否则，try-rule 返回 nil。

函数定义

```
(defun (try-rule rule)
       (cond ((and (test-if rule)(use-then rule))
              (setq rules-used (cons rule rules-used))
              t)))
```

(2) 反向推理机中函数 try-rule+的重新定义

函数表达式　(try-rule+　rule)

函数功能　若规则变量 rule 中的一条规则的前件包含的所有条件都能直接或间接地反向推理确认为真，且规则后件中有不在 facts 中的结论，则把规则后件中不在 facts 中的结论都添加到 facts 中，并把 rule 中的这条规则添加到表 rules-used 的表头，try-rule+返回 t ；否则，try-rule+返回 nil 。

函数定义
```
(defun (try-rule+ rule)
  (cond (and (test-if+ rule)(use-then rule))
        (setq rules-used (cons rule rules-used))
        t)))
```

(3) 函数 usedp

函数表达式　(usedp　rulen)

函数功能　若推理过程中使用过由规则名变量 rulen 记载的一个规则名指定的规则，则 usedp 返回 t ；否则，usedp 返回 nil。

函数定义
```
(defun (usedp rulen)
  (prog (poss)
  (setq poss rules-used)
  loop
  (cond ((null poss)(return nil))
        ((equat rulen (car poss))(return t)))
  (setq poss (cdr poss))
  (go loop)))
```

2. How 解释

用 LISP 语言设计的解释器，其中各种解释被定义为相应的函数，在用户的询问下，调用相应的函数，在屏幕上给出回答。How 解释用于回答用户关于“系统是怎样得出这一结论”的询问。

函数表达式　(how　fact)

函数功能　告诉用户变量 fact 中结论是怎样得出的，若 fact 中的结论或假设经推理确认为真，则找到支持这一结论或假设成立的规则，显示该规则前件的所有事实。若 fact 中的结论或假设是已在 facts 中的给定事实，则给出相应的说明。否则，fact 中的结论或假设没有被确认。

函数定义
```
defun (how fact)
  (prog (poss success)
  (setq poss rules-used)
  loop
  (cond ((null poss)
```

```
(cond (success(return  t))
      ((recall  fact)(print  fact | WAS GIVEN |)(return  t))
      (t  (print  fact | IS NOT ESTABLISHED |)(return
      nil))))
((thenp  fast  (car  poss))
(setq  success  t)
(print  fact | IS DEMONSTRATED BY |)
(mapcar '(lambda  (a)(print  a)
                  (cdadr (car  poss))))
(setq  poss (cdr  poss))
(go  loop)))
```

若使用过规则表 rules-used 且不为空，则调用函数 thenp 对规则表 rules-used 中的逐个规则进行判断，判断其 fact 是否是规则后件中的一个结论，若是，则逐一把这些规则的前件包含的所有事实作为“是怎样得出 fact 结论”的解释，并在屏幕上给出说明：

fact 中的结论“IS DEMONSTRATED BY”规则前件的所有事实

若使用过规则表 rules-used 且为空，或 rules-used 中没有一条规则的后件中有 fact 结论，则调用函数 recall 进行判断，判断 fact 是否是 facts 中的一个给出的事实，若是，则在屏幕上给出说明：

fact 中事实“WAS GIVEN”

若不是上述两种情况，则在屏幕上直接给出说明：

fact 中的假设“IS NOT ESTABLISHED”

3. Why 解释

Why 解释用于回答用户关于“为什么需要这一事实”的询问。

函数表达式 (Why fact)

函数功能 告诉用户为什么需要变量 fact 中的事实，若 fact 中的事实在推理过程中作为一条规则的前件事实被使用过，则找到这条规则，向用户说明 fact 中的事实用以支持这条规则的后件结论。若 fact 中的事实是已在 facts 中的给定事实，则给出相应的说明。否则，fact 中的事实没有被确认。

函数定义

```
(defun  (why  fact)
  (prog  (poss  success)
    (setq  poss  rules-used)
    loop
    (cond  ((null  poss)
      (cond  (success  (return  t))
             ((recall  fact  )(print  fact | WAS HYPOTHESIS |)
             ( return t))
```

```
        (t  print  fact | IS NOT ESTABLISHED | ) ( return  nil))))
    ((ifp  fact  (car  poss))
    (setq  success  t)
    (print  fact | NEEDED TO SHOW | )
    (mapcar  '(lambda  (a)  (print  a)
                            (cdaddr  (car  poss))))
  (setq  poss  (cdr  poss))
  (go  loop)))
```

其中，函数 ifp 用于判定 fact 是否在规则的 if 部分中。ifp 函数定义：

```
(defun  (ifp  fact  rule) (member  fact  (cdadr  rule)))
```

若 fact 中的事实在推理过程中作为使用过规则表 rules-used 中某些规则的前件事实使用过，则逐一把这规则的后件结论作为“为什么需要 fact 事实”的解释，在屏幕上给出说明：fact 中的事实“NEEDED TO SHOW”规则后件的结论。

若 fact 中的事实没有被规则使用过，则判断 fact 是否是 facts 中已被确认的事实，若是，则屏幕上给出说明：fact 中事实“WAS HYPOTHESIS”。

若不是上述两种情况，则屏幕上直接给出说明：fact 中的事实“IS NOT ESTABLISHED”。

5.5　知识获取

拥有知识是专家系统有别于其他计算机软件系统的重要标志，而知识的质量和数量又是决定专家系统性能的关键因素。知识获取就是要解决如何使专家系统获得高质量的知识。

知识获取通常是由知识工程师与专家系统中的知识获取模块共同完成的，知识工程师负责通过领域专家抽取知识，并用适当的知识表示方式把知识表示出来。专家系统中的知识获取模块负责把知识转换为计算机可存储的内部形式，把它们存入知识库。在知识存储的过程中，要对知识进行一致性、完整性的检测。

5.5.1　知识获取的任务与方式

知识获取的基本任务是为专家系统获取知识，建立起健全、完善、有效的知识库，以满足求解领域问题的需要。按知识获取的自动化程度划分，知识获取可分为非自动知识获取和自动知识获取两种方式。

1. 知识获取的任务

知识获取需要做以下几项工作。

(1) 抽取知识

抽取知识是指把蕴含于知识源中的知识经过识别、理解、筛选、归纳等抽取出来，以用于建

立知识库。

知识的主要来源是领域专家及相关的专业技术文献，但知识库所需知识并不是以某种现成的形式存在于这些知识源中，为了从中得到所需的知识需要做大量的工作。领域专家虽然可以自如地处理领域内的各种困难问题，但不一定能给出这些实例处理的一般规则，有的经验性知识甚至只可意会而不能言传。另外，领域专家一般都不熟悉专家系统的有关技术，不知道应该提供一些什么知识，以及用什么样的形式来表达这些知识。由于不能强求领域专家按专家系统的要求来提供知识，因此，为了从领域专家得到有用的知识，需要反复多次地与领域专家交谈，并且有目的地引导交谈的内容，然后通过分析、综合，去粗取精、去伪存真，归纳出可供建立知识库的知识。

知识的另一来源是系统自身的运行实践，这就需要从实践中学习、总结出新的知识。一般来说，一个系统初步建成后是很难做到完美无缺的，通过运行才会发现知识不够健全，需要补充新的知识。此时除了请领域专家提供进一步的知识外，还可由系统根据运行经验从已有的知识或实例中演绎、归纳出新知识，补充到知识库中去，这就要求系统自身具有一定的“学习”能力，这将对知识获取提出更高的要求。

(2) 知识转换

知识转换是指把知识由一种表示形式转换为另一种表示形式。

人类专家或科技文献中的知识通常是用自然语言、图形、表格等形式表示的，而知识库中的知识是用计算机能够识别、运用的形式表示的，两者之间有较大的差别。为了把从专家及有关文献中抽取出来的知识送入知识库供求解问题使用，需要进行知识表示形式的转换。知识转换一般分两步进行：第一步是把从专家及文献资料抽取的知识转换为某种知识表示模式，如产生式规则、框架等；第二步是把该模式表示的知识转换为系统可直接利用的内部形式。

(3) 知识输入

知识输入是指把用适当的知识表示模式表示的知识经过编辑、编译送入知识库的过程。目前，知识的输入一般通过两种途径实现：一种是利用计算机系统提供的编辑软件；另一种是利用专门编制的知识编辑系统，称之为知识编辑器。前一种的优点是简单，可直接拿来使用，减少了编制专门的知识编辑器的工作；后一种的优点是专门的知识编辑器可根据实际需要实现相应的功能，使其具有更强的针对性和适用性，更加符合知识输入的需要。

(4) 知识检测

知识库的建立是通过对知识进行抽取、转换、输入等环节实现的，任何环节上的失误都会造成知识错误，直接影响专家系统的性能，因此，必须对知识库中的知识进行检测，以便尽早发现并纠正错误。另外，经过抽取转换后的知识可能存在知识的不一致和不完整等问题，也需要通过知识检测来发现是否有知识的不一致和不完整，并采取相应的修正措施，使专家系统的知识具有一致性和完整性。

2. 知识获取方式

(1) 非自动知识获取

在非自动知识获取方式中，知识获取分两步进行，首先由知识工程师从领域专家和有关技

术文献获取知识，然后由知识工程师用某种知识编辑软件输入到知识库中。

如前所述，领域专家一般不熟悉知识处理，不能强求他们把自己的知识按专家系统的要求进行知识抽取和转换。另外，专家系统的设计和建造者虽然熟悉专家系统的建造技术，却不掌握专家知识。因此，需要在这两者之间有一个中介专家，他既懂得如何与领域专家打交道，能从领域专家及有关文献中抽取专家系统所需的知识，又熟悉知识处理，能把获得的知识用合适的知识表示模式或语言表示出来，这样的中介专家称为知识工程师。实际上，知识工程师的工作大多是由专家系统的设计与建造者担任。知识工程师的主要任务是：

① 与领域专家进行交谈，阅读有关文献，获取专家系统所需要的原始知识。这是一件很费力费时的工作，知识工程师往往需要从头学习一门新的专业知识。

② 对获得的原始知识进行分析、整理、归纳，形成用自然语言表述的知识条款，然后交领域专家审查。知识工程师与领域专家可能需要进行多次交流，直至有关的知识条款能完全确定下来。

③ 把最后确定的知识条款用知识表示语言表示出来，通过知识编辑器进行编辑输入。

(2) 自动知识获取

自动知识获取是指系统自身具有获取知识的能力，它不仅可以从专家提供的实例信息中“学习”到专家系统所需的知识，而且还能从系统自身的运行实践中总结、归纳出新的知识，发现知识中可能存在的错误，不断自我完善，建立起性能优良、知识完善的知识库。为达到这一目的，自动知识获取至少应具备以下能力：

① 理解、分析、归纳的能力。领域专家提供的知识通常是处理具体问题的实例，不能直接用于知识库。为了把这些实例转变为知识库中的知识，必须对实例集进行分析、归纳、综合，从中抽取专家系统所需的知识送入知识库。在非自动知识获取方式中，这一工作是由知识工程师完成的，而在自动知识获取方式中则由系统自动完成。

② 从运行实践中学习的能力。在知识库初步建成投入使用后，随着应用的发展，知识库的不完备性就会逐渐暴露出来。知识的自动获取系统应能在运行实践中学习，产生新知识，纠正可能存在的错误，不断地对知识库进行更新和完善。

总之，在自动知识获取系统中，原来需要知识工程师做的工作都由系统来完成，并且还应做更多的工作。自动知识获取是一种理想的知识获取方式，它的实现涉及人工智能的多个研究领域，例如，模式识别、自然语言理解、机器学习等，对硬件也有更高的要求。

5.5.2　知识的检测与求精

知识库中知识的一致性、完整性是影响专家系统性能的重要因素。

知识库的建立过程是知识经过一系列变换后存入计算机系统的过程，在这个过程中存在各种因素会导致知识不健全。例如：

① 领域专家提供的知识中存在某些不一致、不完整、甚至错误的知识。由于专家系统是以专家知识为基础的，因而专家知识中的任何不一致、不完整必然影响知识库的知识一致性与完整性。

② 知识工程师未能准确、全面地理解领域专家的意图，使得所抽取的知识条款隐含着种种错误，影响到知识的一致性和完整性。

③ 对知识库中的知识进行增、删、改时没有充分考虑到可能产生的影响，以致在对知识库进行这些知识更新操作后使知识库出现了知识不一致或不完整的情况。由于知识之间存在着复杂的联系，因此，对知识的更新操作可能产生意想不到的后果。

由于这些原因，知识库中可能出现各种问题，主要表现为知识冗余、矛盾、从属、环路、不完整等方面。下面以产生式表示模式为例说明各种问题的表现形式及处理方法。

1. 知识冗余

知识冗余是指知识库中存在多余的知识或者存在多余的约束条件。知识冗余有以下三种表现形式。

(1) 等价规则

当两条产生式规则在相同条件下有相同的结论时，称它们为等价规则。例如，若有如下产生式规则：

R_1： if $P \wedge Q$ then R

R_2： if $Q \wedge P$ then R

则它们是等价的。此时，R_1与 R_2中有一条规则是多余的，可从知识库中删去。

(2) 冗余规则链

如果两条规则链中第一条规则的条件相同，且最后一条规则的结论等价，则称此两条规则链之间存在冗余。例如，若有如下产生式规则：

R_1： if P then Q

R_2： if Q then R

R_3： if P then S

R_4： if S then R

其中，如果 Q 只是在 R_2的条件中出现，且不再在其他规则的条件中出现，则 R_1和 R_2都是冗余规则，可从知识库中删去。同理，如果 S 只是在 R_4的条件中出现，且不再在其他规则的条件中出现，则 R_3和 R_4都是冗余规则，可从知识库中删去。如果除 R_2与 R_4外，Q 与 S 同时都不再在别的规则的条件中出现，则 $R_1 \sim R_4$都是冗余规则，均可从知识库中删去，但应补充如下一条规则：

if P then R

否则，将破坏知识库的完整性。

(3) 冗余条件

如果两条规则有相同的结论，但一条规则中的某个子条件在另一条规则的条件中被否定，而其他子条件保持一致，则称这两条规则有多余的条件。例如，若有如下两条产生式规则：

R_1： if $P \wedge Q$ then R

R_2： if $P(\wedge \neg Q)$ then R

则子条件 Q 与 $\neg Q$ 都是多余的，此时需要删去这两条规则，并增加如下一条规则：

if　P　then　R

2. 知识矛盾

如果两条规则或规则链在相同条件下得到的结论是互斥的；或者它们虽有相同的结论，但规则强度不同，则称它们是矛盾的。例如，若有如下两条产生式规则：

R_1：　if　P　then　Q_1

R_2：　if　P　then　Q_2

且 $Q_1=\neg Q_2$，则 R_1 与 R_2 是矛盾的。

再如，若有如下产生式规则：

R_1：　if　P　then　Q

R_2：　if　Q　then　R

R_3：　if　R　then　S_1

R_4：　if　P　then　T

R_5：　if　T　then　S_2

其中，R_1、R_2、和 R_3 是一条规则链，R_4 和 R_5 是另一条规则链，它们有相同的初始条件，即条件 P。若 $S_1=\neg S_2$，则这两条规则链是矛盾的。对于矛盾规则或矛盾规则链，不能共处于同一个知识库中，必须从中舍弃一个。至于舍弃哪一个，需要征询领域专家意见。

又如，若有如下两条产生式规则：

R_1：　if　P　then　Q　(CF_1)

R_2：　if　P　then　Q　(CF_2)

R_1 与 R_2 的条件及结论都分别相同，但有不同的规则强度$(CF_1\neq CF_2)$，则称它们是矛盾的。它们也不能共处于同一个知识库中，需根据领域专家的意见保留其中一个。

3. 知识从属

如果规则 R_1 与 R_2 有相同的结论，但 R_1 比 R_2 有更多的条件，则称 R_1 是 R_2 的从属规则。例如，若有如下两条产生式规则：

R_1：　if　$P\wedge Q$　then　R

R_2：　if　P　then　R

则 R_1 是 R_2 的从属规则。

当出现从属规则时，需征询领域专家的意见，分别进行以下几种处理：

① 若领域专家认为 R_1 比 R_2 能更准确地描述实际情况，则保留 R_1，舍弃 R_2。

② 若领域专家认为 R_1 中的 Q 是可有可无的，则舍弃 R_1，保留 R_2。

③ 若领域专家认为这两条规则分别适用于不同的情况，则把它们都保留，但需把适用条件增设到相应规则的条件部分中去。

4. 知识环路

当一组规则形成一条循环链时，则称它们构成了一个环路。例如，若有如下一组产生式规

则：

R_1： if P then Q

R_2： if Q then R

R_3： if R then S

R_4： if S then P

则这 4 条规则构成一个环路。对环路无论先执行哪一条规则，最终又回到出发点。环路可能使推理陷入死循环，需要引起足够的重视。

当知识库中出现环路时，应征询领域专家的意见，修改或舍弃其中的一条规则，破坏形成环路的条件。

5. 不可达知识

不可达知识是指知识的约束条件永远得不到满足的知识，这种知识在推理过程中不会被激活，不会出现在任何推理链中，应被舍弃。

6. 知识不完整

知识不完整是指知识库中的知识不完全，不能满足预先定义的约束条件，即当存在应该推出某一结论的条件时，却推不出这一结论，不能形成产生这一结论的推理链；或者虽能推出结论，但却是错误的。

当知识库的知识不完整时，需要通过知识求精，不断改进、完善，使其能满足问题求解的需要。

5.5.3 知识的检测方法

为了保证知识库的正确性，需要做好知识的检测。知识检测分为静态检测和动态检测。静态检测是指在知识输入之前由领域专家及知识工程师所做的检查工作。动态检测是指在知识输入过程中及对知识库进行增、删、改时由系统所进行的检查。在系统运行过程中出现错误时也需要对知识库进行动态检测。这里，讨论动态检测方法。

对知识冗余、矛盾等的检测实际上是通过对知识的相应部分进行比较来实现的。例如，对两条产生式规则的条件部分进行比较，看其是否等价，再对它们的结论部分进行比较，看其是否一致等。因此，检测中的基本环节是检查两个逻辑表达式的等价性。

1. 逻辑表达式等价性的检测

产生式规则的条件部分和结论部分都可以分别表示为一个合取式，两个合取式的等价性的检测就是逐一比较它们的合取项是否一致。设 A 和 B 分别是两个合取式，$L(A)$ 与 $L(B)$ 分别表示 A 与 B 的长度，即 A 与 B 分别包含的合取项的个数，A_i 表示 A 的第 i 个合取项，B_j 表 B 的第 j 个合取项。若 $L(A)\neq L(B)$，则 A 与 B 不等价。若 $L(A)=L(B)$，则把 B 的所有的合取项逐一地与 A 的所有合取项进行比较，每当 B 的某一个合取项 B_j 与 A 的某个合取项 A_i

等价时，就为 B_j 做一标记，不再把 B_j 与 A 的其他合取项进行比较。如果 B 的所有合取项都被做上标记，则 B 与 A 等价；否则，B 与 A 不等价。

2. 冗余的检测

以逻辑表达式等价性检测方法为基础，就可以进行冗余的检测。

(1) 等价规则的检测

产生式规则的条件部分和结论部分都是合取式，只要对两条规则的条件部分及结论部分分别检查等价性就可得知这两条规则是否等价。

(2) 冗余规则链的检测

为了发现冗余规则链，首先应该检查两条规则链的第一条规则的条件是否等价；若有等价条件，则检查这两条规则链的最后一条规则的结论是否等价；若又有等价结论，则可能有冗余规则链。

冗余规则链的检测可分为以下三步进行。

第一步：建立两个二维表，一个表称为 IF-IF 表，用以存放不同规则条件部分的等价比较结果；另一个表称为 THEN-THEN 表，用以存放不同规则结论部分的等价比较结果。

第二步：根据 IF-IF 表，取出条件部分等价的两条规则，并分别建立它们的推理链。由这两条推理链的最后一条规则的结论部分查找 THEN-THEN 表，若两个结论部分是不等价的，则这两条推理链不是冗余的；若两个结论部分是等价的，则这两条推理链可能是冗余推理链，由第三步进一步确定。

第三步：把一条推理链中的所有规则的结论部分与其他所有规则的条件部分进行等价比较，若都不等价，则这条推理链是一条多余的推理链，可将这条推理链的所有规则舍弃。若两条推理链都是多余推理链，则可将这两条推理链的所有规则都舍弃，但应补充一条新的规则，新规则的条件部分是推理链第一条规则的条件部分，新规则的结论部分是推理链最后一条规则的结论部分。

(3) 冗余条件的检测

为了发现冗余条件，首先应检查两条规则的结论是否等价，这可从 THEN-THEN 表得到。当两条规则的结论等价而条件部分不等价时，可把其中一条规则条件部分的各个合取项逐一变为否定，并逐次检测这两条规则条件部分的等价性。若等价，则刚才被否定的子条件及在另一条规则条件部分中与之对应的那个子条件都是冗余条件。

3. 矛盾规则及矛盾规则链的检测

首先根据 IF-IF 表找出两个条件部分等价的规则，然后将其中一条规则的结论部分变为否定后再与另一条规则的结论部分进行等价比较，若等价，则这两条规则是矛盾的。

对于由规则强度不同引起的矛盾，先检查两条规则是否等价，若等价，再检查它们的规则强度是否相同，若不同，则这两条规则是矛盾的。

矛盾规则链的检测与冗余规则链的检测方法类似。首先根据 IF-IF 表找出两个条件部分等价的规则，并分别建立它们的推理链；然后遍历这两条推理链，若发现这两条推理链有一对

规则的结论部分是矛盾的，则从这一对规则分别沿各自推理链回溯至第一条规则的两条子推理链也是矛盾的。

4. 从属规则的检测

为了发现从属规则，首先要检查两条规则的结论是否等价，这可从 THEN-THEN 表得到，若两条规则的结论等价，再检查一条规则的条件部分是否是另一条规则条件部分的一部分，若是，则表明后者比前者要求更多的约束条件，故而后者是前者的从属规则。

5. 环路的检测

为了检测知识之间是否存在规则链环路，需要找到这样的规则，即该规则的结论与其他规则的条件等价，这样的规则可能会是一个环路中的一条规则。为此，可建立一个名为IF-THEN的二维表，表中存放每条规则的条件部分与其他规则的结论部分等价比较的结果。检测时，首先根据 IF-THEN 表找出一条结论与其他规则的条件等价的规则，然后从这条规则开始，根据 IF-THEN 表沿着规则链进行查找，找出下一条结论与其他规则条件等价的规则，直到出现如下两种情况之一时为止：

① 在规则链中找到了一条规则，它的结论与规则链中已有的某条规则的条件等价，这说明这个规则链有一个环路。

② 沿规则链找不到一条结论与规则链中已有规则的条件等价的规则，且规则链结束，这说明这个规则链没有形成环路。

若对每条规则链都进行环路检测，则可把知识库中的环路都找出来，并进行相应的处理。

5.6 专家系统工具

知识工程师建造一个专家系统时，使用专家系统工具可以极大地简化建立专家系统的工作，减少建造的工作量，提高建造的专家系统的性能。专家系统工具的出现，是知识工程的重大进展，它帮助知识工程师完成了许多工作，能大大缩短专家系统的研制周期。对于各领域的专家来说，工具比一般的程序设计语言更易学习和使用，因此，在专家系统的开发中应更多地使用专家系统工具来建造一个专家系统。

专家系统工具按其功能主要分为两类，一类是用于生成专家系统的工具，称为生成工具；另一类用于改善专家系统性能的工具，称为辅助工具。

1. 系统生成工具

系统生成工具主要帮助知识工程师构造专家系统中的推理机和知识库结构。按照生成工具的本身特征又可分为以下 4 类。

(1) 程序设计语言

程序设计语言是开发专家系统的最基本的工具。典型的程序设计语言是 LISP 语言和

PROLOG 语言，用这两种人工智能语言能方便地表示知识和设计各种推理机。具有面向对象风格的语言 C++，以及传统语言 C 和 PASCAL 等也是构造专家系统的常用语言。

(2) 骨架系统

骨架系统是把一个成功的专家系统删去其特定领域知识而留下的系统框架。例如，系统生成工具 EMYCIN 就是删去医疗诊断专家系统 MYCIN 的医疗诊断知识而获得的骨架系统。骨架系统继承了原专家系统中行之有效的知识表示方式、推理机和知识库结构及全部辅助工具。因此，利用骨架系统建造专家系统时，只要把特定领域的知识按照该骨架系统的知识表示方式输入到知识库中，就构成了一个特定领域的专家系统。

骨架系统之所以能快速方便地构造一个专家系统，是由于专家系统的推理机与知识库是分离的，只要知识库的知识表示方式和知识库的结构确定后，推理机就随之确定了。

由于骨架系统生成的专家系统完全继承了原系统的知识表示方式和知识库结构及推理机等，因此，限制了专家系统设计者的设计选择。另一方面，选择某个骨架系统生成一个特定领域的专家系统时，若生成的专家系统与骨架系统的原系统是属于同一类问题领域，就更易生成，其效果也更好。或者说，骨架系统作为生成工具，缺乏通用性和灵活性，每一个骨架系统只适合于某一类特定的问题领域。

(3) 知识工程语言

知识工程语言是专门用于构造和调试专家系统的通用程序设计语言，它能够处理不同的问题领域和问题类型，提供各种控制结构。用知识工程语言设计推理机和知识库，比用一般的人工智能程序设计语言(如 LISP 或 PROLOG 等)更为方便。由于知识工程语言并不与特定的结构和方法紧密联系，因此比骨架系统更为灵活和通用。

常见的知识工程语言工具如下。

① OPS(Official Production System)是著名的通用型语言工具，从 OPS1 到 OPS5、OPS5+、OPS5e、OPS83，已经历了许多版本的修改。OPS 是用 LISP 语言实现的，主要采用基于规则的知识表示和正向推理。用 OPS83 能很快地编写出紧凑的产生式专家系统程序，其数据类型特征、用户定义的函数及过程均与当前的过程型程序设计语言 PASCAL 和 C 语言类同。OPS 已广泛用于开发计算机设计、模式识别、图像处理、故障诊断、实时控制等方面的专家系统。

② KEE(Knowledge Engineering Environment)也是著名的通用语言工具，是 Intellicorp 公司于 1984 年开发的。它把基于框架、基于规则、面向过程和面向对象的方法结合在一起，提供了一种多范例程序设计环境，适于设计范围较大的实用专家系统。KEE 已用来建造了卫星失灵诊断、加热冷却管理、金融保险分析、工厂控制模拟、遗传工程研究、光导纤维制造及原子能发电站紧急状态报警等方面的数以千计规模不等的实用专家系统。

③ ROSIE(Rule-Oriented System for Implementing Expertise)是第一个基于专家系统的通用程序设计系统，主要特点是基于规则和面向英语解释，具有英语语法。ROSIE 是 Rand 公司于 1981 年开发的，用 InterLISP 和 C 语言实现的。它已用来开发了法律决策、战况分析、空战规划等专家系统。1982 年，Rand 公司推出了它的另一个专家系统工具 ROSS，主要特点是基于规则和面向对象。

④ CLIPS(C Language Integrated Production System)是 20 世纪 80 年代中期以来使用广泛的通用语言工具,其特点是具有产生式系统的实用特征,并集成了 C 语言的基本语言成分。把人工智能与传统语言相结合,是当代通用型语言工具的重要发展方向。CLIPS 是美国航空航天管理局(NASA)于 1985 年推出的,它可以在许多机器上运行,已广泛用于开发各类专家系统和知识处理系统。

⑤ ART(Automatic Reasoning Tool)是美国 Inference 公司于 1984 年推出的一种通用语言工具,它是采用 LISP 语言和 C 语言实现的。主要特点是基于规则和基于框架的知识表示、面向过程的正向与反向推理及双向推理。

(4) 专家系统开发环境

专家系统开发环境是以一种或多种工具和方法为核心,加上与之配套的各种辅助工具和界面环境的、完整的集成系统。近几年来,专家系统的规模越来越大,出现了知识数量达数千条乃至数万条规则,知识层次包括元知识、经验性知识、原理性知识和常识性知识等几个层次的专家系统。因此,超大规模知识库的组织和管理变得突出起来,不同的知识表示系统之间及人工智能技术与数据库等传统主流技术之间的系统集成技术引起了人们的高度重视。把数据库、逻辑推理、模块化技术、面向对象程序设计方法、支持智能体通信及多媒体用户界面等先进技术集成到一个智能系统开发工具中,已成为专家系统和智能系统开发工具的主要发展方向。

目前,有些知识工程语言系统已经发展成这样的集成系统,集成系统中有一组预先定义的称为组件的程序模块,每个组件实现一种人工智能技术。这种环境可提供多种类型的推理机制和多种知识表示方法,帮助专家系统的建造者选择结构、设计规则语言和使用各种组件,使之成为一个完整的专家系统。

美国 Inference 公司于 1993 年推出的 ART * Enterprise 是一种集成化的智能应用软件开发工具,它具有面向对象、多种数据库管理、基于事例的推理(Case-Based Reasoning)和多媒体用户界面(GUI with Multimedia)等特点。对于金融业、汽车工业、电子工业、钢铁工业、航空航天部门、通信部门、计算机设计与制造业等的信息咨询与决策、故障诊断、设计规划已有广泛的应用。

2. 系统辅助工具

系统辅助工具主要用于帮助建造高质量的知识库和调试专家系统。

知识获取工具和知识库管理与维护工具是最重要的辅助工具。知识获取工具有自动知识获取工具,知识库编辑工具、面向问题求解方法的知识获取工具、面向特定知识生成技术的知识获取工具、面向特定问题领域的知识获取工具及基于特定语言的知识获取工具等类型。其中,自动知识获取工具采用机器学习方法来获取知识,例如,EXPERT-EASY 能通过归纳学习自动生成问题领域的求解规则。知识库编辑工具能把专家领域知识加工、编辑到知识库中,这样的编辑工具有 TEIRESIAS 编辑器。知识库管理与维护工具能检查输入知识的一些常见错误,自动维护知识库中知识的一致性和完备性。这些工具不仅能帮助知识工程师加快建造专家系统的速度,还能保证和提高知识库的质量,调试和改进专家系统。

习　题　五

5.1　何谓专家系统？按照专家系统处理的问题类型，专家系统可分为哪几种类型？

5.2　简述专家系统的一般特点。

5.3　画出专家系统一般的组成框图，说明各组成部分的主要功能。

5.4　根据专家系统开发过程的瀑布模型，说明开发过程各阶段应完成的主要工作内容。

5.5　简述 LISP 语言的基本特点。

5.6　用 LISP 的非结构迭代方式定义一个计算前 n 个自然数的和的函数 sumup(n)。

5.7　用 LISP 的结构迭代方式定义一个计算 m^n 的函数 expt(m,n)，其中 m 和 n 都是自然数。

5.8　用 prog 函数定义函数 our-member，该函数的功能与函数 member 相同。

5.9　简述产生式系统的知识库与推理机能够相互独立的原因。

5.10　已知规则库 rules=(R_1,R_2,R_3,R_4,R_5)中的各条规则分别为

R_1：　$F_1 \wedge F_2 \rightarrow M_1$

R_2：　$F_3 \wedge F_4 \rightarrow M_2$

R_3：　$F_3 \wedge F_5 \rightarrow M_3$

R_4：　$M_1 \wedge M_2 \rightarrow H_1$

R_5：　$M_1 \wedge M_3 \rightarrow H_2$

当前黑板中的初始事实集为 facts=(F_1,F_2,F_3,F_5)，应用正向推理机 deduce 进行推理。

(1) 请说明正向推理机 deduce 的终止条件。

(2) 依序给出每一个推理步结束时，facts 中的内容。

5.11　应用正向推理机 deduce 求解例 3.3 的最短路径问题。假设已对 deduce 中的功能进行了如下扩展：①计算节点的代价，并从黑板中查找代价最小的节点作为当前节点，根据当前节点状态 S_i 寻找可用规则；②能自动实现将黑板中的当前节点状态 S_i 与目标状态 S_g 进行比较，若 $S_i = S_g$，则推理成功终止，若 $S_i \neq S_g$，且在规则集中找不到可用规则，则推理失败终止。

(1) 将例 3.3 中的 12 个算符表示成 12 条产生式规则，组成规则库 rules。

(2) 若黑板 facts 的初始内容为 facts=($S_0=A$，$S_g=A\$E$)，依序给出每一个推理步结束时，facts 中的内容。

5.12　应用正向推理机 deduce 求解例 3.6 的最短路径问题。假设已对 deduce 中的功能进行了如下扩展：①计算节点的代价，并从黑板中查找代价最小的节点作为当前节点，根据当前节点状态 S_i 寻找可用规则；②能自动实现将黑板中的当前节点状态 S_i 与目标状态 S_g 进行比较，若 $S_i = S_g$，则推理成功终止，若 $S_i \neq S_g$，且在规则集中找不到可用规则，则推理失败终止。

(1) 将例 3.6 中的 18 个算符表示成 18 条产生式规则，组成规则库 rules。

(2) 若黑板 facts 的初始内容为 facts =($S_0=A$，$S_g=A\$F$)，依序给出每一个推理步结束时，facts 中的内容。

5.13　应用正向推理机 deduce 求解例 3.7 的推销员旅行问题。

(1) 需要对推理机程序 deduce 进行哪些主要功能扩展。

(2) 将例 3.7 中的 20 个算符表示成 20 条产生式规则，组成规则库 rules。

(3) 若黑板 facts 的初始内容为 facts=($S_0=A$，$S_g=Ax_1x_2x_3x_4A$)，依序给出每一个推理步结束时，facts 中的内容。

5.14　应用正向推理机 deduce 求解例 3.9 的三阶梵塔问题。

(1) 需要对推理机程序 deduce 进行哪些主要功能扩展。

(2) 将例 3.9 中的 9 个算符表示成 9 条产生式规则,组成规则库 rules。

(3) 若黑板 facts 的初始内容为 facts=(S_0=(CBA,∅,∅),S_g=(∅,∅,CBA)),依序给出每一个推理步结束时,facts 中的内容。

5.15 对如例 5.1 的动物识别专家系统规则库,若最初赋给假设表 hypo 的值为 H_4,即 hypo=(H_4),且调用反向推理机 diagnose 进行反向推理。若用户在反向推理过程中只能确认事实 F_{15}、F_7、F_2为真,那么,推理机能否推理出 H_4为真的结论?若能够,请画出 H_4的与/或树。若用户只能确认事实 F_{15}、F_8、F_7为真,那么,能否推出 H_4为真的结论?为什么?

5.16 简述专家系统解释机制的几种实现方法。

5.17 简述知识获取的方式及一般应完成的工作内容。

5.18 在规则库中,何谓冗余规则链?何谓矛盾规则链?何谓循环规则链?试分别举例说明。

5.19 如何检测和处理规则库中的冗余规则链?

5.20 如何检测和处理规则库中的循环规则链?

5.21 专家系统的生成工具可分为哪几类?试比较它们的特点。

第 6 章

不确定推理方法

在各种实际应用领域中，精确的或确定的知识并不多见，大量的知识是不精确的或不确定的，需要采用不确定推理或称为不精确推理对不确定知识进行处理。可以说，不确定性(Uncertainty)是智能问题的本质特征，因此，智能系统的能力更主要反映在求解不确定性问题的能力上。

6.1 知识的不确定性

有时，人们难于用精确的概念来描述事物，例如，可以按身高把人分为“高”与“不高”两类，但是，难以给出精确的度量来区分“高”与“不高”。用精确的推理去处理不精确的事实，有时会得出荒谬的结论。例如，“秃子”是一个模糊的概念，人们难以用有多少根头发来区分秃子或不是秃子。但是，可以有一条精确的推理规则：比秃子多一根头发的人仍是秃子。如果反复地用这条精确的规则处理“秃子”这个不精确的概念，就会得出一个荒谬的结论：比秃子多 100 万根头发的人仍是秃子。这就是著名的“秃子悖论”。显然，对于不确定的知识要用不确定推理来处理。

专家系统中的不确定性表现在三个方面，第一是证据或事实的不确定性，第二是规则的不确定性，第三是推理的不确定性。

1. 证据的不确定性

证据或事实的不确定性主要反映在以下方面。

(1) 证据的歧义性

证据的歧义性是指证据可具有多种含义明显不同的解释，如果离开证据所在环境和证据的上下文，往往难以确定其真正的含义。例如，“同意总理开会”，一般理解为某人赞同总理去开会，但也可以理解为某人陪同意大利总理去开会。又如，“I saw a man with a telescope”既可理解为我看见一个带着望远镜的人，也可理解为我用望远镜看见一个人。

歧义的消除是机器翻译和自然语言理解着重要解决的问题。在专家系统中，证据的歧义性要在知识获取阶段中消除掉，通过与专家交谈来了解可能引起歧义的证据的确切含义，并用不会引起歧义的语言来表达证据。

(2) 证据的不完全性

证据的不完全性有两方面的含义，一是证据尚未收集完全；二是证据的特征值不完全。任何一个专业领域的知识都是发展变化和不断积累的，因此，领域专家的大部分决策都是在知识

不完全的情况下作出的。例如，在经济预测中，市场信息瞬息万变，要获得全部完整的信息才作出决策几乎是不可能的。

(3) 证据的不精确性

证据的不精确性是指证据表示的值与证据的真实值之间存在一定的差异。例如，某人的身高大约是 1.70 米，这里“大约是 1.70 米”就是不精确的表示。这种证据的不精确性与因测量误差引起的数据不精确性不是同一概念，这种证据的不精确表示是智能问题的本质特征，这正是专家系统不确定推理的一个重要的研究对象。

(4) 证据的模糊性

证据的模糊性是指证据的取值范围的边界是模糊的、不明确的。例如，“年轻”就是一个模糊的概念，很难具体指明哪一个年龄层次的人为年轻人，其边界是不明确的。证据的模糊性也是智能问题的本质特征。

(5) 证据的可信性

证据的可信性是指专家主观上对证据的可靠性的信任程度。如果证据不是完全可信任的，那么，在进行决策时，对这样的证据要打一定的折扣并经过处理后才能使用。

(6) 证据的随机性

证据的随机性是指证据是随机出现的。例如，某张牌分发给某人是随机的。

证据的上述不确定性没有穷尽客观世界中不确定性所具有的丰富内涵，它们相互之间的区分也不是绝对的。

2. 规则的不确定性

专家系统的知识库中包含大量的启发式知识，这些知识来源于领域专家处理问题的知识和经验。既然领域专家的知识和经验是不确定的，因而，知识库中的规则也就必然具有不确定性。

产生式规则的不确定性主要有以下几个方面。

(1) 规则前件的条件的不确定性

例如，有一条产生式规则：“如果患者发高烧且常流清鼻涕，则患者感冒”。这条规则的前件有两个条件，即“发高烧”和“常流清鼻涕”，这两个条件都是模糊的概念，难以明确指明“发高烧”的体温值边界和发烧的时间值边界，也难以明确指明“清鼻涕”的鼻涕浓度值边界和“常流”的时间值边界。

(2) 规则前件的证据组合的不确定性

例如，有一位患者的体温一直都是 40℃，流清鼻涕，但并不是常流清鼻涕，那么，这两个证据组合起来在多大程度上符合规则前件的条件，也包含着不确定性。

(3) 规则本身的不确定性

上述产生式规则是从发高烧且常流清鼻涕引出感冒的结论，实际上，判断一位患者是否患有感冒还会有其他的证据；另外，发高烧和常流清鼻涕也会是患有其他疾病的证据。因此，领域专家对这一条规则持有某种信任程度。也就是说，每一条规则并不都具有 100％的信任程度，这就是规则的不确定性，或称为规则强度。

(4) 规则结论的不确定性

由于规则的前件包含有各种不确定性因素，运用不确定的规则，导出的结论也就不可避免地是不确定的。由于上述产生式规则引出的结论应是“患者可能感冒”，这是一个不确定的结论，但更符合领域专家的判断。

3. 推理的不确定性

推理的不确定性是指：由于证据的不确定性和规则的不确定性在推理过程中的动态积累和传播，从而导致推理结论的不确定性。因此，需要采用某种不确定性的测度，并在推理过程中传递和计算这种不确定性的测度，最终得到结论的不确定性测度。

在不确定推理中，不确定性测度的计算有以下三种基本的计算模式。

(1) 证据组合的不确定性测度计算模式

已知证据 $e_1,e_2,\cdots,e_n$ 的不确定性测度为 $\mathrm{MU}_1,\mathrm{MU}_2,\cdots,\mathrm{MU}_n$，求出逻辑组合的不确定性测度 MU。

证据的逻辑组合有三种基本形式。因此，证据逻辑组合的不确定性测度有以下三类。

① 证据的合取组合的不确定性测度。

$e_1,e_2,\cdots,e_n$ 的合取组合为 $e_1 \wedge e_2 \wedge \cdots \wedge e_n$，$n$ 个证据合取组合的不确定性测度表示为

$$\mathrm{MU}=f(\mathrm{MU}_1,\mathrm{MU}_2,\cdots,\mathrm{MU}_n)$$

② 证据的析取组合的不确定性测度。

$e_1,e_2,\cdots,e_n$ 的析取组合为 $e_1 \vee e_2 \vee \cdots \vee e_n$，$n$ 个证据析取组合的不确定性测度表示为

$$\mathrm{MU}=g(\mathrm{MU}_1,\mathrm{MU}_2,\cdots,\mathrm{MU}_n)$$

③ 证据的否定的不确定性测度。

证据 e_i 的否定为 $\bar{e}_i$，证据 e_i 的否定的不确定性测度表示为

$$\mathrm{MU}=q(\mathrm{MU}_i)$$

更复杂的证据逻辑组合都可由上述三种基本组合来表示，其不确定性测度可由上述三种基本组合的不确定性测度的计算得出。

(2) 并行规则的不确定性测度计算模式

已知有多条规则 if e_i then h 有相同的结论 h，各条规则的不确定性测度为 MU_i，$i=1,2,\cdots,n$。若 n 条规则都被满足，那么，结论 h 的不确定性测度表示为

$$\mathrm{MU}=p(\mathrm{MU}_1,\mathrm{MU}_2,\cdots,\mathrm{MU}_n)$$

这一计算模式也称为并行法则。并行法则给出了推理过程中有多条路径导致同一结论的情况下，规则的不确定性导致结论的不确定性测度的计算模式。

(3) 顺序(串行)规则的不确定性测度计算模式

已知两条规则 if e then e' 和 if e' then h 的规则不确定性测度分别为 MU_1 和 MU_2，那么，规则 if e then h 的规则不确定性测度表示为

$$\mathrm{MU}=s(\mathrm{MU}_1,\mathrm{MU}_2)$$

这一计算模式也称为顺序法则。顺序法则给出了规则不确定性在推理链中传播的计算模式。

对于一个专家系统，一旦给定上述三种计算模式的不确定性测度计算方法，即给出 f、g、q、p、s 的具体计算方法，就可获得证据不同组合的不确定性测度值，并根据在推理过程中使用规则的情况，由并行法则和顺序法则最终得出结论的不确定性测度值。专家系统中的不确定性推理就是上述三种计算模式的组合。但是，在不同的专家系统中，不确定性测度的计算方法可以不同。根据不确定性测度计算方法的不同，不确定推理可以有基于概率理论的不确定性推理、基于可信度理论的不确定推理和基于模糊理论的不确定性推理等。

6.2 基于可信度的不确定推理方法

基于概率的不确定推理虽然具有概率论严密的理论依据，但是，它要求给出知识的概率，即使富有经验的领域专家也难以直接给出，因而其应用受到限制。人们对一个事物的认识，往往可根据经验判断这个事物的真伪程度。根据经验对一个事物或一件事情为真的相信程度称为可信度(Certainty Factor)。

6.2.1 可信度与可信度推理计算方法

基于可信度的不确定推理方法是肖特里菲(E. H. Shortliffe)等人在确定性理论(Theory of Confirmation)的基础上提出的一种不确定推理方法，并首先在医疗诊断专家系统 MYCIN 中得到成功的应用。这种方法直观，不确定性测度的计算也比较简便，因而在许多专家系统中得到有效的应用。

1. 信任度与不信任度

定义 6.1 信任度 MB (h,e)表示证据 e 出现时，对结论 h 成立的信任程度的增加量。不信任度 MD (h,e)表示证据 e 出现时，对结论 h 成立的不信任程度的增加量。MB(h,e)和 MD(h,e)的取值范围为$[0,1]$。它们形式化地定义为

$$\mathrm{MB}(h,e)=\begin{cases}1 & P(h)=1\\ \dfrac{\max(P(h\mid e),P(h))-P(h)}{1-P(h)} & P(h)\neq 1\end{cases} \tag{6-1}$$

$$\mathrm{MD}(h,e)=\begin{cases}1 & P(h)=0\\ \dfrac{\min(P(h\mid e),P(h))-P(h)}{-P(h)} & P(h)\neq 0\end{cases} \tag{6-2}$$

其中，$P(h)$为结论 h 成立的先验概率；$P(h|e)$为在证据 e 出现的条件下，结论 h 成立的条件概率。

显然，MB(h,e)和 MD(h,e)有下述性质。

性质 6.1（互斥律） 一个证据 e 不可能既支持又不支持某个结论 h，因此，有

如果 MB$(h,e)>0$，那么 MD$(h,e)=0$；

如果 MD$(h,e)>0$，那么 MB$(h,e)=0$。

性质 6.2　若 $P(h|e)>P(h)$，表明证据 e 的出现增加了对结论 h 成立的信任程度，但是，不改变对结论 h 成立的不信任程度。

若 $P(h|e)>P(h)$，由(6-1)式可见，有 $\mathrm{MB}(h,e)>0$；由(6-2)式可见，有 $\mathrm{MD}(h,e)=0$。

性质 6.3　若 $P(h|e)=P(h)$，表明证据 e 的出现不改变对结论 h 成立的信任程度，也不改变对结论 h 成立的不信任程度，即表明证据 e 与结论 h 之间相互独立。

若 $P(h|e)=P(h)$，由(6-1)式可见，有 $\mathrm{MB}(h,e)=0$；由(6-2)式可见，有 $\mathrm{MD}(h,e)=0$，即信任程度与不信任程度都不变。

性质 6.4　若 $P(h|e)<P(h)$，表明证据 e 的出现增加了对结论 h 成立的不信任程度，但是，不改变对结论 h 成立的信任程度。

若 $P(h|e)<P(h)$，由(6-1)式可见，有 $\mathrm{MB}(h,e)=0$；由(6-2)式可见，有 $\mathrm{MD}(h,e)>0$。

2. 可信度

在可信度不确定推理模型中，把信任度 MB 与不信任度 MD 组合成一个单一的不确定性测度，这就是可信度。

定义 6.2　可信度形式化地定义为

$$\mathrm{CF}(h,e)=\mathrm{MB}(h,e)-\mathrm{MD}(h,e) \tag{6-3}$$

由 $\mathrm{CF}(h,e)$、$\mathrm{MB}(h,e)$ 和 $\mathrm{MD}(h,e)$ 的定义(6-3)式、(6-1)式和(6-2)式以及 MB 与 MD 的互斥性质，可得出 $\mathrm{CF}(h,e)$ 的计算公式为

$$\mathrm{CF}(h,e)=\begin{cases}1 & \text{若 } P(h)=1\\ \dfrac{P(h\mid e)-P(h)}{1-P(h)} & \text{若 } P(h\mid e)>P(h)\\ 0 & \text{若 } P(h\mid e)=P(h)\\ \dfrac{P(h\mid e)-P(h)}{P(h)} & \text{若 } P(h\mid e)<P(h)\\ -1 & \text{若 } P(h)=0\end{cases} \tag{6-4}$$

由(6-4)式可直观地看出可信度 $\mathrm{CF}(h,e)$ 具有下述意义。

① 若 $\mathrm{CF}(h,e)>0$，则 $P(h|e)>P(h)$。这说明证据 e 的出现增加了结论 h 为真的概率，即增加了 h 为真的可信度，$\mathrm{CF}(h,e)$ 的值越大，增加 h 为真的可信度就越大。若 $\mathrm{CF}(h,e)=1$，则可推出 $P(h|e)=1$，即证据 e 的出现使 h 为真。

② 若 $\mathrm{CF}(h,e)<0$，则 $P(h|e)<P(h)$。这说明证据 e 的出现减少了结论 h 为真的概率，即增加了 h 为假的可信度，$\mathrm{CF}(h,e)$ 的值越小，增加 h 为假的可信度就越大。若 $\mathrm{CF}(h,e)=-1$，则可推出 $P(h|e)=0$，即证据 e 的出现使 h 为假。

③ 若 $\mathrm{CF}(h,e)=0$，则 $P(h|e)=P(h)$，这表示 h 与 e 独立，即证据 e 的出现对 h 没有影响。

当已知 $P(h)$ 和 $P(h|e)$ 时，通过上述计算公式就可求出 $\mathrm{CF}(h,e)$。但是，在实际应用中，获得 $P(h)$ 和 $P(h|e)$ 的值是比较困难的，$\mathrm{CF}(h,e)$ 的值反而比较容易通过领域专家直接给出。设定 $\mathrm{CF}(h,e)$ 的值的原则是：若相应证据 e 能增加结论 h 为真的可信度，则使 $\mathrm{CF}(h,e)>0$，证据 e 越是支持 h 为真，就使 $\mathrm{CF}(h,e)$ 的值越大；反之，使 $\mathrm{CF}(h,e)<0$，证据 e 越是支持 h 为假，就使 $\mathrm{CF}(h,e)$ 值越小；若证据 e 与 h 无关，则使 $\mathrm{CF}(h,e)=0$。领域专家根据拥有的专业知识

和实践经验，不难对领域知识给出可信度。用可信度表示不确定性知识的方法比较直观、简单，效果也比较好。

定义 6.3 若证据 e 的可信度为 $\mathrm{CF}(e)$，规则 $e \to h$ 的可信度（亦称为规则强度）为 $\mathrm{CF}(h,e)$，可得出结论 h 的可信度为

$$\mathrm{CF}(h) = \mathrm{CF}(h,e) \times \max(0,\mathrm{CF}(e)) \tag{6-5}$$

由(6-5)式可见：

① 若 $\mathrm{CF}(e) < 0$，即证据 e 在某种程度为假，表明可信度推理不考虑证据 e 为假时对结论 h 可信度的影响。

② 若 $\mathrm{CF}(e)=1$，即证据 e 肯定为真，则 $\mathrm{CF}(h)=\mathrm{CF}(h,e)$，即结论 h 可信度完全由规则可信度确定。

3. 三种基本不确定性测度的计算

CF 模型给出了不确定推理中证据组合的不确定性测度、并行规则的不确定性测度和顺序规则的不确定性测度这三种基本不确定性测度的计算方法。

(1) 证据组合的不确定计算

① 证据合取组合的不确定计算。

定义 6.4 若证据 e_1 的可信度为 $\mathrm{CF}(e_1)$，证据 e_2 的可信度为 $\mathrm{CF}(e_2)$，则证据 e_1 与证据 e_2 的合取组合 $e_1 \wedge e_2$ 的可信度为

$$\mathrm{CF}(e_1 \wedge e_2) = \min(\mathrm{CF}(e_1),\mathrm{CF}(e_2)) \tag{6-6}$$

同样给出信任度与不信任度的计算方法为

$$\mathrm{MB}(e_1 \wedge e_2) = \min(\mathrm{MB}(e_1),\mathrm{MB}(e_2))$$

$$\mathrm{MD}(e_1 \wedge e_2) = \max(\mathrm{MD}(e_1),\mathrm{MD}(e_2))$$

② 证据析取组合的不确定计算。

定义 6.5 若证据 e_1 的可信度为 $\mathrm{CF}(e_1)$，证据 e_2 的可信度为 $\mathrm{CF}(e_2)$，则证据 e_1 与证据 e_2 的析取组合 $e_1 \vee e_2$ 的可信度为

$$\mathrm{CF}(e_1 \vee e_2) = \max(\mathrm{CF}(e_1),\mathrm{CF}(e_2)) \tag{6-7}$$

同样给出信任度与不信任度的计算方法为

$$\mathrm{MB}(e_1 \vee e_2) = \max(\mathrm{MB}(e_1),\mathrm{MB}(e_2))$$

$$\mathrm{MD}(e_1 \vee e_2) = \min(\mathrm{MD}(e_1),\mathrm{MD}(e_2))$$

(2) 并行规则的不确定计算(并行法则)

定义 6.6 若证据 e_1 的出现使结论 h 成立的可信度为 $\mathrm{CF}(h,e_1)$，证据 e_2 的出现使结论 h 成立的可信度为 $\mathrm{CF}(h,e_2)$，则证据 e_1 与 e_2 同时出现使结论 h 成立的可信度 $\mathrm{CF}(h,e_1e_2)$ 为

$$\mathrm{CF}(h,e_1e_2) = \begin{cases} \mathrm{CF}(h,e_1)+\mathrm{CF}(h,e_2)-\mathrm{CF}(h,e_1)\mathrm{CF}(h,e_2) & \text{若 } \mathrm{CF}(h,e_1) \geqslant 0, \mathrm{CF}(h,e_2) \geqslant 0 \\ \dfrac{\mathrm{CF}(h,e_1)+\mathrm{CF}(h,e_2)}{1-\min(|\mathrm{CF}(h,e_1)|,|\mathrm{CF}(h,e_2)|)} & \text{若 } \mathrm{CF}(h,e_1) \text{ 与 } \mathrm{CF}(h,e_2) \text{ 异号} \\ \mathrm{CF}(h,e_1)+\mathrm{CF}(h,e_2)+\mathrm{CF}(h,e_1)\mathrm{CF}(h,e_2) & \text{若 } \mathrm{CF}(h,e_1) < 0, \mathrm{CF}(h,e_2) < 0 \end{cases} \tag{6-8}$$

(3) 顺序规则的不确定计算(顺序法则)

定义 6.7　若顺序使用两条规则：

if　e_1　then　e_2

if　e_2　then　h

且证据 e_1 **出现使中间结论** e_2 **成立的可信度为** $\mathrm{CF}(e_2,e_1)$**，中间结论** e_2 **出现使结论** h **成立的可信度为** $\mathrm{CF}(h,e_2)$**，则证据** e_1 **出现使结论** h **成立的可信度** $\mathrm{CF}(h,e_1)$**为**

$$\mathrm{CF}(h,e_1) = \mathrm{CF}(h,e_2) \times \max(0,\mathrm{CF}(e_2,e_1)) \tag{6-9}$$

同样给出信任度与不信任度的计算方法为

$$\mathrm{MB}(h,e_1) = \mathrm{MB}(h,e_2) \times \max(0,\mathrm{CF}(e_2,e_1))$$

$$\mathrm{MD}(h,e_1) = \mathrm{MD}(h,e_2) \times \max(0,\mathrm{CF}(e_2,e_1))$$

在定义 6.7 **中，在** CF **模型的顺序推理链的可信度** CF **和信任度** MB **与不信任度** MD **的计算方法中，都对** $\mathrm{CF}(e_2,e_1)$**取** $\max(0,\mathrm{CF}(e_2,e_1))$**，因此，若** $\mathrm{CF}(e_2,e_1)<0$**，则** $\mathrm{CF}(h,e_1)=0$，$\mathrm{MB}(h,e_1)=0$，$\mathrm{MD}(h,e_1)=0$。**也就是说，**CF **模型在顺序推理链中只传播信任度，不传播不信任度。**

例 6.1　现有下述 5 **条规则及相应规则的可信度为**

R_1：　$a \to c$　$\mathrm{CF}(c,a)=0.8$

R_2：　$b \to c$　$\mathrm{CF}(c,b)=0.5$

R_3：　$c \to d$　$\mathrm{CF}(d,c)=0.7$

R_4：　$d \to f$　$\mathrm{CF}(f,d)=0.9$

R_5：　$e \to f$　$\mathrm{CF}(f,e)=-0.3$

计算证据 a、b、e **同时出现时结论** f **成立的可信度** $\mathrm{CF}(f,abe)$。**且原始证据** a、b、e **都肯定为真，即有** $\mathrm{CF}(a)=1$，$\mathrm{CF}(b)=1$，$\mathrm{CF}(e)=1$。

解　由给出的 5 **条规则，可画出推理与/或图如图** 6.1 **所示。**

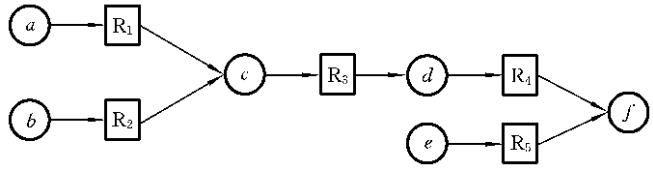

图 6.1　例 6.1 的推理与/或图

① **由**(6-5)**式计算** R_1 **单独推出结论** c **的可信度为**

$$\mathrm{CF}_1(c)=\mathrm{CF}(c,a)\times\max(0,\mathrm{CF}(a))=\mathrm{CF}(c,a)=0.8$$

计算 R_2 **单独推出结论** c **的可信度为**

$$\mathrm{CF}_2(c)=\mathrm{CF}(c,b)\times\max(0,\mathrm{CF}(b))=\mathrm{CF}(c,b)=0.5$$

由于 $\mathrm{CF}(c,a)\geqslant 0$，$\mathrm{CF}(c,b)\geqslant 0$**，故由并行法则**(6-8)**式计算证据** a、b **同时出现时** c **成立的可信度** $\mathrm{CF}(c,ab)$**为**

$$\mathrm{CF}(c,ab) = \mathrm{CF}(c,a) + \mathrm{CF}(c,b) - \mathrm{CF}(c,a)\mathrm{CF}(c,b)$$

$$= 0.8 + 0.5 - 0.8 \times 0.5 = 0.9$$

② 再由顺序法则(6-9)式计算证据 a、b 同时出现时，d 成立的可信度 $CF(d,ab)$ 为

$$CF(d,ab)=CF(d,c)\times\max(0,CF(c,ab))$$
$$=0.7\times0.9=0.63$$

③ 再由顺序法则(6-9)式计算证据 a、b 同时出现时，f 成立的可信度 $CF(f,ab)$ 为

$$CF(f,ab)=CF(f,d)\times\max(0,CF(d,ab))$$
$$=0.9\times0.63=0.567$$

④ 由 R_5 单独推出结论 f 的可信度为

$$CF(f)=CF(f,e)\times\max(0,CF(e))=CF(f,e)=-0.3$$

由于 $CF(f,ab)>0$，$CF(f,e)<0$，故由并行法则(6-8)式计算证据 a、b、e 同时出现时，f 成立的可信度 $CF(f,abe)$ 为

$$CF(f,abe)=\frac{CF(f,ab)+CF(f,e)}{1-\min(|CF(f,ab)|,|CF(e,f)|)}=\frac{0.567-0.3}{1-0.3}=0.381$$

由于 $CF(f,ab)=0.567$，$CF(f,abe)=0.381$，说明证据 e 的出现，反而降低了结论 f 成立的可信度。事实上，由 $CF(f,e)=-0.3<0$，已表示证据 e 的出现使结论 f 成立更不可信。

6.2.2 带有阈限的可信度推理方法

为了使可信度方法能求解更多的问题，在 CF 模型的基础上提出了一些更具有一般性的处理方法。首先讨论带有阈限的可信度不确定推理方法。

1. 带有阈限的规则表示

在带有阈值限度的不确定性推理中，规则用下述形式表示：

$$\text{if}\quad e\quad\text{then}\quad h\quad(CF(h,e),\lambda)$$

其中：

① e 可以是一个简单证据，也可以是多个证据的合取与析取。

② $CF(h,e)$ 是规则的可信度，也称为规则强度，它指出规则为真的可信程度，取值范围为 $0<CF(h,e)\leqslant1$。

③ λ 为规则的阈值，它对规则的使用规定了一个范围，只有当规则的相应证据 e 的可信度 $CF(e)\geqslant\lambda$ 时，该规则才有可能被使用，λ 的取值范围为 $0<\lambda\leqslant1$。

2. 三种基本不确定性测度的计算

(1) 证据组合的不确定计算

证据组合的不确定计算与 CF 模型一样，多个证据的合取组合的可信度为多个证据可信度的极小，多个证据的析取组合的可信度为多个证据可信度的极大。

定义 6.8 多个证据的合取组合与析取组合的可信度为

$$CF(e_1\wedge e_2\wedge\cdots\wedge e_n)=\min(CF(e_1),CF(e_2),\cdots,CF(e_n))$$
$$CF(e_1\vee e_2\vee\cdots\vee e_n)=\max(CF(e_1),CF(e_2),\cdots,CF(e_n))\qquad(6\text{-}10)$$

(2) 并行规则的不确定计算(并行规则)

定义 6.9　设有多条规则有相同的结论,即

$$\begin{aligned}&\text{if}\quad e_1\quad \text{then}\quad h\quad (\mathrm{CF}(h,e_1),\lambda_1)\\&\text{if}\quad e_2\quad \text{then}\quad h\quad (\mathrm{CF}(h,e_2),\lambda_2)\\&\qquad\qquad\vdots\\&\text{if}\quad e_n\quad \text{then}\quad h\quad (\mathrm{CF}(h,e_n),\lambda_n)\end{aligned}$$

若 n 条规则都满足:

$$\mathrm{CF}(e_i)\geqslant\lambda_i\quad i=1,2,\cdots,n$$

且都被使用,则首先分别对每条规则求出结论可信度 $\mathrm{CF}_i(h)$,即

$$\mathrm{CF}_i(h)=\mathrm{CF}(h,e_i)\times\mathrm{CF}(e_i)\quad i=1,2,\cdots,n$$

然后选择下述方法中的一种方法求出结论 h 的可信度 $\mathrm{CF}(h)$。

① 求极大值法。对 n 个 $\mathrm{CF}_i(h)$取极大,即

$$\mathrm{CF}(h)=\max(\mathrm{CF}_1(h),\mathrm{CF}_2(h),\cdots,\mathrm{CF}_n(h)) \tag{6-11}$$

② 加权求和法。用下式计算 $\mathrm{CF}(h)$:

$$\mathrm{CF}(h)=\frac{1}{\sum_{i=1}^{n}\mathrm{CF}(h,e_i)}\sum_{i=1}^{n}\mathrm{CF}(h,e_i)\times\mathrm{CF}(e_i) \tag{6-12}$$

③ 有限和法。对 n 个 $\mathrm{CF}_i(h)$用下式计算 $\mathrm{CF}(h)$:

$$\mathrm{CF}(h)=\min\Big(\sum_{i=1}^{n}\mathrm{CF}_i(h),1\Big) \tag{6-13}$$

(3) 顺序规则的不确定计算(顺序法则)

定义 6.10　有以下两条规则:

$$\begin{aligned}&\mathrm{R}_1:\quad \text{if}\quad e_1\quad \text{then}\quad e_2\quad (\mathrm{CF}(e_2,e_1),\lambda_1)\\&\mathrm{R}_2:\quad \text{if}\quad e_2\quad \text{then}\quad h\quad (\mathrm{CF}(h,e_2),\lambda_2)\end{aligned}$$

若 $\mathrm{CF}(e_1)\geqslant\lambda_1$,且规则 R_1被使用,则 e_2的可信度为

$$\mathrm{CF}(e_2)=\mathrm{CF}(e_2,e_1)\times\mathrm{CF}(e_1)$$

若 $\mathrm{CF}(e_2)\geqslant\lambda_2$,且规则 R_2被使用,则得出 h 的可信度为

$$\begin{aligned}\mathrm{CF}(h)&=\mathrm{CF}(h,e_2)\times\mathrm{CF}(e_2)\\&=\mathrm{CF}(h,e_2)\times\mathrm{CF}(e_2,e_1)\times\mathrm{CF}(e_1)\end{aligned} \tag{6-14}$$

这里,“×”既可以是“乘”运算,也可以是“取极小”或其他运算。

6.2.3　加权的可信度推理方法

在前面讨论的可信度不确定推理的方法中,若一条规则的前件有多个证据,则认为各证据都是平等的。在实际问题中并非都是这样。一般来说,一条规则的多个证据对结论的支持程度不相同,或者说各个证据对结论具有不同的重要程度。为此,可在规则中为每个证据引入加权因子,使不同的证据具有不同的“权”。

1. 加权的规则表示

在加权的不确定性推理中，规则用下述形式表示：

$$\text{if}\quad e_1(w_1)\wedge e_2(w_2)\wedge\cdots\wedge e_n(w_n)\quad\text{then}\quad h\quad(\mathrm{CF}(h,e),\lambda)$$

其中：

① w_i称为证据e_i的加权因子，权值的取值范围一般规定为$0\leqslant w_i\leqslant 1, i=1,2,\cdots,n$，且应满足归一条件，即

$$\sum_{i=1}^{n} w_i = 1$$

② 如果证据e_i对结论h成立的重要性较高，则应使e_i具有较大的权值；如果一个证据具有较大的独立性，而其他证据对它有依赖关系，则应使这个证据具有较大的权值。证据权值的确定应通过领域专家给出。

2. 三种基本不确定性测度的计算

(1) 证据组合的不确定计算

定义 6.11 对于证据组合

$$e=e_1(w_1)\wedge e_2(w_2)\wedge\cdots\wedge e_n(w_n)$$

若n个证据的权值满足归一条件，则n个证据的合取组合的可信度用下式计算：

$$\mathrm{CF}(e)=\sum_{i=1}^{n} w_i\times\mathrm{CF}(e_i) \tag{6-15}$$

若n个证据的权值不满足归一条件，则n个证据的合取组合的可信度用下式计算：

$$\mathrm{CF}(e)=\frac{1}{\sum_{i=1}^{n} w_i}\sum_{i=1}^{n}(w_i\times\mathrm{CF}(e_i)) \tag{6-16}$$

(2) 并行规则与顺序规则的不确定计算

对于加权规则的并行法则和顺序法则，根据定义 6.11 计算出各规则的证据组合的可信度$\mathrm{CF}(e)$之后，与带有阈限的并行法则和顺序法则一样，采用(6-11)式、(6-12)式和(6-13)式中的一个计算式计算并行规则的结论可信度$\mathrm{CF}(h)$，采用(6-14)式计算顺序规则的结论可信度$\mathrm{CF}(h)$。

3. 加权规则的不确定推理

在规则中引入证据的加权因子，不仅解决了多个证据对结论支持的重要程度不同和各个证据之间的独立性与依赖性不同的表示问题，而且解决了当证据不完整时不确定推理问题。

例 6.2 在动物识别专家系统的规则库中有下述一条加权规则：

if 动物有蹄(0.3)
且动物有长颈(0.2)
且动物有长腿(0.2)
且动物是黄褐色(0.13)

且动物体上有暗黑色斑点(0.13)
且动物体重在 100kg 以上(0.04)
then　动物是长颈鹿(0.95,0.8)

若综合数据库中有以下证据：

动物有蹄	可信度 $CF(e_1)=1$
动物有长颈	可信度 $CF(e_2)=1$
动物有长腿	可信度 $CF(e_3)=1$
动物是黄褐色	可信度 $CF(e_4)=0.8$
动物体上有暗黑色斑点	可信度 $CF(e_5)=0.6$

问该动物是什么动物？

解　由于数据库中提供的证据比规则前提要求的证据少了一个，即“动物体重在 100 kg 以上”，因此，若使用不带加权因子的不确定推理，那么这条规则就不能被使用。但是，应用加权的不确定推理，这条规则就可能被使用，因为通过匹配后，并根据(6-15)式计算出该规则不完全匹配的组合证据可信度为

$$\begin{aligned} CF(e) &= \sum_{i=1}^{n} w_i \times CF(e_i) \\ &= 0.3\times 1+0.2\times 1+0.2\times 1+0.13\times 0.8+0.13\times 0.6 \\ &= 0.882 \end{aligned}$$

由于规则的阈值 $\lambda=0.8$，故有 $CF(e)>\lambda$，所以该规则可被使用，并推出结论“动物是长颈鹿”，且结论 h 的可信度为

$$\begin{aligned} CF(h) &= CF(h,e)\times CF(e) \\ &= 0.95\times 0.882 \\ &= 0.84 \end{aligned}$$

由此例可见，对规则引入加权因子，使加权不确定推理可对提供的不完整证据进行推理。在实际情况中，经常会出现某些不重要的证据没有被提供的情况，如果因此就不能进行推理，从而推不出本应推出的结论，那将是不合情理的。

4. 加权规则的冲突消解

对规则引入加权因子后，有利于加权规则的冲突消解。首先说明加权不确定推理的冲突消解方法。

设有以下两条规则：

R_1：　if　$e_1(w_1)\wedge e_2(w_2)$　then　h_1　(CF_1,λ_1)

R_2：　if　$e_3(w_3)\wedge e_4(w_4)\wedge e_5(w_5)$　then　h_2　(CF_2,λ_2)

若与数据库中已有证据经过匹配，并计算得到

$$CF(e_1\wedge e_2)>\lambda_1$$

$$CF(e_3\wedge e_4\wedge e_5)>\lambda_2$$

则两条规则都可被使用，而且它们发生了冲突。冲突消解的方法是：比较发生冲突的规则的组

合证据的可信度值，选择组合证据可信度大的规则优先执行。若 $CF(e_3 \wedge e_4 \wedge e_5) > CF(e_1 \wedge e_2)$，则优先执行规则 R_2。

例 6.3 设有以下规则：

R_1：if $e_1(0.6) \wedge e_2(0.4)$ then e_6 $(0.8, 0.75)$

R_2：if $e_3(0.5) \wedge e_4(0.3) \wedge e_5(0.2)$ then e_7 $(0.7, 0.6)$

R_3：if $e_6(0.7) \wedge e_7(0.3)$ then h $(0.75, 0.6)$

若数据库中提供的证据有 e_1、e_2、e_3、e_4 和 e_5，它们的可信度分别是

$CF(e_1)=0.9$ $CF(e_2)=0.8$ $CF(e_3)=0.7$

$CF(e_4)=0.6$ $CF(e_5)=0.5$

求结论 h 的可信度 $CF(h)$ 的值。

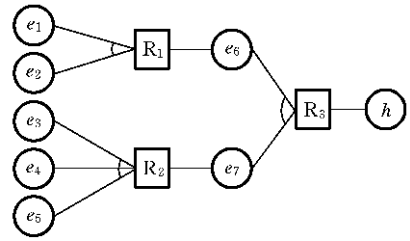

图 6.2 例 6.3 的推理与/或图

解 规则 R_1 可匹配，并计算组合证据可信度为

$$\begin{aligned} CF(e_1 \wedge e_2) &= w_1 \times CF(e_1) + w_2 \times CF(e_2) \\ &= 0.6 \times 0.9 + 0.4 \times 0.8 \\ &= 0.86 \end{aligned}$$

且因为 $\lambda_1 = 0.75$，有 $CF(e_1 \wedge e_2) > \lambda_1$，故 R_1 可被使用。

规则 R_2 也可匹配，并计算组合证据可信度为

$$\begin{aligned} CF(e_3 \wedge e_4 \wedge e_5) &= w_3 \times CF(e_3) + w_4 \times CF(e_4) + w_5 \times CF(e_5) \\ &= 0.5 \times 0.7 + 0.3 \times 0.6 + 0.2 \times 0.5 \\ &= 0.63 \end{aligned}$$

且因为 $\lambda_2 = 0.6$，有 $CF(e_3 \wedge e_4 \wedge e_5) > \lambda_2$，故 R_2 也可被使用。

又因为 $CF(e_1 \wedge e_2) > CF(e_3 \wedge e_4 \wedge e_5)$，故 R_1 被优先使用，在数据库中增加证据 e_6，且 e_6 的可信度为

$$\begin{aligned} CF(e_6) &= CF(e_6, e_1 \wedge e_2) \times CF(e_1 \wedge e_2) \\ &= 0.8 \times 0.86 \\ &= 0.69 \end{aligned}$$

然后，执行规则 R_2，在数据库中增加证据 e_7，且 e_7 的可信度为

$$\begin{aligned} CF(e_7) &= CF(e_7, e_3 \wedge e_4 \wedge e_5) \times CF(e_3 \wedge e_4 \wedge e_5) \\ &= 0.7 \times 0.63 \\ &= 0.44 \end{aligned}$$

最后，规则 R_3 可匹配，并计算组合证据可信度为

$$\begin{aligned} CF(e_6 \wedge e_7) &= w_6 \times CF(e_6) + w_7 \times CF(e_7) \\ &= 0.7 \times 0.69 + 0.3 \times 0.44 \\ &= 0.615 \end{aligned}$$

且因为 $\lambda_3 = 0.6$，有 $CF(e_6 \wedge e_7) > \lambda_3$，故 R_3 被使用，在数据库中增加结论 h，且 h 的可信度为

$$\begin{aligned} CF(h) &= CF(h, e_6 \wedge e_7) \times CF(e_6 \wedge e_7) \\ &= 0.75 \times 0.615 \\ &= 0.46 \end{aligned}$$

6.3　模糊推理方法

自从 1965 年查德(Zadeh)发表关于模糊集合论的具有开创性的论文以后，模糊理论及应用的研究发展很快，其应用领域几乎包括了自然科学、社会科学和工程技术的每一个分支。1978 年，查德又提出了模糊逻辑(Fuzzy Logic)和可能性理论(Possibility Theory)，立即引起了人工智能领域的广泛关注。经过众多研究者的不断努力，模糊逻辑和可能性理论已经得到进一步发展，并在专家系统不确定推理模型的设计中显示了很强的生命力。人们认为，人工智能的更大突破在很大程度上有赖于人工智能与模糊逻辑的更深层次上的结合。

6.3.1　模糊集合的定义与运算

在经典集合论中，论域是要讨论的问题所涉及的对象的全体，是一个普通集合，通常用大写字母 U、V、W 等表示论域。论域 U 的子集 A 在经典集合论中可以有以下两种表示方式：

① A 为满足某种性质 $p(x)$ 的对象集合，即

$$A=\{x \mid x\in U,\quad \text{且 } x \text{ 满足 } p(x)\}$$

② 用特征函数表示，即

$$\mu_A(x)=\begin{cases}1 & x\in A\\ 0 & x\notin A\end{cases}$$

也就是说，若对象 x 属于集合 A，则 x 的特征函数 $\mu_A(x)=1$；若对象 x 不属于集合 A，则 x 的特征函数 $\mu_A(x)=0$。因此，经典集合论只能表示出对象“非此即彼”或“非真即伪”。但是，现实世界中许多事物的性质或特征的取值边界是模糊的，使得对事物的分类存在“亦此亦彼”的现象，不能简单地把某一事物划归某个集合或者排除在这个集合之外。因此，需要把经典集合扩展到模糊集合，并形成完整的模糊集合论和模糊推理的方法。

在经典集合论中，若论域 U 中的子集 A 和 B 的运算用特征函数来表示，则并集 $A\cup B$、交集 $A\cap B$、补集 $\overline{A}$ 的特征函数分别为

$$\mu_{A\cup B}(x)=\max\{\mu_A(x),\mu_B(x)\}\quad \forall x\in U$$
$$\mu_{A\cap B}(x)=\min\{\mu_A(x),\mu_B(x)\}\quad \forall x\in U$$
$$\mu_{\overline{A}}(x)=1-\mu_A(x)\quad \forall x\in U$$

1. 模糊集合的定义与表示

定义 6.12　论域 $U=\{x\}$ 上的集合 A 可由隶属函数 $\mu_A(x)$ 表示，$\mu_A(x)$ 在闭区间 $[0,1]$ 中的取值称为 x 属于模糊集合 A 的隶属度，隶属度越接近于 1，x 属于 A 的程度就越大，反之就越小。

这就是说，论域 $U=\{x\}$ 上的模糊集合是指 U 中的元素 x 具有某种特征或性质的元素集合，论域元素总是分明的，但元素 x 属于集合 A 的程度是不分明的，因此，集合 A 是一个模糊

集合。

例如，论域 U 是 0 到 120 之间的年龄值，模糊集合“年轻”可以用隶属函数表示为

$$\mu_{\text{年轻}}(x)=\frac{1}{1+\left(\frac{x}{30}\right)^2}\qquad x=0,1,2,\cdots,120$$

这个“年轻”的隶属函数的函数图形如图 6.3 所示。

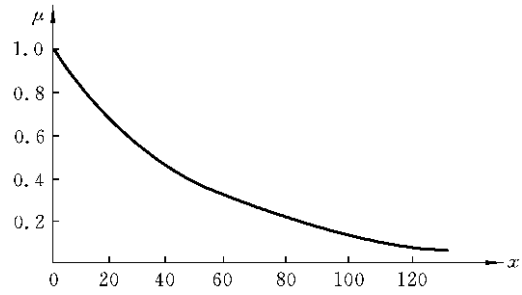

图 6.3 “年轻”的隶属函数的函数图形

模糊集合还有一种表示形式，由定义 6.13 和定义 6.14 给出。

定义 6.13 设论域 U 是有限域，即 $U=\{x_1,x_2,\cdots,x_n\}$，U 上的任一模糊集合 A 可表示为

$$\begin{aligned}A&=\mu_A(x_1)/x_1+\mu_A(x_2)/x_2+\cdots+\mu_A(x_n)/x_n\\&=\sum_{i=1}^{n}\mu_A(x_i)/x_i\end{aligned}$$

其中，$\mu_A(x_i)$ 是 x_i 属于 A 的隶属度。这里，$+$ 与 $\sum$ 不是表示相加与求和，$\mu_A(x_i)/x_i$ 也不是分数，它们只是为表示模糊集而引用的符号。若 $\mu_A(x_i)=0$，则模糊集 A 的上述表示中的相应 $\mu_A(x_i)/x_i$ 项可以省略。

例如，模糊集“年轻”可以表示为

$$\text{年轻}=\sum_{i=0}^{120}\left(1+\left(\frac{x_i}{30}\right)^2\right)^{-1}\Big/x_i$$

设论域 $U=\{1,2,\cdots,9\}$，若 A 为接近 5 的整数集合，则 A 可以表示为

$$A=0.1/1+0.2/2+0.4/3+0.7/4+1/5+0.7/6+0.4/7+0.2/8+0.1/9$$

定义 6.14 设论域 U 是无限域，U 上的任一模糊集 A 可表示为

$$A=\int_{x\in U}\mu_A(x)/x$$

同样，$\int$ 不是积分符号，只是表示无限论域上的一个模糊集的符号。

例如，设论域 U 是实数集 R，A 为小实数的集合，则 A 可以表示为

$$A=\int_R(1+x^2)^{-1}/x$$

定义 6.15 当且仅当对于论域 U 中的任意元素 x，有 $\mu_A(x)=\mu_B(x)$，则称两个模糊集 A 和 B 相等，记为 $A=B$。

定义 6.16 若 $\forall x\in U$，有 $\mu_A(x)\leqslant\mu_B(x)$，则称模糊集 A 是模糊集 B 的子集，记为 $A\subseteq B$。若 $A\subseteq B$，且 $\forall x\in U$，有 $\mu_A(x)<\mu_B(x)$，则称模糊集 A 是模糊集 B 的真子集，记为 $A\subset B$。

2. 模糊集合的运算

模糊集合是经典集合的扩展，因此，可类似经典集合定义模糊集合的并集、交集和补集的运算。

定义 6.17　设 A、B 是 U 上的模糊集，A 和 B 的并集 $A\cup B$、交集 $A\cap B$ 和补集 $\overline{A}$ 的隶属函数定义分别为

$$\mu_{A\cap B}(x)=\max(\mu_A(x),\mu_B(x))=\mu_A(x)\vee\mu_B(x)$$
$$\mu_{A\cap R}(x)=\min(\mu_A(x),\mu_B(x))=\mu_A(x)\wedge\mu_B(x)$$
$$\mu_{\overline{A}}(x)=1-\mu_A(x)$$

在模糊集合论中，常用“$\vee$”表示 max，用“$\wedge$”表示 min，分别称为取极大、取极小运算。不要把它们与谓词逻辑中的析取符号“$\vee$”与合取符号“$\wedge$”混淆起来，在不同应用场合，它们的含义是不同的。

定义 6.18　按照论域 U 分别是有限域和无限域，模糊集 A 和 B 的并、交和补的计算分别如下：

① 论域 $U=\{x_1,x_2,\cdots,x_n\}$，且 $A=\sum_{i=1}^{n}\mu_A(x_i)/x_i$，$B=\sum_{i=1}^{n}\mu_B(x_i)/x_i$，则

$$A\cup B=\sum_{i=1}^{n}(\mu_A(x_i)\vee\mu_B(x_i))/x_i$$
$$A\cap B=\sum_{i=1}^{n}(\mu_A(x_i)\wedge\mu_B(x_i))/x_i$$
$$\overline{A}=\sum_{i=1}^{n}(1-\mu_A(x_i))/x_i$$

② 论域 U 为无限域，且 $A=\int_{x\in U}\mu_A(x)/x$，$B=\int_{X\in U}\mu_B(x)/x$，则

$$A\cup B=\int_{x\in U}(\mu_A(x)\vee\mu_B(x))/x$$
$$A\cap B=\int_{x\in U}(\mu_A(x)\wedge\mu_B(x))/x$$
$$\overline{A}=\int_{x\in U}(1-\mu_A(x))/x$$

例 6.4　设论域 $U=\{x_1,x_2,x_3,x_4,x_5\}$，且有

$$A=0.2/x_1+0.7/x_2+1/x_3+0.5/x_5$$
$$B=0.5/x_1+0.3/x_2+0.1/x_4+0.7/x_5$$

计算 $A\cup B$、$A\cap B$ 和 $\overline{A}$。

解　由定义 6.18，可得

$$\begin{aligned}A\cup B&=(0.2\vee0.5)/x_1+(0.7\vee0.3)/x_2+(1\vee0)/x_3+(0\vee0.1)/x_4+(0.5\vee0.7)/x_5\\&=0.5/x_1+0.7/x_2+1/x_3+0.1/x_4+0.7/x_5\\A\cap B&=(0.2\wedge0.5)/x_1+(0.7\wedge0.3)/x_2+(1\wedge0)/x_3+(0\wedge0.1)/x_4+(0.5\wedge0.7)/x_5\\&=0.2/x_1+0.3/x_2+0.5/x_5\\\overline{A}&=(1-0.2)/x_1+(1-0.7)/x_2+(1-1)/x_3+(1-0)/x_4+(1-0.5)/x_5\\&=0.8/x_1+0.3/x_2+1/x_4+0.5/x_5\end{aligned}$$

模糊集还有一个重要的运算，这就是笛卡儿乘积运算。

定义 6.19　设 $A_1,A_2,\cdots,A_n$ 分别是论域 $U_1,U_2,\cdots,U_n$ 上的模糊集，$A_1,A_2,\cdots,A_n$ 的笛

卡儿乘积记为 $A_1\times A_2\times\cdots\times A_n$，它是论域 $U=U_1\times U_2\times\cdots\times U_n$ 上的一个模糊集，其隶属函数定义为

$$\mu_{A_1\times A_2\times\cdots\times A_n}(x_1,x_2,\cdots,x_n)=\mu_{A_1}(x_1)\wedge\mu_{A_2}(x_2)\wedge\cdots\wedge\mu_{A_n}(x_n)$$

若论域是有限域，模糊集的笛卡儿乘积为

$$A_1\times A_2\times\cdots\times A_n=\sum(\mu_{A_1}(x_1)\wedge\mu_{A_2}(x_2)\wedge\cdots\wedge\mu_{A_n}(x_n))/(x_1,x_2,\cdots,x_n)$$

若论域是无限域，模糊集的笛卡儿乘积为

$$A_1\times A_2\times\cdots\times A_n=\int_U(\mu_{A_1}(x_1)\wedge\mu_{A_2}(x_2)\wedge\cdots\wedge\mu_{A_n}(x_n))/(x_1,x_2,\cdots,x_n)$$

例 6.5 设 $U_1=U_2=\{3,5,7\}$，$A_1=0.5/3+1/5+0.6/7$，$A_2=1/3+0.6/5$，计算 $A_1\times A_2$。

解
$$\begin{aligned}A_1\times A_2&=(0.5\wedge1)/(3,3)+(1\wedge1)/(5,3)+(0.6\wedge1)/(7,3)\\&\quad+(0.5\wedge0.6)/(3,5)+(1\wedge0.6)/(5,5)+(0.6\wedge0.6)/(7,5)\\&=0.5/(3,3)+1/(5,3)+0.6/(7,3)+0.5/(3,5)+0.6/(5,5)+0.6/(7,5)\end{aligned}$$

3. 模糊关系

模糊关系是经典集合的普通关系的扩展。普通关系描述两个集合的元素之间是否关联，模糊关系描述两个集合的元素之间的关联程度有多大。

定义 6.20 设 U 和 V 分别是论域，模糊关系 R 是笛卡儿乘积 $U\times V=\{(x,y)\mid x\in U,y\in V\}$ 中的模糊集，R 的隶属函数表示为 $\mu_R(x,y)$。

若 $U=\{x_1,\cdots,x_m\}$，$V=\{y_1,\cdots,y_n\}$，隶属度 $\mu_{ij}=\mu_R(x_i,y_j)$ 表示 U 中的元素 x_i 与 V 中的元素 y_j 的关联程度，则二元模糊关系 R 可以表示成隶属度矩阵的形式：

$$R=\begin{bmatrix}\mu_{11}&\cdots&\mu_{1n}\\\vdots&&\vdots\\\mu_{m1}&\cdots&\mu_{mm}\end{bmatrix}$$

一般地，笛卡儿乘积 $U_1\times U_2\times\cdots\times U_n$ 上的 n 元模糊关系 R 的隶属函数表示为

$$\mu_R(x_1,x_2,\cdots,x_n)$$

论域 $U\times U$ 上的模糊关系也称为论域 U 上的模糊关系。

例如，设 $U=\{x_1,x_2,x_3\}$ 表示三个人的集合，$U\times U$ 上表示“彼此熟悉”的模糊关系 R 可以表示为

$$\begin{aligned}R=&1/(x_1,x_1)+0.7/(x_1,x_2)+0.5/(x_1,x_3)+0.9/(x_2,x_1)+1/(x_2,x_2)\\&+0.4/(x_2,x_3)+0.5/(x_3,x_1)+0.1/(x_3,x_2)+1/(x_3,x_3)\end{aligned}$$

也可以表示为隶属度矩阵的形式：

$$\mathrm{R}=\begin{bmatrix}1&0.7&0.3\\0.9&1&0.4\\0.5&0.1&1\end{bmatrix}$$

式中，$\mu_{ii}=1$，$i=1,2,3$，表示每个人最熟悉自己；$\mu_{32}=\mu_R(x_3,x_2)=0.1$，表示 x_3 对 x_2 稍稍有点熟悉，而 $\mu_{23}=\mu_R(x_2,x_3)=0.4$，表示 x_2 对 x_3 比较熟悉，显然，x_2 对 x_3 的熟悉程度高于 x_3 对 x_2 的熟悉程度。

由于模糊关系也是模糊集，因此，可类似模糊集来定义模糊关系的包含、相等、并、交、补的运算。

定义 6.21　设 R_1、R_2 是 $U\times V$ 上的两个模糊关系，则有

① 包含：若 $R_1\subseteq R_2$，当且仅当 $\mu_{R_1}(x,y)\leqslant\mu_{R_2}(x,y)$，$\forall x\in U$，$\forall y\in V$。

② 相等：若 $R_1=R_2$，当且仅当 $\mu_{R_1}(x,y)=\mu_{R_2}(x,y)$，$\forall x\in U$，$\forall y\in V$。

③ 模糊关系 R_1 和 R_2 的并集 $R_1\cup R_2$ 的隶属函数为

$$\mu_{R_1\cup R_2}(x,y)=\max(\mu_{R_1}(x,y),\quad \mu_{R_2}(x,y))=\mu_{R_1}(x,y)\vee\mu_{R_2}(x,y)$$

④ 模糊关系 R_1 和 R_2 的交集 $R_1\cap R_2$ 的隶属函数为

$$\mu_{R_1\cap R_2}(x,y)=\min(\mu_{R_1}(x,y),\quad \mu_{R_2}(x,y))=\mu_{R_1}(x,y)\wedge\mu_{R_2}(x,y)$$

⑤ 模糊关系 R 的补集 $\overline{R}$ 的隶属函数为

$$\mu_{\overline{R}}(x,y)=1-\mu_R(x,y)$$

模糊关系有一个独特的运算，就是模糊关系的合成。

定义 6.22　若 R_1 是论域 $U\times V$ 上的模糊关系，R_2 是论域 $V\times W$ 上的模糊关系，则 R_1 和 R_2 的合成 $R_1\circ R_2$ 是 $U\times W$ 上的模糊关系，$R_1\circ R_2$ 的隶属度定义为

$$\begin{aligned}\mu_{R_1\circ R_2}(x,z)&=\max(\min(\mu_{R_1}(x,y),\mu_{R_2}(y,z))\\&=\bigvee_{y\in V}(\mu_{R_1}(x,y)\wedge\mu_{R_2}(y,z))\quad\forall(x,z)\in(U\times W)\end{aligned}$$

由于二元模糊关系可以用一个隶属度矩阵来表示，因此，模糊关系的并、交、补和合成运算也有相应的矩阵运算形式。

定义 6.23　若有二元模糊关系 $A=[a_{ij}]_{m\times n}$，$B=[b_{ij}]_{m\times n}$，矩阵元素 a_{ij} 和 b_{ij} 分别是 A 和 B 中相应的隶属度 μ_{ij}，则模糊关系 A 和 B 的并、交、补运算分别是

$$A\cup B=[a_{ij}\vee b_{ij}]_{m\times n}$$

$$A\cap B=[a_{ij}\wedge b_{ij}]_{m\times n}$$

$$\overline{A}=[1-a_{ij}]_{m\times n}$$

若有二元模糊关系 $A=[a_{ij}]_{m\times q}$，$B=[b_{kj}]_{q\times n}$，则模糊关系 A 和 B 的合成

$$A\circ B=[r_{ij}]_{m\times n}$$

其中，

$$r_{ij}=\max_{k=1}^{q}(\min(a_{ik},b_{kj}))$$

记为

$$r_{ij}=\bigvee_{k=1}^{q}(a_{ik}\wedge b_{kj})$$

例 6.6　若有二元模糊关系 $A=\begin{bmatrix}0.1 & 0.2\\0.3 & 0.1\end{bmatrix}$，$B=\begin{bmatrix}0.3 & 0.2\\0.1 & 0.7\end{bmatrix}$，计算 $\overline{A}$、$A\cup B$、$A\cap B$ 和 $A\circ B$。

解

$$\overline{A}=\begin{bmatrix}1-0.1 & 1-0.2\\1-0.3 & 1-0.1\end{bmatrix}=\begin{bmatrix}0.9 & 0.8\\0.7 & 0.9\end{bmatrix}$$

$$A\cup B=\begin{bmatrix}0.1\vee0.3 & 0.2\vee0.2\\0.3\vee0.1 & 0.1\vee0.7\end{bmatrix}=\begin{bmatrix}0.3 & 0.2\\0.3 & 0.7\end{bmatrix}$$

$$A\cap B=\begin{bmatrix}0.1\wedge 0.3 & 0.2\wedge 0.2\\ 0.3\wedge 0.1 & 0.1\wedge 0.7\end{bmatrix}=\begin{bmatrix}0.1 & 0.2\\ 0.1 & 0.1\end{bmatrix}$$

$$A\circ B=[r_{ij}]_{2\times 2}$$

其中，

$$r_{11}=\vee(a_{11}\wedge b_{11},a_{12}\wedge b_{21})=\vee(0.1\wedge 0.3,0.2\wedge 0.1)$$
$$=\vee(0.1,0.1)=0.1$$
$$r_{12}=\vee(a_{11}\wedge b_{12},a_{12}\wedge b_{22})=\vee(0.1\wedge 0.2,0.2\wedge 0.7)$$
$$=\vee(0.1,0.2)=0.2$$
$$r_{21}=\vee(a_{21}\wedge b_{11},a_{22}\wedge b_{21})=\vee(0.3\wedge 0.3,0.1\wedge 0.1)$$
$$=\vee(0.3,0.1)=0.3$$
$$r_{22}=\vee(a_{21}\wedge b_{12},a_{22}\wedge b_{22})=\vee(0.3\wedge 0.2,0.1\wedge 0.7)$$
$$=\vee(0.2,0.1)=0.2$$

所以

$$A\circ B=\begin{bmatrix}r_{11} & r_{12}\\ r_{21} & r_{22}\end{bmatrix}=\begin{bmatrix}0.1 & 0.2\\ 0.3 & 0.2\end{bmatrix}$$

4. 模糊变换

定义 6.24 设

$$A=\{\mu_A(u_1),\mu_A(u_2),\cdots,\mu_A(u_n)\}$$

是论域 U 上的模糊集，R 是 $U\times V$ 上的模糊关系，则

$$A\circ R=B$$

称为模糊变换。

例 6.7 设模糊集 A 和模糊关系 R 分别为

$$A=\{0.2,0.5,0.3\}$$

$$R=\begin{bmatrix}0.2 & 0.7 & 0.1 & 0\\ 0 & 0.4 & 0.5 & 0.1\\ 0.2 & 0.3 & 0.4 & 0.1\end{bmatrix}$$

求模糊集 $B=A\circ R$。

解 由模糊变换的合成运算，可得

$$B=A\circ R=\{0.2,0.4,0.5,0.1\}$$

例 6.8 设对某产品进行评判，评判的标准是：质量(u_1)、价格(u_2)、售后服务(u_3)，它们构成了论域 U：

$$U=\{u_1,u_2,u_3\}$$

由若干评委根据评判标准对产品进行模糊评判，评判的等级是：好(v_1)、较好(v_2)、一般(v_3)、差(v_4)，它们构成了论域 V：

$$V=\{v_1,v_2,v_3,v_4\}$$

若对“质量”(u_1)评判的结果是：60%的评委认为是“好”，20%的评委认为是“较好”，20%的评委认为是“一般”，则对该产品的“质量”的评价是$\{0.6,0.2,0.2,0\}$。

若对“价格”(u_2)的评价是$\{0.8,0.1,0.1,0\}$，若对“售后服务”(u_3)的评价是$\{0.3,0.3,$

0.3,0.1\},则可得出多名评委对产品的模糊评价 R 为

$$R=\begin{bmatrix}0.6 & 0.2 & 0.2 & 0\\ 0.8 & 0.1 & 0.1 & 0\\ 0.3 & 0.3 & 0.3 & 0.1\end{bmatrix}$$

若根据客户的反馈意见,30%的客户最关心产品的质量(u_1),30%的客户最关心产品的价格(u_2),40%的客户最关心产品的售后服务(u_3),则它们构成了 U 上的一个模糊集 A:

$$A=\{0.3,0.3,0.4\}$$

由 R 和 A 可得出对该产品的模糊综合评判为 B:

$$\begin{aligned}B&=A\circ R\\&=\{0.3,0.3,0.4\}\circ\begin{bmatrix}0.6 & 0.2 & 0.2 & 0\\ 0.8 & 0.1 & 0.1 & 0\\ 0.3 & 0.3 & 0.3 & 0.1\end{bmatrix}\\&=\{0.3,0.3,0.3,0.1\}\end{aligned}$$

B 是 V 上的模糊集,由 B 可得出综合评判结果:30%评委认为该产品是“好”,30%的评委认为该产品是“较好”,30%评委认为该产品是“一般”,10%的评委认为该产品是“差”。当然,上述结论是用客户反馈意见 A 对评委评判意见 R 进行了模糊变换后的结果。

6.3.2　模糊知识表示与模糊匹配

首先说明模糊知识的表示方式,然后讨论模糊知识的匹配问题。

1. 模糊知识表示

(1) 模糊命题

人们在日常生活中经常会用到一些模糊概念或模糊数据,例如:

李明是一个年轻人,他的身高在 1.75m 左右

这里,“年轻”是一个模糊概念,“1.75m 左右”是一个模糊数据。另外,人们在表述一个事件时,通常会对事件发生的可能性或确信程度作出判断,例如:

今年冬季不会太冷的可能性很大

这里,用模糊语言值“很大”描述了模糊事件“不会太冷”出现的可能性或确信程度。

含有模糊概念、模糊数据或带有确信程度的语句称为模糊命题。它的一般表示形式为:

$$x \quad \text{is} \quad A$$

或者

$$x \quad \text{is} \quad A \quad (\text{CF})$$

其中,x 是论域上的变量,用以代表所论对象的属性;A 是模糊概念或模糊数,用相应的模糊集及隶属函数刻画;CF 是该模糊命题的确信度或相应事件发生的可能性程度,它既可以是一个确定的数,也可以是一个模糊数或模糊语言值。

所谓模糊语言值是指表示大小、长短、轻重、快慢、多少等程度的修饰词。具体应用时可根据实际需要来确定自己的模糊语言值集合,例如,表示大小程度的一种模糊语言值集合可以是

$V=\{$最大，极大，很大，相当大，比较大，稍大，稍小，比较小，相当小，很小，极小，最小$\}$

扎德等人主张对模糊语言值用定义在$[0,1]$上的模糊集来表示。

(2) 模糊规则

模糊规则的一般形式为

$$\text{if} \quad E \quad \text{then} \quad H \quad (\text{CF},\lambda)$$

其中，E 是用模糊命题表示的模糊条件，它既可以是由单个模糊命题表示的简单条件，也可以是由多个模糊命题构成的组合条件；H 是用模糊命题表示的模糊结论；CF 是模糊规则的可信度因子，它既可以是一个确定的数，也可以是一个模糊数或模糊语言值；λ 是规则的阈值，用于指出规则可被使用的限制。

2. 模糊匹配

由于模糊规则的模糊条件是模糊命题，欲与之匹配的证据也是模糊命题，因此需要采用模糊匹配的方法来计算两个模糊命题的相似程度，即匹配度。例如，设有下述模糊规则及证据：

$$\text{if} \quad x \quad \text{is} \quad \textbf{小} \quad \text{then} \quad y \quad \text{is} \quad \textbf{大} \quad (0.6)$$

$$x \quad \text{is} \quad \textbf{较小}$$

其中，x 是论域 U 上的一个变元，规则的前提条件“x is 小”可表示成模糊集 A，规则的阈值 $\lambda=0.6$，证据“x is 较小”可表示成模糊集 B：

$$U=\{1,2,3,4,5,6,7,8,9,10\}$$

$$A=1/1+0.8/2+0.6/3+0.4/4+0.2/5$$

$$B=1/1+0.89/2+0.77/3+0.63/4+0.45/5$$

为了确定规则的条件是否可与证据模糊匹配，就需要对两个模糊集 A 和 B 来计算匹配度 $\delta(A,B)$，若$\delta(A,B)\geqslant\lambda$，就认为 A 与 B 匹配。

(1) 贴近度

贴近度是指两个模糊概念互相贴近的程度，可用来作为匹配度。

定义 6.25 设 A 与 B 分别是论域 $U=\{u_1,u_2,\cdots,u_n\}$ 上的表示两个模糊概念的模糊集，则它们的贴近度定义为

$$(A,B)=\frac{1}{2}[A\cdot B+(1-A\odot B)]$$

其中，$A\cdot B$ 称为 A 与 B 的内积，$A\odot B$ 称为 A 与 B 的外积，分别为

$$A\cdot B=\bigvee_U(\mu_A(u_i)\wedge\mu_B(u_i))$$

$$A\odot B=\bigwedge_U(\mu_A(u_i)\vee\mu_B(u_i))$$

当用贴近度作为匹配度时，贴近度越大表示越匹配。若贴近度大于等于规则指定的阈值时，就认为规则的模糊条件与证据匹配。

例 6.9 设 $U=\{a,b,c,d,e,f\}$

$$A=0.6/a+0.8/b+1/c+0.8/d+0.6/e+0.4/f$$

$$B=0.4/a+0.6/b+0.8/c+1/d+0.8/e+0.6/f$$

求 A 与 B 的贴近度。

解　由定义 6.25，可计算得出：

$$A \cdot B = 0.4 \vee 0.6 \vee 0.8 \vee 0.8 \vee 0.6 \vee 0.4 = 0.8$$

$$A \odot B = 0.6 \wedge 0.8 \wedge 1 \wedge 1 \wedge 0.8 \wedge 0.6 = 0.6$$

$$(A,B) = \frac{1}{2}[0.8 + (1 - 0.6)] = 0.6$$

$$\delta(A,B) = (A,B) = 0.6$$

(2) 语义距离

可以通过计算两个模糊集 A 与 B 的语义距离 $d(A,B)$ 来计算 A 与 B 的匹配度。语义距离有多种计算方法。

定义 6.26　设 A 与 B 分别是论域 $U=\{u_1,u_2,\cdots,u_n\}$ 上的表示两个模糊概念的模糊集，则它们之间的海明距离定义为

$$d(A,B) = \frac{1}{n}\sum_{i=1}^{n} |\mu_A(u_i) - \mu_B(u_i)|$$

欧几里德距离定义为

$$d(A,B) = \frac{1}{\sqrt{n}} \times \sqrt{\sum_{i=1}^{n} (\mu_A(u_i) - \mu_B(u_i))^2}$$

明可夫斯基距离定义为

$$d(A,B) = \left[\frac{1}{n} \times \sum_{i=1}^{n} |\mu_A(u_i) - \mu_B(u_i)|^q\right]^{1/q} \quad q \geqslant 1$$

切比雪夫距离定义为

$$d(A,B) = \max_{1 \leqslant i \leqslant n} \{|\mu_A(u_i) - \mu_B(u_i)|\}$$

实际上，海明距离和欧几里德距离是明可夫斯基距离的特例，当 $q=1$ 时，就得到海明距离；当 $q=2$ 时，就得到欧几里德距离。

由语义距离 $d(A,B)$，可得出匹配度为

$$\delta(A,B) = 1 - d(A,B)$$

例 6.10　设 $U=\{u_1,u_2,u_3\}$

$$A = 0.3/u_1 + 0.5/u_2 + 0.2/u_3$$

$$B = 0.5/u_1 + 0.8/u_2 + 0.4/u_3$$

由定义 6.26，可计算 A 与 B 的海明距离为

$$d(A,B) = \frac{1}{3} \times (|0.3 - 0.5| + |0.5 - 0.8| + |0.2 - 0.4|) = 0.233$$

欧几里德距离为

$$d(A,B) = \frac{1}{\sqrt{3}} \times \sqrt{(0.3 - 0.5)^2 + (0.5 - 0.8)^2 + (0.2 - 0.4)^2}$$

$$= 0.237$$

切比雪夫距离为

$$d(A,B) = \max\{|0.3 - 0.5|, |0.5 - 0.8|, |0.2 - 0.4|\} = 0.3$$

(3) 相似度

除了贴近度和语义距离可用来确定模糊条件与证据是否模糊匹配外，相似度也可作为匹

配度用于模糊匹配。相似度也有多种计算方法。

定义 6.27 设 A 与 B 分别是论域 $U=\{u_1,u_2,\cdots,u_n\}$ 上的表示两个模糊概念的模糊集，则 A 与 B 之间的相似度定义为

$$r(A,B)=\frac{\sum_{i=1}^{n}\min\{\mu_A(u_i),\mu_B(u_i)\}}{\sum_{i=1}^{n}\max\{\mu_A(u_i),\mu_B(u_i)\}}$$

算术平均相似度定义为

$$r(A,B)=\frac{\sum_{i=1}^{n}\min\{\mu_A(u_i),\mu_B(u_i)\}}{\frac{1}{2}\times\sum_{i=1}^{n}(\mu_A(u_i)+\mu_B(u_i))}$$

几何平均相似度定义为

$$r(A,B)=\frac{\sum_{i=1}^{n}\min\{\mu_A(u_i),\mu_B(u_i)\}}{\sum_{i=1}^{n}\sqrt{\mu_A(u_i)\times\mu_B(u_i)}}$$

例 6.11 设 $U=\{a,b,c,d\}$

$$A=0.3/a+0.4/b+0.6/c+0.8/d$$

$$B=0.2/a+0.5/b+0.6/c+0.7/d$$

计算 A 与 B 的相似度、算术平均相似度和几何平均相似度。

解 由定义 6.27，可计算 A 与 B 的相似度为

$$r(A,B)=\frac{0.3\wedge0.2+0.4\wedge0.5+0.6\wedge0.6+0.8\wedge0.7}{0.3\vee0.2+0.4\vee0.5+0.6\vee0.6+0.8\vee0.7}=0.86$$

算术平均相似度为

$$r(A,B)=\frac{0.3\wedge0.2+0.4\wedge0.5+0.6\wedge0.6+0.8\wedge0.7}{\frac{1}{2}\times(0.3+0.2+0.4+0.5+0.6+0.6+0.8+0.7)}=0.93$$

几何平均相似度为

$$r(A,B)=\frac{0.3\wedge0.2+0.4\wedge0.5+0.6\wedge0.6+0.8\wedge0.7}{\sqrt{0.3\times0.2}+\sqrt{0.4\times0.5}+\sqrt{0.6\times0.6}+\sqrt{0.8\times0.7}}=0.93$$

(4) 精确概念与模糊概念的匹配

两个模糊集的匹配可采用上述计算两个模糊集的匹配度来确定是否匹配。实际上，规则的前提条件与证据也可能一个是精确概念，另一个是模糊概念。对任意 $u\in U$，$\mu_A(u)$ 表示 u 属于模糊概念 A 的隶属度，若精确概念 x 与 u 相匹配，则 x 与 A 的匹配程度就是 $\mu_A(u)$。

例 6.12 设 $U=\{1,2,3,4,5,6,7,8,9,10\}$

$$\mu_{小}=1/1+0.8/2+0.6/3+0.4/4+0.2/5$$

$$\mu_{较小}=1/1+0.89/2+0.77/3+0.63/4+0.45/5$$

$$\mu_{大}=0.2/4+0.4/5+0.6/6+0.8/7+1/8+1/9+1/10$$

且有如下模糊规则：

$$\begin{aligned}
&R_1:\ \text{if}\ \ x\ \ \text{is}\ \ \textbf{小}\ \ \ \text{then}\ \ y\ \ \text{is}\ \ B_1\ \ (0.15)\\
&R_2:\ \text{if}\ \ x\ \ \text{is}\ \ \textbf{较小}\ \ \text{then}\ \ y\ \ \text{is}\ \ B_2\ \ (0.25)\\
&R_3:\ \text{if}\ \ x\ \ \text{is}\ \ \textbf{大}\ \ \ \text{then}\ \ y\ \ \text{is}\ \ B_3\ \ (0.3)
\end{aligned}$$

若提供的证据为

$$e:\ \ x\ \ \text{is}\ \ 5$$

计算证据 e 与三条模糊规则的匹配度。

解　由于精确证据分别与模糊集“小”、“较小”、“大”中的 $u=5$ 匹配，因此，匹配度分别是 $\mu_{小}(5)$、$\mu_{较小}(5)$和 $\mu_{大}(5)$，即

$$\begin{aligned}
&\delta(\textbf{小},5)=\mu_{小}(5)=0.2\\
&\delta(\textbf{较小},5)=\mu_{较小}(5)=0.45\\
&\delta(\textbf{大},5)=\mu_{大}(5)=0.4
\end{aligned}$$

由于证据 e 与规则 R_1、R_2和 R_3的前提条件的匹配度都分别大于规则给定的阈值，即

$$\begin{aligned}
&\delta(\textbf{小},5)>\lambda_1=0.15\\
&\delta(\textbf{较小},5)>\lambda_2=0.25\\
&\delta(\textbf{大},5)>\lambda_3=0.3
\end{aligned}$$

因此，这三条规则发生冲突，需要按一定的冲突消解策略选出一条规则来执行。若冲突消解策略为匹配度大的规则优先，那么，将选择规则 R_2执行。

(5) 模糊规则组合条件与证据的匹配

上述讨论的方法都只考虑模糊规则的前提条件是单一条件与单一证据的模糊匹配问题，实际上，模糊规则的前提条件可以是多个条件的组合条件。

组合条件与多个证据的模糊匹配的步骤如下。

① 选择一种计算单一条件与单一证据匹配度的方法，分别对规则组合条件中的每一个子条件计算出与相应证据的匹配度。

例如，对一条规则的组合条件

$$E=(x_1\ \ \text{is}\ \ A_1)\wedge(x_2\ \ \text{is}\ \ A_2)\wedge(x_3\ \ \text{is}\ \ A_3)$$

及相应证据 E'

$$\begin{aligned}
&x_1\ \ \text{is}\ \ A_1'\\
&x_2\ \ \text{is}\ \ A_2'\\
&x_3\ \ \text{is}\ \ A_3'
\end{aligned}$$

分别计算出 A_i与 A_i'的匹配度 $\delta(A_i,A_i')$，$i=1,2,3$。

② 选择一种计算总匹配度的方法，计算组合条件与证据的总匹配度。对于合取组合条件，计算总匹配度的方法有“取极小”和“相乘”方法，即

$$\delta(E,E')=\min\{\delta(A_i,A_i')\mid i=1,2,\cdots,n\}$$

或

$$\delta(E,E')=\prod_{i=1}^{n}\delta(A_i,A_i')$$

③ 若总匹配度 $\delta(E,E')$大于等于规则的阈值 λ，则可匹配；否则，不可匹配。

6.3.3 简单模糊推理方法

首先给出模糊推理的基本模式，然后讨论简单模糊推理的几种方法。

1. 模糊推理的基本模式

模糊推理有三种基本的推理模式，即模糊假言推理、模糊拒取式推理和模糊三段论推理。

(1) 模糊假言推理

设 A 和 B 分别是论域 U 和 V 上的模糊集，且具有下述规则：

$$\text{if}\quad x\quad \text{is}\quad A\quad \text{then}\quad y\quad \text{is}\quad B$$

若有 U 上的模糊集 A' 可以与 A 模糊匹配，则可推出 y is B'，且 B' 是 V 上的模糊集，称这种模糊推理为模糊假言推理。

模糊假言推理可直观地表示为

规则：$\text{if}\quad x\quad \text{is}\quad A\quad \text{then}\quad y\quad \text{is}\quad B$

证据：$x\quad \text{is}\quad A'$

结论：$y\quad \text{is}\quad B'$

对组合条件的规则，模糊假言推理可以表示为

规则：$\text{if}\quad (x_1\ \text{is}\ A_1)\wedge(x_2\ \text{is}\ A_2)\wedge\cdots\wedge(x_n\ \text{is}\ A_n)\quad \text{then}\quad y\quad \text{is}\quad B$

证据：$x_1\ \text{is}\ A_1'\quad x_2\ \text{is}\ A_2'\quad \cdots\quad x_n\ \text{is}\ A_n'$

结论：$y\quad \text{is}\quad B'$

如果在规则和(或)证据中带有可信度因子，则还需要对结论的可信度因子按某种算法进行计算。

(2) 模糊拒取式推理

设 A 和 B 分别是论域 U 和 V 上的模糊集，且具有下述规则：

$$\text{if}\quad x\quad \text{is}\quad A\quad \text{then}\quad y\quad \text{is}\quad B$$

若有 V 上的模糊集 B' 可以与 B 模糊匹配，则可推出 x is A'，且 A' 是 U 上的模糊集。称这种模糊推理为模糊拒取式推理。

模糊拒取式推理可直观地表示为

规则：$\text{if}\quad x\quad \text{is}\quad A\quad \text{then}\quad y\quad \text{is}\quad B$

证据：$y\quad \text{is}\quad B'$

结论：$x\quad \text{is}\quad A'$

(3) 模糊三段论推理

设 A、B、C 分别是论域 U、V、W 上的模糊集，若由规则链

$$\text{if}\quad x\quad \text{is}\quad A\quad \text{then}\quad y\quad \text{is}\quad B$$
$$\text{if}\quad y\quad \text{is}\quad B\quad \text{then}\quad z\quad \text{is}\quad C$$

可以推出

$$\text{if}\quad x\quad \text{is}\quad A\quad \text{then}\quad z\quad \text{is}\quad C$$

则称这种模糊推理为模糊三段论推理，或称为模糊顺序推理法则。

2. 简单模糊推理方法

简单模糊推理是指模糊规则的前提条件是单一条件，且规则和证据都不带可信度因子的

模糊推理。

按照扎德等人提出的模糊推理方法，对于规则：

$$\text{if}\quad x\quad \text{is}\quad A\quad \text{then}\quad y\quad \text{is}\quad B$$

首先要构造出模糊集 A 与 B 之间的模糊关系 R，然后通过 R 与证据的合成求出结论。

如果已知证据是 x is A'，且 A 与 A' 可以模糊匹配，则由模糊假言推理得到结论 y is B'，且模糊集 B' 由下述合成运算求出：

$$B'=A'\circ R$$

现在的问题是如何构造模糊关系 R，扎德等人分别提出了多种构造 R 的方法。

(1) 扎德方法

扎德提出了两种构造模糊关系的方法，一种称为条件命题的极大极小规则来构造模糊关系，记为 R_{m}；另一种称为条件命题的算术规则来构造模糊关系，记为 R_{a}。

设 A 和 B 分别是论域 U 和 V 上的模糊集，且

$$A=\int_U \mu_A(u)/u \qquad B=\int_V \mu_B(v)/v$$

R_{m} 与 R_{a} 的定义分别为

$$\begin{aligned}R_{\mathrm{m}}&=(A\times B)\bigcup(\overline{A}\times V)\\&=\int_{U\times V}(\mu_A(u)\wedge\mu_B(v))\vee(1-\mu_A(u))/(u,v)\\R_{\mathrm{a}}&=(\overline{A}\times V)\oplus(U\times B)\\&=\int_{U\times V}1\wedge(1-\mu_A(u)+\mu_B(v))/(u,v)\end{aligned}$$

① 对于模糊假言推理，由 R_{m} 与 R_{a} 分别求得 B'_{m} 与 B'_{a} 分别为

$$B'_{\mathrm{m}}=A'\circ R_{\mathrm{m}}=A'\circ[(A\times B)\bigcup(\overline{A}\times V)]$$

$$B'_{\mathrm{a}}=A'\circ R_{\mathrm{a}}=A'\circ[(\overline{A}\times V)\oplus(U\times B)]$$

它们的隶属函数分别为

$$\mu_{B'_{\mathrm{m}}}(v)=\bigvee_{u\in U}\{\mu_{A'}(u)\wedge[(\mu_A(u)\wedge\mu_B(v))\vee(1-\mu_A(u))]\}$$

$$\mu_{B'_{\mathrm{a}}}(v)=\bigvee_{u\in U}\{\mu_{A'}(u)\wedge[1\wedge(1-\mu_A(u)+\mu_B(v))]\}$$

② 对于模糊拒取式推理，由 R_{m} 与 R_{a} 分别求得 A'_{m} 与 A'_{a} 分别为

$$A'_{\mathrm{m}}=R_{\mathrm{m}}\circ B'=[(A\times B)\bigcup(\overline{A}\times V)]\circ B'$$

$$A'_{\mathrm{a}}=R_{\mathrm{a}}\circ B'=[(\overline{A}\times V)\oplus(U\times B)]\circ B'$$

它们的隶属函数分别为

$$\mu_{A'_{\mathrm{m}}}(u)=\bigvee_{v\in V}\{[\mu_A(u)\wedge\mu_B(v)\vee(1-\mu_A(u))]\wedge\mu_{B'}(v)\}$$

$$\mu_{A'_{\mathrm{a}}}(u)=\bigvee_{v\in V}\{[1\wedge(1-\mu_A(u)+\mu_B(v))]\wedge\mu_{B'}(v)\}$$

例 6.13　设 $U=V=\{1,2,3,4,5\}$

$$A=1/1+0.5/2$$

$$B=0.4/3+0.6/4+1/5$$

且模糊规则为

$$\text{if} \quad x \quad \text{is} \quad A \quad \text{then} \quad y \quad \text{is} \quad B$$

(1) 求模糊集 A 与 B 的模糊关系 R_m 与 R_a。

(2) 若有模糊证据 x is A'，且模糊集 A' 为

$$A'=1/1+0.4/2+0.2/3$$

应用模糊假言推理求结论的模糊集 B'_m 与 B'_a。

(3) 若有模糊证据 y is B'，且模糊集 B' 为

$$B'=0.2/1+0.4/2+0.6/3+0.5/4+0.3/5$$

应用模糊拒取式推理求结论的模糊集 A'_m 与 A'_a。

解 (1) A 与 B 的模糊关系 R_m 与 R_a 的第 i 行第 j 列的元素分别表示为 $R_m(i,j)$ 与 $R_a(i,j)$，且有

$$R_m(i,j)=(\mu_A(u_i)\wedge\mu_B(v_j))\vee(1-\mu_A(u_i))$$

$$R_a(i,j)=1\wedge(1-\mu_A(u_i)+\mu_B(v_j))$$

从而可计算出 R_m 与 R_a 的各元素的值，例如

$$\begin{aligned}R_m(1,3)&=(\mu_A(u_1)\wedge\mu_B(v_3))\vee(1-\mu_A(u_1))\\&=(1\wedge 0.4)\vee(1-1)\\&=0.4\end{aligned}$$

$$\begin{aligned}R_a(2,3)&=1\wedge(1-\mu_A(u_2)+\mu_B(v_3))\\&=1\wedge(1-0.5+0.4)\\&=0.9\end{aligned}$$

由此可得出 R_m 与 R_a 分别为

$$R_m=\begin{bmatrix}0&0&0.4&0.6&1\\0.5&0.5&0.5&0.5&0.5\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

$$R_a=\begin{bmatrix}0&0&0.4&0.6&1\\0.5&0.5&0.9&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

(2) 由模糊假言推理得到结论

$$y \quad \text{is} \quad B'$$

若模糊关系采用 R_m，则结论的模糊集 B' 为 B'_m；若模糊关系采用 R_a，则结论的模糊集 B' 为 B'_a。可分别求得 B'_m 与 B'_a 为

$$B'_m=A'\circ R_m$$

$$= \{1,0.4,0.2,0,0\} \circ \begin{bmatrix} 0 & 0 & 0.4 & 0.6 & 1 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \{0.4,0.4,0.4,0.6,1\}$$

$$B'_a = A' \circ R_a$$

$$= \{1,0.4,0.2,0,0\} \circ \begin{bmatrix} 0 & 0 & 0.4 & 0.6 & 1 \\ 0.5 & 0.5 & 0.9 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \{0.4,0.4,0.4,0.6,1\}$$

即得出结论中的模糊集 B' 分别为

$$B'_m = 0.4/1 + 0.4/2 + 0.4/3 + 0.6/4 + 1/5$$

$$B'_a = 0.4/1 + 0.4/2 + 0.4/3 + 0.6/4 + 1/5$$

这里，$B'_m = B'_a$ 只是巧合，一般来说，它们不一定相同。

(3) 由模糊拒取式推理得到结论

$$x \quad \text{is} \quad A'$$

若模糊关系采用 R_m，则结论的模糊集 A' 为 A'_m；若模糊关系采用 R_a，则结论的模糊集 A' 为 A'_a。可分别求得 A'_m 与 A'_a 为

$$A'_m = R_m \circ B'$$

$$= \begin{bmatrix} 0 & 0 & 0.4 & 0.6 & 1 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \circ \begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \\ 0.5 \\ 0.3 \end{bmatrix}$$

$$= \{0.5,0.5,0.6,0.6,0.6\}$$

$$A'_a = R_a \circ B'$$

$$= \begin{bmatrix} 0 & 0 & 0.4 & 0.6 & 1 \\ 0.5 & 0.5 & 0.9 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \circ \begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \\ 0.5 \\ 0.3 \end{bmatrix}$$

$$= \{0.5,0.6,0.6,0.6,0.6\}$$

即得出结论中的模糊集 A' 分别为

$$A'_m = 0.5/1 + 0.5/2 + 0.6/3 + 0.6/4 + 0.6/5$$

$$A'_a = 0.5/1 + 0.6/2 + 0.6/3 + 0.6/4 + 0.6/5$$

(2) 麦姆德尼方法

麦姆德尼(Mamdani)**提出了称为条件命题的最小运算规则来构造模糊关系,记为** R_c。**模糊关系** R_c**定义为**

$$\begin{aligned} R_c &= A \times B \\ &= \int_{U\times V} \mu_A(u) \wedge \mu_B(v)/(u,v) \end{aligned}$$

① **对于模糊假言推理,由** R_c**求得结论的模糊集** B'_c**为**

$$\begin{aligned} B'_c &= A' \circ R_c \\ &= A' \circ (A \times B) \end{aligned}$$

它的隶属函数为

$$\mu_{B'_c}(v) = \bigvee_{u\in U} [\mu_{A'}(u) \wedge (\mu_A(u) \wedge \mu_B(v))]$$

② **对于模糊拒取式推理,由** R_c**求得结论的模糊集** A'_c**为**

$$\begin{aligned} A'_c &= R_c \circ B' \\ &= (A \times B) \circ B' \end{aligned}$$

它的隶属函数为

$$\mu_{A'_c}(u) = \bigvee_{v\in V} [\mu_A(u) \wedge \mu_B(v) \wedge \mu_{B'}(v)]$$

例 6.14 对例 6.13 给出的数据,应用麦姆德尼方法求推理结论的模糊集。

解 (1) A **与** B **的模糊关系** R_c**的第** i **行第** j **列的元素为**

$$R_c(i,j) = \mu_A(u_i) \wedge \mu_B(v_j)$$

从而可计算出 R_c**各元素的值,得出** R_c**为**

$$R_c = \begin{bmatrix} 0 & 0 & 0.4 & 0.6 & 1 \\ 0 & 0 & 0.4 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(2) **由模糊假言推理得到结论** y is B',**若模糊关系采用** R_c,**则结论的模糊集** B'**为** B'_c,**且** B'_c**为**

$$\begin{aligned} B'_c &= A' \circ R_c \\ &= \{1, 0.4, 0.2, 0, 0\} \circ \begin{bmatrix} 0 & 0 & 0.4 & 0.6 & 1 \\ 0 & 0 & 0.4 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \\ &= \{0, 0, 0.4, 0.6, 1\} \end{aligned}$$

即得出结论中的模糊集 B'**为**

$$B'_c = 0.4/3 + 0.6/4 + 1/5$$

(3) 由模糊拒取式推理得出结论 x is A'，若模糊关系采用 R_c，则结论的模糊集 A' 为 A'_c，且 A'_c 为

$$A'_c = R_c \circ B' = \begin{bmatrix} 0 & 0 & 0.4 & 0.6 & 1 \\ 0 & 0 & 0.4 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \circ \begin{bmatrix} 0.2 \\ 0.4 \\ 0.6 \\ 0.5 \\ 0.3 \end{bmatrix} = \{0.5, 0.5, 0, 0, 0\}$$

即得出结论中的模糊集 A' 为

$$A'_c = 0.5/1 + 0.5/2$$

(3) 米祖莫托方法

米祖莫托(Mizumoto)等人提出了一组构造模糊关系的方法，由此构造的模糊关系分别记为 R_s、R_g、R_{sg}、R_{gg}、R_{gs}、R_{ss}、R_b 等。这里，仅给出 R_s 和 R_g 的定义：

$$R_s = \int_{U \times V} [\mu_A(u) \xrightarrow[s]{} \mu_B(v)] / (u, v)$$

$$R_g = \int_{U \times V} [\mu_A(u) \xrightarrow[g]{} \mu_B(v)] / (u, v)$$

其中，

$$\mu_A(u) \xrightarrow[s]{} \mu_B(v) = \begin{cases} 1 & \mu_A(u) \leqslant \mu_B(v) \\ 0 & \mu_A(u) > \mu_B(v) \end{cases}$$

$$\mu_A(u) \xrightarrow[g]{} \mu_B(v) = \begin{cases} 1 & \mu_A(u) \leqslant \mu_B(v) \\ \mu_B(v) & \mu_A(u) > \mu_B(v) \end{cases}$$

例 6.15　对例 6.13 给出的数据，应用米祖莫托方法求模糊关系 R_s 与 R_g，并分别求出模糊假言推理得出的结论。

解　(1) A 与 B 的模糊关系 R_s 的第 i 行第 j 列的元素为

$$R_s(i,j) = \mu_A(u_i) \xrightarrow[s]{} \mu_B(v_j) = \begin{cases} 1 & \mu_A(u_i) \leqslant \mu_B(v_j) \\ 0 & \mu_A(u_i) > \mu_B(v_j) \end{cases}$$

由给出的模糊集 A 和 B，可得出模糊关系 R_s 为

$$R_s = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

A 与 B 的模糊关系 R_g 的第 i 行第 j 列的元素为

$$R_g(i,j) = \mu_A(u_i) \xrightarrow[g]{} \mu_B(v_j) = \begin{cases} 1 & \mu_A(u_i) \leqslant \mu_B(v_j) \\ \mu_B(v_j) & \mu_A(u_i) > \mu_B(v_j) \end{cases}$$

由给出的模糊集 A 和 B，可得出模糊关系 R_g 为

$$R_g=\begin{bmatrix}0 & 0 & 0.4 & 0.6 & 1\\ 0 & 0 & 0.4 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\end{bmatrix}$$

(2) 由模糊假言推理得到结论 y is B'，若模糊关系采用 R_s，则结论的模糊集 B' 为 B'_s；若模糊关系采用 R_g，则结论的模糊集 B' 为 B'_g，且可分别计算出 B'_s 和 B'_g：

$$\begin{aligned}B'_s&=A'\circ R_s\\&=\{0.2,0.2,0.2,0.4,1\}\\B'_g&=A'\circ R_g\\&=\{0.2,0.2,0.4,0.6,1\}\end{aligned}$$

即可得出结论 y is B' 中的模糊集 B' 分别为

$$B'_s=0.2/1+0.2/2+0.2/3+0.4/4+1/5$$

$$B'_g=0.2/1+0.2/2+0.4/3+0.6/4+1/5$$

由例 6.13～例 6.15 可以看出，对相同的模糊规则及证据使用不同的模糊关系进行推理时，得到的结论一般是不同的。这说明不同的模糊关系使模糊推理的性能表现出一定的差异。通过分析，可以得知：无论是对于模糊假言推理还是对于模糊拒取式推理，R_s 是性能比较好的模糊关系，R_g 和 R_c 次之，R_m 和 R_a 等模糊关系的性能较差。

3. 模糊三段论推理方法

模糊三段论推理又称为模糊顺序推理法则。设有以下模糊规则：

R_1： if x is A then y is B

R_2： if y is B then z is C

其中，A、B、C 分别是论域 U、V、W 上的模糊集。由规则 R_1 给出的模糊集 A 与 B 可构造出定义在 $U\times V$ 上的模糊关系 $R(A,B)$，由规则 R_2 给出的模糊集 B 与 C 可构造出定义在 $V\times W$ 上的模糊关系 $R(B,C)$。若选择的构造模糊关系的方法满足模糊三段论推理，那么可得出 $U\times W$ 上的模糊关系 $R(A,C)$ 为

$$R(A,C)=R(A,B)\circ R(B,C)$$

若已知证据为 x is A'，则可由模糊假言推理得出结论 z is C'，且模糊集 C' 为

$$C'=A'\circ R(A,C)$$

若已知证据为 z is C'，则可由模糊拒取式推理得出结论 x is A'，且模糊集 A' 为

$$A'=R(A,C)\circ C'$$

在前述的各种模糊关系的构成方法中，有一些能满足模糊三段论推理，有一些不能满足模糊三段论推理。通过下面一个例子予以说明。

例 6.16 设

$$U=V=W=\{1,2,3,4,5\}$$

$$A=1/1+0.6/2+0.2/3$$

$$B=0.3/3+0.7/4+1/5$$

$$C=0.09/3+0.49/4+1/5$$

请分别验证构成的模糊关系 R_m 和 R_g 是否满足模糊三段论。

解　给出的模糊集 A、B、C 分别为

$$A=\{1,0.6,0.2,0,0\}$$

$$B=\{0,0,0.2,0.7,1\}$$

$$C=\{0,0,0.09,0.49,1\}$$

① 由模糊关系 R_m 的构成方法，可分别得出

$$R_m(A,B)=\begin{bmatrix}0&0&0.3&0.7&1\\0.4&0.4&0.4&0.6&0.6\\0.8&0.8&0.8&0.8&0.8\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

$$R_m(B,C)=\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\0.7&0.7&0.7&0.7&0.7\\0.3&0.3&0.3&0.49&0.7\\0&0&0.09&0.49&1\end{bmatrix}$$

$$R_m(A,C)=\begin{bmatrix}0&0&0.09&0.49&1\\0.4&0.4&0.4&0.49&0.6\\0.8&0.8&0.8&0.8&0.8\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

将 $R_m(A,B)$ 与 $R_m(B,C)$ 合成，得到

$$R_m(A,B)\circ R_m(B,C)=\begin{bmatrix}0.3&0.3&0.3&0.49&1\\0.4&0.4&0.4&0.49&0.6\\0.8&0.8&0.8&0.8&0.8\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

显然，$R_m(A,B)\circ R_m(B,C)\neq R_m(A,C)$。由此说明 R_m 不满足模糊三段论。

② 由模糊关系 R_g 的构成方法，可分别得出

$$R_g(A,B)=\begin{bmatrix}0&0&0.3&0.7&1\\0&0&0.3&1&1\\0&0&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

$$R_g(B,C)=\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\0&0&0.09&1&1\\0&0&0.09&0.49&1\\0&0&0.09&0.49&1\end{bmatrix}$$

$$R_g(A,C)=\begin{bmatrix}0&0&0.09&0.49&1\\0&0&0.09&0.49&1\\0&0&0.09&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

将 $R_g(A,B)$ 与 $R_g(B,C)$ 合成，得到

$$R_g(A,B)\circ R_g(B,C)=\begin{bmatrix}0&0&0.09&0.49&1\\0&0&0.09&0.49&1\\0&0&0.09&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

显然，$R_g(A,B)\circ R_g(B,C)=R_g(A,C)$。实际上 R_g 满足模糊三段论。

实际上，满足模糊三段论的模糊关系有 R_c、R_s、R_g、R_{sg}、R_{gg}、R_{gs}、R_{ss}；不满足模糊三段论的模糊关系有 R_m，R_a，R_b 等。

6.3.4 多维模糊推理方法

多维模糊推理是指组合条件规则的模糊假言推理，多维模糊推理的一般模式为

规则：if$(x_1$ is $A_1)\wedge(x_2$ is $A_2)\wedge\cdots\wedge(x_n$ is $A_n)$ then y is B

证据：x_1 is A_1' x_2 is A_2' $\cdots$ x_n is A_n'

结论：y is B'

其中，A_i 与 A'_i 是论域 U_i 上的模糊集，$i=1,2,\cdots,n$，B 与 B' 是论域 V 上的模糊集。

多维模糊推理求结论的模糊集 B' 的方法主要有以下几种方法。

1. 扎德方法

扎德方法进行多维模糊推理的步骤如下。

① 求出 $A_1,A_2,\cdots,A_n$ 的交集，记为 A，即

$$A=A_1\cap A_2\cap\cdots\cap A_n=\int_{U_1\times U_2\times\cdots\times U_n}\mu_{A_1}(u_1)\wedge\mu_{A_2}(u_2)\wedge\cdots\wedge\mu_{A_n}(u_n)/(u_1,u_2,\cdots,u_n)$$

② 用一种构造模糊关系的方法构造出 A 与 B 的模糊关系 $R(A,B)$，记为 $R(A_1,A_2,\cdots,A_n,B)$。

③ 求出证据中 $A_1',A_2',\cdots,A_n'$ 的交集，记为 A'，即

$$A'=A_1'\cap A'\cap\cdots\cap A_n'$$

$$=\int_{U_1\times U_2\times\cdots\times U_n}\mu_{A'_1}(u_1)\wedge\mu_{A'_2}(u_2)\wedge\cdots\wedge\mu_{A'_n}(u_n)/(u_1,u_2,\cdots,u_n)$$

④ 根据模糊假言推理，由 A' 与 $R(A,B)$ 的合成求出结论的模糊集 B'，即

$$B'=A'\circ R(A,B)=(A_1'\cap A_2'\cap\cdots\cap A_n')\circ R(A_1,A_2,\cdots,A_n,B)$$

例 6.17　设论域 $U=V=W=\{1,2,3,4,5\}$，规则的组合条件分别在 U 与 V 上有模糊集 $A_1=\{1,0.6,0,0,0\}$ 与 $A_2=\{0,1,0.5,0,0\}$，规则的结论在 W 上有模糊集 $B=\{0,0,1,0.8,0\}$。若已知证据分别在 U 与 V 上有模糊集 $A_1'=\{0.8,0.5,0,0,0\}$ 与 $A_2'=\{0,0.9,0.5,0,0\}$，分别采用 R_a、R_m 和 R_s 求推理结论的模糊集 B'。

解　这是一个二维模糊推理（即 $n=2$）问题，首先分别求出组合条件及证据的交集，即

$$A_1\cap A_2=\{0,0.6,0,0,0\}$$

$$A_1'\cap A_2'=\{0,0.5,0,0,0\}$$

（1）若用 R_a 构造模糊关系，则得到

$$R_a(A_1,A_2,B)=\begin{bmatrix}1&1&1&1&1\\0.4&0.4&1&1&0.4\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

$$B_a'=(A_1'\cap A_2')\circ R_a(A_1,A_2,B)=\{0.4,0.4,0.5,0.5,0.4\}$$

（2）若用 R_m 构造模糊关系，则得到

$$R_m(A_1,A_2,B)=\begin{bmatrix}1&1&1&1&1\\0.4&0.4&0.6&0.6&0.4\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

$$B_m'=(A_1'\cap A_2')\circ R_m(A_1,A_2,B)=\{0.4,0.4,0.5,0.5,0.4\}$$

（3）若用 R_s 构造模糊关系，则得到

$$R_s(A_1,A_2,B)=\begin{bmatrix}1&1&1&1&1\\0&0&1&1&0\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

$$B_s'=(A_1'\cap A_2')\circ R_s(A_1,A_2,B)=\{0,0,0.5,0.5,0\}$$

2. 祖卡莫托方法

祖卡莫托（Zukamoto）方法进行多维模糊推理的步骤如下。

① 用一种构造模糊关系的方法对组合条件的每个单一条件 A_i 与 B 构造模糊关系

$$R(A_i,B)\quad i=1,2,\cdots,n$$

② 对每个单一证据 A_i' 求出简单模糊推理的结论 B_i'，即

$$B_i'=A_i'\circ R(A_i,B)\quad i=1,2,\cdots,n$$

③ 求出所有 B_i' 的交集，从而得到 B'，即

$$B'=B_1'\cap B_2'\cap\cdots\cap B_n'$$

例 6.18 对例 6.17 给出的数据，若采用 R_s 构造模糊关系，试用祖卡莫托方法求推理结论的模糊集 B'。

解 由给出的 A_1、A_2 和 B，按 R_s 构造模糊关系，得出

$$R_s(A_1,B)=\begin{bmatrix}0&0&1&0&0\\0&0&1&1&0\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

$$R_s(A_2,B)=\begin{bmatrix}1&1&1&1&1\\0&0&1&0&0\\0&0&1&1&0\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

分别由给出的证据 A_1' 和 A_2'，按简单模糊推理，得出

$$B_{s1}'=A_1'\circ R_s(A_1,B)=\{0,0,0.8,0.5,0\}$$

$$B_{s2}'=A_2'\circ R_s(A_2,B)=\{0,0,0.9,0.5,0\}$$

最后得出二维模糊推理结论的模糊集 B_s' 为

$$B_s'=B_{s1}'\cap B_{s2}'=\{0,0,0.8,0.5,0\}$$

3. 苏更诺方法

苏更诺(Sugeno)方法采取逐维递推计算方式求出多维模糊推理的结论的模糊集，即

$$B_1'=A_1'\circ R(A_1,B)$$

$$B_2'=A_2'\circ R(A_2,B_1')$$

$$\vdots$$

$$B'=B_n'=A_n'\circ R(A_n,B_{n-1}')$$

例 6.19 对例 6.17 给出的数据，若采用 R_s 构造模糊关系，试用苏更诺方法求解推理结论的模糊集 B'。

解 首先由给出的 A_1 和 B，按 R_s 构造模糊关系，并计算 B_{s1}'，即

$$R_s(A_1,B)=\begin{bmatrix}0&0&1&0&0\\0&0&1&1&0\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

$$B'_{s1}=A'_1\circ R_s(A_1,B)=\{0,0,0.8,0.5,0\}$$

然后由给出的 A_2 和得到的 B'_{s1}，按 R_s 构造模糊关系，即

$$R_s(A_2,B'_{s1})=\begin{bmatrix}1&1&1&1&1\\0&0&0&0&0\\0&0&1&1&0\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

最后由给出的 A'_2 和得到的 $R_s(A_2,B'_{s1})$ 的合成，得出 B'_s，即

$$B'_s=B'_{s2}=A'_2\circ R_s(A_2,B'_{s1})=\{0,0,0.5,0.5,0\}$$

6.3.5　带有可信度的模糊推理方法

随机性与模糊性是知识的两种主要的不确定性，我们已分别在可信度的不确定推理和模糊推理中讨论了这两种不确定性知识的表示方法和处理方法。但是，现实世界中的许多事物不仅具有模糊性，而且具有随机性，这就要求把这两种方法结合起来，既能表示和处理模糊性，又能表示和处理随机性。带有可信度的模糊推理就是用来解决此类问题的一种方法。在这种方法中，由随机性引起的不确定性用可信度 CF 表示，由模糊性引起的不确定性仍用模糊集表示。

1. 带有可信度的简单模糊推理

带有可信度的简单模糊推理的推理模式为

规则：if　x　is　A　then　y　is　B　CF_1

证据：　　x　is　A'　　CF_2

结论：　　y　is　B'　CF

其中，CF_1 与 CF_2 分别是给出的规则和证据的可信度，CF 是待求的结论的可信度。

由上述推理模式可以看出，带有可信度的模糊推理需要解决两个问题：一是如何推出结论 y　is　B'，二是如何求出结论的可信度 CF。实际上，解决第一个问题的方法就是不带可信度的模糊推理方法，所以，问题在于如何在推理过程中求出结论的可信度 CF。下面分两种情况讨论。

(1) 条件模糊集与证据模糊集相等

若条件模糊集 A 与证据模糊集 A' 相等，即 $A=A'$，则结论的可信度 CF 可分别用下述三种方法来计算：

① $\mathrm{CF}=\mathrm{CF}_1\times\mathrm{CF}_2$

② $\mathrm{CF}=\min\{\mathrm{CF}_1,\mathrm{CF}_2\}$

③ $\mathrm{CF}=\max\{0,\mathrm{CF}_1+\mathrm{CF}_2-1\}$

(2) 条件模糊集与证据模糊集不等

若条件模糊集 A 与证据模糊集 A' 不等，即 $A\neq A'$，但 A 与 A' 可模糊匹配，即 A 与 A' 的匹

配度 $\delta(A,A')\geqslant\lambda$，$\lambda$ 是规则的阈值，则结论的可信度 CF 可分别用下述四种方法来计算：

① $\mathrm{CF}=\delta(A,A')\times\mathrm{CF}_1\times\mathrm{CF}_2$

② $\mathrm{CF}=\delta(A,A')\times\min\{\mathrm{CF}_1,\mathrm{CF}_2\}$

③ $\mathrm{CF}=\delta(A,A')\times\max\{0,\mathrm{CF}_1+\mathrm{CF}_2-1\}$

④ $\mathrm{CF}=\min\{\delta(A,A'),\mathrm{CF}_1,\mathrm{CF}_2\}$

2. 带有可信度的多维模糊推理

带有可信度的多维模糊推理的推理模式为

规则：if $(x_1$ is $A_1)\wedge(x_2$ is $A_2)\wedge\cdots\wedge(x_n$ is $A_n)$ then y is B CF_E

证据：x_1 is A_1' CF_1

x_2 is A_2' CF_2

$\vdots$

x_n is A_n' CF_n

结论：y is B' CF

求多维模糊推理的结论可信度 CF 的步骤如下。

首先按照 6.3.2 节介绍的组合条件与多个证据模糊匹配的方法，求得组合条件与多个证据的总匹配度 $\delta(E,E')$，总匹配度的计算方法是：

$$\delta(E,E')=\min\{\delta(A_i,A_i')\mid i=1,2,\cdots,n\}$$

或者

$$\delta(E,E')=\delta(A_1,A_1')\times\delta(A_2,A_2')\times\cdots\times\delta(A_n,A_n')$$

若 $\delta(E,E')\geqslant\lambda$，则规则的前提条件与证据可匹配；否则，不可匹配。

若规则的组合条件与证据是可匹配的，则计算证据的总可信度 $\mathrm{CF}_{E'}$。证据的总可信度的计算方法是

$$\mathrm{CF}_{E'}=\min\{\mathrm{CF}_i\mid i=1,2,\cdots,n\}$$

或者

$$\mathrm{CF}_{E'}=\mathrm{CF}_1\times\mathrm{CF}_2\times\cdots\times\mathrm{CF}_n$$

总匹配度与总可信度求出后，多维模糊推理的结论可信度 CF 的计算方法就可等同于简单模糊推理中的计算方法，可分别用下述四种方法来计算：

① $\mathrm{CF}=\delta(E,E')\times\mathrm{CF}_E\times\mathrm{CF}_{E'}$

② $\mathrm{CF}=\delta(E,E')\times\min\{\mathrm{CF}_E,\mathrm{CF}_{E'}\}$

③ $\mathrm{CF}=\delta(E,E')\times\max\{0,\mathrm{CF}_E+\mathrm{CF}_{E'}-1\}$

④ $\mathrm{CF}=\min\{\delta(E,E'),\mathrm{CF}_E,\mathrm{CF}_{E'}\}$

3. 带有可信度的并行模糊推理

设有当前证据经过两条独立的模糊推理链分别推出了如下的两个结论：

y is B_1' CF_1

y is B_2' CF_2

则可用下述方法计算出当前证据共同支持的结论 y is B' 中的模糊集 B' 及结论的可信度 CF：

$$B' = B'_1 \cap B'_2$$

$$\mathrm{CF} = \mathrm{CF}_1 + \mathrm{CF}_2 - \mathrm{CF}_1 \times \mathrm{CF}_2$$

习　题　六

6.1　证据的不确定性主要反映在哪些方面？规则的不确定性主要反映在哪些方面？

6.2　何谓推理的不确定性？请简述不确定推理的不确定性测试的三种基本计算模式。

6.3　设有下述 5 条规则及其规则的可信度为

R_1：if e_1 then h $\mathrm{CF}_1 = 0.8$

R_2：if e_2 then h $\mathrm{CF}_2 = 0.6$

R_3：if e_3 then h $\mathrm{CF}_3 = -0.5$

R_4：if $e_4 \wedge (e_5 \vee e_6)$ then e_1 $\mathrm{CF}_4 = 0.7$

R_5：if $e_7 \wedge e_8$ then e_3 $\mathrm{CF}_5 = 0.9$

且已知有关证据的可信度分别为

$\mathrm{CF}(e_2) = 0.8$，$\mathrm{CF}(e_4) = 0.5$，$\mathrm{CF}(e_5) = 0.6$

$\mathrm{CF}(e_6) = 0.7$，$\mathrm{CF}(e_7) = 0.6$，$\mathrm{CF}(e_8) = 0.9$

求结论 h 的可信度 $\mathrm{CF}(h)$。

6.4　设有下述 3 条带有阈限的推理规则：

R_1：if $e_3 \wedge e_4 \wedge e_5$ then e_1 $(0.6, 0.5)$

R_2：if e_1 then h $(0.7, 0.3)$

R_3：if e_2 then h $(0.8, 0.6)$

且已知有关证据的可信度分别为

$\mathrm{CF}(e_2) = 0.9$，$\mathrm{CF}(e_3) = 0.8$，$\mathrm{CF}(e_4) = 0.6$，$\mathrm{CF}(e_5) = 0.7$

试分别应用求极大值法、加权求和法、有限和法求结论 h 的可信度 $\mathrm{CF}(h)$。

6.5　设有下述带有加权因子的 3 条推理规则：

R_1：if $e_1(0.7) \wedge e_2(0.3)$ then h_1 $(0.9, 0.6)$

R_2：if $e_3(0.4) \wedge e_4(0.3) \wedge e_5(0.3)$ then h_2 $(0.8, 0.5)$

R_3：if $e_6(0.3) \wedge h_1(0.5) \wedge h_2(0.2)$ then h $(0.6, 0.2)$

且已知有关证据的可信度分别为

$\mathrm{CF}(e_1) = 0.9$，$\mathrm{CF}(e_2) = 0.85$，$\mathrm{CF}(e_3) = 0.85$

$\mathrm{CF}(e_4) = 0.7$，$\mathrm{CF}(e_5) = 0.75$，$\mathrm{CF}(e_6) = 0.9$

请应用加权的不确定推理方法求结论 h 的可信度 $\mathrm{CF}(h)$。

6.6　设有论域 $U = \{x_1, x_2, x_3, x_4, x_5\}$，$A$ 与 B 是 U 上的两个模糊集，且有

$A = 0.6/x_1 + 0.4/x_2 + 0.3/x_3 + 0.2/x_4 + 0.1/x_5$

$B = 0.8/x_3 + 0.6/x_4 + 1/x_5$

请分别计算 $A \cap B$，$A \cup B$，$\overline{A}$。

6.7　学生域 $U = \{u_1, u_2, u_3\}$ 中的 3 名学生 u_1、u_2 和 u_3 分别对球类运动域 $V = \{v_1, v_2, v_3, v_4\}$ 中的 4 种球类运动有不同程度的爱好。若学生 u_i 最爱好某项球类运动 v_j，则打分为“1”；若最不爱好 v_j，则打分为“0”。

请按照学生对球类运动爱好的模糊语言设计一个打分表，并由此给出 U 与 V 的模糊关系。

6.8 设有如下两个模糊关系：

$$R_1=\begin{bmatrix}0.4 & 0.5 & 0.1\\ 0.2 & 0.6 & 0.2\\ 0.5 & 0.3 & 0.2\end{bmatrix},\quad R_2=\begin{bmatrix}0.2 & 0.8\\ 0.4 & 0.6\\ 0.6 & 0.4\end{bmatrix}$$

求 R_1 与 R_2 的合成 $R_1\circ R_2$。

6.9 若对食品的评价标准是：色(u_1)、香(u_2)、味(u_3)，它们组成评价标准论域 $U=\{u_1,u_2,u_3\}$。由若干名评委对食品按色、香、味标准评判打分，打分的等级是：好(v_1)、较好(v_2)、一般(v_3)和差(v_4)。它们组成等级论域 $V=\{v_1,v_2,v_3,v_4\}$，v_i 的值为选择该等级的评委百分比(若 30% 的评委认为是好，则 $v_i=0.3$)。若对某种食品评判的结果如下：

对“色”的评价：$\{0.6,0.2,0.2,0\}$

对“香”的评价：$\{0.8,0.1,0.1,0\}$

对“味”的评价：$\{0.3,0.3,0.3,0.1\}$

且根据该食品的特点和评委的一致意见，对该食品的评价标准增设了权重，“色”的权重为 0.3，“香”的权重为 0.3，“味”的权重为 0.4。应用模糊变换，求出对该食品的综合评价，并对得出的综合评价予以解释。

6.10 设有下述带阈值的模糊规则与证据：

$$\text{if}\quad x\quad \text{is}\quad A\quad \text{then}\quad y\quad \text{is}\quad B\quad (0.7)$$
$$x\quad \text{is}\quad A'$$

且 A 与 A' 分别是论域 $U=\{u_1,u_2,u_3\}$ 上的模糊集：

$$A=0.3/u_1+0.5/u_2+0.2/u_3$$
$$A'=0.5/u_1+0.8/u_2+0.4/u_3$$

(1) 分别求 A 与 A' 的贴近度，海明距离，欧几里德距离和切比雪夫距离。

(2) 匹配度采用哪种方式时，规则与证据可匹配？

6.11 由 5 名篮球运动员组成论域 $U=V=\{a,b,c,d,e\}$，设有下述模糊规则：

$$\text{if}\quad x\quad \text{is}\quad \text{球技较好}\quad \text{then}\quad y\quad \text{is}\quad \text{得分较多}$$

$x\in U,y\in V$。根据不完全记载，得出对这 5 名运动员的模糊评价为

球技较好 $A=1/a+0.5/b$

得分较多 $B=0.4/c+0.6/d+/e$

(1) 分别构造出 A 与 B 的模糊关系 $R_m(A,B)$ 与 $R_a(A,B)$。

(2) 根据一场球赛的表现，若得出运动员的球技水平的模糊评价为

$$A'=1/a+0.4/b+0.2/c$$

应用模糊假言推理，求对 5 名运动员得分能力的模糊评价。

(3) 根据一场球赛的表现，若得出运动员得分能力的模糊评价为

$$B'=0.2/a+0.4/b+0.6/c+0.5/d+0.3/e$$

应用模糊拒取式推理，求对 5 名运动员球技水平的模糊评价。

6.12 对 6.11 题，应用麦姆德尼方法构造模糊关系 $R_c(A,B)$，并求出模糊假言推理与模糊拒取式推理的模糊结论。

6.13 对 6.11 题，应用米祖莫托方法构造模糊关系 $R_s(A,B)$ 与 $R_g(A,B)$，并分别由 $R_s(A,B)$ 与 $R_g(A,B)$ 求出模糊假言推理与模糊拒取式推理的模糊结论。

6.14 对 6.11 题，若规则是带有可信度和阈值的模糊规则，即

$$\text{if}\quad x\quad \text{is}\quad \text{球技较好}\quad \text{then}\quad y\quad \text{is}\quad \text{得分较多}\quad \mathrm{CF}_1=0.7\quad \lambda=0.8$$

对论域 $U=V=\{a,b,c,d,e\}$ 中运动员的模糊评价为

球技较好　　$A=1/a+0.5/b$

得分较多　　$B=0.4/c+0.6/d+/e$

且根据一场球赛得出运动员的球技水平的模糊评价为

$$A'=1/a+0.4/b+0.2/c \quad \mathrm{CF}_2=0.75$$

(1) 计算 A 与 A' 的贴近度，并由此确定规则与证据是否可匹配？

(2) 若规则与证据可匹配，则构造模糊关系 $R_{\mathrm{m}}(A,B)$，应用模糊假言推理求出模糊结论，并应用 4 种方法分别计算出结论的可信度。

6.15　设有论域

$$U=V=W=\{1,2,3,4,5,6\}$$

且设有下述模糊规则：

R_1：　if　x　is　A　then　y　is　B

R_2：　if　x　is　B　then　z　is　C

R_3：　if　x　is　A　then　z　is　C

其中，A,B,C 的模糊集分别为

$$A=1/1+0.8/2+0.5/3+0.4/4+0.1/5$$

$$B=0.1/2+0.2/3+0.4/4+0.6/5+0.8/6$$

$$C=0.2/3+0.5/4+0.8/5+1/6$$

请分别采用 R_{m}、R_{c}、R_{g} 构造模糊关系，并验证其是否满足模糊三段论。

6.16　设有论域

$$U=V=W=\{1,2,3,4,5\}$$

且设有下述模糊规则：

if　$(x_1$　is　$A_1)\wedge(x_2$　is　$A_2)$　then　y　is　B

已知事实为

x_1　is　A_1'

x_2　is　A_2'

其中，A_1、A_2、B、A_1' 及 A_2' 的模糊集分别为

$$A_1=1/1+0.8/2+0.6/3+0.4/4$$

$$A_2=0.2/1+0.4/2+0.6/3+0.8/4+1/5$$

$$B=0.5/3+0.7/4$$

$$A_1'=0.8/1+0.7/2+0.5/3+0.3/4$$

$$A_2'=0.2/1+0.3/2+0.5/3+0.7/4+\ 0.9/5$$

请分别用 R_{a} 与 R_{s} 构造模糊关系，并用扎德方法求出 B_{a}' 和 B_{s}'。

6.17　对 6.16 题，请用祖卡莫托方法求出 B_{s}'。

6.18　对 6.16 题，请用苏更诺方法求出 B_{s}'。

第 7 章

机 器 学 习

20 世纪 80 年代以来，机器学习的理论、方法与技术发展很快，已成为人工智能的一个十分重要的研究领域。主要的机器学习方法是归纳学习方法。遗传算法的研究和人工神经网络的兴起，都给机器学习的研究与应用领域带来强有力的生命力，并对人工智能的其他研究与应用领域产生了很大的影响。

7.1 机器学习的概念与方法

机器学习过程是智能系统不断地积累和更新知识以改善系统性能，从而使系统具有自适应能力的过程。具有学习能力的系统对于环境的变化，系统能随之产生适应性变化，从而使系统能更有效地完成同样或类似的任务。在系统内部，机器学习表现为新知识结构的建立和完善及知识的更新。

1. 机器学习系统的主要特征

一个机器学习系统通常应该具有如下主要特征。

(1) 目的性

系统的学习行为有高度的目的性，即系统必须知道学习什么。

(2) 结构性

系统必须具备适当的知识存储结构来记忆学到的知识，能够修改和完善知识表示与知识的组织形式。

(3) 有效性

系统学习到的知识应受到实践的检验，新知识必须对改善系统的行为起到有益的作用。

(4) 开放性

系统的能力应在实际使用过程中和在同环境进行信息交互的过程中不断改进。

2. 机器学习的基本方法

机器学习的方法很多，而且，机器学习的研究正处在发展的高峰时期，各种新思想、新方法和新技术不断涌现，很难对它们进行系统的分类，这里，仅列举下述一些比较重要的学习方法。

(1) 归纳学习

归纳学习(Inductive Learning)是由环境提供一系列正例和反例，通过归纳推理，机器将这些例子进行推广，产生一个或一组一般的概念描述。

(2) 遗传算法

遗传算法(Genetic Algorithm)是借鉴生物遗传机制的一种随机化非线性计算算法。它通过对对象系统第一代群体及其后代群体中的个体不断地选优汰劣与随机性遗传变异来获得对象系统的一个非线性映射模型。这个映射模型就是采用遗传算法对对象系统的第一代群体学习的结果,也就是对象系统的知识表示。

(3) 人工神经网络

人工神经网络(Artificial Neural Networks)由一些类似神经元的单元及单元间带权的连接弧构成,其中每个单元具有一个状态。通过各类实例(样本)的反复训练,人工神经网络不断调整各连接弧上的权值及神经元的内部状态,当神经网络达到一定的稳定状态后,神经网络就能恰当地反映网络输入模式对输出模式的映射关系,从而达到学习的目的。一个对象系统通过实例训练获得一个稳定权值分布的神经网络,就是这个对象系统的知识表示。

7.2　归 纳 学 习

归纳学习是从特例推导一般规则的学习方法。人类知识的增长主要得益于归纳学习,例如,通过观察"燕子会飞"、"麻雀会飞"等特殊事实,可以归纳得到"鸟会飞"这样的一般结论。

在机器学习领域中,可把归纳学习形式化地描述为使用训练实例以导出一般规则的搜索问题。

7.2.1　CLS 归纳学习方法

Hunt 提出的概念学习系统 CLS 是一种基于决策树的归纳学习系统。CLS 算法不仅能方便地表示概念的"属性-值"信息的结构,而且能从大量实例中有效地生成相应的决策树模型。1979 年,J. R. Quinlan 在 CLS 算法基础上进行了发展,提出了 ID3 算法。

1. CLS 决策树构造算法

在 CLS 算法生成的决策树中,节点对应于待分类对象的属性,由某一节点引出的弧对应于这个属性的取值,叶节点对应于分类的结果。

设分类对象的属性表为 AttrList$=(A_1,A_2,\cdots,A_n)$,每一个属性 A_i 的值域记为 Value(A_i),值域可以是离散的,也可以是连续的。需要分类的分类结果组成分类结果表 Class$=\{C_1,C_2,\cdots,C_m\}$。一般应有 $n\geqslant 1,m\geqslant 2$。训练实例集 $T=\{<X,C>\}$ 中的一个元素就是一个训练实例$<X,C>$,训练实例$<X,C>$的特征向量为 $X=(a_1,a_2,\cdots,a_n)$,其中 a_i 为这个实例的第 i 个属性 A_i 的取值,C 为这个实例的分类结果,$C\in$ Class。可递归地描述下述决策树的构造算法 CLS。

CLS 算法

(1) 初始化:

输入初始学习实例，组成初始学习实例集合 T；

输入分类属性，组成初始属性表 $\mathrm{AttrList}=(A_1,A_2,\cdots,A_i,\cdots,A_n)$；

输入各属性的值，组成各属性的值域表

$$\mathrm{Value}(A_1),\mathrm{Value}(A_2),\cdots,\mathrm{Value}(A_i),\cdots,\mathrm{Value}(A_n)$$

(2) 若所有的叶节点都是终叶节点，即算法成功终止；否则，转步骤(3)。

(3) 若当前属性表 $\mathrm{AttrList}=(\ )$，则算法失败终止；否则，转步骤(4)。

(4) 取当前属性表第 1 个属性 A_i，逐一划分所有非终叶节点中的实例集，并用属性 A_i 标记划分的非终叶节点。

(5) 若属性 A_i 的值域表 $\mathrm{Value}(A_i)$ 有 s 个值，则将一个非终叶节点中的实例集划分为 s 个子集，同一个子集节点中的实例属性 A_i 有相同的属性值，并用属性值标记与子集节点的连接弧。

若一个子集中的所有实例的分类结果均为 C_j，则该子集节点称为终叶节点，且用分类结果 C_j 标记该节点；若一个子集为空集，则该子集节点也称为终叶节点，且予以删除，并删除相应的连接弧。

(6) 删除当前属性表中的第 1 个属性，转步骤(2)。

2. CLS 算法的应用

例 7.1 根据人员的外貌特征属性进行分类，3 个分类特征属性分别是身高、头发颜色、眼睛颜色；分类结果是 P 和 N。

特征属性的值域分别是

Value(发色)={亚麻色，棕色，黑色}

Value(身高)={矮，高}

Value(眼色)={蓝色，黑色}

初始属性表为 AttrList=(身高，发色，眼色)。

现有 8 个实例组成学习实例集 T，如表 7.1 所示。

表 7.1 例 7.1 的学习实例集

序号	身高	发色	眼色	类别	序号	身高	发色	眼色	类别
1	矮	亚麻色	蓝色	P	5	高	黑色	蓝色	N
2	高	亚麻色	黑色	N	6	高	亚麻色	蓝色	P
3	高	棕色	蓝色	P	7	高	黑色	黑色	N
4	矮	黑色	蓝色	N	8	矮	亚麻色	黑色	N

(1) 应用 CLS 算法生成人员分类决策树。

(2) 由决策树得出产生式规则集。

(3) 对规则集中的规则进行合并，给出没有冗余的规则集。

解 (1) 由表 7.1 给出的学习实例集可得出初始实例集 T 如下(T 中只表示了实例的序号及用括号括起来的分类类别)：

$T=(1(P),2(N),3(P),4(N),5(N),6(P),7(N),8(N))$

初始属性表　AttrList=(**身高**,**发色**,**眼色**)

Loop1　**取属性表第 1 个属性“身高”划分 T 为 2 个子集。**

身高=“矮”的子集 $T_1=(1(P),4(N),8(N))$

身高=“高”的子集 $T_2=(2(N),3(P),5(N),6(P),7(N))$

删除属性表中的第 1 个属性,使 AttrList=(**发色**,**眼色**)

Loop2　**取属性表中第 1 个属性“发色”划分** T_1 **和** T_2**,两个子循环分别是**

Loop21　**属性“发色”划分** T_1 **为 3 个子集:**

发色=“亚麻色”的子集 $T_{11}=(1(P),8(N))$

发色=“棕色”的子集 $T_{12}=()$　　**终叶节点,删除**

发色=“黑色”的子集 $T_{13}=(4(N))$　　**终叶节点,标记为** N

Loop22　**属性“发色”划分** T_2 **为 3 个子集:**

发色=“亚麻色”的子集 $T_{21}=(2(N),6(P))$

发色=“棕色”的子集 $T_{22}=(3(P))$　　**终叶节点,标记为** P

发色=“黑色”的子集 $T_{23}=(5(N),7(N))$　　**终叶节点,标记为** N

删除属性表中第 1 个属性,使 AttrList=(**眼色**)

Loop3　**取属性表中第 1 个属性“眼色”划分** T_{11} **和** T_{21}**,两个子循环分别是**

Loop31　**属性“眼色”划分** T_{11} **为 2 个子集:**

眼色=“蓝色”的子集 $T_{111}=(1(P))$　　**终叶节点,标记为** P

发色=“黑色”的子集 $T_{112}=(8(N))$　　**终叶节点,标记为** N

Loop32　**属性“眼色”划分** T_{21} **为 2 个子集:**

发色=“蓝色”的子集 $T_{211}=(6(P))$　　**终叶节点,标记为** P

发色=“黑色”的子集 $T_{212}=(2(N))$　　**终叶节点,标记为** N

删除属性表中第 1 个属性,使 AttrList=( )

Loop4　**所有叶节点都是终叶节点,故算法成功终止。**

由 CLS **算法生成的决策树如图** 7.1 **所示。**

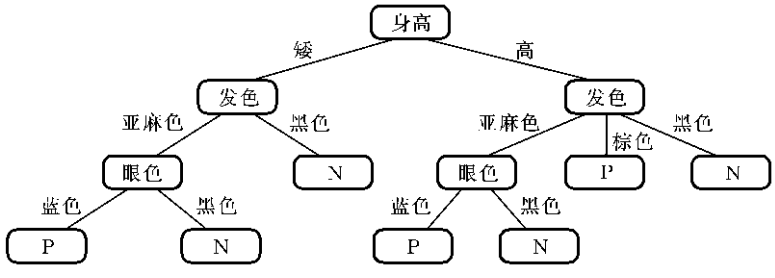

图 7.1　例 7.1 由 CLS 算法生成的决策树

(2) **由决策树得出产生式规则集**

R_1:(**身高=“矮”**)∧(**发色=“亚麻色”**)∧(**眼色=“蓝色”**)→**类别=“P”**

R_2:(**身高=“矮”**)∧(**发色=“亚麻色”**)∧(**眼色=“黑色”**)→**类别=“N”**

R_3：(身高＝“矮”)∧(发色＝“黑色”)→类别＝“N”

R_4：(身高＝“高”)∧(发色＝“亚麻色”)∧(眼色＝“蓝色”)→类别＝“P”

R_5：(身高＝“高”)∧(发色＝“亚麻色”)∧(眼色＝“黑色”)→类别＝“N”

R_6：(身高＝“高”)∧(发色＝“棕色”)→类别＝“P”

R_7：(身高＝“高”)∧(发色＝“黑色”)→类别＝“N”

(3) 对规则集中的规则合并

对后件相同的规则，可考察前件包含的条件，合并成新规则来代替原有规则。

R_3与 R_7可合并为：发色＝“黑色”→类别＝“N”

R_2与 R_5可合并为：(发色＝“亚麻色”)∧(眼色＝“黑色”)→类别＝“N”

R_1与 R_4可合并为：(发色＝“亚麻色”)∧(眼色＝“蓝色”)→类别＝“P”

R_6：(身高＝“高”)∧(发色＝“棕色”)→类别＝“P”

7.2.2 ID3 算法

CLS 决策树构造算法每次从属性表 AttrList 中取第一个属性 A_i来扩展生成决策树，按照训练实例集 T 中所有实例的属性 A_i的取值来划分实例子集，因此，CLS 算法的开销较大，效率较低。Quinlan 提出了 ID3 算法，对属性的选择给出一种启发式搜索方法，通过计算实例集的熵来选择平均信息量最小的属性来划分实例集，扩展生成决策树。

若从属性表 $\text{AttrList}=(A_1,A_2,\cdots,A_i,\cdots,A_n)$ 中任选属性 A_i，则由 A_i的值域 $\text{Value}(A_i)=\{V_1,\cdots,V_x,\cdots,V_s\}$的 s 个取值把实例集 T_k中的 e_k个实例分为 s 个子集：$T_{k1}^{(i)}$，$T_{k2}^{(i)},\cdots,T_{kx}^{(i)},\cdots,T_{ks}^{(i)}$。子集 $T_{kx}^{(i)}$中的所有实例的属性 A_i的取值为 V_x。

T_k中的实例的分类结果组成分类结果表 $\text{Class}=\{C_1,C_2,\cdots,C_j,\cdots,C_m\}$，若子集 $T_{kx}^{(i)}$的实例数为 e_{kx}，其中，分类结果为 C_j的实例数为 $e_{kx}(C_j)$，$1\leqslant j\leqslant m$，且 $\sum_{j=1}^{m}e_{kx}(C_j)=e_{kx}$，则实例集 $T_{kx}^{(i)}$实例分类结果为 C_j的概率为 $P_j=e_{kx}(C_j)/e_{kx}$。

定义训练实例集 $T_{kx}^{(i)}$的熵为

$$I(T_{kx}^{(i)})=-\sum_{j=1}^{m}P_j\log_2 P_j=-\sum_{j=1}^{m}\frac{e_{kx}(C_j)}{e_{kx}}\log_2\frac{e_{kx}(C_j)}{e_{kx}}$$

若选择属性 A_i将训练实例集 T_k分为 s 个子集后，可以由各子集的熵来表示实例集 T 的实例平均信息量

$$I(T_k,A_i)=\sum_{x=1}^{s}\frac{e_{kx}}{e_k}I(T_{kx}^{(i)})$$

选择属性将实例集 T_k划分子集的启发式规则为：选择的属性 A_i把实例集 T_k划分为若干子集，应使实例集 T_k的实例平均信息量最小，即有

$$\min\{I(T_k,A_i)\mid A_i\in\text{AttrList}\}$$

也就是说启发式规则要求选择使实例平均信息量最小的属性 A_i来划分实例集。

ID3 算法在 CLS 算法的基础上引入了启发函数 $I(T_k,A_i)$来生成决策树。

ID3 算法

(1) 初始化：

输入初始学习实例，组成初始学习实例集合 T；

输入分类属性，组成初始属性表 $\mathrm{AttrList}=(A_1,A_2,\cdots,A_i,\cdots,A_n)$；

输入各属性的值，组成各属性的值域表

$$\mathrm{Value}(A_1),\mathrm{Value}(A_2),\cdots,\mathrm{Value}(A_i),\cdots,\mathrm{Value}(A_n)$$

(2) 若所有的叶节点都是终叶节点，即算法成功终止；否则，转步骤(3)。

(3) 若当前属性表 $\mathrm{AttrList}=(\ )$，则算法失败终止；否则，转步骤(4)。

(4) 从当前属性表中选择一个属性 A_i，使得用 A_i 划分非终叶节点实例集 T_k 的平均信息量 $I(T_k,A_i)$ 最小，并用属性 A_i 标记该划分的非终叶节点。

① 若属性 A_i 的值域表 $\mathrm{Value}(A_i)$ 有 s 个值，则将非终叶节点实例集 T_k 中的 e_k 个实例划分为 s 个子集：$T_{k1}^{(i)},T_{k2}^{(i)},\cdots,T_{kx}^{(i)},\cdots,T_{ks}^{(i)}$。同一个子集节点中的实例属性 A_i 有相同的属性值。

② 计算每个子集的熵：

$$I(T_{kx}^{(i)})=-\sum_{j=1}^{m}\frac{e_{kx}(C_j)}{e_{kx}}\log_2\frac{e_{kx}(C_j)}{e_{kx}}$$

其中，e_{kx} 是子集 $T_{kx}^{(i)}$ 的实例数，$e_{kx}(C_j)$ 是其中分类结果为 C_j 的实例数。

③ 计算用属性 A_i 划分 T_k 的平均信息量：

$$I(T_k,A_i)=\sum_{x=1}^{s}\frac{e_{kx}}{e_k}I(T_{kx}^{(i)})$$

④ 比较当前属性表中的不同属性划分 T_k 的平均信息量，取使

$$\min\{I(T_k,A_i)\}$$

的属性 A_i 来实际划分实例集 T_k，并用 A_i 的属性值标记与子集节点的连接弧。

⑤ 若一个子集中的所有实例的分类结果均为 C_j，则该子集节点称为终叶节点，且用分类结果 C_j 标记该节点；若一个子集为空集，则该子集节点也称为终叶节点，且予以删除，并删除相应的连接弧。

(5) 删除当前属性表中已用于实际划分实例集 T_k 的属性 A_i，转步骤(2)。

例 7.2　应用 ID3 算法重做例 7.1，初始属性表为 $\mathrm{AttrList}=$(身高，发色，眼色)。

(1) 应用 ID3 算法生成人员分类决策树。

(2) 由决策树得出产生式规则集。

解　(1)由表 7.1 给出的学习实例集可得出初始实例集 T 如下：

$$T=(1(\mathrm{P}),2(\mathrm{N}),3(\mathrm{P}),4(\mathrm{N}),5(\mathrm{N}),6(\mathrm{P}),7(\mathrm{N}),8(\mathrm{N}))$$

初始属性表 $\mathrm{AttrList}=$(身高，发色，眼色)

Loop1　选择属性划分 T，T 的实例数 $e=8$。当前属性表中有如下 3 个属性，3 个子循环。

Loop11　取属性表中第 1 个属性“身高”划分 T 为 2 个子集。

① 身高=“矮”的子集 $T_1^{(1)}=(1(\mathrm{P}),4(\mathrm{N}),8(\mathrm{N}))$，实例数 $e_1=3$，其中 $e_1(\mathrm{P})=1$，$e_1(\mathrm{N})=2$，计算 $T_1^{(1)}$ 的熵：

$$I(T_1^{(1)})=-\left(\frac{e_1(\mathrm{P})}{e_1}\log_2\frac{e_1(\mathrm{P})}{e_1}+\frac{e_1(\mathrm{N})}{e_1}\log_2\frac{e_1(\mathrm{N})}{e_1}\right)=-\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right)=0.918$$

② **身高**＝“高”的子集 $T_2^{(1)}=(2(\mathrm{N}),3(\mathrm{P}),5(\mathrm{N}),6(\mathrm{P}),7(\mathrm{N}))$，**实例数** $e_2=5$，**其中** $e_2(\mathrm{P})=2$，$e_2(\mathrm{N})=3$，**计算** $T_2^{(1)}$ **的熵**：

$$I(T_2^{(1)})=-\left(\frac{2}{5}\log_2\frac{2}{5}+\frac{3}{5}\log_2\frac{3}{5}\right)=0.971$$

计算使用属性“身高”划分 T **的平均信息量**：

$$I(T,\text{身高})=\frac{e_1}{e}I(T_1^{(1)})+\frac{e_2}{e}I(T_2^{(1)})=\frac{3}{8}\times0.918+\frac{5}{8}\times0.971=0.951$$

Loop12　**取属性表中第** 2 **个属性“发色”划分** T **为** 3 **个子集**。

① **发色**＝“亚麻色”的子集 $T_1^{(2)}=(1(\mathrm{P}),2(\mathrm{N}),6(\mathrm{P}),8(\mathrm{N}))$，**实例数** $e_2=4$，**其中** $e_2(\mathrm{P})=2$，$e_2(\mathrm{N})=2$，**计算** $T_1^{(2)}$ **的熵**：

$$I(T_1^{(2)})=-\left(\frac{2}{4}\log_2\frac{2}{4}+\frac{2}{4}\log_2\frac{2}{4}\right)=1$$

② **发色**＝“棕色”的子集 $T_2^{(2)}=(3(\mathrm{P}))$，**实例数** $e_2=1$，**其中** $e_2(\mathrm{P})=1$，$e_2(\mathrm{N})=0$，**计算** $T_2^{(2)}$ **的熵**：

$$I(T_2^{(2)})=-\left(\frac{1}{1}\log_2\frac{1}{1}+\frac{0}{1}\log_2\frac{0}{1}\right)=0$$

③ **发色**＝“黑色”的子集 $T_3^{(2)}=(2(\mathrm{N}),5(\mathrm{N}),7(\mathrm{N}))$，**实例数** $e_3=3$，**其中** $e_3(\mathrm{P})=0$，$e_3(\mathrm{N})=3$，**计算** $T_3^{(2)}$ **的熵**：

$$I(T_3^{(2)})=-\left(\frac{0}{3}\log_2\frac{0}{3}+\frac{3}{3}\log_2\frac{3}{3}\right)=0$$

计算使用属性“发色”划分 T **的平均信息量**：

$$I(T,\text{发色})=\frac{e_1}{e}I(T_1^{(2)})+\frac{e_2}{e}I(T_2^{(2)})+\frac{e_3}{e}I(T_3^{(2)})=\frac{4}{8}\times1+\frac{1}{8}\times0+\frac{3}{8}\times0=0.5$$

Loop13　**取属性表中第** 3 **个属性“眼色”划分** T **为** 2 **个子集**。

① **眼色**＝“蓝色”的子集 $T_1^{(3)}=(1(\mathrm{P}),3(\mathrm{P}),4(\mathrm{N}),5(\mathrm{N}),6(\mathrm{P}))$，**实例数** $e_1=5$，**其中** $e_1(\mathrm{P})=3$，$e_1(\mathrm{N})=2$，**计算** $T_1^{(3)}$ **的熵**：

$$I(T_1^{(3)})=-\left(\frac{3}{5}\log_2\frac{3}{5}+\frac{2}{5}\log_2\frac{2}{5}\right)=0.971$$

② **眼色**＝“黑色”的子集 $T_2^{(3)}=(2(\mathrm{N}),7(\mathrm{N}),8(\mathrm{N}))$，**实例数** $e_2=3$，**其中** $e_2(\mathrm{P})=0$，$e_2(\mathrm{N})=3$，**计算** $T_2^{(3)}$ **的熵**：

$$I(T_2^{(3)})=-\left(\frac{0}{3}\log_2\frac{0}{3}+\frac{3}{3}\log_2\frac{3}{3}\right)=0$$

计算使用属性“眼色”划分 T **的平均信息量**：

$$I(T,\text{眼色})=\frac{e_1}{e}I(T_1^{(3)})+\frac{e_2}{e}I(T_2^{(3)})=\frac{5}{8}\times0.971+\frac{3}{8}\times0=0.607$$

比较不同属性划分 T **的平均信息量，有**

$$I(T,\text{身高})>I(T,\text{眼色})>I(T,\text{发色})$$

故选择属性“发色”将 T **划分为** 3 **个子集**：

发色＝“亚麻色”的子集 $T_1=(1(\mathrm{P}),2(\mathrm{N}),6(\mathrm{P}),8(\mathrm{N}))$

发色＝“棕色”的子集 $T_2=(3(\mathrm{P}))$　　**终叶节点，标记为** P

发色＝“黑色”的子集 $T_3=(2(\mathrm{N}),5(\mathrm{N}),7(\mathrm{N}))$　　终叶节点，标记为 N

删除属性表中的属性“发色”，AttrList＝(身高，眼色)。

Loop2　选择属性划分 T_1，T_1 的实例数 $e=4$。当前属性表有 2 个属性，2 个子循环分别如下。

Loop21　取属性表中第 1 个属性“身高”划分 T_1 为 2 个子集。

① 身高＝“矮”的子集 $T_{11}^{(1)}=(1(\mathrm{P}),8(\mathrm{N}))$，实例数 $e_1=2$，其中 $e_1(\mathrm{P})=1$，$e_1(\mathrm{N})=1$，计算 $T_{11}^{(1)}$ 的熵：

$$I(T_{11}^{(1)})=-\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right)=1$$

② 身高＝“高”的子集 $T_{12}^{(1)}=(2(\mathrm{N}),6(\mathrm{P}))$，实例数 $e_2=2$，其中 $e_2(\mathrm{P})=1$，$e_2(\mathrm{N})=1$，计算 $T_{12}^{(1)}$ 的熵：

$$I(T_{12}^{(1)})=-\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right)=1$$

计算使用属性“身高”划分 T_1 的平均信息量：

$$I(T_1,\text{身高})=\frac{e_1}{e}I(T_{11}^{(1)})+\frac{e_2}{e}I(T_{12}^{(1)})=\frac{2}{4}\times1+\frac{2}{4}\times1=1$$

Loop22　取属性表中第 2 个属性“眼色”划分 T_1 为 2 个子集。

① 眼色＝“蓝色”的子集 $T_{11}^{(2)}=(1(\mathrm{P}),6(\mathrm{P}))$，实例数 $e_1=2$，其中 $e_1(\mathrm{P})=2$，$e_2(\mathrm{N})=0$，计算 $T_{11}^{(2)}$ 的熵：

$$I(T_{11}^{(2)})=-\left(\frac{2}{2}\log_2\frac{2}{2}+\frac{0}{2}\log_2\frac{0}{2}\right)=0$$

② 眼色＝“黑色”的子集 $T_{12}^{(2)}=(2(\mathrm{N}),8(\mathrm{N}))$，实例数 $e_2=2$，其中 $e_2(\mathrm{P})=0$，$e_2(\mathrm{N})=2$，计算 $T_{12}^{(2)}$ 的熵：

$$I(T_{12}^{(2)})=-\left(\frac{0}{2}\log_2\frac{0}{2}+\frac{2}{2}\log_2\frac{2}{2}\right)=0$$

计算使用属性“眼色”划分 T_1 的平均信息量：

$$I(T_1,\text{眼色})=\frac{e_1}{e}I(T_{11}^{(2)})+\frac{e_2}{e}I(T_{12}^{(2)})=\frac{2}{4}\times0+\frac{2}{4}\times0=0$$

比较不同属性划分 T_1 的平均信息量，有

$$I(T_1,\text{身高})>I(T_1,\text{眼色})$$

故选择属性“眼色”将 T_1 划分为 2 个子集：

眼色＝“蓝色”的子集 $T_{11}=(1(\mathrm{P}),6(\mathrm{P}))$　　终叶节点，标记为 P

眼色＝“黑色”的子集 $T_{12}=(2(\mathrm{N}),8(\mathrm{N}))$　　终叶节点，标记为 N

删除属性表中的属性“眼色”，AttrList＝(身高)。

Loop3　所有叶节点都是终叶节点，故算法成功终止。

由 ID3 算法生成的决策树如图 7.2 所示。

(2) 由决策树得出产生式规则集：

R_1：(发色＝“亚麻色”)∧(眼色＝“蓝色”)→类别＝“P”

R_2：(发色＝“亚麻色”)∧(眼色＝“黑色”)→类别＝“N”

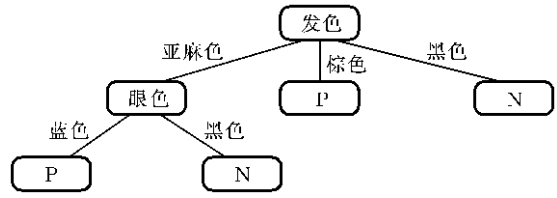

图 7.2 例 7.2 由 ID3 算法生成的决策树

R_3：发色＝“棕色”→类别＝“P”

R_4：发色＝“黑色”→类别＝“N”

7.2.3 归纳学习方法的应用

例 7.3 根据长方体工件的外部尺寸加工误差对工件进行分类，3 个分类特征属性分别是长误差范围、宽误差范围、高误差范围；分类结果是 Y(可用)和 N(不可用)。特征属性的值域分别是

$$\text{Value}(\text{长误差范围})=\{a_1,a_2,a_3\}$$

$$\text{Value}(\text{宽误差范围})=\{b_1,b_2\}$$

$$\text{Value}(\text{高误差范围})=\{h_1,h_2\}$$

初始属性表为 AttrList＝(长误差，宽误差，高误差)。

现从工件检测数据库中随机抽取 8 个工件组成学习实例集 T 如表 7.2 所示。

表 7.2 例 7.3 的学习实例集

序号	长误差	宽误差	高误差	类别	序号	长误差	宽误差	高误差	类别
1	a_1	b_1	h_1	Y	5	a_3	b_2	h_1	N
2	a_1	b_2	h_2	N	6	a_1	b_2	h_1	Y
3	a_2	b_2	h_1	Y	7	a_3	b_2	h_2	N
4	a_3	b_1	h_1	N	8	a_1	b_1	h_2	N

(1) 应用 CLS 算法生成工件分类决策树。

(2) 由决策树得出产生式规则集。

(3) 对规则集中的规则进行合并，给出没有冗余的规则集。

(4) 若初始属性表为 AttrList＝(宽误差，长误差，高误差)，请应用 CLS 算法重做本题。

(5) 应用 ID3 算法重做本题。

解 (1)由表 7.2 给出的学习实例集可得出初始实例集为

$$T=(1(\text{Y}),2(\text{N}),3(\text{Y}),4(\text{N}),5(\text{N}),6(\text{Y}),7(\text{N}),8(\text{N}))$$

初始属性表 AttrList＝(长误差，宽误差，高误差)

Loop1 取属性表中第 1 个属性“长误差”划分 T 为 3 个子集：

长误差＝“a_1”的子集 $T_1=(1(\text{Y}),2(\text{N}),6(\text{Y}),8(\text{N}))$

长误差＝“a_2”的子集 $T_2=(3(\text{Y}))$ 终叶节点，标记为 Y

长误差＝“a_3”的子集 T_3＝(4(N),5(N),7(N))　　终叶节点，标记为 N

删除属性表中第 1 个属性，AttrList＝(宽误差，高误差)。

Loop2　取属性表中第 1 个属性“宽误差”划分 T_1 为 2 个子集：

宽误差＝“b_1”的子集 T_{11}＝(1(Y),8(N))

宽误差＝“b_2”的子集 T_{12}＝(2(N),6(Y))

删除属性表中第 1 个属性，AttrList＝(高误差)。

Loop3　取属性表中第 1 个属性“高误差”划分 T_{11} 和 T_{12}。

Loop31　属性“高误差”划分 T_{11} 为 2 个子集：

高误差＝“h_1”的子集 T_{111}＝(1(Y))　　终叶节点，标记为 Y

高误差＝“h_2”的子集 T_{112}＝(8(N))　　终叶节点，标记为 N

Loop32　属性“高误差”划分 T_{12} 为 2 个子集：

高误差＝“h_1”的子集 T_{121}＝(6(Y))　　终叶节点，标记为 Y

高误差＝“h_2”的子集 T_{122}＝(2(N))　　终叶节点，标记为 N

删除属性表中第 1 个属性，AttrList＝( )。

Loop4　所有叶节点都是终叶节点，故算法成功终止。

由 CLS 算法生成的决策树如图 7.3 所示。

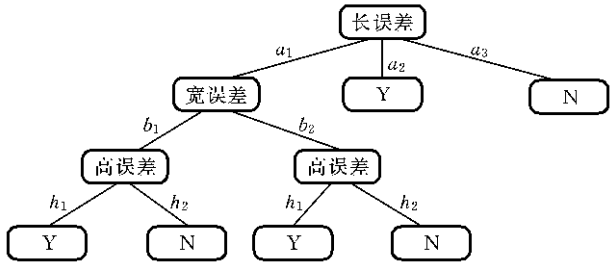

图 7.3　例 7.3 由 CLS 算法生成的决策树

(2) 由决策树得出产生式规则集

R_1：(长误差＝“a_1”) ∧ (宽误差＝“b_1”) ∧ (高误差＝“h_1”)→类别＝“Y”

R_2：(长误差＝“a_1”) ∧ (宽误差＝“b_1”) ∧ (高误差＝“h_2”)→类别＝“N”

R_3：(长误差＝“a_1”) ∧ (宽误差＝“b_2”) ∧ (高误差＝“h_1”)→类别＝“Y”

R_4：(长误差＝“a_1”) ∧ (宽误差＝“b_2”) ∧ (高误差＝“h_2”)→类别＝“N”

R_5：长误差＝“a_2”→类别＝“Y”

R_6：长误差＝“a_3”→类别＝“N”

(3) 对规则集中的规则合并

对后件相同的规则，可考察前件包含的条件，合并成新规则来代替原有规则。

R_1 与 R_3 可合并为：(长误差＝“a_1”) ∧ (高误差＝“h_1”)→类别＝“Y”

R_2 与 R_4 可合并为：(长误差＝“a_1”) ∧ (高误差＝“h_2”)→类别＝“N”

R_5：长误差＝“a_2”→类别＝“Y”

R_6：长误差＝“a_3”→类别＝“N”

(4) 初始实例集

$$T=(1(Y),2(N),3(Y),4(N),5(N),6(Y),7(N),8(N))$$

初始属性表 AttrList=(宽误差,长误差,高误差)

Loop1 取属性表中第 1 个属性"宽误差"划分 T 为 2 个子集:

宽误差="b_1"的子集 $T_1=(1(Y),4(N),8(N))$

宽误差="b_2"的子集 $T_2=(2(N),3(Y),5(N),6(Y),7(N))$

删除属性表中第 1 个属性,AttrList=(长误差,高误差)。

Loop2 取属性表中第 1 个属性"长误差"划分 T_1 和 T_2。

Loop21 属性"长误差"划分 T_1 为 2 个子集:

长误差="a_1"的子集 $T_{11}=(1(Y),8(N))$

长误差="a_2"的子集 $T_{12}=(\ )$ 终叶节点,删除

长误差="a_3"的子集 $T_{13}=(4(N))$ 终叶节点,标记为 N

Loop22 属性"长误差"划分 T_2 为 2 个子集:

长误差="a_1"的子集 $T_{21}=(2(N),6(Y))$

长误差="a_2"的子集 $T_{22}=(3(Y))$ 终叶节点,标记为 Y

长误差="a_3"的子集 $T_{23}=(5(N),7(N))$ 终叶节点,标记为 N

删除属性表中第 1 个属性,AttrList=(高误差)。

Loop3 取属性表中第 1 个属性"高误差"划分 T_{11} 和 T_{21}。

Loop31 属性"高误差"划分 T_{11} 为 2 个子集:

高误差="h_1"的子集 $T_{111}=(1(Y))$ 终叶节点,标记为 Y

长误差="h_2"的子集 $T_{112}=(8(N))$ 终叶节点,标记为 N

Loop32 属性"高误差"划分 T_{21} 为 2 个子集:

高误差="h_1"的子集 $T_{211}=(6(Y))$ 终叶节点,标记为 Y

高误差="h_2"的子集 $T_{212}=(2(N))$ 终叶节点,标记为 N

删除属性表中第 1 个属性,AttrList=( )。

Loop4 所有叶节点都是终叶节点,故算法成功终止。

由 CLS 算法生成的决策树如图 7.4 所示。

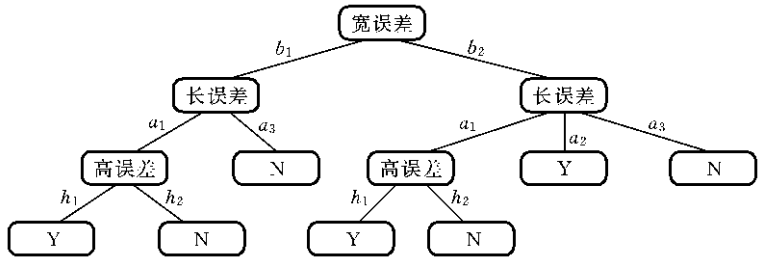

图 7.4 例 7.3 由 CLS 算法生成的另一种决策树

由决策树得出产生式规则集

R_1:(宽误差="b_1")∧(长误差="a_1")∧(高误差="h_1")→类别="Y"

R_2:(宽误差="b_1")∧(长误差="a_1")∧(高误差="h_2")→类别="N"

R_3:(宽误差="b_1")∧(长误差="a_3")→类别="N"

R_4:(宽误差="b_2")∧(长误差="a_1")∧(高误差="h_1")→类别="Y"

R_5:(宽误差="b_2")∧(长误差="a_1")∧(高误差="h_2")→类别="N"

R_6:(宽误差="b_2")∧(长误差="a_2")→类别="Y"

R_7:(宽误差="b_2")∧(长误差="a_3")→类别="N"

对规则集中的规则合并

R_1与 R_4可合并为:(长误差="a_1")∧(高误差="h_1")→类别="Y"

R_2与 R_5可合并为:(长误差="a_1")∧(高误差="h_2")→类别="N"

R_3与 R_7可合并为:长误差="a_3"→类别="N"

R_6:　　(宽误差="b_2")∧(长误差="a_2")→类别="Y"

(5) 初始实例集为

$$T=(1(Y),2(N),3(Y),4(N),5(N),6(Y),7(N),8(N))$$

初始属性表 AttrList=(长误差,宽误差,高误差)。

Loop1　选择属性划分 T,T 的实例数 $e=8$。当前属性表有 3 个属性,3 个子循环分别如下。

Loop11　取属性表中第 1 个属性"长误差"划分 T 为 3 个子集。

① 长误差="a_1"的子集 $T_1^{(1)}=(1(Y),2(N),6(Y),8(N))$,实例数 $e_1=4$,其中 $e_1(Y)=2$,$e_1(N)=2$,计算 $T_1^{(1)}$ 的熵:

$$I(T_1^{(1)})=-\left(\frac{e_1(Y)}{e_1}\log_2\frac{e_1(Y)}{e}+\frac{e_1(N)}{e_1}\log_2\frac{e_1(N)}{e_1}\right)=-\left(\frac{2}{4}\log_2\frac{2}{4}+\frac{2}{4}\log_2\frac{2}{4}\right)=1$$

② 长误差="a_2"的子集 $T_2^{(1)}=(3(Y))$,实例数 $e_2=1$,其中 $e_2(Y)=1$,$e_2(N)=0$,计算 $T_2^{(1)}$ 的熵:

$$I(T_2^{(1)})=-\left(\frac{1}{1}\log_2\frac{1}{1}+\frac{0}{1}\log_2\frac{0}{1}\right)=0$$

③ 长误差="a_3"的子集 $T_3^{(1)}=(4(N),5(N),7(N))$,实例数 $e_3=3$,其中 $e_3(Y)=0$,$e_3(N)=3$,计算 $T_3^{(1)}$ 的熵:

$$I(T_3^{(1)})=-\left(\frac{0}{3}\log_2\frac{0}{3}+\frac{3}{3}\log_2\frac{3}{3}\right)=0$$

计算使用属性"长误差"划分 T 的平均信息量:

$$I(T,\text{长误差})=\frac{e_1}{e}I(T_1^{(1)})+\frac{e_2}{e}I(T_2^{(1)})+\frac{e_3}{e}I(T_3^{(1)})=\frac{4}{8}\times1+\frac{1}{8}\times0+\frac{3}{8}\times0=0.5$$

Loop12　取属性表中第 2 个属性"宽误差"划分 T 为 2 个子集。

① 宽误差="b_1"的子集 $T_1^{(2)}=(1(Y),4(N),8(N))$,实例数 $e_1=3$,其中 $e_1(Y)=1$,$e_1(N)=2$,计算 $T_1^{(2)}$ 的熵:

$$I(T_1^{(2)})=-\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right)=0.918$$

② 宽误差="b_2"的子集 $T_2^{(2)}=(2(N),3(Y),5(N),6(Y),7(N))$,实例数 $e_2=5$,其中 $e_2(Y)=2$,$e_2(N)=3$,计算 $T_2^{(2)}$ 的熵:

$$I(T_2^{(2)})=-\left(\frac{2}{5}\log_2\frac{2}{5}+\frac{3}{5}\log_2\frac{3}{5}\right)=0.971$$

计算使用属性"宽误差"划分 T 的平均信息量:

$$I(T,\text{宽误差})=\frac{e_1}{e}I(T_1^{(1)})+\frac{e_2}{e}I(T_2^{(1)})=\frac{3}{8}\times 0.918+\frac{5}{8}\times 0.971=0.951$$

Loop13　取属性表中第 3 个属性“高误差”划分 T 为 2 个子集。

① 高误差=“h_1”的子集 $T_1^{(3)}=(1(\mathrm{Y}),3(\mathrm{Y}),4(\mathrm{N}),5(\mathrm{N}),6(\mathrm{Y}))$，实例数 $e_1=5$，其中 $e_1(\mathrm{Y})=3$，$e_1(\mathrm{N})=2$，计算 $T_1^{(3)}$ 的熵：

$$I(T_1^{(3)})=-\left(\frac{3}{5}\log_2\frac{3}{5}+\frac{2}{5}\log_2\frac{2}{5}\right)=0.971$$

② 高误差=“h_2”的子集 $T_2^{(3)}=(2(\mathrm{N}),7(\mathrm{N}),8(\mathrm{N}))$，实例数 $e_2=3$，其中 $e_2(\mathrm{Y})=0$，$e_2(\mathrm{N})=3$，计算 $T_2^{(3)}$ 的熵：

$$I(T_2^{(3)})=-\left(\frac{0}{3}\log_2\frac{0}{3}+\frac{3}{3}\log_2\frac{3}{3}\right)=0$$

计算使用属性“高误差”划分 T 的平均信息量：

$$I(T,\text{高误差})=\frac{e_1}{e}I(T_1^{(3)})+\frac{e_2}{e}I(T_2^{(3)})=\frac{5}{8}\times 0.971+\frac{3}{8}\times 0=0.607$$

比较不同属性划分 T 的平均信息量，有

$$I(T,\text{宽误差})>I(T,\text{高误差})>I(T,\text{长误差})$$

故选择属性“长误差”将 T 划分为 3 个子集：

长误差=“a_1”的子集 $T_1=(1(\mathrm{Y}),2(\mathrm{N}),6(\mathrm{Y}),8(\mathrm{N}))$

长误差=“a_2”的子集 $T_2=(3(\mathrm{Y}))$　　终叶节点，标记为 Y

长误差=“a_3”的子集 $T_3=(4(\mathrm{N}),5(\mathrm{N}),7(\mathrm{N}))$　　终叶节点，标记为 N

删除属性表中“长误差”，AttrList=(宽误差，高误差)。

Loop2　选择属性划分 T_1，T_1 的实例数 $e=4$。当前属性表有 2 个属性，2 个子循环分别如下。

Loop21　取属性表中第 1 个属性“宽误差”划分 T_1 为 2 个子集。

① 宽误差=“b_1”的子集 $T_{11}^{(1)}=(1(\mathrm{Y}),8(\mathrm{N}))$，实例数 $e_1=2$，其中 $e_1(\mathrm{Y})=1$，$e_1(\mathrm{N})=1$，计算 $T_{11}^{(1)}$ 的熵：

$$I(T_{11}^{(1)})=-\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right)=1$$

② 宽误差=“b_2”的子集 $T_{12}^{(1)}=(2(\mathrm{N}),6(\mathrm{Y}))$，实例数 $e_2=2$，其中 $e_2(\mathrm{Y})=1$，$e_2(\mathrm{N})=1$，计算 $T_{12}^{(1)}$ 的熵：

$$I(T_{12}^{(1)})=-\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right)=1$$

计算使用属性“宽误差”划分 T_1 的平均信息量：

$$I(T_1,\text{宽误差})=\frac{e_1}{e}I(T_{11}^{(1)})+\frac{e_2}{e}I(T_{12}^{(1)})=\frac{2}{4}\times 1+\frac{2}{4}\times 1=1$$

Loop22　取属性表中第 2 个属性“高误差”划分 T_1 为 2 个子集。

① 高误差=“h_1”的子集 $T_{11}^{(2)}=(1(\mathrm{Y}),6(\mathrm{Y}))$，实例数 $e_1=2$，其中 $e_1(\mathrm{Y})=2$，$e_1(\mathrm{N})=0$，计算 $T_{11}^{(2)}$ 的熵：

$$I(T_{11}^{(2)})=-\left(\frac{2}{2}\log_2\frac{2}{2}+\frac{0}{2}\log_2\frac{0}{2}\right)=0$$

② 高误差=“h_2”的子集 $T_{12}^{(2)}=(2(\mathrm{N}),8(\mathrm{N}))$，实例数 $e_2=2$，其中 $e_2(\mathrm{Y})=0$，$e_2(\mathrm{N})=2$，计算 $T_{12}^{(2)}$ 的熵：

$$I(T_{12}^{(2)}) = -\left(\frac{0}{2}\log_2\frac{0}{2} + \frac{2}{2}\log_2\frac{2}{2}\right) = 0$$

计算使用属性“高误差”划分 T_1 的平均信息量：

$$I(T_1, \text{高误差}) = \frac{e_1}{e}I(T_{11}^{(2)}) + \frac{e_2}{e}I(T_{12}^{(2)}) = \frac{2}{4}\times 0 + \frac{2}{4}\times 0 = 0$$

比较不同属性划分 T_1 的平均信息量，有

$$I(T_1, \text{宽误差}) > I(T_1, \text{高误差})$$

故选择属性“高误差”将 T_1 划分为 2 个子集：

高误差＝“h_1”的子集 T_{11}＝(1(Y),6(Y))　　终叶节点，标记为 Y

高误差＝“h_2”的子集 T_{12}＝(2(N),8(N))　　终叶节点，标记为 N

删除属性表中“高误差”，AttrList＝(宽误差)。

Loop3　所有叶节点都是终叶节点，故算法成功终止。

由 ID3 算法生成的决策树如图 7.5 所示。

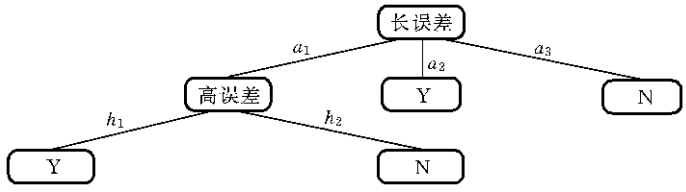

图 7.5　例 7.3 由 ID3 算法生成的决策树

由决策树得出产生式规则集

R_1：(长误差＝“a_1”) ∧ (高误差＝“h_1”)→类别＝“Y”

R_2：(长误差＝“a_1”) ∧ (高误差＝“h_2”)→类别＝“N”

R_3：长误差＝“a_2”→类别＝“Y”

R_4：长误差＝“a_3”→类别＝“N”

例 7.4　根据机场起降条件、飞机类型、起降特殊要求和天气情况等 4 种属性应用 ID3 算法得出允许起降(Y)和禁止起降(N)的产生式规则集。属性的值域分别是

Value(机场起降条件)＝$\{a_1, a_2, a_3\}$

Value(飞机类型)＝$\{b_1, b_2, b_3\}$

Value(起降特殊要求)＝$\{c_1, c_2\}$

Value(天气情况)＝$\{d_1, d_2, d_3\}$

初始属性表为 AttrList＝(机场起降条件，飞机类型，起降特殊要求，天气情况)。

现从数据库中随机抽取 24 个实例组成学习实例集 T 如表 7.3 所示。

表 7.3　例 7.4 的学习实例集

序号	起降条件	飞机类型	特殊要求	天气情况	类别	序号	起降条件	飞机类型	特殊要求	天气情况	类别
1	a_2	b_1	c_1	d_1	N	13	a_2	b_2	c_1	d_1	N
2	a_2	b_1	c_1	d_2	N	14	a_2	b_2	c_1	d_3	N
3	a_2	b_1	c_1	d_3	N	15	a_2	b_3	c_2	d_1	Y
4	a_1	b_1	c_1	d_1	Y	16	a_2	b_3	c_2	d_3	Y

续表

序号	起降条件	飞机类型	特殊要求	天气情况	类别	序号	起降条件	飞机类型	特殊要求	天气情况	类别
5	a_1	b_1	c_1	d_3	Y	17	a_3	b_2	c_2	d_1	N
6	a_3	b_2	c_1	d_1	N	18	a_3	b_2	c_2	d_3	N
7	a_3	b_2	c_1	d_3	N	19	a_2	b_2	c_2	d_3	Y
8	a_3	b_1	c_2	d_1	Y	20	a_2	b_2	c_2	d_2	Y
9	a_3	b_3	c_2	d_3	N	21	a_1	b_2	c_1	d_2	Y
10	a_3	b_1	c_2	d_2	N	22	a_1	b_2	c_1	d_3	Y
11	a_1	b_3	c_2	d_2	Y	23	a_1	b_1	c_2	d_1	Y
12	a_1	b_3	c_2	d_3	Y	24	a_3	b_2	c_1	d_2	N

(1) **应用** ID3 **算法生成决策树**。

(2) **由决策树得出产生式规则集**。

解 (1)**由表** 7.3 **给出的学习实例集可得出初始实例集为**

T=(1(N),2(N),3(N),4(Y),5(Y),6(N),7(N),8(Y),9(N),10(N),11(Y),12(Y),13(N),14(N),15(Y),16(Y),17(N),18(N),19(Y),20(Y),21(Y),22(Y),23(Y),24(N))

初始属性表 AttrList=(**起降条件**,**飞机类型**,**特殊要求**,**天气情况**)。

Loop1 **选择属性划分** T,T **的实例数** e=24。**当前属性表有** 4 **个属性**,4 **个子循环分别如下**。

Loop11 **取属性表中第** 1 **个属性"起降条件"划分** T **为** 3 **个子集**。

① **起降条件**="a_1"**的子集** $T_1^{(1)}$=(4(Y),5(Y),11(Y),12(Y),21(Y),22(Y),23(Y))。

实例数 e_1=7,**其中** e_1(Y)=7,e_1(N)=0,**计算** $T_1^{(1)}$ **的熵**:

$$I(T_1^{(1)}) = -\left(\frac{e_1(\mathrm{Y})}{e_1}\log_2\frac{e_1(\mathrm{Y})}{e_1}+\frac{e_1(\mathrm{N})}{e_1}\log_2\frac{e_1(\mathrm{N})}{e_1}\right) = -\left(\frac{7}{7}\log_2\frac{7}{7}+\frac{0}{7}\log_2\frac{0}{7}\right) = 0$$

② **起降条件**="a_2"**的子集** $T_2^{(1)}$=(1(N),2(N),3(N),13(N),14(N),15(Y),16(Y),19(Y),20(Y))。

实例数 e_2=9,**其中** e_2(Y)=4,e_2(N)=5,**计算** $T_2^{(1)}$ **的熵**:

$$I(T_2^{(1)}) = -\left(\frac{4}{9}\log_2\frac{4}{9}+\frac{5}{9}\log_2\frac{5}{9}\right) = 0.991$$

③ **起降条件**="a_3"**的子集** $T_3^{(1)}$=(6(N),7(N),8(Y),9(N),10(N),17(N),18(N),24(N))。

实例数 e_3=8,**其中** e_3(Y)=1,e_3(N)=7,**计算** $T_3^{(1)}$ **的熵**:

$$I(T_3^{(1)}) = -\left(\frac{1}{8}\log_2\frac{1}{8}+\frac{7}{8}\log_2\frac{7}{8}\right) = 0.7499$$

计算使用属性"起降条件"划分 T **的平均信息量**:

$$I(T,\text{起降条件}) = \frac{e_1}{e}I(T_1^{(2)})+\frac{e_2}{e}I(T_2^{(1)})+\frac{e_3}{e}I(T_3^{(1)})$$

$$= \frac{7}{24}\times 0+\frac{9}{24}\times 0.991+\frac{8}{24}\times 0.7499 = 0.6261$$

Loop12 **取属性表中第** 2 **个属性"飞机类型"划分** T **为** 3 **个子集**。

① **飞机类型**＝“b_1”**的子集** $T_1^{(2)}=(1(\mathrm{N}),2(\mathrm{N}),3(\mathrm{N}),4(\mathrm{Y}),5(\mathrm{Y}),8(\mathrm{Y}),10(\mathrm{N}),23(\mathrm{Y}))$。

实例数 $e_1=8$,**其中** $e_1(\mathrm{Y})=4$,$e_1(\mathrm{N})=4$,**计算** $T_1^{(2)}$ **的熵**：

$$I(T_1^{(2)})=-\left(\frac{4}{8}\log_2\frac{4}{8}+\frac{4}{8}\log_2\frac{4}{8}\right)=1$$

② **飞机类型**＝“b_2”**的子集** $T_2^{(2)}=(6(\mathrm{N}),7(\mathrm{N}),13(\mathrm{N}),14(\mathrm{N}),17(\mathrm{N}),18(\mathrm{N}),19(\mathrm{Y}),20(\mathrm{Y}),21(\mathrm{Y}),22(\mathrm{Y}),24(\mathrm{N}))$。

实例数 $e_2=11$,**其中** $e_2(\mathrm{Y})=4$,$e_2(\mathrm{N})=7$,**计算** $T_2^{(2)}$ **的熵**：

$$I(T_2^{(2)})=-\left(\frac{4}{11}\log_2\frac{4}{11}+\frac{7}{11}\log_2\frac{7}{11}\right)=0.9456$$

③ **飞机类型**＝“b_3”**的子集** $T_3^{(2)}=(9(\mathrm{N}),11(\mathrm{Y}),12(\mathrm{Y}),15(\mathrm{Y}),16(\mathrm{Y}))$。

实例数 $e_3=5$,**其中** $e_3(\mathrm{Y})=4$,$e_3(\mathrm{N})=1$,**计算** $T_3^{(2)}$ **的熵**：

$$I(T_3^{(2)})=-\left(\frac{4}{5}\log_2\frac{4}{5}+\frac{1}{5}\log_2\frac{1}{5}\right)=0.7218$$

计算使用属性“飞机类型”划分 T **的平均信息量**：

$$\begin{aligned}I(T,\text{飞机类型})&=\frac{e_1}{e}I(T_1^{(2)})+\frac{e_2}{e}I(T_2^{(2)})+\frac{e_3}{e}I(T_3^{(2)})\\&=\frac{8}{24}\times1+\frac{11}{24}\times0.9456+\frac{5}{24}\times0.7218=0.9171\end{aligned}$$

Loop13　**取属性表中第** 3 **个属性“特殊要求”划分** T **为** 2 **个子集**。

① **特殊要求**＝“c_1”**的子集** $T_1^{(3)}=(1(\mathrm{N}),2(\mathrm{N}),3(\mathrm{N}),4(\mathrm{Y}),5(\mathrm{Y}),6(\mathrm{N}),7(\mathrm{N}),13(\mathrm{N}),14(\mathrm{N}),21(\mathrm{Y}),22(\mathrm{Y}),24(\mathrm{N}))$。

实例数 $e_1=12$,**其中** $e_1(\mathrm{Y})=4$,$e_1(\mathrm{N})=8$,**计算** $T_1^{(3)}$ **的熵**：

$$I(T_1^{(3)})=-\left(\frac{4}{12}\log_2\frac{4}{12}+\frac{8}{12}\log_2\frac{8}{12}\right)=0.9183$$

② **特殊要求**＝“c_2”**的子集** $T_2^{(3)}=(8(\mathrm{Y}),9(\mathrm{N}),10(\mathrm{N}),11(\mathrm{Y}),12(\mathrm{Y}),15(\mathrm{Y}),16(\mathrm{Y}),17(\mathrm{N}),18(\mathrm{N}),19(\mathrm{Y}),20(\mathrm{Y}),23(\mathrm{Y}))$。

实例数 $e_2=12$,**其中** $e_2(\mathrm{Y})=8$,$e_2(\mathrm{N})=4$,**计算** $T_2^{(3)}$ **的熵**：

$$I(T_2^{(3)})=-\left(\frac{8}{12}\log_2\frac{8}{12}+\frac{4}{12}\log_2\frac{4}{12}\right)=0.9183$$

计算使用属性“特殊要求”划分 T **的平均信息量**：

$$\begin{aligned}I(T,\text{特殊要求})&=\frac{e_1}{e}I(T_1^{(3)})+\frac{e_2}{e}I(T_2^{(3)})\\&=\frac{12}{24}\times0.9183+\frac{12}{24}\times0.9183=0.9183\end{aligned}$$

Loop14　**取属性表中第** 4 **个属性“天气情况”划分** T **为** 3 **个子集**。

① **天气情况**＝“d_1”**的子集** $T_1^{(4)}=(1(\mathrm{N}),4(\mathrm{Y}),6(\mathrm{N}),8(\mathrm{Y}),13(\mathrm{N}),15(\mathrm{Y}),17(\mathrm{N}),23(\mathrm{Y}))$。

实例数 $e_1=8$,**其中** $e_1(\mathrm{Y})=4$,$e_1(\mathrm{N})=4$,**计算** $T_1^{(4)}$ **的熵**：

$$I(T_1^{(4)})=-\left(\frac{4}{8}\log_2\frac{4}{8}+\frac{4}{8}\log_2\frac{4}{8}\right)=1$$

② 天气情况="d_2"的子集 $T_2^{(4)}$=(2(N),10(N),11(Y),20(Y),21(Y),24(N))。

实例数 $e_2=6$,其中 $e_2(Y)=3$,$e_2(N)=3$,计算 $T_2^{(4)}$ 的熵:

$$I(T_2^{(4)}) = -\left(\frac{3}{6}\log_2\frac{3}{6}+\frac{3}{6}\log_2\frac{3}{6}\right)=1$$

③ 天气情况="d_3"的子集 $T_3^{(4)}$=(3(N),5(Y),7(N),9(N),12(Y),14(N),16(Y),18(N),19(Y),20(Y))。

实例数 $e_3=10$,其中 $e_3(Y)=5$,$e_3(N)=5$,计算 $T_3^{(4)}$ 的熵:

$$I(T_3^{(4)}) = -\left(\frac{5}{10}\log_2\frac{5}{10}+\frac{5}{10}\log_2\frac{5}{10}\right)=1$$

计算使用属性"天气情况"划分 T 的平均信息量:

$$I(T,\text{天气情况}) = \frac{e_1}{e}I(T_1^{(4)})+\frac{e_2}{e}I(T_2^{(4)})+\frac{e_3}{e}I(T_3^{(4)})$$

$$= \frac{8}{24}\times 1+\frac{6}{24}\times 1+\frac{10}{24}\times 1=1$$

比较不同属性划分 T 的平均信息量,有

$$I(T,\text{天气情况})>I(T,\text{特殊要求})>I(T,\text{飞机类型})>I(T,\text{起降条件})$$

故选择属性"起降条件",将 T_1 划分为 3 个子集:

- 起降条件="a_1"的子集

T_1=(4(Y),5(Y),11(Y),12(Y),21(Y),22(Y),23(Y)) 终叶节点,标记为 Y

- 起降条件="a_2"的子集

T_2=(1(N),2(N),3(N),13(N),14(N),15(Y),16(Y),19(Y),20(Y))

- 起降条件="a_3"的子集

T_3=(6(N),7(N),8(Y),9(N),10(N),17(N),18(N),24(N))

删除属性表中的属性"起降条件",AttrList=(飞机类型,特殊要求,天气情况)。

Loop2 选择属性划分 T_2,T_2 的实例数 $e=9$。当前属性表有 3 个属性,3 个子循环分别如下。

Loop21 取属性表中第 1 个属性"飞机类型"划分 T_2 为 3 个子集。

① 飞机类型="b_1"的子集 $T_{21}^{(1)}$=(1(N),2(N),3(N)),实例数 $e_1=3$,其中 $e_1(Y)=0$,$e_1(N)=3$,计算 $T_{21}^{(1)}$ 的熵:

$$I(T_{21}^{(1)}) = -\left(\frac{0}{3}\log_2\frac{0}{3}+\frac{3}{3}\log_2\frac{3}{3}\right)=0$$

② 飞机类型="b_2"的子集 $T_{22}^{(1)}$=(13(N),14(N),19(Y),20(Y)),实例数 $e_2=4$,其中 $e_2(Y)=2$,$e_2(N)=2$,计算 $T_{22}^{(1)}$ 的熵:

$$I(T_{22}^{(1)}) = -\left(\frac{2}{4}\log_2\frac{2}{4}+\frac{2}{4}\log_2\frac{2}{4}\right)=1$$

③ 飞机类型="b_3"的子集 $T_{23}^{(1)}$=(15(Y),16(Y)),实例数 $e_3=2$,其中 $e_3(Y)=2$,$e_3(N)=0$,计算 $T_{23}^{(1)}$ 的熵:

$$I(T_{23}^{(1)}) = -\left(\frac{2}{2}\log_2\frac{2}{2}+\frac{0}{2}\log_2\frac{0}{2}\right)=0$$

计算使用属性“飞机类型”划分 T_2 的平均信息量：

$$I(T_2,\text{飞机类型}) = \frac{e_1}{e}I(T_{21}^{(1)}) + \frac{e_2}{e}I(T_{22}^{(1)}) + \frac{e_3}{e}I(T_{23}^{(1)})$$

$$= \frac{3}{9}\times 0 + \frac{4}{9}\times 1 + \frac{2}{9}\times 0 = 0.4444$$

Loop22　取属性表中第 2 个属性“特殊要求”划分 T_2 为 2 个子集。

① 特殊要求＝“c_1”的子集 $T_{21}^{(2)}$＝(1(N),2(N),3(N),13(N),14(N))。

实例数 $e_1=5$，其中 $e_1(\text{Y})=0$，$e_1(\text{N})=5$，计算 $T_{21}^{(2)}$ 的熵：

$$I(T_{21}^{(2)}) = -\left(\frac{0}{5}\log_2\frac{0}{5} + \frac{5}{5}\log_2\frac{5}{5}\right) = 0$$

② 特殊要求＝“c_2”的子集 $T_{22}^{(2)}$＝(15(Y),16(Y),19(Y),20(Y))。

实例数 $e_2=4$，其中 $e_2(\text{Y})=4$，$e_2(\text{N})=0$，计算 $T_{22}^{(2)}$ 的熵：

$$I(T_{22}^{(2)}) = -\left(\frac{4}{4}\log_2\frac{4}{4} + \frac{0}{4}\log_2\frac{0}{4}\right) = 0$$

计算使用属性“特殊要求”划分 T_2 的平均信息量：

$$I(T_2,\text{特殊要求}) = \frac{e_1}{e}I(T_{21}^{(2)}) + \frac{e_2}{e}I(T_{22}^{(2)})$$

$$= \frac{5}{9}\times 0 + \frac{4}{9}\times 0 = 0$$

Loop23　取属性表中第 3 个属性“天气情况”划分 T_2 为 3 个子集。

① 天气情况＝“d_1”的子集 $T_{21}^{(3)}$＝(1(N),13(N),15(Y))。

实例数 $e_1=3$，其中 $e_1(\text{Y})=1$，$e_1(\text{N})=2$，计算 $T_{21}^{(3)}$ 的熵：

$$I(T_{21}^{(3)}) = -\left(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}\right) = 0.9183$$

② 天气情况＝“d_2”的子集 $T_{22}^{(3)}$＝(2(N),20(Y))。

实例数 $e_2=2$，其中 $e_2(\text{Y})=1$，$e_2(\text{N})=1$，计算 $T_{22}^{(3)}$ 的熵：

$$I(T_{22}^{(3)}) = -\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right) = 1$$

③ 天气情况＝“d_3”的子集 $T_{23}^{(3)}$＝(3(N),14(N),16(Y),19(Y))。

实例数 $e_3=4$，其中 $e_3(\text{Y})=2$，$e_3(\text{N})=2$，计算 $T_{23}^{(3)}$ 的熵：

$$I(T_{23}^{(3)}) = -\left(\frac{2}{4}\log_2\frac{2}{4} + \frac{2}{4}\log_2\frac{2}{4}\right) = 1$$

计算使用属性“天气情况”划分 T_2 的平均信息量：

$$I(T_2,\text{天气情况}) = \frac{e_1}{e}I(T_{21}^{(3)}) + \frac{e_2}{e}I(T_{22}^{(3)}) + \frac{e_3}{e}I(T_{23}^{(3)})$$

$$= \frac{3}{9}\times 0.9183 + \frac{2}{9}\times 1 + \frac{4}{9}\times 1 = 0.9727$$

比较不同属性划分 T 的平均信息量，有

$$I(T_2,\text{天气情况}) > I(T_2,\text{飞机类型}) > I(T_2,\text{特殊要求})$$

故选择属性“特殊要求”将 T_2 划分为 2 个子集：

● 特殊要求＝“c_1”的子集 T_{21}＝(1(N),2(N),3(N),13(N),14(N)) 终叶节点，标记为 N

● 特殊要求＝“c_2”的子集 T_{22}＝(15(Y),16(Y),19(Y),20(Y)) 终叶节点，标记为 Y

删除属性表中的属性“特殊要求”，AttrList＝(飞机类型，天气情况)。

Loop3 选择属性划分 T_3，T_3 的实例数 e＝8。当前属性表中有 2 个属性，2 个子循环分别如下。

Loop31 取属性表中第 1 个属性“飞机类型”划分 T_3 为 3 个子集。

① 飞机类型＝“b_1”的子集 $T_{31}^{(1)}$＝(8(Y),10(N))。

实例数 e_1＝2，其中 e_1(Y)＝1，e_1(N)＝1，计算 $T_{31}^{(1)}$ 的熵：

$$I(T_{31}^{(1)}) = -\left(\frac{1}{2}\log_2\frac{1}{2}+\frac{1}{2}\log_2\frac{1}{2}\right)=1$$

② 飞机类型＝“b_2”的子集 $T_{32}^{(1)}$＝(6(N),7(N),17(N),18(N),20(N))。

实例数 e_2＝5，其中 e_2(Y)＝0，e_2(N)＝5，计算 $T_{32}^{(1)}$ 的熵：

$$I(T_{32}^{(1)}) = -\left(\frac{0}{5}\log_2\frac{0}{5}+\frac{5}{5}\log_2\frac{5}{5}\right)=0$$

③ 飞机类型＝“b_3”的子集 $T_{33}^{(1)}$＝(9(N))。

实例数 e_3＝1，其中 e_3(Y)＝0，e_3(N)＝1，计算 $T_{33}^{(1)}$ 的熵：

$$I(T_{33}^{(1)}) = -\left(\frac{0}{1}\log_2\frac{0}{1}+\frac{1}{1}\log_2\frac{1}{1}\right)=0$$

计算使用属性“飞机类型”划分 T_3 的平均信息量：

$$I(T_3,\text{飞机类型}) = \frac{e_1}{e}I(T_{31}^{(1)})+\frac{e_2}{e}I(T_{32}^{(1)})+\frac{e_3}{e}I(T_{33}^{(1)})$$

$$= \frac{2}{8}\times 1+\frac{5}{8}\times 0+\frac{1}{8}\times 0 = 0.25$$

Loop32 取属性表中第 2 个属性“天气情况”划分 T_3 为 3 个子集。

① 天气情况＝“d_1”的子集 $T_{31}^{(2)}$＝(6(N),8(Y),17(N))。

实例数 e_1＝3，其中 e_1(Y)＝1，e_1(N)＝2，计算 $T_{31}^{(2)}$ 的熵：

$$I(T_{31}^{(2)}) = -\left(\frac{1}{3}\log_2\frac{1}{3}+\frac{2}{3}\log_2\frac{2}{3}\right)=0.9183$$

② 天气情况＝“d_2”的子集 $T_{32}^{(2)}$＝(10(N),24(N))。

实例数 e_2＝2，其中 e_2(Y)＝0，e_2(N)＝2，计算 $T_{32}^{(2)}$ 的熵：

$$I(T_{32}^{(2)}) = -\left(\frac{0}{2}\log_2\frac{0}{2}+\frac{2}{2}\log_2\frac{2}{2}\right)=0$$

③ 天气情况＝“d_3”的子集 $T_{33}^{(2)}$＝(7(N),9(N),18(N))。

实例数 e_3＝3，其中 e_3(Y)＝0，e_3(N)＝3，计算 $T_{33}^{(2)}$ 的熵：

$$I(T_{33}^{(2)}) = -\left(\frac{0}{3}\log_2\frac{0}{3}+\frac{3}{3}\log_2\frac{3}{3}\right)=0$$

计算使用属性“天气情况”划分 T_3 的平均信息量：

$$I(T_3,\text{天气情况}) = \frac{e_1}{e}I(T_{31}^{(2)})+\frac{e_2}{e}I(T_{32}^{(2)})+\frac{e_3}{e}I(T_{33}^{(2)})$$

$$= \frac{3}{8} \times 0.9183 + \frac{2}{8} \times 0 + \frac{3}{8} \times 0 = 0.3444$$

比较不同属性划分 T_3 的平均信息量，有

$$I(T_3, \text{天气情况}) > I(T_3, \text{飞机类型})$$

故选择属性“飞机类型”将 T_3 划分为 3 个子集：

- 飞机类型＝“b_1”的子集 T_{31}＝(8(Y),10(N))
- 飞机类型＝“b_2”的子集 T_{32}＝(6(N),7(N),17(N),18(N),20(N))　　终叶节点，标记为 N
- 飞机类型＝“b_3”的子集 T_{33}＝(9(N))　　终叶节点，标记为 N

删除属性表中的属性“飞机类型”，AttrList＝(天气情况)。

Loop4　选择属性划分 T_{31}，T_{31} 的实例数 e＝2。

取属性表中第 1 个属性“天气情况”划分 T_{31} 为 3 个子集。

① 天气情况＝“d_1”的子集 $T_{311}^{(1)}$＝(8(Y))，实例数 e_1＝1，其中 e_1(Y)＝1，e_1(N)＝0，计算 $T_{311}^{(1)}$ 的熵：

$$I(T_{311}^{(1)}) = -\left(\frac{1}{1}\log_2\frac{1}{1} + \frac{0}{1}\log_2\frac{0}{1}\right) = 0$$

② 天气情况＝“d_2”的子集 $T_{312}^{(1)}$＝(10(N))，实例数 e_2＝1，其中 e_2(Y)＝0，e_2(N)＝1，计算 $T_{312}^{(1)}$ 的熵：

$$I(T_{312}^{(1)}) = -\left(\frac{0}{1}\log_2\frac{0}{1} + \frac{1}{1}\log_2\frac{1}{1}\right) = 0$$

③ 天气情况＝“d_3”的子集 $T_{313}^{(1)}$＝()，实例数 e_3＝0，空集的熵为 0，故

$$I(T_{313}^{(1)}) = 0$$

计算使用属性“天气情况”划分 T_{31} 的平均信息量：

$$I(T_{31}, \text{天气情况}) = \frac{e_1}{e} I(T_{311}^{(1)}) + \frac{e_2}{e} I(T_{312}^{(1)}) + \frac{e_3}{e} I(T_{313}^{(1)})$$

$$= \frac{1}{2} \times 0 + \frac{1}{2} \times 0 + \frac{0}{2} \times 0 = 0$$

属性表中已无其他属性，Loop4 终止。

故选择属性“天气情况”将 T_{31} 划分为 3 个子集：

- 天气情况＝“d_1”的子集 T_{311}＝(8(Y))　　终叶节点，标记为 Y
- 天气情况＝“d_2”的子集 T_{312}＝(10(N))　　终叶节点，标记为 N
- 天气情况＝“d_3”的子集 T_{313}＝( )　　终叶节点，删除

删除属性表中的属性“天气情况”，AttrList＝( )。

Loop5　所有节点都是终叶节点，故算法成功终止。

由 ID3 算法生成的决策树如图 7.6 所示。

(2) 由决策树得出产生式规则集

R_1：机场起降条件＝“a_1”→允许起降(Y)

R_2：(机场起降条件＝“a_2”) ∧ (起降特殊要求＝“c_1”)→禁止起降(N)

R_3：(机场起降条件＝“a_2”) ∧ (起降特殊要求＝“c_2”)→允许起降(Y)

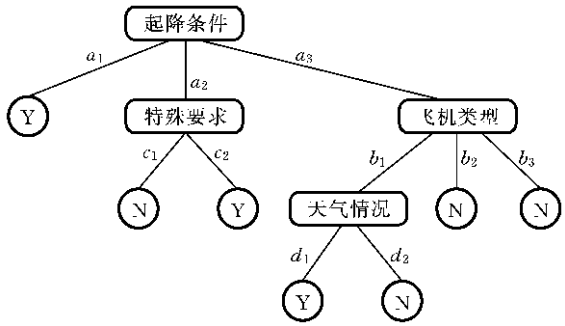

图 7.6 例 7.4 由 ID3 算法生成的决策树

R_4:(机场起降条件="a_3")∧(飞机类型="b_1")∧(天气情况="d_1")→允许起降(Y)

R_5:(机场起降条件="a_3")∧(飞机类型="b_1")∧(天气情况="d_2")→禁止起降(N)

R_6:(机场起降条件="a_3")∧(飞机类型="b_2")→禁止起降(N)

R_7:(机场起降条件="a_3")∧(飞机类型="b_3")→禁止起降(N)

7.3 遗传算法

遗传算法(Genetic Algorithm,GA)是借鉴生物遗传机制的一种随机搜索算法,其主要特点是群体搜索和群体中的个体之间的信息交换。遗传算法尤其适用于处理传统方法难以解决的、复杂的和非线性的问题,可广泛用于机器学习、组合优化和自适应控制等领域。遗传算法的概念和思想由美国 J. Holland 教授提出后,已在许多领域有广泛的应用,它已成为智能计算的主要技术之一。

7.3.1 遗传算法的概念与计算方法

遗传算法是遗传学和计算机科学相互结合与渗透而形成的一种新的计算方法,它使用了遗传学的一些概念和基本术语来描述它的计算方法,因此,了解这些术语对于学习和应用遗传算法是十分必要的。

生物遗传物质的主要载体是染色体,主要的遗传物质是 DNA。染色体由基因组成,基因在染色体中的位置称为基因座,基因座所取的值称为等位基因。基因座和基因值决定了染色体的特征,也决定了生物个体的性状。

染色体有两种表示模式,即基因型(Genetype)表示模式和表现型(Phenotype)表示模式。表现型是指生物个体所表现出来的性状。基因型是指与生物个体表现出来的性状密切相关的基因组成。

在遗传算法中,与染色体相对应的是数据或数组。在标准的遗传算法中,染色体可用一维的串结构数据来表示,串的各个位置对应染色体的基因座,各位置上所取的值对应等位基因。遗传算法对染色体进行处理,或者称为对基因个体(Individuals)进行处理。一定数量的个体

组成群体(Population)，群体中的个体数称为群体大小(Population Size)或群体规模。各个个体对环境的适应程度称为适应度(Fitness)。

执行遗传算法时有两个必需的数据转换操作，一个操作是把搜索空间中的参数或解数据转换成遗传空间的染色体或个体，或者说是把染色体数据从表现型表示转换成基因型表示，这个过程称为编码操作；另一个操作称为译码操作，它是同编码操作相反的数据转换操作，译码操作把染色体数据从基因型表示转换成表现型表示。

遗传算法的基本流程如图 7.7 所示。遗传算法以群体中的所有个体为对象，选择(Selection)、杂交(Crossbreed)和变异(Mutation)是遗传算法的 3 个主要操作算子，它们构成了所谓的遗传操作(Genetic Operations)，使遗传算法有了其他传统方法所没有的特征。

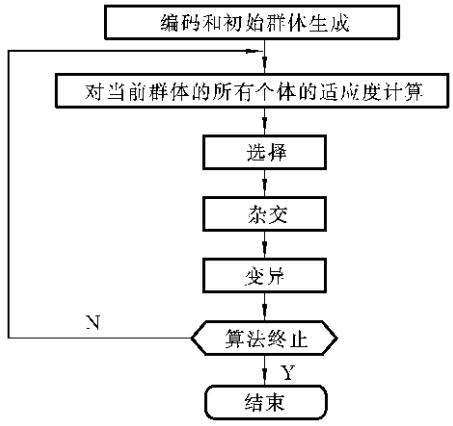

图 7.7　遗传算法的基本流程

应用遗传算法求解问题，需要根据求解的具体问题进行下述 5 个方面的工作：

① 参数编码的格式设定及参数编码；

② 初始群体的设定；

③ 适应度函数的设计；

④ 遗传操作设计；

⑤ 控制参数设计，主要是指群体规模和遗传操作中所需使用的有关控制参数与算法终止条件的设定和设计。

上述 5 个方面的工作是遗传算法的核心内容，也称为遗传算法的 5 个基本要素。下面以一个简单的函数极值求解为例，来说明遗传算法的计算方法和处理过程。

例 7.5　应用遗传算法求 $f(x)=x^2$ 的最大值的解，$x\in[0,31]$。

解　按遗传算法的 5 个基本要素来说明遗传算法对此例的计算方法和处理过程。

(1) 编码

此问题给出的数据 0～31 是表现型表示模式，需要通过编码把它们转换成遗传空间的基因型表示。基因型表示模式通常是串结构形式，十进制数可以很方便地转换成二进制数串，因此，可以方便地把 0～31 编码为 5 位长的二进制无符号整数表示形式。例如，$x=8$ 的基因型表示为 01000，$x=13$ 的基因型表示为 01101 等。

(2) 初始群体的生成

遗传算法是对一个群体中的所有个体为对象进行遗传操作的，为了简化此例的说明，假定取群体规模 $n=4$，即群体由 4 个个体组成。初始群体的每个个体都是从遗传空间中随机选择的，假设随机选择的 4 个个体分别是 $x_1=13=01101$，$x_2=24=11000$，$x_3=8=01000$，$x_4=19=10011$。初始群体也称为进化的初始代或第一代(First Generation)群体。

(3) 适应度函数设计和适应度计算

遗传算法在搜索求解过程中一般不需要其他外部信息，仅用设计的适应度函数来评估个体的优劣，作为对个体进行遗传操作的依据。某个个体 x_i 的适应度函数值 f_i 称为这个个体 x_i

的适应度。此例需求 $f(x)=x^2$ 的最大值的解，可以设定 $f(x)=x^2$ 为个体 x 的适应度函数，由此适应度函数计算各个体的适应度，按适应度的大小来评估个体的优劣，适应度越大的个体越优良，选优汰劣，遗传进化的结果将得到 $f(x)$ 值最大的解 x。

显然，利用 $f(x)=x^2$ 来计算个体 x_i 的适应度 f_i 时，要把个体 x_i 的基因型表示译码成为表现型表示，即将 x_i 的二进制表示转换成十进制表示，然后计算个体 x_i 的适应度 x_i^2。

(4) 选择操作

选择操作是从当前群体中选出优良的个体，使它们有机会作为父代来繁殖下一代。判断个体优良的依据就是个体的适应度，个体的适应度越高，该个体被选择的机会就越多。

选择操作的实现方法很多，最简单易行的操作方法就是直接按适应度 f_i 选择个体，也可以把选择操作算子设计成适应度 f_i 的某种函数 $s(f_i)$，由 $s(f_i)$ 的值 s_i 来选择个体，在此例中，介绍一种常用的选择操作算子，即适应度概率选择算子。采用适应度概率选择算子来实现选择操作的步骤如下。

① 若个体 x_i 的适应度为 f_i，$i=1,2,\cdots,n$，由适应度计算各个体的选择概率 $p_i=f_i\Big/\sum_{i=1}^{n} f_i$，$i=1,2,\cdots,n$，且 $\sum_{i=1}^{n} p_i=1$。

② 根据群体 n 个个体的选择概率的大小，对个体选优汰劣。本例采用的选优汰劣的方法是：按个体选择概率 p_i 的值从大到小将 n 个个体排队，对排队序列 $\mathrm{LI}=(x_1,x_2,\cdots,x_n)$ 的队首个体 x_1 选择 2 次，对队尾个体 x_n 不选择，其他个体选择 1 次，被选择的个体经编码成为基因型表示后送入配对池(Mating Pool)中。

如表 7.4 所示，第一代群体的 4 个个体分别是 $x_1=24$、$x_2=19$、$x_3=13$ 和 $x_4=8$，个体适应度 $f_i=x_i^2$ 分别是 $f_1=576$、$f_2=361$、$f_3=169$ 和 $f_4=64$，选择概率 $p_i=f_i\Big/\sum_{i=1}^{n} f_i$ 分别是 $p_1=0.49$、$p_2=0.31$、$p_3=0.14$ 和 $p_4=0.06$。4 个个体的排队序列是(24,19,13,8)。对 x_1 选择 2 次，对 x_2 和 x_3 各选择 1 次，淘汰 x_4。送入配对池，则配对池 MP 中的表现型个体为{24,24,19,13}，实际上，配对池中的个体是经编码后的基因型个体，所以，配对池为{11000,11000,10011,01101}。

(5) 杂交操作

杂交操作是对配对池的当前群体中的基因型个体随机配对，然后使配成一对的个体彼此交换部分信息，从而得到这一对个体的 2 个后代个体。

杂交操作可分为下述两个步骤来实现。

① 随机配对。对配对池中的个体 x_i 的编号 i 随机指定一个配对的个体编号 j，即把配对池中的个体 x_i 与 x_j 配成一对($1\leqslant i,j\leqslant n,i\neq j$)。

② 杂交繁殖。个体采用基因型表示时，个体通常具有由 m 个基因座组成的一维串结构。例如，本例中的个体的基因型表示是 5 位二进制数串，即有 $m=5$。杂交繁殖是指：对任何一对配对个体随机指定一个交叉位置 k，$1\leqslant k\leqslant m$，2 个个体从基因座 1 至基因座 k 上的基因值彼此对应交换(等位基因互换)，从而产生 2 个新的后代个体。基因座的编号是从串的低位到高位依序上升，即串的最低位为基因座 1，串的最高位为基因座 m。

本例第一代群体经选择操作后，配对池 MP={11000,11000,10011,01101}。若随机配对

时，对个体编号 $i=1$ 随机指定配对个体编号 $j=4$，对个体编号 $i=2$ 随机指定配对个体编号 $j=3$，则把 MP 中的 $x_1=11000$ 与 $x_4=01101$ 配对，把 $x_2=11000$ 与 $x_3=10011$ 配对。

对上述 2 对分别进行杂交繁殖。若对(x_1,x_4)随机指定交叉位置 $k=3$，则把 x_1 和 x_4 的低 3 位的值互换，得到 2 个后代个体分别为 11101 和 01000，这 2 个后代个体的表现型分别为 29 和 8。若对(x_2,x_3)随机指定交叉位置 $k=1$，则把 x_2 和 x_3 的低 1 位的值互换，得到 2 个后代个体分别为 11001 和 10010，这 2 个后代个体的表现型分别为 25 和 18。从而，在配对池中得到第二代群体 $MP=\{29,25,18,8\}$。

(6) 变异操作

变异操作是指把一个个体的基因座的基因值进行改变，从而得到一个新的个体。对于二进制编码的个体来说，若某位(bit)是“0”，则通过变异操作后就变为“1”；若某位是“1”，则变异操作将其改变为“0”。变异操作也是随机选择基因座进行的。一般来说，变异概率都取得很小，在本例中，若取变异概率为 0.01，那么，需要对群体中 $20\times0.01=0.2$ 位进行变异，这意味着群体中没有一位可以变异。变异操作是十分微妙的遗传操作，它需要和杂交操作妥善地配合使用，目的在于挖掘群体中个体的多样性，克服有可能陷入局部最优解的弊端。

本例通过 5 次迭代处理，由初始群体繁殖生成第六代群体，迭代处理过程如表 7.4 所示。由此得到问题的解为 $x=31$。

表 7.4　例 7.5 遗传算法处理过程

排队序列 x_i	适应度 f_i	选择概率 p_i	选择次数	配对池（表现型）	配对池（基因型）	配对个体 x_j	交叉位置 k	后代个体（基因型）	后代个体（表现型）
24	576	0.49	2	24	11000	x_4	3	11101	29
19	361	0.31	1	24	11000	x_3	1	11001	25
13	169	0.14	1	19	10011	x_2	1	10010	18
8	64	0.06	0	13	01101	x_1	3	01000	8
29	841	0.453	2	29	11101	x_3	1	11101	29
25	625	0.337	1	29	11101	x_4	2	11110	30
18	324	0.175	1	25	11001	x_1	1	11001	25
8	64	0.035	0	18	10010	x_2	2	10001	17
30	900	0.339	2	30	11110	x_4	1	11111	31
29	841	0.317	1	30	11110	x_3	4	11101	29
25	625	0.235	1	29	11101	x_2	4	11110	30
17	289	0.109	0	25	11001	x_1	1	11000	24
31	961	0.293	2	31	11111	x_2	2	11111	31
30	900	0.275	1	31	11111	x_1	2	11111	31
29	841	0.256	1	30	11110	x_4	1	11111	31
24	576	0.176	0	29	11101	x_3	1	11100	28
31	961	0.262	2	31	11111	x_2	1	11111	31
31	961	0.262	1	31	11111	x_1	1	11111	31
31	961	0.262	1	31	11111	x_4	2	11111	31
28	784	0.214	0	31	11111	x_3	2	11111	31

在表 7.4 中，第一代群体 $LI(1)=(24,19,13,8)$经过选择、杂交和变异后，得到第二代群体 $LI(2)=(29,25,18,8)$。

对 LI(2)，随机指定的配对为$(x_1,x_3)=(19,18)$和$(x_2,x_4)=(25,8)$，对(x_1,x_3)随机指

定的交叉位置 $k=1$，对(x_2,x_4)随机指定的交叉位置 $k=2$，生成第三代群体 LI(3)=(30,29,25,17)。

对 LI(3)，随机指定的配对为$(x_1,x_4)=(30,17)$和$(x_2,x_3)=(29,25)$，对(x_1,x_4)随机指定的交叉位置 $k=1$，对(x_2,x_3)随机指定的交叉位置 $k=4$，生成第四代群体 LI(4)=(31,30,29,24)。

对 LI(4)，随机指定的配对为$(x_1,x_2)=(31,30)$和$(x_3,x_4)=(29,24)$，对(x_1,x_2)随机指定的交叉位置 $k=2$，对(x_3,x_4)随机指定的交叉位置 $k=1$，生成第五代群体 LI(5)=(31,31,31,28)。

对 LI(5)，随机指定的配对为$(x_1,x_2)=(31,31)$和$(x_3,x_4)=(31,28)$，对(x_1,x_2)随机指定的交叉位置 $k=1$，对(x_3,x_4)随机指定的交叉位置 $k=2$，生成第六代群体 LI(6)=(31,31,31,31)。

第六代群体中的所有个体之间没有差异，因此，迭代处理过程结束，得到问题的解为 $x=31$。

7.3.2 遗传算法的应用

例 7.6 应用遗传算法求解 $f(x)=x^2$ 的最大值的解，$x\in[0,63]$，设群体规模 $n=4$，随机选择的初始群体为$\{56,31,24,8\}$，适应度函数为 $f(x)=x^2$。写出遗传算法的求解过程和问题的解。算法终止条件为：群体中的所有个体无差异。

(1) 选择算子每次选择适应度最大的个体 3 次，淘汰适应度最小的 2 个个体，其他个体选择 1 次。

(2) 选择算子每次选择适应度最大的个体 2 次，淘汰适应度最小的 1 个个体，其他个体选择 1 次。

解 (1)求解过程如表 7.5 所示。

表 7.5 例 7.6 遗传算法的处理过程(1)

个体 x_i	基因编码	适应度 f_i	选择次数	选择个体 x_i	配对个体 x_j	交叉位置 k	杂交个体	后代个体
56	1 1 1 0 0 0	3136	3	1 1 1 0 0 0	x_4	3	1 1 1 1 1 1	63
31	0 1 1 1 1 1	961	1	1 1 1 0 0 0	x_3	1	1 1 1 0 0 0	56
24	0 1 1 0 0 0	576	0	1 1 1 0 0 0	x_2	1	1 1 1 0 0 0	56
8	0 0 1 0 0 0	64	0	0 1 1 1 1 1	x_1	3	0 1 1 0 0 0	24
63	1 1 1 1 1 1	3969	3	1 1 1 1 1 1	x_3	1	1 1 1 1 1 1	63
56	1 1 1 0 0 0	3136	1	1 1 1 1 1 1	x_4	2	1 1 1 1 0 0	60
56	1 1 1 0 0 0	3136	0	1 1 1 1 1 1	x_1	1	1 1 1 1 1 1	63
24	0 1 1 0 0 0	576	0	1 1 1 0 0 0	x_2	2	1 1 1 0 1 1	59
63	1 1 1 1 1 1	3969	3	1 1 1 1 1 1	x_2	2	1 1 1 1 1 1	63
60	1 1 1 1 0 0	3969	1	1 1 1 1 1 1	x_1	2	1 1 1 1 1 1	63
63	1 1 1 1 1 1	3600	0	1 1 1 1 1 1	x_4	1	1 1 1 1 1 1	63
59	1 1 1 0 1 1	3481	0	1 1 1 1 1 1	x_3	1	1 1 1 1 1 1	63

第三代群体中的所有个体无差异，算法终止。问题的解为 $x=63$。

(2) 求解过程如表 7.6 所示。

表 7.6　例 7.6 遗传算法的处理过程(2)

个体 x_i	基因编码	适应度 f_i	选择次数	选择个体 x_i	配对个体 x_j	交叉位置 k	杂交个体	后代个体
56	1 1 1 0 0 0	3136	2	1 1 1 0 0 0	x_3	3	1 1 1 1 1 1	63
31	0 1 1 1 1 1	961	1	1 1 1 0 0 0	x_4	3	1 1 1 0 0 0	56
24	0 1 1 0 0 0	576	1	0 1 1 1 1 1	x_1	3	0 1 1 0 0 0	24
8	0 0 1 0 0 0	64	0	0 1 1 0 0 0	x_2	3	0 1 1 0 0 0	24
63	1 1 1 1 1 1	3969	2	1 1 1 1 1 1	x_4	2	1 1 1 1 0 0	60
56	1 1 1 0 0 0	3136	1	1 1 1 1 1 1	x_3	2	1 1 1 1 0 0	60
24	0 1 1 0 0 0	576	1	1 1 1 1 0 0	x_2	2	1 1 1 0 1 1	59
24	0 1 1 0 0 0	576	0	0 1 1 0 0 0	x_1	2	0 1 1 0 1 1	27
60	1 1 1 1 0 0	3600	2	1 1 1 1 0 0	x_3	1	1 1 1 1 0 0	60
60	1 1 1 1 0 0	3600	1	1 1 1 1 0 0	x_4	2	1 1 1 1 1 1	63
59	1 1 1 0 1 1	3481	1	1 1 1 1 0 0	x_1	1	1 1 1 1 0 0	60
27	0 1 1 0 1 1	729	0	1 1 1 0 1 1	x_2	2	1 1 1 0 0 0	56
63	1 1 1 1 1 1	3969	2	1 1 1 1 1 1	x_2	2	1 1 1 1 1 1	63
60	1 1 1 1 0 0	3600	1	1 1 1 1 1 1	x_1	2	1 1 1 1 1 1	63
60	1 1 1 1 0 0	3600	1	1 1 1 1 0 0	x_4	1	1 1 1 1 0 0	60
56	1 1 1 0 0 0	3163	0	1 1 1 1 0 0	x_3	1	1 1 1 1 0 0	60
63	1 1 1 1 1 1	3969	2	1 1 1 1 1 1	x_3	2	1 1 1 1 1 1	63
63	1 1 1 1 1 1	3969	1	1 1 1 1 1 1	x_4	1	1 1 1 1 1 0	62
60	1 1 1 1 0 0	3600	1	1 1 1 1 1 1	x_1	2	1 1 1 1 1 1	63
60	1 1 1 1 0 0	3600	0	1 1 1 1 0 0	x_2	1	1 1 1 1 0 1	61
63	1 1 1 1 1 1	3969	2	1 1 1 1 1 1	x_2	1	1 1 1 1 1 1	63
63	1 1 1 1 1 1	3969	1	1 1 1 1 1 1	x_1	1	1 1 1 1 1 1	63
62	1 1 1 1 1 0	3844	1	1 1 1 1 1 1	x_4	1	1 1 1 1 1 0	62
61	1 1 1 1 0 1	3721	0	1 1 1 1 1 0	x_3	1	1 1 1 1 1 1	63
63	1 1 1 1 1 1	3969	2	1 1 1 1 1 1	x_3	1	1 1 1 1 1 1	63
63	1 1 1 1 1 1	3969	1	1 1 1 1 1 1	x_4	4	1 1 1 1 1 1	63
63	1 1 1 1 1 1	3969	1	1 1 1 1 1 1	x_1	1	1 1 1 1 1 1	63
62	1 1 1 1 1 0	3844	0	1 1 1 1 1 1	x_2	4	1 1 1 1 1 1	63

第七代群体中的所有个体无差异，算法终止。问题的解为 $x=63$。

预测预报是人们对客观事物发展变化趋势的一种认识与估计。预测预报对人类社会的重要性早已为人们所认识，“凡事预则立，不预则废”说明了人们对预测的重要性的认识。预测可采用的方法很多，例如，传统的回归分析方法、基于灰色理论的灰色模型(Grey Model)，以及近几年发展起来的遗传算法和人工神经网络模型等。对某个对象系统选择哪一种方法来建立预测预报模型，关键在于用此方法建立的预测模型的计算值与实际值的拟合程度。下面用一个实例来说明遗传算法在建立预测预报模型中的应用。

例 7.7　已知某港口从 1984 年至 2005 年各年的货物年吞吐量(万吨)如表 7.7 所示，应用遗传算法建立该港口货物年吞吐量预测模型。

表 7.7 某港口 1984 年至 2005 年货物年吞吐量 单位:万吨

年度	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994
年吞吐量	720	542	189	714	846	1106	1616	3097	2586	2381	2235
年度	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005
年吞吐量	1819	1985	1854	2011	2636	3185	2816	3187	3926	4427	4550

解 应用遗传算法建立预测模型可分为以下两个步骤。

(1) 建模分析

由表 7.7 给出的 1984 年至 2005 年的货物年吞吐量可作出如图 7.8 所示的实际年吞吐量曲线。由此曲线可见,港口年吞吐量的总趋势在增长,但有准周期性的振荡,因此,年吞吐量预测模型应考虑由曲线总体增长趋势的描述和振荡规律的描述两部分组成。

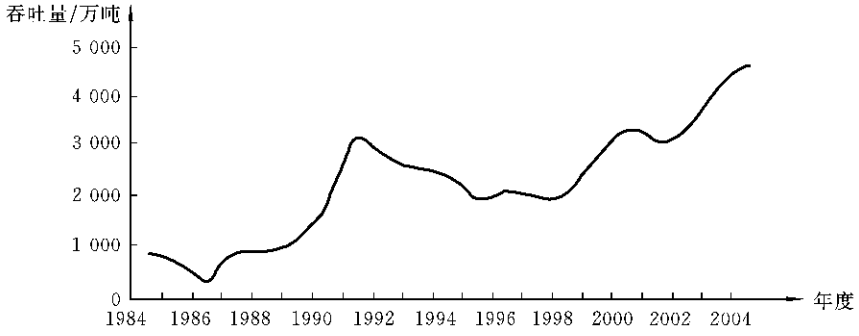

图 7.8 某港口货物实际年吞吐量曲线

① 港口年吞吐量总体增长趋势的描述。

在一个港口的成长发展时期,港口货物年吞吐量的变化情况可分为发生、发展、成熟三个阶段。在发生阶段,随着港口基础设施的逐步建设,货物年吞吐量增长速度较慢,并逐渐加大增长趋势;在发展阶段,随着港口建设的发展和港口经济腹地的扩展,货物年吞吐量增长速度较快,并维持一段时间后进入成熟期;在成熟阶段,其增长速度将逐渐变慢,增长趋于饱和。港口货物的年吞吐量的这种总体发展变化规律可采用 Logistic 生长曲线描述。Logistic 生长曲线是下述非齐次线性常微分方程

$$\frac{\mathrm{d}y}{\mathrm{d}t} = a(b-y)$$

的解

$$y = \frac{b}{1+m\mathrm{e}^{-ct}}$$

式中,各参数在本例中的含义是:y 为年吞吐量,t 为时间(年),a 为年吞吐量增长速度系数,b 为港口年吞吐量的极限值,m 和 c 为与 a、b 有关的参数。

Logistic 生长曲线如图 7.9 所示,可用于反映一般处于成长发展时期的对象系统随时间的生长情况。由于 Logistic 曲线符合港口年吞吐量随港口建设发展的变化规律,所以采用 Logistic 曲线描述港口年吞吐量的总体增长趋势是比较合适的。

考虑到 1984 年以前的年吞吐量应对 Logistic 生长曲线的起点定位及其预测值有“抬高”

的影响，对 Logistic 曲线进行如下修改：

$$y = d + \frac{b}{1 + m\mathrm{e}^{-ct}}$$

② 港口年吞吐量振荡规律的描述。

由图 7.8 所示的实际年吞吐量曲线可见，从 1984 年至 2005 年，实际年吞吐量的变化与图 7.9 所示的 Logistic 曲线有较大差异。实际年吞吐量曲线有较大幅度的振荡，是一个多峰曲线。港口年吞吐量出现振荡多峰的原因是复杂的，但是，并不需要去探讨振荡的原因及其各种原因是如何影响预测模型的，只需要在预测模型中给出振荡规律的大致描述。

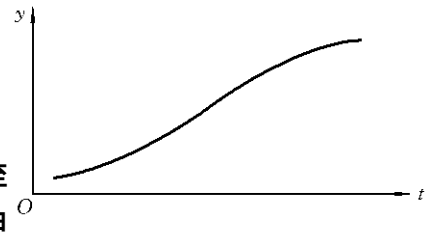

图 7.9　Logistic 生长曲线

由图 7.8 所示的实际年吞吐量曲线的振荡具有准周期性衰减的特征，周期性可以用正弦函数描述，幅值的衰减性可以用负指数函数描述。因此，考虑到年吞吐量曲线的振荡特征后，需要对由 Logistic 曲线表示的年吞吐量总体增长趋势乘以一个振荡修正因子，即有

$$y = \left(d + \frac{b}{1 + m\mathrm{e}^{-ct}}\right) \times \left(1 - p\mathrm{e}^{-gt} \sin\left[\pi\left(\frac{t}{h} + u\right)\right]\right)$$

式中，p 为相对振荡幅值，h 为振荡半周期长度(年)，u 为振荡的初始幅角，g 为振荡衰减系数。

上式中含有 8 个待定参数 b、c、d、m、p、g、h 和 u。采用遗传算法求解这 8 个参数，从而得出港口年吞吐量的预测模型 $y(t)$。

(2) 采用遗传算法求解模型的待定参数

① 编码。

把待定的 8 个参数表示成一维数组 $x=(b,c,d,m,p,g,h,u)$，数组各元为带符号的十进制数。

② 初始群体的随机生成。

设群体规模为 n，随机生成初始群体的 n 个个体 $x_i=(a_{i1},a_{i2},a_{i3},a_{i4},a_{i5},a_{i6},a_{i7},a_{i8})$，$i=1,2,\cdots,n$。个体 x_i 中基因座 k 上的基因值 a_{ik} $(1\leqslant k\leqslant 8)$ 就是数组 x_i 的第 k 元的值。

③ 适应度函数。

待定参数的确定应使得预测模型的年吞吐量计算值 $y(t)$ 与年吞吐量实际值 $y^*(t)$ 的拟合最好，即使 1984 年至 2005 年各年的计算值 $y(t)$ 与实际值 $y^*(t)$ 的误差累计最小。因此，建立适应度函数

$$f(b,c,d,m,p,g,h,u) = \sum_{t=1}^{22} \left| y(t) - y^*(t) \right|$$

④ 选择操作。

为减小计算的复杂程度，可直接选择适应度 f_i 最小的个体 x_i 作为优良个体，对其复制一次，并淘汰适应度最大的一个个体。对规模较大的群体，优良个体被选择的次数可多于 2 次，并相应淘汰多个劣质个体，并保持群体规模 n 不变。

⑤ 杂交操作。

对配对池 $\mathrm{MP}=\{x_1,\cdots,x_i,\cdots,x_j,\cdots,x_n\}$ 中的 n 个个体随机配对进行杂交繁殖。若随机选择 $x_i=(a_{i1},a_{i2},a_{i3},a_{i4},a_{i5},a_{i6},a_{i7},a_{i8})$ 与 $x_j=(a_{j1},a_{j2},a_{j3},a_{j4},a_{j5},a_{j6},a_{j7},a_{j8})$ 配对，则杂交操作为

$$\begin{cases} a'_{ik} = a_{ik}(1-\beta) + \beta a_{jk} \\ a'_{jk} = a_{jk}(1-\beta) + \beta a_{ik} \end{cases} \quad k = 1,2,\cdots,8$$

其中，杂交参数 β 可以是一个随机数，β 的取值范围为 $0<\beta<1$，a'_{ik} 是个体 x_i 与 x_j 杂交生成的后代个体 x_i' 中第 k 个基因的值，a'_{jk} 是个体 x_j 与 x_i 杂交生成的后代个体 x_j' 中第 k 个基因的值。

⑥ 变异操作。

本例选取的变异操作的概率为 1%，即每次对群体中的 $8n$ 个基因座随机选择 $8n\times1\%$ 个基因进行变异操作。若随机选择对基因 a_{ik} 进行变异操作，则变异后的基因为

$$a'_{ik} = (1-\eta)a_{ik}$$

式中，变异参数 η 可以是一个随机数，η 的取值范围为 $0<\eta<1$。

⑦ 算法终止条件。

遗传算法是一种迭代算法，对生成的后代群体的 n 个个体继续进行选择、杂交和变异操作，直至满足算法终止条件。理想的算法终止条件是连续若干代（例如 10 代）不再产生适应度更小的个体，即不再能繁殖出更优良的个体。这个终止条件是比较严格的，需要迭代计算许多次。为了减少算法运算的时空开销，比较一般的终止条件可为：若 $f_i\leqslant\varepsilon$，则算法终止，适应度最小的个体 x_i 即是问题的解。x_i 中的 8 个基因值即分别为待定的 8 个参数值。ε 为指定的拟合精度误差。

由例 7.7 的讨论可见，要建立一个对象系统的时序预测模型，只要能先用一个参数待定的非线性函数来大致描述该对象系统的变化规律，则无论这个非线性函数有多复杂，都可以用一组实际值作为学习实例，采用遗传算法来训练模型，求解出模型的待定参数，从而建立起这个模型。另外，适应度函数和算法终止条件能保证建立的模型的计算值与实际值会有很好的拟合精度。因此，基于遗传算法的时序预测模型的建模方法与传统建模方法比较，前者建立的预测模型会有较高的预测精度。

7.4 人工神经网络

人工神经网络（Artificial Neural Network，ANN）是反映人脑结构及功能的一种抽象数学模型，一个人工神经网络是由大量神经元节点互连而成的复杂网络，用以模拟人类进行知识的表示与存储及利用知识进行推理的行为。

建立一个基于人工神经网络的智能系统是通过学习获取知识后建立的。从本质上讲，人工神经网络的学习是一种归纳学习方式，它通过对大量实例的反复学习，由内部自适应过程不断修改各神经元之间互连的权值，最终使神经网络的权值分布收敛于一个稳定的权值分布。神经网络的互连结构及各连接权值的稳定分布就表示了经过学习获得的知识。这同基于符号的知识表示方法有很大的不同。一个已建立的人工神经网络可用于相关问题的求解，对于特定的输入模式，神经网络通过前向计算可得出一个输出模式，从而得到输入模式的一个特定解。

7.4.1　人工神经元与感知器

人工神经元是人工神经网络的基本单元，神经网络是由大量的神经元互连而成的。感知器(Perceptron)是由美国学者罗森勃拉特(F. Rosenblat)于 1957 年提出的，它是一种具有单层计算单元的神经网络。最初的感知器算法相当于单个神经元，有很大的局限性，但是，它提出了自组织、自学习的思想，后来的许多神经网络模型都是在这种思想指导下建立的。感知器对能够解决的问题有一个收敛的算法，并从数学上给出了严格的证明，它在神经网络的研究中有着重要的意义，推动了人工神经网络研究的发展。

1. 人工神经元模型

人工神经元是对生物神经元的简化和模拟。生物神经元由细胞体、树突和轴突三部分组成，树突是细胞的输入端，轴突是细胞的输出端。树突通过连接其他细胞体的“突触”接受周围细胞由轴突的神经末梢传出的神经冲动；轴突的端部有众多神经末梢作为神经信号的输出端子，用于传出神经冲动。生物神经元具有兴奋与抑制两种状态，当传入的神经冲动使细胞膜电位升高到阈值(约为 40mV)时，细胞进入兴奋状态，产生神经冲动，由轴突输出；若传入的神经冲动使细胞膜电位低入阈值，则细胞进入抑制状态，没有神经冲动输出。

为了模拟生物神经细胞，可以把一个神经细胞简化为一个人工神经元，人工神经元用一个多输入、单输出的非线性节点表示，如图 7.10 所示。

神经细胞 i 的人工神经元的输入输出关系可描述为

$$\begin{cases} I_i = \sum_{j=1}^{n} w_{ji} x_j - \theta_i \\ y_i = f(I_i) \end{cases}$$

式中，x_j 是由细胞 j 传送到细胞 i 的输入量，w_{ji} 是从细胞 j 到细胞 i 的连接权值，θ_i 是细胞 i 的阈值，f 是传递函数，y_i 是细胞 i 的输出量。

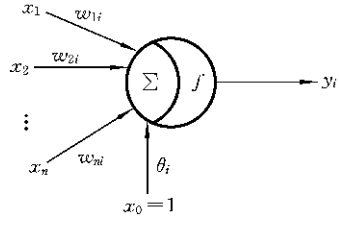

图 7.10　人工神经元

有时为了方便，将 I_i 表示成

$$I_i = \sum_{j=0}^{n} w_{ji} x_j$$

式中，$w_{0i} = -\theta_i$，$x_0 = 1$。

传递函数 f 可为线性函数，或为具有任意阶导数的非线性函数。常见的传递函数有如下几种。

(1) 阶跃函数

阶跃函数的形式为

$$f(x) = \begin{cases} 1 & x \geqslant 0 \\ 0 & x < 0 \end{cases}$$

阶跃函数的图形如图 7.11(a)所示。

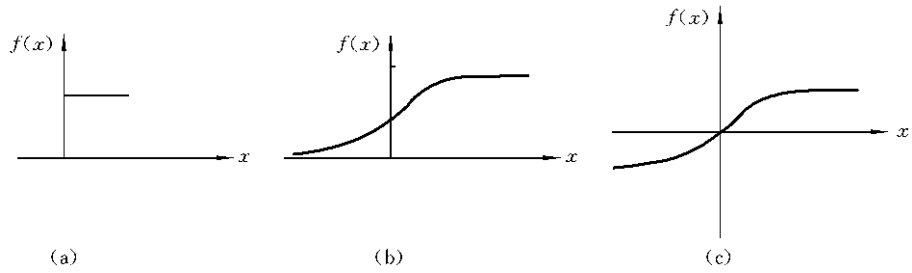

图 7.11 常用传递函数的函数图形

(2) Sigmoid 型函数

Sigmoid 型函数是函数图形如 S 形状的一类可微函数。常用的 Sigmoid 型函数有

$$f(x)=\frac{1}{1+\mathrm{e}^{-x}}$$

$$f(x)=\mathrm{th}(x)=\frac{\mathrm{e}^{x}-\mathrm{e}^{-x}}{\mathrm{e}^{x}+\mathrm{e}^{-x}}$$

函数 $1/(1+\exp(-x))$的函数图形如图 7.11(b)所示，双曲正切函数 $\mathrm{th}(x)$的函数图形如图 7.11(c)所示。双曲正切函数 $\mathrm{th}(x)$的特点是函数图形关于坐标原点对称。

(3) 高斯型函数

在径向基神经网络中，神经元的输入输出关系用高斯函数表示为

$$y_i=\exp\left(-\frac{1}{2\sigma_i^2}\sum_{j=1}^{n}(x_j-w_{ji})^2\right)$$

式中，σ_i^2为标准化参数。

2. 感知器及其学习算法

感知器是一种具有单层计算单元的前向神经网络模型，如图 7.12 所示，其中，n 个输入 $x_1,x_2,\cdots,x_n$均为实数，$w_1,w_2,\cdots,w_n$分别是 n 个输入的加权，θ 是感知器的阈值，f 是感知器的传递函数，y 是感知器的输出。

感知器的输出与输入的关系可描述为

$$y=\begin{cases}1, \sum_{i=1}^{n}w_i x_i-\theta\geqslant 0\\ 0, \sum_{i=1}^{n}w_i x_i-\theta<0\end{cases}$$

实际上，感知器的传递函数 f 是阶跃函数。

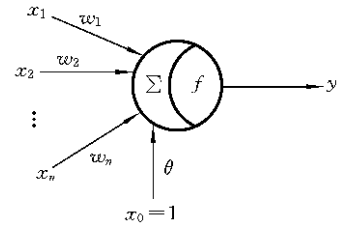

图 7.12 感知器

感知器的学习过程为：由已知的 N 个实例(X_k,Y_k^*)，$k=1,2,\cdots,N$，组成一组学习样本，其中，实例 k 的输入 X_k可表示为一个 n 元向量 $X_k=(x_{1k},x_{2k},\cdots,x_{ik},\cdots,x_{nk})$，实例 k 的期望输出为单一输出 $Y_k^*=y_k^*$。通过对实例(X_k,Y_k^*)的学习，调整可表示为 n 元数组的权值分布 $W=(w_1,w_2,\cdots,w_n)$中各连接权的值及阈值 θ。从理论上讲，在 n 维欧氏空间 E^n中，若点集$\{X_k\mid f(X_k)=1\}$与点集$\{X_k\mid f(X_k)=0\}$是线性可分的，即存在一个超平面将两个点集分开，那么感知器学习过程在有限次

迭代后可收敛于一个稳定的权值分布和阈值。

如果把阈值 θ 表示成 $\theta=-w_0$，相应对每一个输入向量 X_k 也增加一个分量 $x_{0k}=1$，那么，感知器的计算输出可以表示为 $y_k=f(\sum_{i=0}^{n} w_i x_{ik})$。

感知器的学习过程就是通过对实例集 $\{(X_k,Y_k^*)\}$ 的反复学习，不断地修改权值分布 W 中各连接权 w_i 的值的过程，直至感知器对各实例的计算输出 $y_k=f(\sum_{i=0}^{n} w_i x_{ik})$ 满足各实例的期望输出，即有 $|y_k-y_k^*|\leqslant\varepsilon, k=1,2,\cdots,N$，$\varepsilon$ 为指定的输出误差。此时，称学习过程收敛于一个稳定的权值分布。修改权值分布 W 是迭代进行的，设 t 为迭代次数，初始化权值分布为 $W(0)$，第 $t+1$ 次迭代将权值分布 $W(t)$ 修改为 $W(t+1)$。显然，权值分布 W 的修改方法是感知器学习算法的核心。

感知器学习算法

(1) 初始化权值分布 $W(0)$，对 $W(0)=(w_0(0),w_1(0),\cdots,w_i(0),\cdots,w_n(0))$ 中的各权值分别赋给较小的随机非零值。初始化学习实例序号 $k=1$，初始化迭代次数 $t=0$。

(2) 取第 k 个学习实例 (X_k,Y_k^*)，由给定的学习实例 (X_k,y_k^*) 的输入向量 $X_k=(1,x_{1k},x_{2k},\cdots,x_{ik},\cdots,x_{nk})$，以及权值分布 $W(t)=(w_0(t),w_1(t),\cdots,w_i(t),\cdots,w_n(t))$，计算感知器的输出 $y_k(t)$ 为

$$y_k(t)=f\left(\sum_{i=0}^{n} w_i(t)x_{ik}\right)$$

(3) 修改权值分布 $W(t)$ 为 $W(t+1)$。权值修改的计算为

$$w_i(t+1)=w_i(t)+\eta(y_k^*-y_k(t))x_{ik},\quad i=0,1,\cdots,n$$

其中，学习率 η 为设定的修改速度控制参数，$0<\eta\leqslant 1$。

(4) 若对全部学习实例都有 $y_k=y_k^*$，即 W 不再被修改，说明 W 已经收敛于稳定的权值分布，则学习过程终止；否则，$t=t+1$，$k=k+1(\mathrm{mod}\ N)$，转步骤(2)。

权值修改计算公式表明：若感知器的计算输出 $y_k(t)$ 与实例给定的期望输出 y_k^* 相等，即 $y_k(t)=y_k^*$，则 $w_i(t)$ 不变；若 $y_k(t)<y_k^*$，则增大正输入 $(x_{ik}>0)$ 的权值 $w_i(t)$，减小负输入 $(x_{ik}<0)$ 的权值 $w_i(t)$；若 $y_k(t)>y_k^*$，则对权值修改的情况相反，减小正输入的权值，增大负输入的权值。

对权值修正的速度由 η 的取值决定，若设定的 η 值较大，则每次对权值的修改量较大。若 η 取值过大，则权值的修改过程将可能围绕稳定权值振荡，反而延长了修改过程收敛的时间。若 η 取值过小，则权值的修改过程将缓慢地收敛于稳定权值，同样也延长了修改过程收敛的时间。一般来说，学习率 η 可以设定为 $0.3\leqslant\eta\leqslant 0.9$；还可以采用变学习率方法来提高学习过程的收敛速度，即在学习的初期，学习率 η 可设定较大，随着迭代次数 t 的增加而逐步减小学习率 η 的值。

实际上，学习算法的终止条件可以设定宽松一些，例如，可设定为满足以下两个条件之一即可终止：对全部学习实例满足 $|y_k^*-y_k|\leqslant\varepsilon$，其中，$\varepsilon$ 是初始设定的允许误差；或者迭代次数 t 达到初始设定的最大迭代次数 $t_{\max}$。

需要特别指出的是，只有当提供的学习实例集是线性可分时，学习过程才会经有限次迭代

而收敛，才能得出稳定的权值分布 W，因此，需要对感知器及其学习算法进行改进，传递函数也不采用阶跃函数，而改用可微函数，例如采用 Sigmoid 型函数。

7.4.2 人工神经网络模型

自 1943 年美国心理学家麦克洛奇(Mc Culloch)和数学家皮兹(Pitts)提出神经元的抽象数学模型(MP 模型)以来，经过多年的发展，人工神经网络理论及其技术已经取得了大量的研究成果，并在模式识别、图像处理、专家系统、组合优化、智能控制及机器人等领域得到了广泛的应用，提出了各种神经网络模型。这里，讨论几种常用的神经网络模型。

1. 神经网络的互连结构

根据神经网络中神经元之间互连的结构不同，把神经网络可以分成以下几种类型。

(1) 不含反馈的前向网络

不含反馈的前向网络的结构形态如图 7.13(a)所示。网络中的神经元分层排列，接受输入量的神经元节点组成输入层，产生输出量的神经元节点组成输出层，中间层亦称为隐层，可以有若干层隐层。每一层的神经元只接受前一层神经元的输入，输入向量经过各层的顺序变换后，由输出层得到输出向量。

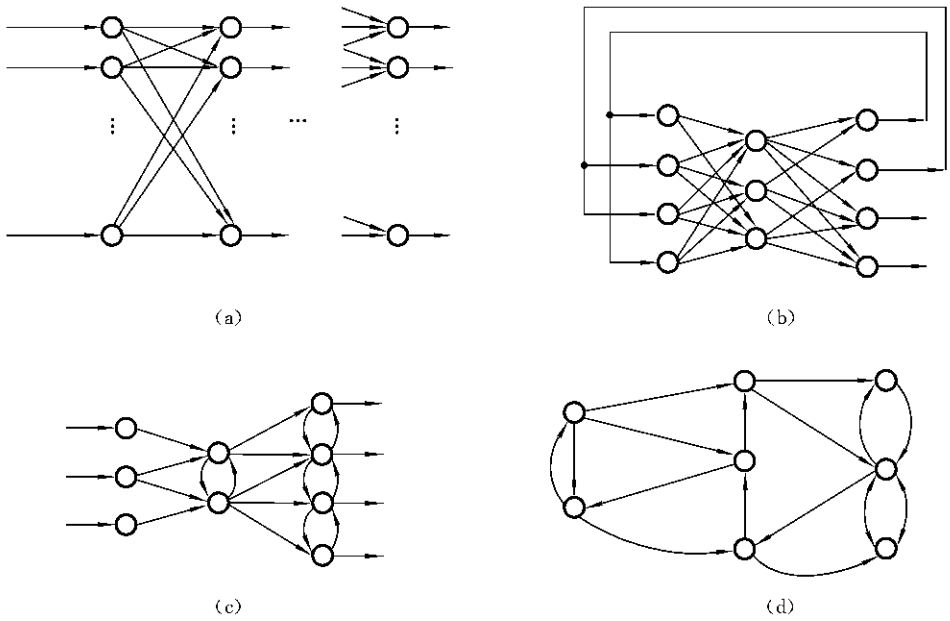

图 7.13 人工神经网络互连结构

(2) 从输出层到输入层有反馈的前向网络

从输出层到输入层有反馈的前向网络简称为反馈神经网络，其结构形态如图 7.13(b)所示。网络中的神经元也是分层排列，但是输入层神经元在学习过程中接受输出层神经元或部分输出层神经元的反馈输入。

在反馈神经网络中，由输入数据决定反馈神经网络的初始状态，经过一系列状态转移后反馈神经网络逐渐收敛于一个稳定状态，这个稳定状态就是反馈神经网络的最后计算结果。不含反馈的前向网络（简称前向神经网络）是一种强有力的学习系统，结构简单且易于编程。从系统观点来看，它是一种静态非线性映射，通过大量的简单的非线性神经元的复合映射，可获得复杂的非线性映射处理能力，但是，它缺乏丰富的动力学行为。从系统观点来看，反馈神经网络是反馈动力学系统，它比前向神经网络具有更强的计算能力。反馈神经网络的一个重要特点就是它应具有稳定状态，稳定性是研究反馈神经网络的最重要的问题之一。

(3) 层内有相互结合的前向网络

层内有相互结合的前向网络的结构形态如图 7.13(c)所示。每一层的神经元除接受前一层神经元的输入之外，也可接受同一层神经元的输入。

通过层内神经元之间的相互结合，可以实现同层神经元之间横向的抑制或兴奋机制，从而限制一层内能同时动作的神经元的个数，或者实现把一层内的神经元分为若干组，每一组作为一个整体来动作。例如，可以利用横向抑制机制把同层中具有最大输出的神经元挑选出来，而抑制其他神经元使其处于无输出的状态。

(4) 相互结合型网络

相互结合型网络的结构形态如图 7.13(d)所示，这种网络中任意两个神经元之间都可能有连接。在不含反馈的前向网络中，输入信号一旦通过某个神经元就将输出这个信号的变换值。但是，在相互结合型网络中，输入信号要在神经元之间反复往返传递，网络处于一种不断改变状态的动态之中。从某一初态开始，经过若干次的状态变化，网络才会到达某种稳定状态，根据网络的结构和神经元的映射特性，网络还有可能进入周期振荡或其他如混沌等平衡状态。

以上 4 种类型的神经网络，从结构形态来看，(1)、(2)、(3)可看成是(4)的一种特殊形态，但是，不论从神经网络的计算和学习机制来看，还是从网络的应用来看，这 4 种类型的神经网络都是有很大的区别的。

2. BP 神经网络

BP 神经网络(Backpropugation Neural Network)是一种单向传播的多层前向神经网络，其结构如图 7.14 所示。BP 网络除有输入层和输出层之外，还有一层或多层隐层，同层节点间无任何连接，每个节点都是单个神经元，神经元的传递函数通常为 Sigmoid 型函数。有时，输入层或输出层的神经元的传递函数选取线性函数。由于同层节点间无任何耦合，因此，每一层的神经元只接受前一层神经元的输入，每一层神经元的输出只影响后一层神经元的输出。

网络的输入数据 $X=(x_1,x_2,\cdots,x_n)$ 从输入层依次经过各隐层节点，然后到达输出层节

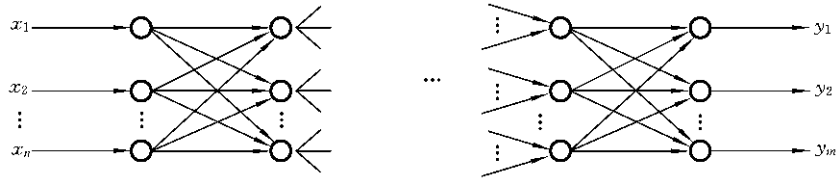

图 7.14　BP 神经网络

点，从而得到输出数据 $Y=(y_1,y_2,\cdots,y_m)$。可以把 BP 神经网络看成是一个从输入到输出的高度非线性映射，即 $f:R^n\to R^m$，$f(X)=Y$。

对于学习实例集$\{(X_k,Y_k^*)\}$，可以认为所有实例的输入 $X_k\in R^n$和实例的期望输出 $Y_k^*\in R^m$之间存在某一映射函数 g，使得 $g(X_k)=Y_k^*$，$k=1,2,\cdots,N$。

现在的问题是：BP 神经网络是否存在这样的非线性映射 f，且 f 高度地逼近映射函数 g？如果 f 能逼近函数 g，就可以用 BP 神经网络来实现许多复杂函数的映射关系，使得许多难以通过求得连续函数解决的问题转化为用神经网络来解决。

下述的两个定理可以回答这个问题。

定理 7.1 （Kolmogorov 定理）给定任一连续函数 $f:[0,1]^n\to R^m$，f 可以精确地用一个三层前向神经网络实现，此网络的第一层即输入层有 n 个神经元，中间层有 $2n+1$ 个神经元，第三层即输出层有 m 个神经元。

定理 7.2 给定任意 $\varepsilon>0$，对于任意的 L_2 型连续函数 $f:[0,1]^n\to R^m$，存在一个三层 BP 神经网络，它可在任意 ε 平方误差精度内逼近 f。

这两个定理的证明见有关参考文献。它们说明了任一连续函数 $f:[0,1]^n\to R^m$都可以用一个结构为 $n\times(2n+1)\times m$ 的三层前向神经网络来精确地逼近。

定理 7.1 和定理 7.2 不仅证明了映射网络的存在，而且说明了映射网络的结构。但是，在实际应用中，有时需要使用具有多个隐层的 BP 神经网络。如何合理地选取 BP 网络的隐层数及隐层的节点数，目前尚无准确有效的理论和方法。BP 网络输入层的节点数由学习实例的输入向量 $X_k=(x_{1k},x_{2k},\cdots,x_{nk})$的输入量的个数 n 决定，输出层的节点数由学习实例的输出向量 $Y_k^*=(y_{1k}^*,y_{2k}^*,\cdots,y_{mk}^*)$的输出量的个数 m 决定。

7.4.3 BP 神经网络的学习算法

人工神经网络卓越的非线性映射能力一是来自于网络中各神经元的非线性传递函数，二是来自于各神经元之间的连接权分布。神经网络的连接权分布一般不能预先准确地确定，而是通过对学习实例的反复学习来逐渐调整和修改权值分布，使神经网络收敛于稳定状态，从而完成学习过程。一个稳定的神经网络就是一个特定的知识表示，可用于对相应领域问题的求解。

1. 神经网络的两类学习方法

神经网络的学习方法可分为两类：有教师学习方法和无教师学习方法。

(1) 无教师学习方法

无教师学习的基本思想是：当输入的学习实例进入神经网络后，网络按预先设定的规则自动调整权值。

常见的无教师学习方法是 Hebb 学习规则，它来源于心理学家赫布（Hebb）关于生物神经元的下述学习假设：当两个神经元同时处于兴奋状态时，它们之间的连接应该加强。若设 $w_{ij}(t)$表示神经元 i 连接神经元 j 的当前权值，I_i和 I_j分别表示神经元 i 和神经元 j 的激活水

平，那么，关于神经元的 Hebb 学习规则可表示为

$$w_{ij}(t+1) = w_{ij}(t) + I_i I_j$$

对于图 7.10 所示的人工神经元，有

$$\begin{cases} I_i = \sum_{j=1}^{n} w_{ji} x_j - \theta_i \\ y_i = f(I_i) \end{cases}$$

于是，图 7.10 所示的人工神经元的 Hebb 学习规则可表示为

$$w_{ij}(t+1) = w_{ij}(t) + y_i y_j$$

(2) 有教师学习方法

有教师学习的基本思想是：对实例 k 的输入，由神经网络根据当前的权值分布 $W(t)$ 计算网络的输出 $Y_k(W)$，把网络的计算输出 $Y_k(W)$ 与实例 k 的期望输出 Y_k^* 进行比较，根据两者之间的差的某个函数的值来调整网络的权值分布，最终使差的函数值达到最小。

前述的感知器及其学习算法就是一种有教师学习方法。

常见的有教师学习方法是梯度下降法，其基本思想是根据实例 k 的期望输出 Y_k^* 与网络计算输出 $Y_k(W)$ 的误差的平方最小的原则来修改权值分布。

定义误差函数 $J(W)$ 为

$$J(W) = \frac{1}{2}(Y_k{}^* - Y_k(W))^2$$

式中，Y_k^* 是实例 k 的输出，$Y_k(W)$ 是由当前权值分布 $W(t)$ 对实例 k 的输入经网络计算的输出。梯度下降法就是沿着 $J(W)$ 的负梯度方向不断修正 W 的值，直至 $J(W)$ 达到最小值。

梯度下降法对权值的修改可表示为

$$W(t+1) = W(t) + \eta(t)\left(-\frac{\partial J(W)}{\partial W}\right)\Big|_{W=W(t)}$$

式中，$\eta(t)$ 是可变学习率，用以控制权值修改的速度。在学习过程中，$\eta(t)$ 也可以不变，取成一个常量 η，$0<\eta<1$。

2. BP 学习算法的计算方法

1982 年，鲁梅尔哈特(D. Rumelhart)和麦克莱伦德(McClelland)及他们的同事们成立了一个 PDP(Parallel Distributed Processing)小组，研究并行分布信息处理方法，探索人类认知的微结构。1985 年，他们提出了 BP 神经网络及其学习算法，这个 BP 网络的学习算法称为误差反向传播算法或 BP 学习算法。

BP 学习算法是图 7.14 所示的 BP 神经网络的学习算法。BP 学习算法是一种迭代算法，一次学习过程由输入数据的正向传播和误差的反向传播两个子过程组成。设有 N 个学习实例 (X_k, Y_k^*)，$k=1,2,\cdots,N$，对实例 (X_k, Y_k^*)，在正向传播过程中，实例 k 的输入向量 $X_k=(x_{1k}, x_{2k}, \cdots, x_{nk})$ 从输入层的 n 个节点输入，经隐层逐层处理，由输出层的 m 个节点的输出端得到实例 k 的网络计算输出向量 $Y_k=(y_{1k}, y_{2k}, \cdots, y_{mk})$，比较 Y_k 和实例 k 的期望输出向量 $Y_k^*=(y_{1k}^*, y_{2k}^*, \cdots, y_{mk}^*)$，若 N 个学习实例的计算输出都达到期望的结果，则学习过程结束；否则，进入误差反向传播过程，把 Y_k 与 Y_k^* 的误差由网络输出层向输入层反向传播，在反向传播过程中，修改各层神经元的连接权值。

设 $I_{jk}^{(l)}$ 表示实例 k 的输入向量 X_k 传播到第 l 层节点 j 时的输入，$O_{jk}^{(l)}$ 表示第 l 层节点 j 的输出，$w_{ij}^{(l-1)}$ 为第 $l-1$ 层的节点 i 连接第 l 层节点 j 的权值，$n^{(l-1)}$ 为第 $l-1$ 层的节点数，f 为节点神经元的传递函数，BP 网络的神经元传递函数一般使用可微的 Sigmoid 型函数。BP 网络神经元的输入、输出关系，有

$$I_{jk}^{(l)}=\sum_{i=1}^{n(l-1)}w_{ij}^{(l-1)}O_{ik}^{(l-1)} \tag{7-1}$$

$$O_{jk}^{(l)}=f(I_{jk}^{(l)}) \tag{7-2}$$

实例 k 对节点 j 的期望输出 $O_{jk}^{*(l)}$ 与节点 j 对实例 k 的网络计算输出 $O_{jk}^{(l)}$ 的误差定义为

$$E_{jk}^{(l)}=\frac{1}{2}(O_{jk}^{*(l)}-O_{jk}^{(l)})^2 \tag{7-3}$$

若第 l 层是 BP 网络的输出层，即节点 j 是输出节点，则 $O_{jk}^{*(l)}=y_{jk}^{*}$，$O_{jk}^{(l)}=y_{jk}$，实例 k 的输出误差为

$$E_{jk}^{(l)}=\frac{1}{2}(y_{jk}^{*}-y_{jk})^2 \tag{7-4}$$

若输出层的 m 个输出节点的计算输出都分别满足实例 k 的 m 个期望输出，即有 $E_{jk}^{(l)}\leqslant\varepsilon$，$j=1,2,\cdots,m$，则学习过程结束，$\varepsilon$ 为指定的允许误差；否则，由误差反向传播过程修改权值分布 W。

按误差的负梯度来修改权值，即

$$\begin{cases}w_{ij}^{(l-1)}=w_{ij}^{(l-1)}+\Delta w_{ij}^{(l-1)}\\ \Delta w_{ij}^{(l-1)}=-\eta\dfrac{\partial E_{jk}^{(l)}}{\partial w_{ij}^{(l-1)}}\end{cases} \tag{7-5}$$

其中，η 为学习率，$0<\eta<1$。

由(7-4)式、(7-2)式和(7-1)式，可有

$$\frac{\partial E_{jk}^{(l)}}{\partial w_{ij}^{(l-1)}}=\frac{\partial E_{jk}^{(l)}}{\partial O_{jk}^{(l)}}\frac{\partial O_{jk}^{(l)}}{\partial I_{jk}^{(l)}}\frac{\partial I_{jk}^{(l)}}{\partial w_{ij}^{(l-1)}}=\delta_{jk}^{(l)}\frac{\partial I_{jk}^{(l)}}{\partial w_{ij}^{(l-1)}}=\delta_{jk}^{(l)}O_{ik}^{(l-1)} \tag{7-6}$$

其中，

$$\delta_{jk}^{(l)}=\frac{\partial E_{jk}^{(l)}}{\partial I_{jk}^{(l)}}=\frac{\partial E_{jk}^{(l)}}{\partial O_{jk}^{(l)}}\frac{\partial O_{jk}^{(l)}}{\partial I_{jk}^{(l)}}=\frac{\partial E_{jk}^{(l)}}{\partial O_{jk}^{(l)}}f'(I_{jk}^{(l)}) \tag{7-7}$$

为了得出计算 $\delta_{jk}^{(l)}$ 的表达式，进行下述讨论。

① 若第 l 层是输出层，则由(7-4)式可得出：

$$\frac{\partial E_{jk}^{(l)}}{\partial O_{jk}^{(l)}}=\frac{\partial E_{jk}^{(l)}}{\partial y_{jk}}=-(y_{jk}^{*}-y_{jk}) \tag{7-8}$$

由(7-7)式和(7-8)式可得(7-9)式：

$$\delta_{jk}^{(l)}=-(y_{jk}^{*}-y_{jk})f'(I_{jk}^{(l)}) \tag{7-9}$$

由(7-5)式、(7-6)式和(7-9)式可得(7-10)式：

$$\Delta w_{ij}^{(l-1)}=-\eta\delta_{jk}^{(l)}O_{ik}^{(l-1)}=\eta(y_{jk}^{*}-y_{jk})f'(I_{jk}^{(l)})O_{ik}^{(l-1)} \tag{7-10}$$

② 若第 l 层不是输出层，则根据 BP 网络的误差反向传播定义，第 l 层节点 j 的误差 $E_{jk}^{(l)}$ 对节点 j 的输出 $O_{jk}^{(l)}$ 的变化率为第 $l+1$ 层的 $n^{(l+1)}$ 个节点的各节点误差对其输出的变化率之和。若 $E_{qk}^{(l+1)}$ 表示第 $l+1$ 层的节点 q 的误差，$O_{qk}^{(l+1)}$ 表示节点 q 的计算输出，则有

$$\frac{\partial E_{jk}^{(l)}}{\partial O_{jk}^{(l)}} = \sum_{q=1}^{n^{(l+1)}} \frac{\partial E_{qk}^{(l+1)}}{\partial O_{qk}^{(l+1)}} \tag{7-11}$$

对第 $l+1$ 层的节点 q，类似第 l 层的节点 j 的(7-1)式、(7-2)式、(7-3)式和(7-7)式，有

$$I_{qk}^{(l+1)} = \sum_{j=1}^{n^{(l)}} w_{jq}^{(l)} O_{jk}^{(l)} \tag{7-12}$$

$$O_{qk}^{(l+1)} = f(I_{qk}^{(l+1)}) \tag{7-13}$$

$$E_{qk}^{(l+1)} = \frac{1}{2}(O_{qk}^{*(l+1)} - O_{qk}^{(l+1)})^2 \tag{7-14}$$

$$\delta_{qk}^{(l+1)} = \frac{\partial E_{qk}^{(l+1)}}{\partial I_{qk}^{(l+1)}} \tag{7-15}$$

因此，可把(7-11)式表示为

$$\frac{\partial E_{jk}^{(l)}}{\partial O_{jk}^{(l)}} = \sum_{q=1}^{n^{(l+1)}} \frac{\partial E_{qk}^{(l+1)}}{\partial O_{qk}^{(l+1)}} = \sum_{q=1}^{n^{(l+1)}} \frac{\partial E_{qk}^{(l+1)}}{\partial I_{qk}^{(l+1)}} \times \frac{\partial I_{qk}^{(l+1)}}{\partial O_{qk}^{(l)}} = \sum_{q=1}^{n^{(l+1)}} \delta_{qk}^{(l+1)} w_{jq}^{(l)} \tag{7-16}$$

由(7-7)式和(7-16)式可得

$$\delta_{jk}^{(l)} = f'(I_{jk}^{(l)}) \sum_{q=1}^{n^{(l+1)}} \delta_{qk}^{(l+1)} w_{jq}^{(l)} \tag{7-17}$$

由(7-5)式、(7-6)式和(7-17)式可得

$$\Delta w_{ij}^{(l-1)} = -\eta \delta_{jk}^{(l)} O_{jk}^{(l-1)} \tag{7-18}$$

(7-9)式和(7-17)式给出了误差反向传播时，计算输出层各节点和隐层各节点的 δ 的关系，由(7-9)式计算出输出层的各节点的 δ 值后，就可由(7-17)式反向逐层计算出各隐层的所有节点的 δ 值。由各节点的 δ 值，用(7-10)式或(7-18)式，就可计算出各节点的权值修改量 Δw，从而对权值进行修改。

3. BP 学习算法

根据误差反向传播计算方法的分析，可给出下述 BP 学习算法。

BP 学习算法

(1) 输入 N 个学习实例 (X_k, Y_k^*)，$k=1,2,\cdots,N$。

(2) 建立 BP 网络结构。由学习实例输入向量 X_k 的长度 n 确定网络输入层节点数为 n，由学习实例输出向量 Y_k^* 的长度 m 确定网络输出层节点数为 m。确定网络层数 $L \geqslant 3$ 和各层节点数，第 l 层的节点数为 $n^{(l)}$，且 $n^{(1)}=n$，$n^{(L)}=m$。定义各层间连接权矩阵，第 l 层连接第 $l+1$ 层的连接权矩阵为 $W^{(l)} = [w_{ij}^{(l)}]_n^{(l)} \times n^{(l+1)}$，$l=1,2,\cdots,L-1$，初始化各连接权矩阵的元素值。设定各层节点的传递函数 f 及其导数函数 f'。

(3) 输入允许误差 ε 和学习率 η，初始化迭代计算次数 $t=1$，学习实例序号 $k=1$。

(4) 取第 k 个学习实例 (X_k, Y_k^*)，$X_k = (x_{1k}, x_{2k}, \cdots, x_{nk})$，$Y_k^* = (y_{1k}^*, y_{2k}^*, \cdots, y_{mk}^*)$。

(5) 由 X_k 进行正向传播计算。

计算输入层各节点的输出为

$$O_{jk}^{(l)}(t) = f(x_{jk}), \quad j = 1,2,\cdots,n$$

逐层计算各层的各节点的输入和输出为

$$I_{jk}^{(l)}(t)=\sum_{i=1}^{n^{(l-1)}}w_{ij}^{(l-1)}(t)O_{ik}^{(l-1)}(t)$$

$$O_{jk}^{(l)}(t)=f(I_{jk}^{(l)}(t))$$

$$j=1,2,\cdots,n^{(l)},l=2,\cdots,L$$

(6) 计算输出层(第 L 层)的各输出节点误差为

$$y_{jk}(t)=O_{jk}^{(L)}(t)$$

$$E_{jk}(t)=\frac{1}{2}(y_{jk}^{*}-y_{jk}(t))^{2},j=1,2,\cdots,m$$

(7) 若对 N 个学习实例,有 $E_{jk}\leqslant\varepsilon,j=1,2,\cdots,m$,则学习过程结束;否则,进行误差反向传播修改各连接权矩阵。

(8) 误差反向传播计算。

修改第 $L-1$ 层隐层至输出层(第 L 层)的连接权矩阵:

$$\delta_{jk}^{(L)}(t)=-(y_{jk}^{*}-y_{jk}(t))f'(I_{jk}^{(L)}(t))$$

$$\Delta w_{ij}^{(L-1)}(t)=\eta\delta_{jk}^{(L)}(t)O_{ik}^{(L)}(t)$$

$$W_{ij}^{(L-1)}(t+1)=w_{ij}^{(L-1)}(t)+\Delta w_{ij}^{(L-1)}(t)$$

$$j=1,2,\cdots,m,\quad i=1,2,\cdots,n^{(L-1)}$$

反向逐层修改连接各隐层的连接权矩阵:

$$\delta_{jk}^{(l)}(t)=f'(I_{jk}^{(l)}(t))\sum_{q=1}^{n^{(l+1)}}\delta_{qk}^{(l+1)}(t)w_{jq}^{(l)}(t)$$

$$\Delta w_{ij}^{(l-1)}(t)=-\eta\delta_{jk}^{(l)}(t)O_{ik}^{(l-1)}(t)$$

$$W_{ij}^{(l-1)}(t+1)=w_{ij}^{(l-1)}(t)+\Delta w_{ij}^{(l-1)}(t)$$

$$j=1,2,\cdots,n^{(l)},i=1,2,\cdots,n^{(l-1)},l=L-1,\cdots,2,1$$

(9) $k=k+1(\mathrm{mod}\ N),t=t+1$,转步骤(4)。

7.4.4 BP 学习算法的改进

BP 学习算法具有理论依据充分、推导过程严谨、物理概念清晰及通用性好等优点,使它至今仍然是 BP 网络的主要学习算法。但 BP 算法也存在一些不足之处,主要有

① BP 学习算法的收敛速度慢,对此,已提出各种改进的方法。

② BP 学习算法采用梯度下降法进行学习,这就有可能出现局部极小问题。当局部极小点产生时,BP 算法所求得的解就不是全局最优解。

③ 网络隐层的层数及节点个数的优化选择尚无理论上的指导。

④要求所有学习实例(X,Y^{*})的输入 $X=(x_1,x_2,\cdots,x_n)$的特征数目 n 相同。

为了解决 BP 学习算法陷入局部极小的问题,通常使用一些全局最优化算法与 BP 算法相结合的方法。这里,主要针对 BP 学习算法的①和③的不足之处,讨论对 BP 学习算法的改进方法。

1. 提高 BP 学习算法收敛速度的方法

BP 学习算法的主要问题是学习过程的收敛速度比较慢,对于提供的 N 个学习实例需要

反复学习，迭代计算次数 t 通常为几百次至几千次，才能使所有输出节点的误差小于指定的允许误差 ε。为提高 BP 学习算法的收敛速度，可采用下述方法改进 BP 算法。

(1) 变学习率方法

学习率 η 的取值为 $0<\eta<1$，η 越大，对权值的修改越大。但是，当 BP 网络的权值分布 W 已接近稳定的权值分布时，过大的 η 将使对权值分布 W 的修改过程发生振荡，难以收敛于稳定的权值权分布。若 η 较小，则每次迭代计算对权值的修改较小，修改过程将缓慢地收敛于稳定的权值分布。

变学习率方法是指：在学习过程的初期，学习率取较大的值，随着学习过程的进行，逐渐减小学习率。

一种简单易行的变学习率方法可以设定学习率是迭代学习次数 t 的线性函数，使学习率随学习次数 t 的增长而线性递减。假设学习过程的最大迭代学习次数为 $t_{\max}$，第 1 次学习的学习率为 $\eta(1)$，第 $t_{\max}$ 次学习的学习率为 $\eta(t_{\max})$，每经过一次学习后，使学习率变化量为 $\Delta\eta=(\eta(1)-\eta(t_{\max}))/t_{\max}$，那么，学习率可表示为

$$\eta(t)=\eta(1)-\frac{\eta(t_{\max})}{t_{\max}}\times t$$

例如，若估计学习过程收敛的最大迭代学习次数为 $t_{\max}=800$，设定第 1 次学习的学习率为 $\eta(1)=0.9$，学习率最小值为 $\eta(t_{\max})=0.1$，则每次学习后学习率递减 $\Delta\eta=0.001$。

也可以在连续地对 N 个学习实例都学习一次后，才使学习率线性递减 $\Delta\eta$。若将对 N 个学习实例依次学习一次称为一遍，那么，$t_{\max}$ 次学习可分为 $t'_{\max}=\lceil t_{\max}/N\rceil$ 遍学习，遍学习的学习遍数为 $t'=\lceil t/N\rceil$，那么，学习率可表示为

$$\eta(t')=\eta(1)-\frac{\eta(1)-\eta(t'_{\max})}{t'_{\max}}\times t'$$

根据 BP 网络学习过程的收敛特征来设计适当的变学习率函数，既可加快学习速度，又不至于引起学习过程的振荡。变学习率方法是 BP 学习方法的主要改进方法之一，变学习率函数的设计是影响改进效果的决定因素。

(2) 附加冲量方法

附加冲量法也称为惯性校正法。附加冲量法要求在第 t 次计算连接权的修改量 $\Delta w_{ij}^{(l-1)}(t)$ 时，附加一项冲量，冲量为第 $t-1$ 次计算的相应连接权的修改量 $\Delta w_{ij}^{(l-1)}(t-1)$ 的加权，即将(7-18)式修改为

$$\Delta w_{ij}^{(l-1)}(t)=-\eta\delta_{jk}^{(l)}O_{ij}^{(l-1)}+\beta\Delta w_{ij}^{(l-1)}(t-1) \tag{7-19}$$

其中，β 称为冲量系数，$0<\beta<1$。

附加冲量使权值的修改增大，合适的冲量系数 β 可加快学习过程的收敛速度。但是，β 值过大，也会引起学习过程的振荡。一般取 $0.7<\beta<0.9$。当然，也可以使冲量系数 β 在学习过程中是可变的，那么，冲量系数函数的设计将直接影响学习过程的改进效果。变学习率和附加冲量都是通过在学习过程中，调整连接权修改量 Δw 来希望提高学习过程的收敛速度。无论采用什么方法来调整连接权修改量 Δw，都有一个适量调整的问题。适量调整对学习过程的影响可以用图 7.15 来说明。

如果每次学习的权值调整量 Δw 较小，即对 Δw 欠调，则学习过程缓慢地收敛于稳定的权

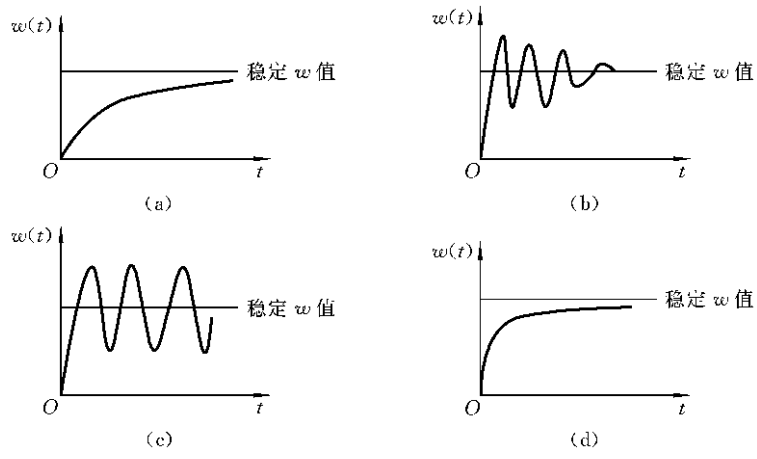

图 7.15 调整 Δw 对学习过程的影响

(a)欠调的收敛过程;(b)超调的衰减振荡过程;(c)超调的发散振荡过程;(d)适量调整的收敛过程

值,如图 7.15(a)所示,学习过程费时较多。如果每次学习的权值调整量 Δw 过大,则学习过程是一个衰减振荡过程,如图 7.15(b)所示。这是对 Δw 超调而导致学习过程振荡,即使振荡过程是逐渐衰减的,也需要较长的时间才会达到稳定状态。过度的超调将导致学习过程成为一个发散振荡过程,如图 7.15(c)所示,那么,学习过程不能收敛于稳定状态。理想的学习过程如图 7.15(d)所示,适量调整 Δw,使学习过程尽可能快地收敛于稳定的权值分布。因此,需要研究 Δw 的调整方法来获得适当的 Δw 调整量。

2. 动态结构 BP 网络学习算法

定理 7.1 给出三层 BP 网络的结构为:输入层为 n 个节点,隐层为 $2n+1$ 个节点,输出层为 m 个节点。在 BP 网络的实际应用中,输入层节点数 n 和输出层节数 m 是由学习实例的输入模式和输出模式确定的。但是,如何选择隐层数和各隐层节点数,以有利于网络学习过程的收敛、网络计算的逼近程度和效率等网络性能的提高,仍然缺乏理论的支持。一般认为,较多的隐层节点有利于提高网络的收敛性和非线性映射能力。但是,较多的隐层节点显然会增加网络学习与使用的时间与存储空间的开销。因此,希望能改进 BP 学习算法,不仅能获得一个稳定的权值分布,还希望能获得满足误差要求且隐层节点尽可能少的网络结构。为此,我们提出下述动态结构 BP 网络学习算法。

动态结构 BP 网络学习算法是在网络学习过程中,逐步递减隐层节点数,使网络收敛于 $E \leqslant \varepsilon$,且有最少的隐层节点。为此,对给出的 BP 学习算法中的步骤(7)进行下述改进。

计算 N 个学习实例的平均输出误差为

$$E(t) = \left(\sum_{k=1}^{N}\sum_{j=1}^{m} E_{jk}(t)\right) \Big/ mN$$

若 $E \leqslant \varepsilon$,说明学习过程设定的网络结构已收敛,则从所有的隐层中删去一个节点,但每个隐层的节点数最少为 $2n+1$,其中,n 为输入层节点数。即

若 $n^{(l)} > 2n+1$,则 $n^{(l)} = n^{(l)} - 1, l = 2, \cdots, L-1$

转步骤(2);否则,学习过程结束。

若 $E(t)>\varepsilon$，且 $t\geqslant t_{\max}$，则每个隐层增加一个隐层节点，即

$$n^{(l)}=n^{(l)}+1,\quad l=2,\cdots,L-1$$

转步骤(2)；若 $E(t)>\varepsilon$，且 $t<t_{\max}$，则转步骤(8)进行误差反向传播修改各连接权矩阵。

7.4.5　人工神经网络的应用

人工神经网络是一种非逻辑、非语言、非静态和非线性的信息处理方法。虽然单个神经元的结构和功能极其简单和有限，但是，由大量神经元构成的神经网络系统所能实现的行为却是极其丰富多彩的。需要指出的是，在人工智能的研究和应用领域中，基于非线性数值计算的人工神经网络的知识表示与推理和基于符号逻辑的知识表示与推理的理论与方法虽然有很大的不同，但是，它们可以在人工智能的应用研究中结合起来，发挥各自的作用，并逐渐走向融合。

1. 神经网络专家系统

神经网络的发展为专家系统的研究开辟了新途径，人们可以利用神经网络的学习功能和并行推理能力解决专家系统中的知识自动获取与知识表示及推理等问题。

(1) 神经网络的知识表示

知识获取与知识表示是人工智能的基本技术。知识表示是知识的模型化和形式化，已有的知识表示形式有产生式、语义网络、谓词逻辑、框架等，虽然各自采用不同的结构和组织形式描述和表示知识，但都是把知识变换成计算机可以存储的形式存入知识库。当需要推理时，根据设计的搜索匹配算法在知识库中搜索可匹配的知识规则。当知识规则很多时，这种知识表示存在以下缺点：

① 采用何种策略组织知识库，使得对知识库的搜索有较高的效率，是一个很困难的问题。

② 对知识库搜索匹配规则是对知识库的各条规则进行串行搜索匹配，对于有多个可匹配规则的情况，还必须解决可用规则的冲突问题。若知识库中的规则存在不一致性，或者采用了不适当的规则搜索策略，那么，搜索过程将可能导致无穷递归问题。

神经网络的知识表示采用与传统人工智能完全不同的思想。传统的知识表示都可以看成是知识的一种显式表示，而神经网络的知识表示可看成是知识的一种隐式表示。神经网络的知识表示也不像基于规则的专家系统那样把局部的知识表示成一条规则，若干知识表示成相互独立的规则组成一个规则集；神经网络的知识表示是把某一问题领域的若干知识彼此关联地表示在一个神经网络中。

一个神经网络可以用一个加权有向图表示。加权有向图中的节点连接关系和权值分布可以用一个矩阵来表示，这个矩阵称为邻接权矩阵。一个有 m 个节点的神经网络的邻接权矩阵 $W=[w_{ij}]_{m\times m}$ 的定义为

$$w_{ij}=\begin{cases}w_{ij}, & \text{若节点 } i \text{ 有至节点 } j \text{ 的邻接且权值为 } w_{ij}\\ 0, & \text{若节点 } i \text{ 无至节点 } j \text{ 的邻接}\end{cases}$$

邻接权矩阵可看成是神经网络专家系统的知识表示形式。对于单向传播的神经网络(例如 BP 神经网络)，它具有的稳定的层间连接权 $W^{(1)},\cdots,W^{(L)}$ 就是它的知识表示。

基于神经网络的知识表示方法有以下优点：

① 能够表示事物的复杂关系，如模糊因果关系。

② 具有统一的知识表示形式，便于知识的组织，知识表示形式的通用性强。

③ 便于实现知识的自动获取。

④ 便于知识推理。

(2) 神经网络的知识自动获取

知识表示与知识获取是知识推理的基础，只有获取的知识存于知识库中，才能通过推理来求解问题。

神经网络是通过实例学习来实现知识自动获取的。在进行知识获取时，要求领域专家提供学习实例及其相应的期望解，经过网络自适应学习算法不断修改网络的权值分布，一旦网络稳定后，就把领域专家求解该问题的知识和经验(通过提供的学习实例来表示)分布到网络的互连结构及权值分布上，从而得到推理所需要的知识库。所以说，一个经过学习训练而达到权值分布稳定的神经网络就是神经网络专家系统的知识库。

神经网络专家系统的基本结构如图 7.16 所示。知识获取模块的功能主要是输入和存储专家提供的学习实例、确定网络结构及调用网络学习算法，以获得知识库而完成知识获取。

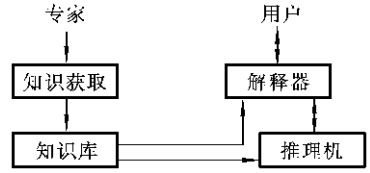

图 7.16 神经网络专家系统的基本结构

知识库可以通过知识获取模块不断更新知识。当表示新知识的新学习实例输入后，知识获取模块可通过对新学习实例的学习，自动获得表示包括新知识在内的更多知识与经验的新的权值分布，从而更新原来的知识库。

(3) 神经网络的知识推理与解释

神经网络专家系统的知识推理与基于符号逻辑的专家系统的推理机制是不同的，神经网络的推理是一种正向的非线性数值计算过程。

对于一个特定输入模式，神经网络通过正向传播计算，产生一个输出模式，由各输出节点代表的多个逻辑概念被同时并行地计算出来，因此，神经网络的推理是一种并行推理机制。

由于神经网络各输出节点的输出是数值的，因此，需要一个解释器对输出模式进行解释。解释器的主要任务是将输出的数值向量转换成高层逻辑概念。对于不同的知识领域，解释器中对输出模式的解释规则也不相同。同样，解释器中也需要有一套解释规则将用户输入的逻辑概念转换成网络输入模式要求的数值向量。

神经网络的知识推理有以下特点：

① 同一层神经元的计算是完全可并行的，层间的传播是逐层串行的，但是，同一层神经元的个数比神经网络的层数大得多，因此，神经网络的知识推理机制很适合实现知识的并行推理。

② 在基于规则的推理方法中，若多条规则的前件均与给定的事实相匹配，则出现可用规则的冲突问题。为了消解冲突，需要增加冲突消解的开销，从而降低了推理速度。神经网络的推理过程不需要搜索匹配规则，也不存在可用规则冲突问题。基于神经网络的专家系统的推理时间开销同基于规则的专家系统的推理时间的开销比较，前者要少得多。

③ 基于神经网络的专家系统的解释器可以有一套输入模式解释规则和一套输出模式解释规则，分别将用户输入的逻辑概念转换成网络的数值输入模式和把网络的数值输出模式转换成输出的逻辑概念提供给用户。由于基于神经网络的专家系统的知识表示是隐式的、非局部的，因此，神经网络专家系统的解释器难以像基于规则的专家系统那样，通过记录推理过程使用的规则链，来向用户提供 why 询问和 how 询问的解释。

2. 基于神经网络的模糊分类器

模糊分类的基本思想是：根据对象的若干特征值计算出这个对象属于哪一类的隶属度值，从而确定这个对象属于隶属度值最高的那一类。

用下面的一个例子来说明神经网络在模糊分类中的应用。

例 7.8　根据分类对象的两个特征 $X=(x_1,x_2)$，把对象分为 R_1 和 R_2 两类，请建立基于神经网络的模糊分类器。提供的训练实例如表 7.8 所示，共有 20 个训练实例，x_{1k} 和 x_{2k} 分别是实例 k 的两个特征值，y_{1k}^* 和 y_{2k}^* 分别是实例 k 属于类 R_1 和 R_2 的隶属度值，若 $y_{1k}^*=1$，$y_{2k}^*=0$，则实例 k 肯定属于 R_1 类；若 $y_{1k}^*=0$，$y_{2k}^*=1$，则实例 k 肯定属于 R_2 类。

表 7.8　例 7.8 模糊分类器的训练实例

k	1	2	3	4	5	6	7	8	9	10
x_{1k}	0.05	0.09	0.12	0.15	0.20	0.75	0.80	0.82	0.90	0.95
x_{2k}	0.02	0.11	0.20	0.22	0.25	0.75	0.83	0.80	0.89	0.89
y_{1k}^*	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
y_{2k}^*	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0
k	11	12	13	14	15	16	17	18	19	20
x_{1k}	0.09	0.10	0.14	0.18	0.22	0.77	0.79	0.84	0.94	0.98
x_{2k}	0.04	0.10	0.21	0.24	0.28	0.78	0.81	0.82	0.93	0.99
y_{1k}^*	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
y_{2k}^*	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0

解　选择一个 4 层的 BP 神经网络来建立模糊分类器，BP 网络的结构为 $2\times3\times3\times2$，即输入层和输出层都是 2 个节点，2 个隐层都是 3 个节点，如图 7.17 所示。输入层的 2 个节点分别输入对象的 2 个特征值 x_1 和 x_2，输入层节点的传递函数采用简单的线性函数 $f(x_j)=x_j$，$j=1,2$。输出层的 2 个节点分别输出对象分类的隶属度值 y_1 和 y_2。隐层和输出层节点的传递函数采用 Sigmoid 函数 $f(x)=\dfrac{1}{1+\mathrm{e}^{-x}}$，由(7-1)式和(7-2)式可知，第 2、3、4 层的节点 j 的输入 $I_j^{(l)}$ 和输出 $O_j^{(l)}$ 分别为

$$I_j^{(l)}=\sum_{i=1}^{n(l-1)} w_{ij}^{(l-1)}O_i^{(l-1)}$$

$$O_j^{(l)}=f(I_j^{(l)})=\frac{1}{1+\exp(-\sum_i w_{ij}^{(l-1)}O_i^{(l-1)})}$$

当 $l=2$ 时，有 $n^{(l-1)}=n^{(1)}=2$，　$j=1,2,3$。

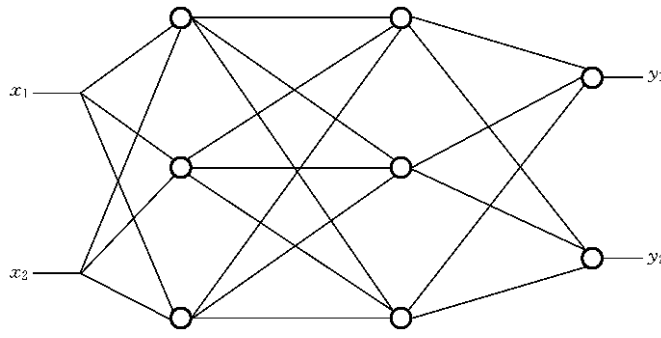

图 7.17　例 7.8 的神经网络结构

当 $l=3$ **时，有** $n^{(l-1)}=n^{(2)}=3$，　$j=1,2,3$。

当 $l=4$ **时，有** $n^{(l-1)}=n^{(3)}=3$，　$j=1,2$，$y_1=O_1^{(4)}$，　$y_2=O_2^{(4)}$。

图 7.17 所示的 BP **网络有 3 个连接权矩阵，分别是** $W^{(1)}=[w_{ij}^{(1)}]_{2\times3}$，$W^{(2)}=[w_{ij}^{(2)}]_{3\times3}$，$W^{(3)}=[w_{ij}^{(3)}]_{3\times2}$，**对连接权矩阵初始化，设初始化随机值分别为**

$$W^{(1)}=\begin{bmatrix}w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)}\\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)}\end{bmatrix}=\begin{bmatrix}0.5 & 0.4 & 0.1\\ 0.2 & 0.6 & 0.2\end{bmatrix}$$

$$W^{(2)}=\begin{bmatrix}w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)}\\ w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)}\\ w_{31}^{(2)} & w_{32}^{(2)} & w_{33}^{(2)}\end{bmatrix}=\begin{bmatrix}0.10 & 0.55 & 0.35\\ 0.20 & 0.45 & 0.35\\ 0.25 & 0.15 & 0.60\end{bmatrix}$$

$$W^{(3)}=\begin{bmatrix}w_{11}^{(3)} & w_{12}^{(3)}\\ w_{21}^{(3)} & w_{22}^{(3)}\\ w_{31}^{(3)} & w_{32}^{(3)}\end{bmatrix}=\begin{bmatrix}0.30 & 0.35\\ 0.35 & 0.25\\ 0.45 & 0.30\end{bmatrix}$$

取第一个对象 $k=1$ **的特征向量** $X_1=(x_1,x_2)=(0.05,0.02)$**进行第 1 次迭代计算。**

(1) **输入正向传播计算**

第 1 层(输入层)的输出为

$$O_1^{(1)}=f(x_1)=x_1=0.05$$
$$O_2^{(1)}=f(x_2)=x_2=0.02$$

第 2 层的输出为

$$\begin{aligned}O_1^{(2)}&=1/[1+\exp(-\sum_{i=1}^{2}w_{i1}^{(1)}O_i^{(1)})]\\&=1/[1+\exp(-(w_{11}^{(1)}O_1^{(1)}+w_{21}^{(1)}O_2^{(1)}))]\\&=1/[1+\exp(-(0.5\times0.05+0.2\times0.02))]\\&=0.507249\end{aligned}$$

$$\begin{aligned}O_2^{(2)}&=1/[1+\exp(-\sum_{i=1}^{2}w_{i2}^{(1)}O_i^{(1)})]\\&=1/[1+\exp(-(w_{12}^{(1)}O_1^{(1)}+w_{22}^{(1)}O_2^{(1)}))]\\&=1/[1+\exp(-(0.4\times0.05+0.6\times0.02))]\\&=0.507999\end{aligned}$$

$$\begin{aligned}O_3^{(2)} &= 1/[1+\exp(-\sum_{i=1}^{2} w_{i3}^{(1)} O_i^{(1)})] \\ &= 1/[1+\exp(-(w_{13}^{(1)} O_1^{(1)} + w_{23}^{(1)} O_2^{(1)}))] \\ &= 1/[1+\exp(-(0.1\times 0.05 + 0.2\times 0.02))] \\ &= 0.502250\end{aligned}$$

第 3 层的输出为

$$\begin{aligned}O_1^{(3)} &= 1/[1+\exp(-\sum_{i=1}^{3} w_{i1}^{(2)} O_i^{(2)})] \\ &= 1/[1+\exp(-(w_{11}^{(2)} O_1^{(2)} + w_{21}^{(2)} O_2^{(2)} + w_{31}^{(2)} O_3^{(2)}))] \\ &= 1/[1+\exp(-(0.10\times 0.507249 + 0.20\times 0.507999 + 0.25\times 0.502250) \\ &= 0.569028\end{aligned}$$

$$\begin{aligned}O_2^{(3)} &= 1/[1+\exp(-\sum_{i=1}^{3} w_{i2}^{(2)} O_i^{(2)})] \\ &= 1/[1+\exp(-(w_{12}^{(2)} O_1^{(2)} + w_{22}^{(2)} O_2^{(2)} + w_{32}^{(2)} O_3^{(2)}))] \\ &= 1/[1+\exp(-(0.55\times 0.507249 + 0.45\times 0.507999 + 0.15\times 0.502250))] \\ &= 0.641740\end{aligned}$$

$$\begin{aligned}O_3^{(3)} &= 1/[1+\exp(-\sum_{i=1}^{3} w_{i3}^{(2)} O_i^{(2)})] \\ &= 1/[1+\exp(-(w_{13}^{(2)} O_1^{(2)} + w_{23}^{(2)} O_2^{(2)} + w_{33}^{(2)} O_3^{(2)}))] \\ &= 1/[1+\exp(-(0.35\times 0.507249 + 0.35\times 0.507999 + 0.60\times 0.502250))] \\ &= 0.658516\end{aligned}$$

第 4 层(输出层)的输出为

$$\begin{aligned}O_1^{(4)} &= 1/[1+\exp(-\sum_{i=1}^{3} w_{i1}^{(3)} O_i^{(3)})] \\ &= 1/[1+\exp(-(w_{11}^{(3)} O_1^{(3)} + w_{21}^{(3)} O_2^{(3)} + w_{31}^{(3)} O_3^{(3)}))] \\ &= 1/[1+\exp(-(0.30\times 0.569028 + 0.35\times 0.641740 + 0.45\times 0.658516))] \\ &= 0.666334\end{aligned}$$

$$\begin{aligned}O_1^{(4)} &= 1/[1+\exp(-\sum_{i=1}^{3} w_{i2}^{(3)} O_i^{(3)})] \\ &= 1/[1+\exp(-(w_{12}^{(3)} O_1^{(3)} + w_{22}^{(3)} O_2^{(3)} + w_{32}^{(3)} O_3^{(3)}))] \\ &= 1/[1+\exp(-(0.35\times 0.569028 + 0.25\times 0.641740 + 0.30\times 0.658516))] \\ &= 0.635793\end{aligned}$$

由此得出第一个对象实例的网络计算隶属度值为 $y_1 = O_1^{(4)} = 0.666334$，$y_2 = O_2^{(4)} = 0.635793$。**与该实例给出的隶属度值** $y_1^* = 1.0$，$y_2^* = 0.0$ **进行比较，计算误差。误差函数定义为**

$$E_j = y_j - y_j^* \qquad j = 1,2$$

第一个实例的输出误差为

$$E_1^{(4)} = y_1 - y_1^* = 0.666334 - 1.0 = -0.333666$$

$$E_2^{(4)} = y_2 - y_2^* = 0.635793 - 0.0 = 0.635793$$

设定模糊分类器的允许误差为 $\varepsilon=0.01$，因为 $|E_j|>\varepsilon$，$j=1,2$，因此，由误差反向传播来修改连接权值。

(2) 误差反向传播计算

为简化误差反向传播的计算，可以简化传递函数 f 的导数 f' 的计算。对本例的误差反向传播修改权值的计算使用下述计算公式：

$$E_i^{(l)}=O_i^{(l)}(1-O_i^{(l)})\sum_{j=1}^{n(l+1)}w_{ij}^{(l)}E_j^{(l+1)}$$

$$\Delta w_{ij}^{(l)}=\eta E_j^{(l+1)}O_i^{(l)}$$

$$w_{ij}^{(l)}=w_{ij}^{(l)}+\Delta w_{ij}^{(l)}=w_{ij}^{(l)}+\eta E_j^{(l+1)}O_i^{(l)}$$

对 $l=3$，$n^{(l)}=n^{(3)}=3$，故 $i=1,2,3$；$n^{(l+1)}=n^{(4)}=2$，故 $j=1,2$。

对 $l=2$，$n^{(l)}=n^{(2)}=3$，故 $i=1,2,3$；$n^{(l+1)}=n^{(3)}=3$，故 $j=1,2,3$。

对 $l=1$，$n^{(l)}=n^{(1)}=2$，故 $i=1,2$；$n^{(l+1)}=n^{(2)}=3$，故 $j=1,2,3$。

取学习率 $\eta=0.3$。

① 由第 4 层的误差 $E_1^{(4)}$ 和 $E_2^{(4)}$，更新第 3 层连接第 4 层的权值分布 $W^{(3)}=[w_{ij}^{(3)}]_{3\times2}$ 为

$$w_{11}^{(3)}=w_{11}^{(3)}+\eta E_1^{(4)}O_1^{(3)}=0.30+0.3\times(-0.333666)\times0.569028=0.243040$$

$$w_{21}^{(3)}=w_{21}^{(3)}+\eta E_1^{(4)}O_2^{(3)}=0.35+0.3\times(-0.333666)\times0.641740=0.285762$$

$$w_{31}^{(3)}=w_{31}^{(3)}+\eta E_1^{(4)}O_3^{(3)}=0.45+0.3\times(-0.333666)\times0.658516=0.384083$$

$$w_{12}^{(3)}=w_{12}^{(3)}+\eta E_2^{(4)}O_1^{(3)}=0.35+0.3\times0.635793\times0.569028=0.458535$$

$$w_{22}^{(3)}=w_{22}^{(3)}+\eta E_2^{(4)}O_2^{(3)}=0.25+0.3\times0.635793\times0.641740=0.372404$$

$$w_{32}^{(3)}=w_{32}^{(3)}+\eta E_2^{(4)}O_3^{(3)}=0.30+0.3\times0.635793\times0.658516=0.425604$$

② 将误差 $E_1^{(4)}$ 和 $E_2^{(4)}$ 按修改前的权值分布 $W^{(3)}$ 反向传播至第 3 层，计算第 3 层的 $n^{(3)}=3$ 个节点的输出误差：

$$\begin{aligned}E_1^{(3)}&=O_1^{(3)}(1-O_1^{(3)})\sum_{j=1}^{2}w_{1j}^{(3)}E_j^{(4)}\\&=O_1^{(3)}(1-O_1^{(3)})(w_{11}^{(3)}E_1^{(4)}+w_{12}^{(3)}E_2^{(4)})\\&=0.569028\times(1-0.569028)\times[0.30\times(-0.333666)+0.35\times0.635793]\\&=0.030024\end{aligned}$$

$$\begin{aligned}E_2^{(3)}&=O_2^{(3)}(1-O_2^{(3)})\sum_{j=1}^{2}w_{2j}^{(3)}E_j^{(4)}\\&=O_2^{(3)}(1-O_2^{(3)})(w_{21}^{(3)}E_1^{(4)}+w_{22}^{(3)}E_2^{(4)})\\&=0.641740\times(1-0.641740)\times[0.35\times(-0.333666)+0.25\times0.635793]\\&=0.009694\end{aligned}$$

$$\begin{aligned}E_3^{(3)}&=O_3^{(3)}(1-O_3^{(3)})\sum_{j=1}^{2}w_{3j}^{(3)}E_j^{(4)}\\&=O_3^{(3)}(1-O_3^{(3)})(w_{31}^{(3)}E_1^{(4)}+w_{32}^{(3)}E_2^{(4)})\\&=0.658516\times(1-0.658516)\times[0.45\times(-0.333666)+0.30\times0.635793]\\&=0.009127\end{aligned}$$

由第 3 层的误差 $E_1^{(3)}$、$E_2^{(3)}$ 和 $E_3^{(3)}$，更新第 2 层连接第 3 层的权值分布 $W^{(2)}=[W_{ij}^{(2)}]_{3\times3}$ 为

$$w_{11}^{(2)}=w_{11}^{(2)}+\eta E_1^{(3)}O_1^{(2)}=0.10+0.3\times0.030024\times0.507249=0.104968$$
$$w_{21}^{(2)}=w_{21}^{(2)}+\eta E_1^{(3)}O_2^{(2)}=0.20+0.3\times0.030024\times0.507999=0.204576$$
$$w_{31}^{(2)}=w_{31}^{(2)}+\eta E_1^{(3)}O_3^{(2)}=0.25+0.3\times0.030024\times0.502250=0.254524$$
$$w_{12}^{(2)}=w_{12}^{(2)}+\eta E_2^{(3)}O_1^{(2)}=0.55+0.3\times0.009694\times0.507249=0.551475$$
$$w_{22}^{(2)}=w_{22}^{(2)}+\eta E_2^{(3)}O_2^{(2)}=0.45+0.3\times0.009694\times0.507999=0.451477$$
$$w_{32}^{(2)}=w_{32}^{(2)}+\eta E_2^{(3)}O_3^{(2)}=0.15+0.3\times0.009694\times0.502250=0.151461$$
$$w_{13}^{(2)}=w_{13}^{(2)}+\eta E_3^{(3)}O_1^{(2)}=0.35+0.3\times0.009127\times0.507249=0.351389$$
$$w_{23}^{(2)}=w_{23}^{(2)}+\eta E_3^{(3)}O_2^{(2)}=0.35+0.3\times0.009127\times0.507999=0.351391$$
$$w_{33}^{(2)}=w_{33}^{(2)}+\eta E_3^{(3)}O_3^{(2)}=0.60+0.3\times0.009127\times0.502250=0.601375$$

③ 将误差 $E_1^{(3)}$、$E_2^{(3)}$ 和 $E_3^{(3)}$ 按修改前的权值分布 $W^{(2)}$ 反向传播至第 2 层，计算第 2 层的 $n^{(2)}=3$ 个节点的输出误差：

$$\begin{aligned}E_1^{(2)}&=O_1^{(2)}(1-O_1^{(2)})\sum_{j=1}^{3}w_{1j}^{(2)}E_j^{(3)}\\&=O_1^{(2)}(1-O_1^{(2)})(w_{11}^{(2)}E_1^{(3)}+w_{12}^{(2)}E_2^{(3)}+w_{13}^{(2)}E_3^{(3)})\\&=0.507249\times(1-0.507249)\times(0.10\times0.030024+0.55\times0.009694\\&\quad+0.35\times0.009127)=0.002882\end{aligned}$$

$$\begin{aligned}E_2^{(2)}&=O_2^{(2)}(1-O_2^{(2)})\sum_{j=1}^{3}w_{2j}^{(2)}E_j^{(3)}\\&=O_2^{(2)}(1-O_2^{(2)})(w_{21}^{(2)}E_1^{(3)}+w_{22}^{(2)}E_2^{(3)}+w_{23}^{(2)}E_3^{(3)})\\&=0.507999\times(1-0.507999)\times(0.20\times0.030024+0.45\times0.009694\\&\quad+0.35\times0.009127)=0.003390\end{aligned}$$

$$\begin{aligned}E_3^{(2)}&=O_3^{(2)}(1-O_3^{(2)})\sum_{j=1}^{3}w_{3j}^{(2)}E_j^{(3)}\\&=O_3^{(2)}(1-O_3^{(2)})(w_{31}^{(2)}E_1^{(3)}+w_{32}^{(2)}E_2^{(3)}+w_{33}^{(2)}E_3^{(3)})\\&=0.502250\times(1-0.502250)\times(0.25\times0.030024+0.15\times0.009694\\&\quad+0.60\times0.009127)=0.003609\end{aligned}$$

由第 2 层的误差 $E_1^{(2)}$、$E_2^{(2)}$ 和 $E_3^{(2)}$，更新第 1 层连接第 2 层的权值分布 $W^{(1)}=[w_{ij}^{(1)}]_{2\times3}$ 为

$$w_{11}^{(1)}=w_{11}^{(1)}+\eta E_1^{(2)}O_1^{(1)}=w_{11}^{(1)}+\eta E_1^{(2)}x_1=0.50+0.3\times0.002882\times0.05=0.500043$$
$$w_{21}^{(1)}=w_{21}^{(1)}+\eta E_1^{(2)}O_2^{(1)}=w_{21}^{(1)}+\eta E_1^{(2)}x_2=0.20+0.3\times0.002882\times0.02=0.200017$$
$$w_{12}^{(1)}=w_{12}^{(1)}+\eta E_2^{(2)}O_1^{(1)}=w_{12}^{(1)}+\eta E_2^{(2)}x_1=0.40+0.3\times0.003390\times0.05=0.400051$$
$$w_{22}^{(1)}=w_{22}^{(1)}+\eta E_2^{(2)}O_2^{(1)}=w_{22}^{(1)}+\eta E_2^{(2)}x_2=0.60+0.3\times0.003390\times0.02=0.600020$$
$$w_{13}^{(1)}=w_{13}^{(1)}+\eta E_3^{(2)}O_1^{(1)}=w_{13}^{(1)}+\eta E_3^{(2)}x_1=0.10+0.3\times0.003609\times0.05=0.100054$$
$$w_{23}^{(1)}=w_{23}^{(1)}+\eta E_3^{(2)}O_2^{(1)}=w_{23}^{(1)}+\eta E_3^{(2)}x_2=0.20+0.3\times0.003609\times0.02=0.200022$$

至此，已完成第一次学习对网络权值分布的修改。取第 2 个训练实例对网络进行第 2 次学习训练。在将 20 个训练实例都学习一遍之后，若网络输出误差仍有 $|E_j|>\varepsilon$，$j=1,2$，则再取第 1 个实例对网络进行第 21 次学习训练。反复迭代，直至输出误差达到期望要求，这时，满足期望要求的模糊分类器就建成了。

任一分类对象由模糊分类器进行分类的过程就是该分类对象的特征向量 $X=(x_1,x_2)$ 在分类器神经网络中的正向传播计算过程，按正向传播计算公式逐层计算，就可在输出层得到分类结果的输出向量 $Y=(y_1,y_2)$。

由模糊分类器的解释器对输出向量进行模糊解释。对象 x 的隶属度 $\mu(x)$ 的值为 $0\leqslant\mu(x)\leqslant1$，$\mu(x)$ 与模糊语言修饰词的简单对应关系如表 7.9 所示。若某一个对象 x 的分类结果向量是 $y_1=0.872312$，$y_2=0.122810$，那么，由解释器得出的解释是：对象 x“几乎是”属于 R_1 类，“稍稍像是”属于 R_2 类。

表 7.9 模糊修饰词的隶属度

修饰词	隶属度	修饰词	隶属度
肯定是	1	比较像是	0.4
几乎是	0.9	有些像是	0.3
极是	0.8	有点像是	0.2
很是	0.7	稍稍像是	0.1
相当是	0.6	肯定不是	0
差不多是	0.5		

3. 神经网络在预测中的应用

通过一个应用实例来说明神经网络在预测中的应用，并讨论不同结构的 BP 网络及随机初始化对预测结果的影响。

例 7.9 某远洋船舶航运集团从 1995 年至 2005 年的各年货运量如表 7.10 所示，建立该航运集团基于神经网络的年货运量时序预测模型。

表 7.10 1995 年至 2005 年货运量

单位：百万吨

年度	运量	年度	运量	年度	运量
1995	1399	1999	1588	2003	1685
1996	1467	2000	1622	2004	1789
1997	1567	2001	1611	2005	1790
1998	1595	2002	1615		

解 首先需要确定预测模型的神经网络的结构。可以采用 n 输入单输出的三层 BP 神经网络结构。模型建立后，一次使用可在单输出节点得到下一年度的运量。输入层节点数 n 与学习实例的输入模式的选择有关。如果选择连续 5 年的各年运量作为输入模式，则输入节点数 $n=5$，第 6 年的运量为实例的输出，那么，由表 7.10 给出的 1995 年至 2005 年的各年运量，可以获得 6 个学习实例，第 1 个实例的输入向量为 $X_1=(1399,1467,1567,1595,1588)$，输出为 $y_1^*=1622$；第 2 个实例的输入向量为 $X_2=(1467,1567,1595,1588,1622)$，输出为 $y_2^*=1611$；……，第 6 个实例的输入向量为 $X_6=(1622,1611,1615,1685,1789)$，输出为 $y_6^*=1790$。当然，也可以选择输入模式为连续 6 年的运量，则 $n=6$，第 7 年的运量为实例的输出，那么，由

1986 年至 1996 年的各年运量，只能获得 5 个学习实例。若选择 $n=7$，则只能获得 4 个学习实例。由于本例可提供的学习实例数量较少，且只有一个隐层，故设置较多的隐层节点，设置隐层节点数为 $3n$，所以，预测模型的 BP 网络结构为 $n\times 3n\times 1$。

输入节点的传递函数采用线性函数 $f(x)=x$。隐层节点和输出节点的传递函数采用 Sigmoid型的双曲正切函数 $f(x)=\mathrm{th}(x)=\dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$。双曲正切函数 $\mathrm{th}(x)$的函数图形是关于原点对称的，当输入 $x=0$ 时，输出 $f(x)=0$，与预测要求一致。误差反向传播过程需要使用传递函数的导数为 $f'(x)=\mathrm{th}'(x)=\left(\dfrac{2}{e^{x}+e^{-x}}\right)^{2}$。

证明映射网络存在性的定理 7.1 和定理 7.2 说明对连续函数 $f:[0,1]^{n}\to R^{m}$，可用一个三层的 BP 网络精确地逼近 f。由于网络的输入量取值应有 $x_i\in[0,1]$，$i=1,2,\cdots,n$，所以，对本例学习实例的数据要进行归一化。只需要对所有的输入数据除以 10^{4}，然后对网络计算的输出结果乘以 10^{4}即可。

由表 7.10 得出网络结构为 $5\times 15\times 1$ 的 6 个归一化后的学习实例如表 7.11 所示。

表 7.11　5×15×1 的归一化学习实例

k	x_{1k}	x_{2k}	x_{3k}	x_{4k}	x_{5k}	y_k^{*}
1	0.1399	0.1467	0.1567	0.1595	0.1588	0.1622
2	0.1467	0.1567	0.1595	0.1588	0.1622	0.1611
3	0.1567	0.1595	0.1588	0.1622	0.1611	0.1615
4	0.1595	0.1588	0.1622	0.1611	0.1615	0.1685
5	0.1588	0.1622	0.1611	0.1615	0.1685	0.1789
6	0.1622	0.1611	0.1615	0.1685	0.1789	0.1790

网络的输入节点数为 n，隐层节点数为 $m=3n$，故输入层至隐层的连接权矩阵为 $W=[w_{ij}]_{n\times m}$。输出层只有一个输出节点，故隐层至输出层的连接权表示为权向量 $U=(u_j)_m$。根据 BP 学习算法，对实例 k 的学习计算过程如下。

① 计算隐层各节点输入量

$$a_{jk}=\sum_{i=1}^{n}w_{ij}x_{ik},\quad j=1,2,\cdots,m$$

② 计算隐层各节点输出量

$$b_{jk}=\mathrm{th}(a_{jk}),\quad j=1,2,\cdots,m$$

③ 计算输出节点输入量

$$c_k=\sum_{j=1}^{m}u_jb_{jk}$$

④ 计算输出节点输出量

$$y_k=\mathrm{th}(c_k)$$

⑤ 计算隐层 m 个节点至输出节点的连接权的修正量

$$\delta_k=-(y_k^{*}-y_k)f'(c_k)=-(y_k^{*}-y_k)\mathrm{th}'(c_k)$$

$$\Delta u_j=\eta\delta_kB_{jk},\quad j=1,2,\cdots,m$$

⑥ 计算输入层 n 个节点至隐层 m 个节点的连接权的修正量

$$\delta_{jk}=f'(a_{jk})\sum_{j=1}^{m}u_j\delta_k=\mathrm{th}'(a_{jk})\sum_{j=1}^{m}u_j\delta_k,\quad j=1,2,\cdots,m$$

$$\Delta w_{ij}=\eta\delta_{jk}x_{ik},\quad i=1,2,\cdots,n,\quad j=1,2,\cdots,m$$

⑦ 修改权值分布

$$w_{ij}=w_{ij}+\Delta w_{ij},\quad i=1,2,\cdots,n,\quad j=1,2,\cdots,m$$

$$u_j=u_j+\Delta u_j,\quad j=1,2,\cdots,m$$

其中，步骤①～④是输入正向传播过程，步骤⑤～⑦是误差反向传播过程。

对 6 个实例（如表 7.11 所示）学习 $t_{\max}$ 次后，得到的 BP 网络就是一个年运量预测模型。用 2001～2005 年的运量作为输入模式输入预测模型，经过由步骤①～④组成的正向传播计算，模型的输出值乘以 10^4 就是 2006 年年运量的预测值。类似地，用 2002～2006 年的运量作为输入模式就可得到 2007 年年运量的预测值。也就是说，每次预测都是把上一年度的预测值作为本年度预测的输入模式的 x_n 的值，按时序进行预测。

以本例为例，对 BP 网络进行下述有关讨论。

(1) 预测模型的知识更新

学习实例越多，或不断增加新的学习实例以更新网络记忆的知识，那么，时序预测模型获得的知识就越可靠，预测精度越高。

本例只提供了 6 个实例，可采取知识更新的方法来提高模型预测精度。具体方法是：每次得到一个预测值后，就把它作为一个新的学习实例的期望输出 y^*，并把这个新实例增加到原来的实例集中，重新训练网络，用知识更新后的网络再预测下一年的预测值。当然，每作一次预测就重新用新实例集训练网络，然后才作下一次预测，这样所需的时间的开销会成倍增加，但是，会使预测精度提高很多。

(2) 网络权值初始化对输出的影响

在网络的学习训练过程中，连接权值的初始值是随机设置的。随机设置的初始化权值对网络学习过程是有影响的。随机初始化的权值不同时，对指定的允许误差 ε，有的学习过程能在设定的最大学习次数 $t_{\max}$ 范围内收敛，但有的学习过程在学习了 $t_{\max}$ 次之后不能达到期望的允许误差。

由表 7.11 给出的实例集，对 5×15×1 的 BP 网络重新独立地训练 6 次，每次训练都对权值重新随机初始化，得出 6 个学习实例的输出值如表 7.12 中的一列所示（已乘以 10^4）。每一列有 6 个输出值 y_1～y_6，分别是实例集的 6 个实例的输出值，$\bar{y}_k$ 是实例 k 的 6 个输出值（同一行的值）$y_k(1)$～$y_k(6)$的算术平均值。

表 7.12 5×15×1 的 BP 网络输出结果

k	$y_k(1)$	$y_k(2)$	$y_k(3)$	$y_k(4)$	$y_k(5)$	$y_k(6)$	$\bar{y}_k$
1	1556.36	1552.96	1664.99	1550.46	1538.47	1581.94	1547.20
2	1610.40	1606.85	1610.55	1603.05	1599.24	1631.34	1610.24
3	1629.54	1622.75	1623.61	1640.05	1636.15	1619.04	1628.52
4	1686.96	1691.98	1700.33	1673.83	1665.13	1704.25	1687.08
5	1707.27	1706.67	1715.38	1698.63	1693.20	1728.24	1708.23
6	1770.16	1758.84	1772.07	1760.41	1752.60	1790.05	1767.36

由表 7.12 可见，当设定有限次学习次数 $t_{\max}$ 作为算法终止条件时，权值的初始化对输出值有一定的影响，表中同一行的输出值 $y_k(1)\sim y_k(6)$ 之间的差异就反映了权值随机初始化的影响。

(3) 网络结构对输出的影响

网络的结构不同，对网络的输出也有影响。BP 网络结构分别采用 5×15×1，6×18×1 和 7×21×1 三种结构，对由 1995 年至 2005 年年运量组成的学习实例训练后的输出值如表 7.13 所示。对 5×15×1 网络，可提供 6 个实例；对 6×18×1 网络，由于一个实例的输入模式需要连续的 6 年运量，所以只能提供 5 个实例；对 7×21×1 网络，只能提供 4 个实例。

表 7.13　三种不同结构的 BP 网络的输出值与误差

年　度	实际运量 y_k^*	5×15×1 输出值		6×18×1 输出值		7×21×1 输出值	
		y_k	$y_k-y_k^*$	y_k	$y_k-y_k^*$	y_k	$y_k-y_k^*$
2000	1622	1556.36	−65.64				
2001	1611	1610.40	−0.60	1612.92	1.92		
2002	1615	1629.54	14.54	1613.99	−1.01	1612.25	−2.75
2003	1685	1686.96	1.96	1687.10	2.10	1693.69	8.69
2004	1789	1707.27	−81.73	1789.36	0.36	1788.35	−0.65
2005	1790	1770.16	−19.84	1790.38	0.38	1794.05	4.05

由表 7.13 可见，6×18×1 网络输出的平均平方误差 $E=\frac{1}{N}\sum_{k=1}^{N}(y_k-y_k^*)^2$ 最小，采用 6×18×1 网络结构预测的拟合精度最好，7×21×1 网络结构的预测精度次之，5×15×1 网络结构的预测精度最差。这里，我们忽略了网络权值随机初始化和学习实例数不同对网络输出值的影响。

采用 6×18×1 网络结构预测 2006 年的运量为 1829.25 百万吨，同有关资料公布的 2006 年货运量 1829 百万吨比较，预测的精度高很多。

(4) 多输入多输出 BP 网络的并行推理

采用多输入单输出的 BP 网络，在使用时，对一个输入模式只能得到一个输出结果。若采用多输入多输出的 BP 网络，那么，在使用时，对一个输入模式同时得到多个输出结果，称为神经网络的并行推理。

在本例中，可以采用 m 个输出节点的 BP 网络来建立预测模型，一次预测能同时获得连续 m 年的 m 个年运量的预测值。当然，对 m 个输出节点的 BP 网络，学习实例的输出模式 $Y_k^*=(y_{1k}^*,y_{2k}^*,\cdots,y_{mk}^*)$ 由已知的连续 m 年的年运量 $y_{1k}^*,y_{2k}^*,\cdots,y_{mk}^*$ 组成。

(5) 神经网络预测方法与其他方法的比较

采用传统的一元线性回归、指数曲线拟合和对数曲线拟合方法获得的 2000～2005 年的各年运量估计值和误差如表 7.14 所示。

表 7.14 三种传统预测方法的运量估计量与误差

年度	实际运量 y_k^*	一元线性回归		指数曲线拟合		对数曲线拟合	
		估计值 y_k	$y_k-y_k^*$	估计值 y_k	$y_k-y_k^*$	估计值 y_k	$y_k-y_k^*$
2000	1622	1629.36	7.36	1625.73	3.73	1637.18	15.18
2001	1611	1666.79	55.79	1664.42	53.42	1670.03	59.03
2002	1615	1679.11	64.11	1677.36	62.36	1680.44	65.44
2003	1685	1703.79	18.79	1703.58	18.58	1700.71	15.71
2004	1789	1728.02	−60.98	1729.71	−59.29	1719.93	−69.07
2005	1790	1755.24	−34.76	1759.56	−30.44	1740.75	−49.25

比较表 7.13 与表 7.14 的对应误差，可见 $6\times18\times1$ 和 $7\times21\times1$ 这两种 BP 网络的拟合情况是很好的，拟合精度大大超过传统的预测方法。

习 题 七

7.1 何谓机器学习？简述机器学习系统应具有的主要特征。

7.2 机器学习有哪些主要的学习方法？简述它们之间的区别。

7.3 简述遗传算法的主要特点和基本流程。

7.4 适应度函数在遗传算法中的作用是什么？试举例说明如何构造适应度函数。

7.5 何谓人工神经网络？人工神经网络的互连结构有哪几种基本类型？简述这几种类型在网络结构上的基本特征。

7.6 简述感知器的学习过程，给出感知器学习算法的流程框图。

7.7 为什么说人工神经网络是一个非线性映射系统？如果 BP 网络中的所有节点的传递函数都为线性函数，BP 网络还是一个非线性映射系统吗？为什么？

7.8 何谓有教师学习？何谓无教师学习？试分别举例说明神经网络的这两类学习方法。

7.9 BP 学习算法有哪些主要不足？试举一例说明可采取的改进方法。

7.10 为什么说 BP 学习算法的收敛速度由权值调整量 Δw 决定？Δw 的欠调和超调对学习过程有何影响？

7.11 简述神经网络专家系统的知识表示形式和推理机制，试举一例说明。

7.12 某商场随机抽取 8 种商品作为学习实例，通过归纳学习得出用于决策哪些商品进入超市销售的一般规则。实例的两个分类属性分别是：商品定价合理程度，商品品牌级别。分类结果是畅销(Y)和滞销(N)。分类属性的值域分别是

Value(价格合理)$=\{a_1,a_2,a_3\}$

Value(品牌级别)$=\{b_1,b_2,b_3\}$

初始属性表为 AttrList=(价格合理，品牌级别)。

8 个实例组成学习实例集 T 如下：

序 号	价格合理	品牌级别	类别	序 号	价格合理	品牌级别	类别
1	a_1	b_1	N	5	a_3	b_2	Y
2	a_1	b_2	Y	6	a_1	b_3	Y
3	a_2	b_2	N	7	a_3	b_2	Y
4	a_3	b_1	Y	8	a_1	b_1	N

(1) 应用 CLS 算法生成商品分类决策树。

(2) 由决策树得出产生式规则集。

7.13　根据圆柱体工件的外部尺寸加工误差对工件进行分类，3 个分类特征属性分别是：外圆直径误差范围、内圆直径误差范围、高误差范围；分类结果是：Y(可用)和 N(不可用)。特征属性的值域分别是：

Value(外圆直径误差范围)$=\{a_1,a_2,a_3\}$

Value(内圆直径误差范围)$=\{b_1,b_2,b_3\}$

Value(高误差范围)$=\{h_1,h_2\}$

初始属性表为 AttrList=(外圆误差，内圆误差，高误差)。

现有随机抽取的 8 个工件组成学习实例集 T 如下：

序号	外圆误差	内圆误差	高误差	类别	序号	外圆误差	内圆误差	高误差	类别
1	a_1	b_1	h_1	Y	5	a_3	b_2	h_1	N
2	a_1	b_2	h_2	N	6	a_1	b_2	h_1	Y
3	a_2	b_3	h_1	Y	7	a_3	b_2	h_2	N
4	a_3	b_1	h_1	N	8	a_1	b_3	h_2	N

(1) 应用 CLS 算法生成工件分类决策树。

(2) 由决策树得出产生式规则集。

(3) 对规则集中的规则进行合并，给出没有冗余的规则集。

(4) 若初始属性表为 AttrList=(内圆误差，外圆误差，高误差)，请重做本题。

(5) 应用 ID3 算法重做本题。

7.14　电信公司为了建立稳定的客户群，希望从现有客户的基本属性中挖掘出稳定客户的分类规则。根据固定电话的所在地区(DQ)、机主职业(JZZY)、月均主叫次数范围(YJZJ)、月均话费范围(YJHF)等四种属性，应用 ID3 算法归纳出哪些固话客户可发展为稳定客户的规则集合。属性的值域分别是：

Value(DQ)$=\{a_1,a_2,a_3\}$

Value(JZZY)$=\{b_1,b_2,b_3\}$

Value(YJZJ)$=\{c_1,c_2,c_3\}$

Value(YJHF)$=\{d_1,d_2\}$

初始属性表为 AttrList=(DQ,JZZY,YJZJ,YJHF)。

现从客户资料数据库中随机抽取 24 个客户组成学习实例集 T 如下：

序号	DQ	JZZY	YJZJ	YJHF	类别	序号	DQ	JZZY	YJZJ	YJHF	类别
1	a_2	b_1	c_1	d_1	N	13	a_2	b_2	c_1	d_1	N
2	a_2	b_1	c_2	d_1	N	14	a_2	b_2	c_3	d_1	N
3	a_2	b_1	c_3	d_1	N	15	a_2	b_3	c_1	d_2	Y
4	a_1	b_1	c_1	d_1	Y	16	a_2	b_3	c_3	d_2	Y

续表

序号	DQ	JZZY	YIZJ	YJHF	类别	序号	DQ	JZZY	YIZJ	YJHF	类别
5	a_1	b_1	c_3	d_1	Y	17	a_3	b_2	c_1	d_2	N
6	a_3	b_2	c_1	d_1	N	18	a_3	b_2	c_3	d_2	N
7	a_3	b_2	c_3	d_1	N	19	a_2	b_2	c_3	d_2	Y
8	a_3	b_1	c_1	d_2	Y	20	a_2	b_2	c_2	d_2	Y
9	a_3	b_3	c_3	d_2	N	21	a_1	b_2	c_2	d_1	Y
10	a_3	b_1	c_2	d_2	N	22	a_1	b_2	c_3	d_1	Y
11	a_1	b_3	c_2	d_2	Y	23	a_1	b_1	c_1	d_2	Y
12	a_1	b_3	c_3	d_2	Y	24	a_3	b_2	c_2	d_1	N

(1) 应用 ID3 算法生成决策树。

(2) 由决策树得出发展 VIP 客户的一般规则集。

7.15 某单位人力资源部希望根据现有员工的属性和工作情况，挖掘出人员招聘时的一般规则，供招聘时参考。随机抽取的在岗员工作为学习实例，实例采用 5 个分类属性：学历、性别、外语水平、性格特征、岗位性质。分类结果是：称职(Y)、不称职(N)。属性的值域分别是

Value(学历)={研究生，本科，专科}

Value(性别)={男，女}

Value(外语水平)={4 级以下，4 级，6 级}

Value(性格特征)={a_1，a_2，a_3}

Value(岗位性质)={b_1，b_2，b_3}

初始属性表为 AttrList=(学历，性别，外语水平，性格特征，岗位性质)。

现从员工数据库中随机抽取 16 名员工组成学习实例集 T 如下：

序 号	学 历	性 别	外语水平	性格特征	岗位性质	类 别
1	本科	男	4 级	a_1	b_1	Y
2	本科	女	6 级	a_3	b_1	N
3	本科	男	6 级	a_2	b_2	Y
4	本科	男	4 级	a_1	b_3	Y
5	本科	女	4 级	a_2	b_2	Y
6	本科	男	4 级以下	a_3	b_3	Y
7	本科	女	4 级	a_2	b_3	N
8	本科	女	4 级	a_1	b_1	Y
9	专科	男	4 级以下	a_1	b_1	N
10	专科	男	4 级	a_2	b_2	Y
11	专科	女	4 级	a_3	b_3	Y
12	专科	女	4 级以下	a_3	b_3	Y
13	研究生	男	6 级	a_2	b_2	Y
14	研究生	女	6 级	a_2	b_2	Y
15	研究生	男	4 级	a_1	b_3	N
16	研究生	女	6 级	a_1	b_1	Y

(1) 应用 ID3 算法生成员工招聘决策树。

(2) 由决策树得出一般规则集。

(3) 根据学历和外语水平的取值之间的关系，对规则集中的规则进行改进与合并，给出更简便的规则集。

7.16　应用遗传算法求 $f(x)=x^2$ 的最大值的解，$x\in[0,127]$。设群体规模 $n=6$，第一代群体 LT(1)=(110,86,68,47,32,17)。选择算子每次选择适应度最大的个体 3 次，淘汰适应度最小的 2 个个体，其他个体选择 1 次。

7.17　应用遗传算法求 $x^2-1=63$ 的解，$x\in[0,31]$。设群体规模 $n=4$，第一代群体 LT(1)=(24,19,13,8)。设适应度函数为 $f(x)=|x^2-64|$。选择算子每次选择适应度最小的个体 2 次，淘汰适应度最大的个体，其他个体选择 1 次。

主要参考文献

[1] 尹朝庆，尹皓. 人工智能与专家系统[M]. 北京：中国水利水电出版社，2002.

[2] 高济，朱淼良，何钦铭. 人工智能基础[M]. 北京：高等教育出版社，2002.

[3] 王永庆. 人工智能原理与方法[M]. 西安：西安交通大学出版社，1998.

[4] 王士同. 神经模糊系统及其应用[M]. 北京：北京航空航天大学出版社，1998.

[5] 蔡自兴，徐光佑. 人工智能及其应用[M]. 2 版. 北京：清华大学出版社，1996.

[6] 焦李成. 神经网络计算[M]. 西安：西安电子科技大学出版社，1996.

[7] 陈国良等. 遗传算法及其应用[M]. 北京：人民邮电出版社，1996.

[8] 吴泉源，刘江宁. 人工智能与专家系统[M]. 长沙：国防科技大学出版社，1995.

[9] 刘有才，刘增良. 模糊专家系统原理与设计[M]. 北京：北京航空航天大学出版社，1995.

[10] 刘叙华. 基于归结方法的自动推理[M]. 北京：科学出版社，1994.

[11] 何新贵. 模糊知识处理的理论与技术[M]. 北京：国防工业出版社，1994.

[12] Payne E C，McArthur R C. Developing Expert System[M]. John wiley &sons，New York，1990.

[13] Patterson D W. Introduction to Artificial Intelligence and Expert System[M]. Prentice Hall，New Jersey，1990.

[14] Krentzer W，Mckenzie B. Programming for Artificial Intelligence，Methods，Tools and Applications [M]. Addison-Wesley，1991.

[15] Horikawa S. On fuzzy modelling using fuzzy neural networks with BP algorithm[J]. IEEE Trans. Neural Networks，No. 2，1992.

[16] Pedrycz W. Fuzzy neural networks with reference neurons as pattern classifiers[J]. IEEE Trans. Neural Networks，No. 3，1992.

[17] Mitra S. Self-organizing neural network as a fuzzy classifier[J]. IEEE Trans. On System，Man，Cybernetics，No. 3 1994.