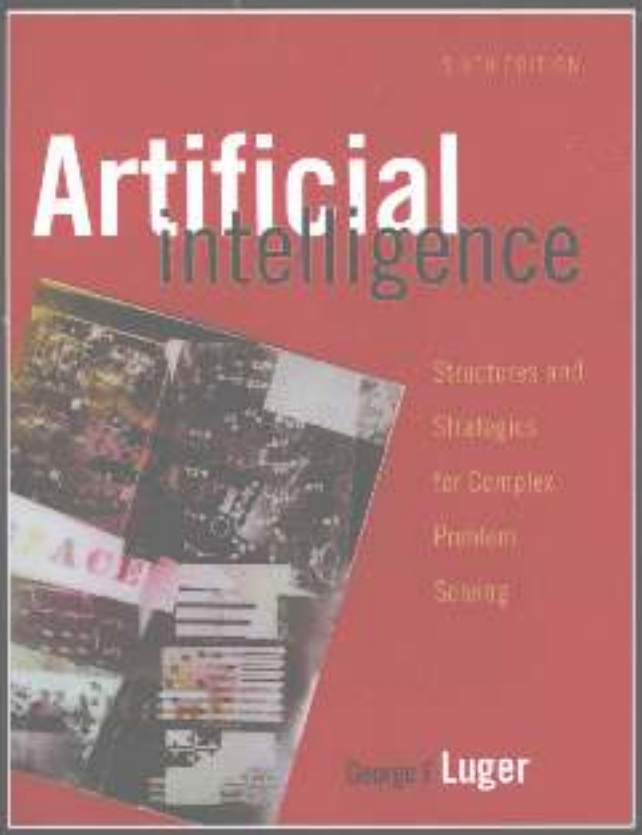
计 算 机 科 学 丛 书

原书第6版

人工智能

复杂问题求解的结构和策略

(美) George F. Luger 著 郭茂祖 刘 扬 玄 萍 王春宇 等译



Artificial Intelligence

Structures and Strategies for Complex Problem Solving

Sixth Edition

机械工业出版社 China Machine Press

PEARSON Education

人工智能 复杂问题求解的结构和策略（原书第6版）

"本书对人工智能课程的方方面面，包括最新的进展，都做了很好的介绍。本书的特点是对每个问题都给出了详细的、易懂的伪代码描述。"

—— Jonic，圣迭戈州立大学

"本书是人工智能课程的完美教科书，它既给出了现有技术的全面概述，又给出了一些前沿的研究。这是一本可读性很强的人工智能书籍。"

—— Paolo Pecevski，波尔多拉斯大学

"这本书写作风格和全面的程度使它成为人工智能领域很有价值的文献。"

—— Ivan V. Cibin，利兹城市大学

本书是一本经典的人工智能教材，全面阐述了人工智能的基础理论，有效结合了求解智能问题的数据结构以及实现的算法，把人工智能的应用程序应用于实际环境中，并从社会和哲学、心理学以及神经生理学角度对人工智能进行了全面的讨论。

本版新增内容

- 新增一章，介绍用于机器学习的随机方法，包括一阶贝叶斯网络、各种隐马尔可夫模型、马尔可夫随机场推理和循环信念传播。
- 介绍针对期望最大化学习以及利用马尔可夫链蒙特卡罗抽样的结构化学习的参数选择，强化学习中马尔可夫决策过程的利用。
- 介绍智能体技术和本体的使用。
- 介绍自然语言处理的动态规划（Earley语法分析）以及Viterbi等其他概率语法分析技术。
- 书中的许多算法采用Prolog、LISP和Java语言来构建。

作者简介



George F. Luger 1973年在宾夕法尼亚大学获得博士学位，并在之后的5年间在爱丁堡大学人工智能系进行博士后研究。现在是新墨西哥大学计算机科学研究、语言学及心理学教授。



影印版
ISBN 978-7-111-29059-4
定价：45.00元

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379604
读者信箱：hzjsj@hzbook.com
华章网站 http://www.hzbook.com
网上购书：www.china-pub.com
PEARSON
www.pearsonhighered.com



ISBN 978-7-111-28345-4



定价：79.00元

计 算 机 科 学 丛 书

原书第6版

人工智能

复杂问题求解的结构和策略

[美] George F. Luger 著 郭茂祖 刘扬 玄萍 王春宇 等译

Artificial Intelligence

Structures and Strategies for Complex Problem Solving

Sixth Edition

机械工业出版社
China Machine Press

本书是一本经典的人工智能教材，全面阐述了人工智能的基础理论，有效结合了求解智能问题的数据结构以及实现的算法，把人工智能的应用程序应用于实际环境中，并从社会和哲学、心理学以及神经生理学角度对人工智能进行了独特的讨论。新版中增加了“基于随机方法的机器学习”等内容，并提出了一些新的主题，如涌现计算、本体论、随机分割算法等。

本书适合作为高等院校计算机专业人工智能教材，也可供人工智能领域的研究者及相关工程技术人员参考。

Simplified Chinese edition copyright © 2010 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Sixth Edition* (ISBN 978-0-321-54589-3) by George F. Luger, Copyright © 2009 by Pearson Education Inc.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Addison-Wesley.

本书封面贴有 Pearson Education（培生教育出版集团）激光防伪标签，无标签者不得销售。

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2009-1290

图书在版编目（CIP）数据

人工智能：复杂问题求解的结构和策略（原书第 6 版）/（美）卢格（Luger, G. F.）著；郭茂祖等译. —北京：机械工业出版社，2009.12

（计算机科学丛书）

书名原文：Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Sixth Edition

ISBN 978-7-111-28345-4

Ⅰ. 人…　Ⅱ. ①卢…　②郭…　Ⅲ. 人工智能－研究　Ⅳ. TP18

中国版本图书馆 CIP 数据核字（2009）第 172303 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）

责任编辑：迟振春

北京市荣盛彩色印刷有限公司印刷

2010 年 1 月第 1 版第 1 次印刷

184mm × 260mm · 31.75 印张

标准书号：ISBN 978-7-111-28345-4

定价：79.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991；88361066

购书热线：（010）68326294；88379649；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@ hzbook. com

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域获得了坚实的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰斗北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅奠划了研究的范畴，还揭示了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展试猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短的现状下，美国等发达国家在其计算机科学发展的几十年间积淀和发展的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起到积极的推动作用，也是与世界接轨，建设真正的世界一流大学的必由之路。

机械工业出版社华章分社较早意识到“出版要为教育服务”。自 1998 年开始，华章分社就将工作重点放在了引进、翻译国外优秀教材上。经过多年的不懈努力，我们与 Pearson，McGraw-Hill，Elsevier，MIT，John Wiley & Sons，Cengage 等世界著名出版公司建立了良好的合作关系，从他们现有的数百种教材中甄选出 Andrew S. Tanenbaum，Bjarne Stroustrup，Brian W. Kernighan，Dennis Ritchie，Jim Gray，Afred V. Aho，John E. Hopcroft，Jeffrey D. Ullman，Abraham Silberschatz，William Stallings，Donald E. Knuth，John L. Hennessy，Larry L. Peterson 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及珍藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅开列了书单，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近两百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍。其影印版“经典原版书库”作为姊妹篇也被越来越多实施双语教学的学校所采用。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证。随着计算机科学与技术专业学科建设的不断完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都将步入一个新的阶段，我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。华章分社欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

华章网站：www.hzbook.com

电子邮件：hzsj@hzbook.com

联系电话：(010) 88379604

联系地址：北京市西城区百万庄南街 1 号

邮政编码：100037

华章教育

华章科技图书出版中心

译者序

人工智能（AI）最开始的动机是想创造一种机器，它不仅能够思考，而且还有可能和人类交流，这是计算的最高级目标。在过去的许多年里，人工智能的研究者们在探索智能机制的同时，还使人工智能在实际领域取得了更广泛的应用。我们可以使用不同的人工智能策略解决很多在实际应用计算机技术时出现的复杂问题。众所周知，智能本身是非常复杂的，难以用单一的理论来描述。因此，产生了一系列的理论从不同的抽象层次刻画这个主题。在最低层次，神经网络、遗传算法以及其他形式的理论可以辅助理解适应性原理、感知机制以及与物理世界的交互机制。在更加抽象的层次，专家系统的设计、智能主体、随机模型以及自然语言理解程序反映了知识在智能中的角色和创建、传递、保持的社会过程。更深一层，逻辑学家提出了演绎、反绎、归纳、真值维护以及其他的推理模型和方式。

在第6版中，George F. Luger 阐述了复杂问题求解结构和策略的所有这些层次的理论，同时，他还指出了智能研究本身的令人兴奋之处，演示了怎样使用不同的软件工具和技术去解决计算机科学家面临的复杂问题。

这本畅销教材的主要特点是：

- 彻底和全面阐述人工智能的基础理论。
- 有效结合了求解智能问题的数据结构以及实现的算法。
- 用 LISP、Prolog 或 Java 语言编写实例程序。
- 把人工智能的应用程序应用于实际环境中。
- 从社会和哲学角度出发对人工智能进行全面的讨论。

与第5版相比，第6版主要扩充了获取人工智能的随机方法的相关材料，包括：

- 修改9.3节、加入第13章来介绍基于概率的机器学习。
- 扩充了关于有限状态自动机和概率接受器以及动态规划的使用，尤其是使用随机测量的动态规划算法（Viterbi 算法）的例子。

此外，第6版除了介绍 Prolog 和 LISP，还收集了一些用 Java 实现的人工智能算法。

本书共16章。第1章（第一部分）简单介绍人工智能；第2、3、4、5、6章（第二部分）介绍 AI 问题求解的研究工具；第7、8、9章（第三部分）介绍人工智能和知识密集型问题求解以及变化和模糊情况下推理的表示法；第10、11、12、13章（第四部分）提供机器学习中问题的扩充说明；第14、15章构成本书的第五部分，第14章讨论这一领域中最早的程序，第15章介绍自然语言理解；第16章（第六部分）作为本书的结束语。

本书适合作为高等院校计算机、自动化、电子等专业本科生及研究生的人工智能教材。同时，本书也是人工智能领域的研究者或那些想了解和应用当前人工智能技术的工作人员的一本宝贵的参考资料。

本书第6版的翻译工作由郭茂祖主持，郭茂祖审校了全部译稿，玄萍负责校对。其中，郭茂祖翻译了第1、2、3、4、5章，刘扬翻译了第7、8、9、10章，玄萍翻译了第11、12、13、14章，王春宇翻译了第6、15、16章。在本书的翻译过程中，李建伏、邓超、邹权、于建涛、王峻、李艳娟、吴伟宁、徐磊、艾森、邢志安给予了很多帮助，对他们表示由衷的感谢。另外，本书的翻译参考了第5版的中文版，在此一并表示感谢。

前 言

我们通过做来学习我们必须做的……

——亚里士多德《伦理学》(Ethics)

欢迎阅读第6版

我很高兴被邀请写作这本《人工智能》教材的第6版。这是对20多年来前几个版本的褒奖，说明我们获取AI的方法得到了大家的高度评价。同样令人兴奋的是，随着人工智能领域的新的发展，我们能够在每个新版本中介绍这些新发展和新技术。我们感谢许多读者、同事和学生督促我们及时地更新相关主题。

前几版中的许多章节现在仍然具有很好的适用性，包括对逻辑、搜索算法、知识表示、产生式系统、机器学习以及用LISP、Prolog和本版中的Java开发的程序设计技术的介绍。这些仍然是人工智能的中心问题，也是这个新版本的重点。

这本书仍然是容易理解的。我们介绍了关键的表示技术，包括逻辑、语义和连接主义网络、图模型等。我们首先用伪代码的形式清楚地描述了搜索算法，然后在补充材料中介绍了许多用Prolog、LISP和/或Java实现的搜索算法。希望有学习主动性的学生能够掌握我们这些核心实现并把它们扩展到新的应用领域。

第6版中增加了新的一章（第13章）来介绍基于随机方法的机器学习。随机技术对人工智能的影响日益增大，尤其是在诊断推理、预测推理、自然语言理解、机器人学和机器学习这些领域。为了支持这些新兴的技术，我们扩充了对贝叶斯定理、马尔可夫模型、贝叶斯网络以及相关图模型的介绍。另外，还扩充讨论了概率有限状态自动机、隐马尔可夫模型、基于Earley解析器和Viterbi算法实现的动态规划。其他一些主题（如涌现计算、本体论、随机分割算法）在前几版中写得有些仓促，现在已成为比较重要的主题，值得更加全面地讨论。第6版中所做的修改反映出了新出现的人工智能问题，同时也是人工智能领域具有持久活力的证明。

随着AI研究课题的不断扩大，我们得到了很多人（包括出版商、编辑、朋友、同事，尤其是广大读者）的支持，他们使我们的工作具有长久而又有意义的生命力。能得到这样的机会，我们仍然很兴奋：因为很少有人会鼓励科学家从他们自己较窄的研究兴趣出发写出宏篇巨著。我们对出版商和读者为我们提供这样的机会表示感谢。我们也很受鼓舞，因为本书前几版已经用于全世界许多人工智能领域，而且被翻译成了许多不同语言（包括德语、波兰语、葡萄牙语、俄语以及中文简体和繁体）。

虽然人工智能和大多数工程学科一样，必须通过为实际问题提供解决方案来证明自己对商业领域的价值，但是我们与许多同事和学生却以相同的原因进入AI领域：我们想要理解和探索能够进行智能思考和动作的头脑的工作机制。我们避免使用“智能是人类的特有能力”这样粗浅的说法，而是相信我们能够通过设计和评估人工智能产物来有效地研究可能的智能空间。尽管我们的经历使我们没有理由改变初衷，但是我们的工作在广度、复杂度和创新性方面又上了一个新的台阶。在前面版本的前言中，我们概括了三点主张，我们相信坚持这三点主张可以使我们的方法在人工智能教学过程中取得卓越的成效。在这一版的前言中，自然要回到这些主题看看它们是如何随着领域发展而延续的。

第一点主张是通过详细讨论人工智能的理论基础来统一不同的分支。在我们首次采纳这点主张时，主要问题似乎是协调两种研究者，一种着重于对人工智能进行详细描述和形式化理论分析（整洁派），另一种相信智能本身是某种豪华的工具并且能够用应用驱动的特定方法获得（杂乱派）。事实证明，这种二分法过于简单。

在当代的 AI 中，整洁派和杂乱派之间的争论，已经引起了物理符号系统支持者和神经网络研究者、逻辑学家和非逻辑方式进化人工生命形式的设计者、专家系统建造者和基于案例的推理者以及相信已经获得人工智能的人和认为人工智能永远无法获得的人之间的许多争论。AI 是一门前沿科学，在这里，离经叛道者、探索者、极端预言家和其他梦想家被形式主义和经验主义慢慢驯服。我们最初的这种印象现在让位于一个不同的比喻：在一个巨大、无秩序但大体上和平的城市里，有秩序的中产阶级邻居从不同的、混乱的、狂放不羁的地区来此谋生。在我们致力于写作本书不同版本的几年中，一幅关于智能结构的引人注目的图画开始从城市的结构、艺术和工业中显现出来。

智能太复杂了，无法用任何单一的理论来描述；研究者正在构建一个从多个抽象层次上刻画智能的理论层次结构。在这个结构的底层，神经网络、遗传算法和其他形式的涌现计算使我们能理解所有形式的智能行为之下必然隐含的适应、感知、表现和与物质世界交互的过程。通过某些仍然是部分理解的归结，这个由盲目原始的参与者构成的混乱群体引出了逻辑推理的更冷静模式。在这个更高的层次上，逻辑学家在亚里士多德贡献的基础上，描绘演绎、反绎、归纳、真值维护和推理的无数其他方式和方法。在更高层抽象上，专家系统、智能主体和自然语言理解程序的设计者渐渐认识到社会过程在产生、传播和维持知识方面所起的作用。

从这一点来看，在 AI 事业中，理性主义和经验主义两种极端都只能得到一些片面的结论，两种极端的适用性和普遍性都受到限制。作者采取了另一种态度，即认为经验主义者的条件作用（语义网、脚本、包容结构）和理性主义者清晰明确的观点（谓词演算、非单调逻辑、自动推理）构成了第三个观点——贝叶斯定理。关联不变性的经验以智能的多个智能主体的期望为条件，学习这些不变性反过来可以使将来的期望发生偏离。作为哲学家，我们有责任分析人工智能事业在认识论上的合法性。为完成这一任务，我们在第 16 章讨论了理性主义工程、经验主义难题，并提出了一个基于贝叶斯的构造论者的一致观点。在第 6 版中，我们触及了呈现这个 AI 事业的所有层次。

我们在前面版本中提出的第二点主张是在 AI 方法学中高级表示形式和搜索技术占有中心位置。这可能是前面版本及 AI 许多早期工作中最有争议的方面，许多涌现计算的研究者质疑符号推理和指称语义在思考中是否有作用。虽然为事物给定名字的表示法思想受到了神经网络或人工生命的涌现模式提供的隐含表示法的挑战，但我们相信对于任何严谨的人工智能研究者，理解表示法和搜索仍然是基本的。同时，我们认为第 1 章对历史传统和先驱的回顾是人工智能教育不可或缺的部分。而且，高级表示形式和搜索技术也是分析非符号人工智能的多个方面（如分析神经网络的表达能力或者通过遗传算法的适应度范围计算候选问题解的进展）的无价工具。第 16 章中对现代 AI 的不同方法进行了比较和评论。

我们在前面版本中提出的第三点主张是将人工智能放到经验科学的上下文中。引用以前版本中 Newell and Simon（1976）的图灵奖演讲：

> ……AI 不是对科学传统的某种奇怪的偏离，而……是对关于智能知识的寻求和对智能本身的理解的一部分。另外，AI 程序设计工具和研究程序设计的方法学……对探究外界环境都是完美的。我们的工具提供了一种理解和质疑的方法。我们通过逐步近似来察觉和了解自然现象。
>
> 这样，将每个设计和程序都看成是对自然的一个试验：我们提出一个表示，生成一个搜索算法，并质疑这样的描述是否能够适当地说明部分智能现象。自然世界给我们的疑问一个回答。我

们的实验可能会被拆解、修订、扩充，然后再次运行。模型可能被改进，同时我们的理解范围也进一步扩大了。

第 6 版的新内容

第 6 版最主要的修改是扩充了获取人工智能的随机方法。为此，我们修改了 9.3 节，并且加入了第 13 章来介绍基于概率的机器学习。我们更加全面地介绍了随机的人工智能工具及其在学习和自然语言中的应用。

我们从集合论和计数的基础知识开始，逐步引出概率、随机变量和独立性等概念。接着，以一种症状和一种疾病为例，介绍并使用了贝叶斯定理，然后将其推广到最通用的形式。我们分析研究了应用贝叶斯定理的假设，介绍了 argmax 和简单贝叶斯方法。我们介绍了随机推理的一些例子，包括语言现象分析和 Viterbi 算法。我们还介绍了条件独立性的思想，随后在第 9 章引出了关于贝叶斯信念网络（BBN）的介绍。

第 13 章介绍了隐马尔可夫模型（HMM），并通过几个例子说明了其使用方法。我们也介绍了 HMM 的几种变形，包括自回归和层次化的隐马尔可夫模型，还介绍了动态贝叶斯网络（DBN），并举例说明了它们的使用方法。我们讨论了参数和结构学习，介绍了期望最大化方法并举例说明了它们在松散信念传播中的使用。最后，我们在扩展早期的强化学习表示的基础上，介绍了马尔可夫决策过程（MDP）以及部分可观测的马尔可夫决策过程（POMDP）。

我们扩充了一些实例，这些例子用来解释概率有限状态自动机和概率接受器以及动态规划的使用，尤其是解释如何使用随机测量的动态规划算法（Viterbi 算法）。我们添加了一个随机英语语法分析器（以爱丁堡大学 Mark Steedman 的工作为基础）以及 Earley 解析器中动态规划的使用。

我们做的一个主要决定是从本书中删除介绍 Prolog 和 LISP 的章节，部分原因是它们太庞大了。我们也收集了一些用 Java 实现的人工智能算法。当增加新的一章（第 13 章）来介绍基于随机方法的机器学习时，我们觉得这本书太厚、太笨重了。这样，第 6 版比第 5 版少了 150 多页（指的是英文原版书），以 Prolog、LISP 和 Java 语言实现的 AI 算法以补充材料的形式发行。在人工智能的研究刚开始时，我们总是认为这样的方法对于理解 AI 算法的优势（和局限性）是非常有益的——那就是要实现这些算法！我们鼓励这一代的读者切实这样去做：浏览这些补充材料，实现我们提出的这些算法并进行实验。

最后，我们按照惯例更新了参考文献[⊖]和一些材料。在修订后的第 16 章中，我们又回来讨论智能本性的更深入问题和产生智能机器的可能性。

第 6 版的内容

第 1 章简单介绍人工智能。我们从哲学、心理学和其他研究领域中试图了解头脑和智能的历史。从重要意义上讲，AI 是一门古老的科学，至少可以追溯到亚里士多德。对这些背景的了解是理解现代研究中主要问题的基本条件。我们还介绍了 AI 中一些重要应用领域的概要情况。第 1 章的目的是为后面的理论和应用提供背景知识和动机。

第 2、3、4、5、6 章（第二部分）介绍 AI 问题求解的研究工具。第 2 章介绍描述问题本质特征的谓词演算，谓词演算既是一种数学系统，也是一种表示语言。第 3 章介绍搜索以及用来实现搜索的算法和数据结构，搜索技术用来组织问题情景的探测。第 4 章讨论启发式在聚焦和约束

⊖ 限于篇幅，“参考文献”以 PDF 形式放在本书的网站（http://www.hzbook.com）上，有需要的读者可下载使用。——编辑注

基于搜索的问题求解中所起的基本作用。第5章介绍随机方法，这是一种在不确定情况下推理的重要技术。第6章介绍构建这些搜索算法所需的软件体系结构，包括黑板和产生式系统。

第7、8、9章构成本书的第三部分：人工智能、知识密集型问题求解、变化和模糊情况下推理的表示法。第7章介绍AI表示法的发展历程。我们从对基于关联的网络的讨论开始，扩展这个模型以包含概念依赖理论、框架和脚本。然后介绍一种特别的形式方法——概念图的深入检验，强调知识表示中包含的认识论问题，并且说明这些问题是如何在现代表示语言中表示的。在第14章扩展了这个形式方法，说明了如何用概念图来实现自然语言数据库前端。第7章的最后介绍更多的现代表示方法，包括Copycat结构和面向主体的结构。

第8章介绍基于规则的专家系统以及基于案例和基于模型的推理系统，包括来自NASA空间计划的例子。这些问题求解方法是作为第二部分内容的自然延伸来进行介绍的：用谓词演算表达式的产生式系统来协调一个图搜索。这一章最后分析每种知识密集型问题求解方法的优势和不足。

第9章介绍用不确定信息和不可靠信息进行推理的模型。我们讨论用于不确定情况推理的贝叶斯模型、信念网络、Dempster－Shafer、因果模型以及Stanford确信度代数。第9章还包括真值维护算法、最小模型推理、基于逻辑的反绎以及贝叶斯信念网络的团树算法。

第四部分包括第10章到第13章，提供机器学习中问题的扩充说明。在第10章，我们对基于符号的学习算法进行了详细的介绍，这是一个硕果累累的研究领域，产生了大量的问题和解决方法。这些学习算法在其目标、训练数据、学习策略和使用的知识表示法上各不相同。基于符号的学习包括归纳、概念学习、变形空间搜索和ID3。归纳偏置的作用也要考虑，即从数据模式中泛化，同时还要考虑在基于解释的学习中有效利用从单个示例中学习到的知识。分类学习（或者说概念聚类）与无监督学习一起介绍。这一章最后介绍强化学习，即把来自环境的反馈结合到决策策略中的能力。

第11章介绍了神经网络，这类网络经常称为学习的子符号或连接模型。在神经网络中，信息隐含在一个连接的处理机集合的组织和权重中，学习包括结点权重和系统结构的重新排列和修改。我们介绍了许多连接结构，包括感知机学习、反传和逆传。我们展示了Kohonen、Grossberg和Hebbian模型。我们介绍了联想学习和吸引子模型，包括Hopfield网络的例子。

第12章介绍学习的遗传算法和进化方法。从这个观点来看，学习可以看成是一个涌现和适应过程。在介绍了几个基于遗传算法的问题求解实例后，我们介绍遗传技术在更通用的问题求解器中的应用，这其中包括分类器系统和遗传程序设计。然后结合人工生命研究中的例子描述基于社会的学习。我们用圣达菲研究所（Santa Fe Institute）关于涌现计算研究的一个例子结束本章。

第13章介绍机器学习的随机方法。首先定义隐马尔可夫模型，然后介绍几个重要的变形，包括自回归和层次化的隐马尔可夫模型。我们接下来介绍动态贝叶斯网络（是HMM的一个推广），它也能够通过时间周期跟踪系统。对于在诊断和预测推理中模拟复杂环境的改变，这些技术是非常有用的。最后，我们为第10章介绍的强化学习增加了一个概率部分，介绍了马尔可夫决策过程（MDP）和部分可观测的马尔可夫决策过程（POMDP）。

第五部分包括第14章和第15章，介绍自动推理和自然语言理解。定理证明（常常称为自动推理）是AI研究最早的领域之一。在第14章，我们讨论这一领域中最早的程序，包括逻辑理论家和通用问题求解器。本章的焦点是二元归结证明过程，特别是归结反驳，还介绍了使用超归结和参数调制的高级推理。最后，我们把Prolog解释器描述为一个基于Horn子句和归结的推理系统，并将Prolog计算看成是逻辑程序设计范例的一个实例。

第15章介绍自然语言理解。我们在第7章中已经介绍过传统的自然语言理解方法，就是用

许多语法结构进行例示。这里我们又补充了随机方法，包括马尔可夫模型、图解析（Earley 算法）、交互信息聚类和基于统计的解析。本章的最后是几个例子，包括将这些自然语言理解技术应用于数据库查询系统（一个文本摘要系统）以及使用机器学习方法去泛化从万维网获得的结果。

最后，第 16 章是本书的后记。其中提到了智能系统科学的可能性问题，考虑了现在 AI 面临的挑战，讨论了目前 AI 的限制，并设计了 AI 激动人心的未来。

使用本书

人工智能是一个很大的领域，因此这是一本很厚的书。尽管学习本书时可能会需要超过一个学期才能覆盖里面所有的内容，但是我们对本书进行了仔细设计，使得大家可以采取不同的路径来学习其中的内容。通过选择内容的子集，我们既可以将这本教材用于一个学期的课程，也可以用于整个学年（两个学期）的课程。

我们假设绝大多数学生已经学习了离散数学的入门课程，包括谓词演算、集合论、计数和图论。如果学生没有学过这些课程，那么教师应该花更多的时间讲解本书开始部分（2.1 节、3.1 节和 5.1 节）的那些概念。我们还假设学生已经学习了数据结构课程，包括树、图、递归搜索，会使用堆栈、队列和优先级队列。如果学生没有学过这些课程，那么多花些时间在第 3、4、6 章的开始部分。

在一个学期的课程中，我们会很快扫过本书的前两部分。有了这个准备，学生就能够看懂第三部分的内容。接着考虑补充材料中的 Prolog、LISP 或者 Java 代码，要求学生实现第二部分中的主要表示和搜索技术。也可以选择在课程中早点介绍一门语言（如 Prolog），来检验遇到的数据结构和搜索技术。我们觉得语言部分介绍的元解释器对于构建基于规则的和其他知识密集型问题的求解器很有帮助。Prolog、LISP 和 Java 都是构建自然语言理解和学习系统的极好工具，这些结构在第二部分和第三部分加以介绍，在补充材料中有相关例子。

在两个学期的课程中，可以覆盖第四部分和第五部分中的所有应用领域，特别是机器学习的几章，要进行较为详细的讲解。我们也期望学生做一个更详细的程序设计项目。我们认为这对于学生在第二学期复习 AI 文献中的许多基本资源非常重要。让学生知道我们处于 AI 事业发展中的什么位置、我们如何到达这里并且让学生对人工智能未来的进展有一个认识都是至关重要的，为此我们使用互联网上的资料或者选择一个读物合集，比如《Computation and Intelligence》（Luger 1995）。

本书中的算法采用类 Pascal 的伪代码进行描述。符号中使用了 Pascal 的控制结构及英语描述的检查和操作。我们在 Pascal 控制结构中加入了两个有用的结构。第一个是修改过的 case 语句，它使每一个分支都可以用一个任意的布尔检检来标记，而不是像在标准 Pascal 中只能将一个变量的值与不变的 case 标记相比较。case 依次测试这些条件，直到一个为真，然后执行相关的动作；所有其他动作被忽略。熟悉 LISP 的人会注意到这和 LISP 的 cond 语句有同样的语义。

另一个增加的有用结构是 return 语句，它可以有一个参数，可以出现在过程或函数中的任何位置。当遇到 return 时，会使程序立即退出该函数，将其参数作为结果返回。除此之外，我们都使用 Pascal 的结构，再加上英语描述，这可以使算法清晰易读。

可以获得的补充材料

第 6 版附带了由我的研究生维护的网站。这个网站由新墨西哥大学的两个研究生 Alejandro CdeBaca 和 Cheng Liu 建立。网站的内容包括大多数章节后面建议的补充材料、一些示例问题及其解决方法以及对学生项目的许多建议。除了补充材料中的 Prolog、LISP 和 Java 程序外，网站上还有许多用 Java 和 C ++ 写的 AI 算法。欢迎同学们使用这些材料，并添加自己的评论、代码和批

评。网站地址是 www. cs. unm. edu/ ~ luger/ai - final/。

我的 E-mail 地址是 luger@ cs. unm. edu，欢迎各位读者来信交流心得体会。

致谢

虽然我是第 6 版的署名作者，但本书一直是我在新墨西哥大学做计算机科学、心理学和语言学教授时努力的成果，其中也离不开我的同事们、研究生和朋友们的贡献。第 6 版同时也是许多读者通过 E-mail 给我评价、更正和建议的成果。本书将继续这种方法，反映“团体”的成果。因此，介绍内容时，我将继续使用“我们”、“我们的”这些人称代词。

我要感谢 Bill Stubblefield，他是本书前 3 版的合著者，对本书有 20 多年的贡献，更重要的是我们的友谊。我也要感谢许多帮助完成第 6 版和以前版本的审稿人，包括 Dennis Bahler、Leonardo Bottaci、Skona Brittain、Philip Chan、Peter Collingwood、Mehdi Dastani、John Donald、Sarah Douglas、Christophe Giraud-Carrier、Andrew Kosoresow、Terran Lane、Chris Malcolm、Ray Mooney、Marek Perkowski、Barak Pearmutter、Dan Pless、Bruce Porter、Stuart Shapiro、Julian Richardson、Jude Shavlik、John Sheppard、Carl Stern、Leon van der Torre、Marco Valtorta 和 Bob Veroff。我们还要感谢读者通过 E-mail 给我们寄来无数建议和评价。最后，感谢 Chris Malcolm、Brendan McGonnigle 和 Akasha Tang 审定第 16 章。

我们新墨西哥大学的同事中，感谢 Dan Pless、Nikita Sakhanenko、Roshan Rammohan 和 Chayan Chakrabarti 在扩充第 5、9 和 13 章的材料中起到了主要作用，感谢 Joseph Lewis 对第 9 章和第 16 章的贡献，感谢 Carl Stern 帮助扩充第 11 章连接学习，感谢 Bob Veroff 审定第 14 章中自动推理部分的材料，感谢 Jared Saia、Stan Lee 和 Paul dePalma 帮助完成第 15 章自然语言理解的随机方法。

我们要感谢 Academic Press 允许重印第 11 章中的许多材料，这些材料最早出版于《Cognitive Science：The Science of Intelligent Systems》（Luger 1994）。最后，我们感谢 20 多年来新墨西哥大学使用本书的学生，他们拓宽了我们的视野，帮助改掉了书中许多印刷错误和失误。

我们感谢 Benjamin-Cummings、Addison-Wesley-Longman 和 Pearson Education 的许多朋友，他们对我们完成第 6 版的写作工作给予了很多支持和鼓励。特别是 Alan Apt 帮助我们完成第 1 版，Lisa Moller 和 Mary Tudor 帮助我们完成第 2 版，Victoria Henderson、Louise Wilson 和 Karen Mosman 帮助我们完成第 3 版，Keith Mansfield、Karen Sutherland 和 Anita Atkinson 帮助我们完成第 4 版，Keith Mansfield、Owen Knight、Anita Atkinson 和 Mary Lince 帮助我们完成第 5 版，以及 Simon Plumtree、Matt Goldstein、Joe Vetere 和 Sarah Milmore 帮助我们完成第 6 版。Addison-Wesley 美国公司的 Katherine Haratunian 在分发教师指导手册和 PowerPoint 演示材料工作上起了巨大的作用（这些是由 Addison-Wesley Pearson 维护的，只能通过读者当地的销售代表得到）。感谢新墨西哥大学的 Linda Cicarella 帮助我们准备了许多图表。

我们感谢 Thomas Barrow，一位国际知名的艺术家、新墨西哥大学艺术教授（退休），他绘制了书中的插图。

人工智能是一门激动人心的、回报丰厚的学科，祝你在认识到其力量和挑战的同时享受学习的快乐。

George Luger

2008 年 1 月 1 日

Albuquerque

目 录

出版者的话
译者序
前言

第一部分 人工智能的历史渊源及研究范围

第1章 人工智能的历史及应用 …… 3
1.1 从伊甸园到第一台电子计算机：对智能、知识和人类技能的态度 …… 3
1.1.1 人工智能基础的简要历史 …… 4
1.1.2 理性主义和经验主义学派对人工智能的影响 …… 6
1.1.3 形式逻辑的发展 …… 7
1.1.4 图灵测试 …… 9
1.1.5 智能的生物和社会模型：主体理论 …… 11
1.2 人工智能应用领域概述 …… 14
1.2.1 博弈 …… 15
1.2.2 自动推理和定理证明 …… 15
1.2.3 专家系统 …… 16
1.2.4 自然语言理解和语义学 …… 17
1.2.5 对人类表现建模 …… 18
1.2.6 规划和机器人学 …… 18
1.2.7 人工智能的语言和环境 …… 19
1.2.8 机器学习 …… 20
1.2.9 其他表示：神经网络和遗传算法 …… 20
1.2.10 AI和哲学 …… 21
1.3 人工智能小结 …… 22
1.4 结语和参考文献 …… 22
1.5 习题 …… 23

第二部分 作为表示和搜索的人工智能

第2章 谓词演算 …… 32
2.0 简介 …… 32
2.1 命题演算（选读） …… 32
2.1.1 符号和语句 …… 32
2.1.2 命题演算的语义 …… 33
2.2 谓词演算 …… 35
2.2.1 谓词的语法和语句 …… 35
2.2.2 谓词演算的语义 …… 39
2.2.3 语义含义的积木世界例子 …… 42
2.3 使用推理规则产生谓词演算表达式 …… 43
2.3.1 推理规则 …… 43
2.3.2 合一算法 …… 45
2.3.3 合一的例子 …… 48
2.4 应用：一个基于逻辑的财务顾问 …… 50
2.5 结语和参考文献 …… 53
2.6 习题 …… 54
第3章 状态空间搜索的结构和策略 … 56
3.0 简介 …… 56
3.1 状态空间搜索的结构 …… 58
3.1.1 图论（选读） …… 58
3.1.2 有限状态自动机（选读） …… 60
3.1.3 问题的状态空间表示 …… 61
3.2 用于状态空间搜索的策略 …… 65
3.2.1 数据驱动搜索和目标驱动搜索 …… 65
3.2.2 图搜索的实现 …… 67
3.2.3 深度优先搜索和宽度优先搜索 …… 69
3.2.4 迭代加深的深度优先搜索 …… 74
3.3 利用状态空间来表示命题演算和谓词演算的推理 …… 75
3.3.1 逻辑系统的状态空间描述 …… 75
3.3.2 与或图 …… 76
3.3.3 进一步的例子和应用 …… 77
3.4 结语和参考文献 …… 84
3.5 习题 …… 85
第4章 启发式搜索 …… 86
4.0 简介 …… 86
4.1 爬山法和动态规划法 …… 89
4.1.1 爬山 …… 89
4.1.2 动态规划 …… 90

4.2 最佳优先搜索算法 …… 92
4.2.1 实现最佳优先搜索 …… 92
4.2.2 实现启发评估函数 …… 95
4.2.3 启发式搜索和专家系统 …… 100
4.3 可采纳性、单调性和信息度 …… 101
4.3.1 可采纳性度量 …… 101
4.3.2 单调性 …… 102
4.3.3 信息度更高的启发是更好的启发 …… 103
4.4 在博弈中使用启发 …… 104
4.4.1 在可穷举搜索图上的极小极大过程 …… 104
4.4.2 固定层深的极小极大过程 …… 106
4.4.3 α-β 过程 …… 108
4.5 复杂度问题 …… 111
4.6 结语和参考文献 …… 113
4.7 习题 …… 113
第5章 随机方法 …… 116
5.0 简介 …… 116
5.1 计数基础（选读） …… 117
5.1.1 加法和乘法规则 …… 117
5.1.2 排列与组合 …… 118
5.2 概率论基础 …… 119
5.2.1 样本空间、概率和独立性 …… 120
5.2.2 概率推理：一个道路/交通例子 …… 122
5.2.3 随机变量 …… 123
5.2.4 条件概率 …… 125
5.3 贝叶斯定理 …… 127
5.4 随机方法学的应用 …… 130
5.4.1 “tomato”是如何发音的 …… 130
5.4.2 道路/交通例子的扩展 …… 132
5.5 结语和参考文献 …… 133
5.6 习题 …… 134
第6章 为状态空间搜索建立控制算法 …… 136
6.0 简介 …… 136
6.1 基于递归的搜索（选读） …… 137
6.1.1 递归 …… 137
6.1.2 一个递归搜索的例子：模式驱动推理 …… 138
6.2 产生式系统 …… 141
6.2.1 定义和历史 …… 141
6.2.2 产生式系统的例子 …… 143
6.2.3 产生式系统中的搜索控制 …… 148
6.2.4 AI 产生式系统的优点 …… 152
6.3 用于问题求解的黑板结构 …… 153
6.4 结语和参考文献 …… 155
6.5 习题 …… 155

第三部分 捕获智能：AI 中的挑战

第7章 知识表示 …… 160
7.0 知识表示问题 …… 160
7.1 AI 表示模式的简要历史 …… 161
7.1.1 语义关联理论 …… 161
7.1.2 语义网的早期研究 …… 163
7.1.3 网络关系的标准化 …… 165
7.1.4 脚本 …… 169
7.1.5 框架 …… 172
7.2 概念图：网络语言 …… 175
7.2.1 概念图简介 …… 175
7.2.2 类型、个体和名字 …… 176
7.2.3 类型层次 …… 178
7.2.4 泛化和特化 …… 178
7.2.5 命题结点 …… 181
7.2.6 概念图和逻辑 …… 181
7.3 其他表示方法和本体 …… 182
7.3.1 Brooks 的包容结构 …… 183
7.3.2 Copycat 结构 …… 184
7.3.3 多种表示、本体和知识服务 …… 186
7.4 基于主体的和分布式的问题求解方法 …… 187
7.4.1 基于主体的定义 …… 187
7.4.2 基于主体的应用 …… 189
7.5 结语和参考文献 …… 190
7.6 习题 …… 192
第8章 求解问题的强方法 …… 195
8.0 简介 …… 195
8.1 专家系统技术概览 …… 196
8.1.1 基于规则的专家系统设计 …… 196
8.1.2 问题选择和知识工程的步骤 …… 197
8.1.3 概念模型及其在知识获取中的作用 …… 199
8.2 基于规则的专家系统 …… 201
8.2.1 产生式系统和目标驱动问题求解 …… 201
8.2.2 目标驱动推理中的解释和透明性 …… 204

8.2.3 利用产生式系统进行数据驱动推理 …… 205
8.2.4 专家系统的启发和控制 …… 207
8.3 基于模型系统、基于案例系统和混合系统 …… 209
8.3.1 基于模型推理简介 …… 209
8.3.2 基于模型推理：来自 NASA 的例子 …… 211
8.3.3 基于案例推理介绍 …… 213
8.3.4 混合设计：强方法系统的优势和不足 …… 217
8.4 规划 …… 219
8.4.1 规划简介：机器人学 …… 219
8.4.2 使用规划宏：STRIPS …… 223
8.4.3 teleo-reactive 规划 …… 226
8.4.4 规划：来自 NASA 的例子 …… 227
8.5 结语和参考文献 …… 230
8.6 习题 …… 231
第 9 章 不确定条件下的推理 …… 233
9.0 简介 …… 233
9.1 基于逻辑的反绎推理 …… 234
9.1.1 非单调推理逻辑 …… 234
9.1.2 真值维护系统 …… 237
9.1.3 基于最小模型的逻辑 …… 241
9.1.4 集合覆盖和基于逻辑的反绎 …… 242
9.2 反绎：逻辑之外的办法 …… 244
9.2.1 Stanford 确信度代数 …… 244
9.2.2 模糊集推理 …… 246
9.2.3 Dempster-Shafer 证据理论 …… 249
9.3 处理不确定性的随机方法 …… 253
9.3.1 有向图模型：贝叶斯信念网络 …… 254
9.3.2 有向图模型：d－可分 …… 255
9.3.3 有向图模型：一个推理算法 …… 256
9.3.4 有向图模型：动态贝叶斯网络 …… 258
9.3.5 马尔可夫模型：离散马尔可夫过程 …… 259
9.3.6 马尔可夫模型：变形 …… 261
9.3.7 BBN 概率建模的一阶替代方案 …… 262
9.4 结语和参考文献 …… 263
9.5 习题 …… 265

第四部分 机器学习

第 10 章 基于符号的机器学习 …… 269
10.0 简介 …… 269
10.1 基于符号学习的框架 …… 271
10.2 变形空间搜索 …… 275
10.2.1 泛化操作符和概念空间 …… 275
10.2.2 候选解排除算法 …… 276
10.2.3 LEX：启发式归纳搜索 …… 281
10.2.4 评估候选解排除算法 …… 283
10.3 ID3 决策树归纳算法 …… 284
10.3.1 自顶向下决策树归纳 …… 286
10.3.2 测试选择的信息论方法 …… 287
10.3.3 评价 ID3 …… 289
10.3.4 决策树数据问题：打包、推进 …… 290
10.4 归纳偏置和学习能力 …… 290
10.4.1 归纳偏置 …… 290
10.4.2 可学习性理论 …… 292
10.5 知识和学习 …… 293
10.5.1 Meta-DENDRAL …… 294
10.5.2 基于解释的学习 …… 295
10.5.3 EBL 和知识层学习 …… 298
10.5.4 类比推理 …… 298
10.6 无监督学习 …… 300
10.6.1 发现和无监督学习 …… 301
10.6.2 概念聚类 …… 302
10.6.3 COBWEB 和分类知识的结构 …… 303
10.7 强化学习 …… 307
10.7.1 强化学习的组成部分 …… 307
10.7.2 一个例子：九宫游戏 …… 308
10.7.3 强化学习的推理算法和应用 …… 310
10.8 结语和参考文献 …… 312
10.9 习题 …… 313
第 11 章 机器学习：连接机制 …… 315
11.0 简介 …… 315
11.1 连接网络的基础 …… 316
11.2 感知机学习 …… 318
11.2.1 感知机训练算法 …… 318
11.2.2 例子：用感知机网络进行分类 …… 319
11.2.3 通用 delta 规则 …… 322

11.3 反传学习 …… 324
11.3.1 反传算法的起源 …… 324
11.3.2 反传算法实例1：NETtalk …… 327
11.3.3 反传算法实例2：异或 …… 328
11.4 竞争学习 …… 329
11.4.1 对于分类的“胜者全拿”学习 …… 329
11.4.2 学习原型的 Kohonen 网络 …… 330
11.4.3 outstar 网络和逆传 …… 332
11.4.4 支持向量机 …… 334
11.5 Hebbian 一致性学习 …… 336
11.5.1 概述 …… 336
11.5.2 无监督 Hebbian 学习的例子 … 336
11.5.3 有监督 Hebbian 学习 …… 339
11.5.4 联想记忆和线性联想器 …… 340
11.6 吸引子网络或“记忆” …… 342
11.6.1 概述 …… 342
11.6.2 双向联想记忆 …… 343
11.6.3 BAM 处理的例子 …… 345
11.6.4 自相关记忆和 Hopfield 网络 … 347
11.7 结语和参考文献 …… 350
11.8 习题 …… 350
第12章 机器学习：遗传性和涌现性 …… 352
12.0 社会性和涌现性的学习模型 …… 352
12.1 遗传算法 …… 353
12.1.1 两个例子：CNF 可满足性问题和巡回推销员问题 …… 355
12.1.2 遗传算法的评估 …… 357
12.2 分类器系统和遗传程序设计 …… 360
12.2.1 分类器系统 …… 360
12.2.2 用遗传算子进行程序设计 …… 364
12.3 人工生命和基于社会的学习 …… 367
12.3.1 生命游戏 …… 368
12.3.2 进化规划 …… 370
12.3.3 涌现的实例研究 …… 371
12.4 结语和参考文献 …… 374
12.5 习题 …… 375
第13章 机器学习：概率理论 …… 377
13.0 学习中的随机模型和动态模型 …… 377
13.1 隐马尔可夫模型（HMM） …… 377
13.1.1 隐马尔可夫模型的介绍和定义 …… 377
13.1.2 隐马尔可夫模型的重要变形 … 379
13.1.3 使用 HMM 和 Viterbi 解码音素串 …… 382
13.2 动态贝叶斯网络和学习 …… 384
13.2.1 动态贝叶斯网络 …… 385
13.2.2 学习贝叶斯网络 …… 385
13.2.3 期望最大化：一个例子 …… 388
13.3 强化学习的随机扩展 …… 390
13.3.1 马尔可夫决策过程 …… 391
13.3.2 部分可观测的马尔可夫决策过程 …… 392
13.3.3 马尔可夫决策过程实现的例子 …… 392
13.4 结语和参考文献 …… 394
13.5 习题 …… 395

第五部分 人工智能问题求解的高级课题

第14章 自动推理 …… 398
14.0 定理证明中的弱方法 …… 398
14.1 通用问题求解器和差别表 …… 399
14.2 归结定理证明 …… 403
14.2.1 概述 …… 403
14.2.2 为归结反驳生成子句形式 …… 405
14.2.3 二元归结证明过程 …… 408
14.2.4 归结策略和简化技术 …… 412
14.2.5 从归结反驳中抽取解答 …… 415
14.3 Prolog 和自动推理 …… 417
14.3.1 概述 …… 417
14.3.2 逻辑程序设计和 Prolog …… 418
14.4 自动推理进一步的问题 …… 422
14.4.1 弱方法求解的统一表示法 …… 422
14.4.2 可选推理规则 …… 424
14.4.3 归结反驳支持下的问答机制 …… 426
14.4.4 搜索策略及其使用 …… 426
14.5 结语和参考文献 …… 427
14.6 习题 …… 427
第15章 自然语言理解 …… 429
15.0 自然语言理解问题 …… 429
15.1 解构语言：分析 …… 431
15.2 语法 …… 433
15.2.1 使用上下文无关文法说明和解析 …… 433
15.2.2 Earley 解析器：动态规划二次访问 …… 434

15.3 转移网络解析器及语义学 …… 438
15.3.1 转移网络解析器 …… 439
15.3.2 乔姆斯基层次和上下文相关文法 …… 442
15.3.3 ATN 解析器的语义 …… 444
15.3.4 结合句法和语义知识的 ATN … 447
15.4 语言理解的随机工具 …… 451
15.4.1 概述：语言分析中的统计技术 …… 451
15.4.2 马尔可夫模型方法 …… 452
15.4.3 决策树方法 …… 453
15.4.4 解析的概率方法 …… 454
15.4.5 概率上下文无关解析器…… 456
15.5 自然语言应用 …… 457
15.5.1 故事理解和问题解答 …… 457
15.5.2 数据库前端 …… 458
15.5.3 Web 信息抽取和摘要系统 …… 460
15.5.4 用学习算法来泛化抽取的信息 …… 462
15.6 结语和参考文献 …… 463
15.7 习题 …… 464

第六部分 后 记

第 16 章 人工智能是经验式的学科 … 469
16.0 简介 …… 469
16.1 人工智能：修订的定义 …… 470
16.1.1 人工智能和物理符号系统假设 …… 470
16.1.2 连接或者“神经”计算 …… 474
16.1.3 主体、涌现和智能 …… 476
16.1.4 概率模型和随机技术 …… 478
16.2 智能系统科学 …… 480
16.2.1 心理学约束 …… 480
16.2.2 认识论问题 …… 481
16.3 人工智能：当前的挑战和未来的方向 …… 487
16.4 结语和参考文献 …… 490

第一部分　人工智能的历史渊源及研究范围

用桑乔的话说，万物都必须有个开头，而且这个开头一定与过去曾存在的事物有联系。印度教徒认为大象支撑世界，但又认为大象是站在龟背上的。发明并不是要异想天开，而是要脚踏实地，必须得到社会的承认……

——玛丽·雪莱《弗兰肯斯坦》

人工智能：一个尝试性的定义

可以把人工智能（Artificial Intelligence，AI）定义为计算机科学的一个分支，它关心智能行为的自动化。本书恰好使用这样的定义，因为这个定义强调了AI是计算机科学的一部分，因而必须建立在坚实的理论基础之上并应用计算机科学领域的原理。这些原理包括用于知识表示的数据结构、应用该知识所需的算法以及用来实现算法的语言和编程技术。

然而，这个定义由于"智能"本身并没有被很好地定义和理解而受到影响。虽然大多数人确信在看到智能行为时能识别出它是智能的，但是似乎没有人能够使"智能"的定义既足够具体以评估计算机程序的智能性，同时又反映了人类意识的生动性和复杂性的特性。

这样实现一般智能这个困难的任务的结果是AI研究者通常表现为工程师的角色，即常寻找特定智能的人工制品。这些制品通常以诊断、预测或可视化工具的形式出现，能够使得人类使用者完成复杂的任务。例如，用以语言理解的马尔可夫模型，提供新数学理论的自动推理系统，通过大量皮层网络跟踪信号的动态贝叶斯网络，以及基因表达数据模式的可视化，这些例子将会在1.2节的应用中加以介绍。

因此，定义人工智能完全领域的问题就变成了定义智能本身的问题：智能是一种独立的才能，还是一系列独一无二且不相关联的能力的总称？在多大程度上可以说智能是学到的而不是预先存在的？准确地说，学习时发生了什么？什么是创造力？什么是自觉？智能是从可观察行为推断出的，还是需要特定内部机制的证据？在一个生物体的神经组织中，知识是以何种形式表示的，这对智能机器的设计有什么启示？什么是自觉，它在智能中起着什么样的作用？另外，有必要按照已知的人类智能模式来设计智能计算机程序吗？或者说，严格的"工程"方法足以解决这个问题吗？甚至有可能在计算机上实现智能吗？智能实体是不是需要具有在生物中存在的丰富感受和经历？

这是一些很难回答的问题，但这些问题帮助我们勾勒出了现代人工智能所研究的核心问题及求解方法。实际上，人工智能的魅力很大程度上就在于它提供了一种独特而又强大的工具来精确探索这些问题。AI为智能理论提供了一种媒介和实验台：首先用计算机程序语言表达出这些理论，然后在实际计算机上执行来进行测试和验证。

因此，我们前面给人工智能下的定义看上去有些含糊。但是如果将定义表述得面面俱到，很可能会产生一些更深层的问题和相互矛盾的概念，因为该领域的主要目标已经包含了它自己的定义。不过，在探索AI精确定义的过程中遇到一些困难是正常的。人工智能还是一个年轻的学科，它的结构、目标以及方法肯定不像物理这样的成熟学科定义得那么明确。

相对于定义人工智能的边界来说，我们在考虑人工智能时一向更关心如何扩展计算机科学的能力。因此，如何保证这种探索始终建立在合理的理论依据之上是AI研究者面临的一个挑战，对本书来说更是如此。

因为人工智能范围和目标的复杂性，很难给出简单定义。目前，我们将其定义为“*人工智能研究者所研究的问题和方法的集合*”。这个定义看上去好像有些可笑、没有意义，但它强调了非常重要的一点：人工智能像所有学科一样，是人类努力的成果，而且只有在这个背景下它才能被最好地理解。

任何学科，包括AI，所关心的都是某一特定的问题集，并建立一套特定的技术体系来求解这些问题，这是有很多原因的。在第1章中，我们将介绍人工智能的简短历史以及建立这一学科的学者和各种假定，并以此来解释：为什么某些问题集已经成为这一领域的主导，以及为什么本书所讨论的方法已经用来求解这些问题。

第 1 章　人工智能的历史及应用

求知是人的天性……

——亚里士多德，《形而上学》的开篇句

听完下面的话，你会更惊叹于我所创造的技术与智慧。最伟大的是：从前，人类对疾病束手无策，他们既没有药吃，也没有药喝或药膏敷；因为没有药物医治，很多人渐渐衰弱，直到我向他们传授了如何调制解痛药剂，他们才会用药物驱除各种疾病……

是我，使人类看到了天上的火焰，此前则是一片昏暗。大地之下，隐藏的财富如此之多：金、银、铜、铁——谁敢说比我先发现了它们？我确信没有人会这么说，只要他说的是真的、中肯的。总而言之，人类所拥有的一切技术都来自普罗米修斯。

——埃斯库罗斯《被缚的普罗米修斯》

1.1　从伊甸园到第一台电子计算机：对智能、知识和人类技能的态度

普罗米修斯这样评价他违反奥林匹斯山神灵所取得的战果：他不仅仅为人类盗取了火种，而且用智慧也就是心灵的钥匙——一种“理性思想”——启蒙了人类。智慧是一切人类技术乃至文明之本。古希腊剧作家埃斯库罗斯用这个神话阐述了一种对知识的非凡威力的深邃而又古朴的理解。今天，人工智能已经被应用到普罗米修斯贡献过的所有领域——医学、哲学、生物学、天文学、地理学，以及埃斯库罗斯无法预见的许多其他科学领域。

虽然普罗米修斯把人类从无知的禁锢中解放出来，但是他的行为却激怒了宙斯。由于普罗米修斯窃取了本来只属于奥林匹斯山神灵的智慧，因此宙斯下令把普罗米修斯锁在光秃秃的石头上经受永世的折磨。关于人类获取知识的努力触犯了神灵或自然法则的观点在西方思想中根深蒂固，这就是伊甸园故事的基础，也出现在但丁和米尔顿的著作中。莎士比亚和古希腊悲剧家们都把对知识的渴望描述为灾难之源。这种寻求知识最终会导致灾难的观念自产生以来一直存在，无论是文艺复兴时期、还是启蒙运动时期，甚至包括在科学和哲学大发展的 19 世纪和 20 世纪。因此，人工智能在学术领域和大众思想中引发的诸多争议根本不足为奇。

事实上，现代科技并没有使人们消除这种自古而来的恐惧——即渴望知识会导致恶果，而是使人们意识到产生那样的后果是很有可能的，甚至是即将来临的。今天，人们用科技社会的语言重新讲述着普罗米修斯、夏娃和浮士德的神话。玛丽·雪莱在她的《弗兰肯斯坦》（该书的副标题很有趣，叫《现代普罗米修斯》）一书的序言中这样写道：

> 大多是拜伦勋爵和雪莱之间的对话，而我只是一个虔诚、安静的听众。其中有一次，他们讨论了各种哲学学说，以及有关生命原理的问题，并且谈到这些原理有否可能曾被发现和讨论过。他们谈及了达尔文博士的实验（我不能确认达尔文博士是否真正做过这个实验，我只是说当时有人讲他做过这样的实验），他把一段蠕虫（vermicelli）储藏在玻璃罐中，在采取了一些特殊方法之后，它开始自发运动。难道生命不是这样形成的吗？或许死尸还可能复活；流电电流实验已经让我们看到了这样的迹象：生命体的组成部分可以被制造、组合并注入活力（Butler 1998）。

玛丽·雪莱告诉我们，诸如达尔文的进化论和发现电流这样的科学进步已经使普通民众相信：自然法则并非奥妙无穷，而是可以被系统分析和理解的。弗兰肯斯坦的魔鬼并不是“萨满教”咒语或与地狱可怕交易的产物；而是由一个个单独“制造”的部件组装起来的，并且被注

入了强大的电能。尽管19世纪的科学还不足以使人认识到理解和创造一个完全智能主体的意义，但它至少加深了这样的认识：生命和智慧的奥秘可以被纳入到科学分析中。

1.1.1　人工智能基础的简要历史

在玛丽·雪莱最终并可能必然地将普罗米修斯神话与现代科学结合在一起时，当代人工智能研究的哲学基础已经发展了几千年。虽然人工智能引发的伦理和文化方面的问题既有趣又重要，但是我们的介绍更应该关注AI的智能渊源。这一历史进程的逻辑起点应该是天才亚里士多德，但丁在《神曲》中称其为“智者先师”。亚里士多德用精细分析和严谨思维（这种分析方法和思维模式后来逐渐被更现代的科学引为标准）把早期希腊传统文化中的发现、困惑和恐惧编织在一起。

对亚里士多德来说，自然世界最奇妙的一面就是变化。他在著作《物理学》中把他的“自然哲学”定义为“对变化事物的研究”。他指出了物质（matter）与形式（form）之间的区别：雕像是用青铜材料做成人的形式。当青铜被改铸成新的形式时，变化便发生了。把物质和形式区分开来为很多现代概念奠定了哲学基础，比如符号计算和数据抽象。在计算（即使是数字计算）中，我们是在操纵电磁材料形式的模式，这些材料形式的改变代表求解过程的多个方面。把形式从它所表示的介质中抽象出来不仅使我们可以用计算方式操纵这些形式，而且还为数据结构理论——现代计算机科学的核心——打下了基础。它也支持“人工的”智能的产生。

亚里士多德在著作《形而上学》的开篇就指出“求知是人的天性”，他发展了永不改变的事物的科学，包括他的宇宙论和神学。不过和人工智能关系更密切的是亚里士多德在其著作《逻辑学》中所讨论的认识论，即人类如何“认识”世界的分析。亚里士多德把他的逻辑称为“工具”（《工具论》），因为他感到对思想本身的研究是所有知识的基础。在《逻辑学》中他发现：说某个命题为真是因为它与其他已知为真的事物相联系。因此，如果我们知道“所有人都会死”和“苏格拉底是人”，那么我们就可以推出“苏格拉底会死”。这便是亚里士多德所说的三段论的一个例子，其中使用了假言推理法的演绎推理形式。尽管推理的形式化公理形式是在2000多年后的哥特洛布·弗雷格、贝特朗·罗素、库尔特·哥德尔、艾伦（阿兰）·图灵（Alan Turing）、艾尔弗雷德·塔尔斯基等人的努力下完全发展起来的，但是其根源可以追溯到亚里士多德。

建立在希腊传统文化之上的文艺复兴思想激发人们以新的、更有力的方式来思考人类及其与自然界的关系。科学开始取代神秘主义成为理解自然的手段，时钟和最终的工厂时间表改变了成千上万城市居民的自然节奏。大多数现代社会科学和自然科学都可以在自然或人工过程中找到其根源，并可从数学上分析和理解。特别是，科学家和哲学家们意识到思想本身（即知识在人类头脑中的表示和操纵方式）是科学研究中一个艰难而又本质的课题。

现代世界观发展中最主要的事件或许就是哥白尼革命，即地球和其他行星实际上是在围绕太阳旋转的思想，它取代了古老的以地球为中心的宇宙模型。此前的数个世纪中，人们习惯了“显而易见”的秩序，在这种秩序中，对宇宙特征的科学解释与宗教教义和常识知识是吻合的，而哥白尼提出了一种完全不同的而且并非显而易见的模型来解释天体的运动。或许这是第一次发现：人类对世界的看法与世界的表象有了根本的差异。这种人类思想与其周围实体的分离，也就是对事物的看法与事物本身的分离，对现代科学中关于智慧及其组织的研究起到了关键作用。伽利略的著作扩大了哥白尼的突破，他的科学观察进一步反驳了关于世界的所谓“显而易见”的真理，他以数学作为描述世界的工具，强调了世界本身与我们对世界看法的差异。现代的意识概念正是从这一突破演化而来的：自省成为文学的一种常见主题；哲学家开始研究认识论和数

学；而且科学方法的系统应用使人们不再把感觉作为理解世界的工具。

1620 年，弗朗西斯·培根在《新工具论》（Novum Organun）中为这些新出现的科学方法学提出了一套搜索技术。基于亚里士多德学派和柏拉图哲学的思想，即实体的形式等价于其充分必要的特征的总和，培根清楚地描述了一个确定实体本质特征的算法。首先，他为实体的所有实例构造了一个有效集合，将每个实例的特征列举到一张表中。然后，他收集实体反例的一个类似列表，特别注意实体的相近实例，即那些因个别特征而背离实体形式的实例。接着培根尝试做出实体所有本质特征系统的列表，具体来说，就是那些所有正例都具有且所有反例都不具有的特征，但是这一步还不完全清楚。

有趣的是，现代人工智能算法中的变形空间搜索也用到了弗朗西斯·培根的概念学习方法的一种形式，10.2 节有相关介绍。培根的算法的一个扩展也是发现学习智能程序的一部分，相应地称为培根算法（Langley et al. 1981）。这个程序能够从与现象有关的数据集中归纳出许多物理定律。值得注意的问题是，一个通用算法的目的在于能够提供科学的证据。20 世纪早期的数学家希尔伯特挑战过这个问题（判定问题）；当代的天才艾伦·图灵也对这个问题做出了回答（他的图灵机、可计算性的证明和停机问题）；参见 Davis 等（1976）。

最早的计算机器是中国的算盘，虽然早在公元前 26 世纪就发明了，但直到 17 世纪欧洲的技术才制造出能够处理代数运算的更先进的机械。1614 年，苏格兰数学家约翰·纳皮尔（John Napier）提出了对数，即能够将乘法和指数运算分别降低为加法和乘法的转换机制。纳皮尔还创立了用来表示代数运算溢出值的框架。后来，德国数学家、图宾根的牧师——威廉·谢卡特（Wilhelm Schickard，1592—1635）——将这些框架应用于 1623 年发明的能执行加减法的计算钟，这个机器能够通过钟表铃声记录运算中的溢出。

另一个有名的计算机器是法国哲学家和数学家布莱兹·帕斯卡 1642 年制造的齿轮加法器（称为 Pascaline）。虽然谢卡特和帕斯卡的机械装置仅限于加法和减法——包括进位和借位，但它们标志了以前被认为是需要人类思维的过程能够完全自动化。正如帕斯卡后来在其著作《静思录》（Pensees　1670）中写到，“算术机器提供了关于哪种方法比动物行为更接近思维的一些印象”。

莱布尼兹（Gottfried Wilhelm von Leibniz）从帕斯卡在计算机器方面取得的成功中得到了灵感，在 1694 年完成了一个机器，后来称为莱布尼兹轮子（Leibniz Wheel）。这台机器包括一个可移动的托架和一个手动曲柄，曲柄能够驱动轮子和滚筒执行更复杂的乘除法运算。莱布尼兹还迷上了定理证明的自动逻辑推理的可能性。回到培根的实体特化算法，概念被刻画为实体必要且充分特征的集合，莱布尼兹假想了一台能够用这些特征进行计算并生成逻辑上正确的结论的机器。莱布尼兹（1887）还预想了一台机器，用这台机器可以使科学知识的产生变成全自动的，这反映了现代演绎推理和证明思想，是一种推理演算。

17 世纪和 18 世纪，涌现了大量关于认识论问题的讨论，最有影响的可能是笛卡儿的研究，他是现代思想概念和意识理论发展的核心人物。在他的著名作品《沉思》中，笛卡儿试图完全通过认知自省来寻找现实基础。在完全否认感觉输入的可信赖性后，笛卡儿甚至被迫怀疑物理世界的存在，仅保留了思想的真实性；甚至他自己的存在也要由思想来判定：“我思故我在”。在建立了他自己完全是以思考实体存在的思想后，笛卡儿推论出上帝是以不可缺少的创造者身份存在的，并且重申了物理宇宙的真实性，因为它是产生仁慈上帝的必要条件。

这里我们可以观察两个有趣的现象：第一，意识和物理世界的分离已经变得如此彻底以至于可以脱离开特定的感知输入或世界中的物质来讨论思考过程；第二，意识和物理世界的联系是如此脆弱以至于要得到支持物理世界的可靠知识竟需要仁慈上帝的干预。这种意识和物理世

界之间关系的两面性成为支撑所有笛卡儿思想的基础，包括他的解析几何学。不然的话，他怎么能把几何这样一个表面上比较物化的数学分支和代数这样一个抽象数学框架统一起来呢？

我们为什么要在关于人工智能的书中讨论包含心身的问题呢？原因很简单，以上分析的两点结论对人工智能来说是至关重要的：

1）通过把意识从物理世界中分离出来，笛卡儿和有关的思想家建立了一个重要的理论：对世界看法的结构不一定要与其对应的物质结构完全相同。这正是 AI、认识论、心理学、高等数学以及大多数现代文学等领域的方法学基础，即精神过程有其自身的存在形式；遵循其自身的法则；其本身可以被研究和学习。

2）在把意识和身体分离开后，哲学家们发现有必要找到一种方式把它们联系起来，因为精神和肉体的相互作用是人类存在的关键。

虽然已经有数百万的文字论述心身（mind-body）问题，而且已经提出了许多答案，但是没有人能成功地解释精神状态与肉体行为之间明显的相互作用，即使可以断言二者间存在根本的差异。这一问题最被广泛接受的说法是意识与身体根本不是完全不同的实体。根据这种观点，精神过程实际上是通过像大脑（或计算机）这样的物理系统来实现的。精神过程像物理过程一样最终要通过形式化的数学语言来描述。或者，正如 17 世纪英国哲学家托马斯·霍布斯（1651）在《利维坦》（Leviathan，《圣经》中描写的一种怪兽）中所承认的，“我认为逻辑推理就是计算”。

1.1.2　理性主义和经验主义学派对人工智能的影响

人工智能领域目前的研究问题，与其他科学学科一样，是在历史、社会和文化的共同影响下形成和发展的。在人工智能发展中最显著的两个影响是哲学上的经验主义和理性主义学派。

前面章节曾经提到过，早在柏拉图的著作中就支持理性主义学派，后来在帕斯卡、笛卡儿和莱布尼兹的著作中也支持理性主义学派。对理性主义者来说，外部世界可以通过清晰明确的数学思想重建。对这种二元方法的批评之一是表示系统被强制从参考领域中脱离出来。问题在于能否独立于应用条件来定义表示所具有的含义。如果世界是与我们对世界的信念不同的，那么我们创造的概念和符号是否仍然有意义？

许多人工智能程序都有很多这种理性主义的味道。例如，早期的机器人规划者将其应用领域或“世界”描述为谓词演算语句的集合，动作的“规划”能够通过证明关于“世界”的定理来生成（Fikes et al. 1972，另见 8.4 节）。在现代人工智能中，大多数人将纽维尔（Newell）和西蒙（Simon）的*物理符号系统假设*（第二部分和第 17 章有介绍）看作是这种方法的原型。几个批评家将这种理性主义偏见评价为人工智能在解决复杂任务时的部分失误，例如对人类语言的理解（Searle 1980，Winograd and Flores 1986，Brooks 1991a）。

经验主义者不认为真实世界可以清晰明确地描述，而是不断提醒我们“所有东西都是通过感官进入大脑的”。这一约束导致了后来的问题，即人类如何能够认识一般性的概念或者柏拉图洞穴的纯粹形式（Plato 1961）。亚里士多德是早期的一个经验主义者，在他的《论灵魂》中，强调了人类感知系统的局限。更多现代的经验主义者强调，知识必须通过内省而非实验心理学来解释，特别是霍布斯（Hobbes）、洛克（Locke）和休姆（Hume）。他们区分了两类精神现象，一方面是感知，另一方面是思考、记忆和想象。例如，苏格兰哲学家戴维·休姆区分了*印象*和*思想*。印象是真实的、栩栩如生的，反映了外部物体的真实存在，并不依赖于主观控制（参考Dennett (2005) 的感受性）。而另一方面思想是缺乏生动和细节的，更加依赖于主体的主观控制。

印象和思想存在不同，那么知识如何产生呢？霍布斯、洛克和休姆认为基本的解释机制是联

想。特殊的知觉属性通过重复的经验而联想到一起，这种重复联想在头脑中创造了一种关联对应思想的倾向，这一描述的基本特点用休姆的怀疑论进行了表述。休姆声称，他对思想产生根源的纯描述性的说明无法支持起因信任。甚至在这种基本的经验主义者认识论中都不能理性地支持逻辑和归纳的使用。

在《人类理解研究》（1748）中，休姆的怀疑论扩展到了对神迹的分析。虽然休姆没有直接提到神迹的本来面目，但是他质疑了神迹中基于证词的证据。这种怀疑论当然被圣经的信徒和其他宗教学派的传播者视作直接威胁。尊敬的托马斯·贝叶斯（Thomas Bayes）既是一位数学家也是一位牧师。他的一篇文章从数学上提到了休姆的问题，这篇文章是 Essay towards Solving a Problem in the Doctrine of Chances（1763）。贝叶斯定理形式地证明了如何通过学习动作效果的相互关系来确定它们发生的概率。

知识的联想说在人工智能表示的结构和程序发展中起了很大作用，例如使用语义网的记忆组织、MOPS 和自然语言理解方面的研究（见 7.0 节、7.1 节和第 15 章）。联想说对机器学习也有重要影响，特别是连接网络（见 10.6 节、10.7 节和第 11 章）。联想主义在认知心理学方面也起了很大作用，包括巴特利特（Bartlett）和皮亚杰（Piaget）的表示模式以及行为主义学派的所有延伸（Luger 1994）。最后，由于使用了随机分析的人工智能工具，包括贝叶斯信念网络（BBN）及其对随机模型一阶图灵完全系统的扩展，联想理论具有坚实的数学基础和成熟的表达能力。贝叶斯工具对诊断学、机器学习和自然语言理解都非常重要（见第 5 章和第 13 章）。

德国哲学家伊曼纽尔·康德（Immanuel Kant），受到理性主义学派的训练，被休姆的著作深深地影响。结果，他开始对这两种学派进行综合。康德认为知识包含两个协作的活力，一个是来自主体原因的先验成分，另一个是来自主动经验的后验成分。只有通过主体的作用，经验才有意义。没有主体计划的主动组织，世界将仅仅是流逝的短暂的感觉。最后，在决断的层次上，康德声称流逝的图像或表示是被主动的主体绑定在一起的，并作为同一“物体”的不同表现。康德的现实主义成为了现代心理学家的研究工作，如巴特利特、布鲁纳（Bruner）和皮亚杰。康德的工作影响了现代人工智能机器学习（第四部分）的研究以及构建主义认识论（见第 16 章）的持续发展。

1.1.3 形式逻辑的发展

一旦把思考看成是一种计算形式，那么下一步显然就是考虑它的形式甚至是机制了。在 1.1.1 节中提到，戈特弗里德·威廉·冯·莱布尼兹在他的《微积分哲学论》中介绍了第一个形式逻辑系统并建立了自动计算机器（Leibniz 1887）。这一机械性解答的工作步骤和流程能够表示为树或图上的状态的移动。莱昂哈德·欧拉在 18 世纪通过分析连接哥尼斯堡城岛屿和河岸桥梁的“连通性”（见第 3 章的“简介”部分）介绍了他对表示问题的研究，即如何抽象地获取客观世界的关系结构以及这些关系结构中计算的离散步骤（Euler 1735）。

图论的产生也为状态空间搜索（state space search）这种人工智能的重要概念工具提供了解决问题的可能性。有了这一理论，我们就可以使用图来对问题的更深层结构建模。状态空间图（state space graph）的结点表示问题解的可能阶段；图中的弧表示推理的过程、博弈中的移动或者问题解中的其他步骤。于是，求解问题就是搜索状态空间图以寻找解路径的过程（见第二部分第 3 章）。状态空间图通过描述问题解的整个空间，提供了一个强大的工具来衡量问题的结构和复杂度并分析解策略的高效性、正确性和通用性。

查尔斯·巴贝奇（Charles Babbage）是 19 世纪的数学家，他是运筹学的创始人之一，也是第一台可编程机械计算机器的设计者，同时还可以把他看成是人工智能的早期实践者（Morrison

and Morrison 1961）。巴贝奇的差分机是一种专门用来计算特定多项式函数值的机器，也是其分析机的先驱。分析机（由巴贝奇设计，但在他的有生之年没有被成功制造出来）是一种通用的可编程计算机器，它预言了现代计算机的很多结构。

巴贝奇的朋友、支持者和合作者阿达·洛夫莱斯（Ada Lovelace 1961）在描述分析机时说：

> 我们可以毫不过分地说，分析机所编织的代数图案就像杰卡德（Jacquard）的织机所编织的鲜花和绿叶一样。在我们看来，这里蕴涵了比差分机多得多的创造性。

巴贝奇的动机是想应用当时的科技把人们从繁重的数学计算中解放出来。在这种热情鼓舞下，凭借着他从机械设备角度建立的计算机观念，巴贝奇完全以19世纪的术语构思着计算机器的蓝图。但是，他的分析机也包含了很多现代思想，比如存储器和处理器的分离（巴贝奇分别将它们称为仓库（store）和工厂（mill）），数字机器而不是模拟机器的概念，以及基于执行一系列编码在穿孔卡片上的操作的可编程能力。阿达·洛夫莱斯的描述以及巴贝奇的研究的最惊人特征是它们把代数关系“模式”看成是可以被研究、刻画并最终实现的实体，而且可以在不关心特定值的情况下机械地操作这些实体，具体值最终可以通过计算机器的“工厂”传入。这实现了最初由亚里士多德和莱布尼兹描述的“形式的抽象和操纵”。

创建思考的形式语言这一目标也出现在乔治·布尔（George Boole）的著作中，他是另一位19世纪的数学家，要讨论人工智能的根源不能不提到他的研究（Boole 1847，1854）。虽然他对很多数学领域都做出了贡献，但是他最著名的研究是对逻辑定律的数学形式化，这一成就形成了现代计算机科学的核心。虽然布尔代数在逻辑电路的设计中非常著名，但是布尔自己的目标是开发一个系统，这个系统似乎更接近于现代AI学者的研究方向。在《思维规律的研究》（逻辑和概率数学理论是在此基础上研究的）的第1章，Boole（1854）把他的目标描述为：

> 为了研究关于人类推理的思维运动的基本定律：用微积分符号语言来表示它们，并在这个基础上建立逻辑科学和指导它的方法；……最终目的是从这些科学研究过程中看到的各种不同要素里搜集到一些和人类思维的本质与构造有关的一些可能线索。

布尔成就的重要性在于他所设计的系统具有非常的能力和简洁性：逻辑计算的核心只包含3种运算，即“与”（表示为*或∧）、“或”（表示为+或∨）和“非”（表示为¬），$X*X=X$（也就是，一旦已经知道某件事为真，那么重复不能增大这一信息）。这样便把布尔值限制到满足这一等式的仅有的两个数字：1和0。布尔乘法（与）和加法（或）的标准定义就是依据这一定律给出的。

布尔系统不仅为二进制算术运算提供了基础，而且说明了非常简单的形式系统也具有逻辑的全部能力。这个假定以及布尔为了说明它而开发的系统奠定了所有现代形式化逻辑研究的基础，从罗素和怀海德的《数学原理》（Whitehead and Russell 1950），到图灵和哥德尔的研究，也包括现代的自动推理系统。

戈特洛布·弗雷格（Gottlob Frege）在他的《算术基础》（Frege 1879，1884）中创建了一种数学说明语言，目的是以一种明了而且精确的方式描述算术的基本概念。利用这种语言，弗雷格形式化了亚里士多德在其《逻辑学》中最先提出的很多问题。今天人们把弗雷格的语言称为一阶谓词演算，它不仅提供了一种工具来描述命题和真值指派这样的数学推理要素，而且为这些表达式的“意义”建立了公理基础。谓词演算形式系统（包括谓词符号、函数理论和变量）的目的是成为一种描述数学及其哲学基础的语言，但是它在创建人工智能的表示理论中也起到了非常重要的作用（见第2章）。一阶谓词演算为自动推理提供了必要工具：为建立表达式提供了语言；为表达式的意义提供了理论依据；为推断新的真表达式提供了逻辑合理的演算规则。

罗素和怀海德的著作（Russell and Whitehead 1950）对奠定 AI 的基础起到了非常重要的作用，因为他们预定的目标是通过对一系列公理的形式运算导出数学的全部内容。尽管已经从基本定理建立起了很多数学系统，但不同的是罗素和怀海德从纯形式系统的角度来研究。这意味着公理和定理会完全被当成字符串，证明的过程就是应用完善定义的规则来操纵这些字符串。这样证明的基础就不再依赖于直觉或定理的意义。证明的所有步骤都是严格地把形式（语法）规则应用到公理或者以前证明的定理，即使对于那些被当成是“显而易见”步骤的传统证明也是如此。系统的定理和公理对世界的含义独立于它们的逻辑推导过程。这种纯粹从形式角度（因而也就可以从机械角度）处理数学推理的方法为数学推理在物理计算机上的自动化奠定了关键基础。罗素和怀海德开发的逻辑语法和形式推理规则至今仍然是自动定理证明系统（在第 14 章介绍）的基础，同时也是人工智能的理论基础。

艾尔弗雷德·塔斯基（Alfred Tarski）是另一位为人工智能奠定关键基础的数学家。塔斯基创立了*指称理论*（theory of reference），可以说该理论使用了弗雷格或罗素和怀海德的*合式公式*（well-formed formulae），以精确的方式指称物理世界（Tarski 1944，1956；见第 2 章）。这一成果成为大多数形式语义理论的基础。在他的论文“The Semantic Concept of Truth and the Foundation of Semantics”（真理的语义概念和语义基础）中，塔斯基描述了他的指称理论和真值关系。一些现代计算机科学家，尤其是 Scott、Strachey、Burstall（Burstall and Darlington 1977）以及 Plotkin，已经把这一理论和程序设计语言以及计算的其他规范联系起来。

虽然在 18、19 以及 20 世纪的初期，科学和数学的形式化就为人工智能的研究创造了智能方面的必备条件，但是直到 20 世纪数字计算机被引入时 AI 才成为一门充满活力的科学学科。在 20 世纪 40 年代末期，电子数字计算机显示出了智能程序所需的存储和处理能力，在计算机上实现形式推理系统并通过试验测试它们是否足以显示出智能成为了可能。人工智能科学的一个关键特征就是选择数字计算机作为创建和检验智能理论的工具。

数字计算机不仅是一个检验智能理论的工具，计算机的体系结构还为智能理论提出了一种特殊的模式：智能是信息处理的一种形式。例如，把搜索作为求解问题方法的概念应更多地归功于计算机操作的序列化特征，而不是任何生物智能模型。大多数 AI 程序用某种形式语言来表示知识，然后再用算法来操纵这些表示，这种数据和程序的分离是冯·诺依曼（von Neumann）计算方式的基础。形式逻辑已经逐渐成为 AI 研究的一种重要表示工具，就像图论在问题空间分析中所起的不可缺少的作用以及为语义网和相似语义表示模型提供了基础一样。本书的正文部分详细讨论了这些技术和形式，我们在这里提到这些内容只是为了强调数字计算机与人工智能的理论基础间的共生关系。

我们经常忘记：我们为自己设计的工具往往会以它们的结构和局限影响我们关于世界的概念。虽然这有明显的局限性，但是这种相互作用也是人类知识发展的一个关键因素：为了求解某个问题，人们就会开发一种工具（科学理论最终都只是工具），随着应用和改进这种工具，人们会从工具本身想到其他的应用，这便产生了新的问题，而且最终会导致新工具的开发。

1.1.4 图灵测试

最早一篇专门论述机器智能与现代数字计算机关系问题的论文是由英国数学家艾伦·图灵于 1950 年发表的。《计算机器和智能》（Turing 1950）这篇文章至今仍值得一读，无论是它对制造智能计算机器可能性争论的评价，还是它对这些争论的回答，都有着现实的意义。图灵——主要因为他对可计算性理论所作出的贡献而著名——苦苦考虑着是否能够制造出真正可以思考的机器。考虑到这个问题本身存在的严重模糊性（什么是思考？什么是机器？）阻碍了理性的思

考，于是他提出用一个定义更明确的实验测试来描述这一智能问题。

图灵测试用人类的表现来衡量假设的智能机器的表现，这无疑是评价智能行为的最好的且惟一的标准。这个被图灵称为模仿游戏的测试是这样进行的：将一个人与一台机器分别置于一个房间中，而另一个称为询问者的人与它们分隔（如图1-1所示）。询问者看不到屋中任意一方，也不能直接与他们说话，因此他不知道到底哪一个房间内的是机器，只可以通过一个类似终端的文本设备与他们联系。然后让询问者仅根据收到的答案辨别出哪个是计算机，哪个是人。如果询问者不能区别出机器和人，那么根据图灵的理论，就可以认为这个机器是智能的。

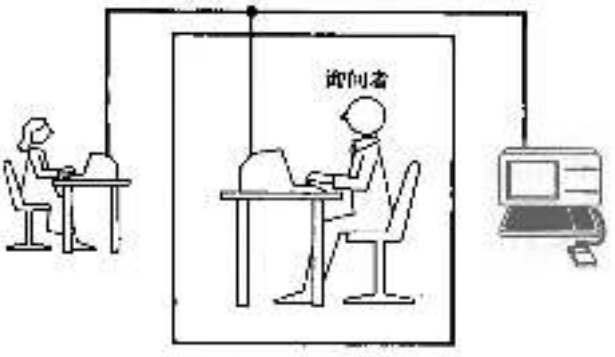


图1-1　图灵测试

把询问者与机器和另一个人类参与者分开，这样做可以使图灵测试的询问者不会因为机器的外观或者机器声音的机械特性而产生偏见。不过，询问者可以自由询问任何问题，不管如何拐弯抹角，目的是尽力揭示计算机的特性和身份。比如，询问者可以给他们出一道十分复杂的算术计算，并假定计算机比人更可能计算出正确结果；与这条战略相反，计算机要知道它什么时候算不出正确答案，以使它看上去更像人。为了根据感情特征来识别出人类的身份，询问者可以向双方询问他们对于一首诗或是一件艺术品的反应；这一策略要求计算机具有人类情感特性方面的知识。

图灵测试的重要特征有：

1）它给出了一个客观的智能概念，也就是根据对一系列特定问题的反应来确定是否是智能体的行为。这为判断智能提供了一个标准，从而避免了有关智能“真正”特征的经常性争论。

2）这项实验使我们免于受到诸如以下目前无法回答的问题的牵制：计算机使用的内部处理方法是否恰当或者机器是否真的意识到其动作。

3）通过使询问者只关注问题回答的内容消除了有关他所具有的生命体方面的偏好。

由于有这些优点，图灵测试为许多实际用于现代AI程序评价的方案提供了基础。如果一个程序已经有可能在某个专业领域实现了智能，那么可以通过把它对一系列给定问题的反应与人类专家的反应相比较来对其进行评估。这种评估技术仅是图灵测试的一个变体：请一些人对计算机和人类专家对一组特定问题的反应结果作出封闭式的比较。正如我们所看到的，这种方法已经成为现代专家系统开发和验证的关键工具。

尽管图灵测试具有直观上的吸引力，图灵测试还是受到了很多无可非议的批评。其中一个最重要的质疑是它倾向于纯粹的符号问题求解任务。它并不测试感知技能或要实现手工灵活性所需的能力，而这些都是人类智能非常重要的部分。另一方面，有人提出图灵测试没有必要把机器智能强行套入人类智能的模具之中。或许机器智能就是不同于人类智能，试图按照人类的方式来评估它，可能根本上就是一个错误。我们真的希望一台机器做起数学题来像人一样又慢又不准吗？难道不应该让智能机器发挥它自身的优点（比如庞大、快速、可靠的存储器），而宁愿去让它模仿人类的认知特征吗？实际上，很多现代AI实践者（例如Ford and Hayes 1995）对图灵测试提出了全面的质疑，认为图灵测试是一个错误，分散了我们的注意力，妨碍了目前更重要的工作：研究通用的理论来解释人类智能和机器智能的机制，并应用这些理论来开发可以解决具体实践问题的工具。尽管我们大体上同意Ford和Hayes的观点，但是仍然认为图灵测试是将

测和证实现代 AI 软件智能性的一个重要部分。

图灵也论述了在数字计算机上建立智能程序的切实可行性。他根据一个具体的计算模型（电子离散状态计算机器）进行思考，对以下方面做出了周密的推测：存储容量、程序复杂度以及这类系统所需的基本设计哲学。最后，他从实际技术的角度论述了建立这样的智能程序在道德、哲学及科学方面可能受到的很多反对意见。推荐读者自己阅读图灵的文章，以领略他对智能机器可能性争论的透彻切题的概述。

图灵在其论文中提到的两个反对观点也值得我们深刻思考。第一个是阿达·洛夫莱斯最先提出的洛夫莱斯伯爵夫人的异议（Lady Lovelace's Objection），该观点认为计算机只能按照它被告知的方式去做而不能完成创造性的（也就是智能的）行为。图灵对这一观点的反对意见已经成为一种结论，它打消了人们对当今技术神话中某些含糊部分的怀疑。专家系统（见第 1.2.3 节和第 8 章）已经得到了其设计者们没有预料到的结论，尤其是在诊断推理领域。实际上，许多学者认为人类的创造力是可以用计算机程序来表达的。

另一个相关的反对观点是行为的非形式化性理论，这个观点断言：不可能创建出这样一套规则，它可以准确地告诉一个个体在每一种可能情况下要做什么。生物智能具有灵活性，可以对几乎无限范围的情况做出最佳或者合理的反应，毫无疑问，这种灵活性是智能行为的一个重要特征。不过，虽然大多数传统计算机程序使用的控制结构并没有显示出强大的灵活性和创造性，但是这并不是说所有的程序都必须按照这种模式来编写。实际上，在过去 25 年的 AI 发展中，人们已经做了大量工作来开发程序设计语言和模型，比如产生式系统、基于对象系统、神经网络表示以及本书讨论的试图弥补这种不足的其他方法。

许多现代 AI 程序通常由一系列模块化的组件或者行为规则构成，它们不是按照严格顺序执行的，而是根据处理特定问题实例的需要而被调用。模式匹配允许把通用规则应用到某一范围内的实例，这样的系统具有极强的灵活性，相当小的程序就可以展示出很广范围内的可能行为，可以对不同的问题和情况作出反应。

这些系统是否能够最终被制造出来，以展示出生物体才能表现出的那种灵活性至今仍然是一个很有争议的课题。诺贝尔奖获得者赫伯特·西蒙（Herbert Simon）认为，生物体行为所显示的创造性和变化性大多应该归功于生物体所处环境的丰富性而不是它们自身内部程序的复杂性。在《人工科学》（Simon 1981）中，西蒙描述了一只蚂蚁沿着崎岖而又混乱的道路行走。尽管蚂蚁走过的路径看上去非常复杂，但是西蒙认为蚂蚁的目标非常简单：那就是尽可能快地返回到它的群落。蚂蚁路径上的盘旋和转弯是由它在路上遇到的障碍所导致的。西蒙这样总结道：

> 如果把蚂蚁看成是一个行为系统，那么它是非常简单的。其行为的表面复杂性很大程度上是其所处环境复杂性的反映。

如果这种思想最终被证明不仅适用于像昆虫这样的简单生物体，还适用于高等智能生物体，那么它就有力地证明了这样的系统是相对简单并且易于理解的。非常有趣的是，如果将这种思想应用于人类，那么它就成为证明文化对智能形成具有重要性的一个强大论据。智能并不是像蘑菇那样生长在黑暗之中，它似乎离不开与所处的丰富环境相互作用。文化在创造人类方面的重要性与人类创造文化是同等重要的。这种思想并不是要贬低我们的智慧，而是强调了为什么从彼此分隔开的人类生活中产生的文化具有不可思议的丰富性和一致性。实际上，智能是从社会个体元素间的相互作用中产生的，这一思想是支持下一节将要介绍的 AI 技术的论据之一。

1.1.5 智能的生物和社会模型：主体理论

到现在为止，我们已经从数学角度阐述了建立智能机器的问题，其中蕴涵了把逻辑推理作

为智能本身范例的观点，并带有对逻辑推理“客观”基础的肯定。这种看待知识、语言和思想的方式反映了西方哲学的理性传统，这种传统是从柏拉图、伽利略、笛卡儿、莱布尼兹以及本章前面讨论过的其他哲学家那里演化而来的。它也反映了图灵测试所依赖的假定，尤其是图灵测试对符号推理在智能测试中地位的强调，以及与人类行为进行直接比较就足以确认机器智能的思想。

这种把逻辑作为表示知识方式的依赖性，以及把逻辑推理作为智能推理首要机制的依赖性，在西方哲学中占有统治地位，以至于它们的“正确性”经常是显而易见和不容置疑的。因此也就不奇怪为什么基于这些假定的方法自始至终一直统治着人工智能科学。

不过，在20世纪的后半段已经出现了很多对理性哲学的挑战。各种形式的哲学相对论对语言、科学、社会和思想本身的客观基础提出了质疑。路德维格·维特根斯坦（Ludwig wittgenstein）的后期哲学（Wittgenstein 1953）已经迫使我们重新考虑自然语言和形式语言的语义基础。哥德尔（Nagel and Newman 1958）和图灵的著作已经对数学基础本身产生怀疑。后现代思想已经改变了我们对艺术和社会的价值及意义的理解。人工智能一直以来都屡受非议；事实上，AI在实现其目标的过程中所碰到的困难经常被用于说明理性观点失败的证据（Winograd and Flores 1986，Lakoff and Johnson 1999，Dennett 2005）。

两种传统哲学，即维特根斯坦（Wittgenstein 1953）以及胡塞尔（Husserl 1970，1972）和海德格尔（Heidegger 1962）的哲学思想，对重新评估西方传统哲学起到了核心作用。在维特根斯坦的后期著作中，对理性传统假定（包括语言、科学和知识本身的基础）提出了很多质疑。自然语言是维特根斯坦分析的一个焦点：他对人类语言的语义来源于某种客观基础的思想提出了挑战。

对于维特根斯坦以及奥斯汀（Austin 1962）及其追随者格赖斯和瑟尔（Grice 1975，Searle 1969）创立的言语行为理论来说，任何语言的含义都依赖于它所处的人文和文化背景。举例来说，我们对“椅子”这个词含义的理解依赖于它具有一种与坐姿一致的物理结构以及人类使用椅子的文化习惯。例如，大且平坦的石头何时是椅子？为什么把国王的宝座称为椅子听起来很奇怪？人类对椅子的理解与对猫或狗的理解有什么不同，是因为从人类的角度考虑不能坐在猫狗身上吗？维特根斯坦基于他对语义基础的抨击，认为我们应该在一个运动的文化背景下根据具体的选择和行为来观察语言的用法。维特根斯坦甚至把他的批评延伸到科学领域和数学领域，认为从社会概念来看它们和语言使用相差不多。

胡塞尔（Husserl 1970，1972）——现象学之父，坚信抽象应该源于具体的*生活世界*：理性模型相对于支撑它的具体世界来说完全是第二位的。胡塞尔和他的学生海德格尔（Heidegger 1962）以及他们的支持者Merleau-Ponty（1962）认为，智能不是知道什么是正确的，而是知道如何应对不断变化和发展的世界，伽达默尔（Gadamer 1976）对这一学派也有贡献。因此，对于存在主义者/现象学家来说，智能被看成是在世界中的生存能力，而不是关于世界的一系列逻辑命题（结合了某一种推理模式）。

很多学者，例如德莱弗斯（Dreyfus 1985）以及威诺格拉德和弗洛里斯（Winograd and Flores 1986），他们在对AI的批评中吸收了维特根斯坦以及胡塞尔/海德格尔的思想。虽然许多AI实践者还在继续发展和完善理性/逻辑方法，又被称为优秀的老式人工智能（Good Old Fashioned AI，GOFAI），但是已经有越来越多的学者参与到这些批评之中，以探索新的智能模型。在符合维特根斯坦强调知识的人文根源的基础上，他们已经转向智能行为的社会模型（*有时称为基于主体或情景的模型*）以寻找灵感。

作为与基于逻辑的方法相对的一个实例，连接学习中的研究（见1.2.9节和第11章）不再

强调逻辑，以及理性思维的功能，并努力通过对人类大脑体系结构建模来实现智能。智能的神经模型强调大脑通过调整神经元个体间的关系来适应其所处世界的能力。而且不再用显式的逻辑语句来表示知识，而是用关系模式的属性隐性地获取知识。

另一种基于生物的智能模型是从整个生物群体适应其周围环境的过程汲取灵感的。人工生命和遗传算法的研究（见第12章）应用生物进化原理来寻找某些困难问题的解。这些程序并不通过逻辑推理来求解问题，而是产生出大量候选解，组成一个竞争群体，并模仿生物进化的模式使它们不断进化为更好的解，同时很差的候选解往往会逐渐消失，那些有希望解决问题的解会生存下来，将它们的成功父辈组合起来再生新解。

社会系统阐释了智能的又一内涵：全局行为可以求解任何个体无法解决的问题。举例来说，虽然没有哪个人可以准确地预测出纽约这座城市某一天要吃掉多少块面包，但是由纽约面包店组成的整个系统出色地完成了保证这座城市的面包供应的任务，而且使浪费量最少。股票市场很好地做了一件工作：设定数千家公司的相对值，而每个投资个体仅知道几家公司的有限信息。最后一个例子来自现代科学。处于大学、工厂或政府等不同环境中的个体关注一些普遍问题。通过以会议或杂志作为主要的通信媒介，这些半独立工作的个体解决了许多对全社会都很重要的问题（尽管很多情况下取得的进步也是由一些基金组织来推动的）。

这些例子都有两个特征：第一，认为智能来源于文化和社会，因此是自然发生的。第二，智能是由大量非常简单、相互影响的半自动个体组成的集体行为来反映的，这些个体被称为主体。不论我们把主体看成什么：神经细胞、物种的单个成员或者社会中的单个人，智能都是通过主体的相互作用产生的。

关于智能的面向主体和自然发生的观点包含了如下要点：

1）主体是自动的或半自动的。也就是说，每个主体在问题求解中具有特定的职责，它对其他主体在做什么以及如何做都知之甚少或者根本就不知道。每个主体处理它自己的、独立的问题片段，要么自己产生结果（执行某动作），要么把结果报告给团体（通信主体）中的其他主体。

2）主体是“基于情景的”。每个主体只对其自身周围的环境做出反应，（通常）不具有任何关于所有主体组成的整个域的知识。因此，主体的知识只限于和要处理的任务有关的信息：“我在处理的文件”或“靠近我的墙壁”，没有关于所有文件或问题求解任务中全部物理约束的任何全局性知识。

3）各个主体是相互影响的。也就是说，它们组成了一个集体来共同完成特定的任务。从这个意义上说，可以把它们看成是一个“社会”，而且和人类社会一样，当把它们看成是集体时，知识、技能和职责可以分布到每个个体上。

4）构成一个主体社会。在大多数面向主体的问题求解方法中，虽然每个主体具有自己的独特环境和技能，但是在整个问题求解中各个主体是相互合作的。因此，最终解是通过组合和合作得到的。

5）最后，在这种环境中智能现象是“自然发生的”。虽然单个主体具有自己的一组技能和职责，但是主体社会总的合作成果要大于单个个体贡献的总和。智能是存在于社会并从社会中浮现出来的一种现象，而不是单个主体的属性。

基于这些思想，我们把主体定义为可以感知其环境特征（经常仅仅局限于此）并直接或通过与其他主体的合作来影响环境的群体元素。通常，求解一个智能问题需要很多种主体。包括仅获取并交流信息片段的机械主体；支持与其他主体交互的协调主体；可以分析多种信息片段并返回选取的一小部分信息的搜索主体；可以分析一系列信息并形成概念或进行泛化的学习主体；

以及既可以分配任务又可以根据有限的信息和处理能力来得出结论的决策主体。如果回到一个关于智能的较老的定义，那么可以把主体看成是一种支持在有限处理资源情况下做出决策的机制。

设计和建立这样的主体社会需要的主要条件是：

1）用来表示信息的结构。

2）用来搜索备选解的策略。

3）创建可以支持主体交互的体系结构。

本书的后续章节，特别是第7.4节，介绍了构建这种主体社会的支持工具和方案，以及许多基于主体的问题求解的例子。

以上对自动智能理论可能性的初步讨论决不是企图夸大今天已取得的进步，也没有竭力缩小我们面前工作的难度。正如本书通篇所强调的，认识到我们的不足并诚实地看待我们的成功是很重要的。举例来说，在机器学习方面仅取得了很有限的成果。在对自然语言（比如英语）的语义复杂度建模方面我们的成就也非常有限。我们还需要更深入地研究一些非常根本的问题，比如组织知识以及完全管理异常庞大的计算机程序（例如庞大的知识库）的复杂度和正确性。基于知识的系统，尽管已经工程化并进入市场，但是在推理质量和通用性方面还存在很多不足。包括无法进行常识推理，也就是不能运用初步的自然知识，比如事物如何随时间变化。

不过我们必须保持一种合理的态度。当我们面对仍然要做的工作时很容易忽略已经取得的成就。在下一节，我们将通过综述人工智能研究和发展的许多重要领域来给出这个领域的远景。

1.2　人工智能应用领域概述

分析机不能自主创新，它只能做我们已经知道的如何去执行的事。

——阿达·拜伦，洛夫莱斯伯爵夫人

对不起，戴夫，我不能让你那样做。

——阿瑟·克拉克的《2001太空漫游》中的HAL 9000

现在我们回到定义人工智能的话题，并把视角转向这一领域工作者的梦想和成就。AI研究者们所关心的两个最基本的问题是*知识表示*（knowledge representation）和*搜索*（search）。前者所针对的是如何以一种形式化的（适合计算机操纵的）语言来表征智能行为所必需的全部知识。第2章介绍了谓词演算语言，可以用这种语言来描述问题域中对象间的属性和关系，这种问题需要做出定性推理而不是算术计算。后面第三部分讨论了人工智能已经开发出来的用于表示区域模糊性和复杂情况的工具，这些工具是针对常识推理和自然语言理解这样的区域开发的。

搜索是一种系统探索问题状态空间的问题求解技术，*问题状态*（problem state）就是问题求解过程中的各种连续且可选的步骤。例如博弈中的不同棋局或推理过程的各种中间步骤。我们可以搜索这些备择解所组成的空间以找到最终的答案。Newell and Simon（1976）认为这也是人类求解问题的关键基础。例如，当棋手分析不同走法的效果或医生考虑很多个备择诊断时，他们确实是在搜索各种备择方案。第3、4、6和16章讨论了这种模型的内涵以及实现技术。本书的补充材料中提供了用LISP、Prolog和Java对这些算法的实现。

和大多数科学一样，我们也可以把AI分解为很多个学科的分支，这些分支在求解问题的核心方法方面是相同的，但各自又致力于不同的应用。本节介绍几个主要的应用领域以及它们对AI整体的贡献。

1.2.1　博弈

状态空间搜索的大多数早期研究都是针对常见的棋盘游戏来实现的，比如西洋跳棋、国际象棋以及 15 格拼图游戏（15-puzzle）等。除了明显的智能性外，棋盘游戏还有很多属性使其成为研究的理想对象。大多数游戏都有定义好的竞技规则：这样便可以很容易地产生搜索空间，使研究者摆脱那些由于没有固定结构问题而产生的模糊性和复杂性。博弈中的棋局易于在计算机中表示，根本不需要表征更复杂问题所必需的复杂格式。博弈的简单性使测试博弈程序没有任何经济和道德上的负担。第 3 章和第 4 章介绍了状态空间搜索，这是大多数博弈研究的基础。

博弈过程可能产生惊人庞大的搜索空间。要搜索这些庞大而且复杂的空间需要使用强大的技术来判断备择状态，探索问题空间。这些技术被称为启发（heuristic），而且成为 AI 研究的一个主要领域。启发是一种很有用但是可能出错的问题求解策略，例如，在认为一个无反应的器具损坏前先检查一下它的插头是否插好；或者在下国际象棋时为了保护王不被抓而出车。我们通常称其为智能的大多数行为似乎都属于人类用来求解问题的启发。

因为我们大多数人都有玩这些简单游戏的经历，所以我们可以很容易地设计出我们自己的启发并测试其有效性。若是求解某个深奥领域（比如医学和数学领域）的问题就必须寻找并咨询该领域的专家（国际象棋明显是个例外）。由于这些原因，博弈为启发式搜索的研究提供了广阔的空间。第 4 章介绍了针对这些简单博弈的启发式。尽管博弈程序具有简洁性，但本身也具有一些挑战性问题，比如不能确定性地预测出对手的走法（见第 5 章和第 9 章），必须考虑博弈策略中的心理和战术因素。

1.2.2　自动推理和定理证明

可以说自动定理证明是人工智能的最古老分支，其根源可以从纽维尔和西蒙的“逻辑理论家”（Newell and Simon 1963a）以及“通用问题求解器”（Newell and Simon 1963b），追溯到罗素和怀海德关于“可以把数学看成是从基本公理推导出定理的纯形式化推导”的努力，一直到最初巴贝奇和莱布尼兹的著作。无论如何，它都理所当然是人工智能领域中最硕果累累的分支之一。定理－证明研究肩负了 AI 早期研究中的很多任务，包括形式化搜索算法以及开发形式化的表示语言，例如谓词演算（见第 2 章）和逻辑程序设计语言 Prolog。

自动定理证明的吸引力主要在于逻辑的严谨性和一般性。因为它是一个形式化系统，所以是逻辑本身适合于自动化。这种系统可以处理广泛范围内的问题，只要把问题描述和相关背景信息表示为逻辑公理，把问题的实例表示为要证明的定理。这就是自动定理证明和数学推理系统（见第 14 章）的基础。

不幸的是，编写定理证明程序的很多早期努力都无法开发出一个能够求解各种复杂问题的系统。这是因为任何具有一定复杂度的逻辑系统都不能产生无限数量的可证明定理：缺乏有效的技术（启发）来引导搜索，自动定理证明程序在找到正确解之前要证明数量非常庞大的无关定理。为了克服这种低效性，很多人认为引导搜索的纯形式化的、依据语法的方法在处理如此庞大的空间时存在固有的缺陷，惟一的解决办法是依赖人类在求解问题时使用的非形式化的专门策略。这就是开发专家系统的基本思想（见第 8 章），而且事实证明这也是行之有效的。

不过，基于形式化数学逻辑的推理具有太强的吸引力，以至于难以视而不见。很多重要的问题，比如设计和验证逻辑电路、验证计算机程序的正确性以及复杂系统的控制等仍与这种方法相关。此外，定理证明研究人员已经通过设计有效的启发式算法取得了成功，这些启发式算法主要依赖于评估逻辑表达式的语法形式，从而降低了搜索空间的复杂度，不必求助于大多数人类

问题求解器所使用的专门技术。

对自动定理证明程序保持浓厚兴趣的另一个原因是：这样的系统不一定要在离开人类帮助的情况下独立求解非常复杂的问题。很多现代的定理证明程序往往是充当智能助手，让人类完成要求更高的任务，人类把一个大的问题分解为多个子问题，并设计出搜索可能解空间的启发式方法，然后让定理证明程序完成比较简单但仍需它来完成的任务，比如证明引理、验证较小的推测以及完成人类所给出思路的形式化证明（Boyer and Moore 1979，Bundy 1988，Veroff 1997，Veroff and Spinks 2006）。

1.2.3　专家系统

从早期问题求解研究中得到的一个主要启示就是对特定领域知识的重视。例如，某个医生的诊断很准确并不是因为他与生俱来就掌握某种通用的问题求解技巧，而是因为他对医学非常了解。类似地，地质学家善于发现矿藏是因为他能够将大量的理论和实践知识应用于现有的问题。专家知识融合了对问题的理论理解以及大量被经验所证实的启发式问题求解规则。专家系统就是从人类专家那里获取这些知识，然后将其进行形式化编码，使计算机可以应用这些知识来求解类似的问题。

依赖人类领域专家知识来建立系统的问题求解策略是专家系统的一个主要特征。虽然某些程序的设计者也是问题域知识的来源，但是更典型的情况是这些程序来自于问题域专家（比如医生、化学家、地质学家或工程师）与人工智能专家的合作。领域专家提供问题域中的必要知识，他可以通过对其问题求解方法的一般介绍或者以仔细选择的样例来展示他的技巧。AI 专家，或者按专家系统设计者的叫法称其为*知识工程师*的专业人员的任务是用程序实现知识，程序不仅要高效，而且其行为又要具有明显的智能性。程序写出之后，必须通过求解样例问题来精炼其智能水平，让领域专家来评判它的行为，并对程序的知识做出必要的修改和补充。反复重复这个过程直到这个程序满足预定的性能要求。

利用特定领域知识求解问题的最早系统之一是 DENDRAL，它是在 20 世纪 60 年代后期在斯坦福开发的（Lindsay et al. 1980）。人们设计 DENDRAL 是用来根据化学分子式和有关分子中化学键的大量光谱信息来推断有机分子的结构。因为有机分子的数量往往非常庞大，所以这些分子可能结构的数量往往也很大。DENDRAL 通过将化学专家的启发性知识应用到构造解释问题来处理这个搜索空间非常庞大的问题。DENDRAL 的方法被证实是非常有效的，只要试几次就可以从数以百万的可能结构中找到正确的结构。这种方法如此成功，以致于该系统的后继版本和扩展版本被应用到全世界的化学和药学实验室中。

虽然 DENDRAL 是使用特定领域知识达到专家级问题求解水平的最早程序之一，但是 MYCIN 奠定了当代专家系统方法的基础（Buchanan and Shortliffe 1984）。MYCIN 使用专业的医学知识来诊断脊髓脑膜炎和血液传染病，并开具治疗处方。MYCIN 是在 20 世纪 70 年代中期在斯坦福开发的，它是使用不确定性或不完整信息进行推理的最早程序之一。MYCIN 可以利用适合于特定问题域的控制结构为它的推理提供清楚的逻辑解释，并能可靠地评估其性能的标准。目前正在使用的很多专家系统开发技术正是在 MYCIN 项目中被首次研制的（见第 8 章）。

其他经典的专家系统包括 PROSPECTOR 程序，它可以根据一个地点的地理信息判断可能位置和矿床类型（Duda et al. 1979a，1979b）；INTERNIST 程序，它可进行内科诊断；Dipmeter Advisor，它可以用来解释油井钻探记录的结果（Smith and Baker 1983）；以及配置 VAX 计算机的 XCON。XCON 是在 1981 年开发的，而且曾经有一段时间 DEC 公司所卖的每台 VAX 都是由这个软件来配置的。大量的其他专家系统解决了很多领域的问题，比如医学、教育、商务、设计和科

学研究（Waterman 1986，Durkin 1994）。也可以查阅人工智能创新应用（Innovative Applications of Artificial Intelligence，IAAI）会议的当前论文集。

值得注意的一个有趣现象是：已经开发出的大多数专家系统都是为比较专业的、专家级的领域设计的。这些领域通常已经有了很深入的研究并明确定义了问题求解策略。使用这种方法来求解那些依赖“常识”来定义的比较松散的问题要困难得多。虽然专家系统有着广阔的前景，但是过高地估计其能力也是错误的。目前，专家系统有以下几点不足：

1）难以表征问题域的深层知识。例如，MYCIN 缺乏人类生理系统的真正知识。它并不知道血液是用来干什么的以及脊髓的作用是什么。听说曾经发生过这样一件事，当为一个脑膜炎患者选择治疗药物时，MYCIN 询问患者是否怀孕了，尽管这个患者已经告诉 MYCIN 他是男性了，但还是出现了这样啼笑皆非的情况。不管这个故事是否真实，它确实说明了专家系统中知识存在潜在的狭义。

2）缺乏鲁棒性和灵活性。如果人类遇到一个不能立即解决的问题，那么通常会回过头来分析基本的原理并想出某种策略来攻克这个问题。但是专家系统通常没有这个能力。

3）不能提供深入的解释。因为专家系统缺乏问题域的深层知识，所以它给出的解释通常仅限于其寻找解时所采取的步骤的描述。例如，它们通常不能说出“为什么”要选取某种方法。

4）难以验证。虽然任何庞大的计算机系统的正确性都是难以验证的，但是专家系统的验证就更加困难。因为专家系统技术被应用在关键应用中，比如空中交通控制、核反应堆操作以及武器系统，所以这是一个严重的问题。

5）不会根据经验学习。目前的专家系统大多是手工的；一旦系统完成，其性能就不再提高，除非程序员对其进行进一步的修改。这使人们对这种系统的智能性产生了很大的怀疑。

尽管专家系统存在这些局限，但它们还是在很多重要应用中证实了它们的价值。专家系统是本书的一个主要论题，具体讨论见第 7 章和第 8 章。当前的应用通常可以在 IAAI 会议的论文集中找到。

1.2.4 自然语言理解和语义学

人工智能一个经久不衰的目标就是开发出可以理解并产生人类语言的程序。这不仅是因为使用和理解人类语言的能力似乎是人类智能的一个基本特征，而且这种自动化会对计算机本身的用途和效力产生惊人的影响。人们已经付出了很多努力来编写理解自然语言的程序。尽管这些程序已经在某些特定的环境下取得了成功，但目前的方法学还无法让一个使用自然语言的系统具有刻画人类会话的灵活性和一般性的能力。

理解自然语言涉及很多问题，远远要比把语句分解为各个部分然后在字典中查到这些单词的含义要来得复杂。真正的理解必须依赖于对话领域的广泛背景知识、该领域的习惯用语，以及能够应用上下文知识处理人类言语中正常省略和模糊性的能力。

例如，考虑和一个懂英语但不懂棒球规则、选手以及历史的人谈论该项目时的困难。这个人会理解下面这句话的含义吗：“With none down in the top of the ninth and the go – ahead run at second，the manager called his relief from the bull pen”？虽然这句话中所有单词的单个含义是很好理解的，但是这句话对于不是棒球迷的高智商人也是无法理解的。

自然语言理解自动化的主要问题就是完成以下任务：收集和组织这种背景知识，以及如何以一种有助于领悟语言的方式来组织这些知识。针对这一需求，研究者们已经开发出了很多技术用来构建人工智能中使用的语义含义（见第 7 章和第 15 章）。

因为理解自然语言需要的知识量大得惊人，所以大多数工作都是在那些已被深入理解的专

业领域内完成。最早开拓这种“微小世界”方法学的程序之一是威诺格拉德的 SHRDLU，这个自然语言系统可以“谈论”不同形状和颜色积木的简单布局（Winograd 1973）。SHRDLU 可以回答类似这样的询问：“在蓝色方块上方的积木是什么颜色?”以及规划这样的动作：“将红的锥体移到绿色的砖块上”。这种问题涉及积木的简单排列方式的描述和操作，该类问题在早期 AI 研究中频繁出现，我们称其为“积木世界”问题。

尽管 SHRDLU 可以成功地就积木的排列进行交谈，它的方法还是不能从积木世界中推广到其他情况。用在这个程序中的表示技术过于简单，以致于无法表征更丰富、更复杂领域的语义结构。目前对于自然语言理解的大多数研究主要致力于寻找足够通用的具有代表性的形式方法，它既可以适用于很广范围内的应用，也可以很好地适用于某个给定领域的特定结构。针对这一目标，人们已经开发出了很多不同的技术（其中大多是对语义网的扩展或修改），并使用这些技术研制出可以理解特定但有趣知识域中自然语言的程序。最后要指出的是，在目前的研究（Marcus 1980，Manning and Schutze 1999，Jurafsky and Martin 2000）中采用了随机模型来刻画语法和语义，这些模型描述了单词和语言结构是如何在语言中“共现”的。不过，对语言的完全计算理解仍然是目前尚无法达到的。

1.2.5　对人类表现建模

虽然前面讨论的大多数领域都是以人类智能作为人工智能的参考目标，但是这并不等于说那些程序就是以人类思想的组织为模式的。事实上，很多 AI 程序就是针对求解问题的需要建立的，并没有考虑和人类大脑体系结构的相似性。即使对于专家系统，虽然它的大多数知识是从人类专家那里获得的，但是它并没有真正模拟人类大脑求解问题的内部过程。如果性能是判断系统好坏的惟一标准，那么根本没有必要试图模拟人类的问题求解方法。实际上，没有采用人类解决问题（国际象棋）的方法的程序相比对应的采用人类方法的程序经常可以更成功。不过，设计可以显式地模拟人类行为的某些方面的系统，一直是人工智能和心理学中的重要研究领域。

对人类行为建模除了为 AI 提供了很多基本方法学以外，还已经证明这是阐明和测试人类认知理论的一个有力工具。计算机科学家所开发出的问题求解方法学已经为心理学家探索人类思想提供了新的思路。很多心理学家不再采用早期研究中的模糊语言（也就是根据某些行为学家的建议完全放弃描述人类大脑的内部工作过程）来表述认知理论，而是采用计算机科学的语言和理论来阐释人类智能的模型。这些技术不仅为描述人类智能提供了很多新的词汇，而且这些理论的计算机实现为心理学家提供了一个机会去实验测试、评论和提炼他们的思想（Luger 1994；参考认知科学协会的期刊和会议）。第 16 章归纳了人工智能和人类智能之间的关系。

1.2.6　规划和机器人学

对规划的研究开始于设计具有一定灵活性并对外界具有响应性的机器人。简单地说，规划就是假定机器人可以执行特定的原子动作，并试图找到能够完成更高层任务的动作序列，比如穿过一个充满障碍的屋子。

很多原因导致规划成为一种复杂的问题，并不只是要考虑可能移动序列的空间大小。即使是一个相当简单的机器人也能产生大量的可能移动序列。举例来说，设想一个机器人可以向前、后、左、右移动，然后考虑这个机器人绕房间移动一周的方式有多少种。同时假定在这个房间里存在障碍，而且机器人必须选择一条路径以某种高效的方式绕过这些障碍。编写一个程序，它可以在这种情况下智能地发现最佳路径，而不被数量庞大的可能性所淹没。编写这个程序需要周密的技术来表示空间知识并控制对可能环境的搜索过程。

人类在规划中使用的一种方法是分层问题分解（hierarchical problem decomposition）。如果你在规划一次从阿尔伯克基到伦敦的旅行，那么你通常会分别处理以下一些问题：准备机票、到达机场、转机、寻找伦敦的地面交通方式，即使每个步骤都是较大规划的一部分，也需要进行分别处理。这些步骤中的每一个可能还会被分解为更小的子问题，比如寻找城市地图、乘坐地铁以及寻找合适的旅店。这种方法不仅有效地限制了必须搜索的空间的大小，而且可以把经常使用的子规划保存起来以备将来使用。

虽然人类做出规划一般不需要花费太大力气，但是产生一个可以完成同样任务的计算机程序却是非常富有挑战性的。要实现看起来非常简单的把一个问题分解为若干独立子问题的任务，实际上却需要复杂的启发式和规划领域的广泛知识。判断应该保存什么样的子规划以及如何把它们推广到将来的应用之中也具有同等的难度。

一个盲目执行动作序列而不对其所处环境的变化做出响应或者不能检测其自身规划并纠正其中误差的机器人很难被认为是具有智能的。机器人可能没有足够的传感器来定位预定路径上的所有障碍。因此机器人还必须根据它已经“感受”到的情况开始在房间中移动，并在检测到其他障碍物时纠正它的路线。以一种允许对不断变化的环境做出响应的方式来组织规划是规划研究中的一个主要问题（Lewis and Luger 2000，Thrun et al. 2007）。

最后要指出的是，通过机器人这一人工智能研究领域，我们可以看到面向主体的问题求解策略的很多细节（见1.1.4节）。在研究者们被维护庞大表示空间的复杂性所阻碍而且难以设计出胜任传统规划的搜索算法之后，包括Agre and Chapman（1987）、Brooks（1991a）以及Thrun等（2007）在内的研究者们开始把较大的问题重新表示成多个半自动主体的交互。每个主体负责整个问题任务中它自己的那一部分，并通过合作逐步使更高层解浮现出来。

规划研究目前已经远远超出了机器人学领域的范围，包含了协调任何复杂的任务或目标集合。现代规划程序已经被应用到主体（Nilsson 1994）以及粒子束加速器的控制之中（Klein et al. 1999，2000）。

1.2.7 人工智能的语言和环境

人工智能研究的最重要的副产品之一就是促进了程序设计语言和软件开发环境的发展。很多原因迫使AI程序员去开发一套强大的程序设计方法学，这些原因包括：很多AI应用程序都很大，原型设计方法学的重要性，使用搜索算法的倾向产生了庞大的空间，难以预测用启发式驱动的程序的行为。

程序设计环境包括各种知识构造技术，比如面向对象程序设计。像LISP和Prolog这样的高级语言支持模块化开发，这有助于控制程序的大小和复杂度。跟踪工具使程序员可以重新构造复杂算法的执行过程，从而有可能解决启发式搜索算法的复杂性。如果没有这些工具和技术，那么很多重要的AI系统能否建立起来是值得怀疑的。

目前这些技术大多已成为软件工程的标准工具，而与AI理论的核心已经没有什么关系了。其他一些类似于面向对象程序设计这样的方法在理论和实践中都起到了显著的作用。最后要说明的是，目前也在用一些比较传统的计算语言（比如C++和Java）来建立AI算法。

为人工智能程序设计开发的语言与这一领域的理论结构结合得更加紧密。我们用Prolog、LISP和Java实现了本书介绍的很多表征结构，读者可以通过Luger和Stubblefield（2009）得到，也可以通过互联网得到它们。本书仍然避开对不同语言的优劣的争论。相反，我们坚信这句名言：“好工匠都有自己善用的工具”。

1.2.8　机器学习

学习一直是 AI 中最具挑战的领域。学习的重要性是毋庸置疑的，因为这种能力是智能行为的最重要的特征。专家系统可以通过执行大量且费用昂贵的计算来求解问题。不过不同于人类的是，如果专家系统第二次遇到相同或类似的问题，它通常并不记得上一次的解，而是再执行一次相同的计算过程。对于第三次、第四次甚至任何次的重复出现，它的处理方式都是如此——很难说这是一个智能的问题求解器的行为。解决这一问题的显而易见的方法是让程序自己能够学习，不管是从经验、类比、实例中，通过被“告知”如何去做，还是根据结果对其进行奖惩。

虽然让程序自己学习是一个很难的领域，但是很多程序说明这是可能的。一个早期的程序是被设计用来发现数学定律的自动数学家（Automated Mathematician，AM）（Lenat 1977，1982）。在被赋予了集合理论的概念和公理后，AM 能够导出很多重要的数学概念，比如集合的基数、整数运算以及数论的很多结果。AM 通过修改其现有的知识库来猜想新的定理，然后启发式地从大量备择定理中搜寻“最佳者”。最近，科顿等人（Cotton et al. 2000）设计了一个程序，这个程序可以自动发明“有趣的”整数序列。

早期有影响的研究还包括温斯顿的研究，他根据积木世界中的一系列样例导出像“拱形”这样的结构概念（Winston 1975a）。ID3 算法已经被证明可以成功地从样例中学习通用模式（Quinlan 1986a）。Meta - DENDRAL 可以从具有已知结构的化合物数据样例中学习规则，用于解释有机化学中的大量光谱数据。Teiresias——专家系统的智能“前端”——可以把顶层的建议转化为新的规则放入知识库（Davis 1982）。计算机高手可以设计完成积木世界操作的规划，方法是使用一个迭代过程不断地设计规划、测试规划，然后纠正在候选规划中发现的所有不足（Sussman 1975）。对基于解释的学习的研究已经表明先验知识在学习中的有效性（Mitchell et al. 1986，DeJong and Mooney 1986）。目前还出现了很多重要的生物和社会学习模型，我们将在连接学习和涌现学习的章节（第 11 章和第 12 章）中介绍这些内容。

机器学习程序的成功表明存在一系列通用的学习原则，这些原则将使我们构造出具有在实际领域内进行学习的能力的程序。第四部分讨论了一些机器学习的方法。

1.2.9　其他表示：神经网络和遗传算法

本书介绍的大多数技术都是使用显式表示的知识和周密设计的搜索算法来实现智能。还有一些建立智能程序的方法与此完全不同，它们有的使用一种类似于人类大脑中神经元的模型，有的使用遗传算法和人工生命中发现的进化模式。

图 1-2 是神经元的简单示意图，它是由具有很多枝状突起（称为树突（dendrite））的细胞体和单个分支（称为轴突（axon））组成的。树突可以接收来自其他神经元的信号。当这些树突接收脉冲的混合结果超过某个阈值后，神经元便会激发，从而产生一个脉冲（或者尖峰）传向轴突。轴突末端的分支形成了与其他神经元相连的突触（synapse）。突触是神经元彼此之间的联系点；突触可以分为激发型（excitatory）和抑制型（inhibitory），对应于把到达的脉冲加入到总信号中或是从总信号中减去这个脉冲。

这种对神经元的描述有些过于简单，但是它抓住了与神经元的计算模型有关的那些特征。尤其是，每个计算单元以它的输入为参数来计算某个函数，并把结果传递到网络中的其他相连单元：最终的结果是由网络（神经系统的连接和阈值权）通过并行分布处理产生的。

神经元体系结构是实现智能的有力机制，这有很多原因。传统的 AI 程序比较脆弱而且对噪声过于敏感。人类智能要灵活得多，而且善于解释有干扰的输入，比如暗室内的人脸或嘈杂聚会

中的对话。因为神经体系结构使用分布在网络中精密组织的大量单元来获取知识，所以它们似乎更善于模糊地匹配带有干扰的数据和不完整的数据。

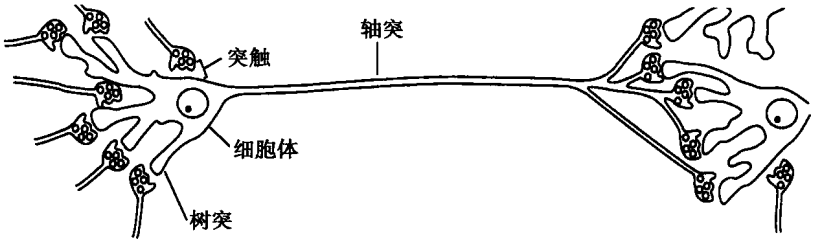


图 1-2　神经元的简单示意图

[摘自 Crick and Asanuma（1986）]

使用遗传算法和人工生命，我们从以前解决方案的各部分中演化出新的问题解决方案。遗传算子（如交叉和变异）更像自然界中它们的遗传等价物，它们为每个新后代产生更好的潜在的问题解决方案。人工生命根据以前各代中它的邻居的“质量”产生新后代。

神经元体系结构和遗传算法都提供了一种并行化的自然模型，因为每个神经元或解的片段都是一个独立的单元。希利斯（Hillis 1985）评论了这一事实：人类取得的知识越多，处理任务的速度越快，而计算机的速度往往会减慢。造成这种减慢的原因是顺序搜索知识库要花大量的时间；像人脑这样的高度并行体系结构没有这个问题。总而言之，从神经系统或遗传的角度处理智能问题具有某种固有的吸引力。毕竟，进化的大脑成就了智能，而大脑是使用神经元体系结构产生智能的。我们在第 10、11 章中介绍了神经网络、遗传算法和人工生命。

1.2.10　AI 和哲学

在 1.1 节中，我们介绍了人工智能的哲学、数学和社会学根源。这里要说的是，现代 AI 不仅是这些传统学科的产物，而且也为这些学科做出了贡献。

举例来说，图灵提出的关于智能程序的问题反过来也有助于我们对智能本身的理解。什么是智能，怎样才可以描述它？知识的属性是什么？知识可以表示吗？如何才能把某一应用领域内的知识与该领域的问题解决技巧联系起来呢？“*知道什么是正确的*”（即亚里士多德的理论）与“*知道如何去做*”（亚里士多德的实践）是如何联系起来的？

上面提出的这些问题构成了 AI 研究者和设计者所思考的一个重要部分。从科学的意义上来说，可以把 AI 程序看做是实验。程序使设计成为一个实体，因此运行程序就是实验的过程。程序设计者观察实验的结果，然后再重新设计，然后再进行实验。通过这种方式，我们可以判断我们的表示和算法是否足以成为智能行为的模型。纽维尔和西蒙（Newell and Simon 1976）在 1976 年获得图灵奖时所做的演讲中，提出了这种从科学角度来理解的方法（第六部分）。他们还提出了一种使用物理符号系统假设来表示智能的强模型。该假设的内容是：*一个物理系统表现出智能的充要条件是它必须是一个物理符号系统*。我们在第六部分中将继续讨论这个假设的含义，并介绍许多现代思想家是如何批评这个假设的。

许多 AI 的应用领域还揭开了深层的哲学问题。我们是从什么意义上来说计算机可以理解自然语言表达式？要产生或理解语言需要相应的符号解释。但我们没有充分的理由说明符号串是形式完善的。理解机制必须能够在上下文中估算语义或解释符号。语义是什么？解释是什么？从什么意义上来考核解释的效果？

很多 AI 应用领域都出现了类似的哲学问题，不论是与人类的问题求解高手合作建立专家系

统，设计计算机视觉系统，还是设计机器学习算法。本书的很多章节中都涉及这样的问题，第六部分再次探讨与哲学相关的一般问题。

1.3　人工智能小结

前面我们讨论了人工智能的主要研究和应用领域，我们希望通过这些讨论可以为人工智能下一个定义。前面的纵览向我们展现了一个年轻而且充满希望的研究领域，其宗旨是寻找一种有效的方式来把智能的问题求解、规划和通信技巧应用到更广泛的实际问题中。虽然人工智能研究所针对的问题非常广泛，但是很多重要的特征是该领域的所有分支所共有的，这些特征包括：

1）利用计算机来进行推理、模式识别、学习或其他形式的推断。

2）集中于不存在算法解的问题。这也是为什么启发式搜索是一种主要的 AI 问题求解技术的原因。

3）致力于那些利用不精确、不完全或没有良好定义的信息来求解的问题，而且要通过表示的形式方法来使程序员可以弥补这些问题。

4）推理目标是问题域的显著定性特征。

5）除了处理语法形式问题外，还要试图处理语义含义问题。

6）答案可能既不精确也不最优，但从某种意义上来说是“充分的”。这是依赖启发式问题求解方法所导致的结果，有些情况下得到优化或精确解的代价太高或者根本就不可能得到这样的解。

7）使用大量针对某一领域的知识来求解问题。这是专家系统的基础。

8）使用元层次的知识来实现对问题求解策略的更周密控制。虽然这是一个非常难的问题，目前的系统还很少采用，但是它已经逐渐成为一个关键的研究领域。

我们希望以上介绍可以使读者对人工智能的总体结构和价值有一定的了解。我们也希望那些对搜索和表示这样的技术问题的简单介绍不会让读者感到过于晦涩难懂，本书后文将展开讨论这些问题。这里提到它们是为了说明它们在人工智能理论体系中的重要性。

正如我们在讨论面向主体的问题求解策略中所提到的，对象是通过与其他对象的关系来体现其意义的。这一点对于构成一个科学研究领域的事实、理论和技术来说也是正确的。本书也想要突出这种相互关系，这样当介绍某个独立的技术主题时，读者便可以在人工智能的总体框架中找到该主题的位置，来理解其实质。这种观察方法是从心理学家和系统理论家格里戈里·贝特森那里借鉴来的（Bateson 1979）：

> 打断联系学习元素的模式，必定会破坏学习的质量。

1.4　结语和参考文献

AI 领域反映了西方文明某些最古老方面对现代计算模型的影响。理性化、表示和推理这些概念目前正经受着前所未有的详细研究，因为我们这些 AI 领域的工作者们要求用算法来理解它们。同时，政治、经济和伦理形势也迫使我们要对我们发明的影响负责。

关于本章提出的这些主题，以下著作中也有精彩的论述：《Mind Design》（Haugeland 1997）、《Artificial Intelligence：The Very Idea》（Haugeland 1985）、《Brainstorms》（Dennett 1978）、《Mental Models》（Johnson-Laird 1983）、《Elbow Room》（Dennett 1984）、《Body in the Mind》（Johnson 1987）、《Consciousness Explained》（Dennett 1991）以及《Darwin's Dangerous Idea》（Dennett 1995）、《Prehistory of Android Epistemology》（Glymour，Ford and Hayes 1995a）和《Sweet Dreems》（Dennett 2006）。

也可以参阅一些原始的资料，包括亚里士多德的《物理学》、《形而上学》和《逻辑学》；Frege 的论文；以及巴贝奇、布尔和罗素与怀海德的著作。图灵的论文也是非常值得一读的，特别是他对智能特征和设计智能程序可能性的讨论（Turing 1950）。图灵 1937 年在著名的论文《论可计算数及其在可判定性问题中的应用》（On Computable Numbers, with an Application to the Entscheidungsproblem）中提出了图灵机理论和可计算性的定义。图灵的传记《阿兰·图灵：谜》（Hodges 1983）也是一本精彩的读物。Selfridge 的《万魔殿》（Pandemonium）（Selfridge 1959）是个早期的学习典范。人工智能早期文章的一个重要汇总可以在 Webber 和 Nilsson（1981）中找到。

《Computer Power and Human Reason》（Weizenbaum 1976）和《Understanding Computers and Cognition》（Winograd and Flores 1986）对 AI 的不足以及 AI 中的伦理问题进行了冷静的评论。《The Sciences of the Artificial》（Simon 1981）对人工智能的可能性和它对社会的作用给予了肯定。

1.2 节中所提到的 AI 应用旨在向读者介绍 AI 研究者的广泛兴趣并勾勒出一些正在探索的重要问题。每个小节都在介绍主题时参考了原始领域资料。《人工智能手册》（Barr and Feigenbaum 1989）对这些领域进行了逐一介绍。《Encyclopedia of Artificial Intelligence》（Shapiro 1992）对人工智能领域进行了清晰全面的分析。

自然语言理解是 AI 研究的一个活跃领域，以下著作介绍了一些重要的观点：《Natural Language Understanding》（Allen 1995）、《Language as a Cognitive Process》（Winograd 1983）、《Computer Models of Thought and Language》（Schank and Colby 1973）、《Grammar, Meaning and the Machine Analysis of Language》（Wilks 1972）、《The Language Instinct》（Pinker 1994）、《Philosophy in the Flesh》（Lakoff and Johnson 1999）以及《Speech and Language Processing》（Jurafsky and Martin 2009）。本书的第 7 章和第 15 章讨论了这一领域。

本书第 17 章简要地讨论了如何使用计算机对人类的行为建模，以下著作更深入地讨论了这一课题：《Human Problem Solving》（Newell and Simon 1972）、《Computation and Cognition》（Pylyshyn 1984）、《Arguments Concerning Representations for Mental Imagery》（Anderson 1978）、《Cognitive Science: the Science of Intelligent Systems》（Luger 1994）、《Problem Solving as Model Refinement: Towards a Constructivist Epistemology》（Luger et al. 2002）以及《Bayesian Brain》（Doya et al. 2007）。

机器学习在第四部分讨论；《Machine Learning》合集（Michalski et al. 1983, 1986; Kodratoff and Michalski 1990）、《Journal of Artificial Intelligence》和《Journal of Machine Learning》都是关于这一主题的重要资料。进一步的参考文献可以在第四部分的 4 章中查找。

最后，第 12 章从社会和自然角度探讨了智能，这一观点强调了智能的模块化结构和适应性。明斯基的《Society of Mind》（Minsky 1985）是最早提出这一观点和最有思想深度的著作之一。其他著作包括《Android Epistemology》（Ford et al. 1995b）和《Artificial Life》（Langton 1995）。

1.5 习题

1. 给出你自己对人工智能的定义，并说明理由。
2. 给出几个例子来说明亚里士多德关于物质和形式的差异。并说明你的例子是如何适应一种抽象理论的。
3. 多数传统的宗教思想都详细地论述了心身（mind-body）的关系。心智思想和身体的关系是：
   a）以某种方式相互作用的不同实体。
   b）心智思想是“肉体过程行为”的表达。
   c）身体只是理性思想心智的幻影。
   探讨你对心身问题的看法，以及这一问题对人工智能理论的重要性。

4. 讨论图灵关于计算机软件“智能”标准的不足。
5. 描述一下你自己的计算机软件“智能”标准。
6. 虽然计算机是一个相当新的学科，但是数千年来哲学家和数学家都在思考自动问题求解中所涉及的问题。你认为这些哲学问题和智能问题求解设备的设计有关系吗？为什么？
7. 我们知道人类大脑的体系结构和现代计算机的体系结构有很大的差异，那么你认为研究生物系统的生理结构和功能对建立 AI 程序有价值吗？为什么？
8. 挑选一个你认为最富挑战性的专家系统应用领域，适当地展开这个问题。根据你的直觉，这个问题的哪些部分最难以自动化？
9. 除了书中列出的优点之外，再列举专家系统的两个优点。从智能性、社会性和经济性的角度讨论这些优点。
10. 讨论为什么说机器“学习”问题是非常难的。
11. 你认为计算机是否有可能理解并使用自然（人类）语言，为什么？
12. 列举并讨论人工智能技术对社会发展可能造成的两种负面影响。

第二部分　作为表示和搜索的人工智能

关于达特茅斯夏季人工智能研究项目的一项提议：

我们提议1956年夏天在新汉普郡汉诺威市达特茅斯学院举行一次10人参加为期两个月的人工智能研讨会。这次研讨会的主题是建立在一项猜想的基础上，即原则上学习的每个方面或智能的任何特征都能被精确地描述到用机器来模拟的程度。我们将探寻如何制造机器来使用语言、形成抽象化和概念化、解决目前只有人类才能解决的问题，并能实现自我改进。我们相信，精选一组科学家共同工作一夏天，就能在一个或多个问题上取得有重大意义的进展。

麦卡锡（J. McCarthy），达特茅斯学院
明斯基（M. L. Minsky），哈佛大学
罗彻斯特（N. Rochester），IBM公司
香农（C. E. Shannon），贝尔语音实验室
1955年8月31日

表示与搜索简介

从工程的观点来看，1.3节中关于人工智能的描述可总结为：对表示和搜索的研究，通过这些研究，智能行为将在机械设备上产生。这个观点主导了人工智能的起源与成长过程。

人工智能实践者们的第一个现代研讨会/学术会议于1956年夏在达特茅斯学院举行。本书第二部分的介绍就是引用这次研讨会的提议，这次研讨会选择了人工智能作为名称，聚集了许多当时专注于将计算和智能结合的研究者。当时还编写出了几个计算机程序来表现这些早期的想法。我们精简原始的会议提议，列出这次会议讨论的主要议题：

1）自动计算机。如果机器能够做一项任务，那么可以编程让自动计算器模拟这台机器。

2）如何通过编程让计算机使用语言。可以这样推测：人类思维的很大部分是按照推理规则和猜测规则来操纵词句。

3）神经网络。如何将许多（假想的）神经元组织起来形成概念？

4）计算范围理论。给定一个定义明确的问题（比如机械地测试给出的一个答案是否正确），一个解决方法是依次尝试所有可能的答案。这种方法是效率极低的，必须有某种评判标准才能验证这种低效的方法。

5）自我改进（机器学习）。或许一台真正智能的机器能够执行一些自我改进的行为是最合适的说法。

6）抽象概念。许多类型的“抽象概念”能够清楚地定义，而另外一些不够清楚。尝试将它们区分出来，并且机器方法描述从感官数据和其他数据形成抽象概念是很有价值的。

7）随意性和创造性。有一个非常吸引人但显然不完备的猜测，即创造性思维和缺乏想象力的思维之间的区别在于随机性的加入。

值得注意的是，第一次人工智能学术会议提议的主题抓住了许多重要问题，例如复杂性理论、抽象概念的方法学、语言设计和机器学习，这些问题构成了现代计算机科学关注的焦点。实

际上，计算机科学中许多今天已知的典型定义都能在人工智能中找到起源。人工智能也有自身的历史和策略上的竞争，早期提出的几种研究主题，如“神经网络”和“随机性和创造性”，已经退居后台数十年了。

这时，一种强有力的新型计算工具 LISP 语言出现了。LISP 语言是在约翰·麦卡锡的指导下开发出来的，麦卡锡是达特茅斯研讨会最初发起者之一。LISP 着重解决研讨会的几个主题，支持创建关系的能力，这些关系能够由语言的其他结构来操纵。LISP 为人工智能带来了具有很高级表达能力的语言、丰富的抽象，以及解释表达式的工具。

LISP 程序设计语言的有效性促成了 AI 早期的许多进步，特别是作为表达工具的谓词演算和探测不同逻辑选择效果的搜索的运用，现在称为*图搜索*。创建于 20 世纪 70 年代的 Prolog 为人工智能提供了一个同样强有力的计算工具。

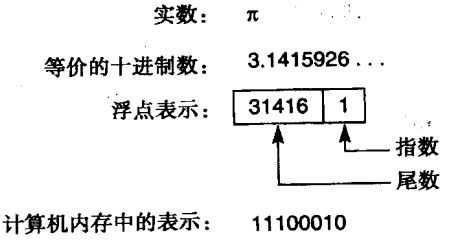
第二部分的 5 章介绍支持人工智能研究的基本表示法和搜索技术，具体包括谓词演算、图搜索、启发式和随机方法以及用于智能问题求解的体系结构（控制系统）。这些方法反映了人工智能领域最初 20 年中占统治地位的技术。

表示系统

所有表示模式的功能都是要捕捉（通常称为抽取）问题域中的本质特征并使这一信息能够被问题求解过程所访问。*抽象*（abstraction）是处理复杂性的一种关键工具，也是保证最终程序计算高效性的一个重要因素。*表现力*（特征抽取的结果）和*效率*（特征抽取算法的计算复杂度）是评价知识表示语言的主要尺度。有时，为了提高效率必须牺牲表现力。但这是以不限制捕捉关键的问题求解知识的表示能力为前提的。因此，使效率和表现力达到最佳平衡是智能程序设计者的一个主要任务。

知识表示语言也是帮助人类求解问题的工具。因此，表示应该为表达问题求解的知识提供一种*自然框架*；它应该使这些知识能够被计算机所使用并有助于程序员组织这些知识。

下面以浮点数的计算机表示（见图Ⅱ-1）为例来说明这种平衡。通常，要完全描述实数需要无限位；这无法在一个诸如计算机这样的有限设备上实现。对于这个问题，一种解决方法是把数字表示为两部分：*有效位*和其中小数点的位置。尽管不可能在计算机中实际存储一个实数，但是要建立一种足以满足大多数实践应用的表示是有可能的。



图Ⅱ-1　实数 π 的不同表示

在这个例子中，浮点表示为了效率而牺牲了完全表示所具有的表达力。这种表示也允许算法进行多精度的计算，通过把缩减误差限制在预先指定的允许范围内有效实现了很高精度。它还保证了缩减误差得到很好处理。与所有表示一样，它仅是一种抽象，是一种表征实际实体的符号模式，而不是实体本身。

数组是计算机科学中另一种常见的表示。对于很多问题来说，它比计算机硬件所实现的内存体系结构更加自然和高效。这种在自然性和高效性方面的优势是由表示力的折中换来的，以下图像处理的例子说明了这一点。图Ⅱ-2 是人类基因在一个称为*转位期*（metaphase）的阶段中的数字图像。现在要处理这幅图像来分析基因的数量和结构，并寻找断裂、残缺的片段和其他异常。

图中的可视画面是由很多个图像点组成的。而且每个图像点（即*像素*）都有两个属性：位

量和表示其亮度或灰度的一个数字值。那么，很自然的做法是把整幅画面存入一个二维数组，行和列的地址对应像素位置（X 和 Y 坐标），数组元素的内容是对应点的灰度。然后设计算法进行这样一些操作：寻找孤立点以去除图像中的噪声，找到一个阈值来辨识对象和边缘，汇总邻接的要素以确定大小或密度，以其他方法变换像素数据。如果确定了这种数组表示和一种程序设计语言（例如 FORTRAN），那么实现这些算法是很简单的。如果使用其他的表示（比如谓词演算、记录或汇编代码），那么这个任务就会非常复杂，因为这些表示方法与被表示的物质间不存在一种自然的契合。

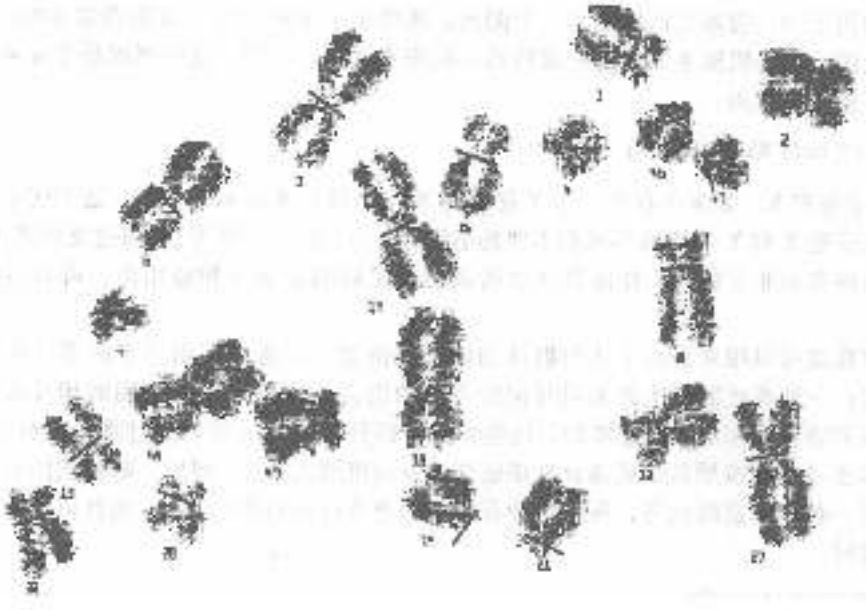


图 II-2 染色体在转位期中的数字图像

当我们把图片表示为像素点数组时，我们牺牲了优良的画质（好比报纸上的照片和照片的原版在品质上存在很大的差距）。此外，像素数组不能表示图像的更深层语义结构。例如，一个像素数组不能表示一个细胞核中的基因结构，它们的遗传特征或者转位期对细胞分裂的作用。这种知识更易于被谓词演算（见第 2 章）和语义网（见第 7 章）这样的表示来捕捉。总之，表示模式应足以表示所有必要信息，支持最终代码的有效执行，且为表示所需的知识提供一个自然的框架。

通常，AI 要解决的问题是数组这样的传统形式所无法解决的。因为人工智能更加关心定性的问题求解而不是定量的问题求解，更关心推理而不是演算数值计算，更关心如何组织庞大的数量变化的知识而不是实现单一的已经具有完善定义的算法。

例如，考虑图 II-3 中表示上积木的假设。假设我们希望获取控制机械人手臂所需的属性和关系。我们必须确定哪些积木块位于其他积木块上面，哪些积木块上面是空的可以抓起。谓词演算提供了一种获取这些描述信息的手段。每个表达式的第一个单词（on、ontable 等）都是一个谓词，代表了某种属性或其参数（出现在括号中）之间的某种关系。参数是表示问题域中对象（积木）的符号。下面的一系列逻辑语句描述了这个积木世界中的重要属性和关系：

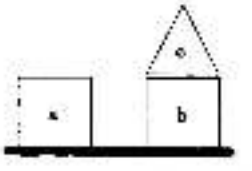


图 II-3 积木世界

clear(c)
clear(a)
ontable(a)
ontable(b)
on(c, b)
cube(b)
cube(a)
pyramid(c)

谓词演算为人工智能程序员提供了一种定义完善的语言，可用来描述和推理系统定性特征。积木世界的例子中，假如我们要定义一个测试，来确定一个积木块上方是否是 clear 的，即没有东西在其上面。如果机械手要抓起它或将另一块积木放到它上面，这个测试是非常重要的。我们可以定义一条通用规则：

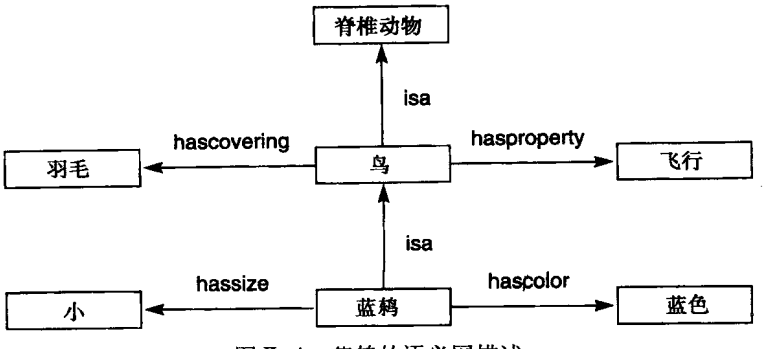
$$\forall X \neg \exists Y \, on(Y,X) \Rightarrow clear(X)$$

读作："对于所有 X，如果不存在一个 Y 使其在 X 上，那么 X 是 clear 的"。这个规则可以用于各种情况，只要把 X 和 Y 替换成不同积木块的名字（如 a、b、c，等等）。通过支持通用推理规则，谓词演算使得表示非常精简，并使设计出的系统有足够的灵活性和通用性，可对各种情况做出智能反应。

谓词演算也可以用来表示个体与群体的属性。例如，仅通过列出汽车的零件来描述一辆汽车是不够的；一种有效的描述必须同时包括零件的组合方式以及它们之间的相互关系。这种结构视图在很多情况下都是十分重要的，这些情况包括分类信息（例如通过物种来分类植物）、复杂对象的描述（例如按照其组成部分来描述柴油发动机或人体）。例如，对蓝鹀的简单描述可以是"蓝鹀是一种小的蓝颜色鸟，鸟是一种有羽毛的会飞行的脊椎动物"，而且可以将其表示为一系列逻辑谓词：

hassize(bluebird,small)
hascovering(bird,feathers)
hascolor(bluebird,blue)
hasproperty(bird,flies)
isa(bluebird,bird)
isa(bird,vertebrate)

也可以不使用谓词描述，而是通过图中的弧（arc）或连接（link）实现用图形方式表示关系（见图Ⅱ-4）。这种语义网（semantic network）是表示语义含义的一种技术。因为关系是在图中显式表示的，所以推理算法只要顺着连接就可以找出相关的联系。在蓝鹀的例子中，系统只需跟踪一个连接就可以看到蓝鹀会飞，跟踪两个连接就可以确定蓝鹀是脊椎动物。



图Ⅱ-4　蓝鹀的语义网描述

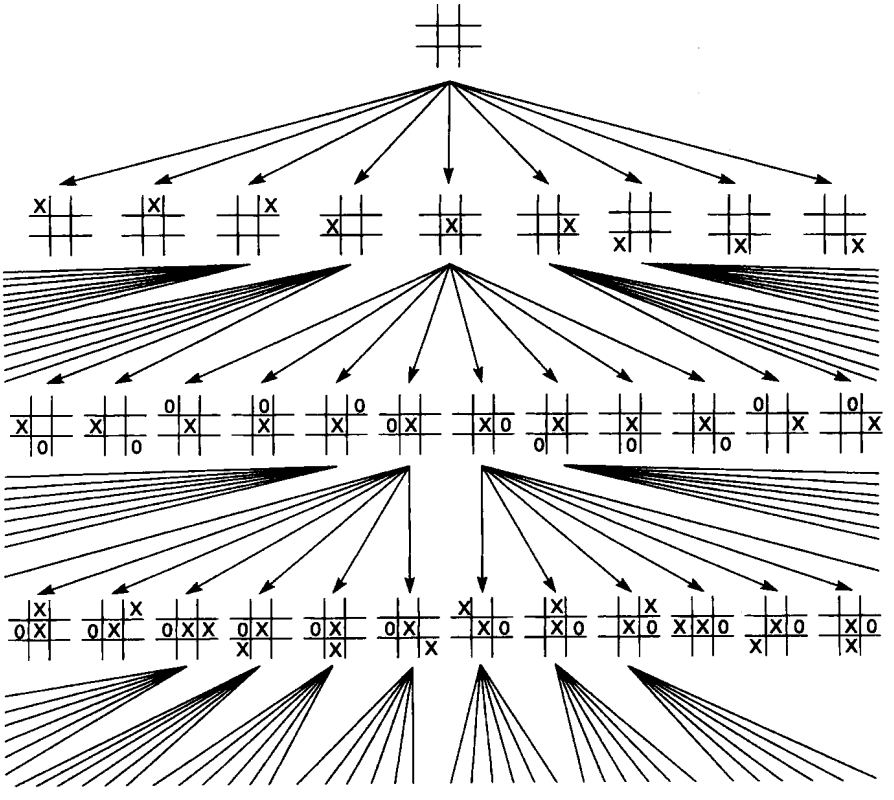
语义网的最重要应用也许就是在语言理解程序中表示语义内涵。当需要理解童话、杂志文章的细节或网页的内容时，就有必要用语义网对反映这些应用中知识的信息和关系进行编码。我们将在第7章中讨论语义网，并在第15章中介绍它在语言理解方面的应用。

搜索

有了表示，智能问题求解的第二个组成部分就是*搜索*。人类在求解问题时通常考虑很多候选策略。棋手一般要考虑很多可选走法，再从这些方案中选择“最佳”的一种。他们选择的根据是对手的可能反应或不同走法对某个全局策略的支持程度等。当然，棋手也会考虑短期的收获（比如吃掉对方的王后），以牺牲一个棋子来取得位置优势的机会，或猜测对手的心理素质和技术水平。智能行为的这一特征为*状态空间搜索*的问题求解技术奠定了基础。

下面举一个九宫游戏的简单例子。对于任何给定的棋盘状态，棋手仅有有限数量的走法。从空的棋盘开始，第一个棋手可能放一个X在9个位置中的任一个。但不论放在哪里，留给对手的可能反应都是8种，依此类推。我们可以把这些可能走法和反应用图表示出来，每一种棋盘格局作为图中的一个*结点*或*状态*。图的*连接*表示从一种棋局到另一种棋局的合法移动。这样产生的结构被称为*状态空间图*（state space graph）。

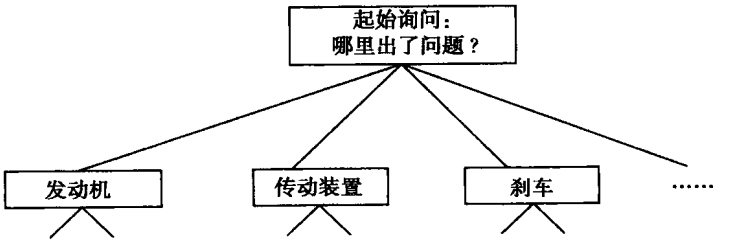
状态空间表示使我们可以把九宫游戏的所有可能对弈过程看成是穿过状态空间图的不同路径。有了这个表示，一种有效博弈策略就是遍历这个图搜索出最多胜利、最少失败的路径，并总是尽可能地沿这些最优路径之一移动，如图Ⅱ-5所示。



图Ⅱ-5　九宫游戏状态空间的一部分

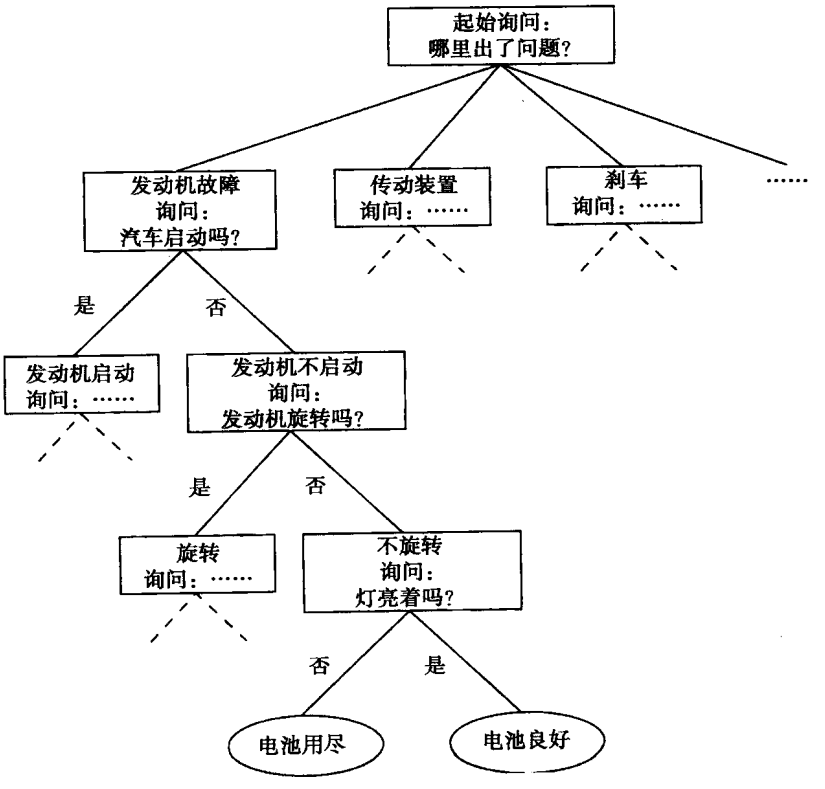
为了说明如何使用搜索解决更复杂的问题，下面考虑一个诊断汽车机械故障的例子。尽管这个问题不像九宫游戏那样容易发现状态空间搜索的使用，但它的确很适合这种策略。在这个

问题中，我们让状态空间图的每个结点不再表示一个“棋盘状态”，而让它表示汽车机械故障的一部分知识。我们可以把分析故障症状和归纳故障起因的过程看成是对渐增的知识状态的搜索。这个图的起始结点是空的，这表示还根本不知道问题的起因。修理工做的第一件事可能是询问客户哪一个主要系统（发动机、传动装置、转向装置、刹车等）看起来出了问题。图Ⅱ-6 用起始状态到代表汽车某一独立子系统状态的一系列弧表示出了这个过程。



图Ⅱ-6　诊断汽车故障第一步的状态空间描述

图中的每个状态都有弧（对应于基本的诊断检查）指向表示诊断过程中进一步知识积累的状态。例如，“发动机故障”结点有连接到标有“发动机启动”和“发动机不启动”结点的弧。从“不启动”结点我们可以移动到标有“旋转”和“不旋转”的结点。从“不旋转”结点有连接到标有“电池用尽”和“电池良好”结点的弧，见图Ⅱ-7。问题求解过程可以通过从这幅图中搜索出一条与损坏车辆的症状相吻合的路径来诊断这辆车的故障。尽管这个问题与寻找九宫游戏或国际象棋的最佳玩法的问题差异很大，但是同样能使用状态空间搜索来求解它。



图Ⅱ-7　汽车诊断问题的状态空间描述

尽管看起来似乎无所不包，但是状态空间搜索本身无法独立完成自动智能问题求解行为，相反它只是智能程序设计的一个重要工具。如果状态空间搜索能够独立完成，那么写一个下棋程序是相当简单的，只要遍历整个空间找到获胜的走法序列就可以了。这种问题求解方法称为*穷举搜索*（exhaustive search）。尽管穷举搜索可以应用到任何状态空间，但实际问题对应的空间可能惊人的庞大，这使得这种方法在实践中不可行。例如，国际象棋大约有10^{120}种不同的棋盘状态。这个数字比宇宙中的分子数或自从大爆炸以来过去的纳秒数还大。搜索这样大的空间远远超过了任何计算设备的能力，因为任何设备的尺度都以已知的宇宙为极限，而且它的执行过程必须在宇宙崩溃前完成。

人类使用智能搜索：棋手考虑很多可能的走法，医生分析几种可能的诊断，计算机科学家在开始编码前要深思熟虑不同的设计。但是人类不使用穷举式搜索：棋手仅分析经验表明有效的走法，医生不需要做与患者症状无关的检验。因此，人类求解问题似乎是以一些判断性规则为基础的，这些规则使我们的搜索仅寻找状态空间中那些似乎最“奏效”的部分。

这些规则被称为*启发*，这是AI研究的中心课题之一。启发（这个名字来源于希腊语中的“to discover”）是有所选择地搜索问题空间的策略。它引导搜索沿高成功概率的路线前进，避免做多余或明显愚蠢的努力。人类在求解问题时使用大量启发。如果你问一个修理工为什么你的轿车过热，她可能说一些像这样的话，“这通常意味着自动调温器坏了。”如果你问医生什么会导致恶心和胃痛，他可能会说“不是胃炎，就是食物中毒”。

状态空间搜索为我们提供了一种形式化问题求解过程的手段，启发使我们可以为这个过程注入智能。本书的前些章节详细地讨论了这些技术，而且在大多数现代AI研究中它们始终处于核心地位。概括地讲，状态空间搜索是独立于任何特定搜索策略的一种形式，被用做很多问题求解途径的切入点。

本书自始至终都在从不同角度探索知识表示和搜索的理论以及如何使用这一理论建立有效的程序。对知识表示的讨论是从第2章的谓词演算开始的。第3章以博弈图和其他应用为背景介绍搜索。在第4章，我们介绍启发，并用它来实现图搜索，包括博弈。在第5章，我们介绍随机（概率）技术来建立和组织搜索空间；这些方法在后面的机器学习和自然语言处理部分将用到。最后，第6章介绍建立智能问题求解器需要的产生式系统、黑板和其他软件体系结构。

第2章 谓词演算

我们正在逐步掌握推理的能力，并最终使之成为我们自己的才能；因为推理并不是一种天赋，而是需要漫长而艰苦的努力才能够具备的一种才能。

——C. S. 皮尔斯

证明的本质在于使人信服。

——费马

2.0 简介

在这一章中，我们介绍谓词演算（predicate calculus），它是用于人工智能的一种表示语言。谓词演算的重要性已经在第二部分的介绍中有所提及，它的优点包括明确定义的形式语义（formal semantics）以及可靠（sound）和完备（complete）的推理规则。在这一章中我们首先简要（可选）介绍命题演算（2.1节）。然后在2.2节中定义谓词演算的语法和语义。在2.3节中讨论谓词演算推理规则和它们在问题求解中的用法。最后，实现一个投资建议知识库，用于演示谓词演算的应用。

2.1 命题演算（选读）

2.1.1 符号和语句

命题演算和下一节要讨论的谓词演算是所有语言的基础。使用它们的单词、短语和语句，我们可以表示和推理客观世界的属性和关系。描述一种语言的第一步应该介绍组成它的片段，即它的符号集。

定义（命题演算符号） 命题演算的**符号**包括命题符号：

P, Q, R, S, …

真值符号：

true, false

和连结词：

∧, ∨, ¬, →, ≡

命题符号表示命题或对世界的陈述，它可能是真的也可能是假的，比如“那辆轿车是红色的”或“水是湿的”。英语字母表靠近末尾的大写字母表示命题。命题演算中的语句是由这些原子符号根据以下规则组成的。

定义（命题演算语句） 每个命题符号和真值符号都是一条语句。

例如：true、P、Q 和 R 都是语句。

语句的**非**（negation）是一条语句。

例如：¬ P 和¬ false 是语句。

两条语句的**合取**（conjunction）或**与**（and）是一条语句。

例如：P ∧ ¬ P 是一条语句。

两条语句的**析取**（disjunction）或**或**（or）是一条语句。

例如：P∨¬P是一条语句。

一条语句对另一条的**蕴涵**（implication）也是一条语句。

例如：P→Q是一条语句。

两条语句的**等价**（equivalence）是一条语句。

例如：P∨Q≡R是一条语句。

合法的语句又被称为**合式公式**（well-formed formula）或WFF。

在P∧Q形式的表达式中，P和Q被称为合取项（conjunct）。在P∨Q形式的表达式中，P和Q被称为析取项（disjunct）。在蕴涵P→Q中，P是前提（premise）或者叫前项（antecedent）；Q是结论（conclusion）或者叫后项（consequent）。

在命题演算语句中，符号（ ）和[ ]用来把符号组合成为子表达式，从而控制计值顺序和含义。例如，(P∨Q)≡R与P∨(Q≡R)是完全不同的，利用真值表可以看出它们的不同，参见2.1.2节。

一个表达式是命题演算的一条语句或合式公式，其充要条件是，它是由合法的符号按上述规则的某个序列来形成的。例如：

((P∧Q)→R)≡¬P∨¬Q∨R

是命题演算的一条合式语句，因为：

P、Q和R是命题符号，因此它们都是语句。

P∧Q是两条语句的合取，是一条语句。

(P∧Q)→R是一条语句蕴涵另一条语句，所以也是一条语句。

¬P和¬Q是语句的非，都是语句。

¬P∨¬Q，两条语句的析取是一条语句。

¬P∨¬Q∨R，两条语句的析取是一条语句。

((P∧Q)→R)≡¬P∨¬Q∨R是两条语句的等价，是一条语句。

这就是我们的原始语句，它是通过应用一系列的合法规则来构建的，所以是合式的（well-formed）。

2.1.2 命题演算的语义

2.1.1节通过定义产生合法语句的一系列规则给出了命题演算的语法。在这一节中，我们形式化地定义这些语句的语义（semantic），或者说“含义”。因为AI程序必须借助它们的表示结构进行推理，所以证实推理结论的真实性仅依赖于用于推理的初始知识或前提的正确性，这一点是非常重要的（即推理过程没有引入逻辑错误）。对语义的精确分析是实现这个目标的关键。

一个命题符号对应于对世界的一种陈述。例如，P可能代表“正在下雨”这一陈述；Q可能代表“我家的房子是褐色的”。一个命题要么是真要么是假，给出了世界的某种状态。为命题语句赋真值被称为**解释**（interpretation），即关于它们在某个可能世界的一个断言。

严格地讲，解释就是把命题符号映射到集合{T, F}。正如上一小节提到的，符号true和false是命题演算的合式语句集的组成部分，它们不同于赋给语句的真值。为了明确这种差异，在真值赋值时使用符号T和F。

对命题的每一种可能真值赋值对应于解释的一种可能世界。例如，如果P指称命题“正在下雨”，Q指称“我在工作”，那么命题集合{P, Q}到真值集合{T, F}有4种不同的映射。这些映射对应于4种不同的解释。命题演算的语义和它的语法一样是通过归纳来定义的：

定义（命题演算语义）　对一个命题集合的**解释**就是为每一个命题符号赋真值，要么是 T，要么是 F。

符号 true 的赋值总是 T，符号 false 的赋值总是 F。

语句的解释（也就是真值）按以下原则决定：

- 非¬ 的真值是这样的：¬ P，其中 P 是任意命题符号，如果对 P 的赋值是 T，那么对¬ P 的赋值是 F；如果对 P 的赋值是 F，那么对¬ P 的赋值是 T。
- 仅当两个合取项的真值都是 T 时，**合取**∧的真值是 T，否则是 F。
- 仅当两个析取项的真值都是 F 时，**析取**∨的真值是 F，否则是 T。
- 仅当**蕴涵**→的前提（即蕴涵前面的符号）是 T 而且后项（即蕴涵后面的符号）是 F 时，蕴涵的真值时 F，否则是 T。
- 仅当**等价**≡两边表达式对于所有可能解释都有相同真值时，等价的真值是 T，否则是 F。

经常使用真值表（truth table）来描述复合命题的真值赋值。真值表列出了对表达式原子命题的所有可能赋值，并给出对于每一种可能赋值而言表达式的真值。因此，真值表枚举了一个表达式解释的所有可能情况。例如图 2-1 是 P∧Q 的真值表，列出了对操作数的每一种可能赋值而言表达式的真值。从表中可以看出，仅当 P 和 Q 都是 T 时，P∧Q 是真。可以用类似的方式定义或（∨）、非（¬）、蕴涵（→）和等价（≡）。这些真值表作为练习留给大家来做。

在命题演算中，两个表达式等价的条件是对于任何赋值，它们的真值都相同。可以使用真值表来证实这种等价性。例如，图 2-2 中的真值表证明了 P→Q 和¬ P∨Q 的等价性。

P	Q	P∧Q
T	T	T
T	F	F
F	T	F
F	F	F

图 2-1　操作符∧的真值表

P	Q	¬P	¬P∨Q	P⇒Q	(¬P∨Q)=(P⇒Q)
T	T	F	T	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

图 2-2　证实 P→Q 和¬ P∨Q 等价性的真值表

通过显示命题演算中两个不同语句有相同的真值表，我们可以证明以下等价性。对于命题表达式 P、Q 和 R：

¬ (¬ P) ≡P

(P∨Q) ≡(¬ P→Q)

换质换位定律：(P→Q) ≡(¬ Q→¬ P)

德·摩根定律：¬ (P∨Q) ≡(¬ P∧¬ Q)和¬ (P∧Q) ≡(¬ P∨¬ Q)

交换律：(P∧Q) ≡(Q∧P)和(P∨Q) ≡(Q∨P)

结合律：((P∧Q) ∧R) ≡(P∧(Q∧R))

结合律：((P∨Q) ∨R) ≡(P∨(Q∨R))

分配律：P∨(Q∧R) ≡(P∨Q) ∧(P∨R)

分配律：P∧(Q∨R) ≡(P∧Q) ∨(P∧R)

可以利用像这样的恒等式把命题演算变换成语法不同但逻辑等价的形式。也可以利用这些恒等式代替真值表来证明两个表达式的等价性：通过一系列恒等式把一个表达式变换成另一个表达式。一个早期的 AI 程序，Newell、Simon 和 Shaw 设计的**逻辑理论家**（Logic Theorist）（Newell and Simon 1956），使用表达式间的恒等变换，证明了 Whitehead 和 Russell 的《数学原理》（1950）中的很多定理。有些推理规则（2.3 节中的假言推理和第 14 章中的归结）要求表达式必

须为某种特定的形式，这时把一个逻辑表达式变成具有等价真值的不同形式的能力也是很重要的。

2.2 谓词演算

在命题演算中，每个原子符号（P、Q，等等）表示一个命题。我们没有办法做到判断断言的各个部分。谓词演算提供了这种能力。例如，不再是让一个命题符号 P 表示整个句子“星期二下了雨”，而是创建一个谓词 weather 来描述日子和天气的关系：weather(tuesday, rain)。我们通过推理规则可以操纵谓词演算表达式，判断它的每个组成成分，并且推理出新的语句。

谓词演算还允许表达式中含有变量。变量使我们可以建立关于实体类的通用断言。例如，我们可以声明对于所有的 X 值，其中 X 是某一周的一天，陈述 weather(X, rain) 是真的；也就是说这一周一直下雨。和讨论命题演算一样，我们先定义这种语言的语法，然后讨论它的语义。

2.2.1 谓词的语法和语句

在定义谓词演算中正确表达式的语法之前，我们先定义字母表和创建这种语言所用符号（symbol）的语法。这对应于一种程序设计语言的词法部分。谓词演算的符号与程序设计语言中的记号（token）一样是不可约简的语法元素：不可用语言中的操作把它们拆散为更小的部分。

在本书中，我们把谓词演算符号表示为字母和以字母开始的字母数字串。空格和字母与数字之外的字符不可以出现在串中，但下划线字符（_）例外，因为它可以提高可读性。

定义（谓词演算符号） 用以构成谓词演算符号的字符表由以下 3 部分组成：

1）英文字母，包括大写和小写。

2）数字 0，1，…，9。

3）下划线_ 。

谓词演算的符号以字母开始，后面可以跟有这些合法字符的任意序列。

谓词演算符号字符表中的合法字符包括：

a R 6 9 p _ z

不在这个字符表中的字符例子包括：

% @ / &

合法的谓词演算符号包括：

George fire3 tom_ and_ jerry bill XXXX friends_ of

下面的字符串实例不是合法的谓词演算符号：

3jack no blanks allowed ab%cd * * *71 duck!!!

在 2.2.2 节中，我们会看到符号被用来表示所讨论世界中的对象、属性或关系。与大多数程序设计语言相同，使用能够暗示出符号潜在含义的“单词”作为符号有助于理解程序代码。因此，即使 l(g, k) 和 likes(george, kate) 形式上是等价的（也就是它们有相同的结构），但是（对人类阅读者而言）第二种非常有助于指出表达式所表示的关系。必须强调，这些描述性的名字完全是为了提高表达式的可读性。谓词表达式可以具有的惟一含义必须通过它们的正式语义来确定。

括号“()”、逗号“,”和句点“.”只是用于构建合式表达式，不表示论域中的对象或关系。这些符号被称为不规范符号（improper symbol）。

谓词演算符号可以表示变量（variable）、常量（constant）、函数（function）或谓词（predi-

cate）中的任一种。常量名是世界中的特定对象和属性。常量符号必须以小写字母开始。因此 george、tree、tall 和 blue 是合式常量符号的例子。常量 true 和 false 被保留作为真值符号。

变量符号用于指代世界中的一类对象或属性。变量是用以大写字母开始的符号来表示的。因此 George、BILL 和 KAte 是合法的变量，而 geORGE 和 bill 不是合法的变量。

谓词演算还允许使用所讨论世界中对象的函数。函数符号（像常量）以小写字母开始。函数表示一个集合（称为函数的定义域（domain））中的一个或多个元素到另一个集合（称为函数的值域（range））中的一个元素的映射。定义域和值域中的元素是所讨论世界中的对象。除了加和乘这样的常见算术函数外，还可以定义非数字域间映射的函数。

请注意，我们对谓词演算符号的定义不包括数字或算术操作符。数字系统没有包含在谓词演算原语中；相反可以利用“纯”谓词演算作为基础把它们定义为公理（Manna and Waldinger 1985）。尽管这种推导的细节是很有理论研究乐趣的，但是对于把谓词演算用做 AI 的一种表示语言来说，这些推导是不太重要的。为了方便，我们采用这种推导并把算术运算包含在这种语言中。

每个函数符号有一个与之联系的元数（arity），这个元数指出被映射到值域中每个元素的定义域中的元素个数。因此 father 可能表示一个元数为 1 的函数，把人映射到他们的（惟一的）双亲中的男性。plus 可能是一个元数为 2 的函数，把两个数字映射到它们的代数和。

函数表达式（function expression）是带有参数的函数符号。参数是函数定义域中的元素；参数的数量等于函数的元数。参数用括号括起来，并用逗号分隔。例如：f(X, Y)、father(david)、price(bananas) 都是合式的函数表达式。

每一个函数表达式把参数映射到值域中的一个元素，称为函数的值。例如，如果 father 是一个元数为 1 的函数，那么 father(david) 是一个函数表达式，它的值（在笔者讨论的世界中）为 george。如果 plus 是一个元数为 2 的函数，且它的定义域为整数，那么 plus(2,3) 是一个值为整数 5 的函数表达式。把函数代换为它的值的行为叫做*求值*。

谓词演算符号或项的概念可以总结为以下的定义：

定义（*符号和项*）　*谓词演算符号包括：*

1）**真值符号** true 和 false（这是保留符号）。

2）**常量符号**是以小写字母开始的符号表达式。

3）**变量符号**是以大写字母开始的符号表达式。

4）**函数符号**是以小写字母开始的符号表达式。函数具有附带的元数用以指出被映射到值域中每一元素的定义域的元素个数。

函数表达式是由元数为 n 的一个函数常量，后跟用括号包围并用逗号分隔的 n 个参数项 $t_1, t_2, \cdots, t_n$ 组成的。

谓词演算项是常量、变量或函数表达式中的一种。

因此，谓词演算项可以用来表示问题域中的对象和属性。例如：

cat
times(2,3)
X
blue
mother(sarah)
kate

谓词演算中的符号也可以表示谓词。谓词符号像常量和函数名一样是以小写字母开始的。谓词名是世界中的0个或多个对象间的关系。这样联系起来的对象数是谓词的元数。例如以下都是谓词：

likes equals on near part_ of

原子语句（atomic sentence）是谓词演算语言中的最基本单位，它是元数为n的谓词，后跟以用括号包围并且逗号分隔的n个参数项。原子语句的例子包括：

likes(george,kate)	likes(X,george)
likes(george,susie)	likes(X,X)
likes(george,sarah,tuesday)	friends(bill,richard)
friends(bill,george)	friends(father_of(david),father_of(andrew))
helps(bill,george)	helps(richard,bill)

在这些表达式中，谓词符号是likes、friends和helps。可以给一个谓词符号提供不同数量的参数。在这个例子中，有两个不同的likes，一个有2个参数，另一个有3个参数。当语句中使用不同元数的谓词符号时，应该将其视为表示两种不同的关系。因此一个谓词关系是由它的名字和元数定义的。虽然没有理由说两个不同的likes不可以表示对世界某一部分的相同描述，但是我们要避免这样做，因为这样经常会产生混乱。

在上面的谓词中，bill、george、kate等是常量符号，表示问题域中的对象。谓词的参数是谓词项，而且也可以包含变量或函数表达式。例如：

friends(father_of(david), father_of(andrew))

是一个谓词，它描述了所讨论域中的两个对象之间的关系。这些参数被表示为函数表达式，它们的映射结果（给定father_of david是george和father_of andrew是allen）是谓词的参数。如果求值这个函数表达式，表达式就变成

friends(george, allen)

这些思想可以用以下定义表述。

定义（*谓词和原子语句*） *谓词符号是以小写字母开始的符号。*

谓词有一个与之关联的正整数，被称为谓词的元数或“参数个数”。具有相同名字但不同元数的谓词被视为是不同的谓词。

原子语句是元数为n的谓词常量，后跟以用括号包围并用逗号分隔的n个谓词项 $t_1, t_2, \cdots, t_n$。

*真值*true *和*false *也是原子语句。*

原子语句又被称为原子表达式、原子或命题。

我们可以使用逻辑操作符结合原子语句来形成谓词演算中的语句。这些逻辑连接词和命题演算中使用的逻辑连接符是一样的，它们分别是：∧、∨、¬、→和≡。

当变量是以语句中的参数形式出现时，它是指定义域中未确定的对象。一阶（见2.2.2节）谓词演算包括两个变量量词（variable quantifier）符号∀和∃，约束包含变量的语句的含义。量词出现在一个变量和一条语句之前，比如：

∃Y *friends*(Y, *peter*)

∀X likes(X, ice_cream)

全称量词（universal quantifier）∀指出其后的语句对于变量的所有值为真。例子中，∀X

likes(X, ice_cream)对于X定义域中所有值为真。*存在量词*（existential quantifier）∃，指出至少对于定义域中的一个值语句为真。对于∃Y friends(Y, peter)，如果至少存在一个Y所表示的对象是peter的friend，那么这个语句就为真。在2.2.2节中会更加详细地讨论量词。

谓词演算中的语句按以下方式归纳定义。

定义（谓词演算语句）　每一条原子语句都是一条语句。

1）如果s是一条语句，那么它的非¬ s是一条语句。

2）如果s_1和s_2是语句，那么它们的合取$s_1 \wedge s_2$是一条语句。

3）如果s_1和s_2是语句，那么它们的析取$s_1 \vee s_2$是一条语句。

4）如果s_1和s_2是语句，那么它们的蕴涵$s_1 \rightarrow s_2$是一条语句。

5）如果s_1和s_2是语句，那么它们的等价$s_1 \equiv s_2$是一条语句。

6）如果X是一个变量，s是一条语句，那么∀Xs是一条语句。

7）如果X是一个变量，s是一条语句，那么∃Xs是一条语句。

下面给出一些合式语句的例子。设times和plus是元数为2的函数符号；equal和foo分别是元数为2和3的谓词符号。

plus(two, three)是一个函数，因此它不是一条原子语句。

equal(plus(two, three), five)是一条原子语句。

equal(plus(2,3), seven)是一条原子语句。注意：如果plus和equal使用标准的解释，那么这条语句是假的。合式性和真值是两个独立的问题。

∃X foo(X, two, plus(two, three)) ∧ equal(plus(two, three), five)是一条语句，因为两个合取项都是语句。

(foo (two, two, plus(two, three)))→(equal(plus(three, two), five) ≡true)是一条语句，因为它的所有部分都是语句，并使用逻辑操作符恰当地连接起来。

刚刚给出的谓词演算语句的定义和实例指出了检验一个表达式是否是语句的一种方法。可以把这种方法写成一个递归的算法verify_sentence。verify_sentence以一个候选表达式作为参数，如果表达式是一条语句就返回success。

```
function verify_sentence(expression);
begin
  case
    expression is an atomic sentence: return SUCCESS;
    expression is of the form Q X s, where Q is either ∀ or ∃, X is a variable,
      if verify_sentence(s) returns SUCCESS
      then return SUCCESS
      else return FAIL;
    expression is of the form ¬ s:
      if verify_sentence(s) returns SUCCESS
      then return SUCCESS
      else return FAIL;
    expression is of the form s1 op s2, where op is a binary logical operator:
      if verify_sentence(s1) returns SUCCESS and
        verify_sentence(s2) returns SUCCESS
      then return SUCCESS
      else return FAIL;
    otherwise: return FAIL
  end
end.
```

我们举一个使用谓词演算来描述一个简单世界的例子结束这一小节。论域是圣经家谱中的一组家族关系：

mother(eve,abel)
mother(eve,cain)
father(adam,abel)
father(adam,cain)

∀ X ∀ Y father(X, Y) ∨ mother(X, Y) → parent(X, Y)
∀ X ∀ Y ∀ Z parent(X, Y) ∧ parent(X, Z) → sibling(Y, Z)

在这个例子中，我们使用谓词 father 和 mother 定义了一系列亲子关系。这里的蕴涵语句利用这些谓词给出了其他关系（比如 parent 和 sibling）的一般定义。很明显，可以使用这些蕴涵推理出像 sibling(cain, abel) 这样的事实。为了用公式表示这个过程使其可以在计算机中进行，必须小心定义推理算法，而且要保证这样的算法确实可以从一系列谓词演算断言中得出正确的结论。为了做到这一点，我们先定义谓词演算的语义（见 2.2.2 节），然后再讨论推理规则（见 2.3 节）。

2.2.2 谓词演算的语义

在定义谓词演算的合式表达式之后，重要的是用世界中的对象、属性和关系确定它们所表示的含义。谓词演算语义提供了决定合式表达式真值的形式基础。表达式的真值依赖于常量、变量、谓词和函数讨论域中的对象和关系的映射。论域中关系的真值决定了对应表达式的真值。

例如，关于一个人 George 和他的朋友 Kate 和 Susie 的信息可以表示为

friends(george, susie)
friends(george, kate)

如果 George 是 Susie 的朋友而且 George 是 Kate 的朋友这些确实都是真的，那么这两个表达式每一个的真值（赋值）都是 T。如果 George 是 Susie 的朋友而不是 Kate 的朋友，那么第一个表达式的赋值是 T，第二个的赋值是 F。

为了使用谓词演算作为问题求解的一种表示，我们用一系列合式表达式来描述解释域中的对象和关系。这些表达式的项和谓词表示了域中的对象和关系。谓词演算表达式（每个的真值都是 T）的这种数据库描述了“世界的状态”。对 George 和他的朋友的描述是这样的数据库的一个简单例子。另一个例子是在第二部分的简介中给出的积木世界的例子。

基于这些直觉的认识，下面我们形式地定义谓词演算的语义。首先我们定义在论域 D 上的解释。然后使用这种解释确定语言中语句的真值赋值（truth value assignment）。

定义（解释） 设论域 D 是非空的集合。

D 上的**解释**就是 D 中的实体对谓词演算表达式的每个常量、变量、谓词和函数符号的指派。具体来说：

1）把每个常量指派为 D 中的一个元素。

2）把每个变量指派为 D 中的一个非空子集；这些是对变量允许进行的置换。

3）把每个元数为 m 的函数 f 定义在 D 的 m 个参量之上并定义一种从 D^m 到 D 的映射。

4）把每个元数为 n 的谓词 p 定义在 D 的 n 个参量之上并定义一种从 D^n 到{T, F}的映射。

给定一种解释，表达式的含义就是对解释所赋的真值。

定义（谓词演算表达式的真值）　假定 E 是一个表达式，I 是 E 在非空定义域 D 上的解释。E 的真值由以下方式决定：

1）常量的值是 I 所指派的 D 的元素。

2）变量的值是 I 所指派的 D 的元素集合。

3）函数表达式的值是 D 的一个元素，这个元素通过对由解释 I 所指派的参数值求值函数的方式获得。

4）真值符号“true”的值是 T，“false”的值是 F。

5）原子语句的值要么是 T，要么是 F，这是由解释 I 决定的。

6）如果语句的值是 F，那么它的非的值是 T；如果语句的值是 T，那么它的非的值是 F。

7）如果两个语句的值都是 T，那么它们的合取的值是 T，否则是 F。

8～10）使用 $\vee$、$\rightarrow$和 $\equiv$ 的表达式的真值是根据它们的操作数的值按 2.1.2 节中的定义决定的。

最后，对于变量 X 和含有 X 的语句 S：

11）如果在解释 I 之下对 X 的所有赋值 S 都是 T，那么 $\forall$XS 的值是 T，否则是 F。

12）如果在解释中存在一个对 X 的赋值使 S 的值为 T，那么 $\exists$XS 的值是 T，否则是 F。

变量的量化是谓词演算语义的一个重要部分。当一个变量出现在语句中时，比如 X 出现在 likes(george, X) 中时，变量的作用只是占位符。可以用解释所允许的任何常量来替换表达式中的变量。把 likes(george, X) 中的 X 替换为 kate 或 susie，便形成了之前我们看到的陈述 likes（george，kate）和 likes（george，susie）。

变量 X 代表了可以出现在语句的第二个参数位置的所有常量。这个变量名可以替换为任何一个其他的变量名，比如 Y 或 PEOPLE，而且不会改变表达式的含义。由于这个原因，这种变量被称为哑元（dummy）。在谓词演算中，量化变量的方式有以下两种：全称量化和存在量化。如果一个变量集既不在全称量词内又不在存在量词内，那么它被认为是自由的。如果表达式的所有变量都是量化了的，那么它被认为是封闭的。一个基表达式（ground expression）根本不存在变量。在谓词演算中，所有的变量都必须被量化。

经常使用括号来表示量化的范围，也就是量化所控制的一个变量名的多个实例。因此对于表明全称量化的符号 $\forall$：

$$\forall X\,(p(X) \vee q(Y) \rightarrow r(X))$$

表示在 p(X) 和 r(X) 中 X 都是全称量化的。

全称量化给计算语句的真值带来了问题，因为必须测试变量符号的所有可能值以观察表达式是否始终为真。例如，为了测试表达式 $\forall$X likes(george, X) 的真值，其中 X 的范围是所有人的集合，必须测试 X 的所有可能值。如果解释的域是无限的，那么穷举式地测试一个全称量化变量的所有替换在计算方式上是不可能的：算法也许永远不会停止。由于这个问题，这个谓词演算被称为是不可判断的（undecidable）。由于命题演算不使用变量，因此语句仅有有限数量的可能真值赋值，从而我们可以穷举式地测试所有可能的赋值。这是真值表所要完成的工作（见 2.1 节）。

如 2.2.1 节所述，变量也可以被存在量化，用符号 $\exists$ 表明存在量词。在这种情况下，包含变量的表达式至少对于来自定义域的一个替换是真的。被存在量化的变量的范围也是通过把变量的量化取值包围在括号中来表明的。

求解一个含有存在量化变量的表达式的真值可能不比求解含有全称量化变量的表达式简单。

假定我们试图决定这种表达式的真值，方法是用实例替代变量直到找到使表达式为真的一个替换。如果变量的定义域是无限的，而且对于所有替换表达式都是假的，那么这个算法永远不会结束。

下面给出了全称量词和存在量词与否定词之间的几种关系。这些关系将用于第 14 章中介绍的归结反驳（resolution refutation）系统中。还该注意用变量名作为哑元符号来代表一个常量集合的思想。对于谓词 p 和 q 以及变量 X 和 Y：

$\neg \exists X\, p(X) \equiv \forall X \neg p(X)$

$\neg \forall X\, p(X) \equiv \exists X \neg p(X)$

$\exists X\, p(X) \equiv \exists Y\, p(Y)$

$\forall X\, q(X) \equiv \forall Y\, q(Y)$

$\forall X\, (p(X) \wedge q(X)) \equiv \forall X\, p(X) \wedge \forall Y\, q(Y)$

$\exists X\, (p(X) \vee q(X)) \equiv \exists X\, p(X) \vee \exists Y\, q(Y)$

在我们已经定义的语言中，全称量化变量和存在量化变量仅可以指论域中的对象（常量）。谓词和函数名不可以被量化变量所替换。这种语言被称为一阶谓词演算。

定义（一阶谓词演算） **一阶谓词演算**（first-order predicate calculus）允许量化变量指向论域中的对象，不允许指向谓词或函数。

例如，

$\forall$ (Likes) Likes(george, kate)

不是一阶谓词演算中的合式表达式。在高阶谓词演算中，这样的表达式是有意义的。一些研究人员（McCarthy 1968，Appelt 1985）已经使用高阶语言来表示自然语言理解程序中的知识。

使用一阶谓词演算中的符号、连接词和变量符号可以表示很多语法正确的英文语句。必须注意：从句子到谓词演算表达式的映射是不惟一的；实际上，一个英文语句可以有任意数量的不同谓词演算表示。AI 程序员的一个主要难题就是寻找一种使用这些谓词的表示模式，使最终表示的表现力和效率最优。以下是一些用谓词演算来表示英文语句的例子：

如果星期一不下雨，Tom 会去登山。

$\neg$ weather(rain, monday) $\rightarrow$ go(tom, mountains)

Emma 是一只德国种的短毛猎犬，而且是一只好犬。

gooddog(emma) $\wedge$ isa(emma, doberman)

所有篮球运动员都很高。

$\forall X$ (basketball_player(X) $\rightarrow$ tall(X))

一些人喜欢凤尾鱼。

$\exists X$ (person(X) $\wedge$ likes(X, anchovies))

如果愿望都能实现，乞丐早就发财了；愿望不能代替实际。

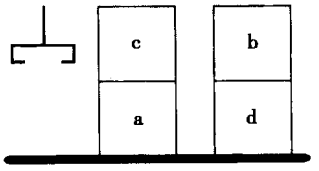
equal(wishes, horses) $\rightarrow$ ride(beggars)

没人喜欢纳税。

$\neg \exists X$ likes(X, taxes)

2.2.3　语义含义的积木世界例子

我们通过一个为一系列谓词演算表达式赋真值的例子来结束这一节的内容。假定我们想为图2-3中的积木世界建模，比如说设计一个机械手的控制算法。我们可以使用谓词演算语句来表示这个世界中的定性关系：一个给定积木的上表面是否为空？我们是否可以拿起积木a？等等。假定计算机掌握了每块积木和机械手位置的知识，并且当机械手在桌面上把积木移来移去时能够跟踪这些位置（使用三维坐标）。



on(c,a)
on(b,d)
ontable(a)
ontable(d)
clear(b)
clear(c)
hand_empty

图2-3　带有谓词演算描述的一个积木世界

我们必须非常明确我们打算在这个“积木世界”的例子中做些什么。首先，我们创建一系列谓词演算表达式来表示这个“积木世界”问题域的静态快照。我们将在第2.3节中看到，这个积木的集合为这一系列谓词演算表达式提供了一种解释和一个可能的模型（model）。

其次，谓词演算是说明性的（declarative），也就是说，考虑每个表达式时不假定计时或顺序。虽然如此，但是在本书第8.4节的对规划方法的讨论中，我们将加入“过程语义”，也就是相对时间对这些表达式求值的特定方法。谓词演算表达式的过程语义的具体事例之一就是14.3节中的Prolog。我们现在创建的这种情景演算会有很多问题，包括框架问题（frame problem）和逻辑表示的不单一（non-monotonicity）问题，这些都是本书后面章节要论述的问题。然而对于这个例子，只要知道我们的谓词演算表达式是按从上到下、从左到右的方式求解就足够了。

要搬起一块积木并把它堆在另一块积木上面，那么两块积木的上表面都必须是空的。在图2-3中，积木a不是空的。因为机械手可以搬动积木，所以它可以改变这个世界的状态，使一块积木的上表面为空。假若它把积木c从a上移去并更新知识库来反映这个变化，也就是删除断言on(c, a)。程序应该能够推理出a的上表面已经变为空。

下面的规则描述了积木上表面为空的状态：

$$\forall X (\neg \exists Y \, on(Y,X) \rightarrow clear(X))$$

也就是说，对于所有X，如果不存在一个Y在X上，那么X的上表面就是空的。

这个规则不仅定义了积木为空的含义，而且为决定如何清理不为空的积木提供了基础。例如，积木d不为空，因为如果变量X的值为d，那么用b替换Y会使上面规则的结果为假。所以，要使这个定义为真，必须把积木b从d上移开。因为计算机记录了所有积木和它们的位置，所以这是容易完成的。

除了使用蕴涵来定义积木为空的状态外，还可以加入其他的规则来描述像堆一块积木到另一块积木上这样的操作。例如，要把X堆在Y上，先空出机械手，然后清理X，然后清理Y，再搬起（pick-up）X并放到（put-down）Y上。

$$\forall X \forall Y((hand\_empty \wedge clear(X) \wedge clear(Y) \wedge pick\_up(X) \wedge put\_down(X,Y)) \rightarrow stack(X,Y))$$

注意：在实现上面的描述时，有必要为每个像pick_up(X)这样的谓词“附加”一个机械手动作。正像刚才所指出的，对于这样的实现必须扩充这个谓词演算的语义，指出需要精确地按规则中的出现顺序来执行动作。然而，把这些问题从使用谓词演算来定义论域中的关系和操作中分离出来是很有好处的。

图2-3中给出了对这些谓词演算表达式的一种语义解释。这个解释把表达式中的常量和谓词映射到论域D，也就是这里的积木和它们之间的关系。解释将真值T赋值给描述中的每个表达式。还可以给出其他的解释，比如另一位置的不同积木集合，或者或许用一个4人的杂技队也可以。重要的问题不是解释的惟一性，而是解释是否可以给出所有表达式的真值，以及这些表达式能否足够细致地描述这个世界，从而可以通过操纵这些符号表达式进行所有必要的推理。下一节使用这些思想来给出谓词演算推理规则的一个正式基础。

2.3 使用推理规则产生谓词演算表达式

2.3.1 推理规则

谓词演算的语义为*逻辑推理*（logical inference）的形式理论提供了基础。从一系列真实断言推理出新的正确表达式的能力是谓词演算的一个重要功能。衡量这些新表达式正确的标准是它们与对原始表达式集合的所有以前解释是*一致的*（consistent）。我们先非形式化地讨论这些思想，然后再创建一些定义使其精确化。

如果一种解释使一条语句为真，那么就说它*满足*这条语句。如果一个解释满足表达式集合中的所有成员，那么就说它满足这个集合。如果满足谓词演算表达式集合S的所有解释也满足表达式X，那么就说X*逻辑派生*（logically follow）S。这个概念为验证推理规则的正确性提供了基础；因为逻辑推理的功能就是要产生逻辑派生一个给定的谓词演算语句集合的新语句。理解*逻辑派生*的准确含义是很重要的：对于表达式X，如果它要*逻辑派生*S，那么它必须对于满足原始表达式集合S的所有解释都是真的。举例来说，这意味着，加入到图2-3的积木世界的新的谓词演算表达式必须对那个世界是真的，而且对于那个表达式集合可能有的其他解释也是真的。

“逻辑派生”这个术语本身可能有点令人误解。它并不意味X是从S演绎出的，或可以从S演绎。它仅意味着对于满足S的所有解释（可能无限多）X都是真的。然而。因为谓词系统可能潜在具有无限数量的可能解释，所以试验所有的解释是不太可行的。相反，*推理规则*（inference rule）提供了计算上可行的方法，来决定一个表达式（解释的一个部分）何时逻辑派生那个解释。“逻辑派生”的概念为证明推理规则的合理性和正确性提供了形式基础。

推理规则在本质上是一种从其他语句产生新的谓词演算语句的机械方式。换句话来说，推理规则是基于给定逻辑断言的语法形式来产生新的语句。当推理规则从逻辑表达式集合S产生的每一条语句都逻辑派生自S时，我们就说这个推理规则是*可靠的*。

如果推理规则能够产生所有从S逻辑派生的表达式，那么就说这个推理规则是*完备的*。下面要介绍的*假言推理*（modus ponens）和第11章中要介绍的*归结*（resolution）是可靠的推理规则的例子，而且当与合适的应用策略一起使用时它们也是完备的。虽然后面几章（第4、9和15章）中讨论启发式推理和常识推理，但是逻辑推理系统通常都使用可靠的推理规则，启发式推理和常识推理中任何一个均可以满足需求。

下面把这些思想形式化表达为如下的定义。

定义（*满足、模型、有效、不一致*） *对一个谓词演算表达式X和一种解释I：*

*如果在I下存在一个特定的变量赋值使X的值为T，那么就说I*__满足__*X。*

*如果对于所有的变量赋值来说I都满足X，那么就说I是X的一个*__模型__（model）。

X*是*__可满足的__（satisfiable），*当且仅当存在一种满足它的解释和变量赋值；否则它是*__不可满__

足的（unsatisfiable）。

一个表达式集合是**可满足的**，当且仅当存在满足所有元素的一种解释和变量赋值。

如果表达式集合不是可满足的，那么就说它是**不一致的**（inconsistent）。

如果对于所有的可能解释来说 X 的值都为 T，那么就说 X 是**有效的**（valid）。

在图 2-3 积木世界的例子中，积木世界是揭示它的逻辑描述的一个模型。在这个解释下，例子中的所有语句都是真的。当把一个知识库实现为一系列问题域的真实断言时，这个域是这个知识库的一个模型。

表达式 $\exists X (p(X) \wedge \neg P(X))$ 是不一致的，因为在任何解释或变量赋值下它都不能被满足。而表达式 $\forall X (p(X) \vee \neg P(X))$ 是有效的。

可以使用真值表来测试任何不包含变量的表达式的有效性。因为并不总是可以决定含有变量表达式的有效性（如上面所提到的，这个过程可能永不终止），所以完全的谓词演算是“不可决定的”。然而，存在可以产生逻辑派生自一个表达式集合的任何表达式的证明过程。这被称为完备证明过程。

定义（证明过程）　**证明过程**（proof procedure）是推理规则和应用这个规则到逻辑表达式的集合以产生新的语句算法的结合。

我们将在第 11 章中给出**归结**推理规则的证明过程。

使用这些定义，我们可以形式化地定义“逻辑派生”。

定义（逻辑派生、可靠的和完备的）

如果满足谓词演算表达式集合 S 的所有解释也满足谓词演算表达式 X，那么 X **逻辑派生**自 S。

如果一个来自谓词演算表达式集合 S 的推理规则产生的所有谓词演算表达式也逻辑派生自 S，那么就说这个推理规则是**可靠的**。

如果给定一个谓词演算表达式的集合 S，一个推理规则可以推理出逻辑派生自 S 的所有表达式，那么就说这个推理规则是**完备的**。

取式假言推理是一种正确的推理规则。如果给定一个形式为 $P \rightarrow Q$ 的表达式和另一个形式为 P 的表达式，使得在一种解释 I 下它们都是真的，那么取式假言推理允许我们推理出对于这个解释来说 Q 也是真的。事实上，因为取式假言推理是可靠的，所以对于所有的 P 和 $P \rightarrow Q$ 都为真的解释，Q 也是真的。

取式假言推理和其他一些有价值的推理规则按如下方式定义。

定义（取式假言推理、拒式假言推理、与消除、与引入、全称例化）

如果已知语句 P 和 $P \rightarrow Q$ 是真的，那么根据**取式假言推理**可以推断 Q 为真。

在推理规则的**拒式假言推理**（modus tollens）内，如果已知 $P \rightarrow Q$ 是真的，并且 Q 是假的，那么可以推断 P 是假的：$\neg P$ 为真。

与消除（and elimination）使我们可以根据合取语句的真值来推断任一个合取项的真值。例如，从 $P \wedge Q$ 为真可以推出 P 和 Q 都是真的。

与引入（and introduction）使我们可以从合取项的真值来推断合取的真值。例如，如果 P 和 Q 都是真的，那么 $P \wedge Q$ 是真的。

全称例化（universal instantiation）指出如果可以把一个真语句，比方说 p(X)，中的任意的全称量化变量替换为定义域中的任意的适当项，那么结果是一个真的语句。因此，如果 a 来自 X

的定义域，那么从∀X p(X)可以推断p(a)。

下面举一个简单的例子来说明在命题演算中取式假言推理的用法。假定以下的观察结果："如果在下雨，那么地面会是湿的"和"在下雨"。如果P表示"在下雨"，Q表示"地面是湿的"，那么第一个表达式变成P→Q。因为现在确实在下雨（P是真的），所以我们的公理集变成

P→Q
P

通过应用取式假言推理，地面是湿的（Q）这个事实可以被加入到真的表达式的集合中。

也可以把取式假言推理应用到含有变量的表达式中。以常见的三段论为例，"所有人都会死，苏格拉底是人；所以苏格拉底会死。"可以用谓词演算把"所有人都会死"表示为：

∀X (man(X)→ mortal(X))

而"苏格拉底是人"也可以用谓词演算表示为：

man(socrates)

因为在蕴涵中X是全称量化的，所以我们可以用定义域中的任何值来替换X，而且根据全称例化的推理规则仍会得到真的陈述。把蕴涵中的X替换为socrates，便可推断出如下表达式：

man(socrates)→ mortal(socrates)

现在我们可以应用取式假言推理并推断出结论mortal(socrates)。这被加入到逻辑派生自原始断言的表达式集合中。一种称为合一（unification）的算法可被自动问题求解器用来决定取式假言推理可以用socrates替换X。我们在2.3.2节中讨论这种合一算法。

第14章中还讨论了一种更为强大的推理规则，称为*归结*，它是很多自动推理系统的基础。

2.3.2 合一算法

为了应用推理规则（比如取式假言推理），推理系统必须能够判断两个表达式何时相同，也就是这两个表达式何时匹配。在命题演算中，这是显而易见的：两个表达式是匹配的当且仅当它们在语句构成上相同。在谓词演算中，表达式中变量的存在使匹配两个语句的过程变得复杂。全称例化允许用定义域中的项来替换全称量化变量。这需要一个决策处理来判断是否可以使变量替换产生的两个或更多个表达式相同（通常是为了应用推理规则）。

合一是一种判断什么样的替换可以使产生的两个谓词演算表达式匹配的算法。我们在上一节中已经看到了这个过程，∀X(man(X)⇒mortal(X))中的X替换成了man(socrates)中的socrates。合一的另一个例子是在前面讨论哑元时看到的。因为p(X)和p(Y)是等价的，所以可以用Y替换X使语句匹配。

合一和像假言推理这样的推理规则允许我们对一系列逻辑断言做出推理。为了做到这一点，必须把逻辑数据库表示为合适的形式。

这种形式的一个根本特征是要求所有的变量都是全称量化的。这样便允许在计算替代时有完全的自由度。存在量化变量可以从数据库语句中消除，方法是用使这个语句为真的常量来替代它们。例如，可以把∃X parent(X, tom)替代为表达式parent(bob, tom)或parent(mary, tom)，假定在当前解释下bob和mary是tom的双亲。

消除存在量化变量的处理会因这些替换的值可能依赖于表达式中的其他变量而变得复杂。

例如，在表达式 ∀X ∃Y mother(X, Y) 中，存在量化变量 Y 的值依赖于变量 X 的值。斯柯伦化（skolemization）方法把每个存在量化变量用一个函数来替换，这个函数是关于语句中某个或所有其他变量的函数，它的返回值是一个常量。在上面的例子中，因为 Y 依赖于 X，所以可以用一个关于 X 的斯柯伦函数 f 来替换 Y。这样便得到了谓词 ∀X mother(X, f(X))。斯柯伦化处理还可把全称量化变量绑定到常量，在 14.2 节中我们会更加详细地讨论这种处理。

一旦消除了逻辑数据库中的存在量化变量，就可以使用合一来匹配语句以便应用像假言推理这样的推理规则。

合一会由于一个变量可以替换为任何项（包括其他变量和任意复杂度的函数表达式）而变得复杂。这些表达式本身可能含有变量。例如，可以用 father(jack) 来替换 man(X) 中的 X 推断出 jack 的父亲是会死的。

下面给出了对表达式 foo(X, a, goo(Y)) 的合法替换所产生的一些实例：

1）foo(fred, a, goo(Z))

2）foo(W, a, goo(jack))

3）foo(Z, a, goo(moo(Z)))

在这个例子中，使初始的表达式和其他三个表达式中的每一个都恒等的替换实例或合一写为如下集合：

1）{fred/X, Z/Y}

2）{W/X, jack/Y}

3）{Z/X, moo(Z)/Y}

记号 X/Y 表示用 X 替换原始表达式中的变量 Y。替换又称为绑定（binding）。我们可以说：变量被绑定到替换它的值上。

在定义用于计算使两个表达式匹配所需的替换的合一算法时，必须考虑很多问题。首先，尽管常量可以系统地替代变量，但是任何常量都被视为一个“基例”（ground instance），不可以被替代。两个不同的基例也不可以被一个变量替代。其次，一个变量不能与含有这个变量的项合一。X 不可以被 p(X) 替代，因为这样会产生无限的表达式：p(p(p(p(…X)…)))。对这种情况的检测被称为重现检查（occurs check）。

进一步讲，问题求解过程经常需要多重推理，从而必然需要多个连续的合一处理。逻辑问题求解器必须维护变量置换的一致性。保证被匹配的两个表达式中变量作用域内出现的所有变量都进行一致的合一替换是非常重要的。在前面的例子中我们已经看到了这一点，当替换 socrates 时，不仅替换了 man(X) 中的 X，而且替换了 mortal(X) 中的 X。

一旦一个变量已经被绑定，将来的合一和推理必须把这个绑定的值考虑进来。如果一个变量被绑定到一个常量，那么在以后的合一处理中不可以再对这个变量进行新的绑定。如果变量 X_1 被替代为变量 X_2，而且后来 X_1 又被替代为常量，那么 X_2 也应该反映这个绑定。用在推理序列中的替换集合是很重要的，因为它可能包含原始查询的答案（见 14.2.5 节）。例如，如果通过替换{a/Y, X/Z}用 p(Y, Z)⇒q(Y, Z) 的前提来合一 p(a, X)，那么取式假言推理使我们可以在相同替换下推断 q(a, X)。如果我们把这个结果和 q(W, b)⇒r(W, b) 匹配，那么通过替换集合{a/W, b/X}，我们可以推断 r(a, b)。

另一个合一替换的重要概念是复合（composition）。如果 S 和 S′是两个替换集合，那么 S 和 S′的复合（写为 SS′）是这样得到的：对 S 的每一个元素应用 S′，再把结果加到 S。例如考虑复合下面这个替换序列：

{X/Y, W/Z}, {V/X}, {a/V, f(b)/W}

将第三个集合{a/V, f(b)/W}与第二个集合{V/X}复合，得到：

{a/X, a/V, f(b)/W}

将结果与第一个集合{X/Y, W/Z}复合，得到替换集合：

{a/Y, a/X, a/V, f(b)/Z, f(b)/W}

复合是下面要给出的一个递归函数 unify 所使用的一种方法，该函数用复合这种方法来组合多个合一置换并作为结果返回。可以证明复合是支持结合律但不支持交换律的。本章的练习更详细地讨论了这些问题。

对合一算法的进一步要求是合一式要尽可能地通用：也就是要发现两个表达式的最一般的合一式（most general unifier）。在下面的例子中会看到这一点的重要性，因为如果求解过程失去了一般性，就会缩小最终解的适用范围，或者甚至完全排除解的可能性。

例如，在合一 p(X) 和 p(Y) 时，任何常量表达式，比如{fred/X, fred/Y}，都会获得成功。然而，fred 不是最一般的合一式；可以使用任何变量，而且会产生更一般的表达式：{Z/X, Z/Y}。第一种置换得到的解总是受约束的，常量 fred 限制了最终的推理；也就是说，fred 是一个合一式，但它降低了结果的一般性。

定义（最一般的合一式（mgu））　*如果 s 是表达式 E 的任意一个合一式，g 是这个表达式集合的最一般的合一式，那么对于应用到 E 的 s，存在另一个合一式 s′使 Es = Egs′，其中 Es 和 Egs′是应用到表达式 E 的合一式的复合。*

一个表达式集合的最一般的合一式是惟一的，除非字母变种（alphabetic variation）；也就是说，不论变量最终被称为 X 还是 Y，实际上对最终合一的一般性没有任何不同。

合一对于任何使用谓词演算作为表示的人工智能问题求解器都是很重要的。合一确定了两个（或更多个）谓词演算表达式等价的条件。这允许我们通过逻辑表示来使用像归结这样的推理规则，归结是经常需要回溯以找到所有可能解释的一种过程。

下面我们给出函数 unify 的伪代码，它计算两个谓词演算表达式之间的合一替换（当可能合一时）。unify 函数以两个谓词演算表达式为参数，返回最一般的合一替换或常量 FAIL（当不可能合一时）。它被定义为一个递归的函数：首先，它试图递归合一表达式的初始部分。如果这成功了，再把这个合一返回的任何替换应用到两个表达式的其余部分。然后把应用后的结果作为参数传进第二个对 unify 的递归调用，并尝试结束这个合一过程。当遇到以下两种情况中的任一种时递归会停止：参数是一个符号（一个谓词、函数名、常量或变量）；表达式的元素都已经被匹配过。

为了简化对表达式的操作，这个算法假定对语法进行了轻微的修改。因为 unify 仅进行语法模式的匹配，所以它可以忽略谓词、函数和参数之间的谓词演算差异，使算法更加有效。通过把表达式表示成一个列表（元素的一个有序序列）——谓词或函数名作为第一个元素，参数跟在后面——我们可以简化表达式的操作。在表达式中，如果一个参数本身是一个谓词或函数表达式，那么这种表达式可以表示成列表中的列表，从而保持表达式的结构。各个列表间用括号（）分界，列表的元素之间用空格分隔。下面的例子列出了一些表达式的谓词演算（PC）形式和相应的符合列表语法的形式。

PC 语法	列表语法
p(a,b)	(p a b)
p(f(a),g(X,Y))	(p (f a) (g X Y))
equal(eve,mother(cain))	(equal eve (mother cain))

下面给出了函数 unify：

```
function unify(E1, E2);
  begin
    case
      both E1 and E2 are constants or the empty list:          %recursion stops
        if E1 = E2 then return {}
          else return FAIL;
      E1 is a variable:
        if E1 occurs in E2 then return FAIL
          else return {E2/E1};
      E2 is a variable:
        if E2 occurs in E1 then return FAIL
          else return {E1/E2}
      either E1 or E2 are empty then return FAIL      %the lists are of different sizes
      otherwise:                                        %both E1 and E2 are lists
        begin
            HE1 := first element of E1;
            HE2 := first element of E2;
            SUBS1 := unify(HE1,HE2);
            if SUBS1= FAIL then return FAIL;
            TE1 := apply(SUBS1, rest of E1);
            TE2 := apply (SUBS1, rest of E2);
            SUBS2 :=unify(TE1, TE2);
            if SUBS2 = FAIL then return FAIL;
                else return composition(SUBS1,SUBS2)
        end
    end                                                  %end case
  end
```

2.3.3　合一的例子

可以通过跟踪下面的调用来分析前面的算法的行为：

unify((parents X (father X) (mother bill)), (parents bill (father bill) Y))

当第一次调用 unify 时，因为两个参数都不是原子符号，所以函数试图递归地合一每个表达式的第一个元素，即调用

unify(parents, parents)

这个合一是成功的，返回空的替换 {}。把这个替换应用到表达式的其余部分没有产生变化，因此接下来算法调用

unify((X (father X) (mother bill)), (bill (father bill) Y))

这一阶段的执行情况以树的形式显示在图 2-4 中。

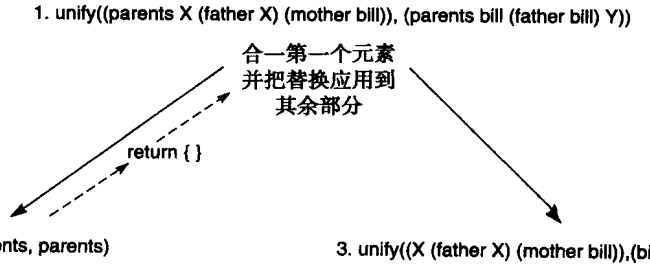


图 2-4　（parents X（father X）（mother bill））与（parents bill（father bill）Y）合一的初始步骤

在第二次调用 unify 时，两个表达式都不是原子的，所以算法把每个表达式分割成它的第一个元素和其余部分。这导致如下的调用：

unify(X, bill)

这个调用是成功的，因为两个表达式都是原子的，而且其中之一是一个变量。这个调用返回的替换是{bill/X}。这一替换被应用到每个表达式的其余部分，然后对应用后的结果再调用 unify（如图 2-5 所示）：

unify(((father bill)(mother bill)),((father bill) Y))

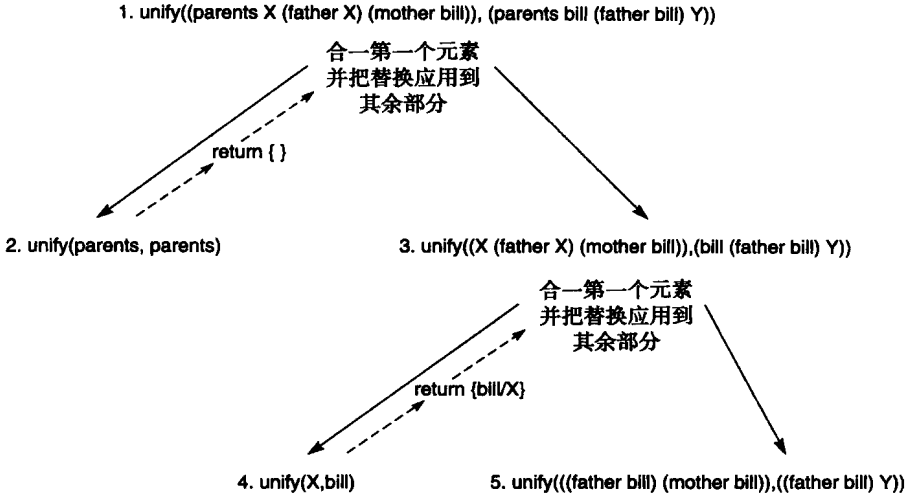


图 2-5 (parents X (father X)(mother bill))与(parents bill (father bill) Y)合一的其他步骤

这次调用的结果是要使（father bill）和（father bill）合一。这导致如下的调用

unify(father, father)
unify(bill, bill)
unify((),())

所有这些调用都是成功的，返回空的替换集合，如图 2-6 所示。然后对表达式的其余部分调用unify：

unify(((mother bill)),(Y))

这依次产生如下调用：

unify((mother bill), Y)
unify((),())

在上面的第一个调用中，(mother bill) 和 Y 合一。注意：这个合一是用整个结构（mother bill）代替变量 Y。因此，合一成功并返回一个替换{(mother bill)/Y}。第二个调用 unify((),()) 返回{}。所有这些替换被组合在一起作为每次递归调用的结果，返回答案{bill/X(mother bill)/Y}。整个执行过程显示在图 2-6 中。图中用数字标出了每个调用以便表明调用的次序；每个调用返回的替换标记在树中的弧上。

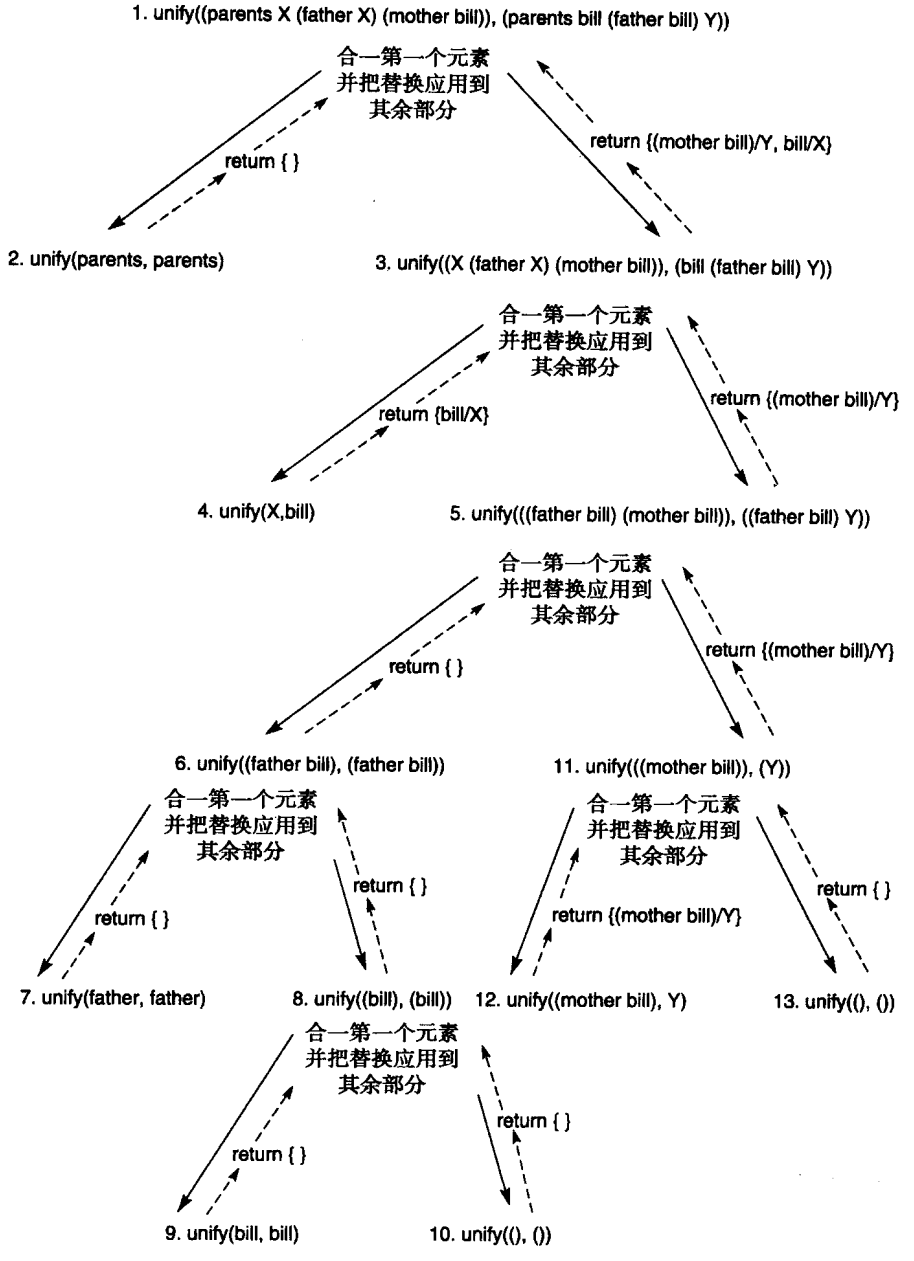


图 2-6　(parents X (father X) (mother bill)) 与 (parents bill (father bill) Y) 合一的最终图示

2.4　应用：一个基于逻辑的财务顾问

作为使用谓词演算来表示和推理问题域的最后一个例子，我们利用谓词演算设计一个简单的财务顾问。尽管这个例子很简单，但它演示了现实应用中涉及的很多问题。

这个顾问的功能是帮助用户决策是应该向存款账户中投资还是向股票市场中投资。一些投资者可能想要把他们的钱在这两者之间分摊。推荐给每个投资个体的投资策略依赖于他们的收入和他们已有存款的数量，需根据以下标准制定：

1）存款数额还不充足的个体始终该把提高存款数额作为他们的首选目标，无论他们收入

如何。

2）具有充足存款和充足收入的个体应该考虑风险较高但潜在投资收益也更高的股票市场。

3）收入较低的已经有充足存款的个体可以考虑把他们的剩余收入在存款和股票间分摊，以便既能提高存款数额又能尝试通过股票提高收入。

存款和收入的充足性可以由个体要供养的人数决定。我们的标准是为供养的每个人至少在银行存款 5 000 美元。充足的收入必须是稳定的所得，每年至少补充 15 000 美元，再加额外的给每个要供养的人 4 000 美元。

为了自动化这个咨询过程，我们把这些准则翻译成以谓词演算表示的语句。第一个任务是决定必须要考虑的主要特征。这里是存款和收入的充足性。把它们分别用谓词 savings_account 和 income 表示。它们都是一元谓词，它们的参数是 adequate（充足）或 inadequate（不充足）。于是，

savings_account(adequate)
savings_account(inadequate)
income(adequate)
income(inadequate)

是它们的可能值。

结论是用一元谓词 investment 表示的，它的参数的可能值是 stocks（股票）、savings（存款）或 combination（组合）（意味应该分摊投资）。

利用这些谓词，不同的投资策略被表示为蕴涵的形式。第一条规则，存款数额还不充足的个体始终该把提高存款数额作为他们的首选目标，被表示为：

$$\text{savings\_account(inadequate)} \rightarrow \text{investment(savings)}$$

类似地，剩下的两条备选投资方案被表示为：

$$\text{savings\_account(adequate)} \wedge \text{income(adequate)} \rightarrow \text{investment(stocks)}$$
$$\text{savings\_account(adequate)} \wedge \text{income(inadequate)} \rightarrow \text{investment(combination)}$$

接下来，这个顾问必须判断存款和收入何时是充足的以及何时是不充足的。这也可以使用蕴涵来实现。算术计算的需要要求使用函数。为了计算充足存款的最小值，定义了函数 minsavings。minsavings 具有一个参数，即要供养的人数，并且返回该参数的 5000 倍。

利用 minsavings，存款的充足性可以由以下规则来判断：

$$\forall X\ \text{amount\_saved(X)} \wedge \exists Y(\text{dependents(Y)} \wedge \text{greater(X, minsavings(Y))}) \rightarrow \text{savings\_account(adequate)}$$

$$\forall X\ \text{amount\_saved(X)} \wedge \exists Y\ (\text{dependents(Y)} \wedge \neg\ \text{greater(X, minsavings(Y))}) \rightarrow \text{savings-account(inadequate)}$$

其中：$\text{minsavings(X)} \equiv 5000 * X$。

在这些定义中，amount_saved(X) 和 dependents(Y) 断言投资者的当前存款额度和要供养的人数；greater(X, Y) 是标准的算术测试，用来测试一个数字是否大于另一个数字，在这个例子中我们没有正式定义它。

类似地，函数 minincome（最低收入）定义为：

$$\text{minincome(X)} \equiv 15000 + (4000\ *\ X)$$

当给定要供养的人数后，使用 minincome 计算充足收入的最小值。投资者的当前收入被表示为一个谓词 earnings。因为充足的收入必须既是稳定的又要超过最小值，所以 earnings 带两个参数，第一个参数是所得数额，第二个参数必须等于 steady（稳定）或 unsteady（不稳定）中的一个。这个顾问需要的其他规则是：

∀X earnings(X, steady) ∧ ∃Y (dependents(Y) ∧ greater(X, minincome(Y))) → income (adequate)

∀X earnings(X, steady) ∧ ∃Y (dependents(Y) ∧ ¬ greater(X, minincome(Y))) → income (inadequate)

∀X earnings(X, unsteady) →income(inadequate)

为了进行一次咨询，要使用谓词 amount-saved、earnings 和 dependents 把一个特定投资个体的描述加入到谓词演算的语句集合中。于是，一个要供养 3 个人、有 22 000 美元存款、25 000 美元稳定收入的投资者被描述为：

amount_saved(22000)
earnings(25000, steady)
dependents(3)

这样得到了由以下语句组成的一个逻辑系统：

1. savings_account(inadequate) → investment(savings)
2. savings_account(adequate) ∧ income(adequate) → investment(stocks)
3. savings_account(adequate) ∧ income(inadequate) → investment(combination)
4. ∀X amount_saved(X) ∧ ∃Y (dependents(Y) ∧ greater(X, minsavings(Y))) → savings_account(adequate)
5. ∀X amount_saved(X) ∧ ∃Y (dependents(Y) ∧ ¬ greater(X, minsavings(Y))) → savings_account(inadequate)
6. ∀X earnings(X, steady) ∧ ∃Y (dependents (Y) ∧ greater(X, minincome(Y))) → income(adequate)
7. ∀X earnings(X, steady) ∧ ∃Y (dependents(Y) ∧ ¬ greater(X, minincome(Y))) → income(inadequate)
8. ∀X earnings(X, unsteady) → income(inadequate)
9. amount_saved(22000)
10. earnings(25000, steady)
11. dependents(3)

其中：minsavings(X) ≡ 5000 * X, minincome(X) ≡ 15000 + (4000 * X)。

这个逻辑语句集合描述了我们的问题域。我们把这些断言进行了编号，以便在下文中引用。

利用合一和取式假言推理，可以把适合这个投资个体的投资策略推理成这些描述的逻辑结论。第一步要做的工作就是把断言 10 和断言 11 的合取与断言 7 的前提的前两个部分合一，也就是使

earnings(25000, steady) ∧ dependents(3)

与

earnings(X, steady) ∧ dependents(Y)

合一，使用的替换是{25000/X, 3/Y}。这个替换得到了一个新的蕴涵：

earnings(25000, steady) ∧ dependents(3) ∧ ¬ greater(25000, minincome(3)) → income(inadequate)

计值函数 minincome 得到如下表达式：

earnings(25000, steady) ∧ dependents(3) ∧ ¬ greater(25000, 27000) → income(inadequate)

因为根据断言 10、断言 3 和 greater 的数学定义，以上前提的所有三个部分都为真，所以它们的合取为真，整个前提也为真。因此可以应用取式假言推理得到结论 income（inadequate）。这作为断言 12 加入系统。

12. income(inadequate)

类似地，把

amount_saved(22000) ∧ dependents(3)

与断言 4 前提的前两个元素使用替换{22000/X, 3/Y}合一，得到以下蕴涵：

amount_saved(22000) ∧ dependents(3) ∧ greater(22000, minsavings(3)) → savings_account(adequate)

这里，计值函数 minsavings(3)得到表达式：

amount_saved(22000) ∧ dependents(3) ∧ greater(22000, 15000) → savings_account(adequate)

接着，因为这个蕴涵的前提的所有三个部分都为真，所以整个前提的结果为真，再一次使用取式假言推理，得到结论 savings_account(adequate)，并把它作为表达式 13 加入系统。

13. savings_account(adequate)

对表达式 3、表达式 12 和表达式 13 加以分析，不难看出蕴涵 3 的前提也是真的。于是我们第三次应用取式假言推理，得出的结论是 investment(combination)，即给我们的投资个体的投资建议。

这个例子说明了如何使用谓词演算来实际推理问题，把推理规则应用到初始的问题描述，从而得出正确的结论。我们还没有严密地讨论算法如何确定为解决给定问题作出正确的推理，以及把这种算法在计算机上实现的方式。这些主题会在第 3、4 和 6 章中进行讨论。

2.5 结语和参考文献

在这一章中，我们把谓词演算作为 AI 问题求解的一种表示语言对其进行了介绍。我们描述并且定义了这种语言的符号、项、表达式和语义。以谓词演算的语义为基础，我们定义了可以推导出逻辑派生自给定表达式集合的新语句的推理规则。我们定义了一种合一算法，用以求出使两个表达式匹配的变量替换，这是应用推理规则所必需的。我们用一个财务顾问的例子归纳了本章的内容，这个例子用谓词演算表示财务知识，并演示了问题求解的逻辑推理技术。

很多计算机科学的专业书籍都详细讨论了谓词演算，包括：Zohar Manna 和 Richard Waldinger 所著的《The Logical Basis for Computer Programming》（1985）、Jean H. Gallier 所著的《Logic for Computer Science》（1986）、Chin-liang Chang 和 Richard Char-tung Lee 所著的《Symbolic Logic and Mechanical Theorem Proving》（1973）、Peter B. Andrews 所著的《An Introduction to Mathematical Logic and Type Theory》（1986）。我们在第 14 章的自动推理中给出了更多的现代证明技术。

把谓词演算作为人工智能表示语言来讨论的书籍包括：Michael Genesereth 和 Nils Nilsson 所著的《Logical Foundations of Artificial Intelligence》（1987）、Nils Nilsson 所著的《Artificial Intelligence》（1998）、Larry Wos 所著的《The Field of Automated Reasoning》（1995）、Alan Bundy 所著的《Computer Modelling of Mathematical Reasoning》（1983，1998）以及 Ronald Brachman 和 Hector Levesque 所著的《Readings in Knowledge Representation》（1985）。要了解自动推理的有趣应用，可以参见 Bob Veroff 所著的《Automated Reasoning》（1997）。期刊《Journal for Automated Reasoning（JAR）》和会议《Conference on Automated Deduction（CADE）》则涵盖了当前研究的主题。

2.6　习题

1. 使用真值表来证明 2.1.2 节中的恒等式。
2. 可以用下面的真值表来定义一个新的操作符⊕，也就是异或：

P	Q	P⊕Q
T	T	F
T	F	T
F	T	T
F	F	F

仅使用∧、∨和¬来建立与 P⊕Q 等价的命题演算表达式。
使用真值表证明它们的等价性。

3. 把逻辑操作符“↔”读为“当且仅当”（if and only if）。P↔Q 被定义为与(P→Q) ∧(Q→P)等价。根据这个定义，用两种方法证明 P↔Q 逻辑等价于(P∨Q)→(P∧Q)：
   a）使用真值表证明；
   b）利用 2.1.2 节中的恒等式通过一系列替换来证明。
4. 证明命题演算中的蕴涵服从传递律，也就是((P→Q) ∧(Q→R))→(P→R)。
5. a）证明取式假言推理对于命题演算是可靠的。提示：使用真值表来枚举所有可能的解释。
   b）反绎（abduction）是一种从 P→Q 和 Q 推断 P 的推理规则。证明反绎是不可靠的（见第 7 章）。
   c）证明拒式假言推理((P→Q) ∧¬ Q)→¬ P 是可靠的。
6. 尝试合一下面的表达式对。要么给出它们的最一般的合一式，要么解释为什么它们不能合一。
   a）p(X, Y)和 p(a, Z)
   b）p(X, X)和 p(a, b)
   c）ancestor(X, Y)和 ancestor(bill, father(bill))
   d）ancestor(X, father(X))和 ancestor(david, george)
   e）q(X)和¬ q(a)
7. a）组合替换集合{a/X, Y/Z}和{X/W, b/Y}。
   b）证明替换集合的组合是服从结合律的。
   c）举一个例子说明替换集合的组合是不服从交换律的。
8. 用你喜欢的计算机语言在计算机上实现 2.3.2 节中的 unify 算法。

9. 给出图2-3的积木世界描述的另外两种解释。
10. Jane Doe要供养4个人，她有30 000美元的稳定收入，她的存款账户中有15 000美元。向2.4节中的通用投资顾问例子中加入描述她的情况的适当谓词，然后进行必要的合一和推理以求出提供给Jane Doe的投资建议。
11. 写出一系列逻辑谓词，这些谓词将完成简单的汽车故障诊断（例如，如果发动机熄火，车灯不亮，那么电池就有问题）。不必过于繁琐，只需要包括以下几种情况：电池问题、油用完了、火花塞坏了、发动机启动器坏了。
12. 下面的故事摘自N. Wirth所著的《算法 + 数据结构 = 程序》（Algorithms + data structures = programs）(1976)。

    我娶了个寡妇（让我们称她为W），她有个成年的女儿（称她为D）。我的父亲（称他为F）经常来我们家，爱上了我的继女并和她结婚了。于是我的父亲变成了我的女婿，我的继女变成了我的妈妈。几个月后，我的妻子给我生了个儿子（S_1），这个孩子变成了我父亲的小舅子，也就是我的舅舅。我父亲的妻子，也就是我的继女，也生了个儿子（S_2）。

    利用谓词演算，创建一系列表达式来表示上面故事的情节。加入一些表达式来定义基本的家庭关系，比如father-in-law（丈夫或妻子的父亲）的定义。然后对这个系统使用假言推理来证明这个结论“我是我自己的祖父”。

第3章　状态空间搜索的结构和策略

为了生存，主体要么用防护器具把自己武装起来并祈祷一切平安（如不能移动的植物），要么就要想方设法脱离危险境地并进入安全环境。如果是后一种情况，那么所有主体都会不断面临一个基本问题：现在该怎么做？

——丹尼尔·丹尼特《意识的解释》

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;
Then took the other…
两条路在黄色的树林间分叉，
可惜我不能都踏行。
我这个过客久久徘徊，
极目望断一条途径，
蜿蜒地进入树林；
于是我选择了另一条路……

——罗伯特·佛洛斯特《没有走的路》

3.0　简介

我们在第2章中把谓词演算作为人工智能表示语言的一个示例做了介绍。合式谓词演算表达式提供了一种描述问题域中对象和关系的手段，诸如假言推理这样的推理规则使我们可以从这些描述中推理出新的知识。这些推理定义了一个从中搜索问题解的空间。本章介绍状态空间搜索理论。

为了成功地设计和实现搜索算法，程序员必须能够分析和预测它们的行为。需要回答的问题包括：

- 问题求解器保证可以找到解吗？
- 问题求解器总是可以终止吗，也就是说它是否可能陷入无限循环？
- 当找到解时，能保证这个解是最优解吗？
- 搜索过程的时间复杂度如何？内存使用呢？
- 怎样使解释程序最有效地降低搜索复杂度？
- 怎样设计解释程序才能使它最有效地利用表示语言？

状态空间搜索（state space search）理论是我们回答这些问题的首选工具。只要把问题表示为状态空间图（state space graph），我们就可以利用图论工具来分析问题的结构和复杂度以及用以求解问题的搜索过程的结构和复杂度。

图是由一系列结点（node）和一系列连接结点对的弧（arc）或连接（link）构成的。在问题求解的状态空间模型中，图的结点被用来表示问题求解过程中的离散状态，比如逻辑推理的结果或棋盘的不同格局。图的弧表示状态之间的转换。这些转换对应于逻辑推理或博弈中的一

次合法移动。例如在专家系统中，状态描述了在推理过程某一阶段我们对问题实例的知识。专家知识是“if … then”形式的规则，允许我们产生新的信息；应用规则的动作被表示为状态之间的弧。

图论是推理对象之间结构和关系的最佳工具。实际上，也正是这种需求导致了它的产生。18世纪初期，瑞士数学家莱奥哈尔德·欧拉（Leonhard Euler）为了解决“哥尼斯堡七桥问题”而发明了图论。哥尼斯堡城中有一条河，河中有两个岛屿，而且共有七座桥连接着岛屿和河岸，如图3-1所示。

哥尼斯堡七桥问题是：一个散步者能否在仅经过每座桥一次的情况下遍历七座桥。尽管这里的居民无法找到这样的路线并且怀疑这是否可能，但是没有人能证明它是不可能的。欧拉以图论的形式建立了这幅图的另一种表示，如图3-2所示。河岸（rb1和rb2）和岛屿（l1和l2）被表示为图中的结点；桥被表示为结点之间带标签的弧（b1，b2，…，b7）。图表示保持了这个桥系统的关键特征，忽略了无关的特征，比如桥长、距离和桥的行走顺序。

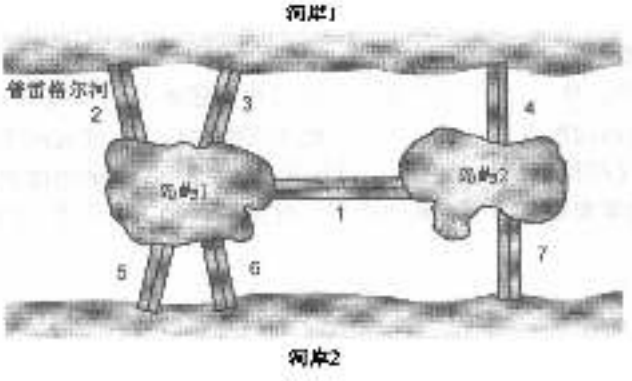


图3-1 哥尼斯堡城

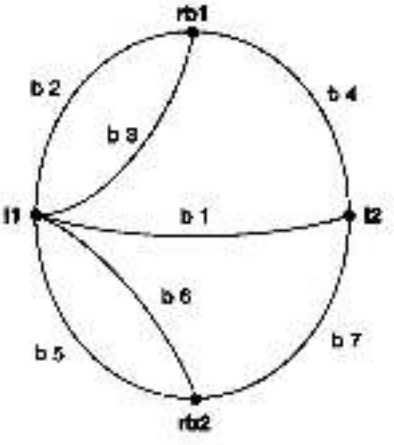


图3-2 哥尼斯堡桥系统的图

此外，我们可以使用谓词演算来表示哥尼斯堡桥系统。谓词connect对应于图的弧，断言两个地方是通过某个桥连接的。每座桥需要两个connect谓词，对应于穿越这座桥的两种方向。使用一个补充谓词表达式connect(X, Y, Z) = connect(Y, X, Z)表示可以以两个方向中的任意一个方向穿越任一座桥，那么就可以把上面的connect谓词删除一半。

connect(i1, i2, b1)	connect(i2, i1, b1)
connect(rb1, i1, b2)	connect(i1, rb1, b2)
connect(rb1, i1, b3)	connect(i1, rb1, b3)
connect(rb1, i2, b4)	connect(i2, rb1, b4)
connect(rb2, i1, b5)	connect(i1, rb2, b5)
connect(rb2, i1, b6)	connect(i1, rb2, b6)
connect(rb2, i2, b7)	connect(i2, rb2, b7)

如果使用一个补充谓词表达式connect(X, Y, Z) = connect(Y, X, Z)表示可以以两个方向中的任意一个方向穿越任一座桥，那么就可以把上面的connect谓词删除一半。

从所保持的连接性来看，谓词演算表示和图表示是等价的。实际上，算法可以在这两种表示之间相互翻译，而且不丢失任何信息。然而图表示可以使问题结构可视化，更加直观；而在谓词

演算表示中这些信息是隐含的。欧拉的证明说明了这个差异。

在证明不可能存在满足要求的路线时，欧拉把问题的焦点集中在图中结点的度（degree）上。注意，所有结点要么是奇数度的，要么是偶数度的。偶数度的结点具有偶数条弧与相邻结点连接。奇数度的结点具有奇数条弧与相邻结点连接。除了起点和终点以外，满足要求的路线必须符合：离开某个结点的度数与进入这个结点的度数精确相等。因此，奇数度的结点仅可用做路线的开始和结束，因为这些结点被穿越几次后就成了“死胡同”，散步者若不选择以前走过的弧就无法离开这个结点。

欧拉指出，除非一个图正好包含 0 或 2 个奇数度的结点，否则不可能存在满足前面要求的路线。如果有两个奇数度结点，那么这种路线应该从其中的一个结点开始，在另一个结点结束；如果没有奇数度结点，那么应该在同一个结点出发和结束。对于任何含有其他数量奇数度结点的图，这种行走是不可能的，哥尼斯堡城的情况也是如此。这个问题现在被称为“寻找遍历图的欧拉路径”。

注意：虽然谓词演算表示捕捉到了城中桥和地面之间的关系，但是它无法让人想起结点度数的概念。在图表示中，每个结点仅有一个实例，结点之间有弧相连，并不像谓词集合中一个结点多次作为参数出现。由于这个原因，图表示激发了结点度数的概念，这是欧拉证明的关键。这说明了图论在分析对象结构、属性和关系方面具有强大的功能。

下面，3.1 节将回顾基本的图论知识，然后介绍有限状态自动机和问题的状态空间描述。3.2 节将介绍如何通过作为问题求解方法的图搜索来求解问题。深度优先搜索和宽度优先搜索是搜索状态空间的两种策略。我们比较了这两种策略，还分析了目标驱动搜索和数据驱动搜索的差异。3.3 节说明了如何利用状态空间搜索来刻画基于逻辑的推理。贯穿本章的主线是使用图论来分析各种问题的结构和复杂度。

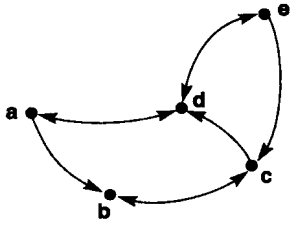
3.1　状态空间搜索的结构

3.1.1　图论（选读）

图是结点（或状态）的集合和连接这些结点的弧的集合。带标签的图用一个或多个描述符（标签）附加在每个结点上，以把这个结点与图中其他结点区分开来。在状态空间图中，描述符标识了问题求解过程中的各个状态。如果两个结点的描述没有差异，那么就认为它们是相同的。两个结点之间的弧是用它所连接结点的标签来描述的。

图的弧也可以有标签。弧标签一般有两种用法：一种是用来说明这个弧代表的是一种命名关系（就像在语义网中那样）；另一种是把权附加到弧上（就像在巡回推销员问题中那样）。如果在两个相同结点间有不同的弧（比如在图 3-2 中），那么也可以通过加标签来区分它们。

如果图中的弧具有方向性，那么这个图便是有向的。有向图中的弧通常用箭头来指示方向。具有两个方向的弧可以带两个箭头，但更多时候是根本不画出方向指示符。图 3-3 是一个带标签的有向图：弧(a, b)只可以从结点 a 穿越到结点 b，但弧(b, c)可以从两个方向穿越。



结点 = {a,b,c,d,e}

弧 = {(a,b),(a,d),(b,c),(c,b),(c,d),(d,a),(d,e),(e,c),(e,d)}

图 3-3　带标签的有向图

穿越图的路径（path）通过多个相继的弧把一系列结点连接起来。可以用一个有序链表来表示路径，方法是按结点在路径上的出现顺序把它

们记录在链表中。以图 3-3 为例，[a, b, c, d] 代表了依次穿过结点 a、b、c、d 的路径。

有根图有一个特别的结点，从这一结点到图中所有结点都存在一条路径，这个结点被称为根（root）。在画有根图时，通常把根画在页面的上方，高于其他结点。博弈的状态空间图通常是有根图，把开局作为根。图Ⅱ-5 中的有根图表示出了九宫游戏博弈图的初始部分。这是一个所有弧都只有一个方向的有向图。另外，注意这幅图中不包含环；棋手不可以取消移动（尽管他们有时很希望能取消！）。

树（tree）是两个结点之间至多有一条路径的图。树通常是有根的，这时树根经常被画在上方，像有根图那样。因为树的每个结点仅有一条路径供其他结点访问，所以不可能存在反复或循环穿越一系列结点的路径。

对于有根树或有根图，可以把结点之间的关系分为双亲（parent）、孩子（child）和兄弟（sibling）。这些关系的用法与通常的家庭关系一样，在有向弧上双亲出现在它的孩子之前。一个结点的多个孩子被称为兄弟。类似地，在有向图的某个路径上，祖先（ancestor）出现在后代（descendant）之前。在图 3-4 中，b 是结点 e 和 f 的双亲（所以，e 和 f 是 b 的孩子，它们彼此间是兄弟）。结点 a 和 c 是状态 g、h 和 i 的祖先，g、h 和 i 是 a 和 c 的后代。

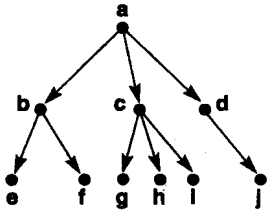


图 3-4　用来例示家庭关系的有根树

下面我们先给出这些概念的形式定义，然后再介绍如何使用状态空间来表示问题。

定义（图）　图是由以下元素构成的：

一系列**结点** N_1，N_2，N_3，…，N_n，…，不要求一定是有限的。

连接结点对的一系列**弧**。

弧是有序的结点对；例如，弧(N_3, N_4)把结点 N_3 连接到结点 N_4。这说明从 N_3 到 N_4 存在直接的连接，但不能说明 N_4 到 N_3 也如此，除非(N_4, N_3)也是一个弧，这种情况下连接 N_3 和 N_4 的弧便是**无向的**。

如果一个**有向**弧把 N_j 和 N_k 连接起来，那么 N_j 被称为 N_k 的**双亲**，N_k 是 N_j 的**孩子**。如果图中还包含一个弧(N_j, N_l)，那么 N_k 和 N_l 是**兄弟**。

有根图具有惟一的结点 N_s，N_s 是图中所有路径的原点。也就是说，根在图中没有双亲。

末端结点或**叶**结点就是没有孩子的结点。

如果图中存在一个有序的结点序列$[N_1, N_2, N_3, \cdots, N_n]$，序列中的每一个相邻结点对 N_i 和 N_{i+1}都是一个弧，即(N_i, N_{i+1})，那么就把这个序列称为图中长度为 $n-1$ 的**路径**。

在有根图的一条路径上，可以称一个结点是其后（其右侧）所有结点的**祖先**，同时又是其前（其左侧）所有结点的**后代**。

如果一条路径包含某一结点多次（某个结点 N_j 重复出现在路径上），那么便称这条路径含有**循环**或**回路**。

树是表示每个结点对之间只有一条惟一路径的图（所以，树上的路径不包含环）。

有根树的边是指向离开根的方向。有根树中的每个结点都有惟一的双亲。

如果存在一条路径包含某两个结点，那么就说这两个结点是**连通的**。

下面我们介绍有限状态自动机，它是计算设备的一种抽象表示方式，可以看作是遍历图中路径的一种自动机。

3.1.2　有限状态自动机（选读）

我们可以将机器看作是能够接受输入值、可以产生输出值并能用某种内部机制（状态）保存跟踪前面输入信息的系统。有限状态自动机（Finite State Machine，FSM）是有限的有向连通图，包括状态的集合、输入值的集合和状态转换函数，状态转换函数描述输入流中的元素对图的状态的作用。输入值流在图中生成一条经过有限状态自动机状态的路径。有限状态自动机可看作计算的抽象模型。

这种机器最初被用来识别形式语言的成员。这些成员通常是字符构成的串（“字母表”上字母构成的“单词”）。在5.3节，我们会将其定义扩展到概率（probabilistic）有限状态自动机。这些状态自动机在语言表达式分析中起着重要作用，不管是计算机语言还是人类语言，参见5.3节、9.3节和第15章。

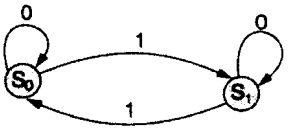
定义（有限状态自动机）　一个**有限状态自动机**是一个有序三元组(S, I, F)，其中：

S是连通图中**状态**的有限集合$s_1, s_2, s_3, \cdots, s_n$。

I是**输入值**的有穷集合$i_1, i_2, i_3, \cdots, i_m$。

F是状态转换函数，描述任意$i \in I$对机器状态S的作用，即$\forall i \in I, F_i: (S \rightarrow S)$。如果机器正处于状态$s_j$，此时输入$i$，那么机器的下一个状态是$F_i(s_j)$。

举一个有限状态自动机的简单例子，令$S = \{s_0, s_1\}$，$I = \{0, 1\}$，$f_0(s_0) = s_0$，$f_0(s_1) = (s_1)$，$f_1(s_0) = s_1$，$f_1(s_1) = s_0$。用这个装置，有时称为**触发器**，输入值0不改变状态，而输入1会改变机器的状态。我们可以用两种等价的形式来形象化地描述这个机器，第一种是带标签和有向弧的有限图，如图3-5a所示；第二种是状态转换表，如图3-5b所示。在状态转换表中，输入值列在最顶上一行，状态列在最左边一列，一个输入应用到一个状态时的输出列在对应的行列相交的格中。

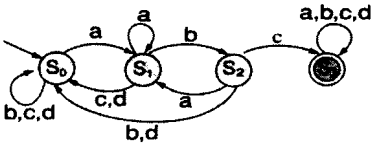


	0	1
S_0	S_0	S_1
S_1	S_1	S_0

a）有限状态图　　b）状态转换表

图3-5　触发器的有限状态图和状态转换表

有限状态自动机的第二个例子由图3-6a中的有向图和图3-6b的状态转换表描述。图3-6的有限状态自动机描述了什么呢？在两个假定下，这个机器可以看作是字母表{a, b, c, d}上所有包含序列“abc”的字符串的识别器。第一个假定是状态s_0是起始状态，第二个假定是s_3是接受状态。这样，输入流的第一个元素将作用于状态s_0。如果输入流最后结束时机器处于状态s_3，那么机器将识别出输入流中有序列“abc”。



	a	b	c	d
→ S_0	S_1	S_0	S_0	S_0
S_1	S_1	S_2	S_0	S_0
S_2	S_1	S_0	S_3	S_0
◉ S_3	S_3	S_3	S_3	S_3

a）有限状态图　　b）状态转换表

图3-6　字符串识别例子

上面介绍的是有限状态接受自动机，有时又称为摩尔机（Moore machine）。我们约定用一个没有起点状态的箭头指出摩尔机的起始状态，并用特殊的圆圈（通常是双圈）表示接受状态（可能有多个），如图 3-6 所示。现在给出摩尔机的形式化定义：

定义（有限状态接受器（摩尔机））　一个**有限状态接受器**是一个有限状态自动机(S, I, F)，其中：

$\exists s_0 \in S$ 使得输入流从 s_0 状态开始。

$\exists s_n \in S, s_n$ 是**接受**状态。输入流被接受，如果它使机器停止在接受状态。实际中很可能有一个接受状态的集合。

有限状态接受器表示为$(S, s_0, \{s_n\}, I, F)$。

我们给出了一种强有力的概念的两个相当简单的例子。在自然语言理解（第 15 章）部分，我们将看到有限状态识别器是确定一类字符、词或句子是否具有需要的属性的重要工具。一个有限状态接受器基于它接受的字母（字符）集合和接受的词（串）的集合隐含定义了一个形式语言。

我们只介绍了确定性（deterministic）有限状态自动机，任意输入值对一个状态的转换函数给出惟一的后续状态。概率有限状态自动机是另一种重要的建模技术，其转换函数为一个状态下的每个输入定义一个输出状态的分布。我们将在 5.3 节和第 15 章考虑这种模型。下面我们考虑问题求解分析的一种更通用的图形表示：状态空间。

3.1.3　问题的状态空间表示

在问题的状态空间表示（state space representation）中，图的结点对应于部分问题解的状态，弧对应于问题求解过程中的各个步骤。一个或多个初始状态（initial state）对应于问题实例中的给定信息，并成为图的根。图还定义了一个或多个目标情形，即问题实例的解。状态空间搜索（state space search）把问题求解过程当作寻找一条从起始状态到目标状态的解路径（solution path）。

目标可以描述一种状态，比如九宫游戏中胜利的棋盘状态（图Ⅱ-5），或 8 格拼图游戏中的目标布局（见图 3-7）。或者，也可以描述解路径本身的某种属性。在巡回推销员问题（见图 3-9 和图 3-10）中，当发现了穿越图中所有结点的“最短”路径时便终止搜索。在语法分析的例子（见 3.3 节）中，如果找到了可以成功分析语句结构的路径，那么终止搜索。

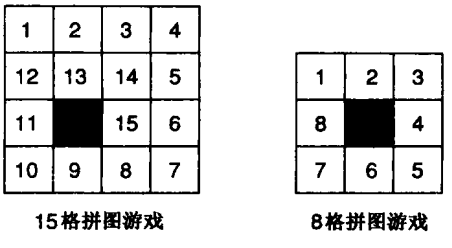


图 3-7　15 格拼图游戏和 8 格拼图游戏

状态空间的弧对应于求解过程的各个步骤，穿越空间的路径代表不同完成阶段中的解。路径是这样被搜索的：从起始状态开始并顺着图继续，直到满足目标描述或放弃搜索。通过对路径上已存在状态应用算子可以生成新的状态，算子可以是博弈中的“合法移动”也可以是逻辑问题或专家系统中的推理规则。搜索算法的任务是找到一条穿越这种问题空间的解路径。因此，搜索算法必须把起始结点到目标结点的路径记录下来，因为这些路径包含了产生问题解的操作序列。

现在我们给出问题的状态空间表示的形式定义。

定义（状态空间搜索）　一个**状态空间**可表示为一个四元组$[N, A, S, GD]$，其中：

N 是图的结点或状态的集合。图的结点对应于问题求解过程中的各个状态。

A 是结点之间弧（也就是连接）的集合。图的弧对应于问题求解过程中的各个步骤。

S 是 N 的非空子集，含有问题的起始状态。

GD 是 N 的非空子集，含有问题的目标状态。可以通过以下任一种方式来描述 GD 中的状态：

1）搜索中遇到的状态的一种可测量属性。

2）搜索中探索到的路径的一种可测量属性，例如路径的弧的穿越成本（transition cost）。

解路径是穿越图的一条特殊路径，它从 S 中的一个结点开始，到 GD 中的一个结点结束。

很多图都具有这样一个一般特征，即有时可以从不同的路径到达相同的状态，这是图搜索算法设计中经常遇到的问题之一。例如，在图 3-3 中，从状态 a 到状态 d 的路径既可以经过 b 和 c，又可以直接从 a 到 d。这使得根据问题的需要选取最佳路径尤其重要。此外，到达一个状态的多条路径可能导致解路径上出现循环或回路，阻碍算法达到目标。例如，以图 3-3 中状态 e 为目标的盲目搜索可能永不停止地搜索这个状态序列：abcdabcdabcd……

如果要搜索的空间是树（如图 3-4 所示），那么就不会发生循环问题。因此，把状态空间是树的问题和状态空间可能含有回路的问题区别开来是很重要的。通用的图搜索算法必须能够检测和消除解路径上的潜在循环，而树搜索免除了这种测试和相应的开销，从而可以得到更高的效率。

下面以九宫游戏和 8 格拼图游戏为例来介绍简单博弈的状态空间。这两个例子的终止条件都是上述我们状态空间搜索定义中的第一种类型。例 3.1.3 的“巡回推销员问题”的目标是描述第二种类型和路径自身的总代价。

例 3.1.1　九宫游戏

图Ⅱ-5 显示了九宫游戏的状态空间表示。起始状态是空的棋盘，终止条件（即目标描述）是三个 X 在一行、一列或对角线上这样的棋盘状态（假定目标是 X 方胜）。从起始状态到目标状态的路径代表了赢得一盘棋的走子序列。

空间中的状态就是游戏中可能产生的所有不同 X 和 O 的布局。当然，尽管在 9 个空格中排列{空, X, O}有 3^9 种方式，但是它们中的大多数是从来不会在一盘实际对弈中发生的。弧是博弈中的合法移动所产生的，合法移动就是交替向未用过的位置放 X 和 O。这个状态空间是图而不是树，因为可以从不同路径到达第三层或更深层的某些结点。然而在这个状态空间中不存在环，因为图的有向弧决定了移动是不可以撤销的。一旦已经到达某个状态就不可能返回到这个结构的更上层。因此在生成路径时没有必要检查环。我们把具有这种特征的图结构称为*有向非循环图*（directed acyclic graph），或 DAG，这种图在状态空间搜索和图模型（见第 13 章）中是很常见的。

状态空间表示也为确定问题的复杂度提供了一种途径。在九宫游戏中，第一次移动有 9 种情况，对每一种情况有 8 种可能的回应，对这 8 种中的每一种又有 7 种可能的回应，依此类推。这可产生 $9 \times 8 \times 7 \times \cdots$ 也就是 9！种不同的路径。尽管对计算机来说，穷举搜索这一数量（362 880）的路径不是不可能的，但是很多重要的问题尽管规模要大得多，但是其复杂度也是阶乘级的或指数级的。国际象棋有 10^{120} 种可能的博弈路径；西洋跳棋有 10^{40} 种可能的博弈路径，其中某些情况可能从来不会在实际的博弈中发生。穷举搜索这样的空间是非常困难的，也是不可能的。用来搜索这种庞大空间的策略必须使用启发式搜索来降低搜索的复杂度（见第 4 章）。

例 3.1.2　8 格拼图游戏

在图 3-7 所示的 15 格拼图游戏中，15 张不同的牌被嵌进具有 16 个空位的方格中。留出一个空位是为了可以把牌移来移去以组成不同的花样。我们的目标是找到一个移动序列使棋盘呈现

出某种目标格局。这是一种很常见的游戏，我们很多人小时候都玩过（我记得的版本是棋盘大约有 3 英寸见方，是黑色的，牌是红色的和白色的）。

这种棋的很多有趣特征使它对研究问题求解的人员来说非常有价值。它的状态空间大小足以令人感兴趣，但又不是特别难以处理（如果把对称状态当作独立情况，那么是 16!）。这种游戏的博弈状态易于表示；其走法的丰富性足以提供很多有趣的启发（见第 4 章）。

8 格拼图游戏是 15 格拼图游戏的 3 ×3 版本，可以把 8 张牌在 9 个空位中移来移去。因为 8 格拼图游戏产生的状态空间比完整的 15 格拼图游戏要小，且它的状态空间图容易放在一页内，所以本书的很多例子都使用了 8 格拼图游戏。

尽管在实际游戏中具体的操作是移动每张牌（比如“如果 7 这张牌的右侧是空位，那么就可以把 7 向右移”或“把 3 这张牌向下移”），但是按“移动空位”来思考会更加简单。这简化了移动规则的定义，因为牌有 8 张，但空位仅有一个。为了完成移动，我们必须确保不把空位移到棋盘之外。所以，4 种移动不是在所有时候都是适用的；例如，当空位在角落时，只可能有两种移动。

因此，合法移动有以下 4 种：

向上移动空位　↑
向右移动空位　→
向下移动空位　↓
向左移动空位　←

如果我们确定了 8 格拼图游戏的起始状态和目标状态，那么就可以给出针对这个问题求解过程的状态空间（见图 3-8）。可以利用一个简单的 3 × 3 数组来表示状态。如果采用谓词演算表示，就可以使用一个带 9 个参数（对应格子上号的位置）的“状态”谓词。4 种移动过程（描述了空位的 4 种可能移动）定义了状态空间中的弧。

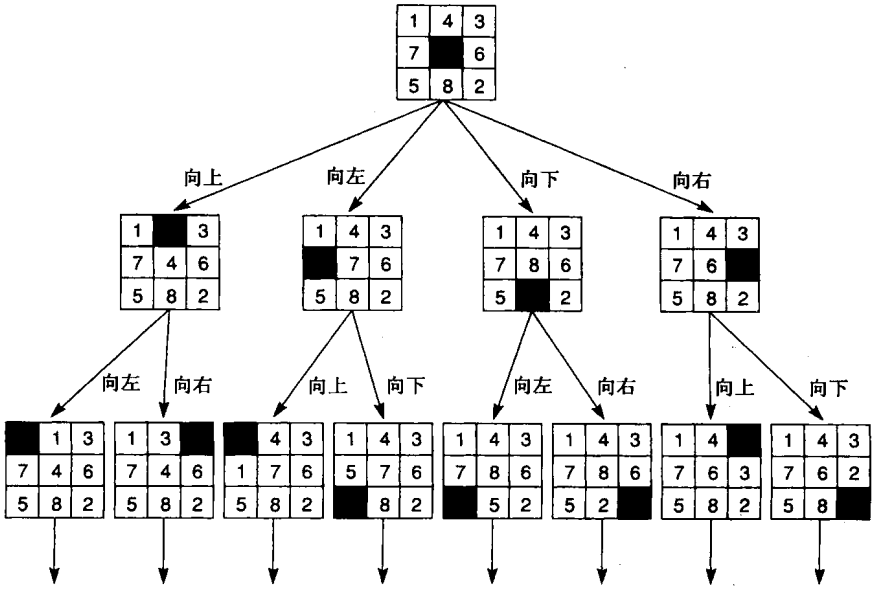


图 3-8　通过移动空位产生的 8 格拼图游戏的状态空间

与九宫游戏一样，8 格拼图游戏的状态空间也是图（大多数状态有多个双亲），但与九宫游戏不同的是，可能存在回路。状态空间的 GD（也就是目标描述），是一种特定状态，或者说棋

盘格局。当在一条路径上发现了这种状态时，搜索便终止了。从起始到目标的路径就是满足要求的移动序列。

一个有趣的现象是 8 格拼图游戏和 15 格拼图游戏的整个状态空间是由两个不相连的（而且是大小相等的）子图组成。这使得从任何给定起始状态开始的搜索都只能到达搜索空间中的一半可能状态。如果我们交换或撬动两块近邻的牌，那么就可以访问到搜索空间的其他部分。

例 3.1.3　巡回推销员问题

假定一个推销员要访问五个城市然后回到家。这个问题的目标是找到推销员访问所有城市并回到出发城市的最短旅行路径。图 3-9 给出了这个问题的一个实例。图中结点表示城市，每个弧标有一个权表示通过这个弧的开销。这个开销可能是在两个城市之间乘汽车所必需的里程或乘飞机所必需的费用。为了简便，我们假定推销员住在 A 城并要返回那里，不过这个假定仅是把 N 个城市的问题缩减为(N－1)个城市的问题。

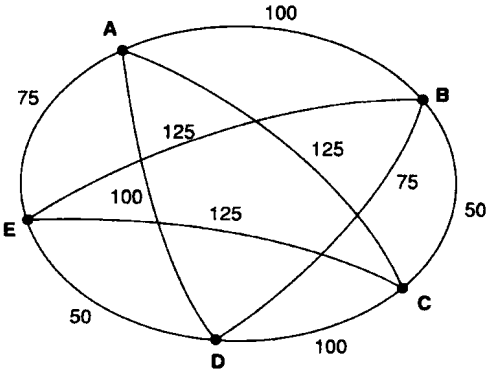


图 3-9　巡回推销员问题的一个实例

路径[A, D, C, B, E, A]是可能路线的一个例子，它的关联成本是 450 英里。目标描述需要一个费用最低的完整路线。注意这里的目标描述是整个路径的属性，而不是单一的状态。这对应于状态空间搜索定义中的第二类目标描述。

图 3-10 显示了产生和比较可能解路径的一种方式。从结点 A 出发，加入可能的下一个状态直到路径中包含了所有城市，而且返回到家。最终目标是要找到具有最小开销的路径。

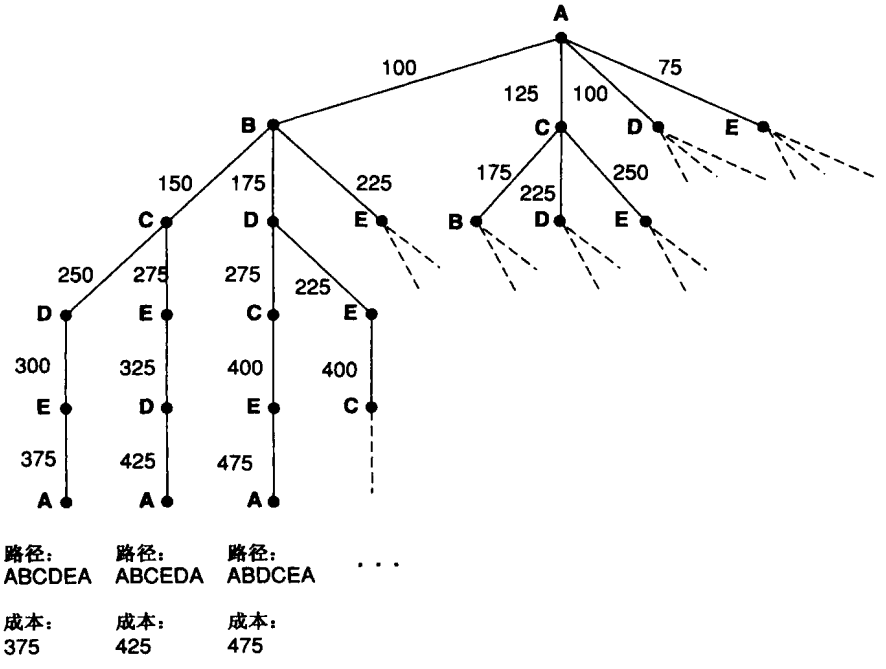


图 3-10　巡回推销员问题的搜索过程

注：每个弧标出了从起点（A）到弧端点的路径的权累加值

根据图3-10可以得出，穷举搜索巡回推销员问题的复杂度是(N－1)!，其中N是图中城市的个数。对于9个城市的情况，我们可以穷举试验所有的路径，但对于令人感兴趣的任意规模的问题实例，比如50个城市，我们就无法在可行的时间内完成简单的穷举搜索。实际上，对于复杂度为N!的搜索，由于其空间增长速度非常快，所以搜索组合很快变得难以控制。

我们可以使用几种技术来降低搜索复杂度。一种技术叫做*分支定界*（branch and bound）（Horowitz and Sahni 1978）。分支定界每次仅产生一条路径，并记录目前为止已发现的最佳路径。而且把这个值作为以后候选对象的界限（bound）。该算法一边逐一加入城市构建路径，一边对目前路径进行分析。如果发现这个路径的最佳可能扩展（即分支）的成本大于界限的成本，那么它就排除这个路径和它的所有可能扩展。这样做虽然大大降低了搜索工作量，但还剩下指数数量（1.26^N，而不是N!）的路径。

控制搜索的另一种策略是根据规则“去最近的未访问过城市”来构建路径。穿越图3-11的*最近邻路径*是[A, E, D, B, C, A]，它的成本是375英里。这种方法是非常高效的，因为仅要试验一种路径。最近邻（也称作*贪婪*）启发式搜索难免会有错误，对于有些图它找不到最短的路径，见图3-11，但是当时间要求使穷举搜索不可行时可以把它作为一种补充的方法。

我们在3.2节分析用于状态空间搜索的策略。

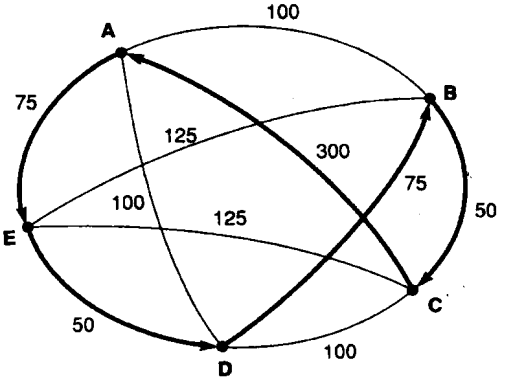


图3-11 巡回推销员问题的一个实例

注：图中的粗线是最近邻方法找到的最低成本路径。注意，这个路径的成本是550英里，并不是最短的路径。弧(C, A)相对很高的成本使这种启发失败

3.2 用于状态空间搜索的策略

3.2.1 数据驱动搜索和目标驱动搜索

可以按两个方向来搜索状态空间：从问题实例的给定数据出发向目标搜索；或从目标出发返回到数据。

数据驱动搜索（data－driven search）（有时也称为*正向追索*（forward chaining））中，问题求解器从问题的给定事实和改变状态的合法移动或规则的集合入手。然后把规则应用到事实产生新的事实，接下来新的事实又被规则用来产生更多新的事实。搜索便如此继续下去，直到（我们希望）产生满足目标条件的一条路径。

也可以采用另一种策略：从想要求解的目标着手。先分析怎样使用规划或合法移动来产生这个目标并求出要应用这些规则或移动必须具有哪些条件。这些条件成为要搜索的新目标，或者称为*子目标*（subgoal）。然后继续反向追溯相继的子目标，直到（我们希望）返回到问题中的事实。这样便找到了从数据到目标的移动或规则链，但顺序是倒序的。这种方法称为*目标驱动*（goal-driven）推理，或者称为*反向追索*（backward chaining），它让我们想起了一种简单的儿童游戏，从终点到起点反向地寻找穿越迷宫的路线。

概括地讲，数据驱动推理是从问题的事实入手，然后应用规则或合法移动来产生通往目标的新的事实；目标驱动推理是把焦点集中在目标上，寻找可以产生这个目标的规则，并环环相扣地反向追索相继的规则和子目标直到到达问题中给定的事实。

归根到底，无论是数据驱动的问题求解器还是目标驱动的问题求解器，它们搜索的都是同一个状态空间图；不过，这两种问题求解者搜索的顺序和搜索的实际状态数量有所不同。到底应该优先选择哪一种策略应该由问题本身的特征来决定。这些特征包括规则的复杂度、状态空间的“形状”、问题中的已知数据和这些数据的属性。面所有这些特征都会因问题的不同而有所差异。

下面举例说明搜索策略对搜索复杂度可能造成的影响。考虑这样一个问题：确认或否认这个命题“我是托马斯·杰斐逊的后代”，解释是“我”和托马斯·杰斐逊间的直系血统路径。可以从两个方向搜索这个空间：从“我”开始顺着先辈的路线到托马斯·杰斐逊；或者从托马斯·杰斐逊开始顺着他的后代展开分析。

如果做一些简单的假定，那么就可以估计出每个方向要搜索的空间大小。托马斯·杰斐逊大约出生在250年前，如果我们假定25年繁衍一代，那么要求的路径长度大约是10。因为每个人的双亲数是固定的（两个），那么从“我”向上搜索需要分析2^{10}数量级个前辈。从托马斯·杰斐逊开始的正向搜索需要分析更多的状态，因为人们往往有两个以上的孩子（尤其是在18世纪和19世纪）。如果我们假定每个家庭平均有3个孩子，那么需要分析3^{10}数量级个结点的家族树。所以，从“我”向上搜索要分析的结点数较少。但应该注意到两个方向的复杂度都是指数级的。

对数据驱动方法和目标驱动方法的选择依赖于要解决的问题的结构。对于以下情况建议使用目标驱动搜索：

1）目标或假设是在问题陈述中给出的，或者它们很容易被形式化。例如，在数学定理证明程序中，目标是要证明的定理。很多诊断系统以一种系统的方式考虑可能的诊断，并利用目标驱动推理来确认或排除这些可能的诊断。

2）与问题事实匹配的规则数量非常多，因而产生的结论或目标也越来越多。因为较早地选择目标可以有效排除大量分支，所以目标驱动搜索可以更高效地剪除这种空间（见图3-12）。例如，在数学定理证明程序中，用来产生给定定理的规则数通常远远小于可以应用到整个公理集的规则数。

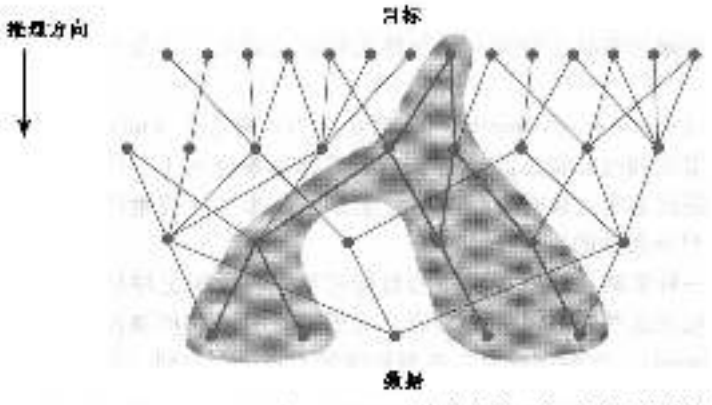


图3-12　目标驱动的状态空间搜索。目标驱动搜索有效地剪除了无关的搜索路径

3）问题数据不是给定的，而是要由问题求解器来获取的。在这种情况下，目标驱动搜索可以引导如何获取数据。例如在医疗诊断程序中，现有的诊断化验种类非常多。医生仅建议那些对确认或排除某个假设有必要的化验。

可见，目标驱动搜索是使用关于预期目标的知识来引导搜索，通过相关的规则来排除空间分支。

数据驱动搜索（见图3-13）适合的问题包括：

1）问题的初始陈述给出了所有或大部分数据。解释问题经常符合这种情况，给出一系列数据并要求系统提供一个高层的解释。分析特定数据的系统适合采用数据驱动方法，比如勘探程序（PROSPECTOR）或 Dipmeter 程序，它们对地质数据进行解释，或者分析某个地方可能蕴藏什么矿。

2）潜在目标的数量非常庞大，但是使用特定问题实例的给定信息和事实的方式很有限。DENDRAL 程序是这样的一个例子，它是一个用来分析有机化合物分子结构的专家系统，这个系统的分析依据是化合物的分子式、大规模的光谱数据以及其化学知识。对任何一种有机化合物来说，都存在数量极其可观的可能结构。然而，大规模的化合物光谱数据使 DENDRAL 可以排除这些可能结构中的大多数，仅留下其中的很少几个结构。

3）难以形成目标或假设。比如在使用 DENDRAL 的情况中，初始时可能对化合物的可能结构知之甚少。

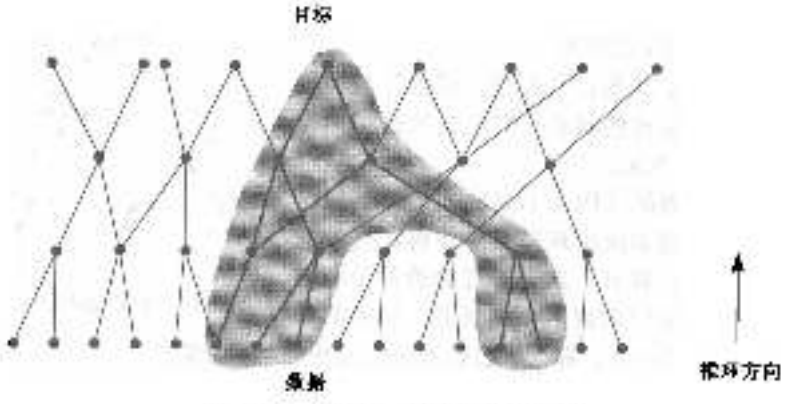


图3-13 数据驱动的状态空间搜索

数据驱动搜索使用了从一个问题的已经数据中提取的知识和约束，引导搜索沿着已知的路线到正解。

总而言之，必须针对待解决的特定问题进行深入的分析，这是必不可少的。要考虑的因素包括：规则应用的分支因子（branching factor）（即平均起来，按每个方向规则应用所产生的新状态数目，参见第4章）、数据的可用性以及确定潜在目标的简易性。

3.2.2 图搜索的实现

不论是使用目标驱动搜索还是数据驱动搜索来求解问题，问题求解器都必须在状态空间图中找到一条从起始状态到目标的路径。这条路径的顺序列对应了有序的求解步骤。如果问题求解器被赋予了神谕或其他确实可靠的机制来选取解路径，那么就不再需要搜索，问题求解器会不犯任何错误地穿越空间到达预期目标，而且一边行进一边就建立了解路径。因为对于我们感兴趣的问题来说神谕是不存在的，所以问题求解器必须考虑穿越空间的不同路径直到找到目标。回溯（backtracking）是系统地尝试穿越状态空间的所有路径的一种技术。

我们从回溯开始讨论搜索方法的原因是，回溯技术是计算机科学家最早研究的搜索算法之一，而且它可以在面向堆栈的递归环境中自然地实现。3.2.3 节将结合深度优先搜索（depth-first search）给出回溯算法的一个较简单版本。

回溯搜索从起始状态出发沿一条路径前进直到要么到达目标，要么到达一个“死端”。如果

发现了目标，它退出搜索并返回解路径。如果到达的是一个死端，那么它便回溯到路径上含有未分析过最近兄弟结点，并沿这个分支继续下去，如下面的递归规则所述：

如果当前状态 S 不满足目标描述的要求，那么便产生它的第一个后代 S_{child1}，并对这个结点递归地应用回溯过程。如果回溯没有在以 S_{child1} 为根的子图上发现目标结点，那么便对它的兄弟 S_{child2} 应用递归过程。继续上面的过程直到一个孩子的某个后代是目标结点或已经搜索了所有的孩子。如果 S 的孩子没有一个可以通向目标，那么回溯便“无功而返”到 S 的双亲，并在那里对 S 的兄弟应用以上过程，依此类推。

这种算法不断搜索直到找到一个目标或穷举了状态空间。图 3-14 显示了把回溯算法应用到一个假想状态空间的情况。虚箭头方向代表搜索沿空间上下的过程。每个结点旁边的数字指出了它被访问的顺序。下面定义一种回溯算法，我们使用 3 个列表来记录状态空间中的结点：

SL：状态列表，列出当前正在试验路径的状态。如果发现了目标，那么 SL 便包含了解路径上状态的有序列表。

NSL：新状态列表，含有等待评估的结点，也就是其后代还没有被产生和搜索的结点。

DE：用来记录死端，列出已经发现其后代不包含目标的状态。如果再次遇到这些状态，它们会被检测为是 DE 中的元素并立刻不再考虑。

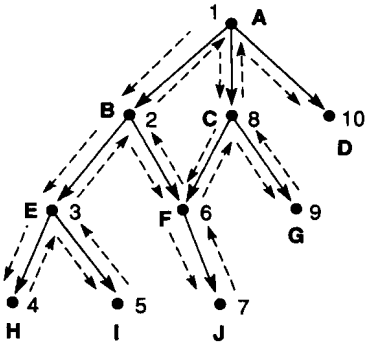


图 3-14　对假想状态空间的回溯搜索

在定义可用于一般情况（图而不是树）的回溯算法时，必须探测任何状态的多次出现以便不会再次进入这种状态而导致（无限的）循环。这是通过检验每个新产生状态是否是这三个列表中的成员来实现的。如果新的状态属于这些列表中的任一个，那么它已经被访问过并可以被忽略。

```
function backtrack;
begin
  SL := [Start]; NSL := [Start]; DE := [ ]; CS := Start;                    % initialize:
  while NSL≠[ ] do                                  % while there are states to be tried
    begin
      if CS = goal (or meets goal description)
        then return SL;                        % on success, return list of states in path.
      if CS has no children (excluding nodes already on DE, SL, and NSL)
        then begin
          while SL is not empty and CS = the first element of SL do
            begin
              add CS to DE;                                  % record state as dead end
              remove first element from SL;                                  %backtrack
              remove first element from NSL;
              CS := first element of NSL;
            end
          add CS to SL;
        end
        else begin
          place children of CS (except nodes already on DE, SL, or NSL) on NSL;
          CS := first element of NSL;
          add CS to SL
        end
    end;
    return FAIL;
end.
```

在回溯中，当前正被考虑的状态叫做当前状态 CS。它总是等于最近加入到 SL 中的状态，因此代表了当前正被探索解路径的“前线”。然后向 CS 应用推理规则、博弈中的移动或其他合适的问题求解操作符。这样便得到一系列新的有序状态（也就是 CS 的孩子）。这些状态中的第一个状态被用做新的当前状态，其余的状态被依次放入 NSL 供以后分析。新的当前状态被加入到 SL 并继续搜索。如果 CS 没有孩子，那么算法会将它从 SL 中删除（这便是算法的“回溯”），然后分析它的前驱的其他孩子。

如果对图 3-14 应用回溯（backtrack）算法，那么其过程如下：

初始化：SL = [A]; NSL = [A]; DE = [ ]; CS = A;

迭代后	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[B A]	[B C D A]	[ ]
2	E	[E B A]	[E F B C D A]	[ ]
3	H	[H E B A]	[H I E F B C D A]	[ ]
4	I	[I E B A]	[I E F B C D A]	[H]
5	F	[F B A]	[F B C D A]	[E I H]
6	J	[J F B A]	[J F B C D A]	[E I H]
7	C	[C A]	[C D A]	[B F J E I H]
8	G	[G C A]	[G C D A]	[B F J E I H]

正如上面所呈现的，回溯实现了数据驱动搜索，把根作为起始状态然后评估它的孩子以搜索目标。如果使目标成为图的根然后评估它的后代来寻找起始状态，那么就可以把这种算法看成是目标驱动的搜索。如果目标描述属于第二种类型（见 3.1.3 节），那么算法必须分析 SL 上的路径来判断目标状态。

回溯是一种搜索状态空间图的算法。随后将要介绍的图搜索算法（包括深度优先搜索、宽度优先搜索和最佳优先搜索）都使用了回溯中所用的思想，这些思想包括：

1）未处理状态列表（NSL）的使用使算法可以返回（回溯）到这些状态中的任一个状态。

2）“bad”状态列表（DE）防止算法重试无用的路径。

3）如果发现了目标，就返回当前解路径的结点列表（SL）。

4）显式检查新的状态是否是这些列表的成员以防止死循环。

下一节介绍的搜索算法与回溯一样，使用列表来记录搜索空间中的状态。这些算法包括*深度优先搜索*、*宽度优先搜索*和*最佳优先搜索*（见第 4 章），与回溯不同的是，它们为实现其他图搜索策略提供了一种更灵活的基础。

3.2.3 深度优先搜索和宽度优先搜索

除了指定搜索方向（数据驱动还是目标驱动），搜索算法还必须决定按什么顺序来解析树上或图上的状态。这一节介绍两种方法来考虑图中结点的可能顺序：*深度优先*（depth-first）搜索和*宽度优先*（breadth-first）搜索。

考虑图 3-15 所示的图。我们用标签（A, B, C, …）标出了各个状态以便在下文的讨论中引用它们。在深度优先搜索中，当分析一个结点时，在分析它的任何兄弟之前分析它的所有孩子和

后代。深度优先搜索尽可能地向搜索空间的更深层前进。只有找不到状态的后代时才会考虑它的兄弟。对于图 3-15 所示的图来说，深度优先搜索分析状态的顺序是 A，B，E，K，S，L，T，F，M，C，G，N，H，O，P，U，D，I，Q，J，R。3.2.2 节的回溯算法实现的就是深度优先搜索。

与深度优先搜索正好相反，宽度优先搜索一层一层地探索空间。只有给定层上不再存在要探索的状态时算法才转移到下一个更深层次。对于图 3-15 所示的图，宽度优先搜索分析状态的顺序是 A，B，C，D，E，F，G，H，I，J，K，L，M，N，O，P，Q，R，S，T，U。

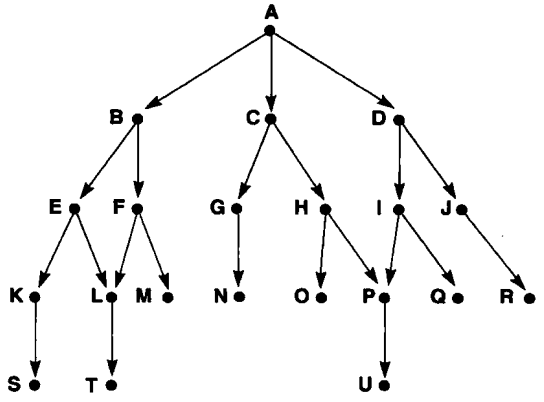


图 3-15　深度优先搜索和宽度优先搜索例子中所用的图

在宽度优先搜索的实现中，我们使用列表 open 和 closed 来跟踪穿越空间的过程。open 类似于回溯算法中的 NSL，它列出已经产生但是它的孩子还未被分析的状态。从 open 中删除状态的顺序是由搜索的顺序决定的。closed 记录了已经分析过的状态。closed 是回溯算法中 DE 和 SL 列表的联合。

```
function breadth_first_search;

begin
  open := [Start];                                              % initialize
  closed := [ ];
  while open≠[ ] do                                             % states remain
    begin
      remove leftmost state from open, call it X;
        if X is a goal then return SUCCESS                      % goal found
          else begin
            generate children of X;
            put X on closed;
            discard children of X if already on open or closed; % loop check
            put remaining children on right end of open         % queue
          end
    end
  return FAIL                                                   % no states left
end
```

孩子状态是由推理规则、博弈中的合法移动或其他的状态转换操作符生成的。每一次迭代产生状态 X 的所有孩子并且将这些孩子都加入到 open 中。注意：open 被当作一个队列（也就是一种先入先出（FIFO）的数据结构）来维护。状态是从右侧加入到这个列表并从左侧移出。这使搜索优先考虑已经在 open 中时间最长的状态，这使这种搜索是宽度优先的。如果孩子状态是已经发现的（已经出现在 open 或 closed 中的）状态，那么它会被丢弃。如果算法是因“while”循环条件（open = [ ]）不再满足而终止，那么说明算法已经搜索了整个状态空间，但并未发现预期目标：搜索失败了。

跟踪图 3-15 的宽度优先搜索的结果如下。每个连续的数字 1，2，3，…代表“while”循环的次数。U 是预期的目标状态。

1）open = [A]；closed = [ ]。

2) open = [B, C, D]; closed = [A]。

3) open = [C, D, E, F]; closed = [B, A]。

4) open = [D, E, F, G, H]; closed = [C, B, A]。

5) open = [E, F, G, H, I, J]; closed = [D, C, B, A]。

6) open = [F, G, H, I, J, K, L]; closed = [E, D, C, B, A]。

7) open = [G, H, I, J, K, L, M], 因为 L 已经在 open 中; closed = [F, E, D, C, B, A]。

8) open = [H, I, J, K, L, M, N]; closed = [G, F, E, D, C, B, A]。

9) 依此类推，直到找到了 U 或 open = [ ]。

图 3-16 给出了对图 3-15 所示的图进行 6 次宽度优先搜索迭代后的情况。open 和 closed 中的状态被突出显示出来。不带阴影的状态是算法还没有发现的状态。注意：open 记录了任何阶段中搜索的“前线”状态，而 closed 记录了已经访问过的状态。

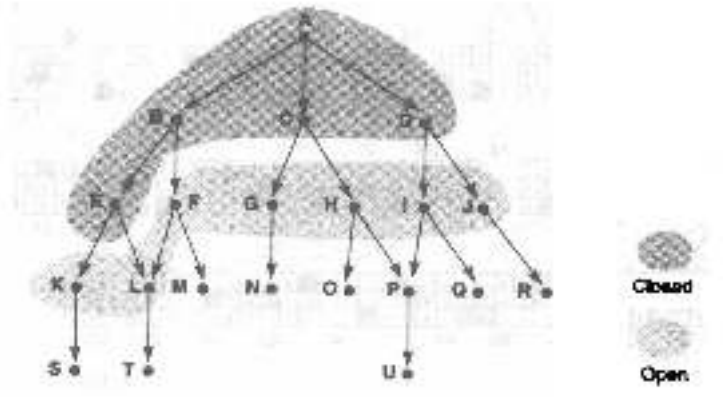


图 3-16 对图 3-15 的宽度优先搜索

注：图中突出显示了第 6 次迭代时的 open 和 closed 中的状态

因为宽度优先搜索先考虑图中每一层的所有结点然后再向图的更深空间前进，所以距离起始状态最短路径的所有状态先被访问。因此宽度优先搜索保证可以找到从起始状态到目标状态的最短路径。而且，因为第一次发现的所有状态都是在最短路径上，所以任何第二次遇到的状态都是在相等或更长的路径上。因为沿一条更好的路径不会发现重复的状态，所以算法中施关于重复的状态

很多时候，在 open 和 closed 中保存除了状态名以外的其他信息是有价值的。例如，宽度优先搜索并不像 backtrack 那样在 SL 列表中维护通往目标的当前路径；所有访问过的状态被放入 closed。如果需要一个解路径，则宽度优先算法无法返回这个路径。可通过将每个状态与它的祖先信息一起存储，找到这个解路径。例如，可以把一个状态和它的双亲状态一起存储为一个对（状态，双亲）。如果在对图 3-15 的搜索中这样做，那么第 4 次迭代时 open 和 closed 中的内容将是：

open = [(D, A), (E, B), (F, B), (G, C), (H, C)]

closed = [(C, A), (B, A), (A, nil)]

从这个信息中可以简单地构建出从 A 到 F 的路径（A，B，F）。当发现了目标时，算法可以顺着双亲反向追索出从目标到起始状态的解路径。注意：A 的双亲是 nil（空），表示它是起始状态。这停止了路径的构建。因为宽度优先搜索沿最短的路径寻找每一个状态并且保留每个状态的第一个版本，所以它发现的解是从起点到目标的最短路径。

图 3-17 显示了对 8 格拼图游戏的图进行宽度优先搜索的情况，图中显示的状态就是从 open 中删除和分析的状态。和以前一样，弧对应于空位的上下左右移动。每个状态旁边的数字指出了它从 open 中删除的顺序。当算法终止时还留在 open 中的状态没有显示出来。

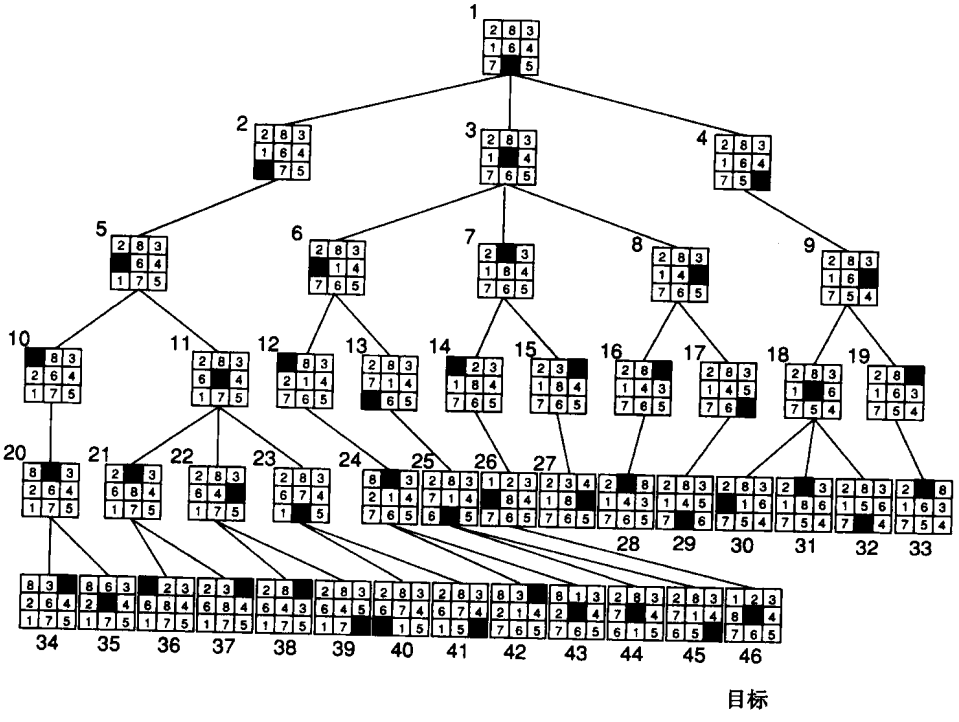


图 3-17　8 格拼图游戏的宽度优先搜索

注：图中显示了各个状态从 open 中删除的顺序

下面，我们建立深度优先算法，它是 3.2.3 节已经给出的回溯算法的简化。在这一算法中，后继状态从 open 的左端加入或删除，open 被当作一个堆栈（也就是一种后进先出（LIFO）的结构）来维护。把 open 组织为一个堆栈使搜索先考虑最近产生的状态，这导致搜索顺序是深度优先的：

跟踪图 3-15 的深度优先搜索的结果如下。每个连续的数字 1，2，3，…代表“while”循环的次数。第一行中给出的是 open 和 closed 的初始状态。假定 U 是目标状态。

1）open = [A]；closed = [ ]。

2）open = [B, C, D]；closed = [A]。

3）open = [E, F, C, D]；closed = [B, A]。

4）open = [K, L, F, C, D]；closed = [E, B, A]。

5）open = [S, L, F, C, D]；closed = [K, E, B, A]。

6）open = [L, F, C, D]；closed = [S, K, E, B, A]。

7）open = [T, F, C, D]；closed = [L, S, K, E, B, A]。

8）open = [F, C, D]；closed = [T, L, S, K, E, B, A]。

9）open = [M, C, D]，因为 L 已经在 closed 中；closed = [F, T, L, S, K, E, B, A]。

10）open = [C, D]；closed = [M, F, T, L, S, K, E, B, A]。

11）open = [G, H, D]；closed = [C, M, F, T, L, S, K, E, B, A]。

依此类推，直到发现了U或open=[ ]。

与宽度优先搜索的情况相同，open列出了已经发现但还没有评估的所有状态（搜索的当前“前线”），closed记录了已经考虑过的状态。图3-18显示了对图3-15所示的图进行第6次深度优先搜索迭代时的情况。open和closed中的内容被突出显示。这个算法也可以像宽度优先搜索那样把每个状态和它的双亲一起存储，这使算法可以重建从起点到目标的路径。

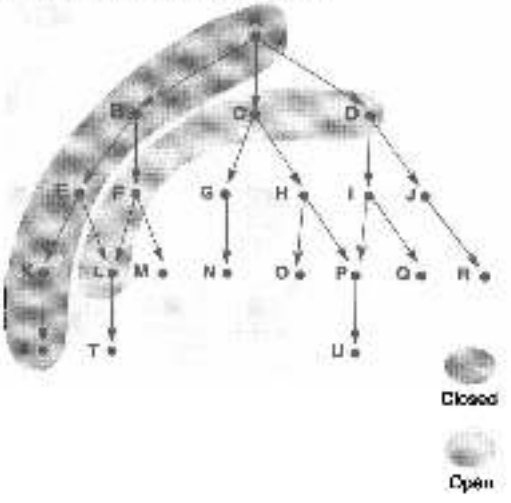


图3-18 对图3-15的深度优先搜索

注：图中突出显示了第6次迭代时的open和closed中的状态

与宽度优先搜索不同，深度优先搜索不保证第一次碰到某个状态时，找到的就是到这个状态的最短路径。在这个算法的后期，可能发现任何状态的不同路径。如果路径长度是问题求解器所关心的，那么当算法碰到一个重复状态时，这个算法应该保存沿最短路径到达的版本。这可以通过把每个状态保存成一个三元组（状态，双亲，路径长度）来实现。当产生孩子时，路径长度值只要加1并把它和这个孩子保存在一起。如果沿多条路径到达了同一个孩子，那么可以用这个信息来保留最好的版本。在第4章的A算法中我们会更详细地讨论这一点。注意：在简单的深度优先搜索中保留一个状态的最佳版本不能保证是沿最短路径到达目标。

图3-19给出了8格拼图游戏的深度优先搜索。正如前面所指出的，这个空间是由4个“移动空位”的规则（上、下、左、右）所产生的。状态旁边的数字指出了状态被考虑的顺序，也就是从open中删除的顺序。当发现目标时还留在open中的状态没有被画出来。另外，在这种搜索中我们将深度界限设为5以防止算法向空间深入时“迷失”。

与选择数据驱动搜索还是目标驱动搜索一样，选取深度优先搜索还是宽度优先搜索依赖于要解决的具体问题。要考虑的主要特征包括发现目标的最短路径的重要性、空间的分支因子、计算时间的可行性、计算空间的可用性、到达目标结点的平均路径长度以及需要所有解还是仅仅需要第一个发现的解。对于以上这些要素，每种方法都有其优势和不足。

宽度优先 因为宽度优先搜索总是在分析第n+1层之前分析第n层上的所有结点，所以宽度优先搜索找到的到目标结点的路径总是最短的。在已经知道存在一个简单解的问题中，宽度优先搜索可以保证发现这个解。不幸的是，如果存在一个不利的分支因子，也就是各个状态都有相对很多个后代，那么组合爆炸可能使算法无法在现有可用内存的条件下找到解。这是由每一层的未展开结点都必须存储在open中这一事实造成的。对于很深的搜索，或状态空间的分支因子很高的情况，这个问题可能变得非常棘手。

宽度优先搜索的空间使用量（以open中的状态数量来衡量）在任何时间都是路径长度的指数函数。如果每个状态平均有B个孩子，那么在一个给定层上的状态数是上一层状态数的B倍。这样在第n层上的状态数为B^n。当宽度优先搜索开始分析第n层时，它要把所有这些状态放入open中。例如，在国际象棋游戏中，当解路径很长时，这可能是不允许的。

深度优先 深度优先搜索可以迅速地深入搜索空间。如果已知解路径很长，那么深度优先搜索不会浪费时间来搜索图中的大量“浅层”状态。另一方面，深度优先搜索可能在深入空间

时“迷失”，错过了到达目标的更短路径，甚至陷入不能到达目标的无限长路径。

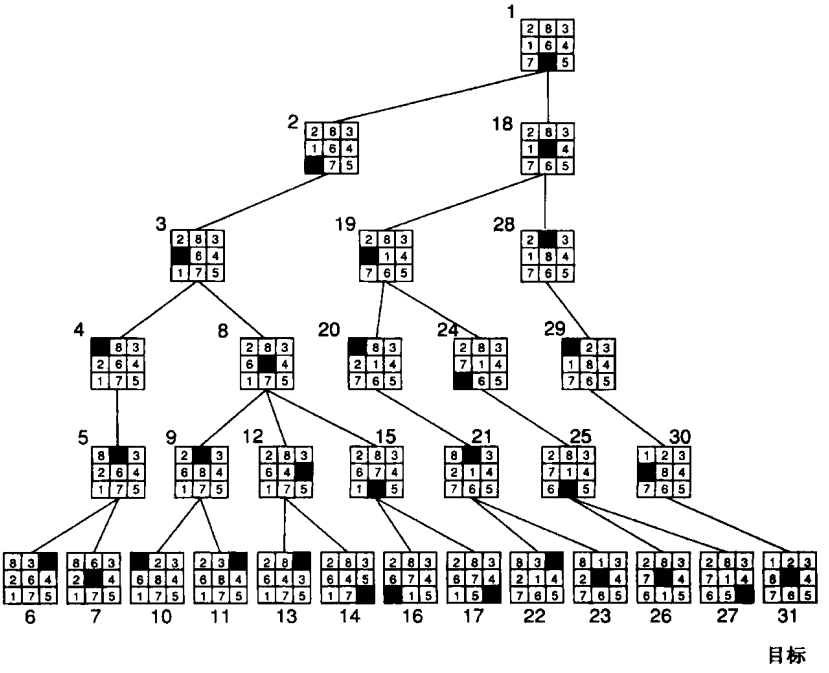


图 3-19　8 格拼图游戏的深度优先搜索（深度界限为 5）

如果要搜索具有很多分支的空间，那么深度优先搜索的效率更高，因为它不必把给定层的所有结点保存到 open 列表中。深度优先搜索使用的空间是路径长度的线性函数。在每一层，open 仅保存一个状态的孩子。如果图中每个状态平均有 B 个孩子，那么要深入到空间的第 n 层需要的总空间量就是 $B \times n$。

选择深度优先搜索还是宽度优先搜索的最佳答案是仔细分析问题空间并向这个领域的专家咨询。例如，对于国际象棋来说，宽度优先搜索就是不可能的。在更简单的游戏中，宽度优先搜索不仅是可能的，而且可能是避免迷失的惟一方法。

3.2.4　迭代加深的深度优先搜索

平衡深度优先搜索和宽度优先搜索的一个很好折中是对深度优先搜索使用一个界限。一旦搜索的深度在某个层次以下，深度界限便强制停止对这条路径的搜索。这形成一种对被搜索空间某个深度的“横扫”。当已知解位于某个深度范围内或当时间有限制时（比如在搜索像国际象棋这样的庞大空间时要限制被考虑的状态数量），具有深度界限的深度优先搜索可能是最合适的搜索。图 3-19 所示的对 8 格拼图游戏的搜索就是深度界限为 5 的深度优先搜索，算法以这一深度扫过空间。

根据这种思想产生的算法弥补了深度优先搜索和宽度优先搜索二者的很多不足。*迭代加深的深度优先*（depth-first iterative deepening）（Korf 1987）对空间进行一种深度界限为 1 的深度优先搜索。如果它找不到目标，便进行另一个深度界限为 2 的深度优先搜索。这样继续下去，每次迭代把深度界限加 1。在每一次迭代中，算法执行一次当前深度界限范围内的完全深度优先搜索。在两次迭代之间不保存任何状态空间信息。

因为这种算法一层层地搜索空间，所以它保证所发现的目标路径是最短的。因为这种方法

在每次迭代中所做的仅仅是深度优先的搜索，所以在任意层次 n 上的空间使用量是 B × n，其中 B 是结点的平均孩子数量。

有趣的是，尽管看起来似乎迭代加深的深度优先方法在效率方面会比深度优先方法和宽度优先方法低，但实际上它的时间复杂度与另两个方法处于同一数量级：O（B^n）。Korf 在 1987 年提出的对此的一个直观解释看起来有些自相矛盾，这个解释如下所述：

> 因为树上某一给定层的结点数量随着深度呈指数增长，所以几乎所有时间都花在最深层上，因此即使以算术递增的速度多次产生较浅层也无所谓。

但是，可以证明这一章讨论的所有搜索策略——深度优先、宽度优先和迭代加深的深度优先——都显示出具有最糟糕的指数时间复杂度。这一结论对于所有的*无信息*搜索算法来说都是正确的。降低这一复杂度的惟一途径是采用启发来引导搜索。*最佳优先搜索*（best-first search）是与刚刚给出的深度优先和宽度优先算法类似的搜索算法。然而，最佳优先搜索对 open 列表（搜索的当前目标）中的状态进行排序，排序的根据是衡量状态的某个启发性指标。最佳优先搜索是第 4 章的主要话题。

3.3　利用状态空间来表示命题演算和谓词演算的推理

3.3.1　逻辑系统的状态空间描述

在 3.1 节中定义状态空间图时，我们注意到结点必须是相互可区别的，每个结点表示解过程的某个状态。可以使用命题演算和谓词演算作为形式说明语言来表示这种差别性，并把图的结点映射到状态空间。而且，可以使用推理规则来创建和描述状态间的弧。通过这种方式，就可以使用搜索来解决谓词演算中的问题，比如判断一个表达式是不是一组给定断言的逻辑结论。

谓词演算推理规则的可靠性和完备性保证了这种基于图的推理所导出的结论的正确性。这种通过与产生解相同的算法来形式证明解完整性的能力是大多数人工智能和基于定理证明的问题求解方法的一个特有属性。

尽管许多问题的状态（比如九宫游戏）用数组这样的数据结构来表示更加自然，但是逻辑的强大性和通用性使大多数 AI 问题求解过程可以使用命题演算和谓词演算描述和推理规则。其他的 AI 表示（比如规则（见第 8 章）、语义网或框架（见第 7 章））采用的搜索规则与 3.2 节给出的规则类似。

例 3.3.1　命题演算

下面举例说明如何把一系列逻辑关系看作对图的定义。第一个例子是关于命题演算的。如果 p，q，r，…是命题，并假定有如下断言：

$q \rightarrow p$
$r \rightarrow p$
$v \rightarrow q$
$s \rightarrow r$
$t \rightarrow r$
$s \rightarrow u$
s
t

根据这组断言和假言推理规则，可以推断出某些命题（p、r 和 u）；但推断不出其他的命题（比如 v 和 q），因为它们实际上不是逻辑派生自这些断言。图 3-20 所示的有向图表达了初始断言与这些推断间的关系。

在图 3-20 中，弧对应于逻辑蕴涵（→）。给定为真的命题（s 和 t）对应于问题的给定数据。这组断言的逻辑结论命题对应于从表示真命题的状态沿一条有向路径可以到达的结点；这样的路径相当于应用一系列假言推理。例如，路径［s，r，p］对应于以下推断序列：

s 与 s→r 得到 r

r 与 r→p 得到 p

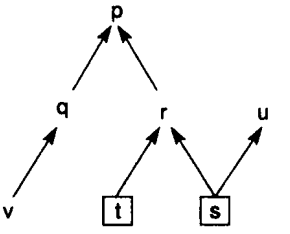


图 3-20　命题演算中的一组蕴涵的状态空间图

有了这种表示，判定一个给定命题是否是某个命题集合的逻辑结论的问题就变成了寻找一条从框起来的结点（起始结点）到这个命题（目标结点）的路径的问题。这样就可以把这种任务看作一种图搜索问题。这里使用的搜索策略是数据驱动的，因为它从已知的真实命题开始向目标前进。另外，也可以对同样的状态空间应用目标驱动搜索策略，从要被证明的命题（目标）入手，然后沿各个弧反向搜索，直到找到支持目标的真命题。我们也可以使用深度优先和宽度优先方式中的任一种来搜索这个推理空间。

3.3.2　与或图

在 3.3.1 节的例子中，所有断言都是像 p→q 这样的蕴涵形式。我们没有讨论在这种图中表示逻辑操作符与（and）和或（or）的方式。要表示这些逻辑操作符所定义的逻辑关系需要对基本的图模型进行扩展，这种扩展后的图被称为与或图（and/or graph）。与或图是描述很多 AI 问题（包括基于逻辑的定理证明程序和专家系统要解决的问题）所产生的搜索空间的一种重要工具。

在形式为 q∧r→p 的表达式中，p 为真的条件是 q 和 r 都为真。在形式为 q∨r→p 的表达式中，q 和 r 中任一个为真便足以证明 p 为真。因为含有析取前提的蕴涵可以写为两个独立的蕴涵，所以后一个表达式经常被写为 q→p，r→p。为了图形化地表示这些不同的关系，与或图把与结点和或结点区别开来。如果蕴涵的前提是由操作符∧连接的，那么称它们为图中的与结点，而且用曲线把到这个结点的弧联合起来。图 3-21 所示的与或图表示了表达式 q∧r→p。

图 3-21 中联合两个弧的连接表示要证明 p 必须 q 和 r 都为真。如果前提是由或操作符连接的，那么它们被当作图中的或结点。或结点的弧不像与结点那样被联合起来（见图 3-22）。这表示任一个前提为真都足以决定结论为真。

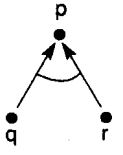


图 3-21　表达式 q∧r→p 的与或图

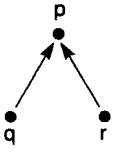


图 3-22　表达式 q∨r→p 的与或图

与或图实际上是超图（hypergraph）的特例，在超图中各个结点是由一系列弧而不是由单一弧连接的。超图的定义如下：

定义（超图）　超图由下列要素组成：

N，结点集合。

H，由有序偶定义的超弧集合，有序偶的第一个元素是来自 N 的一个结点，第二个元素是 N 的一个子集。

普通图是当超图的所有后继结点集合的势为 1 时的特例。

超弧又被称作 k 连接符（k-connector），其中 k 是后继结点集合的势。如果 k = 1，那么可以认为后继集合元素是或结点。如果 k > 1，那么可以认为后继集合元素是与结点。在这种情况下，连接符被画为从父结点到每一个后继结点的单独边；例如可参见图 3-21 中的曲线。

例 3.3.2 与或图搜索

第二个例子也来自命题演算，但是产生的图包含了与后继和或后继。假定下面的命题为真：

a
b
c
$a \wedge b \rightarrow d$
$a \wedge c \rightarrow e$
$b \wedge d \rightarrow f$
$f \rightarrow g$
$a \wedge e \rightarrow h$

这个断言集合产生的与或图如图 3-23 所示。

可以问（答案可以通过对这幅图的搜索演绎出来）的问题有：

1）h 是真的吗？

2）如果 b 不再为真，那么 h 是真的吗？

3）证明 X（某个命题）为真的最短路径（也就是最短的推理序列）是什么？

4）证明命题 p（注意 p 是不被支持的）是假的。这意味着什么？要得到这个结论哪些条件是必要的？

与前面讨论过的常规的图搜索（比如前面讨论的回溯算法）相比，与或图搜索仅需要稍微多保存一些记录。检查或后继的方法与回溯中一样：一旦找到了沿或结点把目标连接到起始结点的一条路径，那么这个问题就解决了。如果一条路径失败了，那么算法可以回溯并试验另一个分支。然而，在搜索与结点时，要证明双亲结点为真就必须证明这个结点的所有与后继为真。

在图 3-23 所示的例子中，判断 h 真值的目标驱动策略首先试图证明 a 和 e 都为真。a 的真值是直接的，但是 e 的真值依赖于 c 和 a 二者的真值；给定这些值为真的。一旦问题求解器已经向下跟踪所有弧到真实的命题，那么就可以在与结点重组这些真值以验证 h 的真值。

另一方面，判断 h 真值的数据驱动策略从已知的事实（c、a 和 b）开始，并根据与或图的约束向这个已知事实集合加入新的命题。e 或 d 可能是被加入到这个事实集合的第一个命题。有了这些新加入的命题，便可能推导出新的事实。这个过程继续下去，直到已经证明了预期目标 h。

观察与或图搜索的一种方式是使用操作符 ∧（由此导致图中的与结点）表示把问题拆散为多个子问题，要证明原来的问题就必须证明所有的子问题。把问题的谓词演算表示中的操作符 ∨ 看作问题求解中的一个选择点，在这一点可以从多个备选问题求解路径和策略中选择其一，只要它们中的任一个是成功的，就足以解决这个问题。

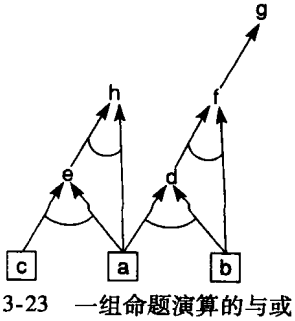


图 3-23 一组命题演算的与或图

3.3.3 进一步的例子和应用

例 3.3.3 MACSYMA

与或图的一个自然例子是对数学函数进行符号积分。MACSYMA 是为数学家广泛使用的一个著名程序。可以把 MACSYMA 的推理过程表示为一个与或图。在进行积分时，一类重要的策略是把表达式分解为每一个可以被独立积分的

子表达式，它们的结果可以用代数方法组合为解表达式。这类策略的例子包括分部分积分的规则以及把和的积分分解为各个单个项的积分和的规则。这些策略代表了把问题分解为独立子问题的思想，因此可以把它们表示为图中的与结点。

另一类策略是通过不同的代数置换来简化一个表达式。因为任何给定的表达式都有很多种不同的置换方法，所以每一种都代表了一种独立的解策略，可以把它们表示为图中的或结点。图3-24画出了这样的问题求解器所搜索的空间。这幅图的搜索是目标驱动的，它从“找到特定函数的积分”入手，逆向搜索直到找到已定义积分的代数表达式。注意：这是明显应该使用目标驱动搜索的一个例子。对问题求解器来说，想要不从目标查询开始逆向寻找形成预期积分的代数表达式的做法实际上是行不通的。

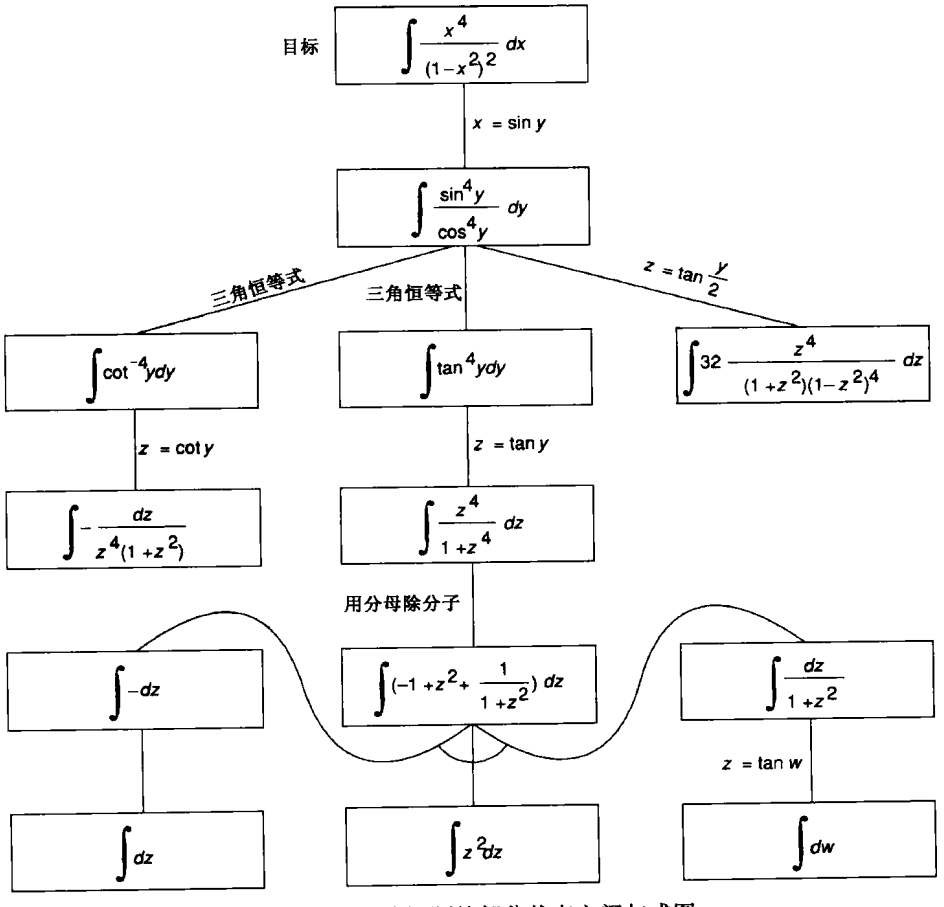


图3-24　积分问题的部分状态空间与或图

[摘自 Nilsson（1971）]

例3.3.4　目标驱动与或搜索

这个例子摘自谓词演算，它代表了对图的目标驱动搜索。要证明的目标是一个含有变量的谓词演算表达式。其中的公理是狗 Fred 和其主人 Sam 之间关系的逻辑描述。我们假定寒冷的一天就是非温暖的一天，以便绕过谓词的等价表达所带来的复杂问题，这个问题会在第7章和第14章中详细讨论。这个例子的事实和规则是用下列语句及其等价的谓词演算来表示的：

1）Fred 是一只柯利狗。

collie(fred)

2）Sam 是 Fred 的主人。
master(fred, sam)

3）这一天是星期六。
day(saturday)

4）星期六很冷。
¬ (warm(saturday))

5）Fred 是训练有素的。
trained(fred)

6）长毛垂耳狗是很好的狗，训练有素的柯利狗也是如此。
∀X [spaniel(X) ∨ (collie(X) ∧ trained(X)) →gooddog(X)]

7）如果一条狗是好狗，而且它有主人，那么他应该和主人在一起。
∀(X, Y, Z)[gooddog(X) ∧ master(X, Y) ∧ location(Y, Z) →location(X, Z)]

8）如果是星期六，而且天气很暖和，那么 Sam 就在公园里。
(day(saturday) ∧ warm(saturday)) →location(sam, park)

9）如果是星期六，而且天气不暖和，那么 Sam 就在博物馆里。
(day(saturday) ∧ ¬ (warm(saturday))) →location(sam, museum)

这个问题的目标是表达式 ∃X location(fred, X)，意思是“fred 在哪里?”反向搜索算法从分析建立这一目标的方式入手：“如果 fred 是一条好狗，而且 fred 有主人，而且 fred 的主人在某个位置，那么 fred 也在这个位置。”然后分析这个规则的前提：gooddog（好狗）的条件是什么？依此类推。继续这个过程，便建立起如图 3-25 所示的与或图。

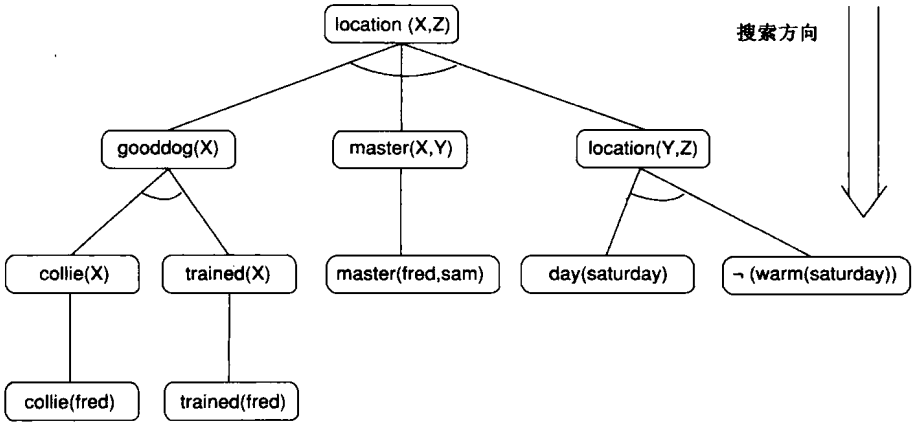


图 3-25　表明 fred 在博物馆的解子图

下面我们仔细地分析这种搜索，原因是这个例子里使用谓词演算进行了目标驱动搜索，而且演示了合一对产生搜索空间的作用。这个问题要解决的目标是“fred 在哪里?”更形式化地讲，这个问题可以被看作求出变量 X 的一个置换，如果存在这样的一个置换，那么在这个置换下 location(fred, X) 是初始断言的逻辑结论。

既然要决定 Fred 的位置，那么就该分析以 location 为结论的子句，第一个就是子句 7。然后把这个子句的结论 location（X, Z）通过置换 {fred/X, X/Z} 和 location（fred, X）合一。这个规则的前提组成了最顶端目标的与后继，对其使用同样的置换便得到：

gooddog(fred) ∧ master(fred, Y) ∧ location(Y, X)

可以把以上表达式的含义解释为：找到 fred 的方式是要看 fred 是否是一条好狗，并查明 fred 的主人是谁以及其主人在哪里。于是可以使用这 3 个子目标来代替初始目标。这些都是与结点，因此必须将它们全都求解出来。

为了求解这些子目标，问题求解器首先判断 fred 是否是一条好狗。利用置换{fred/X}，这正好与子句 6 相匹配。子句 6 的前提是两个表达式的或：

spaniel(fred) ∨ (collie(fred) ∧ trained(fred))

这两个或结点中第一个是 spaniel(fred)。数据库中不包含这个断言，所以问题求解器必须认为它是假的。另一个或结点是(collie(fred) ∧ trained(fred))，也就是说，fred 是 collie 且 fred 被专门训练过。二者都需要为真，而这正好是子句 1 和子句 5。

这证明了 gooddog(fred)是真的。接下来问题求解器分析子句 7 的第二个前提：master(X, Y)。使用置换{fred/X}，master(X, Y)就变成了 master(fred, Y)，再把它与 master(fred, sam)的事实（子句 2）合一。这产生了合一置换{sam/Y}，这一置换也把子句 7 的第三个子目标赋值为 sam，形成了新的子目标 location(sam, X)。

在求解这个子目标时，假定问题求解器是按顺序试验规则的，目标 location(sam, X)会先与子句 7 的结论合一。注意：这时我们是在用不同的 X 绑定试验同一条规则。回忆在第 2 章中我们曾说明，X 是哑元，因此可以取任何名字（任何以英文大写字母开头的字符串）。因为任何变量名的含义范围是局限在它所出现的子句中的，所以谓词演算中根本不存在全局变量。对此的另一种解释是，变量值是作为参数传入其他子句的，没有固定的（内存）位置。因此在这个例子的不同规则中 X 的多次出现是代表不同的形式参数（见 14.3 节）。

在利用这些新的绑定来求解规则 7 的前提时，问题求解器会因 sam 不是 gooddog 而失败。于是，搜索会回溯到目标 location(sam, X)并试验下一个分支，即规则 8 的结论。这也会失败，导致再次回溯并与子句 9 的结论 location(sam, museum)合一。

因为子句 9 的前提是由一组断言（子句 3 和子句 4）支持的，所以得出子句 9 的结论为真。这个最终的合一走了一圈又回到树顶，用 location(fred, museum)给出了 ∃X location(fred, X)的最终答案。

仔细分析图的目标驱动搜索，并将其与例 3.3.2 中的数据驱动搜索相比较是很有意义的。本书的其他部分包含了对这个问题的进一步讨论：在下个例子中更细致地比较了搜索图的这两种方法，但是全面的详细讨论是在第 6 章中的“产生式系统”和第四部分中的“专家系统应用”中给出的。这个例子蕴涵的另一个要点是子句的顺序影响了搜索的顺序。在上面的例子中，依次试验了多个 location 子句，回溯搜索排除了那些无法证明为真的子句。

例 3.3.5　再谈财务顾问程序

在第 2 章的最后一个例子中我们使用谓词演算表示了一系列规则，目的是给出投资建议。在那个例子中，我们用假言推理来为某个投资个体推断合适的投资方案。当时我们没有讨论程序是如何给出合适建议的。当然这是一个搜索问题；本例演示了一种实现基于逻辑的财务顾问程序的途径，采用的是目标导向的带回溯的深度优先搜索。这里的讨论仍然使用 2.4 节中的谓词；在此不再赘述。

假定某个投资个体要供养两个人，有 20 000 美元存款，30 000 美元稳定收入。正如第 2 章所讨论的，我们可以把描述这些事实的谓词演算表达式加到本来的谓词演算表达式集合中。另一种做法是，程序开始搜索时不使用这些信息，当需要时再要求用户加入。这样做的优点是不需要

询问和表示那些与问题解没有必要关系的数据。这种方法是经常在专家系统中使用的，本例中也使用了这种方法。

在咨询中，我们的目标是找到一种投资方案；因此可以将其表示为谓词演算表达式∃X investment(X)，其中X是我们需要绑定的目标变量。有三条规则（1、2和3）给出了投资结论，所以我们的查询要与这三个规则的结论合一。如果我们选择规则1开始探索，那么前提savings_account(inadequate)就成为子目标，也就是说接下来要展开它的孩子结点。

在产生savings_account(inadequate)的孩子结点时，惟一可以应用的规则就是规则5。这产生了与结点：

amount_saved(X) ∧ dependents(Y) ∧ ¬ greater(X, minsavings(Y))

如果我们打算从左到右分析这些前提，那么amount_saved(X)将成为第一个子目标。因为系统不包含这个子目标，所以它会向用户查询。当加入了amount_saved(20000)时，用20000置换X进行合一，这个子目标便完成了。注意：因为正在搜索的是与结点，所以如果这里失败了就不必再分析表达式的其余部分。

类似地，子目标dependents(Y)会导致一个用户查询，然后把回答dependents(2)加入到逻辑描述中。置换{2/Y}可以把这个子目标与表达式匹配。接下来搜索要演算以下表达式的真值：

¬ greater(20000, minsavings(2))

求出的结果为假，导致整个与结点失败。然后搜索回溯到双亲结点savings_account(inadequate)并试图找到另一种方式证明这个结点为真。这对应于产生搜索中的下一个孩子。因为没有其他的规则可以得出这个子目标，所以搜索失败返回到顶层的目标investment(X)。结论与目标合一的下一个规则是规则2，于是又产生了新的子目标：

savings_account(adequate) ∧ income(adequate)

继续搜索，根据规则4的结论可以证明savings_account(adequate)为真，根据规则6的结论可以得出income(adequate)。这个搜索的其余部分留给读者，最终得到的与或图显示在图3-26中。

例3.3.6 英语语法分析程序和语句生成程序

这个最后的例子不是关于谓词演算的，而是由用于语法分析的一系列改写规则组成，这些规则是英语语法的一个子集。改写规则把一个表达式转换成另一个表达式，做法是把箭头（↔）一侧的模式替换为另一侧的模式。例如，可以订一个改写规则集把用一种语言（比如英语）写的表达式转换成另一种语言（可能是法语或一个谓词演算子句）。本例给出的改写规则把英语语句的一个子集转换成更高层次的语法结构，比如名词短语、动词短语以及语句。可以使用这些规则来解析单词序列，也就是判断它们是不是合法语句（即判断语法正确还是不正确），并对语句的语言结构建模。

以下是英语语法的一个简单子集，包含5条规则：

1）sentence↔np vp（句子是名词短语后面跟有动词短语）

2）np↔n（名词短语是名词）

3）np ↔art n（名词短语是冠词后面跟有名词）

4）vp↔v（动词短语是动词）

5）vp↔v np（动词短语是动词后面跟有名词短语）

除了这些语法规则，语法分析程序还需要一个字典，里面含有语言中的单词。这些单词被称

为语法的终端终结符。它们是由改写规则按这些单词的语言成分定义的。在下面的“字典”中，“a”、“the”、“man”、“dog”、“likes”和“bites”是我们的简单语法的终端终结符：

6）art↔a

7）art↔the（“a”和“the”是冠词）

8）n↔man

9）n↔dog（“man”和“dog”是名词）

10）v↔likes

11）v↔bites（“likes”和“bites”是动词）

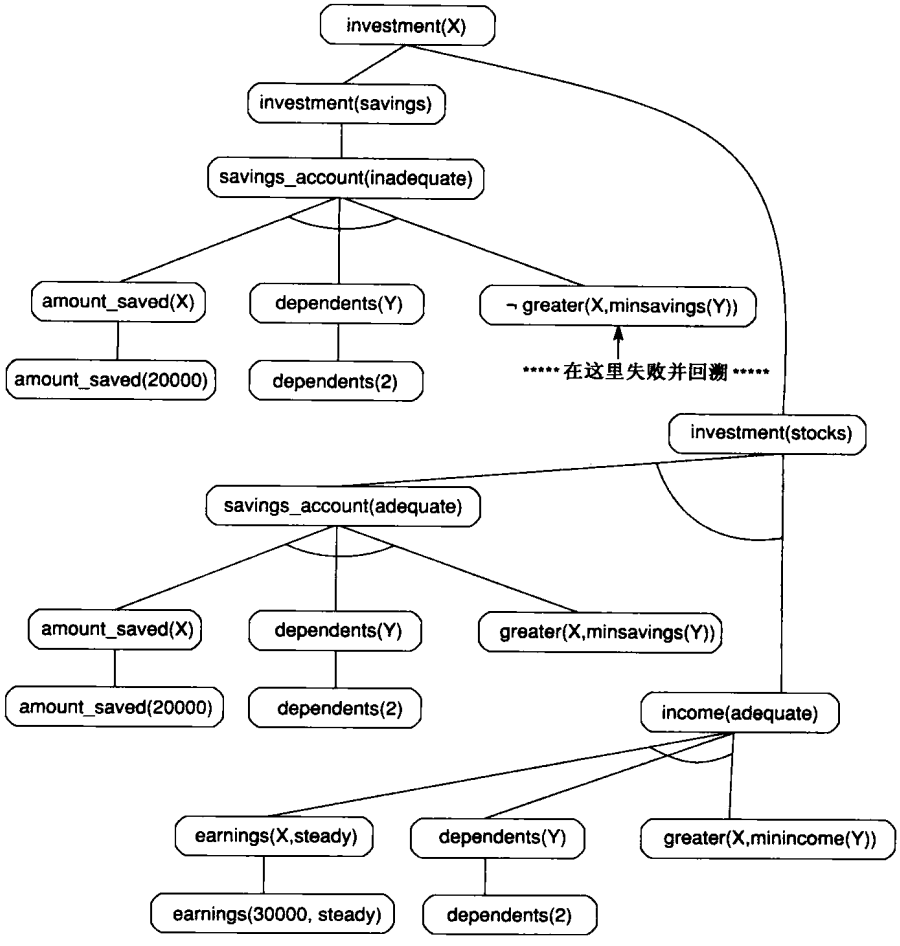


图 3-26　财务顾问程序搜索的与或图

这些改写规则所定义的与或图如图 3-27 所示，其中 sentence 是根。改写规则左边的元素对应于图中的与结点。同一结论的多个规则形成了图中的或结点。注意：这幅图的叶子（或末端）结点就是语法中的英文单词（所以把单词称为终端终结符）。

根据语法，一个表达式为合式表达式的条件是：这个表达式完全是由终结符号构成的；存在一系列置换可以利用改写规则把表达式简化为符号 sentence。另外，也可以把这个条件看作构建一个语法解析树（parse tree），表达式的单词是树的叶子，符号 sentence 是树的根。

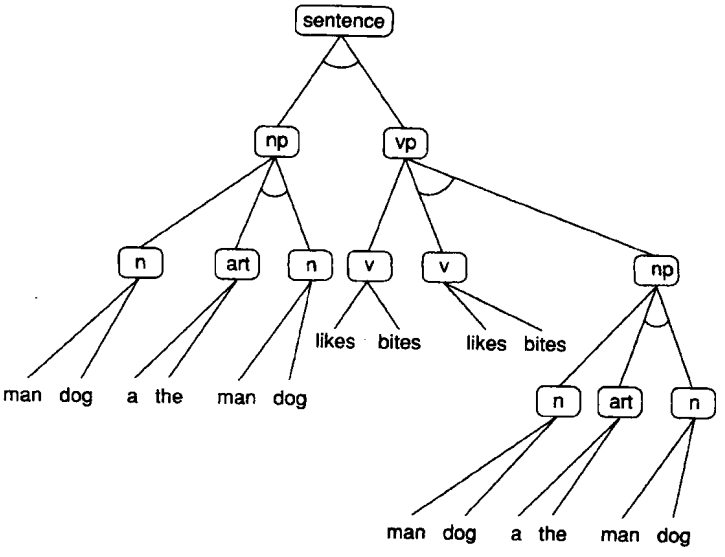


图 3-27 例 3.3.6 中语法的与或图

注：为了画图简单，我们使某些结点（np、art 等）多次出现

例如，我们可以通过构建图 3-28 所示的语法解析树来解析“the dog bites the man”这句话。这个树是图 3-27 所示的与或图的一个子树，由搜索图的过程构建。*数据驱动的语法解析*算法可以这样实现这种解析：把改写规则的右边部分与语句中的模式匹配，按规则的书写顺序试验这些匹配。一旦发现了一种匹配，就用规则左边的模式代替表达式中与规则右端匹配的那一部分。继续这个过程，直到这个语句简化为符号 sentence（表明解析成功）或所有规则都已应用过（表明解析失败）。解析“the dog bites the man”的过程如下：

1）第一个匹配的规则是规则 7，把 the 改写为 art。于是得到：art dog bites the man。

2）下一轮迭代找到匹配的规则还是规则 7，得到 art dog bites art man。

3）接下来匹配的是规则 8，得到 art dog bites art n。

4）匹配规则 3，得到 art dog bites np。

5）匹配规则 9，得到 art n bites np。

6）又可以应用规则 3，得到 np bites np。

7）应用规则 11，得到 np v np。

8）应用规则 5，得到 np vp。

9）匹配规则 1，将语句简化为 sentence，表达式正确。

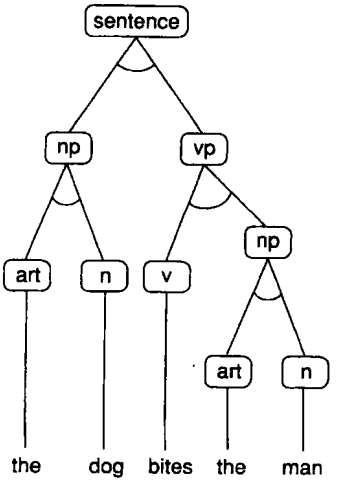


图 3-28 句子“The dog bites the man”的解析树

注：这是图 3-27 的一个子图

上面的例子实现了数据导向的深度优先解析，因为它总是向表达式应用最高层的规则。例如，在把 bites 简化为 v 前把 art n 简化为 np。也可以用目标导向的方式来实现语法解析，把 sentence 作为起始字符串，寻找一系列匹配规则左边的替代模式，产生一系列匹配目标语句的终端终结符。

语法解析是很重要的，它不仅可用于自然语言理解（第 15 章），而且可用于计算机语言的

编译器和解释器（Aho and Ullman 1977）。文献中有很多用于解析各种语言的算法。例如，很多目标导向的语法解析算法预测（look ahead）输入流中的内容，以判断接下来要应用的规则。

在这个例子中，我们使用了一种非常简单的途径以无信息的方式搜索与或图。其中，很有趣的是搜索的实现方法。该方法保持当前表达式的记录并试着依次寻找匹配的规则，这是利用产生式系统实现搜索的一个例子。利用产生式系统实现搜索是第6章的一个主要论题。

也可以使用改写规则根据语法规则来产生合法语句。可以使用目标驱动搜索来生成语句，从 sentence 开始把它作为顶层目标，当不再有可以应用的规则时结束。这产生了终结符号组成的字符串，这个串是符合语法的合法语句。例如：

sentence 是 np 后面跟有 vp（规则1）。

用 n 代替 np（规则2），得到 n vp。

man 是第一个可用的 n（规划8），得到 man vp。

现在 np 已经满足要求，接下来替代 vp。规则3用 v 替代 vp，于是得到 man v。

规则10用 likes 代替 v。

于是得到了第一个可接受的语句 man likes。

如果想要产生所有可接受的语句，那么可以系统地重复这个搜索直到试验了所有可能，穷举搜索了整个状态空间。这样产生的句子包括：a man likes，the man likes，等等。通过穷举搜索，产生了大约80个正确的语句。这其中包括像 the man bites the dog 这样语义异常的句子。

可以通过各种方式把语法解析和语句生成放在一起使用来解决不同的问题。例如，如果希望找出补全字符串“the man”的所有句子，那么可以给问题求解器一个不完整的字符串 the man…。问题求解器可以先按数据驱动的方式向上探索，以产生补全语句的规则（规则1），其中 np 被 the man 代替；然后再以目标驱动的方式来求出所有可能的 vp 以补全这个句子。这会产生像这样的语句：man likes，the man bites the man，等等。再次强调，这个例子仅考虑语法的正确性。语义问题（这个字符串是否可以被正确映射到某个“世界”）完全是另一回事。第2章分析了为表达式构建以逻辑为基础的语义的问题；对于自然语言中的要检验的句子，这个问题要复杂得多，我们将在第15章中讨论这部分内容。

在下一章中我们将讨论如何使用启发来使搜索集中在状态空间尽可能小的部分。第6章讨论了产生式系统和其他的一些软件“体系结构”，用来控制状态空间搜索。

3.4　结语和参考文献

本章介绍了状态空间搜索的基本理论，即利用图论来分析问题求解策略的结构和复杂度。本章还比较了数据驱动推理和目标驱动推理，以及深度优先搜索和宽度优先搜索。与或图使我们可以应用状态空间搜索来实现以逻辑为基础的推理。

很多关于计算机算法的教材讨论了基本的图搜索理论，其中包括：Thomas Cormen、Charles Leiserson 和 Ronald Rivest（1990）所著的《Introduction to Algorithms》，Paul Helman 和 Robert Veroff（1986）所著的《Walls and Mirrors》，Robert Sedgewick（1983）所著的《Algorithms》以及 Ellis Horowitz 和 Sartaj Sahni（1978）所著的《Fundamentals of Computer Algorithms》。有限自动机在 Lindenmayer 和 Rosenbery（1976）的文章中有介绍。关于与或搜索的更多算法将在第14章“自动推理”中给出。

Alan Newell 和 Herbert Simon（1972）所著的《Human Problem Solving》中介绍了使用图搜索来对智能问题求解建模的方法。讨论搜索策略的人工智能教材包括：Nils Nilsson 的《Artificial Intelligence》（1998）、Patrick Winston 的《Artificial Intelligence》（1992）以及 Eugene Charniak 和

Drew McDermott（1985）所著的《Artificial Intelligence》。Judea Pearl（1984）所著的《Heuristics》介绍了搜索算法并为我们要在第 4 章中讨论的内容打了基础。开发新的图搜索技术是每年 AI 会议的经常性主题。

3.5　习题

1. 汉米尔顿路径（Hamiltonian path）就是正好使用图中每个结点一次的路径。存在这种路径的必要条件是什么？在哥尼斯堡地图中是否存在这样的路径？
2. 问题：一个农夫带着狼、羊还有菜来到河边要过河。河边有一条船，只有农夫能划船，而且每次只能有两个物品在船上（划船者也是物品）。如果把狼和羊单独放一起，狼将吃掉羊，如果把羊与菜单独放一起，羊将吃掉菜。设计一个过河的序列，使得四样物品都能安全地运到河对岸。
   请给出农夫、狼、山羊和卷心菜问题的图表示，用结点表示这个世界中的状态。例如，农夫和山羊在西岸，狼和卷心菜在东岸。讨论宽度优先和深度优先搜索这个空间的优势。
3. 建立一个有限状态接受器来识别下面的二进制数字字符串：
   a）包含“111”。
   b）以“111”结尾。
   c）包含“111”但不包含多于 3 个连续“1”。
4. 给出一个最近邻策略无法找到最优路径的巡回推销员问题的实例。提出解决这个问题的另一种启发式方法。
5. 针对图 3-29，“徒手运行”回溯算法。从状态 A 开始，记录 NSL、SL、CS 等的逐项值。
6. 选择一种程序设计语言实现回溯算法。
7. 判断是目标驱动搜索还是数据驱动搜索更适合解决下面的每个问题。并说出你的理由。
   a）诊断汽车的机械故障。
   b）你遇到一个人，她自称是你的远房堂妹，你们有一个共同的名叫 John Doe 的前辈。你想要核实她的说法。
   c）另一个人自称是你的远房堂弟，他不知道你们的共同前辈的名字但知道你们的亲缘不超过八代。你想找到这个前辈或判定根本不存在这样的前辈。
   d）平面几何的定理证明程序。
   e）一个分析声纳读数和解释这些读数的程序，比如可以辨别大的潜水艇、小的潜水艇、鲸鱼和鱼群。
   f）一个专家系统用来帮助人们按种、属等来分类植物。

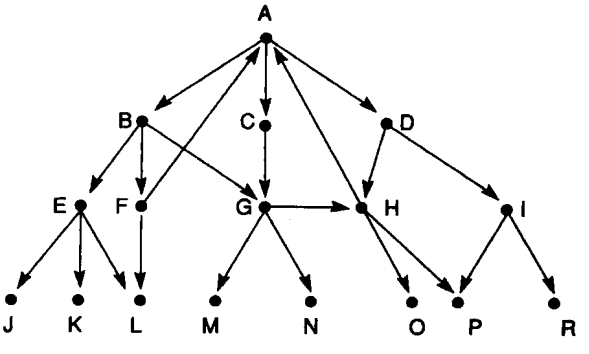


图 3-29　要搜索的图

8. 为练习 7 中的例子选取深度优先搜索或宽度优先搜索方法中的一种，并说明选择理由。
9. 编写一个用于与或图的回溯算法。
10. 以数据驱动方式跟踪例 3.3.4 中目标驱动的 gooddog 问题的执行过程。
11. 再给出一个与或图搜索的例子，并展开一部分搜索空间。
12. 跟踪例 3.3.5 中财务顾问问题的数据驱动方式执行过程。针对的案例是这样一个投资个体：要供养 4 个人，有 18 000 美元存款，每年有25 000美元稳定收入。把这个问题与文中的例子进行比较，并据此提出一种解决这种问题的通用“最佳”策略。
13. 向例 3.3.6 的英语语法中加入适用于形容词和副词的规则。
14. 向例 3.3.6 的英语语法中加入适用于（多个）介词短语的规则。
15. 向例 3.3.6 的英语语法中加入语法规则，使之允许这样的复合语句：

    sentence↔sentence AND sentence

第4章　启发式搜索

当问题和问题空间确定后，符号系统所要解决的任务就是如何使用有限的处理资源来产生可能解，直到发现一个可以通过（问题所定义）检验的解。如果符号系统可以对可能解的产生顺序进行某种控制，并对这个产生顺序加以组织以使高可能性的解先出现，那么这是非常理想的。如果符号系统成功地做到了这一点，那么它便展示出了智能。对于一个处于有限资源下的系统来说，智能体现在明智地选择下一步该做什么……

——纽维尔和西蒙，1976年度图灵奖获奖演说

我一直在寻觅……
寻觅……哦，
寻觅逝去的一切……

——Lieber and Stoller

4.0　简介

乔治·波利亚（George Polya）把*启发*定义为“对发现和发明的方法和规律的研究”（Polya 1945）。这一含义可以追溯到这个词的希腊词根——动词 eurisco，意为“我发现了”。当阿基米德在洗澡时突然发现了验证皇冠是否为纯金的方法时，他喊道“Eureka!”，意思是“我已经找到了!”。在状态空间搜索中，*启发*被形式化定义为一些规则，这些规则能选取状态空间中最可能产生可接受解的分支。

AI 问题求解器在以下两种基本情况下将采用启发：

1）因为在问题陈述或现有数据中存在固有的模糊性，所以问题可能没有精确解。医疗诊断就是这样的一个例子。对于已知的一系列症状，可以有很多种可能的导致原因；因此，医生必须使用启发来选取最可能的诊断并制定治疗计划。视觉是存在固有不精确性的另一个例子。视觉画面经常是存在歧义的，允许对对象的连通性、范围和朝向有多种解释。视觉幻像就是这种模糊性的一个例子。视觉系统通常使用启发从对给定场景的几种可能解释中选取最可能的解释。

2）问题可能有精确解，但是找到这个解的运算开销可能是让人难以承受的。在许多问题（比如国际象棋）中，状态空间的组合增长是爆炸性的，可能状态的数量随着搜索深度的增大呈指数或阶乘增长。在这些情况下，类似深度优先或宽度优先这样的穷举性的*蛮力搜索*技术可能无法在可行时间限度内找到解。启发通过引导搜索沿最有希望的路径穿越空间来降低这种复杂性。启发式算法把没有希望的状态以及这些状态的后代从考虑中排除，这样便可以克服这种组合爆炸，找到可接受的解。

不幸的是，和所有有关发现和发明的规律一样，启发也是很容易失误的。启发仅是对问题求解过程中下一步要采取的措施的一种精明猜测。很多时候，它是建立在经验或直觉基础之上的。因为启发所使用的信息是有限的，比如当前情景的知识或对当前 open 列表中状态的描述，所以它很少能够预测出距离当前搜索位置很远的状态空间的准确状态。启发可能引导搜索算法得到一个并非最优的解或根本找不到任何解。这是启发式搜索的固有局限性，是不会因使用“更好的”的启发或更高效的搜索算法而消除的（Garey and Johnson 1979）。

长期以来，启发和实现启发式搜索的算法的设计一直是人工智能研究的一个核心问题。博弈和定理证明是人工智能的两个最古老应用，而这两个问题都需要用启发来修剪可能解的空间。

分析一个数学问题定义域中可以做出的每一种推理或国际象棋中可以采取的每种可能移动都是不可行的。启发式搜索经常是惟一可行的方案。

最近，专家系统的研究已经证实了启发作为问题求解的关键要素具有不可替代的重要性。当人类专家解决问题时，他先分析现有的信息然后做出决策。人类专家用以高效解决问题的“经验法则”很大程度上来说就是启发式的。如我们在第8章所见，专家系统设计者的一个主要任务就是如何抽取并形式化这些启发。

可以把启发式搜索看作由两方面组成：启发度量；及使用这个度量进行空间搜索的算法。在4.1节我们将用爬山和动态规划算法实现启发。在4.2节我们将介绍一个最佳优先搜索的算法。4.3节将介绍启发的设计和评估启发的有效性。4.4节将介绍博弈论中的启发。

考虑图Ⅱ-5中的九宫游戏的启发。如果使用穷举搜索，那么它的组合数尽管很高，但仍是可控制的。对于首次的9种走法，每一种有8种可能反应，这8种中的每一种又有7种后续走法，依此类推。通过简单分析可以知道，穷举搜索九宫游戏要考虑的状态总数是$9\times8\times7\times\cdots$，即9!。

通过对称抵消可以大大缩小这个状态空间。只要把棋盘做对称变换，就会发现很多种格局实际上是等价的。所以，事实上初始的走法并非9种，而是3种：放到一角、放到一边的中心以及放到棋盘的中央。在第二层上的对称抵消进一步把穿越空间的路径数下降到$12\times7!$，如图4-1所示。可以用数学恒等式来描述博弈空间中的这种对称性，当存在这样的恒等式时，经常可以用它们来大大缩小搜索范围。

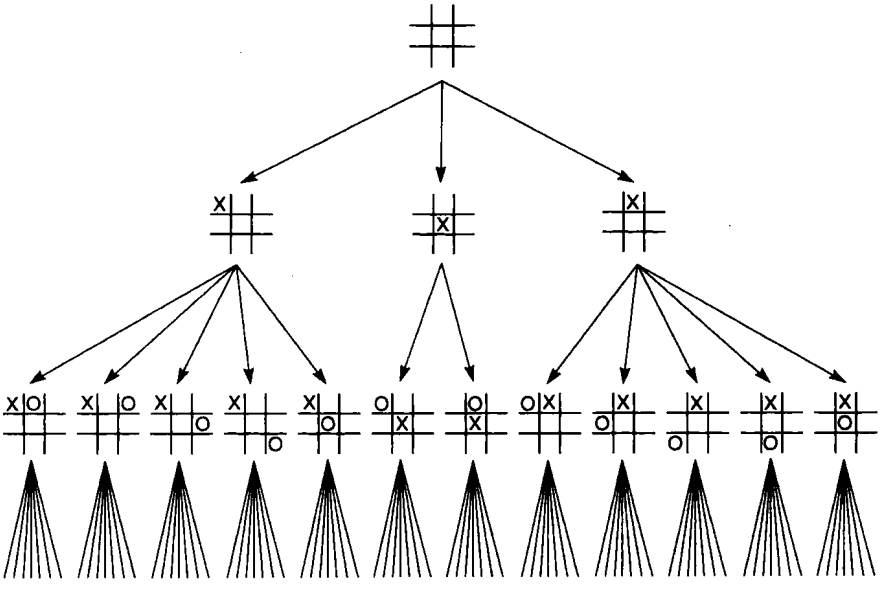


图4-1 经过对称抵消后的九宫游戏状态空间的前三层

不过，简单的启发可以排除搜索空间的绝大部分，比如我们可以使用这一启发：移动到X方胜利路线最多的状态（在图4-2中这种做法衡量了九宫游戏的三个初始状态）。万一几个状态具有相同数量的可能胜利路线，那么就取第一个发现的路线。接下来，算法只要选择并移向具有最高启发值的状态就可以了。该情况下X选择了下在棋盘的中间。注意，这样做不仅排除了其他两种候选状态，而且还排除了它们的所有后继。从图4-3可以看出，仅仅第一次移动就剪除了2/3的空间。



图 4-2　应用到九宫游戏中前三个状态的“最多胜利路线”启发

第一次移动后，对手可能选取另两种移动中的一种（如图 4-3 所示）。无论选取了哪一种移动，都可以把这个启发应用到新产生的状态，也就是再次利用“最多胜利路线机会”启发从可能移动中做出选择。搜索便这样继续下去，每一次移动仅评估单一结点的孩子，不需要穷举搜索。图 4-3 显示了前三步搜索后的情况。图中还标出了各个状态的启发值。尽管难以精确计算这种用于九宫游戏的“最多胜利”策略必须要分析的状态数量，但是可以大致计算出一个上限，因为一盘棋一个人最多有 5 次移动，每次有 5 种选择。事实上，实际的状态数要少得多，因为棋盘的填充减少了我们的选择。这个大致的上限（25 种状态）相对 9! 来说改善了 4 个数量级。

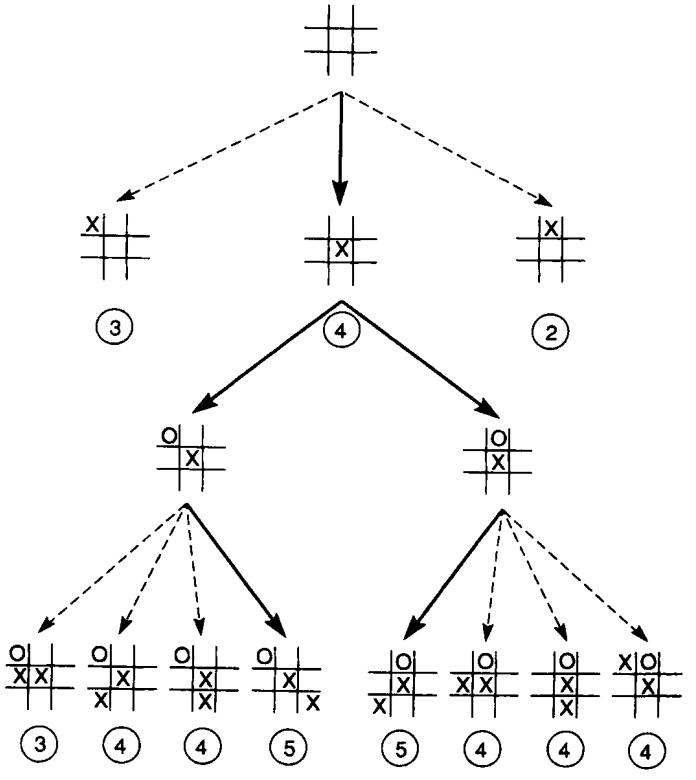


图 4-3　九宫游戏中利用启发减小的状态空间

下一节（4.1 节）将介绍实现启发式搜索的两个算法：爬山和动态规划。4.2 节将介绍最佳优先搜索中优先级队列的使用。4.3 节将讨论有关启发式搜索的一些理论问题，比如可采纳性（admissibility）和单调性（monotonicity）。4.4 节将分析如何使用极小极大（minimax）和 α-β 剪枝把启发应用到两人博弈中。最后一节将分析启发式搜索的复杂度并再次强调它在智能问题求解中的核心作用。

4.1 爬山法和动态规划法

4.1.1 爬山

实现启发式搜索的最简单方式是采用一种称为*爬山*（hill-climbing）的过程（Pearl 1984）。爬山策略在搜索中先扩展当前状态，然后再评估它的孩子。而后选择“最佳的”孩子做进一步的扩展；而且在这个过程中既不保留它的兄弟姐妹，也不保留它的双亲。之所以叫爬山策略是因为一个积极的盲人登山者可能使用这种策略：沿尽可能陡峭的路向上爬，直到无法继续向上。因为这种策略不保存任何历史记录，所以它不具有从失败中恢复的能力。在4.0节我们曾介绍过九宫游戏中一个爬山的例子，即“选择最有可能胜利的状态”。

爬山策略的一个主要问题是容易陷入*局部最大值*。如果这种策略到达了一个比其任何孩子都好的状态，那么它便停止。如果这个状态不是目标，仅是一个局部最大值，那么算法就无法找到最优解。也就是说，在某些有限的情况中其性能可能很好，但由于它无法把握整个空间的形状，所以它可能永远不能到达全局最佳值。8格拼图游戏中就存在局部最大值的例子。很多时候，为了把某一张牌移动到它的目的地，不得不把其他已经在目标位置的牌移开。这对完成拼图目标来说是必须的，但这暂时性地恶化了棋盘状态。因为从绝对角度来讲，“较好的”未必是“最好的”，所以没有回溯或其他恢复机制的搜索方法无法辨别局部和全局最大值。

图4-4是陷入局部最大困境的一个例子。假设，在搜索空间中搜索时，为找到最大状态值，我们到达了状态X。从X的儿子、孙子和曾孙的值可以看出，爬山方法甚至在使用多层预判的情况下也会搞错。解决这个问题的方法有很多种，比如随机扰动评估函数，但是通常没有办法保证利用爬山技术可以得到最佳性能。Samuel（1959）的西洋跳棋程序使用了爬山算法的一种变体。

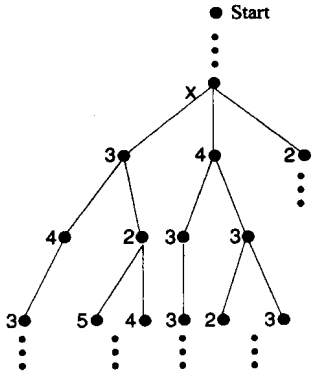


图4-4 使用3层预判的爬山方法遇到的局部最大化问题

Samuel的程序在当时（1959年）是非凡的，特别是还受当时的计算机能力的限制。Samuel的程序不仅为走棋应用了启发式搜索，而且用了算法来优化有限内存的使用，还实现了简单形式的学习。实际上，这个程序开创的许多技术至今还用在很多博弈和机器学习程序中。

Samuel的程序用几个不同启发尺度的带权和来评估棋盘的状态：

$$\sum_i a_i x_i$$

这个和中的x_i代表一些棋盘特征，比如棋子的优势、棋子的位置、棋盘中心的控制、牺牲棋子换取优势的机会，甚至是（一个棋手的）棋子关于棋盘轴线的惯性力矩。这里x_i的系数是可以调整的权，这些权取值的标准是尽可能表现出每个因素对整个棋局的重要性。因此，如果棋子的优势比控制棋盘中心更重要，那么棋子优势的系数会把这一点反映出来。

在博弈中，Samuel程序会预判搜索空间的一定层数（通常受计算机的空间和时间条件限制），并使用评估多项式来评估预判层的所有状态。它利用*极小极大*（见4.3节）过程的一种变体来向上传播这些值。然后移动到下一个最佳状态；等对手移动了之后，再对新的棋盘状态重复以上过程。

如果评估多项式产生的一系列移动都很失败，那么程序会调整它的系数以努力改善其性能。较大系数的评估项对失败承担主要责任，它们的权被减小；同时较小的权被增大以扩大这项评估的影响。如果程序胜利了，那么进行相反的操作。可以通过让这个程序与人类或与它自己的另一个版本对弈来训练它。

Samuel 程序采用一种爬山方法来进行学习，即通过局部改进评估多项式来提高程序的性能。这样，可以使该程序的性能达到非常高的水平。Samuel 通过检查各个加权启发尺度的有效性来分析爬山搜索中的一些局限性，并把这些不太有效的特征替换掉。当然，这个程序也还有一些有趣的局限性。比如，因为采用有限的全局策略，所以评估函数容易导致进入陷阱（即局部最优）。程序的学习部分也比较脆弱，容易因对手走法的不一致而出错；例如，如果对手使用变化非常大的策略，或者干脆采用愚蠢的走法，那么评估多项式的权会变得杂乱无章，导致性能全面退化。

4.1.2　动态规划

动态规划（dynamic programming，DP）有时也称为*前前后后*（forward-backward）算法，当使用概率时称为 Viterbi（*韦特比*）算法。动态规划由 Richard Bellman（1956）提出，解决由多个互相影响互相关联的子问题构成的问题中的受限内存搜索。动态规划在大问题的解决方案中不断记录和重用已被解决的子问题的解决方案。一个例子是在斐波那契（Fibonacci）数列中重用子数列。为重用而采用子问题缓存的技术有时称为*记录部分子目标解决方案*。动态规划的成果是一个重要算法，经常在字符串匹配、拼写检查和自然语言处理相关领域中使用（见 9.4 节和 14.4 节）。

我们用两个例子来演示动态规划，这两个例子都来自文本处理。第一个例子是找到两个字符串的最佳全局对齐；第二个例子是找到两个字符串间的最小编辑差别。假设我们要找到字符串 BAADDCABDDA 和 BBADCBA 可能的最佳对齐。几种对齐中的一种选择可能是：

```
BAADDCABDDA
BBA DC B  A
```

动态规划需要一个数据结构来跟踪与当前正在处理的状态相关的子问题。我们用的是数组。因为初始化的需要，数组的每一维数的尺寸都比字符串的长度大 1，在我们的例子中是 12×8，如图 4-5。数组中每个行列相交处元素的值表示匹配过程中当前点的最佳对齐。对当前状态有三种可能的代价：如果短字符串中的字符为了最佳对齐被向前移动，那么代价为 1 并用数组的列下标值记录。如果新字符被插入，那么代价为 1 并用数组的行下标值记录。如果要对齐的字符不同，那么移动并插入，代价为 2；如果相同，那么代价为 0，可以从对角线上看出来。图 4-5 显示了初始状态，第一行和第一列中数值是加 1 增长的，表示对字符串“_”（空字符串）不断地移动或插入字符。

在动态规划算法的前向阶段，我们从左上角开始填充数组，方法是考虑到方案的当前点为止成功的部分匹配。也就是说，第 x 行和第 y 列交点（数组中下标为(x, y)的元素）的值是关于 x - 1 行 y 列、x - 1 行 y - 1 列或 x 行 y - 1 列三个值之一的一个（最小对齐问题，最小代价）函数。这三个数组位置保存着方案中到目前为止的对齐信息。如果位置(x, y)的字符匹配，那么将位置(x - 1, y - 1)处的值加 0 放到位置(x, y)；如果不匹配，那么加 2（因为移动和插入）。我们通过移动短的字符串（加到 y 列中上一个值上）或插入一个字符（加到 x 行中上一个值上）来加 1。继续这个过程就可以填充完整个数组，如图 4-6。可以观察得出，最小匹配通常出现在靠近数组从左上角到右下角的“对角线”的位置。

一旦数组填充完，我们就开始算法的后向阶段。这一阶段生成特定的解决方案。也就是说，从最佳可能对齐开始，从数组回头来选择一个特定对齐。我们从最大行列处的值开始这个过程，在我们的例子中是第7行12列的6。从该处开始，我们向后穿过数组，每一步选择一个（前向阶段），生成当前状态的直接前驱，或者是左上对角，或者是上面，或者是左面。只要出现数值减小，我们就选择斜线上的前驱，因为这是匹配前进过来的地方，例如，匹配开始附近的6和5；否则我们使用上一行或上一列的值。图4-7中的回溯过程生成了前面例子的一个最佳字符串对齐（几种可能对齐之一）。



图4-5 数组的初始阶段和第一步（使用动态程序设计来进行字符对齐）



图4-6 完成后的数组，反映字符串的最大对齐信息

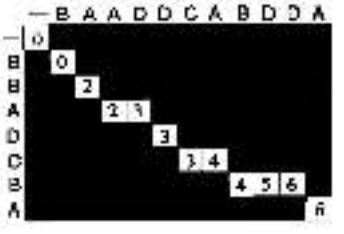


图4-7 完成后的回溯部分给出了一个（多种可能的）字符串对齐

在动态规划的第二个例子中，我们考虑两个串间的最小编辑差别问题。例如，如果我们要建立一个拼写检查器，希望能确定拼写正确的单词集合中那些单词与某个未知字符串最近似。类似的方法可以用来确定已知单词（表现为语音串）中的哪些与某个特定的语音串最紧密匹配。下面的例子一一为两个字符串建立编辑距离——改编自 Jurafsky 和 Martin（2009）。

假定你输入了一个无法识别的字符串，拼写检查器的任务是产生一个有序列表，列出在字典中最相似的单词，即你本来打算输入的单词。那么问题是，两个字符串之间的距离如何度量，即你刚刚输入的字符串和字典中的字符串。对于你的字符串，我们希望能提供一个字典中可能正确单词的有序列表。对于字典里的每个单词，我们希望有一个能表示这个单词与你输入的字符串的差别的数值度量。

两个串间的最小编辑差别可以定义为将第一个（源）串变为第二个（目标）串所必需的字符插入、删除和替换的数量。最小编辑差别有时也称为 Levenshtein 距离（Jurafsky and Martin 2009）。现在，我们实现一个动态规划搜索来确定两个字符串之间的最小编辑差别。令 intention 作为源串，execution 作为目标串。将第一个串转化为第二个串的编辑代价为：字符插入或删除为1，字符替换为2（一个删除加上一个插入）。我们希望确定这两个字符串之间的最小代价差别。

记录子目标的数组每一维的尺寸还是比每个字符串的长度大1，这里是 10×10。初始状态如图4-8所示（从空串开始，必须要进行一系列的插入才能使得两个字符串相同，数组中（2，2）处为2，是因为将 i 变为 e 需要一个替换（或者说一个删除加插入）。

图4-9给出了将动态规划算法应用到 intention 到 execution 的变换后的完成结果。数组中每个位置 (x, y) 处的值就是到该点为止的最小编辑代价再加上一个插入、删除或替换的（最小）代价。这样，(x, y) 处的代价就是 (x − 1, y) 处的最小编辑代价加上一个插入的代价，或者是 (x − 1, y − 1) 处的代价加上一个替换的代价，或者是 (x, y − 1) 处的代价加上一个删除的代价。这个算法的伪代码是一个函数，以两个字符串（一个源和一个目标）及其长度为输入，返回最小编辑差别代价：

		e	x	e	c	u	t	i	o	n
	0	1	2	3	4	5	6	7	8	9
i	1	2								
n	2									
t	3									
e	4									
n	5									
t	6									
i	7									
o	8									
n	9									

图 4-8　intention 和 execution 间最小编辑差别矩阵的初始状态
（改编自 Jurafsky 和 Martin 2000）

		e	x	e	c	u	t	i	o	n
	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	8	9	10
n	2	3	4	5	6	7	8	9	10	11
t	3	4	5	6	7	8	9	10	11	12
e	4	5	6	5	6	7	8	9	10	11
n	5	6	7	6	7	8	9	10	11	12
t	6	7	8	7	8	9	8	9	10	11
i	7	8	9	8	9	10	9	8	9	10
o	8	9	10	9	10	11	10	9	8	9
n	9	10	11	10	11	12	11	10	9	8

图 4-9　完成后的 intention 和 execution 间最小编辑差别矩阵
（改编自 Jurafsky 和 Martin 2000）

```
function dynamic (source, sl, target, tl) return cost (i, j);

create array cost(sl + 1, tl + 1)
cost (0,0) := 0                                                    % initialize
for i := 1 to sl + 1 do
    for j := 1 to tl + 1 do
        cost (i, j) := min [ cost (i − 1, j) + insertion cost target_j                % add 1
                             cost (i − 1, j − 1) + replace cost source_i , target_j   % add 2
                             cost(i, j − 1) + delete cost source_ i ]                 % add 1
```

用图4-9中的结果（加粗表示），下面的编辑将把 intention 变为 execution，全部编辑代价为8：

intention	
ntention	delete i, cost 1
etention	replace n with e, cost 2
exention	replace t with x, cost 2
exenution	insert u, cust 1
execution	replace n with c, cost 2

在拼写检查中，只需要提供一个替换无法识别单词的基于代价的有序单词列表即可，不需要动态规划算法的回溯过程。一旦为相关字符串集合中的每个单词计算出最小编辑度量，就能够提供一组有先后顺序的选择，用户可以从中选择一个正确的字符串。

之所以使用动态规划是考虑到计算中时间/空间的代价。如我们的两个例子中所见，动态规划的代价为 n^2，其中 n 是最长字符串的长度。最坏的情况下是 n^3，即需要考虑其他相关子问题（其他行/列的值）来确定当前状态。比较两个字符串的穷举搜索是指数级的，代价在 2^n 和 3^n 之间。

有许多明显的启发可以用来剪除动态规划中的搜索。首先，有用的答案通常会位于数组中左上到右下的对角线周围；这样可以不用计算数组的两极。其次，在数组构造的过程中可以剪枝搜索，例如，对编辑距离可以传入一个特定阈值，剪掉该解路径甚至放弃整个问题，意思就是，源串与目标串的距离太远了，以至于没有用了。在 5.3 节我们还将介绍一种随机方法，来解决模式比较问题。

4.2　最佳优先搜索算法

4.2.1　实现最佳优先搜索

尽管形式如同回溯、爬山策略和动态规划的算法只有其局限性，但是如果评估函数在足够

广博的知识信息量可以避免搜索空间中的局部最大值、死端和有关异常的话，那么它们还是非常有效的策略。然而，通常使用启发式搜索需要一种更灵活的算法：最佳优先搜索（best-first search）提供了这种灵活性，这一搜索使用了一个优先级队列，这使得从诸如陷入局部最优等情况中恢复成为可能。

和第3章中介绍的深度优先搜索和宽度优先搜索一样，最佳优先搜索也使用列表来维护状态：用open列表来记录搜索的当前搜索带；用closed列表记录已经访问过的状态。在这种算法中新加的一步是对open中的状态进行排序，排序的依据是对状态与目标“接近程度”的某种启发性估计。因此，每一轮循环都是考虑open列表中最“有希望的”状态。最佳优先搜索算法的伪代码如下：

```
function best_first_search;

begin
  open := [Start];                                          % initialize
  closed := [ ];
  while open ≠ [ ] do                                       % states remain
    begin
      remove the leftmost state from open, call it X;
      if X = goal then return the path from Start to X
      else begin
            generate children of X;
            for each child of X do
            case
                the child is not on open or closed:
                  begin
                      assign the child a heuristic value;
                      add the child to open
                  end;
                the child is already on open:
                  if the child was reached by a shorter path
                  then give the state on open the shorter path
                the child is already on closed:
                  if the child was reached by a shorter path then
                    begin
                      remove the state from closed;
                      add the child to open
                    end;
            end;                                            % case
            put X on closed;
            re-order states on open by heuristic merit (best leftmost)
          end;
return FAIL                                                 % open is empty
end.
```

在每一次迭代中，best-first-search删除open列表中的第一个元素。如果这个元素满足了目标条件，那么算法便返回产生这个目标的解路径。注意：每一个状态都保存了祖先信息，这样做有两个目的，一是可以判断以前是否曾以更短的路径到达过这个状态，二是算法利用这些信息来返回最终的解路径（见3.2.3节）。

如果open列表中的第一个元素不是目标，那么算法应用所有匹配的产生规则或操作符来产生这个元素的后继。如果孩子状态不在open或closed中，best-first-search评估open中状态的启发性，并根据这些状态的启发值对它们进行排序。这样就把“最佳”状态排在了open列表的最前

而。注意：因为这些估计实质上是启发式的，所以要分析的下一个“最佳”状态可能来自状态空间的任一个层次。当把 open 维护为一个有序列表时，经常把它称为优先级队列（priority queue）。

如果孩子状态已经在 open 或 closed 中，那么算法检查对应的部分解路径以确保这个状态记录的是较短的路径。重复状态是不会被保留的。当再次发现结点状态时，通过更新 open 和 closed 中结点的祖先历史信息，算法更有可能找到到达目标的更短路径（在所考虑的状态内）。

图 4-10 显示了一个假想的状态空间，其中的一些状态附加了启发式评估值。这些被附加了评估值的状态是在 best-first-search 中确实产生的状态。由启发式搜索算法所展开的状态是用粗体表示的；注意算法并没有搜索整个空间，最佳优先搜索的目标是考虑尽可能少的状态；使用的启发越广博（见 4.2.3 节），为了找到目标而处理的状态数就越少。

下面给出了对这幅图执行 best-first-search 的跟踪结果。假定 P 是图 4-10 所示的图的目标状态。因为 P 是目标，所以通往 P 路径上的状态的启发值往往较低。启发难免会存在错误：状态 O 比目标本身的启发值更低，因此它被先分析。与爬山策略不同，爬山策略不维护用来选取“下一个”状态的优先队列，所以无法从错误中恢复，而这种算法可以从这个错误中恢复并找到正确解。

1）open = [A5]；closed = [ ]

2）评估 A5；open = [B4, C4, D6]；closed = [A5]

3）评估 B4；open = [C4, E5, F5, D6]；closed = [B4, A5]

4）评估 C4；open = [H3, G4, E5, F5, D6]；closed = [C4, B4, A5]

5）评估 H3；open = [O2, P3, G4, E5, F5, D6]；closed = [H3, C4, B4, A5]

6）评估 O2；open = [P3, G4, E5, F5, D6]；closed = [O2, H3, C4, B4, A5]

7）评估 P3；解找到了！

图 4-11 显示了 while 循环 5 次后的状态空间。图中表示出了 open 和 closed 中所包含的状态。open 中记录了搜索的当前前线，closed 中记录了已经考虑过的状态。注意：搜索前线是非常不均衡的，这反映了最佳优先搜索的机会性。

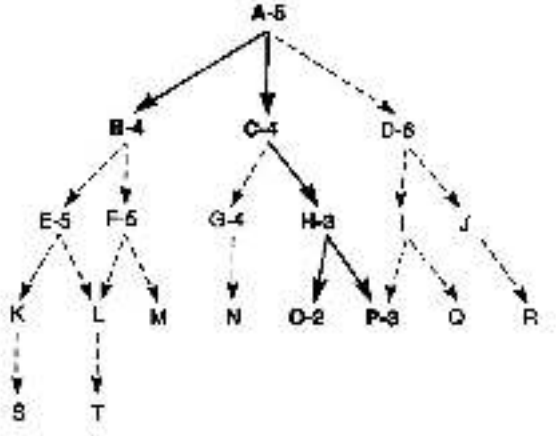


图 4-10　对一个假想状态空间的启发式搜索

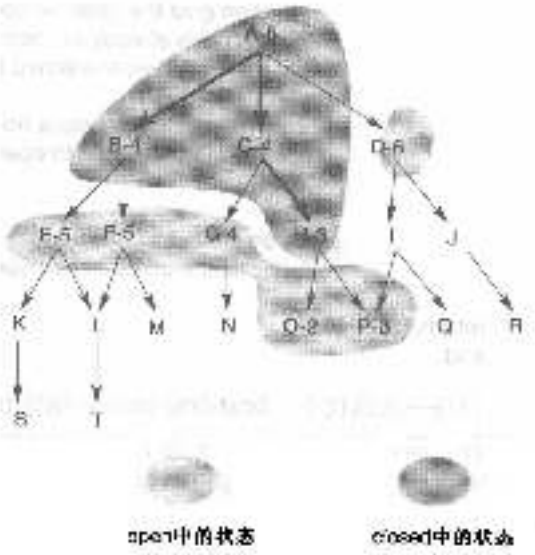


图 4-11　对一个假想状态空间的启发式搜索

注：图中突出表示了 open 和 closed 列表中的状态

最佳优先搜索算法总是选择 open 中最有希望的状态做进一步的扩展。然而，由于它正在使用的启发可能被证明是错误的，所以它并不放弃所有其他状态而是把它们维护在 open 中。一旦发现启发将搜索引导到一条证明不正确的路径，那么算法会从 open 中取出一些以前产生的“次最优”状态，从而把搜索的焦点转移到空间的另一部分。在图 4-10 的例子中，在发现了状态 B 的孩子的启发值都很差时，搜索的焦点转移到状态 C。状态 B 的孩子被保留在 open 中以防止算法以后需要返回到它们。在 best-first-search 中，open 列表允许从无法产生目标的路径中回溯，这一点与第 3 章中的算法相同。

4.2.2 实现启发评估函数

现在我们评估几种不同的求解 8 格拼图游戏问题的启发。图 4-12 显示了 8 格拼图游戏的起始状态和目标状态，以及搜索中产生的前三个状态。

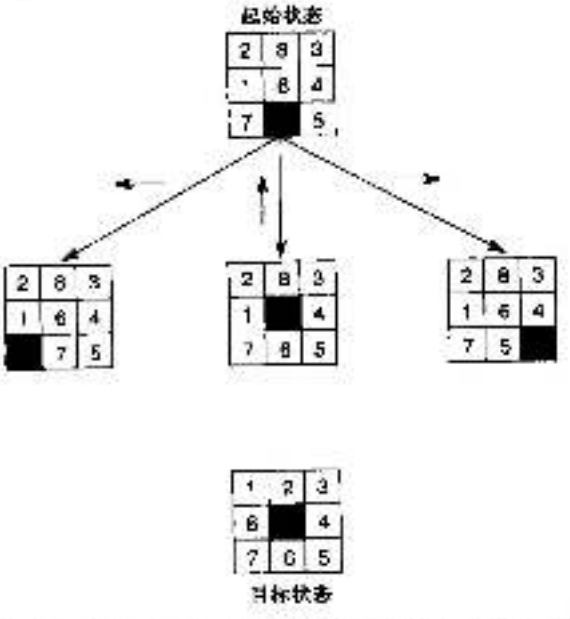


图 4-12 8 格拼图游戏实例的起始状态、首次移动集合和目标状态

最简单的启发是数出每个状态与目标状态相比错位的牌数。虽然直觉这种方法是令人感兴趣的，因为它似乎意味着如果其他条件相同，那么错位牌数最少的状态可能最接近预期目标，所以它是接下来要分析的最佳状态。

然而，这个启发没有使用从棋盘格局中可以得到的所有信息，因为它没有把牌必须要移动的距离纳入考虑。一个“更好一点”的启发是对错位的牌必须要移动的距离求和，为了到达目标位置，每张牌必须要移动的每个方格算为 1 个距离单位。

这两种启发都存在一种不足，就是没有认识到倒倒牌的难度。也就是说，如果两张牌是彼此相邻的，而且目标是要求互相颠倒它们的位置，那么要把它们放到适当的位置需要远不止两次的移动，因为这些牌必须彼此间“绕来绕去”（见图 4-13）。

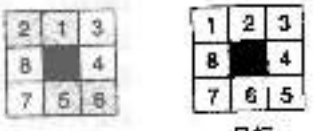


图 4-13 8 格拼图游戏的目标状态和一个存在颠倒的状态（1 和 2，5 和 6）

一种把这个问题纳入考虑的启发对每个直接颠倒（两张相邻的牌要满足目标顺序必须交换位置）乘以

一个小的数（比如2）。图4-14显示了对图4-12中的三个孩子状态应用这三种启发的结果。

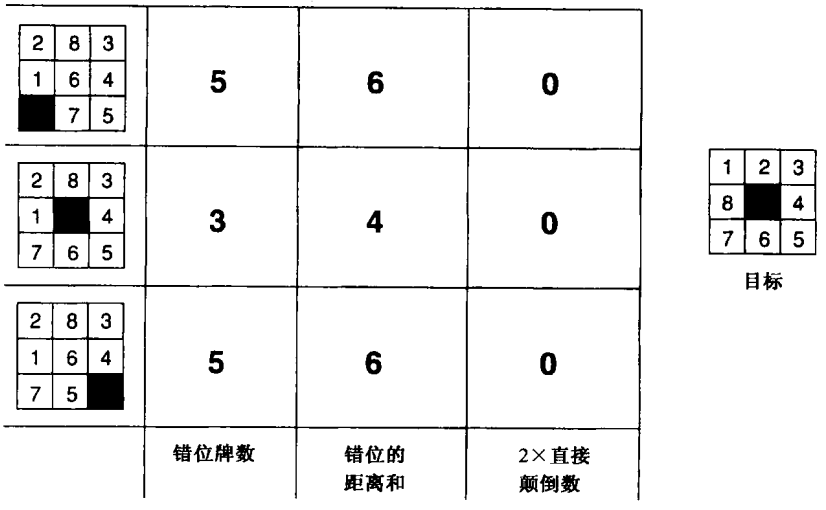


图4-14　应用于8格拼图游戏的三种启发

从图4-14中对评估函数的归纳来看，“距离和”启发确实看起来比简单的错位计数估计得更加精确。还有，牌颠倒启发无法区分图4-12中的三个状态，对每个状态的评估都是0。尽管根据直觉它是一种吸引人的启发，但是它失败了，因为这三种状态中没有一种存在直接颠倒。第四种启发是把错位牌的距离和与直接颠倒数的二倍相加，这样就克服了单纯颠倒启发的不足。

这个例子说明了设计好启发的难度。我们的目的是使用从单一状态描述符中可以得到的有限信息来做出智能的抉择。上面提出的每一种启发都因忽视了一些关键信息而有待改善。要设计出好的启发离不开丰富的经验；判断和直觉是有帮助的，但衡量启发好坏的最终尺度是问题实例的实际性能。

如果两个状态具有相同的或接近相同的启发值，那么通常优先分析最靠近图中根结点的状态。这个状态更有可能位于通往目标的最短路径上。可以通过维护每个状态的深度计数来衡量从起始状态到其后继的距离。对起始状态来说，这个计数是0，搜索每深入一层便递增1。如果希望搜索优先考虑图中的较浅层状态，那么可以把这个值加入到每个状态的启发值中。

这构成了一个评估函数f，它是两个分量的和：

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

其中：$g(n)$是从任意状态n到起始状态的实际路径长度，$h(n)$是对状态n到目标的距离的启发性估计。

例如，在8格拼图游戏中，我们可以用$h(n)$表示错位的牌数。当把这个评估函数应用到图4-12中的三个孩子状态时，它们的f值分别是6、4和6，如图4-15所示。

图4-16给出了使用上面定义的启发f对8格拼图游戏图进行最佳优先搜索的全过程。图中用字母标出了每个状态以及它的启发权$f(n) = g(n) + h(n)$。每个状态上面的数字指出了它被从open列表中取出的顺序。有些状态(h、g、b、d、n、k和i)没有这样编号，这是因为算法终止时它们还在open列表中。

下面给出了产生这个图的各个阶段open和closed列表中的状态：

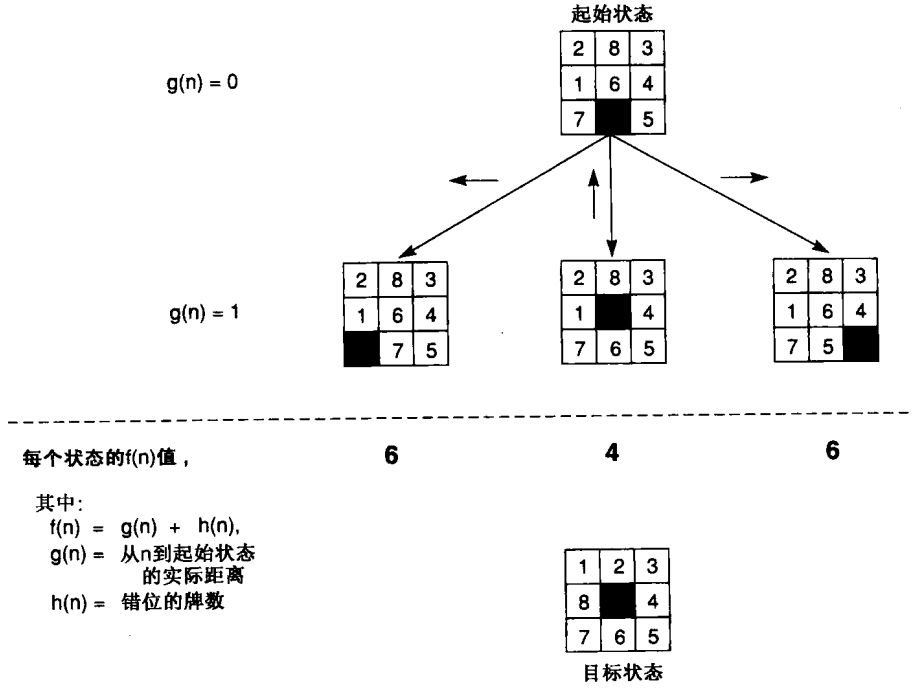


图4-15 把启发f应用到8格拼图游戏中的三个状态

1) open = [a4];
   closed = [ ]
2) open = [c4, b6, d6];
   closed = [a4]
3) open = [e5, f5, b6, d6, g6];
   closed = [a4, c4]
4) open = [f5, h6, b6, d6, g6, i7];
   closed = [a4, c4, e5]
5) open = [j5, h6, b6, d6, g6, k7, i7];
   closed = [a4, c4, e5, f5]
6) open = [l5, h6, b6, d6, g6, k7, i7];
   closed = [a4, c4, e5, f5, j5]
7) open = [m5, h6, b6, d6, g6, n7, k7, i7];
   closed = [a4, c4, e5, f5, j5, l5]
8) m = 目标, 成功了!

在第3步中，e和f的启发值都为5。先分析了状态e，产生孩子h和i。尽管h（e的孩子）和f的错位牌数相同，但是它在空间中的深度较大。深度尺度g(n)导致算法在第4步中选取f作为评估对象。这样，算法返回到较浅的状态继续搜索目标。这一阶段搜索的状态空间图显示在图4-17中，图中突出显示了open和closed列表中的状态。注意观察最佳优先搜索算法的机会性。

实际上，评估函数的g(n)分量使这种搜索带有更多的宽度优先性。这防止了搜索被错误的评估所误导：如果启发对一条无法到达目标的路径上的状态连续给出“好的”评估，那么g值

会上升来控制 h，从而强迫搜索返回到一条较短的解路径。这保证了算法不会永久迷失，沿无限的错误路径一直下去。4.3 节分析了在什么条件下利用这一评估函数的最佳优先搜索算法可以保证产生到达目标的最短路径。

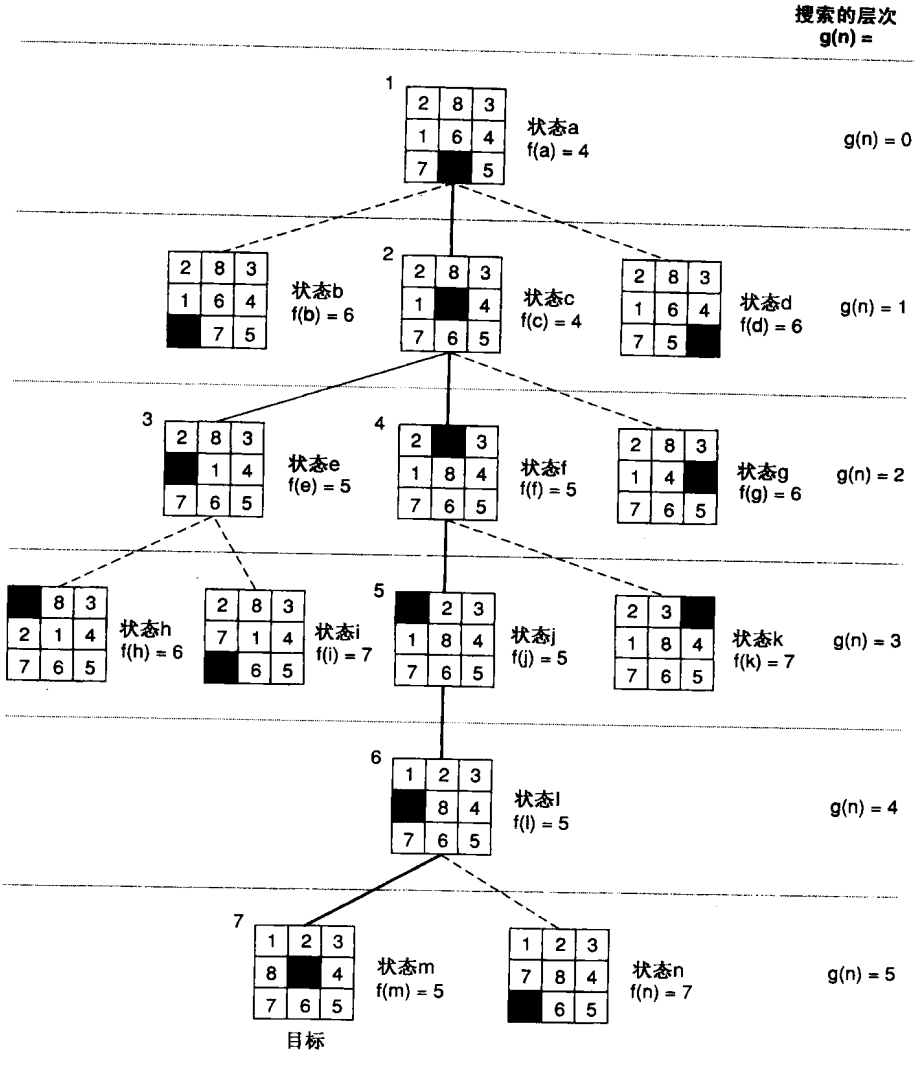


图 4-16　对 8 格拼图游戏进行启发式搜索而产生的状态空间

归纳起来，最佳优先搜索算法就是：

1）操作当前状态以产生新的孩子。

2）检查每个新的状态，看其是否已经（在 open 或 closed 中）出现过，以防止循环。

3）给出每个状态 n 的 f 值，这个值等于该状态在搜索空间中的深度 g(n) 和它与目标距离的启发性估计 h(n) 的和。h 值把搜索引向更有希望的状态，而 g 值防止搜索沿无用的路径无限跑下去。

4）open 中的状态是按它们的 f 值排序的。在分析了所有状态或发现目标之前，所有的状态都被保存在 open 中，这样做使算法可以从死端（dead end）恢复。

5）从实现的角度来看，可以通过改善维护 open 和 closed 列表的方法来提高算法的效率，

比如可以把它们维护为堆（heap）或左翘子树（leftist tree）。

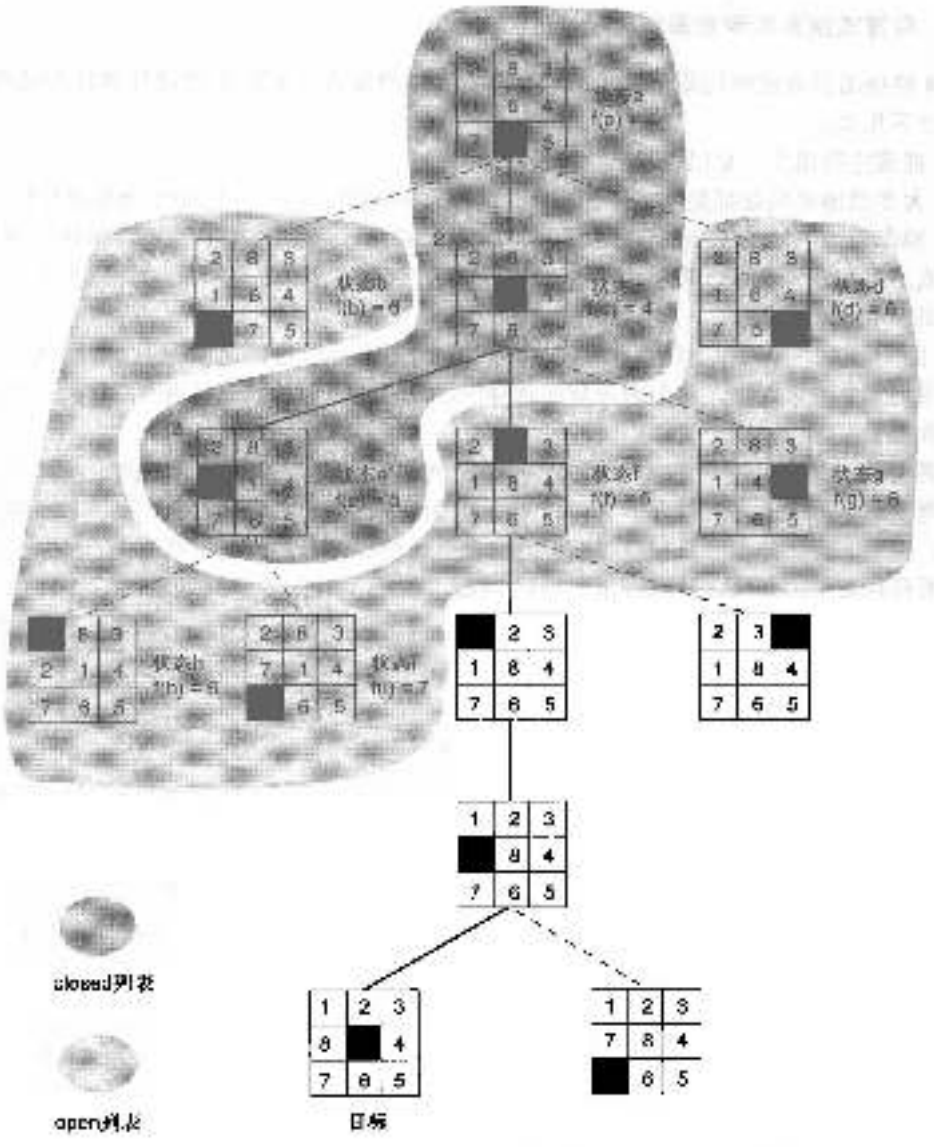


图4-17 对8格拼图游戏进行启发式搜索第三次迭代后的情况

最佳优先搜索是一种启发式搜索任何状态空间的通用算法（这一点与前面给出的宽度优先算法和深度优先算法相同）。它既适用于数据驱动搜索，也适用于目标驱动搜索，而且支持不同的启发评估函数。它也是分析启发式搜索特性的基础（见4.3节）。由于它具有很好的通用性，因此可以把它和很多种不同的启发方法一起使用，从对状态的“优秀程度”的客观估计，到基于概率（状态产生目标的概率）的复杂度量。贝叶斯统计定理（见第5章和第9章）是概率方法的一个重要实例。

实现启发的另一种有趣途径是专家系统中所使用的用置信度来衡量规则结果。人类专家采用启发时，我们通常可以估计出他们所得结论的可信度。专家系统采用置信度（confidence measure）来选取具有最大成功可能性的结论。置信度特别低的状态可能被完全排除。下一节将分析

使用这一方法的启发式搜索。

4.2.3　启发式搜索和专家系统

像8格拼图游戏这样的简单博弈问题是用于探索启发式搜索算法的设计和行为的理想工具，原因有以下几点：

1）搜索空间很大，足以需要启发式剪枝。

2）大多数博弈问题都足够复杂，蕴涵了丰富多样的启发评估方法用于分析和比较。

3）博弈通常不涉及复杂的表示问题。状态空间的一个结点就是一种棋盘描述，而且通常用一种直截了当的方式就可以表示出来。这使研究者可以把注意力集中在启发行为上，而不是集中在知识表示问题上。

4）因为状态空间的每个结点都有共同的表示（也就是棋盘描述），所以可以把一种简单的启发应用到整个搜索空间。这与财务顾问这样的系统形成了对比，对后者来说，每个结点表示了具有各自相异描述的不同子目标。

在更现实一些的问题中，启发式搜索的分析和实现会复杂很多，这是因为可能需要多个启发来处理问题空间中的不同情况。然而，从简单博弈中得到的经验可以推广到其他问题，比如专家系统、规划、智能控制和机器学习。但这些领域的情况与8格拼图游戏不同，不能把一个启发应用到所有状态上。相反，针对特定条件的问题求解启发是按各个问题求解操作符的语法和内容编码的。每个求解步骤与它自己的启发相配合，决定何时该应用它；模式匹配程序把适当的操作（启发）与空间中的有关状态匹配起来。

例4.2.1　再谈财务顾问问题

使用启发度量来引导搜索是AI中的一种常见方法。下面再次考虑第2、3章的财务顾问问题。在这个问题中，知识库是由一系列逻辑蕴涵（其结论要么是真要么是假）组成的。例如，下面的规则指出一个有充足存款和收入的投资个体应该投资股票：

$$\text{savings\_account(adequate)} \wedge \text{income(adequate)} \Rightarrow \text{investment(stocks)}$$

实际中，这样的投资个体有可能更喜欢更加安全的组合策略或者甚至倾向于把所有投资款都存入银行。因此，这个规则是一个启发，问题求解器应该尽可能考虑这种不确定性。我们可以考虑其他的附加因素，比如这个投资者的年龄和投资者的职务晋升和风险等长期因素。这样可以使规则具有更高的信息度，从而可以更好地区别不同情况。不过，这并没有改变理财建议的基本启发特征。

专家系统处理这一问题的方法是给每个规则的结论附加一个数字类型的权（称为*置信度*或*确信度*（certainty factor））。这个指标衡量了这些结论所具有的可信程度。

这样，就可以给每个规则的结论赋予一个置信度，也就是一个在-1和1之间的实数，1对应于一定（真）；-1对应于一个明确的假值。介于这两者间的值反映了对结论的不同置信度。例如，可以给前面的规则赋予一个置信度，比如说0.8，那么这反映了这个规则不正确的可能性很小。类似地，可以为得出的其他结论赋予不同的信心权值：

$$\text{savings\_account(adequate)} \wedge \text{income(adequate)} \Rightarrow \text{investment(stocks)}\ \text{置信度} = 0.8$$
$$\text{savings\_account(adequate)} \wedge \text{income(adequate)} \Rightarrow \text{investment(combination)}\ \text{置信度} = 0.5$$
$$\text{savings\_account(adequate)} \wedge \text{income(adequate)} \Rightarrow \text{investment(savings)}\ \text{置信度} = 0.1$$

这些规则反映了尽管是对同样有充足存款和收入的投资个体，通常也会有几种建议，最强烈推荐的做法是投资股票，其次是如果有可能应该采取组合策略并找机会继续投资存款。启发

式搜索算法可以以多种方式使用这些确信度。例如，所有可应用规则的结果可以与这些结果相关联的置信度一并产生。这种对所有可能性的穷举式搜索可能适合于类似医疗这样的领域。另一种做法是，如果对其他解不感兴趣，那么程序可以仅返回具有最大置信度的结果。这样可以使程序忽略其他的规则，彻底剪除其对应的搜索空间。更稳妥的剪枝策略是设定一个特定值（比如0.2），如果一个规则结论的置信度小于这个值，那么就忽略这个规则。

在使用置信度来衡量规则的结论时必须注意很多重要的问题。例如，确定一个“数字类型的置信度”的实质目的是什么？如果一个规则的结论要用做另一个规则的前提，那么该如何处理置信度？当从多个规则得出同一结论时该如何合并置信度？如何给规则赋予合适的置信度？第8章将更详细地讨论这些问题。

4.3 可采纳性、单调性和信息度

我们可以使用很多度量来评估一个启发的好坏。例如，我们可能希望得到解而且需要算法找到到达目标的最短路径。当一个应用需要为多余的解步骤付出额外开销时（比如为机器人规划一条穿越危险环境的路径），最短路径解是很重要的。如果只要存在到达一个目标的最短路径，启发就可以找到这条最短路径，那么就可以说这些启发是*可采纳的*。在其他一些应用中，解路径是否最短可能没有总体的问题求解效率重要（见图4-28）。

我们可能想知道是否还有更好的启发。从什么意义上来说一个启发好于另一个启发？这就是启发的*信息度*（informedness）。

当启发式搜索发现了一个状态时，是否有什么可以保证继续搜索也不会以一个更低廉的开销（从起始状态经过一条更短的路径）到达同样状态？这就是*单调性*（monotonicity）。对这些问题的回答以及与启发的有效性相关的其他问题就是本节要讨论的内容。

4.3.1 可采纳性度量

如果只要存在到达目标解的最短路径，搜索算法就可找到这样的路径，那么这个算法就是*可采纳的*。宽度优先搜索是一种可采纳的搜索策略。因为它总是在考虑图的第 $n+1$ 层的任何状态之前观察第 n 层的所有状态。不幸的是，宽度优先搜索对实践应用来说效率太低。

下面，我们利用上一节介绍的评估函数 $f(n) = g(n) + h(n)$ 来刻画一类具有可采纳性的启发式搜索策略。如果 n 是状态空间图中的一个结点，那么 $g(n)$ 度量了这个已发现状态在路径上的深度；而 $h(n)$ 是对 n 到目标的距离的启发式估计。从这个意义上来讲，$f(n)$ 估计了从起始状态经过 n 到达目标状态的总代价。为了确定可采纳启发的特征，我们定义一个评估函数 f^*：

$$f^*(n) = g^*(n) + h^*(n)$$

其中：$g^*(n)$ 是从起始结点到结点 n 的*最短路径*的代价，h^* 是从 n 到目标的最短路径的*实际代价*。这样 $f^*(n)$ 就是从起始结点经过结点 n 到达目标的最优路径的实际代价。

可以证明，当把这个评估函数应用到 best-first-search 时，产生的搜索策略是可采纳的。尽管像 f^* 这样的*神谕*（oracles）在现实问题中是不存在的，但是我们希望评估函数 f 是对 f^* 的一种非常接近的估计。在A算法中，到状态 n 的当前路径的代价 $g(n)$ 是对 g^* 的一个合理估计，但它们可能不相等：$g(n) \geqslant g^*(n)$。只有当已经发现了到状态 n 的最优路径时这两者才是相等的。

类似地，我们用对到目标状态的最小代价的启发式估计 $h(n)$ 代替 $h^*(n)$。尽管我们通常不可能求出 h^*，但很多时候有可能判断出启发式估计 $h(n)$ 是否是以 $h^*(n)$ 为上限，即 $h(n)$ 总是小于等于最短路径的实际代价。如果A算法使用的评估函数 f 满足 $h(n) \leqslant h^*(n)$，那么就把它称为 A^* *算法*。

定义（A 算法、可采纳性、A* 算法）　考虑评估函数 f(n) = g(n) + h(n)，其中：

n 是搜索中遇到的任意状态。

g(n) 是从起始状态到 n 的代价。

h(n) 是对 n 到目标状态代价的启发式估计。

如果在 4.1 节中的 best-first-search 算法中使用了这个评估函数，那么就把这种算法称为 **A 算法**。

如果在任何图中只要存在从起始结点到达目标状态的最优路径，搜索算法就可以找到这样的路径，那么这个算法就是**可采纳的**。

如果在 A 算法中使用的评估函数满足 h(n) 小于等于从 n 到目标的最短路径代价，那么就把这种算法称为 **A* 算法**（A* 读为“A 星”）。

A* 算法的一个特性是：所有的 A* 算法都是可采纳的。

A* 算法的可采纳性是一个定理。本章末尾的练习中提示了如何推导出它的证明过程（也可以参见 Nilsson 1980，p76 ~ p78）。这个定理指出任何 A* 算法（即使用启发 h(n)，满足对于所有的 n 都有 $h(n) \leqslant h^*(n)$ 的算法）都保证可以找到从 n 到目标的最短路径，只要这样的路径存在。

注意：可以把宽度优先搜索算法看作是 f(n) = g(n) + 0 的 A* 算法。考虑状态的决策完全依赖于从起始状态到状态 n 的距离。在 4.3.3 节中我们会说明 A* 算法所考虑的结点集是宽度优先搜索所考虑的结点集的一个子集。

前面给出的对 8 格拼图游戏的几种启发都是可用于 A* 算法的例子。尽管我们可能无法求出 8 格拼图游戏的 $h^*(n)$ 值，但是只要启发是以到达目标的最短路径实际代价为上限，我们就可以确定它是满足 A* 算法要求的。

举例来说，“对不在目标位置的牌进行计数”这一启发肯定小于等于把它们移到目标位置所需的移动步数。那么这个启发是可采纳的，也就是可以保证找到最优解（也就是最短路径）。所有错位牌与它目标位置的直接距离之和也是小于或等于实际最短路径。将直接颠倒牌的个数乘一个小的数也是可采纳的启发。

可以把这种证明 8 格拼图游戏启发的可采纳性的方法应用到任何启发式搜索问题中。虽然到达目标的实际最短路径并不总是可以计算的，但是我们经常可以证明一个启发是以这个值为上限的。只要做到了这一点，产生的搜索就会发现到目标的最短路径（条件是这样的路径存在）。

4.3.2　单调性

A* 算法的定义并不需要 $g(n) = g^*(n)$。这意味着可采纳的启发初始时可能沿着并非最优的路径到达非目标状态，但它最终会找到通往目标路径上所有状态的最优路径。那么，一个很自然的问题是，是否存在“局部可采纳”的启发，即是否总是可以找到到达搜索中遇到的每个状态的最短路径。这个特性被称为单调性。

定义（单调性）　启发函数 h 单调的条件是：

1）对于所有的状态 n_i 和 n_j，其中 n_j 是 n_i 的后继，

$h(n_i) - h(n_j) \leqslant cost(n_i, n_j)$

其中：$cost(n_i, n_j)$ 是从状态 n_i 到 n_j 的实际代价（以移动次数为单位）。

2）目标状态的启发值为零，即 h(Goal) = 0。

描述单调特征的一种方式是，搜索空间的每一处都与所采用的启发具有局部一致性。一个状态的启发性尺度与其任何后继的启发性尺度间的差异是以从这个状态到其后继的实际代价为

上限的。这就是说，这个启发在搜索空间的每一处都是可采纳的，可以从每个状态的祖先沿最短的路径到达这个状态。

如果采用最佳优先搜索的图搜索算法使用了单调启发，那么就可以省略一个重要的步骤。因为这个启发第一次发现任何一个状态时所用的路径总是最短的，所以当第二次遇到某个状态时，就没有必要检查这条新的路径是否是更短的。它肯定不会更短。这样，当算法在空间中再次发现任何状态时，它就可以立刻丢弃这个状态，不必去更新保留在 open 和 closed 中的信息。

当使用一个单调启发时，随着搜索在空间中的移动，对每一个状态 n 的启发尺度被替换为产生到 n 的那段路径的实际代价。因为这个实际代价大于或等于每个实例中的启发，所以 f 不会下降；也就是说 f 是单调非递减的（所以叫单调性）。

一个简单的证明可以说明任何单调的启发是可采纳的。可以把空间中的任何路径看作是状态 s_1，$s_2, \cdots, s_g$ 的序列，其中 s_1 是起始状态，s_g 是目标状态。对于这条任意选择的路径中的移动序列：

根据单调属性，从 s_1 到 s_2 有 $h(s_1) - h(s_2) \leqslant \text{cost}(s_1, s_2)$

根据单调属性，从 s_2 到 s_3 有 $h(s_2) - h(s_3) \leqslant \text{cost}(s_2, s_3)$

根据单调属性，从 s_3 到 s_4 有 $h(s_3) - h(s_4) \leqslant \text{cost}(s_3, s_4)$

……

根据单调属性，从 s_{g-1} 到 s_g 有 $h(s_{g-1}) - h(s_g) \leqslant \text{cost}(s_{g-1}, s_g)$

汇总每一列，并应用单调性中的 $h(s_g) = 0$：

从 s_1 到 s_g 的路径有 $h(s_1) \leqslant \text{cost}(s_1, s_g)$

这意味着单调启发 h 满足 A* 算法的要求，即它是可采纳的。那么启发的可采纳性是否蕴涵了单调性呢？这个问题留作练习。

4.3.3 信息度更高的启发是更好的启发

这个小节的最后一个问题是如何比较两种启发发现最短路径的能力。当启发满足 A* 算法要求时会产生一个有趣的现象。

定义（信息度） 对于两个 A* 启发 h_1 和 h_2，如果对于搜索空间中的所有状态 n 都满足 $h_1(n) \leqslant h_2(n)$，那么就说 h_2 比 h_1 具有**更高的信息度**。

我们可以使用这个定义来比较为求解 8 格拼图游戏而提出的几种启发。正如前面所指出的，宽度优先搜索等价于使用启发 h_1（对于所有的状态 x，$h_1(x) = 0$）的 A* 算法。这显然小于 h*。我们也已经证明了 h_2（和目标状态相比错位的牌数）的下限是 h^*。在这个例子中，$h_1 \leqslant h_2 \leqslant h^*$。这可以得出“错位牌数”这个启发比宽度优先搜索的信息度更高。图 4-18 比较了这两种启发所搜索的空间。h_1 和 h_2 都找到了最优路径，但是 h_2 在搜索过程中估算的状态更少。

类似地，我们可能想知道计算错位牌离目标的直接距离之和这一启发是不是又比计算错位牌数具有更高的信息度，事实上是这样的。我们可以画出一系列搜索空间，每一个都小于前面的一个，并收敛到直接的最优解路径。

如果启发 h_2 比 h_1 的信息度更高，那么 h_2 分析的状态集合是 h_1 所展开状态的一个子集。这个结论可以用反证法来验证：找到某个状态，h_2 展开了，但是 h_1 没有展开；但因为 h_2 比 h_1 具有更高的信息度，即对于所有的 n，$h_1(n) \geqslant h_2(n)$，而且二者都是以 h^* 为上限，所以我们的假定是不成立的。

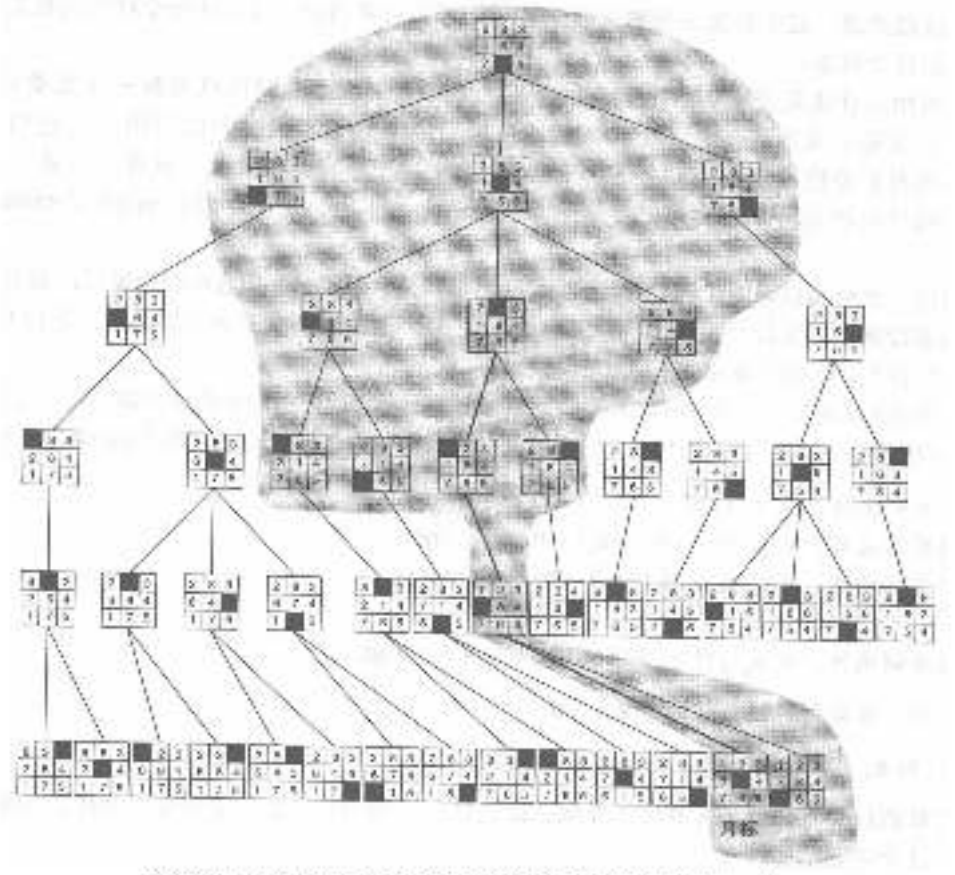


图 4-18　启发式搜索和宽度优先搜索所探索的状态空间比较
注：图中加有阴影的部分是启发式搜索所搜索的空间。它使用的启发函数 f(n) = g(n) + h(n)，其中 h(n) 是错位牌的数量。加粗的路径是最优解路径

通常，A* 算法的信息度越高，要得到最优解而需要展开的空间就越小。然而我们必须注意，采用高信息度启发所需的计算量可能会增大，以至于抵消了搜索状态数量降低所带来的优势。

计算机国际象棋程序为这种折中提供了一个有趣例证。一种观点是使用简单的启发，依赖计算机的速度深入搜索状态空间。这些程序经常使用特殊的硬件对状态进行评估从而增加搜索深度。另一种观点依赖更加密集复杂的启发来降低要搜索的状态数量。这样的启发包括：对各个棋子优势的计算、对棋盘格局的控制、可能的袭击策略、兵的升变、等等。这些启发本身的计算复杂度可能是指数级的（4.5 节讨论这个问题）。因为在博弈中，前 40 步的总时间是有限制的，所以在搜索和启发评估之间选择最佳平衡点是很重要的。但如何把盲目搜索和启发最佳地结合起来还是计算机象棋中依赖经验并有待继续研究的一个问题，参见 Gary Kasparov 和 Deep Blue 的对弈（Hsu 2002）。

4.4　在博弈中使用启发

那时有两种相对立的博弈概念，称为前向评价和对论。最早的棋手把博弈分成两种基本类型：形式博弈和心理博弈……

——赫尔曼·赫塞，"Magister Ludi"（一种玻璃珠游戏）

4.4.1　在可穷举搜索图上的极小极大过程

博弈一直是启发式算法所针对的一个重要领域。由于存在"对抗性"，两人对弈比简单的拼

图游戏更加复杂，因为对手的反应是根本无法预测的。这为探索启发提供了更广阔的机会，同时也使开发搜索算法的难度更大。

我们首先考虑状态空间足够小的适合穷举搜索的博弈；这里的问题就是对可能的移动和对手的可能反应所组成的空间进行系统搜索。然后我们再分析不能进行穷举搜索或不适合这样做的博弈。因为仅可以产生和搜索状态空间的一个部分，所以棋手必须使用启发来引导博弈过程沿一条通往胜利状态的路径进行。

我们先考虑余一棋（nim）的一种变体，它的状态空间是可以穷举搜索的。这种棋是这样玩的，两个对弈者之间的桌子上放着很多筹码；在每一步，棋手必须把一堆筹码分成不等数量的非空两堆将牌。因此，6张将牌可以被分割成5个筹码和1个筹码的两堆或者4个筹码和2个筹码的两堆，但不可以是3个筹码和3个筹码的两堆。最先无法继续分堆的棋手失败。对于正常数量的将牌，我们可以对这种棋的状态空间进行穷举搜索。图4-19画出了使用7张将牌玩这种游戏时的状态空间。

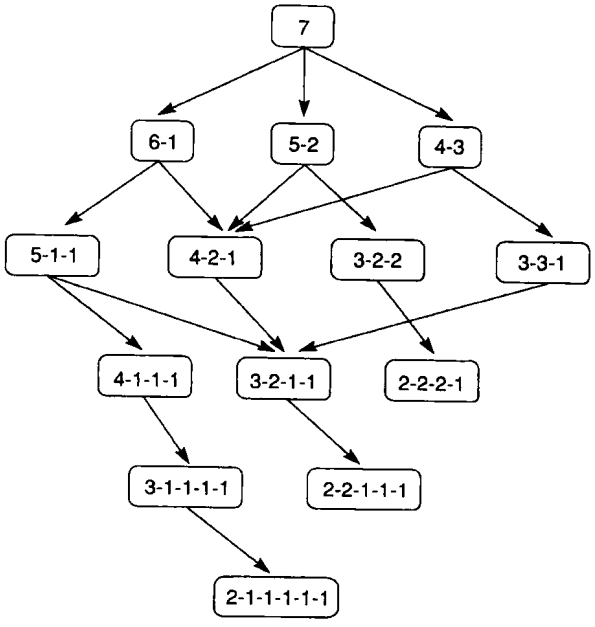


图4-19 一种余一棋变体的状态空间

注：每个状态把7个将牌分割成一或多堆

在状态空间可以穷举描述的博弈中，主要的难点是如何考虑对手的动作。处理这一问题的一种简单方式是假定对手具有相同的关于状态空间的知识，而且他也是应用这一知识以一致的方式争取赢得比赛。尽管这个假定具有局限性（4.4.2节中会讨论这个问题），但是它为预测对手的行为提供了一个合理的基础。对博弈空间的极小极大（minimax）搜索就是基于这一假定。

我们把博弈中的对手分别称为MIN和MAX。尽管这种称呼部分是由于历史原因而形成的，但主要是因为这样称呼的含义更直接：MAX代表努力争取胜利的棋手，也就是要最大化（MAXimize）它的优势。MIN是对手，它总是试图最小化（MINimize）MAX的得分。我们假定MIN使用同样的信息而且总是企图移动到对MAX最不利的状态。

在实现极小极大搜索时，我们把搜索空间的每一层根据博弈中这一步的移动方标上MIN或MAX。在图4-20所示的例子中，MIN被允许先移动。每个叶结点有一个1或0的值，代表是MAX胜利了还是MIN胜利了。极小极大搜索根据下面的规则沿连续的父结点向上传播这些值：

如果父状态是一个MAX结点，那么把孩子中的最大值赋给它。

如果父状态是一个MIN结点，那么把孩子中的最小值赋给它。

于是赋给每个状态的值表示了这个棋手可以期望达到的最佳状态值（假定对手的走法与极小极大算法所预测的走法一样）。然后，算法便使用这样推导出的值来选择移动方法。图4-20显示了对余一棋状态空间图应用极小极大算法的结果。

叶结点的值使用极小极大规则向上传播。因为MIN第一步的所有可能移动所产生的结点的值都是1，所以不论MIN第一步如何移动，第二个棋手MAX总可以使博弈胜利。只有MAX下得很愚蠢时MIN才会赢。图4-20以粗的箭头表示出了MAX的赢棋路线，MIN可以选取第一步可能

移动的任何一种方法。

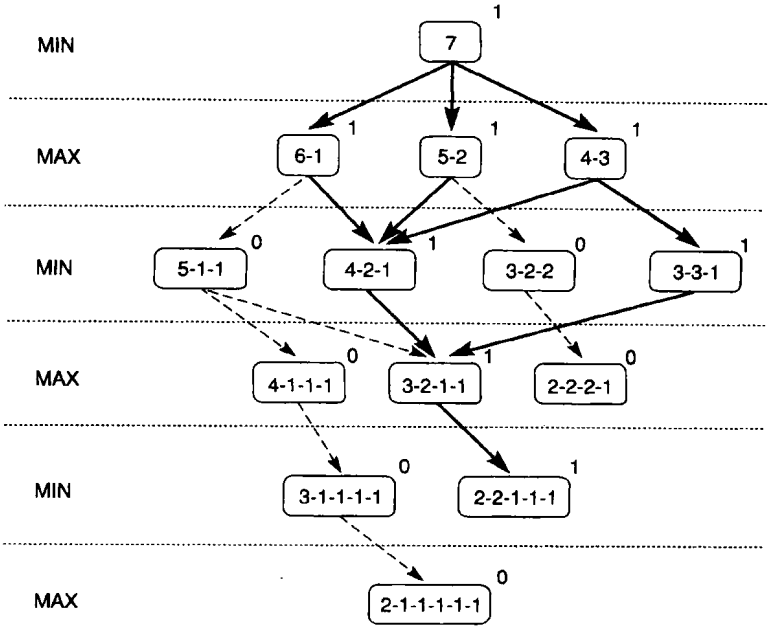


图 4-20　对余一棋应用穷举式的极小极大算法

注：粗线表示 MAX 的赢棋路线。在极小极大搜索中，每个结点都被标上它自己的导出值（0 或 1）

尽管有些博弈的状态空间是可以穷举搜索的，但是大多数实际情况是不允许进行穷举搜索。下一节将讨论固定深度的搜索。

4.4.2　固定层深的极小极大过程

在对更复杂的博弈应用极小极大算法时，很多时候都不可能把状态空间一直展开到叶结点。而是搜索状态空间中预先定义的一定层数，具体数字是由可用的时间和内存资源来决定的。我们把这种策略称为 n－层预判（n-ply look-ahead），其中 n 是要探索的层数。由于这个子图的叶结点不是博弈的最终状态，所以按照胜利或失败来为这些状态赋值是不可能的。相反，需要根据某个启发评估函数给每个结点赋一个值。这个向上传播的值并不表示是否可以胜利（就像前一个例子那样），它只代表从当前起始结点通过 n 次移动可以达到的最佳状态的启发值。预判把启发应用到状态空间的一个更大区域，所以它提高了启发的作用。极小极大过程把这些分离的评估值合并成祖先状态的一个单一值。

在博弈对抗中，每个棋手都企图战胜对方，所以很多博弈启发直接衡量一个棋手相对于另一个棋手的优势。在西洋跳棋或国际象棋中，棋子的优势是很重要的，所以一种简单的启发是计算属于 MAX 和 MIN 的子数的差异，并尽可能最大化这个差异。一种更复杂的策略是给不同的子赋予不同的值，依赖于它们的价值（比如国际象棋中的皇后和卒或西洋跳棋中的王与普通棋子的价值显然不同）或在棋盘上的位置。大多数博弈都为设计启发提供了无限的想象空间。

博弈图是按层搜索的。正如图 4-20 所示，MAX 和 MIN 轮换移动。棋手的每一次移动定义了一个新的图层。博弈程序通常预判固定的层数，大多时候这由计算机的空间和时间限制决定。在被预判的那一层，博弈程序计算各个状态的启发值，然后按照极小极大规则向上传播这个值。接

下来搜索算法使用这些导出值（derived value）选择下一步的移动。

在确定了所选层中每个状态的评估值后，程序把这个值向上传播到每个父状态。如果父状态在 MIN 层，那么孩子中的最小值被传递上去。如果父状态在 MAX 层，那么极小极大算法把孩子中的最大值赋给它。

努力使 MAX 层中的父状态最大化，MIN 层中的父状态最小化，算法就这样把值传递到当前状态的孩子。然后当前状态使用这些值来选择它的孩子。图 4-21 显示了对假想的状态空间使用 4 层预判极小极大算法的情况。

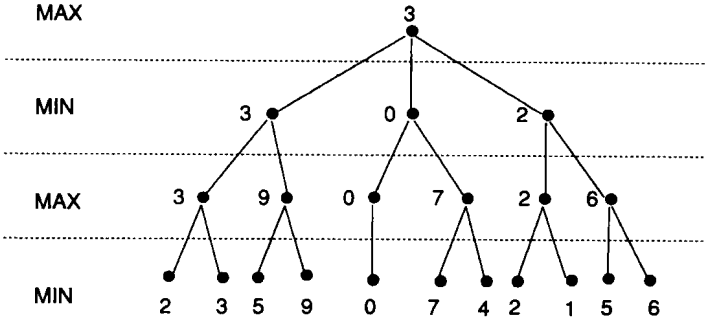


图 4-21 假想状态空间的极小极大过程

注：叶结点显示了启发值，内部状态显示了向上传播的值

最后，我们归纳一下极小极大过程的几点不足。首先，也是最重要的一点是，在（预先确定的）固定层深情况下它所做出的评估可能完全是误导性的。因为，当把一个启发用于有限深度预判时，在这个预判深度内无法探测出一个有希望的路径是否会在以后的博弈中产生坏的结果。如果对手给出一个车作为引诱来取你的皇后，而且评估过程仅预判到对手献出车这一层，那么评估就会优先考虑这个状态。但不幸的是，选取这个状态可能导致输掉整盘棋。这就是所谓的*地平线效应*（horizon effect）。即由于被搜索有限层深度时所发现的特别好状态引诱而遭到还击。不过，即使在一些重要的领域中，这种有选择的搜索深入方法也无法避免地平线效应。因为搜索必须在某个深度停止，那么就看不见这一点以外的状态了。

在对基本的启发性评估应用极小极大算法的过程中还可能发生另一种效应。当对空间中非常深层的状态进行评估时，评估结果可能由于发生在过大的深度而存在偏差（Pearl 1984）。与乘积的平均不同于平均的乘积一样，极小极大的估计（这是我们需要的）不同于估计的极小极大（这是我们所做的）。从这个意义上来说，尽管对于评估结果的更深的搜索通常意味着更好的搜索，但未必总是如此。如果要了解关于这个问题的进一步讨论以及可能的补救措施，请参阅 Pearl（1984）。

为了总结极小极大过程，我们以九宫游戏为例描述如何应用这个过程（见 4.0 节），这个例子摘自 Nilsson（1980）。在这个应用中使用了一个更复杂的启发，这个启发试图度量博弈中的对抗。具体来说，对于一个要度量的状态，这个启发数出对于 MAX 来说存在的所有胜利路线，然后减去对于 MIN 来说的所有胜利路线。搜索的任务就是努力使这种差异最大化。如果一个状态对于 MAX 来说是必胜的，那么它被评估为 $+\infty$，如果一个状态对于 MIN 来说是必胜的，那么它被评估为 $-\infty$。图 4-22 显示了把这一启发应用到几个实例状态的情况。

图 4-23、图 4-24、图 4-25 画出了应用图 4-22 所示启发的两层极小极大过程。图中显示了很多重要的过程，比如启发评估、极小极大倒推、MAX 的移动，还有对于等值的移动应用某种类型的平分决胜局（tiebreaker）（摘自 Nilsson，1971）。

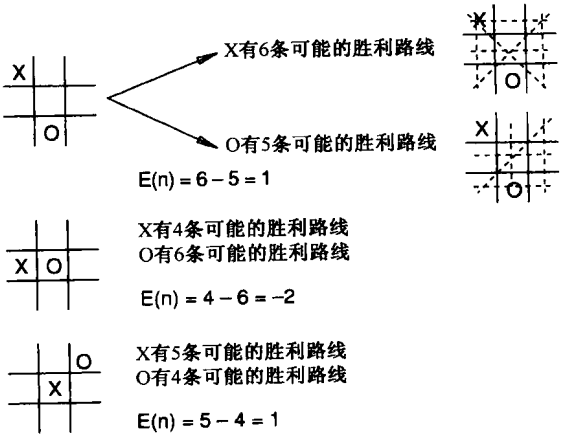


图 4-22　应用到九宫游戏中各个状态的度量对抗的启发

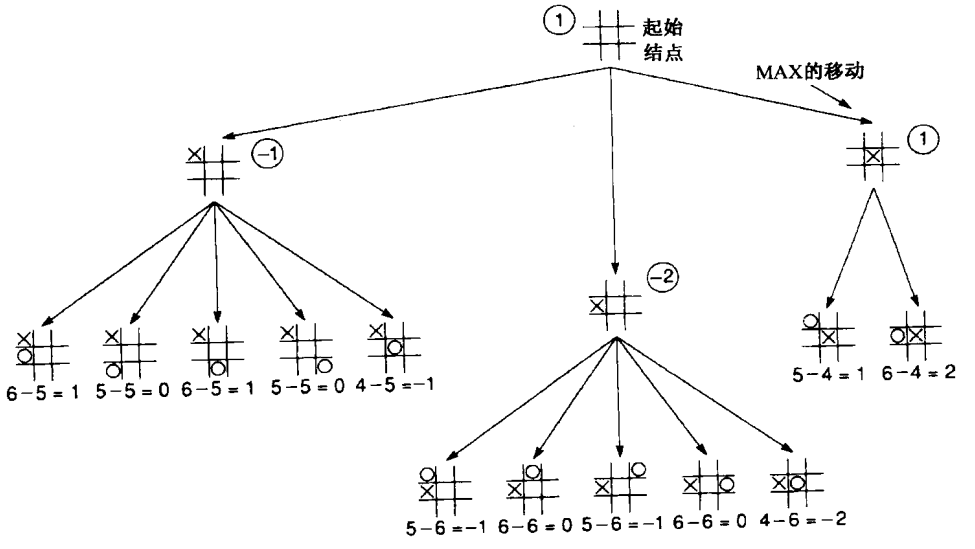


图 4-23　应用到九宫游戏开局移动的两层极小极大过程
[摘自 Nilsson（1971）]

4.4.3　α-β 过程

单纯的极小极大过程需要对搜索空间进行两遍分析，第一遍是向下降到预判层并在那里应用启发评估，第二遍是沿树向上传播评估值。极小极大过程追索空间中的所有分支，包括许多可以被更智能算法所忽略或修剪掉的分支。为此，博弈研究者开发出一种称为 α-β 剪枝（alpha-beta pruning）的技术来提高双人博弈的搜索效率。这种技术最初是在 20 世纪 50 年代提出的（Newell and Simon 1976）。

α-β 搜索的基本思想是很简单的：α-β 搜索并不搜索预判深度的整个空间，而是以深度优先的方式前进。在搜索中产生两个值，分别称为 α 和 β。α 值与 MAX 结点相关联从不减小，而 β

值与 MIN 结点相关联从不增大。假定 MAX 结点的 α 值为 6。那么 MAX 不必考虑其下任何关联值小于等于 6 的 MIN 结点。α 是假定 MIN 也尽最大努力时 MAX 可以获得的最差“分数”。类似地，如果 MIN 的 β 值为 6，那么它不需要考虑它下面的关联值等于或大于 6 的 MAX 结点。

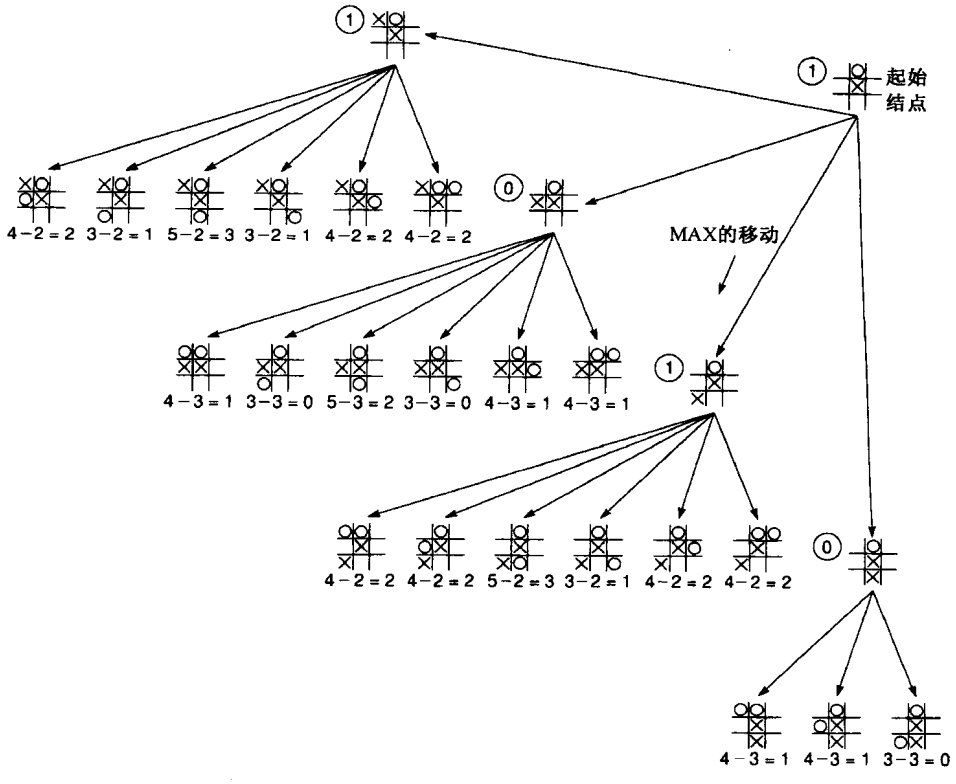


图 4-24　两层极小极大过程和两种可能的 MAX 第二步移动

[摘自 Nilsson（1971）]

要开始 α-β 搜索，我们先以深度优先的方式下降到预判层并对每一个状态和它的所有孩子应用启发性评估。假设这些孩子是 MIN 结点，那么这些 MIN 值中的最大值被倒推回父结点（MAX 结点，就像极小极大过程一样）。然后这个值被提供给这些 MIN 结点的祖父结点作为一个潜在的 β 截止点。

接下来，算法下降到其他的曾孙结点，如果它们中的任一个值等于或大于这个 β 值，那么就终止探索它们的父结点。对 MAX 结点的曾孙结点的 α 剪枝过程与此类似。

建立在 α 和 β 值基础上的两条搜索终止规则如下：

1）在任一个 MIN 结点下，如果发现了一个 β 值小于或等于它的任一个 MAX 祖先的 α 值，就可以终止搜索。

2）在任一个 MAX 结点下，如果发现了一个 α 值大于或等于它的任一个 MIN 祖先的 β 值，就可以终止搜索。

因此，α-β 剪枝表述了第 n 层结点和第 n+2 层结点之间的关系，根据这一关系可以从考虑中排除始于第 n+1 层的整个子树。以图 4-21 所示空间为例，对其应用 α-β 剪枝，其结果显示在图 4-26 中。注意：倒推值的结果与极小极大过程的结果是完全相同的，但是它与极小极大过程相比所节约的搜索是很可观的。

如果搜索空间中状态的排列顺序很有利，那么在同等的计算机时间和空间条件下 α-β 剪枝

可以使搜索速度提高一倍（Nilsson 1980）。如果状态的排列顺序特别不利，那么 α-β 搜索的空间也不比正常的极小极大过程多，而且仅需要扫描空间一遍。

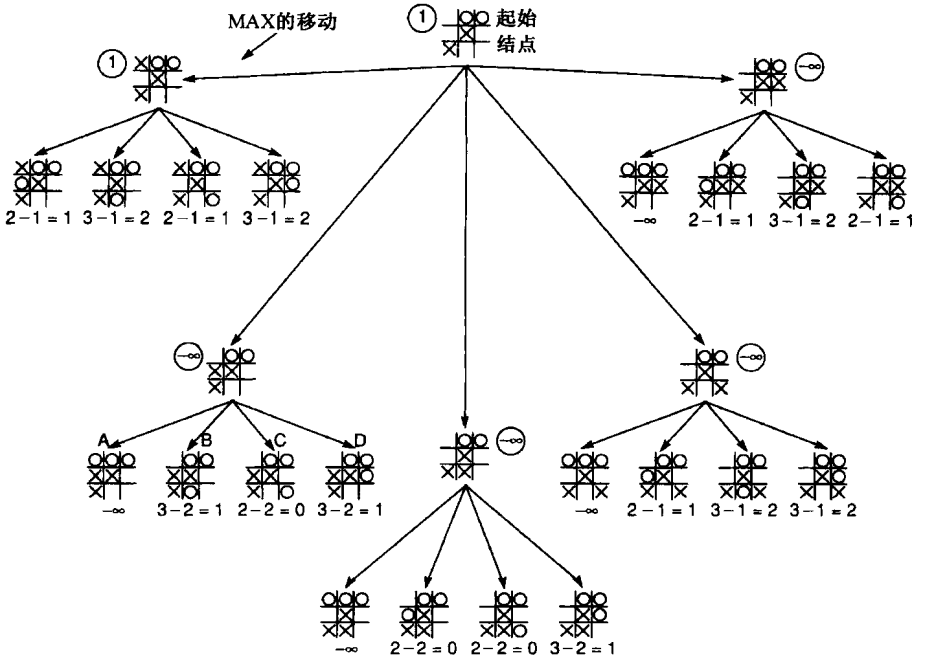
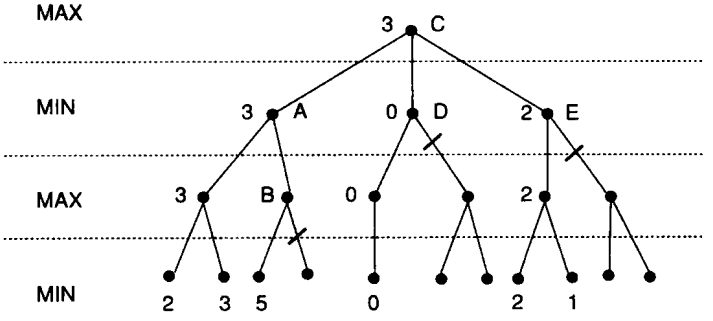


图 4-25　对 X 在接近终局的移动应用两层极小极大过程

［摘自 Nilsson（1971）］



A有β=3 (A将不会大于3)
B是β剪枝，因为5＞3
C是α=3(C将不会小于3)
D是α剪枝，因为0＜3
E是α剪枝，因为2＜3
C是3

图 4-26　对图 4-21 的状态空间应用 α-β 剪枝

注：没有数字的状态将不会被估算

4.5 复杂度问题

组合问题的最大难处在于组合“爆炸”经常发生在程序设计者没有意识到它会发生的情况下。因为大多数人类活动（计算活动和其他活动）都是发生在线性时间世界中，所以我们很难理解指数增长。我们会听到这样的抱怨：“只要我的计算机更大（或者说更快或并行程度更高），那么我的问题就会解决。”这样的感慨经常是对因组合爆炸而导致的失败而发的，但大多时候这都是一些废话。因为，真正的原因往往是没有恰当地理解问题或者没有针对具体条件下的组合增长采取合适的措施。

组合增长的速度是令人惊愕的。据估计，对国际象棋中可能移动所组成的空间进行完全搜索所产生的状态总数大约是10^{120}。这不是一般意义上的庞大数字，它可以与宇宙中的分子总数或自从“大爆炸”迄今经过了多少个纳秒相比。

已经开发了一些度量尺度用来帮助我们计算复杂性。其中之一就是空间的分支因子（branching factor）。我们把分支因子定义为可以从空间中的任意状态展开的平均分支数（孩子数）。在搜索深度为n时，要搜索的状态数等于分支因子的n次方。一旦计算出了空间的分支因子，那么就有可能估计出产生一个任意确定长度的路径所需的搜索代价。图4-27给出了B（分支因子）、L（路径长度）和T（搜索的状态总数）之间的关系（较小值的情况）。这个图是按T的对数绘制的，所以在图中L看起来不是直线。

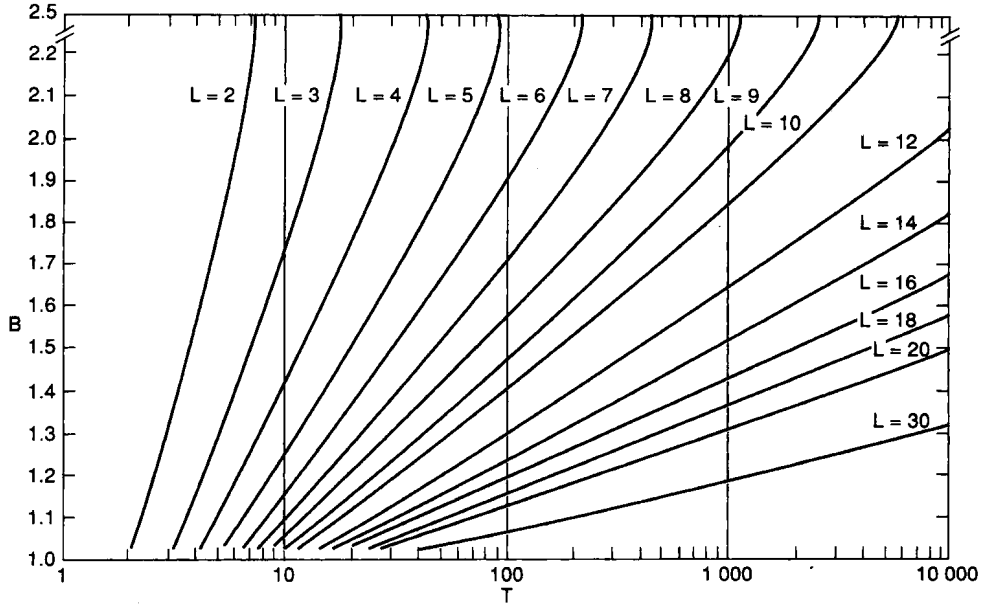


图4-27 产生的结点数相对分支因子B和不同解路径长度L的函数曲线

注：它们的关系公式为：$T = B(B^L - 1) / (B - 1)$［摘自Nilsson（1980）］

分析这幅图的几个实例可以说明事情是如何变糟糕的。如果分支因子为2（比如在二叉树中），那么要分析扩展到搜索空间6层深度的所有路径需要大约搜索100个状态。如果要考虑12次移动这样深度的路径，那么需要搜索大约10000个状态。如果分支因子被减小到1.5（通过某种启发），那么长为12的路径只需考虑几百个状态。

产生图4-27所示关系的数学公式是：

$$T = B + B^2 + B^3 + \cdots + B^L$$

其中，T 是状态总数，L 是路径长度，B 是分支因子。这个公式可以简化为：

$$T = B(B^L - 1)/(B - 1)$$

对搜索空间的度量通常是一种试验过程，要反复多次求解这个问题并检验它的变化。举例来说，假定我们希望得出 8 格拼图游戏的分支因子。我们先计算可能移动的总数：从每个角有两种移动总共有 8 种；从每个边的中心点有 3 种移动总共有 12 种移动；从方格的中央有 4 种移动；所以全部加起来移动数有 24 种。把这个数字除以 9——空位的不同可能位置数，得出平均的分支因子为 2.67。从图 4-27 可以看出，这个数字对于很深的搜索是非常不利的。如果我们排除直接返回到父状态的移动（已经加入到本章讨论的算法中），那么每个状态的可能移动数就减少 1 种。这就使分支因子变为 1.67，这是一个很可观的改善，在某些状态空间中，这甚至使穷举搜索成为可能。

正如我们在第 3 章中所考虑的，算法的复杂度也可以用 open 和 closed 列表的大小来衡量。一种使 open 列表的大小保持在一个合理范围内的方法是仅在 open 中保存一定数量的（启发性）最好状态。这可以使搜索更加集中，但也有排除最好的或者甚至是惟一的解路径的风险。这种维护列表大小上限的技术被称为*定向搜索*（beam search）。

为了降低搜索的分支（即限制搜索的空间），我们介绍了*更高信息度*启发的概念。搜索策略所用启发的信息度越高，那么为了得到最短路径解所必须搜索的空间就越小。但正如在 4.4 节中所指出的，进一步削减搜索空间所需额外信息的运算开销可能是不可接受的。在计算机上求解问题时，只找到最短路径还是不够的。我们还必须最小化 CPU 的总开销。

图 4-28 所示的分析摘自 Nilsson（1980），这是对以上问题的一种非形式描述。“信息度”坐标表示为了提高性能在评估启发中所包含的信息量。“计算开销”坐标标志了为实现状态评估和算法的各个其他要素所需的计算开销。随着启发中所包含信息的上升，启发所需的计算开销也在上升。类似地，随着启发信息度的提高，评估状态的计算开销变小，因为要考虑的状态数变少了。然而最关键的开销是计算启发**加上**评估状态的总开销，通常最小化这个开销才是我们所希望的。

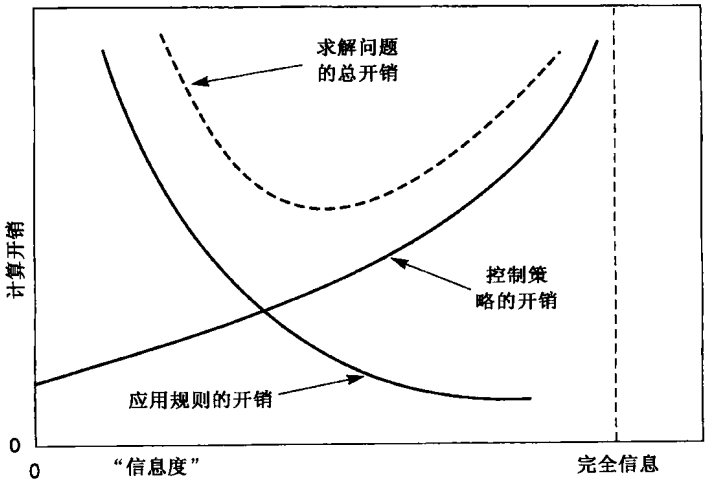


图 4-28　搜索开销和计算启发评估的开销相对启发信息度的粗略曲线

［摘自 Nilsson（1980）］

最后要指出，与或图的启发搜索是值得关注的一个重要领域，因为专家系统所用的状态空

间经常是这种形式。对这种结构的搜索渗透到本章和前一章的各个部分。因为要发现目标就必须搜索所有的与孩子结点，所以对搜索与结点代价的启发性估计就是对搜索孩子代价估计的总和。

然而，除了对个体与状态的数值评估，在与或图研究中还有许多其他启发式问题，比如在基于知识的系统中用到的启发式。举例来说，如果解决父状态需要与孩子集合都满足，那么应该先考虑哪个孩子？哪个状态最值得评估？哪个状态最有可能失败？哪个状态人类专家会优先考虑？这些决策对计算知识系统的有效性和整体代价都很重要，如医学或其他诊断测试。这些问题以及其他一些相关的启发式问题将在第8章再次探讨。

4.6 结语和参考文献

很多有趣问题的搜索空间往往是按指数增长的；启发式搜索是对付组合爆炸复杂性的首要工具。本章介绍了实现启发式搜索的各种控制策略。

本章以两个传统算法开始，爬山和动态规划，它们都继承自运筹学。我们建议阅读 Arthur Samuel（1959）的文章，其中讨论了他的下棋程序以及程序中对爬山和极小极大搜索的精妙运用。Samuel 还介绍了早期两个有意思的例子，精妙的内存管理系统和能学习的程序。Bellman（1956）设计的动态规划算法在必须要比较字符串、单词或语音的领域仍然很重要，比如自然语言处理。动态规划经常被称为前前后后算法或 Viterbi 算法。用动态规划进行语言分析的重要例子在 Jurafsky 和 Martin（2009）的文章和第15章中介绍。

接着，我们在关于传统的状态空间搜索的上下文中介绍了启发式。我们介绍了实现最佳优先搜索的 A 算法和 A* 算法。启发式搜索是通过8格拼图等简单游戏来说明的，然后扩展到基于规则的专家系统产生的更复杂的问题空间（第8章）。本章还将启发式搜索应用到了二人游戏，使用带极小极大和 α-β 剪枝的预判来尝试预测对手的行为。在讨论了 A* 算法后，我们分析了它们的行为，考虑了可采纳性、单调性和信息度等属性。

复杂度理论这门学科已经渗透到计算科学的几乎所有分支，特别是状态空间的增长和启发性修剪。复杂度理论分析的是问题（与应用于这些问题的算法相对）的固有复杂性。复杂度理论中的一个核心假说是存在一类固有的难以驾驭的问题。这类被称为 NP-hard（Nondeterministically Polynomial）的问题是由那些如果不借助启发就不可以在少于指数时间内求解的问题所组成的。几乎所有的搜索问题都属于这一类。关于这方面的材料我们特别推荐 Michael R. Garey 和 David S. Johnson（1979）所著的《Computers and Intractability》以及 Bernard Moret 和 Henry Shapiro（1991）所著的《Algorithms from P to NP, Vol. I: Design and Efficiency》。

Judea Pearl（1984）所著的《Heuristics》一书全面讨论了启发算法的设计和分析。R. E. Korf（1987, 1998, 1999, 2004）一直在研究各种搜索算法，包括迭代加深算法的分析和 IDA* 算法的开发。IDA* 算法把迭代加深和 A* 算法集成到一起，目的是在启发式搜索中把 open 列表限制在线性范围内。国际象棋和其他博弈程序一直是 AI 历史的永恒主题（Hus 2002），各种年会上经常发表和讨论这方面的成果。

我们非常感谢 Nils Nilsson（1980），因为本章的很多例子和方法来源于此。

4.7 习题

1. 扩展九宫游戏的“最多胜利”启发式，在图4-3的搜索空间中多深入两层。用这一启发式检查的全部状态有多少个？传统的爬山算法在这种情况下能工作吗？为什么？
2. 用动态规划算法的回溯部分找出图4-6中字符串的另一种最佳对齐。一共有几种最佳对齐？
3. 以4.1.2节的 Levenshtein 距离为度量，用动态规划确定源串 sensation 和 excitation 到目标串 execution 的最

小编辑距离。

4. 给出一种启发使堆积木程序可以用它来求解这种形式的问题“把积木 X 堆在积木 Y 上”。这个启发是可采纳的吗？单调吗？
5. 滑动将牌游戏由三个黑色将牌和三个白色将牌组成，而且留有一个空位，如图 4-29 所示。

B	B	B		W	W	W

图 4-29　滑动将牌游戏

这个游戏有两种不同代价的合法移动：

一个将牌可以移动到紧邻的空位。这种移动的代价是 1。一个将牌可以跳跃一或两个其他的将牌进入空位。这种移动的代价等于它所跨越的将牌数。

游戏的目标是使所有的白色将牌在所有的黑色将牌左边。空位的位置是无关紧要的。

a）从复杂度和循环的角度分析这个状态空间。

b）提出一种可以用来求解这种游戏的启发，并从可采纳性、单调性和信息度方面对其进行分析。

6. 把图 4-14 所示的三种 8 格拼图游戏的启发与错位牌的距离和加上直接颠倒数乘 2 这个启发进行比较。考虑以下这几个方面：
   a）估计到目标距离的精度。这需要你首先导出最短路径解，然后再以其为标准。
   b）信息度。哪一个启发会最有效地修剪空间？
   c）8 格拼图游戏的这三种启发都是单调的吗？
   d）可采纳性。这三种启发中的哪一个是以到达目标的路径的实际代价为上限的？在一般情况下证明你的结论，或者给出一个反例。
7. a）正如文中所指出的，最佳优先搜索使用 closed 列表来实现循环探测。取消这一探测，依赖对深度的测试 g(n) 来探测循环会有什么影响呢？比较这两种方法的效率。
   b）best-first-search 直到状态从 open 中移出时才测试它是否满足了目标。也可以在产生一个新的状态时进行这种测试。这样做在算法的效率方面会有什么影响？对可采纳性有影响吗？
8. 证明 A* 算法是可采纳的。提示：这个证明应该说明：
   a）A* 搜索会终止。
   b）在算法的执行期间，在 open 中总是存在一个结点是位于通往目标的最优路径上。
   c）如果存在到达目标的路径，那么 A* 算法会以发现最优路径而终止。
9. 启发的可采纳性是否蕴涵了单调性？如果不是，你能否描述出何时可采纳性蕴涵了单调性？
10. 证明 A* 算法所展开的状态集合是宽度优先搜索所展开的状态集合的子集。
11. 证明信息度更高的启发所展开的搜索空间更少或相等。提示：参见 4.3.3 节中的论据。
12. 凯撒加密（Caesar cipher）是一种简单的加密方案。它的原理是对字母表进行循环排列，即把字母表的第 i 个字母替换为第（i + n）个字母。例如，如果对“Caesar”应用偏移为 4 的凯撒加密，那么它会被加密为“Geiwev”。
    a）给出可以用于求解凯撒加密问题的三种启发。
    b）在一种简单的替换加密中，根据某种任意的一对一映射，每个字母被另一个字母替代。你提出的用于凯撒加密的启发中的哪一种启发可用于求解这种替换加密问题？请给出解释。（感谢 Don Morrison 提出这个问题。）
13. 对图 4-30 所示的树执行极小极大过程。
14. 对练习 13 中的树执行从左到右的 α-β 剪枝。再对同一棵树进行从右到左的 α-β 剪枝。讨论为什么所发生的剪枝是不同的？
15. 考虑三维的九宫游戏。讨论表示的问题；分析状态空间的复杂度。提出一种玩这种游戏的启发。
16. 对图 4-23、图 4-24 和图 4-25 中的九宫游戏搜索进行 α-β 剪枝。分别可以修剪掉多少个叶结点？

17. a）定义一种启发式搜索与或图的算法。注意，如果要求解双亲结点，那么必须求解一个与结点的所有后继。因此，在计算到目标代价的启发性估计时，求解一个与结点的代价的估计至少等于求解不同分支的估计的总和。

b）用这一算法搜索图 4-31 所示的图。

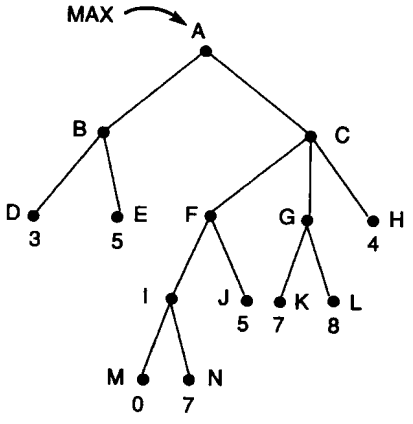
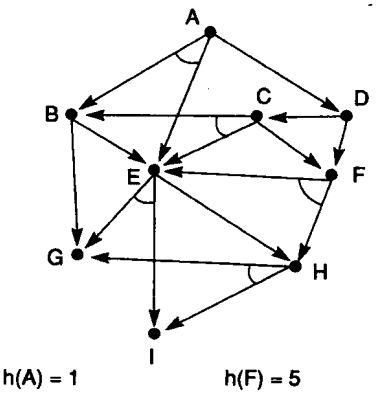


图　4-30



h(A) = 1　　h(F) = 5
h(B) = 2　　h(G) = 1
h(C) = 5　　h(H) = 3
h(D) = 3　　h(I) = 1
h(E) = 2

图　4-31

第5章 随机方法

上帝不掷骰子……

——爱尔伯特·爱因斯坦（他对有关量子论的可靠性的回答）

上帝不但掷骰子，而且有时还把它们投到人们看不到的地方……

——史蒂芬·霍金

几乎的不可能也比不大可能的可能要好得多……

——亚里士多德，鲍比·沃尔夫在《Aces on Bridge》中的引句，2003

5.0 简介

第4章介绍了启发式搜索方法，它可以对一个没有精确解的问题或者是盲目搜索整个状态空间代价太高的领域进行问题求解。在本章，将介绍适于解决上述问题的随机方法学。概率推理也适合于通过在信息库中抽样获取状态信息和通过数据学习而得出因果模型的场合。

随机方法学的一个重要的应用领域是诊断推理。在诊断推理中，因果关系并非是以一种完全纯粹的确定性方式存在。这点在第2、3、4章中的基于知识的问题求解的方法中可以看到，并且在第8章中我们还将再次看到。一个诊断的场合通常只是呈现出迹象，比如发烧或头痛，而没有给出更进一步的原因分析。实际上，一种迹象通常可以表现出好几个不同的原因，例如，发烧可以起因于流感或传染。在这些场合，我们将利用相关概率信息，它们不但经常能指出一个迹象的可能解释原因，并且还能将它们按优先次序列出来。

随机方法学的另一个有趣应用是在赌博上。在赌博中，通常是由诸如扔骰子、打牌或是转轮盘赌轮这样的随机事件来决定谁是赢家。实际上在18世纪，帕斯卡（Pascal）以及后来的拉普拉斯（Laplace）都一直在研究概率演算，而试图为赌博提供一种数学依据成为他们研究的一个极大的激励。

最后，需要指出的是，正如1.1.4节所评述的，在某一情形下，对智能的考察表明人类的决断力经常是从复杂的、关键性时刻的、具体化的周围环境中显露出来的，不过在这些环境下，完全机械的演算几乎是无法定义的，或者即便能被定义，它也可能无法在一个可用时间框架下计算出结果。在这些场合中，智能行为可能最适合于被看作为对预测成本和利益的随机反应。

接下来，我们将描述几个问题域。在其中的许多方面，随机方法学都将经常会被用于智能计算的实现，而这些方面也将是以后几章讨论的主题。

1）**诊断推理**。在医疗诊断中，例如，在病人的一系列症状和产生这些症状的原因之间并不总是有明显的因果关系。实际上，同样一类症状经常表现出多个可能的原因。另外，随机模型在涉及复杂的机械力学的场合也是同样重要的，如监控飞机或直升机的系统。基于规则（第8章）和概率性（第9章和第13章）的系统都已应用在了这些及其他的一些诊断领域。

2）**自然语言理解**。如果一台计算机要理解和使用人类的语言，那么它必须能够表示人类如何使用他们自己的语言。可以学习词、表达和比喻，而且随着时间的推移，会不断变化和发展。随机方法学能支持对语言的理解；例如，一个计算系统在一个使用专用语言（称作*语料库语言学*）的数据库上进行训练。本章的后边以及第15章讲解自然语言理解问题。

3）**规划和调度**。当主体（agent）制定一个规划，例如，小汽车度假旅游，往往情况是根本

没有一系列确定性操作以确保整个计划的成功。如果汽车抛锚了，如果在某个特殊的日子汽车渡船停航，如果旅馆完全满了，即使你之前预订过，试问这些情况你该怎么办？特定的规划，无论对于人还是机器来说，都常用概率语言表示。规划会在8.4节介绍。

4）**学习**。上述的三个针对随机方法技术的方面，可以看作为自主学习领域。许多随机系统的一个重要组件是它们能够选择场合及不断学习。一些先进的随机系统不仅能采样数据和预测结果，而且能根据获得的数据和结果，学习新的概率关系。本书第四部分介绍学习理论，利用随机方法进行学习将在第13章介绍。

随机方法学有自己的基本计数性质。在一个特定环境下，一个事件的概率可描述成此事件能发生的方式数除以所有可能结果的总数。因此，任投一个骰子，所得点数是偶数的概率就是所有偶数结果（2、4或6）的总数除以所有可能结果（1、2、3、4、5或6）的总数，即为1/2。同样，从一袋大理石中取出某一种颜色的大理石的概率，就是袋中该颜色的大理石个数除以大理石的总个数。在5.1节，要介绍基本的计数方法，包括求和规则和乘法规则。因为其在计算中的重要性，我们也介绍随机事件的*排列*和*组合*。这一节是选读的，对于具有良好离散数学知识背景的读者可以跳过。

在5.2节，介绍一种采用随机方法学进行推理的形式语言，包括对相互独立以及不同类型的随机变量的定义。例如，对于一个概率统计问题，随机变量可能是*布尔型*——真或假，可能是*离散型*，像掷公平骰子时从整数1到6，也可能是*连续型*，比如一个在实数上定义的函数。

在5.3节，我们将给出贝叶斯定理，它支持概率模型的大部分方法以及学习理论（9.3节和第13章）。贝叶斯规则在具有先验知识或有状态经验的背景下对解释新的迹象很重要。在5.4节，我们将给出随机方法的两种应用，包括概率有限状态自动机和根据抽样数据预测英语单词的方法。

在第9、13章，我们会继续对概率法和推理进行介绍，包括*贝叶斯信念网络*（Bayesian Belief Network，BBN）、*隐马尔可夫模型*（Hidden Markov Model，HMM）和支持随机模型的一阶表示系统。这些模型利用有向无环图（DAG）的形式，被称为*图模型*。在第13章我们介绍机器学习的随机方法。

5.1 计数基础（选读）

随机方法学的基础要能够统计出某一应用领域的元素个数。用于收集和计数元素的基础当然是集合论。在集合论中，必须能明确判定一个元素是否是一个集合中的元素。只有确定这个，才可能讨论对集合、集合的补的元素个数进行计数，以及对多个集合求并集和交集等的方法。在本节中，我们将回顾这些方法。

5.1.1 加法和乘法规则

有一个集合A，集合A的元素个数用$|A|$来表示，$|A|$称作为A的*基数*。集合A可能是空集（元素个数为0）、有限集、可数的无限集或者是不可数的无限集。每个集合都是根据一个让人感兴趣的域或者是全集U中的元素进行定义的。例如，在一个教室里，男同学所构成的集合是以房间里所有人为背景或全集定义的。类似地，在掷骰子游戏中，扔出一个为3的数可视为具有6个可能结果（元素）集合的其中一个元素。

集合A的域或全集U也是一个集合，而且利用全集U，可以得出集合A的*补集* $\overline{A}$。例如，上面提到的男同学集合的补集就是教室里所有女同学构成的集合，扔出一个为3的数所构成的集合$\{3\}$的补集是$\{1, 2, 4, 5, 6\}$。如果集合A中的每一个元素都是集合B中的元素，那么称

集合 A 是集合 B 的子集，记为 A⊆B。因此，显而易见，每个集合都是自己的子集，任何一个集合 A 都是全集的子集，空集（通常记为 {} 或∅）是任意一个集合的子集。

集合 A 和 B 的并集（A∪B）的所有元素，要么在集合 A 中要么在集合 B 中。两个集合的并集元素个数是这两个集合各自的元素个数的总和，减去这两个集合中所共有的元素的个数。毫无疑问，只要考虑到集合中每个不同的元素仅可以计数一次，上述结论就不难证明了。一般地，如果两个集合没有相同的元素，那么它们并集的元素个数就等于这两个集合各自的元素个数的总和。

集合 A 和 B 的交集（A∩B）的所有元素为集合 A 和 B 所共有。下面就刚才定义的几个概念举一些例子。

假设全集 U={0,1,2,3,4,5,6,7,8,9}，A={1,3,5,7,9}，B={0,2,4,6,8}，C={4,5,6}，那么|A|=5, |B|=5, |C|=3, |U|=10。而且 A⊆U, B⊆U, A∪B=U, |B|=|A|, $A=\overline{B}$。此外|A∪B|=|A|+|B|=10，因为 A∩B={}；但是|A∪C|=|A|+|C|−|A∩C|=7，因为 A∩C={5}。

刚刚已给出了用于合并两个集合的*加法规则*的主要组成部分。对任意两个集合 A 和 C，它们并集的元素个数按如下式子计算：

$$|A \cup C| = |A| + |C| - |A \cap C|$$

注意，上面的式子对于任意两个集合都适用，不论它们是否有共同的元素。类似地，还有适用于三个集合（A，B，C）的加法规则，同样也不管它们之间是否有共同元素：

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

在本章的前面曾有过类似的论证，同样可以用来证明该等式。类似的包含/排斥等式也能用于计算三个以上集合的并集的元素个数。

*用于计数的乘法规则*指出：如果有两个集合 A 和 B，分别有 a 个和 b 个元素，那么将有 a×b 种方式把这两个集合的元素组合在一起。因为，集合 A 中有 a 个元素，而且每个元素都能与集合 B 中的元素形成 b 对匹配。乘法规则适用于计数中的许多方法，包括集合的笛卡儿乘积及集合的排列与组合问题。

集合 A 和 B 的*笛卡儿乘积*用 A×B 来表示，是所有的有序对(a,b)的集合，其中(a,b)中的 a 是集合 A 中的元素，b 是集合 B 中的元素。可形式地表示为：

$$A \times B = \{(a,b) \mid (a \in A) \wedge (b \in B)\}$$

用计数的乘法规则表示则为：

$$|A \times B| = |A| \times |B|$$

笛卡儿乘积也可以在任意多个集合上定义。n 个集合的笛卡儿乘积将得到一个 n 元组的集合，该 n 元组的第一个分量是第一个集合中的任意一个元素，第二个分量是第二个集合中的任意一个元素，依此类推。n 元组的总个数是所有集合的元素数目的乘积。

5.1.2　排列与组合

对一个集合中的元素进行*排列*是将此集合中的元素按照某一顺序重排。在对这些元素进行排列时，每个元素只可能使用一次。一个排列的例子是将 10 本书按某种顺序搁放到一个能容纳 10 本书的书架上，另一个排列的例子是给 6 个孩子中的 4 个孩子分配特殊的任务。

我们常常想知道一个具有 n 个元素的集合，到底会有多少种（不同的）排列。我们使用乘

法规则来计算。如果集合 A 中有 n 个元素，那么这些元素的排列是一个长度为 n 的序列。该序列的第一个元素是集合 A 中 n 个元素的任意一个，第二个元素是集合 A 中剩余的（n－1）个元素的任意一个，第三个元素是集合 A 中剩余的（n－2）个元素的任意一个，依此类推。

一个元素在排列序列中的位置并不重要，也就是说，集合的 n 个元素的任意一个可首先放在序列中的任意一个位置，剩下的（n－1）个元素的任意一个可放在序列中剩下的（n－1）个位置的其中一个，依此类推。最后，由乘法规则可得，一个具有 n 个元素的集合有 n! 种排列。

我们可以限制一个集合 A 的排列中集合 A 的元素个数，不过该数目应该大于等于 0 且小于等于原集合 A 的元素总个数 n。例如，我们可能想知道将 10 本书放到一个一次只能容纳 6 本书的书架上，会有多少种不同的顺序。如果想计算从一个具有 n 个元素的集合中一次取出 r（$0 \leqslant r \leqslant n$）个元素会有多少种排列，仍可以用如前所述的乘法规则，只不过现在每个排列序列中只有 r 个位置空间：

$$n \times (n-1) \times (n-2) \times (n-3) \times \cdots \times (n-(r-1))$$

换一种方式，还可以将上式写成：

$$\frac{n \times (n-1) \times (n-2) \times (n-3) \times \cdots \times (n-(r-1)) \times (n-r) \times (n-r-1) \times \cdots \times 2 \times 1}{(n-r) \times (n-r-1) \times \cdots \times 2 \times 1}$$

或者等价地，将从一个有 n 个元素的集合中一次取出 r 个元素的排列数记为 ${}_nP_r$，那么这个排列数为：

$${}_nP_r = \frac{n!}{(n-r)!}$$

一个具有 n 个元素的集合的组合是 n 个元素形成的任一子集。和排列一样，对于一个给定的项的集合，我们也常常想计算出项所形成的组合数。因此，从一个有 n 个元素的集合中取 n 个元素的组合只有一个。核心思想体现在组合的数目是由整个集合所产生的相应子集个数决定的。在上述书架例子中，组合表示从整个 10 本书的集合中抽取 6 本书放在书架上形成的各个不同子集。另一个组合的例子是从 15 个人中选出 4 个人，组成一个委员会，每个人要么在委员会中要么不在，并且第一个选中的和最后一个选中的委员会成员没有任何的差别。更进一步的组合例子是一个 5 张扑克牌的纸牌游戏，在该游戏中，一手牌的最终好坏不取决于发牌的顺序（当然，如果是在赌博中，而且前 4 张牌均为正面的情况下，最后一张牌至关重要；不过本例中发牌的顺序与牌的最终好坏无关）。

从 n 个元素中一次取 r（$0 \leqslant r \leqslant n$）个元素的组合数用 ${}_nC_r$ 表示。确定此组合数的直接方法为，先按照前面介绍的方法，求出从 n 个元素的集合中取出 r 个元素的排列数 ${}_nP_r$，然后减去重复集合的数目。因为任意一个 n 个元素的集合的 r 个元素的子集都有 r! 种排列，所以从 n 个元素中一次取 r 个元素的组合数，可由从一个具有 n 个元素的集合中一次取出 r 个元素的排列数除以 r! 求得。因此有：

$${}_nC_r = \frac{{}_nP_r}{r!} = \frac{n!}{(n-r)!\,r!}$$

上面提到的计数原理还有许多其他变化，其中一些会在本章习题中出现。要想对这些计数方法有进一步的了解，可以看看离散数学教程。

5.2 概率论基础

在已有 5.1 节所提到的计数规则基础的前提下，我们现在可以介绍概率论。首先，在 5.2.1

节，我们考虑一些基本的定义，比如两个或更多事件是否相互独立。在5.2.2节，我们将示范如何在特定的数据集合上进行推理论证。这会有助于我们对后面5.3节概率推理和5.4节贝叶斯定理中所用到的例子的理解。

5.2.1　样本空间、概率和独立性

下列定义是概率论的基础。这些定义是由法国数学家拉普拉斯在19世纪初（1816年）首先提出的。正如本章简介中提到的，当时的拉普拉斯正在尝试创造赌博领域的微积分！

定义

基本事件

一个**基本事件**或**原子事件**不能由其他的事件组成。

事件E

事件是基本事件的集合。

样本空间S

一个事件E的所有可能结果组成的集合称为**样本空间**S或该事件的**全集**。

概率p

对于一个事件E，S是它的样本空间，E的**概率**为E中元素个数除以样本空间S的所有可能结果数，即$p(E) = |E|/|S|$。

例如，随机掷两个骰子，所得点数为7或11的概率是多少？首先确定问题的样本空间。使用计数的乘法原理，每个骰子均有6个不同的点数，因此投两个骰子会有36种不同的结果。两个骰子的点数相加和为7的所有情况如下：1，6；2，5；3，4；4，3；5，2和6，1，一共有6种。掷两个骰子，所得点数和为7的概率是6/36 = 1/6。两个骰子的点数相加和为11有两种情况：5，6；6，5。于是，掷出11的概率为2/36 = 1/18。所以随机的掷两个骰子，得到点数为7或11的概率为1/6 +1/18，即2/9。

在上面的例子中，两颗骰子点数之和为7是一个事件，和为11也是一个事件。基本事件是掷这两个骰子得到的结果。因此，两颗骰子点数之和为7的这一事件是由（1，6），（2，5），（3，4），（4，3），（5，2）和（6，1）六个基本事件构成。整个样本空间是两个骰子的点数所组成的有序对集合，是所有36个基本事件的并集。而且因为例子中的两个事件之间没有共同的原子事件，所以它们是相互独立的，它们的和（并集）的概率就是各自概率的和，这点很快就会看到。

再举一个发扑克牌的例子，试问发5张扑克牌，其中4张牌大小相同的概率是多少？首先，该问题的样本空间是从52张牌中一次取5张牌的组合。使用乘法原理，就可以得到这个4张牌大小相同的概率。我们先确定如果5张牌中4张具有相同大小，则一共有13种选法（A，2，3，…，10，J，Q，K）。由于一副牌中，只有4种花色，因此要选4张大小相同的牌就必须全部选中这4种花色；最后一张牌将从挑出4张牌后剩下的48张牌中选出。因此，发5张扑克牌，其中4张牌是相同大小的概率为：

$$({}_{13}C_1 \times {}_4C_4 \times {}_{48}C_1) \,/\, {}_{52}C_5 = 13 \times 1 \times 48 / 2\,598\,960 \approx 0.000\,24$$

从上面给出的定义中，可以得出以下几个结论：

第一，样本空间S的任意一个事件E的概率为：

$0 \leqslant p(E) \leqslant 1$，其中 $E \subseteq S$

第二，样本空间S中的所有可能结果的概率和为1。样本空间S的定义表明它是所有独立事

件 E 的并集，知道了这一点就不难理解该结论。

第三，一个事件的对立事件的概率为：

$$p(\overline{E}) = (|S| - |E|) / |S| = (|S| / |S|) - (|E| / |S|) = 1 - p(E)$$

两事件互补的关系是一个很重要的关系。利用一个事件的对立事件求解该事件的概率常常更简单，例如，对于随机产生的一个 n 位的二进制数串，串中至少有一位为 1 的概率是多少？我们先得出此事件的对立事件是串中的每一位都为 0，然后求出该事件的概率是 2^{-n}。因此，原事件的概率为 $1-2^{-n}$。

最后，根据求事件的对立事件的概率公式，可以计算出不可能事件的概率：

$$p(\{\}) = 1 - p(\overline{\{\}}) = 1 - p(S) = 1 - 1 = 0, \text{或}$$
$$= |\{\}| / |S| = 0 / |S| = 0$$

两事件之间还有一个非常重要的关系：两事件的和（并集）的概率可由 5.1 节介绍的计数定理确定，即对于任何两个集合 A 和 B 有 $|A \cup B| = |A| + |B| - |A \cap B|$。由此可以确定一个样本空间 S 的任意两事件和（并集）的概率：

$$p(A \cup B) = |A \cup B| / |S| = (|A| + |B| - |A \cap B|) / |S|$$
$$= |A| / |S| + |B| / |S| - |A \cap B| / |S| = p(A) + p(B) - p(A \cap B)$$

当然，利用 5.1 节提到的容斥原理，能求出任意多个事件和的概率。

我们已经给出了一个求解两个事件和的概率的例子：随机地掷两个骰子，求出所得点数和为 7 或 11 的概率。在这个例子中，因为这两个事件是不相容的，所以上面的公式只是用来求解两个不相容事件和的概率。实际上，同时也可以把这个公式用于更加一般的场合，比如集合间是相容的。假设掷两个骰子，要计算这两个骰子点数和为 8 或出现相同点数的概率，只需简单地利用上述公式即可求出，其中这两个事件有一个共同的基本事件——(4,4)。

接下来，考虑两个独立事件的概率。假设 4 个人在打牌，每个人都分到同样多的牌。如果你没有黑桃 Q，那么可以推断其他三人每个人有黑桃 Q 的概率是 1/3。类似地，还可以推断其他每个玩家有红桃 A 的概率是 1/3，以及每个玩家同时有这两张牌的概率是 1/3 × 1/3，即 1/9。在本情形下，认为这两个事件是相互独立的，尽管目前看来只是大概正确。我们用一个定义来对它进行形式化。

定义（独立事件） 两个事件 A 和 B 是**独立的**当且仅当它们同时发生的概率等于两个事件各自发生的概率的乘积。该独立关系可如下表示：

$$p(A \cap B) = p(A) * p(B)$$

注：对于 $p(s \cap d)$，有时也用 $p(s, d)$ 来表示。5.2.4 节中会介绍条件概率，届时进一步地阐明独立的概念。

因为上述对事件的独立描述是一个当且仅当关系，所以可以通过得出两个事件的概率关系来判断它们是否独立。考虑随机产生的一个四位的二进制数。我们想知道该串含有偶数个 1 的事件与此位串以 0 结束的事件是否独立。利用乘法原理，由于每一位都有两种取值（0，1），将得到总共 $2^4 = 16$ 种不同的四位二进制数。

有 8 个四位二进制数是以 0 结束：{1110，1100，1010，1000，0010，0100，0110，0000}；同样也有 8 个四位二进制数，都包含偶数个 1：{1111，1100，1010，1001，0110，0101，0011，0000}；既以 0 结束又含有偶数个 1 的位串有四个：{1100，1010，0110，0000}。这两个事件是

独立的因为

$p(\{串有偶数个1\} \cap \{串以0结尾\}) = p(\{串有偶数个1\}) \times p(\{串以0结尾\})$

$4/16 = 8/16 \times 8/16 = 1/4$

再考虑随机产生的四位二进制数，试问位串含有偶数个1的事件与此位串以1结束的事件是否独立？如果两个或更多的事件不是独立的，也就是说，其中任一事件的概率都会影响其他事件的概率，那么我们需要用条件概率的概念来解决它们的关系。我们会在5.2.4节中学习它。

在结束本节之前，需要指出可能存在其他的支持概率论基础的公理系统，例如，对命题演算（2.1节）的一个扩展系统。关于基于集合的方法，俄国数学家Kolmogorov在1950年提出了下列定理的一个变体，该变体相当于上述对概率的定义。利用这3个定理，Kolmogorov系统地构建了整个概率论。

1）样本空间S的事件E的概率在0和1之间，即$0 \leqslant p(E) \leqslant 1$。

2）所有事件E的并集等于S，$p(S) = 1$，$p(\overline{S}) = 0$。

3）事件A和B的并集的概率是：

$$p(A \cup B) = p(A) + p(B) - p(A \cap B)$$

在下一节中，给出一个简单的概率推理的例子。

5.2.2　概率推理：一个道路/交通例子

现在举一个用刚刚介绍的思想进行推理的例子。假定你正在州际公路上驾驶，因为交通拥挤，你在逐渐地减速。你开始寻找导致减速的可能原因。莫非前方道路施工？或出现了交通事故？不过，你所能确定的只是你在不断地减速。但等等！假定你还能获取该州际公路的统计数字，并且通过使用车中的基于图形用户界面和推理的系统，可以把这些统计数据下载到汽车的计算机中。不过问题是，你拥有了这些数据，要如何处理它们？

对于本实例，假定有三个真或假参数（5.2.4节把该类型参数定义为布尔随机变量）。第一，整条公路上交通是否行驶缓慢或者你是否在减速，这种情形用S来表示，S可取值t或f；第二，是不是发生了交通事故，这种情形用A来表示，A可取值t或f；最后一种情况是此刻是否有道路施工，用C来表示，同样它的取值也是t或f。而且，利用汽车中的数据下载系统，能表示出该州际公路交通系统的相互关系，见表5-1。

表5-1　交通缓慢S，交通事故A和道路施工C的联合概率分布，其中S、A和C是5.4.2节例子中的随机变量

S	C	A	p
t	t	t	0.01
t	t	f	0.03
t	f	t	0.16
t	f	f	0.12
f	t	t	0.01
f	t	f	0.05
f	f	t	0.01
f	f	f	0.61

表5-1的各项几乎与2.1节所介绍的真值表一样，除了右边一列给出了左边相应栏的情形的概率。因此，从表中第3行的数据，我们可以得出整个交通行驶缓慢以及发生交通事故但无道路施工的概率为0.16：

$$S \cap \overline{C} \cap A = 0.16$$

应该注意到，我们一直都是在事件集合的背景下研究概率演算。图5-1显示了如何用传统的文氏图表示表5-1中的概率。同样也可以通过为每一个命题赋予概率真值来描述上述情形，在这种情况下，用∧代替∩，同时表5-1会解释成命题合取的真值表。

进一步，我们注意到表5-1中的联合分布的所有可能结果的概率和为1.0；这与5.2.1节中给出的概率定理是一致的。利用表5-1，可计算出此问题中的任何一个简单或复杂事件的概率。例如，要计算整个道路交通行驶缓慢S的概率，就是求所有S=t（true）的情形下的概率。我们对表格5－1中前4行的概率求和，求得值为0.32，这就是所求的概率。有时，我们称上述求得的概率为S的*无条件概率*或*边缘概率*。该处理称为*边缘化*，因为除了交通行驶缓慢这个随机变量以外的所有其他变量均不考虑。说得更精确些，一个变量的分布可以通过不考虑包含该变量的联合分布中的其他所有的变量来求得。

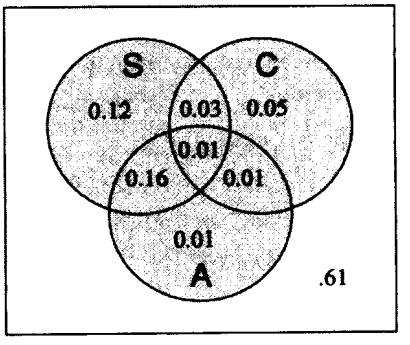


图5-1 表5-1中概率分布的文氏图表示，S表示交通缓慢，A表示事故，C表示道路施工

类似地，可以计算出当整个交通行驶并不缓慢（$\overline{S}$），但前方有道路施工（C）（这在新墨西哥州十分常见）的概率。本状态下概率为$p(\overline{S} \cap C) = t$，即将表5-1中的第5行和第6行的两概率相加，得到结果为0.06。如果考虑$C \cap \overline{S}$状态的对立面，使用德·摩根律得到$p(\overline{C} \cup S)$。利用5.2.1节中介绍的计算两集合并集的概率公式，可以得到：

$$0.16 + 0.12 + 0.01 + 0.61 + 0.01 + 0.03 + 0.16 + 0.12 - (0.16 + 0.12) = 0.94$$

不难发现，$C \cap \overline{S}$和它的补集（$\overline{C} \cup S$）的概率和为1.0。

5.2.3 随机变量

在概率论中，单独的概率要么是通过组合的方法分析计算，要么是凭经验通过对某一事件的总体抽样获得。例如，一个骰子有6个面，一枚硬币有两面。当我们说骰子或者硬币是“公平”的，意味着掷此骰子或硬币所得到的每一个结果都将等可能地出现。因此，很容易就能确定这些随后称之为*参数*的问题情景的事件空间。

然而，更多有趣的概率推理源于对实际事件领域中情形的基于抽样的分析。这些情形往往缺乏一个定义明确的用来支持概率的分析计算的特性。还有一些情形，即使理论上可以进行概率计算分析，但却很复杂，时间和计算成本根本不足以应付概率结果的确定性计算。在这些情形下，通常采用一种基于经验的抽样方法学。

最重要的是，假定实验中的所有结果并非等可能地出现。但是，我们仍然保留在上一节得到的一些基本公理或假定，即事件的概率处于0和1之间（包括0和1），所有结果的概率和为1。同样，还保留计算事件和的概率的公式。为了使演算精确，下面给出随机变量的定义。

定义（随机变量）　**随机变量**是定义在样本空间上的函数，输出结果通常是实数。一个随机变量容许我们用和事件空间相关的数值讨论概率，不再是使用一个特定问题的事件空间。

布尔随机变量是定义在从样本空间到{true, false}或实数子集{0.0, 1.0}上的函数。一个布尔随机变量有时也称为一次**伯努利试验**。

离散随机变量是定义在样本空间上的函数，其中布尔型随机变量可看成是它的一个子集，值域是实数集合[0.0, 1.0]的任一个可数子集。

连续随机变量可在整个实数范围取值。

看一个离散随机变量的例子：整个**季节**域原子事件是{春天, 夏天, 秋天, 冬天}，Season 是定义在这个季节域上的随机变量，给（Season = 春天）赋值为 0.75，在此情形下，Season = 春天的概率即为 0.75。另外，在该季节域上还可定义另一个布尔随机变量 Season，相对应地，（Season = 春天）赋值 true，映射为 Season = 春天的概率为 true（true 即 1）。我们所要考虑的大部分概率实例都属于离散随机变量。

另一个使用布尔随机变量的例子是计算对一个公平硬币扔 7 次，其中 5 次头像朝上的概率。这实际上就等于 7 次扔硬币 5 次头像朝上的组合数乘以每次扔硬币头像朝上的概率（1/2）的 5 次幂再乘上每次扔硬币头像不朝上的概率（1/2）的 2 次幂，即：

$$_7C_5 \times (1/2)^5 \times (1/2)^2$$

该扔硬币的情形就是大家所熟悉的二项式分布的例子。实际上，对于一个 n 次试验，其中 p 是每次试验成功的概率，得到 r 次成功试验的概率可以表示为：

$$_nC_r \times p^r \times (1-p)^{(n-r)}$$

对事件进行相关概率测量的一个重要的自然延伸是结果的预期成本或盈利的概念。例如，翻一张牌或旋转一次轮盘赌轮，可以计算预期能获得的盈利。定义一个随机变量或事件的期望为 ex(E)：

定义（数学期望）　一个事件 E 的概率为 p(E)，它发生所能得到的回报是 r，事件不发生的成本是 c，概率为 1 - p(E)，那么该事件发生的数学期望 ex(E) 为：

$$ex(E) = r \times p(E) + c \times (1 - p(E))$$

举例来说：假设现有一个公平的轮盘赌轮，上面有从 0 到 36 的整数，这些整数在轮的各个槽中等间隔排列。在游戏中，每个游戏者在她选的任一个数字槽上放 1 美元。旋转转轮，如果转轮停在她所选的号码上，她将赢得 35 美元；否则，将输掉 1 美元。赢了有 35 美元的奖励，输了将失去 1 美元，赢的概率为 1/37，输的概率为 36/37，因此该事件的数学期望值 ex(E) 是：

$$ex(E) = 35(1/37) + (-1)(36/37) \approx -0.027$$

我们发现每转一次轮盘，每个游戏者平均都要失去大约 0.03 美元！

在结束本节前，对用于随机推理的概率值的来源进行简要讨论和概述。如上所述，在迄今为止所举的大部分例子中，考虑的都是已知情形下的概率，像扔硬币或转轮盘。一旦了解了这些情形，就可以得出与该情形的概率空间相关的许多结论，如它的平均值，基于统计的概率度量，抽取的样本值与平均值的差通常是如何变化以及此领域下相应随机变量的标准差。

我们把这些易于理解的情形看成是产生样本结果空间的参数方法。事实上，只要在某情形下，能预先获得试验结果的期望，那么参数法就完全适用。我们的任务就是求解此情形所需要的参数。例如，扔硬币得到的结果是服从二项式分布的，然后就可以用结果的二项式模型得出期望。

参数法有许多优点。其一是需要更少的数据点用来校准期望结果，因为结果曲线的形状预先已经知道。其二是，预先确定所有可能结果或者获取作出良好概率估计需要的训练数据往往是可能的。实际上，在参数法中，除了计算期望平均值和标准差以外，还可以对某一个数据点是否超出正常的期望作出精确判断。

然而，许多（如果不是大多数）人们感兴趣的问题并没有明确的期望结果。一个例子是医学上的诊断推理。另一个例子是对一种自然语言表达的使用和解释。对于自然语言的表达推理，通常是从大量的状态中取样，如在一个语言语料库中取样，采用非参数法来实现。通过分析在报纸中或在会话中用到的语言例子，能够推断出这些领域中一些表达不明确的意思。5.3 节将说明分析此可能音素的方法学。

在非参数的情况下，利用足够多的数据点，离散型分布常常可通过插值转化为连续型分布。然后，新的情形可以在该连续型分布的背景下推断。非参数法的一个主要缺点是：由于缺少来自于先验期望的约束，常常需要补偿大量训练数据。在 5.3 节，我们会给出这类推理的例子。

5.2.4 条件概率

到目前为止，所讨论的概率均称为先验概率，因为在获得与事件期望结果相关的新信息之前，它们就已经被计算出来了。本节将考虑一个事件的条件概率，即此事件加上一些新信息或约束后的概率。

正如本章前面看到的，扔一个骰子，所得点数为 2 或 3 的先验概率是两个单独的结果除以所有可能的结果数，即 2/6。一个人染上某一种疾病的先验概率是染病人数除以研究区域的总人数。

举一个条件（后验）概率的例子，假定一个病人来到某医生的办公室，表现出头痛和恶心的症状。该医生非常有经验，事先就知道所有可能导致这些症状的不同疾病的期望，不过，现在他要做的是为病人开出一个具体的诊断。为了使这些概念更精确，给出两个重要的定义。

定义

先验概率

先验概率通常是一个事件的**无条件概率**。它是基于导致该事件发生与不发生的知识下得到的概率，也就是说，是在得到任何新的证据之前计算的事件概率。一个事件的先验概率表示为：p（事件）。

后验概率

后验概率通常是一个事件的**条件概率**。它是在给定一些新证据的情况下得到的事件概率，一个事件的后验概率表示为：p（事件 | 证据）。

接下来开始介绍贝叶斯定理，其一般形式会在 5.3 节中给出。贝叶斯定理的核心思想是：给定一个证据，该证据由某一假设导致的概率，可以看成是由该假设得出此证据的概率函数。用数学式表示为 $p(h \mid e) = f(p(e \mid h))$。通常是通过等式右边的值来确定等式左边的值，因为等式右边的值更容易计算。

现在用一种症状和一种疾病来证明贝叶斯定理。根据上述定义，假设一个人表现出某症状 s（自症状集合 S），那么他染上疾病 d（自疾病集合 D）的后验概率是：

$$p(d|s) = |d \cap s| \ / \ |s|$$

与 5.1 节相同，集合两边的“1”表示集合的势或者集合中包含元素的个数。等式右边是既表现出症状 s 又染上疾病 d 的人数除以表现出症状 s 的总人数。图 5-2 给出了该情况的文氏图。因为

用于确定分子和分母的概率样本空间是相同的，扩展等式的右边得到：

$$p(d \mid s) = p(d \cap s) / p(s)$$

同样对于 $p(s \mid d)$，见图 5-2 也有：

$$p(s \mid d) = p(s \cap d) / p(d)$$

下一步利用 $p(s \mid d)$ 求得 $p(s \cap d)$：

$$p(s \cap d) = p(s \mid d) p(d)$$

将 $p(d \mid s)$ 等式应用到上面的这个等式中，得到一种疾病和一种症状的贝叶斯规则：

$$p(d|s) = \frac{p(s|d)p(d)}{p(s)}$$

因此，已知某一症状，染上相应疾病的后验概率为此疾病导致出现该症状的概率与该疾病的发病率的乘积再除以出现此症状的概率。此公式会在 5.3 节中进行泛化。

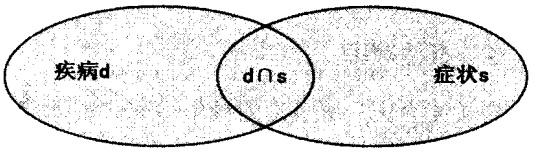


图 5-2　说明 $p(d \mid s)$ 是 $p(s \mid d)$ 的函数的文氏图

下面，给出链式规则，这是一种应用于许多随机推理领域（尤其是自然语言处理领域）的重要方法。刚刚得到的等式（对任两个集合 A_1 和 A_2）：

$$p(A_1 \cap A_2) = p(A_1 \mid A_2) p(A_2) = p(A_2 \mid A_1) p(A_1)$$

现在，对该等式泛化，对于多个集合 A_i，有：

$$p(A_1 \cap A_2 \cap \cdots \cap A_n) = p(A_1)\, p(A_2 \mid A_1)\, p(A_3 \mid A_1 \cap A_2) \cdots p\left(A_n \mid \bigcap_{i=1}^{n-1} A_i\right)$$

该式子称为链式规则。

采用归纳论证法来证明链式规则，考虑 n 个集合的情况：

$$p(A_1 \cap A_2 \cap \cdots \cap A_{n-1} \cap A_n) = p((A_1 \cap A_2 \cap \cdots \cap A_{n-1}) \cap A_n)$$

根据两个集合的交集规则：

$$p((A_1 \cap A_2 \cap \cdots \cap A_{n-1}) \cap A_n) = p(A_1 \cap A_2 \cap \cdots \cap A_{n-1}) p(A_n \mid A_1 \cap A_2 \cap \cdots \cap A_{n-1})$$

可得：

$$p(A_1 \cap A_2 \cap \cdots \cap A_{n-1}) = p((A_1 \cap A_2 \cap \cdots \cap A_{n-2}) \cap A_{n-1})$$

然后考虑 $n-1$ 个集合的情形：如此向下，直到得到 $p(A_1 \cap A_2)$，对于 $p(A_1 \cap A_2)$，我们已经对其进行了证明。

在结束本小节之前，将根据链式规则关系再给出几个定义。首先，在条件概率的背景下重新定义独立事件（见 5.2.1 节）；然后，定义条件独立事件。

定义

独立事件

两个事件 A 和 B 是相互独立的当且仅当 $p(A \cap B) = p(A) p(B)$。*当* $p(B) \neq 0$ *时，可将* $p(A \cap B) = p(A) p(B)$ *写成* $p(A) = p(A \mid B)$。*该式表明，B 是否为真均不影响 A 为真的概率。*

条件独立事件

两个事件 A 和 B 是相互**条件独立的**当且仅当对任意给定的事件 C，有 $p((A \cap B) \mid C) = p(A \mid C)p(B \mid C)$。

由于引入条件独立事件定义，一般形式的链式规则将得到简化，我们就能以相对较小的计算成本构造出更大、更复杂的随机系统。也就是说，条件独立事件简化了联合分布。仍然引用前面提到的交通问题的例子：假定在减速的时候，我们注意到公路旁的橙色控制桶。那么，我们就应能得出，我们减速的原因更有可能是前方道路施工而不是发生了交通事故，并且橙色桶的出现也有自己的概率测度。表示交通减速的变量和表示橙色桶出现的变量是条件独立的，因为两者都起因于道路施工。因此，道路施工变量把交通减速和橙色桶出现两个变量区分开了。

由于条件独立获得的统计效率，所以在构建相对较大的随机计算系统过程中，一个主要任务是把一个复杂问题分成多个弱连接的子问题。这些子问题相互作用的关系是受各种条件分离的相互关系控制的。我们将在 9.3 节进一步分析此概念，并在那里将其形式化定义为 d - 可分。

在 5.3 节，将给出贝叶斯定理的一般形式，并说明在一些复杂的场合，贝叶斯推理需要的计算会非常难处理。5.4 节会给出根据基于贝叶斯概率方法进行推理的例子。

5.3 贝叶斯定理

著名的托马斯·贝叶斯（Thomas Bayes）是一个数学家和牧师。他的著名定理在其死后的第 4 年（1763）出版。一篇名为《用概率的原理解决问题》（Essay towards Solving a Problem in the Doctrine of Chances）的论文在英国《伦敦皇家学会的哲学会刊》（Philosophical Transactions of the Royal Society of London）上发表。贝叶斯定理是如此描述因果关系的：通过理解结果，可以获悉导致它发生的原因的概率。因此，贝叶斯定理是非常重要的，不仅可用于确定导致如癌症等疾病发生的原因，而且能用于判定某个具体的药疗法对疾病的疗效。

概述

概率论的一个最重要的结果是贝叶斯定理的通用形式。首先，回顾一下在 5.2.4 节中得到的一个结果：一种疾病和一种症状的贝叶斯等式。为了能更好地描述诊断的相互关系，我们对前面（5.2.4 节中的诊断例子）用到的变量重命名，h_i 表示某一假设，H 是一组假设，E 代表一组证据。而且认为各个单独的假设 h_i 是互不相关的，所有 h_i 的并集等于 H。

$$p(h_i \mid E) = (p(E \mid h_i) \times p(h_i))/p(E)$$

该等式的意思是：已知一个证据集合 E，确定假设 h_i 的概率。观察此等式：首先，注意到等式右边的分母 $p(E)$，它对假设集 H 中的任一个假设 h_i 来说，非常像一个归一化因数。在某些情况下，给定一组具体的证据 E，常用贝叶斯定理来从一组假设中判定哪个假设是最有可能的。在此情况下，我们还常略去分母 $p(E)$，因为 $p(E)$ 对于所有假设 h_i 来说都是完全相同的，这样能大大节省相应的计算成本。去掉分母后，我们要建立假设的*最大后验值*的概念：

$$\arg\max(h_i)\, p(E \mid h_i)\, p(h_i)$$

我们把该表达式读作所有 h_i 上的 $p(E \mid h_i)p(h_i)$ 的最大值。刚刚描述的化简对于诊断推理及自然语言处理是非常重要的。当然，arg max（也称为最大似然假设）已不再是先前所定义的随机变量。

接下来，考虑对分母 $p(E)$ 的计算。在该情形中，整个样本空间由一组假设 h_i 所划分。集合的划分定义为：把该集合分割成若干个分离的不相重叠的子集，并且所有这些子集的并集是全

集。假定一组假设 h_i 划分了该样本空间得到：

$$p(E) = \sum_i p(E|h_i)\, p(h_i)$$

这个关系并不难证明，只要考虑这样两个事实：利用该组假设 h_i 可形成证据组 E 的一个划分及两集合交集的概率规则。因此：

$$E = (E \cap h_1) \cup (E \cap h_2) \cup \cdots \cup (E \cap h_n)$$

根据 5.2.1 节中介绍的集合并集的一般规则，因为该组假设 h_i 形成了证据组 E 的一个划分（即各 h_i 之间的交集为空集），于是：

$$\begin{aligned} p(E) &= p((E \cap h_1) \cup (E \cap h_2) \cup \cdots \cup (E \cap h_n)) \\ &= p(E \cap h_1) + p(E \cap h_2) + \cdots + p(E \cap h_n) - p(E \cap h_1 \cap E \cap h_2 \cap \cdots \cap h_n) \\ &= p(E \cap h_1) + p(E \cap h_2) + \cdots + p(E \cap h_n) \end{aligned}$$

由对 $p(E)$ 的计算，得到贝叶斯定理的一般形式，其中该组假设 h_i 划分了证据组 E：

$$p(h_i|E) = \frac{p(E|h_i) \times p(h_i)}{\sum_{k=1}^{n} p(E|h_k) \times p(h_k)}$$

$p(h_i \mid E)$：已知证据 E，h_i 的概率，即 h_i 的后验概率

$p(h_i)$：h_i 的先验概率

$p(E \mid h_i)$：当 h_i 为真时，发生证据 E 的概率

n：所有可能假设的个数

对于任意假设，其导致给定证据的概率已知，那么利用贝叶斯定理，就可以计算出该证据是由某一个假设 h_i 导致的概率。

举例来说，假定我们想通过调查某地方的地质迹象，判定此地是否有可能会发现铜。我们必须预先知道发现每种矿物质的概率及每种矿物质发现时显现某一迹象的概率。然后使用贝叶斯定理，通过在该地发现的迹象，断定出在此地发现铜的可能性。此方法应用于地矿勘测系统 PROSPECTOR 中，该系统是由斯坦福国际研究所和斯坦福大学开发的，用于矿产勘探（如：铜、钼等）。PROSPECTOR 曾在不少地方发现了有商业意义的矿床（Duda et al. 1979a）。

下面举一个简单的数字示例来说明贝叶斯定理。假设你要去购买一辆汽车，你去第 1 家经销商 d_1 的概率为 0.2，去第 2 家经销商 d_2 的概率为 0.4，由于你目前只考虑 3 家经销商，所以去第 3 家经销商 d_3 的概率为 0.4。在经销商 d_1 处购买汽车 a_1 的概率为 0.2，在经销商 d_2 处购买汽车 a_1 的概率为 0.4，在经销商 d_3 处购买汽车 a_1 的概率为 0.3。现在已知你已经购买了汽车 a_1，试问你在经销商 d_2 处购买的概率是多少？

首先应知道：要求解上述概率，实际上是计算 $p(d_2 \mid a_1)$。然后，利用此例中涉及的变量，给出用于计算 $p(d_2 \mid a_1)$ 的贝叶斯定理的变量形式。

$$\begin{aligned} p(d_2|a_1) &= (p(a_1|d_2)\, p(d_2)) / (p(a_1|d_1)\, p(d_1) + p(a_1|d_2)\, p(d_2) + p(a_1|d_3)\, p(d_3)) \\ &= (0.4)\,(0.4) / ((0.2)\,(0.2) + (0.4)\,(0.4) + (0.4)\,(0.3)) \\ &= 0.16 / 0.32 \\ &= 0.5 \end{aligned}$$

在使用贝叶斯定理的时候有两个主要的约束：首先，与证据相关的各种假设的概率及各证据之间的概率关系都必须已知。其次，必须估计出或凭经验抽样获得所有证据与假设的关系，即

$p(E \mid h_k)$，这在有些场合是很难做到。回想一下贝叶斯定理的一般形式，在计算 $p(E)$ 时，要求该组假设 h_i 划分了证据组 E。通常，尤其是在像医学和自然语言处理这样的领域中，预先无法假定一个合理的划分。

然而有趣的是，在许多违背此假定（即支持每个假设的证据划分整个证据集合）的情形中，贝叶斯公式仍能很好地适用。使用此划分假定，甚至在那些不符合假定的场合中（使用），被称为是使用简单贝叶斯（naive Bayes）或贝叶斯分类器（Bayes classifier）。关于简单贝叶斯，其假定是：对于任一个假设 h_j：

$$p(E|h_j) \approx \prod_{i=1}^{n} p(e_i| h_j)$$

即对于一个具体给定的假设，我们认为各证据间是相互独立的。

给定一证据组 E，使用贝叶斯定理计算证据组 E 是由某个假设 h_i 导致的概率 $p(h_i \mid E)$，我们常通过获取等式右边的值来计算。因为相比于直接获取等式左边的值（即直接计算概率 $p(h_i \mid E)$ 的值），更容易得到等式右边的值。举例来说，因为患脑膜炎的人数远远少于头痛的人数，所以相比于去计算那些头痛的人中患脑膜炎的百分比，确定脑膜炎病人中有头痛症状的人数要容易得多。需要注意的是，对于简单的单个疾病和单个症状的例子，并不需要计算太多的数。然而，当考虑整个疾病域 H 中的多种疾病 h_i 和所有可能症状集 E 中的多种症状 e_n 时，就麻烦了。当分别考虑 H 中的每种疾病和 E 中的每种症状时，我们将收集和综合 $m \times n$ 个量（实际上是 $m \times n$ 个后验概率加 $m+n$ 个先验概率）。

不幸的是，我们的分析还会变得更复杂。到目前为止，只是单独地考虑每种症状 e_i。在现实情况中，单一症状是很少的。例如，当一个医生在给某病人看病时，通常必须考虑多种综合症状。因此，需要用到贝叶斯定理的一种特殊形式，该形式是在多个症状 e_i 并集的背景下考虑任意单一的假设 h_i。

$$p(h_i \mid e_1 \cup e_2 \cup \cdots \cup e_n) = (p(h_i)p(e_1 \cup e_2 \cup \cdots \cup e_n \mid h_i))/p(e_1 \cup e_2 \cup \cdots \cup e_n)$$

对于一种疾病和单个症状的情形，我们只需要 $m \times n$ 个量。现在，对每一对症状 e_i、e_j 和一个具体的疾病假设 h_i，必须同时知道 $p(e_i \cup e_j \mid h_i)$ 和 $p(e_i \cup e_j)$。如果症状集 E 中有 n 种症状，那么形成的症状对数目为 $n \times (n-1)$，近似等于 n^2。如果要利用贝叶斯进行推理，那会有大约 $m \times n^2$ 个条件概率、n^2 个症状对的先验概率和 m 种疾病的先验概率需要求解，或者通俗地讲，要有 $m \times n^2 + n^2 + m$ 条信息待收集。在一个现实的医疗系统中，它内含 200 种疾病和 2000 种症状，那么这意味着需要收集的信息数将超过 800 000 000 条！

然而，我们可以进行某些简化。正如在介绍条件无关性时讨论的，许多症状对是独立的，也就是说 $p(e_i \mid e_j) = p(e_i)$。当然，这意味着 e_i 的概率是不受 e_j 影响的。例如，在医学上，大多数症状是毫不相关的，比如掉头发和肘痛。但尽管如此，对于此医疗系统来说，即使其中只有不过 1/10 的症状不是独立的，那还会有大约 80 000 000 条的信息留待考虑。

在许多诊断的情形中，也必须处理一些否定的信息，例如，当该病人没有某一症状时，如不良血压。需要求出：

$$p(\bar{e}_i) = 1 - p(e_i) \text{ 和 } p(\bar{h}_i \mid e_i) = 1 - p(h_i \mid e_i)$$

还应注意到 $p(e_i \mid h_i)$ 和 $p(h_i \mid e_i)$ 并不是一样的而且它们几乎总有不同值。这些相互关系及避免循环推理对贝叶斯信念网的设计非常重要。贝叶斯信念网的设计会在 9.3.1 节中介绍。

此外，还有一个决定性的问题使维护复杂的贝叶斯系统几乎是难以办到的：当假设和证据

之间的某些新相互关系得以发现时，系统中的概率表需要重建。而在许多活跃的研究领域（如医学），新的发现会连续不断地产生。因此，如果要使贝叶斯推理得到的结论保持正确性，就要不断地更新概率表以使系统有完全的、最新的概率，包括联合概率。不过在许多领域，这样广泛的数据收集和核对是不可能的，或者即使可能，代价也太高。

无论如何，一旦贝叶斯定理中的假定得以满足，那么贝叶斯就能在数学上很好地对不确定性进行完善的处理。然而，大部分的专家系统领域并不满足这些必要条件而必须依赖启发式方法，该知识将会在第 8 章中介绍。由于复杂度问题，我们知道即使对于十分强大的计算机系统，它也无法胜任使用完全的贝叶斯技术来进行成功的实时问题求解。我们举两个例子来结束本节，这两个例子说明了如何利用贝叶斯分析法来组织假设/证据关系。但是我们需要先定义概率有限状态自动机/接受器。

5.4　随机方法学的应用

你说［t ow m ey t ow］，我说［t ow m aa t ow］……

——IRA GERSHWIN，“让我们把整个事情都取消”

在本节，我们给出几个使用概率测度来推理解释模糊信息的例子。首先，根据 3.1 节中的有限状态自动机来定义一个重要的建模工具——概率有限状态自动机。

定义

概率有限状态自动机

概率有限状态自动机是一个有限状态自动机，其下一个状态转换函数是在该自动机所有状态全集上的概率分布。

概率有限状态接受器

概率有限状态接受器是一个**接受器**，既有一个或多个**起始**状态又有一个或多个**接收**状态。

由此可见，这两个定义是对 3.1 节中提出的有限状态自动机和摩尔机的简单扩展。扩展主要是针对不确定性，它也使下一个状态转换函数不再是一个严格意义上的函数。也就是说，对于问题域中的每个输入值和状态，没有惟一的转换状态。更确切地说就是对于任意状态，其下一个状态的函数都是在其下一所有可能状态上的一个概率分布。

5.4.1　“tomato”是如何发音的

图 5-3 给出一个概率有限状态接受器，能表示“tomato”这个单词的不同发音。图中从起始状态到接受状态的一条路径表示单词 tomato 的一个可接受发音。弧上的数值代表说话者使状态机做出转换的概率。例如：在所有统计的说话者当中，60% 的人在念 tomato 的时候直接从 t 音到 m 音，而不会在这两个音之间发出其他的音。

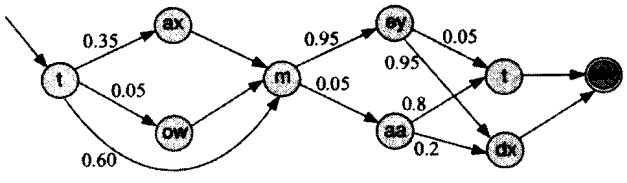


图 5-3　用于单词“tomato”发音的概率有限状态接受器，此图改编自（Jurafsky and Martin 2009）

除了能表征数据库中的人们对单词 tomato 的各种发音方式以外，该模型还可用于帮助解释某

些模糊的音素集。通过观察这些音素与某词及相关词的状态机中的路径的匹配程度，我们就能实现。此外，对于一个部分形成的词，该状态机努力确定能最好地补全该词的概率路径。

在接下来的例子中，我们考虑*音素识别*问题，常称为*解码*。此例子是从 Jurafsky 和 Martin 在 2009 年出版的一本书中摘录的。假定一个音素识别算法已经识别出音 ni（例如在单词“knee”中有该音），并且在此之前已识别出词（音）I，现在我们想把该 ni 音与某个词或某词的开头关联起来。在这种情况下，可以用语言语料库（Brown 和 Switchboard 语料库）来帮助我们。

Brown *语料库*收集有上百万的词，这些词来自于报纸、小说、学术论文等 500 篇文章，它是在 20 世纪 60 年代由 Brown 大学收集的（Kucera and Francis 1967，Francis 1979）。Switchboard *语料库*收集有 140 万在语音交谈中用到的词。这两个语料库一共包含大约 2 500 000 个词，使我们可以从读和写的信息库中抽样。

有很多方式可用于识别与 ni 音相关的最有可能的词。首先，我们可以确定以 ni 音开头的最常使用的词。表 5-2 给出这些词的初始使用频数和它们发生的概率，即该词的初始频数除以 Brown 和 Switchboard 语料库中的总词数。（此表同样来自于 Jurafsky 和 Martin 在 2009 年出版的那本书中）。由表中的数据可见，词“the”的使用频数最大，因此看起来它是匹配音 ni 的第一选择。

表 5-2 从具有 2 500 000 个词的 Brown 和 Switchboard 语料库中统计的含有 ni 音的单词频数和概率，改编自 Jurafsky 和 Martin（2009）

单词	频数	概率
knee	61	0.000 024
the	114 834	0.046
neat	338	0.000 13
need	1417	0.000 56
new	2625	0.001

接下来应用上一节提出的贝叶斯定理的一种形式来推理。我们再次使用该公式的一个简化形式，分析在词 I 后的那个与 ni 音相关的可能词。该简化形式忽略了分母：

$$p(\text{word} \mid [\text{ni}]) \propto p([\text{ni}] \mid \text{word}) \times p(\text{word})$$

表 5-3 中的计算结果是按照推荐的优先顺序从大到小排列。通过该表，我们得出词 the 中不可能含有发音 ni，而且可以判断与音 ni 相关的最有可能的词是 new。但是这两个词的组合（I new）看起来没有多大意义，而其他的组合，比如 I need 这两词组合更有意义。此情形下的一部分问题是我们仍然在音级上进行推理，也就是说，仅仅通过计算出概率 p(ni | new) 的值最大而判断最有可能的词是 new。实际上，有一种解决此问题的简单方法，就是在语料库中寻找明显的两个词的组合。遵循这个推理原则，我们就能得出 I need 是要比 I new 更有可能的连续字对。同样道理，I need 组合也要比词 I 和其他词（如 neat、knee）的组合更有可能。

表 5-3 来源于 Brown 和 Switchboard 语料库的单词中含有 ni 发音的概率，改编自 Jurafsky 和 Martin（2009）

word	p([ni] | word)	p(word)	p([ni] | word) x p(word)
new	0.36	0.001	0.000 36
neat	0.52	0.000 13	0.000 068
need	0.11	0.000 56	0.000 062
knee	1.0	0.000 024	0.000 024
the	0.0	0.046	0.0

从语料库的两词或三词组合中导出概率的方法学称为 n 连词分析（n-gram analysis）。对于两个词的组合，称为二连词分析（bigrams），三个词的则称为三连词分析（trigrams）。通过使用 n 连词分析而得到的单词组合概率是很重要的，这点在第 15 章会再次看到。

5.4.2　道路/交通例子的扩展

我们再次给出 5.2.2 节中的例子并对其进行扩展。假如你在州际公路上驾驶，因为交通拥挤你意识到在逐渐地减速。你开始寻找导致减速的可能解释：莫非是前方道路施工？或者发生了交通事故？或许你还有其他的可能解释。不过几分钟后，你发现公路旁摆放有橙色桶开始切断外车道的交通。此时，你能断定导致交通缓慢的最有可能的解释是道路施工。同时你认为，发生交通事故的可选假设基本可以排除。类似地，如果你已经在前方远处看到有闪光灯，就像是从警车或救护车上发出来的，那么在得到这个新的证据后，我们能判断出导致交通缓慢的最可能解释是发生了交通事故，而作为道路施工的假设则基本可以排除。然而，我们说某假设是基本可排除的，并不意味着该假设就完全不可能。更确切些地说，我们只能说在发现新证据的背景下，此假设的可能性减少。

图 5-4 就我们刚刚所说的给出了一个贝叶斯叙述。道路施工与橙色桶和交通差是有关的。同样，交通事故与闪光灯和交通差是相关的。通过分析图 5-4，我们构造出道路施工和交通差的联合概率分布。首先简化这两个变量，其取值为真（t）或假（f）。这样，我们就得到两个变量的概率分布表，如表 5-4。注意到如果变量 C（道路施工）为 f，那么出现交通差的可能性相对较小；若 C 是 t，则交通差的可能性就较大。同时注意到，在该州际公路上施工的概率（即 C = true）为 0.5，出现交通差的概率（即 T = true）为 0.4（这与新墨西哥州的道路状况很相似！）。

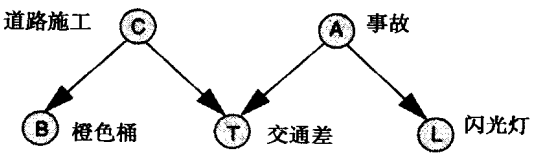


图 5-4　对交通问题的可能解释的贝叶斯表示

表 5-4　图 5-3 中交通差和道路施工这两个随机变量的联合概率分布

	C	T	p	
C is true = 0.5	t	t	0.3	T is true = 0.4
	t	f	0.2	
	f	t	0.1	
	f	f	0.4	

我们接下来考虑：已知现在的交通很差，那么该情形是由于道路施工所导致的概率，即 p(C|T) 或 p(C = t|T = t)。

$$p(C \mid T) = p(C=t, T=t) / (p(C=t, T=t) + p(C=f, T=t)) = 0.3/(0.3+0.1) = 0.75$$

该公路上出现施工的先验概率是 0.5，如果在已知道路上发生了交通拥挤的情形下，道路施工的概率将上升为 0.75。而且由于橙色桶的出现，可以基本排除发生交通事故的假设，那么此概率将进一步地升高。

在对某具体问题应用贝叶斯公式推理的时候，我们应对其状态中的任何一个参数都有所了解。除了这个必要条件外，还必须处理 5.4.1 节提到的复杂度问题。考虑图 5-4 中所有参数的联合概率的计算（使用链式规则和变量的拓扑排序次序）：

$$P(C, A, B, T, L) = P(C) \times P(A \mid C) \times P(B \mid C, A) \times P(T \mid C, A, B) \times P(L \mid C, A, B, T)$$

此结果是对概率测度的一种通用的始终为真的分解。其中，构造一个联合概率表的代价是与其涉及的参数数目成指数比的。在本情况中，我们需要一个大小为32（2^5）的表。当然，对于只有5个参数的问题，这是非常简单的。但是在一个令人感兴趣的情形中，它所涉及的参数常常至少有30个，这也意味着此情形下的联合分布表中将有大约十亿个元素！不过，贝叶斯信念网和d-可分将为我们提供进一步处理表现复杂性和计算复杂性的工具。这些知识会在9.3节中看到。

5.5 结语和参考文献

机会对策的产生至少可以回溯至希腊和罗马文明时期。而概率论的数学分析直到欧洲文艺复兴时期才开始。正如本章所介绍的，概率推理起始于计数与组合学的确定原则。事实上，第一个组合的机器来自一个叫Ramon Llull（Ford et al. 1995）的西班牙哲学家和圣芳济会的修道士，他制造了一个可以自动历数神明的品性的装置，用来改造异教徒。第一部概率出版物《De Ratiociniis Ludo Aleae》是由基督徒惠更斯（1657）完成的。惠更斯总结了早期的Blaise Pascal用于计算概率和条件概率的方法。Pascal不仅对博弈世界进行了“客观”分析，而且对信念系统像上帝的存在等问题进行了“主观”分析。

5.2.1节中介绍的概率的定义是基于法国数学家皮尔西蒙·拉普拉斯提出的形式论的。拉普拉斯在其《Theorie Analytique des Probabilitiees》（1816）一书中给出了这一方法。拉普拉斯的研究是以早期的Gotlob Leibnitz与James Bernoulli发表的一些结果为基础而开始的。

托马斯·贝叶斯是一位数学家和牧师。他的著名定理在他死后的1764年问世。他的一篇名为《用概率的原理解决问题》的论文在英国《伦敦皇家学会哲学会刊》上出版。不过具有讽刺意味的是，尽管贝叶斯定理是客观存在的，但在这篇论文中却从未被明确地指出。此外，贝叶斯还对统计度量的“现实性”有过广泛的讨论。

贝叶斯的研究在一定程度上是由对苏格兰哲学家David Hume提出的哲学上的怀疑主义的回答所推动的。休姆的不考虑因果性的做法，破坏了支持基于神迹的上帝存在理由的所有基础。事实上，在给英国皇家学会的一篇1763年的论文中，牧师Richard Price运用贝叶斯定理说明有许多充分的证据可用来支持解释在新约圣经中描述的神迹，以驳斥那些怀疑论者。

20世纪早期的数学家，包括Fisher（1922）、Popper（1959）和Carnap（1948），延续了对概率本质的主观/客观争论，并在此基础上完成了现代概率论的基础。Kolmogorov（1950，1965）对概率推理的基础进行了公理化（见5.2.1节）。

可以从几个有利方面来介绍概率推理论题。目前两个最流行的方法是以命题演算和集合论为基础。对于命题演算（见2.1节），我们在[0.0，1.0]的范围为命题分配一个置信度，该置信度又被称为概率真值。这种方法为命题演算的语义提供了一个自然扩展。然而，我们更愿意选择第二种方法作为研究概率推理的方法，因为我们觉得该方法更为直观，只要看一下5.1节中介绍的有关集合论的计数方法和其他技术即能得出。在第9章和第13章，我们将随机系统的表示扩展到一阶的（基于变量的）表示机制。

贝叶斯定理已经为20世纪70年代和80年代的几个专家系统提供了基础，包括像在格拉斯哥大学医院对急性腹痛所做的广泛分析（de Dombal et al. 1974）和来自斯坦福大学开发的矿产勘探专家系统——PROSPECTOR（Duda et al. 1979a）。简单的贝叶斯方法已经用在了许多分类问题上，像模式识别（Duda and Hart 1973）、自然语言处理（Mooney 1996）和其他领域。对于那些不满足贝叶斯定理的独立性假定的情形，Domingos和Pazzani（1997）也为成功使用简单贝叶斯分类器提供了依据。

在人工智能领域，现在有大量的概率推理研究，其中有些将会在第9、13和16章介绍。不确定推理现已是AAAI、IJCAL、NIPS和UAI等人工智能会议上的一个重要部分。在概率推理方面有一些优秀的介绍性的文章（Ross 1988，DeGroot 1989），而在人工智能应用中随机方法的使用也有若干介绍文章（Russell and Norvig 2003，Jurafsky and Martin 2009，Manning and Schutze 1999）。

Sandia国家实验室的Dan Pless为我们这个章节提供了通用的方法、部分例子和一些编辑上的建议，对此我们非常感激。

5.6 习题

1. 将一个有6个面的公平骰子扔5次，5次所得的5个骰子点数构成一个序列，试问可能有多少种不同的序列？
2. 对单词MISSISSIPPI中的字母进行排列，试问有多少种不同的排列，单词ASSOCIATIVE又有多少种？
3. 假如一个缸中包含15个球，其中8个是红色，7个是黑色。从中取出5个球：
   a）所有5个球都是红色，试问有多少种取法？所有5个球都是黑色呢？
   b）其中2个球是红色另3个是黑色，有多少种取法？
   c）至少有2个球是黑色的取法？
4. 从5个教职员和7个学生中选出3名教职员和两名学生来组成一个委员会，试问有多少种选法？
5. 在一份对250名电视观众的调查中，88个人喜欢看新闻，98人爱看体育比赛，而其他94个人喜欢看喜剧。并且有33人喜欢看新闻和体育比赛，31人喜欢看体育比赛和喜剧，35个人喜欢看新闻和喜剧。另外，有10个人对新闻、体育比赛和喜剧都喜欢看。现在从这些人中随机地抽取出一个人：
   a）此人喜欢看新闻但不爱看体育比赛的概率是多少？
   b）此人爱看新闻或体育比赛但不喜欢看喜剧的概率是多少？
   c）此人既不喜欢看新闻又不喜欢看体育比赛的概率是多少？
6. 一个四位十进制整数，最高位不能为0，试问：
   a）该整数有一位是数3，5或7的概率是多少？
   b）该整数最高位是3且最低位为5或其中有一位是数7的概率是多少？
7. 扔两个骰子求：
   a）所得点数和为4的概率。
   b）所得点数和为7或偶数的概率。
   c）所得点数和大于等于10的概率。
8. 从52张牌中取一张牌求：
   a）此牌有人物脸（即是J、Q、K或A）的概率。
   b）此牌是Q或是黑桃的概率。
   c）此牌有人物脸或是梅花的概率。
9. 当你在玩5张牌的游戏时，求：
   a）出现同花牌的概率（即5张牌要么都是红桃，要么都是黑桃，或者都是梅花，或者都是方块）。
   b）5张牌中，其中3张牌的大小相同且另外2张的大小也相同的概率。
   c）出现同花大顺的概率（即5张牌不仅要么都是红桃，要么都是黑桃，或者都是梅花，或者都是方块，而且5张牌的大小恰巧就是10，J，Q，K，A）。
10. 求一个随机变量的期望，实际上就是求该随机变量的均值。例如：对一个骰子扔很多次，统计我们想要的结果的次数，将统计得到的次数除以扔骰子的总次数，所得商即为结果。
    a）扔一个骰子，所得点数的期望是多少？
    b）转动一个有37（0～36）个数的轮盘赌轮，所得数的期望是多少？
    c）从一副牌中取牌，所得牌的面值的期望是多少（其中认为A的面值为1，其他有人物脸的牌的牌值

为10)?

11. 假设我们在玩一个游戏：扔一个骰子，然后我们就会得到与所投的点数相应的美元。例如，如果扔到3，那么我们将获得3美元。但是，在每次扔骰子之前，我们需要先交出4美元，试问你是否能从该游戏中获利?
12. 考虑一个随机产生的四位二进制串。说明这两个情形：该位串含有偶数个1和该位串以1结束，它们是否相互独立?
13. 已知 $p(A \mid B, C) = p(A \mid C)$ 和 $p(B \mid A, C) = p(B \mid C)$，证明 $p(A, B \mid C) = p(A \mid C)\,p(B \mid C)$。
14. 在制造一个产品的过程中，生产的产品有85%是没缺陷的。现对该产品进行审查，其中有10%的好产品会检测为有缺陷而不能出厂，同时会有5%的有缺陷的产品未被检测出而准予出厂。现在已知该产品准予出厂，试问它是有缺陷的概率?
15. 一次验血对检测某一疾病是90%的有效，而且它检测错（即此人是健康的但却诊断出患该病）的概率为3%。如果验血的人中有10%的人患有此病，试问当某一个人检测出血样呈阳性，那么他真的患有此病的概率是多少?
16. 假设一个汽车保险公司将驾驶员分成好、一般或差三类。在所有该保险公司投保的驾驶员中，25%可归为好一类，50%归为一般，而剩下的25%则归为差的一类。假定这一年中，一名好的驾驶员有5%的机会会出事故，一般的驾驶员有15%的机会会出事故，而一名差的驾驶员将有25%的机会会出事故。现在，如果你在这一年发生过事故，那么试问你是一名好的驾驶员的概率?
17. 有三名囚犯A、B和C，他们被告知明天会有一个人被处决而另两名囚犯会得到赦免，并且只有州长知道谁会被处决。囚犯A于是向门卫寻求帮助，他让门卫去问问州长到底谁会被处决，然后让他告诉囚犯B或C他会被赦免。门卫照他的话做了，然后他回来跟囚犯A说他已经告诉囚犯B他（B）将被赦免。试问囚犯A将被处决的概率?该信息对囚犯A有用吗?和向门卫求助前相比，囚犯A是否得到更多的信息呢?

    注：此问题摘自Pearl（1988）。

第6章　为状态空间搜索建立控制算法

如果我们仔细地把任务环境的影响从底层的硬件部件和构造的影响中分离出来，那么适应系统的真正的简单性就会呈现出来。因为，正如我们看到的，我们仅需要假定一种非常简单的信息处理系统，就可以说明人类求解诸如国际象棋、逻辑和加密算法等任务的方法。信息处理系统在具体环境中反映出的表面复杂性是由多个因素相互作用而导致的，比如系统中少数几个基本参数（特别是其内存的特性）与环境需求间的矛盾。

——A. 纽厄维尔和 H. A. 西蒙，《人类问题求解》（Newell and Simon，1972）

我们称其为开始的经常就是结束
而结束又是新的开始。
因为我们常是在终点起步……

——托马斯·斯特恩斯·埃略特，《四首四重奏》

6.0　简介

到此为止，第二部分已经把问题求解表示为对问题情景或状态集合的搜索。第2章介绍了谓词演算，谓词演算是描述问题状态的中间媒介，也是产生新状态的可靠推理规则的手段。第3章介绍了图，用来表示和连结问题状态。回溯算法以及深度优先、广度优先等算法可用来进行这些图的搜索。第4章介绍了启发式搜索算法。第5章给出了世界的可能性状态，并用随机推理来生成新状态。总结一下，第二部分介绍了以下内容：

1）把问题的解表示为从起始状态到目标的一条路径。

2）通过使用搜索，系统地检验通往目标的候选路径。

3）回溯或其他某种机制使算法可以从无法找到目标的路径恢复。

4）使用列表显式记录纳入考虑的状态。

a）open 列表使算法可以在必要时探索还未试验过的状态。

b）记录已访问过的状态的 closed 列表使算法可检测循环，避免重复没有希望的路径。

5）open 列表在深度优先搜索中实现为堆栈（stack），在宽度优先搜索中实现为队列（queue），在最佳优先搜索中实现为优先级队列（priority queue）。

本章介绍实现搜索算法的更多技术。在6.1节将介绍*递归搜索*，用一种比第3章中更简洁自然的方式来实现深度、广度和最佳优先搜索。同时，将进一步介绍通过使用合一（unification）的递归，来搜索谓词演算断言产生的状态空间，并进一步探讨递归搜索。这种模式导向（pattern-directed）的搜索算法是 Prolog（见14.3节）和第8章要讨论的许多专家系统的基础。接下来在6.2节中将讨论产生式系统（production system），一种用在模式导向问题求解方法中的通用体系结构，它的应用已经非常广泛，例如已广泛用于为人类问题求解方法建模（第16章）以及其他很多 AI 应用，包括专家系统（第7章）。最后，在6.3节中，我们将给出介绍另一种 AI 问题求解表示方法控制结构——黑板（blackboard）。

6.1 基于递归的搜索（选读）

6.1.1 递归

在数学中，递归定义是指在定义中使用了其自身定义的一部分。在计算机科学中，可以用递归来定义和分析数据结构和过程。一个递归过程由以下两部分组成：

1）递归步骤：过程调用自身来重复一系列动作。

2）使递归过程从无限递归（无穷循环的递归）中停止的终止条件。

这两部分都是非常关键的，所有递归定义和算法都应该包含这两部分。递归可以非常自然地控制那些具有规则结构而又不限定大小的数据结构，比如列表、树和图，因此递归特别适用于状态空间搜索。

可以把第3章中的深度优先搜索算法直接翻译成递归形式，这说明了递归和迭代的等价性。下面这个算法使用全局变量 closed 和 open 来维护状态列表：

```
function depthsearch;                                    % open & closed global

  begin
    if open is empty
      then return FAIL;
    current_state := the first element of open;
    if current_state is a goal state
      then return SUCCESS
      else
        begin
          open := the tail of open;
          closed := closed with current_state added;
          for each child of current_state
            if not on closed or open                          % build stack
                then add the child to the front of open
        end;
    depthsearch                                                % recur
  end.
```

可以用几乎完全相同的算法来设计宽度优先搜索和最佳优先搜索，即可以把 closed 维护为一个全局数据结构，把 open 列表实现为一个队列而不是堆栈。

刚才给出的深度优先搜索并没有发挥出递归的全部威力。还可以进一步简化这个过程，方法是使用递归本身（而不是显式的 open 列表）来组织状态空间中的状态和路径。在以下这个版本的算法中，我们使用全局 closed 列表来探测重复状态并防止循环，而 open 列表则隐含在递归环境的激活记录中。既然不能对 open 列表进行显式的操作，广度优先搜索和最佳优先搜索就不再是下列算法的扩展算法。

```
function depthsearch (current_state);                    % closed is global

begin
  if current_state is a goal
    then return SUCCESS;
  add current_state to closed;
  while current_state has unexamined children
    begin
```

```
    child := next unexamined child;
    if child not member of closed
      then if depthsearch(child) = SUCCESS
        then return SUCCESS
  end;
  return FAIL                                    % search exhausted
end
```

这种算法不再把一个状态的所有孩子都放到一个 open 列表中，而是一次产生一个孩子状态并在产生这个孩子的兄弟状态之前递归搜索它的后继。注意：这个算法假定状态产生过程是按一定顺序的。在递归搜索一个孩子状态时，如果这个状态的某个后继是目标，那么递归调用返回成功，同时算法忽略兄弟结点。如果对孩子状态的递归调用没有找到目标，那么就产生下一个兄弟并搜索它的所有后继。通过这种方式，算法按深度优先顺序搜索整个图。读者应该验证一下，它对图的搜索顺序与 3.2.3 节中深度优先搜索算法的顺序是否相同。

省略显式的 open 列表是通过递归实现的。在程序设计语言实现递归的机制中为每次递归调用建立了一个*激活记录*（activation record）（Aho and Ullman 1977）。每个激活记录存储了每次过程调用的执行状态和局部变量。当以一个新的状态递归调用这个过程时，就会产生一个新的激活记录把过程的参数（这个状态）、所有局部变量和执行的当前状态存储起来。在递归搜索算法中，当前路径的状态序列记录在递归调用的激活记录序列中。每一次调用的记录还包含用来产生孩子状态的最后一个操作；这允许必要时可以产生下一个兄弟。

当一个状态的所有后继都没有包括目标，导致这次递归调用失败时，回溯便会产生作用。算法向展开双亲结点的过程返回 fail，后者生成下一个兄弟状态并对其进行递归。在这种情况下，递归的内部机制承担了算法迭代版本中 open 列表所做的工作。递归实现允许编程人员把他们的注意力集中在单一的状态和其孩子上，而不必显式地维护状态的 open 列表。递归以闭合形式表示全局概念的能力是其强大性的一个主要体现。

以上两种算法表明状态空间搜索在本质上是一种递归过程。为了发现从当前状态到达目标的路径，我们可以移动到一个孩子状态并递归。如果这个孩子状态没有产生目标，那么就按顺序试验它的兄弟。递归把一个庞大的难题（搜索整个空间）分解成了多个更小、更简单的问题（产生一个单一状态的孩子）并对它们中的每一个小部分（递归）应用了这种策略（递归）。然后，继续这个过程一直持续直到发现目标或遍历了整个空间。

在下一节中，我们将把这种通过递归求解问题的方法扩展到一个用于控制基于逻辑的问题求解器的控制部分方法的程序中，这个基于逻辑的问题求解方法使用合一和推理来产生并搜索逻辑关系空间。这个算法支持多个目标的“与”以及从目标反向追索前提。

6.1.2　一个递归搜索的例子：模式驱动推理

在本节中，我们把递归搜索应用到由逻辑推理所组成的空间中；结果得到一种适用于基于谓词演算的问题描述的通用搜索过程。

假定我们想要编写一个算法判断一个谓词演算表达式是否为某个断言集合的逻辑结论。这使我们想起目标导向搜索：初始查询组成了目标，假言推理定义了状态之间的转换。对于一个给定目标（比如 p(a)），算法可以使用合一来选择结论与目标匹配的蕴涵（比如 q(X)→p(X)）。因为这个算法把蕴涵当作求解查询的可能规则，所以经常把蕴涵简称为规则。把目标和蕴涵（也就是规则）的结论合一并把得到的置换应用到整个规则后，这个规则的前提变成了新的目标（q(a)）——称其为*子目标*。然后算法递归求解这个子目标。如果一个子目标与知识库中的事实相匹配，那么搜索便终止，从初始目标到给定事实的推理序列证明了原始目标的真实性。

```
function pattern_search (current_goal);

begin
  if current_goal is a member of closed                    % test for loops
    then return FAIL
    else add current_goal to closed;
  while there remain in data base unifying facts or rules do
    begin
      case
        current_goal unifies with a fact:
          return SUCCESS;
        current_goal is a conjunction (p ∧ ...):
          begin
            for each conjunct do
              call pattern_search on conjunct;
            if pattern_search succeeds for all conjuncts
              then return SUCCESS
              else return FAIL
          end;
        current_goal unifies with rule conclusion (p in q → p):
          begin
            apply goal unifying substitutions to premise (q);
            call pattern_search on premise;
            if pattern_search succeeds
              then return SUCCESS
              else return FAIL
          end;
      end;                                                 % end case
    end;
    return FAIL
  end.
```

在 pattern_search 函数中，搜索是按 2.3.2 节中递归搜索算法的一个修改版本进行的，算法使用合一来判断两个表达式是否匹配，并使用假言推理来产生状态的孩子。搜索的当前焦点表示为变量 current_goal。如果 current_goal 和事实匹配，那么算法返回 success。否则算法试图把 current_goal 和某个规则的结论相匹配，递归地尝试求解规则的前提。如果 current_goal 没有与任何给定的断言匹配，那么算法返回 fail。这个算法也可以处理用合取表示的目标，因为合取经常出现在规则的前提中。

为简单起见，这个算法没有考虑维护变量一致性的问题，即合一产生的各个变量置换间的一致性。这在用共享变量求解合取查询时（比如在 p(X) ∧ q(X) 中）是很重要的。不但两个合取项要成功，而且它们必须是在对 X 的可合一绑定上获得成功（见 2.3.2 节）。

使用合一和假言推理这些通用方法的主要优势是得到的算法可以搜索任何由逻辑推断所组成的空间，其中对问题的指定性说明可以用谓词演算的断言来描述。这样，我们便可以用这种手段把求解问题的知识从它在计算机上的控制和实现中分离出来。pattern_search 算法为我们提供了一种方法，首次实现了这种知识和控制的分离。

尽管搜索算法（pattern_search）的初始版本定义了用于谓词演算表达式的搜索算法的行为，但是有几个细微之处还有待完善。这些细节包括算法以什么样的顺序试验候选匹配以及如何恰当地处理整套逻辑操作符（∧、∨和¬）。逻辑是说明性的，并没有规定搜索策略：它定义了一个由可能推理所组成的空间，但没有说明问题求解器如何做出有用的推理。

为了使用谓词演算进行推理，需要一种控制策略来系统地搜索空间，以避免分析毫无意义

的路径和陷入循环。像 pattern_search 这样的控制算法必须按某种顺序来试验候选匹配。知道了这个顺序，程序设计者就可以通过对知识库中规则的适当排序来控制搜索。定义这个顺序的一个简单方式是要求算法按规则和事实在知识库中出现的顺序试验它们。

第二个问题是在规则的前提中存在逻辑连接词：例如，以下形式的蕴涵："p←q∧r"或"p←q∨(r∧s)"。和与或图的情况一样，合取操作符表明要使这个前提为真，则必须证明两个合取项都为真。此外，还必须用一致的变量绑定来求解表达式的合取。这就是说，要求解 p(X)∧q(X)，用置换{a/X}解出 p(X)，用置换{b/X}解出 q(X)是不够的。两部分都必须是使用同一个可以合一的 X 绑定来求解的。另一方面，或操作符表明只要发现两个表达式之一为真就可以了。搜索算法必须考虑这一点。

对算法的最后一个补充是使它能够求解含有逻辑非（¬）操作符的目标。pattern_search 通过最先求解非的操作数来处理含有非的目标。如果这个子目标成立，那么 pattern_search 返回 fail。如果这个子目标不成立，那么 pattern_search 返回一个空的置换集，表明成功。注意：即使子目标中可能包含变量，求其非的结果中也不能包含任何置换。这是因为只有非（¬）的操作数不成立时才成立，所以它不可能返回任何对操作数的绑定。

最后，这个算法不该返回 success，而该返回解中所包含的绑定。下面是 pattern_search 的完整版本，返回满足每个子目标的合一置换集合。

```
function pattern_search(current_goal);

begin
  if current_goal is a member of closed                    % test for loops
    then return FAIL
    else add current_goal to closed;
  while there remain unifying facts or rules do
begin
  case

    current_goal unifies with a fact:
      return unifying substitutions;
    current_goal is negated (¬ p):
      begin
        call pattern_search on p;
        if pattern_search returns FAIL
          then return {};                                  % negation is true
          else return FAIL;
      end;

    current_goal is a conjunction (p ∧ ...):
      begin
        for each conjunct do
          begin
            call pattern_search on conjunct;
            if pattern_search returns FAIL
              then return FAIL;
              else apply substitutions to other conjuncts;
          end;
        if pattern_search returns SUCCESS for all conjuncts
          then return composition of unifications;
          else return FAIL;
      end;
```

```
        current_goal is a disjunction (p ∨ ...):
          begin
            repeat for each disjunct
              call pattern_search on disjunct
            until no more disjuncts or SUCCESS;
            if pattern_search returns SUCCESS
              then return substitutions
              else return FAIL;
          end;

        current_goal unifies with rule conclusion (p in p ← q):
          begin
            apply goal unifying substitutions to premise (q);
            call pattern_search on premise;
            if pattern_search returns SUCCESS
              then return composition of p and q substitutions
              else return FAIL;
          end;
      end;                                                    %end case
    end                                                       %end while
    return FAIL
  end.
```

这个用于搜索谓词演算规则和事实的算法 pattern_search 是 Prolog 的基础（在 Prolog 中使用谓词的 Horn 子句形式，见 14.3 节），而且已经应用于很多目标导向的专家系统外壳中（见第 8 章）。另一种控制目标导向搜索的结构是产生式系统，这部分内容将在下一节中讨论。

6.2 产生式系统

6.2.1 定义和历史

产生式系统（production system）是 AI 中一种特别重要的计算模型，它不仅用来实现搜索算法，而且用来对人类的问题求解方法建模。产生式系统为控制问题求解过程提供了一种面向模式的手段。一个产生式系统由以下三部分构成：产生式规则集合、工作内存和识别 - 动作控制循环。

定义（产生式系统） 一个**产生式系统**由以下三部分定义：

1）**产生式规则集合**（the set of production rules）。这些规则经常简称为**产生式**（production）。一个产生式就是一个**条件 - 动作**对，定义了求解问题的一个知识块。规则的**条件部分**是一种模式，用来确定何时可以把这个规则应用到问题实例。规则的**动作部分**定义了相关联的求解问题步骤。

2）**工作内存**（working memory）包含了推理过程中对**世界当前状态**的描述。这个描述就是要和产生式的条件部分匹配的模式，以选择合适的问题求解动作。当一个规则的条件要素和工作内存的内容相匹配时，便可以执行和这个条件相关联的动作。设计产生式规则的动作时也特意将其设计为用来改变工作内存的内容。

3）**识别 - 动作循环**（the recognize - act cycle）。产生式系统的控制结构是很简单的：先使用问题描述初始化**工作内存**，并把求解问题的当前状态维护为工作内存中的模式集合。然后把这些模式与产生式规则的条件进行匹配；条件和工作内存中的模式相匹配的规则形成一个子集，称为**冲突集合**（conflict set）。冲突集合中的产生式称为是**使能的**。然后在冲突集合中选择一个产

生式（**冲突消解**（conflict resolution）），并**激发**这个产生式。激发一个规则就是执行它的**动作**，修改工作内存的内容。在选择的产生式规则激发之后，对修改后的工作内存重复这个控制循环。当工作内存的内容不与任何规则的条件匹配时这个过程终止。

冲突消解（conflict resolution）就是从冲突集合中选取一个要激发的规则。冲突消解策略可以很简单，比如选取条件与世界状态匹配的第一个规则；它也可以很复杂，包含复杂的启发性选取策略。后者是在产生式系统中向搜索算法加入启发控制的一种重要方式。

纯粹的产生式系统模型不包含任何从搜索死端恢复的机制；它只是不断执行识别－动作循环直到不再有任何已经使能的产生式便停止。在实践中，许多产生式系统的实现允许在这种情况下回溯到工作内存的前一个状态。

图 6-1 画出了产生式系统的示意图。

图 6-2 给出了一个非常简单的产生式系统例子的执行情况。这是一个产生式系统程序，它对由字母 a、b 和 c 组成的字符串进行了排序。在这个例子中，如果一个产生式的条件和工作内存中字符串的一部分匹配，那么它便得以使能。当一个规则激发时，和这个规则的条件匹配的子串替代为这个规则的右边部分。产生式系统是一种通用的计算模型，可以通过编程用它来完成在计算机上可以做的任何事情。然而，它的真正强大之处是为基于知识的系统提供了一种重要的体系结构。

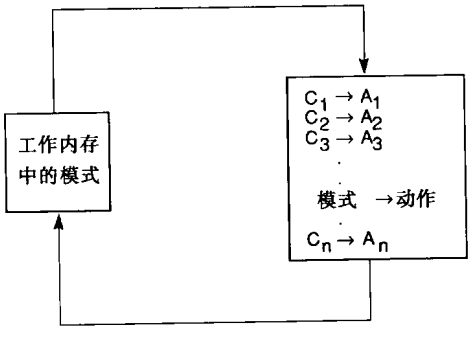


图 6-1　产生式系统

注：重复这个循环，直到工作内存中的规则不再与任何产生式的条件匹配

产生式集合：

1. ba → ab
2. ca → ac
3. cb → bc

迭代	工作内存	冲突集合	激发的规则
0	cbaca	1, 2, 3	1
1	cabca	2	2
2	acbca	2, 3	2
3	acbac	1, 3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	Ø	Halt

图 6-2　一个简单产生式系统的执行过程

这种基于产生式的计算设计思想起源于 Post（1943），Post 最早把产生式规则模型作为正式的计算理论提出。这一理论的主要概念是一系列用于字符串的重写规则（在很多方面都很像例 3.3.6 中的语法解析规则）。它也与马尔可夫算法（Markov 1954）所用的方法有着非常密切的关系，而且就像马尔可夫算法一样，其威力可以与图灵机相提并论。

产生式规则的一个有趣应用是用它来对人类认知建模，在 20 世纪 60 年代和 70 年代，Newell 和 Simon 在卡内基工学院（现在的卡内基－梅隆大学）进行了这项研究。很大程度上来说，是他们开发的程序（例如通用问题求解器（General Problem Solver））奠定了产生式系统在 AI 中的重要地位。在这项研究中，他们观察并记录了人类在求解各种问题时的行为，比如求解谓词演算这样的逻辑问题和国际象棋这样的博弈问题。他们把接受实验者的 protocol（行为模式，包括对问题求解过程、眼睛转动等的原始描述）记录下来，并把这些记录分解为一些基本要素。把这些要素当作接受实验者求解问题知识的基本二进制位，并按照图（称为问题行为图（problem be-

havior graph)) 搜索的方式来编写二进制位。然后使用产生式系统来实现对这种图的搜索。

这些产生式规则表示了一系列接受实验者求解问题的技巧。目前注意的焦点表示为世界的当前状态。在产生式系统的执行过程中，问题求解器把当前焦点与产生式规则进行匹配，然后用匹配的规则来修改当前状态，使焦点转移到另一个编码为产生式的技巧，并这样继续下去。

有必要指出的是，在这项研究中 Newell 和 Simon 使用了产生式系统，不仅把它作为一种实现图搜索的工具，而且还把它作为人类求解问题的实际模型。产生式对应于人类长期记忆中的问题求解技能。和长期记忆中的技能一样，这些产生式是不随系统的执行而改变的；它们会被特定问题实例的“模式”所调用，而且不需对以前存在的知识“重新编码”就可以加入新的技能。产生式系统的工作内存对应于短期记忆或人类当前注意的焦点，而且产生式系统的工作内存还描述了求解一个问题实例的当前阶段。在问题已经得以解决后，通常不保留工作内存中的内容。

这些产生式系统技术的起源在 Newell 和 Simon 所著的《人类问题求解》（1972）和 Luger（1978，1994）中有进一步的介绍。Newell 和 Simon 等人还用产生式规则对初学者和专家在求解各种问题（比如代数应用题和物理问题）时的差异进行了建模（Larkin et al. 1980；Simon and Simon 1978）。产生式系统还是研究人类和计算机学习的基础（Klahr et al. 1987）；ACT* （Anderson 1983b）和 SOAR（Newell 1990）就是以这种方式建立的。

产生式系统提供了一种模型，利用这个模型不仅可以把人类的专业技能以规则的形式编码，而且还可以设计模式驱动的搜索算法，而这些恰恰是设计基于规则的专家系统的核心任务。在专家系统中，产生式系统未必就一定是用来对人类的求解问题行为建模；然而，产生式系统所具有的对人类求解问题建模的能力（规则的模式化、知识和控制的分离、工作内存和求解问题知识的分离）使它成为设计和建立专家系统（见 8.1 节和 8.2 节）的理想工具。

在卡内基·梅隆大学进行的对产生式系统语言的研究产生了一类非常重要的 AI 语言。这就是 OPS 语言；OPS 代表公认产生式系统（Official Production System）。尽管 OPS 最初是为对人类求解问题方法建模而设计的，但已经证明它对设计专家系统程序和其他 AI 应用也非常有效。VAX 的配置系统 XCON 以及数字设备公司（DEC）开发的其他很多专家系统就是用 OPS5 实现的（McDermott 1981，1982；Soloway et al. 1987；Barker and O'Connor 1989）。在很多地方可以得到供 PC 和工作站使用的 OPS 解释程序。CLIPS 是美国航空航天局（NASA）用 C 语言实现的面向对象产生式系统，CLIPS 的应用很广。JESS 是由 Sandia 国家实验室用 Java 实现的一个产生式系统。

在下一节中将给出一些例子，用来说明如何应用产生式系统来求解不同的搜索问题。

6.2.2 产生式系统的例子

例 6.2.1 再谈 8 格拼图游戏问题

第 3 章介绍的 8 格拼图游戏问题产生的搜索空间，不仅其复杂度足以引起我们的兴趣，而且其空间较小足以驾驭，所以它经常被用来试验不同的搜索策略，比如深度优先搜索、宽度优先搜索以及第 4 章介绍的启发式搜索。现在我们用产生式系统来求解这个问题。

我们仍采用前面章节的做法，把不同标号将牌的移动看成“空位的移动”，这样可以使表述大大简化。图 6-3 中给出了用产生式定义的合法移动。当然，只有当空位在正中央时这 4 种移动才都适用；当空位在一角时仅有两种移动适用。如果现在确定了 8 个将牌的开始状态和结束状态，那么就可以建立起针对该问题搜索空间的产生式系统。

对该问题的一种实现可以是这样的：用一个包含 9 个参数（对应 8 张将牌和一个空位的位置）的状态谓词表示每一种棋盘布局；把规则写为蕴涵的形式，利用其前提检查执行该规则所需的条件。或者，也可以使用数组或列表结构来表示棋盘状态。

起始状态：

2	8	3
1	6	4
7		5

目标状态：

1	2	3
8		4
7	6	5

产生式集合：

条件		动作
目标状态在工作内存中	→	终止
空位不在左侧的边上	→	向左移动空位
空位不在靠上的边上	→	向上移动空位
空位不在右侧的边上	→	向右移动空位
空位不在靠下的边上	→	向下移动空位

工作内存中是目前的棋盘状态和目标状态。

控制制度：

1）依次试验每个产生式。
2）不允许循环。
3）当发现目标时停止。

图 6-3　用产生式系统求解 8 格拼图游戏问题

图 6-4 给出了求解图 6-3 定义的搜索空间的一个例子，这个例子摘自 Nilsson（1980）。因为如果不加限制，那么解路径可能走得非常深，所以向搜索中加入了深度限制（加入深度限制的一个简单途径是跟踪当前路径的长度/深度，如果超过了深度限制就强制搜索回溯）。图 6-4 所示情况使用的深度限制是 5。注意：工作内存中的可能状态数量随搜索深度呈指数增长。

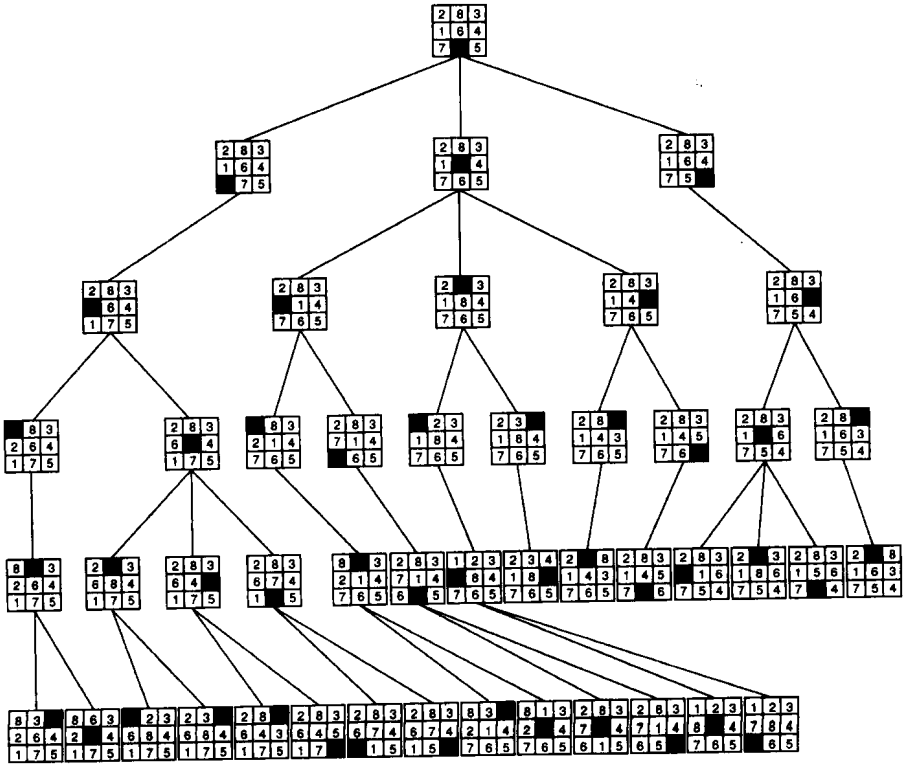


图 6-4　产生式系统对 8 格拼图游戏问题空间的搜索

注：这个系统采用循环探测，并且搜索的深度限制不超过 5。摘自 Nilsson（1971）

例6.2.2 骑士周游问题

在国际象棋中，骑士可以横向或纵向移动两个方格后再沿垂直方向移动一个方格（只要不超出棋盘的限制）。因此骑士最多可以有8种可能的移动（见图6-5）。

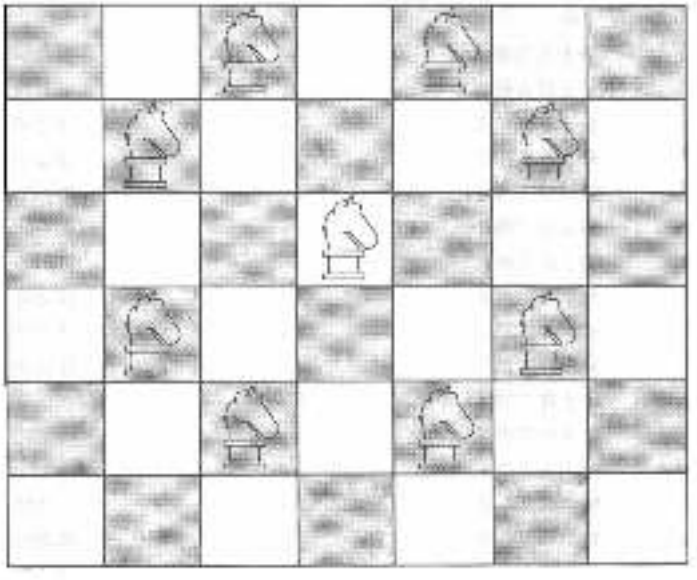


图6-5 国际象棋中骑士的合法移动

骑士周游问题的传统定义是这样的：找一个合法移动序列使骑士走过棋盘的每一个方格恰好一次。这个问题一直是开发和展示搜索算法的一个主要素材。在本章的例子中，我们使用这个问题的一种简化版本。我们的问题是，在一个缩小的棋盘上（3×3），是否存在一系列合法移动使骑士可以从一个方格移动到另一个方格。

图6-6显示了一个3×3的棋盘，并且用1到9九个整数对其中的每个方格进行标号。使用这种标号方案而不使用更常规的方法——给出每个空格的行列号——是为了进一步简化这个例子。由于问题的规模减小了，因此我们干脆枚举出不同的移动，而不去设计通用的移动操作符。我们用谓词演算来描述棋盘上的合法移动，并把所用的谓词称为move，它的参数是合法移动的起始方格和结束方格。例如，move(1,8)是把骑士从左上角移动到底行的中间。图6-6中的谓词枚举了3×3棋盘上的所有可能移动。

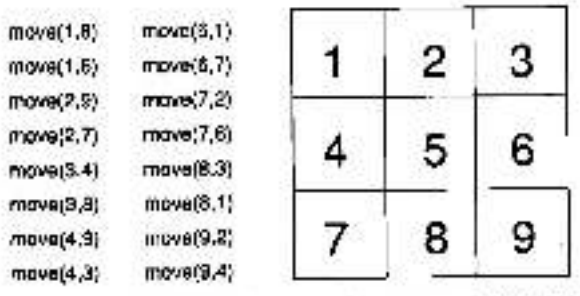


图6-6 简化骑士周游问题的3×3棋盘以及相应的移动规则

可以使用产生式系统来求解3×3版本的骑士周游问题。把每一个移动表示为这样的一个规

则：规则的条件是骑士在特定方格的位置，规则的动作把骑士移动到另一个方格。表6-1中的16个产生式表示了骑士的所有可能移动。

表6-1　3×3版本骑士周游问题的产生式规则

规则编号	条　件		动　作
1	骑士在方格1	→	移动骑士到方格8
2	骑士在方格1	→	移动骑士到方格6
3	骑士在方格2	→	移动骑士到方格9
4	骑士在方格2	→	移动骑士到方格7
5	骑士在方格3	→	移动骑士到方格4
6	骑士在方格3	→	移动骑士到方格8
7	骑士在方格4	→	移动骑士到方格9
8	骑士在方格4	→	移动骑士到方格3
9	骑士在方格6	→	移动骑士到方格1
10	骑士在方格6	→	移动骑士到方格7
11	骑士在方格7	→	移动骑士到方格2
12	骑士在方格7	→	移动骑士到方格6
13	骑士在方格8	→	移动骑士到方格3
14	骑士在方格8	→	移动骑士到方格1
15	骑士在方格9	→	移动骑士到方格2
16	骑士在方格9	→	移动骑士到方格4

接着，我们定义一个递归过程来实现产生式系统的控制算法。因为path(X, X)仅与path(3, 3)或path(5,5)这样的谓词合一，所以它定义了所要求的终止条件。如果path(X, X)不成功，我们就检索产生式规则寻找可能的下一个状态，然后重复这一过程。下面的两条谓词演算公式给出了通用的递归路径定义：

$\forall X\ path(X, X)$

$\forall X, Y\ [path(X, Y) \leftarrow \exists Z[move(X, Z) \wedge path(Z, Y)]]$

工作内存（递归谓词path的参数）中既包含了当前棋盘状态，又包含了目标状态。控制制度不断地应用规则，直到当前状态等于目标状态，然后停止。一种简单的冲突消解方案是激发第一个“不导致搜索进入循环”的规则。因为这个搜索可能进入死端（每一种可能的移动都是曾经访问过的状态，从而导致循环），所以控制制度应该必须允许回溯；图6-7中的表格说明了这个产生式系统在确定是否存在从方格1到方格2的路径时的执行情况。图6-8给出了将path定义为产生式系统的详细说明。

	工作内存			
迭代次数	当前方格	目标方格	冲突集合（规则编号）	激发规则
0	1	2	1, 2	1
1	8	2	13, 14	13
2	3	2	5, 6	5
3	4	2	7, 8	7
4	9	2	15, 16	15
5	2	2		终止

图6-7　3×3版本骑士周游问题的一个产生式系统解

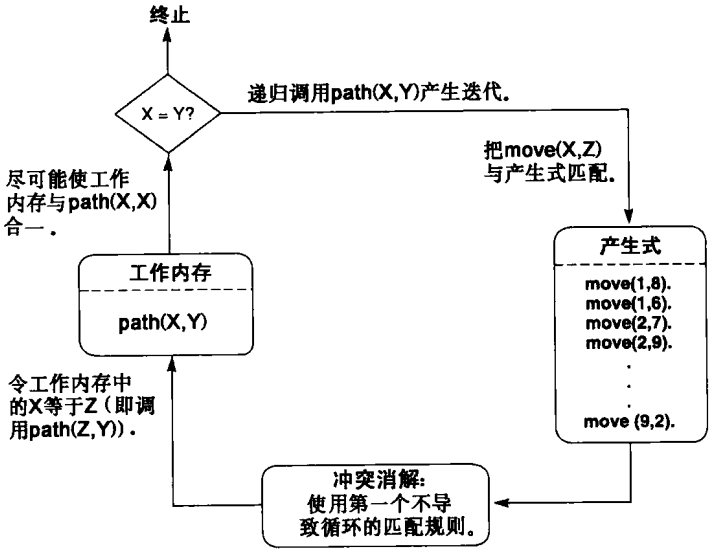


图 6-8　递归的 path 算法：产生式系统

当用产生式系统搜索状态空间图时很容易产生无限的循环。在产生式系统中特别难以辨认这些循环，因为系统可能以任意顺序激发规则。也就是说，在系统的执行中可能出现循环，但是从对规则集的语法检查中很难发现这些循环。例如，对于骑士周游问题中的“移动”规则，如果把它们按 6.1 节中那样排序，并且使用的冲突消解策略是选择第一个匹配的规则，那么模式 move(2, X) 将与 move(2,9) 匹配，指出要移动到方格 9。在下一次迭代中，模式 move(9, X) 会与 move(9,2) 匹配，把搜索带回到方格 2，导致循环。

为了防止循环，pattern_search 检查一个包含已访问过状态的全局列表（closed）。所以，实际的冲突消解策略是：选择第一个匹配的通向未访问过状态的移动。在产生式系统中，记录这种特定案例数据作为以前访问过状态的列表的恰当地方，不是全局的 closed 列表，而是工作内存本身。我们可以修改 path 谓词以使用工作内存进行循环探测。假定对我们的谓词演算语言进行扩充，加入一个特定的结构 assert(X)，该谓词的作用是使它的参数 X 进入工作内存。assert 不是一个普通的谓词，而是一种要执行的动作，所以它总是成功的。

引入 assert 的目的是向工作内存中放入一个“标记”，来表明何时已访问过一个状态。这个标记表示为一个一元谓词 been(X)，它以棋盘的一个方格作为参数。当访问一个新的状态 X 时，将 been(X) 加入到内存中。于是冲突消解策略可以要求在引发 move(X, Z) 之前必须保证 been(Z) 不在工作内存中。对于一个具体的 Z 值，可以通过把一个模式与工作内存匹配来测试这个要求。

可以把修改后的递归路径定义写为：

$\forall X$ path(X, X)

$\forall X, Y$ [path(X, Y) $\leftarrow \exists Z$ move(X, Z) $\wedge \neg$ (been(Z)) $\wedge$ assert(been(Z)) $\wedge$ path(Z, Y)]]

根据这个定义，move(X, Z) 在第一个匹配的 move 谓词上成功，这会绑定一个值到 Z。如果 been(Z) 和工作内存中的一个入口（entry）匹配，那么¬ (been(Z)) 会失败（也就是它为假）。于是 pattern_search 回溯并尝试另一个与 move(X, Z) 匹配的谓词。如果方格 Z 是一个新的状态，那么继续搜索，并把 been(Z) 放入工作内存以防止以后的循环。产生式的实际引发发生在 path 算法递归时。这样便通过向工作内存中放入 been 谓词实现了产生式系统的循环探测。

例 6.2.3　完整的骑士周游问题

我们可以把求解 3×3 版本骑士周游问题的方法推广到完整的 8×8 版本。因为对于这样一个复杂的问题，枚举移动是没有意义的，所以我们用 8 条规则代替 16 种移动实例来产生合法的骑士移动。这些移动（产生式）对应于骑士可以移动的 8 种可能方式（见图 6-5）。

如果用行号和列号来检索国际象棋棋盘，那么我们把骑士向下移动两个方格并向右移动一个方格定义为：

条件：当前行≤6 ∧ 当前列≤7

动作：新的行 = 当前行 +2 ∧ 新的列 = 当前列 +1

如果我们使用谓词演算来表示产生式，那么可以用谓词 square(R, C) 来定义棋盘的一个方格，代表棋盘第 R 行和第 C 列的方格。这样可以用谓词演算把上面的规则重新写为：

```
move(square(Row, Column), square(Newrow, Newcolumn)) ←
    less_than_or_equals(Row, 6) ∧
    equals(Newrow, plus(Row, 2)) ∧
    less_than_or_equals(Column, 7) ∧
    equals(Newcolumn, plus(Column, 1))
```

plus 是用来做加法的函数；less_than_or_equals 和 equals 的数学解释也是显而易见的。类似地，可以设计出另外 7 种有关骑士的可能移动规则。这些规则代替了该问题的 3×3 版本中的 move 实例。

3×3 版本中的 path 的定义也为这个问题提供了一个控制循环策略。正如我们所看到的，当谓词演算描述由具体过程（比如 pattern_search 算法）解释时，要对谓词演算的语义进行一些细微的改变。其中一种改变就是求解目标的序列化方式。这向谓词演算表达式强加了顺序性，或者称其为*过程性语义*（procedural semantics）。另一种变化是引入了像 assert 这样的*元逻辑*（meta-logical）谓词，这些谓词表明动作超出了对谓词演算表达式的真值解释范围。

例 6.2.4　以产生式系统实现财务顾问程序

在第 2 章和第 3 章中，我们开发了一个小的财务顾问程序——使用谓词演算表示财务知识并用图搜索推理出适当的结论。产生式系统为实现这个财务程序提供了一种更自然的工具。逻辑描述的蕴涵组成了产生式。针对案例的信息（咨询者的工资、要供养的人数等）放入工作内存；当满足规则的前提时，便使能这个规则；然后从冲突集合中选取一个规则并将其激发，再把它的结论加入工作内存。重复这个过程直到所有可能的顶层结论都已经被加入到工作内存。事实上，很多专家系统的“外壳”（包括 JESS 和 CLIPS）都是扩充了的产生式系统，扩充的部分包括用户界面、处理推理中的不确定性、编辑知识库以及跟踪执行过程等。

6.2.3　产生式系统中的搜索控制

产生式系统模型为向搜索算法中加入启发式控制提供了广泛的机会。包括选取数据驱动或目标驱动搜索策略，以及选取不同的冲突消解策略。

通过选取数据驱动或目标驱动搜索策略来控制搜索

数据驱动搜索从问题的描述（比如一系列逻辑公理、疾病的症状或需要解释的数据体）开始，从数据中推理出新的知识。这些是这样实现的：首先对世界的当前描述应用推理规则、在博弈中选取合法的移动，或者应用其他的“状态 - 产生”操作，把得到的结果加入到问题的描述中。重复这个过程直到达到目标状态。

数据驱动推理和计算的产生式系统模型有着密切的关系。“世界的当前状态”（要么是已经

假定为真的数据，要么是前面应用产生式规则推出的数据）放入工作内存。然后识别－动作循环把当前的状态和产生式规则的（有序）集合进行匹配。当这些数据与某一产生式规则的条件匹配（合一）时，这个产生式的动作便（通过修改工作内存）向世界的当前状态加入一条新的信息。

所有产生式都是“条件→动作”的形式。当它的条件和工作内存中的某个元素匹配时，便执行其动作。如果把产生式规则写成逻辑蕴涵的形式，并且把产生式的动作设计成向工作内存中加入断言，那么引发规则的行为就相当于应用假言推理的推理规则，结果会在图中产生一个新的状态。

图 6-9 给出了对表示为命题演算蕴涵的产生式集合的一种简单数据驱动搜索。这里使用的冲突消解策略就是以下两个原则：选取最近最少激发过的（或根本没有激发过的）使能规则；如果出现“平局”的情况，选取其中的第一个规则。当达到目标时终止执行。图中也给出了规则激发的顺序和执行过程中各个阶段的工作内存及搜索的空间图。

产生式集合：

1. $p \wedge q \rightarrow goal$
2. $r \wedge s \rightarrow p$
3. $w \wedge r \rightarrow q$
4. $t \wedge u \rightarrow q$
5. $v \rightarrow s$
6. $start \rightarrow v \wedge r \wedge q$

执行过程：

迭代次数	工作内存	冲突集合	激发的规则
0	start	6	6
1	start, v, r, q	6, 5	5
2	start, v, r, q, s	6, 5, 2	2
3	start, v, r, q, s, p	6, 5, 2, 1	1
4	start, v, r, q, s, p, goal	6, 5, 2, 1	终止

执行过程所搜索的空间：

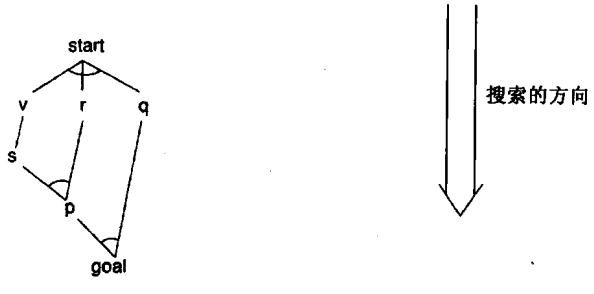


图 6-9　产生式系统的数据驱动搜索

前面我们以数据驱动的方式讨论了产生式系统；另一方面，还可以用产生式系统来产生目标驱动的搜索。根据第 3 章中的定义，目标驱动搜索从目标开始，倒推回满足目标的问题事实。为了在产生式系统中实现目标驱动搜索，我们把目标放入工作内存，并和产生式规则的动作匹配。就像在数据驱动的推理中匹配（比如通过合一）产生式的条件那样匹配产生式的动作。结论（动作）和目标匹配的所有产生式规则组成冲突集合。

当一个规则的动作匹配后，它的条件加入到工作内存成为新的搜索子目标（状态）。然后再把新的状态和其他产生式规则的动作进行匹配。继续这个过程，直到找到了事实，事实通常在问题的初始描述中，但也可能像专家系统经常使用的方法那样，通过直接向用户询问特定信息得到事实。当按这种倒推的方式找到所有激发产生式的条件都为真时，搜索终止。这些条件和通向原始目标的激发规则链组成了对目标的证明，这种证明是通过一系列环环相扣的推理（比如假言推理）实现的。图 6-10 中给出了一个目标驱动推理的例子，它使用的产生式与图 6-9 中使用的产生式相同。注意：目标驱动搜索与相应的数据驱动版本引发的产生式是不同的，而且搜索空间也不一样。

产生式集合：

1. p ∧ q → goal
2. r ∧ s → p
3. w ∧ r → p
4. t ∧ u → q
5. v → s
6. start → v ∧ r ∧ q

执行过程：

迭代次数	工作内存	冲突集合	激发的规则
0	goal	1	1
1	goal, p, q	1, 2, 3, 4	2
2	goal, p, q, r, s	1, 2, 3, 4, 5	3
3	goal, p, q, r, s, w	1, 2, 3, 4, 5	4
4	goal, p, q, r, s, w, t, u	1, 2, 3, 4, 5	5
5	goal, p, q, r, s, w, t, u, v	1, 2, 3, 4, 5, 6	6
6	goal, p, q, r, s, w, t, u, v, start	1, 2, 3, 4, 5, 6	终止

执行过程所搜索的空间：

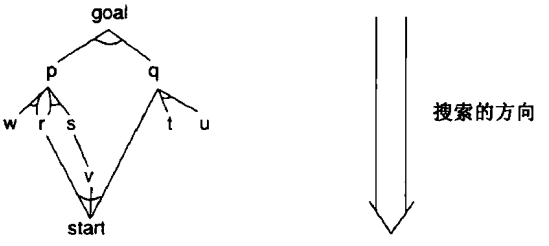


图 6-10　产生式系统的目标驱动搜索

通过以上讨论可以看出，产生式系统既可用于数据驱动搜索又可用于目标驱动搜索。产生式规则就是改变图中状态的推理集合（基于规则的专家系统中的“知识”）。一种情况是，把世界的当前状态（描述这个世界的一组真实陈述集）与产生式规则的条件匹配，然后执行匹配规则的动作来产生对世界的另一种（真实）描述，这称为数据驱动搜索。

另一种做法是，把目标和产生式规则集合中规则的动作部分匹配，然后把匹配规则的条件作为要证明的子目标（证明的方法是在产生式系统的下一轮循环中匹配规则的动作），这便是目标驱动的问题求解方法。

因为既可以按数据驱动的方式又可以按目标驱动的方式执行规则，所以我们可以把每一种方法控制搜索的效率进行对比。这两种策略的复杂度都可以用像分支因子这样的概念来衡量。利用这些搜索复杂性尺度，我们可以对问题求解器的数据驱动和目标驱动版本的代价进行估计，从而帮助我们选择最高效的策略。

我们还可以采用组合策略。例如，我们可以先按正向搜索直到状态数变得很大，然后切换到目标导向的搜索使用可能的子目标在备选状态中做出选择。这种情况的风险是，当使用了启发式搜索或最佳优先搜索（见第 4 章）时，每种策略实际搜索图的部分可能相互遗漏，从而导致最终要做的搜索比简单方法更多，如图 6-11 所示。然而，当空间的分支固定不变并使用穷举搜索时，组合搜索策略可以大大削减搜索的空间量，如图 6-12 所示。

通过规则结构控制搜索

产生式系统的规则结构——包括规则和动作的区分以及试验条件的顺序——决定了搜索空间的方式。在把谓词演算作为一种表示语言介绍时，我们强调了其语义的声明（declarative）性。也就是说，谓词演算表达式仅定义了问题域中的真实关系，没有对以什么顺序来解释它的各个部分做出任何规定。因此，如果一个规则为 $\forall X(foo(X) \wedge goo(X) \rightarrow moo(X))$，那么根据谓词演算规则，可以把这个规则表示为另一种形式：$\forall X(foo(X) \rightarrow moo(X) \vee \neg goo(X))$。可以用 2.1 节中的真值表方法来说明这两个子句的等价关系已经作为一个练习布置过（第 2 章）。

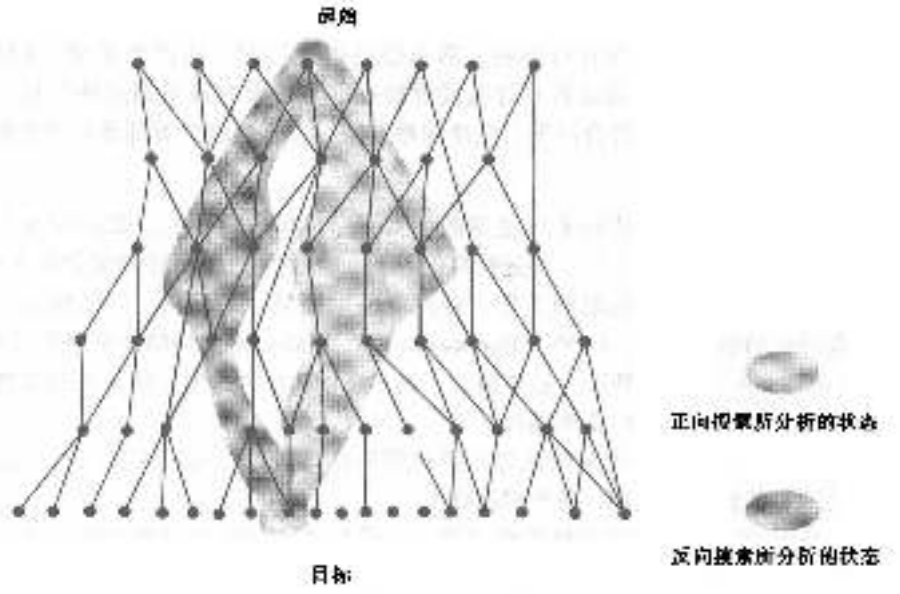


图6-11 双向搜索相互迷惑导致了额外搜索

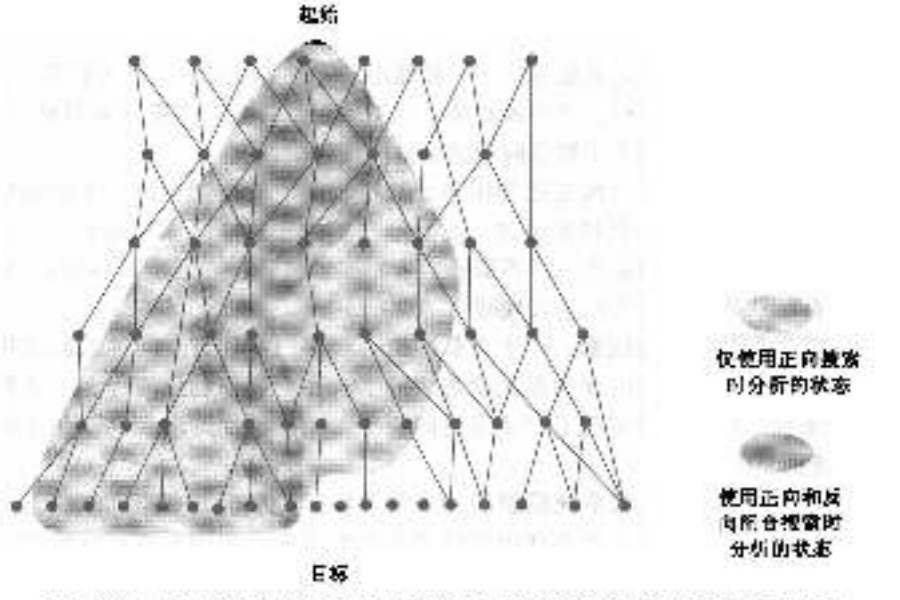


图6-12 双向搜索在中间相遇，排除了绝大部分非双向搜索所分析的空间

尽管这两个公式是逻辑等价的，但是当把它们解释为产生式对就会产生不同的结果，因为产生式系统的实现对规则的匹配和激发强加了顺序性。由于这个原因，规则的特定形式决定了把一个规则和一个问题实例匹配的简易性（或者可能性）。这种产生式系统解释规则方式的不同而产生的结果。产生式系统对用来组成规则的声明性语言强加了过程性语义。

因为产生式系统以特定的顺序试验每一条规则，所以程序员可以通过修改产生式集合中规则的顺序和结构来控制搜索。尽管 $\forall X(foo(X) \wedge goo(X) \rightarrow moo(X))$ 和 $\forall X(foo(X) \wedge \neg moo(X) \rightarrow \neg goo(X))$ 在逻辑上是等价的，但是它们在搜索实现中的行为并不相同。

人类专家把关键性的启发编码进他们的技能规则中。作为前提的顺序信息编码了求解问题

的重要过程信息。在建立“像专家那样求解问题”的程序时保持这种顺序是很重要的。比如当一个机修工说“如果引擎不转，并且灯不亮，那么检查电池”时，他指出了动作的特定顺序。一条逻辑上等价的语句“引擎转动或者车灯亮或者检查电池”没有捕捉到这种信息。规则的形式对于控制搜索、使系统的行为符合逻辑、使规则激发的过程更好理解等都是非常关键的。

通过冲突消解策略控制搜索

尽管产生式系统（像基于知识的系统使用的所有体系结构一样）允许把启发按照规则本身的知识内容编码，但是它还提供了另一种进行启发性控制的手段，即通过冲突消解来控制搜索。尽管最简单的冲突消解策略就是选取和工作内存内容匹配的第一个规则，但是原则上可以应用任何策略来实现冲突消解。例如，OPS5（Brownston et al. 1985）所支持的冲突消解策略包括：

1）折射（refraction）。一个规则一旦被激发，就不可以再次被激发，除非与其条件匹配的工作内存元素已经修改了。这样可以抵制循环。

2）最新性（recency）。最新性策略优先选择规则条件与最近加入到工作内存中的模式相匹配的规则。这使搜索的焦点集中在一条推理路线上。

3）特殊性（specificity）。这个策略假定使用一个更特殊的问题求解规则会比使用更一般的规则更合适。一个规则比其他规则更特殊的标准是其条件更多，这也意味着与其匹配的工作内存模式会更少。

6.2.4　AI 产生式系统的优点

正如前面的例子所说，产生式系统提供了一种通用的框架实现搜索。由于它的简洁性、可修改性和应用求解问题知识的灵活性，产生式系统已经被证明是建立专家系统和其他 AI 应用的重要工具。使用产生式系统来求解人工智能问题的主要优点包括：

知识和控制的分离。产生式系统是把知识和计算机程序中的控制分离开来的较好模型。控制是由产生式系统的识别 - 动作循环提供的，而求解问题的知识编码成规则本身。这种分离所带来的优点包括修改知识库的简易性——不需要对程序控制代码进行修改；并且从另一方面说，也可以修改程序控制代码而不改变产生式规则。

可以自然地映射到状态空间搜索。产生式系统的各个部分与状态空间搜索的结构之间存在一种非常自然的对应关系。工作内存的逐次状态组成了状态空间图的结点。产生式规则是状态之间可能变换的集合，冲突消解实现了对状态空间中分支的选择。这些规则简化了搜索算法的实现、调试和建档。

产生式规则的模块性。产生式系统模型的一个重要特征是产生式规则之间缺乏语法上的相互作用性。规则仅可以通过改变工作内存中的模式来影响其他规则的激发；它们不可以像子过程那样直接“调用”其他规则，也不可以设定其他规则中的变量值。规则的变量范围就局限在规则个体中。这种语法上的独立性允许我们逐步开发专家系统，可以不断地加入、删除或修改系统中的知识（规则）。

模式导向控制。AI 程序所针对的问题要求其具有很大的灵活性。产生式系统中的规则可以以任意顺序激发，这恰好迎合了灵活性目标。组成世界当前状态的问题描述决定了冲突集合，从而决定了特定的搜索路径和解。

有很多种方式可以用于对搜索的启发式控制（在前面一节中描述了启发式控制搜索的几种技术）。

易于跟踪和解释。产生式系统中规则的模块性和规则执行的迭代性使跟踪产生式系统的执行过程更简单。在识别 - 动作循环的每一阶段，我们都可以显示出已选出的规则。因为每一条规

则对应于问题求解知识的一个知识块，所以规则的内容提供了对系统当前状态和动作的有意义解释。而且，在一个求解过程中使用的规则链不仅反映了图中的一条路径，而且反映了人类专家的“推理路线”，在第 8 章中我们会更详细地讨论这一点。相反，在像 Pascal 和 FORTRAN 这样的传统语言中，单独一行代码或一个过程几乎是没有意义的。

语言独立性。产生式系统的控制模型独立于规则和工作内存所选取的表示，只要选择的表示支持模式匹配。在前面我们把产生式规则描述为 A⇒B 这样的谓词演算蕴涵形式，我们可以利用 A 的真值和假言推理这一推理规则得出结论 B。尽管使用逻辑作为表示知识的基础和可靠推理规则的来源有很多优点，但是也可以在产生式系统中使用其他表示，例如 CLIPS 和 JESS。

尽管谓词演算推理具有逻辑可靠推理的优点，但是很多问题需要非逻辑意义上的可靠推理。这些问题需要使用不确定的证据和默认的假定进行概率推理。后面几章（第 7、8、9 章）讨论了具有这种能力的其他推理规则。然而，不管使用什么类型的推理规则，都可以采用产生式系统来搜索其状态空间。

为人类问题求解提供了一种合理的模型。对人类问题求解建模是产生式系统的最早应用之一，参见 Newell and Simon（1972）。在很多认知科学研究领域中仍然使用产生式系统来对人类的行为建模（见第 16 章）。

模式导向搜索使我们可以用谓词演算探索逻辑推理空间。很多建立在这一技术基础上的应用使用谓词演算对世界的某一特征（比如时间和变化）建模。在下一节中我们介绍了*黑板系统*，它是产生式系统方法学的一种变体。在该方法学中，针对任务的各组产生式规则组合成多个*知识源*，通过全局工作内存或*黑板*进行通信，相互协作求解问题。

6.3　用于问题求解的黑板结构

黑板（blackboard）是本章要介绍的最后一种控制机制。当我们需要以非常确定的方式分析逻辑推理空间中的状态时，产生式系统提供了较大的灵活性，允许我们在工作内存中同时表示多个局部解，并通过冲突消解选取下一个状态。黑板扩展了产生式系统，允许我们把工作内存组织成多个独立的模块，每个模块对应于产生式规则的不同子集。黑板把这些产生式规则的独立集合集成起来，并在一个全局的结构（即*黑板*）中协调这些问题求解器（有时也称为*知识源*（knowledge sources））的动作。

很多问题需要协同处理许多不同类型的主体。例如，语音理解程序必须首先把话语处理成数字波形。随着解码过程的继续，必须找到这句话中的单词，把这些单词组成句子，最后产生话语含义的语义表示。

当多个进程必须协同工作求解一个问题时也会发生类似的问题。这样的一个例子是传感器合成问题（Lesser and Corkill 1983）。假定有一个传感器网络，其中的每一个传感器由一个进程所监控。另外还假定各个进程可以通信，并且对每个传感器数据的正确解释依赖于网络中其他传感器接收的数据。这种问题在很多场合中都存在，比如跟踪一个越过多个雷达站的飞机以及合并生产过程中的多个传感器读数。

黑板结构（blackboard architecture）作为一种控制模型已经应用到以上问题中，它也用于解决其他需要协调多个过程或知识源的问题。*黑板*是一个全局性的中央数据库，用来实现各个独立异步知识源间的通信，每个知识源集中表示一个特定问题的有关方面。图 6-13 给出了黑板设计的示意图。

在图 6-13 中，每个*知识源*（knowledge source）KS_i 从黑板取得数据，处理数据，并将其结果返回到黑板供其他知识源使用。每个知识源 KS_i 都是一个分离的根据其自身规范运作的过程，

从这个意义上来说每个知识源都是独立的，而且当使用多处理器系统时，它是独立于问题中其他处理过程的。不论何时只要知识源 KS_i 发现黑板中投入了合适的输入数据，便开始它的操作，从这个意义上来说它是一个异步系统。当它完成处理后，把结果投递到黑板并等待新的合适的输入数据。

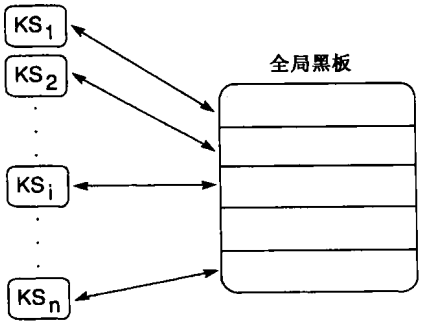


图 6-13　黑板体系结构

使用黑板方法来组织庞大的程序最早出现在 HEARSAY-Ⅱ的研究中（Erman et al. 1980，Reddy 1976）。HEARSAY-Ⅱ是一个语音理解程序；设计它的最初目的是用做计算机科学文献图书馆数据库的前端。图书馆的用户以讲英语的方式对计算机提出类似这样的查询“有 Feigenbaum 和 Feldman 编著的任何资料吗?”这台计算机会用图书馆数据库中的信息回答这个问题。语音理解把多种不同的处理集成到一起，其中的每个部分需要迥然不同的知识和算法，而且每一部分都可能具有指数级的复杂度。在语音翻译中，如下处理是相互制约的：信号处理；音素、音节和单词的识别；语法分析；语义分析。

HEARSAY-Ⅱ使用黑板结构来协调这个复杂任务所需的多个不同知识源。黑板通常是按两维来组织的。在 HEARSAY-Ⅱ中，这两维分别是产生语音变化的时间和讲话的分析层次。每个分析层次由不同的知识源来处理。这些分析层次是：

KS_1　声音信号的波形。

KS_2　声音信号的音素或可能的声音片段。

KS_3　音素可能产生的音节。

KS_4　一个 KS 分析出的可能单词。

KS_5　第二个 KS 分析出的可能单词（通常从数据的不同部分考虑单词）。

KS_6　努力产生可能单词序列的 KS。

KS_7　把单词序列组成可能短语的 KS。

我们可以把这些处理可视化为图 6-13 中的各个部分。在处理口语时，语音信号波形输入到最底层。激发负责处理这个项目的知识源，并将其解释投递到黑板供其他合适的处理过程来拾取。由于口语语言的含糊性，因此可以在黑板的每个层次给出多个竞争的假设。较高层次的知识源会尝试消除这些竞争假设的歧义。

不应该把 HEARSAY-Ⅱ的分析简单看成：较低的层次产生数据，然后较高的层次分析数据。它要比这复杂得多。如果一个层次上的 KS 不能处理（理解）发给它的数据，那么这个 KS 可以要求向它发送数据的 KS 再试一次，也就是对数据做出另一种假设。此外，不同的 KS 可以同时工作在一句话的不同部分上。所有的处理就像前面提到的那样是异步的，而且是数据驱动的；当有输入数据时它们就工作，而且持续工作直到完成所有任务，然后投递出它们的结果并等待下一个任务。

有一个称为*调度程序*（scheduler）的 KS，处理 KS 之间“消费数据和投递结果”的通信。这个调度程序对每个 KS 的活动结果进行评定，因此能够通过优先级队列提供一些求解问题的指导。如果没有 KS 是活动的，那么调度程序确定任务已经完成而终止。

当 HEARSAY 程序使用大约 1000 个单词的数据库时工作得非常好，尽管有点慢。但当进一步扩充数据库时，用于知识源的数据变得更加复杂超出了它们的处理能力。HEARSAY-Ⅲ（Balzer et al. 1980，Erman et al. 1981）是对 HEARSAY-Ⅱ所采用方法的推广。不再需要 HEARSAY-Ⅱ的时间维，但保留了用于各个分析层次的多个知识源。HEARSAY-Ⅲ的黑板是用来和通用的关系

数据库系统进行交互。事实上，HEARSAY-Ⅲ是用来设计专家系统的一个外壳；见8.1.1节。

HEARSAY-Ⅲ的一个重要改进是把调度KS分离出来（和前面描述的用于HEARSAY-Ⅱ的调度KS相同），并使其作为一个分离的黑板控制器对第一块（问题域）黑板进行控制。这第二块黑板允许把调度处理分解成多个独立的KS，负责不同特征的解过程（例如，什么时候和如何应用域知识）。因此，第二块黑板可以比较和平衡每个问题的不同解（Nii and Aiello 1979，Nii 1986a，1986b）。另一种黑板模型把知识库的重要部分保留在黑板中，而不是把所有知识都分布到各个知识源中（Skinner and Luger 1991，1992）。

6.4 结语和参考文献

本章讨论了如何实现第3章和第4章介绍的搜索策略。因此这两章结语中列出的参考文献也适用于本章。6.1节把递归作为编写图搜索程序的一个重要工具进行了详细介绍，并以递归形式实现了第3章中的深度优先算法和回溯算法。模式导向搜索利用第2章给出的合一和推理规则简化了对逻辑推理空间搜索的实现。

在6.2节介绍的产生式系统是用于对求解问题方法建模和实现搜索算法的一种自然结构。这节介绍了使用产生式系统实现数据驱动搜索和目标驱动搜索的例子。实际上，产生式系统一直是AI程序设计的一个重要典范，起始于Newell和Simon及其同事在卡内基·梅隆大学的研究（Newell and Simon 1976，Klahr et al. 1987，Neches et al. 1987，Newell et al. 1989）。产生式系统还一直是支撑认知科学研究的一种重要结构（Newell and Simon 1972，Luger 1994；也可参考第16章）。

关于实现产生式系统（特别是基于OPS语言的）的参考文献有Lee Brownston等人（1985）编著的《Programming Expert Systems in OPS5》以及Donald Waterman和Frederick Hayes-Roth（1978）编著的《Pattern Directed Inference Systems》。关于现代的用C和Java语言实现的产生式系统，请参考CLIPS和JESS的网站。

在HEARSAY-Ⅱ的研究（Reddy 1976，Erman et al. 1980）中描述了黑板模型的早期成果。描述黑板模型后来发展的资料包括对HEARSAY-Ⅲ的研究（Lesser and Corkill 1983，Nii 1986a，Nii 1986b）及Robert Engelmore和Tony Morgan（1988）编著的《Blackboard Systems》。

对产生式系统、规划和黑板结构的研究依然是人工智能研究领域中的一个活跃部分。我们推荐有兴趣的读者查阅人工智能会议及人工智能国际联合会议美国协会的近期学报。Morgan Kaufmann和AAAI出版公司已经出版了一些其他的会议学报以及关于AI课题的文集。

6.5 习题

1. a）编写一个member-check算法，算法通过递归来判断一个给定元素是否是一个列表的成员。
   b）写一个算法以计算出一个列表中的元素个数。
   c）写一个算法以计算出一个列表中的原子个数。
   （原子和元素之间的区别是元素本身可以是一个列表。）
2. 写一个递归算法（使用open和closed列表）来实现宽度优先搜索。在实现宽度优先搜索时可以利用递归来省略open列表吗？解释原因。
3. 跟踪对图3-14所示状态空间进行递归深度优先搜索（这个版本不使用open列表）的执行过程。
4. 在古印度的一次茶庆典上，有三个参加者：一个老者、一个仆人和一个少年。他们要执行4项任务：烧火、上点心、倒茶和颂诗；这4项任务的重要性依次递减。在庆典开始时，由少年执行所有4项任务。每次传递一项任务给下一个人来执行，少年做完后轮到仆人，再轮到老者，直到庆典的最后，是老者在执行所有4项任务。任何人不可以做比他们已经执行任务的重要性更低的任务。产生一个移动序列把所

有4项任务从少年移交给老者。写一个递归算法来执行这个移动序列。

5. 使用6.2.2节中骑士周游问题的move和path的定义，跟踪pattern_search对以下目标的执行过程：
   a）path(1,9)
   b）path(1,5)
   c）path(7,6)

   当依次试验move谓词时，经常在搜索中出现循环。讨论这种情况下的循环探测和回溯。
6. 写出pattern_search（见6.1.2节）的宽度优先版本的伪代码。讨论这种算法的时间和空间效率。
7. 使用6.2.3节中的规则作为模型，写出8×8版本的完整骑士周游问题需要的8种移动规则。
8. 利用图6-3中的起始状态和目标状态，手工运行8格拼图游戏问题的产生式系统解：
   a）以目标驱动方式。
   b）以数据驱动方式。
9. 考虑第2、3和4章讨论的财务顾问问题。利用谓词演算作为表示语言：
   a）用产生式系统描述这个问题。
   b）回忆第3章中对这个问题的数据驱动解，生成这种解法的状态空间和各阶段的工作内存。
10. 改用目标驱动方法，解决问题9b。
11. 6.2.3节给出了几种通用的冲突消解策略：折射、最新性和特殊性。再提出两种这样的策略，并说明理由。
12. 提出两种适合用黑板结构求解的应用。简述为了实现每一种应用，应该如何组织黑板和知识源。

第三部分　捕获智能：AI 中的挑战

我们的忧患在很大程度上是由于我们总是使用昨天的工具做今天的工作……

——Marshall McLuhan

将大脑看作具体行为的控制器不是更有成果的想法吗？这个在思考角度上的微小变化对如何建立我们关心的科学产生了深远的影响。事实上，它需要我们彻底改变思考智能行为的方式，需要我们抛弃（从笛卡儿开始便普遍存在的）精神与肉体完全分离的思想；抛弃在感知、认知和行为之间存在明确分界线的思想；抛弃大脑在一个执行中心中进行高层推理的思想；最重要地是抛弃人为地把思考从肉体行为中分离出来的研究方法……

——Andy Clark，《Being There》（1997）

表示与智能

在 60 多年的 AI 历史中，表示问题（也就是如何最佳地捕获智能行为的关键特征以供计算机使用，或者说以供与人交流）一直是 AI 的一个经典主题。本部分首先回顾多年来一直影响 AI 研究团体的三种主要表示方法。第一个主题是 20 世纪 50 年代和 60 年代 Newell 和 Simon 在研究逻辑理论家（Newell and Simon 1956，1963a）程序时提出的，称为强方法问题求解。第二个主题是 20 世纪 70 和 80 年代非常普遍的弱方法问题求解，这种方法得到了早期专家系统设计者的拥护（参见第 8 章开篇引用的 Feigenbaum 的论述）。近年来，特别是在机器人领域和互联网领域（Brooks 1987，1989；Clark 1997），研究的焦点已转向分布式、具体化或基于主体的智能表示。下面对这三种表示智能的方法学逐一介绍。本部分的三章详细地介绍了这些方法。

在 20 世纪 50 年代后期和 60 年代初期，Alan Newell 和 Herbert Simon 写了几个计算机程序，验证启发式搜索可以产生智能行为的假设，他们与 J. C. Shaw 开发的逻辑理论家程序（Newell and Simon 1963a），利用 Whitehead and Russell（1950）的《数学原理》中的符号和公理，成功地证明了初级逻辑中的一些定理。作者将他们的研究目的描述为旨在理解：

> 有效求解问题的复杂过程（启发）。所以，我们感兴趣的不是保证能得到解的方法，而是需要大量计算的方法。举例来说，我们更希望理解数学家如何能证明定理，即使他在开始时并不知道如何去做，或者不知道能否成功。

在后来的通用问题求解器（GPS）中，Newell and Simon（1963b，1972）继续努力寻找智能问题求解的通用原则。GPS 可以求解形式化为状态空间搜索的问题；合法的问题求解步骤是修改状态表示的操作集合，搜索的方式和本书前面章节中讨论的算法一样。

GPS 使用手段－目的分析（means-ends analysis）——一种引来在多个备选状态变换操作中进行选择的通用启发——来引导对问题空间的搜索。手段－目的分析检查当前状态和目标状态间的语法差异，并选择一种可以降低差异的操作符。举例来说，假定 GPS 现在要证明两个逻辑表达式的等价性。如果当前状态包含一个 $\wedge$ 操作符，而目标状态不包含这样的操作符，那么手段－目的分析就会选择一种像德·摩根定律这样的转换，把 $\wedge$ 操作符从表达式中移去（14.1 节对此进行了更详细的描述，介绍了该项研究的一些例子）。

因为使用的启发仅仅分析状态的语法形式，所以研究者们希望 GPS 成为求解智能问题的通用体系结构，适用于所有领域的问题。像 GPS 这样仅使用基于语法的策略、但目标应用非常广泛的程序称为*弱方法问题求解器*。

不幸的是，似乎不存在一种单一的启发可以成功地求解所有领域的问题。通常，在求解每个问题的方法中都使用了有关具体情况的大量知识。医生能够诊断疾病是因为他们除了具有一般的问题求解能力外，还具有医学方面的广泛知识。建筑师可以设计房屋是因为他们懂建筑学。事实上，用在医疗诊断中的启发对办公建筑设计可能毫无用处。

弱方法和*强方法*形成了鲜明的对比，后者大量使用有关特定问题域的知识。考虑一个用来诊断和分析汽车故障的规则：

如果发动机不转，而且灯不亮；
那么是电池或电缆出了问题（0.8）。

这一启发——用“如果……那么……”短语表达的规则——搜索的焦点集中在汽车的电池/电缆子系统，排除了其他部件，修剪了搜索空间。注意：不同于手段－目的分析，这个启发使用了汽车如何工作的经验性知识，比如电池、车灯和发动机之间关系的知识。这个启发对车辆修理领域外的问题是没有用的。

强方法问题求解器不仅使用具体领域知识，而且通常需要大量这样的知识才能有效工作。例如，在诊断化油器故障时，电池损坏的启发是没多大用处的，而且对于其他领域来说是完全没用的。因此，设计基于知识程序的一个主要难题是如何获取和组织领域中的大量知识。问题域中的规则还可以包含一个尺度（比如上面的0.8），反映诊断者对这部分领域知识的信心。

在使用强方法时，程序设计者同时也对智能系统的特征做出了一些假定。Brian Smith（1985）将这些假定归纳为*知识表示假设*，该假设指出：

> 任何机械表达的智能过程都由具有以下特征的结构化因素构成：（a）外部观察者可以自然给出整个过程展现的知识的命题表示；（b）独立于这种外部语义属性，在产生表现知识的行为中起到了形式作用。

这一假设的第一个重要内涵是，假定知识是*通过命题表示*的，也就是显式地表示出问题中的知识，而且在外部观察者看来，这种表示是对知识的“自然”描述。第二个主要假定是系统中的行为，形式上可以看作是由知识库中的命题直接导致，而且这个行为应该与产生它的命题含义一致。

最后一种 AI 表示模式经常描述为*基于主体的、具体化的、或者涌现的*问题求解。一些致力于机器人、游戏设计和互联网等应用领域的学者对前面讲的两种模式提出了挑战，对需要集中式的知识库或者通用的推理表示模式提出了质疑，这些人包括 Brooks（1987，1989）、Agre and Chapman（1987）、Jennings（1995）、Wooldridge（2000）、Wooldridge 等（2006，2007）、Fatima 等（2004，2005，2006）以及 Vieira 等（2007）。他们把问题求解器设计成很多分布的主体，这些主体具有*情景化、自主、灵活*等特征。

根据这一观点，问题求解过程是分布式的，多个主体各自执行问题域中不同环境下的任务，例如互联网浏览器或安全主体的活动。因此，问题求解任务分解为多个部分，这些部分间很少或者没有总体协作。情景化的主体可以从它所处的特定环境中接收到传感器输入，并根据环境做出反应，而不必等待总控制器的指令。例如在交互式游戏中，主体都忙于特定的局部任务，比如抵御特定入侵或者发出报警，它们根本不需要了解整个问题域的总体情况。

很多情况下，主体需要在没有人类或某个全局控制过程的直接干预的情况下做出反应，从

这个意义上说主体是自主的。自主主体可以独立控制自己的动作和内部状态。一些主体系统甚至能够从经历中学习。主体的最后一个特征是它的灵活性，能够对所处局部环境的情况做出迅速反应。它们也具有前瞻性（proactive），因为它们能够对环境的变化预先做出准备。最后必须能够和问题域中的其他主体灵活合作，就任务、目标和处理方法进行沟通。

机器人领域的几个学者已经建成了基于主体的系统。麻省理工学院机器人研究实验室的 Rodney Brooks（1987，1989）已经设计出了一种称为*包容体系结构*（subsumption architecture）的系统，由一系列分层的有限状态自动机序列构成，每一层在自己的环境中活动，同时支持更高层的功能。卡内基·梅隆大学机器人研究实验室的 Manuela Veloso 等（2000）设计出了一队英式足球队员机器人主体，可以在英式足球锦标赛对抗环境中相互协作。

一些研究科学问题的哲学家，包括 Dan Dennett（1991，1995）和 Andy Clark（1997），认为这种情境化的具体表示方法是对人类智能的恰当描述。在本书后面的章节中介绍了更多的表示模式，包括连接（见第 11 章）和遗传（见第 12 章）方法。第 16 章中再次讨论了通用表示问题。

第 7 章详细分析了 AI 通信中使用的很多表示方法。我们首先介绍了早期的表示方法，包括*语义网*、*脚本*、*框架*和*对象*。我们从发展的角度介绍这些表示模式，目的是说明现代工具是如何从这些早期的 AI 研究中发展起来的。接下来给出了 John Sowa 的*概念图*（conceptual graph），这是自然语言理解中使用的一种表示。最后介绍了基于主体和情景的表示方法，包括 Rodney Brooks 的*包容结构*，并由此讨论了是否需要显式的中央知识库及通用目的控制器。

第 8 章讨论知识密集型系统，并分析与获取、形式化和调试知识库有关的多个问题。我们将介绍用于规则系统的不同推理表示模式，包括目标驱动和数据驱动推理。除了基于规则的推理系统外，还将介绍基于模型和基于案例的推理系统。这些系统的一种做法是显式表示问题域（比如电子电路）中的理论基础和功能，而另一种做法是建立一个累积的数据库，记录问题域中以前成功和失败的案例，用于求解将来的问题。最后，我们通过简单浏览*规划*（在像机器人控制这样的复杂领域中，组织知识来控制问题求解过程）来总结第 8 章。

第 9 章介绍了表示具有模糊性和（或）不确定性的推理问题的技术。在这些情况下，我们的目标是努力得到对模棱两可信息的最佳解释。这种类型的推理经常称为*反绎*（abductive）。首先介绍了非单调以及真值维持逻辑，并对传统谓词逻辑进行了扩展，以处理带有不确定性的情况。然而，很多有趣而且重要的问题求解方法无法在演绎逻辑框架下发挥最大的作用。因此我们介绍了一些其他的工具处理这样的情况，这些工具包括贝叶斯技术、Dempster-Shafer 方法以及斯坦福确定度代数。还将用一节来介绍模糊推理。我们用不确定情况下的随机方法学结束第三部分，介绍了贝叶斯信念网络、马尔可夫模型、隐马尔可夫模型和一些相关的方法。

第7章 知识表示

这部巨著乃至宇宙……都是用数学语言写的，其字符是三角、圆和其他的几何图形。离开这些图形，人类根本无法理解其中的只言片语；离开这些图形，就会陷入漆黑的迷宫……

——伽利略（Galileo Galilei 1638）

因为任何生物体都无法一一处理无限的多样性，所以所有生物体的一个基本功能就是把环境分解为一个个类，通过分类可以把不完全相同的刺激看成是等价的刺激……

——Eleanor Rosch，《分类原理》（1978）

我们总可以谈论两个世界，不妨称其为“物理世界”和“感知世界”，或者用你喜欢的其他称呼。前者用来处理定量而且具有形式结构的问题，后者用来描述构成“世界”的各种特征。所有人都有自己独特的精神世界，有自己的心路历程和美好蓝图，而且对于我们中的大多数人来说，根本不需要把这些和神经学“关联”起来。

——Oliver Sacks，《错把太太当帽子的人》（1987）

7.0 知识表示问题

智能问题求解中的信息表示是AI中的一个核心问题，在这一领域中始终存在着很多重要又难以逾越的挑战。在7.1节中简要地回顾了早期表示研究的历史，讨论的主题包括语义网、概念依赖性、脚本和框架。7.2节中介绍了一种用于自然语言程序中的更新的表示方法—John Sowa的概念图。在7.3节中对是否需要创建集中式的知识库和显式的表示模式提出了质疑。Brooks采取的另一种方法是用于机器人的包容体系结构。7.4节介绍了替代集中控制的另一种方案——主体。在后面的章节中，把对表示问题的讨论扩展到了一些更广泛的领域，包括随机性（见9.3节和第13章）、连接（见第11章）以及遗传/涌现（见第12章）。

下面先介绍一种历史较久的观点，以此开始关于知识表示问题的讨论。这种观点把知识库描述为“问题域中的对象和关系”与“程序中的计算对象和关系”间的映射（Bobrow 1975）。知识库中的推理结果应该和现实世界中的动作或观察结果对应。知识表示语言就是计算对象、关系和程序员可用推理间的媒介。

很多关于知识组织的一般原理适用于不同的领域，而且可以由表示语言直接支持。举例来说，类层次既出现在科学分类系统中，又出现在日常分类系统中。如何才能给出一种表示类层次的通用机制呢？如何表示定义及例外？智能系统何时使用默认假定弥补残缺信息，如果这些假定证明是错误的，那么该如何调整推理？怎样表示时间才是最佳的？以及怎样表示因果关系？不确定性？要想在建立智能系统方面取得突破，就必须发现新的知识组织原理并用更高层的表示工具支持这一原理。

把表示模式（scheme）和实现它的媒介（medium）区分开是有意义的。这类似于把数据结构和程序设计语言区分开来。程序设计语言是实现的媒介；数据结构是表示模式。一般来讲，知识表示语言比谓词演算或者程序设计语言具有更强的约束性。这些约束就是表示各类知识的显式结构。实现这些约束的媒介可能是Prolog、LISP或是像C ++或Java这样的语言。

以上讨论简述了对AI表示模式的传统观点，该观点经常包括一个全局性的语言结构的知识库，组成知识库的语言结构是以对“真实世界”的“预先偏置”为基础的，是静态的。很多近

期的研究成果已经对这种传统方法提出了质疑，这包括机器人领域的最新研究（Brooks 1991a, Lewis and Luger 2000）、情景认知（Agre and Chapman 1987, Lakoff and Johnson 1999）、基于主体的问题求解（Jennings et al. 1998; Wooldridge 2000; Fatima et al. 2005, 2006; Vieira et al. 2007）以及哲学方面的研究（Clark 1997）。这些问题域需要分布式知识、一个本身可以用做部分知识结构的世界、使用部分信息推理的能力以及随着问题域的变化而进化的表示。7.3 节和 7.4 节中介绍了这些方法。

7.1 AI 表示模式的简要历史

7.1.1 语义关联理论

逻辑表示源于哲学家和数学家对描述正确推理原则的努力。逻辑所关心的问题是开发出具有可靠且完备的推理规则的形式表示语言。因此，谓词演算语义强调了对合式表达式的保持真值运算。另一种研究路线起源于心理学家和语言学家为刻画人类理解的本质所做的努力。这类研究不太关心能否建立出正确推理的规则，更关心如何描述人类用来获取、关联和使用所处世界知识的实际方式。已经证明这种方法对于自然语言理解和常识推理这样的应用领域特别有效。

在把常识推理映射到形式逻辑时会产生很多问题。例如，很多人认为操作符 $\wedge$ 和 $\rightarrow$ 对应于英语中的“or”和“if…then…”。然而，在逻辑中这些操作符关心的是真实性，忽略了英语“if …then…”中前提和结论之间的特定关系（经常是一种相关的关系，而不是因果关系）这一事实。例如，可以把这句话“如果一只鸟是北美红雀（cardinal），那么它就是红色的”（把北美红雀和红色联系起来）写成谓词演算的形式：

$$\forall X(\text{cardinal}(X) \rightarrow \text{red}(X))$$

可以通过第 2 章介绍的一系列保持真值运算修改这个表达式，使其变成如下的等价形式：

$$\forall X(\neg\ \text{red}(X) \rightarrow \neg\ \text{cardinal}(X))$$

这两个表达式在逻辑上是等价的，也就是当且仅当第一个表达式为真时第二个表达式为真。然而，在这种情况下真值等价性却不适用。假设我们要寻找证实以上两个陈述真实性的证据，那么“本书页不是红色的，而且也不是北美红雀”这一事实可以作为证明第二个表达式为真的证据。因为两个表示式是逻辑等价的，所以可以推出它也是证明第一个表达式为真的证据。便得到了这一结论：“本书页的白色性”可以作为证明“北美红雀为红色”的证据。

这个推理路线是没有意义的，而且非常可笑。导致这种不相容性的原因是逻辑蕴涵仅表达一种操作数的真值之间的关系，而英语句子蕴涵了“一个类中的成员”和“类中成员拥有这个类的属性”间的正相关性。事实上，鸟的遗传性导致它具有某种颜色。这种关系在第二个版本的表达式中丢失了。尽管“这页纸不是红色”这一事实和两个句子的真值是一致的，但是这和鸟的颜色这一因果特征没有关系。

传统的来源于哲学中经验主义的联想论者（associationist）通过一个对象与其他对象间的关联网络来定义这个对象的含义。在联想论者看来，当人感受或者思考一个对象时，这种感知首先映射为一个概念。这个概念是我们对世界的整个知识的一部分，并且通过合适的关系和其他概念相连。这些关系形成一种对于对象（比如雪）行为和属性的理解。例如，根据经验，我们会把“雪”这个概念和像“寒冷、白色、雪人、滑和冰”这样的其他概念关联起来。我们对雪的理解以及像“雪是白色的”这样的陈述都来自于关联网络。

心理学上的证据表明人类除了具有关联概念的能力外，还可以分层地组织知识，把信息放

在最合适的分类层次上。Collins 和 Quillian（1969）用语义网（见图 7-1）对人类的信息存储和管理进行了建模。这种层次结构是从对接受实验者（受试者）的测试结果中得到的。在实验中，受试者要回答很多关于鸟属性的问题，比如“金丝雀是鸟吗?”“金丝雀会唱歌吗?”以及“金丝雀会飞吗?”

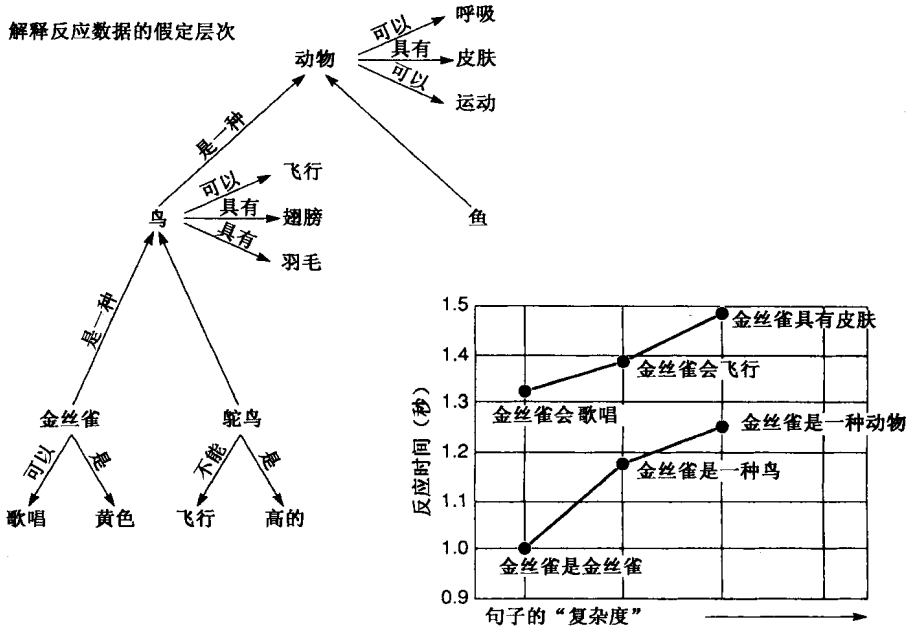


图 7-1　语义网

注：这是 Collins 和 Quillian 开发的语义网，用来研究人类的信息存储和反应时间（Harmon and King 1985）

这些问题的答案就像看起来那么简单，反应时间研究指出受试者回答“金丝雀会飞吗?”比回答“金丝雀会唱歌吗?”所用的时间长。Collins 和 Quillian 通过论证人类存储的是最抽象层信息来解释反应时间上的差异。他们认为，人类并不是努力去回忆金丝雀会飞、知更鸟会飞、燕子会飞以及对应每一种鸟的情况的信息；而是记住金丝雀是一种鸟，而且鸟（通常）具有飞行的属性。像进食、呼吸和运动这样更一般的特征存储在“动物”层，因此要回忆金丝雀能否呼吸比回忆它是否会飞行要用更长的时间。这是因为人类必须向上追溯更多的记忆结构层次来取得答案。

回忆最快的是这种鸟特有的属性，比如说，它可以唱歌或者它是黄色的。例外处理似乎也是在最特别的层次上完成。当询问受试者鸵鸟是否会飞时，产生回答的速度要比询问鸵鸟是否会呼吸快。因此要得到这种例外信息似乎并不是经过“鸵鸟→鸟→动物”这样的层次，例外信息是直接存储的，在继承系统中使用了这种知识组织方法。

继承系统让我们可以在最高抽象层存储信息，这降低了知识库的大小并有助于防止更新上的不一致。举例来说，如果我们在建立一个关于鸟的知识库，那么可以定义一个通用的类“鸟”，包含所有鸟都具有的特征，比如飞行或有羽毛。然后允许某一特定的鸟类继承这些特征。这样只需要定义这些特征一次，从而降低了知识库的大小。继承还可以帮助我们在加入新的类和个体时维护知识库的一致性。假定我们现在要向现有知识库中加入知更鸟这个类。当我们断言“知更鸟”是“会唱歌的鸟”的子类时，那么知更鸟便继承了“会唱歌的鸟”和“鸟”的公共属性。这不取决于程序员是否记住或忘记加入这个信息。

图提供了一种利用弧和结点显式表示关系的手段，已经证明图是形式化联想理论的理想工

具。语义网把知识表示为一种图，结点对应事实或者概念，弧对应概念之间的关系和关联。通常不论是结点还是弧线都带标签。例如，图7-2中的语义网定义了“雪”和“冰”的属性。可以（结合适当的推理工具）利用这个网络回答很多关于雪、冰和雪人的问题。这些推理是通过跟踪有关概念间的连接来完成的。语义网还实现了继承；比如有霜的雪人继承了雪人的所有属性。

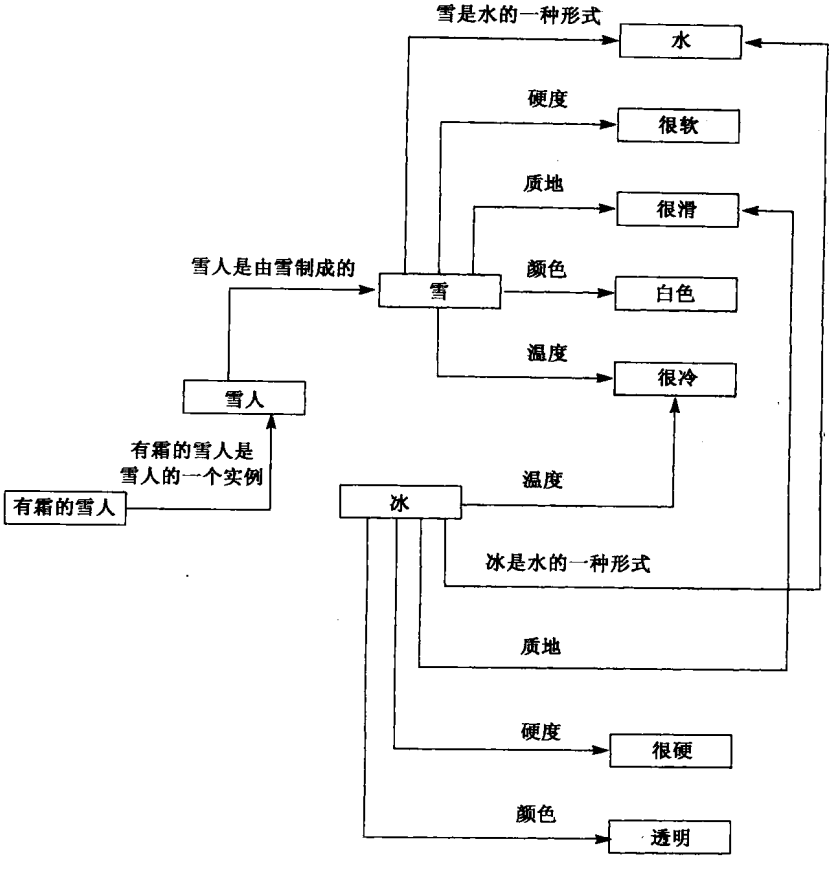


图7-2 “雪”和“冰”属性的网络表示

“语义网”这一术语涵盖了一组基于图的表示。这些表示在以下两方面有所不同：允许结点和连接使用的名字；在这些结构上可以进行的推理。然而，所有的网络表示语言都具有一些共同的假定和目标；对网络表示历史的讨论阐述了这些特征。在7.2节中，我们将分析概念图（Sowa 1984），它是一种更新的网络表示，集成了很多这种思想。

7.1.2 语义网的早期研究

网络表示的历史几乎和逻辑的历史一样长。希腊哲学家Porphyry发明了以树为基础的层次结构来描述亚里士多德的分类（Porphyry 1887），特点是根在顶部。Frege为逻辑表达式开发了一种树表示法。对现代语义网有直接影响的最早研究也许是Charles S. Peirce的存在图系统，开发于19世纪（Roberts 1973）。Peirce的理论具有一阶谓词演算的所有表达能力，是以公理为基础的形式推理规则。

心理学从很久以前便开始用图来表示概念和关联结构。Selz（1913，1922）首先进行了这一研究，用图来表示概念层次和属性继承。他还开发了一种图解预测理论，影响了AI在框架和图

示方面的研究。Anderson、Norman、Rumelhart 等人已经用网络对人类记忆和智能行为建模（Anderson and Bower 1973，Norman et al. 1975）。

网络表示的大多数研究工作都是在自然语言理解这一领域中完成的。通用情况下的语言理解需要对常识的理解，即物理对象的行为方式、和人之间的交互以及人类机构的组织方式等。一个自然语言程序必须理解意图、信念、假设推理、规划及目标等概念。因为需要非常多的知识，所以自然语言理解始终是推动知识表示研究的力量。

语义网第一次在计算机上实现是在 20 世纪 60 年代初期用于开发机器翻译系统。Masterman（1961）定义了 100 种基本概念类型，并使用这一集合定义了包含 15 000 个概念的词典。Wilks（1972）继续了 Masterman 的工作。Shapiro（1971）的 MIND 程序第一个实现了基于命题演算的语义网。其他探索网络表示的早期 AI 工作者还包括 Ceccato（1961）、Raphael（1968）、Reitman（1965）及 Simmons（1966）。

Quillian 在 20 世纪 60 年代后期编写了一个很有影响的程序，演示了早期语义网的很多特征（Quillian 1967）。这个程序以几乎和字典相同的方式定义英语单词：一个单词是通过其他单词定义的，定义中的各个部分也是按这种方式定义的。不是使用基本公理来定义单词，每个定义只是以一种没有固定结构的而且可能循环的方式引向其他的定义。在查一个单词时，遍历这个“网络”，直到我们已经满意于对原来单词的解释。

在 Quillian 的网络中每个结点对应于一个单词概念，并带有相关的连接指向组成其定义的其他单词。他把知识库组织为多个平面（plane），每个平面是定义一个单词的图。图 7-3 引自 Quillian（1967）的论文，图中列出了三个平面，表示了单词“plant”的三种定义：一种活着的组织（plant 1）；人们工作的地方（plant 2）；把种子植入地下的动作（plant 3）。

plant:1) 不是动物的活着的结构，经常带有叶子，从空气、水和土地中吸取养分。
2) 工业生产中使用的设备。
3) 把种子、植物等植入地下让它生长。

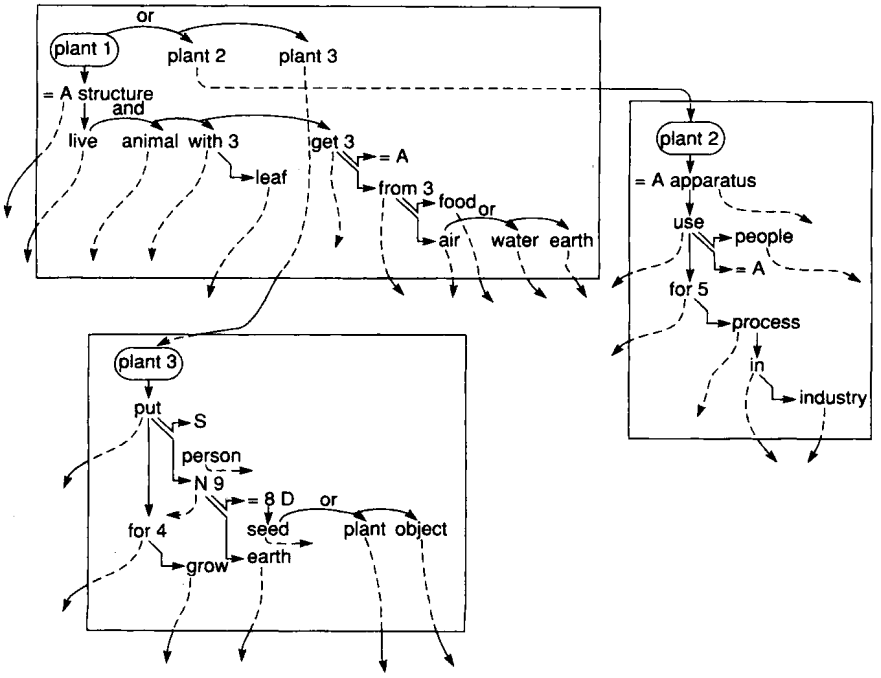


图 7-3　表示单词“plant”三种含义的三个平面（Quillian 1967）

Quillian 的程序使用这个知识库来寻找英语单词对之间的关系。给定两个单词，它就以宽度优先的方式从每个单词开始向外搜索这个图，目的是寻找一个共同的概念——即相交结点（intersection node）。到这个结点的路径代表了这两个单词概念之间的关系。例如，图 7-4（引自同一论文）显示了 cry 和 comfort 之间的相交路径（intersection path）。

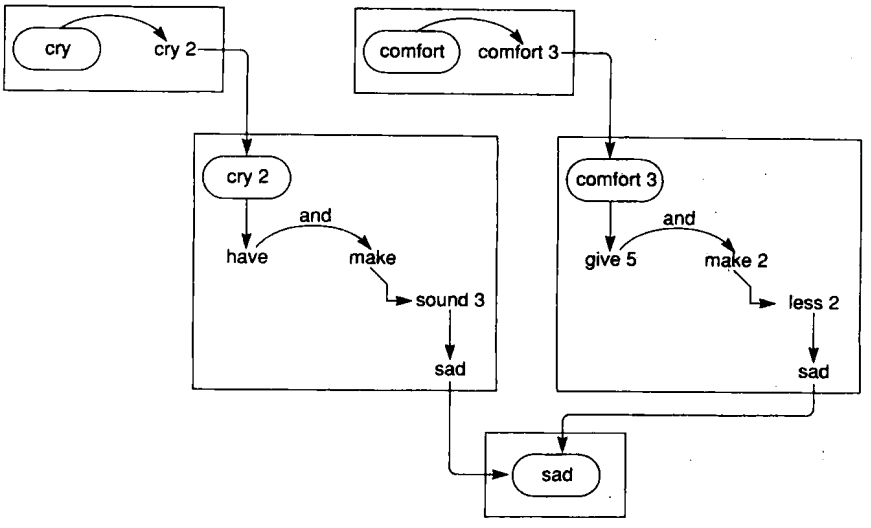


图 7-4 “cry”和“comfort”之间的相交路径（Quillian 1967）

利用这个相交路径，该程序能够总结出：

> 哭（cry 2）是产生悲伤声音的其他行为之一。要想舒适（comfort 3）可以做（make 2）一些不太悲伤的事（Quillian 1967）。

结果中的数字代表程序选择的是单词的多个不同语义中的哪一个。

Quillian（1967）提出这种面向语义的方法，可以提供具有如下能力的自然语言理解系统：

1）通过建立相交结点判断一段英语文本的含义。

2）选择多义词在句子中的语义，方法是寻找到达句中其他单词的相交路径，选取具有最短相交路径的语义。例如，对于“Tom went home to water his new plant”中的“plant”，可以根据单词概念“water”和“plant”相交选出它的含义。

3）以查询中的单词概念和系统中概念之间的关联为基础回答各种查询。

尽管这一研究以及其他早期研究确立了图在关联含义建模方面的地位，但是其形式方法的绝对通用性导致其功能是有限的。大多数知识都是按特定关系组织的，这些特定关系可以是对象/属性、类/子类、主体/动词/目标。

7.1.3 网络关系的标准化

就其本身而言，关系的图表示法比谓词演算没有什么进步；它仅是对象之间关系的另一种表示法。实际上，语义网处理系统（Semantic Net Processing System，SNePS）（Shapiro 1979，Shapiro et al. 2006）是一个与一阶谓词演算具有同样能力的定理证明器。网络表示的强大之处来源于对连接和相关推理规则的定义，比如继承定义了一种特殊的推理关系。

尽管 Quillian 的早期研究确立了语义网的多数重要特征，比如带有标签的弧和连接、层次继承及沿相关连接进行的推理。但是已经证明，这种方法在处理很多领域中的复杂问题时，其能力是很有限的。导致这种有限性的一个主要原因是缺乏捕获知识的深层语义特征的关系（连接）。

大多数连接代表的都是结点之间的普通关联，没有提供一种切实的机制来建立语义关系。使用谓词演算捕获语义也会遇到这个问题。尽管这种形式方法具有很高的表达力，可以表示几乎任何种类的知识，但是它过于没有约束了，把建立合适的事实和规则集合的压力都放到了程序员身上。

Quillian 的网络表示研究的大多数后续工作都集中在定义更丰富的连接标签（关系）集合上，目的是可以更完全地对自然语言的语义建模。通过把自然语言的基本语义关系实现为形式方法的一部分，而不是作为领域知识的一部分要由系统建立者来加入，知识库所需的手工劳动更少了，而且实现了更好的通用性和一致性。

Brachman（1979）曾经指出：

> 这里的关键问题是语义网语言原语（primitive）的孤立。网络语言的原语是已经预先编入解释器的内容，它们通常不是用网络语言本身来表示的。

Simmons（1973）把研究的焦点集中到英语动词的格结构（case structure）上，以解决对标准关系的需要。在这种以 Fillmore（1968）早期的研究为基础的面向动词方法中，连接定义了名词和名词短语在句子动作中的角色。格关系包括主体、对象、工具、位置以及时间。一个句子表示为一个动词结点，并用不同的格连接指向参与动作的其他结点。这一结构称为案例框架（case frame）。在分析一个句子时，程序先找到动词，并从知识库中提取出这个动词的案例框架。然后再把主体、对象等绑定到案例框架的适当结点上。利用这种方法，可以把句子“Sarah fixed the chair with glue”表示为图 7-5 所示的网络。

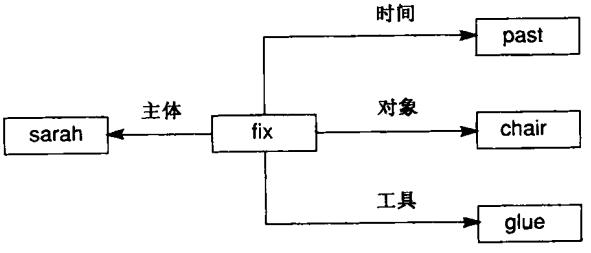


图 7-5　句子“Sarah fixed the chair with glue”的案例框架表示

这样，表示语言本身捕获了自然语言的大多数深层结构，比如动词与其主语之间的关系（主体关系）以及动词和对象之间的关系。关于英语语言格结构的知识是网络形式方法本身的一部分。在分析一个句子时，这些内建的关系指出 Sarah 是执行修理的人，并使用粘合剂把椅子结合起来。注意，这些语言学关系是以一种独立于实际语句乃至用表达语句的语言的方式来存储的。Norman（1972）以及 Rumelhart 等（1972，1973）也提出了在网络语言中使用类似的方法。

有很多研究者试图把连接名进一步标准化（Masterman 1961，Wilks 1972，Schank and Colby 1973，Schank and Nash-Webber 1975）。每一种努力都致力于建立起一个完整的原语集合，然后可以用这个集合以统一方式表示自然语言表达的语义结构。这样做的目的是有助于使用语言结构进行推理，并且独立于语言个体或短语表达的特质。

或许最雄心勃勃的尝试是 Roger Schank 的概念依赖（conceptual dependency）理论（Schank and Rieger 1974），目标是为自然语言的深层语义结构建立形式模型。概念依赖理论提供了一个包含 4 种原子概念的集合，根据这 4 种原子来建立语义世界。这些概念是平等而且独立的。它们是：

ACT　动作
PP　对象（picture producers）
AA　动作的修饰语（action aiders）

PA 对象的修饰语（picture aiders）

举例来说，该理论假定所有动作都可以简化为一个或多个原子 ACT。下面列出的这些原语作为基本的动作部件，通过修改和组合这些部件可以形成更多的特定动词。

ATRANS 转移一种关系（give）
PTRANS 转移一个对象的物理位置（go）
PROPEL 对一个对象施加力（push）
MOVE 拥有者移动身体部件（kick）
GRASP 行动者抓取一个对象（grasp）
INGEST 动物摄取一个对象（eat）
EXPEL 从一个动物的体内排出（cry）
MTRANS 转移精神信息（tell）
MBUILD 产生新的精神信息（decide）
CONC 概念化或者思考一个想法（think）
SPEAK 产生声音（say）
ATTEND 集中感觉器官（listen）

这些原语用来定义概念依赖关系（conceptual dependency relationship），概念依赖关系描述了各种语义结构，比如格关系或者对象和值之间的关联关系。概念依赖关系是概念化的句法规则（conceptual syntax rule），并且组成了一种包含丰富语义关系的语法。可以用这些关系来建立英语语句的内部表示。图 7-6 给出了基本概念依赖关系的列表（Schank and Rieger 1974）。这些依赖性捕捉了，即它们的构造者感知到了，自然语言的基本语义结构。举例来说，图 7-6 中的第一种概念依赖关系描述了主语和动词之间的关系，第三种描述了动词 - 宾语关系。组合这两条关系便可以表示一种简单的动宾语句，比如“John throws the ball”（见图 7-7）。

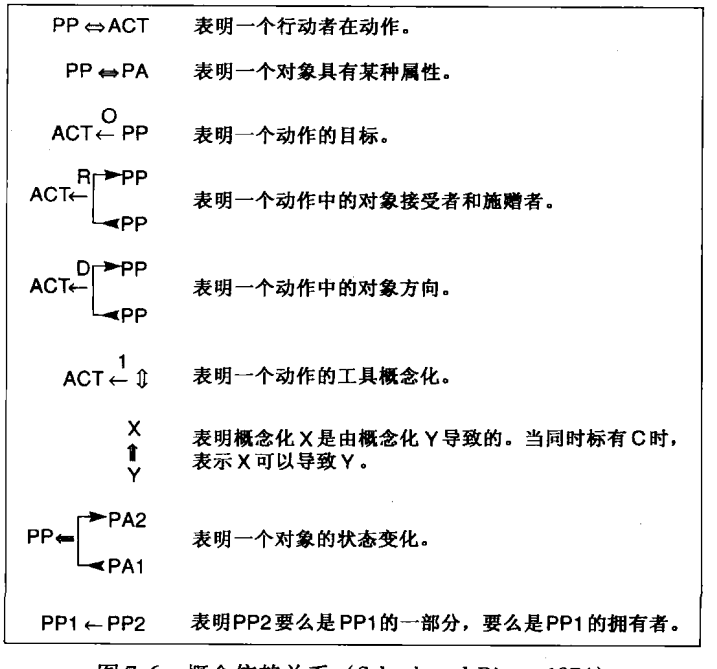


图 7-6 概念依赖关系（Schank and Rieger 1974）

最后可以把时态信息加入到概念集合中。Schank 提供了一个列表，其中包含了关系的附件，也就是修饰语。下面是这个列表的一部分：



图 7-7　语句“John throws the ball”的概念依赖性表示

p　过去
f　将来
t　转换
k　进行
t_s　开始转换
?　疑问
t_f　完成转换
c　条件
/　反义
nil　现在
delta?不受时间影响的

这些关系是这一理论的第一级结构元素，它们是最简单的语义关系，由其可以派生出更复杂的结构。图 7-8 中给出了一些更复杂的例子，这些例子说明了如何利用这些基本概念依赖关系来表示简单英语语句的含义。

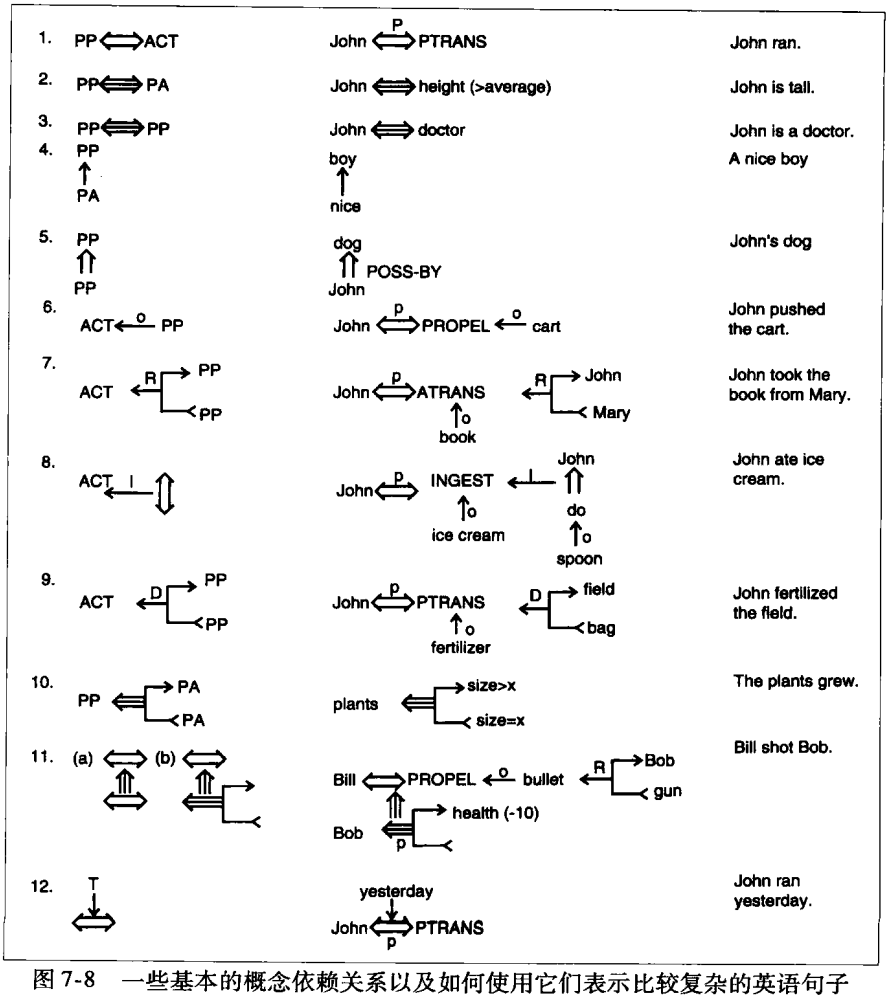


图 7-8　一些基本的概念依赖关系以及如何使用它们表示比较复杂的英语句子
[摘自 Schank 和 Colby (1973)]

基于这些原语，可以把这句英语“John ate the egg”表示成图7-9，其中的符号含义如下：

←	指出依赖性的方向
⇔	表明“主体－动词”关系
p	指出是过去时态
INGEST	是该理论中的一个原子动作
O	对象关系
D	指出动作中的对象方向

下面再举一个例子来说明利用概念依赖性可以建立的结构。图7-10表示的是“John prevented Mary from giving a book to Bill”这句话的概念依赖图。这个例子的有趣之处在于它演示了如何表示因果关系。

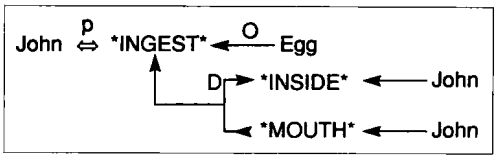


图7-9 表示“John ate the egg”的概念依赖图（Schank and Rieger 1974）

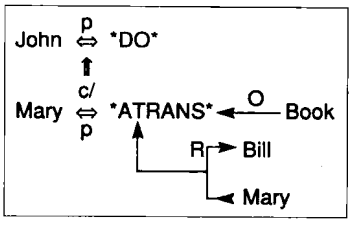


图7-10 句子“John prevented Mary from giving a book to Bill”的概念依赖表示（Schank and Rieger 1974）

概念依赖理论产生了很多重要的影响。首先，它为自然语言语义提供了一套形式理论，这大大减少了模棱两可的问题。其次，它为语句的含义提供了一种*范式*（canonical form），直接捕获了自然语言的大部分语义。也就是说，具有同一含义的所有语句会在内部表示为*语法相同*（不仅是语义等价）的图，这种正规表示简化了理解所需的推理。例如，我们可以通过匹配简单的概念依赖图来说明两个语句指的是同一件事；而没有提供范式的表示可能就需要在不同结构的图上进行大量的运算。

然而，是否可以写出一个把语句简化为范式的可靠程序却是值得怀疑的。正如Woods（1985）等人指出的，可以证实把独异点（monoid）（一种比自然语言简单的代数群）简化为范式，在计算来说是不可行的。此外，根本没有证据表明人类是按任何范式来存储知识的。

对这种观点的其他批评意见是对要把所有语句都简化成底层原语所需的计算代价不满意。此外，这些原语还不足以表示出自然语言中一些很重要的微妙概念。举例来说，图7-8中第二个语句的“tall”的表示，在表达这个词的二义性方面就没有模糊逻辑系统做的那么细致（参见Zadeh 1983以及9.2.2节）。

不过，没有人怀疑概念依赖理论还一直在广泛地研究和理解。Schank领导的研究已经持续了十几年，他们一直致力于如何提炼和扩展这一模型。概念依赖模型的重要扩充包括*脚本*和*内存组织包*（MOP）。脚本方面的研究分析了知识在内存中的组织，以及这种组织在推理中的作用（见7.1.4节）。MOP为基于案例的推理程序的设计提供了支持，见8.3节。总之，概念依赖理论是一套完整的自然语言语义模型，具有广泛的适用性。

7.1.4 脚本

自然语言理解程序即使要理解非常简单的会话，也需要使用非常大量的背景知识（见15.0节）。有证据表明人类是把知识组织成与各种典型情况相对应的结构（Bartlett 1932）。如果在阅

读一篇关于饭店、篮球或者警探的故事，那么我们会以一种和饭店、篮球或警探一致的方式来解决文中的二义性。如果故事的主题发生了意想不到的改变，那么有证据表明阅读会发生短暂的停顿，这被认为是在改变知识结构。一篇组织或者结构很差的故事是很难理解的，这可能就是因为我们不能很容易地把它和现有的任何知识结构拟合起来。当对话的主题突然改变时，也可能存在理解上的错误，这是因为我们混淆了解决对话中的代词所指或其他歧义应该使用的上下文。

脚本（script）是一种结构化的表示，用来描述特定上下文中固定不变的事件序列。脚本最先是由 Schank 和他的研究小组设计的（Schank 和 Abelson 1977），用来作为一种把概念依赖结构组织为典型情况描述的手段。自然语言理解系统使用脚本来根据系统要理解的情况组织知识库。

大多数成年人在饭店中都不会感到任何不自在（因为，作为顾客，他们知道想要什么以及如何去做）。他们在饭店入口处受到接待，或者通过标志继续向前找到桌子。如果菜单没在桌上，服务员也没有送过来，那么顾客会向服务员要菜单。也就是说，顾客理解整个流程：点菜、食用、付账，然后离开。

事实上，饭店脚本和其他的进餐脚本大不相同，比如“快餐”模式或者“正规的家庭餐”。在快餐模式中，顾客进来、排队等待点单、付款（在食用之前）、等待装有所点食品的餐盘、接过餐盘并找一张干净的桌子，等等。这是两个不同的套路化的事件序列，每个都有一个潜在的脚本。

脚本由以下几部分组成：

进入条件（entry condition）也就是要调用这个脚本必须满足的世界描述。在前面的示例脚本中，进入条件包括一家营业的饭店和一个有钱的饥饿顾客。

结果（result）也就是脚本一旦终止就成立的事实。例如，顾客吃饱了同时钱少了，饭店老板的钱增多了。

道具（prop）也就是支持脚本内容的各种“东西”。这可能包括桌子、服务员以及菜单。道具集合支持合理的默认假定：假定饭店拥有桌子和椅子，除非特别说明。

角色任务（role）也就是各个参与者所执行的动作。服务员拿菜单、上菜以及拿账单。顾客点菜、食用以及付账。

场次（scene）。Schank 把脚本分解成一系列场次，每一场次呈现一段脚本。在饭店中有进入、点菜、食用等场次。

脚本的要素——语义含义的基本“片段”——是用概念依赖关系来表示的。这些要素放入一种类似框架的结构，它们代表了一个含义序列，也就是一个事件序列。图 7-11 给出了摘自这项研究的饭店脚本。

当程序读一小段关于饭店的故事时，它可以把故事解析为内部的概念依赖表示。因为这种内部描述中的关键概念符合这段脚本的进入条件，所以这个程序就把故事中提到的人和事绑定到脚本中提到的角色和道具上。这样便得到了一种对故事内容的扩充表示，利用脚本和默认假定填补了故事中残缺的信息。然后这个程序便可以通过引用脚本来回答有关这个故事的问题了。这种脚本使我们可以在适当的时候应用默认假定，这对自然语言理解来说是至关重要的。例如：

例 7.1.1

昨天晚上，约翰去饭店吃饭，他点了 1 份牛排。付账时发现没有带钱，于是匆忙赶回家，当时天已经开始下雨了。

利用这一脚本，系统可以正确地回答很多问题，比如：约翰昨晚吃晚餐了吗（故事中仅仅隐含了这一点）？约翰使用现金或者信用卡了吗？约翰如何拿到菜单的？约翰买什么了？

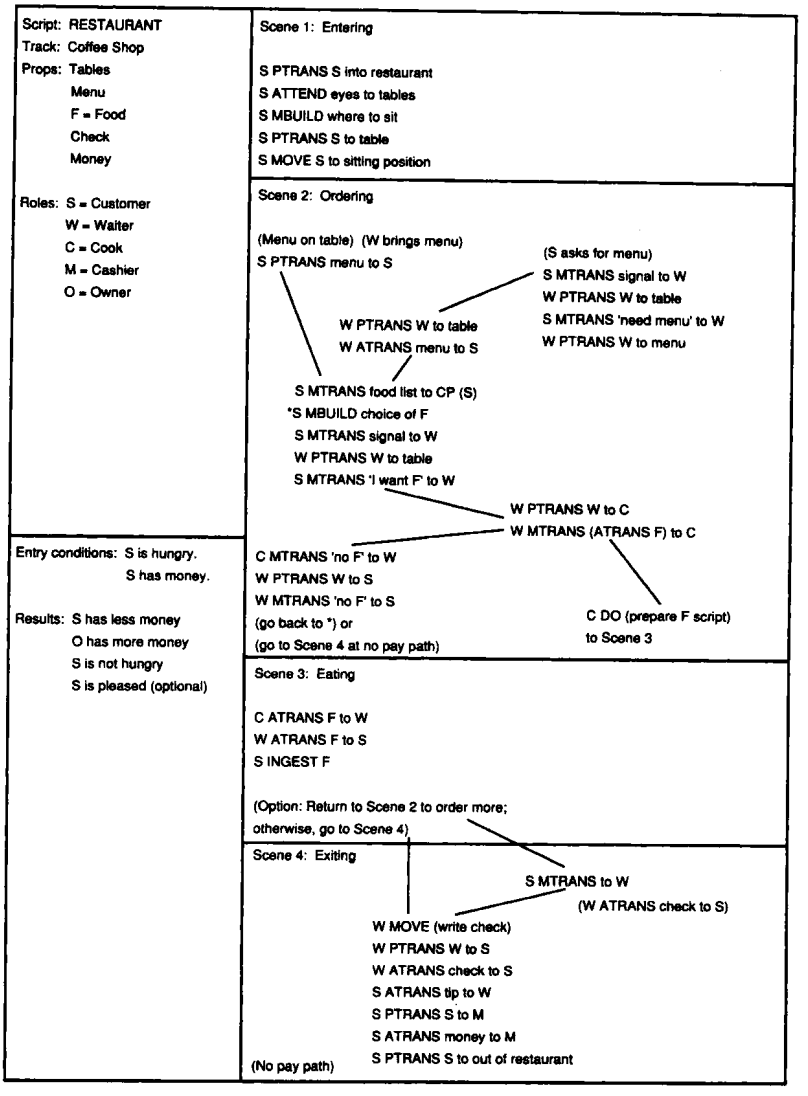


图 7-11 饭店脚本（Schank and Abelson 1977）

例 7.1.2

Amy Sue 到外面吃午饭。她坐到一张桌子旁，然后叫来女服务员，女服务员递给她菜单。她点了一份三明治。

关于这个故事，可以提出的合理问题包括：为什么服务员递给 Amy Sue 菜单？Amy Sue 当时是在饭店中吗（本例并没有说 Amy Sue 在饭店中）？谁付的账？点三明治的“她”是谁？最后一个问题是有难度的。文中最后一次明确提到的女性是服务员，但这是错误的代词指代。脚本的角色任务可以帮助我们解决这种指代问题以及其他模棱两可问题。

脚本还可以用来解释意外结果或者是脚本行为中的中断。因此，在图 7-11 的第 2 场中，有一个关于为顾客上了食物（“food”）或者是没有上食物（“no food”）的选择点。这样便可以理解下面的例子了。

例 7.1.3

凯特去饭店。女服务员介绍她到一张桌子边，点了一份三明治。她坐在那里等了很久。最

后，她等得太焦急便离开了。

利用饭店脚本可以回答的关于这个故事的问题包括：坐在那里等的“她”是谁？她为什么等？等得焦急了离开的“她”是谁？她为什么焦急？注意还有一些问题这个脚本没法回答，比如为什么当服务员没有迅速来时人们会焦急？和任何知识库系统一样，脚本需要知识工程师正确预测所需的知识。

脚本，像框架和其他结构化表示一样，都受制于特定的问题，包括脚本匹配问题和暗示问题。考虑例 7.1.4，它还调用了饭店或音乐会脚本。如何做出选择是很关键的，因为“bill”既可以指饭店的账单，也可以指音乐会的节目单。

例 7.1.4

约翰在听音乐会的路上顺便到了他最喜欢的一家饭店。他对账单/节目单（bill）很满意，因为他喜爱莫扎特。

因为选取脚本通常是基于“关键字”匹配的，所以很多时候很难判断应该使用两个或更多个可能脚本中的哪一个。目前的所有算法都不能保证做出正确的选择，从这个意义上来说，脚本匹配是一个难度很大的问题。它需要关于世界组织的启发性知识，甚至可能需要一些以往知识，脚本仅仅有助于组织这一知识。

暗示问题也是很难的，因为事先不可能知道某个可能事件会打断脚本。例如：

例 7.1.5

梅利莎正在她最喜欢的饭店吃饭，这时一大片石膏从天花板上坠落到她的椰枣上。

问题：梅利莎在吃椰枣色拉吗？梅利莎的椰枣涂上石膏了吗？她接下来做了什么？正如这个例子所说的，结构化表示可能不够灵活。推理可能被锁定到某个单一脚本，即使这个脚本可能是不合适的。

内存组织包（MOP）是一种用来处理脚本不灵活问题的策略，它把知识表示成较小的部分——MOP，然后再把这些包与规则一起动态组合成适合当前情况的表示模式（Schank 1982）。知识在内存中的组织对基于案例的推理的实现特别重要，因为在案例推理中，问题求解器必须高效地从内存中检索有关的以前问题的解（Kolodner 1988a，见 8.3 节）。

组织和检索知识的问题是很难的，而且对于语义建模来说这一问题是固有的。Eugene Charniak（1972）论述了即使要理解简单的童话，也需要大量的知识。考虑一句生日 PARTY 上的话：“玛丽在过生日时收到两只风筝，所以她把一个退还给商店。”我们必须知道在 PARTY 上送礼物的传统；还必须知道风筝是什么以及为什么玛丽可能不需要两个；我们还必须知道商店是什么，以及它们的退换制度。尽管存在这些问题，利用脚本和其他语义表示的程序可以在有限的领域中理解自然语言。这项工作的一个例子是翻译电报消息的程序。利用关于自然灾害、政变或者其他套路化故事的脚本，这些程序已经在这一有限却很实际的领域中取得了很大的成功（Schank and Riesbeck 1981）。

7.1.5　框架

还有一种在很多方面都和脚本很相似的表示模式，称为框架（frame），其目的是在显式组织的数据结构中捕获问题域中隐含的信息连接。这种表示支持把知识组织成更复杂的单元，以反映问题域中对象的组织方式。

Minsky 在 1975 年的一篇论文中介绍了框架：

> 下面是框架理论的精髓：当一个人遇到新的情况（或其看待问题的观点发生实质性变化）

时，他会从记忆中选择一种结构，即“框架”。这是一种记忆下来的轮廓，按照需要改变其细节就可以用其拟合真实情况（Minsky 1975）。

根据 Minsky 的说法，可以把框架看成是一种静态的数据结构，用来表示熟知的典型情况。我们对世界的认识是按照像框架一样的结构组织的。通过调用根据过去经验构造出的信息，我们可以把框架调整到所有新的情况。然后我们可以对这些过去经验的细节进行更新和修正，以表示新情况的个别差异。

在宾馆房间中住过一两次的人都不会因住进一个新的宾馆房间而不知所措。他会期望看到一张床、一间浴室、一个衣柜、一部电话、价单以及门后的紧急疏散信息等。每个房间的细节是在需要时提供的，比如窗帘的颜色、电灯开关的位置和用法，等等。宾馆房间框架中还提供了一些默认的信息，比如没有床单、叫客房服务、需要冰水等。我们并不需要为见过的每个新宾馆房间建立一个框架。将宾馆房间的所有部分组织成一个概念结构，便 于登记入住宾馆时使用；单个房间的细节是根据需要提供的。

我们可以把这些较高层结构直接表示为语义网，方法是将其组织为多个独立网络的集合，每个元素表示一种典型的情况。框架（以及面向对象系统）为我们提供了一种组织工具，利用其可以将实体表示为结构化的对象，对象可以带有命名槽和相应的值。因此可以把框架或模式看成是一种简单的复合体。

例如，可以用很多个独立的框架来描述宾馆房间以及它的各个部分。以下这个框架除了表示床之外还可以表示椅子：期望高度为 20 到 40 厘米，默认有 4 个腿，设计用途是供人休息。以下是一个更复杂的表示宾馆电话的框架：这是常规电话的一种特例——它的费用是计入房间费用的；有一个专门的宾馆接线员（默认）；可以用宾馆电话订餐享受客房餐饮服务、拨打外线或接受其他服务。图 7-12 画出了表示宾馆房间的框架。

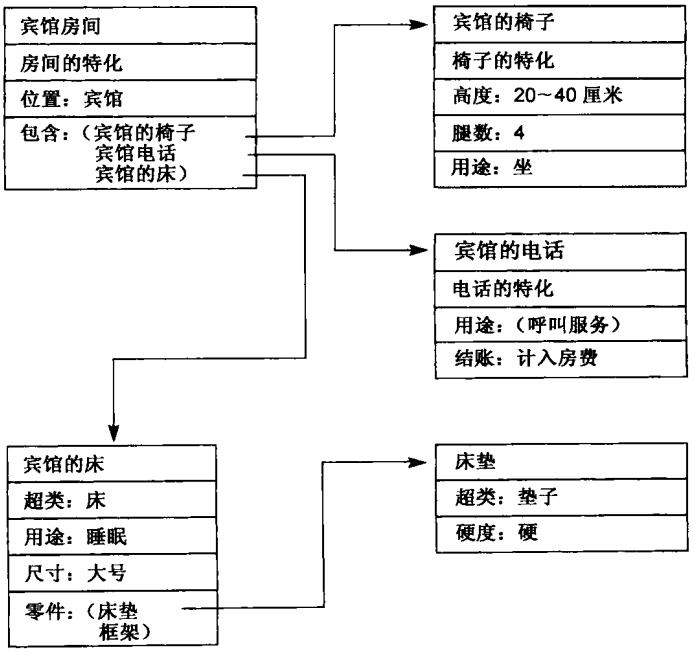


图 7-12 宾馆房间的框架描述（部分）

注：“特化”指出了一个指向超类的指针

可以把每个框架看成是一个数据结构，它包含了与典型实体相关的信息，在许多方面与传

统的“记录”很类似。框架中的槽通常包含以下信息：

1）框架标识信息。

2）这个框架和其他框架的关系。“宾馆电话”是“电话”的特例，也是“通信设备”的一个实例。

3）框架的特征描述。例如，椅子的平面离地高度是20～40厘米，靠背高出椅面60厘米等。当决定一个新对象是否适合某个框架时需要这些特征。

4）关于描述结构用法的过程信息。框架的一个重要特征是可以在槽中附加过程性代码。

5）框架的默认信息。当没有发现相反的证据时，便取这些槽值。例如，椅子有4条腿，电话是按键式的，宾馆的床是由其员工整理的。

6）新实例信息。许多框架的槽值可以是未指定的，直到遇到了特定实例或者为了求解某些问题时才指定。例如，可以不指定床单的颜色。

框架在很多重要方面扩展了语义网。尽管图7-12中的宾馆房间框架描述可能和网络描述差不多，但是框架版本对床的描述要清楚得多。在网络版本中，只看到很多结点，通过解释我们才能看到床是描述的主要对象。这种把知识组织到结构中的能力是知识库的一个重要属性。

通过框架更容易层次化地组织知识。在网络中，所有概念表示为同一个层上的结点和连接。然而，很多时候我们可能出于某种目的喜欢把一个对象看成是一个单一实体，有时又出于另一个目的希望仅考虑内部结构的细节。举例来说，通常我们并不关心汽车的机械结构；除非它出了故障，这时才打开“汽车发动机说明书”寻找问题。

过程性附件是框架的一个重要特征，通过这一特征可以把特定的代码片段与框架表示中的适当实体联系起来。例如，我们可能希望知识库具有产生图片图像的能力。这方面图形语言比网络语言更合适。我们也用过程性附件来创建*守护程序*（demon）。守护程序是作为某个其他动作的副作用被调用的过程。例如，每当一个槽值改变时，我们可能希望系统进行类型检查或一致性检验。

框架系统支持类继承。一个类框架的槽和默认值可以通过类/子类和类/成员层次继承。例如，宾馆电话是常规电话的子类，除了以下两种情况：（1）拨打所有外线要通过宾馆总机（为了记账）；（2）可以直接拨打宾馆的服务。只要没有其他的信息可以使用，那么默认值便赋给所选择的槽，例如宾馆房间中有床，因此是睡眠的合适地方；如果不知道如何拨打宾馆的前台，那么可以试一下拨“0”；宾馆的电话是按键式的（如果没有相反的证据）。

当创建类框架的实例时，系统会尽可能填写它的各个槽，可以采用的方法有：通过向用户查询、从类框架中接受默认值；或者执行某个过程或守护程序来得到实例值。和语义网的情况一样，槽和默认值可以通过类/子类层次继承。当然，默认信息可能会与问题的数据描述发生冲突，因此我们假定默认值未必总是正确的（见9.1节）。

Minsky的视觉研究内容为框架和框架默认值在推理中的用法提供了一个例子，他研究的问题是识别出一个对象的不同视图实际上表示的是同一个对象。例如，图7-13所示的一个立方体的三个视图看起来确实有很多差异。Minsky（1975）提出了一个框架系统，该系统通过把隐藏面作为默认假定来识别这些视图是同一个对象的多个角度。

图7-13中的框架系统表示了立方体的四个面。虚线表示对应的面从这个角度看不到。框架之间的连接指出了框架所表示的角度间的关系。当然，如果面上有颜色或图案，那么结点可能更复杂。实际上，一个框架中的每个槽可以是一个指针指向另一个框架。还有，因为给定的信息可以填写很多不同的槽（见图7-13中的E面），所以在存储的信息中必须没有任何冗余。

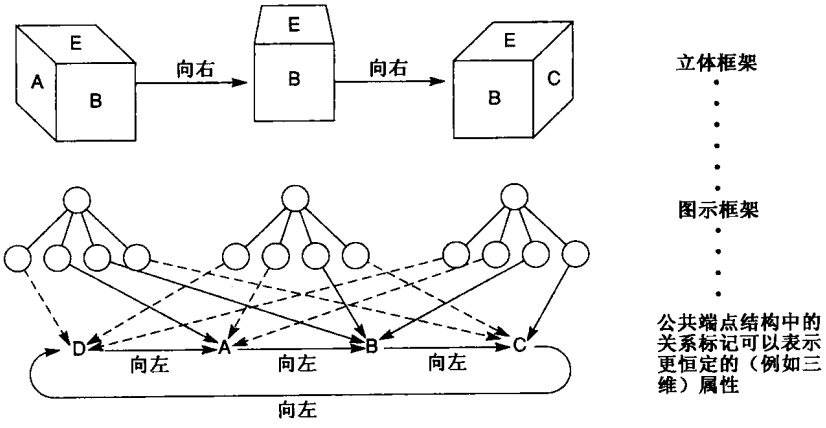


图 7-13 观察立方体的立体框架（Minsky 1975）

框架允许把复杂的对象表示成一个框架，而不是表示为庞大的网络结构。因此它大大增加了语义网的表示力。它也为表示典型实体、类、继承和默认值提供了一种自然的方式。尽管框架是一种强大的工具，但与逻辑和网络表示一样，要解决很多获取和组织复杂知识库的问题还必须依靠程序员的技巧和洞察力。最终，这项 20 世纪 70 年代在 MIT 的研究——以及在施乐帕洛阿尔托研究中心的类似研究——导致了“面向对象”程序设计原理的产生，并建立了重要的实现语言，包括 Smalltalk、C + + 和 Java。

7.2 概念图：网络语言

继开发 AI 表示模式（见 7.1 节）的早期研究工作之后，学者们又开发出了很多网络语言，用来对自然语言的语义以及其他领域建模。在这一节中，我们详细地分析一种特定的形式，以说明网络语言是如何处理语义表示问题的。John Sowa 的概念图（Sowa 1984）是网络表示语言的一个典范。本节将定义形成和操纵概念图的规则，以及表示类、个体和关系的约定。15.3.2 节将说明如何使用这种形式方法表示自然语言理解中的含义。

7.2.1 概念图简介

概念图是一种有限、连接、二部图。图的结点要么是概念；要么是概念关系。概念图不使用带标签的弧；而是用概念关系结点来表示概念之间的关系。因为概念图是二部的，所以概念仅有指向关系的弧，反之亦然。在图 7-14 中，dog（狗）和 brown（棕色）是概念结点，color（颜色）是概念关系。为了区分这两种类型的结点，我们将概念表示成矩形，将概念关系表示为椭圆。

在概念图中，概念结点要么表示具体对象，要么表示抽象对象。具体概念（比如猫、电话、饭店）是可以用脑海中形成的一幅图像来刻画的。注意具体概念既包

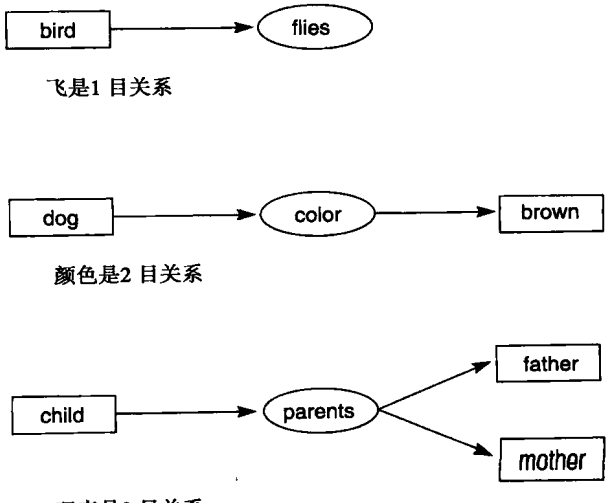


图 7-14 不同“目”的概念关系

括一般的概念，比如猫和饭店；又包括特定猫和饭店这样的概念。我们还可以形成一般猫的图像。抽象的概念包括爱、美丽和忠诚等，这些概念不对应于脑海中的图像。

概念关系结点指出一种涉及一或多个概念的关系。把概念图表达成二部图而不用带标签弧线的一个优点是可以更简单地表示任何目（arity）的关系。正如图 7-14 所示，一个 n 目关系表示为一个具有 n 个弧的概念结点。

每个概念图可以表示一个命题。典型的知识库将包含大量这样的图。图的复杂度是任意的，但必须是有限的。例如，图 7-14 中的第二幅概念图表示了命题“A dog has a color of brown”。图 7-15 是一个更复杂一些的概念图，表示的是“Mary gave John the book（玛丽把书给了约翰）”。这个图使用概念关系来表示动词“to give”的各种格，这个例子指出了使用概念图对自然语言语义建模的方式。

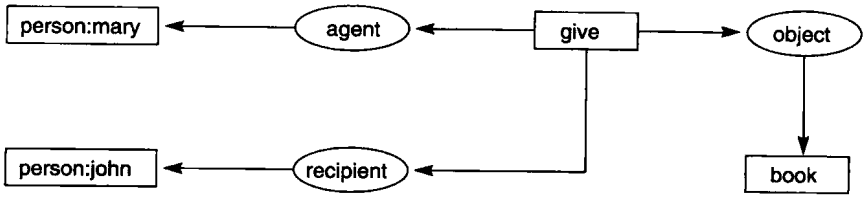


图 7-15　“玛丽把书给了约翰”的概念图

7.2.2　类型、个体和名字

语义网的很多早期设计者草率地定义类/成员和类/子类关系，导致了语义混乱。例如，个体与类之间的关系与类（比如狗）和超类（食肉动物）之间的关系是不同的。类似地，某些属性属于个体，而某些属于类本身。“有毛”和“喜欢骨头”这两个属性属于狗个体；“狗”类没有毛也不吃东西。适合于“狗”类的属性包括类名和它在动物分类学中的隶属关系。

在概念图中，所有的概念都是特定类型的惟一个体。每个概念矩形都带有一个类型标签，用来指出这个结点所表示的个体的类型或类。因此，一个带有 dog 标签的结点表示的是该类的一个个体。各个类型是按层次组织的。类型 dog 是（carnivore）的子类型，carnivore 是（mammal）的子类型，等等。具有相同类型标签的矩形表示同一类型的多个概念；不过这些矩形可以表示也可以不表示同一个个体概念。

可以用标签标出每个概念矩形的类型名以及它所表示的个体。类型标签和个体标签用冒号“:”分开。图 7-16 中的概念图指出狗“Emma（艾玛）”是棕色的。图 7-17 断言 dog 类型的某个未指定实体的颜色是 brown（棕色的）。如果标签中未指定个体名，那么这个概念表示该类型的未指定个体。

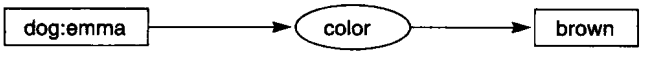


图 7-16　这个概念图指出一只名为艾玛的狗是棕色的

我们还可以用概念图来表示特定但未命名的个体。用一种称为标志的记号来指出话语世界中的每个个体。标志以“#”号开始，后面跟一个数字。标志不同于名字，因为标志是惟一的：个体可以有一个名字、多个名字、或者根本没有名字，但是它的标志有且仅有一个。类似地，不同的个体可以有相同的名字，但是不可以有相同的标志。这些区别为我们处理对象命名时的语义模糊奠定了基础。图 7-17 中的概念图断言，一条特定的 dog（#1352）是 brown（棕色的）。



图 7-17 这个概念图指出一只特定的（但未命名）狗是棕色的

我们可以通过标志把个体和它的名字分开。例如，如果狗#1352 的名字是“艾玛”，那么我们可以使用一个叫 name 的概念关系将其加到概念图中。结果如图 7-18 所示。名字用双引号包围起来，表明它是一个字符串。在不会造成歧义的情况下，可以直接用名字来引用个体，以简化概念图。根据这个约定，图 7-18 中的概念图与图 7-16 中的是等价的。

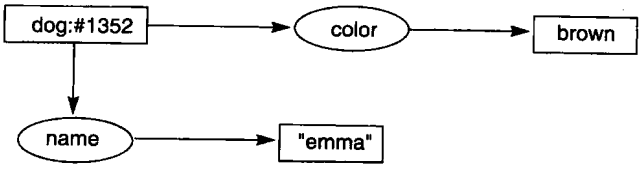


图 7-18 这个概念图指出一只名为艾玛的狗是棕色的

尽管无论是在随便的对话中，还是在正式的表示中我们都经常忽略个体和名字之间的差异，但这一差异是表示语言要支持的一个重要特征。举例来说，如果我们说“约翰”是一个常见的男性名字，那么我们是在断言这个名字本身的属性，而不是在谈任何名为“约翰”的个体。有了这一功能，我们就可以表示类似这样的英语句子“‘黑猩猩’是一种灵长类动物的名字”（“Chimpanzee” is the name of a species of primates）。类似地，我们可能还需要表示一个个体有多个名字这一事实。图 7-19 表示了一首歌中所描述的情况：她的名字叫 McGill，她称自己为 Lil，但是大家都叫她 Nancy（Lennon and McCartney 1968）”。

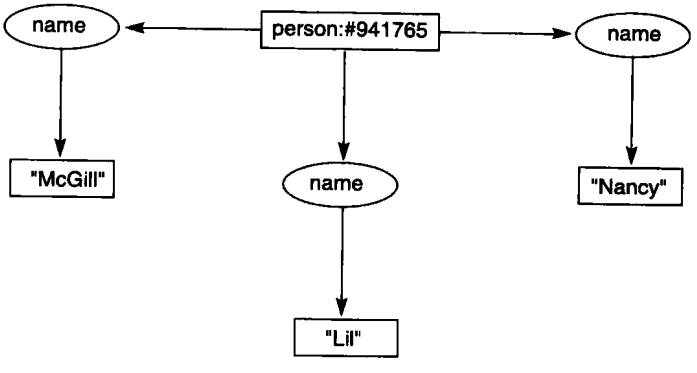


图 7-19 这个概念图表示一个人有三个名字

除了用标志或名字来指示个体外，我们还可以用标志 * 来表示未指定的个体。根据约定，可以经常省略 * 标志，因此仅给出类型标签 dog 的结点等价于标有“dog：*”的结点。除了一般标志外，概念图还允许使用命名变量。变量以星号开始，后面跟随变量名（例如 * X 或 * foo）。如果两个分隔开的结点表示同一个未指定的个体，那么这时变量是很有用的。图 7-20 表示了这个断言：“那只狗用爪子抓耳朵”（The dog scratches its ear with its paw）。尽管我们不知道哪只狗在抓耳朵，但是变量 * X 指出了“爪子”（paw）和“耳朵”（ear）是属于同一只在做“抓”（scratch）动作的狗。

总而言之，每个概念结点可以表示特定类型的一个个体。这个个体是这个概念的指向（referent）。这种指向既可以是个别的，也可以是泛指的。如果这个指向使用了个体标志，那么这个概念就是一个个体概念；如果指向使用了一般标志，那么概念就是一般概念。7.2.3 节给出了概

念图的类型层次。

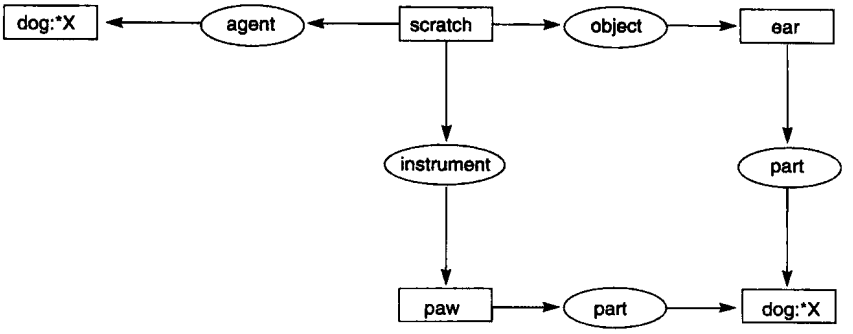


图 7-20　表示“那只狗用爪子抓耳朵”的概念图

7.2.3　类型层次

概念图的类型层次是一种用≤符号表示的部分有序关系，如图 7-21 所示。如果 s 和 t 是两种类型，并且 t≤s，那么就说 t 是 s 的子类型；s 是 t 的超类型。因为这是一种部分有序关系，所以一个类型可以有一个或多个超类型，也可以有个一或多个子类型。如果 s、t 和 u 都是类型，并且 t≤s，t≤u，那么就说 t 是 s 和 u 的公共子类型。类似地，如果 s≤v，u≤v，那么 v 便是 s 和 u 的公共超类型。

概念图的类型层次形成了网格——多继承系统的常见形式。在网格结构中，类型可以有多个双亲和孩子。不过，每个类型对必须有一个最小公共超类型和一个最大公共子类型。对于类型 s 和 u，v 是最小公共超类型的条件是：s≤v，u≤v，并且对于 s 和 u 的任意公共超类型 w 有 v≤w。最大公共子类型的定义与此类似。一组类型的最小公共超类型适于定义这些类型的公共属性（而且这些属性仅对于这些类型来说是公共的）。因为很多类型没有明显的公共子类型或公共超类型，比如“情绪”和“石头”，所以有必要增加一些类型来充当这些角色。为了使类型层次成为一个真正的网格，概念图包含了两种特殊的类型。通用类型（universal type）是所有类型的超类型，用 T 表示。荒谬类型（absurd type）是所有类型的子类型，用⊥表示。

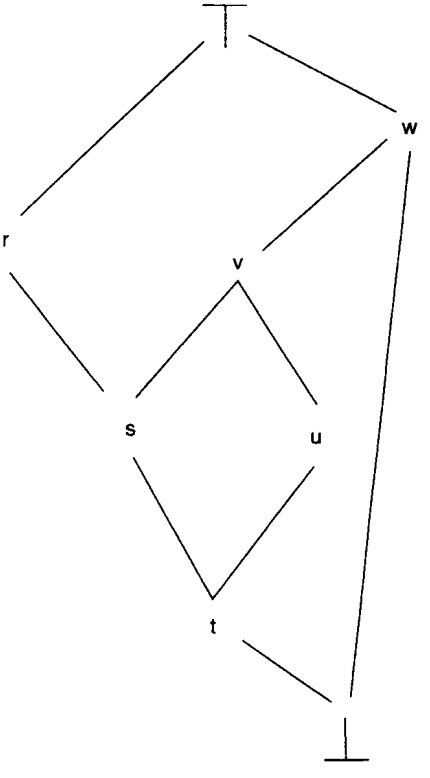


图 7-21　说明子类型、超类型、通用类型和荒谬类型的类型网格
注：弧代表关系

7.2.4　泛化和特化

概念图理论还包括很多根据现有图产生新图的运算。可以通过特化或泛化产生新的概念图并运行，这对表示自然语言的语义很重要。图 7-22 显示了四种运算：复制（copy）、限定（restrict）、联合（join）和简化（simplify）。假定 g_1 和 g_2 是两幅概念图。那么：

复制规则可以形成一幅新图 g，是 g_1 的精确拷贝。

限定运算使我们可以用结点的特例来替换结点。这有两种情况：

1）如果概念带有一般标志，那么可以把一般标志替换为个别标志。

2）可以把类型标签替换为它的子类型之一，但条件是和这个概念的指向一致。例如，我们可以把图 7-22 中的 animal（动物）替换为 dog（狗）。

联合规则可以把两幅图组合为一幅。如果图 s_1 中的结点 c_1 和 s_2 中的结点 c_2 是完全一致的，那么我们可以通过删除 c_2 并把与 c_2 关联的关系都连接到 c_1 上而组成一幅新的图。联合是一种特化规则，因为得到的图比组成它的任一幅图都更特殊。

如果图中包含两个重复的关系，那么可以删除其中之一以及它的所有弧。这就是简化规则。联合运算经常产生重复的关系，比如图 7-22 中的 g_4。

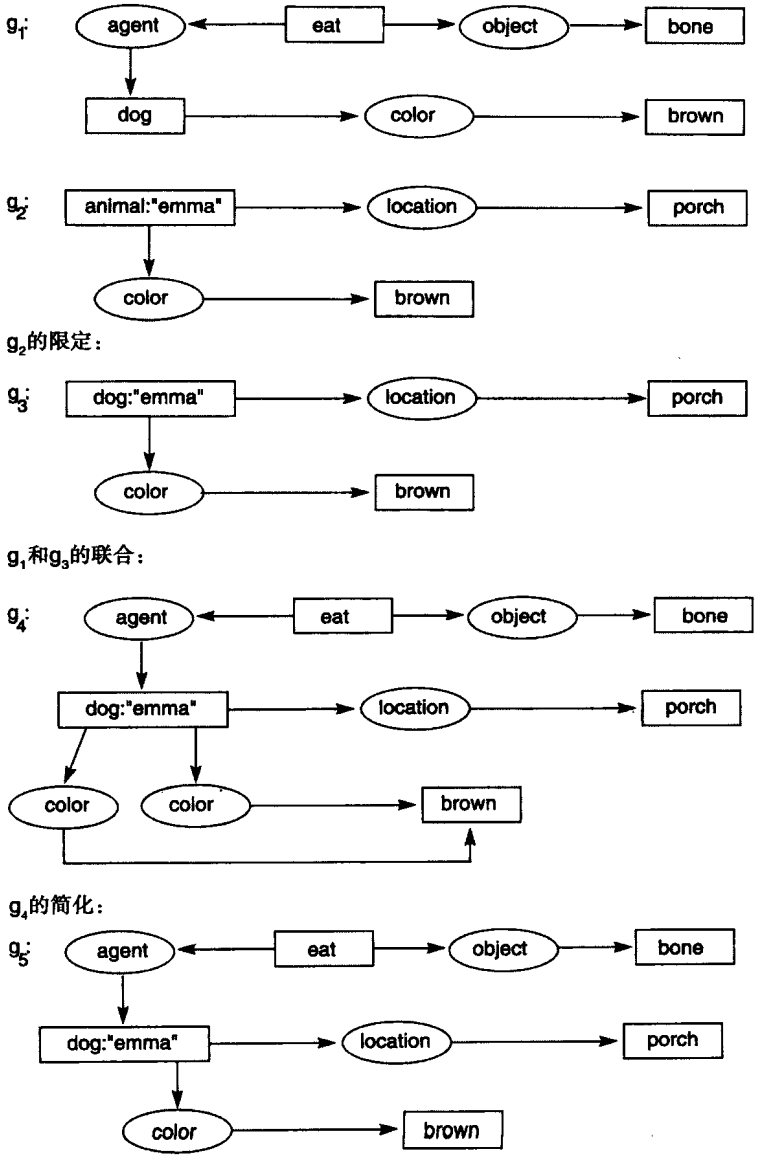


图 7-22 限定、联合和简化运算的例子

限定规则的一种用途是使两个概念匹配以便进行联合。同时使用联合和限定便可以实现继

承。举例来说，用一个个体代替一个一般标志便实现了个体对类型属性的继承。用一个子类型标签代替类型标签便实现了类型和子类型之间的继承。通过把一幅图和另一幅图联合在一起并对某些概念结点加以限定，便实现了对各种属性的继承。图 7-23 显示了如何通过用子类型标签替代类标签来实现“黑猩猩”继承“灵长类”的“有手”属性。该图也显示了个体“Bonzo”是如何通过实例化一般概念来继承这个属性的。

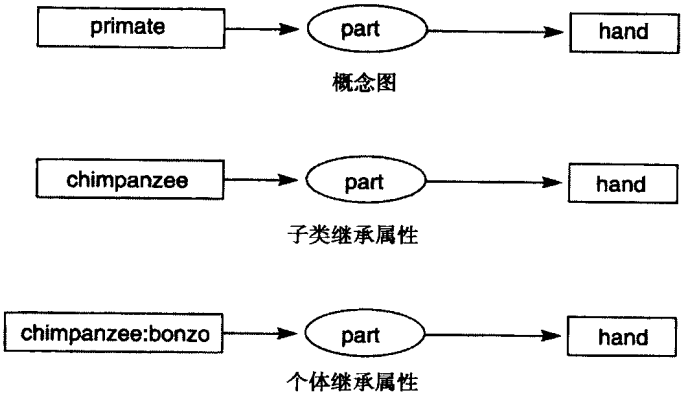


图 7-23　概念图中的继承

类似地，还可以用联合和限定来实现对理解自然语言有重要作用的似乎真实的假定（plausible assumption）。举例来说，如果有人告诉我们“玛丽和汤姆曾一起出去吃比萨”，那么我们会自动地做出一些假定：他们是吃一种圆形的上面覆盖着奶酪和番茄汁的意大利饼；他们是在一个饭馆吃的；他们一定使用了某种支付方式。可以使用联合和限定运算来实现这种推理。我们先画出这句话的概念图，然后再把这幅图和比萨及饭店的概念图（取自知识库）联合起来。通过最终的概念图我们可以认为他们吃了番茄汁并支付了账单。

联合和限定都是特化规则。它们在可导出图的集合上定义了一种部分有序关系。如果 g_1 是 g_2 的特化，那么便可以说 g_2 是 g_1 的泛化。泛化层次在知识表示中有着非常重要的作用。泛化层次不仅为继承和其他常识推理模式提供了基础，而且还用于很多学习方法中，比如从一个特定的训练实例建立更一般的断言。

以上这些规则不是推理规则，不保证从真实图中导出的图也是真实的。举例来说，图 7-22 所示概念图的限定可能是不真实的：艾玛可能是一只猫。类似地，图 7-22 的联合也没有保持它的真实性：走廊（porch）上的狗和吃（eat）骨头（bone）的狗可能是不同的。这些运算是*范式规则*（canonical formation rule），尽管它们不能保持真实性，但是具有保持“丰富语义”的重要特征。当使用概念图来实现自然语言理解时，这个保证是非常重要的。考虑下面三个句子：

阿尔伯特·爱因斯坦创立了相对论。

阿尔伯特·爱因斯坦在洛杉矶湖人队打中场位置。

概念图是黄色的飞行冰棒。

这三个句子中的第一个是真实的，第二个是假的。而第三个是没有意义的：尽管它的语法是正确的，但没有任何意义。第二个句子尽管是假的，但它是有意义的。因为我们可以想象阿尔伯特·爱因斯坦站在篮球场上。范式规则在语义上强加了约束；也就是说，它不允许根据有意义的图组成没有意义的图。尽管范式规则不是可靠的推理规则，但是它为自然语言理解和常识推理中要做的大多数似真推理奠定了基础。我们在 15.5 节进一步讨论该方法。

7.2.5 命题结点

除了使用图来定义世界中对象之间的关系外，还可以使用图来定义命题之间的关系。例如，考虑这个陈述："汤姆相信简喜欢比萨"（Tom believes that Jane likes pizza）。"相信"（believe）就是一种用命题作参数的关系。

概念图包含了一种命题（proposition）概念类型，它以概念图集合作为指向，使我们可以定义包含命题的关系。命题概念是用包含另一个概念图的矩形来表示的。我们可以使用具有适当关系的命题概念来表示有关命题的知识。图 7-24 显示了代表简（Jane）、汤姆（Tom）和比萨（pizza）间断言的概念图。其中的感受者（experiencer）关系有点类似于连接主语和动词的主体（agent）关系。感受者关系表示一种信念状态：感受到而不是做。

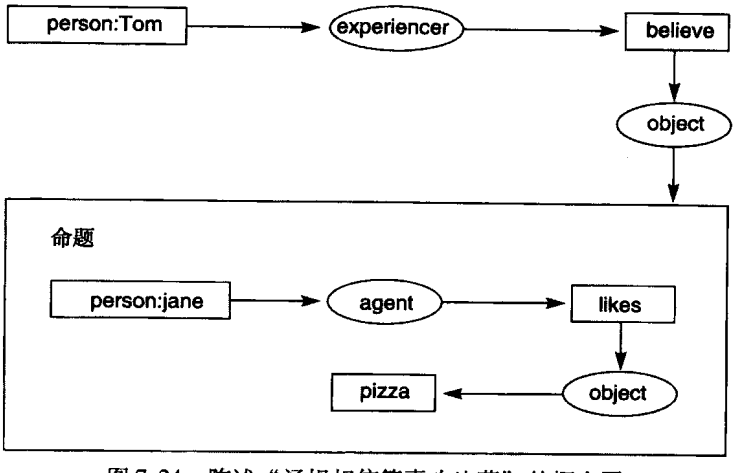


图 7-24 陈述"汤姆相信简喜欢比萨"的概念图

注：该图显示了命题概念的用法

图 7-24 显示了如何使用带有命题结点的概念图来表达知识和信念的模态（modal）概念。模态逻辑（modal logic）关心的是接受命题的不同方式：相信、断言为可能、很可能或一定是真实的、希望成为行为的结果、与事实相反的（Turner 1984）。

7.2.6 概念图和逻辑

利用概念图，可以非常容易地表示像"The dog is big and hungry"这样的合取概念，但是我们还没有建立表示非或析取的方法，也还没有讨论变量量化的问题。

可以使用命题概念和称为 neg 的一目操作来实现非。neg 以一个命题概念为参数，断言这个概念是假的。图 7-25 所示的概念图使用 neg 表示了"There are no pink dogs"这一陈述。

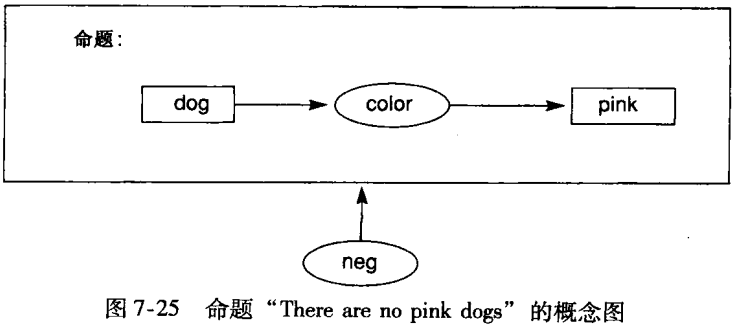


图 7-25 命题"There are no pink dogs"的概念图

根据逻辑规则，可以利用非和合取组成表示析取断言的图。为了简化这一过程，也可以定义一种或关系，它代表两个命题的析取。

在概念图中，一般概念总是假定为是存在量化的。例如，图 7-14 中的一般概念 dog 实际上表示了一个存在量化变量。这幅图对应于以下逻辑表达式：

$$\exists X \exists Y(\text{dog}(X) \wedge \text{color}(X, Y) \wedge \text{brown}(Y))$$

利用非和存在量化（见 2.2.2 节），还可以表示全称量化。例如，可以认为图 7-25 表示的是如下逻辑断言：

$$\forall X \forall Y(\neg (\text{dog}(X) \wedge \text{color}(X, Y) \wedge \text{pink}(Y))$$

从表达能力来看，概念图等价于谓词演算。正如以上例子所说明的，总是存在一个直接的映射，把概念图表示映射到谓词演算表示。下面是把概念图 g 转化到谓词演算表达式的算法（摘自 Sowa（1984））：

1）为 g 中的 n 个一般概念中的每一个分配一个惟一变量 $x_1, x_2, \cdots, x_n$。

2）为 g 中的每个个体概念分配一个惟一的常量。这个常量可以是用来表示概念指向的名字或标志。

3）用一元谓词表示每个概念结点，谓词的名字和结点的类型相同，谓词的参数是赋给该结点的变量或常量。

4）将 g 中的所有 n 元概念关系表示为 n 元谓词，谓词的名字和该关系的名字相同。谓词的每一个参数就是赋给与该关系有连接的相应概念结点的变量或常量。

5）取第 3 步和第 4 步形成的所有原子语句的合取。这便是谓词演算表达式的主体。这个表达式中的所有变量都是存在量化的。

举例来说，图 7-16 中的概念图是对应于以下谓词演算表达式的：

$$\exists X_1(\text{dog}(\text{emma}) \wedge \text{color}(\text{emma}, X_1) \wedge \text{brown}(X_1))$$

尽管可以把概念图重新形式化成谓词演算的形式，但是概念图也支持很多在 7.2.4 节介绍的特殊目的的推理机制（比如联合和限定），这些机制并不是谓词演算的正常组成部分。

以上介绍了概念图的语法并定义了限定运算（作为实现继承的一种手段）。不过我们还没有全面分析可以在这些图上进行的运算和推理，也没有讨论要处理像自然语言理解这样的问题需要定义哪些概念和关系。在 15.3.2 节将再谈这些问题并用概念图为一个简单的自然语言理解程序实现知识库。

7.3　其他表示方法和本体

近年来，AI 研究者们对显式表示在智能中的作用提出了很多质疑。除了第 11 章和第 12 章中的连接方法和涌现方法，对传统表示作用的另一个更深一层的挑战来自于 Rodney Brooks 在麻省理工学院的研究工作（Brooks 1991a）。Brooks 对在设计一个机器人探测器时中央式的表示模式的必要性提出了质疑，并利用包容体系结构试图证明一般智能是如何从较低的智能形式演化而来的。

对显式和静态表示问题的第二个挑战来自印第安那大学的 Melanie Mitchell 和 Douglas Hofstadter 的研究。Copycat 体系结构是一种进化网络，通过探索外部世界的试验来发现语义关系，并据此调整自己。

最后，在复杂环境下建立问题求解器（problem solver）通常需要使用多个不同的表示模式。

每个表示模式被称为一个本体。为了支持知识管理、沟通、备份和问题求解的其他方面，在这些不同的表示之间建立沟通和其他链接是必要的。我们在知识管理技术（knowledge management technology）总标题下，介绍这些问题和可能的解决方法。这个技术可能也支持将在7.4节中介绍的基于主体的问题求解方法。

7.3.1 Brooks的包容结构

Brooks推测（并用他研制的机器人提供了例证）智能行为并非来自于像定理证明程序或者传统专家系统（见8.2节）那样的脱离实体的系统。Brooks认为，智能是一个合理设计的系统与其环境之间相互作用的产物。此外，Brooks赞成这样的观点，智能行为是在具有更简单行为的结构的交互过程中涌现出来的，这种有组织的结构就是Brooks的包容结构。

这一包容结构是很多任务处理者的分层合集。每个任务是通过一个有限状态自动机来完成的，有限状态自动机不断地把感知器输入映射成一个面向动作的输出，很多情况下这是通过简单的“条件→动作”产生式规则（见6.2节）来完成的。这些规则以一种非常盲目（也就是说根本没有任何全局性的状态知识）的方式来决定什么动作适合子系统的当前状态。Brooks允许使用更底层系统的某些反馈。

图7-26显示了一个三层的包容结构（摘自（Brooks 1991a））。每一层都是由一个具有固定拓扑结构的有限状态自动机网络组成的，每个有限状态自动机有几种状态，并具有一两个内部寄存器和一两个内部时钟，可以访问简单的计算设备，例如可以计算向量和。这些有限状态自动机是异步运行的，通过导线发送和接收固定长度的消息。系统中不存在任何中央控制。每个有限状态自动机是由它收到的消息驱动的。消息的到达或过期导致状态自动机改变状态。因为不存在对全局数据的访问或任何动态创建的通信连接，所以根本没有可能进行全局控制。

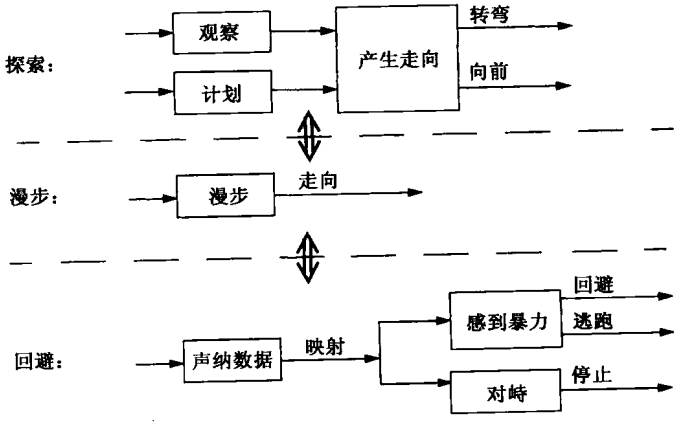


图7-26 三层包容结构的功能

注：这些层是用“回避”、“漫步”和“探索”这三个行为描述的。摘自Brooks（1991a）

图7-26显示了一个早期机器人采用的三层结构的部分功能（Brooks 1991a）。这个机器人周身装有12个声纳感知器。这些声纳感知器每秒钟会给出12个辐射深度测量结果。包容结构的最底层（“回避”层）实现了防止机器人与其他物体（不论是静止的还是运动的）相撞的行为。标有“声纳数据”的状态机发出即时“映射”，传递到“对峙”和“感到暴力”状态机，后者可以为负责使机器人向前运动的状态机产生“停止”消息。当“感到暴力”状态机被激活时，它可以产生“逃跑”或“回避”指令，以躲避危险。

这个最底层有限状态自动机网络在结构中为整个系统产生所有“停止”和“回避”指令。

下一层（“漫步”层）大约每隔十秒钟为机器人产生一个随机的“走向”指令。“回避”层的状态机取得来自“漫步”层的“走向”指令，并把它和“回避”层计算出的情况结合起来。这样，“漫步”层便既抑制了较低层的行为，强迫机器人按照接近“漫步”状态机决定的方向移动，但是同时也回避了障碍物。最后，如果“转弯”和“向前”状态机（在最顶层）被激活了，那么它们将抑制任何来自于“漫步”层的新脉冲。

最顶端的“探索”层使机器人可以探索它周围的环境，寻找远处的物体并努力“规划”一条路径抵达这个目标。这一层能够抑制“漫步”指令并观察底层如何让机器人躲避障碍物。它会纠正这些转向行为保证机器人朝着目标方向前进，在约束机器人漫游行为的同时又允许最底层的躲避功能正常工作。当底层偏离了目标时，这个“探索”层会调用“规划”状态机来使系统向着目标前进。Brooks 包容结构的关键之处是系统中不需要任何集中的符号推理过程，系统在采取任何动作时都不需要搜索可能的下一个状态。尽管这种基于行为的有限状态自动机是基于其自身当前状态产生行为提示的，但是整个系统的行为是建立在与下层系统的交互之上的。

刚才介绍的三层结构摘自 Brooks 对一种漫步探索目标机器人的早期设计。后来，他的研究小组又设计了一种更复杂具有更多层结构的系统（Brooks 1991a，Brooks and Stein 1994）。该系统可以在 MIT 机器人实验室漫步，寻找人们桌子上的空饮料罐。这需要具有发现办公室、寻找桌子和识别饮料罐的层。此外还要有能够指导机器人手臂把找到的饮料罐放到垃圾桶里的层。

Brooks 坚信顶层行为会随着下面各层的设计和检验而涌现出来。最终的连贯设计（需要跨层的和层间的通信）是通过实验发现的。尽管这种设计很简洁，但是这种包容结构已经在很多应用中表现出了很好的性能（Brooks 1989，1991a，1997）。

不过，在这种包容结构以及其他设计控制系统的有关方法中仍存在很多重要的问题（见 16.2 节）：

1）在每个系统层都存在局部信息的充足性问题。因为在每一层，纯粹的反应状态自动机是根据局部信息来做出决策的，所以这种决策很难考虑不属于这个局部层的任何信息。因此这种决策必然是“近视”的。

2）如果绝对不存在关于整个环境的任何“知识”或“模型”，那么关于局部环境的有限输入怎么能足以决策出适合全局情况的动作呢？怎么实现顶层的一致性呢？

3）一个具有有限状态的纯反应组件如何来学习所处的环境呢？如果可以说总的主体具有智能的话，那么在系统的某一层必须有足够的状态产生学习机制。

4）存在一个规模问题。尽管 Brooks 和他的助手声称已经建立了 6 层甚至 10 层的包容体系结构，但是应该遵循什么设计原则来扩展到更进一步的感兴趣行为呢？也就是这种方法可以推广到大型的复杂系统吗？

最后，我们还要问什么是“涌现”？它是魔术般的吗？在科学不断发展的今天，“涌现”一词似乎是用来搪塞我们还无法做出其他解释的现象的。这仿佛告诉我们建立一个系统并把它放入世界中检验，它便会显示出智能。不幸的是，如果没有更深入的设计指令，那么“涌现”就变成了“尚不理解”的代词。因此，很难想象我们如何能使用这种技术建立更复杂的系统。在下一节我们将描述 Copycat 结构，该结构可以通过探索发现和利用问题域中的恒定性。

7.3.2　Copycat 结构

经常听到的对传统 AI 表示模式的一种批评是这些模式是静态的，不可能反映思考过程和智能的动态特征。例如，当一个人感知到一种新的情况时，他经常是不由自主地想起这个新的情况和已知的或类似的情况的关系。事实上，很多情况下我们发现人类的感知既是从底至上的（也

就是说是由新环境中的模式激发的），又是从上至下的（是受主体的感知期望影响的）。

Copycat 是由 Melanie Mitchell（1993）在印第安那大学建立的一种问题求解结构，最初是以博士论文（受 Douglas Hofstadter 的指导）形式发表的（1995）。Copycat 结构是建立在此前的众多表示技术之上的，包括黑板结构（见 6.3 节）、语义网（见 7.2 节）、连接网络（见第 11 章）以及分类器系统（Holland 1986）。它也参考了 Brooks 的在问题域中动态调节的问题解决方法。然而与 Brooks 的结构以及连接网络不同的是，Copycat 需要一种全局的“状态”作为问题求解器的一部分。另外，Copycat 所用的表示就是这个全局状态的不断演变的特征。Copycat 支持语义网的某些特征，比如根据经历改变和成长的机制。

Copycat 最初针对的问题是感知并建立简单的类推。从这个意义上说，它是建立在 Evans（1968）和 Reitman（1965）的早期研究之上的。这一领域的很多例子是完成这样的模式：hot is to cold as tall is to {wall, short, wet, hold}；或者 bear is to pig as chair is to {foot, table, coffee, strawberry}。Copycat 可以发现合适的字符串将如下的模式补充完整：abc is to abd as ijk is to? 或者，abc is to abd as iijjkk is to? 如图 7-27。

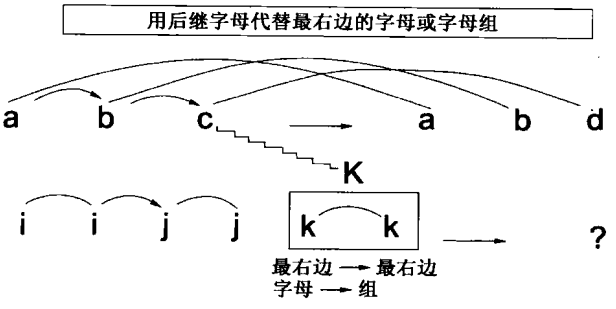


图 7-27 Copycat 工作空间的一种可能状态

注：图中画出了字母间连接和结合的几个例子。摘自 Mitchell（1993）

Copycat 是由三个主要部分组成的：工作空间（workspace）、滑动网络（slipnet）和代码轨道（coderack）。当这三个部分交互时是由一种温度尺度来仲裁的。这个温度尺度捕获了系统根据感知进行组织的程度、并控制在决策中使用随机性的程度。较高的温度反映了决策依赖的信息很少，这样做出的决策更具随机性。温度下降表明系统建立的决策是普遍认可的，低的温度表明一种答案正在浮现，同时也反映了程序对产生解的“信心”。

工作空间是一种全局结构，类似于 6.3 节中的黑板，可以用它创建系统的其他部分，也可以创建观察到的结构。从这个意义上来说，它非常像 Holland（1986）的分类器系统中的消息区。工作空间就是在输入（三个字符串，如图 7-27）层之上建立感知结构的地方，给出了这个工作空间的可能状态，有关字符串之间用线（箭头）连接起来。

滑动网络反映了类推各个部分之间的概念网络或潜在关系。观察滑动网络的一种方法是将其看成是一种可以动态变形的语义网，每个结点都有一个激活水平。网络中的连接可以由其他结点加标签。被连接结点是根据加标签结点的激活水平伸缩的。系统以这种方式根据具体情况来改变结点之间的关联程度。系统更加鼓励激活那些与当前上下文关系更密切的结点。

代码轨道是一种包含代码片段（codelet）的有偏概率优先队列。代码片段就是一小段可执行代码，用来和工作空间中的其他对象进行交互，并试探扩展当前解的某个很小部分，或者更简单地说，其用途就是为了从不同的角度来探索问题空间。在 Holland（1986）的系统中，代码片段非常像一个个单独的分类器。

滑动网络上的结点产生代码片段并将其发送到代码轨道上，按照一定的概率执行该过程。

因此，系统并行地维护着很多段代码，这些代码段竞争得到在工作空间中寻找和建立结构的机会。这些代码与它们所来自的结点是相对应的。这是由上至下活动，目的是寻找已经产生兴趣的事物的更多例子。代码段也可以从下至上工作，标识并建立工作空间中已经存在的关系。

在工作空间中建立结构的同时，系统会为建立此结构的代码段所对应的结点加一个激活尺度。这样便使系统的当前行为可以影响它的将来行为。

最后，温度尺度起到了衡量工作空间中各个结构之间“内聚力”的反馈机制的作用。如果没什么内聚力，也就是没什么结构可以提供有希望的解，那么系统所作的概率选择的偏向便不太重要：一种选择和另一种的作用差不多。当内聚力很高时，概率选择的偏向就非常重要了：做出的选择与解的发展密切相关。

目前仍有一些后续项目在研究 Copycat 结构，目的是检验设置更丰富的 Copycat 结构。Hofstadter 和他的学生（Marshall 1999）在继续研究对类推关系建模。Lewis 和 Luger（2000）已经把 Copycat 结构推广应用到移动机器人控制系统之中。这样便把 Copycat 放入了一个可以不断学习的具体环境。根据与环境的对话进行演进便得到了机器人所探索空间的地图，规划或重新规划了路径。

7.3.3　多种表示、本体和知识服务

许多复杂的人工智能应用要求设计多个表示模式，要为每个特定任务建立一个表示模式，这样结合成一个面向用户的软件平台。建立一个虚拟的会议空间是一个简单的例子。从不同地方来的不同的参加者要在这个虚拟的空间中相互交流。除了语音技术，虚拟会议空间可能需要视频流、文本和演示共享、记录访问，等等。解决该任务的一个方法是改变各种现有部件，把它们结合成一个支持所需交互的知识服务。

刚才描述的自动虚拟会议服务是结合多个表示模式来提供一个服务的实例。我们通常使用术语本体（ontology）来刻画服务的某一特定部件的表示模式。最终的知识服务需要这些部件的结合。本体来源于希腊单词“to be”，意思是存在。因此，这个术语指一个软件模块的部件和支持这些部件交互的结构，也就是，它们的功能或“组成物”（constituent being）是模块的一部分。

当用结合起来的模块建立知识服务时，必须恰当地链接每个模块以形成无缝服务。我们假定合并每个模块的本体可以得到一个新的比较复杂的本体，其支持所有服务。要创造一个新的知识服务，例如提供虚拟会议的软件，每个部件的本体必须能被理解，并且为了结合被暴露。目前有很多用于创建知识服务和其他相关软件产品的本体语言。包括 Owl 家族语言（Mika et al. 2004）、Java API 开放源码 SOFA、简单的本体框架（见网址 http：//sopha. projects. semwetcentral. org/）。

实际上，在现代计算中，有多种使用本体信息的方式，一些方式比较直接，比如对传统数据库中存储的知识的最优访问，另外一些相当新颖。由于万维网的重要性，复杂的网页检索器和搜索引擎是至关重要的支持工具。虽然目前大部分网络蜘蛛（web spider）在与文本文件直接相连的平台上搜索，但是也设想有一个更复杂的检索器，其能详细研究相当复杂表示的类和链接，也就是能够“解释”包括图和图像的比较有用的实体。

虽然我们仍然没有能力进行全局“图像解释”，但它的创立有助于支持我们创建一个语义网的愿望。我们可能想浏览我们的假期照片，从中找出所有满足某个条件的照片，例如包含一个特定的人或动物的照片。在万维网的一个较大的工作站，我们可能不仅要跟踪和找到描述自然灾害的所有新闻，而且要定位表现这些灾害的图像。目前，我们通常局限于搜索图的标题或其他（文本）描述材料，这些信息是图的内容的一个线索。

除了解释图形和图像数据外，也希望能定位关于某个主题的文本数据，并给出其内容的概要。例如，我们可能想在线搜索某领域的科技论文，并给出它们内容的报告。虽然这是自然语言理解（见第15章）领域的当前研究主题，但其实现仍属于基于计算机“语义”能力的边缘。目前，我们只能搜索较少的关键字（这些关键字可能是文章内容的指示），也许能打印出摘要，但是除了这些，我们的概括能力是有限的（Ciravegna 和 Wilks 2004，Wilks 2004）。

网络服务的进一步发展可能是搜索“帮助-需求（help wanted）”广告。要找一个具有 Java 和 C ++ 技能的、愿意在某个城市定居的软件专家。网络检索器必须知道这类广告的一些信息，包括公司名和具体位置。这些广告除了给出需要的工作能力，还描述了必要的经验和可能的薪水及待遇。这样雇佣网络服务必须具有访问雇佣“需求广告（want ads）”设计中使用的本体结构的知识和能力。15.5 节我们给出了一个这样的知识服务的例子。

然而，可能基于本体的知识服务最重要的应用是面向主体的问题求解方法。在处理任务时，主体应该是自治的、独立的、灵活的，且能适当地通信。基于主体的问题求解方法是高度分布的，例如可能要求构建一个智能网服务。具体主体能力的创立可以看作是建立和链接本体的问题。在下节基于主体问题求解方法部分，我们进一步讨论表示的分布性和面向部件的观念。

7.4 基于主体的和分布式的问题求解方法

在20世纪80年代的AI研究领域有两种对表示在智能问题求解中作用的认识，对后来的研究产生了重要的影响。第一种便是“分布式人工智能”学派（也就是DAI学派）的研究方法。第一个DAI工作室于1980年在MIT成立，主要研究包含多个问题求解器的系统，特别是与这个系统有关的智能问题求解。DAI学派当时决定不关心底层的并行问题，比如如何分布处理不同的机器或如何把复杂的算法并行化。相反，他们的目标是如何有效协调分布的问题求解器来智能地求解问题。事实上，分布式处理在人工智能中的历史比这还要早，例如使用并协调 actors 和 demons，以及黑板系统的设计（见6.3节）。

20世纪80年代的第二种研究方法便是 Rodney Brooks 及其在 MIT 的小组的观点，对此我们已经在7.3节进行了介绍。Brooks 对传统 AI 表示和推理观点的挑战产生了重要的影响（见7.3.1节）。第一，智能问题求解不需要中央知识库的思想导致了分布协作智能模型的产生，在该模型中分布表示的每个元素在问题求解过程中各负其责。第二，智能的情景动作理论使问题求解器可以把解过程的不同部分下放到其各自的环境中去。这可以使某个求解器只针对某个任务，不必具有如何在整个问题域中寻找解的知识。例如，可以让一个网络主体来检查存货信息，而让另一个主体检查顾客的信用度，两个主体都不必关心更高层的决策（比如是否批准这笔交易）。这两种研究方法至今仍影响着智能主体的设计和使用。

7.4.1 基于主体的定义

在进一步讨论有关主体（agent）的研究之前，先给出“主体”、“基于主体的系统”和“多主体系统”的定义。在关于主体的研究学派中有很多不同的小组，他们对“什么是基于主体的问题求解”持有的看法并不完全相同。本书的定义和讨论是建立在以下研究之上的：Jennings，Sycara，and Wooldridge（1998）、Wooldridge（2000）、Wooldridge et al.（2006，2007）以及 Lewis and Luger（2000）。

我们认为，多主体系统是这样一个计算机程序：包含多个置于交互环境中的问题求解器，每个求解器可以执行灵活、自主、但符合社会要求的动作，每个动作可以但不一定向着预先定义的目标前进。因此，智能主体系统的四个标准是：情景化（situated）、自主性（autonomous）、灵活

性（flexible）及社会性（social）。

智能主体的情景化是指这个主体可以接收环境的输入，并且还可以影响环境的变化。情景化主体的环境可以是互联网、比赛环境或者机器人所处的环境。例如，在机器人足球世界杯赛（ROBOCUP）竞技中（Veloso et al. 2000），每个主体必须与足球和对手恰当地交互，而它并不完全知道比赛中所有其他队员的位置、不足和优势。这种情景化明显不同于传统的 AI 问题求解器，比如 8.4 节中的 STRIPS 规划程序、8.3 节中的 MYCIN 专家系统，这些程序都维护着一个中央知识库，里面详尽包含了其所在领域中的大量知识。

自主系统的含义是它可以在没有其他主体干预的情况下和环境进行交互。要做到这一点，它必须能够控制自己的行为和内部状态。某些自主系统还可以从经历中学习，不断改善性能（参见机器学习，第四部分）。例如，在互联网上，自主主体可以对信用卡的有效性进行检验，而且这与交易中的其他问题是独立的。在机器人足球世界杯赛的例子中，主体可以根据所处的情况传球给队员或者射门。

主体的灵活性是指它可以根据目前情况做出智能响应，而且具有前摄性。响应主体可以接受环境的刺激并做出及时恰当的反应。前摄主体不仅可以对环境做出反应，而且还可以计划，寻找机会实现目标，并且对于不同的情况采取不同的合适策略。例如，信用主体可以要求用户重新提供模棱两可的信息，或者在查询一个信用卡机构不够时寻找另一个机构。足球队员主体可以根据对手的防御措施改变带球方式。

最后，主体的社会性是指它可以和其他软件或人类主体进行恰当的交互，毕竟一个主体仅是复杂问题求解过程的一部分。通过交互可以使主体的目标与其上级（更大的多主体）系统的目标相适应。主体系统的这种社会性必须考虑很多复杂的情况，包括如何为不同的主体分配子任务？各个主体如何进行通信以实现更高层的目标：在机器人足球世界杯赛例子中就是射门得分。一个主体如何支持其他主体的目标？比如处理互联网任务的安全问题。所有这些关于主体社会性的问题还是正在研究的课题。

我们已经介绍了创建多主体系统的一些基本问题。多主体系统特别适合于包含很多问题求解方法、多个观察角度以及多个实体的表示问题。在这些领域，多主体系统具有分布式、并发求解问题的优点，以及进行复杂（随机调配）交互的优势。交互的例子包括为实现一个共同目标而进行协同工作；协同组织问题求解活动以避免有害的交互、利用有益的机会；协调子问题约束以实现可接受的性能。正是这些社会化交互的灵活性把多主体系统与传统软件区别开来，并使基于主体的系统具有令人激动的强大功能。

近年来，多主体系统一词是指由多个半自主组件组成的所有类型的软件系统。分布式主体系统考虑如何用很多个模块（主体）来求解一个特定的问题，这些模块通过分割和共享问题及演化解的知识来协作。多主体系统研究的焦点是通过多个自主主体（有时已经存在）组成的群落来求解一个给定问题。也可以把多主体系统看成是一个松耦合的问题求解器网络，它们一起工作来解决可能超出其中任一个的单独能力的问题（Durfee and Lesser 1989）。知识服务和本体的设计见 7.3.3 节。

除了自主外，还可以把多主体系统的问题求解器设计为异质的。根据 Jennings、Sycara 和 Wooldridge（1998）及 Wooldridge 等（2006，2007）的分析，多主体问题求解具有四个重要的特征。第一，每个主体的信息和能力对于求解整个问题来说都是不完全的，因此有可能因为观察角度的有限而遇到困难。第二，不存在任何全局性的控制器来控制整个问题求解过程。第三，关于问题的知识和输入数据也是不集中的。第四，推理过程经常是异步的。

有趣的是，传统的面向对象程序员经常不明白基于主体的系统有什么新意。如果考虑主体

和对象的相对属性，那么这是可以理解的。对象是一种具有封装状态的计算系统，具有与状态相关联的方法，可以在环境中进行各种动作，而且可以通过传递消息来通信。

对象和主体的差异之一是对象很少对自身的行为进行控制。我们不说一个主体调用另一个主体的方法，而是说请求执行那个动作。此外，主体设计为具有灵活性（也就是响应性和前摄性）和社会性。最后，交互的主体通常具有自己的控制线程。但这些差异并不是说面向对象的程序设计语言（如 C++、Java 或 CLOS）不适合用来建立主体系统；恰恰相反，它们的灵活性和强大功能使其成为建立主体系统的理想工具。

7.4.2 基于主体的应用

为了更具体地理解上一节介绍的思想，本节介绍一些适合使用基于主体问题求解方法的应用领域。同时也介绍一些关于这些领域研究的参考文献。

- **生产** 可以用工作区层次模型来为生产领域建模。比如转床工作区、车床工作区、喷漆工作区、组装工作区，等等。然后可以把这些工作区组成多个生产子系统，每个子系统完成更大生产过程的一个功能。然后再把这些子系统组成一个工厂。更大的实体（比如公司）可以控制这些工厂的各个要素，例如管理订单、存货、利润，等等。关于基于主体生产的研究包括生产调度（Chung and Wu 1997）生产操作（Oliveira et al. 1997）以及产品的协作设计（Cutosky et al. 1993，Darr and Birmingham 1996 以及 Woldridge et al. 2007）。
- **自动控制** 因为过程控制程序往往是一种自主、可响应（reactive）、而且经常是分布式的系统，所以主体模型对其有着重要的作用。这方面的研究包括传输系统（Corera et al. 1996）、宇航器控制（Schwuttke and Quan 1993）、粒子束加速器（Perriolat et al. 1996，Klein et al. 2000）、空中交通控制（Ljunberg and Lucas 1992）等。
- **电信** 电信系统是一个由大量需要实时监控和管理的交互部件组成的大型分布式网络。基于主体系统一直应用在这一领域中，比如网络控制和管理（Schoonderwoerd et al. 1997，Adler et al. 1989，Fatima et al. 2006）、信息发送和交换（Nishibe et al. 1993）以及服务（Busuoic and Griffiths 1994）。参见（Veloso et al. 2000），其中有非常全面的讨论。
- **运输系统** 交通系统具有固有的分布性、情景化和自主特征。这方面的应用包括协调乘客和出租车（Burmeister et al. 1997）以及运输调度（Fischer et al. 1996）。
- **信息管理** 当前社会中，可用信息的丰富性、多样性和复杂性是出人意料的。主体系统使信息的智能管理成为可能，特别是在互联网上。不论是人的因素还是信息的组织方式似乎都为轻松地访问信息设置了障碍。因此，该领域中的两种关键主体任务是信息过滤（可以访问的信息中仅有很小一部分是我们真正想要的）和信息收集（搜集我们确实需要的信息片段并对其按优先级排序）。这方面的应用包括 WEBMATE（Chen and Sycara 1998）、电子邮件过滤（Maes 1994）、网络浏览助手（Lieberman 1995）以及专业的定位（expert locator）主体（Kautz et al. 1997）。知识服务见 7.3.3 节。
- **电子商务** 商业目前看起来好像由人类活动推动：我们决定什么时候买或卖，合适数量和价格，甚至每次哪些信息是合适的。当然，商业领域很适合使用主体模型。尽管要开发出完全的电子商务主体还是将来的事，但是目前已经开发出了一些可用的系统。例如，可以在股票市场中做出买卖决定的程序，它是建立在很多不同的分布信息基础上的。正在开发的主体系统还有证券管理（Sycata et al. 1996）、采购助手（Doorenbos et al. 1997，Krulwich 1996）以及交互式分类（Schrooten and van de Velde 1997，Takahashi et al.

1997）。

- **交互式游戏和剧场**　游戏和剧场提供了一种丰富的交互式模拟环境。这些主体可以把我们置身于战争游戏、财务管理，甚至体育运动的情景之中。剧场主体可以表演人类演员的角色，可以产生有情感的生活幻觉，模拟医疗紧急情况，或经过训练完成其他各种任务。这一领域的研究包括计算机游戏（Wavish and Graham 1996）、交互式人物角色（Hayes-Roth 1995，Trappl and Petta 1997）和谈判（Fatima et al. 2006）。

当然还有很多其他的领域也适合使用基于主体的方法。

尽管主体技术在智能问题求解方面具有很多优势，但是它也存在很多要克服的不足。以下这些问题是以 Jennings et al.（1998）和 Bond and Gasser（1988）中的思想为基础的。

- 如何系统地形式化和分解问题，以及如何把问题分配到各个主体？此外，如何综合各个主体的结果？
- 如何使各个主体相互通信和交互？目前有什么样的通信语言和协议可供使用？什么样的通信内容以及何时进行通信是合适的？
- 如何保证主体在行动和决策时协调一致？主体怎样处理非局部信息并避免有害的相互作用？
- 单个主体如何表示并推理其他主体的动作、计划和知识以便与其他主体合作？主体如何对协作过程中的状态进行推理？
- 如何识别并避免对整个系统有害的动作，比如杂乱无序或来回振荡的动作？
- 如何分配并管理有限的资源，不论是单个主体的还是整个系统的？
- 最终，支持和开发主体系统的最佳硬件平台和软件技术是什么？

要设计基于主体的问题求解软件所需的技巧贯穿于本书的全部章节中。第一，智能问题求解要求的表示技术是本书的一个重要主题。第二，搜索技术，尤其是启发式搜索，可以在本书的第二部分中找到。第三，8.4 节介绍的规划技术提供了如何在组织问题解的过程中排列和协调各个子目标的方法。第四，9.3 节给出了不确定情况下的随机推理技术。最后，第四部分和第五部分是针对学习、自动推理和自然语言理解问题的。这些传统的 AI 子领域在创建主体结构中都发挥着重要的作用。

还有一些用于主体模型的设计技术超出了本书的范围，例如，主体的通信语言、投标方案以及分布控制技术。这些问题是那些专门针对主体的文献（Jennings 1995；Jennings et al. 1998；Wooldridge 1998；Wooldridge et al. 2006，2007；Fatima et al. 2005，2006），尤其是很多国际会议（AAAI、IJCAI 以及 DAI）的讨论对象。

7.5　结语和参考文献

在这一章中，分析了用于知识表示的一些主要替代方法，包括逻辑、规则、语义网和框架。我们还介绍了不带中央数据库的系统，以及通用的推理模式。最后，讨论了使用主体的分布式问题求解技术。详细介绍这些方法的目的是为了深入理解这些表示方法的优势和不足。不过，对每种方法的自然性、高效性和恰当性的争论还在继续。下面简要讨论知识表示领域的几个重要问题，并以此结束本章。

首先是用于知识表示的*原子符号的选择和粒度*。世界中的对象构成了映射的定义域；知识库中的计算对象是映射的值域。因此，表示语言的原子要素特征很大程度上决定了可以对世界做出什么样的描述。例如，如果“汽车”是表示中的最小原子，那么这个系统就无法对发动机、轮胎以及汽车的任何部件做出推理。不过，如果让原子对应于这些部件，那么要把“汽车”作

为一个单一概念来表示就需要更大的结构，操纵这种更大的结构势必带来效率上的更大开销。

在自然语言理解中也需要折中考虑选择什么样的原子符号。例如，使用单个单词作为语义元素的程序，在表示没有单个词表示的复杂概念时便可能遇到困难。在分辨同一单词的多个含义或同一含义的多个单词时也会有困难。解决这一问题的一种方法是使用语义原语——独立于语言的概念单位——作为表示自然语言含义的基础。尽管这种方法避免了使用单个单词作为语义单位的问题，但是它带来了另一些问题：需要用复杂的结构来定义某些单词；依赖少量原语很难表达许多微妙的差异，比如 push 和 shove 以及 yell 和 scream。

恰当的表示应该有助于实现知识库的*详尽性*（exhaustiveness）。如果某一属性的所有情况或一类对象中的所有个体都对应于某一明确的表示元素，那么这种映射便是*详尽的*。相对某一详细度来说地图是详尽的，如果缺少某个城市或某条河流，那么便不能称其为导航工具。尽管大多数知识库是不详尽的，但是相对特定属性或对象具有详尽性是知识库的期望目标。例如，如果可以假定表示具有详尽性，那么规划程序便可以忽略*框架问题*的可能影响。

当我们把问题描述为由一系列动作或事件改变的世界状态，而且这些动作或状态通常仅改变这种描述的几个部分时；程序必须能够推断这个世界描述中的副作用和隐含变化。这种表示动作副作用的问题称为*框架问题*。举例来说，如果让一个机器人向卡车上装很重的箱子，那么这个机器人必须对由于箱重导致的卡车车厢降低进行补偿。如果表示具有详尽性，那么就不存在未指定的副作用，同时也就可以有效地消除框架问题。框架问题的难点在于对于大多数领域来说要建立完全可详尽的知识库是不可能的。表示语言应该帮助程序员决定可以安全地忽略哪些知识，并帮助程序员处理这些忽略所导致的结果（8.4 节讨论了规划中的框架问题）。

与详尽性相关的是*可塑性*（plasticity），或者叫做表示的可修改性，因为增加知识是解决详尽性不足的首要方法。由于大多数知识库都不具备详尽性，所以它应该可以被很容易地修改和更新。除了增加知识的语法要简洁外，表示还应该有助于保证知识库在增删知识时的信息一致性。例如，继承（使新的实例可以继承类的属性）便有助于保证信息的一致性。

有些系统，包括 Copycat（Mitchell 1993）和 Brooks 的机器人（7.3 节），通过设计随自然世界的约束而变化或演变的网络结构来处理可塑性问题。在这些系统中，表示是在自底向上获取新数据的过程中形成的（同时受感知系统的期望约束）。类比推理是这种系统的一个例子。

表示的另一个有用属性是知识库与世界之间映射的*同构程度*。这里，同构是指世界中的对象或动作与表示语言中的计算对象和运算操作是一一对应的。通过同构映射，知识库可以更自然而且直观地反映和组织问题域中的结构。

除了自然性、直接性和易用性外，评估表示模式好坏的标准还有*计算效率*。Levesque 和 Brachman（1985）讨论了如何在表达力和高效性之间做出折中。当用逻辑语言作为表示模式时，其完备性带来了很高的表达力，不过基于无约束逻辑的系统在效率方面要付出很高的代价，参见第 14 章。

刚刚提到的大多数表示问题关系到任何用计算机表达信息的场合。在设计分布式主体系统时还必须考虑很多更深层的问题。这些问题包括利用部分（局部）信息做出决策、分解任务、主体的通信语言，以及开发算法以实现主体间的合作和信息共享。7.4 节对这方面的许多问题作了介绍。

最后要说明的是，如果要按照 Clark（1997）、Haugeland（1997）以及 Dennett（1991，1995，2006）提出的分布式的基于环境的智能方法实现所谓的“leaky”系统——利用环境和其他主体作为存储和使用知识的关键介质，那么可能还需要发明一种全新的表示语言。另外，

基于或支持 Brooks（1991a）提出的“使用世界作为其自身模型”思想的表示工具还不知在何处。

下面我们为本章介绍的内容提供一些更深入的参考资料。Selz（1913，1922）、Anderson and Bower（1973）、Sowa（1984）、Collins and Quillian（1969）研究了关联理论，该理论一直用来对计算机和人类记忆以及推理建模。

语义网处理系统（Shapiro 1979，Shapiro et al. 2007）是最早的基于联想主义/逻辑（associationist/logic）表示之一，见 7.1 节。最近，语义网处理系统作为一个工具被扩展用于元认知，其中谓词允许以其他谓词作为参数，因此支持对知识库结构进行推理。

结构化知识表示语言方面的重要工作包括 Bobrow 和 Winograd 的 KRL 表示语言（Bobrow and Winograd 1977）、Brachman（1979）的 KL-ONE 表示语言，后者特别强调了结构化表示的语义基础。

我们对概念图的介绍很大一部分取材于 John Sowa 的著作《Conceptual Structures》（1984）。建议读者参考该书以了解本书忽略的细节。在完整的概念图中，可以把谓词演算、模型和高阶逻辑的表示力融合在一起，并内建来自认识论、心理学和语言学的大量概念和关系。

针对表示问题的其他方法还有很多。例如，Brachman、Fikes 和 Levesque 提出了一种强调功能说明的表示方法，功能说明包括，可以向知识库询问哪些问题，以及它可以告诉我们哪些信息（Brachman 1985，Brachman et al. 1985，Levesque 1984）。

很多参考书可以帮助我们进一步研究这些问题。Brachman 和 Levesque（1985）编辑的《Readings in Knowledge Representation》汇编了这一领域的很多重要但已过时的文章。在该书中可以找到本章引用的很多文章，只不过本章所指出的是这些文章的原始出处。以下读物都是很重要的：Bobrow and Collins（1975）编著的《Representation and Understanding》、Davis（1990）编著的《Representation of Commonsense Knowledge》、Weld and deKleer（1990）编著的《Readings in Qualitative Reasoning about Physical Systems》。Brachman 等人（1990）编著的《Principles of Knowledge Representation and Reasoning》、Mylopoulos and Levesque（1984）编著的《Overview of Knowledge Representation》，以及人工智能年会的论文集都是很有用的资源。

还有很多论文和文章继续研究由 Brooks 提出的研究课题以及他提出的包容结构；尤其要参见 Brooks（1991a）、Brooks and Stein（1994）、Maes（1994）和 Veloso 等人（2000）。关于分布式和物化本质的知识和智能的哲学观点，参见 Andy Clark 的《Being There》（1997）。

对基于主体方法的研究在现代 AI 中非常流行。Jennings 等人（1998）对这一领域进行了介绍。目前在 AI 年会（IAAI 以及 IJCAI）中有些部分完全是针对主体研究的。推荐读者参考这些会议的最新学报以了解该领域的最新研究动态。我们也推荐读者阅读一下分布式人工智能（DAI）领域的会议学报和读物。7.4 节简要介绍了有关主体研究重要应用的参考文献。

在 AI 和认知科学之间的中间领域也存在知识表示问题，参见《Cognitive Science：The Science of Intelligent Systems》（Luger 1994）以及《Being There》（Clark 1997）。George Luger（1995）的《Computation and Intelligence》收集了很多集中讨论各种不同表示模式的经典论文。

7.6　习题

1. 常识推理采用的是因果、类比和等价等方法，但使用的方式不同于形式语言。例如，如果我们说“Inflation caused Jane to ask for a raise.”，那么暗示了一种在简单定律中找不到的更复杂原因。如果我们说“Use a knife or chisel to trim the wood.”，那么是在暗示一种重要的等价概念。请讨论如何把这些语言以及类似这样的概念翻译成形式语言。
2. 在 7.2.1 节中给出了反对用逻辑表示常识知识的理由。请给出一个支持使用逻辑来表示常识知识的理

由。McCarthy and Hayes（1969）进行了有趣的讨论。

3. 将下面的话分别翻译为谓词演算、概念依赖和概念图形式：

   "Jane gave Tom an ice cream cone."
   "Basketball players are tall."
   "Paul cut down the tree with an axe."
   "Place all the ingredients in a bowl and mix thoroughly."

4. 阅读 Woods（1985）所写的《What's in a Link》一文。该文的第 4 节列出了知识表示中的很多问题。利用逻辑、概念图和框架表示为这些问题各提出一种解决方法。
5. 将图 7-28 中的概念图翻译成英语语句。

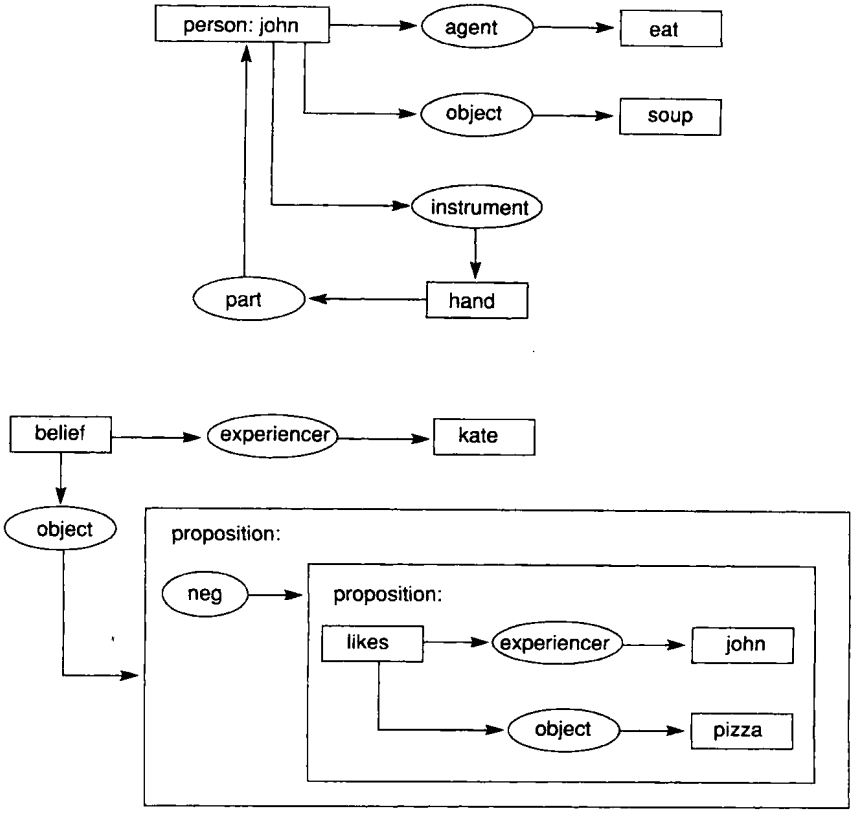


图 7-28 要翻译成英文的两个概念图

6. 联合与约束运算定义了一种针对概念图的泛化序，说明这种泛化关系是可传递的。
7. 利用联合与约束的概念图特化不是保真操作。给出一个例子说明真实图的约束未必是真实的。然而，真实图的泛化总是为真的；证明这一点。
8. 利用概念图定义一种专用的语言来描述公共图书馆的行为。语言中应该包含一系列用概念图表示的概念和关系。然后再为零售业也定义一套同样的语言，找出这两套语言中都有的概念和关系，并指出哪些概念和关系在两种语言中都有但含义不同。
9. 将图 7-28 中的概念图翻译成谓词演算表示。
10. 将 2.4 节中的财务顾问知识库翻译成概念图形式。
11. 从你的经历中找出一些证据来说明人类的记忆是以类似脚本或类似框架的方式组织的。
12. 利用概念依赖表示为以下情况定义脚本：

    a）快餐店。

b）和二手车销售员交互。

c）听歌剧。

13. 为“交通工具”这个概念建立子类型层次图；例如，它的子类型包括“陆上交通工具”和“水上交通工具”。这两个子类型又包括进一步的子类型。这种关系最好表示为树、格子、还是一般的图？对“运动”和“愤怒”这两个概念重复以上的过程。

14. 建立一种类型层次，其中某些类型没有公共的超类型。加入类型使其成为一种网格形式。这种层次可以用树表示吗？这样做会有什么问题？

15. 下面的每个字符序列都是按照某个一般规则产生的。为每个序列建立一种表示来描述该序列中的规则或关系。

a）2，4，6，8，…

b）1，2，4，8，16，…

c）1，1，2，3，5，8，…

d）1，a，2，c，3，f，4，…

e）o，t，t，f，f，s，s，…

16. 7.3.2 节中介绍了两个类比推理的例子。设计一种恰当的表示和搜索策略来识别每种情况下的最佳答案，再建立两个类推的例子证明你提出的表示是有效的。能解决下面这两个实例吗？

a）hot is to cold as tall is to {wall, short, wet, hold}

b）bear is to pig as chair is to {foot, table, coffee, strawberry}

17. 描绘一种表示可以用来求解类似图 7-29 中的类比问题。T. G. Evans（1968）讨论了这类问题。这种表示必须能够表示大小、形状和相对位置等关键特征。

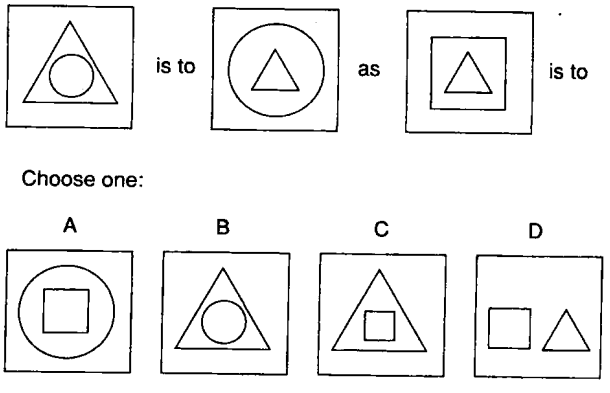


图 7-29　类比测试问题的例子

18. Brooks 的论文（1991a）中包含了对表示在传统 AI 中作用的重要讨论。阅读这篇论文，并谈一谈显式的、通用表示模式的不足。

19. 在 7.3.1 节的末尾，介绍了 Brooks 包容结构存在的 5 个潜在问题。选择其中的一个或多个加以评论。

20. 如果要用一种主体语言来实现面向主体的互联网服务，那么这种语言应该具有哪些特征（列出 5 种）？对 Java 作为一种通用主体语言在建立互联网服务方面的作用加以评论。你发现 CLOS 也有类似的作用吗？为什么？关于这一问题在互联网上有很多资料。

21. 在 7.4 节的接近末尾部分提到了面向主体方法中存在的很多重要困难。选择其中之一加以评论。

22. 假定你在设计一个主体系统来表示一个足球队或橄榄球队。为了在防守或进攻策略中进行合作，这些主体必须对其他主体的计划和可能反应有所了解。建立一个模型来描述其他合作主体的目标和计划。

23. 在 7.4 节中归纳的主体结构应用领域中选择一个领域。选择并阅读该领域中的一篇研究或应用论文。然后设计一种针对该问题的主体结构，把这个问题分解到每个主体，列出恰当的合作过程。

第 8 章　求解问题的强方法

智能主体性能表现出的问题求解能力主要是由其知识库决定的，其次才是它所采用的推理方法，这是知识工程的第一原则。专家系统必须具备丰富的知识，即使方法差一些。这是一个非常重要的成果，但直到最近它才被 AI 界充分理解。在很长的一段时期内，AI 一直把几乎所有的注意力都集中在开发聪明的推理方法上；其实几乎所有推理方法都可以胜任。求解问题的威力在于知识。

——Edward Feigenbaum，斯坦福大学

知识就是力量。

——弗朗西斯·培根

8.0　简介

本章继续研究智能表示问题，探讨 AI 的一个重要组成部分：求解问题的知识密集型方法，即强方法（strong method）。

人类专家能够表现出很高的推理水平是因为他们对自己所处的专业领域了解得非常透彻。这个简单的道理是设计强方法或基于知识的问题求解器的理论基础（参见第三部分的简介）。例如，专家系统就是这样的程序，它使用针对某一问题域的知识为该领域提供“专家级”的服务。概括地讲，专家系统设计者首先在人类专家的帮助下获取知识，然后再用专家系统来模仿人类专家的方法和能力。和人类专家一样，专家系统往往是专门针对某一狭窄领域的。另外和人类一样，专家系统也可以通过在求解问题实践中获得的知识来增加技巧、捷径和启发，从而提高它们对问题域的理论理解。

因为专家系统都有使用启发和知识密集型的特征，所以它们通常：

1）支持观察推理过程，既可以给出中间步骤，也可以回答有关求解过程的问题。

2）允许很容易地向知识库中增加或从中删除技巧。

3）启发式推理，利用（经常是有缺陷的）知识得到有用的解。

专家系统的推理过程是对观察开放的，它可以提供问题求解状态的信息，也可以对程序的决策或选择做出解释。如果要让人类专家接受计算机的推荐，那么解释是非常重要的。因为很少有哪个专家在没有理解某个建议时轻易接受它，更不用说计算机提出的建议了。

AI 和专家系统程序设计的探索性要求这些程序必须可以很容易地被原型化、检验和修改。设计 AI 程序设计语言和环境时的一个目标便是如何支持这种循环的开发方法。例如在纯粹的产生式系统中，修改某个单一规则根本不会对全局语法产生任何副作用。因此，在加入或删除规则时不需要对更上级程序进行修改。专家系统设计者经常解释知识库的易修改性是生产一个成功的程序的主要因素。

专家系统的另一个特征是使用启发式的问题求解方法。专家系统设计者们已经发现，非正式的“窍门”和“经验法则”是对书本上正式理论的重要补充。有时这些规则以可理解的方式扩展了理论知识，而且经常成为行之有效的捷径。

专家系统可以用来求解很多领域的问题，比如医疗、数学、工程、化学、地质学、计算机科学、商业、法律、国防和教育等领域的问题。这些程序所针对的问题非常广泛，下面列出了一些常见的问题（摘自 Waterman 1986）：

解释——从大量原始数据总结出高层结论。
预测——推测出给定情况下可能发生的结果。
诊断——根据可观察的症状决定复杂环境中的故障原因。
设计——对系统组件进行配置，以达到目标性能，同时要满足一系列设计约束。
规划——根据给定的起始条件和运行期约束，设计出一系列动作以实现目标。
监控——将观察到的系统行为和它的期望行为进行比较。
指导——对技术领域的教学过程提供帮助。
控制——对复杂环境下的行为进行管理。

本章首先分析基于知识的问题求解方法。成功的知识工程必须有全面的考虑，从选择合适的应用领域，到选择恰当地获取和形式化问题求解知识的方法。8.2 节介绍基于规则的系统，并给出了一个产生式系统以说明求解和解释过程的软件体系结构。8.3 节分析了基于模型的和基于实例的推理技术。8.4 节讨论了规划——将知识片组织成一系列可以实现目标的动作的过程。第 9 章中将介绍的不确定情况下的推理技术对设计强方法问题求解器来说也是非常重要的。

8.1　专家系统技术概览

8.1.1　基于规则的专家系统设计

图 8-1 画出了典型专家系统的各个组成模块。用户接口是用户与系统之间的交互渠道，它简化了通信过程并隐藏了大多数复杂细节，比如规则库的内部结构。专家系统的人机接口有很多种，包括问答式、菜单驱动或图形接口等。接口类型的最终选择应当折中考虑用户的需求以及知识库系统和推理系统的需要。

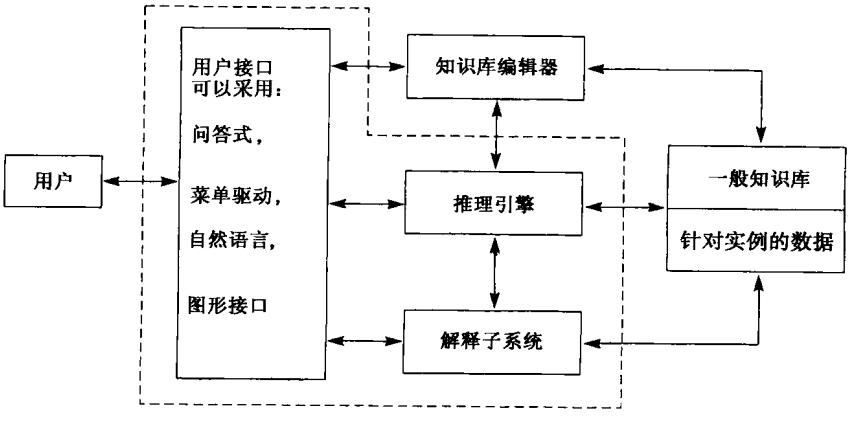


图 8-1　专家系统的典型体系结构

专家系统的核心是包含特定应用领域知识的知识库。在基于规则的专家系统中，知识被表示为“如果……那么……”形式的规则，如 8.2 节的例子所示。知识库既包含一般知识，又包含针对实例的信息。

推理引擎应用知识来求解实际的问题。它实质上是知识库的一个解释器。在产生式系统中，推理引擎执行“识别 - 动作”控制循环。这个实现控制循环的过程与产生式规则本身是分开的。把知识库和推理引擎分开维护是很重要的，主要原因有：

1）这种分离使我们可以更自然地表示知识。例如，“如果……那么……”形式的规则比底层的计算机代码更接近人类描述问题求解技巧的方式。

2）因为知识库与程序的底层控制结构是分离的，所以专家系统设计者可以把精力集中在获取和组织求解问题的知识上，而不是计算机实现的细节。

3）理想情况下，知识和控制的分离可以使修改一部分知识库不会对其他部分产生副作用。

4）知识和控制要素的分离使一套控制和接口软件可用于多个系统。*专家系统外壳*具有图 8-1 中除知识库和针对实例的数据外的所有部分——这两个部分可以根据新的应用加入。图 8-1 中的虚框指出了外壳模块。

专家系统必须记录针对实例的数据，也就是和实例有关的事实、结论和要考虑的其他信息。具体包括问题实例中给出的数据、部分结论、结论的置信度以及搜索过程中的死端。这些信息和一般知识库是分开的。

*解释子系统*用来向用户对推理进行解释。解释包括系统是如何做出结论的——回答 how 查询（见 8.2 节）；为什么系统需要特定的数据片——回答 why 查询（见 8.2 节）；以及关于程序动作的有价值的辅导性解释或者深层的理论依据。

很多系统还包括*知识库编辑器*。知识库编辑器可以帮助程序员定位和修改程序的执行错误，因此经常要访问解释子系统所提供的信息。知识库编辑器还可以帮助增加新的知识、维护和纠正规则语法、对更新后的知识库进行一致性检查。

目前很多专家系统的设计和发布时间大大降低的一个重要原因是可以立刻得到*专家系统外壳*。例如，NASA 建立了 CLIPS；JESS 可以从 Sandia 国家实验室得到；在补充资料里我们提供了一个用 LISP 和 Prolog 写的外壳。不幸的是，外壳程序并不能解决建立专家系统所要解决的所有问题。尽管知识和控制的分离、产生式系统体系结构的模块化以及使用合适的知识表示语言都有助于专家系统的建立，但是获取和形式化应用领域内的知识仍然是很难的任务。

8.1.2　问题选择和知识工程的步骤

开发专家系统需要投入相当多的人力和财力。企图求解过于复杂、还未充分理解或目前技术还难以胜任的问题将导致代价昂贵的、难堪的失败。为此，研究人员总结出了一些原则来判断一个问题是否适合用专家系统来求解：

1）**需求程度决定了开发专家系统值得投入的成本和精力**。很多领域已经建立起了专家系统，比如探矿、商务、国防和医疗等，挖掘这些现有系统的潜力可以大大节约金钱、时间和人力。

2）**人类专家无法出现在所有需要他们的地方**。例如在地质领域，偏远的挖掘和钻探站点也很需要专家。因此，地质学家和工程师们很多时候要长途跋涉到各个站点，导致很大的经济开销而且浪费了很多时间。如果在这些站点安装一套专家系统，那么很多问题不需专家亲自访问就可以解决了。

3）**利用符号推理可以求解的问题**。问题求解过程不应该需要敏捷的物理运动或感知技术。机器人和机器视觉系统目前还缺乏人类的技巧和灵活性。

4）**结构性好而且不需要常识推理的领域**。高科技领域具有研究深入和规范的优点：有定义周密的术语和清晰的概念模型。相反，常识推理是难以自动化的。

5）**使用传统计算技术不可以解决的问题**。专家系统技术不该用在那些不必使用专家系统的领域。如果一个问题可以用更传统的方法得到满意的解，那么便没有必要考虑专家系统。

6）**有乐于合作又善于表达的专家**。专家系统的知识来源于工作在这个领域的人的经验和判断。因此这些专家愿意而且能够分享知识是非常重要的。

7）**问题的大小和范围很合适**。举例来说，试图用一个程序来捕获一个医生的所有医疗技能

是不可能的；相反，用一个程序为医学博士们提出诊断建议或关于某一医疗设备用法的建议更可行一些。

建立专家系统过程中涉及的主要人员是知识工程师、领域专家和最终用户。知识工程师是AI语言和表示专家。他们的主要任务是为这个项目选择软硬件工具，帮助领域专家表示必要的知识，并准确高效地实现知识库。很多时候，知识工程师起初是不了解应用领域的。

领域专家的任务是提供问题域中的知识。领域专家通常已经在该领域工作多年而且理解该领域中的各种问题求解技术，比如各种捷径、处理不精确的数据、评估部分解的好坏以及人们认为专家该具有的其他技能。领域专家主要负责向知识工程师表达出这些技能。

和大多数应用一样，最终用户决定着主要的设计约束。如果用户不高兴，那么开发所付出的精力基本上是白费了。因此必须在整个设计循环中始终考虑用户的需求和技能：这个程序是否会使用户的工作更简单、更迅速、更舒适？用户需要什么程度的解释？用户是否能为系统提供正确的信息？用户接口是否合适？用户的工作环境是否会限制程序的使用？例如，需要打字输入的接口就不适合用在飞行器的座舱中。

和开发大多数AI程序一样，建立专家系统需要一种非传统的开发循环：先以早期的原型为基础，然后再逐步改进代码。一般来说，最初的工作是从知识工程师了解和熟悉问题域开始的。这有助于和领域专家沟通。具体做法可以是和专家面谈或观察专家工作的方法。接下来，知识工程师和专家开始提取专家求解问题的知识。这经常是通过向专家询问一系列问题并让他们解释他们在求解问题时所用的技术来实现的。在这个过程中使用录像和录音工具是很必要的。

很多时候，知识工程师不熟悉问题域是很有用的。众所周知，人类专家常常不屑于解释求解复杂问题过程中的精确细节。很多时候他们忘记提到那些对他们来说很明显的步骤，他们已经在该领域工作多年，所以有些步骤下意识地或本能地完成了。如果知识工程师对该领域是比较陌生的，那么他们就可以发现这些概念上的跳跃并提出来。

一旦知识工程师已经对问题域有了概括性的了解，并和专家参加了几个解决问题的会议，那么就可以开始实际的系统设计了：选择一种表示知识的方法，比如规则或框架；决定一种搜索策略，正向、反向、深度优先还是最佳优先；并设计用户接口。在设计好了这些事项后，知识工程师便建立起了一个原型。

这个原型应该能够求解该领域中某个小范围的问题并为初步的设计假设提供一个检验平台。实现了原型之后，知识工程师和领域专家便可以通过给出一些问题来检验并改进它的知识，以纠正它的不足。如果设计原型时所作的假定被证实是正确的，那么就可以不断地扩展这个原型了，直到实现最终的系统。

专家系统是通过循序渐进的逼近过程建立起来的，不断发现程序的不足，在解决不足时又不断地增加或纠正知识库。从这个意义上来说，知识库是“长”起来的，而不是“建”起来的。图8-2所示的流程图描述了这种探索程序设计开发的循环过程。Seymour Papert在他的LOGO语言中使用了这种程序设计方法（Papert 1980），Alan Kay在施乐公司帕洛阿尔托研究中心研究Smalltalk也探讨了这种方法。LOGO的逻辑是，观察到计算机输出一种表示不当的格式可以引导开发者用更精确的代码来纠正并澄清这些部分。这种试验－纠正候选方案的过程在专家系统的开发中是非常常见的，这与那些纯粹的自顶向下设计的层次结构是完全不同的。

如果原型变得过于笨重或者设计者决定修改求解问题的基本方法，那么抛弃目前的原型也是可以理解的。原型使程序设计者可以通过建立实际程序探索问题和各种重要的关系。在这个循序渐进的探索过程完成之后，设计者经常可以用更少的规则写出更简洁的版本。

专家系统编程的第二个特征是永远不要考虑这个项目“结束”。一个庞大的启发式知识库总是存

在不足的。产生式系统模型的模块化使增加新规则变得很自然，而且可以随时弥补知识库中的不足。

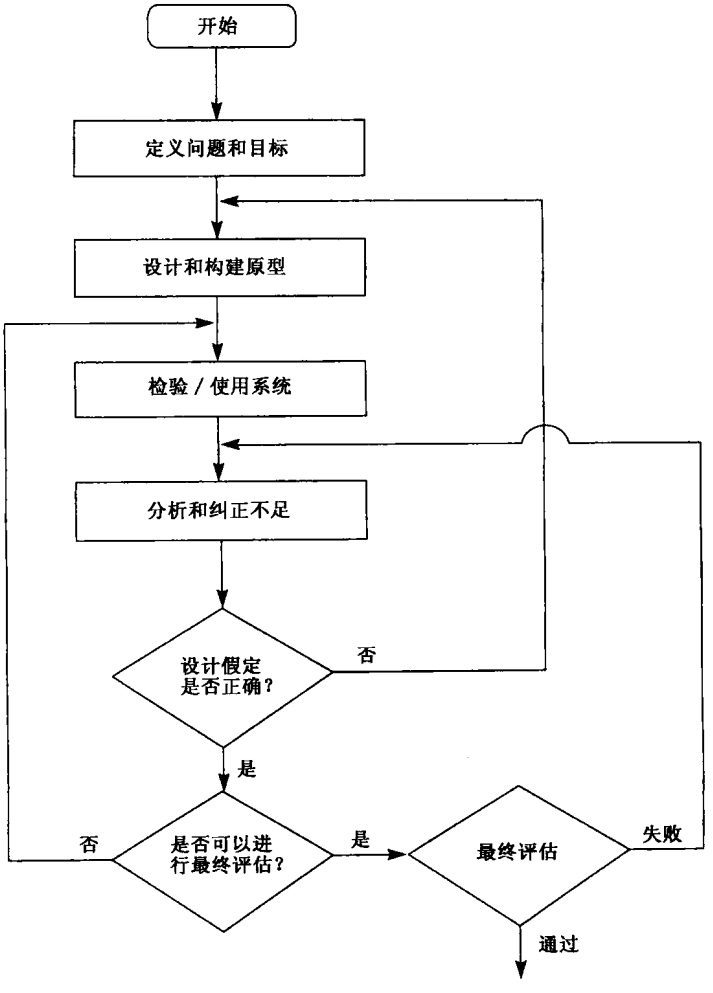


图 8-2　探索性开发循环

8.1.3　概念模型及其在知识获取中的作用

图 8-3 给出了知识获取过程的一个简化模型，可以将其当作一个“初步近似”，以帮助我们理解如何获得并形式化人类的专业技能。工作在某一应用领域的人类专家可以娴熟地运用该领域的知识和技能并进行实践。这种知识经常是含糊的、不精确的、而且表述也是不完全的。因此知识工程师必须将这些非正式的专业技能翻译成适合计算机系统使用的形式语言。在形式化人类技能时会遇到很多重要的问题：

1）人类的技能经常是下意识的。正如亚里士多德在他的《伦理学》中所指出的，“我们必须学习如何实践，但我们是通过实践来学习的”。例如，医学博士所拥用的技能主要是通过多年的实习和临床经验以及对患者的长期关注而获得的。医疗技能很大程度上是由实践产生的。在工作了多年之后，这些技能已经高度集成到一起而且基本上可以下意识地完成。对专家来说，要精确地描述他求解问题的详细过程可能是有困难的。

2）人类专家经常是知道如何处理某种情况，但却不知道这种情况的理性特征是什么。他们

能够熟练地工作，但并不关心其中蕴含的原理是什么。一个明显的例子便是骑独轮车：实际上，成功的独轮车骑手无须求解保持平衡的微分方程组，而是依赖对“重力”、“动量”和“惯性”的直觉来形成一种有效的控制过程。

3）我们通常把知识获取看成是取得关于客观实体（即所谓的“真实世界”）的事实知识的过程。理论和实践都已经证明，人类专家使用模型来表示世界中的个体或团体，而且这些模型除了受试验方法影响外，还受个人习惯、社会过程等的影响。

4）专业技能是变化的。人类专家不仅不断地获取新知识，而且还经常对现有的知识进行更迭，这一点已经被科学和社会领域中的很多争论所证实。

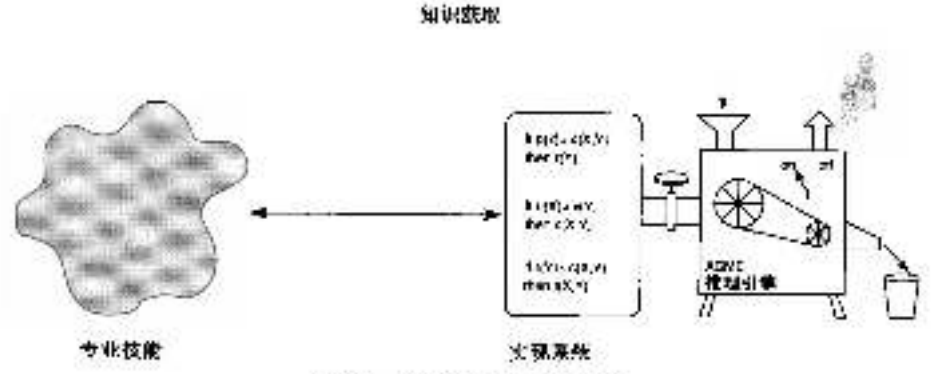


图 8-3　专家系统的建立过程

因此，知识获取是一项难度很大的工程，而且它对延长专家系统的生命周期有着重要的作用。为了简化这一工程，图 8-4 所示的概念模型是很有用的，该模型将人类的专业技能和程序实现联系起来。这里所说的概念模型就是指知识工程师关于领域知识的概念。尽管这无疑与领域专家的作用有所不同，但是正是这种模型决定了形式知识库的建立。

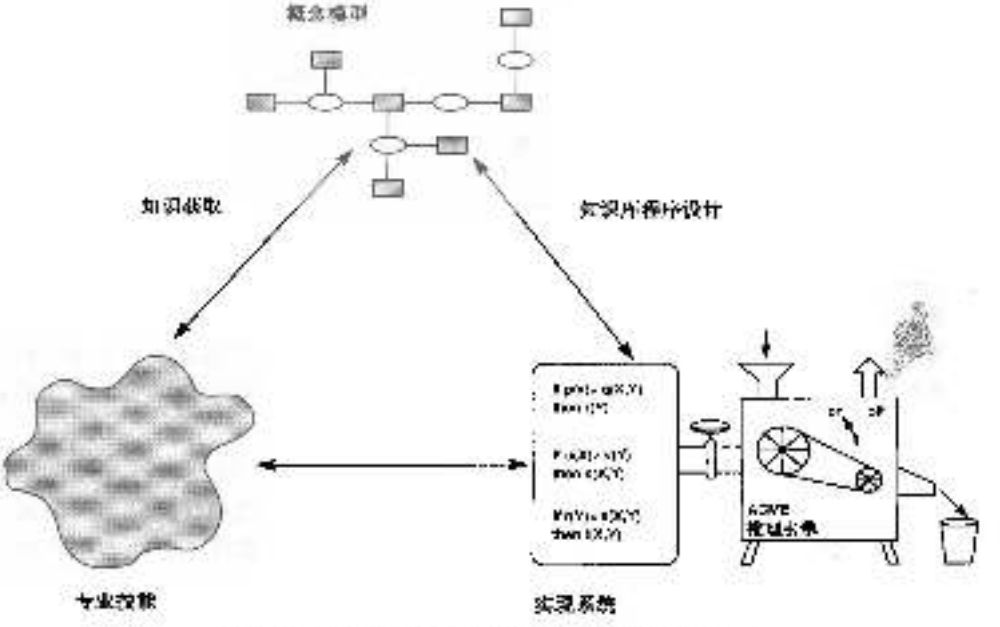


图 8-4　概念（或智力）模型在问题求解中的作用

因为我们感兴趣的大多数问题都是很复杂的，所以我们不应该对这种中间阶段想当然。知识工程师们应该依据软件工程方法学写出文档并公布他们的设想。专家系统应该包括需求文档，不过由于探索性程序设计的局限，专家系统的需求应该随着原型的发展而发展。数据词典、状态空间的图形表示以及代码本身的注释都是这个模型的组成部分。通过公开这些设计决策，可以大大减少在实现和维护程序过程中所出现的错误。

知识工程师应该保存和领域专家的谈话记录。很多时候，随着知识工程师对该领域理解的加深，他们会对该领域产生新的解释或者发现新的信息。记录和解释文档对评估设计决策和检验原型有着重要的价值。总而言之，模型在知识形式化的过程中起到了中介的作用。表示语言的选择对知识工程师的领域模型有很大的影响。

概念模型并不是可以在计算机上直接执行的形式代码。它只是一种中间设计结构，一种用来约束和编撰人类技能的模板。如果知识工程师使用的是谓词演算模型，那么他可以从一些简单的代表典型情况推理状态的网络表示开始。只有经过进一步的改进之后，这些网络才会变成显式的“如果……那么……”规则。

在概念模型中经常遇到的问题有：问题解是确定的还是基于搜索的？推理是数据驱动的（可能带有“产生 - 检验”的风格）还是目标驱动的（建立在少量可能假设的集合基础上）？问题域是已被深入理解而且能够提供深入描述模型的，还是所有问题 - 求解知识实质上都是启发式的？我们可以使用过去问题的例子以及它们的解来直接解决将来的问题，还是必须先把这些例子转变成通用的规则？知识是精确的还是“模糊”和近似的（要对确定性进行数字评估）（见第 9 章）？推理策略是否允许对问题域做出稳定的结论，或者说是否需要对系统内的变化和不确定性进行非单调推理，即对以后可能变化的领域做出断言的能力（见 9.1 节）？最后，领域知识的结构是不是不适合基于规则推理，而要使用像神经网络和遗传算法这样的候选的表示模式（见第四部分）？

概念模型中还应该考虑最终用户的需要：他们对最终程序的期望？他们的专业水平：新手、中等还是专家？解释到什么程度合适？什么样的界面最能满足他们的需要？

有了对这些（和其他）问题的答案、从领域专家那里获取的知识以及相应的概念模型，我们便可以开始开发专家系统。因为产生式系统（最先是在第 6 章介绍的）为组织和应用知识提供了很多内在的强大支持，所以在基于规则的专家系统中经常把它作为知识表示的基础。

8.2　基于规则的专家系统

基于规则的专家系统把求解问题的知识表示为“如果……那么……”形式的规则。这种方法所对应的就是图 8-1 所示的体系结构，是在专家系统中表示域知识的最古老技术之一。它也是最自然的而且仍然在实际的和试验性的专家系统中广泛应用的技术之一。

8.2.1　产生式系统和目标驱动问题求解

可以从第二部分介绍的产生式系统模型角度来理解基于规则专家系统的体系结构。二者的相似之处从这个比喻可见一斑：产生式系统是现代专家系统体系结构的智能先驱，通过应用产生式规则可以提炼对特定问题域的理解。当 Newell 和 Simon 开发产生式系统时，他们的目标就是对人类在问题求解中的行为建模。

如果把图 8-1 中的专家系统体系结构看成是一个产生式系统，那么针对特定领域的知识库就是产生式规则集合。在基于规则的系统中，这些条件和动作对被表示为“如果……那么……”形式的规则，规则的前提（“如果”部分）对应于条件；结论（“那么”部分）对应于动作：当

条件被满足时，专家系统便执行断言结论为真对应的动作。可以把针对实例的数据放在工作内存中。推理引擎实现了产生式系统的识别－动作循环；控制方法可以是数据驱动的，也可以是目标驱动的。

对于许多问题域来说，使用正向搜索方法更加自然。例如，在解释问题中，问题的大多数数据在一开始便已给出，而且很难归纳出假设和目标。这暗示我们应该使用正向的推理过程，把现有数据放入工作内存，然后系统搜索解释，参见 3.2 节。

在目标驱动的专家系统中，起始时便把目标表达式放入工作内存。系统把规则的结论和目标匹配，找到匹配的规则后便把它的前提放入工作内存。这相当于把问题的目标分解为更小的子目标。然后在下一次迭代中把这些结论变成新的目标与规则的结论匹配，继续这个过程。因此这个系统是从原始目标开始反向工作的，直到工作内存中的所有子目标都已知为真，说明这个假设被证实。因此，专家系统中的反向搜索大体对应于人类求解问题时的假设检验过程，参见 3.2 节。

在专家系统中，可以通过向用户询问信息来求解子目标。某些专家系统允许系统设计者指定哪些子目标可以通过询问用户求解。更简单的做法就是询问知识库无法匹配的所有子目标，也就是，如果系统无法推理子目标的真实性，那么它便询问用户。

下面举一个分析汽车传动系统故障的专家系统例子，以此来说明目标驱动的问题求解方法（可以向用户询问）。这不是一个完整的诊断系统，因为它仅有四条非常简单的规则。我们的目的是通过这个例子来说明如何追索规则、集成新数据以及使用解释工具。

规则 1：如果
发动机在抽油而且发动机会旋转，
那么
火花塞有问题。

规则 2：如果
发动机不旋转而且灯不亮，
那么
电池或电缆有问题。

规则 3：如果
发动机不旋转而且灯亮，
那么
启动马达有问题。

规则 4：如果
油箱中有油而且化油器中有油，
那么
发动机在抽油。

如果要通过目标导向的控制体系来运行这个知识库，那么只要像图 8-5 所示的那样把顶层目标（问题在 X）放入工作内存。X 是一个可以与任何短语匹配的变量，例如“问题在电池或电缆”；当问题被解决时，它会绑定在解上。

有三个规则（规则 1、规则 2 和规则 3）和工作内存中的表达式匹配。如果我们优先考虑最小编号的规则，那么规则 1 便会被激发。这使 X 被绑定到一个新的值“火花塞”上，而且规则 1 的前提被放入工作内存，如图 8-6 所示。这样系统便选择了“火花塞损坏”这个可能假设。也可以把这个过程看成是系统选择了与或图的一个或分支（见第 3 章）。

注意规则 1 有两个前提，要证明结论为真必须使这两个前提都被满足。这相当于搜索图中的与分支，即把问题（分析“问题是否在火花塞”）分解为两个子问题（分析“发动机是否在抽油”和“发动机是否旋转”）。这样便激发了规则 4，因为它的结论与“发动机在抽油”匹配，于是它的前提被放入工作内存，如图 8-7 所示。

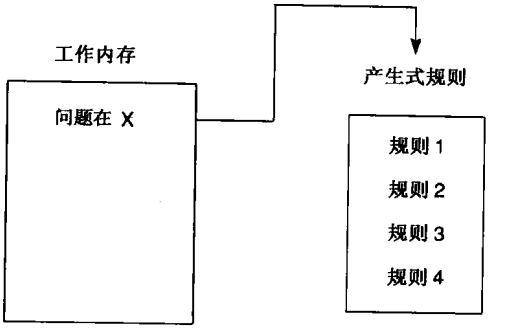


图 8-5　用来诊断汽车故障的产生式系统的初始状态

到目前为止，工作内存中有三项与任何规则的结论都不匹配。这种情况下，我们的专家系统会直接向用户询问这些子目标。如果用户确认这三项都为真，那么系统便会成功地断定这辆汽车不启动的原因是火花塞坏了。在寻找这个解的过程中，系统搜索了图 8-8 所示的与或图的最左分支。

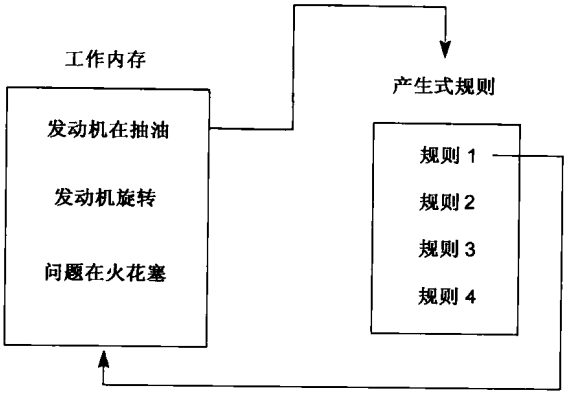


图 8-6　图 8-5 中的产生式系统在规则 1 被激发后的状态

当然这是一个非常简单的例子。不仅关于汽车的知识很有限，而且忽略了很多重要的实现要素。规则是用口语表示的，而不是用形式语言。当找到解时，真正的专家系统会告诉用户它的诊断结果（我们的系统只是简单地停下来）。另外，还应该维护足够的推理过程信息，以允许必要的回溯。在我们的例子中，要是在判断火花塞故障时失败了，那么就必须先回到最顶层，然会再试验规则 2。注意这一信息是隐含在工作内存的子目标排序中的，参见图 8-7 和图 8-8。尽管这个例子很简单，但是它说明了基于产生式系统的搜索方法和与或图表示在基于规则的专家系统中的重要性。

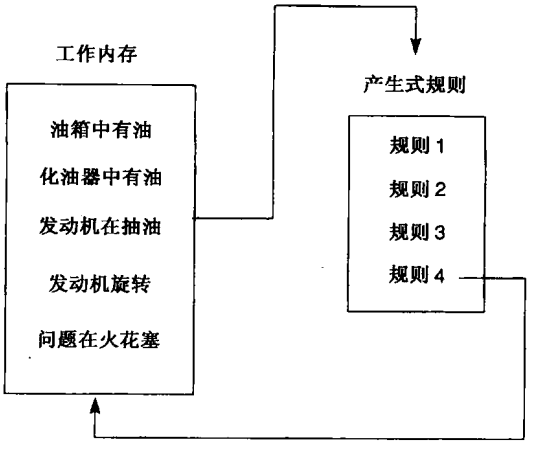


图 8-7　规则 4 被激发后的状态

注：注意这种缩小目标的方法是以堆栈为基础的

前面我们曾经强调专家系统必须具有开放性、易修改性以及启发性。产生式系统体系结构可以很好地满足所有这些要求。以易修改性为例，产生式规则的语法独立性（每个规则是一个可以独立修改的知识块）可以很好地支持这一特征。不过从各个规则的含义是关联的这个意义上来说，语义约束是存在的。所以在编辑和修改过程中一定要考虑各规则语义的一致性。下面我们讨论解释的生成和推理的控制。

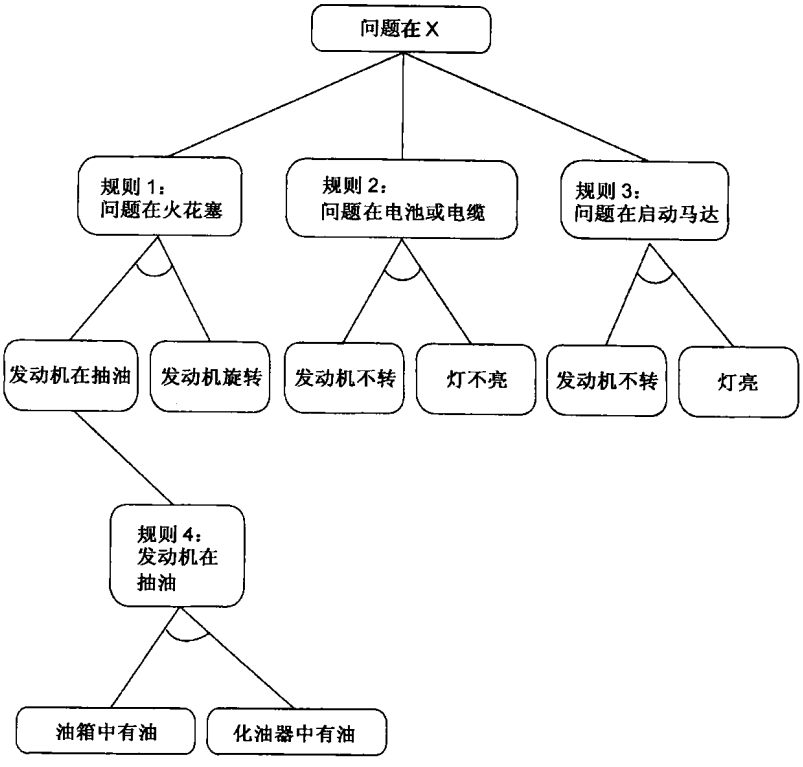


图 8-8　诊断汽车例子中搜索的与或图

注：规则 4 的结论与规则 1 的第一个前提匹配

8.2.2　目标驱动推理中的解释和透明性

产生式系统实现了图搜索。图 8-1 中的解释例程跟踪图搜索的踪迹，并用这些跟踪信息来回答用户的查询。产生式规则本身记录了推理过程的每一步。

通常，基于规则的专家系统要回答的问题有两类：why 和 how。why 查询发生在程序向用户询问信息时，意思是“你为什么要这个信息?”。答案就是产生式系统正在试图激发的当前规则。how 查询对应于“你是如何得到这个结果的?”，答案是得出目标结论的规则序列。

假定前面的汽车专家系统具有解释机制，并假定咨询过程已经到了图 8-7 中工作内存所示的阶段。下面的对话是从计算机向用户询问工作内存中的目标开始的。用户反馈用黑体来表示。

油箱中有油吗?

是的

化油器中有油吗?

是的

发动机旋转吗?

为什么

这里用户输入了一个 why 查询，也就是“你为什么需要知道发动机是否旋转?”系统是使用当前规则的字面表示来回答的：

已经确认：

1. 发动机在抽油，

所以，如果

2. 发动机旋转，

那么就是火花塞有问题。

因为用户对前两个问题的回答都是肯定的，所以可以激发规则 4，这确认了规则 1 的前提。当用户提出 why 查询时，系统正在努力解决规则 1 的前提。注意系统对其行为的解释仅仅是把当前正在考虑的规则重新表述一下。不过，这已经对程序为什么要知道“发动机是否旋转”提供了足够的解释，系统指出它已经得到了“发动机在抽油”这个结论。假定用户又问：

是怎样得出“发动机在抽油”这个结论的?

那么系统便会告诉用户它是如何归纳出“发动机在抽油”的。这个回答就是得出这个结论的推理过程，只要从目标开始沿着支持它的规则反向工作就可以了：

这是从规则 4 得出的：

如果

油箱中有油，而且

化油器中有油

那么

发动机在抽油

油箱中有油是用户给出的

化油器中有油是用户给出的

可见，产生式系统体系结构为解释提供了关键基础。控制循环每执行一次都选择并激发一个规则。在每次循环后可以停止程序并对其进行观察。因为每个规则表示了求解问题的一个完整知识块，所以当前规则为解释提供了背景。把这种产生式系统方法和更传统的程序结构进行对比就可以看出它的优势：如果在中间执行过程中停止 C 或 C + + 程序，那么很难相信刚才的说法还有什么意义。

总而言之，基于知识系统通过显示正准备激发的当前规则来回答 why 查询；通过列出产生目标或子目标的推理过程来回答 how 查询。尽管这种机制在概念上来看非常简单，但是它表现出了强大的解释能力，只要知识库是按逻辑方式组织的。Java、LISP 和 Prolog 这些辅助资料说明了如何使用堆栈和证明树来实现这种解释。

要使解释表现出很强的逻辑性，不仅仅要求可以在知识库找到正确的答案，而且还要求每个规则都对应于问题求解过程的单一逻辑步骤。如果知识库把几个步骤合并到一个规则中，或者随意地拆分规则，那么可能虽然得到了正确的答案，但是却看起来很模糊、独断，或和 how 或 why 查询没有逻辑对应性。这不仅降低了用户对系统的信任度，而且也使程序难以理解和修改。

8.2.3　利用产生式系统进行数据驱动推理

8.2.1 节介绍的汽车诊断例子演示了如何使用产生式系统来实现目标驱动搜索。搜索的方式也是深度优先的，因为它沿着规则库中的一个子目标搜索完后才搜索下一个并列目标。不过，正如在 6.3 节中所看到的，产生式系统也是数据驱动推理的理想体系结构。例 6.3.1 演示了针对 8 格拼图游戏问题的数据驱动搜索过程，例 6.3.2 和例 6.3.3 是针对骑士周游问题的。在这些例子中，我们都是通过取知识库中发现的第一个规则来解决冲突的，然后沿着这个规则的结果搜索下去。这使搜索具有深度优先的味道，不过没有任何机制（比如回溯）来处理搜索空间中的“死端”问题。

宽度优先搜索在数据驱动推理中更加常见。其算法非常简单：就按照规则库中的顺序把工

作内存中的内容和规则库中每个规则的条件进行匹配。如果工作内存中的数据支持激发某个规则，那么便把其结果放入工作内存，然后把控制移向下一个规则。一旦已经考虑过了所有规则，搜索便再次从规则集合的头开始。

举例来说，考虑汽车诊断的例子和 8.2.1 节中的规则。如果构成规则前提的某个信息不是其他任何规则的结论，那么便认为这个信息是“可询问的”。例如，规则 1 前提中的“发动机在抽油”就不是“可询问的”，因为这是另一个规则（规则 4）的结论。

下面针对上一节的例子介绍一下宽度优先数据驱动搜索，开始状态和图 8-5 类似，只不过工作内存中没有任何信息，如图 8-9 所示，然后我们分析四个规则的前提以发现“可询问的”的信息。“发动机在抽油”这个前提不是“可询问的”，所以控制转向规则 2。“发动机不转”是可询问的。假定对这个问题的回答是假，那么“发动机旋转”被放入工作内存，如图 8-10 所示。

因为规则 2 的两个与前提中的第一个为假，所以规则 2 失败了，于是系统考虑规则 3，但是第一个前提也失败了。在规则 4，两个前提都是“可询问的”。假定对这两个询问的回答都是肯定的，那么“油箱中有油”和“化油器中有油”便会被放入工作内存，并得到结论“发动机在抽油”。

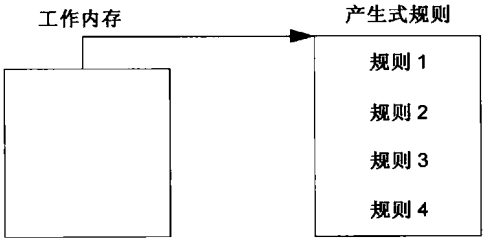


图 8-9　用产生式系统进行数据驱动推理时的初始状态

到目前为止，系统已经考虑过所有规则所以搜索返回到起始位置，准备用工作内存中新的内容第二次依次考虑规则。从图 8-11 可以看到，工作内存与规则 1 是相匹配的，因此它的结论“问题在火花塞”被放入工作内存。在这个例子中，不再有其他匹配的规则，因此问题求解过程结束了。图 8-12 画出了以上搜索过程的示意图，工作内存（WM）的内容对应于图中的结点。

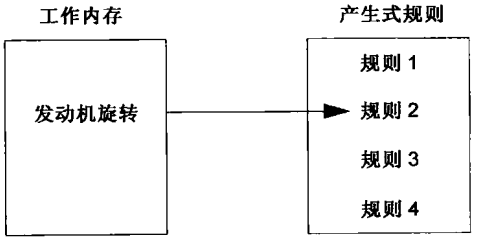


图 8-10　在规则 2 因第一个前提为假而失败后的产生式系统

对上面例子中宽度优先搜索策略的一种重要改进就是所谓的*机会搜索*。这种搜索策略很简单：每当激发规则并推断出新的信息时，便把控制转移到以新信息为前提的那些规则。这使新的推断信息（搜索不会因“可询问”前提的结果而改变）成为寻找下一个要激发规则的控制动力。之所以称其为机会搜索是因为新的推断信息左右了搜索的方向。因为规则排列的偶然性，刚才给出的简单例子恰好也是“机会的”。

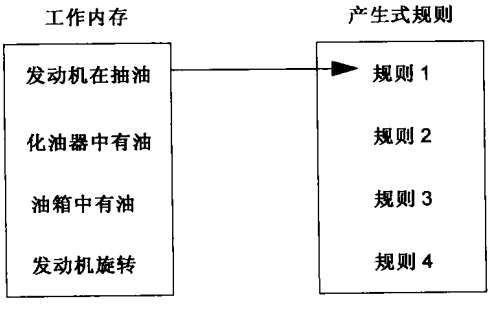


图 8-11　在考虑规则 4 后，数据驱动的产生式系统开始第二轮扫描规则

下面对数据驱动系统的解释性和透明性做几点评价，并以此总结本节的内容。第一，和 8.2.1 节及 8.2.2 节的目标驱动系统相比，数据驱动推理的搜索不那么集中。导致这一点的原因很明显：在目标驱动系统中，推理就是追索某个特定的目标；这个目标被分解为支持顶层目标的子目标，而且子目标还可以被进一步分解。因此搜索总是由这个目标和子目标层次来导向的。在数据驱动系统中这种

目标面向性不存在了。相反，搜索的移动方向仅依赖于规则的排列顺序和发现的新信息。结果，搜索的过程往往看起来非常散乱、不集中。

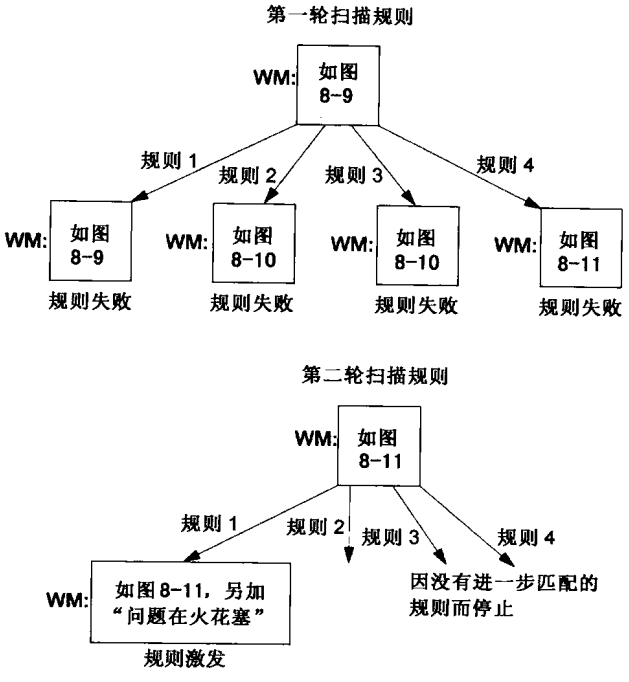


图 8-12　8.2.1 节数据驱动宽度优先搜索规则集的搜索图

注：结点是通过工作内存（WM）中的内容来描述的

第二，这也是第一点所直接导致的结果，不论是搜索的哪个阶段，可以给用户的解释都是非常有限的。从某种意义上来说，系统的可说明性仅限于单个的规则，因为当用户询问为什么需要某个信息时（即 8.2.2 节中所说的 why 查询），系统只能给出正在考虑的规则。不能做出进一步的解释，除非向系统（比如采用机会搜索策略的系统）中加入显式的规则跟踪机制。数据驱动搜索的散乱性使得难以做到这一点。最终，当找到目标后，要得到一个对该目标的完全解释也是很困难的。或许惟一的方法是把工作内存中的内容或激发规则的列表用作一种局部性的有限解释。但是，这也无法提供像目标驱动推理中那样的集中一致的可追索性。

8.2.4　专家系统的启发和控制

因为知识库和推理引擎是分离的，而且推理引擎所提供的控制体制是固定的，所以程序员控制搜索的一种重要方法就是改变知识库中规则的结构和顺序。这种对规则集的微观管理提供了一种非常重要的机会，特别是因为专家级的问题求解所需的控制策略往往是针对领域的而且是知识密集型的。虽然一个形式为“如果 p、q 和 r，那么 s”的规则类似一个逻辑表达式，但是它也可以被解释为求解问题的一系列过程和步骤：“要做 s，先要做 p，然后做 q，然后再做 r”。刚刚给出的 8.2 节中的例子隐含了规则和前提排序的作用。

这种解释规则的过程方法是实践中使用知识的一个关键策略，很多时候它反映了人类专家的求解策略。例如，我们可以对规则的前提进行排序使最可能失败的或最易于确认的排在最前面。这样就可以尽早地排除无关的规则（因而也就缩小了搜索空间）。在汽车的例子中，在询问发动机是否旋转之前首先试的是规则 1 以判断发动机是否抽油。这很不高效，因为在试图判断发

动机是否抽油时涉及了另一个规则而且因此向用户询问了两个问题。如果把规则 1 的两个前提颠倒顺序，那么对查询“发动机是否旋转”的否定就会排除这个规则，从而也就不需要分析包含条件，这便使系统更高效。

在检查发动机是否抽油之前判断发动机是否旋转也更有意义：如果发动机不旋转了，那么它是否抽油都没关系。在规则 4 中，要求用户在检查化油器是否有油前检查油箱。因为在这种情况下先检查更简单的好一些。这里要强调的一点是，如果要使整个系统更加高效，那么必须考虑各种因素：规则顺序、前提的组织、检验的成本、回答检验可以消除的搜索量、大多数实例的发生方式，等等。

因此，规则顺序的规划、规则前提的组织、不同检验的成本通常都是基本的启发。在大多数基于规则的专家系统中，这些启发反映了人类专家所采取的方法，而且实际上可能产生错误的结果。在前面的例子中，如果发动机在抽油而且发动机旋转，那么问题可能在分电器而不是火花塞。

数据驱动推理为推理控制带来了一些额外的问题和机会。其中包括高层的启发，比如 6.3.3 节介绍的折射、最新性（机会搜索）和特殊性。领域针对性更强的方法根据求解过程的不同阶段来对规则集分组。例如，在诊断汽车的问题中，我们可以创建 4 个相互独立的阶段：1）组织条件；2）数据采集；3）数据分析（被诊断的汽车可能有多个故障）；4）报告结论并推荐修理方法。

可以这样实现这种分阶段的问题求解方法：先为每个阶段创建一个描述符，然后把这个描述作为第一个前提放入属于该阶段的所有规则中。举例来说，开始时把断言“组织条件”放入工作内存。如果工作内存中没有其他的阶段描述符，那么就仅有前提中包含“组织条件”的规则才会被激发。当然，这应该是每个这样规则的第一个前提。我们可以这样转到下一个阶段：让组织阶段要激发的最后一个规则删除现阶段是“组织条件”的事实，并断言现阶段是“数据采集”这一事实。数据采集阶段的所有规则的第一个前提都是“如果阶段 = 数据采集”。当数据采集阶段完成时，它的最后一个规则将收回这个断言，并把“数据分析”这个事实放入工作内存中，目的是仅匹配那些前提以“如果阶段 = 数据分析……”开始的那些规则。

在截至目前的讨论中，我们已经描述了穷举考虑规则库的产生式系统的行为。尽管这样做的搜索代价很高，但是它捕获了产生式系统的预期语义。不过，有很多算法可以通过搜索特别有用的规则来优化这个过程，例如 RETE（Forgy1982）。本质上讲，RETE 算法把规则汇编成一种网络结构，这样系统便可以直接通过指向规则的指针来把规则和数据进行匹配。这种算法大大加快了执行速度，特别是对比较庞大的规则集合，同时也保持了本节已经描述的语义行为。

归纳来讲，规则是专家系统中表示知识的最古老方法，而且仍然是建立知识密集型问题求解器的重要技术。专家系统的规则捕获了人类专家在实践中所使用的知识，因此，规则往往融合了理论知识、经验性启发以及处理古怪实例和其他异常情况的特殊规则。很多情况证明这种方法是有效的。尽管如此，强启发系统可能因为碰到任何现有规则都不适用的问题，或者把启发规则误用于不恰当的情况而失败。人类专家不会受到这种问题的困扰，因为他们对问题域的理解更加深入，这使得他们可以智能地应用启发规则，或者在新的情况中根据“第一原则”来推理。基于模型的方法（8.3.1 节介绍）就是为了使专家系统具有这种能力和灵活性而做的一种努力。

根据实例学习的能力是知识密集型问题求解器要模拟的另一项人类技能。基于案例的推理程序（8.3.3 节介绍）所维护的知识库是由已经解决问题的样例（即案例）组成的。当面临一个新问题时，这种推理程序从存储的案例中选择和当前问题类似的案例，然后把它的求解策略应用到这个问题。法律中用以前案例作论据便是基于案例推理的最常见实例。

8.3　基于模型系统、基于案例系统和混合系统

8.3.1　基于模型推理简介

人类的专业技能是相当复杂的，它融合了理论知识、以经验为基础的问题求解启发、过去的问题实例和解、感知和解释技巧以及我们尚未理解因此仅能称其为直觉的其他能力。通过多年的经历，人类专家建立了非常强大的规则，可以处理通常遇到的各种情况。这些规则通常是被高度“编译”的，其形式把可观察的符号与最终诊断直接关联起来，隐藏了更深层的解释基础。

举例来说，MYCIN 专家系统可以根据像“头痛”、“恶心”或“发烧”这样的可观察症状做出诊断。尽管这些参数表明了患者得了某种疾病，但是把这些症状与诊断联系起来的规则并没有反映任何更深层的生理角度解释。MYCIN 的规则指出诊断结果是传染病，但是并未解释致病的原因。更深入的解释方法应该检查是否存在传播媒介，注意是否导致细胞膜感染；检查颅内压力；然后推断出这些检验结果与观察到的头痛、体温升高和恶心症状的因果联系。

下面举一个分析半导体故障的基于规则专家系统的例子，该系统根据以下症状诊断电路故障：器件上的污点（可能表明这个部件已经烧掉了）、类似设备的故障历史或者用电子仪表检查器件的内部特征。然而，把观察情况和诊断结果联系起来的规则失去了深入分析设备结构和功能的好处。更鲁棒的、可深入解释的方法是从这个电路物理结构的详细模型以及描述每个部件和部件间预期行为的公式着手。它把诊断建立在来自设备不同位置的数字读数上，使用这些数据和它的电路模型来判断确切的故障点。

因为第一代专家系统依赖于从人类专家那里获得的启发性规则，所以具有很多局限性（Clancy 1985）。如果问题实例与系统的启发不匹配，那么即使通过理论分析可以找到解，这个解也是失败的。很多时候，专家系统把启发应用于不适当的情况，例如，较深入地理解问题可能预示着一个不同的过程。这便是基于模型方法所要解决的不足。如果一个基于知识的推理程序把分析直接建立在物理系统的特征和功能之上，那么就称其为基于模型系统。基于模型的推理程序在设计和使用中都创建一个软件来模拟（经常被称为“定性”）要被理解的或修理对象的功能（当然，还有其他类型的基于模型系统，特别是第 9 章要介绍的基于逻辑的和随机的基于模型系统）。

最早的基于模型推理程序出现在 20 世纪 70 年代中期，80 年代后逐渐成熟（Davis and Hamscher 1992）。值得注意的有趣的一点是，最早的一些研究是出于教学目的而创建各种物理设备（比如电子电路）的软件模型（deKleer 1976，Brown et al. 1982）。在这些早期的教学系统中，设备或电路的特征说明是以规则集（例如基尔霍夫定律和欧姆定律）反映的。这些教学系统既检验了学生关于设备和电路的知识，又向学生传授了他们可能忽视的知识。规则既表示了硬件的功能，同时又是向学生传输这种知识的媒介。

基于模型推理程序从这些早期的教学系统（其任务既是对系统的功能建模又是教授这些功能）逐步转向查找故障的系统。在查找物理系统中的故障时，模型会产生一系列预期的行为，然后通过分析预期行为和观察到的行为之间的差异来发现故障。基于模型系统会告诉用户：期望行为是什么、观察情况与期望情况的差异以及系统是如何根据这些差异推断故障的。

定性的基于模型推理包括：

1）对设备中每个组件的描述。这些描述可以模拟组件的行为。

2）对设备内部结构的描述。这些描述通常表示出各个部件以及它们的互连方式，应该具有模拟部件间相互作用的能力。所需内部结构知识的程度依赖于应用的深度和预期诊断的层次。

3）诊断特定问题时需要观察设备的实际工作情况，通常是输入和输出测量值。输入输出测量是最容易获得的，但在实际过程中，也可能还需要测量其他指标。

接下来的任务就是根据观察到的行为判断哪个部件可能已经出现了故障。这需要有额外的规则来描述不同部件及其互连的已知故障模式。推理程序必须找到可以解释观察到的行为的最可能故障。

可以使用多种数据结构来表示模型中的因果和结构化信息。许多基于模型程序设计者使用规则来反映设备的因果关系和功能。规则也可能用于反映部件之间的关系。面向对象系统也是一种反应设备和部件结构的出色工具，它用对象槽来表示设备或部件的状态，用对象或类方法来表示其功能。

为了更具体地了解模型的设计和评估，下面考虑几个分析设备和电路的例子（摘自 Davis and Hamscher 1992）。我们用一系列表达式来表示设备行为，这些表达式描述了设备终端值之间的关系。对于图 8-13 所示的加法器来说，将有三个表达式：

如果我们知道 A 和 B 的值，那么 C 的值是 A + B（实线）。

如果我们知道 C 和 A 的值，那么 B 的值是 C − A（虚短线）。

如果我们知道 C 和 B 的值，那么 A 的值是 C − B（虚点线）。

并非一定要使用代数形式来表示这些关系。也可以使用关系数组或者使用 LISP 函数来表示这些约束，效果也完全等价。我们的目标就是表示出加法器功能这个知识。

再举一个例子，考虑如图 8-14 所示的三个加法器和两个减法器连接成的电路。在这个例子中输入值是由 A～E 点给出的，输出值是由 F 和 G 点给出的。期望输出值是用（ ）表示的，实际输出值是用［ ］表示的。我们的任务是判断造成差异的故障在哪里。在 F 点存在问题，期望值是 12，而得到的值是 10。因此我们需要检查这一点的依赖性，它是 ADD-1 的函数，ADD-1 又依赖于 MULT-1 和 MULT-2。这三个设备中一定有一个出了故障。因此有二种可能需要考虑：加法器的行为有问题；加法器的两个输入之一不正确；问题出在这个电路的更深处。

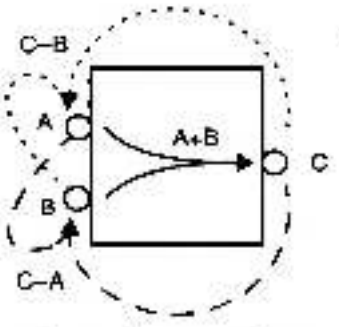


图 8-13　加法器的行为描述
［摘自 Davis and Hamscher（1992）］

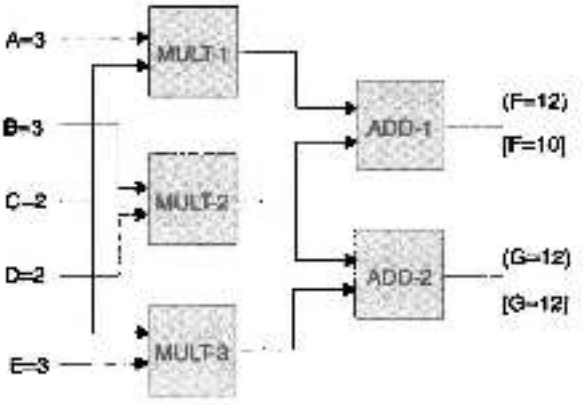


图 8-14　利马信息流动的方向
［改自 Davis and Hamscher（1988）］

不妨从 F 点的结果（10）开始推理，并假定 ADD－1 以及它的输入 X（6）的行为是正确的，那么 ADD-1 的输入 Y 的值必须是 4，但这与 MULT-2 以及输入 B 和 D 的正确行为有冲突。我们已经观察到这些输入而只知道它们是正确的，所以 MULT-2 一定有故障。如果并行推论，那么第二个假设是 ADD-1 是正确的，MULT-1 是有问题的。

继续我们的推理，如果 ADD-1 的第一个输入 X 是正确的而且 ADD-1 本身是正确的，那么第

二个输入 Y 一定是4。如果它是4，那么 G 就是10 而不是12，所以 MULT-2 的输出一定是6 而且是正确的。于是剩下的假设就是 MULT-1 或 ADD-1 有故障，可以通过进一步检验来约束这些设备。

在对图 8-14 所示的情况进行推理时，采取了三个步骤：

1）假设生成：对于给定的故障，推测设备的哪些部件导致了这个故障。

2）假设检验：对于确定的可能故障部件，判断哪个可以解释已经观察到的行为。

3）假设判别：当有多个假设通过了检验阶段时（就像图 8-14 中发生的情况），我们必须决定怎样继续搜集信息以进一步搜索故障。

最后，我们应该注意到在图 8-14 的例子中，我们是假定了仅有一个设备出故障。实际情况并不这么简单，不过单一故障假定是有价值的，而且它在很多时候是正确的、具有启发意义的。

因为定性的基于模型技术是建立在对目标设备的理论理解基础上的，所以它弥补了更依赖启发方法的很多不足。基于模型方法不是根据观察到的现象直接推理因果解释，而是尽可能在因果关系或功能层上表示设备及设备的配置。程序代码既反映了设备的功能，又反映了设备在系统内的依赖性。不过，这种对功能显式建模方法的不足是对知识获取阶段的要求非常高；而且会导致程序比较大、笨重和速度慢。因为启发式方法把典型的案例“编译”成单一的规则，所以很多时候只要系统的其他约束恰当，启发方法就会更加高效。

不过这种方法还存在更深层的问题。和基于规则推理一样，一个系统的模型仅仅是个模型。它毕竟是对系统的一种抽象（详细到某一层次），因此它可能是不正确的。就拿图 8-14 中的输入导线来说，在我们的讨论中，我们认为这些值是给定的、正确的。我们没有分析这些导线本身的状态以及导线另一端和乘法器连接的情况。如果导线短了或者和乘法器的连接有问题怎么办呢？如果用户没有发现这个连接问题，那么模型就和实际设备不匹配了。

所有模型都尽可能描述理想的情况，即期望系统做什么，但系统未必确实这么做。“桥接”故障是指系统连接点处的两个导线或两个设备间意外地连接了，就像一个有问题的焊点把两根不该连接的导线连起来一样。大多数基于模型推理系统难以推测桥接故障，因为用来判断异常的搜索方法和模型使用的假定是预先设置的。桥接故障相当于在原始设计之外出现了“新的”导线。因此，在基于模型方法中存在一个隐含的“封闭世界假定”（见 9.1 节）：假定模型的结构描述是完整的，不存在不属于模型中的事物。

尽管存在这些不足，基于模型推理仍是一种重要的知识工程工具的补充。研究者们一直在探索各种诊断方法：不仅考虑人类专家是如何做到那么高效的，而且还考虑如何才能在机器上实现更好的算法（Stern and Luger 1997，Pless et al. 2006）。

8.3.2　基于模型推理：来自 NASA 的例子

NASA（美国航空航天局）为了支持其太空计划，开发了一系列智能太空探测器用来自动探索太阳系（Williams and Nayak 1996a，Bernard et al. 1998）。这一努力是从 1997 年为第一个探测器设计软件开始的，1998 年发射了太空 1 号。为了在苛刻的太空旅行条件下能够成功飞行多年，飞船必须能够对故障做出反应，迅速地重新配置好控制体制，并规划在以后的飞行中避免这些故障再次发生。为了以可接受的代价快速完成重新配置，系统必须能够从每一类模块中选取一个，并迅速整合到一起自动生成飞行软件。NASA 预料到潜在的故障现场和可能的反应数将过于庞大，以至于在飞行前枚举出所有偶然事故是不可能的。因此，探测器必须自动全盘考虑各种重新配置选项的所有后果。

Livingstone（Williams and Nayak 1996b）是为基于模型的自配置（self – configuring）自动反

应系统实现的一个内核。Livingstone 所用的基于模型推理表示语言是命题演算，是对第 2 章介绍的一阶谓词演算的发展，也是用在基于模型诊断中的传统表示语言。Williams 和 Nayak 感到，根据他们以前建立基于冲突的命题逻辑快速诊断算法的经验（de Kleer and Williams 1989），开发出可以在感知/响应循环中进行演绎推理的快速反应系统是可能的。

人们对基于模型推理的一种长期观点一直是使用一个单一的中央模型来支持不同的工程任务。对基于模型自动系统来说这意味着要使用单一模型来支持多种多样的执行任务。包括跟踪开发计划（见 8.4 节）、确认硬件模式、重新配置硬件、探测异常、故障恢复。Livingstone 利用单一模型和一套单一核心算法来自动化这些任务，因此为基于模型问题求解器的设计方法的形成做出了重要贡献。

图 8-15 是 Cassini 太空探测器主引擎的简化示意图，Cassini 是迄今为止建立的最复杂的探测器之一。这个引擎是由一个氦箱、一个油箱、一个氧化剂箱、一对主引擎、多个调节器、多个自锁阀、多个热阀和导管组成的。氦箱为两个推进箱加压，同时调节器动作将很高的氦压降低到较低的工作压力。当通往主引擎的推进路径打开时（阀门图标是空心的），被加压的箱体将油和氧化剂压入主引擎，并在那里混合、自燃，从而产生推力。热阀仅能改变状态一次，要么从打开到关闭，要么是相反情况。其功能是把主引擎子系统的各个部分隔离开，即永久的隔离故障部件。自锁阀是利用阀驱动器（图 8-15 中未画出）和感知主引擎产生推力的加速计（Acc）来控制的。

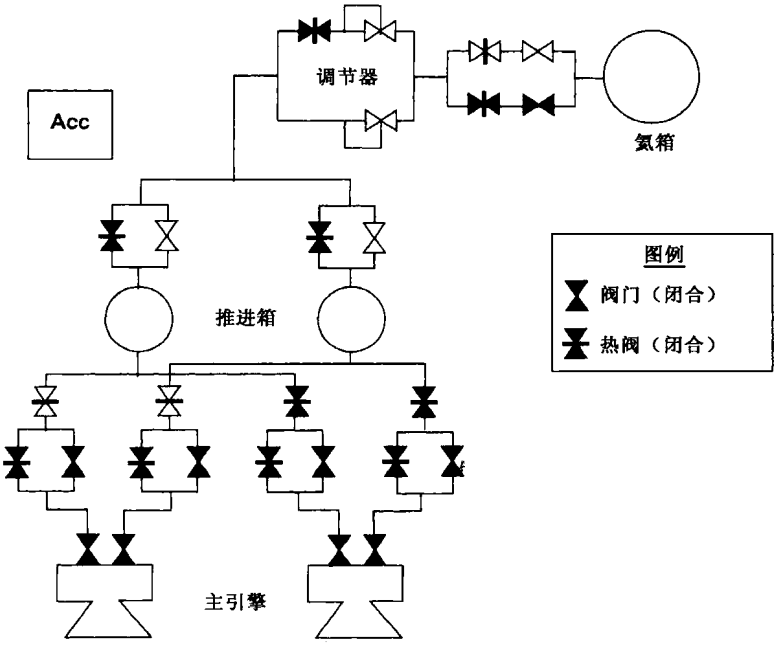


图 8-15　Livingstone 推进系统的简化示意图

[摘自 Williams and Nayak（1996b）]

根据图 8-15 所示的结构布局，可以使用多种不同的配置来实现产生推力这一顶层目标：可以通过两个主引擎中的任一个来产生推力，而且有很多打开通往主引擎路径的方式。举例来说，可以通过打开通往左侧引擎的自锁阀来产生推力，或者通过引发一对热阀并打开一系列通往右侧主引擎的自锁阀。还有一些其他的配置方法，对应于不同的热阀引发组合。不同的配置具有不同的特征，因为热阀引发是不可恢复的动作，而且引发热阀需要的力量比开关自锁阀要大得多。

假定主引擎子系统已经被配置为使用左侧的主引擎来提供推力，即通往它的自锁阀是打开

的。举例来说，如果这个主引擎因为过热出故障了，并因此无法提供必需的推力了。为了保证在这种情况下仍然可以提供所需的推力，探测器必须被切换到一种新的配置，使用右侧的主引擎来提供动力。理想情况是通过引发通向右引擎的两个热阀并打开相继的自锁阀来实现这个目标，而不要引发额外的热阀。

配置管理程序（configuration manager）不断努力选取最佳的配置，使飞船以最低的开销实现不断变化的顶层目标。当飞船因为故障偏离选定的配置时，管理程序会分析感知器数据来识别当前的飞船配置，并把飞船转移到一种仍然可以实现预期配置目标的新配置。配置管理程序是一个离散的控制系统，它保证了飞船配置总是可以达到配置目标所定义的指标。8.4.4 节将介绍支持配置管理程序的规划算法。

要对一个系统的配置（和自动重新配置）进行推理需要建立以下概念：运行和故障状态、可修复故障和配置变化。NASA 把这些概念表示为状态空间图：可修复故障是故障状态和正常状态之间的过渡；配置改变是正常状态之间的变化；故障是正常状态到故障状态的过渡。

Williams 和 Nayak（1997）把自动系统看成是配置管理程序和顶层规划程序的组合。规划程序（详见 8.4 节）产生一系列硬件配置目标。配置管理程序使应用领域的过渡系统（这里是飞船的推进系统）按照要求的轨迹运行。因此，配置管理是通过感知并控制过渡系统的状态来实现的。

基于模型的配置管理程序就是使用一个过渡系统的规范来计算理想控制值序列的配置管理程序。在图 8-15 的例子中，对每个硬件部件用一个过渡系统组件建模。通信组件（图中是用导线（连线）表示的）是用对应过渡系统组件之间的共享变量来建模的。

配置管理程序利用这个模型来推断当前状态并选择满足配置目标的最佳控制动作。而不是使用简单的尝试然后再纠正误差，这在一旦出错便会导致灾难的情况下是至关重要的。基于模型的配置管理程序用两个步骤来决定理想的控制序列：模式估计和模式重配置（图 8-16 中的 ME 和 MR）。ME 不断产生与过渡模型、系统控制序列和感知到的值相一致的所有系统轨线。MR 利用过渡模型和 ME 产生的轨线来决定控制值使所有预测出的轨线实现下一状态的配置目标。ME 和 MR 都是可响应的。ME 可以根据以前状态的知识和观察到的当前状态情况推断当前状态。MR 仅考虑可以实现下一状态内配置目标的动作。

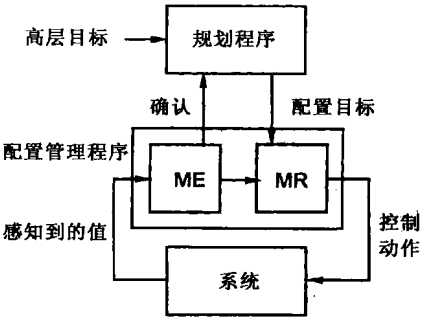


图 8-16　基于模型的配置管理系统
［摘自 Williams and Nayak（1996b）］

下一节将考虑基于案例推理，通过这种知识密集型技术我们可以复用过去的经历来处理问题域中的新情况。在 8.4 节中将讨论规划，并回过头来讨论 NASA 控制的例子。

8.3.3　基于案例推理介绍

启发式规则和理论模型是人类专家用来求解问题的两种信息。专家们所使用的另一种强大策略是根据案例推理，案例即过去问题及其解的样例。*基于案例推理*（case-based reasoning，CBR）使用一个显式的问题解的数据库来处理新的问题求解情况。这些解可能是通过知识工程过程从人类专家那里搜集的，也可能是以前基于搜索方法成功或失败的结果。举例来说，医疗训练不能只依赖解剖学、心理学和疾病的理论模型；很大程度上还要靠病例史和医师在治疗其他患者时积累的经验。CASEY（Koton 1988a，b）和 PROTOS（Bareiss et al. 1988）是在医疗领域应用

基于案例推理的两个例子。

律师选择与其当事人情况类似而且可以支持有利判决的过去法律案例来说服法庭这些相似性足以说明应该采取相似的判决。尽管一般法律都是通过民主过程制定的，但是其解释通常都是建立在司法判例基础上的。一项法律在以前的某个情况中是如何解释的对该法律的当前解释是至关重要的。因此，法律推理的一个重要部分就是根据法律判例判决某个特定案例。Rissland（1983）以及 Rissland and Ashley（1987）已经设计出了基于案例推理程序来支持法律辩论。

计算机程序师经常修改旧的程序使其适合新的具有相似结构的情况，从而实现代码的复用。建筑师根据他们关于以前建造的具有感官美而且实用的建筑的知识来设计新的建筑。历史学家使用过去的故事来帮助政治家、官员和市民理解过去的事件和计划未来。根据案例推理的能力是人类智能的一个主要部分。

其他明显使用案例推理的领域包括设计（一个成功作品的某些方面可能也适合某个新的情况）和诊断（过去发生的故障可能再次发生）。硬件诊断是这方面的一个很好例子。这一领域的专家除了使用广泛的电子和机械系统理论知识外，还使用过去诊断中的成功或失败经验来处理当前问题。CBR 已经成为很多硬件诊断系统的一个重要组成部分，包括在地球轨道卫星中维护信号源和电池（Skinner and Luger 1992）以及对半导体分立部件进行故障分析（Stern and Luger 1997）。

CBR 为建立专家系统提供了很多优势。如果我们记录下人类专家对很多问题的解，并让基于案例推理程序选择合适的案例并据此推理，那么知识获取的过程会大大简化。这便使知识工程师省去了根据专家样例建立通用规则的麻烦，而推理程序会通过把规则应用到新情况的过程自动泛化规则。

基于案例方法还可以使专家系统具有从其经历中学习的能力。在找到了一个基于搜索的解后，系统可以把这个解保存起来，这样当下次再有类似情况发生时便不必再次搜索了。在案例库信息中保留以前探索解的成功或失败结果也是非常重要的；因此，CBR 提供了一种强大的学习模型。这方面的一个早期例子是 Samuel（1959，4.1.1 节）的西洋跳棋程序，在这个程序中，通过搜索发现的棋盘布局（即经历）被保留起来，以预防这种布局在以后的博弈中再次发生。

基于案例推理程序共享一种共同的结构。对于每个新的问题，它们采取以下行动：

1）**从内存中检索恰当的案例**。如果一个案例的解可以被成功地应用到新的情况，那么这个案例便是恰当的。因为推理程序不可能预先知道这一情况，所以它们通常使用启发来选取类似于问题实例的案例。不论是人类还是人工智能程序都是基于一些共同的特征来判断相似性：例如，如果两个患者的症状和病史具有很多共同特征，那么他们就有很大的可能患有同一种疾病而且会对同一种治疗方法感觉有效。高效的案例检索还需要对案例存储方式进行组织使之有助于检索。典型情况下，案例是按它们的最显著特征来索引的，这样便可以高效地检索那些与当前问题具有大多数公共特征的案例。标识突出特征的方法依赖于问题的具体情况。

2）**修改检索出的案例以便应用到当前的情况**。通常一个案例推荐了一系列从起始状态转变到目标状态的操作。推理程序必须把保存的解转变成适合当前问题的操作。有时可以使用解析方法，比如在决定自动焊接的合适温度和材料时把存储案例和新问题的参数进行曲线拟合是很常见的。当无法使用案例间的解析关系时，启发性方法可能是合适的，就像硬件诊断中的咨询台那样。

3）**应用转换后的案例**。第 2 步对存储的案例做了修改，但是当应用它时，并不能保证可以得到满意的问题解。这可能需要对解进行修改，并反复重复前三个步骤。

4）**将解以成功或失败记录的形式保存起来供将来使用**。存储新案例需要对索引结构进行更

新。有很多方法可以用来维护索引，包括聚类算法（Fisher 1987）和来自机器学习领域的其他技术（Stubblefield and Luger 1996）。

用来进行基于案例推理的数据结构可能千差万别。在最简单的情况下，案例被记录为一个关系组：一部分数据记录了要被匹配的特征，余下的数据指向求解步骤。还可以把案例表示为更复杂的结构，比如证明树。一种相当常见的案例存储机制是把案例表示为一个很大的“条件 - 动作”规则集合。描述规则条件的事实是被记录案例的最突出特征，组成规则动作的运算就是要用在新条件下的变换。当使用这种规则表示时，可以使用像 RETE（Forgy 1982）这样的算法来组织和优化对恰当案例的搜索。

基于案例问题求解中的最难点是如何选择用来索引和检索案例的突出特征（不论选择什么样的数据结构来表示案例）。Kolodner（1993）和其他一些著作把基于案例问题求解作为规则动态地包含进系统，案例是按照问题求解器的目标和需要来组织的。也就是说，对案例描述的详细分析是根据这些案例被用在解过程中的具体情况进行的。

举例来说，假定卫星在格林威治标准时间 10 : 24 : 35 发生了*通信信号微弱*的问题。通过分析判断能源系统偏低。能源偏低可能是因为太阳能电池板没有正好面向太阳导致的。地面控制程序对卫星的朝向作了调整，于是通信信号恢复正常了。有很多突出特征可以用来记录这个案例，但最明显的是通信信号变弱或能源供应不足。描述这个案例的另一个特征是这个故障的发生时间是格林威治标准时间 10 : 24 : 35。在这个案例中问题求解器的目标和需要表明突出的特征是通信信号微弱和/或能源供应不足；发生时间很可能是无关的，当然除非故障恰好发生在太阳从地平线上消失以后（Skinner and Luger 1995）。

要考虑的另一个关键问题是如何表示像信号微弱或能源不足这样的概念。因为如果精确地描述案例情况，比如用准确的实数来描述信号强度，那么同样的情况也许再也不会发生，所以推理程序很可能把这些值表示为一个实数区间，例如分成多个等级：良好、接近良好、弱和危险警报。

Kolodner（1993）归纳了一系列可能的优先启发来辅助案例的存储和检索。包括：

1）*目标指向优先*。在一些情况下，至少可以通过目标描述来组织案例。检索和当前情况具有相同目标的案例。

2）*突出特征优先*。优先考虑匹配最主要特征或者匹配重要特征数最多的案例。

3）*指定优先*。先寻找尽可能精确匹配的特征，然后再考虑更一般的匹配。

4）*频率优先*。先检查匹配次数最多的案例。

5）*最新性优先*。优先考虑最近使用过的案例。

6）*易适应优先*。先使用最易于适应当前情况的案例。

基于案例推理为专家系统的设计提供了很多优势。一旦知识工程师找到了合适的案例表示方法，那么接下来的知识获取过程就非常简单：只要搜集并存储更多的案例就可以了。很多时候，可以通过历史记录或监视当前操作来实现案例获取，这大大节约了人类专家的时间。

此外，CBR 提出了很多与人类学习和推理有关的重要理论问题。其中最棘手的、最至关重要的问题之一就是*相似性*的定义问题。尽管“相似性是两个案例的共同特征数量的函数”这个概念很合理，但是它掩盖了很多深刻的细微特征。例如，大多数对象和情况都具有无数的潜在描述属性；基于案例推理程序通常是根据很小的检索词汇表来选择案例的。典型情况下，基于案例推理程序需要知识工程师定义一个合适的词汇表来描述那些高度相关的特征。尽管已经有研究可以使推理程序根据它自己的经历来判断相关特征（Stubblefield 1995），但是判断相关性仍然是一个难以解决的问题。

基于案例推理中的另一个重要问题是如何处理存储和计算之间的平衡。基于案例程序获取的案例越多，就会越智能，可以更好地解决不同的目标问题。随着向推理程序加入案例，它的性能确实会提高——直到某一点。问题是随着案例库的不断增长，检索和处理一个合适案例所需的时间也在增长。庞大案例库中的叠概念、噪声以及问题类型的分布等也可能造成其效率下降。解决这个问题的一种方法是仅保存“最佳的”或者“原型”案例，删除那些冗余的或者不常使用的案例；也就是，忽略那些不能证明其有用的案例。Samuel（1959）用来保存西洋跳棋布局的保持算法是这种策略的一个早期例子。不过，通常并没有明确的做法来自动化这种决策；这依然是一个正在研究的领域（Kolodner 1993）。

对基于案例推理程序来说，对推荐的解做出自动解释也是很难的。当用户询问为什么选择这个解来处理当前情况时，系统可以提供的惟一解释是说这种特定方法以前有效过。也可以根据当前情况和存储案例间的顶层目标描述的相似性做出解释，但不是很有力。在卫星通信问题的例子中，系统是根据通信信号微弱来选择案例的。对这种推理模式来说，除了“在以前类似情况中有效过”这一解释外，不存在更深层的解释。但是正如前面所指出的，对于很多情况来说这一解释已经足够了。

但是，很多学者感到简单重复顶层目标和要被应用的案例所提供的解释是不够的（Leake 1992，Stern and Luger 1992），特别是在需要对为什么某种方法不行做出解释的时候。不妨再以卫星的情况为例。假定调整太阳能电池板朝向有效了，但在三个小时后信号又变弱了。不论是使用频率启发还是最新性启发，我们都会再次调整电池板朝向。在三个小时后信号微弱又发生了。而且再过三个小时后又再次发生，我们总是应用同样的修理方法。这个例子是以在卫星上发生的真实的情况为基础的，实际上，人们发现存在一个更复杂的问题，也就是回转仪过热并给出了失真的读数，使卫星的朝向出现偏差。最终解决这个问题的系统使用了基于模型推理程序来模拟卫星的行为以判断信号微弱的根本原因（Skinner and Luger 1995）。

基于案例推理与类比学习问题有着密切的关系。为了重用过去的知识两种情况下都必须识别突出特征并建立一种映射，这种映射定义了如何把过去的经历用在当前的情况中。*转换类比*（transformational analogy）（Carbonell 1983）是基于案例求解问题的一个例子。它通过不断地修改现存解直到可以把它应用到新实例来求解新问题。修改完整问题解的操作符定义了一种更高层的抽象，或者叫 T－空间，在这个空间中状态就是问题的解，操作符转换这些解，如图 8-17 所示。系统的目标是把源解转换为目标问题的可能解。操作符通过以下方式修改解：在解路径中插入或删除步骤；记录解步骤；把新解嫁接到旧解上；修改当前解中的参数绑定关系。

转换类比是基于案例问题求解方法的一个代表。后来的研究对这种方法进行了改进，考虑了更多问题，比如表示案例的方法、组织内存和以前案例的策略、检索以前有关案例的方法以及在求解新问题中如何使用案例等。如果要了解基于案例推理的更多信息，可以参阅 Hammond（1989）和 Kolodner（1988a，b）。在讨论基于符号的机器学习（见 10.5.4 节）时将进一步讨论类比推理。

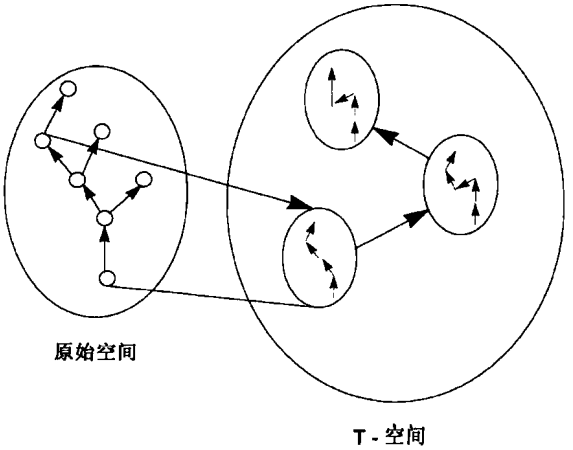


图 8-17　转换类比
［摘自 Carbonell（1983）］

8.3.4　混合设计：强方法系统的优势和不足

专家系统成功地求解了很多复杂的实际问题，这证明了基于知识系统背后的核心思想（推理程序的威力在于它的领域知识，而不是推理方法的复杂性）是正确的。不过，这又回到了人工智能的一个核心问题：知识表示。在实践中，所有的知识工程师必须做出选择：如何以一种最适合给定领域的方式来表示知识。知识表示又会引发出很多理论上非常重要但难度很大的问题，比如处理残缺和不确定的信息、衡量表示的表达力、表示语言之间的关系以及诸如学习、知识获取和推理程序效率这样的问题。

在本章中，我们考虑了很多基本的知识表示方法：基于规则的专家系统、基于模型的推理程序和基于案例的推理程序。为了对知识工程实践有所帮助，我们把这些知识密集型的问题求解方法的优势和不足归纳如下。

基于规则推理

基于规则推理的优点包括：

1）能够以非常直接的方式使用从人类专家那里获取的经验知识。这对于那些高度依赖启发来管理复杂性和/或残缺信息的领域尤其重要。

2）可以把规则映射为状态空间搜索。解释功能支持调试。

3）知识和控制的分离简化了专家系统的开发过程，可以循环地组织开发过程，即不断重复获取、实现和检验规则这一过程。

4）在有限的领域内可以得到很好的性能。因为智能问题求解所需的知识数量极其庞大，所以专家系统的应用仅限于很窄的一些领域。不过，已经证明在很多领域中都已经取得了非常好的效果（只要设计得当）。

5）解释机制很好。尽管基本的基于规则框架支持灵活的针对问题的解释，但是必须注意到这些解释的最终质量依赖于规则的结构和内容。数据驱动系统和目标驱动系统中的解释机制差异很大。

基于规则推理的缺点包括：

1）很多时候从人类专家那里获取的规则具有很强的启发性，没有捕获到领域中的功能性——基于模型的知识。

2）启发性规则往往比较“脆弱”，不能处理有残缺的信息或意外的数据值。

3）规则脆弱性的另一表现就是在领域知识的边缘附近其性能迅速退化。和人类不同，基于规则系统通常不能在遇到新奇问题时回退到基本的推理原则。

4）解释功能仅是描述层的，无法做出理论性解释。这是由以下事实所导致的：启发式规则的强大性主要来源于把问题症状和解答直接联系起来，并不需要更深层的推理。

5）知识往往具有很强的任务依赖性。形式化的领域知识往往就是直接针对当前应用的。目前的知识表示语言还无法接近人类推理的灵活性。

基于案例推理

基于案例推理的优点包括：

1）直接编码历史知识的能力。在很多领域中，可以从存在的案例历史、修理记录或其他资料中获取案例，这缓解了从人类专家那里获取大量知识的压力。

2）使快捷推理成为可能。如果可以找到恰当的案例，那么可以非常快地求解问题，比根据规则或模型产生解要快得多。

3）它使系统可以避免过去的错误但使用过去的成功之处。基于案例推理提供了一个很好的

学习模型，不仅有理论研究价值，而且有足够的实践价值用来求解复杂的问题。

4）不需要对领域知识进行广泛的分析。在基于规则的系统中，知识工程师必须预料到规则的相互作用，与此不同，CBR 可以在知识获取中采用简单的相加模型。这需要恰当的案例表示、有效的索引检索以及合理的案例适应策略。

5）恰当的索引策略增加了系统的洞察力和问题求解能力。区别目标问题的差异并选择恰当案例的能力是基于案例推理程序的关键，很多时候索引算法可以自动产生这种效果。

基于案例推理的缺点包括：

1）很多时候案例不能包含更深层的领域知识。这影响了解释功能，而且在很多情况中可能造成错误应用案例，导致错误的或质量很差的建议。

2）庞大的案例库可能受到存储/计算平衡等问题的困扰。

3）难以确定好的索引和匹配案例标准。目前，单词检索和相似匹配算法必须经过仔细的手工处理；这大大抵消了 CBR 所具有的知识获取优点。

基于模型推理

基于模型推理的优点包括：

1）在问题求解中使用功能性/结构性领域知识的能力。这提高了推理程序处理不同问题的能力，包括那些系统设计者可能未预料到的问题。

2）基于模型推理往往非常鲁棒。和人类经常在遇到新问题时重复基本原则的原因相同，基于模型推理程序往往具有非常好的鲁棒性和灵活性。

3）某些知识可以在任务之间转移。基于模型推理程序经常是使用科学的理论性知识建立的。科学所力求的是广泛适用的理论，这种一般性也为基于模型推理程序提供了优势。

4）很多时候，基于模型推理程序可以提供因果关系解释。这可以向用户表达对故障的更深层理解，因而能起到很重要的辅导作用。

基于模型推理的缺点包括：

1）缺少领域的经验性（描述性）知识。基于规则系统使用的启发方法反映了经验知识是一项有价值的技能。

2）它需要一个显式的领域模型。很多领域（例如电子电路的诊断）有很强的理论基础，可以很好地支持基于模型方法。不过，很多领域（例如某些医疗专业、大多数设计问题或者很多财经应用）缺乏完善定义的科学理论。基于模型方法不能用在这些领域中。

3）复杂度高。基于模型推理通常是在非常复杂的层次运行，这也是人类专家把启发放在第一位的主要原因之一。

4）意外情况。例如，异常的环境、桥接故障或电子部件中多个故障的相互影响可能会以事先难以预测到的方式影响系统功能。

混合设计

目前研究和应用的一个重要热点就是把不同的推理模型组合在一起。在混合体系结构中，两种或更多种模式被集成到一起，以得到一种协作效果：用一种策略的优点来弥补其他策略的不足。通过组合，我们可以弥补前面讨论所提到的各种不足。

例如，基于规则系统和基于案例系统的组合可以：

1）提供一种自然的解决方法：先检查已知案例，然后再进行基于规则推理和有关的搜索。

2）通过规则库中保存的内容提供解的样例和异常。

3）把基于搜索得到的结果记录为案例供将来使用。通过保存合适的案例，推理程序可以避免重复代价很高的搜索。

基于规则系统和基于模型系统的组合可以：

1）用功能性知识加强解释能力。这在教学应用中是特别有价值的。

2）提高了当规则失败时的鲁棒性。如果没有任何启发式规则适用于给定的问题实例，那么推理程序可以根据基本原则进行推理。

3）向基于模型搜索中加入了启发式搜索。这有助于管理基于模型推理的复杂度并使推理程序可以在备择的可能解之间做出智能的选择。

基于模型系统和基于案例系统的组合可以：

1）对案例中记录的情况给出更成熟的解释。

2）在开始基于模型推理和更深入搜索前首先自然地检查存储案例。

3）在案例库中提供样例和异常记录用来引导基于模型推理。

4）记录基于模型推理的结果以备将来使用。

混合方法很值得研究人员和应用开发人员注意。不过，建立这样的系统并不是一件简单的事，需要解决很多问题，比如决定在给定情况下使用哪种推理方法；决定何时改变推理方法；解决推理方法之间的差异；以及设计允许知识共享的表示。

下一节讨论规划，也就是如何把过程片段组织成可能的问题解。

8.4　规划

规划程序的任务是寻找一个动作序列使问题求解器（如控制系统）可以用其完成某个特定的任务。传统规划具有很强的知识密集型特征，因为创建计划需要把知识片段和部分计划组织成一个求解过程。规划在专家系统的时序事件推理中起着重要的作用。规划在生产领域有很多应用，如过程控制。同时，它对设计机器人的控制器也非常重要。在自然语言理解中它也是非常重要的，因为计划、目标和动机是人类经常讨论的话题。

8.4.1　规划简介：机器人学

这一节中，以传统机器人学为例进行讨论（大多数现代机器人学使用的是反应控制而不是协商规划，见 7.3.1 节和 8.4.3 节）。一个传统机器人规划的步骤是由机器人的*原子动作*（atomic action）组成的。对于规划这个目标来说，我们不必关心硬件或微观层次的细节（比如“旋转第 6 个步进电机一圈”）。相反，规划程序在较高层次上定义动作，如动作对周围世界的效果。举例来说，积木世界机器人可能包括这些动作：“拾起物体 a”或“走到位置 x”。实际使机器人执行计划的微观控制步骤是被建立在这些高层动作中的。

因此，“从房间 b 出发去取积木 a”的动作序列可能是：

1）放下目前拿着的所有东西

2）走到房间 b

3）走近积木 a

4）拾起积木 a

5）离开房间 b

6）回到原来的位置

可以通过搜索可能动作的空间直到发现完成任务所必需的序列来创建计划。空间中包含了通过应用动作而产生的世界状态。当目标状态（世界的适当描述）出现时搜索便停止了。因此，启发式搜索中的很多课题（包括寻找 A* 算法）在规划中也存在。

规划行为并不依赖于实际执行该计划的机器人是否存在。在计算机规划的早期阶段（20 世

纪60年代），在机器人执行第一个动作之前就已经制定好了整个规划。因此没有任何机器人在场也可以设计计划。最近，随着视觉感知和反应设备的出现，研究的焦点已经集中到如何设计到动作序列的集成度更高。

传统的规划依赖于搜索技术，同时也提出了很多特有的问题。例如，对世界状态的描述可能比前面的搜索例子更加复杂。考虑要描述一个机器人环境（房间、走廊和其他对象）所需的谓词数量。我们不仅必须要表示机器人的世界，而且还必须表示原子动作对世界的影响。因此对问题空间中每个状态的描述可能相当复杂。

规划中的另一个不同是需要描述某个特定动作没有改变的情况。拾起一个对象改变了：a）这个对象的位置；b）机器人手中现在抓的物体。没有改变：a）门和房间的位置；b）其他对象的位置。说明世界状态中什么是真的并确定“执行某个动作后什么改变了”已经成为一个重要的问题，即所谓的框架问题（McCarthy 1960，McCarthy and Hayes 1969）。随着问题空间复杂度的增大，跟踪每个动作产生的变化并描述保持不变的状态特征变得越来越重要。我们将介绍两种对付框架问题的方法，但是正如读者将要看到的，两种方法都不是令人完全满意的。

其他的重要问题包括：产生计划、保存和泛化好的计划，从意外的计划失败中恢复（世界的某些部分可能是未预测到的，或许是因为意外移动造成的），维护世界与程序中的内部世界模型之间的一致。

在这一节的例子中，把桌面上的积木集合作为机器人世界，并把机器人的动作限制为手臂的动作：可以堆放、移去和在桌上移动积木。在图 8-18 中，我们有 5 个积木（标有 a、b、c、d、e）都被放在桌面上。这些积木都是同样大小的立方体，而且正如图中所示的，堆放积木是使一块积木放在另一块的正上方。机械手具有一个钳子可以抓起任何清洁的积木（即其上没有任何其他积木的积木），并可以把积木移动到桌上的任何位置或放在任何其他清洁的积木上。

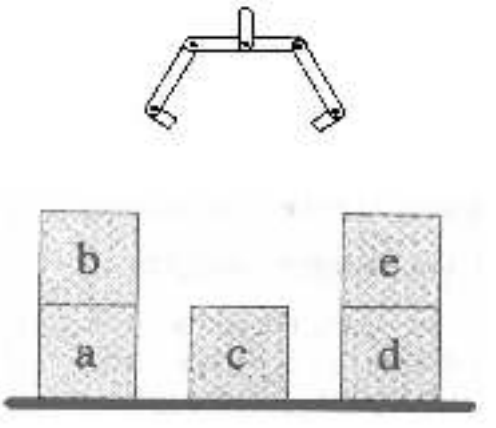


图 8-18　积木世界

机械手可以执行以下几个任务（U、V、W、X、Y 和 Z 是变量）：

goto(X, Y, Z)　移动到由 X、Y 和 Z 坐标描述的位置。这个位置可能包含在 pickup（W）命令中，因为积木 W 有 X、Y、Z 坐标。

pickup(W)　从积木 W 的当前位置把它拾起并抓在手中。假定 W 上是清洁的，机械手是空的，而且计算机知道 W 的位置。

putdown(W)　把 W 放在桌面的某个位置上，并记录下 W 的新位置。W 必须在机械手中。

stack(U, V)　把积木 U 放在积木 V 上。U 必须在机械手中，而且 V 上必须是清洁的。

unstack(U, V)　把积木 U 从 V 上取下。执行这个命令前，U 上必须是清洁的，U 必须在 V 上，机械手必须是空的。

可以用一系列谓词和谓词关系来描述世界状态：

location(W, X, Y, Z)　积木 W 在坐标 X，Y，Z 处。

on(X, Y)　积木 X 在积木 Y 上，而且是紧贴的。

clear(X)　积木 X 上什么也没有。

gripping(X)　机械手中握着积木 X。

gripping()　机械手是空的。

ontable(W)　W 在桌面上。

ontable(W) 是 location(W, X, Y, Z) 在 Z 等于桌面高度时的简洁形式。类似地，on(X, Y) 表示积木 X 的下表面与积木 Y 的上表面是重合的。可以通过让计算机记录每个积木的当前位置 location(X, Y, Z) 来简化以上描述。在这个位置假定下，goto 命令就没有必要了，因为 pickup(X)、stack(X) 命令已经隐含了 X 的位置。

现在就可以用以下的谓词集合来描述图 8-18 中的积木世界了。我们把这个谓词集合称为状态 1。因为这些谓词描述了图 8-18 中的世界状态，它们是同时为真的，所以完整描述就是这些谓词的合取（∧）。

状态1

ontable(a).	on(b, a).	clear(b).
ontable(c).	on(e, d).	clear(c).
ontable(d).	gripping( ).	clear(e).

下面为 clear(X)、ontable(X) 和 gripping() 创建一些真值关系（从声明这个意义上来说），也就是一些供执行的规则（从过程这个意义上来说）：

1. $(\forall X)(\text{clear}(X) \leftarrow \neg(\exists Y)(\text{on}(Y,X)))$
2. $(\forall Y)(\forall X) \neg(\text{on}(Y,X) \leftarrow \text{ontable}(Y))$
3. $(\forall Y)\ \text{gripping}( ) \leftrightarrow \neg(\text{gripping}(Y))$

第一个语句是说如果积木 X 是清洁的，那么不存在任何积木 Y 满足积木 Y 在积木 X 上。如果从过程角度解释，那么就是“要清洁积木 X 就要过去并且移除 X 上面的任何积木 Y”。

现在设计用来对状态进行操作和产生新状态的规则。在这样做的时候，我们再次把过程语义隐含到谓词逻辑表示中。这些操作符（pickup、putdown、stack、unstack）是：

4. $(\forall X)(\text{pickup}(X) \rightarrow (\text{gripping}(X) \leftarrow (\text{gripping}( ) \wedge \text{clear}(X) \wedge \text{ontable}(X))))$.
5. $(\forall X)(\text{putdown}(X) \rightarrow ((\text{gripping}( ) \wedge \text{ontable}(X) \wedge \text{clear}(X)) \leftarrow \text{gripping}(X)))$.
6. $(\forall X)(\forall Y)(\text{stack}(X,Y) \rightarrow ((\text{on}(X,Y) \wedge \text{gripping}( ) \wedge \text{clear}(X)) \leftarrow (\text{clear}(Y) \wedge \text{gripping}(X))))$.
7. $(\forall X)(\forall Y)(\text{unstack}(X,Y) \rightarrow ((\text{clear}(Y) \wedge \text{gripping}(X)) \leftarrow (\text{on}(X,Y) \wedge \text{clear}(X) \wedge \text{gripping}( )))$.

考虑第 4 个规则：对于所有积木 X，pickup(X) 意味着在手中为空和 X 清洁的条件下 gripping(X)。注意这些规则都具有如下形式：A→(B←C)。这是说操作符 A 在条件 C 为真时产生谓词 B。我们可以用这些规则来产生状态空间中的新状态。也就是，如果谓词 C 在一个状态中为真，那么 B 在它的子状态中为真。换句话来说，当谓词 C 为真时，可以用操作符 A 产生一个新的用谓词 B 描述的状态。创建这些操作符的其他方法包括 STRIPS（Nilsson 1980 和 8.4.2 节）以及 Rich and Knight（1991）中的 do 函数。

不过，我们必须在使用这些规则关系产生新的积木世界状态前解决*框架问题*。*框架公理*（frame axiom）是一种规则，它可以告诉我们哪些描述状态的谓词不因规则应用而改变，因此它携带了有助于描述新世界状态的不变信息。举例来说，如果我们对图 8-18 中的积木 b 应用 pickup 操作符，那么和其余积木相关的所有谓词在子状态中仍然为真。对于这个积木世界来说，我们可以确定几个这样的框架规则：

8. (∀ X) (∀ Y) (∀ Z) (unstack(Y,Z) → (ontable(X) ← ontable(X))).

9. (∀ X) (∀ Y) (∀ Z) (stack(Y,Z) → (ontable(X) ← ontable(X))).

这两个规则指出 ontable 不受 satck 和 unstack 操作符的影响。即使当 X 和 Z 相同时这也是成立的；如果 Y = Z，那么上面的规则 6 和 7 中有一个不为真。

其他框架公理指出仅当特定的 on 关系被 unstack 或 clear 关系被 stack 时，on 和 clear 才受 stack 和 unstack 操作符的影响。因此，在我们的例子中，on(b, a) 不受 unstack(c, d) 的影响。

类似地，还可以用框架公理指出仅当 X = Y 或 gripping() 为真时，clear(X) 关系不受 gripping(Y) 的影响。另外，gripping 不影响 on(X, Y) 关系，但是当 X 是被抓对象时 gripping 会影响 ontable(X) 关系。因此，我们可以为自己的例子确定很多框架公理。

总而言之，这些操作符和框架公理定义了一个状态空间，下面以 unstack 操作符为例做进一步说明。unstack(X, Y) 需要三个条件同时为真，也就是 on(X, Y)、gripping() 和 clear(X)。当这些条件满足时，便可以通过应用 unstack 操作符产生新的谓词 gripping(X) 和 clear(Y)。状态 1 中为真的很多其他谓词在状态 2 中仍保持为真。这些状态是通过框架公理保持到状态 2 的。现在通过向状态 1 的 9 个谓词应用 unstack 操作符以及框架公理来产生描述状态 2 的 9 个谓词，这个过程的净结果是移去积木 e：

状态2

ontable(a).	on(b,a).	clear(b).
ontable(c).	clear(c).	clear(d).
ontable(d).	gripping(e).	clear(e).

归纳起来：

1）可以把规划看成是状态空间搜索。

2）通过应用像 stack 和 unstack 这样的通用操作符以及框架规则来产生新的状态。

3）可以应用图搜索技术来寻找从起始状态到目标状态的路径。这种路径上的操作便组成了计划。

图 8-19 显示了应用上面描述的操作符来搜索状态空间的例子。如果把目标描述加入到这个问题求解过程之中，那么就可以把计划看成是一个操作符集合，这些操作符产生了从图中的当前状态通往目标的路径（见 3.1.2 节）。

以上规划问题的特征性定义了规划在状态空间搜索、谓词演算表示和推理方面的理论基础。不过，应该注意到这种解决方式可能是非常复杂的。特别是，使用框架规则来推导状态之间保持不变的谓词可能使搜索复杂度呈指数变化，这一点可以从非常简单的积木世界问题中看到。事实上，当引入任何新的谓词描述符时（比如为了表示颜色、形状和大小），必须定义新的框架规则。

以上讨论还假定组成任务的子问题是独立的，因此可以按任意顺序求解。在实际的复杂问题域中这种情况是非常少见的，相反，实现一个子目标所需的前提和动作经常是与实现另一个目标的前提和动作相冲突的。下一节将介绍这些问题并讨论一种非常有助于处理这种复杂度的规划方法。

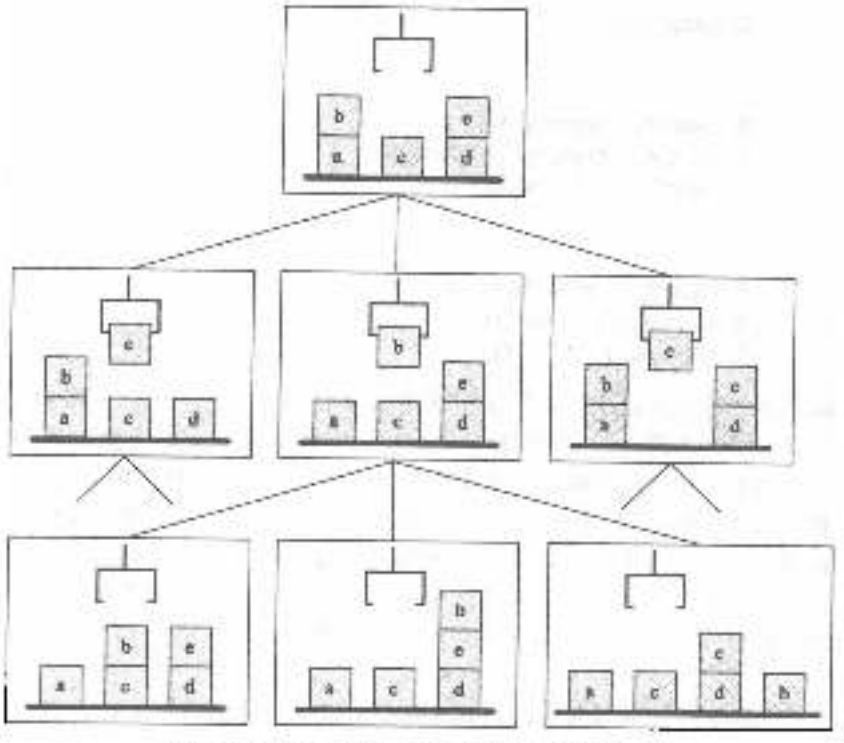


图 8-19 积木世界的一个部分的状态空间部分

8.4.2 使用规划宏：STRIPS

STRIPS 是由现在的斯坦福研究所（SRI International）开发的，意思是斯坦福研究所规划系统（Stanford Research Institute Planning System）（Fikes and Nilsson 1971，Fikes et al. 1972）。这个控制程序是用来驱动 20 世纪 70 年代的 SHAKEY 机器人的。STRIPS 解决有效地表示和实现规划程序的操作时出现的问题。它考虑了子目标冲突问题并提供了一个早期的学习模型：成功的计划被保存成宏操作符（macro operator），宏操作符可以泛化用在将来的类似情况中。在本节的余下部分中，介绍一种 STRIPS 风格的规划和三角表格——这是一种用来组织和存储宏操作的数据结构。

仍以前面的积木世界为例，但这里用一个三元组来表示 pickup、putdown、stack 和 unstack 操作符。三元组的第一个元素是前提集合（P），也就是要应用一个操作符时必须满足的条件。三元组的第二个元素是增加列表（A），也就是应用操作符的结果导致的状态描述增加。最后一个元素是删除列表（D），也就是当应用操作符时要从状态描述中删除的项目。这些列表消除了对独立框架公理的需要。按照这种方式可以把四个操作符表示为：

pickup(X)
P: gripping( ) ∧ clear(X) ∧ ontable(X)
A: gripping(X)
D: ontable(X) ∧ gripping( )

putdown(X)
P: gripping(X)
A: ontable(X) ∧ gripping( ) ∧ clear(X)

D: gripping(X)

stack(X,Y)
P: clear(Y) ∧ gripping(X)
A: on(X,Y) ∧ gripping( ) ∧ clear(X)
D: clear(Y) ∧ gripping(X)

unstack(X,Y)
P: clear(X) ∧ gripping( ) ∧ on(X,Y)
A: gripping(X) ∧ clear(Y)
D: gripping( ) ∧ on(X,Y)

增加和删除列表的重要之处是它们指定了满足框架公理所需的一切。在增加和删除列表方法中存在某些冗余。举例来说，在 unstack 中增加 gripping(X)隐含了删除 gripping()。但这种冗余的好处是没有被增加或者删除列表提到的所有状态描述都在新的状态描述中将保持不变。

这种“前提－增加－删除列表”方法的一个弱点是我们不再使用定理－证明过程来（演绎推理）产生新的状态。不过这不是一个严重的问题，对这两种方法等价性的证明可以保证“前提－增加－删除列表”方法的正确性，可以使用“前提－增加－删除列表”方法产生与前面例子使用推理规则和框架公理产生的结果相同的结果。两种方法的状态空间搜索是相同的，如图 8-19 所示。

截至目前介绍的两种方法还没有解决规划中存在的很多其他固有问题。在求解一个目标时，我们经常将它分割成多个子问题，例如，unstack(c, d)和 unstack(b, a)。如果实现一个目标所必需的动作实际上撤销了另一个动作，那么就无法独立地求解这些子目标。不兼容的子目标可能导致子目标线性（独立）假定不成立。计划/动作空间的非线性可能使搜索异常困难或者甚至不可能。

下面举一个非常简单的例子来说明不兼容子目标问题，我们使用图 8-18 的状态 1 作为起始状态。假定规划的目标是图 8-20 所示的状态 C，即 on(b, a) ∧ on(a, c)，积木 d 和 e 仍然保持状态 1 中的位置。可以注意到合取目标 on(b, a) ∧ on(a, c)中第一个合取项（也就是 on(b, a)）在状态 1 中是成立的。但是为了实现第二个子目标 on(a, c)，必须先破坏这部分已经满足的目标。

三角表格表示（Fikes and Nilsson 1971，Nilsson 1980）是一种旨在缓解以上问题的数据结构。其目的是用来组织一个计划的动作序列，包括可能不兼容的子目标。三角表格通过表示动作序列的全局性相互作用来处理子目标冲突问题。它把一个动作的前件与其前面动作的后件——合并的增加和删除表——联系起来。

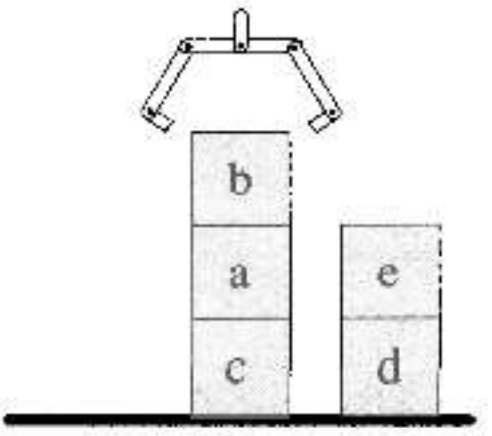


图 8-20　积木世界的目标状态

在建立计划过程中，可以使用三角表格来判断何时可以使用一个宏操作符。通过保存并复用这些宏操作符，STRIPS 大大提高了它的规划搜索效率。事实上，我们可以对宏操作符进行泛化，方法是用变量名来代替特定例子中的积木名。然后就可以调用这些泛化后的新宏来修剪搜索。在第 10 章中介绍基于符号学习时，将讨论泛化宏运算的技术。

宏操作符的复用还有助于解决子目标冲突问题。正如下面的例子所说明的，一旦规划程序已经制定了一个实现 stack(X, Y) ∧ stack(Y, Z) 目标的计划，那么就可以保存并复用这个计划了。这便避免了把这个目标分解成子目标以及随之带来的麻烦。

图 8-21 显示了一个表示宏动作 stack(X, Y) ∧ stack(Y, Z) 的三角表格。可以把这个宏动作应用到 on(X, Y) ∧ clear(X) ∧ clear(Z) 为真的状态。如果令 X = b，Y = a，Z = c，那么就可以把这个三角表格应用到状态 1。

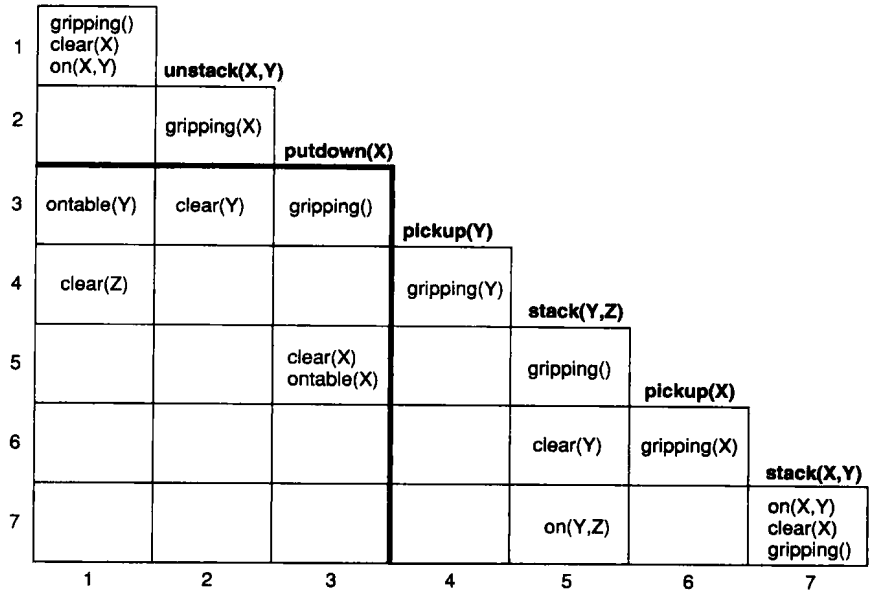


图 8-21　三角表格

[摘自 Nilsson (1971)]

计划的原子动作是沿对角线记录的。例如，前面讨论的四个动作 pickup、putdown、stack 和 unstack。这些动作的前件集合位于这个动作之前的行中，后件位于其下的列中。举例来说，第 5 行列出了动作 pickup(X) 的前件，第 6 列列出了 pickup(X) 的后件（增加和删除列表）。后件被放在使用它们作为前件的动作行中，而且是按照和进一步动作有关的方式来组织的。三角表格的目标是使构成较大目标的较小动作的前件和后件正确交错。因此，三角表格是在宏操作符这个层次上来考虑规划中的非线性问题的，偏序规划程序（Russell and Norvig 1995）和其他一些方法进一步考虑了这些问题。

三角表格的一个优点是有助于从未预料到的情况（比如积木略微错位）或事故（比如落下一块积木）中恢复。很多时候如果发生了事故，那么需要后退很多步才能恢复计划。当某些东西出问题时，规划程序可以回溯到三角表格的行和列中检查仍然为真的情况。一旦规划程序已经分析出行和列中仍然为真的情况，那么它便知道了要重新开始较大的求解目标应该从哪里开始。这种方法是从核（kernel）概念衍生而来的。

n 次核就是第 n 行和其下的所有行与第 n 列及其左边的所有列组成的交叉区域。图 8-21 中用粗线勾画出了三次核。在执行一个用三角表格表示的计划时，第 i 个操作（也就是第 i 行中的操作）仅当 i 次核所包含的所有谓词都为真时才可以执行。这便提供了一种非常简单的方法来核对是否可以执行一个步骤；同时也允许我们系统地恢复计划的任何中断。给定一个三角表格，我们只要寻找并执行满足核的最大序号动作便可以了。这不仅使我们可以备份计划，而且还有可能

使未预料到事件促成计划的向前跳跃。

最左列中的条件是整个宏动作的前件。最下面一行中的条件是这个宏动作加入世界中的条件。因此，可以用三角表格自己的前件/后件集合和添加/删除列表把它保存为一个宏操作符。

当然，三角表格方法确实丢失了前面所介绍的规划模型的部分语义。例如，在一个动作的所有后件中，仅有同时也是后面动作前件的那些部分被保存下来。因此，如果希望保证正确性，那么就需要对三角表格进行进一步的验证，或许可以通过附加信息编写一系列三角表格来解决这个问题。

在规划中使用宏操作符也产生了另一些问题。随着宏操作符数量的上升，规划程序可用的操作也越来越强大，这减小了必须搜索的状态空间大小。但不幸的是，在搜索的每一步，都必须分析所有这些操作符。判断应用哪个操作符所需的模式匹配过程会给搜索增加可观的开销。判断何时该保存一个宏操作以及下一步应该使用哪个宏操作符仍然是有待研究的问题。在下一节介绍一种可以同时满足多个子目标的算法——teleo-reactive 规划。

8.4.3　teleo-reactive 规划

从前一节所介绍的对规划的早期研究（Fikes and Nilsson 1971）到现在，这一领域已经有了很多重要的发展。早期研究大多数是独立于具体领域规划的，但是最近人们已经意识到了针对具体领域（采用一种分布式的感知/反应机制）的重要性。下面，介绍一个独立于具体领域的规划系统——teleo-reactive 规划，以及一个针对领域的规划程序——来自 NASA 的 Burton。如果要了解最早期规划研究的详细情况，推荐读者阅读（Allen et al. 1990）。

teleo-reactive（TR）规划最初是由 Nilsson（1994）和 Benson（1995）在斯坦福大学提出的。TR 规划提供了一种通用的体系结构，可以应用到必须协调控制各个复杂子系统以实现更高层目标的很多领域。因此它融合了层次控制体系结构的从顶至下方法和“基于主体的”（见 7.4 节）从底至上方法。产生的系统可以通过针对任务的简单主体的合作来实现复杂的问题求解。这样做的理由是简单主体具有工作在较小的具有更多约束的问题空间内的优势。另一方面，较高层的控制程序可以做出关于整个系统的更加全局性的决策，例如当前的纯粹局部决策可能如何影响整个问题求解任务的结果。

teleo-reactive 控制组合了基于反馈控制和离散动作规划的特征。teleo-reactive 程序序列化地执行已经汇编进一个面向目标计划的动作。与更传统的 AI 规划环境（Weld 1994）不同，它不对动作的离散性和不可中断性以及每个动作效果的完全可预测性做出任何假定。相反，teleo 动作可以维持很长一段时期，也就是说只要 teleo 动作的前件是被满足的而且与其关联的目标还没有实现，那么就执行这个动作。Nilsson（1994）称这种类型的动作为可持续的。可持续动作可以在某个其他的更靠近顶层目标的动作被激活时打断。一个很短的感知 - 反应循环保证了当环境变化时控制动作也会迅速改变以反映问题解的最新状态。

teleo-reactive 动作序列使用一种被称为 TR 树的数据结构来表示，如图 8-22 所示。TR 树是用一系列条件 - 动作对来描述的（也就是产生式规则，见 6.3 节）。例如：

$C_0 \rightarrow A_0$
$C_1 \rightarrow A_1$
$C_2 \rightarrow A_2$
...
$C_n \rightarrow A_n$

其中 C_i 是条件，A_i 是与之关联的动作。我们把 C_0 称为树的最顶层目标，把 A_0 称为空动作，意思是一旦最顶层目标已经实现了，那么就不要再做任何事了。在 teleo-reactive 系统的每次循环中，从规则的最顶层向下评估每个 C_i（C_0，C_1，…，C_n），直到找到第一个成立的条件。然后便执行与这个成立条件相关联的动作。然后重复这个评估循环，重复频率与电路控制的反应频率接近。

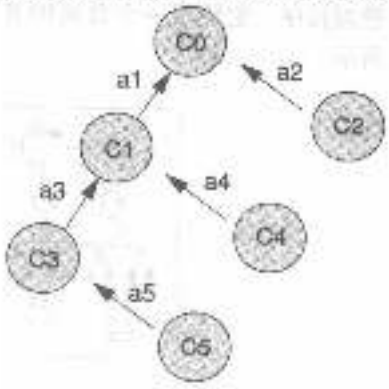


图 8-22　一个简单的 TR 树

注：它显示了支持最顶层目标为条件动作规则，源自 Klein et al.（2000）

$C_i \rightarrow A_i$ 产生式是以这样一种方式组织的：对于每个动作 A_i 来说，如果在正常情况下不断执行这个动作，那么它最终可以使 TR 规则树（见图 8-22）中某个更高的条件为真。可以把 TR 树的执行看成是可适应的，因为如果在控制环境中某个未预料到的事件逆转了前面动作的效果，那么 TR 执行过程会返回到较低层反映那个条件的规则条件。从这一点来看，它会根据满足所有更高层目标的需要重新启动。类似地，如果意外发生了某个好的现象，那么 TR 执行过程会自动切换到对应于这个成立条件的动作，从这个意义上来说，它具有机会性。

可以使用一种 AI 中很普遍的目标缩减方法来构建 TR 树。规划程序从最顶层目标开始搜索其结果包含要实现目标的动作。搜索到的动作的前件产生了新的子目标集合，因此这个过程是递归的。当环境中的当前状态满足叶结点的前件时搜索便终止。因此这种规划算法是从最顶层目标通过目标缩减方法回到当前状态。当然，一些动作经常会有副作用，因此规划程序必须仔细地验证任何层的动作不会改变那些是 TR 树上更高层动作前件的条件。因此，目标缩减是在有约束的情况下进行的，在这个过程中可以采用各种不同的动作排序策略来消除可能的约束冲突。

因此，TR 规划算法最用来建立问题域中当前状态可以满足叶结点的计划。它通常不建立完整的计划——即可以从任何世界状态启动的计划，因为这样的计划通常过于庞大，难以存储或被高效地执行。这一点是很重要的，因为有时一个未预料到的环境事件可能把世界切换到一种 TR 树中的所有动作前件都不满足的状态，这通常是通过重新激活 TR 规划程序来完成的。

Benson（1995）和 Nilsson（1994）已经把 teleo－reactive 规划应用到很多领域，包括控制分布式机器人主体和建立飞行模拟器。Klein et al.（1999，2000）已经使用 teleo－reactive 规划程序来建立和检验一种可移植的控制体系结构来加速粒子束。这些研究者指出在粒子束控制领域使用 teleo－reactive 控制程序有很多好处：

1）加速器粒子束及其相关的诊断通常是动态的而且是有噪声的。

2）加速器调节目标的实现经常是通过随机过程、RF 衰弱或粒子源的摆动来实现的。

3）调节所需的很多动作是可持续的，尤其是"需要继续调整和优化操作直到满足特定标准的时候"。

4）TR 树为加速器设计者提供了一种直观的框架用来编排调节计划。事实上在一点点帮助下，设计者们便能够开发出自己的 TR 树。

这些应用的进一步细节可以在参考文献中找到。下面再回到 8.3 节中 NASA 的基于模型推理例子，以进一步描述航天飞船推进系统的控制/规划算法。

8.4.4　规划：来自 NASA 的例子

本节描述如何在基于模型推理程序的背景下实现规划程序。我们继续讨论 8.3.2 节中引入的

来自 NASA 的例子，这个例子摘自 Williams and Nayak（1996b）。Livingstone 是一个反应式的配置管理程序，它使用一个合成的基于组件模型来决定太空飞船推进系统的配置动作，如图 8-23 所示。

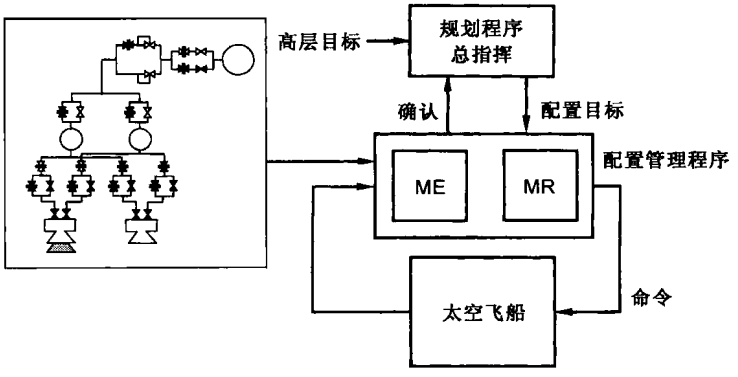


图 8-23　基于模型的反应式配置管理

［摘自 Williams and Nayak（1996b）］

每个推进部件被模型化为一个过渡系统，这个过渡系统指定了该部件工作和故障状态的行为、状态之间的正常和故障过渡以及过渡的成本和可能性，如图 8-24 所示。在图 8-24 中，open 和 closed 是正常运转状态，stuck open 和 stuck closed 是故障状态。open 命令具有单位成本，会导致状态从 closed 过渡到 open，close 命令与此类似。故障过渡把阀门从正常运转状态过渡到一种故障状态，对应的概率为 0.01。

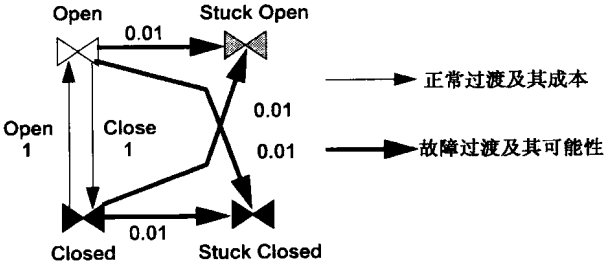


图 8-24　一个阀门过渡系统模型

［摘自 Williams and Nayak（1996a）］

状态行为是使用命题逻辑中的公式来指定的，但是状态之间的过渡是利用限定时态命题逻辑中的公式来指定的。限定时态命题逻辑足以对以下情况建模：数字硬件、使用定量抽象的模拟硬件（de Kleer and Williams 1991，Weld and de Kleer 1990）以及使用实时反应系统模型的实时软件（Manna and Pnueli 1992）。太空飞船过渡系统模型是其部件过渡系统的组合，太空飞船的配置集合是部件状态集合的交积。我们假定部件过渡系统是同步运行的，即对于太空飞船的每次过渡，每个部件也过渡一次。

基于模型的配置管理程序使用以上过渡系统模型来完成两个重要的行为：识别太空飞船的当前配置（称为*状态估计* ME）；把太空飞船转移到一种实现配置目标的新配置（称为*状态重置* MR）。ME 不断产生相对于前一配置的所有太空飞船过渡，目的是使得到的配置与当前的观察结果一致。在图 8-25 所显示的情况中，左侧的引擎在前一状态中正常引发了，但是在当前状态中并没有观察到任何推力。ME 的任务是识别太空飞船已经过渡到哪种配置。图中显示了两种可能的过渡，对应于主引擎之一的阀门故障。出故障的阀门（stuck closed）被圈了出来。很多其他

过渡（包括不太可能的双重故障）也可以导致这种观察结果。

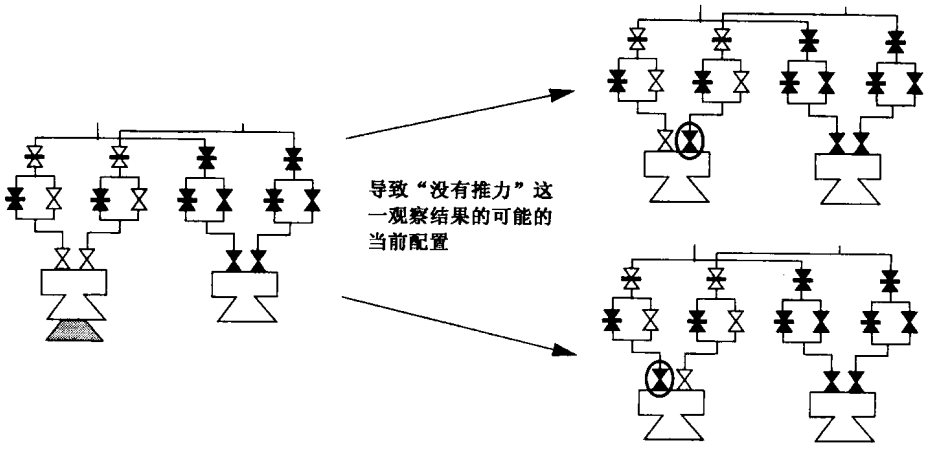


图 8-25　状态估计（ME）
［摘自 Williams and Nayak（1996a）］

MR 的任务是决定要发送给太空飞船的命令，从而使产生的过渡可以把太空飞船置于一种实现下一状态（如图 8-26 所示）配置目标的配置之中。图 8-26 显示了一种方式识别器已经标识出连向最左引擎的故障阀门的情况。MR 推断如果打开通往右侧主引擎的适当阀门，那么在下一状态中推力将恢复正常。图中显示了可以实现这一目标的两种配置，圈出的阀门是推荐要改变状态的阀门。过渡到上边的配置需要的成本较低，因为仅需要引发必要的热阀（见 8.3.2 节）。通往左侧引擎的阀门被关闭以满足同一时间至多引发一个引擎的约束。在 ME 和 MR 中使用太空飞船模型保证了配置目标的正确实现。

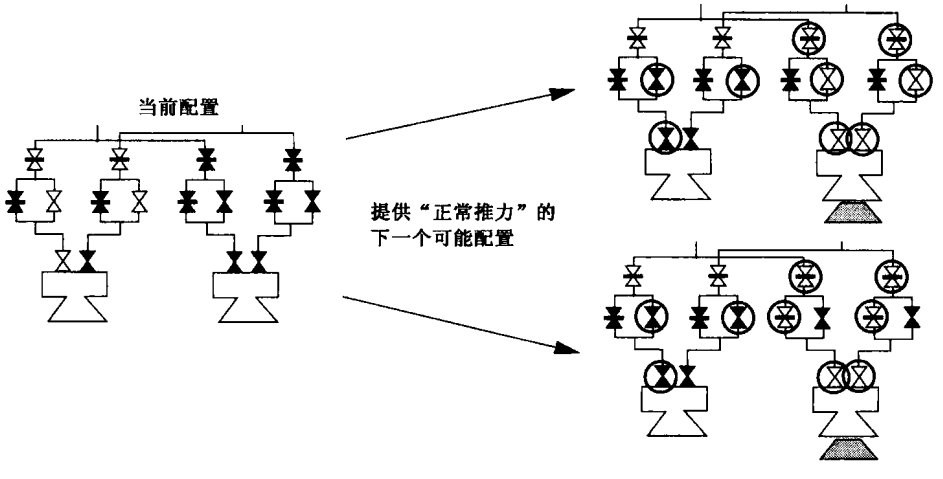


图 8-26　状态重置（MR）
［摘自 Williams and Nayak（1996a）］

不论 ME 还是 MR 它们都是反应式的（见 6.4 节）。ME 根据前一配置的知识和当前观察到的信息推断当前配置。MR 仅考虑实现下一状态配置目标的命令。基于此，决定使用同步模型来对组件过渡过程建模是个关键。另一种可选的方法是通过交叉来对多个组件过渡建模。不过，交叉可能使当前配置与目标配置之间的距离任意化，无法满足把推断限制在很少的固定数量状态之

内的要求。如果底层软硬件中的部件过渡不是同步的，那么 Livingstone 的建模假定是，所有的过渡交叉是正确的而且可以实现预期的配置。这个假定在 Burton 中被删除了，Burton 是 Livingstone 的继续，它的规划程序决定一系列动作来产生所有预期的过渡（Williams and Nayak 1997）。NASA 在一项名为“Tech Sat 21”的任务中使用了 Burton 体系结构。

对于 Livingstone 来说，ME 和 MR 不需要分别产生所有过渡和控制命令。相反，它们只要产生最可能的过渡和最佳的控制命令。这是通过把 ME 和 MR 重组为组合优化问题而实现的。即 ME 总是用最可能的过渡来延伸导致当前配置的各个轨迹，然后 MR 识别出具有最低预期成本的命令，该命令把太空飞船从可能的当前配置过渡到实现预期目标的配置。我们可以使用一种冲突导向的最佳优先搜索算法来高效地求解这一组合优化问题。对 ME 和 MR 的更形式化的描述以及对搜索和规划算法的详细介绍，请参见 Williams and Nayak（1996a，1996b，1997）。

8.5　结语和参考文献

基于规则专家系统所使用的体系结架是产生式系统。不论最终产生式是数据驱动的还是目标驱动的，这种软件的模型都是产生式系统产生的图搜索（见第 6 章）。本书补充材料中分别用 Java、Prolog 和 LISP 实现了以产生式系统为基础的专家系统外壳。这些系统的规则可以包含某种形式的确定性尺度，以用于设计基于启发的搜索。

关于本章内容的资料非常多，特别推荐介绍 MYCIN 的最初出版物，即斯坦福大学 Buchanan 和 Shortliffe（1984）所著的《Rule-Based Expert Systems》。早期其他的一些专家系统，在 Waterman（1968）、Michie（1979）、Patil 等（1981）、Clancy（1983）以及 Clancy 和 Shortliffe（1984a，1984b）的文章中有介绍。对于数据驱动的基于规则搜索的鲁棒性实现，推荐使用由 NASA 发布的 CLIPS 软件。关于 CLIPS 的出色的参考书是 Giarratano and Riley（1989，2005）。

关于一般知识工程的其他重要书籍包括 Hayes-Roth et al.（1984）所著的《Building Expert Systems》、Waterman（1986）所著的《A Guide to Expert Systems》、Alty and Coombs（1984）所著的《Expert Systems：Concepts and Examples》、Johnson 和 Keravnou（1985）所著的《Expert Systems Technology：A Guide》、Negoita（1985）所著的《Expert Systems and Fuzzy Systems》、Jackson（1986）所著的《Introduction to Expert Systems》以及 Harmon et al.（1988）所著的《Expert Systems：Tools and Applications》。也可以参考 Ignizio（1991）所著的《Introduction to Expert Systems》、Mockler 和 Dologite（1992）所著的《An Introduction to Expert Systems》、John Durkin（1994）所著的《Expert Systems：Design and Development》以及人工智能应用与专家系统会议集 IEA/AIE（2004 ~ 2007）。

因为专家系统求解方法具有较强的领域针对性，所以案例研究是了解这一领域知识的一个重要方法。这方面的书籍包括 Klahr 和 Waterman（1986）所著的《Expert Systems：Techniques，Tools 和 Applications》、Keravnou 和 Johnson（1986）所著的《Competent Expert Systems：A Case Study in Fault Diagnosis》、Smart 和 Langeland - Knudsen（1986）所著的《The CRI Directory of Expert Systems》、Coombs（1984）所著的《Developments in Expert Systems》以及 Prerau（1990）所著的《Developing and Managing Expert Systems》。特别推荐 Durkin（1994）所著的《Expert Systems：Design and Development》，因为这本书给出了大量有关建立专家系统的实践性建议。

目前已经开发出了很多知识获取技术。如果要了解具体方法的详细信息，请参阅 McGraw 和 Harbison-Briggs（1989）所著的《Knowledge Acquisition：Principles and Guidelines》、Chorafas（1990）所著的《Knowledge Engineering》、Mockler 和 Dologite（1992）所著的《An Introduction to Expert Systems》，以及关于专家系统的其他书籍和论文集。

基于案例推理是从 Schank 小组在耶鲁大学所做的早期研究以及他们对脚本的研究中衍生而来的（见 7.1.3 节和 7.1.4 节）。Kolodner（1993）所著的《Case - Based Reasoning》是对这一领域的极好的介绍。Kolodner 和她的研究小组在基于案例推理领域的其他有关工作包括（Kolodner 1987，1991）。Leake（1992，1996）对基于案例系统中的解释提出了一些重要的意见。现在已经有很多商业软件产品支持基于案例推理技术。

基于模型推理是从医药、逻辑电路和其他数学领域的显式表示发展起来的，一般用于教学目的（deKleer 1975、Weiss et al. 1977、Brown and Burton 1978、Brown and VanLehn 1980、Genesereth 1982、Brown et al. 1982）。Hamscher 等人（1992）所编辑的《Readings in Model-based Diagnosis》、Davis et al.（1982）所著的《Diagnosis Based on Description of Structure and Function》和关于多故障的上下文诊断（deKleer and Williams 1987，1989）介绍了该领域的一些最新研究。Skinner and Luger（1995）以及其他一些主体体系结构方面的作者（Russell and Norvig 1995，2003）介绍了混合专家系统，混合方法通过多种途径的相互补充来求解问题，以产生一种综合效果，发挥一些系统的强处，弥补其他系统的不足。

8.4 节中介绍了可以用来构建通用规划程序的一些数据结构和搜索技术。进一步的资料包括 ABSTRIPS（即 STRIPS 产生器关系的摘要说明）（Sacerdotti 1974）、Warren 的工作（1976）以及用于非线性或者层次规划的 NOAH（Sacerdotti 1975，1977）。关于 teleo - reactive 规划的更多信息，可参阅 15.3 节以及 Benson 和 Nilsson（1995）。

元规划是一种不仅对计划而且对规划过程进行推理的技术。这在专家系统方法中可能很重要。参考读物包括关于 DENDRAL 的 Meta - DENDRAL（Lindsay et al. 1980）以及关于 MYCIN 的 Teiresias（Davis 1982）。如果要建立与世界不断交互的计划，也就是对变化的环境建模，请参见 McDermott（1978）。

进一步的研究包括使用黑板的机会规划以及建立在面向对象规范之上的规划（Smoliar 1985）。在《Handbook of Artificial Intelligence》（Barr and Feigenbaum 1989，Cohen and Feigenbaum 1982）和《Encyclopedia of Artificial Intelligence》（Shapiro 1992）中有一些关于规划研究的调查。Russell 和 Norvig（1995）中介绍了非线性规划问题和偏序规划问题。《Readings in Planning》（Allen et al. 1990）与本章内容的关系也很密切。

我们介绍的来自 NASA 的基于模型推理和规划算法摘自 Williams 和 Nayak（1996a，1996b，1997）。感谢 Williams 和 Nayak 以及 AAAI 出版社允许我们引用这些材料。

8.6　习题

1. 在 8.2 节中介绍了一系列用来诊断汽车故障的规则。为这一应用确定可能的知识工程师、领域专家以及潜在的最终用户。分别描述一下对这几类人的期望和要求。
2. 以练习 1 为例。用汉语或伪代码创建 15 条“如果……那么”规则（不包括 8.2 节中已介绍的那些规则）来描述这一领域内的关系。创建一幅图来表示这 15 个规则之间的关系。
3. 考虑练习 2 中的图。你推荐使用数据驱动搜索还是目标驱动搜索？宽度优先搜索还是深度优先搜索？怎样用启发来辅助搜索？请给出你的答案并说明理由。
4. 挑选另一个适合使用专家系统的领域。针对这一应用回答练习 1 ~ 3。
5. 利用商业外壳程序实现一个专家系统。这样的程序很多，有用于个人计算机的，也有用于大型机的。特别推荐 NASA 的 CLIPS（Giarratano and Riley 1989）或者 Sandia 国家实验室的 JESS。
6. 对你完成练习 5 所用的外壳加以评论。它有什么优点和缺点？你想如何改进它？它适合你的问题吗？这个工具最适合什么样的问题？
7. 为一个简单的电子设备创建一个基于模型的推理系统。合并几个小的设备以组成一个较大的系统。你可

以使用“如果……那么”规则来描述这个系统的功能。

8. 阅读并评价论文《Diagnosis based on description of structure and function》（Davis et al. 1982）。
9. 阅读一篇关于利用基于模型推理教授儿童代数（Brown and Burton 1978）或者电子技术（Brown and VanLehn 1980）的早期论文。对这种方法加以评论。
10. 为你选择的一种应用建立一个基于案例推理程序。可以选择计算机科学与技术课程的本科生或硕士生需要完成的项目。
11. 使用商业软件包（从万维网上查找）建立练习 10 的基于案例推理系统。如果找不到这样的软件包，那么可以考虑使用 Prolog、LISP 或 Java 来建立这个系统。
12. 阅读并评议《Improving human decision making through case-based decision aiding》（Kolodner 1991）。
13. 为 8.4 节中规则 4 到 7 所描述的四个操作符 pickup、putdown、stack 和 unstack 建立必要的框架公理。
14. 使用上一练习中的操作符和框架公理产生图 8-19 所示的搜索空间。
15. 8.4 节使用框架公理从状态 1 产生了状态 2，说明如何使用*增加*和*删除*列表代替框架公理来完成这一过程。
16. 使用*增加*和*删除*列表生成图 8-19 所示的搜索空间。
17. 设计一个自动推理控制程序，它可以使用*增加*和*删除*列表生成类似于图 8-19 所示的图搜索。
18. 考虑图 8-19 所示的积木世界，再举出两个不兼容的（前件）子目标。
19. 阅读有关 ABSTRIPS 的研究资料（Sacerdotti 1974），说明它如何处理规划中的线性问题（也就是不兼容子目标问题）。
20. 在 8.4.3 节中介绍了一种由 Nilsson 和他的学生在斯坦福大学建立的规划程序（Benson and Nilsson 1995）。teleo-reactive 规划允许使用*可持续的*动作。为什么相对类似 STRIPS 的规划程序来说，teleo-reactive 规划更受青睐？用 Prolog、LISP 或 Java 建立一个 teleo-reactive 规划程序。
21. 阅读 Williams and Nayak（1996，1996a）关于他们的基于模型规划系统的更全面的讨论。
22. 对 Williams and Nayak（1996a，973 页）中介绍的用来描述太空飞船推进系统状态过渡的命题演算表示模式进行扩充。

第9章　不确定条件下的推理

所有传统逻辑习惯上总是假定当前使用的是精确的符号。正因为如此，传统逻辑难以应用于现实生活，而只是存在于虚幻的想象之中。

——伯特兰·罗素

满足于事物自身允许的精确度，在只要近似解就能满足的情况下而不去寻求精确解，这是很有指导意义的。

——亚里士多德

适用于现实世界的数学定律都不具有确定性，具有确定性的数学定律则不适用于现实世界。

——阿尔伯特·爱因斯坦

9.0　简介

在第一部分、第二部分、第三部分，我们的推理过程遵循着谓词演算所用的推理模型：可靠的推理规则从正确的前提推出新的正确的结论。但是，像我们在第5章和第8章所看到的，许多情况下这种方法并不适合，也就是说，我们必须用不可靠的推理规则从形式不规则和不确定的证据中得出有用的结论。

用不可靠的推理从不完备和不精确的数据中得出有用的结论不是不可能的，我们在现实生活中的几乎所有方面都做到了这一点。我们从不明确的症状得出正确的医疗诊断并给出相应的治疗方案，我们分析自己的汽车和立体声系统的问题，我们能理解晦涩或者成分不全的语句，我们能通过朋友们的声音或者姿势来辨认他们，等等。

为了演示不确定条件下推理的问题，我们来看一下8.2节中的汽车专家系统的规则2：

如果
发动机不旋转而且灯不亮，
那么
电池或电缆有问题。

表面上，这条规则看起来像一个用在可靠推理（取式假言推理）中的普通的谓词关系。但是，实际上它不是，它本质上是启发式的。一种可能情况是，电池和电缆是好的，汽车只是发动机坏了，前灯烧了，虽然这并不太可能。发动机不能工作和灯不亮不一定意味着电池和电缆是坏的。

有趣的是这条规则的逆为真：

如果
电池或电缆有问题，
那么
发动机不旋转而且灯不亮。

除非有超自然的力量，否则如果电池坏了，灯和发动机都不能工作！

我们的专家系统提供了一个反绎推理的例子。形式上，反绎推理认为从 P→Q 和 Q 可能推出 P。反绎推理是不可靠的推理，意思是说在前提为真的任一解释下，结论不一定为真（见

2.3 节）。

虽然反绎是不可靠的，它对于解决问题来说却是很重要的。前面的“逻辑上正确”的规则对诊断汽车故障不是很有用是因为它的前提，电池坏了是我们的目标，它的结论是我们看到的症状，我们必须用这些结论来进行诊断。但是，这条规则可以以反绎的形式使用，就像许多诊断型专家系统中的规则一样。故障或疾病引起（蕴涵）症状，而不是相反的方式，但是诊断必须要由症状往回找原因。

在知识库系统中，我们经常为每条规则添加一个确信因子来度量结论的确信度。比如，规则 P→Q(0.9) 表示“如果你认为 P 为真，则 Q 为真的可能性为 90%”。这样，启发式的规则可以清晰地表示信念的确信度。

专家系统推理的另外一个问题是怎么从带有丢失、不完全、或者不正确信息的数据中提取有用的结果。我们可以用确定性的度量来反映我们对数据质量的信念，比如，断言 the lights have full power（0.2）说明前灯确实亮了，但灯光微弱，几乎看不见。信念和不完全的数据可以通过规则传播来对结论加以限制。

在本章中，我们讨论几个控制反绎推理和不确定性的方法，尤其是在知识密集型的问题求解中。9.1 节主要讲怎样扩展基于逻辑的形式方法来解决反绎的问题，包括由真值维护算法支持的非单调逻辑系统的运用。9.2 节考虑逻辑之外的其他方法，包括 Stanford 确信度代数、模糊推理和 Dempster-Shafer 证据理论。这些简单的演算是为了解决贝叶斯建造专家系统的方法的复杂性问题而提出来的。

9.3 节介绍用随机的方法进行不确定推理。这些技术是建立在贝叶斯对事件频率推理的理论的基础上的，这就需要这些事件的先验信息。我们用对图形模型的介绍总结本章，包括贝叶斯信念网络以及可观察的、隐式马尔可夫模型。有关用于学习的随机方法将在第 13 章介绍。

9.1　基于逻辑的反绎推理

首先，我们介绍基于逻辑的方法进行反绎。像第 8 章看到的那样，通过逻辑，一条条的知识精确地用于推理中，还能作为导出结论的部分解释。但是传统逻辑也有它的局限性，尤其是在信息丢失或不确定的情况下，在这些情况下，传统推理过程可能就难以派上用场。9.1 节给出几个传统逻辑的扩展让它来支持反绎推理。

9.1.1 节扩展逻辑来让它描述充满变化信息和信念的世界。传统的数理逻辑是单调的：它从一个公理集合开始，假定它为真，然后推出结论。如果我们对系统增加新的信息，就会引起真命题集合扩大。增加知识永远不会让真命题集合减小。当我们试图为建立在信念和假设基础上的推理建模的时候，这种单调性就会带来问题。在进行不确定推理的时候，人们从当前的信念集合来作结论，但是，不像数学公理那样，当更多信息可供使用时这些信念连同它们的结论会发生变化。

非单调推理解决变化信念的问题。非单调推理系统处理不确定性是通过借助于不确定信息做最合理的假设的方式来实现的。然后它进行推理，好像这些假设是真的。稍后，信念会改变，这就需要对从那个信念得出的结论进行重新验证。9.1.2 节中的真值维护算法紧接着就会被用来保持知识库的一致性。对逻辑的其他反绎扩展包括 9.1.3 节的“最小模型”和 9.1.4 节的“集合覆盖法”。

9.1.1　非单调推理逻辑

非单调性是人类求解问题和常识推理的一个重要特征。在大量的规划问题中，比如当我们

开车去上班的时候，我们对道路和交通情况作大量的假设。如果我们发现其中的一个假设被推翻了，或许是道路施工或者是路上的交通事故，我们会改变我们的计划，寻找其他的路线。

使用谓词逻辑的常规推理基于三个重要的假设。首先，谓词描述对于我们的应用领域来说必须是充分的。也就是说，解决问题所需的信息都能被表示出来。其次，信息库必须是一致的，也就是说，各条知识不能相互矛盾。最后，通过运用推理规则，得到的信息单调增长。如果上述三条假设中的任意一条不能满足，常规的基于逻辑的方法就不起作用了。

非单调系统解决了上述三个问题。首先，推理系统经常会缺少领域知识。一个重要的问题是：假设我们没有关于谓词 p 的知识，但缺少知识意味着我们对 p 是否为真不太确定还是我们确信 not p 为真呢？这个问题可以用许多种方式来回答。Prolog（参见 14.3 节）采用封闭世界假设把推理系统不能证明为真的都认为是假的。作为人类自身，我们经常采用其他的办法来假定一些事情为真，除非它可以被明确地看出是假的。

另外一个解决缺少知识的问题的办法是对为真的情况做明确的假定。在人们推理的时候，我们假定无辜的人是与犯罪无关的人。我们可能会进一步假定无辜的人是不能从犯罪中获益的人。进一步假定能有效地补充我们缺少的知识细节，基于这样的假定我们能扩展我们的推理得到新的结论。在 9.1.3 节中将讨论封闭世界假设和其他的一些办法。

人的推理是基于现实通常的情况。大多数鸟会飞。父母通常爱护并抚养它们的孩子。我们的推理是建立在同用现实世界的假设进行推理相一致的基础上的。在本节中，我们将讨论模态操作符的扩充，比如，is consistent with 和 unless，用它们来进行基于假设的推理。

传统的基于逻辑的系统所需的第二条假设是对知识的推理必须是一致的。对于我们推理者来说，这个假定给我们很大的限制。在分析一个问题时，我们通常会对一个情景给出多种解释，并假定一些是真的直到其他假定被证实更为有效。比如，在分析飞机事故的原因的时候，空难专家会给出多种原因，直到发现新的信息才排除（解释过去）一部分原因。有多种可能情况时，我们人用常识来试图指导推理。我们也希望逻辑系统能给出多种可能的假设。

最后，要想运用逻辑系统就必须解决知识库更新的问题。这里有两个问题：一个是怎么添加只基于假设的知识，另外一个是在后来发现假设不正确的时候怎么办。对于第一个问题，可以允许添加基于假设的新知识。这些新知识被假定是正确的，因此它们可以用来推出新的知识。这种做法的代价是跟踪基于假设的所有证据和推理：必须做好准备对基于假设的知识进行调整。

由于结论可能会调整，非单调推理被认为是可废除的，也就是说，新的信息可能会使前面的结果失效。跟踪逻辑系统中各步推理的表示和搜索程序称为真值维护系统，或 TMS。在可废除的推理中，TMS 保持知识库的一致性，记录可能会出现问题的结论。在 9.1.2 节中考虑多种真值维护的办法。现在考虑能让传统基于逻辑的推理系统可废除的操作符。

在实现非单调推理时，我们可以用操作符 unless 来扩展我们的逻辑。unless 支持在证据不为真的情况下基于信念的推理。假定我们有以下谓词逻辑语句：

```
p(X) unless q(X) → r(X)
p(Z)
r(W) → s(W)
```

第一条规则表明如果 p(X) 为真并且我们不认为 q(X) 为真的情况下我们可以推出 r(X)。当这些条件满足时，我们可以推出 r(X)，然后用 r(X)，我们可以推出 s(X)。这样，当我们改变信念，或者发现 q(X) 为真的时候，r(X) 还有 s(X) 就要收回。需要指出的是，unless 是来处理信念的问题而不是真假的问题。所以，把证据的值从“未知或者被认为是假”变成“被认为或者已

知为真”就会导致我们收回基于这些信念的推理。通过把逻辑扩展为用可能会被收回的信念来推理，我们把非单调性引入到了系统中。

上面讨论的推理机制还可以引入默认规则（Reiter 1980）。如果我们用 p(X) unless ab p(X)→r(X) 来代替 p(X) unless q(X)→r(X)，其中 ab p(X) 代表 abnormal p(X)，我们说除非我们有 p 的一个反常的实例，比如一只翅膀断了的鸟，我们可以有以下推理：如果 X 是一只鸟，那么 X 会飞。

扩展逻辑系统的另外一个模态操作符是由 McDermott 和 Doyle（1980）提出的。他们用模态操作符 M 来扩展一阶谓词逻辑，它置于谓词前面读作 is consistent with（与……相一致）。例如：

∀ X good_student(X) ∧ M study_hard(X) → graduates(X)

这个子句可以被读作：对于任意 X，X 是一个好学生，如果 X 学习刻苦与我们所知的其他事情相一致的话，X 就可以毕业。当然，这里困难的是对与我们所知的其他事情相一致的含义进行定义。

首先要指出，与我们所知的其他事情相一致这句话可能是不可判定的。原因是模态操作符会形成一个已知不可判定系统的超集，参见 2.2.2 节。有两种办法来解决不可判定的问题。首先，我们可以用失败即否定的证明来说明相一致。在我们的例子中，试图寻找 not(study_hard(X)) 的证据，如果不能证明 X 不学习，那就假定 X 学习。在类似于 Prolog 的近似于谓词逻辑的系统中，我们经常用这种办法。然而，这种办法会不恰当地缩小我们解释的空间。

解决“与……相一致”问题的第二种办法是对谓词的真假做基于启发式的和时间空间有限的搜索。例中的谓词 study_hard(X)，如果没有相矛盾的证据，就假定它为真，这意味着可能会收回毕业这个结论以及其他基于此得出的结论。

用 is consistent with 操作符我们可能会产生相矛盾的结论。假定某人 Peter 是一个好学生而且喜欢社团活动。我们可以用以下谓词集合来进行描述：

∀ X good_student(X) ∧ M study_hard(X) → graduates(X)
∀ Y party_person(Y) ∧ M not(study_hard(Y)) → not(graduates(Y))
good_student(peter)
party_person(peter)

我们没有 Peter 的学习习惯、他学习是否刻苦等进一步的信息，但是我们使用这些子句可以推出 Peter 毕业和 Peter 不毕业。

防止出现这种矛盾结论的办法是记录下与模态操作符 is consistent with 绑定在一起的变量。这样，一旦 Peter 结合到谓词 study_hard 或者 not(study_hard) 中，系统就会防止 Peter 再结合到其他谓词中。其他模态逻辑系统（McDermott 和 Doyle 1980）做法更为保守，它防止任何可能矛盾的子句集得出的结论。我们可以产生另外一个不规则集：

∀ Y very_smart(Y) ∧ M not(study_hard(Y)) → not(study_hard(Y))
∀ X not(very_smart(X)) ∧ M not(study_hard(X)) → not(study_hard(X))

从这些子句可以推出一个新的子句：

∀ Z M not(study_hard(Z)) → not(study_hard(Z))

模态操作符 is consistent with 的语义的进一步扩展解决了这样的不规则推理的问题。一个扩展就是自认知逻辑（Moore 1985）。

另外的一个非单调逻辑系统是由 Reiter（1980）提出的默认逻辑。默认逻辑采用以下新的推

理规则形式：

$$A(Z) \wedge : B(Z) \rightarrow C(Z)$$

读作：如果 A(Z) 可被证实并且它与我们所知道的对 B(Z) 的假定相一致，我们就可以推出结论C(Z)。

现在看起来默认逻辑与刚讨论过的 McDermott 和 Doyle 的非单调逻辑很相似。这两者之间的一个重要不同是推理的方式。在默认逻辑中，这些特殊的推理规则从原始的公理/定理集合中推出似真的推论。每个推论都是用默认逻辑推理规则从用原始公理/定理集合表示的知识推出来的。这样，一个原始知识库产生很多似真的推论就是很自然的了。我们可以从 graduates 子句中看出来：

$$\forall X\ good\_student(X) \wedge : study\_hard(X) \rightarrow graduates(X)$$
$$\forall Y\ party(Y) \wedge : not(study\_hard(Y)) \rightarrow not(graduates(Y))$$

基于原始知识集每个子句都能推出一个特有的似真的推论。

模态逻辑允许用似真的推论推出的任一定理作为进一步推理的公理。这就必须要有一些其他的策略来决定哪一个推论用来进行问题求解。默认逻辑没有涉及怎样从一个知识库的所有似真推论中进行选择的问题。Reiter（1978）、Reiter 和 Criscuolo（1981）和 Touretzky（1986）进一步解决了这些问题。

最后，非单调逻辑推理情况，它是通过继承搜索那些继承了多个父结点的对象表示时产生的。前面提到的喜欢社团的好学生 Peter，可以继承好学生的属性集，也就是说他很可能毕业。Peter 还可以从其他的作为社团成员那里继承属性，这种情况下会有点矛盾，这样，他就不能毕业。

非单调推理系统的主要问题是怎样有效地来修改根据变化的信念得出的结论。比如，如果我们用前提 r 推出 s，那么除掉 r 也就除掉了对 s 和其他用 s 得出的结论的支持。除非由一个独立的推理集来支持 s，否则 s 都会被收回。在坏的情况下，实现收回这一过程需要我们在信念每次变化的时候重新计算所有的结论。接下来将要讨论的*真值维护系统*将提供保持知识库一致性的机制。

9.1.2 真值维护系统

真值维护系统（TMS）可以被用来保持一个推理系统的逻辑完整性。像在前面指出的那样，每当用知识的子句来表示的信念被修改时，对知识库中的条目的支持情况重新计算是很有必要的。推理维护系统是通过存储每条推理的理由，再重新考虑根据新的信念得出的结论的支持情况。

解决这个问题的一个办法是对在 3.2.2 节中提到的回溯算法进行修改。回溯算法是在基于搜索的问题求解中在决策点遍历所有可能路径的一种系统方法。但是，回溯算法的主要缺点是它系统地（和盲目地）从空间的死端状态退回，在最近的选择中寻找可能路径。这种方法有时被称为*时序回溯*。我们承认时序回溯会系统地检查空间的所有可能情况，但是，它遍历的方式是费时的、低效率的、大空间的和无效的。

在基于逻辑的搜索中，我们真正想要的是直接回溯到空间中出现问题的点，并在那个状态对解进行修正的能力。这种方法被称为*相关性指导回溯*。我们来看一个非单调推理的例子。我们需要找到 p，它不能由直接推理得到。但有一个似真的假设 q，如果它为真，可以推出 p。所以我们假定 q 然后得到 p。我们的推理继续，由 p 我们得到 r 和 s。我们继续推理，没有 p、r 或者

s 的支持，我们可以得到结论 t 和 u。最后，我们得出先前的对 q 的假设是错误的。然后我们要做什么呢？

时序回溯会重新遍历我们推理的步骤，顺序同生成的时候正好相反。相关性指导回溯会立即退回到矛盾信息的源头，比如上面例子中的对 q 的假设。然后它向前收回 p、r 和 s。这时，我们检查 r 和 s 是否与 p 和 q 无关。这是因为它们是从不正确的假设推出来的，这并不意味着它们不被其他支持。最后，因为 t 和 u 是在没有 p、r 和 s 的情况下得到的，我们就不用对 t 和 u 进行调整。

要想在推理系统中运用相关性指导回溯算法，我们必须：

1）把每条结论与它的理由联系在一起。这个理由说明了结论得到的过程。它必须包括所有事实、规则和用来推出结论的假设。

2）提供一种机制，在给定矛盾和它的理由时，能找到导致矛盾的理由中的错误假设。

3）收回错误假设。

4）产生一种机制能追溯到收回的假设，并收回基于理由中收回的错误的假设的结论。

当然，所有收回的结论并不一定是假的，所以要重新检查它们看它们在没有收回的子句的情况下能否被证实。我们接下来给出两种建造相关性指导回溯系统的办法。

Jon Doyle（1979）研制了基于理由的真值维护系统（JTMS），它是最早的真值维护系统之一。Doyle 是最早明确地把真值维护系统、命题和理由组成的网络从一些领域中的推理系统分离出来的学者。分离的结果是 JTMS 与问题求解器（或许是自动定理证明程序）进行交互，它接收新的命题和理由，然后提供给问题求解器有关信息，即通过现有理由来判断哪个命题被认为是真的。

JTMS 主要有三个操作。首先，JTMS 检查理由网络。这个检查操作可以由问题求解器的查询来触发，比如：我是否要相信命题 p？我为什么要相信命题 p？什么假设支持命题 p？

JTMS 的第二个操作是修改相关性网络，由问题求解器提供的信息引起修改。修改包括添加新的命题，添加或删除前提，增加矛盾，对命题中的信念进行证实。JTMS 的最后一个操作是更新网络。只要相关性网络发生变化，这个操作就会被执行。更新操作重新计算所有的与现存理由相一致的命题的标签。

我们建造一个简单的相关性网络来演示 JTMS。下面的模态操作符 M 在 9.1.1 节中提到过，它放在谓词前面读作 is consistent with（与……相一致）。例如：

$$\forall X\ good\_student(X) \wedge M\ study\_hard(X) \rightarrow study\_hard(X)$$
$$\forall Y\ party\_person(Y) \rightarrow not\ (study\_hard(Y))$$
$$good\_student(david)$$

我们现在就把上面的命题集合放到理由网络中去。

在 JTMS 中，每个代表一个信念的谓词由其他两个信念集合来进行关联。第一个集合是对于一个信念来说支持它成立的信念集合，在图 9-1 中被标记为 IN。第二个集合是对于一个信念来说不支持它成立的信念集合，被标记为 OUT。图 9-1 表示了支持 study_hard(david) 的理由，它由图中列出的谓词推导出来。图 9-1 中的符号根据 Goodwin（1982）改编的，图 9-2 给出了解释。图 9-2a 表示理由的前提假设，图 9-2b 表示支持结论的命题的合取。

由图 9-1 网络中的信息，问题求解器能推出 study_hard(david) 是成立的，这是因为前提假设 good_student(david) 被认为是真，并且与好学生学习刻苦这个事实相一致。而且在这个例子中没有证据或隐含说明 David 学习不刻苦。

假定我们加入前提假设 party_person(david)。这样，我们就会得出 not(study_hard(da-

vid))，信念 study_hard(david)不再成立。图 9-3 反映了这种情况下的理由。需要指出的是 IN 和 OUT 发生了变化（重新标记）。

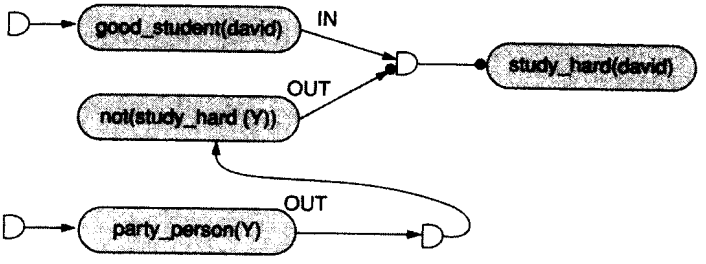
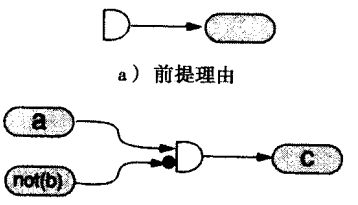


图 9-1 相信 David 学习刻苦的理由网络



a）前提理由

b）两个信念 a 和 not b 的并来支持 c

图 9-2 图 9-1 的符号解释（Goodwin 1982）

像图 9-1 和图 9-3 显示的那样，JTMS 并不直接反映原始命题集中所表示的谓词关系。JTMS 是一个简单的网络，它只考虑原子命题与原子命题的非之间的关系，并把它们组织起来支持信念之间的关系。整个谓词连接和推理机制（∀X、∧、∨、→，等等）用于问题求解器内部使用。McAllester（1978）和 Martins 及 Shapiro（1988）把 TMS 和问题求解器合并起来，使它们采用单一的表示。

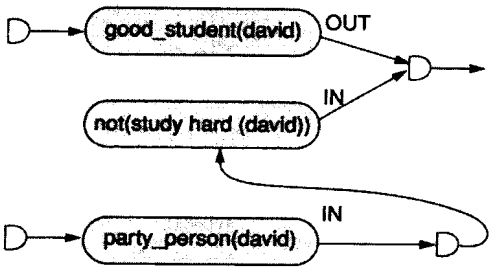


图 9-3 图 9-1 的新标记，即与新的前提 party_person（david）相联系

JTMS 只关心信念之间的相关情况，而不关心信念的内容。所以，我们可以用形如 n_1，n_2，…的标识符来代替信念，用它们来表示网络中连接的结点。像我们在 study_hard 例子中所看到的那样，IN 和 OUT 代数可以让 JTMS 对信念的支持情况进行推理。

总结一下，JTMS 就是在结点和理由的集合上来工作的。结点代表信念，理由支持结点上的信念。结点之间的联系是用 IN 和 OUT 来标记，它表示了连接结点的信念状态。通过与结点有 IN 和 OUT 关系的结点的情况，我们可以推出该结点是否成立。JTMS 的主要操作有完成检查、修改，以及更新上面提到的操作符。因为理由的检查是它自身回溯理由网络的连接来实现的，我们就举了一个相关性指导回溯的例子。想了解 JTMS 方法的更多信息，请参阅 Doyle（1983）或 Reinfrank（1989）。

第二种真值维护系统是基于假设的真值维护系统（ATMS）。虽然 Martins 与 Shapiro（1983）提出了相近的词语，但基于假设这个名词是由 deKleer（1984）首先提出来的。在这种系统中，网络中结点的标签不再是 IN 和 OUT，而是决定推导的前提（假设）集。deKleer 还区分了两种结点，前提结点可以通用，而由问题求解器通过前提推出的结点可能会被收回。

ATMS 比 JTMS 优越的地方就体现在 ATMS 提供了处理信念的多种可能状态的灵活性。通过对信念标记支持它的前提集，信念就不会只是单一状态了（在 JTMS 中所有的结点被标记为 IN），而是一组可能的状态，所有支持前提集的所有子集的集合。创建不同的信念集就会允许对选择不同前提得出的结论的比较，问题的不同的答案的存在，还允许矛盾的发现和恢复。ATMS 的不足包括不能表示自身非单调的前提集，不能控制问题求解器。有关解决问题的办法的讨论请参阅 Dressler（1988）和 Forbus and deKleer（1988）。

ATMS 和问题求解器的交互同 JTMS 和问题求解器之间的交互很相似，都有检查、修改和更

新操作。惟一的不同是在 ATMS 中不再是单一状态的信念，而是所有支持前提集的子集。ATMS 当中计算的目标就是寻求足以支持每个结点的最小前提集。这个计算就是从前提的标记开始，合并和传播标记。

接下来给出从 Martins（1991）改编的一个例子。假定我们有如图 9-4 的 ATMS 网络。在这个网络中，n_1、n_2、n_4 和 n_5 是前提，并假定为真。这个相关性网络还反映了这样的关系：从前提 n_1、n_2 我们得出 n_3，从 n_3 我们得出 n_7，从 n_4 我们得出 n_7，从 n_4 和 n_5 我们得出 n_6，最后，我们从 n_6 得出 n_7。

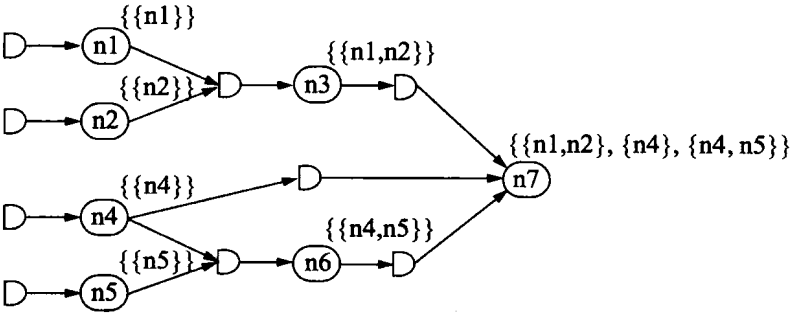


图 9-4　ATMS 对相关性网络中结点的标记

图 9-5 表示在图 9-4 中前提依赖关系的超集/子集格。这个子集格提供有效的方式让我们直观地观察前提集的组合空间。这样，如果我们怀疑某个前提假设是错的，就能知道与这个前提相关的其他的支持前提子集。比如，图 9-4 中的结点 n_3 会被图 9-5 中{n_1, n_2}上面的所有集合支持。

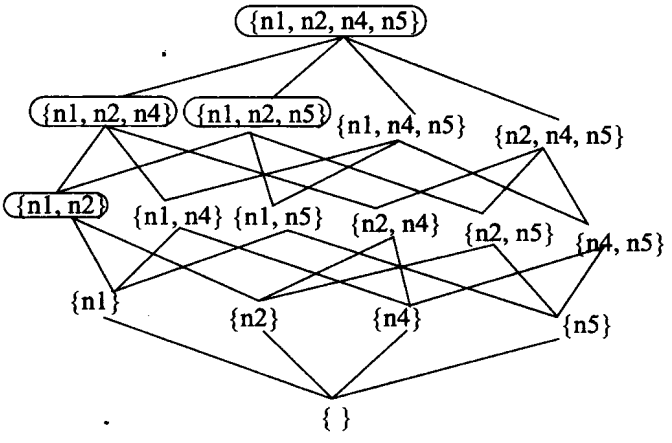


图 9-5　图 9-4 中网络前提的格

注：方框中的部分表示不一致的层次，改编自 Martins（1991）

ATMS 的推理程序通过删除已发现不一致的结点的前提集来消除矛盾。比如，假定我们修改图 9-4 中所反映的推理情况，使 n_3 成为矛盾结点。因为 n_3 的标记是 {n_1, n_2}，这个前提集就是不一致的。当这个不一致性被发现以后，构成图 9-5 中 {n_1, n_2} 的超集的前提集都会被标记为不一致，并从相关性网络中删除。这种情况下，一个支持 n_7 的标记就会被删除。对矛盾删除算法的详细描述请参阅 deKleer（1986）。

真值维护系统推理方面还有其他一些重要的成果。基于逻辑的真值维护系统（LTMS）是建立在 McAllester（1978）的工作的基础上的。命题之间的关系是由子句来表示的，这些子句可以

用来演绎任何它们所描述的命题的真假。另外一个方法——多信念推理程序（MBR），同 ATMS 推理程序很相似，只不过 MBR 的问题求解程序和真值维护系统合并成了一个单一的系统。MBR 是建立在描述知识状态的名叫 SWM * 的逻辑语言基础上的。每一知识状态由一个描述符对组成，第一个描述符描述知识库，第二个描述知识库中已知的不一致的前提集。推理中检查不一致性的算法可以在 Martins（1991）找到。关于真值维护系统请参见《The Encyclopedia of Artificial Intelligence》（Shapiro 1992）。

9.1.3 基于最小模型的逻辑

在前面的章节中，我们用几个模态操作符扩展了逻辑，引入这些逻辑符是为了对现实世界通常的情况进行推理，放松对于知识完备性的要求，希望能用一种更加灵活和可更正的观点来看待现实世界。在本节中，我们将给出专门为两种情况设计的逻辑：第一种情况是，给定一个断言的集合，其中只指定了为真的断言；第二种情况是，由于问题求解的本质，合取范式的集合通常是正确的。第一种情况下我们用封闭世界假设，对第二种情况我们用限定（circumscription）。这两种办法我们都可以看成是在最小模型上的推理。

在2.3节中，我们看到，模型就是对所有变量赋值满足谓词表达式集合 S 的解释。有很多种方式来定义最小模型的含义。我们定义最小模型为对所有变量赋值满足谓词表达式集合 S 的模型当中没有比它更小的那个模型。

最小模型思想产生的非常重要的原因是可以有无限多个谓词来描述现实世界中的某种情景。比如，我们可以考虑有无数的谓词来描述“农夫－狼－山羊－卷心菜”问题（3.5节，练习2）的情景：船没有缓慢下沉，河的两岸离得很近，划船就可以过去，风和水流的因素不予考虑。当我们描述一个问题时，我们不愿意去做过多无用的描述，我们只愿意去创建与问题相关并且是求解问题所必需的谓词。

封闭世界假设是建立在世界的最小模型的基础上的。求解所必需的谓词都会被创建。封闭世界假设实现否定的语义。比如，我们想知道一个学生是否是一个班级的注册成员，我们可以去察看注册数据库，如果在数据库（最小模型）中没有明确地列出来，那么他或她就不是注册成员。类似地，如果我们想知道两个城市之间是否有直达的班机，我们会察看所有航班的列表。如果直达航班不在列表（最小模型）中，我们就会认为不存在直达航班。

封闭世界假设认为如果我们的计算机系统中不含有 p(X) 为真，那么 not(p(X)) 就为真。在14.4节中，将会看到封闭世界假设支持 Prolog 推理。在14.4节中，我们还将看到在使用最小模型时所隐含的三个假设（公理）。这三个公理是：名称惟一性，即所有的原子都有惟一的名称；封闭世界，关系的惟一实例是由现有的子句推出来的；域封闭，即域中的原子正好是模型中的原子。当这三个条件都满足时，最小模型就是一个完整的基于逻辑的规范系统。如果公理不能满足，就需要真值维护算法。

如果封闭世界需要给定组成模型的所有谓词，限定（McCarthy 1980，Lifschitz 1984，McCarthy 1986）只需要给定问题求解相关的那些谓词。在限定中，公理是加到系统中去的，系统中知识库的谓词必须有最小解释。“元谓词”（有关问题定义的陈述谓词）描述特定的谓词是怎么被解释的。就是说，它们划界或者限定谓词可能的解释。

McCarthy（1980）提出了限定的概念，并试着用来解决传教士和食人者问题。这个问题要求问题求解器给出六个物体在一定的约束条件下用船过河的一系列的动作描述。McCarthy 给出了问题相关的大量的情形，有些情形看起来有些可笑，却是合理的。其中的一部分，如正在下沉的船或风的因素，在前面提到过。McCarthy 加到问题描述中的限定公理会精确地对问题的描述加以

限制。

作为限定的另外一个例子，看一下 9.4 节面向对象常识推理中的一个谓词表达式：

$$\forall X\ bird(X) \wedge not(abnormal(X)) \rightarrow flies(X)$$

在对鸟会飞这一属性进行推理时，就可能会用到这一表达式。但是怎么限定谓词 abnormal 的含义呢？是鸟不是企鹅，是翅膀没断掉，还是鸟没有死？谓词 abnormal 的含义是不定的。

在一阶谓词演算中用公理机制或元规则集来产生问题域的谓词。元规则会使一些公式产生最小扩展。例如，如果 B 是包括全局知识 K 和有关谓词 p 的领域知识 A(p) 的信念系统。我们就会认为 p 是最小化的，即满足谓词 $p(a_i)$ 并且与 K 和 A(p) 相一致的原子 a_i 最少。全局知识 K 连同 A(p) 和限定机制是用来用标准的一阶谓词演算来推出结论。这些结论接着再添加到信念系统 B 中。

假设是在 8.4 节的积木世界中，有表达式：

$$isblock(A) \wedge isblock(B) \wedge isblock(C)$$

断言 A、B 和 C 是积木。限定谓词 isblock 给出：

$$\forall X(isblock(X) \leftarrow ((X = A) \vee (X = B) \vee (X = C)))$$

这个表达式断定仅有的积木是 A、B 和 C，即谓词 isblock 指定的积木只有 A、B 和 C。以类似的方式，谓词：

$$isblock(A) \vee isblock(B)$$

可以被限定为：

$$\forall X(isblock(X) \leftarrow ((X = A) \vee (X = B)))$$

要了解更多的细节，包括用来推出结论的表示模式公理（schema axiom），请参阅 McCarthy (1980，第 4 节)。

当使用诸如 abnormal 的操作符时，限定很像封闭世界假设，因为它正好产生出 abnormal 能支持的那些变量绑定。但是限定代数允许我们在谓词表示上扩展这个推理，因为，像我们刚才指出的，如果我们有谓词 $p(X) \vee q(X)$，我们可以限定谓词 p 或 q，或者它们两个。于是，不同于封闭世界假设，限定允许我们在谓词描述集合上描述可能的例示。

对限定逻辑进一步的研究可以在 Genesereth and Nilsson（1987）中找到。Lifschitz（1986）做出了重要贡献，他提出了 point - wise 限定，最小模型可以为特定的谓词和它们可能的例示来执行，而不是对整个域。另外一个重要的贡献是 Perlis（1988）提出的关于一个缺少知识的主体的推理。更多讨论可以参阅 Shapiro（1992）。

9.1.4　集合覆盖和基于逻辑的反绎

像在本章概述中所提到的那样，在反绎推理中，我们给定规则 $p \rightarrow q$ 和 q 的合理信念。然后我们希望在某种解释下得到谓词 p 为真。反绎推理是不可靠的，但因为 q 的原因，它又被称为*最佳解释推理*。在本节中，我们将讨论反绎推理域中解释的产生。

除了已经提到的反绎推理的方法外，人工智能的研究者还用集合覆盖和基于逻辑的分析的办法。*集合覆盖*的办法试图解释这样的作法，即对某些可解释的假设采用可废除的信念，因为它解释了用别的方法不可解释的集合。*基于逻辑*的办法描述反绎的规则连同它们合法形式的定义。

*集合覆盖法*定义一个反绎的解释为谓词的覆盖，谓词通过描述假设来描述现象。Reggia et

al. (1983) 描述了一个基于因果关系 R 的覆盖，R 是集合 {Hypotheses X Observations} 的子集。于是，现象集合 S2 的反绎解释是足以引起 S2 的另一个假设集合 S1。根据集合覆盖法，最佳解释是 S2 的最小集合覆盖。这种方法的缺点是它把解释约简为原因假设（从 S1 中）的一个简单列表。在有相关的或交互作用的原因假设的情况下，或者需要理解原因交互作用结构或次序时，集合覆盖模型就是不充分的。

基于逻辑的办法则是建立在解释的更高级的概念的基础上。Levesque (1989) 定义某些前面无法解释的现象集合 O 为假设集 H 中与主体背景知识 K 一致的最小集合。假设 H 连同背景知识 K 必须能推导解释出 O。更形式化一点：

abduce(K, O) = H，当且仅当：

1) K 不能推导解释出 O；

2) H∪K 能推导解释出 O；

3) H∪K 是一致的，并且；

4) 不存在 H 的子集有属性 1、2 和 3。

需要指出的是，总的来说，可能会存在许多假设集，也就是说，对一个给定的现象 O 可能会有很多潜在的反绎解释集。

基于逻辑的反绎解释的定义暗示了发现知识库系统中的内容的解释有相应的机制。如果可解释的假设必须能推导解释出现象 O，则建立一个完整的解释的方式就是从 O 向后推理。像 3.3 节和 8.2 节那样，我们可以从 O 的合取范式开始，从结果推回祖先。

这种后向式推理的办法是很自然的，因为支持后向推理的条件可以被认为是因果律，于是我们得出因果知识在产生解释时起关键作用。这个模型是方便的，因为它很好地符合了人工智能界已有的经验：演绎的后向式推理和计算模型。

还有很多寻找反绎解释的完整集合的方法。基于假设的真值维护系统 ATMS（deKleer 1986，9.2.3 节）包含了计算最小支持集的算法，支持集是能逻辑上推导出给定命题的命题（非公理）集合。为了找到一个现象集合的所有可能的反绎解释，我们只在支持集上采用 Cartesian 产生式。

基于逻辑的反绎具有简单、精确、方便的优点，它还有两个相应的缺点：高的计算复杂性和语义缺陷。Selman 与 Levesque (1990) 发现反绎任务的复杂性与计算 ATMS 的支持集的复杂性相似。ATMS 问题是 NP - hard 的标准证明依赖于有指数个解的问题实例的存在性。Selman 与 Levesque 通过询问是否找到也是 NP - hard 的更小的解集合的方法来避免潜在的解的复杂性问题。给定 Horn 子句的知识库，见 14.2 节，Selman 与 Levesque 给出了一个算法，找到一个解释的阶为 O(k * n)，其中 K 表示命题变量的数量，n 表示文字出现的数目。但是，当对要找的解释的种类加以限制时，问题就又变为 NP-hard，即使是对 Horn 子句。

Selman 与 Levesque (1990) 分析得出的令人感兴趣的结论是这样一个事实：对反绎任务增加一定的目标或限制会使计算变得很难。从人类问题求解的观点来看，这个增加的复杂性有些让人不解：人们通常假想对相关解释的搜索进行进一步的限制会使任务变容易。在基于逻辑的模型中反绎任务变难的原因在于它只是对问题的额外的子句有影响，而对问题求解有用的额外的结构没有影响。

基于逻辑的模型中解释的发现是以找到具有一定逻辑属性的假设集为特征的。这些属性，包括与背景知识的一致性和能推导解释出要解释的现象，是指抓住解释的必要条件：可解释的假设集成为反绎解释所必需满足的最小条件。这种办法的支持者认为通过增加附加限制，这种办法可以通过扩展来提供好的或合理的解释的特征。

产生有质量的解释的一个简单的策略是定义反绎的事实子句集，也就是说，候选假设必须

选上。这个子句集允许搜索事先限定一些因素，这些因素在被选域中是潜在的原因。另外一个策略是增加评价和挑选解释的选择标准。人们已经提出了很多不同的选择标准，包括集合最小性，它是指当两个假设都是一致的且都能推导解释出要解释的现象时，如果第一个假设包含于第二个假设，则选择第一个假设。简单性标准优先选择简单的假设集，即含有更少的通用假设的假设集（Levesque 1989）。

最小性和简单性都可以看成是奥卡姆剃刀的应用。不幸的是，集合最小性作为搜索修剪工具能力有限，它只去掉是已有解释的超集的最终解释。只是简单性自身作为搜索选择标准有效性也值得怀疑。建立这样的例子是不难的，需要大的解释集的解释优先于简单却浅显的解释集。实际上，复杂的因果机制通常需要大的假设集，但是，这种因果机制的反绎可以被证实，尤其是当这个机制的某个关键元素的存在性已经被现象证实的情况下。

解释选择的另外两种机制也很令人感兴趣，因为它们不仅考虑了假设集的属性，还考虑了证明过程的属性。第一个机制，基于代价的反绎对于潜在的假设和规则赋一个代价值。解释的总代价是用假设的代价加上用来反绎假设的代价计算出来的。竞争假设集然后根据代价来进行比较。这种方案附带的自然的启发是概率（Charniak and Shimony 1990）。假设的代价高代表不太可能的事件，规则的代价高代表不太可能的因果机制。基于代价的度量可以与最小代价搜索算法结合起来，如最佳优先搜索算法，参见第 4 章，这种算法大大降低了任务的计算复杂性。

第二个机制，基于一致性选择（coherence-based selection），当要解释的不是一个命题而是命题集合时尤其适用。Ng 与 Mooney（1990）认为在分析自然语言文本过程中选择解释时一致性度量比简单性要重要。他们把一致性定义为证据图的一个属性，图中任意对现象之间有更多连接和更少分开的部分的解释是更一致的。一致性标准是基于这样一个启发式的假设：我们要解释的是一个单一的事件或者有多个方面的动作。自然语言理解中一致性度量（coherence metric）的证明是基于 Gricean 幸运（felicity）条件，即说话者的职责是要一致和相关（pertinent）（Grice 1975）。我们不难把他们的论点扩展到其他情形中去。例如在诊断中，包含初始要解释的事情的现象被综合在一起，因为它们被认为与同一个潜在的故障或失败机制相关。

在 9.1 节中，我们讨论了传统逻辑的扩展，以支持不确定或缺少数据的推理。我们接下来讨论非逻辑的方法在不确定情况下进行推理，包括 Stanford 确信度代数、模糊集推理和 Dempster-Shafer 证据理论。

9.2　反绎：逻辑之外的办法

9.1 节中基于逻辑的办法对许多应用尤其是专家系统来说非常麻烦，难于处理。许多早期的专家系统（像 PROSPECTOR）试图采用另外一种办法，即 9.3 节中介绍到的用贝叶斯方法进行反绎推理。这种办法的局限性在于对独立性的要求，对统计数据不断地更新以及进行随机推理所需的演算。另一种减轻这些限制的推理机制在斯坦福大学用于开发早期的专家系统，包括 MYCIN（Buchanan and Shortliffe 1984）。

当人类专家在用启发式的知识进行推理时，他们能够给出对结论的充分有效的估计。人们用类似“很有可能”、“不太可能”、“几乎是”、或“可能的”这样的词语来度量结论的确信度。这些度量显然没有基于对概率的精确分析，而是他们自身从对问题推理的经验中得出的启发式信息。在 9.2 节中将介绍三种反绎推理的方法理论，即 Stanford 确信度理论、模糊推理和 Dempster-Shafer 证据理论。在 9.3 节中用随机的办法来进行不确定性推理。

9.2.1　Stanford 确信度代数

Stanford 确信度理论是建立在很多现象的基础上的。首先在传统的概率论中，一个关系的支

持它的确信度和不支持它的确信度的和加起来为1。但是，现实中常有这种情况——专家可能对某个关系为真的确信度为0.7，对于关系为假的情况则毫无概念，不置可否。另外一个支撑确信度理论的假设是规则中含有的知识远比计算确信度的代数重要。确信度的度量要与专家对结论的非形式化的估计相一致，比如“它可能为真”，“它几乎为真”，或者“它不太可能”。

Stanford 确信度理论对建立确信度的度量做了一些简单的假定，在推理时也有一些简单的规则来合并这些确信度。第一个假定是把一个关系的支持度和不支持度分开：

MB(H | E)称为给定证据 E 时假设 H 的可信度量。

MD(H | E)称为给定证据 E 时假设 H 的不可信度量。

现在或者

当 MD(H | E) =0 时 1 > MB(H | E) >0, 或者

当 MB(H | E) =0 时 1 > MD(H | E) >0

这两个度量互相限制，给定的证据对于特定的假设或者支持，或者不支持，这是确定性理论同概率论的一个重要不同。一旦可信度量和不可信度量的联系建立起来，它们就能连接在一起，这是通过下面的公式实现的：

CF(H | E) = MB(H | E) - MD(H | E)

当确信度因子（CF）接近于1时，证据倾向于支持假设，当 CF 接近于 -1 时，证据倾向于不支持假设，当 CF 在大约为0时，表明只有极少的证据支持或者不支持假设，或者支持或者不支持假设的证据大体相当。

当专家要建一个规则库时，它们必须为每一条规则确定一个 CF。CF 反映了专家对这条规则的可信赖程度的信心。确信度的度量可以被用来调整系统的性能，虽然度量的小的改变对整个系统的总体运行情况几乎没有多大影响。确信度的度量的这个作用验证了“知识产生力量”这句话，也就是说，完整的知识能对正确的诊断提供最好的支持。

每条规则的前提由 and、or 和一些事实组成。当用到产生式时，前提中每个条件的确信度就会按照如下方式被合并成对整个前提的确信度。对规则的前提 P1 和 P2：

CF(P1 and P2) = MIN(CF(P1), CF(P2))

CF(P1 or P2) = MAX(CF(P1), CF(P2))

用上面的规则合并后的前提的 CF 接着乘上规则自身的 CF 就得到规则结论的 CF。例如，知识库中的规则：

(P1 and P2) or P3→R1(0.7) and R2(0.3)

其中 P1、P2 和 P3 是前提，R1 和 R2 是规则的结论，确信度 CF 分别为0.7 和0.3。如果规则所有的前提是完全确定的，当这条规则被用来代表专家对结论的确信度时，这些数字就会附加到规则上。如果当前系统已经产生了 P1、P2 和 P3，确信度 CF 分别是0.6、0.4 和0.2，R1 和 R2 这时的确信度 CF 就是0.28 和0.12。下面是计算过程：

CF(P1(0.6) and P2(0.4)) = MIN(0.6, 0.4) = 0.4

CF((0.4) or P3(0.2)) = MAX(0.4, 0.2) = 0.4

R1 的确信度 CF 是0.7，所以 R1 添加到特例知识集中去的时候的确信度 CF 为 (0.7) × (0.4) = 0.28。

R2 的确信度是 0.3，所以 R2 添加到特例知识集中去的时候的确信度 CF 为(0.3) × (0.4) =0.12。

当两个或多个规则支持同一结论 R 时，怎么来合并这么多个 CF 呢？这个度量生成规则反映了确信度理论与概率论中合并独立证据时度量相乘的相似之处。循环运用这条规则，我们可以合并支持结论 R 的任意数目的规则。假定此时结论 R 的当前的确定度为 CF(R1)，以前没用到的规则也得出结论 R，确信度为 CF(R2)。那么 R 的新确定度 CF 用下面的公式来计算：

当 CF(R1)和 CF(R2)都为正时 CF(R1) +CF(R2) -(CF(R1) ×CF(R2))

当 CF(R1)和 CF(R2)都为负时 CF(R1) +CF(R2) +(CF(R1) ×CF(R2))

其他情况下，

$$\frac{CF(R1)+CF(R2)}{1-MIN(|CF(R1)|,|CF(R2)|)}$$

其中符号 | X | 代表 X 的绝对值。

除了容易计算之外，这些合并规则还有其他一些好的特性。首先，计算出来的 CF 总是在 1 和 -1 之间。第二，在合并符号相反的 CF 时，它们互相削弱，这是我们想要得到的。第三，合并后的 CF 是一个单调递增（递减）的函数，这也是我们合并证据时所期望的。

最后，Stanford 确信度代数中对确信度的度量是人们对于症状或起因的概率值的主观估计。像在 5.4 节中指出的那样，在贝叶斯方法中，如果 A、B 和 C 都对 D 有影响，当我们要对 D 进行推理时，我们需要区分和适当地合并所有的先验和后验概率，包括 P(D)、P(D | A)、P(D | B)、P(D | C)、P(D | AB)，等等。Stanford 确信度代数允许把所有的关系合成一个确信度 CF 来附加到规则上，即 if A and B and C then D(CF)。这种简化的办法更好地反映了人类专家是怎么合并和传播信念的。

确信度理论可能会被认为过于特殊。虽然它有形式代数定义，但是确信度度量的含义与形式概率论不完全一致。另一方面，确信度理论并不是用来做“正确”的推理的，而是能让专家系统在解决问题时合并确信度使得推理继续下去的“润滑剂”。确信度的度量有些特殊，同样，人类专家对自己结论的确信度也是近似的、启发式的和非形式的。当 MYCIN 运行时，确信度被用来在启发式搜索中给出目标的优先级，在一个目标不用再被考虑时给出终止点。但即便是确信度被用来维持程序的运行和搜集信息，系统的性能好坏仍然主要取决于规则的质量。

9.2.2　模糊集推理

使用形式集合论时，有两个重要的假定。第一个是关于集合成员关系：对于任意一个元素和在某个全域中的一个集合来说，这个元素或者在这个集合中，或者在这个集合的补集中。第二个假定称为排中律，是指一个元素不能既属于一个集合，又属于这个集合的补。Lotfi Zadeh 的模糊集理论冲破了这两个假定。事实上，在模糊集理论的观点看来，传统集合论的集合和推理被称为干脆的。

Zadeh 的主要思想（Zadeh 1983）是：虽然概率论是对信息的随机性进行度量的，但是用它来对信息的含义进行度量是不恰当的。实际上，我们运用英语单词和词组时经常混淆，这应该算是不清楚（含糊）而不是随机性。对于分析语言结构这是很重要的一点，在确定产生式规则的确定性时也同样重要。就像概率论度量随机性一样，Zadeh 提出了可能性理论（possibility theory）来度量模糊程度。

Zadeh 用量化的方式来表达不精确性，为此他引入了取值在 0 和 1 之间的集合隶属函数。模

糊集的概念可以描述如下：令 S 为集合，s 是这个集合的一个元素。S 的模糊子集 F 由隶属函数 mF(s)来定义，这个函数度量了 s 属于 F 的“程度”。

图 9-6 给出了模糊集的一个例子，S 是正整数集合，称为小整数集合的 F 是 S 的模糊子集。于是不同的整数对于小整数集合的模糊隶属度有一个“可能性”分布：mF(1) =1.0，mF(2) =1.0，mF(3) =0.9，mF(4) =0.8，…，mF(50) =0.001，等等。对于语句正整数 X 是一个小整数，mF 创建对于所有正整数（S）的一个可能性分布。

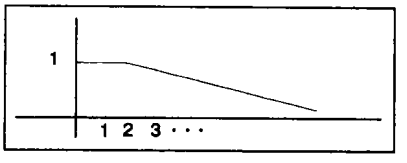


图 9-6 “小整数”的模糊集表示

模糊集理论并不关心这些可能性分布是怎么创建的，而是关心用什么规则来计算合并含有模糊变量的表达式的可能性。所以，它包含合并含有模糊变量的表达式的可能性的值的规则。对于表达式中的 or、and 和 not，合并规则与 Stanford 确定度代数相似：参见 9.2.1 节。

图 9-6 所示，对于代表小整数的模糊集，每个属于这个集合的整数都有相对应的置信度。对传统逻辑的“干脆”集来说，集合中元素的隶属值或者是 1 或者是 0。图 9-7 给出了男性对于矮、中等、高这三个概念的隶属函数。需要指出的是，每个人可以属于不只一个集合，例如，一个 5 英尺 9 英寸的男性既属于中等集合，又属于高的男性集合。

接下来求解一个问题来演示一下模糊集的合并和传播规则，即对一个倒立摆怎样控制的问题，它现在成了模糊集著作中很经典的问题。图 9-8 给出了一个倒立摆，我们希望摆能保持平衡，并指向上方。我们通过移动系统底部来抵消作用于摆的重力的办法来保持系统的平衡。有一组微分方程可以使摆保持平衡（Ross 1995）。用模糊集方法来控制摆系统的优点在于生成的算法可以有效和实时地控制系统。我们接下来将讨论这种控制机制。

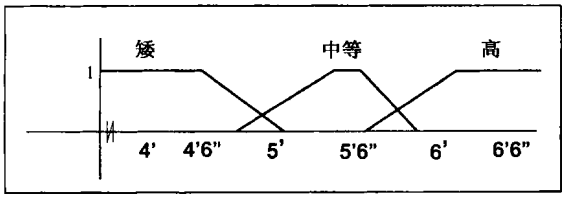


图 9-7 矮、中等和高个男性集合的模糊集表示

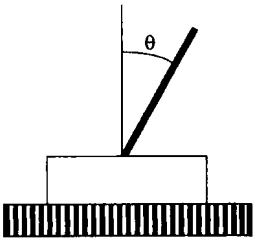
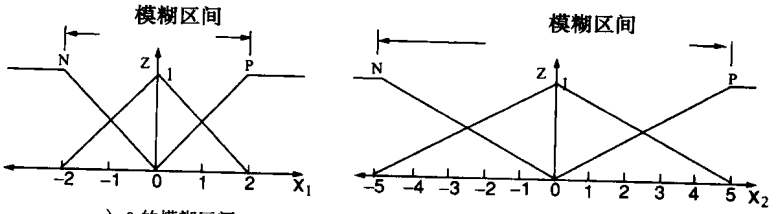


图 9-8 倒立摆，角度 θ 和 dθ/dt 的输入值

我们用两个维度来刻画摆问题，以简化这个问题。控制程序有两个输入量度，在图 9-8 中我们可以看到：第一个是角度 θ，是摆与垂直中线的偏移量，第二个是 dθ/dt，是钟摆移动的速度。这两个量度都是垂直线的右边为正，左边为负。系统每次反复，这两个量都要提供给模糊控制程序。控制程序的输出是对系统的底部给出一个移动和一个方向。移动和方向的指示是为了保持钟摆的平衡。

为说明控制程序的动作，用模糊集描述求解过程。描述钟摆状态的数据 θ 和 dθ/dt 被解释成模糊量度，如图 9-9 所示，并送到模糊规则集。这一步通过称为模糊联想矩阵（FAM）的结构可以有效地实现，如图 9-12 所示，矩阵中输入/输出的关系直接用编码表示了出来。这里的规则并不像传统的基于规则的问题求解那样链在一起。而是所有匹配的规则结合在一起，它们的结果合并起来。如图 9-10 所示，这个结果通常表示为模糊输出参数空间中的一个区间，然后这个结果再经过模糊反变换，返回控制响应。需要指出的是，开始的输入和最终的输出都是纯数字。它们是作为输入的某些监视器的准确读数和作为输出的对控制的精确指示。



a）θ 的模糊区间　　b）d θ /dt 的模糊区间

图 9-9　输入值的模糊区间

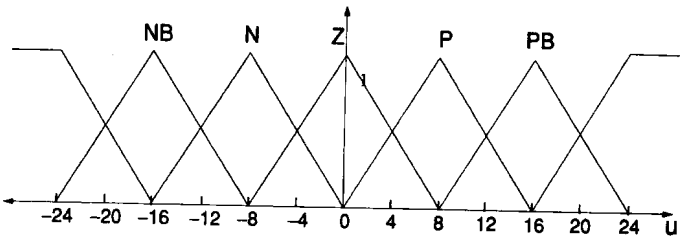


图 9-10　输出值 u 的模糊区间，u 指示摆底部的移动

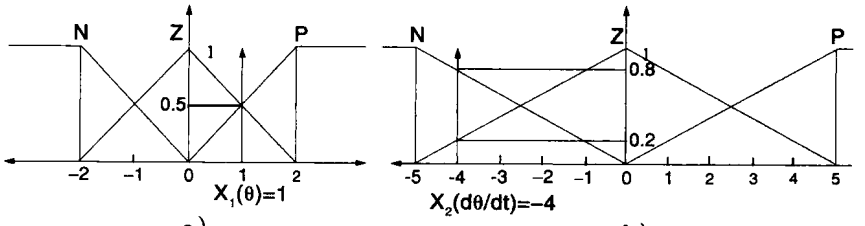


图 9-11　输入值 $x_1 = 1$、$x_2 = -4$ 的模糊化

接下来讨论输入值 θ 和 dθ/dt 的模糊区间。这个例子简化了一些情形，比如，输入一个输入值模糊区间中的数，但显示整个程序的过程和对控制器的响应。输入值 θ 被划分成三个区间：负、零和正，如图 9-9a 所示，θ 在 -2 ~ +2 弧度的区间取值。图 9-9b 给出了第二个输入值 dθ/dt 被划分成的三个区间，同样是负、零和正，取值区间是 -5 ~ +5 度/秒。

图 9-10 表示输出区间的取值，我们用中间的五个区间，大负、负、零、正和大正。度量值介于 -24 ~ +24 之间，表示每个响应的移动和方向。

X_1 \ X_2	P	Z	N
P	PB	P	Z
Z	P	Z	N
N	Z	N	NB

图 9-12　倒摆问题的模糊联想矩阵（FAM）。输入值在左侧和顶部

现在假定模拟开始，第一次送给控制器的值是 θ = 1 和 dθ/dt = −4。图 9-11 反映了对这些输入量的模糊变换。对于每种情况，输入值作用于两个模糊输入空间中的两个区间。对于 θ，对零的可能性度量为 0.5，对正的可能性度量为 0.5。对于 dθ/dt，对负的可能性度量为 0.8，对零的可能性度量为 0.2。

图 9-12 给出了针对这个问题的模糊联想矩阵的简化形式。表的左边表示输入 θ，或者 x_1，上边表示输入 dθ/dt，或者 x_2。FAM 表右下角的值表示输出值。例如，如果 θ 是正，并且 dθ/dt 是负，FAM 返回给摆系统的移动值是零。需要指出的是响应必须由图 9-10 对输出区间零进行模糊反变换。

在这个例子中，因为每个输入值都处在输入空间的两个区间上，所以必须用到四个规则。像

前面指出的那样，模糊集的合并规则与 Stanford 确信度代数相似。事实上，Zadeh（Buchanan 和 Shortliffe 1984）是第一个（历史上）提出模糊推理代数的合并规则的人。如果两个前提的度量用 AND 来连接，则度量中最小的那个被作为这条规则的度量。如果两个前提用来 OR 连接，则选度量中最大的。

在我们的例子中，所有的前提用 AND 来连接，所以将之中最小的度量作为规则结果的度量：

IF x_1 = P AND x_2 = Z THEN u = P
　　min(0.5, 0.2) = 0.2 P

IF x_1 = P AND x_2 = N THEN u = Z
　　min(0.5, 0.8) = 0.5 Z

IF x_1 = Z AND x_2 = Z THEN u = Z
　　min(0.5, 0.2) = 0.2 Z

IF x_1 = Z AND x_2 = N THEN u = N
　　min(0.5, 0.8) = 0.5 N

接下来进行结果的合并。在我们的例子中，按照规则集的结果所示画出图 9-10 中的联合区域。有许多种模糊反变换的技术（Ross 1995）。我们选择最常用的一种：质心法。这种办法是把联合区域的质心作为控制程序输出给摆的最终结果。联合区域以及区域的质心如图 9-13 所示。在输出作用于系统之后，θ 和 dθ/dt 又被抽样，控制循环又重复下去。

在描述模糊推理系统时，还有很多问题没有提到，包括属于收敛过程的摆动的模式和最佳抽样率的选取。模糊系统尤其是在控制领域给我们提供了一个强大有效的工具来处理度量不准确的问题。

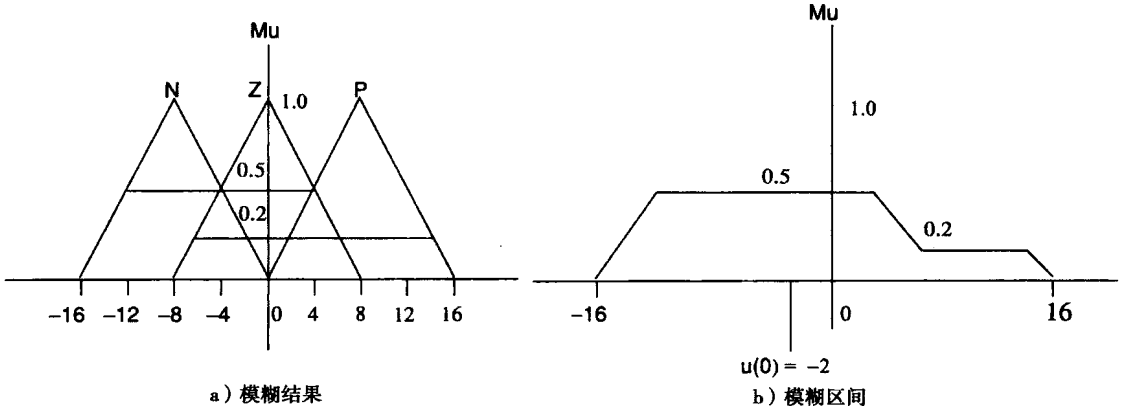


图 9-13　模糊结果和它们的区间

注：这个区间的质心（-2）是干脆的输出

9.2.3　Dempster-Shafer 证据理论

到现在我们讨论的不确定条件下的推理技术是给定证据集，我们赋给单个命题可能起因或者信任程度的估计值。用概率的办法来求解不确定问题的一个局限是它们用一个单一的数字来度量一个可能很复杂的情形。通常，不确定性来源于缺少证据的合并、启发式规则固有的局限和人们知识的局限性。

另外一种办法称为 Dempster-Shafer 证据理论，考虑的是命题集，并赋给区间值［belief,

plausibility]，每个命题的可信度（belief measure）必须在这个区间内。这个可信度表示为 bel，取值从 0 到 1，取值为 0 表示没有证据支持这组命题，取值为 1 表示完全确定。命题 p 的似真性 pl（p）是这样定义的：

$$pl(p) = 1 - bel(not(p))$$

于是，似真性也是从 0 到 1 取值，反映了 not(p) 的证据是怎么与信任 p 的可能性相联系的。如果我们对 not(p) 有确定的证据，那么 bel(not(p)) 就为 1，pl(p) 就为 0。对 bel(p) 的惟一可能的值也是 0。

假定我们有两个竞争的假设（competing hypotheses）h_1 和 h_2。当我们没有支持每个假设的信息时，它们的信任度/似真性在 [0, 1] 上取值。当我们有了证据，我们希望区间缩小，表示对于假设支持度增加。在贝叶斯定理中，我们开始可能（这时没有证据）把先验概率等分到两个假设中去，即每个 $P(h_i) = 0.5$。Dempster-Shafer 明确说明开始时没有证据，另一方面，贝叶斯方法则不管我们有多少数据也能得出相同的概率值。这样，在基于收集到的证据数量来做结论的时候，Dempster-Shafer 证据理论就非常有用。

总结一下，Dempster 和 Shafer 通过区分不确定和不知道来解决度量确定程度的问题。在概率论中，我们必须对假设 h 用一个数字给出它的概率，p(h)。Dempster 和 Shafer 指出概率论办法的问题在于我们并不一定总是知道概率的值，因此我们给出的 p(h) 可能没有经过证实。

Dempster-Shafer 信念函数能满足概率论不能满足的问题，也就是说，当所有的概率都给定时，它就成为概率论了。信念函数允许我们在缺少准确的概率的情况下，用知识对事件的概率赋值。

Dempster-Shafer 理论的基础思想有两点：第一，是从相关问题的主观概率来得到一个问题可信度的思想；第二，当基于相互独立的证据时，合并可信度的规则的使用。合并规则是由 Dempster（1968）最早提出来的。下面我们给出 Dempster-Shafer 推理的一个不很正式的例子，然后给出规则。最后，把规则运用到一个更现实的情况下。

假定我对朋友 Melissa 的可信赖程度有一个主观的概率。她是可信赖的可能性为 0.9，不可信赖的可能性为 0.1。假设 Melissa 告诉我，我的计算机被别人侵入了。如果 Melissa 是可信赖的，则这是真的，但如果她是不可信赖的，这句话则不一定为假。所以 Melissa 一个人的陈述证实我的计算机被侵入的可信度为 0.9，没有被侵入的可信度为 0.0。0.0 的可信度与 0.0 的概率不同，它并不意味着我确信计算机没有被侵入，而只是表明 Melissa 的陈述没有给我理由来相信计算机没被侵入。在这种情况下，似真性 pl 为：

$$pl(computer\_broken\_into) = 1 - bel(not(computer\_broken\_into)) = 1 - 0.0$$

我对 Melissa 的可信度为 [0.9, 1.0]。需要指出的是仍然没有证据表明我的计算机没被侵入。

我们接下来讨论 Dempster 理论中合并证据的规则。假设我的朋友 Bill 也告诉我，我的计算机被侵入了。假定 Bill 可信赖的概率为 0.8，不可信赖的概率为 0.2。我还必须假定 Melissa 和 Bill 的有关计算机的陈述互相独立，也就是说，他们分别有各自的原因来告诉我这件事。事件 Bill 是可信赖的必须独立于事件 Melissa 也是可信赖的。Melissa 和 Bill 都是可信赖的概率是 0.72，都是不可信赖的概率是 0.02。至少有一人是可信赖的概率是 1 - 0.02，即 0.98。因为他们两人都说我的计算机被侵入，并且至少有一人是可信赖的概率是 0.98，我将给事件计算机被侵入的信任度赋值为 [0.98, 1.0]。

假设 Melissa 和 Bill 的陈述不一致：Melissa 说我的计算机被侵入了，Bill 说没有侵入。在这种情况下，他们不可能都是可信赖的。或者他们都是不可信赖的，或者只有一人是可信赖的。只

有 Melissa 是可信赖的先验概率是 0.9 ×（1 - 0.8）= 0.18，只有 Bill 是可信赖的先验概率是0.8 ×（1 - 0.9）= 0.08，两人都不可信赖的先验概率是 0.2 × 0.1 = 0.02。给定至少有一人是不可信赖的概率为（0.18 + 0.08 + 0.02）= 0.28，我们还可以算出只有 Melissa 是可信赖的后验概率是 0.18/0.28 = 0.643，这时计算机被侵入，或者只有 Bill 是可信赖的后验概率是 0.08/0.28 = 0.286，这时计算机没被侵入。

我们刚才运用了 Dempster 规则来合并信念。当 Melissa 和 Bill 都说计算机被侵入时，总结一下支持侵入的三种情形：Melissa 和 Bill 都是可信赖的；只有 Bill 是可信赖的；只有 Melissa 是可信赖的。信念 0.98 是这几种支持情形的和。第二次运用 Dempster 规则时，两人的意见不一致。同样，我们也总结一下所有可能的情形。惟一不可能的情形是他们都是可信赖的；因此，或者只有 Melissa 是可信赖的，或者只有 Bill 是可信赖的，或者两人都是不可信赖的。这三种情形给出了对于计算机被侵入的可信度 0.643。计算机没被侵入（Bill 的观点）的可信度为 0.286，由于计算机被侵入的似真性为 1 - bel(not(break in))，即 0.714，所以可信度为 [0.28，0.714]。

在运用 Dempster 规则时，我们从另外一个问题（这两个人是否可信赖）的概率来获取这个问题（计算机是否被侵入）的概率。这个规则开始时假定我们的概率是独立的，但这只是前提假定。当不同的证据矛盾时，这个前提假定就不成立了。

在特殊情况下运用 Dempster-Shafer 的办法需要解决两个相关的问题。首先，我们把不确定的情况转化为假定独立的证据。第二，我们执行 Dempster 规则。这两个任务是相关的。假定 Melissa 和 Bill 都告诉我计算机被侵入了，而且是独立的。并假定我还找了个维修工来检查我的计算机，而且 Melissa 和 Bill 都看到了。因此，就不用再比较可信度了。但是，如果明确考虑维修工来检查计算机的可能性，那么就有三个独立的证据：Melissa 的可信赖性、Bill 的可信赖性和维修工到场的证据，接下来就可以用 Dempster 规则来合并它们。

假定有一个互相不相关的假设的完整集合，称为 Q。目标是对 Q 的不同子集 Z 给出信念的度量 m，m 有时称为 Q 的子集的概率密度函数。现实中，不是所有的证据直接支持 Q 的元素。实际上，证据通常支持 Q 的不同子集 Z。另外，由于 Q 的元素假定是互相独占的，支持其中一部分的证据可能会对其他部分的信念有影响。在纯贝叶斯系统中，见 9.3 节，我们是通过列出所有条件概率的合并式的办法来解决上述情况中的问题的。在 Dempster-Shafer 系统中，我们是通过直接操作假设集来处理这些问题的。数 $m_n(Z)$ 度量了赋给子集 Z 的信念，n 代表证据的数量。

Dempster 规则指出：

$$m_n(Z) = \frac{\sum_{X \cap Y = Z} m_{n-2}(X) m_{n-1}(Y)}{1 - \sum_{X \cap Y = \phi} m_{n-2}(X) m_{n-1}(Y)}$$

例如，假设为 Z 有 n = 3 个证据的信念 $m_3(Z)$ 是 $m_1(X)$ 和 $m_2(Y)$ 两种假设情形的和，它们同时成立时支持 Z，即 $X \cap Y = Z$。Dempster 规则的分母表示（在后面的例子中会看到）X 和 Y 可以有空的交集，确信度的和必须用 1 减去它们的和来常规化。

接下来在医疗诊断的情况中运用 Dempster 规则。假设 Q 表示我们关注的领域，包含四个假设：一个病人患有伤寒（C）、流感（F）、偏头疼（H）或者脑膜炎（M）。我们的任务是对 Q 之内的假设集赋予信念的度量。像已经提到的那样，它们是假设集，因为证据不需要专门支持这些假设。例如，发烧可能支持{C, F, M}。由于 Q 的元素可看成是相互独占的假设，支持其中一部分的假设可能影响其他的信任度。像已经提到的那样，Dempster-Shafer 方法是通过直接操作假设集来处理交叉影响问题。

对概率密度函数 m 和所有 Q 的子集 Z，数 $m(q_i)$ 代表当前赋给 Q 的每个 q_i 的信念，所有 $m(q_i)$ 的和等于 1。如果 Q 包含 n 个元素，那么有 2^n 个 Q 的子集。虽然赋 2^n 个值可能会令人感到沮丧，但通常很多子集并不出现。这样，求解过程就可以简化，因此这些值就可以忽略掉。最后，Q 的似真性 $p1(Q) = 1 - \sum m(q_i)$，其中 q_i 是有一些支持信念的假设集。如果没有任何有关假设的信息，就像开始诊断时的情形，那么 $p1(Q) = 1.0$。

假设第一条证据是病人发烧，对{C, F, M}的支持为 0.6。我们称第一条信念为 m_1。如果这是仅有的一条假设，那么 $m_1\{C, F, M\} = 0.6$，而 $m_1(Q) = 0.4$，表示信念的剩余的分布。需要指出的是 $m_1(Q) = 0.4$ 表示的信念的剩余分布是指 Q 中所有其他可能的信念，而不是{C, F, M}补集的信念。

假设现在获取一些新的数据来用于诊断，即病人反胃恶心，它对{C, F, H}的支持度为 0.7。我们把这个信念称为 m_2，有 $m_2\{C, F, H\} = 0.7$，$m_2(Q) = 0.3$。我们用 Dempster 规则来合并这两个信念，m_1 和 m_2。令 X 为 m_1 赋给一个非零值的 Q 中的子集，Y 为 m_2 赋给一个非零值的 Q 中的子集。然后用 Dempster 规则产生一个定义在 Q 的子集 Z 上的合并信念 m_3。

在运用 Dempster 规则来进行诊断时，首先要指出的是没有集合 $X \cap Y$ 是空的，所以分母是 1。m_3 的信念分布见表 9-1。

表 9-1　用 Dempster 规则来获取 m_3 的信念分布

m_1	m_2	m_3
$m_1\{C,F,M\} = 0.6$	$m_2\{C,F,H\} = 0.7$	$m_3\{C,F\} = 0.42$
$m_1(Q) = 0.4$	$m_2\{C,F,H\} = 0.7$	$m_3\{C,F,H\} = 0.28$
$m_1\{C,F,M\} = 0.6$	$m_2(Q) = 0.3$	$m_3\{C,F,M\} = 0.18$
$m_1(Q) = 0.4$	$m_2(Q) = 0.3$	$m_3(Q) = 0.12$

运用 Dempster 规则，四个集合 Z，所有可能的 X 和 Y 的交集，构成了表 9-1 的最右边的一列。它们的信念量度是通过 m_1 和 m_2 下 X 和 Y 对应元素的信念相乘而得到的。还要指出的是在这个例子中 Z 的每个集合都是惟一的，但通常不是这种情况。

最后再扩展一下我们的例子来说明空信念集在分析中是怎么出现的。假设我们有一个新的事实，实验室文化与脑膜炎相关。现在有 $m_4\{M\} = 0.8$，$m_4(Q) = 0.2$。可以用 Dempster 规则来合并 m_4 和前面的分析结果 m_3 来得到 m_5，见表 9-2。

表 9-2　用 Dempster 的规则来合并 m_3 和 m_4 来得到 m_5

m_3	m_4	m_5 (without denominator)
$m_3\{C,F\} = 0.42$	$m_4\{M\} = 0.8$	$m_5\{\} = 0.336$
$m_3(Q) = 0.12$	$m_4\{M\} = 0.8$	$m_5\{M\} = 0.096$
$m_3\{C,F\} = 0.42$	$m_4(Q) = 0.2$	$m_5\{C,F\} = 0.084$
$m_3(Q) = 0.12$	$m_4(Q) = 0.2$	$m_5(Q) = 0.024$
$m_3\{C,F,H\} = 0.28$	$m_4\{M\} = 0.8$	$m_5\{\} = 0.224$
$m_3(C,F,M) = 0.18$	$m_4\{M\} = 0.8$	$m_5\{M\} = 0.144$
$m_3\{C,F,H\} = 0.28$	$m_4(Q) = 0.2$	$m_5\{C,F,H\} = 0.056$
$m_3(C.F,M) = 0.18$	$m_4(Q) = 0.2$	$m_5\{C,F,M\} = 0.036$

注意，$m_5\{M\}$是两对不同集合的交集，所以总的 $m_5\{M\} = 0.240$。还有这种情况：不同集合的交集为空集 $\{\}$。这样 Dempster 规则的分母是 $1 - (0.336 + 0.224) = 1 - 0.56 = 0.44$。$m_5$ 最后的信念函数是：

$m_5\{M\} = 0.545$　　$m_5\{C,F\} = 0.191$　　$m_5\{\} = 0.56$
$m_5\{C,F,H\} = 0.127$　　$m_5\{C,F,M\} = 0.082$　　$m_5(Q) = 0.055$

最后有三点评论。第一，如果赋予空集一个很大的信念，如 $m_5\{\} = 0.56$，那么说明在信念集中有矛盾的证据。实际上，设计这个例子是用来展示 Dempster-Shafer 推理的多个特征的，因此牺牲了医学含义的完整性。第二，虽然如同前面指出的那样，Dempster-Shafer 推理比贝叶斯推理计算量要小得多，但是当假设集很大而且证据集很复杂时，信念集的计算会变得很麻烦。第三，在较强的贝叶斯假设没有证实的情况下，Dempster-Shafer 办法是一个很有用的工具。

Dempster-Shafer 是支持推理中运用主观概率的代数的一个例子。有时我们感觉主观概率更好地反映了人类专家的推理。在接下来的 9.3 节中，将讨论更多基于贝叶斯规则扩展的推理技术，贝叶斯规则在 5.3 节介绍过。

9.3 处理不确定性的随机方法

使用概率论，我们可以从先验论证中确定事件发生的几率。还可以描述事件的合并是怎么相互影响的。虽然最终建立概率论是由 20 世纪初的数学家（包括 Fisher、Neyman 和 Pearson）完成的，构造合并代数的研究却可以追溯到中世纪的希腊人，包括 Llull、Porphyry 和 Plato（Glymour et al. 1995a）。概率论的现实基础是，如果我们可以知道这个事件以前发生的频率，那么就可以用这个信息（作为归纳偏置）来解释和推理现在的数据。

在第 5 章，我们列举了很多适合概率分析的情况。例如，真正随机的环境，像用洗好牌的扑克玩游戏或旋转均匀的轮盘赌轮。此外，虽然世界上有很多事情不是真正随机的，但通常不可能估量所有条件及其相互作用以预测事件的发生；统计相关性是解释世界的有用的工具。统计的另一个用途是作为自动归纳和机器学习的基础（如 10.3 节中的 ID3 算法）。最后，近年来人们试图把概率和因果关系的概念联系起来（Glymour and Cooper 1999，Pearl 2000）。

在随机领域中，最初的推理机制是某种形式的贝叶斯规则。然而，在 5.3 节介绍过，在复杂领域中完全使用贝叶斯推理会使问题很快就变得难以处理。专门设计的概率图模型可说明这个复杂性。概率图模型是“……概率论和图论相结合的产物”（Jordan 1999）。

在机器学习（见第 13 章）中，概率图模型变得越来越重要，因为它们可以描述不确定的和复杂的问题，描述对于现代人工智能有基本局限的已知问题。Jordan 指出，模块性的问题是图模型的基础：简单的模块通过概率论结合在一起，这符合整体系统的一致性，并同时将图模型和应用数据相结合（Jordan 1999）。同时图论为图模型既提供了用于表示强交互组件集的直观方法，也提供了支持高效推理算法的数据结构。

在 9.3 节和第 13 章讨论两类概率图模型：有向的、贝叶斯信念网络和多种形式的马尔可夫模型；无向的、团树和马尔可夫随机场。马尔可夫场和其他无向图模型可以表示很多循环依赖，这用有向图模型无法表示。另一方面，与无向图模型相比，在获取隐含推理关系和依赖方面，有向图模型贝叶斯信念网络是更为精确和有效的。

下面的几节中，我们介绍贝叶斯信念网络和几种专门设计的推理技术，它们可以描述无向图模型和/或全贝叶斯推理的计算复杂性。之后，在 9.3.5 节中，引入马尔可夫过程并提及几个有代表性的重要变量。其中一个是动态贝叶斯网络，另一个是离散马尔可夫过程。

贝叶斯信念网络和传统的隐马尔可夫模型中的主要局限之一就是它们固有的命题本质。我

们通过几种将命题逻辑合并的尝试来说明这种局限，如使用概率图模型，全基于变量的谓词计算等。在 9.3.7 节中对这些方法进行了概括，并在第 13 章再次涉及一阶概率系统。

9.3.1　有向图模型：贝叶斯信念网络

第 5 章介绍过，虽然贝叶斯信念网络为不确定条件下的推理提供了一个数学基础，但是其复杂性使它难以应用到实际问题领域中去。幸运的是，可以通过把搜索集中于更相关的事件和证据的小集合上的办法来削减复杂性。一种方法是使用贝叶斯信念网络（Pearl 1988），它提供了在问题域中期望的因果关系下推理到数据集合的最佳解释的一个计算模型。

贝叶斯信念网络能够显著减少完整贝叶斯模型的参数数量，并展示了域中的数据（甚至没有数据）是怎么划分和推理的。更进一步的，问题域的模块性使得程序设计者可以做出许多完整贝叶斯方法中不允许的独立性假设。在大多数推理条件下，没有必要建立一个大的联合概率表把所有可能的事件和证据的组合的概率都列出来。实际上，人类专家只是选取他们认为会有影响的现象，并获取反映这一小部分事件的概率或影响数值。专家假定所有其他的事件或者是独立的，或者是它们的关联很小可以忽略掉。贝叶斯信念网络使这种直觉更为精确。

作为贝叶斯信念网络的一个例子，再次考虑 5.4 节中的交通问题，表示在图 9-14 中。回忆一下，道路施工是 C，事故是 A，存在橙色桶是 B，堵车是 T，闪烁的灯是 L。要计算例子中所有参数的联合概率，采用完全贝叶斯方法，需要特定状态下所有参数的知识或度量方法。因此，用变量的拓扑排序，联合概率就是：

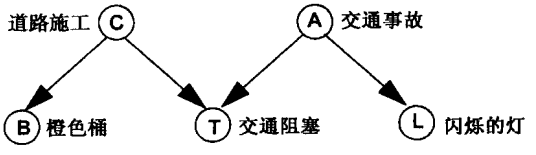


图 9-14　5.3 节中首次介绍的交通问题的图形模型

$$p(C, A, B, T, L) = p(C)\,p(A \mid C)\,p(B \mid C, A)\,p(T \mid C, A, B)\,p(L \mid C, A, B, T)$$

这个联合概率中参数的数量是 2^5，即 32。这个表会随着包含的参数的数量而呈指数增长。对于一个任意复杂度的问题，比如说有 30 个以上的参数，那么联合分布表将会有超过 1 百万个元素！

然而注意，如果可以假定这个问题中的所有参数只与双亲结点的概率相关，即假定结点与其他祖先无关，给定它们双亲结点的知识，$P(C, A, B, T, L)$ 的计算就变为：

$$P(C, A, B, T, L) = P(C)\,P(A)\,P(B \mid C)\,P(T \mid C, A)\,P(L \mid A)$$

为了更好地看一下所做的简化，来看以前等式中的 $P(B \mid C, A)$。在上面的等式中，把它简化为 $P(B \mid C)$。这是基于道路施工不是交通事故的起因这个假设。类似地，橙色桶的出现不是交通阻塞的起因，但是道路施工和交通事故是交通阻塞的起因，所以有结果 $P(T \mid C, A)$ 而不是 $P(T \mid C, A, B)$。最后，$P(L \mid C, A, B, T)$ 简化为 $P(L \mid A)$。$P(C, A, B, T, L)$ 的概率分布现在只有 20 个（而不是 32 个）参数。如果看一个带有 30 个变量的更实际的系统，并且每种状态至多有两个双亲结点，那么分布中最多只有 240 个元素。如果每种状态有 3 个双亲结点，分布中最多只有 490 个元素：这比完全贝叶斯方法所需的 10 亿要少得多多了！

在信念网络中，只需要验证结点和它的双亲结点之间的独立性。信念网络中结点之间的连接表示因果关系下的条件概率。专家运用因果推理时隐含的假定是这些影响是有向的，即某个事件不知为何会引起网络中的其他事件。还有，因果推理是不能循环的，即结果不能推回原因自身。因为这些原因，贝叶斯信念网络可以用有向无环图（DAG）（见 3.1.1 节）来表示，这样一个连贯的推理可以看成是因果关系的路径。贝叶斯信念网络是所谓的图模型的一个实例。

在交通的例子中，我们的条件更强，即没有无向环路。这就让我们对每个结点的概率分布的计算非常简单。没有双亲结点的分布可以直接查到。孩子结点的数值的计算只用到双亲结点的概率分布，即对孩子结点的条件概率表和双亲结点的分布做适当的计算。这是可能的，因为我们不必考虑其他非后继结点的双亲结点之间的关联（因为该网络是有向无环图）。这就产生了一种很自然诱导的分离，如图9-14中，交通事故与橙色桶的出现无关。

我们用下面的定义来总结关于贝叶斯信念网络的讨论。

定义 一个图模型称为**贝叶斯信念网络**（Bayesian belief network，BBN），如果图上标注条件概率，并且是有向无环的。另外，给定父结点的知识，BBN假定结点独立于非后继结点。

动态贝叶斯网络（dynamic Bayesian network，DBN）是一些等价的贝叶斯网络，结点之间以时间维（有向）连接。我们在9.3.4节和第13章中对一般的DBN进行讨论，也可以参阅Friedman（1998）或Ghahramani和Jordan（1997）。

下一节，将考虑人类专家推理时隐含的假定：（隐含因果）域中数据的有无可以划分和集中搜索解释。这种现象对于减小搜索空间复杂性也有重要的意义。

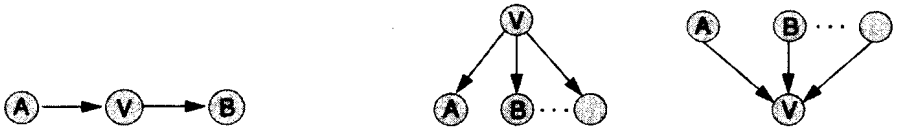
9.3.2 有向图模型：d－可分

将应用领域表示为图模型的一个重要优点是信息的有无可能使模型可分割，从而控制搜索的复杂度。下面我们给出几个例子和支持这些直觉知识的d－可分（d-separation）的概念。

第一个例子，考虑汽车发动机的油量诊断问题：假设用旧的活塞环引起油的过量消耗，这又会引起很低的油量读数。这种情况如图9-15a所示，图中A代表用旧的活塞环，V代表油的过量消耗，B代表很低的油量。现在，如果并不知道油的过量消耗，那我们就会有用旧的活塞环和很低的油量之间的因果关系。但是，如果某些测试能给出油过量消耗的状态，那么用旧的活塞环和很低的油量就是相互独立的。

第二个例子：假设用旧的活塞环可以引起排出蓝色废气，还可以引起很低的油量。这种情况如图9-15b所示，图中V代表用旧的活塞环，A代表排出蓝色废气，B代表很低的油量。如果我们知道用旧的活塞环或者为真，或者为假，则我们不知道排出蓝色废气和很低的油量是否是相关的；如果不知道用旧的活塞环的真或假，则排出蓝色废气和很低的油量是相关联的。

最后，如果很低的油量可以由油的过量消耗或者漏油引起，给定油量是否很低的知识，它的两个可能的起因就是相关的。如果很低的油量的状态未知，则这两个可能的起因是独立的。而且如果很低的油量为真，则建立漏油为真，就会排除解释油的过量消耗。在上面两种情况中，有关很低的油量的信息是推理过程的关键因素。在图9-15c中就可以看到这种情况，图中A代表油的过量消耗，B代表漏油，V代表很低的油量。



a）顺序连接　　b）分支连接　　c）汇合连接

图9-15 图9-15a是结点的顺序连接，如果V没有实例化，则A和B之间有影响。图9-15b是分支连接，如果V没有实例化，V的孩子结点之间有影响。图9-15c是汇合连接，如果V是未知的，则它的双亲结点是独立的，否则它的双亲结点之间存在关联

通过定义信念网络（Pearl 1988之后版本）中结点的d－可分的方法来使这些直觉知识更加

精确。

定义（d－可分）　有向循环图的两个结点 A 和 B 是 d－**可分的**，如果它们之间的所有路径都是阻塞的。一条路径是图中任何连续的连接序列（结点之间以任意方向来连接，例如图 9-15b 中有一条从 A 到 B 的路径）。如果一条路径上有这样的中间结点 V，这条路径就是阻塞的，结点 V 有下面两个属性的任意一个属性：

- 连接是**连续的**或者**有分支的**，并且结点 V 的状态是已知的。
- 连接是**汇合的**，并且结点 V 和它的孩子结点都没有证据。

下面给出连续、分支、汇合结点关系的实例，还有 d－可分是怎么影响关键路径的，如图 9-16所示。

我们再回过头来看一下图 9-15，说明一下贝叶斯信念网络的假定是怎样简化条件概率的计算的。用贝叶斯公式，任意联合概率分布可以转化为条件概率相乘的形式。在图 9-15a 中，变量 A、V 和 B 的联合概率是：

$$P(A, V, B) = P(A)P(V \mid A)P(B \mid A, V)$$

运用贝叶斯信念网络的假定，即给定所有前驱结点的知识的一个变量的条件概率等于只给定其双亲结点的知识的条件概率，可将上面等式中的 P(B | A, V) 变成 P(B | V)，因为 V 是 B 的直接双亲结点，而 A 不是。图 9-15 中三个网络的联合概率分布是：

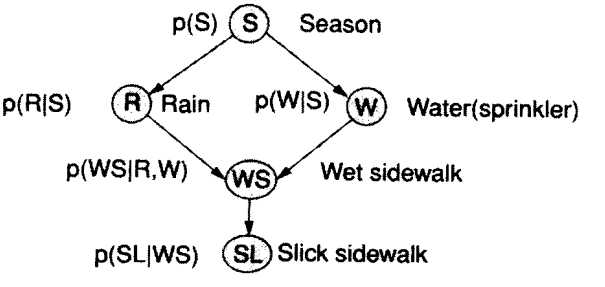


图 9-16　贝叶斯概率网络示例，每个结点旁边标出了概率依赖。该例子来自于 Pearl（1988）

a）$P(A, V, B) = P(A)P(V \mid A)P(B \mid V)$

b）$P(V, A, B) = P(V)P(A \mid V)P(B \mid V)$

c）$P(A, B, V) = P(A)P(B)P(V \mid A, B)$

像在交通的例子中显示（见图 9-14）的那样，对一个大的贝叶斯信念网络，条件概率中的很多变量可以消掉。正是这些简化才使得信念网络和其他图模型比完全贝叶斯网络统计起来更容易处理。下面我们介绍一个更加复杂的包含无向环的图模型，并且提供一个有效的推理算法，团树传播（clique tree propagation）。

9.3.3　有向图模型：一个推理算法

接下来的一个例子，改编自 Pearl（1988），给出了一个更复杂的贝叶斯网络。如图 9-16 所示，季节（season）决定了下雨（rain）和洒水系统洒水（water）的概率。湿的人行道（wet sidewalk）与下雨和洒水有关。最后，人行道是否滑（slick）取决于它是否是湿的人行道。在图中，表示出了概率的联系，这些参数每个都有自己的双亲结点。还要指出的是，同交通的例子相比，人行道的例子如果是无向图，则有环路。

我们要问，湿的人行道的概率 p(WS) 怎么来表示？这不能像前面那样去完成，即 $p(W) = p(W \mid S)p(S)$ 或者 $p(R) = p(R \mid S)p(S)$。WS 的两个起因不是互相独立的，例如，如果 S = summer，那么 p(W) 和 p(R) 都会增大。这样，这两个变量的相互关系和它们与 S 的相互关系就必须计算出来。在这种情况下，我们可以做，但是这个计算是随 WS 的原因数目呈指数增长的。这个计算如表 9-3 所示。现在只计算表中的一个条目 x，其中 R 和 W 都为 true；我们简化一下，

假定季节 S 或者是 hot，或者是 cold。

$$x = p(R=t, W=t) \text{ 对于季节 } S \text{ 的两种 (hot, cold) 情况}$$
$$= p(S=hot)\,p(R=t \mid S=hot)\,p(W=t \mid S=hot) +$$
$$P(S=cold)\,p(R=t \mid S=cold)\,p(W=t \mid S=cold)$$

用类似的方法，表9-3中剩余部分也可以计算出来。这样就构成了下雨和洒水系统洒水的联合概率。这个较大的“宏元素”代表 $p(WS) = p(WS \mid R, W)\,p(R, W)$。我们这里摆脱掉了一个更为合理的计算，如上所述，问题是这个计算是随状态的双亲结点数呈指数增长的。

表9-3 P(WS)的概率分布，P(WS)是给定 S 的影响，P(W)和 P(R)的函数。我们计算在 R=t 和 W=t 时 x 的影响

R	W	p(WS)	
t	t	x	S=hot
t	f		S=cold
f	t		
f	f		

为计算 p(WS)，我们称这个宏元素为组合变量（combined variable）或团（clique），用于计算 p(WS)。我们引入团的概念是为了用无环团树（clique tree）来替换图9-16中 DAG 的约束传播，如图9-17所示。图9-17a中的矩形框表示与它相邻的上面和下面的团所共有的变量。传递相关参数给下一个相邻的团的表，随着参数的数目呈指数增长。需要指出的是团中必须有一个连接变量和它所有的双亲结点。于是，在建立信念网络或其他图模型（知识工程的过程）时，需要注意有多少个变量是一个状态的双亲结点。团还是重叠的，如图9-17b所示，通过团的完全树来传递信息，团的完全树称为连接树。接下来给出由 Lauritzen 和 Spiegelhalter（1988）提出的从一个贝叶斯信念网络产生一个连接树的算法。

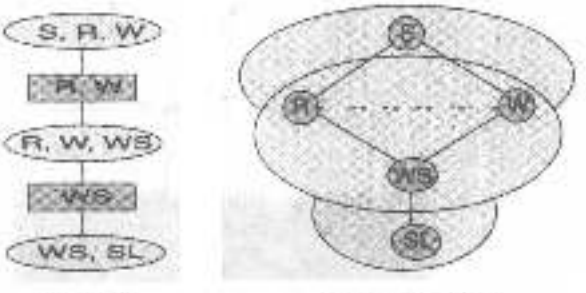
1）对信念网络中的所有结点，把有向的连接变为无向连接。

2）对任意结点，在它的双亲结点之间连线（图9-17b中 R 和 W 之间的虚线）。

3）寻找结果图中长度超过3的循环，增加额外的连接使循环的长度减到3。这个过程称为三角测量。

4）用得到的三角化的结构形成连接树。这是通过寻找最大团（是完全子图且不是一个更大的团的子图的团）的方法来完成的。这些团中的变量被放入连接中，通过连接任意两个至少共有一个变量的连接来产生连接树，如图9-17a所示。

上面第3步中描述的三角测量过程至关重要，因为我们想要连接树在传播信息时计算代价最小。不幸的是，这个设计最优代价连接树的决策是一个 NP-hard 问题。幸运的是，通常一个简单的贪心算法就足以产生有用的结果。注意图9-17中的连接树传递信息所需的表的大小是：2*2*2、2*2*2 和2*2。



a）连接树　　b）贝叶斯概率网络

图9-17 贝叶斯概率网络的连接树

注：注意我们开始建立对矩形 R、W 的转移表

最后，我们采取图9-16中网络的例子，并返回到 d－可分的问题。记住，d－可分的要点是通过某些信息，信念网络的某些部分在计算概率分布时可以忽略掉。

1）给定 WS 是已知的，SL 从 R、S 和 W 中 d－可分。

2）d－可分是对称的，即给定 WS 是已

知的，S 从 R，SL 和 W 中 d－可分。

3）由于 S、R 和 W 是依赖的，但知道 S 时，R 和 W 是 d－可分的。

4）如果我们知道 WS，则 R 和 W 不是 d－可分的；如果不知道 WS，则 R 和 W 是 d－可分的。

5）给定链 R→WS→SL，如果我们知道 WS，则 R 和 SL 是 d－可分的。

在知道一个特定状态的后代的信息时，我们必须小心。例如，如果知道 SL，则 R 和 W 不是 d－可分的，因为在知道 WS、R 和 W 不是 d－可分的情况下，SL 与 WS 相关。

一个最终的评论：贝叶斯信念网络近似地反映了人们怎么在这样的情况下进行推理：在复杂的领域中某些区域是已知的，并与其他相联系形成一个先验知识。因为推理仅随着信息的进一步的实例化，搜索被进一步限制，结果搜索更有效率。这个搜索的效率与使用完整联合分布的方法形成强烈的对比，信息越多，对统计联系的需要越强烈，并呈指数增长，同时也需要更广泛的搜索。

关于建立信念网络和当获取到新的证据时传播论点，有许多算法可用。我们尤其推荐 Pearl（1988）的消息传递方法，以及 Lauritzen 与 Spiegelhalter（1988）提出的团树三角形算法。Druzdel 与 Henrion（1993）还提出了在网络中传播影响的算法。Dechter（1996）给出了桶排除算法作为概率推理的一个统一框架。

在下一节中，我们介绍动态贝叶斯网络。

9.3.4　有向图模型：动态贝叶斯网络

下面我们来看一种一般化的贝叶斯信念网络，动态（或时间）贝叶斯网络（dynamic，or temporal，Bayesian network，DBN），支持 DBN 的思想是用一组随机变量表示域的状态或观测的时间，而不仅仅是单独的随机变量。使用这种扩展，贝叶斯信念网络可用于表示不同视角的变量之间的条件独立性，如分布于多个时间片中。

大多数人类已经遇到的和准备说明应对的事件，都和时间有关；它们都会在某段时间表现出来。图 9-18 和图 9-19 是两个例子。在图 9-18 中，我们使用前面的图 9-14 的交通模型，将其映射到时间片上，这或许能反映司机的视点随时间的变化，假设逐分钟地，由贝叶斯信念网络的参数改变来反映。在第一个时间片司机只知道车速降低了，第二个时间片注意到橙色桶的出现，第三个时间片发现了还有更多橙色桶，等等。我们希望 DBN 能反映这种不断增长的对道路建设的察觉，同时减少对发生事故的估计。

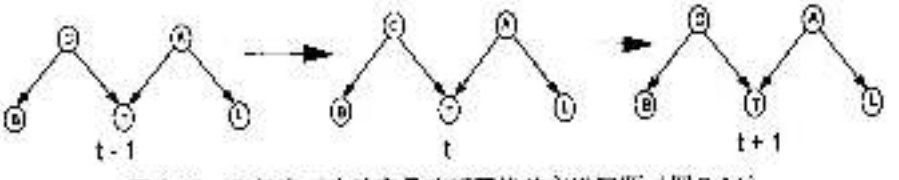


图 9-18　重新表示为动态贝叶斯网络的交通问题（图 9-14）

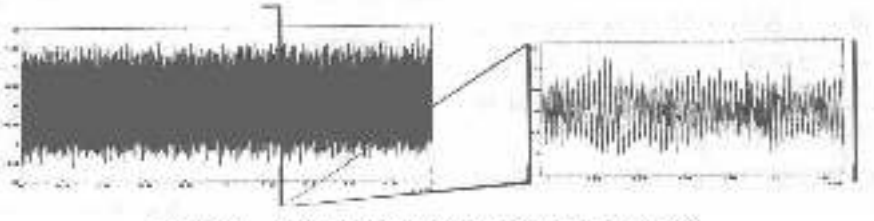


图 9-19　动态贝叶斯网络分析的典型时间序列数据

图9-19是一个来自复杂环境的信号的采样数据。(实际是直升机旋翼系统中热量、振动和其他多种信号的乘积。)这种情况下DBN必须测量的是，这些信号总体上如何改变以及如何为控制整个系统的“运行健康”提供策略。在13.2节中，会进一步讨论这个例子，并更为精确地定义DBN。

DBN最常用来跟踪两类随时间变化的系统。一种是图9-18和图9-19中描述的时变序列，另一种常见于随时间的自然现象，比如自然语言分析应用中的音素或单词抽样。注意，在图模型的有向或无向边上，表示同一时间片内不同随机变量之间的相互关系（瞬时相互关系）是可能的。然而图中反映时序数据的元素之间的相互关系一定要由有向图采集。

总结：如果连接一组表示时间相关数据的图模型的每一条边都是针对时间维的，那么这个模型称为动态贝叶斯网络。这里的DBN也可以用于捕获非时序的数据，比如语言的处理，新的信息反映了状态的变化。其他关于DBN的细节请参考Friedman（1998）或Ghahramani和Jordan（1997）以及13.2节。

9.3.5 马尔可夫模型：离散马尔可夫过程

在3.1.2节，我们介绍了*有限状态自动机*的图形表示，其状态根据输入流的内容转变。状态及其转换反映了形式语言的属性。下面我们介绍*有限状态接受器*（finite state acceptor）或者叫*摩尔机*（Moore machine），能够“识别”具有不同属性的字符串的状态自动机。在5.3节，我们介绍了*概率有限状态自动机*（probabilistic finite state machine），状态自动机的下一个状态函数表示为对当前状态的概率分布。*离散马尔可夫过程*（discrete Markov process）就是这种方法的特例，系统忽略其输入值。

图9-20是一个有四个不同状态的*马尔可夫状态机*（Markov state machine），有时也称为*马尔可夫链*（Markov chain）。系统的状态可以描述为在任何时间段都处于状态集合S中的状态之一，s_1，s_2，s_3，…，s_n。系统在规则间隔的离散的时间点进行状态变化，并且有可能保持同一状态不变。我们将时间的有序集合记为T，集合中的元素表示离散时间点，记为t_1，t_2，t_3，…，t_t。系统根据与每个状态有关的概率分布改变状态。我们将自动机在时刻t所处的实际状态记为σ_t。

通常情况下，这个系统的完整概率描述需要用前驱状态的形式定义当前状态σ_t。这样，系统在任何特定状态σ_t的概率就是：

$$p(\sigma_t) = p(\sigma_t \mid \sigma_{t-1}, \sigma_{t-2}, \sigma_{t-3}, \cdots)$$

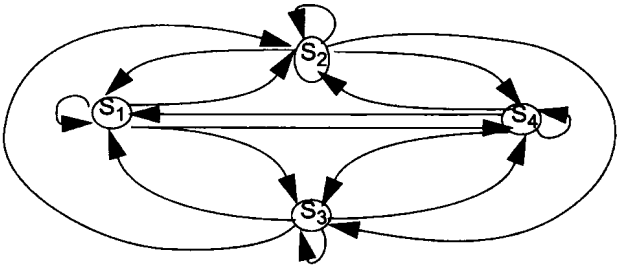


图9-20 有s_1，s_2，…，s_4四个状态的马尔可夫状态机或者说马尔可夫链

其中σ_{t-i}表示σ_t的前驱状态。在*一阶马尔可夫链*（first-order Markov chain）中，当前状态的概率是仅与直接前驱状态有关的函数：

$$p(\sigma_t) = p(\sigma_t \mid \sigma_{t-1})$$

其中 σ_{t-1} 是 σ_t 的前驱。（通常，n 阶马尔可夫链中的状态依赖于它的前 n 个状态。）我们也假设这个等式的右边是时间不变的，也就是说，假设在系统的所有时间范围内，特定状态之间的转换保持相同的概率关系。基于这个假设，现在我们能够建立任意两个状态 s_i 和 s_j 之间的转移概率 a_{ij}：

$$a_{ij} = p(\sigma_t = s_i \mid \sigma_{t-1} = \sigma_j), 1 \leqslant i, j \leqslant N$$

记着 i 有可能等于 j，这种情况下系统保持同一个状态不变。对这些概率分布的一般性约束也要满足；对每个状态 s_i：

$$a_{ij} \geqslant 0, \text{并且} \sum_{i=1}^{N} a_{ij} = 1$$

刚才描述的系统称为一阶可观测的马尔可夫模型（first - order observable Markov model），因为系统的输出是每个离散时刻的状态集合，而且系统的每个状态对应一个物理（可观察的）事件。下面定义可观测的马尔可夫模型，然后给出一个例子。

定义（（可观测的）马尔可夫模型）　一个图模型称为**（可观测的）马尔可夫模型**，当图为有向图，并且在某个离散时刻 t，到达一组状态 S 中任一状态 s_t 的概率是 S 中前一时刻对应状态的概率分布的函数。S 中的每个状态 s_t 对应一个物理可观察的情况。

一个可观测的马尔可夫模型是**一阶的**，当其在任一时刻 t 处于状态 s_t 的概率只是与前一时刻 t - 1 所处状态 s_{t-1} 有关的函数，s_t 和 s_{t-1} 都是 S 中可观察的状态。

注意，任何概率分布都具有马尔可夫模型的属性。这一方法的动力来自于一阶假设。作为可观测的一阶马尔可夫模型的一个例子，可以考虑某一特定地点中午时的天气。假设这个地点的天气变量 weather 具有四个不同的离散状态：s_1 = sunny（晴），s_2 = cloudy（多云），s_3 = fog（雾），s_4 = precipitation（降水）。这个马尔可夫模型的时间间隔是连续的每天的中午。我们还假设 weather 的状态之间的转变在长时间内保持恒定（对许多地点不成立!），并且可观测的 weather 状态可以许多天保持不变。这一情况表示在图 9-20 中，由状态转换矩阵 a_{ij} 支持：与 3.1.2 节中的有限状态自动机类似，在这个转换矩阵 a_{ij} 中，第一行表示从 s_1 到每个状态的转换，包括保持相同状态；第二行表示从 s_2 到每个状态的转换，以此类推。注意，在每个状态的转换的概率分布需要满足一定的属性，和为 1（即每一行的和都为 1）。

a_{ij} =

	s_1	s_2	s_3	s_4
s_1	0.4	0.3	0.2	0.1
s_2	0.2	0.3	0.2	0.3
s_3	0.1	0.3	0.3	0.3
s_4	0.2	0.3	0.3	0.2

现在可以向这个模型提问了。假设今天 s_1 是 sunny；那么未来 5 天保持 sunny 的概率是多少？或者问未来 5 天分别是 sunny、sunny、cloudy、cloudy、precipitation 的概率是多少？我们来求解第二个问题。希望用这个模型在第一天 s_1 是 sunny 的情况下确定下面的状态集合的概率：

$$O = s_1, s_1, s_1, s_2, s_2, s_4$$

根据一阶马尔可夫模型，这个可观察状态序列的概率 M 是：

$$
\begin{aligned}
P(O \mid M) &= P(s_1, s_1, s_1, s_2, s_2, s_4 \mid M) \\
&= P(s_1) \times P(s_1 \mid s_1) \times P(s_1 \mid s_1) \times P(s_2 \mid s_1) \times P(s_2 \mid s_2) \times P(s_4 \mid s_2) \\
&= 1 \times a_{11} \times a_{11} \times a_{12} \times a_{22} \times a_{24} \\
&= 1 \times (0.4) \times (0.4) \times (0.3) \times (0.3) \times (0.3) \\
&= 0.00432
\end{aligned}
$$

这个等式是从一阶马尔可夫模型假设得出的，其中，每天天气的状态只是前一天天气的函数，并且我们观察到的事实是今天晴。

我们可以扩展这个例子，已知今天的天气，确定接下来的 t 天天气保持不变的概率，即天气在第 t+1 天发生变化。对于任意天气状态 s_i 和马尔可夫模型 M，我们有观察 O：

$O = \{s_i, s_i, s_i, \cdots, s_i, s_j\}$，其中有 t+1 个 s_i，并且 $s_i \neq s_j$，那么：

$$P(O \mid M) = 1 \times a_{ii}^{t} \times (1 - a_{ii})$$

其中 a_{ii} 是状态 s_i 转换到自己的概率。这个值称为模型 M 的状态 s_i 的 t 时间区间离散概率密度函数（discrete probability density function）。该时间密度函数是马尔可夫模型状态驻留时间的一种表示。基于这个值，给定任意第一个可观察的状态 s_i，我们可以用模型 M 来计算该状态的期望值，或者说期望区间 d_i：

$$d_i = \sum_{d=1}^{n} d(a_{ii})^{(d-1)}(1 - a_{ii})$$

当 n 趋向于无穷时，化为：

$$d_i = \frac{1}{1 - a_{ii}}$$

例如，用此模型算出的连续降水天的期望值是 1/（1 - 0.3），即 1.43。同样，连续晴天的期望值是 1.67。下面我们总结一下各种马尔可夫模型的变形，其中多数细节会在第 13 章中讨论。

9.3.6　马尔可夫模型：变形

此前我们所见到的马尔可夫模型中，每个状态对应于一个可观察的离散物理事件，比如一天中特定时刻的天气值。这类模型实际上有很大局限，现在将其一般化为更为广泛的一类模型。我们简单说明几个方法。13.1 节中会提出更易理解的隐马尔可夫模型表示，以及几个重要的变形。

隐马尔可夫模型

在前面提出的马尔可夫模型中，每个状态对应于一个可观察的离散物理事件。在隐马尔可夫模型（HMM）中，观察到的值假设为当前隐含状态的概率函数。

例如，已观察到的人的语音是说话者所要发出的某个语音的反映。口中的声音仅在概率上和说话者的意图或实际状态相关联。因此，HMM 是双重嵌套的随机过程，其中可观察的随机过程（说话者的声音）由不可观察的随机过程（说话者的状态或意图）支持。如 9.3.2 节所见，HMM 是 DBN 的一个特例。其他细节请参见 13.1 节、13.2 节和 Rabiner（1989）。

半马尔可夫模型

半马尔可夫模型（semi-Markov model）是二分量（two-component）马尔可夫链。第一个分量负责下一个状态转换（next state transition）的选择，第二个分量负责转换时间（transition time）。当半马尔可夫模型进入 s_i 状态并在 t_i 时间保持不变，其时间由状态驻留时间的概率分布确定。

一旦过了 t_i 时间，状态就根据状态转换概率分布转移到下一个状态 s_{i+1}，另一个状态转换时间 t_{i+1} 再次被选中。与传统马尔可夫模型特定的固定转移时间相比，半马尔可夫模型支持任意的状态驻留概率分布。

马尔可夫决策过程

另外两种广泛用于强化学习中的马尔可夫模型分别是：马尔可夫决策过程（Markov decision process，MDP）和部分可观测的马尔可夫决策过程（partially observable Markov decision process，POMDP）。MDP 是定义在两种空间上的：所研究问题的状态空间和可能的动作空间。状态空间中到新状态的转换依赖于当前的状态和当前的动作，并受条件概率分布的影响。在每个状态上都有根据当前状态和动作计算出的奖赏值。典型的，强化学习的目的就是在当前条件下最大化奖赏值的累积函数。

当 MDP 在使用基于概率的转换矩阵之前使用已观测到的（已确定的）状态选择下一个状态时，POMDP 仅有关于当前状态的概率知识（和用于决定下一个状态的概率矩阵）。因此 POMDP 可以通过计算为复杂环境验证难解性。我们将在 10.7 节和 13.3 节进一步学习 MDP 和 POMDP，以及强化学习。

9.3.7　BBN 概率建模的一阶替代方案

到此为止，9.3 节中已经提出了不确定推理基于命题演算的表示。当然要问在什么意义上基于概率谓词逻辑的模型可以用于推理呢？全谓词逻辑表示的优点是说明性语义和基于变量表示的框架。传统基于命题逻辑方法的典型局限包括处理噪声和不确定性的有限能力以及表示的静态观点。

因此 9.3 节中提出的贝叶斯和马尔可夫模型局限在命题级的表示上，在表示某种分布上一般的（基于变量的）关系上会显得笨重，例如，$\forall X\ \text{male}(X) \rightarrow \text{smart}(X)$ (0.49，0.51)。很多负责问题域需要这种表达式级别。

能够表达重复、递归和潜在的无穷结构，对于复杂域尤其是诊断与预测领域也很重要。所以，如果想表示状态随时间变化的系统必须要有具代表性的能力。这需要一个递归的或“循环直至完成”式的控制体系。如果想要动态重建传统的 BBN，必须被迫重建全部时间片上的整个网络，因为这些结构缺少说明性的和时间导向的语义。

近来的研究活动已经为图模型创建了多种重要的扩展，包括许多一阶（基于变量且递归的）系统。也有许多多级可分解贝叶斯模型。在这些新的建模形式方法中，我们列出几个支持这种表示的方法。第 13 章中讲述机器学习的概率方法时，给出了几种这些表示的例子。

在知识库中建立贝叶斯网络（Haddawy 1994，Ngo and Haddawy 1997，Ngo et al. 1997）

Haddawy、Ngo 及其同事使用贝叶斯信念网络合并了一个一阶逻辑系统。结果他们创建了一种一阶概率逻辑，用作表示语言以知识库的形式描述一类特定的网络。（贝叶斯）知识库的形式语义通过库中严格定义的语句来起作用。Haddawy 和 Ngo 提出了一种正确性可证的贝叶斯网络生成算法，用于实现潜在逻辑型语言的形式语义与知识库的独立语义相结合。这种生成算法的一个特点是避免通过用户指定的建议在网络中产生不相关结点。

Haddawy 和 Ngo 认为当知识库变大时推理的效率会变低。他们提出使用上下文信息为知识库的概率句子索引的方法来纠正。当模型生成算法创建网络时，会忽略知识库中和当前任务不相关的句子。生成的模型会比使用了整个知识库创建的模型显著减小。由 Haddawy、Ngo 及其同事们提出的这种方法是随机逻辑建模领域的研究之一，这种建模方法明确地使用有关域中的上下文信息，并以此集中关心相关的信息，降低用于解决特定推理任务模型的大小。

贝叶斯逻辑程序（Kersting and DeRaedt 2000）

贝叶斯逻辑程序（Bayesian logic program，BLP）提供了一种代表性的框架，包括逻辑程序设计和图模型。具体地，BLP 结合了贝叶斯信念网络和霍恩子句逻辑（Prolog，见 14.3 节）。如果给定具体查询，系统使用带有不确定参数的一阶规则集，产生贝叶斯网络来回答查询。结果表示易于解释，设计者为模型指定的理论语义与 2.3 节的方法相似。

概率关系模型（Friedman et al. 1999，Getoor et al. 2001）

Friedman、Getoor 与其同事研究了另一种一阶概率系统的方法——概率关系模型（probabilistic relational model，PRM）。PRM 没有创建说明性逻辑类语言，而是在对象类别上建立了概率模型。PRM 在类级别上定义概率约束，这样这些约束可用于同类的任意对象。使用 PRM 可以描述关于类别属性之间的概率依赖性。许多 PRM 的扩展允许支持不同细节级别概率依赖的子类（Getoor et al. 2001）。类别层次结构可以在个体级别（传统贝叶斯信念网络）和类级别上建模。PRM 更进一步的扩展，除了可以处理固定的模型关系结构，还可以处理不确定的模型结构。从这方面看，PRM 和关系数据库有紧密的联系，比如它们之间相似的结构。

马尔可夫逻辑网（Richardson and Domingos 2006）

Richardson 和 Domingos 提出了另一种基于逻辑的概率系统，称为马尔可夫逻辑网（Markov logic network，MLN）。多数先前的基于逻辑的随机建模方法都通过最普通的霍恩子句表示来使用普通逻辑的有限子集。为了消除逻辑的局限，Richardson 和 Domingos 使用普通一阶逻辑，其中的语句被转换为合取范式（conjunctive normal form，CNF）。他们也使用马尔可夫随机场（Pearl 1988）作为逻辑的概率相对部分，主要区别于前面描述的系统（贝叶斯信念网络是目前使用最多的表示方法）。所以，从 CNF 中逻辑句子到马尔可夫随机场的映射是直接的。MLN 是第一个带有完整映射的理论框架，该映射是从具有函数符号的一阶谓词演算到概率分布的映射。

多环逻辑（Pless et al. 2006，Chakrabarti et al. 2006，Sakhanenko et al. 2006）

多环逻辑（loopy logic）是一阶图灵完备的说明性概率语言。其名字来源于所使用的多环信念传播算法（loopy belief propagation algorithm，Pearl 1998）。其基于逻辑的说明性表示第一次翻译到马尔可夫随机场中。具有期望最大化（expectation maximization，EM）算法的参数学习与多环信念传播结合良好。我们将在 13.2 节中演示多环逻辑系统，以及它在马尔可夫随机场和基于 EM 的参数学习中的使用。

对传统 BBN 在一阶图灵完备方面的进一步扩展有很多种。一阶统计建模的进一步例子请参考以下内容：Pffefer（2001）的 IBAL 语言，Pless 和 Luger（2002）的随机 Lambda 演算。面向对象随机表示请参考：Koller 和 Pfeffer 1997；Pfeffer et al. 1999；Xiang et al. 2000；Pless et al. 2000。随机方法在 AI 领域中非常重要，例如用概率主体进行问题求解（Kosoresow 1993）。我们会在 13 章中看到随机方法在机器学习上的应用，在 15.4 节看到其在自然语言处理上的应用。

9.4 结语和参考文献

从人工智能的研究一开始，就有一部分学者认为逻辑和它的扩展足以用来表示和刻画智能。人们提出了很多一阶谓词演算之外的办法来描述不确定条件下的推理：

1）**多值逻辑**。它采用增加新的真值变量的方法来扩展逻辑，比如在 true 和 false 之外增加 unknown。它可以用来区分断言，例如已知为假和那些不能简单认为为真的断言。

2）**模态逻辑**。它增加操作符来处理知识与信念、必要性与可能性的问题。在本章中介绍了模态操作符 unless 和 is consistent with。

3）**时序逻辑**。它使我们可以量化与时间有关的表达式，比如表达式一直为真或者将来某个时间为真。

4）**高阶逻辑**。很多类知识涉及高阶的概念，它们的谓词不仅仅是变量，也可以被量化。我们真的需要高阶逻辑来处理这些知识吗，或者说只用一阶逻辑可以完成吗？如果要用到高阶逻辑，最好用多少阶呢？

5）**定义、原型和例外的逻辑表示**。例外常被看成是一个可定义系统的必要特征。但是，对例外的不慎使用会破坏表示的语义。另外一个问题是定义和原型之间的区别，原型即个体的典型代表。什么才是一个类的属性和一个典型成员的属性之间的明确差别呢？典型个体怎么来表示呢？何时原型比定义更为适合呢？

基于逻辑的表示仍然是一个重要的研究领域（McCarthy 1968，Hayes 1979，Weyhrauch 1980，Moore 1982，Turner 1984）。

还有很多真值维护系统（TMS）推理方面的成果。基于逻辑的 TMS 建立在 McAllester（1978）工作的基础之上。在 LTMS 中，命题之间的关系是用子句来表示的，这些子句用来推出它们所描述的命题的真值。另外一种办法——多信念推理程序 MBR 与 ATMS 推理程序差不多（deKleer 1984）；类似的观点可以在 Martins 和 Shapiro（1983）中看到。MBR 基于称为 SWM * 的逻辑语言，这种语言用来描述知识的状态。用来检查知识库推理中不一致性的算法可以在 Ginsburg（1997）和 Martins（1990，1991）中找到。想了解更多支持 JTMS 的算法请参阅 Doyle（1983）或者 Reinfrank（1989）。默认逻辑允许系统扩展推出的任何定理作为更进一步推理的公理。Reiter 与 Criscuolo（1981）和 Touretzky（1986）发展了这些问题。

除了该领域最初的论文（Doyle 1979；Reiter 1985；deKleer 1986；McCarthy 1977，1980）外，还有丰富的文献是关于非单调推理、信念逻辑和真值维护的。对于随机模型请参阅 Pearl（1988）的《Probabilistic Reasoning in Intelligent Systems》、Shafer 和 Pearl（1990）的《Readings in Uncertain Reasoning》、Davis（1990）的《Representations of Commonsense Knowledge》，还有近期 AAAI、UAI 和 IJCAI 的（会议录中的）大量文章。我们推荐 Stuart Shapiro 写的《The Encyclopedia of Artificial Intelligence》（1992 年第 2 版），这本书涵盖了本章中的很多推理模型。Josephson 和 Josephson（1994）出版了在反绎推理方面的论文集：《Abductive Inference：Computation，Philosophy，and Technology》。还可以看《Formal Theories of the Commonsense World》（Hobbs and Moore 1985）。在因果推理方面，Pearl（2000）对理解世界上的因果关系的概念做出了贡献。

对限定逻辑和最小模型逻辑更进一步的研究可以查阅 Genesereth 与 Nilsson（1987）、Lifschitz（1986）和 McCarthy（1986）。限定逻辑的另外一个贡献是 Perlis（1988）关于智能主体在缺少知识情况下的推理。Ginsburg（1987）编辑了有关非单调系统的重要论文集：《Readings in Nonmonotonic Reasoning》。

模糊系统方面的更深入的读物，我们推荐 Lotfi Zadeh（1983）的最初论文，这种技术的新的集成可参阅 Yager 和 Zadeh（1994）的《Fuzzy Sets，Neural Networks and Soft Computing》以及 Timothy Ross（1995）的《Fuzzy Logic with Engineering Applications》。9.2.2 节中给出的倒立摆问题就是从 Ross 中改编来的。

贝叶斯信念网络推理的算法包括 Pearl（1988）的消息传递方法和 Lauritzen 与 Spiegelhalter（1988）的团树三角测量（见 9.3 节）。对这些算法的进一步讨论参见《Encyclopedia of AI》（Shapiro 1992）和（Huang 和 Darwiche 1996）。AISB Quarterly 的 1996 年春季的合集包含了对贝叶斯信念网络的介绍（van der Gaag 1996）；另外推荐 Wellman（1990）和 Druzdel（1996）的对量化概率网络的讨论。

随机表示和算法一直是非常活跃的研究领域（Xiang et al. 1993，Laskey and Mahoney 1997）。贝叶斯表示的局限已经引起了人们对分层和压缩贝叶斯模型的研究（Koller and Pfeffer 1997，1998；Pfeffer et al. 1999；Xiang et al. 2000）。对于我们介绍的基于命题的图形模型的更多扩展可以在相关文献中找到。我们推荐阅读 Koller 和 Pfeffer（1998）及 Pless 等（2000）的著作来了解面向对象的随机表示思想。Pfeffer（2000）的 IBAL 语言、Pless 与 Luger（2002）的《Stochastic Lambda Calculus》是一阶随机函数语言的例子。Cussens（2001）以及 Pless 和 Luger（2003）提出了随机推理的基于逻辑的一阶（说明性）语言。还有 9.3.7 节中给出的参考文献，这里并没有重复列出。

从 9.3 节之后，尤其在第四部分第 13 章中，我们又会看到很多关于机器学习概率方法的表示问题。

9.5 习题

1. 确定三个应用领域，在这些应用领域中不确定条件下的推理是必需的。选择其中的一个领域，设计反映该领域推理的 6 个推理规则。
2. 给定一个“反向链”专家系统的下列规则：

   $A \wedge \text{not}(B) \Rightarrow C(0.9)$

   $C \vee D \Rightarrow E(0.75)$

   $F \Rightarrow A(0.6)$

   $G \Rightarrow D(0.8)$

   系统可以包含下列事实（带有确信度）：

   $F(0.9)$

   $B(-0.8)$

   $G(0.7)$

   运用 Stanford 确信度代数来决定 E 和它的确信度。
3. 考虑类似 MYCIN 的规则：if $A \wedge (B \vee C) \Rightarrow D(0.9) \wedge E(0.75)$。讨论一下用贝叶斯方法处理这些不确定性时出现的问题。在 Dempster-Shafer 推理中，这条规则会怎么处理呢？
4. 创建一个医疗诊断的新例子，并用 9.2.3 节中的 Dempster – Shafer 等式来获取如表 9-1 和表 9-2 的信念分布。
5. 运用在 McCarthy（1980，第 4 节）中给出的表示模式公理来产生 9.1.3 节中给出的限定结果。
6. 创建另外一个类似于图 9-4 的推理网络，并给出它的前提的相关格，如图 9-5。
7. 用最小模型的假设来推理在人们的日常生活中非常重要。给出假定最小模型的两个以上的例子。
8. 继续看 9.2.2 节中倒立摆的例子。让模糊控制器进行多于两次的迭代，其中每次迭代的输出作为下次迭代的输入。
9. 写一个程序实现 9.2.2 节中的模糊控制器。
10. 找文献，例如 Ross（1995），说出模糊控制适用的另外两个领域。为这些领域建立一套模糊规则。
11. 汤姆将自己的积蓄投资到 5 种股票的投资组合，H = {Bank of China，Citibank，Intel，Nokia，Legend}。这些股票可以分为*银行股票*（Bank of China 和 Citibank），*高科技股票*（Intel、Nokia 和 Legend）和*中国股票*（Bank of China 和 Legend）。你被任命建立一个专家系统，用以给他管理投资组合的建议。为此，你需要咨询很多金融顾问并为专家系统得到一组规则集，这些规则告诉汤姆在投资组合中如何分配。分配应直接与股价上涨的可能性或股票类型成比例。

    规则 1：如果利率上升，那么分别以 0.8、0.15 和 0.05 比例买入银行股票、高科技股票和 Nokia。

    规则 2：如果就业率上升，那么分别以 0.5、0.2 和 0.3 比例买入 Bank of China、Intel 和高科技股票。

    规则 3：如果通货膨胀率下降，那么分别以 0.1、0.4 和 0.5 比例买入 Intel、Citibank 和高科技股票。

大卫在注意到利率刚刚上升时，使用 Dempster－Shafer 理论，计算了三类股票（即银行股票、中国股票和高科技股票）以及 Nokia 的置信区间。那么，在 Nokia 和高科技股票之间，信念和置信区间是否有什么不同？原因是什么？（这个问题由香港理工大学的 James Liu Nga Kwok 提出，指导手册中的解由学生 Qi Xianming 给出。）

12. 在图 9-16 中加入一个连接，即把季节同滑的人行道直接相连，然后建立团树来表示这种情形。同图 9-17 比较一下复杂性。
13. 完成表 9-3 未完成的符号的估计。
14. 请给出贝叶斯信念传播的算法，并把它用到 9.3.2 节中滑的人行道问题中去。你可以用 Pearl（1988）的消息传递方法或者 Lauritzen 与 Spiegelhalter（1988）提出的团树三角测量的方法。
15. 建立对另外一个应用的贝叶斯信念网络，比如医疗诊断、地质勘测或者汽车故障诊断。指出 d－可分的例子，并建立这个网络的团树。
16. 建立以下情形（见图 9-21）的团树和连接树。掠夺、故意破坏和地震都能引起房屋的报警。这还可以度量出潜在危险可能发生在该房屋的邻居中。
17. 对于 9.2.3 节中 Dempster-Shafer 模型描述的表 9-1 和表 9-2 的诊断推理情形，用贝叶斯信念网络来表示。比较和对比这两种诊断的途径。
18. 假设你想设计一个二阶马尔可夫模型，即每个可观察状态依赖于前面的两个可观察状态。你应该如何设计？转换概率矩阵应该是什么样的？
19. 给定 9.3.4 节中的 weather 的可观察马尔可夫模型：
    a）计算未来 5 天是晴天的概率。
    b）恰好 3 天晴天然后 1 天降水然后又 1 天晴天的概率。

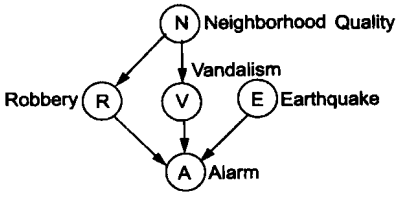


图 9-21　一个信念网络表示房屋警报的可能性出在危险的邻居中

第四部分 机器学习

逻辑不是智能的结束，而是开始……

——Spock，《星际迷航 VI》

"我知道你在想什么，"叮当说道，"但是那不是真的，嘿，不是的。"

"正相反，"叮当答辩道，"如果那是真的，那就可能是真的；如果那曾经是真的，它就是真的过；但是既然现在它不是真的，那么现在它就是假的。这是逻辑。"

——Lewis Carroll，《镜中世界》（1871）

让我们来猜想……

——乔治·波利亚

基于符号的、连接的、遗传的和随机的机器学习方法

当人们被问及什么智能技能被是人类具有的最本质的又是计算机最难以实现的这个问题时，除了艺术创造力、伦理判断能力和社会责任感之外，人们通常会想到语言和学习。这些年来，这两个领域被当作人工智能的目标、挑战和人工智能进展的试金石。语言和学习是很难也是很重要的研究领域的一个原因是它们包含了许多其他的人类智能功能。如果我们想说我们已经创造了人工智能，我们就必须要解决自然语言理解、自动推理和机器学习的问题。在第四部分讨论机器学习的一些办法，在第五部分讨论自动推理和自然语言理解。

在第 10 章，讨论基于符号的机器学习，它基于代表问题域中实体和关系的符号集合。符号学习算法试图用这些符号来推出新颖的、有效的和有用的也是用这些符号来表达的一般规则。

第 11 章讨论的连接的办法把知识表示为由小的单个的处理单元组成的网络的激活或者抑制状态模式。受动物大脑体系结构的启发，连接网络的学习是通过训练数据来修改网络结构和连接权值的办法实现的。连接不是通过搜索符号语言表示的规则而进行学习的，它在数据中发现不变的模式，并用网络自身的结构进行表示。

正像连接网络受生物神经系统的启发一样，第 12 章的进化模型是受遗传和进化的启发。遗传算法开始时有一组问题的候选解，候选解根据它们解决问题的能力来进化：只有最适者生存，并相互组合来产生下一代的解。这样，解不断地增强，就像达尔文所描述的现实世界的进化。在探求生命起源的过程中去寻找智能的起源，这二者之间相当得奇怪。

最后，在第 13 章中，我们给出了机器学习的随机方法。随机学习的基础是对贝叶斯规则的理解：在某领域的经验制约着理解者对世（或其他）领域新数据解释时的期望。这里我们采用的主要方法是给出和讨论各种类型的马尔可夫过程，并用这些模型演示诊断推理数据和语言的机器理解领域中的几个数据集的解释。我们也给出了强化学习，即智能体在问题求解过程中得到其他问题领域的反馈时，如何做出反应的学习方法。

机器学习研究中存在着很多重要的哲学和认识论的问题。首先有泛化的问题，或者说是机器在数据中发现的模式怎样才能有良好的推广能力，比如，模式是否适合新的数据。其次是机器学习中归纳偏置的问题。这个偏置指的就是程序设计者是怎么运用直觉和启发式信息来设计支

持学习过程的表示、模型和算法的。偏置的例子有对问题域中的“重要”概念的识别，“特殊搜索或者激励算法”的使用，或者一种特定的神经网络体系结构的选择。问题是这些归纳偏置使我们能够学习，但同时也限制了我们学习的内容。最后一个哲学问题称为“经验主义者的困境”，让我们看一下以下两种情况：如果学习系统中没有先验的偏置，怎么才能学习到有用的知识，或者，我们怎么知道已经学到了知识呢？这个问题经常在无监督和涌现学习模式中出现。这3个问题将在16.2节中讨论。

第 10 章　基于符号的机器学习

像我已经讲过的那样，意识是由大量简单的感性认识提供而成，感性认识通过感官接受进来，如从外部事物中发现，或从对其自身动作的反射，还要注意，有一些这样简单的感官认识会经常结合起来……一不留神，我们会很容易地把它们直接说出来并作为一种单一思想进行考虑。

——约翰·洛克，《人类理解论》

纯粹观察一件事情是没有用处的。观察变为注视，注视变为思考，思考变为建立联系，这样可以说我们每个留心的匆匆一瞥都是一个理论化的行为。但是，这应该有意识地完成，并伴随着自我批评，伴随着自由，可以用大胆的话，可以讽刺。

——歌德

10.0　简介

任何宣称具有一般智能的系统必须具备学习的能力。实际上，在我们的符号和解释系统中，不变的智慧这个概念似乎是矛盾的。智能主体必须能够在与环境交互的过程和处理自己内部状态和步骤的过程中进行改变。我们将用 4 章来讨论机器学习的问题，从而反映解决这一问题的四个途径：第一是基于符号的方法，第二是连接的方法，第三是遗传或进化的方法，第四是动态或随机的方法。

学习对于人工智能的实际应用是很重要的。Feigenbaum 和 McCorduck（1983）把妨碍智能系统广泛应用的主要障碍称为“知识工程的瓶颈”。这个“瓶颈”是指用 7.1 节中的传统知识获取技术来建造专家系统的代价和困难。解决这个问题的一种办法是程序开始时装有最少的知识，然后从实例、高水平指导信息或者系统自身的探索中进行学习。

Herbert Simon 对学习定义如下：

> 能够让系统在执行同一任务或相同规模的另外一个任务时比前一次执行得更好的任何改变（Simon，1983）。

这个定义虽然简洁，却指出了设计学习器要注意的问题。学习包括对经验的泛化：不仅是重复同一任务，而且是域中相似的任务都要执行得更好。因为感兴趣的领域可能很大，学习器通常只研究所有可能例子中的一小部分；从有限的经验中，学习器必须能够正确泛化域中未见的数据。这是个*归纳*的问题，也是学习的中心问题。对于大多数的学习问题，不管用哪种算法，能用的数据都不足以保证最优的泛化。学习器必须启发式地泛化，也就是说，它们必须选取经验中对未来更为有效的部分。这样的选择标准就是*归纳偏置*。

Simon 的定义把学习描述为让系统“第二次执行得更好”。像前一段中所表明的那样，选择能让系统改进的可能改变是很难的。对学习的研究必须解决一些改变会降低系统性能的问题。防止和检测这些情况是学习算法的另外一个问题。

学习包括了学习器的改变，这是很明显的。但是，这些改变的本质和表示它们的最佳方式就远不是那么明显的事情了。一种办法把学习看成是明确表示的领域知识的获取。学习器基于它的经验用形式语言（例如逻辑）来创建或者修改表达式，并把知识保留下来以供将来使用。*基于符号的方法*，以第 10 章中的算法为代表，是建立在这样的假设基础之上的：对程序行为的最初影响是明确表示的领域知识的集合。

相反，第11章（神经网络或者连接网络）并不用符号化的语言获取知识的办法来学习。像由大量相连的神经细胞组成的动物大脑一样，神经网络是相连的人工神经元组成的系统。系统的知识隐含在这些神经元的组织和相互作用中。神经网络不是通过向知识库中增加表示来学习，而是通过修改它的结构来学习，以适应学习领域中的突然性。第12章讨论遗传和进化学习。我们的一个最强有力的学习模型可以在人类和动物的系统中看到，这一系统在朝着生态平衡的方向进化。这种通过适应来学习的方法反映在遗传算法、遗传程序设计和人工生命的研究中。

最后在第13章中，我们提出学习的动态方法和概率方法。很多复杂的或没有被完全理解的情况，比如人类语言的产生和理解，使用概率工具“理解”最好。类似地，也可以用来诊断动态系统，比如运行着的机械引擎中的故障。

第10章我们开始探索基于符号的机器学习方法，这些算法随多种情况变化，比如目标、可用的训练数据、学习策略和所用的知识表示语言等。但是，所有这些算法都是这样学习的：对可能的概念空间进行搜索来寻找可接受的泛化。在10.1节中，给出基于符号机器学习的大体框架，主要说明这种学习方法隐含的通用假设。

虽然10.1节给出了很多学习任务的大体框架，但这一章主要讨论的是归纳学习。归纳是最基础的学习任务之一，是从实例集合中学习一般通用的知识。概念学习是一种典型的归纳学习问题：给定某个概念的实例，比如“猫”、“大豆的病虫害”或“好的股票投资”，我们想要推出一个定义，能让学习器正确地识别这些概念的未来的实例。10.2节和10.3节讨论用于概念归纳的两种算法，变形空间搜索算法和ID3算法。

10.4节讨论学习中归纳偏置的角色。学习中涉及的搜索空间会变得极大，即使是用基于搜索的问题求解的标准。这些问题的复杂性会因为在不同训练数据支持的泛化规则之中进行的选择而加剧。归纳偏置是指学习器中用来约束泛化空间的任何方法。

10.2节和10.3节中的算法是数据驱动的。它们没有用领域中的先验知识，而是依赖于大量的例子来定义一个一般概念的必要属性。基于训练数据的模式来泛化的算法称为基于相似性的学习算法。

与基于相似性的学习相反，学习器可以运用领域中的先验知识来指导泛化。例如，人就不需要大量例子来有效地学习。通常，一个例子、类比或者高水平的指导就足以获取一般的概念。有效地运用这些知识可以帮助主体更加有效地学习，减少错误的可能性。10.5节讨论基于解释的学习，它是通过类比和其他运用先验知识从有限的训练数据中学习的技术来进行学习的。

10.2节到10.5节中给出的算法虽然在搜索策略、表示语言和用到的先验知识的数量上各不相同，但它们都假定训练数据已经由教师或其他方式进行了分类。学习器要被告知一个目标概念的实例是正例还是反例。这种依赖于已经分类好的数据的学习称为监督学习。

10.6节继续讨论称为无监督学习的归纳学习，它解决了智能主体对没有正确分类的训练数据获取有用知识的问题。分类形成（或概念聚类）是无监督学习的基础问题。给定具有不同属性的对象集，主体怎么才能对对象进行有效的分类呢？我们又怎么知道这个分类是否有效呢？在这一节中，讨论一下这两种分类形成算法：CLUSTER/2和COBWEB。

最后，在10.7节中，给出强化学习。这里，主体处在环境之中，并从环境中得到反馈。这里的学习需要主体产生动作，然后对从这些动作得来的反馈进行解释。强化学习与监督学习不同的地方在于它没有“教师”对每个动作直接响应，而是主体自身必须创建一种策略来解释所有的反馈。强化学习很好地符合了构造论者的认识论，可以参见16.2节。

本章所讨论的学习有一个共同点：它们都可以看成是一种状态空间的搜索。即使是强化学习也是从状态空间中获得估价函数。我们接下来给出机器学习中基于搜索的大体框架。

10.1　基于符号学习的框架

基于符号的学习算法可以从几个维度进行表征，如图 10-1 所示。

1）**学习任务的数据和目标**。我们表征学习算法的一个主要方式就是看学习器的目标和给定的数据。例如，10.2 节和 10.3 节中的概念学习算法，初始状态是目标类的一组正例（通常也有反例），学习的目标是得出一个通用的定义，它能够让学习器辨识该类的未来的实例。与这些算法采用大量数据的方法相反，基于解释的学习（见 10.5 节）试图从单一的训练实例和预先给定的特定领域的知识库中推出一个泛化的概念。10.6 节中讨论的概念聚类算法阐释了归纳问题的另外一种情况：这些算法的初始状态是未分类的实例集合，而不是从已经分好类的实例集合进行学习。它们的学习任务是发现对于学习器有用的分类信息。

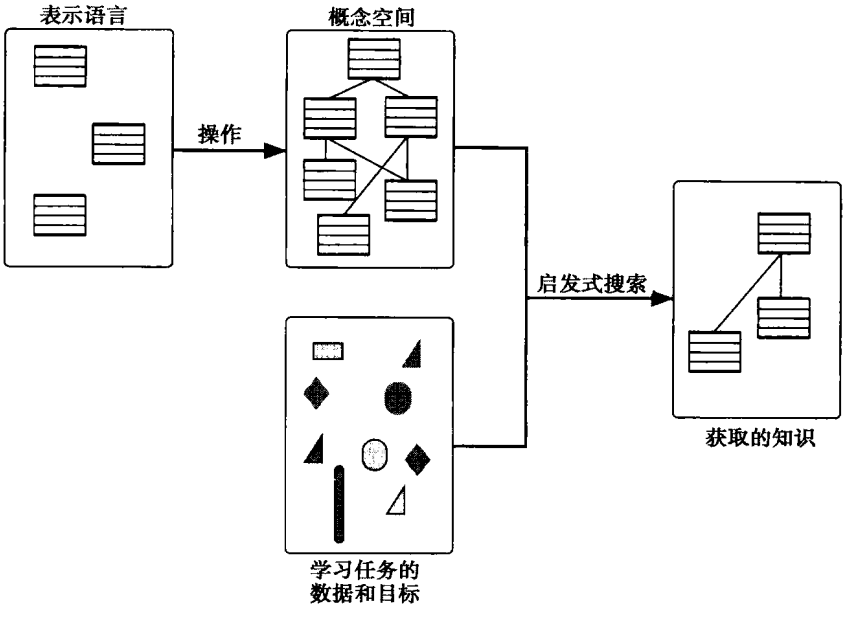


图 10-1　学习过程的总体模型

实例并不是训练数据的惟一来源。例如，人们经常从高水平的指导中学习。教授在教程序设计时，通常会告诉学生所有的循环必须达到终止条件。这个指导虽然正确，却不是直接有用的：它必须转化为在程序设计语言中操作循环变量或逻辑条件的特定规则。类比（见 10.5.4 节）是另外一种训练数据，在使用它们之前必须对其加以正确的解释。如果教师告诉学生电流像水，学生必须得出类比的正确内容：像水在管道中流动一样，电流在电线中流动。像水流一样，我们可以测量电流的数量（安培）和使电流流动的压力（伏特）。但是，与水不同的是，电流不会把东西弄湿或者帮助我们洗手。类比的解释包括发现有意义的相似之处，并避免错误的或者无意义的推理。

我们还可以通过学习器的目标或者靶（target）来表征一个学习算法。许多学习算法的目标是一个概念，或者物体的类的通用描述。学习算法还可以获取计划，问题求解的启发式信息，或者其他形式的过程性知识。

训练数据本身的属性和质量是我们划分学习任务的另外一个尺度。数据可能来自外界环境中的教师，也可能是程序自身产生的。数据也许是可信赖的，也可能包含噪声。这些数据可能是以较好结构的形式给出来的，也可能是未经组织的。它可能既包括正例也包括反例，或只包括正

例。数据可能已经可用，也可能程序不得不建立试验，或者进行其他形式的知识获取。

2）**所学知识的表示**。机器学习程序已经利用了本书讨论的所有知识表示语言。例如，对物体分类的学习程序可能把这些概念表示为谓词演算的表达式，或者它们可能用结构化的表示，如框架或对象。计划可以用操作的序列来描述，或者用三角表来描述。启发式信息可以用问题求解规则来表示。

概念学习问题的一个简单表述把概念的实例表示为包含变量的合取语句。例如，两个“球”的实例（但是不足以学习概念）可以表示如下：

size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, round)
size(obj2, large) ∧ color(obj2, red) ∧ shape(obj2, round)

“球”的泛化概念可以定义为：

size(X, Y) ∧ color(X, Z) ∧ shape(X, round)

符合这个通用定义的任何语句都表示一个球。

3）**操作的集合**。给定训练实例集，学习器必须建立满足目标的泛化、启发式规则或者计划。这就需要对表示进行操作的能力。典型的操作包括泛化或者特化符号表达式、调整神经网络的权值，或者其他方式的对程序表示的修改。

在刚才的概念学习的例子中，学习器可以通过用变量替换常量的方法来泛化出定义。如果初始概念为：

size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, round)

用变量来替换单个常量，产生出下面的泛化：

size(obj1, X) ∧ color(obj1, red) ∧ shape(obj1, round)
size(obj1, small) ∧ color(obj1, X) ∧ shape(obj1, round)
size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, X)
size(X, small) ∧ color(X, red) ∧ shape(X, round)

4）**概念空间**。上面讨论的表示语言和操作定义了潜在概念定义的空间。学习器必须搜索这个空间来寻找所期望的概念。概念空间的复杂度是学习问题难度的主要度量。

5）**启发式搜索**。学习器必须给出搜索的方向和顺序，并且要利用可用的训练数据和启发式信息来有效地搜索。在我们学习“球”的概念的例子中，算法可以把第一个实例当作候选概念，对它进行泛化，使之能够包含接下来的实例。例如，给定单一的训练实例：

size(obj1, small) ∧ color(obj1, red) ∧ shape(obj1, round)

学习器就会把这个实例当作候选概念，这个概念能对当前仅有的一个正例进行正确的分类。

如果现在给定算法的第二个正例：

size(obj2, large) ∧ color(obj2, red) ∧ shape(obj2, round)

学习器会通过用变量替换常量来泛化候选概念，以使概念能够匹配这两个实例。这个结果比候选概念更加泛化，更接近于我们的目标概念“球”。

size(X, Y) ∧ color(X, red) ∧ shape(X, round)

Patrick Winston 关于从正反例中学习概念的论文（1975a）解释了框架的这几个组成部分。他的程序学习结构化概念的一般定义，如积木世界中的“拱门”。训练数据是一系列概念的正例和反例：属于该类别的积木实例，以及小差别（near miss）的实例。后者大体上属于该类别，只

有一个属性或者关系不同。小差别能够让程序找出那个特征，用以从目标概念中排除反例，图 10-2 给出了概念“拱门”的正例和有小差别的实例。

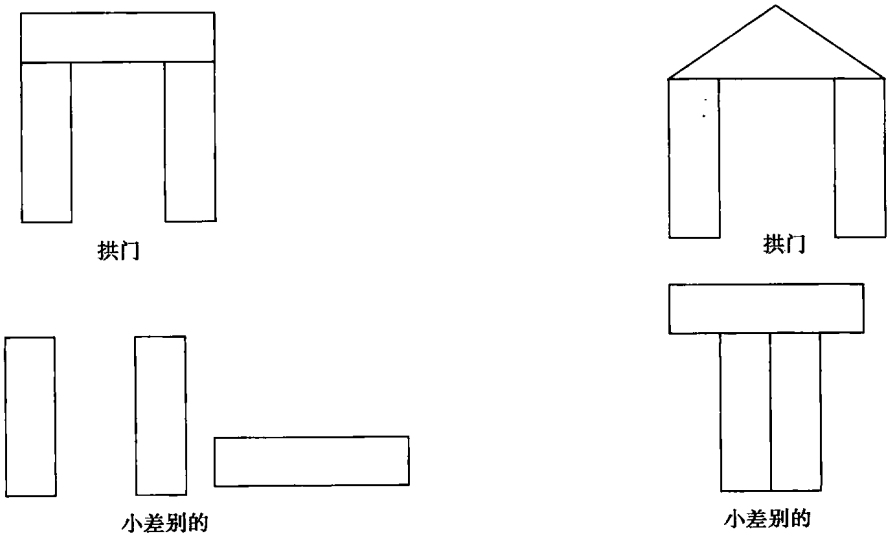
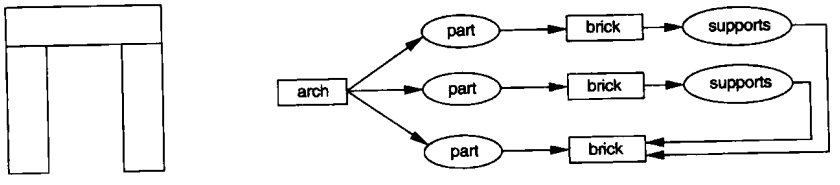
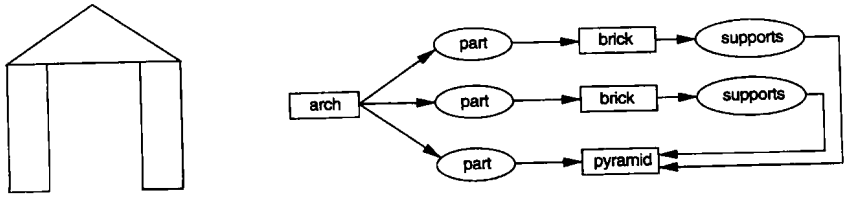


图 10-2　概念“拱门”的实例和小差别的实例

这个程序用语义网来表示概念，如图 10-3 所示。它是通过给定训练实例时提取目标概念的候选描述方式来学习的。Winston 的程序通过泛化和特化来提取候选描述。泛化修改图让它来适应新实例。图 10-3a 给出了由三块砖构成的拱门和描述它的语义网图。下一个训练实例顶端是一个棱锥而不是砖，如图 10-3b 所示。这个实例与候选描述并不匹配。程序对这两个图进行匹配，试图找到它们相同的部分。图匹配程序用结点名称来指导匹配过程。一旦程序对图进行匹配，就可能发现图之间的差异。在图 10-3 中，除了第一张图顶端是砖（brick），而第二张图顶端是棱锥（phramid）外，两张图其余部分都匹配。程序的部分背景知识是这些概念的通用层次，如图 10-3c 所示。程序对图进行泛化，用砖和棱锥的最小公共超类来替换相应结点，在这个例子中，这个超类是多边形（polygon）。结果是如图 10-3d 所示的概念。

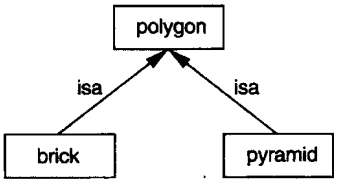


a) 拱门的一个例子和它的网络描述

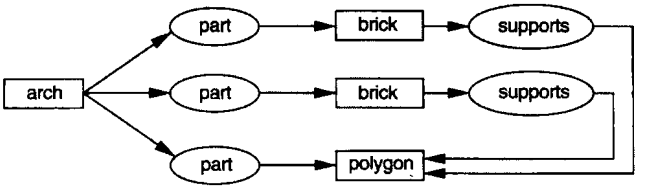


b) 另外一个拱门的例子和它的网络描述

图 10-3　对包含多个实例的描述的泛化



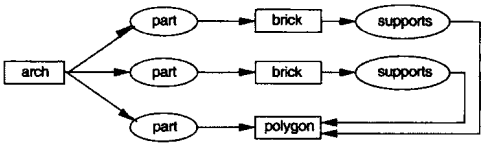
c) 给定砖和棱锥都是多边形的背景知识



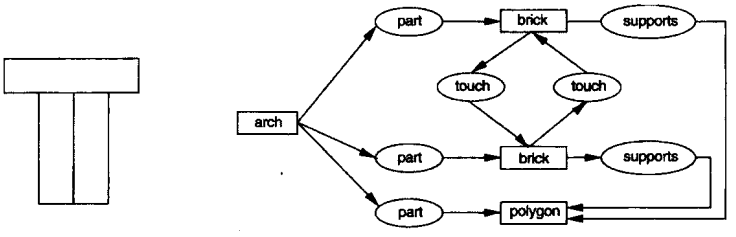
d) 包含着这两个例子的泛化

图 10-3　（续）

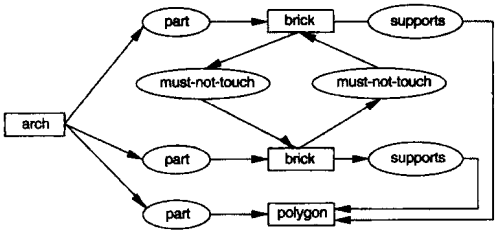
当给出有小差别的实例时，即与目标概念只有一个属性不同的实例，程序特化候选描述来排除这个实例。图 10-4a 是一个候选描述。它与图 10-4b 中实例的小差别在于 touch 关系。程序通过增加 must-not-touch 连接来排除这个小差别，从而特化图，如图 10-4c 所示。需要指出的是这个算法极其依赖于反例与目标概念的相近程度。由于与目标仅有一处不同，小差别可以帮助算法确定怎样来特化候选概念。



a) 拱门的候选描述



b) 小差别和它的描述



c) 转化来排除小差别的拱门的描述

图 10-4　排除小差别的描述的特化

注：在图 10-4c 中为 10-4a 加了约束以使其不与 10-4b 匹配

这些操作——增加连接来特化网络和用更泛化的概念来替换结点或连接名——定义了一个可能的概念定义的空间。Winston 的程序在训练数据的驱动下，采用爬山法对概念空间进行搜索。因为程序并不回溯，它的性能对于训练数据的顺序就会非常敏感，差的顺序会导致程序在搜索空间中走入死胡同。就像教师组织课程来帮助学生学习一样，提供给程序的训练数据必须要以能辅助学习期望概念的顺序给出来。训练数据的质量和顺序对于程序的图匹配算法也很重要，有效的匹配要求图不能差异太大。

虽然是归纳学习的一个较早的例子，Winston 的程序却阐明了本章介绍的大多数机器学习技术所共有的特征和问题：用泛化和特化操作来定义一个概念空间；用数据来指导对空间的搜索；学习算法对训练数据质量的敏感性。接下来的几节将讨论一下这些问题和机器学习对它们的求解技术。

10.2　变形空间搜索

变形空间搜索（version space search）（Mitchell 1978，1979，1982）把归纳学习解释为对概念空间的搜索。变形空间搜索利用了这样一个事实：泛化操作对空间中的概念进行排序，然后用这个顺序来指导搜索。

10.2.1　泛化操作符和概念空间

泛化和特化是定义一个概念空间的最常用的两种操作。机器学习中用到的最主要的泛化操作有：

1）用变量替换常量。例如：

color(ball, red)

泛化为：

color(X, red)

2）从合取表达式中去掉一些条件。

shape(X, round) ∧ size(X, small) ∧ color(X, red)

泛化为：

shape(X, round) ∧ color(X, red)

3）对表达式增加一个析取式。

shape(X, round) ∧ size(X, small) ∧ color(X, red)

泛化为：

shape(X, round) ∧ size(X, small) ∧ (color(X, red) ∨ color(X, blue))

4）用属性的超类来替换属性。如果我们知道 primary_color 是 red 的超类，则：

color(X, red)

泛化为：

color(X, primary_color)

可以用集合论的术语来考虑泛化：令 P 和 Q 分别为匹配谓词演算表达式 p 和 q 的语句集合。表达式 p 比 q 更泛化当且仅当 P⊇Q。在最上面的例子中，匹配 color(X, red) 的语句集包含匹配 color(ball, red) 的语句集。类似地，在第二个例子中，我们可以认为红的圆的对象集合是小的红的圆的对象集合的超集。“更泛化”关系定义了逻辑语句空间上的一个偏序关系。我们用符号“≥”来表示这种关系，p≥q 表示 p 比 q 更泛化。这种顺序是学习算法搜索时约束的重要来源。

我们通过*覆盖*这个概念来形式化这种关系。如果概念 p 比 q 更泛化，我们说 p *覆盖* q。我们这样定义覆盖关系：令 p(x) 和 q(x) 为分类对象概念的正例描述。换句话说，对于物体 x，有

p(x)→positive(x)和 q(x)→positive(x)。p 覆盖 q 当且仅当 q(x)→positive(x)是 p(x)→positive(x)的逻辑结果。

例如，color(X, Y)覆盖 color(ball, Z)，于是又覆盖 color(ball, red)。作为一个简单的例子，考虑有这样一些属性和值的对象域：

Sizes = {large, small}
Colors = {red, white, blue}
Shapes = {ball, brick, cube}

这些对象可以用谓词 obj(Sizes, Color, Shapes)来表示。变量替换常量这个泛化操作定义了如图 10-5 所示的空间。我们可以把归纳学习看成是对与所有训练实例一致的概念空间的搜索。

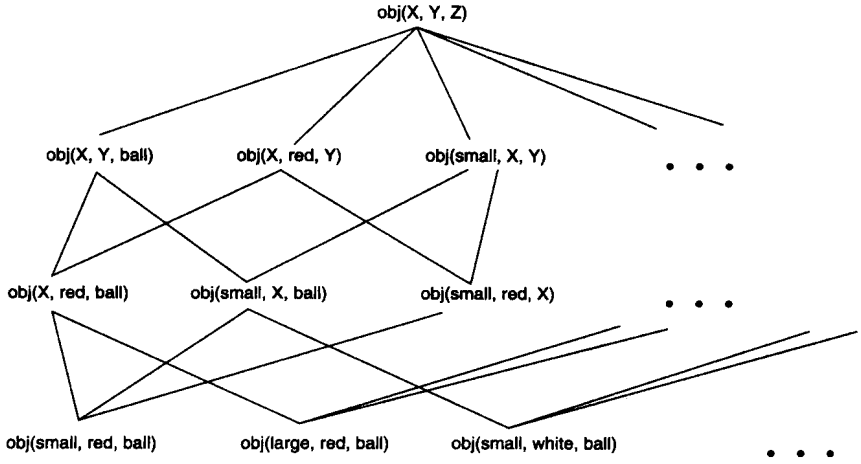


图 10-5　一个概念空间

10.2.2　候选解排除算法

本节讨论三种搜索概念空间的算法（Mitchell 1982）。这些算法依赖于变形空间这个概念，它是指与训练实例一致的所有概念描述的集合。这些算法在有更多实例可用时缩减变形空间的大小。前两个算法缩减变形空间的方向分别是：由特殊到一般和由一般到特殊。第三种算法，称为候选解排除，它把这两种办法结合起来成为双向搜索。接下来将描述并评估这些算法。

这些算法是数据驱动的，它们基于训练数据中发现的规则来泛化。这些算法用的是已经分类好的数据，所以也是监督学习（supervised learning）。

像 Winston 的学习结构型描述的程序那样，变形空间的搜索使用目标概念的正例和反例。虽然也可以只用正例来泛化，但反例对于防止算法超泛化有重要的作用。理想情况下，学习到的概念不仅要足以覆盖所有的正例，还要能排除所有的反例。在如图 10-5 所示的空间中，一个能够覆盖所有正例集合的概念是 obj(X, Y, Z)。但是，这个概念可能太泛化了，因为它概括了属于目标概念的所有实例。避免超泛化的一种方法是尽可能小地泛化使它只覆盖正例，另外一种方法是用反例来排除超泛化了的概念。如图 10-6 所示，反例是通过让学习器特化概念的方法来防止超泛化的。本节中的算法都用到了这两种技术。对假设集 S，我们如下定义特殊到一般的搜索：

开始
初始化 S 为第一个正例；
N 为目前所见的所有反例集合；

对每个正例 p
　　开始
　　对每个 s∈S，如果 s 不匹配 p，用匹配 p 的最特殊的泛化来替换 s；
　　排除 S 中比 S 中其他假设更一般的假设；
　　排除匹配 N 中已有反例的 S 中的假设；
　　结束；
对每个反例 n
　　开始
　　排除匹配 n 的 S 的所有成员；
　　添加 n 到 N 中，以检查后来的假设防止超泛化；
　　结束；
结束



图 10-6　反例在防止超泛化中的作用

特殊到一般算法维护一假设集 S，或者说是候选概念定义集。为了避免超泛化，这些候选定义是训练数据的最特殊的泛化（maximally specific generalization）。一个概念 c 是最特殊的，如果它覆盖所有的正例，而不覆盖反例，并且对任意其他覆盖正例的概念 c′，有 c≤c′。图 10-7 给出了在图 10-5 所示的变形空间中运用这种算法的例子。由特殊到一般的变形空间搜索算法已经在补充材料中用 Prolog 实现。

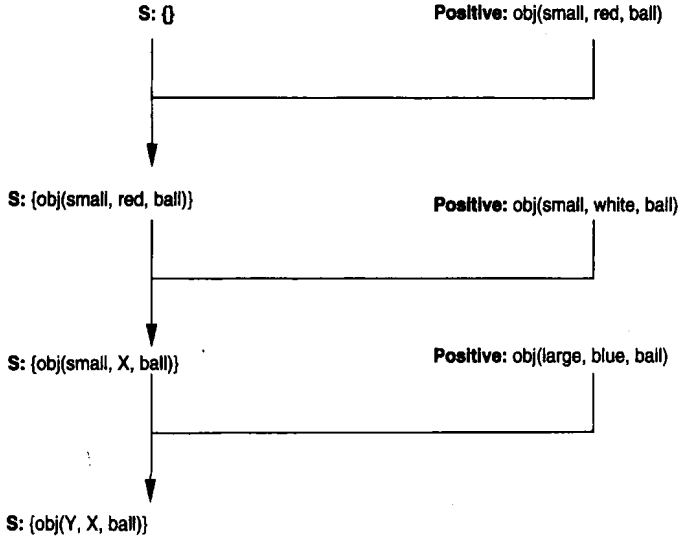


图 10-7　学习概念“球”的概念空间由特殊到一般的搜索

我们还可以沿一般到特殊的方向搜索。这种算法维护一个最一般概念（maximally general concept）的集合，最一般概念覆盖所有正例，而不覆盖反例。一个概念 c 是最一般的，如果它不覆盖反例，并且对任意其他不覆盖反例的概念 c′，有 c≥c′。在这个算法中，反例导致了对候选概念的特化，算法用正例来排除过于特殊化的概念。

开始
初始化 G 使其包含空间中最一般的概念;
P 包含当前见到的所有正例;
对每个反例 n
　开始
　对每一个匹配 n 的 g∈G, 用不匹配 n 的最一般特化来替换 g;
　排除 G 中比 G 中其他假设更特殊的所有假设;
　排除 G 中不能匹配 P 中某些正例的所有假设;
　结束;
对每个正例 p
　开始
　排除 G 中不能匹配 p 的所有假设;
　把 p 添加到 P 中;
　结束;
结束

图 10-8 给出了在图 10-5 所示的变形空间中运用这种算法的例子。在这个例子中，算法运用背景知识：size 有值{large, small}，color 有值{red, white, blue}，shape 有值{ball, brick, cube}。如果算法要用常量替换变量的方法来特化概念，这个知识就是很必要的。

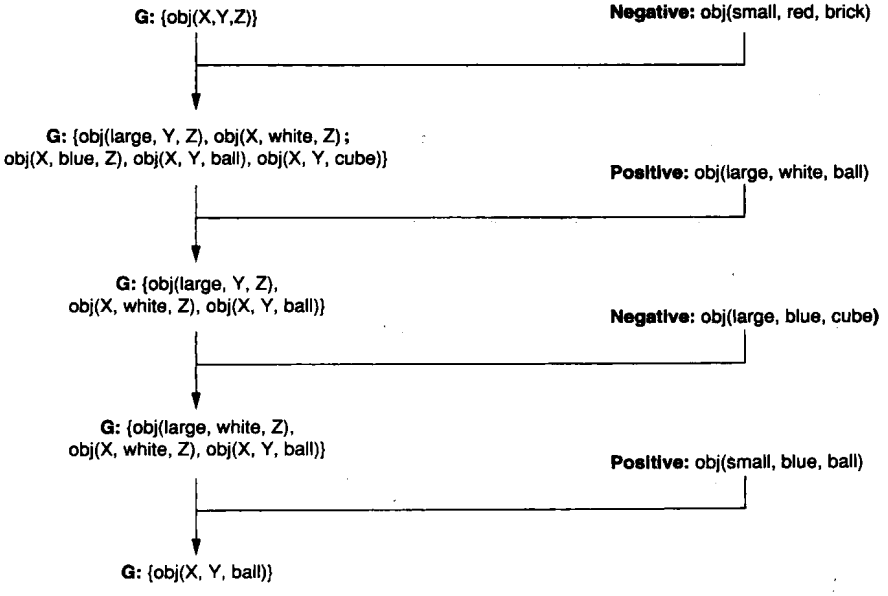


图 10-8　学习概念“球”的概念空间由一般到特殊的搜索

候选解排除算法综合上面的两种办法成为双向的搜索。双向的方法对于学习器来说有很多优点。算法维护两个候选概念集合：G 是最一般的候选概念集合，S 是最特殊的候选概念集合。算法特化 G 并泛化 S 直到它们收敛在目标概念上。算法定义如下：

开始
初始化 G 为空间中最一般的概念;
初始化 S 为第一个训练正例;
对每个新正例 p
　　开始
　　排除 G 中不能匹配 p 的所有成员;
　　对每个 $s \in S$, 如果 s 不匹配 p, 用匹配 p 的最特殊的泛化来替换 s;
　　排除 S 中比 S 中其他假设更一般的假设;
　　排除 S 中比 G 中某些假设更一般的假设;
　　结束;
对每个新反例 n
　　开始
　　排除匹配 n 的 S 的所有成员;
　　对每一个匹配 n 的 $g \in G$, 用不匹配 n 的最一般的特化来替换 g;
　　排除 G 中比 G 中其他假设更特殊的所有假设;
　　排除 G 中比 S 中某些假设更特殊的所有假设;
　　结束;
　　如果 G = S 并且两个集合都只有一个概念, 则算法就找到了与所有数据相一致的概念, 算法就终止;
　　如果 G 和 S 为空集, 则不存在这样的概念: 它覆盖所有的正例, 而不覆盖一个反例。
　　结束

图 10-9 解释了候选解排除算法在搜索如图 10-5 所示的变形空间时的行为。需要指出的是图中没有给出通过泛化或特化而产生的概念，而是给出了由于过于一般或特殊而被排除的概念。算法这一部分的详细细节留作练习，在补充材料中给出了用 Prolog 实现的一部分算法。

把两种方向的搜索结合在一个算法中有几个优点。G 和 S 分别概括了正例和反例的信息，于是没有必要再存储这些实例了。例如，为了覆盖一个正例对 S 进行泛化之后，算法用 G 排除 S 中不覆盖任意反例的概念。因为 G 是不匹配任意反例的最一般概念的集合，S 中任何比 G 中任意成员更一般的成员都肯定匹配某些反例。类似地，因为 S 是覆盖所有正例的最特殊的泛化集，G 中比 S 的成员更特殊的任何新成员肯定不能覆盖某些正例，也可能被排除。

图 10-10 给出了候选解排除算法的一个抽象描述。符号“+”代表正例；“-”号代表反例。最里面的圆圈内是已知的正例，它们被 S 中的概念覆盖。最外面的圆圈围住的是 G 中的概念所覆盖的实例，这个圆圈之外的任何实例都是反例。图中阴影部分包含目标概念，还有过于一般或特殊的概念（用? 表示）。搜索尽可能地缩小外面的概念来排除反例，并扩展最里面的概念来包含新的正例。最后，这两个集合收敛于目标概念。以这种方式，候选解排除算法能够知道何时它已经找到了单一、一致的目标概念。当 G 和 S 收敛于同一个概念时，算法就终止。如果 G 和 S 为空集，则不存在这样的概念：它覆盖所有的正例，而不覆盖一个反例。如果训练数据是

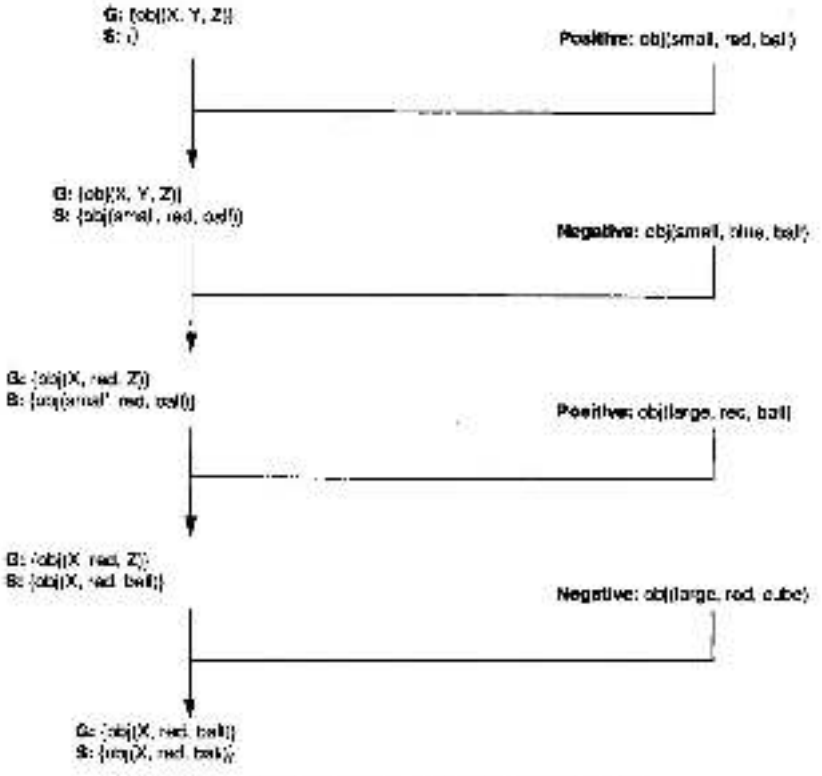


图 10-9 学习概念“红球”的候选解排除算法

不一致的或者目标概念可能不能用表示语言来表达出来的时候，这种情况就可能出现（见 10.2.4 节）。

候选解排除算法另一个令人感兴趣的方面是它的增量式的特点。增量式学习（即强化学习）算法每次接收一个训练数据，对每个实例形成一次可用但可能不完全的泛化。这与批处学习算法（参见 10.3 节中 ID3 的例子）相反。批量学习算法在开始学习之前需要给出所有的训练实例。即使在候选解排除算法收敛于单一概念之前，集合 G 和 S 也对那个概念提供了有用的限制：如果 c 是目标概念，则对所有的 $g \in G$ 和 $s \in S$，有 $s \leq c \leq g$。任何比 G 中某些概念更一般的概念会覆盖反例；任何比 S 中某些概念更特殊的概念不能覆盖某些正例。这暗示：较好地符合 G 和 S 限定的概念的实例至少是这个概念的似真实例。

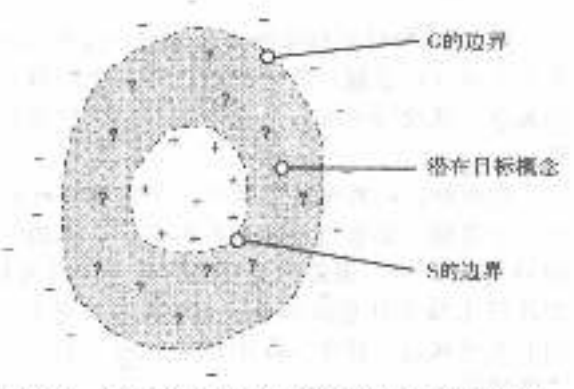


图 10-10 候选解排除算法中 G 和 S 集合的收敛边界

下一节将用一个程序的例子来阐明这种方法，这个程序用候选解排除算法来学习搜索启发式的信息。LEX（Mitchell et al. 1983）学习启发式信息来解决符号积分问题。它不仅演示了运用 G 和 S 来定义部分概念，还解释了另外的一些问题，如学习多步任务的复杂性、信用/赋罚的分配，以及一个复杂系统的学习和问题求解部件之间的关系。

10.2.3　LEX：启发式归纳搜索

LEX 学习启发式信息来求解符号积分问题。LEX 通过启发式搜索求算术表达式的积分，初始状态是待积分的表达式，然后搜索目标：不包含积分符号的表达式。系统的学习部件用问题求解器的数据来推导能改善问题求解器性能的启发式信息。

LEX 对由代数表达式的操作定义的空间进行搜索。操作符是用于积分的典型转换。它们包括：

OP1:　$\int r\, f(x)\, dx \rightarrow r \int f(x)\, dx$
OP2:　$\int u\, dv \rightarrow uv - \int v\, du$
OP3:　$1 * f(x) \rightarrow f(x)$
OP4:　$\int (f_1(x) + f_2(x))\, dx \rightarrow \int f_1(x)\, dx + \int f_2(x)\, dx$

操作符是规则，它的左半边决定了何时应用这条规则。虽然左半边定义了什么情况下会用到这个操作符，但它并没有包含何时这个操作符应该用到的启发式信息。LEX 必须通过它自身的经验来学习有用的启发式信息。启发式信息是如下形式的表达式：

If the current problem state matches P then apply operator O with bindings B.

例如，LEX 可能学习到的一个典型启发式信息：

If a problem state matches $\int$ x transcendental(x) dx,
　then apply OP2 with bindings
　　u = x
　　dv = transcendental(x) dx

这里，启发式信息暗示运用分部积分来求解 x 乘以 x 为变量的某个先验函数（例如，三角函数）的积分。

LEX 的表示概念的语言由图 10-11 所描述的符号组成。需要指出的是这些符号存在于泛化层次中，任何符号匹配比它层次低的子孙。LEX 通过用一个符号来替换它的祖先来泛化表达式。

例如，给定表达式：

$\int 3x \cos(x)\, dx$

LEX 可以用 trig 替换 cos。这样产生出表达式：

$\int 3x \,\mathrm{trig}(x)\, dx$

或者，它可以用表示任意整数的符号 k 来替换 3：

$\int kx \cos(x)\, dx$

图 10-12 给出了由这些泛化定义的操作 2（OP2）的变形空间。

LEX 的整体结构由四部分组成：

1）用候选解排除算法来寻找启发式信息的泛化程序。

2）产生问题解路径的问题求解器。

3）对问题产生正例和反例的判别标准。

4）产生新的候选问题的问题产生程序。

LEX 维护一组变形空间。每个变形空间与一个操作符相关，并表示对这个操作符学到的部分启发式信息。泛化程序用这个操作符应用的正例和反例来更新变形空间，正例和反例根据判别标准产生。一旦接收到一个正例，LEX 看这个操作符相关的变形空间是否包含这个实例。如果实例被 G 中的某些概念覆盖，变形空间就包含这个正例。LEX 然后就用这个正例来更新启发

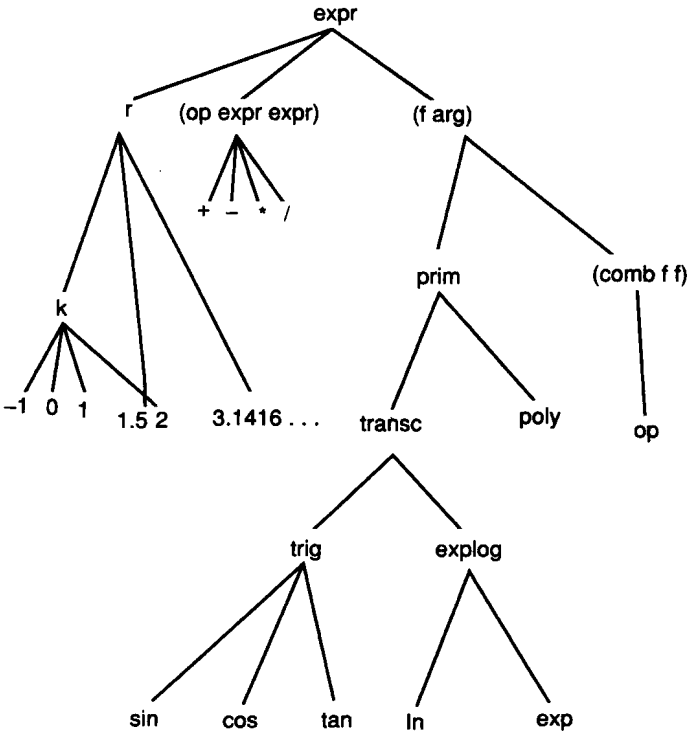


图 10-11　LEX 符号层次的一部分

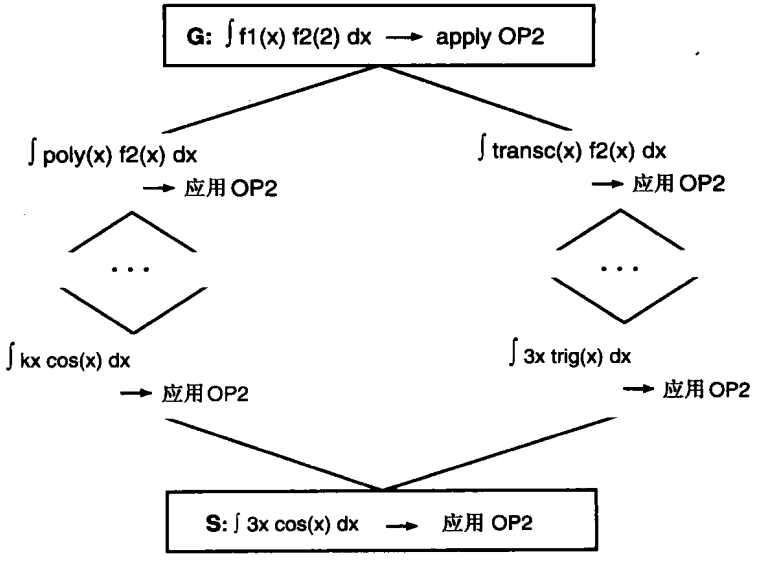


图 10-12　操作 2 的变形空间，摘自 Mitchell 等（1983）

式信息。如果现有的启发式信息都不匹配这个实例，LEX 就产生一个新的变形空间，并把这个实例当做第一个正例。这样对一个操作符就会产生多个变形空间，对应不同的启发式信息。

LEX 的问题求解器建立一棵求解积分问题要搜索的空间树，它限制了问题求解器用来求解问题的 CPU 时间。LEX 用它自己得到的启发式信息进行最佳优先搜索。LEX 执行的一个令人感兴趣的方面是它用 G 和 S 来作为启发式信息的部分定义。如果不止一个操作符可以用到给定状

态，LEX 就选择对问题状态部分匹配程度最高的那个操作符。部分匹配程度定义为匹配当前状态的概念数与介于 G 和 S 之间的所有概念数的百分比。由于对所有候选概念都测试状态的计算开销非常大，LEX 用匹配状态的概念数与实际在 G 和 S 中的概念数的百分比来估计匹配程度。需要指出的是，当 LEX 改进启发式信息时，执行的性能应该稳步改善。经验结论已经证实了这个猜测。

LEX 从问题求解器产生的解路径来得到操作符应用的正例和反例。由于没有教师，LEX 必须把操作符应用区分为正例或反例；这是个信用分配问题。在多步问题求解的过程中，程序通常不清楚序列中的哪个动作要对结果负责。如果问题求解器最后得到了错误的解，我们怎么知道哪一步导致了这个错误呢？LEX 对这个问题的判别标准是假定问题求解器返回的解的路径代表到目标的最短路经。LEX 把沿着（假定的）最短路径的操作符应用当作正例，脱离这条路径的被认为是反例。

但是，因为把问题求解器的解路径当作解的最短路径，所以判别标准必须处理这样的情况：LEX 的逐步启发式信息不能保证被承认（见 4.3 节）。问题求解器找到的解路径可能不是真正的最短解路径。为了保证它没有把一个操作符应用错误地划分为反例，LEX 先扩展由这些操作符开始的路径来确保它们不会得出一个更好的解。通常，一个问题的解会产生 2 ~ 20 个训练实例。LEX 把这些正例和反例传给泛化程序，泛化程序用它们来更新与操作符相关的变形空间。

问题产生程序是系统中最不发达的部分。虽然有很多策略用于自动问题选择，但是大多数例子涉及手工选择实例。另一方面，问题产生程序是浏览了大量策略建立起来的。一种方法会生成被两个不同的操作符的部分启发式信息所覆盖的实例，以使 LEX 学习两者之间的区别。

经验测试显示 LEX 学习有用的启发式信息是很有效的。在一个测试中，我们给定 LEX 5 个测试问题和 12 个训练问题。在训练之前，它解决 5 个测试问题，平均用 200 步；这些解没有用启发式信息来指导搜索。在学习完 12 个训练问题的启发式信息之后，它求解相同的测试问题，平均用 20 步。

LEX 解决了学习中的很多问题，包括信用分配、训练实例的选择、学习算法中的问题求解和泛化部件之间的关系。LEX 还强调概念的适当表示的重要性。LEX 的效率大多来自于概念等级的组织。这个等级很小，可以用来限制潜在启发式信息的空间并进行有效的搜索，它虽然很小，却足以表示有效的启发式信息。

10.2.4　评估候选解排除算法

候选解排除算法演示了知识表示和状态空间搜索用于机器学习问题的方式。但是，和最重要的研究一样，这个算法不能只用成功来评价它。它的一些问题仍然在机器学习研究中占了很大一部分。

像所有的搜索问题一样，基于搜索的学习必须处理问题空间的合并问题。因为候选解排除算法进行宽度优先搜索，所以它可能是没有效率的。如果应用程序中 G 和 S 过量增长，那么开发启发式方法来修剪 G 和 S 中的状态，先实现最有效的部分，或进一步对空间进行定向搜索（见 4.5 节）来学习启发式信息，会是很有用的。

解决这个问题的另外一种办法是采用归纳偏置来进一步缩减概念空间的大小，10.4 节中讨论了这种方法。这个偏置限制了用来表示概念的语言。LEX 通过选择泛化等级中的概念来施加偏置。虽然不完备，但 LEX 的概念语言足以得出很多启发式信息；同样重要的是，它把

概念空间的大小缩小到了可以控制的比例。施加了偏置的语言对缩减概念空间的复杂性是很必要的，但它们可能会使学习器不能表示想要学习的概念。在这种情况下，候选解排除算法就不能收敛于要学习的概念，使得 G 和 S 为空。表达能力和有效性之间的权衡是学习的重要问题。

算法不能收敛的原因还可以归结为训练数据中的一些噪声或不一致性。从有噪声的数据中学习的问题在实际应用中尤为重要，在实际中数据不能保证是完整的或者一致的。候选解排除算法是不能有噪声的，即使是一个错误分类的训练实例也能阻止算法在一致概念上收敛。这个问题的一个解决办法是维护多个 G 和 S 集合。除了从所有训练实例得出的变形空间外，还维护额外的空间，这个空间基于除了一个之外的所有实例，除了两个之外的所有实例，等等。如果 G 和 S 不能收敛，算法可以寻找其他仍然一致的空间。不幸的是，这种办法会产生大量的候选集合，它在大多数情况下没有效率，很不实用。

这个研究引起的另外一个问题是学习中先验知识的作用。LEX 的概念等级综合了代数的大量知识，这个知识对算法的性能非常重要。更多数量的领域知识能使学习更有效吗？10.5 节将讨论这个问题。

这些工作的重要贡献是说明了知识表示、泛化和归纳学习中的搜索之间的关系。虽然候选解排除算法只是很多学习算法中的一种，但它引出了有关复杂性、表达能力，以及运用知识和数据指导泛化的通用问题。这些问题是所有机器学习算法的中心问题；在本章中，我们将继续讨论这些问题。

10.3　ID3 决策树归纳算法

ID3（Quinlan 1986a）和候选解排除算法一样，从实例中归纳概念。它尤为吸引人的地方在于它对学到知识的表示，控制计算复杂性的办法，它选择候选概念的启发式信息，和它处理有噪声数据的潜力。ID3 用决策树来表示概念，这种表示使我们能够用测试特定属性值的方法决定对象的分类。

例如，估计个人信用风险的问题，要基于这样一些属性，如信用历史、当前债务、抵押和收入。表 10-1 列出了已知信用风险的个人的样本。图 10-13 的决策树表示了表 10-1 中的分类，因为这棵树能对表中的所有对象进行正确的分类。在决策树中，每个内部结点表示对某些属性的测试，如信用历史或债务，这个属性的每个可能的值对应树的一个分支。叶结点表示类别，如低或中等风险。通过遍历这棵树对未知类别的个人进行分类：在每个内部结点，测试此人这个属性的值，取相应的边。继续这一操作，直到到达叶结点，就找到了该对象的分类。

表 10-1　借贷应用中信用历史的数据

No.	风险	信用历史	债务	抵押	收　入
1	高	坏	高	无	0～15 000 美元
2	高	未知	高	无	15 000～35 000 美元
3	中等	未知	低	无	15 000～35 000 美元
4	高	未知	低	无	0～15 000 美元
5	低	未知	低	无	超过 35 000 美元
6	低	未知	低	充分	超过 35 000 美元

（续）

No.	风险	信用历史	债务	抵押	收　入
7	高	坏	低	无	0 ~ 15 000 美元
8	中等	坏	低	充分	超过 35 000 美元
9	低	好	低	无	超过 35 000 美元
10	低	好	高	充分	超过 35 000 美元
11	高	好	高	无	0 ~ 15 000 美元
12	中等	好	高	无	15 000 ~ 35 000 美元
13	低	好	高	无	超过 35 000 美元
14	高	坏	高	无	15 000 ~ 35 000 美

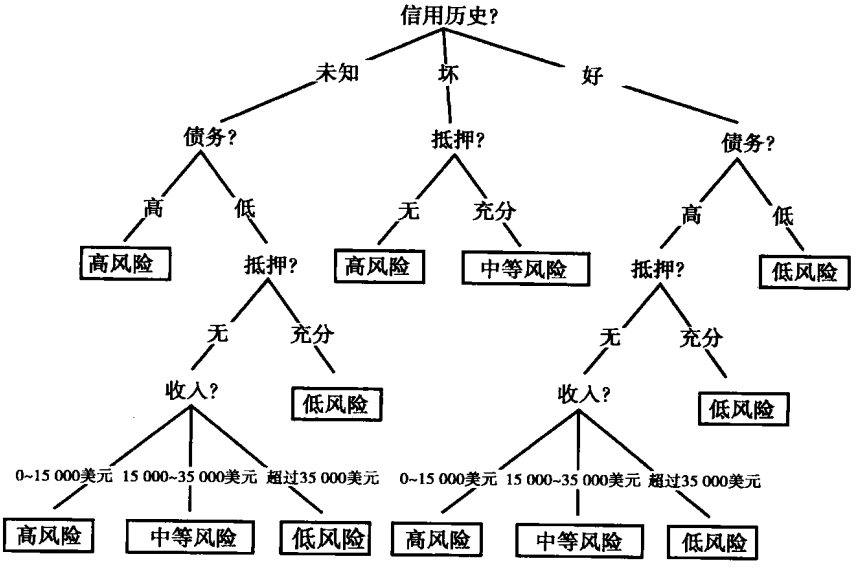


图 10-13　信用风险评定的决策树

需要注意的是在对任何给定的实例进行分类时，这棵树并没有用到表 10-1 给出的所有属性。例如，如果一个人有好的信用历史和低的债务，根据这棵树，我们可以忽略掉他的抵押和收入，而把他分类为低风险。尽管省略了某些测试，这棵树仍然对所有的实例进行了正确的分类。

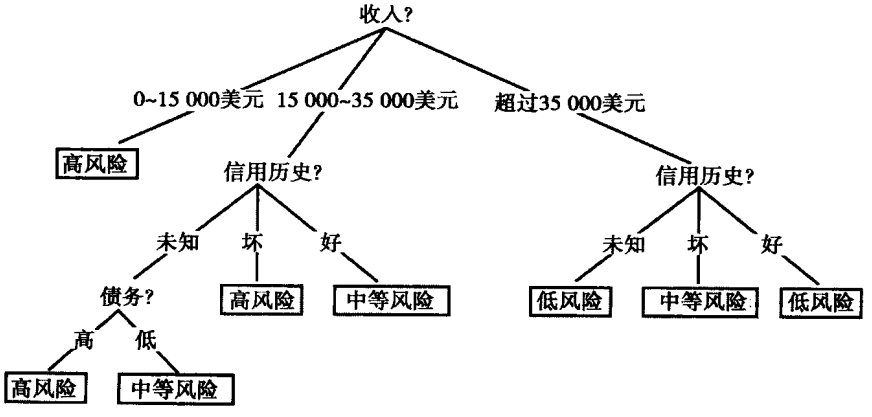


图 10-14　信用风险评定的简化的决策树

通常，对给定实例集分类所必需的树的大小，随测试属性的顺序而不同。图 10-14 给出的树比图 10-13 中的树要简单得多，但同样能对表 10-1 中的实例进行正确的分类。

给定训练实例集和能对它们正确分类的一组不同的决策树，我们可能要问哪棵树对未来实例正确分类的可能性最大。ID3 算法假定可能性最大的树是能覆盖所有训练实例的最简单的决策树。这个假定的基本原理有着悠久的历史，它强调简单性避免冗余。这个原则就是我们熟知的*奥卡姆剃刀*，它最早是由中世纪的逻辑学家 William of Occam 在 1324 年提出来的：

> 如果少做就能完成，多做的就是徒劳——如无必要，勿增实体。

奥卡姆剃刀用现在的话说是我们要接受正确符合数据的最简单解。在我们的例子中，则是能够对所有给定实例正确分类的最小决策树。

虽然奥卡姆剃刀已经证明它自己是所有智慧活动的通用启发式信息，它在这种情况下的应用有一个更特定的理由。如果假定给定的实例足以建立一个有效的泛化，则我们的问题就变成了区分必要属性和无关属性的问题。覆盖所有实例的最简单的决策树应该最不可能包含不必要的限制。虽然这个想法直觉上来想，非常吸引人，然而它是一个必须由经验验证的假定；10.3.3 节给出了一些经验结果。在研究这些结果之前，我们先给出从实例中归纳出决策树的 ID3 算法。

10.3.1　自顶向下决策树归纳

ID3 算法以自顶向下的方式建造决策树。对于任意属性，我们可以把训练实例集合划分成不相连的子集，一个划分中的所有实例有这个属性的共同的值。ID3 选择一个属性在树的当前结点测试，并用这个测试来划分实例集；然后算法递归地建立每个划分的子树。这一直持续直到划分中的所有成员在同一类别中；这个类别成为树的叶结点。由于测试的顺序对于建造简单的决策树非常重要，ID3 非常依赖于在每个子树的根结点选择测试的标准。为了简化我们的讨论，本章节描述的建造决策树的算法假定有一个适当的测试选择函数。在 10.3.2 节中，我们给出 ID3 算法选择的启发式信息。

例如，考虑 ID3 从表 10-1 建造图 10-14 中的树的方式。初始时有整个表的实例，ID3 用 10.3.2 节中描述的选择函数选择收入为根属性。它对实例集的划分如图 10-15，每个划分的元素用表中它们的编号列出来。

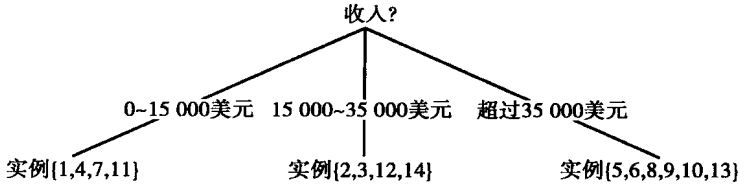


图 10-15　一个部分建立的决策树

归纳算法初始状态是对目标类别正确分类的成员的样本。ID3 根据以下算法建造决策树：

```
function induce_tree (example_set, Properties)

begin
if all entries in example_set are in the same class
  then return a leaf node labeled with that class
  else if Properties is empty
    then return leaf node labeled with disjunction of all classes in example_set
```

```
    else begin
      select a property, P, and make it the root of the current tree;
      delete P from Properties;
        for each value, V, of P,
          begin
            create a branch of the tree labeled with V;
            let partition_v be elements of example_set with values V for property P;
            call induce_tree(partition_v, Properties), attach result to branch V
          end
    end
end
```

ID3 对每个划分递归运用函数 induce_tree。例如，划分{1, 4, 7, 11}包含全部高风险的个人；ID3 相应地创建一个叶结点。ID3 选择属性信用历史作为划分{2, 3, 12, 14}子树的根结点。在图 10-16 中，信用历史把这 4 个元素进一步划分成{2, 3}、{14}和{12}。继续以这种方式选择测试和建立子树，ID3 最终产生图 10-14 中的树。读者可以自己做一下这个建立过程剩余部分的工作；我们在补充材料中给出了一个 LISP 的实现。

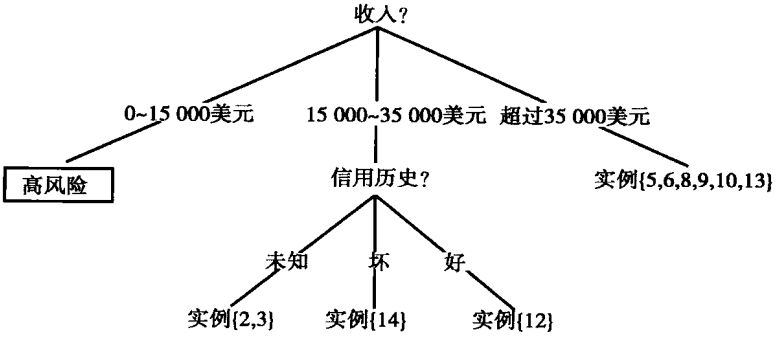


图 10-16　另外一个部分建立的决策树

在给出 ID3 的测试选择启发式信息之前，研究一下决策树建造算法和把搜索看成是对概念空间搜索的关系是很有意义的。我们可以把所有可能的决策树集合看成是定义一个变形空间。在这个空间中移动的操作包括对树增加测试。ID3 在所有可能树的空间中实现了一种贪心搜索：对当前树增加一个子树，并继续搜索，而且不回溯。这使算法非常高效而且不依赖于选择测试属性的判别标准。

10.3.2　测试选择的信息论方法

我们可以把实例的每个属性看成是对它的分类增加一定数量的信息。例如，如果我们的目标是确定一个动物的类别，它下蛋这个发现对于目标增加了一定数量的信息。ID3 通过把每个属性当作当前子树的根结点来度量信息增益。然后算法选取提供最大信息增益的属性。

信息论（Shannon 1948）提供了度量一条消息的信息含量的数学基础。可以把一条消息看成是可能消息空间的一个实例；传播消息的动作就相当于从可能的消息中挑选出一个消息。从这个观点来看，定义一条消息的信息含量既依赖于这个空间的大小，又依赖于每个可能消息出现的频率，这是很合理的。

可能消息的数量很重要，我们可以从赌博的例子中看出来：比较正确预测转盘旋转的结果的消息和正确预测投掷硬币的结果的消息。因为转盘比硬币有更多的结果出现，正确预测转盘的消息对我们更有价值：赌赢转盘赢的钱要比赌赢硬币多。所以，我们认为正确预测转盘的消息

传达了更多的信息。

每条消息出现概率对信息含量的影响可以从另外一个赌博例子中看出来。假设我对硬币进行作弊使它有头像的一面出现概率为3/4。因为我已经知道打赌猜对硬币的概率为3/4，告诉我投掷给定的硬币结果的消息就不如关于未作弊的硬币的消息有价值。

Shannon对这些进行了形式化，定义一条消息中的信息量为每条可能消息出现概率p的函数，即$-\log_2 p$。给定消息空间$M=\{m_1, m_2, \cdots, m_n\}$和每条消息出现的概率$p(m_i)$，消息M的信息含量期望如下式：

$$I[M]=\left(\sum_{i=1}^{n} -p(m_i)\log_2(p(m_i))\right) = E[-\log_2 p(m_i)]$$

消息的信息含量用比特来度量。例如，告诉掷普通硬币结果的消息的信息含量是：

$$\begin{aligned} I[\text{Coin toss}] &= -p(\text{heads})\log_2(p(\text{heads})) - p(\text{tails})\log_2(p(\text{tails})) \\ &= -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) \\ &= 1 \text{ bit} \end{aligned}$$

但如果作弊后的硬币出现头像的可能性为75%，则这条消息的信息含量是：

$$\begin{aligned} I[\text{Coin toss}] &= -\frac{3}{4}\log_2\left(\frac{3}{4}\right) - \frac{1}{4}\log_2\left(\frac{1}{4}\right) \\ &= -\frac{3}{4}*(-0.415) - \frac{1}{4}*(-2) \\ &= 0.811 \text{ bits} \end{aligned}$$

这个定义对我们直觉上对消息的信息含量的很多理解进行了形式化。信息论广泛应用于计算机科学和电信中，应用包括确定通信信道的信息传输容量，研制数据压缩算法和研制抗噪声的通信策略。ID3用信息论来选择对训练实例分类时给出最大信息增益的测试。

可以把决策树看成是传递有关决策表中训练实例分类的信息；树的信息含量由不同分类的概率来计算。例如，假定表10-1中的实例以相同的概率出现，则：

$p(\text{risk is high}) = {}^{6}/_{14},\ p(\text{risk is moderate}) = {}^{3}/_{14},\ p(\text{risk is low}) = {}^{5}/_{14}$

接下来是表10-1中的分布$D_{9.1}$和覆盖这些实例的树的信息含量：

$$\begin{aligned} I[D_{9.1}] &= -\frac{6}{14}\log_2\left(\frac{6}{14}\right) - \frac{3}{14}\log_2\left(\frac{3}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \\ &= -\frac{6}{14}*(-1.222) - \frac{3}{14}*(-2.222) - \frac{5}{14}*(-1.485) \\ &= 1.531 \text{ bits} \end{aligned}$$

在当前树的根结点做测试所提供的信息增益与树的总的信息量减去测试完之后完成分类所需的信息量相等。完成树所需的信息量定义为所有子树的信息量的加权平均值。我们通过把每棵子树的信息量乘以这棵子树中的实例对所有实例的总和的百分比来计算加权平均值。

假设有训练实例集C。如果我们把有n个值的属性P作为当前树的根结点，这将把C分成子集$\{C_1, C_2, \cdots, C_n\}$。在把P当作根结点后完成树所需的信息的期望为：

$$E[P] = \sum_{i=1}^{n} \frac{|C_i|}{|C|} I[C_i]$$

从属性 P 得到的增益通过树的总信息量减去完成树的信息期望来计算：

$$gain(P) = I[C] - E[P]$$

在表 10-1 的例子中，如果我们把属性收入（income）当作树的根结点，表中的实例划分为 $C_1 = \{1,4,7,11\}$、$C_2 = \{2,3,12,14\}$ 和 $C_3 = \{5,6,8,9,10,13\}$。完成树所需的信息期望为：

$$\begin{aligned} E[income] &= \frac{4}{14} * I[C_1] + \frac{4}{14} * I[C_2] + \frac{6}{14} * I[C_3] \\ &= \frac{4}{14} * 0.0 + \frac{4}{14} * 1.0 + \frac{6}{14} * 0.650 \\ &= 0.564 \text{ bits} \end{aligned}$$

对表 10-1 的分布的信息增益为：

$$\begin{aligned} gain(income) &= I[D_{9.1}] - E[income] \\ &= 1.531 - 0.564 \\ &= 0.967 \text{ bits} \end{aligned}$$

类似地，可以得到：

$$\begin{aligned} gain(credit\ history) &= 0.266 \\ gain(debt) &= 0.063 \\ gain\ (collateral) &= 0.206 \end{aligned}$$

由于收入提供了最大的信息增益，ID3 会选择它作为根结点。算法继续递归地对每个子树做这种分析，直到它完成树的建造。

10.3.3　评价 ID3

虽然 ID3 算法产生简单的决策树，但这种树对预测未知实例的分类不见得一定有效。人们已经用可以控制的测试和应用对 ID3 进行了评价，并证明在实际中性能良好。

例如，Quinlan 已经评价了 ID3 在学习象棋最后阶段棋局分类问题上的性能（Quinlan 1983）。最后阶段白方一个王一个车，对黑方一个王一个骑士。ID3 的目标是学习辨认导致黑方在 3 步以内输掉的棋局。特性是棋局的不同的高层次的属性，如“不能安全地移动王”。测试用到了 23 个这样的特性。

一旦将棋局的对称性考虑在内，整个问题域包含 140 万个不同的棋局，其中有 474 000 个棋局黑方在 3 步之内输掉。测试 ID3 是先给出一个随机选择的训练集，然后在随机选的 10 000 个不同棋局上测试。Quinlan 的测试给出了如表 10-2 所示的结果。预测的最大错误从 ID3 在问题域中行为的统计模型得出。进一步的分析和细节参见 Quinlan（1983）。

表 10-2　ID3 的评价

训练集合的大小	在整个空间的所占百分比	10 000 次试验中的错误数	预测的最大错误数
200	0.01	199	728
1 000	0.07	33	146
5 000	0.36	8	29
25 000	1.79	6	7
125 000	8.93	2	1

进一步的经验研究和应用的结果支持这些结果。ID3 的变种常用来处理诸如噪声和超大训练集的问题。更多的细节参见 Quinlan（1986a，b）。

10.3.4　决策树数据问题：打包、推进

Quinlan（1983）第一个提出在决策树学习中用信息论来产生子树，其工作是我们所给出的内容的基础。但我们的例子是独立的，可以直接使用。还有很多我们没有提到的问题，这些问题经常在大数据集中出现：

1）数据是坏的。可能会出现两个（或多个）相同特性集给出不同的结果。如果我们没有先验常识来去除数据，该怎么办呢？

2）某些特性集中的数据丢失，可能是因为获取它的代价太高。我们进行推测？创建一个新的值“未知”？我们怎么来平整这些不规则性？

3）某些特性集是连续的。我们通过把“收入”的连续值划分成方便的值的子集，然后用这些划分的办法来处理这个问题。还有更好的方法吗？

4）数据集对于学习算法来说可能太大了。我们怎么来处理这个问题？

这些问题的提出产生了 ID3 之后的新一代的决策树学习算法。这当中最值得一提的是 C4.5（Quinlan 1996）。这些问题还导致了诸如打包和推进技术的出现。由于分类器学习系统的数据是属性值向量或者实例，操纵数据来看是否产生了不同的分类器是很吸引人的。

*打包*通过从训练实例的置换中抽样来产生复制的训练集。*推进*使用每个置换中的所有实例，但对训练集中的每个实例赋予一个权值，用这个权值来反映向量的重要性。当权值被调整时，就产生了不同的分类器，这是因为权值引起学习器聚焦于不同的实例。在每种情况下，产生的多类分类器通过投票形成一个综合的分类器被合并起来。在打包中，每个组件分类器有同样的选票，而推进基于组件分类器的精确性分配不同的选票。

在处理大数据量的问题时，通常把数据划分成子集，对子集建造决策树，然后测试它对于其他子集的精确性。现在有关决策树学习的文献非常广泛，有大量的联机数据集，和对这些数据运用不同版本的决策树算法大量的经验结果显示。

最后，把决策树转化为相应的规则集是很简单的。我们要做的是把决策树每个可能的路径转化为一条规则。规则的模式（即它的左半边）由产生叶结点的决策组成。动作（即它的右半边）是叶结点，或者说是树的结果。这个规则集可以进一步定制来匹配决策树中的子树。这个规则紧接着可以用来对新数据进行分类。

10.4　归纳偏置和学习能力

到目前为止，我们的讨论着重于用经验数据来指导泛化。但是，成功的归纳还依赖于先验知识和对要学习的概念本质的假定。*归纳偏置*（inductive bias）是指学习器用来限制概念空间或者在这个空间中选择概念的任何标准。在下一节，我们讨论偏置需要什么和学习器运用的典型的偏置类型。10.4.2 节介绍量化归纳偏置有效性的理论结果。

10.4.1　归纳偏置

学习空间变得越来越大；如果没有一些方法来修剪它们，基于搜索的学习就没有实用性。例如，考虑从正反例中学习位串（0 和 1 组成的串）的分类问题。因为分类只是所有可能串的集合的子集，分类的总数目等于总集数目的幂集，或者所有子集的集合。如果有 m 个实例，就有 2^m 个可能的分类。但对于 n 个位的串，有 2^n 个不同的串。于是，对长度为 n 的位串，有 2 的 2^n 个不

同的分类。对 n=50，这个数目比已知世界中的分子数目还大。如果没有一些启发式的限制，学习器就不可能有效地在整个空间里却是最小的域中搜索。

偏置很必要的另一个原因是归纳泛化自身的本质。泛化并不保真。例如，如果我们遇到一个诚实的政治家，有理由假定所有政治家都是诚实的吗？遇到多少个诚实的政治家之后我们才能有理由做这个假定？Hume 在几百年前讨论了这个问题，即我们所知道的归纳问题：

> 你说一个命题是从另外的命题中推出来的；但是你必须承认推理是不直观的，也不能演示。那它的本质是什么呢？说它是实验性的是回避问题的实质。从经验所得到的推理，如同它们推理的基础一样，假设将来与过去相似，相似的力量与相似的特征同时起作用（Hume 1748）。

顺便说一下，在 18 世纪 Hume 的工作被认为是对智慧的威胁，尤其是宗教团体试图用数学来证明神的存在性和属性。在相反的理论中，试图拯救“确实性”的理论是由一个英国的传教士 Rev. Bayes 提出的理论，见第 5、9 和 13 章。下面再回到归纳中来。

在归纳学习中，训练数据只是域中所有实例的一个子集；所以，任意一个训练集可以支持很多不同的泛化。在我们位串分类器的例子中，假设给定学习器串 {1100,1010} 为某些串类别的正例。很多泛化与这些实例一致：以“1”开头以“0”结尾的所有串的集合，以“1”开头的所有串的集合，偶数奇偶性的所有串的集合，或者包含 {1100,1010} 的其他的子集。学习器要从这些泛化中选哪个来用呢？仅有数据是不充分的；所有这些选择都与数据一致。学习器必须对“可能的”概念做进一步的假设。

在学习中，这些假设经常以选择搜索空间的启发式信息的形式出现。ID3（见 10.3.2 节）中用到的信息论测试选择函数就是这种启发式信息的一个例子。ID3 对决策树空间进行爬山法搜索。在搜索的每个阶段，它研究可以用来扩展树的所有测试，并选择有最大信息增益的那个。这是“贪心”启发式信息：它选择朝目标状态移动最长距离的分支。

这个启发式信息使 ID3 能对决策树空间进行有效的搜索，还解决了从有限的数据中选择似真的泛化的问题。ID3 假定能对所有给定实例正确分类的最小的树最有可能对未来的实例正确分类。这个假定的基本原理是小的决策树最不可能做出不被数据支持的假设。如果训练集足够大而且是对象总体的如实反映，这样的树应该包含且仅包含决定类别隶属度的必要检验。像在 10.3.3 节所讨论的，经验的评价显示这个假设可以被很好地证实。很多学习算法都用到了这种对简单概念定义的偏好，如 10.6.2 节中的 CLUSTER/2 算法。

另外一种形式的归纳偏置由表示对学到概念的语法限制组成。这种偏置不是在概念空间中选择分支的启发式信息。而是，它们通过要求学到的概念用限制的表示语言来表示的方法来限制空间的大小。例如，决策树是比完全谓词演算有更多限制的语言。相应的概念空间的缩减对 ID3 的效率至关重要。

语法偏置的一个例子限制概念描述为集合 {0,1,#} 中符号的模式，被证实对位串分类非常有效。一个模式定义了所有匹配串的类，匹配由以下规则而定：

> 如果模式在某个位置是“0”，则目标串的相应位置必须是“0”。
> 如果模式在某个位置是“1”，则目标串的相应位置必须是“1”。
> 给定位置上的“#”可以匹配“1”或者“0”。

例如，模式“1##0”定义串集合{1110,1100,1010,1000}。

只考虑可以表示为单个这样模式的类，可以很大程度上缩减概念空间的大小。对长为 n 的串，我们可以定义 3^n 个不同的模式。这比没有限制的空间中 2 的 2^n 个可能概念要小得多了。这个偏置还允许很简单的变形空间搜索的实现，泛化只是把候选模式中的 1 或者 0 替换为#。但是，

我们采用这种偏置招致的代价是不能表示（也就不能学习）某些概念。例如，这种类型的一个模式不能表示偶数奇偶性的所有串的类。

这个有效性和表达能力之间的权衡是很典型的。例如，LEX 在它的符号分类法中不能区分奇数和偶数。于是，它不能学习依赖于这种区分的任何启发式信息。虽然做了一些在程序中改变偏置来响应数据的工作（Utgoff 1986），大多数的学习器仍然假定固定的归纳偏置。

机器学习已经探索出大量表象上的偏置：

合取偏置　限制学到的知识表示为合取范式的形式。因为析取范式在概念表示中的运用引起泛化的问题，合取偏置尤为常用。例如，假定在候选解排除算法中我们允许武断地运用析取范式来表示概念。因为一个正例集的最特殊泛化只是所有实例的析取，学习器就不能进行泛化。它会增加析取词*趋向无限*（ad infinitum），实现一种死记式学习的形式（Mitchell 1980）。

限制析取的数量　纯粹的合取偏置对于很多应用来说有太多的限制。解决析取范式的问题并能增加表示语言的表达能力的一种方法是允许小的数量有限的析取。

特征向量　是把对象描述为特征集合的一种表示，对象之间特征值不同。表 10-1 中的对象是作为特征集表示出来的。

决策树　是 ID3 中已经被证实有效的一种概念表示。

Horn 子句　需要对蕴涵式的形式加以限制，在自动推理和从实例中学习规则的大量程序中都用到了蕴涵式。在 14.2 节详细介绍 Horn 子句。

除了本节中讨论的语法偏置，很多程序运用特定领域知识来考虑域中已知或者假定的语义。这些知识可以提供极为有效的偏置。10.5 节讨论基于知识的方法。但是，在考虑知识在学习中的作用之前，简单地讨论一下量化归纳偏置的理论结果。在 16.2 节中还会给出学习系统中归纳偏置的总结。

10.4.2　可学习性理论

归纳偏置的目标是用这样一种方式来限制目标概念集：我们既可以有效地搜索集合，还可以找到高质量的概念定义。理论工作的一个有趣的部分是解决量化归纳偏置的有效性问题。

我们定义概念的质量为它们能对不包含在训练实例集合中的物体正确分类的能力。写一个能对已看到的所有实例正确分类的算法并不难；死记式学习就满足这一点。但是，由于在大多数问题领域中，有庞大数量的实例，或者某些实例不能用，算法只能研究一部分可能的实例。于是，学到的概念对新实例的性能就非常重要。在测试学习算法时，通常把所有实例的集合划分为训练实例集和测试实例集这两个不相交的子集。在用训练集训练完程序之后，我们用测试集来对程序进行测试。

把效率和正确性看成是概念表示语言的属性是很有用的，也就是归纳偏置而不是特定的学习算法的属性。学习算法搜索概念空间；如果这个空间可管理，并且包含性能很好的概念，则任何合理的学习算法都应该能找到这些定义；如果空间非常复杂，算法的成功完成就会受到限制。一个极端的例子会阐明这一点。

给定描述对象属性的合适的语言，“球”（ball）这个概念就是可学习的。在看到数量较少的球之后，人就能准确地定义它们：球是圆的。与一个不可学习的概念对比一下：假设有一队人围着行星跑，随机地选取几百万个对象的集合，我们称结果类为 bunch_of_stuff。不但从 bunch_of_stuff 的样本归纳出的概念需要极为复杂的表示，而且这个概念对这个集合的未来成员也不见得能正确分类。

这些现象没有对用到的学习算法做任何假定，它们只是在空间中寻找与数据相一致的概念。

ball 是可学习的，因为我们可以用几个特征来定义它：这个概念可以用偏置语言表示出来。要描述 bunch_of_stuff 这一概念需要确定类中所有对象的所有属性的概念定义。

于是，我们不是用特定的算法来定义可学习性，而是用表示概念的语言来定义它。还有，为了完成泛化，我们不把可学习性定义在特定问题域上，如学习 bunch_of_stuff。我们用概念定义语言的语法属性来定义它。

在定义可学习性时，我们不能只考虑效率；必须还要处理有限数据的问题。总的来说，我们不能指望从随机的实例样本中找到完全正确的概念。就像统计估计集合的平均数一样，我们试图找到一个近似正确的概念。所以，概念的准确性是能正确分类的实例对所有样本实例个数的概率。

除了学到的概念的准确性之外，我们还必须考虑算法找到这样概念的可能性。这就是说，有很小的可能性看到的样本是非典型的以至于无法对它学习。于是，正例的特定分布或者从这些实例中选出的特定的训练集，不一定够挑选出高质量的概念。于是考虑两个概率：样本不是非典型的概率和算法找到符合要求的概念的概率，符合要求指有正常的估计误差。在下面给出的 PAC 可学习性的定义中，这两个概率分别由 δ 和 ε 限定。

小结一下，可学习性是概念空间的一个属性，它由要表示概念的语言来决定。在评价这些空间时，我们必须既要考虑数据恰巧用尽的概率，又要考虑结论概念能对未来实例正确分类的概率。Valiant（1984）对这些进行了形式化，用的是“可能近似正确的”（PAC）学习理论。

如果存在一个算法它能有效地执行，并很有可能找到近似正确的概念，则这一类概念是 PAC 可学习的。近似正确意味着概念能正确分类大部分的新实例。这样，我们需要算法以较大的可能性找到几乎正确的概念，还需要算法自身很有效率。这个概念令人感兴趣的一个方面是它不必依赖于实例空间中正例的分布。它依赖于概念语言的本质，即偏置和正确度的期望。最后，通过做有关实例分布的假设，通常可能会得到更好的性能，也就是说，对理论上需要的要少的样本进行操作。

形式上，Valiant 定义 PAC 可学习性如下。令 C 为概念 c 的集合，X 为实例集。这个概念可以是算法、模式或划分 X 为正例和反例的某些其他手段。如果有以下属性的算法存在，则 C 是 PAC 可学习的：

1）如果对于概念错误的概率 ε 和失败的概率 δ，存在一个算法，对给定一个大小 $n = |X|$ 的实例的随机样本，在多项式时间 $1/\varepsilon$ 和 $1/\delta$ 之内，算法产生一个概念 c（C 的一个元素），使得 c 的泛化误差大于 ε 的概率小于 δ。也就是说，X 中的样本服从同一分布，y 服从：

$$P[P[y \text{ 被 } c \text{ 错误分类}] \geqslant \varepsilon] \leqslant \delta$$

2）算法的执行时间为多项式时间 n，$1/\varepsilon$ 和 $1/\delta$ 之内。

使用 PAC 可学习性这个定义，研究者已经给出几个归纳偏置是可以处理的。例如，Valiant（1984）证明了 k-CNF 表达式类是可学习的。k-CNF 表达式是合取范式形式的子句，它对析取词数量有限制；表达式形如 $c_1 \wedge c_2 \wedge \cdots \wedge c_n$，每个 c_i 是不超过 k 个文字的析取式。这个理论结果支持在很多学习算法中用到的概念为合取形式的限制。在这里我们不再证明，读者可以参阅 Valiant 的论文，他证明了这个结论还有其他偏置的可学习性。可学习性和归纳偏置的更多的结论，可以看 Haussler（1988）和 Martin（1997）。

10.5　知识和学习

ID3 和候选解排除算法基于训练数据中的规律来泛化。这样的算法常被称为基于相似性，因为泛化主要是训练实例相似性的函数。这些算法用到的偏置仅限于在学到的知识上的语法限制；它们对域的语义没有做任何大的假定。在本节中，讨论用先验领域知识来指导泛化的算法，如基

于解释的学习。

最初，认为先验知识对学习很必要的观点似乎有些自相矛盾。但是，机器学习和认知科学研究者都找到了符合上述观点的例子，他们指出当学习器有相当的领域知识时学习最有效。一种反驳学习中知识重要性的观点认为基于相似性的学习依赖于大量的训练数据。相反，人可以从少到一个的训练实例中得出可信赖的泛化，很多实际应用要求学习器也做到这一点。

另外一种反驳先验知识重要性的观点认为任意训练实例集都可以支持无限数目的泛化，它们中大多数是不相关的或者是无意义的。归纳偏置是区分它们的一个手段。在本节中，我们讨论算法，不去考虑纯粹的语法偏置，而是考虑学习中强领域知识的作用。

10.5.1　Meta-DENDRAL

Meta-DENDRAL（Buchanan and Mitchell 1978）是在归纳学习中运用知识的最早而且目前仍然是最好的例子之一。Meta－DENDRAL 获取规则，DENDRAL 程序用这些规则来分析大量的光谱数据，从化学公式和大量光谱数据中推出有机分子的结构。

大量的光谱用电子来轰击分子，致使某些化学键断裂。化学家测量结果碎片的重量，用化合物内部结构的变化来解释这些结果。DENDRAL 用规则知识来解释大量光谱数据。DENDRAL 规则的前提是部分分子结构的图。规则的结论是指出断裂位置的图。

Meta-DENDRAL 基于已知的结构从分子的光谱结果来推这些规则。我们给定 Meta－DENDRAL 一个已知化合物的结构和光谱产生的大量充足的碎片。它给出断裂发生的原因来进行解释。这些特定分子中断裂的解释用作创建通用规则的实例。

在确定一次训练中的断裂位置时，DENDRAL 运用有机化学中的“半序理论”。这个理论虽然不足以支持 DENDRAL 规则的直接创建，但它确实支持已知分子断裂的解释。半序理论由规则、限制和如下的启发式信息组成：

双键和三键不断裂。

只有大于两个碳原子的碎片在数据中出现。

运用半序理论，Meta-DENDRAL 建立对断裂的解释。这些解释指出断裂的可能位置和断裂处原子可能的迁移。

这些假设成为一个规则归纳程序的正例集。这个程序通过从一般到特殊的搜索，归纳出 DENDRAL 规则中前提的限制。它的初始状态是对一个断裂的总的一般描述：$X_1 * X_2$。这个模式意味着一个断裂可以在任何两个原子间发生，断裂由 * 来表示。它的特化模式是：

增加原子：$X_1 * X_2 \rightarrow X_3 - X_1 * X_2$

其中“－”操作符表示一个化学键，或者

实例化原子或者原子的属性：$X_1 * X_2 \rightarrow C * X_2$

Meta－DENDRAL 只从正例中学习，对概念空间采用爬山法搜索。通过限制候选规则只覆盖大约一半的训练实例来防止超泛化。程序接下来的部分评价并精炼这些规则，寻找多余的规则或者修改可能过于一般或特殊的规则。

Meta－DENDRAL 的能力在于它运用领域知识来把原始数据变成更可用的形式。通过用它的理论来排除无关系的或者潜在错误的数据，它能用相对少的训练实例进行学习，并且程序具有抗噪声的能力。训练数据必须被充分解释成完全有用的见解是基于解释的学习的基础。

10.5.2　基于解释的学习

基于解释的学习用明确表示的领域知识来建立对一个训练实例的解释，通常这个实例逻辑上服从这个理论的证明。通过泛化实例的解释，而不是泛化实例自身，基于解释的学习可以滤除噪声，选取经验的相关方面，并把训练数据组织成系统的、一致的结构。

这种思想有很多种形式。例如，表示规划的通用操作符的 STRIPS 程序（见 8.4 节）已经对这方面的研究产生了重大影响（Fikes et al. 1972）。在 10.5.1 节中我们可以看到 Meta - DENDRAL 显示了基于理论对训练实例解释的威力。最近，大量的学者（DeJong and Mooney 1986，Minton 1988）提出了这种思想的其他形式。Mitchell et al.（1986）的基于解释的泛化算法也是这种流派的典型。在本节中，讨论由 DeJong 与 Mooney（1986）提出来的基于解释的学习（EBL）算法的一个变种。

EBL 的初始状态：

1）一个目标概念。学习器的任务是确定这个概念的有效定义。依赖于特定的应用，目标概念可以是一个分类，要证明的一个定理，达到目标的一个计划，或者是一个问题求解器的启发式信息。

2）一个训练实例，目标（target）的一个实例。

3）领域理论知识，用于解释训练实例怎么成为目标概念的一个实例的规则和事实集合。

4）操作标准，概念定义可以采取的形式的某种方式的描述。

为了解释 EBL，我们给出学习什么时候物体是一个杯子的例子。这是 Winston et al.（1983）讨论的一个问题的变种，Mitchell et al.（1986）将它改编用于基于解释的学习。目标概念是可以用来推断一个物体是否是杯子的一条规则：

$$\text{premise}(X) \rightarrow \text{cup}(X)$$

其中 premise 是包含变量 X 的合取表达式。

假设有关杯子的领域理论知识包含以下规则：

$$\text{liftable}(X) \wedge \text{holds\_liquid}(X) \rightarrow \text{cup}(X)$$
$$\text{part}(Z, W) \wedge \text{concave}(W) \wedge \text{points\_up}(W) \rightarrow \text{holds\_liquid}(Z)$$
$$\text{light}(Y) \wedge \text{part}(Y, \text{handle}) \rightarrow \text{liftable}(Y)$$
$$\text{small}(A) \rightarrow \text{light}(A)$$
$$\text{made\_of}(A, \text{feathers}) \rightarrow \text{light}(A)$$

训练实例是目标概念的一个实例，即我们有：

```
cup(obj1)
small(obj1)
part(obj1, handle)
owns(bob, obj1)
part(obj1, bottom)
part(obj1, bowl)
points_up(bowl)
concave(bowl)
color(obj1, red)
```

最后，假设操作性标准需要目标概念用物体的可以观察的结构化的属性（如 part 和 points_up）来定义。可以提供能够让学习器推断一个描述是否可操作的领域规则，或者简单地列出可操作的谓词。

利用这个理论，定理证明程序可以建立为什么一个实例真是训练概念的一个实例的解释：

目标概念逻辑上是从实例中得出的证明，如图 10-17 中的第一棵树。需要注意的是，这个解释从训练数据中排除诸如 color(obj1, red) 这样的不相关概念，并捕获已知与目标相关的实例的那些方面。

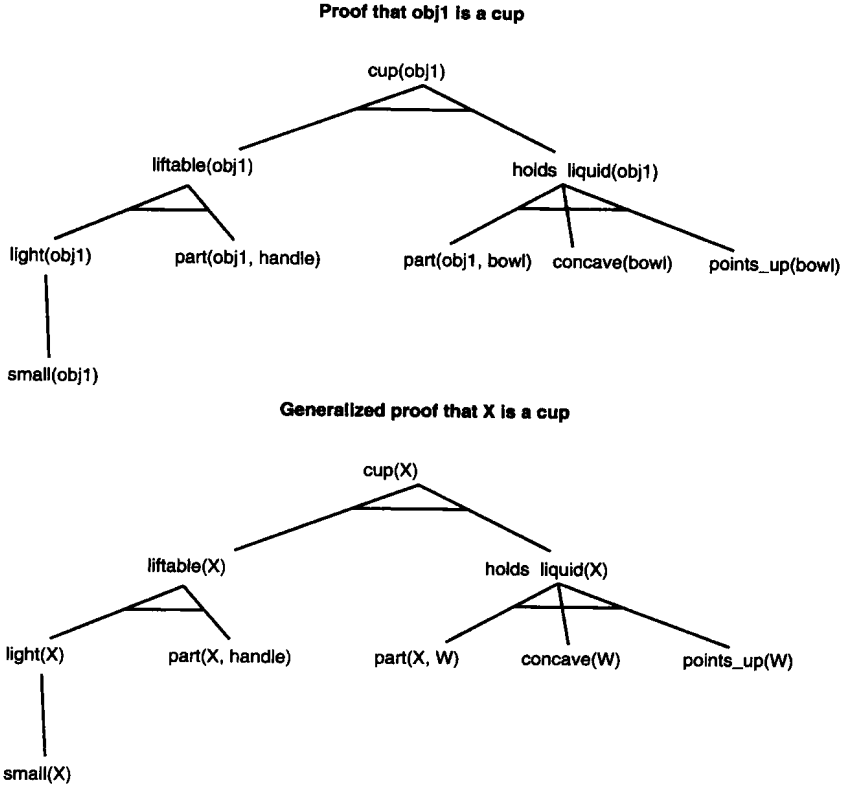


图 10-17　物体 X 是一个杯子的特化和泛化证明

基于解释的学习的下一个阶段泛化解释来产生可以用于辨识其他杯子的一个概念定义。EBL 通过用变量替换证明树中的一些常量来完成上面这一动作，这些常量只依赖于特定的训练实例，如图 10-17 所示。基于泛化后的树，EBL 定义一条新的规则，规则的结论是树的根，前提是叶结点的合取：

$$\text{small}(X) \wedge \text{part}(X, \text{handle}) \wedge \text{part}(X, W) \wedge \text{concave}(W) \wedge \text{points\_up}(W) \rightarrow \text{cup}(X)$$

在建造泛化的证明树时，目标是用变量替换那些是训练实例的一部分的常量，而保留是领域理论知识的一部分的那些常量和限制。在这个例子中，常量 handle 是领域知识的一部分，不在训练实例中。我们保留它作为获取的规则中的一个重要限制。

用一个训练实例作为向导，我们可以用很多种方式来建造一棵泛化的证明树。Mitchell et al. (1986) 完成的过程是首先建立一棵特定于训练实例的证明树，接下来通过称为*目标回归*的过程来泛化这棵证明树。目标回归用证明树的根来匹配泛化的目标（在我们的例子中是 cup(X)），用匹配所需的变量来替换常量。算法递归地对树运用这种替换，直到所有适当的常量都已经被泛化。这个过程的详细描述参见 Mitchell et al. (1986)。

DeJong and Mooney (1986) 提出了另外一种办法，实质是并行地建造泛化树和特化树。这是通过维护一棵证明树的变种来实现的，这棵树由用于证明目标的规则组成，这不同于实际证明中的变量替换。这被称为*解释结构*，如图 10-18 所示，它表示证明的抽象结构。学习器对解释结

构维护两个不同的替换列表：用来解释训练实例的特化替换列表和用来解释泛化目标的泛化替换的列表。在建造解释结构的同时，建造这些替换列表。

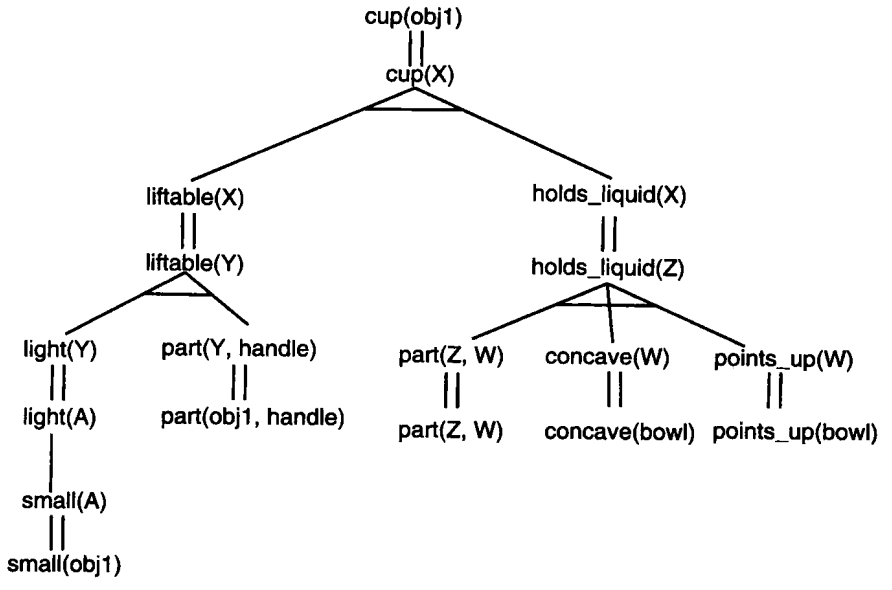


图 10-18　杯子例子的解释结构

我们以如下的方式建造泛化和特化替换的列表：令 s_s 和 s_g 分别为特化和泛化替换列表。在解释结构中对表达式 e_1 和 e_2 之间的每个替换，根据以下规则更新 s_s 和 s_g：

if e_1 is in the premise of a domain rule and e_2 is the conclusion of a domain rule
then begin
  T_s = the most general unifier of e_1s_s and e_2s_s　　% unify e_1 and e_2 under s_s
  s_s = s_sT_s　　% update s_s by composing it with T_s
  T_g = the most general unifier of e_1s_g and e_2s_g　　% unify e_1 and e_2 under s_g
  s_g = s_gT_g　　% update s_g by composing it with T_g
end
if e_1 is in the premise of a domain rule and e_2 is a fact in the training instance
then begin　　% only update s_s
  T_s = the most general unifier of e_1s_s and e_2s_s　　% unify e_1 and e_2 under s_s
  s_s = $s_s T_s$　　% update s_s by composing it with T_s
end

在图 10-18 的例子中：

s_s = {obj1/X, obj1/Y, obj1/A, obj1/Z, bowl/W}

s_g = {X/Y, X/A, X/Z}

对图 10-18 中的解释结构运用这些替换，得到图 10-17 所示的特化和泛化证明树。

基于解释的学习有以下优点：

1）训练实例经常含有不相关的信息，如前面例子中的杯子的颜色。领域理论允许学习器选择训练实例中相关的方面。

2）一个给定的实例可以允许大量可能的泛化，它们中大部分是无用的、无意义的或者错误

的。EBL 形成的泛化与特定的目标相关，并保证与领域理论相一致。

3）通过运用领域知识，EBL 允许从单个训练实例中学习。

4）解释的建立允许学习器对它的目标和经验之间未指出的关系做假设，如基于杯子的结构属性来演绎杯子的定义。

EBL 已经应用于大量的学习问题中。例如，Mitchell et al.（1983）讨论了对 LEX 算法附加 EBL。假设运用操作 1 的第一个正例是解实例 $\int 7x^2 dx$。LEX 会把这个实例放入最特殊泛化集 S 中。但是，人会立即发现解这个实例时用到的技术不依赖于系数和指数的特定的值，而是只要指数值不等于 -1，任何实数值都可以。学习器被证实操作 1 可以用于形如 $\int r_1 x^{(r_2 \neq -1)} dx$ 的任何实例，其中 r_1 和 r_2 是任意实数。LEX 被扩展成用通过基于解释的学习以运用代数知识来进行这种类型的泛化。最后，在本书的补充材料中用 Prolog 实现了一个基于解释的学习算法。

10.5.3　EBL 和知识层学习

虽然 EBL 很好地说明了知识在学习中的作用，但它还引出了很多重要的问题。其中比较明显的一个问题是基于解释的学习器真正学到了什么。纯粹的 EBL 只能从已有理论的演绎闭包中学习规则。这意味着学到的知识可以不用训练实例，而从知识库中推出来。训练实例的惟一作用是使定理证明程序着眼于与问题领域相关的方面。所以，EBL 常被看成是加速学习的一个形式或者知识库的再形成；EBL 可以让学习器执行得更快，因为它不必对新规则重新建立证明树。但是，因为它可能总是重新建立证明树，EBL 不能使学习器做任何新的东西。Dietterich（1986）在对知识层学习的讨论中对这种差别进行了形式化。

EBL 能够使规则集中不明确的信息更加清楚。例如，棋类比赛：当附加上无限制的向前预测棋局状态的能力时，棋的规则的最小知识可以让计算机下得非常好。不幸的是，对于这种方法，棋太复杂了。为了实用的目的，可以掌握下棋策略的基于解释的学习器需要能真正学到新的知识。

EBL 还允许我们抛弃学习器要有完全和正确的领域理论的要求，而着眼于在 EBL 的内容之内精炼不完全的理论。这里，学习器建造一棵部分解树。不能完成的证明的分支指明了理论的不足。在这方面还有很多有趣的问题有待于研究。它们包括探索对有缺点的理论推理的启发式信息，信用分配方法论，以及从几个失败的证明中挑选哪个来修复的问题。

基于解释的学习的一个更进一步的应用是把它与基于相似性的学习的方法结合起来。同样，有大量基本模式来运用它们，如用 EBL 在理论作用的地方精炼数据，然后把这些部分泛化的数据传递给基于相似性的学习器来做进一步的泛化。作为选择，我们可以用失败的解释作为寻找理论缺点的手段，从而引导数据搜集到基于相似性的学习器的数据收集。

EBL 研究中的其他问题包括用不可靠理论推理的技术，定理证明的方法作为建立解释的手段，处理噪声或者丢失的训练数据的方法，以及确定保存哪个泛化规则的方法。

10.5.4　类比推理

鉴于“纯粹的”EBL 仅限于演绎学习，类比提供了更为灵活的运用已有知识的方式。类比推理假定如果两种情况在某些方面相似，很可能它们在其他方面也相似。例如，如果两个房子有相似的位置、建筑和环境，则它们很可能有相同的售价。与 EBL 中用到的证明不同，类比逻辑上并不可靠。从这个意义上说它与归纳相似。如 Russell（1989）和其他人所观察到的，类比是一种单实例归纳：在房子的例子中，我们从一个已知房子的属性来归纳另一个房子的属性。

像讨论过的基于案例推理（见 7.3 节）一样，类比对把已知知识运用于新的情况非常有用。

例如，假设一个学生试图学习有关电的行为，并假定教师告诉她电类似于水，电压对应于水压，安培数对应于水流量，电阻对应于管道的容量。运用类比推理，学生可以很容易地掌握如欧姆定律这样的概念。

类比的标准计算模型定义类比的源（source）为问题的解、实例或者易于理解的理论。目标（target）是还没有完全理解的理论。类比在源和目的相应元素之间建立一个映射。类比推理把这个映射扩展到目标领域中新的元素。继续讨论“电与水类似”的类比，如果我们知道这些类比映射：电闸对应于阀门，安培数对应于水流量，和电压对应于水压，可以合理地推断还应该有对于水管容量（即水管的横断面积）的类比；这就导致了我们对于电阻的理解。

已经有很多学者提出了类比推理计算模型的统一框架（Hall 1989，Kedar - Cabelli 1988，Wolstencroft 1989）。一个典型的框架由下述阶段组成：

1）检索。给定一个目标问题，选择一个可能的源类比是很必要的。类比检索中的问题包括选取目标和源的某些特征，这些特征能增加检索有用的源类比和进行索引的可能性。总的来说，检索建立类比映射的初始元素。

2）细化。一旦检索出源之后，通常有必要获取源的额外的特征和关系。例如，可能很有必要在源领域中得出一个特定的问题求解路径（或解释）来作为对目标类比的基础。

3）映射和推理。这个阶段要做把源属性映射到目标领域中去的映射。这涉及已知的相似性和类比推理。

4）核准。这里我们确定映射是否真正有效。这个阶段可能要对映射进行修改。

5）学习。在这个阶段，获得的知识用一种对以后有用的形式来存储。

这些阶段在类比推理的很多计算模型中已经被发展和改进。例如，结构映射理论（Falkenhainer 1990，Falkenhainer et al. 1989，Gentner 1983）不仅解决了建立有用类比的问题，还给出了人怎么理解类比的一种似真的模型。运用类比时的一个中心问题是我们怎么区别有表现力的、深层的类比和比较表面化的类比。Gentner 认为真正的类比应该着重于域中的系统的和结构化的特征而不是较为表面化的类比。例如，类比“原子像太阳系”比类比“向日葵像太阳”更深层次，因为前者抓住了两个旋转体之间整个系统的因果联系，而后者描述浅显的类比，如向日葵和太阳都是圆的和黄色的。类比映射的这个属性称为系统性。

结构映射对此进行了形式化。考虑原子/太阳系类比的例子，如图 10-19 所示，Gentner（1983）对此进行了说明。源领域包括谓词：

```
yellow(sun)
blue(earth)
hotter-than(sun, earth)
causes(more-massive(sun, earth), attract(sun, earth))
causes(attract(sun, earth), revolves-around(earth, sun))
```

类比要解释的目的领域包括：

```
more-massive(nucleus, electron)
revolves-around(electron, nucleus)
```

结构映射试图把源中的因果结构迁移到目的中去。映射由以下规则限制：

1）从源中去除属性。因为类比更强调系统联系，第一个阶段就是排除源中描述表面属性的谓词。结构映射通过排除源中单一参数的谓词（一元谓词）对此形式化。这里的基本原理是高阶谓词，由于描述两个或多个实体之间的联系，更有可能抓住类比需要的系统联系。在例子中，它排除如 yellow(sun) 和 blue(earth) 的断言。注意源可能仍然包含与类比不相关的断言，如 hot-

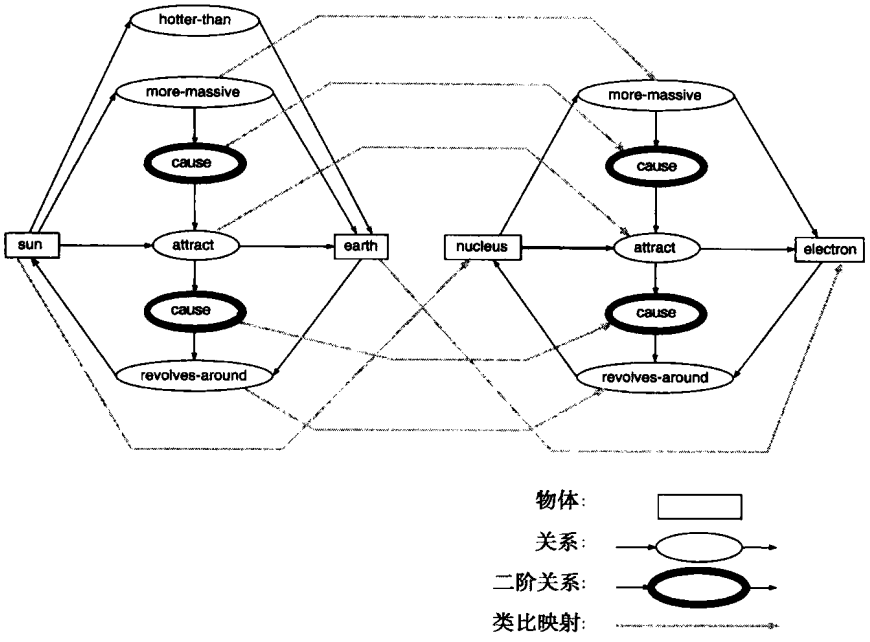


图 10-19　一个类比映射

ter-than(sun, earth)。

2）从源到目的的关系映射不变；参数到关系的映射可能不同。在例子中，如 revolves-around 和 more-massive 的关系在源和目的中是相同的。这条限制是通过很多类比的理论得出的，并大大缩减了可能映射的数量。这与映射中关系优先的启发式信息相一致。

3）在建立映射时，比较而言，高阶关系是映射的焦点。在例子中，causes 是一个高阶关系，因为它把其他的关系作为自己的参数。这种映射偏置被称为*系统性原则*。

由这些限制得到映射：

sun→nucleus
earth→electron

扩展映射得到推理：

causes(more-massive(nucleus, electron), attract(nucleus, electron))
causes(attract(nucleus, electron), revolves-around(electron, nucleus))

结构映射理论已经在很多领域中实现并进行了测试。虽然它还远不是完整的类比理论，不能解决诸如源类比检索的问题，但它已经被证实在计算复杂性方面实用可行，而且能够解释人类类比推理的很多方面。最后，像我们在 7.3 节中*基于案例推理*里指出的那样，类比在创建和运用一个有用的案例库方面起着重要的作用。

10.6　无监督学习

我们到目前为止讨论的学习算法都是*监督学习*的实现形式。它们假定教师（某些适当性度量）或其他外部的对训练实例分类的方法的存在。*无监督学习*没有教师，它需要学习器自己形成和评价概念。科学可能是人类中无监督学习的最好的例子。科学家没有教师的指点。他们提出假设来解释现象；用诸如简单性、通用性和优美性的标准来评价假设；并通过自己设计的实验来

测试假设。

10.6.1　发现和无监督学习

AM（Davis and Lenat 1982，Lenat and Brown 1984）是最早和最成功的发现程序之一，它获取数学中许多有趣（interesting）的概念，即使不是最初的概念。AM 的初始状态是集合论的概念，通过修改和合并已有概念来产生新知识的操作，和发现“有趣的”概念的启发式信息集合。通过搜索这个数学概念空间，AM 发现自然数和几个重要的数论的概念，如质数的存在性。

例如，AM 通过修改“元包”（bag）的概念来发现自然数。元包是允许同一元素多次出现的集合的泛化。例如，{a, a, b, c, c} 是一个元包。通过特化元包的定义来只允许元素的单一类型，AM 发现自然数的一个类比。例如，元包{1, 1, 1, 1} 对应于数字 4。元包的并引出加的概念：{1, 1} ∪ {1, 1} = {1, 1, 1, 1}，或者 2 + 2 = 4。我们看一下这些概念的进一步修改，AM 发现一系列的加为乘。运用通过转化已有的运算符来定义新运算符的启发式信息，AM 发现整除。它通过指出某些数只有两个约数（自身和 1）来建立质数的概念。

当产生出一个新概念时，AM 根据一些启发式信息来对它进行评价，并保持证实为“有趣”的概念。基于质数出现的频率 AM 确定质数是有趣的。在用启发式信息来评价概念时，AM 产生概念的实例，测试每个实例来看这个概念是否成立。如果一个概念对所有的实例都为真，它是同义反复（tautology），AM 给它一个较低的评价。类似地，AM 拒绝对所有实例都不为真的概念。如果一个概念对相当一部分实例为真（如质数的情况），AM 评价它为有趣的，并选出它来做进一步的修改。

虽然 AM 发现了质数和几个其他有趣的概念，但不能过多超越基本的数论来执行。在对这个工作后来的分析中，Lenat 与 Brown（1984）研究了这个程序的成功之处和它的局限。虽然 Lenat 最初认为 AM 的启发式信息是它的威力的主要源泉，后来的评价把程序的成功归因于用来表示数学概念的语言。AM 用一种 LISP 程序设计语言的变种把概念表示为递归式的结构。因为它建立在一种设计良好的程序设计语言的基础之上，这个表示定义了含有高密度的有趣概念的空间。这在搜索的开始阶段尤为正确。当探索继续，空间组合式地增长，有趣概念的百分比则变小。这个现象进一步强调了表示与搜索的关系。

AM 不能继续像早期发现那样的令人振奋的速度的另一个原因是它不能“学会学习”。在得到数学知识的同时，它不能获取新的启发式信息；结果，随着数学知识变得很复杂，搜索的质量在下降。从这个意义上，AM 没有建立对数学的深层理解。Lenat 在他后来的工作中解决了这个问题，这个程序称为 EURISKO，它试图学习新的启发式信息（Lenat 1983）。

很多程序继续探索自动发现的问题。IL（Sims 1987）运用多种学习技术来进行数学发现，包括定理证明和基于解释的学习（见 10.5 节）的方法。参见 Cotton et al.（2000）中的整数序列的自动创造。

BACON（Langley et al. 1987）发展了量化科学定律的形式的计算模型。例如，用与行星和太阳间的距离及行星的旋转周期相关的数据，BACON“重发现”行星运动的开普勒定律。通过提供在多个领域中人们怎么进行发现的似真的计算模型，BACON 为研究人类科学发现史的进程提供了一个有用的工具和方法论。SCAVENGER（Stubblefield 1995，Stubblefield and Luger，1996）用 ID3 算法的一个变种来改进它形成类比的能力。Shrager 与 Langley（1990）描述了许多其他的发现系统。

虽然科学发现是一个重要的研究领域，到目前它的进展很小。在无监督学习中一个更基本并且可能是更富有成果的问题是有关类别发现的问题。Lakoff（1987）认为分类是人类认知的基

本问题：高层次的理论知识取决于把我们的点滴经验组织成一致的分类的能力。我们大多数有用知识是有关物体分类的，如母牛，而不是特定的单个母牛，如 Blossom 或 Ferdinand。Nordhausen 与 Langley 强调分类的形成是科学发现的统一理论的基础（Nordhausen and Langley 1990）。在得出为什么化学物品以它们的方式反应时，化学基于以前的工作对化合物进行分类，如“酸”和“碱”。

在下一节，讨论概念聚类，它是在未分类的数据中发现有用分类的问题。

10.6.2　概念聚类

聚类问题的初始状态是一组未分类的物体和度量物体相似性的一种方法。目标是把物体分成符合某些质量标准的类别，如在同一类中物体的相似性最大。

数字分类法是聚类问题最老的办法之一。数字分类法依赖于物体的表示作为一组特征，其中每个可能有某些数值。一个合理的相似性度量把每个物体（n 个特征值的一个向量）看成是 n 维空间的一个点。两个物体的相似性是它们在这个空间中的欧几里德距离。

用这个相似性度量，一个普通的聚类算法以自底向上的方式建立聚类。这个方法通常称为凝聚的聚类策略，它形成聚类是通过：

1）研究所有的物体对，选择有最高相似度的那对，并把这对作为一个聚类。

2）定义聚类的特征为它的成员的某些函数，如平均，然后用这个聚类定义来替换这个聚类中的物体。

3）对这组物体重复这一过程，直到所有的物体都归为单一的聚类。

4）很多无监督学习算法可以看成是进行最大似然密度估计，即寻找数据最有可能服从的分布。一个例子是自然语言程序中的音素集合的解释，见第 15 章。

这个算法的结果是一棵二叉树，它的叶结点是实例，内部结点是大小逐步增大的聚类。

我们可以扩展这个算法以用于用符号集表示的物体，而不是数字特征。惟一的问题是度量用符号而不是数字定义的物体之间的相似性。一个合理的方法定义两个物体的相似性为它们的共同特征的比例。给定物体：

```
object1 = {small, red, rubber, ball}
object2 = {small, blue, rubber, ball}
object3 = {large, black, wooden, ball}
```

这个准则计算相似值为：

$$similarity(object1, object2) = 3/4$$
$$similarity(object1, object3) = similarity(object2, object3) = 1/4$$

但是，基于相似性的聚类算法没有充分地捕获聚类形成时起根本作用的语义知识。例如，我们对天体中的星座的描述既基于它们在天空中的相近性又结合人们已有的概念，如“北斗七星”。

在定义类别时，我们不能给所有的属性相同的权值。在任何给定的环境中，一个物体的某个特性要比其他特征都重要；简单的相似性度量对所有的特征相同对待。人分类时更多依赖于分类的目标和领域的先验知识，而不只是表面的相似性。例如，人们把鲸鱼分类为哺乳动物，而不是鱼类。表面的相似性不能对这个分类做解释，这个分类依赖于生物分类的广义目标和广泛的生理学和进化的证据。

传统的分类算法不仅不能考虑目标和背景知识，而且不能对分类结果做出有意义的语义解释。这些算法外延地表示聚类，意思是通过枚举它们的所有成员来表示聚类。算法没有产生内涵

定义，或者定义类别的语义和可以用来对类别的已知和未来成员进行分类的通用规则。例如，一个联合国秘书长的外延定义是把这些人简单地列出来。一个内涵定义，如

{X | X 曾被选为联合国秘书长}

增加了定义类别语义的优点，允许我们辨识这个类别的未来成员。

*概念聚类*通过用机器学习的技术来产生通用的概念定义和把背景知识运用于类别形成来解决这些问题。CLUSTER/2（Michalski 与 Stepp 1983）是这种方法的一个很好的例子。它用的背景知识是以在表示类别的语言上的偏置的形式给出来的。

CLUSTER/2 通过建立在 k 个种子对象周围的个体来形成 k 个类别。k 是一个参数，可以由用户来调整。CLUSTER/2 评价结果聚类，选出新的种子，重复这一过程直到达到质量标准。算法定义为：

1）从观察到的对象集合中选出 k 个种子。这可以是随机的，或者根据某些选择标准。

2）对每一个种子，用这个种子作为正例，所有其他的种子作为反例，产生覆盖所有正例且不覆盖任何反例的最一般定义。注意这可能导致其他没有种子物体的多个分类。

3）根据这些描述对样本中的所有物体分类。用覆盖类别中所有物体的最特殊描述来替换每个最一般描述。这减少了在类间重叠的不可见样本的相似度。

4）分类可能对给定物体交叉重叠。CLUSTER/2 包含调整交叉重叠定义的一个算法。

5）用距离度量，选出与每个类别中心最近的元素。距离度量可能与上面讨论的相似度量有些相似。

6）用这些中心元素作为新的种子，重复步骤 1 ~ 5。当聚类令人满意时停止。一个典型的质量度量是类别的通用描述的复杂性。例如，一个奥卡姆剃刀的变种可能会选择语法上简单定义的聚类，如那些有很少数目的合取词的定义。

7）如果聚类不令人满意并且几次迭代之后没有什么改进，选取与聚类的边缘最近的元素为新的种子，而不是取中心的元素。

图 10-20 显示了 CLUSTER/2 算法执行的各个阶段。

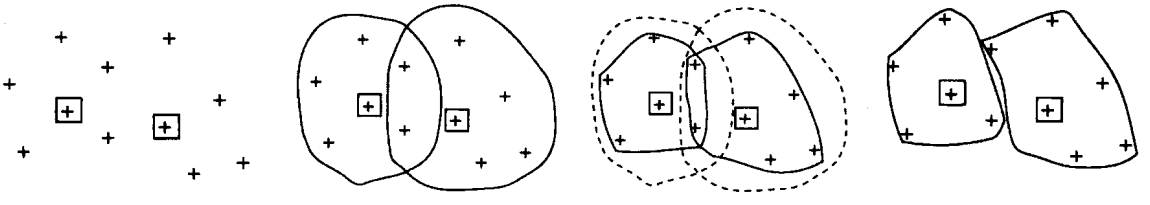


图 10-20　CLUSTER/2 运行的步骤

10.6.3　COBWEB 和分类知识的结构

很多聚类算法和很多监督学习算法（如 ID3）用成员关系的必要和充分条件来定义类别。这些条件是属性的集合，这些属性是这个类别所有成员共有的，并且只有这个类别的成员才有。虽然很多类别，如联合国代表的集合，可能是这样定义的，人类分类时并不总是遵循这个模型。实际上，人类分类比我们目前所讨论的有更大的灵活性和更丰富的结构。

例如，如果人类分类真是由成员关系的充分和必要条件定义的，我们就不能区别成员对

类别的隶属度。但是，心理学者指出了人类分类时对典型性的强烈感觉（Rosch 1978）。例如，我们认为知更鸟更像是鸟而不是鸡；与棕榈相比，橡树（至少在北纬度）是树的更典型的例子。

家庭成员相似性理论（Wittgenstein 1953）支持典型性的概念，它的观点是类别是由成员之间相似性的复杂系统来定义的，而不是由成员关系的充分必要条件来定义的。这样的类别可能不含有所有成员共有的属性。Wittgenstein 引用了游戏的例子：不是所有的游戏都需要两个或更多个玩游戏的人，如单人纸牌（solitaire）；不是所有的游戏对玩者来说都是有趣的，如“石头、剪子、布”游戏（rochambeau）；不是所有的游戏都有清晰明确的规则，如儿童玩的假装游戏（make believe）；不是所有的游戏都涉及竞争，如跳绳。虽然如此，我们认为类别是良好定义的和明确的。

人类分类也不同于最正式的遗传等级，因为人类分类的所有层次不是同样重要的。心理学者（Rosch 1978）证明了基于层次分类的存在。基于层次分类是描述物体时最常用的分类，这个分类儿童首先学会，层次某种意义上说是一个物体最基础的分类。例如，类别“椅子”比它的泛化等级（如“家具”）或者它的特化（如“办公室椅子”）更基础。“汽车”比“私家轿车”或者“交通工具”更基本。

表示类的成员关系和等级的常用方法，如逻辑、遗传系统、特征向量或者决策树都不能解释它们的作用。但做这些事情不仅是对认知科学家很重要，他们的目标是理解人类的智能；它还对人工智能的应用工程很有价值。用户用灵活性、鲁棒性和按人的标准看起来很合理的方式产生行为的能力来评价一个程序。虽然我们不需要人工智能的算法与人类的意识的结构相一致，用来发现类别的任何算法必须达到用户对于类别的结构和行为的期望的要求。

COBWEB（Fisher 1987）解决了这些问题。虽然它没有设计成人类认知的模型，它确实解释了基于层次分类和类别的隶属度的问题。另外，COBWEB 进行增量式的学习：它不要求在开始学习时给出所有的实例。在很多应用中，学习器随着时间的增长逐渐获取数据。在这些情况下，它必须从初始的数据集中建立有用的概念描述，在有更多数据可用时更新这些描述。COBWEB 还解决了确定正确的聚类数量的问题。CLUSTER/2 产生提前规定的类别数目。虽然用户可以改变这个数字，或者为了改进分类算法可以尝试不同的值，这些办法并不是特别灵活。COBWEB 用全局质量度量来确定聚类的数目，等级的深度和新实例的类别成员关系。

与我们目前看到的算法不同，COBWEB 或然性地表示类别。它不是定义类别成员关系为物体的每个特征的值都要给出来的值的集合，COBWEB 用每个特征值出现的概率来表示。$p(f_i = v_{ij} \mid c_k)$（见 5.2 节）是给定一个物体属于类别 c_k，特征 f_i 有值 v_{ij} 的条件概率。

图 10-21 解释了取自 Gennari et al.（1989）的一个 COBWEB 分类。在这个例子中，算法形成图的底部所示的 4 个单细胞动物的分类。每个动物由这些特征的值来定义：颜色、尾巴的数目及核子的数目。例如，类别 C3 有 2 条尾巴的概率为 1，有 0.5 的概率有浅的颜色，有 2 个核子的概率为 1。

如图 10-21 所示，在等级中的每个类别包含所有特征的所有值出现的概率。这对于新实例分类和修改类别结构来适应新实例都很重要。实际上，作为一个增量式算法，COBWEB 没有把这些动作分开。当给定一个新实例，COBWEB 考虑或者把实例归入已知类别或者修改等级来适应这个实例的全面的质量。COBWEB 用来评价分类质量的标准被称为分类工具（Gluck and Corter 1985）。分类工具常用于人类分类的研究。它解释基层效应和人类分类结构的其他方面。

分类工具试图最大化同一类别中的两个物体有共同值的概率和在不同类别中的物体有不同属性值的概率。分类工具定义为：

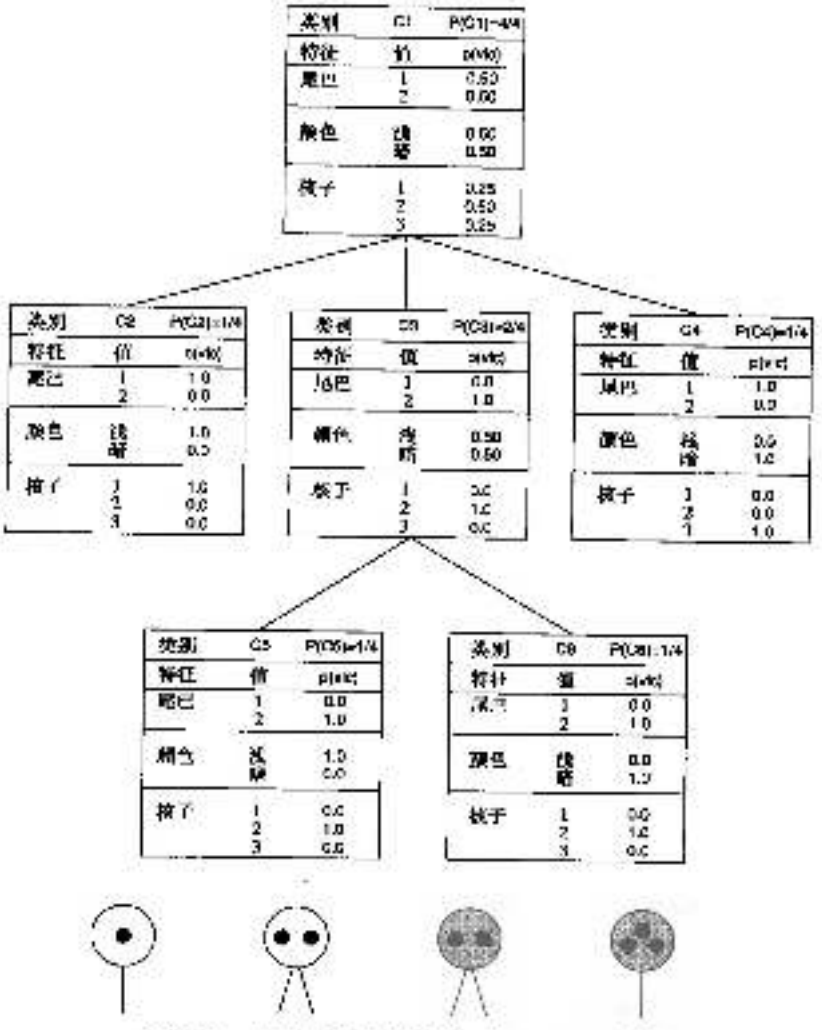


图 10-21 对4个单细胞组织的一个 COBWEB 聚类
[改编自 Cossari 等 (1989)]

$$\sum_k \sum_i \sum_j p(f_i=v_{ij})p(f_i=v_{ij}|c_k)p(c_k|f_i=v_{ij})$$

这个和取自于所有类别 c_k、所有特征 f_i 和所有特征值 v_{ij}。$p(f_i=v_{ij}|c_k)$ 称为可预言性（predictability），是给定物体属于类别 c_k 的情况下对特征 f_i 物体有值 v_{ij} 的概率。这个概率越高，同一类别中的两个物体有相同特征值的可能性越大。$p(c_k|f_i=v_{ij})$ 称为预言性（predictiveness），是给定一个物体对特征 f_i 物体有值 v_{ij} 的情况下该物体属于类别 c_k 的概率。这个概率越大，不在这一类别中的物体有这些特征值的可能性越小。$p(f_i=v_{ij})$ 可以当作一个权值，断言经常发生的特征值对评价有较大的影响。通过合并这些值，高的分类工具度量意味着同一类别中的物体有相同属性的可能性很高，而不同类别中的物体有共同属性的可能性则较低。

COBWEB 算法定义为：

```
cobweb(Node, Instance)
begin
  if Node is a leaf
    then begin
      create two children of Node, L1 and L2;
      set the probabilities of L1 to those of Node;
      initialize the probabilities for L2 to those of Instance;
      add Instance to Node, updating Node s probabilities;
    end
    else begin
      add Instance to Node, updating Node s probabilities;
      for each child, C, of Node, compute the category utility of the clustering
        achieved by placing Instance in C;
      let S1 be the score for the best categorization, C1;
      let S2 be the score for the second best categorization, C2;
      let S3 be the score for placing instance in a new category;
      let S4 be the score for merging C1 and C2 into one category;
      let S5 be the score for splitting C1 (replacing it with its child categories)
    end
    If S1 is the best score
      then cobweb(C1, Instance)                    % place the instance in C1
      else if S3 is the best score
        then initialize the new category s probabilities to those of Instance
        else if S4 is the best score
          then begin
            let Cm be the result of merging C1 and C2;
            cobweb(Cm, Instance)
          end
          else if S5 is the best score
            then begin
              split C1;
              cobweb(Node, Instance)
          end;
end
```

COBWEB 用分类工具来评价和选择可能的分类，采用爬山法搜索可能的分类空间。它初始化分类为一个单一的类别，特征是第一个实例的特征。对每一个接下来的实例，算法从根类别开始，沿着树移动。在每一层它用分类工具来评价如下产生的分类：

1）把实例置于最佳的现存类别。

2）增加一个只含有一个实例的新类别。

3）合并两个现存的类别为一个新类别，增加这个实例到这个新类别。

4）把一个现存类别分成两个，把实例置于这两个中最佳的类别。

图 10-22 解释了合并和划分结点的过程。为了合并两个结点，算法产生一个新结点，并把现存结点当作这个结点的孩子结点。它通过合并孩子结点的概率来计算新结点的概率。划分结点是用它的孩子结点替换结点自身。

这个算法是有效的，产生合理数目的类别的分类。因为它允许概率成员关系，它的类别是灵活的、鲁棒的。另外，它还证明了基层分类效应和通过部分类别匹配的概念来支持典型性和隶属度的概念。COBWEB 不依赖于二值逻辑，而是像模糊逻辑一样，把类别成员关系的“模糊性”看成是以灵活和智能的方式来学习和推理的必要部分。

接下来讨论强化学习，像 11.2 节中的分类器系统一样，解释从环境得到的反馈来学习在那个环境中问题求解的条件/响应关系的最佳集合。

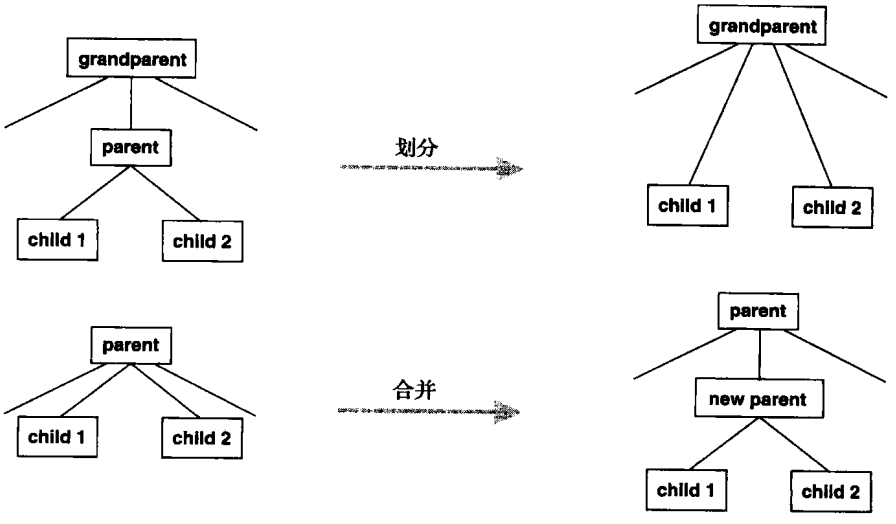


图 10-22　结点的合并和划分

10.7　强化学习

人类（通常!）从与外界环境的交互中学习。但是，思考的一瞬间使我们想起由动作得到的反馈并不总是立即的和直接的。例如，在人类的关系中，经常要费好长时间才能充分欣赏动作所得出的结果。与外界的交互给我们原因和结果（Pearl 2000），动作的结果，甚至我们怎么才能完成复杂的目标。作为智能主题，我们制定在和从我们的世界中工作的策略。“世界”是一个教师，但她的课程经常是精细的，有时难以完成。

10.7.1　强化学习的组成部分

在强化学习中，设计算法来把外界环境转化为最大化奖励量的方式的动作。我们并没有直接告诉主体要做什么或者要采取哪个动作，而是主体通过看哪个动作得到了最多的奖励来自己发现。主体的动作的影响不只是立即得到的奖励，而且还影响接下来的动作和最终的奖励。试错搜索（trial-and-error search）和延期强化（delayed reinforcement）这两个特性是强化学习中两个最重要的特性。所以，强化学习是比本章中前面所看到的学习更通用的方法学。

强化学习不是通过特殊的学习方法定义的，而是通过在环境中并响应外界环境的动作定义的。任何解决这种交互的学习方法都是一个可接受的强化学习方法。强化学习也不是监督学习，在有关机器学习的部分我们都可以看出来。在监督学习中，“教师”用实例来直接指导或者训练学习器。在强化学习中，学习主体自身通过训练、误差和反馈来学习在环境中完成目标的最佳策略（见 11.2 节的分类器学习）。

强化学习器必须考虑的另外一个问题，是在只用目前所知道的和进一步探索世界之间权衡。为了优化它获得奖励的可能性，主体必须不仅知道已经知道的，而且要探索未知的那部分世界。探索允许主体（可能的）在未来做出更好的选择；这样很明显的，或者一直探索或者从不探索的主体通常会失败。主体必须探索大量的选择，同时选出看起来是最好的那部分选择。对于有随机参数的任务，探索动作必须做多次来得到可信赖的奖励的估计（关于随机强化学习问题的细节见第 13 章）。

本书前面给出的很多问题求解算法，包括计划程序、决策制定程序和搜索算法，可以看成是

在强化学习的上下文环境中。例如，我们可以用 teleo-reactive 控制程序（见 7.4 节）来建立一个计划，然后用一个强化学习算法评价它是否成功。事实上，DYNA - Q 强化学习算法（Sutton 1990，1991）综合了学习与计划和动作的模型。这样，强化学习提供了一种评价在复杂环境下完成任务的计划、模型和它们的工具的方法。

现在介绍强化学习的一些术语：

t 是问题求解过程中的一个离散时间步。

s_t是 t 时刻的问题状态，它取决于 s_{t-1}和 a_{t-1}。

a_t是时刻 t 的动作，取决于 s_t。

r_t是时刻 t 的奖励，取决于 s_{t-1}和 a_{t-1}。

π 是在一个状态采取一个动作的策略。因此，π 是从状态到动作的一个映射。

π^*是最优策略。

V 映射一个状态到它的值。因此，$V^{\pi}(s)$是在策略 π 下状态 s 的值。

在 10.7.2 节中，时序差分学习在静态的 π 下对每一 s 学习 V 的值。

强化学习由四部分组成：策略 π，奖励函数 r，值映射 V 和一个环境的模型（通常情况）。策略定义在任何给定时刻学习主体的选择和动作的方法。这样，策略可以通过一组产生式规则或者一个简单的查找表来表示。像刚才指出的，特定情况下的策略可能也是广泛搜索，查询一个模型或计划过程的结果。它也可以是随机的。策略是学习主体中重要的组成部分，因为它自身在任何时刻足以产生动作。

奖励函数 r_t定义了在时刻 t 问题的状态/目标关系。它把每个动作，或更精细的每个状态 - 响应对，映射为一个奖励量，以指出那个状态完成目标的愿望的大小。强化学习中的主体有最大化总的奖励的任务，这个奖励是它在完成任务时所得到的。

赋值函数 V 是环境中每个状态的一个属性，指出对从这个状态继续下去的动作系统可以期望的奖励。奖励函数度量状态 - 响应对的立即的期望值，而赋值函数指出环境中一个状态的长期的期望值。一个状态从它自己内在的品质和可能紧接着它的状态的品质来得到值，也就是在这些状态下的奖励。例如，一个状态/动作可能有一个低的立即的奖励，但有一个较高的值，因为通常紧跟它的状态产生一个较高的奖励。一个低的值可能同样意味着状态不与成功的解路径相联系。

如果没有奖励函数，就没有值，估计值的惟一目的是为了获取更多的奖励。但是，在做决定时，值最使我们感兴趣，因为值指出带来最高的回报的状态和状态的综合。但是，确定值比确定奖励困难。奖励由环境直接给定，而值是估计得到的，然后随着时间推移根据成功和失败重新估计值。事实上，强化学习中最重要也是最难的方面是创建一个有效的确定值的方法。在 10.7.2 节中将给出一种方法，时序差分学习规则。

强化学习的一个最后和可选择的元素是环境的模型。一个模型是捕获环境行为的各个方面的一个机制。如 7.3 节中我们所看到的，模型不仅可以用来确定故障（像诊断推理），还可以作为确定动作计划的一部分。模型让我们在没有实际试验它们的情况下估计未来可能的动作。基于模型的计划是强化学习案例的一个新的补充，因为早期的系统趋向于基于纯粹的一个主体的试验和误差来产生奖励和值参数。

10.7.2　一个例子：九宫游戏

我们接下来用九宫游戏来说明一个强化学习算法，这个游戏的问题我们曾经讨论过（见第 4 章），并给出 Sutton 与 Barto（1998）的涉及强化学习的文献。对强化学习和其他求解方法（例如

最大最小法）进行比较和对比是很重要的。

提示，九宫游戏是在一个 3 ×3 格子上玩的两人游戏，如图 Ⅱ －5 所示。玩者 X 和 O，交替地在格子上放置他们的记号，首先在一行（水平、垂直或者对角线）放上 3 个记号的玩者为胜者。如读者所知，当这个游戏用完美的信息和可回溯的值来玩时，总是平局（见 4.3 节）。但是，用强化学习，我们能做一些更有趣的事情。我们将展示怎么来捕获一个不完美的对手的表现，并建立让我们对对手的优势最大化的一个策略。我们的策略还可以随着对手改进他的游戏而改进，运用模型我们能够产生叉子和其他的进攻移动。

首先，必须建立一个数字表，每个数表示一个游戏可能的状态。这些数字，也就是状态值，反映从这个状态获胜可能性的当前估计。这会支持一个必须赢的策略，也就是说，或者对手赢，或者平局算我们输。这种平算输的方法允许我们建立聚焦于赢的策略，这与 4.3 节中我们的完美信息的赢－输－平模型有所不同。实际上，这是一个很重要的不同；我们试图获取一个实际选手的技巧，而不是一些理想化选手的完美信息。这样，我们初始化表，用 1 表示我们能赢的每个位置，用 0 表示平或输的位置，用 0.5 表示其余位置，它反映了我们初始时估计从这些状态有 50% 的可能性赢。

我们现在与对手玩这个游戏。为简单起见，假定我们是 X，我们的对手是 O。图 10-23 反映了游戏中一种可能的移动序列，既有考虑到的又有选择的移动。为了产生一步移动，首先考虑从当前状态一步合法移动到的每个状态，即任何 X 可能移动的开放状态。我们查找表中保存的那个状态的当前值。在大多数时刻，我们可以做一步贪婪的移动，即取有最佳赋值函数的状态。偶尔，我们会做一步探测的移动，从其他状态中随机选取一个状态。这些探测的移动是为了考虑在游戏情形中可能看不到的一些选择，来扩大可能值的最优化。

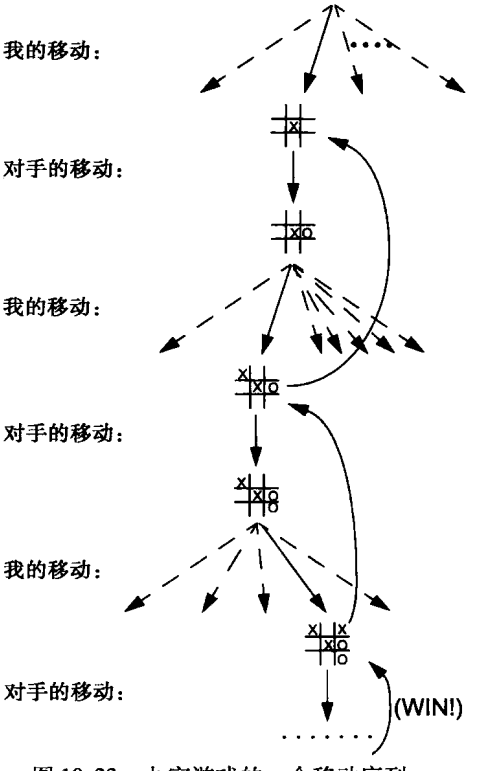


图 10-23　九宫游戏的一个移动序列

注：向下指向棋盘的虚箭头表示可能的移动选择，向上的实箭头表示当奖励函数改变状态值时的奖励

在玩游戏时，我们改变选择的每个状态的*赋值函数*。我们试图使它们的最新值反映它们在成功路径上的可能性。在前面我们把这个称为一个状态的*奖励函数*。为了做到这一点，我们退回到已经选择的一个状态的值，把它作为下一个要选择的状态的值的函数。如图 10-23 中“向上指”的箭头所示，这个退回的动作越过我们对手的选择，但它确实反映了指导我们选择下一个状态的值的集合。这样，选择的前一个状态的当前值就被修正来更好反映后来状态的值（并且最终，当然是反映赢或输的值）。我们通常是通过移动前一个状态一部分差分值来完成这个任务，这个差分值是前一个状态自身与我们选择的新状态之间的差异值。这个部分度量，称为*步长参数*，通过等式中的乘数 c 来反映：

$$V(s_n) = V(s_n) + c(V(s_{n+1}) - V(s_n))$$

在这个等式中，s_n 表示在时刻 n 选择的状态，s_{n+1} 表示在时刻 n + 1 选择的状态。这个更新的等式是*时序差分学习规则*的一个例子，因为改变是在两个不同时刻 n 和 n + 1 估计值的差分 $V(s_{n+1}) - V(s_n)$ 的一个方程。在下一节，进一步讨论这些学习规则。

时序差分方程对九宫游戏执行得非常好。我们想随着时间来减小步长参数 c，目的是为了伴随着系统的学习，对状态值进行连续减小的调整。给定对手，这能保证每个状态的赋值函数收敛于赢的概率。还有，除了周期性的探索式的移动，做出的选择实际上是最佳的移动，即对这个对手最佳的策略。但是，更令人感兴趣的是这样一个事实：如果步长从未真正减到 0，则这个策略会持续改变来反映对手玩时的任何改变/改进。

九宫游戏解释了强化学习的很多重要特征。首先，有在与环境交互时的学习，这里有对手。第二，有（反映在很多目标状态上的）清晰的目标和最佳行为，这需要计划和预做准备，以便为特定移动的延期效应保留余地。例如，强化学习算法有效地建立对低级对手的多步移动的计策。这是强化学习的一个重要特征，在没有对手的清晰模型或不进行扩展搜索的情况下，预做准备和规划的效果可以在实际中完成。

在九宫游戏的例子中，学习初始时除了游戏规则没有其他先验知识（我们只是把所有的非终态的状态初始化为 0.5）。强化学习不需要这种“新的开始”的观点。任何能用的先验知识可以使它成为初始状态值的组成部分。处理没有可用信息的状态也是可能的。最后，如果一个情况的模型是可用的，则结果模型所依据的信息可以用于状态的值。但是，重要的是要记住强化学习可以用于以下任一种情况：不需要模型，但如果模型能用或者模型可以被学习到，则可以使用模型。

在九宫游戏的例子中，奖励是随着每个状态 - 动作的决定分期付给的。我们的主体是近视的，它只考虑最大化立即的奖励。实际上，如果使用强化学习进行更深入的预先准备，我们将需要度量最终奖励的*折扣回报*（discounted return）。令折扣率 γ 表示一个未来奖励的当前值：未来 k 个时间步得到的奖励的价值是立即得到的奖励的价值的 γ^{k-1} 倍。这个奖励量的折扣在下一节给出的强化学习的动态规划方法中很重要。

九宫游戏是两人游戏的一个例子。强化学习还可以用于没有对手，而只是从环境得到反馈的情形。九宫游戏的例子的状态空间还是有限的（实际上相当小）。强化学习还可以用于当状态空间很大，或者甚至是无限的时候。在后面一种情况下，只有当状态遇到和用于一个解中时才产生状态值。例如，Tesauro（1995）用刚才描述的时序差分规则建造一个神经网络来学习下西洋双陆棋。虽然西洋双陆棋的状态空间的估计大小为 10^{20}，Tesauro 的程序与最好的人类棋手下棋的水平差不多。

10.7.3　强化学习的推理算法和应用

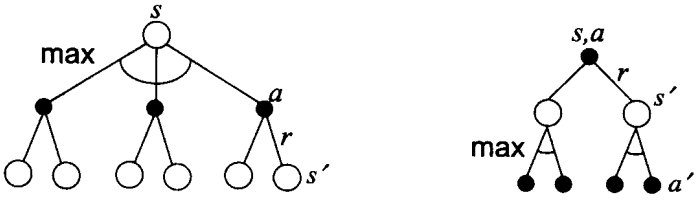
根据 Sutton 和 Barto（1998），有三大类强化学习的推理算法：*时序差分学习*、*动态规划*和*蒙特卡罗*（Monte Carlo）方法。这三种方法形成了所有当前强化学习的方法的实质上的基础。时序差分方法从取样的轨道学习，并状态到状态地回溯值。在前一节通过九宫游戏看到了时序差分学习的一个例子。

动态规划方法通过从后继状态回溯值到前驱状态来计算赋值函数。动态规划方法基于下一个状态分布的模型来接连地更新状态。强化学习的动态规划方法是基于这样一个事实：对任意策略 π 和任意状态 s，有下列递归的一致的等式成立：

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) * \sum_{s'} \pi(s \rightarrow s' \mid a) * (R^{a}(s \rightarrow s) + \gamma(V^{\pi}(s')))$$

$\pi(a \mid s)$ 是给定在随机策略 π 下状态 s 时动作 a 的概率。$\pi(s \rightarrow s' \mid a)$ 是在动作 a 下状态 s 转到状态 s′的概率。这就是对 V^{π} 的 Bellman（1957）等式。它表示了一个状态的值和它的后继状态递归计算的值之间的关系（见 4.1 节中的动态规划算法）。在图 10-24a 中，我们给出了第一步计

算，从状态 s 我们向前看三个可能的后继。对策略 π，动作 a 出现的概率为 π(a | s)。从这三个状态中的每个状态，环境可能响应其中的一个状态，比如说 s′有奖励 r。Bellman 等式对这些概率取平均值，对每个出现的可能性进行加权。它指出起始状态 s 的值必须等于期待的下一个状态的折扣（γ）值加上从这一路径产生的奖励。



a) 对V*的回溯图　　b) 对Q*的回溯图

图 10-24　对 a) V^* 和 b) Q^* 的回溯图

［改编自 Sutton 与 Barto（1998）］

典型的动态规划模型作用有限，因为它们有对一个完美模型的假定。如果 n 和 m 表示状态和动作的数目，那么即使确定性策略的总数目为 n^m，动态规划方法也能保证在多项式时间找到最优策略。在这个意义上，动态规划比任何策略空间中的直接搜索快指数级，因为完成同一保证，直接搜索必须对每个策略都进行评价。

蒙特卡罗方法不需要一个完整的模型。它们对状态的整个轨道进行抽样，基于抽样点的最终结果来更新赋值函数。蒙特卡罗方法不需要经验，即从与环境联机的或者模拟的交互中抽样状态、动作和奖励的序列。联机的经验是令人感兴趣的，因为它不需要环境的先验知识，却仍然可以是最优的。从模拟的经验中学习的功能也很强大。它需要一个模型，但它可以是生成的而不是分析的，即一个模型可以生成轨道却不能计算明确的概率。于是，它不需要产生在动态规划中要求的所有可能转变的完整的概率分布。

于是，蒙特卡罗方法通过对抽样返回值取平均的方法来解决强化学习问题。为了确保良好定义的返回值，蒙特卡罗方法定义为完全抽样，即所有的抽样点必须最终终止。而且，只有当一个抽样点结束，估计值和策略才会改变。这样，在抽样点意义上说，蒙特卡罗方法在一个抽样点上是增加的，而不是逐步的。词语“蒙特卡罗”常广泛用于操作涉及重要的随机组件的任何估计方法。这里它特定地用于基于对完全返回值取平均的方法。在第 13 章中我们将重新讨论蒙特卡罗强化学习方法。

还有用于强化学习的其他方法，其中最重要的是 Q 学习（Watkins 1989），一个时序差分方法的变种。在 Q 学习中，Q 是状态 - 动作对到学习到的值的一个函数。对所有的状态和动作：

$$Q: (\text{state} \times \text{action}) \rightarrow \text{value}$$

对 Q 学习中的一步：

$$Q(s_t,a_t) \leftarrow (1 - c) * Q(s_t,a_t) + c * [r_{t+1} + \gamma * \max_a Q(s_{t+1},a) - Q(s_t,a_t)]$$

其中 c 和 γ 都小于等于 1，r_{t+1}是状态 t+1 的奖励。我们可以在图 10-24b 中看到 Q 学习的方法，它与图 10-24a 不同，它的开始结点是一个状态 - 动作对。这个倒推规则更新每个状态 - 动作对，因此图 10-24b 的顶部的状态，倒推的根结点，是一个动作结点和产生它的状态。

在 Q 学习中，倒推从动作结点开始，最大化下一个状态的所有可能动作和它们的奖励。在完全递归定义的 Q 学习中，倒推树的底部结点是一个从根结点开始的动作和它们的后继动作的

奖励的序列可以到达的所有终端结点。联机的 Q 学习从可能的动作向前扩展，不需要建立一个完全的世界模型。Q 学习还可以脱机执行。我们可以看到，Q 学习是一种时序差分的方法。进一步的细节可以在 Watkins（1989）和 Sutton 与 Barto（1998）中找到。在第 13 章中将再次对 Q 学习进行讨论。

现在用强化学习解决了许多重要的问题，包括西洋双陆棋（Tesauro 1994，1995）。Sutton 与 Barto（1998）从强化学习的观点分析了 4.3 节中 Samuel 的西洋跳棋程序。他们还讨论了用强化学习解决杂技（acrobat）、电梯派遣、动态水道分配、加工车间分配和其他问题的方法（Sutton and Barto 1998）。

10.8 结语和参考文献

机器学习是人工智能实践中最令人振奋的分支之一，它解决了智能行为的一个中心问题，并引出了有关知识表示、搜索、甚至是人工智能自身的基本假设的许多重要问题。关于机器学习的出色综述包括 Pat Langley（1995）的《Elements of Machine Learning》、Tom Mitchell（1997）的《Machine Learning》和 Anthony Martin（1997）的《Computational Learning Theory：An Introduction》。Nils Nilsson 的《An Introduction to Machine Learning》可以在 http：//robotics. stanford. edu/people/nilsson/mlbook. html 下载也可参见 16.2 节。

关于学习的一个早期综述包括：《Machine Learning：An Artificial Intelligence Approach》（Kodratoff and Michalski 1990；Michalski et al. 1983，1986）。《Readings in Machine Learning》（Shavlik and Dietterich 1990）收集了这个领域中重要的论文，一直回溯到 1958 年。通过把所有这些研究集中于一卷中，编者为研究者和寻找这个领域介绍的读者都提供了非常有价值的服务。归纳学习在 Vere（1975，1978）以及 Dietterich 和 Michalski（1981，1986）的文章中有介绍。《Production System Models of Learning and Development》（Klahr et al. 1987）收集了机器学习中的许多论文，包括反映使用更为认知的方法进行学习的工作（SOAR）。

《Computer Systems That Learn》（Weiss and Kulikowski 1991）是对整个领域简介性的一个综述，包括神经网络、统计方法和机器学习技术。对类比学习更深入讨论感兴趣的读者可以看 Carbonell（1983，1986）、Holyoak（1985）、Kedar-Cabelli（1988）和 Thagard（1988）。对发现和理论形成感兴趣的读者可以看《Scientific Discovery：Computational Explorations of the Creative Processes》（Langley et al. 1987）和《Computational Models of Scientific Discovery and Theory Formation》（Shrager and Langley 1990）。《Concept Formation：Knowledge and Experience in Unsupervised Learning》（Fisher et al. 1991）给出了有关聚类、概念形成和其他形式的无监督学习的许多论文。

ID3 在机器学习界有很长的历史。Feigenbaum and Feldman（1963）中的基本感知器和存储器（EPAM）用一种称为差别网（discrimination net）的决策树来组织无意义音节的序列。Quinlan 是最先用信息论在决策树中产生孩子结点的。Quinlan 和其他人扩展 ID3 为 C4.5，并解决了数据中噪声和连续属性的问题（Quinlan 1996，Auer et al. 1995）。Stubblefield 和 Luger（1996）将 ID3 应用于改善一个类比推理系统对源的检索的问题。

Michie（1961）和 Samuel（1959）提供了强化学习的早期的例子。我们有关强化学习的讨论很多来源于 Sutton 和 Barto（1998）的《Reinforcement Learning》。对 Q 学习更详细的内容，推荐 Watkins（1989）的文章；对时序差分学习的分析，推荐 Sutton（1988）的最初的论文；对所有强化学习算法的更形式化的表述，推荐 Bertsekas 与 Tsitsiklis（1996）的《Neuro-Dynamic Programming》。我们将在第 13 章的概率学习中再次讨论强化学习。

《Machine Learning》是这个领域的主要期刊。当前研究的其他来源包括机器学习的国际会议

和机器学习的欧洲会议的年度会议论文集，还有美国人工智能协会（现为人工智能进展协会）会议和国际人工智能联合会议的会议论文集（对机器学习领域问题的最新更新）。

第 11 章给出了连接的学习，第 12 章给出了社会和涌现学习，第 13 章给出了动态规划和概率学习。在 16.2 节讨论学习中的归纳偏置、泛化以及学习中的其他限定条件。

10.9　习题

1. 考虑 Winston 的概念学习器在学习概念“台阶”时的行为，如图 10-25 所示，一个台阶由一个矮盒子和一个高盒子放在一起而组成。创建表示三四个例子和小差别的语义网，并给出概念的扩展。

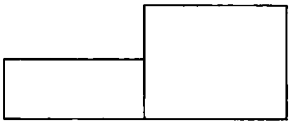


图 10-25　一个台阶

2. 图 10-9 所示的候选解排除算法的运行，没有显示产生的候选概念，而是产生由于过于一般，或过于特殊，或由于被其他概念包含而被排除的候选概念。重新做一次执行处理，显示这些概念和每个概念被排除的原因。
3. 用 Prolog 或者你选择的其他语言建立变形空间搜索算法。如果你用 Prolog 或 Java，可以在补充材料中获取关于变形空间搜索的提示。
4. 用 10.4.3 节中的信息论选择函数，详细说明 ID3 是怎么从表 10-1 中的实例建造图 10-14 中的树的。一定要说明用于每个测试的信息增益的计算和给出测试选择结果的计算。
5. 用 Shannon 的公式，说明有关轮盘赌轮结果的消息是否比有关硬币结果的消息含有更多的信息量。如果轮盘赌轮消息是“不是 00”，结论是什么呢？
6. 为某个领域的一些实例建立一个简单的表格（例如用物种对动物分类），并通过 ID3 算法建立一棵决策树。
7. 用你选择的一种语言实现 ID3 算法，用课文中信用历史的实例来运行它。如果你用 LISP，可以考虑参考 15.13 节给出的算法和数据结构。
8. 讨论用数据的连续属性时产生的问题，如货币价值，美元和美分，或者一个实体的高度，一个实数值。提出几种解决连续数据问题的方法。
9. ID3 的其他问题是坏的或丢失的数据。如果一个属性集合有两个不同的结果，那么数据是坏的。如果缺少部分数据，可能是获取这些数据的代价太高，则数据丢失。ID3 算法的扩展中怎么来解决这些问题？
10. 从 Quinlan（1993）获取 C4.5 决策树算法，并在一数据集上对它测试。这篇参考文献中有完整的程序和能用于 C4.5 的数据集。
11. 在你选择的一个问题领域建立基于解释的学习的领域理论。对几个训练实例运用这些理论，说明基于解释的学习器的行为。
12. 用你选择的语言实现一个基于解释的学习算法。如果你用 Prolog，考虑补充材料中给出的算法。
13. 考虑 10.7.2 节中的九宫游戏。用你选择的语言实现时序差分学习算法。如果你设计的算法把问题的对称性考虑在内，你预计会出现什么情况？它怎么限制了你的解？
14. 如果练习 13 中的时序差分算法与它自身玩九宫游戏，会出现什么情况？
15. 从强化学习的观点分析 Samuel 的西洋跳棋程序。Sutton 和 Barto（1998，11.2 节）对这个分析提供了建议和参考。
16. 你能从强化学习的观点分析倒立摆问题（8.2.2 节中图 8－8 所示）吗？在你的分析中，建立一些简单的奖励度量，并运用时序差分算法。
17. 另外一个适于强化学习的问题类型是所谓的方格世界。在图 10-26 中我们给出一个简单的4×4的方格世界。两个灰色的角是主体要达到的最终状态。在所有其他的状态，主体的移动可以是上、下、左、

右。主体不能离开方格移动：试图离开时不改变状态。除了最终状态，所有转移的奖励是 -1。基于 10.7.2 节给出的时序差分算法对方格给出一个解序列。参见第 13 章中关于方格世界问题的更多讨论。

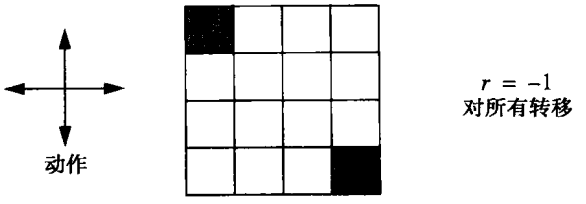


图 10-26　一个 4 ×4 方格世界的例子

[摘自 Sutton 和 Barto（1998）]

第 11 章　机器学习：连接机制

一只曾经站在热炉子上的猫将永远不会再站在热炉子上，也不再站在冷炉子上……

——马克·吐温

每一件事物在一定程度上都是模糊的直到你努力使它变得清晰……

——伯特兰·罗素

……仿佛有幻灯把神经的图样投到幕上……

——T. S. 艾略特，《J. 阿尔弗雷德·普鲁弗洛克的情歌》

11.0　简介

第 10 章重点介绍了基于符号学习方法。这种假设最主要的特征是用符号来表征具体领域中的对象和关系。在这一章中，将介绍受神经系统或生物学激励的学习方法。

神经元激励模型，有时也被认为是分布式并行处理（PDP）系统或连接系统，不再强调运用符号来求解问题。这种连接主义者认为智能出现在这种简单的、相互作用的部件（生物的或人工的神经元）组成的系统中，而且进一步可通过学习和调整神经元之间的连接来提高系统的智能。系统的处理过程分布在神经网络的各个集合或层次之间。各个集合或层次的所有神经元同时相互独立地处理各自的输入，从这个意义上说神经网络的问题求解是并行的。系统也倾向于适当的退化，因为信息和处理分布在整个网络的结点和层之上。

然而，在这种连接模型中无论在创建输入参数还是解释输出值时都存在一个强表示特性。例如，为了创立一个神经网络，设计者必须创造一种编码方法来把现实世界中的模式转变成网络中的数字量。编码方法的选择决定着神经网络的学习的成败。

在连接系统中，处理都是并行和分布式的，没有符号系统中的符号处理。领域中的模式被编码成数字向量，组件之间的连接也被数字值所代替。最后，模式的转换也是数字操作的结果——通常用矩阵乘法。这些设计者对于连接体系结构的选择就构成了系统的归纳偏置。

应用这些技术的算法和体系结构一般使用训练和有条件的方法而不是直接的程序设计。这也是这种方法最强有力的地方：一种恰当设计的网络体系结构和学习算法通常不是通过直接运行识别现实世界中的不变量，而是通过捕获现实世界中的不变量，甚至以奇怪的吸引子的方式来获取这些不变量。所有的这些实现方式构成本章的主要内容。

连接方法应用最多的领域有：

分类，决定输入的值属于哪个类别或组；

模式识别，鉴定数据中的模式或结构；

存储器收回，包括内容可寻址存储器的问题；

预测，例如从症状诊断疾病，从结果推测原因；

优化，如寻找有约束的“最好”组织；

噪声过滤，如从背景中分离信号，分辨信号中的不相干因素。

以上的这些问题用这章介绍的方法能很好地解决，但是用符号模型可能比较困难。尤其是对于这些缺乏清晰的语法定义而又需要基于感知的技能的问题。

11.1 节从历史的角度介绍神经元激励学习模型。我们介绍神经网络学习的各种构件，包括“机械”神经元；描述一些早期的重要工作，如 McCulloch-Pitts（1943）神经元。从 60 多年来神

经网络训练范例的进展中，我们可洞察今天该领域的情况。

11.2 节介绍感知机学习和 delta 规则。我们将描述用于分类的感知机的例子。11.3 节介绍有隐含层的神经网络，以及反传（backpropagation）学习规则。为了解决早期系统的数据非线性可分问题，在人工神经网络开始引入了上述的方法。反传算法用连续的阈值对多层系统中的不正确响应结点进行修正。

11.4 节介绍了由 Kohonen（1984）和 Hecht-Nielsen（1987）提出的竞争学习模型。在这种模型中，网络权值向量表示的是模式而不是连接强度。胜者全拿算法就是选择模式的权值跟输入的向量最相似的结点，然后调整这个结点的模式的权值使之更接近于输入的向量。这是一种无监督学习方法，获胜结点的选择方式是通过比较模式向量和输入向量的相似性大小，最相似的就是获胜结点。在一个神经网络系统中组合 Kohonen 和 Grossberg（1982）模型将形成一个有趣的刺激－响应学习模型，这叫做*逆传*（counter-propagation）学习。

11.5 节介绍 Hebb（1949）的加强学习模型。Hebb 认为每当一个神经元对另外一个神经元的激活有贡献时，这条路径的连接强度将得到加强。Hebb 学习通过一个调整连接强度的简单算法实现。我们将分别介绍有监督和无监督两种 Hebbian 学习方法，及一种从记忆中检索模式的基于 Hebbian 规则的模型。

11.6 节介绍一个叫做*吸引子网络*的重要网络结构。它用反馈连接在网络中不断循环一个信号。它的输出是网络达到稳定时的状态值，网络的权值被创建成一些吸引子的集合，一个吸引子范围内的输入模式最终会在该吸引子达到平衡状态。因此这种吸引子通常用于存储记忆中的模式。给出一个输入模式，网络将取出网络中存储的最相似的模式或者是与网络中存储的最相似的模式相关联的模式。第一种记忆类型叫做*自相关*，第二种叫做*异相关*。理论物理学家 John Hopfield（1982）定义了一类吸引子网络，他通过最小能量函数使网络系统收敛。Hopfield 网络能被用于求解约束可满足问题，例如巡回推销员问题，方法是将优先函数映射为一个能量函数（见 11.6.4 节）。

在第 12 章介绍进化学习模型，例如遗传算法和人工生命，在第 13 章中将介绍动态规划和随机学习模型。我们讨论学习理论方面具有代表性的观点和流派，在 16.3 节还会讨论各种学习范型的优点长处。

11.1　连接网络的基础

早期历史

连接体系结构一般认为是近期发展的，其实，它可追溯到早期的计算机科学、心理学以及哲学工作。例如，John von Neumann 就对细胞自动机和神经元激励方法计算有很深的研究。早期神经学习的工作曾受动物学习的生理理论的影响，尤其是 Hebb（1949）的理论。这一节我们略述神经网络学习的基本组成成分，展示这个领域早期工作的重要意义。

最基本的神经网络是如图 11-1 的人工神经元。人工神经元包括：

输入信号 x_i。这些数据可能来自环境，也可能来自其他神经元的刺激，不同模型所允许的输入参数的范围也是不同的，大部分的输入是离散的，如从集合{1, 0}或{－1, 1}中取值，也有取实数值的。

实数权值 w_i。权值描述连接强度。

激励层 $\sum w_i x_i$。激励层就是求取输入层的输入信号和相应输入线上连接强度 w_i 的乘积的累加。因此，激励层就是计算输入信号的带权和，即 $\sum w_i x_i$。

阈值函数 f。这个函数计算神经元的最终状态或者说是输出状态，这个状态由激励层相对阈值的高低决定。这个阈值函数一般产生类似于实际神经元的开/关状态。

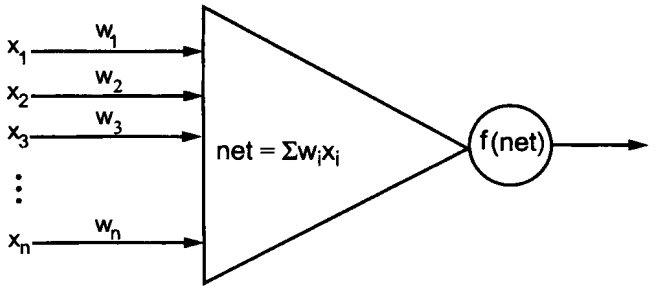


图 11-1　一个人工神经元、输入向量 x_i、每条输入线上的权值以及用来确定神经元输出值的阈值函数 f

注：将本图与图 1-2 中的实际神经元进行比较

除了这些单个神经元的性质，神经网络还有以下的全局特性：

网络的拓扑结构。网络的拓扑结构就是单个神经元之间的连接模式。这种拓扑结构是神经网络归纳偏差的主要来源。

使用的学习算法。本章将介绍很多种学习算法。

编码方法。这包括输入数据和处理输出结果的表示和解释。

最早的神经计算例子是 McCulloch-Pitts 神经元模型（McCulloch and Pitts 1943）。McCulloch-Pitts 神经元的输入只是 +1 和 -1 两个值，激励函数就是对输入信号和相应的权值的乘积求和，如果其和大于或等于 0，则神经元返回 +1，否则，返回 -1。McCulloch 和 Pitts 演示了怎样用这种神经元构造逻辑函数，认为这种神经元系统提供了一种完整的可计算模型。

图 11-2 表示的是计算逻辑函数的 McCulloch-Pitts 神经元模型。这个神经元有三个输入：x 和 y 是逻辑函数的变量，第三个是一个常量 +1，有时也叫做*偏差*。输入变量和偏差各自对应的权值分别是 +1、+1 和 -2。因此，对于任何值 x 和 y，这个神经元计算表达式 x + y - 2 的值：如果这个值小于 0，则返回 -1，否则是 +1。表 11-1 展示了计算 x ∧ y 的神经元。同样，如图 11-2 所示，除非 x 和 y 都等于 -1，否则“或”神经元输入数据的加权和大于或等于 0。

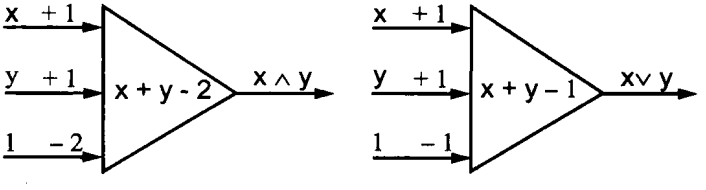


图 11-2　计算逻辑函数“与”和“或”的 McCulloch-Pitts 神经元模型

表 11-1　逻辑“与”的 McCulloch-Pitts 模型

x	y	x + y - 2	输出
1	1	0	1
1	0	-1	-1
0	1	-1	-1
0	0	-2	-1

尽管 McCulloch 和 Pitts 展示了神经计算的强大力量，但是直到一种可用的学习算法出现才使得神经计算得到蓬勃发展。早期的学习模型大都从心理学家 Hebb 的工作中得出，Hebb 认

为大脑通过改变神经元突触的连接强度进行学习，通过反复刺激这种突触来增加它的灵敏度，使得今后相似的刺激能得到有效响应。如果一类特殊的刺激反复激活一个细胞群，这些细胞将形成很强的相关性。在将来，相似的刺激会激活相同的神经元路径，从而识别出这种刺激（见 Hebb 的实际描述，11.5.1 节）。Hebb 的学习模型强调的只是增强路径，而忽视了对错误和矛盾的抑制、惩罚。现代心理学家试图实现 Hebb 模型，但是他们发现没有另外的抑制机制就得不出通用的结果（Rochester et al. 1988，Quinlan 1991）。我们将在 11.5 节讨论 Hebb 学习模型。

在下一节，将在神经网络中增加相互作用的中间层，扩展 McCulloch-Pitts 神经元模型，它最初的原型叫做*感知机*。

11.2　感知机学习

11.2.1　感知机训练算法

Frank Rosenblatt（1958，1962）设计了*感知机*学习算法，感知机是只有一层的神经网络。感知机的信号传播方法跟 McCulloch-Pitts 神经元模型相似（McCulloch-Pitts 神经元模型在 11.2.2 节还会介绍）。感知机的输入和输出值都是 1 或者 -1；权值是实数。其激励层就是计算权重的和，$\sum x_i w_i$。感知机的阈值函数是一个简单的严格限制阈值函数，当激励层的权值和大于阈值其输出就是 1，否则是 -1。假如输入是 x_i，权值是 w_i，阈值是 t，则感知机的输出值计算公式如下：

$$1 \quad \text{如果} \sum x_i w_i \geqslant t$$
$$-1 \quad \text{如果} \sum x_i w_i < t$$

感知机是一种简单的有监督学习。在试图解决一个问题实例后，教师会给出正确的答案。感知机然后改变他的权值，减少其误差。下面来看应用的规则。c 是一个常数，其大小表示学习率，d 表示期望的输出。于是对应第 i 个分量的权值的改变量 Δw_i 如下：

$$\Delta w_i = c(d - \text{sign}(\Sigma x_i w_i))\, x_i$$

$\Delta(\sum x_i w_i)$ 是感知机的输出值，其值取 +1 或者 -1。期望输出与实际输出的差就是 0、2 或者 -2。因此，对于权值向量的分量：

如果期望输出和实际输出相同，不改变权值；

如果实际输出是 -1 而期望的是 1，则对第 i 个分量增加 $2cx_i$；

如果实际输出是 1 而期望的是 -1，则对第 i 个分量减少 $2cx_i$。

这个过程将慢慢地减少对于整个训练集和的平均误差。如果存在这样一组权值，它使每一个训练元素都能得到正确的输出，则感知机的学习过程就能通过学习得到这样的值（Minsky and Papert 1969）。

感知机最开始得到了很好的发展，但是，Nils Nilsson（1965）和其他学者分析了感知机模型的局限。他们指出感知机不能解决一类叫非线性可分的问题。尽管在那个时候还提出了很多改进的模型，包括多层感知机，但是，Marvin Minsky 和 Seymour Papert 在《Perceptrons》（1969）一书中证明了这样一个结论：任何感知机模型都不可能解决非线性可分问题。

一个线性不可分的例子是*异或*。表 11-2 是表示异或的真值表。

考虑这样一个感知机，其输入为 x_1、x_2；权值为 w_1、w_2；阈值是 t。为了学习这个函数，这个神经网络必须找到这样一组值，它满足如下的不等式方程，见图 11-3：

$w_1 * 1 + w_2 * 1 < t$，真值表的第一行。

$w_1 * 1 + 0 > t$，真值表的第二行。

$0 + w_2 * 1 > t$，真值表的第三行。

$0 + 0 < t$ 或 t 为正数，最后一行。

表 11-2　异或真值表

x_1	x_2	输出
1	1	0
1	0	1
0	1	1
0	0	0

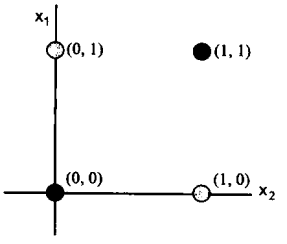


图 11-3　异或问题

注：在二维空间中没有可分离点集 {(0,0), (1,1)} 和 {(0,1), (1,0)} 的直线

这组关于 w_1、w_2 和 t 的不等式方程组没有解，这就证明了感知机不能解决异或问题。尽管多层神经网络将能够解决异或问题（见 11.3.3 节），但是感知机学习算法只能工作在单层神经网络上。

异或问题不能用感知机来解决的原因是待区分的类别不是线性可分的。这可从图 11-3 中看得很清楚。在二维空间中不可能画出这样的直线，这条直线把点集{(0,0),(1,1)}和{(0,1),(1,0)}分开。

可把输入网络的数据集当作一个空间，每一个输入值对应空间中的一维，整个输入就构成空间中的一点。如在异或的例子中，四个输入值构成了图 11-3 中的点，坐标用 x_1 和 x_2 标记。于是学习一个对于训练集的分类器的问题就转换成了一个把点集分离成两个组的问题。在 n 维空间中，如果两个类别可被一个 n－1 维的超平面分离，则这个分类是线性可分的（在二维空间中，n 维超平面是直线，三维空间中是一个平面，依此类推）。

因为线性可分的局限，很多学者转移到了基于符号的领域，于是减慢了连接方法学的发展。然而，20 世纪 80 年代和 90 年代的研究工作又表明这些问题是可解的（见 Ackley et al. 1985, Hinton and Sejnowski 1986, 1987）。

在 11.3 节，将讨论反传算法，它是感知机学习算法的扩展，工作于多层神经网络系统。在学习反传算法之前，我们先看一个感知机用于分类的例子。在本节的结尾我们定义了通用 delta 规则，它是感知机学习算法的泛化，在包括反传算法的很多神经网络中得到广泛的应用。

11.2.2　例子：用感知机网络进行分类

图 11-4 是分类问题的一个概述。从可能的问题空间中选择原始数据，然后变换成新的数据/模式空间。在这新的模式空间中，空间特征被确定，最后，对这些特征所表示的实体分类。例如在数字记录设备中声音的识别。通过设备，有关声音的信号转换成振幅和频率模型集合，最后，分类系统从这些特征中识别出某一个人的讲话。另外一个例子：通过医疗设备（如心脏除纤颤器）获取各种信息，这些模式空间中的特征就用于根据症状特征集断定可能的疾病类型。

在我们的分类例子中，图 11-4 所示的转换器和特征抽取器就是把问题领域信息转换成二维笛卡儿空间的模型。图 11-5 代表了在表 11-3 由感知机分析的信息的二性特征。表中前面两列是网络进行训练的样本数据，第三列表示类别 +1 或 −1，在学习训练中它一般起反馈作用。图 11-5 是问题训练数据的二维平面图，从中可看出，训练网络作用于全部输入数据后，一个线性可分

数据类就产生了。

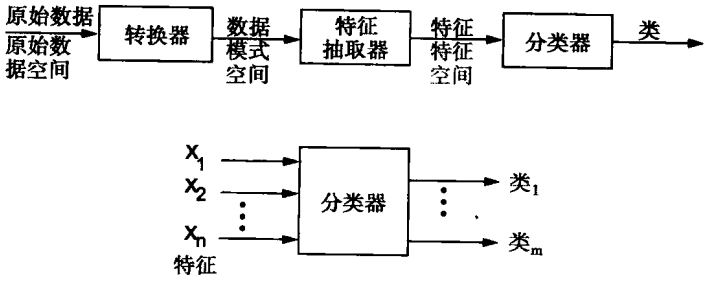


图 11-4　一个全面的分类系统

表 11-3　感知机分类的数据集

x_1	x_2	输出
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
7.9	8.4	-1
7.0	7.0	-1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	-1

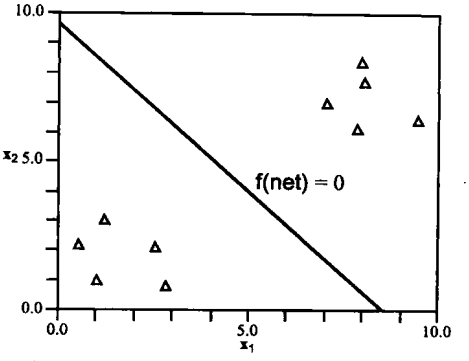


图 11-5　表 11-3 中数据点的二维平面图

注：11.2.1 节介绍的感知机提供了此数据集的线性分离

首先我们讨论一般的分类理论。每一类别确定的数据集合都可表示成多维空间中的一个区域。每一个类别 R_i 都有一个分类函数 g_i，在类别 R_i 所在的区域，分类函数 g_i 具有最大值：

$$g_i(x) > g_j(x) \text{ 对于所有的 } j, 1 < j < n$$

在表 11-3 所示的简单例子中，输入数据将在特征空间中产生两个明显的区域或者类别，一个用 1 表示，另一个用 -1 表示。

有一类重要的特殊的分类函数是判别函数，这种函数根据与区域中心点的距离评估类别关系。基于这种函数的分类叫做最小距离分类。一个简单的定理表明线性可分问题必有最小距离分类。

如果类别 R_i 和 R_j 相邻，就如图 11-5 中的两个区域，则存在一个明显的分界区域，在此处判别函数的值相等：

$$g_i(x) = g_j(x) \text{ 或 } g_i(x) - g_j(x) = 0$$

如果类别是线性可分的，如图 11-5，分离区域的判别函数是直线，或说 $g_i(x) - g_j(x)$ 是线性函数。既然直线是到两个固定点距离相等的点的集合，则判别函数 $g_i(x)\, g_j(x)$ 是最小距离函数，度量到笛卡儿坐标区域中心点的距离。

图 11-6 所示的感知机将计算这样的线性函数。它有两个输入参数，有一个偏置值，取值为常量 1。感知机计算：

$$f(net) = f(w_1 * x_1 + w_2 * x_2 + w_3 * 1),\ f(x) \text{ 是 } x \text{ 的符号函数}$$

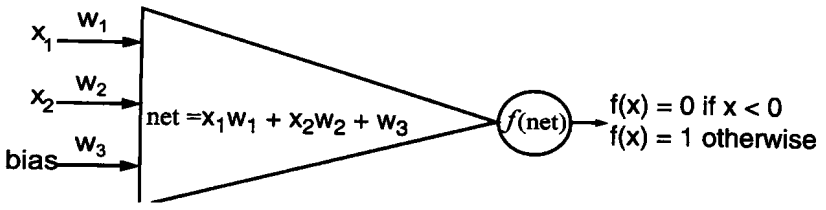


图 11-6　表 11-3 对应的感知机网络

注：阈值函数是线性的和双极的（见图 11-7a）

当 f(x)是 +1 时，x 被解释成一类，当是 -1 时，被解释成另一类。这种被划分为 +1 和 -1 的阈值称为线性双极阈值（见图 11-7a）。偏置值用于改变在水平轴上的阈值函数。在训练的过程中，这种改变量通过调整权值 w_3 进行学习得到。

现在用表 11-3 中的数据去训练图 11-6 所示的感知机。假定任意初始化权值向量为［0.75，0.5，-0.6］，采用 11.2.1 节介绍的感知机训练算法。上标（如 $f(net)^1$ 中的 1）训练数据表中数据的指针（或说数据表中的第几行）。我们从数据表中的第一行开始：

$$f(net)^1 = f(.75*1 + .5*1 - .6*1) = f(.65) = 1$$

既然 $f(net)^1 = 1$，是正确的输出，并不需要调整权值。因此，$W^2 = W^1$。对于第二组数据：

$$f(net)^2 = f(.75*9.4 + .5*6.4 - .6*1) = f(9.65) = 1$$

这次的期望输出是 -1，所以必须应用 11.1.1 节所述的学习规则：

$$W^t = W^{t-1} + c(d^{t-1} - sign(W^{t-1} * X^{t-1}))X^{t-1}$$

其中 c 是学习常数，X 和 W 是输入向量和权值向量，t 是网络的迭代次数。d^{t-1} 表示第 t-1 次的期望输出，在我们现在的情形中 t=2。感知机网络在 t=2 次的输出是 1。因此感知机网络的期望输出和实际输出之差 $d^2 - sign(W^2 * X^2)$ 是 -2。实际上，在强限制的双极感知机中，学习增量要么是 +2c，要么是 -2c。我们使学习常量 c 为一个很小的正数，取值为 0.2。修改权值向量为：

$$W^3 = W^2 + 0.2(-1-1)X^2 = \begin{bmatrix} 0.75 \\ 0.50 \\ -0.60 \end{bmatrix} - 0.4 \begin{bmatrix} 9.4 \\ 6.4 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix}$$

我们现在用刚得到的新权值考虑第三组数据：

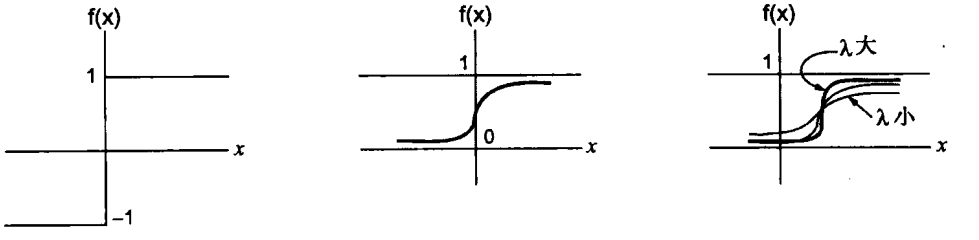
$$f(net)^3 = f(-3.01*2.5 - 2.06*2.1 - 1.0*1) = f(-12.84) = -1$$

同样，这次的输出和期望输出不一样。对 W^4 进行如下调整：

$$W^4 = W^3 + 0.2(1-(-1))X^3 = \begin{bmatrix} -3.01 \\ -2.06 \\ -1.00 \end{bmatrix} + 0.4 \begin{bmatrix} 2.5 \\ 2.1 \\ 1.0 \end{bmatrix} = \begin{bmatrix} -2.01 \\ -1.22 \\ -0.60 \end{bmatrix}$$

感知机网络经过 10 次训练就可得到图 11-5 所示的分割直线。感知机对数据集中的数据进行大约 500 次左右的训练后，权值向量将收敛成向量［-1.3，-1.1，10.9］。我们感兴趣的是两类线性分划问题。对于分类函数 g_i 和 g_j，直线定义为 $g_i(x) = g_j(x)$ 或 $g_i(x) - g_j(x) = 0$ 的点的集合，也就是说，在此处感知机网络的输出为 0。感知机输出公式如下：

输出 $= W_1X_1 + W_2X_2 + W_3$



a）强限制和双极性的线性阈值　　b）S形和单极性阈值　　c）S形、偏置和挤压的阈值。当λ变大时，S形阈值逼近线性阈值

图 11-7　阈值函数

因此，这两类分划直线定义为如下线性方程：

$$-1.3 * x_1 + -1.1 * x_2 + 10.9 = 0$$

11.2.3　通用 delta 规则

扩展感知机网络最直接的方法是用其他的激励函数代替强限制的阈值函数。例如，连续的激励函数提供更好的误差测量粒度，可提供更精确的学习算法。

图 11-7 显示了一些阈值函数的图形：图 11-7a 是线性双极阈值函数，它与感知机中使用的函数相似；还有两个 S 形函数（因为其形是一个“S”形曲线所以称之为 S 形函数），如图 11-7b 所示。一般的 S 形激励函数叫做 logistic 函数，其公式如下：

$$f(net) = 1/(1 + e^{-\lambda * net}), \text{ where net } = \Sigma x_i w_i$$

和前面定义的函数一样，x_i是第 i 行的输入，w_i是对应第 i 个输入的权值，λ 是一个“挤压参数”，它用于调节 S 形函数曲线。当 λ 取值增大时，S 形函数在区间[0, 1]就接近线性阈值函数；当它接近 1 时，S 形函数就接近一条直线。

这些阈值图绘制出输入值、神经元的激励水平，对照成比例的神经元的激励或输出。S 形激励函数是连续函数，它允许更精细的误差量度。像强限制阈值函数一样，S 形函数把域中的值映射到接近 0 或 1 的区域。然而，在 0 和 1 之间有一个快速和连续的转换区域。在一定意义上，当提供连续的输出时，S 形函数在行为上接近一个阈值函数。指数中的 λ 用于调节在转换区域中 S 形函数图形的坡度。加权偏差改变了沿 x 轴的阈值。

具有连续激励函数的神经网络的出现暗示了一种新的降低误差的学习方法的来临。Widrow-Hoff（1960）学习规则独立于激励函数，它总是力求减少在期望输出和网络实际输出（$net_i = WX_i$）之间的误差的平方。也许对于连续激励函数来说最重要的学习规则是 delta规则（Rumelhart et al. 1986a）。

直观地看，delta 规则基于误差曲面的思想，如图 11-8 所示。所谓误差曲面就是神经

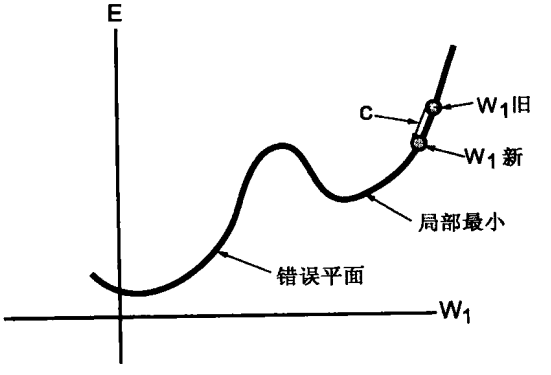


图 11-8　二维坐标中的误差曲面（在 n 维坐标中将更实际更形象化）

注：常数 c 指示了学习步幅的大小

网络权的函数在数据集上的累计误差。每一个神经网络权值向量都对应 n 维误差曲面中的一个点。给出一个权值向量，我们希望通过学习算法在这个曲面上找到一个方向，沿这个方向误差减少得最快。因为梯度是图形陡峭程度的度量，所以这种方法叫做*梯度下降学习*。

应用 delta 规则时，神经网络的激励函数必须是连续的和可微分的。刚才介绍的 logistic 公式具有这种性质。delta 学习规则调整第 i 个神经元结点的第 j 个输入权值的公式如下：

$$c\,(d_i - O_i)\,f'\,(net_i)\,x_j$$

其中 c 是控制学习率的常数，d_i 和 O_i 是第 i 个结点的期望输出和实际输出，第 i 个结点的激励函数的导数是 f′，x_j 是结点 i 的第 j 个输入。现在求这个公式的导数。

中间的平方神经网络误差是每一个结点误差的平方和：

$$\text{Error} = (1/2)\sum_i (d_i - O_i)^2$$

其中 d_i 是输出结点的期望输出，O_i 是结点的实际输出。因为单个的误差有可能取正值，也有可能取负值，为了不让它们的值相互抵消，对单个误差取平方。

这里考虑结点在输出层的情况；在 11.3 节介绍含隐含层的神经网络时将考虑一般的情况。首先，我们必须度量那个输出结点的网络误差的变化率，为此，我们应用*偏导数*的概念，在多变量的函数中，偏导数给出了某个特殊变量的变化率。对于总误差中结点 i 的偏导数是：

$$\frac{\partial \text{Error}}{\partial O_i} = \frac{\partial (1/2)*\Sigma (d_i - O_i)^2}{\partial O_i} = \frac{\partial (1/2)*(d_i - O_i)^2}{\partial O_i}$$

因为我们考虑的输出层中的结点的误差并不影响任何其他的结点，因此二次简化是可能的。根据偏导数的这个性质，可得到：

$$\frac{\partial (1/2)*(d_i - O_i)^2}{\partial O_i} = -(d_i - O_i)$$

我们所需要的是网络的误差与结点 i 的权值的变化量的函数。为了得到特定权值 w_k 的变化量，我们要用到偏导数，这次是取结点的误差对相应结点的权值 w_k 的偏导数。下面等式右边的展开是根据偏导数的链规则得到的：

$$\frac{\partial \text{Error}}{\partial w_k} = \frac{\partial \text{Error}}{\partial O_i} * \frac{\partial O_i}{\partial w_k}$$

这给了我们解这个方程所需要的各部分。根据前面的结果，可得到：

$$\frac{\partial \text{Error}}{\partial w_k} = -(d_i - O_i) * \frac{\partial O_i}{\partial w_k}$$

我们继续考虑右边的因素，偏导数是结点 i 的实际输出对相应结点的权值的偏导数。结点 i 的输出式是它的权值的函数，如下：

$$O_i = f(W_i X_i), \text{ where } W_i X_i = net_i$$

由于 f 是连续函数，对它求导数得到：

$$\frac{\partial O_i}{\partial w_k} = x_k * f'(W_i X_i) = f'(net_i) * x_k$$

代入前面的方程得到：

$$\frac{\partial Error}{\partial w_k} = -(d_i - O_i)f'(net_i)*x_k$$

误差最小化需要权值变化的方向是对应的梯度分量的负方向。因此有：

$$\Delta w_k = -c\frac{\partial Error}{\partial w_k} = -c[\ (d_i - O_i)*f'(net_i)*x_k] = c(d_i - O_i)f'(net_i)*x_k$$

显然，delta 规则类似于爬山法（见 4.1 节），在每一步中通过导数寻找在误差曲面中某个特定点局部区域的斜率，它总是应用这个斜率试图减小局部误差，因此，delta 学习不能区分误差空间中的全局最小点和局部最小点。

从对图 11-8 进一步的分析可知，学习常数 c 对 delta 学习规则的性能有很重要的影响。学习常数 c 决定了在单个学习事件中权值变化的快慢，c 的取值越大，则权值朝最优值移动的速度也越快。然而，如果 c 值取得太大，则算法有可能超过最优值或者在最优值附近震荡。如果 c 取值较小，这种可能性不大，但是它会使系统学习的速度不快。学习率的最优值有时加上一个动态因子（Zurada 1992），成为一个可随着应用变化而调整的参数。

尽管 delta 规则本身不能克服单层神经网络的局限，但是它的一般形式是反传算法的核心，反传算法是多层神经网络中的学习算法。这种算法将在下一节介绍。

11.3　反传学习

11.3.1　反传算法的起源

从上面两节可看到，单层神经网络局限于它们所能做的分类，在 11.3 节和 11.4 节表明多层神经网络能克服很多这样的局限。在 16.3 节可看到多层神经网络是完全可计算的，也就是说与图灵机类是等价的。然而，早期的研究人员不能设计出供多层神经网络使用的学习算法。我们这节介绍泛化的 delta 规则，它能解决这个问题。

多层神经网络（见图 11-9）中的神经元层与层之间是相互连接的，第 n 层的神经元只传递其刺激到第 n+1 层的神经元。多层信号处理意味着分散在网络中的误差可通过连续的网络层，以复杂的不可预测的方式传播和变化。因此，输出层的误差源的分析变得很复杂。反传算法可分配误差，然后调整相应的权值

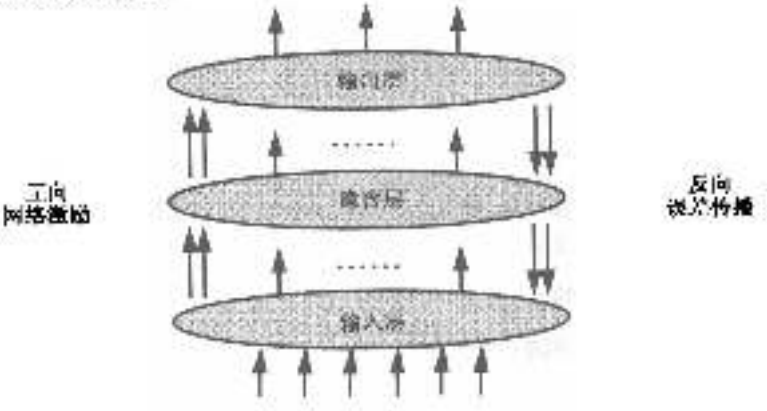


图 11-9　含隐含层的连接网络中的反传

反传算法采用的方法是从输出层开始，通过隐含层反向传播误差。当我们分析 delta 学习方法时，可看到更新一个神经元的权值所需要的信息局限于神经元的局部范围，误差总量除外。对于输出结点，计算期望输出和实际输出之差是比较容易的。对于隐含层的结点，决定一个结点引起的误差是比较困难的。反传算法的激励函数通常是 logistic 函数：

$$f(net) = 1/(1 + e^{-\lambda * net}), \text{其中 } net = \Sigma x_i w_i$$

用这个函数有四个理由：第一，它有 S 形状；第二，它是处处都可导的连续函数；第三，既然 S 形函数变化最快的地方导数值也最大，因此很多误差的分配都可归结于那些激励最不确定的结点；最后，导数很容易通过减法和乘法计算得到：

$$f(net) = 1/(1 + e^{-\lambda * net}) = \lambda(f(net) * (1 - f(net)))$$

反传训练使用泛化的 delta 规则。同样的使用 11.2 节介绍的梯度下降法。对于隐含层的结点我们考察其对输出层误差的贡献。计算权值 w_{ki}（表示从结点 k 到结点 i 的连接的权重）的调整量的公式如下：

1) $\Delta w_{ki} = -c(d_i - O_i) * O_i (1 - O_i) x_k$, 输出层结点

2) $\Delta w_{ki} = -c * O_i (1 - O_i) \sum_j (-delta_j * w_{ij}) x_k$, 隐含层的结点

在 2）式中，j 是结点 i 输出信号所到达的下一层的结点的标号，还有：

$$delta_j = -\frac{\partial Error}{\partial net_j} = (d_i - O_i) * O_i (1 - O_i)$$

我们现在求这些公式的导数。首先导出 1）式，输出层结点权值调整量的公式。和前面的相同，我们所需要的是网络误差的改变率与结点 i 的第 k 个权值 w_k 的函数。在 11.2.3 节中对 delta 规则求导时处理了这种情况，公式如下：

$$\frac{\partial Error}{\partial w_k} = -((d_i - O_i) * f'(net_i) * x_k)$$

由于 f 可以是任意函数，我们这里取 logistic 激励函数，导数如下：

$$f'(net) = f'(1/(1 + e^{-\lambda * net})) = f(net) * (1 - f(net))$$

还有 $f(net_i)$ 等于 O_i，代入上面的方程得到：

$$\frac{\partial Error}{\partial w_k} = -(d_i - O_i) * O_i * (1 - O_i) * x_k$$

既然误差的最小化要求权值变化的方向是梯度向量的负方向，我们乘以常数 - c 得到输出层第 i 个结点的权值调整量：

$$\Delta w_k = c(d_i - O_i) * O_i * (1 - O_i) * x_k$$

下一步我们推导隐含层结点的权值调整量。为了简单起见假设只有一个隐含层。我们考虑隐含层中的结点 i，分析它对整个网络误差的贡献。首先，我们考虑结点 i 对输出层结点 j 的误差的贡献，然后对所有输出层结点误差贡献求和。最后，我们考察结点 i 的第 k 个输入权值对网络误差的贡献。图 11-10 表明了这种情况。

我们首先计算网络误差对隐含层结点 i 的输出的偏导数，应用链式法则可得：

$$\frac{\partial Error}{\partial O_i} = \frac{\partial Error}{\partial net_j} * \frac{\partial net_j}{\partial O_i}$$

等式右边第一项 $(\partial Error)/(\partial net_j)$ 的相反数是 $delta_j$，因此，等式可化为：

$$\frac{\partial Error}{\partial O_i} = -delta_j * \frac{\partial net_j}{\partial O_i}$$

输出层结点 j 的输入刺激 net_j 是隐含层中所有结点的输出与权值乘积的总和：

$$net_j = \sum_i w_{ij} O_i$$

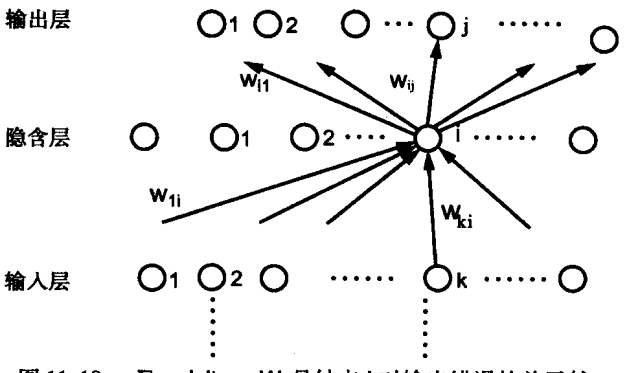


图 11-10　$\sum_j -delta_j * W_{ij}$ 是结点 i 对输出错误的总贡献

注：导数表示了 w_{ki} 的调整量

由于偏导数只考虑总和中的一个分量，也就是结点 i 和结点 j 的连接权重，可得：

$$\frac{\partial net_j}{\partial O_i} = w_{ij}$$

此处的 w_{ij} 是隐含层的结点 i 到输出层结点 j 的连接权值，变量代换得：

$$\frac{\partial Error}{\partial O_i} = -delta_j * w_{ij}$$

现在，总结所有结点 i 到输出层结点的连接：

$$\frac{\partial Error}{\partial O_i} = \sum_j -delta_j * w_{ij}$$

这就表示网络误差对隐含层结点 i 的灵敏度。下一步将确定 $delta_i$，网络误差对隐含层结点 i 的激励函数的灵敏度。这就给出了网络误差对隐含层结点 i 的输入权值的灵敏度。再用链式法则可得：

$$-delta_i = \frac{\partial Error}{\partial net_i} = \frac{\partial Error}{\partial O_i} * \frac{\partial O_i}{\partial net_i}$$

由于我们用 logistic 激励函数，

$$\frac{\partial O_i}{\partial net_i} = O_i * (1 - O_i)$$

把这个值代入 $delta_i$ 的方程，得到：

$$-delta_i = O_i * (1 - O_i) * \sum_j -delta_j * w_{ij}$$

最后，计算输出层结点的网络误差对隐含层结点输入权值的灵敏度。考察结点 i 的第 k 个输入权值 w_k，由链式法则得：

$$\frac{\partial Error}{\partial w_{ki}} = \frac{\partial Error}{\partial net_i} * \frac{\partial net_i}{\partial w_{ki}} = -delta_i * \frac{\partial net_i}{\partial w_{ki}} = -delta_i * x_k$$

其中 x_k 是结点 i 的第 k 个输入。

代入 $-delta_i$ 的值可得到：

$$\frac{\partial Error}{\partial w_{ki}} = O_i(1-O_i)\sum_j(-delta_j * w_{ij})x_k$$

由于误差最小化要求权值变化的方式为梯度向量的负方向，对结点 i 的第 k 个权值的调整量乘以一个负的学习常量：

$$\Delta w_{ki} = -c\frac{\partial Error}{\partial w_{ki}} = c * O_i(1-O_i)\sum_j(delta_j * w_{ij})x_k$$

对于超过一个隐含层的神经网络，同样的过程递归调用把误差从第 n 层传递到第 n－1 层。

尽管反传算法对多层神经网络的学习问题提供了解决办法，但是，它也有其自身的缺点。因为它是爬山法，有可能收敛于局部最小值（如图 11-8 所示），最后，反传算法计算的代价很高，尤其是当网络收敛很慢时。

11.3.2　反传算法实例 1：NETtalk

NETtalk 是用神经网络解决困难学习问题的有趣的例子（Sejnowski and Rosenberg 1987）。NETtalk 学习朗读英语文本。因为英语发音是非常不规则的，对于符号系统（例如基于规则的系统）这是一个很难的问题。

NETtalk 学习朗读一个文本字符串，返回一个音素和相关的字符串中每一个字母的重音。音素是一种语言发音中的基本元素；重音是相关的声音的大小。因为单个字母的发音与上下文和它周围的字母有关，NETtalk 使用一个有 7 个字母的窗口。当文本经过窗口时，NETtalk 返回每一个字母的音素/重音对。

图 11-11 是 NETtalk 的结构。网络有三层，输入单元对应文本窗口中的 7 个字母。窗口中的每一个位置用 29 个输入单元表示，一个单元表示字母表中的字母，3 个单元表示标点和空格。每一个位置的字母激活相应的结点。输出单元用人类发音的 21 个特征对音素进行编码。剩下的 5 个单元对重音和音素边界编码。NETtalk 有 80 个隐含单元，26 个输出值，以及 18 629 个连接。

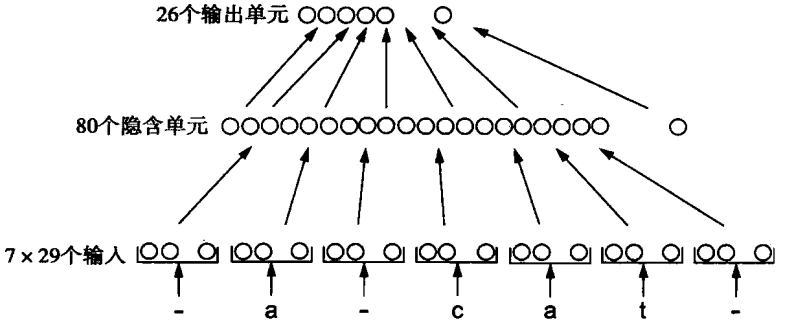


图 11-11　NETtalk 的网络拓扑图

对 NETtalk 进行训练时，给定一个 7 个字母的窗口，学习尝试对中间的字母发音。然后拿它跟正确的发音比较，用反传算法调整网络的权值。

这个例子显示了神经网络的很多特性，其中一些特性反映了人类学习的本质。例如，当我们用正确反应的百分比度量时，学习首先速度很快，然后当正确反应率增大时，学习速度变慢。与

人一样，网络学习发音的单词越多，它对陌生单词的反应就越正确。有这样的实验：对充分训练的神经网络的权值进行随机改变，实验表明当权值改变时，网络将有抗破坏性，将慢慢退化。研究者也发现受损网络的重新学习是高效的。

第二个很有意思的特点是多层神经网络中隐含层的角色。任何学习算法必须学习泛化，也就是能应用于问题领域中未知的实例。隐含层在神经网络的泛化中起了很重要的作用。跟其他的反传网络一样，NETtalk 的隐含层比输入层的神经元数少。这意味着在模式的训练中隐含层中很少的结点用于信息编码，并且其中有一些抽象过程发生。这种很少的编码方式表明，在输入层不同的模式可映射到隐含层中的同一个模式。这种简化就是泛化。

NETtalk 的学习是有效的，尽管它需要大量的训练实例。反传算法和 ID3 算法的一系列的试验比较表明（Shavlik et al. 1991），尽管两者训练和使用数据的方法不同，但是其结果是一样的。ID3（见 9.3 节）和 NETtalk 在经过 500 个例子的训练后，能对训练数据中的 60% 进行正确的发音。但是，ID3 对训练数据只需要使用一次，而 NETtalk 却需要多次反复使用训练数据。在这个研究中，NETtalk 对训练数据使用了 100 次。

我们的例子表明，连接和符号学习之间的关系比我们开始看到的复杂得多。在下一个例子中，我们用反传算法解决异或问题。

11.3.3　反传算法实例 2：异或

本节我们提出一个简单的隐含层来解决异或问题。如图 11-12，该网络有两个输入结点、一个隐含结点和一个输出结点。该网络还有两个偏置结点：第一个到隐含层结点，第二个到输出层结点。网络中隐含层结点和输出层结点的值计算方法如通常所用方法，即输入数据与其训练权值的向量乘积。偏置值加入总和中。权值用反传算法进行训练，激励函数是 S 形函数。

需要指出的是：输入结点连同已训练结点也和输出结点直接相连。这种额外的连接通常可使网络隐含层所需结点更少，收敛速度更快。实际上，图 11-12 所示网络没有任何特殊；还有很多其他的网络可用于解决异或问题。

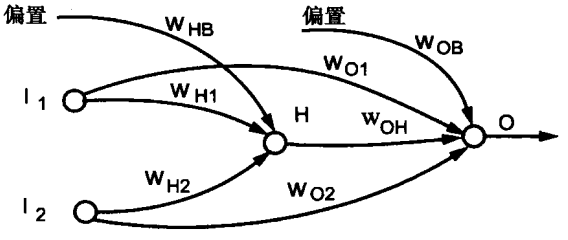


图 11-12　反传网络解决异或问题

注：w_{ij}是权值，H 是隐含结点

先任意初始化网络权值，然后用异或真值表中所示的模式训练该网络：

(0, 0) → 0; (1, 0) → 1; (0, 1) → 1; (1, 1) → 0

用上面 4 个实例经过 1400 次训练循环产生下面的权值，其中最接近的 10 个，即图 11-12 中的加权参数：

$W_{H1} = -7.0$　$W_{HB} = 2.6$　$W_{O1} = -5.0$　$W_{OH} = -11.0$
$W_{H2} = -7.0$　$W_{OB} = 7.0$　$W_{O2} = -4.0$

输入数据（0,0），隐含层结点的输出是：

$$f(0*(-7.0) + 0*(-7.0) + 1*2.6) = f(2.6) \rightarrow 1$$

对于（0,0），输出层结点的输出是：

$$f(0*(-5.0) + 0*(-4.0) + 1*(-11.0) + 1*(7.0)) = f(-4.0) \rightarrow 0$$

对于（1,0），隐含层的输出是：

$$f(1*(-7.0) + 0*(-7.0) + 1*2.6) = f(-4.4) \rightarrow 0$$

对于（1,0），输出层结点的输出是：

$$f(1*(-5.0) + 0*(-4.0) + 0*(-11.0) + 1*(7.0)) = f(2.0) \rightarrow 1$$

输入值（0,1）是相似的。最后，我们用输入值（1,1）检验异或神经网络，其隐含结点的输出是：

$$f(1*(-7.0) + 1*(-7.0) + 1*2.6) = f(-11.4) \rightarrow 0$$

输出层结点的输出是：

$$f(1*(-5.0) + 1*(-4.0) + 0*(-11.0) + 1*(7.0)) = f(-2.0) \rightarrow 0$$

读者到此就可看到，反传算法的反馈神经网络可解决非线性可分的问题。激励函数是 S 形函数（图 11-7b），学习到的偏置值使它在 x 轴上极轻微地向正方向转变。

下面思考竞争学习模型。

11.4　竞争学习

11.4.1　对于分类的“胜者全拿”学习

“胜者全拿”算法（Kohonen 1984，Hecht-Nielsen 1987）作用于一层结点中单个的结点，而这一结点对输入模式有强烈的反应。“胜者全拿”可被看成是一个网络结点集合中的竞争，如图 11-13。在这个图中，输入向量值 $X = (x_1, x_2, \cdots, x_m)$ 输入网络中包含结点 A, B, …, N 的网络层。该图显示结点 B 是竞争获胜者，其输出是信号 1。

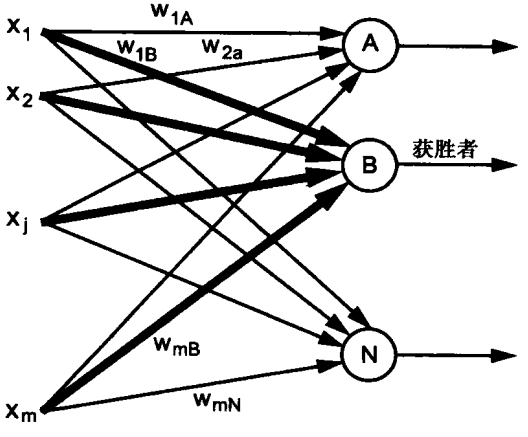


图 11-13　应用“胜者全拿”算法的网络层
注：旧输入向量支持获胜结点

“胜者全拿”是无监督学习，获胜者是通过“最大响应”测试决定的。获胜者的权值向量将得到奖励：它的每一个分量都向输入向量靠近。对于获胜者的权值向量 W 和输入向量 X，其增量为：

$$\Delta W^t = c(X^{t-1} - W^{t-1})$$

其中 c 是学习常量，取一个很小的正值，并且其值随着学习过程减少。获胜者的权值向量于是通过增加 ΔW^t 来调节。

获胜结点权值向量的每一分量的增加量或减少量取决于 $x_i - w_i$ 的值。当然，其结果是使获胜结点更好匹配输入向量。“胜者全拿”算法并不需要直接计算激励函数的值以找出最大响应结点。对于结点 i（规范化权值向量为 W_i）和输入向量 X，激励值 W_iX 是 W_i 和输入模式 X 之间的

欧拉距离的函数。这通过计算欧拉距离可以看出：

$$\|X - W_i\| = \sqrt{(X - W_i)^2} = \sqrt{X^2 - 2XW_i + W_i^2}$$

从这个等式可看出，对一组规范化的权值向量，如果一个权值向量和 X 的欧拉距离 $\|X - W\|$ 最小，则它有最大的激励值 WX。在很多情况下，对于规范化的权值向量，通过计算欧拉距离比通过比较激励水平来决定获胜者更有效率一些。

我们认可“胜者全拿”Kohonen 学习规则有很多理由。第一，我们把它当作分类方法，用它与感知机分类比较。第二，可用它与其他的网络体系结构组合，形成更加复杂的学习模型。我们将看到 Kohonen 学习原型与 outstar（一种有监督的学习网络）的组合模型。这种混合物叫做*逆传网络*，首先由 Robert Hecht-Nielsen（1987，1990）提出。在 11.4.3 节，将看到怎样用逆传描述有条件的学习。

在这个介绍之前，还有很多对于“胜者全拿”算法重要的论点。为了防止某个结点总是不能成为获胜结点，有时在循环中设置和更新“意识”参数。这样可保证网络中的所有结点可最终参与表示模式空间。在一些算法中，不是一个结点获胜，而是一组结点都获胜，然后它们的权值都将得到不同程度的增量。另外一种方法是对获胜结点周围的结点给予不同回报。权值开始被任意地初始化，然后在学习中逐步规范化（Zurada 1992）。Hecht-Nielsen（1990）表明了“胜者全拿”算法与数据集的 K－均值分析方法有多大的相似性。下面将介绍用于聚类的无监督 Kohonen“胜者全拿”算法。

11.4.2　学习原型的 Kohonen 网络

数据分类和原型在学习中的角色问题一直是心理学家、语言学家、计算机科学家和认知科学家关心的对象（Wittgenstein 1953，Rosch 1978，Lakoff 1987）。人工智能中的分类和原型也是本书的重要内容。在 9.5 节中，介绍了基于符号系统的分类方法和概率聚类算法。在连接系统中，在 11.2 节介绍了基于感知机分类算法，现在介绍 Kohonen（1984）的“胜者全拿”聚类算法。

图 11-14 再次给出了表 11-3 中的数据点。将网络训练中产生的一系列原型叠加在这些数据点上。经过多次迭代，感知机训练算法收敛，形成一组网络连接权值的配置，它相当于在两类之间定义了一个线性分类器。我们可看出，用权值定义的直线也可通过隐式地计算每一个聚类的欧几里得距离“中心”得到。这个聚类中心在感知机分类中起类别原型的作用。

另一方面，Kohonen 学习是无监督的，开始有很多任意创建的原型，然后不断训练，直到它明白无误地表示一个数据集。在学习过程中，学习常数慢慢减少，因此输入向量不会在原型中引起较大的混乱。

像 CLUSTER/2 一样，Kohonen 学习有一个强的归纳偏置，因为需要的原型的数目在算法开始时必须明确定义。这就允许网络算法设计者定义一个合适的数目来表达数据群集的类别。逆传算

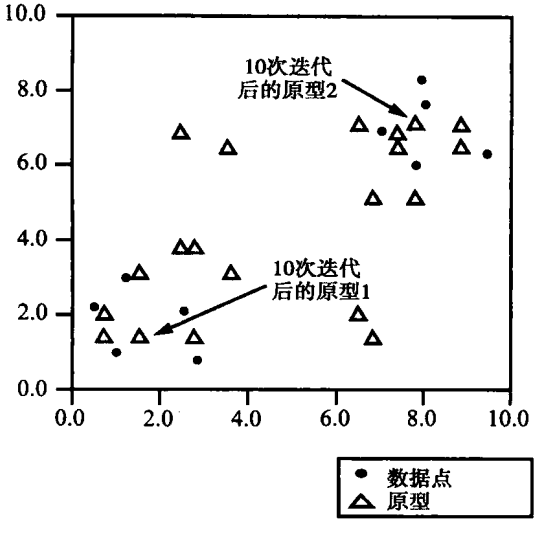


图 11-14　应用无监督的 Kohonen 算法产生可以代表表 11-3 中数据类别的一系列原型

法（见 11.4.3 节）允许对选择的原型数目数进行更多的操作。

图 11-15 是一个对表 11-3 中数据进行分类的 Kohonen 学习网络。数据在二维笛卡儿空间中表示，因此表示数据群集的原型也应该是一个有序数据对。我们选择两个原型，一个表示一个数据群集。任意初始化结点 A 为（7,2），结点 B 为（2,9）。任意初始化一般只用于简单问题；替代的方法是把权值设置成等于数据群集中有代表性的点。

获胜结点将有与输入向量更接近的权值。这个获胜结点的权值向量将向输入向量接近，同时没有获胜的结点的权值将不会改变。既然我们能确切计算出输入向量和每个原型之间的欧拉距离，就不需要对向量进行 11.4.1 节所述的规范化。

因为根据数据点和各原型之间的简单距离测度来决定获胜者的选择，所以 Kohonen 学习是无监督的。类别将在自组织网络中发现。尽管 Kohonen 学习任意选择数据点进行分析，我们这里选择表 11-3 中数据时按从上往下的顺序选取。对点（1,1），我们测量到每个原型的距离：

$$\|(1,1)-(7,2)\| = (1-7)^2 + (1-2)^2 = 37$$
$$\|(1,1)-(2,9)\| = (1-2)^2 + (1-9)^2 = 65$$

因为结点 A(7,2) 更接近点(1,1)，点 A 是获胜结点。$\|(1,1)-(7,2)\|$ 表示了这两个点之间的距离。因为大小关系是不变的，在计算欧拉距离时并不需要进行开平方。我们现在奖赏获胜结点，学习常数为 0.5。对于第二次迭代：

$$\begin{aligned} W^2 &= W^1 + c(X^1 - W^1) \\ &= (7,2) + 0.5((1,1)-(7,2)) = (7,2) + 0.5((1-7),(1-2)) \\ &= (7,2) + (-3,-0.5) = (4,1.5) \end{aligned}$$

在第二次学习算法迭代时，对于数据点（9.4,6.4）：

$$\|(9.4,6.4)-(4,1.5)\| = (9.4-4)^2 + (6.4-1.5)^2 = 53.17$$
$$\|(9.4,6.4)-(2,9)\| = (9.4-2)^2 + (6.4-9)^2 = 60.15$$

点 A 又是获胜结点。对于第三次迭代的权值变成：

$$\begin{aligned} W^3 &= W^2 + c(X^2 - W^2) \\ &= (4,1.5) + 0.5((9.4,6.4)-(4,1.5)) \\ &= (4,1.5) + (2.7,2.5) = (6.7,4) \end{aligned}$$

在第三次迭代时，对于数据点（2.5,2.1）：

$$\|(2.5,2.1)-(6.7,4)\| = (2.5-6.7)^2 + (2.1-4)^2 = 21.25$$
$$\|(2.5,2.1)-(2,9)\| = (2.5-2)^2 + (2.1-9)^2 = 47.86$$

点 A 还是获胜结点，我们继续计算它的新的权值向量。图 11-14 显示了今后 10 次迭代后的进化情况。用于产生图 11-14 的算法是任意从表 11-3 中选择数据，因此显示的原型将不同于刚才我们产生的。原型的慢慢的进步可看成是朝数据群集的中心移动。强调一下，“胜者全拿”算法是无监督学习方法。它产生一系列演变和明确的表示数据群集的原型。很多研究者（包括 Zurada（1992）和 Hecht-Nielsen（1990））

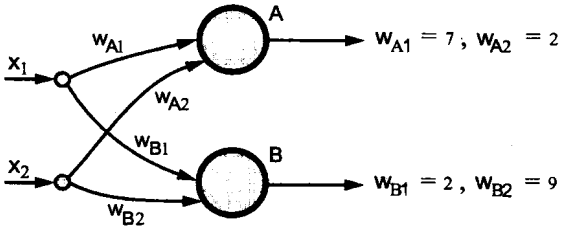


图 11-15　对应于表 11-3 中数据和图 11-14 中分类器的 Kohonen 学习网络的体系结构图

指出 Kohonen 无监督分类器从根本上说同 k－均值分析是一样的。

下一节学习 Grossberg 或说 outstar 算法，它是 Kohonen“胜者全拿”算法的扩展，它将会扩大我们选择原型的能力。

11.4.3　outstar 网络和逆传

到这一节为止，我们考虑了输入数据的无监督聚类。这里的学习需要问题领域的先验知识。逐步发现的数据特征和训练的历史记录将有助于类别的鉴定，发现类别之间的边界。一旦数据点按照向量表示的相似性进行了聚类，教师就可辅助进行校正，或者给每个数据类别相应的名字。这可由有监督的训练来完成，把“胜者全拿”网络的输出层的结点当作另外一个网络层的输入结点，然后在输出层明确地加强这种决定。

有监督的训练和加强的输出可让我们把 Kohonen 网络的结果映射到特定的输出模式和类别。Grossberg（1982，1988）层实现了叫做 outstar 的算法，它可完成上面的工作。由 Kohonen 网络和 Grossberg 网络组合而成的网络叫做*逆传网络*，它首先由 Robert Hecht-Nielsen（1987，1990）提出。

在 11.4.2 节，我们学习了 Kohonen 网络层的具体细节，这里开始学习 Grossberg 网络层。图 11-16 显示了一个网络层，有结点 A, B, …, N，其中结点 J 是被选择的获胜结点。向量 Y 是教师的反馈，我们通过反馈希望加强结点 J 到输出层结点 I 的连接权重，结点 I 是假设要被激活的，可见 Grossberg 是有监督的学习。通过 outstar 学习，我们确认和加强在外向连接 J 到 I 上的权值 w_{JI}。

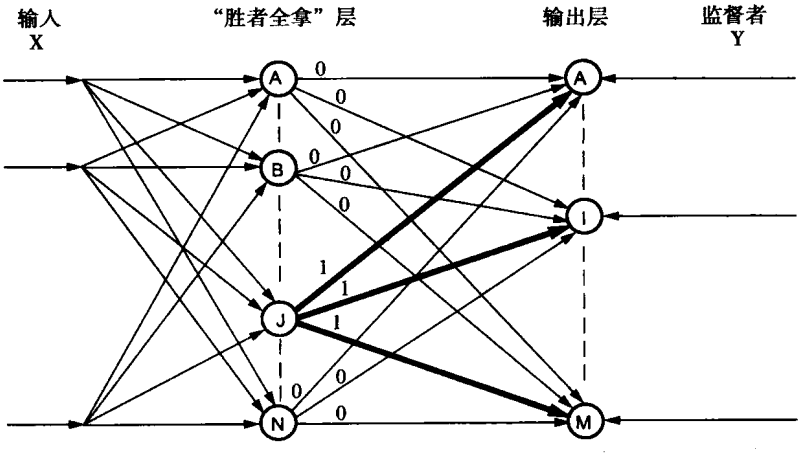


图 11-16　“outstar”结点 J，也是“胜者全拿”网络中的获胜者

注：Y 向量监督 Grossberg 训练中输出层的响应。“outstar”结点的连接是加粗的，权值为 1，其他的权值为 0

为了训练逆传网络，我们必须先训练 Kohonen 网络层。当一个获胜者找出时，从这个结点出去的所有连线的权值都赋值为 1，其他结点的连线为 0。这个结点和与之相连的输出层的所有结点形成一个“outstar”（见图 11-16）。训练 Grossberg 网络层是基于 outstar 分量。

如果输入向量的每一个聚类都表示同一个类别，则类别中的所有成员可都映射到输出层的同一个值，且不需要一个迭代的训练。只需要决定“胜者全拿”Kohonen 网络中的哪一个结点连向哪一个输出层的类别，然后根据类别和期望输出的关系在这些结点之间赋予适当的权值。例如，如果“胜者全拿”网络层中第 J 个单元是获胜者，I = 1 是期望的网络输出，那么就设置 $w_{JI} = 1$，J-outstar的所有其余连接权重 $w_{JK} = 0$。

如果聚类元素的期望输出改变了，就需要利用监督向量 Y 进行一次迭代过程处理，调整 outstar 的权值。这种训练的结果是平均某特定群集中的元素的期望输出。按照以下方程，我们训练 outstar 中获胜结点到输出结点的连接权重：

$$W^{t+1} = W^t + c(Y - W^t)$$

其中 c 是学习常量，取值为小正数。W^t 是 outstar 中的权值向量，Y 是期望输出向量。注意，这种算法有增强权值的效果：如果输出层的结点 I 是获胜结点，即输出为 1，而 Kohonen 层中结点 J 的期望输出也是 1，则可加强结点 I 和结点 J 的连接权值。这是 Hebbian 学习规则的一个实例，Hebbian 规则描述的是当一个结点对另外一个结点激活做出贡献时，则这条神经路径将得到加强。将在 11.5 节具体讨论 Hebbian 规则。

下面应用这一规则训练逆传网络去识别表 11-3 中的类别。我们也将用这个例子阐述逆传网络怎样实现有条件学习。假设表 11-3 中的 x_1 表示一动力系统中发动机的速度，x_2 表示发动机的温度。系统的速度和温度都校正到区间 [0,10] 中的点。我们的监控系统以一定的间隔抽取样本点。当速度和温度都超高时，我们希望能广播警告。对表 11-3 的输出重命名，+1 改为“safe”，-1 改为“dangerous”。我们设计的逆传网络如图 11-17。

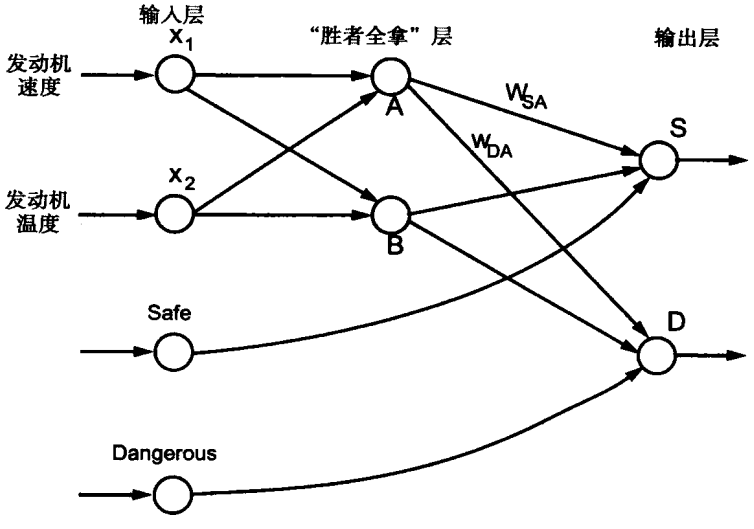


图 11-17　识别表 11-13 中类别的逆传网络。训练结点 A 的 outstar 权值 w_{SA} 和 w_{DA}

既然知道 Kohonen 层中的获胜结点映射到 Grossberg 网络输出层的结点需要什么样的值，可直接设置这样的值。然而，为了阐述 outstar 学习，我们将用上面给出的公式训练网络。如果（任意地）假定输出层中的结点 S 是 safe 的标志，结点 D 是 dangerous 的标志，对于 Kohonen 网络输出层结点 A，其 outstar 权值向量必须为 [1,0]，结点 B 的 outstar 权值向量为 [0,1]。因为对称关系，我们只讨论结点 A 的训练。

在 Grossberg 网络训练之前，Kohonen 网络必须是稳定的。我们在 11.4.2 节讨论了 Kohonen 的收敛。训练 outstar 结点 A 的输入向量形式是[x_1, x_2, 1, 0]，x_1和 x_2是表 11-3 中的值，这些数据将在 Kohonen 输出层结点 A 聚类，后面两个分量表明当 A 是 Kohonen 获胜者时，safe 取值为“true”，dangerous 取值为“false”，如图 11-15。我们初始化结点 A 的 outstar 权值为 [0,0]，0.2 是学习常量。

$W^1 = [0, 0] + 0.2[[1, 0] - [0, 0]] = [0, 0] + [0.2, 0] = [0.2, 0]$
$W^2 = [0.2, 0] + 0.2[[1, 0] - [0.2, 0]] = [0.2, 0] + [0.16, 0] = [0.36, 0]$
$W^3 = [0.36, 0] + 0.2[[1, 0] - [0.36, 0]] = [0.36, 0] + [0.13, 0] = [0.49, 0]$
$W^4 = [0.49, 0] + 0.2[[1, 0] - [0.49, 0]] = [0.49, 0] + [0.10, 0] = [0.59, 0]$
$W^5 = [0.59, 0] + 0.2[[1, 0] - [0.59, 0]] = [0.59, 0] + [0.08, 0] = [0.67, 0]$

从上面可看出，不断训练，权值慢慢向［1,0］移动。当然，既然在这种情况下与结点 A 相关的聚类元素通常映射到［1,0］，我们可采用简单赋值的算法，而不用平均训练算法。

现在来看逆传网络怎样做出适当的反应。当我们首先从表 11-3 中输入第一个向量，应用于图 11-17 的逆传网络，结点 A 的 outstar 权值的激化结果是［1,1］，而结点 B 的 outstar 权值的激化结果是［0,0］。激化结果点和输出层结点 S 的权值是［1,0］×［1,0］，把激化结果 1 送到输出结点 S。结点 B 的 outstar 权值是［0,1］，结点 D 的激化结果是［1,0］×［0,1］，这是我们所希望的值。对表 11-3 中的第二行数据进行测试，在“胜者全拿”层从结点 A 得到激化结果［0,0］，从结点 B 得到结果［1,1］，这些值的结果点和训练的权值赋 S 结点为 0，结点 D 为 1，这也是我们所要求的值。读者可继续测试表 11-3 中的其他值。

从认知的角度看，我们可对逆传网络做出关联解释。重新考虑图 11-17，因为网络是学习事件模式，Kohonen 层的学习可认为是获得条件刺激。另一方面，Grossberg 层的学习是结点（非条件刺激物）和某些反应的关联。在我们的情形中，当数据适合一定条件时，系统学习广播一个危险的警告。一旦学会做出适当的反应，即使没有教师的监督，系统也能对新的数据做出适当的反应。

对逆传的第二种认知解释是当作记忆中对现象模式的关联进行加强的形式。这有点像为数据模式的响应建立字典表。

在某些情形下，逆传算法和反传算法相比有很多的优点。像反传算法一样，逆传算法也能解决非线性可分问题。然而，逆传是通过 Kohonen 层中进行的预处理实现的，在 Kohonen 层中数据集被分划到具有相同性质的集合中。因为用这种把数据分划到离散集合的方法代替反传网络中在隐含层进行广义搜索的办法，所以逆传算法与反传算法相比有更大的优越性。

11.4.4　支持向量机

支持向量机（Support Vector Machine，SVM）提供了另一个竞争学习的例子。在支持向量机中，使用统计方法来确定能最大区分要学习的概念的正例和反例的最小数据点（即支持向量）集合。这些支持向量代表从概念的正例和反例中选出的数据点，隐含定义了一个区分这两个数据集的超平面。例如，运行 SVM 算法，将点(2.5,2.1)和(1.2,3.0)作为表 11-3 和图 11-5 中数据的正例的支持向量，将点(7.0,7.0)和(7.8,6.1)作为反例的支持向量。一旦支持向量被学习了，那么其他数据点就不需要保留了，支持向量自己就足以确定分割超平面。支持向量机是线性分类器，对支持向量的学习是有监督的。供 SVM 学习的数据假定是从一个确定的（虽然未知）数据分布上独立地用相同方式产生的。支持向量隐含定义的超平面将正例和反例区分开。靠近超平面的数据点位于*决策范围*（decision margin）（Burges 1998）中。任何支持向量的加入或删除都将改变超平面的边缘。前面曾提到，训练结束后，可以从支持向量重建超平面并分类新的数据集。

SVM 算法通过计算数据点到分割超平面的距离来分类数据元素，类似于优化问题。要想成功地控制特征空间（即要学习的实例的参数）不断增加的灵活性，需要一个复杂的归纳理论。这一理论必须能精确描述一定要控制的特征，才能形成一个好的归纳。在统计学中，这一问题称为*一致收敛率*（the rates of uniform convergence）研究。我们在 10.4.2 节曾经看过一个例子，介绍了可能近似正确（PAC）学习模型。PAC 学习的结果可以看作是为样本的数量建立一个范围，以保证某个特定的误差界限。为完成归纳，可以利用贝叶斯或其他数据压缩技术。SVM 学习中

经常使用 Vapnik-Chervonenkis 理论。

Vapnik-Chervonenkis（VC）维数定义为能够用一些函数对分到两类中的训练点的最大数量（Burges 1998）。这样 VC 理论就在相容假设泛化（Cristianini and Shawe-Taylor 2000）上提供了一个自由分布范围。SVM 算法用 VC 理论来计算超平面并控制误差的范围，以提高归纳的精确性，有时也称为函数的容量（capacity）。

SVM 算法使用类似于点积的方法从特征空间映射数据。点积的结果表示被映射的向量用解二次方程得到的权值进行线性组合（Scholkopf et al. 1998）。一个核心函数用来建立特征向量映射，如多项式、样条函数或高斯函数，选择哪个核心函数主问题分布决定。SVM 算法计算距离来确定数据元素的分类。SVM 生成的决策规则表示的是数据的统计规律。一旦 SVM 训练好，新数据点的分类就成为简单的与支持向量比较的问题。在支持向量中，刻画要学习的概念的关键特征集中在超平面的一边，描述相反特征的集中在另一边，没有这种区分的特征没有用到。

对于 11.2 节中的感知机算法，数据的线性可分离性很重要。如果数据不可分，那么算法无法收敛。相反，SVM 试图最大化决策间隔，处理由重叠数据点造成的难以区分的情况的能力更加健壮。还可以引进松弛（slack）变量来放宽线性约束，以找到一个更柔性的间隔，松弛变量的值指示分类边界的信心水平（Cristianini and Shawe-Taylor 2000）。结果，为了生成超平面，一些突在外面的支持向量可能被错误分类：相应的，数据很嘈杂时，决策间隔会变窄。

SVM 能够从两个分类的分类问题推广到多分类的辨别问题，方法是多次重复运行 SVM 算法，每次将关心的分类作为一个分类，所有其他分类作为另一个分类。SVM 算法最适用于具有数值数据的问题，而不是绝对的分类问题，因此，它在许多具有定性边界的经典分类问题中的适用性受到了限制。SVM 的能力依赖于其数学基础：凸二次函数在线性不等式约束下的最小值。

SVM 被用于许多学习情况中，包括网页分类。在文本分类中，搜索和/或其他相关功能占有重要位置，每个文本文档都成为 SVM 的输入数据，目的是以词频信息为基础进行分类（Harrison and Luger 2002）。SVM 也被用到图像识别中，集中在使用灰度或颜色亮度信息的图像的边缘探测和形状描述问题（Cristianini and Shawe-Taylor 2000）。在图 11-18 中（改编自 Cristianini and Shawe-Taylor 2000），SVM 区分了棋盘的边界。关于 SVM 算法的更多细节和完整算法描述，可以查阅文章（Burges 1998）。

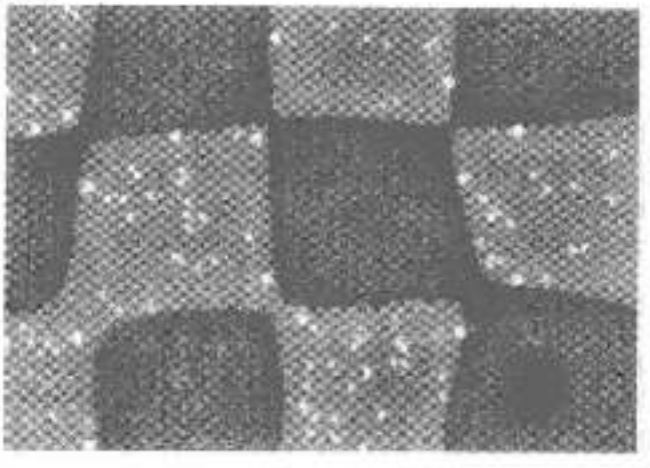


图 11-18 学习国际象棋棋盘边界的 SVM

注：数据点用高斯核心函数根据一致分布生成。图中的点是数据点，并用较大的点表示支持向量，较暗的区域表示分类的信心。改编自 Cristianini 和 Shawe-Taylor（2000）

11.5　Hebbian 一致性学习

11.5.1　概述

Hebbian 学习理论是通过对生物系统的观察得出来的：当一个神经元对另一神经元激活有贡献时，则在这两个神经元之间的连接或路径将会得到加强。Hebb（1949）陈述如下：

> 如果细胞 A 的轴突离细胞 B 很近，且反复不断地激活细胞 B，那么在一个细胞或全部细胞中将会出现新陈代谢变化，作为激活细胞 B 的细胞 A，它激活细胞 B 的效率将大大提高。

Hebbian 学习是很吸引人的，因为它在神经细胞层建立了基于行为的奖励概念。神经心理学研究证实了 Hebb 的观点：相互连接的神经元之间刺激的短暂接近能够改变神经元细胞之间的连接强度，虽然与 Hebb 的简单“增加有效性”相比要复杂得多，但是说明了 Hebbian 的观点在一定程度上是正确的。尽管 Hebbian 学习理论有点抽象，在这一节陈述的学习规则还是引用 Hebbian 学习理论。这种学习属于一致性学习的范畴，在这种学习中神经元处理中局部事件引起权值的变化。我们通过局部时间和空间属性来描述这种学习规则。

Hebbian 学习用于很多神经网络体系结构。既用于有监督的学习，也用于无监督的学习。当一个神经元对另一个神经元的激活有贡献时，在两个神经元之间的连接强度可得到加强，通过它们的输出值的符号乘上一个常数调整两者之间的连接权值来实现。

让我们看看这是怎样工作的。假如 i 和 j 是相互连接的，i 的输出是 j 的输入。定义两者连接权重的调整量为 ΔW，当作 $c*(o_i*o_j)$ 的符号，其中 c 是控制学习率的常量。在表 11-4 中，O_i 是 i 的输出值的符号，O_j 是 j 的输出。从表的第一行可以看出，当 O_i 和 O_j 都是正值时，权值的调整量 ΔW 也是正的。当 i 对 j 的激活有贡献时，这就有增强 i 和 j 之间的连接强度的效果。

表 11-4　结点输出值的符号和运算结果

O_i	O_j	$O_i * O_j$
+	+	+
+	−	−
−	+	−
−	−	+

在表 11-4 的第二和第三行，i 和 j 是相反的符号。因为它们的符号相反，我们要阻止 i 对 j 的输出做贡献。因此，我们通过一个负增量来调整两者之间的权值。最后，在第四行，i 和 j 又有相同的符号，这意味着我们要增强它们的连接。这种调整权值的方法可增强连接路径，当它们有相同的符号时，否则减弱。

在下一节，我们学习有监督和无监督的两种 Hebbian 学习。我们首先考虑无监督的情形。

11.5.2　无监督 Hebbian 学习的例子

回忆一下，在无监督的学习中关键的是不能提供“正确”输出值；因此，权重调整只能依靠神经元输入和输出值之间的函数。这种网络的训练有加强网络对已有模式的响应的效果。在下一个例子中，我们将看到 Hebbian 技术怎样用于有条件反应模型学习，这里一个任意选定的刺激可当作我们期望反应的条件。

对于无监督 Hebbian 学习中结点 i 的权重 ΔW 可调整：

$$\Delta W = c * f(X, W) * X$$

其中 c 是学习常量，一个很小的正数。f(X, W)是 i 的输出，X 是 i 的输入向量。

现在来看通过 Hebbian 学习，怎样把网络对无条件的刺激做出响应转换到对有条件的刺激做出响应的情形。从 Pavlov 的试验中可推出这种学习模型，在 Pavlov 的试验中，每当把食物给狗吃时伴随着铃声出现，这样，狗的唾液反应就从食物转移到铃声。图 11-19 的网络有两层，输入层有 6 个结点，输出层有一个结点。输出层返回 +1，表示输出神经元被激活，返回 -1，表示输出神经元静止。

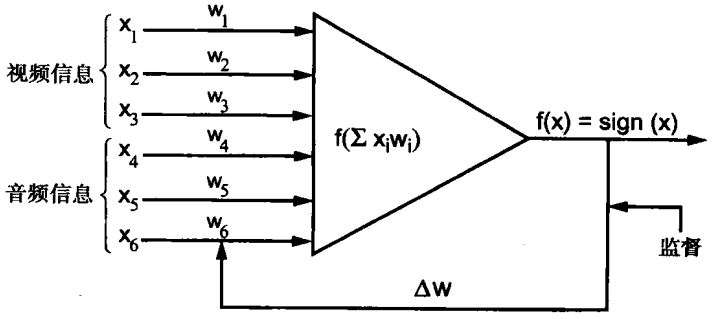


图 11-19　一个应用混合 Hebbian 结点的神经元例子，这里学习是有监督的

我们让学习常量为很小正实数 0.2。在这个例子中，对网络训练模型[1, -1, 1, -1, 1, -1]，它是两个模型[1, -1, 1]和[-1, 1, -1]的关联，[1, -1, 1]表示无条件的刺激，[-1, 1, -1]表示新的刺激。

假定网络已经能对无条件刺激做出正确的反应，但是对于新的刺激没反应。权值向量[1, -1, 1]刚好匹配输入模式，因此用它来模仿网络对无条件刺激的正确反应；用权值向量[0, 0, 0]模仿网络对新刺激的无反应状态。这两个权值向量连接一起就是网络的初始权值向量[1, -1, 1, 0, 0, 0]。

我们现在训练网络的输入模式，希望产生这样的一组权值向量，它能对新刺激做出正确的反应。第一次循环如下：

$$W * X = (1 * 1) + (-1 * -1) + (1 * 1) + (0 * -1) + (0 * 1) + (0 * -1)$$
$$= (1) + (1) + (1) = 3$$
$$f(3) = sign(3) = 1$$

于是得到新权值向量 W^2：

$$W^2 = [1, -1, 1, 0, 0, 0] + 0.2 * (1) * [1, -1, 1, -1, 1, -1]$$
$$= [1, -1, 1, 0, 0, 0] + [0.2, -0.2, 0.2, -0.2, 0.2, -0.2]$$
$$= [1.2, -1.2, 1.2, -0.2, 0.2, -0.2]$$

把新的权值向量应用于原始的输入模式：

$$W * X = (1.2 * 1) + (-1.2 * -1) + (1.2 * 1) + (-0.2 * -1) + (0.2 * 1) + (-0.2 * -1)$$
$$= (1.2) + (1.2) + (1.2) + (+0.2) + (0.2) + (0.2) = 4.2$$
$$sign(4.2) = 1$$

产生新的权值向量 W^3：

$$W^3 = [1.2, -1.2, 1.2, -0.2, 0.2, -0.2] + 0.2 * (1) * [1, -1, 1, -1, 1 -1]$$
$$= [1.2, -1.2, 1.2, -0.2, 0.2, -0.2] + [0.2, -0.2, 0.2, -0.2, 0.2, -0.2]$$
$$= [1.4, -1.4, 1.4, -0.4, 0.4, -0.4]$$

可看出向量结果 W * X 将继续朝正方向增长，向量的每一个分量在每次循环中绝对值都增长 0.2。经过 10 次以上的 Hebbian 训练，权值向量如下：

$$W^{13} = [3.4, -3.4, 3.4, -2.4, 2.4, -2.4]$$

我们现在用这个向量测试网络对两个模式的反应。我们关注网络是否继续能对无条件的刺激做出正确的反应，更重要的是，网络能否对新的、有条件的刺激做出正确的反应。我们首先用无条件刺激[1, -1, 1]测试，对于输入向量的后三个元素任意赋值为 1 和 -1。例如，用向量[1, -1, 1, 1, 1, -1]测试：

$$\begin{aligned}\text{sign}(W*X) &= \text{sign}((3.4*1) + (-3.4*-1) + (3.4*1) \\ &\quad + (-2.4*1) + (2.4*1) + (-2.4*-1)) \\ &= \text{sign}(3.4 + 3.4 + 3.4 - 2.4 + 2.4 + 2.4) \\ &= \text{sign}(12.6) = +1\end{aligned}$$

网络还是能对原先的无条件刺激做出积极的反应。我们现在做第二个测试，前面还是无条件刺激，后面三个分量取与上面不同的值[1, -1, 1, 1, -1, -1]

$$\begin{aligned}\text{sign}(W*X) &= \text{sign}((3.4*1) + (-3.4*-1) + (3.4*1) \\ &\quad + (-2.4*1) + (2.4*-1) + (-2.4*-1)) \\ &= \text{sign}(3.4 + 3.4 + 3.4 - 2.4 - 2.4 + 2.4) \\ &= \text{sign}(7.8) = +1\end{aligned}$$

第二个向量也能产生一个积极的网络反应。实际上，这两个例子表明，网络对原始刺激的灵敏度得到加强，这是因为反复暴露于那个原始刺激。

我们现在测试网络对新刺激的反应，新刺激模式是输入向量的后面三个分量编码为 [-1, 1, -1]。对于前面三个分量任意设置为 1 和 -1，然后用向量[1, 1, 1, -1, 1, -1]测试网络：

$$\begin{aligned}\text{sign}(W*X) &= \text{sign}((3.4*1) + (-3.4*-1) + (3.4*1) \\ &\quad + (-2.4*1) + (2.4*1) + (-2.4*-1)) \\ &= \text{sign}(3.4 - 3.4 + 3.4 + 2.4 + 2.4 + 2.4) \\ &= \text{sign}(10.6) = +1\end{aligned}$$

第二种刺激的模式也被识别出来了！

用轻微退化的向量模式做最后一个测试。这代表这样的刺激情形：也许是因为新的食物或者不同的铃声被使用，输入信号有轻微的改变。我们用向量[1, -1, -1, 1, 1, -1]测试网络，这里前三个分量有一个跟原来的无条件刺激模式不同，后三个分量也有一个同条件刺激模式不同：

$$\begin{aligned}\text{sign}(W*X) &= \text{sign}((3.4*1) + (-3.4*-1) + (3.4*1) \\ &\quad + (-2.4*1) + (2.4*1) + (-2.4*-1)) \\ &= \text{sign}(3.4 + 3.4 - 3.4 - 2.4 + 2.4 + 2.4) \\ &= \text{sign}(5.8) = +1\end{aligned}$$

就连部分退化的刺激也被识别出来了！

Hebbian 学习模型产生了什么？通过反复地把老刺激和新刺激一起呈现，于是在新刺激和老反应之间建立了一种关联。网络就是在没有监督的情况下学习把这种反应转移到新的刺激上。这种加强的灵敏度也允许对轻微退化的刺激模型做出同样正确的反应。这些可通过用 Hebbian 一致性学习来实现，这种学习增加网络对整个模型反应的强度，而且一样可增加网络对整个模型中各个分量模型做出正确反应的强度。

11.5.3　有监督 Hebbian 学习

Hebbian 学习规则是基于这样的原理：当一个神经元对另一个神经元的激活有贡献时，则在这两个神经元之间的连接强度将加强。当连接权重的调整是基于期望的输出，而不是实际输出时，这个原理也可适用于有监督的学习。例如，如果输入神经元 A 引起神经元 B 的是正反应，而且神经元 B 的期望输出也是正反应，则从 A 到 B 的连接权重将加强。

我们来看一个有监督 Hebbian 学习的应用，它显示了网络怎样被训练成可识别模式之间的关联。这种关联以有序对的形式给出，$\{<X_1, Y_1>, <X_2, Y_2>, \cdots, <X_t, Y_t>\}$，其中 X_i 和 Y_i 是被关联的向量模式。假如 X_i 的长度是 n，Y_i 的长度是 m。我们设计一个适合这种情形的网络。因此，这个网络有两层，输入层大小为 n，输出层大小为 m，如图 11-20。

上一节中关于 Hebbian 网络学习的公式可继承过来。

$$\Delta W = c * f(X, W) * X$$

其中 f（X,W）是网络结点实际输出。在有监督的学习中，我们用期望的输出向量 D 代替实际输出，公式如下：

$$\Delta W = c * D * X$$

从关联模式集合中给出向量对 $<X, Y>$，我们对输出层中结点 K 应用学习规则：

$$\Delta W_{ik} = c * d_k * x_i$$

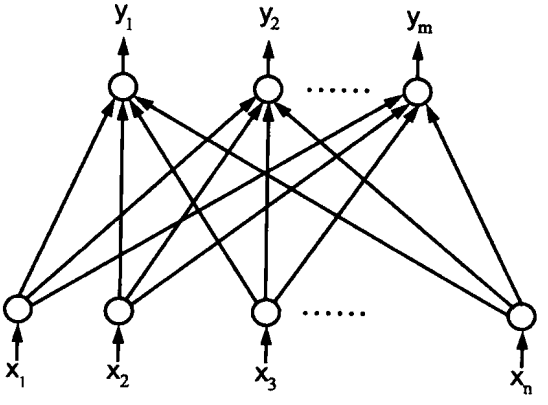


图 11-20　学习模式关联的有监督 Hebbian 网络

其中 ΔW_{ik} 是输入结点 i 到输出结点 k 的权值的调整量，d_k 是结点 k 的期望输出，x_i 是 X 的第 i 个分量。我们应用这个公式调整输出层中所有结点的所有权值。向量 $<x_1, x_2, \cdots, x_n>$ 是输入向量 X，向量 $<d_1, d_2, \cdots, d_m>$ 是输出向量 Y。把这个公式用于输出层的每一个结点和权重，写成矩阵形式，得到输出层权重调整的公式：

$$\Delta W = c * Y * X$$

其中向量乘积 Y * X 是广义向量积。广义向量积 YX 写成矩阵形式如下：

$$YX = \begin{bmatrix} y_1 \bullet x_1 & y_1 \bullet x_2 & \cdots & y_1 \bullet x_m \\ y_2 \bullet x_1 & y_2 \bullet x_2 & \cdots & y_2 \bullet x_m \\ \cdots & \cdots & \cdots & \cdots \\ y_n \bullet x_1 & y_n \bullet x_2 & \cdots & y_n \bullet x_m \end{bmatrix}$$

为了对整个关联模式对集合训练网络，我们循环使用这些模式对，利用公式对每一个模式对 $<X_i, Y_i>$ 调整其权值：

$$W^{t+1} = W^t + c * Y_i * X_i$$

对于整个训练集合得到：

$$W^1 = W^0 + c\,(Y_1 * X_1 + Y_2 * X_2 + \cdots + Y_t * X_t)$$

其中 W^0 是初始权重值。如果我们把 W^0 初始化为 0 向量 $<0,0,\cdots,0>$，学习常数 c 设为 1，则可得到如下设置网络权值的公式：

$$W = Y_1 * X_1 + Y_2 * X_2 + \ldots + Y_t * X_t$$

用这个公式设置权值把输入向量映射到输出向量的网络叫做线性联想器（linear associator）。从上面可看出，线性联想器网络是基于 Hebbian 学习规则的。在实际应用中，可直接用这个公式初始化网络的权值，而不用经过训练。

下面我们分析线性联想器的属性。我们可看出，在这个模型中通过一个权值矩阵存储了多个关联模式。这加大了存储模式之间相互作用的可能性。在下一节我们分析这种相互作用产生的问题。

11.5.4　联想记忆和线性联想器

线性联想器首先由 Tuevo Kohonen（1972）和 James Anderson 等（1977）提出。在这一节，我们用线性联想器网络在记忆中存储模式和从记忆中恢复模式。我们讨论不同的记忆检索模式，包括异相关模式、自相关模式和插入模式。我们把线性联想器网络当作基于 Hebbian 学习的插入记忆模式的实现来分析。这一节的最后考虑在记忆中多模式编码的冲突。

我们先学习几个定义。模式和记忆值都用向量表示。在把问题表示成一个特征向量集合时通常会诱发归纳偏置。将存入记忆中的关联模式用向量对集合 $\{<X_1,Y_1>,<X_2,Y_2>,\cdots,<X_t,Y_t>\}$ 表示。对于每一个向量对 $<X_i,Y_i>$，X_i 是用来检索模式 Y_i 的关键字。主要有三种相关记忆方式。

1）*异相关模式*：从 X 到 Y 的映射是，如果任意的向量 X 离模式 X_i 比离其他的模式更近，则返回相关的向量 Y_i。

2）*自相关模式*：映射方式同异相关，只是对于所有的模式对 $X_i = Y_i$。因为模式 X_i 关联到自身，所以这种记忆方式主要用于从部分或退化的模式回忆到全模式的情形。

3）*插入模式*：从 X 到 Y 有影射函数 ø，当 X 不同于样本模式时，也就是说 $X = X_i + \Delta_i$，则输出 $ø(X) = ø(X_i + \Delta_i) = Y_i + E$，其中 $E = ø(\Delta_i)$。在插入模式的映射中，如果输入向量是样本模式 X_i，则相关的模式 Y_i 被检索到；如果与样本模式向量相差 D，则输出向量相差 E，此处 $E = ø(\Delta)$。

自相关和异相关记忆用于检索原有模式。它们构成真正意义上的记忆，因为被检索的模式是存储在记忆中的一个拷贝。我们有时也需要一种系统的方法构造输出模式，这个模式与存储在记忆中的模式不同。这就是插入记忆的功能。

图 11-21 所示的线性联想器网络实现了一种形式的插入记忆。如 11.5.3 节所述，它是基于 Hebbian 学习模型。网络初始权值由以下方程得到：

$$W = Y_1 * X_1 + Y_2 * X_2 + \cdots + Y_t * X_t$$

由这个权值，网络可正确地检索匹配的样本模式；否则，它就形成插入记忆模式映射。

我们下一步介绍一些相关概念和符号，以帮助我们分析这个网络的行为。首先我们要介绍一种度量标准，它用于精确定义两向量之间的距离。例子中的所有模式向量都是汉明（Hamming）向量，它只由 +1 和 −1 组成。我们用汉明距离描述两个汉明向量之间的距离。我们形式地定义汉明空间：

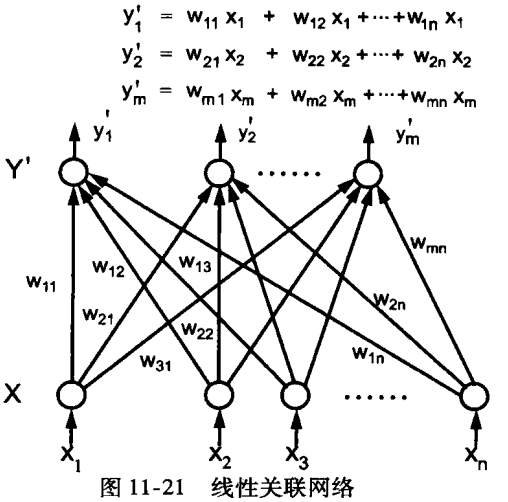


图 11-21　线性关联网络

注：向量 X_i 是当输入，关联的向量 Y_i 当输出，y'_i 是输入 X 的线性组合。在训练中，每一个 y'_i 都有正确的输出信号

$H^n = \{X = (x_1, x_2, \cdots, x_n)\}$

其中每个 x_i 取值于 $\{+1, -1\}$。

对于汉明空间中的两个汉明向量的汉明距离定义如下：

$\| X, Y \|$ = X 和 Y 中对应分量不同的个数。

例如，对于四维汉明空间：

(1, -1, -1,1) 和 (1,1, -1,1) 的汉明距离是 1

(-1, -1, -1,1) 和 (1,1,1, -1) 的汉明距离是 4

(1, -1,1, -1) 和 (1, -1,1, -1) 的汉明距离是 0

我们还需两个定义。第一是补向量，汉明向量的补向量是把原向量中的每一个分量求反，从 +1 变到 -1，从 -1 变到 +1。例如，(1, -1, -1, -1) 的补向量是 (-1,1,1,1)。

第二是标准正交向量，标准正交向量是正交的，同时还是单位向量。两个标准正交向量的点积等于 0。因此，对于标准正交向量集任意两个不同的向量 X_i 和 X_j 相乘，其点积等于 0。

$X_i X_j = \delta_{ij}$　　当 i = j 时 $\delta_{ij} = 1$，否则为 0

上面所述带有映射函数 ø(X) 的线性联想器网络具有以下的两个属性，第一，对于与样本模式匹配的输入向量 X_i，网络输出 ø(X_i) 是 Y_i，与 X_i 关联的样本模式；第二，对于不与样本模式匹配的输入向量 X_k，网络输出 ø(X_k) 是 Y_k，它是 X_k 的线性插入值。更准确地说，如果 $X_k = X_i + \Delta_i$，X_i 是样本模式，网络返回值是：

$Y_k = Y_i + E$，其中 $E = ø(\Delta_i)$

我们先看当输入 X_i 是一个样本模式时，网络返回相关的样本模式：

$ø(X_i) = WX_i$，根据网络激励函数的定义

因为 $W = Y_1X_1 + Y_2X_2 + \cdots + Y_iX_i + \cdots + Y_nX_n$，可得到：

$$\begin{aligned}\Phi(X_i) &= (Y_1X_1 + Y_2X_2 + \cdots + Y_iX_i + \cdots + Y_nX_n)X_i \\ &= Y_1X_1X_i + Y_2X_2X_i + \cdots + Y_iX_iX_i + \cdots + Y_nX_nX_i\end{aligned}$$

由上面的定义，$X_iX_j = \delta_{ij}$：

$$\Phi(X_i) = Y_1\delta_{1i} + Y_2\delta_{2i} + \cdots + Y_i\delta_{ii} + \cdots + Y_n\delta_{ni}$$

由标准正交的条件，当 i = j 时，$\delta_{ij} = 1$，否则等于 0。因此可得到：

$$\Phi(X_i) = Y_1*0 + Y_2*0 + \cdots + Y_i*1 + \cdots + Y_n*0 = Y_i$$

同样可以看出，对于不等于任何样本模式的向量 X_k，网络执行插入映射。也就是说，对于 $X_k = X_i + D_i$，其中 X_i 是样本模式，

$$\begin{aligned}\Phi(X_k) &= \Phi(X_i + \Delta_i) \\ &= Y_i + E\end{aligned}$$

其中 Y_i 是与模式 X_i 相关联的向量，且

$$E = \Phi(\Delta_i) = (Y_1X_1 + Y_2X_2 + \cdots + Y_nX_n)\,\Delta_i$$

我们省略了详细的证明。

我们现在给出线性联想器的处理过程。如图 11-22 的简单线性联想器网络，它把四维向量 X 映

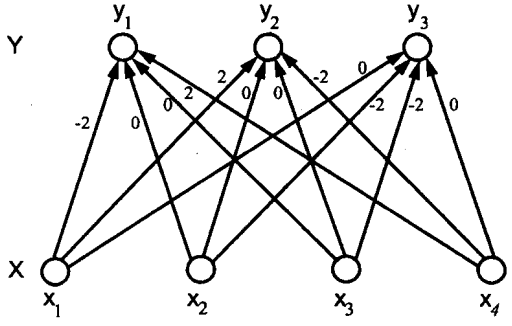


图 11-22　11.5.4 节例子中的线性联想器网络权值矩阵的计算用前一节所介绍的公式

射到三维向量 Y。因为我们是在汉明空间中，网络激励函数 f 还是前面的符号函数 sign。

如果我们想存储以下的两个相关向量 $<X_1, Y_1>$ 和 $<X_2, Y_2>$：

$X_1 = [1, -1, -1, -1] \leftrightarrow Y_1 = [-1, 1, 1]$
$X_2 = [-1, -1, -1, 1] \leftrightarrow Y_2 = [1, -1, 1]$

线性联想器用权值初始化公式，它是前面定义的向量的广义向量积：

$$W = Y_1X_1 + Y_2X_2 + Y_3X_3 + \cdots + Y_nX_n$$

我们现在计算 $Y_1X_1 + Y_2X_2$，网络的权值矩阵是：

$$W = \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 0 & 0 & 2 \\ 2 & 0 & 0 & -2 \\ 0 & -2 & -2 & 0 \end{bmatrix}$$

在一个样本模式上运行线性联想器。从第一个模式对中取出 X = [1, -1, -1, -1] 以找到其关联的向量 Y：

$y_1 = (-2*1) + (0*-1) + (0*-1) + (2*-1) = -4, \text{sign}(-4) = -1$
$y_2 = (2*1) + (0*-1) + (0*-1) + (-2*-1) = 4, \text{sign}(4) = 1$
$y_3 = (0*1) + (-2*-1) + (-2*-1) + (0*-1) = 4, \text{sign}(4) = 1$

因此有 Y_1 = [-1,1,1]，返回的是模式对的另外一半。

现在举一个样本模式的线性插入的例子。对向量 X = [1, -1,1, -1]：

$y_1 = (-2*1) + (0*-1) + (0*1) + (2*-1) = -4, \text{sign}(-4) = -1$
$y_2 = (2*1) + (0*-1) + (0*1) + (-2*-1) = 4, \text{sign}(4) = 1$
$y_3 = (0*1) + (-2*-1) + (-2*1) + (0*-1) = 0, \text{sign}(0) = 1$

注意到 Y = [-1,1,1] 不是原始的样本模式 Y，这个映射保存了相同的值，因为这两个样本 Y 是相同的。实际上，向量 X = [1, -1,1, -1] 与两个样本模式 X 之间的汉明距离都是 1；输出向量 [-1,1,1] 与两个样本模式向量 Y 的汉明距离也都是 1。

现在对线性联想器进行总结。要想得到线性联想器期望的属性，样本模式向量必须是标准正交向量集。因为以下两个原因，这限制了它的应用：第一，也许没有显示的映射，从现实世界的情形映射到标准正交向量模式集；第二，存储的模式的数目受到向量空间的维数的限制。当标准正交的条件不满足时，在存储的模式之间就会出现冲突，这种冲突会引起串扰。

还可观察到，只有当输入向量刚好匹配样本模式 X 时，线性联想器才能确切检索到关联的向量 Y。如果在输入模式里没有确切的匹配，则结果是插入映射的结果。有人认为插入映射并不是真正意义上的记忆。人们经常需要真正的记忆检索功能，当输入与样本近似的向量时，能检索到确切的与之关联的模式。

在下一节，我们阐述线性联想器网络的带有吸引子的版本。

11.6　吸引子网络或“记忆”

11.6.1　概述

这之前讨论的网络是前馈网络。在前馈网络中，信息表示在输入结点集合中，信号通过结点或结点层往前传播，直到一定的结果出现。另一种重要的连接网络是反馈网络。反馈网络的体系结构与前馈网络不同，它的结点的输出信号直接或间接输入到结点中，形成一个循环。

反馈网络与前馈网络相比有以下几个主要的区别：

1）结点之间反馈连接的存在形式。

2）时延，例如非瞬时信号的传播。

3）网络输出是网络收敛的状态。

4）网络的可用性取决于其收敛属性。

当反馈网络不再变化时，它就被认为是达到了平衡状态。网络到达的平衡状态就是网络的输出。

在 11.6.2 节的反馈网络中，网络状态用一个输入模式初始化。网络通过一系列的状态处理这个模式，直到它达到平衡状态。网络达到的平衡状态就是从记忆中检索到的模式。在 11.6.3 节，我们讨论一个实现异相关记忆的网络。在 11.6.4 节，讨论自相关记忆。

这些记忆模式的认知特性是很有意思也很重要的。它给我们提供了一种可按内容寻址的记忆。这种相关可用于很多方面，可检索电话号码，从旧记忆中寻找悲伤的感受，甚至可从人的部分脸部图像识别人。很多研究者曾试图在基于符号的数据结构中抓住这些记忆中的关联特征，包括第 7 章所述的语义网、框架和对象系统。

吸引子定义为一个状态，在一个邻域中的状态都随着时间的推移朝吸引子转变。网络中的每一个吸引子都有一个邻域，在这个邻域中的任何网络状态都朝吸引子转变。这个邻域叫做吸引子的基。一个吸引子可由一个网络状态组成，也可由一系列循环的网络状态组成。

为了从数学意义上理解吸引子和它们的基，引入了网络能量函数（Hopfield 1984）的概念。带有能量函数的反馈网络，如果每一次网络转换都会减少网络的能量，则可保证这个网络能收敛。在 11.6.3 节讨论这种网络。

把需要的模式当吸引子装载在记忆中，吸引子网络就能实现内容可寻址的记忆。通过在优化问题的费用函数和网络能量之间创建映射关系，它也可用来解决优化问题，例如巡回推销员问题。这个问题的解决就转换成了减少整个网络能量。这类问题可用 Hopfield 网络解决。

11.6.2　双向联想记忆

双向联想记忆（Bi-directional Associative Memory，BAM）网络首先由 Bart Kosko（1988）提出，它由两个相互连接的网络层组成。也有连接到自身的反馈连接。从 n 维输入向量 X_n 到 m 维输出向量 Y_m 的 BAM 映射如图 11-23。因为从 X 到 Y 的连接是双向的，所以应该在每一个信息流或连接方向上都有一个相应的权重。

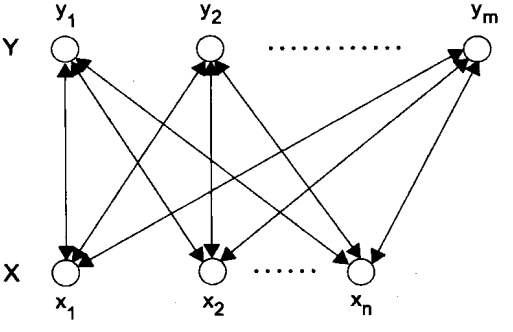


图 11-23　11.6.2 节中例子的 BAM 网络每一个结点还有可能连接到本身

像线性联想器的权值一样，BAM 网络的权值也可事先计算出来。实际上，我们用同样的方法计算 BAM 的网络权值。BAM 网络体系结构中的向量是从汉明向量集中选取。

假如我们希望存储的样本集合是 N 个向量对，则我们要像 11.5.4 节那样计算权值矩阵：

$$W = Y_1 * X_1 + Y_2 * X_2 + \cdots + Y_t * X_t$$

这个方程给出了从 X 层到 Y 层的连接权值，如图 11-23 所示。例如，w_{32} 是从 X 层的第二个结点到 Y 层的第三个结点的连接权值。假设在任意两个结点之间只有一条路径，那么，在 X 层和 Y

层的结点之间连接权值在两个方向上是一样的。于是，从 Y 层到 X 层的权值矩阵是 W 的转置矩阵。

在关联集合 $\{<X_1,X_1>,<X_2,X_2>,\cdots\}$ 上使用相同的权值初始化公式，BAM 网络就转化成了自相关网络，因为这个过程中的 X 层和 Y 层是相同的，我们可不用 Y 层，于是产生像图 11-24 所示的网络。在 11.6.4 节将看到自相关网络的例子。

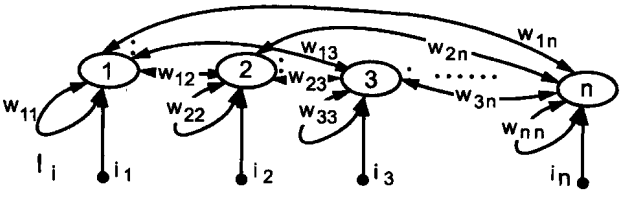


图 11-24　输入向量 I_i 的自相关网络

注：我们假定在不同的结点之间只有单个连接，于是，$w_{ij}=w_{ji}$，权值矩阵是对称矩阵

用输入模式初始化 X 层，BAM 网络就可用来从记忆中检索模式。如果输入模式有噪声，或者是不完全的样本，BAM 网络通常能完成模式，并检索到关联的模式。

用 BAM 回忆数据，通常要做以下工作：

1）把初始化向量对（X,Y）当作处理元素。X 是我们希望用来检索样本的模式，Y 被随机初始化。

2）把信息从 X 层传播到 Y 层，并更新在 Y 层的值。

3）把 Y 层的更新信息传回到 X 层，更新 X 单元。

4）继续上面两步，直到向量稳定，也就是说直到在 X 层和 Y 层的向量值不再有变化。

上面的算法表明 BAM 网络是一个反馈流，它双向流通，最后达到平衡。在上面的算法步骤中，也可从 Y 层开始达到收敛，这样导致选择的是 X 样本向量。它完全是双向的：我们可拿 X 向量当输入，最后得到关联向量 Y；也可把 Y 向量当作输入向量，最后返回 X。在下一节我们会见到具体的例子。

在收敛时，最后的平衡状态返回一个样本模式，这个模式用来构建原始权值矩阵的。如果所有的都是希望的，我们从样本向量对中拿出一个已知属性的向量，可能是相同的，也可能稍微不同。我们就用这个向量去检索样本对中的另一个向量。这里用的距离是汉明距离，汉明距离通过比较向量间的分量来度量，计算不同分量的个数。因为标准正交的限制，当 BAM 网络在一个向量收敛时，它也同时在它的补向量处收敛。因此提醒大家的是吸引子的补向量也会转变成吸引子。我们将在下一节给出一个这样的例子。

有几个因素可影响 BAM 网络的收敛。如果太多的样本映射到权值矩阵，则样本本身可能也会太近，在网络中产生假稳定。这种现象叫做串扰（crosstalk），在网络能量空间的局部最小处出现。

我们然后主要考虑 BAM 的处理。输入向量和权值矩阵的乘积计算的是配对向量的乘积，它的值对应输出向量的每一个分量。一个简单的阈值函数把结果向量转换成了汉明空间中的向量。因此有：

$$net(Y)=WX,\text{或对于分量 } Y_i,\ net(Y_i)=\sum w_{ij}*x_j$$

对于 X 层是相同的关系。在 t+1 时刻对于 net（Y）的阈值函数 f 也是很简单的：

$$f(net^{t+1})=\begin{cases}+1 & \text{若 } net>0\\ f(net^t) & \text{若 } net=0\\ -1 & \text{若 } net<0\end{cases}$$

在下一节，我们用几个具体的例子阐述双向联想记忆。

11.6.3　BAM 处理的例子

图 11-25 所示的是一个小型的 BAM 网络，它是 11.5.4 节介绍的线性联想器的简单变化改造的。这个网络把一个四维向量 X 映射到一个三维向量 Y。假定我们想创建两个样本向量对：

$x_1 = [1, -1, -1, -1] \leftrightarrow y_1 = [1, 1, 1]$
$x_2 = [-1, -1, -1, 1] \leftrightarrow y_2 = [1, -1, 1]$

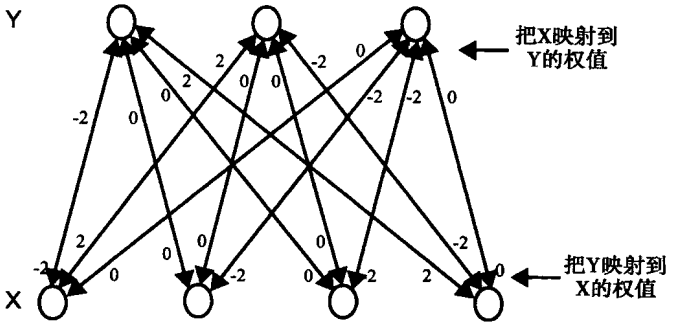


图 11-25　11.6.3 节中例子的 BAM 网络

按照上一节所示的公式我们创建权值矩阵：

$$W = Y_1X_1^t + Y_2X_2^t + Y_3X_3^t + \ldots + Y_NX_N^t$$

$$W=\begin{bmatrix}1&-1&-1&-1\\1&-1&-1&-1\\1&-1&-1&-1\end{bmatrix}+\begin{bmatrix}-1&-1&-1&1\\1&1&1&-1\\-1&-1&-1&1\end{bmatrix}=\begin{bmatrix}0&-2&-2&0\\2&0&0&-2\\0&-2&-2&0\end{bmatrix}$$

从 Y 映射到 X 的权值矩阵是 W 的转置：

$$\begin{bmatrix}0&2&0\\-2&0&-2\\-2&0&-2\\0&-2&0\end{bmatrix}$$

我们现在选择几个向量测试 BAM 联想器。先从一个样本对开始，选择 X 模式，看能否得到 Y。让 X = [1, -1, -1, -1]：

$Y_1 = (1*0) + (-1*-2) + (-1*-2) + (0*-1) = 4, f(4) = 1$
$Y_2 = (1*2) + (-1*0) + (-1*0) + (-1*-2) = 4, f(4) = 1$
$Y_3 = (1*0) + (-1*-2) + (-1*-2) + (-1*0) = 4, f(4) = 1$

可见，样本对的另一半正确返回。读者可把 Y 向量当输入，看能否正确返回原始向量 X = [1, -1, -1, -1]。

对于下一个例子，向量 X = [1,1,1, -1]，向量 Y 任意初始化。我们用 BAM 映射 X：

$Y_1 = (1*0) + (1*-2) + (1*-2) + (-1*0) = -4, f(4) = -1$
$Y_2 = (1*2) + (1*0) + (1*0) + (-1*-2) = 4, f(4) = 1$
$Y_3 = (1*0) + (1*-2) + (1*-2) + (-1*0) = -4, f(4) = -1$

经过阈值函数 f 作用在向量 [-4,4, -4] 上，结果是 [-1,1, -1]。从 Y 返回到 X：

$X_1 = (-1*0) + (1*2) + (-1*0) = 2$
$X_2 = (-1*-2) + (1*0) + (-1*-2) = 4$
$X_3 = (-1*-2) + (1*0) + (-1*-2) = 4$
$X_4 = (-1*0) + (1*-2) + (-1*0) = -2$

应用阈值函数后得到原始向量 [1,1,1, -1]。因为在第一次变换中用开始的向量产生了一个稳定的结果，我们可认为发现了另外一个原型样本对。实际上，我们所选择的例子是原始向量对 $<X_2, Y_2>$ 的补向量！在证明：在 BAM 网络中，当一个向量对建立了一个稳定的样本模型，它的补向量也是稳定的模型，BAM 网络于是产生了另外两个模型：

$X_3 = [-1, 1, 1, 1] \leftrightarrow Y_3 = [-1, -1, -1]$
$X_4 = [1, 1, 1, -1] \leftrightarrow Y_4 = [-1, 1, -1]$

这次选择靠近样本向量 X 的向量 [1, -1,1, -1]。注意，它到四个 X 样本向量的最近汉明距离是 1。然后任意初始化向量 Y 为 [-1, -1, -1]：

$Y_1^{t+1} = (1*0) + (-1*-2) + (1*2) + (-1*0) = 0$
$Y_2^{t+1} = (1*2) + (-1*0) + (1*0) + (-1*-2) = 4$
$Y_3^{t+1} = (1*0) + (-1*-2) + (1*-2) + (-1*0) = 0$

当 $y_i^{t+1} = 0$ 时，从 11.6.2 节末尾的阈值方程可得到 $f(Y_i^{t+1}) = f(Y_i^t)$。因此，由于任意初始化时第一个和第三个分量是 -1，得到 Y = [-1,1, -1]。然后从 Y 返回到 X：

$X_1 = (-1*0) + (1*2) + (-1*0) = 2$
$X_2 = (-1*-2) + (1*0) + (-1*-2) = 4$
$X_3 = (-1*-2) + (1*0) + (-1*-2) = 4$
$X_4 = (-1*0) + (1*-2) + (-1*0) = -2$

阈值函数把这个向量映射到向量 X = [1, 1, 1, -1]。我们重复这个处理过程把这个向量返回到 Y：

$Y_1 = (1*0) + (1*-2) + (1*-2) + (-1*0) = -4$
$Y_2 = (1*2) + (1*0) + (1*0) + (-1*-2) = 4$
$Y_3 = (1*0) + (1*-2) + (1*-2) + (-1*0) = -4$

阈值函数应用到 [-4,4, -4] 得到 Y = [-1,1, -1]。这个向量与最近返回的向量是相同的，因此网络达到了稳定。这表明了，经过 BAM 网络的两次处理，一个靠近 X_4 的模式收敛到了稳定的样本。这有点类似于识别一个部分信息丢失或模糊的人脸和其他存储图像。原始 X 向量 [1, -1,1, -1] 和原型向量 X_4 [1,1,1, -1] 之间的汉明距离是 1。这个向量归属于样本对 $<X_4, Y_4>$。

在 BAM 网络的例子中，我们从样本对的 X 分量的开始处理。当然，当有必要时，我们可设计例子从分量 Y 开始，初始化 X 向量。

Hecht-Nielsen（1990，p. 82）发表了对 BAM 网络很有意思的分析。他认为支持 BAM 网络的线性联想器网络具有标准正交的特性，但这个特性是一个非常严格的限制。他举出一个例子表明：创建一个这样的网络的要求是向量线性无关，也就是说，在样本空间中一个向量不能通过其他向量的线性组合得到。

11.6.4　自相关记忆和 Hopfield 网络

John Hopfield 是加利福尼亚州技术研究院的物理学家，他的研究构成了现在连接体系结构具有可靠性的重要理由。他用物理中能量最小化的概念研究网络收敛的属性。在此原理上它也设计了一个大家非常熟悉的网络。作为一个物理学家，Hopfield 把物理现象的稳定状态理解成物理系统的能量最小。这种方法的一个具体例子是模拟金属冷却中退火分析。

首先复习一下反馈关联网络的基本特征。这些网络从一个包括输入向量的状态开始。然后网络通过反馈路径处理这些信号，直到它达到一个稳定状态。为了能使这个体系结构当作一个联想记忆，我们希望网络具有两个属性。第一，从一个初始状态开始，我们必须保证网络将收敛到某个稳定状态。第二，我们希望这个稳定状态在某种一定的距离测度中最靠近输入状态。

先看一个自相关网络，它的构建规则同 BAM 网络相同。上一节提到过使样本对中的 X 和 Y 相同，则可把 BAM 网络转换成一个自相关网络。这个转换的结果就是一个对称的权值矩阵。11.6.2 节的图 11-24 提供了一个例子。

存储了一系列样本向量$\{X_1, X_2, \cdots, X_n\}$的自相关网络的权值矩阵由以下公式计算：

$$W = \Sigma X_i X_i^t \qquad \text{对于 } i = 1, 2, \cdots, n$$

当我们从异相关网络创建自相关网络时，从 x_i 结点到 x_j 结点的权值与从 x_j 到 x_i 是相同的，因此权值矩阵是对称矩阵。这个假设只需要处理的两个结点只有一条路径和一个权值相互连接。我们有时也有一些特殊的要求：网络中的结点没有连接到自身的，也就是说，没有 x_i 到 x_i 的连接。在这种情形中主对角线上的元素都为 0。

跟 BAM 网络一样，用存储记忆的模式计算权值矩阵。我们用一个简单的例子阐明这一点。考虑以下的 3 个向量样本集：

$$X_1 = [1, -1, 1, -1, 1]$$
$$X_2 = [-1, 1, 1, -1, -1]$$
$$X_3 = [1, 1, -1, 1, 1]$$

下面我们用公式 $W = \Sigma X_i X_i^t$（i = 1，2，3）计算：

$$W = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & -1 \\ -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \\ -1 & -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

$$W = \begin{bmatrix} 3 & -1 & -1 & 1 & 3 \\ -1 & 3 & -1 & 1 & -1 \\ -1 & -1 & 3 & -3 & -1 \\ 1 & 1 & -3 & 3 & 1 \\ 3 & -1 & -1 & 1 & 3 \end{bmatrix}$$

我们用的阈值函数是：

$$f(net^{t+1}) = \begin{cases} +1 & \text{若 } net > 0 \\ f(net^t) & \text{若 } net = 0 \\ -1 & \text{若 } net < 0 \end{cases}$$

我们用样本 $X_3 = [1,1,-1,1,1]$ 测试网络，得到：

$$X_3 * W = [7, 3, -9, 9, 7]$$

用阈值函数得到 $[1,1,-1,1,1]$。可见这个向量马上自身达到稳定状态。这表明样本向量是它本身的稳定状态或者吸引子。

现在测试一个与样本 X_3 汉明距离为 1 的向量。网络必须返回这个样本。这与从部分退化的数据中检索记忆模式是等价的。选择 $X = [1,1,1,1,1]$：

$$X * W = [5, 1, -3, 3, 5]$$

用阈值函数得到向量 X_3 $[1,-1,-1,1,1]$。

现在来看第三个例子，这次向量离原型最近的汉明距离是 2，让 $X = [1,-1,-1,1,-1]$，可验证这个向量离 X_3 的汉明距离是 2，离 X_1 的汉明距离是 3，离 X_2 的汉明距离是 4。我们开始：

$X * W = [3, -1, -5, 5, 3]$，用阈值函数得到 $[1, -1, -1, 1, 1]$

这不与任何样本相似，也不是稳定点，因为：

$[1, -1, -1, 1, 1] * W = [9, -3, -7, 7, 9]$ 对应 $[1, -1, -1, 1, 1]$

网络达到稳定，但是不与记忆中存储样本相似！是不是发现了另外一个能量最小点？更进一步观察可得，新向量是样本 $X_2 = [-1,1,1,-1,-1]$ 的补向量。与异相关的 BAM 网络的情形一样，自相关网络在为原样本创建吸引子的同时，也为它的补向量创建了吸引子。这样我们总共有 6 个吸引子。

在上面所述中，我们看到了基于线性联想器记忆模型的自相关网络。John Hopfield 的目标是给出一个通用自相关网络，它能应用于任何单层网络，满足一定的约束集合。对这一类单层神经网络，Hopfield 证明了总存在一个网络能量函数，它可保证网络收敛。

Hopfield 的更进一步的目标是在处理神经元时，用更接近连续时间的方式代替离散时间。一种普通的模拟用连续时间异步更新 Hopfield 网络的方法是单独更新各个结点，而不是更新整个网络层。实现它时用一个随机选择的过程挑选要更新的下一个结点，同时会用一些方法保证网络中的所有结点将平等机会地得到更新。

Hopfield 网络的结构跟上面所讲的自相关网络是相同的：单个网络层中的所有结点都相互连接（见图 11-23）。激励和阈值函数也同前面一样工作。对于结点 i，

$$x_i^{new} = \begin{cases} +1 & 若 \sum_j w_{ij} x_j^{old} > T_i \\ x_i^{old} & 若 \sum_j w_{ij} x_j^{old} = T_i \\ -1 & 若 \sum_j w_{ij} x_j^{old} < T_i \end{cases}$$

对于给定的这个结构，需要设定一个更进一步的限制来刻画一个 Hopfield 网络。如果 w_{ij} 是从结点 i 到结点 j 的连接权值，Hopfield 网络的权值有如下的限制：

$w_{ii} = 0$　对所有的 i

$w_{ij} = w_{ji}$　对所有的 i 和 j

Hopfield 网络并没有与之相关的典型的学习算法，像 BAM 网络一样，它的权值通常是预先计算出来的。

Hopfield 网络的行为比任何其他类型的网络都好理解，除了感知机之外。因为它的行为可用一个 Hopfield 发明的简洁能量函数来刻画：

$$H(X) = -\sum_i \sum_j w_{ij} x_i x_j + 2\sum_i T_i x_i$$

这个能量函数具有这样的属性：网络的每一次转换都会减少整个的网络能量。假如 H 有一个预先确定的最小值，且每次 H 的减少量至少是一个确定的最小值。于是可推导出从网络的任何状态都可收敛。

对于任意处理元素 k，它是最近更新的结点，k 改变当且仅当 H 的值减少。能量的改变量 ΔH 是：

$$\Delta H = H(X^{new}) - H(X^{old})$$

用 H 的定义式替代这个方程得：

$$\Delta H = -\sum_i \sum_j w_{ij} x_i^{new} x_j^{new} - 2\sum_i T_i x_i^{new} + \sum_i \sum_j w_{ij} x_i^{old} x_j^{old} + 2\sum_i T_i x_i^{old}$$

因为只有 x_k 发生变化，对于不等于 k 的结点 i 有：$x_i^{new} = x_i^{old}$，这意味着总和中不包含 x_k 的项相互抵消。整理上式得到：

$$\Delta H = -2x_k^{new} \sum_j w_{kj} x_j^{new} + 2T_k x_k^{new} + 2x_k^{old} \sum_j w_{kj} x_j^{old} - 2T_k x_k^{old}$$

还有 $w_{ii} = 0$，$w_{ij} = w_{ji}$，上面公式最终写成：

$$\Delta H = 2(x_k^{old} - x_k^{new}) \left[\sum_j w_{kj} x_j^{old} - T_k \right]$$

为了说明 ΔH 是一个负值，我们考虑两种情形。第一，假设 x_k 从 -1 到 +1。为了使 x_k^{new} 是 +1，方括号的项必须是负值，因为 $x_k^{old} - x_k^{new}$ 等于 -2，ΔH 又必须为负数。假如 x_k 从 1 到 -1，出于同样的理由，H 必须是负数。如果 x_k 没有发生变化，$x_k^{old} - x_k^{new} = 0$ 于是 $\Delta H = 0$。

于是可给出结论，从任何状态出发网络都可收敛。更进一步，网络的收敛状态必定是局部能量最小点。如果不是的话，必定存在一个变化，它可进一步减少网络整体能量，更新选择算法将逐步地选择这个结点进行更新。

在实现自相关记忆的网络中，Hopfield 网络需要两个属性。然而，我们将会看到 Hopfield 网络并不一定总是具有第二个属性：它们收敛的稳定状态并不一定是初始状态。到目前对这个问题还没有通用的解决方法。

Hopfield 网络可用来解决优化问题，例如巡回推销员问题。为了解决这问题设计者必须找到一个方法把问题的费用函数对应到 Hopfield 能量函数。在能量减少的过程中，网络也将减少相应问题的费用。尽管对于一些有趣的问题（例如巡回推销员问题）这样的对应关系已经被发现，但一般来说，从问题状态到能量状态的映射是很难找到的。

在这一节介绍了异相关和自相关反馈网络。我们分析了这些网络的动态属性，展示了一些例子表明系统怎样朝吸引子演变。我们阐述了怎样将线性联想器网络改造成叫做 BAM 的吸引子网络。在讨论连续时间的 Hopfield 网络时，看到了网络行为可以用能量函数来描述。因为 Hopfield 网络的每一步变换都减少网络整体能量，因此这类网络可保证收敛。

这里连接网络的能量解决方法依然还有一些问题。第一，到达的能量状态不一定是系统全

局最小点。第二，Hopfield 网络不一定收敛在离输入向量最近的吸引子，这使得它不适合实现内容可寻址记忆。第三，用 Hopfield 网络进行优化时，没有通用的方法创建从约束到 Hopfield 能量函数的映射。最后，能够用于网络存储和检索的能量函数极小点的数目是受限制的，而且更重要的是，这个数目不能精确设定。网络的经验测试表明吸引子的数目是网络中结点数目的一小部分。这些和其他的主题都正在研究之中（Hecht-Nielsen 1990，Zurada 1992，Freeman and Skapura 1991）。

基于生物学的方法（例如遗传算法和细胞自动机）试图模仿隐含在生命进化中的学习规律。这些模型的学习方式是并行的、分布式的。例如，在遗传算法中模式的群体数目代表了解决问题的候选解决方案。在算法循环过程中，这些模式群体通过复制、变异、选择等操作进行演化。我们在第 12 章讨论这种方法。

11.7 结语和参考文献

在这一章我们介绍了连接学习。在 11.1 节我们对历史进行了回顾。要了解历史，还可以参阅 McCulloch 和 Pitts（1943），Oliver Selfridge（1959），Claude Shannon（1948），和 Frank Rosenblatt（1958）。早期心理学模型也是很重要的，尤其是 Donald Hebb（1949）。认知科学家继续在探索认知和脑体系结构的关系。当代的资源包括有：《An Introduction to Natural Computation》（Ballard 1997）、《Artificial Minds》（Franklin 1995）、《The Cognitive Neuro science of Action》（Jeannerod 1997）以及《Rethinking Innateness：A Connectionist Perspective on Development》（Elman et al. 1996）。

我们没有重点阐述连接体系结构的数学和可计算性方面的特点。想对它有个大致了解的话我们推荐 Robert Hecht-Nielsen（1990）、James Freeman 和 David Skapura（1991）、Jacek Zurada（1992）以及 Nello Cristianini 和 John Shawe-Taylor（2000）。Christopher Burges（1988）写了一本极好的关于支持向量机的教程。Bertsekas 和 Tsitsiklis（1996）介绍了神经动态规划。关于连接应用可以参考 P. M. Bhagat（2005）的《Pattern Recognition in Industry》。

研究学习机制的科学家在知识的表示和可计算性等方面还有很多问题需要考虑。这些问题包括网络的体系结构和联接选择，决定环境的哪些认知参数需要处理，输出结果的意义等。也包括神经元－符号混合系统，以及这些东西对智能的特性有什么影响。

反传网络也许是连接体系结构中用得最普遍的网络，因此我们对它的起源、应用以及发展给予了详细的讨论。《Parallel Distributed Processing》（Rumelhart et al. 1986b）的两卷从计算性和认知工具的角度介绍了神经网络。《Neural Networks and Natural Intelligence》（Grossberg 1988）是另一篇对这个主题的完整介绍。

应用反传网络时也还有很多的问题，包括隐含层的数目及隐含层中结点的数目，选择训练集合，调节学习常量，偏置结点的使用，等等。这些课题可以归入一般的标题“归纳偏置”：知识的作用、期望以及可供问题求解器找出求解方案的工具。我们将在第 16 章讨论这些问题。

很多连接体系结构的设计者对他们的工作都有描述。包括 John Anderson 等（1977）、Stephan Grossberg（1976，1988）、Geoffrey Hinton 和 Terrance Sejnowski（1986）、Robert Hecht－Nielsen（1989，1990）、John Hopfield（1982，1984）、Tuevo Kohonen（1972，1984）、Bart Kosko（1988）以及 Carver Mead（1989）。Michael Jordan（1999）和 Brendan Frey（1998）提出了很多现代的方法（包括图示模型）。Christopher Bishop（1995）写了一本很好的相关教科书。

11.8 习题

1. 构建一个 McCulloch－Pitts 神经元，它能计算逻辑函数蕴涵“⇒”。

2. 用 LISP 语言构建一个感知机网络，并运行 11.2.2 节的分类的例子。
   a）产生一个与表 11-3 类似的数据集，在其上运行分类器程序。
   b）取运行分类器程序的结果并用权值去确定该线性可分集合的类别。
3. 用 LISP 语言或 C + + 实现反传网络，并用它解决 11.3.3 节的异或问题。尝试用不同的反传体系结构解决异或，也许可用两个隐含结点和没有偏置结点的结构。比较不同体系结构的网络的收敛速度。
4. 用 LISP 语言或 C + + 实现 Kohonen 网络，并用它对表 11-3 中数据进行分类。与 11.2.2 节和 11.4.2 节中的结果进行比较。
5. 写一个逆传网络实现异或问题。并与 11.3.3 节的反传网络的结果进行比较。用逆传网络分辨表 11-3 中的类别。
6. 用反传网络识别 10 个（手写）数字。一种方法是创建一个 4 × 6 的点阵。当一个数被写在这个网格上时，它将覆盖一些元素，这些元素取值为 1，其他的取值为 0。这个 24 个元素的向量将构成你的网络的输入。你要创建你自己的训练向量。用逆传网络做相同的事，然后比较结果。
7. 选择一个与我们在 11.5.2 节使用的不同的输入模式，用无监督的 Hebbian 学习算法识别这个模式。
8. 在 11.5.4 节中，用线性联想器算法使两个向量对关联。选择第三个（新的）关联向量对，并解决相同的问题。然后测试你的线性联想器是否是插入模式；也就是说，它能关联到与之邻近的样本的遗失？使你的线性联想器自相关。
9. 考虑 11.6.3 节中的双向联想记忆（BAM），改变例子中给出的关联向量对，为关联创建权值矩阵。选择新的向量并测试你的 BAM 联想器。
10. 描述 BAM 记忆和线性联想器的区别。什么是串扰（crosstalk），它是怎样进行干扰的？
11. 构建一个 Hopfield 网络解决十个城市的巡回推销员问题。

第 12 章　机器学习：遗传性和涌现性

对于这种在悠久年代中发生作用并严格检查每一生物的整个组成、构造和习性——助长好的并排除坏的——的力量能够加以限制吗？对于这种缓慢地并美妙地使每一类型适应于最复杂的生命关系的力量，我无法看到有什么限制。

——查尔斯·达尔文，《物种起源》

预言的第一定律：

如果一位德高望重的科学家指出某件事是可能的，那他几乎肯定是正确的。如果他说某件事是不可能的，那他也许是非常错误的。

第二定律：

要发现某件事情是否可能的界限，惟一的途径是跨越这个界限，从不可能跑到可能中去。

第三定律：

任何非常先进的技术，初看都与魔法无异。

——阿瑟·查尔斯·克拉克，《未来的轮廓》

12.0　社会性和涌现性的学习模型

就像连接网络从创建人工神经系统的工作中得到最初的支持和灵感，同样的也有很多的生物类比方法影响了机器学习算法的设计。这一章考虑遵循基本演化过程形成的学习算法：通过适者生存的个体形成群体。跨越不同个体群体的选择能力，已经展现在物种自然进化的必然过程中以及基本文明变革的社会进程中。这些在细胞自动机、遗传算法、遗传程序设计、人工生命及其他的涌现计算的研究中得到形式化。

涌现学习模型模仿大自然中最优美、最强大的适应形式：植物和动物的生命演化形式。达尔文认为："……对于这种缓慢地并美好地使每一类型适应于最复杂的生命关系的力量，我无法看到有什么限制……"。在连续的世代中注入变异，有选择地淘汰适应性低的个体，通过这样简单的过程，生物体的适应能力得到提高，在群体中呈现个体的多样性。演化和涌现出现在包含个体的群体中。个体的行为影响其他个体，反过来，它也受其他个体的影响。因此，选择的压力不仅来自外部环境，而且也来自群体中个体相互作用中。一个生态系统包含很多成员，每一个成员都充当一定的角色，都有适应生存的一定技能，但更重要的是，它的积累的行为影响群体中的其他成员，同时也受其他成员的影响。

因为它们的简单性，潜在的进化过程是非常普遍的。在生物中通过基因改变的选择，生物进化产生了物种。同样，在文化进化中，通过对信息的社会传播和改进产生了知识。通过对候选问题解决办法的操作，遗传算法和其他类似的算法就具有增量式解决问题的能力。

当遗传算法用于求解问题时，它有三个明显的阶段：第一阶段，问题领域单独的潜在解决方案要编码成这样的形式，它们支持必要的变化和选择操作；通常，这种表示是类似于位串的形式。第二阶段，交配和变异算法，它和生物有性繁殖的形式类似，它会产生新的后代，这种后代组合了双亲的特征。最后，适应函数判断哪种个体是最优的个体，也就是说，最适合逐步解决这个问题。这样的个体容易生存、复制，形成下一代潜在的解决方案。逐渐地，将形成这样一代个体，它们可当作问题的解决办法去解释问题领域中的问题。

遗传算法也可用更加复杂的表示方法，包括产生式规则，为了适应与环境的相互作用形成

相应的规则集。例如，遗传程序设计就是组合和变化计算机代码的程序段，试图形成一个程序解决诸如提取数据集中不变量等问题。

社会交互学习的例子能从生命游戏这个游戏中看出，这个游戏最早由数学家 John Horton (1970，1971) 发明，然后被 Martin Gardner 在《Scientific American》推广。在这个游戏中，个体的产生、生存或死亡状态是它自己状态及它邻域中个体状态的函数。典型地，很少数量的规则(通常是三四个) 就可充分定义这个游戏。尽管它简单，但是这个生命游戏的实验表明它能进化出有非凡的复杂性和能力的结构，包括可自我复制的多细胞“组织”(Poundstone 1985)。

人工生命的一种很重要的方法是通过有限状态自动机的相互作用模仿生物进化的条件，用一个状态集合和转换规则集合完成其功能。这些细胞自动机能从它们自身以外接收信息，尤其是从它们的邻域中。转换规则包括产生指令、继续生命指令和死亡指令。当这些细胞自动机群体比较分散地设定在领域中，且可以像异步并行操作的合作主体一样行动时，我们有时可见到这种看似独立的“生命形式”的进化。

另一个例子是 Rodney Brooks (1986，1987) 和他的学生们设计制造的简单机器人，这些机器人像自治的主体一样可解决实验环境中的问题。这里没有中央控制算法，而是出现像个体的分布式的和自治的相互作用一样的合作。人工生命学界有定期的会议和刊物，反映他们的工作(Langton 1995)。

在 12.1 节介绍采用遗传算法 (Holland 1975) 的进化或基于生物学的模型，遗传算法是一种学习方法，它利用并行性、相互交互，并通常采用位级的表示方法。12.2 节介绍分类器系统和遗传程序设计，与遗传算法相比，这是一个新的研究领域，它采用更复杂的表示方法，例如建立和提炼产生式规则集 (Holland et al. 1986)，生成和调试计算机程序 (Koza 1992)。12.3 节介绍人工生命 (Langton 1995)。12.3 节开始介绍“生命游戏”，在结尾介绍了来自圣达菲研究所的一份研究报告 (Crutchfield and Mitchell 1995)，其中讨论了涌现性行为。

12.1 遗传算法

跟神经网络一样，遗传算法也是仿生算法，它们把学习看作是在进化的候选问题解决方案群体中的一种竞争。适应函数评估每一个方案，然后决定它是否对下一代方案的形成做出贡献。通过类似于染色体中基因转换的操作方法，算法生成新一代的候选解决方案。

假如 P(t) 定义了一个候选解决方案群体，x_i^t 在时刻 t：

$$P(t) = \{x_1^t, x_2^t, \ldots, x_n^t\}$$

下面是遗传算法的一般形式：

```
procedure genetic algorithm;

    begin
        set time t:= 0;
        initialize the population P(t);
        while the termination condition is not met do
            begin
                evaluate fitness of each member of the population P(t);
                select members from population P(t) based on fitness;
                produce the offspring of these pairs using genetic operators;
                replace, based on fitness, candidates of P(t), with these offspring;
                set time t := t +1
            end
    end
```

这个算法清晰明白地说明了遗传学习的基本框架；这个算法的不同实现用不同的方法例示了这个框架。群体保留的比例有多大？配对和产生后代的比例有多大？遗传算法怎么应用，对谁应用？“替代 $P(t)$ 中最弱的候选个体”的过程也许用这样简单的过程：删除固定比例的最弱的候选个体。更复杂一些的方法也许是用适应度对群体排序，然后计算它被删除的概率，这个概率是适应度的反函数。最后，替代算法用这个概率选择要被删除的候选个体。尽管对于群体中最好的个体它的删除概率很低，但是这个最好的个体依然有被删除的可能。这个算法的优点是：它可以保存这样的个体，它的全局适应度很低，但是它的分量对产生强大的个体有贡献。这个替代算法有很多叫法，包括*蒙特卡罗方法*、*适应度按比例选择方法*和*轮盘赌轮选择方法*。

尽管在 12.1.1 节将介绍一些更复杂的表示方法，但是在这里还是要介绍与遗传算法相关的简单的表示方法，它用简单的字符串表示问题解决方案。例如，假设我们要用遗传算法学习对 0 和 1 组成的字符串进行分类。我们用 1、0 和#的模式代表字符串群体，其中#是一个通用匹配符，它既可匹配 1，也可匹配 0。因此，模式 1##00##1 代表的是有 8 个位的字符串，它以 1 开头和结尾，中间是两个 0。

遗传算法用一个候选模式群体初始化 $P(0)$。一般情况下，初始化群体任意选择。假定有一个适应度函数 $f(x_i^t)$ 评估每一个候选方案，$f(x_i^t)$ 返回时刻 t 时候选个体的适应度。一个普通的候选个体适应度测度方法在一个训练例子集中测试，返回正确分类的百分比。用这样的一个适应函数，赋给每个候选个体适应值：

$$f(x_i^t)/m(P,t)$$

其中 $m(P,t)$ 是整个群体的平均适应值。显然，适应度值会随着时间的改变而改变，因此，适应度是整个问题解决方案阶段的函数，即 $f(x_i^t)$。

对每一个候选个体评估后，算法选择两个个体重新组合。重新组合使用*遗传算子*（genetic operator），产生新的个体，新个体由双亲分量组合而成。按照自然进化规律，候选个体适应值决定了它复制可能性的大小，适应值大的个体它被复制的可能性也大。就如上面所述，选择是基于概率的，比较弱的个体它复制的可能性也就很小，但也不是直接就删除。一些适应值小的个体的存活是很重要的，因为它们可能含有解决方案中的一些必不可少的因素（例如部分位模式），而且复制能吸收这种成分。

有很多遗传算子可产生具有双亲特征的后代：最普通的是*交叉*（crossover）。交叉算子选择两个候选个体，分解每一个个体，然后交换分量形成两个新的候选个体。图 12-1 显示了长度为 8 的交叉算子。交叉算子先从中间分开双亲个体，产生两个孩子：它们的前半部分来自一个双亲个体，后半部分来自另外一个双亲个体。注意，从中间分解候选个体是任意选择的。分解可从表达式的任意位置进行，而且，这种分解位置还可在解决过程中进行调整和改变。

例如，假定目标类是以 1 开头和结尾的字符串集合。图 12-1 中的双亲都能很好地完成这个任务。然而，第一个孩子比任何一个双亲个体都要好：它不含有假信息，也不太可能遗漏一些正确的字符串。应注意的是，它的同胞兄弟比它的任何一个双亲个体都要差，很有可能在下一次进化中被删除。

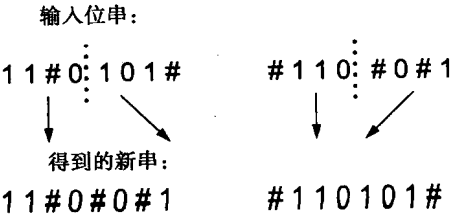


图 12-1　两个长度为 8 的位串上的交叉#是“通配符”

变异（mutation）是另一种很重要的遗传算子。变异算子选择单个个体，然后任意改变它的部分特征。例如，变异算子任意从模式中选择一位改变它，从 1 变换到 0 或者#。因为在初始群体中可能不含有解决方案的分量，于是变异算子就显得很重要。在我们

的例子中，如果初始群体中的任何个体的第一个分量都不是 1，则交叉算子因为它保持双亲个体的前半部分的分量不变，所以不能产生需要的后代。于是就需要变异算子改变这个位。其他的遗传算子（例如反转算子）也可完成这样的任务，这部分内容将在 12.1.3 节介绍。

遗传算法继续直到它的终止条件得到满足，例如是有一个或多个候选个体的适应值超过了某一个阈值。在下一节举例介绍遗传算法编码、遗传算子和适应度评估，并给出两个例子：CNF 约束满足问题和巡回推销员问题。

12.1.1　两个例子：CNF 可满足性问题和巡回推销员问题

接下来选择两个问题讨论表示问题和针对问题的适应函数的设计。首先要注意三个问题：第一，并不是所有问题都能很容易地用位串的表达式；第二，遗传算子要保存群体的一些重要关系，例如，巡回推销员问题中每一个城市必须都出现一次且仅一次；最后，讨论问题状态的适应函数和问题编码之间的重要关系。

例 12.2.1　CNF 可满足性问题

合取范式（CNF）的可满足性问题是很直接的：当一个命题表达式是用 and（“∧”）连接的子句序列时，它就是一个合取范式。每一个子句是用 or（“∨”）连接的文字组成。例如，如果有文字 a、b、c、d、e 和 f，则表达式

$$(\neg a \vee c) \wedge (\neg a \vee c \vee \neg e) \wedge (\neg b \vee c \vee d \vee \neg e) \wedge (a \vee \neg b \vee c) \wedge (\neg e \vee f)$$

就是 CNF。这个表达式是 5 个子句的合取，每一个子句是两个或更多文字的析取。在第 2 章介绍了命题逻辑和可满足性。在 14.2 节将介绍命题逻辑表达式的合取范式及其转换的方法。

CNF 可满足意味着要为这 6 个文字找出 true 或 false（1 或 0）的赋值，使得 CNF 表达式的值为 true。读者可验证 a、b 和 e 赋值为 false 就可使表达式为真，另一种方法是 e 为 false，c 为 true。

对于这个 CNF 可满足性问题一种比较自然的编码是一个 6 位长的位串，每一个位表示一个文字的赋值：true（1）或 false（0），按照 a、b、c、d、e、f 的顺序。因此：

1　0　1　0　1　0

代表了 a、c 和 e 赋值为 true，b、d 和 f 赋值为 false，整个 CNF 表达式的值是 false。读者可尝试其他使表达式为真的赋值。

我们要求每个遗传算子产生的后代是使 CNF 表达式为真的赋值，因此每个算子必须产生一个 6 位长的真值赋值模式。这里选择用位模式表示 CNF 表达式中文字的真假值的重要原因是：每个遗传算子都会产生一个合法的可能解。也就是说，交叉算子和变异算子产生的位串是问题的一个可能解。而且其他常用的遗传算子产生的也是 CNF 表达式的合法可能解，例如反转（inversion）算子（转换 6 位模式中位串的顺序）和交换算子（交换这个模式中不同的两个位）。实际上，从这个观点看很难找到一种比位串模式更适合 CNF 可满足性问题的编码方法。

为位串群体选择适应函数就不是那么直观了。从一个角度看，对于文字的任何一个真值都会使表达式要么为 true 要么为 false。如果一个特殊的赋值使表达式为 true，则表示找到了解；否则未找到。初看似乎很难决定一个适应函数，它能决定一个作为潜在解的位串的“质量”。

然而有很多的可供选择的办法。整个 CNF 表达式是由 5 个子句的合取组成。因此可构造一个等级测定系统，它考察模式满足的子句的个数，允许对潜在的位模式解在 0～5 的范围内进行排序。例如：

110010 适应度为 1，
010010 适应度为 2，
010011 适应度为 3，
101011 适应度为 5，它就是解。

这个遗传算法提供了 CNF 可满足性问题的一种合理的解决方法。遗传算法的一个重要特性是解群体提供的隐含的并行性。遗传算子对表达式有天然的适应性。最后，解搜索似乎也符合一种并行的“分而治之”的策略，因为适应度是由满足的问题分量的个数决定的。在本章的练习中读者要考虑这个问题的其他解决方法。

例 12.2.2　巡回推销员问题

巡回推销员问题（traveling salesperson problem，TSP）是人工智能和计算机科学中的典型问题。在 3.1 节中结合图论介绍了这个问题。它的整个状态空间需要的状态数是 N!，N 是要访问的城市的数目。它被证明是 NP 难题，很多学者用启发式方法对它进行过求解。这个问题的陈述如下：

> 一个推销员要求访问 N 个城市，在任意路线的两个城市之间有一个关联的费用（例如公里数、航空费用等）。找出一个费用最少的路径，从一个城市出发，经过所有其他的城市一次仅且一次，然后回到出发点。

TSP 问题有很多很好的应用，包括电路板布线、X 射线晶体学（X-ray crystallography）以及超大规模集成电路布线等。在这些问题中需要访问成千上万的点（城市）。在分析 TSP 时一个很有意思的问题是，是否值得花几个小时在昂贵的工作站上得到一个接近最优解的方案，或在廉价的 PC 上运行几分钟得到一个“充分好”的结果。TSP 是一个很有意思和难度的问题，它有很多的研究分支。

我们怎样用遗传算法解决这个问题？首先，表示访问城市路径的编码的选择和一组遗传算子的创建不是很简单。但是，适应函数的设计很直观：我们要做的是评估路径的长度成本。然后按照长度成本对路径进行排序，成本越低的越好。

我们先看看显而易见的编码方法。假如我们有 9 个城市要访问，编号为 1，2，…，9，因此用这 9 个数字的有序链对路径进行编码。假如简单地用一个四位模式（0001，0010，…，1001）来代表每一个城市。模式

0001　0010　0011　0100　0101　0110　0111　1000　1001

表示按照它所在序列号的顺序被访问。为了方便阅读，我们在字符串中加入了空格。现在，怎么选择遗传算子？交叉算子首先被排除，因为从两个双亲中产生的新字符串极有可能不代表一条每个城市只访问一次的路径。实际上，通过交叉算子，有些城市会被移出而另外一些城市会被重复访问。变异算子怎么样？假如第 6 个城市（0110）的最左边位变成了 1？1110 或 14，不再是一个合法的城市。在路径表达式内反转和交换（交换代表城市的 4 位串）也许是一种可接受的遗传算子，但是它们是否足够有效能得到一个满意的方案？实际上，搜索最少花费路径的过程也就是产生和评估 N 个城市的所有可能排列。遗传算子必须能产生所有的排列。

另外一种方法是不用位串表示，而用一个字母或者数字（例如 1，2，…，9）表示一个城市；通过对这 9 个数字排序构造路径，然后选择合适的遗传算子产生路径。只要是任意交换路径中的两个城市，变异算子就可以，但是在两条路径之间的交叉算子也就没有用了。交换同一路径中的两个片断，或者是任何其他只是改变路径中字母的顺序（没有移走、添加和重复）的算子都是可行的。然而，这些方法都很难将两个双亲个体中优秀的成分组合到后代中去。

很多研究者（Davis 1985，Oliver et al. 1987）设计了能克服这些问题的交叉算子。例如，Da-

vis 定义了一种叫做有序交叉（order crossover）的算子。假定我们有 9 个城市 1，2，…，9，这些数字的顺序代表城市访问的顺序。

有序交叉通过在一个双亲的路径中选择城市子序列创建后代。它依然保持从另外一个双亲中来的城市的相对顺序。首先，选择两个划分点，用“|”标志，划分点任意插入到双亲路径中的相同位置。划分点的位置是任意的，但是一旦选择了，双亲中的另外一个就必须选择相同的位置。例如，对于双亲路径 p1 和 p2，划分点在第 3 个和第 7 个位置（城市）：

p1 = (1 9 2 | 4 6 5 7 | 8 3)
p2 = (4 5 9 | 1 8 7 6 | 2 3)

两个孩子 c1 和 c2 按下面的方法产生。首先，将双亲中划分点之间的部分复制到后代中：

c1 = (x x x | 4 6 5 7 | x x)
c2 = (x x x | 1 8 7 6 | x x)

然后，从一个双亲路径的第二个划分点开始，从另外一个双亲路径中来的城市按相同的顺序复制。当到达字符串的结尾时，再从字符串的开始继续。于是，从 p2 中得到的城市序列是：

2 3 4 5 9 1 8 7 6

因为 4，6，5 和 7 已经是第一个孩子的一部分，于是它们被移走，得到缩短序列 2，3，9，1 和 8，它们构成了 c1 中剩余的访问城市序列，且保留了在 p2 中的顺序：

c1 = (2 3 9 | 4 6 5 7 | 1 8)

用相同的办法产生第二个孩子 c2：

c2 = (3 9 2 | 1 8 7 6 | 4 5)

在有序交叉中，双亲路径 p1 的中间片段遗传给了 c1，同时 c1 的其余城市的顺序从另外一个双亲路径 p2 中继承得到。这个方法支持直观知识，城市的顺序在生成最少花费路径时非常重要，于是双亲路径中的顺序信息片段传递到孩子路径中去是很关键的。

有序交叉算法还保证了孩子路径是合法路径，每个城市访问一次且仅一次。如果想在这个结果中加入变异算子，我们首先提醒你要注意在同一路径中交换城市。反转算子（如果只是简单地改变整个路径的方向）是没有效的，当改变整个路径方向时，它没有产生新的路径。然而，如果路径的一个片段被截出来，改变方向，然后重新代入，这种反转算子是可接受的。例如：

c1 = (2 3 9 | 4 6 5 7 | 1 8)

在反转中间的片断后有：

c1 = (2 3 9 | 7 5 6 4 | 1 8)

可以这样定义一种新的变异算子：任意选择一个城市，然后任意选择路径中一个新的位置插入。这种变异算子也可用于路径的一个片段，例如，可选择一个 3 个城市的子序列，插入路径中的一个新位置，保持子序列的顺序不变。其他建议将在习题中给出。

12.1.2　遗传算法的评估

上面的例子重点介绍了遗传算法的知识表示的惟一性问题、遗传算子的选择和适应函数的设计。表示方法的选择必须支持遗传算子。有时，比如在 CNF 可满足性问题中，位串表示是最自然的方法。在这种情形中，传统的遗传算子（如交叉和变异）能直接用于产生潜在的解。巡

回推销员问题是完全不同的情况。第一，位串编码对它显得很不方便。第二，要设计新的变异和交叉算子能保证后代必须是合法的路径，也就是必须通过每一个城市一次且仅一次。

最后，遗传算子必须传递给后代信息丰富的潜在解的片断。如在 CNF 可满足性问题中，这种信息是真值赋值，遗传算子必须在后一代中保持它。在 TSP 问题中，路径的组织是关键，因此路径信息的分量必须传递到后代中。这种信息成功转移不仅依靠表示的选择，也依靠遗传算子的设计。

我们留下了表示的最后一个问题，也就是选择表示的自然性问题。我们考虑一个简单的例子，假如我们要遗传算子能区别数字 6、7、8 和 9。整数表示给出了一个很自然且平均的有序空间，因为在十进制的数中，下一个数只是在前一数上加 1。然而，改为用二进制编码的话，这种自然的属性就会消失。考虑 6、7、8 和 9 的位模式：

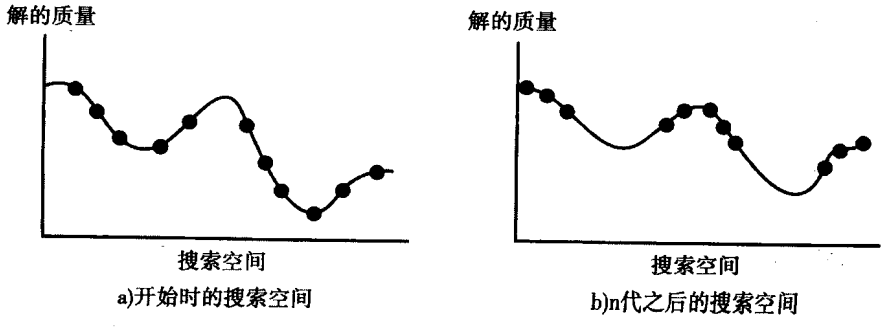
0110　0111　1000　1001

我们观察一下，在 6 和 7 之间还有 8 和 9 之间只有一位发生改变，但在 7 和 8 之间四位全部不相同了。这种表现的不规则在试图产生一个需要全部组织这些 4 位模式的解决方案时，会愈加明显。有很多技术用于解决这种不统一表现问题，大都在*葛莱编码*（gray coding）的基础上进行。例如。在表 12-1 显示了前 16 个二进制数的葛莱编码版本。在相邻两个数之间恰好只相差一位，用葛莱编码代替标准的二进制数，这样会使相邻状态间的遗传算子的转换变得自然和平滑。

表 12-1　对于二进制数 0，1，…，15 进行葛莱编码的位模式

二进制	葛莱码	二进制	葛莱码
0000	0000	1000	1100
0001	0001	1001	1101
0010	0011	1010	1111
0011	0010	1011	1110
0100	0110	1100	1010
0101	0111	1101	1011
0110	0101	1110	1001
0111	0100	1111	1000

遗传算法的一个重要特性是它搜索的并行性。遗传算法实现了一种强大的爬山法：它保持多个候选解，删除没有希望的，提高好的解决方案。图 12-2 从 Holland（1986）改编，它表明了多个候选解在搜索空间中朝最优点收敛。在这个图中，水平轴代表在解空间中的可能点，而垂直轴反应这些解的质量。曲线上的点是遗传算法中当前群体中的候选解的成员。开始时，候选解分散在可能解空间中。经过几代的进化后，它们趋向于聚集在解质量较高的区域。



a)开始时的搜索空间　　b)n代之后的搜索空间

图 12-2　用并行爬山法显示遗传算法

［改编自 Holland（1986）］

当我们把遗传搜索说成是“爬山法”时，默认了通过适应度曲线图决定移动方向的方法。这曲线有波峰和波谷，也就是局部最大值和最小值。实际上，在空间中的某些不连续是在表示和遗传算子的选择时人为造成的。例如，就如刚才讨论的那样，这种不连续性可能因为没有用葛莱编码而引起。应注意的是遗传算法不像在 4.1 节介绍的串行爬山法，它不会立即删除没有希望的候选解。通过遗传算子，即使是较弱的候选解也能继续对形成更好的候选解做出贡献。

遗传算法和第 4 章介绍的状态空间的启发式搜索的另一个区别是对当前状态和目标状态的区别分析。支持 A * 算法的信息内容需要能评估从当前状态移到目标状态的代价（详见 4.2 节）。而遗传算法并没有那样的要求，它只要对每一个当前潜在解的适应值进行评估。它也不需要严格地在 open 列表中对下一个状态进行排序，就如在状态空间搜索中看到的那样。遗传算法只有针对问题的适宜解群体，每一个个体都可能在并行搜索中对产生新的可能解有所帮助。

遗传算法力量的一个重要源泉是天生存在于进化算子中的*隐含的并行性*（implicit parallelism）。与状态空间搜索和有序 open 列表相比，搜索并行移动，对整个候选解群体进行操作。通过限制复制比较弱的候选个体，遗传算法不仅删除了该候选解，而且删除它的所有后代。例如，字符串 101#0##1，如果从它的中间点断开，能形成一个形如 101#_____的群体。如果发现这个字符串不适合，则删除它的同时将删除整个潜在的子孙群体。

随着遗传算法在应用领域和科学建模等方面的广泛应用，它吸引人们去探索它的理论基础。以下几个问题自然而然被提出：

1）我们能否刻画具有什么类型特征的问题用遗传算法能较好地解决？

2）什么类型的问题用遗传算法执行较差？

3）对一类问题用遗传算法执行效果好或者坏意味着什么？

4）有没有其他的规则可描述遗传算法行为的宏观意思？尤其是是否可做出预测，群体的子集合的适应性将怎样随时间而变化？

5）是否有方法可描述不同遗传算子（例如交叉、变异、反转等）的不同影响？

6）在什么样的环境下（什么样的问题和什么样的遗传算子）遗传算法比传统的人工智能搜索方法更具有优势？

这上面的很多议题超出了本书的范围。实际上如 Mitchell（1996）指出的：在遗传算法的基础方面的有待研究的问题远多于它被接受的答案。虽然如此，在遗传算法一开始提出很多研究人员（包括 Holland 1975）就试图去了解它的运行机制。尽管他们是从宏观角度提出的这些问题，就像上面提出的六个问题，但是他们的分析都是从微观和位编码层次进行的。

Holland（1975）引入了*表示模式*（schema）的概念，把它当作一种通用模式和解的“构造块”。表示模式是一个位串的模式，这个模式用由 1，0 和#（不关心）组成的模板描述。例如，表示模式 10##01 表示一个 6 位的字符串集合，它以 10 开头，以 01 结尾。因为中间的模式##代表 4 个位模式 00、01、10 和 11，整个表示模式就表示 4 个 6 位长的 0、1 串。按传统说法，人们常说一个表示模式描述了一个超平面（Goldberg 1989）；在这个例子中，这个超平面截出了所有可能的 6 位长的编码集合。传统遗传算法理论的核心原则是，表示模式是解集合的构造块。遗传算子（交叉和变异）操纵着这些表示模式朝潜在解转换，这个操纵的特殊描述称为*表示模式理论*（schema theorem）（Holland 1975，Goldberg 1989）。按照 Holland 的说法，自适应系统必须识别、测试和合并结构性质，假定这样是为了在一定的环境中产生更好的效果。表示模式就被认为是这些结构性质的形式化。

Holland 的表示模式分析理论认为适应性选择算法应逐步将搜索的焦点定在估计具有最佳适应性的搜索空间的子集上。也就是说，子集由具有较好适应性的表示模式来描述。交叉算子把两

个具有高适应值的构造块放在同一个字符串中，试图产生一个具有更高适应值的新串。变异算子保证了从整个搜索过程中几乎不会消除个体的（遗传）多样性，也就是说，我们将继续探究适应性范围内的新内容。因此遗传算法可被看成是一种拉力，在打开一个通用搜索过程和捕获并保留该搜索空间中的重要（遗传）特征之间维系着紧密的联系。尽管 Holland 对遗传算法搜索的初始分析集中在位层次水平，但他最近更多的工作扩展到了分析多种表示模式上（Goldberg 1989）。在下一节将把遗传算法技术应用于更复杂的表示。

12.2　分类器系统和遗传程序设计

遗传算法的早期工作大都集中在低层次编码方法上，诸如字符串{0, 1, #}。除了支持遗传算子的直观例示，位串和类似的编码方法给遗传算法带来了其他子符号方法（如连接网络）的能力。然而，对于像巡回推销员之类的问题，在复杂一些的层次上有更自然的编码方法。更进一步，我们会问是否有更加丰富的表示法，诸如“如果……那么……”规则和计算机程序代码。这种编码方法的一个重要的特性是它们能通过规则链和函数调用组合不同的、高层次的知识以迎合特定问题实例的需求。

不幸的是，我们很难定义一种这样的遗传算子，它能抓住逻辑关系中的语法和语义结构，而又能有效地应用交叉和变异这样的算子。一种可能的结合规则推理能力和遗传学习的方法是把逻辑句子转换为位串，然后使用标准的交叉和变异算子。但是，在这种转换下，交叉和变异算子产生的位串不能对应到丰富的逻辑语句中。作为一种代替位串的编码方法，我们可以定义各种交叉算子的变形，它们能直接应用到高层的编码中，例如“如果……那么……”规则和高级程序设计语言代码。这一节讨论扩展遗传算法能力的每种方法的例子。

12.2.1　分类器系统

Holland（1986）提出了一个叫做*分类器系统*（classifier system）的问题求解体系结构，它在产生式系统中对规则应用遗传学习。分类器系统（见图 12-3）含有产生式系统中常见的元素：产生式规则（这里叫做分类器）、工作内存、输入传感器（或说解码器）和输出设备（或说受动器）。分类器系统中的不常见特征包括在有冲突决策时的竞争选择，遗传算法用于学习，在学习时对规则进行奖赏和惩罚时使用*桶链算法*（bucket brigade algorithm），从外部环境的反馈提供一种评估候选分类器的方法。图 12-3 中的分类器系统有以下一些主要部分：

1）从环境中得到输入信息的解码器。
2）从环境中得到反馈信息的解码器。
3）把应用规则结果转变到环境中去的受动器。
4）产生式规则集合构成了分类器群体。每一个分类器都有关联的适应度。
5）分类器规则的工作内存。它集成了输入信息激活的产生式规则的结果。
6）用于产生式规则修正的遗传算子集合。
7）对产生成功结果的规则进行奖赏的系统。

在问题求解系统中，分类器系统的执行跟传统的产生式系统一样。从环境中发送一个信息到分类器系统中，例如一个游戏中的移动。这个事件先进行解码，当作一个模式放到内部信息列表中，也就是产生式系统中的工作内存。在通常的数据驱动的产生式系统中，这些信息匹配分类器规则模式的条件。通过投标方法选择“最强的被激活分类器”，这个投标是分类器累计适应值和输入刺激与模式条件的匹配质量的函数。这个最匹配的分类器添加信息（激发规则的动作）到工作内存中。这个修订的信息列表发送信息到受动器中，这个受动器作用于外部环境，当产生

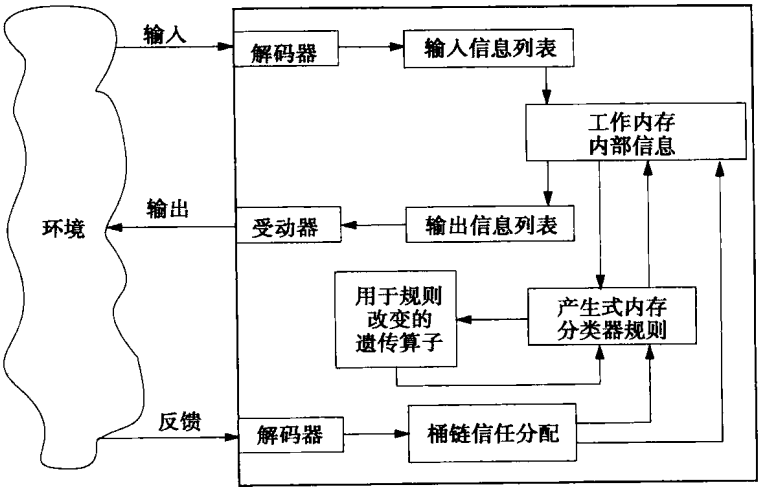


图 12-3　与外部环境交互的分类器系统

［改编自 Holland（1986）］

式系统继续运行时激活新的分类器规则。

分类器系统实现了一种强化学习（10.7 节）。基于教师的反馈或评估函数的适应值，学习器计算候选规则群体的适应值，用遗传学习的变种调整候选群体。分类器系统通过两种方法学习。第一，奖赏系统调整分类器规则的适应度，成功者给予奖赏，错误者给予惩罚。信任分配算法传递部分奖赏或惩罚回分类器规则，这些规则对形成最终的规则激发会有贡献。在相互作用的分类器间的不同奖赏的分配经常是由桶链算法实现。当在系统的输出是一系列规则激发的结果这样的环境时，桶链算法解决信任和惩罚的分配问题。在一个错误事件中，我们怎么知道哪条规则需要惩罚？它是最后被激活的那条规则的责任，还是因为前面的规则提供了错误的信息？根据每条规则对最终结论贡献的大小，桶链算法为一系列规则应用分配信任和惩罚值。在 11.3 节的反向传播算法中提到过一种相似的错误惩罚机制，更多细节可参见 Holland（1986）。

第二，用遗传算子（交叉和变异）改变规则本身。这让成功的规则得以保存，且可组合成新的分类器规则，同时不成功的规则会消失。

每一个分类器规则包括三个部分：典型的产生式系统中，规则条件匹配工作内容中的数据。在学习中，遗传算子能改变产生式规则的条件和动作部分。第二个部分是规则的动作部分，它能改变列表中的内部信息。最后，每一个规则都有一个适应度，这个值会因为成功的动作和不成功的动作而发生改变。在由遗传算子生成规则时会赋予一个适应度。例如，它可能取值为两个双亲的适应值的平均值。

下面用一个简单的例子阐述分类器规则中这些部分之间的相互影响。假定被分类的目标集合有 6 个特征（条件 c1，c2，…，c6），进一步假设每一个特征有 5 个不同的值。尽管每个特征的可能取值是不相同的（例如可能 c3 代表颜色，c5 代表天气），但为了不失一般性，我们给每个特征的赋值为 $\{1,2,\cdots,5\}$。假设这些规则的条件将它们的匹配对象放入以下 4 个类之一：A1、A2、A3、A4。

基于这些限制，每个分类器将有以下形式：

(c1 c2 c3 c4 c5 c6)→Ai，其中 i = 1,2,3,4

其中条件模式中的 ci 表示第 i 个条件的特征值 $\{1,2,\cdots,5\}$。通常，条件特征值也可赋值为#

（不关心），表示可为任意值。Ai 表示分类 A1、A2、A3 和 A4。表 12-2 表示一个分类器集合。注意，不同的条件模式可能有相同的分类，如规则 1 和 2；或者相同的条件模式可对应到不同的分类，如规则 3 和 5。

表 12-2　用于学习的条件→动作分类器集合

条件（特征）	动作（分类）		规则号
(1###1#)	→	A1	1
(2##3##)	→	A1	2
(##43##)	→	A2	3
(1#####)	→	A2	4
(##43##)	→	A3	5
等等			

像上面描述的，分类器系统是通用产生式系统的另一种形式。在这个例子中，分类器规则的真正有新意的地方是用数字串和#表示条件模式。就是这个条件编码方式限制了遗传算法对规则的应用。剩下的讨论描述分类器系统中的遗传学习。

为了简单起见，我们仅考虑在学习分类 A1 过程中分类器系统的性能。也就是说，我们忽略其他的分类，给条件模式赋值 0 或 1 取决于它是否支持分类 A1。注意：这种简化没有损失其一般性；很容易扩展到学习多个分类的系统中，我们只要用一个向量表示特定条件模式匹配的分类。例如，表 12-2 中的分类器可总结如下：

(1 # # # 1 #) → (1 0 0 0)
(2 # # 3 # #) → (1 0 0 0)
(1 # # # # #) → (0 1 0 0)
(# # 4 3 # #) → (0 1 1 0)

在这个例子中，最后一个分类摘要表明条件特征支持分类规则 A2 和 A3，而不支持 A1 或 A4。通过在这些向量中代换 0 或 1，学习算法评估这个规则的适应性。

在这个例子中，我们要用表 12-2 中的规则显示正确的分类，本质上，它们在学习系统中发挥着规则适应性的教师或评估者的作用。跟大多数遗传学习器一样，我们从任意规则群体开始。每一个条件模式赋予了一个强度或适应值参数，取值范围为实数 0.0（无强度）到 1.0（满强度）。这个强度参数值 s 从每个规则的双亲的适应值中计算得出。

在每个学习周期中，规则试图对输入进行分类，然后由教师或适应度量对规则进行排序。例如，在某一个学习周期中，分类器有以下的候选分类规则群体，结论为 1 代表这个模式导致正确的分类，而 0 则表示没有：

(# # # 2 1 #) → 1　　s = 0.6
(# # 3 # # 5) → 0　　s = 0.5
(2 1 # # # #) → 1　　s = 0.4
(# 4 # # # 2) → 0　　s = 0.23

假定从环境中输入一个新的信息（1 4 3 2 1 5），教师（用表 12-2 中第一条规则）把这个输入向量当作规则 A1 的正例。让我们看看当工作内存接受这个模式和四个候选分类器规则试图匹配它时发生了什么。规则 1 和 2 匹配。冲突消解通过匹配规则间的竞争性投标实现。在我们的例子中，投标是匹配属性值的总和与规则强度值的乘积的函数。通配符匹配的值为 0.5，而确切匹配的值为 1.0。为了规一化，我们用结果除以这个输入向量的长度。因为输入向量与第一个分类

器规则是两个确切匹配，四个通配符匹配，它的标值是（（4×0.5+2×1）×0.6）/6，即是0.4。第二个分类器规则也是两个确切匹配，四个通配符匹配，它的标值是0.33。在我们的例子中，只有具有最高标值的分类器规则能激发。但在复杂的系统中，需要一定比例的规则被接受。

第一条规则获胜，于是得到结果1，表明这个模式是A1的实例。因为这个结果是正确的，规则1的适应度将在现在值和1.0之间有一个增量。如果这个规则的结果不正确，它的适应度将会减少。如果系统规则集合的多个规则得到激发，能在环境中产生一定的结果，那么所有对应这个结果的规则都将得到一定程度的奖赏。计算规则的适应值的具体过程随系统的不同而有所不同，有可能很复杂，会用到桶链算法或类似的信任分配技术。详见 Holland（1986）。

一旦候选规则的适应值计算出来了，学习算法就可应用遗传算子产生下一代规则。首先，选择算法将决定规则集合中最适合的成员。这一节是基于适应度测定的，但也可能包括额外的随机值。这个随机值给适应度差的规则复制的机会，避免过早删除这样的规则，它们可能含有结合成期望解的成分。假定例子中的前面两条规则被选择用来存活和复制。任意选择交叉位置为第4和第5个元素之间：

(# # # 2 | 1 #) → 1　s = 0.6
(# # 3 # | # 5) → 0　s = 0.5

产生后代：

(# # 3 # | 1 #) → 0　s = 0.53
(# # # 2 | # 5) → 1　s = 0.57

孩子的适应度是双亲适应度的带权函数。权重值由交叉点的位置决定。第一个孩子有1/3的适应度是0.6的分类器规则，2/3是0.5的分类器规则。因此第一个孩子的适应度是(1/3×0.6)+(2/3×0.5)=0.53。用相同的办法，第二个孩子的适应度是0.57。生成的分类器规则的结果（通常是0或1）与主要属性的规则结果保持相同。在典型的分类器系统中，这两个新规则，还有它们的双亲将构成分类器规则的子集，它们在系统的下一步中将会被执行。

我们也可以定义变异算子。简单的变异规则是任意改变属性模式到其他的合法属性模式；例如，5可能变异成1、2、3、4或#，如我们在遗传算法中讨论的一样，交叉算子试图保持成功的双亲模式中的片段，然后重建新的孩子；而变异算子是保持分类器搜索空间的多样性。

我们的例子是简单的，主要是介绍分类器系统的主要组成部分。在实际的系统中，通常有多条规则激发，每一条规则的结果都会传到工作内存中。为了防止某条规则在求解过程中变得太突出，在每次它赢得投标时都会有一个课税程序减少它的适应值。我们也没有介绍桶链算法，它对支持正确输出信息到环境的规则进行不同程度的奖赏。遗传算子并不是在系统的每一次运行中都会用在规则上，规则的评估、遗传算子的应用通常由一些一般的参数决定，这些参数可能是由环境中得到的反馈的分析结果。

最后，我们的例子摘自 Holland（1986）在密歇根大学提出的分类器系统。密歇根方法被视为是认知的一个计算模型，在其中，认知实体的知识（分类器）被暴露于一个反应环境中，并最终经历了随时间的修改。我们评价整个系统在很长一段时间内都是成功的，而同时单个分类器的重要性相对而言是最小的。替代的分类器系统也经过了审查，其中包括科学家们在匹兹堡大学的工作（Michalski et al. 1983）。匹兹堡分类器的关注点集中于产生新一代分类器的过程中个体规则发挥的作用。这种方法实现了由 Michalski 提出的归纳学习的一个模型。

在下一节我们讨论遗传算法另外一个特别有意思的应用——计算机程序演化。

12.2.2　用遗传算子进行程序设计

尽管在上面几节中遗传算法应用到了更大的表示结构上。从最开始的在位串上的转换进展到了在“如果……那么……”规则上的操作。很自然有人会问遗传算法技术是否可以应用在那些更大规模的计算工具产生的结果上。这里有两个主要的例子：计算机程序生成和有限状态自动机系统的演化。

Koza（1991，1992，2005）认为成功的计算机程序可以通过连续应用遗传算子演化得到。在遗传程序设计中，改编的结构是按层次组织的计算机程序段。学习器保持一个候选程序群体。程序的适应度由该程序解决问题的能力决定，通过对程序子树应用交叉和变异算子可修改程序。遗传程序设计搜索一个不同大小和复杂度的计算机程序空间；实际上，搜索空间是由适合问题域的终结符和函数组成的所有可能的计算机程序空间。像所有的遗传学习器一样，这个搜索是任意的、盲目的，但有令人吃惊的效果。

遗传程序设计从一个初始的任意产生的程序群体开始，这些程序由合适的程序片段组成。这些片段适合问题域，可能包括标准的算术操作、其他相关的程序操作、数学函数以及逻辑的和特殊域的函数。程序的组成部分包括通常类型的数据项：boolean、integer、floating point、vector、symbolic 或 multiple-valued。

初始化后，成千上万的计算机程序开始遗传繁殖。应用遗传算子产生新的程序。交叉算子、变异算子和其他的繁殖算法都必须修改用于产生计算机程序。稍后我们将看到几个例子。新程序的适应度由它在特定的问题领域环境中的执行表现决定。适应度的本质是随问题领域的变化而变化的。任何在执行任务中表现好的程序将会存活，然后产生后一代的程序。

总之，遗传程序设计包括 6 个部分，很多与遗传算法的要求相同：

1）通过遗传算子进行转换的结构集。

2）适合问题域的初始结构集。

3）评估结构的适应度测定，它与领域相关。

4）转换结构的遗传算子集。

5）描述每一代成员的参数和状态描述。

6）终止条件集合。

在下面几段文字中详细讨论这些问题。

遗传程序设计操作按层次组织程序模型。LISP 曾经是（现在还是）程序设计语言成分的主要表达方法：Koza 用 LISP 符号表达式或者 s-表达式（关于 s-表达式的讨论参见补充材料，它的自然表达方式是用树结构）代表程序段。

遗传算子控制 s-表达式。特别是，算子映射 s-表达式的树结构（LISP 的程序段）到新树（新的 LISP 程序段）。尽管 s-表达式是 Koza 早期工作的基础，但其他研究者很多都把这个方法用于不同的程序设计范例语言。

如果可以得到原子片段和问题域的可估计预测，遗传程序设计将可生成很有用的程序。当我们为产生能解决问题的程序创建域时，我们首先必须分析什么是问题求解的终结符，以及需要什么样的函数去产生这样的终结符。就如 Koza（1992，p. 86）所说“……遗传程序设计的使用者必须明白……他所提供的函数和终端的某种构成能产生问题的解。”

为了初始化遗传算子改编的结构，我们必须创建两个集合：函数集 F 和域需要的终结符集合 T。F 可以是简单的{ +，*，-，/}，也可以是更复杂的函数如 sin（X）、cos（X）或者矩阵运算。T 可以是整数、实数、矩阵或更复杂的表达式。T 中的符号必须对 F 中定义的函数是封

闭的。

然后，从集合 F 和 T 的并集中随机选择元素产生初始程序群体。例如，如果我们先从 T 中选择元素，我们就形成一棵只有一个根结点的退化树。更有意思的是，如果我们先从 F 中取，比如是 +，我们就得到一个根结点和两个潜在孩子的树。假定下一步初始化程序从 F 中选择 * 当作该树的第一个孩子，从 T 中选 6 当作第二个孩子。然后任意选择终结符 8 和 F 中的函数 +。最后从 T 中选择 5 和 7。

我们随意产生的程序如图 12-4 所示。图 12-4a 是首次选择 + 后的树结构，图 12-4b 是选择终结符 6 之后的树结构，图 12-4c 是最后的程序。一个类似的程序群体产生了，它构成遗传程序设计的初始化过程。约束集合，例如程序演化的最大深度，能帮助修剪这个群体。这些约束的描述，以及产生初始化群体的不同方法在 Koza（1992）中有详细描述。

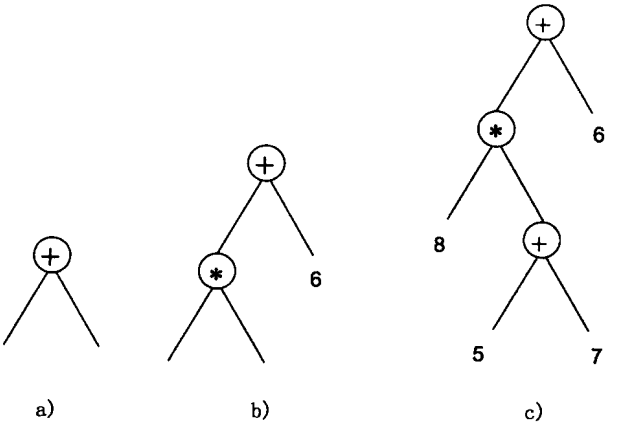


图 12-4　初始化时任意产生的程序

注：圆圈结点是从函数集中得到的

到此我们介绍了表示方法（s-表达式）和程序演化的初始化环境所必须的树结构。然后，我们还需要程序群体的适应度度量。适应度是与问题领域相关的，通常包括一个演化程序执行的任务集。适应度度量本身是一个这样的函数：度量每一个程序在任务集上执行结果的好坏。一个简单的方法是*原始适应性计分法*，它把程序产生的结果与实际结果的差异累加，因此，原始适应性计分法可看成是作用在任务集上的误差之和。当然，其他的适应度度量也是可能的。规一化的适应性是原始适应性分数除以所有可能的误差之和，因此，适应度的范围就是 0 ~ 1。适应度度量也能包含程序大小的调整，如奖励一个小的吝啬程序。

在程序上的遗传算子既包括一棵树本身的转化，也包括在树之间结构的交换。Koza（1992）将主要的转换描述为*复制*和*交叉*。复制是简单地从当前代中选择程序，然后把它（无变化地）拷贝到下一代中。交叉是交换代表两个程序的两棵树的子树结构。例如，假如图 12-5 中的两棵树是双亲程序，在双亲程序 a 和 b 中用“|”表示的任意位置是被选择的交叉点，交叉结果显示在图 12-6 中。交叉也可用于改变单个双亲程序，内部交换双亲程序的两棵子树即可。两个相同的双亲程序任意选择不同的交叉点能产生不同的子树。程序的根结点也可当作交叉点。

还有其他的程序树结构的遗传转化，只是用得比较少。包括*变异*转化只是在一个程序结构的内部进行改变。例如，用另一个数值或者是表示函数的子树代替一个终结符。*置换*转化与在字符串中的反转算子类似，它也是作用于单个程序，交换终结符或者子树。

当前的程序群体反应了解的状态。因为既没有保持回溯的记录，也没有其他方法使之在适

应值曲面上跳跃。从这个角度看，遗传程序设计更像在4.1节描述的爬山算法。遗传程序设计范例在新程序的演化是一个正在继续的过程这一点上带有并行性。不过，时间和资源不是无限的，因此要设置终止条件。它们通常是程序适应度和计算资源的函数。

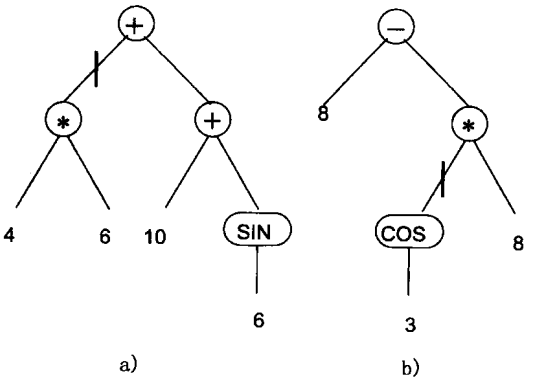


图12-5　基于适应性选出交叉的两个程序

注：a和b中的“|”所示位置是任意选择的交叉位置

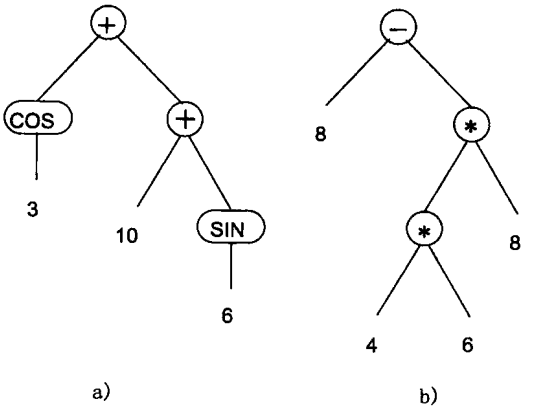


图12-6　图12-5中的交叉产生的子程序

因为遗传程序设计是一种产生计算机程序的技术这个事实，有人把它放入传统的自动程序设计的范畴。在早期的人工智能研究中，研究者就进行了从程序段信息自动产生计算机程序的工作（Shapiro 1992）。遗传程序设计可看作是这个重要领域中另外一种工具。我们用Mitchell（1996）的一个简单的遗传程序设计的例子结束本节。

例3.2.1　演化一个反映开普勒行星运动第三定律的程序

Koza（1992，2005）用遗传程序设计解决了很多有趣的问题，但是很多这样的例子对我们现在来说太复杂。然而，Mitchell（1996）创建了一个简单的例子阐述遗传程序设计的很多概念。开普勒行星运动第三定律描述的是行星运行周期P和它到太阳的平均距离A之间的函数。

开普勒第三定律的函数是：

$P^2 = cA^3$　其中c是常数

如果我们假设P用地球年作为单位，A用地球到太阳的平均距离作为单位，则c=1。这个关系的s-表达式是：

P=(sqrt(*A(*A A)))

因此，我们想得到的程序可用图12-7所示的树结构表示。

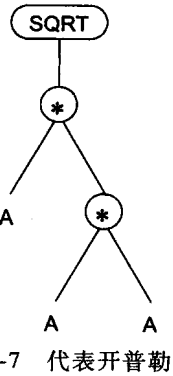


图12-7　代表开普勒第三定律的轨道和周期关系的目标程序

在这个例子中终结符号集的选择是很简单的，它是由A给定的一个实数。函数集合也相对比较简单，就是{+，−，*，/，sq，sqrt}。下一步我们要创建开始的随机程序群体。初始程序群体可包括：

(* A (− (* A A) (sqrt A)))	适应度：1
(/ A (/ (/ A A) (/ A A)))	适应度：3
(+ A (* (sqrt A) A))	适应度：0

（我们马上解释附加适应度度量）。在这一节的前面提出过，初始群体在大小和深度方面有一定的限制，且与问题领域的知识相关。这三个例子用图12-8中的树结构表示。

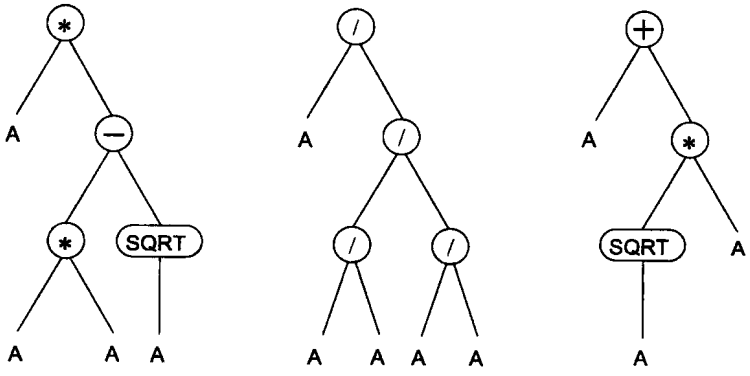


图 12-8　解决轨道周期问题初始程序群体中的成员

下一步我们确定程序群体的合适测验。假定我们知道一些行星的相关数据。例如，表 12-3 中的行星数据来自 Urey（1952），它给出了一个我们的演化程序必须解释的数据点集。

表 12-3　适应度事例集合，行星数据来自 Urey（1952）

注：A 是地球轨道的主半轴，P 是用地球年作为单位

行星	A（输入）	P（输出）
Venus	0.72	0.61
Earth	1.0	1.0
Mars	1.52	1.87
Jupiter	5.2	11.9
Saturn	9.53	29.4
Uranus	19.1	83.5

因为适应度度量是我们要解释的数据点的函数，我们定义适应度为程序的输出个数，在正确输出值 20% 的范围内。我们用这个定义去创建图 12-8 中三个程序的适应度度量。剩余的工作还有：生成更多的初始群体成员；创建能产生新一代程序的交叉和变异算子；确定终止条件。这些都留给读者去完成。

12.3　人工生命和基于社会的学习

在这一节的开始，我们先介绍一个“生命游戏”的简单版本。这个游戏在计算机视觉模拟中经常出现，视觉模拟中连续生成的各代快速变化，然后在屏幕上显示。它首先由数学家 John Horton Conway 提出，通过 Martin Gardner 在《Scientific American》上的讨论，它名声远扬。生命游戏是一个叫做*细胞自动机*（CA）的计算模型的简单例子。细胞自动机与简单的有限状态自动机相似。

定义（*有限状态自动机或细胞自动机*）

1）一组**输入字母** I。

2）一组自动机可能**状态** S。

3）指定状态 s_0 为**初始状态**。

4）**下一状态函数** $N: S \times I \rightarrow S$，为含有当前状态和当前输入的有序对指定下一个状态。

在3.1节曾介绍过，有限状态自动机（FSM）的输出是一个关于当前状态和输入值的函数。细胞自动机将当前状态的输入定义为“相邻”状态的函数。因此，在时刻（t+1）的状态是它本身在时刻t的状态和它的邻居的时刻t的状态的函数。通过与邻居的相互作用，细胞自动机集合与简单定义的单个自动机相比可以有更加丰富的行为。因为状态的输出是相邻状态的函数，我们称这种相邻有限状态自动机集合的演化是基于社会的适应和学习。

对于这一节描述的社会，没有外在的对于个体成员的适应性评估。群体中个体的相互作用会产生适应性，相互作用可导致单个自动机“死亡”。适应性是隐含在个体从一代存活到另一代的过程中的。细胞自动机中的学习是典型的无监督学习。像自然进化一样，适应性是由群体中其他的共同进化成员的作用形成的。

全局的或面向社会的观点让我们对学习获得重要的看法。我们不再把注意力集中在单个个体上，而是看整个社会的不变性和呈现的规律。这是12.3.2节将介绍的Crutchfield－Mitchell研究的重要方面。

最后，不像有监督学习，进化不需要目的性。也就是说，个体组成的社会不需要认定是“到哪儿去”，或说到达某个终点。在这一章的前几节我们用到显式的适应值度量时，我们有收敛的偏置。但是Stephen Jay Gould（1977，1996）指出，进化不应该被看作是使事情“变得更好”，而仅仅是应该适应生存。惟一的成功是继续生存，出现的模式就是社会模式。

12.3.1　生命游戏

考虑图12-9中的简单的二维网格或说是游戏板。这里有一个方框被占用（有一个位值1），涂成了黑色，它的8个邻居用灰色阴影表示。在游戏过程中这个板面会发生变化，在时刻t+1任意方框的状态是它本身在时刻t的状态和它邻居在时刻t的状态的函数。三条简单的规则可让游戏进化：第一，任意方框，不管占用与否，如果恰好有三个邻居被占用，在下一时刻它将被占用；第二，任何被占用了的方框如果恰好有两个邻居被占用，它在下一时刻将继续被占用；最后，对于所有其他的情形，下一时刻都不占用。

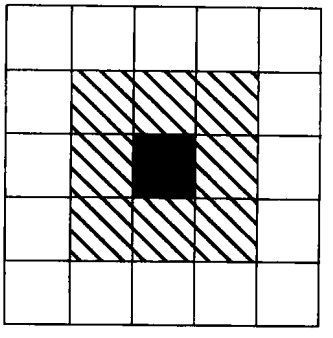


图12-9　阴影区域代表“生命游戏”中的相邻集合

对于这些规则的一个解释是：对于每一代或时间周期，任何位置的生命也就是说方框是否被占用或状态值是否为1，它是本身和在当前时刻邻居的状态相互作用的结果。特别是，任何时刻如果邻域群体的密度太大（超过三个）或太小（少于两个），则在下一时刻将不会有生命出现。

例如，考虑图12-10a中的状态。这里只有两个方框（用x标出）恰好有三个被占用了的邻居。在下一生命周期产生图12-10b。同样这里只有两个方框（用y标出）恰好有三个被占用了的邻居。显而易见，这个场景的状态在图12-10a和图12-10b之间循环。读者可算出图12-11a和图12-11b下一时刻的状态。Poundstone（1985）描述了一个非常丰富的具有多样性的结构，它可出现在生命游戏中，例如“滑行器”，它是一个细胞模型，它的形状在生命周期中会不断变化，详见图12-12。

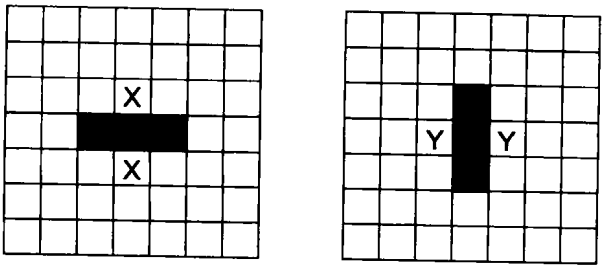


图 12-10　产生闪光现象的邻域集合

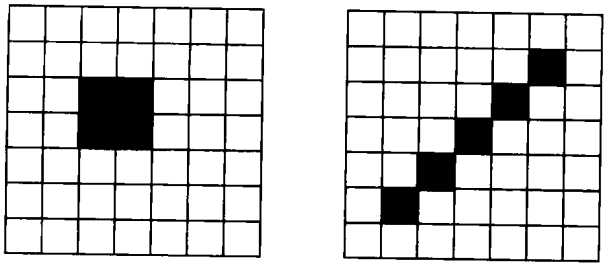


图 12-11　在下一个生命周期中这些模式会发什么变化

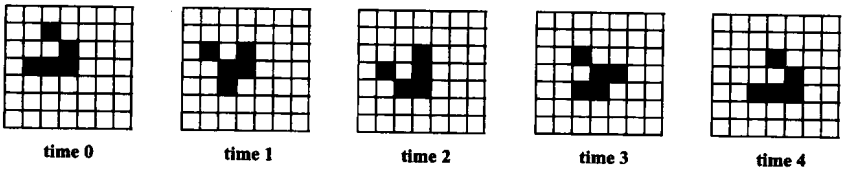


图 12-12　“滑行器”在屏幕中移动

因为细胞自动机能通过简单细胞的相互作用产生丰富的行为集合，它被证明是研究从简单的无生命成分进化到生命的数学方法的重要工具。人工生命定义为通过人工的努力而不是自然力量产生生命。从前面的例子可看出，人工生命有很强的“冒出”的味道，也就是说，生命系统的原子被定义和组合，然后它们物理的相互作用就出现。这种生命形式的规则由有限状态自动机捕获。

但是一个生命结构有什么用？例如，在生物学上，自然界产生的生命实体集合可能很分散，很复杂，它们被意外的历史的偶然性所统治。我们相信在产生这些生命集时有逻辑的规则，但是它们不需要存在。并不是我们把眼光限定在自然界实际提供的生命集合上，我们就能发现很多那些可能的规律。探索整个可能的生物规律是很重要的，部分因为历史的意外而被消除。我们经常感到疑惑，如果恐龙不意外地消失的话我们现在的世界会怎样。为了发现实际的规律，去理解可能性的限制条件是必须的。

除了人类学家和其他科学家的努力填补了实际进化知识的空白，也有人思索重现进化本身的过程。如果进化从不同的初始条件出发，将会发生什么事情？如果在我们的物理和生物环境中出现意外事故，将会出现什么事物？什么事物会保持不变？在地球上发生的实际进化路径是多种可能情况中的一种。如果我们能产生很多可能的生物个体，那么这些问题就可能得到解答。（见 12.3.3 节中可编程人工细胞演变（Programmable Artificial Cell Evolution，PACE）的例子。）

人工生命技术不仅仅局限于计算机产生的人造物或生物领域。来自不同领域（诸如化学和药理学）的科学家构造了综合人造体，很多与我们现实中实体的知识相关。例如，在化学领域，对物质内部结构和很多自然界提供的化合物的研究使我们能够分解化合物、化合物组成成分以及化合物的化学键。这种分解和重新组合产生了很多自然界中本来没有的化合物。可见，通过对自然界中化合物的仔细分析，我们开始了解那些可能的化合物集合。

了解可能世界的一个工具是在运动和相互作用影响的基础上模拟和分析社会。我们在生命游戏中有一个简单的例子。图 12-12 展示的时间周期序列实现了我们前面提到的“滑行器”。滑行器经过游戏空间时只有很少的几个模式在循环。在四个时间周期中，它的行动是简单地移动到一个新位置：整个阴影图形往右移动一格，往下移动一格。

生命游戏的一个有意思的特征是像滑行器这样的实体在与社会的其他成员相互作用之前它能保持不变；相互作用后会发生什么就很难了解和预测了。例如，在图 12-13 中，我们可看到这样的情形，两个滑行器出现和接近。经过四个周期后，向下边和左边移动的滑行器被另外一个实体“消灭”了。看看我们的本体论描述是很有意思的，也就是说，我们用“实体”、“闪光物”、“滑行器”和“消灭”这些词的时候，反映了我们在观察生命形式和相互作用时，不管是人工的还是自然的，我们都带有以人为中心的偏见。

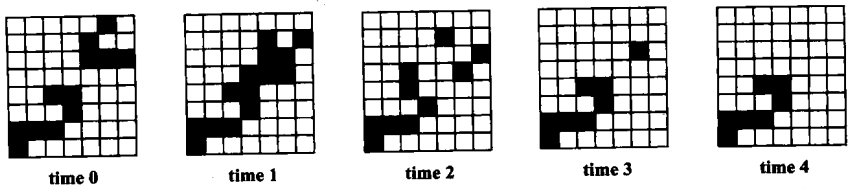


图 12-13　一个“滑行器”被另一个实体消灭

12.3.2　进化规划

“生命游戏”是细胞自动机的一种直观的、高度可描述性的例子。通过将细胞自动机描绘成有限状态自动机，可以推广我们关于细胞自动机的讨论。现在讨论由多个 FSM 连接的社会，并把它们作为突现的实体来进行分析。这个研究有时也叫做进化规划（evolutionary programming）。

进化规划的历史可追溯到计算机本身的开端。John von Neumann 于 1949 年在一系列的讲座中提出这样的问题：当自我复制出现时需要什么层次的有组织的复杂性（Burks 1970）。Burks 引述 von Neumann 的目标：“……不是试图模拟自然界中在遗传和生物化学层次上的自我复制，他希望抽象出自然界自我复制问题的逻辑形式。”

避开化学的、生物的和机械的细节，von Neumann 能表示出自我复制的本质需求。von Neumann 继续努力想设计出（一直没有做到）一种自我复制的机器，它包括一个两维的细胞组织结构，这个组织包含很多有 29 个状态的自动机，对于每一个自动机，它的下一个状态是它本身的当前状态和它的四个直接邻居的状态的函数（Burks 1970，1987）。

有意思的是，von Neumann 设计的自我复制机器据估计包含至少 40 000 个细胞，它有通用图灵机的功能。这种通用计算装置也是一种通用构造，因为该装置能够读出纸带中的输入，解释纸带中的数据，并通过使用一个机械手臂，在未被占用的细胞空间中创建纸带中描述的配置。通过把构造自动机本身的描述输入纸带中，von Neumann 创建了能自我复制的自动机（Arbib 1966）。

后来 Codd（1968）减少了可计算通用自复制自动机中所需要的状态数，从 29 个减少到了 8 个，但是整个设计估计需要 100 000 000 个细胞。后来，Devore 简化了 Codd 的机器，只占用 87 500个细胞。现在，Langton 创建了一个不具有通用可计算性的自我复制的自动机，它占用 100

个细胞，每个细胞仅有 8 个状态（Langton 1986，Hightower 1992，Codd 1992）。这个研究领域的当前进展可在人工生命会议的论文集中发现（Langton 1989，Langton et al. 1992）。

因此，自复制自动机的形式分析在计算理论中有很深的根源。也许更令人兴奋的结果隐含在人工生命形式的经验研究中。这些程序的成功不是由某个先验的适应度函数来表示，而是由它们能存活和复制的简单事实来表示。成功的标志是它们能够存活。这些程序也有不光彩一面，我们经历过计算机病毒和蠕虫，它们能进入外部主机，复制它们本身（通常会摧毁在内存中的复制所需要的信息），并移动继续感染其他的外部主机。

我们通过讨论几个人工生命计算（a-life computing）的研究课题来总结一下本节。第 12 章最后一节有一个应急运算（Sante Fe Institute 的 Melane Mitchell 以及他的学生）的例子。我们先讨论两个本书先前介绍过的课题，它们是由 MIT 的 Rodney Brooks 和斯坦福大学的 Nils Nilsson 及他的学生提出的。在稍早一些的论述中，Brooks 的工作是在表示的一般标题下进行的（见 6.3 节），而 Nilsson 的工作是在规划主题下进行的（见 7.4.3 节）。在本章中，我们要在人工生命和涌现现象的上下文中再次讨论这两个课题。

MIT 的 Rodney Brooks（1991a，b）在人工生命的前提上创建了一个研究课题，也就是通过很多的简单自治的主体的相互作用产生智能。Brooks 的方法常常被描述成是“没有表示的智能”。Brooks 制造了一系列的机器人，它们能感知障碍物，并在 MIT 的办公室和走廊中走来走去。基于包容结构，这些主体能徘徊、探索和避开其他的目标。这个系统的智能是简单组织和与环境相互作用的结果。Brooks 说：“我们把有限状态自动机捆绑成控制器的控制层。每个层都在已存在层顶部创建。低层次的构造层从来不依靠高层次的构造层的存在。”更多文献包括：McGonigle（1990，1998），Brooks（1987，1991a），Lewis and Luger（2000）。

Nils Nilsson 和他的学生（Nilsson 1994，Benson 1995，Benson and Nilsson 1995）为主体控制设计了一个 teleo-reactive（T-R）程序，这个程序不断考虑改变的环境因素，为主体提供向目标前进的方向。这个程序的运行非常类似产生式系统，但也支持持续动作，或是在任意的时间周期中发生的动作，例如“向前直到……”。因此，与普通的产生式系统不同，条件值必须被不断地计算，与当前最准确的条件关联的动作通常是一个要执行的动作。对于进一步的细节，感兴趣的读者可参见 Nilsson（1994）、Benson（1995）、Benson 和 Nilsson（1996）、Klein 等（2000）。

欧洲联盟委员会正在支持 PACE（可编程人工细胞演变）项目。现在已研究出新一代合成化学细胞。关键点在于生命是在生命计算系统中进化的，并且是自组织与自修复的。而完成的人造细胞也是可以自我繁殖和进化的，并且它们可以用更为简单的环境资源来维持复杂的结构。

斯坦福大学的 John Koza 又公布了一些遗传程序设计方面的研究成果。最近的成果包括集成电路和控制器的自动发展，电路的拓扑、测量和布置路线的合成，以及其他人造工件。相关信息可参见 Koza（2005）。

这四项研究成果是庞大的基于自动机的研究项目中的样本。这些项目是基础性的实验。它们会提出现实世界中的一些问题。现实世界中成功的算法能存活，进一步成长，同时不具有适应性的系统将会消亡。

最后，我们考虑来自圣达菲研究所的一个研究：涌现的实例研究。

12.3.3　涌现的实例研究

Crutchfield 和 Mitchell 探索在简单系统间进化和相互作用的能力，以便产生处理相互关系的高层的集合信息。他们的研究提供了一个涌现（在进化或遗传算法的支持下）的例子，它是一个空间系统全局计算的实例的涌现，这个系统由分布的和局部的相互作用的细胞或处理器组成。

术讲涌现计算（emergent computation）描述的是这些系统中处理各个结构的全局信息的涌现。Crutchfield 和 Mitchell 的目标是描述这样的一种体系结构和机制：它能充分进化和支持涌现计算的方法。

细胞自动机由很多单个细胞组成；实际上，我们例子中的每个自动机都含有 149 个细胞。这些二值状态的细胞分布在一维空间中。每个细胞状态的改变是它本身的状态和它的两个直接邻居的状态的函数。细胞自动机随着时间周期不断进化，形成一个二维的方格。这个方格从任意产生的 N 个细胞集合开始。在图 12-14 的例子中，149 个细胞进行最开始的 149 个周期的进化（从 0 周期开始，用数字 0、1、…、148 表示细胞）。这些细胞自动机行为的两个例子如图 12-14 的时空图所示。在这个图中，1 用黑色细胞表示，0 用白色细胞表示。当然，表示细胞邻域的不同规则将会在细胞自动机的时空图中产生不同的模式。

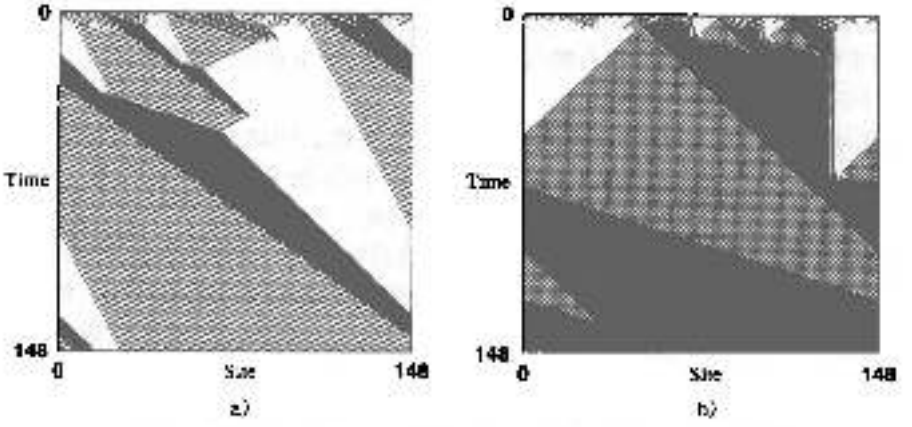


图 12-14 显示两个细胞自动机行为的时空图，用遗传算法发现规则

对于非局部计算或一般涌现模型，它们使用很大数字。每一个时空图在一定范围的时间周期内循环，1 用黑色细胞表示，0 用白色细胞表示；时间沿向下的方向递增，引自 Crutchfield 和 Mitchell（1995）下一步，我们描述规则集合，它们决定组成细胞自动机的细胞行为。图 12-15 表示了一个一维二值状态的近邻细胞自动机，有 N = 11 个细胞。图中举例说明了方格和更新方格的规则表。方格每经一步都会发生变化。方格实际应该是圆柱形的，它的最左边和最右边的细胞是邻居（应用规则集时这一点很重要）。它的规则表支持局部“多数投票”规则：如果邻域的三个细胞大部分是 1，则中间的细胞在下一时刻就是 1；否则，它在下一时刻就变成 0。

Crutchfield 和 Mitchell 想找到一种自动机，它能执行以下的计算，这里叫做“多数赢”：如果初始的方格大多数都是 1，则细胞自动机最终进化到全 1；否则，它进化到全 0。这个研究的一个有趣问题是很难设计支持“多数赢”计算的细胞自动机规则。实际上，在 Mitchell 等（1996）中表明，简单的 7 邻居的“多数投票”规则不能执行“多数赢”计算。遗传算法可用来搜索这个规则。

12.1 节的遗传算法对于细胞自动机的不同试验创建相应的规则表。特别是，可以使用遗传算法来进化针对一维的二值状态细胞群体（组成每个细胞自动机）的相应规则。适应性函数用于奖赏那些支持“多数赢”策略的规则。因此，最后遗传算法创建一个规则集合，它的适应性是一个加强全局多数规则以最终成功的函数。适应性好的规则得到存活，并通过交叉组合产生后代，每一个子孙都会经历一个小概率的变异算子。这个过程循环 100 代，在每一代中对于新的初始细胞集评估适应值。详细的内容参见 Crutchfield 和 Mitchell（1995）。

我们怎样量化更成功的细胞自动机支持的涌现计算？就像很多空间扩展的自然过程一样，

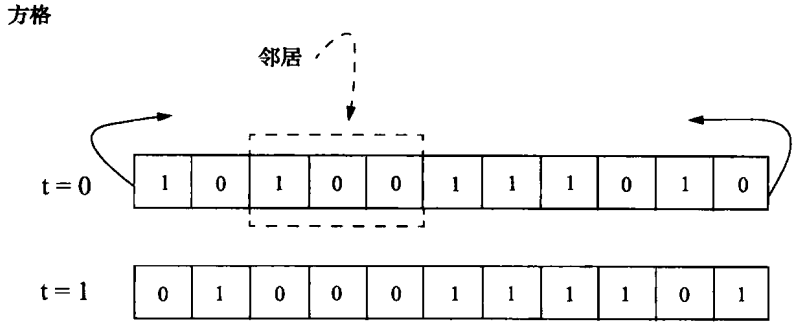


图 12-15　阐述一维的两状态最近邻细胞自动机，其中 N = 11

注：图中举例说明了方格和更新方格的规则表。还画出了经过一个时间周期的方格结构细胞自动机是圆形的，因为两端的值是邻居

细胞结构经常从随时间变化进行组织扩展到随与时间同构的空间区域变化进行组织。分析和决定执行规则应该是一个自动的过程。事实上，Hanson 和 Crutchfield（1992）创立一种用于最小确定性有限自动机的语言，用于描述每个细胞自动机内部的吸引基（attractor-basin）。这种语言可用来描述我们的例子。

有时，在图 12-14a 中，不变区域对于观察者来说是很显而易见的，也就是说，相同的模式在该区域反复出现。我们标志这样的区域为 L，为了更好地描述相互作用影响的计算，我们过滤这些不变元素。表 12-4 描述了三个 L 区域：L^0，重复的 0 区域；L^1，重复的 1 区域；L^2，重复的模式 10001 区域。在图 12-14 中还有其他的 L 区域，但是我们只讨论这个子集。

表 12-4　规则域、粒子（域边界）、粒子速率（圆括号中的数）和图 12-14a 的细胞自动机的空间 - 时间行为的粒子相互作用的目录。记号 $p \sim \Lambda^x\Lambda^y$ 表示 p 是构成规则域 Λ^x 和 Λ^y 之间边界的粒子

规则域		
$\Lambda^0 = 0^*$	$\Lambda^1 = 1^*$	$\Lambda^2 = (10001)^*$
粒子（速率）		
$\alpha \sim \Lambda^1\Lambda^0$ (1)	$\beta \sim \Lambda^0\Lambda^1$ (0)	
$\gamma \sim \Lambda^2\Lambda^0$ (−2)	$\delta \sim \Lambda^0\Lambda^2$ (1/2)	
$\eta \sim \Lambda^2\Lambda^1$ (4/3)	$\mu \sim \Lambda^1\Lambda^2$ (3)	
相互作用		
衰减	$\alpha \rightarrow \gamma + \mu$	
反作用	$\alpha + \delta \rightarrow \mu$，$\eta + \alpha \rightarrow \gamma$，$\mu + \gamma \rightarrow \alpha$	
湮灭	$\eta + \mu \rightarrow \phi_1$，$\gamma + \delta \rightarrow \phi_0$	

过滤掉 L 域中的不变元素后，能看到这些域间的相互作用。在表 12-4 中，描述了 6 个 L 域的相互作用，例如，在 L^1 和 L^0 的交界处的粒子。所有 1 域和 0 域的交界处叫做嵌入粒子 α。Crutchfield 和 Mitchell 认为嵌入粒子的集合是在长期的时间和空间连续区间中传送信息（或信号）的主要机制。当这些粒子碰撞时，逻辑操作就会作用在它们上。因此，细胞自动机的域集合、域边界、粒子和粒子的相互作用代表了基本的信息处理元素，它们嵌入在细胞自动机的行为中，也

就是细胞自动机的固有计算中。

例如，图 12-16 描述了图 12-14a 的涌现逻辑。为了域边界更容易观察，L 域中不变的内容被过滤了。图 12-16 中每一个被扩大的区域显示了两个相互作用的嵌入粒子的逻辑。粒子相互作用 $\alpha+\delta\rightarrow\mu_\alpha$ 显示在图的右上方，它实现了一个这样的逻辑：把代表空间结构的信号 α 和 δ 映射到信号 μ_α。类似细节可见粒子相互作用 $\mu+\gamma\rightarrow\alpha$，把代表空间结构的信号 μ 和 γ 映射到信号 α。图 12-16 的更完全的粒子相互作用表可见表 12-4。

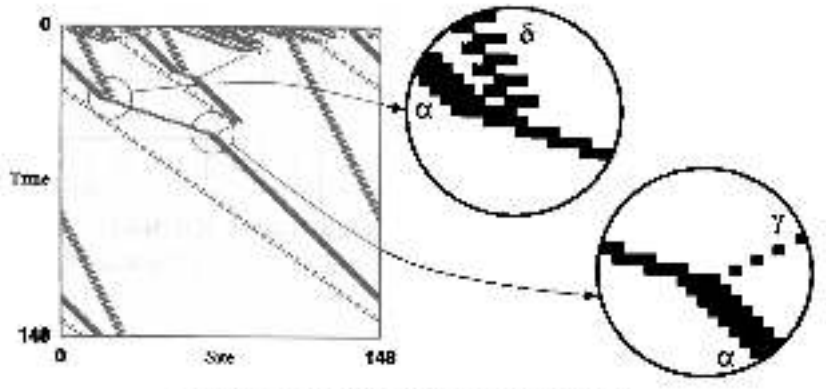


图 12-16　对应图 12-14a 中涌现逻辑分析

注：这个细胞自动机有 3 个域、6 个粒子、6 个粒子循环，如表 12-4 所述。域已经用一个 18 状态的非线性转移算法进行了过滤；引自 Mitchell 等（1994）

总结一下，Crutchfield－Mitchell 研究的一个重要结果是发现了一种方法，它在空间分布的系统中可描述涌现计算，这个系统包括局部相互作用的细胞单元。在细胞间通信的局部性限制了全局通信。遗传算法的作用是去发现局部细胞规则，这个规则的效果是完成跨越“大块”空间－时间距离的信息处理。Crutchfield-Mitchell 使用多个取自形式语言理论的想法来刻画空间－时间模式。

对于 Crutchfield 和 Mitchell，进化自动机的结果反映了一个完全新层次的行为，它是与分布的细胞之间低层次的相互作用不同的。全局的基于粒子的相互作用显示了在简单的个体行为集合中会出现多么复杂的协调性。作用在局部细胞规则上的遗传算法结果表明一个进化过程怎样产生新层次的行为。精巧的平衡对于涌现计算的有效性是必要的。

Crutchfield 和 Mitchell 的研究成果是很重要的，在遗传算法的支持下，他们演示了在细胞自动机间高层次的不变性的涌现。更进一步，他们根据形式语言理论进行改写，提出了计算工具，它能用于描述这种不变性。更进一步的研究可去阐述复杂性的涌现：生命物体特征的定义和理解心智、物种、生态系统起源的基础。

12.4　结语和参考文献

遗传算法和基于生物学习的研究是从 John Holland 的遗传算法设计开始的。他的早期研究包括：《Adaptation in Natural and Artificial Systems》（1975）、一个关于自我复制系统涌现现象的研究（1976）和介绍分类器的《Escaping Brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems》（1986）。遗传系统分析方面工作的例子可从 Forrest 和 Mitchell（Forrest 1990，Mitchell 1996，Forrest and Mitchell 1993a，1993b）中找到。其他研究者，尤其是 Goldberg（1989），Mitchell（1996）和 Koza（1992，1994）继续了在遗传算法和学习形式分析方面的工作。

前面提到过，Holland（1986）对分类器系统的设计也做出过贡献。分类器创立了一个宏观或完整系统的学习视角。另外一个类似的观点在 SOAR 项目中有所体现（Laird et al. 1986a, 1986b; Rosenbloom and Newell 1987; Rosenbloom et al. 1993）。

很多热门书籍，包括 James Gleick（1987）写的《Chaos》，已经建立了涌现和混沌系统的重要性。这些系统中的概念（例如，奇怪的吸引子）已经成为连接体系结构的一部分了（见第 11 章）。

John Koza 是遗传程序设计的研究领域的奠基者。他的主要贡献体现在：《Genetic Programming: On the programming of computers by means of natural selection》（1992）和《Genetic Programming II: Automatic discovery of reusable programs》（1994）。在 12.2.2 节中用遗传程序设计学习开普勒第三定律的例子是由 Mitchell（1996）提出的。

生命游戏最早由数学家 John Horton Conway 提出，但是 Martin Gardner 在《Scientific American》（1970, 1971）中提出讨论后，它才名声远扬。有限状态自动机的计算能力的研究可追溯到第一台数字计算机的设计。John von Neumann 在这一领域非常活跃，实际上，也是他第一个证明了有限状态自动机有和图灵机一样的计算能力。von Neumann 的很多早期研究可见 Arthur Burks（1966, 1970, 1987）的著作。其他的研究者（Hightower 1992, Koza 1992）描述了人工生命研究怎样从早期的有限状态自动机的工作取得进展。研究人工生命的其他研究者包括：Langton（1986）、Ackley 与 Littmann（1992）。早期人工生命会议的论文集都由 Langton（1989, 1990）编录。

Dennett 著有《Darwin's Dangerous Ideas》（1995）和《Sweet Dreams》（2006），他和其他的哲学家从哲学思想角度阐述了一些进化概念的重要性。另外我们推荐《Full House: The Spread of Excellence from Plato to Darwin》（Gould 1996）。

除了 12.3 节简要介绍的主体研究（Brooks 1986, 1987, 1991a, 1991b; Nilsson 1994; Benson and Nilsson 1995），这个领域还有其他项目，包括 Maes（1989, 1990）在行为网络中伸展激活的模型，以及 Hayes-Roth 等（1993）的黑板体系结构的扩展。AAAI、IJCAI 和人工生命会议的会议记录包括这个重要研究领域的很多论文。Crutchfield 和 Mitchell（1995）支持我们在 12.3.3 节的描述。

12.5　习题

1. 遗传算法试图支持搜索时的遗传多样性，同时又要保持重要特征（用遗传模式表示）的存活，描述一种不同的遗传算子，它同时支持这两个目标。
2. 讨论为了搜索不同域中的解时，遗传算子的表示设计问题。*归纳偏置*在这里的作用是什么？
3. 考虑 12.1.1 节中的 CNF - 可满足性问题。CNF 中的析取的数目在解空间中有什么样的影响？考虑其他的表示方法和遗传算子。你能设计出其他适应度吗？
4. 设计遗传算法解决 CNF - 可满足性问题。
5. 考虑 12.1.1 节中的巡回推销员问题。讨论这个问题的合适编码问题，并设计针对这个问题的其他合适的遗传算子和适应度度量。
6. 构建一个遗传算法搜索巡回推销员问题的解。
7. 论类似于葛莱编码的编码技术在改造遗传算法搜索空间中的作用。描述两个其他的问题域，类似的编码技术在其中是很重要的。
8. 阅读 Holland 的表示模式理论（Mitchell 1996, Koza 1992）。考虑 Holland 的表示模式理论是怎样描述遗传算法解空间的进化的？为什么必须说问题不能用位串编码？
9. 桶链算法（Holland 1986）是怎样与反传算法关联的（见 14.3 节）？

10. 用 12.2.2 节描述的基本的编码方法写一个程序来解决开普勒第三定律的运动问题。
11. 讨论在用遗传算法进行程序设计来解决问题时的约束（见 12.2.2 节）。例如，解中的什么成分不能用遗传程序设计范例进化？
12. 读《Scientific American》（1970，1971）中 Gardner 专栏里关于生命游戏的早期讨论，并讨论其他的类似于 12.3.1 节中的滑行器的人工生命结构。
13. 写一个人工生命程序，实现图 12-10 到图 12-13 中的功能。
14. 在 12.3 节中介绍了基于主体研究领域，建议对这个课题更深一步了解，尤其是 Brooks、Nilsson 和 Benson 或 Crutchfield 和 Mitchell 的研究。请从中选择一个主题写一篇短论文。
15. 讨论第 12 章中，学习模型的归纳偏置方法在表示、搜索策略和算子中的作用，讨论这个问题是否是可解的？也就是说，遗传学习能工作是不是因为它的典型假设？还有，它能否应用于更广泛的领域？
16. 阅读和讨论《Darwin Dangerous Idea》（Dennett 1995）或者《Full House：The Spread of Excellence from Plato to Darwin》（Gould 1996），以便进一步了解进化和复杂性涌现。

第 13 章　机器学习：概率理论

概率论就是化为计算的判断力……

——LaPlace

计算的目的是深刻理解，并非是计数……

——Richard Hamming

13.0　学习中的随机模型和动态模型

正如前面的章节所述，使用概率工具来理解和预测现实中的现象有两个主要原因：其一，事件之间确定具有概率关联关系；其二，在变化的世界中，情况间确定性因果关系可能很复杂，因而它们之间的相互作用最好通过随机模型处理。根据以上两点，人工智能（AI）领域采用概率模型，并且，随机技术在机器学习算法的设计、能力和灵活性上产生了深远的影响。

在 5.3 节中首次阐述的贝叶斯规则是机器学习概率模型的基础。贝叶斯方法是基于先前学习到的关系来对新经验加以解释：它把对当前事件的理解看作一个从先前事件中学习得到的期望的函数。同样，马可尔夫模型及其变形给随机学习方法提供了数学基础。在马尔可夫链中，事件在任何时间点的发生概率是一个关于该点之前时刻发生的事件概率函数。在 9.3 节介绍的一阶马尔可夫链中，时间点 t 的事件发生概率是它之前的 t－1 时刻事件概率的函数。

第 13 章包含三节：13.1 节继续阐述马尔可夫模型，介绍隐马尔可夫模型（HMM）及马尔可夫模型的几个重要变形和扩展。13.2 节对 9.3 节中的贝叶斯网络进行了扩展，集中介绍了动态贝叶斯网络（DBN）及其若干扩展。13.3 节是 10.7 节关于强化学习的后续，并且为强化引入了概率化度量。

13.1　隐马尔可夫模型（HMM）

在 20 世纪初，俄国数学家马尔可夫（Andrey Markov）就提出了一种用于对概率相关的离散事件进行建模的数学方法。与当计算机执行一系列固定的指令集时，人们期待看到的确定序列事件的情形相比，马尔可夫的形式方法支持这样一种观点，即每一个事件与它之前发生的事件是概率相关的，就像音素模型或口语中的单词。在计算机通过程序学习变化世界中的现象的尝试中，马尔可夫的观点被证明是极为重要的。

13.1.1　隐马尔可夫模型的介绍和定义

隐马尔可夫模型是 9.3 节中介绍的传统马尔可夫链（或马尔可夫过程）的一般化形式。传统马尔可夫链中的每一个状态都对应一个离散的物理上可观测的事件，例如一天中某个时刻的天气。但是，这类模型有很多限制条件，现在我们把它推广到更加广泛的一类问题上。在这一节中，我们把传统马尔可夫模型扩展到观察数据当前隐藏状态本身的概率函数。所以，这个被称为隐马尔可夫模型的模型是一个双重嵌入的随机过程。

隐马尔可夫模型可以描述为由一个不可观测的、隐含的随机过程支持的可观测的随机过程。使用 HMM 的一个例子就是通过对有噪声的声学信号的解释来实现单词识别的计算机。语音模型本身（即说话者想要使用的语言中产生特定单词的一部分音素）构成了马尔可夫模型的隐含层次。这些观察数据，即所听到的有噪声的声学信号是这些可能音素的随机函数。这个说话者使用

的音素只能通过噪声声学信号的顶层（可观察）流被随机地观察到。如下定义一个 HMM：

定义（隐马尔可夫模型）　一个图模型被称作**隐马尔可夫模型**，如果一个马尔可夫模型的状态是隐藏的且不能直接观察，并通过另外的随机系统解释它的输出，那么这个模型就称为**隐马尔可夫模型**。更形式化的定义是，给定一个状态集合 $S=s_1, s_2, \cdots, s_n$，以及一个状态转换概率集合 $A=a_{11}, a_{12}, \cdots, a_{1n}, a_{21}, a_{22}, \cdots, \cdots, a_{nn}$，得到观测的似然值的集合 $O=p_i(o_t)$，集合中的每一个元素表示在观察 o_t（时间 t）由状态 s_t 产生的概率。

现给出改编自 Rabiner（1989）的 HMM 的两个例子。首先，考虑抛硬币的情形。假设一个人在房间里抛硬币，我们不知道在房间中发生的情况。这个人可能每次只抛一个硬币，也可能从很多个硬币中随机选择几个抛，每一个硬币都可能有自己的偏置输出，即出现两面的概率不等。我们在房间外面所能观察到的是一系列抛硬币的结果，例如，观察到 O = H, H, T, H, T, H, …。我们的任务就是设计一个模型，它可以产生房间里抛硬币的观测结果。

在给定的情形下，尝试给从房间里传出的观测结果集建模。从一个简单模型开始，假设只抛一枚硬币。在这种情况下，只需要确定这枚硬币的偏置以及在抛硬币时产生的头/尾观测集即可。这就得到了可直接观测（零阶）马尔可夫模型，它只是一个伯努利（Bernoulli）实验的集合。实际上，这个模型可能过于简单，而不能够可靠地捕获抛多枚硬币的情形。

接下来考虑两个硬币的模型，有两个隐含的状态 s_1 和 s_2，如图 13-1 所示。得到一个概率转换矩阵 A，该矩阵用于控制系统在任意时刻的状态（即抛出了哪枚硬币）。每个硬币有不同的 H/T 偏置 b_1 和 b_2。假设观测到的抛硬币产生的串：O = H, T, T, H, T, H, H, H, T, T, H。这个观测集可以用以下的状态转换路径进行解释（如图 13-1 所示）：$s_2, s_1, s_1, s_2, s_2, s_2, s_1, s_2, s_2, s_1, s_2$。

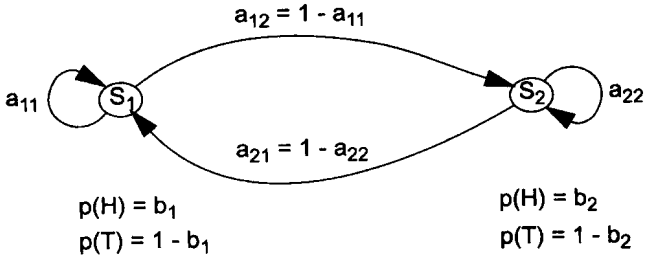


图 13-1　描述抛硬币问题的隐马尔可夫模型的状态转换图

注：a_{ij} 的值由 2×2 转换矩阵 A 的元素确定

第三个模型如图 13-2 所示，使用三个状态（三枚硬币）解释抛硬币的结果。每个硬币的状态还是 s_i，并都有自己的偏置 b_i，系统的状态（抛出了哪枚硬币）是一些诸如掷骰子这样的概率事件和转换矩阵 A 的函数。此三状态机能对状态序列 $s_3, s_1, s_2, s_3, s_3, s_1, s_1, s_2, s_3, s_1, s_3$ 进行解释。

抛硬币问题的难题是选择一个最佳模型。最简单的模型有 1 个参数，是单个硬币的偏置。两个状态的模型有 4 个参数，是状态转换矩阵和每枚硬币的偏置。三个状态的模型有 9 个参数，如图 13-2 所示。自由度越大的大模型越有能力表示一个（可能）更复杂的情形。尽管如此，给大模型确定参数的复杂性问题对应用来说是需要考虑的。例如，在一个简单模型下产生的真实观测值情形下，大模型可能就不正确，在特定情况下，需要更多的数据来建立不同级别可信度。类似的问题，如过拟合，也会给小模型造成困难。最后，每个模型都有一个隐含不变的假设，即系统的转换和偏置概率不随时间的改变而改变。

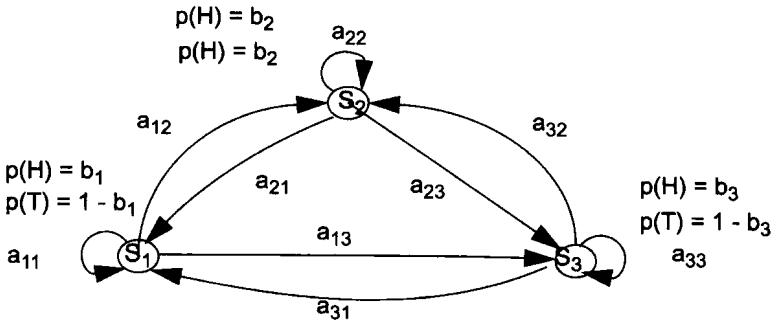


图 13-2　描述抛硬币问题的三状态隐马尔可夫模型的状态转换图。
注：S_i 是每一个硬币/状态，b_i 是相应的偏置

另一个例子，考虑 N 个容器的问题，每一个容器包含 M 个不同颜色的球。获得观测值的物理过程是根据某一随机过程从 N 个容器中挑出一个。一旦选定一个容器，就拿出一个球，并且在输出序列中记录这个球的颜色。然后将这个球放回，根据与当前容器关联的随机过程来选择下一个（也许是同一个）容器继续该过程。这个过程将产生包含球的颜色的一个观测序列。很明显，与该过程对应的一个最简单的 HMM 模型是，模型中的每一个状态对应一个特定的容器，用状态转换矩阵表示对下一个状态的选择。最后，该模型中球颜色的概率被定义为每个状态（容器）的球的颜色和数量。但是，除非模型有先验知识，否则通过确定所有参数来选取最优的 HMM 是一项艰巨任务。

下一节将介绍几个隐马尔可夫模型的变形。

13.1.2　隐马尔可夫模型的重要变形

在上一节，已经说明了隐马尔可夫模型是一个简单且能有力地表达不确定问题的模型。目前所能看到的 HMM 的计算简单性遵循了一阶系统的马尔可夫原则：当前状态只依赖于前一个状态。为了使模型更富有表达能力，在这一节将给出一个改进的隐马尔可夫模型。由于 13.1.1 节给出的常规隐马尔可夫模型在特定情况下会受到一些限制，因此在这一节介绍几种能够解决这些限制的隐马尔可夫模型的变形。

一个被广泛使用的扩展的隐马尔可夫模型称为*自回归隐马尔可夫模型*，在该模型中，之前的观测能够影响当前的观测值，这个扩展模型用于从过去获得更多的信息，并且把这些信息引入对当前状态的解释中。为了对由一系列简单子过程构成的复杂过程进行建模，通常采用因子化的 HMM。另一个隐马尔可夫模型的扩展是*层次化的隐马尔可夫模型*（HHMM），该模型假设一个系统的每一个状态本身是一个隐马尔可夫模型。下面详细介绍隐马尔可夫模型的这些扩展。

自回归隐马尔可夫模型（AR-HMM）

在隐马尔可夫模型一阶特性中，假设当前状态只依赖于前一个状态，并且已观测到的变量是无关联的。这是一个局限，因为当前状态不能充分捕获前面状态所获得的信息。例如在诊断推理中，这会导致比较差的泛化能力，特别是给时间序列数据建模时。实际上，在复杂的环境中，一个时间段的观测值经常与后续观测值相关，而这必须分解成 HMM 的结构。

我们克服这方面受限的一般化的方法是，在考虑一般的隐马尔可夫模型前一个状态影响的情况下，允许以前观测影响对当前观测值的解释。这个隐马尔可夫模型被称为*自回归隐马尔可夫模型*，该辐射模型（emission model）被定义为非零概率模型：

$$p(O_t \mid S_t, O_{t-1})$$

其中 O 是输出状态或者观测值，S 代表隐含层，如图 13-3a 所示。传统的隐马尔可夫模型与 AR HMM 的区别是 O_{t-1} 和 O_t 存在链接（影响）。建立这个影响的动机是一目了然的，因为一个时间段观测是下一个观测结果的预示。换句话说，时间序列分析观测到的输出值能够根据一些基本分布改变。

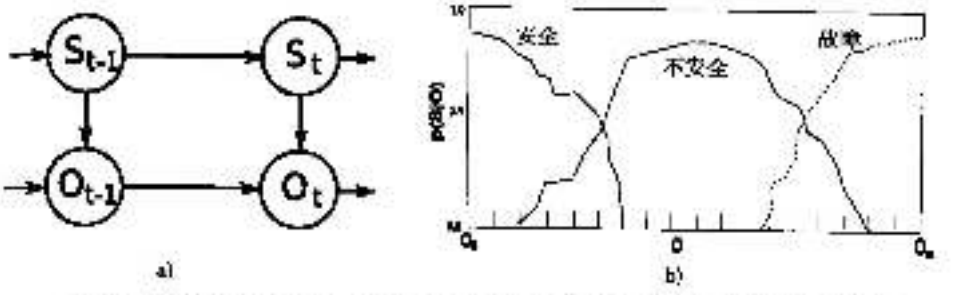


图 13-3 a) $p(Q_t \mid S_t, Q_{t-1})$ 时的 AR-HMM，S 是隐藏状态，O 是可观察状态
b) AR-HMM 的 S_t 中隐藏状态 S_t 的值：安全、不安全和故障

AR-HMM 通常比普通的 HMM 收敛到更高的似然值，特别是给极其复杂的有噪声时间序列建模时。我们将用一个改编自 Chakrabari 等（2007）工作的例子来说明 AR-HMM。这个模型用于监测直升机旋翼系统的运行健康情况。图 13-4 描述了从运行系统中得到的一个数据样本。这些数据是几个传感器的输出，包含来自多个部件的热量和振动度量的信息。图 13-5 显示了图 13-4 的使用快速傅里叶变换（FFT）的数据处理结果，这样能够生成在频域上的等价数据，然后使用这些数据训练一个 AR-HMM。如图 13-3 所示，隐含结点 S_t 的三个状态是安全、不安全和故障。

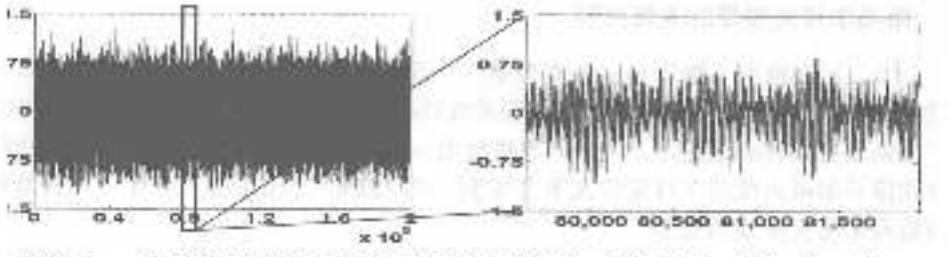


图 13-4 一个用于 AR-HMM 的跨越多个时间周期的实时数据，及一个单时间片扩展

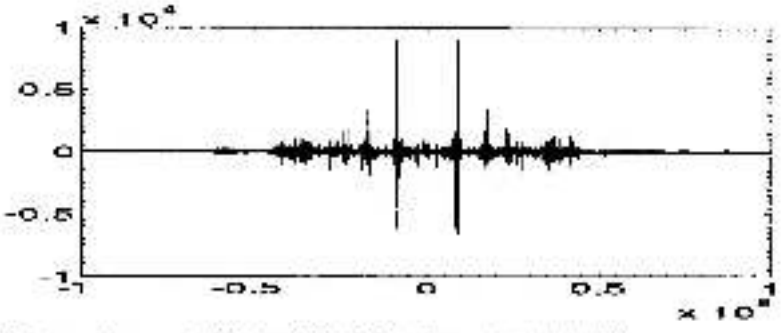


图 13-5 图 13-4 中时间序列数据的快速傅里叶变换后频域信息
是每个时间周期提供给 AR-HMM 的数据

这个项目的目的是在机械系统中研究故障监测和诊断的技术。通过使用来自传感器监测机制过程的一系列时间数据来建立一个整个过程的量化模型，并比对数据集（诸如那张这样的故

障）上训练这个模型，用于以后的实时诊断和未来故障预测。训练完 AR-HMM 后，这个模型被应用到实时中。虽然报告的精确性（Chakrabari et al. 2006）只有 75%，但如何设计这些预测工具也是很重要的。

因子化的 HMM

在之前讨论到的 HMM 和它的变形中，我们描述了状态空间被部分隐藏的系统，我们只能通过观测到的序列得到状态空间的线索，并且尝试使用从观测到的系统产生的数据序列信息来推测基本的隐藏状态。

但是如果潜在的过程很复杂并且不能用简单的状态集来有效地表示，那将会发生什么？如果潜在过程是由几个子过程的结合而成，将会发生什么？常规的 HMM 不能有效地获得这些状态中的过程信息，需要一个更丰富的模型表示更复杂的情形。

因子化 HMM 是一个可能的解决方法。图 13-6 给出了一个自回归因子化 HMM。可以看出，一个基本过程被分成 n 个子过程，这 n 个过程的每一个过程都对当前观测值 O_t 产生影响。此外，每个子过程也遵循一阶马尔可夫特性，即当前的状态只依赖于之前的一个状态。与可观测到随机变量的关系可以用条件概率分布（CPD）定义：

$$P(O_t \mid O_{t-1}, s_t^1, s_t^2, \cdots, S_t^i)$$

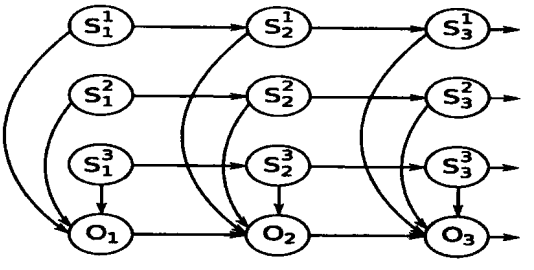


图 13-6　一个自回归的分层 HMM，其中可观察状态 O_t 在时间 t 依赖于复合（S_t）子过程 S_t^i 和 O_{t-1}

层次化的 HMM（HHMM）

层次化的 HMM 用于给极其复杂的系统建模。在该种模型中每个状态都被看作是一个自包含的概率模型。更精确地说，HHMM 的每个状态自身是一个 HHMM。这暗示着 HHMM 的状态产生的观测序列不是简单的观测，而是标准的 HMM 和 HMM 的变形。

当到达 HHMM 中的一个状态时，它能够激活自己的概率模型，即能够激活一个基本的 HHMM 状态，而这个状态又能够激活自己的潜在 HHMM，等等，基本情形是传统的 HMM。这个过程被重复，直到到达一个*生成状态*。从通常 HMM 角度来说，只有产生状态可发射可观测符号。当生成状态发出一个符号时，控制回到激活生成状态的状态。这个不能直接发出观测符号的状态称为*内部状态*。在 HHMM 的一个内部状态下，激活一个状态被称为*垂直转换*。在一个垂直转换后是一个完全发生在同一层次的*水平转换*。当一个水平转换导致一个*终止状态*时，控制回到在 HHMM 中更高层次的状态，这个状态生成最后一个垂直转换。

值得注意的是，在到达生成状态的序列和最终回到最顶层之前，一个垂直转换能够导致更多的垂直转换。所以，到达生成状态产生一个观测符号的序列，该符号由这个状态在最顶层产生。

n-gram HMM

正如先前提到的，在传统的一阶马尔可夫模型中，当前状态只依赖于前一个状态而独立于所有更往前的状态。在这种情况下，虽然使用马尔可夫过程的规则降低了计算复杂性，但仅用前一个状态很可能不能完全获得问题域中能够得到的信息。我们将详细地阐述 n-gram *隐马尔可夫模型*（或 n-gram HMM）。这些模型对于将在 15.4 节中详细介绍的自然语言理解的随机模型是很重要的，特别是在 2-gram 和 3-gram 形式上。

根据 2-gram 马尔可夫模型，当前状态的解释依赖前一个状态的解释，如 $P(O_t \mid O_{t-1})$；等价于传统的一阶马尔可夫链。即使一阶马尔可夫假设很明显，考察 2-gram 的处理手段也是很有意义的。例如，在自然语言理解中，对于单词或音素识别。假设在给定先前单词的序列的情况

下，当只使用前一个单词的解释时，一个特定单词的发生概率能够得到最好的估计。因此，与不采用先前单词信息或者先前单词的更长序列的知识情况相比，2-gram 能够得到更好的结果。例如，给定一个语言数据的集合，相对于 p(lamb | a little)、p(lamb | had a little) 或者 p(lamb | Mary had a little)，p(lamb | little) 对当前单词 lamb 有比较好的预测。

类似地，3-gram 模型是当前观测状态只依赖于前两个状态的解释的马尔可夫模型。3-gram 被表示成 $P(O_t \mid O_{t-1}, O_{t-2})$。继续以自然语言理解为例，相对于 p(lamb | little)，或者包括更多的前面的单词，如 p(lamb | Mary had a little)，3-gram 模型 p(lamb | a little) 是对当前单词 lamb 的比较好的预测。

n-gram 马尔可夫模型扩展成任意长度 n，直到认为能够捕捉到想要建模的数据中存在的不变性。n-gram 马尔可夫模型是一个当前状态依赖于 n-1 个先前状态的马尔可夫链。

正如将在第 15 章进一步看到的，n-gram 模型特别适合在语言理解或其他语言识别任务中，捕获音素、音节和单词的序列信息。n 的值根据想要表示的域的期望复杂度来设定，如果 n 太小，模型的表达能力相对较弱，会失去捕获前面长字符串的预测能力。另一方面，如果 n 的值太大，验证模型的计算开销会很高甚至无法计算。这个问题的主要原因是，语音或者单词结合的先验期望来自收集语言现象的大型数据库（通常是组合数据集），这种大型数据库也被称为语料库。我们没有充分的数据来验证 n 很大的 n-gram。例如，我们可能没有任何 p(lamb | Mary had a little) 的信息，而有很多 p(lamb | little)、p(lamb | a little) 或者 p(lamb | the) 的信息。典型的例子是在语音或者文本识别任务中使用 n 不大于 3 的 n-gram 模型。

还有很多我们没有介绍的 HMM 的变形，包括*混合存储* HMM，该模型使用多个低阶马尔可夫模型来克服问题复杂性。自回归 HMM 的进一步扩展被称为*隐藏*（buried）*马尔可夫模型*，允许可观测结点之间的非线性依赖。*输入 - 输出* HMM 把输入序列映射到输出序列。比子过程简单组合更复杂的是子过程间具有相互作用，对于这种情况，我们通常使用*耦合*（coupled）HMM。*隐半马尔可夫模型*一般化 HMM 来避免其几何衰退效应，它通过使隐藏的状态之间的转换概率依赖于自上一个状态转换的时间长短达到目的，参见 13.4 节。

下一节应用 2-gram HMM 技术解释英语音素组合问题，并且给出采用 Viterbi 算法的动态规划实现 HMM 搜索。

13.1.3　使用 HMM 和 Viterbi 解码音素串

在隐马尔可夫模型的最后一节给出一个 HMM 技术的重要应用：在自然口语中识别模式，其中假设大型语言语料库支持语音组合的先验期望。最后，使用 4.1 节中介绍的动态规划算法实现 HMM 推理。用动态规划算法找到后验字符串的最大概率，这通常被称作 Viterbi *算法*。

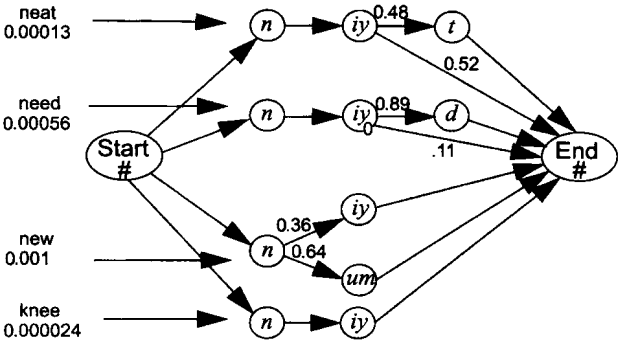


图 13-7　表示相关英语单词的音素集的概率有限状态自动机，反映了出现的下一个单词的概率，引自 Jurafsky 和 Martin（2008）

接下来的人类语音模式的计算分析和使用 Viterbi 算法来解释音素串的例子改编自 Jurafsky 和 Martin（2008）。概率机中的输入是一个语音串，或者基本语音发音。这些输入来自口语中的声学信号的分解。在自动语音理解中，很难获得没有歧义的串的声学信号，因此语音

信号更倾向于概率化地解释为特定的语音。为了简化 HMM 过程的 Viterbi 算法描述，这里假设信号能被无歧义地解释。

图 13-7 描述了单词数据库的片段，并通过相似性把组成声音成分的音素集关联起来。虽然由 neat、new、need 和 knee 组成的单词集是英语词汇中的一小部分，但是可以想象与它们相关的大量聚集可以支持一个语音理解系统。图 13-7 是一个概率有限状态机的例子，这个例子最早在 5.3 节中介绍过。假设语言语料库是一个类似的有限概率状态机的集合，它能够提供不同可能语音组合的先验概率度量，并且最后把这些语音看作单词的一部分。

分析的目的是从单词数据库中确定能够最好地表达输入的声学信号的英语单词。这要求使用 HMM 技术，因为可能单词的表示自身是随机的。在非确定有限状态机中，如图 13-7 所示，语音串给了我们一个解释观测集。假设观测字符串由语音中的#、n、iy、#组成，其中#表示声音间的中断暂停。使用 Viterbi 算法可以看出哪条概率有限状态机路径能最好地获得这些观测。使用前向模式的 Viterbi 重复地找到下一个最佳的状态和值，并且设置一个指针。在后向模式中，回溯这些指针以获得最优的路径。这样，Viterbi 的输出是一个状态的优化路径，在这条路径上含有该路径的概率。

使用 Viterbi 算法搜索的每个状态都关联一个值。在时间 t 状态 s_i 的值为 viterbi[s_i, t]。在状态机中，与下一个状态 s_j 关联的值记为 viterbi[s_j, t+1]。下一状态的值的计算方法是当前状态的值 viterbi[s_i, t] 乘以从当前状态进入到下一个状态的转换概率 path[s_i, s_j]，给定 s_i，观测值 s_j 的概率再乘以 p(s_j | s_i)，其中 p(s_j | s_i) 来自已知的英语语言中发生的语音对中的观测似然值，则转换概率 path[s_i, s_j] 来自非确定有限状态机。

下面我们给出 Viterbi 算法的伪代码。注意它与 4.1 节描述的动态规划算法的相似性。存储的数组和并列的值必须迭代超过 R 行——等于概率有限状态机中（PFSM）语音的个数加 2，以控制每个状态再加上开始和结束状态。迭代必须超过 C 列——等于观测值的个数加 2，以控制空语音#的使用。这些列描述了观察时间序列数并且每个状态必须连接的适当观测语音，如图 13-8 所示。

```
函数 Viterbi(观测长度 T, 概率 FSM)
begin
    number: =FSM 的状态数
    建立一个概率矩阵 viterbi[R=N+2, C=T+2];
    viterbi[0,0]:=1.0;
    for 每一步(观测值)t 从 0 到 T do
      for 每一个状态 si 从 i=0 到 number do
        for 每一个从 si 到 sj 在概率 FSM 转换 do
          begin
            new-count:=viterbi[si, t] × path[si, sj] × p[sj, si];
            if((viterbi[sj, t+1] =0)或者(new-count > viterbi[sj, t+1]))
              then
                begin
                  viterbi[s', t+1]:=new-count
                  添加 back-pointer[sj, t+1]到 back-pointer 列表
                end;
          end;
```

```
    return viterbi[ R, C] ;
    return back-pointer 列表
end
```

Start = 1.0	#	*n*	*iy*	#	end
neat 0.00013 2 paths	1.0	1.0 x0.00013 =0.00013	0.00013 x 1.0 =0.00013	0.00013 x0.52 =0.000067 (2 paths)	
need 0.00056 2 paths	1.0	1.0 x0.00056 =0.00056	0.00056 x 1.0 =0.00056	0.00056 x0.11 =0.000062 (2 paths)	
new 0.001 2 paths	1.0	1.0 x0.001 =0.001	0.001 x0.36 =0.00036 (2 paths)	0.00036 x 1.0 =0.00036	
knee 0.000024 1 path	1.0	1.0 x0.000024 =0.000024	0.000024 x 1.0 =0.000024	0.000024 x 1.0 =0.000024	
Total best					0.00036

图 13-8　图 13-7 中几个路径的 Viterbi 算法的描述，行代表在 Viterbi 中每个输入值对应的单词的最大值，引自 Jurafsky 和 Martin（2008）

图 13-8 改编自 Jurafsky 和 Martin（2008）提出的 Viterbi 算法，该算法处理图 13-7 中的概率有限状态机的轨迹以及语音序列#、n、iy、#（观测值的长度 T = 4）。后向轨迹连接表示了通过概率有限状态机的最优路径，这条路径描述了观测语音串的最近似解释是单词 new。

在下一节我们将描述另一个图模型，一个一般化的马尔可夫模型，也被称为动态贝叶斯网络（或 DBN）。

13.2　动态贝叶斯网络和学习

在建模中，我们常遇到这样的问题，即基本过程最好用一个状态序列描述。从序列的过程中收集的数据，常用数据获得的时间标记或者文本串中的位置索引。这种序列性的依赖是确定的或随机的。动态过程本质上的随机性要求模型有非确定性建模的能力。

虽然能够用有限状态机（FSM，见 3.1 节）很好地描述大多数确定的序列，但是一些重要的序列可以是任意长度的。因此，在任意给定时刻，当各自的状态集中的状态数量和状态转换的数量呈指数倍增长时，给每个实例建模就成问题了。所以，这种序列的概率模型使模型能够易于处理，同时能够捕获序列信息。

在 13.2 节中，我们描述图模型范围下介绍的两种不同的学习：结构学习和参数学习。我们简要介绍结构学习技术，包括马尔可夫链蒙特卡罗（或者 MCMC）采样技术，还将详细说明参数学习中流行的 meta-算法，即期望最大化（或者 EM）。我们将展示称为 Baum-Welch 的一个递归动态规划算法是怎样用于 DBM 和 HMM EM meta-算法的。

在 13.3 节中，我们通过介绍马尔可夫决策过程（或者 MDP）和部分被观测的马尔可夫决策过程（或 POMDP）解释如何把马尔可夫模型用于决策支持。然后我们介绍强化学习（在 10.7 节中介绍过），并在其中补充概率状态和反馈模型。最后，我们用一个使用了 MDP 的强化学习方法

设计的机器人导航系统结束本章。

13.2.1　动态贝叶斯网络

在 9.3 节中，我们介绍了贝叶斯信念网络（BBN），这是一个流行的被成功地用于概率建模的形式方法。但是，对于动态或序列过程建模，BBN 很不灵活。在自然语言理解和实时诊断中，BBN 无法用于对时间序列进行建模，其中包括因果过程以及经常出现在复杂任务中的关联序列数据。这个问题是 BBN 的依赖条件，总是假定其在某一时刻处于活动状态。

虽然我们把细节的讨论拖到这一章，但在 9.3 节中，我们介绍了动态贝叶斯网络（或 DBN）。简单地说，一个动态的有时也被称为时序的贝叶斯网络是同一类贝叶斯网络的序列，这个贝叶斯网络的结点沿（有向）时间行进方向连接。动态贝叶斯网络是随机过程的有向图模型序列。它们是流行随机工具，如隐马尔可夫模型和线性动态系统（LDS）的一般化扩展，动态贝叶斯网络把隐（和被观测）状态表示为状态变量，在时间区间上有复杂的相互依赖关系。这个图结构提供了描述条件依赖的方法，因此提供了紧密的参数化模型。

在不随时间变化的静态假设意义上，组成 DBN 的各个 BN 不是动态的，但 DBN 支持动态模型的建模。所以 DBN 可以在定义的参数时不变的假设下工作。但是，正如我们看到的，隐藏的结点能够被加入并表示在当前操作的条件下，建立混合模型，捕获时间维上的周期变化。

在每个时间段（片），一个 DBN 类似于含有 N_h 个表示隐状态的随机变量集合 Q_t 和 N_o 个表示可观测状态的随机变量集合 Q_t 的静态 BBN。每个随机变量可以是离散的也可以是连续的。我们需要定义可以表示一个时间段内的状态间条件分布的模型以及一个转换模型（表示连续时间段间的 BBN 的关系）。所以，如果 $Z_t = \{Q_t \cup O_t\}$ 是两个变量集的并集，这个转换和观测模型可以被定义为没有循环的两个时间段的 BBN 的条件概率分布的乘积，称为 B_t。符号(1:N)意思是从 1 到 N 的所有变量：

$$p(Z_t(1{:}N) \mid Z_{t-1}(1{:}N)) = \prod_{i=1}^{N} p(Z_t(i) \mid Pa(Z_t(i)))$$

其中 $Z_t(i)$ 是时间段 t 的第 i 个结点，$Pa(Z_t(i))$ 是 $Z_t(i)$ 的父结点，并且 $N = N_h + N_o$。简单起见，我们限制这里的结点是一阶马尔可夫过程的定义，定义中在时刻 t 的变量只受在时刻 $t-1$ 的前驱变量影响。我们表示无条件的初始状态分布 $p(Z_1(1{:}N))$ 为一个无条件贝叶斯网 B_1。B_1 和 B_t，对于所有的 t，定义 DBN。在图 13-9 中是一个有两个时间段的简单 DBN。

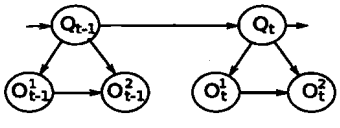


图 13-9　一个两个时间片的 DBN 例子。随机变量的集合 Q 是隐藏的，集合 O 是观测集，t 表示时间

很明显，动态贝叶斯网络是 13.1 节隐马尔可夫模型的一般化。在图 13-9 中贝叶斯网络在时间 t 和 $t+1$ 相连。两个网络在时间上至少要有一个结点相连。

13.2.2　学习贝叶斯网络

学习是智能行为的核心。智能体通常记录过去观测到的事件，期望这些信息可能在将来具有作用。一个智能体在它的生命周期中会遇到很多事件，这些事件可能比它们具有的存储资源

还要多。所以，我们把每一个事件编码成一个抽象摘要而不是每一个观测事件的完整记录。而且，压缩的信息有助于快速检索。所以，学习可以被看作数据压缩、索引和存储的组合。

一个智能体的任何新的观测可能稍不同于先前的观测，但还是属于先前事件的一类。在这种情况下，忽略特定事件下不重要的细节，把这个事件作为先前类的实例来学习是必要的。一个新的事件可能非常不同于之前所有的事件，因此智能体需要学习一个新的类或是特定的不规则事件。一个智能体需要建立足够的类来完全刻画参数空间，同时能够成功处理新的不规则事件。所以，好的学习是通用性和具体性的平衡。

在概率图模型的范畴中，我们遇到了两个普通的学习问题适应我们先前的讨论，包括学习整个新关系（图模型）的*结构学习*和重新学习现有模型成分的*参数学习*。

结构学习和搜索

结构学习可以解决确定变量间是否存在统计依赖的一般问题。例如，以下问题的答案有利于确定天气概率图模型的结构：乌云是否与下雨有关？土星的位置是否与在某一天飓风的次数相关？台风的存在与全球变暖是否有关？北极的天气是否与撒哈拉沙漠的边界变化相关？可以明显看出，在概率模型中，对于这些问题，为真的答案能够保证问题中变量之间的条件依赖，为假的答案允许我们把这些变量看成独立的。

在概率图模型设计中，同时限制保持网络是一个有向无环图（DAG）时，每个变量可能与其他所有变量相依赖。在最坏的情况下，我们会得到一个完全连接的 DAG 作为模型。但是，在任意规模的实际应用中这并不常见，因为条件概率表的创建是很困难的。一个重要的图模型设计是考察被建模在环境中的几个变量是否存在独立性。因为在贝叶斯信念网络中推理的复杂性依赖于模型中变量间的依赖关系个数，我们希望在保持模型有效性的情况下依赖尽量少。换句话说，我们想要从完全连通图中剪掉尽可能多的（不必要的）依赖。

所以结构学习被认为是对给定变量，即表示的应用领域，在可能结构空间中搜索最优结构的问题。正如 Chickering 等（1994）指出的那样，在已有的数据集搜索最优解是 NP 难的。这个最优结构很好地描述了数据并尽可能简单。

但是，学习到的结构要能够描述先前观测的数据并与新的可能相关数据吻合。所以，当我们建立（学习）一个复杂结构，精确刻画积累的数据时，我们可能得到一个很特别的结构而无法适应新数据。在优化文献中，这种情况有时被称为*过拟合*。

虽然用蛮力法在指数级无向图空间中搜索对于实际应用来说不现实，但存在一些方法帮助我们解决这一问题。第一个原则是奥卡姆剃刀，也称*简约原则*。在两个有竞争的结构中，简单的结构优于复杂的结构。在某些情况下，奥卡姆剃刀规定，从简单到复杂地搜索可能的结构，当找到一个能成功地适用于当前的任务并且足够简单的结构时中止，这样一个简单启发式方法实际上把贝叶斯先验知识应用于结构搜索。

但是，通过先验地定义结构的复杂性度量是困难的。建立一个合理的复杂性度量把所有可能的结构按该度量排序并不是显而易见的，因为可能是同等复杂结构的类，在这种情况下我们可能把偏序加到结构空间。即使在同一类的结构中，在所有可能结构中进行穷举搜索也常常是不切实际的。

正如 AI 中的很多难题，通常情况下的结构归纳，不可能用于相当复杂的情况。结果，我们使用启发式方法让结构搜索更容易处理。虽然不令人满意，但对给定工作状态模型，启发式方法在模型的失败结果附近执行了贪心局部搜索。假定目标仅是在图结构中找到局部最优解，那么可能给建模问题提供一个足够好的解决方案。另一个启发方法可能依赖于人类专家在这一问题上的建议来重新考虑模型的结构，同时人类专家可能再次关注当前模型的弱点。模型归纳问题

的一个更一般的方法是在可能模型空间中抽样，我们将介绍该类型抽样的一个通用技术，马尔可夫链蒙特卡罗（Markov chain Monte Carlo）。

马尔可夫链蒙特卡罗

马尔可夫链蒙特卡罗（或 MCMC）算法是从概率分布中抽样的一类算法。MCMC 是基于建立一个将平衡分布作为期望分布的马尔可夫链的方法，在很多步后链上会出现被称为在时间中消失（burn in time）的状态，然后该状态被用作该应用领域的一个可能 DAG 的期望分布的样本。当然，马尔可夫链样本质量的提高是作为一个样本数量的函数存在的。

在结构搜索的问题中，我们假设给定变量集合的可能结构构成了一个马尔可夫链的状态空间。这些状态的转换矩阵通过基于权值的技术更新，开始权值可能一致，在这些结构中一个蒙特卡罗随机行走能够相等地修正所有结构的权值。

当然，在时间上，大的权值被用于高度相关结构的转换。如果结构 A 只在一个链接上不同于结构 B，那么该空间中的一个蒙特卡罗行走是这些结构的启发式组织的邻域中的一个搜索。当被抽样时，每一个结构与一个得分相联系，典型的就是基于得分的最大化后验概率（或 MAP）p(structure | data)。蒙特卡罗抽样技术用于遍历结构空间，通常以大概率选择转换以提高数据的似然值，而不是选择得分差的转换。在多步迭代后，该 MCMC 收敛于较好的结构，与起始点比较，它提高后验概率 p(structure | data)。但是它还是存在在样本空间中陷入局部最优的问题；这个问题能够用诸如随机重新开始或模拟退火（simulated annealing）技术来解决。

参数学习和期望最大化

参数学习可以和遍历了图模型空间的结构学习相比。给定一个问题域的现有模型，并给出观测集，参数学习试图去推理一组变量的最后的联合分布。参数学习通常用于一般结构搜索的子程序。当一个完整的数据集出现时，用于分布的参数学习可以简化为计算。

在图模型的很多有趣的应用中，对某个变量来说，可能会有数据集不完整的问题，一个更深入、更相关的问题是模型中可能会出现隐藏的或者不被察觉的变量。在这种情况下，期望最大化（EM）算法为推断联合分布的估计提供了有力工具。在 A. Dempster、N. Laird 和 D. Rubin（1977）的一篇经典论文中，首次提出和解释了 EM 算法。这篇论文扩展了这种方法并且发展了其后续理论。

EM 已经被证明是统计和概率建模中最为流行的学习方法。在统计学中使用 EM 算法来寻找概率模型中参数的最大似然估计值，而该模型是依赖于可能的不可观测的隐藏变量的。EM 是一个迭代算法，在以下两个步骤中交替进行，首先是 E 步，计算包含隐藏变量的变量的似然期望，仿佛隐藏变量是可观测的一样；然后是 M 步，计算 E 步的似然期望中参数的最大似然估计值。M 步中的参数再次被用于下一轮的 E 步，过程一直重复至收敛为止，也就是说，直到 E 步和 M 步之间只有非常小的变化为止。我们在 13.2.3 节中给出了 EM 算法的一个例子。

HMM 的期望最大化（Baum-Welch）

参数学习的一个重要工具就是 EM 算法的一个变形，称为 Baum-Welch，当只给定发出的（可观测的）训练数据时，用 Baum-Welch 最大似然和后验式来估计 HMM 或其他时序图模型的参数。Baum-Welch 算法含有两步：

1）计算每个时间状态前向和后向概率。

2）在第一步的基础上，确定转换 - 发出对值的频率并把这个值按整个序列的概率分开。

Baum-Welch 的第一步使用前向 - 后向算法。正如最初在 4.1 节中见到的，该算法支持动态

规划，到13.1.3节中被用作语音的解释字符串为止，曾出现过几次。

在给定模型的参数情况下，Baum-Welch算法的第二步用于计算隐马尔可夫模型或其他时序模型中特定输出序列的概率。第二步实际上是计算特定转换－发出对的期望次数。每次发现一个转换，同时这个序列概率上的转换比率值增加，并且这个值能够用来计算新的转换值。

解决HMM参数学习问题的一个蛮力方法是产生所有可能的观测序列，及以两个转换矩阵得到的概率产生隐状态。给定模型时，两个序列的联合概率通过与矩阵中的相应概率相乘计算得到。该过程的时间复杂性为$O(2TN^T)$，其中，T为观测到事件的序列长度，N是状态字母表中符号的个数。这个时间复杂性开销在现实问题中是难以处理的，实现Baum-Welch的一种选择是利用动态规划算法。

接下来我们使用结合了信念循环传播（Pearl 1988）的EM算法来阐述参数学习的例子。

13.2.3　期望最大化：一个例子

我们用一个例子描述期望最大化，并且使用Pearl（1988）的循环信念传播算法展示相关的步骤。这个例子在9.3节中简要讨论一阶随机建模语言时就已经实现了，例子中对于马尔可夫随机场的表示来自于循环逻辑（Pless et al. 2006）一阶随机建模语言。即使简单的方法能够用于解决我们提出的防盗自动报警例子的限制，但我们的目标是在可理解的情况下，说明BBN、循环信念传播和EM算法的一个重要推理方案。

考虑图13-10中的贝叶斯信念网络，它有3个结点：警铃A、行窃B和地震Q。这个模型描述了一个防盗自动报警系统A_t，在时刻t活动，它依赖于一个可能的行窃B_t，或者一个地震Q_t。虽然有一些假设概率度量的模型能够用程序描述成循环逻辑：

$A_t \mid B_t, Q_t$ = [[[0.999,0.001],[0.7,0.3]],[[0.8,0.2],[0.1,0.9]]]

但我们用简化了的实际循环逻辑语法使这个表述变得更清晰。

在这个程序中我们假设所有的变量都是布尔型的，也假设图13-10中的BBN包含上面特定的条件概率分布（CPD）。例如，当给定一个行窃或地震时，表中具体指定警铃不响的概率是0.001；当给定一个行窃或地震时，警铃响起的概率是0.1。

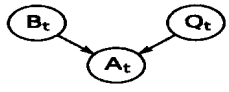


图13-10　警铃、地震、行窃例子的一个贝叶斯信念网

我们也注意到这个程序详细说明了一般化的BBN，更确切地说是一个每个时刻t的BBN。这有些类似于Kersting和DeRaedt（2000）的表述方法。例如，如上面一个例子，循环逻辑把概率逻辑语句转换成一个马尔可夫随机场以应用推理算法——循环信念传播（Pearl 1988）。

一个马尔可夫随机场或马尔可夫网络，是一个概率图模型（见9.3节），该模型的结构是一个无向图（而贝叶斯信念网络是有向图），这个马尔可夫随机场结构图中结点对应随机变量，图中结点之间的连接对应于随机变量之间的依赖。在循环逻辑中，马尔可夫随机场被用作原始图模型（9.3节中使用的团树）的一个中间或等价表述。这个马尔可夫随机场支持概率推理，在我们的例子中就是循环信念传播（Pearl 1988）。

这个马尔可夫随机场量化成分由一个势函数集指定，这个函数集的域是这个马尔可夫随机场图结构的一个团，并且定义了一个所有可能指定连接间的对应，该对应是从团中变量所有可能赋值集合到非负实数集。

在循环逻辑系统中，使用马尔可夫随机场的一个特殊版本就是给随机变量之间的每个依赖指定一个势函数。结果，马尔可夫随机场能用有两类结点的二部图表示，这两类结点是*变量结点*和*簇结点*。变量结点对应随机变量，簇结点对应簇结点邻居指定的随机变量的势函数。图13-11

表述了二部图马尔可夫随机场，这个场具有图 13-10 的 BBN 的势函数。

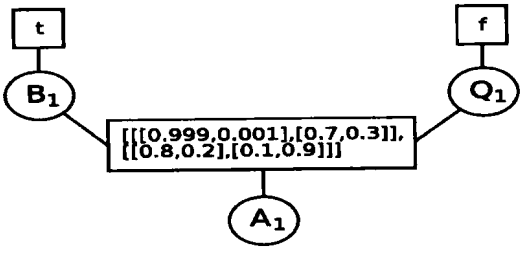


图 13-11　一个反映图 13-10 的 BBN 中随机变量势函数的马尔可夫随机场，连同系统的两个观测

在循环逻辑中，我们能够指定关于模型和提出问题的事实；例如，能够设置三个状态：

B_1 = t.
Q_1 = f.
A_1 = ?

这三个陈述的一个解释是我们已知在时刻 1 有一个行窃，没有地震，并且我们想知道警铃响起的概率。这三个状态和 $A_t \mid B_t$ 一起使用，Q_t = [[[0.999, 0.001,], [0.7, 0.3]], [[0.8, 0.2], [0.1, 0.9]]]，这个循环逻辑系统用事实和图 13-11 包含的势函数来建立马尔可夫随机场。

从图 13-11 中注意到程序中的事实以叶子簇结点加入到马尔可夫随机场中，并且条件概率分布（CPD）编码在连接有 3 个变量结点的簇结点中。为了推测这个查询的答案，该系统把循环信念传播算法应用到马尔可夫随机场。因为阐述了简单的确定事实，并且原始的图模型没有环路，该推理是精确的，并且在一次迭代就可以收敛并返回警铃的分布 [0.8，0.2]。

另外，通过允许它的用户指定可学习的分布，循环逻辑支持选择循环逻辑程序设计规则的学习。通过一个来自期望最大化（Dempster et al. 1977）的信息传递算法来估计参数。为了说明这一点，我们接下来假设不知道图 13-10 中 BBN 的条件概率分布，在循环逻辑中，为了得到这个分布的值，用户把条件概率分布设置成在任意时刻 t 可学习的分布。我们在程序中使用变量 L 定义：

$A_t \mid B_t, Q_t$ = L

现在使用以下的事实表述，给出一个循环逻辑解释在一时间周期上的观测值的集合。为简单起见，只收集三个时间间隔，每个三个数据点：

Q_1 = f
B_1 = t
A_1 = t
Q_2 = t
B_2 = f
A_2 = t
Q_3 = f
B_3 = f
A_3 = f

循环逻辑将一个循环信念传播推理框架与期望最大化方法结合来估计一个模型的可学习参数。为了完成这个，与以前一样解释给模型建立马尔可夫随机场。对于图 13-10 中的警铃 BBN 以及对应的马尔可夫随机场，在图 13-12 中给出，其中可学习参数 L 连接到具有把可学习规则统一到前面介绍的($A_t \mid B_t, Q_t$ = L) 的簇结点上。

下面计算 BBN 已知的分布，并且随机初始化簇结点 R1、R2、R3 和可学习结点 L。最后，三个时间段数据输入到这个系统，其中，已知的事实和以前一样被包含在叶簇结点中。图 13-12 显示了结果。

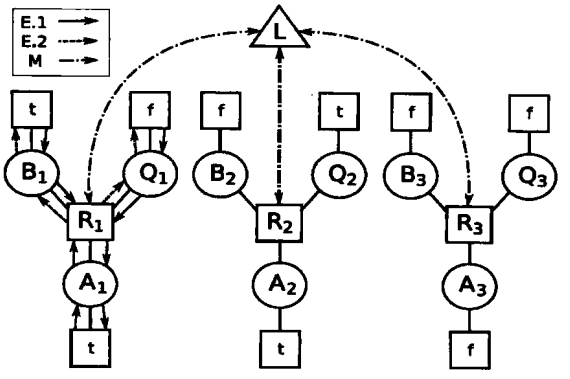


图 13-12　一个可学习的结点 L 被添加到图 13-11 的马尔可夫随机场中。马尔可夫随机场在 3 个时间域进行迭代。为了简单起见，只是证明了时间 1 中的 EM 迭代过程

期望最大化算法是通过消息传递实现迭代过程。每次迭代之后，条件概率的当前近似值，是一个可学习结点传递到每个结点连接的簇结点的消息。当可学习结点更新时，每个相邻的簇结点发送一条消息给 L（图 13-12 中标记 M 的虚箭头）。这条消息是自邻接变量结点进入到簇结点所有消息的产生结果。这些（未归一）表是每个簇结点联合概率的一个估计。这是条件变量和被条件变量的所有状态的一个分布。这个可学习结点计算所有簇消息的总和并且把它们转换成一个归一化的条件概率表。

通过簇结点和变量结点的推理（循环信念传播），我们计算可学习结点消息。为反映基本马尔可夫随机场的两种结构，循环信念传播有两步消息传递：从所有簇结点给它们的相邻变量结点发送消息，即图 13-12 中标记 E. 1 的实箭头，然后再返回，即图 13-12 中标记 E. 2 的虚箭头。

使用这个信念传播算法可以直到收敛产生一个期望值的近似值，这等价于 EM 算法中的 E 步。发生在所有可学习簇结点的平均值返回一个可学习结点中参数的最大似然估计，这对应 M 步。迭代等价于整个 EM 算法，它对跟随可学习结点更新的变量结点和簇结点收敛，支持变量及可学习结点跟随簇结点的更新。

总结这个算法，并且显示在图 13-12 中：

在期望或 E 步：

1）所有簇结点发送消息给变量结点。

2）所有变量结点返回给簇结点以更新簇结点信息。

在最大化或 M 步中：

在 E 步更新的簇结点发消息给可学习结点，并在这些结点中平均。

在刚才所描述的算法中，结点能够以任意顺序改变，并且簇结点和变量结点的更新可能与学习结点的更新重叠。这个迭代更新过程代表了一类 EM 类型算法，其中的一些算法在某些领域比标准的 EM 算法更有效（Pless et al. 2006，Dempster et al. 1977）。这个框架还支持的一个扩展算法是 Yedidia 等（2000）提出的一般化的信念传播算法。

13.3　强化学习的随机扩展

在 10.7 节中首先介绍了具有回报的强化学习，也就是一个智能体学习在外部环境采取不同

动作的集合。智能体的目标是使长期的回报最大化。一般来说，这个智能体想要学习一个策略，或者是世界中状态到动作之间的一个映射。

为了描述强化学习的概念，考虑一个回收机器人的例子（改编自 Sutton and Barto 1998），也就是该移动机器人的工作是从商务办公室中收集空的易拉罐。这个机器人有一个充电电池，还装备有解释传感器和移动手臂。我们假定它还有对感知信息解释系统导航并移动手臂的控制系统。它根据电池的当前电量使用强化学习搜索易拉罐。这个智能体必须能够做出三个决定之一：

1）这个机器人在某段时间内能够主动搜索汽水易拉罐。

2）这个机器人能暂停并且等待有人拿来易拉罐。

3）这个机器人能回到充电基座充电。

机器人有时在固定的时间段决策，有时在找到空易拉罐时作决策，或在某个其他事情发生时决策。结果，机器人有三个动作，且它的状态由电池电量决定。大多数时候这个回报被置零，当收集到一个易拉罐时，其变为正；如果电池没有电，回报为负。注意，在这个例子中，基于强化的学习的智能体是机器人的一部分，它能够同时监视机器人的物理状态和外界环境的情形（状态）：机器人同时监视机器人状态和外界环境。

传统强化学习框架有一个主要的限制，就像我们刚刚描述过的：它实质上是确定性的。为了克服局限性和在一个大范围的复杂活动中使用强化学习，我们通过加入随机成分来扩展确定性框架。在接下来的几节中，我们考虑随机强化学习的两个例子：马尔可夫决策过程和部分可观测的马尔可夫决策过程。

13.3.1　马尔可夫决策过程

到此为止，前面介绍过的强化学习的例子（包括 10.7 节和 13.3 节中介绍的）实质上都是确定性的。实质上，当智能体从某一个时刻 t 的状态中执行一个动作时，其总会确定地指定时刻 t+1的新状态：这个系统是完全确定的。

尽管如此，很多实际的强化学习应用可能不是这样：智能体没有完备的知识或者不能控制世界的下一个状态。更确切地说，从状态 S_t 中选一个动作可能导致时刻 t+1 产生多于一个状态的可能结果。例如，在国际象棋游戏中，当我们做了一个确定的移动时，我们经常不知道对手将要怎么走。我们无法完全知道走了这一步将会是什么状态，实际上，在很多多人游戏中这种不确定性都是存在的。第二个例子是玩彩票或其他机会性游戏，我们选择了可能回报不确定的动作。第三个例子是环境过于复杂，以至于一个状态选择的确定性响应是不可计算的情况，即在一个智能体（而不是一个以上的智能体）中做决定。另外，在这种情形下给出一个基于概率的响应可能是我们认为的最好的方法。在所有的这些情形中，我们需要一个解决强化学习问题的更有力的框架。马尔可夫决策过程（MDP）就是这样一个框架。

MDP 基于一阶马尔可夫特性，这个过程中下一个状态的概率分布只依赖于现在状态和可能动作。MDP 独立于所有现在状态的先行状态。MDP 是一个离散时间随机过程，它包括一状态集、在每一个状态执行的动作集，以及和每个状态相关联的回报函数。结果，MDP 提供了一个能够为一大类强化学习问题建模的有力框架。下面是 MDP 的定义：

定义（马尔可夫决策过程或 MDP）　一个马尔可夫决策过程是一个四元组 $<S, A, P, R>$，其中：

S 是一个状态集，并且

A 是一个动作集。

$P_a(s_t, s_{t+1}) = p(s_{t+1} \mid s_t, a_t = a)$ 是智能体执行动作 $a \in A$，从时刻 t 的状态 s_t 转到时刻 t+1

的状态 s_{t+1} 的概率。由于这个概率 $p_a \in P$ 是定义在整个动作状态空间上，通常用一个转换矩阵表示这个概率。

R(s) 是在状态 s 智能体得到的回报。

我们应该看到强化学习的 MDP 框架和 10.7 节中的强化学习表示之间的惟一区别在于：强化学习表示中的转换函数现在被概率分布函数代替了。在我们的理论中这是一个重要修改，使我们能够在不可计算的复杂世界或实际的随机世界中捕获世界中的不确定性。在这种情况下，一个智能体的动作回报响应最好用概率分布表示。

13.3.2　部分可观测的马尔可夫决策过程

我们看到 MDP 能够对棋类这样的游戏建模，在这个模型中，世界的下一个状态在一个智能体的确定控制之外。我们下国际象棋时，游戏的结果状态依赖于对手的走法。所以，我们要意识到我们所处的状态，也要意识到我们所做的动作（走子），但对于我们来说，结果状态不是立即能够可知的。我们仅有一关于将进入状态的概率分布，而且它依赖于对手的走法。

现在考虑扑克游戏。这里，我们还不确定我们现在的状态！我们知道我们拿到的牌，但是不能很好地知道对手拿的牌。我们也许只能用叫牌的知识来猜测对手的牌。这个环境比 MDP 的环境更具有不确定性。我们不仅不知道我们所处的状态，还必须根据我们对实际环境状态的猜测采取一个动作。结果，对这个动作产生的新环境状态的猜测就更不确定了。所以我们需要一个比 MDP 还丰富的框架给这个双重不确定性情形建模。部分可观测的马尔可夫决策过程（POMDP）是一个能完成这项任务的框架，之所以被称为 POMDP 是因为我们只能观测到部分我们所处的状态以及部分对手将要做出的反应。我们只有可能所处的状态集的概率分布，而不是现在状态的全部信息。下面定义 PODMP：

定义（部分可观测马尔可夫决策过程或 POMDP）　**一个部分可观测马尔可夫决策过程**是一个五元组 $<S, A, O, P, R>$，其中：

S 是一个状态集，并且

A 是一个动作集。

O 是表示智能体能够看到的环境的观测集合。因为智能体不能直接观测到所处的状态，这个观测与环境中根本的真实状态概率相关。

$p_a(s_t, o, s_{t+1}) = p(s_{t+1}, o_t = o \mid s_t, a_{t=a})$ 是当智能体在时刻 t 状态为 s_t 时，执行动作 a，得到观测 o 并导致 t+1 时进入状态 s_{t+1} 的概率。

$R(s_t, a, s_{t+1})$ 是智能体在状态 s_t 执行动作 a 以及转换到状态 s_{t+1} 所得到的回报。

总之，如果观测 o 给我们一个当前状态 s 的精确的描述，则 MDP 可以看成是 POMDP 的一个特殊情况。一个类比有助于理解这个不同：MDP 相对于马尔可夫链就像 POMDP 相对于隐马尔可夫模型。

以下有几个建立 MDP 和 POMDP 的算法。这些算法的复杂性很明显大于之前所说纯确定的情况。实际上 POMDP 算法的求解在计算上是很难解决的。也就是说，使用 POMDP 在多项式时间内为这个问题找到一个最优解是不可能的，而且这个解很有可能是不可计算的。因此，我们用求近似优化解的算法解决这个问题。关于 MDP 和 POMDP 解算法的参考文献包括 Sutton 和 Barto（1998）和 Puterman（1998）。接下来介绍一个 MDP 实现的例子。

13.3.3　马尔可夫决策过程实现的例子

为了描述一个简单的 MDP，我们重新提到 13.3 节开始介绍的回收机器人（改编自 Sutton 和

Barto 1998）的例子。回顾一下，每次这个强化学习智能体遇到一个外部事件时，都有三种决策可供选择：主动搜索回收一个易拉罐，称之为搜索（search）；等待有人拿来一个易拉罐，称之为等待（wait）；基座充电，称之为充电（recharge）。这些决策是机器人只基于电池的能量状态作出的。为简单起见，用两个状态来区分电池的电量：低（low）和高（high），状态空间 S = {low, high}。如果电池现在有电，则它的状态为高，否则，它的状态为低。智能体的动作集合是 A(low) = {search, wait, recharge} 以及 A(high) = {search, wait}。如果 A(high) 包含充电，我们期望智能体学习一个能判断该动作为次优的策略！

电池状态的确定独立于执行的动作。如果电池的状态为高，那么我们期望机器人能够在电量充足的情况下完成一段时间的主动搜索而不会出现没电的这种风险。如果状态为高，那么代理停留在这个状态的概率为 a，变到状态低的概率为 1 - a。当状态为低时，如果代理执行一个主动搜索，那么这个电池的能量水平还保持在相同的状态下的概率为 b，能量用尽的概率为1 - b。

接下来设计一个回报系统。一旦电池能量用尽，S = low。这个机器人被救后，它的电池被充电状态又变为 high，并且得到一个 -3 的回报。每次机器人找到一个易拉罐，得到一个 1 的回报。我们分别用机器人在主动搜索和等待两种状态下收集到的易拉罐的期望数量（智能体收到的期望回报）分别用 R_search 和 R_wait 来表示，并假设 R_search > R_wait。我们还假设当返回充电时，机器人不能够收集任何易拉罐，当电池电量低时机器人不能在一步中收集任何易拉罐。刚刚描述的这个系统能用一个有限的 MDP 表示，其中 MDP 的转换概率($p_a(s_t, s_{t+1})$)以及期望回报见表 13-1。

表 13-1　回收机器人例子的有限 MDP 的转换概率和期望回报

注：表中包含了当前状态 s_t、下一个状态 s_{t+1}、动作和当前状态 a_t 的可能的回报的所有组合

s_t	s_{t+1}	a_t	$p_a(s_t, s_{t+1})$	$R_a(s_t, s_{t+1})$
high	high	search	a	R_search
high	low	search	1 - a	R_search
low	high	search	1 - b	-3
low	low	search	b	R_search
high	high	wait	1	R_wait
high	low	wait	0	R_wait
low	high	wait	0	R_wait
low	low	wait	1	R_wait
low	high	recharge	1	0
low	low	recharge	0	0

为了表示有限 MDP 的动态情况我们可以画一个图，例如图 13-13，这是回收机器人的 MDP 转换图。一个转换图有两个类型的结点：状态结点和动作结点。智能体的每个状态结点由里面含有这个状态名字(s)的大圈来描述。每个状态 - 动作对由一个被表示成一个小黑圈的动作结点连接一个状态结点表示。如果智能体从状态 s 开始，并且执行动作 a，它沿着从状态结点 s 到动作结点(s_t, a)的这条线移动。结果，环境通过离开动作结点(s_t, a)的一个箭头生成一个到下一个状态结点的响应，其中每个箭头对应一个三元组(s_t, a, s_{t+1})，其中，s_{t+1}是下一个状态。这个箭头由转换概率$p_a(s_t, s_{t+1})$以及转换期望回报$R(s_t, s_{t+1})$标记。注意与离开一个动作结点的箭头相关联的转换概率的和总是 1。

我们看到的几个在这章描述的随机学习技术，在第 15 章还要描述，用于对自然语言的理解。

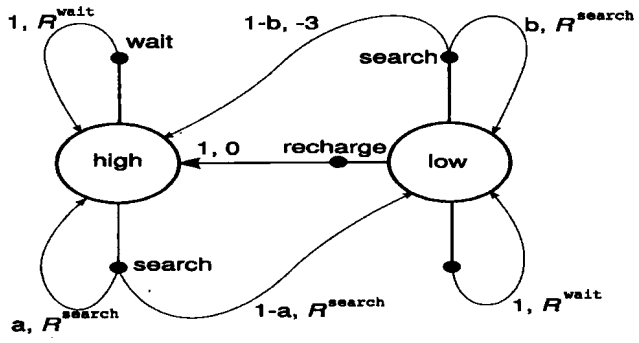


图 13-13　回收机器人的转换图。大环表示状态结点，小黑点表示动作结点

13.4　结语和参考文献

本章从动态的及随机的观点描述了机器学习算法。概率机器学习的核心是贝叶斯算法和马尔可夫算法。这两个工具已经被简化（贝叶斯信念网络和马尔可夫链）来使机器学习的随机方法更加有力和容易处理。贝叶斯定理的首次描述在5.3节中，贝叶斯信念网络的介绍在9.3节中。马尔可夫链以及一阶马尔可夫假设在9.3节中有描述。它们组成了第13章的素材。

在随机方法建模和学习方面有很多入门书和指导。J. Pearl（1988）所著的《Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference》是一个很集中于图模型的很有影响的描述。我们也推荐 F. V. Jensen（1996）的《Bayesian Networks and Decision Graphs》、R. G. Cowell 等（1999）的《Probabilistic Networks and Expert Systems》、S. Lauritzen（1996）的《Graphical Models》、F. Jensen（2001）的《An Introduction to Bayesian Networks》以及 M. I. Jordan（1998）的《Learning in Graphical Models》。

还有几个作者从统计学的角度描述了概率学习，包括 J. Whittaker（1990）的《Graphical Models in Applied Multivariate Statisties》以及 D. Edwards（2000）的《Introduction to Graphical Modeling》。B. Frey（1998）的《Graphical Models for Machine Learning and Digital Communication》从一个工程/商业的角度进行了描述。

J. Pearl（2000）的《Causality》是从哲学角度对图模型支持的一个重要贡献。伴随着尝试澄清偶然性的定义，他描述了偶然性的重要性以及其中概率模型设计的实用性。

图模型的指导介绍包含 E. Charniak（1991）的《Bayesian Networks without Tears》和 P. Smyth（1997）的《Belief Networks, Hidden Markov Models, and Markov Random Fields: A Unifying View》。www. cs. ubc. ca/ ~ murphyk/Bayes/bnintro. html#appl 包含 K. Murphy 的《A Brief Introduction to Graphical Models and Bayesian Networks》，是这方面很不错的在线指导，还有在 www. cs. ubc. ca/ ~ murphyk/Bayes/bayesrule. html 给出了 Murphy 的《A Brief Introduction to Bayes' Rule》。在 www. cs. engr. uky. edu/ ~ dekhtyar/dblab/resourse. html#bnets 上可以找到一个更完整的指导清单。

在一个经典的1977年的论文中，由 A. Dempster、N. Laird 和 D. Rubin 解释了 EM 算法并给出了这个名字。他们指出“这种方法已经在特殊环境中被其他作者提出了很多次”，但他们在1977年把这种方法一般化并且发展了它的理论。

马尔可夫网络和动态贝叶斯网络的推理有两种形式：一种是准确的，其中结果计算开销大但证明正确；另一种是近似的，开销小但结果是近似的，而且可能会使局部最大。对于准确的推

理参见 S. M Aji 和 R. McEliece（2000）的《The Generalized Distributive Law》以及 F. Kschischang 等（2000）的《Factor Graphs and the Sum Product Algorithm》。关于近似算法见 M. I. Jordan 等（1999）的《An Introduction to Variational Methods for Graphical Models》、T. Jaakkola 和 M. I. Jordan（1998）的《Variational Probabilistic Inference and the QMR-DT Database》以及 J. Yedidia 等（2000）的《Generalized Belief Propagation》。循环信念传播是 J. Pearl（1988）的《Probabilistic Reasoning in Intelligent Systems：Networks of Plausible Inference》中描述的一种图模型的近似算法。

近些年，一阶随机推理语言各种形式的创建已经形成了一个重要的研究领域。我们建议阅读 D. Koller 和 A. Pfeffer（1998）的《Probabilistic Frame-Based Systems》、N. Friedman 等（1999）的《Learning Probabilistic Relational Models》、K. Kersting 和 L. DeRaedt（2000）的《Baysian Logic Programs》、R. Ng 和 V. Subrahmanian（1992）的《Probabilistic Logic Programming》以及 Pless 等（2006）的《The Design and Testing of a First-Order Stochastic Modeling Language》。

强化学习的重要贡献包括 R. S. Sutton（1988）的《Learning to Predict by the method of Temporal Differences》、R. S. Sutton 和 A. G. Barto（1998）的《Reinforcement Learning：An Introduction》，还有 M. L. Puterman（1994）的《Markov Decision Processes》。

作者希望感谢他现在和刚毕业的学生特别是一阶随机推理语言循环逻辑（Pless et al. 2006）的创造者 Dan Pless、Chayan Chakrabarti（Chakrabarti et al. 2006）、Roshan Rammohan 以及 Nikita Sakhanenko（Sakhanenko et al. 2007）的很多意见、图表、例子和第 13 章的练习。

13.5　习题

1. 建立一个隐马尔可夫模型表示美式足球游戏的得分序列，其中，达阵 6 分，并跟着两种例外情况会得到 0，1，2 额外的分数。当然，有两个队并且每个都能拿分，如果一队有球且不得分，则发出一个 0。所以假设发出的得分流是 6，1，6，0，3，0，6，1，0，3，0；你的 HMM 将会是什么样？
   a）讨论这个问题以及最优的 HMM 可能是什么样。
   b）再建两个得分流测试你的 HMM。
   c）跟踪一个实际的比赛，看你的系统预测的得分流的效果如何。
2. 建立一个隐马尔可夫模型预测美国棒球游戏的半局分数序列。假设半局序列是 0，0，0，1，0，1，1，2，2，0，0，0，2，0，0，0，0。为简单起见，对于每一队限制得分 0，1，2 在半局击打中。
   a）当允许每半局任何时间段的数量时，怎样变换这个模型？
   b）怎样训练这个系统来得到更实际的得分值？
3. 建立一个图表示 13.1.2 节中分层隐马尔可夫模型。HHMM 可能适合哪种问题情形类型？讨论适用 HMM 结构的应用领域的问题。
4. 给定一个 Viterbi 算法的例子，处理图 13-7 和 13-8 的概率有限状态机：
   a）在观测到的语音中，为什么 new 被看作是比 knee 好的解释？
   b）在概率有限状态机中是怎样处理状态机中 Viterbi 算法的交替的，例如，在单词 new 中对语音 uw 和 iy 的语音选择？
5. 给出 9.3.6 节的隐马尔可夫模型和 Viterbi 算法执行的全过程，包括合适的后向指针的设置，给出观测#，n，iy，t，#如何被处理。
6. 手动运行 13.3.3 节描述的马尔可夫决策过程中的机器人。在决策过程中，使用相同的回报机制并为 a 和 b 选择概率值。
   再次以不同的值 a 和 b 运行机器人。什么决策能给机器人最好的回报机会？
7. 用你选择的语言给 13.3.3 节中机器人支持的 MDP 编程。用不同的 a 值和 b 值对优化回报进行试验。有几个有趣的可能策略：如果充电（recharge）是 A(high) 的一个决策，你的机器人能学习这个决策是次优的吗？在什么情形下机器人总搜索空易拉罐，即 A(low) = recharge 的决策是次优的吗？

8. Jack 是卖车的，他寻找最大化利润的方式。每周 Jack 订一批车，每辆车 d 美元。这些车立即交付。库存中加入新车。然后每个星期，他以每台 c 美元的价格随机卖掉 k 辆车。Jack 也要给他的库存中没有卖掉的车花 u 美元/每台。用马尔可夫决策过程形式化这个问题。状态和动作是什么？回报是多少？转换概率是多少？描述长期回报。
9. 考虑一般领域中方格环境巡航任务，其中有一个目标状态、多个障碍和一个折扣因子 $\gamma < 1$。动作是随机的，智能体尝试移动时可能调到不同的单元。智能体有五个可能的动作：**向北**、**向南**、**向东**、**向西**，或**停留**在相同的位置。考虑撞墙时发生负花销的情形。你能画出一个 3 * 3 的例子环境，其中最好的动作中至少有一个状态是停留吗？如果能，那么给出具体的动作、回报和转换概率。如果不能，请解释原因。
10. 当我们外出就餐时，经常喜欢在离餐馆近的地方停车。假设餐馆在东西向的长街上，长街只允许在一边停车，而且街道上有划分好的停车位。我们从餐馆东边的 D 区开始驶向餐馆，在距离餐馆 s 远的地方有未占用的停车位的概率是 ps，独立于所有停车位。以马尔可夫决策过程形式化这个问题。确定具体 MDP 中的所有元素！（改编自 Puterman 1994）
11. 如何把 13.3 节的 MDP 表示换成 POMDP？把简单的机器人问题和 13.3.3 节中为其建立的马尔可夫转换矩阵，变成 POMDP。提示：考虑使用部分可观测的状态概率矩阵。
12. 在 13.3 节中我们简单讨论了扑克游戏。给定一个玩家现在的（概率）状态是 good bet、questionable bet 或者 bad bet，求解一个 POMDP 来表示这个情形。从概率角度来考虑如何应对对手。
13. 计算给 POMDP 问题完全找到最优策略的复杂性开销。

第五部分　人工智能问题求解的高级课题

精确并不等于完全的真实……

——亨利·马蒂斯

现在的时间和过去的时间也许都存在于未来的时间，而未来的时间又包含了过去的时间……

——T. S. 艾略特，《燃烧的诺顿》

……每一种文的模式的形成都是一系列小动作积累的结果。

——克里斯托夫·亚历山大

自动推理和自然语言

第五部分讲述人工智能的两个重要应用：自然语言理解和自动推理。如果想创造人工智能，一定要提及语言、推理和学习。这些问题的解决方法是基于本书前面章节中介绍过的技术和工具。同时，因为这些问题的重要性，它们也推动了这些工具以及整个人工智能的发展。

在第三部分的介绍中，我们讨论了弱方法问题求解器的应用。使用弱方法的问题包括搜索空间的复杂性和使用通用表示法来表示领域中特殊知识方面的困难。尽管专家系统和类似的强方法求解器取得了很大成功，但许多领域中仍然需要通用方法；实际上，专家系统的控制策略也依赖于好的问题求解方法。自动推理或定理证明组织在弱方法问题求解器方面一直给出了大量有益的工作。这些技术应用于许多重要领域中，包括集成电路设计和验证、程序正确性证明以及同接的 Prolog 语言的创建。第 14 章将围绕自动推理进行介绍。

自然语言理解已经证明是人工智能领域中一项非常困难的任务。原因有很多，最重要的是支持使用语言所需要的知识、能力和经验的数量。成功的语言理解需要理解自然世界、人类心理和社会习俗，这就需要使用与逻辑推理和类比解释完全不同的技术。由于人类语言的复杂性和模糊性，自然语言理解推动了知识表示方面的大量研究。虽然到目前为止，这些努力只取得了部分成功，但是运用基于知识的方法，研究人员成功开发出了能够理解特定领域中自然语言的程序。然而这些技术能否解决语言理解问题仍然是一个有争议的问题。

第 7 章中提到的表示技术，如语义网、脚本和框架一直支配着自然语言方面的早期人工智能工作。最近，语言模式的相关性分析在语言理解中占有重要的地位。语言表达不是声音和单词的随机组合，而是具有一定模式。贝叶斯技术可以为这些语言结构建模。第 13 章介绍的几种模型包括 2-gram 和 3-gram。对于理解语言音素和单词结合非常重要。第 15 章将介绍自然语言理解中的基于知识的语法、语义和随机技术。

随着万维网的发展和人们对搜索引擎的使用，自然语言处理变得更加重要。文本挖掘（或者称为无结构文本中有用信息的通用搜索）和信息摘要（或者称为"理解"文本和抽取关键事件）都是重要的新技术。第 15 章将介绍这些软件工具的基础：基于符号模式识别和随机模式识别。

最后，本书的补充材料中给出了许多支持自然语言理解的数据结构，包括用 Prolog、LISP 和 Java 实现的语义网和框架系统，还用 Prolog 和 Java 创建了一个通用下降语义网解析器和一个基于 Earley 算法与动态规划的解析器。第 16 章讨论了目前语言理解、学习和复杂问题求解中的一些局限性。

第 14 章　自 动 推 理

对于何以可能，敏锐的人说：当给我一个概念时，我能够联想到更多，并把它和另外一些不包括在内的概念联系起来。用这种方式，好像后者必定属于前者？

——伊曼纽尔·康德，“未来形而上学导论”

任何理性决定都可以看作是从特定前提得到的结论……如果一个理性的人把他的决策建立在他为自己规定的重要的和实际前提的基础上，那么他的行为是能够被控制的。

——西蒙，《决策和行政组织》，1944

推理是一门艺术，而不是一门科学……

——Wos 等，《自动推理》，1984

14.0　定理证明中的弱方法

Wos 等（1984）将自动推理程序描述为“使用明确而严格的符号表示信息，用精确的推理规则做出结论，以及用仔细描述的策略控制这些推理规则”的程序。他们还认为将策略应用到推理规则推出新信息的方法是一门艺术：“好的表示法包括能够增加求解问题机会的符号和虽然不必要但是有帮助的信息。好的推理规则要与所选表示法啮合良好。好的策略要能够以急剧提高推理程序效率的方式控制推理规则。”

自动推理使用问题求解的弱方法。它通常使用一种统一的表示法，如一阶谓词演算（见第 2 章）、Horn 子句演算（见 14.3 节）或者归结用的子句形式（见 14.2 节）。其推理规则是可靠的，如果可能的话，它也是完备的。它使用通用策略（如宽度优先、深度优先或者最佳优先搜索）和本章中将提到的启发式方法（如成组支持策略和单个优先策略）来对付穷举搜索中的组合爆炸问题。设计搜索策略，特别是启发式搜索策略，更像是一门艺术；我们无法保证这些策略能只使用合理数量的时间和内存就为一个问题找到一个有用解。

问题求解弱方法是一个重要的工具，同时也是问题求解强方法的本质基础。产生式系统和基于规则的专家系统外壳都是弱方法问题求解的例子。虽然产生式系统和基于规则的专家系统的规则中使用了强问题求解启发式，但是其实现仍然依赖于通用（弱方法）推理策略的支持。

问题求解弱方法的技术从一开始就是人工智能研究的焦点。这些技术通常都列在定理证明这个标题下，但我们推荐使用更加通用的名称——自动推理。本章以一个早期的自动推理的例子开始（见 14.1 节），即通用问题求解器，它使用手段 - 目的分析和差别表来控制搜索。

14.2 节介绍自动推理研究中的一个重要成果——归结反驳（反演）系统（resolution refutation system）。我们将讨论归结定理证明中使用的表示语言、归结推理规则、搜索策略和解答抽取过程。作为 Horn 子句推理的例子，在 14.3 节中讨论 Prolog 的推理引擎，并且用一个基于归结定理证明程序的解释器说明 Prolog 语言如何推动说明性程序设计哲学。我们用对自然演绎、等式处理和更复杂推理规则的一些简要解释来结束本章（见 14.4 节）。

14.1 通用问题求解器和差别表

通用问题求解器（General Problem Solver，GPS）（Newell and Simon 1963b；Ernst and Newell 1969）由 Allen Newell 和 Herbert Simon 在卡内基－梅隆大学和后来的卡内基研究所开发出来。其根源可以追溯到早期 Newell、Shaw 和 Simon（Newell and Simon 1963a）开发的被称为逻辑理论家（Logic Theorist，LT）的计算机程序。LT 程序证明了 Russell 和 Whitehead 的《数学原理》（Whitehead and Russell 1950）中的许多定理。

与所有弱方法问题求解器一样，逻辑理论家使用了统一的表示法和可靠的推理规则，并应用了几个策略或启发式方法来指导求解过程。逻辑理论家使用命题演算（见 2.1 节）作为其表示法。推理规则是代换、置换和分离。

代换使得在任意表达式中，公理或者证明为真的定理中的一个符号可以被代换。例如，(B∨B)→B 中可以用¬ A 代换 B 得到(¬ A∨¬ A)→¬ A。

置换使得连接词可以置换为其定义或者等价的形式。例如，¬ A∨B 和 A→B 是逻辑等价的，所以(¬ A∨¬ A)可以置换为(A→¬ A)。

分离是称为取式假言推理（modus ponens）（见第 2 章）的推理规则。

逻辑理论家以宽度优先、目标驱动的方式将这些推理规则应用到要证明的定理中，用于尝试找到一系列操作，最终可以推导到公理或已知为真的定理。逻辑理论家使用的策略包括组织在一个可执行程序中的四个方法：

- 第一个，直接将代换方法应用到当前目标，尝试与已知的所有公理和定理匹配。
- 第二个，如果没有推出证明，那么将所有可能的分离和置换应用到目标，然后再用代换方法测试每个结果是否成功。如果代换方法无法将这些结果与目标进行匹配，那么将它们加入到子问题列表中。
- 第三个，应用链式方法寻找新的子问题，即利用蕴涵传递性。如果子问题能够解决，那么也能够提供一个证明。因此，如果 a→c 是要证的问题，而且已知 b→c，那么可以将 a→b 设置为一个新的子问题。
- 第四个，如果对最初的问题应用上述三种方法都失败，那么转到子问题列表，选择下一个没有尝试过的子问题。

可执行程序不断地应用这四个方法，直到找到解、子问题列表为空或者分配的内存和时间用尽。按照这种方式，逻辑理论家对问题空间执行目标驱动的宽度优先搜索。

可执行程序中使代换、置换和分离推理规则生效的部分是匹配过程。假设我们要证明 p→(q→p)。首先，匹配过程确定一条公理 p→(q∨p)是最合适的，即从领域定义的差别来看匹配是最接近的，因为主要的连接词→在两个表达式中是相同的。然后，匹配过程确认主连接词左边的表达式是相等的。最后，匹配过程识别主连接词右边表达式的差别。→和∨之间的差别很明显地引出证明定理所需的置换。匹配过程帮助控制应用所有代换、置换和分离所必须的（穷举）搜索。实际上，匹配过程排除了足够多的尝试和错误，使得逻辑理论家成为一个成功的问题求解器。

逻辑理论家证明的一个例子显示了匹配过程的威力。《数学原理》（Principia Mathematica）中的定理 2.02 是 p→(q→p)。匹配发现公理 p→(q∨p)最适合的置换。¬ q 到 q 的代换就证明了该定理。匹配以及对代换和置换规则的控制直接证明了该定理，而没有对其他公理或定理的任何搜索。

另一个例子，假设我们希望逻辑理论家证明：

$(p \rightarrow \neg p) \rightarrow \neg p$

1) $(A \vee A) \rightarrow A$　　匹配过程确定5个公理中最"好"的。
2) $(\neg A \vee \neg A) \rightarrow \neg A$　用$\neg A$代换A，以便应用从$\vee$和$\neg$到$\rightarrow$的置换，然后用p代换A。
3) $(A \rightarrow \neg A) \rightarrow \neg A$
4) $(p \rightarrow \neg P) \rightarrow \neg P$

证毕

最初的逻辑理论家使用5条公理用大约10秒钟证明了这一定理。实际证明只需要两步，不需要搜索。匹配过程第一步选择最合适的公理，因为其形式与要建立的结论最相似：（表达式）$\rightarrow$命题。然后用$\neg A$代换A。这样使得第二步的置换和最后一步可以进行，因为目标需要的是$\rightarrow$，而不是$\vee$。

逻辑理论家不仅是自动推理系统的第一个例子，而且说明了搜索策略和启发式方法在推理程序中的重要性。在许多实例中，逻辑理论家用很少的步骤就能找到穷举搜索可能永远无法找到的答案。有些定理用逻辑理论家无法证明，Newell等人指出了对此类问题可能的改进。

大约在同一时间，卡内基的研究人员和另外一些耶鲁的研究人员（Moore and Anderson 1954）开始考察求解逻辑问题的人类主体的思考－发声（think-aloud）协议。虽然最初的目标是确定能够解决这类问题的人类过程，但是研究人员却开始比较人类与计算机程序（如逻辑理论家）解决问题方法的不同。这成为现在所谓的信息处理心理学的首个实例，基本思想是生物体的可观察行为是由生成行为的原始信息处理程序提供的（Newell et al. 1958）。这一研究也是现代*认知科学*（见16.2节，Luger 1994）的一些奠基性工作。

对这些协议的仔细推敲显示出逻辑理论家解题与人类有许多不同。人类行为被证明是显示出匹配和差别减小机制的，称为*手段－目的分析*。在手段－目的分析方法中，差别减小方法（*手段*）与要减小的差别（*目的*）密切相关：差别减小操作按照能够减小的差别分类。

在一个非常简单的例子中，若开始状态为$p \rightarrow q$，目标是$\neg q \vee q$，那么差别包括开始的$\rightarrow$符号和目标的$\vee$的差别，以及开始的p和目标$\neg q$的差别。差别表中包含将$\rightarrow$置换为$\vee$以及去掉$\neg$的各种不同的方法。可以一次尝试一个变换，直到差别都消除，定理被证明。

在最有意思的问题中，开始和目标之间的差别无法直接减小。这种情况下，先寻找一个能够部分减小差别的操作。整个过程递归地应用到这些结果，直到不存在差别。这可能还需要下面不同的搜索路径，用减小差别的不同应用来表示。

图14-1a取自Newell和Simon（1963b），它包含了12条转换规则，中间一列是要解决的逻辑问题，右边一列给出何时应用这些转换的指示。

图14-1b显示了由一个人类主体产生的一个证明，该证明取自Newell和Simon（1963b）。证明前，图14-1a中的转换规则提供给人类主体，人类主体没有形式逻辑的经验，要求他将$(R \supset \neg P) \cdot (\neg R \supset Q)$变换为$\neg (\neg Q \cdot P)$。在第2章的符号中，~是$\neg$，·是$\wedge$，$\supset$是$\rightarrow$。图14-1a中的$\rightarrow$和$\leftrightarrow$表示合法的置换。图14-1b中最右边的一列表示证明过程中每一步应用的图14-1a中的规则。

Newell和Simon（1963b）将人类主体的问题求解策略称为*差别减小*，把使用适当的转换减小特定问题差别的通用过程称为*手段－目的分析*。使用差别减小应用手段－目的分析的算法是*通用问题求解器*。

R 1.	$A \cdot B \rightarrow B \cdot A$ $A \vee B \rightarrow B \vee A$	只应用于主表达式
R 2.	$A \supset B \rightarrow \sim B \supset \sim A$	只应用于主表达式
R 3.	$A \cdot A \leftrightarrow A$ $A \vee A \leftrightarrow A$	A和B是两个主表达式
R 4.	$A \cdot (B \cdot C) \leftrightarrow (A \cdot B) \cdot C$ $A \vee (B \vee C) \leftrightarrow (A \vee B) \vee C$	A和A⊃B是两个主表达式
R 5.	$A \vee B \leftrightarrow \sim(\sim A \cdot \sim B)$	A⊃B和B⊃C是两个主表达式
R 6.	$A \supset B \leftrightarrow \sim A \vee B$	
R 7.	$A \cdot (B \vee C) \leftrightarrow (A \cdot B) \vee (A \cdot C)$ $A \vee (B \cdot C) \leftrightarrow (A \vee B) \cdot (A \vee C)$	
R 8.	$A \cdot B \rightarrow A$ $A \cdot B \rightarrow B$	只应用于主表达式
R 9.	$A \rightarrow A \vee X$	只应用于主表达式
R 10.	$\left.\begin{matrix} A \\ B \end{matrix}\right\} \rightarrow A \cdot B$	A和B是两个主表达式
R 11.	$\left.\begin{matrix} A \\ A \supset B \end{matrix}\right\} \rightarrow B$	A和A⊃B是两个主表达式
R 12.	$\left.\begin{matrix} A \supset B \\ B \supset C \end{matrix}\right\} \rightarrow A \supset C$	A⊃B和B⊃C是两个主表达式

图 14-1a　逻辑问题的转换规则

[摘自 Newell 和 Simon（1961）]

图 14-2 显示了通用问题求解器的控制图和连接表。目标是将表达式 A 转化为表达式 B。第一步是定位 A 和 B 之间的差别 D。第一行第二个框中确定了子目标：减小 D；第三个框表示差别减小是递归的。“减小”在第二行进行，它为差别 D 确定了操作符 Q。实际上，从连接表中确定的是一个操作符列表。当前选择的操作符无法执行时（例如没有通过可行性检测），这个列表根据为减小差别提供了一个选择顺序，例如，在图 14-2 的第三行，操作符执行完成，D 被减小。

通用问题求解模型需要两个组成部分。第一个是刚刚提到的通用过程，负责比较两个状态描述并减小它们的差别。第二个是*连接表*，给出问题差别和减小它们的特定转换之间的链接，它只适用于特定领域。图 14-2 给出了命题演算表达式的差别及其减小方法（图 14-1a 中的 12 条转换规则）。其他连接表还有：用于减小代数式中的差别的，用于像汉诺塔这样的任务的，或者用于像国际象棋这样的更复杂的博弈问题的。由于差别表的模块化，例如，表格根据不同的应用进行修改，这个问题求解器称为一般化。Ernst 和 Newell（1969）论述了通用问题求解技术的许多不同应用领域。

1.	$(R \supset \sim P) \cdot (\sim R \supset Q)$	$\sim(\sim Q \cdot P)$
2.	$(\sim R \vee \sim P) \cdot (R \vee Q)$	规则6应用于1的左边和右边
3.	$(\sim R \vee \sim P) \cdot (\sim R \supset Q)$	规则6应用于1的左边
4.	$R \supset \sim P$	规则8应用于1
5.	$\sim R \vee \sim P$	规则6应用于4
6.	$\sim R \supset Q$	规则8应用于1
7.	$R \vee Q$	规则6应用于6
8.	$(\sim R \vee \sim P) \cdot (R \vee Q)$	规则10应用于5和7
9.	$P \supset \sim R$	规则2应用于4
10.	$\sim Q \supset R$	规则2应用于6
11.	$P \supset Q$	规则12应用于6和9
12.	$\sim P \vee Q$	规则6应用于11
13.	$\sim(P \cdot \sim Q)$	规则5应用于12
14.	$\sim(\sim Q \cdot P)$	规则1应用于13

图 14-1b　命题演算中的一个定理证明

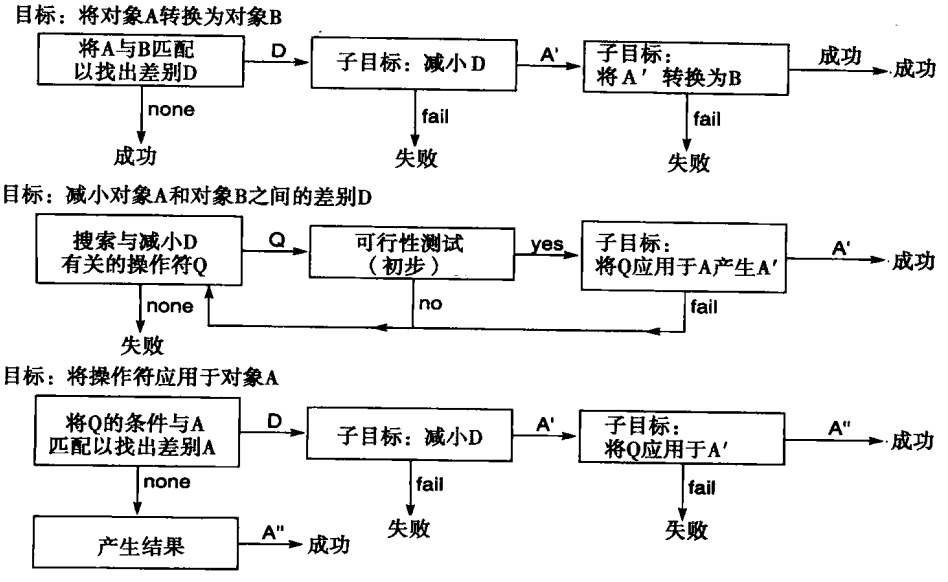
[摘自 Newell 和 Simon（1961）]

问题领域中不同差别减小结构化方法对组织该领域中的搜索很有帮助。在差别减小表中转换规则的顺序中蕴涵了减小不同差别类的启发式或优先级顺序。这种优先级顺序可以将通用转换规则放在特殊转换规则之前，也可以按照领域专家认为最合适的顺序给出。

许多研究方向由通用问题求解器方面的工作发展而来。其中之一是使用人工智能技术分析人类的问题求解行为。需要特别指出的是，产生式系统代替了通用问题求解的手段－目的方法，成为人类信息处理建模的首选形式（见第 16 章）。现代基于规则的专家系统的产生式规则取代了通用问题求解差别表中的特定条目（第 8 章）。

在通用问题求解的另一个有意思的演变中，差别表变成用于规划的操作符表，如 STRIPS 和 ABSTRIPS。规划在机器人问题求解中非常重要。机器人要完成一项任务，如到隔壁房间拿回一件东西，计算机必须生成一个规划。该规划安排机器人的行动：放下现在拿着的所有东西，穿过房门，走到要拿的东西旁边，等等。STRIPS（斯坦福研究所问题求解器，Fikes and Nilsson 1971，Fikes et al. 1972，Sacerdotti 1974）的规划构成中使用一个与通用问题求解差别表类似的操作符表。表中的每个操作符（机器人的基本动作）具有相应的前置条件，这与图 14-2 中的可行性检查非常相似。操作符表还包含增加和删除列表，负责在操作符应用后更新“世界”模型。在 7.4 节介绍了一个类 STRIPS 规划器，并在补充软件材料中用 Prolog 实现了。

总而言之，人工智能中的第一个自动推理模型在卡内基研究所开发的逻辑理论家和通用问题求解器中构建。这些程序为问题求解弱方法提供了所有必备条件：统一的表示法、可靠的推理规则集合以及应用这些规则的方法或策略集合。相同的组成部分构成了归结证明过程，这是自动推理的更先进更有力的基础。



对于文本的逻辑任务：

可行性测试（初步）

中间连接词是相同的吗（例如，A·B→B对P∨Q失效）？

操作符太大吗（例如，A→B）·（A∨C）→A∨（B·C）对P·Q失效）？

操作符太容易吗（例如，A→A·A可应用于任何情况）？

限制条件被满足了吗（例如，R8只应用于主表达式）？

连接表

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
增加条目			X				X		X	X	X	X
删除条目			X				X	X			X	X
改变连接词					X	X	X					
改变记号					X							
改变低层记号		X			X	X						
改变分组				X			X					
改变位置	X	X										

X表示：规则的某个变量是相关的。GPS将挑选合适的变量。

图 14-2 通用问题求解程序的流程图和差别减小表

[摘自 Newell 和 Simon（1963b）]

14.2 归结定理证明

14.2.1 概述

归结是命题演算或谓词演算中的一种定理证明技术，从20世纪60年代中期开始成为人工智能问题求解研究的一部分（Bledsoe 1977，Robinson 1965，Kowalski 1979b）。归结是一个可靠的推理规则，当用来生成一个反驳（见14.2.3节）时，归结又是完备的。在一个重要的实际应用中，归结定理证明（特别是归结反驳系统）使得目前的Prolog解释器成为可能（见14.3节）。

归结原理是Robinson（1965）在一篇重要的论文中提出的，是一种使用最少代换在子句数据库中发现矛盾的方法。归结反驳通过将要证明的命题取反并将此取反后的目标加入到已知为真的公理集合中来证明定理。然后使用推理的归结规则说明这将导致矛盾。如果定理证明程序说明目标的反与给定的公理集合不一致，那么就证明原来的目标是一致的。这样就证明了定理。

归结反驳证明包括以下步骤：

1）将前提或公理转化为*子句形式*（见14.2.2节）。

2）将要证明的命题取反，转化为子句形式，加入公理集合。

3）归结这些子句，生成可以从逻辑上推导出的新子句（见14.2.3节）。

4）通过生成空子句得出矛盾。

5）用来得出空子句的代换是那些使得取反后的目标的反（即最初要证明的）为真的代换（见14.2.4节）。

从第2章得出，归结是一条可靠的推理规则。然而，却是不完备的。归结是*反驳完备的*，即无论子句集中是否存在矛盾，总可以得到空子句。我们将在14.2.4节中介绍反驳策略时对其进行更多的探讨。

归结反驳证明要求将公理和目标的反化为一种正规的形式，称为*子句形式*。子句形式将逻辑数据库表示为文字的析取的集合。文字是一个原子表达式或原子表达式的反。

归结的最一般形式称为*二元归结*。当两个子句中一个包含一个文字，另一个包含该文字的反时，就可以应用二元归结。如果文字中包含变量，必须将其合一，使它们等价。接着生成一个包含两个子句中除该文字和该文字的反之外的所有谓词的析取组成的新子句，称其被“归结掉”。产生的子句要进行使得谓词及其反可以被视为“等价”的合一代换。

在接下来进行更详细的讨论之前，我们来看一个简单的例子。归结提供一个与取式假言推理类似的证明。这并不是说明这些推理规则是等价的（实际上归结比取式假言推理更通用），仅仅是给读者一个感觉。

我们要从“Fido是狗”、“所有的狗都是动物”和“所有动物都会死”中证明“Fido会死”。将这三个前提写为谓词，并应用取式假言推理得到：

1) *所有的狗都是动物*: ∀(X)(dog(X)→animal(X)).

2) Fido *是狗*: dog(fido).

3) *由肯定前件的假言推理和*{fido/X} *可得*: animal(fido).

4) *所有动物都会死*: ∀(Y)(animal(Y)→die(Y)).

5) *由肯定前件的假言推理和*{fido/X} *可得*: die(fido).

归结的等价推理将这些谓词变换为子句形式：

谓词形式	*子句形式*
1) ∀(X)(dog(X)→animal(X))	¬ dog(X) ∨ animal(X)
2) dog(fido)	dog(fido)
3) ∀(Y)(animal(Y)→die(Y))	¬ animal(Y) ∨ die(Y)

将结论“Fido会死”取反：

4) ¬ die (fido)	¬ die (fido)

对具有相反文字的子句进行归结，通过归结生成新的子句，如图14-3所示。这个过程常常称为*冲突*（clashing）。

图14-3中的符号□表示产生了空子句，发现了矛盾。□表示一个谓词与其反的冲突：两条互相冲突的语句出现在子句空间中的情况。这两条语句冲突生成空子句。使得谓词等价的代换（合一）序列给出了目标为真时变量的值。例如，如果问是否有东西会死，那么目标的反就是¬(∃(Z)die(Z))，而不是¬ die(fido)。图14-3中的代换{fido/Z}确定fido是会死的动物的一个实例。在14.2节剩余的部分将会对本例中隐含的问题进行详细的分析。

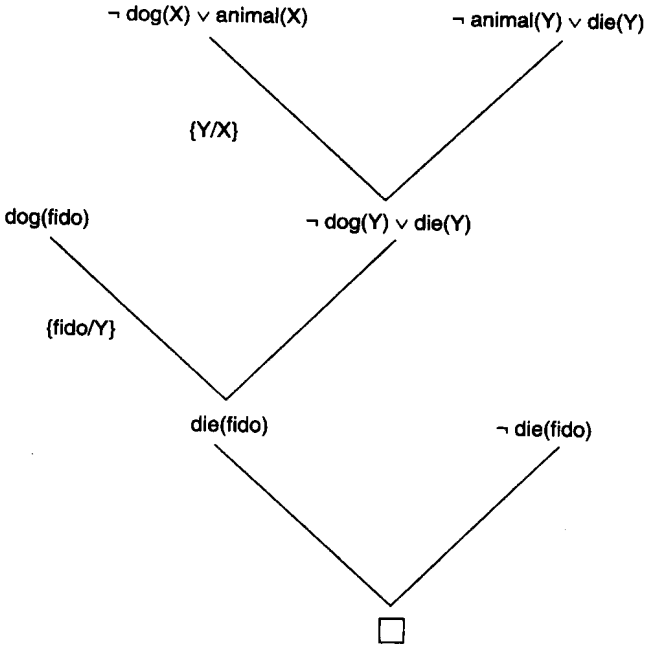


图 14-3　“死狗”问题的归结证明

14.2.2　为归结反驳生成子句形式

归结证明过程需要将数据库中描述一个状态的所有语句转化为一个标准形式，称为子句形式。原因在于归结是一个作用在析取子句对上生成新的析取子句的操作符。数据库采用的形式可以看作是析取子句的合取。合取的原因是组成数据库的所有子句在同一时刻都假定为真。每个单独的子句中都是析取，以析取符号（或 ∨）作为连接符。这样，图 14-3 的整个数据库可以表示为下面的子句形式：

$$(\neg\ \mathrm{dog}(X) \vee \mathrm{animal}(X)) \wedge (\neg\ \mathrm{animal}(Y) \vee \mathrm{die}(Y)) \wedge (\mathrm{dog}(\mathrm{fido}))$$

将要证明的命题的反（用合取）加入到此表达式，在本例中是¬ die（fido）。一般地，数据库可以写成析取式集合的形式，操作符 ∧ 可以省略。

现在介绍一个算法，其中包含一个变换序列，可以将任意的谓词演算语句集合化简为子句形式。已经证明，这些变换可以用来将任意谓词演算表达式集合化简为子句形式，并且当且仅当原来的表达式集合不一致时子句集合不一致（Chang and Lee 1973）。因为有些内容会丢失，子句形式并不严格地等价于原谓词演算表达式集合，原因是斯柯伦化限制了已存在的量化变量的可能的代换（Chang 和 Lee 1973）。但是，不可满足性仍然保持。也就是说，如果在原谓词演算表达式集合中存在矛盾（反驳），那么在子句形式中也存在矛盾。转换并没有牺牲反驳证明的完备性。

我们通过一个例子来说明析取式的正规形式化简过程，并给出每一步简要的合理说明。但这并不证明这些变换在所有谓词演算表达式上都是等价的。

在下面的表达式中，根据第 2 章的讨论，大写字母代表变量（W、X、Y 和 Z）；字母表中部的小写字母代表常量或者约束变量（l、m 和 n）；前面的小写字母代表谓词名称（a、b、c、d 和 e）。为改善表达式的可读性，我们使用两种类型的括号：（）和［］，并且去除多余的括号。作为一

个例子，看下面的表达式，其中 X、Y 和 Z 是变量，I 是常量：

(i) (∀X)([a(X) ∧ b(X)]→[c(X, I) ∧ (∃Y)((∃Z)[c(Y, Z)]→d(X, Y))]) ∨ (∀X)(e(X))

1）首先，使用第 2 章中已经证明的等式：a→b≡¬ a∨b 消去→。上面（i）中的表达式化简为：

(ii) (∀X)(¬ [a(X) ∧ b(X)] ∨ [c(X, I) ∧ (∃Y)((∃Z)[¬ c(Y, Z)] ∨ d(X, Y))] ∨ (∀X)(e(X))

2）然后，化简反的范围。可以利用第 2 章中的几个变换来实现，包括：

¬ (¬ a) ≡ a

¬ (∃X) a(X) ≡ (∀X) ¬ a(X)

¬ (∀X) b(X) ≡ (∃X) ¬ b(X)

¬ (a ∧ b) ≡ ¬ a ∨ ¬ b

¬ (a ∨ b) ≡ ¬ a ∧ ¬ b

使用第 4 个等式，（ii）变为：

(iii) (∀X)([¬ a(X) ∨ ¬ b(X)] ∨ [c(X, I) ∧ (∃Y)((∃Z)[¬ c(Y, Z)] ∨ d(X, Y))]) ∨ (∀X)(e(X))

3）然后，通过重命名所有变量进行标准化，使得不同量词限制的变量具有惟一的名字。第 2 章中已经指出，因为变量名字是“哑元”或“占位符”，所以选择特定的名字并不影响真值或子句的一般性。本步中应用的变换具有以下形式：

((∀X) a(X) ∨ (∀X) b(X)) ≡ (∀X) a(X) ∨ (∀Y) b(Y)

因为（iii）中有变量 X 的两个实例，所以重命名：

(iv) (∀X)([¬ a(X) ∨ ¬ b(X)] ∨ [c(X, I) ∧ (∃Y)((∃Z)[¬ c(Y, Z)] ∨ d(X, Y))]) ∨ (∀W)(e(W))

4）将所有量词不改变顺序全部移到左边。第 3 步消除了变量名称之间可能的冲突，所以本操作可以进行。现在（iv）变为：

(v) (∀X)(∃Y)(∃Z)(∀W) ([¬ a(X) ∨ ¬ b(X)] ∨ [c(X, I) ∧ (¬ c(Y, Z) ∨ d(X, Y))]) ∨ e(W))

第 4 步后，子句称为*前束范式*，因为所有量词都在前面作为*前缀*，表达式或*母式*跟在后面。

5）在此，所有存在量词都可以被一个称为*斯柯伦化*（skolemization）的过程消除。表达式（v）中有一个 Y 的存在量词。当表达式中包含一个存在量化变量时，如（∃Z）（foo（…, Z, …）），可以推断出 Z 有一个赋值可以使得 foo 为真。斯柯伦化确定这样一个值。斯柯伦化不必指出如何得出该值，它只是一种为*必然存在*的赋值给出名称的方法。如果 k 代表该赋值，那么得到 foo（…, k, …）。这样：

(∃X)(dog(X)) 可以置换为 dog(fido)

其中，名字 fido 从 X 的定义域中选出，表示个别的 X。fido 称为*斯柯伦常量*（skolem constant）。如果谓词中含有一个以上的参数，而且存在量化变量在全称量化变量的作用范围之内，那么存在量化变量一定是其他变量的一个函数。可以在斯柯伦化过程中表示为：

(∀X)(∃Y)(mother(X, Y))

此表达式表示每个人都有一个母亲。每个人用 X 表示，存在的母亲是选定的特定人 X 的函数。这样斯柯伦化得到：

$(\forall X)\ mother(X, m(X))$

这个表达式表示每个 X 都有一个母亲（与 X 对应的 m）。另一个例子：

$(\forall X)(\forall Y)(\exists Z)(\forall W)\ (foo(X, Y, Z, W))$

斯柯伦化为：

$(\forall X)(\forall Y)(\forall W)\ (foo(X, Y, f(X, Y), W))$

存在量化变量 Y 和 Z 在全称量化变量 X 的作用范围之内（右边），但是不在 W 的范围之内。所以每个变量都要用 X 的一个斯柯伦函数代替。用斯柯伦函数 f(X) 代替 Y，用 g(X) 代替 Z，（v）变为：

(vi) $(\forall X)(\forall W)\ ([\neg a(X) \vee \neg b(X)] \vee [c(X, I) \wedge (\neg c(f(X), g(X)) \vee d(X, f(X)))]) \vee e(W))$

斯柯伦化以后，就可以进行第 6 步，简单地去掉前缀。

6）去掉所有全称量化。到这一步，只剩全称量化变量（第 5 步），而且没有变量冲突（第 3 步）。这样所有量词都可以去掉，以后的证明过程假定所有变量都是全称量化的。公式（vi）现在变为：

(vii) $[\neg a(X) \vee \neg b(X)] \vee [c(X, I) \wedge (\neg c(f(X), g(X)) \vee d(X, f(X)))] \vee e(W)$

7）然后将表达式转化为析取子句的合取形式。需要使用 ∨ 和 ∧ 的结合率和分配率。回忆第 2 章中：

$a \vee (b \vee c) = (a \vee b) \vee c$
$a \wedge (b \wedge c) = (a \wedge b) \wedge c$

说明 ∨ 或 ∧ 可以按照任意的形式分组。必要时也要使用第 2 章中的分配率。因为

$a \wedge (b \vee c)$

已经是子句形式，所以 ∧ 不需要分配。然而，∨ 必须在 ∧ 上分配：

$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

（vii）的最后形式为：

(viii) $[\neg a(X) \vee \neg b(X) \vee c(X, I) \vee e(W)] \wedge$
$[\neg a(X) \vee \neg b(X) \vee \neg c(f(X), g(X)) \vee d(X, f(X)) \vee e(W)]$

8）将合取分为单独的子句。在（viii）的例子中，有两个子句：

(ixa) $\neg a(X) \vee \neg b(X) \vee c(X, I) \vee e(W)$
(ixb) $\neg a(X) \vee \neg b(X) \vee \neg c(f(X), g(X)) \vee d(X, f(X)) \vee e(W)$

9）最后一步要再次分开地来标准化变量。需要给第 8 步生成的每个子句中的变量取不同的名字。这个过程源自第 2 章中建立的等式

$(\forall X)(a(X) \wedge b(X)) \equiv (\forall X)a(X) \wedge (\forall Y)b(Y)$

由于变量名的本质是占位符。使用新变量名称 U 和 V 后，（ixa）和（ixb）变为：

(xa) $\neg a(X) \vee \neg b(X) \vee c(X, I) \vee e(W)$

(xb) ¬ a(U) ∨ ¬ b(U) ∨ ¬ c(f(U), g(U)) ∨ d(U, f(U)) ∨ e(V)

最后一步标准化的重要性只有在介绍完归结的合一步骤以后才会表现出来。我们寻找最广的合一使得分处于两个子句中的两个谓词等价，然后在每个子句中具有相同名称的变量上进行这个代换。这样，如果某些变量（没必要）与其他变量具有相同的名字，将会被合一过程使用一个代换重命名，使得答案丧失一般性。

这个9步的过程用于将任意谓词演算表达式集合变换为子句形式。同时不丢失归结反驳的完备性。下面说明从这些子句中生成证明的归结过程。

14.2.3　二元归结证明过程

*归结反驳*证明过程通过将子句集化简出一个矛盾（用空子句□代表）来回答提问或推导新结果。通过归结数据库中的子句对来得出矛盾。如果一个归结没有直接得出矛盾，那么将归结得到的子句（即*归结式*）加入子句数据库，并继续归结过程。

在介绍归结过程如何在谓词演算中工作之前，我们先给出一个命题演算或者自由变量演算的例子。考虑下面的两个命题演算的父子句 p1 和 p2：

p1: $a_1 \vee a_2 \vee \cdots \vee a_n$

p2: $b_1 \vee b_2 \vee \cdots \vee b_m$

具有两个文字 a_i 和 b_j，其中 $1 < i \leqslant n$ 且 $1 < j \leqslant m$，使得 $\neg a_i = b_j$。二元归结生成子句：

$$a_1 \vee \cdots \vee a_{i-1} \vee a_{i+1} \vee \cdots \vee a_n \vee b_1 \vee \cdots \vee b_{j-1} \vee b_{j+1} \vee \cdots \vee b_m$$

上面的符号说明归结式由两个父子句中除文字 a_i 和 b_j 外的所有文字的析取组成。

一个简单的论据可以给出隐藏在归结原理背后的直观知识。假设

a ∨ ¬ b 和 b ∨ c

都是成立的语句。观察到总有 b 和¬ b 中一个为真，另一个为假（b ∨ ¬ b 是永真式）。因此

a ∨ c

之一一定为真。a ∨ c 是两个父子句 a ∨ ¬ b 和 b ∨ c 的归结式。

现在考虑一个命题演算的例子，要从下面的公理（当然，对于所有的命题 l 和 m 有 l←m ≡ m→l) 中证明 a：

a←b ∧ c
b
c←d ∧ e
e ∨ f
d ∧ ¬ f

将第一个公理化为子句形式：

a←b ∧ c
a ∨ ¬ (b ∧ c)　　根据 l→m ≡ ¬ l ∨ m
a ∨ ¬ b ∨ ¬ c　　根据德·摩根定律

化简其余的公理，得到下面的子句：

a ∨ b¬ ∨ ¬ c

b
c∨¬ d∨¬ e
e∨f
d
¬ f

归结证明见图 14-4。首先，将要证明的目标 a 取反后加入子句集。导出□表示子句数据库是不一致的。

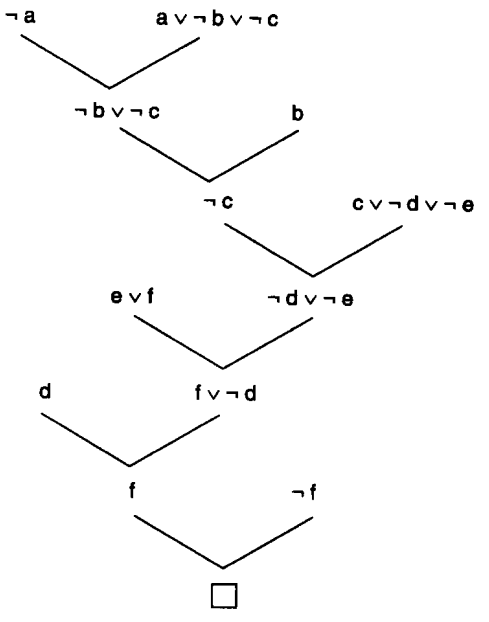


图 14-4　一个命题演算例子的归结证明

要在文字中可能包含变量的谓词演算中运用二元归结，必须有一个过程使得两个文字在具有不同变量名称或者一个是常量时可以看作是等价的。2.3.2 节将合一定义为确定使两个谓词等价的一致最广代换的过程。

谓词演算上的归结算法与命题演算归结算法非常相似，除了以下几点：

1）父子句中文字及其反只有当在某个代换 σ 下合一时才能生成归结式。然后要在把归结式加入子句集之前将 σ 应用到该归结式。要求 σ 是父子句的最广合一子。

2）用来找到矛盾的合一代换提供了原始查询为真的变量绑定。我们将在 14.2.5 节解释这个称为*解答抽取*的过程。

有时，一个子句中两个或两个以上的文字有一个合一代换。出现这种情况时，即使包含该子句的子句集是矛盾的，该子句集中也可能会不存在反驳。例如，考虑下面的子句：

p(X) ∨ p(f(Y))
¬ p(W) ∨¬ p(f(Z))

读者需注意，使用简单的归结，这些子句只能化简为等价的式子或者重言式，但是不能得出矛盾，即没有代换使得它们不一致。

这种情况可以通过这种子句的*因式分解*来处理。如果一个子句中文字的一个子集有一个最广合一子（见 2.3.2 节），那么用一个新的子句置换原子句，这个新子句称为子句的*因式*。因式是原来子句应用最广合一子代换并去掉冗余文字后生成的子句。例如，p(X) ∨ p(f(Y)) 的两个文字在代换{f(Y)/X}下会合一。将代换应用到两个文字后得到新子句，p(f(Y)) ∨ p(f(Y)) 然后用因式 p(f(Y)) 代替此子句。任何包含因式分解的归结反驳系统都是反驳完备的。14.2.2 节中的第 3 步分别对变量标准化，可以解释为因式分解的一次小应用。在 14.4.2 节描述的*超归结*中，因式分解也可以作为推理过程的一部分处理。

下面，我们介绍谓词演算归结反驳的一个例子。看下面“快乐的学生”的故事：

任何通过历史考试和中奖的人都是快乐的。任何努力学习或者幸运的人都能够通过所有考试。约翰不学习但是很幸运。任何幸运的人都能中奖。约翰快乐吗？

首先将句子化为谓词形式：

任何通过历史考试和中奖的人都是快乐的。

$\forall X(pass(X, history) \wedge win(X, lottery) \rightarrow happy(X))$

任何努力学习或者幸运的人都能够通过所有考试。

$\forall X \forall Y(study(X) \vee lucky(X) \rightarrow pass(X, Y))$

约翰不学习但是很幸运。

$\neg study(john) \wedge lucky(john)$

任何幸运的人都能中奖。

$\forall X(lucky(X) \rightarrow win(X, lottery))$

将这 4 条谓词语句转化为子句形式（见 14.2.2 节）：

1. $\neg pass(X, history) \vee \neg win(X, lottery) \vee happy(X)$
2. $\neg study(Y) \vee pass(Y, Z)$
3. $\neg lucky(W) \vee pass(W, V)$
4. $\neg study(john)$
5. $lucky(john)$
6. $\neg lucky(U) \vee win(U, lottery)$

将结论的否定以子句形式加入子句集中：

7. $\neg happy(john)$

图 14-5 的归结反驳图显示了矛盾的一个导出，结果就证明了“约翰是快乐的”。

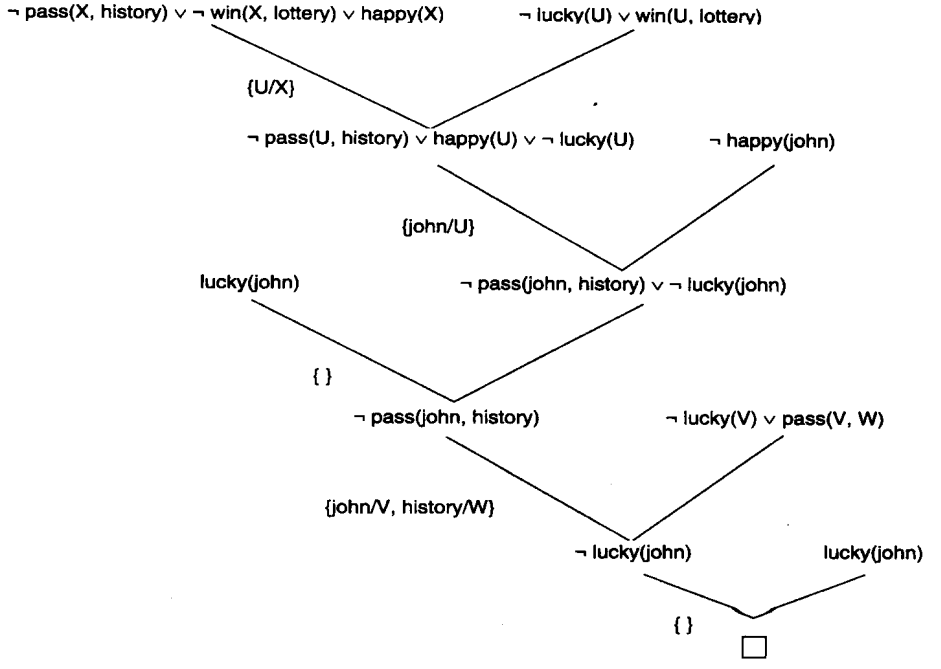


图 14-5　“快乐的学生”问题的一个归结反驳

作为本小节最后一个例子，我们介绍“令人兴奋的生活”问题；假设：

所有不贫穷而且聪明的人是快乐的。读书的人不愚蠢。约翰能读书而且不贫穷。快乐的人过着令人兴奋的生活。能找到过着令人兴奋的生活的人吗？

假定 $\forall X(\text{smart}(X) \equiv \neg\ \text{stupid}(X))$ 且 $\forall Y(\text{wealthy}(Y) \equiv \neg\ \text{poor}(Y))$，可得：

$\forall X\ (\neg\ \text{poor}\ (X) \wedge \text{smart}\ (X) \rightarrow \text{happy}\ (X))$

$\forall Y\ (\text{read}\ (Y) \rightarrow \text{smart}\ (Y))$

$\text{read}\ (\text{john}) \wedge \neg\ \text{poor}\ (\text{john})$

$\forall Z\ (\text{happy}\ (Z) \rightarrow \text{exciting}\ (Z))$

结论的否定是：

$\neg\ \exists W(\text{exciting}(W))$

“令人兴奋的生活”问题的谓词演算表达式可以转化为下面的子句：

$\text{poor}\ (X) \vee \neg\ \text{smart}\ (X) \vee \text{happy}\ (X)$

$\neg\ \text{read}\ (Y) \vee \text{smart}\ (Y)$

$\text{read}\ (\text{john})$

$\neg\ \text{poor}\ (\text{john})$

$\neg\ \text{happy}\ (Z) \vee \text{exciting}\ (Z)$

$\neg\ \text{exciting}\ (W)$

这个例子的归结反驳见图 14-6。

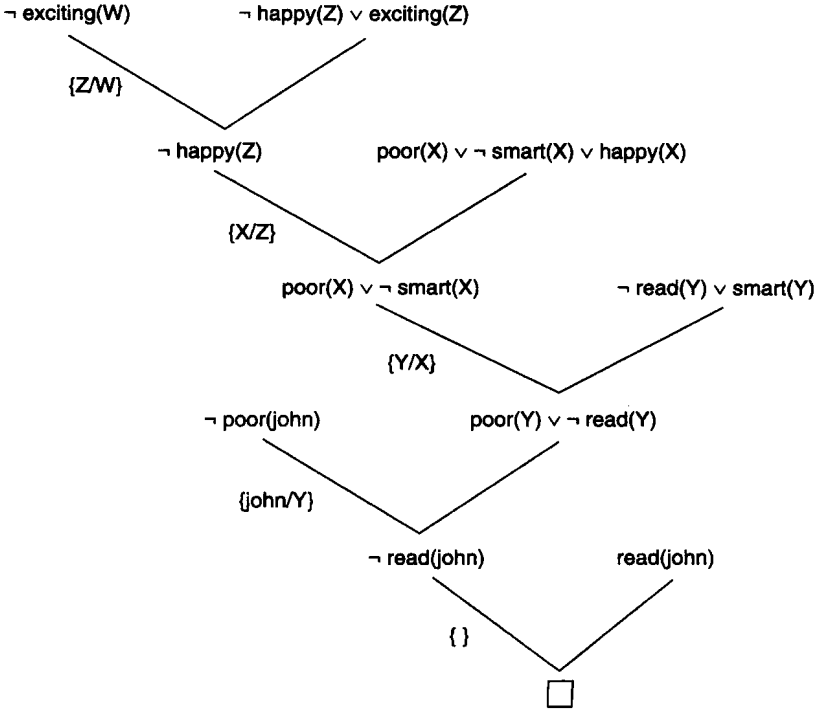


图 14-6　“令人兴奋的生活”问题的归结证明

14.2.4　归结策略和简化技术

图 14-6 问题的搜索空间中的另一棵证明树在图 14-7 中显示。这些证明有一些共同点；例如，都用了 5 步。同时，两个证明中合一代换的关联应用都发现 john 是过着“令人兴奋的生活”的人的一个实例。

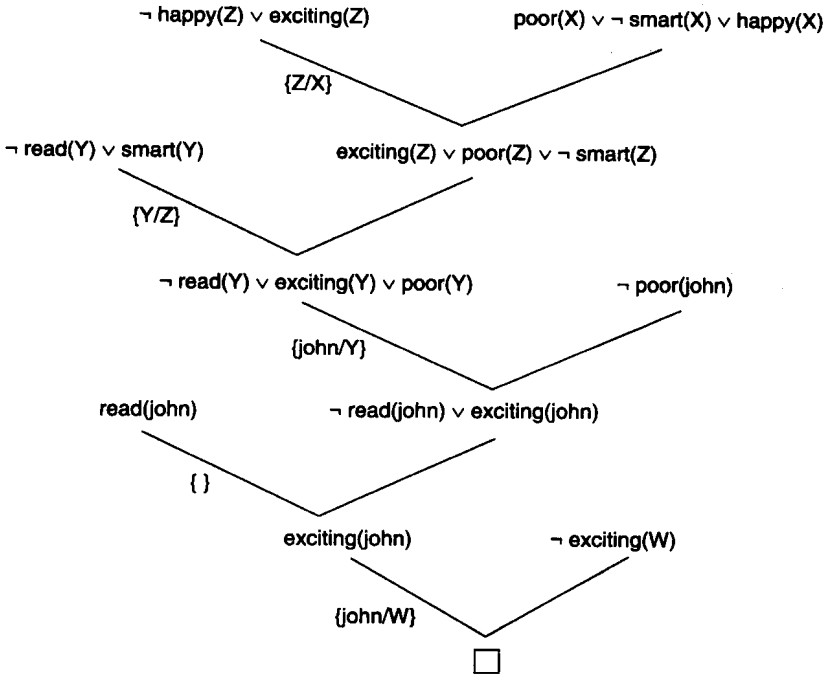


图 14-7　图 14-6 中例子的另一个归结反驳

然而，这两个相似点不一定必然出现。归结证明系统定义时（见 14.2.3 节），没有给出子句组合的顺序。这是一个关键问题：当子句空间中有 N 条子句时，就有 N^2 种方法组合它们，或者检查它们是否能够组合在第一层上！子句的结果集合也很大；即使只有 20% 的子句生成新的子句，下一轮中可能解的组合也会大大超过第一层。在问题较大的情况中，这种爆炸性增长会很快难以控制。

因此，启发式搜索在归结证明过程中非常重要，它也是问题求解弱方法。正如第 4 章中讨论过的启发式，还没有研究出能够确定一种能够解决任何特殊问题的最好的策略。无论如何，一些通用策略还是能够对付组合爆炸问题的。

在讨论策略之前，需要先做几点说明。首先，基于第 2 章中表达式不可满足性的定义，当不存在解释确定一个子句集是可满足的时，该子句集是不可满足的。其次，给定一个不可满足的子句集，如果单独使用一个推理规则可以确定其不可满足性，那么这条推理规则是反驳完备的。使用因式分解的归结具有此属性（Chang and Lee 1973）。最后，当在一条反驳完备的推理规则上使用策略时，如果不论子句集是否是不可满足的都能够保证找到一个反驳，那么这个策略是完备的。宽度优先是完备策略的一个例子。

宽度优先策略　上面提到的子句对比穷举的复杂性分析是基于宽度优先搜索的。第一轮子句空间中的每个子句与子句空间中的每个子句进行归结对比。搜索空间的第二层子句通过归结第一层中的子句和所有最初的子句生成。第 n 层的子句通过归结所有 $n-1$ 层的子句和最初的子

句集中的元素以及之前生成的所有子句生成。

宽度优先策略在处理大型问题时会很快难以控制。然而它确实具有一个很好的特性。同所有宽度优先搜索一样，它保证能发现最短的解路径，因为在进入下一层之前要为每一层生成所有的搜索状态。它也是一个完备的策略，因为如果存在矛盾并且能进行足够长的搜索，它保证能找到一个反驳。这样，当问题很小时，例如我们前面介绍的例子，宽度优先是一个好的策略。图14-8将宽度优先策略应用到了“令人兴奋的生活”问题。

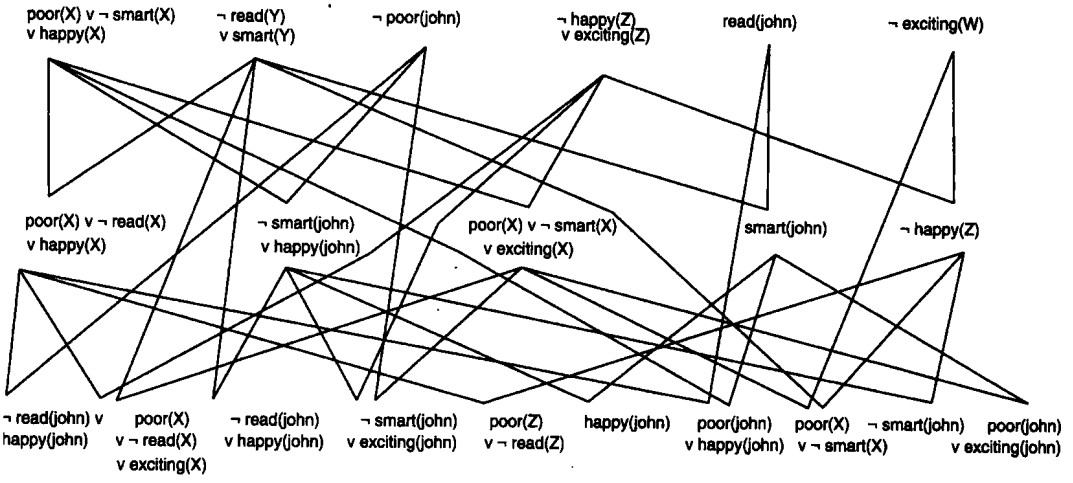


图14-8 宽度优先搜索生成的“令人兴奋的生活”问题的完整状态空间（到两层）

成组支持策略 对于大子句空间来说，成组支持策略（Wos and Robinson 1968）是极好的策略。对于输入的子句集S，我们可以定义一个S的子集T，称为成组支持。本策略要求每次归结的归结式之一有一个祖先在成组支持中。可以证明，如果S是不可满足的并且S-T是可满足的，那么成组支持策略是反驳完备的（Wos et al. 1984）。

如果原子句集是一致的，那么包含原查询的否定的任何成组支持都满足这些要求。这一策略基于对如下事实的洞察：要证明的目标的否定是导致子句空间中矛盾形成的原因。成组支持强制归结的子句之中至少有一个是目标的否定或者是由其归结生成的子句。

图14-6是将成组支持策略应用到“令人兴奋的生活”问题的一个例子。因为只要存在反驳，那么就存在支持反驳集，所以成组支持能够构成完备策略的基础。一个解决方法是对所有可能的支持反驳集进行宽度优先搜索。当然，这比对所有子句的宽度优先搜索要高效得多。只需要保证目标的否定子句的所有归结式及其后代都能被检查到。

单个优先策略 观察迄今为止所有的归结例子，矛盾的导出都是用没有文字的子句表示。这样，每次归结生成一个比原子句包含更少文字的结果子句，距离生成没有文字的子句更近了。特别是，归结一个文字的子句，称为个体子句，将保证归结式比最大的父子句小。单个优先策略是只要存在个体子句就使用个体子句归结。图14-9将单个优先策略应用到“令人兴奋的生活”问题。单个优先策略与成组支持策略一起使用，得到了更加有效的完备策略。

个体归结是与单个优先相关的一个策略，需要归结式之一是个体子句。这是一个比单个优先策略更强的要求。我们可以使用同一个例子说明线性输入形式不完备性，进而说明个体归结是不完备的。

线性输入形式策略 线性输入形式策略是对目标的否定和原始公理的直接使用：给出目标的否定，用公理之一进行归结，得到新的子句。再将结果和公理之一进行归结得到另一个新子

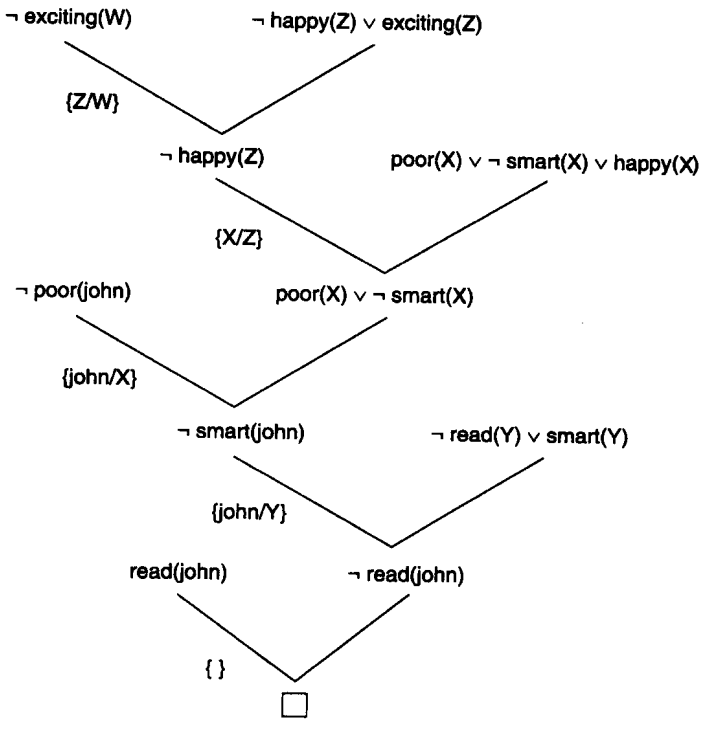


图 14-9　将单个优先策略应用到“令人兴奋的生活”问题

句，再和一个公理进行归结。过程一直进行，直到空子句产生。

每一步中，我们都对最近生成的子句和来源于问题原始描述的一条公理进行归结。不使用前面导出的子句，也不归结两个公理。线性输入形式不是一个完备的策略，通过将其应用到下面的四条子句（很明显是不满足的）可以看出，不管将哪条子句看作是目标的否定，线性输入策略都无法得出矛盾：

¬ a∨¬ b
a∨¬ b
¬ a∨b
a∨b

其他策略和简化技术　我们并不是要介绍所有使用归结推理的定理证明策略或是最复杂的技术。这些内容可以在其他一些著作中查到，如 Wos 等人（1984）和 Wos（1988）的著作。我们的目的是介绍本研究领域中最基本的工具，说明如何在问题求解中使用这些工具。但是归结证明过程是另外一种属于问题求解弱方法的技术。

从这种理解来看，归结可以作为谓词演算的推理引擎，但是需要大量分析和精确应用一些策略才能成功。在规模足够大的问题中，使用归结法随机归结表达式就像随机敲打键盘而希望得到一篇好文章一样毫无希望。因为组合的数量太多了！

本章中使用的例子都很小，而且所有的子句对于解答都是必需的。这些条件在有实际意义的问题中是很少成立的。用于对付这种组合复杂性，我们给出了几个简单的策略，并且还将指出在设计基于归结的问题求解系统时需要考虑的一些更重要的事情。我们在后面（见 14.3 节）还将说明一个具有组合搜索策略的归结反驳系统如何为逻辑程序设计提供“语义”，特别是为 Pro-

log 解释器的设计。

策略的组合对于控制搜索非常有效—如成组支持策略和单个优先策略的组合使用。搜索启发式也可以嵌入到规则的设计中（通过创建一个从左到右文字归结顺序）。这种顺序可以最有效地削减搜索空间。这种策略的隐含使用在 Prolog 程序设计中非常重要（见 14.3 节）。

结论的一般性可以作为设计归结策略的标准。一方面，保持中间解答尽可能通用很重要，因为这样使它们可以在归结中更自由地使用。因此，任何需要通过变量绑定进行特化的归结，如{john/X}，都应该尽可能往后推迟。另一方面，解答需要特殊的变量绑定，例如在分析 John 是否感染了葡萄球菌时，代换{john/Person}和{staph/Infection}可以限制搜索空间，增加找到解答的可能性和速度。

选择策略时一个重要的问题是完备性的概念。在某些应用中，知道会找到解答（如果存在）可能是非常重要的。这一点只能通过完备的策略来保证。

我们还可以通过加速匹配过程来提高效率。我们可以按包含的文字和文字的肯定或否定为每个子句建立索引，通过这种方法可以消除不可能生成新归结式的子句之间的不必要的（高成本的）合一，这使得我们可以直接找到潜在的归结式。同时，我们应该在一些特定的子句生成后就马上将其消除。首先，重复的子句不需要考虑；其次，永远不可能为假的子句；它们在归结尝试中毫无用处。

另一种不给出新信息的子句类型是能够被包含的子句，即在子句空间中已经有一个更一般的实例。例如，如果子句空间中已经包含 $\forall X(p(X))$，又推导出了 p(john)，那么 p(john) 可以扔掉而没有任何损失。实际上，由于子句空间中的子句更少了，所以节省了空间和时间。类似地，p(x) 归入子句 $p(x) \vee q(x)$。当两者都在子句空间中时，欠缺一般性的信息不会为更一般的信息增加任何东西。

最后，过程附加评价或者不用进一步搜索就能处理所有能产生新信息的子句。使用代数方法在公理或子句之间进行比较，或者“运行”其他确定性的程序用于向问题求解中添加具体的信息，或者使用任何方式约束求解过程。例如，当获得了足够的信息时，可以对一个程序计算变量进行绑定。变量绑定接着就限制了可能的解并修剪了搜索空间。

下面我们说明如何从归结反驳过程中抽取解答。

14.2.5 从归结反驳中抽取解答

假设为真的实例就是已发现反驳的代换。因此，保留归结反驳中合一代换的信息就可以给出正确答案的信息。在本节中，我们给出三个例子，并介绍一个从归结反驳中抽取解答的登记法。

解答的记录方法很简单：保留要证明的结论，并将该结论引入到归结过程中所做的每个合一中。原来的结论就成了反驳中所有合一的“记账员”。在归结反驳的计算搜索中，当反驳搜索中同时存在多个可能的选择时，这样做需要额外的指针，可能还需要一个类似回溯的控制机制来生成可选的解路径，但是要小心，添加的信息可能会被保留。

来看这个过程的一些例子。在图 14-6 中，找到了对存在一个人过着令人兴奋的生活的一个证明，应用图 14-10 中的合一。如果保留原目标，并将反驳的所有代换都应用到该子句，就找到

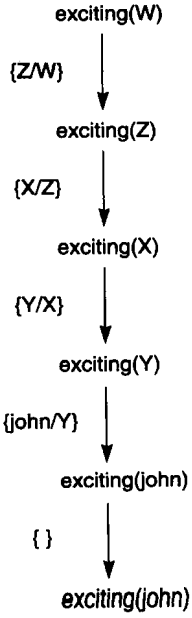


图 14-10 将图 14-6 的合一代换应用到原查询

了“哪个人过着令人兴奋的生活”的答案。

图 14-10 说明了归结反驳不仅能证明“没有人过着令人兴奋的生活”是错的，而且通过演示的过程能够得出快乐的人是 John。这是一个一般的结果，生成反驳的合一与生成原查询为真的实例的合一相同。

第二个例子是一个简单的故事：

主人约翰走到哪里，他的狗菲多就跟到哪里。约翰在图书馆。菲多在哪里？

首先将故事表示为谓词演算表达式，然后化简为子句形式。谓词形式：

$at(john, X) \rightarrow at(fido, X)$

$at(john, library)$

子句形式：

$\neg\ at(john,\ Y) \vee at(fido,\ Y)$

$at(john,\ library)$

结论的否定：

$\neg\ at(fido,\ Z)$，Fido 不在任何地方！

图 14-11 给出了解答抽取过程。对合一的字面上的跟踪就是原问题（Fido 在哪里？）：

$at(fido,\ Z)$

再一次，发现矛盾的合一告诉我们如何使得原查询为真：Fido 在图书馆。

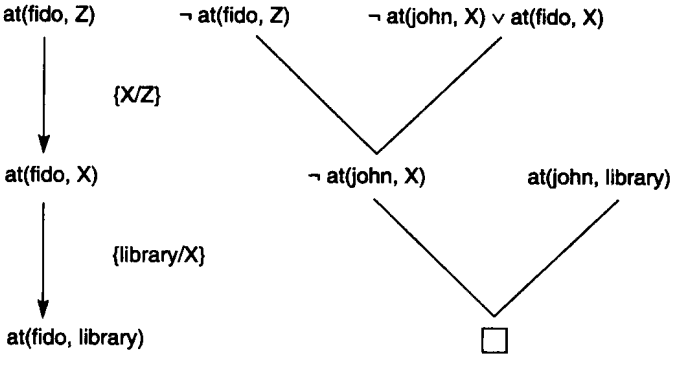


图 14-11　“寻找菲多”问题的解答抽取过程

最后一个例子说明斯柯伦化过程如何能够给出可以抽取出解答的实例。看下面的情景：

每个人都有父（母）亲。父（母）亲的父（母）亲是祖父（母）。给定一个人 John，证明 John 有祖父（母）。

下面的句子表示上面情景中的事实和关系。首先，每个人都有父（母）亲：

$(\forall X)\ (\exists Y)\ p(X,\ Y)$

父（母）亲的父（母）亲是祖父（母）。

$(\forall X)(\forall Y)\ (\forall Z)\ p(X,\ Y) \wedge p(Y,\ Z) \rightarrow gp(X,\ Z)$

目标是找到一个 W 使得 $gp(john, W)$ 或者 $\exists (W)(gp(john, W))$。目标的否定是 $\neg\ \exists (W)(gp$

(john, W))或：

¬ gp(john, W)

在将上面的谓词转化为子句形式来进行归结反驳的过程中，第一个谓词（每个人都有父（母）亲）中的存在量词需要一个斯柯伦函数。这个斯柯伦函数是一个很明显的函数：对给定的 X 找到 X 的父（母）亲。将其称为 pa(X)，表示“找到 X 的父（母）亲”。对于 John 来说，就是他的爸爸或者妈妈。本问题的谓词描述的子句形式是：

p(X, pa(X))
¬ p(W, Y) ∨ ¬ p(Y, Z) ∨ gp(W, Z)
¬ gp(john, V)

归结反驳和解答抽取过程表示在图 14-12 中。注意解答中的合一代换是

gp(john, pa(pa(john)))

对问题 John 是否有祖父（母）的解答是“发现 John 的父（母）亲的父（母）亲”。斯柯伦函数使得我们能够计算出这一结果。

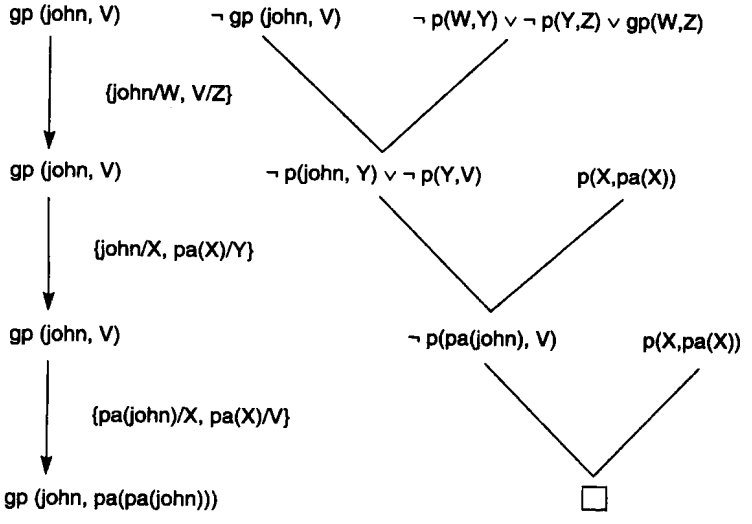


图 14-12　作为解答抽取过程一部分的斯柯伦化

刚刚描述的解答抽取的一般过程可以在所有归结反驳中应用，不管是使用类似图 14-10 和图 14-11 中的一般合一，还是像图 14-12 中一样估计斯柯伦函数。刚刚描述的解答抽取过程都能够得出一个解答。方法非常简单：发现矛盾的实例（合一）就是结论（原查询）否定的反为真的实例。虽然本小节没有在每个例子中证明这个结论是如何成立的，但是用几个例子说明了解答抽取过程是如何工作的；更深入的讨论可以查阅一些著作（Nilsson 1980，Wos et al. 1984）。

14.3　Prolog 和自动推理

14.3.1　概述

只有理解一种计算机语言的实现以后，我们才能正确掌握其使用，控制其副作用，对其结果有信心。本节我们描述 Prolog 的语义，并将其与前面章节介绍的自动推理中的问题联系起来。

14.2 节中归结证明过程存在的严重问题是，它需要一个完全同类的数据库来表示问题。当谓词演算描述化简或转换为子句形式时，重要的问题求解信息也遗漏了。忽略的信息虽然不是问题任何部分的真假，但却可能是对如何使用信息的控制提示或过程描述。例如，归结形式的目标否定子句可能具有以下形式：

$$a \vee \neg b \vee c \vee \neg d$$

其中 a、b、c 和 d 是文字。归结推理机制应用一个搜索策略来推导空子句。策略可以应用到所有文字，应用到哪一个文字依赖于所选的特定策略。指导归结定理证明使用的策略是弱启发式；因为它们与特定问题领域的知识结合并不深入。

例如，上面的归结例子中，目标的否定可能是下面谓词演算语句的一个变换：

$$a \leftarrow b \wedge \neg c \wedge d$$

可以将其理解为“看 a 是否为真，看 b 是否为真和 c 是否为假以及 d 是否为真”。规则亦指求解 a 的过程和对特定启发式信息的利用。实际上，子目标 b 可以提供证明整个断言为假的最简单方法，所以顺序“考察 b 然后看 c 是否为假然后检查 d”能够节省许多问题求解时间。隐含的启发式说明“先尝试用最简单的方法证明问题不成立，如果通过检查再继续前进，生成答案的其余（可能更难）部分”。人类专家设计过程和关系时，不仅保证它们的正确性，而且包含使用的关键信息。在大多数有意义的问题求解情形中，我们无法忽视这些启发式（Kowalski 1979b）。

下一节我们介绍 Horn 子句，并将其程序上的解释作为一个明确的策略使用，该策略保持有启发式信息。

14.3.2　逻辑程序设计和 Prolog

要理解 Prolog 的数学基础，首先要定义*逻辑程序设计*。有了这个定义后，我们将把显式搜索策略用于逻辑程序设计中以实现搜索策略的大致过程，这有时被称为 Prolog 的*程序性语义*。为了全面了解 Prolog，我们还讨论 not 和*封闭世界假设*（closed world assumption）的使用。

现在考虑用以进行归结反驳的子句数据库，如 14.2 节。如果我们限制集合中的子句最多只能有一个肯定的文字（0 个或者更多的否定文字），将得到具有某些有意义性质的子句空间。首先，用此子句集描述的问题对于归结反驳保持了不可满足性，或者说是反驳完备的，见 14.2 节。其次，将表示限制在所有子句的这种子类的一个重大优势是，反驳搜索策略将变得非常有效：线性输入形式、基于单个优先、从左到右和深度优先的目标约简。使用有根据的递归（最后终止的递归调用）和出现检查，这一策略在子句空间不满足的情况下能保证找到反驳（van Emden and Kowalski 1976）。一条 Horn 子句包含最多一个肯定的文字，也就是说具有以下形式

$$a \vee \neg b_1 \vee \neg b_2 \vee \cdots \vee \neg b_n$$

其中 a 和所有 b_i 都是肯定的文字。为了强调一个肯定文字在归结中的关键作用，一般将 Horn 子句写为以肯定文字为结论的蕴涵式：

$$a \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

在进一步讨论搜索策略之前，正式将子句的这个子集定义为 Horn *子句集*。Horn 子句集和*非确定性目标约减策略*一起组成*逻辑程序*。

定义（*逻辑程序*）　**逻辑程序**是具有以下形式的一阶谓词演算全称量化表达式集合：

$a \leftarrow b_1 \wedge b_2 \wedge b_3 \wedge \cdots \wedge b_n$

a 和 b_i 都是肯定的文字，有时称为原子目标。a 是子句**头**，b_i 的合取是子句**体**。

这些表达式是一阶谓词演算 Horn 子句。它有三种形式：第一，原来子句没有肯定的文字；第二，原来子句没有否定的文字；第三，原来子句有一个肯定的文字和一个或多个否定的文字。相应的例子分别见1、2和3：

1）$\leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$ 称为**无头子句或要测试的目标**：b_1 与 b_2 与 b_3 与…与 b_n。

2）$a_1 \leftarrow$

$a_2 \leftarrow$

⋮

$a_n \leftarrow$

称为**事实**。

3）$a \leftarrow b_1 \wedge \cdots \wedge b_n$

称为**规则关系**。

Horn 子句演算只允许上面的形式；←左边只能有一个文字，而且这个文字必须是肯定的。←右边的所有文字也是肯定的。

将最多有一个肯定文字的子句化简为 Horn 形式需要三步。首先，选出子句中的肯定文字（如果存在），并将其移到最左边（运用∨的交换律）。这个肯定的文字就成为了 Horn 子句的**头**。然后，运用下面的规则将整个子句化为 Horn 形式：

$$a \vee \neg b_1 \vee \neg b_2 \vee \cdots \vee \neg b_n \equiv a \leftarrow \neg (\neg b_1 \vee \neg b_2 \vee \cdots \vee \neg b_n)$$

最后，使用德·摩根定律将表达式变为：

$$a \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

其中可以用∧的交换律来安排子目标 b_i 的顺序。

应该注意的是，不是任意子句空间中的子句都能化为 Horn 形式。有些子句，如 $p \vee q$，没有 Horn 形式。要生成 Horn 子句，原子句中至多只能有一个肯定的文字。如果这一标准不满足，那么就需要重新考虑原来对问题的谓词演算描述。Horn 形式表示法的好处是形成一个有效的反驳策略，这在稍后将进行介绍。

逻辑程序的计算算法通过非确定性的目标约简进行。在每个具有以下形式的目标的计算步骤：

$$\leftarrow a_1 \wedge a_2 \wedge \cdots \wedge a_n$$

解释器任意地选择某个 a_i，$1 \leqslant i \leqslant n$。然后非确定性地选择一个子句：

$$a^1 \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

使得 a^1 应用代换 ζ 后与 a_i 合一，并且应用这个子句来约简目标。新的目标变为：

$$\leftarrow (a_1 \wedge \cdots \wedge a_{i-1} \wedge b_1 \wedge b_2 \wedge \cdots \wedge b_n \wedge a_{i+1} \wedge \cdots \wedge a_n)\zeta$$

非确定性目标约简的过程一直进行到目标集为空才结束计算。

在约简子目标时，如果通过施加一个顺序来消除不确定性，那么就不会改变计算的结果。所有非确定性方法能找到的结果用有序穷举搜索也能找到。然而，为了减少不确定性的程度，我们可以定义一些策略，从搜索空间中修剪不必要的分支。因此，实际逻辑程序设计语言主要涉及的问题是为编程人员提供一套控制和减少不确定性（可能的话）的工具。这些工具使得编程人员能够改变目标的约简顺序和约简每个目标所使用的子句集。（和任何图搜索一样，必须要采取预防措施防止证明过程中出现死循环。）

归结反驳系统逻辑程序的抽象说明有清晰的语义，van Emden 和 Kowalski（1976）证明了逻辑程序为真的最小解释是程序的解释。像 Prolog 一样的实际程序设计语言所付出的代价是运行程序只能计算出相关解释的一个子集（Shapiro 1987）。

顺序 Prolog 是对逻辑程序设计模型解释器的一个近似，它可以在冯·诺依曼计算机上有效运行，这也是本书中到目前为止使用的解释器。顺序 Prolog 同时使用目标在子句中的顺序和子句在程序中的顺序来控制证明的搜索过程。当有许多目标要证明时，Prolog 一般是从左到右依次去证明。在搜索一个目标的合一子句时，Prolog 总是按照程序设计人员安排的顺序依次检查每条子句。每做一个选择，就在记录的合一处设置一个回溯指针，当原来对合一子句的选择失败时，可以继续选择其他子句（还是按照程序设计人员安排的顺序）。如果这种搜索在子句空间中的所有可能的子句都失败了，那么求解过程失败。使用 14.1.5 节介绍的剪枝方法可以有效地尝试深度优先回溯搜索，解释器实际上可以不用访问搜索空间中所有的子句组合（解释）。

更形式化地，给定一个目标：

$$\leftarrow a_1 \wedge a_2 \wedge a_3 \wedge \cdots \wedge a_n$$

和程序 P，Prolog 解释器顺序搜索 P 中第一条头部与 a_1 合一的子句。然后使用该子句对目标进行约简。如果：

$$a^1 \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

是具有合一 ξ 的约简子句，那么目标子句就变为：

$$\leftarrow (b_1 \wedge b_2 \wedge \cdots \wedge b_n \wedge a_2 \wedge a_3 \wedge \cdots \wedge a_n)\xi$$

然后，Prolog 解释器继续尝试用程序 P 中第一个与最左边的目标合一的子句进行约简，本例中是 b_1。假设在合一 ϕ 下：

$$b^1 \leftarrow c_1 \wedge c_2 \wedge \cdots \wedge c_p$$

那么目标变为：

$$\leftarrow (c_1 \wedge c_2 \wedge \cdots \wedge c_p \wedge b_2 \wedge \cdots \wedge b_n \wedge a_2 \wedge a_3 \wedge \cdots \wedge a_n)\xi\phi$$

注意目标列表可以看作是一个执行深度优先搜索的 stack，如果 Prolog 解释器找不到一个可以约简目标的合一，那么就回溯到最近的合一选择点，恢复选择点后的所有绑定，并选择能够（按照 P 中的顺序）合一的下一条子句。通过这种方式，Prolog 实现了其从左到右、深度优先的子句空间搜索。

如果目标被约简为空子句（□），那么约简为：

$$\leftarrow (\square)\xi\phi\cdots\omega$$

使用的合一组合（$\xi\phi\cdots\omega$）提供了目标子句为真的一个解释。

除了按程序中子句顺序回溯外，顺序 Prolog 还允许剪枝或“!”。如 14.1.5 节所述，剪枝本身可以作为一个目标放置在子句中。解释器遇到剪枝时提交了当前执行路径，特别是选择包含该剪枝的子句后所做的合一的子集。剪枝同时告诉解释器该子句是约简目标的惟一可选方法。当剪枝后子句中遇到失败时，整个子句失败。

过程上，剪枝使得保留剪枝之前约简子句及其成分的回溯指针变得没必要。因此，剪枝意味着模型中只有某些可能的解释被计算。

作为总结，我们将顺序 Prolog 与 14.2 节的归结反驳模型进行比较。

1）归结子句空间是逻辑程序设计中 Horn 子句表达式的超集。每个子句必须至多包含一个肯定文字才能化为 Horn 形式。

2）下面的结构可以将问题表示为 Horn 形式：

a）目标

$$\leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

是组成归结反驳要检测的目标的子句语句列表。每个 a_i 按顺序被取反、合一和约简，直到发现空语句（如果可能）。

b）事实

$$a_1 \leftarrow$$
$$a_2 \leftarrow$$
$$\vdots$$
$$a_n \leftarrow$$

是归结用的单个子句。

c）Horn 子句或公理

$$a \leftarrow b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

允许约简匹配的子目标。

3）使用单个优先、线性输入形式策略（总是先选择事实子句和使用反目标及其后续分解，见 14.2.4 节）并应用从左到右、深度优先（有回溯）顺序的归结子句选择，归结定理证明程序采用类似 Prolog 解释器的方式工作。因为该策略是归结完备的，所以其使用能够保证找到解答（前提是该部分解释集合没有被剪枝剪掉）。

4）最后，证明中合一的组合提供目标为真的解答（解释）。这与 14.2.5 节中的解答抽取过程完全相同。记录目标文字中合一的组合，生成每个解答解释。

Prolog 的一个重要问题在于 Horn 子句规则右手侧的反例文字的使用。这需要解释器能执行封闭世界假设（the closed world assumption）。在谓词演算中，证明 $\neg p(X)$ 就是证明 $p(X)$ 逻辑上不成立。即 $p(X)$ 在使得公理集为真的所有解释下都不成立。基于第 2 章的合一算法，Prolog 解释器提供了一个比 14.2 节的通用归结反驳更加严格的结果。也就是不是尝试所有的解释，而是检查那些数据库中显式的解释。现在，我们将这些约束公理化以便弄清当前 Prolog 环境中隐含的限制。

对于每个谓词 p，和属于 p 的每个变量 X，假设 $a_1, a_2, \cdots, a_n$ 构成了 X 的域。使用合一的 Prolog 解释器，要求：

1）名称惟一公理。对于域中的所有原子有 $a_i \neq a_j$，除非它们等价。这意味着具有不同名字的原子是不同的。

2）封闭世界公理。

$$p(X) \rightarrow p(a_1) \vee p(a_2) \vee \cdots \vee p(a_n)$$

这就是说，一个关系的惟一可能实例就是问题描述中的子句所包含的关系。

3）域封闭公理。

$$(X = a_1) \vee (X = a_2) \vee \cdots \vee (X = a_n)$$

这条公理保证了在问题描述中出现的原子构成所有的和仅有的原子。

这三条公理隐含于 Prolog 解释器的动作中。它们可以被看作是添加在构成问题描述的 Horn 子句集中，并作为对 Prolog 询问的解释集合的限制。

直观上，这意味着 Prolog 假设所有无法证明为真的问题都不成立。这可能会带来异常：如果目标的真值在当前数据库中实际上不清楚，那么 Prolog 会假设它为假。

Prolog 还隐含一些其他的限制，所有计算语言中似乎都有这些限制。其中除了否定问题外，最重要的是逻辑程序设计语义模型的表示违例。特别是，缺少存在检查（见 2.3 节；一条语句可以与本身的子集合一）和剪枝的使用。目前这一代的 Prolog 解释器需要从程序设计上去考虑。有些问题的出现是因为“目前没有已知有效的方法”来解决问题（存在检查情形）；其他问题的出现是因为试图优化带回溯的深度优先搜索（用剪枝显式控制搜索）。Prolog 的许多异常是由于试图在顺序计算机上实现纯逻辑程序设计的不确定性语义造成的，其中也包括剪枝带来的问题。

在 14.4 节，我们介绍自动推理的可选推理方案。

14.4　自动推理进一步的问题

我们将弱方法问题求解器描述为使用（a）统一表示媒介、（b）面向表示法语法特征的可靠推理规则并用、（c）对付穷举搜索组合问题的方法或策略指导规则的使用。我们将对弱方法求解过程的这些方面进一步解释，作为本章的总结。

14.4.1　弱方法求解的统一表示法

归结证明过程要求将所有公理转化为子句形式。这种统一的表示法使我们可以化简子句，并简化问题求解启发式的设计。这种方法最主要的缺点是许多有价值的启发式信息可能在这种统一编码中丢失。

if…then 格式的规则通常比其他语法上的变体传递更多使用取式假言推理和产生式系统搜索的信息，同时也提供了一种使用规则的有效方法。例如，假设我们要表示反绎推理，见 8.0 节，如果引擎不旋转并且灯不亮，那么可能是电池坏了。这条规则给出了如何检查电池的建议。

规则的析取形式使得关于应该如何应用规则的启发式信息变得模糊了。如果将规则表示为谓词演算¬ turns_over ∧ ¬ lights→battery，本规则的子句形式是：turns_over ∨ lights ∨ battery。这一子句有许多等价形式，每一种形式都表示一个不同的含义。

(¬ turns_over ∧ ¬ lights) → battery
(¬ turns_over → (battery ∨ lights))
(¬ battery ∧ ¬ lights) → turns_over
(¬ battery → (turns_over ∨ lights))

等等。

为了在自动推理过程中保留启发式信息，包括 Nilsson（1980）和 Bundy（1988）在内的几个研究人员提出了依据人类专家可以设计规则关系的方式形成规则来进行编码启发式的推理方法。我们在 3.3 节的与或图推理和 14.3 节的自动推理中已经建议过这种方法。基于规则的专家系统还允许程序设计人员通过规则结构控制搜索。我们通过下面的两个例子进一步发展了这一想法，第一个是数据驱动，第二个是目标驱动。这两种方法都保留了规则的含义，并使用这一信息指导与或图中的搜索。

看下面一些用于数据驱动推理的事实、规则（公理）和目标：

事实：

(a∨(b∧c))

规则（或公理）：

(a→(d∧e))
(b→f)
(c→(g∨h))

目标：

←e∨f

e∨f 的证明过程见图 14-13 中的与或图。注意，数据驱动搜索空间中的∨关系上与连结和∧关系上或连结的使用。如果给定或者 a 为真，或者 b∧c 为真，那么必须对两个析取项都进行推理，以保证论证是真值保持的；因此这两条路径是结合的。另一方面，当与为真，就可以只检查两个合取项之一。规则匹配选择任意中间语句，例如 c，用一条前提与其匹配的规则的结论代替，例如(g∨h)。图 14-13 中语句 e 和 f 的发现说明目标(e∨f)建立了。

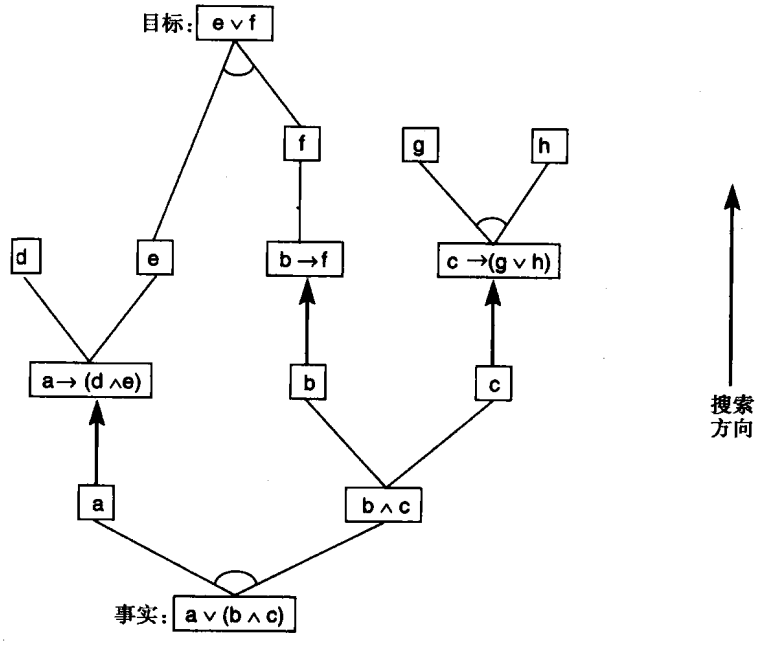


图 14-13　命题演算中使用与/或图的数据驱动推理

以一种类似的方式，我们可以使用目标驱动推理与或图上的规则匹配。当一个目标描述包括一个∨时，如图 14-14 的例子，每个替代方案都可以被独立地开发以建立目标。如果目标是一个合取，则当然，两个合取项都必须被建立。

目标：

(a∨(b∧c))

规则（或公理）：

(f∧d)→a
(e→(b∧c))
(g→d)

事实：

f∧g

虽然这些例子来自于命题演算，但是使用谓词演算事实和规则时产生的是相同的搜索。合一使得文字可以将推理规则应用于搜索空间的不同分支。当然，在搜索空间的不同与分支中，合一必须是一致的（即能统一的）。

本小节给出了问题求解弱方法的一个有助于在表示媒介中保存启发式信息的解决方法，该方法在本质上是专家系统的推理引擎允许程序员在规则中定义控制和启发式信息的方法。专家系统依赖于规则形式来控制搜索，而不是完全依赖一般的问题求解弱方法，如规则顺序或规则中前提的顺序。这种方法损失了将统一的证明过程（如归结）应用到整个规则集合的能力。如图 14-13 和图 14-14 的例子中，还可以使用取式假言推理。使用深度优先、宽度优先或者最佳优先搜索的产生式系统控制为实现规则系统提供了一种弱方法推理体系结构（见第 4、6 和 8 章中的例子）。

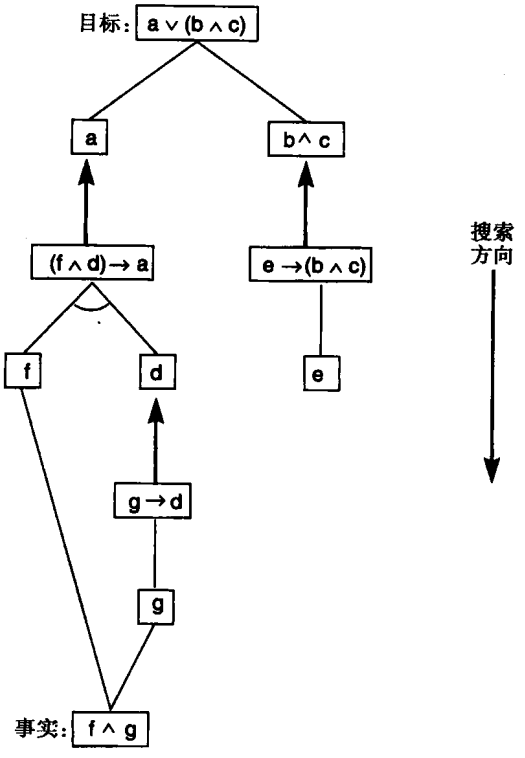


图 14-14　命题演算中使用与或图的目标驱动推理

14.4.2　可选推理规则

归结是目前为止我们所介绍的最通用的可靠推理规则。为了使得归结更有效，还创造了几个更复杂的推理规则。这里我们主要介绍其中的两个：*超归结*和*参数调制*。

前面介绍过，归结实际上是一个特殊的变体，刚好两个子句对消的归结被称为*二元归结*。一个超归结的成功应用可以取代一系列二元归结并生成一个子句。在一个步骤中，超归结消解一个具有否定文字的子句（称为*核*）以及一些全部是肯定文字的子句（称为*卫星*）。这些卫星子句必须具有一个肯定文字能够匹配核的否定文字。同时，核的每个否定文字都必须有一个卫星。因此，超归结应用的结果是一个全部是肯定文字的子句。

超归结的一个优点是每次超归结推理都只产生一条全部是肯定文字的子句。因为这一过程没有中间结果生成，所以子句空间保持在较小的规模。推理步骤中所有子句上的合一必须是一致的。

作为一个超归结的例子，看下面的子句集：

¬ married(X, Y) ∨ ¬ mother(X, Z) ∨ father(Y, Z)
married(kate, george) ∨ likes(george, kate)
mother(kate, sarah)

使用超归结可以在一步中得出结论：

father(george, sarah) ∨ likes(george, kate)

例子中的第一条子句是核；后面的两条是卫星。卫星都是肯定文字，并且有一个可以与核中的否

定文字相匹配。注意核为什么恰好是下面含义的子句形式：

married(X, Y) ∧ mother(X, Z)→father(Y, Z)

这条规则的结论是最终结果的一部分。注意没有中间结果产生，例如：

¬ mother(kate, Z) ∨ father(george, Z) ∨ likes(george, kate)

但在相同子句空间上的任何二元归结证明中都有上述子句产生。

单独使用超归结时，超归结是可靠的和完备的。当超归结与其他策略（如成组支持策略）组合使用时，可能会危及完备性（Wos et al. 1984）。虽然在大多数使用超归结的环境中，子句通常按照名称和每个文字的肯定或否定制定属性索引，然而还是需要特殊的搜索策略来组织卫星以及核子句，这样可以为超归结推理有效地准备核和卫星子句。

在定理证明机制的设计中，一个重要的难题是等价性的控制。特别复杂的是应用领域中，如数学，许多事实和关系都有多种表示法，例如应用结合律和交换律所获得的表达式。用一个非常简单的例子就可以使你明确这一点，考虑一下能够表示算术表达式3+(4+5)的多种方式，包括3+((4+0)+5)。这是一个复杂的问题，因为表达式在自动数学问题求解中需要被代换、合一以及检查与其他表达式的等价性。

*解调*是改写表达式使其自动呈现选定的规范形式的过程。用来生成这种规范形式的个体子句是解调子。解调子规定了不同表达式的等价性，允许用表达式的规范形式代替原表达式。恰当地使用解调子就可以让所有新产生的信息在加入子句空间之前都化简为特定的形式。例如，我们有如下解调子：

equal(father(father(X)), grandfather(X))

及新子句：

age(father(father(sarah)), 86)

在将这条新子句加入子句空间之前，先应用解调子，然后加入下面的子句：

age(grandfather(sarah), 86)

这里的等价问题是命名问题。我们希望将一个人分类为father(father(X))，还是grandfather(X)？类似地，可以为所有家庭关系挑选出规范的名称：brother(father(Y))是uncle(Y)，等等。一旦选定了存储信息的标准名称，接着就可以设计解调子，然后将所有新信息化简为这种形式，如equal子句。注意解调子都是个体子句。

*参数调制*是项层次上等价代换的泛化。例如，给定表达式：

older(mother(Y), Y)

以及等价关系：

equal(mother(sarah), kate)

使用参数调制可以得到结论：

older(kate, sarah)

注意：对kate的{sarah/Y}和mother(sarah)的项层次匹配和置换。解调和参数调制之间的一个主要区别是后者在等式谓词和进行代换的谓词的参数中都允许非平凡变量置换。解调根据解调子进行置换。可能要使用多个解调子才能将一个表达式化为最终形式；参数调制在任何情况下

通常都只使用一次。

我们给出了这些强有力的推理机制的简单的例子。我们应该把它们看作是在归结子句空间中使用的更通用的技术。像我们看过的所有其他推理规则一样，它们与所选的表示法密切相关，并且必须使用正确的策略进行控制。

14.4.3　归结反驳支持下的问答机制

14.2.5 节中阐述了归结反驳过程与对原始问题（即欲证的定理）的回答产生过程之间的一种自然的契合。Stuart Shapiro 和其同事（Burhans 和 Shapiro 2005）证明了归结反驳该如何为问答机制提供有用的范例。

Burhans 和 Shapiro 发现反驳过程可以把归结反驳过程产生的子句划分为三种类型，每一种类型对应于回答不同的问题（即当下所证定理）。在他们的研究中，具有相关性的子句均被认为是可能采取的回答，相关性由问题（欲证的定理）和知识库（证明中的子句集）来决定，而且由否定目标的子句形式衍生而来的任何子句均被认为具有相关性。

反驳证明被分为三种回答类型：特殊、一般和假设。我们依据组成子句的文字所共有的变量来区别这三种类型。所得结果与上下文无关，即结果可以在任何基于支持集的反驳、实际问答机制中得到运用。

特殊回答有时也被称作*扩充*回答（Green 1969a，b），它与基本术语集相关，这些术语对应于反驳产生过程中替换变量值的常量。例如，问题“有人在家吗?”被转换为否定目标“没有人在家”。当发现“Amy 在家”时就会产生反驳。反驳过程产生的个体集合将提供这种特殊回答。

一般回答也被称作*内涵型*回答（Green 1969a，b），对应于回答子句的所有的非基本实例，其中回答子句与反驳相关。例如，对于问题“家里有人吗?”，任何基于变量的子句（假定这些子句蕴涵于子句空间或能由子句空间推理得出），例如“所有的孩子均在家中”，将组成一般回答的集合。

第三种是假设回答，也被称作*条件*或*资格*回答。这种类型的回答采用规则的形式，其先驱有基本的变量，其后继与否定的答案问题相匹配。例如，通过搜索确知 Allen 是否在家后，问题“家里有人吗?”可以与子句“如果 Allen 在家则 Amy 在家”相匹配。更多细节请参见 Burhans 和 Shiparo（2005）。

14.4.4　搜索策略及其使用

有时应用领域对推理规则和指导规则使用的启发式有特殊要求。我们已经看过了解调子在等价代换中的使用。Bledsoe 在他的*自然演绎系统*中为准备归结证明的定理确定了两个重要的策略。他将这两个策略称为分裂（split）和化简（reduce）（Bledsoe 1971）。

Bledsoe 设计的策略用于数学，特别是应用于集合论中。这些策略的作用是将一个定理分裂为几个部分，使其容易用传统方法证明，如归结。$A \wedge B$ 的证明等价于 A 的证明和 B 的证明。类似地，证明 $A \leftrightarrow B$ 就是要证明 $A \rightarrow B$ 和 $A \leftarrow B$。

启发式化简还试图将大的证明分解为几个组成部分。例如，$s \in A \cap B$ 的证明可以分解为 $s \in A$ 的证明和 $s \in B$ 的证明。另外一个例子是，要证明 $\neg (A \cup B)$ 的某些性质为真，可以通过证明 $\neg A$ 和 $\neg B$ 的某些性质为真。Bledsoe 希望通过将大的证明分解成小块来限制搜索空间，他的启发式中还包括有限制地使用等价代换。

正如本书通篇都提到的，适当地使用启发式更像是一门艺术，因为其要将应用领域同表示法和推理规则一起考虑。我们引用一些通用的箴言来结束本章，所有这些箴言在某些情况下都

可能不正确，但是如果小心使用，将非常有效。这些箴言总结了本领域中研究人员（Bledsoe 1971，Nilsson 1980，Wos et al. 1984，Wos 1988）的思想和我们自己在弱方法问题求解器方面的思考。我们中立的引用它们。

尽可能使用包含较少文字的子句。

应用通用推理之前先将任务分解为子任务。

恰当时尽量使用等式谓词。

用解调子生成规范形式。

用等价谓词进行推理时要使用参数调制。

使用保持“完备性”的策略。

对包含潜在矛盾的子句集使用成组支持策略。

在归结中优先使用个体，这样可以缩短结果子句的长度。

对新子句执行包含（subsumption）检查。

在子句和子句中的文字上使用排序机制来反映你的直觉和问题求解经验。

14.5 结语和参考文献

弱方法问题求解器要求选择一种表示方法、推理机制和搜索策略。这三个选择杂乱地交织在一起，无法孤立于其他两个做出决定。应用领域也会影响表示法、推理规则和策略的选择。在做出这些选择时应该认真考虑上一节结尾的“箴言”。

归结是不断约束可能的解释直到发现添加了否定目标的子句空间不一致的过程。本文并不深入讨论归结的可靠性或者归结反驳的完备性。对这些问题的证明是以 Herbrand 定理（Chang and Lee 1973）和子句集可能解释的概念为基础的。我们在此鼓励有兴趣的读者去阅读关于这些证明的参考文献。

许多其他参考资料也是很不错的：《Automated Theorem Proving: A Logical Basis》一书中提出了一种形式化方法（Loveland 1978）。《The Automation of Reasoning: Collected Papers, 1957 to 1970》（Siekmann and Wrightson 1983a, b）这套丛书中收集了本领域中的许多经典早期文章。Nilsson（1980）、Weyhrauch（1980）、Genesereth 与 Nilsson（1987）、Kowalski（1979b）、Lloyd（1984）、Wos 等（1984）和 Wos（1988）等都对自动推理中的重要观点做了有价值的总结。Robinson（1965）和 Bledsoe（1977）对该领域做出了基础性贡献。Boyer 和 Moore（1979）在定理证明方面做出了重要研究贡献。《Computers and Thought》（Feigenbaum 和 Feldman 1963）和《Human Problem Solving》（Newell 和 Simon 1972）上给出了 Newell 和 Simon 及其同事在卡内基技术研究所所做的早期定理证明工作。Burhans 和 Shapiro（2005）对基于归结反驳的问题回答机制做了描述和研究。

CADE（Conference on Automated DEduction）成为了发表自动推理中最新结果的主要论坛。模型检查、验证系统和可伸缩知识表示系统是目前的研究热点（McAllester 1999，Ganzinger et al. 1999，Veroff 和 Spinks 2006）。Wos 及其同事在 Argonne 国家实验室进行了重要的研究。Veroff（1997）为纪念 Wos 发表了一组关于自动推理的论文。爱丁堡大学的 Bundy 在自动推理方面所做的工作（Bundy 1983，1988）很重要。在得克萨斯奥斯汀大学进行的扩展 Boyer-Moore 定理证明程序方面的研究也非常重要，见《Computer-Aided Reasoning: ACL2 Case Studies》（Kaufmann et al. 2000）。

14.6 习题

1. 理解 2.4 节中基于逻辑的财务顾问程序，将问题的谓词描述化为子句形式，并用归结反驳回答一些询

问，如某个特定的投资者是否应该做出 investment(combination)。

2. 用归结法证明第 2 章练习 12 中 Wirth 的描述。
3. 用归结法回答例 3.3.4 中的询问。
4. 我们在第 5 章介绍了骑士周游问题的一个简化形式。将 path3 规则化为子句形式，并使用归结法回答一些询问，如 path3(3,6)。然后，用子句形式的递归路径调用回答询问。
5. 如何用归结法实现“产生式系统”搜索？
6. 如何用归结法进行数据驱动推理？用此方法描述练习 1 的搜索空间。在大规模问题空间中会出现什么问题？
7. 用归结法回答对 15.3 节“农夫、狼、山羊和卷心菜”问题的询问。
8. 用归结法解决下面来自 Wos 等人的（1984）难题。四个人：Roberta、Thelma、Steve 和 Pete 拥有八项不同的工作。每个人正好有两项。这些工作分别是厨师、保安、护士、接线员、警官、教师、演员和拳击手。护士是一个男性。厨师的丈夫是接线员。Roberta 不是拳击手。Pete 没有受过九年级后的教育。Roberta、厨师和警官一起出去打高尔夫球。谁从事哪项工作？说明添加性别偏见会如何改变问题？
9. 举出两个超归结的例子，其中核至少具有 4 个文字。
10. 为 sum 写一个解调子，使得子句 equal(ans, sum(5, sum(6, minus(6)))) 可以化简为 equal(ans, sum(5,0))。再写一个解调子，将结果化简为 equal(ans,5)。
11. 选择六个家庭关系的“规范集”。写出解调子，将关系的可选形式化简为标准集。例如，“母亲的兄弟”是“舅舅”。
12. 理解图 14-5 中的“快乐的学生”问题，并在其求解中应用 14.2.4 节中的三个反驳策略。
13. 将下面的谓词演算表达式化为子句形式：

    $\forall(X)\ (p(X)\rightarrow\{\forall(Y)[p(Y)\rightarrow p(f(X,Y))]\wedge\neg A(Y)[q(X,Y)\rightarrow p(Y)]\})$

14. 为下面的数据驱动谓词演算推理画出与或图。

    事实：$\neg d(f)\vee[b(f)\wedge c(f)]$

    规则：$\neg d(X)\rightarrow\neg a(X)$，$b(Y)\rightarrow e(Y)$，$g(W)\leftarrow c(W)$

    证明：$\neg a(Z)\vee e(Z)$

15. 证明线性输入形式策略不是反驳完备的。
16. 为下面的问题画出与或图，说明为什么不可能得出下面的目标：

    $r(Z)\vee s(Z)$？

    事实：$p(X)\vee q(X)$

    规则：$p(a)\rightarrow r(a)$，$q(b)\rightarrow s(b)$

17. 用因式分解和归结的方法为下面的子句生成一个反驳：$p(X)\vee p(f(Y))$和$\neg p(W)\vee\neg p(f(Z))$。试着不用因式分解来生成一个反驳。
18. 导出图 14-1 定理的一个归结证明。
19. 逻辑程序设计的一个可选语义模型是*平滑并行 Prolog*。试比较 14.3 节中的 Prolog 和平滑并行 Prolog（Shapiro 1987）。

第 15 章　自然语言理解

必须认识到在术语解释已知的情形下关注“一个句子的概率”是毫无意义的……

——诺姆·乔姆斯基，(1965)

我理解你话中的狂怒，但不理解你的话。

——威廉·莎士比亚，《奥赛罗》

他们刚从一场文字的盛宴上，偷了些吃剩的肉皮鱼骨回来。

——威廉·莎士比亚，《爱的徒劳》

我希望有人能告诉我“Ditty wah ditty”的意思。

——Arthur Blake

15.0　自然语言理解问题

用自然语言进行交流，不管是以文字的形式还是以交谈的形式，都严重依赖于参与者的语言技能、感兴趣的领域知识和领域内的谈话预期。理解语言不仅仅是对文字的翻译：还需要推测说话人的目的、知识和假设，以及交谈的上下文语境。实现一个自然语言理解程序需要表示出所涉及领域中的知识和期望，并能进行有效的推理。还必须考虑一些重要的问题，如非单调、信念改变、比喻、规划、学习和人类交互的实际复杂性。然而，这些问题正是人工智能本身的核心问题。

例如，考虑下面从莎士比亚的《十四行词第 18 首》中选出的几行：

我该把你比拟做夏日吗?
你比夏日更可爱，更温婉：
狂风会把五月的娇蕊吹落，
夏天出租的期限又太短暂：

我们无法单纯从字面意思上来理解这几行。我们必须提出以下问题：

1）莎士比亚写作的意图是什么？我们必须了解许多关于人类的爱及其社会习俗才能开始理解这些句子。或者他仅仅是试图拿一些东西给出版商换取稿费？

2）莎士比亚为什么将他心爱的人比作夏日？他是说她有 24 小时那么长并且能导致晒黑吗？又或是她使他感觉到夏天的温暖和美丽？

3）这一段要得出什么样的推论？莎士比亚的原意并没有明确地显示在文字中；必须使用比喻、类比和背景知识才能推断出来。例如，我们该如何想到将狂风和夏天的短暂比作对人类生命和爱情之短暂的悲叹。

4）我们如何理解比喻？文字当中并没有涉及明确的诸如桌子上的方块这样的物体：上述诗歌的意思核心在于选用夏日的特征来比作爱人，有的特征被选用，有的特征被忽略，而且最重要的是，为什么这些特征被选用，而另外一些特征被忽略？

5）基于计算机的文字语音系统必须要知道关于抑扬格的五步格诗的知识吗？计算机如何能总结出这首诗的主要内容，或者从诗集中智能地找到它？

我们不能将莎士比亚的句子按照单词在字典中的意思串连在一起，然后将结果称为对语言的理解。我们必须运用一个复杂的过程来理解词的意思，分析句子，建立对语义意思的表示，并根据我们在问题领域中的知识来解释其中的意思。

第二个例子是关于某计算机科学系在网上招聘的广告内容，下面给出了一部分内容。

新墨西哥大学计算机科学系……有两个预备终身教职空缺。我们希望招聘对以下领域感兴趣的人员：

软件，包括分析、设计和开发工具……

系统，包括体系结构、编译器、网络……

……

申请者必须具有以下专业的博士学位……

我系在自适应计算、人工智能等领域具有国际公认的研究计划……并且与圣达菲研究所以及几个国家实验室开展了深入的研究合作……

理解这条招聘广告时会出现几个问题：

1）只明确表明“预备终身教职”，读者如何知道这条广告是招聘教师？任期多长？

2）在大学环境中工作需要掌握什么软件和软件工具的知识？Cobol、Prolog 还是 UML？这些都没有明确提到。一个人需要有许多关于大学授课和研究的知识才能理解这些要求。

3）为什么要在大学招聘广告中提到国际公认的研究计划和与著名研究所的合作？

4）计算机如何概括广告的主要意思？计算机必须掌握什么知识才能为一个正在找工作的求职博士从万维网上智能地检索到本广告？

自然语言理解中（至少）有三个主要问题。第一，需要具备大量的人类知识。语言动作描述的是复杂世界中的关系。关于这些关系的知识必须是理解系统的一部分。第二，语言是基于模式的：音素构成单词，单词组成短语和句子。音素、单词和句子的顺序不是随机的。没有对这些元素的一种规范性的使用，就不可能达成交流。最后，语言动作是主体（agent）的产物，或者是人或者是计算机。主体处在个体层面和社会层面的复杂环境中。语言动作都是有其目的性的。

本章对自然语言理解以及用来解决该问题的计算技术中的问题进行介绍。虽然本章主要集中在文本理解上，但是语音理解与合成系统也必须解决这些问题，当然还有另外一些与特定上下文中词义识别和消歧有关的问题。

因为理解无约束的语言需要一定的知识，所以早期的人工智能程序比较关注*微观世界*（microworld）并取得了一些进步，即只运用很少的领域知识。Terry Winograd 的 SHRDLU（Winograd 1972）是较早使用这种方法的程序之一，该程序能够交谈一些关于积木世界的问题，其中包含不同形状和颜色的积木以及一只能移动积木的手，如图 15-1，另见 8.4 节。

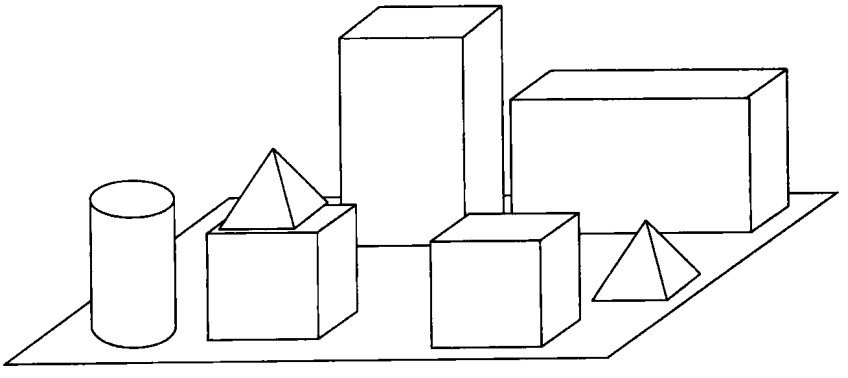


图 15-1　积木世界

［改编自 Winograd（1972）］

SHRDLU 能够响应英语询问，如“What is sitting on the red block?”“What shape is the blue

block on the table?" 或 "Place the green pyramid on the red brick." 它能够处理代词，如 "Is there a red block? Pick it up." 它甚至能理解省略，如 "What color is the block on the blue brick? Shape?" 因为积木世界非常简单，所以能够为系统提供世界的完整知识。因为积木世界不包含常识推理中更难的问题，如时间理解、因果关系、概率或信念，所以知识的表示技术比较简单直接。尽管领域有限，SHRDLU 还是为语法和语义的集成提供了一个模型，并且证明了具有足够所交谈领域知识的程序能够用自然语言进行有意义的交流。

对于上面曾提到的大量知识成分组成的语言的理解，我们需要为语言表达本身的模式和期望建模。马尔可夫链为我们捕获这些规律提供了一个强有力的工具。例如，在语言使用中，冠词和形容词通常在名词之前而不是之后，而且特定的名词和动词往往一起出现。马尔可夫模型还能够捕获语言模式和所描述的世界之间的关系。

在 15.1 节，介绍一种自然语言理解的符号方法。15.2 节介绍语法分析法；15.3 节利用扩充转移网络解析将语法和语义结合起来。15.4 节介绍捕获语言表示规律性的随机方法。最后，在 15.5 节，考察几个自然语言理解程序非常有用的应用：问题回答、访问数据库信息、万维网查询和文本摘要。

15.1　解构语言：分析

语言是一个复杂的现象，包括各种处理，如声音或印刷字母的识别、语法解析、高层语义推论，甚至通过节奏和音调传达的情感内容。为了管理这个复杂性，语言学家定义了自然语言分析的不同层次：

1）*韵律学*（prosody）处理语言的节奏和语调。这一层次的分析很难形式化，经常被省略；然而，其重要性在诗歌和宗教圣歌的强大感染力中是很明显的，就如同节奏在儿童记单词和婴儿呀呀学语中所具有的作用。

2）*音韵学*（phonology）处理的是形成语言的声音。语言学的这一分支对于计算机语音识别和生成很重要。

3）*词态学*（morphology）涉及组成单词的成分（词素）。包括控制单词构成的规律，如前缀（un-、non-、anti-等）的作用和改变词根含义的后缀（-ing、-ly 等）。词态分析对于确定单词在句子中的作用很重要，包括时态、数量和部分语音。

4）*语法*（syntax）研究将单词组合成合法的短语和句子的规律，并运用这些规律解析和生成句子。这是语言学分析中形式化最好因而自动化最成功的部分。

5）*语义学*（semantics）考虑单词、短语和句子的意思以及自然语言表示中传达意思的方法。

6）*语用学*（pragmatics）研究使用语言的方法和对听众造成的效果。例如，语用学能够指出为什么通常用 "知道" 来回答 "你知道几点了吗?" 是不合适的。

7）*世界知识*（world knowledge）包括自然世界、人类社会交互世界的知识以及交流中目标和意图的作用。这些通用的背景知识对于理解文字或对话的完整含义是必不可少的。

虽然这些分析层次看上去是自然而然的而且符合心理学的规律，但是它们在某种程度上是强加在语言上的人工划分。它们之间广泛的交互，既使很低层的语调和节奏变化也会对说的话的意思产生影响，例如讽刺的使用。这种交互在语法和语义的关系中体现得非常明显，虽然沿着这些界线进行某些划分似乎很必要，但是确切的分界线很难定义。例如，像 "They are eating apples" 这样的句子有多种解析，只有注意上下文的意思才能决定。语法也会影响语义，比如短语结构在理解句子含义中所起的作用。虽然我们经常讨论语法和语义之间的精确区别，但是心理学的证据和它在管理问题复杂性中的作用只有有保留的予以探讨。在第 16 章将再次讨论语言理

解和翻译中的这些深层问题。

虽然不同自然语言理解程序的组织采用不同的原理和应用——例如数据库前端、自动翻译系统、故事理解程序——但它们都必须将原句子的含义翻译成一种内部表示。一般情况下，自然语言理解遵循图 15-2 所示的过程。

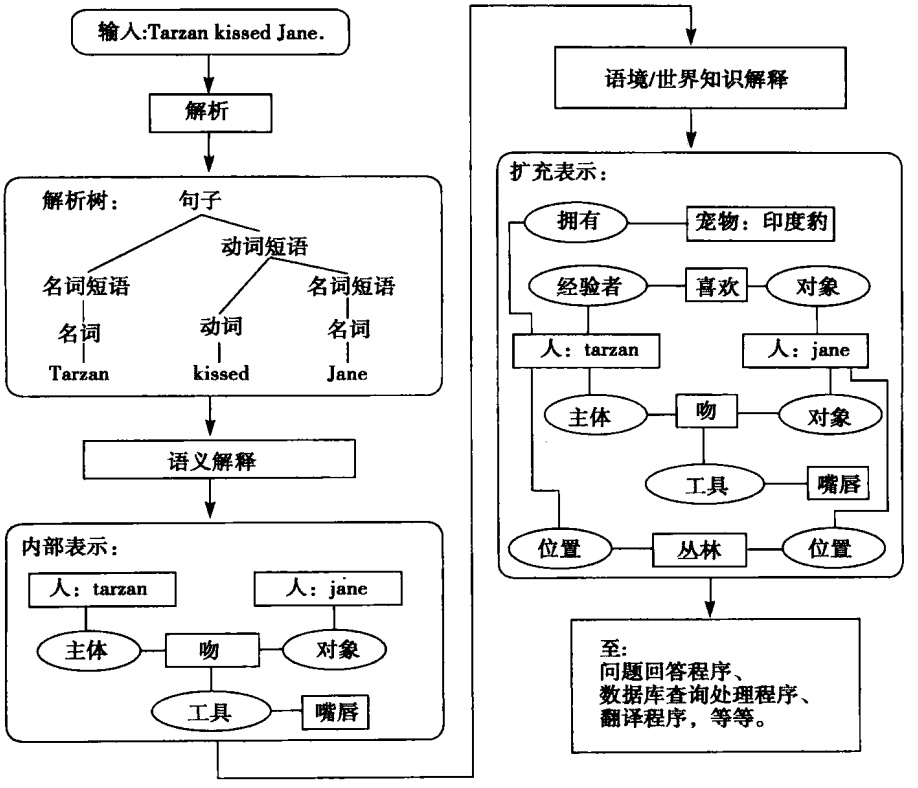


图 15-2　生成句子内部表示的各阶段

第一个阶段是解析，分析句子的句法结构。解析的任务在于既验证句子在句法上的合理构成，又决定语言的结构。通过识别主要的语言关系，如主－谓、动－宾和名词－修饰，解析器可以为语义解释提供一个框架。我们通常用解析树来表示它。解析器运用的是语言中语法、词态和部分语义知识。

第二个阶段是语义解释，旨在对文本的含义生成一种表示，如图 15-2 中的概念图所示。其他的一些通用的表示方法包括概念依赖、框架和基于逻辑的表示法等。语义解释使用如名词的格或动词的及物性等关于单词含义和语言结构的知识。在图 15-2 中，程序利用的知识是：根据单词 kiss 的含义，将默认值 lips（嘴唇）添加到 kissing 的对象中。此外语义一致性检查也在这一阶段完成。例如，动词 kiss 的定义可能包含这样的约束：当主体是人时，吻的对象是人，即正常情况下，Tarzan 吻的是 Jane，而不是印度豹。

第三阶段要完成的任务是将知识库中的结构添加到句子的内部表示中，以生成句子含义的扩充表示。在这一步中，类似“Tarzan 喜欢 Jane”、“Tarzan 和 Jane 生活在丛林中”、“印度豹是 Tarzan 的宠物”这样的用以充分理解语言所必需的世界知识被添加了进来。这样产生的结构表达了自然语言文字的意思，可以被系统用来进行后续处理。

举例来说，在数据库前端，扩充结构可能结合了查询含义的表示和数据库组织的知识。这种结构能够被翻译成相应的数据库语言查询语句（见 15.5.2 节）。而在故事理解程序中，这种扩

充结构可能表示故事的意思，并能够用来回答关于故事的问题（见第 7 章中对脚本的讨论和 15.5.3 节中的文本摘要）。

绝大多数（非概率的）自然语言理解系统中都存在这三个阶段，尽管相应的软件模块不一定被明确划分出来。例如，许多程序不生成明确的解析树，但是直接生成内部语义表示。无论怎样，解析树都隐含在对句子的解析中。*增量解析*（Allen 1987）是一项应用广泛的技术。在这种技术中，句子中的重要部分一旦被解析生成，内部表示的一个片段将随之生成。随着解析的进行，这些片段合并成完整的结构。我们也可以利用这些片段来解决句子模糊性的问题，还可以用来指导解析过程。

15.2 语法

15.2.1 使用上下文无关文法说明和解析

第 3 章介绍了*重写规则*说明文法。下面的规则是关于“The man likes the dog”这样的简单及物句的文法，我们给规则加上了编号。

1. sentence ↔ noun_phrase verb_phrase
2. noun_phrase ↔ noun
3. noun_phrase ↔ article noun
4. verb_phrase ↔ verb
5. verb_phrase ↔ verb noun_phrase
6. article ↔ a
7. article ↔ the
8. noun ↔ man
9. noun ↔ dog
10. verb ↔ likes
11. verb ↔ bites

6 到 11 号规则的右边都是英语单词；这些规则构成了句子中出现的单词的字典。这些单词是文法的*终结符*，定义了语言的*词典*。描述高层语言概念（sentence、noun_phrase，等等）的术语称为*非终结符*。非终结符以**黑字体**表示。注意：终结符不会出现在任何规则的左边。

合法的句子是指那些使用规则*导出*的终结符组成的任意串。推导从非终结符 sentence 开始，通过一系列文法规则定义的代换，得出终结符串。合法代换用规则右边的符号代换与规则左边匹配的符号。在推导的过程中，字符串同时包含终结符和非终结符的，叫做*句型*。

句子“The man bites the dog”的推导过程如下：

STRING	APPLY RULE #
sentence	1
noun_phrase verb_phrase	3
article noun verb_phrase	7
The noun verb_phrase	8
The man verb_phrase	5
The man verb noun_phrase	11
The man bites noun_phrase	3
The man bites article noun	7
The man bites the noun	9
The man bites the dog	

这是一个自顶向下推导的例子：开始于 sentence 符号，向下推导出终结符串。自底向上推导开始于终结符串，用规则的左边代换右边的模式，直到仅剩余 sentence 符号为止。

推导可以表示为树结构，即解析树，其中每个结点是文法规则集中的一个符号。树内部的结点是非终结符；每个结点及其子结点分别对应于一条文法规则的左边和右边。叶结点是终结符，sentence 符号是树的根。句子“The man bites the dog”的解析树见图 15-3。

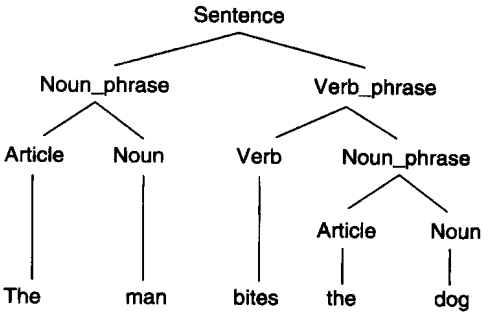


图 15-3　句子“The man bites the dog”的解析树

存在推导或解析树，这不仅能证明句子在文法上是合法的，而且能确定句子的结构。文法的短语结构定义了语言深层的词组织。例如，将 sentence 拆分为 noun_phrase 和 verb_phrase 可以表明动作和其主体之间的关系。这种短语结构在语义解释中起着实质性作用，它定义了包含语义处理的推导的中间阶段，如 14.3 节。

解析是针对输入串，依据文法的形式定义，建立推导过程和解析树的问题。解析算法分为两类：自顶向下解析器和自底向上解析器。自顶向下解析器由顶层的 sentence 符号开始，尝试建立一棵树，其叶子与目标句子相匹配；自底向上解析器由句子中的单词（终结符）开始，尝试发现一系列约简，这些约简能够产生 sentence 符号。

解析面临的一个巨复杂的难题是，在推导的任意一步，从几条潜在可利用规则中如何决定使用哪一条。如果做出错误选择，解析器就可能无法识别一个合法的句子。例如，使用自底向上的方式解析句子“The dog bites”，规则 7、9 和 11 生成字符串 article noun verb。这里如果使用规则 2 将会生成 article noun_phrase verb；无法化简为 sentence 符号。解析器应该使用规则 3。自顶向下解析中也会出现类似的情况。

在任意解析步骤中选择正确规则有两种处理方法：一种是允许解析器设置回溯指针，如果做出了错误的决定就返回到问题情景（见 3.2 节）；另一种是利用预判来检查输入流，寻找有助于确定正确规则的特征。使用这两种方法，都必须在保证得到正确解析的前提下小心地控制执行的复杂性。

相反的问题是从内部语义表示生成或产生合法的句子。生成从一些有意义的内容（如语义网或概念依赖图）开始，构造一个文法正确的句子以表达其意义。生成不完全是解析的反向过程；会遇到特有的困难，从而需要单独的方法予以处理。在补充材料中我们阐述了递归下降的上下文无关和上下文相关的解析器。

由于解析在程序设计语言和自然语言处理中都特别重要，研究人员开发出了许多不同的解析算法，包括自顶向下和自底向上两种策略。纵览所有的解析算法超出了本章的范围，我们将详尽地讨论某几种解析方法。下一节阐述 Earley 解析器，它是一个重要的基于动态规划的多项式时间解析器。在 15.3 节，我们将通过引入转移网络解析器来介绍解析器的语义约束。虽然这些解析器不能完全胜任解析自然语言语义的任务，但是它们构成了扩充转移网络的基础。扩充转移网络已经证明是自然语言处理的一个有用和有力的工具。

15.2.2　Earley 解析器：动态规划二次访问

动态规划最初由 Richard Bellman（1956）提出，本书 4.1 节给出了动态规划的几个例子。它的思想很简单，即把一个复杂的问题分解为许多彼此相关的子问题，将局部解的结果保存下来

以便于在之后的求解过程中能够被再次使用。这种方法有时也被称为对子问题结果的记忆与重用。

在模式匹配中有很多动态规划的例子。如计算两个位串或字符串之间的差异问题。两串之间的总差异是两串内特定元素之间的差异的函数。在拼写检查当中，拼写检查把与误拼写的单词最相近的那个单词呈现出来（见 4.1 节）。还有一个例子是，在语音理解中我们将所识别的单词看作是输入流中可能的音素的函数。当音素以相关概率被识别出来后，那么由单个语音的组合概率值的函数所产生的最匹配的单词也就随之确定了。本节将用动态规划方法阐明 Earley 解析器是如何决定单词串是否组成句法正确的句子的。伪代码根据 Jurafsky 和 Martin（2008）的代码改编而来。

在对可接受的句法结构进行递归、深度优先、从左至右的搜索中，15.2.1 节提到的解析算法常常被使用。在这种查找方法中，串的前端（最左边的）元素的大部分可接受的局部解析结果被反复生成。在整个解析结构范围内，早先生成的局部解被再次访问，这是之后的回溯（backtracking）查找的需要，而且在更大规模的解析中，对局部解的再次访问的复杂性将呈指数级增长。局部解析结果一旦生成，就可以被保存并且在之后对单词串的全盘解析中被再次使用——动态规划方法在这个过程中效果显著。Earley（1970）给出了第一个基于动态规划的解析器。

记忆与句点对

在用 Earley 算法进行解析的过程中，我们利用一种称为图（chart）的数据结构来记录局部解（局部解析结果）。从而我们常常将 Earley 解析方法称为图解析（chart parsing），这种图通过运用句点（dot）文法规则来产生。

句点文法规则提供了一种表示，这种表示能在图中表明在任意时刻解析过程所处的状态。每条句点规则依据句点是否处于文法规则开始端、中间某处或右端（the right hand side，RHS）末尾而被分为三个种类，分别被称为解析的*初始*（initial）、*局部*（partial）和*结束*（completed）阶段：

初始预测: Symbol→ · RHS_unseen
局部解析: Symbol→RHS_seen · RHS_unseen
结束解析: Symbol→RHS_seen ·

此外，我们可以很自然地将含有不同句点规则的状态与解析所产生的解析树的边对应起来。来看下面这个很简单的文法例子，在这个例子中终结符被引号括起来，例如“mary”：

Sentence→Noun Verb
Noun→“mary”
Verb→“runs”

我们对上面这条语句执行自顶向下、自左至右的解析过程，下面给出解析所产生的状态序列：

Sentence→ · Noun Verb	*预测*: *名词后接动词*
Noun→ · Mary	*预测*: mary
Noun→Mary ·	*扫描*: mary
Sentence→Noun · Verb	*结束*: Noun; *预测*: Verb
Verb→ · runs	*预测*: runs
Verb→runs ·	*扫描*: runs

Sentence→Noun Verb ·　　结束: Verb, 结束: sentence

值得注意的是，扫描（Scanning）和结束步骤决定了解析的结果。预测步骤描述了在当前情形下应该被采纳的最可能的解析规则。图 15-4 的解析树给出了扫描和预测所产生的状态。

Earley 算法对给定的输入进行解析时，执行自上至下、自左至右的预测。每次预测结果被记录为一个状态，该状态包含所有与预测相关的信息，而每个状态的一个关键要素是句点规则（在下一节我们将给出第二个要素）。检测完输入句子的一个特定单词后所产生的所有预测结果被称作一个状态集（the state set）。

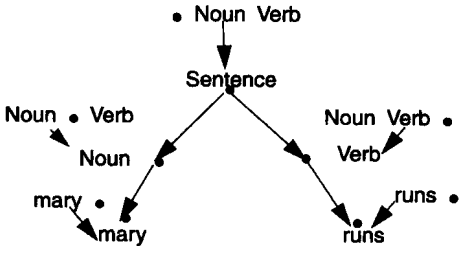


图 15-4　句点规则与解析树的产生之间的对应关系

对于一个由 n 个单词（w_1 至 w_n）组成的输入句子来说，共有 n + 1 个状态集产生：[S_0, S_1, …, S_n]。初始状态集 S_0 含有对输入的任何单词做检测之前所做的预测，状态集 S_1 含有检测单词 w_1 之后的预测结果，依此类推。我们把解析器产生的状态集的全体称为图（chart）。图 15-4 表明了状态集的产生与输入单词的检测之间的相互关系。尽管在传统意义上，状态集指的是解析的每个元素的状态集合，但是状态的产生次序是非常重要的。因此我们把图中的每个元素称为状态列表（state list），表示为 [$State_1$, $State_2$, …, $State_n$]，这种状态列表用 Prolog 可以很好地得以实现（见补充材料），其中状态列表可以被保存为 Prolog 列表。这样我们就可以将状态列表中的每个状态表示为一个由括号括起来的特定的符号序列，例如（$ → · S）。

现在我们用 Earley 算法和上述文法来解析 mary runs 这个很简单的句子。算法以一个虚拟起始状态（$ → · S）开始，此状态是状态列表 S_0 的第一个成员。此状态表示输入串能够被作为一个句子得以解析，而且在检测输入单词之前，该状态可被插入至状态列表 S_0，句法分析成功后将产生一个结束状态列表 S_n，该状态列表包含状态（$ →S · ）。

从状态列表 S_0 开始，解析器循环执行，执行过程按顺序检测当前状态列表中的每个状态 S_i，并产生新的状态。每个新状态由预测器（predictor）、扫描器（scanner）和结束器（completer）中的任意一种步骤产生。究竟采用哪一种步骤，由状态 S 中的句点规则，特别是带文法符号的句点规则来决定。

在上述例子中，待检测的初始状态包含规则（$ → · S）。因为句点后紧跟符号 S，在该状态下我们期望在接下来的输入串中出现 S 的一个范例（instance）。因为 S 是一个文法非终结符，预测器将产生对 S 解析后的所有状态。在这个例子中，S 只能被 S→Noun Verb 所替代，因此只有状态（S→ · Noun Verb）被添加到 S_0 中。在该状态中，句点之后是非终结符 Noun，该状态可能是语言中的一部分内容，算法将检查接下来输入的单词以便检验所做的预测。扫描器将完成这个工作。由于下一个输入的单词确实与预测相匹配，即 mary 确实是一个名词，从而扫描器产生一个新的状态来记录这种匹配：（Noun→mary · ）。因为该状态依赖于输入单词 w_1，所以它没有被插入到状态列表 S_0 中，而是被插入到状态列表 S_1 中，成为 S_1 的第一个状态。至此解析图包含了两个状态列表，如下所示，在每个状态之后我们给出了产生该状态采用的步骤：

S_0: [($ → · S),　　虚拟起始状态
　　(S→ · Noun Verb)]　　预测器

S_1: [(Noun→mary ·)]　　扫描器

状态列表 S_0 中的每个状态现已处理完毕，算法开始处理 S_1 以及状态（Noun→mary · ）。这是一

个结束状态，结束器被启动。这时我们期望出现一个名词，即具备 · Noun 模式，结束器产生一个新的状态，通过在名词标志的前方打句点来记录名词的发现。在这个例子中，结束器在 S_0 中产生了状态（S→ · Noun Verb）且在列表 S_1 中产生了新状态(S→Noun · Verb)。该状态被视为是语音的一部分，扫描器检测下一个输入单词 w_2。由于 w_2 是一个动词，扫描器产生状态（Verb→runs · ）并把此状态加入到 S_2 中，此过程如下所示：

S_0: [($→ · S),	起始
(S→ · Noun Verb)]	预测器
S_1: [(Noun→mary ·),	扫描器
(S→Noun · Verb)]	结束器
S_2: [(Verb→runs ·)]	扫描器

现在处理新状态 S_2，结束器在（S→Noun · Verb）中打点来产生(S→Noun Verb ·)，继而产生状态（ $→S · ），从而表明对句子 mary runs 成功地完成了解析任务。其解析过程产生的最终图包含以下三个状态列表：

S_0: [($→· S),	起始
(S→ · Noun Verb)]	预测器
S_1: [(Noun→mary ·)]	扫描器
(S→Noun · Verb)]	结束器
S_2: [(Verb→runs ·)]	扫描器
(S→Noun Verb ·),	结束器
($→S ·)]	结束器

Earley 算法的伪代码

为了从计算的角度表示由上述句点对规则所产生的状态列表，我们以索引形式来显示对一条文法规则被解析的程度。下面首先说明这种表示方法，而后给出 Earley 算法的执行伪代码。

状态列表中的每个状态被附加一个索引以便表示有多少输入流已被处理。这里我们将每个状态的描述扩展为（句点规则[i, j]）的表示形式，其中[i, j]对表示文法规则的右端（right hand side, RHS）截至当前有多少已被扫描或已被分析。对于一条解析过的规则来说，句点 · 表明了零个或多个已处理和未处理的元素。我们用（A→Seen · Unseen, [i, j]）来表示这条规则，其中 i 是已处理（Seen）的起始位置，j 是句点在单词序列中的位置。

让我们针对句子 mary runs，将索引添加到上述所讨论的解析状态中去：

($→ · S[0,0])	由预测器产生, i = j = 0, 解析内容为零
(Noun→mary · , [0,1])	扫描器处理 w_1, i = 0, j = 1
(S→Noun · Verb, [0,1])	结束器已处理名词(mary), i = 0, j = 1
(S→Noun Verb · , [0,2])	结束器已处理句子 S, i = 0, j = 2

状态索引模式（state indexing pattern）表明了运用单词 w_i 索引的句点规则，由三种状态生成器产生结果的过程。

总结，产生状态列表中的状态包括三个步骤：由预测器产生[j, j] 状态，并进入图 chart[j]；由扫描器处理单词 w_{j+1}，产生[j, j+1] 状态，并进入 chart[j+1]；由结束器基于规则产生[i, j] 状态，其中 i<j，并将状态入口加入到 chart[j]. 注意，句点规则中的状态[i, j] 总是进入 chart[j]

的状态列表。因此，对于一个有 n 个单词的句子来说，状态列表包括 chart[0]，…，chart[n]。基于上述图表示的索引方案，我们给出 Earley 解析器的伪代码。

```
function EARLEY-PARSE(words, grammar)  returns chart
begin
    chart := empty
    ADDTOCHART(($ → • S, [0, 0]), chart[0])              % dummy start state
    for i from 0 to LENGTH(words) do
        for each state in chart[i] do
            if rule_rhs(state) = ... • A ... and A is not a part of speech
            then PREDICTOR(state)
            else if rule_rhs(state) = ... • L ...          % L is part of speech
                then SCANNER(state)
                else COMLETER(state)                        % rule_rhs = RHS •
end

procedure PREDICTOR((A → ... • B ..., [i, j]))
begin
    for each (B → RHS) in grammar do
        ADDTOCHART((B → • RHS, [j, j]), chart[j])
end

procedure SCANNER((A → ... • L ..., [i, j]))
begin
    if (L → word[j]) is_in grammar
        then ADDTOCHART((L → word[j] •, [j, j + 1]), chart[j + 1])
end

procedure COMPLETER((B → ... •, [j, k]))
begin
    for each (A → ... • B ..., [i, j]) in chart[j] do
        ADDTOCHART((A → ... B • ..., [i, k]), chart[k])
end

procedure ADDTOCHART(state, state-list)
begin
    if state is not in state-list
    then ADDTOEND(state, state-list)
end
```

在上面的例子中，对句子 mary runs 的 Earley 解析过程很简单但是很说明问题，在叙述中我们详细给出了对状态列表及其索引的描述。让我们看一个复杂并且模棱两可的例句："John called Mary from Denver"（补充材料中给出了 Earley 算法解析该句的 Prolog、Java 代码）。这个句子的歧义性导致了解析可能产生两种不同的结果。我们通过执行基于 Earley 解析的动态规划的回溯操作来选择不同的解析结果。

15.3　转移网络解析器及语义学

此前我们提出了支持基于符号的句法分析的表示方法和算法。句法关系分析，甚至更小范围内的上下文相关解析（例如，名词-动词约定），忽略了与语义学的联系。本节我们通过介绍*转移网络*来解决这个问题。

15.3.1　转移网络解析器

转移网络解析器将文法表示为有限状态自动机的集合，或者说转移网络。每个网络对应于文法中一个单一非终结符。终结符或非终结符标注在网络弧线上。从开始状态到结束状态的每条路径对应于该非终结符的一些规则；路径中弧上标签的序列是规则右边的符号序列。图 15-5 给出了 15.2.1 节中的文法的转移网络表示。当一个非终结符有多于一条规则时，对应的网络有多条从起点到目标的路径。例如，规则 noun_phrase ↔ noun 和 noun_phrase ↔ article noun 用通过 noun_phrase 网络的可选路径表示。

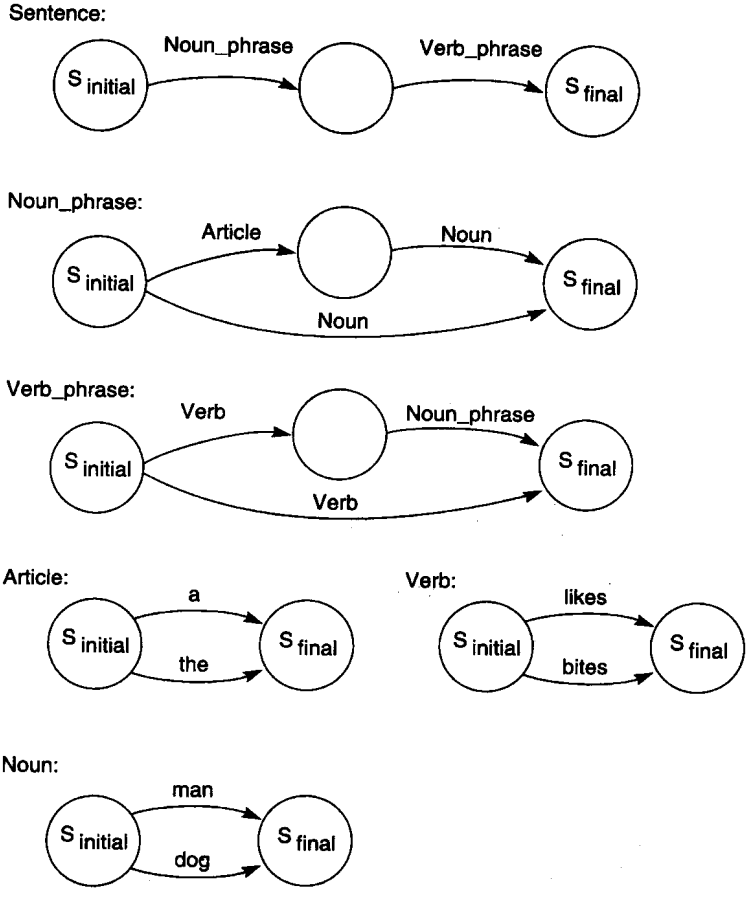


图 15-5　一个简单的英语文法的转移网络定义

我们通过置换文法规则右边的非终结符来寻找一个成功的转移网络。例如，要解析一个句子，转移网络解析器必须找到一个穿过句子网络的转移。转移从初始状态（$S_{initial}$）出发，经过 noun_phrase 转移和 verb_phrase 转移，到达结束状态（S_{final}）。这个过程等价于将最初的 sentence 符号替换成符号对 noun_phrase verb_phrase。

当经过一条弧时，解析器必须检查其标签。如果弧标签是终结符，解析器就检查输入字符串，看下一个单词是否与弧匹配。若不匹配，转移不能进行。如果弧标签是非终结符，解析器就通过检索网络寻找该非终结符，并递归尝试寻找一条穿过它的路径。若解析器无法找到穿过此网络的路径，那么不能通过顶层的弧。这种情况将导致解析器回溯并尝试另一条路径。这样，解析器试图寻找一条穿过 sentence 网络的路径；如果成功，输入字符串就是一个文法上合法的

句子。

让我们以“Dog bites”这个简单的句子为例，对它的分析过程如图 15-6 所示。

1）解析器从 sentence 网络开始，试图沿着标有 noun_phrase 的弧前进。为此，解析器检索网络寻找 noun_phrase。

2）在 noun_phrase 网络中，解析器首先尝试标注 article 的转移，因此到达 article 网络分支。

3）解析器无法找到一条路径到达 article 网络的终点，因为句子的第一个词“Dog”与弧上的标签不匹配。解析器回退到 noun_phrase 网络。

4）解析器尝试在 noun_phrase 网络中沿着 noun 弧前进，到达 noun 网络分支。

5）解析器成功地通过弧“dog”，因为标签与输入流的第一个单词对应。

6）noun 网络成功返回。允许经过 noun_phrase 网络中标注 noun 的弧到达最终状态。

7）noun_phrase 网络成功返回到上层网络，允许通过 noun_phrase 弧的转移。

8）经过类似的步骤序列，解析句子的 verb_phrase 部分。

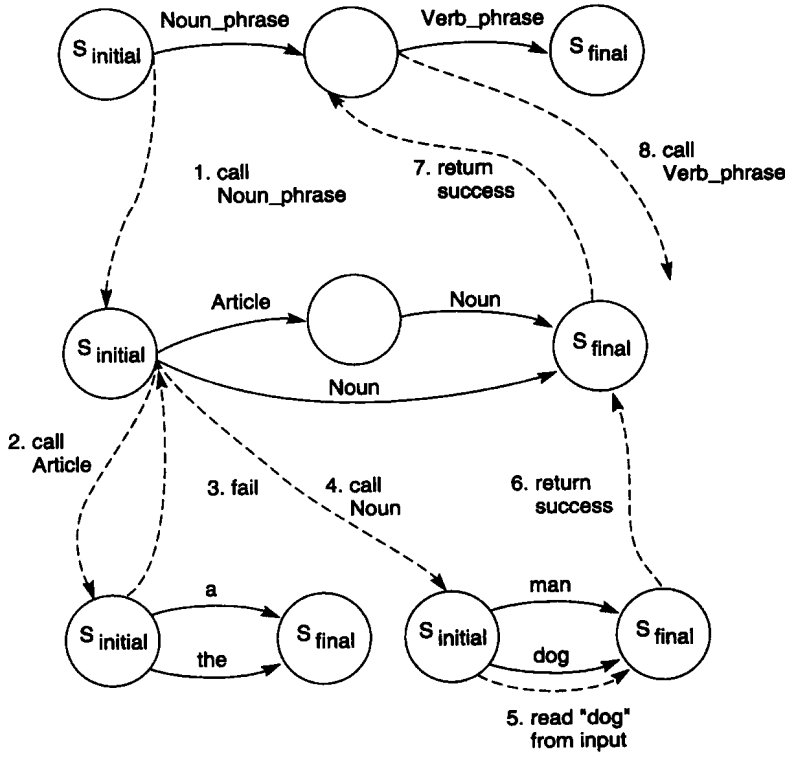


图 15-6　对句子“Dog bites”转移网络解析的跟踪

转移网络解析器的伪代码表示如下。解析器用两个互相递归的函数定义，parse 和 transition。parse 以一个文法符号作为参数：如果符号是终结符，parse 将其与输入流的下一个单词比较检查。如果符号是非终结符，parse 检索与该符号有关的转移网络，并调用 transition 来寻找通过网络的路径。要解析一个句子，需要调用 parse(sentence)。

```
function parse(grammar_symbol);
begin
  save pointer to current location in input stream;
  case
    grammar_symbol is a terminal:
      if grammar_symbol matches the next word in the input stream
        then return (success)
        else begin
          reset input stream;
          return (failure)
        end;
    grammar_symbol is a nonterminal:
      begin
        retrieve the transition network labeled by grammar symbol;
        state := start state of network;
        if transition(state) returns success
        then return (success)
        else begin
          reset input stream;
          return (failure)
        end
      end
  end
end.

function transition (current_state);
begin
  case
    current_state is a final state:
      return (success)
    current_state is not a final state:
      while there are unexamined transitions out of current_state
        do begin
          grammar_symbol := the label on the next unexamined transition;
          if parse(grammar_symbol) returns (success)
            then begin
              next_state := state at end of the transition;
              if transition(next_state) returns success;
                  then return (success)
            end
        end
    return (failure)
  end
end.
```

transition 以转移网络中的一个状态为参数，以深度优先的方式试图寻找穿过该网络的一条路径。因为解析器可能犯错误并且需要回溯，parse 保留了输入流当前位置的指针。这样，当解析器回溯时输入流就可以恢复（reset）到这个位置。

这个转移网络解析器可以确定一个句子文法是否正确，但没有构造解析树。可以通过让函数返回解析树的子树替代符号 success 来实现这一功能。为实现此功能所做的改动如下：

1）每次以终结符符号作为参数调用函数 parse 时，若该终结符与输入的下一个符号匹配，返回一棵只包含以该符号为叶结点的树。

2）当以非终结符 grammar_symbol 作为参数调用函数 parse 时，调用 transition。若 transition 成功，则返回一个子树的有序集合（后面说明）。parse 将这些子树合并为一棵树，根为 grammar_symbol，孩子为 transition 返回的子树。

3）在搜索通过网络路径的过程中，transition 对每个弧的标签调用 parse。成功时，parse 返回表示该符号的一棵子树。transition 将这些子树保存在一个有序集合中，在成功找到通过网络的路径时，返回与路径中弧的标签序列对应的解析树的有序集合。

15.3.2　乔姆斯基层次和上下文相关文法

在 15.3.1 节，用*上下文无关文法*定义了一个很小的英文子集。上下文无关文法允许规则的左边只有一个非终结符。因此，规则可以应用于任意出现的该符号上，而无论其上下文如何。尽管上下文无关文法已被证明是计算机科学中定义程序设计语言和其他公式的有力工具，我们仍然有理由认为它无法单独表示自然语言语法的规则。例如，假如在 15.2.1 节中的文法中加入单复数名词和动词，看会出现什么情况：

noun ↔ men
noun ↔ dogs
verb ↔ bites
verb ↔ like

得到的文法能解析类似“The dogs like the men”的句子，但是同时也接受“A men likes a dogs”。解析器接受这些句子是因为当前的规则不能用上下文确定何时单复数必须协调。将 sentence 定义为 noun_phrase 后跟着 verb_phrase 的规则没有要求主语和动词的单复数一致，或者冠词与名词一致。

可以扩展上下文无关语言来处理这些情况，但是更自然的方法是使文法上下文相关，即分析树中的成分可以设计成互相包含。乔姆斯基（chomsky）（1965）首先提出了一个分层体系和逐级增强的文法（Hopcroft 和 Ullman 1979）。在层次的底部是*正则语言类*。正则语言是文法可以用有限状态自动机定义的语言，见 3.1 节。虽然正则语言在计算机科学中有广泛的应用，但是它们无法表示大多数程序设计语言的语法。

*上下文无关语言*在乔姆斯基层次中处于正则语言之上。上下文无关语言用重写规则定义，如 15.2.1 节；上下文无关规则中左边可以只有一个非终结符。转移网络解析器能够解析上下文无关语言类。需要注意的是，如果在转移网络解析器中不允许递归（即弧上只可以标注终结符，转移不可以“调用”其他网络），那么这样定义的语言类就对应于正则表达式。因此，正则语言是上下文无关语言的一个真子集。

*上下文相关语言*构成了上下文无关语言的一个真超集。它们用*上下文相关文法*定义，允许规则的左边有多于一个的符号，使得定义规则应用的上下文成为可能。这保证了能满足全局约束，如单复数一致和其他语法检查。上下文相关文法规则上的惟一限制是右边至少要和左边一样长（Hopcroft 和 Ullman 1979）。

第四类是*递归可数语言类*，构成了上下文相关语言类的一个超集。递归可数语言可以用无约束产生式规则定义；因为这些规则比上下文相关规则的限制少，所以递归可数语言是上下文

相关语言的真超集。虽然这类语言在计算机科学理论中很重要，但是还没有引起人们的兴趣来定义自然语言语法。本节其余的部分重点在将英语看作是一种上下文相关语言。

下面给出表示 article noun verb 形式的句子的一个简单的上下文无关文法，该文法能处理冠词和名字以及主语和动词之间单复数的一致性：

sentence ↔ noun_phrase verb_phrase
noun_phrase ↔ article number noun
noun_phrase ↔ number noun
number ↔ singular
number ↔ plural
article singular ↔ a singular
article singular ↔ the singular
article plural ↔ some plural
article plural ↔ the plural
singular noun ↔ dog singular
singular noun ↔ man singular
plural noun ↔ men plural
plural noun ↔ dogs plural
singular verb_phrase ↔ singular verb
plural verb_phrase ↔ plural verb
singular verb ↔ bites
singular verb ↔ likes
plural verb ↔ bite
plural verb ↔ like

在这个文法中，非终结符 singular 和 plural 提供限制来确定何时应用不同的 article、noun 和 verb_phrase 规则，保证数的一致。使用本文法对句子“The dogs bite”的推导过程如下：

sentence.
noun_phrase verb_phrase.
article plural noun verb_phrase.
The plural noun verb_phrase.
The dogs plural verb_phrase.
The dogs plural verb.
The dogs bite.

类似地，我们可以用上下文相关文法执行语义一致的检查。例如，可以通过在文法中加入一个非终结符 act_of_biting 来禁止类似“Man bites dog”的句子。这个非终结符在规则中被检查，防止“man”作主语并且包含“bites”的句子出现。

虽然上下文相关文法能够定义那些上下文无关文法无法捕获的语言结构，但是在设计实际解析器的时候有许多不便：

1）上下文相关文法急速地增加了文法中规则和非终结符的数量。可以想象一下包含单复数、人称和英语需要的所有其他一致形式的上下文相关文法的复杂性。

2）它们掩盖了语言的短语结构，而在上下文无关文法中可以清晰表示。

3）由于试图对文法本身所含的一致性和语义一致进行更复杂的检查，从而失去了将语言的句法和语义成分分开处理的许多优点。

4）上下文相关文法没有考虑文本含义语义表示的建立问题。一个仅仅能对句子做出接受或不接受句子的解析器是不够用的；它必须能返回句子语义含义的有用表示。

在补充材料中我们给出了上下文无关和上下文相关的解析器和句子生成器，Prolog 程序控制

采用深度优先的递归下降策略。下一节，我们将研究*扩充转移网络*（Augmented Transition Network，ATN），它能够定义上下文相关语言，但是在解析器设计上比上下文相关文法有优势。

15.3.3　ATN 解析器的语义

上下文相关文法的一个选择是保留上下文无关文法规则的简单结构，但要增加用以进行上下文测试的附加过程。解析过程中，当调用规则时也执行这些过程。我们不是使用文法来描述类似单复数、时态和人称等概念，而是将它们表示为与文法中终结符和非终结符相关的特征。与文法规则相关的过程访问这些特征，为其赋值并执行必要的检查。用上下文无关文法的扩充来实现上下文相关文法包括*扩充短语结构文法*（Heidorn 1975，Sowa 1984）、*逻辑文法扩充*（Allen 1987）和*扩充转移网络*（ATN）。

本节将介绍 ATN 解析，并概要描述关于 15.2.1 节中介绍的关于“dogs world”句子的简单 ATN 解析器。我们考虑图 15-2 的头两步：创建解析树并用其构造句子意义的表示。虽然 ATN 解析器可以使用语义网、脚本、框架或基于逻辑的表示，但是本例中我们使用概念图。

ATN 通过在网络中的弧上附加过程扩充了转移网络。ATN 解析器经过弧时执行这些附加过程。这些过程可以为文法特征赋值以及执行检查，如果特定条件（如单复数一致）不满足将导致转移失败。这些过程还构建解析树，用来生成句子意义的内部语义表示。

我们将终结符和非终结符都表示为带附加特征的标识符（如 verb、noun_phrase）。例如，一个词可以用词根连同其词性、数、人称等特征来表示。文法中的非终结符可用类似的方法描述。名词短语用其冠词、名词、数和人称描述。终结符和非终结符都可以用带槽和值的类似框架的结构来表示。这些槽的值规定了文法特征，或者指向其他结构。例如，句子框架的第一个槽包含一个指向名词短语定义的指针。图 15-7 显示了我们的简单文法中表示 sentence、noun_phrase 和 verb_phrase 的框架。

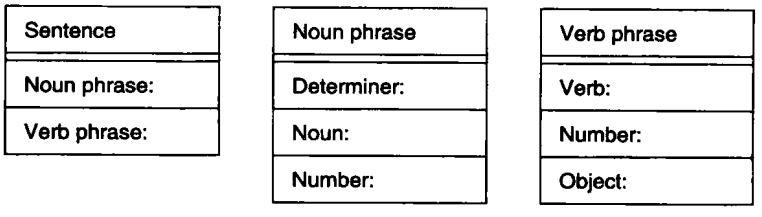


图 15-7　文法中 sentence、noun_phrase 和 verb_phrase 非终结符的结构表示

单独的单词也用相似的结构表示。字典中的每个词用一个框架来定义，框架中定义了其词性（冠词、名词，等等）、词根和重要文法特征。在我们的例子中，只检查单复数一致，而且只记录这一特征。更复杂的文法还要指明人称和其他特征。这些字典条目还可以指明语义解释用的词意的概念图定义。我们文法的完整字典见图 15-8。

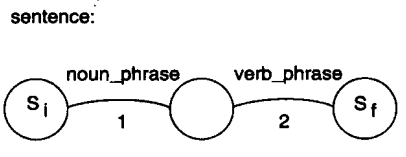
图 15-9 表示文法的 ATN，带有每条弧上执行检查的伪代码描述。弧上标注文法的非终结符（类似图 15-5）和数字；数字用来指明每条弧上附带的函数。要遍历弧必须成功运行这些函数。

当解析器为非终结符调用一个网络时，为该非终结符创建一个新框架。例如，进入 noun_phrase 网络时创建一个新 noun_phrase 框架。框架的槽由该网络的函数填充。槽可以赋以文法特征值或指向句法结构的指针（如 verb_phrase 可以包含 verb 和 noun_phrase）。当到达终点状态时，网络返回该结构。

当网络经过标有 noun、article 和 verb 的弧时，从输入流读入下一个单词，并在字典中检索该单词的定义。如果该单词不是期望的词性，规则失败；否则返回定义框架。

词	定义	词	定义
a	PART_OF_SPEECH: article ROOT: a NUMBER: singular	like	PART_OF_SPEECH: verb ROOT: like NUMBER: plural
bite	PART_OF_SPEECH: verb ROOT: bite NUMBER: plural	likes	PART_OF_SPEECH: verb ROOT: like NUMBER: singular
bites	PART_OF_SPEECH: verb ROOT: bite NUMBER: singular	man	PART_OF_SPEECH: noun ROOT: man NUMBER: singular
dog	PART_OF_SPEECH: noun ROOT: dog NUMBER: singular	men	PART_OF_SPEECH: noun ROOT: man NUMBER: plural
dogs	PART_OF_SPEECH: noun ROOT: dog NUMBER: plural	the	PART_OF_SPEECH: article ROOT: the NUMBER: plural or singular

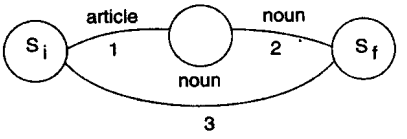
图 15-8　一个简单 ATN 的字典条目




```
function sentence-1;
  begin
    NOUN_PHRASE := structure returned by
          noun_phrase network;

    SENTENCE.SUBJECT := NOUN_PHRASE;
  end.

function sentence-2;
  begin
    VERB_PHRASE := structure returned by
          verb_phrase network;

    if NOUN_PHRASE.NUMBER =
                          VERB_PHRASE.NUMBER
      then begin
        SENTENCE.VERB_PHRASE := VERB_PHRASE;
        return SENTENCE
      end
      else fail
  end.
```

图 15-9　一个检查数一致并建立解析树的 ATN 文法

noun_phrase:

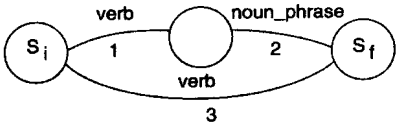



```
function noun_phrase-1;
  begin
    ARTICLE := definition frame for next word of input;

    if ARTICLE.PART_OF_SPEECH=article
      then NOUN_PHRASE.DETERMINER := ARTICLE
      else fail
  end.
```

```
function noun_phrase-2;
  begin
    NOUN := definition frame for next word of input;

    if NOUN.PART_OF_SPEECH=noun and
          NOUN.NUMBER agrees with
               NOUN_PHRASE.DETERMINER.NUMBER
      then begin
        NOUN_PHRASE:NOUN := NOUN;
        NOUN_PHRASE.NUMBER := NOUN.NUMBER
        return NOUN_PHRASE
      end
      else fail
  end.
```

```
function noun_phrase-3
  begin
    NOUN := definition frame for next word of input;

    if NOUN.PART_OF_SPEECH=noun
      then begin
        NOUN_PHRASE.DETERMINER := unspecified;
        NOUN_PHRASE.NOUN := NOUN
        NOUN_PHRASE.NUMBER := NOUN.NUMBER
      end
      else fail
  end.
```

verb_phrase:




```
function verb_phrase-1
  begin
    VERB := definition frame for next word of input;

    if VERB.PART_OF_SPEECH=verb
      then begin
        VERB_PHRASE.VERB := VERB;
        VERB_PHRASE.NUMBER := VERB.NUMBER;
      end;
  end.
```

```
function verb_phrase-2
  begin
    NOUN_PHRASE := structure returned by
                              noun_phrase network;

    VERB_PHRASE.OBJECT := NOUN_PHRASE;
    return VERB_PHRASE
  end.
```

```
function verb_phrase-3
  begin
    VERB := definition frame for next word of input;

    if VERB.PART_OF_SPEECH=verb
      then begin
        VERB_PHRASE.VERB := VERB;
        VERB_PHRASE.NUMBER := VERB.NUMBER;
        VERB_PHRASE.OBJECT := unspecified;
        return VERB_PHRASE;
      end;
  end.
```

图 15-9　（续）

在图 15-9 中，框架和槽用 Frame. Slot 符号来表示；例如 verb 框架的 number 槽表示为 VERB. NUMBER。随着解析的进行，每个函数建立并返回描述相关语法结构的框架。这一结构中包含指向低层网络返回的结构的指针。顶层 sentence 函数返回一个 sentence 结构，表示输入句子的解析树。这个结构被传递给语义解释器。图 15-10 显示句子“The dog likes a man”返回的解析树。

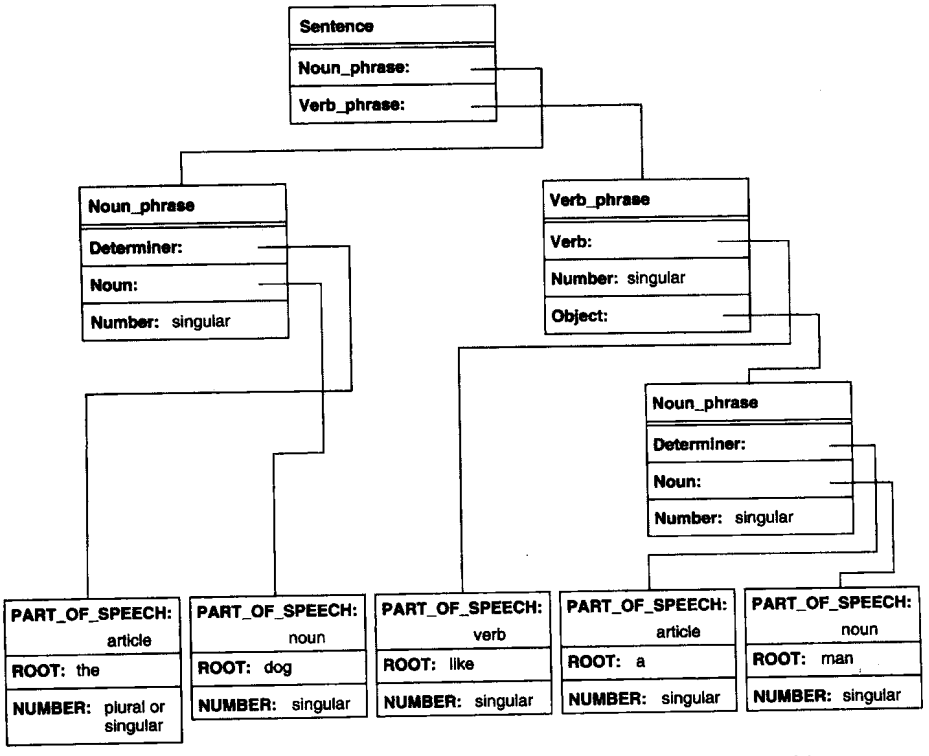


图 15-10　ATN 解析器返回的句子“The dog likes a man”的解析树

自然语言处理的接下来的步骤是获得解析树，例如图 15-10 中的，并构建领域知识和句子含义内容的语义表示。

15. 3. 4　结合句法和语义知识的 ATN

语义解释器通过从根（或 sentence 结点）开始遍历解析树来创建输入字符串含义的一个表示。在每个结点，语义解释器递归解释该结点的孩子，并将结果组成到一个概念图中；该概念图在整棵树上传递。例如，语义解释器建立 verb_phrase 的表示：递归建立 verb_phrase 的孩子、verb 和 noun_phrase 的表示，并将它们连接成为动词短语的一个解释。然后再将它传递给 sentence 结点，并和 subject 的表示组合起来。

递归在解析树的终结符停止。一些终结符（如名词、动词和形容词）会引发知识库中的概念被检索。其他一些终结符（如冠词）不直接与知识库中的概念对应，但是会限制图中其他概念。

我们例子中的语义解释器使用一个知识库来解释“dogs world”。知识库中的概念包括对象 dog 和 man，动作 like 和 bite。这些概念用图 15-11 的类型层次来描述。

除了概念外，我们必须定义概念图中会用到的关系。本例中，我们使用下面的概念：

agent 通过类型为 animate 的概念连接 act。agent 定义了动作和发起动作的活对象之间的关系。

experiencer 通过类型为 animate 的概念连接 state。它定义了精神状态和体验者之间的关系。

instrument 通过 entity 连接 act，定义了动作使用的工具。

object 通过 entity 连接 event 或 state，表示动宾关系。

part 连接类型为 physobj 的概念，定义整体和部分的关系。

动词在建立解释的过程中起着特别重要的作用，因为它定义了句子的主语、宾语和其他成分之间的关系。我们用案例框架表示每个动词，案例框架定义了：

1）语言上适于该动词的关系（主体、对象、工具，等等）。例如，及物动词有直接宾语；不及物动词则没有。

2）可以赋给案例框架任何成分的值的约束。例如，在动词“bites”的案例框架中，我们断言 agent 必须是 dog 类型。这样“Man bites dog”就因为语义不正确被否决。

3）案例框架成分的默认值。在“bites”框架中，我们为与 instrument 关系连接的概念设置默认值 teeth。

动词 like 和 bite 的案例框架见图 15-12。

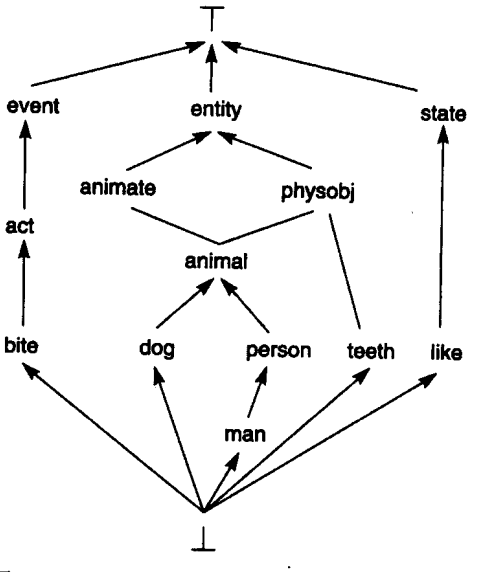


图 15-11　“dog world”例子中使用的类型层次

我们用规则或过程为解析树中每个潜在的结点定义动作，这些动作能够建立一个语义学的表示。我们例子的规则用伪代码过程来描述。在每个过程中，如果规定的连接或其他检查失败，那么该解释就因为语义不正确被否决。

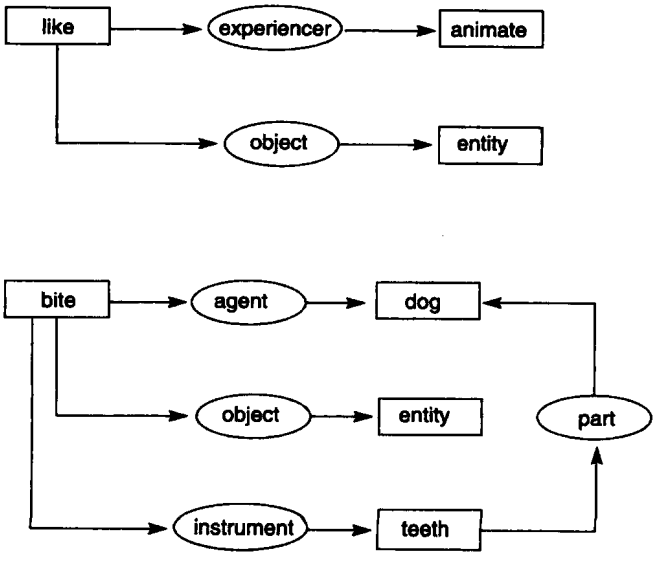


图 15-12　动词“like”和“bite”的案例框架

```
procedure sentence;

  begin
    call noun_phrase to get a representation of the subject;
    call verb_phrase to get a representation of the verb_phrase;
    using join and restrict, bind the noun concept returned for the subject to
      the agent of the graph for the verb_phrase
  end.

procedure noun_phrase;

  begin
    call noun to get a representation of the noun;
    case
      the article is indefinite and number singular: the noun concept is generic;
      the article is definite and number singular: bind marker to noun concept;
      number is plural: indicate that the noun concept is plural
    end case
  end.

procedure verb_phrase;

  begin
    call verb to get a representation of the verb;
    if the verb has an object
      then begin
        call noun_phrase to get a representation of the object;
        using join and restrict, bind concept for object to object of the verb
      end
    end.

procedure verb;

  begin
    retrieve the case frame for the verb
  end.

procedure noun;

  begin
    retrieve the concept for the noun
  end.
```

冠词不对应于知识库中的概念，但是却决定了它其后的名词是泛指还是特指。我们没有讨论复数概念的表示；可以参见 Sowa（1984）概念图的处理方法。

我们使用这些过程以及图 15-11 的概念层次和图 15-12 的案例框架，从图 15-9 的解析树建立句子“The dog likes a man”的语义表示，并跟踪语义解释器的动作。跟踪过程见图 15-13。

跟踪过程中出现的动作有（圆括号中的编号指图 15-13 中的编号）：

1）从句子结点开始，调用 sentence。

2）sentence 调用 noun_phrase。

3）noun_phrase 调用 noun。

4）noun 返回表示名词 dog 的概念（图 15-13 的 1）。

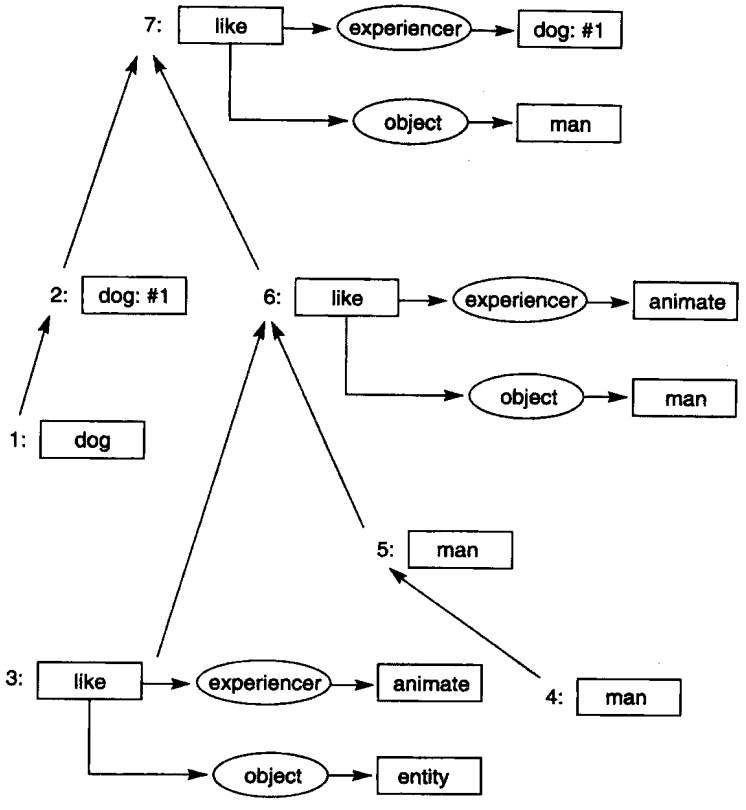


图 15-13　从图 15-10 的解析树构建语义表示

5）因为冠词是定冠词，noun_phrase 绑定一个单独的标记到概念（2），并将这个概念返回给 sentence。

6）sentence 调用 verb_phrase。

7）verb_phrase 调用 verb，找到表示 like（3）的案例框架。

8）verb_phrase 调用 noun_phrase，noun_phrase 接着调用 noun 找到表示 man（4）的概念。

9）因为冠词是不定冠词，noun_phrase 让这个概念保留泛指（5）。

10）verb_phrase 过程限制案例框架中的 entity 概念，并将其与表示 man（6）的概念连接。然后将这个结构返回给 sentence。

11）sentence 将概念 dog: #1 连接到案例框架（7）的 experiencer 结点。

这个概念图表示句子的意义。

语言生成是自然语言理解程序涉及的一个相关问题。英语句子的生成要求从意义的内部表示构建一个语义正确的输出。例如，agent 关系指明了两个概念之间的主—动关系。简单的方法是将适当的单词插入到存储的句子*模板*中。这些模板是句子和片断的模式，例如名词短语和介词短语。输出是通过在概念图中移动并组合这些片断来构建的。更加复杂的语言生成方法使用转换文法将意义映射到一个可用句子范围（Winograd 1972，Allen 1987）。

在本书补充材料中我们用 Prolog 建立了一个完整的递归下降的语义网解析器，解析器使用诸如 join、restrict 这样的图操作符来约束附加于解析树的叶结点上的语义网的表示。

15.5 节将说明程序如何为语言现象建立内部表示。这种表示在程序中有很多使用方式，这取决于具体的应用是什么。在这里我们介绍几个应用。但是在 15.4 节，首先介绍获取语言模式和规律性的随机方法。

15.4　语言理解的随机工具

在 15.1 节，讨论了支持语言理解的一种很自然的句子结构分解方法。在 15.2 节和 15.3 节，我们注意到语言的语义和知识密集特征能够通过单词和句子层次上的表示结构来实现。本节我们介绍在这些层次上支持结构化语言理解的有关模式分析的随机工具。

15.4.1　概述：语言分析中的统计技术

统计语言技术是当我们将自然语言看作随机过程时出现的方法。在日常谈话中，随机性意味着缺乏结构、定义或理解。然而，将自然语言看作随机过程概括了确定性观点。也就是说，统计（或随机）技术既能够对语言中明确定义的部分进行精确建模，也能够对确实有一些随机性的部分进行精确建模。

我们通过将语言看作随机过程，用严格的数学方法来重新定义自然语言理解中的许多基本问题。可以做一个有趣的实验，找几个句子，例如包含句号和圆括号的上一段，然后用一个随机数发生器将这些词排序并打印出来。得到的结果会没有任何含义。需要注意的是（Jurafsky and Martin 2008），这种基于模式的约束在语言分析的许多层面上都起作用，包括声音模式、语音合成、文法结构分析，等等。作为一个使用随机工具的例子，我们来思考词性标注的问题。

许多人从文法课上学到了这个问题。我们希望将句子中的每个词标记为名词、动词、介词、形容词，等等。另外，如果一个词是动词，我们希望知道它是主动的、被动的、及物的或者不及物的。如果一个词是名词，它是单数还是复数，等等。遇到类似“swing”的单词时出现了困难。如果我们说“front porch swing”，swing 是名词；但如果我们说“swing at the ball”，那么 swing 是动词。下面我们引用了毕加索的一句话，并给出了正确的词性标注：

Art	is	a	lie	that	lets	us	see	the	truth
名词	动词	冠词	名词	代词	动词	代词	动词	惯词	名词

为了开始分析，我们首先形式化定义这个问题。我们有语言中单词的集合 $S_w = \{w_1, \cdots, w_n\}$，例如 {a, aardvark, …, zygote}，以及词性或标记集合 $S_t = \{t_1, \cdots, t_m\}$。有 n 个单词的句子是 n 个随机变量的序列 $W_1, W_2, \cdots, W_n$。因为它们有可能取 S_w 中的任意值，所以称它们是随机变量。标记 $T_1, T_2, \cdots, T_n$ 也是一个随机变量序列。T_i 的取值记为 t_i，W_i 的取值记为 w_i。给定句子中的单词，我们希望找到这些标记最可能的取值序列。从形式上，我们希望选取 $t_1, \cdots, t_n$ 以最大化：

$$P(T_1 = t_1, \cdots, T_n = t_n \mid W_1 = w_1, \cdots, W_n = w_n)$$

回忆 5.2 节，$P(X|Y)$ 代表给定 Y 后 X 的概率。通常省略随机变量，记为：

$$P(t_1, \cdots, t_n \mid w_1, \cdots, w_n) \qquad \text{（公式 1）}$$

注意，如果我们确切知道这种概率分布，并且有足够的时间在所有可能标记集合上最大化取值，我们就总能够得到所考虑单词的最可能标记集合。此外，如果每个句子确实只有一个正确标记序列，这是文法教师可能赞成的一个想法，那么这种概率技术将总能找到正确的序列！这样，正确序列的概率就是 1，其他序列的概率就是 0。这就是我们说的统计观点能概括确定性观点时的

含义。

实际中，由于存储空间、数据和时间的限制，我们无法使用这一技术，必须采用某种近似。本节剩下的部分探讨近似公式 1 的渐优的方法。

首先我们将公式 1 重写为更有用的形式：

$$p(t_1, ..., t_n \mid w_1, ..., w_n) = p(t_1, ..., t_n, w_1, ..., w_n) / p(w_1, ..., w_n)$$

因为我们通过选择 $t_1, \cdots, t_n$ 来最大化其取值，所以可以将公式 1 化简为：

$$p(t_1, ..., t_n, w_1, ..., w_n) =$$

$$p(t_1)p(w_1 \mid t_1)p(t_2 \mid t_1, w_1) ... p(t_n \mid w_1, ..., w_n, t_1, ..., t_{n-1}) =$$

$$\prod_{i=1}^{n} p(t_i \mid t_1, ..., t_{i-1}, w_1, ..., w_{i-1})\, p(w_i \mid t_1, ..., t_{i-1}, w_1, ..., w_{i-1}) \qquad \text{（公式 2）}$$

注意，公式 2 等价于公式 1。

15.4.2　马尔可夫模型方法

实际上，通过我们在前面 9.3 节的讨论中可以看到，我们一般采用综合方法来最大化概率的公式（如公式 2），其概率以许多其他随机变量为条件。这样做有三个原因：首先，很难存储以许多其他随机变量为条件的随机变量的概率，因为可能概率的数量随着条件变量的个数呈指数增长。其次，即使我们能够存储所有概率值，通常也很难评估它们的值。经验上评估通常靠统计一个事件在手工标记的训练集中出现的次数来完成，因此若一个事件在训练集中仅出现几次，那么我们将不会得到其概率的好的评估。也就是说，评估 P(cat | the) 比 P(cat | The dog chased the) 容易，因为后者在训练集出现的次数少。最后，寻找最大化类似公式 2 结构的标记串会耗费太多时间，下面将会说明。

首先，我们需要对公式 2 做一些有用的近似。第一个粗略的近似为：

$P(t_i \mid t_1, \cdots, t_{i-1}, w_1, \cdots, w_{i-1})$ 近似为 $P(t_i \mid t_{i-1})$

及

$P(w_i \mid t_1, \cdots, t_{i-1}, w_1, \cdots, w_{i-1})$ 近似为 $P(w_i \mid t_i)$

这些是一阶马尔可夫假设（见 9.3 节），这种假设认为当前所考虑的事情仅依赖于其直接后继（proceeding），即与很久以前发生的事情无关。

将这些近似代回到公式 2 中，我们得到

$$\prod_{i=1}^{n} p(t_i \mid t_{i-1})\, p(w_i \mid t_i) \qquad \text{（公式 3）}$$

我们能够直接应用公式 3，因为其概率能够很容易地评估和存储。记住公式 3 只是 $P(t_1, \cdots, t_n \mid w_1, \cdots, w_n)$ 的一个估计，我们仍然需要通过选择标记（也就是 $t_1, \cdots, t_n$）来最大化其取值。幸运的是，有一个叫做 Viterbi 算法（Viterbi 1967，Forney 1973；动态规划见 4.1 和 15.2 节，Viterbi 算法见 10.3 节）的动态规划算法使我们能完成此工作。Viterbi 算法分别计算句子中每个单词 t_2 个标记序列的概率，其中 t 是可能标记的个数。为了详细介绍步骤，要标记的句子具有下面的形式：

```
article article    {best tail}
article verb       {best tail}
...
article noun       {best tail}
...
...
noun article       {best tail}
...
noun  noun         {best tail}
```

其中，{best tail}是动态发现的最后 n－2 个单词对应于给定 n－1 个标记的最可能的标记序列。

标记数 n－1 和标记数 n 的每个可能值在表中都有一个入口（因此我们得到 t_2 个标记序列）。在每一步，算法找到最大概率并为每个最佳末尾序列添加一个标记。这个算法保证能找到使公式 3 最大的标记序列，并且在 $O(t^2s)$ 时间内返回，其中 t 是标记个数，s 是句子中单词的个数。如果 $P(t_i)$ 以前面的 n 个标签为条件而不是两个，Viterbi 算法将耗费 $O(t^ns)$。因此，我们可以看出建立在过多既往变量上的条件会增加找到最大值的时间耗费。

幸运的是，公式 3 中使用的近似效果很好。当使用大约 200 个可能标记和大的训练集来评估概率时，使用这种方法的标记程序具有约 97% 的准确率，达到了人的准确率。马尔可夫近似令人惊讶的准确性和简单性使其在许多应用中非常有用。例如，大多数语音识别系统使用所谓的*三字母组模型*为系统提供某些“文法知识”来预测用户说的单词。三字母组模型是个简单的马尔可夫模型，根据前面的两个单词评估当前单词的概率。它使用 Viterbi 算法和刚刚讨论过的其他技术。要详细研究这个模型和相关技术，请参阅 Jurasky 和 Martin（2000）。

15.4.3　决策树方法

马尔可夫方法的明显的问题在于它只考虑了局部上下文。如果不是用词性标记单词，而是希望完成类似确定发起者和对象或者确定动词是主动还是被动的工作，那么需要更多的上下文。下面的句子说明了这一问题：

The policy announced in December by the President guarantees lower taxes.

实际上，President 是发起者，但是使用马尔可夫模型的程序很可能会将 policy 确定为发起者，将 announced 确定为主动动词。我们可以设想，如果程序能够提问，如“当前的名词是无生命的吗?”或“单词 by 是否出现在当下考虑的名词之前的几个词的位置?”，那么程序将在选择这种类型句子的发起者的概率统计方面智能很多。

回忆一下，标记问题等价于最大化公式 2，即

$$\prod_{i=1}^{n} p(t_i \mid t_1, \cdots, t_{i-1}, w_1, \cdots, w_{i-1})\, p(w_i \mid t_1, \cdots, t_{i-1}, w_1, \cdots, w_{i-1}).$$

理论上讲，更大上下文语境将包括简单地寻找对这些概率的更好的估计。这意味着我们可以使用上述文法问题的答案来精炼概率。

有几种方法可以解决这个问题。首先，我们可以将马尔可夫方法和本章前三节中介绍的解析技术结合起来。第二种方法利用 ID3 算法（在 10.3 节详细介绍，在补充材料中建立了 LISP 描述）或一些等价算法来发现以是或否（yes 或 no）问题为条件的概率。ID3 树的另一个优点在于，在一个很大的可能问题集合中，它们会只选择那些对精炼概率估计有用的问题。对于解析一类的更复杂的自然语言处理任务，基于 ID3 的树通常比马尔可夫模型更可取。下面我们说明如何

用 ID3 构建用于解析的决策树。

记得上一节我们问了一个问题："当前的名词是无生命的吗?"只有当我们知道哪些词是有生命的，哪些词是无生命的，才能问这样的问题。实际上，有一种自动技术可以为我们对单词进行分类，这种技术叫做*交互信息聚类*。两个随机变量 X 和 Y 之间分享的交互信息定义如下：

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)}$$

要在单词词汇表上做交互信息聚类，从将词汇表中的每个单词分别放入一个不同的集合开始。每一步，我们用双字母组合计算集合之间的平均交互信息，双字母组合是一个最近单词模型，然后选择两个集合合并，使得所有类的平均交互信息丢失最小。

例如，初始时我们有单词 cat、kitten、run 和 green，算法的第一步，我们有以下集合：

{cat} {kitten} {run} {green}

单词 cat 后面的第一个单词的概率与 kitten 后面第一个单词的概率很可能近似相等。换句话说：

P（eats | cat）大致与 P（eats | kitten）相同

P（meows | cat）大致与 P（meows | kitten）相同

这样，若令 X1、X2、Y1 和 Y2 为如下的随机变量：

X1 = {{cat}, {kitten}, {run}, {green}}

Y1 = X1 后跟的单词

X2 = {{cat, kitten}, {run}, {green}}

Y2 = X2 后跟的单词

那么 X2 和 Y2 之间的交互信息和 X1 和 Y1 之间的交互信息相比，少得不多，因此 cat 和 kitten 将很可能被合并。如果继续这个过程直到合并了所有可能的类，我们就得到一棵二叉树。然后，根据树内分支用*位码*给单词赋值，此树可以到达含有单词的叶结点。这反映了单词的语义含义。例如：

cat = 01100011

kitten = 01100010

此外，我们会发现，"类似名词"是最左边一位是 1 的所有单词，而那些极有可能表示无生命物体的单词可能是那些第 3 位为 1 的单词。

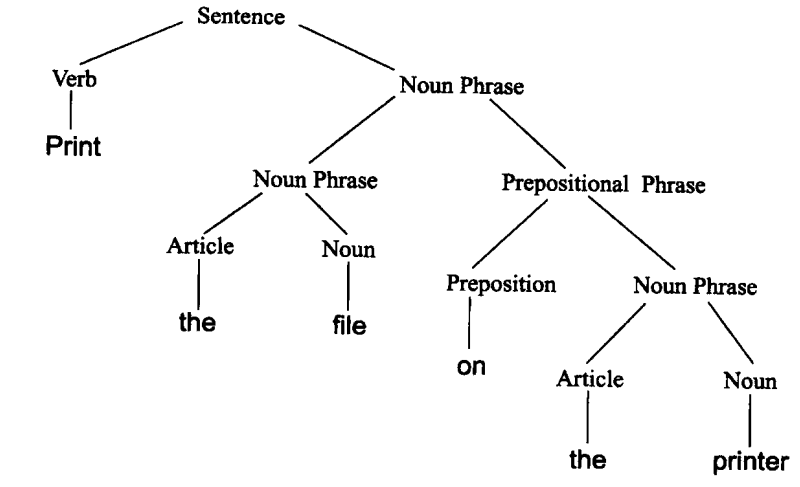
这种新的字典单词编码可以让解析器的提问更加有效。需要注意的是，该聚类没有考虑上下文，以至于即使"book"会被聚类为"类似名词"单词，我们也希望模型在短语"book a flight"中发现它时能将其标记为动词。

15.4.4　解析的概率方法

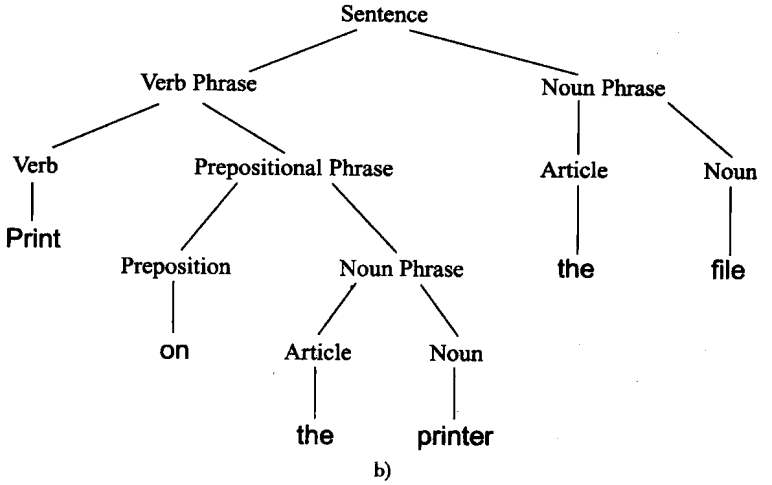
随机技术已经应用于计算语言学的许多领域中，而且仍然有大量的机会可以将它们应用到传统符号方法无法解决的领域。

在解析中使用统计方法是由模糊性问题引起的。模糊性源于以下事实，给定一个句子常常有几个可能的解析，我们需要选择哪个分析可能是最好的。例如，句子 Print the file on the printer 用图 15-14 中介绍的两棵树都可以分析。

在这种情况下，单独用文法规则不足以选出正确的解析。在 Print the file on the printer 例子



a)



b)

图 15-14　“Print the file on the printer” 的两种不同解析

中，我们需要考虑一些关于上下文和语义的信息。实际上，随机技术在解析领域中的最初使用是帮助解决模糊性。在这个例子中，我们可以使用词性标记中使用的相同的工具，ID3 算法。ID3 基于关于句子的语义问题帮助我们预测一个解析正确的概率。在句子中有一些句法模糊性的情况下，我们能够选出具有最高正确概率的解析。同样地，这种技术需要一个带有正确解析的句子的大的训练语料库。

最近，统计自然语言建模领域的人们变得更加雄心勃勃，试图用没有文法的统计技术进行解析。虽然无文法解析的细节超出了本书的范围，但是可以肯定的是，它和模式识别的关系更密切，而不仅仅与本章开始所介绍的传统的解析技术相关。

无文法解析已取得了一定的成功。在传统的基于文法的解析器和无文法解析器针对同一个句子集进行解析的性能比较实验中，基于文法的解析器使用了常用的交叉括号（cross-bracket）度量，获得了 69% 的比分，而无文法解析器获得了 78% 的比分（Magerman 1994）。这样的差异虽不十分突出，但是很有意义。更重要的是，传统解析器中的文法是由训练有素的语言学家花了大约十年才精心开发出来的，而无文法解析器本质上没有使用硬编码的语言信息，只是使用精

致的数学模型从训练数据中推断出需要的信息。要详细了解无文法解析及相关问题，请参阅 Manning 和 Schutze（1999）。

另外三个领域分别是语音理解、语音转化为文本以及手写体的识别。这些领域同样具有很长的使用随机方法为语言建模的历史。这三个领域中最通用的统计方法是预测下一个单词的三字母组模型。该模型的优点在于其简单性：在前面两个单词的基础上预测下一个单词。最近，有一些统计语言领域的工作，在结合了文法约束和更长距离依赖的同时，保持了该模型的简单性和易用性。这种新方法使用所谓的*文法三字母组*。文法三字母组由单词对之间的基本关系（即主语－动词、冠词－名词和动词－对象）构成。这些关系集合起来称作*链接文法*。链接文法比语言学家使用的传统文法简单和容易创建得多，并且与概率方法配合得很好。

Berger 等（1994）描述了一个统计程序：Candide。它可以将法语文本翻译为英语文本。Candide 同时使用统计和信息理论来开发翻译过程的概率模型。它只需在一个很大的法语和英语句子对的语料库上训练，其结果与商用的翻译程序 Systran（Berger et al. 1994）的结果相当甚至在某些情况下还更好。特别有意义的是，Candide 系统在翻译过程中没有使用传统的解析，而是使用了刚才提到的文法三字母组和链接文法。

15.4.5　概率上下文无关解析器

本节阐述两种不同的概率上下文无关解析器，一种是基于结构的解析器，另一种是于词汇化的解析器。基于结构的解析器对每一个出现的文法解析规则进行概率计算，其概率是该规则与其组成元素的概率联合出现的函数。

我们扩展 15.2 节所述的上下文无关的文法规则集，计算规则的概率值，阐明*基于结构的概率解析器*。

1. sentence ↔ noun_phrase verb_phrase p(s) = p(r1) p(np) p(vp)
2. noun_phrase ↔ noun p(np) = p(r2) p(noun)
3. noun_phrase ↔ article noun p(np) = p(r3) p(article) p(noun)
4. verb_phrase ↔ verb p(vp) = p(r4) p(verb)
5. verb_phrase ↔ verb noun_phrase p(vp) = p(r5) p(verb) p(np)
6. article ↔ a p(article) = p(a)
7. article ↔ the p(article) = p(the)
8. noun ↔ man p(noun) = p(man)
9. noun ↔ dog p(noun) = p(dog)
10. verb ↔ likes p(noun) = p(likes)
11. verb ↔ bites p(noun) = p(bites)

单个单词的概率值能够通过这些单词在英语句子特定语料库中出现的概率得到。每条文法规则的概率值 $P(r_i)$ 由该文法规则在该种语言语料库句子中出现的频率决定。这里给出的例子非常简单，这个例子只是用以说明概率上下文无关解析器的基本特征。

为了说明第二种解析器：*概率词汇化上下文无关解析器*，我们继续扩展 15.2 节中的文法规则。在这种情形下我们考虑一条句子的各个方面。首先考虑语言使用中的双字母组或三字母组概率，然后我们依赖适宜的语言语料库对双字母组或三字母组的概率进行计算。与先前所举的例子类似，我们继续从语言语料库出发，依据每条解析规则的出现频率，计算概率值。最后计算特定名词和动词组合的概率值。例如，如果主语名词是 dog，我们感兴趣的是谓语动词是 bites 的概率。值得注意的是即使从语言语料库出发，概率的计算也仍然会加强名词－动词形式的一致性。

由于我们关注的是共现单词的特殊模式的概率，我们称这种解析器为词汇化解析器。为了简便起见，我们只讨论双字母组下的解析器：

1. sentence ↔ noun_phrase(noun) verb_phrase(verb)
   p(sentence) = p(r1) p(np) p(vp) p(verb | noun)
2. noun_phrase ↔ noun
   p(np) = p(r2) p(noun)
3. noun_phrase ↔ article noun
   p(np) = p(r3) p(article) p(noun) p(noun | article)
4. verb_phrase ↔ verb
   p(vp) = p(r4) p(verb)
5. verb_phrase ↔ verb noun_phrase
   p(vp) = p(r5) p(verb) p(noun_phrase(noun)) p(noun |verb)
6. article ↔ a p(article) = p(a)
7. article ↔ the p(article) = p(the)
8. noun ↔ man p(noun) = p(man)

p(man | a)
p(man | the)
etc.
p(likes | man)
etc.

本章的补充材料中给出了构建本节所述的两种概率解析器的 Prolog 代码。

在许多其他领域中，严格的随机语言建模技术尚未得到采用，但是这种技术有可能产生有用的结果。信息提取问题或者是从手写文本中获取一定数量的具体信息问题均是该技术的潜在应用领域。再比如口语语言中，语调匹配模式将与特定的知识库概念相关。其他例子还包括语义网的搜索。关于自然语言处理的随机方法的进一步细节，可以参阅 Jurafsky 和 Martin（2008）以及 Manning 和 Schutze（1999）的论著。

最后，我们给出几个计算语言使用的例子。

15.5　自然语言应用

15.5.1　故事理解和问题解答

对于自然语言理解技术可以进行如下有趣的测试：编写一个程序，该程序能够阅读故事或其他自然语言文本材料，并能回答相关的问题。在第 7 章，我们讨论了故事理解中所涉及的一些表示问题，包括将文本的精确内容与背景知识相结合的重要性。如图 15-2 所示，一个程序可以通过如下方式完成这一工作：在输入的语义解释和知识库中的概念图结构之间进行网络连接。更复杂的表示法（如 7.1.4 节中的脚本）能够对过去出现的事件等更复杂的情况进行建模。

一旦程序建立了文本的扩充表示，就能够智能地回答有关它所阅读的内容的问题。程序将问题解析为一个内部表示，并将询问与故事的扩充表示进行匹配。考虑图 15-2 的例子。程序阅读了句子“Tarzan kissed Jane”，并建立了一个扩充表示。

假设问程序“Who loves Jane?”。在问题的解析中，疑问词 who、what、why 等指出了问题的意图。who 询问动作的发起者；what 询问动作的对象；how 询问执行动作的方法，等等。图 15-15 给出了由问题“Who loves Jane?”生成的图。图的发起者结点上标记了一个?，它指出了问题的意图所在。而后这个结构与原始文本的扩充表示相结合。在询问图中，与 person: ?绑定的概

念就是问题的答案："Tarzan loves Jane"。

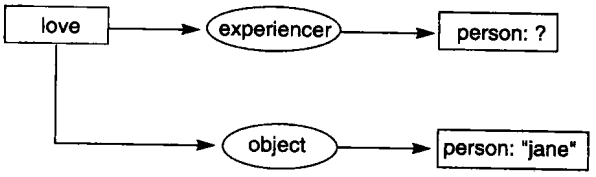


图 15-15　问题"Who loves Jane?"的概念图

15.5.2　数据库前端

设计自然语言理解程序的主要瓶颈是如何获取有关应用领域的足够的知识。现有的技术适用于明确定义了语义的狭窄领域。满足这些标准的一个应用领域是为数据库开发自然语言前端。虽然数据库保存了大量信息，但是这些信息具有高度规则性并且范围很窄；此外，数据库语义定义比较明确。由于具有这些特征，再加上数据库能够接受自然语言查询的特点，使得数据库前端成为自然语言理解技术的一个重要应用。

数据库前端的任务是将自然语言问题翻译成数据库语言格式的查询。例如，用 SQL 数据库语言作为目标语言（Ullman 1982），自然语言前端会将问题"Who hired John Smith?"翻译为查询：

```
SELECT MANAGER
FROM MANAGER_OF_HIRE
WHERE EMPLOYEE = ' John Smith'
```

在翻译的执行过程中，程序不仅要翻译原始查询；还必须决定从数据库中什么地方查询（MANAGER_OF_HIRE 关系），要访问字段的名称（MANAGER），以及查询的约束（EMPLOYEE = ' John Smith'）。原始问题并不包含这些信息，这些信息可以在知识库中找到。知识库提供了数据库的组织形式以及可能的问题所具备的含义。

关系数据库按实体领域间的关系组织数据。例如，假设我们正在创建一个雇员数据库，希望能访问每个雇员的工资以及雇佣她的经理。这个数据库包含三个域，或者说实体集合：经理集、雇员集和工资集。我们可以将这些数据组织为两个关系，关联雇员及其工资的 employee_salary 和关联雇员及其经理的 manager_of_hire。在关系数据库中，关系通常显示为列举关系实例的表。表的列通常具有名字，这些名字称为关系的属性。图 15-16 显示了关系 employee_salary 和 manager_of_hire 的表。manager_of_hire 有两个属性：employee 和 manager。关系的值是雇员和经理的二元组。

manager_of_hire:

employee	manager
John Smith	Jane Martinez
Alex Barrero	Ed Angel
Don Morrison	Jane Martinez
Jan Claus	Ed Angel
Anne Cable	Bob Veroff

employee_salary:

employee	salary
John Smith	$35,000.00
Alex Barrero	$42,000.00
Don Morrison	$50,000.00
Jan Claus	$40,000.00
Anne Cable	$45,000.00

图 15-16　雇员数据库中的两个关系

如果我们假定雇员有惟一的名字、经理和工资，那么雇员名字可以作为工资和经理属性的主键。一个属性是另一个属性的主键，如果它惟一确定另外一个属性的元素取值。一个合法的查询指明一个目标属性并定义一个值或约束集合；数据库返回目标属性的指定值。我们可以用很多种图形化方法指明主键和其他属性之间的关系，包括实体关系图（Ullman 1982）和数据流图（Sowa 1984）。这两个方法都使用有向图显示主键到属性的映射。

我们可以扩展概念图来包含这些关系图（Sowa 1984）。定义映射的数据库关系用菱形表示，并标注有关系的名字。关系的属性在概念图中表示为概念，箭头的方向表示主键到其他属性的映射。employee_salary 关系和 manager_of_hire 关系的实体关系图见图 15-17。

在将英语翻译为标准查询的过程中，我们必须确定包含答案的记录，要返回的记录的字段，以及决定该字段的主键的值。我们不是直接从英语翻译为数据库语言，而是先翻译为一种更有表现力的语言，如概念图。这一步骤是必要的，因为许多英语询问是模糊的，需要额外的解释才能生成标准格式的数据库查询。更有表现力语言的使用将有助于这一过程的完成。

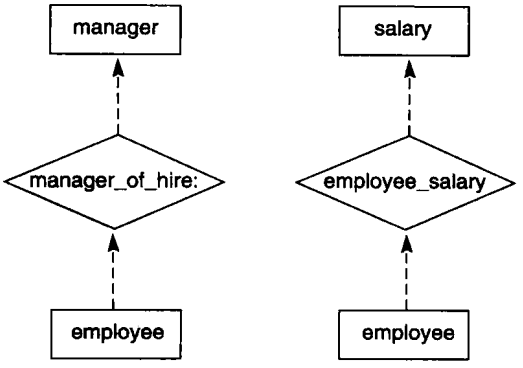


图 15-17　manager_of_hire 与 employee_salary 的实体关系图

如本章前面所描述，自然语言前端将询问解析并解释为一个概念图。然后使用联结和限制操作将这个图与知识库中的信息结合起来。在本例中，我们要处理类似“Who hired John Smith?”或“How much does John Smith earn?”这样的询问。对于每个潜在的询问，我们保存一个定义其动词、动词的事例角色的图以及与问题相关的任意实体关系图。图 15-18 显示了“hire”询问的知识库条目。

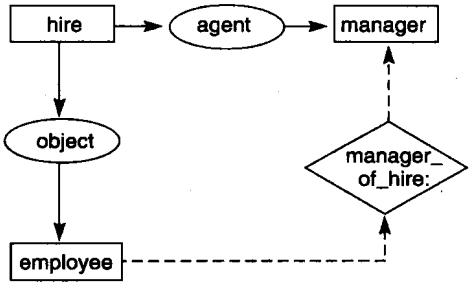


图 15-18　“hire”询问的知识库条目

语义解释器生成一个用户询问的图，并将该图与适当的知识库条目联结。如果有一个附加的实体关系图将主键映射到问题目标，程序就可以使用这个实体关系图形成一个数据库查询。图 15-19 显示了问题“Who hired John Smith?”的询问图，以及将其与图 15-18 中知识库实体联结后得到的结果。它还显示了由该图形成的 SQL 查询。注意正确记录的名字、目标字段和查询的主键在自然语言询问中没有定义。这些都从知识库中推出。

在图 15-18 中，只知道原询问的 agent 和 object 具有类型 person。要将它们与知识库实体 hire 联结，必须首先将它们分别限制为 manager 和 employee 类型。这样类型层次就能够用来对原询问执行类型检查。如果 john smith 不具有类型 employee，那么问题就是非法的，程序也能够探测到。

一旦建立了扩展询问图，程序就检查目标概念，打上?标记，并且决定将一个主键映射到概念的 manager_of_hire 关系。因为主键绑定到值 john smith，所以问题是合法的，程序能形成正确的数据库查询。将实体关系图翻译为 SQL 或其他语言是直接明了的。

虽然这个例子很简单，但是它说明了如何使用基于知识的方法来建立自然语言数据库前端。我们例子中的概念用概念图表示，也可以映射到其他表示法，如框架或基于谓词逻辑的形式方法。

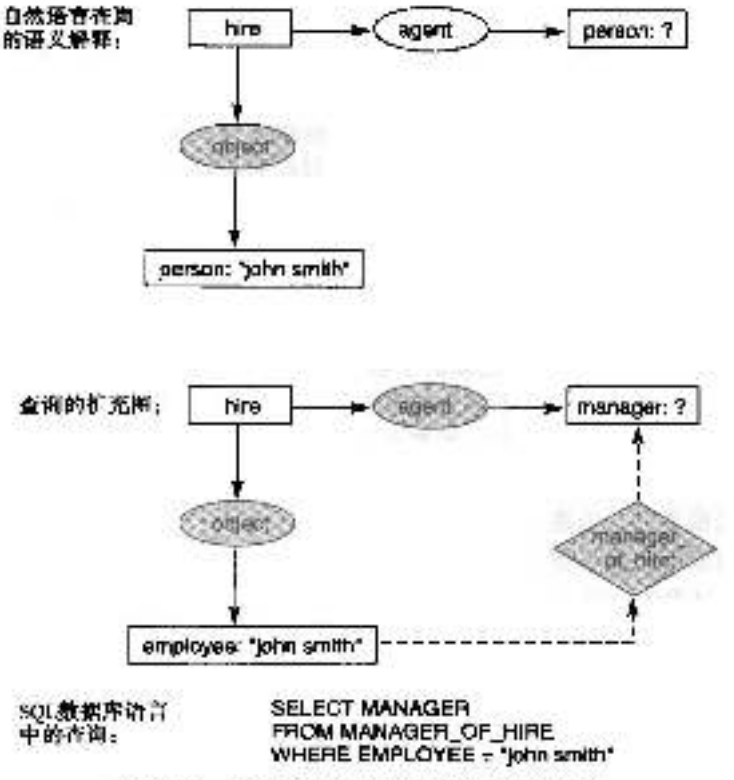


图 15-19　从自然语言输入的获得出数据库查询

15.5.3　Web 信息抽取和摘要系统

万维网既为人工智能和自然语言理解研究带来了激动人心的挑战，也带来了饶有趣味的机遇，其中最有意义的工作是开发这样的智能软件，该软件能够概括出“有兴趣的”基于 Web 的资料。这个问题实际上分为两部分：在网页上定位可能成为兴趣点的信息所在的位置以及提取或挖掘这些感兴趣信息的关键要素。尽管定位有趣信息的第一个问题是关键的（第 10 章、第 13 章和第 15 章的许多技术是很重要的），但是我们现在先考虑一个模板匹配和填充算法，以此为基础再讨论第二个问题。

在定位信息后，可能用关键词匹配或更复杂的搜索引擎，信息抽取系统以这些无任何限制的文本为输入，围绕一个预先定义的领域或感兴趣的主题来进行摘活。它发现关于该领域的有用信息，编码信息的格式使其适于向用户报告或填充到结构化数据库中。

与深入的自然语言理解系统相反，信息抽取系统浏览一个文本，发现有关部分，然后只集中处理这些部分。我们提出了一个基于模板的方法，该方法与许多研究者提出的信息抽取方法（Cardie 1997，Cardie and Mooney 1999）相似。我们的信息抽取方法的示意图如图 15-20 和图 15-21 所示。假如我们想要搜集并概括万维网上计算机科学的大学教员的信息，图 15-20 给出了目标工作，并且提出了信息模板，我们可以使用我们的软件从该信息模板上提取信息并进行相应的工作。

在早期的信息抽取尝试中，自然语言系统（natural language system）使用差别非常大的方法。在一个极端，系统使用传统的工具处理文本：对每个句子进行完整的语法分解，同时进行详细的语义分析，接着还常常进行话语级处理。在另一个极端，系统使用很少甚至没有知识的关键词匹

配技术或者语言级分析。然而，随着更多系统的建立和评估，这些极端方法的局限也越来越明显。图 15-20 和图 15-21 介绍了一个改编自 Cardie（1997）的更先进的信息抽取体系结构。虽然该结构在应用中的细节可能各不相同，但是图中指明了信息抽取过程中执行的主要功能。

计算机科学系招聘广告（摘录）样例：

The Department of Computer Science of the University of New Mexico. . . is conducting a search to fill two tenure-track positions. We are interested in hiring people with research interests in:

Software, including analysis, design, and development tools. . .
Systems, including architecture, compilers, networks. . .
...

Candidates must have completed a doctorate in. . .
The department has internationally recognized research programs in adaptive computation, artificial intelligence, . . . and enjoys strong research collaborations with the Santa Fe Institute and several national laboratories. ...

已部分填充的模板样例：

Employer: Department of Computer Science, University of New Mexico
Location City: Albuquerque
Location State: NM 87131
Job Description: Tenure track faculty
Job Qualifications: PhD in . . .
Skills Required: software, systems, . . .
Platform Experience: . . .
About the Employer: (text attached)

图 15-20　计算机科学广告的样本文本、模板摘要与信息抽取

1. 文本:	The Department of Computer Science of the University of New Mexico is conducting a search to fill two track track positions. We are interested in hiring ...
2. 符号化和标记:	The/det Department/noun of/prep ...
3. 句子分析:	Department/subj is conducting/verb search/obj ...
4. 抽取:	Employer: Department of Computer Science Job Description: Tenure track position ...
5. 合并:	tenure track position = faculty New Mexico = NM ...
6. 模板生成:	As in Figure 15.19

图 15-21　信息抽取的一个体系结构，引自 Cardie（1997）

首先，将“感兴趣”网站的每个句子符号化并标记。可以使用 15.4 节介绍的随机标记器。接着是句子分析阶段，执行解析得到名词组合、动词、介词短语和其他文法结构。接下来抽取阶段找到并标记与抽取主题相关的语义实体。在我们的例子中，这个阶段识别雇员姓名、地址、工作要求（学历、计算机应用技能，等等）、开始任职时间，等等。

抽取阶段是过程中第一个完全具体到领域的阶段。在抽取过程中，系统确定文本相关成分

之间的特定关系。在我们的例子中，计算机科学系看作是雇主，地址是新墨西哥大学。合并阶段必须处理类似同义词关联和指代分析之类的问题。同义词的例子是 tenure-track 和 faculty，以及 New Mexico 和 NM。指代分析将第一句中的 Department of Computer Science 和第二句中的主语 we 连结起来。

合并过程中所做的话语级推理协助模板生成过程，确定文本中不同关系的数量，将这些抽取出来的信息块映射到模板的每个字段，并生成最终的输出模板。

尽管最近取得了一些进步，但是目前的信息抽取系统仍然有许多问题。第一，这些系统的精确性和健壮性还可以有很大改进，因为抽取中的错误似乎源于对输入文本的含义（语义）相当肤浅的理解。第二，在一个新领域中建立信息抽取系统是困难而且费时的（Cardie 1997）。这两个问题都与抽取任务的特定领域特性有关。如果将语言知识来源调整到特定的领域，那么信息抽取过程会有很大改善，但是手工更改领域特定的语言知识既困难，又容易出错。

虽然如此，现在还是有很多有意义的应用。Glasgow 等（1997）建立了一个系统，支持保险商在人寿保险分析中的应用。Soderland 等（1995）建立了一个系统，从用于保险处理的病人医疗记录中抽取症状、身体检查、化验结果和诊断。还有一些分析新闻发现并归纳合资商业企业风险的程序（MUC-5 1994），一些自动分类法律文件的系统（Holowczak and Adam 1997），以及一些从计算机求职列表中抽取详细信息的程序（Nahm and Mooney 2000）。

15.5.4　用学习算法来泛化抽取的信息

另外一个应用是将本章介绍的许多思想与机器学习算法（见 10.3 节）结合起来。Cardie 和 Mooney（1999）以及 Nahm 和 Mooney（2000）提出从文本中抽取出的信息可以用机器学习算法泛化，并将结果重用于信息抽取任务。

方法是简单明了的。从适当的 Web 站点收集到完全甚至部分填充的文本摘要模板，例如图 15-20 中所示的。然后将得到的模板信息存储到关系数据库中，用 ID3 或 C4.5 之类的学习算法抽取出决策树，决策树能反映出隐藏在数据集中的规则关系，见 9.3 节（这种技术称为*数据挖掘*）。

Mooney 和他的同事提出这些新发现的关系应该用来精炼信息抽取中原来使用的模板和知识结构。从 15.5.3 节的计算机科学求职应用分析中发现的这类信息的例子可能包括：如果职位是计算机科学系教员，那么不需要特定的计算平台经验；如果大学正在聘任教员，那么需要研究经验，等等。更详细的内容可以参阅 Nahm 和 Mooney（2000）。

在目前的计算语言学中有很多专题属于我们要研究的目标。我们归纳为三个方面。第一，理解口语的关键单元是什么？已经有很多研究者尝试去回答该问题，他们通过分析英语句子，识别单个单词，关注个人语音。关注音节是解决人类语言理解的关键吗？我们在听到一个单词到反应出来之间到底需要多长时间？最优的分析语速到底是多少（Greenberg 1999）？

第二，基于声音识别来接受知识库中的概念可以进一步为语言的翻译提供条件吗？假如音节就是语言分析的最有用的单元，假如双音节对说话者的单词是有用的，那么这些单词能映射到知识库中的概念中吗？相关的知识能用来提高对接下来音节的理解吗？一个基于计算机的旅游组织程序的调用程序可以识别城市名称的音节，然后计算机预测与浏览该城市相关的问题。

第三，自然语言处理的长期目标就是创建语义网；这是怎么产生的？或者说这会产生吗？这里有计算机可以“理解”的任何方法论的东西吗？或者说还一直是那些基于搜索的模式驱动的“小技巧”，这些非常好的“小技巧”给人造成了计算机理解的假象（又一个图灵机测试）？现在人类做得是否“足够好”？最后一章将会继续讨论这些以及相关的问题。

15.6　结语和参考文献

正如本章中所提到的，自然语言中有许多定义文法和解析句子的方法。作为这些方法的典型例子，我们介绍了 ATN 解析器和马尔可夫模型。读者应该能注意到其他可能的方法，包括*转换文法*、*语义文法*、*案例文法*和*特征与功能文法*（Winograd 1983，Allen 1995，Jurafsky and Martin 2008）。

转换文法使用上下文无关规则表示句子的*深层结构*，或者说含义。这种深层结构可以表示为一棵解析树，不仅包含终结符和非终结符，还包括一个称作*文法标记*的符号集合。这些文法标记表示语言结构中如单复数、时态和其他上下文相关概念之类的特征。而且，称为转换规则的一个高层规则集合在深层结构和*表层结构*之间转化，表层结构更接近于句子的实际形式。例如，“Tom likes Jane” 和 “Jane is liked by Tom” 有相同的深层结构但不同的表层结构。

转换规则作用于解析树，执行需要全局上下文的检查，并生成合适的表层结构。例如，一条转换规则可以检查代表句子主语的结点的单复数特征与动词结点的单复数特征是否相同。转换规则也可以把单个深层结构映射到多个表层结构，如将主动语气变为被动或者肯定变为疑问。虽然本文中不论述转换文法，但是它们是扩张短语结构文法的一个重要选择。

Terry Winograd（1983）在《Language as a Cognitive Process》中提出了一种文法和解析的综合处理方法。该书提出了对转换文法的一个非常彻底的处理。James Allen（1987，1995）写的《Natural Language Understanding》对自然语言理解程序的设计和实现做了一个综述。Mary Dee Harris（1985）的《Introduction to Natural Language Processing》是另一本综述自然语言理解的书，并扩充了本章中遇到的问题。我们还推荐 Gerald Gazdar 和 Chris Mellish（1989）的《Natural Language Processing in Prolog》。Charniak（1993）和 Charniak 等（1993）介绍了语言和语音片段标记的随机方法中的问题。

自然语言的语义分析包括知识表示（见第 7 章）中提到的许多难题。在 Charniak 和 Wilks（1976）的《Computational Semantics》中对这些问题进行了讨论。由于在对自然语言交互需要的知识和社会环境建模时的困难，许多作者对将这一技术应用于不受限领域的可能性提出了质疑。对话语分析的早期研究可以在 Linde（1974）、Grosz（1977）和 Grosz 和 Sidner（1990）中找到。Winograd 和 Flores（1986）的《Understanding Computers and Cognition》、John Searle（1980）的《Minds，Brains，and Programs》以及 Smith（1996）的《On the Origin of Objects》都讨论了这些问题。

Schank 和 Riesbeck（1981）的《Inside Computer Understanding》讨论了使用概念依赖技术的自然语言理解。Schank 和 Abelson（1977）的《Scripts，Plans，Goals and Understanding》讨论了自然语言程序中高层知识组织结构的作用。

John Searle（1969）的《Speech Acts》讨论了对话语建模过程中语用和语境知识的作用。Fass 和 Wilks（1983）指出语义偏好理论（semantic preference theory）是自然语言语义建模的工具。语义偏好是案例文法的泛化，使得案例框架可以转化。这为语义表示提供了更大的灵活性，并可以表示像比喻和类比这样的概念。要完整了解乔姆斯基层次，可以参考 Hopcroft 和 Ullman（1979）。我们对概念图的处理从 John Sowa（1984）的《Conceptual Structures》中获益匪浅。

语言处理技术的近期介绍可以从下面的文章中找到：Jurafsky 和 Martin（2008）的《Speech and Language Processing》、Manning 和 Schutze（1999）的《Foundations of Statistical Natural Language Processing》、Cole（1997）的《Survey of the State of the Art in Human Language Technology》以及《AI Magazine》1997 冬季版。

现在有许多商业出版物关于基于计算机的语言理解、产生和基于语言的数据挖掘。尽管可以用这些产品做实验，但是它们的源代码一般不公开，所以这里我们不做讨论。

追踪自然语言理解研究趋势的最好的参考资料，从传统和随机两方面的观点，是 AI 会议每年的会议录：AAAI 和 IJCAI，由 AAAI 出版社经由 MIT 出版社出版；Association of Computational Linguistics 也有在美国和国际上每年举办的国际会议，另外还有《Journal of the Association for Computational Linguistics》。这些都是目前技术上重要的资源。

15.7　习题

1. 将下列句子分类为语法错误，语法正确但无意义，有意义但不正确，或者正确。理解过程中，在什么地方发现的这些问题？

   Colorless green ideas sleep furiously.

   Fruit flies like a banana.

   Dogs the bite man a.

   George Washington was the fifth president of the USA.

   This exercise is easy.

   I want to be under the sea in an octopus's garden in the shade.

2. 讨论理解下列句子必须的表示结构和知识。

   The brown dog ate the bone.

   Attach the large wheel to the axle with the hex nut.

   Mary watered the plants.

   The spirit is willing but the flesh is week.

   My kingdom for a horse!

3. 用 15.2.1 节的“dogs world”文法解析下面的句子。哪些是不合法的句子？为什么？

   The dog bites the dog.

   The big dog bites the man.

   Emma likes the boy.

   The man likes.

   Bite the man.

4. 扩充“dogs world”文法，使其能够包括练习 3 中不合法的句子。

5. 用 15.2.3 节的上下文相关文法解析下面的句子。

   The men like the dog.

   The dog bites the man.

6. 为下列句子生成解析树。你将不得不用更复杂的语言结构扩充我们简单的文法，如副词、形容词和介词短语。如果一个句子有多种解析，那么画出所有的解析树，并解释可以用来选择解析的语义信息。

   Time flies like an arrow but fruit flies like a banana.

   Tom gave the big, red book to Mary on Tuesday.

   Reasoning is an art and not a science.

   To err is human, to forgive divine.

7. 扩充“dogs world”文法，使得名词短语中包含形容词。要确信能包含不确定个数的形容词。提示：用一条递归规则 adjective_list，或者为空，或者包含一个形容词加上后面跟随的一个形容词列表。将这个文法映射到转移网络。

8. 将下列上下文无关文法规则添加到 15.2.1 节的“dogs world”文法中。将得到的文法映射到转移网络。

   sentence ↔ noun_phrase verb_phrase prepositional_phrase

   prepositional_phrase ↔ preposition noun_phrase

preposition ↔ with

preposition ↔ to

preposition ↔ on

9. 为带形容词（练习 7）和介词短语（练习 8）的“dogs world”文法定义一个 ATN 解析器。
10. 定义表示练习 9 中文法的意义所需的概念图中的概念和关系。定义从解析树建立语义表示的过程。
11. 扩充 15.2.3 节的上下文相关文法来检查主语和动词之间的语义一致。特别是，虽然狗可以喜欢或者咬人，但是人不应该咬狗。对 ATN 文法做相似的改进。
12. 扩充 15.2.4 节的 ATN 文法来包含 who 和 what 问句。
13. 描述如何将 15.4 节的马尔可夫模型与符号化方法结合来理解 15.1 到 15.3 节中的语言。
14. 扩充 15.5.2 节的数据库前端例子，使其可以回答“How much does Don Morrison earn?”形式的问题。你需要扩充文法、表示语言和知识库。
15. 将上一个问题中的单词，包括标点，以随机顺序排列。
16. 15.5.2 节的例子中，假设经理与其他雇员都列在 employee_salary 关系中。扩充该例子，使其能处理类似“Find any employee that earns more than his or her manager”的询问。
17. 15.4 节中的随机方法如何能与 15.5.2 节中的数据库分析技术结合起来。
18. 用随机方法在关系数据库中进行模式发现是目前研究的一个重要领域，有时称为*数据挖掘*（见 10.3 节）。如何用这种方法来回答询问，例如 15.5.2 节中提出的关于关系数据库的询问？
19. 为某些领域建立在万维网上使用的信息抽取系统。参考 15.5.3 节中的建议。
20. 在本书的补充材料中，把 Prolog 的上下文无关和上下文相关分析器的结构、形容词、副词、前置短语都转换为文法。
21. 在本书的补充材料中，把 Prolog 的随机上下文无关分析器结构、形容词、副词、前置短语都转换为文法。

第六部分 后 记

计算机科学的潜能如果能够被全部挖掘和开发出来，它将把我们带到一个更高的关于世界的知识平台。计算机科学将帮助我们对智能处理有更深入的理解，能增长我们在学习过程、思考过程和推理过程中的知识。计算机科学能够为认知科学提供模型化和概念化工具，正如物理学在几乎整个20世纪中统治了人类智能的努力一样。当时人们主要探索物理的本质和宇宙的起源，而现在我们正在开始探索意识、知识结构以及语言的智能宇宙。我预见那些将极大改变我们生活的重大进步将会持续不断地出现……我们能预见人们将能够理解如何去组织和管理知识……

——约翰·麦凯克洛夫特，ACM 图灵奖演说，1987

介意吗？无所谓。（What is mind? No matter.）

无所谓？不介意……（What is matter? Never mind…）

——霍斯·平克莱

知道什么并不重要，重要的是我们从知道的东西中学到了什么。

——厄尔·韦弗，前巴尔的摩金鹰队总教练和经理

对智能本质的思考

尽管本书讲述了人工智能的较大的哲学意蕴，但主要还是强调用来建立智能的基于计算机的人工制品的工程技术。在本书的最后部分，将再次讨论这些深层次的问题，讨论人工智能的哲学基础，重新评价使用人工智能技术来建立智能科学的可能性，并推测这门学科未来的发展进程。

正如在本书一直提到的，人类认知和问题求解方面的研究对人工智能理论和程序实践做出了巨大的贡献。反过来，人工智能方面的工作也为许多学科（包括生物学、语言学和认知心理学）在模型构造和实验测试方面给出了很好的引导。在总结我们所表述的观点时，我们想再采取这种折中的精神，并考虑如下这些问题：表达的局限性、现实具体化到智力过程的重要性，以及在知识的增长和解释中文化的作用。这些问题引导我们更深入地思考科学问题和哲学问题，比如那些有关对模型的歪曲以及科学方法自身的本质和能力的问题。我们的观察使得我们赞成采取一种跨学科的方法，将人工智能的研究与其他各学科的研究结合起来，这些学科包括心理学、语言学、生物学、人类学、认知学以及其他探索人类思维及其产物的学科。我们相信通过探索这些学科的交互与区别，能够很好地理解智能内部的过程，不管是基于生物的还是机械的。

传统上，人工智能的研究通常是基于物理符号系统假设（Newell and Simon 1976）。从这种观点出发的研究将不断地产生复杂的数据结构和搜索策略，它们依次将在两个方面导致许多重要的成功，一方面是构建一种能达到智能行为各元素的工具，另一方面是阐明那些能够理出人类智能的许多组件。不管怎样，需要重点指出的是，那些从较早的方法中形成的人工智能的实践活动还停留在来自于哲学理性主义的假设之上。正如理性主义传统所定义的，智能本身在很大程度上被看作是逻辑推理，科学问题求解以及一种直接的经验式的理解世界的过程，我们认为哲学理性主义已经过度限制了人工智能当前的方法及其探索研究的范围。

在本书中，给出了许多最近的研究与进展，包括学习的可选择模型、基于主体和分布式的问题求解、体现智能的方法，以及进化计算和人工生命等方面的进展。这些理解智能的方法为理性主义的简化论提出了很必要的选择。智能的生物和社会模型表明人类智能正是我们身体和感觉的产物，是我们文化和社会制度的产物，是我们创造和享受艺术的产物，是我们听到的和传递的历史的产物。建立多种方法来构建对这些复杂过程的计算机模拟，比如模拟在人脑中神经模式的进化和适应过程，最近的这些方法为人工智能提供了很多新的强有力的工具集，弥补了传统的技术。

正如计算机科学中的其他分支一样，人工智能还是一个年轻的领域。当物理学和生物学能在几个世纪中来检测其进展时，现代计算还只有几十年的历史。在第 16 章中，我们将把人工智能不同方面的成果整合到一种统一的智能系统的科学中来。我们指出，科学和工程、哲学以及我们自己的审美价值观将引导我们不断地创建新的人工制品和实验，如果使用得当，它将能引出关于更通用的智能系统科学的一些见解。这一章将继续第 1 章中进行的讨论，建立一种人工智能的认知基础，与其说是回答对它的批判（实际上，还有很多挑战等待着去解答），不如说是采用积极肯定的态度，去探索和照亮前进的道路。

第 16 章　人工智能是经验式的学科

计算机科学是一门经验式的学科。我们也可以把它称为是实验性科学，但像天文学、经济学、地质学等学科一样，其一些独特的观察和经历形式并不一定适合那些老一套的实验方法。尽管如此，它仍然是实验性的。构造出的每一台机器都是一个实验。事实上构造这台机器提出了一个问题；同时我们通过观察机器的运行并使用各种可用的分析和测试方法来获取机器返回的答案。编写的每一个新程序都是一个实验。编写新程序一方面提出了一个问题，同时其行为也为答案提供了线索。我们设计的机器和程序，分别作为硬件和软件，它们不是黑匣子，它们都是由人造出来的，而且我们可以打开它们，观察它们内部的东西。我们可以把它们的结构与行为联系在一起，并从一个简单的实验中获得很多经验和教训。

——A. 纽维尔和 H. A. 西蒙，ACM 图灵奖演说，1976

我们对思考机器的研究比通过自省的方法来了解我们自己的大脑，会教给我们更多有关人脑的知识。西方人正在以机械装置的形式来表现自己。

——威廉 S. 伯勒斯，《裸体午餐》

我们在信息中遗失的知识到哪去了？

——T. S. 艾略特，合唱剧《岩石》

16.0　简介

对许多人来说，人工智能研究中最使人惊讶的成果是证明人工智能以及实际上计算机科学的许多分支都是一门经验式的学科。这确实让人惊讶，因为绝大多数人最开始都认为这些领域是建立在他们的数学或者工程学基础上的。从数学的角度来看，有时称之为“优雅”（neat）的观点，理性主义者希望把证明和分析的标准转换为智能计算设备的设计。而从工程的角度来看，有时又称之为“粗陋”（scruffy）的观点，这种任务通常被看作是简单地制造出那些被人们称为“智能”的成功的人工制品。不幸的或幸运的是（这依赖于你的哲学观），不管是从纯数学角度还是纯工程角度来说，智能软件的复杂性和在人机交互活动中与生俱来的不确定性都阻挠了我们的分析。

此外，如果人工智能想达到科学的水平并成为智能系统科学的关键组成部分，那么就必须在它制造的人工制品的设计、执行和分析中包含分析和经验式的方法。从这种观点来看，每个人工智能程序都可以看作是一个实验：它向现实世界提出问题，而答案就是现实世界对此做出的响应。现实世界对我们的设计做出的响应和程序式的承诺构成了我们对于智能的形式方法、机理以及智能本质的理解（Newell and Simon 1976）。

与许多较为传统的人类认知研究不同，我们作为智能计算机这种人工制品的设计者，能检查到我们的“主题”的内部运行。我们还能在以后停止程序的执行，检查内部的状态，或者修改程序的结构。正如纽维尔和西蒙（1976）指出的，计算机及其程序的结构表明了它们潜在的行为：它们可以被检查，且它们的表示和搜索算法能够被理解。计算机作为能够理解智能的有力工具，它是这种二元性的产物。经过适当编程的计算机能够同时达到语义和行为的复杂性的水平，它们一方面需要用心理学术语来刻画其特征，另一方面能够为其内部状态的检查提供机会，而这种检查在很大程度上被研究其他智能生命形式的科学家们所否定。

幸好，对于人工智能的后续研究，以及对于建立一门智能系统的科学，更多现代心理学技术，尤其是跟神经心理学相关的技术，已经促使我们在人类智能的许多研究方面获得了进展与成功。例如，我们现在知道，人的智能不是单一的也不是始终不变的，它更像是模块化的和分布式的。它的能力可以从感知器官（比如人的视网膜）中看到，它可以检测并预处理视觉信息。同样地，人类学习也不是始终不变的、相似的。而学习是对多种环境和不同系统所具有的一种功能，每种学习适于达到特定的目的。fMRI 分析以及 PET 扫描和相关的神经物理成像过程，都支持实际智能系统内部工作中的多样的和协作的图像。

如果人工智能的工作想要达到科学的水平，我们还必须处理一些重要的哲学问题，尤其是那些与认识论相关的问题，或是智能系统是怎样“知道”它的世界的问题。这些论点涉及人工智能研究的对象是什么，以及更深层的问题，如物理符号系统假设的有效性和实用性中存在的问题。还包括更多的问题，如人工智能的符号系统方法中“符号”到底是什么，在连接模型中符号是如何关联到多组带权重的结点。我们还讨论在大多数学习器中能看到的归纳偏置所表现出的理性的作用，并与在无监督学习、强化学习和涌现学习的学习方法中经常能看到的松散的无结构特征相比较。最后，我们还必须讨论问题求解中具体设施、状态以及社会倾向的作用。我们提出了一种构造主义的认识论来归纳我们在哲学问题上的讨论，这种构造认识论很适合于我们将人工智能既作为一门科学又作为一门按经验进行查询的学科的承诺。

因此在这最后一章中，我们又回到第 1 章中提出的问题：智能是什么？它能够被形式化吗？我们如何才能建立一种展现它的机制？人工智能和人类智能如何能够融入到智能系统科学的更大背景中。在 16.1 节中，我们从修订后的人工智能的定义开始，它表明了人工智能现在的研究工作是如何扩展其工具、技术，并在更为广阔的环境中探索的，但是它仍然源自于纽维尔和西蒙的物理符号系统假设。我们探究解决这些智能问题的不同方法，并考虑它们在智能机器设计方面以及作为智能系统科学的部件的能力。在 16.2 节中，我们指出有多少现代的认知心理学、神经科学和认识论的方法可以用来增强人们对人工智能事业的理解。

最后，在 16.3 节中，讨论了现代人工智能研究者和认识论学者所面临的挑战。虽然传统的人工智能方法有理性简化论的嫌疑，而跨学科的新见解和新工具也有不足。例如，遗传算法的创建者和人工生命研究的设计者从达尔文的观点给智能世界下了定义：“智能世界就是生存下来的世界”。在复杂的世界中，知识同样被看作是“知道怎么样”，而不是“知道什么”。对于科学家来说，答案是需要解释的，“成功”或者“生存”并不充分。

在这最后的一章中，将通过探索建立一个智能的计算科学所必须解决的哲学问题来讨论人工智能的未来发展。得到的结论是，人工智能的经验主义的方法学是一个重要的工具，也许它对于探索智能的本质来说是最好的工具。

16.1　人工智能：修订的定义

16.1.1　人工智能和物理符号系统假设

根据我们在前 15 章中的经验，我们对人工智能给出一个经过修订的定义：

> 人工智能研究的是智能行为中的机制，它是通过构造和评估那些试图采用这些机制的人工制品来进行研究的。

在这个定义中，人工智能不像是关于智能机制的理论，而更像是一种经验主义的方法学，它的主要任务是构造和测试支持这种理论的可能模型。它是一种对实验进行设计、运行和评估的科学方法，其目的是精练模型和进行更深入的实验。然而，最为重要的是，这个定义像人工智能

本身这个领域一样，直接抨击了几个世纪以来关于智力本性的哲学蒙昧主义。对于那些将要了解什么是我们定义的人的特征的人们，这个定义给他们提供另一种选择来看待宗教、迷信、笛卡儿二元论和新时代的安慰剂，或者在一些还没有发现的量子力学的巧合中寻找智能（Penrose 1989）。如果人类知识中的很大一部分来自于支持人工智能的科学，那么智能就不再是神秘不可触及的，而是那些在智能机器的设计中能够被理解和应用的一组规则和机制的结果。这里必须指出的是，我们修订后的定义并没有给出“智能”的定义，而只是提出了人工智能在探索智能现象的本质和表示中所起的连贯作用。

从历史的角度来看，人工智能中占主导地位的方法包括表示形式化的构造以及它们相关联的基于搜索的推理机制。早期人工智能方法论的指导原则是由 Newell 和 Simon（1976）首先提出的*物理符号系统*假设。这个假设指出：

> 物理系统表现一般智能行为的充要条件是该系统是一个物理符号系统。
>
> *充分性*是指智能可以通过任意合理组织的物理符号系统来得到。
>
> *必要性*是指任何用来表现一般智能的主体都必须是一个物理符号系统的一个实例。物理符号系统假设的必要性要求任何一个智能主体，不管它是人、外星人还是计算机，都必须通过在符号结构上操作的物理实现来获得智能。
>
> *一般智能行为*是指人类活动中可以看到的相同的动作和行为。在物理范围内，系统将表现适合于其各个终端并适应于其所在环境要求的相应行为。

纽维尔和西蒙对这种假设的*必要性*和*充分性*的观点进行了总结（Newell and Simon 1976，Newell 1981，Simon 1981）。在后来的许多年中，人工智能和认知科学都在这个假设基础上进行了大量的研究。

物理符号系统假设导致了四个重要的方法论方面的约定：（a）用符号以及符号系统作为描述世界的中介；（b）搜索机制的设计，尤其是启发式搜索，用来探索这些符号系统能够支持的可能推理的空间；（c）认知体系结构的分离，这里我们的意思是假定一个合理设计的符号系统能够提供完整的智能的随意说明，而不管其实现的中介是什么；（d）基于这样的观点，最后人工智能变成为经验式和构造式的学科：它试图通过建立智能的工作模型来理解智能。

从符号系统的角度来看，语言中的记号，被称为*符号*，是用来表明或者推断出除其本身之外的其他东西。像自然语言中的语言记号一样，符号代表或者指定智能主体世界中的事物。例如，Tarski（1956，见 2.3 节）能在这些对象引用关系中给出科学含义的可能性。

此外，人工智能中符号的使用已经超出了 Tarski 语义中表述的问题，而且扩展到了表示所有形式的知识、技能、意图和因果关系。所有这些构造的结果是基于这样的事实，即符号及其语义可以嵌入到形式化系统中。这就定义了一种*表示语言*。对于将智能建模为一个运行的计算机程序来说，这种对符号模型进行形式化的能力是非常重要的。我们已经详细研究了多种表示，包括谓词演算、语义网、脚本、概念图、框架和对象。形式化系统中的数学允许我们讨论这些问题的可靠性、完备性、复杂性，以及对知识结构的组织问题进行讨论。表示形式化方面的进展使得我们能够建立更为复杂（也更为丰富的）语义关系。例如，继承系统建立了一套生物分类知识的语义理论。通过对类的继承进行形式化的定义，这种语言简化了智能程序的构造，并为智能本身的可能分类的组织提供了可测试的模型。

“搜索”这个概念在推理中将表示模式及其应用紧密地捆绑在一起。搜索是在问题空间框架（先验的语义约定）中一步一步地检测问题的状态，以寻求当前问题的解决方案、子问题目标、问题的对称性或者问题中需要考虑的其他方面等。表示和搜索是关联在一起的，因为对于一种特定表示的约定决定了搜索的空间。确实，如果选择很差劲的表示语言来对问题进行表示，则一

些问题可以变得很困难甚至不可解。本章后面部分关于归纳偏置的讨论将举例说明这一点。

关于搜索和表示之间的相互影响问题，以及选择合适的表示的难度问题（优选表示法的过程能否自动化?），一个很生动的、经常被引用的例子是，把骨牌放在被截去了两个角的棋盘上。假定我们有一个棋盘和一些骨牌，每个骨牌能覆盖棋盘上的两个方格。同时假定棋盘被去掉了一些方格；在图 16-1 中，左上角和右下角的方格被去掉了。

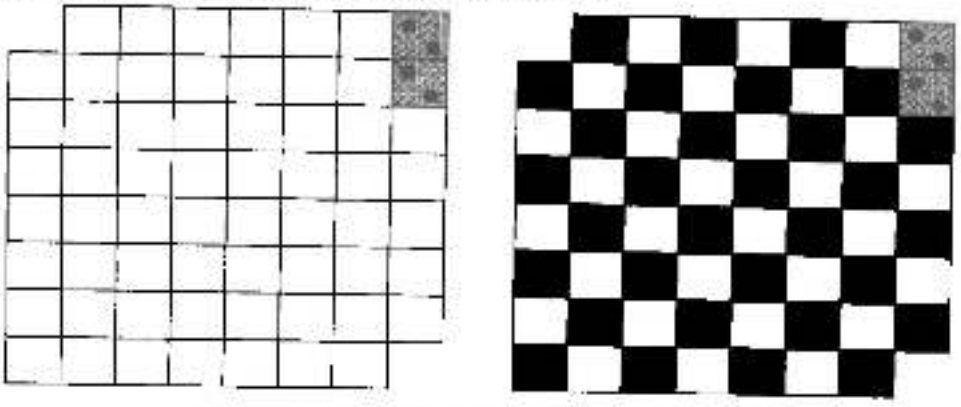


图 16-1 缺角棋盘，且一个骨牌覆盖棋盘上的两个方格

缺角棋盘问题就是询问是否有一种方法能把骨牌放在棋盘上，使得棋盘的所有方格都被覆盖，且每块骨牌恰好盖住两个方格。我们可能会尝试所有的方法来把骨牌放在棋盘上；这显然是一种基于搜索的方法，是把棋盘表示为一个简单的矩阵的自然结果，而忽略了那些似乎不相关的像方格的颜色这样的特征。这种搜索方法的复杂度是巨大的，并且需要启发式才可能得到有效的解答。例如，我们可以对部分解决方案进行剪枝，得到相互分离的单独的方格。我们也可以从小棋盘出发来解决这个问题，比如 2×2 和 3×3 的棋盘，然后把这个解答扩展到 8×8 的情形。

基于更复杂的表示模式，一种更为巧妙的解决方法是利用每个骨牌在放置时必定同时覆盖一个黑色方格和一个白色方格这个现象。而这个缺角棋盘共有 32 个黑色方格和 30 个白色方格，因此想用骨牌把所有的方格覆盖住是不可能的。这就给施辩基于符号的推理者提出一个很严肃的问题：我们是否有这种允许问题解答者使用灵活的有创造性的方法来访问知识库的表示？当已经从一个问题领域中学习到很多东西之后，特定的表示如何能自动修改其结构？

在基于符号的人工智能中，启发式是表示和搜索之后的第三个组件。启发式是一种对由特定表示所提供的选择中的搜索进行组织的机制。设计启发式的目的是克服穷举搜索所具有的复杂性，因为它是多种有趣问题寻找有用解决方法的障碍。正如人一样，在计算中智能需要对“下一步做什么”做出有见识的选择。从人工智能研究的历史来看，启发式具有多种形式。

最早的问题求解技术，如 Samuel 的 checker-playing 程序中的爬山问题（见 4.1 节），纽维尔、肖（shaw）和西蒙的通用问题求解器中的手段－目的分析（见 13.1 节），都是来自于其他学科（如运筹学），它们逐渐成熟，成为人工智能问题求解中的一般技术。搜索的属性，包括可采纳性、单调性和信息度，都是从这些较早的研究中形成的重要结果。这些技术通常被称为是弱方法。弱方法是一般的搜索策略，广泛地用在所有类型的问题领域中（Newell and Simon 1972, Ernst and Newell 1969）。这些方法及其属性的具体内容参见第 2、3、4、6 和 14 章。

在第 8、9、10 章中引入的基于规则的专家系统、基于模型和基于案例的推理，基于符号的学习等方法称为人工智能中的强方法。和弱问题求解器形成对比，强方法侧重于每个问题领域的特定信息，比如内科医学或者积分计算，而不是侧重于对问题领域进行归纳概括的启发式方法的设计。强方法是专家系统和其他知识密集型问题求解方法的基础。它强调的是问题求解所

需的知识量、学习和知识获取、知识的语法表示、不确定性的管理以及与知识质量相关的问题等。

为什么我们还没有建立许多真正的基于符号的智能系统

现在有许多关于物理符号系统智能特征的批评，大多数批评是瞄准智能主体的符号的语义含义和语义基础问题。当然，“含义”的本质也将影响智能的思想，如在预先定义好含义的符号结构中搜索以及启发式的使用中隐含的“效用”。在传统的人工智能中含义的概念充其量是很弱的。不过，向一种更基于数学的语义（例如 Tarski 的可能世界方法）发展的试探似乎是一个错误的方向。它增强了理性主义者的特点，即用一个完全可以达到的清晰的和独特观念的世界取代具体主体灵活的和演化的智能。

含义的基础这个问题，一直同时阻挠着人工智能和认知科学事业的支持者和批评者。这个基础的问题是问符号怎样才能具有含义。约翰·希尔（Searle，1980）提出了他著名的“中文屋（Chinese Room）实验”，对此进行了立论。他将自己关在一个房间中并打算把中文语句翻译为英文；他在屋内收到各种写有中文语句的纸条要他回答，然后他查阅各种工具书（汉英字典及相关专业书），并找到令人满意的英文答题传给屋外等待的人。希尔说，尽管他知道自己一点儿都不懂中文，但是屋外人依然可以根据他的回答判断他是一个懂中文的人，所以他的“系统”也可以看作是一个汉译英的机器。

这里有个问题是，尽管任何从事机器翻译或者自然语言理解（见第 15 章）的研究者都可能会争辩说，Searle 的“翻译机器”盲目地把一个符号集链接到另外一个符号集，可能会生成最差的结果，事实是现在这一代的智能系统只有非常有限的能力来有针对性地翻译这些符号集。在现在的基于感官形态的计算中，不管是视觉的、肌肉运动知觉的还是口头的，支撑语义太弱这个问题是普遍存在的。

在人类语言理解这个领域中，Lakoff 和 Johnson（1999）指出，创造、使用、交流和解释有意义符号等方面的能力是来自于人在进化的社会环境中的体验。这个环境是物理的、社会的、时实的；它支持和允许人类生存、进化和繁殖的能力。它尽可能制造了一个类比推理的世界，在这个世界中人们可以利用和鉴赏幽默，可以体会和欣赏音乐与艺术。而我们现在的人工智能工具和技术离能够编码和利用任意等价的“有意义”系统这个目标还有很大的距离。

作为这种弱语义编码的直接结果，传统人工智能的方法论主要对那些预先解释好的状态和状态环境进行探索。这就是说，一个人工智能程序设计者把语义含义的上下文归于或者放在程序的符号中。这种预先解释编码的一个直接结果就是，包括学习和语言等这样的智能性高的任务只能产生一些已经计算好的那些解释的功能。因此，许多人工智能系统在探索它们所在的环境时，它们只有很有限的能力来演化新的含义联想（Luger et al. 2002）。

最后，作为我们当前有限语义建模能力的直接结果，我们能够使用预先解释好的符号系统从一个丰富具体的社会环境中抽取并同时抓住问题求解的主要部分，这些应用是我们最为成功的努力结果。在本书中我们已经讨论许多这些方面的问题。然而，这些领域仍然很脆弱，没有多样的解释，而只有很有限的能力来从失败中自动恢复。

从人工智能的历史中我们能够看出，人工智能的研究团体已经探索出了物理符号系统假设的一些分支，并已经开始向先前占主导地位的观点发起挑战。正如本书的后面一些章节说明的，明确的符号系统和搜索并不是惟一可能的捕获智能的表示工具。基于动物大脑构架和基于生物进化过程的计算模型同样也通过科学上可知、实验上可再现的过程来提供理解智能的可能框架。本节的余下部分将探讨这些方法的分支问题。

16.1.2　连接或者“神经”计算

除物理符号系统假设以外，另外一个重要的选择是深入研究神经网络和其他从生物得到灵感的计算模型。例如，神经网络是可计算的，是认知的物理实例模型，它并不完全依赖于对明确推理和描述世界特征的符号进行预解释。因为神经网络中的知识是分布在网络结构中的，很难（即使不是没有可能）将个体概念划分到网络的特定结点和权重。实际上，网络的每一部分都可能是表达不同概念的一个部件，因此，至少对于物理符号系统假设的必要子句来说，神经网络是很明显的一个反例。

神经网络和遗传体系结构将人工智能的重点从符号表示和可靠的推理策略的问题转移到学习和适应的问题。同人和其他动物一样，神经网络是适应世界的一种机制：经过训练的神经网络结构是通过学习而形成的，就像通过设计而形成一样。神经网络的智能并不要求世界被重构为一种明确的符号模型。相反，网络是通过与世界的交互形成的，通过经验的不明确的痕迹反映出来。神经网络这种途径对我们理解智能起了极大的作用，给了我们一个在智力过程的物理具体化中可以接受的机制模型，一个更可行的学习和发展的理由，一个简单能力的示范，为反映实际现象而对复杂系统所做的局部修改，以及一个针对认知神经科学的强有力的工具。

正是因为神经网络是很不一样的，所以它们能回答许多基于符号的人工智能所不能解答的问题，其中包括感知问题。现实世界并不能很恰当地把我们的感知传递给严格的谓词演算表达式，而神经网络则提供了一种很好的模型，使用这种模型我们可以在混沌的感觉刺激中识别出“有意义的”模式。

由于神经网络是分布式表示的，因此它们比那些明确的符号系统更为健壮。一个经过适当训练的神经网络能够有效地识别出新的实例，具有像人一样的相似性感知能力，而不需要严格的逻辑判断。同样地，部分神经元的丢失并不会严重地影响这个神经网络的性能，这是因为在网络模型中有大量固有的冗余。

或许这种连接网络最让人惊讶的特征是，它具有学习能力。神经网络不像符号系统一样试图建立一个现实世界的信息符号模型，神经网络依赖于它们自己结构的可塑性可以学习经验。它们并不构建世界的模型，而是在世界中不断地形成其经验。学习是智能最为重要的方面。在神经计算的工作中提出的最难解决的问题也是学习问题。

为什么我们还没有建立一个大脑

实际上，目前这一代工程连接系统与人类神经系统的相似之处非常少！因为神经系统的可行性是一个关键的问题，所以我们从这一问题开始讨论，然后进一步考虑发展和学习。最近的认知神经科学方面的研究（Squire and Kosslyn 1998，Gazzaniga 2000，Hugdahl and Davidson 2003）给我们对人类认知体系结构的理解带来了新的见识。在本节中，将简单地给出一些发现和关于它们是如何与人工智能相关联的评注。我们将考虑三个层次的问题：首先是神经元层次的问题，其次是神经元体系结构层次的问题，最后我们讨论认知表示或者编码问题。

首先，在单独的神经元个体这个层次上，Shephard（1998）和 Carlson（1994）识别出了细胞的多种不同类型的神经元体系结构，每种类型的神经元由于其在这个系统中的功能和作用不同而不同。它们包括感觉接收细胞（sensory receptor cell），主要在皮肤部分，用来将输入信息传递给其他细胞结构，内部神经元的主要任务就是在内部细胞群中进行通信，主要神经元的任务也是在细胞群之间进行通信，而运动神经元则主要负责系统输出。

神经活动是与电有关的，离子在神经元中流入和流出的模式决定了一个神经元是处于活动状态还是休息状态。一个典型的处于休息状态的神经元其电压约为 -70mV。但一个神经细胞处

于活动状态时，不同的化学物质将在神经的轴突端释放出来。这些化学物质称为是*神经传导物质*，它影响了突触后细胞膜，典型地适合于特定的接收器，就像一把钥匙插入一把锁一样，并初始化更多的离子流。当离子流达到一个特定的临界水平（大约 -50mV），就产生一个*动作电位*，一种“全或无”型的触发机制表明这个神经细胞已经触发。因此神经元开始通过二元编码序列进行通信。

由动作电位改变而形成的突触后电位有两种类型：*抑制性突触后电位*和*兴奋性突触后电位*，抑制性突触后电位主要出现在内部神经元结构中。这些正电位或者负电位将通过树状系统的突触而不断地产生。当所有这些事件的网络效应是把相关神经元的膜电位从 -70mV 变为 -50mV 时，就会越过阈值，大量的离子流就又会被传授到这些神经细胞轴突中。

其次，在神经元体系结构层，大脑皮层大约有 10^{10} 个神经元，它们以一层薄片的形式盘绕在大脑的半球上。许多皮层是褶皱的，以增加整个表面积。从可计算性的观点来看，我们不仅需要知道突触的总数，也必须知道扇入和扇出参数。Shephard（1998）估计这两个数字大约都在 10^5 数量级。

最后，除了神经元细胞及其体系结构与计算机系统的区别以外，这里还有一个更深层次的认知表示问题。我们对于大脑还有很多东西不清楚，例如简单的记忆是如何在大脑皮层中编码的，又如人脸是怎样识别的，一个识别出来的人脸又如何能够与高兴或者悲伤的感情联系在一起。我们知道了大量的人脑的物理和化学特征，但却不知道神经系统在一定环境中是如何编码和使用“模式”的。

对于神经系统的研究者和计算机系统的研究者来说，有一个需要共同面对的难题，即先验知识在学习中到底有什么作用。在没有初始的知识和学习经验时，从空白开始能够进行有效的学习吗？或者说一定要从一些先验的归纳偏置开始吗？机器学习程序方面的经验告诉我们，在复杂的环境中进行学习时，通常都需要一些先验知识，这些先验知识一般表示为一种归纳偏置。

神经网络从一定的训练数据中收敛到有意义的泛化的这种能力，经过证明，对于人工神经元的数量、网络拓扑以及使用的具体学习算法来说是很灵敏的。此外，这些因素还能形成与任意符号化表示同样强大的归纳偏置。人类发展方面的研究支持这个结论。还有更多的证据也证明了这一点，例如，婴儿继承了一定的“硬连线的”认知偏置，他们能对语言和常识性物理知识等概念领域进行学习。在神经网络中对先天偏置的刻画问题是一个很活跃的研究领域（Elman et al. 1996）。

如果考虑很复杂的学习问题，那么先验偏置方面的问题将变得更为复杂。例如，假定我们正在开发一个科学发现的计算模型，并想对哥白尼将地心说转变为日心说的宇宙观进行模型化，这就要求我们必须将哥白尼和托勒密的观点用计算机程序表示出来。虽然我们可以在神经网络中将这些观点表示为激活的模式，但我们的网络可能对于他们那些视为理论的行为一无所知。相反，我们更喜欢做出如下的解释：“哥白尼可能会被托勒密学说的复杂性所困扰，因而他给出了一个简单的模型，让行星围绕着太阳转”。做出上述解释也需要用符号来表示。显然，神经网络必须要能支持符号推理；毕竟，人本身就是一个神经网络，他们在处理符号方面至少还算不错。尽管如此，符号推理的神经基础仍然是一个重要的、需要继续研究的问题。

另外一个问题是在学习中发展的作用。小孩不能在可用数据的基础上进行学习，他们在特定领域的学习能力处于一个设定好的发展阶段（Karmiloff-Smith 1992）。一个有趣的问题是，这种发展的进展是否仅仅是人类生物及其具体化的结果，或者说，它是否反映对学习其世界不变性的智能能力的某些逻辑上的必要限制。发展阶段的功能是否起到了一种机制作用，这种机制将学习世界的问题分解为更易于管理的子问题？一系列人为强加的发展约束是否能够提供一种

带有对复杂世界进行学习的必要框架的人工网络？

神经网络在实际问题中的应用提出了一系列其他的研究问题。神经网络的许多令人惊讶的属性，比如对于丢失数据或者模糊数据的适应性和健壮性，也是实际应用中需要解决的问题。由于神经网络是经过训练的，不是事先编好程序的，因此其行为是很难预料的。对于一个给定的问题领域来说，很难指导人们如何去设计相应的神经网络使其在该领域能正常收敛。最后，对于为什么一个网络能得到特定结论这个问题，要解释起来是很困难的，通常只能以统计的形式来给出解释。这些就是当前所有的研究领域。

这时有人可能会问，基于连接的网络和符号人工智能作为智能模型是否不同。它们有一些重要的共性，比如它们都认为智能最终要被编码为计算，有一些基础的形式化的限制，例如丘奇/图灵（Church/Turing）假设（Luger 1994，第2章）。两种方法都提供了由实际问题应用所形成的智力模型。更为重要的是，两种方法都拒绝哲学的二元论，而都认为智能的基础是物理实现设备的结构和功能。

我们相信这两种截然不同探索智能的方法终究会达成一致，融为一体。当它们最终达成一致时，一种关于符号如何简化为网络中的模式，以及如何修改网络的理论将成为特别的贡献。这将会导致一系列的发展，比如将基于网络的感知和知识密集型的推理整合到单一的智能中。同时，两类研究团体都有相当多的工作要做，它们没有理由不共存。有人会觉得这两种不可比较的智能模型的共存很不合适，其实这根本就不必在意，大家可以看到，即使是在物理领域中，也有两种直觉上相互矛盾的概念存在，比如光的波粒二相性（Norton 1999）。

16.1.3　主体、涌现和智能

基于主体的计算和认知模块化理论的出现为人工智能的研究者们提出了另外一些有趣的问题。认知科学中一个重要的学派认为，智能是被组织成一些特定功能单元的集合（Minsky 1985，Fodor 1983）。这些模块是专家级的，它们使用一定范围的先天结构和功能来说明它们所谓应用的主体必须解决的那些问题的多样性，比如从“硬连线”问题求解到归纳偏置。下面这个问题有意义：如何才能训练一个单一的神经网络来具有多种功能，比如感知、运动神经控制、记忆以及高级推理等？智能的模块化理论一方面为解答这些问题提出了一个框架，另一方面也为继续对先天偏置在个体模块以及模块交互机制方面的本质问题的研究指出了方向。

计算的遗传和涌现模型对于理解人和人工智能提供了一种最新的、最激动人心的方法。整体智能行为可以由大量受限的、独立的和个体化的单一主体的协作而形成，通过论证这一点，遗传和涌现理论讨论了那些通过相对简单结构的内部关系而表达出来的复杂智能的问题。

在Holland（1995）的一个例子中，维持一个大城市（如纽约）的面包供应的机制说明了在一个基于主体的系统中智能现象的基本过程。这并不是说我们能够拟定出一个集中式的规划程序，来成功地完成纽约日常的面包供应任务。无论如何，写一个用来保证纽约日常面包供应的集中规划算法有很多实际的困难，但城市中由许多面包师、运输商、原材料供应商以及零售商等松散合作的结果却能很好地解决这个问题。正如在所有基于主体的涌现系统中一样，这里没有集中式的规划。每个面包师很少有关于城市对面包需求的知识，每个面包师只是尽量地把他的生意做好。整个问题的解决方案是在这些独立和局部的主体的集体行动中浮现出来的。

通过论证有目的、健壮的、优化的行为是如何产生自局部单独的主体交互中，这些模型还为智力的起源这个古老的哲学问题给出另外一个解答。智能的这种涌现方法的中心思想是，完全的智能能够且确实是从许多简单的、单独的、局部的和具体化的主体智能的交互中出现的。

涌现模型的第二个主要特征是它们依赖于达尔文的自然选择机制，这种基本机制形成了单

独主体的行为。在前面的面包师例子中，似乎每个面包师并没有按照一定的全局优化的方式去行动。这种优化的来源并不是集中式的设计，而是一个简单的事实：没有做好工作、不能满足客户需要的面包师通常会被淘汰。它是通过对这些选择压力的永不疲劳的、持续的运作，使得每个面包师都去完成相应的行为，即一方面使得他们能够生存下来，另一方面又涌现出某种实用的集体行为。

分布式的、基于主体的体系结构和自然选择的适应性压力综合在一起，形成了智力的起源和运作的最强大的模型。进化心理学家们（Cosmides and Tooby 1992，1994；Barkow et al. 1992）已经给出了一种方法的模型，其中自然选择塑造人类智力的先天结构和偏置的发展。进化心理学的基础是把智能看作是高度模块化（作为一种交互的系统）和专业化的主体。确实，进化心理学方面的讨论经常把智能与瑞士军刀进行比较，因为瑞士军刀集中了多种用来解决各种不同问题的专门工具。

越来越多的证据表明人的智能确实是高度模块化的。Fodor（1983）为智能的模块化结构提供了哲学依据。Minsky（1985）对人工智能的模块化理论的分支问题进行了研究。这种体系结构对于智能的进化理论是很重要的。很难想象进化是怎样形成像思想一样复杂的单独体系。然而，千百万年的进化可能会成功地形成单独的、特殊的认知技能。当大脑在不断地进化时，它同样能够继续模块的结合，以形成一种能够允许模块之间进行交互、共享信息、协同执行越来越复杂的认知任务的机制（Mithen 1996）。

神经选择理论（Edelman 1992）表明了这些相同的过程如何能够说明个体神经系统的适应性。神经达尔文主义使用达尔文的术语对神经系统的适应性进行了建模：大脑中特定电路的强化和其他电路的弱化是对世界做出反映的一种选择过程。这与符号学习方法不同，符号学习方法试图从训练数据中抽取信息并使用这些信息来构建世界的模型，而神经选择理论则对神经元及其交互中的选择电压的效果进行检验。Edelman（1992）指出：

> 考虑到脑科学作为一种认知科学，我的意思是说认知不是一种启发过程。没有直接的信息传递发生，正如在进化或者免疫的过程中没有任何事情发生一样。相反，认知是有选择性的。

主体技术也提供了社会协作的模型。使用基于主体的方法，经济学家构建了市场经济的信息模型。在各种应用中，主体技术发挥着越来越大的作用和影响，如分布式计算系统、因特网搜索工具的构建、协同工作环境的实现等。

最后，基于主体的模型在知觉理论方面也产生了巨大的影响。例如，Daniel Dennett（1991，2006）把意识的功能和结构归结到智能的主体体系结构上。他指出，要问意识位于智能/大脑的什么位置这种问题是不正确的。相反，他的*意识多转移理论*则侧重于意识在分布式智能体系结构中的主体间交互中所起的作用。在感知、运动神经控制、问题求解、学习和其他智力活动的过程中，我们形成了交互主体的联合。这种联合是高度动态的，是随着不同情形的需要而改变的。Dennett 认为，意识是这些联合的一种绑定机制，它支持主体间的交互，并使得交互主体间的临界联合出现在认知过程的最显著位置。

哪些问题限制了基于主体的智能近似

如果它们的承诺会实现的话，那么基于主体的方法和目前涌现的方法则可以解决许多问题。例如，我们现在还必须补足能够进化高级认知能力（比如语言能力）的所有步骤。正如古生物学家在重构物种进化这方面的努力一样，追踪这些高级问题的发展还需要大量额外的、更加细致深入的工作。我们必须同时列举作为智能结构基础的主体，并追踪它们进化的过程。

基于主体的理论的另外一个重要问题是，如何解释各个模块之间的交互。虽然智能的“瑞

士军刀”模型是一种有用的构造模型，然而组成智能的各个模块并不像瑞士军刀中的各种工具那样是独立的。智能表现出了认知领域之间广泛而高度变化的交互：我们可以说出我们看到的东西，这表明视觉和语言模块之间的交互。我们可以构建具有一定社会用途的建筑物，这表明技术和社会智能之间的交互。诗人能构造出从视觉到触觉的暗喻，表明了视觉和触觉模块之间的一种互动交互。定义一种能够允许这些内部模块之间交互的表示和过程是当前一个很活跃的研究方向（Karmiloff-Smith 1992，Mithen 1996，Lakoff and Johnson 1999）。

基于主体技术的实际应用也变得越来越重要。使用基于主体的计算机模拟，我们就有可能对那些没有固定格式数学描述的复杂系统进行建模，而这在此之前是不可能深入研究的。基于模拟的技术已经广泛地应用在许多现象中，比如人的免疫系统的适应性调整、复杂过程的控制（包括粒子加速器）、全球货币市场的行为、天气系统的研究等。实现这种模拟所必须解决的表示和计算问题不断地驱动着知识表示、算法、甚至计算机硬件设计等方面的研究。

主体体系结构必须解决的其他一些实际问题包括主体间通信协议，特别是当本地主体对于问题的解决通常只有有限的知识，而这些知识是其他主体所了解的，或者这些问题是其他主体已经处理过的。此外，现在很少有将大型问题分解为多个连贯的面向主体子问题的算法，或者将有效的资源分配给多个主体的算法。这些问题以及其他与主体相关的问题参见 7.4 节中的讨论。

或许，关于智能的涌现理论中最令人振奋的特点是，它具有把智力活动从混乱状态中归入到一种按涌现顺序建立的统一模型中的这种潜能。本节中的一些简要概述已经引用了一些用涌现理论对广泛过程进行建模的工作，这些过程包括从脑的进化到在个体中能动的学习的力量，再到行为的经济模型和社会模型的建立。这个概念还有某种格外的吸引力，由达尔文进化论过程形成的有序现象的相同过程能够以各种各样的分辨方式来解释智能行为，从单个神经元的交互到脑模块化结构的形成，再到经济市场和社会系统的运行。当相同过程出现在我们看待整个系统的任意一种分辨层次时，似乎智能具有一个分形几何结构。

16.1.4　概率模型和随机技术

20 世纪 50 年代早期，随机技术用来解决自然语言表达的理解和生成问题。Claude Shannon（1948）将概率模型应用于自然语言处理中，包括离散马尔可夫链。Shannon（1951）还借鉴了热力学中熵的概念作为衡量消息中信息容量的方法。差不多同一时间，贝尔实验室创建出了第一个统计系统，能够在使用者朗读时识别出 0 到 9 十个数字。其准确度在 97% ~99%（Davis et al. 1952，Jurafsky and Martin 2008）。

20 世纪 60 到 70 年代，在人工智能研究活动的大背景下，推理的贝叶斯方法持续发展。自然语言技术研究了许多基于符号的方法，在 7.1 节有介绍。虽然许多专家系统建立了自己的“确信度代数”（见 9.2.1 节），如 MYCIN，但还是有几个专家系统采用了贝叶斯方法（Duda et al. 1979a），如 PROSPECTOR。然而，这些系统的复杂性很快变得难以处理。正如我们在 9.3 节中指出的，对实际大小具有 200 疾病和 2000 症状的医学诊断程序来说，完全使用贝叶斯规则将需要收集和综合 8 亿条信息。

20 世纪 80 年代后期，Judea Peal（1988）提出了一个计算上易于处理的模型，解决在问题领域中因果关系的上下文中诊断推理的问题。这个模型就是贝叶斯信念网络。贝叶斯信念网络放宽了完整贝叶斯模型中的两个约束。第一，隐含的“因果关系”在随机推理中做了假定，即推理是从原因推出结果，并且没有环，也就是说结果不能又推导出自身的原因。这样贝叶斯信念网络就可以表示为有向无环图（3.1 节，9.3 节）。第二，贝叶斯信念网络假定结点的直接父结点支持与该结点完整的因果影响。所有其他结点都假定有条件的独立或者因影响太小而忽略。

Peal 的研究（1988，2000）重新引起了用随机方法为世界建模的兴趣。在 9.3 节介绍过，贝叶斯信念网络为诊断（推导）推理提供了一个非常强有力的表达工具。这一点对具有动态特性的人类和随机系统确实非常特别：当世界随时间变化时，我们的理解会不断丰富：生成一些原因来解释更多我们看到的现象，而另一些潜在的原因则对解释没有用了。对随机系统设计以及支持它们的推理框架的研究实际上还处于起步阶段。

20 世纪 80 年代后期，在自然语言处理问题中还出现了一些新的研究动力。在 15.4 节中介绍了这些随机方法，包括消除语言表达式歧义的分析、标记和许多其他新技术。关于这些新方法的全面介绍，可以阅读关于语音识别和语言处理任务方面的书（Manning and Schutz 1999，Jurafsky and Martin 2008）。

伴随着随机方法在刻画智能行为方面取得的成功和激发的兴趣，人们会十分自然而然地想知道随机方法的局限性在哪里。

智能本质上是随机的吗

关于主体在不断变化世界中的交互的随机观点具有极大的吸引力。许多人可能会讨论，人类的“表示系统”本质上是随机的，也就是说，会受感知的世界和出现的原因限制。行为主义者/经验主义者的观点当然会发现这种推测很有吸引力。情景和内置动作理论家可能会走得更远，并假设主体具有的与物理和社会环境的条件关系为主体成功地适应环境提供了足够的解释。某些现代的研究者甚至宣称神经功能方面本质上是随机的（Hawkins 2004，Doya et al. 2007）。

然而，这些假设都过于简单。让我们来看一下语言。乔姆斯基和其他研究者指出，口语和书面表达的优势之一在于其生成性（generative nature）。这就是说，在词汇表和有用的语言形式集合中，新的原来没有的表达式会自然出现。这一现象在新句子以及单词、构词法两个层面都可能出现，例如：“Google 一下！”（Google it）。语言的随机描述必须展示能生成新表达式的能力。目前，无论是树库还是数据语料库，这些收集的语言信息的局限性都会相当大地限制随机技术的使用。这是因为收集到的信息必须为解释当前的新情景提供正确的设置（或优先级）。

应用领域中的随机模型也有同样的局限，如在航空发动机或运输系统中。必要的，对于现实复杂系统的模型，都会有*封闭世界*或*最小模型假设*（见 9.1 节）。这就意味着具有任何模型都能考虑的现象的优先级限制，但没有能力预测新的情况。

在更高的解释层次，考虑模型变化处理本身的解释系统（或范畴）之外的情况是很难的。在 16.1.2 节中提到过，在什么意义下，模型能够解释概念上观点的理论或高层重整，即根据变化到不同观点的需要来进行关于模型充分性的再评估问题。这些主题仍然是重要的研究问题，也限制了用于理解非确定性的随机方法。

但是，事实仍然是，通常在没有精确的指令的情况下，主体也能“创建”相当成功的模型。正如 16.2 节中我们从构建主义者的角度看到的，一些模型是主体理解周围世界的必要条件，也就是说，如果没有世界“大约”是什么的事先约定，那么现象就无法感知，也无法理解！

除了刚才介绍的哲学讨论外，随机系统的应用还有许多实际限制。目前的贝叶斯信念网络本质上是命题逻辑的。直到最近它才仅能表示像下面这样的通用规则或关系：“$\forall X$ male(X)→smart(X)”。此外，希望贝叶斯信念网络中能含有递归关系，尤其是时间序列分析。对于进一步的研究，研究开发一阶的随机表示系统（解决一般性的问题）是很重要的。如 13.3 节介绍的，目前已经有了一阶随机建模工具和图灵完备的随机建模工具（Pless et al. 2006）。探索随机模型在神经/心理系统中的应用也很重要，目前还很少有这方面的应用（Burge et al. 2007，Doya et al. 2007）。

我们下面讨论人类智能心理和哲学方面，它们会影响人工智能的创造、部署和评估。

16.2　智能系统科学

人工智能界中一部分主要的研究者侧重于研究对人类智能的理解，其实这并不是一种巧合。人们为智能活动提供了一种原型实例，而且人工智能的工程师们，即使他们一般并不会承诺“使他们的程序像人一样动作”，也很少忽略人的解决方案。一些应用（比如诊断推理）通常都有意地将模型建立在该领域的权威专家的解决过程上。更为重要地，理解人类智能本身就是一个吸引人的、有待研究的科学挑战。

现代*认知科学*，或者叫*智能系统科学*（Luger 1994）随着电子计算机的出现而开始，尽管之前还有很多这门科学的前辈们进行了卓有成效的研究（见第 1 章），从亚里士多德，到笛卡儿和布尔，到现代的理论家，如图灵、McCulloch 和 Pitts（他们是*神经网络*模型的创立者）、冯·诺依曼（人工生命研究的倡导者）等。这些研究逐步变成了一门科学，它能在这些理论概念的基础上设计和运行实验，并且与计算机的结合而获得重大进展。最后，我们会问：“有包含一切的智能科学吗?”我们也可以进一步问：“存在一种智能系统科学能够支持人工智能的构造吗?”

在接下来的几小节中，将简要讨论心理学、认识论、社会学如何支持人工智能的研究和发展。

16.2.1　心理学约束

认知科学的早期研究对人在解决逻辑问题、简单游戏、规划、概念学习等方面进行了考察（Feigenbaum and Feldman 1963，Newell and Simon 1972，Simon 1981）。纽维尔（Newell）和西蒙（Simon）在做“逻辑理论家”程序的同时（见 14.1 节），开始把自己的计算方法和人们所使用的搜索策略进行比较。他们的数据来自人们在设计一个问题解法（如一个逻辑证明）的*出声思考协议*和描述。然后，纽维尔和西蒙还把这些协议与计算机程序在解决相同问题时的行为进行了比较。研究者们发现在人类所采取的协议和计算机程序面对问题时的行为之间存在着值得注意的相似性和有趣的差异。

这些早期的项目建立了*认知科学*可以使用的方法论，这是在其后几十年中可以采用的：

1）基于人们解决特定类型问题的数据，为问题求解设计一个表示模式和相关的搜索策略；

2）运行基于计算机的模型，产生求解行为的轨迹。

3）观察人是如何解决这些问题的，并记录其解决过程中的可测量参数，比如在出声思考协议方案、眼睛移动和书面记录的部分结果中发现的那些信息。

4）分析和比较人和计算机的解决方案。

5）为下一轮测试和比较修改计算机模型。

这些经验式的方法论具体见纽维尔和西蒙的图灵奖获奖演说，我们在本章的开始已经引用了部分内容。认知科学的一个重要方面是使用实验来验证问题求解的结构，不管它是一个产生式系统，连接系统，或者是基于分布式主体交互的体系结构。

最近几年来，这种范例中又出现了一种全新的维度。现在，不仅仅是程序在问题求解活动中能够被解构和观察，而且人和其他生命形式也一样能够这样。最近也出现了许多新的成像技术，它们可以用来观察大脑皮层的活动。比如*脑磁造影术*（MEG），它能探测到大量神经元所生成的磁场。与这些神经元生成的电位不同，磁场并不会因为头骨和头皮而变得模糊不清，因此其效果可能更好。

第二种成像技术是*正电子发射 X 射线断层显像术*（PET）。首先将一种放射性物质（如 O^{15}）注入到血流中，当大脑的特定区域处于活动状态时，此时通过探测器的标注了这种放射性核素的物质将比此区域处于休息状态时多。比较这些静止的和活动的图像能够潜在地揭示出功能性

的分布，其分辨精度大约在 1 厘米（见 Stytz and Frieder 1990）。

神经分析的另外一种技术是功能性磁共振成像技术（fMRI）。这种方法由基于核磁共振（NMR）的标准结构成像这个原理发展而来。像 PET 一样，这种方法通过比较休息神经状态和活动神经状态来揭示功能区域定位。

对于大脑功能区域定位的更为深远的贡献是由 Barak Pearlmutter 和他的同事共同开发的软件算法（Pearlmutter and Parra 1997，Tang et al. 1999，2000a，2000b），它与上述的成像技术密切相关。这些研究者能够把经常看到的复杂的噪声模式作为不同神经成像技术的输出，并把它们分解为不同的部件。这是分析过程中的重要步骤，因为出现的正常稳定状态的模式，比如眼睛移动、呼吸和心跳，一般是与其他我们想要了解的神经激发模式交织在一起的。

认知神经科学最近的研究结果（Squire and Kosslyn 1998，Shephard 1998，Gazzaniga 2000）大大加深了我们对于与智能活动有关的神经组件的理解。尽管对这些结果的分析和批评已经超出了本书的范围，但我们还是列举和参考了一下其中几个重要问题：

在感知和注意领域，存在着绑定问题。像 Anne Triesman（1993，1998）和 Jeff Hawkins（2004）这样的研究者指出，感知的表示依赖于分布式神经编码，这些神经编码用来将对象的部件和属性相互联系起来。他们还问了这样的问题“需要什么样的机制来‘绑定’与每个对象相关的信息，并把一个对象与其他对象区分开来?”

在视觉搜索领域，哪一种神经机制能支持对嵌入大型复杂背景中的对象的感知？一些实验表明，在搜索目标的选择中，消除无关对象的信息起到了很重要的作用（Luck 1998）。此外，我们是怎样“学习”去看的（Sagi and Tanne 1998）？

在感知的可塑性领域，Gilbert（1992，1998）主张，我们所看到的并不是严格地反映场景的物理特征，而是高度依赖于我们的头脑试图解释这个场景的过程。

大脑皮层系统是如何表示和检索暂时相关的信息的，包括感知的解释和运动神经活动的产生（Ivry 1998）？

在记忆研究中，在情绪激动时所分泌的压力激素调节着记忆的过程（Cahill and McGaugh 1998）。这跟基本的问题相关：思考、词语和感知对于一个主体是如何成为有意义的？什么样的感觉才可能是“悲伤的事情”？维吉尔的诗“lacremae rerum”是吗？

说话的语音声学特点为连接关于认知的神经科学研究和语言学理论提供了很好的组织原则（Miller et al. 1998）。大脑皮层的语法和语义组件是如何整合的（Gazzaniga 2000）？

一个人如何才能掌握一门特定语言，哪些神经心理学阶段能够支持这种智力发育（Kuhl 1993，1998）？

怎样理解发育，什么是临界期可塑性，以及什么是哺乳动物躯体感觉系统中的成年重组（O'Leary et al. 1999）？发育阶段对于“构建”智能是很关键的吗？具体讨论见 Karmiloff-Smith（1992）和 Gazzaniga（2000）。

人工智能实践肯定不要求上述这些课题以及神经/心理学等领域中的广博知识。然而，这种类型的知识一方面能支持制造人工智能制品的工程，另一方面也能帮助把人工智能的研究和发展定位在智能系统的大科学这种环境中。最后，心理学、神经心理学和计算合成等的创建是很激动人心的，但这需要一种成熟的认识论，我们将在下一节讨论。

16.2.2　认识论问题

假如你不知道自己正走向何方，便必须格外小心，以免永远到不了该到的地方……

——约吉·贝拉

人工智能现有的发展是通过许多重要的挑战和质疑而形成的。自然语言理解、规划、不确定情形的推理以及机器学习等就属于这种抓住了智能行为本质特征的典型问题。更为重要的是，在这些领域运作的智能系统需要所处情形和社会环境的目的、实践、性能等方面的知识。为了完成这些目的，我们必须检查那些试图带有“智能”的程序在认识论方面的许诺（epistemological commitment）。

这种认识论方面的许诺同时反映了知识符号使用的语义和所使用符号的结构。这些情形的任务是发现和开拓存在于一个问题领域中的不变性。“不变性”是用来描述复杂环境的规律性或可操作特点的一个术语。在现在的讨论中，经常使用符号和符号系统这两个术语，从 Newell 和 Simon（1976）所采用的明确符号，到连接系统的结点和网络体系结构，再到遗传和人工生命的涌现记号。虽然下一个问题在大多数人工智能领域中都很普遍，但我们将讨论集中在机器学习领域，因为在本书中为学习创建了多个例子和算法。

尽管机器学习领域取得了这些进展，但它仍然是人工智能所面临的最难问题之一。有三个问题限制着我们目前的理解和研究进程：一是泛化和超量学习问题，二是学习中的归纳偏置问题，三是经验主义者的困境问题或者对无约束学习的理解。后面这两个问题是相关的：许多学习算法隐含的归纳偏置是理性主义者由对期望的片面理解而产生问题的具体体现，也就是说，我们经常学习的东西似乎就是我们希望学习的东西直接作用的结果。从相反的观点来看，正如我们所看到的，在人工生命的研究中只有非常少的学习内容是事先期望的。那么“构造它，则它将会发生”这样说是否就够了呢？根据本节开始时提到的约吉·贝拉的话，可能不是这样的！在接下来的几节中将简要讨论这些问题。

泛化问题

我们过去常介绍的不同学习模型的例子具有很大的局限性，这些学习模型包括基于符号的模型、连接模型和涌现模型。例如，连接体系结构通常仅仅包含很少的结点或者部分隐藏的层。这是适当的，因为在单一神经元或者部分层的环境中就足以解释主要的学习规律。这可能会令人很迷惑，因为神经网络应用通常是异常庞大的，而且问题的规模大小是很重要的。例如，对于反馈学习，通常需要采用大型网络中大量的训练实例来解决任何有实际意义的问题。许多研究者发表了大量的评论，涉及的课题包括选择合适数目的输入值，输入参数和隐藏结点的比例，以及收敛前所必需进行的训练试验（Hecht-Nielsen 1990，Zurada 1992，Freeman and Skapura 1991）。事实上，对于任意一个学习算法来说，训练数据的数量和质量都是很重要的问题。一个学习算法如果没有适当的误差或关于一个领域的广泛的内置知识，则它可能会被引入歧途，试图在嘈杂的、稀少甚至错误的数据中去寻找模式。除了承认有这些困难的、重要的且公开的问题以外，本书还没有谈到这种类型的“工程”讨论。

一个相关问题是学习中的“充分性”问题。什么时候我们才能够说我们的算法对于抓住一个问题领域的重要约束或者不变性来说是充分的呢？我们是否预留了一部分原始数据以测试我们的学习算法？我们所提供的数据的多少与学习的质量有关吗？或许这种充分的评价是启发式的或者具有审美学的：我们人类通常都认为自己的算法是“足够好的”。

下面我们举例说明这个泛化问题，这里使用反馈方式从一组数据点中导出一般函数。图16-2表示了我们要求我们的算法去进行泛化的数据点。通过这些点的线表示了学习算法归纳出的函数。记住，一旦这个算法已经训练好，我们就将给它一些新的数据点，让它从这些数据点集合中产生一个好的泛化函数。

归纳函数 f_1 可以表示一个相当准确的最小均方近似。随着进一步的训练，系统可能产生 f_2，它看起来似乎跟这些数据点更贴近了，但是 f_2 还没有准确地与各个数据点匹配。若再进一步训

练，可能就会产生一个恰到好处的函数，它更好地拟合这些数据点，但是这对于新的输入数据来说，可能会产生很糟糕的泛化结果。这种现象称为对网络的过度训练。反馈学习的一个优点是，它在许多应用领域中能够产生有效的泛化结果，也就是说，适合于训练数据的函数近似也能正确地对新的数据进行处理。然而，确定网络所经过的从低训练量状态到过度训练状态的这个临界点是很重要的。有人可能会想，可以依据原始数据给出一种神经网络或者学习工具，然后简单地站在旁边观察，直到对于新的相似问题也产生了最有效的、有用的泛化结果，但这种想法还是太天真了。

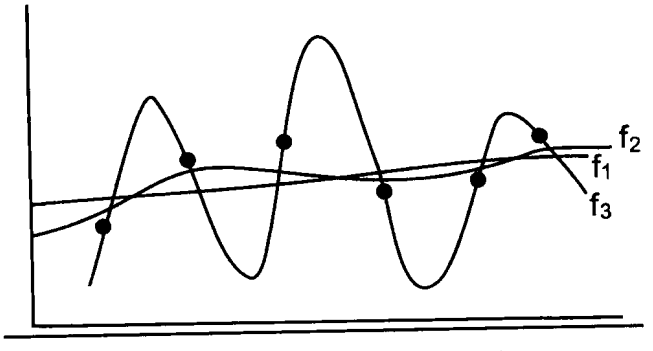


图 16-2　一组数据点和 3 个函数近似

我们把这个泛化问题放回到认识论环境中来结束讨论。当问题求解程序为求解过程生成和使用表示法（无论是符号、网络结点还是任何其他方法）时，它们是在问题或者求解领域中创建不变性或者不变的系统，因此，这些泛化结果的质量和清晰度能够为这种努力的最终成功提供关键性的支持。在下一小节中还将继续深入讨论这个问题。

归纳偏置，理性主义者的先验偏置

第 10 章到第 13 章的自动学习技术以及所有的人工智能技术几乎都反映了其创立者的一种先验偏置。归纳偏置问题就是结果表示和搜索策略给我们提供了一种对已经解释了的世界进行编码的媒介。它们很少为我们提供对解释的提问机制、生成新观点的机制或者当它们没有生成时提供回溯和改变看法的机制。这种隐含的偏置将导致理性主义认识论的陷阱，即“看到的世界恰好只是我们期望看见的或我们从训练中看见的”。

归纳偏置的作用在每种学习范型中必须明确。此外，没有被归纳偏置承认并不是说它不存在或者严重地影响学习的参数。在基于符号的学习中，归纳偏置通常是很明显的，例如，使用用于概念学习的语义网。在 Winston（1975a，10.1 节）的学习算法中，偏置包括合取关系表示以及对约束提纯使用“似是而非”的重要性。我们可以看到类似的偏置，比如在 10.1 节的变形空间搜索的谓词使用中，10.3 节的决策树中，或者甚至在 10.5 节的针对 Meta-DENDRAL 的规则中。

正如我们在第 11 章和第 13 章中明白表示的，不管怎样，连接系统和遗传学习行为的许多方面都假定了归纳偏置。例如，感知机网络的局限性导致了隐结点的引入。我们可能会问：隐结点在解决方案的生成中到底有什么作用。理解隐结点作用的方法之一是在表示空间中增加维度。举一个简单的例子，在 11.3.3 节中，针对异或问题的数据点在二维中不是线性可分的。但是，在隐结点上的学习权重为表示提供了另外一个维度。在三维上使用一种二维平面是可以把这些点分开的。给定输入空间的两个维度和隐结点，这个网络的输出层此时就可以看成是一个普通的感知机，它寻找一个能在三维空间中把这些点分开的平面。

一种补充的看法是，许多“不同”的学习范例（有时是不明显的）共享公共的归纳偏置。

我们指出了其中一些：如 10.5 节中的聚类于 CLUSTER/2 的关系，11.2 节的感知机，以及 11.3 节的原型网络。我们指出，逆传以及在 Kohonen 层使用无监督竞争学习而在 Grossberg 层使用监督 Hebbian 学习的对偶网络，它们在很多方面类似于反馈学习。在逆传中，在 Kohonen 层聚类的数据所起的作用类似于由反传的隐结点所学习的泛化。

在许多重要的方法中，我们给出的工具是很类似的。实际上，甚至原型（表示数据聚类）的发现都提供了函数近似的补充情形。在第一种情况下，我们试图对数据集进行分类；在第二种情况，我们将生成能精确地把数据聚类相互分开的函数。当感知机所使用的最小距离分类算法也给出了定义线性分离的参数时，我们将看到上述的情形。

甚至从不同的观点还能看到产生函数的泛化。例如，长期以来，统计学技术能够发现数据之间的相互关系。重复展开的泰勒级数能够用于逼近大多数函数。使用多项式逼近算法逼近拟合数据点的函数已经有一个多世纪的历史了。

总之，对学习模式（不管是基于符号的、连接的、涌现的还是随机的）做出的承诺在很大程度上调解了我们从问题求解中期望的结果。当我们充分了解这些遍及计算问题求解器整个设计过程中的协同结果时，我们通常能够提高我们成功的机会，也能更有见识地解释我们的结果。

经验主义者的困境

如果当前针对机器学习（尤其是有监督学习）的这些方法拥有占优势的归纳偏置，那么包括许多遗传和进化方法的无监督学习不得不抓住相反的问题，这有时被称为*经验主义者的困境*。这些研究领域的一个主题是，解决方案将会出现，替代方案会被进化，一个群体反映了适者生存。这是很有力的材料，尤其是在并行和分布式的搜索环境中。但这里有一个问题是：当我们不确定往哪里去时，怎样才能知道我们在哪里？

柏拉图在 2000 多年以前就以奴隶米诺的话提出了这个问题：

> 苏格拉底，你怎样才能对你不知道的东西提问？你将提出什么作为询问的主题？如果你遇到了你要找的，你怎么知道那就是你以前不知道的（Plato 1961）？

一些研究者们已经举例说明米诺是正确的，具体见 Mitchell（1997）和 Wolpert 与 Macready（1995）的《没有免费的午餐》。事实上，经验主义者需要理性主义者先验的残余来拯救科学。

尽管如此，关于学习的无监督模型和进化模型，仍然有令人兴奋的东西，例如，在基于例子和能量最小化网络创建中，它可以看作是针对复杂相关不变性的不动点吸引子或吸引域。当数据点向吸引子“下陷”时，我们进行观察，并将这些新结构看成是对动态现象建模的工具。这时我们可能会问，在这些范例中计算有什么限制？

事实上，研究者们已经证明（Siegelman and Sontag 1991）再现网络是计算完备的，即它与图灵机类型等价。这种图灵等价性扩展了早期的结果：Kolmogorov（1957）指出对于任意的连续函数，都存在一个能计算该函数的神经网络。同时还证明，具有一个隐藏层的反馈网络能够逼近一类更严格的连续函数中的任何函数（Hecht-Nielsen 1989）。同样我们在 11.3 节中也看到，冯·诺依曼建立的有限状态自动机也是图灵完备的。因此，连接网络和有限状态自动机看来是两种实质上能够计算任何可计算函数的算法，当然这种算法不止两种。此外，归纳偏置确实能应用于无监督学习以及遗传和涌现的学习模型；表示偏置应用于结点、网络和基因组的设计，算法偏置则应用于搜索算子、奖赏算子和选择算子。

那么什么是无监督学习器，不管是连接的、遗传的还是进化的有限状态机的各种形式，它们能提供这种学习器吗？

1）连接学习中最引人注意的特征是大部分的模型是数据或者实例驱动的。也就是说，尽管

它们的结构是显式设计的，但它们通过实例学习，从特定问题领域的数据中进行概括和泛化。但是这里仍然存在问题，即数据是否充分或者足够纯净而不至于干扰求解过程。设计者如何才能知道这些？

2）遗传算法同时也支持一种很强的、富有弹性的问题空间搜索。遗传搜索既是由变异算子所施加的多样性所驱动，同时也是由交叉和转置算子等（它们为下一代保留了父辈信息的重要特征）所驱动。程序设计者如何才能够保留和培养这种多样性/保留性的折中？

3）遗传算法和连接体系结构可以被视为并行和异步处理的实例。它们确实通过并行异步处理提供了用显式的顺序程序设计不能获得的结果吗？

4）对于许多现代的连接和遗传学习的实践者来说，尽管神经的和社会学的启示并不是很重要的，但是这些技术确实反应了自然进化和选择的许多重要方面。我们看到使用感知机、反传和 Hebb 模型来进行错误归约学习的模型，我们也可以在 11.3.4 节看到自联想 Hopfield 网。多种进化模型可以见第 12 章的范例。

5）最后，所有的学习范例都是按经验进行探索的工具。我们在捕获世界的不变性时，当问到关于感知机、学习和理解的更为深入的问题时，我们的工具是否足够强大和富有表达力？

在下一节，我们提出一种与现代人工智能的实验方法相结合的构造论者的认识论，为继续构造和探索智能系统科学提供了工具和技术。

构造论者的一致观点

理论就像网一样，他不断地撒网，捕获……

——L. Wittgenstein

构造论者假设：所有理解都是世界中能量模式与由智能主体强加在世界上的智力类别之间进行交互的结果（Piaget 1954，1970；von Glasersfeld 1978）。使用 Piaget 的描述，我们依据我们当前的理解吸收外部现象并把我们的理解适应于现象的“需要”。

构造论者经常使用表示模式（schemata）这个术语来描述那些用来组织外部世界的经验的先验结构。这个术语源自于英国心理学家 Bartlett（1932），其哲学根源要追溯到康德（Kant，1781/1964）。基于这样的观点，观察不是被动的、中立的，而是主动的、解释性的。

感知到的信息，即康德所说的后天性知识，根本不能准确地适合于我们的预想和先验结构的表示模式。由于这种情况的牵引，主题用来组织经验的基于表示模式的偏置可以既被修改和扩展也可以被替换。面对与环境不成功的交互而产生的对适应性的需求驱动着认知平衡的进程。因此，构造论的认识论根本上是一种认知进化和精炼。构造论的一个重要结果是任意情形的解释包括了对观察者对实现世界的概念和分类的过分要求（归纳偏置）。

当 Piaget（1954，1970）提出了一种针对理解的构造论方法时，他把它称为遗传认识论。由于缺乏对世界的合适表示模式，这形成了一种认知的牵引力。这种牵引力驱动了表示模式修订的进程。表示模式的修订（Piaget 的适应性）说明对趋于平衡的理解的不断进化。

趋向于平衡的图式修订和移动是一种遗传天性和一种针对社会结构和外部世界的适应性。它同时综合了这两种力量，并表现求生存所体现的天性。表示模式修改既是遗传天性的先验结果，也是社会与世界的后天机能。它反映了一个具体化的时间和空间上的生存驱动的主体，一个生物。

这里存在着经验主义和理性主义的混合，以主体生存目标为中介。作为具体体现，除了知道哪一个首先通过了感觉器官，其他的我们一无所知。作为适应性，我们通过学习外部世界的一般模式而生存。我们所感知到的东西是通过我们所期望的东西来传达的，而我们所期望的是受我们感知的东西所影响的。也就是说，这两种功能只有通过相互理解才能发挥作用。在这种随机模

型上——贝叶斯模型和马尔可夫模型——是适应的，就如同是目前阐明的先验经验条件（Doya et al. 2007）。

最后，作为主体，我们很少有意识地察觉到支持我们与世界进行交互的表示模式。作为同时在科学和社会中的倾向和偏见的来源，我们通常都不能意识到它们的内容。这些是我们与世界平衡的本质，而（通常）不是我们有意识的智力生命的可感觉的元素。

为什么构造论者的认识论对于人工智能的问题是特别有用的呢？环境中的主体如何理解它自己对周围情况的理解呢？我们相信构造论有助于解决哲学和心理学的认识论的访问问题。一个多世纪以来，这两个学科的两个学派之间一直存在着斗争，实证主义派别主张从观察到的物理行为中对智能现象进行推理，而现象学派则允许使用第一人称的报告来了解心理现象。这种学派存在的原因是两种了解心理现象的方式都要求一定形式的推理或者构造。

与那些似乎可以直接简单观察到的物理对象（像椅子和门）相比，一个主体的智力状态和分布似乎很难来加以刻画。实际上，我们相信在直接访问物理现象和间接访问智能现象之间截然分开是不妥的。构造论者分析表明，如果不使用一些用来组织经验的模型或表示模式，则没有什么经验是可能的。在科学探索中以及我们人类的经验中，这就意味着所有对现象的了解都要通过探索、近似和不断的模型改进。

人工智能研究者的研究工作是什么

作为从事人工智能的研究者，我们是构建主义者。我们建立、测试和改进模型。然而，在建模活动中，我们打算逼近什么？下面我们将讨论这个问题，首先让我们来做一个简单的认识论观察：人工智能问题求解器不是要抓住“事物外部”的本质，而是更适合来模拟模型的建立、精化以及智能主体自身启发式的平衡。

只有极端唯我论者否认独立于主观之外的世界的真实性。但这种所谓的“真实世界”到底是什么样的？除了“硬件”和“软件”的复杂混合之外，它还是一个包含原子、分子、夸克、引力、相互作用、细胞、DNA 以及（甚至可能）超弦的系统。所有这些概念都是试探性的模型，由平衡驱动主体的解释性需求驱动。同样，这些试探性的模型不是关于外部世界的。而是，它们抓住社会性智能主体的动态平衡张力，是在空间和时间中进化和不断自我校准的实质智能。

对“真实”的访问和创建也是通过主体约定获得的。具体的主体通过存在主义的确认来创建真实，即感知到的期望模型好用到足以让它进行某些实际需要和目标。这个约定使得主体在具体和社会的环境中使用的符号和符号系统有了基础。这些构建具有了基础，因为可以确定它们对于获得目标来说足够好用。这一基础也可从主体语言的使用中看出来。Searle（1969）在演讲中将现象（phenomena）纠正为动作（acts）。基础问题是为什么计算机在智能表示方面有根本性问题，包括在语言和学习方面的实证。给计算机什么样的部署才能赋予它正确的目的和目标呢？虽然 Dennett（1987）把这些归咎于计算机需要和使用“智能”来解决问题，但缺乏足够的基础，这在计算机的简化、脆弱和有限的环境评估能力中是显而易见的。

有生命的主体对符号的使用和依赖意味着更多含义。人类主体的具体和社会背景中的特殊部分将主体间的交互和外部世界联系起来。听觉和视觉系统感觉特定带宽的声波和光波；作为具有胳膊、腿、手的直立两足动物来观察世界；生活在有天气、季节、太阳和黑暗的世界中；是社会的一部分，有发展的目标和目的；每个个体出生、繁殖和死亡；这些都是支持理解、学习和语言的关键成分；这些关系到艺术、生命和爱的表达。

能不能让我来把你比作夏日？
你可是更加温和，更加可爱：
狂风会吹落五月里开的好花儿，

夏季的生命又未免结束得太快……

莎士比亚，十四行诗（第十八首）

在结束本书时，我们小结了一下人工智能的关键问题，这些问题同时支撑与界定我们当前在推进智能系统科学中所展开的工作。

16.3　人工智能：当前的挑战和未来的方向

如同一位几何学家倾注全部心血，来把那圆形测定，他百般思忖，也无法把他所需要的那个原理探寻，我此刻面对那新奇的景象也是这种情形……

——但丁，《神曲·天堂篇》

虽然使用人工智能技术来解决实际问题说明了它的实用性，但使用这些技术来建立智能的通用科学仍是一个困难的、需要继续研究的问题。在本节，我们将回到把我们引入到人工智能领域并写了本书的那个问题：能否对可以导致智能的过程给出一个形式化的、可计算的说明？

智能的计算特性开始于对计算设备的抽象规范说明。20 世纪 30 年代到 50 年代的研究开启了这一探索，图灵、波斯特、马尔可夫、丘奇等人在对计算的形式化描述方面做出了极大贡献。这些研究的目的不是仅仅指出计算的含义，而是还指出有关什么可以被计算方面的限制。通用图灵机（Turing 1950）是研究得最为普遍的规范说明，但是作为产生式系统计算（Post 1943）基础的波斯特重写规则也有很大的贡献。基于部分递归函数的丘奇模型（1941）是一个重要的成果，支持了现代高级函数式语言，例如 Scheme、Ocaml 和 Standard ML。

理论学家们已经证明，所有这些形式方法都具有等价的计算能力，即一种形式方法可计算的任意函数也能被其他形式方法所计算。事实上，可以说通用图灵机等价于现代的任意计算设备。基于这些结果，丘奇 - 图灵命题给出了更为有力的论据，即没有哪个计算模型能够定义得比这些已知模型更强。一旦我们建立了计算规范的等价性，我们就从这些规范的机械化工具中解放出来：我们可以用电子管、硅芯片、细胞质或者普通玩具实现我们的算法。在一种媒介上的自动设计机制等价于在另外一种媒介上设计的机制。因此当我们在一种媒介上来测试在另一种媒介上实现的机制时，这就使得经验式探索的方法变得更加重要。

有一种可能就是图灵和波斯特的通用机器也许太泛化、太通用了。这里矛盾的是，智能可能并不需要很强的带集中控制的计算机制。Levesque 和 Brachman（1985）建议人的智能可能需要更多的计算性的有效的表示（比如用于推理的 Horn 子句）、对基础文字（ground literal）的实际知识的约束以及可计算跟踪的真值维护系统的使用。智能的基于主体的模型和涌现模型似乎也支持这种哲学。

由我们的机制模型的形式化等价性所引出的另一个论点是，二元性问题和心身（mind-body）问题。从笛卡儿时代（见 1.1 节）以后，哲学家们就提出了智能、意识和身体之间的交互和整合问题。他们给出了每种可能的反映，从完全的唯物主义到对物质存在的否定，甚至到支持上帝的介入！人工智能和认知科学的研究否认了笛卡儿的二元论，而支持基于物理实现或者符号实例的智能的物质模型，支持管理这些符号的计算机制的形式化规范，支持表示范例的等价性，支持在具体模型中知识和技能的机械化。这种研究的成功表明了这种模型的有效性（Johnson-Laird 1988，Dennett 1987，Luger et al. 2002）。

关于物理系统中智能的认识论基础，仍然还有许多问题。我们再次总结了以下几个重要问题：

1）**表示问题**。纽维尔和西蒙假定物理符号系统和搜索对于智能的特性是充分必要的（见 16.1 节）。神经模型或子符号模型的成功、智能的遗传和涌现方法的成功是否是对物理符号假设

的一种驳斥，或者它们是这种假设的简单实例吗？

连这种假设的弱解释——物理符号系统是智能的一个充分模型——在现代认知科学领域中也产生了许多强大的、有用的结果。这种观点认为，我们可以实现那些能说明智能行为的物理符号系统。充分性使得我们能够为人所具有的许多方面的性能创建和测试基于符号的模型（Pylyshyn 1984，Posner 1989）。但是这个假设的强解释——物理符号系统和搜索对于智能活动是必要的——仍然是个有待研究和解决的问题（Searle 1980，Weizenbaum 1976，Winograd and Flores 1986，Dreyfus 1985，Penrose 1989）。

2）**认知中具体化的作用**。物理符号系统假设的主要假定之一就是，物理符号系统的特定实例化是与其性能无关的；其主要内容是其形式化结构。许多研究者都对这一点提出了挑战（Searle 1980，Johnson 1987，Agre and Chapman 1987，Brooks 1989，Varela et al. 1993），他们指出智能行为的需求要求一种允许主体整合到世界中的物理具体化。现代计算机的结构并不允许这种程度的情形，而是要求一个人工智能通过极端有限的窗口（同时代的输入输出设备）来同世界进行交互。如果这种挑战是正确的，则尽管出现机器智能，它仍需要同时代的计算机提供一个非常相同的接口（对这个主题的更多解释见15.0节自然语言理解中的问题和16.2.2节认识论约束）。

3）**文化与智能**。传统上，人工智能侧重于把个体智能作为智能的来源；我们的行动好像在说，对于大脑编码和怎样管理知识的方法的解释可能是原始智能的一种完整解释。然而，我们也会认为知识最好被看作是基于社会的，而不是一个个体所构造的。在基于记忆的智能理论中（Edelman 1992），社会本身也带有智能的本质组件。对于智能理论来说，对知识的社会环境和人类行为的理解，同对个体智能/大脑的理解，是同等重要的。

4）**刻画解释的本质**。在表示传统研究中，大多数模型一般工作在已经解释好的领域中：即对于解释的上下文，系统设计者通常都会给出一些隐含的、先验约定，在这种约定下，很难随着问题求解过程的进展而将上下文、目标或表示进行转换。目前还很少有成果能够阐明人类构造解释的过程。

Tarskian的观点，即将语义作为符号和对象之间的映射，还是太弱并不能解释一些事实，如一个领域在不同实践目标的指引下可能会有不同的解释。语言学家试图通过语用理论（Austin 1962）来弥补Tarskian语义的局限性。论述分析，基本依赖于上下文中符号的使用，已经在近几年中广泛地讨论了这些问题，但是这个问题涉及的内容事实上还要更加广泛，因为它通常还要处理参考工具的失败（Lave 1988，Grosz and Sidner 1990）。

C. S. Peirce（1958）最先倡导符号语言学的传统，后续的研究者还有Eco、Seboek以及其他学者（Eco 1976，Grice 1975，Sebeok 1985），他们对于语言采用了更为激进的方法。这种符号语言学的传统把符号表达式放在广泛的记号和记号解释中，它表明，符号的含义只有在它用做解释的上下文中才能够被理解，即在解释的上下文中或在与环境的交互中才能被理解（见16.2.2节）。

5）**表示的不确定性**。Anderson的表示不确定性猜想（Anderson 1978）指出，在熟练性能的特定动作这种环境下，确定哪种表示模式最接近于人的问题求解器在理论上是不可能的。这种猜想是基于这样的事实，即每个表示模式不可避免地被连接到一个大型的计算结构，就像搜索策略一样。在对人类技能的详细分析中，我们不太可能充分地控制这个过程，使得我们能决定这个表示；也不可能为过程被惟一确定的那些点建立一个表示。由物理的不确定原理，现象可以通过检验这个现象的过程来加以改变，因此，构造智能模型是需要重点关注的，但没有必要限制它们的利用。

但更为重要地，计算模型本身也受到同样的批评，其中符号的归纳偏置和丘奇/图灵假设中的搜索仍未约束一个系统。当科学家简单地要求足够健壮的模型来约束经验问题时，一些优化的表示模式的感知需要可能是理性主义梦想的残留物。一个模型能给出其解释、能预言和能被修改，同样也能够对模型的质量进行证明。

6）**设计可以反证的计算模型的必要性**。Popper（1959）等人指出科学理论必定是可以反证的，这就是说，必定存在一种环境，使得在此环境下的这个理论模型并不是对这种现象的成功的近似。明显的原因是，任何数目的确定性实验实例都不能充分地确定一个模型。许多研究是在已有理论的失败的基础上进行的。

物理符号系统假设的一般本质正如智能的情景模型和涌现模型一样，可能会使它们不可能被歪曲，并作为一种模型在使用上受到限制。同样地，可以对关于现象学传统的假设进行批评（见第7点）。一些人工智能数据结构（比如语义网）还是很普通的，使得它们可以建模几乎所有能够描述出的东西，或者正像通用图灵机一样，使得它们可以建模任意的可计算函数。因此，一个人工智能研究者或者认知科学家被问到在什么条件下他们的智能模型不能用时，给出答案经常是很难的。

7）**科学方法的局限性**。许多研究者们（Winograd and Flores 1986，Weizenbaum 1976）宣称智能的最重要方面就是没有被模型化，并且原则上不可能被模型化，且特别是不能使用任意的符号表示来模型化。这些领域包括学习、自然语言理解、说话动作的产生等。这些问题已经深深地植根于我们的哲学理念中。例如，Winograd 和 Flores 的批评是基于现象学方面的问题（Husserl 1970，Heidegger 1962）。

现代人工智能的大部分假设追其根源，可以回溯到 Carnap、Frege 和 Leibniz，再远回溯到 Hobbes、Locke 和 Hume，直至回溯到亚里士多德。这种传统观点认为：智能的处理过程符合通用法则，并且在原则上是可以理解的。

Heidegger 和他的追随者们描述了一种可选择的方法来理解智能。对于 Heidegger 来说，思考的意识是源于具体经验的世界（一种生命世界）。Winograd、Flores 和 Dreyfus 等人认为一个人对事物的理解是扎根于在每天的世界中“使用”这些理解的实际活动中。这种世界在本质上是一种环境，其中包括按社会方式组织的各种作用和目的。而这种环境以及其中的人的功能不是通过命题解释的，也不是能够被定理所理解的。它更像是一种不断形成的流程。在基本意义上，人类专家并不知道“是什么”，而只是知道在进化的社会标准和隐含的目的不断发展的世界中，它是怎么样的。我们不能自然地就把我们的知识和大多数智能行为放入语言中，不管是形式的，还是自然的。

现在让我们来考虑上述这种观点。首先，作为对纯理性主义传统的批判，这种观点是正确的。理性主义者断言，所有的人类活动、智能和责任，至少原则上能够被表示、形式化和理解。大多数喜欢思考的人们并不相信这种情形，他们认为情感、自我主张和有责任的承诺等（至少）也是很重要的。亚里士多德在他的《论理性行为》（Essay on Rational Action）一书中说，“为什么我并没有觉得我是被迫执行那些必须做的事情呢?”在科学方法的领域之外，还有很多人类活动在可靠的人类交互中起着本质的作用。这些不可能被机器再生或者取消。

然而，检查数据、构造模型、运行实验以及为了进一步实验而使用模型精炼来检查结果等这些科学传统已经进入了理解、解释和预言人类社会能力这样的一个重要的水平。科学方法是提高人类理解能力的一个有力工具。尽管如此，对于这种方法，这里仍然有许多的告诫是科学家们必须理解的。

首先，科学家们不要把这个模型与被建模的现象相混淆。模型能允许我们不断地逼近这种

现象；通常这里必然有一些不能使用经验解释的“残留物”。在这种意义上，表示不确定性并不是一个问题。一个模型是用来探索、解释和预言的；如果它允许科学家来完成这些任务，则它就是成功的（Kuhn 1962）。确实，对于一个简单的现象，不同的模型可以用来解释这种现象的不同方面，例如光的波动理论和粒子理论（光的波粒二相性）。

此外，当研究者们主张智能现象的各个方面已经在科学传统的范围和方法之外时，这种说法本身也只能用那些科学传统来验证。科学方法只是一种工具，它可以用来解释在什么意义上问题仍然是在我们当前的理解之外。每种观点，甚至是来自于现象学传统的观点，如果它是有一定含意的，那么它一定跟我们当前某些解释的概念相关，甚至它是与那些不能解释的现象相关联的。

人工智能研究中最让人振奋的方面是对我们必须解决的这些问题做出不懈的努力和贡献。为了理解问题求解、学习和语言，我们必须领会表示和知识的哲学层面含义。我们被要求用一种谦卑的方式来解决亚里士多德的理论和实践之间的关系问题，以形成理解和实践的统一、理论和实践的统一，在科学与艺术中生活。

人工智能工作者是工具的制造者。我们的表示、算法和语言都是一些工具，用来设计和建立那些展现智能行为的机制。通过实验，我们同时检验了它们解决问题的计算适合性，也检验了我们自己对智能现象的理解。

确实，我们有这些传统：笛卡儿、莱布尼兹、培根、帕斯卡、霍布斯、巴贝奇、图灵等都是做出重要贡献的人，见第 1 章。工程、科学和哲学，思想、知识和技能的本质，形式方法与机制的能力和局限性等，这些课题都是我们所经历的，也是我们需要继续探索和研究的。

16.4 结语和参考文献

读者可以参考第 1 章最后列出的参考文献，另外还增加一些文献：《Computation and Cognition》（Pylyshyn 1984），《Understanding Computers and Cognition》（Winograd and Flores 1986）。关于认知科学方面的问题，请参考 Newell and Simon（1972）、Pylyshyn（1973，1980）、Norman（1981）、Churchland（1986）、Posner（1989）、Luger（1994）、Franklin（1995）、Ballard（1997）、Elman et al.（1996）以及 Jeannerod（1997）。

Haugeland（1981，1997）、Dennett（1978，2006）和 Smith（1996）描述了智能系统科学的哲学基础。Anderson（1990）关于认知心理学的书给出了关于信息处理模型的有价值的实例。Pylyshyn（1984）和 Anderson（1978，1982，1983a）给出了认知科学中许多关键问题的详细描述，包括对表示不确定性的讨论。Dennett（1991）将认知科学的方法论应用到意识结构本身的探索中。我们也给读者推荐科学的哲学方面的文献（Popper 1959，Kuhn 1962，Bechtel 1988，Hempel 1965，Lakatos 1976，Quine 1963）。

最后，《Philosophy in the Flesh》（Lakoff and Johnson 1999），对这些基础问题给出了一些可能的解答。《The Embodied Mind》（Varela et al. 1993）、Suchman（1987）和《Being There》（Clark 1997）描述了体现智能的重要方面。

给读者两个重要组织的地址信息：

The American Association for the Advancement of Artificial Intelligence（美国人工智能促进学会）

445 Burgess Drive

Menlo Park，CA 94025

Computer Professionals for Social Responsibility（计算机专业的社会责任）

P. O. Box 717

Palo Alto，CA 94301