第 1 章　绪　　论

本章我们试图解释为什么我们认为人工智能是一个最值得学习的学科，并试图让学生们对人工智能的定义有一个简单的认识，从而开启人工智能的学习之旅。

1 人工智能基本概念

1.1 人工智能的定义

众所周知，相对于天然河流（如亚马逊河和长江），人类开凿了叫做运河（如苏伊士运河和中国大运河）的人工河流；相对于天然卫星（如地球的卫星——月亮），人类制造了人造卫星；相对于天然纤维（如棉花、蚕丝和羊毛），人类发明了维尼纶和涤纶等人造纤维；相对于天然心脏、天然婴儿、自然受精和自然四肢等，人类创造了人工心脏、试管婴儿、人工授精和假肢等人造物品（artifacts）……2009 年 7 月 8 日，英国一个科学研究小组宣布首次成功利用人类干细胞培育出成熟精子。这就是人工精子，一种很高级的人工制品。我们要探讨的人工智能（artificial intelligence)，又称为机器智能或计算机智能。由此可见，人工智能本质上有别于自然智能，是一种由人工手段模仿的人造智能；至少在可见的未来应当这样理解。

像许多新兴学科一样，人工智能至今尚无统一的定义，要给人工智能下个准确的定义是困难的。人类的自然智能（人类智能）伴随着人类活动时时处处存在。人类的许多活动，如下棋、竞技、解算题、猜谜语、进行讨论、编制计划和编写计算机程序，甚至驾驶汽车和骑自行车等，都需要“智能”。如果机器能够执行这种任务，就可以认为机器已具有某种性质的“人工智能”。不同科学或学科背景的学者对人工智能有不同的理解，提出不同的观点，人们称这些观点为符号主义(symbolism)、连接主义(connectionism)和行为主义(actionism)等，或者叫做逻辑学派（logicism）、仿生学派（bionicsism)和生理学派（physiologism）。在 1.2 节将综述他们的基本观点。

哲学家们对人类思维和非人类思维的研究工作已经进行了两千多年，然而，至今还没有获得满意的解答。下面，我们将结合自己的理解来定义人工智能。

1.1.1 人工智能的定义

定义 1.1 (智能: intelligence). 人的智能是人类理解和学习事物的能力或者说，智能是思考和理解的能力而不是本身做事的能力。

定义 1.2 (智能机器: intelligence machine). 智能机器是一种能够呈现出人类智能行为的机器，而这种智能行为是人类用大脑考虑问题或创造思想。

另一种定义为：智能机器是一种能够在不确定环境中执行各种拟人任务（anthropomorphic tasks）达到预期目标的机器。

定义 1.3 (人工智能学科). 长期以来，人工智能研究者们认为：人工智能（学科）是计算机科学中涉及研究、设计和应用智能机器的一个分支，它的近期主要目标在于研究用机器来模仿和执行人脑的某些智力功能，并开发相关理论和技术。

近年来，许多人工智能和智能系统研究者认为：人工智能（学科）是智能科学(intelligence science）中涉及研究、设计及应用智能机器和智能系统的一个分支，而智能科学是一门与计算机科学并行的学科。

人工智能到底属于计算机科学还是智能科学，可能还需要一段时间的探讨与实践，而实践是检验真理的标准，实践将做出权威的回答。

定义 1.4 (人工智能能力).人工智能（能力）是智能机器所执行的通常与人类智能有关的智能行为，这些智能行为涉及学习、感知、思考、理解、识别、判断、推理、证明、通信、设计、规划、行动和问题求解等活动。

1950 年图灵（Turing）设计和进行的著名实验（后来被称为图灵实验，Turing test)，提出并部分回答了“机器能否思维”的问题，也是对人工智能的一个很好注释。

为了让读者对人工智能的定义进行讨论，以更深刻地理解人工智能，下面综述其他几种关于人工智能的定义。

定义 1.5 人工智能是一种使计算机能够思维，使机器具有智力的激动人心的新尝试[Haugeland, 1985]。

定义 1.6 人工智能是那些与人的思维、决策、问题求解和学习等有关活动的自动化[Bellman, 1978]。

定义 1.7 人工智能是用计算模型研究智力行为[Charniak and McDermott, 1985]。

定义 1.8 人工智能是研究那些使理解、推理和行为成为可能的计算[Winston, 1992]。

定义 1.9 人工智能是一种能够执行需要人的智能的创造性机器的技术[Kurzwell, 1990]。

定义 1.10 人工智能研究如何使计算机做事让人过得更好[Rick and Knight, 1991]。

定义 1.11 人工智能是研究和设计具有智能行为的计算机程序，以执行人或动物所具有的智能任务[Dean et al., 2003]。

定义 1.12 人工智能是一门通过计算过程力图理解和模仿智能行为的学科[Schalkoff, 1990]。

定义 1.13 人工智能是计算机科学中与智能行为的自动化有关的一个分支[Luger and Stubblefield, 1997]。

其中，**定义 1.5** 和**定义 1.6** 涉及拟人思维；**定义 1.7** 和**定义 1.8** 与理性思维有关；**定义 1.9**~**定义 1.11** 涉及拟人行为；**定义 1.12** 和**定义 1.13** 与理性行为有关。

1.1.2 图灵测试

1950 年，英国数学家 Alan Turing 提出了一个测试方法来确定一个机器能否思考。该方法需要两个人对机器进行测试，其中，一人扮演提问者；另一人作为被测人员。这两个人与机器分别处在 3 个不同的房间，提问者通过打印问题和接收打印问题来与被测人员和被测机器进行通信。提问者可以向被测机器和被测人员提问，但他只知道接受提问的是 A 或 B，而不知道被测试者是人还是机器，并试图确定谁是机器、谁是人。这个测试后来被人们命名为“图灵测试”。

在“图灵测试”中，如果人的一方不能区分对方是人还是机器，那么就可以认为那台机器达到了人类智能的水平。要通过这个测试，那么它的目标就是要使得提问者误认为它是人。因此，有时机器要故意伪装一下，如当提问者问“12324 乘 73981 等于多少？”时，机器应等几分钟回答一个有点错误的答案，这样才更显得像人在计算。当然，一台机器要通过图灵测试主要的是它具有的知识总量和具有大量的人的基本常识。

1.2 人工智能的发展历史

不妨按时期来说明国际人工智能的发展过程，尽管这种时期划分方法有时难以严谨，因为许多事情可能跨接不同时期，另外一些事件虽然时间相隔甚远但又可能密切相关。

1.2.1 人工智能起源

无论是计算机科学还是它的分支——人工智能，其最初的理论基础都归功于图灵。图灵是英国数学家、逻辑学家，1912 年生于伦敦少年时就表现出独特的直觉创造能力和对数学的爱好，被称为计算机科学之父，人工智能之父。

1936 年 5 月，图灵向伦敦权威的数学杂志投了一篇论文，题为《论数字计算在决断难题中的应用》。该文于 1937 年在《伦敦数学会文集》第 42 期上发表后，立即引起广泛的注意。图灵在这篇文章中定义了后来被他的导师丘奇称为“图灵机”的计算装置：一条无穷长的纸带，一个读写头在一个控制装置的控制下在纸带上方左移右移，读取纸带上的内容并在纸带上写 0 或 1，并由此有了所谓“丘奇-图灵论题”，也就是说任何计算装置都等价于图灵机。这个论题不是数学定理，但却是整个计算机科学的基础。

有了“图灵机”，我们就很容易把原来是纯逻辑或纯数学的东西（例如递归函数和 λ 演算等）和物理世界联系起来了，函数成了纸带和读写头。如果按照传统哲学的说法：数学唯心，理性；物理唯物，经验，那么计算机科学就是唯物和唯心的桥梁，司马贺把这类东西统统叫作“人工科学”。

1948 年，图灵在英国国家物理实验室的内部报告中区分了“肉体智能”和“无肉体智能”的概念；1950 年，图灵进一步提出关于机器思维的问题，他的论文《计算机和智能》，引起了广泛的注意和深远的影响。

1950 年 10 月，图灵发表论文《机器能思考吗？》这一划时代的作品，使图灵赢得了“人工智能之父”的桂冠。

1956 年 8 月，在美国汉诺斯小镇宁静的达特茅斯学院中，约翰·麦卡锡、马文·闵斯基（人工智能与认知学专家）、克劳德·香农（信息论的创始人）、艾伦·纽厄尔（计算机科学家）、赫伯特·西蒙（诺贝尔经济学奖得主）等科学家聚集在一起，讨论着一个完全不食人间烟火的主题：用机器来模仿人类学习以及其他方面的智能。会议足足开了两个月的时间，虽然大家没有达成普遍的共识，但是却为会议讨论的内容起了一个名字：人工智能。因此，1956 年也就成为了人工智能元年。

1.2.2 人工智能发展阶段

1、孕育时期（1956 年前）

人类对智能机器和人工智能的梦想和追求可以追溯到三千多年前。早在我国西周时代（公元前 1066——前 771 年），就流传有关巧匠偃师献给周穆王一个歌舞艺伎的故事。作为第一批自动化动物之一的能够飞翔的木鸟是在公元前 400 年——前 350 年间制成的。在公元前 2 世纪出现的书籍中，描写过一个具有类似机器人角色的机械化剧院，这些人造角色能够在宫廷仪式上进行舞蹈和列队表演。我国东汉时期（公元 25——220 年），张衡发明的指南车是世界上最早的机器人雏形。

我们不打算列举三千多年来人类在追梦智能机器和人工智能道路上的万千遐想、实践和成果，而是跨越三千年转到 20 世纪。时代思潮直接帮助科学家去研究某些现象。1913 年，年仅 19 岁的维纳（Wiener）在他的论文中把数理关系理论简化为类理论，为发展数理逻辑作出贡献，并向机器逻辑迈进一步，与后来图灵（Turing）提出的逻辑机不谋而合。1948 年维纳创立的控制论（cybernetics），对人工智能的早期思潮产生了重要影响，后来成为人工智能行为主义学派。数理逻辑仍然是人工智能研究的一个活跃领域，其部分原因是由于一些逻辑-演绎系统已经在计算机上实现过。不过，即使在计算机出现之前，逻辑推理的数学公式就为人们建立了计算与智能关系的概念。

丘奇（Church）、图灵和其他一些人关于计算本质的思想，提供了形式推理概念与即将发明的计算机之间的联系。在这方面的重要工作是关于计算和符号处理的理论概念。1936 年，年仅 26 岁的图灵创立了自动机理论（后来人们又称为图灵机），提出一个理论计算机模型，为电子计算机设计奠定了基础，促进了人工智能，特别是思维机器的研究。第一批数字计算机（实际上为数字计算器）看来不包含任何真实智能。早在这些机器设计之前，丘奇和图灵就已发现，数字并不是计算的主要方面，它们仅仅是一种解释机器内部状态的方法。被称为人工智能之父的图灵，不仅创造了一个简单、通用的非数字计算模型，而且直接证明了计算机可能以某种被理解为智能的方法工作。

事过 20 年之后，道格拉斯·霍夫施塔特(Douglas Hofstadter)在 1979 年写的《永恒的金带》（An Eternal Golden Braid）一书对这些逻辑和计算的思想以及它们与人工智能的关系给予了透彻而又引人入胜的解释。

麦卡洛克（McCulloch）和皮茨（Pitts）于 1943 年提出的“拟脑机器”（mindlike machine)是世界上第一个神经网络模型（称为 MP 模型），开创了从结构上研究人类大脑的途径“神经网络连接机制”，后来发展为人工智能连接主义学派的代表。

值得一提的是控制论思想对人工智能早期研究的影响。正如艾伦·纽厄尔(Allen Newell)和赫伯特·西蒙（Herbert Simon）在他们的优秀著作《人类问题求解》（Human Problem Solving）的“历史补篇”中指出的那样，20 世纪中叶人工智能的奠基者们在人工智能研究中出现了几股强有力的思潮。维纳、麦卡洛克和其他一些人提出的控制论和自组织系统的概念集中讨论了“局部简单”系统的宏观特性。尤其重要的是，1948 年维纳发表的《控制论——关于动物与机器中的控制与通信的科学》，不但开创了近代控制论，而且为人工智能的控制论学派（即行为主义学派）树立了新的里程碑。控制论影响了许多领域，因为控制论的概念跨接了许多领域，把神经系统的工作原理与信息理论、控制理论、逻辑以及计算联系起来。控制论的这些思想是时代思潮的一部分，而且在许多情况下影响了许多早期和近期人工智能工作者，成为他们的指导思想。

从上述情况可以看出，人工智能开拓者们在数理逻辑、计算本质、控制论、信息论、自动机理论、神经网络模型和电子计算机等方面做出的创造性贡献，奠定了人工智能发展的理论基础，孕育了人工智能的

胎儿。人们将很快听到人工智能婴儿呱呱坠地的哭声，看到这个宝贝降临人间的可爱身影！

2、形成时期（1956——1970 年）

到 20 世纪 50 年代，人工智能已躁动于人类科技社会的母胎，即将分娩。1956 年夏季，由年轻的美国数学家和计算机专家麦卡锡（McCarthy）、数学家和神经学家明斯基（Minsky）、IBM 公司信息中心主任（Lochester）以及贝尔实验室信息部数学家和信息学家香农（Shannon）共同发起，邀请 IBM 公司莫尔（More）和塞缪尔（Samuel）、MIT 的塞尔夫里奇(Selfridge)和索罗蒙夫(Solomonff)，以及兰德公司和 CMU 的纽厄尔和西蒙共人，在美国的达特茅斯（Dartmouth）大学举办了一次长达两个月的十人研讨会，认真达烈地讨论用机器模拟人类智能的问题。会上，由麦卡锡提议正式使用“人工智能”这一术语。这是人类历史上第一次人工智能研讨会，标志着人工智能学科的诞生，具有十分重要的历史意义。这些从事数学、心理学、信息论、计算机科学和神经学研究的杰出年轻学者，后来绝大多数都成为著名的人工智能专家，为人工智能的发展作出了重要贡献。

最终把这些不同思想连接起来的是由巴贝奇（Babbage）、图灵、冯．诺依曼（von Neumann）和其他一些人所研制的计算机本身。在机器的应用成为可行之后不久，人们就开始试图编写程序以解决智力测验难题、数学定理和其他命题的自动证明，下棋以及把文本从一种语言翻泽成另一种语言。这是第一批人工智能程序。对于计算机来说，促使人工智能发展的是什么？是出现在早期设计中的许多与人工智能有关的计算概念，包括存储器和处理器的概念、系统和控制的概念，以及语言的程序级别的概念。不过，引起新学科出现的新机器的唯一特征是这些机器的复杂性，它促进了对描述复杂过程方法的新的更直接的研究（采用复杂的数据结构和具有数以百计的不同步骤的过程来描述这些方法）。

1965 年，被誉为“专家系统和知识工程之父”的费根鲍姆（Feigenbaum）所领导的研究小组，开始研究专家系统，并于 1968 年研究、功第一个专家系统 DENDRAL，用于质谱仪分析有机化合物的分子结构。后来又开发出其他一些专家系统，为人工智能的应用研究做出了开创性贡献。

1969 年召开了第一届国际人工智能联合会议(International Joint Conference on AI，IJCAI)，标志着人工智能作为一门独立学科登上国际学术舞台。此后，IJCAI 每两年召开一次。1970 年《人工智能国际杂志》（International Journal of AI）创刊。这些事件对开展人工智能国际学术活动和交流、促进人工智能的研究和发展起到了积极作用。

上述事件表明，人工智能经历了从诞生到成人的热烈（形成）期，已形成为一门独立学科，为人工智能建立了良好的环境，打下了进一步发展的重要基础。虽然人工智能在前进的道路上仍将面临不少困难和挑战，但是有了这个基础，就能够迎接挑战，抓住机遇，推动人工智能不断发展。

3、暗淡时期（1966-1974 年）

在形成期和后面的知识应用期之间，交叠地存在一个人工智能的暗淡（低潮）期。在取得“热烈”发展的同时，人工智能也遇到一些困难和问题。

一方面，由于一些人工智能研究者被“胜利冲昏了头脑”，盲目乐观，对人工智能的未来发展和成果做出了过高的预言，而这些预言的失败，给人工智能的声誉造成重大伤害。同时，许多人工智能理论和方法未能得到通用化和推广应用，专家系统也尚未获得广泛开发。因此，看不出人工智能的重要价值。追究其因，当时的人工智能主要存在下列三个局限性：

（1）知识局限性早期开发的人工智能程序包含太少的主题知识，甚至没有知识，而且只采用简单的句法处理。例如，对于自然语言理解或机器翻译，如果缺乏足够的专业知识和常识，就无法正确处理语言，甚至会产生令人啼笑皆非的翻泽。

（2）解法局限性人工智能试图解决的许多问题因其求解方法和步骤的局限性，往往使得设计的程序在实际上无法求得问题的解答，或者只能得到简单问题的解答，而这种简单问题并不需要人工智能的参与。

（3）结构局限性用于产生智能行为的人工智能系统或程序存在一些基本结构上的严重局限，如没有考虑不良结构，无法处理组合爆炸问题，因而只能用于解决比较简单的问题，影响到推广应用。

另一方面，科学技术的发展对人工智能提出新的要求甚至挑战。例如，当时认知生理学研究发现，人类大脑含有 1 个以上神经元，而人工智能系统或智能机器在现有技术条件下无法从结构上模拟大脑的功能。

此外，哲学、心理学、认知生理学和计算机科学各学术界，对人工智能的本质、理论和应用各方面，一直抱有怀疑和批评，也使人工智能四面楚歌。例如，1971 年英国剑桥大学数学家詹姆士（James）按照英国政府的旨意，发表一份关于人工智能的综合报告，声称“人工智能不是骗局，也是庸人自扰”。在这个报告影响下，英国政府削减了人工智能研究经费，解散了人工智能研究机构。在人工智能的发源地美国，连在人工智能研究方面颇有影响的 IBM，也被迫．取消了该公司的所有人工智能研究。人工智能研究在世界范围内陷人困境，处于低潮，由此可见一斑。

任何事物的发展都不可能一帆风顺，冬天过后，春天就会到来。通过总结经验教训，开展更为广泛、深人和有针对性的研究，人工智能必将走出低谷，迎来新的发展时期。

4、 知识应用时期（1970——1988 年）

费根鲍姆(Feigenbaum)研究小组自 1965 年开始研究专家系统，并于 1968 年研究成功第一个专家系统 DENDRAI.01972-1976 年，他们又开发成功 MYCIN 医疗专家系统，用于抗生素药物治疗。此后，许多著名的专家系统，如斯坦福国际人工智能研究中心的杜达（Duda）开发的 PROSPECTOR 地质勘探专家系统，拉特格尔大学的 CASNET 青光眼诊断治疗专家系统，MIT 的 MACSYMA 符号积分和数学专家系统，以及 RI 计算机结构设计专家系统、ELAS 钻井数据分析专家系统和 ACE 电话电缆维护专家系统等被相继开发，为工矿数据分析处理、医疗诊断、计算机设计、符号运算等提供了强有力的工具。在 1977 年举行的第五届国际人工智能联合会议上，费根鲍姆正式提出了知识工程(knowledge engineering)的概念，并预言 20 世纪 80 年代将是专家系统蓬勃发展的时代。事实果真如此，整个 80 年代，专家系统和知识工程在全世界得到迅速发展。专家系统为企业等用户赢得了巨大的经济效益。例如，第一个成功应用的商用专家系统 RI，1982 年开始在美国数字装备集团公司（DEC)运行，用于进行新计算机系统的结构设计。到 1986 年，R1 每年为该公司节省 400 万美元。到 1988 年，DEC 公司的人工智能团队开发了 40 个专家系统。更有甚者，杜珀公司已使用 100 个专家系统，正在开发 500 个专家系统。几乎每个美国大公司都拥有自己的人工智能小组，并应用专家系统或投资专家系统技术。20 世纪 80 年代，日本和西欧也争先恐后地投人对专家系统的智能计算机系统的开发，并应用于工业部门。其中，日本 1981 年发布的“第五代智能计算机计划”就是一例。在开发专家系统过程中，许多研究者获得共识，即人工智能系统是一个知识处理系统，而知识表示、知识利用和知识获取则成为人工智能系统的三个基本问题。

5、 集成发展时期（1986 年至今）

到 20 世纪 80 年代后期，各个争相进行的智能计算机研究计划先后遇到严峻挑战和困难，无法实现其预期目标。这促使人工能研究者们对已有的人工智能和专家系统思想和方法进行反思。已有的专家系统存在缺乏常识知识、应用领域狭窄、知识获取困难、推理机制单一、未能分布处理等问题。他们发现，困难反映出人工智能和知识工程的一些根本问题，如交互问题、扩展问题和体系问题等，都没有很好解决。对存在问题的探讨和对基本观点的争论，有助于人上智能摆脱困境，迎来新的发展机遇。

人工智能应用技术应当以知识处理为核心，实现软件的智能化。知识处理需要对应用领域和问题求解任务有深人的理解，扎根于主流计算环境。只有这样，才能促使人工智能研究和应用走上持续发展的道路。

20 世纪 80 年代后期以来，机器学习、计算智能、人工神经网络和行为主义等研究的深人开展，不时形成高潮。有别于符号主义的连接主义和行为主义的人工智能学派也乘势而上，获得新的发展。不同人工智能学派间的争论推动了人工智能研究和应用的进一步发展。以数理逻辑为基础的符号主义，从命题逻辑到谓词逻辑再至多值逻辑，包括模糊逻辑和粗糙集理论，已为人工智能的形成和发展做出历史性贡献，并已超出传统符号运算的范畴，表明符号主义在发展中不断寻找新的理论、方法和实现途径。传统人工智能（我们称之为(l)的数学计算体系仍不够严格和完整。除了模糊计算外，近年来，许多模仿人脑思维、自然特征和生物行为的计算方法（如神经计算、进化计算、自然计算、免疫计算和群计算等）已被引人人工智能学科。我们把这些有别于传统人工智能的智能计算理论和方法称为计算智能（computational intelligence，CI）。计算智能弥补了传统 Al 缺乏数学理论和计算的不足，更新并丰富了人工智能的理论框架，使人工智能进人一个新的发展时期。人工智能不同观点、方法和技术的集成，是人工智能发展所必需，也是人工智能发展的必然。

在这个时期，特别值得一提的是神经网络的复兴和智能真体（intelligent agent)的突起。

麦卡洛克和皮茨 1943 年提出的“似脑机器”，构造了一个表示大脑基本组成的神经元模型。由于当时神经网络的局限性，特别是硬件集成技术的局限性，使人工神经网络研究在 20 世纪 70 年代进人低潮。直到 1982 年霍普菲尔德（Hopfield）提出离散神经网络模型，]984 年又提出连续神经网络模型，促进了人工神经网络研究的复兴。布赖森（Bryson）和何（He）提出的反向传播（BP）算法及鲁梅尔哈特（Rumelhart）和麦克莱伦德(McClelland)1986 年提出的并行分布处理（PDP）理论是人工神经网络研究复兴的真正推动力，人工神经网络再次出现研究热潮。1987 年在美国召开了第一届神经网络国际会议，并发起成立了国际神经网络学会（]NNS）。这表明神经网络已置身于国际信息科技之林，成为人工智能的一个重要子学科。如果人工神经网络硬件能够在大规模集成上取得突破，那么其作用不可估量。现在，对神经网络的研究出现了 21 世纪以来的一次高潮，特别是基于神经网络的机器学习获得很大发展。近 10 年来，深度学习（deep learning)的研究逐步深人，并已在自然语言处理和人机博弈等领域获得比较广泛的应用。在深度学习的基础上，一种称为“超限学习”（extreme learning）的机器学习方法在近几年得到越来越多的应用。这些研究成果活跃了学术氛围，推动了机器学习的发展。

智能真体（以前称为智能主体）是 20 世纪 90 年代随着网络技术特别是计算机网络通信技术的发展而兴起的，并发展为人工智能又一个新的研究热点。人工智能的目标就是要建造能够表现出一定智能行为的真体，因此，真体（agent）应是人工智能的一个核心问题。人们在人工智能研究过程中逐步认识到，人类智能的本质是一种耳有社会性的智能，社会问题特别是复杂问题的解决需要各方面人员共同完成。人工智能，特别是比较复杂的人工智能问题的求解也必须要各个相关个体协商、协作和协调来完成的。人类社会中的基本个体“人”对应于人工智能系统中的基本组元“真体”，而社会系统所对应的人工智能“多真体系统"也就成为人工智能新的研究对象。

产业的改造与升级、智能制造和服务民生的需求，促进机器人学向智能化方向发展，一股机器人化的新热潮正在全球汹涌澎湃，席卷全世界。智能机器人已成为人工智能研究与应用的一个蓬勃发展的新领域。人工智能已获得越来越广泛的应用，深人渗透到其他学科和科学技术领域，为这些学科和领域的发展作出功不可没的贡献，并为人工智能理论和应用研究提供新的思路与借鉴。例如，对生物信息学、生物机器人学和基因组的研究就是如此。

上述这些新出现的人工智能理论、方法和技术，其中包括人工智能三大学派，即符号主义、连接主义和行为主义，已不再是单枪匹马打天下，而往往是携手合作，走综合集成、优势互补、共同发展的康庄大道。人工智能学界那种势不两立的激烈争论局面，可能一去不复返了。我们有理由相信，人工智能工作者一定能够抓住机遇，不负众望，创造更多更大的新成果，开创人工智能发展的新时期。

我国的人工智能研究起步较晚。纳人国家计划的研究（“智能模拟”）始于 1978 年；1984 年召开了智能计算机及其系统的全国学术讨论会；1986 年起把智能计算机系统、智能机器人和智能信息处理（含模式识别）等重大项目列人国家高技术研究计划；1993 年起，又把智能控制和智能自动化等项目列人国家科技攀登计划。进人 21 世纪后，已有更多的人工智能与智能系统研究获得各种基金计划支持，并与国家国民经济和科技发展的重大需求相结合，力求作出更大贡献。1981 年起，相继成立了中国人工智能学会（CAAI)及智能机器人专业委员会和智能控制专业委员会、全国高校人工智能研究会、中国计算机学会人工智能与模式识别专业委员会、中国自动化学会模式识别与机器智能专业委员会、中国软件行业协会人工智能协会以及智能自动化专业委员会等学术团体。在中国人工智能学会归属中国科学技术协会直接领导和管理之后，又有一些省市成立了地方人工智能学会，推动了我国人 T 智能的发展。1989 年首次召开了中国人工智能控制联合会议(CJCAI)O 已有约 60 部国内编著的具有知识产权的人工智能专著和教材出版，其中，本书就已出版发行 40 多万册。1982 年创刊《人工智能学报》杂志，《模式识别与人工智能》杂志和《智能系统学报》分别于 1987 年和 2006 年创刊。《智能技术学报》英文版即将创刊。2006 年 8 月，中国人工智能学会联合兄弟学会和有关部门，在北京举办了包括人工智能国际会议和中国象棋人机大战等在内的“庆祝人工智能学科诞生 50 周年”大型庆祝活动，产生了很好的影响。今年 4 月又在北京举行《全球人工智能技术大会暨人工智能 60 周年纪念活动启动仪式》，隆重而热烈地庆祝国际人工智能学科诞生 60 周年。2009 年，

中国人工智能学会牵头组织，向国家学位委员会和国家教育部提出“设置'智能科学与技术，学位授权一级学科”的建议，为我国人工智能和智能科学学科建设不遗余力，意义深远。2015 年在中国最热门的话题和产业应该是机器人学，中国机器人学的磅礴热潮推动世界机器人产业的新一轮竞争与发展。2016 年中国最为引人注目的科技应是工人智能，并出现了发展人工智能及其产业的新潮。中国的人工智能工作者，已在人工智能领域取得许多具有国际领先水平的创造性成果。其中，尤以吴文俊院士关于几何定理证明的“吴氏方法”最为突出，已在国际上产生了重大影响，并荣获 2001 年国家科学技术最高奖。现在，我国已有数以万计的科技人员和大学师生从事不同层次的人工智能研究与学习，人工智能研究已在我国深人开展，它必将为促进其他学科的发展和我国的现代化建设做出新的重大贡献。

1.2.3 人工智能前景方向

人工只能三大流派中的符号主义认为只要实现指名功能就可以实现人工智能；联结主义认为只要实现指心功能就可以实现人工智能；行为主义认为只要实现指物功能就可以实现人工智能。人工智能的三大流派虽然取得了很大进展，但各自也面临巨大挑战。简单地说，人工智能三大流派假设之所以能够成立的前提是指名、指物、指心功能等价。

可是这个前提成立吗？早期的人工智能研究使用经典概念，而经典概念至少具有以下 5 个假设：①概念的外延表示可以用经典集合表示；②概念的内涵表示存在命题表示；③指称对象的外延表示与其内涵表示名称一致；④概念表示唯一，即同一个概念的表示与个体无关，对于同一个概念，每个人表示都是一样的；⑤概念的内涵表示与外延表示在指称对象上功能等价。

可以明显看出，在上述 5 个假设之下，经典概念的指心、指物、指名功能是等价的，即指名意味着指物、指心。但是，日常生活中使用的概念一般并不满足经典概念的 5 条假设，因此也不能保证其指心、指物、指名功能等价。《周易 • 系辞上》中也说，“书不尽言，言不尽意"，明确指出了指名、指心与指物不一定等价。下面给出两个例子，以说明日常生活中概念的指名、指物功能并不等价。微信上曾经流传过一个著名段子：一个人说手头有一个亿，谁有项目通知一下，一起投资。不然，再晚一点，就洗手不干了。听的人以为其是指物即真的有一个亿的资金。而实际上，这个人只是在手头上写了三个字“一个亿"而已。在这儿纯粹指名，即“手头有一个亿”仅仅是符号“一个亿”而已。这个段子显然利用了概念的指名与指物不一定等价的性质。

综上所述，概念的指名、指物与指心功能在生活中并不等价，单独实现概念的一个功能并不能保证具有智能。因此，单独遵循一个学派不足以实现人工智能，现在的人工智能研究已经不再强调遵循人工智能的单一学派。很多时候会综合各个流派的技术。比如，从专家系统发展起来的知识图谱已经不完全遵循符号主义的路线了。在围棋上战胜人类顶尖棋手的 AlphaGo。综合使用了三种学习算法——强化学习、蒙特卡罗树搜索、深度学习，而这三种学习算法分属于三个人工智能流派（强化学习属于行为主义，蒙特卡罗树搜索属于符号主义，深度学习属于连接主义）。无人驾驶技术同样是突破了人工智能三大流派限制的综合技术。虽然人工智能发展至今，各个流派依然在发展，也都取得了很好的进展，但是各个流派进行融合已经是大势所趋，特别是在大数据和云计算的助力下，新一代人工智能将带来社会的第四次技术革命。

然而，目前的人工智能还有很大的缺陷，其使用的知识表示还是建立在经典概念的基础之上。图灵测试的文章发表之日在 1950 年，当时人们对于经典概念的普适性还没有提出质疑，因此，其使用的概念是基于经典概念的。在图灵测试中，最重要的概念之一是人，包括提问者和回答者。但是什么样的人才是合适的提问者和回答者，是中国人还是英国人、是圣人还是智力障碍者或者装傻者，图灵测试并没有定义。如果人存在经典定义，在图灵测试里，就容易确定什么样的人作为提问者和回答者。可惜的是，人并不能用经典概念定义来定义。

经典概念的基本假设还是指心、指名与指物等价，这与人类的日常生活经验严重不符，过于简单化了。在人类的现实生活当中，概念的指名指物指心并不总是等价的。在基于经典概念的知识表示框架下，现在的机器表现有时显得极其智障，缺乏常识、缺乏理解能力，严重缺乏处理突发状况的能力。实际上，维特根斯坦在 1953 年出版的《哲学研究》明确提出，日常生活中使用的概念如人等是没有经典概念定义的。这实际上给图灵测试带来了很大的不确定性。因此，在经典概念表示不成立的情形下，如何进行概念表示是

一个极具挑战性的问题。

1.3 人工智能的流派

目前人工智能的主要流派有下列 3 家：

（1）符号主义（symbolicism）：又称为逻辑主义（logicism）、心理学派（psychlogism）或计算机学派（computerism），其原理主要为物理符号系统假设和有限合理性原理。

（2）连接主义（connectionism）：又称为仿生学派（bionicsism）或生理学派（physiologism），其原理主要为神经网络及神经网络间的连接机制与学习算法。

（3）行为主义（actionism）：又称为进化主义（evolutionism）或控制论学派（cyberneticsism），其原理为控制论及感知—动作型控制系统。

人工智能各学派对人工智能发展历史具有不同的看法。不同人工智能学派对人工智能的研究方法问题也有不同的看法。这些问题涉及：人工智能是否一定采用模拟人的智能的方法？若要模拟又该如何模拟？对结构模拟和行为模拟、感知思维和行为、对认知与学习以及逻辑思维和形象思维等问题是否应分离研究？是否有必要建立人工智能的统一理论系统？若有，又应以什么方法为基础？

1.3.1 符号主义

符号主义认为人工智能源于数理逻辑。数理逻辑从 19 世纪末起就获迅速发展；到 20 世纪 30 年代开始用于描述智能行为。计算机出现后，又在计算机上实现了逻辑演绎系统。其有代表性的成果为启发式程序 LT（逻辑理论家），证明了 38 条数学定理，表明了可以应用计算机研究人类智能活动。正是这些符号主义者，早在 1956 年首先采用“人工智能”这个术语。后来又发展了启发式算法—>专家系统—>知识工程理论与技术，并在 20 世纪 80 年代取得很大发展。符号主义曾长期一枝独秀，为人工智能的发展作出重要贡献，尤其是专家系统的成功开发与应用，为人工智能走向工程应用和实现理论联系实际具有特别重要的意义。在人工智能的其他学派出现之后，符号主义仍然是人工智能的主流学派。这个学派的代表人物有纽厄尔、肖、西蒙和尼尔逊（Nilsson）等。

符号主义认为人工智能的研究方法应为功能模拟方法。通过分析人类认知系统所具备的功能和机能，然后用计算机模拟这些功能，实现人工智能。符号主义力图用数学逻辑方法来建立人工智能的统一理论体系，但遇到不少暂时无法解决的困难，并受到其他学派的否定。

1.3.2 连接主义

连接主义认为人工智能源于仿生学，特别是人脑模型的研究。它的代表性成果是 1943 年山生理学家麦卡洛克和数理逻辑学家皮茨创立的脑模型，即 MP 模型，开创了用电子装置模仿人脑结构和功能的新途径。它从神经元开始进而研究神经网络模型和脑模型，开辟了人工智能的又一发展道路。20 世纪 60-70 年代，连接主义，尤其是对以感知机（perceptron）为代表的脑模型的研究曾出现过热潮。由于当时的理论模型、生物原型和技术条件的限制，脑模型研究在 70 年代后期至 80 年代初期落入低潮。直到前述 Hopfield 教授在 1982 年和 1984 年发表两篇重要论文，提出用硬件模拟神经网络后，连接主义又重新抬头。1986 年鲁梅尔哈特等人提出多层网络中的反向传播（BP）算法。此后，连接主义势头大振，从模型到算法，从理论分析到工程实现，为神经网络计算机走向市场打下基础。现在，对 ANN 的研究热情仍然较高，但研究成果未能如预想的那样好。

连接主义主张人工智能应着重于结构模拟，即模拟人的生理神经网络结构，并认为功能、结构和智能行为是密切相关的。不同的结构表现出不同的功能和行为。已经提出多种人工神经网络结构和众多的学习算法。

1.3.3 行为主义

行为主义认为人工智能源于控制论。控制论思想早在 20 世纪 40-50 年代就成为时代思潮的重要部分，影响了早期的人工智能工作者。维纳和麦克洛（McCloe）等人提出的控制论和自组织系统以及钱学森等人提出的工程控制论和生物控制论，影响了许多领域。控制论把神经系统的工作原理与信息理论、控制理论、逻辑以及计算机联系起来。早期的研究工作重点是模拟人在控制过程中的智能行为和作用，如对自寻优、自适应、自校正、自镇定、自组织和自学习等控制论系统的研究，并进行“控制论动物”的研制。到 60-70

年代，上述这些控制论系统的研究取得一定进展，播下智能控制和智能机器人的种子，并在80年代诞生了智能控制和智能机器人系统。行为主义是 20 世纪末才以人工智能新学派的面孔出现的，引起许多人的兴趣。这一学派的代表作首推布鲁克斯（Brooks）的六足行走机器人，它被看作新一代的“控制论动物”，是一个基于感知一动作模式的模拟昆虫行为的控制系统。

行为主义认为人工智能的研究方法应采用行为模拟法，也认为功能、结构和智能行为是不可分的。不同行为表现出不同功能和不同控制结构。行为主义的研究方法也受到其他学派的怀疑与批判，认为行为主义最多只能创造出智能昆虫行为，而无法创造出人的智能行为。

以上三个人工智能学派将长期共存与合作，取长补短，并走向融合和集成，为人工智能的发展做出贡献。

1.4 人工智能的研究内容

人工智能学科有着十分广泛和极其丰富的研究内容。不同的人工智能研究者从不同的角度对人工智能的研究内容进行分类。例如，基于脑功能模拟、基于不同认知观、基于应用领域和应用系统、基于系统结构和支撑环境等。因此，要对人工智能研究内容进行全面和系统的介绍也是比较困难的，而且可能也是没有必要的。下面综合介绍一些得到诸多学者认同并具有普遍意义的人工智能研究的基本内容。

浩斯顿（Houston）等把认知归纳为如下 5 种类型：

（1） 信息处理过程；

（2） 心理上的符号运算；

（3） 问题求解；

（4） 思维；

（5） 诸如知觉、记忆、思考、判断、推理、学习、想象、问题求解、概念形成和语言使用等关联活动。

人类的认知过程是非常复杂的。作为研究人类感知和思维信息处理过程的一门学科，认知科学（或称思维科学）就是要说明人类在认知过程中是如何进行信息加工的。认知科学是人工智能的重要理论基础，涉及非常广泛的研究课题。除了浩斯顿提出的知觉、记忆、思考、学习、语言、想象、创造、注意和问题求解等关联活动外，还会受到环境、社会和文化背景等方面的影响。人工智能不仅要研究逻辑思维，而且还要深入研究形象思维和灵感思维，使人工智能具有更坚实的理论基础，为智能系统的开发提供新思想和新途径。

1、知识表示

知识表示、知识推理和知识应用是传统人工智能的三大核心研究内容。其中，知识表示是基础，知识推理实现问题求解，而知识应用是目的。知识表示是把人类知识概念化、形式化或模型化。一般地，就是运用符号知识、算法和状态图等来描述待解决的问题。已提出的知识表示方法主要包括符号表示法和神经网络表示法两种。

2、专家系统

人工智能能否获得广泛应用是衡量其生命力和检验其生存力的重要标志。20 世纪 70 年代，正是专家系统的广泛应用，使人工智能走出低谷，获得快速发展。后来的机器学习和近年来的自然语言理解应用研究取得重大进展，又促进了人工智能的进一步发展。当然，应用领域的发展是离不开知识表示和知识推理等基础理论以及基本技术的进步的。

3、机器感知

机器感知就是使机器具有类似于人的感觉，包括视觉、听觉、力觉、触觉、嗅觉、痛觉、接近感和速度感等。其中，最重要的和应用最广的要算机器视觉（计算机视觉）和机器听觉。机器视觉要能够识别与理解文字、图像、场景以至人的身份等；机器听觉要能够识别与理解声音和语言等。机器感知是机器获取外部信息的基本途径。要使机器具有感知能力，就要为它安上各种传感器。机器视觉和机器听觉已催生了人工智能的两个研究领域一一模式识别和自然语言理解或自然语言处理。实际上，随着这两个研究领域的进展，它们已逐步发展成为相对独立的学科。

4、机器思维

机器思维是对传感信息和机器内部的工作信息进行有目的的处理。要使机器实现思维，需要综合应用知识表示、知识推理、认知建模和机器感知等方面的研究成果，开展如下各方面的研究工作：

（1） 知识表示，特别是各种不确定性知识和不完全知识的表示。

（2） 知识组织、积累和管理技术。

（3） 知识推理，特别是各种不确定性推理、归纳推理、非经典推理等。

（4） 各种启发式搜索和控制策略。

（5） 人脑结构和神经网络的工作机制。

5、机器学习

机器学习是继专家系统之后人工智能应用的又一重要研究领域，也是人工智能和神经计算的核心研究课题之一。现有的计算机系统和人工智能系统大多数没有什么学习能力，至多也只有非常有限的学习能力，因而不能满足科技和生产提出的新要求。学习是人类具有的一种重要智能行为。机器学习就是使机器（计算机）具有学习新知识和新技术，并在实践中不断改进和完善的能力。机器学习能够使机器自动获取知识，向书本等文献资料和与人交谈或观察环境进行学习。

6、机器行为

机器行为系指智能系统（计算机，机器人）具有的表达能力和行动能力，如对话、描写、刻画以及移动、行走、操作和抓取物体等。研究机器的拟人行为是人工智能的高难度的任务。机器行为与机器思维密切相关，机器思维是机器行为的基础。

7、计算机智能系统

上述直接的实现智能研究，离不开智能计算机系统或智能系统，离不开对新理论、新技术和新方法以及系统的硬件和软件支持。需要开展对模型、系统构造与分析技术、系统开发环境和构造工具以及人工智能程序设计语言的研究。一些能够简化演绎、机器人操作和认知模型的专用程序设计以及计算机的分布式系统、并行处理系统、多机协作系统和各种计算机网络等的发展，将直接有益于人工智能的开发。

1.5 人工智能的典型应用领域

在大多数学科中存在着几个不同的研究领域，每个领域都有其特有的感兴趣的研究课题、研究技术和术语。在人工智能中，这样的领域包括自然语言处理、自动定理证明、自动程序设计、智能检索、智能调度、机器学习、机器人学、专家系统、智能控制、模式识别、视觉系统、神经网络、agent、计算智能、问题求解、人工生命、人工智能方法和程序设计语言等。在过去 60 年中，已经建立了一些具有人工智能的计算机系统；例如，能够求解微分方程的，下棋的，设计分析集成电路的，合成人类自然语言的，检索情报的，诊断疾病以及控制太空飞行器、地面移动机器人和水下机器人的具有不同程度人工智能的计算机系统。

值得指出的是，正如不同的人工智能子领域不是完全独立的一样，这里简介的各种智能特性也不是互不相关的。把它们分开来介绍只是为了便于指出现有的人工智能程序能够做些什么和还不能做什么。大多数人工智能研究课题都涉及许多智能领域。

1、搜索和问题消解

人工智能的第一个大成就是发展了能够求解难题的下棋（如国际象棋）程序。在下棋程序中应用的某些技术，如向前看几步，并把困难的问题分成一些比较容易的子问题，发展成为搜索和问题消解（归约）这样的人工智能基本技术。今天的计算机程序能够下锦标赛水平的各种方盘棋、十五子棋、中国象棋和国际象棋，并取得前面提到的计算机棋手战胜国际和国家象棋冠军的成果。另一种问题求解程序把各种数学公式符号汇编在一起，其性能达到很高的水平，并正在为许多科学家和工程师所应用。有些程序甚至还能够用经验来改善其性能。

如前所述，这个问题中未解决的问题包括人类棋手具有的但尚不能明确表达的能力，如国际象棋大师们洞察棋局的能力。另一个未解决的问题涉及问题的原概念，在人工智能中叫做问题表示的选择。人们常常能够找到某种思考问题的方法从而使求解变易而解决该问题。到目前为止，人工智能程序已经知道如何考虑它们要解决的问题，即搜索解答空间，寻找较优的解答。

2、逻辑推理

早期的逻辑演绎研究工作与问题和难题的求解相当密切。已经开发出的程序能够借助于对事实数据库的操作来“证明"断定；其中每个事实由分立的数据结构表示，就像数理逻辑中由分立公式表示一样。与人工智能的其他技术的不同之处是，这些方法能够完整和一致地加以表示。也就是说，只要本原事实是正确的，那么程序就能够证明这些从事实得出的定理，而且也仅仅是证明这些定理。

逻辑推理是人工智能研究中最持久的子领域之一。特别重要的是要找到一些方法，只把注意力集中在一个大数据库中的有关事实上，留意可信的证明，并在出现新信息时适时修正这些证明。对数学中臆测的定理寻找一个证明或反证，确实称得上是一项智能任务。为此不仅需要有根据假设进行演绎的能力，而且需要某些直觉技巧。1976 年 7 月，美国的阿佩尔(K.Appel)等人合作解决了长达 124 年之久的难题一一四色定理。他们用三台大型计算机，花去 1200 小时 CPU 时间，并对中间结果进行人为反复修改 500 多处。四色定理的成功证明曾轰动计算机界。我国人工智能大师吴文俊院士提出并实现了几何定理机器证明的方法，被国际上承认为“吴氏方法”，是定理证明的又一标志性成果。

3、计算智能

计算智能(computational intelligence）涉及神经计算、模糊计算、进化计算、粒群计算、自然计算、免疫计算和人工生命等研究领域。进化计算(evolutionary computation）是指一类以达尔文进化论为依据来设计、控制和优化人工系统的技术和方法的总称，它包括遗传算法（genetic algorithm)、进化策略(evolutionary strategy)和进化规划(evolutionary programming)。自然选择的原则是适者生存，即物竞天择，优胜劣汰。

自然进化的这些特征早在 20 世纪 60 年代就引起了美国的霍兰（Holland）的极大兴趣。受达尔文进化论思想的影响，他逐渐认识到在机器学习中，为获得一个好的学习算法，仅靠单个策略的建立和改进是不够的，还要依赖于一个包含许多候选策略的群体的繁殖。他还认识到，生物的自然遗传现象与人工自适应系统行为的相似性，因此他提出在研究和设计人工自主系统时可以模仿生物自然遗传的基本方法。70 年代初，霍兰提出了“模式理论”，并于 1975 年出版了《自然系统与人工系统的自适应》专著，系统地阐述了遗传算法的基本原理，奠定了遗传算法研究的理论基础。

遗传算法、进化规划、进化策略具有共同的理论基础，即生物进化论，因此，把这三种方法统称为进化计算，而把相应的算法称为进化算法。

人工生命是 1987 年提出的，旨在用计算机和精密机械等人工媒介生成或构造出能够表现自然生命系统行为特征的仿真系统或模型系统。自然生命系统行为具有自组织、自复制、自修复等特征以及形成这些特征的混沌动力学、进化和环境适应。

人工生命的理论和方法有别于传统人工智能和神经网络的理论和方法。人工生命把生命现象所体现的自适应机理通过计算机进行仿真，对相关非线性对象进行更真实的动态描述和动态特征研究。

人工生命学科的研究内容包括生命现象的仿生系统、人工建模与仿真、进化动力学、人工生命的计算理论、进化与学习综合系统以及人工生命的应用等。

4、分布式人工智能

分布式人工智能(distributed AI，DAI)是分布式计算与人工智能结合的结果。DAI 系统以鲁棒性作为控制系统质量的标准，并具有互操作性，即不同的异构系统在快速变化的环境中具有交换信息和协同工作的能力。

分布式人工智能的研究目标是要创建一种能够描述自然系统和社会系统的精确概念模型。DAI 中的智能并非独立存在的概念，只能在团体协作中实现，因而其主要研究问题是各 agent 间的合作与对话，包括分布式问题求解和多 agent 系统（multi-agent system，MAS)两领域。MAS 更能体现人类的社会智能，具有更大的灵活性和适应性，更适合开放和动态的世界环境，因血备受重视，已成为人工智能乃至计算机科学和控制科学与工程的研究热点。

5、自动程序设计

自动程序设计能够以各种不同的目的描述来编写计算机程序。对自动程序设计的研究不仅可以促进半自动软件开发系统的发展，而且也使通过修正自身数码进行学习的人工智能系统得到发展。程序理论方面

的有关研究工作对人工智能的所有研究工作都是很重要的。

自动编制一份程序来获得某种指定结果的任务与证明一份给定程序将获得某种指定结果的任务是紧密相关的，后者叫做程序验证。

自动程序设计研究的重大贡献之一是作为问题求解策略的调整概念。已经发现，对程序设计或机器人控制问题，先产生一个不费事的有错误的解，然后再修改它，这种做法要比坚持要求第一个解答就完全没有缺陷的做法有效得多。

6、专家系统

一般地，专家系统是一个智能计算机程序系统，其内部具有大量专家水平的某个领域知识与经验，能够利用人类专家的知识和解决问题的方法来解决该领域的问题。

发展专家系统的关键是表达和运用专家知识，即来自人类专家的并已被证明对解决有关领域内的典型问题是有用的事实和过程。专家系统和传统的计算机程序的本质区别在于专家系统所要解决的问题一般没有算法解，并且经常要在不完全、不精确或不确定的信息基础上得出结论。

随着人工智能整体水平的提高，专家系统也获得发展。正在开发的新一代专家系统有分布式专家系统和协同式专家系统等。在新一代专家系统中，不但采用基于规则的方法，而且采用基于框架的技术和基于模型的原理。

7、机器学习

学习是人类智能的主要标志和获得知识的基本手段。机器学习（自动获取新的事实及新的推理算法）是使计算机具有智能的根本途径。此外，机器学习还有助于发现人类学习的机理并揭示人脑的奥秘。

传统的机器学习倾向于使用符号表示而不是数值表示，使用启发式方法而不是算法。传统机器学习的另一倾向是使用归纳(induction）而不是演绎（deduction）。前一倾向使它有别于人工智能的模式识别等分支；后一倾向使它有别于定理证明等分支。

按系统对导师的依赖程度可将学习方法分类为：机械式学习、讲授式学习、类比学习、归纳学习、观察发现式学习等。

近 20 年来又发展了下列各种学习方法：基于解释的学习、基于事例的学习、基于概念的学习、基于神经网络的学习、遗传学习、增强学习、深度学习、超限学习以及数据挖掘和知识发现等。

数据挖掘和知识发现是 20 世纪 90 年代初期新崛起的一个活跃的研究领域。在数据库基础上实现的知识发现系统，通过综合运用统计学、粗糙集、模糊数学、机器学习和专家系统等多种学习手段和方法，从大量的数据中提炼出抽象的知识，从而揭示出蕴涵在这些数据背后的客观世界的内在联系和本质规律，实现知识的自动获取。

深度学习算法是一类基于生物学对人脑进一步认识，将神经—>中枢—>大脑的工作原理设计成一个不断迭代、不断抽象的过程，以便得到最优数据特征表示的机器学习算法；该算法从原始信号开始，先做低级抽象，然后逐渐向高级抽象迭代，由此组成深度学习算法的基本框架。深度学习源于 2006 年加拿大多伦多大学 Geoffrey Hinton 提出了两个观点：

(1)多隐含层的人工神经网络具有优异的特征学习能力，学习特征对数据有更本质的刻画，从而有利于可视化或分类；

(2)深度神经网络在训练上的难度，可以通过逐层初始化来克服。这些思想开启了深度学习的研究与应用热潮。

超限学习作为一种新的机器学习方法，在许多研究者的不断研究下，已经成为了一个热门研究方向。超限学习主要有以下四个特点。

(1)对于大多数神经网络和学习算法，隐层节点/神经元不需要迭代式的调整；

(2)超限学习既属于通用单隐层前馈网络，又属于多隐层前馈网络；

(3)超限学习的相同构架可用作特征学习、聚类、回归和分类问题；

(4)每个超限学习层组成一个隐层，不需要调整层神经元的学习，整个网路构成一个大的单层超限学习机，且每层都可由一个超限学习机学习。

大规模数据库和互联网的迅速发展，使人们对数据库的应用提出新的要求。数据库中包含的大量知识无法得到充分的发掘与利用，会造成信息的浪费，并产生大量的数据垃圾。另一方面，知识获取仍是专家系统研究的瓶颈问题。从领域专家获取知识是非常复杂的个人到个人之间的交互过程，具有很强的个性和随机性，没有统一的办法。因此，人们开始考虑以数据库作为新的知识源。数据挖掘和知识发现能自动处理数据库中大量的原始数据，抽取出具有必然性的、富有意义的模式、成为有助于人们实现其目标的知识，找出人们对所需问题的解答。这些导致了大数据技术的出现与快速发展。

8、自然语言处理

语言处理也是人工智能的早期研究领域之一，并引起进一步的重视。已经编写出能够从内部数据库回答问题的程序，这些程序通过阅读文本材料和建立内部数据库，能够把句子从一种语言翻译为另一种语言，执行给出的指令和获取知识等。有些程序甚至能够在一定程度上翻译从话筒输人的口头指令。

当人们用语言互通信息时，他们几乎不费力地进行极其复杂却又只需要一点点理解的过程。语言已经发展成为智能动物之间的一种通信媒介，它在某些环境条件下把一点“思维结构”从一个头脑传输到另一个头脑，而每个头脑都拥有庞大的、高度相似的周围思维结构作为公共的文本。这些相似的、前后有关的思维结构中的一部分允许每个参与者知道对方也拥有这种共同结构，并能够在通信“动作”中用它来执行某些处理。语言的生成和理解是一个极为复杂的编码和解码问题。

9、机器人学

人工智能研究中日益受到重视的另一个分支是机器人学。一些并不复杂的动作控制问题，如移动式机器人的机械动作控制问题，表面上看并不需要很多智能。然而人类几乎下意识就能完成的这些任务，要是由机器人来实现就要求机器人具备在求解需要较多智能的问题时所用到的能力。

机器人和机器人学的研究促进了许多人工智能思想的发展。它所导致的一些技术可用来模拟世界的状态，用来描述从一种世界状态转变为另一种世界状态的过程。

智能机器人的研究和应用体现出广泛的学科交叉，涉及众多的课题，如机器人体系结构、机构、控制、智能、视觉、触觉、力觉、听觉、机器人装配、恶劣环境下的机器人以及机器人语言等。机器人已在各种工业、农业、商业、旅游业、空中和海洋以及国防等领域获得越来越普遍的应用。近年来，智能机器人的研发与应用已在全世界出现一个热潮，极大地推动智能制造和智能服务等领域的发展。

10、模式识别

计算机硬件的迅速发展，计算机应用领域的不断开拓，急切要求计算机能更有效地感知诸如声音、文字、图像、温度、震动等人类赖以发展自身、改造环境所运用的信息资料。着眼于拓宽计算机的应用领域，提高其感知外部信息能力的学科一一一模式识别便得到迅速发展。

人工智能所研究的模式识别是指用计算机代替人类或帮助人类感知模式，是对人类感知外界功能的模拟，研究的是计算机模式识别系统，也就是使一个计算机系统具有模拟人类通过感官接受外界信息、识别和理解周围环境的感知能力。

实验表明，人类接受外界信息的80%以上来自视觉，10%左右来自听觉。所以，早期的模式识别研究工作集中在对视觉图像和语音的识别上。

模式识别是一个不断发展的新学科，它的理论基础和研究范围也在不断发展。随着生物医学对人类大脑的初步认识，模拟人脑构造的计算机实验即人工神经网络方法已经成功地用于手写字符的识别、汽车牌照的识别、指纹识别、语音识别、车辆导航、星球探测等方面。

11、机器视觉

机器视觉或计算机视觉已从模式识别的一个研究领域发展为一门独立的学科。在视觉方面，已经给计算机系统装上电视输人装置以便能够“看见”周围的东西。在人工智能中研究的感知过程通常包含一组操作。

整个感知问题的要点是形成一个精练的表示以取代难以处理的、极其庞大的未经加工的输人数据。最终表示的性质和质量取决于感知系统的目标。不同系统有不同的目标，但所有系统都必须把来自输人的、多得惊人的感知数据简化为一种易于处理的和有意义的描述。

计算机视觉通常可分为低层视觉与高层视觉两类。低层视觉主要执行预处理功能，如边缘检测、动目标检测、纹理分析、通过阴影获得形状、立体造型、間面色彩等。高层视觉则主要是理解所观察的形象。

机器视觉的前沿研究领域包括实时并行处理、主动式定性视觉、动态和时变视觉、维景物的建模与识别、实时图像压缩传输和复原、多光谱和彩色图像的处理与解释等。

12、神经网络

研究结果已经证明，用神经网络处理直觉和形象思维信息具有比传统处理方式好得多的效果。神经网络的发展有着非常广阔的科学背景，是众多学科研究的综合成果。神经生理学家、心理学家与计算机科学家的共同研究得出的结论是：人脑是一个功能特别强大、结构异常复杂的信息处理系统，其基础是神经元及其互联关系。因此，对人脑神经元和人工神经网络的研究，可能创造出新一代人工智能机一一神经计算机。

对神经网络的研究始于 20 世纪 40 年代初期，经历了一条十分曲折的道路，几起几落，80 年代初以来，对神经网络的研究再次出现高潮。

对神经网络模型、算法、理论分析和硬件实现的大量研究，为神经计算机走向应用提供了物质基础。人们期望神经计算机将重建人脑的形象，极大地提高信息处理能力，在更多方面取代传统的计算机。

13、智能控制

人工智能的发展促进自动控制向智能控制发展。智能控制是一类无需（或需要尽可能少的）人的干预就能够独立地驱动智能机器实现其目标的自动控制。或者说，智能控制是驱动智能机器自主地实现其目标的过程。许多复杂的系统，难以建立有效的数学模型和用常规控制理论进行定量计算与分析，而必须采用定量数学解析法与基于知识的定性方法的混合控制方式。随着人工智能和计算机技术的发展，已可能把自动控制和人工智能以及系统科学的某些分支结合起来，建立一种适用于复杂系统的控制理论和技术。智能控制正是在这种条件下产生的。它是自动控制的最新发展阶段，也是用计算机模拟人类智能的一个重要研究领域。

智能控制是同时具有以知识表示的非数学广义世界模型和以数学公式模型表示的混合控制过程，也往往是含有复杂性、不完全性、模糊性或不确定性以及不存在已知算法的非数学过程，并以知识进行推理，以启发来引导求解过程。智能控制的核心在高层控制，即组织级控制。其任务在于对实际环境或过程进行组织，即决策和规划，以实现广义问题求解。

14、最佳调度和组合问题

确定最佳调度或组合的问题是人们感兴趣的又一类问题。一个古典的问题就是推销员旅行问题（TSP）。许多问题具有这类相同的特性。

在这些问题中有几个（包括推销员旅行问题）是属于理论计算机科学家称为 NP 完全性一类的问题。他们根据理论上的最佳方法计算出所耗时间（或所走步数）的最坏情况来排列不同问题的难度。该时间或步数是随着问题大小的某种量度增长的。

人工智能学家们曾经研究过若干组合问题的求解方法。有关问题域的知识再次成为比较有效的求解方法的关键。智能组合调度与指挥方法已被应用于汽车运输调度、列车的编组与指挥、空中交通管制以及军事指挥等系统。它已引起有关部门的重视。

15、智能检索系统

随着科学技术的迅速发展，出现了“知识爆炸”的情况。对国内外种类繁多和数量巨大的科技文献之检索远非人力和传统检索系统所能胜任。研究智能检索系统已成为科技持续快速发展的重要保证。

数据库系统是储存某学科大量事实的计算机软件系统，它们可以回答用户提出的有关该学科的各种问题。数据库系统的设计也是计算机科学的一个活跃的分支。为了有效地表示、存储和检索大量事实，已经发展了许多技术。

智能信息检索系统的设计者们将面临以下几个问题。首先，建立一个能够理解以自然语言陈述的询问系统本身就存在不少问题。其次，即使能够通过规定某些机器能够理解的形式化询问语句来回语言理解问题，仍然存在一个如何根据存储的事实演绎出答案的问题。第三，理解询问和演绎答案所需要的知识都可

能超出该学科领域数据库所表示的知识。

16、开发新方法

除了直接瞄准实现智能的研究工作外，开发新的方法也往往是人工智能研究的一个重要方面。人工智能对计算机界的某些最大贡献已经以派生的形式表现出来。计算机系统的一些概念，如分时系统、编目处理系统和交互调试系统等，已经在人工智能研究中得到发展。一些能够简化演绎、机器人操作和认识模型的专用程序设计和系统常常是新思想的丰富源泉。几种知识表达语言（把编码知识和推理方法作为数据结构和过程计算机的语言）已在 20 世纪 70 年代后期开发出来，以探索各种建立推理程序的思想。20 世纪 80 年代以来，计算机系统，如分布式系统、并行处理系统、多机协作系统和各种计算机网络等，都有了长足发展。在人工智能程序设计语言方面，除了继续开发和改进通用和专用的编程语言新版本和新语种外，还研究出了一些面向目标的编程语言和专用开发工具。对关系数据库研究所取得的进展，无疑为人工智能程序设计提供了新的有效工具。

课后习题：

参考文献

[1] Fabian M. Suchanek, Gerhard Weikum: Knowledge harvesting from text and Web sources. ICDE 2013: 1250-1253.

[2] Gerhard Weikum, Martin Theobald: From information to knowledge: harvesting entities and relationships from web sources. PODS 2010: 65-76.

[3] A. Singhal. Introducing the Knowledge Graph: things, not strings. Official Google Blog, May, 2012.

[4] Gallagher, Sean. (June 7, 2012). How Google and Microsoft taught search to understand the Web http://arstechnica.com/information-technology/2012/06/inside-the-architecture-of-googles- knowledge-graphand-microsofts-satori/.

[5] Omkar Deshpande, Digvijay S. Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, AnHai Doan: Building, maintaining, and using knowledge bases: a report from the trenches. SIGMOD Conference 2013: 1209-1220.

[6] AMIT S. Introducing the knowledge graph[R]. America:Official Blog of Google, 2012.

[7] Shenshouer. Neo4j[EB/OL]. [2016-05-09]. http://neo4j.com/.

[8] FlockDB Official. FlockDB[EB/OL]. [2016-05-09]. http://webscripts.softpedia.com/script/Database-Tools/FlockDB-66248.html.

[9] Graphdb Official. Graphdb[EB/OL]. [2016-05-09]. http://www.graphdb.net/.

[10] 刘峤, 李杨, 杨段宏, 等. 知识图谱构建技术综述[J]. 计算机研究与发展, 2016, 53(3): 582-600.LIU Qiao, LI yang, YANG Duan-hong, et al. Knowledgegraph construction techniques[J]. Journal of Computer Research and Development, 2016, 53(3): 582-600.

[11] DONG X, GABRILOVICH E, HEITZ G, et al. Knowledge vault: a web-scale approach to probabilistic knowledgefusion[C]//Proc of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. New York: ACM, 2014.

[12] BOLLACKER K, COOK R, TUFTS P. Freebase: a shared database of structured general human knowledge[C]//Proc of the 22nd AAAI Conf on Artificial Intelligence. Menlo Park, CA: AAAI, 2007: 1962-1963.

[13] 孙镇, 王惠临. 命名实体识别研究进展综述[J]. 现代图书情报技术, 2010(6): 42-47.

[14] 赵军, 刘康, 周光有, 等. 开放式文本信息抽取[J]. 中文信息学报, 2011, 25(6): 98-110.

[15] CHINCHOR N, MARSH E. Muc-7 information extraction task definition[C]//Proc of the 7th Message Understanding Conf. Philadelphia: Linguistic Data Consortium, 1998:359-367.

[16] RAU L F. Extracting company names from text[C]//Proc of the 7th IEEE Conf on Artificial Intelligence Applications.Piscataway, NJ: IEEE, 1991: 29-32.

[17] LIU Xiao-hua, ZHANG Shao-dian, WEI Fu-ru, et al. Recognizing named entities in tweets[C]//Proc of the 49th Annual

Meeting of the Association for Computational Linguistics: Human Language Technologies. Stroudsburg, PA: ACL, 2011: 359-367.

[18] LIN Yi-feng, TSAI T, CHOU Wen-chi, et al. A maximum entropy approach to biomedical named entity recognition[C]//Proc of the 4th ACM SIGKDD Workshop on Data Mining in Bioinformatics. New York: ACM, 2004.

[19] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer: Neural Architectures for Named Entity Recognition. HLT-NAACL 2016: 260-270

[20] WHITELAW C, KEHLENBECK A, PETROVIC N, et al. Web-scale named entity recognition[C]//Proc of the 17th ACM Conf on Information and Knowledge Management. New York: ACM, 2008.

[21] JAIN A, PENNACCHIOTTI M. Open entity extraction from web search query logs[C]//Proc of the 23rd Int Conf on Computational Linguistics. Stroudsburg, PA: ACL, 2010 :510-518.

[22] 蔡自兴，刘丽珏等. 人工智能及其应用（第 5 版）[M]. 北京：清华大学出版社，2016.

[23] 李德毅. 人工智能导论[M]. 北京：中国科学技术出版社，2018.

[24] 朱福喜. 人工智能习题解析与实践[M]. 北京：清华大学出版社，2019.

[25] 李娟. 谓词逻辑在人工智能知识表示中的应用[J]. 数码世界， 2017, 06 :168-169.

[26] 张召霞. 面向无人驾驶车辆行为决策的知识库管理系统研究[D]. 安徽：中国科学技术大学，2020.

[27] 张鹏升. 基于产生式模型的人脸正面化研究[D]. 北京：中国人民公安大学，2020.

[28] O. Christensen and M. Hasannasab, "Frame representations via suborbits of bounded operators," 2019 13th International conference on Sampling Theory and Applications (SampTA), Bordeaux, France, 2019, pp. 1-4.

[29] Galka, A., Moontaha, S. & Siniatchkin, M. Constrained expectation maximisation algorithm for estimating ARMA models in state space representation. EURASIP J. Adv. Signal Process. 2020, 23 (2020).

[30] Anitha Florence Vinola, F., Padma, G. A probabilistic stochastic model for analysis on the epileptic syndrome using speech synthesis and state space representation. Int J Speech Technol 23, 355–360 (2020).

[31] Raisi Tousi, R., Kamyabi-Gol, R.A. & Esmaeelzadeh, F. A continuous frame representation for an abstract shearlet group. J. Pseudo-Differ. Oper. Appl. 10, 571–580 (2019).

[32] C. Shu, D. Dosyn, V. Lytvyn, V. Vysotska, A. Sachenko and S. Jun, "Building of the Predicate Recognition System for the NLP Ontology Learning Module," 2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Metz, France, 2019, pp. 802-808.

第 2 章　逻辑推理与知识图谱

本章我们将学习用一阶逻辑及在本体和互联网环境下来表示知识的方法。

2　逻辑推理与知识图谱

人类的智能活动主要是获得并运用知识。知识是智能的基础。为了使计算机具有智能，能模拟人类的智能行为，就必须使它具有知识。但人类的知识需要用适当的模式表示出来，才能存储到计算机中并能够被运用。因此，知识的表示成为人工智能中一个十分重要的研究课题。

知识图谱以结构化的形式描述客观世界中概念、实体间的复杂关系，将互联网的信息表达成更接近人类认知世界的形式，为人类提供了一种更好地组织理解互联网海量信息的能力。知识图谱采用本体知识表示方法，是语义 Web 技术在互联网上的成功应用。

本章将首先介绍知识表示方法，包括知识的概念及特性、然后介绍产生式．框架、状态空间、谓词逻辑等当前人工智能中应用比较广泛的知识表示方法，为后面介绍推理方法、搜索专家系统等奠定基础。接下来介绍知识图谱的概念、生命周期、本体知识表示、语义网络以及典型应用等。

2.1　知识表示方法

2.1.1　知识的概念

知识是人们在长期的生活及社会实践中、在科学研究及实验中积累起来的对客观世界的认识与经验。人们把实践中获得的信息关联在一起，就形成了知识。一般来说，把有关信息关联在一起所形成的信息结构称为知识。信息之间有多种关联形式．其中用得最多的一种是用“如果……，则……”表示的关联形式。在人工智能中，这种知识被称为“规则"，它反映了信息间的某种因果关系。

知识是一个抽象的术语，用于尝试描述人对某种特定对象的理解。柏拉图在《泰阿泰德篇》中将“知识”定义为：真实的（True）、确信的(Belief)、逻辑成立的(Justification)。其后的亚里士多德、笛卡儿、腆德等西方哲学家也对知识论进行了研究探讨。知识论（Epistemology）早已超出哲学的范畴成为西方文明的基石之一。

在智能系统中，知识通常是特定领域的。为了能让智能系统理解、处理知识，并完成基于知识的任务，首先得对知识构建模型，即知识表示。尽管知识表示是人工智能中最基本的，某种程度上来讲最熟悉的概念，但对于“什么是知识表示”这个问题却很少有直接的回答。Davis 试图通过知识在各种任务中扮演的角色来回答这个问题。

知识表示就是将人类知识形式化或者模型化，其目的是能够让计算机存贮和运用人类的知识。根据不同的、不同的知识类型，会有不同的知识表示方法。目前常用的包括产生式表示法、框架表示法、状态空间表示法、谓词逻辑表示法等。对于传统人工智能问题，任何比较复杂的求解技术都离不开两方面的内容一一表示与搜索。对于同一问题可以有多种不同的表示方法，这些表示具有不同的表示空间。问题表示的优劣，对求解结果及求解效率影响甚大。

为解决实际复杂问题，通常需要用到多种不同的表示方法。这是因为，每种数据结构都有其优缺点，而且没有哪一种单独拥有一般需要的多种不同功能。

2.1.2　知识的特性

1、知识的相对正确性

知识是人类对客观世界认识的结品，并且受到长期实践的检验。因此，在一定的条件及环境下，知识是正确的。这里，“一定的条件及环境”是必不可少的，它是知识正确性的前提。因为任何知识都是在一定的条件及环境下产生的，因而也就只有在这种条件及环境下才是正确的。例如，牛顿力学定律在一定的条件下才是正地的。再如，1+1=2，这是一条妇幼皆知的正确知识，但它也只是在十进制的前下才是正确的；如果是二进制，它就不正确了。

知识源于人们生活、学习与工作的实践，知识是人们在信息社会中各种实践经验的汇集、智慧的概括与积累。知识来自于人们对客观世界运动规律的正确认识，是从感性认识上升成为理性认识的高级思维劳动过程的结晶，故相应于一定的客观环境与条件下，知识无疑是正确的。然而当客观环境与条件发生改变时，知识的正确性就要接受检验，必要时就要对原来的认识加以修正或补充，以至全部更新而取而代之。

2、知识的确定与不确定特征

由于现实世界的复杂性，信息可能是精确的，也可能是不精确的、模糊的；关联可能是确定的，也可能是不确定的。这就使知识并不总是只有“真”与“假”这两种状态，而是在“真”与“假”之间还存在许多中间状态，即存在为“真”的程度问题。知识的这一特性称为不确定性。

如前所述，知识由若干信息关联的结构组成。其中有的信息是精确的，有的信息是不精确的。这样，则由该信息结构形成的知识也有了确定或不确定的特征。

造成知识具有不确定性的因素是多方面的。诸如：

（1）证据不足、地域时区不同、各种变化因素及现实世界的复杂性，造成客观后果及其知识的不确定性，（2）生活中，模糊性概念及模糊关系比比皆是，形成了知识的不确定性；

（3）概率事件发生常常不可避免，一般都具有随机不确定性的规律;

（4）经验性及各种不完备的积累过程，导致相关知识的不确定性等。尽管不确定性知识给人们带来了一些迷惑，但它反映了客观世界的多样性、丰富性复杂性。

3、知识的可利用性和可发展性

为了使知识便于传播、学习，使有用的知识得以延续、继承与发展，人们不断地创造了各种生动活泼的形式来记录、描述、表示和利用知识。诸如采用语言、文字，使用书籍，结合文学、戏剧、绘画、摄影等艺术以及电影、电视、多媒体等手段，进行知识的演播、学习与欣赏等。事实上，人类的历史，就是不断地积累知识利用知识创造文明的历史。在人类的发展史中，知识的可利用性与可发展性是不言而喻的。知识的可利用性使得计算机或智能机器能利用知识成为现实；而知识的机器可学习、可表示性使得人工智能不断得以进步与发展成为必然。

伴随着人类社会迈入信息时代，人类知识也进入了大发展时期。一方面在淘汰旧的、老的、无用的知识，另一方面新观念、新思想、新知识不断地被大量地挖掘涌现出来。目前，知识的更新和知识的总量，正以前所未有的速率迅速地增长。大力发展智能科学技术，努力开发人类知识宝库，发展新一代智力工具，这正是作为新时代智能科学工作者的光荣历史使命。

2.1.3 产生式表示法

“产生式”由美国数学家波斯特（E.POST）在 1934 年首先提出，它根据串代替规则提出了一种称为波斯特机的计算模型，模型中的每条规则称为产生式。

1972 年纽厄尔和西蒙在研究人类知识模型中开发了基于规则的产生式系统。

产生式系统是目前已建立的专家系统中知识表示的主要手段之一，如 MYCIN、CLIPS/JESS 系统等。在产生式系统中，把推理和行为的过程用产生式规则表示，所以又称基于规则的系统。

2.1.3.1 规则的表示

一般地，一个规则由前项和后项两部分组成。前项表示前提条件，各个条件由逻辑连接词（合取、析取等）组成各种不同的组合。后项表示当前提条件为真时，应采取的行为或所得的结论。产生式系统中每条规则是一个“条件→结论”或“前提→结论”的产生式，起简单形式为：

$$IF < \text{前提} > \quad THEN < \text{结论} >$$

$$IF < \text{条件} > \quad THEN < \text{动作} >$$

为了严格地描述产生式，下面用巴科斯范式给出它的形式描述和语义：

$$< \text{规则} > ::= < \text{前提} > \rightarrow < \text{结论} >$$

$$< \text{前提} > ::= < \text{简单条件} > | < \text{复合条件} >$$

$$< \text{结论} > ::= < \text{事实} > | < \text{动作} >$$

$$< \text{复合条件} > ::= < \text{简单条件} > \text{And} < \text{简单条件} > [(\text{And} < \text{简单条件} >) \ldots]$$

$$|<\text{简单条件}>\text{Or}<\text{简单条件}>[(\text{or}<\text{简单条件}>)\ldots]$$

$$<\text{动作}>::=<\text{动作名}>[(<\text{变元}>,\ldots)]$$

2.1.3.2 事实的表示

(1) 确定性和不确定性规则知识的产生式表示

确定性规则知识可用前面介绍的产生式的简单形式表示即可。

不确定性规则知识对基本形式作一定的扩充，用如下形式表示：

$$P \rightarrow \text{Q}\ \ (\text{可信度})$$

或者

$$IF\ \ P\ \ \text{THEN}\ \ \text{Q}\ \ (\text{可信度})$$

其中，P 是产生式的前提或条件，用于指出该产生式是否是可用的条件；Q 是一组结论或动作，用于指出该产生式的前提条件 P 被满足时，应该得出的结论或因该执行的操作。这一表示形式主要在不确定推理中当已知事实与前提中的条件不能精确定匹配时，只要按照“可信度”的要求达到一定的相似度，就认为已知事实与前提条件匹配，再按照一定的算法将这种可能性（或不确定性）传递到结论。

(2) 确定性和不确定性事实性知识的产生式表示

确定性事实性知识一般使用三元组：

$$(\text{对象，属性，值})\text{或}(\text{关系，对象 1，对象 2})$$

来表示，其中对象就是语言变量，这种表示的机器内部实现就是一个表。如事实“老李年龄是 35 岁”便可以表示成：

$$(Lee, Age, 35)$$

其中，Lee 是事实性知识涉及的对象，Age 是该对象的属性，而 35 岁是该对象属性的值。而老李、老张是朋友，可表示成：

$$(Friend, Lee, Zhang)$$

而有些事实性知识带有不确定性和模糊性，若考虑不确定性，这种知识就可以用四元组的形式表示如下：

$$(\text{对象，属性，值，不确定度量值})\text{或}$$

$$(\text{关系，对象 1，对象 2，不确定度量值})$$

如不确定性事实性知识“老李年龄可能是 35 岁”，这里老李是 35 岁的可能性取 90%，便可以表示成：

$$(Lee, Age, 35, 0.9)$$

而老李、老张是朋友的可能性不大，这里老李、老张是朋友的可能性取 20%，可表示成：

$$(Friend, Lee, Zhang, 0.2)$$

2.1.3.3 产生式系统的组成部分

产生式系统一般由三个基本部分组成：规则库、综合数据库和推理机。它们之间的关系如下图 2.1.1 所示。

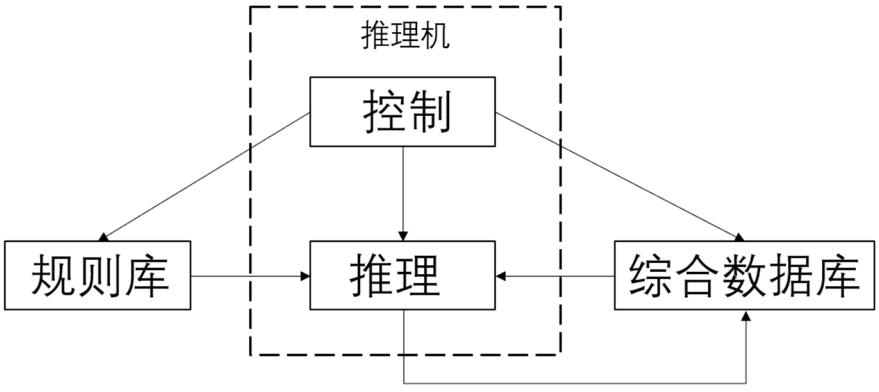


图 2.1.1 产生式系统的基本结构

1、规则库

用于描述某领域内知识的产生式集合，是某领域知识（规则）的存储器，其中的规则是以产生式形式表示的。规则库中包含着将问题从初始状态转换成目标状态（或解状态）的那些变换规则。

规则库是专家系统的核心，也是一般产生式系统赖以进行问题求解的基础，其中知识的完整性和一致性、知识表达的准确性和灵活性以及知识组织的合理性，都将对产生式系统的性能和运行效率产生直接的影响。

2、综合数据库

综合数据库又称为事实库，用于存放输入的事实、从外部数据库输入的事实以及中间结果（事实）和最后结果的工作区。当规则库中的某条产生式的前提可与综合数据库中的某些已知事实匹配时，该产生式就被激活，并把用它推出的结论放入综合数据库中，作为后面推理的已知事实。

显然，综合数据库的内容是在不断变化的，是动态的。

3、推理机

推理机是一个或一组程序，用来控制和协调规则库与综合数据库的运行，包含了推理方式和控制策略。控制策略的作用就是确定选用什么规则或如何应用规则。

通常从选择规则到执行操作分 3 步完成：匹配、冲突解决和操作。

（1）匹配：匹配就是将当前综合数据库中的事实与规则中的条件进行比较，如果相匹配，则这一规则称为匹配规则。

因为可能同时有几条规则的前提条件与事实相匹配，究竟选哪一条规则去执行呢？这就是规则冲突解决。通过冲突解决策略选中的在操作部分执行的规则称为启用规则。

（2）冲突解决：冲突解决的策略有很多种，其中专一性排序、规则排序、规模排序和就近排序是比较常见的冲突解决策略。

（3）操作：操作就是执行规则的操作部分。经过操作以后，当前的综合数据库将被修改，其他的规则有可能将成为启用规则。

（4）检查推理终止条件：检查综合数据库中是否包含了最终结论，决定是否停止系统运行。

2.1.3.4 产生式系统的分类

（1）按产生式所表示的知识是否具有确定性分为：确定性产生式系统和不确定性产生式系统。

（2）按推理机的推理方向分为：正向、反向和双向推理产生式系统。

（3）按规则库及综合数据库的性质与结构特征分为：可交换的产生式系统、可分解的产生式系统和可恢复的产生式系统。

（4）可交换的产生式系统：如果一个产生式系统对规则的使用次序是可交换的，那么无论先使用哪一条规则，都可以达到目的，即规则的使用次序对问题的最终求解是无关紧要的。我们称这样的产生式系统为可交换的产生式系统。

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

（7）可恢复的产生式系统：在可交换产生式系统中，要求每条规则的执行只能为综合数据库增添新的内容，且不能删除和修改综合数据库已有的内容。这一要求是很强的，在许多规则的设计中难以达到。

因此就需要产生式系统具有回溯功能，问题求解到某一步发现无法继续下去时，就撤销在此之前得到的某些结果，恢复到先前的某个状态，然后选用别的规则继续求解。在问题求解过程中既可以对综合数据库添加新内容、又可删除或修改老内容的产生式系统称为可恢复的产生式系统。

2.1.3.5 产生式表示法的应用

有些学者将产生式表示用于人脸识别及正面化的实际问题中，提出了新的基于产生式模型的人脸正面化方法。通过对目前基于深度学习的产生式模型进行研究和分析，在生成对抗学习 GAN 网络框架下对生成器和判别器网络进行改进，设计出了两个能较好实现人脸正面化任务的产生式模型。

产生式模型在概率论和统计学中是指可以随机生成可观测数据的模型。在机器学习中，产生式模型和判别式模型是常见的两个概念。判别式模型是希望反映不同类别数据之间的差异，而产生式模型是希望能够反映同类数据的相似性。

产生式模型是希望学习样本的本质特征，训练好的模型可以生成符合样本分布的新数据。它可以应用于分类任务和生成任务中。对于监督学习的分类任务，产生式模型是通过数据学习联合概率分布$p(x,y)$，对$p(x,y|\theta)$建模，进而根据$p(y|x)=\frac{p(x,y)}{p(x)}$计算样本属于每一类别的类后验概率$p(y|x)$，找到使类后验概率最大的$y$来完成分类。典型的产生式模型有朴素贝叶斯模型、隐马尔科夫模型和混合高斯模型等。产生式模型可以学到比判别式模型更多的信息，能够反映出同类样本之间的相似度。

有的学者将产生式知识表示法用于无人驾驶车辆行为决策，研究无人驾驶车辆的行为决策所需的规则、案例、方案等多种知识的表示、融合方法，针对无人驾驶车辆行为决策信息没有统一的语义描述、无法快速查询所需知识、相关知识难以语义融合等难题，提出一种适用于复杂场景的无人车辆的行为决策的框架，即产生式规则（案例推理）混合的基于多级知识超图的无人驾驶车辆行为决策知识的获取与表示方法，并设计和实现了基于多级知识超图的知识一致性检验方法。

2.1.4 框架表示法

心理学的研究结果表明，在人类日常的思维和理解活动中，当分析和解释遇到新情况时，要使用过去经验积累的知识。这些知识规模巨大而且以很好的组织形式保留在人们的记忆中。例如，当走进一家从未来过的饭店时，根据以往的经验，可以预见在这家饭店将会看到菜单、桌子、服务员等。当走进教室时，可以预见在教室里可以看到椅子、黑板等。人们试图用以往的经验来分析解释当前所遇到的情况，但无法把过去的经验一一都存在脑子里，而只能以一个通用的数据结构的形式存储以往的经验．这样的数据结构称为框架(frame)。框架提供了一个结构，一种组织。在这个结构或组织中，新的资料可以用经验中得到的概念来分析和解释。因此，框架也是一种结构化表示法。

通常框架采用语义网络中的节点、槽、值表示结构“所以框架也可以定义为是一组语义网络的节点和槽，这组节点和槽可以描述格式固定的事物、行动和事件。语义网络可看作节点和弧线的集合，也可以视为框架的集合。框架表示法是一种结构化的知识表示方法，目前已在多种系统中得到广泛的应用。

2.1.4.1 框架的构成

框架（frame）是一种描述所论对象（一个事物、事件或概念）属性的数据结构。

一个框架通常由描述事物的各个方面的槽（slot）组成，每个槽可以拥有若干个侧面，而每个侧面又可以拥有若干个值。一个槽用于描述所论对象某一方面的属性。一个侧面用于描述相应属性的一个方面。槽和侧面所具有的属性值分别被称为槽值和侧面值。在一个用框架表示知识的系统中一般都含有多个框架，一个框架一般都含有多个不同槽、不同侧面，分别用不同的框架名、槽名及侧面名表示。对于框架、槽或侧面，都可以为其附加上一些说明性的信息，一般是一些约束条件，用于指出什么样的值才能填入到槽和侧面中去。这些内容可以根据具体问题的具体需要来取舍，一个框架的一般结构如下所示：

< 框架名 >

槽名 1：	侧面名 $_{11}$	侧面值 $_{111}$，侧面值 $_{112}$，… ，侧面值 $_{11P1}$
	侧面名 $_{12}$	侧面值 $_{121}$，侧面值 $_{122}$，… ，侧面值 $_{12P2}$
	⋮	
	侧面名 $_{1m}$	侧面值 $_{1m1}$，侧面值 $_{1m2}$，… ，侧面值 $_{1mPm}$
槽名 2：	侧面名 $_{21}$	侧面值 $_{211}$，侧面值 $_{212}$，… ，侧面值 $_{21P1}$
	侧面名 $_{22}$	侧面值 $_{221}$，侧面值 $_{222}$，… ，侧面值 $_{22P2}$
	⋮	
	侧面名 $_{2m}$	侧面值 $_{2m1}$，侧面值 $_{2m2}$，… ，侧面值 $_{2mPm}$
	⋮	
槽名 n：	侧面名 $_{n1}$	侧面值 $_{n11}$，侧面值 $_{n12}$，… ，侧面值 $_{n1P1}$

侧面名 $_{n2}$　　侧面值 $_{n21}$，侧面值 $_{n22}$，…，侧面值 $_{n2P2}$

⋮

侧面名 $_{nm}$　　侧面值 $_{nm1}$，侧面值 $_{nm2}$，…，侧面值 $_{nmPm}$

约束：　约束条件 $_1$

约束条件 $_2$

⋮

约束条件 $_n$

上面表示形式可以看出：一个框架可以有任意有限数目的槽；一个槽可以有任意有限数目的侧面；一个侧面可以有任意数目的侧面值。槽值或侧面值既可以是数值、字符串、布尔值，也可以是满足某个给定条件时要执行的动作或过程，还可以是另一个框架的名字，从而实现一个框架对另一个框架的调用，表示出框架之间的横向关系。约束条件是任选的，当不指出约束条件时，表示没有约束。

较简单的情景是用框架来表示诸如人和房子等事物。例如，一个人可以用其职业、身高和体重等项描述，因而可以用这些项目组成框架的槽。当描述一个具体的人时，再将这些项目的具体值填入到相应的槽中。表 2.1.1 给出的是描述 John 的框架。

表 2.1.1　简单框架示例

JOHN		
ISA	:	PERSSON
Profession	:	PROGRAMMER
Height	:	1.8m
Weight	:	79kg

对于大多数问题，不能这样简单地用一个框架表示出来，必须同时使用许多框架，组

成一个框架系统（frame system)。图 2.1.2 所示的就是一个立方体视图的框架表示。图中，最高层的框架，用 ISA 槽说明它是一个立方体，并由 region 槽指示出它所拥有的 3 个可见面 A、B、E。而 A、B、E 又分别用 3 个框架来具体描述。用 must-be 槽指示出它们必是一个平行四边形。

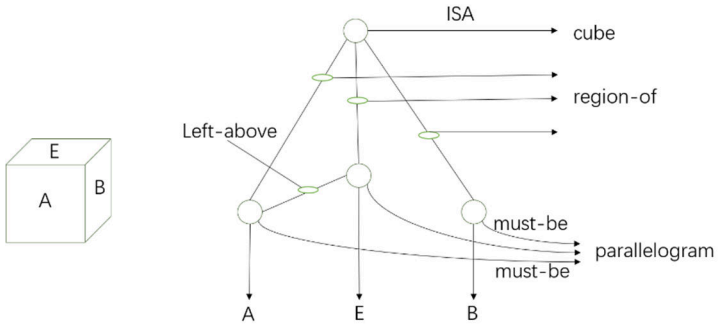


图 2.1.2　一个立方体视图的框架表示

为了能从各个不同的角度来描述物体，可以对不同角度的视图分别建立框架，然后再把它们联系起来组成一个框架系统。图 2.1.3 所示的就是从 3 个不同的角度来研究一个立方体的例子。为了简便起见，图中略去了一些细节，在表示立方体表面的槽中，用实线与可见面连接，用虚线与不可见面连接。

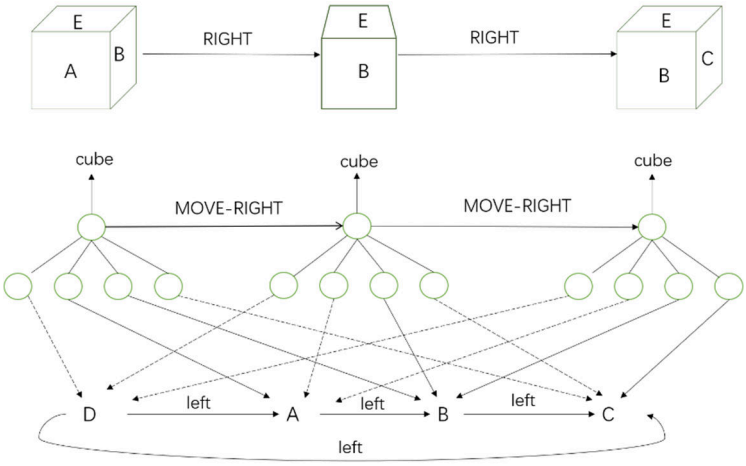


图 2.1.3　表示立方体的框架系统

从图 2.1.3 可见，一个框架结构可以是另一个框架的槽值，并且同一个框架结构可以作为几个不同的框架的槽值。这样，一些相同的信息可以不必重复存储，节省了存储空间。框架的一个重要特性是其继承性。为此，一个框架系统常表示成一种树形结构，树的每一个节点是一个框架结构，子节点与父节点之间用 ISA 或 AKO 槽连接。所谓框架的继承性，就是当子节点的某些槽值或侧面值没有被直接记录时，可以从其父节点继承这些值。例如，椅子一般都有 4 条腿，如果一把具体的椅子没有说明它有几条腿，则可以通过一般椅子的特性，得出它也有 4 条腿。

框架是一种通用的知识表达形式，对于如何运用框架系统还没有一种统一的形式，常常由各种问题的不同需要来决定。框架系统具有树状结构。树状结构框架系统的每个节点具有如下框架结构形式：

框架名

AKO VALUE（值）

PROP DEFAULT（表 1）

SF IF-NEEDED（算术表达式）

CONFLICT ADD（表 2）

其中框架名用类名表示。AKO 是一个槽，VALUE 是它的侧面，通过填写〈值〉的内容表示出该框架属于哪一类。PROP 槽用来记录该节点所具有的特性，其侧面 DEFAULT 表示该槽的内容是可以进行缺省继承的，即当〈表 1〉为非 NIL 时，PROP 的槽值为〈表 1〉，当〈表 1〉为 NIL 时，PROP 的槽值用其父节点的 PROP 槽值来代替。

2.1.4.2　问题状态描述

如前所述，框架是一种复杂结构的语义网络。因此语义网络推理中的匹配和特性继承在框架系统中也可以实行。除此以外，由于框架用于描述具有固定格式的事物、动作和事件，因此可以在新的情况下，推论出未被观察到的事实。框架用以下几种途径来帮助实现这一点：

（1）框架包含它所描述的情况或物体的多方面的信息。这些信息可以被引用，就像已经直接观察到这些信息一样。例如，当一个程序访问一个 ROOM 框架时，不论是否有证据说明屋子里有门，都可以推论出，在屋子里至少有一个门。之所以能这样做，是因为 ROOM 框架中包含对屋子的描述，其中包括在屋子里必须有门的事实。

（2）框架包含物体必须具有的属性。在填充框架的各个槽时，要用到这些属性。建立对某一情况的描述要求先建立对此情况的各个方面的描述。与描述这个情况的框架中的各个槽有关的信息可用来指导如何建立这些方面的描述。

（3）框架描述它们所代表的概念的典型事例。如果某一情况在很多方面和一个框架相匹配，只有少部分相互之间存在不同之处，这些不同之处很可能对应于当前情况的重要方面，也许应该对这些不同之处做出解答。因此，如果椅子被认为应有 4 条腿，而某一椅子只有 3 条腿，那么或许这把椅子需要修理。

当然，在以某种方式应用框架以前，首先要确认这个框架是适用于当前所研究的情况的。这时可以利用一定数量的部分证据来初步选择候选框架。这些候选框架就被具体化，以建立一个描述当前情况的实例。这样的框架将包含若干个必须填入填充值的槽。然后程序通过检测当前的情况，试图找到合适的填充值。如果可以找到满足要求的填充值，就把它们填入到这个具体框架的相应槽中去；如果找不到合适的填充值，就必须选择新的框架。从建立第一个具体的框架试验失败的原因中可为下一个应该试验什么框架提供有用的线索。另一方面，如果找到了合适的值，框架就被认为适合于描述当前的情况。当然，当前的情况可能改变。那么，关于产生什么变化的信息（例如，可以按顺时针方向沿屋子走动）可用来帮助选择描述这个新情况的框架。

用一个框架来具体体现一个特定情况的过程，经常不是很顺利的。当这个过程碰到障碍时，经常不必放弃原来的努力去从头开始，而是有很多办法可想：

（1）选择和当前情况相对应的当前的框架片断，并把这个框架片断和候补框架相匹配。选择最佳匹配。如果当前的框架总的来说差不多是可以接受的，则许多已经做的、有关建立子结构以填入这个框架的工作将可保留。

（2）尽管当前的框架和所要描述的情况之间有不相匹配的地方，但是仍然可以继续应用这个框架。例如，所研究的只有 3 条腿的椅子，可能是一个破椅子或是有另一个在椅子前面的物体挡住了一条腿。框架的某一部分包含关于哪些特性是允许不相匹配的信息。同样的，也有一般的启发性原则，比如一个漏失某项期望特性的框架（可能由于被挡住视线造成的）比另一个多了某一项不应有的特性的框架更适合当前的情况。举例来说，一个人只有一条腿比说一个人有 3 条腿或有尾巴更合乎情理些。

（3）查询框架之间专门保存的链，以提出应朝哪个方向进行试探的建议。这种链的例子与图 2.1.4 所示的网络相似。例如，如果和 CHAIR 框架匹配时，发现没有靠背，并且太宽，这时就建议用 BENCH（条凳）框架；如果太高，并且没有靠背，就建议用 STOOL（凳子）框架。

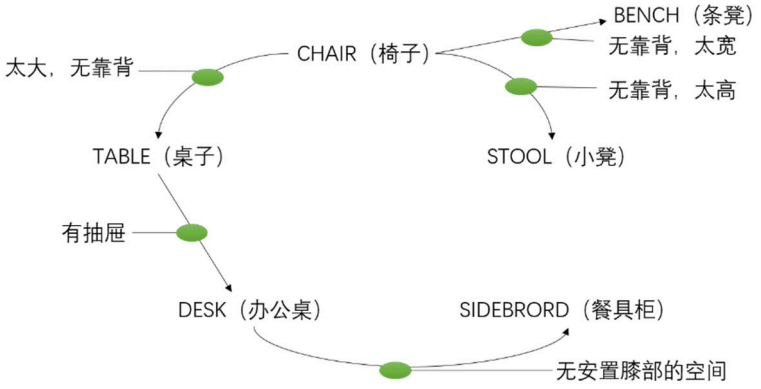


图 2.1.4　相似网络

（4）沿着框架系统排列的层次结构向上移动（即从狗框架—>哺乳动物框架—>动物框架），直到找到一个足够通用并不与已有事实矛盾的框架。如果框架足够具体，可以提供所要求的知识，那就采用这个框架。或者建立一个新的、正好在匹配的框架的下一层的框架。

2.1.4.3　用框架表示知识的例子及应用

本小节将通过举一个具体的例子来说明建立框架的基本方法。

例 2.1.1 教师框架

框架名：<教师>

姓名：单位（姓、名）

年龄：单位（岁）

性别：范围（男、女），缺省：男

职称：范围（教授、副教授、讲师、助教），缺省：讲师

部门：单位（系、教研室）

住址：<住址框架>

工资：<工资框架>

开始工作时间：单位（年、月）

截至时间：单位（年、月），缺省：现在

该框架一共有九个槽，分别描述了“教师”九个方面的情况，或者说关于“教师”的九个属性。在每一个槽里都指出了一些说明性的信息，用于对槽的填值给出某些限制。“范围”指出槽的值只能在指定的范围内挑选，如“职称”槽，其槽值只能是“教授”“副教授”“讲师”“助教”中的某一个，而不能是“工程师”等别的职称；“缺省”表示当相应槽不填入槽值时，就以缺省值作为槽值，这样可以节省一些填槽的工作。例如，对“性别”槽，当不填入“男”或“女”时，默认该值是“男”，这样对于男性教师就可以不填这个槽的槽值。

对于上述框架，当把具体的信息填入槽或侧面后，就得到了相应框架的一个事例框架。例如，把教师的一组信息填入“教师”框架的各个槽，就可得到：

框架名：<教师-1>

姓名：张三

年龄：35

性别：男

职称：教授

部门：计算机系软件教研室

住址：<adr-1>

工资：<sal-1>

开始工作时间：2008.9

截至时间：2017.7

框架表示法应用非常广泛，有研究学者尝试将框架表示法应用于抽象剪切群的表示，作者假设 S 是一个局部紧抽象剪切群，$\sigma: s \to U(H)$是 S 在可分离 Hilbert 空间 H 上的表示，给出了表示系数族是连续剪切集的充分必要条件框架。此外研究了给定表示有关的连续剪切框架与它的不可约字迹表示。

有学者将框架表示与算子相结合提出了有界算子的子轨道的框架表示法。文章考虑了动态抽样的标准设置涉及形式为$\{T^n\varphi\}_n^\infty = 0$的序列的框架性质，其中 T 是 Hilbert 空间 H 和$\varphi \in H$上的有界算子。作者主要考虑这一基本思想的两个推广。首先证明，如果只允许使用$\{T^n\varphi\}_n^\infty = 0$的子集$\left\{T^{\alpha(k)}\varphi\right\}_n^\infty = 0$表示，那么可以使用有界算子迭代来表示的框架类会急剧增加；事实上，任何线性独立框架都会对某个有界算子 T 有这样的表示，然后证明了关于框架性质和$\{\alpha(k)\}_{k=1}^\infty$在 N 中的分布的若干结果。最后，证明了考虑具有任意小公差的近似框架表示也可以消除线性无关的条件，在某种意义上是精确的。

2.1.5 状态空间表示法

问题求解(problem solving)是个大课题，它涉及归约、推断、决策、规划、常识推理、定理证明和相关过程等核心概念。在分析了人工智能研究中运用的问题求解方法之后，就会发现许多问题求解方法是采用试探搜索方法的。也就是说，这些方法是通过在某个可能的解空间内寻找一个解来求解问题的。这种基于解答空间的问题表示和求解方法就是状态空间法，它是以状态和算符（operator）为基础来表示和求解问题的。

2.1.5.1 问题状态描述

首先对状态和状态空间下个定义：

状态（state）是为描述某类不同事物间的差别而引入的一组最少变量q_0，q_1，…，q_n的有序集合，其矢量形式如下：

$$Q=[\ q_0，\ q_1，\ \ldots，\ q_n]^T \tag{2.1.1}$$

式中每个元素q_i（i=0,1,…,n）为集合的分量，称为状态变量，给定每个分量的一组值就得到一个具体的状态，如

$$Q_k = [\ q_{0k}，q_{1k}，\ \ldots，\ q_{nk}]^T \qquad (2.1.2)$$

使问题从一种状态变化为另一种状态的手段称为状态空间。操作符可为走步、过程、规则、数学算子、运算符号或逻辑符号等。

状态空间（state space）是一个表示该问题全部可能状态及其关系的图，它包含三种说明的集合，即所有可能的问题初始状态集合 S，操作符集合 F 以及目标状态集合 G。因此，可把状态空间记为三元状态（S，F，G）。

用十五数码难题（15puzle）来说明状态空间表示的概念。十五数码难题由 15 个编有 1 至 15 并放在 4×4 方格棋上的可走动的棋子组成。棋盘上总有一格是空的，以便让空格周围的棋子走进空格，这也可以理解为移动空格“十五数码难题如图 2.1.5 所示。图中绘出了两种棋局，即初始棋局和目标棋局，它们对应于该问题的初始状态和目标状态。

如何把初始棋局变换为目标棋局呢？问题的解答就是某个合适的棋子走步序列，如“左移棋子 12，下移棋子 15，右移棋子 4，…”,等等。

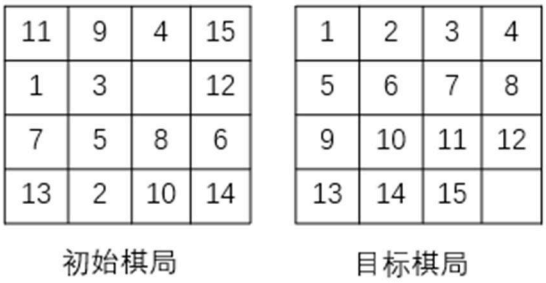


图 2.1.5　十五数码难题

十五数码难题最直接的求解方法是尝试各种不同的走步，直到偶然得到该目标棋局为止。这种尝试本质上涉及某种试探搜索。从初始棋局开始，试探由每一合法走步得到的各种新棋局，然后计算再走一步而得到的下一组棋局。这样继续下去，直至达到目标棋局为止。把初始状态可达到的各状态所组成的空间设想为一幅由各种状态对应的节点组成的图。这种图称为状态空间转换图。图 2.1.6 说明了十五数码难题状态空间图的一部分。图中每个状态图标有它所代表的棋局。首先把适用的算符用于初始状态，以产生新的状态；然后，再把另一些适用算符用于这些新的状态；这样继续下去，直至产生目标状态为止。

一般用状态图的转换来表示下述方法：从某个初始状态开始，每次加一个操作符，递增地建立起操作符的试验序列，直到达到目标状态为止。

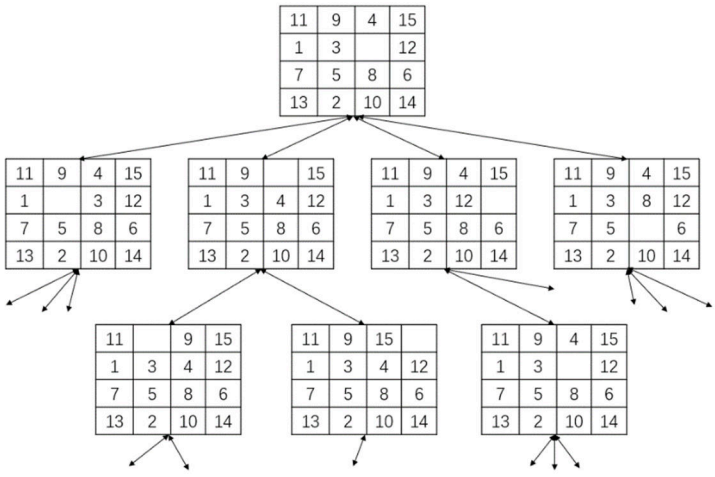


图 2.1.6　十五数码难题部分状态空间图

寻找状态空间的状态包括从旧的状态描述到新的状态描述，以及此后新增的一些新的状态描述，看其是否描述了该目标状态。这种检验往往只是查看某个状态是否与给定的目标状态描述相匹配。不过，有时

还要进行较为复杂的目标测试“对于某些最优化问题，仅仅找到到达目标的任一路径是不够的，还必须找到按某个准则实现最优化的路径（例如，下棋的走步最少）。

综上讨论可知，能用状态图解决问题就必须确定 3 件事：

（1）该状态描述方式，特别是初始状态描述；

（2）操作符集合及其对状态描述的作用；

（3）目标状态描述的特性。

2.1.5.2 状态图表示法

为了对状态空间图有更深人的了解，这里介绍一下状态图中的几个术语和图的正式表示法。

图由节点（不一定是有限的节点）的集合构成。一对节点用弧线连接起来，从一个节点指向另一个节点。这种图叫有向图（directed graph）。如果某条弧线从节点n_i指向节点n_j，那么节点n_j就叫做节点n_i的后继节点或后裔，而节点n_i叫做节点n_j的父辈节点或祖先。一个节点一般只有有限个后继节点。一对节点可能互为后裔，这时，该对有向弧线就用一条棱线代替。当用一个图来表示某个状态空间时，图中各节点标上相应的状态描述，而有向弧线旁边标有算符。

某个节点序列$(n_{i1},n_{i2},\ldots,n_{ik})$，当 j=2,3,…,k 时，如果对于每一个$n_{i,j-1}$都有一个后继节点$n_{ij}$存在，那么就把这个节点序列叫做从节点$n_{i1}$至节点$n_{ik}$的长度为 k 的路径。如果从节点$n_i$至节点$n_j$存在一条路径，那么就称节点$n_j$是从节点$n_i$可达到的节点，或者称节点$n_j$为节点$n_i$的后裔，而且称节点$n_i$为节点$n_j$的祖先。不难发现，寻找从一种状态变换为另一种状态的某个算符序列问题等价于寻求图的某一路径问题。

给各弧线指定的权重用以表示加在相应算符上的代价。用$c(n_i,n_j)$来表示从节点n_i指向节点n_j的那段弧线的代价。两节点间路径的代价等于连接该路径上各节点的所有弧线代价之和。对于最优化问题．要找到两节点间具有最小代价的路径。

对于最简单的一类问题，需要求得某指定节点 s（表示初始状态）与另一节点 t（表示目标状态）之间的一条路径（可能具有最小代价）。

一个图可由显式说明也可由隐式说明。对于显式说明，各节点及其具有代价的弧线由一张表明确给出。此表可能列出该图中的每一节点、它的后继节点以及连接弧线的代价。显然，显式说明对于大型的图是不切实际的，而对于具有无限节点集合的图则是不可能的。

对于隐式说明，节点的无限集合$\{s_i\}$作为起始节点是已知的。此外，引入后继节点算符的概念是方便的。后继节点算符厂也是已知的，它能作用于任一节点以产生该节点的全部后继节点和各连接弧线的代价。把后继算符应用于这，的成员和它们的后继节点以及这些后继节点的后继节点，如此无限地进行下去，最后使得由 Γ 和$\{s_i\}$所规定的隐式图变为显式图。把后继算符应用于节点的过程，就是扩展一个节点的过程。因此，搜索某个状态空间以求得算符序列的一个解答的过程，就对应于使隐式图足够大一部分变为显式以便包含目标节点的过程。这样的搜索图是状态空间问题求解的主要基础。

问题的表示对求解工作量有很大的影响。人们显然希望有较小的状态空间表示。许多似乎很难的问题，当表示适当时就可能具有小而简单的状态空间。

根据问题状态、操作（算）符和目标条件选择各种表示，是高效率问题求解所需要的。首先需要表示问题，然后改进提出的表示。在问题求解过程中，会不断取得经验，获得一些简化的表示。例如，看出对称性或合并为宏规则等有效序列。对于十五数码难题的初始状态表示，可规定 15×0=60 条规则，即左移棋子 1，右移棋子上移棋子 1，下移棋子 1，左移棋子 2，……，下移棋子 15 等。很快就会发现，只要左右上下移动空格，那么就可用 4 条规则代替上述 60 条规则。可见，移动空格是一种较好的表示。

如下图 2.1.7 所示为用有向图描述的状态空间。该图表示对状态S_0允许使用操作算子O_1，O_2及O_3，分别使S_0转换为S_1，S_2及S_3。这样一步步利用操作算子转换下去，如$S_{10}\in G$，则O_2，O_6，O_{10}就是一个解。

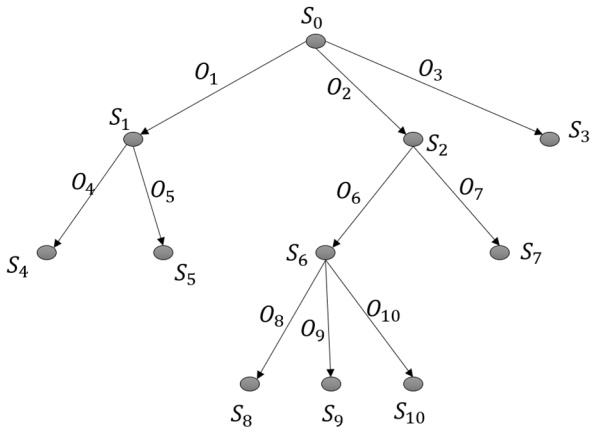


图 2.1.7　状态空间有向图

上述是较为形式化的说明，对于现实中的各种问题，我们都可用状态空间加以表示，并用状态空间搜索法来求解。下面我们以旅行商问题为例，介绍具体问题的状态空间的有向图表示。

在旅行商问题中，各种操作算子的执行是有不同费用的。两城市之间的距离通常不相等，那么，在图中只需要给各弧线标注距离或费用即可。以此题为例说明这类状态空间的图描述，其终止条件则是用解路径本身的特点来描述，即经过图中所有城市的最短路径找到时搜索便结束。

例 2.2 旅行商问题(traveling salesman problem，TSP)或旅行推销员问题：假设一个推销员从出发地到若干个城市去推销产品，然后回到出发地。要求每个城市必须走一次，而且只能走一次。问题是要找到一条最好的路径，使得推销员访问每个城市后回到出发地所经过的路径最短或者费用最少。

图 2.1.8 是这个问题的一个实例，其中结点代表城市，弧上标注的数值表示经过该路径的费用（或距离）。假定推销员从 A 城出发。

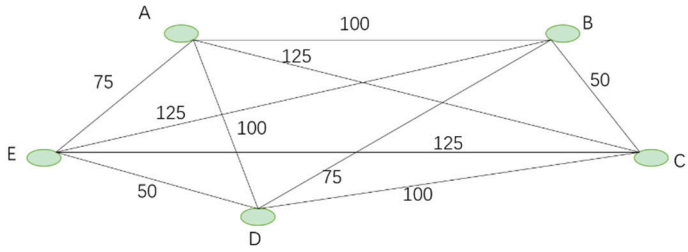


图 2.1.8　旅行商问题实例

图 2.1.9 是该问题的部分状态空间表示。可能的路径有很多，例如，费用为 375 的路径(A，B，C，D，E，A）就是一个可能的旅行路径，但目的是要找具有最小费用的旅行路径。

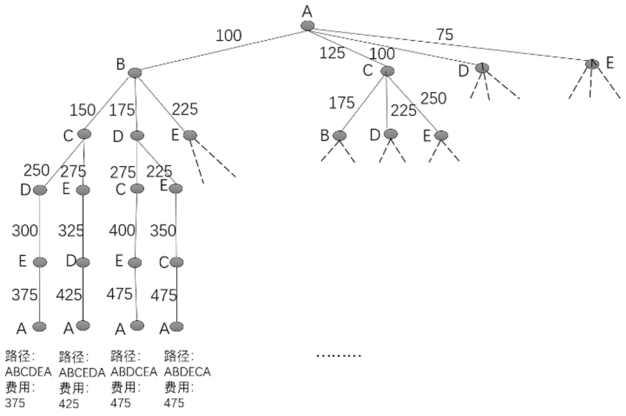


图 2.1.8　旅行商状态空间图

上面两个例子中，只绘出了问题的部分状态空间图。对于许多实际问题，要在有限的时间内绘出问题的全部状态图是不可能的。例如旅行商问题，n 个城市存在$(n-1)!/2$条路径。如果用10^8次/秒的计算机进行穷举，则当 n=7 时，搜索时间为$t=2.5\times10^{-5}$秒；当 n=15 时，$t=1.8$小时；当 n=20 时，$t=350$年；当 n=50 时，$t=5\times10^{48}$年；当 n=100 时，$t=5\times10^{142}$年；当 n=200 时，$t=5\times10^{358}$年。因此，这类显式表示对于大型问题的描述是不切实际的，而对于具有无限结点集合的问题则是不可能的。因此，要研究能够在有限时间内搜索到较好解的搜索算法。

对于实际问题，我们可以将其抽象建模之后用状态空间表示法进行求解。有学者将状态空间表示法应用于疾病预测等问题，提出了基于语音合成和状态空间表示的癫痫综合征概率随机模型。一个概率随机模型处理了无线通信、信号、语音合成、生物医学数据（如血压、心电图、脑电图和人体体温等）的实际应用，其中一类重要的随机过程是马尔可夫过程，它具有过去遗忘特性，那就是每一次事件产生的结果都是依靠现在而不是过去。这种马尔可夫特性使得用模型进行推理和计算成为可能。作者使用马尔可夫链模型作为一种新的技术，讨论了由发热性感染相关癫痫综合征（FIRES）引起的言语障碍，并用图形表示法对其进行了特性分析，以确定有效的言语障碍治疗方法。

有学者提出状态空间表示下 ARMA 模型估计的约束期望最大化算法，主要讨论了线性状态空间模型在四个主要参数矩阵有约束的情况下对给定的多元时间序列的拟合模型。约束部分源于假设模型具有块对角结构，每个块对应一个 ARMA 过程，这样就可以从线性混合中重建独立的源组件，部分原因是需要保持模型可识别的参数拟合的第一阶段是用期望最大化（EM）进行的算法。到期对于可识别性约束，动态噪声协方差矩阵的对角元素的子集需要约束为固定值（通常是统一的），对于这种约束，到目前为止还没有闭式更新规则有空。作者针对这种情况提出新的更新规则，用于直接更新动态噪声协方差矩阵，也可用于更新矩阵的平方根矩阵。

2.1.6 谓词逻辑表示法

虽然命题逻辑（propositional logic）能够把客观世界的各种事实表示为逻辑命题，但是它具有较大的局限性，不适合于表示比较复杂的问题。谓词逻辑（predicate logic）允许表达那些无法用命题逻辑表达的事情。逻辑语句，更具体地说，一阶谓词演算（first order predicate calculus）是一种形式语言，其根本目的在于把数学中的逻辑论证符号化。如果能够采用数学演绎的方式证明一个新语句是从那些已知正确的语句导出的，那么也就能断定这个新语句也是正确的。

2.1.6.1 谓词演算

1、语法和语义

谓词逻辑的基本组成部分是谓词符号、变量符号、数符号和常量符号，并用圆括弧、方括弧、花括弧和逗号隔开，以表示论域内的关系。例如，要表示“机器人（ROBOT）在 1 号房间（ROOMI）内”，可应用简单的原子公式：

$$INROOM(ROBOT,r1)$$

上式中，ROBOT 和 r1 为常量符号，INROOM 为谓词符号。一般，原子公式由谓词符号和项组成。常量符号是最简单的项，用来表示论域内的物体或实体，它可以是实际的物体和人，也可以是概念或具有名字的任何事情。变量符号也是项，并且不必明确涉及是哪一个实体。函数符号表示论域内的函数。例如，函数符号 mother 可用来表示某人与他（或她）的母亲之间的一个映射。用下列原子公式表示“李（LI）的母亲与他的父亲结婚”这个关系：

$$MARRIED(father(LI),mother(LI))$$

在谓词演算中，一个合式公式可以通过规定语言的元素在论域内的关系、实体和函数之间的对应关系来解释。对于每个谓词符号，必须规定定义域内的一个相应关系；对每个常量符号，必须规定定义域内相应的一个实体；对每个函数符号，则必须规定定义域内相应的一个函数。这些规定确定了谓词演算语言的语义。在应用中，用谓词演算明确表示有关论域内的确定语句。对于已定义了的某个解释的一个原子公式，只有当其对应的语句在定义域内为真时，才具有值 T（真）；而当其对应的语句在定义域内为假时，该原子公式才具有值 F（假）。因此 INROOM（ROBOT，r1)具有值 T，而 INROOM（ROBOT，r2）则具有值 F。

当一个原子公式含有变量符号时，对定义域内实体的变量可能有几个设定。对某几个设定的变量，原子公式取值 T；而对另外几个设定的变量，原子公式则取值 F。

2、连词和量词

原子公式是谓词演算的基本积木块，应用连词 Λ（与）、v（或）以及→（蕴含，或隐含）等，能够组合多个原子公式以及构成比较复杂的合法公式。

连词 Λ 用来表示复合句子。例如句子“我喜欢音乐和绘画”可写成：

$$LIKE(I, MUSIC) \wedge LIKE(I, PAINTING)$$

此外，某些较简单的句子也可写成复合形式。例如，“李住在一栋黄色的房子里”即可用

$$LIVES(LI, HOUSE-1) \wedge COLOR(HOUSE-1, YELLOW)$$

来表示，其中谓词 LIVES 表示人与物体（房子）间的关系，而谓词 COLOR 则表示物体与其颜色之间的关系。用连词 Λ 把几个公式连接起来而构成的公式叫做合取（式），而此合取式的每个组成部分叫做合取项。一些合式公式所构成的任一合取也是一个合式公式。

连词 v 用来表示可兼有的“或”。例如，句子“李明打篮球或踢足球”可表示为：

$$PLAYS(LIMING, BASKETBALL) \vee PLAYS(LIMING, FOOTBALL)$$

用连词 v 把几个公式连接起来所构成的公式叫做析取（式），而次析取式的每一组成部分叫做析取项。由一些合式公式所构成的任一析取也是一个合式公式。

合取和析取的真值由其组成部分的真值决定。如果每合取项均取值 T，则值为 T，否则合取值为 F。如果析取项中至少有一个取 T 值，则其析取值为 T，否则取值 F。

连词→用来表示“如果—那么”的词句。例如，“如果该书是何平的，那么它是蓝色（封面）的”可表示为：

$$OWNS(HEPING, BOOK-1) \rightarrow COLOR(BOOK-1, BLUE)$$

又如，“如果刘华跑的最快，那么他取得冠军”可表示为：

$$RUNS(LIUHUA, FASTEST) \rightarrow WINS(LIUHUA, CHAMPION)$$

用连词→连接两个公式所构成的公式叫做蕴涵。蕴涵的左式叫做前项，右式叫做后项。如果前项和后项都是合式公式，那么蕴涵也是合式公式。如果后项取值 T（不管其前项的值为何），或者前项取值 F（不管后项的真值如何），则蕴涵取值 T；否则，蕴涵取值 F。

符号~（非）用来否定一个公式的真值，也就是说，把一个合式公式的取值从 T 变为 F，或从 F 变为 T。例如，子句“机器人不在 2 号房间内”可表示为：

$$\sim INROOM(ROBOT, r2)$$

前面具有符号~的公式叫做否定。一个合式公式的否定也是合式公式。

如果把句子限制为至今已介绍过的造句法所能表示的那些句子，而且也不使用其他项，那么可以把这个谓词演算的子集叫做命题演算。命题演算对于许多简化了的定义域来说，是一种有效的表示，但它缺乏用有效的方法来表达多个命题（如“所有的机器人都是灰色的”）的能力。要扩大命题演算的能力，需要使公式中的命题带有变量。

有时，一个原子公式如$P(x)$，对于所有可能的变量 x 都具有值 T。这个特性可由在$P(x)$前面加上所有的（$\forall\ x$）来表示。如果至少有一个 x 值可以使得$P(x)$具有值 T，那么这一特性可由在$P(x)$前面加上存在符$(\exists\ x)$来表示。例如句子“所有的机器人都是灰色的”可表示为

$$(\forall\ x)(ROBOT(X) \rightarrow \mathrm{COLOR}(\mathrm{x}, \mathrm{GRAY}))$$

而句子“1 号房间内有个物体”可表示为：

$$(\exists\ x)(INROOM(x, r1))$$

这里，x 是被量化了的变量，即 x 是经过量化的。量化一个合式公式中的某个变量所得到的表达式也是合式公式。如果一个合式公式中某个变量是经过量化的，就把这个变量叫做约束变量，否则就称它为自由变量。在合式公式中，感兴趣的主要是所有变量都是受约束的，这样的合式公式叫做句子。

值得指出的是，本书中所用到的谓词演算为一阶谓词演算，不允许对谓词符号或函数符号进行量化。

例如，一阶谓词演算（$\forall\ P$）$P(A)$就不是合式公式，因为P是函数符号。

2.1.6.2 谓词公式

1、谓词公式的定义

定义 2.1 用$P(x_1, x_2, \ldots, x_n)$表示一个n元谓词公式，其中P为n元谓词，$x_1, x_2, \ldots, x_n$为客体变量或变元。通常把$P(x_1, x_2, \ldots, x_n)$叫做谓词演算的原子公式，或原子谓词公式。可以用连词把原子谓词公式组成复合谓词公式，并把它叫做分子谓词公式。为此，用归纳法给出谓词公式的定义。在谓词演算中合式公式的递归定义如下：

（1）原子谓词公式是合式公式。

（2）若A为合式公式，则~A也是一个合式公式。

（3）若A和B都是合式公式，则(A∧B)，（A∨B），（A→B）和（A←→B）也都是合式公式。

（4）若A是合式公式，x为A中的自由变元，则（$\forall\ x$）A和（$\exists\ x$)A都是合式公式。

（5）只有按上述规则（1）至（4）求得的那些公式，才是合式公式。

例2.1 试把下列命题表示为谓词公式：任何整数或者为正或者为负。

解 把上述命题意译如下：

对于所有的x，如果x是整数，则或为正的或者为负的。

用$I(x)$表示“x是整数”，P（x）表示“x是正数”，N（x）表示“x是负数”。于是，可把给定命题用下列谓词公式来表示：

$$(\forall\ x)(I(x) \rightarrow (\mathrm{P(x)} \vee \mathrm{N(x)}))$$

2、合式公式的性质

如果P和Q是两个合式公式，则由这两个合式公式所组成的复合表达式可由下列真值表（表2.1.2）给出。

表 2.1.2 真值表

P	Q	P∨Q	P∧Q	P⇒ Q	~P
T	T	T	T	T	F
F	T	T	F	T	T
T	F	T	F	F	F
F	F	F	F	T	T

如果两个合式公式，无论如何解释，其真值表都是相同的，那么就称此二者是等价的。应用上述真值表，能够确立下列等价关系：

（1）否定之否定

$\sim(\sim P)$等价于P

（2）$P \vee Q$等价于$\sim P \rightarrow Q$

（3）狄摩根定律

$\sim(P \vee Q)$等价于$\sim P \wedge \sim Q$

$\sim(P \wedge Q)$等价于$\sim P \vee \sim Q$

（4）分配律

$P \wedge (Q \vee R)$等价于$(P \wedge Q)(P \wedge R)$

$P \vee (Q \wedge R)$等价于$(P \vee Q)(P \vee R)$

（5）交换律

$P \vee Q$等价于$Q \vee P$

$P \wedge Q$等价于$Q \wedge P$

（6）结合律

$(P \wedge Q) \wedge R$等价于$P \wedge (Q \wedge R)$

$(P \vee Q) \vee R$等价于$P \vee (Q \vee R)$

（7）逆否律

$P \to Q$等价于$\sim Q \to \sim P$

2.1.6.3 置换与合一

1、置换

在谓词逻辑中，有些推理规则可应用于一定的合式公式和合式公式集，以产生新的合式公式。一个重要的推理规则是假元推理，就是由合式公式W_1和$W_1 \to W_2$产生合式公式W_2的运算。另一个推理规则叫做全称化推理，它是由合式公式（∀ x）W（x）产生合式公式 W（A），其中 A 为任意常量符号。同时应用假元推理和全称化推理，例如，可由合式公式$(\forall\ x)(W_1(x) \to W_2(x))$和$W_1(A)$生成合式公式$W_2(A)$，即寻找 A 对 x 的置换(substitution)，使$W_1(A)$与$W_1(x)$一致。

一个表达式的项可为变量符号、常量符号或数表达式。函数表达式由数符号和项组成。一个表达式的置换就是在该表达式中用置换项置换变量。

2、合一

寻找项对变量的置换，以使两表达式一致，叫做合一（unification）。合一是人工智能中很重要的过程。

如果一个置换 s 作用于表达式集$\{E_i\}$的每个元素，则用$\{E_i\}$s 来表示置换例的集。称表达式集$\{E_i\}$是可合一的，如果存在一个置换 s 使得：

$$E_{1s} = E_{2s} = E_{3s} = \cdots$$

那么称此 s 为$\{E_i\}$的合一者（unifier），因为 s 的作用是使集合$\{E_i\}$成为单一形式。

为实现方便，可规定将每个文字和函数符表达成一个表，表中第一元素为谓词名，其余元素为变元。例如，$P(x,y)$表示成$(P\ \ x\ \ y)$，$f(x)$表示成$(f\ \ x)$。若变元为函数，则该变元为一个子表，子表中第一元素为函数名，如$P(f(x),y)$表示成$(P(f\ \ x)\ \ \ y)$这样谓词和函数都统一表示成表。

合一过程$Unify(L_1,L_2)$用一张代换表作为其返回的值。算法如果返回空表NIL，表示可以匹配，但无须任何代换；如果返回由 *F* 值组成的表，则说明合一过程失败。合一算法$Unify(L1,L2)$的过程如下：

（1）若L_1或L_2为一原子，则执行：

①若L_1和L_2恒等，则返回 *NIL*。

②否则，若L_1为一变量，则执行：若L_1出现在L_2中，则返回 *F*；否则返回（L_2/L_1）。

③否则，若L_2为一变量，则执行：若L_2出现在L_1中，则返回 *F*；否则返回（L_1/L_2）。

④否则返回 *F*。

（2）若 *length*(L_1)不等于 *length*(L_2)，则返回 F。

（3）置 *SUBST* 为 *NIL*，在结束本过程时，*SUBST* 将包含用来合一L_1和L_2的所有代换。

（4）对于$i \coloneqq 1$到L_1的元素数|L_1|，执行：

①对合一L_1的第i个元素和L_2的第i个元素调用 *UNIFY*，并将结果放在 *S* 中。

②若 *S*=*F*，则返回 F。

③若 *S* 不等于 *NIL*，则执行：把 *S* 应用到L_1和L_2的剩余部分；*SUBST*：=*APPEND(S,SUBST)*。

④返回 *SUBST*。

2.1.6.4 逻辑谓词表示法的应用

在人工智能中，求解问题的基就是掌握与其相关的知识，将已有的知识利用计算机代码的形式进行描述与存储，并对其进行利用的过程就是知识表示。目前，谓词逻辑是可以表现出人类思维规律的最准确的符号语言，是在人工智能中进行知识表达的最重要的方法。在应用谓词逻辑进行知识表示的过程中，使用谓词逻辑来表示自然语言必须经历三个步骤：首先，将一个基础的命题分解为两个部分，分别为谓词和体词，应用 x，y，z 代替个体变元，并应用 P，Q，R 代替谓词变元，再应用 a，b，c 代替个体常项。其次，在基础命题中找到量词，并应用∃来表示存在，利用∀来表示所有。最后，使用符号∧、∨以及→表示基础命题中的谓词与个体家的关系以及谓词与几个个体词间存在的复合关系。

有学者在自然语言处理方向提出 NLP 本体学习模块谓词识别系统的构建。提出一种识别自然语言句子中描述逻辑谓词的实用方法。方法基于使用开源工具-Link Grammar Parser 与 ProtégéOWL API 的组合。基

于朴素贝叶斯的机器学习识别方法可以区分先前学习的二进制谓词的种类及其属性的作用，其中谓词识别系统包含在 CROCUS 应用程序的本体学习模块中。

2.1.7 实际问题建模

知识图谱中的知识建模包含本体层和实例层，本体通常手工构建，实例通常自动化抽取。构建本体的目的是为了确定知识图谱能描述的知识。知识建模不一定需要本体的形式化表示，不一定需要专业的建模工具，不一定要把本体存储在数据库，可以在程序中直接使用。本体是给知识图谱实施者使用，通过它来确定知识抽取的范围、推理规则、构造查询等。

1、本体的构建

本体构建工程通常是手工完成的，而且可能需要迭代进行，本体的构建大致要经历以下几个步骤：

（1）：确定本体的领域和范围。需要明确一些基本问题，如本体针对的领域、用途，描述什么信息，回答哪一类问题，由谁使用和维护本体等。

（2）：考虑重用现有本体。精炼、扩充、修改网上现有的本体，或从中得到启发和帮助。

（3）：列出本体中的重要术语。主要是列出建模过程中所必需的实体、属性、关系，使得创建的本体不要偏离领域。

（4）：定义类和类的继承。确保类的继承（is-a、kind-of）正确，分析继承结构中的兄弟类，新类或属性值的取舍等。

（5）：定义属性和关系。定义类之后往往还要定义其概念和概念间的内部联系。这里的联系分为内部属性（Datatype Property）和外部属性（Object Property）。内部属性具有通用性，用来连接一个概念和一个值；外部属性，也称为关系，通常用于连接概念间的实例。

（6）：定义属性的限制。如属性的基数，属性的类型，属性的定义域和值域。

（7）：为类创建实例。添加个体作为该类的实例后，同时要为实例的属性赋值。给本体补充实例数据时，需要考虑不同的数据来源：

对于结构化数据，一般是把一个表——类别去对应，一行数据相当于一个类的实例，每个字段就相当于类的属性，利用相关的转化工具可以把这个过程进行转化，比如 D2RQ，可以把表数据转化成虚拟的 RDF 数据，使数据在 RDF 层面实现数据格式的统一。

对于半结构化数据，主要是指那些具有一定的数据结构，但需要进一步提取整理的数据。比如百科的数据，网页中的数据等。对于这类数据，主要采用包装器的方式进行处理。

对于非结构数据，若数据库中已经存在一些单实体以及三元组数据，目前的主要任务就是从文本中抽取相关的数据，来补充现有的知识库。

2.2 知识图谱概念

随着互联网的发展，网络数据内容呈现爆炸式增长的态势。由于互联网内容的大规模、异质多元、组织结构松散的特点，给人们有效获取信息和知识提出了挑战。知识图谱（Knowledge Graph）以其强大的语义处理能力和开放组织能力，为互联网时代的知识化组织和智能应用奠定了基础。最近，大规模知识图谱库的研究和应用在学术界和工业界引起了足够的注意力[1-5]。一个知识图谱旨在描述现实世界中存在的实体以及实体之间的关系。知识图谱于 2012 年 5 月 17 日由 Google 正式提出[6]，其初衷是为了提高搜索引擎的能力，改善用户的搜索质量以及搜索体验。随着人工智能的技术发展和应用，知识图谱作为关键技术之一，已被广泛应用于智能搜索、智能问答、个性化推荐、内容分发等领域。

2.2.1 知识图谱基本原理

一、定义

在维基百科的官方词条中：知识图谱是 Google 用于增强其搜索引擎功能的知识库。本质上, 知识图谱旨在描述真实世界中存在的各种实体或概念及其关系,其构成一张巨大的语义网络图，节点表示实体或概念，边则由属性或关系构成。现在的知识图谱已被用来泛指各种大规模的知识库。 在具体介绍知识图谱的定义，我们先来看下知识类型的定义：

知识图谱中包含三种节点：

实体: 指的是具有可区别性且独立存在的某种事物。如某一个人、某一个城市、某一种植物等、某一种商品等等。世界万物有具体事物组成，此指实体。如图 2.2.1 的“中国”、“美国”、“日本”等。，实体是知识图谱中的最基本元素，不同的实体间存在不同的关系。

语义类（概念）：具有同种特性的实体构成的集合，如国家、民族、书籍、电脑等。 概念主要指集合、类别、对象类型、事物的种类，例如人物、地理等。

内容: 通常作为实体和语义类的名字、描述、解释等，可以由文本、图像、音视频等来表达。

属性(值): 从一个实体指向它的属性值。不同的属性类型对应于不同类型属性的边。属性值主要指对象指定属性的值。如图 1 所示的“面积”、“人口”、“首都”是几种不同的属性。属性值主要指对象指定属性的值，例如 960 万平方公里等。

关系: 形式化为一个函数，它把 kk 个点映射到一个布尔值。在知识图谱上，关系则是一个把 kk 个图节点(实体、语义类、属性值)映射到布尔值的函数。

基于上述定义。基于三元组是知识图谱的一种通用表示方式，即 G=（E，R，S，）,其中 $E=\{e_1,e_2,...,e_{|E|}\}$，是知识库中的实体集合，共包含|E|种不同实体； $R=\{r_1,r_2,...,r_{|R|}\}$ 是知识库中的关系集合，共包含|R|种不同关系；代表知识库中的三元组集合。三元组的基本形式主要包括(实体 1-关系-实体 2)和(实体-属性-属性值)等。每个实体(概念的外延)可用一个全局唯一确定的 ID 来标识，每个属性-属性值对(attribute-value pair，AVP)可用来刻画实体的内在特性，而关系可用来连接两个实体，刻画它们之间的关联。

如下图 1 的知识图谱例子所示，中国是一个实体，北京是一个实体，中国-首都-北京 是一个（实体-关系-实体）的三元组样例北京是一个实体 ，人口是一种属性 2069.3 万是属性值。

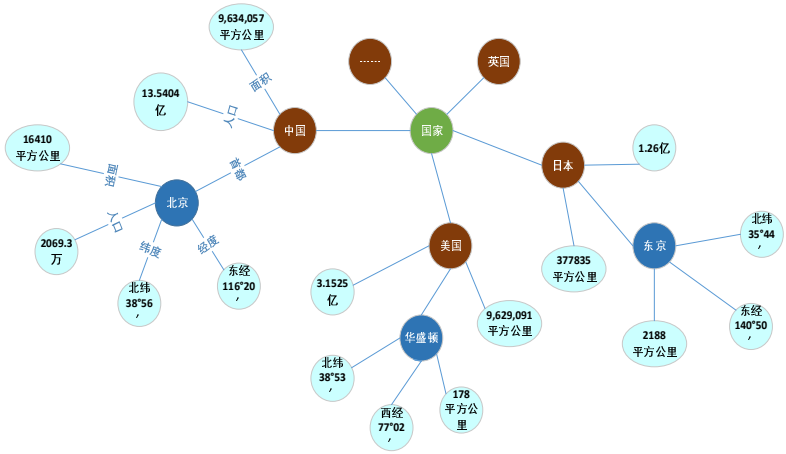


图 2.2.1 知识图谱示例

二、知识图谱的架构

知识图谱的架构包括自身的逻辑结构以及构建知识图谱所采用的技术（体系）架构。

1） 知识图谱的逻辑结构

知识图谱在逻辑上可分为模式层与数据层两个层次，数据层主要是由一系列的事实组成，而知识将以事实为单位进行存储。如果用(实体 1，关系，实体 2)、(实体、属性，属性值)这样的三元组来表达事实，可选择图数据库作为存储介质，例如开源的 Neo4j、Twitter 的 FlockDB、sones 的 GraphDB[]等。模式层构建在数据层之上，是知识图谱的核心，通常采用本体库来管理知识图谱的模式层。本体是结构化知识库的概念模板，通过本体库而形成的知识库不仅层次结构较强，并且冗余程度较小。

2） 知识图谱的体系架构

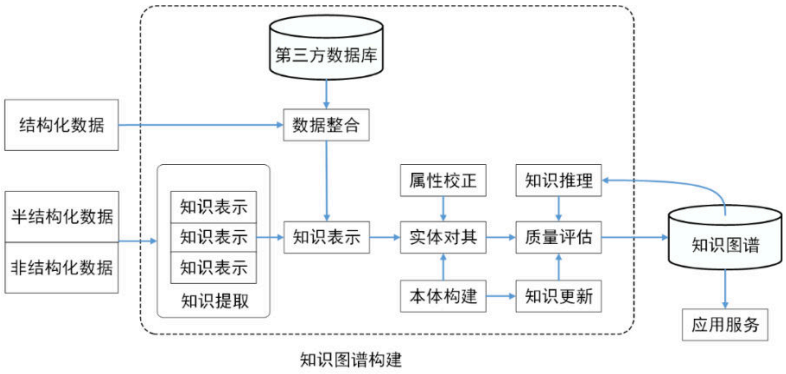


图 2.2.2 知识图谱的技术架构

知识图谱的体系架构是其指构建模式结构，如图 2.2.2 所示。其中虚线框内的部分为知识图谱的构建过程，也包含知识图谱的更新过程。知识图谱构建从最原始的数据（包括结构化、半结构化、非结构化数据）出发，采用一系列自动或者半自动的技术手段，从原始数据库和第三方数据库中提取知识事实，并将其存入知识库的数据层和模式层，这一过程包含：信息抽取、知识表示、知识融合、知识推理四个过程，每一次更新迭代均包含这四个阶段。知识图谱主要有自顶向下(top-down)与自底向上(bottom-up)两种构建方式。自顶向下指的是先为知识图谱定义好本体与数据模式，再将实体加入到知识库。该构建方式需要利用一些现有的结构化知识库作为其基础知识库，例如 Freebase 项目就是采用这种方式，它的绝大部分数据是从维基百科中得到的。自底向上指的是从一些开放链接数据中提取出实体，选择其中置信度较高的加入到知识库，再构建顶层的本体模式。目前，大多数知识图谱都采用自底向上的方式进行构建，其中最典型就是 Google 的 Knowledge Vault 和微软的 Satori 知识库。现在也符合互联网数据内容知识产生的特点。

代表性知识图谱：

根据覆盖范围而言，知识图谱也可分为开放域通用知识图谱和垂直行业知识图谱[12]。开放通用知识图谱注重广度，强调融合更多的实体，较垂直行业知识图谱而言，其准确度不够高，并且受概念范围的影响，很难借助本体库对公理、规则以及约束条件的支持能力规范其实体、属性、实体间的关系等。通用知识图谱主要应用于智能搜索等领域。行业知识图谱通常需要依靠特定行业的数据来构建，具有特定的行业意义。行业知识图谱中，实体的属性与数据模式往往比较丰富，需要考虑到不同的业务场景与使用人员。表 2.2.1 展示了现在知名度较高的大规模知识库。

表 2.2.1 代表性知识图谱库概览

知识图谱库名称	机构	特点、构建手段	应用产品
FreeBase	MetaWeb（2010 年被谷歌收购）	•实体、语义类、属性、关系； •自动+人工：部分数据从维基百科等数据源抽取 而得到；另一部分数据来自人工协同编辑 • https://developers.google.com/freebase/	Google Search Engine , Google Now
Knowledge Vault （谷歌知识图 谱）	Google	•实体、语义类、属性、关系; •超大规模数据库；源自维基百科、Freebase.《世界各国纪实年鉴》 • https://research.google.com/pubs/pub45634	Google Search Engine，Google Now
DBpedia	莱比锡大学、柏林自由大 学、OpenLink Software	•实体、语义类、属性、关系 •从维基百科抽取	DBPedia

维基数据（Wikidata）	维基媒体基金会(Wikimedia Foundation)	•实体、语义类、属性、关系与维基百科紧密结合 •人工（协同编辑）	Wikipedia
Wolfram Alpha	沃尔夫勒姆公司（Wolfram Research）	•实体、语义类、属性、关募知识计算 •部分知识来自于 Mathematica；其它知识来自于 各个垂直网站	Apple Siri
Bing Satori	Microsoft	•实体、语义类、属性、关系知识计算 •自动+人工	Bing Search Engine：Microsoft Cortana
YAGO	马克斯•普朗克研究所	•自动：从维基百科、WordNet 和 GeoNames 提取信息	YAGO
Facebook Social Graph	Facebook	• Facebook 社交网络数据	Social Graph Search
百度知识图谱	百度	•搜索结构化数据	百度搜索
搜狗知立方	搜狗	•搜索结构化数据	搜狗搜索
ImageNet	斯坦福大学	•搜索引擎 •亚马逊 AMT	计算机视觉相关应用

三、知识图谱构建的关键技术

大规模知识库的构建与应用需要多种技术的支持。通过知识提取技术，可以从一些公开的半结构化、非结构化和第三方结构化数据库的数据中提取出实体、关系、属性等知识要素。知识表示则通过一定有效手段对知识要素表示，便于进一步处理使用。然后通过知识融合，可消除实体、关系、属性等指称项与事实对象之间的歧义，形成高质量的知识库。知识推理则是在已有的知识库基础上进一步挖掘隐含的知识，从而丰富、扩展知识库。分布式的知识表示形成的综合向量对知识库的构建、推理、融合以及应用均具有重要的意义。接下来，本文将以知识抽取为重点，选取代表性的方法，说明其中的相关研究进展和实用技术手段 。

1 知识提取

知识抽取主要是面向开放的链接数据，通常典型的输入是自然语言文本或者多媒体内容文档（图像或者视频）等。然后通过自动化或者半自动化的技术抽取出可用的知识单元，知识单元主要包括实体(概念的外延)、关系以及属性 3 个知识要素，并以此为基础，形成一系列高质量的事实表达，为上层模式层的构建奠定基础。

一） 实体抽取

实体抽取也称为命名实体学习(named entity learning) 或命名实体识别 (named entity recognition)，指的是从原始数据语料中自动识别出命名实体。由于实体是知识图谱中的最基本元素，其抽取的完整性、准确率、召回率等将直接影响到知识图谱构建的质量。因此，实体抽取是知识抽取中最为基础与关键的一步。参照文献[13]，我们可以将实体抽取的方法分为 4 种：基于百科站点或垂直站点提取、基于规则与词典的方法、基于统计机器学习的方法以及面向开放域的抽取方法。基于百科站点或垂直站点提取则是一种很常规基本的提取方法；基于规则的方法通常需要为目标实体编写模板，然后在原始语料中进行匹配；基于统计机器学习的方法主要是通过机器学习的方法对原始语料进行训练，然后再利用训练好的模型去识别实体；面向开放域的抽取将是面向海量的 Web 语料。

1) 基于百科或垂直站点提取

基于百科站点或垂直站点提取这种方法是从百科类站点（如维基百科、百度百科、互动百科等）的标题和链接中提取实体名。这种方法的优点是可以得到开放互联网中最常见的实体名，其缺点是对于中低频

的覆盖率低。与一般性通用的网站相比，垂直类站点的实体提取可以获取特定领域的实体。例如从豆瓣各频道(音乐、读书、电影等)获取各种实体列表。这种方法主要是基于爬取技术来实现和获取。基于百科类站点或垂直站点是一种最常规和基本的方法。

2) 基于规则与词典的实体提取方法

早期的实体抽取是在限定文本领域、限定语义单元类型的条件下进行的，主要采用的是基于规则与词典的方法，例如[Chinchor and Marsh , 1998]使用已定义的规则，抽取出文本中的人名、地名、组织机构名、特定时间等实体。[Rau, 1998]首次实现了一套能够抽取公司名称的实体抽取系统，其中主要用到了启发式算法与规则模板相结合的方法。然而，基于规则模板的方法不仅需要依靠大量的专家来编写规则或模板，覆盖的领域范围有限，而且很难适应数据变化的新需求。

3) 基于统计机器学习的实体抽取方法

鉴于基于规则与词典实体的局限性，为具更有可扩展性，相关研究人员将机器学习中的监督学习算法用于命名实体的抽取问题上。例如[LIU Xiao-hua, ZHANG Shao-dian, WEI Fu-ru, et al., 2011]利用 KNN 算法与条件随机场模型，实现了对 Twitter 文本数据中实体的识别。单纯的监督学习算法在性能上不仅受到训练集合的限制，并且算法的准确率与召回率都不够理想。相关研究者认识到监督学习算法的制约性后，尝试将监督学习算法与规则相互结合，取得了一定的成果。例如[LIN Yi-feng, TSAI T, CHOU Wen-chi, et al., 2004]基于字典，使用最大熵算法在 Medline 论文摘要的 GENIA 数据集上进行了实体抽取实验，实验的准确率与召回率都在 70%以上。近年来随着深度学习的兴起应用，基于深度学习的命名实体识别得到广泛应用。在[Guillaume Lample, 2016]中，介绍了一种基于双向 LSTM 深度神经网络和条件随机场的识别方法，在测试数据上取得的最好的表现结果。

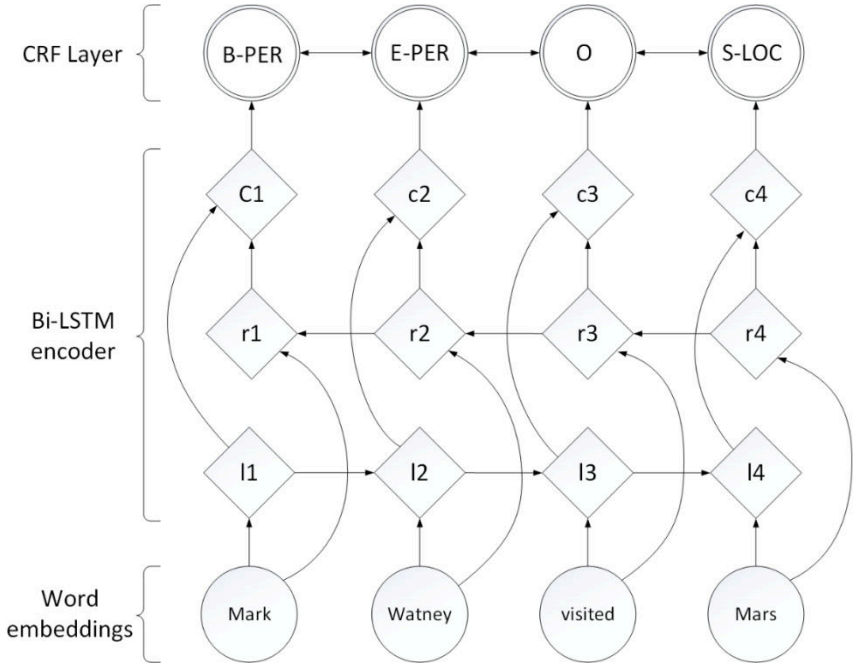


图 2.2.3　基于 BI-LSTM 和 CRF 的架构

4) 面向开放域的实体抽取方法

针对如何从少量实体实例中自动发现具有区分力的模式，进而扩展到海量文本去给实体做分类与聚类的问题，[Whitelaw c, Kehlenbeck a, Petrovic n, et al., 2008]提出了一种通过迭代方式扩展实体语料库的解决方案，其基本思想是通过少量的实体实例建立特征模型，再通过该模型应用于新的数据集得到新的命名实体。文献[21]提出了一种基于无监督学习的开放域聚类算法，其基本思想是基于已知实体的语义特征去搜索日志中识别出命名的实体，然后进行聚类。

2.2.2 知识图谱的生命周期

作为大数据环境下知识工程的标志性产物，知识图谱同样遵循如图 2.2.4 所示的知识建模、知识获取、知识集成、知识共享与应用的生命周期。本节在分析现有知识资源特点的基础上，简要介绍几个典型任务和相应的主要研究方法。传统的知识建模和获取主要是从领域专家处获取专业知识的过程。互联网的迅速发展正在逐渐改变知识的产生方式，知识资源变得丰富多样，而建模和获取任务也就需要针对知识资源的特性进行相应的调整和改变。因此，在介绍具体的学习过程之前，我们首先对学习资源及其特性进行简要分析。

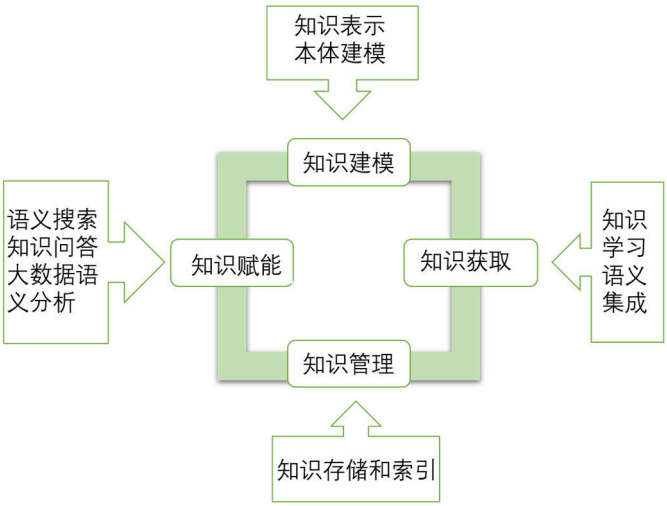


图 2.2.4 知识图谱

简单来说，获取知识的资源对象大体可分为结构化、半结构化和非结构化三类。结构化数据是指知识定义和表示都比较完备的数据，如 DBpedia 和 Freebase 等已有知识图谱、特定领域内的数据库资源等。半结构化数据的典型代表是百科类网站，虽然知识的表示和定义并不一定规范统一，其中部分数据（如信息框、列表和表格等）仍遵循特定表示以较好的结构化程度呈现，但仍存在大量结化程度较低的数据，一些领域的介绍和描述类页面往往也都归在此类，如电脑、手机等电子产品的参数性能分析介绍。非结构化数据则是指没有定义和规范约束的“自由”数据，包括最广泛存在的自然语言文本、音视频等。互联网时代，知识在数据中的分布有如下特点。

（1）多媒体性：同一知识可能表达为不同的媒体形式，如维基百科的词条可能包括文本描述、图片展示以及结构化的信息框等。

（2）隐蔽性：很多有价值的知识可能存在于网页链接或者资源文件中，如语义网知名教授 James Hendler 的个人主页中，关于其个人简介的结构化表达就“藏”在一个单独的 RDF 文件中。

（3）分布性：关于同一事物的不同方面的知识往往分布也各异，如科研人员的基本信息可以在其主页和个人简历中获取，论著发表情况收录于权威的 ACM 或者 DBLP 数据库中，而其参与的学术活动信息则只能通过相关活动的页面获得。

（4）异构性：知识的分布表达和定义不可避免地造成异构性，即不同用户对同一知识的表达和理解存在或多或少的差异。以不同类型信息的组织管理需要用到的分类体系为例，比较著名的有开放式分类目录搜索系统（open directory project ,ODP），但是不同的门户网站和导航网站往往会根据需要定义各自的分类系统。

（5）海量性：较之传统人工编撰的知识库，互联网上知识的规模巨大。例如，维基百科共收录了约 6 開万的英文词条和 90 万的中文词条，事实知识的条目更是数以亿计。

上述特点给知识图谱构建带来了机遇和挑战。

知识建模：是定义领域知识描述的概念、事件、规则及其相互关系的知识表示方法，建立知识图谱的概念模型。主要包括领域概念及概念层次（上下位关系）学习、概念属性学习等。

概念是人们理解客观世界的线索，是人们对客观世界中的事物在不同层次上的概念化描述，概念层次

是知识图谱的“骨骼”。如表 2.2.2 所示，即使是现有结构化比较好的知识图谱，其概念体系也存在诸多问题，如概念数量少、知识覆盖率低；上下位关系稀疏、概念扁平化组织、知识的精确度低；上下位关系错误和噪声多、概念结构混乱。这些问题都严重影响知识图谱对客观世界的描述能力，制约和限制了知识的共享和应用。因此，概念层次学习的目的是确定概念与子概念之间的关系，判断两个概念之间是否存在上下位关系。基本步骤是首先进行概念抽取，然后对概念间上下位关系进行识别，最后将概念以识别得到的上下位关系为基础组织成树或有向无环图的结构。

表 2.2.2　不同知识图谱概念信息的统计

知识图谱	概念数	上下位关系数
Freebase	1450	24483434
WordNet	25229	283070
WikiTaxonomy	111654	105418
YAGO	352297	8277227
DBpedia	259	1900000
ResearchCyc	≈ 120000	≈ 5000000
KnowItAll	N/A	54753
TextRunner	N/A	≈ 11000000
OMCS	173398	1030619
NELL	123	242453
Probase	2653872	20757545

属性是概念的内涵表示，描述概念的特征或性质，具有描述和鉴别概念的功能。属性获取是概念知识建模和领域知识库自动构建的关键步骤。概念属性抽取旨在对给定概念从不同类型的数据源中自动获取其属性集合。

知识获取：是对知识建模定义的知识要素进行实例化的过程。知识图谱中实例的属性描述以三元组的形式表示，其数量决定了知识图谱的丰富程度。因此，建模完成之后，通常采用不同类型的机器学习方法从多源异构的数据源中进行事实型知识的学习。其中有监督的方法包括基于规则、分类、序列标注等方法，半监督则以扩展（boosting）方法和远程监督（distant supervision）方法为主。除此之外，开放信息抽取(open information extraction, open IE）是无监督的事实知识获取的典型代表， KnowItAll、TextRunner 和 NELL 都是此类方法的典型成果。

知识图谱间的分布性和异构性阻碍了知识在整个语义网上的共享。语义集成就是在异构知识图谱之间发现实体（概念、属性或实例）间的等价关系，从而实现知识共享。由于知识图谱多以本体的形式定义和描述，因此知识图谱语义集成的核心是本体模式层和实例层的匹配问题，即本体映射。Shvaiko 和 Euzenat 将匹配方法分为基于实体和基于结构两类。前者在计算相关性时，独立地对实体进行分析，而不考虑实体与其他实体的关系，多利用实体相关的文本信息如字符串、自然语义等后者通过分析实体与其他实体在结构中的关系来计算相关性，主要是基于图结构的匹配。高质量大规模的知识获取和语义集成是大数据环境下知识图谱构建的一项艰巨任务，还有许多问题亟待研究。

知识管理：主要研究图谱知识的存储和索引，方便快速访问和查询。知识图谱是典型的图结构，知识图谱的管理利用图数据库实现，感兴趣的同学可自行查阅相关资料。

知识赋能：知识图谱最初提出的目的是增强搜索结果，改善用户搜索体验，即语义搜索。但其应用方式远不止如此，知识图谱还可以应用于知识问答、领域大数据分析等。知识问答是通过对问句的语义分析，将非结构化问句解析成结构化的查询，从已有结构化的知识图谱中获取答案，使用户可直接获得问题的答案。知识驱动的大数据分析与决策是另一种典型的应用方式，借助知识图谱丰富准确的知识结点和广泛的关系网络对语义稀疏的领域大数据进行分析理解，为行业决策提供有力支撑。

2.2.3　本体知识表示

本体是对领域实体存在本质的抽象，他强调实体间的关联，并通过多种知识表示元素将这些关联表达

和反映出来，这些知识表示元素也被称为元本体，主要包括：

概念——表示领域知识元，包括一般意义上的概念以及任务、功能、策略、行为、过程等，在本体的实现中，概念通常用类（class）来定义，而且通常具有一定的分类层次关系；

属性——描述概念的性质，是一个概念区别于其他概念的特征，通常用槽（slot）或者类的属性（Properties）来定义；

关系——表示概念之间的关联，例如一些常用的关联：父关系、子关系、相等关系；

函数——表示一类特殊的关系，即由前 n-1 个要素来唯一决定第 n 个要素，如：长方形的长和宽唯一决定其面积；

公理——表示永真式，在本体论中，对于属性、关系和函数都具有一定的关联和约束，这些约束就是公理，公理一般用槽的侧面（facet）来定义；

实例——表示某个概念类的具体实体。

本体的每一个知识表示元素也可以被看作一个知识片，每一个知识片都包含名称、定义和文档说明。

总的来说，构造本体的目的都是为了实现某种程度的知识共享和重用。从广义来讲，本体的作用主要有以下两方面：

本体的分析澄清了领域知识的结构，从而为知识表示打好基础。本体可以重用，从而避免重复的领域知识分析；

统一的术语和概念使知识共享成为可能。

较为具体来讲，本体的作用包括三个方面：即交流（communication）、互操作（inter-operability）和系统工程（system engineering）：

交流：主要为人与人之间或组织与组织之间的交流提供共同的词汇；

互操作：在不同的建模方式、范式、语言和软件工具之间进行翻译和映射，以实现不同系统之间的互操作和集成；

系统工程：本体分析能够为系统工程提供以下方面的好处：重用（reusability）：本体是领域内重要实体、属性、工程及其相互关系形式化描述的基础。这种形式化描述可成为软件系统中可重用和共享的组件；知识获取（knowledge acquisition）：当构造基于知识的系统时，用已有的本体作为起点和基础来指导知识的获取，可以提高其速度和可靠性；可靠性（reliabililty）：形式化的表达使得自动的一致性检查成为可能，从而提高了软件的可靠性：规范描述（specification）：本体分析有助于确定 IT 系统（如知识库）的需求和规范。

本体作为一种知识表示方法，与谓词逻辑、框架等其他方法的区别在于他们属于不同层次的知识表示方法，本体表达了概念的结构、概念之间的关系等领域中实体的固有特征，即“共享概念化”，而其他的知识表示方法如语义网络等，可以表达某个体对实体的认识，不一定是实体的固有特征。这正是本体层与其它层的知识表示方法的本质区别。

知识工程师将本体概念引入知识工程，详细说明模型中涵盖的概念。实例、关系和公理等实体，并以此建立本体。通过使用元属性对属性进行分析，并对属性提出了一种针对本体建模概念化分析的形式化方法，解决了知识共享中的一些问题，有效地促进了来自不同领域的研究人员和组织间的交流。显而易见，基于本体的知识表示法在知识表示方面有很大的潜力。

2.2.4 语义网络的定义和原理

一、定义

语义网络（semantic network）是一种以网络格式表达人类知识构造的形式。是人工智能程序运用的表示方式之一。由奎林(J. R. Quillian)于 1968 年提出。开始是作为人类联想记忆的一个明显公理模型提出，随后在 AI 中用于自然语言理解，表示命题信息。在 ES 中语义网络由 PROSPEUTOR 实现，用于描述物体概念与状态及其间的关系。它是由结点和结点之间的弧组成，结点表示概念(事件、事物)，弧表示它们之间的关系。在数学上语义网络是一个有向图，与逻辑表示法对应。

二、原理

语义网络（semantic network）是一种用图来表示知识的结构化方式。在一个语义网络中，信息被表达为一组结点，结点通过一组带标记的有向直线彼此相连，用于表示结点间的关系。

在人工智能的程序中，谓词及其变元可以看作是语义网络中的结点；而格关系则相当于结点之间的连结形式。语义网络是一种面向语义的结构，它们一般使用一组推理规则，规则是为了正确处理出现在网络中的特种弧而专门设计的。

语义网络对表达典型的陈述句子的内容尤其有用。例如，语句 John gave the book to Mary 的语义网络表示如下图。

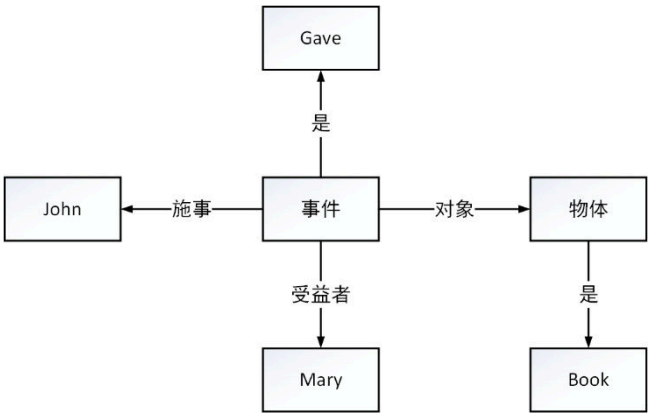


图 2.2.4　语义网络表示图

语义网络的一个重要特性是属性继承。凡用有向弧连结起来的两个结点有上位与下位关系。例如“兽”是“动物”的下位概念，又是“虎”的上位概念。所谓“属性继承”指的是凡上位概念具有的属性均可由下位概念继承。在属性继承的基础上可以方便地进行推理是语义网络的优点之一。

语义网络的特点是：

(1)可以深层次地表示知识，包括实体结构、层次及实体间的因果关系;

(2)推理的非有规则，无推理规律可循;

(3)知识表达的自然性直接从语言语句强化而来。

它的优点是：

(1)直接而明确地表达概念的语义关系，模拟人的语义记忆和联想方式;

(2)可利用语义网络的结构关系检索和推理，效率高。但它不适用于定量、动态的知识;不便于表达过程性、控制性的知识。

与逻辑推理相比，其特点是：语义网络能表示各种事实和规则，具有结构化的特点;逻辑术语把事实与规则当作独立的事实处理，语义网络则从整体上进行处理;逻辑系统有特定的演绎结构，而语义网络不具有特定的演绎结构;语义网络推理是知识的深层次推理，是知识的整体表示与推理。

2.2.5　知识图谱应用实例

这里以开源的医疗知识图谱的问答项目[1]为例子让读者能够对知识图谱的构建有进一步的了解。想进一步实现的同学可以参考这篇博客 https://blog.csdn.net/lhy2014/article/details/82953792。

该项目是一个基于 Python 和 neo4j 实现以疾病为中心具有一定规模的并能实现自动问答和分析的对话系统。其中实体规模 4.4 万，实体关系规模 30 万。

2.2.5.1　项目配置及运行

(1)配置要求:neo4j 数据库以及相应的 Python 依赖。

(2)知识图谱数据的导入:从项目中导入 build_medicalgraph.py

(3) 启动问答：python chat_graph.py

[1] https://github.com/zhihao-chen/QASystemOnMedicalKG

2.2.5.2 知识驱动的知识图谱构建框架

由知识驱动的知识图谱构建框架主要分为三层即：需求层、模型层和应用层。如图 2.2.5 所示。

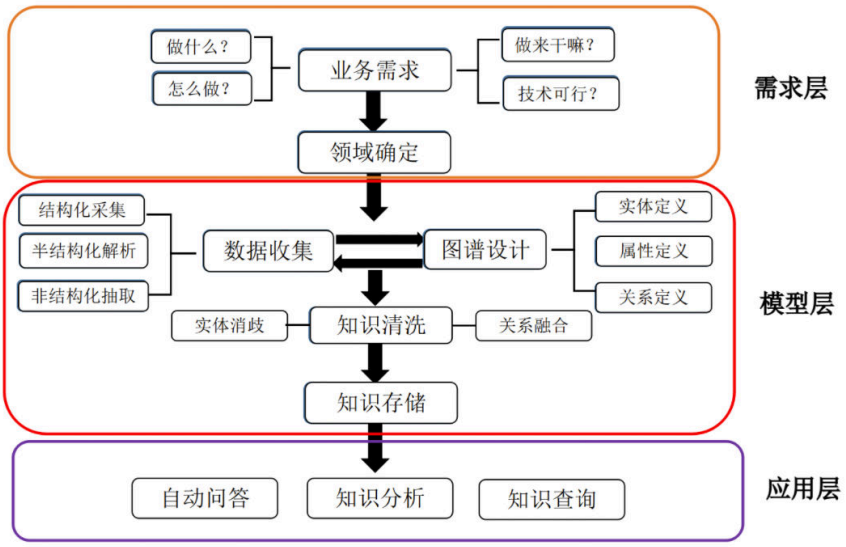


图 2.2.5 知识驱动的知识图谱构建框架

2.2.5.3 医疗知识图谱规模

本项目是基于 neo4j 图数据库存储的，neo4j 图数据库存储规模如图 2.2.6 所示。

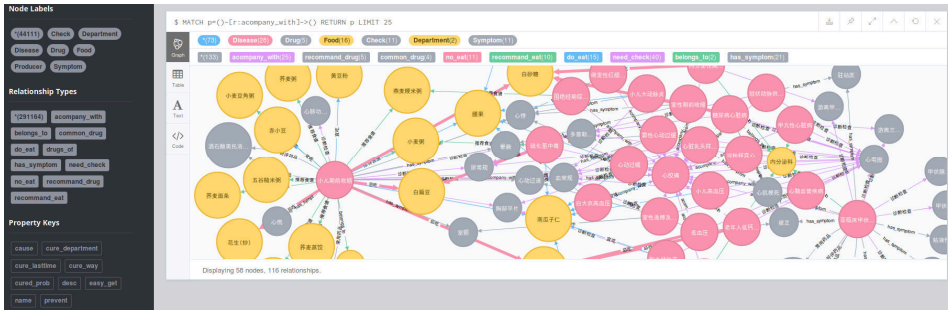


图 2.2.6 neo4j 图数据库存储规模

2.2.5.4 知识图谱实体类型

该项目实体规模 4.4 万，实体类型如表 2.2.3 所示。

表 2.2.3 知识图谱实体类型

实体类型	中文含义	实体数量	举例
Check	诊断检查项目	3,353	支气管造影;关节镜检查
Department	医疗科目	54	整形美容科;烧伤科
Disease	疾病	8,807	血栓闭塞性脉管炎;胸降主动脉动脉瘤
Drug	药品	3,828	京万红痔疮膏;布林佐胺滴眼液
Food	食物	4,870	番茄冲菜牛肉丸汤;竹笋炖羊肉
Producer	在售药品	17,201	通药制药青霉素 V 钾片;青阳醋酸地塞米松片
Symptom	疾病症状	5,998	乳腺组织肥厚;脑实质深部出血
Total	总计	44,111	约 4.4 万实体量级

2.2.5.5 知识图谱实体关系类型

该项目关系规模 30 万，实体关系类型如表 2.2.4 所示。

表 2.2.4 知识图谱关系规模

实体关系类型	中文含义	关系数量	举例

belongs_to	属于	8,844	<妇科,属于,妇产科>
common_drug	疾病常用药品	14,649	<阳强,常用,甲磺酸酚妥拉明分散片>
drugs_of	药品在售药品	17,315	<青霉素 V 钾片,在售,通药制药青霉素 V 钾片>
need_check	疾病所需检查	39,422	<单侧肺气肿,所需检查,支气管造影>
no_eat	疾病忌吃食物	22,247	<唇病,忌吃,杏仁>
recommand_drug	疾病推荐药品	59,467	<混合痔,推荐用药,京万红痔疮膏>
recommand_eat	疾病推荐食谱	40,221	<鞘膜积液,推荐食谱,番茄冲菜牛肉丸汤>
has_symptom	疾病症状	5,998	<早期乳腺癌,疾病症状,乳腺组织肥厚>
acompany_with	疾病并发疾病	12,029	<下肢交通静脉瓣膜关闭不全,并发疾病,血栓闭塞性脉管炎>
Total	总计	294,149	约 30 万关系量级

2.2.5.6 知识图谱属性类型

该知识图谱的属性类型包括八种，如表 2.2.5 所示。

表 2.2.5 知识图谱属性类型

属性类型	中文含义	举例
name	疾病名称	喘息样支气管炎
desc	疾病简介	又称哮喘性支气管炎...
cause	疾病病因	常见的有合胞病毒等...
prevent	预防措施	注意家族与患儿自身过敏史...
cure_lasttime	治疗周期	6-12 个月
cure_way	治疗方式	"药物治疗","支持性治疗"
cured_prob	治愈概率	95%
easy_get	疾病易感人群	无特定的人群

2.2.5.7 技术架构

基于医疗知识图谱的自动问答的技术架构如图 2.2.7 所示。

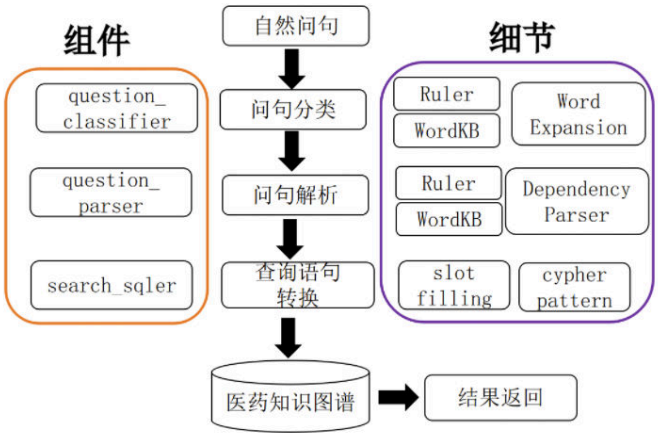


图 2.2.7 基于医疗知识图谱自动问答的技术架构

2.2.5.8 支持问答类型

基于医疗知识图谱的自动问答支持的问答类型如表 2.2.6 所示。

表 2.2.6 基于医疗知识图谱的自动问答支持的问答类型

问句类型	中文含义	问句举例
disease_symptom	疾病症状	乳腺癌的症状有哪些？

symptom_disease	已知症状找可能疾病	最近老流鼻涕怎么办？
disease_cause	疾病病因	为什么有的人会失眠？
disease_acompany	疾病的并发症	失眠有哪些并发症？
disease_not_food	疾病需要忌口的食物	失眠的人不要吃啥？
disease_do_food	疾病建议吃什么食物	耳鸣了吃点啥？
food_not_disease	什么病最好不要吃某事物	哪些人最好不好吃蜂蜜？
food_do_disease	食物对什么病有好处	鹅肉有什么好处？
disease_drug	啥病要吃啥药	肝病要吃啥药？
drug_disease	药品能治啥病	板蓝根颗粒能治啥病？
disease_check	疾病需要做什么检查	脑膜炎怎么才能查出来？
check_disease	检查能查什么病	全血细胞计数能查出啥来？
disease_prevent	预防措施	怎样才能预防肾虚？
disease_lasttime	治疗周期	感冒要多久才能好？
disease_cureway	治疗方式	高血压要怎么治？
disease_cureprob	治愈概率	白血病能治好吗？
disease_easyget	疾病易感人群	什么人容易得高血压？
disease_desc	疾病描述	糖尿病

2.2.5.9 运行效果

基于医疗知识图谱的自动问答系统运行结果如图 2.2.8 所示。

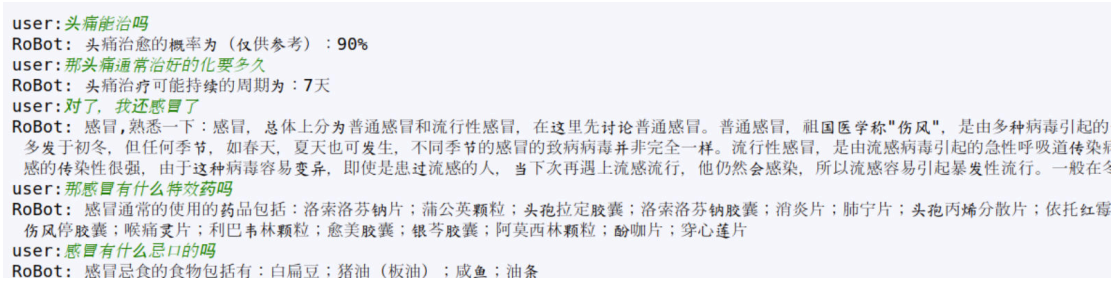


图 2.2.8 基于医疗知识图谱的自动问答系统运行效果

2.2.5.10 实验代码

(1) 图数据库、实体节点和实体节点关系建立

```
def read_nodes(self):

    # 构建共 7 类节点
    drugs = [] # 药品
    foods = [] #  食物
    checks = [] # 检查
    departments = [] #科室
    producers = [] #药品大类
    diseases = [] #疾病
    symptoms = []#症状
    disease_infos = []#疾病信息
```

```
# 构建节点实体关系
        rels_department = [] #    科室－科室关系
        rels_noteat = [] #  疾病－忌吃食物关系
        rels_doeat = [] #  疾病－宜吃食物关系
        rels_recommandeat = [] #  疾病－推荐吃食物关系
        rels_commonddrug = [] #  疾病－通用药品关系
        rels_recommanddrug = [] #  疾病－热门药品关系
        rels_check = [] #  疾病－检查关系
        rels_drug_producer = [] #  厂商－药物关系

        rels_symptom = [] #疾病症状关系
        rels_acompany = [] #  疾病并发关系
        rels_category = [] #    疾病与科室之间的关系



        count = 0
        for data in open(self.data_path):
            disease_dict = {}
            count += 1
            print(count)
            data_json = json.loads(data)
            disease = data_json['name']
            disease_dict['name'] = disease
            diseases.append(disease)
            disease_dict['desc'] = ''
            disease_dict['prevent'] = ''
            disease_dict['cause'] = ''
            disease_dict['easy_get'] = ''
            disease_dict['cure_department'] = ''
            disease_dict['cure_way'] = ''
            disease_dict['cure_lasttime'] = ''
            disease_dict['symptom'] = ''
            disease_dict['cured_prob'] = ''

            if 'symptom' in data_json:
                symptoms += data_json['symptom']
                for symptom in data_json['symptom']:
                    rels_symptom.append([disease, symptom])
```

```
if 'acompany' in data_json:
    for acompany in data_json['acompany']:
        rels_acompany.append([disease, acompany])

if 'desc' in data_json:
    disease_dict['desc'] = data_json['desc']

if 'prevent' in data_json:
    disease_dict['prevent'] = data_json['prevent']

if 'cause' in data_json:
    disease_dict['cause'] = data_json['cause']

if 'get_prob' in data_json:
    disease_dict['get_prob'] = data_json['get_prob']

if 'easy_get' in data_json:
    disease_dict['easy_get'] = data_json['easy_get']

if 'cure_department' in data_json:
    cure_department = data_json['cure_department']
    if len(cure_department) == 1:
        rels_category.append([disease, cure_department[0]])
    if len(cure_department) == 2:
        big = cure_department[0]
        small = cure_department[1]
        rels_department.append([small, big])
        rels_category.append([disease, small])

    disease_dict['cure_department'] = cure_department
    departments += cure_department
if 'cure_way' in data_json:
    disease_dict['cure_way'] = data_json['cure_way']

if  'cure_lasttime' in data_json:
    disease_dict['cure_lasttime'] = data_json['cure_lasttime']

if 'cured_prob' in data_json:
    disease_dict['cured_prob'] = data_json['cured_prob']
```

```
if 'common_drug' in data_json:
                        common_drug = data_json['common_drug']
                        for drug in common_drug:
                            rels_commonddrug.append([disease, drug])
                        drugs += common_drug

                    if 'recommand_drug' in data_json:
                        recommand_drug = data_json['recommand_drug']
                        drugs += recommand_drug
                        for drug in recommand_drug:
                            rels_recommanddrug.append([disease, drug])

                    if 'not_eat' in data_json:
                        not_eat = data_json['not_eat']
                        for _not in not_eat:
                            rels_noteat.append([disease, _not])

                        foods += not_eat
                        do_eat = data_json['do_eat']
                        for _do in do_eat:
                            rels_doeat.append([disease, _do])

                        foods += do_eat
                        recommand_eat = data_json['recommand_eat']

                        for _recommand in recommand_eat:
                            rels_recommandeat.append([disease, _recommand])
                        foods += recommand_eat

                    if 'check' in data_json:
                        check = data_json['check']
                        for _check in check:
                            rels_check.append([disease, _check])
                        checks += check
if 'drug_detail' in data_json:
                        drug_detail = data_json['drug_detail']
                        producer = [i.split('(')[0] for i in drug_detail]
                        rels_drug_producer += [[i.split('(')[0], i.split('(')[-1].replace(')', '')] for i in
drug_detail]
                        producers += producer
                    disease_infos.append(disease_dict)
```

```
return set(drugs), set(foods), set(checks), set(departments), set(producers), set(symptoms),
set(diseases), disease_infos,\
                    rels_check, rels_recommandeat, rels_noteat, rels_doeat, rels_department,
rels_commonddrug, rels_drug_producer, rels_recommanddrug,\
                    rels_symptom, rels_acompany, rels_category


    '''建立节点'''
    def create_node(self, label, nodes):
        count = 0
        for node_name in nodes:
            node = Node(label, name=node_name)
            self.g.create(node)
            count += 1
            print(count, len(nodes))
        return


    '''创建知识图谱中心疾病的节点'''
    def create_diseases_nodes(self, disease_infos):
        count = 0
        for disease_dict in disease_infos:
            node = Node("Disease", name=disease_dict['name'], desc=disease_dict['desc'],
                        prevent=disease_dict['prevent'] ,cause=disease_dict['cause'],

easy_get=disease_dict['easy_get'],cure_lasttime=disease_dict['cure_lasttime'],
                        cure_department=disease_dict['cure_department']
                        ,cure_way=disease_dict['cure_way'] ,
cured_prob=disease_dict['cured_prob'])
            self.g.create(node)
            count += 1
            print(count)
        return
```

```
'''创建知识图谱实体节点类型 schema'''
    def create_graphnodes(self):
        Drugs, Foods, Checks, Departments, Producers, Symptoms, Diseases,
disease_infos,rels_check, rels_recommandeat, rels_noteat, rels_doeat, rels_department,
rels_commonddrug, rels_drug_producer, rels_recommanddrug,rels_symptom, rels_acompany,
rels_category = self.read_nodes()
        self.create_diseases_nodes(disease_infos)
        self.create_node('Drug', Drugs)
        print(len(Drugs))
        self.create_node('Food', Foods)
        print(len(Foods))
        self.create_node('Check', Checks)
        print(len(Checks))
        self.create_node('Department', Departments)
        print(len(Departments))
        self.create_node('Producer', Producers)
        print(len(Producers))
        self.create_node('Symptom', Symptoms)
        return


    '''创建实体关系边'''
    def create_graphrels(self):
        Drugs, Foods, Checks, Departments, Producers, Symptoms, Diseases, disease_infos,
rels_check, rels_recommandeat, rels_noteat, rels_doeat, rels_department, rels_commonddrug,
rels_drug_producer, rels_recommanddrug,rels_symptom, rels_acompany, rels_category =
self.read_nodes()
        self.create_relationship('Disease', 'Food', rels_recommandeat, 'recommand_eat', '推荐食谱')
        self.create_relationship('Disease', 'Food', rels_noteat, 'no_eat', '忌吃')
        self.create_relationship('Disease', 'Food', rels_doeat, 'do_eat', '宜吃')
        self.create_relationship('Department', 'Department', rels_department, 'belongs_to', '属于')
        self.create_relationship('Disease', 'Drug', rels_commonddrug, 'common_drug', '常用药品')
        self.create_relationship('Producer', 'Drug', rels_drug_producer, 'drugs_of', '生产药品')
        self.create_relationship('Disease', 'Drug', rels_recommanddrug, 'recommand_drug', '好评药
品')
        self.create_relationship('Disease', 'Check', rels_check, 'need_check', '诊断检查')
        self.create_relationship('Disease', 'Symptom', rels_symptom, 'has_symptom', '症状')
        self.create_relationship('Disease', 'Disease', rels_acompany, 'acompany_with', '并发症')
        self.create_relationship('Disease', 'Department', rels_category, 'belongs_to', '所属科室')
```

```
'''创建实体关联边'''
    def create_relationship(self, start_node, end_node, edges, rel_type, rel_name):
        count = 0
        # 去重处理
        set_edges = []
        for edge in edges:
            set_edges.append('###'.join(edge))
        all = len(set(set_edges))
        for edge in set(set_edges):
            edge = edge.split('###')
            p = edge[0]
            q = edge[1]
            query = "match(p:%s),(q:%s) where p.name='%s'and q.name='%s' create (p)-
[rel:%s{name:'%s'}]->(q)" % (
                start_node, end_node, p, q, rel_type, rel_name)
            try:
                self.g.run(query)
                count += 1
                print(rel_type, count, all)
            except Exception as e:
                print(e)
        return
```

(2) 问题解析

```
class QuestionClassifier:
    def __init__(self):
        cur_dir = '/'.join(os.path.abspath(__file__).split('/')[:-1])
        #  特征词路径
        self.disease_path = os.path.join(cur_dir, 'dict/disease.txt')
        self.department_path = os.path.join(cur_dir, 'dict/department.txt')
        self.check_path = os.path.join(cur_dir, 'dict/check.txt')
        self.drug_path = os.path.join(cur_dir, 'dict/drug.txt')
        self.food_path = os.path.join(cur_dir, 'dict/food.txt')
        self.producer_path = os.path.join(cur_dir, 'dict/producer.txt')
        self.symptom_path = os.path.join(cur_dir, 'dict/symptom.txt')
        self.deny_path = os.path.join(cur_dir, 'dict/deny.txt')
```

```
    # 加载特征词
        self.disease_wds= [i.strip() for i in open(self.disease_path) if i.strip()]
        self.department_wds= [i.strip() for i in open(self.department_path) if i.strip()]
        self.check_wds= [i.strip() for i in open(self.check_path) if i.strip()]
        self.drug_wds= [i.strip() for i in open(self.drug_path) if i.strip()]
        self.food_wds= [i.strip() for i in open(self.food_path) if i.strip()]
        self.producer_wds= [i.strip() for i in open(self.producer_path) if i.strip()]
        self.symptom_wds= [i.strip() for i in open(self.symptom_path) if i.strip()]
        self.region_words = set(self.department_wds + self.disease_wds + self.check_wds +
self.drug_wds + self.food_wds + self.producer_wds + self.symptom_wds)
        self.deny_words = [i.strip() for i in open(self.deny_path) if i.strip()]
        # 构造领域 actree
        self.region_tree = self.build_actree(list(self.region_words))
        # 构建词典
        self.wdtype_dict = self.build_wdtype_dict()
        # 问句疑问词
        self.symptom_qwds = ['症状', '表征', '现象', '症候', '表现']
        self.cause_qwds = ['原因','成因', '为什么', '怎么会', '怎样才', '咋样才', '怎样会', '如何会', '为啥
', '为何', '如何才会', '怎么才会', '会导致', '会造成']
        self.acompany_qwds = ['并发症', '并发', '一起发生', '一并发生', '一起出现', '一并出现', '一同
发生', '一同出现', '伴随发生', '伴随', '共现']
        self.food_qwds = ['饮食', '饮用', '吃', '食', '伙食', '膳食', '喝', '菜' ,'忌口', '补品', '保健品', '食谱',
'菜谱', '食用', '食物','补品']
        self.drug_qwds = ['药', '药品', '用药', '胶囊', '口服液', '炎片']
        self.prevent_qwds = ['预防', '防范', '抵制', '抵御', '防止','躲避','逃避','避开','免得','逃开','避开
','避掉','躲开','躲掉','绕开',
                             '怎样才能不', '怎么才能不', '咋样才能不','咋才能不', '如何才能不',
                             '怎样才不', '怎么才不', '咋样才不','咋才不', '如何才不',
                             '怎样才可以不', '怎么才可以不', '咋样才可以不', '咋才可以不', '如何
可以不',
                             '怎样才可不', '怎么才可不', '咋样才可不', '咋才可不', '如何可不']
self.lasttime_qwds = ['周期', '多久', '多长时间', '多少时间', '几天', '几年', '多少天', '多少小时', '几个小
时', '多少年']
        self.cureway_qwds = ['怎么治疗', '如何医治', '怎么医治', '怎么治', '怎么医', '如何治', '医治方
式', '疗法', '咋治', '怎么办', '咋办', '咋治']
self.cureprob_qwds = ['多大概率能治好', '多大几率能治好', '治好希望大么', '几率', '几成', '比例', '可能
性', '能治', '可治', '可以治', '可以医']
        self.easyget_qwds = ['易感人群', '容易感染', '易发人群', '什么人', '哪些人', '感染', '染上', '得上
']
        self.check_qwds = ['检查', '检查项目', '查出', '检查', '测出', '试出']
```

```
self.belong_qwds = ['属于什么科', '属于', '什么科', '科室']
        self.cure_qwds = ['治疗什么', '治啥', '治疗啥', '医治啥', '治愈啥', '主治啥', '主治什么', '有什么用', '有何用', '用处', '用途',
                          '有什么好处', '有什么益处', '有何益处', '用来', '用来做啥', '用来作甚', '需要', '要']

        print('model init finished ......')

        return
'''分类主函数'''
    def classify(self, question):
        data = {}
        medical_dict = self.check_medical(question)
        if not medical_dict:
            return {}
        data['args'] = medical_dict
        #收集问句当中所涉及到的实体类型
        types = []
        for type_ in medical_dict.values():
            types += type_
        question_type = 'others'

        question_types = []

        # 症状
        if self.check_words(self.symptom_qwds, question) and ('disease' in types):
            question_type = 'disease_symptom'
            question_types.append(question_type)

        if self.check_words(self.symptom_qwds, question) and ('symptom' in types):
            question_type = 'symptom_disease'
            question_types.append(question_type)
        # 原因
        if self.check_words(self.cause_qwds, question) and ('disease' in types):
            question_type = 'disease_cause'
            question_types.append(question_type)
        # 并发症
        if self.check_words(self.acompany_qwds, question) and ('disease' in types):
            question_type = 'disease_acompany'
            question_types.append(question_type)
```

```
# 推荐食品
        if self.check_words(self.food_qwds, question) and 'disease' in types:
            deny_status = self.check_words(self.deny_words, question)
            if deny_status:
                question_type = 'disease_not_food'
            else:
                question_type = 'disease_do_food'
            question_types.append(question_type)

        #已知食物找疾病
        if self.check_words(self.food_qwds+self.cure_qwds, question) and 'food' in types:
            deny_status = self.check_words(self.deny_words, question)
            if deny_status:
                question_type = 'food_not_disease'
            else:
                question_type = 'food_do_disease'
            question_types.append(question_type)

        # 推荐药品
        if self.check_words(self.drug_qwds, question) and 'disease' in types:
            question_type = 'disease_drug'
            question_types.append(question_type)

        # 药品治啥病
        if self.check_words(self.cure_qwds, question) and 'drug' in types:
            question_type = 'drug_disease'
            question_types.append(question_type)

        # 疾病接受检查项目
        if self.check_words(self.check_qwds, question) and 'disease' in types:
            question_type = 'disease_check'
            question_types.append(question_type)
        # 已知检查项目查相应疾病
        if self.check_words(self.check_qwds+self.cure_qwds, question) and 'check' in types:
            question_type = 'check_disease'
            question_types.append(question_type)
        #  症状防御
        if self.check_words(self.prevent_qwds, question) and 'disease' in types:
            question_type = 'disease_prevent'
            question_types.append(question_type)
```

```
# 疾病医疗周期
        if self.check_words(self.lasttime_qwds, question) and 'disease' in types:
            question_type = 'disease_lasttime'
            question_types.append(question_type)

        # 疾病治疗方式
        if self.check_words(self.cureway_qwds, question) and 'disease' in types:
            question_type = 'disease_cureway'
            question_types.append(question_type)

        # 疾病治愈可能性
        if self.check_words(self.cureprob_qwds, question) and 'disease' in types:
            question_type = 'disease_cureprob'
            question_types.append(question_type)

        # 疾病易感染人群
        if self.check_words(self.easyget_qwds, question) and 'disease' in types :
            question_type = 'disease_easyget'
            question_types.append(question_type)

        # 若没有查到相关的外部查询信息，那么则将该疾病的描述信息返回
        if question_types == [] and 'disease' in types:
            question_types = ['disease_desc']

        # 若没有查到相关的外部查询信息，那么则将该疾病的描述信息返回
        if question_types == [] and 'symptom' in types:
            question_types = ['symptom_disease']

        # 将多个分类结果进行合并处理，组装成一个字典
        data['question_types'] = question_types

        return data
```

```
'''构造词对应的类型'''
    def build_wdtype_dict(self):
        wd_dict = dict()
        for wd in self.region_words:
            wd_dict[wd] = []
            if wd in self.disease_wds:
                wd_dict[wd].append('disease')
            if wd in self.department_wds:
                wd_dict[wd].append('department')
            if wd in self.check_wds:
                wd_dict[wd].append('check')
            if wd in self.drug_wds:
                wd_dict[wd].append('drug')
            if wd in self.food_wds:
                wd_dict[wd].append('food')
            if wd in self.symptom_wds:
                wd_dict[wd].append('symptom')
            if wd in self.producer_wds:
                wd_dict[wd].append('producer')
        return wd_dict
    '''构造 actree，加速过滤'''
    def build_actree(self, wordlist):
        actree = ahocorasick.Automaton()
        for index, word in enumerate(wordlist):
            actree.add_word(word, (index, word))
        actree.make_automaton()
        return actree
    '''问句过滤'''
    def check_medical(self, question):
        region_wds = []
        for i in self.region_tree.iter(question):
            wd = i[1][1]
            region_wds.append(wd)
        stop_wds = []
        for wd1 in region_wds:
            for wd2 in region_wds:
                if wd1 in wd2 and wd1 != wd2:
                    stop_wds.append(wd1)
        final_wds = [i for i in region_wds if i not in stop_wds]
        final_dict = {i:self.wdtype_dict.get(i) for i in final_wds}
        return final_dict
```

```
'''基于特征词进行分类'''
    def check_words(self, wds, sent):
        for wd in wds:
            if wd in sent:
                return True
        return False
```

(3) 回答搜索

```
class QuestionPaser:
    '''构建实体节点'''
    def build_entitydict(self, args):
        entity_dict = {}
        for arg, types in args.items():
            for type in types:
                if type not in entity_dict:
                    entity_dict[type] = [arg]
                else:
                    entity_dict[type].append(arg)

        return entity_dict
'''解析主函数'''
    def parser_main(self, res_classify):
        args = res_classify['args']
        entity_dict = self.build_entitydict(args)
        question_types = res_classify['question_types']
        sqls = []
        for question_type in question_types:
            sql_ = {}
            sql_['question_type'] = question_type
            sql = []
            if question_type == 'disease_symptom':
                sql = self.sql_transfer(question_type, entity_dict.get('disease'))

            elif question_type == 'symptom_disease':
                sql = self.sql_transfer(question_type, entity_dict.get('symptom'))
            elif question_type == 'disease_cause':
                sql = self.sql_transfer(question_type, entity_dict.get('disease'))

            elif question_type == 'disease_acompany':
                sql = self.sql_transfer(question_type, entity_dict.get('disease'))
```

```
elif question_type == 'disease_not_food':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))

                elif question_type == 'disease_do_food':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))

                elif question_type == 'food_not_disease':
                    sql = self.sql_transfer(question_type, entity_dict.get('food'))

                elif question_type == 'food_do_disease':
                    sql = self.sql_transfer(question_type, entity_dict.get('food'))

                elif question_type == 'disease_drug':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))

                elif question_type == 'drug_disease':
                    sql = self.sql_transfer(question_type, entity_dict.get('drug'))

                elif question_type == 'disease_check':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))

                elif question_type == 'check_disease':
                    sql = self.sql_transfer(question_type, entity_dict.get('check'))

                elif question_type == 'disease_prevent':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))

                elif question_type == 'disease_lasttime':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))
                elif question_type == 'disease_cureway':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))

                elif question_type == 'disease_cureprob':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))

                elif question_type == 'disease_easyget':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))

                elif question_type == 'disease_desc':
                    sql = self.sql_transfer(question_type, entity_dict.get('disease'))
```

```
                if sql:
                    sql_['sql'] = sql

                    sqls.append(sql_)

        return sqls

    '''针对不同的问题，分开进行处理'''
    def sql_transfer(self, question_type, entities):
        if not entities:
            return []

        # 查询语句
        sql = []
        # 查询疾病的原因
        if question_type == 'disease_cause':
            sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.cause".format(i) for
i in entities]

        # 查询疾病的防御措施
        elif question_type == 'disease_prevent':
            sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.prevent".format(i)
for i in entities]

        # 查询疾病的持续时间
        elif question_type == 'disease_lasttime':
            sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name,
m.cure_lasttime".format(i) for i in entities]

        # 查询疾病的治愈概率
        elif question_type == 'disease_cureprob':
            sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name,
m.cured_prob".format(i) for i in entities]

        # 查询疾病的治疗方式
        elif question_type == 'disease_cureway':
            sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name,
m.cure_way".format(i) for i in entities]
```

```
# 查询疾病的易发人群
        elif question_type == 'disease_easyget':
            sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.easy_get".format(i) for i in entities]

        # 查询疾病的相关介绍
        elif question_type == 'disease_desc':
            sql = ["MATCH (m:Disease) where m.name = '{0}' return m.name, m.desc".format(i) for i in entities]

        # 查询疾病有哪些症状
        elif question_type == 'disease_symptom':
            sql = ["MATCH (m:Disease)-[r:has_symptom]->(n:Symptom) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]

        # 查询症状会导致哪些疾病
        elif question_type == 'symptom_disease':
            sql = ["MATCH (m:Disease)-[r:has_symptom]->(n:Symptom) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]

        # 查询疾病的并发症
        elif question_type == 'disease_acompany':
            sql1 = ["MATCH (m:Disease)-[r:acompany_with]->(n:Disease) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]
            sql2 = ["MATCH (m:Disease)-[r:acompany_with]->(n:Disease) where n.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]
            sql = sql1 + sql2
        # 查询疾病的忌口
        elif question_type == 'disease_not_food':
            sql = ["MATCH (m:Disease)-[r:no_eat]->(n:Food) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]

        # 查询疾病建议吃的东西
        elif question_type == 'disease_do_food':
            sql1 = ["MATCH (m:Disease)-[r:do_eat]->(n:Food) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]
            sql2 = ["MATCH (m:Disease)-[r:recommand_eat]->(n:Food) where m.name = '{0}' return m.name, r.name, n.name".format(i) for i in entities]
            sql = sql1 + sql2
```

```
# 已知忌口查疾病
        elif question_type == 'food_not_disease':
            sql = ["MATCH (m:Disease)-[r:no_eat]->(n:Food) where n.name = '{0}' return m.name,
r.name, n.name".format(i) for i in entities]

        # 已知推荐查疾病
        elif question_type == 'food_do_disease':
            sql1 = ["MATCH (m:Disease)-[r:do_eat]->(n:Food) where n.name = '{0}' return m.name,
r.name, n.name".format(i) for i in entities]
            sql2 = ["MATCH (m:Disease)-[r:recommand_eat]->(n:Food) where n.name = '{0}' return
m.name, r.name, n.name".format(i) for i in entities]
            sql = sql1 + sql2

        # 查询疾病常用药品－药品别名记得扩充
        elif question_type == 'disease_drug':
            sql1 = ["MATCH (m:Disease)-[r:common_drug]->(n:Drug) where m.name = '{0}' return
m.name, r.name, n.name".format(i) for i in entities]
            sql2 = ["MATCH (m:Disease)-[r:recommand_drug]->(n:Drug) where m.name = '{0}'
return m.name, r.name, n.name".format(i) for i in entities]
            sql = sql1 + sql2

        # 已知药品查询能够治疗的疾病
        elif question_type == 'drug_disease':
            sql1 = ["MATCH (m:Disease)-[r:common_drug]->(n:Drug) where n.name = '{0}' return
m.name, r.name, n.name".format(i) for i in entities]
            sql2 = ["MATCH (m:Disease)-[r:recommand_drug]->(n:Drug) where n.name = '{0}'
return m.name, r.name, n.name".format(i) for i in entities]
            sql = sql1 + sql2
        # 查询疾病应该进行的检查
        elif question_type == 'disease_check':
            sql = ["MATCH (m:Disease)-[r:need_check]->(n:Check) where m.name = '{0}' return
m.name, r.name, n.name".format(i) for i in entities]

        # 已知检查查询疾病
        elif question_type == 'check_disease':
            sql = ["MATCH (m:Disease)-[r:need_check]->(n:Check) where n.name = '{0}' return
m.name, r.name, n.name".format(i) for i in entities]

        return sql
```

2.3 专家系统

专家系统是一个智能计算机程序系统，其内部含有大量的某个领域专家水平的知识与经验，能够利用人类专家的知识和解决问题的方法来处理该领域问题。也就是说，专家系统是一个具有大量的专门知识与经验的程序系统，它应用人工智能技术和计算机技术，根据某领域一个或多个专家提供的知识和经验，进行推理和判断，模拟人类专家的决策过程，以解决那些需要人类专家处理的复杂问题。简而言之，专家系统是一种模拟人类专家解决领域问题的计算机程序系统。

2.3.1 专家系统概述

一、定义

对于专家系统存在各种不同的定义。

定义 1 专家系统是一个智能计算机程序系统，其内部含有大量的某个领域专家水平的知识与经验，能够利用人类专家的知识和解决问题的方法来处理该领域问题。也就是说，专家系统是一个具有大量的专门知识与经验的程序系统，它应用人工智能技术和计算机技术，根据某领域一个或多个专家提供的知识和经验，进行推理和判断，模拟人类专家的决策过程，以便解决那些需要人类专家处理的复杂问题。简而言之，专家系统是一种模拟人类专家解决领域问题的计算机程序系统。

此外，还有其他一些关于专家系统的定义。这里首先给出专家系统技术先行者和开拓者、美国斯坦福大学教授费根鲍姆 1982 年对人工智能的定义。为便于读者准确理解该定义的原意，下面用英文原文给出：

定义 2 Expert system is“an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solutions.”That is， an expert system is a computer system that emulates the decision-making ability of a human expert.The term emulate means that the expert system is intended to actin all respects like a human expert.

下面是韦斯(Weiss) 和库利柯夫斯基(Kulikowski) 对专家系统的界定。

定义 3 专家系统使用人类专家推理的计算机模型来处理现实世界中需要专家做出解释的复杂问题，并得出与专家相同的结论。

二、专家系统的特点

在总体上，专家系统具有一些共同的特点和优点。

专家系统具有下列 3 个特点：

(1)启发性。专家系统能运用专家的知识与经验进行推理、判断和决策。世界上的大部分工作和知识都是非数学性的，只有一小部分人类活动是以数学公式或数字计算为核心的(约占 8%)。即使是化学和物理学科，大部分也是靠推理进行思考的；对于生物学、大部分医学和全部法律，情况也是这样。企业管理的思考几乎全靠符号推理，而不是数值计算。

(2)透明性。专家系统能够解释本身的推理过程和回答用户提出的问题，以便让用户了解推理过程，提高对专家系统的信赖感。例如，一个医疗诊断专家系统诊断某病人患有肺炎，而且必须用某种抗生素治疗，那么，这一专家系统将会向病人解释为什么他患有肺炎，而且必须用某种抗生素治疗，就像一位医疗专家对病人详细解释病情和治疗方案一样。

(3)灵活性。专家系统能不断地增长知识，修改原有知识，不断更新。由于这一特点，使得专家系统具有十分广泛的应用领域。

三、专家系统的优点

近 20 年来，专家系统获得迅速发展，应用领域越来越广，解决实际问题的能力越来越

强，这是专家系统的优良性能以及对国民经济的重大作用决定的。具体地说，专家系统的

优点包括下列几个方面：

(1)专家系统能够高效率、准确、周到、迅速和不知疲倦地进行工作。

(2)专家系统解决实际问题时不受周围环境的影响，也不可能遗漏忘记。

(3)可以使专家的专长不受时间和空间的限制，以便推广珍贵和稀缺的专家知识与经验。

(4)专家系统能促进各领域的发展，它使各领域专家的专业知识和经验得到总结和精练，能够广泛有力地传播专家的知识、经验和能力。

(5)专家系统能汇集和集成多领域专家的知识和经验以及他们协作解决重大问题的能力，它拥有更渊博的知识、更丰富的经验和更强的工作能力。

(6)军事专家系统的水平是一个国家国防现代化和国防能力的重要标志之一。

(7)专家系统的研制和应用，具有巨大的经济效益和社会效益。

(8)研究专家系统能够促进整个科学技术的发展。专家系统对人工智能的各个领域的发展起了很大的促进作用，并将对科技、经济、国防、教育、社会和人民生活产生极其深远的影响。

2.3.2 专家系统的一般结构

专家系统的结构是指专家系统色缉成部分的构造方法租组织形式。系统结构选择恰当与否，是与专家系统的适用性和有效性密切相关的。选则什么结构最为恰当，要根据系统的应用环境和所执行的任务而定。例如，MYCIN 系统的任务是疾病诊断与解释，其问题的特点是需要较小的可能空间、可靠的数据及比较可靠的知识，这就决定了它可采用穷尽检索解空间和单链推理等较简单的控制方法和系统结构。与此不同的，HEARSAY- II 系统的任务是进行口语理解。这一任务需要检索巨大的可能解空间，数据和知识都不可靠，缺少问题的比较固定的路线，经常需要猜测才能继续推理等。这些特点 决定了 HEARSAY- H 必须采用比 MYCIN 更为复杂的系统结构。

图 2.2.9 表示专家系统的简化结构图，图 2.2.10 则为理想专家系统的结构图。由于每个专家系统所需要完成的任务和特点不相同，其系统结构也不尽相同，一般只具有图中部分模块。

图 2.2.9 专家系统简化结构图

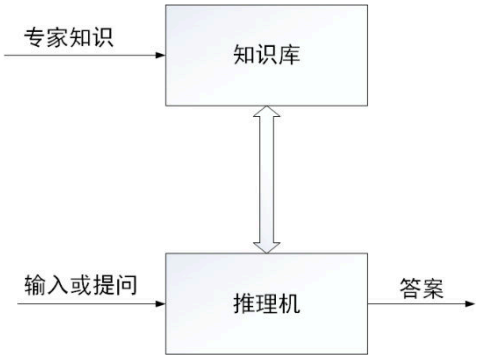
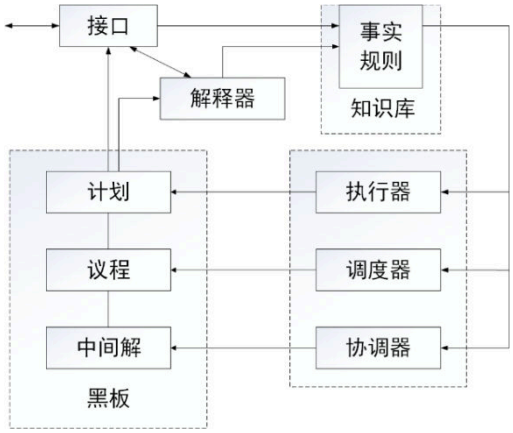


图 2.2.10 理想专家系统结构图



接口是人与系统进行信息交流的媒介，它为用户提供了直观方便的交互作用手段。 接口的功能是识别与解释用户向系统提供的命令、问题和数据等信息，并把这些信息转化为系统的内部表现形式。另一方面，接口也将系统向用户提出的问题、得出的结果和作出的解释以用户易于理解的形式提供给用户。

黑板是用来记录系统推理过程中用到的控制信息、中间假设和中间结果的数据库。它包括计划、议程和中间解 3 部分。计划记录了当前问题总的处理计划、目标、问题的当前状态和问题背景。议程记录了一些待执行的动作，这些动作大多是由黑板中已有结果与知识库中的规则作用而得到的。中间解区域中存放当前系统已产生的结果和候选假设。

知识库包括两部分内容。一部分是已知的同当前问题有关的数据信息；另一部分是 进行推理时要用到的一般知识和领域知识。这些知识大多以规则、网络和过程等形式表示。

调度器按照系统解释器的功能是向用户解释系统的行为，包括解释结论的正确性及系统输出其他候选解的原建造者所给的控制知识(通常使用优先权办法)，从议程中选择一个项作为系统下一步要执行的动作。执行器应用知识库中的及黑板中记录的信息，执行调度器所选定的动作。协调器的主要作用就是当得到新数据或新假设时，对已得到的结果进行修正，以保持结果前后的一致性。

解释器的功能是向用户解释系统的行为，包括解释结论的正确性及系统输出其他候选解的原因。为完成这一功能，通常需要利用黑板中记录的中间结果、中间假设和知识库中的知识。

前已指出，专家系统是一种智能计算机程序系统。那么，专家系统程序与常规的应用 程序之间有何不同呢？

一般应用程序与专家系统的区别在于:前者把问题求解的知识隐含地编入程序，而后者则把其应用领域的问题求解知识单独组成一个实体，即为知识库。知识库的处理是 通过与知识库分开的控制策略进行的。更明确地说，一般应用程序把知识组织为两级：数据级和程序级；大多数专家系统则将知识组织成三级：数据、知识库和控制。

在数据级上，是已经解决了的特定问题的说明性知识以及需要求解问题的有关事件的当前状态。在知识库级是专家系统的专门知识与经验。是否拥有大量知识是专家系统 成功与否的关键，因而知识表示就成为设计专家系统的关键。在控制程序级，根据既定的控制策略和所求解问题的性质来决定应用知识库中的哪些知识，这里的控制策略是指推理方式。按照是否需要概率信息来决定采用非精确推理或精确推理。推理方式还取决于所需搜索的程度。

下面把专家系统的主要组成部分归纳于下。

(1)知识库(knowledge base)

知识库用于存储某领域专家系统的专门知识，包括事实、可行操作与规则等。为了建立知识库，要解决知识获取和知识表示问题。知识获取涉及知识工程师(konwledge engineer)如何从专家那里获得专门知识的问题；知识表示则要解决如何用计算机能够理解的形式表达和存储知识的问题。

(2)综合数据库(global database)

综合数据库又称全局数据库或总数据库，它用于存储领域或问题的初始数据和推理过程中得到的中间数据(信息),即被处理对象的一些当前事实。

(3)推理机(reasoning machine)

推理机用于记忆所采用的规则和控制策略的程序，使整个专家系统能够以逻辑方式协调地工作。推理机能够根据知识进行推理和导出结论，而不是简单地搜索现成的答案。

(4)解释器(explanator)

解释器能够向用户解释专家系统的行为,包括解释推理结论的正确性以及系统输出其他候选解的原因。

(5)接口 (interface)

接口又称界面工它能够使系统与用户进行对话，使用户能够输入必要的数据、提出问题和了解推理过程及推理结果等。系统则通过接口，要求用户回答提问，并回答用户提出的问题，进行必要的解释。

2.3.3 专家系统的类型

根据专家系统的工作机制可以分为基于规则的专家系统、基于框架的专家系统、基于模型的专家系统和基于 Web 的专家系统。

1.基于规则专家系统的工作模型

产生式系统的思想比较简单，然而却十分有效。产生式系统是专家系统的基础，专家系统就是从产生式系统发展而成的。基于规则的专家系统（rule-based expert system）是一个计算机程序，该程序使用一套包含在知识库内的规则对工作存储器内的具体问题信息（事实）进行处理，通过推理机推断出新的信息。其工作模型如图 2.2.11 所示。

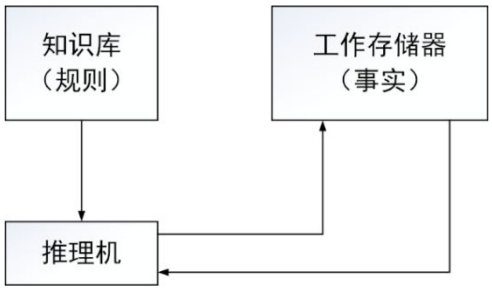


图 2.2.11　基于规则的工作模型

从图 2.2.11 可见，一个基于规则的专家系统采用下列模块来建立产生式系统的模型：

（1）知识库。以一套规则建立人的长期存储器模型。

（2）工作存储器。建立人的短期存储器模型，存放问题事实和由规则激发而推断出的新事实。

（3）推理机。借助于把存放在工作存储器内的问题事实和存放在知识库内的规则结合起来，建立人的推理模型，以推断出新的信息。推理机作为产生式系统模型的推理模块，并把事实与规则的先决条件（前项）进行比较，看看哪条规则能够被激活。通过这些激活规则，推理机把结论加进工作存储器，并进行处理，直到再没有其他规则的先决条件能与工作存储器内的事实相匹配为止。

基于规则的专家系统不需要一个人类问题求解的精确匹配，而能够通过计算机提供一个复制问题求解的合理模型。

2.基于框架的专家系统

基于框架的专家系统(frame-based expert system)就是建立在框架的基础之上的。一般概念存放在框架内，而该概念的一些特例则被表示在其他框架内并含有实际的特征值。基于框架的专家系统采用了面向目标编程技术，以提高系统的能力和灵活性。现在，基于框架的设计和面向目标的编程共享许多特征，以至于应用“目标”和“框架”这两个术语时,往往引起某些混淆。

面向目标编程涉及其所有数据结构均以目标形式出现。每个目标含有两种基本信息，即描述目标的信息和说明目标能够做些什么的信息。应用专家系统的术语来说，每个目标具有陈述知识和过程知识。面向目标编程为表示实际世界目标提供了一种自然的方法。我们观察的世界，一般都是由物体组成的，如小车、鲜花和蜜蜂等。

在设计基于框架系统时，专家系统的设计者们把目标叫做框架。现在，从事专家系统开发研究和应用者，已同时使用这两个术语而不产生混淆。

3.基于神经网络的专家系统

神经网络模型从知识表示、推理机制到控制方式，都与目前专家系统中的基于逻辑的心理模型有本质的区别。知识从显式表示变为隐式表示，这种知识不是通过人的加工转换成规则，而是通过学习算法自动获取的。推理机制从检索和验证过程变为网络上隐含模式对输入的竞争。这种竞争是并行的和针对特定特征的，并把特定论域输入模式中的各个抽象概念转化为神经网络的输入数据，以及根据论域特点适当地解释神经网络的输出数据。

如何将神经网络模型与基于逻辑的心理模型相结合是值得进一步研究的课题。从人类求解问题来看，知识存储与低层信息处理是并行分布的，而高层信息处理则是顺序的。 演绎与归纳是不可少的逻辑推理，两者结合起来能够更好地表现人类的智能行为。从综合两种模型的专家系统的设计来看，知识库由一些知识元构成，知识元可为一神经网络模块，也可以是一组规则或框架的逻辑模块。只要对神经网络的输入转换规则和输出解释 规则给予形式化表达，使之与外界接口及系统所用的知识表达结构相似，则传统的推理机 制和调度机制都可以直接应用到专家系统中去,神经网络与传统专家系统相集成，协同工作，优势互补。根据侧重点不同，其集成有三种模式：

神经网络支持专家系统。以传统的专家系统为主，以神经网络的有关技术为辅。例如对专家提供的知识和样例，通过神经网络自动获取知识。又如运用神经网络的并行推理技术以提高推理效率。

专家系统支持神经网络。以神经网络的有关技术为核心，建立相应领域的专家系统，采用专家系统的相关技术完成解释等方面的工作。

协同式的神经网络专家系统。针对大的复杂问题，将其分解为若干子问题，针对每个子问题的特点，选择用神经网络或专家系统加以实现，在神经网络和专家系统之间建立一种耦合关系。

4.基于 web 的专家系统

随着互联网技术的发展，Web 逐步成为大多数软件用户的交互接口，软件逐步走向网络化，体现为 Web 服务。专家系统的发展也离不开这个趋势，专家系统的用户界面已逐步向 Web 靠拢，专家系统的知识库和推理机也都逐步与 Web 接口交互起来。Web 已成为专家系统一个新的重要特征。

基于 Web 的专家系统是集成传统专家系统和 Web 数据交互的新型技术。这种组合技术可简化复杂决策分析方法的应用，通过内部网将解决方案递送到工作人员手中，或通过 Web 将解决方案递送到客户和供应商手中。

传统的专家系统主要面向人与单机进行交互，最多通过客户端/服务器网络结构在局域网内进行交互。基于 Web 的专家系统将人机交互定位在 Internet 层次，专家、知识工程师和普通用户通过浏览器可访问专家系统应用服务器，将问题传递给 Web 推理机，然后 Web 推理机通过后台数据库服务器对数据库和知识库进行存取，来推导出一些结论，然后将这些结论告诉用户。基于 Web 专家系统的简单结构如图 2.2.12 所示，主要分为 3 个层次：浏览器、应用逻辑层和数据库层，这种结构符合 3 层网络结构。

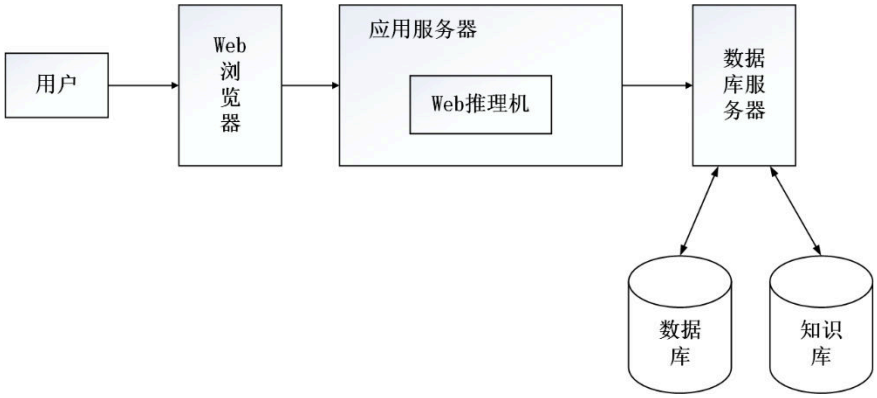


图 2.2.12　基于 Web 专家系统的结构

2.3.4　推理方法

按照推理过程所用知识的确定性，推理可分为确定性推理和不确定性推理。确定性推理方法在推理时所用的知识与证据都是确定的，推出的结论也是确定的，其真值或者为真或者为假。而不确定性推理在推理时所用的知识与证据不都是确定的，推出的结论也是不确定的。自然演绎推理和归结推理是经典的确定性推理，它们以数理逻辑的有关理论、方法和技术为理论基础，是机械化的、可在计算机上加以实现的推理方法。

本章首先介绍了专家系统中的推理机利用知识库进行推理的实现。其推理方法会根据知识表示方法的不同而有所不同。在专家系统中，规则是最常用的知识表示方法，也就是确定性推理方法，本章节主要以规则为例进行说明。接着对非确定性推理进行介绍。

2.3.4.1　确定性推理方法

按照推理的方向，推理方法可以分为正向推理和逆向推理。正向推理就是正向地使用规则，从已知条件出发向目标进行推理。其基本思想是：检验是否有规则的前提被动态数据库中的已知事实满足，如果被满足，则将该规则的结论放人动态数据库中，再检查其他的规则是否有前提被满足；反复该过程，直到目标被某个规则推出结束，或者再也没有新结论被推出为止。由于这种推理方法是从规则的前提向结论进行推理，所以称为正向推理。由于正向推理是通过动态数据库中的数据来“触发”规则进行推理的，所以又称为数据驱动的推理。

例 2.3.1 设有规则：

r_1: IF A and B THEN C

r_2: IF C and D THEN E

r_3: IF E THEN F

并且已知 A、B、D 成立，求证 F 成立。

初始时 A、B、D 在动态数据库中，根据规则r_1，推出 C 成立，所以将 C 加入动态数据库中；根据规则r_2，推出 E 成立，将 E 加入动态数据库中；根据r_3，推出 F 成立，将 F 加入动态数据库中。由于 F 是求证的目标，结果成立，推理结束。

如果在推理过程中，有多个规则的前提同时成立，如何选择一条规则呢？这就是冲突消解问题。最简单的办法是按照规则的自然顺序，选择第一个前提被满足的规则执行。也可以对多个规则进行评估，哪条规则前提被满足的条件多，哪条规则优先执行；或者从规则的结论距离要推导的结论的远近来考虑。

逆向推理又被称为反向推理，是逆向地使用规则，先将目标作为假设，查看是否有某条规则支持该假设，即规则的结论与假设是否一致，然后看结论与假设一致的规则其前提是否成立。如果前提成立（在动态数据库中进行匹配），则假设被验证，结论放人动态数据库中；否则将该规则的前提加人假设集中，一个一个地验证这些假设，直到目标假设被验证为止。由于逆向推理是从假设求解目标成立、逆向使用规则进行推理的，所以又称为目标驱动的推理。

例 2.3.2 在**例 2.3.1** 中，如何使用逆向推理推导出 F 成立？

首先将 F 作为假设，发现规则r_3的结论可以推导出 F，然后检验r_3的前提 E 是否成立。目前动态数据库中还没有记录 E 是否成立，由于规则r_2的结论可以推出 E，依次检验r_2的前提 C 和 D 是否成立。首先检验 C，由于 C 也没有在动态数据库中，再次找结论含有 C 的规则，找到规则r_1，发现其前提 A、B 均成立（在动态数据库中），从而推出 C 成立，将 C 放人动态数据库中。再检验规则r_2的另一个前提条件 D，由于 D 在动态数据库中，所以 D 成立，从而r_2的前提全部被满足，推出 E 成立，并将 E 放人动态数据库中。由于 E 已经被推出成立，所以规则r_3的前提也成立了，从而最终推出目标 F 成立。

在逆向推理中也存在冲突消解问题，可采用与正向推理一样的方法解决。

一般的逻辑推理都是确定性的，也就是说前提成立，结论一定成立。比如在几何定理证明中，如果两个同位角相等，则两条直线一定是平行的。但是在很多实际问题中，推理往往具有模糊性、不确定性。比如“如果阴天则可能下雨”，但我们都知道阴天了不一定就会下雨，这就属于非确定性推理问题。

2.3.4.2 非确定性推理方法

在前面确定性推理方法的例子中，每个规则都是确定性的，也就是说满足了什么条件，结果就一定是什么。用户给出的事实也是确定性的，有羽毛就是有羽毛，会游泳就是会游泳。但现实生活中的很多实际问题是非确定性问题。比如：如果阴天则下雨。阴天就是一个非确定性的东西，是有些云彩就算阴天呢？还是乌云滚滚算阴天？即便是乌云滚滚也不确定就一定下雨，只是天阴得越厉害，下雨的可能性就越大，但不能说阴天就一定下雨。这就是非确定性的问题，需要非确定性推理方法。

随机性、模糊性和不完全性均可导致非确定性。解决非确定性推理问题至少要解决以下几个问题：

①规则的表示

②逻辑运算

③规则运算

④规则的合成

目前有不少非确定性推理方法，各有优缺点，下面我们以著名的专家系统 MYCIN 中使用的可信度方法（certainty factor，CF 方法）为例进行说明。

一、事实的表示

事实 A 为真的可信度用 CF（A）表示，取值范围为[-1,1]，当 CF（A）=1 时，表示 A 肯定为真；当 CF（A）=-1 时，表示 A 为真的可信度为-1，也就是 A 肯定为假。CF（A）>0 表示 A 以一定的可信度为真；CF（A）<0 表示 A 以一定的可信度(-CF（A））为假，或者说 A 为真的可信度为 CF（A），由于此时 CF（A）为负，实际上 A 为假；CF（A）=0 表示对 A 一无所知。在实际使用时，一般会给出一个绝对值比较

小的区间，只要在这个区间就表示对 A 一无所知，这个区间一般取[-0.2,0.2]CF(A)。

例如：

CF（阴天）=0.7，表示阴天的可信度为 0.7。

CF（阴天）=-0.7，表示阴天的可信度为-0.7，也就是晴天的可信度为 0.7。

二、规则的表示

具有可信度的规则表示为如下形式：

$$IF \quad A \quad THEN \quad B \quad CF(B,A)$$

其中，A 是规则的前提；B 是规则的结论；CF（B，A）是规则的可信度，又称规则的强度，表示当前提 A 为真时，结论 B 为真的可信度。同样，规则的可信度 CF（B，A）取值范围也是[-1,1]，取值大于 0 表示规则的前提和结论是正相关的，取值小于 0 表示规则的前提和结论是负相关的，即前提越是成立则结论越不成立。

一条规则的可信度可以理解为当前提肯定为真时，结论为真的可信度。

例如：IF 阴天 THEN 下雨 0.7

表示：如果阴天，则下雨的可信度为 0.7。

例如：IF 晴天 THEN 下雨 -0.7

表示：如果晴天，则下雨的可信度为-0.7，即如果晴天，则不下雨的可信度为 0.7。

若规则的可信度 CF(B,A)=0，则表示规则的前提和结论之间没有任何相关性。

例如：IF 上班 THEN 下雨 0

表示：上班和下雨之间没有任何联系。

例如：IF 阴天 and 湿度大 THEN 下雨 0.6

表示：如果阴天且湿度大，则下雨的可信度为 0.6。

三、逻辑运算

规则前提可以是复合条件，复合条件可以通过逻辑运算表示。常用的逻辑运算有“与”“或”“非”，在规则中可以分别用"and" "or" "not"表示。在可信度方法中，具有可信度的逻辑运算规则如下：

①CF(A and B)=min{CF(A),CF(B)}

②CF(A or B)=max{CF(A),CF(B)}

③CF(not A)=-CF(A)

①表示“A and B”的可信度，等于 CF(A）和 CF(B）中小的一个；②表示“A or B”的可信度，等于 CF（A）和 CF（B）中大的一个；③表示“not A”的可信度等于 A 的可信度的负值。

例如，已知：

CF（阴天）=0.7

CF（湿度大）=0.5

则：

CF（阴天 and 湿度大）=0.5

CF（阴天 or 湿度大）=0.7

CF（not 阴天）=-0.7

四、规则运算

前面提到过，规则的可信度可以理解为当规则的前提肯定为真时，结论的可信度。如果已知的事实不是肯定为真，也就是事实的可信度不是 1 时，如何从规则得到结论的可信度呢？在可信度方法中，规则运算的规则按照如下方式计算：

已知：

IF A THEN B CF(B,A)

CF(A)

则：CF(B)=max{0,CF(A)}×CF(B,A)

由于只有当规则的前提为真时，才有可能推出规则的结论，而前提为真意味着 CF(A)必须大于 0；CF（A）<0 的规则，意味着规则的前提不成立，不能从该规则推导出任何与结论 B 有关的信息。所以在可信度的规则运算中，通过 max{0,CF(A)}任筛选出前提为真的规则，并通过规则前提的可信度 CF（A）与规则的可信度 CF(B,A)相乘的方式得到规则的结论 B 的信度 CF(B)。如果一条规则的前提不是真，即 CF（A）<0，则通过该规则得到 CF（B）=0，表示该规则得不出任何与结论 B 有关的信息。注意，这里 CF（B）=0，只是表示通过该规则得不到任何与 B 有关的信息，并不表示对 B 就一定是一无所知，因为还有可能通过其他的规则推导出与 B 有关的信息。

例如，已知：

IF　阴天　THEN　下雨　0.7

CF(阴天)=0.5

则：CF(下雨)=0.5×0.7=0.35，即从该规则得到下雨的可信度为 0.35。

已知：

IF　湿度大　THEN　下雨　0.7

CF(湿度大)=-0.5

则：CF(下雨)=0，即通过该规则得不到下雨的信息。

五、规则合成

通常情况下，得到同一个结论的规则不止一条，也就是说可能会有多个规则得出同一个结论，但是从不同规则得到同一个结论的可信度可能并不相同。

例如，有以下两条规则：

IF　阴天　THEN　下雨　0.8

IF　湿度大　THEN　下雨　0.5

且已知：

CF(阴天)=0.5

CF(湿度大)=0.4

从第一条规则，可以得到：CF(下雨)=0.5×0.8=0.4

从第二条规则，可以得到：CF(下雨)=0.4×0.5=0.2

那么究竟 CF(下雨)应该是多少呢？这就是规则合成问题。

在可信度方法中，规则的合成计算如下：

设：从规则 1 得到 CF1(B)，从规则 2 得到 CF2(B)，则合成后有：

$$CF(B)=\begin{cases} CF1(B)+CF2(B)-CF1(B)\times CF2(B)，当CF1(B)、CF2(B)均大于0\ 时 \\ CF1(B)+CF2(B)+CF1(B)\times CF2(B)，当CF1(B)、CF2(B)均小于0\ 时 \\ CF1(B)+CF2(B)，其他 \end{cases}$$

这样，上面的例子合成后的结果为：

CF(下雨)=0.4+0.2-0.4×0.2=0.52

如果是三个及三个以上的规则合成，则采用两个规则先合成一个，再与第三个合成的办法，以此类推，实现多个规则的合成。

下面给出一个用可信度方法实现非确定性推理的例子。

已知：

r_1:　IF　A1　THEN　B1 CF(B1,A1)=0.8

r_2:　IF　A2　THEN　B1 CF(B1,A2)=0.5

r_3:　IF　B1 and A3　THEN　B2 CF(B2,B1 and A3)=0.8

CF(A1)=CF(A2)=CF(A3)=1

计算：CF(B1)，CF(B2)

由r_1：CF1(B1)=CF(A1)×CF(B1,A1)=1×0.8=0.8

由r_2：CF2(B1)=CF(A2)×CF(Bl,A2)=1×0.5=0.5

合成得到：CF(B1)=CF1(B1)+CF2(B1)-CFI(B1)×CF2(B2)=0.8+0.5-0.8×0.5=0.9

CF(B1 and A3)=min{CF(B1),CF(A3)}=min{0.9,1}=0.9

由r_3：CF(B2)=CF(B1 and A3)×CF(B2,B1 and A3)=0.9×0.8=0.72

答：CF(B1)=0.9，CF(B2)=0.72

2.3.5 专家系统应用实例

本小节给出一个基于 Python 的动物识别专家系统的案列，源代码来源于一条博客[1]。希望能够让读者对专家系统的构建有更进一步的了解[2]。

要求：设计一个可以识别 7 种动物的专家系统，可以根据前提推导出结论，如果只有部分前提，询问提示。

RD.txt 是规则库，一行一条，每条空格分隔，前面是前提，最后一个是结论；用 PyQt5 设计界面。

RD.txt 文件如下：

```
有毛发 哺乳动物
有奶 哺乳动物
有羽毛 鸟
会飞 下蛋 鸟
吃肉 食肉动物
有犬齿 有爪 眼盯前方 食肉动物
哺乳动物 有蹄 有蹄类动物
哺乳动物 嚼反刍动物 有蹄类动物
哺乳动物 食肉动物 黄褐色 暗斑点 金钱豹
哺乳动物 食肉动物 黄褐色 黑色条纹 虎
有蹄类动物 长脖子 长腿 暗斑点 长颈鹿
有蹄类动物 黑色条纹 斑马
鸟 长脖子 长腿 黑白二色 不飞 鸵鸟
鸟 会游泳 不飞 黑白二色 企鹅
鸟 善飞 信天翁
```

[2] https://www.cnblogs.com/lepeCoder/p/8094136.html

Python 代码如下：

```
from PyQt5 import QtCore, QtGui, QtWidgets
import sys
class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.setGeometry(100, 200, 623, 300)
        self.groupBox = QtWidgets.QGroupBox(Form)
        self.groupBox.setGeometry(QtCore.QRect(10, -20, 600, 311))
        self.groupBox.setTitle("")
        self.groupBox.setObjectName("groupBox")
        self.label = QtWidgets.QLabel(self.groupBox)
        self.label.setGeometry(QtCore.QRect(30, 40, 61, 18))
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.groupBox)
        self.label_2.setGeometry(QtCore.QRect(470, 40, 101, 18))
        self.label_2.setAlignment(QtCore.Qt.AlignCenter)
        self.label_2.setObjectName("label_2")
        self.pushButton = QtWidgets.QPushButton(self.groupBox)
        self.pushButton.setGeometry(QtCore.QRect(230, 35, 88, 27))
        self.pushButton.setObjectName("pushButton")
        self.pushButton_2 = QtWidgets.QPushButton(self.groupBox)
        self.pushButton_2.setGeometry(QtCore.QRect(475, 190, 88, 27))
        self.pushButton_2.setObjectName("pushButton_2")
```

```
        self.pushButton_2.clicked.connect(self.btn2_click)
        self.pushButton_3 = QtWidgets.QPushButton(self.groupBox)
        self.pushButton_3.setGeometry(QtCore.QRect(475, 240, 88, 27))
        self.pushButton_3.setObjectName("pushButton_3")
        self.pushButton_3.clicked.connect(QtCore.QCoreApplication.instance().quit)
        self.pushButton_4 = QtWidgets.QPushButton(self.groupBox)
        self.pushButton_4.setGeometry(475, 140, 88, 27)
        self.pushButton_4.setObjectName("pushButton_4")
        self.pushButton_4.clicked.connect(self.topological)
        self.textEdit = QtWidgets.QTextEdit(self.groupBox)
        self.textEdit.setGeometry(QtCore.QRect(20, 80, 80, 211))
        self.textEdit.setObjectName("textEdit")
        self.textEdit_2 = QtWidgets.QTextEdit(self.groupBox)
        self.textEdit_2.setGeometry(QtCore.QRect(110, 80, 331, 211))
        self.textEdit_2.setObjectName("textEdit_2")
        self.textEdit_2.setReadOnly(True)
        self.lineEdit = QtWidgets.QLineEdit(self.groupBox)
        self.lineEdit.move(460, 90)
        self.lineEdit.setReadOnly(True)
        self.pushButton.clicked.connect(self.go)
       self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def btn2_click(self):
        if self.pushButton_2.text() != "确定输入":
            self.pushButton_2.setText("确定输入")
        else:
            self.pushButton_2.setText("修改知识库")

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "动物识别专家系统"))
        self.label.setText(_translate("Form", "输入事实"))
        self.label_2.setText(_translate("Form", "显示推理结果"))
        self.pushButton.setText(_translate("Form", "进行推理"))
        self.pushButton_2.setText(_translate("Form", "修改知识库"))
        self.pushButton_3.setText(_translate("Form", "退出程序"))
        self.pushButton_4.setText(_translate("From", "整理知识库"))

    # 将知识库做拓扑排序
    def topological(self):
        Q = []
```

```
P = []
ans = ""    #  排序后的结果
for line in open('RD.txt'):
    line = line.strip('\n')
    if line == '':
        continue
    line = line.split(' ')
    Q.append(line[line.__len__() - 1])
    del (line[line.__len__() - 1])
    P.append(line)

#  计算入度
inn = []
for i in P:
    sum = 0
    for x in i:
        if Q.count(x) > 0:    #  能找到，那么
            sum += Q.count(x)
    inn.append(sum)

while (1):
    x = 0
    if inn.count(-1) == inn.__len__():
        break
    for i in inn:
        if i == 0:
            str = ' '.join(P[x])
            # print("%s %s" %(str, Q[x]))
            ans = ans + str + " " + Q[x] + "\n"    #  写入结果
            # print("%s -- %s" %(P[x],Q[x]))
            inn[x] = -1
            #  更新入度
            y = 0
            for j in P:
                if j.count(Q[x]) == 1:
                    inn[y] -= 1
                y += 1
        x += 1
print(ans)

#  将结果写入文件
fw = open('RD.txt', 'w', buffering=1)
```

```
        fw.write(ans)
        fw.flush()
        fw.close()

    # 进行推理
    def go(self, flag=True):

        # 将产生式规则放入规则库中
        # if P then Q
        # 读取产生式文件

        self.Q = []
        self.P = []
        fo = open('RD.txt', 'r', encoding='utf-8')
        for line in fo:
            line = line.strip('\n')
            if line == '':
                continue
            line = line.split(' ')
            self.Q.append(line[line.__len__() - 1])
            del (line[line.__len__() - 1])
            self.P.append(line)
        fo.close()
        print("go 按钮按下")
        self.lines = self.textEdit.toPlainText()
        self.lines = self.lines.split('\n')    # 分割成组
        self.DB = set(self.lines)
        print(self.DB)
        self.str = ""
        print(self.str)
        flag = True
        temp = ""
        for x in self.P:    # 对于每条产生式规则
            if ListInSet(x, self.DB):    # 如果所有前提条件都在规则库中
                self.DB.add(self.Q[self.P.index(x)])
                temp = self.Q[self.P.index(x)]
                flag = False    # 至少能推出一个结论
                # print("%s --> %s" %(x, self.Q[self.P.index(x)]))
                self.str += "%s --> %s\n" % (x, self.Q[self.P.index(x)])
```

```
        if flag:    #  一个结论都推不出
            print("一个结论都推不出")
            for x in self.P:    #  对于每条产生式
                if ListOneInSet(x, self.DB):    #  事实是否满足部分前提
                    flag1 = False            #  默认提问时否认前提
                    for i in x:    #  对于前提中所有元素
                        if i not in self.DB:    #  对于不满足的那部分
                            btn = s.quest("是否" + i)
                            if btn == QtWidgets.QMessageBox.Ok:
                                #  确定则增加到 textEdit
                                self.textEdit.setText(self.textEdit.toPlainText() + "\n" + i)
                                self.DB.add(i)    #  确定则增加到规则库中
                                flag1 = True       #  肯定前提
                                # self.go(self)
                    if flag1:    #  如果肯定前提，则重新推导
                        self.go()
                        return
        self.textEdit_2.setPlainText(self.str)
        print("----------------------")
        print(self.str)
        if flag:
            btn = s.alert("啥也推不出来!!!")
            # if btn == QtWidgets.QMessageBox.Ok:    #  点击确定
            #         self.textEdit.setText(self.textEdit.toPlainText() + "\n 确定")
        else:
            self.lineEdit.setText(temp)

    #  判断 list 中至少有一个在集合 set 中
    def ListOneInSet(li, se):
        for i in li:
            if i in se:
                return True
        return False

    #  判断 list 中所有元素是否都在集合 set 中
    def ListInSet(li, se):
        for i in li:
            if i not in se:
                return False
        return True
```

```
class SecondWindow(QtWidgets.QWidget):
    def __init__(self, parent=None):
        super(SecondWindow, self).__init__(parent)
        self.setWindowTitle("修改知识库")
        self.setGeometry(725, 200, 300, 300)
        self.textEdit = QtWidgets.QTextEdit(self)
        self.textEdit.setGeometry(8, 2, 284, 286)
    # 警告没有推导结果
    def alert(self, info):
        QtWidgets.QMessageBox.move(self, 200, 200)
        QtWidgets.QMessageBox.information(self, "Information", self.tr(info))
    # 询问补充事实
    def quest(self, info):
        # 如果推理为空，需要询问用户是否要添加已知条件
        QtWidgets.QMessageBox.move(self, 200, 200)
        button = QtWidgets.QMessageBox.question(self, "Question",
                                                self.tr(info),
                                                QtWidgets.QMessageBox.Ok |
                                                QtWidgets.QMessageBox.Cancel,
                                                QtWidgets.QMessageBox.Cancel)
        return button
    def handle_click(self):
        if not self.isVisible():
            # 读取文件放到多行文本框中
            str = ""
            fo = open('RD.txt', 'r', encoding='utf-8')
            for line in fo:
                line = line.strip('\n')
                if line == '':
                    continue
                str = str + line + "\n"
            fo.close()
            self.textEdit.setText(str)
            self.show()
        else:
            # 输出文本框内容
            self.str = self.textEdit.toPlainText()
            print(self.str)
            # 将文本框内容写入文件
            self.fw = open('RD.txt', 'w')
            self.fw.write(self.str)
            self.fw.close()
```

```
                self.close()    # 关闭窗口
            def handle_close(self):
                self.close()

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    widget = QtWidgets.QWidget()
    ui = Ui_Form()
    ui.setupUi(widget)
    widget.show()
    s = SecondWindow()
    ui.pushButton_2.clicked.connect(s.handle_click)
    sys.exit(app.exec_())
```

注：在 Windows 下运行可能需要将 open 函数的 encoding 设定为 gbk

课后习题：

1、用产生式表示：如果一个人发烧、呕吐、出现黄疸，那么得肝炎的可能性有 7 成。

2、使用两种以上的 XML 表示“张三选修了李老师的人工智能导论课程”这条知识。

3、在某军事专家系统中，有如下 3 条纵深进攻的规则。

R1：If 我师任务为歼灭$\{P_1, P_2, P_3\}$地域之敌，且P_3到直线P_1P_2的距离为 D

Then 纵深进攻的距离为 D；

R2：If 我师当面之敌据守在$\{P_1, P_2, P_3\}$地域，且P_3到直线P_1P_2的距离为 D

Then 纵深进攻的距离为 D；

R3：If 我师任务为歼灭 P 地狱附近之敌，且某敌方据守在$\{P_1, P_2, P_3\}$地狱，且 P 在到$\{P_1, P_2, P_3\}$地域之内，且P_3到直线P_1P_2的距离为 D

Then 纵深进攻的距离为 D；

上述 3 条规则应该如何排序系统执行才最有效？

参考文献

[1] Fabian M. Suchanek, Gerhard Weikum: Knowledge harvesting from text and Web sources. ICDE 2013: 1250-1253.

[2] Gerhard Weikum, Martin Theobald: From information to knowledge: harvesting entities and relationships from web sources. PODS 2010: 65-76.

[3] A. Singhal. Introducing the Knowledge Graph: things, not strings. Official Google Blog, May, 2012.

[4] Gallagher, Sean. (June 7, 2012). How Google and Microsoft taught search to understand the Web http://arstechnica.com/information-technology/2012/06/inside-the-architecture-of-googles- knowledge-graphand-microsofts-satori/.

[5] Omkar Deshpande, Digvijay S. Lamba, Michel Tourn, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, AnHai Doan: Building, maintaining, and using knowledge bases: a report from the trenches. SIGMOD Conference 2013: 1209-1220.

[6] AMIT S. Introducing the knowledge graph[R]. America:Official Blog of Google, 2012.

[7] Shenshouer. Neo4j[EB/OL]. [2016-05-09]. http://neo4j.com/.

[8] FlockDB Official. FlockDB[EB/OL]. [2016-05-09]. http://webscripts.softpedia.com/script/Database-Tools/FlockDB-66248.html.

[9] Graphdb Official. Graphdb[EB/OL]. [2016-05-09]. http://www.graphdb.net/.

[10] 刘峤, 李杨, 杨段宏, 等. 知识图谱构建技术综述[J]. 计算机研究与发展, 2016, 53(3): 582-600.LIU Qiao, LI yang, YANG Duan-hong, et al. Knowledgegraph construction techniques[J]. Journal of Computer Research and Development, 2016, 53(3): 582-600.

[11] DONG X, GABRILOVICH E, HEITZ G, et al. Knowledge vault: a web-scale approach to probabilistic knowledgefusion[C]//Proc of the 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. New York: ACM, 2014.

[12] BOLLACKER K, COOK R, TUFTS P. Freebase: a shared database of structured general human knowledge[C]//Proc of the 22nd AAAI Conf on Artificial Intelligence. Menlo Park, CA: AAAI, 2007: 1962-1963.

[13] 孙镇, 王惠临. 命名实体识别研究进展综述[J]. 现代图书情报技术, 2010(6): 42-47.

[14] 赵军, 刘康, 周光有, 等. 开放式文本信息抽取[J]. 中文信息学报, 2011, 25(6): 98-110.

[15] CHINCHOR N, MARSH E. Muc-7 information extraction task definition[C]//Proc of the 7th Message Understanding Conf. Philadelphia: Linguistic Data Consortium, 1998:359-367.

[16] RAU L F. Extracting company names from text[C]//Proc of the 7th IEEE Conf on Artificial Intelligence Applications.Piscataway, NJ: IEEE, 1991: 29-32.

[17] LIU Xiao-hua, ZHANG Shao-dian, WEI Fu-ru, et al. Recognizing named entities in tweets[C]//Proc of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies. Stroudsburg, PA: ACL, 2011: 359-367.

[18] LIN Yi-feng, TSAI T, CHOU Wen-chi, et al. A maximum entropy approach to biomedical named entity recognition[C]//Proc of the 4th ACM SIGKDD Workshop on Data Mining in Bioinformatics. New York: ACM, 2004.

[19] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, Chris Dyer: Neural Architectures for Named Entity Recognition. HLT-NAACL 2016: 260-270

[20] WHITELAW C, KEHLENBECK A, PETROVIC N, et al. Web-scale named entity recognition[C]//Proc of the 17th ACM Conf on Information and Knowledge Management. New York: ACM, 2008.

[21] JAIN A, PENNACCHIOTTI M. Open entity extraction from web search query logs[C]//Proc of the 23rd Int Conf on Computational Linguistics. Stroudsburg, PA: ACL, 2010 :510-518.

[22] 蔡自兴，刘丽珏等. 人工智能及其应用（第 5 版）[M]. 北京：清华大学出版社，2016.

[23] 李德毅. 人工智能导论[M]. 北京：中国科学技术出版社，2018.

[24] 朱福喜. 人工智能习题解析与实践[M]. 北京：清华大学出版社，2019.

[25] 李娟. 谓词逻辑在人工智能知识表示中的应用[J]. 数码世界， 2017, 06 :168-169.

[26] 张召霞. 面向无人驾驶车辆行为决策的知识库管理系统研究[D]. 安徽：中国科学技术大学，2020.

[27] 张鹏升. 基于产生式模型的人脸正面化研究[D]. 北京：中国人民公安大学，2020.

[28] O. Christensen and M. Hasannasab, "Frame representations via suborbits of bounded operators," 2019 13th International conference on Sampling Theory and Applications (SampTA), Bordeaux, France, 2019, pp. 1-4.

[29] Galka, A., Moontaha, S. & Siniatchkin, M. Constrained expectation maximisation algorithm for estimating ARMA models in state space representation. EURASIP J. Adv. Signal Process. 2020, 23 (2020).

[30] Anitha Florence Vinola, F., Padma, G. A probabilistic stochastic model for analysis on the epileptic syndrome using speech synthesis and state space representation. Int J Speech Technol 23, 355–360 (2020).

[31] Raisi Tousi, R., Kamyabi-Gol, R.A. & Esmaeelzadeh, F. A continuous frame representation for an abstract shearlet group. J. Pseudo-Differ. Oper. Appl. 10, 571–580 (2019).

[32] C. Shu, D. Dosyn, V. Lytvyn, V. Vysotska, A. Sachenko and S. Jun, "Building of the Predicate Recognition System for the NLP Ontology Learning Module," 2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Metz, France, 2019, pp. 802-808.