《人工智能》课程系列

命题逻辑与一阶逻辑基础*

Preliminaries on Propositional Logic and First-Order Logic

武汉纺织大学数学与计算机学院

杜小勤

2020/08/10

Contents

*本系列文档属于讲义性质，仅用于学习目的。Last updated on: October 7, 2020。

1 概述

形式逻辑 (Formal Logic) 是研究演绎推理及其规律的科学，它的主要研究内容包括：词项和命题形式的逻辑性质；思维结构与推理技术。它使用一系列规则与推理方法，帮助人们正确地思考问题和表达思想，是人们认识世界和改造世界的必要工具，也是人类认识水平发展到一定阶段后出现的思维方法。康德首先使用了“形式逻辑”这一术语。

概念、判断、推理是形式逻辑的三大基本要素。概念包括外延和内涵：外延表示概念所涉事物范围的大小；内涵表示概念的含义及性质。判断，从质上分为肯定判断和否定判断，从量上分为全称判断、特称判断和单称判断。推理是思维的最高形式——由概念形成判断，而判断构成推理。总体上而言，人的思维是由这三大要素决定的。

形式逻辑已有两千多年的历史，其发源地有三个，即古代中国、古印度和古希腊。一般地，将 19 世纪中叶以前的形式逻辑称为传统逻辑，将 19 世纪中叶以后发展起来的现代形式逻辑称为数理逻辑 (即符号逻辑)。

中国在春秋战国时期就产生了称之为“名学”、“辩学”的逻辑学说，代表学派为墨家与名家，此外还有儒家的荀子。《荀子·正名》，尤其是《墨经》集其大成，系统地研究了名、辞、说、辩等相当于词项、命题、推理与论证之类的对象，

逻辑思想十分丰富，但由于与一定的政治、道德理论掺杂在一起，未能形成独立的学科体系。有意思的是，墨家研究逻辑，是为了找到逻辑的原则，而名家为的是建立诡辩体系。墨家对于逻辑的认识集中体现在《墨经》中，该书对于逻辑已有了系统的论述。例如，它区分了充分条件与必要条件，提出“大故（充分条件），有之必然，无之必不然”与“小故（必要条件），有之不必然，无之必不然”。而名家的惠施则提出了“合同异”的诡辩原则，目的是取消概念的边界。与惠施相反，同属名家的公孙龙则提出了“离坚白”的诡辩原则，认为任何独立的概念都有且只能有单一的属性。名家提出了许多诡辩命题，例如“白马非马”、“鸡有三足”、“孤犊无母”、“连环无扣”、“白狗黑”以及“今适越而昔来”等。

显然，名家的研究方法是中国特有的——它之所以能够建立诡辩体系，主要原因在于当时的逻辑水平尚处于较低的层次，逻辑体系中还存在着大量的漏洞。或者，从另一个角度来看，这也是逻辑发展的必要阶段——名家此举也使得这些漏洞得到了充分的暴露，为后人的研究提供了垫脚石：若要发展逻辑，就必须去克服名家的诡辩命题。实际上，名家的诡辩命题中也存在着一些合理因素，其中的一些也确实击中了形式逻辑的要害，另一些命题也反映了当时人们对自然界以及人的认知过程的合理而自然的认识。例如，如果命题“天下之中央，燕之北越之南”要成立，则必须以“地球是圆的”作为前提条件。在当时天圆地方“盖天说”占主导的情况下，名家能有这样的认识，实属不易。

在欧洲，形式逻辑由古希腊的亚里士多德创建。亚里士多德建立了第一个逻辑系统，即三段论理论；其论述形式逻辑的代表作包括《形而上学》和《工具论》。继亚里士多德之后，麦加拉-斯多阿学派逻辑揭示出命题联结词的一些重要性质，发现了若干与命题联结词有关的推理形式和规律，发展了演绎逻辑。而古希腊的另一位哲学家伊壁鸠鲁则认为归纳法是唯一科学的方法。中世纪的一些逻辑学家发展和丰富了形式逻辑。到了近代，培根和约翰·缪勒则进一步发展了归纳法。

在古印度，公元前四世纪时，胜论派和正理派开创了因明学，至六世纪时陈那将其完善为新因明学。“因”指的是推理的根据与理由；“明”指的是知识与智慧。陈那的《因明正理门论》与商羯罗主的《因明人正理论》是其代表，它们对推理从形式上作了探讨，提出了“三支论式”。但是，为佛教服务的因明学，也未能撇开思维的具体内容而上升为数学形式的科学。

17 世纪末，德国哲学家莱布尼兹设想用数学方法处理传统演绎逻辑，进行思维演算，数理逻辑由此发端。19 世纪 40 年代，英国数学家布尔的逻辑代数首先使

该设想成为现实。19 世纪 70 年代，G. 康托尔创立了集合论。G. 弗雷格在 1879 年发表的《概念语言》一书中，建立了第一个一阶逻辑体系。集合论，特别是第一个一阶逻辑体系的建立，是形式逻辑的发展进入现代阶段的标志。但直到 20 世纪初，在弗雷格等人研究的基础上，罗素和怀德海的《数学原理》建立了完全的命题演算和谓词演算，才确立了数理逻辑的基础，从此产生了现代演绎逻辑。此后，现代逻辑蓬勃发展，方兴未艾，演绎部分出现了模态逻辑、多值逻辑等非经典或非标准逻辑分支群，归纳逻辑也与概率、统计等方法相结合，开拓了许多新的研究领域。

本章介绍的命题逻辑与一阶逻辑 (即谓词逻辑)，允许智能体 (Agent) 对知识进行表示，并基于知识库执行某种推理，以指导智能体自身的决策与行动。

在“约束满足问题 (CSP)”章节中，使用变量及其赋值来表示对象的状态，这使得 CSP 具有几个优点，例如问题的结构化表示、使用通用策略求解、通过识别违反约束的变量来消除无效子空间的搜索等。

在本章，我们将继续往前推进这一策略——将逻辑作为支持智能体决策的通用表示与推理方法，这使得智能体能够将接收到的新信息与知识库中的原有知识进行综合，执行推理，并获得有效的决策，以适应环境的新变化。这种智能体被称为逻辑智能体 (Logical Agent)。要想实现逻辑智能体，必须为它配备知识库 (Knowledge Base，KB)。

知识库是句子或语句的集合——每条语句使用知识表示语言 (Knowledge Representation Language) 来表达，它们是关于某个领域的断言 (Assertion)。例如，我们所熟知的一些公理 (Axiom) 就属于这样的 (知识) 语句，只不过它们是直接给定的，而不是通过推导得来的。具体地，知识库是由描述环境及智能体目标的逻辑语句组成的，即知识库包含了智能体在环境中进行决策与行动所需的一切知识。

从某种角度看，通常意义上的程序，将需要表现的行为，或需要执行的策略，或对环境的响应，以程序代码的方式将知识及其决策写入智能体的决策循环中，这被称为过程式方法 (Procedural Approach)。还有一种方法，将所需的知识，通过语法与语义形式化为知识写入到知识库中；在决策时，智能体依据新信息及知识库中的知识，执行逻辑推理，以获得决策支持，这被称为陈述性方法 (Declarative Approach)。

我们也可以把知识库理解成一个存储与检索逻辑语句的数据库，它通常包括如下 2 个主要的操作：

1. TELL

   向知识库中添加新的事实或知识。

2. ASK

   向知识库查询智能体感兴趣的事实或知识是否成立，即使这些事实或知识并不是显而易见的事实。因此，本操作通常涉及到逻辑推理 (Inference)，这是知识库的重要功能。

然而，一个成熟的智能体体系结构，应该能够将上述 2 种方法有机地集成在一个框架中，并且能够让智能体具备学习能力。这也应该是现代人工智能与机器学习系统需要重点考虑的问题。

2 命题逻辑

通常意义下，逻辑 (Logic) 指的是基于形式化知识的有效推理 (Valid Reasoning)——通过建立一套形式化方法与推理规则，使得推理过程更加有效，推理结果更加精确。此外，逻辑及其形式化，也有助于知识的简洁表示、存储与提取。逻辑学，作为一门学科，已经在哲学、计算机科学、数学等领域得到了非常广泛的研究。逻辑的正确使用，离不开形式化系统 (Formal System)。形式化逻辑系统，也已经出现在许多应用与领域中，例如程序设计、工作流程、游戏策略、电路设计与仿真、人工智能以及数据库等。

形式化系统通常包括以下几个要素：

- 定义良好的语法 (A Well-Defined Syntax)

  即具有良好格式的公式 (Well-Formed Formula, WFF)。

- 定义良好的语义 (A Well-Defined Semantics)

  即明确每条语句的含义。

- 定义良好的证明理论 (A Well-Defined Proof-Theory)

  即依据一定的推理规则，将公式从一种形式变换为另一种形式，得到有效的推理结果。

下面，给出命题 (Proposition) 的概念。所谓命题，指的是能够判断为真 (True) 或假 (False) 的陈述句。例如：

- $2+4>3$：$True$
- $2+4>8$：$False$
- 一年有 12 个月：$True$
- 北京是中国的首都：$True$
- 中国是世界上最伟大的国家：$True$
- ⋮

而类似于“$2+5$”这样的句子，不是命题[1]；“我认为，这是一部非常好看的电影”，通常情况下，也不会被当作命题[2]。

命题逻辑 (Propositional Logic) 是基于命题的逻辑系统，它包括一套形式化方法及推理规则。下面，将逐一介绍。

2.1 基本的逻辑运算符

原子命题 (Atomic Proposition) 指的是其真值 (Truth Value)[3]不依赖于任何其它命题的命题。例如，前面给出的命题就属于原子命题。常识与公理就属于原子命题。

通常情况下，在某个领域或问题中，命题表现为语句或句子的形式，用来陈述某个事实或表达一定的含义。为了方便地形式化命题及其推理过程，使用命题符号 (Propositional Symbol) 或命题变量 (Propositional Variable) 来表示命题。一般情况下，使用小写字母表示命题，例如使用 p 表示命题“$2+4>3$”，使用 q 表示命题“一年有 12 个月”等。另一些情况下，在不引起混淆的情况下，偶尔也会使用大写字母来表示。

为了构建复杂的命题 (Complex Proposition) 或复合命题 (Compound Proposition)，需要定义若干连接词 (Connective) 或逻辑运算符 (Logical Operator)：

[1]虽然在一些程序设计语言中，将其作为逻辑表达式使用时，会得到 $True$ 这样的结果。但是，在逻辑系统中，不会将它看作命题。

[2]通常，这属于某人发表的个人观点，不属于命题。

[3]真值表示命题的取值：$True$ 或 $False$。

- $\wedge$：合取 (Conjunction) 或 *and*
- $\vee$：析取 (Disjunction) 或 *or*
- $\neg$：否定 (Negation) 或 *not*
- $\rightarrow$：蕴含 (Implication) 或条件 (Conditional)，即 if-*then*
- $\leftrightarrow$：当且仅当 iff(*if-and-only-if*) 或双向条件 (Biconditional)

其中，$\neg$ 为一元 (Unary) 逻辑运算符，其余逻辑运算符均为二元 (Binary) 运算符。关于上述连接词的具体含义及真值表 (Truth Table)，后面将会给出详细的定义。下面，首先给“具有良好格式的公式 (Well-Formed Formula, WFF)”下一个定义。

定义 2.1（*WFF*）一个命题可以被表示为具有良好格式的公式(*WFF*)，如果它通过以下规则构造而成，或符合以下规则：

1. 任何原子命题为 *WFF*;
2. 如果 p 为 *WFF*，那么 $\neg p$ 也为 *WFF*;
3. 如果 p 与 q 为 *WFF*，那么由它们通过二元逻辑运算符 $\wedge$、$\vee$、$\rightarrow$ 和 $\leftrightarrow$ 构造而成的 $p \wedge q$、$p \vee q$、$p \rightarrow q$ 和 $p \leftrightarrow q$ 均为 *WFF*;
4. 除非使用上述方法构造命题，否则命题不是 *WFF*;

实际上，上述 WFF 定义也给出了构造复合命题的方式。下面，详细论述逻辑运算符及其相关概念，它们是命题逻辑的理论基础，为命题逻辑的推理功能提供了有力支持。

首先，讨论否定“$\neg$”运算符，它为一元运算符，仅需一个参数：如果 p 表示 (任意) 命题，那么其否定命题可表示为 $\neg p$。其真值表为：

p	$\neg p$
F	T
T	F

对于具体的命题而言，其含义非常明确。例如，如果 p 表示“The apple is red”，则 $\neg p$ 表示“The apple is not red”。显然，依据真值表，可以得到 $\neg(\neg p) \equiv p$，

其中 $\equiv$ 表示左右 2 个命题具有逻辑等价性 (Logical Equivalence)。关于逻辑等价的定义，后面将会给出。

下面，讨论合取“$\wedge$”运算符，它为二元运算符，需要 2 个参数：如果 p 和 q 表示 2 个命题，那么合取可表示为 $p \wedge q$ 或 $q \wedge p$。其真值表为：

p	q	$p \wedge q$
F	F	F
F	T	F
T	F	F
T	T	T

很容易验证，合取具有可交换性 (Commutative)，即 $p \wedge q \equiv q \wedge p$。从真值表可以看出，只有当 p 和 q 均为 T 时，其合取结果才为 T。对于具体的命题，合取的含义也是非常明确的。例如，如果 p 表示“小明同学的数学成绩好”，q 表示“小明同学的语文成绩好”，则 $p \wedge q$ 表示“小明同学的数学且语文成绩好”这一含义[4]。

“$\vee$”为析取运算符，属于二元运算符，需要两个参数，析取式可表示为 $p \vee q$ 或 $q \vee p$，也服从交换律，即 $p \vee q \equiv q \vee p$。其真值表为：

p	q	$p \vee q$
F	F	F
F	T	T
T	F	T
T	T	T

如果 p 与 q 仍然分别表示上述 2 个具体命题，那么 $p \vee q$ 表示“小明同学的数学或语文成绩好”这一含义。从真值表可以看出，只要 p 或 q 中有一个取值为 T，析取结果就为 T。

“$\rightarrow$”为条件 *if-then* 运算符，属于二元运算符，需要两个参数，条件式可表

[4]对照真值表，依据小明的真实成绩，分别对命题 p 与 q 进行赋值：T 或 F，那么就能够知道 $p \wedge q$，即命题“小明同学的数学且语文成绩好”这一断言或事实是否成立。实际上，这就是一种简单的推理。

示为 $p \to q$，其字面含义为“如果 p，则 q”。其中，p 被称为前提，q 被称为结论。一般情况下，$p \to q$ 不服从交换律，即 $p \to q \not\equiv q \to p$。

需要注意的是，$p \to q$ 属于条件命题公式[5]，它会返回真值：$True$ 或 $False$——当 p 为 $True$ 时，如果 q 为 $True$，则 $p \to q$ 为 $True$，否则 $p \to q$ 为 $False$；其余情况下，$p \to q$ 均为 $True$[6]。其真值表如下：

p	q	$p \to q$
F	F	T
F	T	T
T	F	F
T	T	T

乍看起来，$p \to q$ 的这种取值规则或规定有些怪异，不太符合直觉与习惯。然而，如果清楚下面 2 点，或许就能明白 $p \to q$ 真正想要表达的含义：

1. $p \to q$ 并不强制要求 p 与 q 之间需要存在相关性或因果关系

   例如，令 p 表示“$3+3>2$”，令 q 表示“北京是中国的首都”，那么 $p \to q$ 仍然会返回 $True$，尽管 p 与 q 是毫无关联的 2 个命题。

2. $p \to q$ 命题强调的是“p 成立时 q 一定要成立”这一关系。如果违背这一关系，即“p 成立时 q 不成立”，则 $p \to q$ 返回 $False$，就不足为奇了。至于“p 不成立时，q 成立与否”这一关系，并不是该运算符所关心的。

在推理时，大多使用 $p \to q$ 的等价形式 $\neg p \vee q$，即 $p \to q \equiv \neg p \vee q$。可直接依据真值表得到该结论。

实际上，对于 $p \to q$ 所表达的数学含义，我们并不感到陌生。如果 $p \to q$ 所表达的数学关系或定理成立，那么 p 被称为 q 的充分条件 (Sufficient Condition)，而 q 被称为 p 的必要条件 (Necessary Condition)。显然，如果 $p \to q$ 且 $q \to p$ 同时成立，那么 p 和 q 互为充分必要条件。实际上，这种关系正是运算符 $\leftrightarrow (iff$ 或双条件)

[5]注意，不要与程序设计语言中的 *if-then* 选择结构混淆。在程序设计与语言中，*if-then* 表示程序执行流程的二选一结构。

[6]后一种情况下取得的 $True$ 被称为 Vacuously True 或 True by Default。

所要表达的。正式地，将 p 与 q 的双向条件关系记为 $p \leftrightarrow q \triangleq (p \rightarrow q) \wedge (q \rightarrow p)$[7]。其真值表为：

p	q	$p \rightarrow q$	$q \rightarrow p$	$p \leftrightarrow q$
F	F	T	T	T
F	T	T	F	F
T	F	F	T	F
T	T	T	T	T

可以看出，只有当 p 与 q 同时取相同的值时，$p \leftrightarrow q$ 的取值才为 $True$。

上面讨论了 5 种逻辑运算符，再加上括号“()”，它们的优先级从高到低依次为：()、¬、∧∨、→、↔。对于同一优先级的运算符，如果没有添加括号，则按从左到右的顺序进行。

2.2 摩根定律

下面直接给出摩根定律 (De Morgan's Law) 或摩根公式，它们的正确性可直接通过真值表得到验证：

- $\neg (p \wedge q) \equiv \neg p \vee \neg q$
- $\neg (p \vee q) \equiv \neg p \wedge \neg q$

2.3 恒真、恒假、逻辑等价及命题的相容性

在一个命题中，对原子命题的所有可能取值，如果命题总是取值为 $True$，那么称该命题为恒真式、永真式或重言式 (Tautology)。反之，对原子命题的所有可能取值，如果命题总是取值为 $False$，那么称该命题为恒假式、永假式或矛盾式 (Contradiction)。如果一个命题，既不是恒真式，也不是恒假式，则称该命题为偶然式 (Contingent)。

例如，$p \vee \neg p$ 是一个恒真式，$p \wedge \neg p$ 是一个恒假式或矛盾式，而 $(p \wedge q) \leftrightarrow (p \vee q)$ 是一个偶然式，其真值表为：

[7] $\triangleq$ 表示“定义为”。需要注意的是，$p \leftrightarrow q$ 还包括取值为 $False$ 的情形，即双向条件 $p \leftrightarrow q$ 不成立的情形。实际上，如果 $p \leftrightarrow q$ 始终成立，则称 p 与 q 逻辑等价，后面将对其进行讨论。

p	q	$p \wedge q$	$p \vee q$	$(p \wedge q) \leftrightarrow (p \vee q)$
F	F	F	F	T
F	T	F	T	F
T	F	F	T	F
T	T	T	T	T

下面，我们来看一个稍复杂的恒真式：$((p \wedge q) \to k) \leftrightarrow (p \to (q \to k))$。如何验证它为恒真式呢？一种方法是构造其真值表：

p	q	k	$p \wedge q$	$(p \wedge q) \to k$	$q \to k$	$p \to (q \to k)$
F	F	F	F	T	T	T
F	F	T	F	T	T	T
F	T	F	F	T	F	T
F	T	T	F	T	T	T
T	F	F	F	T	T	T
T	F	T	F	T	T	T
T	T	F	T	F	F	F
T	T	T	T	T	T	T

从上面的真值表可以看出，第 5 列与第 7 列的对应行都相同，这意味着 $((p \wedge q) \to k) \leftrightarrow (p \to (q \to k))$ 恒为真，因而它是一个恒真式。

实际上，对任意命题 a 与 b，如果 $a \leftrightarrow b$ 恒为真，这意味着对原子命题的每个赋值，a 与 b 的取值都相等，此时我们称 a 与 b 逻辑等价 (Logical Equivalence)，简称为等价，记为 $a \equiv b$。换句话说，如果 2 个命题的真值表完全一样，则这 2 个命题一定 (逻辑) 等价。显然，如果令 $\mathcal{T}$ 表示恒真式，令 $\mathcal{C}$ 表示恒假式，则 $\neg\mathcal{T} \equiv \mathcal{C}$ 与 $\neg\mathcal{C} \equiv \mathcal{T}$ 成立。

再回到上面的例子，我们可以利用逻辑等价的定义来证明 $((p \wedge q) \to k) \leftrightarrow (p \to (q \to k))$ 为恒真式或者 $((p \wedge q) \to k) \equiv (p \to (q \to k))$。利用前面介绍的逻

辑等价公式 $a \to b \equiv \neg a \vee b$，可以得到[8]：

$$\begin{aligned}(p \wedge q) \to k &\equiv \neg (p \wedge q) \vee k \\ &\equiv \neg p \vee \neg q \vee k \\ &\equiv \neg p \vee (q \to k) \\ &\equiv p \to (q \to k)\end{aligned}$$

设命题集合为 $\{p_1, p_2, \cdots, p_n\}$ 如果 $p_1 \wedge p_2 \wedge \cdots \wedge p_n \equiv \mathcal{C}$，即集合中的所有命题不可能同时都为真，那么就称该命题集合是不相容的或不一致的 (Inconsistent)；否则，称该命题集合是相容的或一致的。例如，$\{p_1, p_1 \to p_2, \neg p_2, p_3, \cdots, p_n\}$ 是不相容的。

显然，上述方法行之有效，它利用逻辑等价将命题从一种形式变换为等价的另一种形式。实际上，这也是一种有效的推理方法。当然，要进行真正的逻辑推理，还需要其它的推理方法，后面将着重进行讨论。推理是形式逻辑的核心问题。

2.4 基本的逻辑等价式

下面，列出一些常用的基本逻辑等价式[9]：

1. 双重否定 (Double Negation)

   $\neg\neg p \equiv p$

2. 交换律 (Commutative Law)

   $p \wedge q \equiv q \wedge p$、$p \vee q \equiv q \vee p$

3. 结合律 (Associative Law)

   $(p \wedge q) \wedge k \equiv p \wedge (q \wedge k)$、$(p \vee q) \vee k \equiv p \vee (q \vee k)$

4. 分配律 (Distribution Law)

   $p \vee (q \wedge k) \equiv (p \vee q) \wedge (p \vee k)$、$p \wedge (q \vee k) \equiv (p \wedge q) \vee (p \wedge k)$

5. 等幂律 (Idempotent Law)

   $p \wedge p \equiv p$、$p \vee p \equiv p$

[8]参见2.1节，直接依据真值表，易证 $a \to b \equiv \neg a \vee b$。

[9]为了维持完整性，将已列出的基本逻辑等价式也一并列出。

6. 同一律 (Identity Law)

   $p \vee False \equiv p$、$p \wedge True \equiv p$

7. 零律 (Dominance Law)

   $p \vee True \equiv True$、$p \wedge False \equiv False$

8. 恒真式 (Tautology) 或排中律 (Law of Excluded Middle)

   $p \vee \neg p \equiv True$

9. 恒假式或矛盾式 (Contradiction) 或矛盾律 (Law of Contradiction)

   $p \wedge \neg p \equiv False$

10. 摩根律 (De Morgan's Law) 或摩根定律

    $\neg (p \wedge q) \equiv \neg p \vee \neg q$、$\neg (p \vee q) \equiv \neg p \wedge \neg q$

11. 吸收律 (Absorption Law)[10]

    $p \vee (p \wedge q) \equiv p$、$p \wedge (p \vee q) \equiv p$

12. 条件等价式 (Conditional Equivalence)

    $p \rightarrow q \equiv \neg p \vee q$

13. 双向条件等价式 (Biconditional Equivalence)

    $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$

14. 逆反等价式 (Contrapositive Equivalence)

    $p \rightarrow q \equiv \neg q \rightarrow \neg p$

    证明：

$$\begin{aligned} \neg q \rightarrow \neg p &\equiv \neg (\neg q) \vee \neg p \\ &\equiv q \vee \neg p \\ &\equiv \neg p \vee q \\ &\equiv p \rightarrow q \end{aligned}$$

□

[10]利用真值表可以验证，且由 $p \vee (p \wedge q) \equiv p$ 和 $p \wedge (p \vee q) \equiv p$，可知 $p \vee (p \wedge q) \equiv p \wedge (p \vee q)$。

15. 双向条件否定等价式 (Biconditional Negation Equivalence)

$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$

证明：

$$\begin{aligned} p \leftrightarrow q &\equiv (p \rightarrow q) \wedge (q \rightarrow p) \\ &\equiv (\neg q \rightarrow \neg p) \wedge (\neg p \rightarrow \neg q) \\ &\equiv \neg p \leftrightarrow \neg q \end{aligned}$$

□

16. 归谬法 (Reductio ad Absurdum)[11]

$(p \rightarrow q) \wedge (p \rightarrow \neg q) \equiv \neg p$

证明：

$$\begin{aligned} (p \rightarrow q) \wedge (p \rightarrow \neg q) &\equiv (\neg p \vee q) \wedge (\neg p \vee \neg q) \\ &\equiv \neg p \vee (q \wedge \neg q) \\ &\equiv \neg p \end{aligned}$$

□

例 **2.1** 试证明 $p \vee q \rightarrow k \equiv (p \rightarrow k) \wedge (q \rightarrow k)$。

证明：

$$\begin{aligned} p \vee q \rightarrow k &\equiv \neg (p \vee q) \vee k \\ &\equiv (\neg p \wedge \neg q) \vee k \\ &\equiv (\neg p \vee k) \wedge (\neg q \vee k) \\ &\equiv (p \rightarrow k) \wedge (q \rightarrow k) \end{aligned}$$

□

2.5 论证、有效论证与谬误推理

论证 (Argument) 是一个命题序列，它的最后一个命题被称为结论 (Conclusion)，而其它所有的命题被称为前提 (Premise) 或假设 (Assumption/Hypothesis)。

[11] 归谬法常常被反证法用来否定反面观点。但是，归谬法与反证法还是有所区别的，它们属于两种不同的论证方法。反证法着眼于证明，而归谬法则立足于反驳，即使是在被反证法所使用时也是一样。此外，归谬法在被反证法用来否定反面观点时，只是在反证法整个论证过程的局部起作用，并不具备反证法的整体功能。同时，归谬法也不是反证法用来否定反面观点的唯一方法。反证法还常常使用其他的方法 (如例证法) 来否定反面观点。具体论述，请参考文献 [10]。

论证可以表示为 $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c$ 形式。其中，$p_i(1 \leqslant i \leqslant n)$ 表示前提，c 表示结论。从形式上看，论证就是条件 $if\text{-}then$ 运算符构成的复杂命题。根据前面的讨论，如果所有的前提为真，而结论为假，那么论证取值为 $False$。此时，我们称该论证为无效论证 (Invalid Argument)。因此，为了确保推理能够有效地进行，使得论证在整个推理进程中成为有效的一个环节，要求所有的前提与结论要构成一个有效的论证 (Valid Argument)。有效论证指的是，当所有的前提为真时结论必须为真的论证。另一方面，如果前提的合取 $p_1 \wedge p_2 \wedge \cdots \wedge p_n$ 取值为 $False$，则依据 $if\text{-}then$ 的定义，论证 $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c$ 的取值也为 $True$。因此，有效论证也可以定义为：

$$(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c \equiv \mathcal{T}$$

其中 $\mathcal{T}$ 表示恒真式。

对于有效论证 $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c$，我们称 $\{p_1, p_2, \cdots, p_n\}$ 蕴涵 (Entailing)c，记为：

$$\{p_1, p_2, \cdots, p_n\} \models c$$

现在，假设前提集合为 $\{p \rightarrow q, p\}$，试判断论证 $((p \rightarrow q) \wedge p) \rightarrow q$ 是否为有效论证？一种简单的方法是，使用真值表：

p	q	$p \rightarrow q$	$(p \rightarrow q) \wedge p$	$((p \rightarrow q) \wedge p) \rightarrow q$
F	F	T	F	T
F	T	T	F	T
T	F	F	F	T
T	T	T	T	T

可以看出，论证 $((p \rightarrow q) \wedge p) \rightarrow q \equiv \mathcal{T}$，确为恒真式，因而为有效论证。实际上，可以对上面的方法进行简化。根据有效论证的定义，我们只需要关注真值表中所有前提的合取式取值为 $True$ 的那些行即可。我们把这些行称为关键行 (Critical Row)。在本例的真值表中，第 4 行就是关键行。

对于有效论证，我们可以使用某种形式的“推理”模板来表示。例如，对于上例，有效论证可表示为：

$$\frac{p \rightarrow q; p}{q}$$

可以看出，所有的前提位于上方，结论位于下方。

谬误推理 (Fallacy) 指的是在推理的过程中出现了错误，从而导致了一个无效论证[12]。例如：

$$\frac{p \to q; q}{p}$$

它从给定的结论出发，使用 *if-then* 条件式直接得到前提成立这一结论。为证明该论证为无效论证，可以列出真值表中的一个关键行，并表明其为反例 (Counter Example)，从而间接地表明该论证不是一个恒真式，因而为无效论证：

p	q	$p \to q$	p
F	T	T	F

下面，列出一些常见的有效论证：

1. 假言推理 (Modus Ponens)

$$\frac{p \to q; p}{q}$$

2. 否定后件推理 (Modus Tollens)

$$\frac{p \to q; \neg q}{\neg p}$$

利用逻辑等价公式 $p \to q \equiv \neg q \to \neg p$ 与假言推理，即可得到。

3. 合取消去 (And-Elimination)

$$\frac{p \wedge q;}{p}$$

列出真值表中的关键行：

p	q	$p \wedge q$	$(p \wedge q) \to p$
T	T	T	T

可知，$(p \wedge q) \to p \equiv \mathcal{T}$，因而为有效论证。

[12] 当然，也存在着谬误推理导致无效论证，但由于某种原因 (例如偶然性或者结论本身为一个事实) 而使得结论依然正确的情形。例如，令 p 表示“美国是一个“超级强国””，令 q 表示“美国的军事力量很强大”，则依据谬误推理 $\frac{p \to q; q}{p}$ 得出 p 成立的结论，即“美国的军事力量很强大”，因而“美国是一个“超级强国””。实际上，p 和 q 都在描述一个“事实”。

4. 合取添加 (Conjunctive Addition)

$$\frac{p; q}{p \wedge q}$$

列出真值表的关键行，易证它为有效论证。此处及下面类似的地方，不再赘述。

5. 析取添加 (Disjunctive Addition)

$$\frac{p}{p \vee q}$$

6. 析取三段论 (Disjunctive Syllogism) 或归结规则 (Resolution Rule)

$$\frac{p \vee q; \neg p}{q}$$

7. 矛盾规则 (Contradiction Rule)

$$\frac{\neg p \rightarrow False}{p}$$
$$\frac{\neg p \rightarrow \mathcal{C}}{p}$$

8. 两难推理 (Dilemma Inference)

$$\frac{p \vee q; p \rightarrow r; q \rightarrow r}{r}$$

考察真值表的关键行：要确保 $p \vee q$、$p \rightarrow r$ 与 $q \rightarrow r$ 的取值均为 $True$ 值，则至少要保证 p 与 q 中至少 1 个取 $True$ 值——不论哪种情况，r 必须也取 $True$ 值，才能确保后 2 个前提的取值为 $True$。因而，上述论证为有效论证。

9. 假言三段论 (Hypothetical Syllogism)

$$\frac{p \rightarrow q; q \rightarrow r}{p \rightarrow r}$$

证明：

$$
\begin{aligned}
(p \to q) \wedge (q \to r) &\equiv (p \to q) \wedge (\neg q \vee r) \\
&\equiv ((p \to q) \wedge \neg q) \vee ((p \to q) \wedge r) \\
&\equiv (((p \to q) \wedge \neg q) \vee (p \to q)) \wedge (((p \to q) \wedge \neg q) \vee r) \\
&\equiv (p \to q) \wedge (((p \to q) \wedge \neg q) \vee r) \\
&\equiv (p \to q) \wedge (((\neg p \vee q) \wedge \neg q) \vee r) \\
&\equiv (p \to q) \wedge ((\neg p \wedge \neg q) \vee r) \\
&\equiv (p \to q) \wedge (\neg q \vee r) \wedge (\neg p \vee r) \\
&\equiv (p \to q) \wedge (q \to r) \wedge (p \to r)
\end{aligned}
$$

因此，依据合取消去规则，得到：

$$
\frac{p \to q; q \to r; p \to r}{p \to r} \equiv \frac{p \to q; q \to r}{p \to r}
$$

□

除了上述列出的有效论证外，所有的逻辑等价都可以作为推理规则，它们都是可靠的推理。

2.6 命题逻辑与 SAT 问题

满足性 (Satisfiability, SAT) 问题，又被称为布尔满足性 (Boolean Satisfiability, B-SAT) 问题，指的是为逻辑语句中所有的命题变量 (Propositional Variables) 找到一组或全部的真值赋值 (Truth Value Assignment)，使得所给定的逻辑语句成立 (即结果为 $True$)。例如，现有 2 个语句 $A \vee B \vee \neg C$ 和 $\neg A \vee D$，通过设置 $A = False$、$B = True$、$C = False$、$D = False$，可以使得上述 2 条语句取值为 $True$。

SAT 是第 1 个依据 NP-Complete 定义被证明的问题，由 Cook 完成[13]。

从约束满足问题的角度看，SAT 就是一个特殊的 CSP 问题，它的值域被限制为 $\{T, F\}$，而约束表示为子句 (Clause)。当然，还有其它形式的 SAT，例如 3-SAT。

[13]此后，许多问题都陆陆续续被归约为 SAT 问题，因而被证明也属于 NP-Complete 问题，例如 Hamilton 回路与 TSP 问题等。需要注意的是，NP-Complete 问题仅仅指的是最坏情况下的时间复杂度。在实际应用中，许多问题还是可以得到快速求解。

对于 SAT 而言，无论从理论的角度，还是从实际的角度，它都占据着非常重要的地位：

- 许多问题可以转换或归约为 SAT 问题，其中也包括约束满足问题；
- 已经开发出了一些非常高效的 SAT 求解器 (Solver)，并且还一直不断地在进行优化。SAT 求解器可以应用在许多实际问题中；

SAT 模型 (Model) 指的是使逻辑语句成立的一组真值赋值。例如，$A \vee B \vee \neg C$ 的一个模型为 $\{A = True, B = False, C = True\}$。有时候，使用 $M(s)$ 表示逻辑语句 s 的所有模型，并由此得到恒真式与恒假式的另一个定义：如果 $M(s)$ 包括所有可能的真值赋值 (模型)，则 s 为恒真式；如果 $M(s)$ 为空集，则 s 为恒假式。

因此，SAT 求解器的目标是找到 $M(s)$ 或其中的 1 个元素。

2.6.1 实例：n 皇后问题

下面，讨论如何使用命题逻辑与 SAT 求解器解决 n 皇后问题。

首先，从最简单的 2 皇后问题开始。当然，2 皇后问题是没有解的，但是可以从中学习到两者结合的方法。下表给出了 2 皇后问题中的所有逻辑变量：

A	B
C	D

在该表中，总共有 4 个逻辑变量，分别表示 4 个方格中有无放置皇后。例如，A 表示第 1 行第 1 列放置了皇后，而 $\neg A$ 则表示该处没有放置皇后，其余变量的语义，依此类推。

下面，需要使用命题逻辑来表达 2 皇后问题上的约束关系。约束关系体现在行、列及对角线上，即同一行与同一列上只能放置 1 个皇后，同一对角线上至多只能放置 1 个皇后。例如，第 1 行约束为：

$$(A \wedge \neg B) \vee (\neg A \wedge B)$$

但是，一些 SAT 求解器需要使用合取范式 (Conjunction Normal Form, CNF) 来表示逻辑语句[14]。所谓合取范式，指的是由若干单元子句组成的合取式，其中单元

[14]可以证明，所有的命题逻辑语句都可以转换为合取范式。

子句为简单析取式。例如，$(A \vee B) \wedge (\neg A \vee C) \wedge \neg B$ 就是合取范式，其中简单析取式 $(A \vee B)$、$(\neg A \vee C)$ 与 $\neg B$ 为单元子句[15]。于是，应用 2 次分配律可以将第 1 行约束转换为合取范式：

$$\begin{aligned}&(A \wedge \neg B) \vee (\neg A \wedge B)\\ \equiv\ &((A \wedge \neg B) \vee \neg A) \wedge ((A \wedge \neg B) \vee B)\\ \equiv\ &(A \vee B) \wedge (\neg A \vee \neg B)\end{aligned}$$

上式表达的含义是明确的，要使该合取范式成立，A 与 B 不能同时为 $True$，也不能同时为 $False$，即只能 1 个 $True$ 和 1 个 $False$。实际上，还可以这样处理第 1 行的约束：

- A 与 B 二选一：$A \vee B$；
- 如果 A 成立，意味着 $A \rightarrow \neg B \equiv \neg A \vee \neg B$；
- 如果 B 成立，则意味着 $B \rightarrow \neg A \equiv \neg B \vee \neg A$；

将上面 3 项合取，同样得到：

$$(A \vee B) \wedge (\neg A \vee \neg B) \wedge (\neg B \vee \neg A) \equiv (A \vee B) \wedge (\neg A \vee \neg B)$$

类似地，可以分别得到第 2 行、第 1 列、第 2 列、主对角线、次对角线的合取范式为：

$$(C \vee D) \wedge (\neg C \vee \neg D)$$

$$(A \vee C) \wedge (\neg A \vee \neg C)$$

$$(B \vee D) \wedge (\neg B \vee \neg D)$$

$$(A \vee D) \wedge (\neg A \vee \neg D)$$

$$(B \vee C) \wedge (\neg B \vee \neg C)$$

于是，得到 2 皇后问题的合取范式为：

$$(A \vee B) \wedge (\neg A \vee \neg B) \wedge (C \vee D) \wedge (\neg C \vee \neg D) \wedge (A \vee C) \wedge (\neg A \vee \neg C) \wedge$$

$$(B \vee D) \wedge (\neg B \vee \neg D) \wedge (A \vee D) \wedge (\neg A \vee \neg D) \wedge (B \vee C) \wedge (\neg B \vee \neg C)$$

在实际应用中，如果使用 Minisat 求解器，则可以将上述合取范式转换为 Minisat 求解器所需的特定格式，并保存到文本文件中[16]。例如，以下内容表示了 2 皇后问题的合取范式：

[15] 可以将 $\neg B$ 理解为 $\neg B \vee False$，它是一个简单析取式。

[16] 关于 Minisat 求解器的安装与运行示例，请阅读附录5.1。

```
 c N=2 queens problem
 p cnf 4 12
 1  2 0
-1 -2 0
 3  4 0
-3 -4 0
 1  3 0
-1 -3 0
 2  4 0
-2 -4 0
 1  4 0
-1 -4 0
 2  3 0
-2 -3 0
```

说明如下：

- 第 1 行，c 表示本行为注释行；
- 第 2 行，4 表示逻辑变量的个数。此例，逻辑变量的编号分别为 $A=1$、$B=2$、$C=3$ 及 $D=4$。12 表示子句的个数；
- 第 3-14 行，每行表示析取子句，0 表示子句结尾；数字前的“-”表示否定 $\neg$；

下面，讨论 3 皇后问题。下表给出了该问题中的所有逻辑变量：

A	B	C
D	E	F
G	H	I

在该表中，总共有 9 个逻辑变量，分别表示 9 个方格中有无放置皇后。显然，与 2 皇后问题一样，3 皇后问题也无解。第 1 行约束可表示为：

$$(A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C)$$

但是，上式不是一个合取范式，需要将其转换为合取范式，比较麻烦。不过，还有一种更直接的方式来表达上述约束关系：

- A、B 与 C 三选一：$A \vee B \vee C$；
- 如果 A 成立，则 $(A \rightarrow \neg B) \wedge (A \rightarrow \neg C) \equiv (\neg A \vee \neg B) \wedge (\neg A \vee \neg C)$；
- 如果 B 成立，则 $(B \rightarrow \neg A) \wedge (B \rightarrow \neg C) \equiv (\neg B \vee \neg A) \wedge (\neg B \vee \neg C)$；

- 如果 C 成立，则 $(C \to \neg A) \wedge (C \to \neg B) \equiv (\neg C \vee \neg A) \wedge (\neg C \vee \neg B)$；

将上面三项合取合并，去掉重复项，得到合取范式：

$$(A \vee B \vee C) \wedge (\neg A \vee \neg B) \wedge (\neg A \vee \neg C) \wedge (\neg B \vee \neg C)$$

上式表达的含义是，A、B 与 C 中至少有 1 个为 $True$，但是任意一对逻辑变量不能同时为 $True$。这意味着 A、B 与 C 中有且仅有 1 个 $True$。

类似地，可以分别得到第 2-3 行与第 1-3 列的合取范式分别为：

$$(D \vee E \vee F) \wedge (\neg D \vee \neg E) \wedge (\neg D \vee \neg F) \wedge (\neg E \vee \neg F)$$
$$(G \vee H \vee I) \wedge (\neg G \vee \neg H) \wedge (\neg G \vee \neg I) \wedge (\neg H \vee \neg I)$$
$$(A \vee D \vee G) \wedge (\neg A \vee \neg D) \wedge (\neg A \vee \neg G) \wedge (\neg D \vee \neg G)$$
$$(B \vee E \vee H) \wedge (\neg B \vee \neg E) \wedge (\neg B \vee \neg H) \wedge (\neg E \vee \neg H)$$
$$(C \vee F \vee I) \wedge (\neg C \vee \neg F) \wedge (\neg C \vee \neg I) \wedge (\neg F \vee \neg I)$$

在对角线方向上，长的主次对角线分别为 AEI 与 CEG，短的主次对角线分别为 BF、DH、BD 与 FH。需要注意的是，无论长短主次对角线，其上至多 (可能没有) 出现 1 个皇后，因而需要将所有变量的析取式去掉[17]，从而得到：

$$(\neg A \vee \neg E) \wedge (\neg A \vee \neg I) \wedge (\neg E \vee \neg I)$$
$$(\neg C \vee \neg E) \wedge (\neg C \vee \neg G) \wedge (\neg E \vee \neg G)$$
$$(\neg B \vee \neg F)$$
$$(\neg D \vee \neg H)$$
$$(\neg B \vee \neg D)$$
$$(\neg F \vee \neg H)$$

于是，将上面所有的约束合取，便得到了 3 皇后问题的合取范式，在此不再赘述。以下内容展示了 Minisat 所需要的约束数据内容：

```
c 3-queens problem
p cnf 9 34
1 2 3 0
-1 -2 0
```

[17]对于 2 皇后问题，也可以这么处理，但没有多大意义：仅有 2 条对角线，假设其中的 1 条对角线上没有放置 1 个皇后，那么另 1 条对角线上必然有 2 个皇后。无论如何处理，都无法使逻辑语句得到满足。

```
-1 -3 0
-2 -3 0
4 5 6 0
-4 -5 0
-4 -6 0
-5 -6 0
7 8 9 0
-7 -8 0
-7 -9 0
-8 -9 0
1 4 7 0
-1 -4 0
-1 -7 0
-4 -7 0
2 5 8 0
-2 -5 0
-2 -8 0
-5 -8 0
3 6 9 0
-3 -6 0
-3 -9 0
-6 -9 0
-1 -5 0
-1 -9 0
-5 -9 0
-3 -5 0
-3 -7 0
-5 -7 0
-2 -6 0
-4 -8 0
-2 -4 0
-6 -8 0
```

下面，讨论 4 皇后问题。下表给出了该问题中的所有逻辑变量：

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

在该表中，总共有 16 个逻辑变量，分别表示 16 个方格中有无放置皇后。尽管逻辑变量的个数较之 3 皇后问题增加了一些，但是我们仍然只需要表达行与列的“4 个逻辑变量中有且仅有 1 个 $True$”、长短主次对角线 (变量个数从 2-4 不等) 的“2-4 个逻辑变量中至多有 1 个 $True$”这样的约束关系。因此，合取范式的基本模式与 3 皇后问题中的类似。例如，对于行逻辑变量 A、B、C 与 D，其约束关系可表示为：

$$(A \vee B \vee C \vee D) \wedge$$
$$(\neg A \vee \neg B) \wedge (\neg A \vee \neg C) \wedge (\neg A \vee \neg D) \wedge (\neg B \vee \neg C) \wedge (\neg B \vee \neg D) \wedge (\neg C \vee \neg D)$$

而对于长短主次对角线上的逻辑变量，其约束关系需要去掉所有变量的简单析取式。例如，对于主对角线上的逻辑变量 A、F、K 与 P，其约束关系可表示为：

$$(\neg A \vee \neg F) \wedge (\neg A \vee \neg K) \wedge (\neg A \vee \neg P) \wedge (\neg F \vee \neg K) \wedge (\neg F \vee \neg P) \wedge (\neg K \vee \neg P)$$

以下内容展示了 4 皇后问题中 Minisat 所需要的约束数据内容：

```
c 4-queens problem (sat)
p cnf 16 84
-1 -2 0
-1 -3 0
-1 -4 0
-2 -3 0
-2 -4 0
-3 -4 0
1 2 3 4 0
-5 -6 0
-5 -7 0
-5 -8 0
-6 -7 0
-6 -8 0
-7 -8 0
5 6 7 8 0
-9 -10 0
-9 -11 0
-9 -12 0
-10 -11 0
-10 -12 0
-11 -12 0
9 10 11 12 0
-13 -14 0
-13 -15 0
-13 -16 0
-14 -15 0
-14 -16 0
-15 -16 0
13 14 15 16 0
-1 -5 0
-1 -9 0
-1 -13 0
-5 -9 0
-5 -13 0
-9 -13 0
1 5 9 13 0
-2 -6 0
-2 -10 0
-2 -14 0
-6 -10 0
-6 -14 0
-10 -14 0
2 6 10 14 0
-3 -7 0
-3 -11 0
-3 -15 0
-7 -11 0
-7 -15 0
-11 -15 0
3 7 11 15 0
-4 -8 0
-4 -12 0
-4 -16 0
-8 -12 0
-8 -16 0
-12 -16 0
4 8 12 16 0
-3 -8 0
-2 -7 0
```

```
-2 -12 0
-7 -12 0
-1 -6 0
-1 -11 0
-1 -16 0
-6 -11 0
-6 -16 0
-11 -16 0
-5 -10 0
-5 -15 0
-10 -15 0
-9 -14 0
-2 -5 0
-3 -6 0
-3 -9 0
-6 -9 0
-4 -7 0
-4 -10 0
-4 -13 0
-7 -10 0
-7 -13 0
-10 -13 0
-8 -11 0
-8 -14 0
-11 -14 0
-12 -15 0
```

该问题有 2 个解，下面是其中的一个[18]：

```
-1 -2 3 -4 5 -6 -7 -8 -9 -10 -11 12 -13 14 -15 -16 0
```

所获得的模型对应下面的解：

·	·	Q	·
Q	·	·	·
·	·	·	Q
·	Q	·	·

表2-1给出了 n 皇后问题解的个数。

2.6.2 SAT 求解算法

SAT 求解算法主要分为以下 2 种：

- 回溯法

  例如，Minisat 使用回溯法作为求解器的主要框架，具有完备性，可以证明逻辑语句是否具备可满足性。

[18] 4 皇后问题有 2 个解。Minisat 求解器每次只能找到 1 个解，需要使用一定的技巧找到所有解。详情可参考本讲义的配套代码。

表 2-1: n 皇后问题的解个数

n-Queens	Number of Solutions
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	2680
12	14200
13	73712
14	365596
15	2279184
16	14772512
17	95815104
18	666090624
19	4968057848
20	39029188884
21	314666222712

- 局部搜索算法

  例如，Walksat 在大多数情况下可以找到解；但在没有找到解的情况下，不能证明逻辑语句是否具备可满足性。

20 世纪 60 年代提出的 DPLL 算法 (Davis、Putnam、Logemann 与 Loveland, DPLL)，它将合取范式作为处理对象，属于回溯算法，其伪代码如下：

```
def DPLL(S):
    if S is empty:
        return ("satisfiable")
    while there is some unit clause {p}, or some pure literal p:
       if {p} and {~p} are both in S:
           return ("unsatisfiable")
       else:
           S = Simplify(S,p)
    if S is empty:
        return ("satisfiable")
    pick some literal p from a shortest clause in S
    if DPLL(Simplify(S,p))=="satisfiable": #Assign it True
       return ("satisfiable")
    else:
       return (DPLL(Simplify(S,~p)) #Assign it False

def Simplify(S,p):
    delete every clause in S containing p
    delete every occurrence in S of ~p
    return (S)
```

DPLL 算法在以下几个方面做了改进：

- 早期终止

  例如，$(A \vee B) \wedge (A \vee C)$，在已知 $A = True$ 的情况下，没有必要进一步对 B 和 C 进行赋值。由于 DPLL 使用合取范式，即简单析取式的合取，因此只要能够早期确定表达式的求值结果，就可以避免对一些逻辑变量进行赋值。

- 纯逻辑变量

  一个逻辑变量是纯的，如果它在所有的子句中总是以一种取值形式出现。例如，对变量 A，或者全部以 $\neg A$ 的形式出现，或者全部以 A 的形式出现。

  例如，在逻辑语句 $(A \vee B \vee \neg E) \wedge (B \vee C \vee \neg D) \wedge (E \vee \neg B \vee D)$ 中，A 和 C 为纯变量，B、D 与 E 为非纯变量。在这种情况下，可以对纯变量赋予不会导致子句取值为 $False$ 的值。例如，本例中，赋予 A 与 C 为 $True$。

  需要注意的是，对纯变量赋值之后，会使得子句求值的结果为 $True$，而值为 $True$ 的子句可以被忽略掉，并进而使得其它非纯变量变为纯变量。例如，本例中，A 与 C 赋值为 $True$ 之后，$\neg B$、D 与 E 就变为纯变量了。

这种情形，可能会引起进一步的连锁反应[19]。

- 单逻辑变量子句

  有时候，子句中只有 1 个逻辑变量，或者原本子句还包含其它一些逻辑变量，但是由于那些变量的值已经被确定，而导致子句实为单逻辑变量子句。例如，在语句 $(A \vee \neg B \vee \neg C)$ 中，当 $A = False$ 与 $B = True$ 时，它实际上就是一个单逻辑变量子句。此时，令 $C = False$ 可确保子句的求值结果为 $True$。

  这种情形，也可能引起进一步的连锁反应。

  例如，在语句 $(\neg A \vee C) \wedge (\neg C \vee \neg B) \wedge (B \vee D)$ 中，假设 $A = True$，则第 1 个子句中的 C 成为单逻辑变量子句，于是令 $C = True$；在第 2 个子句中，$\neg C = False$，$\neg B$ 成为单逻辑变量子句，于是令 $\neg B = True$；在第 3 个子句中，$B = False$，D 成为单逻辑变量子句。

下面，以一个合取范式为例，详解 DPLL 算法流程：

1. 为清晰起见，将合取范式以如下形式表示：

$$
\begin{aligned}
S_1 = & \neg n \vee \neg t \\
& m \vee q \vee n \\
& l \vee \neg m \\
& l \vee \neg q \\
& \neg l \vee \neg p \\
& r \vee p \vee n \\
& \neg r \vee \neg l \\
& t
\end{aligned}
$$

   其中，每一行为一个子句，它们之间的 $\wedge$ 符号已被省略掉。

2. 存在单逻辑子句 t，执行 $S_1 = Simplify(S_1, t)$。该简化相当于令 $t = True$，

[19]这类似于 CSP 问题中的前向检测 (Forward Checking) 算法。

则 $\neg t = False$，从而得到：

$$\begin{aligned} S_1 = & \neg n \\ & m \vee q \vee n \\ & l \vee \neg m \\ & l \vee \neg q \\ & \neg l \vee \neg p \\ & r \vee p \vee n \\ & \neg r \vee \neg l \end{aligned}$$

3. 类似地，$\neg n$ 为单逻辑子句，执行 $S_1 = Simplify(S_1, \neg n)$。该简化相当于令 $\neg n = True$，则 $n = False$，从而得到：

$$\begin{aligned} S_1 = & m \vee q \\ & l \vee \neg m \\ & l \vee \neg q \\ & \neg l \vee \neg p \\ & r \vee p \\ & \neg r \vee \neg l \end{aligned}$$

4. Pick m，执行 $Simplify(S_1, m)$。该简化相当于令 $m = True$，则 $\neg m = False$，从而得到：

$$\begin{aligned} S_1 = & l \\ & l \vee \neg q \\ & \neg l \vee \neg p \\ & r \vee p \\ & \neg r \vee \neg l \end{aligned}$$

5. 执行深度优先递归 $DPLL(Simplify(S_1, m))$，其中 $Simplify(S_1, m)$ 刚刚讨论过。为区别，将上面简化的语句命名为 S_2，继续讨论其递归调用。存在单逻辑子句 l，对其执行简化 $S_2 = Simplify(S_2, l)$。该简化相当于令 $l = True$，

则 $\neg l = False$，从而得到：

$$\begin{aligned} S_2 = & \neg p \\ & r \vee p \\ & \neg r \end{aligned}$$

6. 存在单逻辑子句 $\neg p$，执行简化 $S_2 = Simplify(S_2, \neg p)$。该简化相当于令 $\neg p = True$，则 $p = False$，从而得到：

$$\begin{aligned} S_2 = & r \\ & \neg r \end{aligned}$$

r 与 $\neg r$ 都在语句 S_2 中，因而此分支返回“$unsatisfiable$”，需要回溯执行另一个递归分支。

7. 回溯返回，执行此递归分支：执行 $Simplify(S_1, \neg m)$。该简化相当于令 $\neg m = True$，则 $m = False$，从而得到：

$$\begin{aligned} S_1 = & q \\ & l \vee \neg q \\ & \neg l \vee \neg p \\ & r \vee p \\ & \neg r \vee \neg l \end{aligned}$$

8. 递归执行 $DPLL(Simplify(S_1, \neg m))$。，其中 $Simplify(S_1, \neg m)$ 刚刚讨论过。为区别，将上面简化的语句命名为 S_2，继续讨论其递归调用。存在单逻辑子句 q，执行简化 $S_2 = Simplify(S_2, q)$。该简化相当于令 $q = True$，则 $\neg q = False$，从而得到：

$$\begin{aligned} S_2 = & l \\ & \neg l \vee \neg p \\ & r \vee p \\ & \neg r \vee \neg l \end{aligned}$$

9. 存在单逻辑语句 l,执行简化 $S_2 = Simplify(S_2, l)$。该简化相当于令 $l = True$,

则 $\neg l = False$，从而得到：

$$S_2 = \neg p$$
$$r \vee p$$
$$\neg r$$

10. 存在单逻辑语句 $\neg p$，执行简化 $S_2 = Simplify(S_2, \neg p)$。该简化相当于令 $\neg p = True$，则 $p = False$，从而得到：

$$S_2 = r$$
$$\neg r$$

11. r 与 $\neg r$ 都存在与语句 S_2 中，因而分支返回“$unsatisfiable$”。于是，整个逻辑语句返回“$unsatisfiable$”，表示不可满足。

上面介绍的 DPLL 算法，其核心框架为基于深度优先搜索的回溯法，它可以证明语句是否具备可满足性。下面介绍的 Walksat 算法，属于局部搜索算法，能够有效地为逻辑语句找到可满足的变量赋值，在有些情况下甚至比 DPLL 求解器要更快些。这种情形与 CSP 问题类似——CSP 问题也能够使用局部搜索算法 (例如 Min-Conflicts 等) 来快速地找到约束解。然而，需要注意的是，与 CSP 中的 Min-Conflicts 算法类似，Walksat 不能证明解是否存在，因为它是一个随机算法——如果 Walksat 没有找到解，那么只能表明该问题有较大的概率没有解，而并不能证明该问题没有解。

下面，直接给出 Walksat 算法伪代码。

```
choose a random initial truth assignment
while not termination condition do:
    randomly choose an unsatisfied clause
    flip one variable of this clause as follows:
        with fixed probability p:
             select the variable randomly
        or, with probability 1-p:
             select the variable greedily, i.e. select
             the variable that maximizes the number of satisfied
             clauses overall
```

显然，Walksat 算法比 DPLL 算法要简单得多。在实际应用中，Walksat 算法能够有效地解决许多大的 SAT 问题。

2.6.3 有效论证与 SAT 求解

在给定若干前提的情况下，为了论证一个逻辑结论是否成立，我们可以借助于模型检测 (Model-Checking) 的方法 (例如生成一个真值表)，也可以借助于 SAT

求解器 (例如 SAT 求解 n 皇后问题)。还有一种方法，即利用各种有效的逻辑推理规则，从前提出发，逐步推导出有效的结论，这就是所谓的有效论证。

在第2.5节中，我们讨论了有效论证：对于若干前提 p_1、p_2、$\cdots$、p_n，如果 $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c \equiv \mathcal{T}$，即 $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c$ 是一个恒真式，那么我们称 $\{p_1, p_2, \cdots, p_n\}$ 蕴涵 c。

实际上，另一方面，如果 $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c$ 为恒真式，那么它的对立面，即恒假式，$\neg((p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c)$ 就是一个不可满足的 (Unsatisfiable) 逻辑语句。这意味着，可以使用 SAT 求解器来判断一个论证是否有效：如果该式不可满足，那么此论证即为有效论证，结论成立，于是命题便得到了证明。为了应用 SAT 求解器或下面即将介绍的归结证明，通常需要将其转换为合取范式：

$$
\begin{aligned}
&\neg((p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c) \\
&\equiv \neg(\neg(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \vee c) \\
&\equiv (p_1 \wedge p_2 \wedge \cdots \wedge p_n) \wedge \neg c \\
&\equiv p_1 \wedge p_2 \wedge \cdots \wedge p_n \wedge \neg c
\end{aligned}
$$

2.7 归结证明

一般而言，逻辑智能体的知识库中通常包括了许多关于环境的事实。在每一个逻辑推理步，逻辑智能体需要从众多的推理规则中，依据合理的事实，选择合适的推理规则，逐步迭代推理，最终才能得出所需要的结论。此外，在推理的过程中，会产生大量的中间知识，这也会极大地影响推理效率。因此，如何合理地运用推理规则以及提高推理的效率，是一个非常重要的问题。

前面的若干单元中，已经引入了各式推理规则，并演示了如何模仿人类使用自然语言进行推理的思维过程。这种方法被称为自然演绎推理。然而，完全按照人的思维方式进行自然演绎推理，具有诸多不便：

- 将知识库中关于环境的事实建模为环境的状态，将推理规则的选择建模为逻辑智能体在当前状态下动作的选择，这一建模方式使得问题变得更加复杂；而且，在迭代推理的过程中，需要不断地将新事实加入到环境中，即环境是动态变化的。

- 如果单纯地使用回溯法或随机方法，也会碰到上述问题。

- 然而，更为重要的是，假设前提与结论之间的蕴涵关系存在，我们如何知道当前的推理规则是完备的 (Complete)，即能否确定这些推理规则的合适组合能够证明蕴涵关系确实成立。

下面介绍的归结证明 (Proof by Resolution) 能够解决上述问题，它于 1965 年由 Robinson 在 Herbrand 理论的基础上提出，依赖于一个单一且简单的规则——归结规则 (Resolution Rule)。归结证明是推理与定理证明的基础，其理论基础为 Herbrand 定理，它采用了反证法的思想，将恒真式的证明问题转换为不可满足性的证明问题。具体而言，将待证明逻辑公式的结论取反，并将其作为附加前提，最后再证明该逻辑公式是不可满足的。此部分内容已在上一节 (2.6.3) 的末尾部分进行了讨论。Herbrand 理论为自动定理的证明奠定了理论基础，而 Robinson 的归结证明使得自动定理证明得以实现。从某种意义上而言，大部分人工智能推理问题都可以转化为定理证明问题。因此，归结证明在人工智能推理领域有着非常重要的作用。

例如，对于 2 变量，归结规则为：

$$\frac{p \vee q; \neg q}{p}$$

上式的直观意义为，如果 $p \vee q$ 成立，且 q 不成立，则 p 一定成立。其真值表为：

p	q	$p \vee q$	$\neg q$	$((p \vee q) \wedge \neg q) \rightarrow p$
F	F	F	T	T
F	T	T	F	T
T	F	T	T	T
T	T	T	F	T

显然，$((p \vee q) \wedge \neg q) \rightarrow p$ 为恒真式，表明归结规则成立。实际上，也可以直接利用逻辑等价：

$$p \vee q \equiv \neg(\neg p) \vee q \equiv \neg p \rightarrow q \equiv \neg q \rightarrow p$$

然后，依据假言推理规则得到：

$$\frac{\neg q \rightarrow p; \neg q}{p} \equiv \frac{p \vee q; \neg q}{p}$$

进一步地，可以将归结规则泛化为：

$$\frac{p_1 \vee p_2 \vee \cdots \vee p_i \vee \cdots \vee p_n; \neg p_i}{p_1 \vee p_2 \vee \cdots \vee p_{i-1} \vee p_{i+1} \vee \cdots \vee p_n}$$

证明方法同上[20]。实际上，下面的归结规则同样成立：

$$\frac{p_1 \vee p_2 \vee \cdots \vee \neg p_i \vee \cdots \vee p_n; p_i}{p_1 \vee p_2 \vee \cdots \vee p_{i-1} \vee p_{i+1} \vee \cdots \vee p_n}$$

只要第 i 个位置上的逻辑变量为互补关系。从合取范式的角度看，$p_1 \vee p_2 \vee \cdots \vee \neg p_i \vee \cdots \vee p_n$ 为归结规则应用前的合取范式子句，$p_1 \vee p_2 \vee \cdots \vee p_{i-1} \vee p_{i+1} \vee \cdots \vee p_n$ 为归结规则应用后的合取范式子句，仍然为简单析取式，满足合取范式对子句的要求。这是归结规则的优点之一。

还有另一种泛化形式的归结规则——全归结规则 (Full Resolution Rule)：

$$\frac{p \vee q; r \vee \neg q}{p \vee r}$$

可以看出，前提中包含 2 个简单析取式，且 2 个析取式中分别包含了 2 个互补的逻辑变量 q 与 $\neg q$。全归结规则也是成立的：

$$p \vee q; r \vee \neg q \equiv \neg(\neg p) \vee q; \neg q \vee r \equiv \neg p \to q; q \to r$$

根据假言三段论规则，可得：

$$\frac{\neg p \to q; q \to r}{\neg p \to r} \equiv \frac{p \vee q; r \vee \neg q}{p \vee r}$$

类似地，可以将全归结规则推广到如下情形：

$$\frac{p_1 \vee p_2 \vee \cdots \vee p_i \vee \cdots \vee p_n; r_1 \vee r_2 \vee \cdots \vee r_i \vee \cdots \vee r_n}{p_1 \vee p_2 \vee \cdots \vee p_{i-1} \vee p_{i+1} \vee \cdots \vee p_n \vee r_1 \vee r_2 \vee \cdots \vee r_{i-1} \vee r_{i+1} \vee \cdots \vee r_n}$$

其中，假设 p_i 与 r_i 为互补变量。需要注意的是，如果前提中的 2 个子句存在相同的逻辑变量，则在合并时，简单地去掉其中的一个重复变量，因为 $p \vee p \equiv p$。例如：

$$\frac{p \vee q \vee \neg r; p \vee q \vee r \vee s}{p \vee q \vee s}$$

如果 2 个前提中包含 2 对互补的逻辑变量，应用全归结规则之后，将得到 $True$。例如：

$$\frac{p \vee q \vee \neg r; p \vee \neg q \vee r \vee s}{p \vee \neg r \vee r \vee s} \equiv \frac{p \vee q \vee \neg r; p \vee \neg q \vee r \vee s}{True}$$

[20]例如，将 $p_1 \vee p_2 \vee \cdots \vee p_{i-1} \vee p_{i+1} \vee \cdots \vee p_n$ 看作一个整体，再利用前述方法加以证明。

可以证明，只要逻辑语句之间的蕴涵 $\models$ 关系存在或成立，则归结规则具有完备性，即使用归结证明可以得到所需要的结论。需要注意的是，与大多数 SAT 求解器的需求一样，归结规则要求逻辑语句使用合取范式[21]。如前面的示例所示，我们在介绍归结规则时，全部使用了合取范式。

例 **2.2** 证明公式 $((p \vee q) \wedge (p \rightarrow r) \wedge (q \rightarrow r)) \rightarrow r$。

证明：此证明将借助第2.6.3节末尾的结论。首先，将所有的子句转换为简单析取式[22]：

Step	Disjunction	Derivation
1	$p \vee q$	Given, KB
2	$\neg p \vee r$	$p \rightarrow r$, KB
3	$\neg q \vee r$	$q \rightarrow r$, KB
4	$\neg r$	Negated Conclusion, KB

然后，迭代执行归结规则：在归结 2 个子句时，如果生成了空语句，这意味着产生了形如 $s \wedge \neg s$ 的矛盾式，因而逻辑公式成立；或者，反复执行归结规则，直至没有可归结的子句为止，即没有新子句产生，这表明逻辑公式不成立。执行归结：

Step	Disjunction	Derivation
5	$q \vee r$	1,2
6	$\neg p$	2,4
7	$\neg q$	3,4
8	r	5,7
9	$\cdot$	4,8

可以看出，最后产生了矛盾式 $\neg r \wedge r$，因而逻辑公式成立[23]。 □

[21]任何逻辑语句可以转换为合取范式。

[22]子句也包括结论，需要对结论取反，得到 $\neg r$，它本身就是一个最简单的析取式。

[23]需要注意的是，不同的归结顺序，会使得归结步数有长有短。在设计归结算法时，可以使用启发式规则，选取合理的归结顺序。

下面给出逻辑语句的归结算法：需要将逻辑公式 $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow c$ 的证明问题，转换为 $p_1 \wedge p_2 \wedge \cdots \wedge p_n \wedge \neg c$ 的可满足性证明问题。为描述方便，使用“p-entail-c”表示待证明的逻辑公式。

```
Resolution Algorithm

First p-entail-c is converted to CNF.
clauses with complementary literals are chosen and resolved and added
as a new clause in the knowledge base if it not already present.
this continues until one of two things happens:
    1. no new clause can be added, which means that p does not entail
       c.
    2. two clauses resolve to yield the empty clause, which means
       that p entails c.
```

3 一阶逻辑

3.1 命题逻辑的局限性

命题逻辑存在着一定的局限性，它不能对命题进行进一步的分解——即便是一些简单命题，它们也包含着一些重要的逻辑元素——缺少这些元素，将导致逻辑系统无法执行有效的推理，或导致无效的论证，甚至无法处理一些简单而又常见的推理过程。换句话说，命题逻辑没有研究命题的内部结构及内在联系，缺乏关键的表达要素。

例如，令 p 表示“每一个高中生都要学习数学”，q 表示“小李是一名高中生”，r 表示“小李也要学习数学”。从三段论的角度看，依据 p 与 q 的前提，可以很自然地得出 r 这一结论。然而，如果使用命题逻辑来表示，其论证可以表示如下：

$$\frac{p; q}{r}$$

显然，对于命题逻辑而言，这完全是一个无效的论证——$(p \wedge q) \rightarrow r$ 不是一个恒真式。其主要原因在于，p、q、r 之间没有任何关系——并没有使用具有关联性的原子命题以及必要的逻辑连接词来表示这些命题之间的 (结构) 关系[24]，因而不能形成有效的推理。为了解决上述推理问题，需要把对象 (高中生与数学) 之间的关系表示出来，还需要表示对象 (高中生) 的量词概念 (全称量词与存在量词) 或者全体与个体之间的关系。

[24]从这个角度而言，命题逻辑被称为“0 阶逻辑”(Zeroth-Order Logic)。而谓词逻辑，被称为“1 阶逻辑”(First-Order Logic)。该逻辑的表达粒度要比命题逻辑细些，能够表达语句中的精细结构与内容——例如，通过引入量词等逻辑要素来表达命题的关键内容。

命题逻辑是一种表达能力很弱的语言，无法以简洁的方式表示复杂环境的知识。例如，在 8 皇后问题中，如果要表达“每一行、每一列或对角线上，不能出现 2 个皇后”这一命题，命题逻辑不得不为每个皇后枚举一个约束规则。然而，人类的自然语言却能够简洁地对此命题进行描述。因此，如果我们能够总结自然语言的通用法则，并将之应用于表示与推理系统中，那么推理系统就能够从无数的自然语言描述的内容中受益。我们的目标是，构造一种比命题逻辑更强大而比自然语言更简洁实用且通用的逻辑——能够将知识与推理分开，且推理完全独立于问题域。

构造新逻辑的基本思路是，在命题逻辑的基础上，保留其描述性、上下文无关性、通用性且无歧义性等特点，借用自然语言的表达思想，同时避开它的缺点，构造出一种更具表达能力的逻辑。

在自然语言中，使用名词与名词短语来指代对象，使用动词与动词短语来表示对象之间的关系。在众多的关系中，有一些关系可以使用函数来表示——对于给定的输入，有一个输出。实际上，几乎每一条断言都涉及到对象或者对象之间的关系。对于一元关系，我们称之为属性，例如 (桌子) 红色的、(草) 绿色的等。对于命题“$1+3=4$”，其中 1、3 与 4 是对象，“$+$”表示对象 1 与 3 之间的函数关系，另一个关系为“$=$”，表示前述函数输出的对象与对象 4 之间的关系。

本节讨论的一阶逻辑，也被称为谓词逻辑 (Predicate Logic)，是围绕对象及其关系建立起来的。与命题逻辑一样，也包括语法、语义及证明理论。它在数学、哲学与人工智能中占据着非常重要的地位。主要原因在于，本质上，人们的日常生活、学习与工作的主要任务是处理对象及其关系。

在一阶逻辑中，引入了 (谓词) 变量，用来表示对象。更为重要的是，引入了全称量词与存在量词，可以表达问题域中某些或全部对象，这使得一阶逻辑可以表示通用规律或者规则。

实际上，除了上述 2 种逻辑之外，还存在着诸多逻辑，例如时态逻辑、高阶逻辑等。一般地，从本体论约定 (Ontological Commitment) 与认识论约定 (Epistemological Commitment) 这 2 个角度可以对逻辑进行分类。即便是各种偏于数学的模型系统 (概率论、模糊数学等)，也都可以从上述 2 个性质的视角，将它们纳入到逻辑体系中。表3-2列出了各种逻辑系统的对比。

从表3-2可以看出，命题逻辑与一阶逻辑的本质区别在于本体论约定，即关于“世界是什么”的假设不同——命题逻辑以命题的形式描述世界 (问题域) 中的事

表 3-2: 各种逻辑系统的对比

逻辑	本体论约定 (世界是什么)	认识论约定 (智能体所相信的事实)
命题逻辑	事实	$True$、$False$、未知
一阶逻辑	事实、对象与关系	$True$、$False$、未知
时态逻辑	事实、对象与关系、时间	$True$、$False$、未知
概率论	事实	可信度 ($[0,1]$)
模糊逻辑	事实，真实度 ($[0,1]$)	区间值

实成立与否，并赋值为 $True$ 或 $False$，不考虑中间状态；而一阶逻辑所建模的对象及其关系要丰富许多，它以谓词的形式描述对象之间的关系成立与否，并赋值为 $True$ 或 $False$，也不考虑中间状态。一阶逻辑的形式系统要比命题逻辑更加复杂。时态逻辑需要将特定的时间与事实进行关联，且时间是有序的。高阶逻辑将一阶逻辑中的关系与函数本身也视为对象，因而表达能力要强于一阶逻辑——一些高阶逻辑的语句无法使用有限数目的一阶逻辑语句来替换。

3.2 语法与语义

3.2.1 个体变量与谓词

在一阶逻辑中，谓词语句使用 3 个关键元素来表示，它们分别是谓词 (Predicate)、变量 (Variable) 与量词 (Quantifier)[25]。我们可以将一阶逻辑看作命题逻辑的扩展，原有的真值、逻辑连接词等仍然有效。但是，命题的内部结构被细化为谓词、变量与量词，这大大增强了一阶逻辑的表达能力。

命题是一个具有唯一真值 ($True$ 或 $False$) 的陈述句，而陈述句主要由主语、谓语、宾语和补语组成。其中，主语与谓语是句子的主要成分。主语是谓语描述的对象，被称为个体、对象或客体。谓语用于描述主语的性质或个体之间的关系。

个体可以是独立存在的客体，可以是一个具体的对象，也可以是一个抽象的概念。例如，小李、苹果、自然数、定理、宇宙、思想、精神等都可以作为个体。个体常量用来表示具体或特定的个体，一般使用小写英文字母 a、b、c 等来表示。

[25]当谓词语句中的个体变量取某个具体值而固定为具体个体时，谓词语句才实例化为具有固定真值的命题，但这并不妨碍我们为谓词语句给出类似于命题那样的真值定义——可以将命题的真值定义扩展到谓词语句上。例如，第3.2.2节给出的全称量词与存在量词的真值定义。

个体变量用来表示抽象或泛指的个体，一般使用小写英文字母 x、y、z 等来表示。个体域 (Domain) 表示个体变量的取值范围，一般使用 D 来表示。个体域可以是有限集，例如 $\{1, 2, 3, \cdots, 100\}$ 等；也可以是无限集，例如自然数集合、实数集合等。

在一阶逻辑中，刻画个体性质和关系的谓语被称为谓词。具体而言，谓词是一个包含变量 (谓词变量或个体变量) 的语句。谓词一般使用大写英文字母 P、Q、R 来表示或者直接使用具有一定意义的短语来表示，并将变量作为其参数，两者一起就构成了具有一定语义的谓词语句。当谓词变量的值固定之后，谓词 (语句) 的取值也将固定为 $True$ 或 $False$。此时，我们称谓词语句实例化为命题。例如，令谓词 $P(x)$ 表示语句 "x^2 is greater than x"，则它包含一个参数或变量 x。当 x 取某个固定值时，$P(x)$ 将依 x 的值取 $True$ 或 $False$。谓词 $P(x)$ 用来表示 2 个个体 x^2 与 x 之间的大小关系[26]。其它的一些谓词例子，列举如下：

- $P(x)$："x is an engineer"

  $P(x)$ 是表示个体 x 属性或性质的谓词。

- $P(x, y)$："x buy y, e.g. y=book"

  $P(x, y)$ 是表示主宾 2 个个体 x 与 y 之间 (动作) 关系的谓词。

- $P(x, y)$："x and y are good friends"

  $P(x, y)$ 是表示 2 个个体 x 与 y 之间关系的谓词。

- $Father(x)$："x's father"

  此时，谓词 $Father(x)$ 被称为函数。

- $1 + 5 > 3$

  "$+$" 为谓词 (函数)，"$>$" 为表示大小关系的谓词。它们都是特殊形式的谓词，直接使用运算符来表示。

谓词可分为一元谓词和多元谓词。前者关联 1 个个体，一般表示该个体的某种性质。后者关联 n 个个体，表示多个个体之间的某种关系。

[26] 一种理解方式为，x^2 和 x 是 2 个个体，只是使用 1 个变量来表示。或者，也可以理解为 1 个个体 (即 x^2)，谓词 $P(x)$ 用来表达该个体的 (大小) 属性。

3.2.2 量词

从前面的描述可以看出，个体变量的取值来自于个体域。为了刻画个体取值的数量特征，引入 2 个量词，即全称量词 (Universal Quantifier) 和存在量词 (Existential Quantifier)。

前者表示“一切”或“所有”，使用符号“$\forall$”表示。例如，$\forall x \in D, P(x)$ 表示 x 所属个体域 D 内的每个个体都有性质 $P(x)$。后者表示“存在着”或“至少有一个”，使用符号“$\exists$”表示。例如，$\exists x \in D, P(x)$ 表示 x 所属个体域 D 内至少有一个个体具有性质 $P(x)$。

下面将命题逻辑中的真值赋值扩展至带有量词的语句，给出相应的真值定义：

- 全称量词语句 $\forall x \in D, P(x)$ 为 $True$，当且仅当对于 D 中的每个个体 x，$P(x)$ 均为 $True$。设 $D = \{x_1, \cdots, x_n\}$，则：

$$(\forall x \in D, P(x)) \triangleq (P(x_1) \wedge P(x_2) \wedge \cdots \wedge P(x_n))$$

- 存在量词语句 $\exists x \in D, P(x)$ 为 $True$，当且仅当 D 中存在某个个体 x，$P(x)$ 为 $True$。设 $D = \{x_1, \cdots, x_n\}$，则：

$$(\exists x \in D, P(x)) \triangleq (P(x_1) \vee P(x_2) \vee \cdots \vee P(x_n))$$

显然，对于全称量词与存在量词语句，存在如下关系：

$$\begin{aligned} \neg(\exists x \in D, P(x)) &\equiv \neg(P(x_1) \vee P(x_2) \vee \cdots \vee P(x_n)) \\ &\equiv (\neg P(x_1) \wedge \neg P(x_2) \wedge \cdots \wedge \neg P(x_n)) \\ &\equiv \forall x \in D, \neg P(x) \end{aligned}$$

类似地，也存在如下关系：

$$\begin{aligned} \neg(\forall x \in D, P(x)) &\equiv \neg(P(x_1) \wedge P(x_2) \wedge \cdots \wedge P(x_n)) \\ &\equiv (\neg P(x_1) \vee \neg P(x_2) \vee \cdots \vee \neg P(x_n)) \\ &\equiv \exists x \in D, \neg P(x) \end{aligned}$$

进一步地，如果 $P(x)$ 为复合语句，那么可以直接利用命题逻辑中的各种逻辑等价式，对 $P(x)$ 进行等价变换。例如，如下的变换使用了摩根定律：

$$\begin{aligned} \neg(\forall x \in D, (Q(x) \wedge R(x))) &\equiv \exists x \in D, \neg(Q(x) \wedge R(x)) \\ &\equiv \exists x \in D, (\neg Q(x) \vee \neg R(x)) \end{aligned}$$

如何为带有量词的语句确定真值呢？[27]一般而言，有如下几种方法：

1. 穷举法 (Method of Exhaustion)

   例如，设 $D=\{1,2,3,4\}$，试确定语句 $\exists x \in D, x>6$ 的真值：$1>6$ 为 $False$，$2>6$ 为 $False$，$3>6$ 为 $False$，$4>6$ 为 $False$，因而语句的真值为 $False$。

2. 例证法 (Method of Case)

   例如，确定语句 $\exists x \in \mathbb{Z}, x^2=x$ 的真值：例举 1 个正例 (Positive Example)，如 $x=0$，关系成立，因而语句的真值为 $True$。

   例如，确定语句 $\forall x \in \mathbb{R}, x^2>x$ 的真值：例举 1 个反例 (Counter Example)，如 $x=0.5$，关系不成立，因而语句的真值为 $False$。

   可以看出，为了确定带存在量词的语句真值为 $True$，可以例举正例；而为了确定带全称量词的语句真值为 $False$，可以例举反例。需要注意的是，不能例证带存在量词的语句真值为 $False$，这等价于不能例证带全称量词的语句真值为 $True$[28]。

3. 逻辑推导 (Method of Logic Derivation)

   逻辑推导的基本思路是，利用各种逻辑等价式或推理规则，将一个逻辑表达式转换为另一个逻辑表达式，为语句的真值确定提供了便利。

   一个简单的示例如下：

$$\begin{aligned}\forall x \in D, (P(x) \wedge Q(x)) &\equiv ((P(x_1) \wedge Q(x_1)) \wedge \cdots \wedge (P(x_n) \wedge Q(x_n))) \\ &\equiv (P(x_1) \wedge \cdots \wedge P(x_n)) \wedge (Q(x_1) \wedge \cdots \wedge Q(x_n)) \\ &\equiv (\forall x \in D, P(x)) \wedge (\forall x \in D, Q(x))\end{aligned}$$

   关于更多的逻辑推导规则，将在后面进行讨论。

量词可以嵌套使用，被称为嵌套量词 (Nested Quantifier)。例如，$\forall x \in \mathbb{R}, \exists y \in \mathbb{R}, y=2*x+3$，该语句表达的含义为“对于任意的 $x \in \mathbb{R}$，存在着 $y \in \mathbb{R}$，使得 $y=2*x+3$ 成立。我们可以使用谓词 $P(y,x)$ 来表示上述关系或含义，即

[27]需要注意的是，有些谓词语句不具有固定的真值——依据个体变量的取值，或得到 $True$，或得到 $False$，即其具体取值只有在个体变量确定之后才能被确定。

[28]即不能使用有限的几个例子来确定带存在量词的语句真值为 $False$ 以及确定带全称量词的语句真值为 $True$；否则，推理就犯了“以偏概全”的错误。

$\forall x \in \mathbb{R}, \exists y \in \mathbb{R}, P(y,x)$。该谓词语句准确地表达了二维直角坐标系上 y 与 x 之间的函数关系，属于真命题。然而，如果将该谓词语句的量词顺序颠倒，即变为 $\exists y \in \mathbb{R}, \forall x \in \mathbb{R}, P(y,x)$，那么其语义将变为“存在 $y \in \mathbb{R}$，对于任意的 $x \in \mathbb{R}$，都有 $y = 2 * x + 3$ 关系成立”。显然，其语义与原语义明显不同，且 y 与 x 之间的函数关系不再成立，属于假命题。因此，要特别注意量词的嵌套顺序，否则可能改变了语义，也可能使得命题的性质发生了根本变化。一般而言，$\forall x \in \mathbb{R}, \exists y \in \mathbb{R}, P(y,x) \not\equiv \exists y \in \mathbb{R}, \forall x \in \mathbb{R}, P(y,x)$。

如果存在诸如“$\forall x \forall y \forall z \cdots$”与“$\exists x \exists y \exists z \cdots$”这样的量词，可以将它们简记为“$\forall x, y, z \cdots$”与“$\exists x, y, z \cdots$”。

下面将条件 *if-then* 运算符扩展至带全称量词的谓词语句。全称量词语句 $\forall x \in D, (P(x) \rightarrow Q(x))$ 为 $True$，当且仅当对于 D 中的每个个体 x，$P(x) \rightarrow Q(x)$ 均为 $True$。设 $D = \{x_1, \cdots, x_n\}$，则：

$$\forall x \in D, (P(x) \rightarrow Q(x)) \triangleq ((P(x_1) \rightarrow Q(x_1)) \wedge \cdots \wedge (P(x_n) \rightarrow Q(x_n)))$$

注意到，如果 $P(x_i)$ 本身不成立，即取值为 $False$，那么依据 *if-then* 逻辑运算符的定义，$P(x_i) \rightarrow Q(x_i)$ 的取值仍然为 $True$。这意味着，这些合取子项对我们真正需要关注的其它合取子项不会产生任何影响，简单地忽略掉即可。因此，我们只需要关注那些 $P(x_i)$ 取值为 $True$ 的谓词，对其对应的 $Q(x_i)$ 取值进行断言。从这个角度来看，为 *if-then* 定义“奇特的”的真值规则是合理而完美的；而利用全称量词语句 $\forall x \in D, (P(x) \rightarrow Q(x))$ 来表达某些个体应该遵循的规则 $P(x) \rightarrow Q(x)$ 也是非常恰当的——无需给那些前提 $P(x)$ 不成立的个体任何断言。

例如，设 $D = \{1, 3, 5, 7, 9, 10\}$，令 $P(x)$ 表示“x 是偶数”，令 $Q(x)$ 表示“$x\%2 == 0$”。容易验证，语句 $\forall x \in D, (P(x) \rightarrow Q(x))$ 取值为 $True$。该语句简洁地表达了“集合 D 中所有偶数具备的性质”，尽管 D 中还包含着奇数——这些奇数不满足前提 $P(x_i)$，即 $P(x_i) = False$，则 $P(x_i) \rightarrow Q(x_i) \equiv True$，其中 $i = 1, 2, \cdots, 5$：

$$\begin{aligned}
&\forall x \in D, (P(x) \rightarrow Q(x)) \\
&\equiv ((P(x_1) \rightarrow Q(x_1)) \wedge \cdots \wedge (P(x_6) \rightarrow Q(x_6))) \\
&\equiv True \wedge True \wedge True \wedge True \wedge True \wedge (P(x_6) \rightarrow Q(x_6)) \\
&\equiv (P(x_6) \rightarrow Q(x_6)) \\
&\equiv True
\end{aligned}$$

类似地，可以将条件 $if\text{-}then$ 运算符扩展至带存在量词的谓词语句。存在量词语句 $\exists x \in D, (P(x) \to Q(x))$ 为 $True$，当且仅当 D 中至少存在 1 个个体 x，使得 $P(x) \to Q(x)$ 取值为 $True$。设 $D = \{x_1, \cdots, x_n\}$，则：

$$\exists x \in D, (P(x) \to Q(x)) \triangleq ((P(x_1) \to Q(x_1)) \vee \cdots \vee (P(x_n) \to Q(x_n)))$$

然而，$if\text{-}then$ 使得此类谓词语句的陈述能力过弱，或者认为 $if\text{-}then$ 推理的意义不大。换句话说，这样的语句或推理并没有表达真正有用的信息。主要原因在于，如果前提 $P(x)$ 本身不成立，则无论 $Q(x)$ 取何值，$P(x) \to Q(x)$ 总返回 $True$。

从上面的讨论可以看出，$if\text{-}then$ 对于全称量词 $\forall$ 而言是非常自然的，但对于存在量词 $\exists$ 而言是不恰当的。然而，$\wedge$ 对于存在量词 $\exists$ 而言却是非常自然的，但对于全称量词 $\forall$ 而言，则显得过于严厉。还是对上面的例子进行讨论，对于全称量词：

$$\begin{aligned}\forall x \in D, (P(x) \wedge Q(x)) &\equiv ((P(x_1) \wedge Q(x_1)) \wedge \cdots \wedge (P(x_6) \wedge Q(x_6))) \\ &\equiv False \wedge False \wedge False \wedge False \wedge False \wedge (P(x_6) \wedge Q(x_6)) \\ &\equiv False\end{aligned}$$

上述语句取值为 $False$。显然，它无法表达前述语句的含义。对于存在量词：

$$\begin{aligned}\exists x \in D, (P(x) \wedge Q(x)) &\equiv ((P(x_1) \wedge Q(x_1)) \vee \cdots \vee (P(x_6) \wedge Q(x_6))) \\ &\equiv False \vee False \vee False \vee False \vee False \vee (P(x_6) \wedge Q(x_6)) \\ &\equiv (P(x_6) \wedge Q(x_6)) \\ &\equiv True\end{aligned}$$

上述语句取值为 $True$。它很自然地表达了前述语句的含义。

再举一例，令 $P(x)$ 表示“$x > 1$”，$Q(x)$ 表示“$x^2 > 1$”，其中 $x \in \mathbb{R}$，则全称量词语句：

$$\forall x \in \mathbb{R}, (P(x) \to Q(x))$$

其真正要表达的含义是，对于 $\forall x \in \mathbb{R}$，如果 $P(x)$ 成立，那么 $Q(x)$ 成立，即如果 $x > 1$，则 $x^2 > 1$。

对于带全称量词的 $if\text{-}then$ 语句，其逆否语句仍然成立，即它们逻辑等价。设

$D=\{x_1,\cdots,x_n\}$，则：

$$\begin{aligned}\forall x\in D,(P(x)\to Q(x)) &\equiv ((P(x_1)\to Q(x_1))\wedge\cdots\wedge(P(x_n)\to Q(x_n)))\\ &\equiv ((\neg Q(x_1)\to\neg P(x_1))\wedge\cdots\wedge(\neg Q(x_n)\to\neg P(x_n)))\\ &\equiv \forall x\in D,(\neg Q(x)\to\neg P(x))\end{aligned}$$

无论是全称量词，还是存在量词，其对应的变量都存在着自己的作用范围、作用域或辖域 (Scope)。例如，设 $D=\{Zhang,Luo,Li,Jia,Lu\}$，令 $Football(x)$ 表示“x 会踢足球”，则在语句中：

$$(\exists x\in D,Football(x))\to Football(Li)$$

个体变量 x 的作用范围已经被符号“$\to$”之间的括号所界定。该语句表达的含义为“如果 (D 中) 有人会踢足球，那么 Li 就会踢足球 (意即那个人肯定就是 Li)”。然而，如果将语句修改为：

$$\exists x\in D,(Football(x)\to Football(Li))$$

即修改了 x 的作用范围 (仍然由括号界定)，那么新语句的意义就会发生变化。例如，假设 x 表示是 Luo，而 Luo 并不会踢足球，则前提 $Football(Luo)$ 取值为 $False$，但是依据 $if\text{-}then$ 规则，$Football(x)\to Football(Li)$ 仍然将返回 $True$。显然，此语句并不符合我们所要传达的信息或表达的意思。

下面来讨论一下指导变量、自由出现、约束出现等相关概念。假设谓词语句形式为 $\forall x\,(P(x)\to\exists yQ(x,y))$，则在 $\exists yQ(x,y)$ 语句中，$\exists y$ 中的 y 被称为指导变量，y 的作用域为 $Q(x,y)$，$Q(x,y)$ 中的 y 为约束出现。此处，x 为自由出现。对于整个语句而言，$\forall x$ 中的 x 也是指导变量，x 的作用域为 $(P(x)\to\exists yQ(x,y))$，x 与 y 均为约束出现。x 约束出现 2 次，y 约束出现 1 次。与上述概念相关的规则为：

- 换名规则

  指的是将量词作用域内出现的指导变量及其约束出现，换名为不与其它变量相冲突的名称。例如，$\forall xP(x,y)\wedge Q(x,y)$，可以转换为 $\forall zP(z,y)\wedge Q(x,y)$；而 $\forall x\,(P(x,y)\wedge Q(x,y))$，则要转换为 $\forall z\,(P(z,y)\wedge Q(z,y))$。

- 替代规则

  指的是将自由出现的变量，替换为不与其它变量相冲突的名称。例如，$\forall xP(x,y)\wedge Q(x,y)$，可以转换为 $\forall xP(x,y)\wedge Q(z,y)$。

换名规则与替代规则的使用，在谓词逻辑归结的子句集求取的过程中是必不可少的。只有进行适当的换名和替代，才能正确地得到前束范式和 Skolem 标准型。

3.2.3 谓词公式

对于一阶逻辑，需要给谓词 WFF 下一个定义。我们把符合该定义的谓词语句称为谓词公式。

首先，给出原子谓词公式的定义：

1. 一个命题是原子谓词公式；
2. 一个命题变量是原子谓词公式；
3. 由 n 个个体变量 x_1、x_2、$\cdots$、x_n 及 n 元谓词所组成的谓词语句也是一个原子谓词公式；

定义 3.1（*WFF*）一个谓词语句可以被表示为具有良好格式的公式(*WFF*)，如果它通过以下规则构造而成，或符合以下规则：

1. 任何原子谓词公式为 *WFF*；
2. 如果 P 为 *WFF*，那么 $\neg P$ 也为 *WFF*；
3. 如果 P 与 Q 为 *WFF*，那么由它们通过二元逻辑运算符 $\wedge$、$\vee$、$\rightarrow$ 和 $\leftrightarrow$ 构造而成的 $P \wedge Q$、$P \vee Q$、$P \rightarrow Q$ 和 $P \leftrightarrow Q$ 均为 *WFF*；
4. 如果 P 为 *WFF*，x 为 P 中的个体变量，则 $\forall x, P(x)$、$\exists x, P(x)$ 也为 *WFF*；
5. 除非使用上述方法构造谓词语句，否则语句不是 *WFF*；

3.2.4 推理规则

除了命题逻辑中的推理规则之外，下面再新增几条适用于一阶逻辑的推理规则：

1. 全称实例化 (Universal Instantiation)

$$\frac{\forall x \in D, P(x)}{P(c), \forall c \in D}$$

其中，c 表示个体常量。该推理规则可直接应用于演绎三段论中。

2. 全称泛化 (Universal Generalization)

$$\frac{P(c), \forall c \in D}{\forall x \in D, P(x)}$$

其中，c 表示个体常量。例如，令 $P(x)$ 表示“x^2 是非负数”；若任取一常数 $c \in \mathbb{R}$，则 $P(c)$ 成立，那么可以将 $P(c)$ 泛化为 $\forall x \in \mathbb{R}, P(x)$。

3. 存在实例化 (Existential Instantiation)

$$\frac{\exists x \in D, P(x)}{P(c), c \in D}$$

其中，c 为符合条件的某个个体常量。

4. 存在泛化 (Existential Generalization)[29]

$$\frac{P(c), c \in D}{\exists x \in D, P(x)}$$

其中，c 为符合条件的某个个体常量。

3.2.5 证明方法

有效证明 (Valid Proof) 或有效论证 (Valid Argument)，指的是当所有的前提为真时，相应的结论也必须为真，或者说，其论证 *if-then* 命题是一个恒真式[30]。一般地，有如下 3 种主要的证明方法：

- 直接证明 (Direct Proof)
- 数学归纳法 (Mathematical Induction)
- 反证法 (Proof by Contradiction)

直接证明法直接从前提出发，应用各种推理方法得到结论。数学归纳法适用于形如 $\forall n, P(n)$ 的谓词公式的证明。其证明步骤可以划分为：

- 基本步：表明 $P(1)$ 成立；
- 归纳步：假设 $P(k)$ 成立，证明 $P(k+1)$ 成立；

[29] 虽然上面四条规则相当简单，但是如果需要实现形式化验证，则它们是必不可少的。
[30] 具体定义，请参见第2.5节。

数学归纳法为什么有效？首先，观察归纳步，它可以表示为对任一 k，$P(k) \to P(k+1)$；然后，对其应用全称实例化规则，得到 $P(1) \to P(2)$，再根据基本步可知，$P(1)$ 成立，因而 $P(2)$ 也成立；重复上面的步骤，可知命题序列 $P(3)$、$P(4)$、$\cdots$ 成立，再依据全称泛化规则，$\forall n, P(n)$ 成立。因此，数学归纳法的证明是有效的。

例 **3.1** 试用数学归纳法证明：

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \forall n \in \mathbb{N}$$

证明：令 $P(n)$ 表示 $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$，则上述结论可以表示为谓词公式 $\forall n \in \mathbb{N}, P(n)$[31]。

1. 基本步

$$P(1) = 1 = \frac{1(1+1)}{2}$$

2. 归纳步

假设对任一 $k \in \mathbb{N}$,$P(k)$ 成立，即 $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ 成立，则对于 $P(k+1)$：

$$\begin{aligned}\sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + (k+1) = \frac{k(k+1)}{2} + (k+1) \\ &= \frac{k(k+1) + 2(k+1)}{2} = \frac{(k+1)(k+2)}{2}\end{aligned}$$

上式表明，对任一 k，$P(k) \to P(k+1)$ 成立。因此，$\forall n \in \mathbb{N}, P(n)$ 成立。

□

使用数学归纳法进行证明的实例有很多，请阅读相关文献，在此不再赘述。

反证法也是一种应用非常广泛的非直接证明方法：为了从 $P(n) \to Q(n)$ 得到当 $P(n)$ 成立时 $Q(n)$ 也成立的结论，可以通过假设 $Q(n)$ 不成立，即 $\neg Q(n)$ 成立，继而得到一个矛盾式 (即恒假式)[32]，矛盾的产生就表明了原假设“$Q(n)$ 不成立”不成立，这样就间接地证明了 $Q(n)$ 成立的事实。其具体步骤如下：

[31] 可以将 $P(n)$(即 $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$) 中的“=”理解为“等词”，该谓词表明 2 个项实为指向同一个对象。

[32] 因此，反证法是基于矛盾律 (Law of Contradiction) 的。矛盾律表明，一个断言不可能既处于 $True$ 又处于 $False$ 的状态。对矛盾律取反并应用摩根定律，就可以得到排中律 (Law of Excluded Middle)。从这个角度而言，反证法也是基于排中律的。需要注意的是，一些逻辑系统并不接受排中律，其中最著名的是直觉逻辑，并因此拒绝把反证法作为一种可靠的证明方法。

1. 假设 $P(n) \to Q(n)$ 中的结论 $Q(n)$ 不成立，即 $\neg Q(n)$ 成立；

2. 从假设 $\neg Q(n)$ 出发，推导出矛盾式 $C \wedge \neg C$；

3. 因为矛盾式 $C \wedge \neg C$ 是恒假式 $\mathcal{C}$ 且它是由假设 $\neg Q(n)$ 出发并依据有效推理导出的，所以结论的错误或矛盾只能表明其前提假设 $\neg Q(n)$ 是不成立的，则 $Q(n)$ 一定成立[33]。

为了进一步表明反证法的有效性，下面我们从另一个角度来讨论上述推导过程及形成的结论。假设 $P(n)$ 取值为 $True$，现需要证明 $P(n) \to Q(n)$。对于反证法，将假设 $Q(n)$ 取值为 $False$，此时 $P(n) \to Q(n)$ 取值也为 $False$，则 $\neg (P(n) \to Q(n))$ 取值为 $True$。在随后的推导过程中，假设推导出了一个矛盾结论或矛盾式，即 $C \wedge \neg C$，则整个推理过程可表示为：

$$\neg (P(n) \to Q(n)) \to (C \wedge \neg C)$$

显然，上述断言是经过有效推理导出的，这意味着，要使该断言为真，必须要使 $\neg (P(n) \to Q(n))$ 为 $False$，即 $P(n) \to Q(n)$ 为 $True$，因 $P(n)$ 取值为 $True$，则 $Q(n)$ 必为 $True$，这相当于否定了 $Q(n)$ 为 $False$ 的假设。其真值表为：

$\neg (P(n) \to Q(n))$	$C \wedge \neg C$	$\neg (P(n) \to Q(n)) \to (C \wedge \neg C)$
F	F	T
T	F	F

实际上，下面的逻辑等价变换也表明了这一点：

$$\begin{aligned}(P(n) \to Q(n)) &\equiv (P(n) \to Q(n)) \vee (C \wedge \neg C) \\ &\equiv \neg (\neg (P(n) \to Q(n))) \vee (C \wedge \neg C) \\ &\equiv \neg (P(n) \to Q(n)) \to (C \wedge \neg C)\end{aligned}$$

在前面，虽然我们以断言 $P(n) \to Q(n)$ 为例引入了反证法，并对其有效性进行了证明，但是实际上，反证法适用于任何形式的断言——我们笼统地以 $Q(n)$ 来

[33]此处，再次应用了矛盾律：$Q(n)$ 只能处于 $True$ 或 $False$ 中的某一个状态。或者，从逆否命题的角度看，$\neg Q(n) \to (C \wedge \neg C) \equiv \neg (C \wedge \neg C) \to \neg (\neg Q(n)) \equiv True \to Q(n)$，则依据有效论证的定义，只能有 $Q(n)$ 取值为 $True$。

表示该断言[34]，只要由假设 $\neg Q(n)$ 出发，能够推导出一个矛盾式 $C \wedge \neg C$，就可以得出 $Q(n)$ 成立的结论。

下面，从前提 $P(n)$ 与结论 $Q(n)$ 关系的角度，比较了几种形式的证明方法：

- 直接证明

  由前提 $P(n)$ 出发，推导 $Q(n)$。

- 逆否证明 (Proof by Contrapositive)[35]

$$P(n) \to Q(n) \equiv \neg Q(n) \to \neg P(n)$$

  假设 $\neg Q(n)$，推导 $\neg P(n)$。

- 反证法

  欲证明任何形式的断言 $Q(n)$ 是否成立，可以先假设 $Q(n)$ 不成立，即 $\neg Q(n)$ 成立；然后，如果能够推导出矛盾式，则表明假设 $\neg Q(n)$ 不成立，即 $Q(n)$ 成立。这相当于：

$$\begin{aligned} Q(n) &\equiv Q(n) \vee (C \wedge \neg C) \\ &\equiv \neg (\neg Q(n)) \vee (C \wedge \neg C) \\ &\equiv \neg Q(n) \to (C \wedge \neg C) \end{aligned}$$

显然，反证法的适用范围比逆否证明更广。

例 **3.2** 试证明“如果 n^2 为偶数，则 n 必为偶数”[36]。

证明：假设 n 不是偶数，而是奇数，则令 $n = 2k + 1$，可得 $n^2 = (2k+1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$，该式表明 n^2 为奇数，而这与 n^2 为偶数的前提相矛盾。因此，假设不成立，即 n 一定是偶数。 □

例 **3.3** 试证明 $\sqrt{2}$ 是无理数。

[34]实际上，对于 $P(n) \to Q(n)$ 形式的断言，只要令 $P(n)$ 取值为 $False$，则前述的反证法证明步骤及有效性证明仍然有效。

[35]此种情况下，逆否证明等同于反证法。逆否证明是反证法的一种特殊情况。

[36]为简便，后面的反证法证明，没有严格地定义命题或谓词公式，但证明过程是明确且没有歧义的。

证明：假设 $\sqrt{2}$ 是有理数，则它具有 $\frac{b}{a}$ 的形式，其中 a、b 均为正整数且互质或互素 (Mutually Prime)，则：

$$\sqrt{2} = \frac{b}{a} \quad \Rightarrow \quad b^2 = 2a^2$$

显然，b^2 为偶数，而只有偶数的平方才是偶数，因而 b 也是偶数。于是，令 $b = 2c$，代入上式，得到 $a^2 = 2c^2$，因而 a^2 为偶数，则 a 也为偶数。因此，a 与 b 均为偶数，而这与 a 与 b 互质的假设相矛盾，这意味着假设不成立。因此，$\sqrt{2}$ 为无理数。□

4 练习

1. 使用命题逻辑和 Minisat 求解器，找出 8 皇后问题的所有解。

5 附录

5.1 Minisat 求解器的安装与运行示例

在 Windows 下，Minisat 求解器需要在 Cygwin 下运行[37]。步骤如下：

1. 从http://www.cygwin.com/下载 Cygwin 安装程序：64 位系统，下载 setup-x86_64.exe；32 位系统，下载 setup-x86.exe。为能顺利运行 Minisat，建议安装 32 位 Cygwin(64 位操作系统下也可以安装 32 位 Cygwin)；

2. 依操作系统的类型，选择一个安装程序运行；

3. 在弹出的对话框中，选择“Install from Internet”，点击“下一步”，选择一个 Cygwin 目标路径，其余操作依提示进行。一般情况下，缺省的镜像列表无法正常下载。此时，可以手动输入国内镜像，例如“http://mirrors.163.com/cygwin/”。下载及安装速度非常快[38]。依提示安装即可；

4. 从http://minisat.se/downloads/MiniSat_v1.14_cygwin下载 Cygwin 预编译二进制文件 MiniSat_v1.14_cygwin；

[37]当然，也可以从源程序编译出 Minisat.exe。本节只讨论 Cygwin 下的运行。

[38]已经下载的镜像安装文件，以后可以直接使用，执行 Cygwin 安装程序时，选择“Install from Local Directory”即可。

5. 将文件“MiniSat_v1.14_cygwin”复制到 Cygwin 根目录下的 bin 子目录，重命名为“Minisat.exe”；

6. 在系统环境变量下，为 PATH 添加路径，例如“D:/cygwin64/bin”；

7. 进入 CMD 命令行，输入 Minisat.exe，如果有提示信息输出，表明运行成功；

8. 也可以直接在“Cygwin Terminal”中运行 Minisat.exe，这种方式，不需要设置 PATH 环境变量；

9. 新建文本文件，重命名为“queens2.txt”，将下列内容输入到该文件中：

```
 c N=2 queens problem
 p cnf 4 12
 1  2 0
-1 -2 0
 3  4 0
-3 -4 0
 1  3 0
-1 -3 0
 2  4 0
-2 -4 0
 1  4 0
-1 -4 0
 2  3 0
-2 -3 0
```

进入 CMD 命令行或“Cygwin Terminal”，并进入“queens2.txt”文件所在目录，输入如下命令：

```
minisat queens2.txt out
```

将得到如下信息：

```
...
restarts              : 1
conflicts             : 2              (133 /sec)
decisions             : 1              (67 /sec)
propagations          : 4              (267 /sec)
conflict literals     : 1              (0.00 % deleted)
Memory used           : 5.31 MB
CPU time              : 0.015 s

UNSATISFIABLE
```

再输入如下命令：

```
cat out
```

将显示“UNSAT”信息，表明约束不满足。

6 参考文献

1. Stuart J. Russell, Peter Norvig, 殷建平等译。《人工智能：一种现代的方法》，第 3 版，清华大学出版社。
2. 马少平，朱小燕。《人工智能》，清华大学出版社，2004 年 8 月第 1 版。
3. 形式逻辑。https://wiki.mbalib.com/wiki/形式逻辑。
4. Propositional Logic. https://www.cs.ox.ac.uk/people/michael.wooldridge/teaching/soft-eng/lect07.pdf.
5. First-Order Logic. https://www.cs.ox.ac.uk/people/michael.wooldridge/teaching/soft-eng/lect10.pdf.
6. Propositional Logic. http://www1.spms.ntu.edu.sg/ frederique/dm2.pdf.
7. Predicate Logic. http://www1.spms.ntu.edu.sg/ frederique/dm3.pdf.
8. Propositional Logic. https://brilliant.org/wiki/propositional-logic/.
9. Predicate Logic. https://brilliant.org/wiki/predicate-logic/.
10. 归谬法. https://baike.so.com/doc/6299761-6513284.html.
11. Proof by Contradiction. https://en.wikipedia.org/wiki/Proof_by_contradiction.
12. Law of excluded middle. https://en.wikipedia.org/wiki/Law_of_excluded_middle.
13. Law of noncontradiction. https://en.wikipedia.org/wiki/Law_of_noncontradiction.
14. Notes on Chapter 7: Logical Agents and Propositional Satisfiability. http://www.sfu.ca/ tjd/310summer2019/chp7_proplogic.html.
15. Boolean satisfiability problem. https://en.wikipedia.org/wiki/Boolean_satisfiability_problem.
16. N Queens Problem (number of Solutions). http://www.ic-net.or.jp/home/takaken/e/queen/.

17. DPLL Algorithm. https://www.cs.miami.edu/home/geoff/Courses/CSC648-12S/Content/DPLL.shtml.