计算机解题逻辑

[英] R·科瓦尔斯基著
郑守淇 译

西安交通大学出版社

计 算 机 解 题 逻 辑

（英）R. 科瓦尔斯基 著

郑 守 淇 译

西 安 交 通 大 学 出 版 社

内 容 提 要

本书论述逻辑学在问题求解和计算机程序设计方面的应用。第一章、第二章讲述与机器无关的逻辑子句形式语义学以及子句形式在表示信息方面的应用。着重指出逻辑与其它大部分形式体系的不同。第三章至第八章叙述子句形式推理系统。第九章至第十三章探讨子句形式的扩充及更有效的解题方法。

本书是逻辑学、问题求解理论和计算机程序设计的一本入门书。可作为机器证明定理学科领域中的一本教科书。本书供高等学校学生使用，也可供从事人工智能、计算科学、机器证明定理等领域工作的科学技术人员参考。

Logic for Problem Solving
Robert Kowalski
1979 by Elsevier North Holland, Inc.
计算解题逻辑
(英) R. 科瓦尔斯基 著
[illegible]
责任编辑 魏廷德
*
西安交通大学出版社出版
《西安市咸宁路28号》
西安交通大学出版社印刷厂印装
陕西省新华书店发行　　各地新华书店经售
*
开本 787×1092　1/16　印张 13　字数：308千字
1986年11月第1版　　1986年11月第1次印刷
印数：1——4,000
统一书号：15340·103　定价：2.05 元

原　　序

本书探讨逻辑学在问题求解和计算机程序设计方面的应用。它假定读者先前没有这方面的知识，因而它可作为逻辑学、问题求解理论和计算机程序设计的一本入门书。

逻辑

逻辑是分析和表达论据的重要工具。它探讨假说是否蕴含结论的问题，但是与这些假说和结论的是否真实或谬误无关，也与它们的题材无关。本书的主要目的就是将传统的逻辑方法应用到现代的问题求解理论和计算机程序设计中去。

本书作为一本逻辑入门书，它与众不同之处在于使用了逻辑中的子句形式（clausal form）。这样做有若干优点：子句形式比逻辑中的标准形式简单，但同样有效；子句形式简单到足以直接讲授而无须象通常那样要预学命题逻辑；且与标准形式相比，它与数据处理和程序设计中所用的其它形式体系有更多相似之处。

本书不涉及逻辑数学但涉及它的应用。关于逻辑与语言关系的有益和更彻底的论述，建议读者参阅 Quine[1941]和 Hodges[1977]的书。

解题

在认知心理学(cognitive psychology) 和人工智能学科中发展起来的各种解题模型可以用逻辑子句形式来说明和比较。本书探讨启发式搜索、问题分解(reduction)和计算机解题的程序执行模型；并且证明逻辑推理能提供一个既较简单又较有效的模型。

将逻辑推理解释为问题求解，这是建筑在**自底向上推理**(即从假说出发正向推导出结论)和**自顶向下推理**（即从推理的目标出发逆向推导出若干子目标）之间的差异上的。推理的解题解释主要是自顶向下的解释。自底向上推理是解答的一般表达和证明方式，而自顶向下推理往往是解答的发现方式。自底向上推理是从旧信息得到新信息的**综合**过程；自顶向下推理是从推理目标得到子目标的**分析**过程。

本书涉及的范围大致与 Nilsson [1971]，Winston[1971] 和 Bundy [1978] 等书中问题求解章节的范围相似。但在上述各书中分别采用了产生式系统、LISP 和 LOGO 作为合一(unifying)的表达形式，而本书则采用了逻辑子句形式。

计算机程序设计

作为人和计算机通讯语言的逻辑是高级语言，它比其它专为计算机开发的表达形式更面向人类。常用的计算方法学对于表达程序、规范、数据库、查询和完整性条件等等分别采用不同形式；但逻辑则与此不同，对于所有这些任务只提供一个单一和一致的语言。我们将要探讨逻辑在数据库方面的应用，但是只限于逻辑作为一种程序设计语言的应用。

用通常程序设计语言表示的程序的含义是用程序在计算中所引起的动作来定义的，而用逻辑表示的程序的意义则能够用与机器无关而面向人类的术语来定义。因而逻辑程序较易编写，较易理解，较易改进，也较易适应于其它目的。

自顶向下的推理方法不仅给逻辑以解题的解释，而且也能用该方法在计算机上有效地执行逻辑程序。自顶向下推理使问题求解和计算机程序设计统一起来，而且还给执行智能程序提供许多方便，如：非确定性(non-determinism)、并行性和由模式匹配引起的过程调用等等。这些都是当今为常用程序语言所开发的新功能。另有一种有效的新程序设计语言，称为

PROLOG[Colmerauer等，1972],[Roussel 1975],[Bruynooghe 1976],[Warren, Pereira和Pereira 1977]和[Clark和McCabe 1979]，它以逻辑子句形式作为基础，并已在人工智能、数据库和工程中得到了应用。

机器证明定理

逻辑子句形式和它的关联推理系统的应用是以计算机自动证明定理的研究为基础的。Robinson的消解(resolution)规则[1965 a]和Loveland的模型消去法证明过程[1968,1969]是本书探讨的推理系统的主要历史先导，而Herbrand[1930]和Prawitz[1960]的更早研究工作又是上述推理方法的基础。

虽然本书中的推理方法是专为在计算机上应用而设计的，但该方法也能给人使用。为高效的机器证明定理所开发的解题策略，与计算机模拟人类解题方面研究人员所探讨的解题策略是相似的。尤其体现在本书作者曾经打算提出一种逻辑观点，它使面向机器的消解观点与Bledsoe[1971, 1977]及其同事们提出的启发式证明过程符合一致。

本书也可作为机器证明定理学科领域中的一本教科书，它同Chang与Lee[1973], Loveland[1978]和Robinson[1979]的书类似。但是它在形式上不甚严格，也不打算在该学科领域内作广泛详细的论述。

本书组成

本书由三部分组成。第一部分为第一章和第二章，它们讲述与机器无关的逻辑子句形式的语义学以及子句形式在表示信息方面的应用。第二部分为第三章至第八章，它们叙述子句形式的推理系统。第三部分为第九章至第十三章，它们探讨子句形式的扩充以及更为有效的解题方法。

本书第一部分着重指出逻辑与其它大部分形式体系的不同，即使不了解逻辑的动作方式也能理解该逻辑本身。书中举出了一些逻辑在描述程序和数据库方面的应用示例，并将子句形式同语义网络在表示自然语言的句子含义方面作了比较。

第二部分按照复杂程度分段介绍了子句形式的推理方法，其中第三章讲述Horn子句的推理方法。Horn子句是简化的句子，主要形式为

A if B1 and B2 and … and Bm.

(如果B1, B2,…,Bm都成立，则A成立。)

在第三章又介绍了自顶向下和自底向上的两种推理方法，它们被看成是关于上下文无关文法(context-free grammar)的自顶向下和自底向上的两种语法分析过程的一般推广。第四章讲述自顶向下推理的解题解释而第五章则讲述它的程序设计语言的解释。第六章描述Horn子句逻辑在编制规划问题上的应用。关于非Horn子句问题的推理方法和他们的解题解释则在第七章和第八章中探讨。

第九章讲述子句形式的全局解题方法；余下的各章探讨子句形式的各类推广。虽然子句形式和逻辑标准形式同样有效，但有时候它是欠自然的。在第十章中探讨逻辑标准形式以及它与子句形式的关系。在第十一章中单独处理了"当且仅当"(if-and-only-if)形式的定义。在第十二章中论述一种逻辑的扩充，它将句子的使用和说明联系起来，所用方式与自然语言中的方式相似。

最后一章讲述变化信息系统的运动学，对矛盾在决定变化方向上的作用给予特别注意。它将逻辑的解题解释与逻辑在分析人类知识信念中的经典运用联系了起来。

本书水平

本书是以1974年三月在阿姆斯特丹教学中心举行的计算机科学基础高级讲习班上所使用的讲稿 [Kowalski 1974 b] 为基础，再加以扩充而成的。在1973和1975年之间作者又用该材料在爱丁堡、米兰、罗马和斯德哥尔摩等地进行过短期讲学。1975年以来，本书的部分内容曾在英国帝国学院(Imperial College)对计算专业学生讲授逻辑与解题的入门课程中使用过。后在1978年锡腊丘兹(Syracuse)大学中曾讲授过本书的全部内容。

本书是在非形式层次上编写的，它几乎没有包含证明。本书不要求读者预先有逻辑学、问题求解和计算机科学方面的基础，因此它可能对于大学一年级的学生是合适的。然而在习题中有许多题目是水平较高的。此外，在第五章中将逻辑与通常程序设计语言作了对比，其中某些讨论对先前没有程序设计经验的读者来说可能是不易全部理解的。

致谢

书中许多材料曾从同事们的工作成果中得到启发，他们是：Keith Clark, Alain Colmerauer, Pat Hayes, Maarten van Emden 和 David Warren。我对他们表示衷心感谢。又感谢 Frank Brown, Alan Bundy, Tony Hoare, Wilfred Hodges, Chris Hogger, Jan Nilsson, George Pollard, Ray Reiter, Richard Waldinger 和 George Winterstein 对本书早期手稿所提出的宝贵意见。同时感谢 Karen King, Frank McCabe, Kevin Mitchell 和 Chris Moss 在制作照相排印稿上的帮助。再衷心感谢英国科学院(Science Research Council)对我工作的支持。

非常感谢我的妻子 Danusia 和孩子们 Dania, Tania 与 Janina 对我的关心和鼓励。

目　录

第一章 导 论

逻辑学研究假说和结论之间的蕴含关系。例如，逻辑学告诉我们下列假说：

“鲍勃喜欢逻辑。”和

“鲍勃喜欢任何喜欢逻辑的人。”

它们蕴含了结论：

“鲍勃喜欢他自己。”

但是并不蕴含结论：

“鲍勃只喜欢那些喜欢逻辑的人。”

逻辑学所涉及的不是个别句子的真实、谬误或可接受性，而是涉及它们之间的相互关系。假如一个结论被真实的或者可接受的假说所蕴含，那么逻辑学教我们接受这个结论。假如若干给定的假说蕴含了一个不可接受的或谬误的结论，那么逻辑学建议我们至少要放弃其中一个假说。因此，假如要否定“鲍勃喜欢他自己”的结论，那么就要在逻辑上被迫放弃“鲍勃喜欢逻辑”的假说或者放弃“鲍勃喜欢任何喜欢逻辑的人”的假说。

若要说明假说蕴含着一个结论，那么作出一个包含若干推理步的证明是有益的。为了使证明令人信服，各个推理步必须是直接而清楚的，并且它们应当在整体上互相适应。要达到这一目的，这类句子必须是无歧义的，而且使该类句子的文法尽可能简单则是有用的。要求证明中所使用的语言既是无歧义的又是文法上简单的，那就激励我们宁可采用一种符号语言而不采用一种象英语那样的自然语言。

在本书开始九章中，所使用的逻辑子句形式的符号语言是极其简单的。最简单的子句是**原子句**(atomic sentence)，它命名两个个体之间的关系，例如：

鲍勃**喜欢**逻辑。

约翰**喜欢**玛丽。

约翰**比**玛丽**大 2 岁**。

（句中**黑体**词是关系的名字，**非黑体词**是个体的名字。）更为复杂的句子则表达若干原子条件 (atomic conditions) 蕴含若干原子结论 (atomic conclusions) 的关系，例如：

如果约翰**喜欢**玛丽则玛丽**喜欢**约翰。

如果 x **喜欢**逻辑则鲍勃**喜欢** x 。

句中 x 是一个变量，它是任何个体名字。某些句子可以有若干联合条件(joint conditions)或若干不同结论(alternative conclusions)。例如：

如果玛丽**喜欢** x，则玛丽**喜欢**约翰或玛丽**喜欢**鲍勃。

（如果玛丽喜欢什么人的话，她就喜欢约翰或鲍勃）。

如果 x **是**鲍勃**的一个学生**，并且 x **喜欢**逻辑，则 x **喜欢**鲍勃。

句子也称为子句(clause)。一般而论，每个子句都表示若干联合条件（也可能没有）蕴含若干不同结论(也可能没有)。条件和结论表示个体之间的关系。个体可以是固定不变的，且用一些称为**常量符号**(constant symbols)（这样称呼也许有些混淆）的词，如：

Bob, John, logic 或 2

作为它们的名字；或者个体可以是任意变化的，且用一些**变量**(variables)，如：

u, v, w, x, y, z

作为它们的名字。应用**函数符号**(function symbols)可构成更复杂的名字，如：

dad(John)　　（即约翰的爸爸）

fraction(3,4)　　（即分数四分之三）

这些将在以后讲述。

上述逻辑子句形式的形式上不严格的概述将在本章下一节中再详细地说明并略加修改。但是子句形式与自然语言相比的极大简单性在此已非常明显。子句形式竟然具有许多自然语言的表现能力，这是令人叹赏的。在本书最后四章中，探讨子句形式的某些缺点并对克服它们的方法提出建议。

亲属关系示例和子句形式

作为子句中条件和结论的原子公式(atomic formula)，用简化形式表示，即使有些不太自然，也是方便的。关系的名字写在原子公式的前面，在它后面写上这关系所使用的一串个体名字。我们写 Father(Zeus,Ares)来代替写 Zeus **is father of** Ares（宙斯**是亚莱斯的父亲**），并写 Fairy-Princess(Harmonia)来代替写 Harmonia **is a fairy princess**（哈蒙尼娅**是一个仙女公主**）。在这里，严格地讲，"fairy-princess（仙女公主）"是个体的一个性质而非个体之间的关系。不过为了简化术语起见，当我们说到关系时，一并包括性质在内（称之为**谓词**）。而且为了使术语彻底混用起见，将关系的名字称为**谓词符号**(predicate symbol)。

我们使用符号"←"表示蕴含关系，读做"if（如果，当）"。例如：

Female(x)←Mother(x,y)

表示

x is female if x is mother of y

（如果 x 是 y 的母亲，则 x 是女性）。

为了使以后的记号表示法和推理规则简化起见，把所有子句都作为蕴含关系来对待是方便的，即使有些子句中没有条件或没有结论也仍然这样做。因此，把

Father(Zeus,Ares)

写成

Father(Zeus,Ares)←。

没有结论的蕴含关系是否定式，例如子句

←Female(Zeus)

表示 Zeus is not female（宙斯不是女性）。

下面的子句描述希腊神话中一些神的特点和亲属关系。

F1　Father(Zeus,Ares)←

F2　Mother(Hera,Ares)←

F3　Father(Ares,Harmonia)←

F4　Mother(Aphrodite,Harmonia)←

F5　Father(Cadmus,Semele)←

F6　Mother(Harmonia,Semele)←

F7　Father(Zeus,Dionysus)←

F8　Mother(Semele, Dionysus)←

F9　God(Zeus)←

F10　God(Hera)←

F11　God(Ares)←

F12　God(Aphrodite)←

F13　Fairy-Princess(Harmonia)←

上述子句所表示的意义是显而易见的。下列子句对这些意义加上约束，从而使意义更加明确。

F14　Female(x)←Mother(x,y)

F15　Male(x)←Father(x,y)

F16　Parent(x,y)←Mother(x,y)

F17　Parent(x,y)←Father(x,y)

上述子句说明，对于所有的x和y：

如果x是y的母亲则x是女性，

如果x是y的父亲则x是男性，

如果x是y的母亲则x是y的双亲之一，

如果x是y的父亲则x是y的双亲之一。

即使有些变量具有相同的名字，但它们在不同子句中的意义是不同的。例如，子句F14中的变量x与子句F15中的变量x是没有联系的。一个变量的名字只有在它出现的子句的上下文中才有意义。如果两个子句仅仅在所包含的变量的名字上不同，那么他们是等价的，并且一个子句被称为另一子句的**变体**(variant)。

在子句形式中，一个子句的全部条件是合取的（即它们由联接词“与(and)”联接），然而它的全部结论是析取的（即它们由“或(or)”联接）。因此，联接词“与”和“或”都可安全地被逗号所代替。在两个条件之间的逗号读成“与”而两个结论之间的逗号读成“或”。于是可写成

F18　Grandparent(x,y)←Parent(x,z), parent(z, y)

F19　Male(x), Female(x)←Human(x)

句中x，y和z是变量；对于所有的x，y和z，它们表示：

如果x是z的双亲之一“与” z是y的双亲之一，则x是y的祖父母之一，

如果x是人，则x是男性“或” x是女性。

如果若干结论为同样条件所蕴含，那么对每一结论须有一个单独的子句。类似地，如果同一结论为若干不同条件所蕴含，那么对于每一条件须有一个单独的子句。例如，下列能直接表示成逻辑标准形式（定义见第十章）的句子：

Female(x) **and** Parent(x,y)
←Mother(x,y)

可以等价地用下列两个子句表示：

Female(x)←Mother(x,y)

Parent(x,y)←Mother(x,y).

这两个子句是暗中被“与”联结着的，即：如果x是y的母亲则x是女性**与**如果x是y的**母**亲则x是y的双亲之一。

与此相似，句子

Parent(x,y)←Mother(x,y) **or** Father(x,y)

可用下列子句表示：

Parent(x,y)←Mother(x,y)

Parent(x,y)←Father(x,y)

（如果x是y的母亲则x是y的双亲之一**与**

如果x是y的父亲则x是y的双亲之一。）

谓词符号可用于命名两个以上的个体间的关系。例如，原子公式

Parents(x,y,z)

能用于表达　　x是z的父亲与y是z的母亲

即　　Parent(x,y,z)←Father(x,z),Mather(y,z)

子句形式的更精确定义

本节将更精确地定义子句形式的语法（文法）并且同时指明相应的解释。

子句是具有下列形式的表达式：

$$B_1,\cdots,B_m\leftarrow A_1,\cdots,A_n$$

式中$B_1,\cdots,B_m,A_1,\cdots,A_n$是原子公式，且$n\geqslant 0$和$m\geqslant 0$，原子公式$A_1,\cdots,A_n$是子句的联合条件而$B_1,\cdots,B_m$是不同结论。如果子句中包含变量$x_1,\cdots,x_k$，则解释为

对于所有$x_1,\cdots x_k$，

如果A_1与…与A_n成立则B_1或…或B_m成立。

如$n=0$则解释为：

对于所有$x_1,\cdots,x_k$，

B_1或…或B_m无条件地成立。

如$m=0$则解释为：

对于所有$x_1,\cdots,x_k$，

A_1与…与A_n不成立。

如$m=n=0$则写成□，并解释为永远谬误的句子。

原子（或称**原子公式**，atomic formula）是下列形式的表达式：

$$P(t_1,\cdots,t_m)$$

式中P是一个m元谓词符号，$t_1,\cdots,t_m$是项(term)，且$m\geqslant 1$。上述原子被解释为：在个体$t_1,\cdots,t_m$之间存在一个称为P的关系。

项是一个变量，或一个常量符号，或一个具有下列形式的表达式：

$$f(t_1,\cdots,t_m)$$

式中f是m元函数符号，$t_1,\cdots,t_m$是项，且$m\geqslant 1$。

谓词符号、函数符号、常量符号和变量等集合是互不相交的集合。习惯上，把下列带有修饰或不带修饰的小写体字母：

u,v,w,x,y,z

作为变量的符号。其它各种符号类型可以从它们在子句中所占的位置来识别。

子句形式中箭头符号“←”所指的方向与通常标准逻辑形式中的方向相反。本书中写成

B←A　　（B成立，如果A成立）

的地方，通常写成

A→B　　（如果A成立，则B成立）

两者的差别仅仅是表面上的。在本书中所以记作“B←A”，其目的是引起大家对子句结论的注意。

谓词符号或函数符号中各个变元，称为谓词或函数的自变量（arguments）。在原子 $P(t_1,\cdots,t_n)$ 中的第一个自变量是 t_1，而最后一个自变量是 t_n。

要使数目有限的子句能涉及数目无限的个体，需要复合项(composite term)。例如，非负整数可用下列各项表示：

0，s(0)，s(s(0))，…，s(s(…s(0)…))，…式中0是常量符号，s是一元函数符号（s代表“后继元(successor)”）。以项s(t)命名的数要比以项t命名的数增加1，即在整数序列中s(t)是t的**后继元**。

下列子句

Num1　　Num(0)←

Num2　　Num(s(x))←Num(x)

说明：

0是一个数，且

如果x是一个数，则s(x)也是数。

自顶向下和自底向上的定义表示方式

在上节中已用自顶向下方式给出了子句形式的定义。在其中第一条定义中，用原子概念说明子句概念，即目标概念（goal concept）（虽然那时原子公式还没有被定义）。然后原子公式成为新的目标概念，它在下一条定义中被分析成两个子目标概念，即谓词符号和项。项的概念是递归定义的；最终它被分析为常量符号、变量和函数符号的概念。因此，原来的原子公式概念最后被分析成四个概念——谓词符号、常量符号、变量和函数符号。这些符号的具体对象是什么，那是无关紧要的，只要它们能互相区别并且不与“保留符号”混淆就行了。保留符号为

←，（和）。

自顶向下的定义表达方式具有目标始终明确的优点。其缺点是当子目标概念尚未定义时就根据子目概念定义目标概念，因而使人感到它们突然出现而不能完全理解。

自底向上的定义表达方式正与上述相反。它从一些没有定义的概念出发，这些概念是“原始的”，它不能被定义，或者是早已为人们所熟知而无须定义的。然后用这些已知的概念给新概念下定义。最后，当目标概念被定义之后，定义过程便告结束。在定义过程中，每条定义一经给出之后就能立刻被人理解，但是在所有定义完成之前，如此做法的动机是不能被人

们了解的。

关于自顶向下和自底向上之间差别的解释，不但适用于定义的表达，而且也适用于证明的表达和发现，以及计算机程序的编写过程。证明过程能用传统的，自底向上的数学方式表达；即从给定的条件出发正向推理，根据已知结论推导出新的结论，并且一经推导出目标之后，推理过程即行结束。另一种方法，证明过程能用一种反映它们被发现过程的自顶向下方式说明；即从目标出发逆向推理，将目标化成子目标，并且当所有子目标都被认为是可解的时候，证明即行结束。

计算机程序也能用自底向上方式编写，即从计算机已能理解的原始程序出发，利用旧程序编写新程序。在每一阶段，程序能在计算机上执行和测试。如果已写好的低层次程序不能组装成合适的高层次程序，那么它们必须被重写。经验告诉我们，用自顶向下的方式编写程序比较好，即首先根据未曾写好的低层次程序来编写最高层次的程序。将低层次程序留在以后编写而保证它们组装合适。在以后，低层次程序还能够修改和改进，并不影响程序的其余部分。

自顶向下和自底向上推理之间的区别，以及采用逻辑符号表示信息所得的利益是本书的主要议题之一。它也是分析（自顶向下）和综合（自底向上）之间的区别，或者是目的论（自顶向下）和决定论（自底向上）之间的区别。再者在运用上，自顶向下推理方式优先于自底向上推理方式，这样使推理的经典逻辑学观点（按照**应该执行**的方式推理）和推理的心理学观点（按照人们在实践中**实际**执行的方式推理）得到协调一致。

自顶向下的推理方式使得将目标化为子目标的人类解题策略，与计算机程序中以过程体代换被调用过程的执行方法联系起来。它又使逻辑学的研究与人类解题方法的研究以及计算机程序设计的研究统一起来。

子句形式的语义学

语法(syntax)论述句子的文法(grammer)。在历史上，语法也论述推理规则和证明方法。在另一方面，语义学(semantics)论述句子的含义。将子句译成英语只是语义学的形式上不严格的入门向导。

我们用自然语言随意说出具有含义的词和句子，但用符号逻辑则要较为谨慎。与某个谓词符号、常量符号、函数符号或与句子有关的任何含义都是相对于一个表达全部相关假说的句子集合而言的。在亲属关系的示例中，如果 F1～19 表示全部假说，那么就不能排除下列断言：

F　　Mother(Zeus,Ares)←

是成立的。这一可能性是与上述假说 F1～19 一致的，F1～19 单独确定与下列符号

"Mother"，"Father"，"Zeus"，等等

有关的任何含义。为了排除F成立的可能性，需要一个额外的假说，如

F20　　←Male(x)，Female(x)。

F和 F1～19 是一致的，但和 F1～20 相矛盾。

如果给定一个子句集合，它表示与某一问题的定义域有关的全部假说，那么为了了解一个个别符号或子句的含义，就必须先明确这些假说在逻辑上所蕴含的内容。一个谓词符号，例如 "Mother" 所含的意义只能从那些包含该谓词符号并在逻辑上被假说所蕴含的全部句子集合中认出。因此在 F1～20 中"Mother"的含义中包括否定式：