第一章　逻辑程序的基础知识

1.1　逻辑程序的发展历史

逻辑程序设计(Logic Programming)始于70年代早期。它是早期自动定理证明和人工智能发展的自然结果。1930年,Herbrand为定理机器证明奠定了理论基础[1]。在60年代早期,自动定理证明的研究工作十分活跃,如Prawitz[2],Gilmore[3],Davis,Putnam[4]等的工作。1965年,J. A. Robinson提出了归结原理[5],它为计算机提供了机械证明的方法。

1972年Kowalski[6]和Colmerauer[7]提出了逻辑可以作为程序设计语言的基本思想,把逻辑和程序这两个截然不同的概念协调统一为一个概念——逻辑程序设计。这个思想产生了深远而持久的影响。1972年,Colmerauer和他的研究小组在马赛(Marseille)大学用ALGOL-W实现了第一个逻辑程序设计语言Prolog[8]。紧接着又用FORTRAN对原系统进行了改进[9],从而使逻辑程序迈向了可用阶段。早期的逻辑程序系统采用解释的方法,因而执行速度极慢,一度遭到了人们的非议。80年代初期,D. H. D Warren提出Warren抽象机模型WAM[10],采用编译的方法大大提高了逻辑程序的时空效率。与最初的解释方法相比,编译方法使Prolog的执行速度提高了一个数量级之多。从而逻辑程序迈向了实用阶段。

当今计算机技术迅猛发展。它已经历了数值计算,数据处理和信息处理阶段,而今正迈向知识信息处理阶段。未来信息处理发展的基本方向是知识信息处理[11]。基于一阶谓词逻辑的逻辑程序设计语言,将逻辑推理对应于计算,它的丰富表达能力、非确定性等特点,适合于表达人工智能和知识工程中的问题,从而使逻辑程序越来越得到广泛的应用。80年代逻辑程序设计受到了世界许多国家高技术计划的青睐。许多国家提出了研究新一代的计算机系统的计划,以提高推理速度。例如,1981年秋,日本提出了第五代计算机系统研究计划。逻辑程序不仅用作为应用编程语言,而且也用作为系统编程语言。1984年,日本生产出第一台个人顺序推理机,并首次用逻辑程序实现了个人顺序推理机的操作系统SIMPOS。

80年代初期起,逻辑程序并行处理技术的研究成为一个热点。在并行逻辑程序语言方面:Clark和Gregory提出了关系语言和Parlog[12][13]。Shapiro提出了Concurrent Prolog[14],Ueda提出了GHC[15],R. Yang和H. Aiso提出了P-Prolog[16][17],L. M. Péreira和R. Nasr提出了Delta-Prolog[18][19]。Haridi提出了Andorra Prolog[20],Wise提出了EPILOG[21][22]等等。在逻辑程序并行执行模型和体系结构方面,代表性的工作有:Pollard首创的并行执行模型[23]。Conery的AND/OR进程模型[24][25],DeGroot的RAP模型[26][27],Ciepielewski的TOKEN模型[28],Goto的G-R模型[29],孙成政的PSOF模型[30],Tung的并行模型[31],Hermenegildo的RAP-WAM抽象机模型[32][33][34],D. H. D Warren的SRI抽象机模型[35] Hausman等的VV-WAM抽象机模型[36],以及其它一些模型[37][38][39][40][41]。

1.2 逻辑程序的理论基础

1.2.1 一阶逻辑

一阶谓词逻辑提供了一种形式方法来表示某个域上的前提和结论，并处理它们之间的蕴含关系。

1. 语法

一阶逻辑包含两个方面：语法和语义。一阶逻辑的语法涉及到形式语法所认可的合式公式及证明论。它的语义涉及到与合式公式中符号相关的含义。

一阶理论由字母表、一阶语言、一个公理集合(简称一集公理)和一个推理规则集合(简称一集推理规则)组成。

下面我们将定义字母表和一阶语言。

定义 1.1 字母表由下列六类符号组成：变量符号、函数符号、谓词符号、连结词、量词和标点符号。变量符号是以大写字母开头的有限字母数字串。n 元函数符号和 n 元谓词符号是以小写字母开头的有限字母数字串(可带下标)。0 元函数符号称之为常量。连结词由～、∧、∨、→和↔组成。量词包括∃ 和∀ 。标点符号由“(”,“)”和“,”组成。

定义 1.2 一个项(term)归纳定义如下：

(1) 变量是项

(2) 常量是项

(3) 如 f 是一个 n 元函数，$t_1,\cdots,t_n$ 是项，则 $f(t_1,\cdots,t_n)$是项。

定义 1.3 一个合式的公式(或简称公式)归纳定义如下：

(1) 如果 P 是一个 n 元谓词，$t_1,\cdots,t_n$ 是项，那么

$P(t_1,\cdots,t_n)$是一个合式公式(或称原子公式)。

(2) 如 F 和 G 是合式公式，那么(～F)，(F∧G)，(F∨G)，(F→G)和(F↔G)也是合式公式。

(3) 如果 F 是合式公式，X 是一个变量，那么(∀ x F)和(∀ x F)也是合式公式。

定义 1.4 一阶语言是由字母表中的符号构造的合式公式集。

量词和连结词的非形式语义为：～是否定，∧是合取(and)，∨是析取(or)，→是蕴含，↔是等价。∃ 是存在量词，∃ X 意为“存在一个 X”，∀ 是全称量词，∀ X 意为“所有的 X”。

∀x((p(x,g(x)))←q(x)∧～r(x))的非形式语义为“对于所有的 x，如果 q(x)为真且 r(x)是假，那么 p(x,g(x))为真”。

定义 1.5 公式∀ xF 中∀ x(或公式∃ xF 中∃ x)的作用域是 F。如果一个变量在量词的作用域中出现，则称此变量是约束出现；否则称此变量是自由出现。

例如：公式(∃ x)(x≠f(a)↔～r(x,f(a)))中∃ x 的作用域是公式(x≠f(a)↔～r(x,f(a)))。

定义 1.6 无自由变量出现的公式称为闭公式。

定义 1.7 如 F 是一个公式，对于 F 中的每一个自由出现的变量，加一个全称量词得到∀(F)，它称为 F 的全称闭包。类似地对于 F 中的每一个自由出现的变量，加一个存在量

词得到∃(F),它称为F的存在闭包。

2. 解释和模型

当我们讨论公式的真假时,必须首先赋予公式中每一个符号某种含义。量词和连结词有固定的含义,但常量、函数和谓词的含义可以变化。

定义 1.8 一阶语言L的解释由下列组成.

(1) 一个非空集D,称为解释域。

(2) 对于L中的每一个常量,赋予D中的一个元素。

(3) 对于L中的每一个n元函数,赋予D^n到D的一个映射。

(4) 对于L中的每一个n元谓词,赋予D^n到{true,false}的一个映射。

定义 1.9 设L是一阶语言L的解释,将L中的每一个变量赋予I中的一个元素,这样的赋值称为关于I的变量赋值。

定义 1.10 设I是一阶语言D域上的一个解释。A是一个变量赋值。L中关于I的项赋值定义如下:

(1) 根据A赋值每一个变量。

(2) 根据I赋值常数。

(3) 如$t_1',\cdots,t_n'$是$t_1,\cdots,t_n$的项赋值,f'是f的赋值,则$f'(t_1',\cdots,t_n')\in D$是$f(t_1,\cdots,t_n)$的项赋值。

定义 1.11 设I是一阶语言D域上的一个解释,A是一个变量赋值,那么,L中公式关于I和A的真值给定如下:

(1) 原子公式$P(t_1,\cdots,t_n)$的真值通过求值$P'(t_1',\cdots,t_n')$得到。其中P'是根据I赋予P的映射,$t_1',\cdots,t_n'$是关于I和A的项赋值。

(2) 公式~F,F∧G,F∨G,F→G,F↔G的真值由下表给出。

F	G	~F	F∧G	F∨G	F→G	F↔G
true	true	false	true	true	true	true
true	false	false	false	true	false	false
false	true	true	false	true	true	false
false	false	true	false	false	true	true

(3) 如果存在d∈D使得F关于I和A(x/d)的真值为真,则公式∃ xF的真值为真,否则为假。其中A(x/d)与A的不同仅在于A(x/d)中,x赋予d。

(4) 如果对于所有d∈D,F关于I和A(x/d)的真值为真,则公式∀ xF的真值为真,否则为假。

定义 1.12 设I是一阶语言L的解释,F是L的闭公式,如果F关于I的真值为真,则I称为F的模型。

一阶理论的公理是该理论的语言中指定的闭公式子集。

定义 1.13 设T是一阶理论,L是T的语言,如L的一个解释是T中每一个公理的模型,则称此解释为T的模型。

类似地,此概念可推广到一集闭公式。

定义 1.14 如果S是一阶语言L的闭公式,I是L的解释,如果I是S中每一个公式的模型,则称I是S的模型。

定义 1.15 设S是一阶语言L的闭公式集,如L有一个解释,它是S的模型,则称S是可满足的。如L的每一个解释是S的模型,称S是有效的。如S没有模型,则称S是不可满足的。

3. 逻辑结论

定义 1.16 设S是一阶语言的闭公式集,F是一个闭公式。如果对于L的每一个解释I,I是S的模型,则意味着I是F的模型,那么F称为S的逻辑结论。

命题 1.1 设S是一阶语言的闭公式集,F是一个闭公式。F是S的逻辑结论当且仅当$S\cup\{\sim F\}$是不可满足的。

定义 1.17 不含变量的项称为基础项(ground term);不含变量的原子称为基础原子(ground atom)。

定义 1.18 设L是一阶语言,L的Herbrand通域U_L是由L中的常量和函数构造的所有基础项组成。

定义 1.19 设L是一阶语言,L中满足下列条件的解释称为Herbrand解释:

(1) 解释域为Herbrand通域U_L;

(2) L中常量赋予U_L中自身常量;

(3) 如f是L中的n元函数,f赋予$(U_L)^n$到U_L的映射。

定义 1.20 设L是一阶语言,S是L的闭公式集,如L的Herbrand解释是S的一个模型,则称它为S的Herbrand模型。

命题 1.2 设S是一集公式

(1) 如S有一个模型,则S有一个Herbrand模型;

(2) S不可满足,当且仅当S没有Herbrand模型。

4. 合一(unifier)

定义 1.21 一个置换是形如$\{V_1/t_1,\cdots,V_n/t_n\}$的有穷集。其中$V_i$为变量,$t_i$是不同于$V_i$的项,$V_i,\cdots,V_n$是不同的变量。每一个元素$V_i/t_i$称为$V_i$的一个约束。

定义 1.22 设S是形如$\{E_1,\cdots,E_n\}$的有穷简单表达式集,如果置换θ使得$E_1\theta=E_2\theta=\cdots=E_n\theta$,则$\theta$为S的合一。如果对于S的每一个合一$\sigma$,存在一个置换$\gamma$使得$\sigma=\theta\gamma$,则$\theta$是S的最一般的合一(most general unifier,mgu)。

1.2.2 Horn 子句和逻辑程序

1. 逻辑程序的语法

定义 1.23 一个文字或者是一个正原子,或者是一个负原子。

定义 1.24 一个子句是形式为$\forall x_1,\cdots,\forall x_s(L_1\vee,\cdots,\vee L_m)$的公式,其中每一个$L_i$是一个文字,$x_1,\cdots,x_s$是$L_1\vee,\cdots,\vee L_m$中出现的所有变量。

上述子句$\forall x_1,\cdots,\forall x_s(L_1\vee,\cdots,\vee L_m)$可等价变换为:

$$\forall x_1,\cdots,\forall x_s(A_1\vee,\cdots,\vee A_m\vee\sim B_1\vee,\cdots,\vee\sim B_n)$$

其中,$A_1,\cdots,A_m,B_1,\cdots,B_n$为原子,$x_1,\cdots,x_s$为这些原子中出现的所有变量。我们将上述

形式等价地写成：

$\forall x_1,\cdots,\forall x_s(A_1\vee,\cdots,\vee A_m\leftarrow B_1\wedge,\cdots,\wedge B_n)$

因此，一个子句可简写为：

$A_1\vee,\cdots,\vee A_m\leftarrow B_1\wedge,\cdots,\wedge B_n$

定义 1.25 一个程序子句是形式为 $A\leftarrow B_1,\cdots,B_n$ 的子句。A 称为程序子句头，$B_1,\cdots,B_n$ 称为程序子句体。

定义 1.26 一个单位子句是形式为 $A\leftarrow$ 的子句。即程序子句体为空的程序子句。

定义 1.27 一个逻辑程序是一个有限的程序子句集。

定义 1.28 一个目标子句具有如下形式：

$\leftarrow B_1,\cdots,B_n$

即，它是一个子句头为空的子句，其中每个 $B_i(i=1,\cdots,n)$ 被称作该目标子句的子目标。

定义 1.29 空子句是子句头和子句体均为空的子句，用□表示。

定义 1.30 一个 Horn 子句或者是一个程序子句，或者是一个目标子句。

可见，Horn 子句集是一阶逻辑公式的子集。根据定义，Horn 子句有以下三种形式：

$A\leftarrow B_1\wedge,\cdots,\wedge B_n$ (1)

$A\leftarrow$ (2)

$\leftarrow B_1\wedge,\cdots,\wedge B_n$ (3)

通常，我们将(1)称为规则，(2)称为事实，而(3)称为询问。

2. 逻辑程序的语义

(1) 过程语义

定义 1.31 一个计算规则 CR(Computation Rule)是从给定子句体中选择特定目标的函数。

定义 1.32 一个搜索策略 SS(Search Strategy)是从程序中选择匹配子句的策略。一个计算状态可用序偶$\langle G,\theta\rangle$表示，G 是目标的合取，θ 是最一般合一取代。初始计算状态$\langle IG,\Phi\rangle$，IG 为初始目标，Φ 为空取代。

① SLD 归结

R. A. Kowalski 和 Van Emden 等提出的 SLD(Linear resolution with Selection function for Definite clause)归结，是对 Robinson 归结原理的精化。

定义 1.33 SLD 归结理论

设 P 是程序，IG 是初始目标，CR 是计算规则，SS 是搜索策略。PU{G}的推理过程由一系列计算状态$\langle IG,\Phi\rangle$，$\langle G_1,\theta_1\rangle$，… 和一系列子句 $C_1,C_2,\cdots$ 组成，每个 G_{i+1}是通过 CR 和 SS 由 G_i 和 C_{i+1}使用最一般合一取代 θ_{i+1}导出，这里假设：

G_i 形式为$\leftarrow A_1,\cdots,A_m,\cdots,A_k$

C_{i+1}形式为 $A\leftarrow B_1,\cdots,B_q$

(a) A_m 是根据计算规则 CR 选择的原子，

C_{i+1}是根据搜索策略 SS 选择的子句。

(b) $A_m\theta_{i+1}=A\theta_{i+1}$

(c) 得到的新的计算状态为$\langle G_{i+1},\theta_{i+1}\rangle$，其中 G_{i+1}形式为

$\leftarrow(A_1,\cdots,A_{m-1},B_1,\cdots,B_q,A_{m+1},\cdots,A_k)\theta_{i+1}$

当 k=1,q=0 时,则下一个计算状态为〈true,θ〉,则 θ 被看成 p 的一个解。

SLD 归结中的计算规则,每次仅选择一个原子。我们可把计算规则分成两类,深度优先(depth-first)规则和联立规则(coroutining rule)。深度优先规则总是从引入的原子序列 $B_1,\cdots,B_q$ 中选择一个原子。它的一个特例是最左调用规则,即每次总是从引入的 $B_1,\cdots,B_q$ 中选择最左原子 B_1,如 q=0,则选择下一个原子。联立规则并不总是从 $B_1,\cdots,B_q$ 中选择,它可交替从引入的原子序列 $B_1,\cdots,B_q$ 和原来的原子序列 $A_1,\cdots,A_{m-1},A_{m+1},\cdots,A_k$ 中选择。

② SLD 归结原理的变形和扩充

根据计算规则和搜索策略的变化,得到 SLD 归结的变形或扩充。

Naish 提出了 HSLD(Heterogeneous SLD)[42]。设 P 为逻辑程序,计算状态同样用〈G,θ〉表示,与 SLD 归结不同,这里目标 G 的形式为:

$\leftarrow A_1\{C|C\in P,C\text{ 和 }A_1\text{ 可合一}\},\cdots,A_m\{C|C\in P,C\text{ 和 }A_m\text{ 可合一}\}$

即目标中的每个原子对应一个与它可匹配的子句集。

计算规则属联立规则,它选择一系列原子,子句对,且至少有一个子句集中的所有子句全被选择。

Clark[43]提出了 SLDNF(SLD with the Negation as Failure rule)归结。SLDNF 归结允许子句体中出现负文字。即程序子句形式为:

$A\leftarrow L_1,\cdots,L_k$

其中 A 为正文字,$L_i(1\leqslant i\leqslant k)$是正文字或负文字。

计算状态用〈G,θ〉表示,G 形式为$\leftarrow L_1,\cdots,L_m,\cdots,L_k$

当计算规则选择一个文字 L_m 为正文字时,则与 SLD 归结相同;当计算规则选择一个文字 L_m 为负文字$\sim A_m$ 时,且 A_m 为基础的(这样的计算规则称为安全的),如 A_m 的每条计算路径都失败,则$\sim A_m$ 成功,新的计算状态为〈G′,θ′〉,G′ 为:

$\leftarrow L_1,\cdots,L_{m-1},L_{m+1},\cdots,L_k$

Wolfram 等[44]提出了 GLD 归结。设 P 是程序,计算状态为〈G,θ〉,G 是多个原子集合,计算规则从 G 中选择一个原子集$\{B_1,\cdots,B_n\}$,$(n\geqslant 1)$,如果$\{A_1\leftarrow G_1,\cdots,A_n\leftarrow G_n\}\subseteq P$且它与 G 无相同变量,$B_i$ 和 A_i 有相同谓词,则下一个计算状态为

〈$G\cup G_1\cdots\cup G_n,\theta'$〉

在逻辑程序 P 中,n 元谓词 p 的指称是定义在 Herbrand 通域上的 n 元关系,由过程语义确定的程序 P 中 p 的指称为:

$D_1(p)=\{(t_1,\cdots,t_n)|$给定逻辑程序 P 中的子句作为公理,谓词 $p(t_1,\cdots,t_n)$通过归结可证明$\}$。

逻辑程序的实现通过构造性证明产生 D_1 的 n 元序偶。

(2) 模型论语义

在模型论语义中,逻辑程序 P 中谓词 P 的指称定义为:

$D_2(p)=\{(t_1,\ldots,t_n)|$P 蕴含 $p(t_1,\cdots,t_n)\}$

(3) 不动点语义

首先我们定义与逻辑程序 P 对应的映射 T_p，它将一个 Herbrand 解释映射到另一个 Herbrand 解释：

$T_p(I)=\{A\in B_p \mid A\leftarrow B_1,\cdots,B_m$ 是程序 P 中子句的基础例示(ground instance)，且 $\{B_1,\cdots,B_m\}\subseteq I\}$，

其中 B_p 为以 H_p(P 的 Herbrand 通域)中元素作为变元的 P 中所有谓词形成的基础原子集合。

则 P 中谓词 P 的指称为：

$D_3(P)=\{(t_1,\cdots,t_n)\mid P(t_1,\cdots,t_n)$属于 T_p 的最小不动点 $lfp(T_p)\}$

定理 1.1 $D_1(p)=D_2(p)=D_3(p)$(参见[45])。

1.3 逻辑程序设计范型与其它程序设计范型的比较

程序设计范型(paradigm)指设计和实现程序的模型。不同的模型导致不同的程序设计技术和方法学。尽管至今已有上百种程序设计语言，但它们可归类为几种程序设计范型(见图 1.1)。例如，过程型程序设计，函数型程序设计，逻辑程序设计和面向对象程序设计范型。不同程序设计范型的着重点不同，但同一范型的程序设计语言存在许多共性之处。

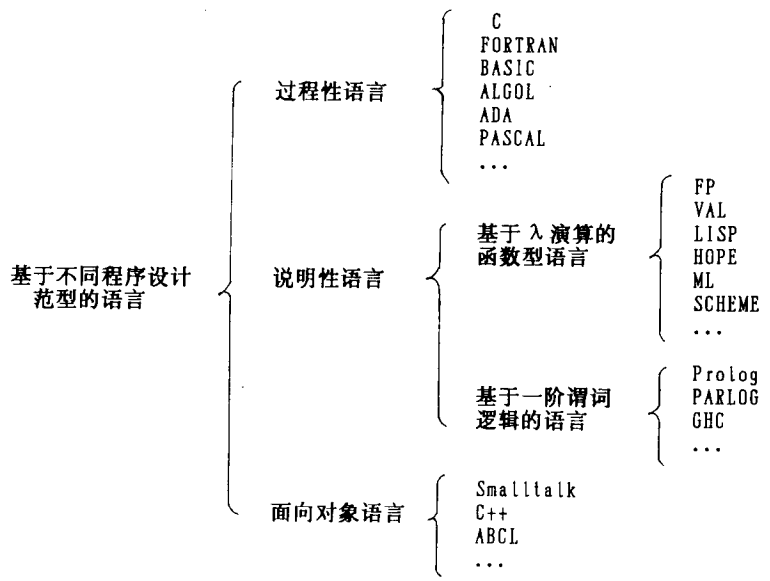

图 1.1 不同范型的程序设计语言

在基于过程性范型的传统程序设计语言中，程序是由传递参数的过程或函数的集合组成 。每个过程或函数处理它的参数，有时更新它们并有可能返回一个值。即过程或函数实现了计算抽象。传统的过程性语言以过程或函数为中心，仔细分析传统过程型程序设计范型，我们不难看出它具有如下的特点：

(1) 按照程序事先显式确定的顺序执行计算。

(2) 每个语句是算法中的一步。执行一个语句产生付作用(side-effect，指不能通过逆

运算消除的操作)。

(3) 程序的执行最终产生出所要的状态。

(4) 必须通过追踪程序的执行路径和相应的状态变化来理解程序。必须访问机器的运行状态才能决定语句的正确性。

(5) 程序常常依赖机器。

随着软件技术的发展,出现了模块化程序设计,从过程为中心转到的数据为中心,从而增强了程序的可读性和可维护性。80 年代出现了面向对象程序设计范型。面向对象系统由对象组成,它是物理世界的自然描述。对象间的消息传递刻划了对象间的关系。其特点是数据类型抽象,模块化,信息隐蔽,重载和动态约束,继承与代码重用,它特别适合于大型软件的开发。

逻辑程序设计和函数程序设计被称之为说明性程序设计。它将问题的逻辑与控制相分离。程序只需指明"需干什么",而由执行系统处理"如何干"的问题。说明性语言的特点是:

(1) 程序是问题的形式描述。

(2) 程序由一系列语言组成,一般来说,语言之间的相互次序是无关的。

(3) 可以用"分而治之"的方法开发程序,即结构程序设计的思想进一步应用到语句级。

(4) 不必考虑与其它语句的关系,就可证明程序中语句的有效性。

(5) 程序与机器独立。

函数型程序设计范型基于 Lambda 演算。其特点归纳如下:

(1) 函数程序设计涉及到表示各种值和函数。

(2) 函数的定义指定输入参数到输出结果的映射。

(3) 函数应用涉及到在函数体部,用实际参数置换形式参数,然后求值函数体。

(4) 函数体中可包含函数的应用,直到函数应用返回所要的值。

基于一阶谓词逻辑的逻辑程序设计范型的特点是:

(1) 逻辑程序设计涉及到指定对象间的关系和推理规则。

(2) 逻辑程序设计具有两类不确定性:如果程序中有若干子句的子句头能与给定的过程调用相匹配时,则试探这些子句的搜索策略是不确定的;当一个子句中含有多个子目标时,执行子目标的顺序是不确定的。

参考文献

[1] Herbrand J. Investigations in Proof Theory. In: vanHerijenoort, J. A Source Book in Mathematical Logic. Cambridge, Mass: Harvard University Press, 1967. 525—581

[2] Prawitz D. An Improved Proof Procedure. Theoria, 1960, 26: 102—139

[3] Gilmore PC. A Proof Method for Quantification Theory. IBM J. Res. Develop. 1960, 4:28—35

[4] Davis M, Putnam H. A Computing Procedure for Quantification Theory. J. ACM,1960, 7:201—215

[5] Robinson J. A. A Machine-oriented Logic Based on the Resolution Principle. J. ACM, 1965, 12(1): 23—41

[6] Kowalski R. A. Predicate Logic as a Programming Language. Information Processing 74. Stockholm, North Holland: 1974, 569—574

[7] Colmerauer A, Kanoui H., Roussel P Yet al. Un Systeme de Communication Homme-Machine en Francais, Groupe de Recherche en Intelligence Artificielle. Universite d'Aix-Marseille, 1973

[8] Battai G, Meloni H. Interpreteurdu Languagede Programmation PROLOG. Groupe d' Intelligence Artificielle, Universite d' Aix-Marseille, 1973

[9] Roussel P. PROLOG: Manuel de Reference etd' Utilization. Groupe d'Intelligence Artificielle, Universite d'Aix-Marseille, 1975

[10] Warrren D H D. An Abstract Prolog Instruction Set. In: Tech. Note 290, AI Center. SRI Inter. University of Edinburgh, 1983

[11] Fuchi K. Hop, Step and Jum p. In: Keynote speech of International Confenence on Fifth Generation Computer Systems

[12] Clark K L, Gregory S. A Relation language for Parallel Programming. In: Arvind and J. Dennis, Proc. of the ACM conf. on Functional Programming Language and Computer Architecture. New, York: ACM

[13] Clard K L, Gvegory S. Parlog: Parallel Programming in Logic. In: Research Report Doc 84/4. Imperial College, London, England

[14] Shapiro E Y. The Static Analysis of Prolog Program. European Computer Industry Research Center, 1985

[15] Ueda K. Guarded Horn Clauses. In: Tech. Report TR-103, ICOT 1985. Tokyo

[16] Yang R. A Parallel Logic Programming Language and Its Implementation. In: Ph. D. Thesis. Department of Electrical Engineering, Keio Univ. 1986

[17] Yang R, Robert P. P-Prolog Computational Complexity of Unification. In: Aiso H Proc. of the International conf. on FGCS, Amsterdam: Elsevier North-

Holland, 1986

[18] Pereira L M, Nasr R J. Delta-Prolog: A Distributed Logic Programming Language. In: Proc. of Inter. Conf. on Fifth Generation Computer Systems. Japan: 1984

[19] Dereira L M et al. Delta-Prolog: A Distributed Backtracking Extension with EVENTS. In: Proc. of the 3rd Inter. Conf. on logic programming. Springer-Verlag, 1986. 69—84

[20] Haridi S, Brand P. ANDORRA PROLOG: An Integration of Prolog and Committed Choice Languages. In: Proc. of Inter. Conf. on Fifth Generation Computer System 1988, Tokyo. 1988. 745—754

[21] Wise M J. A Parallel Prolog: the construction of a data driven model. In: Symp. on Lisp and Functional Programming. ACM. 1982. 56—67

[22] Wise M J. EPILOG=PROLOG+Dataflow. SIGPLAN Notices, 1982, 17(12)

[23] Pollard G H. Parallel Execution of Horn Clause Programs. In: Ph. D. dessertation. UK: Univ. of London, Imperial College of Sci. and Tech. 1981

[24] Conery J S, Kibler D F. Parallel interpretation of logic programs. In: Proc. of the conf. on Functional Programming Languages and Computer Architecture. 1981, 163—170

[25] Conery J S. The AND/OR Model for Parallel Interpretation of Logic Program. In: Ph. D. Thesis. UC Irvine: Dept. of Info. and Comp. Sci. 1983

[26] DeGroot D. Restricted AND-Parallelism. In: Proc. of Inter. Conf. on Fifth Generation Computer System. 1984. 471—478

[27] DeGroot D. Alternate Graph Expressions for Restricted AND-Parallelism. COMPCON 85. 206—210

[28] Ciepielewski A. Towards a computer architecture for OR-parallel execution of logic programs. In: TRITA-CS-8401. Sweden: Ph. D. Thesis. Dept. of Computer Systems, Royal Institute of Tech. 1984

[29] Goto A, Tanaka H, Moto-oka T. Highly parallel inference engine PIE-Goal Rewriting Model and Machine Architecture. New Generation Computing. OHMSHA: 1984, 1: 37—58

[30] Sun Chengzheng, Ci Yungui. PSOF: A Process Model Based on the OR-forest Description. In: Proc. of the Inter. Conf. on Computer and Communication. Beijing: 1986

[31] Tung Y-W. Parallel Processing Model for Logic Programming. In: Ph. D. Thesis. Dept. of EE, Univ. of Southern California, 1986

[32] Hermenegildo M V. An Abstract Machine for Restricted AND-Parallel Execution of Logic Programming. In: Proc. of the Third Inter. Conf. on Logic Programming. Springer-Verlag, 1986. 25—40

[33] Hermenegildo M V, Nasr R I. Efficient Implementation of Backtracking in AND-Parallelism, In: Proc. of the Third Inter. Conf. on Logic Programming. Springer-Verlag, 1986. 40—55

[34] Hermenegildo M V. An Abstract Machine Based Execution Model for Computer Architecture Design and Efficient Implementation of Logic Programs in Parallel. In: Ph.D. Thesis. Univ. of Texas, Austin: 1986

[35] Warren D H D. The SRI Model for OR-Parallel Execution of Prolog-Abstract Design and Implementation Issues. In: the 1987 Inter. Symp. on Logic Programming. San Francisco, California, IEEE: 1987

[36] Hausman B, Ciepielewski A, Haridi S. OR-Parallel Prolog made efficient on shared memory multiprocessors, In: the 1987 Int'l symp. on Logic Programming. San Francisco, California, IEEE: 1987

[37] Gao Yaoqing and Hu shouren. A Parallel Abstract Machine Model for Restricted And - and Limited OR-Parallel Execution of Logic Programs. In: Proc. of Inter. Symp. for Young Computer Professionals. Beijing: 1989

[38] Bahgat R. Non-Deterministic Concurrent Logic Programming in ANDORRA. World Scientific series in Computer Science, 37

[39] Gupta G, Hermenegildo M. ACE:And/Or-Parallel Copying-based Execution of Logic Programs. Lecture Notes in Computer Science, 1991, 569

[40] Westphal H and Robert P. The PEPSys Model:Combining Backtracking AND- and OR-Parallelism. In: International Symposium of Logic Programming. San Francisco, IEEE Computer Society, 1987. 436—448

[41] Ram Kumar R, Kale L V. Compiled Execution of the Reduce-OR Process Model on Multiprocessors. In: 1989 North American Conference on Logic Programming. MIT Press, 1989

[42] Naish L. Heterogeneous SLD Resolution. JIP, 1984

[43] Clark K L. Negation as Failure. In: Gallaire H & Minkker J. Logic and Databases. Plenum Press, 1978. 293—322

[44] Wolfram D A, Maher M J, Lassez J-L. A unified treatment of resolution strategies for logic programs. ICLP, Upsalla University, 1984

[45] Lloyd J W. Foundations of Logic Programming. Berlin: Springer-Verlag, 1984

第二章　顺序逻辑程序设计语言 Prolog 及其应用

在前一章我们介绍了逻辑程序的理论基础[3][5][6][7][8][9][10][11]。本章我们将介绍一种广泛使用的逻辑程序设计语言 Prolog[1][12][13]。

Prolog 是为表示和使用特定领域知识而设计的一种程序设计语言，它的数据结构和表示与传统程序设计语言有相当大的差异。具体而言，它将领域看作一个对象集合，而知识则是这些对象的关系集合。这些关系描述了对象的特征及其相互关系。Prolog 程序就是由描述这些对象和关系的合式公式构成的。

例如，当我们说“John 是 Paul 的父亲”时，我们简单地描述了一个关系(is-father-of)和两个对象(John 和 Paul)，这可用公式表示为：

is-father-of(john, paul).

那么，此时若我们提出“谁是 Paul 的父亲”这个问题时，它可归结为检查关系“is-father-of”中：Paul 是否与另外一个对象联系着，由此找到问题的答案。

在描述上述关系时，我们采用了标识符命名对象及其关系。一般我们称关系 is-father-of 为谓词(predicate)，称谓词中的对象为变元(argument)。

2.1　Prolog 语法

本节我们将对 Prolog 语法进行介绍。在给出纯语法定义之前，我们先给出一个 Prolog 程序的小例子。

2.1.1　法国饭店的 Prolog 程序举例

通过下面简单例子，我们将对 Prolog 程序有一个初步了解。

在饭店中，菜单是由一些条目组成，这些条目可分为开胃小菜(hors d'oeuvres)，肉(meat)或鱼(fish)以及餐后甜点(dessert)三大类。我们可将这三类看成三个关系。那么，这个菜单就构成了一个小型数据库，用 Prolog 语言表述如下：

```
hors-d-oeuvre(artichauts-Melanie).
hors-d-oeuvre(truffes-sous-le-sel).
hors-d-oeuvre(cresson-oeuf-poche).

meat(grillade-de-boeuf).
meat(poulet-au-tilleul).

fish(bar-aux-algues).
fish(chapon-farci).
```

dessert(sorbet-aux-poires).

dessert(fraises-chantilly).

dessert(melon-en-surprise).

上述 Prolog 程序是一个规则集。规则定义了关系，这些关系每次引入一个对象及其分类。例如，hors-d-oeuvre(Cresson-oeuf-poche)表示 Cresson-oeuf-poche 是一道餐前小菜(hors d'oeuvre). 实际上，上述规则只是对应简单的事实。根据上述事实，我们可以回答下面一类问题：

Cresson-oeuf-poche 是餐前小菜吗？

用 Prolog 语言可表示成：

?－hors-d-oeuvre(Cresson-oeuf-poche).

上述问题与规则不同之处是它的前面有符号"?－"。通过检查上述数据库中的事实，可以得到肯定的回答。而对于问题：

?－hors-d-oeuvre(Salade-de-tomatos).

的回答就是否定，因为数据库中不包含这个事实。

现在若我们想知道饭店能提供什么样的餐前小菜，靠询问所有可能的问题是不可能的。因为上述每个问题都只有一个肯定或否定的回答，而我们所关心的问题是：

"什么是餐前小菜?"或"餐前小菜 H 是什么?" 这里 H 并不代表某一特定的对象，而是所有属于餐前小菜集合的对象，是需要程序帮助我们寻找的。我们称 H 为变量。

在本例中，若我们输入问题：

?－hors-d-oeuvre(H).

则 Prolog 解释器将产生如下结果：

{H=artichauts-Melanie}

{H=truffes-sous-le-sel}

{H=cresson-oeuf-poche}

上述对象能与变量 H 匹配，并使断言为真。

从上述初始关系出发，我们可以构造较复杂和通用的关系。例如，我们可定义主食关系 main course，它表示以鱼或肉类为主食，可写成：

maincourse(M) :－meat(M).

maincourse(M) :－fish(M).

上述规则中，" :－"之前的部分为规则头，其后的部分称为规则体。它们用自然语言可解释为，"若 M 是肉类，则 M 是主食"，"若 M 是鱼类，则 M 是主食"。这里我们又使用了变量 M，它在每个规则中都出现，分别表示所有的肉类和所有的鱼类。

必须明确的是，变量的作用域限定在定义它的规则范围内。因此，上面第一个规则中的变量 M 与第二个规则中的 M 无关。根据这两条规则，我们可询问"什么是可能的主食?"，即

?－maincourse(M).

可得到如下答案：

{M=grillade-de-boeuf}

{M=poult-au-tilleul}

{M=bar-aux-algues}

{M=chapon-farci}

现在我们将注意力放在一顿正餐上。遵照通常的习惯，正餐包括一道餐前小菜，一道主食和餐后甜点。因此，一顿饭由三部分组成：H 是餐前小菜，M 是主食，D 为餐后甜点。因此，我们可用规则表示如下：

```
meal(H,M,D) :-hors-d-oeuvre(H),
              maincourse(M),
              dessert(D),
```

它可以这样描述：如果 H 满足关系 hors-d-oeuvre，M 满足关系 maincourse，D 满足关系 dessert，那么，H，M，D 满足关系 meal. 通过上述三个关系的连接，我们定义了新关系。询问“什么是可能的一顿饭？”可写成？ -meal(H,M,D). Prolog 解释器将给出如下结果：

{H=artichauts-Melanie, M=grillade-de-boeuf, D=sorbet-aux-poires}.

{H=artichauts-Melanie, M=grillade-de-boeuf, D=fraises-chantilly}

...

{H=artichauts-Melanie, M=chapon-farci, D=melon-en-surprise}

{H=truffes-sous-le-sel,M=grillade-de-boeuf,D=sorbet-aux-poires}

...

{H=truffes-sous-le-sel,M=chapon-farci, D=melon-en-surprise}

{H=cresson-oeuf-poche,M=grillade-de-boeuf, D=sorbet-aux-poires}

...

{H=cresson-oeuf-poche,M=chapon-farci,D=melon-en-surprise}.

上述结果是 36 种可能的组合。

下面我们可询问一些更具体的问题：我们想知道鱼为主食的一顿饭有哪些。询问为？ -meal(H,M,D),fish(M). 它明确地表示了我们所希望满足的两个条件。当 Prolog 求出满足第一个条件 meal(H,M,D)的解后，变量 H，M，D 均被赋值。例如，H=artichauts-Melanie，M=grillade-de-boeuf，D=sorbet-aux-poires。当我们转向 fish(M)时，由于 M 已被赋值，因此需验证 fish(grillade-de-boeuf). 由于数据库中不包含这一事实，那么对 H，M，D 所赋的值就不能满足上述问题，这种可能将被排除，而必须转而尝试下一个可能的结果。最后，程序将给出以下 18 种可能的结果：

{H=artiehauts-Melanie,M=bar-aux-algues,D=sorbet-aux-poires}

{H=artichauts-Melanie,M=bat-aux-algues,D=fraises-chantilly}

{H=artichauts-Melanie,M=bar-aux-algues,D=melon-en-surprise}

{H=artichauts-Melanie,M=chapon-farci,D=sorbet-aux-poires}

{H=artichauts-Melanie,M=chapon-farci,D=fraises-chantilly}

{H=artichauts-Melanie,M=chapon-farci,D=melon-en-surprise}

{H=truffes-sous-le-sel,M=bar-aux-algues,D=sorbet-aux-poires}

...

{H=truffes-sous-le-sel,M=chapon-farci,D=melon-en-surprise}

{H=cresson-oeuf-poche,M=bar-aux-algues,D=sorbet-aux-poires}

...

{H=cresson-oeuf-poche,M=chapon-farci,D=sorbet-aux-poires}

从上述结果中,我们应注意以下几点:

● 为了满足关系的合取式,我们从左到右进行检查。

● 在运行过程中,某一变量可能被赋值。如果一个变量接受一个值的话,那么,不管它今后出现在哪里都有同样的值。

● 变量局限在它们所出现的规则中。每当规则使用时,它的变量都获得一个新的实例,这象传统程序设计语言中的局部变量。规则中的这些变量的使用可以说是一种重命名方法。

● 在一个关系中,不存在输入变元与输出变元的区别(输入变元是那些在关系求值之前已经赋值的变元;输出变元是那些值随着关系求解过程而改变的变元)。同样的关系可能有不同的作用:如果它所有的变元都是已知的,则我们只须验证关系是否满足;如果它的变元中某一个是未知的,我们可依次计算出满足关系的变元的所有值。

● 程序的执行是不确定的,我们只管求出满足关系的所有可能的组合。

现在我们给每顿饭增加一些信息,即给菜单中每一项增加一个热量值。

calories(artichauts-Melanie,150).

calories(cresson-oeuf-poche,202).

calories(truffes-sous-le-sel,212).

calories(grillade-de-boeuf,532).

calories(pouset-au-tilleul,400).

calories(bar-aux-algues,292).

calories(chapon-farci,254).

calories(sorbet-aux-poires,532).

calories(fraises-chantilly,289).

calories(melon-en-surprise,122).

事实 calories(chapon-farci,254)表示每份 Chapon-farci 包含 254 卡路里热量。

为查询餐前小菜的热量值,我们可询问

?—hors-d-oeuvre(H),calories(H,C).

对满足关系 hors-d-oeuvre(H)的每一个 H 值,程序将对变元 C 赋值 n,n 满足关系 calories(H,n)上述询问的回答是:

{H=artichauts-Melanie,C=150}

{H=truffes-sous-le-sel,C=212}

{H=cresson-oeuf-poche,C=202}

一顿比较考究的晚餐一般希望知道这顿饭总的热量值。为此,我们定义如下关系:

```
value(H,M,D,V) :—calories(H,X),calories(M,Y),
                 calories(D,Z),total(X,Y,Z,V).
```

其中 V 是各部分食品的热量总和。为确定各种组合的热量值，我们可询问：

```
? —meal(H,M,D),value(H,M,D,V).
```

回答为：

```
{H=artichauts-Melanie,M=grillade-de-boeuf,D=sorbet-aux-poires, V=905}
{H=artichauts-Melanie,M=grillade-de-boeuf,D=fraises-chantilly, V=971}
{H=artichauts-Melanie,M=grillcde-de-boeuf,D=Melon-en-surprose, V=804}
  ...
{H=artichauts-Melanie,M=chapon-farci,D=melon-en-surprise, V=526}
{H=truffes-sous-le-sel,M=grillade-de-boeuf,D=sorbet-aux-poires, V=967}
{H=truffes-sous-le-sel,M=grillade-boeuf,D=fraises-chantilly, V=1033}
  ...
{H=truffes-sous-le-sel,M=chapon-farci,D=melon-en-surprise, V=588}
{H=cresson-oeuf-poche,M=grillaed-de-boeuf,D=sorbet-aux-poires, V=957}
{H=cresson-oeuf-poche,M=grillade-de-boeuf,D=fraises-chantilly, V=1023}
  ...
{H=cresson-oeuf-poche,M=chapon-farci,D=melon-en-surprise, V=578}
```

此时，我们可通过如下规则选定一顿热量均衡的正餐：

```
balanced-meal(H,M,D) :—meal(H,M,D)
                       value(H,M,D,V)
                       less-than(V,800).
```

这表示只有当一顿饭的总热量小于 800 卡路里时，才是一顿营养比较均衡的饭。对于询问 ? —balanced-meal(H,M,D). 将有如下回答：

```
{H=artichauts-Melanie,M=poulet-au-tilleul,D=sorbet-aux-poires}
{H=artichauts-Melanie,M=poulet-au-tilleul,D=melon-en-surprise}
{H=truffes-sous-le-sel,M=poulet-au-tilleul,D=melon-en-surprise}
{H=cresson-oeuf-poche,M=poulet-au-tilleul,D=melon-en-surprise}
```

上述答案将所有高热量食品全排除在外。

下面是一个完整的 Prolog 程序：

```
hors-d-oeuvre(artichauts-Melanie).
hors-d-oeuvre(truffes-sous-le-sel).
hors-d-oeuvre(cresson-oeuf-poche).

meat(grillade-de-boeuf).
meat(poulet-au-tilleul).

fish(bar-aux-algues).
```

```
fish(chapon-farci).

dessert(sorbet-aux-poires).
dessert(fraises-chantilly).
dessert(melon-en-surprise).

maincourse(M) :-meat(M).
maincourse(M) :-fish(M).

meal(H,M,D) :-hors-d-oeuvre(H),
              maincourse(M),
              dessert(D),

calories(artichauts-Melanie,150).
calories(cresson-oeuf-poche,202).
calories(truffes-sous-le-sel,212).
calories(griffade-de-boeuf,532).
calories(poulet-au-tilleul,400).
calories(bar-aux-algues,292).
calories(chapon-farci,254).
calories(sorbet-aux-poires,532).
calories(fraises-chantilly,289).
calories(melon-en-surprise,122).

value(H,M,D,V) :-calories(H,X),
                 calories(M,Y),
                 calories(D,Z),
                 total(X,Y,Z,V).

balanced-meal(H,M,D) :- meal(H,M,D),
                        value(H,M,D,V),
                        less-than(V,800).

total(A,B,C,D) :-val(add(A,add(B,C)),D).

less-than(X,Y) :-val(inf(X,Y),1).
```

其中 val,add,inf 为 Prolog 系统中的基本函数。