PROLOG 语言程序设计

张文星、 吴 林 合译

胡孝沁 校 胡笔蕊 审校

刘小芹 封面设计

中 国 天 问 电 脑 公 司

前　言

目前，计算机程序设计语言 PROLOG正在迅速地博得全世界程序员的欢迎。它自从七十年代初问世以来，便在符号计算的许多领域得到了应用，其中包括：

关系型数据库
数字逻辑
解决抽象问题
理解自然语言
结构设计
符号方程式求解
生物化学结构分析
人工智能的许多领域

到目前为止还没有一本关于使用 PROLOG实际编程的教科书；但是，有很多人正在形势促动之下学习这种语言，他们参阅很简明的参考手册和发表了的为数不多的论文以及计算机同行们当中一些口头流传的知识，也许这是对 PROLOG 的一种赞赏姿态吧。然而，由于许多大学生，研究生正开始接触 PROLOG，许多同事们感到很需要一本学习 PROLOG 的指导书。我们希望本书能在某种程度上满足这种需要。

许多初学者往往会发现在传统的计算机语言中，编写一个程序就是完成一个算法，而在PROLOG中却不是这样。PROLOG 程序员更加关心的是在他的问题中有哪些逻辑关系和目标出现，以及所要求的解中哪些关系为“真”。可以把PROLOG 看成既是一种“规定性”语言，又是一种“描述性”语言。PROLOG 方式是描写一个问题的已知事实和关系，而不是规定计算机为解决该问题而采取的工作步骤。当用 PROLOG 编程时‘计算机要执行的实际操作由以下几部分指明：逻辑说明PROLOG 的语义，PROLOG 从给定事实中“推理”出新事实，程序员提供的显式控制信息。

PROLOG 能既实际又有效地实施“智能”程序执行中的许多方面，诸如：无定义，可并行性和模式定向过程的调用。PROLOG 提供了一个一致的数据结构，叫做项，包括 PROLOG程序在内的所有数据都是由项构成的。一个PROLOG 程序是一些子句的集合，其中每一个子句，或者是关于已给信息的一个事实，或者是一条规则，它说明怎样从已知事实“推理”所涉及的结果。于是，可以认为 PROLOG 是迈向“逻辑程序设计”这一最终目标的第一步。本书不准备过多地涉及逻辑程序设计，以及为什么PROLOG 本身不是最终逻辑程序语言这一问题。我们只准备讨论如何利用现有的 PROLOG系统来编写有用的程序。

本书可有多种用途。我们编写这本书的宗旨不是为了讲授编写PROLOG程序的技巧，因为这不是简单地看一本书或听几次课就能学到手的，只有实干才能学会。我们希望一个没有数学基础的初学者能从本书中学到PROLOG，同时，我们建议由一个懂得 PROLOG 的程序员来指导初学者，作为给学生介绍这种程序设计的课程的一部分。当然，在这里我们假定初学

者可以使用一台配备PROLOG系统的计算机并懂得如何使用计算机终端。一个有经验的程序员在学习本书时不需要任何帮助，他可能对本书在数学上的朴实无华感到惊讶，但这正是我们力求做到的。我们曾使用这本书的初稿来教授大学里哲学专业和心理学专业的研究生，尽管他们在中学也没有学多少数学知识。

根据我们的经验，程序设计初学者觉得PROLOG程序比用传统语言编写的同样程序更易理解，他们常常不喜欢传统语言对使用计算手段的种种限制。另一方面，有传统语言经验的程序员能更好地理解象变量和控制流这样一些抽象概念。但是，尽管有这传统语言的经验，他还是可能觉得PROLOG难以适应，要经过许多实例的说服证明，他才能会觉得PROLOG是个有用的程序设计工具。当然，我们也知道已有许多富有经验的程序员成了PROLOG的热心使用者。但说到底，我们写这本书的目的不是为了转变人们对PROLOG的看法，而是为了教授PROLOG本身。

象大多数其他程序设计语言一样，PROLOG也有许多不同的处理系统，各有其自己的语义和语法特点。在本书中，我们采用了一种“核心PROLOG”，本书所有例子都与处理系统相对应的标准版本一致，这种标准版本是在爱丁堡为四种不同的计算机系统研制的，它们是：配有TOPS—10操作系统的DEC system—10计算机；配有Unix操作系统的DEC PDP—11计算机；配有RT—11操作系统的DEC LSI—11计算机；配有EMAS操作系统的CL2980计算机。在DEC计算机上的处理系统也许是最广泛的。本书的所有例子均能在这四种处理系统上运行。在附录中，我们例举了一些现有的PROLOG处理系统的例子，并指明了与标准版本的差异，读者将会注意到大多数的差异只是表面上的。

读者应依次阅读各章节，但当初学者开始编写一个超过十个子句的PROLOG程序时，先读一下第八章将是有益的，而且，最好能参照阅读描述PROLOG各种具体处理系统的相应附录。附录解释怎样进入子句，它有哪些调试功能，以及其它一些实际问题。浏览全书是没有害处的，但要注意不要跳过前面的章节。

书中的每一章都分为若干节，许多节后附有练习，希望读者不要放过。一些练习的答案附于书后。第一章是初步介绍，目的在于让读者略知一点用PROLOG进行程序设计的要求。该章介绍了PROLOG的基本概念，最好能认真理解。第二章对前一章介绍的一些概念进行了更加详细的讨论。第三章涉及数据结构，并引出了一些小的程序例子。第四章详细介绍了“回溯”的概念，并介绍了控制“回溯”的“cut”符号。第五章介绍输入输出的功能。第六章描述了PROLOG标准“核心”中的每一个内部谓词。第七章从许多途径汇集了一些程序实例，还解释了它们的编写过程。第八章提出了一些调试PROLOG程序的意见，还提供了一个控制流程的替换模式。第九章介绍了文法规则的语法，并检验了为利用文法规则分析自然语言的某些设计。第十章描述了PROLOG和它的母体—数学定理的证明及逻辑程序设计—之间的关系。第十一章给出一些实例，有兴趣的读者可能会希望练习一下自己的编程能力。

在此我们向我们的老师—Rod Burstall，Peter Scott Langston和Robin Popplestone表示敬意，他们在程序设计思想方面给予我们很大的影响。我们也感谢我们的朋友们—Alan Bundy，Lawrence Byrd，Robert Kowalski，Fernando Perelra和David Warren，他们的合作促进了Prolog发展成为一个实用而又有力的程序设计工具，他们的鼓励促进了本书的写作。特别是Lawrence Byrd从一开始就提供了许多意见，以此来支持本书的成书。我们还要感谢Jon Cunningham，Richard O'keefe，Helen Pain，Fernando Perdlra，

Gordon Plotkin, Robert Rae, Peter Ross, Maxwell Shorter, Aaroll Sloman和David warren这些朋友，他们对本书的草稿提了许多有益的意见和建议。此时此刻，M·F·Clocksin还需特别感谢他在Epistemics研究院和人工智能系的研究生们，他们许多次都是编程教学试验中的对象。为了在书中举例，我们不受限制地从一般流传的PROLOG程序中改写和发展了许多程序。如有人因此感到受到了轻视或作品被抄袭，我们在此致歉。

本书的准备工作是作者在爱丁堡大学人工智能系任教时做的，系主任Jim Howe为我们提供了许多方便和设备，使这一计划得以完成，我们表示感谢。

M·F·C

C·S·M

一九八一年六月于

苏格兰爱丁堡

第一章　初步知识

PROLOG是一种计算机程序设计语言，可用来解决涉及客体以及客体间关系的问题。本章将给出这种语言一些基本成份，而不顾及许多细节、形式规则和例外情形。目前也不求完整和精确。本章集中阐述了PROLOG的一些基本概念：事实，问题，变量，联结词与规则，以便使读者尽快会编写有用的程序。PROLOG进一步的一些问题，如表与递归将在后面章节里交代。

当我们希望计算机解决涉及客体及其相互关系这一类问题时，就可使用PROLOG语言，例如："约翰有一本书"这句话，阐述了一个关系——所有关系，这个关系存在于一个客体"约翰"和另一个独立的客体"书"之间。更进一步来说，这个关系是有序的：约翰拥有书，而不是书拥有约翰，当我们问："约翰有书吗？"这实际上便是试图找出一个关系。

有些关系并不总是提及所有涉及到的客体。例如，"珠宝是珍贵的"这句话中就含有一个关系，即"是珍贵的"，这个关系与珠宝有关。但我们既不提及谁认为珠宝珍贵，也不说为什么。这都取决于我们要表示些什么。运用PROLOG在计算机上编制有关诸如此类关系的程序时，应根据要计算机完成的任务来决定提供多少细节说明。

在开始编程序之前，首先谈一个哲学问题。我们都熟悉利用一些规则来描述客体间的关系。例如，有这样一条规则"如果两个女性有相同的父母，则她们是姐妹"，它给我们描述了关于"姐妹"的含义。同时也告诉我们怎样判断两人是否为姐妹：只须看她们是否都是女性并是否有相同的父母。规则往往过于简单，但它们作为"定义"却是能为人们所接受，注意到这一点是很重要的。毕竟不能指望一个定义能把所有有关的情况包括无余。比如，在实际生活中存在不少"姐妹"关系超出了以上规则的含义。然而在解决一个特定的问题时，我们应该只注重那些对解决问题有帮助的规则。因此，我们应该认真设计一个能满足意图的抽象而又简化的定义。

PROLOG的程序组成如下：

- □ 说明一些有关客体及其相互间关系的事实
- □ 定义一些客体及其相互间关系的规则
- □ 提出有关客体及其相互间关系的问题

例如：假设已给出了关于"姐妹"的规则，然后提出一个问题"玛丽与简是否为姐妹？"PROLOG便将检查给出的有关玛丽和简的那些事实，然后根据提问回答"yes"或"no"，所以，我们可以把PROLOG看作为一个事实与规则的库，它将依据事实和规则回答问题。编写一个PROLOG程序，就是提供所有的事实和规则。PROLOG系统将使计算机作为一个事实和规则的库来使用，并提供了从一个事实到另一个事实进行推理的手段。

PROLOG是一种会话式语言，执行一个PROLOG程序实际上就是进行一场人机对话。如果通过一个带有键盘和显示器的终端使用PROLOG进行操作，情形将是这样：操作员通过键盘输入字符，计算机通过显示器（屏幕或纸）给出结果。操作时要做的事是将有关要解决的问题的事实与规则键入。然后，如果提出的问题有效，PROLOG则通过显示器给出答案。

此为试读，需要完整PDF请访问：www.ertongbook.com

以下将逐条介绍PROLOG的基础知识。但还不是PROLOG的全部特点，后面的章节将详述这些内容并给出例子。

1.1 事　　实

首先讨论关于客体的事实。设想我们要告诉PROLOG一个事实："约翰喜欢玛丽"，这个事实包含两个客体，John和Mary，和一个关系"likes"。在PROLOG中，用如下标准形式表示事实：

likes(john, mary) .

以下几点是重要的：

- □ 所有的客体与关系必须以小写字母开头。
  例如：likes, john, mary。
- □ 关系写在前面，客体用一对圆括号括住，彼此用逗号隔开。
- □ 每个事实的结尾附以"."

当用事实来定义客体间的关系时，必须注意圆括号中客体的次序。实际上，这个次序是可以任意规定的，但一经规定，以后就必须遵守。例如，在上面这个事实中，括号中的两个客体，"喜欢"者放在前面，而被"喜欢"者放在后面。因此，事实likes(john, mary)就不同于事实likes (mary, John)，根据我们当前的规定，第一个事实表示John喜欢Mary，而第二个事实表示Mary喜欢John。如果我们要陈述Mary喜欢John这样一个事实，我们必须写清楚：

likes(mary, john) .

下面给出了一些事实的例子和可能的解释：

```
Valuable(gold) .      "Gold is valuable"
female(jane) .        "jane is female"
owns(john, gold) .    "John owns gold"
father(john, mary) .  "John is the father of mary"
```

各事实的中文解释是：

"金子是贵重的"
"詹妮是女性"
"约翰拥有金子"
"约翰是玛丽的父亲"

每当使用一个名字的时候，总是指一个特定的独立的客体。由于我们熟悉英语，所以很清楚名字John与Jane分别指单个人。但在其它事实中，我们用了gold这个名字，它指什么就不很明显了。当使用一个名字时，必须确定它的含义。例如名字"gold"指一个客体，可以认为此客体代表某一笔特定的财富，当写出事实Valuable (gold) 时，就意味着命名为"gold"的一笔特定财富是贵重的。另一方面也可以把名字"gold"定义为天然金块，从而Valuable (gold) 的意义为天然金块是贵重的。由此可见，解释一个名字的方法是很多的，完全决取于程序设计者。只要遵守决定，编程中就不会有问题。在上面的事实中，我们可以选择"天然金块"这种解释，也可以选择另一个，但PROLOG事实的形式却是相同的。所以，重要的是首先区分不同的解释，以便确定名字的含义。

下面介绍一些术语。每一个事实中的圆括号里的客体称为“自变量”，而位于括号前的关系名字称为“谓词”，所以，Valuable 是有一个自变量的谓词，而likes是有两个自变量的谓词。

客体和关系的名字是任意选取的。我们完全可以用a (b,c) 来代替likes(john, mary)；只须记住a代表“喜欢”，b代表john，c代表Mary。但我们通常总是选择那些能帮助我们记住它们所代表的客体的名字。所以，事先必须确定名字的含义和自变量的次序，并在以后一直遵守它。

在PROLOG中，一个关系所涉及的自变量的个数是无关紧要的。如果我们要定义一个谓词Play，它涉及两个竞赛者以及他们所参加的一场比赛，这样就要求三个自变量。例如：

```
Play (jane, mary, football) .
Play (jane, jim, badminton) .
```

同样，在PROLOG中，如果给出的事实与实际生活中的不符也没有关系，我们可以给出king (John,france) 来表示“John”是法兰西的国王(把“是国王”看作一种关系)。这在现实世界显然是错误的，特别是因为法国早在1792年就废除了君主制。但PROLOG 对此既不知道，也不关心。PROLOG中的事实不过是表示任定的客体间的关系。

PROLOG中事实的集合被称为“数据库”。当我们为解决一个特定的问题而给出一系列事实（以后还要加上规则）就采用“数据库”这个词。

1.2 问　　题

一旦已知一些事实后，就可以问一些有关它们的问题。PROLOG中的问题看起来象一事实，只不过前面加了一个特殊的符号，它由一个问号（？）和一个连字号（—）构成。例如：

```
? —owns (mary, book) .
```

如果我们把mary解释为mary这个人，而book解释成某一本特定的书，那么这个问题就相当于问“Mary拥有那本书吗？”。或相当于问“Mary”拥有那本书这件事是事实吗？而不是问Mary 是否拥有所有的书，或任意的一本书。

对PROLOG提出一个问题后，它将搜索前面键入的数据库，寻找与问题中的事实匹配的那些事实。两个事实匹配的意思是它们的谓词相同，且对应的自变量也都相同。如找到相匹配的事实，PROLOG就回答“yes”，如数据库中不存在这样的事实，就回答“no”。回答将显示于终端，位于所提问题的下一行上。假设有如下的数据库：

```
likes (joe, fish) .
likes (joe, mary) .
likes (mary, book) .
likes (john, book) .
```

如把这些事实键入PROLOG系统，便可以提出下列问题，且PROLOG将在问题的下一行给出回答：

```
? —likes (joe, money) .
no
? —likes (mary, book) .
```

```
yes
? —king (john, france) .
no
```

前三个问题的回答是很明确的。但PROLOG也对“John是否是法国国王”这样的问题给出了回答“no”，这是由于以上四个likes关系表中并没有王室关系这样的事实。必须记住在PROLOG中回答“No”并不表示“不是”，而是表示“不知道”。举例来说，假设一个希腊名人的数据库仅包括下列三个事实：

```
human(socrates).
human(aristotle).
athenian(socrates).
```

对它提出如下问题：

```
? —athenian(socrates).
yes
? —athenian(aristotle).
no
? —greek(socrates).
no
```

虽然Aristotle确是雅典人(Athens)，但仅从数据库的事实却无法证明。同样，虽然在数据库中给出了socrates是雅典人，但如果数据库中没有更多的信息，同样无法证明他是希腊人。所以，当PROLOG回答“no”时，它的意思是“无法证明”。

在前一个数据库中，John和Mary都喜欢同一个客体。这是由于我们在两个有关John和Mary的事实中，可以找到同样的名字“book”。

到目前为止，讨论过的事实和问题中还没有什么特别有趣的东西，我们能做的只是重新得到我们键入的一些信息。但当引入了“变量”这个概念后，我们就可以提出“Mary喜欢什么？”这样更有用的问题了。

1.3 变　量

若想知道John喜欢什么，如果一个个地提问“John”喜欢书吗？”“John喜欢Mary吗？”……等等，并等待PROLOG一个个地回答yes和no，这是很麻烦的。较聪明的方法是，让PROLOG回答John所喜欢的某件东西。可以构造这样的一个问题，“John喜欢X吗？”这时，我们并不知道X所代表的是什么客体，但我们希望PROLOG告诉我们有哪些可能性。在PROLOG中，不但可以给特指的客体命名，还可以使用诸如X这样的名字来代表那些我们要求PROLOG确定的客体，这种名字称为变量。PROLOG使用一个变量时，该变量可以是已例示的，也可以是未例示的。一个变量已例示，意即该变量已代表了一个客体。未例示，意即还不知道该变量代表什么。一个变量必须以大写字母开头，PROLOG就依此来区别变量和一个特定客体的名字。

当向PROLOG提出了一个带变量的问题后，PROLOG就搜索所有的事实，以找出一个变量可代表的名字。所以，当我们提问“John喜欢X吗？”时，PROLOG就在所有的事实

中寻找John喜欢的客体。

象“X”这样的一个变量，本身并未被命名为一个特定客体，但它却可代表那些我们无法命名的客体。例如，我们无法把“John喜欢的东西”作为一个客体来命名，而PROLOG为我们提供了一条途经——使用变量。我们将用提问：

?—likes(john, X)。

取代提问：

?—likes(john, “Something that john likes”)。

同样可被PROLOG接受，因为变量可以是一个以大写字母开头的任何名字。

假设有一个数据库：

likes(john, flower) .

likes(john, mary) .

likes(paul, mary) .

现在提出一个问题：

?—likes(john, X) .

它含义是：“有什么东西是John喜欢的吗？”，PROLOG将如此回答：

X = flowers

然后它将停下来等待操作员发出进一步的指令，PROLOG回答此问题的过程是这样的：PROLOG搜索数据库，寻找一个与问题匹配的事实。由于这个未例示的变量作为自变量出现，PROLOG便将用事实中处于同样位置的任何其它自变量来匹配。本例中，PROLOG搜索数据库寻找以“likes”为谓调，且第一个自变量是“john”的任何事实，而第二自变量可以是任意的，这正是因为问题中的第二个自变量是未例示的。找到这样一个事实后，不管第二个自变量是什么，变量X都将接受它。由于PROLOG是在数据库中，依次搜索诸事实，所以，首先找到事实likes (john, flowers)，这时X就代表“flowers”，或者说X被例示为“flowers”。同时PROLOG在数据库中该事实的位置上作上标记。为什么要这样做的原因我们马上就要解释。

当PROLOG找到一个与问题匹配的事实后，便打印变量当前所代表的客体。在本例中，仅有一个变量X，与之匹配的客体是flowers。然后，象前面说过的那样，PROLOG停下来等待进一步的指令。如果这时在终端上按RETURN键，则表示没有新的指令，PROLOG停止搜索，但如键入的是分号“；”和RETURN，PROLOG将从数据库中留有标记处起继续搜索。当PROLOG从数据库中被标记的位置，而不是从起始位置开始搜索，我们称这种现象为PROLOG试图“重新满足”该问题。

当PROLOG给出了第一个回答（X = flowers）后，如果再键入分号“；”要求它继续执行。这就意味着我们想用另一个解答来满足这一问题，找出X可能代表的另一个客体，也意味着PROLOG必须“忘掉”X代表“flowers”，而把它当做一个未例示的变量继续搜索。由于我们需要一个新的回答，所以，本次从位标记处继续搜索。下一个能匹配的事实是likes(john, mary)。这一次X用mary例示，同时，PROLOG在事实likes(john,mary)处作位标记，然后打印“X = mary”，并停下来等待进一步的指示。如果我们继续键入“;”，PROLOG将继续搜索。在本例中，已没有更多为John所喜欢的东西了。这样PROLOG将打印出“no”，并停止搜索，这时，允许提出更多的问题或决定更多的事实。

如果对于上述事实，提出这样的问题：

?—likes(X, mary) .

将会怎样？这个问题的意思是“有没有什么客体喜欢Mary”。我们知道在本例中，这样的客体是Paul。如果我们要得到全部的回答，只要在PROLOG给出每一个回答后键入“; ”：

```
?—likes(X, mary) .          (问题)
X = john;                   (第一个回答，之后键入 "; ")
X = Pule;                   (另一个回答，之后键入 "; ")
   no                       (没有别的答案了)
```

1.4 联 结 词

假如我们要想对诸如“John与Mary相互喜欢吗？”这样涉及更多更复杂的关系提出问题，有一个办法就是先提问John是否喜欢Mary，如果PROLOG回答“yes”，再问Mary是否喜欢John。这样，这个问题就包括两个要满足的独立目标。由于PROLOG程序设计者经常要用到这种组合型问题，它有一个特殊的表示法。假设有如下数据库：

```
likes(mary, food) .
likes(mary, wine) .
likes(john, wine) .
likes(john, mary) .
```

我们要知道的是John与Mary是否相互喜欢。为了得到回答，我们应该换一种方式提问：“John喜欢Mary并且Mary喜欢John吗？”“并且”表示我们对两个目标的联结感兴趣，我们要求二者同时得到满足。在PROLOG中，采用在两个目标中安置一个逗号来表示：

?—likes(John, Mary) , likes(Mary, John) .

逗号代表“并且”，它把为回答问题所必须满足的任意个数的不同的目标分隔开，这些用逗号分隔开的目标提供给PROLOG后，PROLOG便搜索数据库，依次为每一个目标寻找匹配的事实。为了问题得到满足，每一个目标都必须得到满足。在本例中，因为数据库中有John喜欢Mary，故第一个目标为真，但由于没有likes(mary, John)这样一个事实，所以第二个目标得不到满足。而我们问的是John和Mary是否相互喜欢，所以整个问题的回答是“no”。

联结词和变量的配合使用，可以引出很有趣的问题。比如，我们已知道John和Mary不相互喜欢，但我们还可以问：

有没有什么东西是John和Mary共同喜欢的？这个也是由两个目标组成的：

- □ 首先，找出是否有Mary喜欢的某物
- □ 其次，找出John是否喜欢X，不论X为何物

用联结词联结两个目标为PROLOG的形式：

?—likes(mary, X) , likes(john, X) .

为了回答这个问题，PROLOG首先试图满足第一个目标，如果该目标在数据库中，则PROLOG将在此处作出标记，然后再试图满足第二个目标，如第二个目标也被满足，PROLOG将标记数据库中的这个目标位置。这样，我们就得到了一个同时满足两个目标的回答。

必须记住，每一个目标都有其各自的位标记。如果第二个目标得不到满足，PROLOG将试图重新满足它的前一个目标（在这里是第一个目标）。值得注意的是，PROLOG能胜任对数据库中每一个目标进行搜索。如果数据库中一个事实被匹配，即满足该目标，则PROLOG在数据库中相应位置作标记，以备今后重新满足该目标时用。当一个目标需要重新满足时，PROLOG从该目标的位标记处，而不是从数据库的头开始搜索。为了解释"回溯"，再来看看PROLOG解决问题："有没有什么东西是John和Mary共同喜欢的"时的几个步骤：

1． 为第一个目标搜索数据库。由于第二个自变量"X"是未例示的，它可与任何客体匹配。在上述数据库中，第一个可匹配的事实是likes(mary, food)。这时问题中所有的X都被例示为"food"。PROLOG标记数据库中的这个位置，以供在需要重新满足该目标时用。另外，我们应该记住这时X已是被例示。当需要重新满足该目标时，PROLOG应"忘记"X。

2． 因为第二个目标是"likes(john, X)"而X代表food。所以应该在数据库中搜索likes(john, food)。但不存在这个事实，所以该目标无法满足。当一个目标失败后，必须重新试图满足前一个目标，故PROLOG试图满足likes(Mary, X)，这次是从数据库中已被标记的地方开始搜索。但首先，应该使X重新变成未被例示的，这样，X又可与任何东西匹配了。

3． 由于被标记的位置在likes(mary, food)处，所以，PROLOG从这个事实的后面开始搜索。因为搜索尚未到达数据库的底部，故并未穷尽所有的Mary喜欢的东西，下一个可匹配的事实是likes(mary, wine)。变量X被例示为wine，同时，PROLOG标记数据库中的位置，以备下次重新满足时用。

4． 如前，PROLOG转向第二个目标，这次要搜索的是likes(john, wine)。这里，PROLOG并不是重新满足该目标，而是再一次进入它（从左至右）。所以，这次搜索是从数据库的头开始。写上就找到了可匹配的事实，并报告结果。由于这个目标已被满足，PROLOG也将标记这个位置，在数据库中，PROLOG为它试图满足的每一个目标都设置了一个位标记。

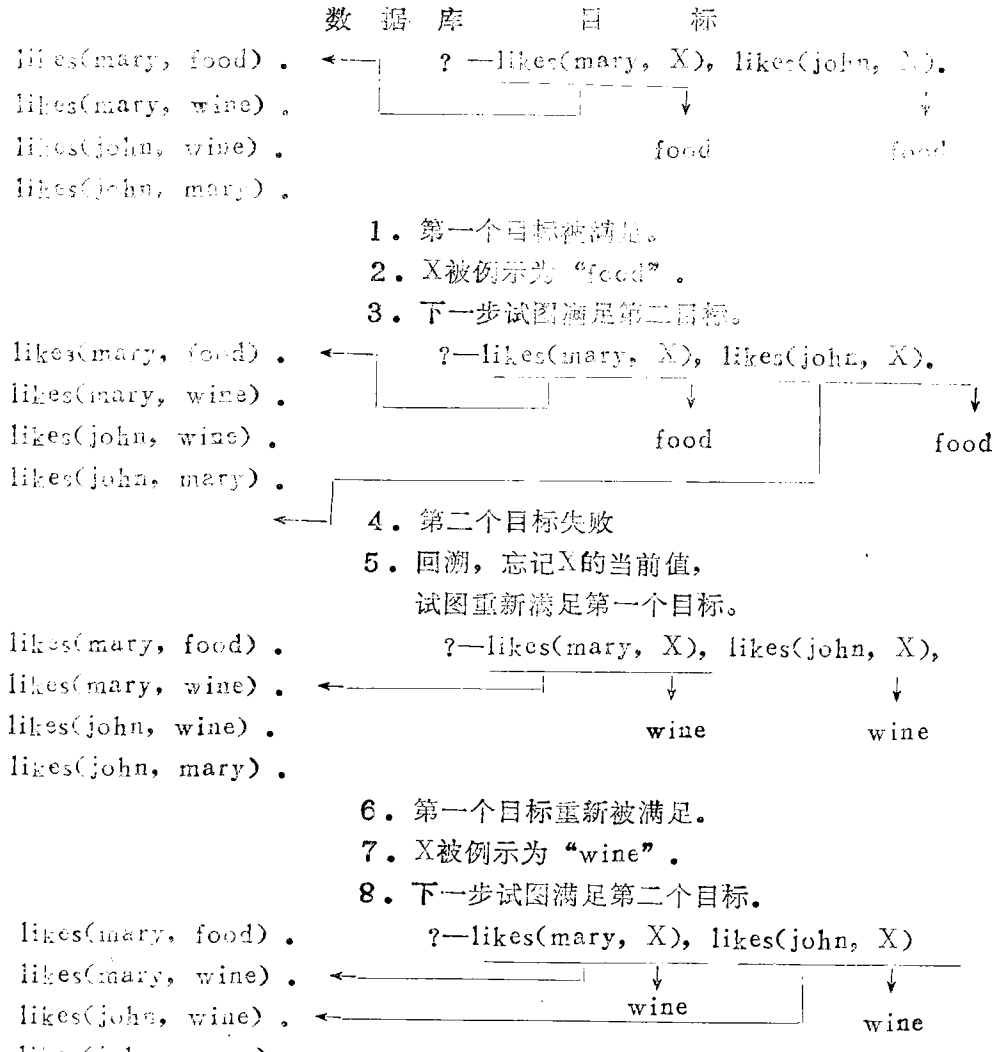
5．现在，两个目标都得到满足。变量X代表名字"wine"。第一个目标在数据库中事实likes(mary, wine)处有一个位标记，第二个目标在事实likes(John, wine)处有一位标记。如果还有其它问题，PROLOG一旦找到答案后，便停下等待进一步的指令。如果键入一个分号，PROLOG就会继续搜索John和Mary都喜欢的东西。很清楚这相当于从以前留下的位标记处开始，试图找出重新满足的两个目标。

总的来说，一个联结型的问题中，目标从左至右排列，并用逗号分隔。每一个目标可有一个左邻和一个右邻。当然最左的目标没有左邻，最右的目标没有右邻。对一个联结型的问题，PROLOG试图从左到右满足每一个目标。如果一个目标被满足，PROLOG在数据库中与此目标相联系之处留下位标记，我们可以把此想象为从目标向数据库中满足目标的位置画一个箭号。此外，一些未例示的变量也被例示了，这便是以上第一步所做的。如果一个变量被例示，意味着问题中所有的该变量均被例示，PROLOG转而试图满足目标的右邻，搜索从数据库头开始。每当一个目标满足，便在数据库中留下标记（从目标向数据库中匹配的事实画另一箭号），以备今后重新满足时用。每当一个目标失败(找不到一个匹配的事实)，PROLOG退回并试图重新满足它的左邻。搜索从位标记开始，但首先，PROLOG要把所

此为试读，需要完整PDF请访问：www.ertongbook.com

有在这个目标（左邻）中被例示的变量重新变成未例示的，即对它们进行“取消”操作。如果从右边进入的每一个目标无法重新满足，那么这一连串的失败将蔓及它左边的目标。如果第一个目标（最左的）也失败了，由于它已没有左邻可试图重新满足，这时整个联结型问题也就失败了。这种操作——PROLOG 重复地试图满足和重新满足联结问题中的目标——被称为回溯。下一章将扼要地介绍回溯，第四章给出更详细完整的描述。

阅读例子时，在每一个目标中的变量下面，写出由于目标成功而已例示的客体将是很有帮助的。还应该从目标向数据库的位标记画一个箭号。下面这个“图解”的例子给出了上述几个不同阶段的操作。



在本书的例子中将尽量指出回溯的地方，以及用它解决问题的效果怎样。回溯是一个非常重要的概念，以至要用第四章整章来讨论它。

练习 1．1 假设在上例中 PROLOG 结束后，操作员又键入一个分号“；”要它继续找出

Mary与john共同喜欢的其它东西，把发生的过程用示意图表示。

1.5 规　　则

设想要表示“John喜欢所有的人”这样一个事实，方法之一就是写出如下一个个相互独立的事实：

```
likes(john, acfred) .
likes(john, bertrand) .
likes(john, chartes) .
eikes(john, bavid) .
        ⋮
```

从而在数据库里对每个人都要给出一个事实。如果在PROLOG程序中有几百人，这样的做法会是很乏味的。另一个方法是这样表示：

“John喜欢任何客体，只要这个客体是人”。

这个事实就是用规则的形式表示john喜欢什么。而不是列出john喜欢所有客体。在一个john可能喜欢任何一个人的世界中，用这条规则比用一个事实表要简明得多。

在PROLOG中，当要说明一个事实依赖于其它一组事实时，就可使用规则。在自然语言中，可用“如果”一词来说明规则。例如：

如果下雨，我就打伞。

如果果味酒比啤酒便宜，John就买果味酒。

也可以用规则来说明定义，例如：

如果X是动物并有翅膀，则X是一只鸟。或者：

如果X是女性并X与Y有相同的父母则X是Y的一个姐妹。

在上述的定义中，使用了变量X，Y。应该特别注意的是在一条规则中，所有相同的变量代表相同的客体，否则就将违背定义的意思。比如，无法从Fred是一个动物和Mary有羽毛推出Fred是一只鸟。所以，要求PROLOG中关于变量解释的一致性，也同样适用于规则。

一条规则是关于客体及其关系的一般陈述。例如，不但可以说“如果Fred是动物且Fred有翅膀，则Fred是一只鸟”。而且也可以这样说：“如果Bertram是动物且Bertram有翅膀，则Bertram是一只鸟。”所以可以在规则的不同使用时，允许变量代表不同的客体。当然，如上述，在规则的同一次使用中，对变量解释的必须一致。

让我们来看一些例子。首先是含有一个联结词的单变量规则：

“如果一个人喜欢酒，John就喜欢他”，或换句话说：

“如果什么东西喜欢酒，John就喜欢它”或用变量：

“如果X喜欢酒，John就喜欢X”。

在PROLOG中，一条规则由头和体组成，其间用一个符号“：—”（读作“如果”）联起来。上例写成PROLOG形式便是：

```
likes(john, X) :—likes(X, wine)。
```

注意规则也用圆点“.”结尾。这条规则的头是likes(john，X)。规则的头说明了这条规则需要定义的事实。这条规则的体，是likes(X，wine)。为使头为真，规则的体的各个目标必须先后被满足。例如，假使John对他所喜欢的东西更加挑剔，只要简单地在体中增

加一些用“，”隔开的目标：

likes(john, X):—likes(X, wine), likes(X,food)。或用自然语言来表示：

“如果一个人既喜欢酒又喜欢食物，John就喜欢他。”

或者，假设“John喜欢的是任意一个喜欢酒的女性”：

likes(john, X):—female(X), likes(X, wime)。

当看到一条PROLOG规则时，就应该注意它的变量。在上面这条规则中，变量X被使用了三次。当X被例示为某个客体时，所有在其作用域中的X便都被例示了。在规则使用的一些特殊场合，X的作用域是从头到“.”的整条规则。所以，在上例中，如X被例示为“Mary”，PROLOG便试图满足目标female(mary)和likes(mary, wine)。

下一个例子是一条多个变量的规则。假设一个数据库由一些关于维多利亚女王家族的事实组成。我们将使用一个有三个自变量的谓词“parents”：

parents(X,Y,Z)。

其中

X的父母是Y与Z，Y是母亲，Z是父亲，例中还要用含义很明显的谓词female与male。数据库的一部分如下：

```
male(albert).
male(edward).
female(alice).
female(victoria).
parents(edward,victoria, albert).
parents(alice, victoria, albert).
```

我们还将用到本章开头指出过的关于“sister—of”的规则。这个规则定义谓词sister—of，它具有两个自变量，这样当X是Y的姐妹时，sister—of (X, Y) 为真。注意在这个谓词名中用了下横线“—”。虽然我们并未完全给出名字的构成法则，但下横线是可出现在名字中的。在下章中我们将扼要地介绍规则。如果：

- □ X是女性
- □ X有母亲M和父亲F
- □ Y与X有同样的父母

那么X是Y的姐妹，用PROLOG规则表示如下：

```
siser—of(X, Y) :—
                female(X),
                parents(X, M, F),
                parents(Y, M, F).
```

我们使用变量名“F”与“M”分别表示“父亲”和“母亲”（而不是含义不同的“男性”和“女性”）。注意在这里使用了的变量未曾出现在规则头中。这些变量，M和F，应与其它变量同样对待。当PROLOG使用此规则时，开始M与F未例示，故在PROLOG满足目标parent (X, M, F) 时，它们可与任何客体匹配，但一旦被例示后，则规则的本次使用中的M与F均被例示。下面的例子可帮助理解这些变量是如何被使用的。

让我们提出如下问题：

?—sister—of(alice, edward)。

当对上述数据库和规则sister—of提出这样一个问题后，PROLOG将如下处理：

1．首先，问题与仅有的一条规则sister—of的头匹配，所以，规则中的X被例示为alice规则。Y例示为edward。该问题的位标记指向规则，现在，PROLOG试图一个个地满足体中的三个目标。

2．由于在上一步中，X已被例示为alice，故第一个目标是female (alice)。又从事实表中看到，此目标为真，故第一个目标成功。此时PROLOG在数据库上标记目标的位置。（数据库中的第三项）。因为没有新的变量被例示，故不作任何记录。然后 PROLOG 试图满足下一个目标。

3．现在，PROLOG搜索parents(alice, M, F)。其中，M与F未例示，故可与任何自变量匹配。事实parents(alice, victoria, albert)与目标匹配，所以这个目标也成功了。PROLOG在数据库中标记目标的位置（第六项），并记录M被例示为victoria，F例示为albert（读者可把它们写在规则中的目标下面）。现在 PROLOG 试图满足下一个目标。

4．因为已知Y为 edward，M与F分别代表 victoria 与 albert，故 PROLOG 搜索 parents(edward, victaria, albert)。由于找到了匹配的事实（第五项），故该目标成功。它是最后一个目标，因此整个目标成功，且事实sister—of(alice,edward) 为真，PROLOG 给出回答“yes”。

假如我们想知道Alice是什么人的姐妹，相应的PROLOG问题应该是：

?—sister—of(alice, X)，

对于这个问题PROLOG将如下处理：

1．问题与唯一的规则sister—of的头匹配。规则中的变量X被例示为alice。由于问题中的变量X是未例示的，所以规则中的变量Y也未例示。不过这两个变量现在变为“共享”，一旦其中的一个被例示为某个客体，另一个也被例示为同一个客体。当然目前它们都是未例示的。

2．第一个目标是female(alice)，同前面一样，它是成功的。

3．第二个目标是parents(alice,M, F)，它与Parets(alice,victoria,albert)匹配，现在M、F，变为已知。

4．由于Y还是未知的，故第三个目标是 parents(Y, victoria, albert)，它与事实parents(edward, victoria, albert) 匹配。现在已知Y为edward。

5．由于所有的目标均获成功，整条规则成功，已知 X 为 alice（问题中给出，Y 为 edward。因为Y（规则中的）与X（问题中的）共享，故X（问题中的）也被例示为edward。PROLOG给出回答“X = edward”。

如通常一样，PROLOG 停下来等待是否要它给出该问题的全部解答。上述问题不止一个解答，本章末有一个练习就是关于找出其余的解答。

至此我们知道，有种方式可把象likes这样的谓词信息提供给PROLOG。一是事实，一是规则。通常，一个谓词可用一些事实与规则的组合来定义。称它们为谓词的子句。单个的事实或规则也叫子句。

作为进一步的例子（在此不涉及王室），考虑如下的规则：

“如果一个人是贼，并且他喜欢某样东西，同时这样东西是贵重的，这个人就可能偷这

样东西。”

用PROLOG形式来写：

may—steal(P, T)：—thief(P), likes(P、T), valuable(T) 这里使用了谓词may-steal，它用两个变量P与T表示“某人p可能偷某物T”，这条规则依赖于thief,likes和v—aluable等各子句。有如下由上述子句构成的PROLOG数据库。在每一行前加上用/*…*/括起来的某些行号。在这里告诉了我们怎样写PROLOG的注解。PROLOG将忽略注解，但在程序中加入它更为方便。

/*1*/thief(john).

/*2*/likes(mary, food).

/*3*/likes(mary, wine).

/*4*/likes(john, X) ：—likes(X, wine).

/*5*/may—steal(X, Y) ：—thief(X) , likes(X, Y).

注意关于likes的定义有三个独立的子句：两个事实和一条规则。对PROLOG来说，这没有什么实际差别，只不过是当为满足一个目标而搜索数据库时，规则将引起进一步的搜索，以满足它的子目标。而事实没有子目标，所以它的成功或失败是立即的。现在让我们看看提出这样一个问题后的情形：

“John可能偷什么？”

首先这个问题要翻译成PROLOG形式：

?—may—steal(john, X).

为回答这个问题，PROLOG将做如上搜索。

1．首先，PROLOG在数据库中搜索关于may—steal的子句，并在/*5*/行上找到了一条如此形式的规则。PROLOG在数据库中标记这个位置。由于这是一条规则，它的一些子目标必须被满足，以证实它是否为真。规则中的X被例示为问题中的John。两个未例示的变量(问题中的X与规则中的Y)匹配。它们“共享”。如果对这一点不太明白，可回过头参阅例sister—of。为使规则成功，必须它的各子目标成功，故首先搜索第一个目标thief(john)。

2．由于thief(john)在数据库中（行/*1*/），故此子目标成功。PROLOG在数据库中标记这个位置，并且没有新变量得到例示。然后，PROLOG试图满足行/*5*/上的第二个目标。由于X仍代表john，所以PROLOG搜索likes(john,Y)。注意其中Y仍未例示。

3．目标likes(John, Y)与行/*4*/上规则的头匹配。这时目标中的Y与规则头中的X“共享”，并均未例示。为满足这一条规则，现在搜索likes(X, wine)。

4．因为目标与行/*3*/上的事实likes(mary, wine)匹配，故目标成功；X代表mary。

5．由于行/*4*/上的目标成功，整条规则成功。因为行/*5*/中的Y与X共享，所以Y也被例示为mary。这样从行/*4*/上建立了事实likes(john, mary)。

6．因此行/*5*/也成功了，Y例示为mary。由于Y与原问题中的第二个自变量共享，故问题中的X现在例示为mary。

John所以偷Mary的推理是这样建立的：

为偷东西，首先John必须是个贼，从行/*1*/发现这是一个事实。其次，John必须喜欢这样东西。从行/*4*/我们可发现John喜欢那些喜欢酒的任何东西。而从行

／＊3＊／又可知道mary喜欢酒。因此，John喜欢mary。这样，“偷窃”的两个条件均得到满足。故John可以偷mary。

注意行／＊2＊／上“Mary喜欢食物”的事实。在这个特定的问题中始终没用到，这是由于我们不需要它。

在本例中，在不同的子句中多次用到了变量X与Y。例如在may—steal规则中，X代表能偷东西的客体，但在likes规则中，X代表被喜欢的客体。为使程序不致混乱，PROLOG必须能分辨X在不同的子句中代表的不同客体。变量的定义域能消除这种混乱。

1.6 小结与练习

现在我们已接触了PROLOG的大部分基本概念。特别是如下几点：

- □ 给出关于客体的事实
- □ 提出关于事实的问题
- □ 使用变量以及它们的定义域是什么
- □ 用联结表示“与”关系
- □ 用规则的形式表示关系
- □ 关于回溯的介绍

这样便可以写出有用的程序，以熟练地使用简单的数据库。完成下面的练习是很有用的。

为了更好地使用本书，应先阅读序言。此外，刚开始利用一个现有的PROLOG系统编程时，应先查阅有关的附录。

当读者掌握了以上内容后，就可以进入下一章，那里介绍了我们本章未提及的一些细节，以及如何在PROLOG中使用数字。在以下几章中，我们会看到PROLOG是一种使用方便、表达能力强的语言。

练习1·2 若在前述维多利亚女王的部分家族关系数据库上，使用sister—of规则，其答案不止一个。试解释怎样获得这些答案，并给出答案本身。

练习1·3 假使已有人编写了定义下列关系的PROLOG子句：

```
father(X, Y) .      ／＊X是Y的父亲＊／
mother(X, Y) .      ／＊X是Y的母亲＊／
male(X) .           ／＊X是男性＊／
famale(X) .         ／＊X是女性＊／
parent(X, Y) .      ／＊X是Y双亲之一＊／
diff(X, Y) .        ／＊X与Y不是同一个人＊／
```

要求写出适用于下列关系的PROLOG规则

```
is—mother(X).       ／＊X是一个母亲＊／
is—father(X).       ／＊X是一个父亲＊／
is—son(X).          ／＊X是一个儿子＊／
sister—of(X, Y).    ／＊X是Y的姐妹＊／
granpa—of(X,Y).     ／＊X是Y的祖父＊／
sibeing(X, Y).      ／＊X是Y的一个亲属＊／
```


此为试读，需要完整PDF请访问：www.ertongbook.com