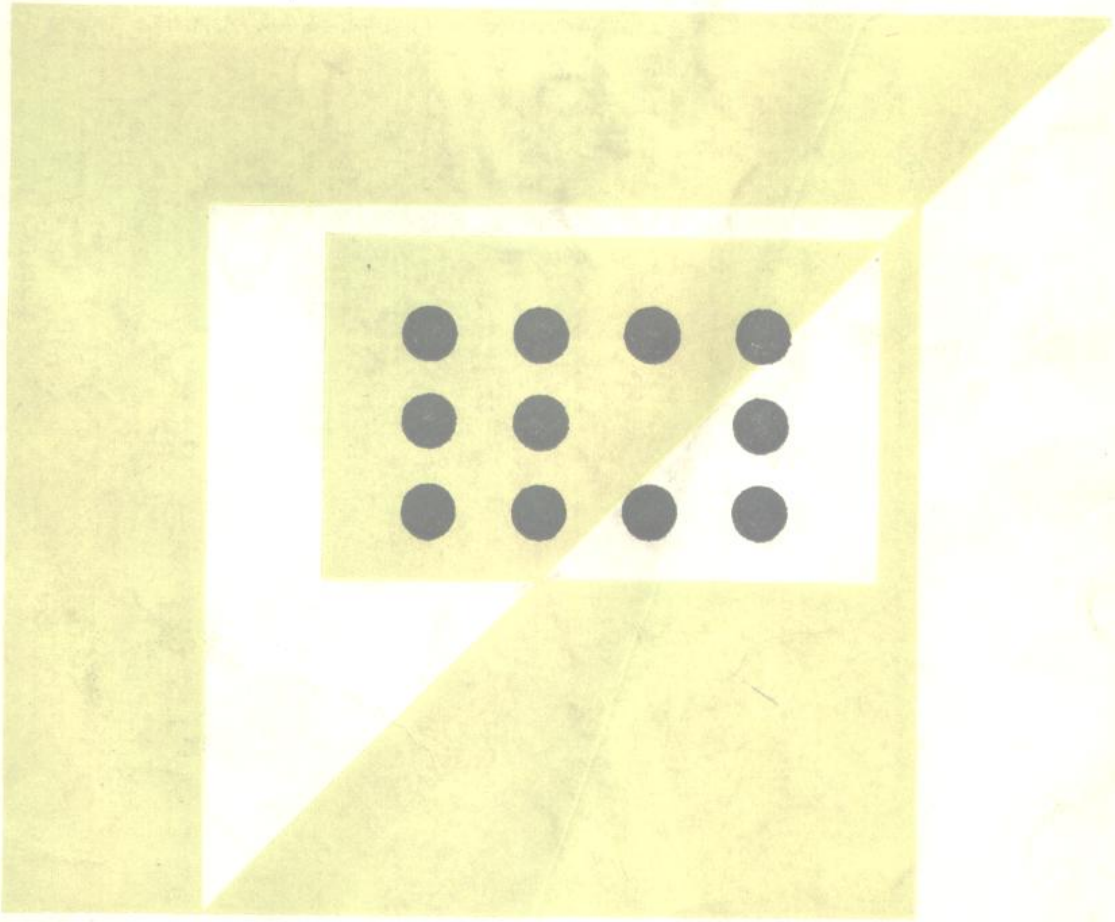
纪有奎　冀　翾　编译

Prolog
语言程序设计及其应用



海　洋　出　版　社

Prolog 语言程序设计及其应用

纪有奎　冀　翔　编译

海　洋　出　版　社

1986 年 · 北京

内 容 简 介

本书是以爱丁堡大学 Clocksin 教授的经典著作为蓝本编译的。内容丰富、深入浅出、理论联系实际，系统地介绍了核心 Prolog 的基本概念、语言规则、编程及上机调试方法，并列举了较多的应用实例。为便于读者迅速掌握这种先进的新型语言，书中各章都给出了习题及答案。

本书适合于高中和大学师生、科技人员、管理人员、计算机工作者阅读学习，并适于作高中及大学教材。

Prolog 语言程序设计及其应用

纪有奎、龚翔 编译

海洋出版社出版（北京市复兴门外大街1号）

新华书店北京发行所发行 卫生印刷厂印刷

开本：787×1092 1/16 印张：12¾ 字数：200千字

1986年3月第一版 1986年3月第一次印刷

印数：1—9,500

统一书号：17193·0725 定价：2.30元

前　言

Prolog 是计算机程序设计语言，已在世界上崭露头角。

Prolog 是由 Program 和 Loigic 二词的词头构成。它具有递归、回溯、模式匹配、数据库等很强的功能。逻辑推理是它的特长。应用范围广：适用于数学论证，非数值处理，自然语言的理解，专家系统等人工智能领域；可用于编写管理程序；用它编写游戏、智力题的程序——简洁、生动。而且，Prolog 语言易学、易用，程序简明、易读。(有人说在易学、易用、人机对话等方面胜过 BASIC 语言)。学习者大都称赞该语言奇特、新颖，容易理解（因为它面向人类语言)。它不仅适于写小程序，更适于写大程序。有人统计过要用 600 多行 PASCAL 语言写的程序，用 LISP 语言写时要用 100 行，而用 Prolog 编程则行数更少。

1973 年左右首先在法国完成 Prolog 处理系统，而后以欧洲为中心开始传播。近年来美国十分重视它。1982 年日本把它选为研制第五代智能计算机的核心语言。目前它正在有关国家迅速传播，一些国家大中小学开设此课。这“超高级、明星般的计算机语言”，受到世人注目。它受到我国有关方面的重视，我国有关杂志发表文章题名“惹人注目的 Prolog”，举办了一些学习班，有的学校讲授它。

本书是以苏格兰爱丁堡大学的 W.F.Clocksin 等人编写的经典著作“Programming in Prolog” 为蓝本进行编译的，是教材型。(原著已在一些国家作为教材，我们综合了 1981 年初版和 1984 年再版的原著)。

本书深入浅出，理论联系实际，内容丰富。共十一章：入门介绍，基本概念，语法规则，应用实例，怎样编写程序和上机调试。每章有习题，书末附答案。第七章不仅有许多实用程序，还有智力、趣味程序（如：最短路径，迷宫，河内塔，高斯八皇后等)。本书难易结合，不同文化程度可选用适当的章节阅读。

同其他程序设计语言一样，Prolog 也存在着不同版本。本书采用“核心 Prolog”，它与 DEC-10 Prolog 系统十分接近，适用 DEC PDP-11 等多个不同的计算机系统。micro-PROLOG 系统软盘 (1984 年 12 月 3.1 版本)，可在 IBM PC 机系列和它的兼容机上参阅附录E运行本书的程序。(本系统软盘由海洋出版社发行)

本书可供大中学师生，科技、管理人员，计算机工作者，计算机房等有关人员阅读。

在编译过程中，范春晓、刘海门等同志给予协助，并由中国科学院电工研究所张汉亭同志审阅，在此一并表示谢意。由于我们水平有限，时间仓促，难免有误，敬请广大读者指正。

北方工业大学计算机系　纪有奎
中国科技大学计算机系　冀　翔

1985年 8 月

目　录

第一章 引 言

Prolog 是一种计算机编程语言，它用于解决有关目标及目标之间关系的问题。本章不准备对 Prolog 做完整、精确的描述，仅通过程序介绍该语言的基本要素。为了使读者尽快地明确何处可写有用程序，我们必须弄清一些基本概念，如事实、询问、变量、连接和规则。Prolog 的其他特点，如表、递归将在以后章节介绍。

如果一个问题可表示成若干对象和它们的关系形式，那么我们用计算机解这类问题时，就用 Prolog 语言。例如，当我们说："约翰有这本书"时，即表明在一个对象"约翰"和另一个独立的对象"书"之间有一个关系——所有。进一步说，这关系有一特殊的次序——约翰有书，而不是书拥有约翰！如果问"约翰有那本书吗？"试找出它们的关系。

并不是所有的关系都要涉及问题所包括的所有的对象。例如："宝石很值钱"，有一个关系，即"值钱"，只涉及一个对象"宝石"，并没牵涉到谁发现宝石值钱。这就是说关系完全依赖于问题的本身。在 Prolog 中，当你为这样一些关系编程时，你所规定的大量细节要依赖于你要计算机去完成什么。

先弄清几个所涉及的观点，然后再开始编程。我们习惯于用一些规则去描述对象间的关系。例如："如果两个人是女的，且有相同的父母，则此二人是姐妹"。它告诉我们什么是姐妹；还告诉我们怎样判断二人是否是姐妹：简单的看看是否都是女的且有相同的父母。应该注意，规则通常是过于简单的，但可被接受为定义。我们毕竟不能有对某件事描述得面面具到的定义。例如：多数人都认为实际生活中的姐妹要比以上定义的多得多。但不管怎样，在解决特殊问题时，只需要注目于可帮助解决问题的那些规则。所以如果一个理想的、简单的定义对所解决问题是充分的，我们就应该承认它。

Prolog 计算机程序由以下三部分组成：

(1) 对客观事物的对象和关系的描述；

(2) 定义有关对象和关系的规则；

(3) 给对象和关系进行询问。

例 已用 Prolog 描述了姐妹的规则。现问玛莉和珍妮是否是姐妹。Prolog 将搜寻我们所给的有关玛莉和珍妮的情况，根据此情况问答"是"或"不是"。因此，我们可把 Prolog 看作事实和规则的存储器，并能根据事实和规则回答问题。用 Prolog 编程时要包括提供的所有事实和规则。Prolog 系统使计算机用于存储事实和规则，且提供从一个事实推出另一个事实的方法。

Prolog 是会话式语言，即有很好的人机对话功能，假设你在计算机终端使用Prolog，且终端有键盘和显示器，你向计算机敲入字符，计算机则用显示器（屏幕或纸带）给你输出结果。Prolog 等待你输入要解问题所需事实和规则后，由你提出一个问题（本题所涉及范围内的问题），Prolog 就给出回答并输出。

下面逐个介绍 Prolog 基本要素。Prolog 的其他特性将在以后几章讲解。

1.1 事　实

首先讨论对象的事实。

"John 喜欢 Mary"，包含两个对象："John" 和 "Mary"，且有一个关系："喜欢"。在 Prolog 里，写成以下的标准形式：

喜欢 (John, Mary)　　likes(john, mary) .

注意：

(1) 对象和关系名必须以小写字母开头，如 likes，john，mary；

(2) 关系打头（在圆括号外），所有对象写在圆括号里，对象间用 "，" 分开；

(3) 事实结束处必须写上实心点 "．"。

在用事实定义对象间的关系时，要注意圆括号中对象的书写顺序。实际此顺序是任意的，但我们规定某顺序后，就要保持它的一致性。如上面的例子，我们把 "喜欢者" 放在括号里的首位，而 "被喜欢者" 放在后面，所以 likes(john, mary) 就不同于 likes(mary, john)。根据我们的约定，第一个是说 John 喜欢 Mary，而第二个说 Mary 喜欢 John。如果要说 "Mary 喜欢 John"，则必须表示为

likes(mary, john) .

以下是英语句子和 Prolog 表达式：

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 .
gives(john, book, mary) .	John gives the book to Mary .

每句中都含有名字，它是特殊的独立对象。由于我们熟悉自然语言，因此对 John，Jane 作独立体名字能理解，但对另一些事实，如 gold（金子）具有多义性，必须指明它的含义，要选定怎样解释它。是指那些值钱的被叫作金子的大量 gold 呢，还是把 gold 解释为金矿。若是后者 valuable(gold)，就是指金矿很值钱了。所以一个名字可根据编程者的选择有多种解释。只要你选择后，总是作同样解释就不会出什么问题。顺便说，以上事实我们称 gold 为 "金矿"，但以前可能有人把它解释为 "金子"，而 Prolog 却视它们为一体，所以早些区别不同意思的名字是非常重要的，以便明确使用。

解释一些术语。每个事实中括号内的对象的名字叫自变量。注意计算机编程人员在使用自变量这一词时，在技术上不能有二义性，也不能是某论题或课题。首先写在圆括号外面的关系名叫谓词。上例中的 valuable 是有一个自变量的谓词，likes 是有两个自变量的谓词。

对象和关系名完全是任意的，我们可以用 a(b, c) 代替 likes(john, mary)。其中 a：likes；b：john；c：mary。因此我们通常起一些帮助记忆的名字，但切记替换量的意义和自变量的顺序，而且对所有相同变量保持一致。

Prolog 中一个关系可有多个自变量，如谓词（play）玩，牵涉两个玩者和玩具，即有三个自变量。例如：

```
play(john, mary, football).        John 和 Mary 踢足球
play(jane, jim, bedminton).        Jane 和 Jim 打羽毛球。
```

Prolog 也可表示实际不存在的事实。如国王（约翰，法国）意为约翰是法国国王，这在现实中显然是不可能的，因为法国 1792 年就废除了君主制，但 Prolog 不知道也不关心这个。Prolog 中事实允许你表达任意对象中的关系。

Prolog 中所有事实的集合叫数据库，当为解决某特殊问题收集许多事实时，就可称为“数据库”。

1.2 询　问

我们可以对一些事实询问。在 Prolog 中一个问题很象一个事实，只是在它前面加一特殊符号——一个问号和一个连字符号。如：

```
?- owns(mary, book) .
```

如果“mary”代表 Mary，“book”代表一本特殊的书，那么这个问题是“Mary 有这本书吗?”，或“Mary 有这本书是真的吗?”，但不能问她是否有所有的书。

当询问时，Prolog 要搜寻事先输入的事实组成的数据库，和是否有事实与问题相匹配。当事实与询问的关系相同且对应的自变量也相同时，我们说二者相匹配。如果 Prolog 找到了一个事实与问题相匹配，则回答“是”；否则，数据库里无此事实则回答“否”。Prolog 回答将输出在终端显示器上询问的下一行。

如对数据库：

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

将它输入 Prolog 系统后，进行以下询问并得如下回答：

```
?- likes(joe, mary) .
yes
?- likes(mary, joe) .
no
?- likes(mary, book) .
yes
?- king(john, france) .
no
```

前三个问题的回答是很明确的，而第四个问题的回答是“no”，是指数据库中没有关于王室的关系。在 Prolog 中，回答“no”通常指“超出我们知识范围，即“不知

道”，而不是指“定义不合法”。

设立关于一些著名希腊人的数据库如下：

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

询问：

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

尽管亚里士多德（Aristotle）生活在雅典（Athens）可能是真的，但不能从已给的数据库中简单证明这点，同样尽管数据库中指明苏格拉底（Socrates）是雅典人，但仍不能认为他是希腊人，除非提供更多的信息数据库，所以 Prolog 只能回答“no”，即“不知道”。

不同的对象可以喜欢同一个对象，如上面的 Mary 和 John 都喜欢书。

到目前为止我们讨论的询问并无特殊的用途，只是把输入的信息再找出来。不过对于“玛莉喜欢什么?”这类询问会很有用，为回答这样的询问引出了变量。

1.3 变　量

如果通过 Prolog 对“约翰喜欢书吗?”，“约翰喜欢玛莉吗?”等问题的回答(yes 或 no) 来寻找“约翰喜欢什么?”的答案，那是非常繁琐的。在 Prolog 中有另一种方法，直接询问：“Does john likes X?”，等待 Prolog 给出可能的回答。这就是说在 Prolog 中不仅可以定义特定的对象名，还可以定义待定对象的名字X。这种名字称为变量。

变量可以是被说明的，也可以不被说明。当它有具体意义时即被说明，否则没被说明。变量一定以大写字母打头，以便 Prolog 识别。

当询问中含有变量时，Prolog 将检索所有事实以便寻找代表变量的对象。

象X这样的变量，不能自己命名一个特殊的对象，但可以用于表示我们所不能命名的对象。如我们不能把“Something that john likes”作为一个对象，若想使Prolog 接受，可以改变形式，如：

```
?- likes(john, “somathing that john likes”) .
```

我们必须用变量如下：

```
?- likes(john, X) .
```

可根据需要把变量名写的很长，如：

```
?- likes(jokn, Somathing that john likes) .
```

Prolog 把后一问题看成变量，因为它是大写字母打头。

现有以下数据库：

```
likes(john, flowers) .
likes(john, mary) .
likes(paul, mary) .
```

询问 ?-likes(john, X) .

这是问“约翰喜欢什么东西?”，Prolog 将回答：X = flowers

然后计算机等待下一条指令。现在来看看计算机是怎样工作的。

Prolog 接受询问时，变量是无值的。它要在数据库中寻找与它匹配的事实，并且第一个自变量是“john”，找到这样的事实后，第二个自变量就是X所代表的。Prolog是按输入数据库的顺序查找的，所以 likes(john, flowers). 先被找到，则X代表 flowers，或说X被赋值为 flowers，Prolog 将记下匹配事实的位置。

Prolog 一次只找出一个与询问匹配的事实，并将变量的值输出。此例中变量X意义为 flowers，此时 Prolog 等待下一个指令。如果你对这一个答案满足了，就按回车键，Prolog 不再继续寻找；如果你还想要更多的回答，则按“；”键后按回车键，Prolog 将从位置指针记下的匹配位置处继续向下寻找，再次回答询问。

假设对本例，我们得到一个回答后再按键“;”和回车键，即要找X还有何意义。那么 Prolog 将丢弃X的 flowers 意义，重新把它作为无意义的。Prolog 从位置指针处向下寻找，发现 likes(john, mary)。即X代表 mary，且位置指针置于 likes(john, mary)。处。Prolog打印“X = mary”，并等待命令。如果再打入一个“；”和回车键，Prolog 将继续寻找，而此后再无匹配事实，则 Prolog 自动停止，但允许再询问或输入新事实。如果对同样的数据库询问：

```
?- likes(X, mary) .
```

即“谁喜欢 Mary?”，从数据库中可以看到回答是 john 和 paul。执行过程如下：

```
?- likes(X, mary) .
X = john ;
X = paul ;
no
```

注“——”者为机器回答，其余为用户打进去。

1.4 连　　接

我们怎样回答象“John 和 Mary 彼此相爱吗?”这样复杂的问题。

一种方法是我们先问“John 喜欢 Mary 吗?”，如回答“是”，再问“Mary 喜欢

John 吗?”，这样询问就包括两个独立对象的询问。因为 Prolog 编程者经常用这种组合，所以给它定义一个新特殊的符号。对以下数据库：

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

要问 “Mary 与 John 彼此相爱吗?”，可以用这种方式“Does John like Mary and does Mary like John?”，这和 “and” 表示把两个有关询问 “连接” 起来一样，需要既满足前者又满足后者。我们用一个 “，” 号分隔它们：

```
?- likes(john, mary), likes(mary, john) .
```

“，” 即 “and” 的意思，还可用 “，” 连接多个目标来回答一个询问。当对 Prolog 询问一系列用 “，” 隔开的目标，Prolog 将一个个去寻找匹配，所有目标都满足时，才回答 yes。如上面的例子，其中第一个目标回答是 yes，而第二个是 no，所以整个询问的答复是 no。

连接和变量可以结合使用，回答如下的询问：

“Is these anything that John and Mary both like?”

这个询问也含有两个目标：

首先找出，Mary 喜欢什么；然后，找出 John 是否也喜欢它。

在 Prolog 中把两个目标写成如下形式：

```
?- likes(mary, X), likes(john, X) .
```

Prolog 先回答第一个目标，如果能满足第一个目标，再去看是否满足第二个目标；如果第二个目标也满足，那么在数据库中记下目标的位置，并且找到了询问的回答。每个目标一定有其自己的位置指针，如果第二个目标没被满足，便重新满足前一个目标即指针继续往下找第一个目标答案，记住每个目标进行第二次查寻都是从其各自的位置指针开始，而不是从数据库首部开始，除非后面的对象位置指针已指向数据库的末尾。Prolog 回答以上的 “Is anything liked by Mary also liked John” 询问的过程如下：

(1) 在数据库中查寻第一个事实，此时未定义的变量X是第二个自变量，最先匹配的事实是 likes(mary, food)，即无论何处的X都是 food 意义。Prolog 记下此事实的位置，以便再查寻此事实时回到此处。从此后的X就代表 food，直到下次需要重新匹配时才能改变。

(2) 下面要查寻 likes(john, food)。因为下一个目标是 likes(john, X)，而X即 food，可以看到数据库没有此事实，即此目标没被满足。

当一个目标没被满足时，必须重新回答它的前一个目标。所以 Prolog 重新查 likes(mary, X)，记住这次是从它的位置指针开始向下查，此时X又作为无定义的变量了。

(3) 从上次的位置指针 likes(mary, food) 向下查，下一个匹配的是 likes (mary, wine)，记下位置，此时X意即 wine。

(4) 象第二步一样，Prolog 现在寻找 likes(john, wine)。记住，这时要从数据库的开始处查起。这次找到了匹配事实，Prolog 输出结果（回答）。此时，因为此目标被满足，Prolog 也记下它的位置。Prolog 使每个目标都有一个位置指针。

(5) 在这时，两个目标都被满足，X意为 wine。第一目标位置指针指向 likes(mary, wine)，第二个指向 likes(john, wine)。

象回答其他询问一样，此时 Prolog 回答一个答案后，等待下一个命令。如果按";"键，Prolog 将寻找 John 和 Mary 喜欢的另一件事物。寻找时从各自的位置指针开始。

总之，我们可以想象从左到右用"，"分隔的目标链，每个目标有一个左邻，一个右邻（当然，最左边的那个目标无左邻；而最右边那个目标无右邻）。处理这些目标时，Prolog 按次序从左到右依次去满足，满足一个，就留下一个位置指针。这可以用一个从目标到匹配位置的箭头（指针）表示。此外：

1) 在第一步里一些原来无定义的变量现在有了定义。如果一个变量有意义，到此询问出现的所有这个变量都有了相应值。

2) Prolog 要去满足此目标右邻，从数据库始端查找，当每个目标被满足时，都留下它的位置（从目标到匹配事实画一箭头）。

3) 当目标没被满足，Prolog 返回重新满足它的右邻，查寻时是从位置指针开始。

如果有一个目标总不能被满足时，则此链（指全部目标）不能说得到满足。

Prolog 反复地去重新满足链中的目标称为"回溯"，这个概念以后要详细介绍。

下面是上例运行过程的图示，有以上描述便很容易理解图 1.4a。

以上可以看出此例中回溯的位置和作用。回溯是非常重要的，将在第四章介绍。

练习1.1 对上例，敲入分号";"，即求"Mary 和 John 还共同喜欢何物"，画出示意图。

1.5 规　则

要表示"John 喜欢所有的人"这一事实，可以写成以下的独立事实形式：

```
likes(john, alfred) .
likes(john, bertrand) .
likes(john, charles) .
likes(john, david) .
...
```

每个人都得写进去（即使有成万人）这是很繁琐也不实际的，我们可以用另一种方法表示："John likes any object provided it is a person"。

这是关于"John 所喜欢的"一个规则形式，它代替了一列清单，而比一列清单要简单得多。

在 Prolog 中，当一个事实依赖于另一组事实时，就用规则。在自然语言中用"如

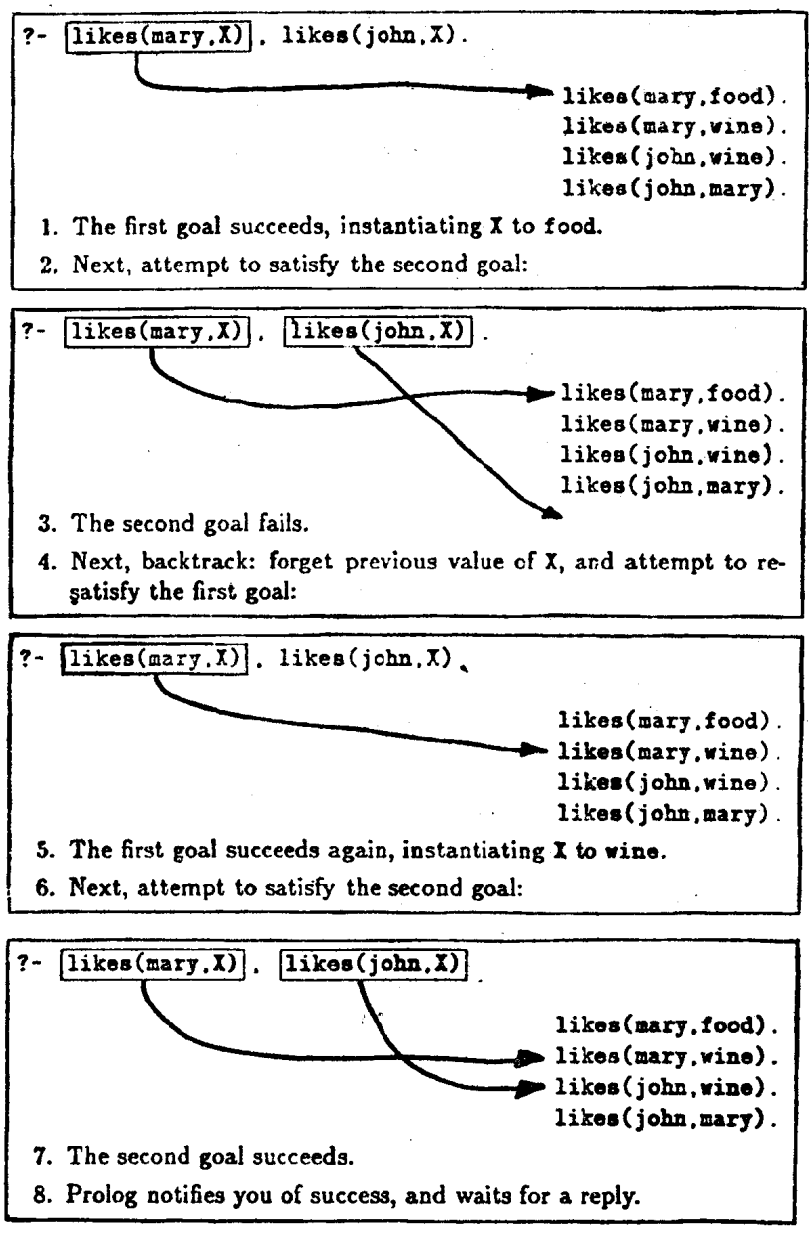


图 1.4 a

果”表示规则。如：

如果下雨，我们就用雨伞。

如果果酒比啤酒便宜，约翰就买果酒。

规则也可用于定义，例如X是鸟，如果：X是动物，并且X有羽毛。

或者，X是Y的姐妹，如果：X是女的，并且X和Y有相同的父母。

以上的定义中用了两个变量，记住每个变量始终代表一个相同的对象。另外定义可以不符合生活中的逻辑，如“因为彼得是动物，玛莉有羽毛，所以弗莱德是鸟”。这个定义是成立的，是符合 Prolog 规则的。

规则是关于对象及其关系的一般陈述句。如：我们可以说：“因为弗莱德是动物，且

有羽毛，所以是鸟”，也可以说：“因为彼查英是动物，且有羽毛，所以是鸟”。

不同规则中的变量可有不同的值。但是在一个规则中每个变量只能具有一个意义，不能有二义性。

下面是一个变量规则的例子：

“John likes anyone who likes wine”

或者 “John likes something if it likes wine”

或用变量

“John likes X if X likes wine”

Prolog 中，规则分为头部和体部两部分，它们用冒号和连字符“:-”连接起来，“:-”意为“如果”。上面的例子在 Prolog 中写成：

```
likes(john, X):- likes(X, wine) .
```

注意规则也是以“.”结束的，这个规则的头部是 likes(john, X).一个规则的头部描述这个规则要定义的事实。体部，此例为 likes(X, wine)，是一些必须满足目标的连接，只有它们被满足，头部才能为真。如：

```
likes(john, X) :-likes(X, wine), likes(X, food) .
```

意即 “John likes anyone who likes wine and food”。

或者“John 喜欢喝酒的女孩”：

```
likes(john, X):- female(X), likes(X, wine) .
```

看一个规则时，要注意变量出现的位置，何时被赋值以及变量的范围。有些特殊规则，从头部到结束都含有变量。上例中如果X被赋“Mary”，则 Prolog 将去查寻 female(mary) 和 likes(mary, wine).

下面举一个多变量规则的例子，此例叙述的是关于维多利亚女王家庭的事。这里用了一个三个自变量的谓词，即 parents(X, Y, Z)，意为：

“The parents of X are Y and Z.”
with Y being the mother, and
with Z being the father .

当然也可以利用谓词 female 和 male。下面是一部分数据库.

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

下面我们还要用以前说过的姐妹的规则，表示为 sister_of(X, Y) 即如果X是

Y的姐妹，则此式为真（注意“_”是下连线，与左右字母底部平，不同于写在中间的连接符“-”），此规则意义：

X是Y的姐妹，如果：

(1) X是女的；

(2) X母亲是M，父亲是F；

(3) Y有相同的父母。

表示成 Prolog 规则如下：

```
sister_of(X, Y) :-
        female(X) ,
        parents(X, M, F) ,
        parents(Y, M, F) .
```

用了变量M和F表示“母亲”和“父亲”。注意这两个变量在头部没出现，但与其他变量一样，开始无定义，匹配后意义固定。

对以上所给数据库和规则进行询问：

```
?- sister_of(alice, edward) .
```

Prolog 工作顺序如下：

(1) 首先此询问与上面规则的头部匹配，从而X代表 alice，Y代表edward，位置指针置此。在下面 Prolog 去寻找体部的三个目标。

(2) 第一个目标是 female(alice)（因为此时X代表 alice)。它从事实中得以匹配，位置指针记下位置（上例第三个事实)。Prolog 去满足下一个目标。

(3) Prolog 寻找 parents(alice, M, F)，而M和F无意义，所以可与任何事实比较，匹配事实为 parents(alice, victoria, albert)，此目标被满足后，记下位置（第六个事实)，此后M即 victoria，F即 albert，下面满足下一个目标。

(4) Prolog 查找 parents(edward, victoria, albert)，(因为Y在第一步里已表示 edward，M和F也被赋为 victoria, albert)，此目标被满足(第五个事实)。这最后一个目标也满足了，所以事实 sister_of(alice, edward) 为真，Prolog 回答“yes”。

假设现在知道 Alice 是姐妹之一，寻找另一姐妹：?-sister_of(alice, X) .

Prolog 作如下工作：

(1) 询问与 sister_of 规则头部匹配，X代表 alice。此时询问中的X与规则中的Y是一个意义，都未定义。

(2) 第一个目标：female(alice) 象上面一样被满足了。

(3) 第二个目标 parents(alice, M, F)，匹配于 parents(alice, victoria, albert)，此时M和F就知道了自己的确切解。

(4) 第三个目标：parents(Y, victoria, albert) 与 parents(edward, victoria, albert)，变量Y也被赋值了。

(5) 所有目标都被满足了，且X代表 alice，Y代表 edward，Y与询问中的X同意义，所以 Prolog 回答：“X = edward”。与以往相同，此时 Prolog 等待下一个命