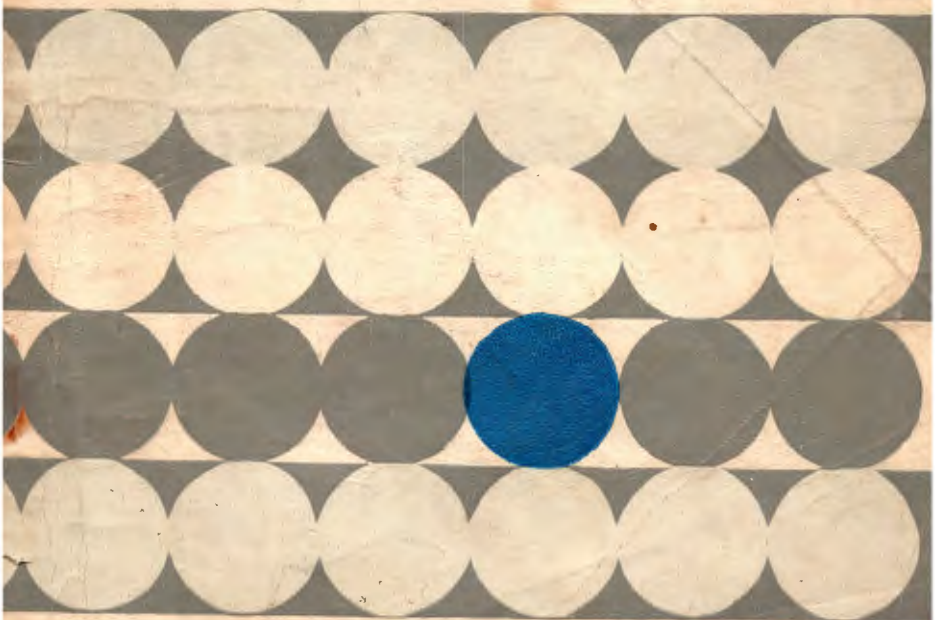
“中国教育电视”播放继续教育节目用书

PROLOG

语言教程



ISBN 7-5027-0044-7/T P·1
统一书号：17193·0975 ¥：1.00元

“中国教育电视”播放继续教育节目用书

PROLOG语言教程

纪有奎　冀　翼　编

海洋出版社
1987年·北京

内 容 简 介

本书是为"中国教育电视"通过卫星播放PROLOG语言课程而编写的。全书共分七章，包括PROLOG语言的特点和它的三个基本语句，自动搜索和回溯，数据结构和递归，算术运算，内部谓词，实例程序和附录。本书内容丰富、通俗易懂，通过若干例题引出概念，便于初学者掌握。凡持有本书的读者在收看电视课时均不需要再抄笔记及屏幕上显示的各种图表资料，课后只要翻阅本书即可复习掌握完整的讲授内容。本书也非常适于PROLOG语言的自学者阅读或作为有关专业教材。

责任编辑　刘莉蕾
责任校对　刘兴昌

"中国教育电视"播放继续教育节目用书
PROLOG语言教程
纪有奎　冀　翔　编
*
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　　外文印刷厂印刷
开本：787×1092 1/32　印张：3 7/8　字数：91千字
1987年4月第一版　1987年4月第一次印刷
印数：1—10000
*
ISBN 7-5027-0044-7/TP·1
统一书号：17193·0975　¥：1.00元

前　言

PROLOG 语言是出现在计算机程序设计语言领域里的一颗新星（英文 PROgramming LOGic 的缩写，即逻辑程序设计）。它具有自动搜索、模式匹配、回溯等独特功能。有人把它和常规的程序设计高级语言相比时，把它叫作超高级语言。逻辑推理是它的特长。

它应用范围广泛，适于非数值处理，应用管理、咨询程序，关系数据库，定理证明，自然语言处理，专家系统和人工智能等许多领域。

它接近于自然语言，具有极强的描述和解题功能。但令人吃惊的是，它只有三个基本语句，它的语法比任何惯用的常规语言都要简单，这要归功于一阶逻辑表示法的简洁和严谨，并具有人机对话的功能。由于它的语法简单，导致直接实现也十分简单。由于这一切，博得人们的欣赏和赞叹，因此PROLOG的推广和传播都较快。

1973年左右，它诞生在法国马赛大学。八十年代初，对它的呼声日高，先后在匈牙利、美国、法国等召开了几次国际性学术交流会。尤其在1982年，日本公布把它作为第五代计算机的核心语言，更引起世界有关人士的关注。一些国家在大、中、小学开设此课，目前它正在许多国家迅速传播。

我国有关方面也很重视它，不少地方和单位曾举办过PROLOG学习班，有关大专院校相继开设此课，北京的有

关单位还为部分中学师生举办过讲座。许多科研单位、大专院校在应用、开发中取得许多成果，这更引起我国有关方面的重视。1985年10月和1986年12月，先后在湖南省大庸和湖北省武汉市，召开了有关PROLOG语言的全国性学术交流会。现在中央广播电视大学决定播放PROLOG语言，这不仅是满足已毕业者继续教育的需要，而且还可满足在校的师生，计算机工作者，科研、管理人员以及广大计算机爱好者等其他人员的需要。

本书是“中国教育电视”卫星频道①播放继续教育节目录相时的讲稿，是收看者必备书。利用参考书②上机、做练习题等。为了缩短学习时间，激发学习兴趣，本书是以例题为主，从例题中引出术语、概念、工作方式等进行讲解。本书由浅入深，力求通俗易懂，因此本书也适于自学者；前四章是基础知识，在此基础上进入第六章，该章具有内容广泛、丰富多样的36个实例题，从而开阔读者的视野，启发应用领域。

中国科学院电工研究所杨正林副研究员，中国人工智能学会理事冯锡福副教授对本书提出许多宝贵意见；南京大学计算机系金志权、陈珮珮同志，武汉测绘科技大学张文星等同志提供部分资料，在此一并表示衷心感谢。由于我们时间仓促，水平有限，难免有误，敬请广大读者指正。

编者　1987年2月

①电视收看的卫星频道，各地不一样，如北京是21频道，要注意播放时间和当地的频道。

②PROLOG语言存在不同的文本，只要学会一种文本，很易转换成其他文本。本书是通用性较强的核心PROLOG文本，书中介绍了如何把它转换成《微型机micro-PROLOG语言及其应用》的文本(海洋出版社出版)——用来上机(IBM PC机)做练习题等，并作为参考书。

目　录

第一章　PROLOG语言的特点和它的三个基本语句

第一节　PROLOG语言的特点

PROLOG作为程序设计语言，它具有两方面的特性，一是它描述求解问题的方式，二是语言本身的特点。

目前，被人们广泛采用的程序设计语言，大致可分为三类：

第一类是过程式语言，如：BASIC，FORTRAN，PASCAL。用这些常规语言解决问题时，需程序员指明算法，需要指定计算机执行的步骤，也就是要告诉计算机“如何做”；

第二类是逻辑型的PROLOG语言，是非过程式语言。它在求解问题时，要求编程序员描述对象之间的关系的事实和规则。它强调对象之间的逻辑关系，程序员一般不必告诉计算机运算执行的先后顺序，就能在数据库中自动搜索、模式匹配和回溯来求解问题。人们把BASIC，FORTRAN，PASCAL 等常规语言叫做高级语言。PROLOG 语言能够描述问题本身，而不必像高级语言求解问题时，需要详细告诉计算机“如何做”，因此有人把PROLOG语言叫做超高级语言。它是人工智能程序设计语言。

第三类是函数型语言LISP，也是人工智能程序设计语言，它比PROLOG语言出现的早。

第二节　PROLOG语言的三个基本语句

PROLOG语言仅有三个基本语句，它们是：事实，规则，询问。以下用例题分别介绍。

例1.1　比尔和玛丽

EX1.PRO

(1) 事实：

① likes(mary, food).	喜欢(玛丽，食品)。
② likes(mary, wine).	喜欢(玛丽，酒)。
③ likes(bill, wine).	喜欢(比尔，酒)。
④ likes(bill, mary).	喜欢(比尔，玛丽)。
⑤ female(mary).	女性(玛丽)。

(2) 规则：

⑥ look-for(bill,X):-female(X),likes(bill,X).

(3) 询问——寻找求解答案：

```
?-likes(mary, X).
X=food;      (表示按";"键)
X=wine
?-look_for(bill, X).
X=mary
?-likes(bill, mary), likes(mary, bill).
no
```

(注意：PROLOG程序没有语句标号，为了方便解释执行过程，在语句前编者自加的①—⑥标号，余同)。

以下分别加以解释：

(1) 事实——是自己定义的，它类似于一个简单句，在运行时，它是搜索的目标，它由关系名如①句中"喜欢"(li-

kes)开始，其后用圆括号括起的是参量组，如①中的(mary food)是两个参量，又叫对象(或个体)。关系名likes使这两个互不相干的孤立对象产生了联系，即玛丽喜欢食品，句子最后置一圆点句号“.”(右下位置)，表示该句结束。

(2) 规则——它由几个目标组成，如⑥中有三个目标。其中符号“:-”表示if (假如)，“:-”左端是规则的头，是总目标的结论，其右端是规则体。规则体中有两个子目标，且被逗号“,”隔开。该逗号是and，即“与”的关系，也就是假如各子目标都成功时，规则头的结论成立。如果一定要和常规语言对比，若把事实比作简单句，那么规则便类似于条件句。

X为变量(第一个大写的英文字母其后可带字符串，即为变量)。

⑥句的意思：比尔寻找的人，第一是女性，第二是比尔喜欢她；或说假若X是女性，并且比尔喜欢她，这便是比尔寻找的X。

把事实与规则键入计算机，便是PROLOG程序，我们把它叫做数据库。

(3) 询问——句首符号为“?-”者便是询问语句。它提出求解的目标，在数据库里搜索找出答案，其过程如下：

若询问 ?-likes(mary, X).

在数据库里自顶向下寻找变量X的解答，首先搜索①句，询问句的关系名与①句的关系名相同(都是likes)，圆括号中都有两个参量，第一个参量都是mary，第二个参量位置上的X便被例示为food，这叫模式匹配(简称匹配)。这时显示屏显示出X=food。因是与①句匹配，要在①句上置目标位置

的标志(可叫作指针，余同)，若试图再寻找一个答案，键入";"再按回车键，这时指针由语句①，下移至语句②，询问句与语句②匹配成功，X被例示为wine。

若询问 ?-look_for(bill, X)。

这询问和⑥规则头匹配，而X仍未被例示。这时再去规则体，⑥规则体中第一个子目标female(X)与⑤匹配成功，X被例示为mary，传递给第二个子目标likes(bill, X)中的X。这时X被例示为mary，即likes(bill, mary)，是否有此事实，还要在数据库里自顶向下搜索检测。当搜索至第④句时，恰好与④事实相同，检测正确，说明⑥规则的第二个子目标成功，由⑥中第一、二个子目标里得出X=mary。这时从右至左即逆向把例示的X传递给⑥规则头look_for(bill, X)，规则头变量X被例示为mary，因此询问句中X被例示为mary。此时显示屏显示出X=mary。

若询问 ?-likes(bill, mary), likes(mary, bill)。

此询问句有两个子目标，第一个子目标与④句匹配（有此事实），第二个子目标在搜索数据库过程中找不到有此事实，询问句里有一个目标不能成立时，询问失败，最后显示屏显示出：no。

术语解释：

(1) 关系名——又叫谓词，如英文句子We study English(我们学习英语)，三个单词在句中成分分别是主、谓、宾。若把谓词study放在句首便成为PROLOG语句：study(we, english)。把谓词放在句首叫前缀(本书用前缀形式表示，参考书中可用前缀、中缀、后缀表示，详见参考书第零章0.1节)，PROLOG谓词不仅用动词，而且可以用形容词，

名词以及短语等等。

(2) 对象——又叫参量(或称变元、自变量、个体)。

(3) 模式匹配(简称匹配)。一个目标(或事实)与另一个目标(或事实)匹配的条件是:

1) 谓词相同;

2) 参量的个数相同;

3) 具备上述1)和2),参量所在位置上的量才允许检测能否对应匹配,如:

```
father(bill, X).        bill(比尔)是X的父亲
father(Y,john).         Y是john(约翰)的父亲
```

这两个事实匹配后: X=john, Y=bill.

(4) 事实和规则又统称为子句(规则是有头、有体的子句,事实是仅有头而无体的子句)。

例1.2 请参看本书附录把例1.1转换成micro-PROLOG程序

本书PROLOG语法	micro-PROLOG语法
likes(mary, food).	likes(mary food)
likes(mary, wine).	likes(mary wine)
likes(bill, wine).	likes(bill wine)
likes(bill, mary).	likes(bill mary)
look_for(bill,X):-female(X), likes(bill, X).	look-for(bill X) if female(X) and likes(bill X)

转化成micro-PROLOG程序后,请用本书7.2节上机操作。

例1.3 捉贼

① thief(bill).

② likes(mary, food).

③ likes(mary, wine).

④ likes(bill, X):-likes(X, wine).

⑤ steal(X, Y):-thief(X), likes(X, Y).

若询问 ?-steal(bill, X).

在数据库中搜索匹配过程如下：

(1) 询问句与⑤规则头匹配，⑤头部 Y与询问句X都是变量。由于位置相同，因此叫变量共享，若其中一个变量被例示，另一变量也获得同样的例示。⑤头部另一变量 X被例示为bill，传递至⑤第一个子目标得 thief(bill)，检测时与①符合。再把例示的X传递给⑤的第二个子目标中的X。

(2) 这时⑤的第二个子目标为likes(bill,Y)，它首先与④匹配，X和Y变量共享，④的体likes(X,wine)又与③匹配得出X=mary。

(3) 上述(2)已使④的X被例示为mary。这时与它匹配的⑤中第二个子目标里Y也被例示为mary，再由右至左传递至⑤的头部Y(也例示为mary)。

(4) 由于询问中X是与⑤头部Y变量共享，这时由于Y被例示为mary，使得X也被例示为mary，所以显示屏出现：

X=mary

即bill偷mary.这个结论是PROLOG推理得出的,前提是bill是贼，另外，bill喜欢喝酒的那个人。

附注：谓词拥有一个参量的是一元，拥有两个参量的是二元，以上介绍了一元和二元的关系。当然也可有更多的元，如三元：

parent(F,M,X).即F和M是X的双亲。

gives(bill,mary,book).即比尔给玛丽书。

参量所在的位置。是用户自己布置的顺序，一旦约定后，程序中每个子句全应按一致的约定来布置，不能自相矛盾而不统一。

习　题

1. 若对例1.1的程序提出如下询问：

（1）？-likes(bill,X).

（2）？-likes(X,wine).

（3）？-female(bill).

试自我模拟程序运行，给出每个询问的回答。

2. 若对例1.2程序提出如下询问：

（1）？-steal(mary,X).

（2）？-likes(X,Y),likes(X,wine).

试在指定的数据库里搜索匹配，给出询问的答案

第二章　自动搜索和回溯

在讲过的例子中已牵涉到自动搜索、模式匹配和回溯，再用例子加以补充和阐述，前面所举的例子都是用英语，其字母或字符都是键盘上具备的。因此，我们可改用汉语拼音编写事实、规则和询问，或英语单词和汉语拼音单字混合使用，程序员约定即可。这样可把PROLOG语言推广到只具备粗浅英语知识的读者中。

第一节　用例题来阐述自动搜索和回溯

例2.1　爱情关系数据库(请读者把中文改为汉语拼音)

①　男性(赵).

②　女性(钱).

③　女性(孙).

④　喜欢(赵，钱).

⑤　喜欢(赵，孙).

⑥　喜欢(孙，赵).

A　　B　　C

⑦　相爱(X,Y):-男性(X),女性(Y),喜欢(X,Y),

D

喜欢(Y,X).

⑧　?-相爱(赵，X).

当键入询问⑧时，在数据库里至顶向下，从左到右自动

搜索。若规则中从左到右正向搜索失败时，需从右到左逆向搜索，这叫回溯（“溯”字“逆”的意思，倒退之意）。其执行过程如下：

⑧→与⑦匹配，⑦头部X为赵→⑦A的X为赵→⑦A在数据库中至顶向下搜索匹配(检测)，与①匹配成功(即得到证实)→⑦B在数据库中自顶向下搜索时，首先与②匹配成功，Y被例示为钱→⑦中已被例示的X和Y传递至⑦C→这时⑦C为喜欢(赵，钱)，与④得到证实→再从左至右传递至⑦D，即喜欢(钱，赵)，而搜索验证时，数据库中并无此事实，所以失败，失败则引起回溯：

从⑦D→⑦C，而⑦C中的X，Y仍由其左则传递来的→⑦B中的Y刚才为钱，说明不对，要解脱掉已例示的钱，重新匹配，而上次指针指向②。由于回溯，指针下移与③匹配，这时⑦B的Y为③句的孙。逆向回溯至此，换了个Y的例示(孙)，再正向从左至右验证对否→⑦C为喜欢(赵，孙)与⑤符合，得到证实→⑦D为喜欢（孙，赵）与⑥匹配得到证实，至此已成功。把⑦D的Y为孙→传送给⑦句头部的Y(Y也为孙)→⑧中的X由于和⑦句头部Y共享，所以⑧中X被例示为孙，其结果显示为

X＝孙

用PROLOG推理结果为赵和孙相爱。

例2.2 在舞会上每个男孩都和每个女孩跳一次舞

```
                                    A       B       C
①  wu_ban(X,Y):-boy(X),girl(Y),write(X,
                    D   E
'and',Y),nl,fail.
```

② boy(m1).

③ boy(m2).

④ boy(m3).

⑤ boy(m4).

⑥ girl(w1).

⑦ girl(w2).

⑧ girl(w3).

⑨ ? -wu-ban(X,Y).

解释例2.2程序

(1) 内部谓词——是系统里带的谓词，它的功能已约定好，用户不能改变，又称固有谓词。如：write是输出谓词，可在屏幕上显示出具体内容。用单引号括起来的是按原样显示。

nl是换行谓词，遇到它便自动换一行。它是只有谓词(不带对象)的子目标。

fail是失败，该子目标也仅有此谓词作为子目标，遇到它说明该子目标失败，失败则引起回溯(强制回溯)。

以上内部谓词仅能满足一次，即仅成功一次，一次性的动作，即回溯到它时不再被满足，不再动作，除非再从左至右正向在遇到它时，它可重新被满足，再执行一次动作。

(2) wn_ban是汉语拼音“舞伴”，boy和girl是谓词，表示男孩和女孩，m1—m4是男孩名字，w1—w3是女孩名字。

(3) 第一个子句是规则，规则体有五个子目标，为了分析方便，我们约定该五个子目标从左至右分别用A—E表示。

(4) 自动搜索求解过程如下：

⑨与子句①头部匹配→子句①第一个子目标A与②匹配，X为m1→①B与⑥匹配，Y为w1→①C输出X和Y，即

m1 and w1跳一次舞→①D换行→①E失败，失败则引起回溯→由于n1，write回溯时遇到它们时，它们不动作→继续由右往左回溯至① B，上次①B与⑥匹配，指针指向⑥，这时的指针下移由⑥指向⑦，①B与⑦匹配成功，Y为w2→再正向由左至右传递信息，①C输出m1 and w2跳一次舞→①D换行→①E失败，再次回溯至①B，指针由⑦下移至⑧，Y为w3→①C输出m1 and w3→①D换行→① E失败，当再次回溯至①B时，指针由⑧下移，这时数据库中已无子句，失败了→回溯至①A。以上的经过使①B回溯多次，但① A始终是与②匹配，即指针指向②，由于初次回溯至此。指针下移，由②指向③，X为m2→正向传递信息至①B，① B在数据库中自动至顶向下搜索，首先与⑥匹配。X为 w1→输出、换行、失败，再回溯，如此下去。直至一个指针指向⑤，另一个指向⑧，若再回溯，指针下移至底，便全部结束。

例2.3 人、虎、兔是动物，兔子吃草，人吃兔子，老虎吃不带枪的动物。张三带枪，李四空手。在进行程序设计时允许加添意思。

下面程序是用汉语拼音编写的，请你阅读、分析询问所得的答案是否正确。

```
dongwu(zhangsan,ren).
dongwu(lisi,ren).
dongwu(hu).
dongwu(tu).

kongshou(lisi).
kongshou(tu).
```

```
chi(tu,cao).
chi(hu,X):-kongshou(X).
chi(X,tu):-dongwu(X,ren).
?-chi(X,tu).
X=zhangsan;X=lisi
?-chi(hu,X).
X=lisi;X=tu
```

若下面的询问，将各得到什么答案：

```
?-dongwn(X,ren).
?-dongwn(X).
?-chi(lisi,Y).
```

例2.4 人、虎、兔两两相遇。兔子吃草；人或虎吃兔子；老虎吃不带枪的人；否则，老虎被吃。张三带枪，李四空手，问谁吃谁？

下面是用汉语拼音编写的程序，请你模拟运行该程序，辨别询问后的回答是否正确：

```
ren(zhansan,qing).
ren(lisi,meiqing).

xiangyu(zhansan,hu).
xiangyu(X,tu):-ren(X,Y).
xiangyu(hu,X):-ren(X,meiqing);chi(X,cao).

chi(tu,cao).
chi(X,hu):-xiangyu(X,hu),ren(X,qing).
chi(hu,X):-xiangyu(hu,X).
```

```
chi(X,tu):-xiangyu(X,tu).

?-chi(X,Y).
X=tu,      Y=cao;
X=zhansa,      Y=hu;
X=hu,      Y=lisi;
X=zhansa,      Y=tu;
X=lisi,      Y=tu;
no
```

注：第五个子句体用“;”表示“或”，其左右子目标有一个被满足则成功。

第二节　cut（截断）的应用

前已介绍PROLOG语言本身具有自动搜索和回溯的功能，有时不需要再回溯，可用cut来阻止回溯。cut是截断之意，截断回溯，程序中用惊叹号“！”表示cut。合理而准确地使用cut，是PROLOG程序设计高技巧之一。

合理使用cut对运行时间和存储空间都有益：

（1）若事先能估计出回溯时找出答案，便用cut截断回溯，节省运行时间。

（2）回溯时要保存回溯点的信息，若截断无用的回溯，便可节省计算机存储空间。cut即“!”符号是一个内部谓词，这个目标是没有参量仅有“!”谓词，遇到它时便立即成功，但不能重新被满足，也不能逆向越过它(因它截断了回溯)。

cut常用于三种情况，通过简单例子说明如下：

（1）仅要求一种答案，找到后便停止回溯。

例2.5 在产生器和测试器后面放cut，如：

```
example(X):-generate(X),test(X),! .
```

产生器generate(X)不断地产生一些值，每产生一个值(用X表示)，正向传递给右侧目标 test(X) 进行测试。若满足要求，再正向搜索右侧目标“!”，而“!”特性是正向遇到它时便立即成功，并截断回溯，即产生出的一个目标经测试成功便终止。若generate(X)产生的值经test(X)测试不成功，便回溯至generate(X)再产生另一个值，直至成功经“!”后而终止；

(2) cut与另一个内部谓词fail联合使用，说明徒劳搜索。

例2.6 假设一个人没有肝炎，或没有胃病，或没有其他疾病，便是健康的人。

试看以下程序：

```
① healthy(Y):-hepatitis(Y),fial.
② healthy(Y):-stomch_disease(Y),fial.
③ healthy(Y):-other_disease(Y),fial.
④ healthy(Y)
```

子句①企图说明，若Y有肝炎，则匹配它的目标将失败。同样，子句②、③说的是，若Y有胃病或有其他疾病，则Y不健康。而子句④企图说明如果Y没有上述疾病，则此人是健康的。但上述程序是不正确的，若询问李芳(li_fang)健康吗：

```
?-healthy(li_fang).
```

若数据库中有李芳患肝炎的事实：hepatitis(li_fang)，本应询问句与子句①匹配时回答 no，但与子句①第一个子目标匹配成功后接着遇到目标 fail，fail 总是失败，便引起

回溯，必然又先后与子句②，③匹配，与②，③匹配也宣告失败。最后与子句④匹配成功，说明李芳健康，而这结论是错误的。原因是由于 PROLOG 回溯功能而引起的结果，为了截断这种不必要的回溯，把cut放在fail 之前，把例2.6程序改写为例2.7。

例2.7 把cut加入到例2.6程序里

```
                    A          B  C
①  healthy(Y):-hepatitis(Y),!,fial.
②  healthy(Y):-stomach_disease(Y),!,fial.
③  healthy(Y):-other_disease(Y),!,fial.
④  healthy(Y).
```

若再询问?-hepatitis(li_fang).

该询问与子句①匹配时，由于谓词 fail立即失败，引起回溯至“!”，而它截断了往左子目标的回溯，失败了又无法回溯，显示屏显示no，即李芳不健康，此回答正确；由此可见，因为cut和fail联合使用，改变了回溯方式，不仅提高了搜索速度，而且推理正确(即寻找健康者是徒劳的)。

(3) 有条件的选择

例2.8 数据库中有如下规则：

```
①  h(X,Y):-A,B,! ,C,D.
②  h(X,Y):-E,F.
```

若询问?-h(S,T).

首先与规则①头部匹配，并在逐个检测过程中A,B之间的回溯不受任何限制，当B成功，“!”也立即成功。这时已冻结A,B所做的选择，也就是说，只取它们当前的选择，其他的选择即使存在也不予考虑。接着检测C，若C失败，则不

许越过“！”向左回溯(因为“！”截断回溯)，于是整个目标失败；若C成功，则检测D。在C，D之间回溯不受限制，当子目标C失败，整个目标便失败了。此时，即使还另有与其头部匹配的目标，也不能再试探匹配。

cut的作用相当于形式“if…then…eles”，用此形式来表达上述规则，就是：

```
h(X,Y):-if A,B then C,D eles E,F
```

再介绍一个内部谓词not，它是否定之意。它与cut联用时要理解其用意，如：

例2.9 cut和file联用来表达命题X为非即not(X)

```
①  not(X):-X,! ,fial.
②  not(X).
```

同一件事，它只能为真或假，不能二者具备。上例若X为假，规则①的第一个子目标X不成立，则回溯。由于左侧再无子目标可回溯，便与事实②匹配。因X为假，则not(X)为真，便成功了。若X为真，与规则①第一个子目标检测成功，接着“!”也成功，而第三个子目标fail失败，回溯时被“!”截断，故总目标失败，即X为真时便失败。

第三节 PROLOG系统内部控制和用户控制

前已叙述，PROLOG系统能自动搜索，自动模式匹配和回溯，这便把人工智能系统中最基本的操作和实现技术加入了PROLOG实现系统内部，从而高度简化了求解问题的表达。

通过前面的学习，已经展示出PROLOG程序不同于常规语言的程序，它没有常规语言里的条件，循环和转向等控

制，它仅具有三个语句——事实、规则和询问。PROLOG系统以用户在定义求解问题时已知的事实和有关规则作为前提，自动地进行搜索查找，判断和推理，最后回答用户提出的问题；因此，PROLOG把实现搜索匹配和回溯的控制操作放进了系统内部，而显现在用户一级上的控制已是更高一级的成分，例如用户规则中的递归，用cut限制搜索查找过程中的回溯。cut操作是用户用来告诉PROLOG系统如何改进(加速)搜索或回溯进程，从而避免那些不必要的徒劳搜索。既节约运行时间，又节省存储空间。

习　题

1．PROLOG擅于逻辑推理，当然可用于三段论法．如提出古代哲学家苏格拉底会死吗？

编写程序的思路是：人总是要死的，苏格拉底是人，所以他也要死的。如下定义一个规则和一个事便可。其程序为

```
总有一死(X)：-人(X).
人(苏格拉底).
？-总有一死(苏格拉底).
yes
```

请你参考上述示例，编写几题三段论法的程序。

2．练习如何定义亲友之间的关系

例如，母亲的定义可写规则为

```
mother(X, Y)：-parent(X, Y), female(Y).
```

X的双亲是Y，Y是女性，则X的母亲是Y。也就是说：母亲是父母辈的女性。

请你根据上述示例，写出下列规则的含义：

```
wife(X, Y)：-father(Z, X), mather(Z, Y).
father(X, Y)：-parent(X, Y), male(Y).
grandfather(X, Z)：-father(X, Y), father(Y, Z).
```

grandmother(X, Z) : -mother(X, Y), mother(Y, Z).

请你再定义：

姐妹关系

兄弟关系

伯父和伯母

孙子和孙女

3．若例2.3的询问？-chi(hu, X)，只要一个答案，应把cut即“!”加在何处？

第三章　数据结构和递归

PROLOG 程序都是由项构造而成的。PROLOG 基本数据结构是树和表；递归是 PROLOG程序重要特性，以下分别介绍。

第一节　项

项
- 常量
  - 原子
  - 常数
- 变量
  - 有名变量
  - 无名变量
- (项)——是项里有项，称为复合项，又叫结构

以下分别简介：

(一) 常量

(1) 原子——用来标识对象的名字，以及谓词名等。原子是以小写字母开头的字符串，中间可带有下连线符号"_"。如 look_for (bill,mary) 是一个事实，它是由三个项(即三个常量、也就是三个原子组成)。把原子可粗略理解为不能再分的一个基本单位，即不能把 mary(玛丽)分为ma 和 ry 等。

(2) 常数——早期文本仅限于整数，后来的PROLOG文本出现了实数。

(二) 变量

(1) 有名变量——第一个英文字母大写，其后可跟字符串，如X，Xa，Xa2。

(2) 无名变量——用下连线"_"表示，因为它的名字从来不会被使用，是名义上的变量，仅占用一个位置，不关心其内容具体如何，是哑元。在同一个子句中的几个无名变量，不需要给出一致的解释，这是它的特性。

(三) 带括号的项

这是项里有项，称为复合项(又叫结构)。复合项突破了事实所表达信息量的局限性，可视为事实的扩展成分。如：

owns(mary, book)。这是事实(简单句)玛丽有书。

owns(mary,book(prolog,auther(clocksin, mellish)))。这是扩展了的事实，下面暂称复合项，是结构，它不仅表达了玛丽有书，而且还表示了书的名字及作者。

有位在婚姻介绍所工作的人，学到PROLOG复合项时，他联想到大龄男女找对象就是匹配的过程，他写了如下复合项作为练习：

女(年龄，个人(政，职，号，父(有否，状况)，母(有否，状况))).

男(年龄，个人(政，职，号，父(有否，状况)，母(有否．状况))).

?-女(A，个人(B，C，D，父(_，_)，母(E,_))).

他写复合项是项里有项，嵌套多层，容纳较多信息。把一个个复合项键入计算机时，便形成数据库。若男方提出如上所示的询问，PROLOG便在数据库里自顶向下搜索，寻找匹配，变量A，B，C，D，E将被例示出具体内容，供选择参考。当然，越是要求条件不多，变量个数也就越少，

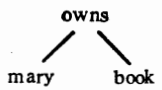
就越容易匹配成功。

请你指出，这位婚姻介绍所工作者上述讲法对否？

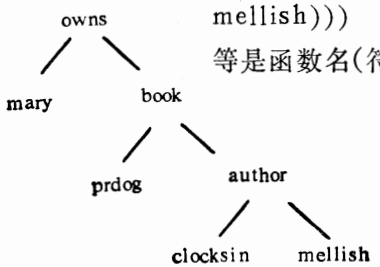
第二节　树

若把一个事实，把一个复合项用一棵树来表示　二者用树来形象对比将更容易理解复合项(结构)，可看出哪个是根、子根、叶结点。例如

①　owns(mary,book)



②　owns(mary,book(prolog,author(clocksin,mellish)))　book 和 author 等是函数名(符)，是子根。



第三节　表

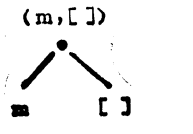
表是非数值程序设计中一种最常用的数据结构，非常有用。可以把表看为一种特殊的复合项，即一种特殊的结构。表中各元素是有序排列的，不能随意颠倒。表中的元素可以是原子、结构或任何其他项(包括变量和复合项)，也可以是另一个表，可见表是递归定义的。表能够表示人们在符号处理中希望使用的任一类结构。

表用“[ ]”表示，如[a，b，c]表里有三个元素依次为a，b，c。若表中没有元素便是空表“[]”，表可以嵌套，即表中表，如[a，b，[1，2]，[]]。

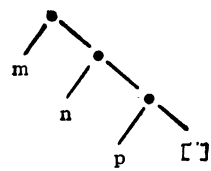
一个表可分表头和表尾，一个表中第一个元素是表头，其余的是表尾，表尾仍是一个表，如[a，b，c]表里，a是表头，是一个元素，其余的[b，c]仍是一个表，不断地把表头取出，最后表尾是空表。(注意，表头不要和规则头相混，各不相干，概念也不同。)

既然表是一种特殊的结构，自然也可用树结构来表示，表是以树的一种特殊情况的面目出现的。如：

把“.”视为函数符(函数名)，m为原子。

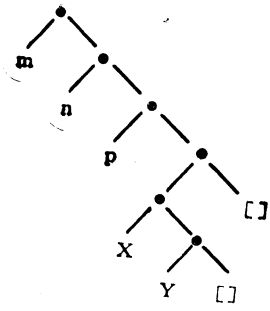


(m,·(n,·(p,[ ])))



在表中包含表或变量是非常有用的，将给我们编写程序带来极大的方便.

如上述·(m，·(n，·(p，[]))),实际它是表[m,n,p]。若表为[m，n，p，[X，Y]]，用树结构可表示为



通常用竖杆"|"把表头和表尾隔开。如：[X|Y]表里X为表头，Y是表尾。如把表[a，b，c，d]用"|"分成表头和表尾，便是[a|[b，c，d]]，其中a元素是表头，表尾是[b，c，d]，可见表尾仍是一个表，若表尾没有元素，便用空表"[]"表示。

第四节　PROLOG的递归

通俗形象来说递归性，是自己调用自己。(即递归目标到最后又把自身作为递归目标)如右图所示，A，B，C为三个目标，若想解决A，需要求助于B(如箭头所示的方向，余同)，而B又求助于C，C还要求助于A，又回到原始的A目标，相当于A递归调用自己。每递归调用一次(循环一次)，是一个层次，若A再递归调用一次，是第二个层次，如此下去，一次次增加层次深度，直至获得求解答案为止。

递归既是PROLOG程序重要特性，又是运用PROLOG语言重要技巧之一，它可使PROLOG程序简洁，它可使程序提炼得像古诗那样精炼而表达力丰富。但递归概念使初学

者往往不易理解，程序员可以通过逻辑思想的想像力来理解；还可通过手画跟踪图来理解；最好上机调用跟踪模块，把递归的深度一层层输出，来形象地观察匹配求解的过程，详见参考书。

下面介绍些常用的递归例题。

（一）表的成员关系

如果一个对象与可能包含这个对象的表之间的关系，叫成员关系，其程序是：

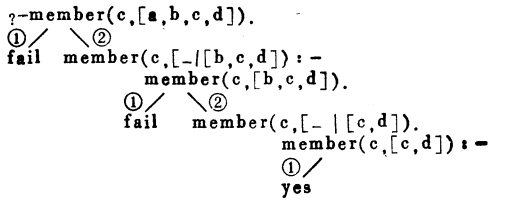
例3.1 表的成员关系

```
① member(X, [X | _]).
② member(X, [_ | Y]):-member(X, Y).
```

①是事实，谓词member拥有两个参量：X和[X | _]。第一个参量即对象X，若在第二个参量即表[X | _]的表头时，则成功，即对象X是表中成员(因为表头便是X)；否则，再看规则②，②的表头第二个参量位置仍是一个表，该表头为无名变量"_"，对它不感兴趣(不检测，去掉即可)，(2)的体中谓词member是递归定义的，其子目标是把②头部中表头去掉，仅剩表尾Y，形式便为member(X, Y)，要检测X是否在表尾Y里，该②的子目标仍是两个参量(X和Y)，它与①匹配时，X都为第一个对象而匹配，②的子目标里第二个对象Y与①的表匹配时，把Y分成一个表头和表尾，再看表头是否是X，若是则成功；否则再进入②的头部，于是又进入②的子目标member(X, Y)，这样递归调用(循环了)一次。如此继续下去，有了答案为止。

若提出询问 ?-member(c, [a, b, c, d])，意思是c是表[a, b, c, d]的成员吗？我们画出手跟踪图来显示求解

过程如下：



上述手跟踪图中的①和②分别表示例 3.1 程序中的①和②两个子句。此图可看出匹配、递归过程，如与①匹配失败时用 fail 表示，最后检测出c是表中的成员(即表 中 有 c)，这时检测成功，回答 yes。上述过程是检测c 若不在[a，b，c，d]的表头，便去掉表头中的一个元素，再检测c是否在表尾。若把表中一个个元素一次次去掉后剩为空表"[]"，便回答no。

又如 ?-member (h, [a, b, c, d, e])，因表中没有 h，将回答no。

学会画手跟踪图，是 PROLOG 学习者基本功之一。当然，复杂的跟踪过程或递归层次过多，需上机察观跟踪的过程，从而加强程序的编写技巧。

(二)表的连接

把两个表连接成一个表，称为表的连 接。如 [a，b]和[c，d]连接成[a，b，c，d]。又如[1,2,3]和[a，b]连接成[1,2,3,a,b]。append 是表连接的谓词。

例3.2 表的连接

① append([], L, L).

② append([X | L1], L2, [X | L3]:—append

(L1,L2, L3).

解释上述程序，把两个表连接成第三个表的规律如下：

(1) 第一个表的第一个元素总是第三个表的第一个元素，如规则②的头部里的X；

(2) 第一个表的尾(L1)与第二个参量L2连接，成为第三个表的表尾(L3)，见规则②的体所示；

(3) 不断从第一个表的余项取其表头元素，最后剩下空表"[]"，如事实①所示，这时便运行结束，所以事实①是边界条件，与①匹配成功便结束。

事实①的第一个参量是空表，空表再加第二个表L便是和第二个表相同的第三个表L。

若询问 ?-append([a, b, c],[1, 2], [a, b, c, 1, 2,]).将回答 yes。

若询问 ?-append([1, 2], [a, b, c], X).将回答 X=[1, 2, a, b, c]

上述第一、二个表连接成第三个表，即共有三个表。若三个表已知任意两个表，便可连接成特定不变的第三个表。若三个表中只知道其中一个表，如只知道结论的第三个表，也可逆序求出其他两个表，这叫PROLOG可逆性程序设计。若只知第三个表来求出其他两个表，而其他两个表有几种可能性——是不固定的，例如：

```
?-append(X, Y, [1, 2, a, b, c]).
X=[], Y=[1, 2, a, b, c];
X=[1], Y=[2, a, b, c];
X=[1, 2], Y=[a, b, c];
X=[1, 2, a], Y=[b, c];
```

X=[1, 2, a, b], Y=[c];

X=[1, 2, a, b, c], Y=[];

no

按分号";"键表示再找一解.

为了画表连接的手跟踪图，把例3.2程序抄如下：

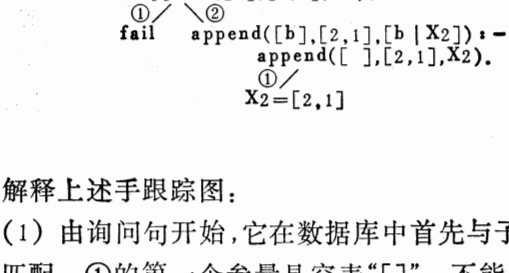
① append([], L, L).

② append([X | L1], L2, [X | L3]:-append (L1, L2, L3).

当询问?-append([a, b], [2, 1], X).时，将回答：

X=[a, b, 2, 1]

以下手跟踪图说明其自动搜索、匹配、递归的过程：



解释上述手跟踪图：

(1) 由询问句开始，它在数据库中首先与子句①的对应参量匹配，①的第一个参量是空表"[]"，不能与询问句的第一个参量[a, b]匹配而失败，即图中第一个左分支(斜线旁写①)在下方写fail(失败)，指针便指向子句②。

(2) 指针指向子句②时，即图中右分支(斜线旁写②。②的头部三个表与询问句中三个表对应匹配后，第三个表成为[a | X1]，因其表头应与第一个表的表头相同，故第三

个表的表头也为a。表尾换了一个变量名为X1，即X=[a | X1]，可见询问句里X与X1所表示的内容不同，为了与X区分而改写为X1。子句②的体是一个目标，其谓词member是递归定义的，同上述(1)，即首先与①匹配时而失败，用左斜线表示。因与子句①匹配失败，指针便指向子句②。

(3) 指针又指向子句②，这时子句②的尾部(唯一子目标)的谓词member递归调用到自身。至此递归目标循环一周，是一个层次。

(4) 重复上述(1)至(3)过程，递归层次一次次加深，每层的第三个表的变量名(如X1，X2…)都在变，因表示不同的常量。递归过程直到目标中第一个表为空表"[]"时，便能与边界条件的子句①匹配成功而停止。

(5).停止后要把获得的常量逆向返回。如图中最底层结果为X2=[2.1]，往上返回给②的子目标里的X2，则②的头部第三表[b | X2]=[b | 2，1]。再往上层返回给②的子目标中的X1，即X1=[b | 2，1]，则②的头部第三个表[a | X1]=[a | b，2，1]=[a，b，2，1]。再往回返给询问句的X，因询问句中X开始匹配时，X=[a | X1]，这时X=[a，b，2，1]，即询问句中[a，b]和[2，1]两个表相加成第三个表X已有正确答案。

附注：

(1) 上述(3)中提到在子句②即规则②的尾部的谓词member是递归定义的，凡是在程序里最后一个规则的尾部运用递归定义，便叫尾递归。又如例3.1的规则②的尾部用谓词member来递归定义，也是尾递归。尾递归是编程技巧之一，有许多优点，如可节省堆栈空间等等。

(2).递归定义的程序，几个子句安排的位置顺序是极关重要的，应把边界条件放在首位。如在例3.1和例3.2程序中第一个子句都是边界条件，这样才能无误地运行，获得正确的答案。若将上述子句位置颠倒，即把边界条件置入数据库底部，读者试分析将会有怎样的结果。有的递归程序边界条件不在首位，如例6，11等。

(3)编写程序时，要防止死循环，如下例：

例3.3 死循环举例。

```
①    parent(X, Y):-child(Y, X).
②    child(A,B):-parent(B, A).
```

规则①的头定义X是Y的双亲，其规则体定义Y是X的孩子。同理，规则②的头定义A是B的孩子，其规则体定义B是A的双亲。所有参量都是变量，当然可以对应匹配，变量共享。首先①的体与②的头匹配，然后②的体与①的头匹配，而后①的体又与②的头匹配…，如此下去，将陷入死循环。

例3.4 求Fibonacci(菲伯纳奇)数。

意大利数学家Fibonacci写出的数列：

1,1,2,3,5,8,13,21，…这是一个很著名的数列，其规律是：从第三项开始，以后的每一项都是它的前两项之和。如2＝1＋1，8＝3＋5。公式为$f_n=f_{n-1}+f_{n-2}$ $n\geqslant 2$，首先定义为：$f_0=1$，$f_1=1$。设fib(N,X)表示第N个Fibonacci数是X，则程序为

```
fib(0, 1):- !.      表示第0项为1
fib(1, 1):- !.      表示第1项为1
fib(N, X):-N1 is N-1,N2 isN-2, fib(N1,Z1),
          fib(N2, Z2),X is Z1+Z2.
```

? -fib(5, X). 询问第5项是几，回答：

X=8

第一、二个规则是边界条件。第三个规则是执行一次都在减1、减2，直至减到所余的数能与第一或第二规则匹配成功而结束。特别需要指出，这是递归调用，因第三规则体里有谓词fib，它在数据库中自顶向下搜索、匹配，每当"自己调用自己"时，递归层次加深一层，如此才能达到屡次减1、减2，直至与边界条件符合而停止。请针对上述询问画出手跟踪图。

程序中的is是内部谓词，其左侧的变量被例示为右侧的值，如X is 10+6时，X=16。还可利用is来检测，如询问63 is 7*9时，将回答yes。

习 题

1. 在例3.1程序里，当询问?-member(d, [a, b, c])时；用手画出跟踪图并回答出yes或no而结束。

2. 用例3.2程序，当询问?-append([1, 2, 3], [a, b], X)时，画出手跟踪图并回答两个表连接成的第三个表X是何内容。

3. Fibonacci程序，若询?-fib(6,X)时，画出手跟踪图。若询问？-fib(21,X)时，请上机调用跟踪程序，它会自动打印出搜索、匹配、回溯和递归过程及其结果。

第四章　算术运算

数值计算不是PROLOG语言的目的，它用于逻辑推理，用于非数值程序设计，因此它不能与常规的算法语言相比拟，只提供基本的运算，而且限于整数的算术运算，如DEC-10PROLOG系统。后来出现了参考书micro-PROLOG系统，把算术运算中的数扩展到实数范围。近年来，出现了编译型的PROLOG系统，提高了运行速度，加强了运算功能，出现了算术函数的内部谓词，像常规语言那样具有三角函数、对数函数、指数函数、平方根函数等等。

因本书PROLOG系统的语法，很易转化为其他系统的语法。所以本书不详细介绍各种PROLOG系统的算术运算，仅通用性一般简介。

第一节　算术运算符和比较运算符

（一）算术运算符

一般的PROLOG系统都能提供五种最基本的算术运算：加、减、乘、除和取模。其运算符分别为＋，－，＊，／，mod。

算术运算符的优先级与通常数学上的规定相同。即乘、除、取模优先于加、减。例如：

A+B/C	A+(B/C)	两者一样
X−Y＊W	X−(Y＊W)	两者一样
A+B/C−D	(A+(B/C))−D	两者一样

上面算术表达式中的运算符是作为函数名出现的，可视为内部谓词，用中缀形式表达的，也可写为如下结构形式(前缀形式)：

```
+(A, /(B, C))
-(X, *(Y, W))
-(+(A, /(B, C), D)
```

用中缀形式的表达式或用前缀形式的表达式，由用户自己决定。

在例 3.4 里已介绍过 is,可视为特殊的运算符,它必须以中缀的形式出现，通常其左边是变量,其右边是算术表达式，其一般形式为:变量is算术表达式,把变量例示为右边算术表达式的值，所以表达式中的所有变量的值在求值时必须是已知的。

is 可用来检测，如询问?-8 is 9，将回答no。如询问?-48 is 16*3时,将回答yes。

必须注意，is 绝不是常规语言的赋值概念。如Y被例示为19，若写Y is Y-1，则Y不是18，而是失败的，因为is左侧为19,右侧为18。当询问?-Y is Y-1时,将回答no。又如?-X is 7，X is 50时,第一个目标X被例示为7，第二个目标把已被例示的X为7与50检测时，是不符合的，故失败,回答no。

例4.1 已知四个国家名字a1，a2，a3，a4，又给出这个国家的人口和面积，求其人口密度(人口以百万为单位,面积以百万平方公里为单位)。程序如下：

```
*        人口(a1, 203).  人口是谓词，可改写为英语或汉语拼音。
```

② 人口(a2, 543).

③ 人口(a3, 800).

④ 人口(a4, 108).

⑤ 面积(a1, 3).谓词面积可改写为英语或汉语拼音

⑥ 面积(a2, 1).

⑦ 面积(a3, 4).

⑧ 面积(a4, 3).

⑨ 密度(X, Y):-人口(X, P), 面积(X,A), Yis P/A.

?-密度(a4, X).

X=36

其搜索、匹配的过程是：询问句与数据库中的规则⑨匹配，规则头的变量X被例示为a4，传送给⑨的规则体中第一个子目标：人口(a4, P)，该子目标与子句④匹配，P被例示为108。⑨的第二个子目标中X也为a4，它与子句⑤匹配，A被例示为3。这时再执行⑨的第三个子目标：Y is P/A，由于is右边的表达式中变量P和A已被例示为108和3，所以Y被例示为36，把Y的值传送给⑨的头部，这时⑨的头部Y也为36。而询问句的X与⑨的头部Y是变量共享，所以询问句的X答案为X=36，即a4国家的人口密度为每平方公里36人。

（二）比较运算符

比较运算符是用于描述两个表达式之间的关系的比较，因此，又称为关系符。一般的PROLOG系统提供了六种常用的比较运算符，如下所示：

<	小于
=<	小于等于

=	等于
>	大于
>=	大于等于
\=	不等于(而有的文本用<>或><表示)

例4.2 判断给定的A，B，C三条边是否能构成等边三角形。

```
three_edge(A, B, C):-A>0, A=B, B=C.
?-three_edge(6, 6, 6).  将回答：
yes
?-three_edge(6, 7, 8).  将回答：
no
```

上述六个比较运算符是作为内部谓词，其中"="和"\="不仅能用于常数比较，也能用于其他对象的比较。

对目标X=Y(其中X和Y是任意两个项)，决定X和Y是否相等的规则是：

(1) 两个原子相等　例如：mary=mary，food=food，都是成功的。atom=atom2是失败的(不成立)。

(2) 两个结构相等(即函数符和参量个数都相同时，则对应的参量便可试图匹配，观察相等)。例如：

owns(mary, book)=owns(X, book)

X被例示为mary。

谓词"\="称为不等。如果X=Y失败，则目标X\=Y成功；若X=Y成功，则X\=Y失败。

有的版本还把其他的运算比较符除了用于比较数值外，还可用来比较字符串以及符号等。例如：

a<b, abc>abb, mary>mar。在比较时，PROLOG

把字符转换成ASCII代码来做比较。

第二节　几个典型的例子

用PROLOG编写求自然数列、累加和以及求连乘积的程序，不仅为了数值计算，而且锻炼如何阅读程序和提高编写程序的能力，从而进一步学习cut的应用，以及学习手画跟踪图再次体会程序的递归性。

(一)求自然数列

例4.3　求1至N的自然数列。定义谓词int为整数。

```
①    int(1).                  A          B
②    int(X):-int(Y), X is Y+1.
```

询问?-int(X).

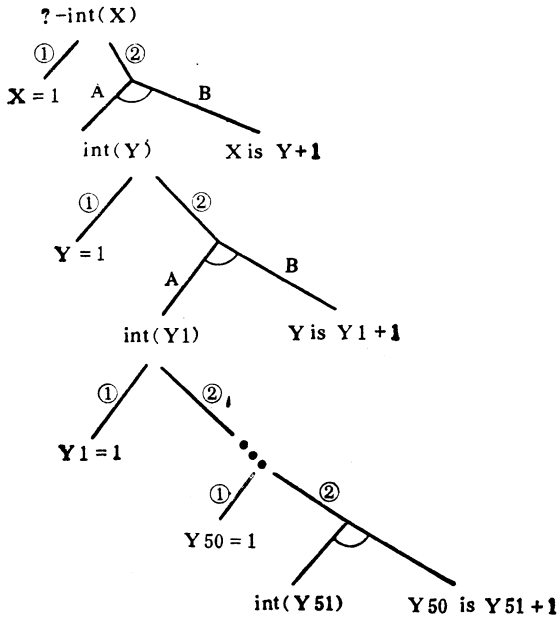
回答X=1;X=2;X=3;…。

回答过程中分号";"表示按此键，再求下一个解，如此下去可以获得无穷多个解，手跟踪图如下所示(用A和B分别表示子句②的第一、二个子目标，用①、②表示子句①和②的序号)(见下页图)。

解释下页手跟踪图：

(1)　询问句在数据库里自顶向下搜索目标，首先与①匹配，如图中左分支旁注①所示，X被例示为1，用X=1表示。

(2)　按分号";"键再求下一个值，这时指针由子句①下移指向子句②，便是图中右分支旁注②的指向。而(2)中又分出左、右两分支。分别旁注A和B，表示②体中两个子目标A和B。子目标A是int(Y)，子目标B是X is Y+1，Y是未被例示的变量。但子目标A是递归定义的，它在



数据库中自顶向下搜索寻找匹配，便与① 匹配成功，Y 被例示为 1，即 Y＝1。从左至右传至第二个子目标B，这时B里的 is 的右侧为 1＋1，左侧 X 被例示为 2，用 X＝2 表示。

（3）上述② 中 A 目标在数据库里搜索时指针指向①，若再按";"键，指针由① 下移指向②，见图中下一层的右分支旁注的② ，它又把② 的体里两个子目标 A，B，分别在图中用左、右两个分支表示(分支旁边写A，B)，这时② 中的A 递归调用到自身，相当循环一次，多了一个层次，变量名自然也要改变，例如，由 Y 改写为Y1，这时子目标 A 写为 int(Y1)。它又与①匹配，Y1 被例示为 1，即图中 Y1＝1。

子目标 B：Y is Y_1+1，即 Y 为 2，再往上返回到 X is Y+1 时，X 被例示为 3（即 2+1），便是询问回答时的 X=3。

（4）重复上述(1)至(3)，可得到无限多个解，便从 1 开始的自然数列 1，2，3，4，5，…。图中 Y_{50}，Y_{51} 是用户定义的变量名，说明递归过程到达层次的深度。

（二）求N 个正整数的累加和

例4.4 求从 1 开始的自然数列的累加和。编写程序和询问求解如下：

① sum(1, 1):-! .

② sum(N, Res):-N_1 is N−1, sum(N_1, Res_1), Res is Res_1+N.

上述程序规则① 的两个参量都为1，第一个参量表示从1开始累加到1，其结果用第二个参量为1表示，这是边界条件。规则②的头有两个参量，其第一个参量N表示从1开始累加到N，其结果用第二个参量 Res 表示。② 的体有三个子目标，第二个子目标用谓询sum及其两个参量定义的，所以是递归定义，它的第一个参量是询问句中的整数减去1，每递归一次都减1，当减到只剩1时，与边界条件规则①匹配成功，①的体中"!"截断回溯而结束。

下面给出两个询问及其答案：

?-sum(10, Res). 询问从1累加到10，答案为：

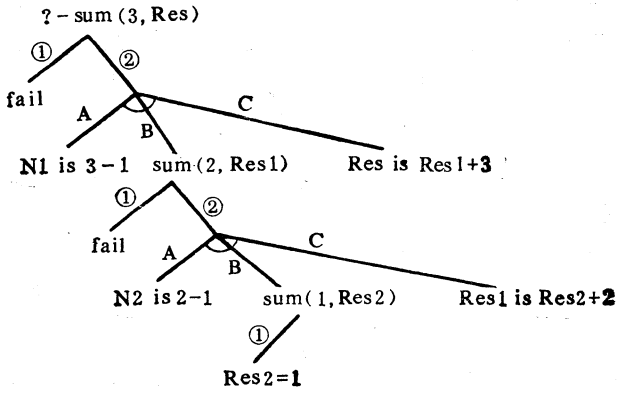
Res=55

?-sum(3, Res).

Res=6

上述第二个询问是从1累加到3的结果为变量Res，即Res=1+2+3=6。手画跟踪图来了解递归演算过程。规则②的体

有三个子目标，我们约定依次分别为A，B，C，用①和②表示规则①和②，详见如下所示：



解释跟踪图：

（1）询问句首先与①匹配而失败，即左分支旁注的①。因二者参量对应位置一个为1，另一个为3，该两个数值不能匹配而失败。

（2）询问句与①匹配失败，指针便由①下移指向②，即图中右分支旁注②。因②体里有三个目标，约定依次命名为A，B，C子目标，见图中②的下方三个分支旁注的A，B，C。这时变量名未变。

（3）上述②体里子目标B的谓词sum是递归定义的，此时它在数据库里至顶向下搜索匹配，首先与①匹配时而失败，因二者对应参量一个是1，另一个是2，1和2不一样才失败，用第二个旁注①的左分支表示。因B子目标与①匹配失败，而指针由①下移指向②，②体中三个子目标分别用A，B，C三个分支表示。注意，此时的B已递归调用到它本身，循环

是一次，加深了一个层次，原有的变量名所代表的内容已不是此层次变量名的内容，必须改变变量名，如N1改为N2，Res1改为Res2，Res改为Res1。

(4) 重复上述(1)至(3)过程，直至B子目标里第一个参量为1时，它才能与①匹配成功而结束。见图中最下面的左分支旁注①，这时Res2=1，此时往上一层返回至C分支：Res1 is Res2+2，其中Res2已被例示为1，则Res1=Res2+2=1+2=3。再把Res1的值往上一层返回至C分支：Res is Res1+3，即询问句里变量Res=Res1+3=3+3=6，这是最终答案。

假若询问 ?-sum(-3, Res)时，与①匹配而失败。指针由①指向②，在②规则体里的子目标B递归调用时，每次都使询问句里第一个参量(-3)的值减去1，由-3变为-4，-5，-6，…。永不能与边界条件①匹配成功，无休止地运行下去。可见例4.4的程序对求负数的累加和不能有效地制止。应把规则①改写为：

```
sum(N, 1):-N=<1, !.
```

例4.5 据上述分析情况把例4.4改写为：

```
① sum(N, 1):-N=<1, !.
② sum(N, Res):-N1 is N-1,sum(N1,Res1),
                Res is Res1+N.
?-sum(-3, Res).
Res=1(显然结果不对，仅制止了运行).
```

这是当询问句与①的对应参量匹配时，N被例示为-3，在①的体中检测成功，遇"!"而截断，制止运行。也可用内部谓词not取代"!"。

例4.6 用not取代例4.5中的"!"。

① sum(1, 1).

② sum(N, Res):—not(N=<1), N1 is N—1,
sum(N1,Res1),Res is N1+Res1

?-sum(—3, Res).

no

这是当询问句与① 匹配失败后，又与② 头部匹配时，N被例示为—3，进入②体第一个子目标时，由于(—3=<1)而成功，但谓词not使之不成功，所以失败，失败引起回溯，因它是最左边的子目标，无法再回溯，故最终失败，所以回答no。

用not代替"!"，使程序易读。应提倡编程者这种尝试。但有时用"!"比用not效率更高。例如：

P:-A, !, B.

P:—C.

显然比

P:—A, B.

P:—not(A), C.

效率更高，因为后者要求两次满足A，而前者只要求满足一次。

（三）求N个正整数的连乘积

例4.7 求N的阶乘积。如4! =1×2×3×4=24。

编写程序并询问如下：

① fac(1, 1):—!.

② fac(N, Res):—M1 is N—1, fac(M1, R1),
Res is N*R1.

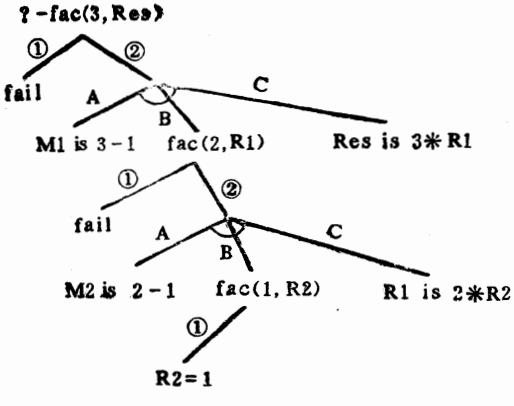
? -fac(4, Res).

Res=24

上述程序规则①的两个参量都为1，第一个参量表示1的阶乘，结果用第二个参量为1表示，是边界条件。

规则②的头部有两个参量，第一个参量N表示N的阶乘，阶乘积用第二个参量Res表示。②的体有三个子目标，第二个子目标是用谓词fac及其两个参量定义的，显然是递归定义，它的第一个参量是询问的整数减去1，每递归一次都减1，当减到只剩1时，便与边界条件规则①匹配成功，其体中"！"截断回溯而结束。

自动搜索、匹配、递归过程，用手画跟踪图来理解，图中①、②表示规则，②里三个子目标，约定分别用A，B，C表示。



解释跟踪图：

（1）询问句首先与①匹配而失败，即左分支旁注的①。因二者对应第一个位置的参量一个为1，另一个为3，该两个

数值不同而匹配失败。

(2) 询问句与①匹配失败，指计便由①下移指向②，即图中右分支旁注②。②体里有三个子目标，已约定依次命名为A，B，C子目标，见图中②分支下方三个旁注A，B，C的三个分支，这时变量名未变。

(3) 上述②体里子目标B的谓词fac是递归定义的，子目标B在数据库里至顶向下搜索匹配，首先与①匹配时而失败，因二者对应参量一个是1，而另一个是2，1和2不一致而失败，用第二个左分支旁注①表示。因子目标B与①匹配失败，指针便由①指向②，②体中三个子目标分别用A，B，C三个分支表示。请注意，此时的B已递归到它本身，循环了一次，增加了一个层次，原有的变量名不能代表此层次的内容，把变量名M1改为M2，R1改为R2等。

(4) 重复上述(1)至(3)过程，直至B子目标第一个参量为1，这时才能与①匹配成功而结束。

(5) 在上述(4)结束时，图中最底层的左分支旁注①所示：R2=1。此时往上一层返回至C分支目标：R is 2 * R2，这时R1被例示为2，因为R1=2×1=2。再往上返回一层至目标C：Res is 3 * R1，即Res=3 * 2=6，便是询问 ?-fac(3, Res)的最后答案。

由上述从底层往上返回逐层求值的过程，也可看出求阶乘的程序，是自顶向下求值过程，具体安排如下(以5的阶乘为例)：

5! =5 * 4!

4! =4 * 3!

3! =3 * 2!

2！ ＝2＊1！

1！ ＝1！ 1

这个依次排列的每层中的最右一项是阶乘，它是待求的未知数，如4！，3！，2！，直到最底层的1，便是子目标B里的第一个参量，这时B才能与①匹配成功，获得1！＝1。然后返到上一层的最右一项(如1！)，则最右一项得到解答，这便使该层的最左一项也获得答案(即2！的答案)。然后把最左项的答案返回到上一层(倒数第三层)的最右一项(如2！)，右项也为已知，因而该层的最左项(即3！)也获得答案，该答案又返回到上一层的最右项…，如此方法逐层往上返回，最后使询问的总目标5！获得答案。

请你利用此分析方法，把例4.4求累加和的程序逐层次写出求累加和的过程。

当用例4.7的程序提出询问 ?-fac(－5,Res)时，也将无休止地运行下去。请用改写例4.4的方法来改写例4.7的程序。

需再次强调，曾介绍内部谓词is这个特殊运算符时，其一般格式:变量 is 表达式，是把表达式的值例示给变量。如Res is 2＊3,则Res被例示为6,它不是赋值概念,不能写成赋值形式的Res＝2＊3＝6。但为了讲解形象，书写方便，在诸例题中，我们把它写成Res＝2＊3＝6的形式。在本书中还有些其他术语、概念，我们也采用了类式的方式来处理。

在上述一些例题中，规则②有几个子目标,因此,有人把该规则的头称为父目标,子目标都得到满足,父目标才成功。手画跟踪图时，为了简化，只画出子目标及其内容，父目标及标注约定的序号如①或②，可不写出其具体内容。手跟踪

图不仅适于递归性程序，也可用于非递归性的程序，用来加深理解程序的运行过程，对缺乏上机条件的初学者尤为重要。

习　题

1. 试编写程序，设X和N都为正整数，求X的N次方。

2. 下面是示意性程序。用一个字母表示规则的各目标，并用内部谓词write，fail，！作为目标。已给出了询问及其相应的答案。请模拟程序运行，分析这些答案是如何得出的。这是综合性的练习PROLOG自动搜索、匹配和回溯的运行过程。

```
a:-write('a').
a:-write('1').
b:-write('b'),fail.
b:-write('2').
c:-write('c'),write('3'),!.
c:-write('4').
m:-a,b,c.
n:-a,b,c,fail,a.
p:-a,!,b,c,fail.
?-m.    答案是a b 2 c 3
?-n.    答案是a b 2 c 3 1 b 2 c 3
?-p.    答案是a b 2 c 3
```

第五章　内部谓词

谓词可由用户定义，也可由PROLOG系统定义，由系统定义的谓词叫内部谓词(或称固有谓词)。一般来说内部谓词功能强。内部谓词的数量是衡量一个系统的优劣的重要指标之一。

本书的PROLOG系统有五十多个内部谓词，我们不一一列出，仅写出与本书程序中有关的或常用的内部谓词。有关上机操作的内部谓词也未列出，需请参看有关资料。

1. arg参量匹配

格式：arg(N,T,A)

参量：N——指明参量序号；T——参量所在的结构，A——被匹配参量。

功能：当T的第N个参量与A匹配时成功，否则失败。

附注：N和T必须已例示。

```
询问  ?-arg(2,related(jone,sister(jane)),X).
      X=sister(jane)
      ?-arg(1,a+(b-c),X).
      X=a
      ?-arg(2,[x,y,z],X).
      X=[y,z]
      ?-arg(1,a+(b+c),b).
      no
```

2. asserta,assertz添加子句

格式:asserta(X)

assertz(X)

参量:X——被添加的子句。

功能:将X所表示的Horn子句添加到数据库中，asserta把子句放到数据库的前面，而assertz则把子句放到数据库的后面。

附注:X必须例示为一个Horn子句，在回溯时，asserta和assertz的效果始终保持，仅当使用内部谓词retract时，才能从数据库中删去子句(Horn子句解释见其他书籍)。

3. atom原子判别

格式:atom(X)

参量:X—— 被判别的对象。

功能:如X是PROLOG的一个原子,则成功,否则失败。

询问
```
?-atom(38).
no
?-atom(pears).
yes
?-atom('I am happy').
no
```

4. atomic原子或整数判别

格式:atomoic(X)

参量:X——被识别对象。

功能:如X代表一整数或原子，则成功，否则失败。

5. call目标调用

格式:call(X)

参量：X——被调用的目标。

功能：设X被例示为一项，该项被解释为一目标。如果目标X满足，则成功，否则失败。

询问　Z=…〔p,X,Y〕,call(Z)

相当于p(X,Y)

6. clause头尾部分别匹配

格式：clause(X,Y)

参量：X——一个子句的头部；Y——一个子句的体。

功能：当X和Y分别与数据库中的某子句的头和体匹配则成功，否则失败。

附注：如库中有多个与X,Y匹配的子句，则取最前面的一个，如以后希望重新满足此内部谓词，则取第二个等等。此外当子句为事实时，则设体为true(真)。

7. cosult从文件中读出数据库

格式：consult(X)

参量：X——存放数据库的文件名。

功能：从文件X中读出数据库，如原数据库中已有Horn子句则文件中读出的Horn子句放在原有数据库所具有同一顶部谓词的子句之后。

8. fail恒失败

格式：fail

功能：此内部谓词恒不满足。

附注：常用于截断和失败来组合使用(如果执行达到此处，则应放弃满足本目标的企图)。

9. functor函数判别

格式：functor(T,F,N)

参量：T——结构；

F——成功时应为T的函数名；

N——成功时应为T的参量个数。

功能：如果T是一个以F为函数名，具有N个参数的结构，本谓词成功，否则失败。

注：如T已例示，则T非原子或结构时失败。否则，才把T函数名和参量个数与F和N相比较(原子可视为无参的函数)。

```
?-functor(a*b,F,N).
F= * , N=2
?-functor([a,b,c,d],F,N).
F= · , N=2
?-functor(pear,F,N).
F=pear, N=0
```

10. integer整数判别

格式：integer(X)

参量：X为判别对象。

功能：X与整数匹配时成功，否则失败。

11. is

格式：X is Y

功能：Y必须例示为一作为算术表达式的结构，此表达式求值结果为一整数，并将此值例示给变量X，如X已被例示，则比较X和求值结果，相等时成功，否则失败。

12. listing列出子句

格式：listing(X)

参量：X应例示为一个原子。

功能：列出所有以原子X为谓词的子句。

13. mod求余

格式：X mod Y

参量：X，Y为操作数，应例示为整数。

功能：求X除以Y所得余数，结果为一整数。

14. name原子和字符表

格式：name(X,L)

参量：X和L应分别与原子和表匹配。

功能：L应是由构成原子X的字符相对应的ASCII码值构成的表。

15. nl输出换行符

格式：nl

参量：无。

功能：在当前输出流上输出一换行符。

16. nonvar判别非变量

格式：nonvar(X)

参量：X为判别对象。

功能：如X不是一个未例示的变量则成功，否则失败。

17. not非

格式：not(X)

参量：X可解释为一目标的项。

功能：如X成功则not(X)失败，X失败not(X)成功。

注：有的PROLOG解释系统并不把not作为内部谓词，而在子句库中定义它：

```
not(X):-Call(X),!,fail.
not(X).
```

18. notrace退出跟踪

格式：notrace

功能：退出跟踪状态。

参见trace

19. repeat重复动作

格式：repeat

参量：无。

功能：通过回溯生成多重解，有的PROLOG系统中不作为内部谓词而应如下定义它：

```
repeat.
rapeat:-repeat.
```

20. read读一项

格式：read(X)

功能：从当前输入流读出一项，并将它与X匹配。它仅成功一次，输入项以句点结束。

21. retract删除子句

格式：retract(X)

参量：X为被删除的子句模式。

功能：X应例示为一子句。retract(X)将删除与X匹配的第一个子句。在回溯时，retract可以反复成功。

22. 进入跟踪状态

格式：trace

参量：无。

功能：进入跟踪状态，在此后的运行过程中将显示所有关键位置的目标。

23. var判别变量

格式：var(X)

参量：X为被判对象。

功能：如X是一个未例示的变量则成功，否则失败。

24.write写当前输出流

格式：write(X)

参量：X为被写的一项。

功能：把项X写到当前输出流中。如X未例示，则写一代下线的数字，例如-30。

25. =..函数和表的转换

格式：X=..L

参量：X代表一个有参量或无参的函数，L为与X相应的表。

功能：表L由函数X的名字后随其参量构成，如X或L被例示，则如二者匹配(在转换后的意义下)，本谓词成功，否则失败。例如：

```
append(x,y,z)=..X
X=[append,x,y,z]
```

等等

26. ,和；目标或子句右部谓词的连接

格式：,和；

功能：目标或子句右部的谓词如以逗号分开，则表示被分开的谓词之间的关系是“与”；如以分号分开，则表示二者关系为“或”。

27. =和\=以及==和\==(或=\=)的相等关系和不等关系(有的版本取消后者，用前者代替)。

格式：X=Y，X\=Y以及X==Y，X\==Y

参量：X，Y为比较对象。

功能：X＝Y当X和Y相匹配时成功，X,Y 变量共享，如其中之一被例示为mary，则X＝Y；否则失败。执行此谓词后，两个未例示的变量，将共享存储；只有一个未例示时，则它将按另一个进行例示。

X＼＝Y是当X＝Y成功时而失败，否则成功。X,Y的情况比较后完全不变。

＝＝和＼＝＝表示左右两个量绝对相等和绝对不等。例如4＝＝4，4＼＝＝5。

28. ＞,＜,＞＝,＝＜关系比较符

格式：X＞Y,X＜Y,X＞＝Y,X＝＜Y

参量：X,Y为比较对象。

功能：X,Y在所指定的关系成立时成功。否则失败。两个参量在事先必须已经例示。其关系意义如下：

＞ 大于　　＜ 小于

＞＝ 大于或等于

＝＜ 等于或小于

29. ＋,－,＊,/,mod算术运算

格式：X＋Y，X－Y，X＊Y，X/Y，X mod Y

参量：X,Y为操作数，必须事先例示为常数。

功能：对操作数进行指定的运算，所得结果可作为is谓词的右参量或作为比较谓词的参量等。所述运算如下：

＋ 加法

－ 减法

＊ 乘法

/ 除法

mod 求余

第六章　实例程序

在前几章里介绍了PROLOG语言的特点、基本知识，由列举的一些例题中，讲解了一些概念，术语，还论述了PROLOG系统功能很强，它能自动搜索、模式匹配、回溯。程序员学习编程技巧时，可采用cut截断回溯，从而控制运行流向达到预期的效果。结合实例讲了递归定义的程序，并画了手跟踪图来形象地理解自动搜索、匹配和递归的过程。在第五章里，介绍了有关经常用到的一些内部谓词。

在上述基础上，应进入本章实例程序。为了开阔读者的视野，列举了丰富多样的36个例题。其中有数学例题，排序，表处理，字符判断和处理，查询，集合运算，关系数据库，趣味智力题，自然语言处理，专家系统等十个方面的应用题。从而启发读者学习阅读程序和培养编写程序的能力。

第一节　数学例题

本节给出求两个数的最大公约数，求两个数的最小公倍数，找两个数中的大数，求绝对值，求指数的方法，以及求素数的两种方法。另外给出符号微分的简洁程序。

例6.1　求最大公约数。

例如求两个正整数的最大公约数，是将它们相同的公因数相乘。如210和63，它们相同的因子是3和7，所以它们最

大公约数是3×7=21。

定义谓词gcd是求两个正整数的最大公约数。如果X和Y的最大公约数为Z，则目标gcd(X,Y,Z)成功。

```
gcd(X,0,X):-! .
gcd(0,Y,Y):-! .
gcd(X,Y,Z):-X>Y,W is X-Y,gcd(W,
                    Y,Z), ! .
gcd(X,Y,Z):-gcd(Y,X,Z), ! .
```

在上述的程序中，每次从大的数减去小的数，再求小的数和差的最大公约数，如此继续下去直到小的数为0，此为数学上的描述。若询问：

```
?-gcd(210,63,N).
N=21
```

当然也可用转辗相除法来求得两个正整数的最大公约数，则程序更为简洁：

```
gcd(X,0,X):-!
gcd(X,Y,Z):-N is X mod Y, gcd(Y,N,
                    Z).
```

如果第一个参量小于第二个参量，则上面两个程序通过最后一个子句颠倒两个参量的位置，变为第一个参量大于第二个参量。例如，若有目标 gcd(63,210,N)，则立即变为gcd(210,63,N)。

例6.2 求最小公倍数。

若求两个正整数的最小公倍数，是将该两数相乘，再除以该两数的最大公约数。例如210和63的最小公倍数=210×63÷21=630。

设有两个正整数X和Y，它们的最小公倍数等于(X * Y)除以它们的最大公约数。所以最小公倍数程序如下：

```
lcm(X,Y,Z):-gcd(X,Y,N), Z is X * Y/N.
gcd(X,0,X).
gcd(X,Y,Z):-N is X mod Y, gcd (Y,N,
        Z).
?-lcm(210,63,M).
M=630
```

例6.3 求一个数的绝对值。

求一个整数的绝对值很简单，用它来练习传递信息的方法，定义谓词abs，其程序如下：

```
abs(X,X):-X>=0, ! .
abs(X,Y):-Y is -X.
```

规则一的头部的第一个参量表示询问的具体数，其绝对值传递给第二个参量。若询问数为5，第一规则的子目标成功，而后被“！”截断，回答为X=5；若询问数为－5，第一规则第一个子目标失败，则引起回溯，便进入第二规则，第二规则的子目标将X变为负数，原来是－5，再置一个负号“－”，变为正号“＋”，即绝对值X=5。例如询问：

```
?-abs(5,X).
X=5
?-abs(-5,X).
X=5
```

例6.4 找两个数中最大的一个数。

下面的程序也是练习PROLOG传递信息的方法

```
max(X,X,X).
```

```
max(X,Y,Y):-X<Y.
max(X,Y,X):-Y<X.
```

上述子句有三个参量，第一、二个参量是询问中的两个数，其中最大的数传递给第三个参量。

如果两个有序的数进行比较时，不外有三种情况：两个数相等，便在子句1里得出答案；若第一个参量位置上的数小于第二个参量位置上的数，便在子句2得出答案；否则在子句3找出答案而结束。例如询问：

```
?-max(6,8,X).
X=8
?-max(9,-31,X).
X=9
```

例6.5 求指数。

定义指数函数谓词exp，定义偶数谓词even。分下列三种情况讨论：

(1) exp(X,Y,Z)表示$X^Y=Z$，例如exp(X,0,1)表示$X^0=1$。

(2) 若Y是偶数，即even(Y)时，可写成$X^Y=(X)^{2*Y/2}$。

(3) 若Y是奇数时，写成$X^Y=X\cdot X^{Y-1}$，分解出一个X，Y−1是偶数，即X^{Y-1}是X的偶数幂，又是上述(2)的情况。

如此下去，便把求指数转化为求连乘积的形式。例如X的4次方可分解成连乘积形式：

$X^4=X\cdot X\cdot X\cdot X$

根据上述三种情况，依次写出以下三个子句再加上求偶数的子句，便是求指数的程序：

```
exp(X,0,1).
exp(X,Y,Z):-even(Y),R is Y/2, P is X*X,
            exp(P,R,Z).
exp(X,Y,Z):-T is Y-1,exp(X,T,Z1),Z is Z1
            *X.
even(Y):-R is Y mod 2, R=0.
?-exp(2,10,Z), writ(Z), nl.
1024
```

例6.6 用伊拉托森斯法求素数。

下面给出产生素数的伊拉托森斯(Eratosthenes)筛算法的程序。其实现思想是：

(1) 把从 2 到 N的所有整数放入筛中；

(2) 选择并去掉筛中最小的整数；

(3) 将该数加到素数集合中；

(4) 一步步把筛中所有是该数倍数的数去掉；

(5) 如果筛不空，便重复步骤(2)至(5)。

如果简单叙述是：先筛去2的倍数，再筛去 3 的倍数、5的倍数等等。根据上面算法，需要以下谓词：

谓词integers，它生成一个整数表，从2到N。

谓词remove，它把筛中所有是某数的倍数的数去掉构成一个新的表。

谓词sift，它通过调用remove和递归来完成上述实现思想中的(2)至(5)。

谓词primes是最高层目标。primes(N,L)把从 2 到N之间的所有素数放在表L中。

```
primes(N,L):-integers(2,N,Set), sift(Set,L).
```

```
integers(Low,High,[Low|Rest]):-Low=<
    High,!, M is Low+1, integers
  (M, High, Rest).
integers(_,_,[]).
sift([], []).
sift([I|Is], [I|Ps]):-remove(I, Is, New),
                       sift(New, Ps).
remove(P, [], []).
remove(P,[I|Is],[I|Nis]):-not(0 is I mod P),
                          !, remove(P, Is,
                          Nis).
remove(P,[I|Is],Nis):-0 is I modP,
                      !, remore(P,Is,Nis).
```

上面谓词integers, remove, 都是典型的用递归来实现通常程序设计语言中的迭代处理。谓词sift逐个把筛中当前最小数(素数)I放入素数集合中，再通过remove谓词把该最小数的倍数去掉，然后对新的筛中内容进行同样的处理。

例6.7 求素数另一种程序

谓词 prime 用来判断一给定的整数是否为素数。若N为素数，则目标prime(N)成功，否则失败。此处用判别一个整数是否为素数的方法是，逐个试验这个数N(设为奇数)能否被3，5，7，…，N−2整除，若不能被整除则为素数。因此测试谓词有如下形式：

test(N):-产生器逐个产生 3，5，…，N.测试器测试N是否能被产生的数整除。

当测试器测试出N不能被产生的数整除时，则测试器失

败而引起回溯，要求产生器产生下一个数再试；当产生器产生出数N时(表明N不能被3，5，…，N－2整除)，测试器最终因N能整除N而成功。

下面程序中的谓词is_prime,在子句2中以 1=N 用来区分：

(1) N被N整除，表示N是素数。

(2) N被某个小于N的数整除，表示 N 为非素数，此时因I=N失败，所以is_prime(N，I)失败。

产生器谓词generate可以模仿integer谓词来编写。含有prime谓词的程序是：

```
prime(2):-! .
prime(N):-N<2,! ,fail.
prime(N):-prime(N, I).
is_prime(N, I):-I is N mod 2, I=0, ! , fail.
is_prime(N, I):-test(N, I), I=N.
test(N, I):-generate(I), M is N mod I,M=0,!.
generate(3).
generate(X):-generate(Y), X is Y+2.
```

程序中用了四个 cut，主要是为了截断回溯不 去匹配下一个子句。程序中谓词test中的最后一个cut，是为了避免因is_prime 子句2中I不等于N所引起的回溯，而再去重新 满足test的不正确情况。

上面的检查条件显然不必测试到 N－2。第一 步 可改进为测试到N/2，进而改为测试到$\sqrt{N}$就够了。因为数论 中 有如下结果：设P_i为第i个素数，P_{i+1} 为第i＋1个素数，则有 $P_{i+1}<P_i^2$。当用N/2判断时，可把test 谓词中的 M=0 改为

(M=0; I>N/2)。再把prime谓词的第二个子句的I=N也应改为I>N/2。

判断素数的更简单方案是：

```
prime(N):-M is N-1, is_prime(N,M).
is_prime(X,1):-!.
is_prime(X, Y):-Y>1, Z is X mod Y,
                Z=\=0, Ny is y-1,
                is_prime(X, Ny).
```

有了谓词prime，就可以编一个程序来接收任一整数N，产生出1到N之间的所有素数。最简单的办法是，首先产生器逐个产生3，5，7，…，接着测试器对每一个产生数检查是否为素数。若是，则测试器成功，并把该数输出，然后再看该数是否已大于等于数N。若大于等于数N，则结束；否则，引起回溯，回溯至产生器，产生器另产生一个数。因此程序是：

```
all_prime(N):-产生器产生I, prime(I), write(I),
              nl, I≥N.
```

但应注意，产生器只能产生到数N。否则当prime(I)中I为N，N又不是素数时回溯到产生器，最后产生一个大于N的素数输而出，这是不符合要求的。

例6.8 符号微分。

符号微分程序是对所给出的含有某一变量的数学式求导数，因此符号微分是一种运算，它把一个算术表达式转变(变换)成另一个算术表达式(称为导数)。假设u和v代表算术表达式(例如x*y-2)，x代表变量，c代表常数或不同于x的量，u对x的导数写为du/dx，下面是10个微分公式(前两个

是边界条件)：

$dc/dx=0$

$dx/dx=1$

$d(u+v)/dx=du/dx+dv/dx$

$d(u-v)/dx=du/dx-dv/dx$

$d(c\cdot u)/dx=c\cdot du/dx$

$d(u\cdot v)/dx=u\cdot dv/dx+v\cdot du/dx$

$d(u/v)/dx=d(u\cdot v^{-1})/dx$

$d(u^c)/dx=c\cdot u^{c-1}\cdot du/dx$

$d(\ln u)/dx=u^{-1}\cdot du/dx$

上述公式反映在程序里，每一个公式都可写成一个子句。不难看出，用PROLOG实现符号微分是容易的。我们可把(被求导的)算术表达式描述为PROLOG的结构，把运算符描述为结构的函数符。在符号微分程序中，PROLOG的模式匹配特点，更能在一个目标匹配一个规则头的过程中显示出来。

对算术表达式的求导数可表示为目标d(E, X, F)，它表示算术表达式E对X的求导的结果为F。例如：

```
?-d(x+1, x, X).
   X=1+0
?-d(x*x-2, x, X).
   X=x*1+1*x-0
```

在上面公式中出现乘幂和单目减运算。而一般PROLOG系统，只对+、-、*和/运算符有内部说明。因此，这里必须对乘幂运算符"∧"如X∧Y表示X的Y次幂。并定义一个单目减运算符"～"如"～X"表示"-X"。整个程序是：

```
?-op(10,yfx,∧).(f为运算符,数字为∧运算优先级)
?-op(9, fx, ∼).
d(X, X, 1):-! .
d(C, X, 0):-atomic(C).
d(∼U, X, ∼A):-d(U, X, A).
d(U+V, X, A+B):-d(U, X, A), d (V, X,
                        B).
d(U-V, X, A-B):-d(U, X, A), d(V, X,
                        B).
d(C*U, X, C*A):- atomic(C), C\=X,d(U,
                        X, A), ! .
d(U*V, X, B*U+A*V):-d(U, X, A), d(V,
                            X, B).
d(U/V, X, A):-d(U*V∧(∼1), X, A).
d(U∧V,X,V*W*U∧(V-1) ):-atomic(V),
                            V\=X, d (U, X,
                                W).
d(ln(U), X, A*U∧(∼1)):-d(U,X, A).
```

注意，程序中有两个地方出现cut。第一子句中的cut，保证了一个变量对其本身求导数时只与第一个子句匹配，消除了第二子句匹配的可能性。其次，对乘法有两个子句，即第六和第七子句。第六个子句处理乘法的特殊情况，其中一个分量为常量，这里cut的作用类似于第一个cut。如果特殊情况成功，便必须消除与一般情况匹配的可能性。

上面程序求导的结果并没简化，可以对求出来的结果进行各种简化工作。例如，x*1应简化为x；x*1+1*x-0应简

化为2*x等等。可以写一个独立的代数化简程序来完成这项工作。

第二节 排 序

本节给出四个排序，即简单排序，插入排序，冒泡排序以及快速排序。

排序是把 n 个数由小到大排好(或由大到小)。如果要排序的元素都是原子(如姓名)，谓词wordless能对任意两个原子进行比较。例如，表〔alpha,beta,gamma〕已排好了序，因为对表中每对相邻原子，目标wordless(X, Y)都能成功。为了使下面的程序适用于一般对象排序，本节介绍的每一个排序程序都用谓词 order 来测试两元素是否已在正确的次序中。实际使用时，根据所要排序的数据对象的结构类型，用具体的谓词来代替order。例如，若用'<'代替 order，则是对常数的排序；若用wordless，则是对原子进行排序。这里假设，如果对象X和Y已排好了序，则目标order(X, Y)将成功。此外n个元素以表的形式存放。

例6.9 简单排序(naive sort)

将数字按算序排列的一种办法是：首先产生数字的某个排列，然后测试它是否为升序，若不是，则再产生另一个排列。此法称为简单排序。

```
sort(L1, L2): -permutation(L1, L2), sorted
                    (L2), !.
permutation(L, [H | T]): -append(V, [H | U]
                    , L), append(V,U,W),
                    permutation(W,T).
```

```
permutation([], []).
sorted(L):-sorted(0, L).
sorted(_, []).
sorted(N, [H | T]):-order(N, H), sorted(H,
                                    T).
```

谓词 append 与第三章的定义相同。程序中各谓词的意义如下：

sort(L_1, L_2)的意义是：L_2是一个表，它是 L_1 排序后的版本。

permutation(L_1, L_2)的意义是：L_2的所有元素构成的表，它以 L_1 的许多可能排列中的一种形式出现——这是一个“产生器”。

谓词 sorted (L) 的意义是：该表中的数字已按上升序排列，这是一个“测试器”。

寻找一个表排序后的版本的目标包括：产生元素的一个排列；测试此排列是否已排好序。如果是，则找到了唯一解答，否则，必须继续生成新的排列。显然这不是一个表排序的有效办法。

例6.10 插入排序(insertion sort)。

对表中所有项每次只考虑一个，而且每项都要放到一个新的表的合适的位置上。在玩扑克牌的游戏中，往往是采用这种办法：每次要摸一张纸牌，然后放在手里已有牌的一个合适位置上。如果Y是表X排好序的表，则目标insort(X, Y)成功。每个元素都是从表的头移出送到 insortx，这个谓词将会把插至表中并送回修改后的表。

```
insort([], []).
```

```
insort([X | L], M):-insort(L, N), insortx(X,
                    N, M).
insortx(X, [A | L], [A | M]):-order(A, X),
                                !, insortx(X,
                               L, M).
insortx(X, L, [X | L]).
```

为得到一般性的插入排序谓词，一个简便办法是用次序谓词作为insort的参量，这里使用了谓词"=··"

```
insort([], [], -).
insort([X | L], M, O):-insort(L, N, O),
                        insrotx(X, N, M, O).
insortx(X, [A | L], [A | M],O):- p=..[O,A,
      X], call(p), !, insortx(X, L, M,O).
insortx(X, L, [X | L], O).
? -insort([5,9,7,2],K).
K=[2,5,7,9]
```

可以使用象insort(A,B,"<")以及insort(A,B,aless)。这样的目标，而且无需谓词 order。此法也可用于本节其余的排序算法。

插入排序的另一种写法为

```
cs([X, Y], [X, Y]):-X=<Y.
cs([X, Y], [Y, X]):-X>Y.
sort([X], [X]).
sort([X, Y | Z], [U | V]): -cs([X, Y], [U,
                        W]), sort([W | Z], V).
sortn([], []).
```

```
sortn([X | Z], V):-sortn(Z, U), sort([X|U],
                                    V).
```

例6.11 冒泡排序(bubble sort)。

从头到尾测试表中两个相邻的元素是否满足次序关系，若不满足，则交换这两个元素的位置。重复此过程直至没有元素再需交换。鉴于此排序使元素"上浮"到合适的位置来排次序，类似大气泡上升，故称为冒泡法。

```
busort(L, S) :-append(X, [A, B|Y], L),or-
                    der(B,A),append(X,[B, A | Y],
                    M), busort(M, S).

busort(L, L)

append([], L, L).
append([H | T], L,[H | V]):-append(T,L,
                                    V).
```

注意谓词 append 与前边见过的完全相同，在此例中，它必须保证在每个找到的结果处能够回溯。因此，第一个子句中不会出现 cut 操作。这也是所谓"不确定"程序设计的另一个例子，因为我们以不确定的方式用append来选择表L的成员。例如：

```
?-append(X, [A, B | Y], [7, 3, 4, 2]).
X=[], [A, B | Y]=[7,3 | [4, 2]];
X=[7], [A, B | Y]=[3, 4 | [2]];
X=[7, 3], [A, B | Y]=[4, 2 | []];
⋮
```

以上写法，已对PROLOG的输出表示作了修改，以便于理解和节省篇幅。

例6.12 快速排序(Quick sort)。

快速排序适合于排序大的表。其基本思想是通过分部排序来完成整个表的排表。首先取出第一个元素，把表中比它小的元素放在它左边一个子表中，而把比它大的元素放在它右边一个子表中。例如：

```
46  55  13  42  94  05  17  70
〔13  42  05  17〕  46  〔55  94  70〕
```

然后，对左、右两个子表再分别作同样的处理：

```
〔05〕  13  〔42  17〕  46  55  〔94  70〕
  ⋮
```

一直到每一部分只剩下一个元素为止。由此可见，为了用PROLOG实现快速排序，首先需要把一个由头H和尾T组成的表分成满足下面条件的两个表L和M：

L的所有元素小于等于H；

M的所有元素大于H；

L和M中元素的次序是与在表〔H | T〕中的次序相同。

这是由谓词split完成的。即目标split(H,T,L,M)把表〔H | T〕分成表L和M。

```
split(H, 〔A | X〕, 〔A | Y〕, Z):-order(A, H),
                                  split(H, X, Y,
                                  Z).
split(H, 〔A | X〕, Y, 〔A | Z〕):-order(H, A),
                                  split(H,X, Y,
                                  Z).
```

```
split(_, [], [], []).
```

一旦把表分为L，M，下面只需要对每一个表递归地进行快速排序，并且把M快速排序后的结果附加到L快速排序后的结果和H后面。快速排序的程序有：

```
quisort([], []).
quisort([H | T], S):-split(H, T, A, B), quisort(A, A1), quisort(B, B1),
                     append(A1, [H | B1], S).
```

第三节 表 处 理

在第二章中，已经介绍过几个表处理谓词，如member，append。它们分别定义为

```
member(X, [X | _]).
member(X, [_ | Y]):-member(X, Y).

append([], L, L).
append([X | L1], L2, [X | L3]):-append(L1, L2, L3).
```

本节再介绍几个有用的表处理谓词。了解这些定义谓词如何进行工作，是十分有意义的。对这些谓词所应用的原则，同样适合于任何一类数据结构的操作。

例6.13 谓词next_to可寻找两个元素是否为表中元素的顺序。

它用来确定两个给定的元素是否是一个给定表中的顺序相邻元素，它有三个参量。如果元素X和Y是表L的两个相邻元素，则目标next_to(X, Y, L)成功。例如，判别m，

n两个元素是否是表[f, g, a, c, m, n]中两顺序相邻的两个元素，写成next_to(a, c[f, g, a, c, m, n])。

与member等谓词处理类似，首先判别这两个元素是否为该表的头两个元素，这也作为一个边界条件。若是，则这两个元素是表中的相邻元素，否则检查这两个元素是否为这表除第一个元素外、尾部中的相邻元素。这是用谓词next_to本身来完成。因而是递归定义。

谓词next_to的定义为

```
next_to(X, Y, [X, Y | _]).
next_to(X, Y, [_ | Z]):-next_to(X, Y, Z).
```

第一个子句必须假设在X和Y元素后面表中还有元素，否则象next_to(a, b, [a, b, c])这样的目标匹配不了子句next_to(X, Y, [X, Y])。

```
?-next_to(m, n, [a, b, c, d, m, n]).
yes

?-next_to(f, c, [f, g, a, c, m, n]).
no

?-next_to(b, a, [a, b, c]).
no
```

由于PROLOG的变量工作方式，因此当企图满足目标next_to(X,Y,L)时，X或Y以及它们两个都可以是未例示的。如：

```
?-next_to(X,c[a,b,c]).
X=b;
no
```

例6.14 谓词rev把元素颠倒次序。

rev把给定表的元素颠倒次序，构成一个新表，它有两

个参量。如果表L的元素颠倒后是表M，则目标rev(L，M)成功。程序采用惯用方法，把一个表的头附加到表尾的颠倒结果后面。终结条件是当第一个参量减少成空表，在这种情况下结果也是空表。例如L=〔a,b,c,d〕，通过谓词rev将得到M=〔d,c,b,a〕谓词rev的定义为

```
rev([], []).
rev([H | T],L):-rev(T,Z),append(Z, [H], L).
```

为了颠倒一个表，还可以采用一个更有效的实现方法。即假若开始有一个空表，以后依次从右向左放入原来表的第一个，第二个，…，第n个元素。而取原来表的第一个，第二个，…，第n个元素只需递归地取一个表(即每次取出后的剩余的表)的头。因此改进的rev谓词，若称为rev2，则有：

```
rev2(L1,L2):-revzap(L1, [], L2).
revzap([X|L],L2, L3): -revzap(L, [X | L2],
                              L3).
revzap([], L, L).
```

若有目标rev2(〔a,b,c〕,X)，则revzap的第二个参量一开始为空表，以后通过revzap的第一个子句不断地在里面从右向左加入元素。

例6.15 定义一个表能选出该表中前n个元素。

定义谓词till_n，它拥有四个参量，第一个参量为原表，第二个参量是把原表中取出前n个元素的终表，第三个参量是从表中取出元素个数的记数变量C，第四个参量是从原表头取出元素的个数N。其程序如下：

```
till_n([], [], _, _).
till_n([X|L],[X|P],C,N):-(var(C),C=1;true),
```

```
                                    c=<N,C1  is  C+1,
                                    till_n(L,P,C1,N).
    till_n(_,[],_,_).
```

子句2的体中第一个目标，开始运行至此，计数器 C赋初值为1，或者“;”右边为内部谓词true(真)。子句2的尾部是递归定义的，每次从原表头取出一个元素，计数器C加1，当N>C时，便回溯至子句3而结束。

子句3 第二个参量是[ ]，表示终表的表虽为空表时已达边界条件而始止。例如询问：

```
?-till_n([1,3,5,7,9],X,C,4).
X=[1,3,5,7]
```

例6.16 用表的形式来做倍乘积。

写一个程序以实现a和b两个表里元素对应乘积关系，要实现从第一个表 a 里 取出 两个表头元素，依次为a_i和a_{i+1}。第二个表b的表头b_i是a_i的二倍，即$b_i=2a_i$。而a_{i+1}是三 倍 的b_i，即$a_{i+1}=3b_i$

定义谓词admis，它有两个参量，第一个参量为a 表，第二个参量为b表，程序如下：子句1是边界条件，子句 2的体是递归定义的。

```
admis([],[]).
admis([X,Y|Z],[U|V]):-U is 2*X,
                       Y is 3*U,
                       admis([Y|Z],V).
admis([X],[U]):-U is 2*X.
```

例如询问及答案如下：

```
?-admis([1|U],V),write([1,U]),
```

```
        write(',',write(V),fail.
[1],[2]
[1,6],[2,12]
[1,6,36],[2,12,72]
    ⋮
```

第四节　字符判断和处理

定义谓词，编写小程序，很容易判断一个或数个单词的性质，以及它们之间的相互关系，并加以处理达到预期的目的。本节举两个例子。

例6.17　判断单词的字母是否左右对称。

判断组成一个英文单词的字母是否是左右对称，如单词madam组成它的字母是左右对称的，而jhom则不是左右对称的字母。

程序中子句1里用read(X)读入一个个单词，每读入一个单词便测试它是否由左右相同的字母组成，若是则回答是(is)…，否则回答(not…)。

```
begin(X):-read(X),(X=stop;test_palindrome
        (X),begin(Y)).

test_palindrome(X):-name(X,Nx),palindrome
                    (Nx),write(X),
                    write('is a palindrome'),nl,!.
test_palindrome(X):-write(X),
                    write('is not a palindrome'),
                    nl.
```

```
palindrome(X):-reverse2(X,X).
reverse2(L1,L):-reverse_append(L1,[],L).
reverse_append([H|T],L,M):-reverse_append(T,
                              [H|L],M).
reverse_append([],L,L).
    EXECUTION:
?-begin(X).
 madam. john. astyuytsa. horse. bull. stop.
 madam is a palindrome
 jone is not a palindrome
 astyuytsa is a palindrome
 horse is not a palindrome
 bull is not a palindrome
```

例6.18 两个单词首尾相连接处不许字母相同。

例如把VACHE和CHEVAL两个法语单词连接在一起，在首尾连接处的共同字母CHE是重复字母，把这三个字母只保留一个单词的CHE，便成为VACHEVAL。

```
begin:-mutation(X),name(Nn,x),write(Nn), nl,
      fail.
begin:-nl,write('Done'),nl.

mutation(X):-animal(Y),name(Y,Ny),
             animal(Z),name(Z,Nz),append(Y1,
             Y2,Ny),Y1\=[],append(Y2,Z2,
             Nz),Y2\=[],append(Y1,Nz,X).

append([],X,X).
```

```
append([A|X],Y,[A|Z]):-append(X,Y,Z).

animal(alligator).
animal(tortue).
animal(caribou).
animal(ours).
animal(cheval).
animal(vache).
animal(lapin).
Execution:
?-begin.
alligatortue
caribours
chevalligator
chevalapin
vacheval
Done.
```

第五节 查 询

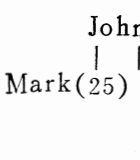
PROLOG在数据库中至顶向下自动搜索，寻找匹配的目标，这搜索就是检索过程，也就是查询过程。因此说PROLOG的查询功能，是它的先天性。它不仅仅是在数据库中简单的查询，而且还能增删库中的信息数据，成为知识库管理系统。

例6.19 查询家谱

给出家庭成员关系数据库，寻找出john有否相差10岁的

两个后代。定义谓词beget(产生)，aged(年龄)，has_descendant(子孙)。事实beget (john,mark)表示john产生makr。aged(john,57)表示john是57岁。

可用树表示这家谱：



家庭关系数据库和询问如下：

```
has_descendant(X,Z):-beget(X,Z)
has_descendant(X,Z):-has_descendant(X,Y),
                     beget(Y,Z).

beget(john,mark).
beget(john,paul).
beget(paul,tom).

is_aged(mark,25).
is_aged(john,57).
is_aged(paul,35).
is_aged(tom,15).
  ?-has_descendant(john.X) ,is-aged(X,N),
       has_descendant(john,y),
       is_aged(Y,M),M is N+10.
```

例6.20 二叉树的搜索与插入

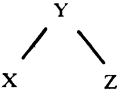
查询数据库时，先给出一个指定的关键字，然后在数据

库中查找时用二叉树的根结点与该关键字进行比较，若大于关键字，则沿该结点的左子树继续搜索查询；否则沿该结点的右子树搜索查询。如此下去，直到获得答案为止。若给出的关键字，二叉树中无此结点，便把该关键字插入作为二叉树的根结点。上述问题的算法用下列程序描述：

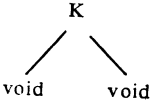
```
binary(t(X,K,Z),K,t(X,K,Z)).
binary(t(X,Y,Z),K,t(X1,Y,Z)):-name(K,
        Nk),name(Y,Ny),N1<Ny,binary(X,K,
        X1).
binary(t(X,Y,Z,),K,t(X,Y,Z1)):-name(K,
        Nk),name(Y,Ny),Nk>Ny,binary(Z,K,
        Z1).
binary(void,K,t(void,K,void)).
```

解释程序如下：

子句2头部相当三个参量，第二个参量是一个复合项(即结构)，t是函数名，复合项t(X,Y,Z)可用二叉树表示：Y是二叉树的根结点，X是左子树，Z是右子树。以t为函数名的复合项也都如此定义。子句2头部第二个参量是给定的关键字K。第三个参量位置上的复合项t(X1,Y,Z)表示；若K＜Y，则继续沿左子树搜索查询，结果以Y为结点的左子树用X1代替X；若K＞Y，则右子树用Z1代替Z，这便是子句3；若K＝Y便查询找到了所需的结果。



当给出的关键字K，左二叉树中查找时无此结点，便用K插入作为结点，这便是子句1。子句3的参量void表示空结点，把K插入作为空结点的根。



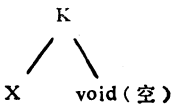
例62.1 删除二叉树的结点。

在数据库中经常用到删除一个二叉树的结点，其程序如下：

```
delete(t(X,K,void),K,X).
delete(t(X,K,t(void,Y1,Z1)),K,t(Y,Y1,t(X,
      Y1,Z1))).
delete(t(X,K,t(X1,Y1,Z1)),K,t(X,Y,t(X2,Y1,
      Z1))):-seccessor(X1,Y,X2).
delete(t(X,Y,Z),K,t(X1,Y,Z)):-K<Y,,delete
      (X,K,X1).
delete(t(X,Y,Z),K,t(X,Y,Z1)):-K>Y,
      delete(Z,K,Z1).
delete(void,K,void).

successor(t(void,Y,Z),Y,Z).
successor(t(X,Y,Z),Y1,t(X1,Y,Z)):-successor
                                  (X,Y1,X1).
```

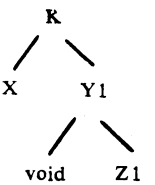
定义谓词delete拥有三个参量。其参量所在位置表示的内容参见例6.20。子句1表示删去二叉树的根结点K，只剩下左子树X，用树表示：



子句2第一个参量描述为

删去K便是第三个参量所表示的二叉树。

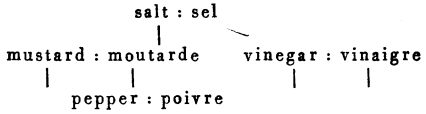
子句8的successor(X,Y1,X1)表示二叉树X中找到其关键字最小的元素Y1，去掉Y1后剩下的二叉树为X1。



子句6只有关键字K，其左右子树为空，即边界条件。

例6.22 在字典中找同义词。

写一个程序，把名字安排在字典里，按字母顺序把英语单词与同义的法语单词配对地找出来。例如：



英语salt(盐)，对应的法语是冒号":"右侧的 sel。英语mustard(芥茉)，vinegar(醋)，…。其程序和询问如下：

```
dictionary(void).
dictionary(dic(X,Y,D1,D2)):-dictionary(D1),
                            dictionary(D2).
dic(salt,sel,dic(mudcard,moutarde,void,dic
                    (pepper,poivre,void,
        void)),dic(vinegar,vinaigre,void,
        void)).

lookup(Name,dic(Name,Value,_,_),
                    Value):-!.
lookup(Name,dic(Name1,_,Before,_),
                Value):-Name<Name1,
                    lookup(Name,Before,
                    Value).
lookup (Name,dic(Name1,_,_,After) ,Value):-
                Name>Name1,lookup(Name,
```

```
                                    After,Value).
?-lookup(salt,D,X1).
D=dic(salt,X1,D1,D2).
?-lookup(mnstard,D,X2).
D=dic(salt,X1,dic(mustard,X,D,D4).D2).
?-lookup(vinegar,D,X3),lookup(pepper,D,X4),
                        lookup(salt,D,X5).
```

编写程序要求：

(1) 在字典中查找一个名字并能找出与它配对的值；

(2) 在一个确定的名字前查找一个名字；

(3) 在一个确定的名字后查找一个名字；并要考虑 如何将此思路扩展下去。

第六节 集合运算

集合是一些不同的确定对象的全体，它是重要的数据结构之一。我们通常注意的是：

(1) 不关心某一个元素在一个集合里出现的次数，因为它并没有多大意义，例如集合〔m,n,m,n〕与集合〔n,m〕是相同的；

(2) 集合中元素并没有排列顺序的关系，如上述一、二个集合是相同的；

(3) 我们关心某一元素是否属于某一集合，也就是该元素是否在该集合里，一个集合也可以作为另一集合的元素。

可用PROLOG 的表描述集合，因为一个表能包含任意元素。由于表可以嵌套，所以表还能包括其他的表。当把集合表示成表时，属于集合的每个成分在表中将安排成只有一

个元素。这样处理可简化删除元素以及其他运算。因此，对集合的运算可看作对表的处理。每次运算结束，把结果表中重复出现的元素去掉，就得到集合运算的结果。本节给出几个例子来描述集合的基本运算，如两个集合的相等，相减，交集和并集等。

我们已在第三章例3.1介绍谓词member用来确定某一元素是否为指定的表中一个成员，即判断一个元素是否属于一个集合，这是一个典型的例题。

例6.23 判两个集合相等。

```
set_equal(X,X):-!
set_epual(X,Y):-equal-lists(X,Y).

equal_lists([],[]).
equal_lists([X|L1],[Y|L2]):-delete(X,L2,L3),
                            equal_lists(L1,
                            [Y|L3]).

delete(X,[X|Y],Y).
delete(X,[Y|L1],[Y|L2]):-delete(X,L1,L2).
```

子句1两个参量都是集合X，表示两个集合相等。

子句2判X和Y集合相等否。

子句3两个参量都是空表。

子句4头部是判两个表相等否，子目标delete(X,L2, L3)是在L2中删去X，剩下为L3。

子句5是将第二个参量位置上的表删去X，剩下的为第三个参量Y(前面已叙述，可用表来表示集合，一个表去

掉表头元素，剩下的仍为一个表）。

例6.24 求两个集合的交集。

定义谓词intersection有三个参量。第一、二个是给定的两个集合，第三个参量是两个集合的交集。

如果X和Y的交集是Z，则目标 intersection (X,Y,Z) 成功。程序如下：

```
intersection([],X,[]).
intersection([X|R],Y,[X|Z]):-member (X,Y),
                              !,intersection
                              (R,Y,Z,).
intersection([X|R],Y,Z):-intersection(R,Y,Z).
```

子句1是边界条件，表示任一个集合X与空集的交集仍是一个空集。例如询问：

```
?-intersection([a,b,c,d],[a,e,d],L).
L=[a,d]
```

例6.25 求两个集合的并集。

定义谓词union 有三个参量，第一、二个参量是给定的两个集合，第三个参量是两个集合的并集。如果X和Y的并集是Z，则目标 union(X,Y,Z) 将成功。在处理中，我们必须遵循本节开始所指出的集合中的任一元素，它只能在用表来表示时，表中它只出现一次。因此，对集合运算的处理就是考虑到这个因素的表处理：

(1) 边界条件：当两个集合(表)中一个为空集，则任何集合与空集的并集仍为该集合。在具体处理中，因为是对第一个参量进行递归处理，因此边界条件是指第一个参量为空集。

(2) 当第一个表为非空表(非空集)时，则把该表分为表头和表尾。若头是第二个表的一个元素，则结果是第一个表的尾和第二个表所代表的两个集合的并集，否则结果是头元素应在并集中。

```
union([ ],X,X).
union([X|R],Y,Z):-member(X,Y),!,union
                    (R,Y,Z).
union([X|R],Y,[X|Z]):-union(R,Y,Z).
```

例6.26 两个集合相减。

两个集合相减是减去这两个集合的交集。如集合[4,5,6]减去集合[7,8,4]，所得集合为[5,6]。

```
subtract(L,[],L):-!.
subtract([H|T],L,U):-member(H,L),!,
                        subtract(T,L,U).
subtract([H|T],L,[H|U]):-!,subtract
                            (T,L,U).
subtract(_,_,[ ]).

member(H,[H|_]).
member(I,[_|T]):-member(I,T).
?-subtract([4,5,6],[7,8,4],X).
X=[5,6]
```

子句1是集合L减去空集合所得仍为L集合。

子句2、子句3的子目标subtract(T,L,U)表示集合T减去集合L得到集合U。

例6.27 去掉表中重复出现的元素。

定义谓词 trans，它有两个参量让它对表进行处理，把一个元素多次出现的去掉(只保留一个)。

例如询问：

```
?-trans([a,b,a,e,b,f],R).
R=[a,b,e,f]
```

程序如下：

```
trans([ ],[ ]).
trans([X|L],S):-member(X,L),! ,
                    trans(L,S).
trans([X|L],[X|S]):-trans(L,S).
```

第七节　关系数据库

前面已说明，PROLOG 程序本身实际上就是一种表示事实和规则间关系的关系数据库，而这种数据库又因具有一定程序的智能而可称为知识库，本节举几个启发性的例子，如排课程表，驾驶员出发，电话寻找人，晶体管咨询等关系库。

例6.28　排课程表。

编写一个程序，询问是否有这样一个学生，一位教授在同一教室里教他两种不同的课程。

有关的数据库给出了选课的学生，教授教的课程，课程的时间以及所在的教室。

定义谓词query，它有S和P两个参量，S代表学生，P表示教授。

student(robert,prolog)，表示学生robert学习prolo-

gprofessor(luis,prolog)，表示教授luis教prolog。

course(prolog,monday,room1)，表示星期一的prolog课在room1教室里进行。

编写的程序如下：

```
query(S,P):-student(S,C1),
            course(C1,D1,R),
            professor(P,C1),
            student(S,C2),
            course(C2,D2,R),
            professor(P,C2),C1\=C2.

student(robert,prolog).
student(john,music).
student(john,prolog).
student(john,surf).
student(mary,science).
student(mary,art).
student(mary,physics).

professor(luis,prolog).
professor(luis,surf).
professor(antonio,prolog).
professor(eureka,art).
professor(eureka,music).
professor(eureka,science).
professor(eureka,physics).
```

```
course(prolog,monday,room1).
course(prolog,friday,room1).
course(surf,sunday,beach).
course(maths,tuesday,room1).
course(maths,friday,room2).
course(science,thursday,room1).
course(science,friday,room2).
course(art,tuesday,room1).
course(physics,thursday,room3).
course(physics,saturday,room2).
```

例如询问?-query(john,luis),请给出答案。

例如询问?-query(mary,eureka),请做出答复。

例6.29 驾驶员出发。

有四个驾驶员，他们的名字分别叫：Tommy，Fred，Pedro，Nuno。驾驶汽车到达了大目标：Monte Carlo。有三个出发地可到达 Monte Carlo。这三个出发地是London(伦敦)，Paris(巴黎)，Lisbon(里斯本)。在每一个出发地里有若干驾驶员想要到达大目标后，再去大目标地区所属的小地区。如London:John(BMW) 表示出发地 London里的John想去大目标所属的BMW小地区。以下分别给出三个出发地，以及每个出发地里的人名和欲到的小地方(用括号括起)。

London: John (BMW), Tommy (FORD),
Fred (BMW), Anne (FORD), and Teddy (BMC)

Paris: Guy (FIAT), Caune (FORD) Jean (FIAT) and Brigitte (BMC)

Lisbon. Luis (FIAT), Carlos (BMW), Lucas (BMC), Pedro (FORD), and Nuno (FIAT).

在数据库中提出询问要回答：

(1) 到达目的地的四个人是从什么地方出发的。

(2) 这四个人到达了大目标所属的什么小地区。

```
rally :-pilots(A,london),pilots(B,Lisbon),
    pilots(C,paris),pilots(D,bmw),pilots(E,
    fiat),pilots(F,ford),pilots(G,bmc),pilots
    (H,monte_carlo),intersection(B,H,Lis),
    intersection(A,H,Lon),intersection
    (C,H,Par),intersection(D,H,Bmw),
    intersection(E,H,Fiat),intersection (G,H,
    Bmc),nl,nl,write('Arrived:'),nl,nl,write
    ('From London:'),write(Lon),nl,write
    ('From Lisbon:'),write(Lis),nl,write
    ('From Paris:'),write(Par).nl,nl,write
    ('in BMW:'),write(Bmw),nl,write
    ('in FIAT:'),write(Fiat),nl,write('in
    FORD:'),write(Ford),nl,write
    ('in BMC:'),write(Bmc),nl.

intersection([ ],X,[ ]),
intersection([X|R],Y,[X|Z]):-member (X,Y),
                              !,intersection
                              (R,Y,Z).
```

```
intersection([X|R],Y,Z):-intersection
                              (R,Y,Z).

member(X,[X|_]).
member(I,[_|T]):-member(I,T).

pilots([john,tommy,fred,anne,teddy], london).
pilots([guy,claude,jean,brigitte],paris).
pilots([luis,carlos,lucas,pedro,nuno],lisbon).
pilots([fred,carlos,john],bmw).
pilots([luis,nuno,jean,guy],fiat).
pilots([anne,tommy,claude,pedro],ford).
pilots([teddy,brigitte,lucas],bmc).
pilots([tommy,fred,pedro,nuno],monte_carlo).
```

例如询问：

```
?-rally.
Arrived:
From London:[tommy,fred]
From Lisbon:[nuno,pedro]
From paris:[ ]
in BMW:[fred]
in FIAT:[nuno]
in FORD:[pedro,tommy]
in BMC:[ ]
```

这四个人的出发地

这三个人到达了大目标中的小地区

解释上述程序如下：

子句里有23个子目标，如第一个子目标pilots(A,lond-

on)，表示从伦敦出发的驾驶员都在集合A里，第二至第八个子目标也都如此理解。

第九个子目标intersection(A,H,Lon)，表示A、H两个集合里的人，其交集为Lon(一群人)。类似目标也如此定义，子句2表示X集合与空集合的交集仍为空集合。

子句3的头有三个参量都是表，第一、二个表的交集为第三个表，即R与Y的交集为Z。

例6.30 电话寻找人。

把以下的事实加在关系数据库里。

number(X,N)，可通过电话号码N找到X人家。N号码不是X人家的，而是其他人家里的电话号码。

visits(X,Y)，表示X人正在访问Y人。

at(X,Y)，表示X人正在Y人的住所里。

phone(X,N)，表示X人的电话号码为N。

根据以上事实可做如下推理：

```
visits(X,Y)and at (Y,Z)→at(X,Z).
at(U,V)and phone(V,N)→number(U,N).
```

根据上述推理，理解以下的关系数据库：

```
find(X):-number(X,N),write('phone:'),
        write(N),nl.
find(_):-write('Don''t Know.'),nl.

number(X,N):-at(X,Y),phone(Y,N),!.
number(X,N):-phone(X,N).

at(X,Z):-visits(X,Y),at(Y,Z).
```

```
phone(coleman,'100001').
phone(gordon,'100002').
phone(wagner,'100003').
phone(smith,'100004').
Execution:

Suppose you want to reach Coleman and
know he is visiting
with Wagner and that Wagner is at Gordon's
house.
You just have to write:
visits(coleman,wagner).
at(wagner,gordon).
```

例如询问：

```
?-find(coleman).
phone:100002
```

例6.31 晶体管应用程序管理。

编写一个能够简单回答用户咨询晶体管的程序。每个有名称的晶体管有八个参数：材料（mat），极性（pol），功能(fun)，功率(pow)，集电极和基极之间的电位(vcb)，集电极和发射之间的电压(vce)，放大倍数(hfe)和capsula类型(cap)。

以五个晶体管为例，列入下页表，表中第一列是晶体管名称，其余各列依次为上述每个晶体管的八个参数。

```
begin:-write('Characteristics(value)'),nl,
        repeat,nl,continue(_).

continue(L):-write('-'),ttyflush,read(C),
```

名称 \ 参数	mat	pol	fun	pow	vcb	vce	hfe	cap
2n2219	si	n	hsa	800	60	30	120	T05
2n2904	si	p	hss	300	60	40	120	T05
2n3055	si	n	lpai	115000	100	70	70	T03
2n3904	si	n	hss	310	60	40	300	T092
2n3906	si	p	hss	310	40	40	300	T092

```
process(C,L).

process('Goodbye',_).
process(stop,L):-(trans(X,L),nl,write
        ('trans(X,L)'),nl,write('Do you want
        it (yes or no)?'),nl,read(S),S=yes,!,
        write(New characteristics'),nl,fail;
        write('No transistor available'),nl,
        continue(L)).
process(C,L):-(subst(C,L),L=L1;replace
              (C,L,L1)),continue(L1).

subst(C,L):-reduct(C,C1),match(C1,L).

reduct(C,C1):-C=..[X,N,mili],C1=..[X,N].
```

```
reduct(C,C1):-C=..[X,N,w],C1=..[X,M],
              M is N*1000.
reduct(C,C1):-C=..[X,N,_],C1=..[X,N].
reduct(C,C).

match(mat(A),[A|_]).
match(pol(B),[_,B|_]).
match(fun(C),[_,_,C|_]).
match(pow(D),[_,_,_,D|_]).
match(vcb(E),[_,_,_,_,E|_]).
match(vce(F),[_,_,_,_,_,F|_]).
match(hfe(G),[_,_,_,_,_,_,G|_]).
match(cap(H),[_,_,_,_,_,_,_,H]).

replace(C,L,L1):-subst(C,L1),replace1(L,L1).
replace1(X,X):-var(X),!.
replace1([],[]):-!.
replace1([H|T],[H1|T1]):-!,replace1 (H,H1),
                         replace1(T,T1).
replace1(_,_).

trans(t2n2219,[si,n,hsa,800,60,30,120,
     'T05']).
trans(t2n2904,[si,p,hss,300,60, 40,120,'T05']).
trans(t2n3055,[si,n,lpa,'115000',100,70,70,
     'T03']).
```

```
trans(t2n3904,[si,n,hss,310,60,40,300,
      'T092']).
trans(t2n3906,[si,p,hss,310,40,40,300,
      'T092']).
```

当询问：

```
?-begin.
Characteristics(value)(用户键入以下信息)
-mat(si).
-pol(p).
-fun(hss).
-pow(300,mili).
-vcb(60).
-stop.
```

给出这个晶体管：

```
trans(t2n2904,[si,p,hss,300,60,40,120,T05]).
Do you want it (yes or no)?(你满意吗?)
   no.                              (不满意)
No transistors available(用户不满意上述给出的
                              晶体管,又输入以下信息)
-fun(hsa).
-pol(n).
-pow(800).
-stop.
```

又给出这个晶体管：

```
trans(t2n2219,[si,n,has,800,60,30,120,T05])
Do you want it (yes or no)?(你满意吗?)
```

```
yes.                                    (是)
New characteristics
-Goodbye.
```

子句3的参量stop表示用户输入信息结束，其子目标trans(X,L)表示根据参数L找出晶体管X。

子句6子目标mach(C_1,L)表示不成功时要改变原来的参数。

子句7和子句8，是把功率单位换算成毫瓦为单位(所以乘1000)。

子句11至18定义谓词match用来与晶体管表中各参数匹配。match有两个参量。第一个参量是变量，第二个参量是要与参数变量匹配后例示出变量的具体参数。注意第二个参量是一个表，表中最多有八个元素位置，与已给出参数中八个参数所在的位置一一对应匹配。如子句1只要求与mat匹配，它在参数表中第一个位置。所以子句1中变量A在表头位置。如子句18要找出参数cap，所以它在子句18第二个参量位置上的表中第八个位置，该位置上变量H与给定的参数表里第八个参数匹配，前七个参量不关心，所以在子句18作为子目标的表里，用七个无名变量"_"表示。按此法理解同类子句的解释。

子句20说明该项参数未填，则保持原样。

子句21的两个参量为空表"[ ]"，表明最后一项也考查完了。

子句22的子目标，表示一一对应比较所填写的参数。

该晶体管关系数据库的用途：

(1) 建立晶体管参数表，用户提出某项参数C，可咨询出该C可选用某型号晶体管。

(2) 若用户准备使用x型号晶体管，可咨询出该晶体管各项参数。

(3) 在上述(1)中，可不断改变所需参数，用read(c)输入，变量C在变，获得L1，这是另一个晶体管的。

(4) 该系统可扩展成把各项参数给出误差范围，若用户提出某项参数不在指定的参数表中，可以咨询选择何种型号类似的晶体管作为参考。

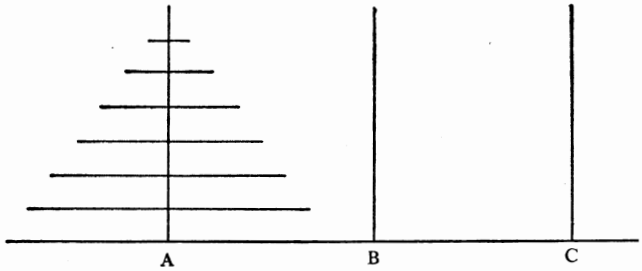
第八节 趣味智力题

无论用哪一种程序设计语言编写趣味智力题的程序都需要编程者对所使用的语言以及其他知识具有相当的基础。PROLOG语言擅长于逻辑推理，尤其适宜用它编写人工智能性的趣味题的程序，如探索迷宫，过河策略等题。也适于编写非人工智能性的游戏题，如梵塔等题。用PROLOG来编写趣味智力题时，其程序非常简洁。本节以梵塔、探索迷宫以及强盗和商人过河为例题，来加以论述。

例6.32 梵塔

梵塔又叫hanoi塔(汉诺塔)，是利用三个柱子A，B，C(如下图所示的左、中、右柱)。

n个中央有孔的圆盘从大到小依次穿在A柱上，即A柱上的圆盘越往顶层其直径越小。要把A柱上的圆盘全摆到B柱上，在摆动过程中，每次只许搬动一个圆盘，而且要遵守只能从顶层往下搬，放在柱子上时必须是大盘在底下，小盘在上层。C柱作为临时暂存柱，利用这三个柱子互



相倒换圆盘，来达到目的。这个问题可分为下述三步来解决：

(1) 把 A 柱顶层n－1 个圆盘搬到空柱C上；

(2) 把 A 柱底层的一个大圆盘搬到 B 柱上；

(3) 把C柱上的 n－1 个圆盘搬到B柱上。

要把 n－1 个圆盘从某一个柱上搬到另一个柱上，又可分为上述三步来解决。只不过比前一次的盘子数减少一个(因为首先搬动的 n－1 个中的 n－1 个圆盘)。如此 n－1 个处理下去，利用PROLOG 的递归定义，直到圆盘数为0时达到边界条件而结束。

根据上述算法思路，编写递归定义的程序如下：

```
hanoi(N):-move(N,left,centre,right).
move(0,_,_,_):-! .
move(N,A,B,C):-M is N-1,move (M,A,C,B).
        inform(A,B),move(M,C,B,A).
inform(X,Y):-write(move,a,disc,from,the,
'X',pole,to,the,'Y' pole),nl.
```

程序中hanoi有一个参量。在A柱上若有n个圆盘，则ha-

noi(N)将输出n个圆盘由A柱搬到B柱的移动轨迹。

谓词move有四个参量，第一个参量为将要移动的盘的个数，第二、三、四参量分别为源柱、目标柱、临时暂存柱，即A，B，C(左、中、右)柱。可见在程序中这三个柱子的作用在相互变动。

谓词inform利用内部谓词write来输出移动的轨迹，它打印出移动圆盘号及其用到的柱子的名字。

当提出询问？-hanoi(3).便有下列结果输出：

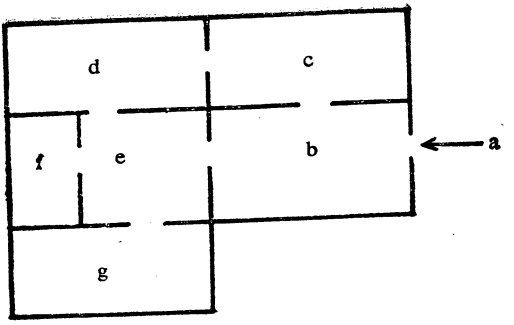
```
move a disc from the A pole to the B pole
move a disc from the A pole to the C pole
move a cisc from the B pole to the C pole
move a disc from the A pole to the B pole
move a disc from the C pole to the A pole
move a disc from the C pole to the B pole
move a disc from rhe A pole to the B pole
```

可见3个圆盘要移动7次，即$2^3-1=7$，若n个圆盘要移动2^n-1次。但上面的输出形式不太令人满意，请对上述程序进行修改，使输出信息中的"a disc"改为第几号盘。

例6.33 探索迷宫

有一座套间互相连通的平房，相邻房间有敞开的门可供出入，如下图所示。每个房间用一个英文字母表示，如b，c…g。如果由入口a进入b房间，由b房间可进入c或e房间，如此走法，可探索到预期要达到的房间。

定义谓词d有两个参量，如d(b,c)表示从b房间可到c房



间，若d(c,b)表示从c房间可到b房间。因此首先要定义这些事实，置入数据库，还要在库里给出最终要达到的房间，假如是g房间，便写成goal(g)。

```
d(a,b).
d(b,c).
d(b,e).
d(d,e).
d(e,f).
d(e,g).
goal(g).
```

上面的事实是单方向的、如只给出b到c、还应加上c到b。逆向省略未写。

定义谓词go有三个参量，go(X,Y,T)表示从X房间进入Y房间，进入Y房间时要记录在T上，以做标记，避免重复地往返走来走去，陷入死循环。除上述事实外，还定义如下规则，便是完整的程序。

```
go(X,X,T).
```

```
go(X,Y,T):-(d(X,Z); d(Z,X)), not(member
(Z,T)),go(Z,Y,[Z | T]).
```

例如提出询问由a进入X房间：

```
? -go(a,X,[ ]),goal(X).
  X=g
```

询问句谓词go的第三个参量为空表"[ ]"，表示开始时还未记录到达的房间，遍历房间时，在表里加入到达的房间。

该数据库和询问可以表达更多的信息。例如欲达到g房间，而限止不经过d和f房间，可提出询问：

```
? -go(a,X,[d,f]), goal(X).
  X=g
```

程序中，子句体里的第一个目标用";"表示或的关系，因为相邻两个房间是双向通行，二者之一只要有一个方向能通行，则成立。第二个子目标外层括号里表示Z与T是否为成员关系，若不是成员关系，即T记录里没有Z，也就是没到过Z房间，Z与T非成员关系，而外层谓词not为非，而"非的非"为真，则该子目标成功，便到递归定义的第三个子目标。这时到达Z房间，便记录在表头，如[Z|T]。假如第二个子目标失败，便回溯到第一个子目标，再探索另外一个房间。

例6.34 强盗和商人过河

有三个化装成为仆人的强盗和三个商人，他们都来到河边，准备渡河，但岸上仅有一条只能坐两个人的小船，船到对岸后，又需有人将船划回。见此情况，三个强盗暗地商定，不论在河的哪岸，只要他们的人(指强盗)比商人多，就把商人弄死，抢劫珠宝逃走。这三个商人有所觉察，但一时又想不出一个好办法，十分为难。请为商人找出一条安全过

河的好方法。

定义谓词 boat表示船的状态，(boat=1 表示船往 源 岸行。boat=0 表示船往对岸行)。boat 有两个参 量，如 boat (M,C)表示船上有M个商人,C个强盗。根据题意有 五 种 过河状态：

boat(0,1)，boat(1，0)，boat(1，1)，boat (2,0)，boat (0,2)。

定义谓词state有三个参量，如state(X,Y,Z)表示商 人数量X，强盗数Y，第三个参量Z表示船的两种状态(1或0)。

源岸的初始状态，其初值为state(3,3,1)。从初始状 态的角度看问题的最终状态，其终值为state(0,0,0)。

定义谓词unigue用来检测是否出现过的状态，避免重复进行，如两个强盗乘船过去又过来，陷入死循环。例如目标unigue(X,Y,0) 是记录已经出现过的状态,即船在0状态时，有X个商人，Y个强盗。

船出发后的状态，加上船上乘坐的人，便是出发前的状态。若源岸有M个商个,C个强盗,那么在对岸便有(3－M)个商人和(3－C)个强盗,这便是state的规则1和规则2所描述的状态。

商人的数量和强盗人数各三人，定义谓词sofe有两个参量来检查是否安全。目标safe(M,C)表示M个商人和C 个强盗相处，按题意条件检查是否为安全状态。安全 状 态 为：safe,(0,c)，safe(3,c)，safe(M,M)。

总之，每次运行要计算两岸人数对比，检查安全状态，还要检查是否重复了前一次的状态(避免无限循环)。若经检查不符合条件，PROLOG提供自动回溯的功能，改变前 一

次的运行状态，如此下去，直至商人全部安全过河为止。其全部程序如下：

```
boat(0,1).
boat(1,0).
boat(1,1).
boat(2,0).
boat(0,2).
state(3,3,1)
state(X,Y,0):-unique(X,Y,0), boat(M,C),M1,
is M +X,M1=<3,C1 is C+Y,C1=<3,safe(M1,C1)
state(M1,C1,1),write('state(',M1,C1,'1)').
state(X,Y,1):-unique(X,Y,1),boat(M,C),M1 is
X-M,M1>=0,C1 is Y-C,C1>=0,safe(M1,C1),state.
(M1,C1,0),write('state(',M1,C1,'O)').

unique(M,C,B):-not(once_upon_a_time(M, C,B)).
        assert(once_upon_a_time(M,C,B)).
safe(0,C).
safe(3,C).
safe(M,M).
?-state(0,0,0).
```

八皇后也是趣味智力方面的名题，详见参考书第十三章的13.6节。

第九节　自然语言处理

对自然语言的理解和处理是PROLOG语言应用在人工

智能领域里一个重要的研究课题，是各种应用系统用户接口智能化方面的基础。目前已有阐述、开拓这方面的知识的专著，参考书的第六章也有论述和程序，本节仅对自然语言句子结构分析做示意性的简介。以下仅对英语简单句，按给定的文法进行语法分析，得出句子的语法树。

例6.35 对下列英语句子进行语法分析并画语法树。

The man eats the apple.

解决这类问题，一个简单的方法是直接根据文法写出相应的PROLOG 规则。例如有以下简单的上下文无关文法：

sentence→noun-phrase,verb-phrase. 句子→名词短语，动词短语

noun-phrase→determiner,noun. 名词短语→定冠词，名词

verb-phrase→verb. 动词短语→动词

verb-phrase→verb,noun-phrase. 动词短语→动词，名词短语

determiner→〔the〕. 定冠词→the

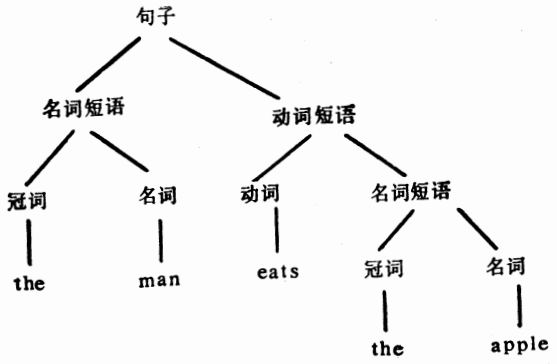
noun→〔apple〕. 名词→apple

noun→〔man〕. 动词→eats

verb→〔eats〕.

下页所示的图为其语法树。

显然，这种上下文无关文法的规则形式正好与PROLOG的规则形式一一对应，因此，很方便地用PROLOG 写分析句子结构，只需把每一条语法规则写成一个句子。下面一个程序能判断一个单词串是否为上述文法的一个句子。子句头sentence(X)意味着“X是符合上述文法的一个句子”。




```
sentence(X):-append(Y,Z,X),noun-phrase(Y),
                verb-phrase(Z).

noun-phrase(X):-append(Y,Z,X),determiner
                (Y),noun(Z).

verb-phrase(X):-append(Y,Z,X),verb(Y),noun
                -phrase(Z).

verb-phrase(X):-verb(X).

determiner([the]).

noun([apple]).

noun([man]).
```

```
verb([eats]).

?-sentence([the,man,eats,the,apple]).
yes
```

上述程序中sentence子句的append(Y,Z,X),是把句子X分成两部分Y和Z，然后判别它们是否分别为名词短语和动词短语。上述这个程序的最后提出询问，并回答yes。

但上面的程序会产生许多不必要的搜索。当询问?-sentence([the,man,runs])时，sentence子句的X被例示，而Y和Z未被例示，于是目标append(Y,Z,X)首先将Y和Z分别例示为Y=[ ]，Z=[the,man,runs]，但因为Y=[ ]不是名词短语，所以目标noun-shrase(Y)失败，失败则回溯，目标append(Y,Z,X)重新将Y、Z例示为Y=[the],Z=[man,runs]。不过Y仍不是名词短语，于是又产生回溯。如此重复，直到Y=[the,man]，Z=[runs]。此时Y为名词短语，Z为动词短语，整个目标成功。

为解决这种不必要的搜索，可以不使用append(Y,Z,X)，不明确地对X划分成分，而让每个语法单位自动地识别出单词串中相应的语法成分。这样把谓词sentence定义成：

```
sentence(S0,S):-noun_phrase(S0,S1),verb_phra-
                se(S1,S).
```

意思是，由sentence谓词从单词串S0中识别出符合句子定义的部分，其余部分放在S中。同样，其中的目标noun_phrase(S0,S1)是从S0中识别出名词短语，把其余部分放在S1。如果要识别一个句子，那么其余的部分S应为空，所以当询

问？-sentence([the,man,],[ ])时，S0被例示为[the, man]。目标noun_phrase(S0,S1)识别出这个句子的名词短语部分为[the,man],供谓词verb_phrase进一步分析，因而省去了不必要的回溯。这种由语法单位自动识别出单词串中相应的语法成分的程序如下：

```
sentence(S0,S):-noun_phrase(S0,S1),verb_phra-
                    se(S1,S).
noun_phrase(S0,S):-determiner(S0,S1),noun
                    (S1,S).
verb_phrase(S0,S):-verb(S0,S).
verb_phrase(S0,S):-verb(S0,S1),noun_phrase
                    (S1,S).
determiner([the|S0],S0).
noun([man | S0], S0)
noun([apple | S0], S0).
verb([eats | S0], S0).
? -sentence([the, man, eats, the, apple],[ ]).
yes.
```

上面程序中谓词determiner，noun和verb的子句中都有([_ | S0], S0)这种形式。例如，determiner([the | S0], S0),其中S0的作用可以通过下面例子说明。设有目标

```
? -noun_phrase([the, man], [ ]).
```

它匹配名词短语：

```
noun_phrase(S0, S):-determiner(S0, S1),
                    noun(S1, S).
```

当其中目标determiner([the, man],S1)匹配子句determ-

iner([the | S0], S0)时S0被例示为[man]，从而S1也被例示为[man]，再通过S1传递给noun(S1, S)。因此determiner([the|S0], S0)中的S起传递信息作用。

于是可以询问：

```
? -sentence([the, apple, eats, man]).
yes
```

这种句子语法结构正确，但含义不正确。

```
? -noun_phrase([the, man,eats,the,apple], L).
L=[eats, the, apple]
```

根据文法规则直接写出程序的方法比较自然，但一个缺点是只能针对某一给定文法，缺少灵活性，文法的修改对程序的确影响很大。应进一步讨论，在此省略。

第十节 专家系统

把某些专业人员行之有效的经验，得到社会公认，把这些专家经验加以整理提炼，用PROLOG语言写成事实和规则作为知识库，当用户提出咨询，要获得该专业知识的某些方面的问题，要在专家程序——知识库里进行推理给出用户所需的答案，或提供给用户有信赖性的参考建议等。这是PROLOG语言重要的应用领域之一。本节给出几个专家系统的实例。

1. 修理收音机专家系统

详见参考书附录三。

2. MYCIN专家系统

这是在医学领域诊断病时用药方面的专家系统。详见参考书第十一章11.2节。

3．小型动物分类专家系统

该系统与用户采用对话方式，来回答用户的询问，告诉用户这是何种动物。

当然，首先需把科学定义的事实和规则作为知识库，而后在库中进行推理来回答用户提出的问题。这个小型动物分类专家系统里，定义了七种动物的事实和规则。下面先介绍为推理提供的这七种动物的规则：

(1) 豹

如果动物是哺乳动物，且

是食肉动物，且

是黄褐色的，且

有黑色斑点，

那么该动物是豹。

```
animal_is(cheetah) if it_is(mammal) and
                      it_is(carnivore) and
                      positive(has, tawny_col-
                      or) and
                      positive(has, black_spots).
```

(2) 虎

如果动物是哺乳动物，且

是食肉动物，且

是黄褐色的，且

有黑条纹，

那么该动物是虎。

```
animal_is(tiger) if it_is(mammal) and
```

```
                    it_is(carnivore) and
                    positive(has, tawny_color)
                    and
                    positive(has, black_stripes).
```

(3) 长颈鹿

如果动物是有蹄类动物，且
　　有长脖子，　且
　　有长腿，　　且
　　有黑斑点，
那么该动物是长颈鹿。

```
animal_is(giraffe) if it_is(ungulate) and
                    positive(has, long_neck)
                    and
                    positive(has,long_legs)and
                    positive(has, dark_spots).
```

(4) 斑马

如果动物是有蹄类动物，且
　　有黑条纹，
那么该动物是斑马。

```
animal_is(zebra) if it_is(ungulate) and
                    positive(has,black_stripes).
```

(5) 驼鸟

如果动物是鸟，且
　不会飞，　且
　有长脖子，且
　有长腿，　且

是黑白色的，

那么该动物是驼鸟。

```
animal_is(ostrich) if it_is(bird) and
                      negative(does, fly) and
                      positive(has,long_neck)and
                      positive(has, long_legs)and
                      positive(has, black_and_
                      white_color).
```

(6) 企鹅

如果动物是鸟，且

不会飞，且

会游泳，且

是黑白色的，

那么该动物是企鹅。

```
animal_is(penguin) if it_is(bird) and
                      negative(does, fly) and
                      positive(does, swin) and
                      positive(has, black_and.
                      white_color).
```

(7) 信天翁

如果动物是鸟，且

善飞，

那么该动物是信天翁。

```
animal_is(albatross) if it_is(bird) and
                        positive(does, fly well).
```

另外再定义几个规则，解释如下：

(1) 如果动物有毛发，

那么该动物是哺乳动物。

it_is(mammal)if positive(has, hair).

(2) 如果动物有奶，

那么该动物是哺乳动物。

it_is(mammal)if positive(does, give_milk).

(3) 如果动物有羽毛，

那么该动物是鸟。

it_is(bird)if positive(has, feathers).

(4) 如果该动物会飞，且

会下蛋，

那么该动物是鸟。

it_is(bird) if positive(does, fly) and
positive(does, lay_eggs).

(5) 如果该动物吃肉，

那么该动物是食肉动物。

it_is(carnivore) if positive(does,
eat_meat).

(6) 如果该动物有犬齿，且

有爪，　且

眼盯前方，

那么该动物是食肉动物。

it_is(carnivore) if positive(has, pointed_
teeth)and
positive(has, claws) and
positive(has, forward_eyes).

(7) 如果该动物是哺乳动物，且

有蹄，

那么该动物是有蹄类动物。

```
it_is (ungulate) if it_is(mammal)and
                        positive(has, hooves).
```

(8) 如果该动物是哺乳动物，且

是嚼食动物，

那么该动物是有蹄类动物。

```
it_is(ungulate)if it_is(mammal)
                    and positive(does, chew_cud).
```

至此，可询问诸如"动物有毛发吗？"之类的问题了。若想推理新的问题，需向知识库加入相应子句。

程序中ask向用户询问，并记住用户的回答。

```
ask(X, Y):-write(X, 'it', Y, 'Yn'),
          readln(Reply), remember(X, Y,
          Reply).
```

若用户回答yes，则借助标准谓词asserta将此肯定事实加入数据库。

```
remember(X,Y,yes):-asserta(xpositive(X, Y)).
remember(X, Y, no):-asserta(xnegative(X,
                        Y)), fail.
```

该专家系统程序如下：

```
run:-animal_is(X),!,write('YnYour animal may
be a(n)',X),nl,nl,clear_facts.
run:-write('YnUnable to determine what'),
```

```
write('your animal is.ynyn'),clear_facts.

positive(X,Y):-xpositive(X,Y),!.
positive(X,Y):-not(xnegative(X,
Y)),!,ask(X,Y).
negative(X,Y):-xnegative(X,Y),!.
negative(X,Y):-not(xpositive(X,Y)),
ask(X,Y).

ask(X,Y):-write(X,'it',Y,'yn'),readln(Reply),
remember(X,Y,Reply).
remember(X,Y,yes):-asserta(xpositive(X,Y)).
remember(X,Y,no):-asserta(xnegative(X,Y)),
fail.

clear_facts:-retract(xpositive(_,_)),fail.
clear_facts:-retract(xnegative(_,_)),fail.
clear facts:-write('ynyb please press the space
            bar to Exit'),readchar( ).

animal_is(cheetah):-it_is(mammal), it
                    _is(carnivore),positive(has,
                    tawny_color),positive(has.
                    black_spots).
animal_is(tiger):-it_is(mammal),it_is(carnivore),
                  positive(has,tawny_color),
```

```
                    positive(has, black_stripes).
animal_is(giraffe):-it_is(ungulate), positive(has,
                    long_neck), positive(has, long_
                    legs), positive(has, dark_
                    spots).
animal_is(zebra):-it_is(ungulate), positive(has,
                    black_stripes).
animal_is(ostrich):-it_is(bird), negative(does, fly),
                    positive(hac, long_neck),
                    positive(has, long_legs),
                    positive(has, black_and_
                    white_color).
animal_is(penguin):-it_is(bird), negative(does,
                    fly), positive(does, swim),
                    positive(has, black_and_
                    white_color).
animal is(albatross):-it_is(bird), positive(does,
                    fly_well).

it_is(mammal):-positive(has, hair).
it_is(mammal):-positive(dose, give_milk).
it_is(bird):-positive(has, feathers).
it_is(bird):-positive(does, fly), positive(does,
                lay_eggs).
it_is(carnivore):-positive(does, eat_meat).
it_is(carnivore):-positive(has, pointed_teeth),
```

```
                positive(has,claws),positive(has,
                forward_eyes).
it_is(ungulate):-it_is(mammal),positive(has,
                hooves).
it_is(ungulate):-it_is(mammal),positive(does,
                chew_cud).
```

该程序共有27个子句，请依次标注序号①至㉗。如果用户提出如下询问：

```
? -run
```

则匹配过程是：子句①→⑬→⑳→③→④→⑦→⑧，得出该询问中的动物是豹。若用户不满意，即不是豹，便回溯至子句⑬，指针下移至⑭至⑲回答出其他六种动物。

习　题

1. 试编写求解下面联立方程的程序

$$X+1=4$$

$$X+Y=5$$

2. 把例6.33探索迷宫改写成求最短路径的程序。
3. 请给出例6.28程序中询问的答案。
4. 试根据你的经验编写个示意性的小型专家系统。

第七章 附 录

为了能应用参考书上的micro-PROLOG系统上机操作，特把本书有关的PROLOG系统规定，转换成参考书的micro-PROLOG的对应形式，不是全部转换成对应关系，只把程序转换过来，便可上机操作，其余规定详见参考书。

第一节 把本书的事实和规则转换成micro-PROLOG

	本书PROLOG语法	参考书micro-PROLOG语法
1	:-	用if
2	子目标之间用“,”号隔开	用and或&
3	参量之间用“,”号隔开	用空格
4	语句最后用句号“.”	不写
5	变量名第一个字母要大写，其后可跟字符	用x,y,z,X,Y,Z打头，其后可跟字符
6	无名变量用下线“_”	用上述有名变量代替
7	单词间用下线连接，如look_for	用中线连接，如look-for
8	谓词在目标的最左侧，前缀形式	可用前缀、中缀或后缀形式
9	询问句符号为？-	用whcih,one,is见参考书第一章
10	cut截断用“!”	用“/”
11	表用〔〕括起来	用( )括起来

据上述两种版本相互转换的规定，其例题程序见例1.1

所示。

第二节 上机操作

micro-PROLOG 3.1 及其以后的版本里，有最适宜初学者使用的simple模块，建议读者使用它调试程序。上述7.1节中的micro-PROLOG语法，主要针对simple而言。用simple模块简述上机操作步骤，把micro-PROLOG系统盘插入驱动器后：

(1) 当出现提出符A>时，键入prolog load simple回车。

(2) 当出现提示符&&.或&. 时，便可键入程序。

(3) 以例1.1程序为例，如下所示，键入语句：

```
&.add(likes(mary food))回车
      ⋮
```

凡是键入语句时都用此方法。询问时：

```
&.which(X:likes(mary X))回车
  food
  wine
No(more)answers
```

(4) 建议调用跟踪程序来观察程序运行过程，方法是：

```
&.load simtrace 回车
&.all-trace(X:look-for(bill,X))回车
```

详细操作过程请见参考书有关章节，并有丰富的练习题供选择使用。

参 考 资 料

[1] Coelho, H., J. C. Cotta, L. M. Pereira, How to Solve it with PROLOG, Laboratorio Nacional de Engenharia Civil, 1980.

[2] W. F. Clocksin, and C. S. Mellish Programming in PROLOG, Second Edition Berlin Heidelberg, 1984

[3] 金志权、陈佩佩，人工智能程序设计，南京大学出版社，1986年。

[4] 一些内部参考资料。