人工智能语言

中科大 自动化系　郑志刚

2018.10

提 纲

- 人工智能语言介绍
- Prolog语言(prolog是Programming in LOGic的缩写，意思就是使用逻辑的语言编写程序)

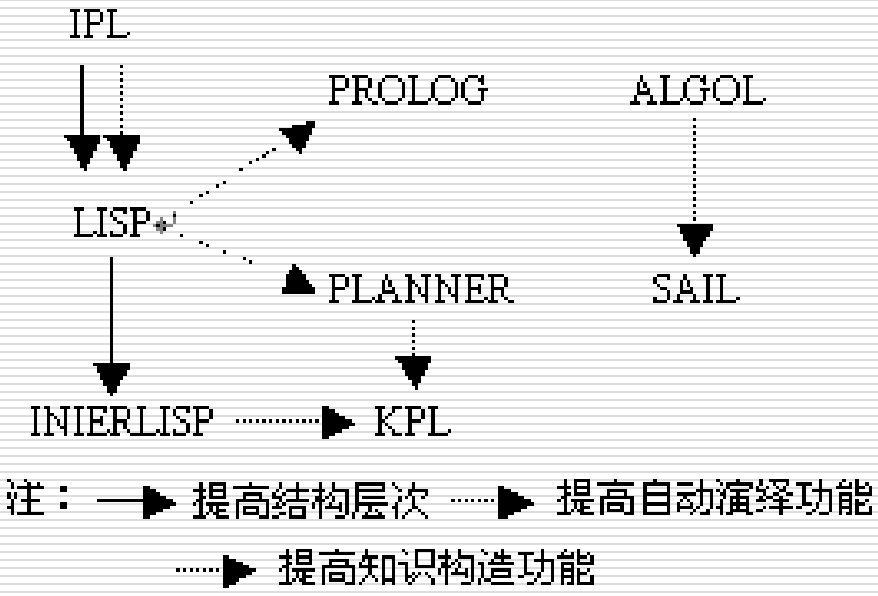
AI语言简介

- 什么是人工智能语言
  - 专用于人工智能和智能系统的、面向任务和知识、以知识表示和逻辑推理为目标的符号和逻辑处理编程语言
- 哪些常见的人工智能语言
  - LISP
  - PROLOG
- 学习AI语言的必要性
  - 人工智能是计算机科学的分支，人工智能的实现是以计算机为工具的，分为硬件实现和软件实现两个层次。前者借助于专用的人工智能机（与通用的计算机体系结构有区别）实现人工智能；后者采用通用的计算机，由软件实现人工智能，是目前实现人工智能的主要途径。因此，计算机软件设计是人工智能的关键

对符号和逻辑处理编程语言的要求

- 具有表结构形式。LISP的处理对象和基本数据结构是S表达式(即符号表达式)，具有一组用于表处理的基本函数，能对表进行比较自由的操作。PROLOG的处理对象是项。它是表的特例。由于这类语言都以结构数据作为处理对象，而且都具有对表的处理能力，因而特别适用于符号处理
- 便于表示知识和逻辑计算。例如，PROLOG是以一阶谓词为基础的，而一阶逻辑是一种描述关系的形式语言(Formal Language)，很接近于自然语言的描述方式。智能控制(如专家控制)系统中的大量知识都是以事实和规则的形式表示的，所以用PROLOG表示知识就十分方便
- 具有识别数据、确定控制匹配模式和进行自动演绎的能力。PROLOG具有搜索、匹配和回溯等推理机制，在编制问题求解程序时，无需编写出专用搜索算法。当用LISP编程时，不仅要对问题进行描述，而且要编写搜索算法或利用递归来完成求解
- 能够建立框架结构，便于聚集各种知识和信息，并作为一个整体存取
- 具有以最适合于特定任务的方式把程序与说明数据结合起来的能力
- 具有并行处理的能力

现有的符号和逻辑处理编程语言



图中：

IPL 很早期的表处理语言

LISP 目前应用最广泛的符号和逻辑处理语言

INTERLISP 新近开发的一种LISP方言，比纯粹LISP的规模大，并提供更广泛的数组能力

SAIL 是ALGOL语言的变种，具有支持相关存储器等附加特性

PLANNER 一种便于目标定向处理的早期语言

KPL 一种能够支持复杂框架结构的语言

PROLOG 一种基于规则的语言，把程序写为提供对象关系的规则

Prolog语言的历史

- 1970年,英国,爱丁堡大学,R.Kowalski提出
- 1972年,法国,马赛大学,Alain Colmerner研制
- 1977年,英国, ,爱丁堡大学,D.Warren , DEC-10
- 1986年,Borland公司, PC版本 Prolog
  - Turbo Prolog Toolbox

Prolog的应用领域

- DB
- ES
- 数理推理
- 抽象问题求解
- 定理证明
- 语义学
- 自然语言理解
- 结构设计

Prolog的语言特点

- 与一般程序设计语言差别很大
  - 一般程序设计语言：过程性语言
    - 需要详细地告诉计算机一步步怎样做
  - Prolog：描述性语言
    - 描述知道的事实和规则，问系统一些问题
    - 怎样求解问题,程序员无须知道
    - 告诉系统做什么，而不用告诉它怎样做
    - 系统会自动给出答案或证明结果

Prolog的语言特点(续1)

- Prolog语言的数据与程序结构统一
  - 结构统一：项
  - 项用于表示程序和数据
- Prolog自动实现模式匹配与回溯
- 大量递归
- 语言简明
  - 规则
  - 事实
  - 目标

Prolog的语言结构

- 数据结构:常量
  - 用途：对特定对象与关系的命名
  - 合法的常量
    - 整数
      - 182,129
    - 原子
      - 标识符：小写字母开头，可以由字母,数字和下划线构成
        aBC12_12, is_，若原子用单引号(’)括住，则可含有任何字符。
      - 符号：Prolog语言规定的符号集的非空序列
        ?, -, =
    - 符号集
      - ab..z
      - AB……Z
      - 01234567890
      - ? ” # $ ǂ & ’ ( ) = _ ~ ^ \ [ ] , { , - @ + ; * < >

Prolog的语言结构(续1)

- 数据结构:变量
  - 用途：用来表示还无法知道而且需要PROLOG来确定的客体
  - 变量的表示:变量名与标识符类似，不同的是以大写字母或下划线开头

    Variable, _ansure
  - 匿名变量：不需要知道它是什么以及具体名字,只是表示留出一个位置，例如：我们可能想知道是否有人喜欢跳舞，但不需知道这个人的名字，这时，就可以用匿名变量
  - 匿名变量的表示：_

Prolog的语言结构(续2)

- 结构
  - 用于构造PROLOG数据对象
  - 结构的构成

    函子(分量1, 分量2, …, 分量n)

    其中分量也可以是结构
  - 结构的例子

person(mary, address(zhongshan, 120))

person(name(mary), address (street (zhongshan), number(120)))

Prolog的语言结构(续3)

- 常用的结构形式
  - 函子(分量1, 分量2, …, 分量n)
  - 表
    - [a,b,c]　　.(a.(b.(c[])))
    - [a,b,[c,d]]
  - 表达式
    - 由PROLOG提供的各种运算符联合各种常量,变量和结构构成
    - X+Y&Z　　+(X&(YZ))

Prolog的语言结构(续4)

- PROLOG的统一数据结构:项(term)
  - 定义:

<项>::=<常量>|<变量>|<结构>|“(”<项>“)”

Prolog的程序结构

- 程序的构成
  - 事实
  - 规则
  - 询问

事实

- 用于说明客体的性质及客体之间的相互关系
  例如： likes(john, mary).
- 谓词：关系或性质
- 客体：自变量
- 事实的语义表示该语句恒为真，谓词和客体都用小写字母开头,变量的客体用大写字母开头,客体间用逗号“，”分开用一对园括号括住，最后用“.”号终结
- 谓词的含义完全由程序设计者定义
  famale(jane).　play(jone, mary, chess).

规则

- 关于客体及其关系的一般陈述，表明某些关系的成立要依赖于其他一些关系的成立
- 规则可以是事实的一些紧凑的表现形式
- 语法:规则头:-规则体

  likes(jihn, X) :- student(X).

  规则头　　如果　规则体　结束符
- 一般形式

  p:-p1,p2,p3,…,pn.

  语义: p1∧ p2∧ …∧ pn→p

事实+规则=知识库

- PROLOG语言根据提供的知识库(事实和规则)
- 推理询问的目标是否能够被满足
- 在这个过程中，推理自动进行

询问

- 询问：问系统的问题
- 形式：

  ?-p1,p2,…,pn.
- 询问的语义
  - p1∧ p2∧ …∧ pn为真吗?

一个简单的PROLOG语言的例子

```
likes(mary, wine).
likes(john, meat).
likes(john,X):-likes(mary, X).
?-likes(mary, wine).
?-likes(john,X).
```

事实

规则

询问

Prolog语言的搜索策略

- Prolog语言描述性语言
- 提供事实和规则,构成知识库
- 询问系统一些问题
- Prolog系统自动进行模式匹配和回溯，回答目标是否成立

例化和匹配

- 基本概念:例化, 解脱, 匹配
- 例子

play(mary, swim).
play(mary, tennis).
play(john,tennis).
play(john, football).
?- play(mary, swim).

问系统: mary 是否喜欢swim?
?-play(mary,swim).
系统的回答: yes
过程…

例化和匹配

- 匹配的含义
  - 一个未例化的变量可以和任何客体匹配,例化结果是该变量代表匹配客体
  - 一个常量只能和自己匹配
  - 如果两个函子一致，而且分量的个数一致，当对应的分量均匹配时，则这2个结构可以匹配
  - 2个事实匹配是它们的谓词相同，且对应的自变量相匹配
  - 与规则匹配实际上只是与规则头匹配，规则头的匹配与事实的匹配相同

例化和匹配

- 例化
  - 为了将目标和知识库中的规则或事实匹配,用常量将某个目标谓词中出现的变量代替
  - 一个变量已例化，表示该变量已代表了某个确定的客体
  - 一个变量未例化，表示该变量在某一时刻尚未代表某个客体
  - 子句使用中，所有变量开始都是未例化的
  - likes(john, swim). likes(john,football).
  - ?-likes(john,X).
  - X被例化成swim, 系统回答 X= swim.
  - 输入：　；继续求解
  - X被解脱(反例化,还原成变量)，系统回答：X＝football.
  - 输入：　；　　系统回答：　no

回溯

- 回溯：PROLOG系统重复地试图满足和重新满足联结问题中的目标的操作
- 例子

```
play(mary, swim).
play(mary,tennis).
play(john, tennis).
play(john,football).
?-play(mary,X), play(john,X).
```

回溯

play(mary, swim).
Play(mary,tennis).
play(john, tennis).
play(john,football).

?-play(mary,X), play(john,X).

PROLOG自动搜索过程

1. play(mary,X) 与 play(mary, swim)匹配成功，X被例化成swim
2. 第二个子目标 变成 play(john, swim)，搜索整个知识库，不能够被满足
3. 目标不能够被满足，X被解脱，变成未例化的变量：回溯
4. 系统重新满足第一个目标：play(mary,X)，与Play(mary,tennis)匹配成功，X被例化成tennis
5. 第二子目标变成play(john,tennis)
6. 系统将之与play(john, tennis)匹配成功
7. X＝tennis.
8. ;

Prolog搜索策略

- 从左到右依次满足各个子目标(深度优先)
- 对任何一个子目标，第一次搜索时，总是从知识库的顶部开始
  - 如果直接可以与事实匹配，则这个子目标成功，并完成在知识库中的可匹配事实处设置标识，例化相应的变量
  - 如果与某个规则的头相匹配，设置相应的标记并再从左到右设法满足规则体的所有子目标
  - 如果没有任何事实和规则与之匹配，该子目标失败，如果有左邻，试图重新满足左邻目标（回溯），否则搜索失败结束
  - 回溯时，需恢复例化的变量(解脱)，从相应的标志以后开始搜索

控制策略的例子

play(mary, swim).

play(mary,tennis).

play(john, tennis).

play(john,football).

diff(mary,john).

likes(X,Y):-play(X,Z), play(Y,Z), diff(X,Y).

?-likes(mary,W).

likes(mary, W) 与规则匹配，　X被例化成mary, Y变成W

3个子目标　play(mary,Z), play(W,Z), diff(mary, W).

控制策略的例子

```
play(mary, swim).
Play(mary,tennis).
play(john, tennis).
play(john,football).
diff(mary,john).
```

3个子目标：play(mary,Z), play(W,Z), diff(mary, W).

play(mary,Z)与play(mary, swim)匹配成功，Z例化成swim

另外2个子目标： play(W,swim), diff(mary, W).

play(W,swim)与play(mary, swim)匹配成功，W被例化成mary

第3个子目标变成：diff(mary,mary)在知识库中得不到满足，失败

回溯play(W,swim)从play(mary, swim)后开始重新匹配

搜索失败，目标1回溯， play(mary,Z)

play(mary,Z)从play(mary, swim)后匹配，与Play(mary, tennis)匹配成功，Z被例化成tennis

2个子目标变成了play(W, tennis), diff(mary, W)

控制策略的例子

play(mary, swim).
Play(mary,tennis).
play(john, tennis).
play(john,football).
diff(mary,john).

子目标 2：play(W, tennis)与play(john, tennis)匹配成功，W被例化成john

子目标 3： diff(mary, john)

与diff(mary,john)匹配成功

W＝john

prolog的内部谓词

- 1 .比较
  - 比较谓词是具有两个自变量中缀形式的内部谓词，有:
    - X＝Y等于
    - X＼＝Y不等于
    - X＜Y小于
    - X＞Y大于
    - X＝＜Y小于等于
    - X＞＝Y大于等于
    - X＝＝Y强等于

2.算术表达式求值

- prolog提供的求值表达式和建立算术表达式的内部谓词如下：
  - Xis Y　Y可被例化为算术表达式，该谓词首先对Y求值，得出一整数。如果X未例化，则X例化为这一结果，且目标成功。若X已例化则“is”退化为“＝”来判断成功与否。
  - X＋Y加法运算，求值时自变量必须例化为求整数值的结构
  - X－Y减法运算，求值时自变量必须例化为求整数值的结构
  - X ＊Y乘法运算，求值时自变量必须例化为求整数值的结构
  - X／Y整除法运算，求值时自变量必须例化为求整数值的结构
  - XmodY〖DW〗取模法运算，求值时自变量必须例化为求整数值的结构

3.输入输出

- prolog的I/O谓词是系统从输入流输入一个项、一个字符或一个文件，或把一个项，一个字符、一个文件输出到输出流，它们仅仅是功能性的，即仅完成一项动作
  - geto(X)若X与输入流中下一个字符匹配，目标成功
  - get(X) X与输入流中下一个打印字符匹配，目标成功
  - skip(X)读入并跳过当前输入流中的字符，直到找到一个字符可与X匹配
  - read(X)读入当前输入流的下一个项，并使之与X匹配
  - put(X)　把整数X输出到当前输出流，X须例化
  - nl.　对当前输出流输出控制符“回车换行”
  - tab(X)　输出X个空格符到当前输出流，X须例化
  - write(X)把项X输出到当前输出流
  - display(X)与write(X)相似，只是它忽略运算符说明以结构形式输出项
  - op(X,Y,Z)　说明优先级为X，结合规则为Y，名为Z的运算符

4 控制谓词

- ture该目标永远成功
- fail　该目标永远失败，使用fail可以强制回溯，从而得到所有的解。
- repeat　repeat在回溯中总使目标成功，它和fail联用可产生循环，例：goal:　repeat,write('a')，fail
- ?：-goal
- 执行这个目标的结果是输出aaaa……，这是因为遇到repeat时目标成功，向下执行write('a')遇到fail，失败引起回溯。回溯到repeat时又使目标成功。从而产生无限循环的结果。
- 截断谓词!

截断谓词! (cut)

- !谓词作用的介绍
  - 回溯是Prolog必要的功能
  - 但有些回溯是不必要的
  - !可以改变回溯次序，使得PROLOG搜索按照设想的方向进行
  - 截断“!”的含义是：作为一个目标，它直接成功，但当回溯到它时，不能重新被满足，并使其双亲目标立即失败。双亲目标是触发含有!的子句的目标。即，穿过“!”的回溯是不可能的

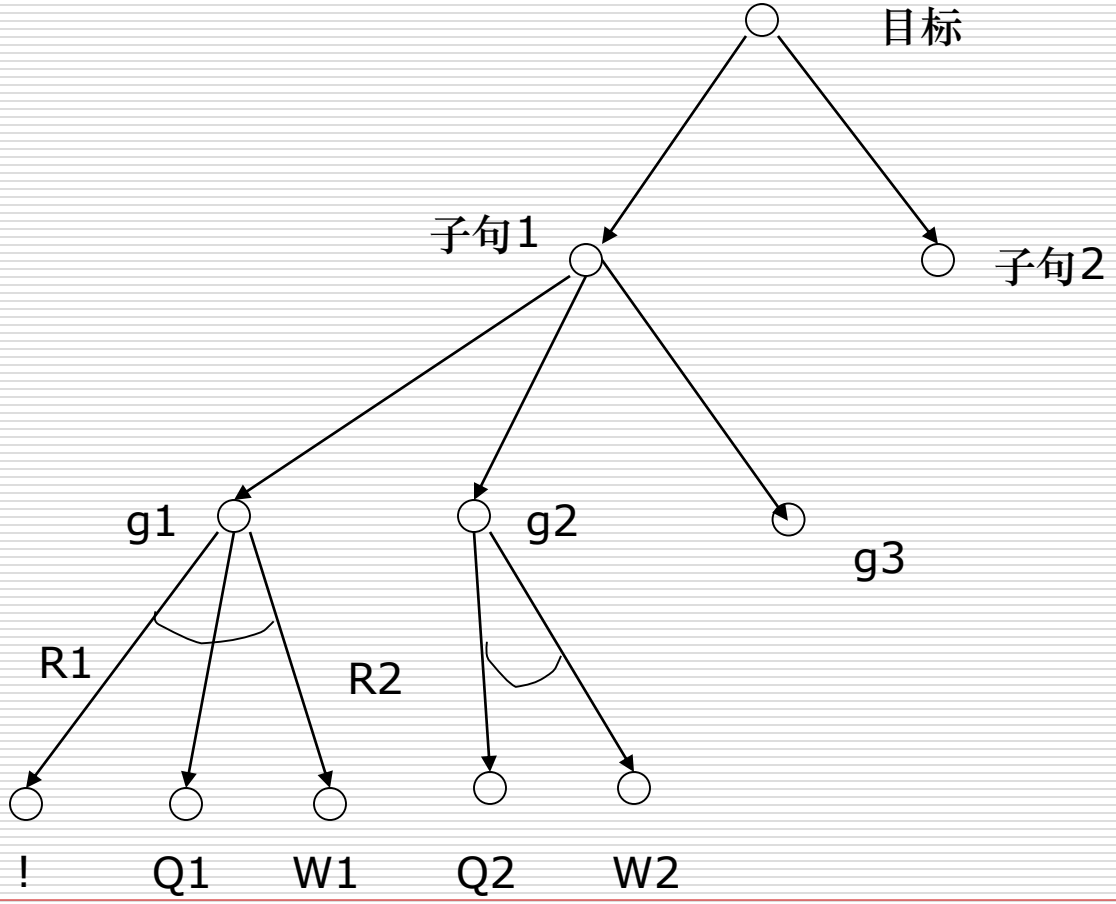
!谓词的优点

- 可以使程序运行的更快,因为它不会试图满足那些事先知道的不可能导致解答的目标
- 程序可以占用较少的内存空间,系统不会记忆那些对求解毫无意义的回溯点

!的作用

- 影响回溯的专用谓词
- !: 要回溯前面一串已满足的目标时，就不必要考虑那些目标的其它可选分枝
- 由程序员指定哪些回溯是无意义的
- !可以被看成一个目标，是一个没有变量的谓词
- 可以立即被满足
- 但不能够重新被满足

!控制回溯的例子



!的用法1:分支选择

- 当情况1->做动作1
- 当情况2->做动作2
- …
- 当情况n->做动作n
- 否则->做动作n+1

!的用法1:分支选择的例子

Rule 1：若X<3， 则Y=0

Rule 2：若X>=3且X<6， 则Y=2

Rule 3：若X>=6， 则Y=4

在prolog中可写成一个二元关系f(X,Y):

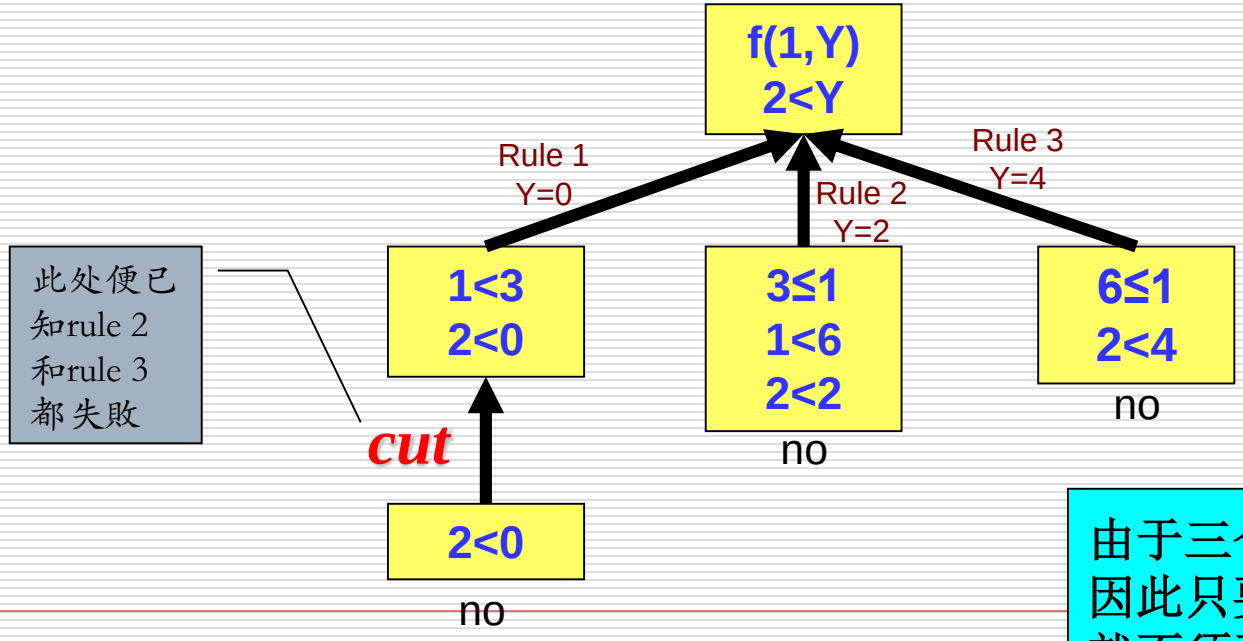
```
f(X,0) :- X<3.
f(X,2) :- 3=<X, X<6.
f(X,4) :- 6=<X.
```

询问:

```
?- f(1,Y), 2<Y.
```

结果会失败

- 上例中，prolog已经透过回溯尝试了两个无用的方法，其详细执行过程如下图：



由于三个**Rule**是互斥的，因此只要有一个**Rule**成功，就不须再去尝试其它的**Rule**

- 切断的表示法：!
  - 范例：将上述三個rule重写

    f(X,0) :- X<3, !.

    f(X,2) :- 3=<X, X<6, !.

    f(X,4) :- 6=<X.
  - 会防止在它出现的地方产生回溯，因此在上例中，prolog将只会产生图左边的分枝。当2<0失败，prolog试着回溯，但不会超过”!”的程序部分，故对应到rule 2和rule 3的分枝将不会产生
  - 使用切断的优点：较有效率

!的用法2: 排除性选择

- 与 fail谓词合用
  - fail- 一种内建谓词
  - 象!一样没有变量
  - fail 作为一个目标总是失败,并引起回溯
  - 与!合用, 排除性选择

排除性选择的例子

- 例：除了蛇以外，Mary喜欢所有的动物
  - 如果X是动物的话，那么Mary喜欢X
    - likes(mary, X) :- animal(X)
  - 上述必须要将「蛇」排除在外，因此改成
    - 如果X是蛇的话，那么Mary喜欢X便不成立，否则如果X是一种动物的话，那么Mary喜欢X
      - likes(mary, X) :- snake(X), !, fail.
      - likes(mary, X) :- animal(X)

!的用法3:结束“生成测试”

- 把！放在“产生器”、“测试器”的后面,形式为：example(X):-generate(X),test(X),!.
- 示例：divide(N1,N2,R):-is_integer(R) ,P1=R*N2,P2=(R+1)*N2,P1=<N1,P2>=N1,!.

  is_integer(0).

  is_integer(X):-is_integer(N),X=N+1.

- 这是一个先生成后测试的例子，规则体中的integer是一个产生器，产生从0开始的整数串，其余目标为测试器。判断生成的整数是否满足目标，当给定N 1 和N2后，仅有一个R能满足，故子句中最后的!，是防止integer产生无限的整数来不断测试，也即当找到唯一解后，就不用找其它解了。
- ?-divide(25,6,R).

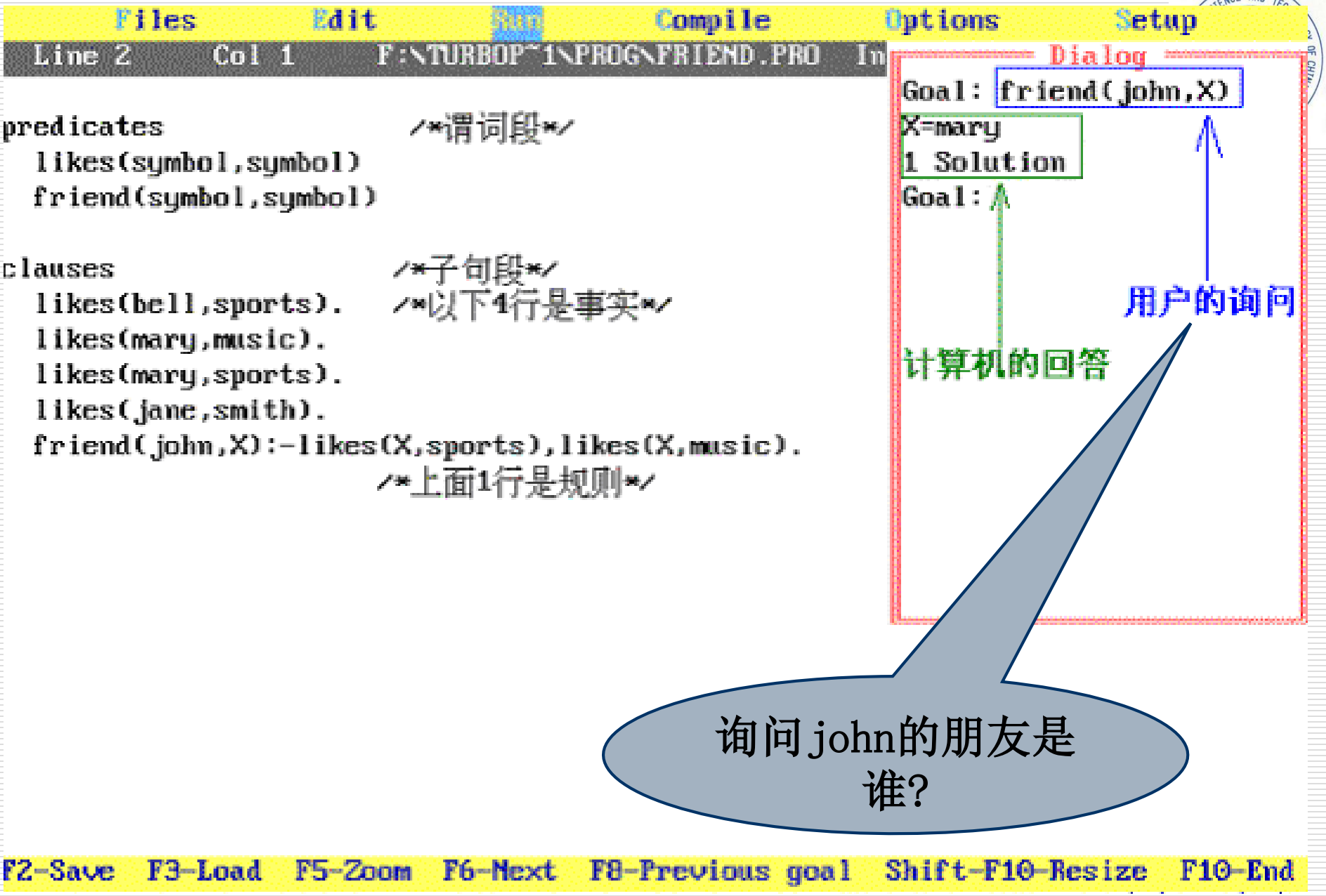
  R=4.

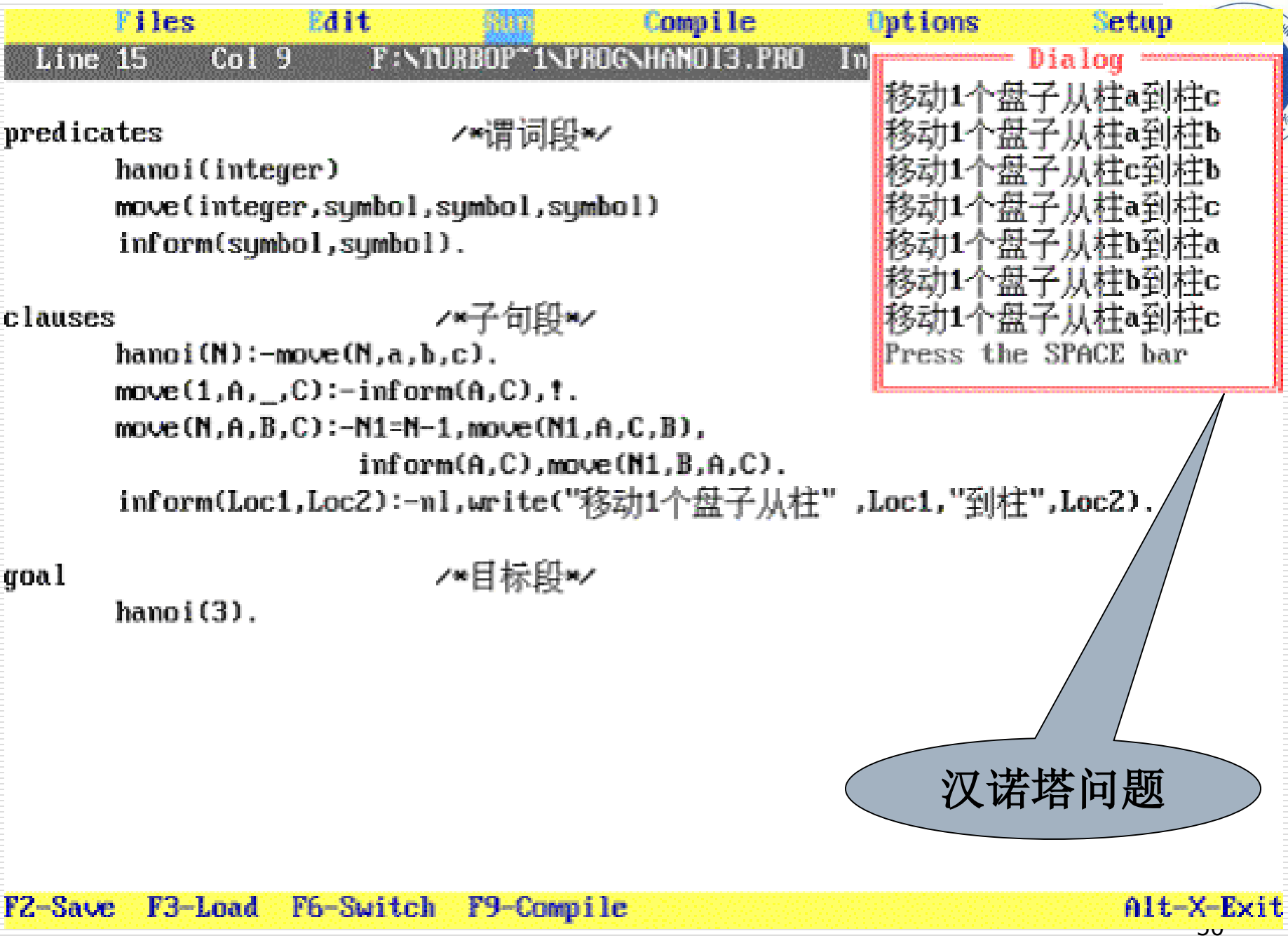
Prolog语言的常用版本

- Turbo Prolog
  - 由美国Prolog开发中心（Prolog Development Center, PDC）1986年开发成功、Borland公司对外发行，其1.0，2.0，2.1版本取名为Turbo Prolog，主要在IBM PC系列计算机，MS-DOS环境下运行。

Prolog语言的常用版本（续）

- Visual Prolog
  - Visual Prolog是基于Prolog语言的可视化集成开发环境，是PDC推出的基于Windows环境的智能化编程工具。
  - 目前，Visual Prolog在美国、西欧、日本、加拿大、澳大利亚等国家和地区十分流行，是国际上研究和开发智能化应用的主流工具之一。
  - Visual Prolog软件的下载地址为：http://www.visual-prolog.com


Files Edit Run Compile Options Setup
Line 2 Col 1 F:\TURBOP~1\PROG\FRIEND.PRO In
predicates /*谓词段*/
likes(symbol,symbol)
friend(symbol,symbol)
clauses /*子句段*/
likes(bell,sports). /*以下4行是事实*/
likes(mary,music).
likes(mary,sports).
likes(jane,smith).
friend(john,X):-likes(X,sports),likes(X,music).
/*上面1行是规则*/
Dialog
Goal: friend(john,X)
X=mary
1 Solution
Goal:
用户的询问
计算机的回答
询问john的朋友是谁?
F2-Save F3-Load F5-Zoom F6-Next F8-Previous goal Shift-F10-Resize F10-End


Files Edit Run Compile Options Setup
Line 15 Col 9 F:\TURBOP~1\PROG\HANOI3.PRO In
predicates /*谓词段*/
hanoi(integer)
move(integer,symbol,symbol,symbol)
inform(symbol,symbol).
clauses /*子句段*/
hanoi(N):-move(N,a,b,c).
move(1,A,_,C):-inform(A,C),!.
move(N,A,B,C):-N1=N-1,move(N1,A,C,B),
inform(A,C),move(N1,B,A,C).
inform(Loc1,Loc2):-nl,write("移动1个盘子从柱" ,Loc1,"到柱",Loc2).
goal /*目标段*/
hanoi(3).
Dialog
移动1个盘子从柱a到柱c
移动1个盘子从柱a到柱b
移动1个盘子从柱c到柱b
移动1个盘子从柱a到柱c
移动1个盘子从柱b到柱a
移动1个盘子从柱b到柱c
移动1个盘子从柱a到柱c
Press the SPACE bar
F2-Save F3-Load F6-Switch F9-Compile Alt-X-Exit

汉诺塔问题

Turbo Prolog程序结构

一个Turbo Prolog程序通常包括5个部分。如下：

```
/*          注释          */
domains
            域说明
database
            数据库说明
predicates
            谓词说明
goal
            目标说明
clauses
            子句说明
/*          注释          */
```

1.域说明部分，说明谓词对象的数据类型。

2.数据库说明部分，包含一些数据库谓词的定义，是说明用于动态数据库管理的谓词。如果程序不需要动态数据库，该部分可省略。

3.谓词说明部分，定义程序中除内部谓词以外的所有谓词。

4.在目标部分，说明程序的目标。一个程序目标可以由多个字目标复合而成。

5.子句部分，列出全部事实和规则，也可看作是程序的静态数据。

Turbo Prolog程序分析

```
/*    程序名：Wordsmith    文件名：prog0501.pro
*/
/*    目  标：查找并打印一单词的同义词或反义词。 */
domains
        word,syn,ant=symbol
predicates
        synonym(word,syn)
        antonym(word,ant)
goal
        synonym(brave,X),
        antonym(brave,Y),
        write("brave  的同义词是:  ",X), nl,
        write("brave  的反义词是:  ",Y), nl.
clauses
        synonym(brave,daring).
        synonym(honest,truthful).
        synonym(modern,new).
        synonym(rare,uncommon).
        antonym(brave,cowardly).
        antonym(honest,dishonest).
        antonym(mordern,ancient).
        antonym(rare,common).
```

Turbo PROLOG的程序结构

…….（编译指令）
constants
……（常量说明）
domains
……（域说明）
database
……（数据库说明）
predicates
……（谓词说明）
goal
……（目标子句）
clauses
……（子句集）

predicates ……► 必需部分
goal ……► 必需部分
clauses ……► 必需部分

Turbo PROLOG的程序结构（续）

```
/*Example 1*/
  domains
      name=symbol
  predicates
      likes(name,name)
      friend(name,name)
  goal
      friend(john,Y),write( “Y=” ,Y).
  clauses
      likes(bell,sports).
      likes(mary,music).
      likes(mary,sports).
      likes(jane,smith).
      friend(john,X):-likes(X,sports),likes(X,music).
      friend(jonn,X):-likes(X,reading),likes(X,music).
```