School of Data and Computer Science, Sun Yat-sen University

prolog 简介

October 1, 2020

目录

1 prolog 简介

Prolog 是一种逻辑编程语言。它创建在逻辑学的理论基础之上，最初被运用于自然语言等研究领域。现在它已广泛的应用在人工智能的研究中，它可以用来建造专家系统，自然语言理解，智能知识库等。

目前来说，Prolog 主要用在人工智能和计算机语言的研究领域。有别于一般的过编程语言，prolog 的程式是基于谓词逻辑的理论。最基本的写法是定立物件与物件之间的关系，之后可以用询问目标的方式来查询各种物件之间的关系。系统会自动进行匹配及回溯，找出所询问的答案。

2 prolog 安装

具体下载可去 https://www.swi-prolog.org/Download.html 下载安装，根据系统不同自行选择。

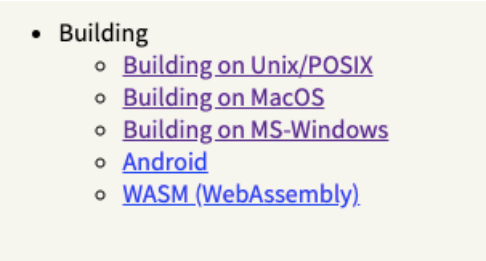

图 1: 根据不同系统自行选择

3 prolog 语法

3.1 启动 prolog

Linux 和 Mac 都可以通过在控制台输入 swipl 启动 prolog，window 系统在搜索框输入 prolog，点击图标即可进入。当你看到?- 的时候，你就成功了。

?- 是命令提示符，我们来看一个例子：

```
?- write("Hello World!").
Hello World!
true.
```

prolog 所有的语句都会用 . 来标识结束。如果想要换行，可以在中间加入 nl 。退出可以使用 halt 命令。

```
?- write("Hello"), nl, write("world!").
Hello
world!
true.
```

3.2 常量，变量

小写字符串开头的就是常量，大写字符开头的就是变量，如：`abc`，这是常量。`Abc`，这是变量。

```
?- write(abc).
abc
true.

?- write(Aewhfrui).
_14070
true.
```

3.3 关系，属性

两个常量之间的关系,我们可以使用小括号来标识,如 mike 是 jack 是爸爸,可以表示成:`father(mike, jack).`，这只是单方面的关系，反过来是不一定成立的。`friend(x, y).` x 是 y 的朋友，但 y 不一定是 x 的朋友(至少 prolog 不这么认为)。如果要用 frind 来表达 x,y 是朋友,则需要写两遍了。`friend(x, y). friend(y, x).`

两个常量之间是用小括号表达,但是如果是只有一个参数,那就表示该常量拥有这个属性。如`male(jack).`，jack 拥有男性这个属性。

3.4 规则

规则也就是推理的依据，我们可以根据规则从一个已有的结论推理出另外一个结论。如 x 和 y 是朋友，我们需要重复两遍：

```
friend(jack, mike).
friend(mile, jack).
```

如果使用规则，我们可以写成：

上述代码的`X，Y`都是大写，所以是变量，符号`:-`代表了推理关系，和蕴含`<-`意思相同，只要右边表达式为真，左边就可以满足。所以如果加上这条规则，只需要写`friend(jack, mike).`就可以表示 jack 和 mike 是朋友了。如果一条规则需要应用多个判断条件，如表示`X`是`Y`的儿子：

```
friend(X,Y):- friend(Y,X).
```

```
son(X,Y):-male(X), ((father(Y, X);mother(Y,X))).
```

X 是 Y 的儿子首先要求 X 是男生，其次 Y 可能是 X 的父亲也可能是母亲，这里就存在了一个或的关系。在 prolog 中，我们使用 , 和 ; 表达“且”和“或”的关系。

如果我们需要对某个条件做否定，比如我们定义单相思，相思用谓语 love(X,Y). 来定义。于是单相思可以定义成:

```
onesidelove(X,Y):-love(X,Y), \+ love(Y,X).
```

其中，\+ 后面紧跟的条件如果是 false 才能匹配当前的规则。

3.5 比较

我们需要知道两个比较操作符号 = 和 \=

```
1 X = a, X = b.
2 X = a, Y = X.
```

第一个会返回 false。如果变量没有被初始化，而等号的另一边是常量，这个时候常量就会赋值给变量；如果变量已经被初始化完成了，那么等号就会起比较的作用了。X = a 中 X 没有被赋值，所以 X = a 相当于把 a 赋值给 X。X = b 要执行时 X 已经被 a 赋值了，所以等于号起了比较的作业，X = b 返回 false。

第二个返回 Y = a，相当于是赋值，首先 X 没有被初始化，所以 X 会被赋值 a，Y 也没有被初始化，而 X 被初始化了，Y 也被初始化成了 a。

```
1 X = a, X \= a.
```

返回 false, 'X = a' 会把 a 赋值给 X, 然后对比 X 和 a，\= 表示如果不等则返回 true。

3.6 Setof

现在存在一个知识库如下所示。

```
1 age(peter, 7).
2 age(ann, 5).
3 age(pat, 8).
4 age(tom, 5).
```

```
age(ann, 5).

like(jack, ann).
age(X, 8) :- like(X, ann).
```

我们可以使用 Setof 来获取所有具有 Age 属性的孩子.(也 age(X, Y).)

```
?- setof(Child, age(Child, Age), Results).
Age = 5,
Results = [ann, tom] ;
Age = 7,
Results = [peter] ;
Age = 8,
Results = [jack, pat].
```

任何出现在“age(Child, Age)”中的变量（如 Child，Age），那个没有出现在第一个参数变量里面的（如 Age），“setof”会根据这个参数返回所有的可能性。

我们可以通过另外一种方式完成:

```
?- setof(Age/Child, age(Child, Age), Results).
Results = [5/ann, 5/tom, 7/peter, 8/jack, 8/pat].
```

如果我们不关心分类，只想返回所有的结果。

```
?- setof(Child, Age^age(Child, Age), Results).
Results = [ann, jack, pat, peter, tom].
```

可以解释成：我们要寻找所有具有年龄属性的孩子，并把结果放到 Results 中。

3.7 查询

规则既然已经定义好了,那我们就应该把规则导进来。先写好一个脚本 rules.pl ,然后通过 ?- [rules] 导入脚本。(如果你不知道你的 prolog 默认读取文件的路径，可以用 ?- pwd 查看路径)

```
friend(jack, mike).
like(a,b).
friend(X,Y):-like(X,Y).

love(jack, mike).
onesidelove(X,Y):-love(X,Y), \+ love(Y,X).
```

图 2: rules.pl 内容

```
[?- pwd.
% /Users/GreenArrow/
true.

[?- [rules].
true.
```

图 3: 先查看 prolog 默认文件路径，把 rules.pl 放置到该路径下，再用 [rules] 命令读取

接下来我们查询即可。

```
[?- friend(a, b).
true.

[?- friend(jack, mike).
[true .

[?- onesidelove(jack,mike).
true.
```

返回 true. 则说明查找成功。还可以查询 jack 的朋友有哪些：

```
[?- friend(jack, Who).
[Who = mike .

?-
```

4 样例

4.1 阶乘

```
factorial(N){
    if(N == 0 || N == 1) return 1;
    return factorial(N-1) * N;
}
```

首先，我们要给出初始条件，当 N 为 1 或者是 0 时需要返回 1。

```
factorial(1, 1).
factorial(0, 1).
```

当 N > 1, N = N - 1, 返回 N * factorial(N-1). 但这里是 factorial(N, Return), prolog 没有返回，所以我们需要先计算 factorial(N-1, Return1), 然后 Return = Return1 * N.

```
factorial(N,Ret) :- N > 1, N1 is N - 1, factorial(N1, Ret1), Ret is N *
    Ret1.
```

4.2 Fibonacci

```
fibonacci(N){
    if(N == 1 || N == 2) return 1;
    return fibonacci(N-1) + fibonacci(N-2);
}
```

相似的，我们先给出初始条件。

```
fibonacci(1, 1).
fibonacci(2, 1).
```

当 N > 2, fibonacci(N-1) + fibonacci(N-2), 但这里是 fibonacci(N, return), N1 = N-1, N2 = N-2, 先计算 fibonacci(N1, Return1) 和 fibonacci(N2, Return2) 得到 Return1 和 Return2, Return = Return1 + Return2。

```
fib(N,Ret) :- N > 2, N1 is N -1, N2 is N -2, fib(N1,Prv1), fib(N2,Prv2),
    Ret is Prv2 + Prv1.
```

如果 N > 2, 然后计算 fib(N1,Prv1), fib(N2,Prv2). Ret = Prv2 + Prv1 就是最后的结果。

4.3 Hanio

```
move(N, A, B, C){
    if(N==1){
        move A to C;
    }
    else
    {
        move(N-1,A,C,B);
        move A to C;
        move(N-1,B,A,C);
    }
}
```

如果 N == 1, 我们可以直接把物体从 A 移动到 C，不需要借助任何东西。但如果 N > 1, 我们必须首先把 N-1 个物体借助 C 从 A 移动到 B, 然后把最后一个物体直接从 A 移动到 C. 最后，重新把 N-1 个物体从 B 借助 A 移动到 C。

定义要解决的物体:

```
hanio(N) :- move(N, a, b, c).
```

我们应该把 N 个物体从 A 移动到 C。

特别的,

```
move(1,A,_,C):-inform(A,C).
```

当只有一个物体，我们不需要借助任何东西就可以直接从 A 移动到 C。

```
move(N,A,B,C):-N1 is N-1,move(N1,A,C,B),inform(A,C),move(N1,B,A,C).
```

下一步，我们将 N-1 个物体从 A 移动到 B 借助 C (move(N1,A,C,B)), 然后把最后一个从 A 移动到 C(inform(A,C)), 最后又重新把 N-1 个物体从 B 移动回 C(move(N1,B,A,C)).

定义 inform 函数:

```
inform(Loc1,Loc2):-nl,write('from '),write(Loc1),write(' to '),write(
    Loc2).
```