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

prolog introduction

October 1, 2020

目录

1 prolog introduction

Prolog is a logic programming language. It was built on the theoretical basis of logic and was initially used in research fields such as natural language. Now it has been widely used in artificial intelligence research, it can be used to build expert systems, natural language understanding, intelligent knowledge bases, etc.

From now on, Prolog is mainly used in the domain of artificial intelligence and computer language. Different from general programming languages, prolog programs are based on the theory of predicate logic. The most basic way of writing is to establish the relationship between the object and the other, and then you can query the relationship between various objects by querying. The system will automatically match and backtrack to find out the answer to the question.

2 prolog installation

Download at https://www.swi-prolog.org/Download.html.According to different systems ,you should choose the one who is compatible.

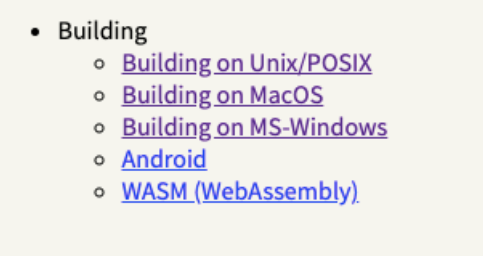

图 1: ch

3 prolog grammar

3.1 Getting started

Both Linux and Mac can start prolog by input ”swipl” in the console, About Window system, you should search ”prolog” and click the icon. When you see ?-, you are successful.

?- is common groups, eg:

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

All clause in prolog will be end with `.`. If you want to wrap, you can add `nl` in the middle.

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

If you want to exit prolog console, you can input `halt.`.

3.2 Constant variable

The beginning of a lowercase string is a constant, and the beginning of an uppercase character is a variable, example: `abc`, this is a constant. `Abc`, this is a variable.

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

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

3.3 Relationship attribute

The relationship between two constants, we can use parentheses to specify , such as mike is jack's father, we can write as: `father(mike, jack).`. This is only a unilateral relationship, and the reverse is not necessarily true. `friend(x, y).` x is a friend of y, but y may be not his friend.If "friend" specify x and y are friends, we have to write again。`friend(x, y). friend(y, x).`

The two constants are expressed in parentheses, but if there is only one parameter, it means that the constant has this attribute.like `male(jack).`, jack is male.

3.4 rules

Rules are the basis of reasoning. We can infer another conclusion from an existing conclusion based on the rules. If x and y are friends, we need to repeat it twice:

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

Expressing by rules:

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

X, Y are uppercase, they are variable, :- is reasoning, the same as <-. As long as the formula on the right side is true ,the formula on the left is true .If we augment with this rule , friend(jack, mike). can expressed as jack and mike are friends. If a rule needs to apply multiple judgment conditions, like X is Y 's son:

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

X is Y 's son. Firstly, X is male, Secondly, Y may be X 's father or mother, there is a "or" relationship.In prolog, the , and ; are "and" and "or" respectively .

If we want to set a rule is false, such as Unrequited love, love is love(X,Y). And Unrequited love can be defined as:

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

If the rule following \+ is false,the whole rule can be matched.

3.5 Compare

There are two compare operations. = and \=

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

The first one will return false. When one of the variables is not assigned and the other is a constant, the variables will be assigned a constant. And then the equal = will compare them.

The second one will return Y = a.When both are variables and one of them is already assigned, the unassigned variable will be assigned.

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

Return false, 'X = a' will assign 'a' for X, and compare X and 'a'.

3.6 Setof

We have a knowledge base .

```
age(peter, 7).
age(ann, 5).
age(pat, 8).
age(tom, 5).
age(ann, 5).

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

We can use Setof to get all the Children who has "age".(satisfy the age(X, Y).)

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

If any variables are used in "age(Child, Age)", which do not appear in the first argument(such as "Age"), "setof" will return a separate results for all possible.

We can do this in another way:

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

If we don't care about "Age" :

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

Read: Find the all Children, such that the Child has an Age, and put the results in Results.

3.7 query

Now, we should import the rules. prepare a script rules.pl , then import the script by ?- [rules] . (If you don' t know the path where your prolog reads the file by default, ?- pwd can help you.)

```
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 contain

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

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

图 3: First check the default file path of prolog,Put rules.pl under this path,Use the [rules] command to import

Query:

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

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

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

return true. means prolog find the matching one .We can also check what friends does Jack have:

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

?-
```

4 Example

4.1 Factorial

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

First, we need to set the initial condition. When N == 1 || N == 0 ,return 1.

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

When N > 1, N = N - 1, return N * factorial(N-1). But here is factorial(N, Return), so we should calculate factorial(N-1, Return1) firstly, and then 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);
}
```

Similarly, set initial condition:

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

When N > 2, fibonacci(N-1) + fibonacci(N-2), but here is fibonacci(N, return), N1 = N-1, N2 = N-2, calculate fibonacci(N1, Return1) and fibonacci(N2, Return2) to get Return1 and 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.
```

If N > 2, then calculate fib(N1,Prv1), fib(N2,Prv2). Ret = Prv2 + Prv1 is the final return.

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);
    }
}
```

If N == 1, we can move the object from A to C directly. But if N > 1, we have to move N-1 objects from A to B with the help of C firstly, then move the last one from A to C directly.Finaly, move N-1 objects from B to C with the help of A.

Definition the problem:

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

We should move N objects from A to C.

In particular

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

When the numbers of objects just one,move from A to C directly.

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

Next, we move N-1 objects from A to B with the help of C (move(N1,A,C,B)), then move A to C.(inform(A,C)), finaly move n-1 objects from B to C with the help of A(move(N1,B,A,C)).

Definotion inform

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