Introduction to AI with Prolog

Recursion

Alexey Sery
a.seryj@g.nsu.ru

Novosibirsk State University
*THE REAL SCIENCE

НГУ ФИТ

Department of Information Technologies
Novosibirsk State University

September 1, 2021

Goals of the lesson

- Introduce recursive definitions in Prolog
- Discuss the mismatches between the declarative meaning of a Prolog program, and its procedural meaning

Recursive definitions

Predicates can be defined recursively. It means that one or more rules in predicate's definition refers to itself.

The most common example is a description of family relationships.

```
ancestor(Anc,Dec) :- child(Dec,Anc).
ancestor(Anc,Dec) :- child(Dec,Par), ancestor(Anc,Par).
child(jane, peter).
child(mary, kate).
child(sam, mary).
child(jason, jane).
child(ann, jason).
```

Predicates can be defined recursively. It means that one or more rules in predicate's definition refers to itself.

Basis

```
ancestor(Anc,Dec) :- child(Dec,Anc).
ancestor(Anc,Dec) :- child(Dec,Par), ancestor(Anc,Par).
child(jane, peter).
child(mary, kate).
child(sam, mary).
child(jason, jane).
child(ann, jason).
```

Predicates can be defined recursively. It means that one or more rules in predicate's definition refers to itself.

Step

```
ancestor(Anc,Dec) :- child(Dec,Anc).
ancestor(Anc,Dec) :- child(Dec,Par), ancestor(Anc,Par).
child(jane, peter).
child(mary, kate).
child(sam, mary).
child(jason, jane).
child(ann, jason).
```

Recursive definitions

Next, let us define natural numbers, having a definition of zero and a rule of succession. This representation is variously called successor arithmetic, successor notation and also **Peano arithmetic**.

```
num(0).
num(s(N)) :- num(N).
```

Recursive definitions

Next, let us define natural numbers, having a definition of zero and a rule of succession. This representation is variously called successor arithmetic, successor notation and also **Peano arithmetic**.

```
num(0).
num(s(N)) :- num(N).

sum(0, N, N).
sum(s(M), N, s(S)) :- sum(M, N, S).
```

Next, let us define natural numbers, having a definition of zero and a rule of succession. This representation is variously called successor arithmetic, successor notation and also **Peano arithmetic**.

```
num(0).
num(s(N)) :- num(N).

sum(0, N, N).
sum(s(M), N, s(S)) :- sum(M, N, S).
```

Our numbers will be something like: $0, s(0), s(s(0)), s(s(s(0))), \ldots$ Not very pleasant if you want to compute something, but sometimes used in automated provers.

Tail recursion

Suppose we want to compute factorial (in normal numbers this time). We already know about recursion, and after a while will end up with a program similar to this one:

```
f(0,1).
f(N,F) :- N > 0, M is N - 1, f(M,Acc), F is Acc*N.
```

Binary predicate `f/2` takes two arguments: natural number N and factorial of N. The table below shows possible queries.

`f(4, F).`	`F = 24.`
`f(5, 120).`	`true.`
`f(X, 120).`	`ERROR`

Tail recursion

A recursive function is **tail recursive** when recursive call is the last thing executed by the function. It works in Prolog just well. Let us rewrite our factorial function, and make it tail recursive.

```
f(N,N,F,F) :- !.
f(N,M,Acc,F) :- M1 is M + 1, Acc1 is Acc * M1, f(N,M1,Acc1,F).
factorial(N,F) :- f(N,1,1,F).
```

As we can see predicate `f` takes 4 arguments now: natural number N, computation step M, result computed on step M, and total result F. Below are some examples.

`factorial(4, F).`	`F = 24.`
`factorial(5, 120).`	`true.`
`factorial(X, 120).`	`X = 5.`

Clause ordering, goal ordering, and termination

Recall our family program which as we know works just fine:

```
ancestor(Anc,Dec) :- child(Dec,Anc).
ancestor(Anc,Dec) :- child(Dec,Par), ancestor(Anc,Par).


child(jane, peter).
child(mary, kate).
child(sam, mary).
child(jason, jane).
child(ann, jason).
```

Clause ordering, goal ordering, and termination

But what happens if we swap goals in the second rule?

```
ancestor(Anc,Dec) :- child(Dec,Anc).


ancestor(Anc,Dec) :- ancestor(Anc,Par), child(Dec,Par).
child(jane, peter).
child(mary, kate).
child(sam, mary).
child(jason, jane).
child(ann, jason).
```

If we pose query `ancestor(jane, ann)` we will get a correct answer (`true`), and then an error message which means that Prolog is looping.

Clause ordering, goal ordering, and termination