CS 420 Advanced Programming Languages
Fall Semester, 2022
Doc 22 Prolog 3
Nov 10, 2022

Problems

Find the last element of a list

?- lastElement(X, [a,b,c,d]).
X = d

Find the K'th element of a list

?- elementAt(X, [a, b, c, d], 3).
X = c

Find the length of a list

Reverse a list

Last

```
my_last(X,[X]).
my_last(X,[_|L]) :- my_last(X,L).
```

K'th element

```
element_at(X,[X|_],1).
element_at(X,[_|L],K) :- K > 1, K1 is K - 1, element_at(X,L,K1).
```

Length

```
lengthOf([_], X) :- X =1.
lengthOf([_|T], X) :-  lengthOf(T, X2), X is X2 +1.
```

Reverse

```
reverseOf(L1,L2) :- reverseOf(L1,L2,[]).
reverseOf([],L2,L2) :- 1.
reverseOf([H|T],L2,Acc) :- reverseOf(T,L2,[H|Acc]).
```

Classifying Terms

?- var(X).
true.

?- var(2).
false.

?- X = Y, Y= 3,var(X).
false.

?- atom(2).
false.

?- atom(z).
true.

?- number(12).
true.

var(X)
is X an uninstantiated variable

atom(X)
is X an atom

number(X)
is X a number

atomic(X)
is X either a number or atom

Call

call(X)

try to satisfy X as a goal

\+

means not or not provable

Problem

mother_child(susan, sally).
mother_child(susan, matt).

father_child(tom, sally).
father_child(tom, erica).
father_child(tom, pete).
father_child(mike, tom).

?- sibling(X,Y).
X = sally,
Y = sally

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).

Backtracking

```
mother_child(susan, sally).
mother_child(susan, matt).

father_child(tom, sally).
father_child(tom, erica).
father_child(tom, pete).
father_child(mike, tom).

sibling(X, Y)  :- parent_child(Z, X), parent_child(Z, Y), \+ X = Y.

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y)
```

```
?- sibling(X,Y).
X = sally, Y = erica ;
X = sally, Y = pete ;
X = erica, Y = sally ;
X = erica, Y = pete ;
X = pete, Y = sally ;
X = pete, Y = erica ;
X = sally, Y = matt ;
X = matt, Y = sally ;
false.
```

Backtracking & Database

Each time a rule/fact matches Prolog keeps track of where in the database the match occurred

Variables

C/C++/Java variables

- Point to a memory location
- Can change the value of a variable
- X = X + 1

Mathematical Variables

- Represent value(s) that make equations true
- X = X + 1 has no solution

Prolog variables are like mathematical variables

Backtracking & Unification

In backtracking Prolog tries out various values for variables

Cut !

Once a cut is reach in a rule

Prolog will not try to re-satisfy any goal between parent goal and cut

Cuts

Reduce the number of paths searched

Reduce bookkeeping needed for backtracking

Common Reasons for using Cut

Tell Prolog that it has found the correct rule

Tell Prolog to fail a goal without trying other solutions (! , fail)

Tell Prolog it has found a correct solution and stop looking for more

What Happens Here?

```
sumTo(1,1).
sumTo(N,Sum) :-
   N1 is N -1,
   sumTo(N1,Sum2),
   Sum is Sum2 + N.
```

```
?- sumTo(5,X).
X = 15 ;
```

```
ERROR: Stack limit (1.0Gb) exceeded
ERROR:   Stack sizes: local: 0.9Gb, global: 77.8Mb, trail: 0K
ERROR:   Stack depth: 10,189,004, last-call: 0%, Choice po
ERROR:   In:
ERROR:     [10,189,004] user:sumTo(-10188989, _2040432
ERROR:     [10,189,003] user:sumTo(-10188988, _2040434
ERROR:     [10,189,002] user:sumTo(-10188987, _2040436
ERROR:     [10,189,001] user:sumTo(-10188986, _2040438
ERROR:     [10,189,000] user:sumTo(-10188985, _2040440
ERROR:
ERROR: Use the --stack_limit=size[KMG] command line opt
ERROR: ?- set_prolog_flag(stack_limit, 2_147_483_648). to
```

Found the correct rule - so stop

```
sumTo(1,1) :- !.
sumTo(N,Sum) :-
    N1 is N -1,
    sumTo(N1,Sum2),
    Sum is Sum2 + N.
```

```
?- sumTo(5,X).
X = 15.
```

Replacing ! with \+

```
sumTo(1,1).
sumTo(N,Sum) :-
   \+(N = 1),
   N1 is N -1,
   sumTo(N1,Sum2),
   Sum is Sum2 + N.
```

\+. mean not or not provable

!, fail

```
average_taxpayer(X) :- foreigner(X), !, fail.

average_taxpayer(X) :-
    spouse(X, Y),
    gross_income(Y, Income),
    Income > 300000,
    !, fail.

average_taxpayer(X) :-
    gross_income(X, Income),
    20000 < Income, Income < 200000.
```

Replace !, fail with \+

```
average_taxpayer(X) :-
    \+ foreigner(X),
    \+(spouse(X, Y), gross_income(Y, SpouseIncome), SpouseIncome > 300000),
    gross_income(X, Income),
    20000 < Income, Income < 200000.

gross_income(X,Y) :-
    \+ (receives_pension(X, Pension), Pension < 20000),
    gross_salary(X, Z),
    investment_income(X,W),
    Y is Z + W.
```

Found a Correct Solution - So Stop

```
is_integer(0).
is_integer(X) :- is_integer(Y), X is Y + 1.


divide(Numerator,Denominator, Result) :-
    is_integer(Result),
    Product is Result * Denominator,
    ProductNext is (Result + 1) * Denominator,
    Product =< Numerator, ProductNext > Numerator,
    !.
```

Cut !

```
mother_child(susan, sally).
mother_child(susan, matt).

father_child(tom, sally).
father_child(tom, erica).
father_child(tom, pete).
father_child(mike, tom).
```

```
?- sibling(X,Y).
X = sally,
Y = erica ;
X = sally,
Y = pete ;
false.
```

```
sibling(X, Y) :- parent_child(Z, X), !, parent_child(Z, Y), \+ X = Y.

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

What Happens Here?

```
mother_child(susan, sally).
mother_child(susan, matt).

father_child(tom, sally).
father_child(tom, erica).
father_child(tom, pete).
father_child(mike, tom).

related(X,Y) :- father_child(_, X), sibling(X,Y).

sibling(X, Y) :- parent_child(Z, X), !, parent_child(Z, Y), \+ X = Y.

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```

```
?- related(X,Y).

   X = sally,
   Y = erica ;
   X = sally,
   Y = pete ;
   X = erica,
   Y = sally ;
   X = erica,
   Y = pete ;
   X = pete,
   Y = sally ;
   X = pete,
   Y = erica ;
```

More Problems With Cut

```
number_of_parents(adam, 0) :- !.
number_of_parents(eve, 0) :- !.
number_of_parents(vishnu, 0) :- !.
number_of_parents(brahma, 1) :- !.
number_of_parents(X, 2).
```

```
?- number_of_parents(eve,X).
X = 0.

?- number_of_parents(roger,X).
X = 2.

?- number_of_parents(eve,2).
true.

?-
```

Improvement

```
number_of_parents(adam, N) :- !, N = 0.
number_of_parents(eve, N) :- !, N = 0.
number_of_parents(vishnu, N) :- !, N = 0.
number_of_parents(brahma, N) :- !, N = 1.
number_of_parents(X, 2).
```

```
?- number_of_parents(eve,2).
false.

?- number_of_parents(X,Y).
X = adam,
Y = 0.
```

The Lesson

A cut may work with one form of a goal

number_of_parents(eve,X)

and not work with another form

number_of_parents(eve,2)

Depending on the use of the goal this may or may not matter

->/2 (if then), ->/3 (if then else)

```
if_example(X,Y) :- X > 2 -> Y is 0; Y is 10.
```

```
?- if_example(1,Y).
Y = 10.

?- if_example(5,Y).
Y = 0.
```

maplist

maplist(:Pred, +List)

Apply Pred to elements of List

Return false when Pred fails

Return true if Pred is true for all elements

?- maplist(number,[1,2,3]).
true.

?- maplist(number,[1,g,3]).
false.

findall

```
mother_child(susan, sally).
mother_child(susan, matt).

father_child(tom, sally).
father_child(tom, erica).
father_child(tom, pete).
father_child(mike, tom).
```

```
?- findall(X,sibling(X,sally),Y).
Y = [erica, pete, matt].
```

```
related(X,Y) :- father_child(_, X),sibling(X,Y).

sibling(X, Y) :- parent_child(Z, X), parent_child(Z, Y), \+ X = Y.

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).
```