CS 520 Advanced Programming Languages
Fall Semester, 2009
Doc 3 Prolog Cut
Sept 9, 2009

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 occured

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 !

```
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).
```

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

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).
```

```
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).
```

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 ;
```

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.
```

!, 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.
```

```
gross_income(X,Y) :-
    receives_pension(X, P),
    P < 20000,
    !, fail.

gross_income(X,Y) :-
    gross_salary(X, Z),
    investment_income(X,W),
    Y is Z + W.
```

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,
        !.
```

Problems with Cut

```
append_lists([],X, X).
append_lists([A|B], C, [A|D]) :-
     append_lists(B,C,D).

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

?- append_lists([a,b,c],X,Y).
Y = [a, b, c|X].

?- append_lists(X,Y,[a,b,c]).
X = [],
Y = [a, b, c] ;
X = [a],
Y = [b, c] ;
X = [a, b],
Y = [c] ;
X = [a, b, c],
Y = [] ;
false.

?-
```

```
append_cut([],X, X) :- !.
append_cut([A|B], C, [A|D]) :-
     append_cut(B,C,D).

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

?- append_cut([a,b,c],X,Y).
Y = [a, b, c|X].

?- append_cut(X,Y,[a,b,c]).
X = [],
Y = [a, b, c].

?-
```

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

Naming Convention

is_integer

verses

isInteger

Some Built-in Predicates

Success & Failure

true

false

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

?- call(X is 1 + 2).
X = 3.

?- call(append([1,2],[3,4],X)).
X = [1, 2, 3, 4].

\+

```
\+ P :- call(P), !, fail.
\+ P.
```

Problems From Last Time

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).
```


Solution from https://prof.ti.bfh.ch/hew1/informatik3/prolog/p-99/

K'th element

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


Solution from https://prof.ti.bfh.ch/hew1/informatik3/prolog/p-99/

Length

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


Solution from https://prof.ti.bfh.ch/hew1/informatik3/prolog/p-99/

Reverse

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


Solution from https://prof.ti.bfh.ch/hew1/informatik3/prolog/p-99/