

COMP210: Artificial Intelligence

Lecture 15. Prolog: Summary and programming features

Trevor Bench-Capon
Room 215, Ashton Building

<http://www.csc.liv.ac.uk/~tbc/COMP210>

Overview

- Recap: Prolog as a declarative language
- New: input/output, built-in predicates
- Revision: Example questions from past class tests

Prolog as a Declarative Language

- You say what to do, not how to do it
 - Contrast imperative languages
- Variables are different from Java
 - Like 'holes' which are filled by the system for you: not persistent
 - Anonymous variables: Compare
 - ?- parent(P,C). with
 - ?- parent(P,_).

Structure of Programs

- Programs consist of procedures
- Procedures consist of clauses
- A clause is either a fact or a rule

Search Engine

- Built-in search engine
 - Depth first search
 - Back tracking
 - Binds variables as it goes: frees them when backtracking
- Recursive rules
- Some control (cuts to inhibit backtracking)

Recursive rules

- Base (trivial) cases – where we expect the recursion to stop
- General cases


```
listlength([], 0).
listlength([_|T], L) :-
    listlength(T, L1),
    L is L1+1.
```

Datatypes

- Structures
 - `parent(ian,pete)`
 - `location(depot1, manchester)`
- Lists
 - Typically processed with recursive rules

Input/Output

- `write/1` writes out a prolog term
`write(trevor)`
- `read/1` reads in a prolog term
`read(Name)`
Always evaluate to true, so no effect on the logic
- From Bratko's book:


```
cube :- read(X), process(X).
process(stop) :- !.
process(N) :-
    C is N*N*N, write(C), cube.
```

Writing a List

```
writelist([]).
writelist([X|L]) :-
    write(X), nl,
    writelist(L).
```

Built-in Predicates: Lists

- Most of the list manipulation predicates are built into SWI Prolog
 - `append(?List1, ?List2, ?List3)`
 - `member(?Elem, ?List)`
 - `length(?List, ?Int)`
- No need to define them for each new program: can just assume they are there

Built-in Predicates: Sorts

- `nonvar(+Term)` Succeeds if Term is not a variable, or Term is already instantiated.

```
?- member(X,[_a]).
X = _G157;
X = a
?- member(X,[_a]), nonvar(X).
X = a
```

More Sorts

- `atom(+Term)` Succeeds if Term is bound to an atom.
- `integer(+Term)` Succeeds if Term is bound to an integer.
- `float(+Term)`
- `atomic(+Term)` Succeeds if Term is bound to an atom, string, integer or floating point number.
- `compound(+Term)` Succeeds if Term is bound to a compound term (a structure).

First Class Test

- Will be held on Friday March 12th in the 10am Lecture Slot
- Will cover Prolog
- Open book
- Will not include I/O and sorts
- Counts 10% towards overall mark for the module

Questions from 2006 test

```
parent(pete,ian).    % Pete is a parent of Ian
parent(ian,peter).
parent(ian,lucy).
parent(iou,pete).
parent(iou,pauline).
parent(cathy,ian).
```

```
female(cathy).     % Cathy is female.
female(lucy).
female(pauline).
female(iou).
```

```
male(ian).         % Ian is male.
male(pete).
male(peter).
```

- Formulate a query to find a person who is a parent of *both* pete and pauline.

```
?- parent(X,pete), parent(X,pauline).
```

Questions from 2006 test

```
parent(pete,ian).    % Pete is a parent of Ian
parent(ian,peter).
parent(ian,lucy).
parent(iou,pete).
parent(iou,pauline).
parent(cathy,ian).
```

```
female(cathy).     % Cathy is female.
female(lucy).
female(pauline).
female(iou).
```

```
male(ian).         % Ian is male.
male(pete).
male(peter).
```

- Define the relation grandchild using the parent relation.

```
grandchild(X,Z) :- parent(Z,Y), parent(Y,X).
```

Questions from 2006 test

```
parent(pete,ian).    % Pete is a parent of Ian
parent(ian,peter).
parent(ian,lucy).
parent(iou,pete).
parent(iou,pauline).
parent(cathy,ian).
```

```
female(cathy).     % Cathy is female.
female(lucy).
female(pauline).
female(iou).
```

```
male(ian).         % Ian is male.
male(pete).
male(peter).
```

- Formulate a query to find all people who are mothers.

```
?- parent(X,_), female(X).
```

Questions from 2006 test

```
parent(pete,ian).    % Pete is a parent of Ian
parent(ian,peter).
parent(ian,lucy).
parent(iou,pete).
parent(iou,pauline).
parent(cathy,ian).
```

```
female(cathy).     % Cathy is female.
female(lucy).
female(pauline).
female(iou).
```

```
male(ian).         % Ian is male.
male(pete).
male(peter).
```

- What is the first solution found for the following query?

```
?- parent(X, Y), male(Y).
```

```
X = pete
Y = ian
```

Questions from 2006 test

- Define the relation relatives(X, Y) which holds true if the two of its arguments are blood relatives, that is, one of its arguments is a predecessor of the other or they have a common predecessor. For example, both the queries `?- relatives(pete,peter).` and `?- relatives(peter,lucy).` should return yes.

```
relatives(X,Y) :- predecessor(X,Y).
relatives(X,Y) :- predecessor(Y,X).
relatives(X,Y) :- predecessor(Z,X),
                    predecessor(Z,Y).
```

```
predecessor(X,Z) :- parent(X,Z).
predecessor(X,Z) :- parent(X,Y),
                    predecessor(Y,Z).
```

Questions from 2006 test

```
big(bear).
big(elephant).
big(lion).
small(cat).
small(dolphin).
small(monkey).

brown(bear).
black(cat).
gray(elephant).
blue(dolphin).
brown(monkey).
orange(lion).

dark(Z) :- black(Z).
dark(Z) :- brown(Z).

friendof(monkey, cat).
friendof(cat, dolphin).
friendof(monkey, lion).
friendof(bear, elephant).
friendof(lion, bear).
friendof(elephant, bear).
friendof(elephant, monkey).
```

- What is the answer to the query `?- dark(bear) .?`

Yes

Questions from 2006 test

```
big(bear).
big(elephant).
big(lion).
small(cat).
small(dolphin).
small(monkey).

brown(bear).
black(cat).
gray(elephant).
blue(dolphin).
brown(monkey).
orange(lion).

dark(Z) :- black(Z).
dark(Z) :- brown(Z).

friendof(monkey, cat).
friendof(cat, dolphin).
friendof(monkey, lion).
friendof(bear, elephant).
friendof(lion, bear).
friendof(elephant, bear).
friendof(elephant, monkey).
```

- What is the first solution found for the following query?
`?- big(X), dark(X) .`

X = bear

Questions from 2006 test

```
big(bear).
big(elephant).
big(lion).
small(cat).
small(dolphin).
small(monkey).

brown(bear).
black(cat).
gray(elephant).
blue(dolphin).
brown(monkey).
orange(lion).

dark(Z) :- black(Z).
dark(Z) :- brown(Z).

friendof(monkey, cat).
friendof(cat, dolphin).
friendof(monkey, lion).
friendof(bear, elephant).
friendof(lion, bear).
friendof(elephant, bear).
friendof(elephant, monkey).
```

- The relation `friendof(X, Y)` stands for 'X is a friend of Y'. Formulate a query to find all brown friends of elephant.

`?- friendof(X, elephant), brown(X) .`

Questions from 2006 test

```
big(bear).
big(elephant).
big(lion).
small(cat).
small(dolphin).
small(monkey).

brown(bear).
black(cat).
gray(elephant).
blue(dolphin).
brown(monkey).
orange(lion).

dark(Z) :- black(Z).
dark(Z) :- brown(Z).

friendof(monkey, cat).
friendof(cat, dolphin).
friendof(monkey, lion).
friendof(bear, elephant).
friendof(lion, bear).
friendof(elephant, bear).
friendof(elephant, monkey).
```

- Define the relation `friends(X, Y)` which holds true if X is a friend of Y and Y is a friend of X.

`friends(X, Y) :- friendof(X, Y), friendof(Y, X) .`

Questions from 2006 test

```
big(bear).
big(elephant).
big(lion).
small(cat).
small(dolphin).
small(monkey).

brown(bear).
black(cat).
gray(elephant).
blue(dolphin).
brown(monkey).
orange(lion).

dark(Z) :- black(Z).
dark(Z) :- brown(Z).

friendof(monkey, cat).
friendof(cat, dolphin).
friendof(monkey, lion).
friendof(bear, elephant).
friendof(lion, bear).
friendof(elephant, bear).
friendof(elephant, monkey).
```

- Define the relation `havemutualfriend(X, Y)` which holds true if there exists a Z such that Z is a friend of both X and Y.

`havemutualfriend(X, Y) :- friendof(Z, X), friendof(Z, Y) .`

Questions from 2006 test

`?- havemutualfriend(X, Y) .`

X = cat
Y = cat ;

X = cat
Y = lion ;

X = dolphin
Y = dolphin ;

X = lion
Y = cat ;

X = lion
Y = lion ;

X = elephant
Y = elephant

Questions from 2006 test

```
different(X,X):-!,fail.
different(X,Y).

havemutualfriend(X,Y) :- friendof(Z,X),
                           friendof(Z,Y),
                           different(X,Y).

X = cat
Y = lion ;

X = lion
Y = cat ;

X = bear
Y = monkey ;

X = monkey
Y = bear ;
```

Questions from 2006 test

- Consider the following append predicate considered in the course:

```
append([], L, L).
append([H|T], L, [H|L2]) :- append(T, L, L2).
```

What is the result of execution of the following query?

```
?- append([1,3], X, [1, 3, 4, 5, 7]).
```

X = [4, 5, 7]

Questions from 2006 test

- Define the predicate `maxlist(List, Max)` so that `Max` is the greatest number in the list of numbers `List`.

```
maxlist([X], X).
maxlist([H|T], X) :- maxlist(T,X1),
                    max(H,X1,X).

max(X,Y,X) :- X>Y.
max(X,Y,Y) :- Y>=X.
```

Questions from 2006 test

- Define the predicate `ordered(List)` which is true if numbers in the list `List` are ordered. For example, `ordered([1,2,6,19])`.

```
ordered([]).
ordered([X]).
ordered([H1|[H2|T]]) :- H1 < H2, ordered([H2|T]).
```

Questions from 2006 test

- Consider the following program.

```
op([X|T], X, T).
op([X|T], Y, [X|T2]) :- op(T, Y, T2).
```

What is the first solution found for the following query?

```
?- op([1,3,6,3], 3, X).
```

X = [1, 6, 3] ;