A little bit of Prolog

Objects and relations between objects
Facts and rules. **Upper case are variables.**

```
parent(pam, bob).
parent(tom, liz).
parent(bob, pat).

? parent(bob, pat).
? parent(bob, liz).
? parent(bob, ben).
? parent(bob, X).
? parent(X, Y).
```

We use parent(x,y) to mean x is a parent of y.
But to the computer it is just a relation, which could mean:

- y is a parent of x
- x and y are parents of children
- “x y” wrote a book on parenting
- x and y are related in some way
**Prolog**

grandparent (X,Y) :- parent(X, Z), parent(Z, Y).

For all X and Y Upper case means a variable  
X is the grandparent of Y if  
X is a parent of Z and  
Z is a parent of Y

sister (X,Y) :- parent(Z, X), parent(Z, Y), female(X).
For all X and Y  
X is the sister of Y if  
Z is the parent of both X and Y and  
X is a female

Try writing cousin
Try writing ancestor

**Horn Clauses**

\[ u :: p, q, ..., t. \]

can be written as \( (p \land q \land ... \land t) \rightarrow u \)

**Written as a Horn clause:** \( \neg p \lor \neg q \lor ... \lor \neg t \lor u \)

Either \( u \) is true or one of the
Clause

A clause is either
- a single predicate called a fact
- a rule of the form
  \[ \text{condition} \Rightarrow \text{conclusion} \]

Conclusion is a single predicate
Condition is a conjunction of predicates

\[ a_1 \land a_2 \land a_3 \ldots \land a_n \]

Note: prolog write it backwards

\[ \text{grandparent}(X,Y) :\!- \text{parent}(X, Z), \text{parent}(Z, Y). \]
\[ \text{parent}(X,Z) \land \text{parent}(Z,Y) \Rightarrow \text{grandparent}(X,Y) \]

Horn Clauses

Horn clauses don’t have any quantifiers

Assumes that the variables in the head are universally quantified and the variables in the body are existentially quantified (if they don’t appear in the head)
Horn Clauses

\[ \text{grandparent}(X, Y) \iff \text{parent}(X, Z) \land \text{parent}(Z, Y) \]

\[ \text{grandparent}(X, Y) \leftarrow \text{parent}(X, Z), \text{parent}(Z, Y) \]

Equivalent to
\[ \forall x : \forall y : \exists z \text{ parent}(X, Z) \land \text{parent}(Z, Y) \Rightarrow \text{grandparent}(X, Y) \]

Trace

It forces Prolog to evaluate queries one step at a time, indicating what it is doing at each step.
Debugging tool – but helps in understanding
Example

my_last(X,[X]).
my_last(X,[_|L]) :- my_last(X,L).
trace.
my_last(X,[1,2,3,4,5]).

Example

What will happen?

f(a).
f(b).
g(a).
g(b).
h(b).
k(X) :- f(X),g(X),h(X).
2 ?- trace.
Example

f(a).
f(b).
g(a).
g(b).
h(b).

k(X) :- f(X),g(X),h(X).

Some more prolog

Recursive rules

example predecessor relation

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

X is a predecessor of Z if X is a parent of Z

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

And so on...

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

1. How do we get an “or” in Prolog?
2. When a variable is instantiated to a value (temporarily bound), we call that ____________?
3. Describe (using technical terms) the procedure Prolog uses to answer queries.

A prolog program comprises clauses - facts, rules and queries

PROGRAM
big(bear). %fact1
big(elephant). %fact 2
small(cat). %fact 3
brown(bear). % fact 4
black(cat). % fact 5
grey(elephant). %c fact 6
dark(Z) :- black(Z). %rule7
dark(Z) :- brown(Z). %rule 8
?dark(X), big(X). Query (from the prompt)
Data Objects
Atoms versus numbers versus Variables

Atoms
- strings of letters, digits, and the underscore character beginning with a lower case letter
- some strings of special characters
- can enclose strings of characters in single quotes e.g. ‘Fred’

Numbers
- integers and floats

Data Objects
Atoms versus Variables

Variables are strings of characters, digits and underscores that begin with an uppercase letter or an underscore
Can use the anonymous underscore

hasachild(X) :- parent(X,Y)

hasachild(X) :- parent(X,_) \textbf{Wants singletons to be } _

? parent(X, _)
Data Objects

Structured objects

objects that have several components

location(X, Y, Orientation).
location(156, 786, 25).
location(156, 786, Orientation).

location is the functor (function object – maintains state between calls in backtracking)

---

**Diagram:**

```
data objects
  ▼
  |   ▼
simple objects  structures
    ▼    ▼
  constants  variables
    ▼    ▼
atoms  numbers
```

These are all terms
Operations on lists

Membership

The member relation `member(X,L)`

? `member(d, [a, b, h, d]).`  
?  
? `member(d, [a,b,[d h], e]).`  
?  
? `member([d, h], [a, [d,h], e f]).`  

Try writing `member`:

Membership

X is a member of L if

– X is the head of L, or
– X is a member of the tail of L.

\[
\text{member}(X, [X|\text{Tail}]). \\
\text{member}(X, [\text{Head } | \text{Tail}]) :- \text{member}(X, \text{Tail}).
\]

Note two separate clauses
More Involved Membership

X is a member of L if
- X is the head of L, or
- X is a member of the tail of L, or
- X is a member of a sublist of L

member(X,[X|Tail]).
member(X,_[|Tail]) :- member(X,Tail).
member(X,[L|Tail]) :- member(X,L).

At seats

Write the append function

append([2,3,4], [1,4], [2,3,4,1,4]).
More bits and pieces

Comparison operators (testing, just succeeds or fails, unlike unification where changes occur to instantiation)

- \( X > Y \)
- \( X < Y \)
- \( X \geq Y \)
- \( X \leq Y \) % unexpected order
- \( X =:= Y \) True if equal \( 1 + 2 =:= 2 + 1 \)
- \( X =\neq Y \) True if values are not equal

Unification

- \( X = Y \)
- \( X \neq Y \)

Not identical

\( X\neq Y \)

Append

Notice there is no concept of input/output

\[
\begin{align*}
\text{append}([],X,X). \\
\text{append}([X|Y],R,[X|T]) & :- \text{append}(Y,R,T).
\end{align*}
\]

\[
\begin{align*}
\text{append2}([],L,L). \\
\text{append2}([X|Xs],Y,R) & :- \\
& \text{append2}(Xs,Y,R1), \\
& R = [X|R1]
\end{align*}
\]

?append([1,2,3], Y, [1,2,3,4,5]).

\( Y = [4, 5] \)
Cut!

Automatic backtracking is one of the most characteristic features of Prolog. But backtracking can lead to inefficiency.

If Prolog finds a cut in a rule, it will not backtrack on the choices it has made. If it has chosen q for the variable X and encounters a cut, Prolog will consider q the only option for X, even if there are other possibilities in the database.
merge([], List, List).
merge(List, [], List).
merge([H1|T1], [H2|T2], [H1|T3]) :-
  H1 =< H2,
  merge(T1, [H2|T2], T3).
merge(List1, [H2|T2], [H2|T3]) :-
  merge(List1, T2, T3).

Why does this happen
merge([1,3,4,5,7], [2,6],X).
X = [1, 2, 3, 4, 5, 6, 7] ;
X = [1, 2, 3, 4, 6, 5, 7] ;
X = [1, 2, 3, 6, 4, 5, 7] ;
X = [1, 2, 6, 3, 4, 5, 7] ;
X = [2, 1, 3, 4, 5, 6, 7] ;
X = [2, 1, 3, 4, 6, 5, 7] ;
X = [2, 1, 6, 3, 4, 5, 7] ;
X = [2, 1, 6, 3, 4, 5, 7] ;
Better as

\[
\text{merge([], List, List) :- !.}
\]
\[
\text{merge(List, [], List):- !.}
\]
\[
\text{merge([H1|T1], [H2|T2], [H1|T3]) :-}
\]
\[
\text{\quad H1 =< H2, !,}
\]
\[
\text{\quad merge(T1, [H2|T2], T3).}
\]
\[
\text{merge(List1, [H2|T2], [H2|T3]) :-}
\]
\[
\text{\quad merge(List1, T2, T3).}
\]

Structures

\[
\text{date(X,Y,Z).}
\]
\[
\text{date(20, june, 2005).}
\]
\[
\text{date(Day, june, 2005).}
\]
\[
\text{date(Day, Month, Year) :- Day > 0, Day =< 31, \ldots.}
\]
\[
\text{classTime(cs4700, 1200,th).}
\]
\[
\text{classTime(cs6100, 0900,th).}
\]
\[
\text{classTime(cs1410,0930,mwf)}
\]

Examples

Or, we could have nested structures.

class(cs4700, time(th, 12:00, 13:15), instructor(allan, vicki), location(bus, 345)).
class(cs1400, time(mwf, 8:30, 9:20), instructor(duhadway, linda), location(main, 155)).
class(cs1400, time(mwf, 10:30, 11:20), instructor(mathias, dean), location(main, 115)).
class(cs1400, time(mwf, 11:30, 12:20), instructor(duhadway, linda), location(main, 115)).
class(cs1400, time(mwf, 13:30, 13:20), instructor(duhadway, linda), location(main, 155)).
class(cs1400, time(mwf, 10:30, 11:20), instructor(duhadway, linda), location(main, 115)).
class(cs1400, time(mwf, 13:30, 13:20), instructor(duhadway, linda), location(main, 155)).
class(cs2420, time(mwf, 10:30, 11:20), instructor(john, mathias), location(main, 225)).
class(cs2420, time(th, 9:30, 11:20), instructor(john, mathias), location(geol, 105)).

class(C,T,I,L) – accesses the four pieces of the structure.

How do we find when cs4700 starts?

How do we find when cs4700 starts? Called a selector

start_time(ClassID, T) :-
class(ClassID, time(_, T, _), _).

31

32
Examples

One way of generating a list is by asking Prolog to find all the solutions to a goal using the built-in findall/3. To create a list of all the times in the class database (instead of asking for them one at a time), we could do this:

\[ \text{findall}(T, \text{start\_time}(C, T), \text{Times}). \]

The query
\[ \text{findall}(\text{Object}, \text{Goal}, \text{List}). \]
produces a list List of all the objects Object that satisfy the goal Goal.

Unification

Matching clauses with variables
Have to find the appropriate substitutions for variables so that the clauses match
Process is called unification
  – Process by which variables are instantiated
Unification in Prolog...

A variable that is uninstantiated unifies with anything and becomes instantiated with “anything” = causes two things to be unified

? me = X.
X = me

? f(a, X) = f(Y, b).
X = b
Y = a

?f(a, X) = f(x, Y)
No

?f(a, Y) = f(a, B).
Y = _G279
B = _G279

Unification in Prolog

A structured term (predicate or function applied to arguments requires
– Same predicate/function name
– Same number of arguments
– Arguments can be unified recursively

?f(X) = g(X) - can’t match
?f(X) = f(a, b) - can’t match
?f(a, g(X)) = f(Y, b) - can’t match
?f(a, g(X)) = f(Y, g(b)) = Y = a, X = b
Unification examples

Unify the following:

\[ p(X,Y) = p(a, Z) \]
\[ p(X,X) = p(a,b) \]
\[ \text{ancestor}(X,Y) = \text{ancestor}(\text{bill}, W) \]
\[ p(X,a,Y) = p(Z,Z,b) \]
\[ p(\text{Marcus}, g(X,Y)) = p(X, g(\text{Caesar}, \text{Marcus})) \]
\[ g(X, X) = g(f(X), f(X)) \]

Unification examples (answers)

Unify the following:

\[ p(X,Y) = p(a, Z) \quad \text{X=a, Y and Z are the same} \]
\[ p(X,X) = p(a,b) \quad \text{Can’t unify} \]
\[ \text{ancestor}(X,Y) = \text{ancestor}(\text{bill}, W) \quad \text{X=bill, Y/W the same} \]
\[ p(X,a,Y) = p(Z,Z,b) \quad \text{X and Z are the same, Z=a, so X=a, Y=b} \]
\[ p(\text{Marcus}, g(X,Y)) = p(X, g(\text{Caesar}, \text{Marcus})) \quad \text{Marcus and X are the same, X = Caesar, so Marcus=Caesar= Y} \]
\[ g(X, X) = g(f(X), f(X)) \quad \text{X=f(X) a problem as is recursive} \]
Adding and deleting

How do you add an item to a list?

Deleting an item from a list – del(X, L, Result)

1. If X is the head of L – result is the tail of L
2. If X is contained in the tail of L then concatenate the head of L onto the del of X from the tail of L
3. If you try to delete X from an empty list, return an empty list.

Try to write the delete function at your seats

del(4,[2,3,5,4,3,6], [2,3,5,3,6]).

Deleting...

del(X,[],[]). % unless we wanted it to fail if no X
del(X, [X|Tail], Tail)).

del(X, [Y|Tail], [Y|Tail2]) :- del(X, Tail, Tail2)

Deletes one occurrence from the list

What happens when:

del(a, [a, b, a, a], X).

What if we changed it to

delall(X,[],[]).
delall(X,[X|Tail],Tail2) :- delall(X,Tail, Tail2).
delall(X,[Y|Tail],[Y|Tail2]) :- delall(X,Tail, Tail2),X\=\=Y.

(not equal – important if we attempt to use “;” to get more solutions)
Relations/Terms/Queries

An important note
You can define different relations with the same name but with different numbers of arguments e.g. member/1, member/2, member/3

Wrong number of arguments - you will get an error message
arity: number of arguments
if there is such a relation predefined, you hide the predefined relation.

Remove the K'th element from a list.

Example:
?- remove_at(2,[a,b,c,d],X,R).
X = b
R = [a,c,d]
Remove the K'th element from a list (answer)

Example:
?- remove_at(2,[a,b,c,d],X,R).
X = b
R = [a,c,d]

remove_at(1,[H|T],H,T).
remove_at(N,[H|T],X,[H|R]) :- N1 is N - 1, remove_at(N1,T,X,R).

At seats
Define two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively. For example

?- evenlength([a,b,c,d])
? yes
Can use f ([X,Y|R]) to look at first two items by name.

?- oddlength([c,d,e])
? yes
Answers

\text{evenlength(\,L)} \,:= \n\text{length(\,L,\,Ct)}, \quad 0 \text{ is } \text{Ct \ mod \ 2.} \n\text{or} \n\text{evenlength(\,[]).} \n\text{evenlength(\,[\,X,Y|R\,]) \,:= \text{evenlength(\,R).} }

---

Reading Code: What do the following do?

\text{quest1} (\,[] \,\rightarrow\, 0). \n\text{quest1}(\,[]|T, \,L) \,:= \n\text{quest1}(T, \,L1), \n\text{L \ is \ } \text{L1 + 1.} \n\text{quest2(1,[F|\_], F).} \n\text{quest2(N,[F|R],Which) \,:=} \n\text{N2 \ is \ } \text{N-1,} \n\text{quest2(N2,R,Which).}
Reading Code

\[\text{quest3}([], \text{[]}).\]
\[\text{quest3}([\_]).\]
\[\text{quest3}([H1,H2|\text{Rest}]) :-
  H1 =\leq H2,
  \text{quest3}([H2|\text{Rest}]).\]
\[\text{quest4}(\text{Item}, \text{[]}, [\text{Item}]) :- !.\]
\[\text{quest4}(\text{Item}, [\text{H}|\text{T}], [\text{Item}, \text{H}|\text{T}]) :-
  \text{Item} =\leq \text{H}, !.\]
\[\text{quest3}([H1,H2|\text{Rest}]) :-
  \text{H1} =\leq \text{H2},
  \text{quest3}([\text{H2}|\text{Rest}]).\]

Sieve of Eratosthenes

\[\text{primes}(\text{Limit}, \text{Ps}) :- \text{integers}(2, \text{Limit}, \text{Is}), \text{sift}(\text{Is}, \text{Ps}).\]
\[\text{integers}(\text{Low}, \text{High}, [\text{Low}|\text{Rest}]) :-
  \text{Low} =\leq \text{High}, !, \text{M} \text{is} \text{Low}+1, \text{integers(M, High, Rest}).\]
\[\text{integers}(_, _, []).\]
\[\text{sift}(\text{[]}, \text{[]}).\]
\[\text{sift}([\text{I}|\text{Is}], [\text{I}|\text{Ps}]) :- \text{remove}(\text{I}, \text{Is}, \text{New}), \text{sift}(\text{New, Ps}).\]
\[\text{remove}(\text{P}, [], []).\]
\[\text{remove}(\text{P}, [\text{I}|\text{Is}], \text{Nis}) :- 0 \text{is} \text{I mod P}, !, \text{remove}(\text{P, Is, Nis}).\]
\[\text{remove}(\text{P}, [\text{I}|\text{Is}], [\text{I}|\text{Nis}]) :-
  \text{remove}(\text{P, Is, Nis}).\]
Translation

translate([], []).  
translate([H|T], [X|Y]) :- change(H, X), translate(T, Y).

change(you, i).  
change(are, [am, not]).  
change(french, german).  
change(do, did).  
change(X, X).

What is the output
?- alter([you, are, a, french, computer], F).

Ask ?- translate([you, are, a, french, computer], F).
F = [i, [am, not], a, german, computer];
F = [i, [am, not], a, french, computer];
F = [i, are, a, german, computer];
F = [i, are, a, french, computer];
F = [you, [am, not], a, german, computer];
F = [you, [am, not], a, french, computer];
F = [you, are, a, german, computer];
F = [you, are, a, french, computer].
Try this problem 1

Write the code to give a specific fibonacci number:

(1,1,2,3,5,8,13…)
fib(6,X).
x = 8

Try these Problems

Write the code to give a specific fibonacci number:

(1,1,2,3,5,8,13…)
fib(6,X).
x = 8

fib(X, Y):-
X > 1, X1 is X - 1, X2 is X - 2,
fib(X1, Z), fib(X2, W), Y is W + Z.
Try this problem 2

Write the code to reverse a list

reverse([3,4,5],X).
X = [5,4,3]

reverse([X], [X]).
reverse([H|T], R):-
    reverse(T, X), append(X, [H], R).
Try this problem 3

Write the code to compute factorial:

factorial(5,X).
X=120

factorial(0,1).
factorial(N,F) :- N1 is N-1,
    factorial(N1,F1), F is N*F1.
Try this problem 4
Write the code to remove multiple copies from a list.

make_set([1,3,5,2,2,4,1,2,6,3], X).
X=[5,4,1,2,6,3]
Try this problem 5

Write the code to perform set difference: remove any elements of the second list from the first list.

difference([1,3,7,4], [5,3,4,2], X).
X=[1,7]

difference([],Y,[]).
difference([X|R],Y,Z) :-
    member(X,Y),! , difference(R,Y,Z).
difference([X|R],Y,[X|Z]) :- difference(R,Y,Z).

Try this problem 6

Write the code to union two lists, but not include duplicates from the two lists.

\[
\text{union}([1,3,5,6], [1,4,6,8], X).
\]
\[
X = [1,3,5,6,4,8]
\]