# Lecture 5

G. Dudek
Topics in AI
McGill University

## Today's lecture

- Administrative issues
  - Comments on assignment
  - PDF files
  - Class notes
- Knowledge representation: wrap-up
  - Prolog details
  - Non-monotonic logic
  - Forward and backward chaining
- Introduction to search

## Don't care

- Symbol _ (underscore) is used to match a predicate that we don't plan to use on the right-hand side.
- It's like a dummy variable.

Eg. **likes(a,b).**

Would return **true** no matter what a & b are.

We can use **likes(a,_).**

…..or **likes(_,_).**

## Prolog (continued)

- Supports **<u>lists</u>** of items

  [] - empty list
  [1,2,3] - 3 items
  [bob, ted, alice] - three objects
  [[a], [1,2,3], []] - a list of lists

To examine a sub-part

[ H | L ]

refers to a list decomposed into a

head:H (the first element)

and a remaining part

tail:T

## Lists: [H|T]

[1,2,3]

- head is **1**

[bob, ted, alice]

- head is **bob**

[ [a], [1,2,3], [ ] ]

- head is **[a]**

[ [ [1,2,3]], [1,2,3], [ ] ]

- head is [ **[1,2,3] ]**

[ ]

- cannot match

## Testing membership

- Now we can easily define a predicate to test for list membership.
- Step 1: the head

**member(H,[H|L]).**

- First argument is an item.
- Second argument is a list.
  - This matches if H is the head of the list.

member(bob,[bob,alice]) unifies with member(H,[H|L]) is we let bob match H and [bob,alice] match [H|L].

**member( H, [H | _ ] ).**

## Membership: the body.

- Step 2: if it's not the head, then there must be a sublist for which it is the head.
  - Recursive definition
- See if the item is the head of the tail portion.

```
member(Item,[Head|Tail]) :-
  member(Item,Tail).
```

## Membership: complete

```
member( Item, [ Item | _ ] ).
member(Item, [ _ | Rest ] ) :-
   member(Item, Rest).
```

## Unification: examining combinations

- **Remember: prolog execution proceeds by repeated unifications, applied recursively.**
- **Consider:**
  - **foo(x) :- bar(x).**
  - **bar(x) :- foo(x).**
  - **This will lead to a problem: the unification will never terminate!**

## Recursion fix

- How can we fix the infinite recursion?
  - Never re-examine an already-considered unifier (i.e. solution).

1. Within the definition, save the previous solutions (unifications).
2. Check if the new unifier (solution) is one of those.
   How?

Use a list!

```
foo(X,L) :- …
```

X is the item,
L is a list of prior unifiers

## Improved foo!

```
member…

foo(X,L) :-  not(member(X,L)),
       bar(X,[X|L]).

bar(X,L) :-  not(member(X,L)),
       foo(X,[X|L]).

foo(a).
```

Improved foo!
foo(y,[]).

## Concept Description Language

- A specialized language for efficient inference.
- Represent
  - classes of objects,
  - sub-classes of classes,
  - instances of classes,
  - properties of instances (and classes).
- Akin to inheritance in object-oriented programming.
- A **semantic network** is a graph-based representation that addresses the same idea.

(See DAA pp. 107-109.)