Even More Prolog

Correction: “is”

- Earlier, I stated that the item on the left of an “is” statement *must* be a variable. This is incorrect (although it is *usually* a variable!). It can also be a number. The item on the right must be arithmetically evaluatable. So, you could have:
- 3 is 2 + 1.

ISMEMBER/2

```
% Checks to see if the atom we’re looking for is the first item in the list.
ismember(A, [A|_]).

% Otherwise, checks to see if it is in the tail of the list (recursively).
ismember(A, [_|T]) :-
  ismember(A, T).
```

CHART THIS!

In-class exercise: Let’s recurse in Prolog!

- ismember/2. Checks if its first argument (an atom) is a member of its second item (a list).
- sum_up/2. Takes a list of numbers as its first argument, and returns the sum of the numbers as its second argument.
- myappend/3. The third argument is the first two arguments, appended.
- reverselist/2. Reverses the order of its first argument and returns the reversed list as its second argument. [Hint: might want to create a second predicate with *3* arguments to do all the work. See discussion here: http://www.learnprolognow.org/lpnpage.php?pagetype=html&pageid=lpn-htmlse25]

Note that member/2, append/3, and reverse/2 are built-in functions. Don’t use in assignments.

My strategy

- The examples on the previous slide cover most typical recursion situations. Is one close to what you’re trying to do?
- What’s the base case?
- For the non-base case, sketch in a typical recursion. Don’t worry too much about the variable names, and trust in *THE MAGIC OF RECURSION*.
- Then, step back, and look at the singleton variables. Which ones can you ignore? Replace these with “_”. For any remaining singletons, how do they relate to each other? Are they the same (often happens!)?

Programming declaratively

- Let’s say you want a test for membership: the first argument is an atom, the second is a list, and you want the test to be “true” if the atom is in the list, “false” otherwise. You program it, it works, great.
- What happens if you make the first argument a variable? What about the second? What about both? What if one or both arguments is a predicate? Or a list?
- In a perfect world, all of these variations would work or fail gracefully.
- In the real world, you just need to make sure that the predicate works as expected in your program, *including any necessary backtracking.*

Tests, functions, generators

Please compare:

- **TEST:** ismember(3, [1, 2, 3]).
- **‘FUNCTION’:** ismember(X, [1, 2, 3]).
- **‘GENERATOR’:** ismember(3, X).

Logically, they’re similar queries: does the first argument appear [or unify with something that appears] in the second?

However, how you would use them in a program is pretty different.

BEWARE: When using a predicate as a generator, it’s pretty common for the first match to work, but for failure-driven recursion (e.g. you hit “;”) to cause DFS to dive off into the infinite depths of the search space. What tool might be handy here?

Assignment hints

- What does “listlength(L, 3)” do? Could this be useful?
- Imagine that count/3 has an atom as its first argument, and a list as its second argument. Its third argument is the number of times the atom appears in the list. Obviously, this works well as a test and as a ‘function’ (with the 3rd arg as a variable). Could it work as a generator?
- What tests would you then need to check the constraints given in the assignment?

Bagof/3

- Probably not necessary for this course – but handy!
- Collects the results of DFS/unification in a list.

Example:

- allmembers(L, L2) :-
-   bagof(A, ismember(A, L), L2).