More Prolog

But first… Assignment 4

- Due Nov 12 at midnight, because I was slow posting it
- All about Prolog!
- Don’t use built-in functions, aside from arithmetic/comparison operators.

Remember:

Prolog tries very hard to be declarative. So:

- There are no functions: nothing is *returned.*
- There are no procedures: you can’t explicitly say “do this, then this, then this.”

- All Prolog can *do* is depth-first search (so, the order of the clauses DOES matter) and unification.

Equality in Prolog

See http://www.swi-prolog.org/pldoc/man?section=arith for exact definitions, but roughly:

- A = B. means: Can they be unified, symbolically?
- A == B. means: Are they exactly the same (barring operator syntax shifts), before evaluation?
- A =:= B. means: Do they evaluate to the same thing? (i.e. arithmetic equal)
- A is B. means: As close as you are going to come in Prolog to assignment of a value to a variable. A is not evaluated, B must evaluate, and they must be unifiable.

In-class exercise: Test cases

Write down what you think will happen in each of these cases. Then, test them in Prolog.

```
a(b,c) = a(C,B).
A(b,c) = a(b,c).
a(b(c),C) = a(C,B).
a(b(C),C) = a(C,B).
a(b,c) == a(b,c).
a(B,c) == a(b,c).
a(b,c) =:= a(b,c).
4+5 =:= 10-1.
4+5 is 10-1.
X is 10-1.
a(X,c) is a(10,c).
```

Lists

- [a, b, c(d), 4, 4+5, [], [a, b]]
- [H|R] form is equivalent to CAR and CDR in LISP
  - [a, b, c, d] = [H|R].
  - Unifies to: H = a, R = [b, c, d]
- Can put multiple items in the head (but not the tail).
  - [A, B, C | T] = [1, 2, 3, 4, 5].
  - Unifies to: A = 1, B = 2, C = 3, T = [4,5]
- [H|R]=[a,b,c,d]. also works.
- Can embed in a predicate. Try these:
  - a([b,C], d) = a([D, 3], d).
  - a([b,C], D) = a(D, b).

Operators

- See: http://www.swi-prolog.org/pldoc/man?section=arithpreds

Note that operators are just a syntactical convenience. These are the same:

- X is 3 + 2
- is(X, +(3,2)).

However, it’s important to know the *precedence* of the operator, which defines the order in which multiple operators will be assessed. For example, “*” has **lower** precedence than “+”, so will be assessed **first**.

Error from last class

```
?- +(3,4)==+(3,4).
ERROR: Syntax error: Operator expected
ERROR: +(3,4
ERROR: ** here **
ERROR: )==+(3,4) .
?- +(3,4) == +(3,4).
true.
```

What’s going on here? Hint:

```
?- atom(==+).
true.
```

Format/2

ARG1 is a string including character codes, and ARG2 is a list of arguments. Causes formatted text to be printed to the screen. For example, try these in Prolog:

- X is 1/3, format('We calculate that ~w equals ~3f. ~nAwesome!', ['1/3', X]).
- format('The speed of light is ~~~3E m/s.~n', [299792458]).

See http://www.swi-prolog.org/pldoc/man?section=format for more codes and explanation.

!, fail

- ! is a “cut”. It says Prolog should *not* backtrack past the cut point, in its attempt to prove a clause. Can be very useful in preventing infinite recursion.
- “fail” is simply that – the clause will fail. Backtracking can continue, though. You don’t often need this on its own, because anything that doesn’t succeed, fails (if it terminates).
- ! and fail are a powerful combination – use carefully! E.g. for a negative test (see example).
- Also consider \+ (“not”). Be aware of the difference between \+ and a logical not in practice.

=..

Remember what happened with this?

```
?- A(b, c) = a(b, c).
```

ERROR etc.

This is because a predicate name must be a atom. But what if you want to compare predicates in this way, or manipulate them in other ways? Use "=..". Examples:

```
?- a(b, c) =.. X.

X = [a, b, c].

?- X =.. [a, b, c].

X = a(b, c).
```

When should you use “_”?

- When you want anything to match.
- Prolog has a lot of not very informative error messages – but the “SINGLETON VARIABLE” warning is very useful!
- So, instead of a singleton variable, if you really don’t care, use “_”.

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.

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).
```