- Assignment 5 is due on Thursday
- Midterm #3 next week — more details on web site

"Contrariwise," continued Tweedledee, "if it was so, it might be; and if it were so, it would be; but as it isn't, it ain't. That's logic."

– Lewis Carroll, Through the Looking-Glass

Since the midterm...

Done:

- Syntax and semantics of propositional definite clauses
- Model a simple domain using propositional definite clauses
- Bottom-up proof procedure computes a consequence set using modus ponens.
- Top-down proof procedure answers a query using resolution.
- The box model provides a way to procedurally understand the top-down proof procedure with depth-first search.
- Syntax of Datalog: Predicate symbols, constants, variables, queries.
- Semantics of Datalog: Interpretations, variable assignments, models, logical consequence.
- Functions applied to arguments refer to individuals.
  Individuals are described using clauses.
  (Function symbols are like Haskell constructors.)

Writing a Prolog program

To write a Prolog program:

- Have a clear intended interpretation – what all predicates, functions and constants mean
- Don't tell lies.
  Make sure all clauses are true given your meaning for the constants, functions, predicates.
- Make sure that the clauses cover all of the cases when a predicate is true.
- Avoid cycles.
- Design top-down, build bottom-up.
- Debug all predicates as you write them.
- To solve a complex problem break it into simpler problems.

Function Symbols

- We extend the notion of term. So that a term can be
  - a variable
  - a constant
  - of form $f(t_1, \ldots, t_n)$ where f is a function symbol and the t_i are terms.

Syntactic Sugar for Lists (lists.pl)

- The empty list is []
- The list with first element H and the rest of the list T is $[H \mid T]$.
- $[\cdots a \cdots \mid [\,]]$ written as $[\cdots a \cdots]$.
- $[\cdots a \cdots \mid [\cdots b \cdots]]$ written as $[\cdots a \cdots, \cdots b \cdots]$.

Examples

- $list(L)$ is true if L is a list
- $member(X, L)$ is true if X is an element of list L
- $append(A, B, C)$ is true if C contains the elements of A followed by the elements of B
- $numeq(X, L, N)$ is true if N is the number of instances of X in L.

Lists examples (lists.pl)

- Define $sum(L, S)$ that is true when S is the sum of the elements of list L.
- Define $sum3(L, A, S)$ is true if S is A plus the sum of the elements of L
- Define: $reverse(L, R)$ is true if R has same elements as L in reverse order.
- Define $reverse3(L, A, R)$ is true if R consists of the elements of L reversed followed by the elements of A

- Compare

```
% append(L,A,R) is true if list R contains the
% elements of list L followed by the elements of list A
append([],R,R).
append([H|T],A,[H|R]) :-
    append(T,A,R).

% reverse3(L,A,R) is true if R contains the
% elements of L reversed followed by the elements of A
reverse3([],R,R).
reverse3([H|T],A,R) :-
    reverse3(T,[H|A],R).
```

Clicker Question

```
% append(L,A,R) is true if R contains the
% elements of L followed by the elements of A
append([],L,L).
append([H|T],A,[H|R]) :-
    append(T,A,R).
```

What is the answer to query

```
?- append([a,b,c],X,Y).
```

A There are no proofs

B `Y = [a, b, c|X]`

C `X = [], Y = [a ,b, c]`

D `X = Y = [a, b ,c]`

B `Y = [a, b, c, X]`

Clicker Question

```
% reverse3(L,A,R) is true if list R consists of
%   the elements of list L reversed
%   followed by the elements of list A
reverse3([],R,R).
reverse3([H|T],A,R) :-
    reverse3(T,[H|A],R).
```

What is the answer to query

```
?- reverse3([a,b,c],X,Y).
```

A There are no proofs

B `Y = [c, b, a|X]`

C `Y = [c ,b, a], X=[]`

D `Y = X = [c, b, a]`

E `Y = [c, b, a, X]`

Clicker Question

```
revapp([],R,R).
revapp([H|T],A,[H|R]) :-
    revapp(T,[H|A],R).
```

What is the answer to query

```
?- revapp([a,b,c],X,Y).
```

- A There are no proofs
- B `Y = [c, b, a, c, b, a|X]`
- C `Y = [a, b, c, a, b, c|X]`
- D `Y = [c, b, a, a, b, c|X]`
- E `Y = [a, b, c, c, b, a|X]`

Trees (bstree.pl)

A binary search tree can be used as a representation for dictionaries.

- A binary search tree is either
  - *empty* or
  - $bnode(Key, Val, T0, T1)$ where Key has value Val and T0 is the tree of keys less than Key and T1 is the tree of keys greater than Key
- Define $val(K, V, T)$ is true if key K has value V in tree T
- Define $insert(K, V, T0, T1)$ true if T1 is the result of inserting $K = V$ into tree T0

Trees (bstreec.pl)

- In Prolog, when $X < Y$ is called, both X and Y must be ground (variable free) numbers
- There are constraint solvers that let Prolog act more logically. `X #< Y` specifies the constraint that $X < Y$.
- Eg, consider the query

```
val(K,V,bnode(2,22, bnode(1,57,empty,empty),
                    bnode(5,105,empty,empty))).
```

- $<$ is much faster as it can be evaluated immediately.
- `#<` requires more sophisticated reasoning.

```
?- val(K,V,bnode(2,22, bnode(1,57,empty,empty),
                   bnode(5,105,empty,empty))), V #< 99.
?- V #< 99, val(K,V,bnode(2,22,
                        bnode(1,57,empty,empty),
                        bnode(5,105,empty,empty))).
```