

# For Friday

- Lectures 7-10 from the Prolog site
- Prolog Handout 2

# Exam 1

- Monday
- Take home portion due on Monday

# Working with Prolog

- You interact with the Prolog **listener**.
- Normally, you operate in a querying mode which produces **backward chaining**.
- New facts or rules can be entered into the Prolog database either by **consulting** a file or by switching to **consult** mode and typing them into the listener.

# Prolog and Logic

- First order logic with different syntax
- Horn clauses
- Does have extensions for math and some efficiency.

# The parent Predicate

- Definition of parent/2 (uses facts only)

```
%parent(Parent,Child).
```

```
parent(pam, bob).
```

```
parent(tom, liz).
```

```
parent(bob, ann).
```

```
parent(bob, pat).
```

```
parent(pat, jim).
```

# Constants in Prolog

- Two kinds of constants:
  - Numbers (much like numbers in other languages)
  - Atoms
    - Alphanumeric strings which begin with a lowercase letter
    - Strings of special characters (usually used as operators)
    - Strings of characters enclosed in single quotes

# Variables in Prolog

- Prolog variables begin with capital letters.
- We make queries by using variables:  
?- parent(bob,X).  
X = ann
- Prolog variables are **logic** variables, **not** containers to store values in.
- Variables become **bound** to their values.
- The answers from Prolog queries reflect the bindings.

# Query Resolution

- When given a query, Prolog tries to find a fact or rule which matches the query, binding variables appropriately.
- It starts with the first fact or rule listed for a given predicate and goes through the list in order.
- If no match is found, Prolog returns no.

# Backtracking

- We can get multiple answers to a single Prolog query if multiple items match:  
?- parent(X,Y).
- We do this by typing a semi-colon after the answer.
- This causes Prolog to backtrack, unbinding variables and looking for the next match.
- Backtracking also occurs when Prolog attempts to satisfy rules.

# Rules in Prolog

- Example Prolog Rule:

```
offspring(Child, Parent) :-  
    parent(Parent, Child).
```

- You can read “:-” as “if”
- Variables with the same name must be bound to the same thing.

# Rules in Prolog

- Suppose we have a set of facts for male/1 and female/1 (such as female(ann).).
- We can then define a rule for mother/2 as follows:

```
mother(Mother, Child) :-  
    parent(Mother, Child),  
    female(Mother).
```

- The comma is the Prolog symbol for **and**.
- The semi-colon is the Prolog symbol for **or**.

# Recursive Predicates

- Consider the notion of an ancestor.
- We can define a predicate, `ancestor/2`, using `parent/2` if we make `ancestor/2` recursive.

# Lists in Prolog

- The empty list is represented as [].
- The first item is called the head of the list.
- The rest of the list is called the tail.

# List Notation

- We write a list as:  $[a, b, c, d]$
- We can indicate the tail of a list using a vertical bar:

$$L = [a, b, c, d],$$

$$L = [\text{Head} \mid \text{Tail}],$$

$$L = [H1, H2 \mid T].$$

$$\text{Head} = a, \text{Tail} = [b, c, d],$$

$$H1 = a, H2 = b, T = [c, d]$$

# Some List Predicates

- member/2
- append/3

# Try It

- `reverse(List,ReversedList)`
- `evenlength(List)`
- `oddlength(List)`

# The Anonymous Variable

- Some variables only appear once in a rule
- Have no relationship with anything else
- Can use `_` for each such variable

# Arithmetic in Prolog

- Basic arithmetic operators are provided for by built-in procedures:

$+$ ,  $-$ ,  $*$ ,  $/$ , `mod`, `//`

- Note carefully:

`?- X = 1 + 2.`

`X = 1 + 2`

`?- X is 1 + 2.`

`X = 3`

# Arithmetic Comparison

- Comparison operators:

>

<

>=

=< (note the order: NOT <=)

== (equal values)

!= (not equal values)

# Arithmetic Examples

- Retrieving people born 1950-1960:  
?- born(Name, Year),  
Year >= 1950,  
Year =< 1960.
- Difference between = and ::==  
?- 1 + 2 ::= 2 + 1.  
yes  
?- 1 + 2 = 2 + 1.  
no  
?- 1 + A = B + 2.  
A = 2  
B = 1

# Length of a List

- Definition of length/2

length([], 0).

length([\_ | Tail], N) :-  
 length(Tail, N1),  
 N is 1 + N1.

- Note: all loops must be implemented via recursion

# Counting Loops

- Definition of sum/3  
sum(Begin, End, Sum) :-  
    sum(Begin, End, Begin, Sum).  
sum(X, X, Y, Y).  
sum(Begin, End, Sum1, Sum) :-  
    Begin < End,  
    Next is Begin + 1,  
    Sum2 is Sum1 + Next,  
    sum(Next, End, Sum2, Sum).

# The Cut (!)

- A way to prevent backtracking.
- Used to simplify and to improve efficiency.

# Negation

- Can't say something is NOT true
- Use a **closed world assumption**
- Not simply means “I can't prove that it is true”

# Dynamic Predicates

- A way to write self-modifying code, in essence.
- Typically just storing data using Prolog's built-in predicate database.
- Dynamic predicates must be declared as such.

# Using Dynamic Predicates

- assert and variants
- retract
  - Fails if there is no clause to retract
- retractall
  - Doesn't fail if no clauses