Prolog

- Prolog is a language that is useful for doing symbolic and logic-based computation.
- Name chosen as an abbreviation for "programming in logic"
- It's declarative: very different from imperative style programming like Java, C++, Python, ...
- A program is partly like a database but much more powerful since we can also have general rules to infer new facts!
- A Prolog interpreter can follow these facts/rules and answer queries by resolution and search.

*Slides based on those of Fahiem Bacchus

A simple prolog program

saved in a file named **family.pl**

```
male(albert).                                   % a fact
male(edward).
female(alice).
female(victoria).
parent(albert,edward).
parent(victoria,edward).
parent(albert,alice).
father(X,Y) :- parent(X,Y), male(X).     % a rule
mother(X,Y) :- parent(X,Y), female(X).
sibling(X,Y) :- parent(Z,X), parent(Z,Y), X \= Y.
                        % use "\=" for "not equal"
```

- A fact/ rule (statement) ends with “.”
- Read “:-” as “if” , read “,” as “and”
- Comment a line with % or use /* */ for multi-line comments

Another example

saved in a file named **fac.pl**

```
factorial(0,1).
factorial(1,1).
factorial(N,R) :-  N > 1, N1 is N - 1,
                   factorial(N1, R1), R is N * R1.
                        % use "is" for "="
```

Running Prolog

```
?- consult(family).    % loading our file
true.                  % loading successfully

?- male(albert).
true.

?- [fac].              % another way to load file
true.

?- factorial(5,R).
R = 120 ;
false.

?- halt.                % exiting prolog
```

Asking queries

```
?- male(victoria).
false.

?- male(mycat).
false.

?- male(X).
X = albert ;    % type ";" or space for more answers
X = edward.

?- male(X).
X = albert .    % type return for no more answers
```

Asking queries (cont'd)

```
?- setof(X,male(X),Ans). % use "setof" to get all answers
Ans = [albert, edward].

?- sibling(alice,X).
X = edward ;
false.                          % no more answers

?- father(F,C).
F = albert,
C = edward ;
false.
```

What to learn?

- Syntax of Prolog
  - Terms, Facts and rules, Programs, Queries
- How does a Prolog interpreter answer queries?

Constants and variables

- Constants
  - identifiers: sequences of letters, digits, or underscore "_" that start with lower case letters,
    e.g., mary, x25, x_25, alpha_beta
  - numbers, e.g., 1.001, 2, 3.03
  - strings enclosed in single quotes, e.g., 'Mary', '1.01', 'string'
    Note: can start with upper case letters, or can be a number now treated as a string
- Variables
  - sequences of letters, digits, or underscore that start with upper case letters or underscore,
    e.g., Anna, _x, Successor_State
  - Underscore by itself is the special "anonymous" variable

Structures and lists

- Structures take the form: <identifier>(Term$_1$, ..., Term$_k$)
  e.g., date(1, may, 1983), point(X, Y, Z)
- Lists are structured terms represented in a special way
  e.g., [a,b,c,d]
- [] is a special constant the empty list.
- Each non-empty list is of the form [<head> | <rest_of_list>]
- Computing the sum of a list of nums: sumlist(List, Sum)

```
sumlist([],0).
sumlist([H|T],N) :-sumlist(T,N1), N is N1+H.
```

Facts

- Atoms take the form: <identifier>(Term$_1$, ..., Term$_k$)
- A fact is an atom terminated by a period "."
- Facts make assertions, e.g.,
  - elephant(mary).
  - taller_than(john, fred).
  - parent(X).
    Note that X is a variable. X is universally quantified so this fact asserts that for every value of X, "parent" is true.

Rules

- Rules take the form:
  atomH :- atom$_1$, ..., atom$_k$.
- The first atom is called the rule head.
- Note that a rule is terminated by a period.
- Rules encode ways of deriving or inferring new facts, e.g.
  - animal(X) :- elephant(X).
  - taller_than(X, Y) :- height(X, H1), height(Y, H2), H1 $>$ H2.
  - father(X,Y) :- parent(X,Y), male(X).

Programs and queries

- A prolog program consists of facts and rules
- A query is a sequence of atoms: $\text{atom}_1, \ldots, \text{atom}_k$
- Prolog tries to prove that this sequence of atoms is true using the facts and rules in the program.

How does Prolog answer queries

- Resolution forms the basis of the implementation of Prolog
- When searching for (), Prolog uses the following strategy: goal-directed, depth-first, top-down, left-right

```
elephant(fred).
elephant(mary).
elephant(joe).
animal(fred) :- elephant(fred).
animal(mary) :- elephant(mary).
animal(joe) :- elephant(joe).

QUERY
animal(fred), animal(mary), animal(joe)

1. elephant(fred), animal(mary), animal(joe)
2. animal(mary),animal(joe)
3. elephant(mary), animal(joe)
4. animal(joe)
5. elephant(joe)
6. EMPTY QUERY
```

Backtracking: an example

```
ant_eater(fred).
animal(fred) :- elephant(fred).
animal(fred) :- ant_eater(fred).
```

QUERY

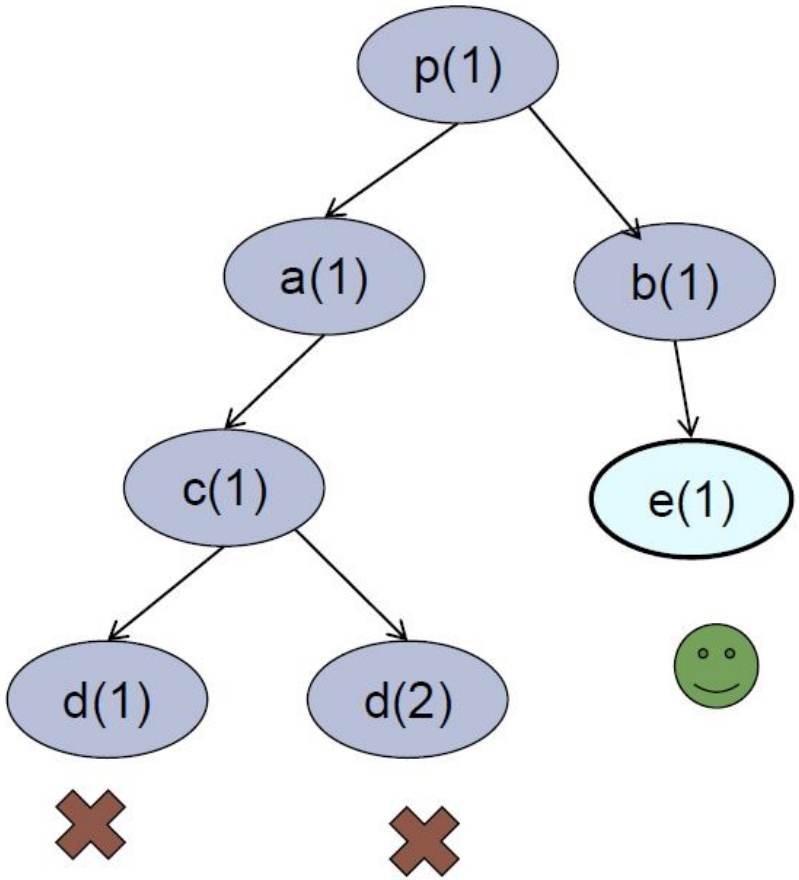
```
animal(fred)
```

1. elephant(fred).
2. FAIL BACKTRACK.
3. ant_eater(fred).
4. EMPTY QUERY

Backtracking: another example

```
p(1) :- a(1).
p(1) :- b(1).
a(1) :- c(1).
c(1) :- d(1).
c(1) :- d(2).
b(1) :- e(1).
e(1).
d(3).
```

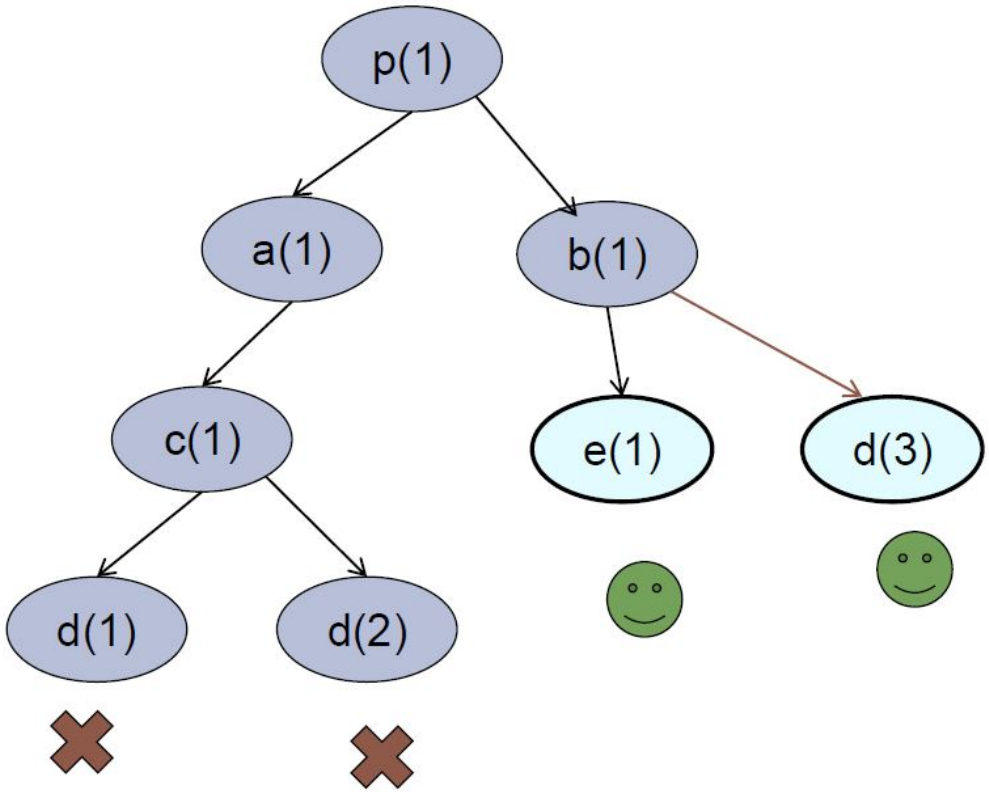
Query: p(1)



Backtracking for more answers

```
p(1) :- a(1).
p(1) :- b(1).
a(1) :- c(1).
c(1) :- d(1).
c(1) :- d(2).
b(1) :- e(1).
b(1) :- d(3).
e(1).
d(3).
```

Query: p(1)



Dealing with variables using unification

```
elephant(fred).
elephant(mary).
elephant(joe).
animal(X) :-elephant(X).

QUERY: animal(fred), animal(mary), animal(joe)

1. elephant(fred), animal(mary), animal(joe)
2. animal(mary),animal(joe)
3. elephant(mary), animal(joe)
4. animal(joe)
5. elephant(joe)
6. EMPTY QUERY
```