Prolog Programming AI(Artificial Intelligence)

Md. Alamgir Hossain

Senior Lecturer

Dept. of CSE, Prime University

Mail: *alamgir.cse14.just@gmail.com*

PROLOG

- Prolog is a logic programming language. It has important role in artificial intelligence.
- Unlike many other programming languages, Prolog is intended primarily as a declarative programming language.
- In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at the logic being applied.
- Formulation or Computation is carried out by running a query over these relations.

Features of Prolog Programming

- **Unification:** The basic idea is, can the given terms be made to represent the same structure.
- **Backtracking:** When a task fails, prolog traces backwards and tries to satisfy previous task.
- **Recursion:** Recursion is the basis for any search in program.

Advantages/Disadantages of Prolog Programming

- **Advantages :**
- ✓1. Easy to build database. Doesn’t need a lot of programming effort.
- ✓2. Pattern matching is easy. Search is recursion based.
- ✓3. It has built in list handling. Makes it easier to play with any algorithm involving lists.
- **Disadvantages :**
- ✓1. LISP (another logic programming language) dominates over prolog with respect to I/O features.
- ✓2. Sometimes input and output is not easy.
- **Applications :**
- ✓Prolog is highly used in artificial intelligence(AI). Prolog is also used for pattern matching over natural language parse trees.

Facts, Predicates of Prolog Programming

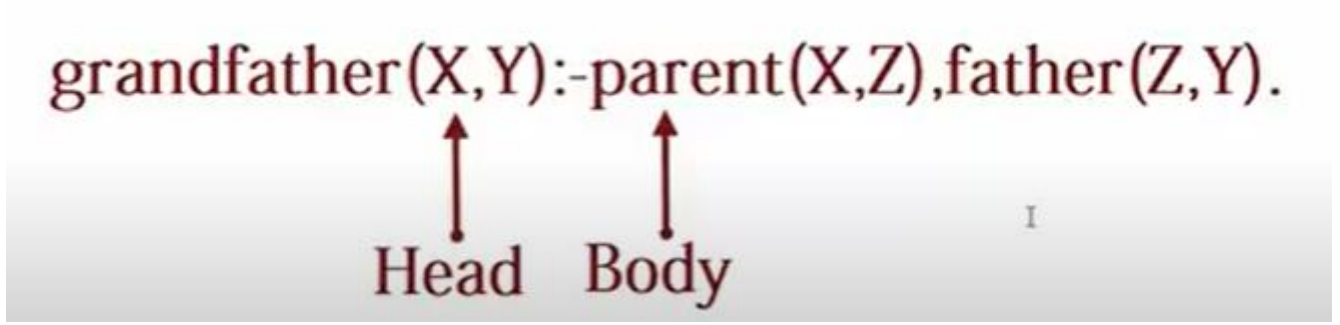
- A fact is a predicate expression that makes a declarative statement about the problem domain. Whenever a variable occurs in a Prolog expression, it is assumed to be universally quantified. Note that all Prolog sentences must end with a period.
- Fact is the relation between two or more objects. Name of fact is called predicate name.
  - ✓ likes(john, susie). /* John likes Susie */
  - ✓ likes(X, susie). /* Everyone likes Susie */
  - ✓ likes(john, Y). /* John likes everybody */
  - ✓ likes(john, Y), likes(Y, john). /* John likes everybody and everybody likes John */
  - ✓ likes(john, susie); likes(john, mary). /* John likes Susie or John likes Mary */
  - ✓ not(likes(john, pizza)). /* John does not like pizza */
  - ✓ likes(john, susie) :- likes(john,mary)./* John likes Susie if John likes Mary.
  - ✓ father(Bill, Alen) /*Bill is the father of Alen*/

Rules of Prolog Programming

- A rule is a predicate expression that uses logical implication (:-) to describe a relationship among facts. Thus a Prolog rule takes the form ***left_hand_side :- right_hand_side***
- **This sentence is interpreted as: *left_hand_side if right_hand_side.***
- The **left_hand_side** is restricted to a **single, positive, literal**, which means it must consist of a positive atomic expression. It cannot be negated and it cannot contain logical connectives.
- **Examples of valid rules:**
  - friends(X,Y) :- likes(X,Y),likes(Y,X). /* X and Y are friends if they like each other */
  - hates(X,Y) :- not(likes(X,Y)). /* X hates Y if X does not like Y. */
  - enemies(X,Y) :- not(likes(X,Y)),not(likes(Y,X)). /* X and Y are enemies if they don't like each other */
- **Examples of invalid rules:**
  - ✓ left_of(X,Y) :- right_of(Y,X) /* Missing a period */
  - ✓ likes(X,Y),likes(Y,X) :- friends(X,Y). /* LHS is not a single literal */
  - ✓ not(likes(X,Y)) :- hates(X,Y). /* LHS cannot be negated */

Clause of Prolog Programming

- In Prolog, the program contains a sequence of one or more clauses.
- The clauses can run over many lines. Using a dot character, a clause can be terminated.
- This dot character is followed by at least one 'white space' character.
- The clauses are of two types: facts and rules.



Clause of Prolog Programming

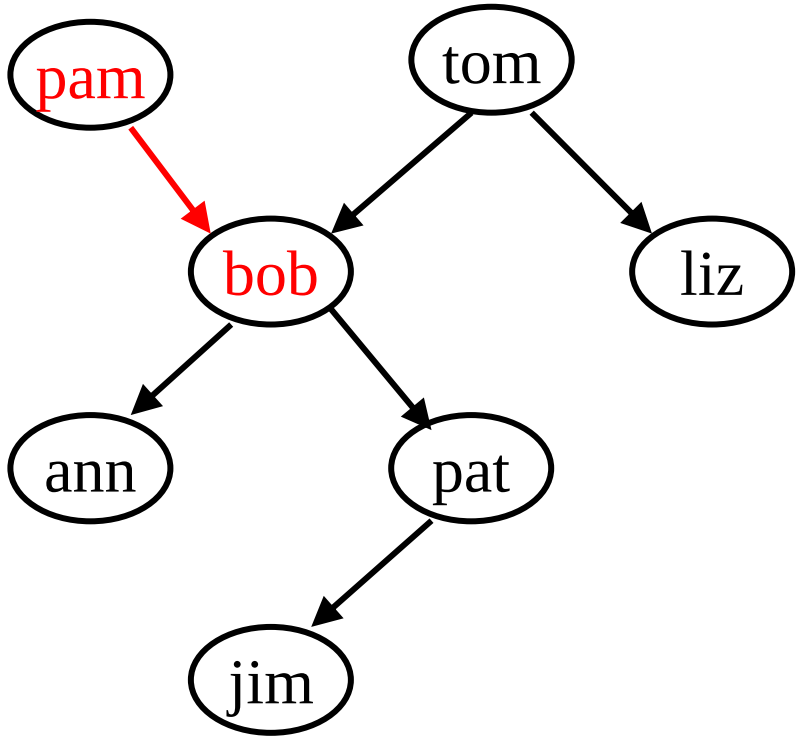
- Prolog clauses consist of
  - ✓Head
  - ✓Body: a list of goal separated by commas (,)
- Prolog clauses are of three types:
  - ✓Facts:
    - declare things that are always true
    - facts are clauses that have a head and the empty body
  - ✓Rules:
    - declare things that are true depending on a given condition
    - rules have the head and the (non-empty) body
  - ✓Questions:
    - the user can ask the program what things are true
    - questions only have the body

Addition of two numbers in Prolog

```
% Addition of two numbers

go:- write('Enter the first number: '), read(X), nl,
     write('Enter the second number: '), read(Y), nl,
     sum(X, Y).

sum(X, Y):-
    S is X + Y,
    write('Summation is: '),
    write(S), nl.

```

Defining Relations by Facts

- Given a whole family tree

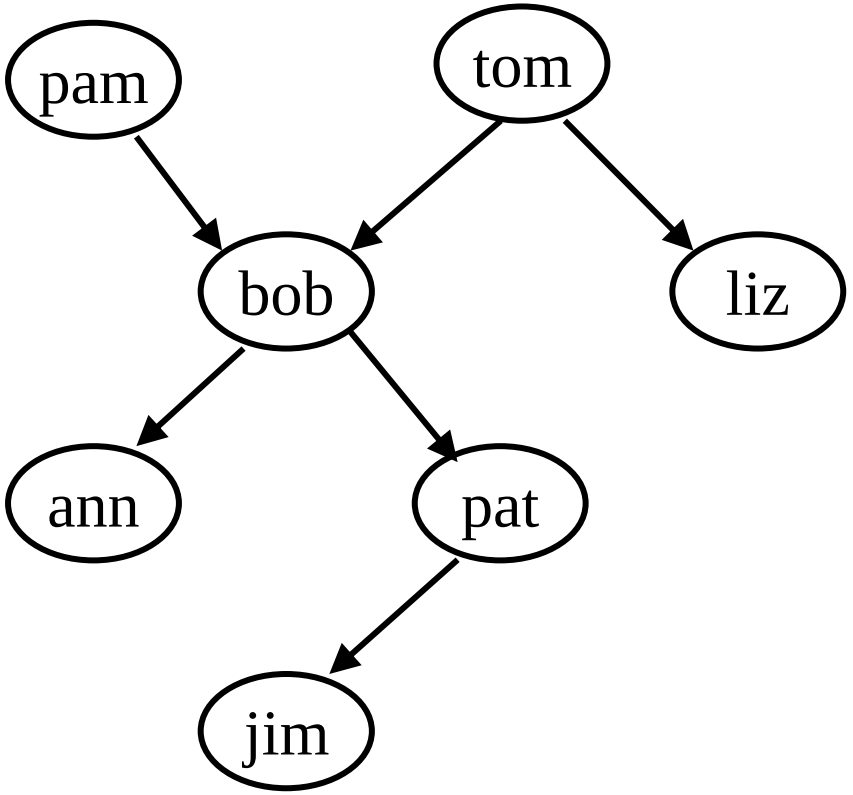


- The tree defined by the Prolog program:

```
parent( pam, bob).      % Pam is a parent of Bob
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

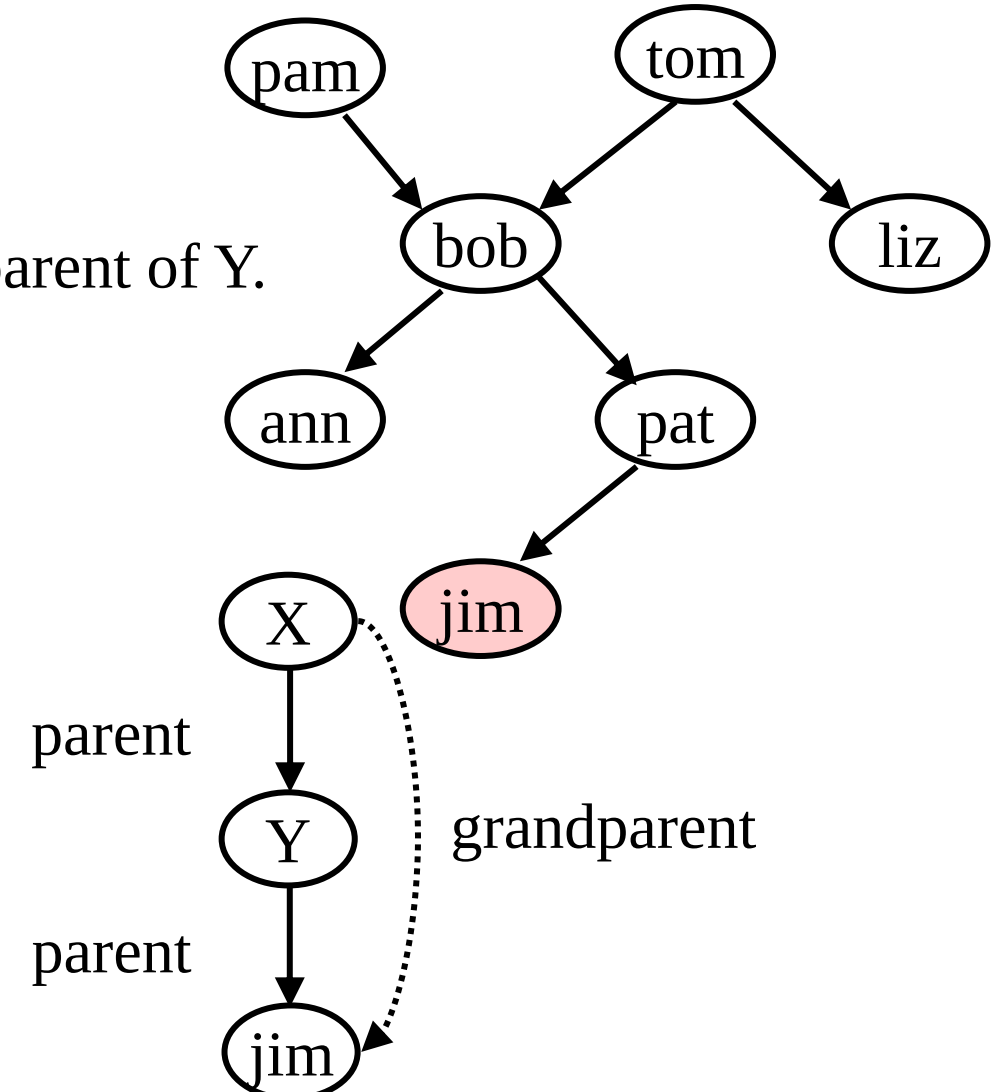
Defining Relations by Facts

- Questions:
  - Is Bob a parent of Pat?
    - ?- parent(bob, pat).
    - ?- parent(liz, pat).
    - ?- parent(tom, ben).
  - Who is Liz's parent?
    - ?- parent(X, liz).
  - Who are Bob's children?
    - ?- parent(bob, X).



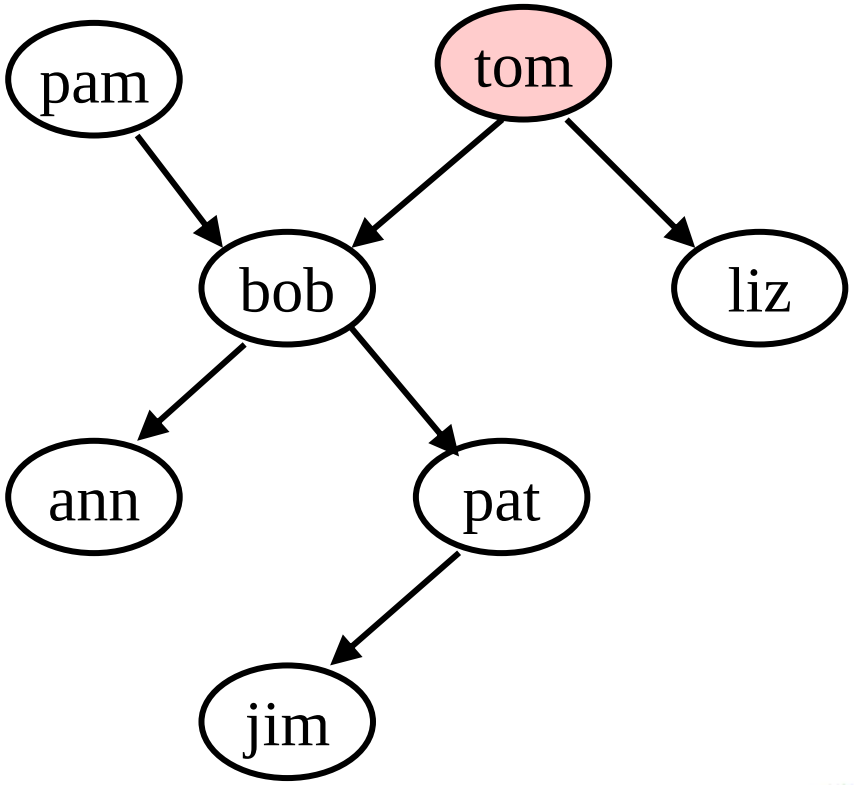
Defining Relations by Facts

- Questions:
  - Who is a parent of whom?
    - Find X and Y such that X is a parent of Y.
    - ?- parent(X, Y).
  - Who is a grandparent of Jim?
    - ?- parent(Y, jim),
      parent(X, Y).



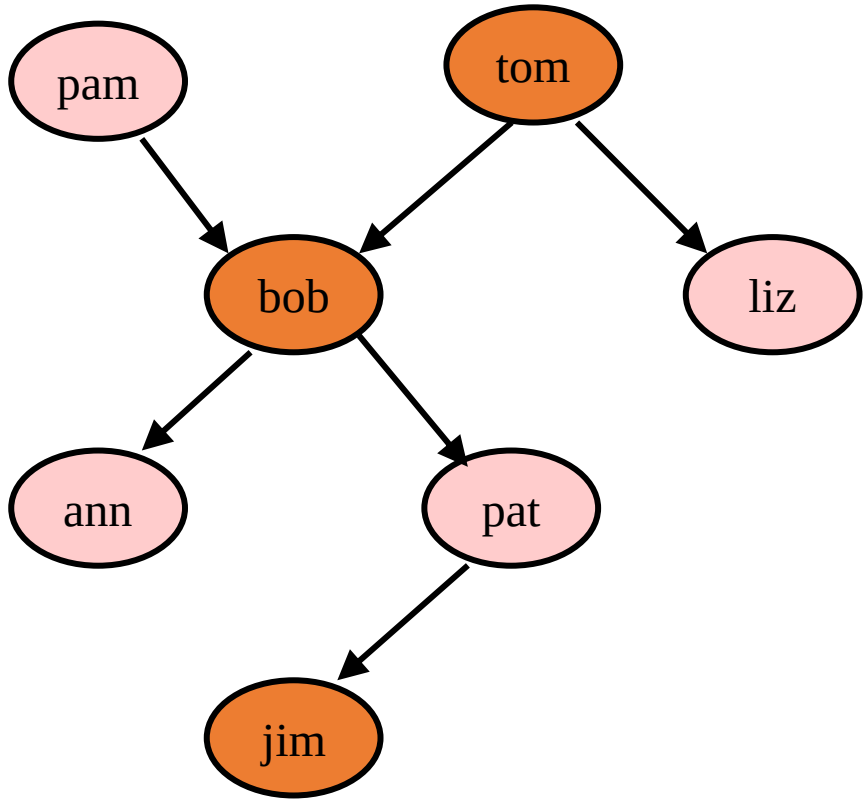
Defining Relations by Facts

- Questions:
  - Who are Tom's grandchildren?
    - ?- parent(tom, X),
      parent(X, Y).
  - Do Ann and Pat have a common parent?
    - ?- parent(X, ann),
      parent(X, pat).



Defining Relations by Rules

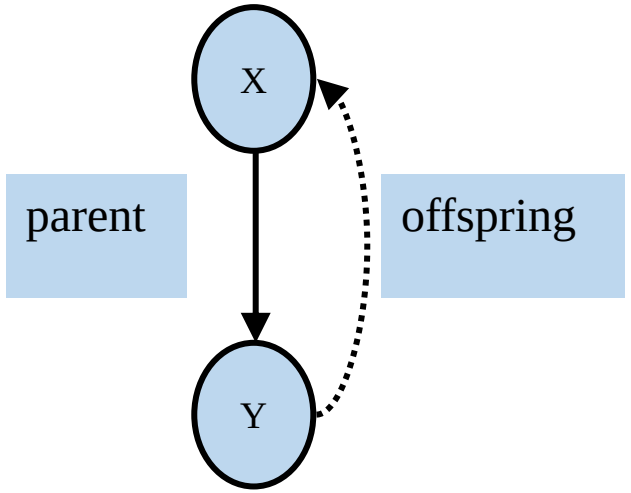
- Facts:
  - female(pam). % Pam is female
  - female(liz).
  - female(ann).
  - female(pat).
  - male(tom). % Tom is male
  - male(bob).
  - male(jim).



- Define the “offspring” relation:
  - Fact: offspring(liz, tom).
  - Rule: **offspring(Y, X) :- parent(X, Y).**
    - For all X and Y,
      Y is an offspring of X if X is a parent of Y.

Defining Relations by Rules

- **Rules** have:
  - A condition part (body)
    - the right-hand side of the rule
  - A conclusion part (head)
    - the left-hand side of the rule



  - Example:
    - **offspring(Y, X) :- parent(X, Y).**
    - The rule is general in the sense that it is applicable to any objects X and Y.
    - A special case of the general rule:
      - offspring(liz, tom) :- parent(tom, liz).
    - ?- offspring(liz, tom).
    - ?- offspring(X, Y).

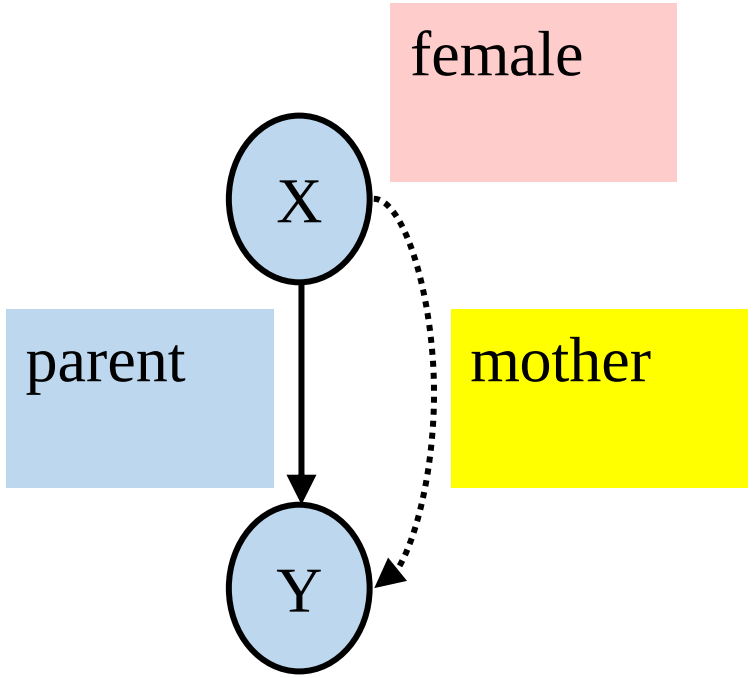
Defining Relations by Rules

- Define the “mother” relation:
  - **mother(X, Y) :- parent(X, Y), female(X).**
  - For all X and Y,

    X is the mother of Y if

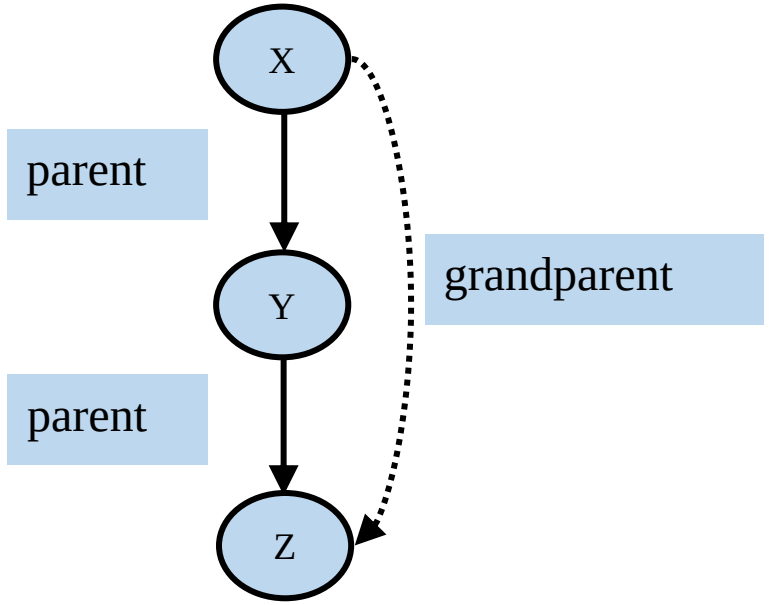
    X is a parent of Y and

    X is a female.



Defining Relations by Rules

- Define the “grandparent” relation:
  - **grandparent(X, Z) :-**
    parent(X, Y), parent(Y, Z).



Defining Relations by Rules

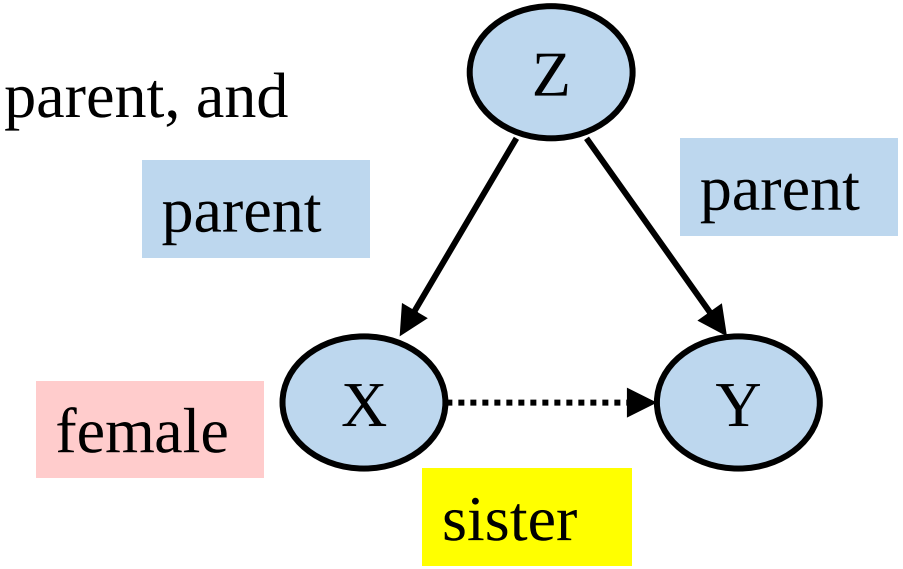
- Define the “sister” relation:
  - **sister(X, Y) :-**

    parent(Z, X), parent(Z, Y), female(X).
  - For any X and Y,

    X is a sister of Y if

    (1) both X and Y have the same parent, and

    (2) X is female.
  - ?- sister(ann, pat).
  - ?- sister(X, pat).
  - ?- sister(pat, pat).
    - Pat is a sister to herself?!

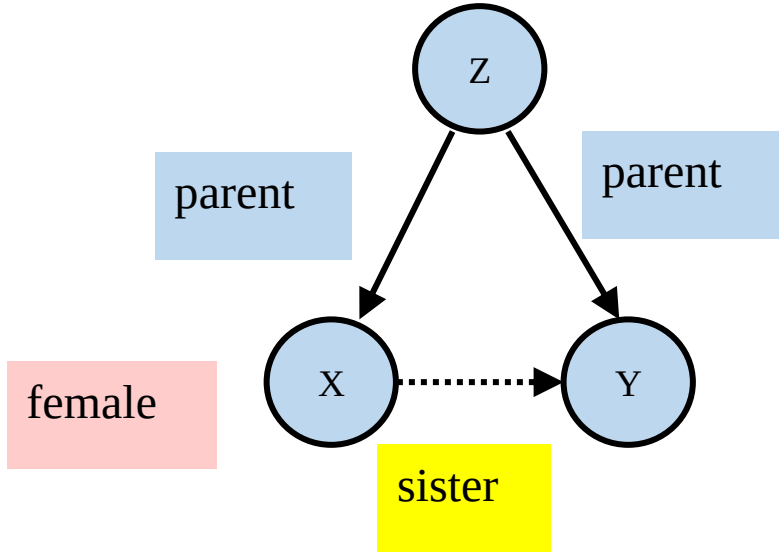


Defining Relations by Rules

- To correct the “sister” relation:
  - **sister(X, Y) :-**

    parent(Z, X), parent(Z, Y), female(X),

    different(X, Y).
  - different (X, Y) is satisfied if and only if X and Y are not equal. (Please try to define this function)



Defining Relations by Rules

- A variable can be substituted by another object.

- Variables are assumed to be universally quantified and are read as "for all".
  - ✓ For example:

    hasachild(X) :- parent(X, Y).

    can be read in two way

    (a) For all X and Y,

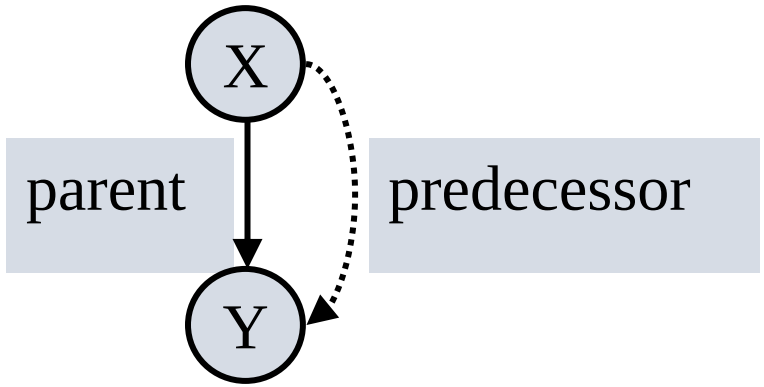
    if X is a parent of Y then X has a child.

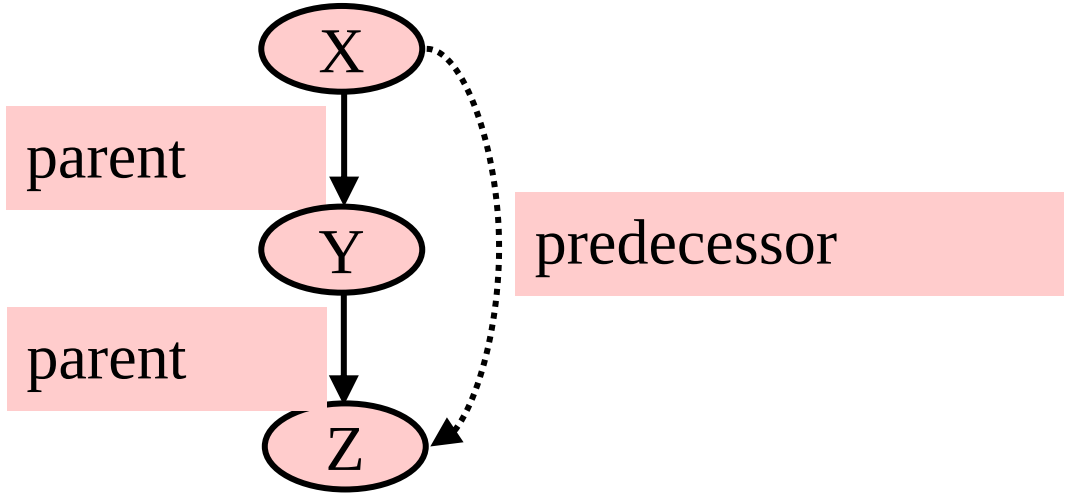
    (b) For all X,

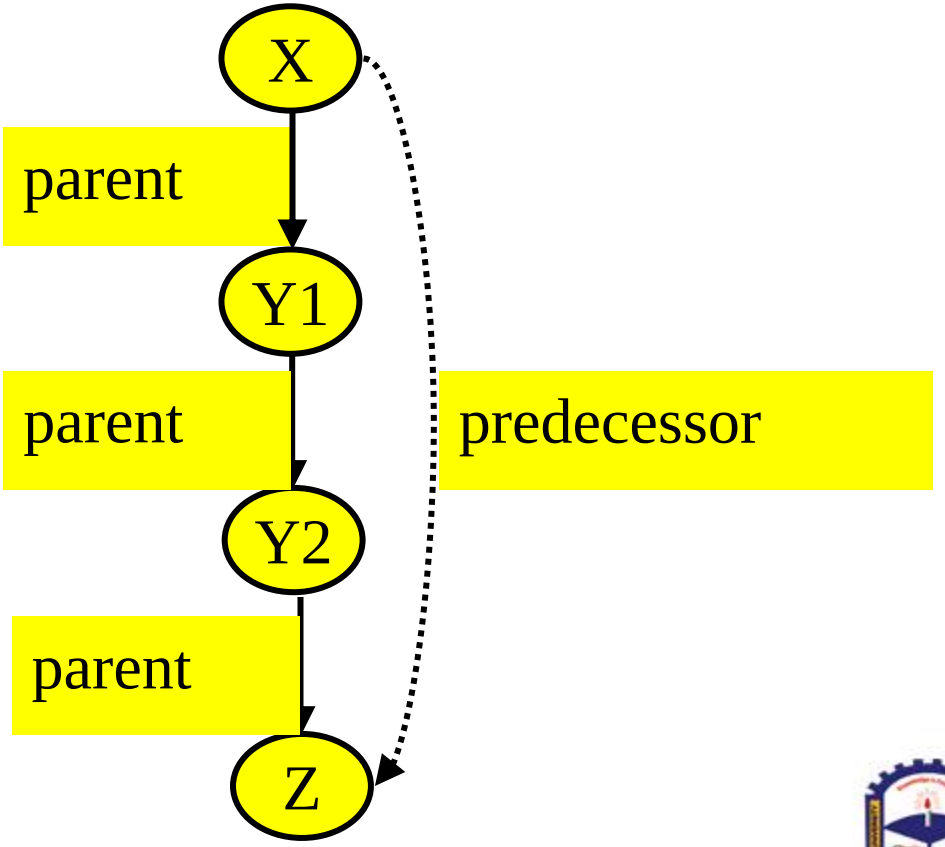
    X has a child if there is some Y such that X is a parent of Y.

Recursive Rules

- Define the “predecessor” relation







Recursive Rules

- Define the "predecessor" relation

  predecessor(X, Z):- parent(X, Z).

  predecessor(X, Z):-

  parent(X, Y), predecessor(Y, Z).

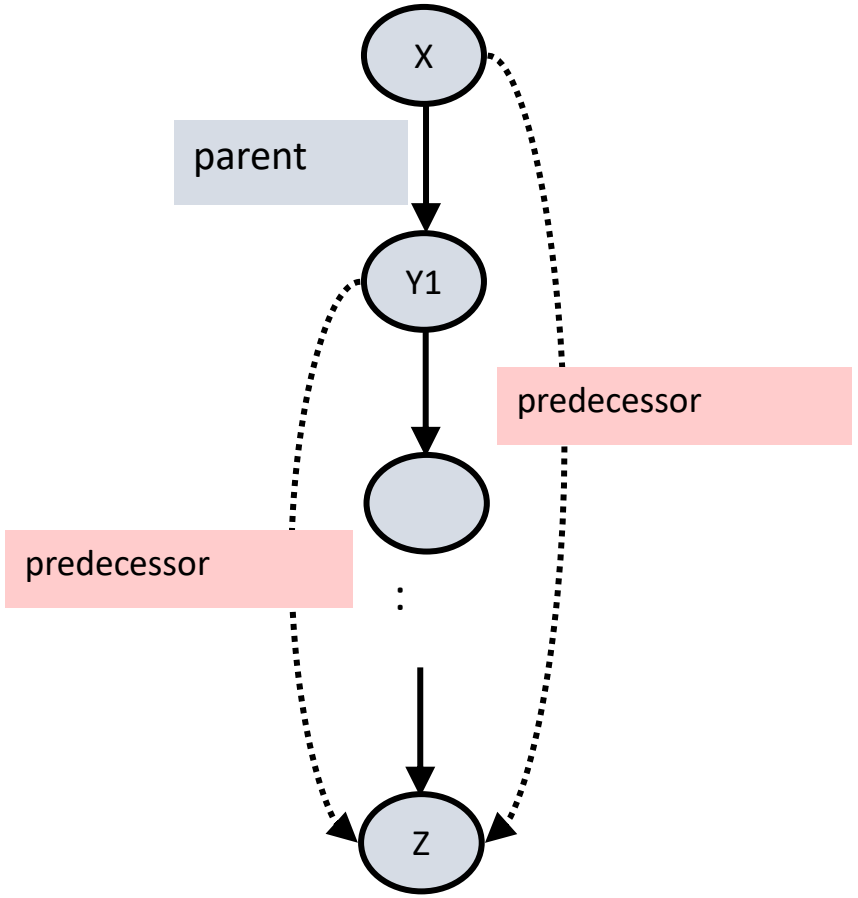
  ✓ For all X and Z,

  X is a predecessor of Z if there is a Y such that

  (1) X is a parent of Y and

  (2) Y is a predecessor of Z.

  ✓ ?- predecessor(pam, X).



Recursive Rules

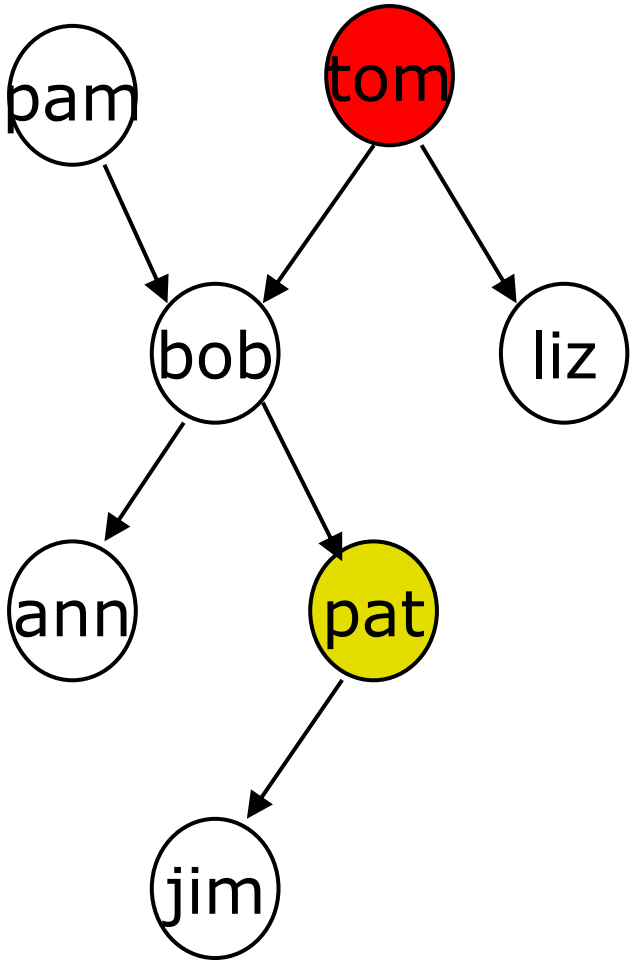
```
%The family program.

parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

```
female( pam).
female( liz).
female( ann).
female( pat).
male( tom).
male( bob).
male( jim).

offspring( Y, X)  :-
   parent( X, Y).
```

```
mother( X, Y)  :-
   parent( X, Y),
   female( X).

grandparent( X,
   Z)  :-
   parent( X, Y),
   parent( Y, Z).
```

```
sister( X, Y)  :-
   parent( Z, X),
   parent( Z, Y),
   female( X),
   X \= Y.

predecessor( X, Z)  :-   % Rule pr1
   parent( X, Z).

predecessor( X, Z)  :-   % Rule pr2
   parent( X, Y),
   predecessor( Y, Z).
```

How Prolog Answer Questions

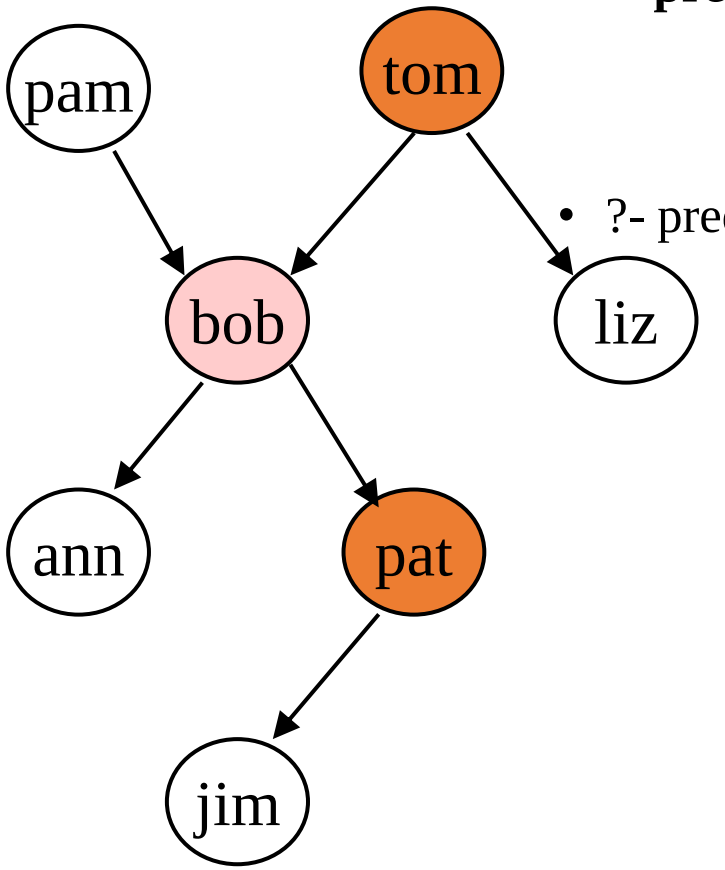



```
predecessor( X, Z)  :- parent( X, Z).        % Rule pr1
predecessor( X, Z)  :- parent( X, Y),        % Rule pr2
                       predecessor( Y, Z).
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

- ?- predecessor(tom, pat).
  - How does the Prolog system actually find a proof sequence?
    - Prolog first tries that clause which appears first in the program. (rule pr1)
    - Now, X= tom, Z = pat.
    - The goal predecessor(tom, pat) is then replace by parent(tom, pat).
    - There is no clause in the program whose head matches the goal parent(tom, pat).
    - Prolog backtracks to the original goal in order to try an alternative way (rule pr2).

How Prolog Answer Questions




```
predecessor( X, Z)  :- parent( X, Z).          % Rule pr1
predecessor( X, Z)  :- parent( X, Y),          % Rule pr2
                       predecessor( Y, Z).
```

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

- ?- predecessor(tom, pat).
  - Apply rule pr2, X = tom, Z = pat, but Y is not instantiated yet.
  - The top goal predecessor(tom, pat) is replaces by two goals:
    - parent(tom, Y)
    - predecessor(Y, pat)
  - The first goal matches one of the facts. (Y = bob)
  - The remaining goal has become
    predecessor(bob, pat)
  - Using rule pr1, this goal can be satisfied.
    - predecessor(bob, pat) :- parent(bob, pat)

Thank You