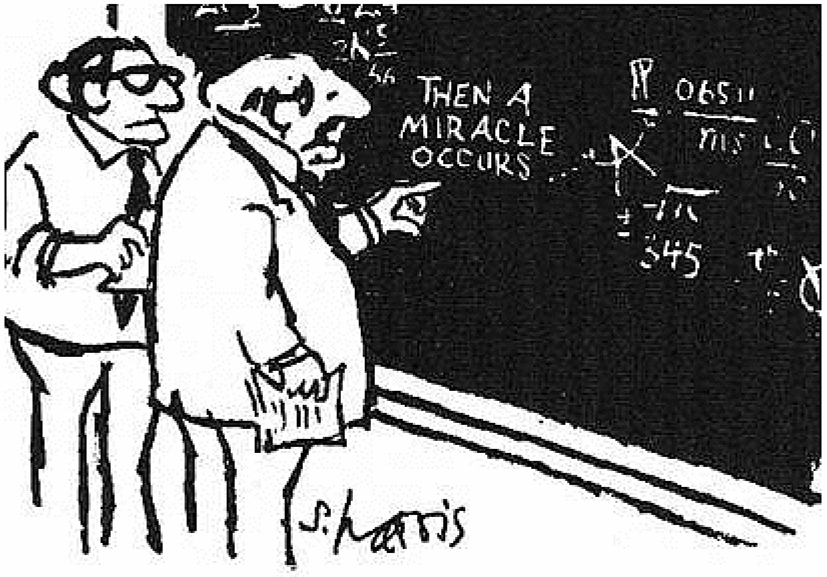
Introduction to Prolog



"I think you should be more explicit here in step two."

Introduction to Prolog

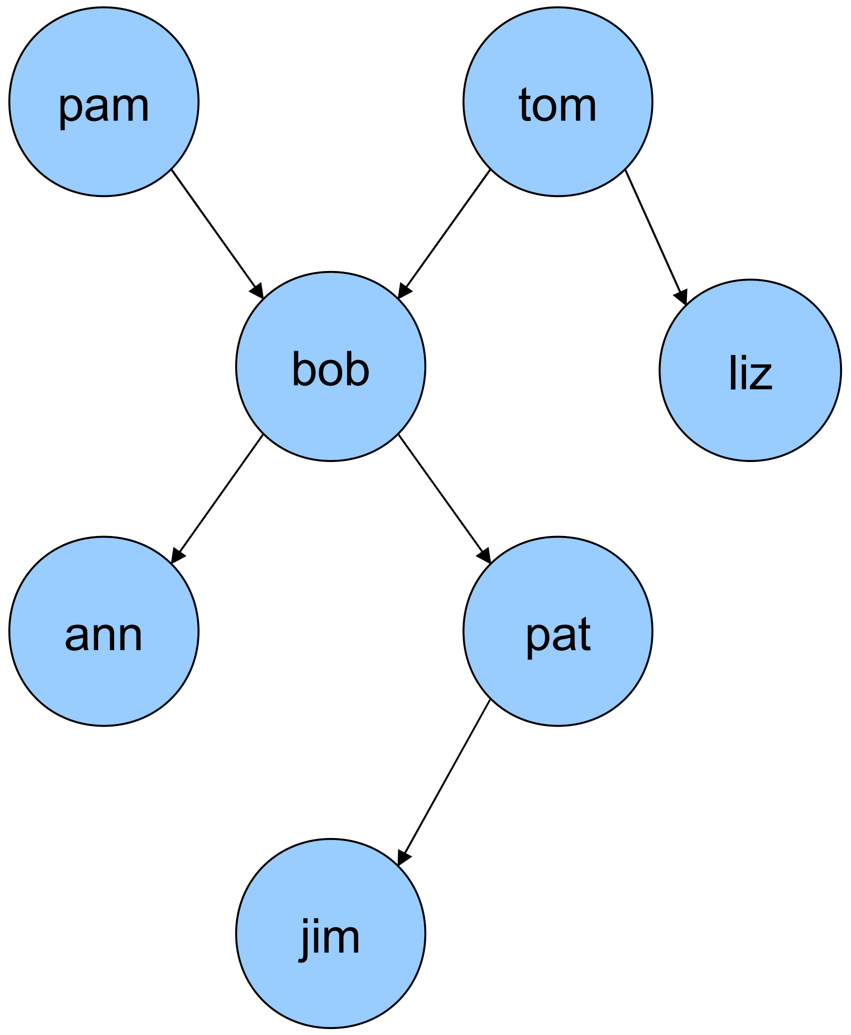
- Material taken from text:
  - PROLOG Programming for Artificial Intelligence by Ivan Bratko.
- Online Tutorials:
  - http://kti.ms.mff.cuni.cz/~bartak/prolog/
  - http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/intro.html

A Family Tree

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

Clauses

A Family Tree



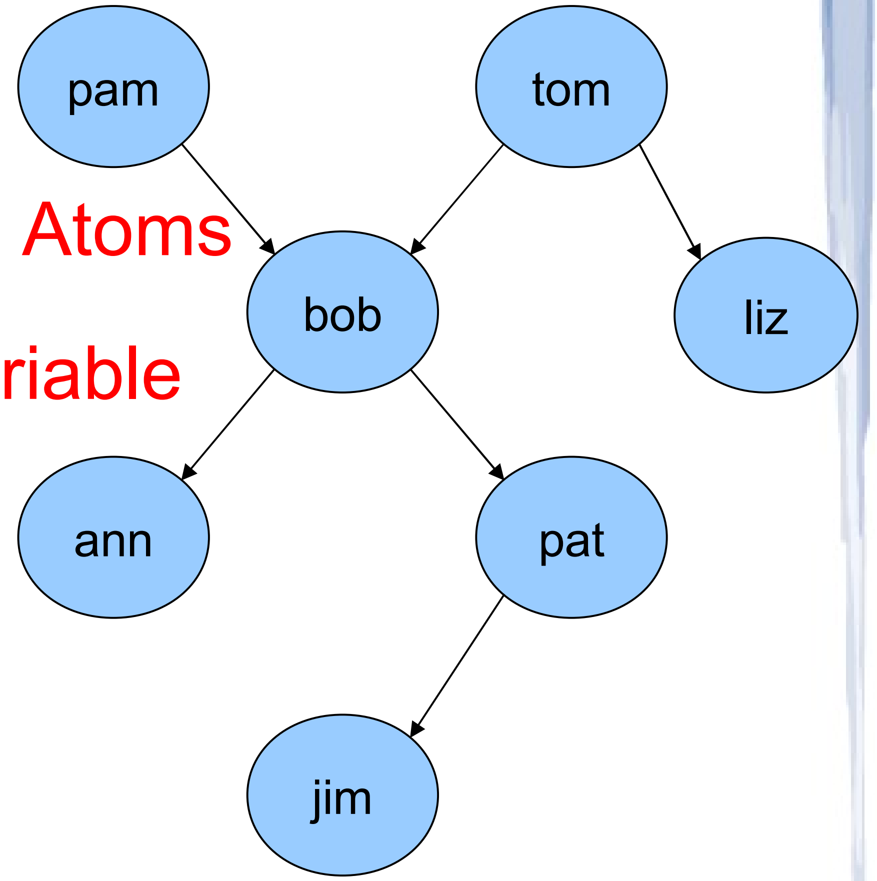
A Family Tree

- ?- parent(bob, pat).
  - yes
- ?- parent(liz, pat).
  - no
- ?- parent(X, liz).
  - X = tom
- ?- parent(bob, X)
  - X = ann;
  - X = pat;
  - no

Atoms

Variable

Continue

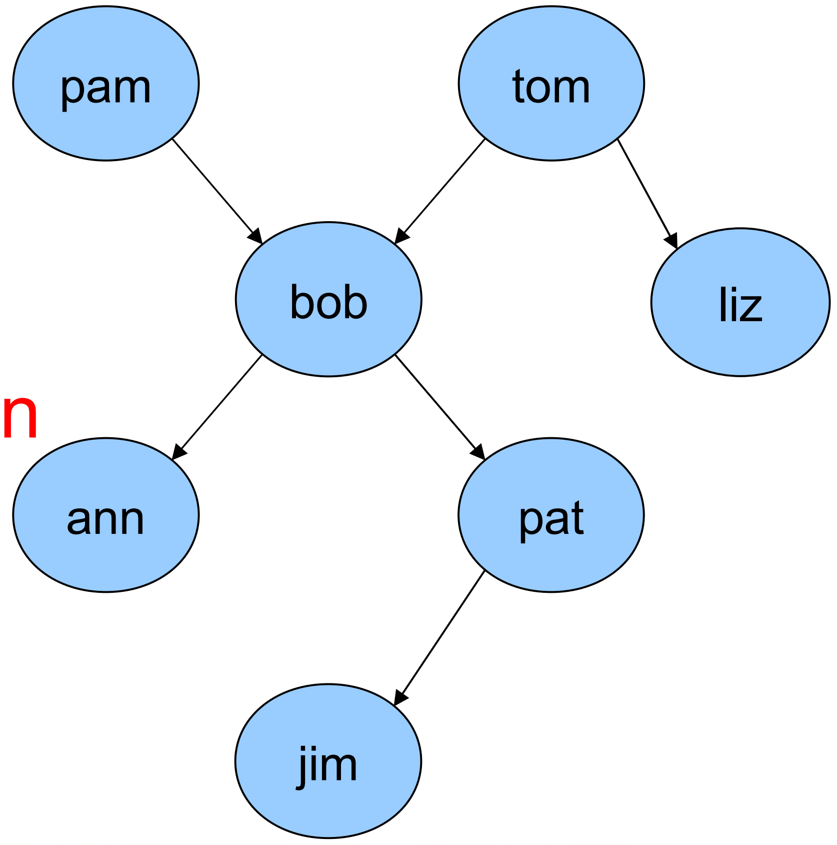


A Family Tree

Who is a grandparent of jim?

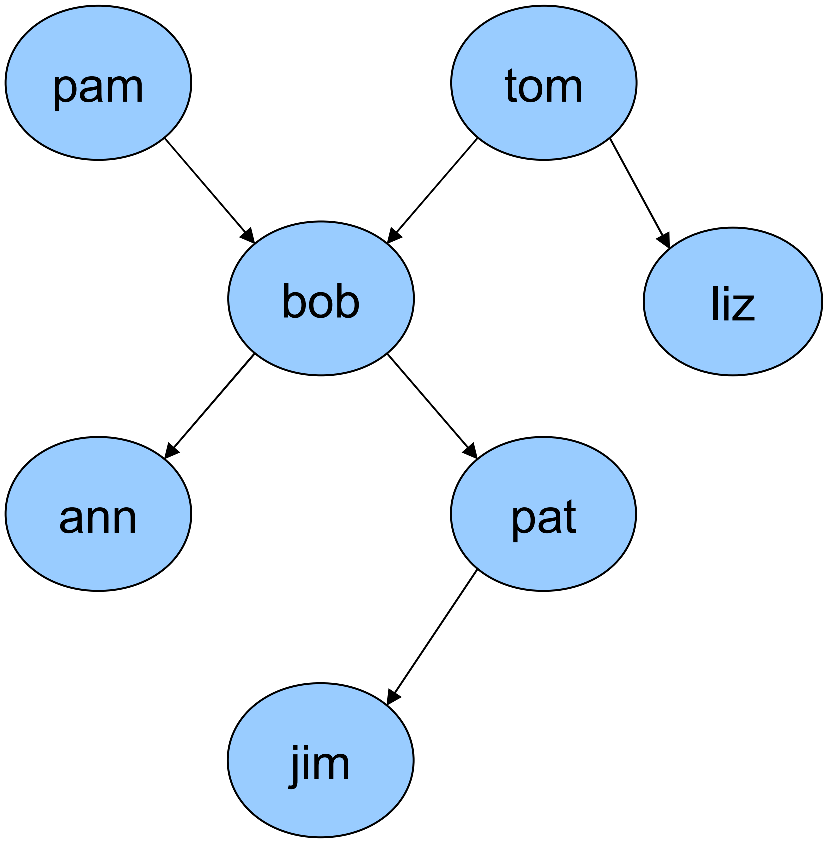
- ?- parent(Y, jim), parent(X, Y).
- X = bob
- Y = pat

Conjunction



A Family Tree

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

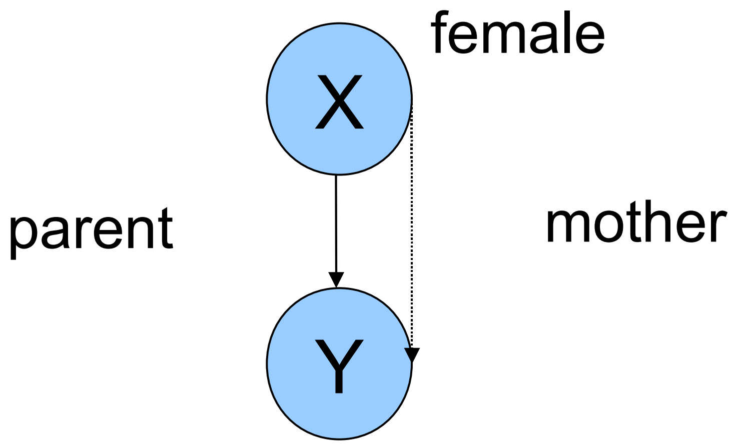


A Family Tree

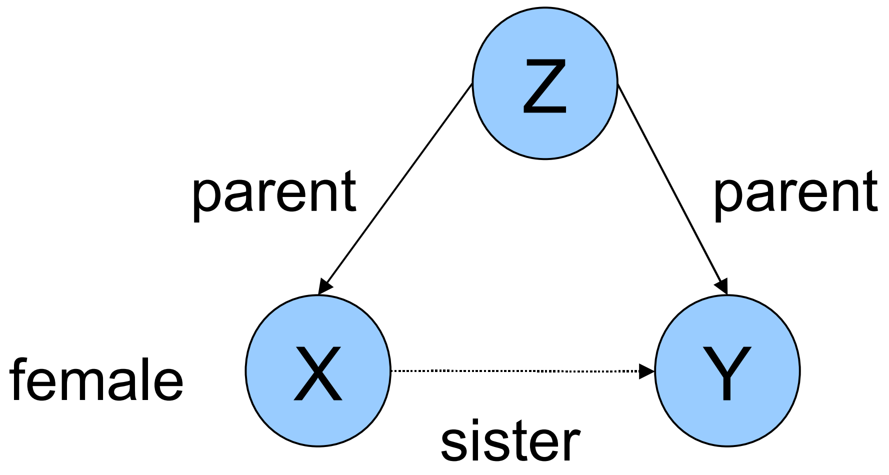
- Offspring
  - For all X and Y,
    - Y is an offspring of X if
    - X is a parent of Y.
- Prolog Rule:
  - $\underbrace{\texttt{offspring(Y, X)}}_{\text{conclusion}}\ \texttt{:-}\ \underbrace{\texttt{parent(X, Y).}}_{\text{condition}}$

A Family Tree

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



A Family Tree



- sister(X, Y) :-

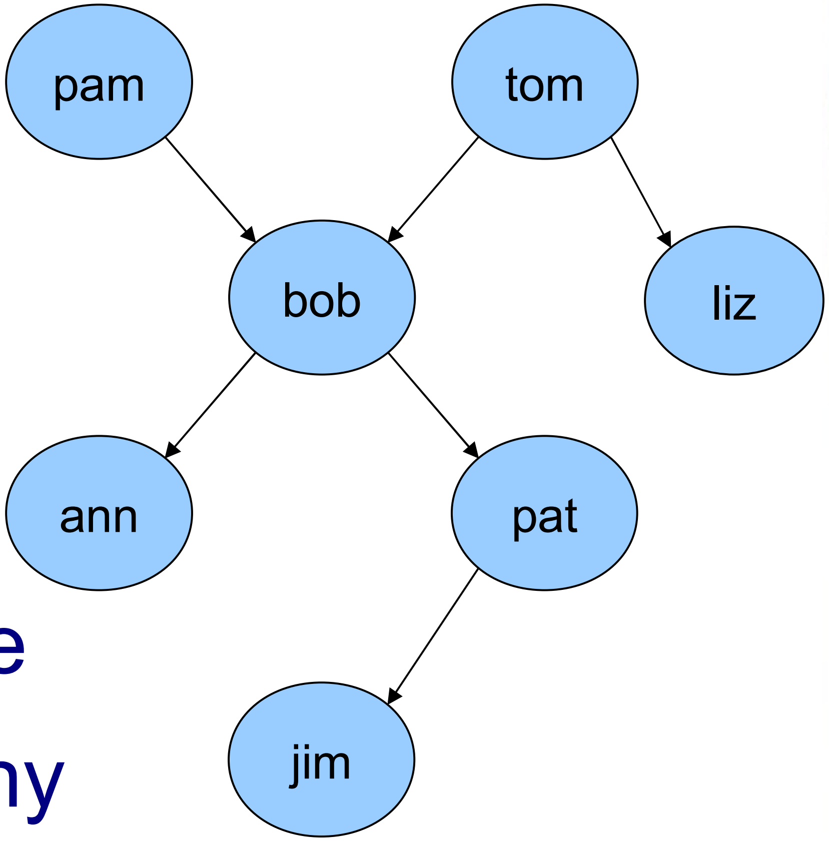
  parent(Z, X),

  parent(Z, Y),

  female(X).

A Family Tree

- sister(X, pat).
  - X = ann;
  - X = pat
- Our rule does not mention that X and Y must not be the same and so will find that any female who has a parent is a sister of herself!



A Family Tree

- New sister relation:
  - sister(X, Y) :-

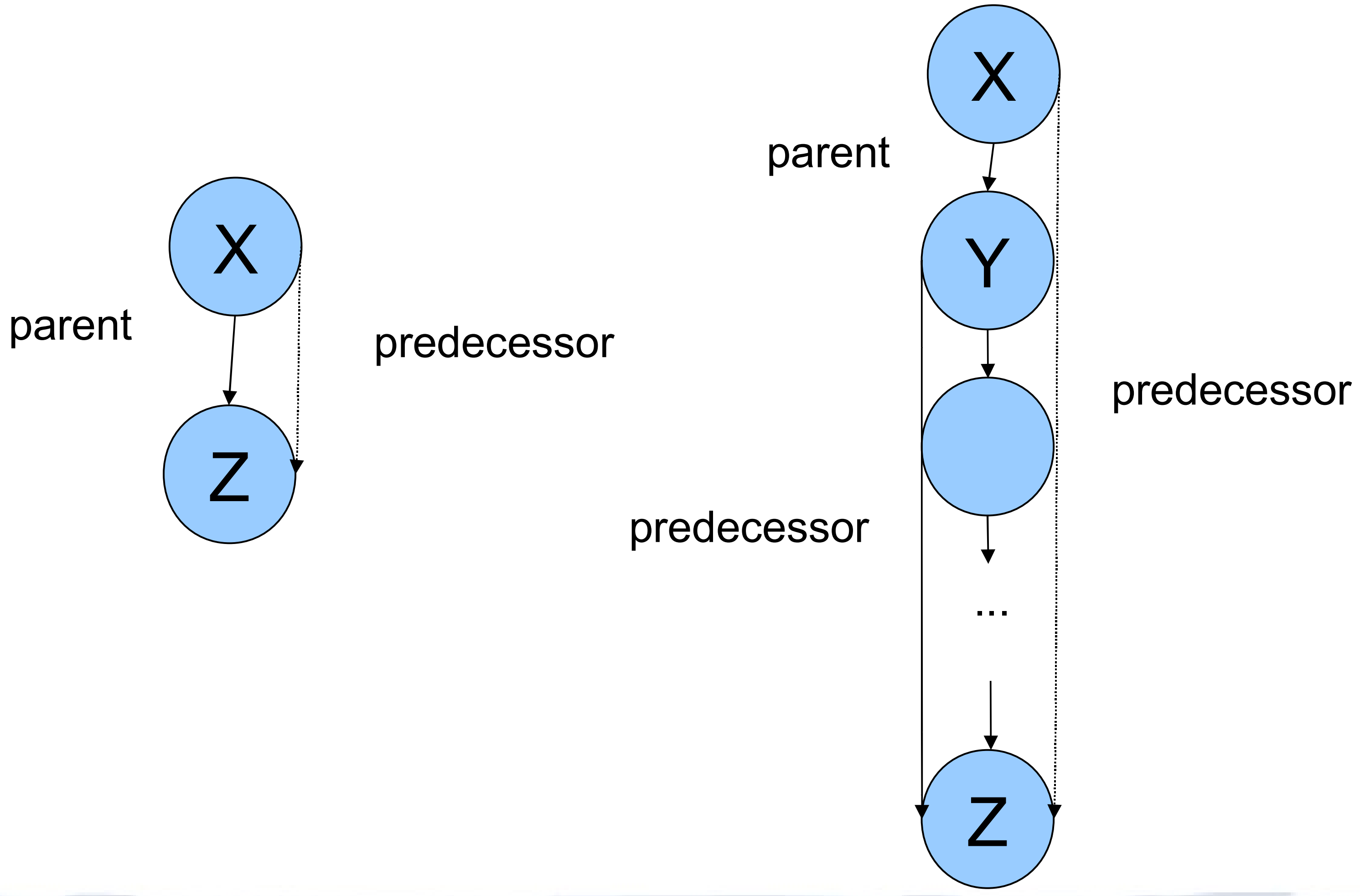
parent(Z, X),
parent(Z, Y),
female(X),
different(X, Y).

A Family Tree

- Recap
  - Male/Female
  - Offspring
  - Mother
  - Grandparent
  - Sister

A Family Tree

- A Recursive Rule (Predecessor)

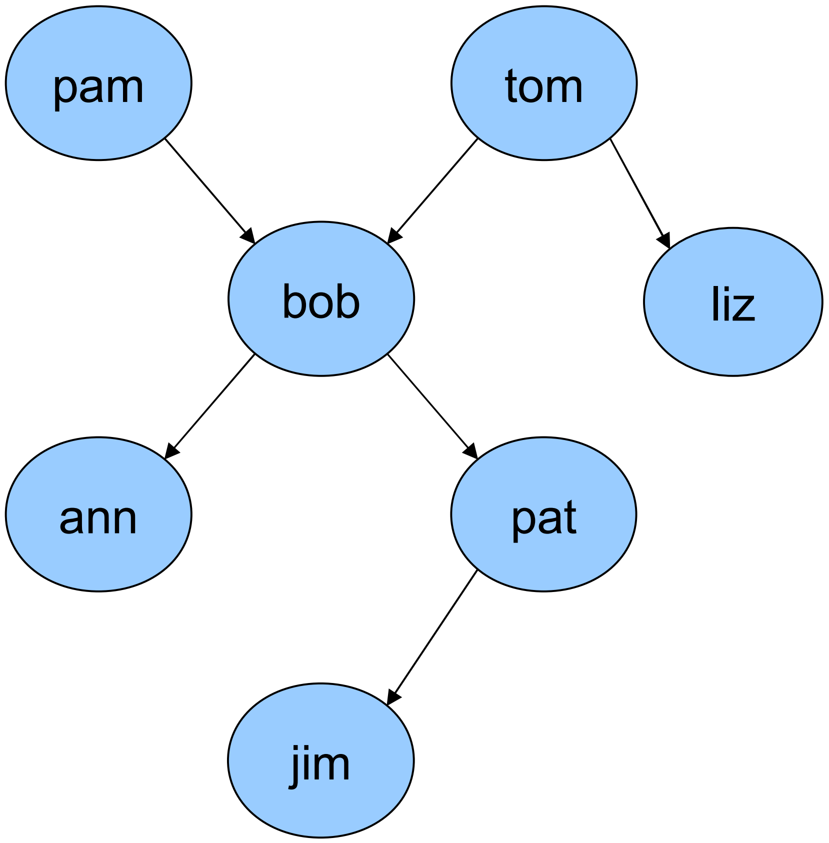


A Family Tree

- Prolog Clause:
  - predecessor(X, Y) :-
    parent(X, Y),
  - predecessor(X, Z) :-
    parent(X, Y),
    predecessor(Y, Z).

A Family Tree

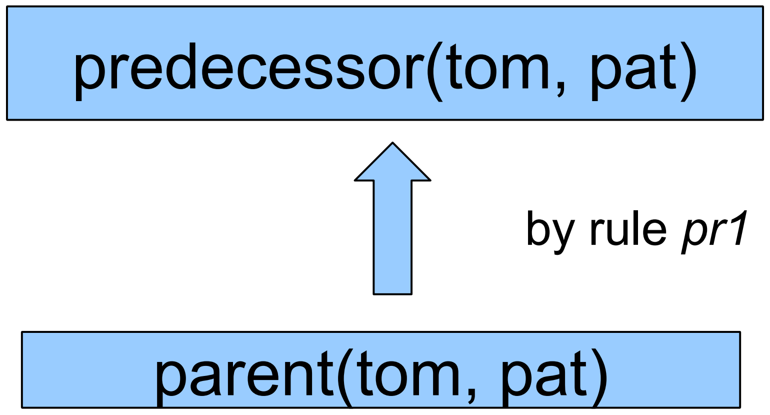
- ?- predecessor(pam, X).
  - X = bob;
  - X = ann;
  - X = pat;
  - X = jim;


$$\begin{tikzcd}
\text{pam} \arrow[dr] & & \text{tom} \arrow[dl] \arrow[dr] & \\
& \text{bob} \arrow[dl] \arrow[dr] & & \text{liz} \\
\text{ann} & & \text{pat} \arrow[dl] & \\
& \text{jim} & &
\end{tikzcd}$$

How prolog answers questions.

- ?- predecessor(tom, pat).
  - Only relevant clauses are *pr1* and *pr2*.
  - Heads of these rules match the goal.
  - Prolog chooses the first clause (*pr1*)
    - predecessor(X, Z) :-
      parent(X, Z).
    - X = tom
    - Z = pat
  - New goal:
    - parent(tom, pat). ✗

How prolog answers questions.



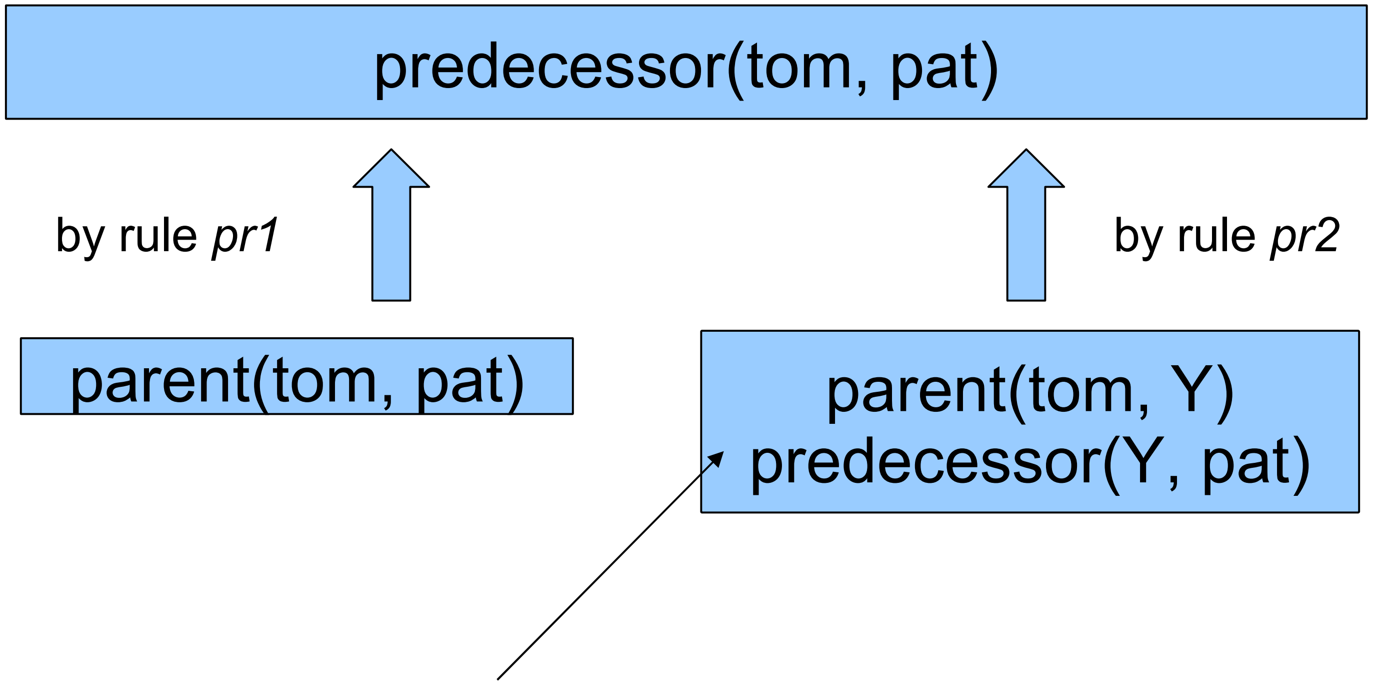
How prolog answers questions.

- Prolog now chooses the second clause (*pr2*)
  - predecessor(X, Z) :-

    parent(X, Y),

    predecessor(Y, Z).
  - X = tom
  - Z = pat
  - Y not yet instantiated

How prolog answers questions.



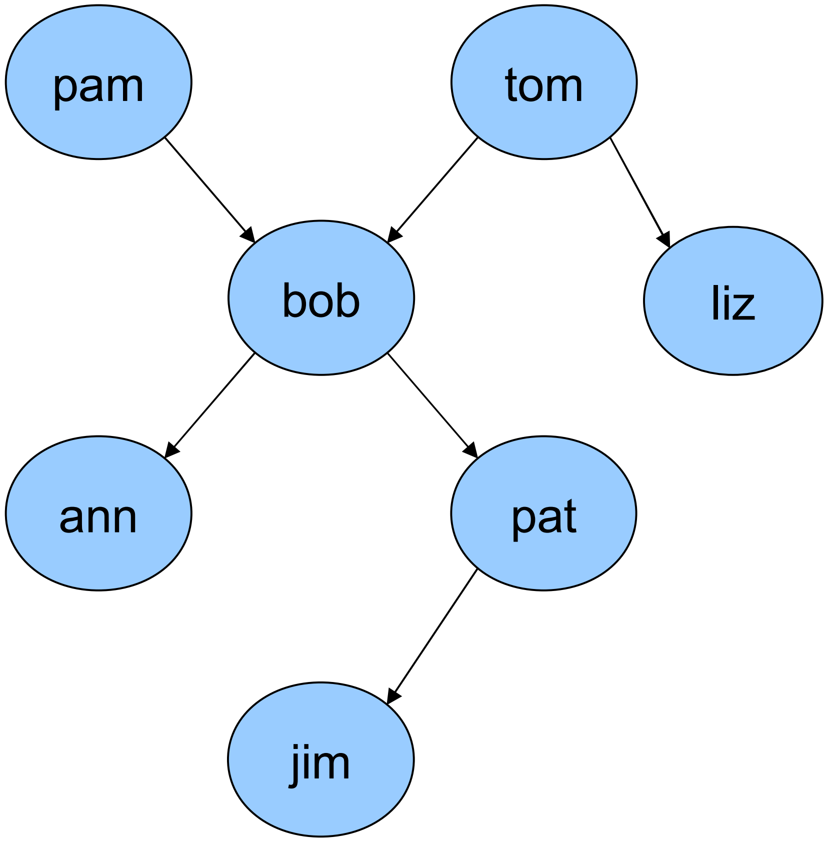
Two new goals

How prolog answers questions.

Current Goals

parent(tom, Y)
predecessor(Y, pat)

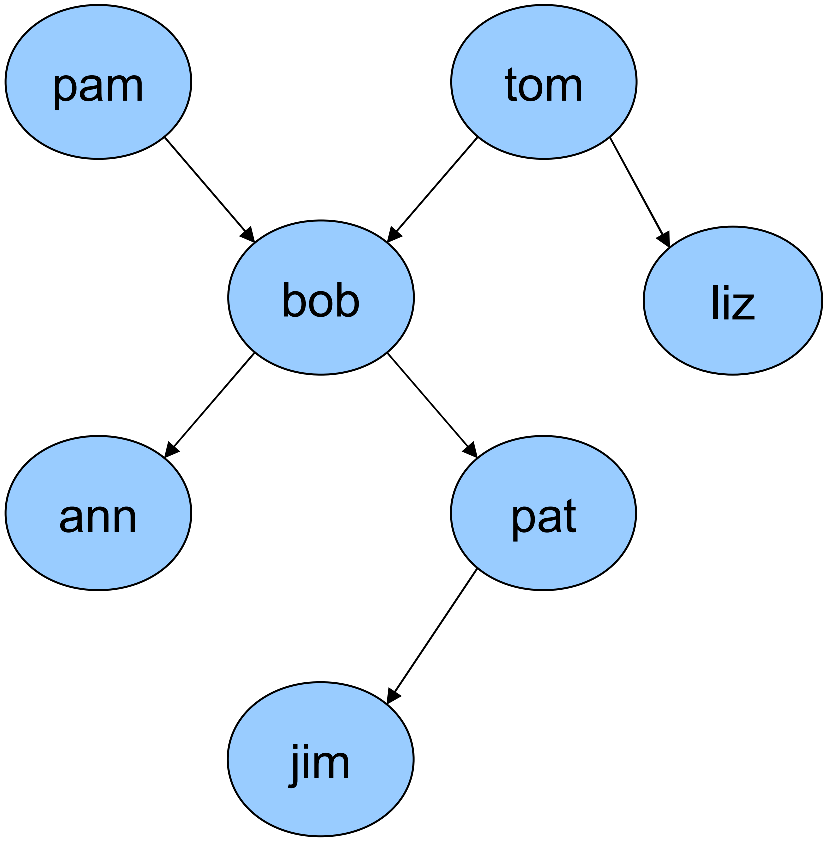
- Y = bob
- parent(tom, bob)
- predecessor(bob, pat)



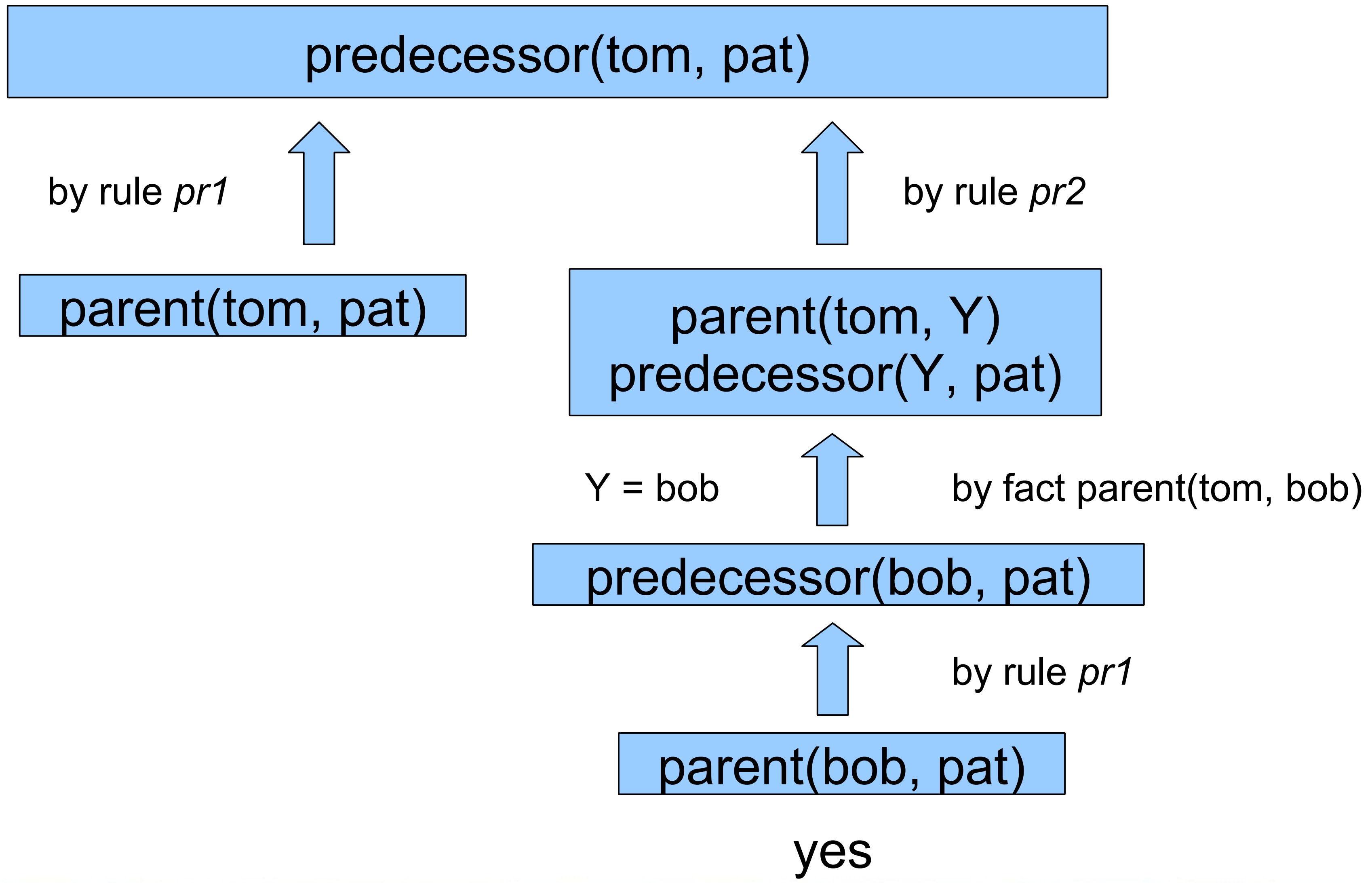
How prolog answers questions.

predecessor(bob, pat)

- By *pr1:*
  - predecessor(X′, Z′) :-
    parent(X′, Z′).
- =>
  - parent(bob, pat) 🙂



How prolog answers questions.



Monkey & Banana

The Problem:

- There is a monkey at the door into a room. In the middle of the room a banana is hanging from the ceiling. The monkey is hungry and wants to get the banana, but he cannot stretch high enough from the floor. At the window of the room there is a box the monkey may use.

Monkey & Banana

- The monkey can perform the following actions:
  - Walk on the floor
  - Climb the box
  - Push the box around (if it is already at the box)
  - Grasp the banana if standing on the box directly under the banana.

Monkey & Banana

- Monkey World is described by some 'state' that can change in time.
- Current state is determined by the position of the objects
- State:
  - Monkey Horizontal
  - Monkey Vertical
  - Box Position
  - Has Banana

Monkey & Banana

- Initial State:
  - Monkey is at the door
  - Monkey is on floor
  - Box is at window
  - Monkey does not have banana
- In prolog:
  - state(atdoor, onfloor, atwindow, hasnot).

functor

Monkey & Banana

- Goal:
  - state(_, _, _, has).

Anonymous Variables

Monkey & Banana

- Allowed Moves:
  - Grasp banana
  - Climb box
  - Push box
  - Walk around
- Not all moves are possible in every possible state of the world e.g. grasp is only possible if the monkey is standing on the box directly under the banana and does not have the banana yet.

Monkey & Banana

- Move from one state to another
- In prolog:
  - move(State1, Move, State2)

Grasp
Climb
Push
Walk

Monkey & Banana

- Grasp
  - move(state(middle, onbox, middle, hasnot),
    grasp,
    state(middle, onbox, middle, has)).

Monkey & Banana

- Climb
  - move(state(P, onfloor, P, H),
    climb,
    state(P, onbox, P, H)).

Monkey & Banana

- Push
  - move(state(P1, onfloor, P1, H),
    push(P1, P2),
    state(P2, onfloor, P2, H)).

Monkey & Banana

- Walk
  - move(state(P1, onfloor, B, H),
      walk(P1, P2),
      state(P2, onfloor, B, H)).

Monkey & Banana

- Main question our program will pose:
  - Can the monkey in some initial state get the banana?
- Prolog predicate:
  - canget(State)

Monkey & Banana

- Canget(State)
  - (1) For any state in which the monkey already has the banana the predicate is true
  - canget(state(_, _, _, has)).

Monkey & Banana

- Canget(State)
  - (2) In other cases one or more moves are necessary. The monkey can get the banana in any state (State1) if there is some move (Move) from State1 to some state (State2), such that the monkey can get the banana in State2 (in zero or more moves).
  - canget(State1) :-

    move(State1, Move, State2),

    canget(State2).

Monkey & Banana

- Questions:
  - ?- canget(state(atwindow, onfloor, atwindow, has)).
  - Yes
  - ?- canget(state(atdoor, onfloor, atwindow, hasnot)).
  - Yes
  - ?- canget(state(atwindow, onbox, atwindow, hasnot)).
  - No

Monkey & Banana

- Clause Order
  - Grasp
  - Climb
  - Push
  - Walk
- Effectively says that the monkey prefers grasping to climbing, climbing to pushing etc...
- This order of preferences helps the monkey to solve the problem.

Monkey & Banana

- Reorder Clauses
  - Walk
  - Grasp
  - Climb
  - Push
- This results in an infinite loop!
  - As the first move the monkey chooses will always be move, therefore he moves aimlessly around the room.

Monkey & Banana

- Conclusion:
  - A program in Prolog may be declaratively correct, but procedurally incorrect.
  - i.e. Unable to find a solution when a solution actually exists.
- However, there are methods that solve this problem.