Part 1 The Prolog Language

Chapter 1
Introduction to Prolog

PROLOG

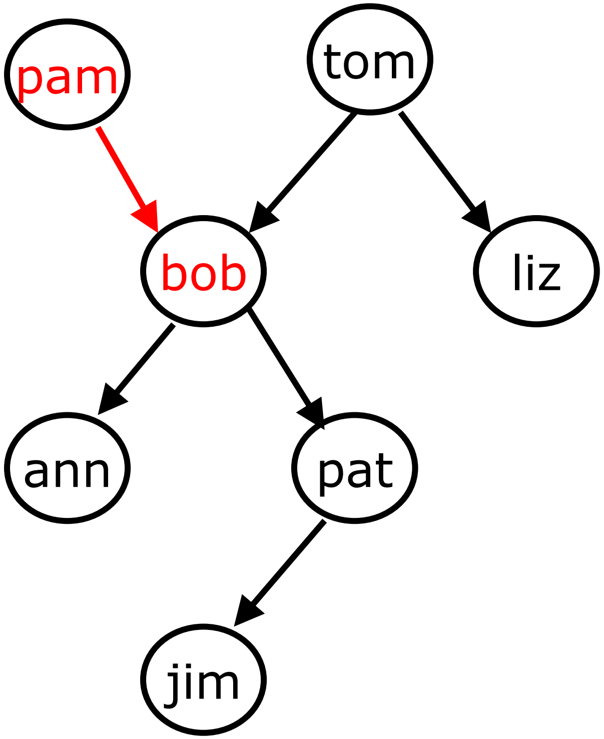
Prolog adalah bahasa pemrograman yang berpusat di sekitar satu set mekanisme dasar, termasuk pencocokan pola, penataan data tree-based, dan automatic backtracking

Prolog cocok untuk menyelesaikan masalah yang berurusan dengan object, khususnya objek terstruktur dan hubungan antara mereka. Karena prolog memiliki fungsi dalam logika matematika, prolog sering diperkenalkan melalui logika logic.

Prolog adalah bahasa pemrograman untuk perhitungan simbolis non-numerik yang sangat cocok untuk memecahkan masalah yang melibatkan objek dan hubungan antar objek

1.1 Defining relations by facts

- Given a whole family tree

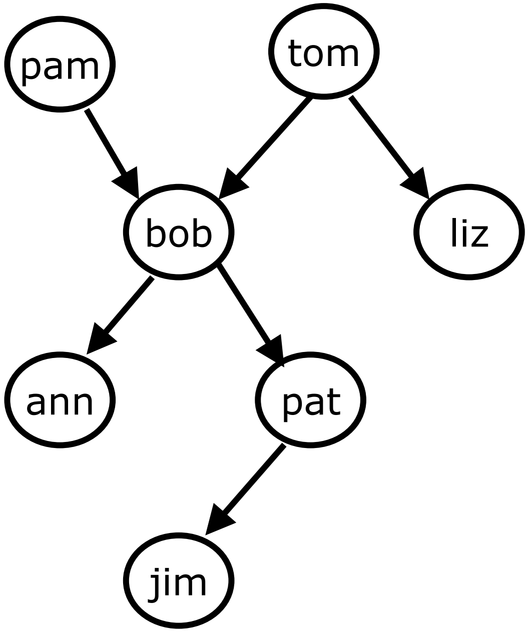


- The tree defined by the Prolog program:

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

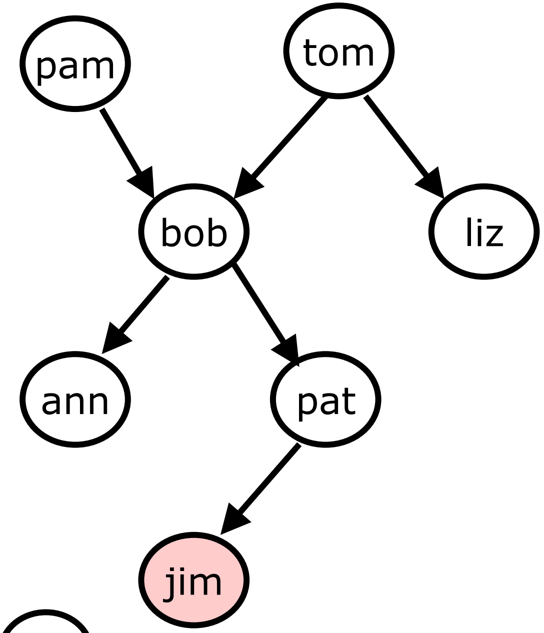
1.1 Defining relations by facts

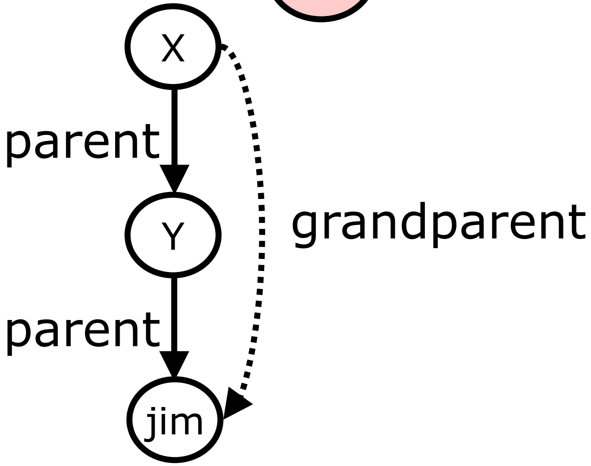
- Questions:
  - Is Bob a parent of Pat?
    - ?- parent(bob, pat).
    - ?- parent(liz, pat).
    - ?- parent(tom, ben).
  - Who is Liz’s parent?
    - ?- parent(X, liz).
  - Who are Bob’s children?
    - ?- parent(bob, X).



1.1 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).



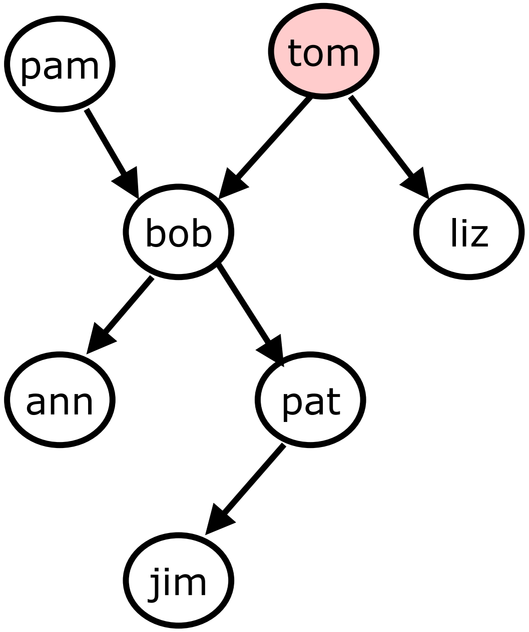


1.1 Defining relations by facts

- 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).
      ```

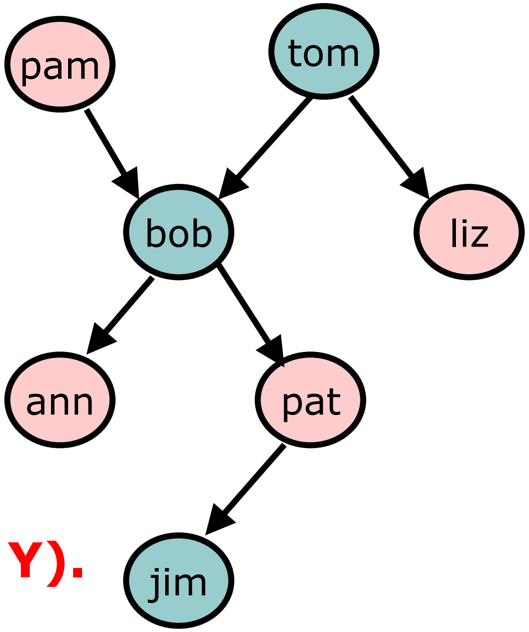


1.1 Defining relations by facts

- It is easy in Prolog to define a relation.
- The user can easily query the Prolog system about relations defined in the program.
- A Prolog program consists of clauses. Each clause terminates with a full stop.
- The arguments of relations can be
  - Atoms: concrete objects (實物) or constants (常數)
  - Variables: general objects such as X and Y
- Questions to the system consist of one or more goals.
- An answer to a question can be either positive (succeeded) or negative (failed).
- If several answers satisfy the question then Prolog will find as many of them as desired by the user.

1.2 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).

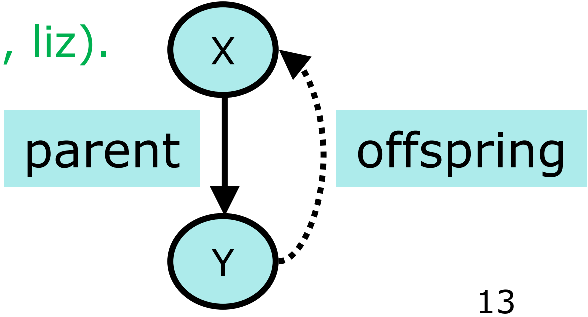


- Define the "offspring(子女)" relation:
  - Fact: offspring(liz, tom).
  - Rule: **offspring(Y, X) :- parent(X, Y).**
    - For all X and Y,
      Y is an offspring of X if
      X is a parent of Y.

1.2 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

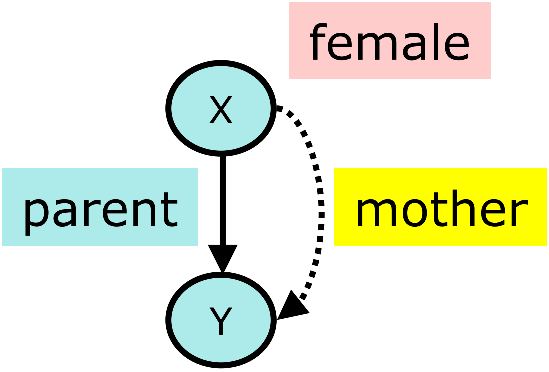
  - Example:
    - **offspring(Y, X) :- parent(X, Y).**
    - The rule is general in the sense that it is applicable to any objects X and Y.
    - A special case of the general rule:
      - offspring(liz, tom) :- parent(tom, liz).
    - ?- offspring(liz, tom).
    - ?- offspring(X, Y).



1.2 Defining relations by rules

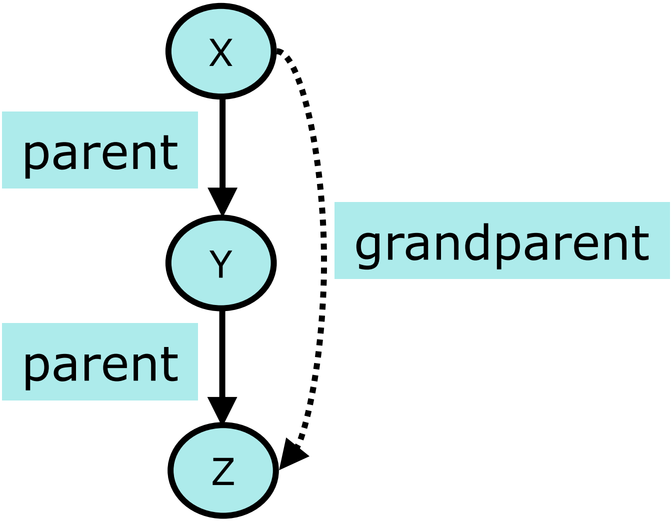
- Define the “mother” relation:
  - **mother(X, Y) :- parent(X, Y), female(X).**
  - For all X and Y,

    X is the mother of Y if
    X is a parent of Y and
    X is a female.



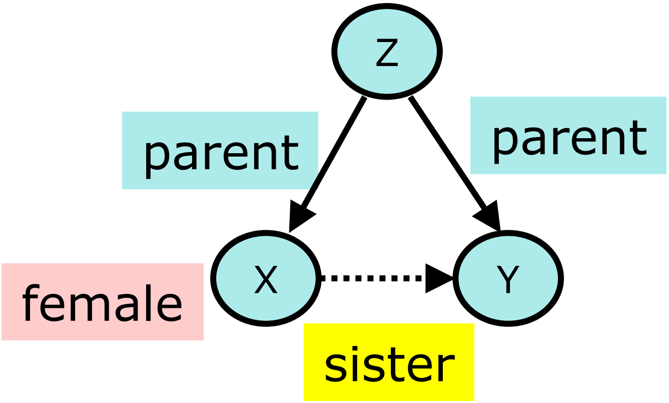
1.2 Defining relations by rules

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



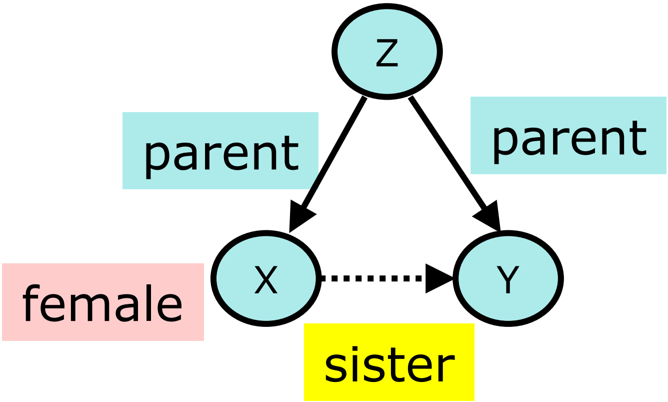
1.2 Defining relations by rules

- 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?!



1.2 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)



1.2 Defining relations by rules

- 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

1.2 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,

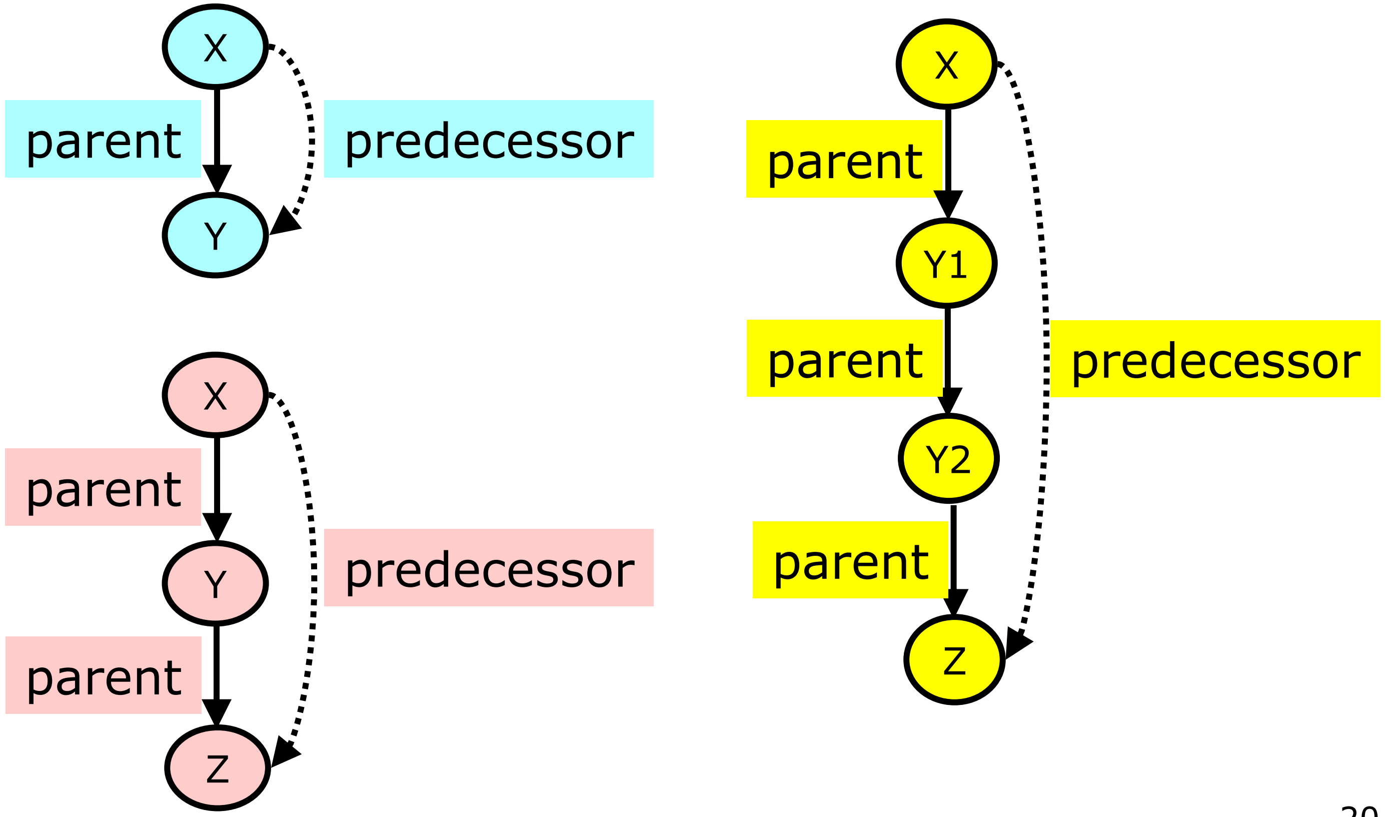
    if X is a parent of Y then X has a child.

    (b) For all X,

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

1.3 Recursive rules

- Define the “predecessor(祖先)” relation

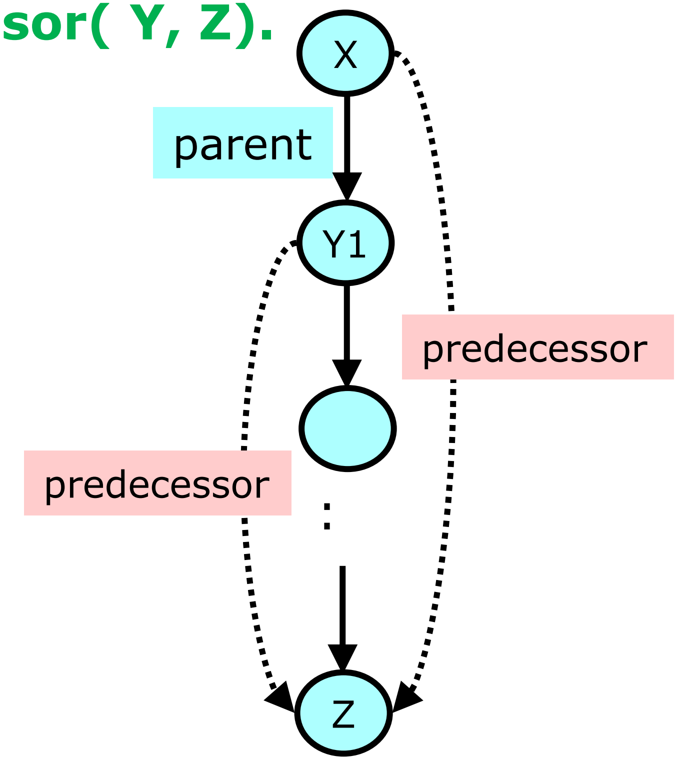


1.3 Recursive rules

- 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).



1.3 Recursive rules

```
% Figure 1.8   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).
```

1.3 Recursive rules

- Procedure:
  - In figure 1.8, there are two “predecessor relation” clauses.

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

  - Such a set of clauses is called a **procedure**.

- Comments:

  ```
  /* This is a comment */
  % This is also a comment
  ```

Trace and Notrace

```
| ?- trace.
The debugger will first creep -- showing everything
     (trace)

(15 ms) yes
{trace}

| ?- predecessor( X, Z).
      1    1  Call: predecessor(_16,_17) ?
      2    2  Call: parent(_16,_17) ?
      2    2  Exit: parent(pam,bob) ?
      1    1  Exit: predecessor(pam,bob) ?

X = pam
Z = bob ? ;
      1    1  Redo: predecessor(pam,bob) ?
      2    2  Redo: parent(pam,bob) ?
      2    2  Exit: parent(tom,bob) ?
      1    1  Exit: predecessor(tom,bob) ?

X = tom
Z = bob ? ;
```

```
…

X = bob
Z = jim
      1    1  Redo: predecessor(bob,jim) ?
      3    2  Redo: predecessor(pat,jim) ?
      4    3  Call: parent(pat,_144) ?
      4    3  Exit: parent(pat,jim) ?
…
      4    3  Fail: parent(jim,_17) ?
      4    3  Call: parent(jim,_144) ?
      4    3  Fail: parent(jim,_132) ?
      3    2  Fail: predecessor(jim,_17) ?
      1    1  Fail: predecessor(_16,_17) ?

(266 ms) no
{trace}

| ?- notrace.
The debugger is switched off

yes
```

1.4 How Prolog answers questions

- To answer a question, Prolog tries to satisfy all the goals.
- To satisfy a goal means to demonstrate that the goal is true, assuming that the relations in the program is true.
- Prolog accepts facts and rules as a set of axioms(公理), and the user’s question as a conjectured (推測的) theorem(定理).
- Example:
  - Axioms: All men are fallible (會犯錯的).
    Socrates is a man.
  - Theorem: Socrates is fallible.
  - For all X, if X is a man then X is fallible.

```
fallible( X) :- man( X).
man( socrates).
```

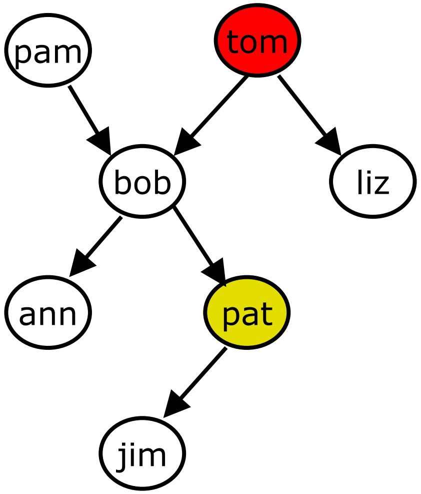
    - ?- fallible(socrates).

1.4 How Prolog answers 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).



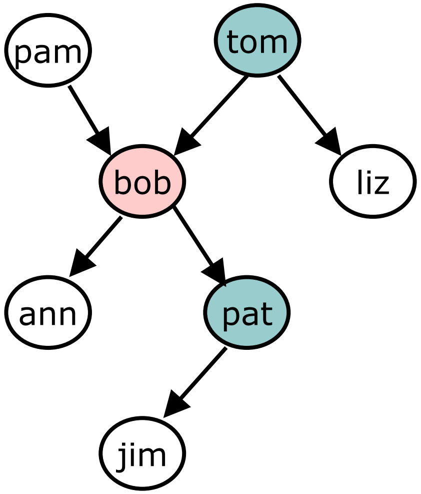
1.4 How Prolog answers questions

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

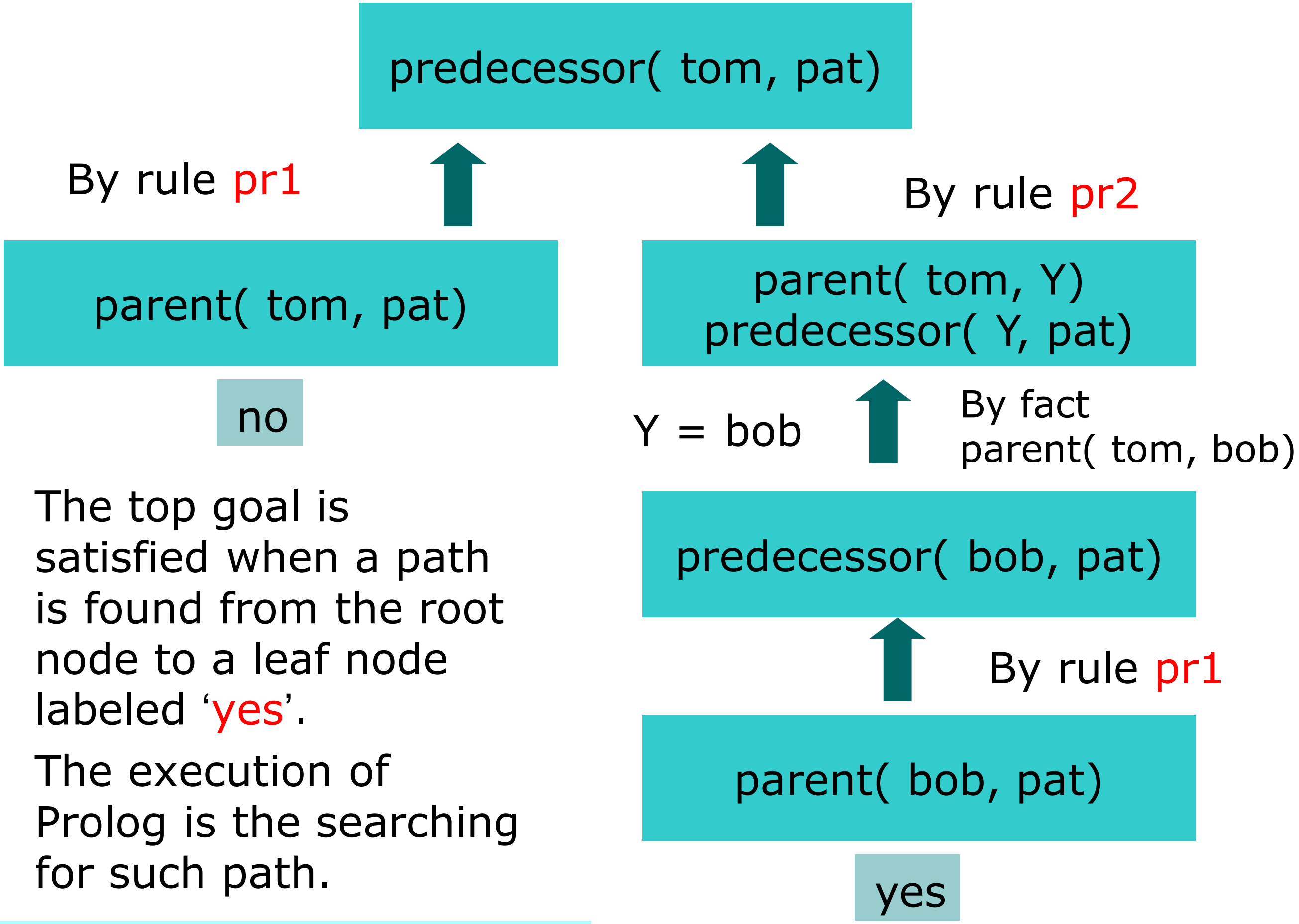
```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

- ?- 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)



1.4 How Prolog answers questions



predecessor(tom, pat)

By rule pr1

parent(tom, pat)

no

By rule pr2

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

Y = bob

By fact
parent(tom, bob)

predecessor(bob, pat)

By rule pr1

parent(bob, pat)

yes

- The top goal is satisfied when a path is found from the root node to a leaf node labeled ‘yes’.
- The execution of Prolog is the searching for such path.

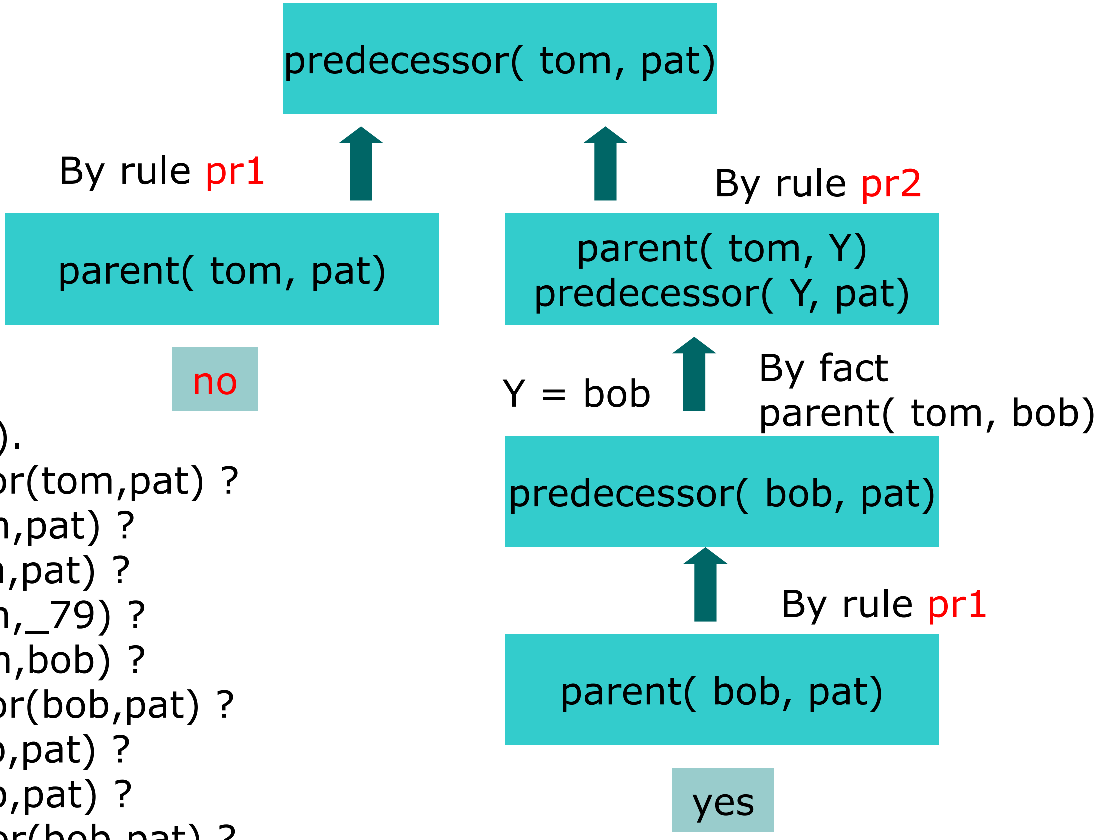
```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
predecessor( X, Z)  :- parent( X, Z).          % Rule pr1
predecessor( X, Z)  :- parent( X, Y),          % Rule pr2
                       predecessor( Y, Z).
```

derivation diagrams

Trace

```
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).
      1     1   Call: predecessor(tom,pat) ?
      2     2   Call: parent(tom,pat) ?
      2     2   Fail: parent(tom,pat) ?
      2     2   Call: parent(tom,_79) ?
      2     2   Exit: parent(tom,bob) ?
      3     2   Call: predecessor(bob,pat) ?
      4     3   Call: parent(bob,pat) ?
      4     3   Exit: parent(bob,pat) ?
      3     2   Exit: predecessor(bob,pat) ?
      1     1   Exit: predecessor(tom,pat) ?

true ?
```

derivation diagrams