UCS1524 – Logic Programming

Introduction to Prolog

Session Meta Data

Author	**Dr. D. Thenmozhi**
Reviewer	
Version Number	1.2
Release Date	20 August 2022

Session Objectives

- Understanding Prolog language with simple hands-on.
- Learn about the usage of Prolog language by specifying relations by facts and rules.

Session Outcomes

- At the end of this session, participants will be able to
  - Develop Prolog programs using facts rules and relations.

Agenda

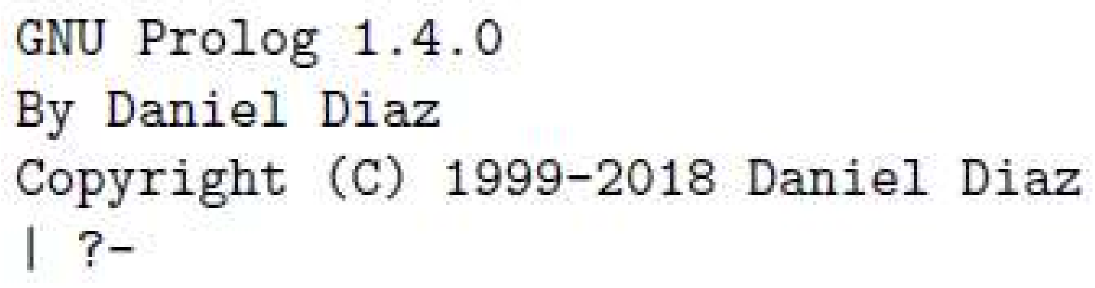
- Prolog Interpreter
- Simple commands in Prolog
- Defining relations by facts
- Defining relation by rules
- Querying

GNU Prolog -GProlog

- This Prolog compiler complies with the ISO standard for Prolog (with useful extensions like global variables, ability to interface with the operating system, etc) and produces a native binary that can be run standalone.
- It is smart enough to avoid linking unused built-in predicates.
- It also has an interactive interpreter and a Prolog debugger as well as a low-level WAM debugger.
- You can interface with C code (both ways). Platforms supported include Linux (i86), SunOS (sparc) and Solaris (sparc).
  - http://www.thefreecountry.com/compilers/prolog.shtml (Free Prolog Compilers and Interpreters)
  - http://www.gprolog.org/ (The GNU Prolog web site)
  - http://www.thefreecountry.com/documentation/onlineprolog.shtml (Online Prolog Tutorials)

GProlog

- Download Gprolog from http://www.gprolog.org/
- After installation



```
GNU Prolog 1.4.0
By Daniel Diaz
Copyright (C) 1999-2018 Daniel Diaz
| ?-
```

- This is known as top-level interpreter.
- It allows the user to execute queries, to consult Prolog programs, to list them, to execute them and to debug them.

Simple commands

- | ?- append([a,b],[c,d],X).
- | ?- append(X,Y,[a,b,c]). (to get next solution press ;)
- | ?- (X=1 ; X=2).
- | ?- L= [1,2,3,4], member(X, L).
- | ?- L= [a, b, c], length(L, X).
- | ?- X is 3 + 4.
- | ?- X = 3 + 4.

Consult Prolog source files.

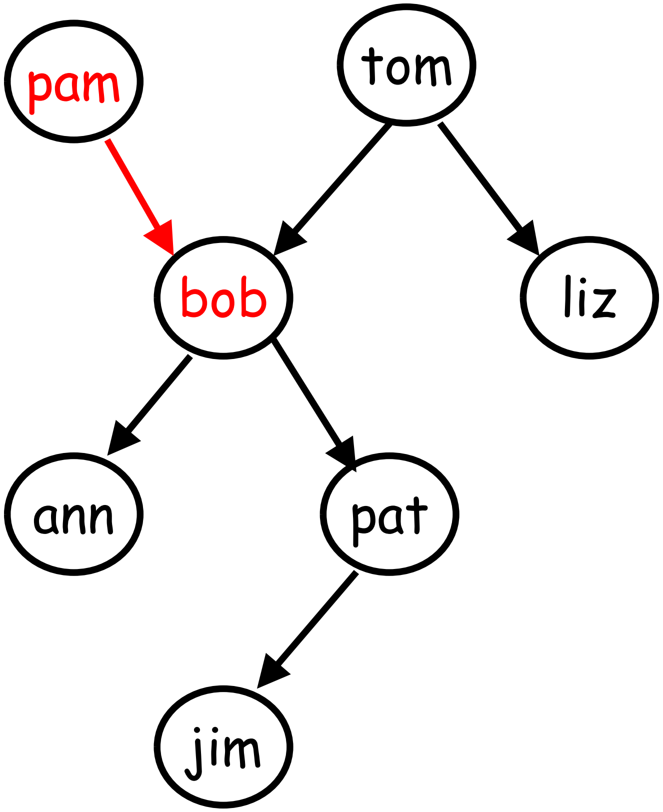
- The top-level allows the user to consult Prolog source files.
- To directly input the predicates from the terminal by the user, use “[user]”

-
  ```
  | ?- [user].
  {compiling user for byte code...}
  even(0).
  even(s(s(X))):-
                even(X).
  Ctrl D
  ```

-
  ```
  | ?- even(X).
  ```

Defining relations by facts

- Given a whole family tree

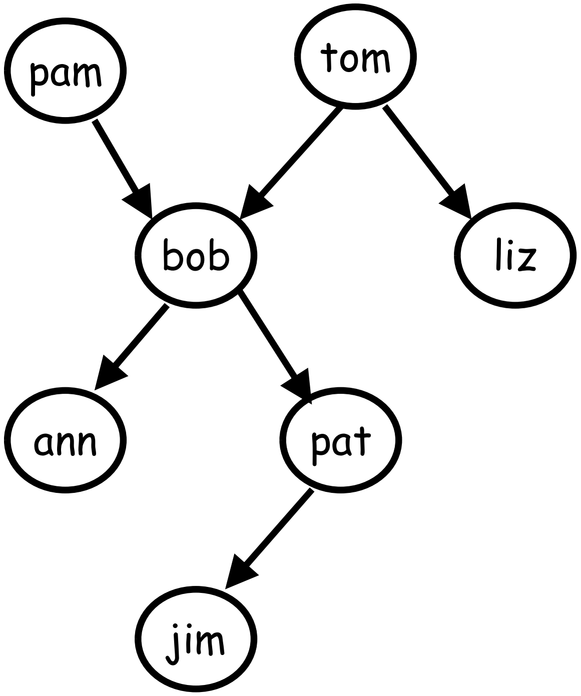


- The tree defined by the Prolog program:

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

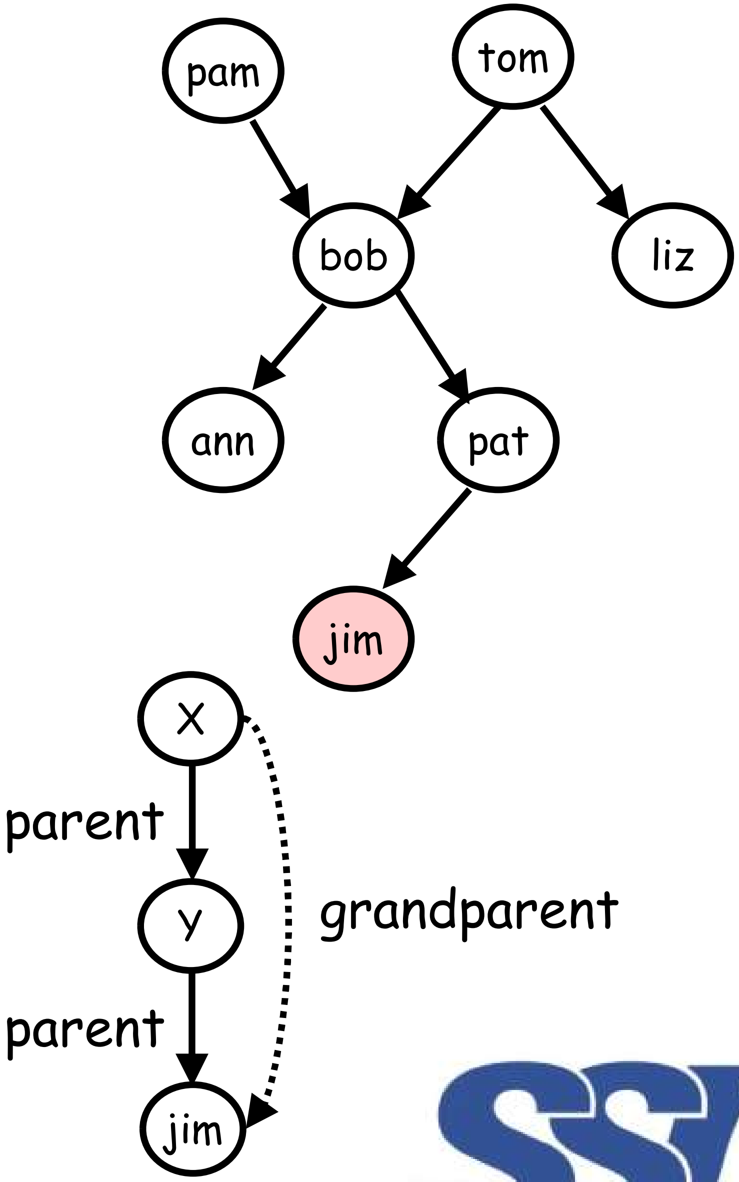
Defining relations by facts

- 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).



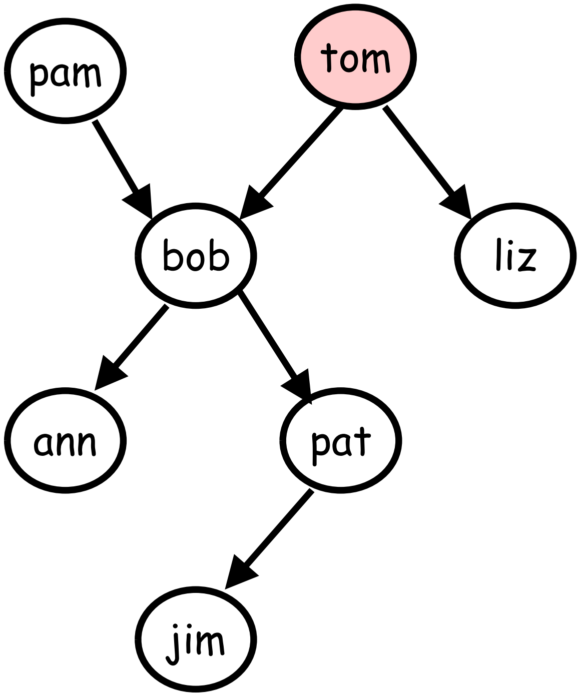
Defining relations by facts

- 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).

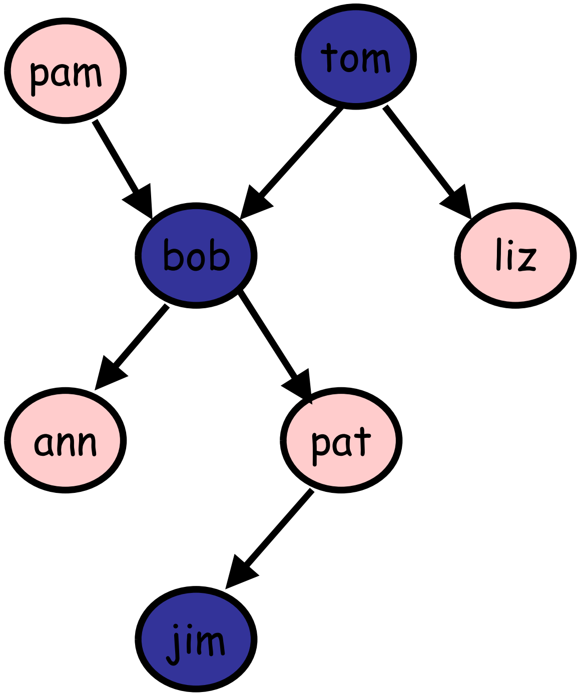


Defining relations by facts

- 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 or constants (tom, ann)
  - 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.

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).

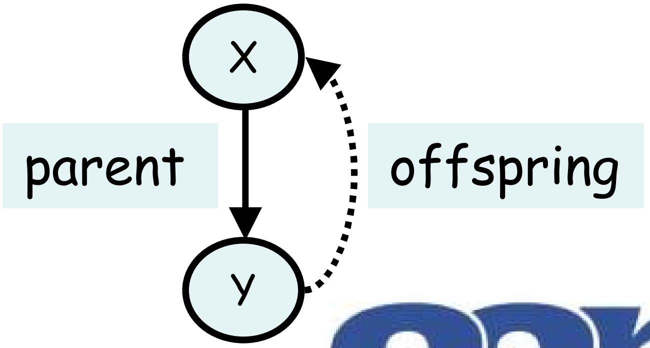


Inverse of parent

- 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).



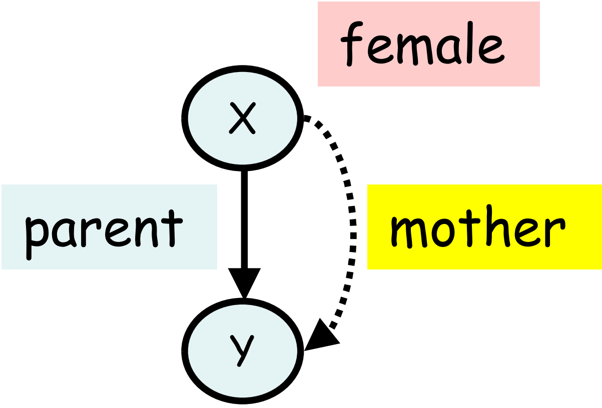
Defining relations by rules

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

    X is the mother of Y if

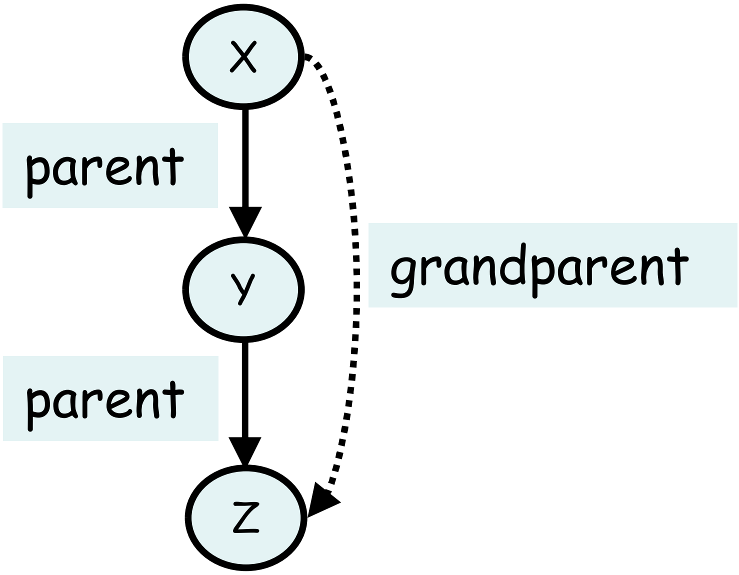
    X is a parent of Y and

    X is a female.



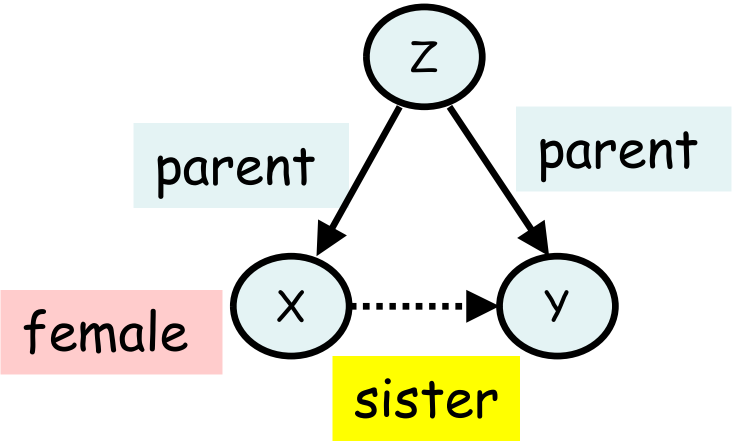
Defining relations by rules

- 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?!

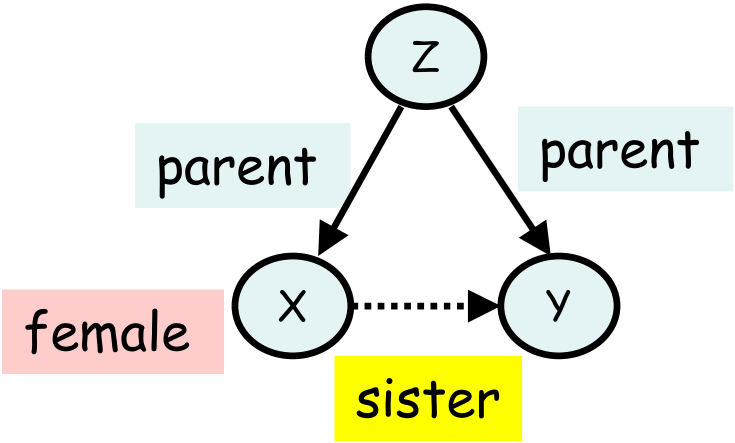


Defining relations by rules

- 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

- 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

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,

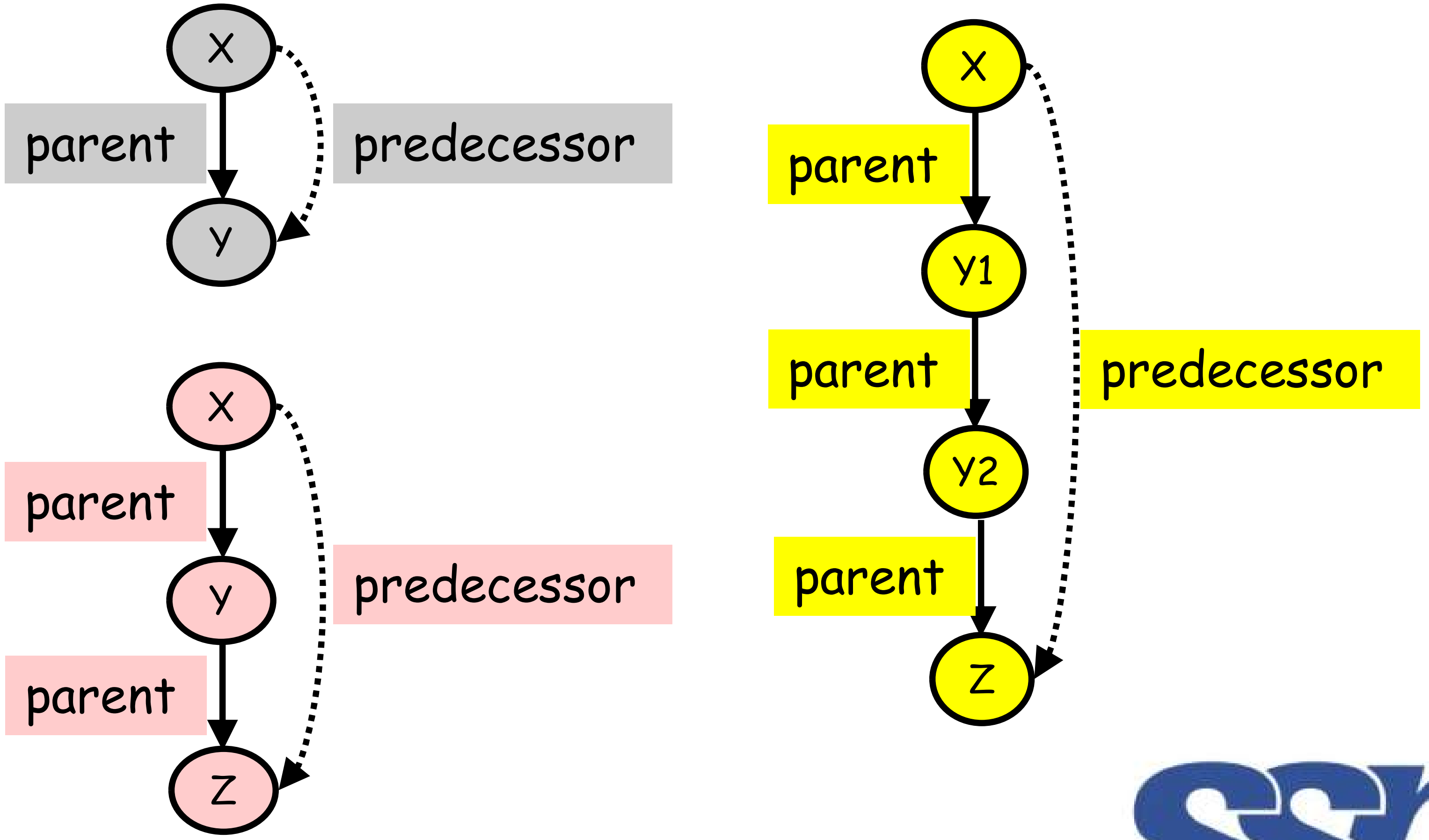
    if X is a parent of Y then X has a child.

    (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



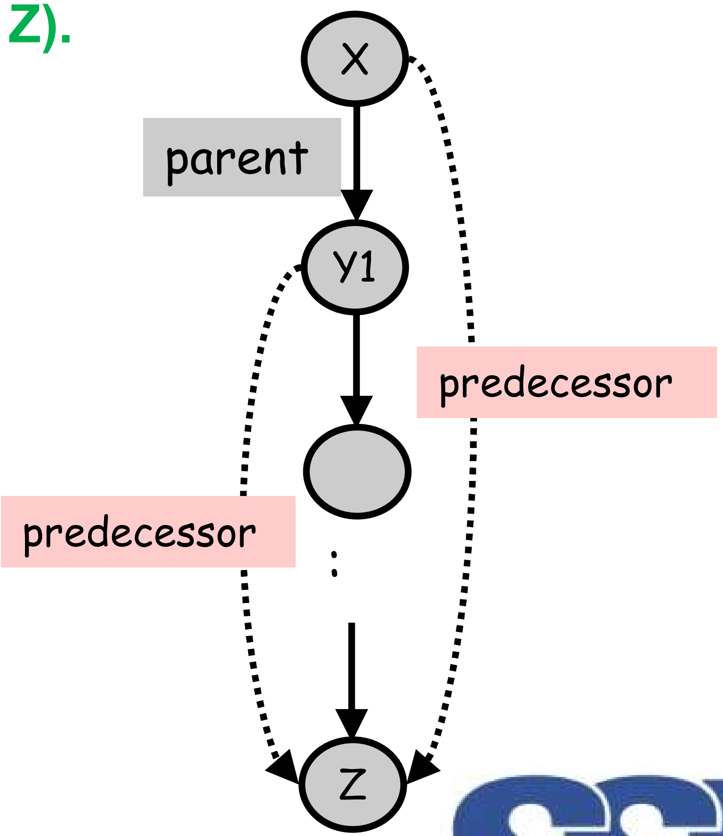
Recursive rules

- 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).
```

Recursive rules

- Procedure:
  - 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
```

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, and the user’s question as a conjectured theorem.
- Example:
  - Axioms: All men are fallible.
    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).

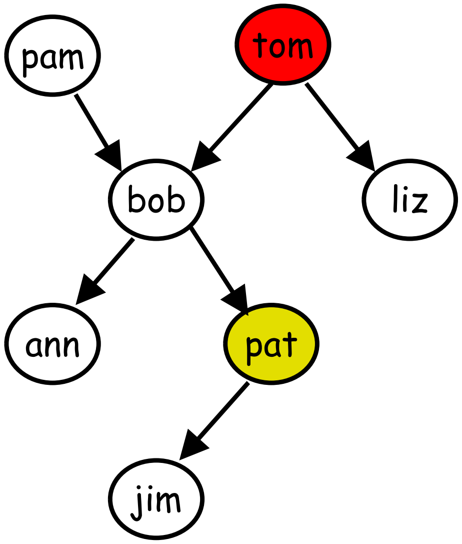
    - ?- fallible(socrates).

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).



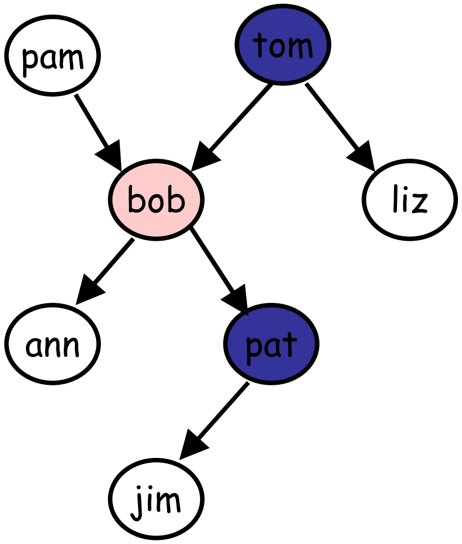
How Prolog answers questions

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

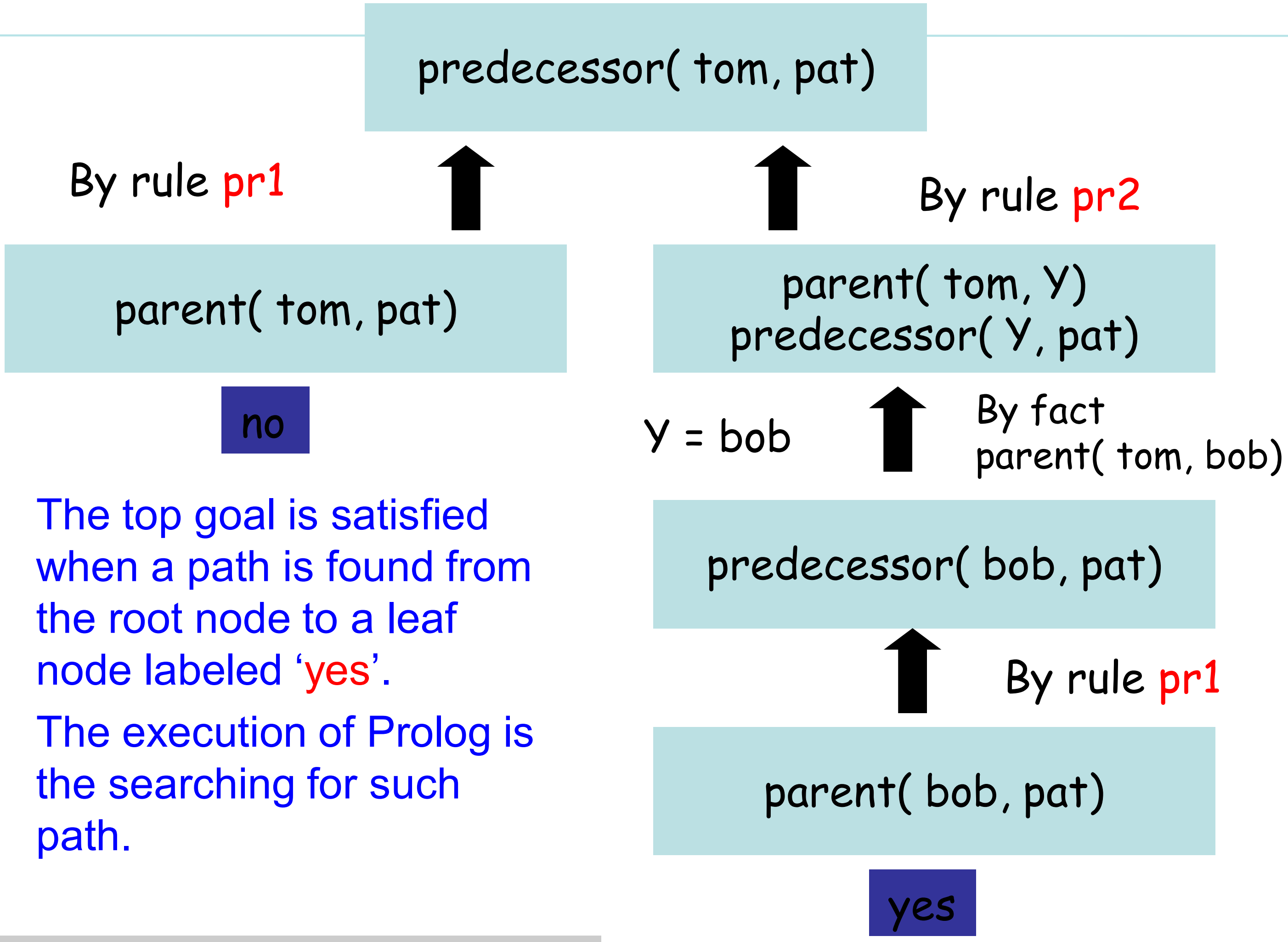
```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

- ?- 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



- 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.

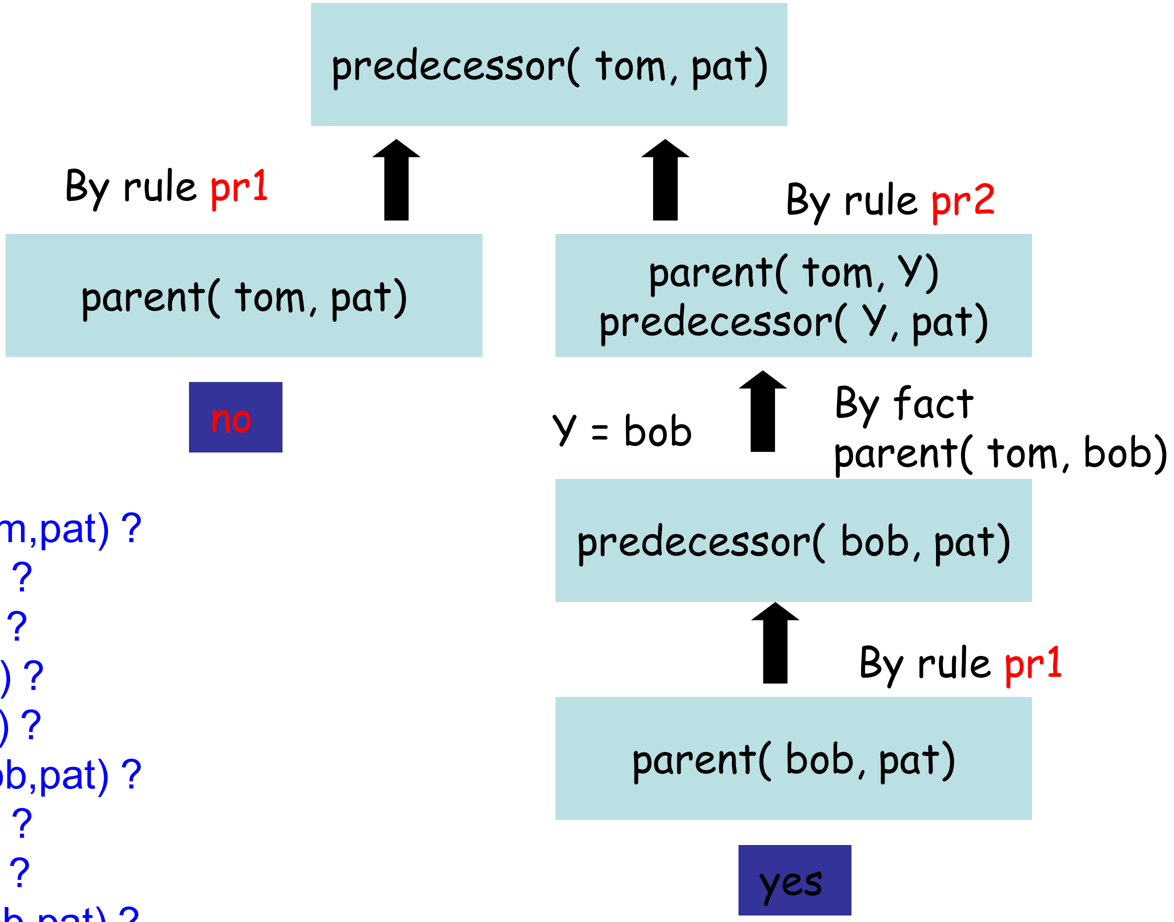
```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
predecessor( X, Z)  :- parent( X, Z).              % Rule pr1
predecessor( X, Z)  :- parent( X, Y),              % Rule pr2
                       predecessor( Y, Z).
```

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

Declarative and procedural meaning of programs

- Two levels of meaning of Prolog programs:
  - The declarative meaning
    - concerned only with the relations defined by the program
    - determines what will be the output of the program
    - The programmer should concentrate mainly on the declarative meaning and avoid being distracted by the executional details.
  - The procedural meaning
    - determines how this output is obtained
    - determines how the relations are actually evaluated by the Prolog system
    - The procedural aspects cannot be completely ignored by the programmer for practical reasons of executional efficiency.

Summary

- Prolog Interpreter
  - GNU interpreter
- Simple commands in Prolog
  - Based on GNU Prolog tutorial
- Defining relations by facts
- Defining relation by rules
  - Recursive rules
- Querying
  - How Prolog answers for the questions
  - Trace and notrace

Check your understanding

- Let set of facts be
  - Henry is a male
  - Tom is a male
  - Jack is a male
  - Anna is a female
  - Janis is a female
  - Tom is married
  - Anna is married
- Use Gprolog, find the answers for the queries
  - List the males
  - List the females
  - Check whether Tom is male
  - Check whether Tom is female

Check your understanding

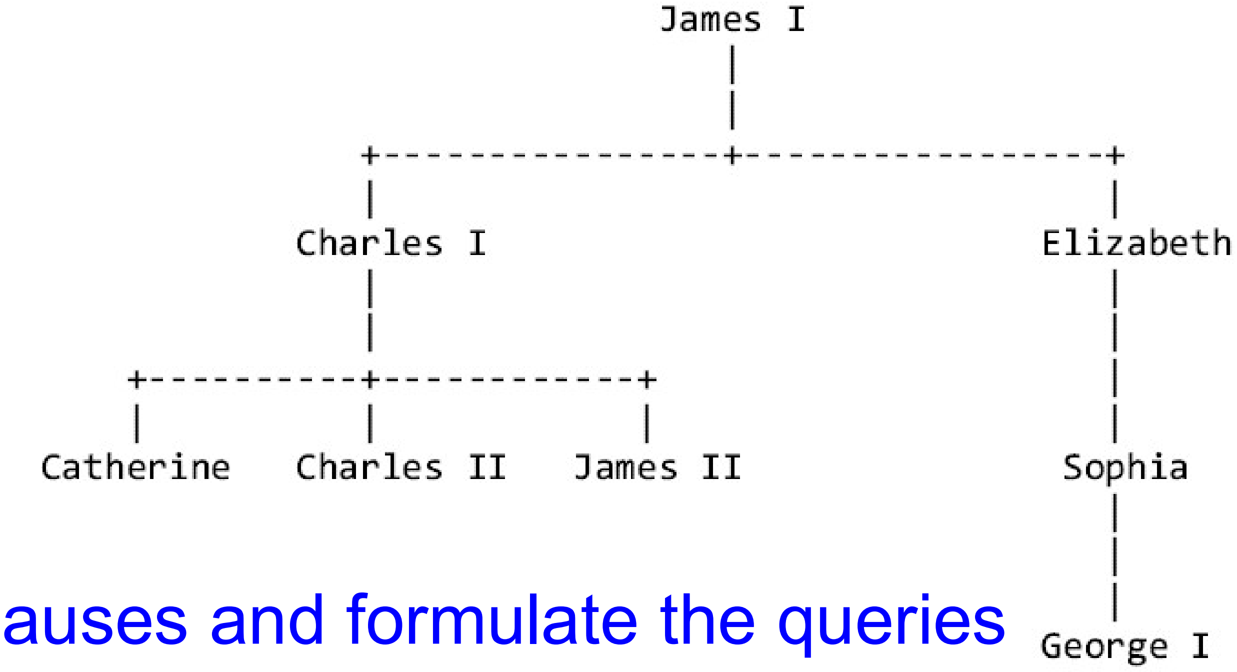
- Let set of facts be
  - Henry is a male
  - Tom is a male
  - Jack is a male
  - Anna is a female
  - Janis is a female
  - Tom is married
  - Anna is married
- Rule to define a bachelor is
  - If a person is male and not married then he is a bachelor.
- Use Gprolog, find the answers for the queries
  - Whether henry is a bachelor
  - Whether tom is a bachelor
  - List the set of bachlors

Check your understanding

- Consider the family tree and given male and female clauses

```
male(james1).
male(charles1).
male(charles2).
male(james2).
male(george1).

female(catherine).
female(elizabeth).
female(sophia).
```



```
                                   James I
                                      |
                                      |
                     +----------------+----------------+
                     |                                 |
                  Charles I                        Elizabeth
                     |                                 |
                     |                                 |
          +----------+-----------+                     |
          |          |           |                     |
      Catherine  Charles II   James II               Sophia
                                                       |
                                                       |
                                                       |
                                                   George I
```

- Define parent clauses and formulate the queries

```
Was George I the parent of Charles I?

Who was Charles I's parent?

Who were the children of Charles I?
```

- Define father and uncle relations