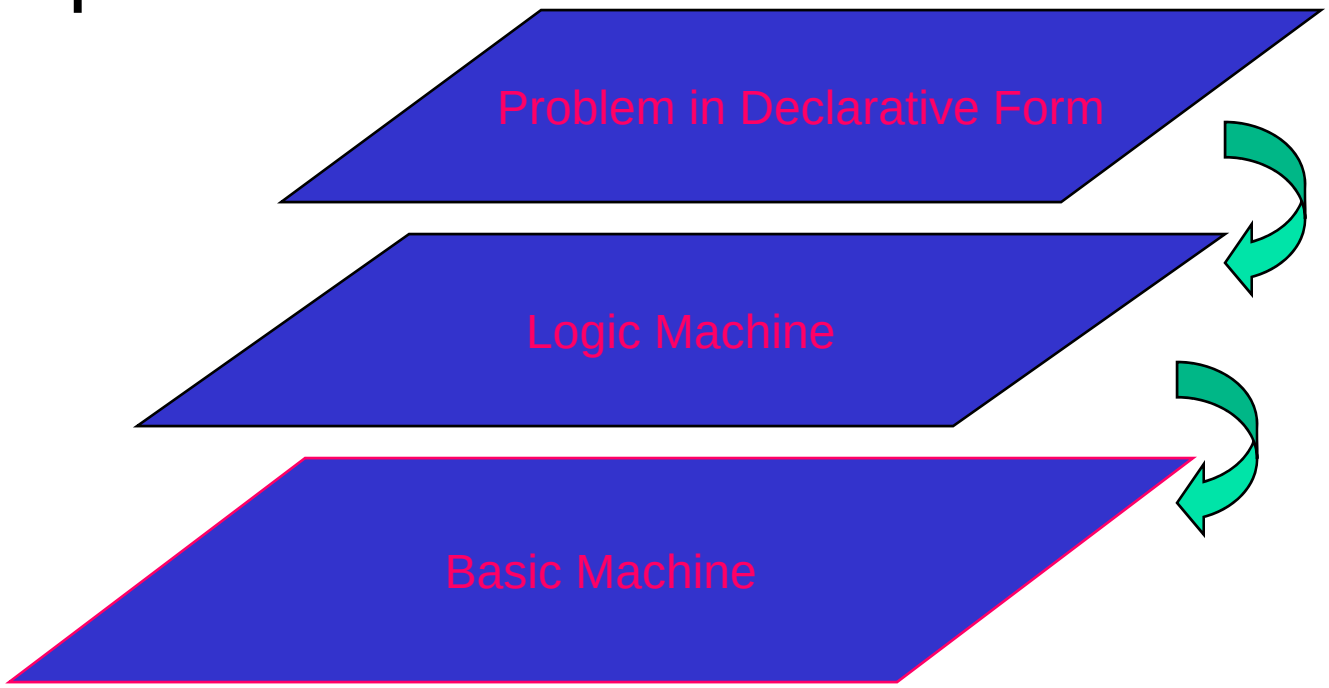
# CS344: Introduction to Artificial Intelligence

Pushpak Bhattacharyya
CSE Dept.,
IIT Bombay

Lecture 11–Prolog

# Introduction

- PROgramming in LOGic
- Emphasis on *what* rather than *how*



# Prolog's strong and weak points

- Assists thinking in terms of *objects* and *entities*
- Not good for *number crunching*
- Useful applications of Prolog in
  - *Expert Systems* (Knowledge Representation and Inferencing)
  - *Natural Language Processing*
  - *Relational Databases*

# A Typical Prolog program

*Compute_length ([],0).*

*Compute_length ([Head|Tail], Length):-*

*Compute_length (Tail,Tail_length),*

*Length is Tail_length+1.*

High level explanation:

*The length of a list is 1 plus the length of the tail of the list, obtained by removing the first element of the list.*

**This is a declarative description of the computation.**

# Fundamentals

*(absolute basics for writing Prolog Programs)*

# Facts

- *John likes Mary*
  - *like(john,mary)*
- Names of relationship and objects must begin with a lower-case letter.
- Relationship is written *first* (typically the *predicate* of the sentence).
- *Objects* are written separated by commas and are enclosed by a pair of round brackets.
- The full stop character '.' must come at the end of a fact.

# More facts

| **Predicate** | **Interpretation** |
|---|---|
| valuable(gold) | Gold is valuable. |
| owns(john,gold) | John owns gold. |
| father(john,mary) | John is the father of Mary |
| gives (john,book,mary) | John gives the book to Mary |

# Questions

- *Questions* based on facts
- Answered by *matching*

Two facts *match* if their predicates are same (spelt the same way) and the arguments each are same.

- If matched, prolog answers *yes*, else *no*.
- *No* does not mean falsity.

# Prolog does *theorem proving*

- When a question is asked, prolog tries to match *transitively.*
- When no match is found, answer is *no.*
- This means *not provable* from the given facts.

# Variables

- Always begin with a capital letter
  - *?- likes (john,X).*
  - *?- likes (john, Something).*
- But *not*
  - *?- likes (john,something)*

# *Example* of usage of variable

Facts:

*likes(john,flowers).*
*likes(john,mary).*
*likes(paul,mary).*

Question:

*?- likes(john,X)*

Answer:

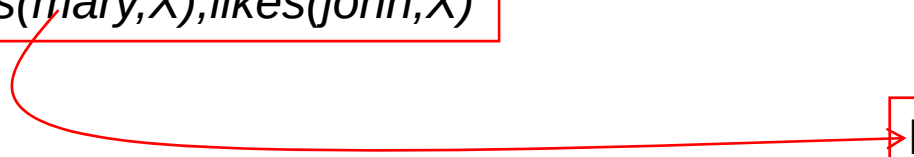
*X=flowers* and wait
*;*
*mary*
*;*
*no*

# Conjunctions

- Use ‘,’ and pronounce it as *and*.
- Example
  - Facts:
    - likes(mary,food).
    - likes(mary,tea).
    - likes(john,tea).
    - likes(john,mary)
- ?-
    - likes(mary,X),likes(john,X).
    - Meaning *is anything liked by Mary also liked by John?*

# Backtracking *(an inherent property of prolog programming)*

*likes(mary,X),likes(john,X)*

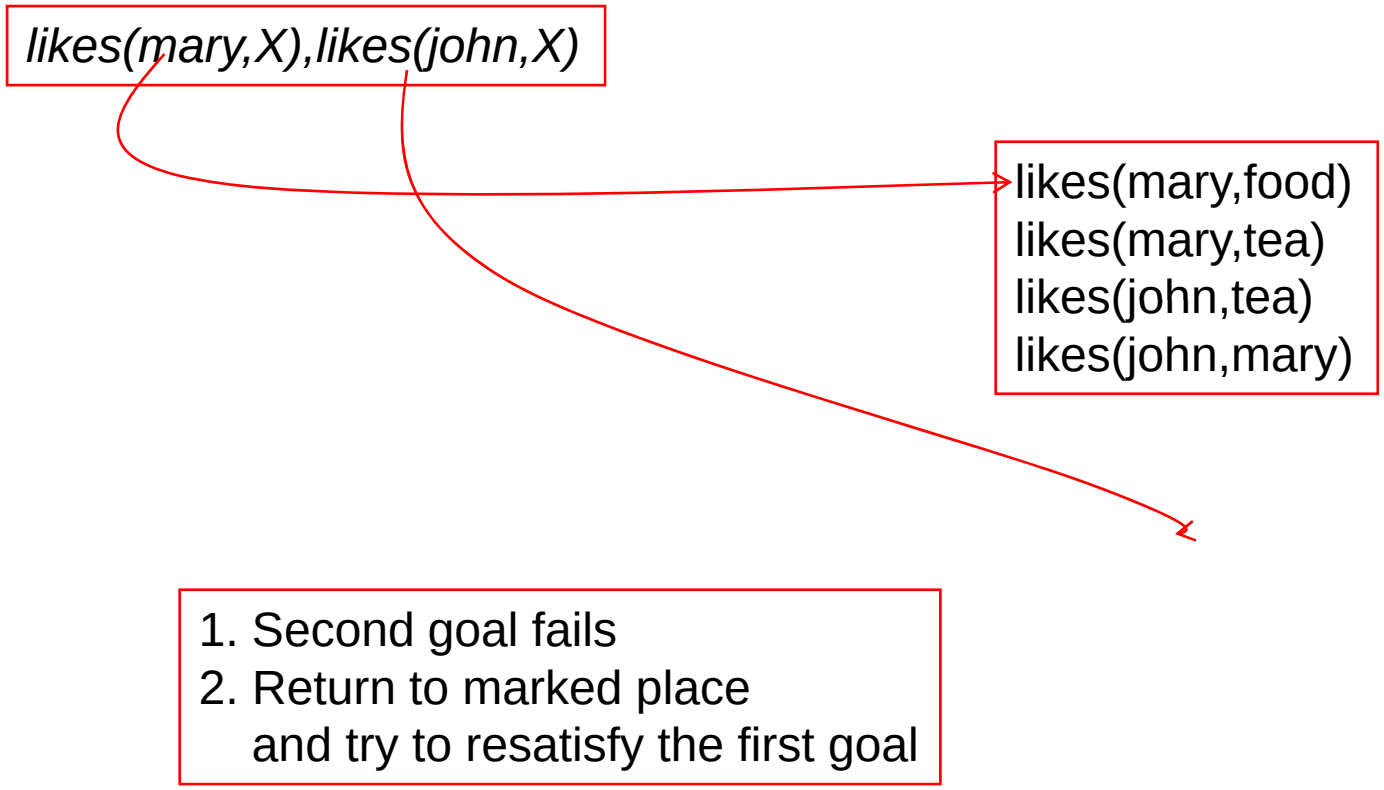


likes(mary,food)
likes(mary,tea)
likes(john,tea)
likes(john,mary)

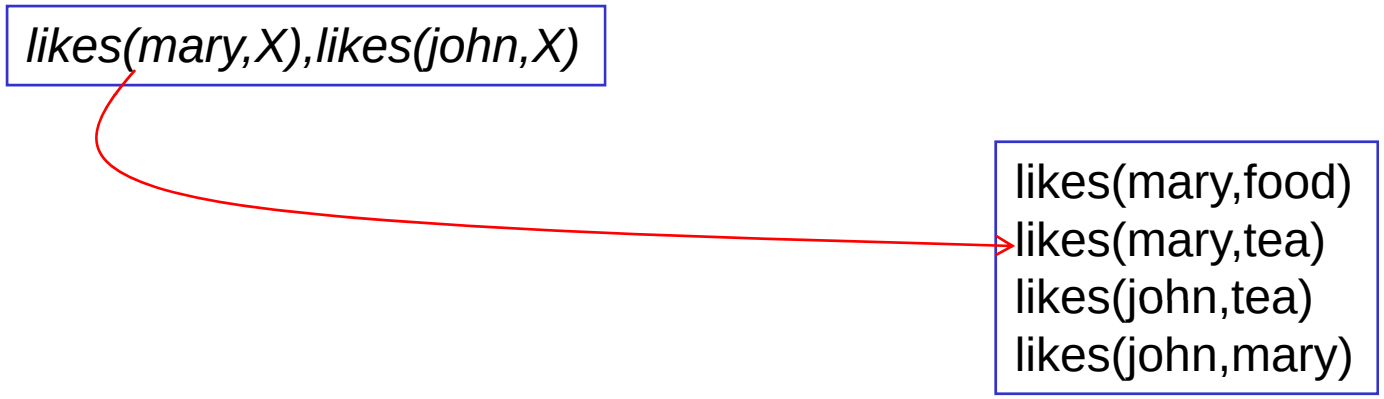
1. First goal succeeds. *X=food*
2. Satisfy *likes(john,food)*

# Backtracking *(continued)*

**Returning to a marked place and trying to resatisfy is called *Backtracking***

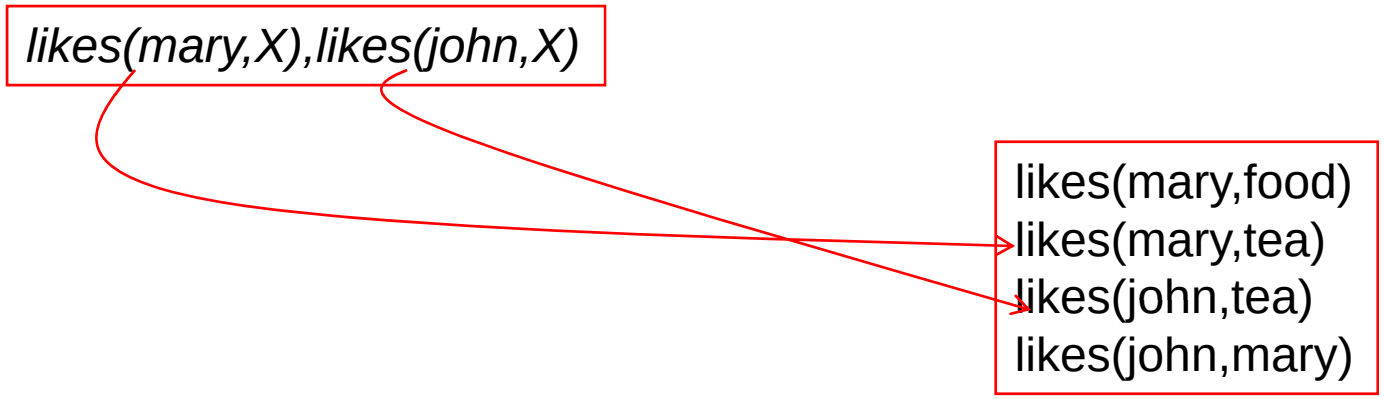


# Backtracking *(continued)*




1. First goal succeeds again, *X=tea*
2. Attempt to satisfy the *likes(john,tea)*

# Backtracking *(continued)*




1. Second goal also suceeds
2. Prolog notifies success and waits for a reply

# Rules

- Statements about *objects* and their *relationships*
- Expess
  - *If-then conditions*
    - *I use an umbrella if there is a rain*
    - *use(i, umbrella) :- occur(rain).*
  - *Generalizations*
    - *All men are mortal*
    - *mortal(X) :- man(X).*
  - *Definitions*
    - *An animal is a bird if it has feathers*
    - *bird(X) :- animal(X), has_feather(X).*

# Syntax

- **<head> :- <body>**
- Read **':-' as 'if'.**
- E.G.
  - *likes(john,X) :- likes(X,cricket).*
  - *"John likes X if X likes cricket".*
  - *i.e., "John likes anyone who likes cricket".*
- Rules always end with '.'.

# Another Example

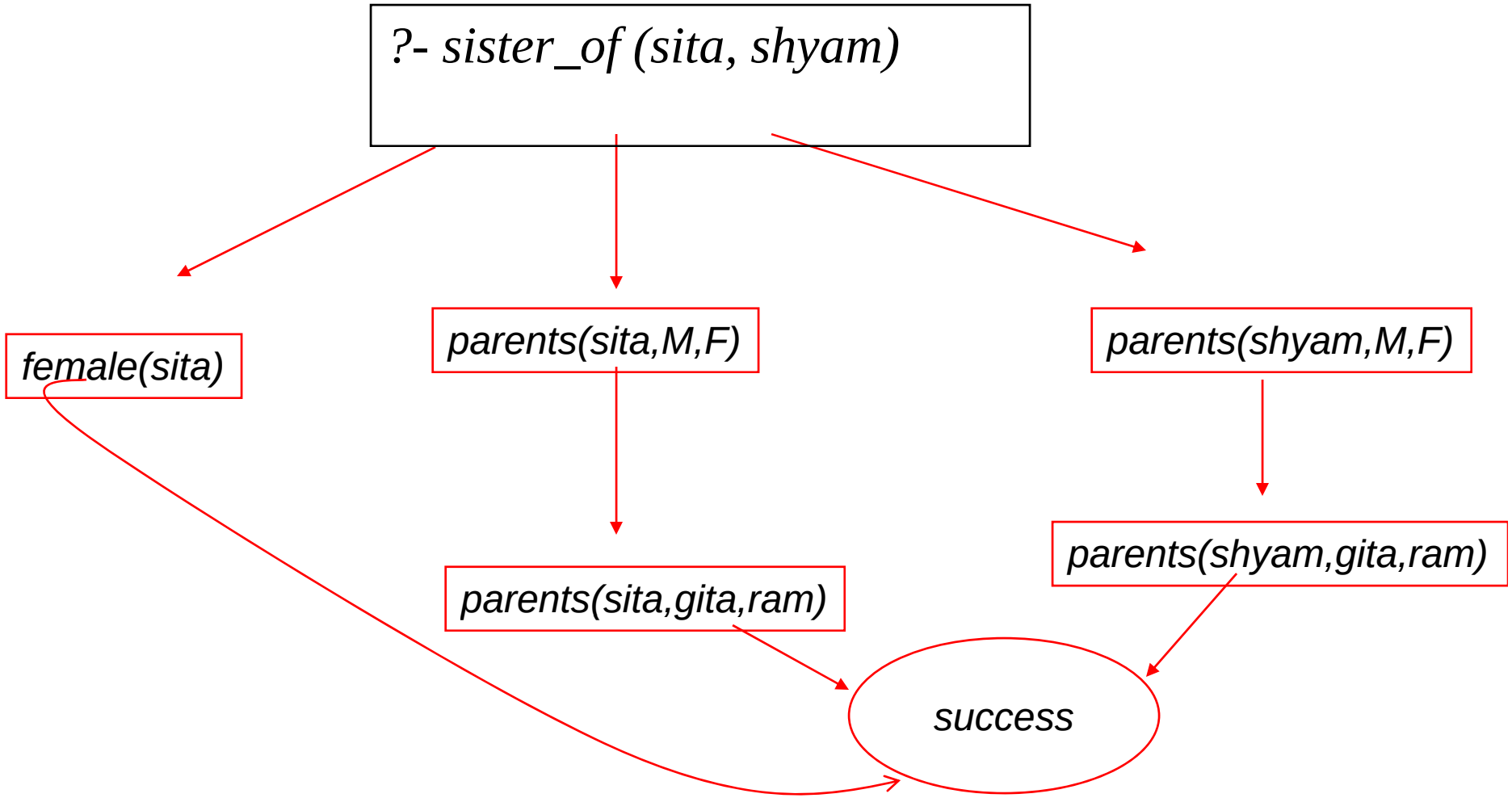
*sister_of (X,Y):- female (X),*
*parents (X, M, F),*
*parents (Y, M, F).*

*X is a sister of Y is*
*X is a female and*
*X and Y have same parents*

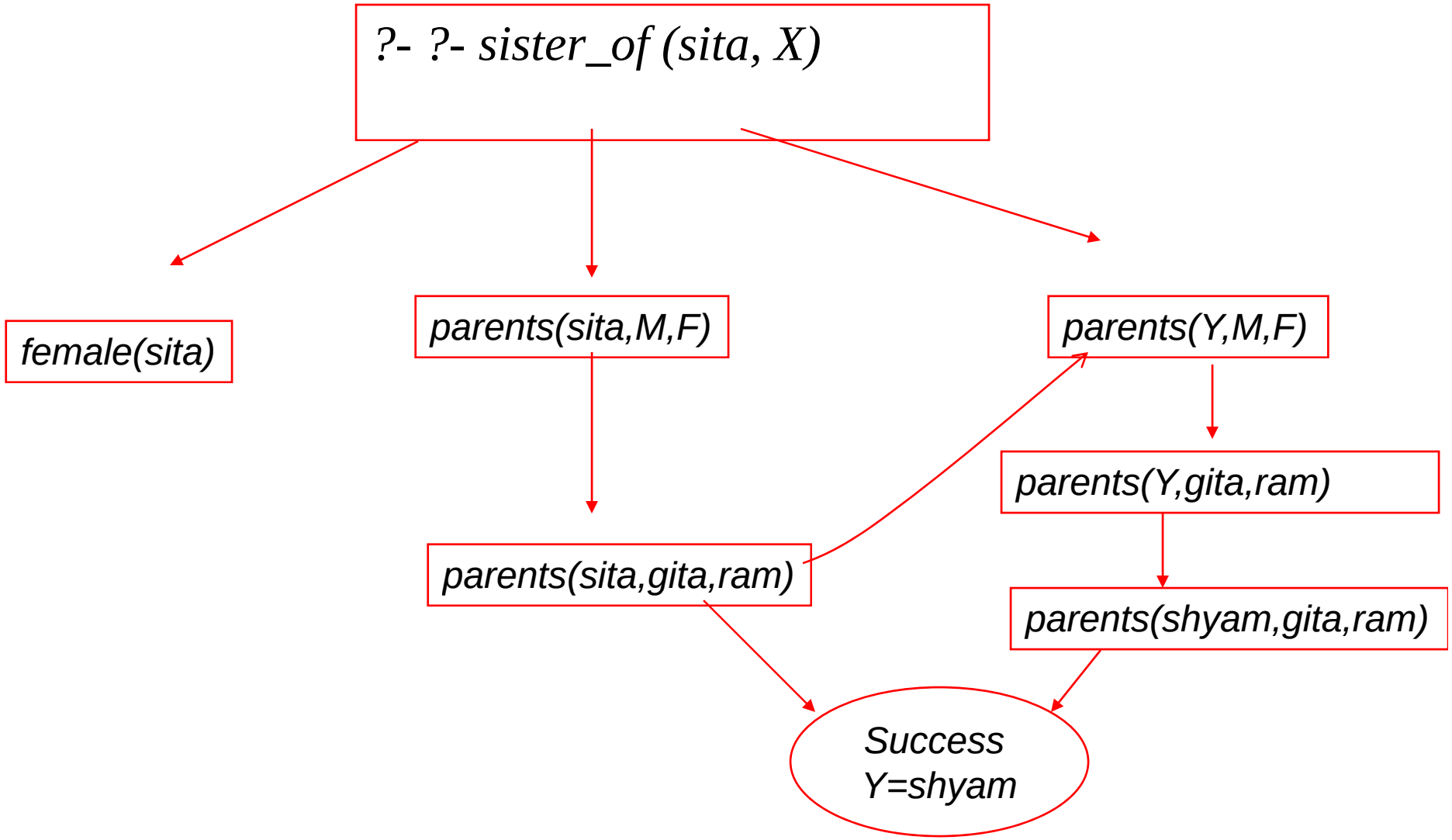
# Question Answering in presence of *rules*

- Facts
  - male (ram).
  - male (shyam).
  - female (sita).
  - female (gita).
  - parents (shyam, gita, ram).
  - parents (sita, gita, ram).

# Question Answering: Y/N type: *is sita the sister of shyam?*



# Question Answering: wh-type: *whose sister is sita?*



# Exercise

1. From the above it is possible for somebody to be her own sister. How can this be prevented?

# An example Prolog Program

# Shows path with mode of conveyeance from city $C_1$ to city $C_2$

- :-use_module(library(lists)).

- byCar(auckland,hamilton).
- byCar(hamilton,raglan).
- byCar(valmont,saarbruecken).
- byCar(valmont,metz).

- byTrain(metz,frankfurt).
- byTrain(saarbruecken,frankfurt).
- byTrain(metz,paris).
- byTrain(saarbruecken,paris).

- byPlane(frankfurt,bangkok).
- byPlane(frankfurt,singapore).
- byPlane(paris,losAngeles).
- byPlane(bangkok,auckland).
- byPlane(losAngeles,auckland).

- go(C1,C2) :- travel(C1,C2,L), show_path(L).

- travel(C1,C2,L) :- direct_path(C1,C2,L).

- travel(C1,C2,L) :- direct_path(C1,C3,L1),travel(C3,C2,L2),append(L1,L2,L).

- direct_path(C1,C2,[C1,C2,' by car']):- byCar(C1,C2).
- direct_path(C1,C2,[C1,C2,' by train']):- byTrain(C1,C2).
- direct_path(C1,C2,[C1,C2,' by plane']):- byPlane(C1,C2).

- show_path([C1,C2,M|T]) :- write(C1),write(' to '),write(C2),write(M),nl,show_path(T).

# Rules

- Statements about *objects* and their *relationships*
- Expess
  - *If-then conditions*
    - *I use an umbrella if there is a rain*
    - *use(i, umbrella) :- occur(rain).*
  - *Generalizations*
    - *All men are mortal*
    - *mortal(X) :- man(X).*
  - *Definitions*
    - *An animal is a bird if it has feathers*
    - *bird(X) :- animal(X), has_feather(X).*

# Prolog Program Flow, BackTracking and Cut

*Controlling the program flow*

# Prolog's computation

- **Depth First Search**
  - **Pursues a goal till the end**
- **Conditional AND; *falsity* of any goal prevents satisfaction of further clauses.**
- **Conditional OR; *satisfaction* of any goal prevents further clauses being evaluated.**

# Control flow (top level)

Given

*g:- a, b, c.* (1)

*g:- d, e, f; g.* (2)

If prolog cannot satisfy (1), control will automatically fall through to (2).

# Control Flow within a rule

Taking (1),

*g:- a, b, c.*

If *a* succeeds, prolog will try to satisfy *b,* succeding which *c* will be tried.

For ANDed clauses, control flows forward till the '.', iff the current clause is *true.*

For ORed clauses, control flows forward till the '.', iff the current clause evaluates to *false.*

# What happens on failure

- **REDO the immediately preceding goal.**

# Fundamental Principle of prolog programming

- **Always place the more general rule AFTER a specific rule.**

# CUT

- **Cut tells the system that**

  ***IF YOU HAVE COME THIS FAR***

  ***DO NOT BACKTRACK***

  ***EVEN IF YOU FAIL SUBSEQUENTLY.***

  **‘CUT’ WRITTEN AS ‘!’ ALWAYS SUCCEEDS.**

# Fail

- This predicate always fails.
- *Cut* and *Fail* combination is used to produce negation.
- Since the LHS of the neck cannot contain any operator, *A* → *~B* is implemented as

*B :- A, !, Fail.*

# Predicate Calculus

- Introduction through an example *(Zohar Manna, 1974):*
  - Problem: A, B and C belong to the Himalayan club. Every member in the club is either a mountain climber or a skier or both. A likes whatever B dislikes and dislikes whatever B likes. A likes rain and snow. No mountain climber likes rain. Every skier likes snow. *Is there a member who is a mountain climber and not a skier?*
- Given knowledge has:
  - Facts
  - Rules

# A wrong prolog program!

1. member(a).
2. member(b).
3. member(c).
4. mc(X);sk(X) :- member(X) /* X is a mountain climber or skier or both if X is a member; operators NOT allowed in the head of a horn clause; hence wrong*/
5. like(X, snow) :- sk(X). /*all skiers like snow*/
6. \+like(X, rain) :- mc(X). /*no mountain climber likes rain; \+ is the not operator; negation by failure; wrong clause*/
7. \+like(a, X) :- like(b,X). /* a dislikes whatever b likes*/
8. like(a, X) :- \+like(b,X). /* a dislikes whatever b likes*/
9. like(a,rain).
10. like(a,snow).

?- member(X),mc(X),\+sk(X).

# Prolog's way of making and breaking a list

Problem: to remove duplicates from a list

```
rem_dup([],[]).
rem_dup([H|T],L) :- member(H,T), !, rem_dup(T,L).
rem_dup([H|T],[H|L1]) :- rem_dup(T,L1).
```

Note: The cut ! in the second clause needed, since after succeeding at member(H,T), the 3rd clause should not be tried even if rem_dup(T,L) fails, which prolog will otherwise do.