CS344 : Introduction to Artificial Intelligence

Lecture 15- Robotic Knowledge Representation and Inferencing; Prolog

A *planning* agent

- An agent interacts with the world via perception and actions
- Perception involves sensing the world and assessing the situation
  - creating some internal representation of the world
- Actions are what the agent does in the domain. Planning involves reasoning about actions that the agent intends to carry out
- *Planning* is the reasoning side of acting
- This reasoning involves the representation of the world that the agent has, as also the representation of its actions.
- Hard constraints where the objectives *have to* be achieved completely for success
- The objectives could also be soft constraints, or *preferences*, to be achieved as much as possible

Interaction with static domain

- The agent has complete information of the domain (perception is perfect), actions are instantaneous and their effects are deterministic.
- The agent knows the world completely, and it can take all facts into account while planning.
- The fact that actions are instantaneous implies that there is no notion of time, but only of sequencing of actions.
- The effects of actions are deterministic, and therefore the agent knows what the world will be like after each action.

Two kinds of planning

- *Projection* into the future
  - The planner searches through the possible combination of actions to find the *plan* that will work
- *Memory based planning*
  - looking into the past
  - The agent can retrieve a plan from its memory

Planning

•*Definition : Planning is arranging a sequence of actions to achieve a goal.*

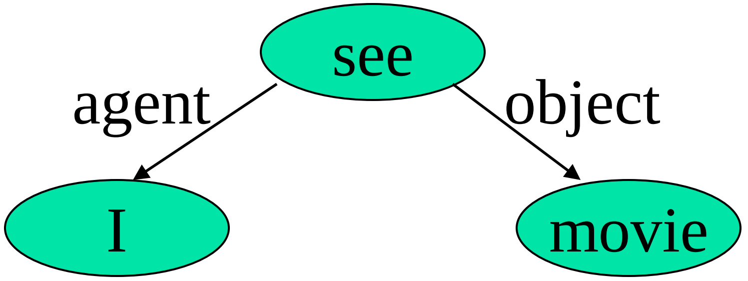
•Uses core areas of AI like searching and reasoning &
•Is the core for areas like NLP, Computer Vision.

•Robotics
- Kinematics (ME)
- Planning (CSE)

•Examples : Navigation , Manoeuvring, Language Processing (Generation)

Language & Planning

- Non-linguistic representation for sentences.



- Sentence generation
  - Word order determination (Syntax planning)

  E.g. I see movie (English)

  I movie see (Intermediate Language)

STRIPS

- Stanford Research Institute Problem Solver (1970s)
  - Planning system for a robotics project : SHAKEY (by Nilsson et.al.)
- Knowledge Representation : First Order Logic.

- Algorithm : Forward chaining on rules.

- Any search procedure : Finds a path from *start* to *goal*.
  - Forward Chaining : Data-driven inferencing
  - Backward Chaining : Goal-driven

Forward & Backward Chaining

- Rule : man(x) → mortal(x)
- Data : man(Shakespeare)

To prove : mortal(Shakespeare)

- *Forward Chaining:*

man(Shakespeare) matches LHS of Rule.

X = Shakespeare

$\Rightarrow$ mortal(Shakespeare) added

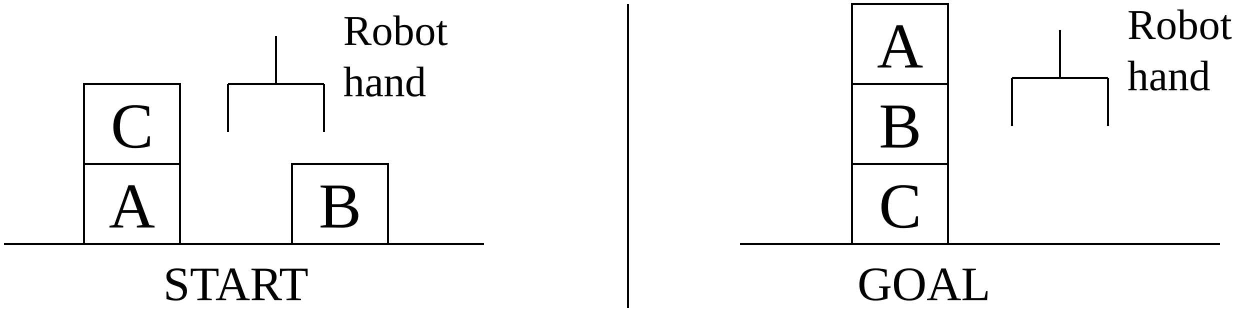
-Forward Chaining used by design expert systems

- *Backward Chaining:* uses RHS matching

- Used by diagnostic expert systems

Example : Blocks World

- STRIPS : A planning system – Has rules with precondition deletion list and addition list



Sequence of actions :

1. Grab C
2. Pickup C
3. Place on table C
4. Grab B
5. Pickup B
6. Stack B on C
7. Grab A
8. Pickup A
9. Stack A on B

Example : Blocks World

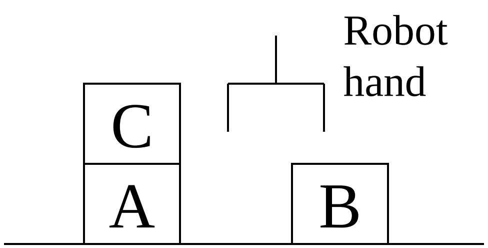
•Fundamental Problem :
The *frame problem* in AI is concerned with the question of what piece of knowledge is relevant to the situation.

•Fundamental Assumption : Closed world assumption
If something is not asserted in the knowledge base, it is assumed to be false.

(Also called "Negation by failure")

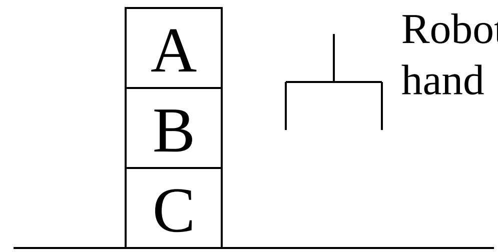
Example : Blocks World

- STRIPS : A planning system – Has rules with precondition deletion list and addition list



START

on(B, table)
on(A, table)
on(C, A)
hand empty
clear(C)
clear(B)



GOAL

on(C, table)
on(B, C)
on(A, B)
hand empty
clear(A)

Rules

- *R1 : pickup(x)*

Precondition & Deletion List : hand empty,
on(x,table), clear(x)

Add List : holding(x)

- *R2 : putdown(x)*

Precondition & Deletion List : holding(x)

Add List : hand empty, on(x,table), clear(x)

Rules

- *R3 : stack(x,y)*

Precondition & Deletion List :holding(x), clear(y) Add List : on(x,y), clear(x)

- *R4 : unstack(x,y)*

Precondition & Deletion List : on(x,y), clear(x)
Add List : holding(x), clear(y)

Plan for the block world problem

- For the given problem, Start → Goal can be achieved by the following sequence :
  1. Unstack(C,A)
  2. Putdown(C)
  3. Pickup(B)
  4. Stack(B,C)
  5. Pickup(A)
  6. Stack(A,B)
- Execution of a plan: achieved through a data structure called Triangular Table.

Triangular Table

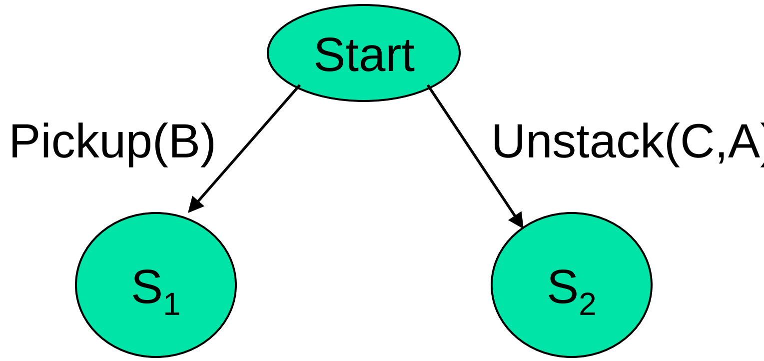
1	on(C,A) clear(C) hand empty	unstack(C,A)					
2		holding(C)	putdown(C)				
3	on(B,table)		hand empty	pickup(B)			
4			clear(C)	holding(B)	stack(B,C)		
5	on(A,table)	clear(A)			hand empty	pickup(A)	
6					clear(B)	holding(A)	stack(A,B)
7			on(C,table)		on(B,C)		on(A,B) clear(A)
	0	1	2	3	4	5	6

Triangular Table

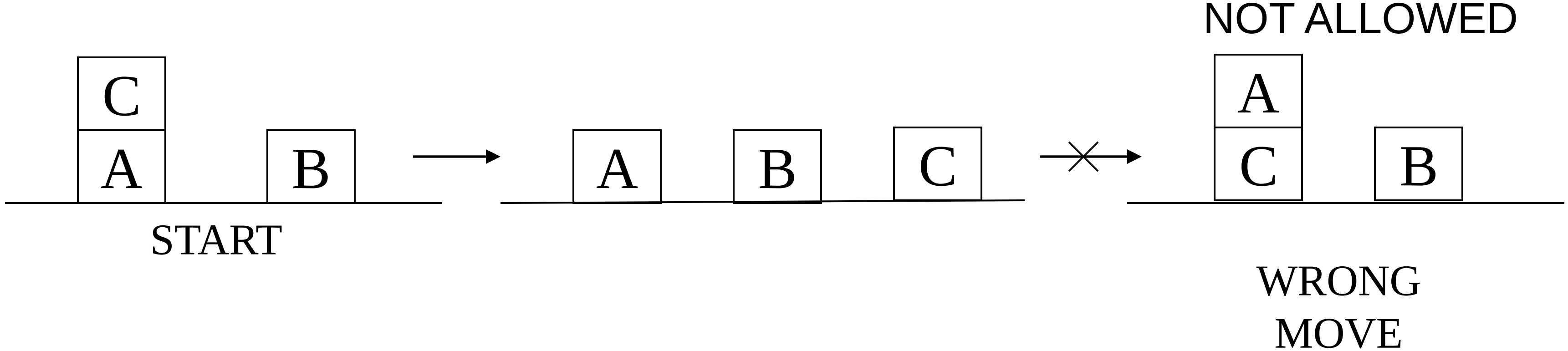
- For n operations in the plan, there are :
  - (n+1) rows : 1 → n+1
  - (n+1) columns : 0 → n
- At the end of the i^{th} row, place the i^{th} component of the plan.
- The row entries for the i^{th} step contain the pre-conditions for the i^{th} operation.
- The column entries for the j^{th} column contain the add list for the rule on the top.
- The $<i,j>^{th}$ cell (where $1 \leq i \leq n+1$ and $0 \leq j \leq n$) contain the pre-conditions for the i^{th} operation that are added by the j^{th} operation.
- The first column indicates the starting state and the last row indicates the goal state.

Search in case of planning

- Ex: Blocks world



- Triangular table leads
- to some amount of fault-tolerance in the robot



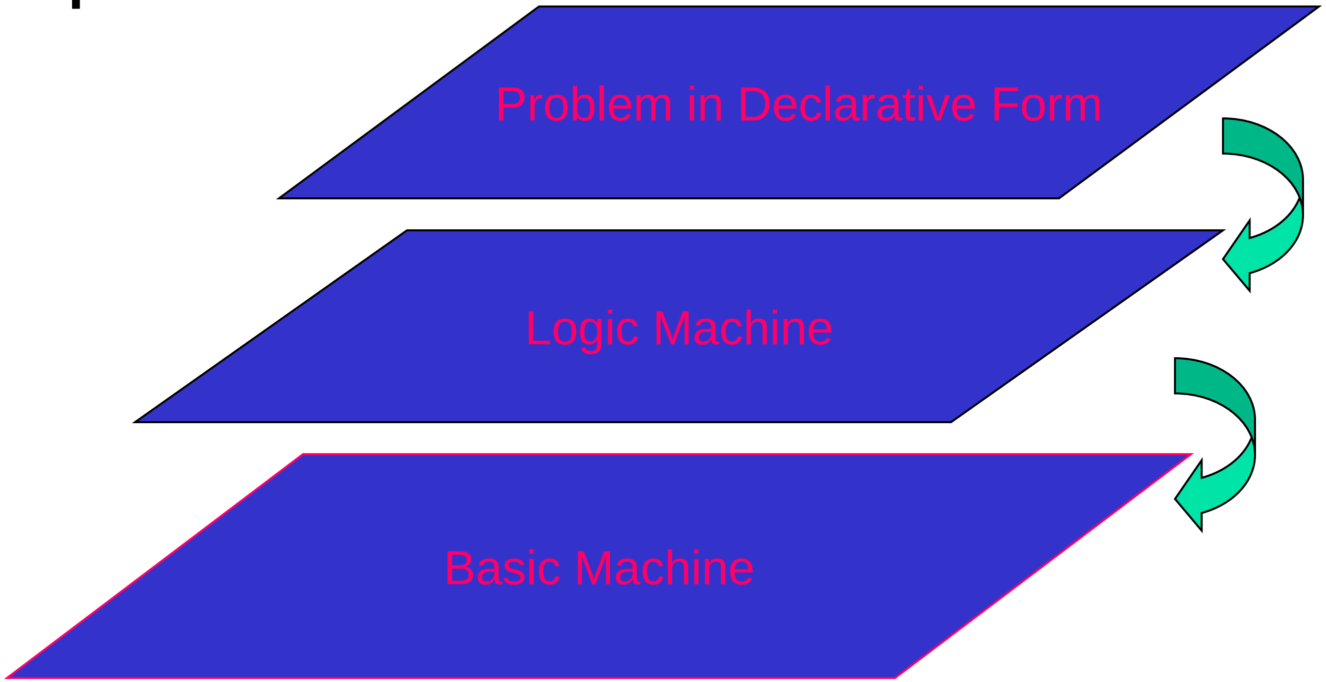
Resilience in Planning

- After a wrong operation, can the robot come back to the right path ?
- *i.e.* after performing a wrong operation, if the system again goes towards the goal, then it has resilience w.r.t. that operation
- Advanced planning strategies
  - Hierarchical planning
  - Probabilistic planning
  - Constraint satisfaction

Prolog Programming

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.

<u>High level explanation:</u>

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*

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

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

1. Second goal fails
2. Return to marked place
   and try to resatisfy the first goal

Backtracking *(continued)*

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

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

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

Backtracking *(continued)*

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

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

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

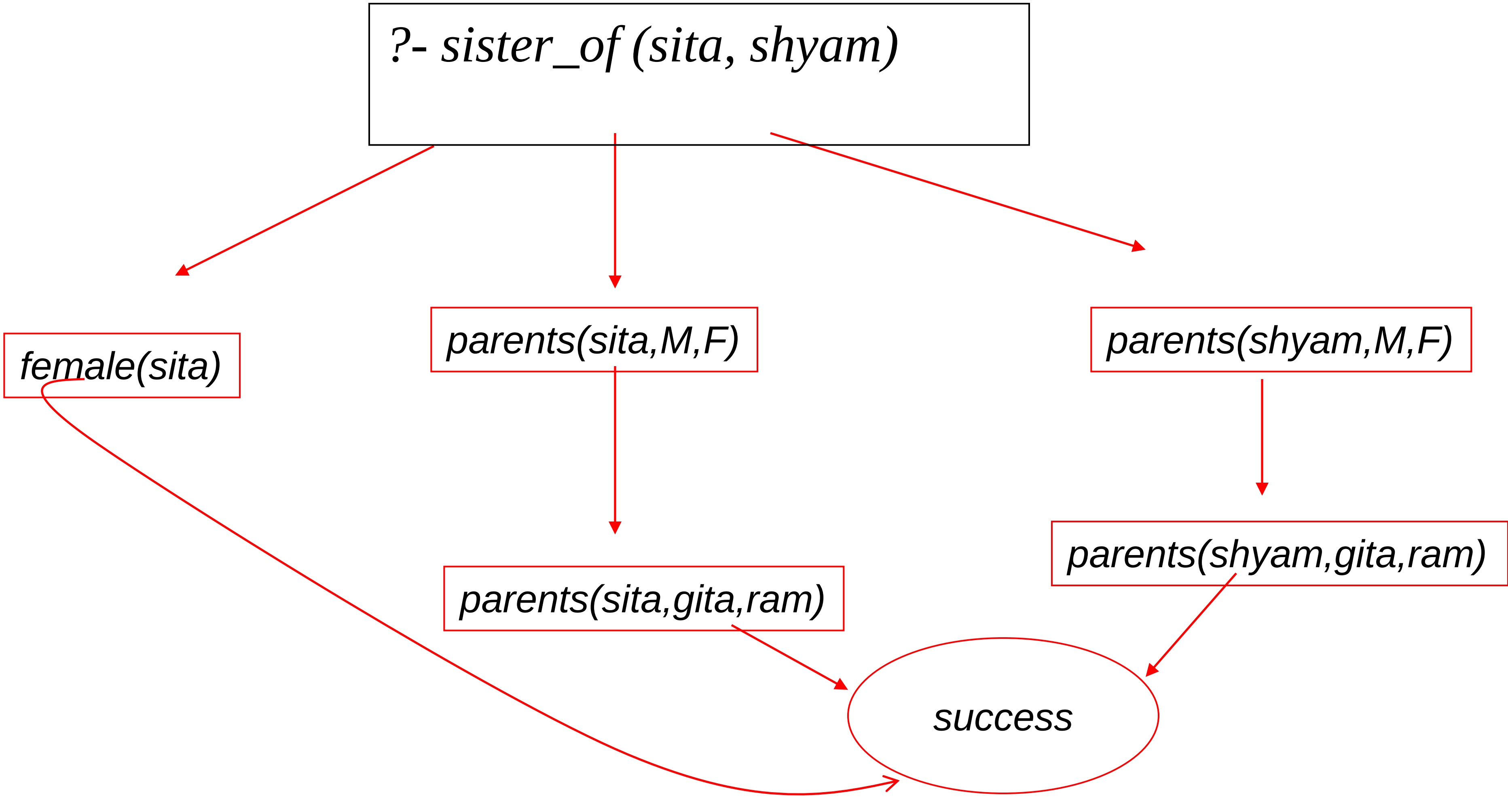
sister_of (X,Y):- female (X),
parents (X, M, F),
parents (Y, M, F).

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

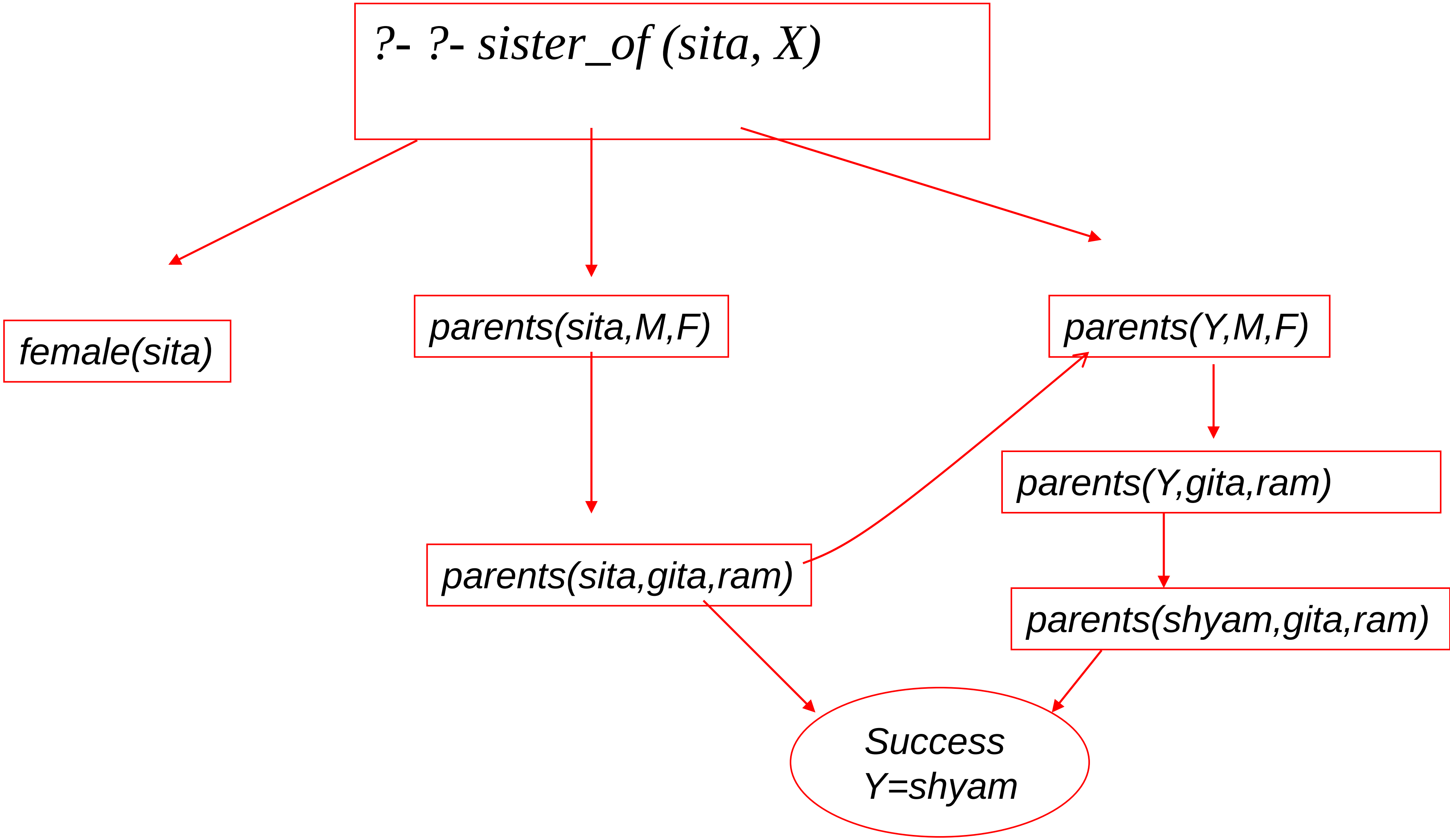
Question Answering in presence of *rules*

- 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?