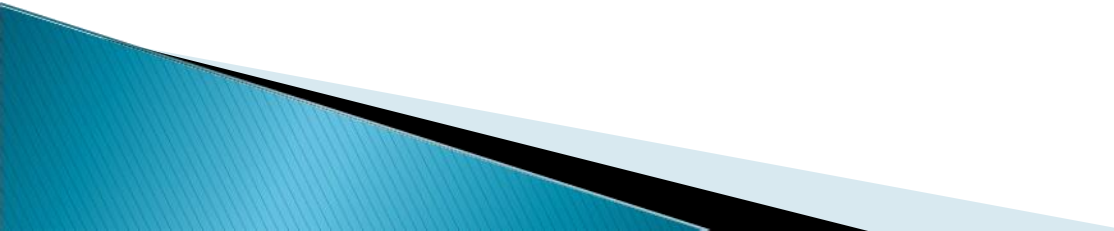


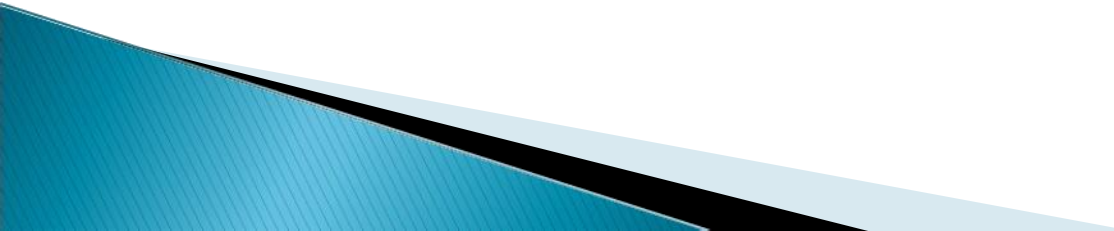
Introduction to PROLOG



Content

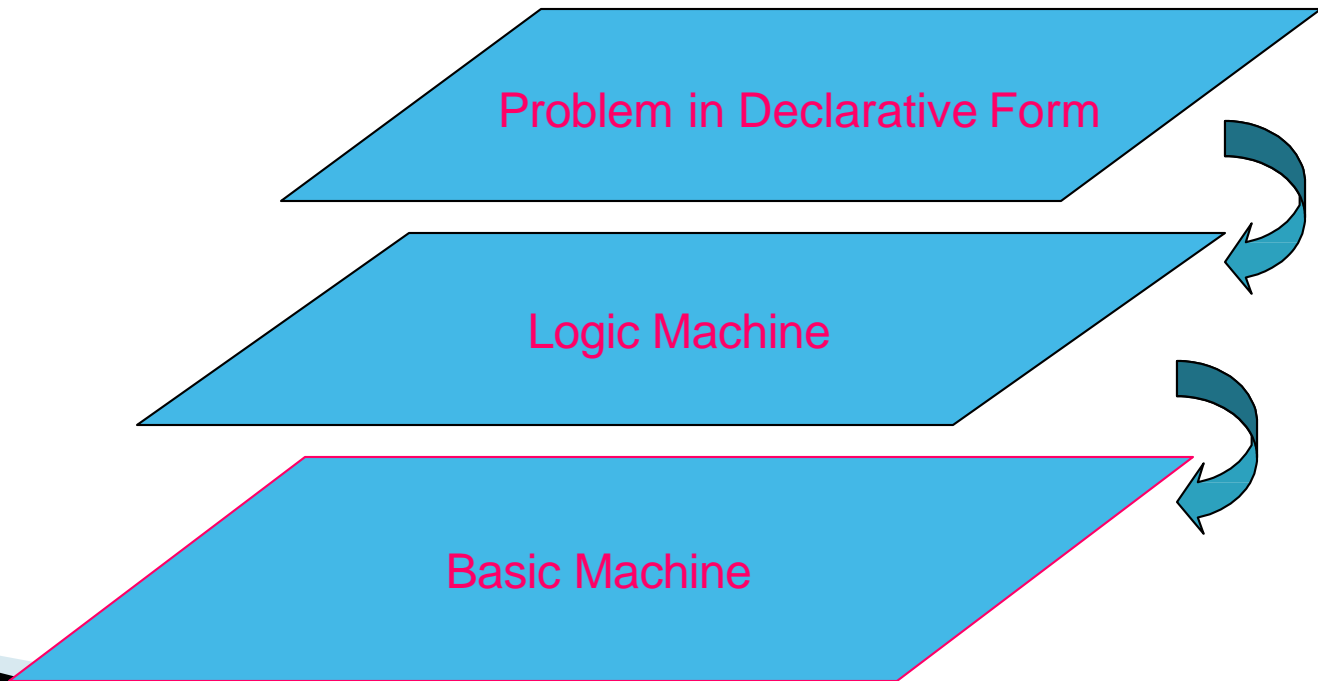
- Introduction
 - SWI_PROLOG
 - Fundamentals of PROLOG
 - Applications
 - References
- 

Introduction

- Invented early seventies by Alain Colmerauer in France and Robert Kowalski in Britain.
 - Prolog = Programmation en Logique (Programming in Logic).
 - Prolog is a declarative programming language unlike most common programming languages.
 - In a declarative language,
the programmer specifies a goal to be achieved
the Prolog system works out how to achieve it
 - relational databases owe something to Prolog
- 

Introduction

- **PRO**gramming in **LOGic**
- Declarative language
- Emphasis on what rather than how
- It is widely used in the field of AI

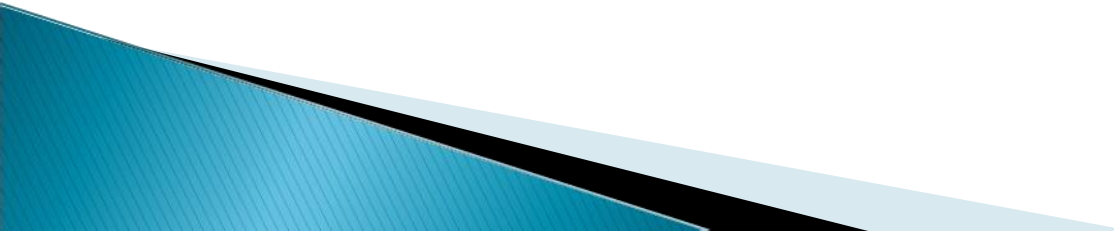


SWI-Prolog

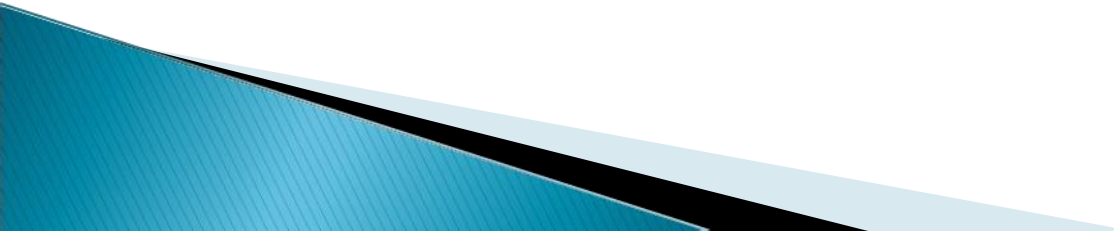
- SWI-Prolog offers a comprehensive *FREE SOFTWARE* Prolog environment.
- Link for downloading:
<http://www.swi-prolog.org/download/stable>
- A Self-installing executable for MS-Windows: **swipl-win.exe**
- Works on Windows XP
- LINUX versions are also available.

Application

some applications of Prolog are:

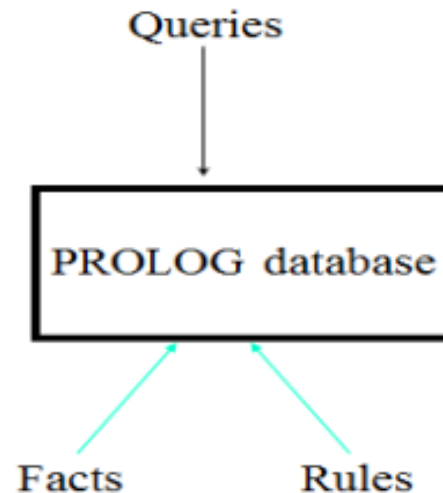
- ☐ intelligent data base retrieval
 - ☐ natural language understanding
 - ☐ expert systems
 - ☐ specification language
 - ☐ machine learning
 - ☐ robot planning
 - ☐ automated reasoning
 - ☐ problem solving
- 

Fundamentals

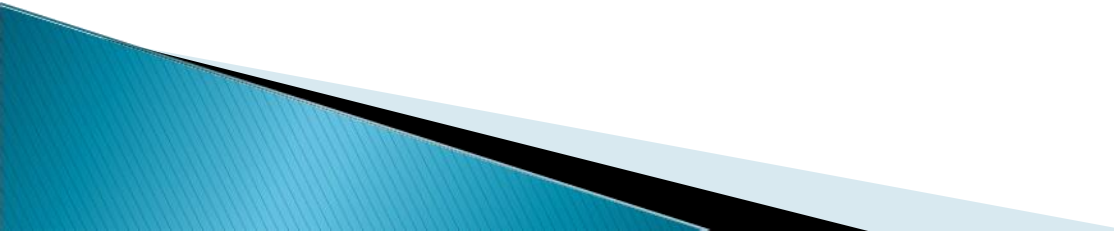
- Facts
 - Rules
 - Query
 - Unification
 - Resolution
 - Backtracing
 - Cuts and
negations
- 

Prolog Program:

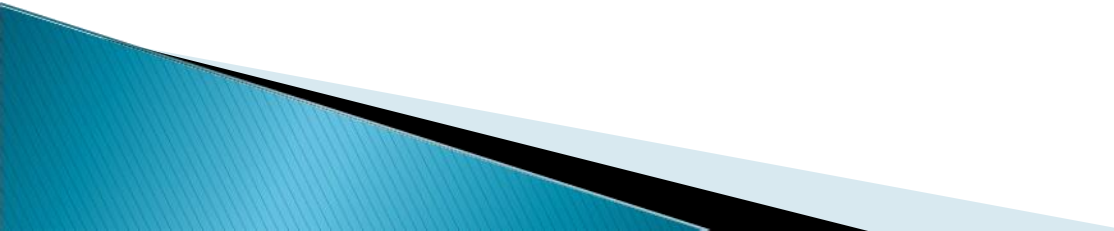
- solve problems by declaring objects and their relationships
- The PROLOG Programmer loads facts and rules into the database.
- Makes queries to the database to see if a fact is in the database or can be implied from the facts and rules therein



FACTS

- Facts are statements about what is true about a problem, instead of instructions how to accomplish the solution.
 - The Prolog system uses the facts to work out how to accomplish the solution by searching through the space of possible solutions.
 - It is defined by an identifier followed by an n-tuple of constants.
 - A relation identifier is referred to as a predicate
 - When a tuple of values is in a relation we say the tuple satisfies the predicate.
- 

Syntax for fact declaration

- 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.
- 

Prolog Syntax:

✓ Constants

- **Atoms:** Alphanumeric atoms - alphabetic character sequence starting with a lower case letter
Examples: apple a1 apple_cart
- Quoted atoms - sequence of characters surrounded by single quote
Examples: 'Apple' 'hello world'
- Symbolic atoms - sequence of symbolic characters
Examples: & < > * - + >>
- Special atoms
Examples: ! ; [] { }
- **Numbers:** Integers and Floating Point numbers
Examples: 0 1 9821 -10 1.3 -1.3E102

✓ **Variable Names:** A sequence of alphanumeric characters beginning with an upper case letter or an underscore

Examples: Anything _var X

✓ Compound Terms (structures)

- an atom followed by an argument list containing terms. The arguments are enclosed within brackets and separated by commas

Example: isa(dog, mammal)

Examples

Predicate

valuable(gold)

owns(john,gold)

father(john,mary)

gives (john,book,mary)

Interpretation

Gold is valuable.

John owns gold.

John is the father of
Mary

John gives the book to
Mary

RULES

- Specifies under what conditions a tuple of values satisfies a predicate.
- The basic building block of a rule is called an *atom*
- $\text{Atom} :- \text{Atom1}, \dots, \text{Atomn}$
- *If each of Atom1,...,Atomn is true, then Atom is also true.*

Cont..

Rules specify:

- ▮ **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 of rule

- └<head> :- <body>
- └Read ‘:-’ as ‘if’.
- └ *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 ‘.’***

QUERIES

There are two types of queries:

- **Ground Query**

- `edge(a,b)`
- This query is called a ground query because it consists only of value identifiers as parameters to the predicate.
- a ground query is posed we expect a yes/no answer.

- **Non Ground Query**

- They have variables as parameters
- `tedge(a,X)`

Variables

Always begin with a capital letter

?- likes (john,X).

?- likes (john, Something).

But not

?- likes (john,something)

Example

Facts: ()

- likes(john,mary).
- likes(john,X). % Variables begin with capital

Queries

- ?- likes(X,Y).
- X=john, Y=Mary. % hit “;” for more
- ?- likes(X,X).
- X=john.

Example

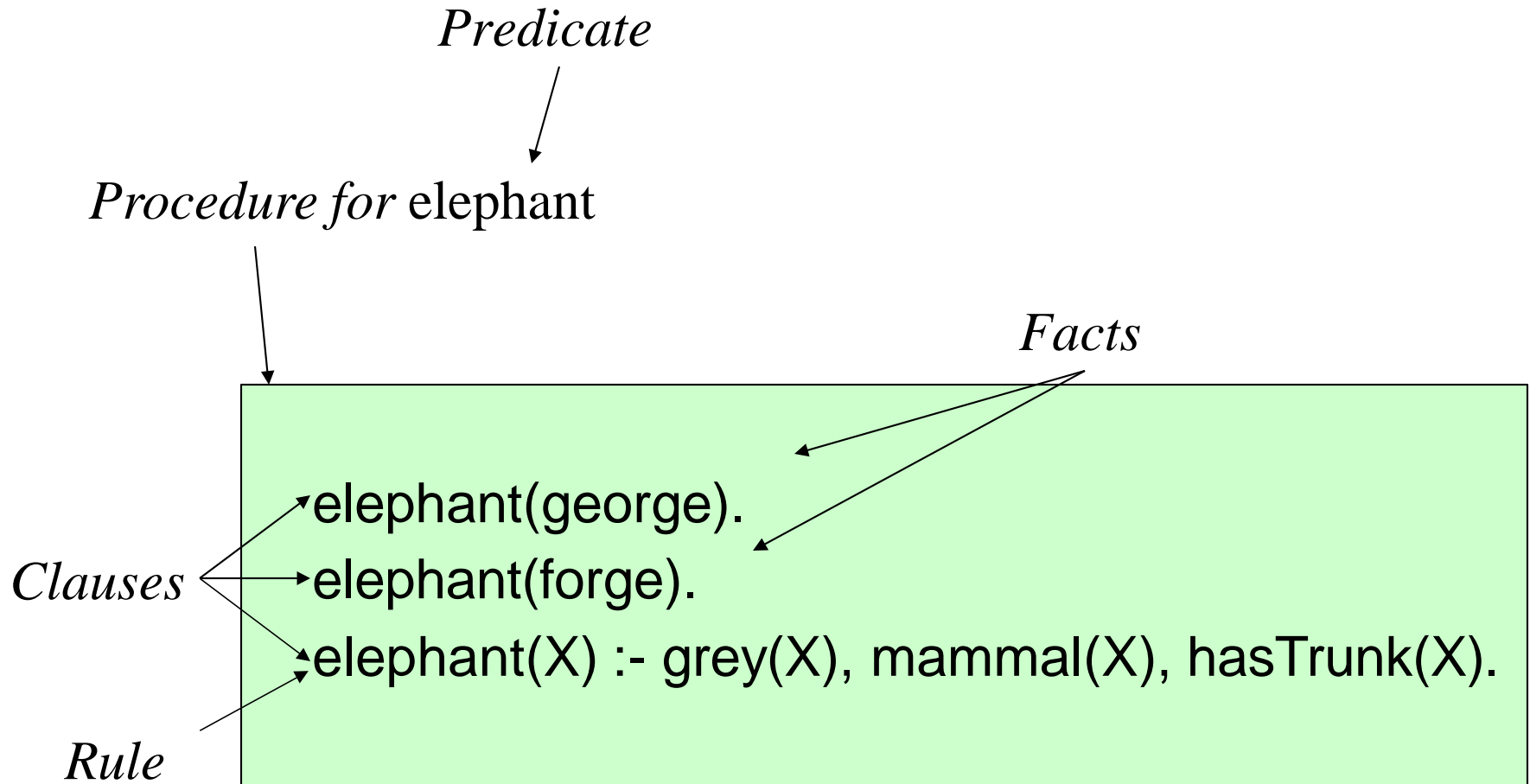
Rules

- likes(john,X) :- likes(X,wine). % :- = if
- likes(john,X):- female(X), likes(X,john).

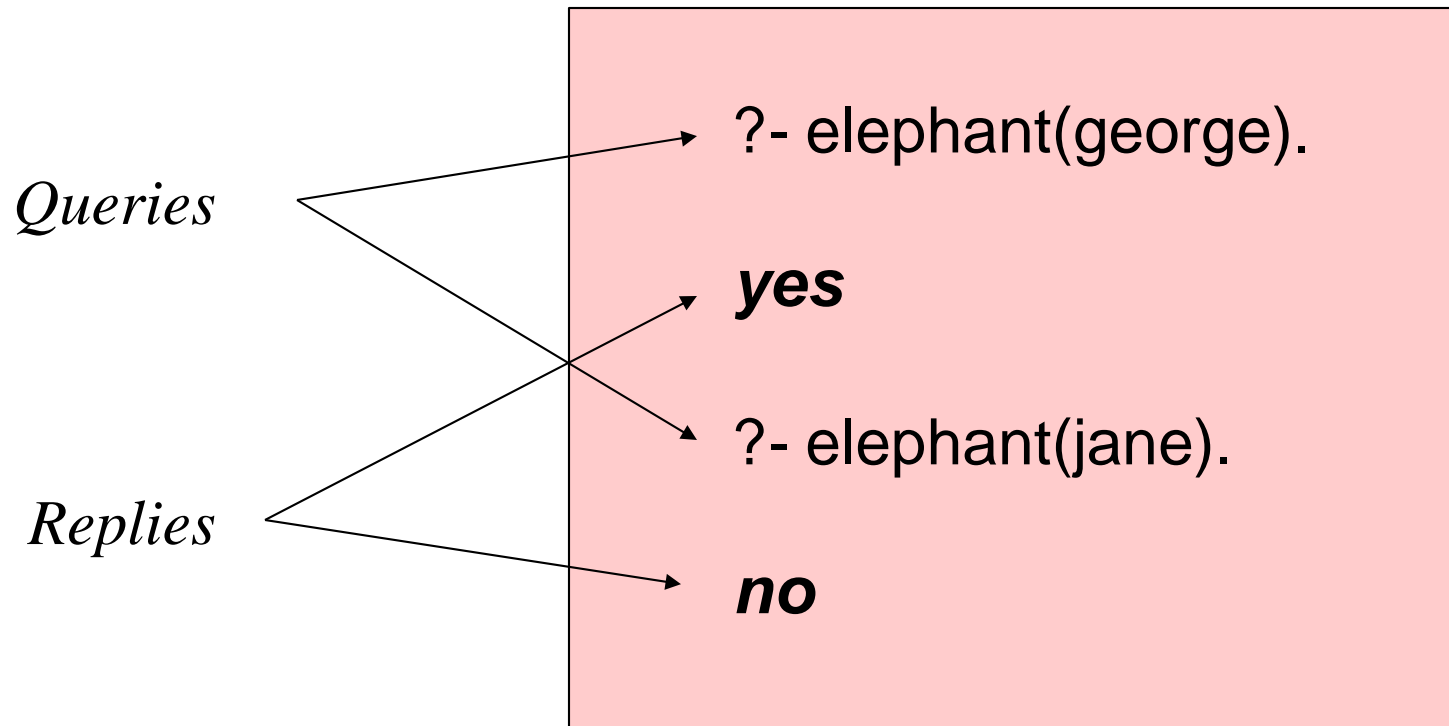
Query: ? - likes(john,Y).

- Y = bill ;
- no

Example



Example



Conjunction & Disjunction

- **Conjunction** of predicates is represented as a sequence of structures, separated by commas“,”.

- It is referred as “AND”

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

- **Disjunction** of predicates is represented as a sequence of structures, separated by semicolon”;”.

- It is referred as “OR”

friend(ram,shyam):-

friend(shyam,sita);friend(shyam,mohn)

Unification

- Questions based on facts are answered by matching
- Unification is the name given to the way **Prolog** does its 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
- This means not provable from the given facts.

Question Answering in presence of rules

Facts

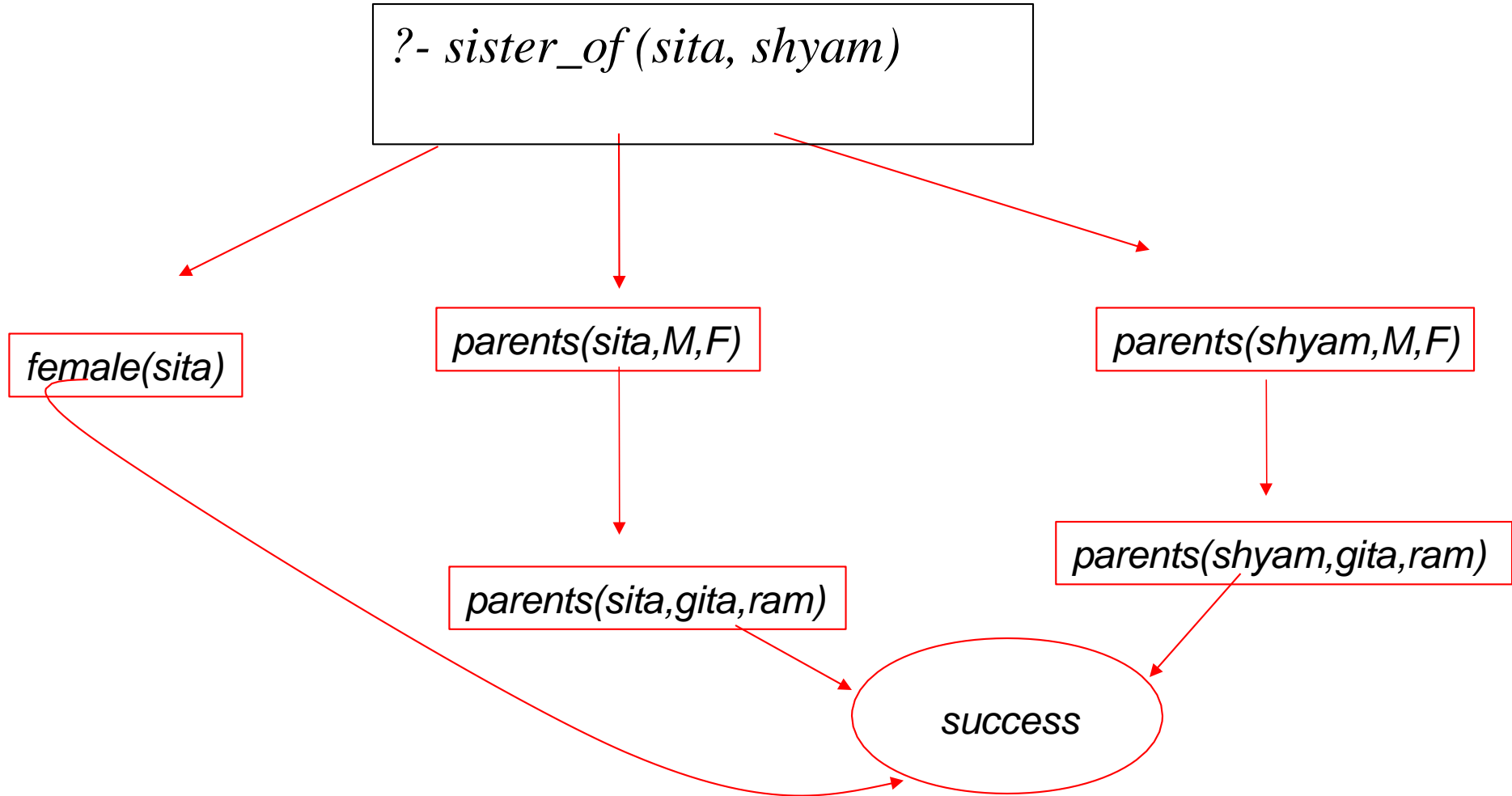
- male (ram).
- male (shyam).
- female (sita).
- female (gita).
- parents (shyam, gita, ram).
- parents (sita, gita, ram).

Rule: *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

Backtracking

sister_of(x, y) : $\text{female}(x)$, $\text{parent}(x, M, F)$,
 $\text{parent}(y, M, F)$



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

?- ?- *sister_of(sita, X)*

female(sita)

parents(sita, M, F)

parents(Y, M, F)

parents(sita, gita, ram)

parents(Y, gita, ram)

parents(shyam, gita, ram)

Success
Y=shyam

Arithmetic in prolog

- Prolog provides a number of basic arithmetic tools.

- **Arithmetic examples Prolog Notation**

- $6 + 2 = 8$ 8 is 6+2.
- $6 * 2 = 12$ 12 is 6*2.

Answers to arithmetic questions by using variables.

For example:

?- X is 6+2.

X=8

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

$g:- a, b, c. \quad (1)$

$g:- d, e, f; g. \quad (2)$

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 , succeeding 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*.

On Failure

REDO the immediately preceding goal.

Always place the more general rule AFTER a specific rule

Cuts and Negation

- Automatic backtracking is one of the most characteristic features of Prolog.
- Backtracking can lead to inefficiency.
- Prolog can waste time exploring possibilities that lead nowhere.
- Cut is a goal that always succeeds
- Commits Prolog to the choices that were made since the parent goal was called
- CUTS are used to control over this aspect of its behaviour
- $p(X) \text{:- } b(X), c(X), \text{!}, d(X), e(X).$

Example

consider the following piece of cut-free code:

`p(X):- a(X).`

`p(X):- b(X), c(X), d(X), e(X).`

`p(X):- f(X).`

`a(1). b(1). c(1). d(2). e(2). f(3).`
`b(2). c(2).`

For query `p(X)` we will get the following responses:

`X = 1 ;`

`X = 2 ;`

`X = 3 ;`

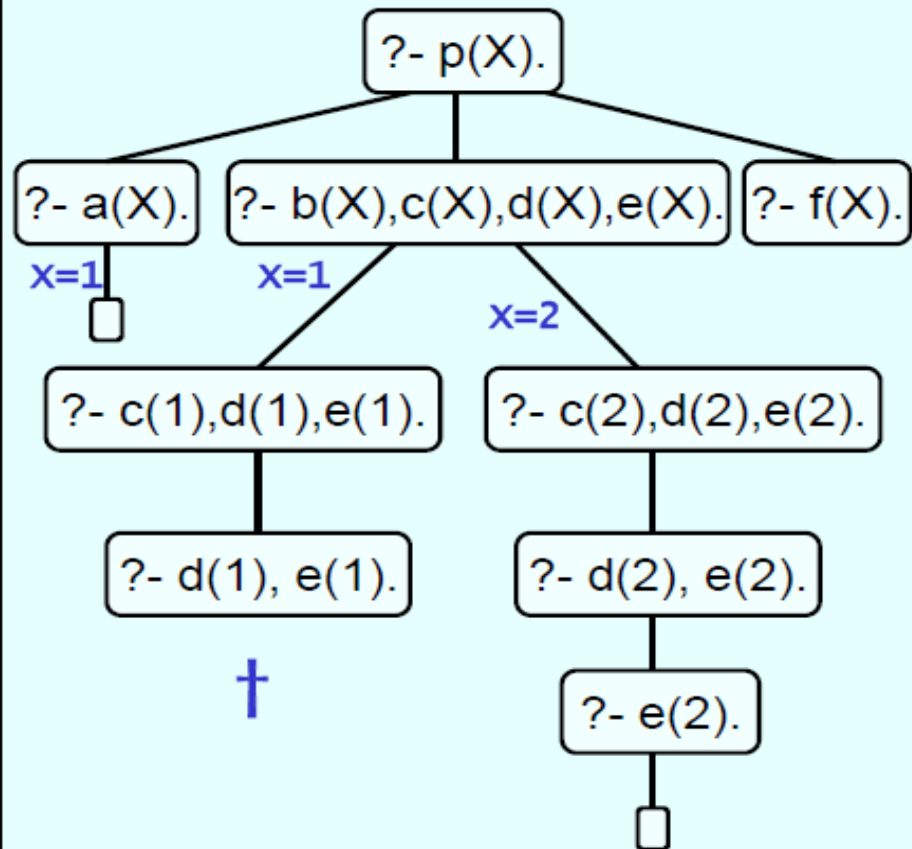
`no`

Here is the search tree that explains how Prolog finds these three solutions. Note that it has to backtrack once, namely when it enters the second clause for `p/1` and decides to unify the first goal with `b(1)` instead of `b(2)`.

Example: cut-free code

```
p(X):- a(X).  
p(X):- b(X), c(X), d(X), e(X).  
p(X):- f(X).  
a(1).  
b(1).  b(2).  
c(1).  c(2).  
d(2).  
e(2).  
f(3).
```

```
?- p(X).  
X=1;  
X=2;
```



- Suppose we insert a cut in the second clause:

```
p(X):- b(X), c(X), !, d(X), e(X).
```

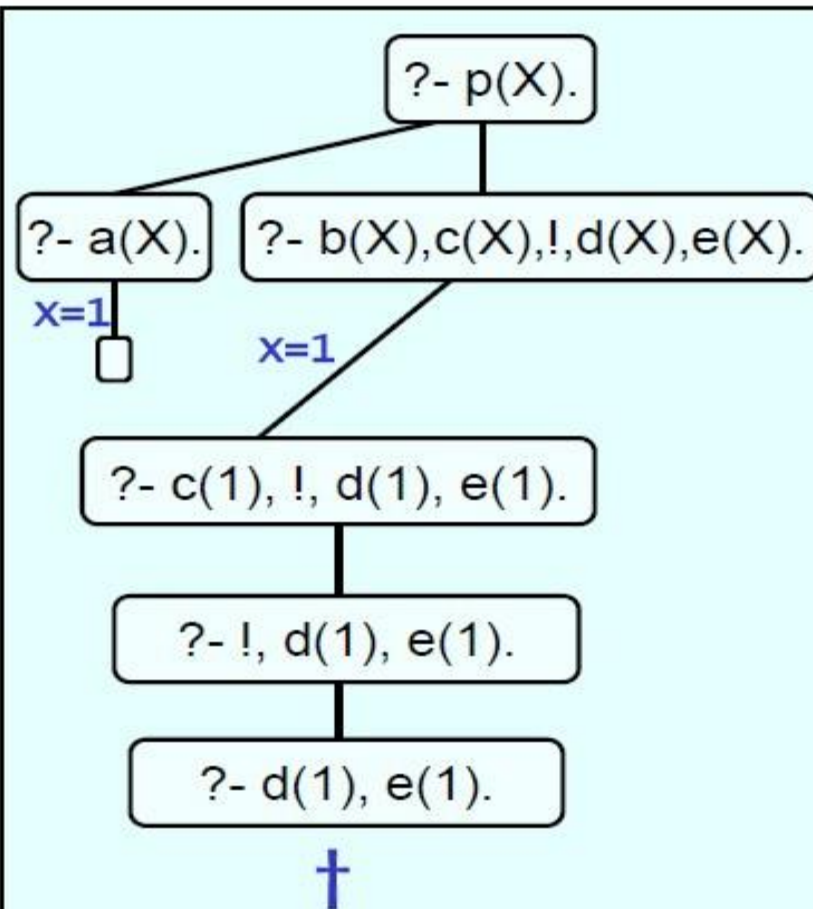
- If we now pose the same query we will get the following response:

```
?- p(X).  
X=1;  
no
```

Using CUT

```
p(X):- a(X).  
p(X):- b(X),c(X),!,d(X),e(X).  
p(X):- f(X).  
a(1).  
b(1).  b(2).  
c(1).  c(2).  
d(2).  
e(2).  
f(3).
```

```
?- p(X).  
X=1;  
no
```



Negation

Consider the following code:

```
enjoys(vincent,X) :- big_kahuna_burger(X),!,fail.  
enjoys(vincent,X) :- burger(X).  
burger(X) :- big_mac(X).  
burger(X) :- big_kahuna_burger(X).  
burger(X) :- whopper(X).  
big_mac(a).  
big_kahuna_burger(b).  
big_mac(c).  
whopper(d).
```

Using Negation

```
enjoys(vincent,X) :- burger(X), neg(big_kahuna_burger(X)).
```

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

Monkey Banana Problem

Write a program to solve the Monkey Banana Problem. Imagine a room containing a monkey, chair and some bananas. That have been hanged from the center of ceiling. If the monkey is clever enough he can reach the bananas by placing the chair directly below the bananas and climb on the chair. The problem is to prove the monkey can reach the bananas. The monkey wants it, but cannot jump high enough from the floor. At the window of the room there is a box that the monkey can use. The monkey can perform the following actions:

1. Walk on the floor
2. Climb the box
3. Push the box around(if it is beside the box)
4. Grasp the banana if it is standing on the box directly under the banana.

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.

Applications

- Expert Systems (Knowledge Representation and Inferencing)
 - Natural Language Processing
 - Definite Clause Grammar
 - intelligent data base retrieval
 - natural language understanding
 - expert systems
 - specification language
 - machine learning
 - robot planning
 - automated reasoning
 - problem solving
- 

THANKYOU

