UNIT-IV: Declarative Programming Paradigm: Logic Programming

Faculty In-charge

Mrinmoyee Mukherjee
Assistant Professor (IT Dept.)
email: mrinmoyeemukherjee@sfit.ac.in
Mob: 9324378409

Academic Year: 2023-24

OUTLINE OF UNIT-4

Sub-Unit	Contents
4.0	Logic programming with PROLOG
4.1	Resolution and Unification
4.2	Lists, Arithmetic execution order
4.3	Imperative control flow
4.4	Database manipulation
4.5	PROLOG facilities and deficiencies

4.0: Logic Programming with PROLOG

INTRODUCTION

- Distinctive style of Programming
- Collection of logical propositions and questions. There are only two components to any program: **facts and rules**.
  - The program is a knowledge base of facts and a series of rules to be applied to knowledge base
  - Programs are based on the techniques developed by logicians to form valid conclusions from available evidence (knowledge base)
- Interaction with the program: Posing queries to an inference engine (also called a query interpreter)
- Most widely used is Prolog. The name stands for Programming in Logic

Introduction to Logic Programming: **Prolog**

- Prolog is based on Horn Clauses.
- Horn Clauses are subset of Predicate Logic
- Predicate Logic is a syntax for reading and writing logical ideas
- Predicate Logic defines how reasoning gets done in logic terms
- To transform English statement to Predicate Logic, we need to remove unnecessary terms. What remains is relationship and entities or arguments

- English Statement: Elephant is bigger than Horse
  - Here the relation is bigger, and entities involve are Elephant and Horse
  - The relation is also called as Functor

- Prolog has three different types of clauses
  - Facts, Rules and Questions (or Queries)

bigger(elephant, horse).

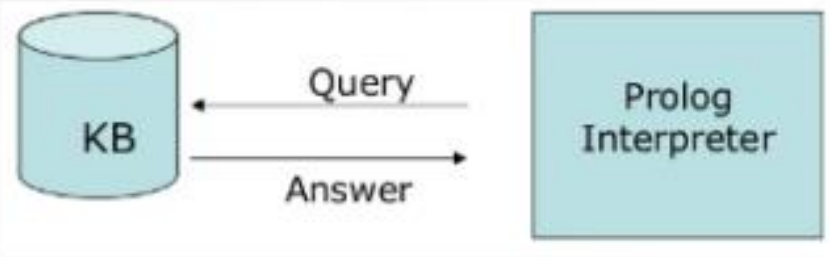
Relation

Arguments

Fact, There is full stop at the end
The entities are given in small case letters

Introduction to Prolog

- Every clause has full stop at the end.
- Clauses that are always True, are called as Facts
  - Ex. father(ali, haya).
- Some clauses are dependent on other clauses for being True, these are called as Rules
  - Ex. parent(person1, person2) :- father(person1, person2).
- To run a program in Prolog, we ask Questions about database



- Key Features of Prolog :
  - Unification : The basic idea is, can the given terms be made to represent the same structure.
  - Backtracking : When a task fails, prolog traces backwards and tries to satisfy previous task.
  - Recursion : Recursion is the basis for any search in program.

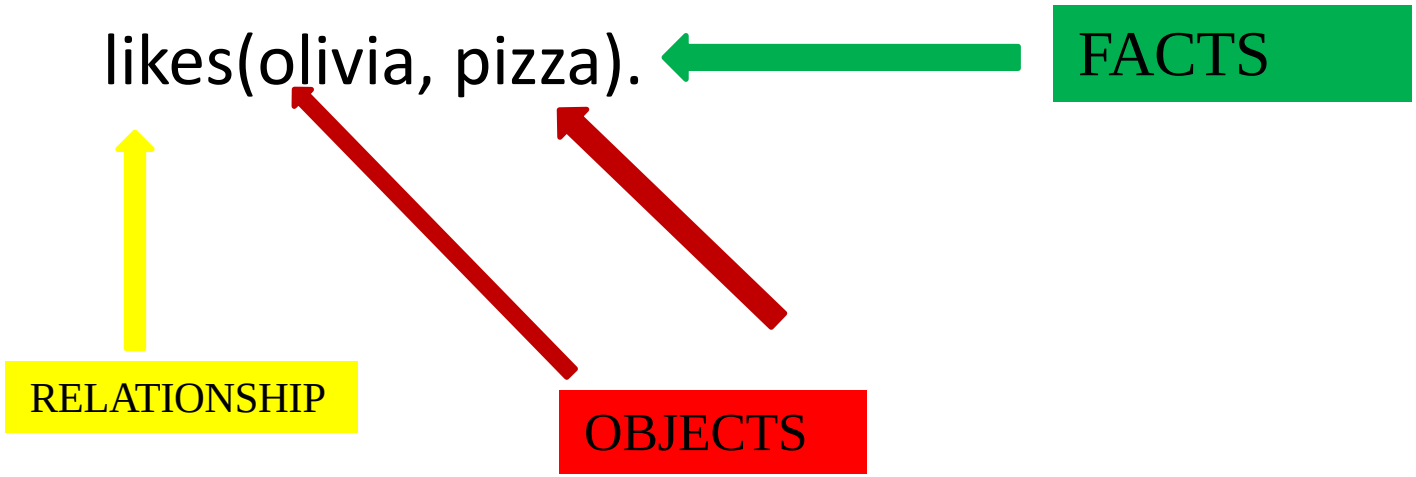
Introduction to Prolog

- Advantages :
  1. Easy to build database. Doesn't need a lot of programming effort.
  2. Pattern matching is easy. Search is recursion based.
  3. It has built in list handling. Makes it easier to play with any algorithm involving lists.
- Disadvantages :
  1. LISP (another logic programming language) dominates over prolog with respect to I/O features.
  2. Sometimes input and output is not easy.
- Applications :
  1. Prolog is highly used in artificial intelligence(AI).
  2. Prolog is also used for pattern matching over natural language parse trees.

Introduction-PROLOG

FACTS OBJECTS RELATIONSHIP

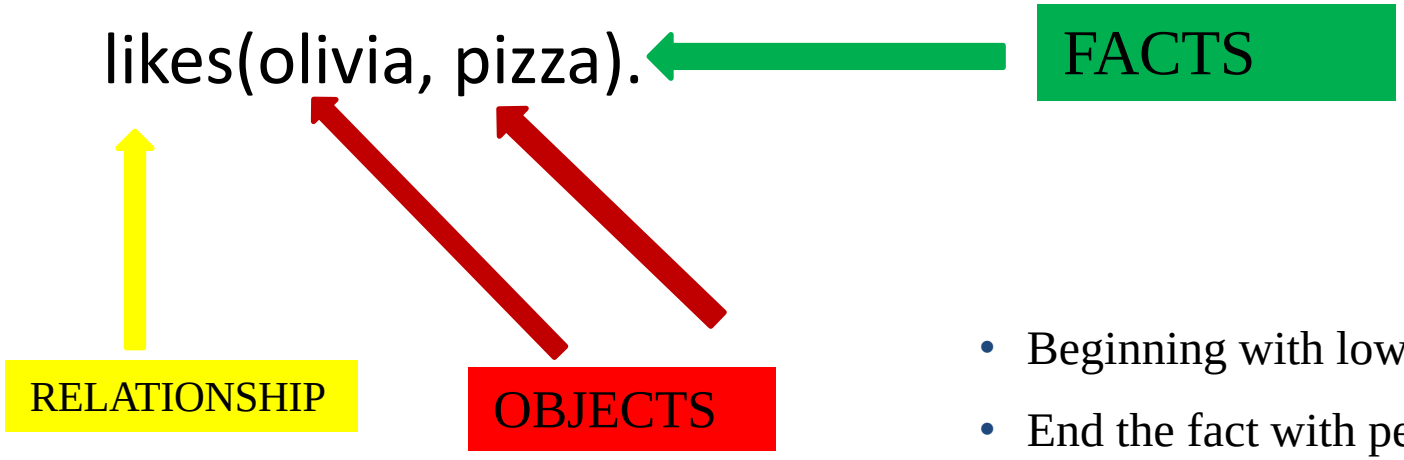
- Olivia likes Pizza.



Introduction-PROLOG

FACTS OBJECTS RELATIONSHIP

- Olivia likes Pizza.

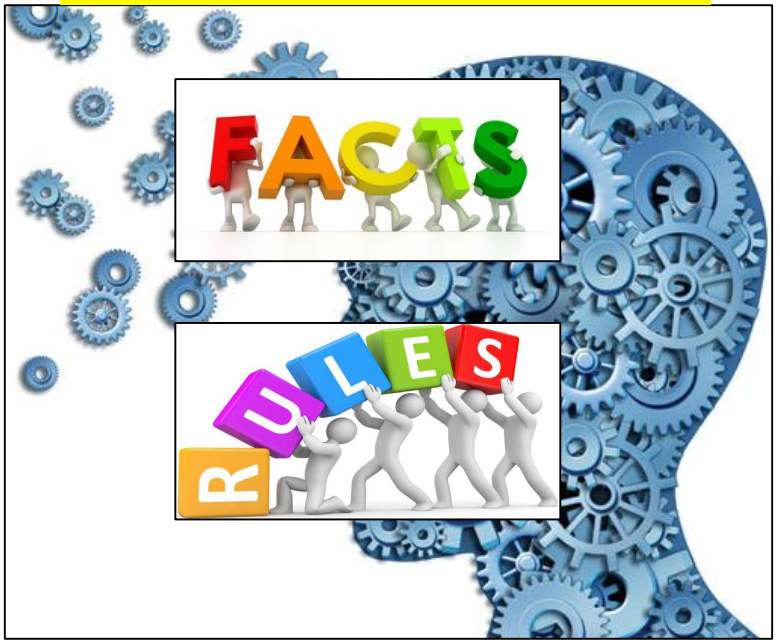


- Beginning with lower case letters.
- End the fact with period (full stop).

Introduction-PROLOG

Clauses

Knowledge Base



Prolog programming is all about writing knowledge base. We fire the query on the knowledge base

Introduction-PROLOG -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

- 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.

EXAMPLES

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

Introduction-PROLOG -RULES

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

Introduction-PROLOG –KNOWLEDGE BASE(CLAUSES)

- reads_a_book(saeed).
- happy(khalid).

FACTS

HEAD OF CLAUSE

NECK OF CLAUSE

BODY OF CLAUSE

- author(saeed) :- reads_a_book(saeed).
- reads_a_book(khalid):- happy(khalid).
- author(khalid) :- reads_a_book(khalid).

RULES

part on the left hand of the " :- " is called the head

part on the right hand of the " :- " is called the body;

rule-> **head :- body**

General Rule: If the body of the rule is true then head implies true

BASIC EXAMPLE

```
dog(fido).
dog(rover).
dog(henry).
cat(felix).
cat(michael).
cat(jane).
animal(X):-dog(X).
```

FACT: Iinterpretation that fido, rover and henry are all dogs.

FACT: Interpretation that felix, michael and jane are all cats

RULE: Saying that anything (let us call it X) is an animal if it is a dog.

- Cat lovers may feel that cats can also claim to be called animals, but the program is silent about this.
- Load the program
- You will come to system prompt, symbol ?-
- To check whether fido is a dog all that is necessary is to type the query dog(fido) followed by a full stop and press the 'return' key

?- dog(fido).

BASIC EXAMPLE continued.....

?-dog(jane). **no**	[Is jane a dog? No - a cat]
?- animal(fido). **yes**	[Is fido an animal?] [yes - because it is a dog and any dog is an animal]
?- dog(X). **X = fido ;** **X = rover ;** **X = henry**	[Is it possible to find anything, let us call it X, that is a dog?] [All 3 possible answers are provided]
?-animal(felix). **no**	[felix is a cat and so does not qualify as an animal, as far as the program is concerned]

- Example shows the two components of any Prolog program, rules and facts
- The use of queries that make Prolog search through its facts and rules to work out the answer.

BASIC EXAMPLE continued.....

?-dog(jane).	[Is jane a dog? No - a cat]
no	
?- animal(fido).	[Is fido an animal?]
yes	[yes - because it is a dog and any dog is an animal]
?- dog(X).	[Is it possible to find anything, let us call it X, that is a dog?]
X = fido ;	
X = rover ;	
X = henry	[All 3 possible answers are provided]
?-animal(felix).	[felix is a cat and so does not qualify as an animal, as far as the program is concerned]
no	

GIVEN THAT
any X is an animal if it is a dog

AND
fido is a dog

DEDUCE
fido must be an animal

Introduction-PROLOG –EXAMPLE

FACTS	ENGLISH MEANING
food(burger).	// burger is a food
food(sandwich).	// sandwich is a food
food(pizza).	// pizza is a food
lunch(sandwich).	// sandwich is a lunch
dinner(pizza).	// pizza is a dinner
RULES	
meal(X) :- food(X).	// Every food is a meal OR Anything is a meal if it is a food
QUERIES/GOALS	
?- food(pizza).	// Is pizza a food?
?- meal(X), lunch(X).	//Which food is meal and lunch?
?- dinner(sandwich).	//Is sandwich a dinner

DATA OBJECTS IN PROLOG

NUMBERS

- All versions of Prolog allow the use of integers (whole numbers).
- They are written as any sequence of numerals from 0 to 9
- Optionally preceded by a + or – sign

Ex: 623, -47 ,+5 , 025

- Most versions of Prolog also allow the use of numbers with decimal points.
- They are written in the same way as integers, but contain a single decimal point, anywhere except before an optional + or – sign

Ex: 6.43 , -.245 , +256.

DATA OBJECTS IN PROLOG continued...

ATOMS

- Atoms are constants that do not have numerical values. There are three ways in which atoms can be written

```
john
today_is_Tuesday
fred_jones
a32_BCD

but not

Today
today-is-Tuesday
32abc
```

Any sequence of one or more letters (upper or lower case), numerals and underscores, beginning with a lower case letter

```
'Today is Tuesday'
'today-is-Tuesday'
'32abc'
```

Any sequence of characters enclosed in single quotes, including spaces and upper case letters, e.g.

Examples

```
+++
>=
>
+--
```

Any sequence of one or more special characters from a list that includes the following + - * / > < = & # @ :

DATA OBJECTS IN PROLOG continued...

VARIABLES

- In a query a variable is a name used to stand for a term that is to be determined

- The name of a variable is denoted by any sequence of one or more letters (upper case), numerals and underscores, beginning with an upper case letter or underscore, e.g.

```
X
Author
Person_A
_123A

but not

45_ABC
Person-A
author
```

4.1: Resolution and Unification

Unification

- When a goal is given to evaluate, Prolog will work through the clauses in the database.
- It attempts to match the goal with each clause.
- The matching process works from left to right.
- The goal will fail if no match is found. If a match is found, the action will be taken.
- Prolog uses a matching technique called unification technique
  - Here one or more variables being given value to make the two call terms identical.
  - This process is called binding the variables to values.
  - For example, Prolog can unify the terms cat(A), and cat(mary) by binding variable A to atom mary that means we are giving the value mary to variable A

UNIFICATION

- Definition: Unification is the process in which one or more variables being given values in order to make two call terms identical. This is known as binding the variables to values.

- dog(X) and dog(fido)

Can be unified by binding variable X to atom fido, giving X the value fido

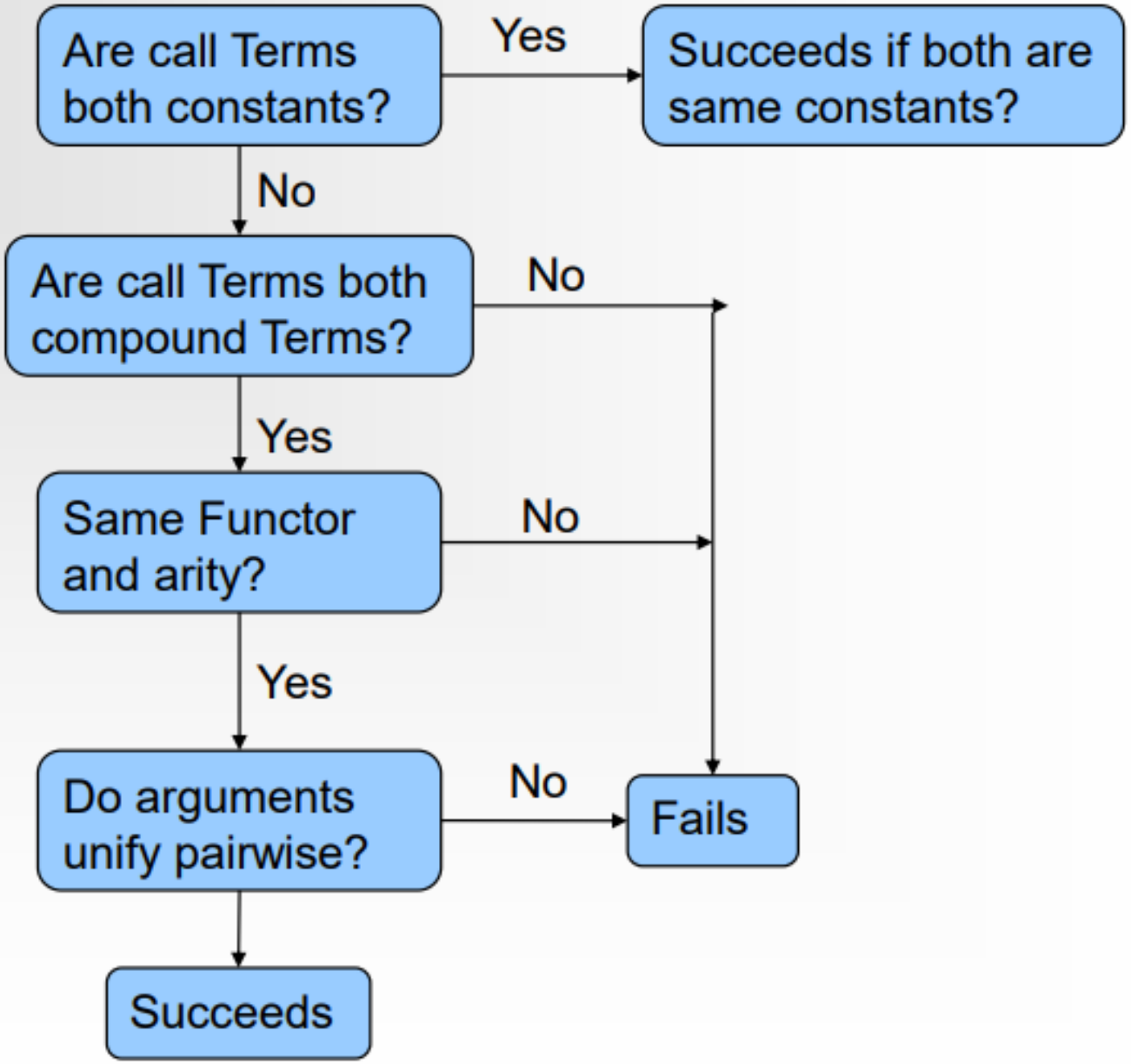
- owns(john,fido) and owns(P,Q)

Can be unified by binding variables P and Q to atoms john and fido, respectively.

- parent(albert, adward) and parent(victoria, Y)

Cannot be unified as albert and victoria do not unify

Unification Algorithm



Unification

- Unification is like assignment
- It is kind of like equality
- It is also like pattern matching
- The explicit unification operation is =
- Any value can be unified with itself E.g. Mary=Mary
- Any variable can be unified with other variable E.g. X=Y
- A variable can be unified with any other Prolog value

E.g. Season = weather(sunny).

Here variable Season is instantiated to value weather(sunny)

- Thus, two terms unify if they are same term or if they contain variable that can be uniformly instantiated with the term in such a way that the resulting terms are equal

Unification

- Two different structures can be unified if their constituents can be unified

female(X) = female(jane).

Here X is unified with jane

- A variable can be unified with a structure containing that same variable
- Look at the following example
  - mother(mary, X) = (Y, father(Z)).
  - means X = father(Z)
- Following examples of no unification

vincent and mia do not unify

woman(mia). and woman(jody). will not unify

Unification

- Unification is symmetric means

Steve = father(isaac) is same as father(isaac) = Steve

- In well written code most unification happens implicitly as a result of parameter matching.
- Suppose in database there is a fact..

teaches(john, nora).

- If we query the database as..

?-teaches(X, nora).

- X and john are unified
- When Prolog unifies two terms, it performs all the necessary instantiations, so that the terms are equal afterwards

Unification Examples

```
?-mia = mia
yes
?-43 = 43
yes

?-mia=vincent
no

?-mia=X
X=mia

?-X=mia,
X=Vincent
no
```

```
?- k(s(g), Y) = k(X, t(k)).
X = s(g)
Y = t(k)
Yes

?- k(s(g), t(k)) = k(X, t(Y)).
X = s(g)
Y = k
Yes

?- vertical(line(point(X,Y), point(X,Z))).
?- horizontal(line(point(X,Y), point(Z,Y))).
?-vertical(line(point(1,1), point(1,3))).
yes
?-vertical(line(point(1,1), point(3,2))).
no
?- horizontal(line(point(1,1), point(1,Y))).
```

Unification Examples

Ex1

- Facts :

  likes(john, jane).

  likes(jane, john).

- Query :

  ?- likes(john, X).

  Answer : X = jane.

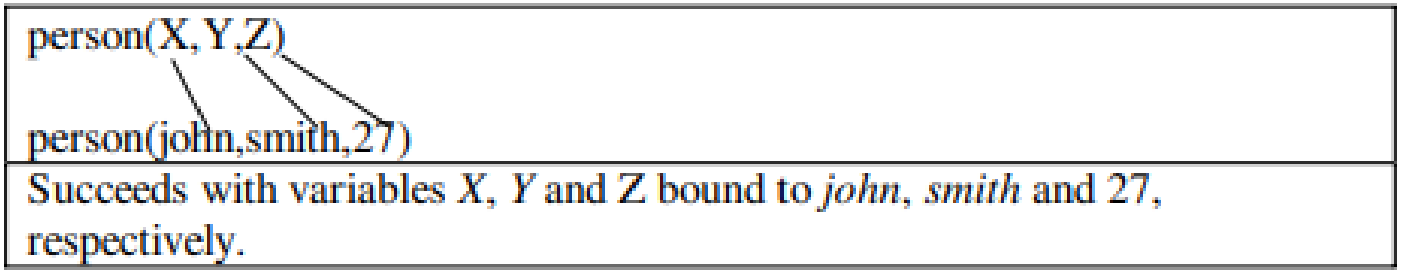
- Here upon asking the query first prolog start to search matching terms in 'Facts' in top-down manner for 'likes' predicate with two arguments and it can match likes(john, ...) i.e. Unification.
- Then it looks for the value of X asked in query and it returns answer

X = jane i.e. Instantiation - X is instantiated to 'jane'.

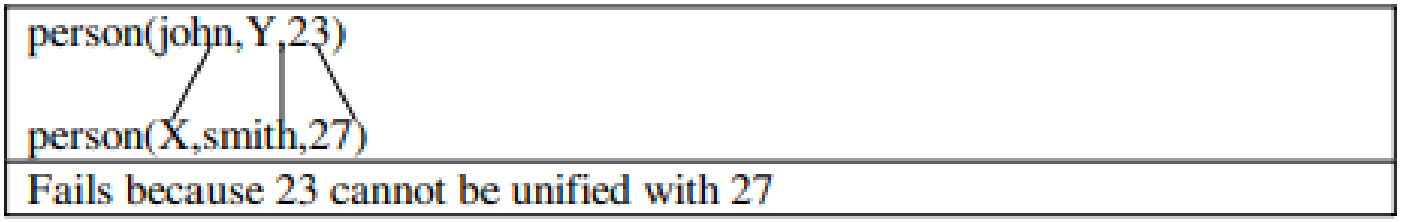
Ex2:

?- owns(X, car(bmw)) = owns(Y, car(C)).

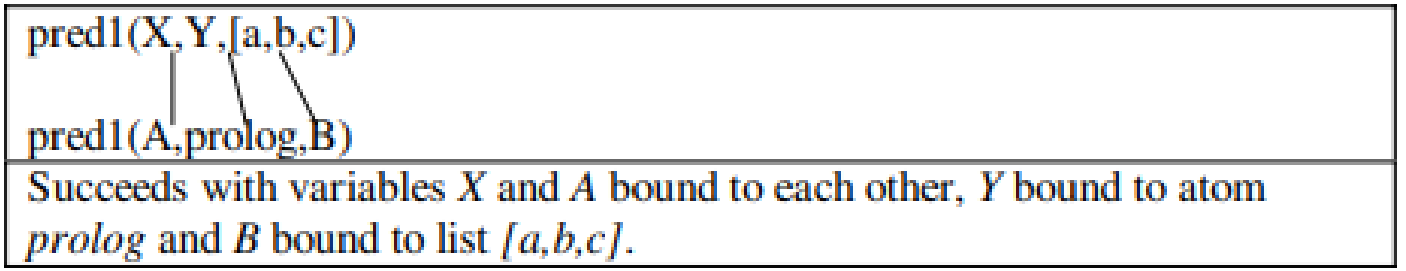
You will get Answer : X = Y, C = bmw.



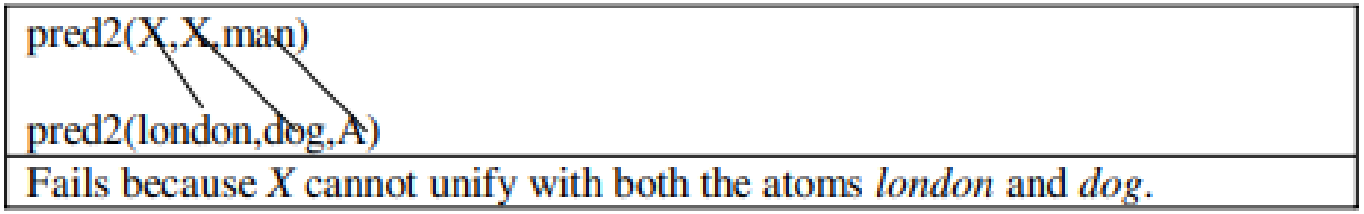
Succeeds with variables *X*, *Y* and Z bound to *john*, *smith* and 27, respectively.



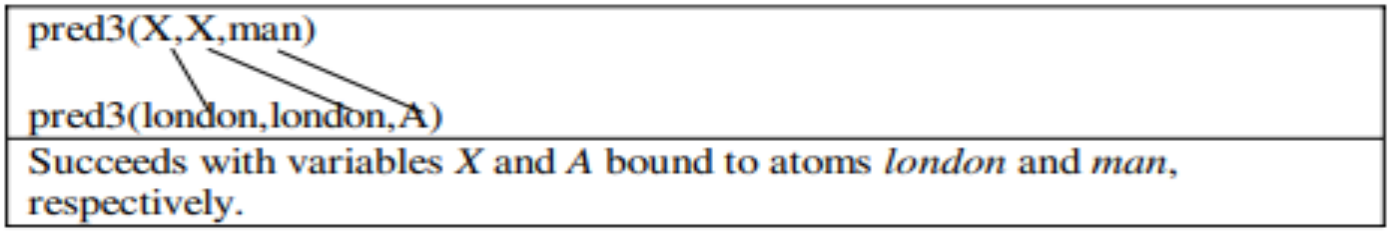
Fails because 23 cannot be unified with 27



Succeeds with variables *X* and *A* bound to each other, *Y* bound to atom *prolog* and *B* bound to list *[a,b,c]*.



Fails because *X* cannot unify with both the atoms *london* and *dog*.



Succeeds with variables *X* and *A* bound to atoms *london* and *man*, respectively.

RESOLUTION

- Resolution in Prolog is basically the inference mechanism

- Means, when you resolve two clauses you get one new clause.

- (1) *All women like shopping.* (2) *Olivia is a woman*. Now we ask query 'Who likes shopping'. So, by resolving above sentences we can have one new sentence *Olivia likes shopping*.

- Formally a clause is defined as an expression of the form
- $(A_1A_2 \ldots A_n) \llcorner (B_1B_2 \ldots B_n)$

- Where each of the As and Bs stand for proposition
-
- A clause is used to express a logical implication of the following sort-
- -If all of Bs are true then one of the As must be true as well

- Resolution in Prolog is basically the inference mechanism

Resolution

- Resolution is inference mechanism.
- Derivation of new statement is called Resolution
- Consider two clauses

m :- b.

t :- p, m, z.

- So from that we can infer

t :- p, b, z.

- that is called resolution.
- Means, when you resolve two clauses you get one new clause.
- Another example, we have two sentences

All women like shopping.

Olivia is a woman.

- Now we ask query 'Who likes shopping'.
- So, by resolving above sentences we can have one new sentence

Olivia likes shopping.

C :- A, B — A and B imply C

D :- C

D :- A, B — If we know that A and B imply C, and that C implies D, then we can deduce that A and B imply D

Resolution Examples

flowery(X) :- rainy(X). Predicate applied to a variable

rainy(Mumbai). Predicate applied to atom

- From above two clauses we can infer third clause

flowery(Mumbai).

```
?- pwd.
% d:/documents/prolog/
true.
```

To check current path

```
?- cd('D:/PCPF/AY-2023-24/Course Lab/Prolog').
true.
```

To change current path

```
?- pwd.
% d:/pcpf/ay-2023-24/course lab/prolog/
true.

?-
```

prolog1.pl - Notepad

File Edit Format View Help

```
woman(mia).
woman(jody).
woman(yolanda).
playsairguitar(jody).
men(bheem).
men(chotu).
playscricket(chintu).
```

% d:/pcpf/ay-2023-24/course lab/prolog/
true.

?- [prolog1].
true.

?- woman(mia).
true.

?- woman(jody).
true.

?- woman(pia).
false.

?- women(mia).
Correct to: "woman(mia)"? yes
true.

?- playsairguitar(jody).
true.

?- men(chotu).
true.

?- men(chotu1).
false.

?-

prolog2.pl - Notepad

File Edit Format View Help

```
happy(yolanda).
happy(jerry).
happy(tom).
sad(bheem).
sad(chutki).
listens2music(mia).
listens2music(yolanda):-happy(yolanda).
playsairguitar(mia):-listens2music(mia).
playsairguitar(yolanda):-listens2music(yolanda).
```

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File Edit Settings Run Debug Help

true.

?- happy(tom).
true.

?- sad(bheem).
true.

?- sad(team).
false.

?- happy(X).
X = yolanda .

?- happy(X).
X = yolanda ;
X = jerry ;
X = tom.

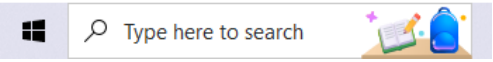
?-

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File Edit Settings Run Debug Help

X is 3 .

```
?- X is 3.
X = 3.

?- X is 17.
X = 17.

?- X is sqrt(26).
X = 5.0990195135927845.

?- X is sqrt(36).
X = 6.0.

?- X is sqrt(36)+2.
X = 8.0.

?- sqrt(36)+4=:=5*11-45.
true.

?-
```



END

UNIT-IV: Declarative Programming Paradigm: Logic Programming

Faculty In-charge

Mrinmoyee Mukherjee
Assistant Professor (IT Dept.)
email: mrinmoyeemukherjee@sfit.ac.in
Mob: 9324378409

Academic Year: 2023-24

4.2: Lists and Arithmetic Execution Order

Prolog Symbols and Operators

English	Prolog
If	:-
Not	Not
Or (disjunction)	;
And (conjunction)	,

Operator	Meaning
X > Y	X is greater than Y
X < Y	X is less than Y
X >= Y	X is greater than or equal to Y
X =< Y	X is less than or equal to Y
X =:= Y	the X and Y values are equal
X =\= Y	the X and Y values are not equal

LIST

1. In Prolog LIST refers to an ordered sequence of elements

2. A single element in a list can be represented as [a]

3. An empty list can be represented as []

4. The elements of lists are separated by commas

5. Compound lists are also possible

6. [first,second,third] = [A|B]

where *A = first* and

B=[second,third]

The unification here succeeds. A is bound to the first item in the list, and B to the remaining list.

LIST continues....

- [ ] /* this is a special list, it is called the empty list because it contains nothing */

- Now lets consider some comparisons of lists:
- [a,b,c] unifies with [Head|Tail] resulting in Head=a and Tail=[b,c]

- [a] unifies with [H|T] resulting in H=a and T=[]

- [a,b,c] unifies with [a|T] resulting in T=[b,c]

- [a,b,c] doesn't unify with [b|T]

- [] doesn't unify with [H|T]

- [] unifies with []. Two empty lists always match

Operations on List

Operations	Definition
Membership Checking	During this operation, we can verify whether a given element is member of specified list or not?
Length Calculation	With this operation, we can find the length of a list.
Concatenation	Concatenation is an operation which is used to join/add two lists.
Delete Items	This operation removes the specified element from a list.
Append Items	Append operation adds one list into another (as an item).
Insert Items	This operation inserts a given item into a list.

Operations on List

- Membership Checking
  - whether a member X is present in list L or not?
  - Consider the predicate name is list_member(X, L).
- To design this predicate, we can write program by checking
- X is a member of L if either −
  - X is head of L, or
  - X is a member of the tail of L
- The prolog program

```
list_member(X, [X| _ ]).
list_member(X, [ _ | TAIL]) :- list_member(X, TAIL).
```

?- list_member(b, [a, b, c]).	Yes
?- list_member(b, [a, [b,c]]).	?
?- list_member([b, c], [a, [b,c]]).	Yes
?- list_member(d, [a,b,c]).	No

LIST continues....

```
?- X=23,Y=24,Z=[alpha,beta],write('List is: '),write([X,Y,Z]),nl.
List is: [23,24,[alpha,beta]]
X = 23,
Y = 24,
Z = [alpha, beta].

?-
```

- Write([1,2,3])
- member(x, [x,y,z]).
- member(p, [x,y,z]).
- member(my(x,y,z),[q,r,s,my(x,y,z),w]).
- member(v,[]).
- length([a,b],M).
- length([1,2,3],M).
- length([1,2,3],a).
- length([1,2,3],X1).
- length([1,2,3],X-1).
- length([[a,c],[e,f],[h,i]],N).
- length([],P).
- length([a,b,c],3).

- reverse([1,2,3],A).
- reverse(B, [1,2,3]).
- reverse([[dog,cat],[1,2],[bird,mouse]],L).
- reverse([1,2,3,4],[4,3,6,8]).
- reverse([1,2,3,4],[4,3,2,1]).
- append([],[1,2,3],L).
- append([a,b],[1,2,3],L).
- append([a,b,23],[1,2,3],L).

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File Edit Settings Run Debug Help

```
?- write([1,2,3]).
[1,2,3]
true.

?- member(x,[x,y,z]).
true .

?- member(p,[x,y,z]).
false.

?- member(my(x,y,z),[q,r,s,my(x,y,z),w]).
true .

?- member(my(x,y,p),[q,r,s,my(x,y,z),w]).
false.

?- member(v,[]).
false.

?-
```

Type here to search

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

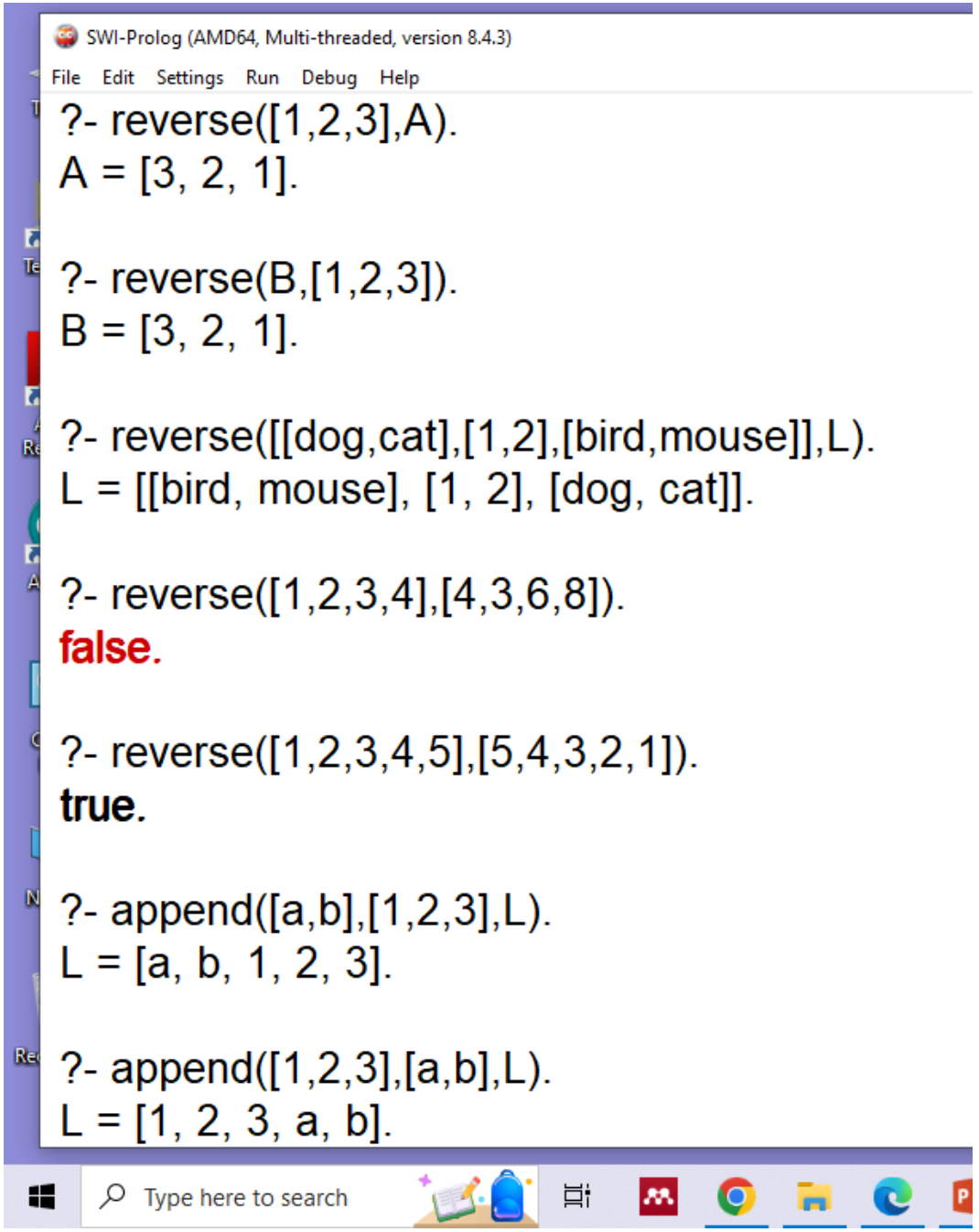
File Edit Settings Run Debug Help

```
?- length([a,b],M).
M = 2.

?- length([1,2,3],M).
M = 3.

?- length([1,2,3],M-1).
ERROR: Type error: `integer' expected, found `_9032-1' (a
compound)
ERROR: In:
ERROR:   [11] throw(error(type_error(integer,...),context(
...,_9096)))
ERROR:    [9] toplevel_call('<garbage_collected>') at c:/p
rogram files/swipl/boot/toplevel.pl:1158
ERROR:
ERROR: Note: some frames are missing due to last-call op
mization.
ERROR: Re-run your program in debug mode (:- debug.) tc
t more detail.
```

Slide 42 of 52

Type here to search

```
?- length([[a,c],[e,f],[h,i]],N).
N = 3.

?- length([],N).
N = 0.

?- length([[],[],[],[]],N).
N = 4.

?- length([a,b,c],3).
true.
```



Length of List

- Length Calculation:
- lengthList(L,N).
- If list is empty, then length is 0.
- If the list is not empty, then L = [Head|Tail], then its length is 1 + length of Tail.
- The prolog program

```
lengthList([], 0).
lengthList([ _ | TAIL], N) :- lengthList(TAIL, N1), N is N1+1.
```

- EX.

```
?-lengthList([a,b,c,d,e,f,g,h,i,j], Len).
Len=10
?-lengthList([[a,b],[c,d],[e,f]],Len).
Len=3
```

Concatenation of Lists

- Concatenation
- If the first list is empty, and second list is L, then the resultant list will be L.
- If the first list is not empty, then write this as [Head|Tail], concatenate Tail with second list L2 recursively, and store into new list in the form, [Head|New List].

```
list_concat([], L, L).
list_concat([X1 | L1], L2, [X1 | L3]) :- list_concat(L1, L2, L3).
```

- EX.

```
?-list_concat([1,2], [a,b,c], NewList).
NewList = [1,2,a,b,c]
?-list_concat([[1,2,3],[p,q,r]],[a,b,c],NewList).
NewList = [1,2,3,p,q,r,a,b,c]
```

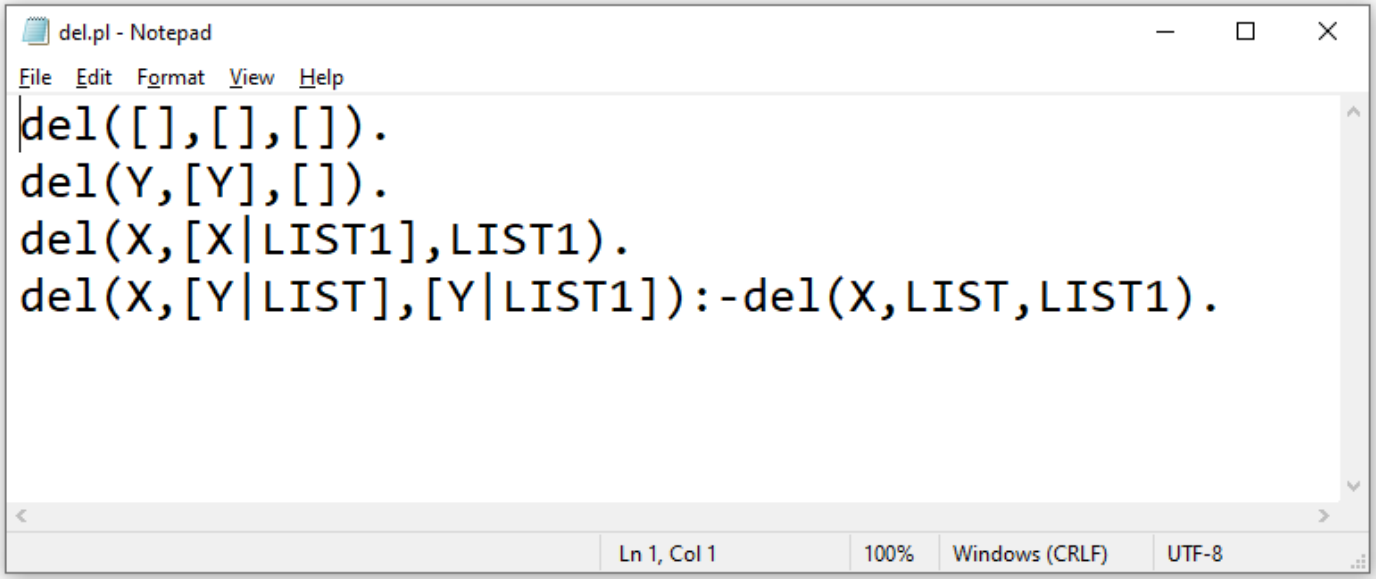
https://www.youtube.com/watch?v=syGMNm25mfY

Delete From List

- Delete element from list
  - If X is the only element, then after deleting it, it will return empty list.
  - If X is head of L, the resultant list will be the Tail part.
  - If X is present in the Tail part, then delete from there recursively.
- The Prolog Program

```
list_delete(X, [X],  []).
list_delete(X, [X | L1], [L1]).
list_delete(X, [Y | L2], [Y|L1]) :- list_delete(X, L2, L1).
```

```
?- del([],[],L).
L = [] ;
false.

?- del([2],[2],L).
false.

?- del(2,[2],L).
L = [] ;
L = [] ;
false.

?- del(3,[3,4,5,6],L2).
L2 = [4, 5, 6] ;
false.

?- del(4,[1,2,3,4],X).
X = [1, 2, 3] ;
X = [1, 2, 3]
```



Arithmetic examples	**Prolog Notation**
$6+2=8$	`8 is 6+2.`
$6*2=12$	`12 is 6*2.`
$6-2=4$	`4 is 6-2.`
$6-8=-2$	`-2 is 6-8.`
$6\div 2=3$	`3 is 6/2.`
$7\div 2=3$	`3 is 7/2.`
1 is the remainder when 7 is divided by 2	`1 is mod(7,2).`

- X is 3+2.
- x is 3+2.
- 3+2=X.
- X is +(3,2).
- 5 is 3+2.
- 3+2 is 5.
- X is 3*2.
- X is 3-2.
- X is -(2,3).
- X is 5-3-1.
- X is -(5,3,1).
- X is -(-(5,3),1).
- X is 3/5.
- X is 3 mod 5.
- X is 5 mod 3.
- X is 5^3.
- X is (5^3)^2.
- X = (5^3)^2.
- 25 is 5^2.
- Y is 3+2*4-1.
- Y is (3+2)*(4)-(1).
- Y is -(*(+(3,2),4),1).
- X is 3*2, Y is X*2.
- X is abs(9).
- X is sin(90).
- X is cos(90).
- X is max(3,4).
- X is min(3,4).
- X is sqrt(49).

- 5=\=5.
- 5==5.
- 5>=5.
- 5>5.

When there is more than one operator in an arithmetic expression, e.g. **A+B*C-D**

-calculate the product of B and C

-add it to A and then subtract D

Usage of Operators

- Arithmetic calc program
- X is 100 + 200,write('100 + 200 is '),write(X),nl.
- Y is 400 - 150,write('400 - 150 is '),write(Y),nl.
- Z is 10 * 300,write('10 * 300 is '),write(Z),nl.
- A is 100 / 30,write('100 / 30 is '),write(A),nl.
- B is 100 // 30,write('100 // 30 is '),write(B),nl.
- C is 100 ** 2,write('100 ** 2 is '),write(C),nl.
- D is 100 mod 30,write('100 mod 30 is '),write(D),nl.

SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File Edit Settings Run Debug Help

```
400-150 is 250
Y = 250.

?- Z is 10*300,write('10*300 is '),write(Z),nl.
10*300 is 3000
Z = 3000.

?- A is 100/30,write('10/30 is '),write(A),nl.
10/30 is 3.3333333333333335
A = 3.3333333333333335.

?- B is 100//30,write('100//30 is '),write(B),nl.
100//30 is 3
B = 3.

?- C is 100**2,write('100**2 is '),write(C),nl.
100**2 is 10000
C = 10000.

?-
```

Usage of Operators

- X+Y, the sum of X and Y
- X-Y, the difference of X and Y
- X*Y, the product of X and Y
- X/Y, the quotient of X and Y
- X^Y, X to the power of Y
- -X, the negative of X
- abs(X), the absolute value of X
- sin(X), the sine of X (for X measured in degrees)
- cos(X), the cosine of X (for X measured in degrees)
- max(X,Y), the larger of X and Y
- sqrt(X), the square root of X
- << >> Left and right shift

Database Manipulation

- Prolog has four database manipulation commands: assert, retract, asserta, and assertz.

- To assert or insert the facts in the database or knowledge-base

?- listing.

yes (It means database is empty)

- To add record to database

?- assert(happy(mia)).

yes

?- listing.

happy(mia)

?- assert(happy(vincent)).

yes

Database Manipulation

```
?- assert(happy(marcellus)).
  yes
?- assert(happy(john)).
  yes
?- assert(happy(vincent)).
  yes
?- assert ( (naive(X):- happy(X)) ).
yes
  ?- listing.
  happy(mia).
  happy(vincent).
  happy(marcellus).
  happy(john).
  happy(vincent).
  naive(A):- happy(A).
  yes
```

Database Manipulation

- To remove information predicate retract is used

```
?-retract(happy(marcellus)).
?-listing.
happy(mia).
happy(vincent).
happy(john).
happy(vincent).
naive(A)  :- happy(A).
```

- To remove all of our assertions

```
?-retract(happy(X)).
X = mia ;
X = john ;
X = vincent ;
no
 ?- listing.
naive(A)  :-  happy(A).
```

Database Manipulation

- If we want more control over where the asserted material is placed, there are two variants of assert/1 , namely:
- assertz. Places asserted material at the end of the database.
- asserta. Places asserted material at the beginning of the database.
- For example, suppose we start with an empty database, and then we give the following command:
- assert(p(b)), assertz(p(c)), asserta(p(a)).
- Then a listing reveals that we now have the following database:

```
?- listing.
p(a).
p(b).
p(c).
yes
```

END

UNIT-IV: Declarative Programming Paradigm: Logic Programming

Faculty In-charge

Mrinmoyee Mukherjee
Assistant Professor (IT Dept.)
email: mrinmoyeemukherjee@sfit.ac.in
Mob: 9324378409

Academic Year: 2023-24

Looping a fixed number of times.....

- Most programming languages provide loops that allow a set of instructions to be executed repeatedly either a fixed number of times or until a given condition is met

- Prolog has no looping facilities.

- The same effects can be obtained (that enable a sequence of events to be evaluated repeatedly) through backtracking, recursion, built in predicates or a combination of both.

loops.pl [modified]

File Edit Browse Compile Prolog Pce Help

loops.pl [modified]

```
loop(0).
loop(N):-N>0,
    write('The value is: '),
    write(N),
    nl,
    M is N-1,
    loop(M).
```

Will display the o/p fixed number of times

SWI-Prolog (AMD64, Multi-threaded, versi

File Edit Settings Run Debug Help

```
?- loop(9).
The value is: 9
The value is: 8
The value is: 7
The value is: 6
The value is: 5
The value is: 4
The value is: 3
The value is: 2
The value is: 1
true
```

Applying recursion.....

sum.pl

File Edit Browse Compile Prolog Pce Help

loops.pl sum.pl

```
/*sum the integers from 1 to N*/
 sumto(1,1).
sumto(N,S):-N>1,
    N1 is N-1,
    sumto(N1,S1),
    S is S1+N.
```

?- sumto(5,N).
N = 15

?- sumto(1,3).
false.

?- sumto(1,1).
true

Ex5 Recursion: Loop till condition satisfied

- Recursion Example5
- **Knowledge base**

```
go :- loop(start).
loop(end).
loop(A) :- A\=end, write('The value is:'), read(Word),
        write('Input value is: '), write(Word), nl, loop(Word).
```

- **Queries**

```
?- go.
The value is:
Input value is a
The value is:
Input value is: hi
The value is:
Input value is end
yes
```

Conjunction and Backtracking

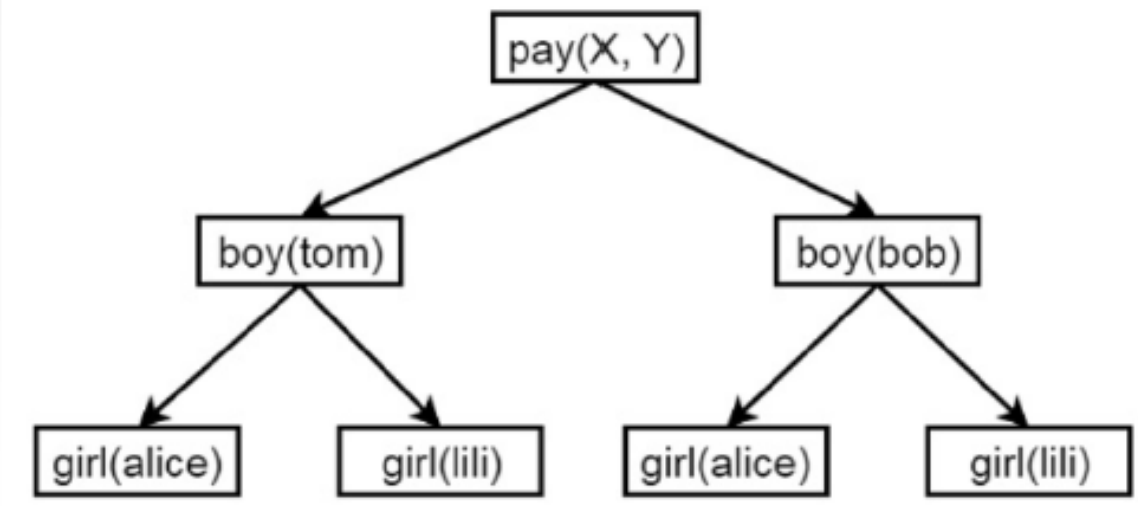
- Conjunction:
  - Conjuction means 'and' it is indicated by use of comma ',' . Ex:

  ?- likes(john, mary), likes(mary, john).

  - Here we have given two subgoals in one query using conjunction.
  - For answering the query as success Prolog has to match and satisfy each subgoals with the knowledge base (i.e. Facts and Rules).
  - When goals are given as conjunction, Prolog finds the match and satisfy goals in conjunction, in left-to-right manner.
- Backtracking:
  - Prolog repeatedly tries to find match and satisfy the goals by looking to the knowledge base in top-down manner which is nothing but backtracking.
  - To control Backtracking , Cut operator ' ! ' is used
- For above mentioned query, prolog will first match and satisfy the left most goal i.e. likes(john, mary). and then the second goal.
- If any of subgoals does not satisfy, then query will be answered as failure.

Backtracking Ex

- Consider two people X and Y can pay each other, but the condition is that a boy can pay to a girl
- **Knowledge Base:**
- boy(tom).
- boy(bob).
- girl(alice).
- girl(lili).
- pay(X,Y) :- boy(X), girl(Y).



Cut in Backtracking

- Sometimes we write the same predicates more than once when our program demands, in such cases uncontrolled backtracking may prove inefficient.
- To resolve this, we will use the Cut in Prolog.
- Ex: Consider we have three mutually exclusive rules and any given time only one of them ill be true.
- **Knowledge Base:**

```
f (X,0) :- X < 3.          % Rule 1
f (X,2) :- 3 =< X, X < 6.  % Rule 2
f (X,4) :- 6 =< X.         % Rule 3
```

```
f (X,0) :- X < 3, !.          % Rule 1
f (X,2) :- 3 =< X, X < 6, !.  % Rule 2
f (X,4) :- 6 =< X.            % Rule 3
```

- **Queries**:

?- f(1,Y), 2<Y.

- There are two subgoals to be satisfied. As per first fact X=1 so Y will be set to 0, but now second clause or query fails since Y is not >2.
- So Prolog backtracks and goes rule 2, here since X is not between 3 and 6, first query clause itself fails. Prolog backtracks and checks third rule, the goal fails here too. To avoid such backtracking cut is used.

Example-2 of Cut

- **Ex: Knowledge Base:**
- animal(dog).
- animal(cat).
- animal(elephant).
- animal(tiger).
- animal(cobra).
- animal(python).

- snake(cobra).
- snake(python).
- likes(mary, X) :- snake(X).
- likes(mary, X) :- animal(X).

- **Queries:**

?- likes(mary, X).

Use semicolon to see different X values

- **Ex: Knowledge Base: (Using cut)**
- animal(dog).
- animal(cat).
- animal(elephant).
- animal(tiger).
- animal(cobra).
- animal(python).

- snake(cobra).
- snake(python).
- likes(mary, X) :- snake(X), !.
- likes(mary, X) :- animal(X).

- **Queries:**

?- likes(mary, X).

Use semicolon to see different X values

PROLOG facilities and deficiencies

- Advantages :
  1. Easy to build database. Doesn't need a lot of programming effort.
  2. Pattern matching is easy. Search is recursion based.
  3. It has built in list handling. Makes it easier to play with any algorithm involving lists.
- Disadvantages :
  1. LISP (another logic programming language) dominates over prolog with respect to I/O features.
  2. Sometimes input and output is not easy.
- Applications :
  1. Prolog is highly used in artificial intelligence(AI).
  2. Prolog is also used for pattern matching over natural language parse trees.

Deficiencies of Prolog

Resolution Order Control:

- Prolog always matches in the same order – user can control the ordering
- Prolog allows explicit control of backtracking using cut which is actually, a goal and not an operator. It always succeeds but can not be resatisfied through backtracking.

The Closed World Assumption:

- In Prolog truths are those that can be proved using its database
- If there is insufficient information in database, to prove a query, it is not actually false, it fails.
- It relates to negation problem.

END