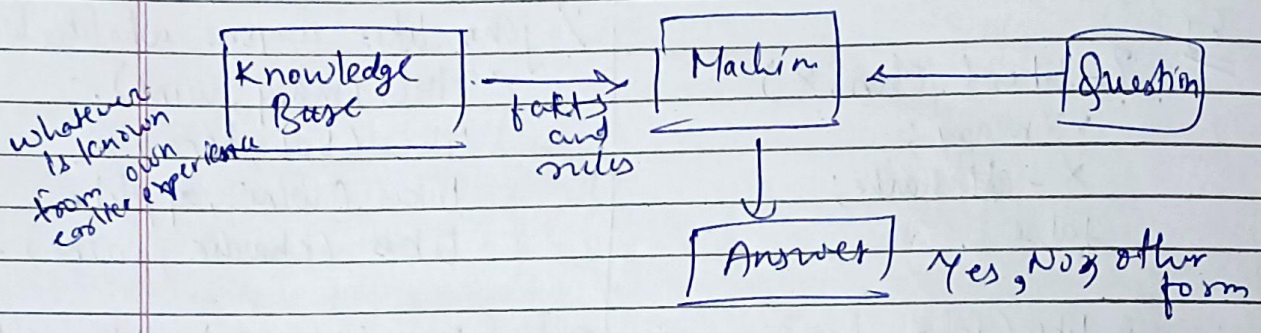
Page No.
Date

~~what to do~~ Prolog → logic programming

John owns the book
person owns relationship object



① Facts & → means rules

small letter

likes (john , mary) .
relationship obj1 obj2

?- owns (mary, book).
No.

② Variables

capital letter

?- likes (john, X)
X = flower ;
X = chicken .

var.pl
likes (john, flower).
likes (john, chicken).
hates (john, oranges).

③ Conjunctions (AND , OR)

AND → ,
OR → ;

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

if → :-

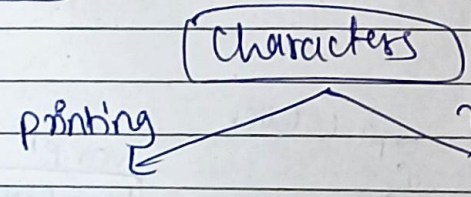
isbird(X) :- animal(X), feathers(X).
issister(X,Y) :- female(X), parent(X,M,F), parents(Y,M,F)

Eg
?- likes(john, X)
X = mary;
X = charlie;
false

?- likes(john, bob)
false

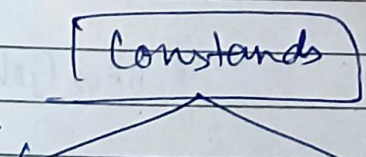
% john likes anyone who likes wine
facts { likes(mary, wine).
likes(bob, beer).
likes(wine, apple).
likes(charlie, wine). }
rule { likes(john, X) :- likes(X, wine). }

⊙ Characters



Uppercase	A, B, C, ..., Z	Blank space
Lowercase	a, b, c, ..., z	New line
digits	0, 1, 2, ... 9	Beeping sound
special character	+, -, /, \, ~, ^, <, >, ...	

⊙ Constants



-17
+8
1.01e2 = 1.01 * 10^2
1.01e-2 = 1.01 * 10^{-2}

*-+/\

letter + digits (begin with lowercase)
uxc2018 ✓
2018uxc ✗
'2018uxc' ✓

Variables

Name begins with uppercase letter or underscore

Eg. Answer ✗
_name ✓
_3_musketeers ✓

Anonymous Variable.

hates(X, oswald).
X = father

anonymous → hates(_, oswald).
true

hates(father, oswald).
hates(mother, neighbour)

Structure — collection of objects

↳ used when we need to organize data into a single entity

Eg. structure : library card
attribute ⇒ { Author name, Title, Date, location etc.

They mean the same book

owns(john, books).
owns(mary, book).

∴ How to diff ?

by writing book name

owns(john, wonder).
owns(mary, alienist).

Nested (author)

owns(john, book(wonder, palacio)).
owns(mary, book(alienist, carr)).

How to query ?

owns(john, book(X, author(Y, Palacio)))

o/p : X = wonder
Y = Raquel

first name

owns(john, book(wonder, author(Raquel, Palacio)))
owns(mary, book(alienist, aut...

Alternative method using Anonymous

owns (john, book (_, author (_, carr)))

① Operator Precedence

$x + y$ → infix

$x + y * z$ → $x + (y * z)$

Prolog : $+(x, *(y, z))$

(lowest precedence, highest)

Eg $2 + 8/2 * 3$

$+(2, *(/(8,2), 3))$

lower value → higher precedence

② Operator Associativity

$x/y/z \Rightarrow (x/y)/z$

Prolog : $/(/(x,y), z)$

Eg $8/2/2 \Rightarrow (8/2)/2$

$/(/(8,2), 2)$

① Equality & Unification

?- x = y → True (if they unify)
          → False (otherwise)

built-in

Rules

(i) X & Y are permitted to be unistantiated variable
(ii) Integers & atoms are always equal to themselves
(iii) 2 structure are equal only if:
  (a) same function
  (b) n No. of components
  (c) corresponding components are equal

$$a(b, c(d, e)) = a(X, c(Y, e))$$

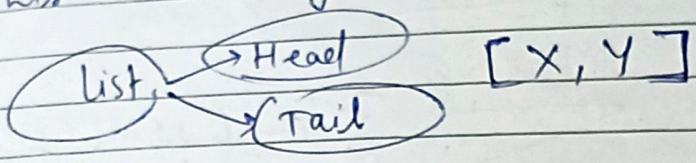
X = b
Y = d

f(X,X) = f(a,b) false
?- 'Student' = student true
?- f(X, a(b,c)) = f(Z, a(Z, c)). X = Z, Z = b

② Arithematic Operators

X =:= Y	equal	X + Y	
X =\= Y	not equal	X - Y	
X < Y		X * Y	
X > Y		X / Y	Quotient
X =< Y	less than =	X // Y	Integer Quotient
X >= Y	greather than =	X mod Y	remainder

⊙ List in Prolog.



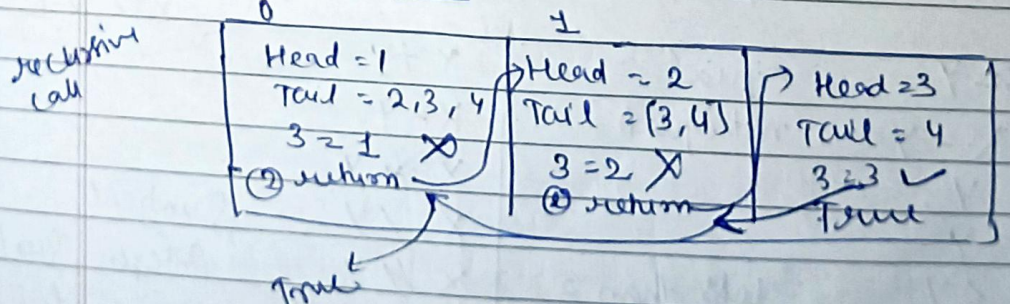
[1, 2, 3] 1 → head
2,3 → tail

Eg: [[the, Y] | Z] = [[X, mine], [is, here]].
Y = mine
Z = [[is, here]],
X = the

Q1. Find element present in list or not
or
List membership

```
member(X, [X|_]).
member(X, [_|T]) :- member(X, T).
```

let, list = [1, 2, 3, 4]
find: 3



Q2 Find number of element in a list in prolog

or

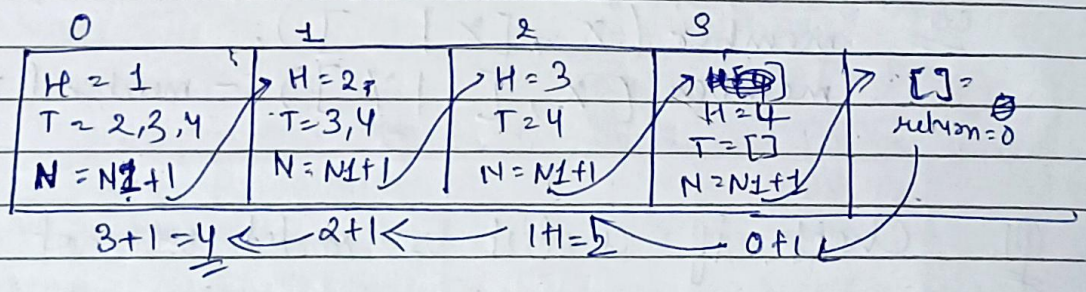
length of list

size ([], 0).

size ([_|T], N) :- size(T, N1), N is N1+1

(passing tail)

[1,2,3,4]

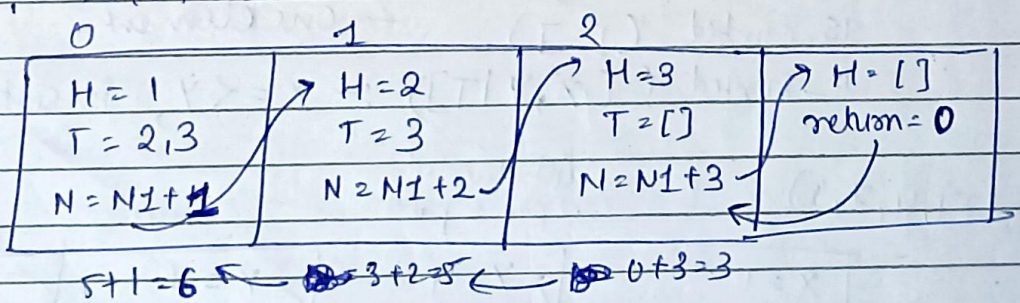


Q3 Find sum of element of a list

sumlist ([], 0).

sumlist ([H|T], N) :- sumlist (T, N1), N is N1+H

[1,2,3]



Page No.
Date

⊙ Recursive search in PROLOG

[adam, harry, john] search = harry

member(X, [X|_]) ← (check head)
member(X, [Y|_]) :- X = Y

member = (harry, [adam|_])

[adam, bobby, harry, john] X = harry

code

```
member(X, [X|_]).
member(X, [_|Y]) :- member(X,Y).
```

Q1. Check if a list is sorted or not

[1, 4, 7, 9] ⇒ [4, 7, 9] → [7, 9]

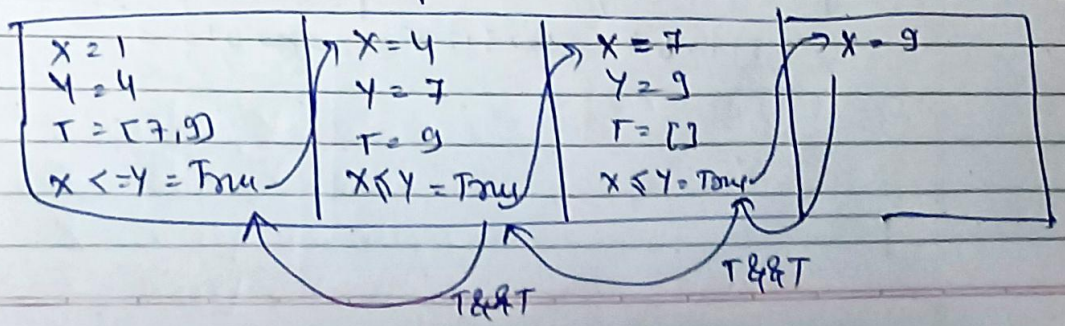
1 ≤ 4 4 ≤ 7 7 ≤ 9

all adjacent pair are sorted in ASC order

```
is_sorted([]).          ← empty list
is_sorted([_]).         ← one element
is_sorted([X,Y|T]) :- X =< Y, is_sorted([Y|T])
```

list divided into 3 parts

[1,4,7,9]



⊙ Append or Concatenate

L1 = [1,2,3] ⟶ L3 = [1,2,3,a,b]
L2 = [a,b]

concate (L1, L2)

[1 | 2,3] [a,b] → R = [H, L3]
H T L2

concate [T, L2, L3]
it will stored in L3

append ([], L2, L2)
append ([H|T], L2, [H, L3]) :- append (T, L2, L3).

[1 | 2,3]	[2 | 3]	[3]	[]
H T	H T		
L2 = [a,b]	L2 = [a,b]	L2 = [a,b]	L2 = [a,b]
R = [1, ...]	R = [2, ...]	R = [3, ...]	R = [a,b]
1,2,3,a,b	2,3,a,b	3,a,b	L3

⊙ Infinite loop in Prolog.

Case 1 :
parrot (X, Y) :- child (Y, X)
child (A, B) :- parrot (B, A)

circular loop then local stack will be full.

Case 2 :
person (adam).
person :- person (Y), mother (X, Y).

Page No.
Date

↑ store or accumulate result

⊙ Accumulator in Prolog.

General code
```
listen([],0)
listen([H|T],N) :- listen(T,N1), N is N1+1.
```

using accumulator (length of list)
```
listen(2,N) :- lenacc(2,0,N)
lenacc([],A,A).
len([H|T],A,N) :- A1 is A+1, lenACC(T,A1,N)
```

[a,b,c].

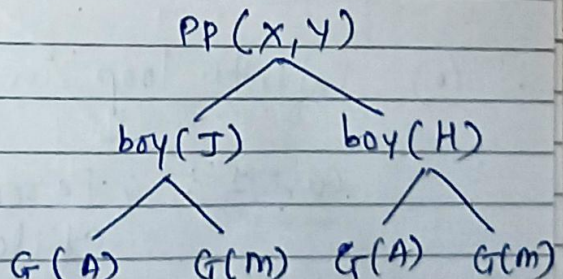
A=0

0	1	2	3
A1 = A+1 = 0+1 = 1	A = 1 A1 = A+1 = 1+1 = 2	A = 2 A1 = A+1 = 2+1 = 3	A = 3
[a,b,c] H T	[b,c] H T	[c] H	[]

⊙ Backtracking.

Rule: possible_pair(X,Y) :- boy(X), girl(Y).

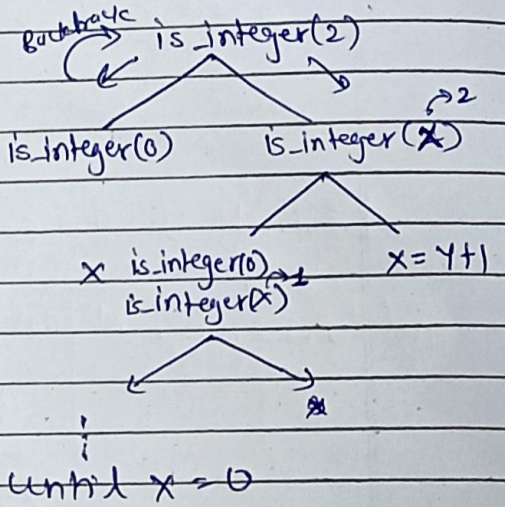
facts:
```
boy(john).
boy(harry).
girl(alisa).
girl(miri).
```



o/p: X = john, Y = alisa
X = john, Y = miri
X = harry, Y = alisa
X = harry, Y = miri

is_integer(0).
is_integer(X) :- is_integer(Y), X is Y+1

is_integer(0) → True
is_integer(2).



is_integer(-2). ← falls in infinite loop
becz stooping condition is 0.

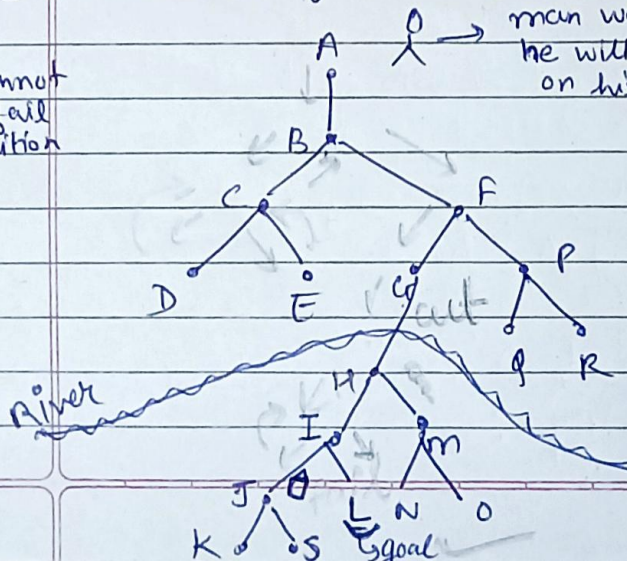
⊙ ! cut (control backtracking)

↳ Let say it is like checkpoint we cannot go back after passing that checkpoint. (i.e we cannot backtrack)

It saves the processing power and time.

⊙ cut & fail in prolog

Note:
we cannot pass fail condition



→ man was fed with info that on his way he will get river and there is no shop on this way

always succeeded
cut defines fail & barrier
↳ never succeed

assume
% 100
true

if info is not true
then
o/p: false

Eg.

foo :- a, b, !, c, d, fail, e, f.