

More examples :

- John likes anyone who likes wine.

$\text{likes}(\text{john}, x) \text{ :- likes}(x, \text{wine}).$

- John likes any female who likes wine.

$\text{likes}(\text{john}, x) \text{ :- female}(x), \text{likes}(x, \text{wine}).$

- Defining sisterof (X, Y) predicate

$\text{sisterof}(x, y) \text{ :- female}(x), \text{parents}(x, m, f),$
 $\text{parents}(y, m, f).$

V Database

male(albert).

male(edward).

female(alice).

female(victoria).

parents(edward, victoria, albert).

parents(alice, victoria, albert).

$\text{sisterof}(x, y) \text{ :- female}(x), \text{parents}(x, m, f),$

goal: Sister of (alice, edward)

1. X b.t. alice
Y b.t. edward

2. PROLOG tries to satisfy female (alice).
female (alice) succeeds.

3. Tries to satisfy parents (alice, M, F)

M b.t. victoria
F b.t. albert

4. Tries to satisfy parents (edward, victoria, albert)

Entire goal succeeds

Yes

Next goal: Find out if alice is anyone's sister.

? - sister of (alice, X).

1. X in the goal and Y in the clause are shared.

X in the clause is bound to alice.

2. PROLOG tries to satisfy: female (alice) which succeeds

3. PROLOG tries to satisfy parents (alice, M, F) which succeeds with M b.t. victoria and F b.t. albert.

4. PROLOG tries to satisfy parents(Y, victoria, albert).

It succeed with Y b.t. edward.

So, a solution is "edward".

If ";" is pressed (In std. PROLOG),

Then ?

Second solution generated is "alice" !!

To eliminate "alice" from the solution,
we can write

sisterof (X, Y) :- female(X), parents(X, M, F),
parents(Y, M, F), $X \neq Y$.

- However, $X \neq Y$ can't be put immediately after female(X) as that would result into an error "Free variable in the expression".

OR alternatively in the goal section:

sisterof(alice, X), Y = alice, $X \neq Y$.

X = edward, Y = alice

1 solution

We can't write sisterof(alice, X), $X \neq$ alice
as these are objects of different types, as
per PROLOG

Example

predicates

likes(symbol, symbol)

clauses

likes(marry, food).

likes(marry, flowers).

likes(john, food).

likes(john, food).

likes(john, wine).

I goal: likes(marry, x), likes(john, x).

X = food

X = food

2 solutions

II goal: likes(john, x), likes(marry, x).

X = food

X = food

2 solutions

III goal: likes(john, x), likes(john, x).

$X = \text{food}$

$X = \text{food}$

$X = \text{food}$

$X = \text{food}$

$X = \text{Wine}$

5 solutions.

IV

goal: likes (John, X), likes (Mary, Y).

$X = \text{food}$ $Y = \text{food}$

$X = \text{food}$ $Y = \text{flowers}$

$X = \text{food}$ $Y = \text{food}$

$X = \text{food}$ $Y = \text{flowers}$

$X = \text{Wine}$ $Y = \text{food}$

$X = \text{Wine}$ $Y = \text{flowers}$

6 solutions.

Database

female(marry).

mother(john, ann).

mother(marry, ann).

father(marry, fred).

father(john, fred).

sisterof(x, y) :- female(x), parent(x, m, f),
parent(y, m, f).

parent(x, m, f) :- mother(x, m),
father(x, f).

goal: sisterof(marry, y).

$y = \text{john}$

$y = \text{marry}$ (Immediately after $y = \text{marry}$,
the 2nd solution is found,
PROLOG returns to prompt.
Further possibilities are
not there, so not tried)

Question: How to exclude 'marry' from the soln?

goal: sisterof(marry, y), $y \neq \text{marry}$.

Error: Turbo-PROLOG says:

"Expression may not contain objects
of this type"

goal: sisterof(marry, Y), $x = \text{marry}$, $x \neq Y$

$Y = \text{john}$, $x = \text{marry}$ 1 solⁿ.

Example /* Only for illustration. */

female(marry).

parent(john, ann, fred).

parent(john, lilly, fred).

parent(marry, ann, fred).

parent(marry, ann, bill).

sisterof(X, Y) :- female(X), parent(X, M, F),
parent(Y, M, F).

goal: sisterof(marry, X).

$M = \text{ann}$

$F = \text{fred}$

$\left[\begin{array}{l} X = \text{john} \\ X = \text{marry} \end{array} \right.$

$M = \text{ann}$

$F = \text{bill}$

$\rightarrow X = \text{marry}$

3 solutions

List

- List is an ordered sequence of elements, where order of elements matter.
- Elements of a list may be constant, variable or list themselves.
- List can contain an unlimited number of elements.
- All the objects in a list must ^{be} of the same type, but may be very complex.
- In LISP, the only data structure available is List apart from constants.
- The list can be bifurcated as Head and Tail.

The notation is `[Head | Tail]`.

Head — Contains the first element of a list

Tail — It's a list which contains all the elements except the Head.

Example

List

Head

Tail

[a, b, c]

a

[b, c]

[the, [cat, sat]]

the

[[cat, sat]]

[]

None

None

Matching of Lists

List 1

List 2

Instantiation

[x, y, z]

[john, likes, fish]

X = john

Y = likes

Z = fish

[cat]

[x | y]

X = cat

Y = []

[x, y | z]

[marry, likes, wine]

X = marry

Y = likes

Z = [wine]

[[the, y] | z]

[[~~x~~, hare], [is, here]]

X = the

Y = hare

Z = [[is, here]]

[vale, horse]

[horse, x]

Don't match

Declaring list in PROLOG

domains

list = symbol *

predicates

member (symbol, list)

clauses

member (x, [x|_]).

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

goal: member (susan, [tom, bill, susan])

Call: member (susan, [tom, bill, susan])

Fail: member (susan, [tom, bill, susan])

Redo: member (susan, [tom, bill, susan])

Call: member (susan, [bill, susan])

Fail: member (susan, [bill, susan])

Redo: member (susan, [bill, susan])

Call: member (susan, [susan])

Ret: member (susan, [susan])

Ret: member (susan, [bill, susan])

Ret: member (susan, [tom, bill, susan])

True

Goal: writelist([a, b, c]).

a

b

c

Yes

domains

list = symbol *

predicates

writelist(list)

clauses

writelist([Head | Tail]) :- write(Head), nl,
writelist(Tail).

Output:

a

b

c

No

How to eliminate "No" from the answer?

Soln: By writing writelist[] as last clause.

- Even, if we write writelist([]) over the given clause, still answer doesn't change.

In Recursive Definitions, we must look for:

(a) Left Recursion

This arises when a rule causes the invocation of a goal that is essentially equivalent to the original goal that caused the rule to be used.

Thus, if we defined:

mother (lilly, adam).

person (x) :- person (y), mother (x, y).

person (adam).

Goal: person (z).

Problem ?

Backtracking is not possible, because in

order to backtrack a clause must fail.

Solⁿ : If we put person (adam) over a rule, then PROLOG has chance to look for finding solutions.

Heuristics: Put facts over rules in DB.

(b) Circular Definitions

$$\text{parent}(x, y) :- \text{child}(y, x).$$

$$\text{child}(x, y) :- \text{parent}(y, x).$$

Example 2

domains

$$\text{list} = \text{integer} *$$

predicates

$$\text{testlist}(\text{list})$$

clauses

$$/* c1 */ \quad \text{testlist}([_ | \text{Tail}]) :- \text{testlist}(\text{Tail}).$$

$$/* c2 */ \quad \text{testlist}([]).$$

$$/* \text{goal: testlist}([_ | \text{Tail}]) :- \text{testlist}(\text{Tail}). */$$
~~Yes~~

$$\text{goal: testlist}([1, 2]).$$

Yes

$$\text{goal: testlist}(x).$$

Stack overflow

If we interchange $c1$ and $c2$, then O/P will be

$$X = []$$

Appending a list

append($[]$, L , L).

append($[x | L1]$, $L2$, $[x | L3]$):-

append($L1$, $L2$, $L3$).

Goal: append($[a, b, c]$, $[d, e, f]$, x)

$x = [a, b, c, d, e, f]$

1 solution.

Goal: append(x , $[a, b, c]$, $[d, e, a, b, c]$)

$x = [d, e]$

1 solution.

Execution Trace:

Call Goal: append($["a", "b"]$, $["c", "d"]$, ~~x~~)

Redo : append($["a", "b"]$, $["c", "d"]$, $-$)

Call: append($["b"]$, $["c", "d"]$, $-$)

Redo: append($["b"]$, $["c", "d"]$, $-$)

Call: append($[]$, $["c", "d"]$, $-$)

Ret: append($[]$, $["c", "d"]$, $["c", "d"]$)

Ret: append($["b"]$, $["c", "d"]$, $["b", "c", "d"]$)

Ret: append($["a", "b"]$, $["c", "d"]$, $["a", "b", "c", "d"]$)

$["a", "b", "c", "d"]$