ICT 4102

Artificial intelligence Lab

Prolog -02

Prolog – Comparison (1)

```
%sectionA
goal(brazil,4).
goal(germany,3).
goal(france,1).

%sectionB
goal(argentina,2).
goal(portugal,5).
goal(japan,1).
```

Prolog – Comparison (2)

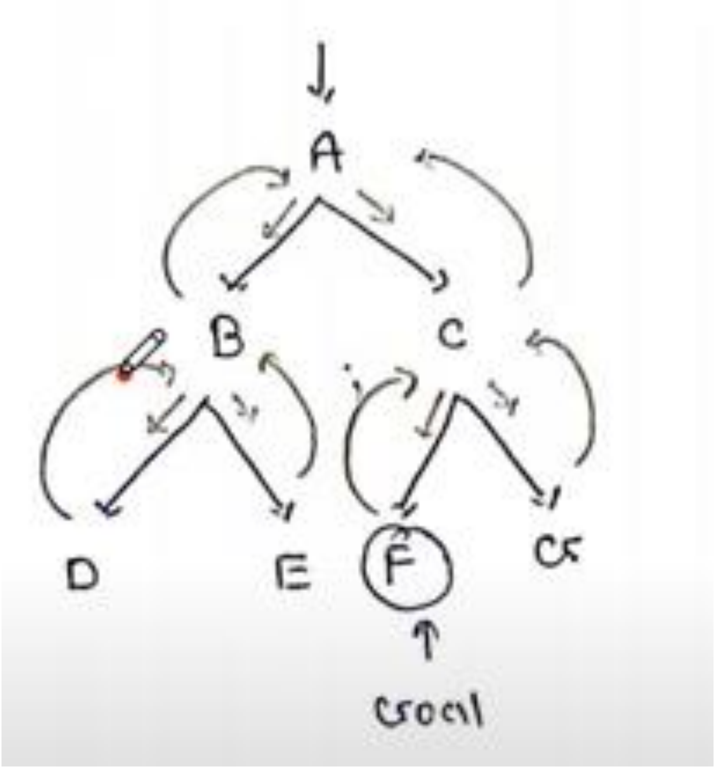
```
go:-
write('enter section A country name'),nl,
read(X),nl,
goal(X,Y),nl,
write('Section A country score is '),nl,
write(Y),nl,

write('enter section B country name'),nl,
read(P),nl,
goal(P,Q),nl,
write('Section B country score is '),nl,
write(Q),nl,
```

```
compare(Y,Q).
compare(Y,Q):-
Y>Q,nl,
write('Section A country is the winner');
Y<Q,nl,
write('Section B country is the winner');
Y=:=Q,nl,
write('Draw in both section').
```

Backtracking

- In prolog, untill it reach it's proper destination it try to backtrack whether the destination is found.

- “;” = try to find another goal.
- “.”=stop,reached destination node.



Example

- Possible pairs for girls & boys (X,Y)

Y is girl and X is boy.

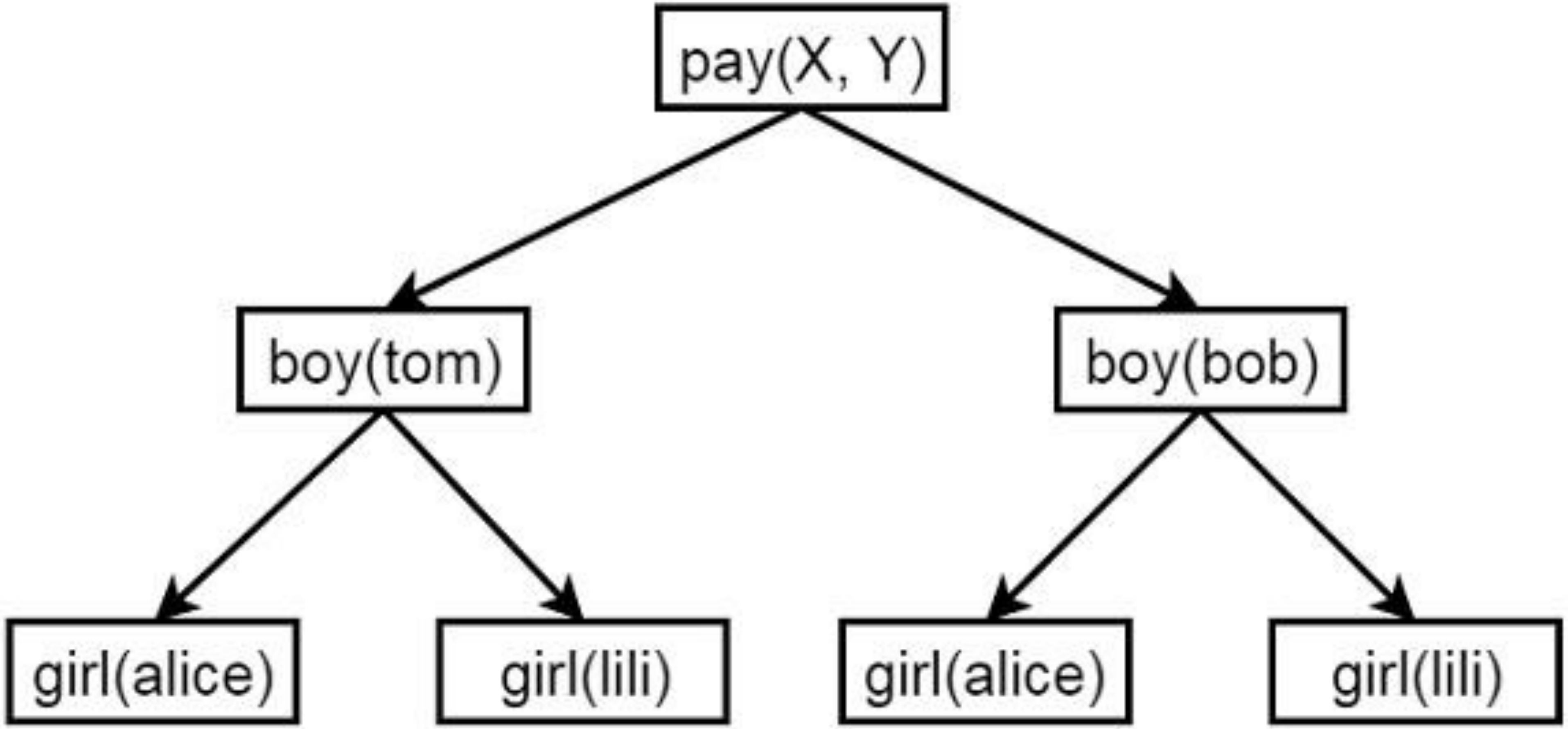
Rule:-

```
        possible_pair(X,Y) :- boys(X),girls(Y).
```

Facts:-

```
boy(tom).
boy(bob).
girl(alice).
girl(lili).
```

Backtracking



possible_pairs(X,Y)

=> X = tom, Y = alice.

=> X is tom, Y = lili.

Left traverse

=>X is bob, Y = alice.

=>X is bob, Y = lili.

Right traverse

Prolog Program

```
boy(tom).
boy(bob).
girl(alice).
girl(lili).
pay(X,Y):-boy(X),girl(Y).
```

Recursion

- Theory
- – Introduce recursive definitions in Prolog
- – Go through four examples
- – Show that there can be mismatches between
- the declarative and procedural meaning of a
- Prolog program

Recursive Definitions

- Prolog predicates can be defined recursively
- A predicate is recursively defined if one or more rules in its definition refers to itself.

Example 1: Eating

```
isDigesting(X,Y):- justAte(X,Y).
isDigesting(X,Y):- justAte(X,Z), isDigesting(Z,Y).

justAte(mosquito,blood(john)).
justAte(frog,mosquito).
justAte(stork,frog).
```

```
?- isDigesting(stork,mosquito).
Yes
```

Example 2: Factorial

```
% Base case: Factorial of 0 is 1
factorial(0, 1).

% Recursive case: Calculate factorial of N as N multiplied by factorial of N-1
factorial(N, Result) :-
   N > 0,
   N1 is N - 1,
   factorial(N1, SubResult),
   Result is N * SubResult.
```

```
?- factorial(0, Result). % Output: Result = 1
?- factorial(5, Result). % Output: Result = 120
?- factorial(10, Result). % Output: Result =
3628800
```

Example 3: Descendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).
descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), child(Z,Y).
```

```
?- descend(anna,donna).
No
```

Example 3: Descendant

```
child(anna,bridget).
child(bridget,caroline).
child(caroline,donna).
child(donna,emily).
descend(X,Y):- child(X,Y).
descend(X,Y):- child(X,Z), descend(Z,Y).
```

```
?- descend(anna,donna).
yes
```

```
?- descend(A,B).
```

LIST

Theory

- Introduce lists, an important recursive data structure often used in Prolog programming
- Define the member/2 predicate, a fundamental Prolog tool for manipulating lists
- Illustrate the idea of recursing down lists

LIST

- A list is a finite sequence of elements

Examples of lists in Prolog:

1. [mia, vincent, jules, yolanda]
2. [mia, robber(honeybunny), X, 2, mia] [ ]
3. [mia, [vincent, jules], [butch, friend(butch)]]
4. [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Important things about lists

- List elements are enclosed in square brackets
- The length of a list is the number of elements it has
- All sorts of Prolog terms can be elements of a list
- There is a special list: the empty list [ ]

Head and Tail

- A non-empty list can be thought of as consisting of two parts
  - The head
  - The tail
- The head is the first item in the list
- The tail is everything else
  - The tail is the list that remains when we take the first element away
  - The tail of a list is always a list

Head and Tail example 1

- [mia, vincent, jules, yolanda]

  Head:

  Tail:

Head and Tail example 1

- [mia, vincent, jules, yolanda]

  Head: mia
  Tail: [vincent,jules,yolanda]

Head and Tail example 2

- [[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

- Head:
- Tail:

Head and Tail example 2

- ?- [**Head** |**Tail**]=[[ ], dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

  Head: [ ]
  Tail: [dead(z), [2, [b,c]], [ ], Z, [2, [b,c]]]

Head and Tail example 3

- [dead(z)]

  Head:
  Tail:

Head and Tail example 3

- [dead(z)]

Head: dead(z)
Tail:

Head and Tail example 3

- [dead(z)]

  Head: dead(z)
  Tail: [ ]

Head and tail of empty list

- The empty list has neither a head nor a tail

- For Prolog, [ ] is a special simple list without any internal structure

- The empty list plays an important role in recursive predicates for list processing in Prolog

The built-in operator |

- Prolog has a special built-in operator | which can be used to decompose a list into its head and tail
- The | operator is a key tool for writing Prolog list manipulation predicates

The built-in operator |

```
?- [Head|Tail] = [mia, vincent, jules, yolanda].

Head = mia
Tail = [vincent,jules,yolanda]
yes


?-
```

The built-in operator |

```
?- [X|Y] = [mia, vincent, jules, yolanda].

X = mia
Y = [vincent,jules,yolanda]
yes


?-
```

The built-in operator |

```
?- [X|Y] = [ ].

no


?-
```

The built-in operator |

```
?- [X,Y|Tail] = [[ ], dead(z), mia] .

X = [ ]
Y = dead(z)
Tail = mia
yes

?-
```

Anonymous variable

- Suppose we are interested in the second and fourth element of a list

```
?- [X1,X2,X3,X4|Tail] = [mia, vincent, marsellus, jody, yolanda].
X1 = mia
X2 = vincent
X3 = marsellus
X4 = jody
Tail = [yolanda]
yes

?-
```

Some Operations on Lists

- Lists can be used to represent sets although there is a difference; the order of elements in a set does not matter while the order of items in a list does; also, the same object can occur repeatedly in a list. Still, the most common operations on lists are similar to those on sets. Among them are:
- checking whether some object is an element of a list, which corresponds to checking for the set membership;
- Concatenation of two lists, obtaining a third list, which corresponds to the union of sets;
- Adding a new object to a list, or deleting some object from it.

Membership

- Let us implement the membership relation as **member(X, L)**

where X is an object and L is a list. The goal member(X, L) is true if X occurs in L. For example,

member(b, [a,b,c])

- is true,

member(b, [a,[b,c]])

- is not true, but

member([b,c], [a,[b,c]])

- Is true.

Membership

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

?- member(b,[a,[b,c]]).
false.

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

Concatenation

- For concatenating lists we will define the relation

conc(L1, L2,L3)

```
Rules:
concat_lists(L1, L2, Concat) :- append(L1, L2, Concat).
```

After writing the rules in the prolog file , we have to consult it from the SWI-Prolog window.

- Here L1 and L2 are two lists, and Concat is their concatenation. For example

```
?- concat_lists([a,b,c],[mia, yolanda], Concat).
Concat = [a, b, c, mia, yolanda].
```

Deleting an item from Last

Rules:

```
delete_last([_], []).
delete_last([Head|Tail], [Head|NewTail]) :- delete_last(Tail, NewTail).
```

```
?- delete_last([1,2,3,4,5], L).
L = [1, 2, 3, 4].
```

Deleting from an item

Rules:

```
delete_item(X,[X|Tail],Tail).
delete_item(X,[Y|Tail],[Y|Tail1]):- delete_item(X,Tail,Tail1).
```

Queries:

```
?- delete_item(a,[d,b,a,c],New_list).
New_list = [d, b, c].
```

Adding an item

In Prolog, lists are immutable, which means you cannot directly modify a list once it is defined. However, you can create a new list by *appending* an item to an existing list. Here's an example of how you can add an item to a list in Prolog:

Rules:

```
add_item(Item, List, NewList) :- append(List, [Item], NewList).


?- add_item(dena,[hiyana,dona,mia],L).
L = [hiyana, dona, mia, dena].
```

Exercise Task

1. Implement a Prolog predicate ‘equal_length/2’ that takes two lists as input and succeeds if both lists have the same length .Give **some example queries and their expected outputs**.
2. Write a Prolog predicate ‘maximum/3’ that takes three integers as input and returns the maximum of the three.
3. Write a Prolog predicate to find the length of a list.