Prolog Programming

Chapter 4

Using Structures: Example Programs

1/16/2010

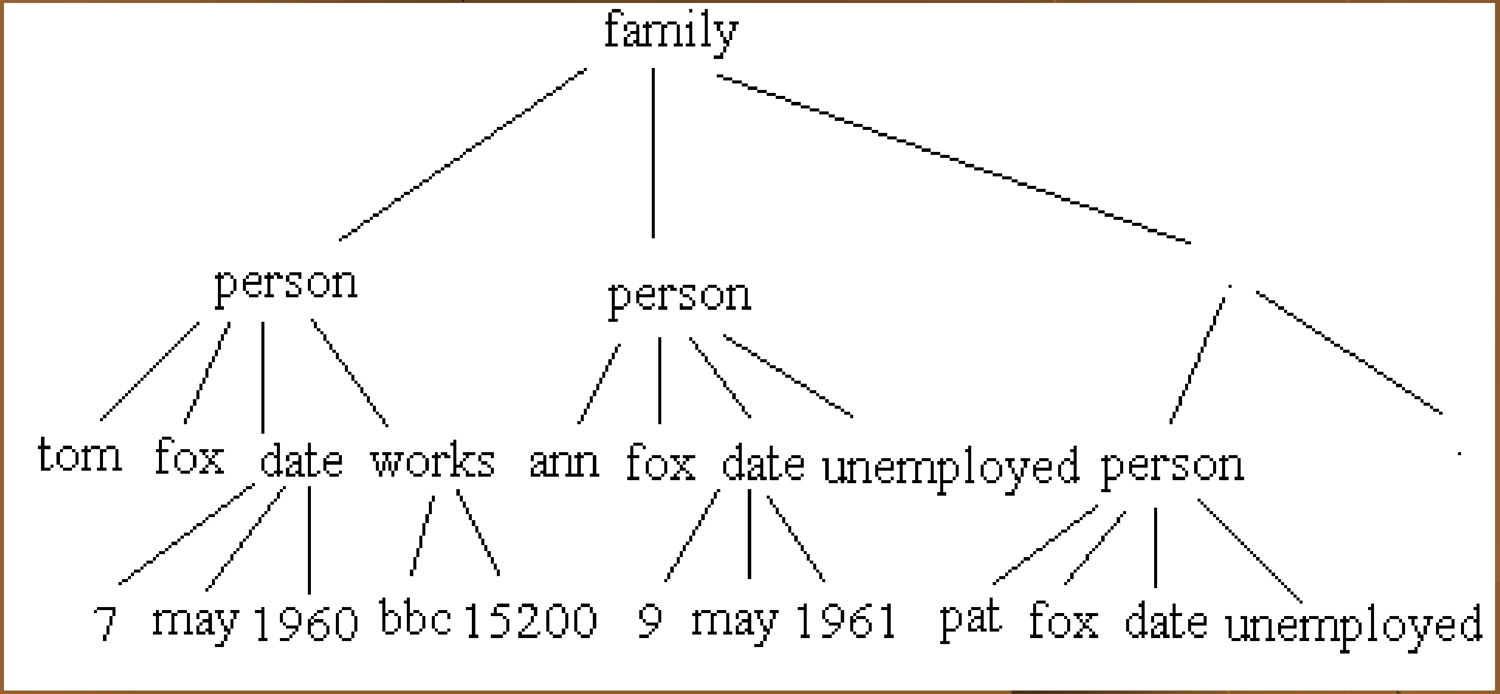
Chapter 4
Using Structures: Example Programs

- Retrieving structured information from a database
- Doing data abstraction
- Simulating a non-deterministic automation
- Travel agent
- The eight queens problem

4.1 Retrieving structured information from a database

- A database can be naturally represented in Prolog as a set of facts
- Prolog is a very suitable language for retrieving the desired information from such a database
- In Prolog, we can refer to objects without actually specifying all the components of these objects

Family database structure



Structure information for family database

```
family(
    person(tom,fox,date(7,may,1960),work(bbc,15200)),
    person(ann,fox,date(9,may,1961),unemployed),
    [person(pat,fox,date(5,may,1983),unemployed),
     person(jim,fox,date(5,may,1983),unemployed)]).
```

Utility procedures for family database

```
husband(X) :-
        family ( X, _, _ ).              % X is a husband
wife(X) :-
        family ( _, X, _ ).              % X is a wife
child(X) :-                              % X is a child
        family ( _, _, Children ),
        member ( X, Children ).          % X in list Children
```

More procedures for family database

```
exits (Person) :-
        husband(Person); wife (Person); child(Person).

dateofbirth (person ( _, _, Date, _ ), Date ) .

salary ( person ( _, _, _, works ( _, S)), S ).

salary ( person ( _, _, _, unemployed), 0 ).
```

Using utilities to query the database

- To find the names of all the people in the database:

```
?- exits ( person ( Name, Surname, _, _ ) ) .
```

- To find all children born in 2000:

```
?- child(X),
   dateofbirth(X,date(_, _, 2000)).
```

- To find the names of unemployed people who were born before 1973:

```
?- exits(person(Name,Surname,date(_,_,Year),unemployed)),
   Year < 1973.
```

Using more utilities to query the database

- To find people born before 1960 whose salary is less than 8000:

```
?- exits(Person),
   dateofbirth(Person, date( _,  _, Year )) ,
   Year < 1960 ,
   salary ( Person, Salary ),
   Salary < 8000 .
```

A program to calculate total income

```
total ( [ ] , 0 ).
total ( [ Person | List ] , Sum ) : -
        salary ( Person, S ),
        total ( List, Rest ),
        Sum is S + Rest .
```

Questions :

- To find income of families can then be found by the question:

?- family (Husband, Wife, Children) ,
total ([Husband, Wife | Children] , Income) .

Questions continued :

- All families that have income per family member of less than 2000 by:

```
?- family ( Husband, Wife, Children),
   total ( [ Husband, Wife | Children ] , Income ),
   length ( [ Husband, Wife | Children ] , N ),   % N size of family
   Income / N < 2000.
```

Chapter 4
Using Structures: Example Programs

- Retrieving structured information from a database
- Doing data abstraction
- Simulating a non-deterministic automation
- Travel agent
- The eight queens problem

4.2 Doing Data Abstraction

- A process of organizing various pieces of information into natural units (possibly hierarchically)
- Structuring the information into some conceptually meaningful form
- Making the use of complex data structures easier, and contributes to the clarity of programs

Example

- In family structure, each family is a collection of pieces of information

- These pieces are all clustered into natural units such as a person or a family, so they can be treated as single objects

- A family is represented as structured object

FoxFamily = family (person (tom, fox, _, _),_,_)

Selectors for relations

- Selector_relation (Object, Component_selected)

```
husband ( family ( Husband, _, _ ), Husband).
wife ( family ( _, Wife, _) , Wife).
children ( family ( _, _, ChildList ), ChildList).
```

- Selectors for particular children:-

```
firstchild(Family, First) :-
        children ( Family, [ First | _ ]).
secondchild(Family, Second) :-
        children ( Family, [ _ , Second | _ ]).
```

Selectors for relations continued

- To select the Nth child :

```
nthchild ( N, Family, Child ) :-
     children ( Family, ChildList ),
     nth_member( N, ChildList, Child ).  % Nth element of a list
```

- Other selectors

```
firstname ( person ( Name, _, _, _ ), Name).
surname ( person ( _, Surname, _, _ ), Surname).
born ( person ( _, _, Date, _ ), Date ).
```

Example

- Tom Fox and Jim Fox belong to the same family and that Jim is the second child of Tom

  firstname (Person1, tom), surname (Person1, fox),
  firstname (Person2, jim), surname (Person2, fox).
  husband (Family, Person1),
  secondchild (Family, Person2).

Example continued

- Person1, Person2 and Family are instantiated as

```
Person1 = person(tom, fox, _, _)

Person2 = person(jim, fox, _, _)

Family = family(person(tom,fox,_,_),_,[_, person(jim,fox)| _ ])
```

Chapter 4
Using Structures: Example Programs

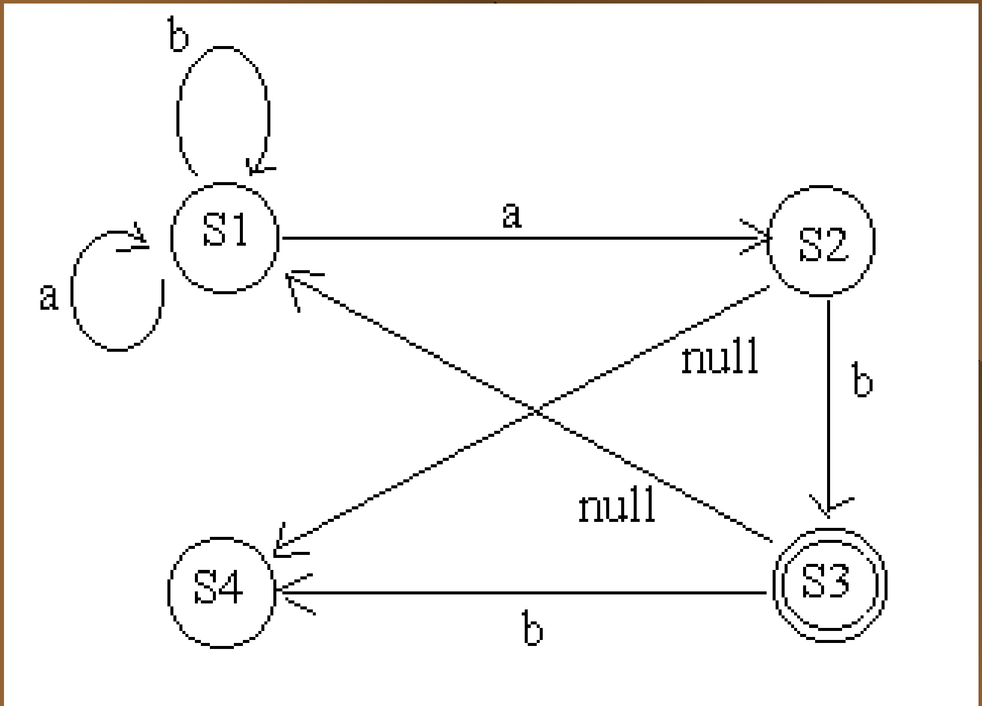
- Retrieving structured information from a database
- Doing data abstraction
- Simulating a non-deterministic automation
- Travel agent
- The eight queens problem

4.3 Simulating a non-deterministic automation

A non-deterministic finite automation

- an abstract machine that reads as input a string of symbols and decides whether to accept or to reject
- has a number of states and it is always in one of the states
- can change from current state to another state

Example of a non-deterministic finite machine



In the previous example

- S1, S2, S3, and S4 are the states of the automation
- starts at initial state S1 and ends at final state S3
- moves from state to state while reading the input string
- null denoting the null symbol that means ‘silent moves’ without reading of any input

Accept the input string if a transition path

- starts with the initial state
- ends with a final state
- the arc labels along the path correspond to the complete input string

- accepts strings such as *ab* and *aabaab*
- rejects strings such as *abb* and *abba*

In Prolog,

- A unary relation *final* to define the final state
- A three-argument relation *trans* to define the state transition such as *trans(S1,X,S2)*
- a binary relation *silent(S1,S2)*

For the example automation

```
final(s3).
trans(s1,a,s1).
trans(s1,a,s2).
trans(s1,b,s1).
trans(s2,b,s3).
trans(s3,b,s4).
silent(s2,s4).
silent(s3,s1).
```

Define the acceptance of a string

- Accepts a given string if (starting from an initial state) after having read the whole input string, the automation can (possibly) be in its final state
- Define a binary relation *accepts(State, String)*
- Initial state *State* and input string *String*

There are three cases:

- empty string [ ] is accepted if *State* is a final state
- a non-empty is accepted from *State* if reading the first symbol in the string can bring the automation into some state *State1* and the rest of the string is accepted from *State1*
- a string is accepted from *State* if the automation can make a silent move from *State* to *State1* and then accepted the whole input string from *State1*

Rules in Prolog Programming

```
accepts(State,[ ]) :-
            final(State).                    % case 1

accepts(State,[X|Rest]) :-
            trans(State,X,State1),
            accepts(State1,Rest).            % case 2

accepts(State,String) :-
            silent(State,State1),
            accepts(State1,String).          % case 3
```

Questions:

```
?- accepts ( s1, [a,a,a,b]).
yes

?- accepts ( S, [a,b]).
S = s1;
S = s3
```

Questions continued:

```
?- accepts ( s1, [X1, X2, X3]).
X1 = a
X2 = a
X3 = b;

X1 = b
X2 = a
X3 = b;

no
```

More Questions:

```
?- String = [ _, _, _ ], accepts( s1, String).
String = [a,a,b];
String = [b,a,b];
no
```

Chapter 4
Using Structures: Example Programs

- Retrieving structured information from a database
- Doing data abstraction
- Simulating a non-deterministic automation
- Travel agent
- The eight queens problem

4.4 Travel agent

- What days of the week is there a direct evening flight from Ljubljana to London ?
- How can I get from Ljubljana to Edinburgh on Thursday?
- I have to visit Milan, Ljubljana and Zurich, starting from London on Tuesday and returning on Friday. In what sequence should I visit these cities so that I have no more than one flight each day of the tour?

Flight Data Base

timetable(Place1, Place2, ListOfFlights)

ListOfFlight is a list of structured items of the form

DepartureTime / ArrivalTime / FlightNumber / ListOfDays

Example,

```
timetable(london,zurich,
        [  9:10/11:45/ba614/alldays,
          14:45/17:20/sr805/ [mo,tu,we,th,fr,su] ).
```

route(Place1,Place2,Day,Route)

start point Place1

end point Place2

all the flight are on the same day of the week, Day

all the flights in Route are in the timetable relation

there is enough time for transfer between flights

- the route is represented as a list of structured objects of the form

  From / To / FlightNumber / Departure_time

Auxiliary predicates

flight(Place1,Place2,Day,FlightNum,DepTime,ArrTime)

- flight is a flight route planner
- there is a flight between Place1 and Place2 on the day of the week Day with the specified departure time DepTime and arrival time ArrTime

Auxiliary predicates continued

deptime(Route,Time)

Departure time of Route is Time

transfer(Time1, Time2)

There is at least 40 minutes between Time1 and Time2 to transfer between flights

Similarities to non-deterministic automation

- The states of the automation correspond to the cities
- A transition between two states corresponds to a flight between two cities
- The transition relation of the automation corresponds to the timetable relation
- The automation simulator finds a path in the transition graph between the initial state and a final state; a travel planner finds a route between the start city and the end city of the tour

Travel Agent Program

```
:- op(50,xfy,:).
route(P1,P2,Day,[P1/P2/Fnum/Deptime]) :-            % direct flight
            flight(P1,P2,Day,Fnum,Deptime,_).
route(P1,P2,Day,[(P1/P3/Fnum1/Dep1)|RestRoute]):-      % indirect flight
            flight(P1,P3,Day,Fnum1,Dep1,Arr1),
            route(P3,P2,Day,RestRoute),
            deptime(RestRoute,Dep2),
            transfer(Arr1,Dep2).

flight(Place1,Place2,Day,Fnum,Deptime,Arrtime) :-
        timetable(Place1,Place2,Flightlist),
        member(Deptime/Arrtime/Fnum/Daylist,Flightlist),
        flyday(Day,Daylist).
```

Travel Agent Program continued :

```
flyday(Day,Daylist) :- member(Day,Daylist).

flyday(Day,alldays) :- member(Day,[mo,tu,we,th,fr,sa,su]).

deptime([P1/P2/Fnum/Dep|_],Dep).

transfer(Hours1:Mins1,Hours2:Mins2) :-
        (60 * (Hours2 - Hours1) + Mins2- Mins1) >= 40.

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

conc([],L,L).
conc([X|L1],L2,[X|L3]) :- conc(L1,L2,L3).
```

Travel Agent Program continued :

% A FLIGHT DATABASE

```
timetable(edinburgh,london,
          [ 9:40/10:50/ba4733/alldays,
           13:40/14:50/ba4773/alldays,
           19:40/20:50/ba4833/[mo,tu,we,th,fr,su]]).

timetable(london,edinburgh,
          [ 9:40/10:50/ba4732/alldays,
           11:40/12:50/ba4752/alldays,
           18:40/19:50/ba4822/[mo,tu,we,th,fr]]).

timetable(london,ljubljana,
          [13:20/16:20/jp212/[mo,tu,we,fr,su],
           16:30/19:30/ba471/[mo,we,th,sa]]).
```

Travel Agent Program continued :

```
timetable(london,zurich,
          [ 9:10/11:45/ba614/alldays,
           14:45/17:20/sr805/alldays]).

timetable(london,milan,
          [ 8:30/11:20/ba510/alldays,
           11:00/13:50/az459/alldays]).

timetable(ljubljana,zurich,
          [11:30/12:40/jp322/[tu,th]]).

timetable(ljubljana,london,
          [11:10/12:20/jp211/[mo,tu,we,fr,su],
           20:30/21:30/ba472/[mo,we,th,sa]]).
```

Travel Agent Program continued :

```
timetable(milan,london,
        [ 9:10/10:00/az458/alldays,
         12:20/13:10/ba511/alldays]).

timetable(milan,zurich,
        [ 9:25/10:15/sr621/alldays,
         12:45/13:35/sr623/alldays]).

timetable(zurich,ljubljana,
        [13:30/14:40/jp323/[tu,th]]).

timetable(zurich,london,
        [ 9:00/ 9:40/ba613/[mo,tu,we,th,fr,sa],
         16:10/16:55/sr806/[mo,tu,we,th,fr,su]]).

timetable(zurich,milan,
        [ 7:55/ 8:45/sr620/alldays]).
```

Questions

```
?- flight(ljubljana,london,Day,_,DeptHour:_,_), DeptHour >= 18.
Day = mo;
Day = we;
…

?- route(ljubljana,edinburgh,th,R).
R = [ ljubljana / zurich / jp322 / 11:30, zurich / london / sr806 /
    16:10, london / edinburgh / ba4822 / 18:40 ]
```

Questions continued

?- permutation ([milan, ljubljana, zurich], City1, City2, City3]),
flight (london, City1, tu, FN1, _, _),
flight (City1, City2, we, FN2, _, _),
flight (City2, City3, th, FN3, _, _),
flight (City3, london, fr, FN4, _, _).

City1 = milan
City2 = zurich
City3 = ljubljana
FN1 = ba510
FN2 = sr621
FN3 = jp323
FN4 = jp211

Questions continued

```
?- conc(R,_,[ _, _, _, _]), route(zurich,edinburgh,mo,R).
```

Limit the list R to length 4 and force the search to consider shortest routes first

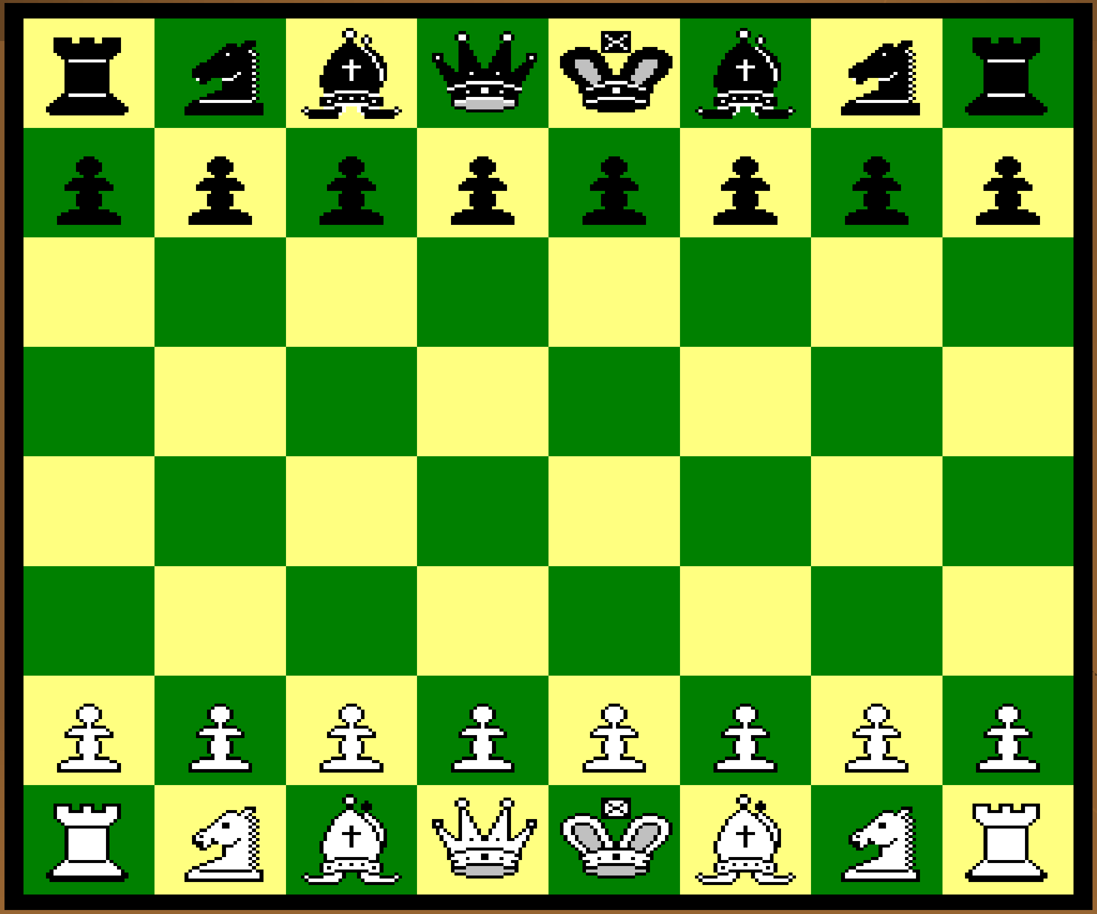
Chapter 4
Using Structures: Example Programs

- Retrieving structured information from a database
- Doing data abstraction
- Simulating a non-deterministic automation
- Travel agent
- The eight queens problem

4.5 The eight queens problem

- To place eight queens on the empty chessboard in such a way that no queen attacks any other queen
- This problem can be approached in different ways by varying the representation of the problem
- the solution will be programmed as a unary predicate solution(Pos) which is true if and only if Pos represents a position with eight queens that do not attack each other

Chess Board 8 X 8



Program 1

- to find solution Pos is a list of the form

  [X1/Y1, X2/Y2, X3/Y3, X4/Y4, X5/Y5, X6/Y6, X7/Y7, X8/Y8]
- all the queens will have to be in different column to prevent vertical attacks
- fix the X-coordinates so that the solution list will fit the following more specific template

  [1/Y1, 2/Y2, 3/Y3, 4/Y4, 5/Y5, 6/Y6, 7/Y7, 8/Y8]
- find Y values such that X/Y does not attack others in the list

The solution

Two cases:

- The list of queens is empty : the empty list is certainly a solution because there is no attack
- The list of queens is no-empty: then it looks like this :
  [X/Y | Others]
  - The first queen is at X/Y and the other queens are at squares specified by the list Others

The following conditions must hold:

- No attack between the queens in the list Others, that is, Others itself must also be a solution
- X and Y must be integers between 1 and 8
- A queen at square X/Y must not attack any of the queens in the list Others

In Prolog

```
solution ( [ X/Y | Others ] ) :-
                solution ( Others ) ,
                member ( Y, [1,2,3,4,5,6,7,8]),
                noattack ( X/Y , Others ).
```

noattack relation is defined as noattack (Q, Qlist)

two cases:

- If Qlist is empty, it is true because there is no queens to be attacked
- If Qlist is not empty and it has the form [Q1 | Qlist1] and
  - the queen at Q must not attack the queen at Q1
  - the queen at Q must not attack any of the queens in Qlist1

template guarantees that all queens are in different columns

- Only to specify explicitly that :
  - the Y coordinates of the queens are different and
  - they are not in the same diagonal, either upward or downward, that is, the distance between the squares in the X-direction must not be equal to that in the Y-direction

Program 1 in Prolog for the eight queens problem

```
solution ( [ ] ) .
solution ( [ X / Y | Others ] ) :-
                solution ( Others ) ,
                member ( Y, [1,2,3,4,5,6,7,8] ) ,
                noattack ( X/Y , Others ) .

noattack ( _ , [ ] ) .
noattack ( X / Y , [ X1 / Y1| Others ] ) :-
                Y =\= Y1,                  % not in the same row
                Y1 - Y =\= X1 - X ,        % not in the same diagonal
                Y1 - Y =\= X  - X1,
                noattack ( X / Y, Others ) .
```

Program 1 in Prolog continued:

```
member ( Item , [ Item | _ ] ) .
member ( Item , [ _ | Rest ] ) : -
                                member ( Item , Rest ) .

template([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
```

Question :

```
?- template ( S ) , solution ( S ) .
S = [ 1/4, 2/2, 3/7, 4/3, 5/6, 6/8, 7/5, 8/1 ]
```

Program 2

- No information is lost if X coordinates were omitted since the queens were simply placed in consecutive columns

- More economical representation of the board position can be used, retaining only the Y-coordinates of the queens:

  [Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8]

Strategy

- To prevent the horizontal attack, no two queens can be in the same row
- Impose a constraint on the Y-coordinates such that all queens have to occupy all the rows 1,2,3,4,5,6,7,8
- Each solution is the order of these eight numbers, that is, a permutation of the list [1,2,3,4,5,6,7,8]

Strategy continued

```
solution (S) :-
        permutation ( [ 1,2,3,4,5,6,7,8 ] , S),
        safe (S) .
```

Two cases for safe,

- S is empty
- S is non-empty list of the form [Queen | Others]. This is safe if the list Others is safe and Queen does not attack any queen in the list Others

```
safe ( [ ] ).
safe ( [ Queen | Others ] ) :-
        safe ( Others ) ,
        noattack ( Queen, Others) .
```

Strategy continued

- Since we do not use X-coordinates, in the goal noattack(Queen,Others) , we need to ensure that Queen does not attack Others when the X-distance between Queen and Others is equal to 1.

- We add X-distance as the third argument of the noattack relation: noattack(Queens,Others,Xdist)

- The noattack goal in safe relation has to be modified to noattack(Queen,Others,1)

Program 2 in Prolog for the eight queens problem

```
solution (Queens) :-
        permutation ( [ 1,2,3,4,5,6,7,8 ] , Queens),
        safe ( Queens) .

permutation ( [ ], [ ] ).
permutation ( [ Head | Tail ] , PermList ) :-
        permutation ( Tail, PermTail ) ,
        del ( Head, PermList, PermTail ) .

del ( Item, [ Item | List ], List ) .
del ( Item, [ First | List ], [ First | List1 ] ) :-
              del ( Item, List, List1 ).
```

Program 2 in Prolog continued :

```
safe ( [ ] ).
safe ( [ Queen | Others ] ) :-
        safe ( Others ) ,
        noattack ( Queen, Others, 1 ) .

noattack ( _ , [ ], _ ) .
noattack (Y, [ Y1 | Ylist ], Xdist ) :-
        (Y1-Y) =\= Xdist,      % not in the same diagonal
        (Y-Y1) =\= Xdist,       %
        Dist1 is Xdist + 1,
        noattack ( Y,Ylist,Dist1 ) .
```

Program 3

- Each queen has to be placed on some square, that is, into some column, some row, some upward diagonal, and some downward diagonal
- Each queen must be placed in a different column, a different row, a different upward and a different downward diagonal

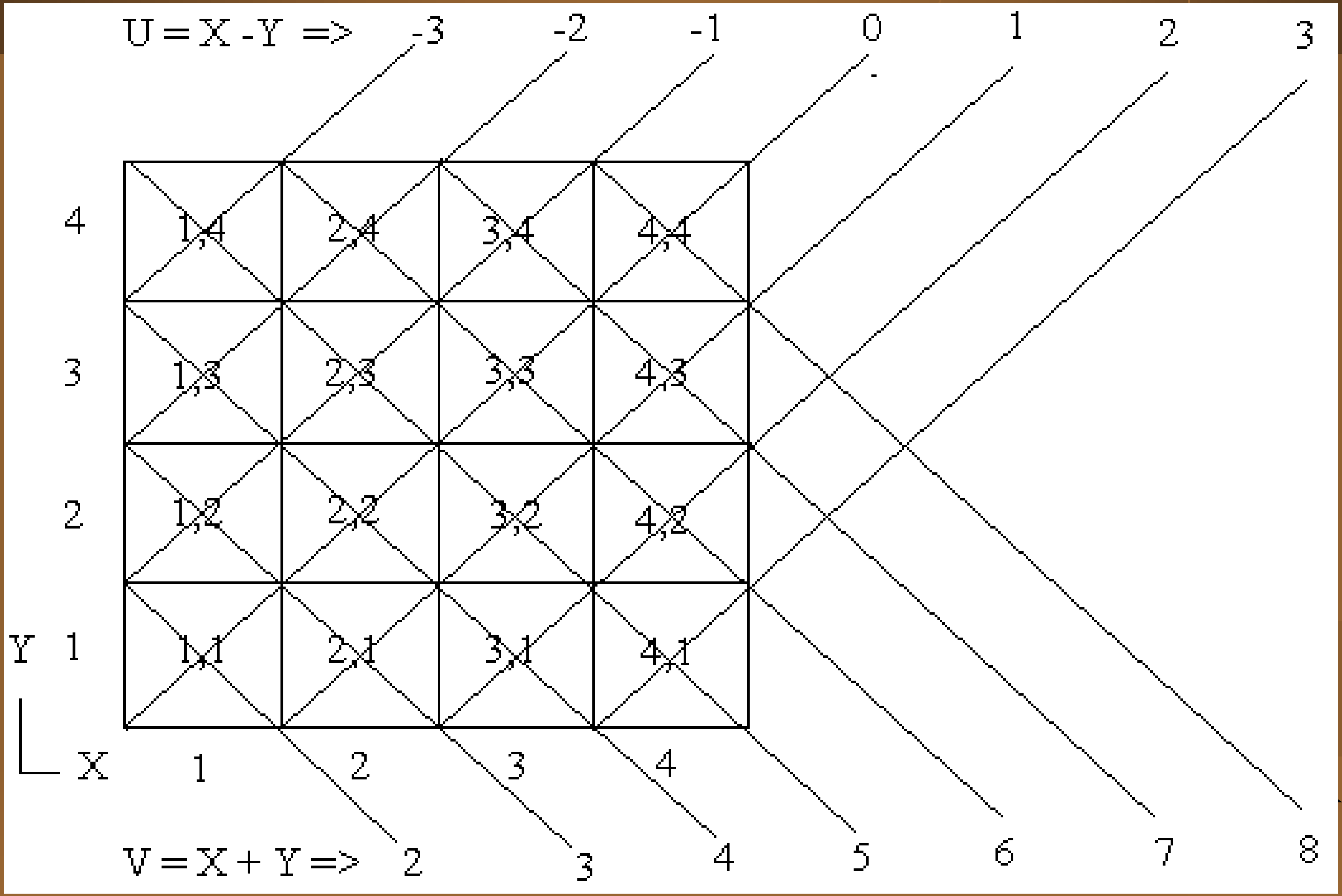
Representation

- X columns
- Y rows
- u upward diagonals
- v downward diagonals

Where u and v are determined :

- $u = x - y$
- $v = x + y$

Diagonals Relationship between X and Y



The domains for all four dimensions in 4X4 chess board

- Dx = [1,2,3,4]
- Dy = [1,2,3,4]
- Du = [-3,-2,-1,0,1,2,3]
- Dv = [2,3,4,5,6,7,8]

So the domains for all four dimensions in 8x8 chess board

- Dx = [1,2,3,4,5,6,7,8]
- Dy = [1,2,3,4,5,6,7,8]
- Du = [-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7]
- Dv = [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]

Strategy

- select eight 4 - tuples (X,Y,U,V) from domains
- never use the same element twice from any of the domains
- once X and Y are chosen, U and V are determined

The solution is that, given all four domains,

- select the position of the first queen
- delete the corresponding items from the four domains
- use the rest of the domains for placing the rest of the queens

Program 3 in Prolog for the eight queens problem

```
solution(Ylist) :-
        sol(Ylist,                                          % Y-coordinate
            [1,2,3,4,5,6,7,8],                              % Domain for X
            [1,2,3,4,5,6,7,8],                              % Domain for Y
            [-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7],         % Up Diagonals
            [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]).        % Down Diagonals
```

Program 3 in Prolog continued

```
sol([ ],[ ],Dy,Du,Dv).
sol([Y|Ylist],[X|Dx1],Dy,Du,Dv) :-
        del(Y,Dy,Dy1),                  % Choose a Y-coordinate
        U is X-Y,                       % Corresponding upward dia
        del(U,Du,Du1),                  % Remove it
        V is X+Y,                       % Corresponding downward
        del(V,Dv,Dv1),                  % Remove it
        sol(Ylist,Dx1,Dy1,Du1,Dv1).     % Use remaining values

del(Item,[Item|List],List).
del(Item,[First|List],[First|List1]) :-
            del(Item,List,List1).
```