1

Control Structures

• Programming languages offer a variety of features to help the programmer define control structures for her program.

• Prolog offers its search strategy as means for defining various control structures. How can this strategy be used to define conventional control structures?

• The Single Clause Structure

```
ripe(X) :- texture(X, soft),
   markings(X, speckles), aroma(X,sweet).
texture(parsimmon, hard).
texture(banana, soft).
markings(parsimmon, none).
markings(banana, speckles).
aroma(parsimmon, none).
aroma(banana, sweet).
```

Unclear where the goal `?- ripe(parsimmon)` fails.

Single clause definitions suitable as main clauses of a Prolog program

©1993 John Mylopoulos

2

Control Structures (continued)

• The Logical OR Structure

```
food(X) :- meat(X).
food(X) :- fish(X).
food(X) :- vegetable(X).
food(X) :- dairy(X).
food(X) :- grain(X).
food(X) :- fruit(X).
```

• The Exclusive OR Structure

```
food(X) :- meat(X), !.
food(X) :- fish(X), !.
food(X) :- vegetable(X), !.
food(X) :- dairy(X), !.
food(X) :- grain(X), !.
food(X) :- fruit(X), !.
```


©1993 John Mylopoulos

3

• The Fail-to-Backtrack Structure

```
printMonths :- month(X), nl, write(X),
               write(' is a month'), fail.
printMonths.
month(january).
month(february).
month(march).  etc.
```

• The Recursive Structure

```
fibo(0, 1).
fibo(1, 1).
fibo(N, F) :- N1 is N - 1, N2 is N - 2,
              fibo(N1, F1), fibo(N2, F2),
              F is F1 + F2.
```

• The Repeat-Fail Loop

```
adInfinitum :- repeat, fish(X), write(X),
          write(' is a fish'), nl, fail.
fish(cod).      fish(mackerel).

|?- adInfinitum.
cod is a fish
mackerel is a fish
cod is a fish....
```


©1993 John Mylopoulos

4

An Example

Suppose you have a database of employee salaries and you want to calculate the average salary and store it in the database:

```
salary(maria, 50).
salary(george, 56).
salary(jane, 68).
```

• Solution using no side-effects

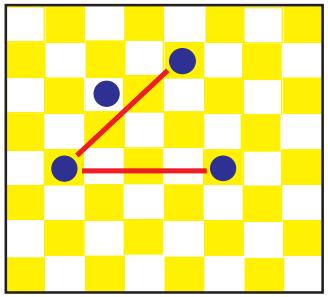
```
calcAvSal(S) :-
   bagof(Sal,Emp^salary(Emp,Sal),Set),
   average(Set,S).

average(Set, S) :- avgL(Set, 0, 0, S).

avgL([], Avg, _, Avg):- print(Avg).
avgL([H | T], X, N, Avg) :-
        X1 is (X*N + H) / N + 1,
        N1 is N + 1,
        avgL(T, X1, N1, Avg).
```


©1993 John Mylopoulos

5

Efficiency

The Eight Queens Problem



Place eight queens on a chessboard so that no two threaten each other

According to chess rules, a queen threatens another piece if they both lie on the same row, column or diagonal

©1993 John Mylopoulos

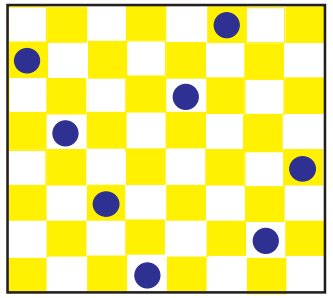
6

Efficiency

The Eight Queens Problem

• **Solutions:** There will be one queen in each column, so a solution can be represented by a vector of length 8 representing the row on which queens of column 1, 2, ..., 8 are placed. Moreover, no two queens can be assigned the same row**.**

A solution then is a permutation of the vector
[1, 2, 3, 4, 5, 6, 7, 8]



A solution: [7, 5, 3, 1, 6, 8, 2, 4]

©1993 John Mylopoulos

7

Towards a Solution

• Check that queen on column 2 does not threaten queen on column 1;
• Check that queen on column 3 does not threaten queen on columns 2,1;
...
• Check that queen on column i does not threaten queens on columns (i-1), (i-2),..., 2,1;
...

Given a vector representing the positions of 8 queens, say

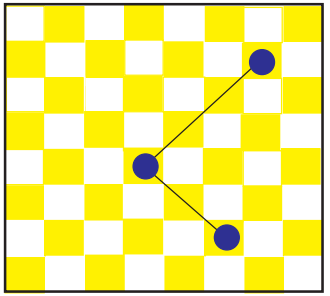
[3,6,4,1,8,2,5,7]

when does a queen threaten another?

©1993 John Mylopoulos

8

When Does a Queen Threaten Another?



|row(queen1) - row(queen2)| =
|col(queen1) - col(queen2)|

We don't need to worry about queens placed on the same row or column (why??)

©1993 John Mylopoulos

9

Solution to the Eight Queens Problem

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

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

noAttack(_ , [], _).
noAttack(Queen,[Queen1|Others],Xdist) :-
   Queen1 - Queen =/= Xdist,
   Queen - Queen1 =/= Xdist,
   Dist is Xdist + 1,
   noAttack(Queen, Others, Dist).
```

Efficiency Rule: Start from a problem statement that reduces the amount of search

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


©1993 John Mylopoulos

10

Efficiency: Another Example

Four Colour Problem: Colour the countries of Europe with four colours so that no two neighbours have the same colour.

Let's use a predicate `ngh(Country,Neighbours)` to represent the list of neighbours for `Country`:

```
ngh(albania, [greece, jugoslavia]).
ngh(austria, [czechoslovakia, hungary,
        italy, lichtenstein, switzerland,
        germany, jugoslavia]).
```

The solution will be given by a list of pairs

```
Country/Colour
```

which indicates the colour assigned to each country. If we assume that the colours are

```
[yellow, blue, red, green]
```

the solution will have the form

```
[albania/C1,andorra/C2,austria/C3,..]
```

with `C1, C2, C3`,...appropriately instantiated

....

©1993 John Mylopoulos

11

Efficiency: The Four Colour Problem

```
colours([]).
colours([Country/Colour | Rest] :-
   colours(Rest),
   member(Colour,
      [yellow, green, red,blue]),
   not(member(Country1/Colour, Rest),
           neighbour(Country, Country1)).

neighbour(Country, Country1) :-
   ngh(Country, Neighbours),
   member(Country1, Neighbours).
```

The Four Colour Problem can then be formulated as:

```
   country(C) :- ngh(C, _).

?- setof(Cntry/Colour,country(Cntry),
                      CountryColourList),
    colours(CountryColourList)
```

Prolog will try out an assignment of colours to countries in a random order (alphabetically?) leading to impossible situations (e.g., all countries surrounding a given country having different colours).

©1993 John Mylopoulos

12

Efficiency: The Four Colour Problem

The search space can be reduced drastically by starting from a country with many neighbours, say Germany, and then considering other countries in a breadth-first fashion.

```
makelist(List) :-
           collect([germany],[],List).

collect([], Closed, Closed).
collect([X | Open], Closed, List) :-
   member(X, Closed), !,
   collect(Open, Closed, List).
collect([X | Open], Closed, List) :-
   ngh(X, Nghs),
   append(Nghs, Open, Open1),
   collect(Open1, [X | Closed], List).
```

This returns a list of the countries ordered according to their relative position with respect to Germany (how topical!)

Now change the definition of `country`:

```
   country(C) :- makelist(L), member(C,L).
```


©1993 John Mylopoulos