Example predicates

Showing things to look for

Infinite loops

- Avoid circular definitions

  parent (A, B) :- child (B, A).

  child (C, D) :- parent (D, C).

- Easy to see here but as database grows you can forget what is in it and circularity can creep in

Infinite loops – Left Recursion – 1

- Left recursion can cause problems

  person (X) :- person (Y) , mother (Y, X).

  person (eve).

  - **The query person (P) loops indefinitely as the first rule is found first on every recursive call.**
  - **Second rule is only tried if first rule fails**

- Reordering the rules will correct the problem

Heuristic
Put facts before rules

Infinite loops – Left Recursion – 2

- Left recursion can cause problems – continued

  person (eve).

  person (X) :- person (Y) , mother (Y, X).

  - **Assuming mother fails, the query person (P) loops indefinitely after P = eve**

- Left recursion is the problem

Do not assume Prolog will find the facts and rules.
Need to know how searching works

Multiple answers – isList, weakList

◊ The following gives the following predicate works for specific lists but loops forever on the query **isList (X)**.

```
isList ( [ A | B ] )  :-   isList ( B ).
isList ( [ ] ).
```

◊ It can be defined by putting the fact first.

```
isList ( [ ] ).

isList ( [ A | B ] )  :-   isList ( B ).
```

◊ But gives more than one answer for the query **isList (X)** but does not loop forever.

◊ For the latter query, to have only one answer, can assert the following.

```
weak_isList ( [ ] ).
weak_isList ( [ _ | _ ] ).
```

Why is weak_isList weak?

- The strong definition says a list must have the correct structure and must end in nil.
- The weak definition simply says the list must have the correct structure for one level and says nothing about nil except for the empty list.
- For example – recall [...] is shorthand for the structure .(...)

isList (.(a , [])).	**==> yes**
isList (.(a , .(b , []))).	**==> yes**
isList (.(a , .(b, .(c , [])))).	**==> yes**
isList (.(a , b)).	**==> no**
isList (.(a , .(b , c , []))).	**==> no**

- But all responses are yes for weak_isList

Mapping

- ◊ Consider the problem of translating a sentence from one form to another
  - » For example as in the following "dialogue" the second sentence is a translation of the preceding sentence
    - > you are a computer
      I am not a computer
    - > do you speak french
      no I speak german
  - » Assume the following simplistic translations
    - > you ==> I
      are ==> am not
      do ==> no
      french ==> german

Mapping – 2

- Let us represent sentences as a list of words

  you are a computer ==> [you , are , a , computer]

- We represent the list of words to change as a set of change rules

```
change ( you , I ).
change ( are ,  [ am , not ] ).
change ( french , german ).
change ( do , no ).
change ( X , X ).      /* catch all to make no
                          changes */
```

Mapping – 3

- Then the translation rules can be the following.

  alter ([ ] , [ ]).
  alter ([H | T] , [X | Y]) :- change (H, X) , alter (T, Y).

- Then we can translate our example sentences

  alter ([you, are, a, computer] , Trans).

  > **Trans = [I , am , not , a , computer]**

  » **Try using ;<return> on the above. Explain why there are multiple answers. Try a trace to see what is happening.**

  > **We need a method to prevent multiple answers**

Mapping – 4

- Try the inverse – with **;<return>**

  alter (Org , [I , am , not , a , computer]).

- Try a variable – with ; <return>

  alter ([you , are , a , X] , Trans)

Warning – Caution – Danger

Logic and a finite database
can lead to strange
and unexpected results.
Use with extreme caution.