1

Knowledge Based Systems

(CM0377)

Lecture 3

(Last modified 5th February 2001)

2

Recall: Resolution - General pattern

- To answer a query

  $?-Q_1,Q_2,\ldots Q_n$

  find a rule

  $A:-B_1,\ldots,B_m$

  such that A matches with Q_1 and answer query:

  $?-B_1,\ldots B_m,Q_2,\ldots Q_n$

  instead *(resolution)*

3

More complicated rules

- Consider the goal *?- brother_of(X, fred).*

```
brother_of|B, fred|:-parent_of|P,B|,parent_of|P,fred|,male|B|,B\--fred.

   parent_of|bill,jim|, parent_of(bill,fred)
   parent_of|jane,jim|, parent_of|jane,fred|, male|jim|, jim\--fred  X=jim
   parent_of|bill,albert|, parent_of(bill, fred)
   parent_of|jane,albert|,parent_of|jane,fred|,male|albert|,
      albert\--fred                                          X=albert
   parent_of|jim,alice|, parent_of|jim,fred|, male(alice)
   parent_of|jane,alice|,parent_of|jane,fred|, male(alice)
   parent_of|jim,fred|,parent_of|jim,fred|,male|fred|, fred\--fred
   parent_of|jane,fred|,parent_of|jane,fred|,male|fred|, fred\--fred
   parent_of|jim,john|,parent_of|jim,fred|,male|john|,john\--fred   X=john
   parent_of|jane,john|,parent_of|jane,fred|,male|john|,john\--fred
                                                              X=john
```

4

Backtracking

- If Prolog fails to satisfy a sub-goal of a clause, it *backtracks* from right to left, searching for a sub-goal that can be satisfied differently.
- If there is more than one clause (e.g. the *fact* example that follows), then having failed completely on one clause Prolog proceeds to the next.

5

Factorial

```
fact(0,1).
fact(X, F):- X>0, X1 is X - 1,
             fact(X1,F1),
             F is X * F1.
```

NB

- `X is X-1` is **never** true: if X has a value, this tests whether X is equal to X-1; if not, X-1 can't be evaluated and Prolog crashes
- `X1 = X-1` makes X1 equal to a structure '-'(X,1). Use **is** to evaluate arith. expressions

6

Tracing the goal fact(2, X)

```
| ?- fact(2, X).
        1      1 Call: fact(2,_170) ?
        2      2 Call: 2>0 ?
        2      2 Exit: 2>0 ?
        3      2 Call: _659 is 2-1 ?
        3      2 Exit: 1 is 2-1 ?
        4      2 Call: fact(1,_651) ?
        5      3 Call: 1>0 ?
        5      3 Exit: 1>0 ?
        6      3 Call: _2650 is 1-1 ?
        6      3 Exit: 0 is 1-1 ?
        7      3 Call: fact(0,_2642) ?
?       7      3 Exit: fact(0,1) ?
        8      3 Call: _651 is 1*1 ?
        8      3 Exit: 1 is 1*1 ?
?       4      2 Exit: fact(1,1) ?
        9      2 Call: _170 is 2*1 ?
        9      2 Exit: 2 is 2*1 ?
?       1      1 Exit: fact(2,2) ?

X = 2 ?

yes
{trace}
| ?-
```

7

What's going on?

- Consider:

```
3      2 Call: _659 is 2-1 ?
```

- Left-most number is, in effect, the step number of the corresponding node in the proof tree being constructed
- Next number is the recursive depth
- _659 is an internal variable name ...

8

Recursive use of **fact**

- This clause is used twice in the proof:

```
fact(X, F):- X>0, X1 is X - 1,
             fact(X1,F1), F is X * F1.
```

- Internally, Prolog invents names for each distinct variable. First time, the goal is:

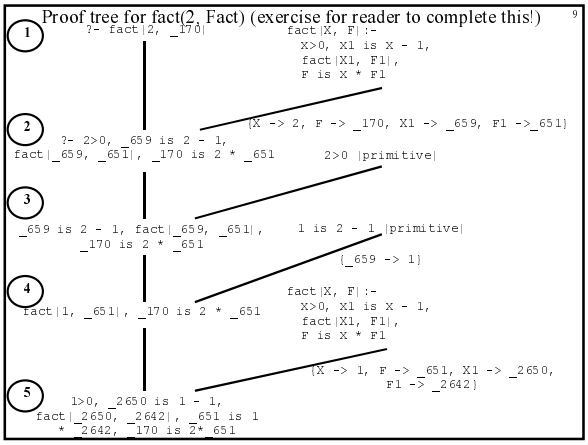
```
fact(2, _170):- 2>0, _659 is 2 - 1,
             fact(_659,_651), _170 is 2 * _651.
```

- Second time, by which _659 is bound to the value 1, the goal is:

```
fact(1, _651):- 1>0, _2650 is 1 - 1,
             fact(_2650,_2642), _651 is 1 * _2642.
```

9

Proof tree for fact(2, Fact) (exercise for reader to complete this!)



10

The box model of goal execution

- CALL port: initial invocation of goal
- EXIT: successful return from goal
- REDO: backtrack to find another solution
- FAIL: the goal couldn't be satisfied any more times

11

Structured terms

- Prolog allows us to have record-like, nested structures, e.g.
- owns(bill, book('A.N. Other', 'The laws of confusion', 'Thomson', 1993))
- These can be used to determine behaviour that is specific to the form of the term, i.e. its *functor* ('book', in the above example) and *arguments.*

12

Example

```
display_item(book(Auth, Title, Pub, Yr)):-
   write('a book: '), nl,
   write(Auth), nl, write('"'), write(Title),
   write('" Publisher: '), write(Pub), write(' Year: '),
   write(Yr), nl.

display_item(car(Manuf, Mod, Miles, Yr)):-
   write('a car: '), nl,
   write(Manuf), write(' '),
   write(Mod), write(' Mileage: '),
   write(Miles), write(' Year: '), write(Yr), nl, nl.
```

- (See 'owns' program on separate sheet)

13

NB

- **write(<atom>)** writes out an atom;
- **nl** moves to a new line;
- **fail** simply fails.

14

NB (ctd.)

- It's always best to implement deterministic predicates (things that perform a task, rather than do reasoning) so as to succeed. Hence:

```
go:-
  owns(X, Item), write(X),
  write(' owns '),
  display_item(Item), fail.
go.
```

15

Thus we can have lists!

```
parent_of(bill, list(jim, list(albert,
  empty))).

father_of(F, Ch):-
  parent_of(F, List), member(Ch, List),
  male(F).

member(X, list(X, _)).
member(X, list(_, Others)):-
     member(X, Others).
```

16

Prolog's special list notation

- A list in Prolog is a sequence of items surrounded by square brackets, e.g.
  `[1, 2, 3]`
- which is equivalent, entirely, to:
  '.'(1, '.'(2, '.'(3, [])))
- Empty list has the special representation []

17

List notation (ctd.)

- Vertical bar '|' in a list - what's to the right is the remainder of the list. So the following patterns match:
- [1, 2, 3] = [X | Y], where X=1 and Y=[2, 3]
- [1, 2, 3] = [X, Y | Z], where X=1, Y=2 and Z= [3]
- [a, b] = [X, Y|Z] where X=a, Y=b and Z=[]

18

Revised genealogy program

```
parent_of(jim, [alice, fred, john]).
parent_of(jane, '.'(jim, '.'(albert,
  [])));.

father_of(F, Ch):-
  parent_of(F, List), member(Ch, List),
  male(F).

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

19

Unassessed exercise (highly recommended!)

- Download the files for today's examples: **fact.pl**, **owns.pl**, **new_fam.pl** and **new_fam1.pl**
- and try them out using SICStus.
- Things you need to know:
  - start Prolog by typing: **sicstus**
  - load programs in by typing **[<file-name>]**, but leaving off the '.pl'
  - run goals by typing the goal followed by full stop
  - exit Prolog by typing <CTRL>D
  - break into execution by <CTRL>C, then type **a<return>** to abort execution.

20

Things you need to know (ctd.)

- to turn tracing on, type the goal:
  ?- **trace.**
- to turn it off again, type the goal:
  ?- **notrace.**
- you can browse the Sicstus manuals at:

http://www.sics.se/isl/sicstus/docs/3.8/html/sicstus.html