University of Auckland

07.363

Functional and Logic Programming

Christian Collberg
October 7, 1994

The Prolog Database

Manipulating the Database

- Allows us to program with side effects.
- Don't do it unless you have to...
- Justified under some circumstances.
- Some programs by definition need to modify the database:
  - Learning Programs
- Often inefficient.

Slide 8–1

Assert I

- `assert(X)` adds a clause to the database.
- `asserta(X)` adds a clause to the *beginning* of the database.
- `assertz(X)` adds a clause to the *end* of the database.
- Always succeeds.
- Backtracking does not undo the assertion.

Slide 8–2

Assert II – Example

```
location(whitehorse, 8, 11).
location(microsoft, 8, 42).
location(condomeria, 8, 43).
location(plunket, 7, 32).

loc(X, Ave, Str) :-
   location(X, Ave, Str), !.
loc(X, Ave, Street) :-
   nonvar(X), var(Ave), var(Str),
   write('-- this '), write(X), nl,
   write('what avenue?  '),
   read(Ave),
   write('what street?  '),
   read(Street),
   assert(location(X, Ave, Str)).
```

Slide 8–3

Assert III – Example

```
?- loc(whitehorse, Ave, St).
   Ave = 8
   St = 11

?- loc(airport, Ave, St).
   -- this airport
   what avenue?  5.
   what street?  32.
   Ave = 5
   St = 32

?- loc(airport, Ave, St).
   Ave = 5
   St = 32
```

Slide 8–4

Retract I

- `retract(X)` removes the first clause that matches `X`.
- When we try to resatisfy `retract(X)`, Prolog removes the next matching clause.
- `retract(X)` fails when no matching clause can be found.
- Backtracking does not undo the removal.

```
retractall(X) :- retract(X), fail.
retractall(X) :- retract((X :- Y))),
                 fail.
retractall(_).
```

Slide 8–5

Clause I

- `clause(X, Y)` finds all clauses in the database with head `X` and body `Y`.

```
append([], X, X).
append([A|B],C,[A|D]) :-
      append(B, C, D).

?- clause(append(X, Y, Z), T).
   X=[], Y=_3, Z=_3, Y=true ;
   X=[_4|_5], Y=_6, Z=[_4|_7],
      Y=append(_5, _6, _7) ;
   no
```

Slide 8–6

Clause II

List all the clauses whose head matches `X`.

```
list(X) :- clause(X, Y),
   print(X, Y),
   write('.'), nl, fail.
list(_).

print(X, true) :- !, write(X).
print(X, Y) :- write((X :- Y))).

?- list(append(X, Y, Z)).
   append([], _4, _4).
   append([_5|_6],_7,[_5|_8]) :-
         append(_6, _8, _8).
```

Slide 8–7

Slide 8–8

Clauses as Data Structures I – Lists

```
list(c).
list(h).
list(r).
list(i).
list(s).

process_list :- list(X), process_item(X), fail.
process_list.
```

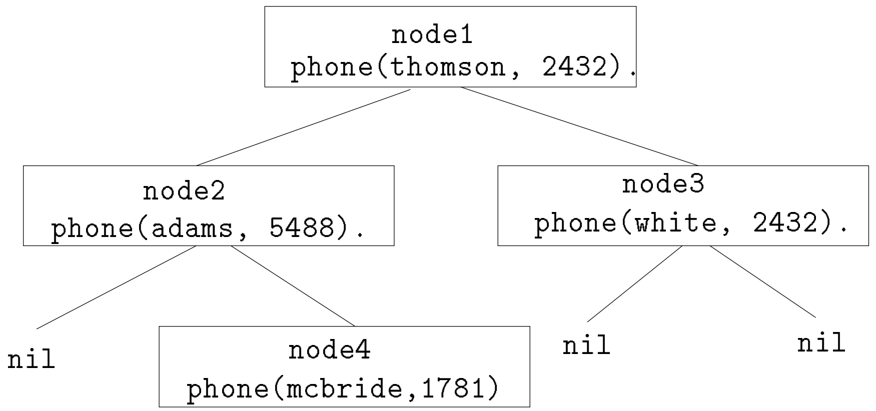
Slide 8–9

Clauses as Data Structures II – Trees

```
t(node1, node2, phone(thompson, 2432), node3).
t(node2, nil, phone(adams, 5488), node4).
t(node3, nil, phone(white, 2432), nil).
t(node4, nil, phone(mcbride, 1781), nil).
```

Slide 8–10

Clauses as Data Structures III – Trees



Slide 8–11

Clauses as Data Struct. IV – Trees

```
inorder(nil).
inorder(Node) :-
    t(Node, Left, P, Right),
    inorder(Left),
    write(P), nl,
    inorder(Right).

?- inorder(node1).
   phone(adams,5488)
   phone(mcbride,1781)
   phone(thompson,2432)
   phone(white,2432)
```

Switches I

```
turnon(Switch) :-
   call(Switch), !.
turnon(Switch) :-
   assert(Switch).

turnoff(Switch) :-
   retract(Switch).
turnoff(_).

flip(Switch) :-
   retract(Switch), !.
flip(Switch) :-
   assert(Switch).
```

Slide 8–12

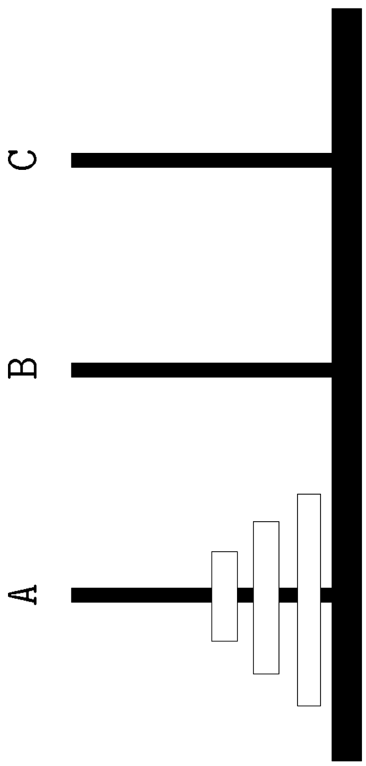
Switches II

```
turnon(terse_mess).
   ....
flip(terse_mess).

message(C) :- terse_mes, write ('Error!'), nl, !.
message(C) :- write ('We are sorry to...'),
   write ('error has occurred near the symbol '),
   write(C), write('.  Please accept our...'),
   nl, !.
```

Slide 8–13

Memoization – Towers of Hanoi I



Slide 8–14

Memoization – Towers of Hanoi II

```
:- op(100, xfx, to).

hanoi(1, A, B, C, [A to B]).
hanoi(N, A, B, C, Ms) :-
   N > 1,
   N1 is N-1,
   hanoi(N1, A, C, B, M1),
   hanoi(N1, C, B, A, M2),
   append(M1, [A to B|M2], Ms).

go(N, Moves) :-
   hanoi(N, a, b, c, Moves).
```

Slide 8–15

Memoization – Towers of Hanoi III

```
?- go(2,M).
   M = [a to c, a to b, c to b]

?- go(3,M).
   M = [a to b, a to c, b to c,
      a to b, c to a, c to b,
      a to b]

?- go(4,M).
   M = [a to c, a to b, c to b,
      a to c, b to a, b to c,
      a to c, a to b, c to b,
      c to a, b to a, c to b,
      a to c, a to b, c to b]
```

Slide 8–16

Memoization – Towers of Hanoi IV

```
hanoi(1, A, B, C, [A to B]).
hanoi(N, A, B, C, Ms) :-
   N > 1, R is N-1,
   lemma(hanoi(R, A, C, B, M1)),
   hanoi(N1, C, B, A, M2),
   append(M1, [A to B|M2], Ms).

lemma(P) :- call(P),
         asserta((P :- !)).

go(N, Pegs, Moves) :-
   hanoi(N, A, B, C, Moves),
   Pegs=[A, B, C].
```

Slide 8–17

Memoization – Towers of Hanoi V

```
hanoi(1, _3, _5, _4, [_3 to _5]) :- !.
hanoi(2, _3, _4, _5,
   [_3 to _5, _3 to _4, _5 to _4]) :- !.
hanoi(3, _3, _5, _4,
   [_3 to _5, _3 to _4, _5 to _4,
    _3 to _5, _4 to _3, _4 to _5,
    _3 to _5]) :- !.
```

Slide 8–18

Example – Gensym I

```
gensym(Root, Atom) :-
   get_num(Root, Num),
   name(Root, Name1),
   int_name(Num, Name2),
   append(Name1, Name2, Name),
   name(Atom, Name).

get_num(Root, Num) :-
   retract(current_num(Root, Num1)),
   !, Num is Num1 + 1,
   asserta(current_num(Root, Num)).
get_num(Root, 1) :-
   asserta(current_num(Root, 1)).
```

Slide 8–19

Example – Gensym II

```
int_name(Int, List) :-
   int_name(Int, [], List).
int_name(I, Sofar, [C|Sofar]) :-
   I<10, !, C is I+48.
int_name(I, Sofar, List) :-
   Tophalf is I/10,
   Bothalf is I mod 10,
   C is Bothalf + 48,
   int_name(Tophalf, [C|Sofar], List).

?- gensym(chris, A).
   A = chris1
?- gensym(chris, A).
   A = chris2
?- gensym(chris, A).
   A = chris3
```

Slide 8–20