Prolog
Programming in Logic

Lecture #7

Ian Lewis, Andrew Rice

Today's discussion

Videos

- Difference
- Empty difference lists
- Difference list example

Q: Attempting the towers of hanoi problem and run into stack space issues which makes me think my state representation is bad, any hints (specific and general)?

Q: Attempting the towers of hanoi problem and run into stack space issues which makes me think my state representation is bad, any hints (specific and general)?

A: if you are running out of stack space you probably have a searching-forever problem are more rules matching than you thought? General state representation hint: as little redundancy as possible.

```
% Towers of Hanoi

% Represent the current state as rings(A,B,C) where
%  A is the peg that the smallest ring is on
%  B is the peg that the middle ring is on
%  C is the peg that the largest ring is on
% Start rings(1,1,1). Finish rings(3,3,3).

range(Min,_,Min).
range(Min,Max,Next) :- N2 is Min+1, N2 < Max, range(N2,Max,Next).

move(rings(Src,A,B),rings(Dest,A,B)) :-
       range(1,4,Src),
       range(1,4,Dest),
       Src \= Dest.

move(rings(A,Src,B),rings(A,Dest,B)) :-
       range(1,4,Src),
       range(1,4,Dest),
       A \= Src, A \= Dest.

move(rings(A,B,Src),rings(A,B,Dest)) :-
       range(1,4,Src),
       range(1,4,Dest),
       A \= Src, A \= Dest,
       B \= Src, B \= Dest.

search(Dest,Dest,_,[]).
search(Src,Dest,Closed,[Mid|Path]) :-
       move(Src,Mid),
       \+member(Mid,Closed),
       search(Mid,Dest,[Mid|Closed],Path).

solve :- search(rings(1,1,1),rings(3,3,3),[rings(1,1,1)],Path),
         print([rings(1,1,1)|Path]).
```

```
% Hanoi puzzle
% Rings number 1,2,3.
% Each tower A,B,C a list of rings (Head = top).
% State stored as state(A, B, C).
% Start state([1,2,3],[],[]). Finish state([],[],[1,2,3]).

% make_move(+Tower1,+Tower2, -Tower1_after, -Tower2_after).
% Will only make valid moves, i.e. onto empty or bigger tower.
make_move([A1|A],[],A,[A1]).
make_move([A1|A],[B1|B],A,[A1,B1|B]) :- A1 < B1.

% move(+State_before, -State_after)
% Generate valid moves
% move A->B or A->C or B->A or B->C or C->A or C->B.
move(state(A,B,C), state(AN,BN,C)) :- make_move(A,B,AN,BN).
move(state(A,B,C), state(AN,B,CN)) :- make_move(A,C,AN,CN).
move(state(A,B,C), state(AN,BN,C)) :- make_move(B,A,BN,AN).
move(state(A,B,C), state(A,BN,CN)) :- make_move(B,C,BN,CN).
move(state(A,B,C), state(AN,B,CN)) :- make_move(C,A,CN,AN).
move(state(A,B,C), state(A,BN,CN)) :- make_move(C,B,CN,BN).

search(State_from, State_to, Path) :- move(State_from,State_to),
                                      print(Path).

search(State_from, State_to, Path) :-
       move(State_from, Next_state),
       \+ member(Next_state, Path),
       search(Next_state,State_to,[Next_state|Path]).

solve :- search(state([1,2,3],[],[]),state([],[],[1,2,3]), []).
```

Q. N > 0 ? extra-logical ?

```
choose(0, L, [], L).
choose(N, [H|T], [H|R], S) :- N > 0, N2 is N-1, choose(N2, T, R, S).
choose(N, [H|T], R, [H|S]) :- N > 0, choose(N, T, R, S).
```

```
:- choose(2, [a,b,c,d], Chosen, Remaining).

:- choose(0, [a,b,c,d], Chosen, Remaining).
```

Q. N > 0 ? extra-logical ?

```
choose(0, L, [], L).
choose(N, [H|T], [H|R], S) :- N > 0, N2 is N-1, choose(N2, T, R, S).
choose(N, [H|T], R, [H|S]) :- N > 0, choose(N, T, R, S).

% Trace version
choose2(0, L, [], L) :-
     print('clause 1- success'), nl.
choose2(N, [H|T], [H|R], S) :-
     print('clause 2-  N='), print(N), print(' from '), print([H|T]), nl,
     N2 is N-1, choose2(N2, T, R, S).
choose2(N, [H|T], R, [H|S]) :-
     print('clause 3-  N='), print(N), print(' from '), print([H|T]), nl,
     choose2(N, T, R, S).
```

Difference lists

Difference lists

Which of these is a difference list:

1. diff(A,B)
2. A-B
3. [1,2,3|A]-A
4. [1,2,3|A]-B
5. []-[]
6. A-A

Difference lists

Which of these is a difference list:

1. diff(A,B)
2. A-B
3. [1,2,3|A]-A
4. [1,2,3|A]-B
5. []-[]
6. A-A

A gentle introduction to difference lists.

```
?- X = [1,2,A,4,5].
X = [1,2,_4096,4,5].
```

A gentle introduction to difference lists.

```
?- X = [1,2,A,4,5].
X = [1,2,A,4,5].
```

A gentle introduction to difference lists.

```
?- X = [1,2,A,4,5], A = woohoo. -- retrospectively fill in the hole.
X = [1,2,woohoo,4,5],
A = woohoo.
```

A gentle introduction to difference lists.

```
?- X = [1,2,A,4,5], A = woohoo.
X = [1,2,woohoo,4,5],
A = woohoo.
```

That's great! But what's the point ????

You can pass around as-yet-incomplete data structures.

e.g. you can add an element to the Tail of a list (the canonical example).

You get to hone your unification comprehension.

A gentle introduction to difference lists.

```
?- X = [1,2,3|A].
X = [1,2,3|A].          --- A list with a hole in it...

                          -- Retrospectively fill in the tail...
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|A].
X = [1,2,3|A].        --- A list with a hole in it...

The tail of a list is always a list. So what about:
?- X = [1,2,3|7].
X = [1,2,3|7].

? X = [1,2,3|7], X = [A,B,C].
false

? X = [1,2,3|7], X = [A,B,C,D].
false
```

In fact you just have a compound term |(3,7) stuck on the end of the list and all the relations expecting |(last_element,[]) simply fail. Depends on implementation.

So [1,2,3|7] IS NOT A LIST.

A gentle introduction to difference lists.

Very few of you will *write* a list [1,2,3|7]... it arises from:

```
?- X = [1,2,3|A], A=7.
```

Correct would be A = [7].

A gentle introduction to difference lists.

A more significant / common / relevant example:

Set the difflist var as the empty list.

```
?- X = [1,2,3|A], A=[].
X = [1,2,3],
A = [].                               -- we are simply terminating the list.
```

A gentle introduction to difference lists.

A more significant / common / relevant example:

```
?- X = [1,2,3|A], A=[].
X = [1,2,3],
A = [].                    -- we are simply terminating the list.
```

For the avoidance of doubt, the 'X' list is equally:

```
?- X = [ 1 | [ 2 | [ 3 | [] ] ] ].
X = [1,2,3].
```

A gentle introduction to difference lists.

```
With two lists:

?- X = [1,2,3|A], Y = [4,5,6|B].
X = [1,2,3|A],
Y = [4,5,6|B].
```

A gentle introduction to difference lists.

With two lists:

```
?- X = [1,2,3|A], Y = [4,5,6|B].
X = [1,2,3|A],
Y = [4,5,6|B].
```

Linking the lists...

```
?- X = [1,2,3|A], Y = [4,5,6|B], A = Y.
X = [1,2,3,4,5,6|B],  -- so we have managed to append, via unification
A = Y,
Y=[4,5,6|B].
```

A gentle introduction to difference lists.

This is great! How to write an append ?

```
?- X = [1,2,3|A], Y = [4,5,6|B], append(X,Y,Z).

append([],Y,Y).
append(?,?,?) :- ...
```

A gentle introduction to difference lists.

This is great! How to write an append ?

```
?- X = [1,2,3|A], Y = [4,5,6|B], append(X,Y,Z).
```

```
append([],Y,Y).
append(?,?,?) :- ...
```

You can't, is the short answer... you need to propagate a reference to A and B.

A gentle introduction to difference lists.

```
?- X = [1,2,3|A]-A, Y = [4,5,6|B]-B, append(X,Y,Z).
```

So you can pass the list and its tail var as a single compound term.

```
append(X,Y,Z) :- X = XL-XVar,   -- get the list/var components of X
                 Y = YL-YVar,   -- get the list/var components of Y
                 Z = ZL-ZVar,   -- make a new diff list for Z
                 XVar = YL,     -- unify the X var with the Y list
                 ZL = XL,       -- unify the X list with the Z list
                 ZVar = YVar.   -- make the var of Z as for Y
?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|XVar]-XVar, Y = [4,5,6|YVar]-YVar, append(X,Y,Z).

So you can pass the list and its tail var as a single compound term.
Collapse the 'parsing' unifications into the head of the clause:
append(X,Y,Z) :- X = XL-XVar,   -- get the list/var components of X
                 Y = YL-YVar,   -- get the list/var components of Y
                 Z = ZL-ZVar,   -- make a new diff list for Z
                 XVar = YL,     -- unify the X var with the Y list
                 ZL = XL,       -- unify the X list with the Z list
                 ZVar = YVar.   -- make the var of Z as for Y
?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|XVar]-XVar, Y = [4,5,6|YVar]-YVar, append(X,Y,Z).

So you can pass the list and its tail var as a single compound term.
Collapse the 'parsing' unifications into the head of the clause:
append(XL-XVar,YL-YVar,ZL-ZVar) :-
                 XVar = YL, -- unify the X var with the Y list
                 ZL = XL, -- unify the X list with the Z list
                 ZVar = YVar. -- make the var of result the same as Y

?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|XVar]-XVar, Y = [4,5,6|YVar]-YVar, append(X,Y,Z).

Now we can propagate the remaining unifications:

append(XL-XVar,YL-YVar,ZL-ZVar) :-
                 XVar = YL, -- unify the X var with the Y list
                 ZL = XL,
                 ZVar = YVar. -- make the var of result the same as Y

?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|XVar]-XVar, Y = [4,5,6|YVar]-YVar, append(X,Y,Z).




append(XL-XVar,XVar-YVar,ZL-ZVar) :-
                 ZL = XL,
                 ZVar = YVar. -- make the var of result the same as Y

?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|XVar]-XVar, Y = [4,5,6|YVar]-YVar, append(X,Y,Z).




append(XL-XVar,XVar-YVar,ZL-ZVar) :-
                 ZL = XL,
                 ZVar = YVar. -- make the var of result the same as Y

?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|XVar]-XVar, Y = [4,5,6|YVar]-YVar, append(X,Y,Z).



Rename XL to A:
append(XL-XVar,XVar-YVar,XL-YVar).

?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|XVar]-XVar, Y = [4,5,6|YVar]-YVar, append(X,Y,Z).
```

Rename XVar to B:

```
append(A-XVar,XVar-YVar,A-YVar).

?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|XVar]-XVar, Y = [4,5,6|YVar]-YVar, append(X,Y,Z).
```

```
Rename YVar to C:
append(A-B,B-YVar,A-YVar).
```

```
?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
?- X = [1,2,3|B]-B, Y = [4,5,6|C]-C, append(X,Y,Z).
```

```
append(A-B,B-C,A-C).
```

```
?- app([1,2,3|A]-A,[4,5,6|B]-B,C).
C = [1, 2, 3, 4, 5, 6|B]-B,
A = [4, 5, 6|B].
```

A gentle introduction to difference lists.

```
% Final empty diff list list thoughts.

?- X = [a,b,c|A]-A, A = [], X = MyList-_.
MyList = [a,b,c].

? X = A-A, A=[], X = MyList-_.
MyList = [].
```

Empty diff list is ALWAYS A-A. But to TEST for it you attempt a unification with something that only matches <freevar>-<freevar>.

FWIW I think of diff lists a bit like complex numbers - with real and the imaginary parts. Ultimately you're interested in the real part.

Challenge: Implement Quicksort

Partition the list into two pieces

Quicksort each half

Implement Quicksort

```
% partition(+Pivot,+List,-Left,-Right) succeeds if Left is all the
% elements in List less than or equal to the pivot and Right is
% all the elements greater than the pivot

% quicksort(+L1,-L2) succeeds if L2 contains the elements in L1 in
% ascending order
```

Implement partition

```
% partition(+Pivot,+List,-Left,-Right) succeeds if Left is all the
% elements in List less than or equal to the pivot and Right is
% all the elements greater than the pivot

partition(_,[],[],[]).
partition(P,[H|T],[H|L],R) :- P <= H, partition(P,T,L,R).
partition(P,[H|T],L,[H|R]) :- P > H, partition(P,T,L,R).

    :- partition(5, [1,3,5,7,9], Left, Right).
    Left = [1,3,5]
    Right = [7,9]
```

Implement quicksort

```
partition(_,[],[],[]).
partition(P,[H|T],[P|L],R) :- P <= H, partition(P,T,L,R).
partition(P,[H|T],L,[P|R]) :- P > H, partition(P,T,L,R).

quicksort([],[]).
quicksort([P|T],Sorted) :-
    partition(P,T,L,R),
    quicksort(L,L1), quicksort(R,R1),
    append(L1,R1,Sorted).
```

Is it useful to turn this into difference lists?

```
partition(_,[],[],[]).
partition(P,[H|T],[P|L],R) :- P <= H, partition(P,T,L,R).
partition(P,[H|T],L,[P|R]) :- P > H, partition(P,T,L,R).

quicksort([],[]).
quicksort([P|T],Sorted) :-
    partition(P,T,L,R),
    quicksort(L,L1), quicksort(R,R1),
    append(L1,[P|R1],Sorted).
```

Step 1: Replace appended lists with difference lists

```
quicksort([],[]).
quicksort([P|T],Sorted) :-
    partition(P,T,L,R),
    quicksort(L,L1), quicksort(R,R1),
    append(L1,[P|R1],Sorted).
```

What do you notice?

Step 1: Replace appended lists with difference lists

```
quicksort([],A-A).
quicksort([P|T],Sorted-S2) :-
    partition(P,T,L,R),
    quicksort(L,L1-L2), quicksort(R,R1-R2),
    append(L1-L2,[P|R1]-R2,Sorted-S2).
```

The input list remains a 'normal' list.

Step 2: Worry about empty difference lists

```
quicksort([],A-A).
quicksort([P|T],Sorted-S2) :-
    partition(P,T,L,R),
    quicksort(L,L1-L2), quicksort(R,R1-R2),
    append(L1-L2,[P|R1]-R2,Sorted-S2).
```

Should this be []-[] or A-A ?

Step 2: Worry about empty difference lists

```
quicksort([],A-A).
quicksort([P|T],Sorted-S2) :-
    partition(P,T,L,R),
    quicksort(L,L1-L2), quicksort(R,R1-R2),
    append(L1-L2,[P|R1]-R2,Sorted-S2).
```

Should this be []-[] or A-A ?

A-A because we are RETURNING an empty list, not TESTING for it.

We will call quicksort(+L,-Sorted) with the answer terminated, i.e.:

```
?- quicksort([2,5,3,9,4,6],Ans-[]).
```

Step 3: Substitutions to make the append irrelevant

Replace L2 with [P|R1]

```
append(A-B,B-C,A-C).

quicksort([],A-A).
quicksort([P|T],Sorted-S2) :-
    partition(P,T,L,R),
    quicksort(L,L1-[P|R1]), quicksort(R,R1-R2),
    append(L1-[P|R1],[P|R1]-R2,Sorted-S2).
```

Step 3: Substitutions to make the append irrelevant

Replace Sorted with L1

```
append(A-B,B-C,A-C).

quicksort([],A-A).
quicksort([P|T],L1-S2) :-
    partition(P,T,L,R),
    quicksort(L,L1-[P|R1]), quicksort(R,R1-R2),
    append(L1-[P|R1],[P|R1]-R2,L1-S2).
```

Step 3: Substitutions to make the append irrelevant

Replace S2 with R2

```
append(A-B,B-C,A-C).

quicksort([],A-A).
quicksort([P|T],L1-R2) :-
    partition(P,T,L,R),
    quicksort(L,L1-[P|R1]), quicksort(R,R1-R2),
    append(L1-[P|R1],[P|R1]-R2,L1-R2).
```

Step 3: Substitutions to make the append irrelevant

```
append(A-B,B-C,A-C).

quicksort([],A-A).
quicksort([P|T],L1-R2) :-
    partition(P,T,L,R),
    quicksort(L,L1-[P|R1]), quicksort(R,R1-R2),
    append(L1-[P|R1],[P|R1]-R2,L1-R2).
    append(A - B ,  B - C, A-C).
```

Replace S2 with R2

Step 4: Remove the append because it doesn't do anything any more.

```
% partition(+Pivot,+List,-Left,-Right).
partition(_,[],[],[]).
partition(P,[H|T],[H|L],R) :- H =< P, partition(P,T,L,R).
partition(P,[H|T],L,[H|R]) :- H > P, partition(P,T,L,R).

% quicksort(+List,-DiffList)
quicksort([],A-A).
quicksort([P|T],L1-R2) :-
    partition(P,T,L,R),
    quicksort(L,L1-[P|R1]), quicksort(R,R1-R2).
```

Step 4: Remove the append because it doesn't do anything any more.

```
% partition(+Pivot,+List,-Left,-Right).
partition(_,[],[],[]).
partition(P,[H|T],[H|L],R) :- H =< P, partition(P,T,L,R).
partition(P,[H|T],L,[H|R]) :- H > P, partition(P,T,L,R).

% quicksort(+List,-DiffList)
quicksort([],A-A).
quicksort([P|T],L1-R2) :-
    partition(P,T,L,R),
    quicksort(L,L1-[P|R1]), quicksort(R,R1-R2).

?- quicksort([2,5,3,9,4,6],Ans-[]).
```

Next time

Videos

- Sudoku
- Constraints