Logic and Functional Programming

Labwork 12: Deep lists. Difference lists. Applications

1 Deep lists

A deep list is a recursive datatype defined by the grammar:

dlist `::=` `[]` `|` `[`*h*`|`*dlist*`]` where
h `::=` *atom* `|` *number* `|` *string* `|` *dlist*

Note that *dlist* is a deep list if and only if it is a list made of atoms, numbers, strings, anf deep lists.

The program ListApps.pl contains, among other things, the implementations of the following predicates for deep lists:

- `depth(+DL,-N)` which instantiates `N` with the depth of the deep list `DL`. For example,

  ```
  ?- depth([],N).  ?- depth([[1,[2,3]],[[],[[4,5],6,[7]]]],N).
  N = 1.           N =  4.
  ```

- `flatten(+DL,-SL)` which instantiates `SL` with the shallow list produced by flattening the deep list `DL`. For example,

  ```
  ?- flatten([[1,[2,3]],[[],[[4,5],6,[7]]]],SL).
  SL = [1,2,3,4,5,6,7].
  ```

Proposed exercises I

Define the following predicates on deep lists:

1. `heads(DL,Hs)` which instantiates `Hs` with the list of all elements which are at the head of a shallow list in `DL`. For example,

```
heads([1,[2,[[3],4],[5,[],6]]],L).
L = [1,2,3,5].
```

2. `member1(X,DL)` which holds if `X` occurs, at any depth, as an element of `DL`. For example,

```
member1([3],[[a,b],[2,[[3],4],[5,[a,b],6]]]).
true.
```

3. `member2(X,DL)` which holds if `X` is non-list which occurs, at any depth, as an element of `DL`. For example,

```
member2(a,[2,[[3],4],[5,[a,b],6]]]).
true.
```

2 Difference lists

An **open list** is a data structure of the form

$openList$ `::=` H `|` `[`$term_1, \ldots, term_n$`|`H`]`

where H is a free variable. Note that an open list is not a list, because lists must end with the empty list.

A **difference list** is a data structure of the form

$diffList$ `::=` `dList(`$openList, H$`)`

where $openList$ is an openList: either H or `[`$term_!, \ldots, term_n|H$`]`.

- `dList(H,H)` represents the empty list `[]`.
- `dList([`$term_1, \ldots, term_n$`|H],H)` represents the list `[`$term_1, \ldots, term_n$`]`.
- The free variable `H` is like a pointer to the end of the list.

The program ListApps.pl contains, among other things, the implementations of the following predicates for difference lists:

- `dAdd(+DL1,+DL2,-DL)`: binds `DL` to the deep list which represents the result of appending the deep lists `DL1` and `DL2`. For example,

  ```
  ?- dAdd([1,2,3,4|H1],[5,6,7|H2],DL).
  H1 = [5,6,7|H2],
  DL = dList([1,2,3,4,5,6,7|H2],H2).
  ```

- `addToEnd(+DL,+E,-L)` binds `L` to the list obtained from `DL` by adding element `E` at its end. For example,

  ```
  ?- addToEnd(dList([1,2,3|H],H),4,L).
  H=[4],
  L=[1,2,3,4].
  ```

- `member_open(?X,+DL)` checks if `X` is an element of the list represented by the deep list `DL`. For example,

  ```
  ?- member_open(X,dList([1,2|H],H)).
  X=1 ;
  X=2 ;
  false.
  ```

We considered binary trees defined by the grammar

btree `::=` `nil` `|` `bt`(*integer*, *btree*, *btree*)

and defined the following predicate on them (see Lecture 12):

- `inorder(+BT,-L)` binds `L` to the list of numbers in the binary tree `BT`, in the order given by the inorder traversal of `BT`. The predicate is implemented efficiently with difference lists.

We considered mazes consisting of rooms connected by doors, and represented by facts of the form

```
door1(A,B).    % there is a door between rooms A and B
```

and defined the following predicates for mazes:

- `go(+X,+Y,-Trail)`: binds `Trail` to a trail (or path) from `X` to `Y`, if there is one,

  A **trail** from `X` to `Y` is a list $[X_1, X_2, \ldots, X_n]$ of rooms, such that $X_1 =$ `X`,$X_n =$ `Y`, and for every $1 \leq 1 < n$, there is a door between rooms X_i and X_{i+1}.

- `goV2(+X,+Y,-Trail)`: does the same thing as `goV2(+X,+Y,-Trail)`, but it is more efficient because it is implemented with difference lists.

- `goBF(+X,+Y,-Trail)`: finds a shortest trail from `X` to `Y`, if there is one, with breadth-first search strategy. Here, 'shortest' means 'minimum number of edges'.

Proposed exercises II

Define the following predicates with difference lists:

1. `flatten(DL,SL)` which instantiates `SL` with the shallow list produced by flattening the deep list `DL`.

2. `preorder(BT,L)` which instantiates `L` with the list of nodes in binary tree `BT` produced by the preorder traversal of `BT`. For example,

```
?- preorder(bt(3,bt(1,bt(5,nil.nil),bt(7,nil,nil))
                bt(4,nil,nil)),L).
L=[3,1,5,7,4].
```

3. `postorder(BT,L)` which instantiates `L` with the list of nodes in binary tree `BT` produced by the postorder traversal of `BT`. For example,

```
?- postorder(bt(3,bt(1,bt(5,nil.nil),bt(7,nil,nil))
                 bt(4,nil,nil)),L).
L=[4,5,7,1,3].
```