Department of Informatics Engineering

Department of Computer Science

Functional and Logic Programming

Bachelor in Informatics and Computing Engineering
2021/2022 – 1st Semester

Prolog

Data Structures / Incomplete Data Structures

13/01/2022

Daniel Castro Silva
dcs@fe.up.pt

Agenda

- Data Structures
  - Binary Trees
- Incomplete Data Structures
  - Difference Lists

Data Structures

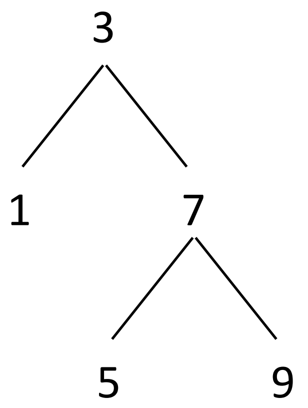
- Even though Prolog doesn’t explicitly define types or data structures, terms can be used to do so
  - Unary predicates can be used to ‘define a type’
    - The type *male* can be defined as the set of terms *X* such that *male(X)* is true
  - Simple types can be defined recursively
    - Lists, Trees, …
  - *Pairs* are typically represented as *X-Y*
  - *Tuples* can be represented as *(X, Y, Z)*
    - However, a properly named functor should be used
  - More complex terms can be used to represent data structures

```
book(‘River God’, author(smith, wilbur, 1933), 1993, Book).
```

Binary Trees

- A binary tree can be recursively defined using node elements
  - Empty node represented as `null`
  - Other nodes as `node(Value, Left, Right)`
- Definition of a binary tree

```
binary_tree(null).
binary_tree( node(Value, Left, Right) ):-
        binary_tree(Left),
        binary_tree(Right).
```




```
node(3,  node(1, null, null),
          node(7,  node(5,null,null),
                   node(9,null,null)  )  ).
```

Binary Trees

- Tree operations are easily implemented from this definition
  - Check if value is a member of the tree

```
tree_member(Val, node(Val, _L, _R) ).
tree_member(Val, node(V, L, _R) ):-
       [Val < V,] tree_member(Val, L).
tree_member(Val, node(V, _L, R) ):-
       [Val > V,] tree_member(Val, R).
```

[code] if we consider the tree to be a binary search tree

  - List all tree elements (in-order traversal)

```
tree_list( null, [] ).
tree_list( node(Val, L, R), List ):-
       tree_list(L, Left),
       tree_list(R, Right),
       append(Left, [Val|Right], List).
```

Binary Trees

- Verify if tree is ordered

```
tree_is_ordered(Tree):-
        tree_list(Tree, List),
        sort(List, List).
```

- Insert an element into the tree

```
tree_insert( null, V, node(V, null, null) ).
tree_insert( node(V, L, R), V, node(V, L, R) ).
tree_insert( node(V, L, R), Val, node(V, NL, R) ):-
        Val < V, tree_insert( L, Val, NL).
tree_insert( node(V, L, R), Val, node(V, L, NR) ):-
        Val > V, tree_insert( R, Val, NR).
```

Binary Trees

- Determine the height of the tree

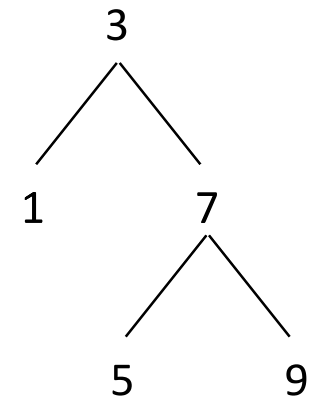
```
tree_height( null, 0).
tree_height( node(Val, L, R), H):-
        tree_height(L, HL),
        tree_height(R, HR),
        H is 1 + max(HL, HR).
```

- Check whether the tree is balanced

```
tree_is_balanced( null ).
tree_is_balanced( node(Val, L, R) ):-
        tree_is_balanced(L),
        tree_is_balanced(R),
        tree_height(L, HL),
        tree_height(R, HR),
        abs(HL-HR) =< 1.
```

Binary Trees

```
| ?- test_tree(_T), tree_member(5, _T).
yes
| ?- test_tree(_T), tree_member(4, _T).
no
| ?- test_tree(_T), tree_list(_T, L).
L = [1,3,5,7,9] ?
yes
| ?- test_tree(_T), tree_is_ordered(_T).
yes
| ?- test_tree(_T), tree_height(_T, H).
H = 3 ?
yes
| ?- test_tree(_T), tree_is_balanced(_T).
yes
| ?- test_tree(_T), tree_insert(_T, 2, NT).
NT = node(3,node(1,null,node(2,null,null)),node(7,node(5,null,null),node
(9,null,null))) ?
yes
| ?- test_tree(_T), tree_insert(_T, 6, NT), tree_is_balanced(NT).
no
```



Difference Lists

- While lists are widely used, some common operations may not be very efficient, as is the case of appending two lists
  - Linear on the size of the first list
- Idea: increase efficiency by 'also keeping a pointer to the end of the list'
  - This is accomplished by using difference lists
    - We can use any symbol to separate the two parts of the difference list
    - With this representation, we can have an incomplete list (when the second list is not instantiated)

```
X = [1, 2, 3]

X = [1, 2, 3, 4, 5, 6]\[4, 5, 6]
X = [1, 2, 3, a, b, c]\[a, b, c]
X = [1, 2, 3]\[]
X = [1, 2, 3 | T]\T
```

Difference Lists

- We can now append two (difference) lists in constant time
  - To append X\Y with Z\W, simply unify Y with Z

```
append_dl(X\Y, Y\W, X\W).
```

  - Note that the two lists must be compatible – the tail of the first list must either be uninstantiated or be equal to the second list

```
| ?- append_dl( [a, b, c | Y ]\Y, [d, e, f | W]\W, A).
Y=[d,e,f|W]
A=[a,b,c,d,e,f|W]\W
```

Incomplete Data Structures

- Implementation of a dictionary using lists

```
lookup(Key, [ Key-Value | Dic ], Value).
lookup(Key, [ K-V | Dic ], Value):-
      Key \= K,
      lookup(Key, Dic, Value).
```

  - When *Key* is present, *Value* is verified/returned
  - When *Key* is not present, the new *Key-Value* pair is added to the dictionary

```
| ?- Dic = [x-1, y-2, z-3 | _D], dilookup(y, Dic, V).
Dic = [x-1,y-2,z-3|_D],
V = 2 ?
yes
| ?- Dic = [x-1, y-2, z-3 | _D], dilookup(w, Dic, 4).
Dic = [x-1,y-2,z-3,w-4|_A] ?
yes
```

Incomplete Data Structures

- A dictionary implemented with a binary search tree

```
lookup(Key, dtnode(Key-Value, _L, _R), Value).
lookup(Key, dtnode(K-_V, L, _R), Value):-
      Key < K, lookup(Key, L, Value).
lookup(Key, dtnode(K-_V, _L, R), Value):-
      Key > K, lookup(Key, R, Value).
```

```
| ?- dtree(DTree).
DTree = dtnode(3-b,dtnode(1-(a),_A,_B),dtnode(7-d,dtnode(5-c,_C,_D),dtnode(9-(e),_E,_F))) ?
yes
| ?- dtree(DTree), lookup(5, DTree, V).
DTree = dtnode(3-b,dtnode(1-(a),_A,_B),dtnode(7-d,dtnode(5-c,_C,_D),dtnode(9-(e),_E,_F))),
V = c ?
yes
| ?- dtree(DTree), lookup(4, DTree, g).
DTree = dtnode(3-b,dtnode(1-(a),_A,_B),dtnode(7-d,dtnode(5-c,dtnode(4-g,_C,_D),_E),dtnode(9-(
e),_F,_G))) ?
yes
```

Q & A


I AM A WIZARD
I AM UNSTOPPABLE
When you're writing Prolog and it succeeds on the first try