Proving Properties of Functions

Exercise: Flatten/Raise Trees/Lists

# In this exercise sheet we will study code which flatten trees into lists, and which then takes the flattened version and turns back into a tree. We will prove that flattening a tree, and raising the resulting list, gives back the original tree. The code for this exercise can be found in the file TreeFlattenRaiseSequences.dfy.

# In this version of the exercise we are using sequences. An alternative formulation in terms of lists is also possible, and is given in the file TreeFlattenRaiseLists.dfy.

# 1. Question – flattening

Consider the definition of trees, and codes for flattening these trees.

datatype Tree<T> = Leaf(T) | Node(Tree<T>, Tree<T>);

datatype Code<T> = Lf(T) | Nd;

function flatten(t: Tree): seq<Code>

{

match t

case Leaf(x) =>

[Lf(x)]

case Node(t1, t2) =>

flatten(t1) + flatten(t2) + [Nd]

}

What is the result of the call  
 var fl := flatten(Node( Node( Leaf(“D”),Leaf(”a’)),

Node( Node(Lead(“f”), Leaf(“n”)),

Leaf(“y”))

Prove that flattening creates a sequence of the same length as the size of the original tree.

function size(t: Tree): nat

{

match t

case Leaf(x) =>

1

case Node(t1, t2) =>

1 + size(t1) + size(t2)

}

ghost method preservesSize (t: Tree)

ensures size(t) == |flatten(t)|;

# 2. Question – tail recursion fibonacci

Consider the definition of a function raise, which takes a flattened tree, and raises it to its original form.

We define the function raise so that it returns a sequence of trees, rather a single tree, because we need to cater for the case where its input does not correspond to a flattened version of a tree. In such a case, it will return an empty sequence. (An alternative apploach would encode optional types with explicit error values – this is how we tackled that issue in the version that uses lists.)

function raise(cds: seq<Code>, ts: seq<Tree>): seq<Tree>

{

if cds == [] then

ts

else

raise(cds[1..], raiseOne(cds[0], ts))

}

function raiseOne(cd: Code, ts: seq<Tree>): seq<Tree>

{

match cd

case Lf(x) =>

[Leaf(x)] + ts

case Nd =>

if |ts| < 2 then

[]

else

[Node(ts[1], ts[0])] + ts[2..]

}

What is the result of the call raise(fl)? And of the call raise(Lf(‘a’),Lb(‘b’))?

Now, prove that

ghost method Theorem(t: Tree)

ensures raise(flatten(t), []) == [t];

**Sample answers** in TreeFlattenRaiseSequences\_SA.dfy, and for the version which is in terms of lists, in file TreeFlattenRaiseLists\_SA.dfy