Reasoning About Programs

Week 4 Assessed PMT - Structural Induction
Answers to all questions to be submitted to the SAO by 4pm on Monday 5th Feb

Sophia Drossopoulou and Mark Wheelhouse*

Aims This exercise sheet is about structural induction on lists and trees, the correct set up of the proofs, and the correct use of quantifiers. It is also about the discovery if necessary auxiliary lemmas, and about finding the appropriate inductive principle given a data type definition.

In some questions you may find it helpful to use some of the the following Lemmas:

(A) $\forall$`xs:[a]`, `ys:[a]`. `length (xs++ys)` = `(length xs) + (length ys)`
(B) $\forall$`x:a`, `xs:[a]`. `ys:[a]`. `(x:xs)++ys` = `x:(xs++ys)`
(C) $\forall$`x:a`. `[x]` = `x:[]`
(D) $\forall$`xs:[a]`, `ys:[a]`, `zs:[a]`. `xs++(ys++zs)` = `(xs++ys)++zs`
(E) $\forall$`x:a`, `xs:[a]`. `[x]++xs` = `x : xs`

1st Question (1 point):

Consider the following definitions, where `enc` flattens a tree into a list of `Code`s:

```
data Code = Lf Int | Nd
data Tree = Node Tree Tree | Leaf Int

enc :: Tree -> [Code]
enc (Leaf i) = [ Lf i ]
enc (Node t1 t2) = (Nd:(enc t1))++(enc t2)
```

Write out the values of the following terms:

a) `enc (Node (Leaf 3) (Node (Leaf 5) (Leaf 7)))`

b) `enc (Node (Node (Leaf 3) (Leaf 5)) (Leaf 7))`

A possible answer:

a) `enc (Node (Leaf 3) (Node (Leaf 5) (Leaf 7)))` = `[Nd,(Lf 3),Nd,(Lf 5),(Lf 7)]`

b) `enc (Node (Node (Leaf 3) (Leaf 5)) (Leaf 7))` = `[Nd,Nd,(Lf 3),(Lf 5),(Lf 7)]`

*with thanks to Tristan Allwood and Alexander J. Summers

2nd Question (10 points):

The following functions count the number of elements in lists or trees

```
length :: [a] -> Int
length [] = 0
length (z:zs) = 1 + length zs

size :: Tree -> Int
size (Leaf i) = 1
size (Node t1 t2) = 1 + (size t1) + (size t2)
```

Prove that

$$\text{(i)} \quad \forall \texttt{t:Tree}.\ \texttt{length(enc t)} = \texttt{size t}$$

In your proof be sure to state what is given, what is to be shown and justify each step.

A possible answer:

We will prove $\forall$`length(enc t)` = `size t` by structural induction on `t`.

Base Case:

To show: $\forall$`i:Int`. `length (enc (Leaf i))` = `size (Leaf i)`

Take an arbitrary `i:Int`.

We want to show that `length (enc (Leaf i))` = `size (Leaf i)`.

We prove the above as follows:

`length (enc (Leaf i))`		
=	`length [(Lf i)]`	by definition of `enc`
=	`length ((Lf i):[])`	by property (C)
=	`1 + length []`	by definition of `length`
=	`1 + 0`	by definition of `length`
=	`1`	by arithmetic
=	`size (Leaf i)`	by definition of `size`

Inductive Step:

Take arbitrary trees `t1:Tree` and `t2:Tree`.

Inductive Hypothesis: `length (enc t1)` = `size t1` $\wedge$ `length (enc t2)` = `size t2`

To show: `length (enc (Node t1 t2))` = `size (Node t1 t2)`

We prove this as follows:

`length (enc (Node t1 t2))`		
=	`length ( ( Nd:(enc t1) )++(enc t2) )`	by definition of `enc`
=	`length ( Nd:( (enc t1)++(enc t2) ) )`	by property (B)
=	`1 + length( (enc t1)++(enc t2) )`	by definition of `length`
=	`1 + length (enc t1) + length (enc t2)`	by property (A)
=	`1 + (size t1) + (size t2)`	by inductive hypothesis
=	`size (Node t1 t2)`	by definition of `size`

3rd Question (1 point):

Consider now the function `dec` which re-creates a tree out of the flattened representation:

```
dec :: [Code] -> Tree
dec cds = t
       where
       (t, []) = decAux cds

decAux :: [Code] -> (Tree, [Code])
decAux  ((Lf i):cds) = ( Leaf i,  cds )
decAux  (Nd:cds)    = ( Node t1 t2, cds2)
      where
      (t1, cds1) = decAux cds
      (t2, cds2) = decAux cds1
```

Note that `dec` is a partial funciton and is therefore not defined for *all* possible inputs.

What is the value of

a) `decAux [(Lf 3),Nd,(Lf 5),(Lf 7)]`

b) `decAux [Nd,(Lf 3),(Lf 5),(Lf 7)]`

c) `dec [(Lf 3),Nd,(Lf 5),(Lf 7)]`

d) `dec [Nd,(Lf 3),(Lf 5)]`

A possible answer:

a) `decAux [(Lf 3),Nd,(Lf 5),(Lf 7)]` = `( (Leaf 3), [Nd,(Lf 5),(Lf 7)] )`

b) `decAux [Nd,(Lf 3),(Lf 5),(Lf 7)]` = `( (Node (Leaf 3) (Leaf 5)), [(Lf 7)] )`

c) `dec [(Lf 3),Nd,(Lf 5),(Lf 7)]` is undefined

d) `dec [Nd,(Lf 3),(Lf 5)]` = `Node (Leaf 3) (Leaf 5)`

4th Question (12 points)

We want to prove that the functions `enc` and `dec` are inverses, i.e that

$$\texttt{(ii)}\quad \forall\texttt{t:Tree}.\ \texttt{dec (enc t)} = \texttt{t}$$

Obviously, we need to prove that

$$\texttt{(iii)}\quad \forall\texttt{t:Tree}.\ \texttt{decAux (enc t)} = \texttt{(t,[])}$$

However, (`iii`) is too weak to be directly proven by induction. Therefore, we strengthen (`iii`) to describe more general properties of the function `decAux`. Therefore, we want to prove:

$$\texttt{(iv)}\quad \forall\texttt{t:Tree}.\ \forall\texttt{cds:[Code]}.\ \texttt{decAux ((enc t)++cds)} = \texttt{(t,cds)}$$

which implies (`iii`).

Prove (`iv`) by structural induction. State what is to be shown and justify each step.

Remarks:

- R1 You need to take some care with the definition and application of the induction hypothesis.
- R2 Assertion (`iv`) clarifies how the program works: `decAux` splits its input into the part that is a an encoded tree and the remainder; it then returns the pair consisting of the tree and the remaining input.

A possible answer:

We prove ∀`t:Tree`.∀`cds:[Code]`. `decAux ((enc t)++cds)` = `(t,cds)` by str. ind. on `t`.

Base Case:

To show: ∀`i:Int`, `cds:[Code]`. `decAux ((enc (Leaf i))++cds )` = `(Leaf i,cds)`

Take arbitrary `i:Int`, `cds:[Code]`.

We want to show that `decAux ((enc (Leaf i))++cds )` = `(Leaf i,cds)`.

We prove the above as follows:

```
decAux ((enc (Leaf i))++cds )
 =  decAux ( [Lf i]++cds )   by definition of enc
 =  decAux ( [Lf i]:cds )    by Lemma (E)
 =  (Leaf i, cds )           by definition of decAux
```

Inductive Step:

Take arbitrary trees `t1` and `t2`.
Inductive Hypothesis:

∀`cds1:[Code]`. `decAux ( (enc t1)++cds1 )` = `(t1,cds1)`
∧
∀`cds2:[Code]`. `decAux ( (enc t2)++cds2 )` = `(t2,cds2)`

To show:

∀`cds:[Code]`. `decAux (enc (Node t1 t2)++cds)` = `(Node t1 t2, cds)`

We prove the above as follows:

Take arbitrary `cds:[Code]`. We now want to show:
`decAux (enc (Node t1 t2)++cds)` = `(Node t1 t2, cds)`

By application of the induction hypothesis, by instantiating the universally quantified `cds1` by `(enc t2)++cds`, we obtain:

(∗) `decAux ((enc t1)++((enc t2)++cds))` = `(t1, (enc t2)++cds)`

Again, by application of the induction hypothesis, and by instantiating the universally quantified `cds2` by `cds`, we obtain:

(∗∗) `decAux ((enc t2)++cds )` = `(t2,cds)`

We now finish the proof:

```
decAux ((enc (Node t1 t2))++cds)
 =  decAux ((Nd:(enc t1)++(enc t2))++cds)      by definition of enc
 =  decAux (Nd:((enc t1)++(enc t2)++cds))      by Lemmas (D) and (B)
 =  (Node t1 t2, cds)                          by definition of decAux, (*) and (**).
```

5th Question (3 points):

Now consider a different pair of encode/decode functions:

```
encd :: Tree -> [Code]
encd (Leaf i) = [ Lf i ]
encd (Node t1 t2) =  (encd t1)++(encd t2)++ [Nd]

decd :: [Code] -> Tree
decd cds = decdAux cds []

decdAux :: [Code] -> [Tree] -> Tree
decdAux [] t:ts  =  t
decdAux ((Lf i):cds) ts  =  decdAux cds  ((Leaf i):ts)
decdAux (Nd:cds) (t1:t2:ts)  =  decdAux cds  ((Node t2 t1):ts)
```

Write out the values of the following terms:

a) `encd (Node (Leaf 3) (Node (Leaf 5) (Leaf 7)))`

b) `encd (Node (Node (Leaf 3) (Leaf 5)) (Leaf 7))`

c) `decdAux [(Lf 9)] [(Node (Leaf 5) (Leaf 7)),(Leaf 3)]`

d) `decdAux [(Lf 5),(Lf 7),Nd,(Lf 9)] [(Leaf 3)]`

A possible answer:

a) `encd (Node (Leaf 3) (Node (Leaf 5) (Leaf 7)))` = `[(Lf 3),(Lf 5),(Lf 7),Nd,Nd]`

b) `encd (Node (Node (Leaf 3) (Leaf 5)) (Leaf 7))` = `[(Lf 3),(Lf 5),Nd,(Lf 7),Nd]`

c) `decdAux [(Lf 9)] [(Node (Leaf 5) (Leaf 7)),(Leaf 3)]` = `(Leaf 9)`

d) `decdAux [(Lf 5),(Lf 7),Nd,(Lf 9)] [(Leaf 3)]` = `(Leaf 9)`

6th Question (7 points) - CHALLENGE:

We want to prove that the functions `encd` and `decd` are inverses, i.e that

(v) $\forall$`t:Tree. decd (encd t)` = `t`

As in the previous question, proving (`v`) requires proving another assertion, namely:

$$(\texttt{vi}) \quad \forall \texttt{t:Tree}.\ \texttt{decdAux (encd t) []} = \texttt{t}$$

This assertion is too weak to be directly proven by induction. Write, *but do not prove*, a stronger assertion (`vii`) which *can* be proven by induction. Outline how (`vii`) implies (`vi`).

Hint: The following thoughts may help to find (`vii`). Notice that the function `decdAux` returns only if its first argument is an empty list, otherwise it calls itself. Therefore, we can prove (`vi`) if we have (`viii`) $\forall$`t:Tree`. `decdAux (encd t) []` = `decdAux [] t:[]`
However, (`viii`) is too weak, and cannot be proven by induction. As we saw in the Week 4 PMT Question 2, and in the lectures, we need to generalize such assertions, so that they talk about many more input values to the function `decdAux`, i.e. generalize the empty lists which appear in (`viii`).

A possible answer:

The following property *can* be proven by structural induction on `t`:

$$(\texttt{vii}) \quad \begin{array}{l} \forall \texttt{t:Tree}.\ \forall \texttt{cds:[Code]}.\forall \texttt{ts:[Tree]}. \\ \texttt{decdAux ((encd t)++cds) ts} = \texttt{decdAux cds (t:ts)} \end{array}$$

Moreover, (`vii`) implies assertion (`vi`) by the following argument: Assume that (`vii`). Take `cds:[Code]` to be `[]`, and `ts` to be `[]`, and obtain:
(`viii`) `decdAux (encd t) [] = decdAux [] [t]`
By application of the definition of `decdAux` on the hand side of (`viii`), we obtain:
`decdAux (encd t) [] = t`
The above is the same as (`vi`).

Note: Assertion (`vii`) clarifies how the program works: `decdAux` uses its first input parameter to read the codes, and its second input parameter as an accumulator for the trees it has decoded so far.

7th Question (6 points):

Consider the following data type `Map a b`, and assume a property $P \subseteq$ `Map a b`.

```
data Map a b = None | Last a | Comb b (Map a b)
```

Write the structural induction principle which can demonstrate the validity of $\forall$`m:Map a b`. P(`m`).

A possible answer:

$$\begin{array}{c} P(\texttt{None}) \ \wedge\ \forall \texttt{x : a. P(Last x)} \ \wedge\ \forall \texttt{x : b.}\ \forall \texttt{y : Map a b.}\,[\ \texttt{P(y)} \rightarrow \texttt{P(Comb x y)}\ ] \\ \rightarrow \\ \forall \texttt{m : Map a b. P(m)} \end{array}$$

Marking Scheme

UTAs have discretion in awarding marks. The following marking scheme is representative of how we would mark the exams. In the following, we refer to the inductive hypothesis as the IH, and to the assertion that is to be proven as the TO-SHOW. Obviously, the TO-SHOW differs in the base case and the inductive step.

1st Question: 1 point :

2nd Question: 10 points

Setting up Base Case 2 points

1 pt for dealing correctly with `i`, i.e. either universally quantified in the TO-SHOW, or taken arbitrarily before the TO-SHOW. **1 pt** for the rest of what is to be shown

Proving Base Case 2 points

1 pt for correct steps, **1 pt** for justifications of steps.

Setting up Inductive Step 3 points

1 pt for taking 2 different, arbitrary trees before the IH. **1 pt** for having an IH which is the conjunction of two assertions. **1 pt** for the TO-SHOW

Proving Inductive Step 3 points

1 pt for correct application of IH, **1 pt** for correct steps, **1 pt** for justifications

3rd Question: 1 point

4th Question: 12 points

Setting up Base Case: 2 points

1 pt for dealing correctly with `i`, similar considerations as in 2ns Question. **1 pt** for the rest of what is to be shown

Proving Base Case: 2 points

1 pt for correct steps, **1 pt** for justifications of steps.

Setting up Inductive Step: 5 points

1 pt for taking 2 different, arbitrary trees before the IH. **1 pt** for universally quantified `cds1`, and `cds2` in the IH. **1 pt** for having an IH which is the conjunction of two assertions. **1 pt** for the TO-SHOW having another universally quantified `cds`. **1 pt** for the rest.

Proving Inductive Step: 3 points

2 pt for correct application of IH (twice), **1 pt** for the rest.

5th Question: 3 points

6th Question: 7 points

4 pt for coming up with a (`vii`) that can be proven by induction. **3 pt** for showing that (`vii`) implies (`vi`).

7th Question: 6 points **1 pt** for the first conjunct, **2 pt** for second conjunct, **3 pt** the third conjunct.

Comparison of the sheet's questions with possible exam questions

Questions 1, 3, and 5 are more straightforward, and they aim to help you think about Question 2, 4, and 6 respectively. Such questions may appear at the beginning of an exam question. Question 2 is representative of half or a third of an exam question. Question 4 is representative of a more challenging half exam question. Question 6 is an interesting challenge question, a question like it might appear as a very advanced part of an exam paper. Question 7 could appear as part of an exam paper.