Reasoning About Programs

Week 4 PMT - Structural Induction
To discuss during PMT - do NOT hand in

Sophia Drossopoulou and Mark Wheelhouse

Aims: To practice developing and applying the structural induction principle for various data structures. To set up proofs and make proof steps explicit. To be aware that some proofs do not require induction. Correct use of universal quantifiers.

1st Question:

Consider the function `rev` defined on lists as follows:

```
rev :: [a] -> [a]
rev [] = []
rev (x:xs) = (rev xs) ++ [x]
```

Consider the following two assertions:

RevConcat $\forall$`xs:[a].` $\forall$`x:a.` `rev (xs ++[x])` $=$ `x:(rev xs)`

RevRev $\forall$`xs:[a].` `rev(rev(xs))` $=$ `xs`

and

(a) Write our the structural induction principle for **RevConcat**

(b) Write our the structural induction principle for **RevRev**

(c) Prove **RevConcat**

(d) Prove **RevRev**

In your proofs be sure to state what is given, what is to be shown and justify each step. Note that part (a) can be proven without induction. You can assume the following Lemmas about lists. For all elements `x` and lists `xs` and `ys`:

(A)	`xs ++ []` $=$ `xs` $=$ `[] ++ xs`
(B)	`(xs ++ ys) ++ zs` $=$ `xs ++ (ys ++ zs)`
(C)	`[x] ++ xs` $=$ `x:xs`
(D)	`rev (xs ++ ys)` $=$ `(rev ys) ++ (rev xs)`
(E)	`rev [x]` $=$ `[x]`

A possible answer:

(a)

$$
\begin{array}{l}
\forall \texttt{x:a}.\ \texttt{rev ([] ++ [x])} = \texttt{x:(rev [])} \\
\qquad\qquad \wedge \\
\forall \texttt{zs:[a]}.\ \forall \texttt{z:a}.[\ \forall \texttt{x:a}.\ \texttt{rev (zs++ [x])} = \texttt{x:(rev zs)} \ \rightarrow \\
\qquad\qquad\qquad\quad \forall \texttt{x:a}.\ \texttt{rev (z:zs++ [x])} = \texttt{x:(rev z:zs)}\ ] \\
\qquad\qquad \longrightarrow \\
\forall \texttt{xs:[a]}.\ \forall \texttt{x:a}.\ \texttt{rev (xs ++ [x])} = \texttt{x:(rev xs)}
\end{array}
$$

(b)

$$
\begin{array}{l}
\texttt{rev (rev [])} = \texttt{[]} \\
\qquad\qquad \wedge \\
\forall \texttt{ys:[a]}.\ \forall \texttt{y:a}.[\ \texttt{rev (rev ys)} = \texttt{ys} \ \rightarrow \\
\qquad\qquad\qquad\quad \texttt{rev (rev y:ys)} = \texttt{y:ys}\ ] \\
\qquad\qquad \longrightarrow \\
\forall \texttt{xs:[a]}.\ \texttt{rev (rev xs)} = \texttt{xs}
\end{array}
$$

(c) **To show:** $\forall$`xs:[a]`. $\forall$`x:a`. `rev (xs ++ [x])` = `x:(rev xs)`

Take arbitrary `xs:[a]` and `x:a`. Then:

`rev (xs ++ [x])`	=	`(rev [x]) ++ (rev xs)`	by (D)
	=	`[x] ++ (rev xs)`	by (E)
	=	`x:(rev xs)`	by (C)

Note: this proof did not *require induction.*

(d) **To show:** $\forall$`xs:[a]`. `rev (rev xs)` = `xs`

Let P(`xs`) be `rev (rev xs)` = `xs`. We will prove $\forall$`xs:[a]`. P(`xs`) by structural induction.

Base Case:

To show: `rev(rev([])) = []`

`rev(rev ([]))`	=	`rev []`	by def. of `rev`
	=	`[]`	by def. of `rev`

Inductive Step:

Take arbitrary `ys:[a]`, and `y:a`.
Inductive Hypothesis: `rev(rev(ys)) = ys`
To show: `rev(rev(y:ys)) = y:ys`

Then:

`rev(rev(y:ys))`	=	`rev (rev(ys) ++ [y])`	by def. of `rev`
	=	`(rev [y]) ++ (rev(rev(ys)))`	by (D)
	=	`(rev [y]) ++ ys`	by Ind.Hyp
	=	`[y] ++ ys`	by E
	=	`y:ys`	by C

2nd Question:

Consider the function `revA` defined on lists as follows:

```
revA :: [a] -> [a] -> [a]
revA [] ys = ys
revA (x:xs) ys =  revA xs (x:ys)
```

Observe that `revA (1:2:3:[]) []` = `3:2:1:[]`. More generally, we want to prove that:

$$(*) \quad \forall \texttt{xs:[a]}.\ \texttt{revA xs []} = \texttt{rev xs}$$

However, if we attempt to prove $(*)$ by structural induction on `xs`, we will get stuck when we try to prove the inductive step. Instead, we need to find a stronger lemma $(**)$, which implies $(*)$, and which *can* be proven by induction.

Write such a $(**)$ which implies $(*)$, and can be proven by structural induction.

A possible answer:

$$(**) \quad \forall \texttt{xs:[a]}.\ \forall \texttt{ys:[a]}.\ \texttt{revA xs ys} = (\texttt{rev xs}){+}{+}\texttt{ys}$$

Note The difference between (*) and (**) is that the second argument of `revA` in (*) is a constant (the empty list `[]`), while in (**) it is universally quantified (...$\forall$`ys:[a]. revA ...  ys` = ...)

3rd Question:

Consider the following definition of a datatype `DT`:

```
data DT = CA [Int] | CB DT Int
```

a Write the structural induction principle for `DT` for a predicate $P \subseteq$ `DT`.

b Write the proof schema that you would employ to prove $\forall$`dt`. $P($`dt`$)$ by induction over `dt`.

A possible answer:

a

$$\forall \texttt{is:[Int]}.\ P(\texttt{CA is}) \ \wedge\ \forall \texttt{dt:DT}.\forall \texttt{i:Int}.[\ P(\texttt{dt}) \to P(\texttt{CB dt i})\ ] \quad \longrightarrow \quad \forall \texttt{dt:DT}.\ P(\texttt{dt})$$

b

Base Case: To show: $\forall$`is:[Int]`. $P($`CA is`$)$

Take `is:[Int]`, arbitrary. To show $P($`CA is`$)$.

a proof here

...

Inductive Step:

Take arbitrary dt:[DT], and i:Int.
Inductive Hypothesis: $P(\texttt{dt})$
To show: $P(\texttt{CB dt i})$

another proof here
...

4th Question (this week's challenge):

Consider the functions guess4 and guess5 from a Logic tutorial in the first term, defined as follows:

```
guess4 :: [Char] -> Bool
guess4 [] = true
guess4 xs = guess5 xs []

guess5 :: [Char] -> [Char] -> Bool
guess5 [] zs                    = true
guess5 (x:xs) zs | (x:xs)==zs  = true
                 |  xs==zs     = true
                 | otherwise   = guess5 xs (x:zs)
```

We define the predicate $Pal \subseteq [\texttt{Char}]$, which expresses that its argument is a palindrome, as follows:

$$Pal(\texttt{cs}) \equiv \exists \texttt{cs'} : [\texttt{Char}], \texttt{c} : \texttt{Char}.\ [\ \texttt{cs} = \texttt{cs'}\texttt{++}(\texttt{rev}\,\texttt{cs'}) \ \vee \texttt{cs} = \texttt{cs'}\texttt{++}[\texttt{c}]\texttt{++}(\texttt{rev cs'})\ ]$$

Prove that

$$(*)\ \ \forall \texttt{cs} : [\texttt{Char}].\ [\ \texttt{guess4 cs} \ \longleftrightarrow\ Pal(\texttt{cs})\ ]$$

In the above, we use the notation
$\texttt{guess4 cs} \ \longleftrightarrow\ some\_Predicate$
as a shorthand for
$\texttt{guess4 cs} = \texttt{true} \ \longleftrightarrow\ some\_Predicate$

You can assume the following Lemmas about lists. For all elements x and lists xs and ys:

(A)	$\texttt{xs ++ []} = \texttt{xs} = \texttt{[] ++ xs}$
(B)	$\texttt{(xs ++ ys) ++ zs} = \texttt{xs ++ (ys ++ zs)}$
(C)	$\texttt{[x] ++ xs} = \texttt{x:xs}$
(D)	$\texttt{rev (xs ++ ys)} = \texttt{(rev ys) ++ (rev xs)}$
(E)	$\texttt{rev [x]} = \texttt{[x]}$
(F)	$\texttt{rev []} = \texttt{[]}$
(G)	$\texttt{rev (x:xs)} = \texttt{(rev xs)++x}$
(H)	$\texttt{rev (rev xs) = xs}$
(K)	$\texttt{x:xs} = \texttt{[x]++xs}$
(L)	$\lvert\texttt{x:xs}\rvert = 1 + \lvert\texttt{xs}\rvert$

A possible answer:

To prove (*) we need some properties of `guess5`. Therefore, we make the following assumption.

Assumption1: $\forall$c : Char.$\forall$cs : [Char]. [guess5 (c:cs) [] $\longleftrightarrow$ Pal(c:cs)]

Proof We will prove **Assumption1** through **Lemma C** which will be stated and proven below.

Lemma A: Pal([])

Proof: From the properties of lists from earlier, we obtain:

[]	=	[]++[]	by (A)
	=	[]++(rev [])	by (F)

Therefore,

$\exists$ cs1:[Char].[[] = cs1++(rev cs1)].

Therefore,

Pal([])

Lemma B: $\forall$cs : [Char]. [guess4 cs $\longleftrightarrow$ Pal(cs)]

This is the proof of (*). We will be using **Assumption1**.

Proof: We take arbitrary `cs` from `[Char]`. We proceed by case analysis on `cs`.

First Case cs=[].

guess4 cs	$\leftrightarrow$	true	by definition of guess5
	$\leftrightarrow$	Pal([])	by **Lemma A**

Second Case There exist c:Char, cs':[Char], such that cs=c:cs'.

guess4 cs	$\leftrightarrow$	guess4 (c:cs')	by case
	$\leftrightarrow$	guess5 (c:cs') []	by definition of guess4
	$\leftrightarrow$	Pal(c : cs')	by **Assummption1**
	$\leftrightarrow$	Pal(cs)	by case

Lemma C

$\forall$cs,ds:[Char]. [guess5 cs ds $\longleftrightarrow$ |cs| $\geq$ |ds| $\wedge$ Pal((rev ds)++cs) $\wedge$ cs $\neq$ []]

Proof We prove **Lemma C** by induction on `cs`:

Base Case:

To show:

$\forall$ds : [Char]. [guess5 [] ds $\longleftrightarrow$ |[]| $\geq$ |ds| $\wedge$ Pal((rev ds)++[]) $\wedge$ [] $\neq$ []]

Both sides of the $\longleftrightarrow$ are false, therefore the base case is proven.

Inductive Step:

Take arbitrary cs:[Char], and c:Char.

Inductive Hypothesis:

$\forall$ds:[Char]. [guess5 cs ds $\longleftrightarrow$ |cs| $\geq$ |ds| $\wedge$ Pal((rev ds)++cs) $\wedge$ cs $\neq$ []]

To show:

$\forall$`ds:[Char]`. [`guess5 c:cs ds` $\longleftrightarrow$ $|$`c:cs`$| \geq |$`ds`$|$ $\wedge$ Pal((`rev ds`)++(`c:cs`))
$\wedge$ `c:cs` $\neq$ `[]`]

We proceed by case analysis on `cs`.

1st Case: `cs=[]`.

`guess5 c:cs ds` $\longleftrightarrow$ `guess5 c:[] ds`
by case
$\longleftrightarrow$ `ds` $=$ `[]` $\vee$ `ds` $=$ `c:[]`
by def. `guess5`
$\longleftrightarrow$ $|$`c:[]`$| \geq |$`ds`$|$ $\wedge$ Pal((`rev ds`)++(`c:[]`)) $\wedge$ `c:[]` $\neq$ `[]`
by def Pal, and also, because $|$`c:cs`$| \geq |$`ds`$|$ and also `c:cs` $\neq$ `[]`
$\longleftrightarrow$ $|$`c:cs`$| \geq |$`ds`$|$ $\wedge$ Pal((`rev ds`)++(`c:cs`)) $\wedge$ `c:cs` $\neq$ `[]`
by case

2nd Case: `c:cs` $=$ `ds` $\vee$ `cs` $=$ `ds`

`guess5 c:cs ds` $\longleftrightarrow$ `true`
by case, and def. `guess5`
$\longleftrightarrow$ $|$`c:cs`$| \geq |$`ds`$|$ $\wedge$ Pal((`rev ds`)++(`c:cs`)) $\wedge$ `c:cs` $\neq$ `[]`
by case, and def Pal

3rd Case: `c:cs` $\neq$ `ds` $\wedge$ `cs` $\neq$ `ds`

`guess5 c:cs ds` $\longleftrightarrow$ `guess5 cs c:ds`
by case, and def. `guess5`
$\longleftrightarrow$ $|$`cs`$| \geq |$`c:ds`$|$ $\wedge$ Pal((`rev c:ds`)++`cs`) $\wedge$ `cs` $\neq$ `[]`
by Ind Hypo
$\longleftrightarrow$ $|$`c:cs`$| \geq |$`ds`$|$ $\wedge$ Pal(`rev ds`++(`c:cs`)) $\wedge$ `c:cs` $\neq$ `[]`
by (L), (G), (B) and (C)

Proof of Assumption1

In **Lemma C** we replace `cs` by `c:cs'`, and `ds` by `[]`, we obtain:

(***) $\forall$`c:Char`.$\forall$`cs':[Char]`.
[`guess5 c:cs' []` $\longleftrightarrow$ $|$`c:cs'`$| \geq |$`[]`$|$ $\wedge$ Pal((`rev []`)++(`c:cs'`)) $\wedge$ `c:cs'` $\neq$ `[]`]

Moreover, for all `c:Char`, and `cs':[Char]`, we have $|$`c:cs'`$| \geq 1 > 0 = |$`[]`$|$, and `c:cs'` $\neq$ `[]`. Also, from (G) and (A) we obtain that (`rev []`)++(`c:cs'`) $=$ `c:cs'`. Therefore, (***) gives us

(**Assumption1**) $\forall$`c:Char`.$\forall$`cs':[Char]`. [`guess5 c:cs' []` $\longleftrightarrow$ Pal(`c:cs'`)]

Discussion There are several different ways to solve this exercise. The challenge is the proof of **Assumption1**, which requires a stronger lemma to be found; the stronger lemma should talk about the execution of `guess5 cs ds` rather than the execution of `guess5 cs []`.

There are many such stronger lemmas, and **Lemma C** is one of them. A way to discover the lemma is as follows. We notice that `guess5` rotates the contents of its first argument onto its second argument. If the two become equal then it reports `true`, and otherwise it keeps rotating until the first argument becomes empty, and in such a case it reports `false`.

Thus, we have that 1) If $|$`cs`$| < |$`ds`$|$, then `guess5 cs ds` evaluates to `false`. Also, 2) when $|$`cs`$| = |$`ds`$|$, or $|$`cs`$| = 1 + |$`ds`$|$, then `guess5 cs ds` $\longleftrightarrow$ Pal(`ds++cs`). Finally, 3) if $|$`cs2`$| \geq |$`cs1++ds`$|$, then `guess5 (cs1++cs2) ds=guess5 cs2 (rev cs1)++ds`.