Reasoning About Programs

Week 4 Tutorial - Structural Induction

Sophia Drossopoulou and Mark Wheelhouse

Aims practice application of structural induction over data types, setting up the inductive principle given a Haskell data structure, and the discovery and use of auxiliary lemmas.

1st Question:

Consider the following data definition

```
data Nat = Zero | Succ Nat
```

(a) Assume a property $P \subseteq \texttt{Nat}$ and another property $Q \subseteq \texttt{Nat} \times \texttt{Nat}$.

i. Write out the structural induction principle over $n : \texttt{Nat}$ as applied to P.

ii. Write the proof schema for proving $\forall \texttt{m:Nat}.P(\texttt{m})$ by structural induction on `m`.

iii. Write the proof schema for proving $\forall \texttt{m:Nat}.\forall \texttt{n:Nat}.Q(\texttt{m}, \texttt{n})$ by structural induction on `m`.

iv. Write the proof schema for proving $\forall \texttt{m:Nat}.\forall \texttt{n:Nat}.Q(\texttt{m}, \texttt{n})$ by structural induction on `n`.

A possible answer:

i.

$$P(\texttt{Zero}) \ \land \ \forall \texttt{m:Nat}.[\ P(\texttt{m}) \rightarrow P(\texttt{Succ m})\ ] \quad \longrightarrow \quad \forall \texttt{m:Nat}.P(\texttt{m})$$

ii. **Proof** by structural induction on `m`.

Base case:

To Show: $P(\texttt{Zero})$

proof of base case

Inductive Step:

Take an arbitrary `k:Nat`.
Inductive Hypothesis: $P(\texttt{k})$
To show: $P(\texttt{Succ k})$

proof of inductive step

Proof by structural induction on `m`.

iii. **Base case:**

To Show: $\forall \texttt{n:Nat}.Q(\texttt{Zero}, \texttt{n})$

proof of base case

Inductive Step:

Take an arbitrary `k:Nat`.
Inductive Hypothesis: $\forall$`n:Nat`.Q(`k`, `n`)
To show: $\forall$`n:Nat`.Q(`Succ k`, `n`)

proof of inductive step

iv. **Proof**

We notice that $\forall$`m:Nat`.$\forall$`n:Nat`.Q(`m`, `n`) $\leftrightarrow$ $\forall$`n:Nat`.$\forall$`m:Nat`.Q(`m`, `n`).

We shall therefore prove $\forall$`n:Nat`.$\forall$`m:Nat`.Q(`m`, `n`) by structural induction on `n`.

Base case:

To Show: $\forall$`m:Nat`.Q(`m`, `Zero`)

proof of base case

Inductive Step:

Take an arbitrary `k:Nat`.
Inductive Hypothesis: $\forall$`m:Nat`.Q(`m`, `k`)
To show: $\forall$`m:Nat`.Q(`m`, `Succ k`)

proof of inductive step

(b) Consider the following function definition:

```
add :: Nat -> Nat -> Nat
add Zero j  =  j
add (Succ i) j  =  Succ (add i j)
```

Using structural induction on `m`, prove the following proposition:

(*) $\forall$`m,n:Nat`. `add m n` = `add n m`

Note: you will need to use some auxiliary lemmas. Formulate and prove these lemmas too.

A possible answer:

We start the proof of the proposition. As we proceed, we will notice that the proof requires auxiliary properties in order to proceed. We formulate these properties as lemmas and prove them below.

Proposition:

$\forall$`m,n:Nat`. `add m n` = `add n m`

Proof: Let P(`m`) be $\forall$`n:Nat`. `add m n` = `add n m`. We will prove $\forall$`m:Nat`. P(`m`) by structural induction on `m`.

Base case:

To Show: $\forall$`n:Nat.` `add Zero n` $=$ `add n Zero`

Follows from Lemma 1, which is stated and proven below.

Inductive Step:

Take an arbitrary `k:Nat`.
Inductive Hypothesis: $\forall$`n:Nat.` `add k n` $=$ `add n k`
To show: $\forall$`n:Nat.` `add (Succ k) n` $=$ `add n (Succ k)`

Take an arbitrary `n:Nat`.

```
add (Succ k) n
 =  Succ (add k n)    by function def of add
 =  Succ (add n k)    by ind. hypothesis
 =  add n (Succ k)    by Lemma 2, stated and proven below
```

We now state and prove the two auxiliary lemmas.

Lemma 1:

$\forall$`n:Nat.` `add Zero n` $=$ `add n Zero`

Proof: Let P(`n`) be `add Zero n` $=$ `add n Zero`. We will prove that $\forall$`n:Nat.` P(`n`) by structural induction on `n`:

Base case:

To show: `add Zero Zero` $=$ `add Zero Zero`

Trivial

Inductive step:

Take an arbitrary `k:Nat`.
Inductive Hypothesis: `add Zero k` $=$ `add k Zero`
To show: `add Zero (Succ k)` $=$ `add (Succ k) Zero`

```
add Zero (Succ k)
 =  Succ k               by function def of add
 =  Succ (add Zero k)    by function def of add
 =  Succ (add k Zero)    by ind. hypothesis
 =  add (Succ k) Zero    by function def of add
```

Lemma 2:

$\forall$`m,n:Nat.` `add m (Succ n)` $=$ `Succ (add m n)`

Proof: Let P(`m`) be $\forall$`n:Nat.` `add m (Succ n)` $=$ `Succ (add m n)`. We will prove $\forall$`m:Nat.` P(`m`) by structural induction on `m`.

Base case:

To show: $\forall$`n: Nat. add Zero (Succ n)` = `Succ (add Zero n)`

Take an arbitrary `n:Nat`.

```
add Zero (Succ n)
 =  Succ n              by function def of add
 =  Succ (add Zero n)   by function def of add
```

Inductive step:

Take an arbitrary `k:Nat`.
Inductive Hypothesis: $\forall$`n:Nat. add k (Succ n)` = `Succ (add k n)`
To show: $\forall$`n:Nat. add (Succ k) (Succ n)` = `Succ (add (Succ k) n)`

Take an arbitrary `n:Nat`.

```
add (Succ k) (Succ n)
 =  Succ (add k (Succ n))   by function def of add
 =  Succ (Succ (add k n))   by Induction Hupothesis
 =  Succ (add (Succ k) n)   by def. of add
```

2nd Question - from exam paper in 2015:

(a) Consider the datatype `Term` defined as follows:

```
data Term = Val Int | UMinus Term | Mult Term Term
```

Write the inductive principe which implies $\forall$ `t:Term`. $P($`t`$)$, for some property $P \subseteq$ `Term`.

A possible answer:

$\forall$`i: Int.` $P($`Val i`$)$
  $\wedge$
$\forall$`t: Term.` $[\,P($`t`$) \rightarrow P($`UMinus t`$)\,]$
  $\wedge$
$\forall$`t1, t2: Term.` $[\,P($`t1`$) \wedge P($`t2`$) \rightarrow P($`Mult t1 t2`$)\,]$
  $\rightarrow$
$\forall$`t: Term.` $P($`t`$)$

(b) Now consider the following functions:

```
eval :: Term -> Int
eval (Val i) = i
eval (UMinus t) = - (eval t)
eval (Mult t1 t2) = (eval t1) * (eval t2)
```

```
rip :: Term -> Term
rip (Val i) = (Val i)
rip (UMinus t) = rip t
rip (Mult t1 t2) =  Mult (rip t1) (rip t2)

pos :: Term -> Bool
pos (Val i) = True
pos (UMinus t) = not (pos t)
pos (Mult t1 t2) = ( (pos t1)==(pos t2) )

wSign ::  Int -> Bool  -> Int
wSign  i True  = i
wSign  i False = -i
```

Take `trm` to stand for
`UMinus (Mult (UMinus (Val -3)) (Val 2))`.
Write out the the result of evaluating the following expressions (but do *not* write any of the intermediate steps):

i. `eval trm`
ii. `rip trm`
iii. `pos trm`
iv. `eval (rip trm)`
v. `wSign (eval (rip trm)) (pos trm)`

A possible answer:

i. `eval trm` = `-6`
ii. `rip trm` = `t Mult (Val -3) (Val 2)`
iii. `pos trm` = `True`
iv. `eval (rip  trm)` = `-6`
v. `wSign (eval (rip trm)) (pos trm)` = `-6`

(c) Prove the following property
∀ `t:Term`. `eval t` = `wSign (eval (rip t)) (pos t)` .
In the proof, state what is given, what is to be shown, what is assumed, which variables are taken arbitrarily, and justify each step.

Prove the base case as normal. In the inductive step, consider *only* the case where `t` has the form `Mult t1 t2`.

You may use the facts that

A ∀ i,j:Int. `i*(-j)` = `-(i*j)`

B ∀ `i`:Int. ∀ `b`:Bool . `wSign i (not b)` = `- (wSign i b)`

C ∀ i1,i2:Int. ∀ b1,b2:Bool.
`wSig n (i1*i2) (b1==b2)` = `(wSign i1 b1)*(wSign i2 b2)`

A possible answer:

Proof by structural induction on `t`.

Base Case Take an `i:Int`, arbitrary.

To Show `eval (Val i)` $=$ `wSign (eval (rip (Val i))) (pos (Val i))`

This can be shown as follows:

`wSign (eval (rip (Val i)))) (pos (Val i))`

$=$	`wSign (eval (Val i)) True`	by def of `rip` and `pos`
$=$	`eval (Val i)`	by def of `wSign`

Inductive Step - Uminus: **NOT REQUIRED – START**

Take `t1:Term`, arbitrary.

Inductive Hypothesis `eval t1` $=$ `wSign (eval (rip t1)) (pos t1)`

To Show `eval (UMinus t1)` $=$ `wSign (eval (rip (UMinus t1))) (pos (UMinus t1))`

This can be shown as follows:

`wSign (eval (rip (UMinus t1))) (pos (UMinus t1))`

$=$	`wSign (eval (rip t1)) (not (pos t1))`	by def of `rip` and `pos`
$=$	$-$(`wSign (eval (rip t1)) (pos t1)`)	by (B)
$=$	$-$(`eval t1`)	by Induction Hypothesis
$=$	`eval (UMinus t1)`	by def of `eval`

NOT REQUIRED – END

Inductive Step - Mult

Take `t1`, `t2:Term`, arbitrary.

Inductive Hypothesis `eval t1` $=$ `wSign (eval (rip t1)) (pos t1)`

$\wedge$

`eval t2` $=$ `wSign (eval (rip t2)) (pos t2)`

To Show

`eval (Mult t1 t2)` $=$ `wSign (eval (rip (Mult t1 t2))) (pos (Mult t1 t2))`

This can be shown as follows:

`wSign (eval (rip (Mult t1 t2))) (pos (Mult t1 t2))`

```
=  wSign (eval (Mult (rip  t1) (rip t2)))  ((pos t1)==(pos t2))
                    by def of rip and pos
=  wSign ((eval (rip  t1))*(eval (rip  t2)))   ((pos t1)==(pos t2))
                    by def of eval
=  (wSign (eval (rip t1))  (pos t1)) * (wSign (eval (rip t2))  (pos t2))
                    by (C)
=  (eval t1) * (eval t1)
                    by Induction Hypothesis
=  eval (Mult  t1  t2)
                    by def of eval
```

Thanks to K. Broda, J. Bradley, F. Toni, I. Hodkinson, X. Fend and A. J. Summers.