Structural Induction: Tutorial sheet 3

Jeremy Bradley

23 January 2005

1. Prove the following by induction on n:

   (a) Show that `sumOddCubes n` $< 2n^4$ for all $n \geq 1$:

   ```
   sumOddCubes :: Int -> Int
   -- Pre-condition: n >= 1
   sumOddCubes 1 = 1
   sumOddCubes n = (2*n-1)^3 + (sumOddCubes (n-1))
   ```

   (b) Show that `sumChoices n` $= 2^n - 1$, for all $n \geq 1$, where

   ```
   sumChoices :: Integer -> Integer
   -- Pre-condition: n >= 1
   sumChoices n = sum [choose (n,r) | r <- [1..n]]

   choose :: (Integer, Integer) -> Integer
   -- Pre-condition: n >= r and n, r >= 0
   choose (n, r) = div (factorial n) ((factorial r)
                                       * (factorial (n-r)))
   factorial :: Integer -> Integer
   -- Pre-condition: n >= 1
   factorial 0 = 1
   factorial n = product [1..n]
   ```

   To relate your induction step to your induction assumption, you may use the fact that for all $1 \leq r \leq n$:

   $$\texttt{choose}\ (\texttt{n}+1, \texttt{r}) = \texttt{choose}\ (\texttt{n}, \texttt{r}) + \texttt{choose}\ (\texttt{n}, \texttt{r}-1)$$

   without proof.

2. Show, using structural induction, that for all `ts` :: `BTree a`:

$$(\texttt{numBTelem ts}) = \texttt{length}\ (\texttt{flattenTree ts})$$

```
data BTree a
  = BTempty
  | BTnode (BTree a) a (BTree a)

flattenTree :: BTree a -> [a]
flattenTree BTempty = []
flattenTree (BTnode lhs i rhs)
   = (flattenTree lhs) ++ [i]
     ++ (flattenTree rhs)

numBTelem :: BTree a -> Int
numBTelem BTempty = 0
numBTelem (BTnode lhs x rhs) = 1 + (numBTelem lhs)
                               + (numBTelem rhs)

length :: [a] -> Int
length [] = 0
length (x:xs) = 1 + (length xs)
```

You may assume the property: `length` (`xs ++ ys`) = `length xs`+`length ys`

3. Inductions over functions with multiple parameters sometimes (not always) require several induction arguments. For instance, in being asked to prove: for all x, for all y, show $F(x, y)$, for some proposition F. In performing induction over the first variable, x, you would start with the proposition $P(x)$ = for all y, $F(x, y)$. In trying to prove the base case, $P(0)$, it may be that a second induction argument is required, i.e. an induction argument over y when $x = 0$. It may also be the case that the induction step $P(k+1)$ requires a further induction argument, again over y and this time when $x = k + 1$.

   The program below is Ackermann's function.

```
ack :: Int -> Int -> Int
-- Pre-condition: m >= 0 and n >= 0
ack m n
    | (m == 0) && (n >= 0)  = n+1
    | (m > 0) && (n == 0)   = ack (m-1) 1
    | (m > 0) && (n > 0)    = ack (m-1) (ack m (n-1))
```

   Prove that:

$$\text{for all } m \geq 0, \text{for all } n \geq 0, (\texttt{ack m n}) \text{ terminates}$$

As usual, take your induction proposition to be:

$$P(m) = \text{for all } n \geq 0, (\texttt{ack m n}) \text{ terminates}$$

and perform induction on m. You will find the base case does not require a further induction, however the induction step does. You will need to remember that the induction assumption, $P(k)$, from the induction over m applies to the entire induction argument over n in the induction step.

You may find it useful to create a second proposition:

$$Q(n) = (\texttt{ack } (\texttt{k} + \texttt{1}) \texttt{ n}) \text{ terminates}$$

at the appropriate moment in your argument.

Why does a single induction over just m fail?

4. Prove $P(\texttt{xs})$ for all lists, `xs`:

$$\begin{aligned} P(\texttt{xs}) &= Q(\texttt{xs}) \wedge R(\texttt{xs}) \\ Q(\texttt{xs}) &= \texttt{x} \notin \texttt{filterX x xs} \\ R(\texttt{xs}) &= (\text{there exists } \texttt{qs} \text{ such that } (\texttt{merge qs (filterX x xs)}) = \texttt{xs}) \\ &\quad \wedge \text{ for all } \texttt{q} \in \texttt{qs} \Rightarrow \texttt{q} = \texttt{x} \end{aligned}$$

by proving $Q(\texttt{xs})$ and $R(\texttt{xs})$ by induction separately.

```
filterX :: (Ord a, Eq a) => a -> [a] -> [a]
-- Pre-condition: input list should be in ascending order
filterX x [] = []
filterX x (y:ys)
    | (x == y)  = filterX x ys
    | otherwise = y : filterX x ys

merge :: Ord a => [a] -> [a] -> [a]
merge [] ys = ys
merge xs [] = xs
merge (x:xs) (y:ys)
    | x < y     = x : (merge xs (y:ys))
    | otherwise = y : (merge (x:xs) ys)
```

5. A datatype `Ball` is defined such that $+$ and $*$ are defined for any pair of variables of type `Ball`:

```
data Ball = ...

instance Num Ball where
   (*) :: Ball -> Ball -> Ball
   b1 * b2 = ...
   (+) :: Ball -> Ball -> Ball
   b1 + b2 = ...
```

Also for any `b` :: `Ball` and positive integer, n: $n\mathtt{b} = \overbrace{\mathtt{b} + \mathtt{b} + \cdots + \mathtt{b}}^{n}$

A function `compD` has the datatype `compD :: Ball -> Ball`.

You do not need to know how `compD`, $+$ or $*$ are implemented or the details of how a `Ball` is represented. All you are given is the following properties of `compD`; for any `h1`,`h2` :: `Ball`:

```
compD (h1 * h2) = (h1 * (compD h2)) + ((compD h1) * h2)
```

and for n, m integers, `compD` is linear, i.e. :

```
compD (n * h1 + m * h2) = n * (compD h1) + m * (compD h2)
```

The function `applyND` applies `compD` n times to a `Ball`:

```
applyND :: Ball -> Int  -> Ball
applyND b 0 = b
applyND b n = compD (applyND b (n-1))
```

Show by induction on n, that for any two balls f, g, the following property holds for all $n \geq 1$:

$$\mathtt{applyND}\ (f * g)\ n = \sum_{r=0}^{n} \left(\begin{array}{c} n \\ r \end{array} \right) (\mathtt{applyND}\ f\ (n-r)) * (\mathtt{applyND}\ g\ r)$$

where:

$$\left(\begin{array}{c} n \\ r \end{array} \right) = \frac{n!}{r!(n-r)!}$$

You are allowed to the use the fact that for $1 \leq r \leq n$:

$$\left(\begin{array}{c} n \\ r \end{array} \right) + \left(\begin{array}{c} n \\ r-1 \end{array} \right) = \left(\begin{array}{c} n+1 \\ r \end{array} \right)$$

without proof.