COL765
Minor Exam solutions

Problem 1:

Program

```
addls_aux []     []     o c = (reverse o, c)
addls_aux (x:xs) (y:ys) o c =
  let ns = (x+y+c)`rem`10 in
  let nc = (x+y+c)`div`10 in
      addls_aux xs ys (ns:o) nc

addls n1 n2 = addls_aux n1 n2 [] 0
```

Proof of correctness/Invariant

From the usual definition of sum of two sequences of digits, we have:

$$Sum(a_{n-1} \dots a_1 a_0, b_{n-1} \dots b_1 b_0, carry) = \begin{cases} rem(s, 10) + 10 * Sum(a_{n-1} \dots a_1, b_{n-1} \dots b_1, s/10), & \text{for } n \geq 1 \\ 0, & otherwise \end{cases} \tag{1}$$

where, $s = (a_0 + b_0 + carry)$.
Similar recursion can be defined for carry.

Inductive invariant:

For input lists $N1$ and $N2$, at each call to `addls_aux n1 n2 o c`, we have

$$reverse(\mathtt{o}){+}{+}SumL(\mathtt{n1}, \mathtt{n2}, \mathtt{c}) = SumL(N1, N2, 0)$$

where `reverse` is reverse of list, `++` is the concatenation operator, and $SumL$ is the Sum definition (eq. (1)) extended to lists.

Proof of correctness:

At entry: `o` is `[]` and `n1`, `n2` are $N1$, $N2$ respectively. Hence the invariant holds.
Maintenance: Assume the invariant is true for the parameters, then we need to prove that it holds for the arguments in the next call as well. We have,

$$reverse(\mathtt{o}){+}{+}SumL(\mathtt{n1}, \mathtt{n2}, \mathtt{c}) = SumL(N1, N2, 0)$$

Expanding `n1` to `x:xs` and `n2` to `y:ys`,

$$reverse(\mathtt{o}){+}{+}SumL(\mathtt{x} : \mathtt{xs}, \mathtt{y} : \mathtt{ys}, \mathtt{c}) = SumL(N1, N2, 0)$$

Using eq. (1),

$$reverse(\mathtt{o}){+}{+}(rem_{10}(\mathtt{x} + \mathtt{y} + \mathtt{c}){:}SumL(\mathtt{xs}, \mathtt{ys}, (\mathtt{x} + \mathtt{y} + \mathtt{c})/10)) = SumL(N1, N2, 0)$$

Moving around the list head,

$$reverse(rem_{10}(\mathtt{x} + \mathtt{y} + \mathtt{c}){:}\mathtt{o}){+}{+}SumL(\mathtt{xs}, \mathtt{ys}, (\mathtt{x} + \mathtt{y} + \mathtt{c})/10) = SumL(N1, N2, 0)$$

Hence, the invariant holds in this case as well.

Termination: At the end, `n1` and `n2` are empty. Thus, $SumL(N1, N2, 0) = reverse(\mathtt{o})$, which is completes the proof.
Similar reasoning can be made for carry.

Problem 2:

```
module MySet where
data SetT a = SetOf [a]
    deriving (Eq, Show)

empty::SetT a
empty = SetOf []

belongs:: Eq a => a -> SetT a -> Bool
belongs x (SetOf l) =
  case l of
    [] -> False
    y:ys -> if x == y
              then True
            else belongs x (SetOf ys)

add:: Eq a => a -> SetT a -> SetT a
add x (SetOf l) =
  if belongs x (SetOf l)
     then (SetOf l)
    else SetOf (x:l)
setUnion:: Eq a => SetT a -> SetT a -> SetT a
setUnion (SetOf l) (SetOf m) =
      case l of
        [] -> (SetOf m)
          x:xs -> if belongs x (SetOf m)
                       then setUnion (SetOf xs) (SetOf m)
                  else setUnion (SetOf xs) (SetOf (x:m))

setIntersection:: Eq a => SetT a -> SetT a -> SetT a
setIntersection (SetOf l) (SetOf m) =
  SetOf (intersectmerge l m)
    where
      intersectmerge [] ys = []
          intersectmerge xs [] = []
          intersectmerge (x:xs) (y:ys)
            | x < y = (intersectmerge xs (y:ys))
            | y < x = (intersectmerge (x:xs) ys)
            | otherwise = x:(intersectmerge xs ys)
```

Problem 3:

Basis. (viz that 01 is valid and #0(01) = #1(01) =1 and length of of 01 is 2.
IH. Assume for all valid bit strings s of length $< k$, for some $k \geq 2$, that #0(s) = #1(s).
Induction step. Consider any valid bitstring u of length k. If $k = 2$ then by rule 0 and rule3 there is exactly one bitstring 01 which is valid and for which we have
#0(01) = #1(01) =1. If $k > 2$ then we have the following cases.

- Case u = 0s1 where s is a valid bitstring of length $< k$. By IH we have #0(s) = #1(s). Hence #0(u) = #0(s) +1 = #1(s) + 1 = #1(u)
- Case u = st where s and t are valid bitstrings each of length $< k$ resp. By IH we have #0(s) = #1(s) and #0(t) = #1(t). It follows that #0(u) = #0(s) + #0(t) = #1(s) + #1(t) = #1(u).