# Functional Programming and Verification

Prof. Dr. H. Seidl, N. Hartmann, R. Vogler WS 2018/19

**Exercise Sheet 12** Deadline: 27.01.2019

## General Information

Detailed information about the lecture, tutorials and homework assignments can be found on the lecture website[1]. Solutions have to be submitted to Moodle[2]. Make sure your uploaded documents are readable. Blurred images will be rejected. Use Piazza[3] to ask questions and discuss with your fellow students.

### Assignment 12.1 (L) What the fact

Consider the following function definitions:

```
let rec fact n = match n with 0 -> 1
  | n -> n * fact (n-1)

let rec fact_aux x n = match n with 0 -> x
  | n -> fact_aux (n*x) (n-1)

let fact_iter = fact_aux 1
```

Assume that all expressions terminate. Show that

`fact_iter n = fact n`

holds for all non-negative inputs $n \in \mathbb{N}_0$.

### Suggested Solution 12.1

We show that `fact_iter n = fact n`, resp. that `fact_aux 1 n = fact n` by induction on `n`.

- Base case: `n = 0`

$$
\begin{aligned}
& \texttt{fact\_aux 1 0} \\
\overset{\texttt{f\_a}}{=}\; & \texttt{match 0 with 0 -> 1 | n -> fact\_aux (n*1) (n-1)} \\
\overset{\texttt{match}}{=}\; & \texttt{1} \\
\overset{\texttt{match}}{=}\; & \texttt{match 0 with 0 -> 1 | n -> n * fact (n-1)} \\
\overset{\texttt{fact}}{=}\; & \texttt{fact 0}
\end{aligned}
$$

[1] https://www.in.tum.de/i02/lehre/wintersemester-1819/vorlesungen/functional-programming-and-verification/

[2] https://www.moodle.tum.de/course/view.php?id=44932

[3] https://piazza.com/tum.de/fall2018/in0003/home

- Inductive step: We assume `fact_aux 1 n = fact n` holds for an input $n \geq 0$. Now we try to prove that it also holds for $n+1$:

$$
\begin{aligned}
& \texttt{fact\_aux 1 (n+1)} \\
\overset{\texttt{f\_a}}{=}\; & \texttt{match n+1 with 0 -> 1 | n -> fact\_aux (n*1) (n-1)} \\
\overset{\texttt{match}}{=}\; & \texttt{fact\_aux ((n+1)*1) ((n+1)-1)} \\
\overset{arith}{=}\; & \texttt{fact\_aux (n+1) n} \\
& \text{our proof fails here} \\
=\; & \texttt{(n+1) * fact n} \\
\overset{arith}{=}\; & \texttt{(n+1) * fact ((n+1)-1)} \\
\overset{\texttt{match}}{=}\; & \texttt{match n+1 with 0 -> 1 | n -> n * fact (n-1)} \\
\overset{\texttt{fact}}{=}\; & \texttt{fact (n+1)}
\end{aligned}
$$

We fail, because we cannot use the induction hypothesis to rewrite one side into the other. The reason is that our hypothesis holds only for the special case where `x` is exactly `1`. Since the value of argument `x` changes between recursive calls, we have to state (and prove) a more general equality between the two sides that holds for arbitrary `x`. It is easy to see that `x` is used as an accumulator here and the function simply multiplies the factorial of `n` onto its initial value. Thus, for an arbitrary `x`, `fact_aux x n` computes $x * n!$. In order for the other side to compute the exact same value, we have also have to multiply by the initial value of `x`:

$$\texttt{fact\_aux acc n = acc * fact n}$$

Now, we try to prove this by induction on `n`:

- Base case: `n = 0`

$$
\begin{aligned}
& \texttt{fact\_aux acc 0} \\
\overset{\texttt{f\_a}}{=}\; & \texttt{match 0 with 0 -> acc | n -> fact\_aux (n*acc) (n-1)} \\
\overset{\texttt{match}}{=}\; & \texttt{acc} \\
\overset{arith}{=}\; & \texttt{acc * 1} \\
\overset{\texttt{match}}{=}\; & \texttt{acc * match 0 with 0 -> 1 | n -> n * fact (n-1)} \\
\overset{\texttt{fact}}{=}\; & \texttt{acc * fact 0}
\end{aligned}
$$

- Inductive step: We assume `fact_aux acc n = acc * fact n` holds for an input

$n \geq 0$. Now, we show that it holds for $n+1$ as well:

```
      fact_aux acc (n+1)
f_a
 =    match n+1 with 0 -> acc | n -> fact_aux (n*acc) (n-1)
match
 =    fact_aux ((n+1)*acc) ((n+1)-1)
arith
 =    fact_aux ((n+1)*acc) n
I.H.
 =    (n+1) * acc * fact n
arith
 =    acc * (n+1) * fact ((n+1)-1)
match
 =    acc * match n+1 with 0 -> 1 | n -> n * fact (n-1)
fact
 =    acc * fact (n+1)
```

This proof succeeds, as we can now make use of the (more general) induction hypothesis.

□

### Assignment 12.2 (L) Arithmetic 101

Let these functions be defined:

```
let rec summa l = match l with [] -> 0
    | h::t -> h + summa t

let rec sum l a = match l with [] -> a
    | h::t -> sum t (h+a)

let rec mul i j a = if i <= 0 then a
  else mul (i-1) j (j+a)
```

Prove that, under the assumption that all expressions terminate, for arbitrary `l` and $c \geq 0$ it holds that:

```
mul c (sum l 0) 0 = c * summa l
```

### Suggested Solution 12.2

Both `sum` and `mul` use an accumulator in their tail recursive implementation. Thus, we have to generalize the claim to:

```
mul c (sum l acc1) acc2 = acc2 + c * (acc1 + summa l)
```

First we prove a lemma by induction on the length $n$ of the list `l`:

**Lemma 1:** `sum l acc1 = acc1 + summa l`

- Base case: `l = []`

```
      sum [] acc1
sum
 =    match [] with [] -> acc1 | h::t -> sum t (h+acc1)
match
 =    acc1
match
 =    acc1 + match [] with [] -> 0 | h::t -> h + summa t
summa
 =    acc1 + summa []
```

- Inductive step: We assume `sum l acc1 = acc1 + summa l` holds for a list `xs` of length $n \geq 0$. Now, we show that it then also holds for a list `x::xs` of length $n+1$:

$$
\begin{aligned}
& \texttt{sum (x::xs) acc1} \\
\overset{\texttt{sum}}{=}\ & \texttt{match x::xs with [] -> acc1 | h::t -> sum t (h+acc1)} \\
\overset{\texttt{match}}{=}\ & \texttt{sum xs (x+acc1)} \\
\overset{I.H.}{=}\ & \texttt{x + acc1 + summa xs} \\
\overset{comm}{=}\ & \texttt{acc1 + x + summa xs} \\
\overset{\texttt{match}}{=}\ & \texttt{acc1 + match x::xs with [] -> 0 | h::t -> h + summa t} \\
\overset{\texttt{summa}}{=}\ & \texttt{acc1 + summa (x::xs)}
\end{aligned}
$$

Next, we prove the initial statement by induction on $c$:

- Base case: `c = 0`

$$
\begin{aligned}
& \texttt{mul 0 (sum l acc1) acc2)} \\
\overset{\texttt{mul}}{=}\ & \texttt{if 0 <= 0 then acc2 else mul (0-1) (sum l acc1) ((sum l acc1)+acc2)} \\
\overset{\texttt{if}}{=}\ & \texttt{acc2} \\
\overset{arith}{=}\ & \texttt{acc2 + 0 * (acc1 + summa l)}
\end{aligned}
$$

- Inductive step: We assume the statement holds for a $c \geq 0$. Now, we show that it also holds for $c+1$:

$$
\begin{aligned}
& \texttt{mul (c+1) (sum l acc1) acc2)} \\
\overset{\texttt{mul}}{=}\ & \texttt{if c+1 <= 0 then acc2 else mul c (sum l acc1) ((sum l acc1) + acc2)} \\
\overset{\texttt{if}}{=}\ & \texttt{mul c (sum l acc1) ((sum l acc1) + acc2)} \\
\overset{I.H.}{=}\ & \texttt{(sum l acc1) + acc2 + c * (acc1 + summa l)} \\
\overset{comm}{=}\ & \texttt{acc2 + c * (acc1 + summa l) + (sum l acc1)} \\
\overset{L.1}{=}\ & \texttt{acc2 + c * (acc1 + summa l) + (acc1 + summa l)} \\
\overset{Distr}{=}\ & \texttt{acc2 + (c+1) * (acc1 + summa l)}
\end{aligned}
$$

This proves the statement. □

### Assignment 12.3 (L) Counting nodes

A binary tree and two functions to count the number of nodes in such a tree are defined as follows:

```
type tree = Node of tree * tree | Empty

let rec nodes t = match t with Empty -> 0
```

```
    | Node (l,r) -> 1 + (nodes l) + (nodes r)

let rec count t =
  let rec aux t a = match t with Empty -> a
    | Node (l,r) -> aux r (aux l (a+1))
  in
  aux t 0
```

Prove or disprove the following statement for arbitary trees `t`:

`nodes t = count t`

**Suggested Solution 12.3**

The statement holds. First, we show that `nodes t = aux t 0` or, more precisely, the generalized statement `acc + nodes t = aux t acc` holds. We prove by induction on the structure of trees:

- Base case: `t = Empty`

```
      acc + nodes Empty
nodes
 =    acc + match Empty with Empty -> 0
          | Node (l,r) -> 1 + (nodes l) + (nodes r)
match
 =    acc
match
 =    match Empty with Empty -> acc
          | Node (l,r) -> aux r (aux l (acc+1))
aux
 =    aux Empty acc
```

- Inductive step: Assume the above equivalence holds for two trees `a` and `b`. Now, we show that it then also holds for a tree `Node (a, b)`:

```
      acc + nodes (Node (a,b))
nodes
 =    acc + match Node (a,b) with Empty -> 0
          | Node (l,r) -> 1 + (nodes l) + (nodes r)
match
 =    acc + 1 + (nodes a) + (nodes b)
I.H.
 =    aux b (acc + 1 + nodes a)
I.H.
 =    aux b (aux a (acc+1))
match
 =    match Node (a,b) with Empty -> acc | Node (l,r) -> aux r (aux l (acc+1))
aux
 =    aux (Node (a,b)) acc
```

Finally, we show:

`nodes t` $\overset{arith}{=}$ `0 + nodes t` $\overset{theor}{=}$ `aux t 0` $\overset{\texttt{count}}{=}$ `count t`

□

**Assignment 12.4 (H) Len or nlen?** [5 Points]

The following functions are defined:

```
let rec nlen n l = match l with [] -> 0
  | h::t -> n + nlen n t

let rec fold_left f a l = match l with [] -> a
  | h::t -> fold_left f (f a h) t

let rec map f l = match l with [] -> []
  | h::t -> f h :: map f t

let (+) a b = a + b
```

Show that the statement

```
nlen n l = fold_left (+) 0 (map (fun _ -> n) l)
```

holds for arbitrary `l` and `n`. Assume that all expressions do terminate.

**Suggested Solution 12.4**

We have to prove the more general statement:

```
acc + nlen n l = fold_left (+) acc (map (fun _ -> n) l)
```

We do so by induction on the length $k$ of list `l`.

- Base case: $k = 0$, so `l = []`

```
        acc + nlen n []
nlen  = acc + match [] with [] -> 0 | h::t -> n + nlen n t
match = acc + 0
arith = acc
match = match [] with [] -> acc | h::t -> fold_left (+) ((+) acc h) t
f_l   = fold_left (+) acc []
match = fold_left (+) acc (match [] with [] -> []
               | h::t -> (fun _ -> n) h :: map (fun _ -> n) t)
map   = fold_left (+) acc (map (fun _ -> n) [])
```

- Inductive step: We assume the statement holds for a list `l = xs` of length $k \geq 0$.

Now, we prove it for `l = x::xs`:

```
     acc + nlen n (x::xs)
nlen
 =   acc + match x::xs with [] -> 0 | h::t -> n + nlen n t
match
 =   acc + n + nlen n xs
I.H.
 =   fold_left (+) (acc + n) (map (fun _ -> n) xs)
(+)
 =   fold_left (+) ((+) acc n) (map (fun _ -> n) xs)
match
 =   match n::map (fun _ -> n) xs with [] -> acc
         | h::t -> fold_left (+) ((+) acc h) t
f_l
 =   fold_left (+) acc (n :: map (fun _ -> n) xs)
fun
 =   fold_left (+) acc ((fun _ -> n) x :: map (fun _ -> n) xs)
match
 =   fold_left (+) acc (match x::xs with [] -> []
         | h::t -> (fun _ -> n) h :: map (fun _ -> n) t)
map
 =   fold_left (+) acc (map (fun _ -> n) (x::xs))
```

□

### Assignment 12.5 (H) Fun with fold [8 Points]

Given are the following functions with semantics as usual:

```
let rec fl f a l = match l with [] -> a
  | x::xs -> fl f (f a x) xs
let rec fr f l a = match l with [] -> a
  | x::xs -> f x (fr f xs a)
let rec rev_map f l a = match l with [] -> a
  | x::xs -> rev_map f xs (f x :: a)
let (+) a b = a + b
```

Prove that, if all expressions terminate, the statement

```
fl (+) 0 (rev_map (fun x -> x * 2) l []) = fr (fun x a -> a + 2 * x) l 0
```

holds for all inputs `l`.

### Suggested Solution 12.5

Note, that both `fl` and `rev_map` have an accumulator argument, which has to be generalized. Since `fl`'s accumulator is just an additional offset on the resulting sum, we merely add its value on the right hand side. However, `rev_map`'s accumulator is more difficult to handle. Since all values initially in this list end up in the final sum, we have to compute the same sum on the right hand side. We do this using `fr`, because using `fl` would introduce yet another accumulator. The general statement to proof is thus:

```
        fl (+) acc2 (rev_map (fun x -> x * 2) l acc1) =
(fr (fun x a -> a + 2 * x) l 0) + (fr (+) acc1 0) + acc2
```

We prove by induction on the length $n$ of list `l`:

- Base case: $n = 0$, so `l = []`:

$$
\begin{aligned}
& \texttt{fl (+) acc2 (rev\_map (fun x -> x * 2) [] acc1)} \\
\overset{\texttt{r\_m}}{=} & \texttt{ fl (+) acc2 (match [] with [] -> acc1} \\
& \quad \texttt{| x::xs -> rev\_map (fun x -> x * 2) xs ((fun x -> x * 2) x :: acc1))} \\
\overset{\texttt{match}}{=} & \texttt{fl (+) acc2 acc1} \\
& \text{we are going to prove this equality separately below} \\
= & \texttt{ (fr (+) acc1 0) + acc2} \\
\overset{\texttt{match}}{=} & \texttt{(match [] with [] -> 0 | x::xs -> (fun x a -> a + 2 * x) x} \\
& \quad \texttt{(fr (fun x a -> a + 2 * x) xs 0)) + (fr (+) acc1 0) + acc2} \\
\overset{\texttt{fr}}{=} & \texttt{ (fr (fun x a -> a + 2 * x) [] 0) + (fr (+) acc1 0) + acc2}
\end{aligned}
$$

Now, we show that `fl (+) acc2 acc1 = (fr (+) acc1 0) + acc2` by induction on the length $k$ of list `acc1`:

  - Base case: `acc1 = []` ($k = 0$)

$$
\begin{aligned}
& \texttt{fl (+) acc2 []} \\
\overset{\texttt{fl}}{=} & \texttt{ match [] with [] -> acc2 | x::xs -> fl (+) ((+) acc2 x) xs} \\
\overset{\texttt{match}}{=} & \texttt{acc2} \\
\overset{\texttt{match}}{=} & \texttt{match [] with [] -> 0 | x::xs -> (+) x (fr (+) xs 0) + acc2} \\
\overset{\texttt{fr}}{=} & \texttt{ (fr (+) [] 0) + acc2}
\end{aligned}
$$

  - Inductive step: We show that the statement holds for list `acc1 = x::xs` of length $k + 1$ under the assumption that it holds for the list `xs` of length $k$:

$$
\begin{aligned}
& \texttt{fl (+) acc2 (x::xs)} \\
\overset{\texttt{fl}}{=} & \texttt{ match x::xs with [] -> acc2 | x::xs -> fl (+) ((+) acc2 x) xs} \\
\overset{\texttt{match}}{=} & \texttt{fl (+) ((+) acc2 x) xs} \\
\overset{\texttt{(+)}}{=} & \texttt{ fl (+) (acc2+x) xs} \\
\overset{I.H.}{=} & \texttt{(fr (+) xs 0) + acc2 + x} \\
\overset{comm}{=} & \texttt{x + (fr (+) xs 0) + acc2} \\
\overset{\texttt{(+)}}{=} & \texttt{ (+) x (fr (+) xs 0) + acc2} \\
\overset{\texttt{match}}{=} & \texttt{(match x::xs with [] -> 0 | x::xs -> (+) x (fr (+) xs 0)) + acc2} \\
\overset{\texttt{fr}}{=} & \texttt{ (fr (+) (x::xs) 0) + acc2}
\end{aligned}
$$

  This concludes the base case.

- Inductive step: We assume the equality holds for a list `xs` of length $n \geq 0$ and then show that it holds for a list `x::xs` of length $n + 1$:

```
        fl (+) acc2 (rev_map (fun x -> x * 2) (x::xs) acc1)
r_m   = fl (+) acc2 (match x::xs with [] -> acc1
            | x::xs -> rev_map (fun x -> x * 2) xs ((fun x -> x * 2) x :: acc1))
match = fl (+) acc2 (rev_map (fun x -> x * 2) xs ((fun x -> x * 2) x :: acc1))
fun   = fl (+) acc2 (rev_map (fun x -> x * 2) xs (x * 2 :: acc1))
I.H.  = (fr (fun x a -> a + 2 * x) xs 0) + (fr (+) (x * 2 :: acc1) 0) + acc2
fr    = (fr (fun x a -> a + 2 * x) xs 0) + (match x*2::acc1 with [] -> 0
            | x::xs -> (+) x (fr (+) xs 0)) + acc2
match = (fr (fun x a -> a + 2 * x) xs 0) + ((+) (x*2) (fr (+) acc1 0)) + acc2
(+)   = (fr (fun x a -> a + 2 * x) xs 0) + 2 * x + (fr (+) acc1 0) + acc2
fun   = ((fun x a -> a + 2 * x) x (fr (fun x a -> a + 2 * x) xs 0)
            + (fr (+) acc1 0) + acc2
match = (match x::xs with [] -> 0 | x::xs -> (fun x a -> a + 2 * x) x
            (fr (fun x a -> a + 2 * x) xs 0)) + (fr (+) acc1 0) + acc2
fr    = (fr (fun x a -> a + 2 * x) (x::xs) 0) + (fr (+) acc1 0) + acc2
```

We have proven our quite generalized equality. Lastly, we show that it follows from there that our initial statement holds:

```
        fl (+) 0 (rev_map (fun x -> x * 2) l [])
      = (fr (fun x a -> a + 2 * x) l 0) + (fr (+) [] 0) + 0
fr    = (fr (fun x a -> a + 2 * x) l 0) + (match [] with [] -> 0
            | x::xs -> (+) x (fr (+) xs 0))
match = fr (fun x a -> a + 2 * x) l 0
```

This concludes our proof. $\square$

**Assignment 12.6 (H) Trees** [7 Points]

Once again, we define binary trees and some functions for them:

```
type tree = Empty | Node of int * tree * tree

let rec fl f a l = match l with [] -> a
  | x::xs -> fl f (f a x) xs

let rec app l1 l2 = match l1 with [] -> l2
  | x::xs -> x::app xs l2

let rec tf f b t = match t with Empty -> b
  | Node (x, l, r) -> f (tf f b l) x (tf f b r)

let rec to_list t = match t with Empty -> []
  | Node (x, l, r) -> app (to_list l) (x::to_list r)

let add3 a b c = a + b + c
let (+) a b = a + b
```

Assume all expressions terminate, then proof for all trees `t`:

```
fl (+) 0 (to_list t) = tf add3 0 t
```

*Hint: If you get stuck during your proof, try to formulate additional equalities that help to reach your goal. Don't forget to prove them, however!*

**Suggested Solution 12.6**

We start by checking whether we need to generalize the above claim. Since `fl` (aka `fold_left`) uses an accumulator argument (that is modified during recursive calls), it is quite likely that we need to replace the initial `0` in the claim by an arbitary `acc`, which will increase the result on the left hand side of our equality by `acc`, so in order to restore equality, we also have to add `acc` to the right hand side. The tree folding function `tf` has no such argument, so `acc` is the only argument which has to be generalized. This results in our proof goal:

```
fl (+) acc (to_list t) = acc + tf add3 0 t
```

Again, we proof by induction on the structure of trees:

- Base case: `t = Empty`

```
      fl (+) acc (to_list Empty)
to_l
 =   fl (+) acc (match Empty with Empty -> []
              | Node (x, l, r) -> app (to_list l) (x::to_list r))
match
 =   fl (+) acc []
fl
 =   match [] with [] -> acc | x::xs -> fl (+) ((+) acc x) xs
match
 =   acc
match
 =   acc + match Empty with Empty -> 0
              | Node (x, l, r) -> app (to_list l) (x::to_list r)
tf
 =   acc + tf add3 0 Empty
```

- Inductive step: We assume our equality holds for two trees `t1` and `t2` and try to show that it then also holds for a tree `Node (v, t1, t2)`. Therefore, we just start a proof using substitution rules until we reach a point at which we cannot proceed (we will then proof additional equalities that may help us there):

```
      fl (+) acc (to_list (Node (v, t1, t2)))
to_l
 =   fl (+) acc (match Node (v, t1, t2) with Empty -> []
              | Node (x, l, r) -> app (to_list l) (x::to_list r))
match
 =   fl (+) acc (app (to_list t1) (v::to_list t2))
 =   how to proceed here?
 =   acc + (fl (+) 0 (to_list t1)) + v + (fl (+) 0 (to_list t2))
add3
 =   acc + add3 (fl (+) 0 (to_list t1)) v (fl (+) 0 (to_list t2))
I.H.
 =   acc + add3 (tf add3 0 t1) v (tf add3 0 t2)
match
 =   acc + match Node (v, t1, t2) with Empty -> 0
              | Node (x, l, r) -> add3 (tf add3 0 l) x (tf add3 0 r)
tf
 =   acc + tf add3 0 (Node (v, t1, t2))
```

At this point, we cannot continue with our existing set of rules and equalities, so we need to prove some (or multiple) lemma first. Two things we might notice here:

1. Somehow, we want to move the `acc` from inside the `fl` expression to the front. Think about how `fl` (`fold_left`) computes the sum with `(+)` for an `acc`. Its not too difficult to see that in fact `fl (+) acc l = acc + fl (+) 0 l` or, in general:

   **Lemma 1:** `fl (+) (u+v) l = u + fl (+) v l`

2. When comparing the top and bottom expressions, a main difference is that in the former, we concatenate two lists and then compute the sum of the resulting list, while in the latter, we compute the sums of the lists individually and add the result. Clearly, $sum(append(l1, l2)) = sum(l1) + sum(l2)$, so we may try to prove this as well:

**Lemma 2:** `fl (+) 0 (app l1 l2) = (fl (+) 0 l1) + (fl (+) 0 l2)`

Let's assume these equalities hold (we will give a proof in the end) and continue the above proof from where we left off:

```
        fl (+) acc (app (to_list t1) (v::to_list t2))
 =L1=   acc + fl (+) 0 (app (to_list t1) (v::to_list t2))
 =L2=   acc + (fl (+) 0 (to_list t1)) + (fl (+) 0 (v::to_list t2))
 =fl=   acc + (fl (+) 0 (to_list t1))
            + (match v::to_list t2 with [] -> 0 | x::xs -> fl (+) ((+) 0 x) xs)
=match= acc + (fl (+) 0 (to_list t1)) + (fl (+) ((+) 0 v) (to_list t2))
 =(+)=  acc + (fl (+) 0 (to_list t1)) + (fl (+) v (to_list t2))
 =L1=   acc + (fl (+) 0 (to_list t1)) + v + (fl (+) 0 (to_list t2))
```

We now prove lemma 1 by induction on the length of the list `l`:

- Base case: `l = []`

```
        fl (+) (u+v) []
 =fl=   match [] with [] -> u+v | x::xs -> fl (+) ((+) (u+v) x) xs
=match= u + v
=match= u + match [] with [] -> v | x::xs -> fl (+) ((+) v x) xs
 =fl=   u + fl (+) v []
```

- Inductive step: We assume it holds for a list `xs` of length $n \geq 0$ and show it for `x::xs` of length $n+1$:

```
        fl (+) (u+v) (x::xs)
 =fl=   match x::xs with [] -> u+v | x::xs -> fl (+) ((+) (u+v) x) xs
=match= fl (+) ((+) (u+v) x) xs
 =(+)=  fl (+) (u+v+x) xs
=I.H.=  u + fl (+) (v+x) xs
 =(+)=  u + fl (+) ((+) v x) xs
=match= u + match x::xs with [] -> v | x::xs -> fl (+) ((+) v x) xs
 =fl=   u + fl (+) v (x::xs)
```

Lastly, we give a proof for lemma 2. Again we use induction on the length of list `l1`:

- Base case: `l1 = []`

  `fl (+) 0 (app [] l2)`<br>
  $\overset{\texttt{app}}{=}$ `fl (+) 0 (match [] with [] -> l2 | x::xs -> x::app xs l2)`<br>
  $\overset{\texttt{match}}{=}$ `fl (+) 0 l2`<br>
  $\overset{\texttt{match}}{=}$ `(match [] with [] -> 0 | x::xs -> fl (+) ((+) 0 x) xs) + (fl (+) 0 l2)`<br>
  $\overset{\texttt{fl}}{=}$ `(fl (+) 0 []) + (fl (+) 0 l2)`

- Inductive step: We assume our statement holds for a list `xs` of length $n$, we now show that it follows that it holds for a list `x::xs` of length $n+1$:

  `fl (+) 0 (app (x::xs) l2)`<br>
  $\overset{\texttt{app}}{=}$ `fl (+) 0 (match x::xs with [] -> l2 | x::xs -> x::app xs l2)`<br>
  $\overset{\texttt{match}}{=}$ `fl (+) 0 (x::app xs l2)`<br>
  $\overset{\texttt{fl}}{=}$ `match x::app xs l2 with [] -> 0 | x::xs -> fl (+) ((+) 0 x) xs`<br>
  $\overset{\texttt{match}}{=}$ `fl (+) ((+) 0 x) (app xs l2)`<br>
  $\overset{\texttt{(+)}}{=}$ `fl (+) x (app xs l2)`<br>
  $\overset{L1}{=}$ `x + fl (+) 0 (app xs l2)`<br>
  $\overset{I.H.}{=}$ `x + (fl (+) 0 xs) + (fl (+) 0 l2)`<br>
  $\overset{L1}{=}$ `(fl (+) x xs) + (fl (+) 0 l2)`<br>
  $\overset{\texttt{(+)}}{=}$ `fl (+) ((+) 0 x) xs) + (fl (+) 0 l2)`<br>
  $\overset{\texttt{match}}{=}$ `(match x::xs with [] -> 0 | x::xs -> fl (+) ((+) 0 x) xs) + (fl (+) 0 l2)`<br>
  $\overset{\texttt{fl}}{=}$ `(fl (+) 0 (x::xs)) + (fl (+) 0 l2)`

This concludes our proof. $\square$