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`

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

[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 $n+1$:

```
     fact_aux 1 (n+1)
f_a
 =   match n+1 with 0 -> 1 | n -> fact_aux (n*1) (n-1)
match
 =   fact_aux ((n+1)*1) ((n+1)-1)
arith
 =   fact_aux (n+1) n
     our proof fails here
 =   (n+1) * fact n
arith
 =   (n+1) * fact ((n+1)-1)
match
 =   match n+1 with 0 -> 1 | n -> n * fact (n-1)
fact
 =   fact (n+1)
```

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 the initial value of `x`:

```
fact_aux acc n = acc * fact n
```

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

- Base case: `n = 0`

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

- 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 $n+1$:

```
      sum (x::xs) acc1
 sum
  =   match x::xs with [] -> acc1 | h::t -> sum t (h+acc1)
match
  =   sum xs (x+acc1)
I.H.
  =   x + acc1 + summa xs
comm
  =   acc1 + x + summa xs
match
  =   acc1 + match x::xs with [] -> 0 | h::t -> h + summa t
summa
  =   acc1 + summa (x::xs)
```

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

- Base case: `c = 0`

```
      mul 0 (sum l acc1) acc2)
 mul
  =   if 0 <= 0 then acc2 else mul (0-1) (sum l acc1) ((sum l acc1)+acc2)
 if
  =   acc2
arith
  =   acc2 + 0 * (acc1 + summa l)
```

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

```
      mul (c+1) (sum l acc1) acc2)
 mul
  =   if c+1 <= 0 then acc2 else mul c (sum l acc1) ((sum l acc1) + acc2)
 if
  =   mul c (sum l acc1) ((sum l acc1) + acc2)
I.H.
  =   (sum l acc1) + acc2 + c * (acc1 + summa l)
comm
  =   acc2 + c * (acc1 + summa l) + (sum l acc1)
 L.1
  =   acc2 + c * (acc1 + summa l) + (acc1 + summa l)
Distr
  =   acc2 + (c+1) * (acc1 + summa l)
```

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 (acc+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.

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`.

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!