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`:

```
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 $n+1$:

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

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

- Base case: `c = 0`

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

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

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

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
```