I2I-2 Functional Verification Example Excercises

Dimitri Tabatadze

July 3, 2023

1 Treecount

The following OCaml functions are given:

```ocaml
type node = Leaf of int | Inner of node * node

let rec countleaves t = match t with
  | Leaf _ -> 1
  | Inner (l, r) -> countleaves l + countleaves r

let rec countinner t = match t with
  | Leaf _ -> 0
  | Inner (l, r) -> 1 + countinner l + countinner r
```

Prove that the equality

```ocaml
count_leaves t = count_inner t + 1
```

holds.

Solution

- Case 1: `t = Leaf`

$$\begin{aligned}
\texttt{countleaves t} &\overset{\texttt{countleaves}}{=} \texttt{match t with Leaf -> 1 | Inner (l, r) ->} \\
&\qquad \texttt{countleaves l + countleaves r} \\
&\overset{\texttt{def t,match}}{=} \texttt{1} \\
&\overset{\texttt{arith}}{=} \texttt{0 + 1} \\
&\overset{\texttt{match,countinner}}{=} \texttt{countinner t + 1}
\end{aligned}$$

- Case 2: `t = Inner (a, b)`

$$\begin{aligned}
\texttt{countleaves t} &\overset{\texttt{countleaves}}{=} \texttt{match t with Leaf -> 1 | Inner (l, r) ->} \\
&\qquad \texttt{countleaves l + countleaves r} \\
&\overset{\texttt{def t,match}}{=} \texttt{countleaves a + countleaves b} \\
&\overset{\texttt{I.H.}}{=} \texttt{(countinner a + 1) + (countinner b + 1)} \\
&\overset{\texttt{arith}}{=} \texttt{(1 + countinner a + countinner b) + 1} \\
&\overset{\texttt{match,couninner,def}}{=} \texttt{countinner t + 1}
\end{aligned}$$

2 Fun With Fold

The following OCaml functions are given:

```
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)
```

Prove that the equality

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

holds.

Solution

To prove the given equality, we first generalize and try to prove that

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

Let's proove that

`fl (+) b a = b + (fr (+) a 0)`

$$
\begin{aligned}
\texttt{fl (+) b a =} \quad & \overset{\texttt{arith,(+)}}{=} && \texttt{fl (+) ((+) 0 b) a} \\
& \overset{\texttt{fl}}{=} && \texttt{fl (+) 0 (b::a)} \\
& \overset{\texttt{I.H.}}{=} && \texttt{0 + fr (+) (b::a) 0} \\
& \overset{\texttt{arith}}{=} && \texttt{fr (+) (b::a) 0} \\
& \overset{\texttt{(+)}}{=} && \texttt{(+) b (fr (+) a 0)} \\
& \overset{\texttt{(+)}}{=} && \texttt{b + (fr (+) a 0)}
\end{aligned}
$$

3 Even list

The following OCaml functions are given:

```
let rec el a lst = match lst with
  | h::_::t -> el (h::a) t
  | [h] -> el (h::a) []
  | [] -> a

let rec de i a lst = match lst with
  | h::t -> de (i+1) (if i mod 2 = 0 then h::a else a) t
  | [] -> a
```

Prove that the equality

```
de 0 [] l = el [] l
```

holds.

Solution

To prove that

$$\texttt{de 0 [] l = el [] l}$$

we generalize and try to prove

$$\texttt{de (2*i) a l = el a l}$$

Induction on the list `l`

- Base case 1: `l = []`

$$\begin{aligned}
\texttt{de (2*i) a l} &\overset{\texttt{de,def l,match}}{=\!=} \texttt{a} \\
&\overset{\texttt{match,def l,el}}{=\!=} \texttt{el a l}
\end{aligned}$$

- Base case 2: `l = [h] = h::[]`

$$\begin{aligned}
\texttt{de (2*i) a l} &\overset{\texttt{de,def l}}{=\!=} \texttt{match h::[] with | h::t -> de (2*i+1) (if 2*i mod} \\
&\qquad \texttt{2 = 0 then h::a else a) t | [] -> a} \\
&\overset{\texttt{match}}{=\!=} \texttt{de (2*i+1) (if 2*i mod 2 = 0 then h::a else a) []} \\
&\overset{\texttt{if}}{=\!=} \texttt{de (2*i+1) (h::a) []} \\
&\overset{\texttt{de,match}}{=\!=} \texttt{h::a} \\
&\overset{\texttt{match,el}}{=\!=} \texttt{el (h::a) []} \\
&\overset{\texttt{match,el}}{=\!=} \texttt{el a [h]} \\
&\overset{\texttt{def l}}{=\!=} \texttt{el a l}
\end{aligned}$$

- Induction step: `l = h1::h2::t`

$$
\begin{aligned}
\texttt{de (2*i) a l} \quad &\overset{\texttt{de,def l}}{=} \quad \texttt{match h1::h2::t with | h::t -> de (2*i+1) (if 2*i mod 2 = 0 then h::a else a) t | [] -> a} \\
&\overset{\texttt{match}}{=} \quad \texttt{de (2*i+1) (if 2*i mod 2 = 0 then h1::a else a) h2::t} \\
&\overset{\texttt{if,de,match}}{=} \quad \texttt{de (2*i+2) (if 2*i+1 mod 2 = 0 then h2::h1::a else h1::a) t} \\
&\overset{\texttt{if}}{=} \quad \texttt{de (2*i+2) (h1::a) t} \\
&\overset{\texttt{I.H.}}{=} \quad \texttt{el (h1::a) t} \\
&\overset{\texttt{match,el}}{=} \quad \texttt{el a (h1::h2::t)} \\
&\overset{\texttt{def l}}{=} \quad \texttt{el a l)}
\end{aligned}
$$

4 Bigotree

The following OCaml functions are given:

```
type node = Empty | Inner of node * int * node

let rec insertintree v t = match t with
  | Empty -> Inner (Empty, v, Empty)
  | Inner (l, u, r) -> if v > u then
      Inner (l, u, insertintree v r)
    else
      Inner (insertintree v l, u, r)

let rec totree a lst = match lst with
  | [] -> a
  | h::t -> insertintree h (totree a t)

let rec tolist t = match t with
  | Empty -> []
  | Inner (l, v, r) -> tolist l @ [v] @ tolist r

let rec insert n lst = match lst with
  | [] -> [n]
  | h::t -> if n > h then
      h::(insert n t)
    else
      n::h::t

let rec sort lst = match lst with
  | [] -> []
  | h::t -> insert h (sort t)
```

Prove that the equality

```
tolist (totree Empty l) = sort l
```

holds.

Solution

There is no solution.

To prove that

$$\texttt{tolist (totree Empty l) = sort l}$$

we need to fist prove that **Lemma 1:**

$$\texttt{tolist (insertintree v t) = insert v (tolist t)}$$

holds. We do this by induction on `t`

- Base case: `t = Empty`

$$\begin{aligned}
\texttt{tolist (insertintree v t)} &\stackrel{\texttt{insertintree,def t}}{=} \texttt{tolist (Inner (Empty, v, Empty))} \\
&\stackrel{\texttt{tolist,match}}{=} \texttt{tolist Empty @ [v] @ tolist Empty} \\
&\stackrel{\texttt{tolist,match}}{=} \texttt{[] @ [v] @ []} \\
&\stackrel{\texttt{@}}{=} \texttt{[v]} \\
&\stackrel{\texttt{match,insert}}{=} \texttt{insert v []} \\
&\stackrel{\texttt{match,tolist}}{=} \texttt{insert v (tolist Empty)} \\
&\stackrel{\texttt{def t}}{=} \texttt{insert v (tolist t)}
\end{aligned}$$

- Induction step: `t = Inner (l, x, r)`

$$\texttt{tolist (insertintree v t)} \overset{\texttt{insertintree,def t}}{=} \texttt{tolist (if v > x then Inner (l, x, insertintree v r) else Inner (insertintree v l, x, r))}$$

  - Case 1: `v > x`

$$\begin{aligned}
&\overset{\texttt{def x}}{=} \texttt{tolist (Inner (l, x, insertintree v r))}\\
&\overset{\texttt{tolist,match}}{=} \texttt{tolist l @ [x] @ tolist (insertintree v r)}\\
&\overset{\texttt{I.H.}}{=} \texttt{tolist l @ [x] @ (insert v (tolist r))}\\
&\overset{\texttt{Lemma 2.}}{=} \texttt{insert v (tolist l @ [x] @ tolist r)}\\
&\overset{\texttt{tolist,def t}}{=} \texttt{insert v (tolist t)}
\end{aligned}$$

  - Case 1: `v <= x`

$$\begin{aligned}
&\overset{\texttt{def x}}{=} \texttt{tolist (Inner (insertintree v l, x, r))}\\
&\overset{\texttt{tolist,match}}{=} \texttt{tolist (insertintree v l) @ [x] @ tolist r}\\
&\overset{\texttt{I.H.}}{=} \texttt{insert v (tolist l) @ [x] @ tolist r}\\
&\overset{\texttt{@,@}}{=} \texttt{insert v (tolist l @ [x] @ tolist r)}\\
&\overset{\texttt{tolist,def t}}{=} \texttt{insert v (tolist t)}
\end{aligned}$$

We now need to prove **Lemma 2**. Invariant: `lst` is a sorted list and every element is `<` `v`. `lst = x::xs`

$$\texttt{lst @ (insert v (tolist r)) = insert v (lst @ tolist r)}$$

$$\begin{aligned}
\texttt{insert v (lst @ tolist r))} &\overset{\texttt{def lst}}{=} \texttt{insert v (x::xs @ tolist r)}\\
&\overset{\texttt{insert,match,if}}{=} \texttt{x::(insert v (xs @ tolist r))}\\
&\overset{\texttt{I.H.}}{=} \texttt{x::(xs @ (insert v (tolist r)))}\\
&\overset{\texttt{associativity}}{=} \texttt{x::xs @ (insert v (tolist r))}\\
&\overset{\texttt{def lst}}{=} \texttt{lst @ (insert v (tolist r))}
\end{aligned}$$

Now we start the main proof with induction on length n of `l`

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

$$\texttt{tolist (totree Empty l)} \overset{\texttt{totree,def l}}{=} \texttt{tolist (Empty)}$$