I2I-2 Functional Verification Example Excercises

Dimitri Tabatadze

July 2, 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 Dacentrili

The following OCaml functions are given:

```
let rec damcentravi a lst = match lst with
  | [] -> a
  | h::t -> damcentravi (h::(List.rev a)) t

let rec get_nth n lst = match lst with
  | [] -> 0
  | h::t -> if n = 0 then h else get_nth (n-1) t

let pick_middle lst = get_nth ((List.length lst) / 2) lst;;
```

Prove that the equality

```
a = pick_middle (damcentravi [] (a::b))
```

holds.

Solution

To prove that

```
a = pick_middle (damcentravi [] (a::b))
```

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::[]`

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

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

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

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

To prove that

`tolist (totree Empty l) = sort l`

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

`tolist t = sort (tolist t)`

holds. We do this by induction on `t`

- Case 1: `t = Empty`

`tolist t`