P1. prove that FindMax satisfies the invariants of OptionFunction (uses induction on the list)

FindMax.domain lst = is_some (FindMax.fn lst)
Case lst = []
FindMax.domain lst
={case}
FindMax.domain []
={domain definition}
```
    Match [] with
        | [] -> false
        | _ -> true
```
={apply match}
False
={reversely match from right side}
={is_some definition}
```
    Match None with
    | Some _ -> true
    | _ -> false
```
Is_some None
={apply match}
= {FindMax.fn definition}
```
    Match [] ->
    | [] -> None
    | x :: xs -> ( match fn xs with
                | None -> Some x
                | Some v -> Some (if x > v then x else v))
```
is_some(FindMax.fn [])
={case}
is_some(FindMax.fn lst)

is_some(FindMax.fn lst)   *Right side
={case}
is_some(FindMax.fn [])
= {FindMax.fn definition}
```
    Match [] ->
    | [] -> None
    | x :: xs -> ( match fn xs with
                | None -> Some x
                | Some v -> Some (if x > v then x else v))
```
={apply match}
Is_some None
={is_some definition}
```
    Match None with
    | Some _ -> true
```

```
          | _ -> false
={apply match}
false
```

<u>Case: lst = h :: tl Case FindMax.tl = None</u>

```
Inductive hypothesis: FindMax.domain tl = is_some (FindMax.fn tl)

FindMax.domain lst
={case}
FindMax.domain h::tl
={FindMax.domain definition}
          Match h::tl with
                    | [] -> false
                    | _ -> true
={apply match}
True
={reverse from right side}
={apply match}
={is_some definition}
          Match (Some h) with
          | Some _ -> true
          | _ -> false
is_some(Some h)
={apply match}
is_some(match fn None with
                              | None -> Some h
                              | Some v -> Some (if h > v then h else v))
={case}
is_some(match fn tl with
                              | None -> Some h
                              | Some v -> Some (if h > v then h else v))
={apply match}
={FindMax.fn definition}
          Match h::tl with
                    | [] -> None
                    | h::tl -> (match fn tl with
                              | None -> Some h
                              | Some v -> Some (if h > v then h else v))
is_some(FindMax.fn h::tl)
={case}
is_some(FindMax.fn lst)

is_some(FindMax.fn lst)                    *right side
={case}
```

```
is_some(FindMax.fn h::tl)
={FindMax.fn definition}
        Match h::tl with
                | [] -> None
                | h::tl -> (match fn tl with
                        | None -> Some h
                        | Some v -> Some (if h > v then h else v))
={apply match}
is_some(match fn tl with
                        | None -> Some h
                        | Some v -> Some (if h > v then h else v))
={case}
is_some(match fn None with
                        | None -> Some h
                        | Some v -> Some (if h > v then h else v))
={apply match}
is_some(Some h)
={is_some definition}
        Match (Some h) with
        | Some _ -> true
        | _ -> false
={apply match}
True
```

Case: lst = h :: tl Case FindMax.tl = Some h

```
FindMax.domain lst
={case}
FindMax.domain h::tl
={FindMax.domain definition}
        Match h::tl with
                | [] -> false
                | _ -> true
={apply match}
True
={reverse from right side}
={apply match}
={is_some definition}
        Match (Some h) with
        | Some _ -> true
        | _ -> false
is_some(Some h)
={apply match}
is_some(match fn Some h with
                        | None -> Some h
```

```
                              | Some v -> Some (if h > v then h else v))
={case}
is_some(match fn tl with
                              | None -> Some h
                              | Some v -> Some (if h > v then h else v))
={apply match}
={FindMax.fn definition}
          Match h::tl with
                    | [] -> None
                    | h::tl -> (match fn tl with
                              | None -> Some h
                              | Some v -> Some (if h > v then h else v))
is_some(FindMax.fn h::tl)
={case}
is_some(FindMax.fn lst)

is_some(FindMax.fn lst)                    *right side
={case}
is_some(FindMax.fn h::tl)
={FindMax.fn definition}
          Match h::tl with
                    | [] -> None
                    | h::tl -> (match fn tl with
                              | None -> Some h
                              | Some v -> Some (if h > v then h else v))
={apply match}
is_some(match fn tl with
                              | None -> Some h
                              | Some v -> Some (if h > v then h else v))
={case}
is_some(match fn Some h with
                              | None -> Some h
                              | Some v -> Some (if h > v then h else v))
={apply match}
is_some(Some h)
={is_some definition}
          Match (Some h) with
          | Some _ -> true
          | _ -> false
={apply match}
True
```

P2. prove that get_somes (map lst) = solve lst (induction on the list)

Case lst = []

get_somes(map lst)
={case}
get_somes(map [])
={map definition}
    Match [] with
    | [] -> []
    | x :: xs -> F.fn x :: map xs
={apply match}
Get_somes []
={get_somes definition}
    Match [] with
    | [] -> []
    | Some x :: xs -> x :: get_somes xs
    | None :: xs -> get_somes xs
={apply match}
[]
={reversely match with right side}
={solve definition}
    Match [] with
    | [] -> []
    | x :: xs -> (match F.fn x with
        | None -> solve xs
        | Some v -> v :: solve xs
Solve []
={case}
Solve lst

Solve lst   right side
={case}
Solve []
={solve definition}
    Match [] with
    | [] -> []
    | x :: xs -> (match F.fn x with
        | None -> solve xs
        | Some v -> v :: solve xs
={apply match}
[]

Case lst = h :: tl Case (F.fn h) = None
Inductive Hypothesis: get_somes (map tl) = solve tl
Get_somes (map lst)
={case}
get_somes(map h::tl)
={map definition}
Get_somes match h::tl with
| [] -> []
| h :: tl -> F.fn h :: map tl
={apply match}
Get_somes F.fn h :: map tl
={get_somes definition}
Match (F.fn h :: map tl) with
| [] -> []
| Some (F.fn h) :: map tl -> F.fn h :: get_somes map tl
| None :: map tl -> get_somes map tl
={case}
Get_somes None :: map tl
={get_somes definition}
Match None :: map tl with
| [] -> []
| Some x :: map tl -> x :: get_somes map tl
| None :: map tl -> get_somes map tl
={apply match}
get_somes map tl
={Inductive Hypothesis}
solve tl

solve lst right side
={case}
Solve h :: tl
={solve definition}
Match h :: tl with
| [] -> []
| h :: tl -> (match F.fn h with
| None -> solve tl
| Some v -> v :: solve tl
={apply match}
match F.fn h with
| None -> solve tl

| Some v -> v :: solve tl

Case lst = h :: tl Case (F.fn h) = Some h
Inductive Hypothesis: get_somes (map tl) = solve tl
Get_somes (map lst)
={case}
get_somes(map h::tl)
={map definition}
Get_somes match h::tl with
| [] -> []
| h :: tl -> F.fn h :: map tl
={apply match}
Get_somes F.fn h :: map tl
={case}
Get_somes Some h :: map tl
={get_somes definition}
Match (Some h :: map tl) with
| [] -> []
| Some h :: map tl -> h :: get_somes map tl
| None :: map tl -> get_somes map tl
={apply match}
h :: get_somes map tl
={Inductive Hypothesis}
H :: solve tl
={reversely match with right side}
={apply match}
Case (F.fn h) = Some h
match F.fn h with
| None -> solve tl
| Some v -> v :: solve tl
={apply match}
={solve definition}
Match h :: tl with
| [] -> []
| h :: tl -> (match F.fn h with
| None -> solve tl
| Some v -> v :: solve tl
Solve h :: tl
={case}
solve lst

P3. prove that solve (filter lst) = solve lst
(induction on the list, requires properties of OptionFunction)

Case lst = []

Solve (filter lst)
={case}
Solve (filter [])
={filter definition}
Solve match [] with
```
        | [] -> []
        | h :: xs -> if F.domain h then h :: filter tl else filter tl
```
={apply match}
Solve []
={solve definition}
```
        Match [] with
        | [] -> []
        | h :: tl -> (match F.fn h with
                        | None -> solve tl
                        | Some v -> v :: solve tl
```
={apply match}
[]
={reversely match with right side}
={solve definition}
```
        Match [] with
        | [] -> []
        | h :: tl -> (match F.fn h with
                        | None -> solve tl
                        | Some v -> v :: solve tl
```
Solve []
={case}
Solve lst

Solve lst *right side
={case}
Solve []
={solve definition}
```
        Match [] with
        | [] -> []
        | h :: tl -> (match F.fn h with
                        | None -> solve tl
                        | Some v -> v :: solve tl
```

={apply match}
[]

Case lst = h :: tl Case: F.fn h = None
Inductive Hypothesis: solve (filter tl) = solve tl
Solve (filter lst)
={case}
Solve (filter h:: tl)
={filter definition}
Solve match h:: tl with
          | [] -> []
          | h :: tl -> if F.domain h then h :: filter tl else filter tl
={apply match}
Solve (if F.domain h then h :: filter tl else filter tl) * previously stuck here
={domain property}
Solve (if is_some(F.fn h) then h:: filter tl else filter tl)
={case}
Solve (if is_some None then h:: filter tl else filter tl)
={is_some definition}
          Match None with
                    | Some _ -> true
                    | _ -> false
={apply match}
Solve(if false then h:: filter tl else filter tl)
={apply conditional}
Solve filter tl
={inductive hypothesis}
Solve tl
={reverse from right side}
={case}
 (match F.fn h with
                    | None -> solve tl
                    | Some v -> v :: solve tl)
={apply match}
Match h:: tl with
          | [] -> []
          | h :: tl -> (match F.fn h with
                    | None -> solve tl
                    | Some v -> v :: solve tl)
={solve definition}
Solve h::tl
Solve lst

```
Solve lst                    * right side
={case}
Solve h::tl
={solve definition}
Match h:: tl with
          | [] -> []
          | h :: tl -> (match F.fn h with
                    | None -> solve tl
                    | Some v -> v :: solve tl)
={apply match}
 (match F.fn h with
                    | None -> solve tl
                    | Some v -> v :: solve tl)
={case}
Solve tl
```

Case lst = h :: tl Case: F.fn h = Some h

Inductive Hypothesis: solve (filter tl) = solve tl

```
Solve ( filter lst)
={case}
Solve (filter h::tl)
={filter definition}
Solve match h::tl with
          | [] -> []
          | h :: tl -> if F.domain h then h :: filter tl else filter tl
={apply match}
Solve ( if F.domain h then h :: filter tl else filter tl)
={domain property}
solve( if is_some(FindMax.fn h) then h:: filter tl else filter tl)
={case}
Solve ( if is_some Some h) then h:: filter tl else filter tl
={is_some definition}
          Match Some h with
                    | Some _ -> true
                    | _ -> false
={apply match}
Solve (if true then h::filter tl else filter tl)
={apply conditional}
Solve h :: filter tl
={solve definition}
```

```
        Match h:: filter tl with
        | [] -> []
        | h :: filter tl -> (match F.fn h with
                        | None -> solve filter tl
                        | Some v -> v :: solve (filter tl)
={apply match}
 (match F.fn h with
                        | None -> solve filter tl
                        | Some v -> v :: solve (filter tl)
={case}
Match Some h with
        | None -> solve filter tl
        | Some h -> h :: solve (filter tl)
={apply match}
H :: solve (filter tl)
 ={Inductive Hypothesis}
H:: Solve tl
={reverse from the right side}
match Some h with
                                | None -> solve tl
                                | Some h -> h :: solve tl)
={case}
(match F.fn h with
                                | None -> solve tl
                                | Some h -> h :: solve tl)
={apply match}
={solve definition}
        Match h:: tl with
                | [] -> []
                | h :: tl -> (match F.fn h with
                                | None -> solve tl
                                | Some h -> h :: solve tl)
Solve h::tl
={case}
Solve lst




Solve lst                   *right side
={case}
Solve h::tl
={solve definition}
```

Match h:: tl with
| [] -> []
| h :: tl -> (match F.fn h with
| None -> solve tl
| Some h -> h :: solve tl)
={apply match}
(match F.fn h with
| None -> solve tl
| Some h -> h :: solve tl)
={case}
match Some h with
| None -> solve tl
| Some h -> h :: solve tl)
h:: solve tl

P4. prove that length (solve lst) = length (filter lst) (induction on the list)
Case lst = []
Length (solve lst)
={case}
Length (solve [])
={solve definition}
Length match [] with
| [] -> []
| x :: xs -> (match F.fn x with
| None -> solve xs
| Some v -> v :: solve xs
={apply match}
Length []
={length definition}
Match [] with
| [] ->0
| _ :: xs -> 1 + length xs
={apply match}
0
={ reversely match}
={length definition}
Match [] with
| [] -> 0
| _ :: xs -> 1 + length xs
Length []
={apply match}
Length match [] with

| [] -> []
| x :: xs -> if F.domain x then x:: filter xs else filter xs
={filter definition}
Length (filter [])
={case}
Length (filter lst)

Length (filter lst)   right side
={case}
Length (filter [])
={filter definition}
Length match [] with
| [] -> []
| x :: xs -> if F.domain x then x:: filter xs else filter xs
={apply match}
Length []
={length definition}
Match [] with
| [] -> 0
| _ :: xs -> 1 + length xs
={apply match}
0

Case lst = h :: tl fn h = None
Inductive Hypothesis: length (solve tl) = length (filter tl)

Length (solve lst)
={case}
Length (solve h::tl)
={solve definition}
Match h::tl with
| [] -> []
| h :: tl -> (match F.fn h with
| None -> solve tl
| Some v -> v :: solve tl)
={case}
Match h::tl with
| [] -> []
| h :: tl -> (match None with
| None -> solve tl
| Some v -> v :: solve tl)
={apply match)
Match h::tl with
| [] -> []

| h :: tl -> solve tl

={apply match}

Length solve tl

={Inductive Hypothesis}

Length (filter tl)

<u>Case lst = h :: tl fn h = Some h</u>

Inductive Hypothesis: length (solve tl) = length (filter tl)

Length (solve lst)

={case}

Length (solve h::tl)

={solve definition}

```
          Match h::tl with
                    | [] -> []
                    | h::tl (match fn h with
                              | None -> Some tl
                              | Some v -> v :: solve tl
```

={case}

```
          Match h::tl with
                    | [] -> []
                    | h::tl (match (Some h)with
                              | None -> Some tl
                              | Some v -> v :: solve tl
```

={apply match}

```
Match h::tl with
                    | [] -> []
                    | h::tl -> solve tl
```

={apply match}

Length solve tl

={Inductive hypothesis}

Length filter tl