FPV Tutorübung

Woche 12

Equational Reasoning

Manuel Lerchner

12.07.2023

Quiz

Courses > Funktionale Programmierung und Verifikation (Sommersemester 2023) > Exercises > Week 12 Quiz

Week 12 Quiz

Points: 20

Open quiz

Exercise details

Release date:	Jul 10, 2023 08:00
Submission due:	Jul 14, 2023 20:00
Complaint possible:	Yes

Passwort:

T01: 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 $n \in \mathbb{N}_0$.

Format

Write your answer as plain text. For all equational proofs that show the equivalence of two MiniOCaml expressions, annotate each step as follows:

```
             e_1
(rule 1)  =  e_2
(rule 2)  =  e_3
       ...
(rule n)  =  e_n
```

For each step, when you:

- apply the definition of a function `f`, **rule** must be `f`
- apply the rule for function application, **rule** must be `fun`
- apply an induction hypothesis, **rule** must be `I.H.`
- simplify an arithmetic expression, **rule** must be `arith`
- select a branch in a match expression, **rule** must be `match`
- expand a `let` defintion, **rule** must be `let`
- apply a lemma that you have already proven in the exercise, **rule** must be the name you gave to the lemma

In each step, apply only a single rule. Write each step on its own line.

Template

```
To prove:
            fact_iter n = fact n

Adaptation:
            fact_aux 1 n = fact n


Proof by Induction on n

Base: n = 0

            fact_aux 1 0
(rules)     = < … >
            = fact 0



Hypothesis: (Does it hold?)
            fact_aux 1 n = fact n

Step:

            fact_aux 1 (n+1)
(rules)     = < … >
            = fact (n+1)
```

```ocaml
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 $n \in \mathbb{N}_0$.

Tipp: This scheme has has a flaw!
(Try to generalize!)

T02: Arithmetic 101

```
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 = match i <= 0 with
  | true -> a
  | false -> mul (i - 1) j (j + a)
```

Prove that, under the assumption that all expressions terminate, for any `l` and $c \geq 0$ it holds that:

```
mul c (sum l 0) 0 = c * summa l
```

Template

```
To prove:
            mul c (sum l 0) 0 = c * summa l

Generalization:
*           mul c (sum l acc1) acc2 = acc2 + c * (acc1 + summa l)


================== Lemma 1 =================================

Lemma 1:
        sum l acc1 = acc1 + summa l

Proof of * by Induction on l

Base: l = []

            sum [] acc1
(rules)     = <....>
            = acc1 + summa []

Hypothesis:
            sum l acc1 = acc1 + summa l

Step:

            sum (x :: xs) acc1
(rules)     = <....>
            = acc1 + summa (x :: xs)

================== End Lemma 1 =============================
```

```
Proof of initial goal by Induction on c:
          To Proof:  mul c (sum l acc1) acc2 = acc2 + c * (acc1 + summa l)

Base: c = 0

            mul 0 (sum l acc1) acc2
(rules)     = < ... >
            = acc2 + 0 * (acc1 + summa l)


Hypothesis: (Does it hold?)
            mul c (sum l acc1) acc2 = acc2 + c * (acc1 + summa l)

Step:

            mul (c + 1) (sum l acc1) acc2
(rules)     = < ... >
            = acc2 + (c + 1) * (acc1 + summa l)
```

T03: Counting Nodes

```
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 (a+1))
  in
  aux t 0
```

Prove or disprove the following statement for arbitary trees `t`:

```
nodes t = count t
```

Template

```
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 (a+1))
  in
  aux t 0
```

Prove or disprove the following statement for arbitary trees `t`:

```
nodes t = count t
```

```
To prove:
|  |  |  |  |  |  nodes t = count t

Adaptation:
|  |  |  |  |  |  nodes t = aux t 0

Generalization:
|  |  |  |  |  |  acc + nodes t = aux t acc


Proof of the generalization (by induction on t):

Base: t = Empty
|
|  |  |  |  |  |  acc + nodes Empty
(rules)           = < ... >
|  |  |  |  |  |  = aux Empty acc


Hypothesis: (Does it hold?)
|  |  |  |  |  |  acc + nodes t = aux t acc

Step:
|
|  |  |  |  |  |  acc + nodes (Node (a,b))
(rules)           = < ... >
|  |  |  |  |  |  = aux (Node (a,b)) acc
```