Week12

IN0003

Jigao

TUM

24. Januar 2019

What about the last week?

- The first stage to Ocaml is gone.
- Last time we had introduced a proving tool **Big Step**
- Today we have induction prooving, which we learned in other lecture.(Math maybe)
- We also use the Ocaml syntax for Big Step. So recrusion.
- Do you have the lectures? They are helpful for this topic.

Statement to this Slide

- This slide is aimed at the presentation in my tutorial.
- Can be some style, which not 100% same of the orginal soltion and the lecture slide.
- **If there are some mismatch with the orginal solution, please following the style in orginal solution**
- And tell me where is wrong.

Induction

- Lecture part **verification**.
- A tool to find the equality of two recrusive functions.
- Because we could use the induction steps to simply the recrusion.
- They are both done step by step. Right?

Assignment 12.1

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

Assignment 12.1

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

- Question: Which is the end recursion function?

Assignment 12.1

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

- Question: Which is the end recursion function?
- Pay attation to this function and it's accumulator.

Assignment 12.1

Intuitive?
I would try to prove that
`fact_iter n = fact n` $\equiv$ `fact_aux 1 n = fact n` by induction on `n`.
Hope we are familiar with **Induction proving steps**.

- **Base case**: `n = 0` ...
- **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 $n + 1$: ...

Assignment 12.1 : Base Case

- Base case: `n = 0`

```
fact_aux 1 0
     =
    ...
     =
  fact 0
```

- Try to simply them to a same thing, then we can say thay are equal.
- Two direction. One goes to the bottom, one goes to top.

Assignment 12.1 : Base Case, First try

- Base case: `n = 0`

$$
\begin{aligned}
& \texttt{fact\_aux 1 0} \\
\overset{\texttt{f\_a}}{=} & \\
\overset{\texttt{fact}}{=} & \ \texttt{fact 0}
\end{aligned}
$$

Assignment 12.1 : Base Case, First try

- Base case: `n = 0`

```
      fact_aux 1 0
f_a
 =    match 0 with 0 -> acc | n -> fact_aux (n*acc) (n-1)
   ...
match
 =    match 0 with 0 -> 1 | n -> n * fact (n-1)
fact
 =    fact 0
```

Assignment 12.1 : Base Case, First try

- Base case: `n = 0`

$$
\begin{aligned}
& \texttt{fact\_aux 1 0} \\
\overset{\texttt{f\_a}}{=}\ & \texttt{match 0 with 0 -> acc | n -> fact\_aux (n*acc) (n-1)} \\
\overset{\texttt{match}}{=}\ & \texttt{1} \\
\overset{\texttt{match}}{=}\ & \texttt{match 0 with 0 -> 1 | n -> n * fact (n-1)} \\
\overset{\texttt{fact}}{=}\ & \texttt{fact 0}
\end{aligned}
$$

Assignment 12.1 : Inductive step, First try

- 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 $n + 1$:

```
fact_aux 1 (n+1)
```

$$\stackrel{\texttt{fact}}{=} \texttt{fact (n+1)}$$

Assignment 12.1 : Inductive step, First try

- 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 $n + 1$:

```
      fact_aux 1 (n+1)
f_a
 =    match n+1 with 0 -> 1 | n -> fact_aux (n*1) (n-1)
match
 =    match n+1 with 0 -> 1 | n -> n * fact (n-1)
fact
 =    fact (n+1)
```

Assignment 12.1 : Inductive step, First try

- 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 $n+1$:

```
        fact_aux 1 (n+1)
 f_a
  =     match n+1 with 0 -> 1 | n -> fact_aux (n*1) (n-1)
match
  =     fact_aux ((n+1)*1) ((n+1)-1)
```

$$\overset{arith}{=} (n+1) * fact((n+1)-1)$$

```
match
  =     match n+1 with 0 -> 1 | n -> n * fact (n-1)
fact
  =     fact (n+1)
```

Assignment 12.1 : Inductive step, First try

- 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 $n+1$:

```
      fact_aux 1 (n+1)
f_a
 =    match n+1 with 0 -> 1 | n -> fact_aux (n*1) (n-1)
match
 =    fact_aux ((n+1)*1) ((n+1)-1)
arith
 =    fact_aux (n+1) n


 =    (n+1) * fact n
```

$$\stackrel{\textit{arith}}{=} (n+1) * fact((n+1)-1)$$

```
match
 =    match n+1 with 0 -> 1 | n -> n * fact (n-1)
fact
 =    fact (n+1)
```

Assignment 12.1 : Inductive step, First try

- 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 $n + 1$:

```
        fact_aux 1 (n+1)
  f_a
   =    match n+1 with 0 -> 1 | n -> fact_aux (n*1) (n-1)
 match
   =    fact_aux ((n+1)*1) ((n+1)-1)
 arith
   =    fact_aux (n+1) n
```

our proof fails here

```
   =    (n+1) * fact n
 arith
   =    (n + 1) * fact((n + 1) - 1)
 match
   =    match n+1 with 0 -> 1 | n -> n * fact (n-1)
 fact
   =    fact (n+1)
```

Assignment 12.1 : Inductive step, First try

- Our first proof fails
- WHY?

Assignment 12.1 : Inductive step, First try

- Our first proof fails
- WHY?
- Because we cannot use the induction hypothesis (induction assuption, Induktionsanahme) in induction step .
- The reason is that **our hypothesis holds only for the special case** where `x` is exactly for `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`**.

Assignment 12.1 : Inductive step, First try

- 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.
- **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
```

Time for you to exercise.

Assignment 12.1 : Base Case

- Base case: `n = 0`

```
fact_aux acc 0
```

$$\stackrel{\texttt{fact}}{=} \texttt{acc * fact 0}$$

Assignment 12.1 : Base Case

- Base case: `n = 0`

```
        fact_aux acc 0
  f_a
   =    match 0 with 0 -> acc | n -> fact_aux (n*acc) (n-1)

 match
   =    acc * match 0 with 0 -> 1 | n -> n * fact (n-1)
 fact
   =    acc * fact 0
```

Assignment 12.1 : Base Case

- Base case: `n = 0`

$$
\begin{aligned}
& \texttt{fact\_aux acc 0} \\
\overset{\texttt{f\_a}}{=}\ & \texttt{match 0 with 0 -> acc | n -> fact\_aux (n*acc) (n-1)} \\
\overset{\texttt{match}}{=}\ & \texttt{acc} \\
\overset{arith}{=}\ & \texttt{acc * 1} \\
\overset{\texttt{match}}{=}\ & \texttt{acc * match 0 with 0 -> 1 | n -> n * fact (n-1)} \\
\overset{\texttt{fact}}{=}\ & \texttt{acc * fact 0}
\end{aligned}
$$

Assignment 12.1 : Inductive step

- 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)`

$\overset{\texttt{fact}}{=}$ `acc * fact (n+1)`

Assignment 12.1 : Inductive step

- 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
 =   acc * match n+1 with 0 -> 1 | n -> n * fact (n-1)
fact
 =   acc * fact (n+1)
```

Assignment 12.1 : Inductive step

- 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:

$$
\begin{aligned}
& \texttt{fact\_aux acc (n+1)} \\
\overset{\texttt{f\_a}}{=}\ & \texttt{match n+1 with 0 -> acc | n -> fact\_aux (n*acc) (n-1)} \\
\overset{\texttt{match}}{=}\ & \texttt{fact\_aux ((n+1)*acc) ((n+1)-1)} \\
\\
\overset{\textit{arith}}{=}\ & \texttt{acc * (n+1) * fact ((n+1)-1)} \\
\overset{\texttt{match}}{=}\ & \texttt{acc * match n+1 with 0 -> 1 | n -> n * fact (n-1)} \\
\overset{\texttt{fact}}{=}\ & \texttt{acc * fact (n+1)}
\end{aligned}
$$

Assignment 12.1 : Inductive step

- 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:

$$
\begin{aligned}
& \texttt{fact\_aux acc (n+1)} \\
\overset{\texttt{f\_a}}{=}\ & \texttt{match n+1 with 0 -> acc | n -> fact\_aux (n*acc) (n-1)} \\
\overset{\texttt{match}}{=}\ & \texttt{fact\_aux ((n+1)*acc) ((n+1)-1)} \\
\overset{arith}{=}\ & \texttt{fact\_aux ((n+1)*acc) n} \\
\overset{I.H.}{=}\ & \texttt{(n+1) * acc * fact n} \\
\overset{arith}{=}\ & \texttt{acc * (n+1) * fact ((n+1)-1)} \\
\overset{\texttt{match}}{=}\ & \texttt{acc * match n+1 with 0 -> 1 | n -> n * fact (n-1)} \\
\overset{\texttt{fact}}{=}\ & \texttt{acc * fact (n+1)}
\end{aligned}
$$

- *I.H.* stands for induction hypothesis.

Assignment 12.2

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
```

Hint: Which one is end recrusion function and how can we handle the accumulator?

Little exercise: find the generalized proving claim.

Assignment 12.2

Both `sum` and `mul` use an accumulator in their tail recursive implementation.

- `sum l 0` has 0 as accumulator → `sum l acc1`
- Right side: `summa l` → `acc1 + summa l`
- `mul c x 0` has 0 as accumulator → `mul c x acc2`
- Combine them together `mul c (sum l acc1) acc2`
- Right side: `acc2 + c * (acc1 + summa l)`

Thus, we have to generalize the claim to:

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

instead of the orginal one:

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

Assignment 12.2

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 = []`
- Time for exercise. 5 Min

Assignment 12.2

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 = []`
- Time for exercise. 5 Min

```
         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 []
```

Assignment 12.2

First we prove a lemma by induction on the length n of the list `l`:

Lemma 1: `sum l acc1 = acc1 + summa l`

- 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 $n + 1$:
- Time for exercise. 8 Min

Assignment 12.2

First we prove a lemma by induction on the length n of the list `l`:

Lemma 1: `sum l acc1 = acc1 + summa l`

- 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 $n + 1$:

$$\texttt{sum (x::xs) acc1}$$

$$\stackrel{\texttt{summa}}{=} \texttt{acc1 + summa (x::xs)}$$

Assignment 12.2

First we prove a lemma by induction on the length n of the list `l`:

Lemma 1: `sum l acc1 = acc1 + summa l`

- 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 $n + 1$:

```
        sum (x::xs) acc1
  sum
   =    match x::xs with [] -> acc1 | h::t -> sum t (h+acc1)

 match
   =    acc1 + match x::xs with [] -> 0 | h::t -> h + summa t
 summa
   =    acc1 + summa (x::xs)
```

Assignment 12.2

First we prove a lemma by induction on the length n of the list `l`:

Lemma 1: `sum l acc1 = acc1 + summa l`

- 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 $n + 1$:

```
         sum (x::xs) acc1
  sum
   =     match x::xs with [] -> acc1 | h::t -> sum t (h+acc1)
 match
   =     sum xs (x+acc1)


 comm
   =     acc1 + x + summa xs
 match
   =     acc1 + match x::xs with [] -> 0 | h::t -> h + summa t
 summa
   =     acc1 + summa (x::xs)
```

Assignment 12.2

First we prove a lemma by induction on the length n of the list `l`:

Lemma 1: `sum l acc1 = acc1 + summa l`

- 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 $n + 1$:

```
      sum (x::xs) acc1
sum
 =    match x::xs with [] -> acc1 | h::t -> sum t (h+acc1)
match
 =    sum xs (x+acc1)
I.H.
 =    x + acc1 + summa xs
comm
 =    acc1 + x + summa xs
match
 =    acc1 + match x::xs with [] -> 0 | h::t -> h + summa t
summa
 =    acc1 + summa (x::xs)
```

Assignment 12.2

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

- Base case: `c = 0`

```
     mul 0 (sum l acc1) acc2)
mul
 =   if 0 <= 0 then acc2 else mul (0-1) (sum l acc1) ((sum
if
 =   acc2
arith
 =   acc2 + 0 * (acc1 + summa l)
```

Assignment 12.2

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

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

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

Assignment 12.2

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

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

```
       mul (c+1) (sum l acc1) acc2)
 mul
  =    if c+1 <= 0 then acc2 else mul c (sum l acc1) ((sum l
 if
  =    mul c (sum l acc1) ((sum l acc1) + acc2)
 I.H.
  =    (sum l acc1) + acc2 + c * (acc1 + summa l)
 comm
  =    acc2 + c * (acc1 + summa l) + (sum l acc1)
 L.1
  =    acc2 + c * (acc1 + summa l) + (acc1 + summa l)
 Distr
  =    acc2 + (c+1) * (acc1 + summa l)
```

This proves the statement. $\square$

Assignment 12.3

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
    | 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 (acc+1))
  in
  aux t 0
```

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

```
nodes t = count t
```

Hint: Which one is end recrusion function and how can we handle the accumulator?

Assignment 12.3: Base case

First, we show that `nodes t = aux t 0` or, more precisely, the generalized statement `acc + nodes t = aux t acc` holds.
We prove by induction on the structure of trees:

- Base case: `t = Empty`

$$\texttt{acc + nodes Empty}$$

$$\stackrel{\texttt{aux}}{=} \texttt{aux Empty acc}$$

Assignment 12.3: Base case

- Base case: `t = Empty`

```
        acc + nodes Empty
nodes
  =     acc + match Empty with Empty -> 0
                | Node (l,r) -> 1 + (nodes l) + (nodes r)

match
  =     match Empty with Empty -> acc
                | Node (l,r) -> aux r (aux l (acc+1))
 aux
  =     aux Empty acc
```

Assignment 12.3: Base case

- Base case: `t = Empty`

```
        acc + nodes Empty
nodes
  =     acc + match Empty with Empty -> 0
                 | Node (l,r) -> 1 + (nodes l) + (nodes r)
match
  =     acc
match
  =     match Empty with Empty -> acc
                 | Node (l,r) -> aux r (aux l (acc+1))
 aux
  =     aux Empty acc
```

Assignment 12.3: Induction Step

- Inductive step: Assume the above equivalence holds for two trees `a` and `b`. Now, we show that it then also holds for a tree `Node (a, b)`:
- Time to exercise.

Assignment 12.3: Induction Step

- Inductive step: Assume the above equivalence holds for two trees `a` and `b`. Now, we show that it then also holds for a tree `Node (a, b)`:
- Time to exercise.

```
acc + nodes (Node (a,b))
```

$\overset{\texttt{aux}}{=}$ `aux (Node (a,b)) acc`

Assignment 12.3: Induction Step

- Inductive step: Assume the above equivalence holds for two trees `a` and `b`. Now, we show that it then also holds for a tree `Node (a, b)`:

$$
\begin{aligned}
& \texttt{acc + nodes (Node (a,b))} \\
\overset{\texttt{nodes}}{=}\ & \texttt{acc + match Node (a,b) with Empty -> 0} \\
& \qquad\quad \texttt{| Node (l,r) -> 1 + (nodes l) + (nodes r)} \\
& \\
\overset{\texttt{match}}{=}\ & \texttt{match Node (a,b) with Empty -> acc | Node (l,r) -> aux} \\
\overset{\texttt{aux}}{=}\ & \texttt{aux (Node (a,b)) acc}
\end{aligned}
$$

Assignment 12.3: Induction Step

- Inductive step: Assume the above equivalence holds for two trees `a` and `b`. Now, we show that it then also holds for a tree `Node (a, b)`:

```
      acc + nodes (Node (a,b))
nodes
  =   acc + match Node (a,b) with Empty -> 0
          | Node (l,r) -> 1 + (nodes l) + (nodes r)
match
  =   acc + 1 + (nodes a) + (nodes b)

I.H.
  =   aux b (aux a (acc+1))
match
  =   match Node (a,b) with Empty -> acc | Node (l,r) -> aux
aux
  =   aux (Node (a,b)) acc
```

Assignment 12.3: Induction Step

- Inductive step: Assume the above equivalence holds for two trees `a` and `b`. Now, we show that it then also holds for a tree `Node (a, b)`:

```
         acc + nodes (Node (a,b))
nodes
  =      acc + match Node (a,b) with Empty -> 0
               | Node (l,r) -> 1 + (nodes l) + (nodes r)
match
  =      acc + 1 + (nodes a) + (nodes b)
   I.H.
    =    aux b (acc + 1 + nodes a)
   I.H.
    =    aux b (aux a (acc+1))
match
  =      match Node (a,b) with Empty -> acc | Node (l,r) -> aux
 aux
  =      aux (Node (a,b)) acc
```

Summary

- The Induction workaround:
- Judge the function: end recrusion?
- Re-form to proving claim abd I.H with *acc* at the end recrusion case.
- Base case and Induction step.
- We start from top and bottom to try to get the same expression.
- More to Induction: refer to the exercise 13 from last year.
- This time we had induction on numbers(12.1), list(12.2), tree(12.3).

Summary

- The Induction like Big Step is just basic work around as a proving tool.
- **Time** is ultimate factor in the exam.
- If you master the time with Induction and Big Step, then this part is 100% easy for you.
- They are almost always same work around and if you get all the exercise (or the old exercise) then you can alreay to slove the exam proving part.
- In exam, then you save time to consider more with Ocaml.
- Any Questions?