Script generated by TTT

Title: Seidl: Functional Programming and Verification (16.11.2018)

Date: Fri Nov 16 08:30:08 CET 2018

Duration: 90:10 min

Pages: 7

2.6 Definition of Functions

```
# let double x = 2*x;;
val double : int -> int = <fun>
# (double 3, double (double 1));;
- : int * int = (6,4)
```

→ Behind the function name follow the parameters.

→ The function name is just a variable whose value is a function.

2.6 Definition of Functions

```
# let double x = 2*x;;
val double : int -> int = <fun>
# (double 3, double (double 1));;
- : int * int = (6,4)
```

→ Behind the function name follow the parameters.

→ The function name is just a variable whose value is a function.

Recursive Functions

A function is recursive, if it calls itself (directly or indirectly).

```
# let rec fac n = if n<2 then 1 else n * fac (n-1);;
val fac : int -> int = <fun>
# let rec fib = fun x -> if x <= 1 then 1
                         else fib (x-1) + fib (x-2);;
val fib : int -> int = <fun>
```

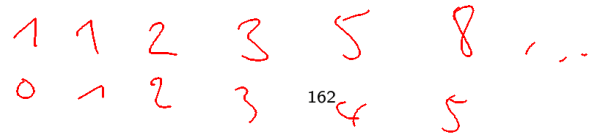
For that purpose, Ocaml offers the keyword `rec`.

Recursive Functions

A function is recursive, if it calls itself (directly or indirectly).

```
# let rec fac n = if n<2 then 1 else n * fac (n-1);;
val fac : int -> int = <fun>
# let rec fib = fun x -> if x <= 1 then 1
                         else fib (x-1) + fib (x-2);;
val fib : int -> int = <fun>
```

For that purpose, Ocaml offers the keyword `rec`.



If functions call themselves indirectly via other other functions, they are called mutually recursive.

```
# let rec even n = if n=0 then "even" else odd (n-1)
      and odd  n = if n=0 then "odd"  else even (n-1);;
val even : int -> string = <fun>
val odd : int -> string = <fun>
```

We combine their definitions by means of the keyword `and`.

If functions call themselves indirectly via other other functions, they are called mutually recursive.

```
# let rec even n = if n=0 then "even" else odd (n-1)
      and odd  n = if n=0 then "odd"  else even (n-1);;
val even : int -> string = <fun>
val odd : int -> string = <fun>
```

We combine their definitions by means of the keyword `and`.