

Objective Caml version 3.09.0

```

(* ===== *)
(* Übung 1, Aufgabe 3: *)
(* ===== *)

(* ----- *)
(* In der Vorlesung hatten wir schon die Addition: *)
(* ----- *)

# let church_add m n =
  fun f -> fun x -> (m f) (n f x);;

val church_add : ('a -> 'b -> 'c) -> ('a -> 'd -> 'b) -> 'a -> 'd -> 'c =
  <fun>

(* ----- *)
(* Multiplikation: *)
(* Es gilt  $(m * n) f = f^{(m*n)} = (f^m)^n = n (f^m) = n (m f)$ , also: *)
(* ----- *)

# let church_mul m n =
  fun f -> n (m f);;

val church_mul : ('a -> 'b) -> ('b -> 'c) -> 'a -> 'c = <fun>

(* ----- *)
(* Potenzieren: *)
(* Für alle f gilt  $(m^n) f = f^{(m^n)} = f^{(m * \dots * m)} = (\dots(f^m)\dots)^m = m (\dots (m f) \dots)$ . *)
(* Also ist  $m^n = m \circ \dots \circ m = n m$ . *)
(* ----- *)

# let church_exp m n = n m;;

val church_exp : 'a -> ('a -> 'b) -> 'b = <fun>

(* ----- *)
(* Zum Testen dieser Operationen definiert man sich am besten Funktionen nat_to_church und *)
(* church_to_nat, die natürliche Zahlen in Church numerals umwandeln und umgekehrt. *)
(* ----- *)

(* ----- *)
(* nat_to_church erhält man aus der Definition von  $f^n$ : *)
(* *)
(*  $f^0 = \text{lambda } x. x$  *)
(*  $f^n = f \circ f^{(n-1)}$  falls  $n > 0$  *)
(* ----- *)

# let rec nat_to_church n =
  if n = 0
  then fun f -> fun x -> x
  else fun f -> fun x -> f (nat_to_church (n-1) f x);;

val nat_to_church : int -> ('a -> 'a) -> 'a -> 'a = <fun>

(* ----- *)
(* church_to_nat erhält man aus der Überlegung:  $n \text{ succ } 0 = \text{succ}^n 0 = n$  *)
(* ----- *)

# let rec church_to_nat n =
  n (fun x -> x + 1) 0;;

val church_to_nat : ((int -> int) -> int -> 'a) -> 'a = <fun>

(* ----- *)
(* So kann man jetzt mit Church numerals rechnen: *)
(* ----- *)

# church_to_nat (church_add (nat_to_church 3) (nat_to_church 4));;

- : int = 7

```

```
# church_to_nat (church_mul (nat_to_church 3) (nat_to_church 4));;
- : int = 12
# church_to_nat (church_exp (nat_to_church 3) (nat_to_church 4));;
- : int = 81
#
```