Funktionsrekelt-count : 11

Zitat des Tages : "Sind des Festes und seine Sänger."
- Buche der Prediger

Frohe Weihnachten

Info Vorlesung 18.12.2018

repeat (repeat) n-fache Komposition einer Funktion f mit sich selbst

(Exponentiation):

$f^0 = id$ (id: (lambda (x)))

$f^n = f^{n-1} \cdot f$ (by Ka:)

```
(: repeat (natural (% a -> % a) -> (% a -> % a)))
(define repeat
  (lambda (n f)
    (cond ((= n 0) (lambda (x)))
          ((> n 0) (compose repeat (- n 1) f) f)))))
```

; Greife auf das n-te Element von xs zu
; (n > 0)

```
(: nth (natural (list-of % a) -> % a))
(define nth
  (lambda (n xs)
    ((compose first (repeat (- n 1) rest)) xs)))
```

(list-of % a) -> (list-of % a)

first: (list-of % a) -> % a

(repeat (- n 1) rest): (list-of % a) -> (list-of % a)

(compose first (repeat (- n 1) rest)): (list-of % a) -> % a

((list-of % a) -> % a)

Reduktion:

((add 1) 47)

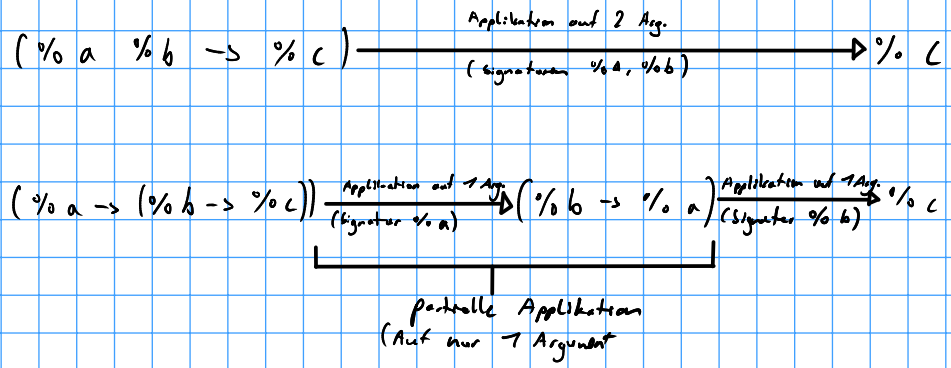
eval: id → (((lambda (x) (lambda (y) (+ x y))) 1) 47)

apply λ(x) → ((lambda (y) (+ 1 y)) 47)

Funktion die 1 auf ihr Argument addiert

apply λ(y) → (+ 1 47)

Currying



- currying (Haskell B. Curry, Moses Schönfinkel)

  Anwendung einer Funktion auf ihr erstes Argument liefert eine Funktion der restlichen Argumente

- jede n-stellige Funktion lässt sich in eine alternative curried Funktion transformieren, die n in Schritten jeweils über 1 Argument konsumiert: curry

  ungelerte Funktion: uncurry

```
(: curry ((% a % b -> % c) -> (% a -> (% b -> % c))))
(define curry
  (lambda (f)
    (lambda (x)
      (lambda (y)
        (f x y)))))

(: uncurry ((% a -> (% b -> % c)) -> (% a % b -> % c)))
(define uncurry
  (lambda (f)
    (lambda (x y) ((f x) y))))
```