

Programmierung 1

Vorlesung 5

Livestream beginnt um 14:15 Uhr

Höherstufige Prozeduren, Teil 2

Programmierung 1

Kaskadierte Prozeduren

Prozeduren die **Prozeduren als Ergebnis** liefern, werden als **kaskadiert** bezeichnet.

Beispiel:

```
fun mul (x:int) = fn (y:int) => x*y
```

```
val mul : int → (int → int)
```

```
mul 7
```

```
fn : int → int
```

```
it 3
```

```
21 : int
```

```
mul 7 5
```

```
35 : int
```

Kaskadierte Prozedurdeklarationen

```
fun f (x:int) (y:int) (z:int) = x+y+z
```

```
val f: int → int → int → int
```

beschreibt dieselbe Prozedur wie

```
fun f (x:int) = fn (y:int) => fn (z:int) => x+y+z
```

► kartesische Darstellung:

```
fun mul (x:int, y:int) = x*y
```

```
val mul: int * int → int
```

► kaskadierte Darstellung (auch: “curried procedure”)

```
fun mul (x:int) = fn (y:int) => x*y
```

```
val mul: int → (int → int)
```



Moses Isajewitsch Schönfinkel
(1889 – 1942)



Haskell Brooks Curry
(1900 – 1982)

Höherstufige Prozeduren

Eine **Prozedur** heißt **höherstufig**,
wenn eines ihrer **Argumente** eine **Prozedur** ist.

Beispiel: $sum : (int \rightarrow int) \rightarrow int \rightarrow int$
 $sum\ f\ n = 0 + f\ 1 \cdots + f\ n$

```
fun sum (f:int->int) (n:int) : int =  
  if n<1 then 0 else sum f (n-1) + f n
```

- ▶ Summe der Zahlen von 1 bis 100

```
sum (fn (i:int) => i) 100  
5050 : int
```

- ▶ Summe der Quadratzahlen von 1^2 bis 10^2

```
sum (fn (i:int) => i*i) 10  
385 : int
```

Klammersparregeln

- ▶ **Prozeduranwendung** klammert **links**

$$e_1 \ e_2 \ e_3 \quad \rightsquigarrow \quad (e_1 \ e_2) \ e_3$$

- ▶ **Pfeil** klammert **rechts**

$$t_1 \rightarrow t_2 \rightarrow t_3 \quad \rightsquigarrow \quad t_1 \rightarrow (t_2 \rightarrow t_3)$$

- ▶ **Stern vor Pfeil**

$$int * int \rightarrow int * int \quad \rightsquigarrow \quad (int * int) \rightarrow (int * int)$$

Tripeldarstellung

► **Beispiel 1:** `fun f (x:int) (y:int) = x+y`
 `val g = f 7`

liefert die Bindungen:

$f := (\text{fun } f \ x \ y = x + y, \ int \rightarrow int \rightarrow int, \ [])$

$g := (\text{fn } y \Rightarrow x + y, \ int \rightarrow int, \ [x := 7])$

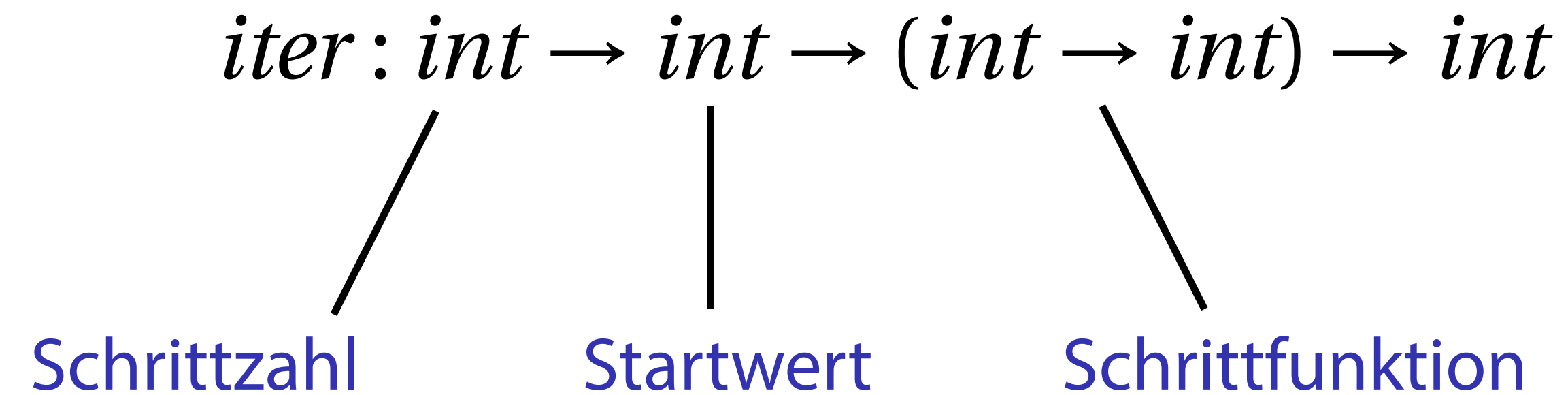
► **Beispiel 2:** `fun f (x:int) (y:int) : int = f x y`
 `val g = f 7`

liefert die Bindungen:

$f := (\text{fun } f \ x \ y = f \ x \ y, \ int \rightarrow int \rightarrow int, \ [])$

$g := (\text{fn } y \Rightarrow f \ x \ y, \ int \rightarrow int, \ [x := 7, \ f := (\text{fun } f \ x \ y = f \ x \ y, \ int \rightarrow int \rightarrow int, \ [])])$

Bestimmte Iteration: Iter



$$iter\ n\ s\ f = \underbrace{f(\dots(f\ s)\dots)}_{n\text{-mal}}$$

```
fun iter (n:int) (s:int) (f:int->int) : int =  
  if n<1 then s else iter (n-1) (f s) f
```

Beispiel

```
fun iter (n:int) (s:int) (f:int->int) : int =  
  if n<1 then s else iter (n-1) (f s) f
```

$$x^n = 1 \cdot \underbrace{x \dots \cdot x}_{n\text{-mal}}$$

```
fun power (x:int) (n:int) = iter n 1 (fn (a:int) => a*x)  
val power : int → int → int
```

```
power 2 10  
1024 : int
```

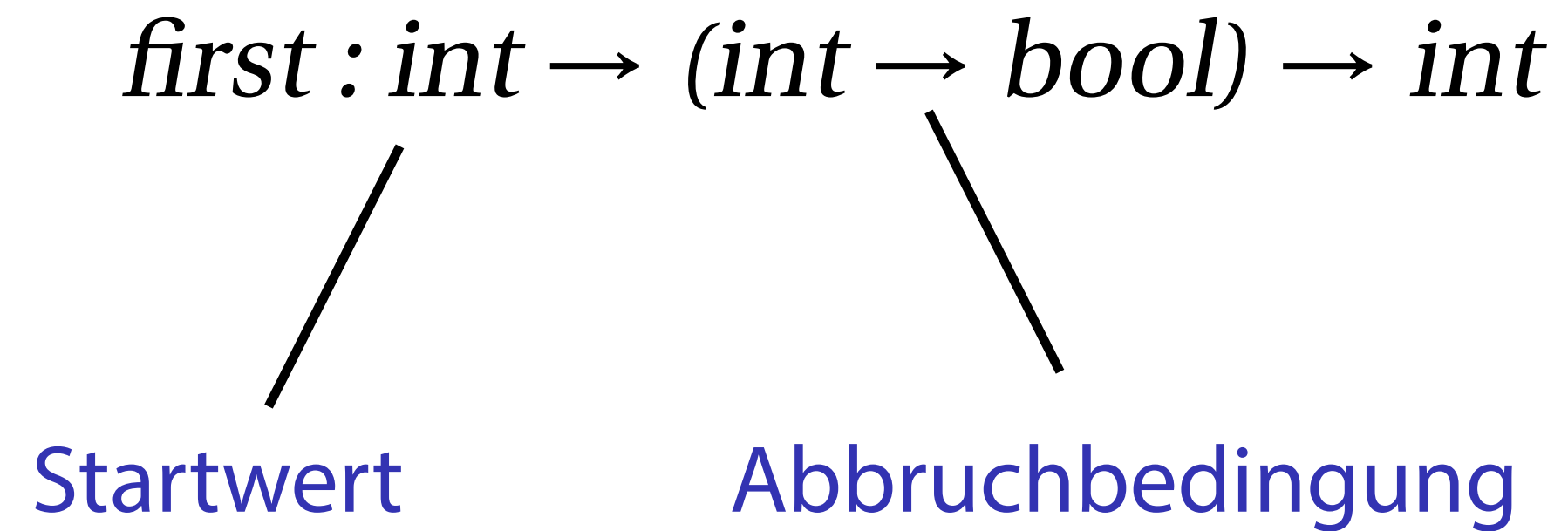
Frage

Welche der folgenden Ausdrücke haben den Wert 5?

- ▶ `iter 1 1 (fn (x:int) => x+2)`
- ▶ `iter 2 1 (fn (x:int) => x+2)`
- ▶ `iter 1 2 (fn (x:int) => x+2)`
- ▶ `iter 2 2 (fn (x:int) => x+2)`

```
fun iter (n:int) (s:int) (f:int->int) : int =  
  if n<1 then s else iter (n-1) (f s) f
```

Unbestimmte Iteration: First



$$\textit{first } s \, p = \min\{x \in \mathbb{Z} \mid x \geq s \text{ und } p \, x = \textit{true}\}$$

```
fun first (s:int) (p:int->bool) : int =  
  if p s then s else first (s+1) p
```

Beispiel

► Natürliche Quadratwurzel

$$\lfloor \sqrt{n} \rfloor = \min\{k \in \mathbb{N} \mid k^2 > n\} - 1$$

```
fun first (s:int) (p:int->bool) : int
  = if p s then s else first (s+1) p
```

```
fun sqrt (x:int)
  = first 1 (fn (k:int) => k*k>x) - 1
```

Variation von Iter

► Gauss Summe $\sum_{i=1}^n i$

```
fun iter' (n:int) (s:int*int)
  (f:int*int->int*int):int*int
  = if n<1 then s else iter' (n-1) (f s) f
```

```
fun gauss (n:int)
  = #2(iter' n (1,0)
    (fn (i:int, a:int) => (i+1, a+i)))
```

Variation von Iter

► Ist x eine gerade Zahl?

```
fun iter' (n:int) (s:bool)
    (f:bool->bool):bool
    = if n<1 then s else iter' (n-1) (f s) f
```

```
fun inv (b:bool) = not b
```

```
fun even (x:int)
    = iter' x true inv
```

Polymorphes Iter

$$iter: int \rightarrow \alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$$

Schrittzahl Startwert Schrittfunktion

Typvariable α steht für **beliebigen Typ**.

(**Lexikalische Syntax**: Typvariablen sind eigene Klasse von Wörtern, die mit dem Hochkomma beginnen.)

```
fun 'a iter (n:int) (s:'a) (f:'a->'a) : 'a =  
    if n<1 then s else iter (n-1) (f s) f  
val  $\alpha$  iter : int  $\rightarrow$   $\alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$ 
```


Beispiele

```
fun power (x:int) (n:int) = iter n 1 (fn (a:int) => a*x)
```

```
val power : int → int → int
```

```
power 2 10
```

```
1024 : int
```

```
fun power' (x:real) (n:int) = iter n 1.0 (fn (a:real) => a*x)
```

```
val power' : real → int → real
```

```
power' 2.0 10
```

```
1024.0 : real
```

```
fun gauss (n:int) =
```

```
  #2(iter n (1,0) (fn (i:int, a:int) => (i+1, a+i)))
```

```
val gauss : int → int
```

```
gauss 10
```

```
55 : int
```

Polymorphe Typisierung

- ▶ **Typschema** beschreibt die **möglichen Typen**:

$$\forall \alpha. \text{int} \rightarrow \alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$$

- ▶ **Typvariable** α ist durch \forall **quantifiziert**.

- ▶ Die **Instanzen des Schemas** sind die möglichen Typen:

$\text{int} \rightarrow \text{int} \rightarrow (\text{int} \rightarrow \text{int}) \rightarrow \text{int}$	$(\alpha = \text{int})$
$\text{int} \rightarrow \text{real} \rightarrow (\text{real} \rightarrow \text{real}) \rightarrow \text{real}$	$(\alpha = \text{real})$
$\text{int} \rightarrow \text{int} * \text{int} \rightarrow (\text{int} * \text{int} \rightarrow \text{int} * \text{int}) \rightarrow \text{int} * \text{int}$	$(\alpha = \text{int} * \text{int})$

Frage

Für welche Typschemen ist
 $\text{int} \rightarrow \text{bool} \rightarrow \text{int}$
eine Instanz?

- ▶ $\forall \alpha \beta . \alpha \rightarrow \alpha \rightarrow \beta$
- ▶ $\forall \alpha \beta . \alpha \rightarrow \beta \rightarrow \beta$
- ▶ $\forall \alpha \beta . \alpha \rightarrow \beta \rightarrow \alpha$
- ▶ $\forall \alpha \beta . \alpha \rightarrow \beta \rightarrow \text{int}$

Beispiel

```
fun ('a, 'b) project2 (x: 'a, y: 'b) = y
```

```
val ( $\alpha$ ,  $\beta$ ) project2 :  $\alpha * \beta \rightarrow \beta$ 
```

► Typschema: $\forall \alpha \beta. \alpha * \beta \rightarrow \beta$

► Instanzen:

$int * int \rightarrow int$

$(\alpha = int, \beta = int)$

$int * real \rightarrow real$

$(\alpha = int, \beta = real)$

$(int * bool) * real \rightarrow real$

$(\alpha = int * bool, \beta = real)$

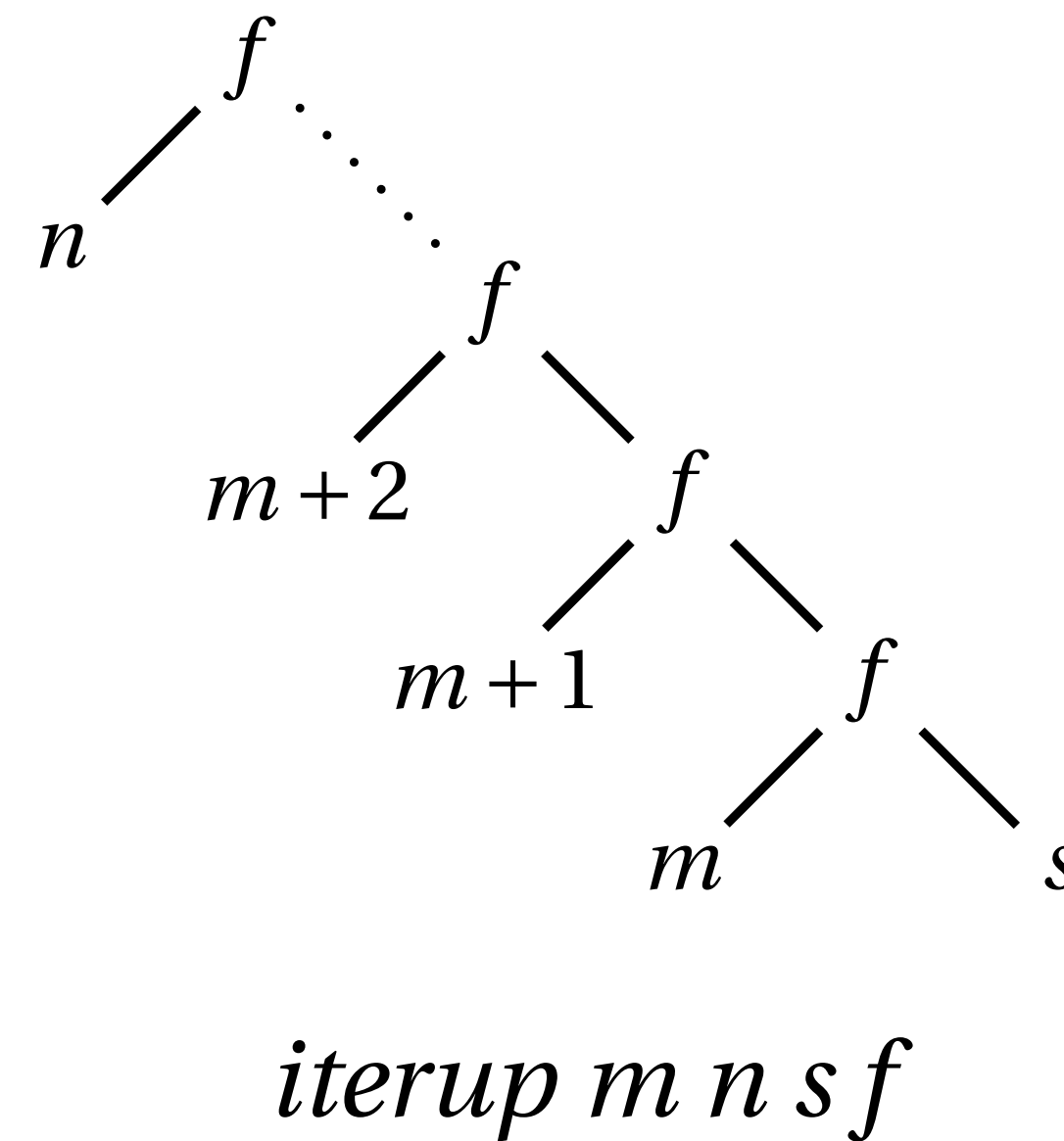
iterup

iterup: $int \rightarrow int \rightarrow \alpha \rightarrow (int * \alpha \rightarrow \alpha) \rightarrow \alpha$

Anfang Ende Startwert Schrittfunktion

$int * \alpha \rightarrow \alpha$

Index Akku



```
fun 'a iterup (m:int) (n:int) (s:'a)
      (f: int * 'a -> 'a) : 'a =
  if m>n then s else iterup (m+1) n (f(m,s)) f
```

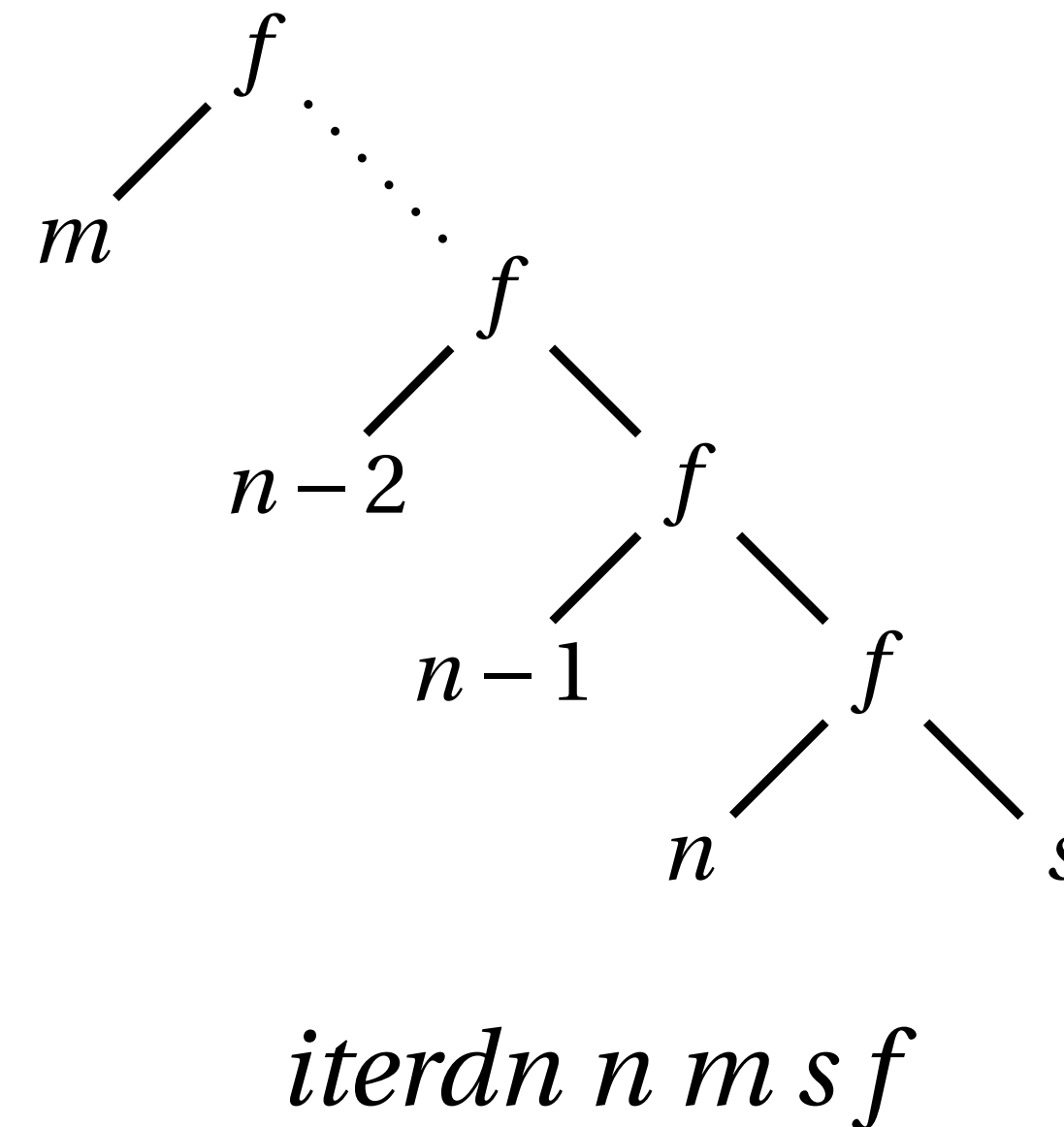
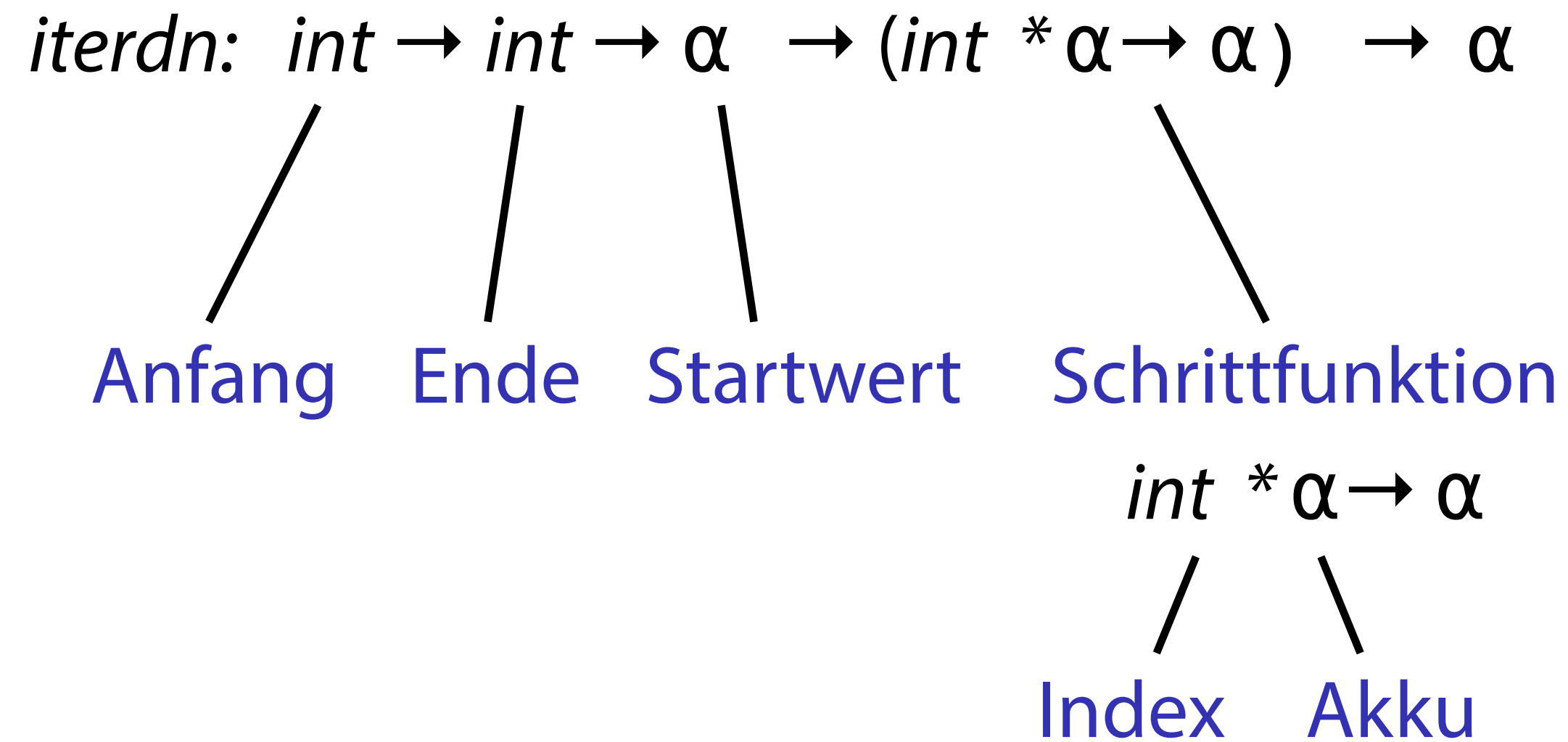
Beispiel

► Gauss Summe $\sum_{i=1}^n i$

```
fun 'a iterup (m:int) (n:int) (s:'a)  
    (f: int * 'a -> 'a) : 'a =  
    if m>n then s else iterup (m+1) n (f(m,s)) f
```

```
fun gauss (n:int) =  
    iterup 1 n 0 (fn (i:int, a:int) => a+i)
```

iterdn



```

fun 'a iterdn (n:int) (m:int) (s:'a)
  (f: int * 'a -> 'a) : 'a =
  if n<m then s else iterdn (n-1) m (f(n,s)) f
  
```

```

fun 'a iterup (m:int) (n:int) (s:'a)
  (f: int * 'a -> 'a) : 'a =
  if m>n then s else iterup (m+1) n (f(m,s)) f
  
```

Frage

Welche der folgenden Prozeduren sind endrekursiv?

- ▶

```
fun 'a iterup (m:int) (n:int) (s:'a) (f: int * 'a -> 'a) : 'a =  
  if m>n then s else iterup (m+1) n (f(m,s)) f
```
- ▶

```
fun 'a iterdn (n:int) (m:int) (s:'a)  
  (f: int * 'a -> 'a) : 'a =  
  if n<m then s else iterdn (n-1) m (f(n,s)) f
```
- ▶ Keine der beiden

Monomorphe und polymorphe Bezeichner

- ▶ Ein **Bezeichner** heißt **polymorph** wenn er mit einem **Typschema** getypt ist
- ▶ Ein **Bezeichner** heißt **monomorph** wenn er mit einem **Typ** getypt ist

Polymorphe Bezeichner können nur mit Hilfe von Deklarationen eingeführt werden.

Argumentvariablen werden immer monomorph getypt.

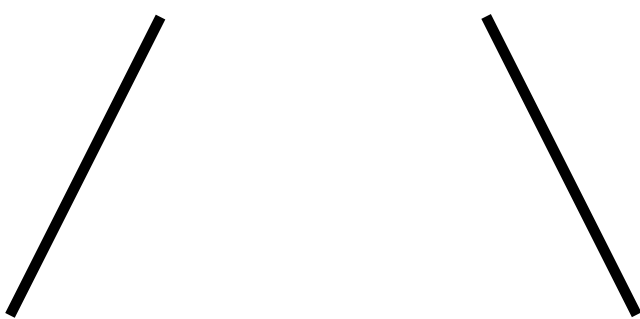
- ▶ Eine **Deklaration** heißt **polymorph** wenn sie **mindestens einen polymorphen** Bezeichner deklariert
- ▶ Eine **Deklaration** heißt **monomorph** wenn sie **nur monomorphe** Bezeichner deklariert

Beispiel

`fun 'a id (x:'a) = x`

polymorph: $\forall \alpha. \alpha \rightarrow \alpha$

monomorph: α



Die Deklaration ist **polymorph**.

Typinferenz

- ▶ **Typangaben für Argumentvariablen und Ergebnisse von Prozeduren können in Standard ML meist weggelassen werden.**
- ▶ **Typinferenz:** Automatisches Verfahren zur Ergänzung fehlender Typen.

```
fun 'a iter (n:int) (s:'a) (f:'a -> 'a) : 'a =  
  if n<1 then s else iter (n-1) (f s) f  
     $\forall \alpha \text{ int} \rightarrow \alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$ 
```

```
fun iter n s f =  
  if n<1 then s else iter (n-1) (f s) f  
     $\forall \alpha \text{ int} \rightarrow \alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$ 
```

```
fun 'a iter n s f =  
  if n<1 then (s:real) else iter (n-1) (f s) f  
     $\forall \alpha \text{ int} \rightarrow \text{real} \rightarrow (\text{real} \rightarrow \text{real}) \rightarrow \text{real}$ 
```

Frage

Inferieren Sie den Typ von
fun f (x,y,z) = if x y then y else z

- ▶ $(\text{int} \rightarrow \text{bool}) * \text{int} * \text{int} \rightarrow \text{int}$
- ▶ $\forall \alpha . (\alpha \rightarrow \text{bool}) * \alpha * \alpha \rightarrow \alpha$
- ▶ $\forall \alpha \beta . (\alpha \rightarrow \text{bool}) * \beta * \beta \rightarrow \beta$
- ▶ $\forall \alpha \beta \gamma \delta . \alpha * \beta * \gamma \rightarrow \delta$

Typinferenz

- ▶ **Grundprinzip:** möglichst **allgemeine** Typisierung
- ▶ **Aber: Überladene Operatoren** liefern eindeutige Typen

Beispiel:

```
fun plus x y = x+y
```

int → *int* → *int*

```
fun plus (x:real) y = x+y
```

real → *real* → *real*

```
fun plus x y : real = x+y
```

real → *real* → *real*

Typen und Gleichheit

Gleichheitstest (=) ist nicht für alle Typen verfügbar!

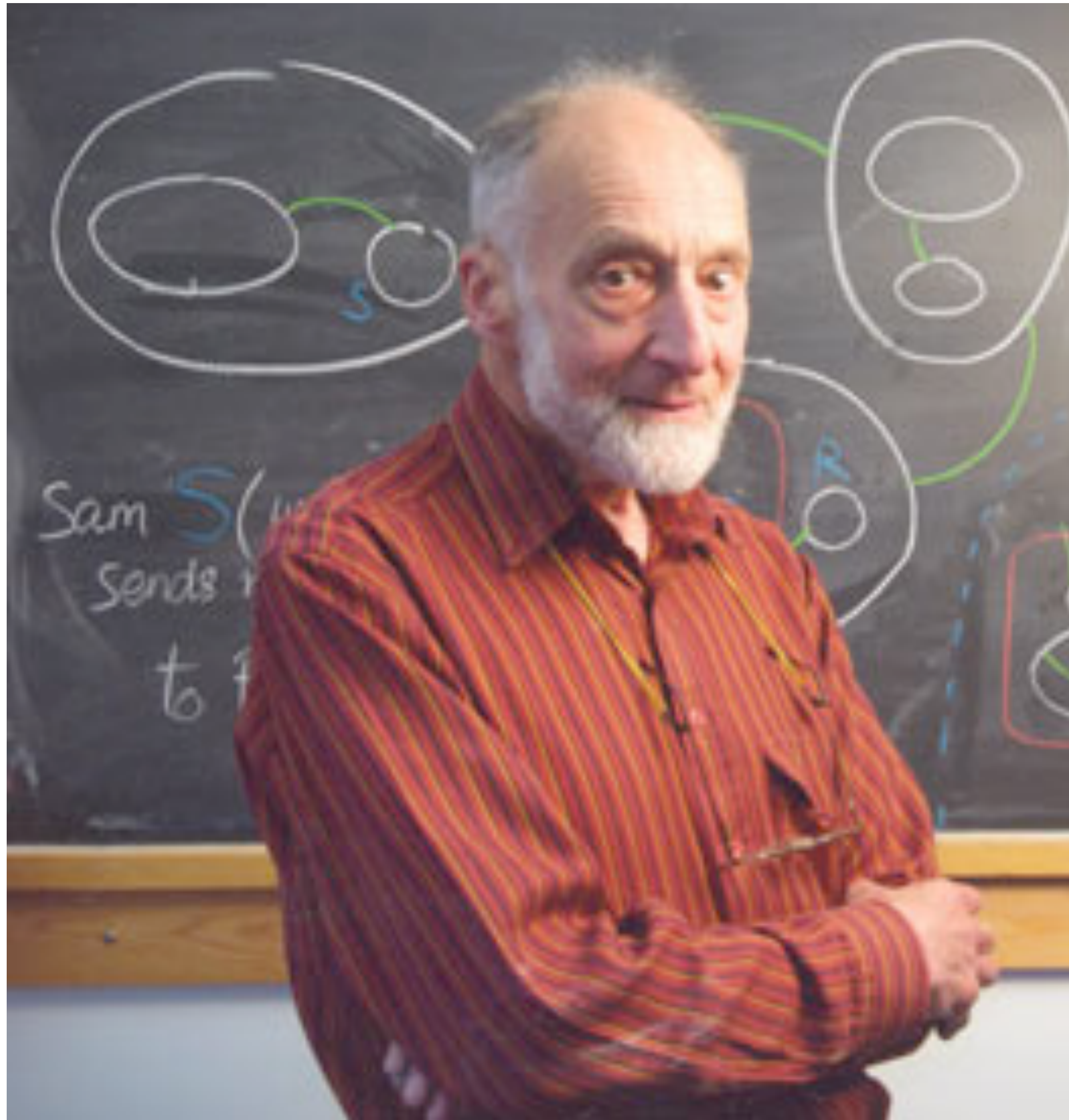
- ▶ **Typen mit Gleichheit:** *bool, int, real, unit* sowie Tupel darüber
- ▶ **Typen ohne Gleichheit:** Prozedurtypen
- ▶ **Typvariablen** für Typen **mit** Gleichheit: *' ' a, ' ' b, ' ' c, ...*
Typvariablen für Typen **ohne** Gleichheit: *' a, ' b, ' c, ...*

```
fun eq x y = x = y  
val ''a eq: ''a → ''a → bool
```

```
fun neq x y = x <> y  
val ''a eq: ''a → ''a → bool
```

```
fun f (x,y,z) = if x=y then x else z  
val ''a f: ''a * ''a * ''a → ''a
```

```
(fn x => 2*x) = (fn x => 2*x)  
! Type clash: int → int is not an equality type
```

Robin Milner
(1934 – 2010)

www.prog1.saarland