Lambda

Contents

[Theorie 2](#_Toc68518283)

[Aufbau 2](#_Toc68518284)

[Äquivalenz 2](#_Toc68518285)

[Umbenennung 2](#_Toc68518286)

[Semantik 3](#_Toc68518287)

[Church 3](#_Toc68518288)

[Zahlen 3](#_Toc68518289)

[Booleans 3](#_Toc68518290)

[Methoden 4](#_Toc68518291)

[Selbstapplikation 5](#_Toc68518292)

[Divergenz 5](#_Toc68518293)

[Y-Kombinator 5](#_Toc68518294)

[Rekursion 6](#_Toc68518295)

[Auswertungstrategien 7](#_Toc68518296)

[Auswertung 7](#_Toc68518297)

[Wert 7](#_Toc68518298)

[Call-By-Name 8](#_Toc68518299)

[Call-By-Value 8](#_Toc68518300)

[Normalreihenfolge 9](#_Toc68518301)

[Beispiel 9](#_Toc68518302)

[FAQ 9](#_Toc68518303)

[Aufgaben 9](#_Toc68518304)

[Call-By-Name/Value 9](#_Toc68518305)

[Unendliche Reduktion 10](#_Toc68518306)

[Zähler (SS16 A4) 10](#_Toc68518307)

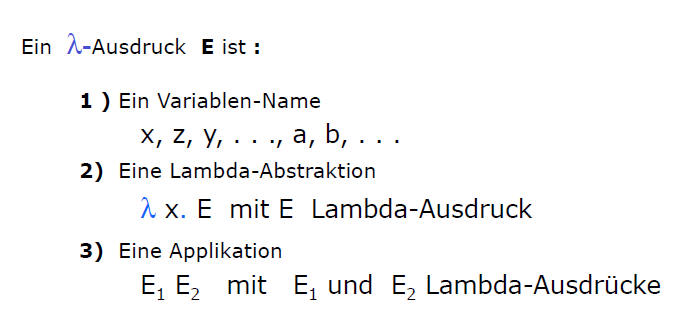
[Gegeben 10](#_Toc68518308)

[Reduktion zeigen 10](#_Toc68518309)

[„new“ Term mit Y Kombinator 10](#_Toc68518310)

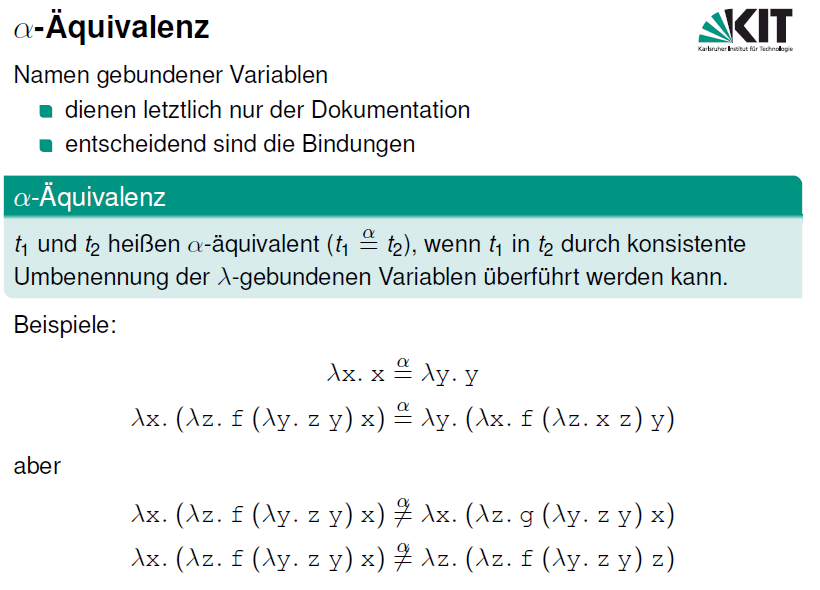
# Theorie

## Aufbau

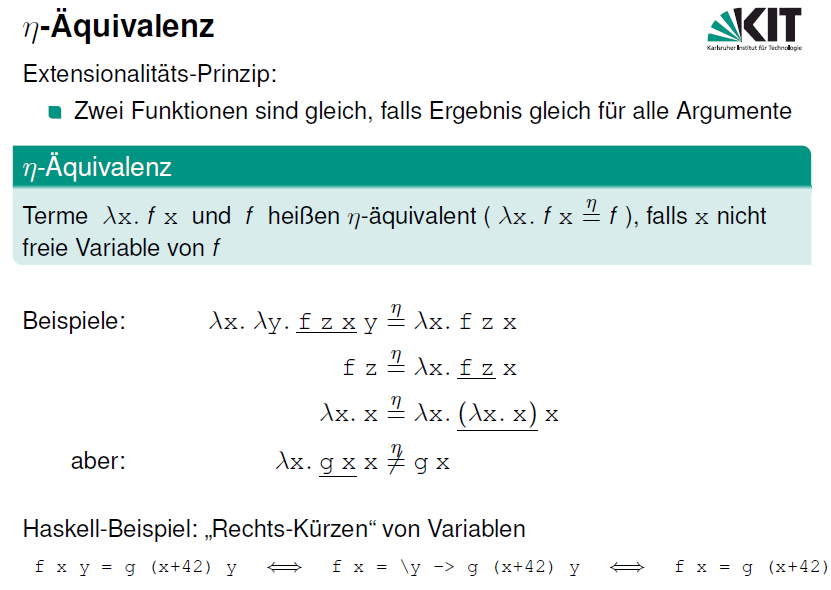


## Äquivalenz

### Umbenennung

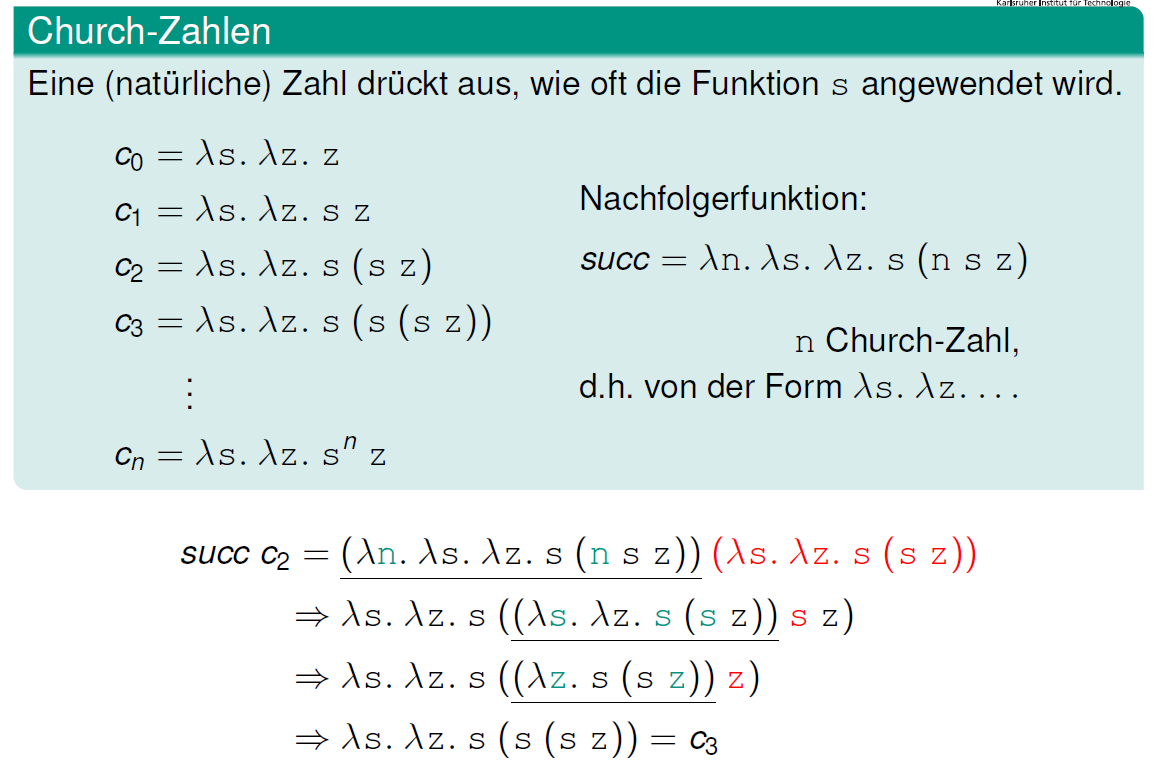


### Semantik



## Church

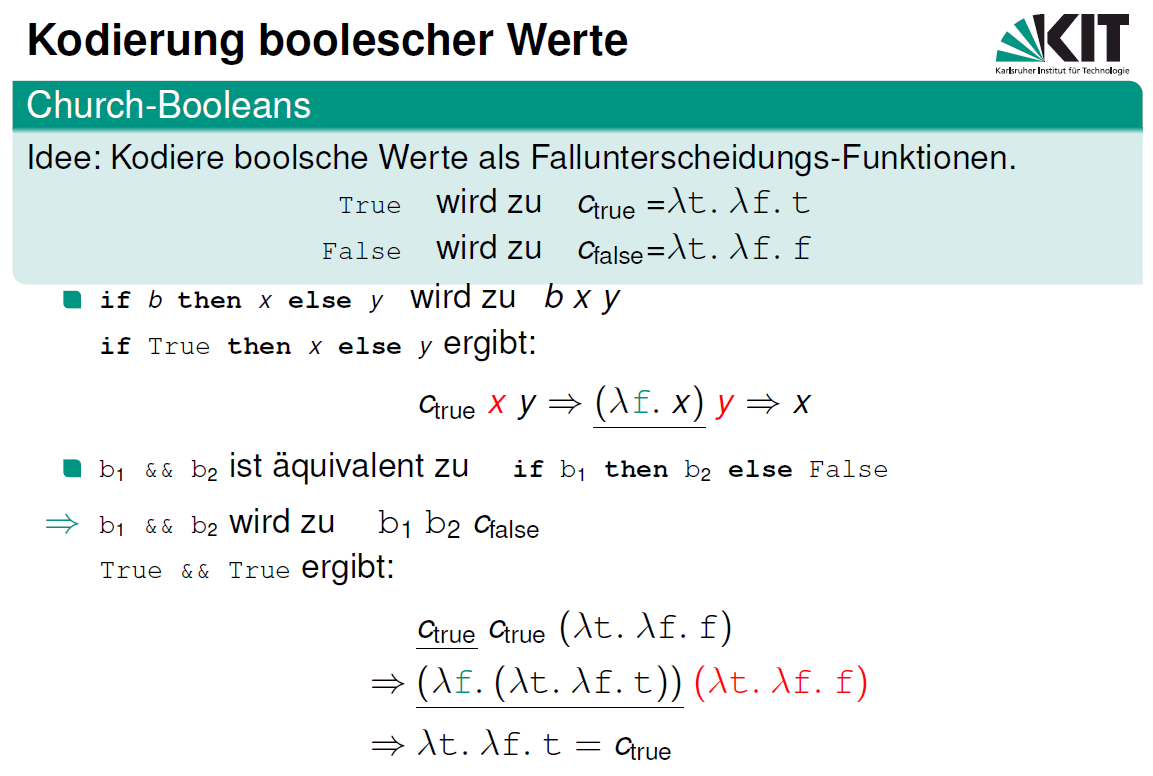
### Zahlen



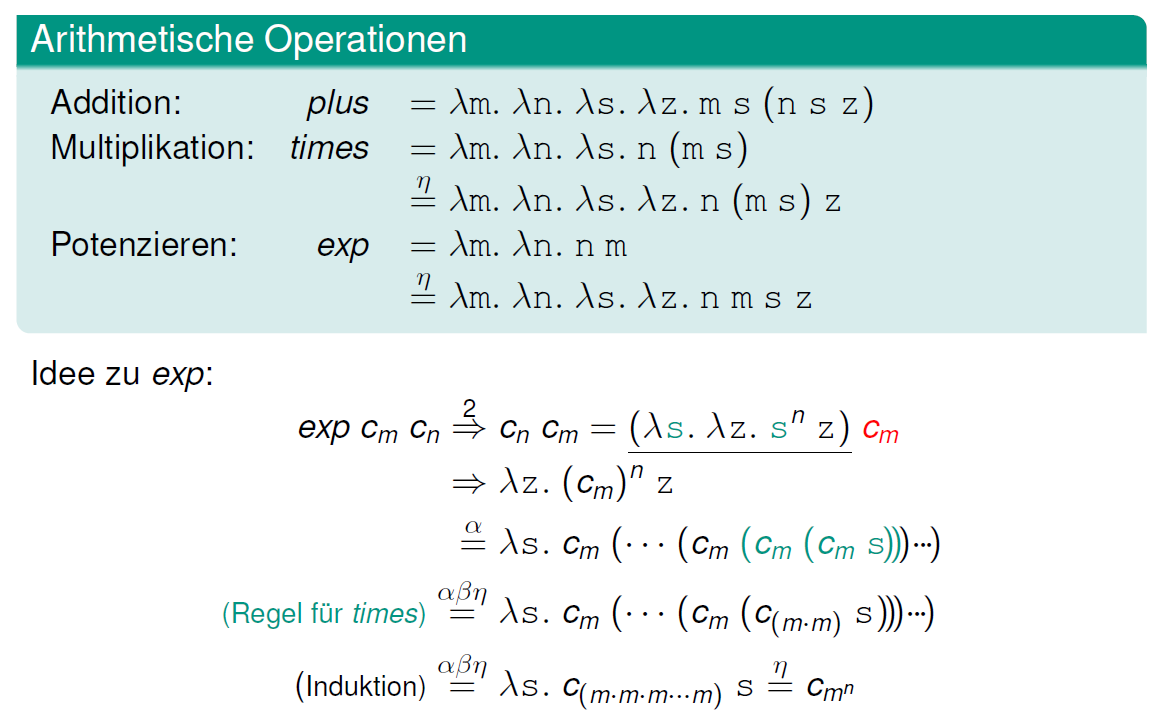
### Booleans

**c\_true a b = a**

**c\_false a b = b**



### Methoden



### sub

sub c2 c1 = c2 - c1

sub = \m. \n. n pred m

Text

Description automatically generated

### if Then Else

Aufbau: Funktion, die c\_true oder c\_false ergibt (z.B isZero c\_n), dann True-Case und False-Case.

(isZero arg) True\_Term False\_term

### equal

Hierbei muss man jedoch beachten, dass Church-Zahlen

*natürliche Zahlen* sind, und Subtraktion auf natürlichen Zahlen “saturierend” ist, d.h. 0−*n* = 0.

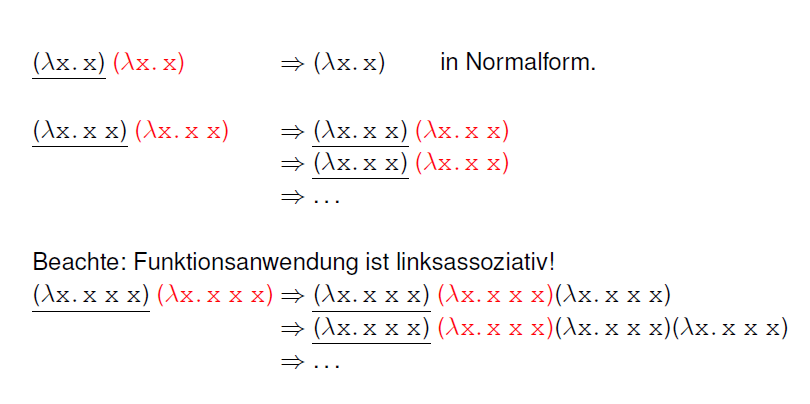
Daher gilt *n* = *m* nur dann, wenn sowohl *n*−*m* = 0 und *m*−*n* = 0 gelten.

eq = \n. \m. (isZero (sub n m)) (isZero (sub m n)) cfalse

--> falls (isZero n - m) then (isZero m - n) else false

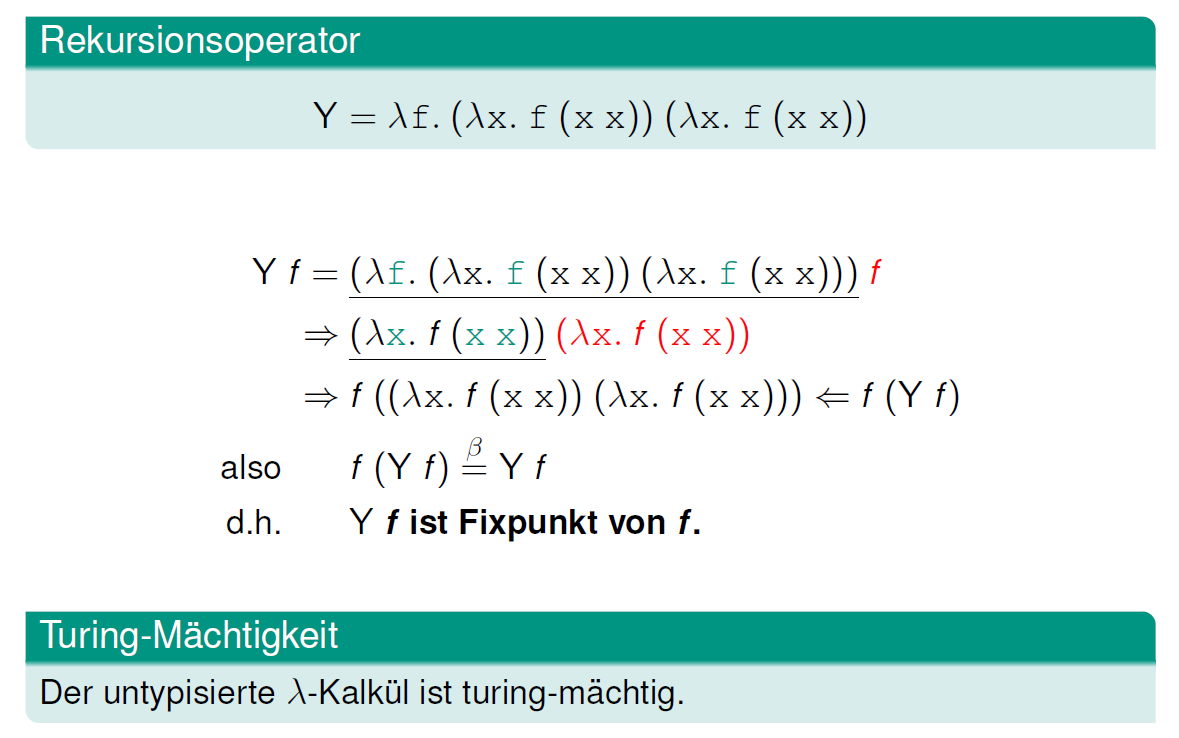
## Selbstapplikation

### Divergenz



### Y-Kombinator

Mit der Selbstapplikation des Lambdas mit zwei Argumente lässt sich die Rekursion implementieren.

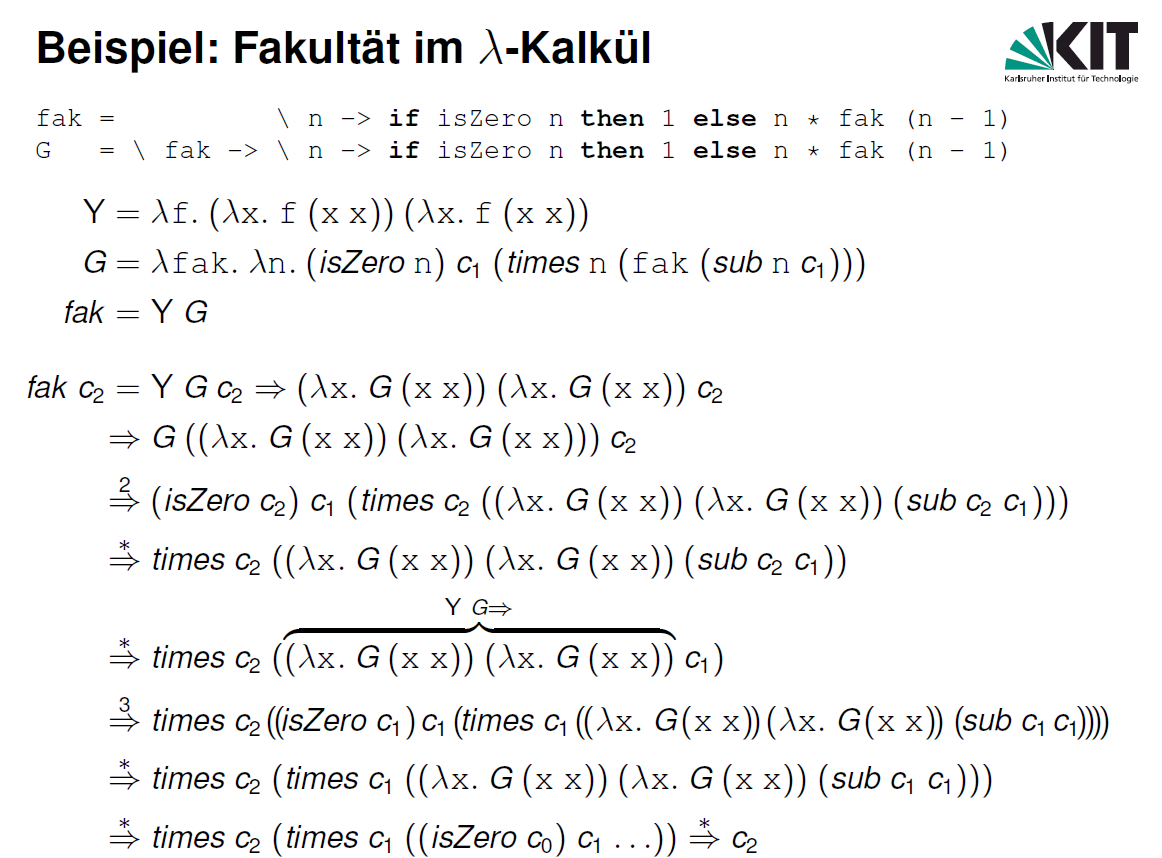


### Rekursion

1) Implementiere rekursive funktion wie immer, mit einem Abbruch-Bedingung

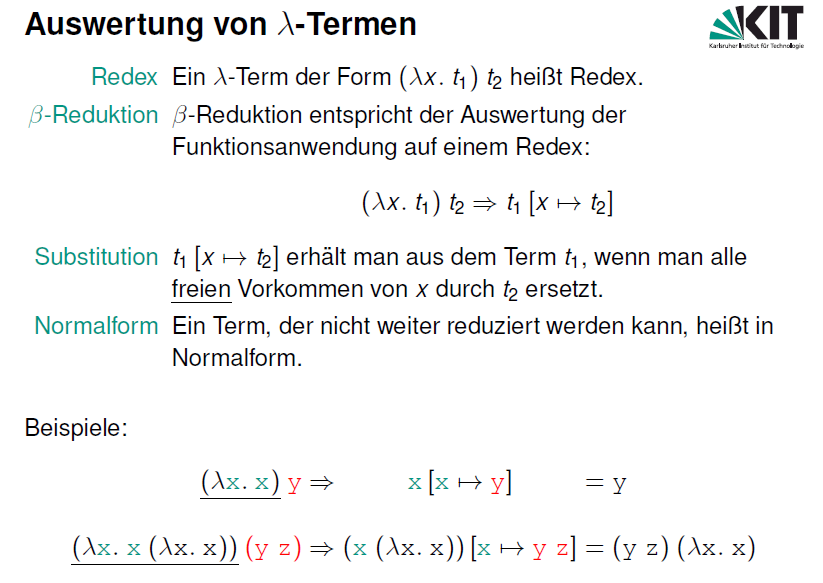
2) Erstelle eine erweiterte Funktion G als erste Funktion + Lamda für sich selbst

3) Rekursive Funktion = Y G

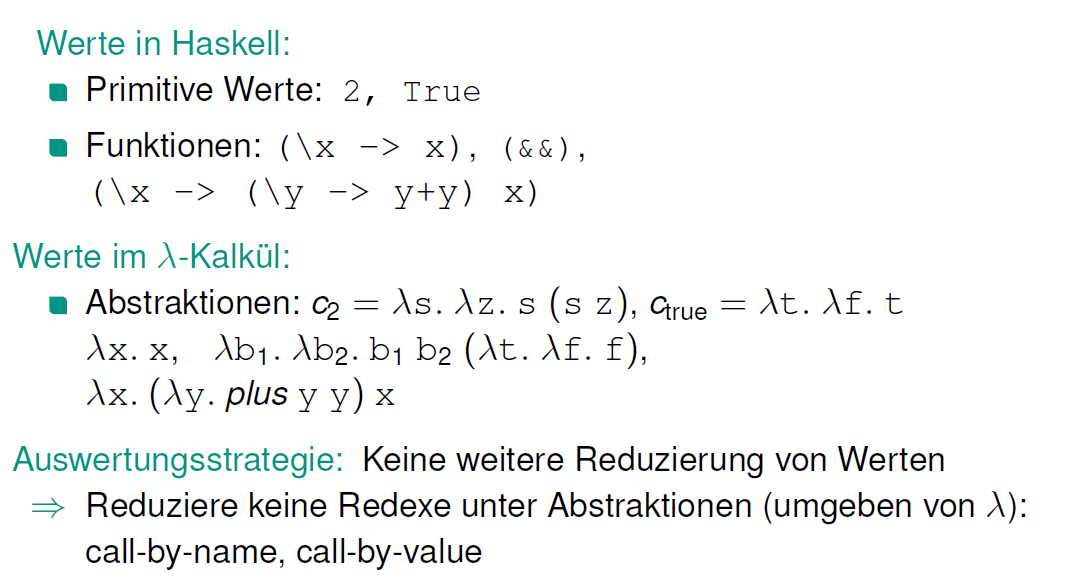


## Auswertungstrategien

### Auswertung

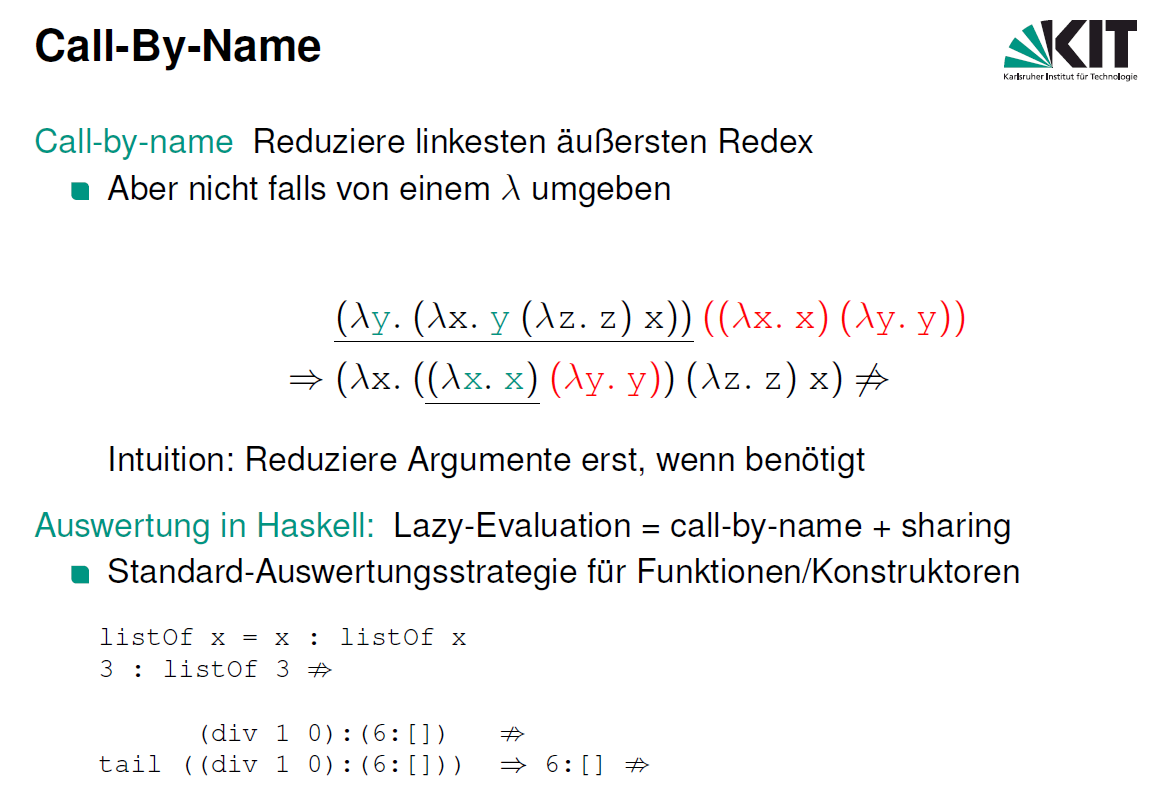


### Wert



So, what is a value? In the pure lambda calculus, any abstraction is a value. Intuitively, a value is an expressionthat can not be reduced/executed/simplified any further.

### Call-By-Name

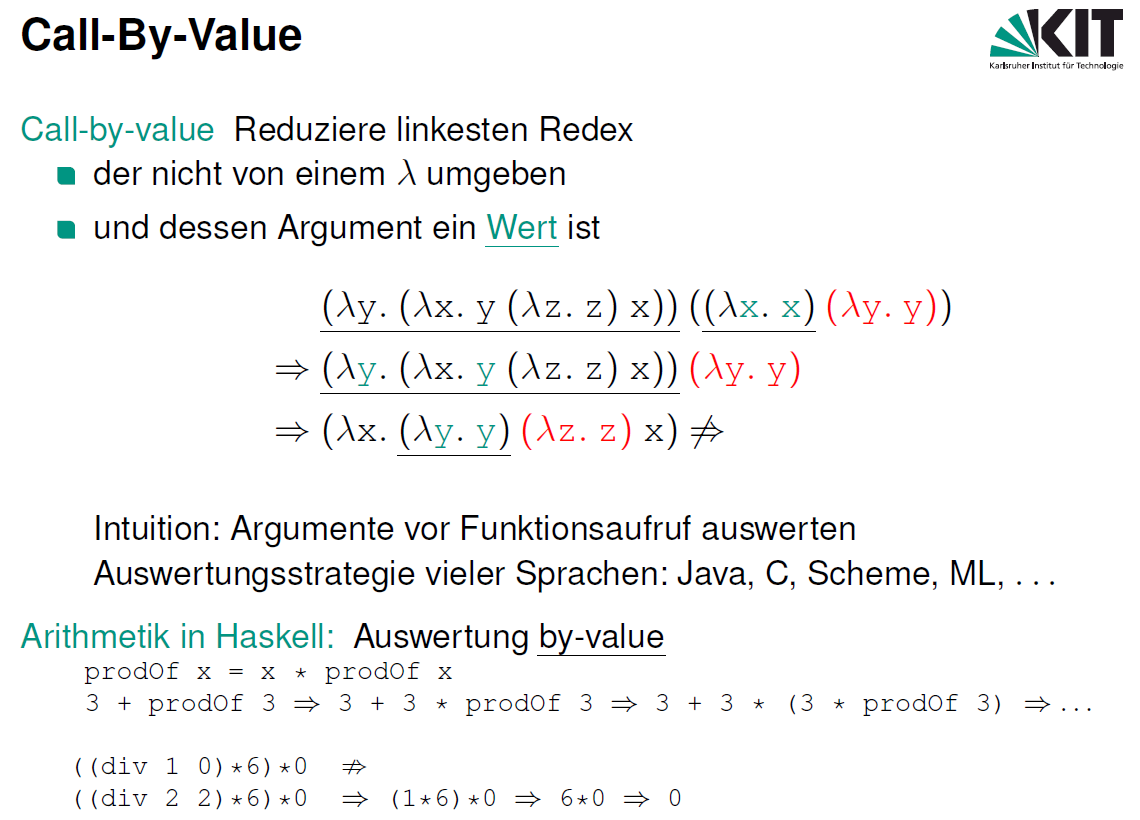


Nur die linke **\y** wird reduziert, da es keine äußere Lambda gibt.

Innere \x wird nicht reduziert, da es noch äußere \x gibt.

Äußere \x kann nicht reduziert werden - kein Redex

### Call-By-Value



Linkeste Redex, der keine äußere Lambda hat, und Argument eine Value ist --> **nicht vereinfachbar** ist.

### Normalreihenfolge

Text

Description automatically generated with low confidence

Einfach immer der linkeste Redex reduzieren

### Beispiel

(\x. x) ((\y. y) z)

------- ~ CBV

------- ~~~~~~~~~~~ NRF, CBN

# FAQ

# Aufgaben

## Call-By-Name/Value

(\t.\f. f) ((\y. (\x. x x) (\x. x x)) ((\x. x) (\x. x))) (\t.\f. f)

------ ~~~~~ CBV

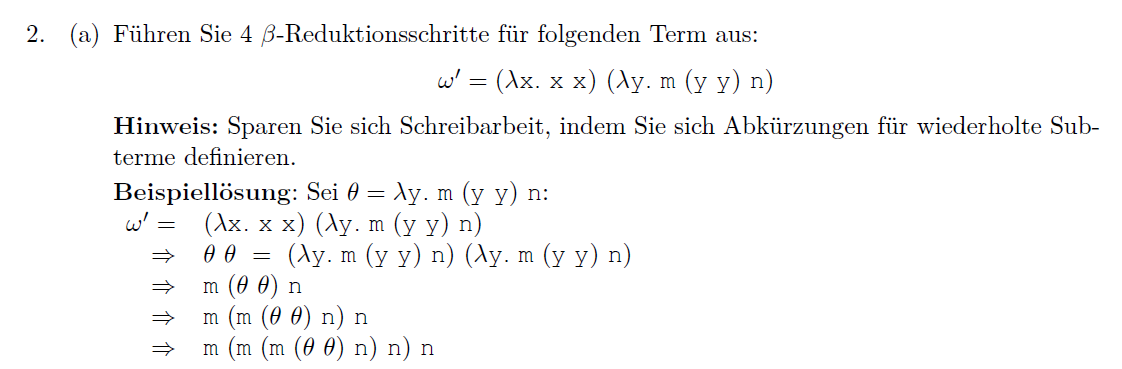
-------- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ CBN, NRF

(\y. (\z. (\x. x) (\x. x) z) y)

---------------------- ~ NRF

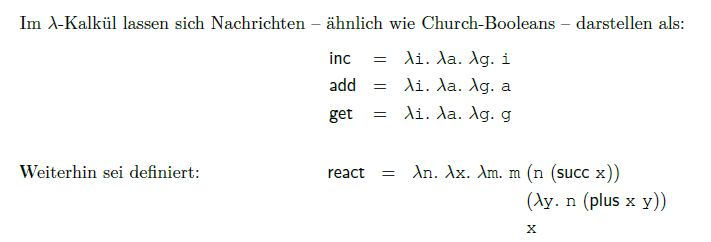
kein CBN und kein CBV (da alles von Lambda umgeben)

## Unendliche Reduktion

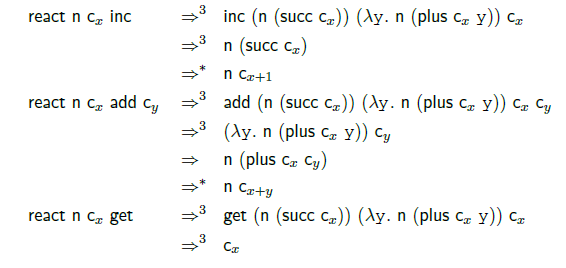


## Zähler (SS16 A4)

### Gegeben



### Reduktion zeigen



### „new“ Term mit Y Kombinator

