Lambda

Contents

[Theorie 2](#_Toc68630338)

[Aufbau 2](#_Toc68630339)

[Äquivalenz 2](#_Toc68630340)

[Umbenennung 2](#_Toc68630341)

[Semantik 3](#_Toc68630342)

[Church 3](#_Toc68630343)

[Zahlen 3](#_Toc68630344)

[Booleans 4](#_Toc68630345)

[Methoden 4](#_Toc68630346)

[sub 4](#_Toc68630347)

[if Then Else 5](#_Toc68630348)

[equal 5](#_Toc68630349)

[Selbstapplikation 5](#_Toc68630350)

[Divergenz 5](#_Toc68630351)

[Y-Kombinator 6](#_Toc68630352)

[Rekursion 6](#_Toc68630353)

[Auswertungstrategien 7](#_Toc68630354)

[Auswertung 7](#_Toc68630355)

[Wert 7](#_Toc68630356)

[Call-By-Name 8](#_Toc68630357)

[Call-By-Value 8](#_Toc68630358)

[Normalreihenfolge 9](#_Toc68630359)

[Beispiel 9](#_Toc68630360)

[FAQ 9](#_Toc68630361)

[Aufgaben 9](#_Toc68630362)

[Call-By-Name/Value 9](#_Toc68630363)

[Unendliche Reduktion 10](#_Toc68630364)

[Zähler (SS16 A4) 10](#_Toc68630365)

[Gegeben 10](#_Toc68630366)

[Reduktion zeigen 10](#_Toc68630367)

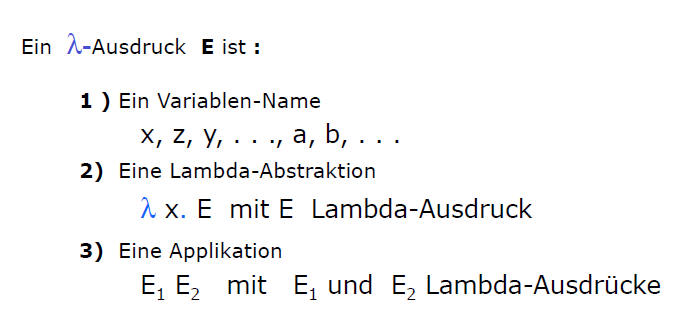
[„new“ Term mit Y Kombinator 10](#_Toc68630368)

[Listen 11](#_Toc68630369)

[Meine Lösung 12](#_Toc68630370)

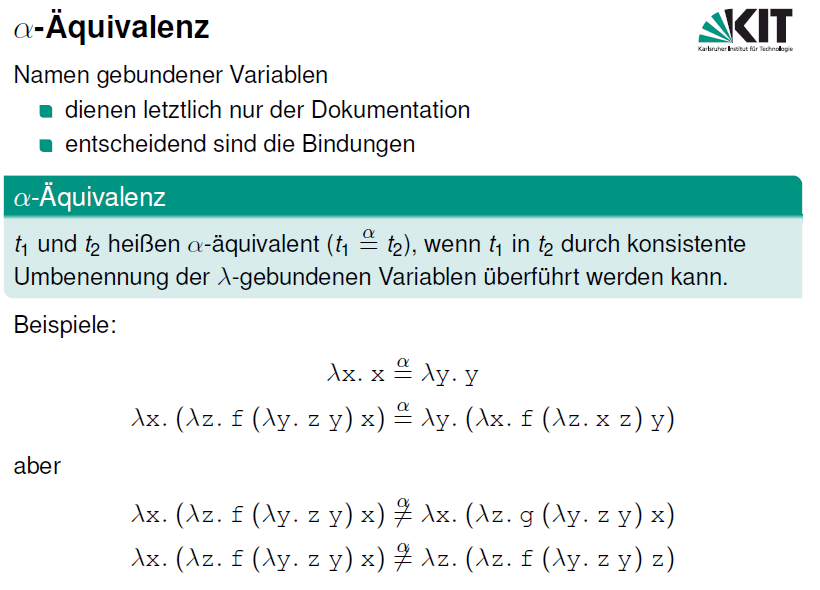
# Theorie

## Aufbau

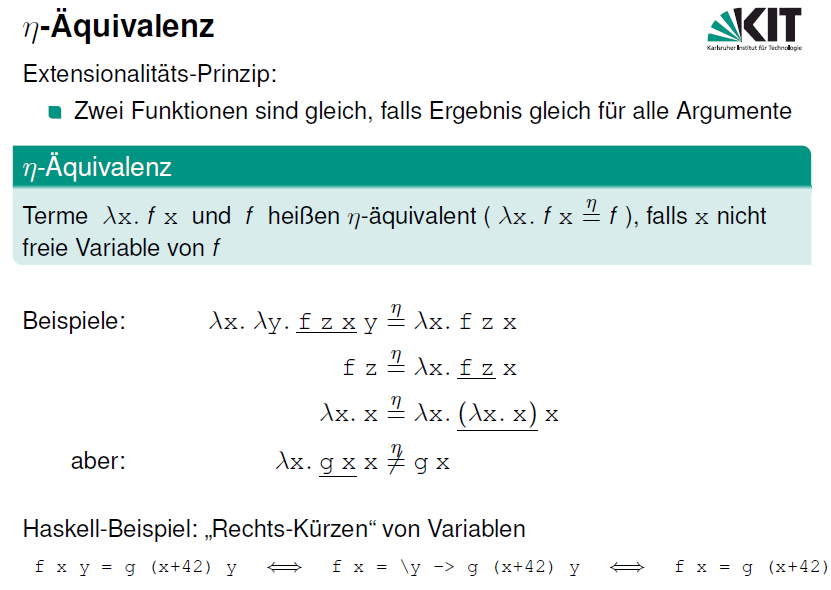


## Äquivalenz

### Umbenennung

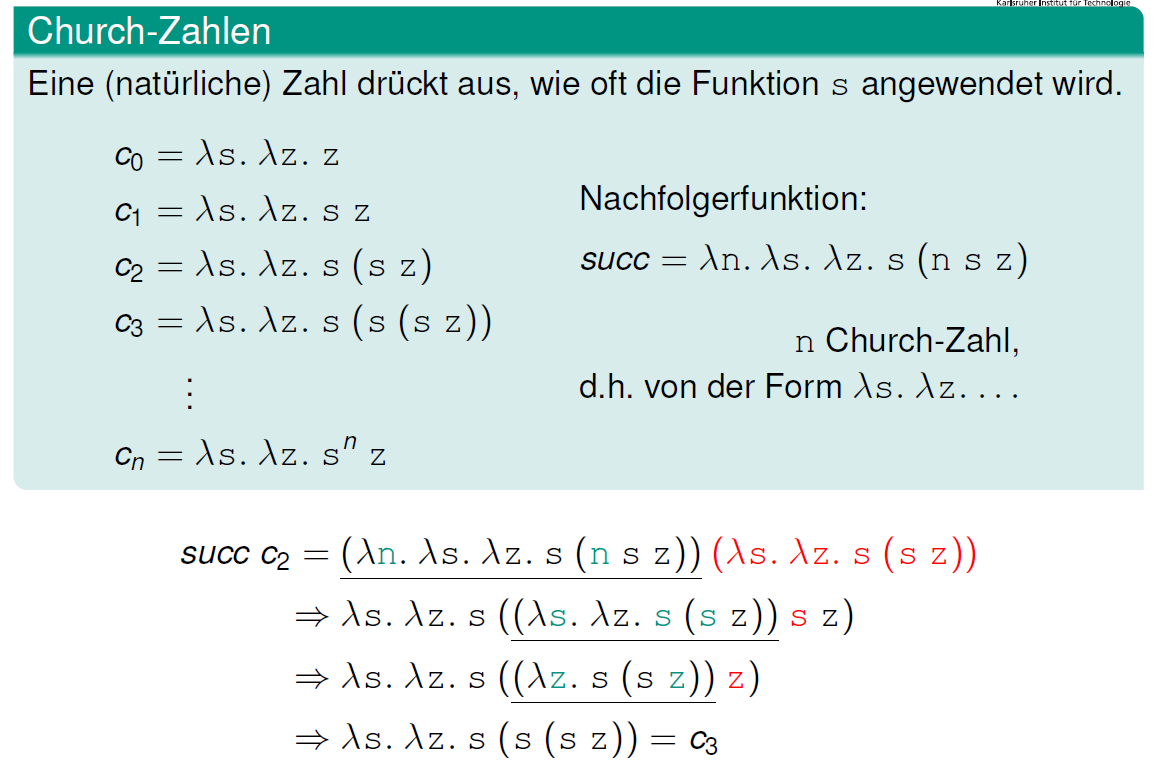


### Semantik



## Church

### Zahlen



Wichtige Eigenschaft:

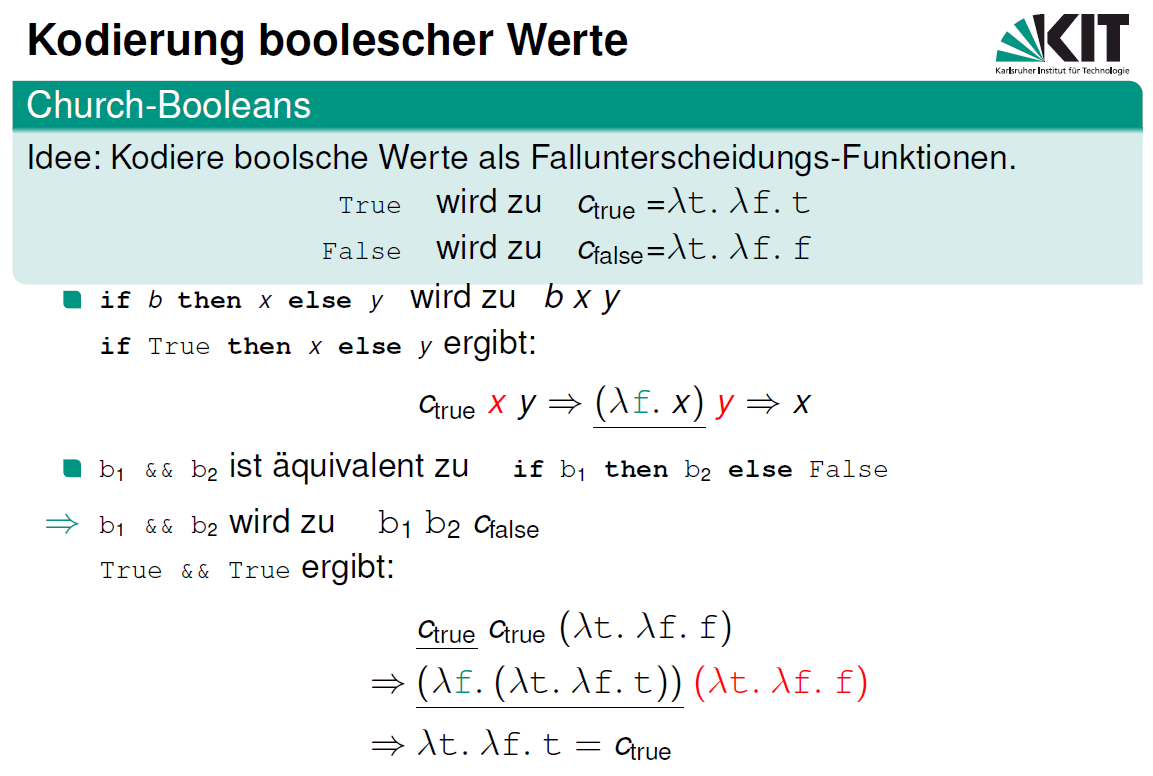
c\_3 (fun a) nil = (\s.\z.s(s(s z))) (fun a) nil = fun a (fun a (fun a nil))

c\_3 fun2 x = (\s.\z.s(s(s z))) fun2 x = fun (fun (fun x))

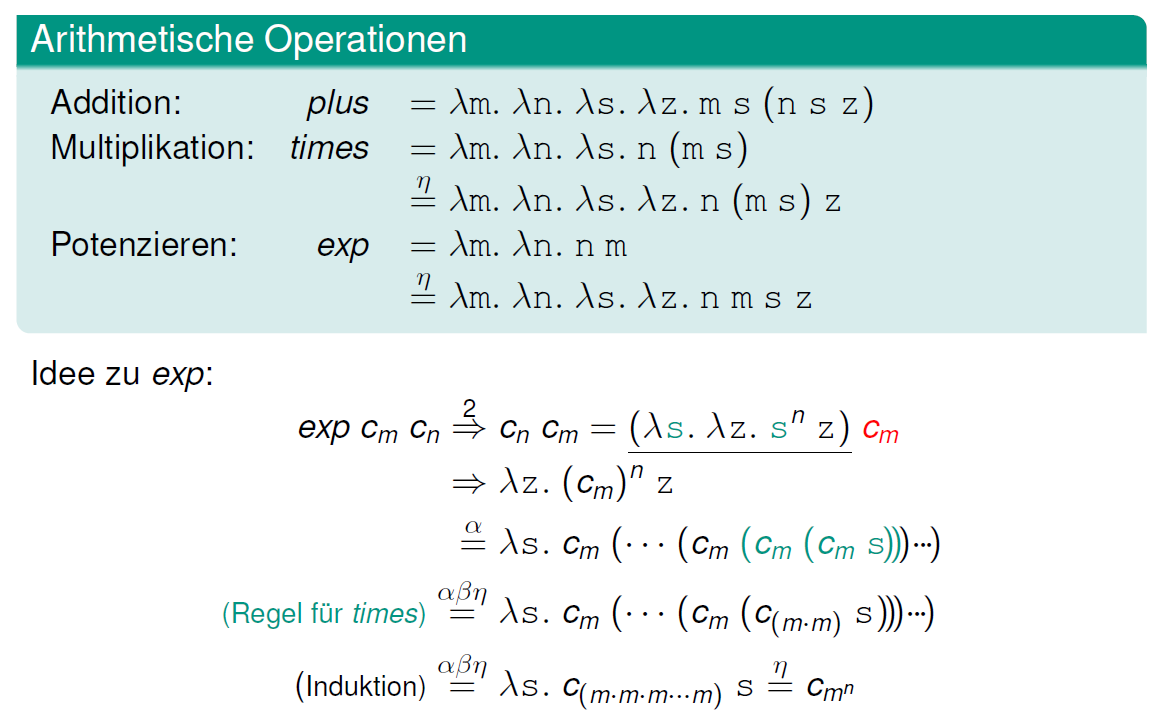
### Booleans

**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.

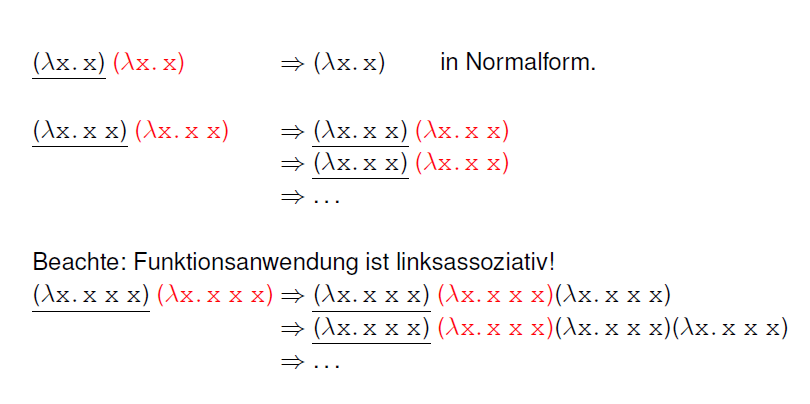
Daher gilt *n* = *m* nur dann, wenn sowohl *n*−*m* = 0 und *m*−*n* = 0 gelten.

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

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

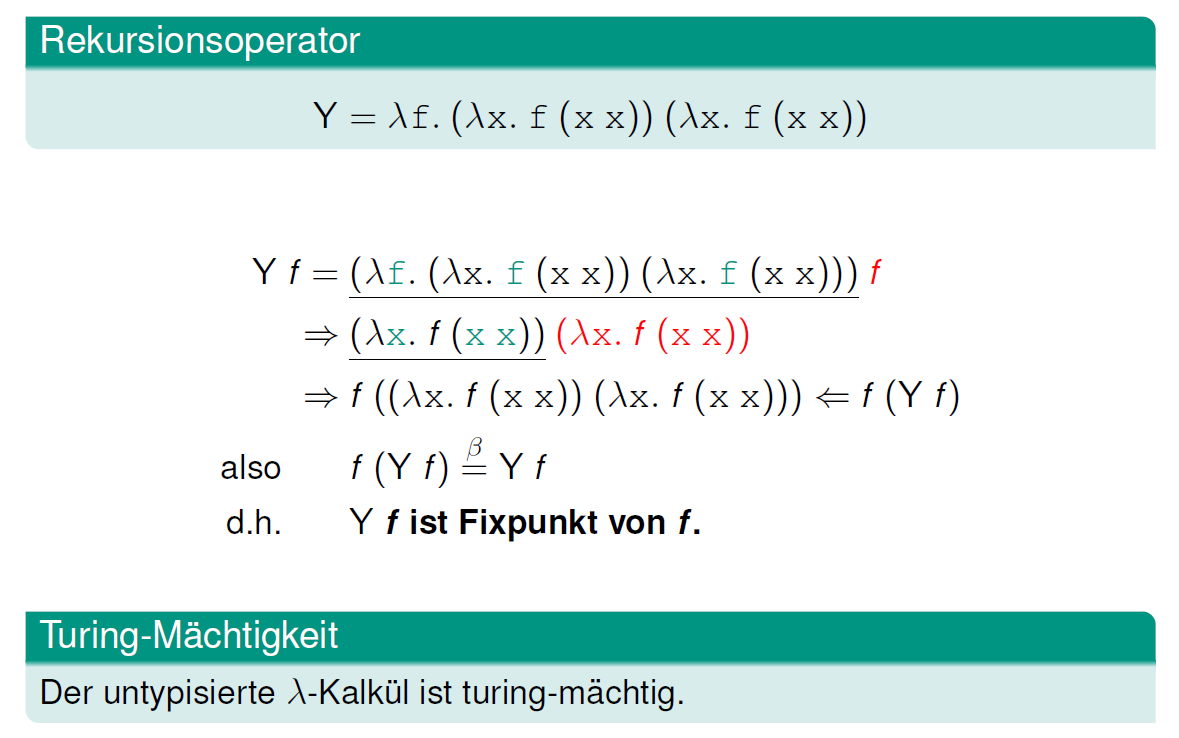
## Selbstapplikation

### Divergenz



### Y-Kombinator

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

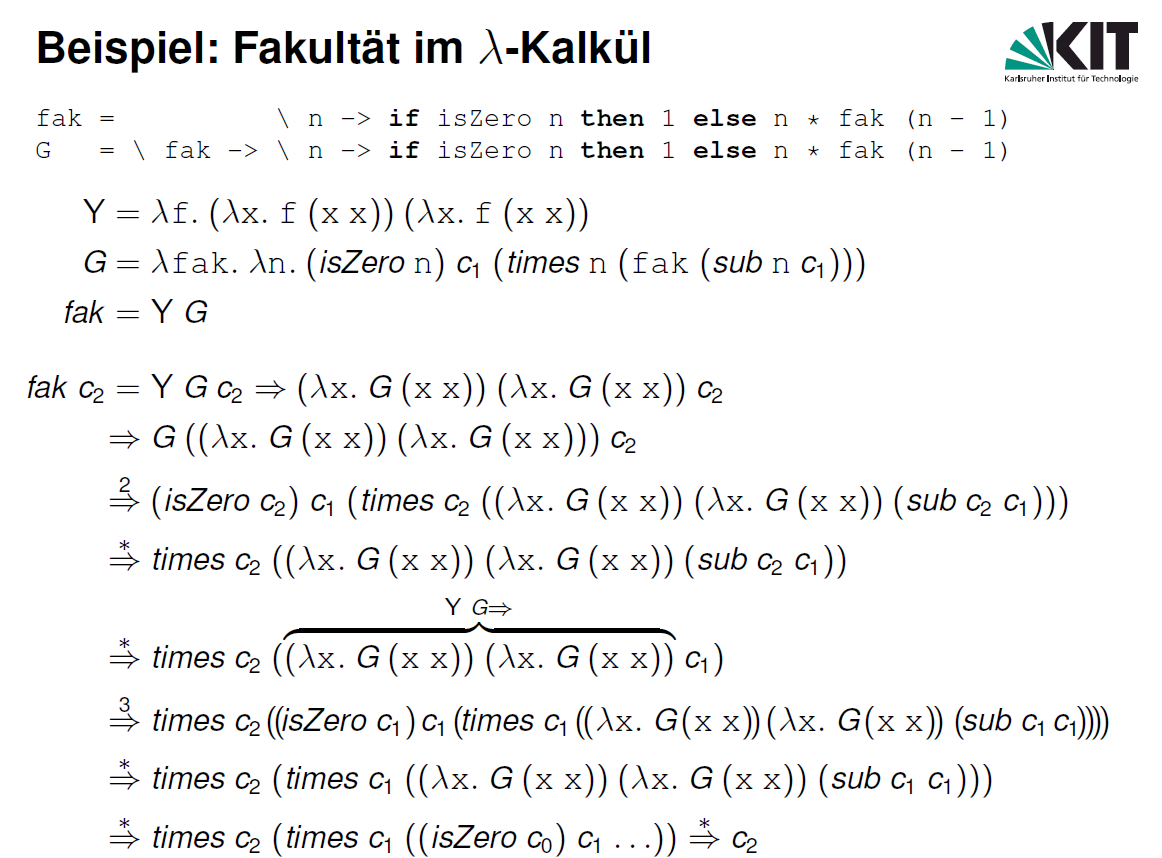


### Rekursion

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

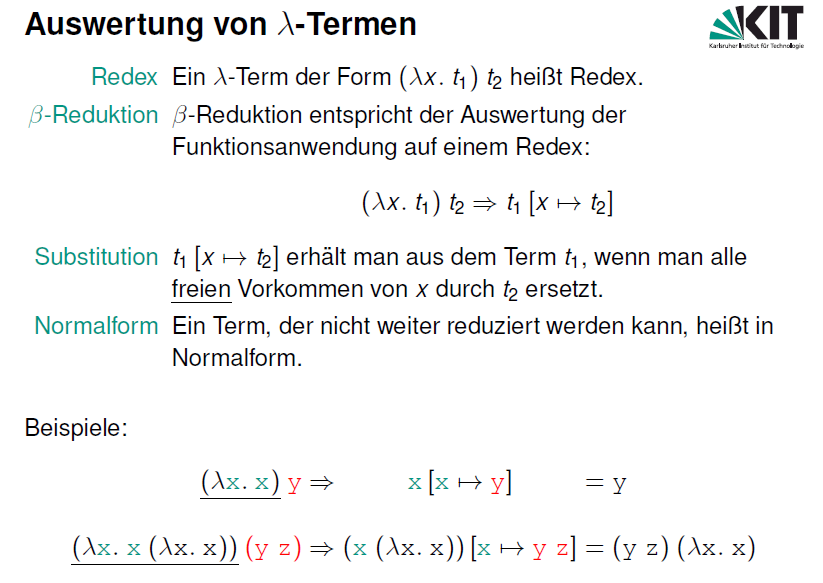
2) Erstelle eine erweiterte Funktion G als erste Funktion + Lamda für sich selbst

3) Rekursive Funktion = Y G

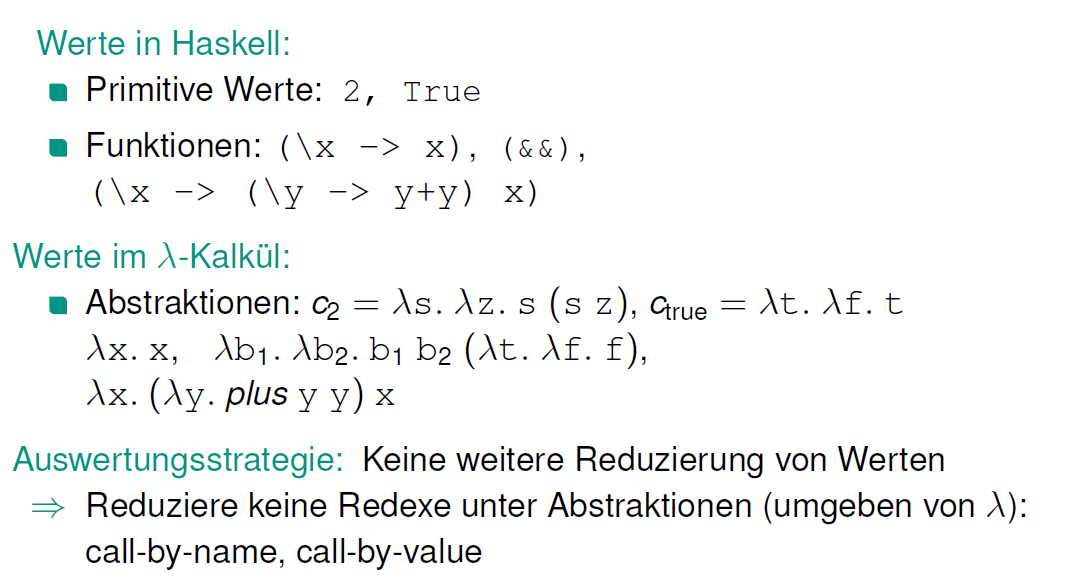


## Auswertungstrategien

### Auswertung

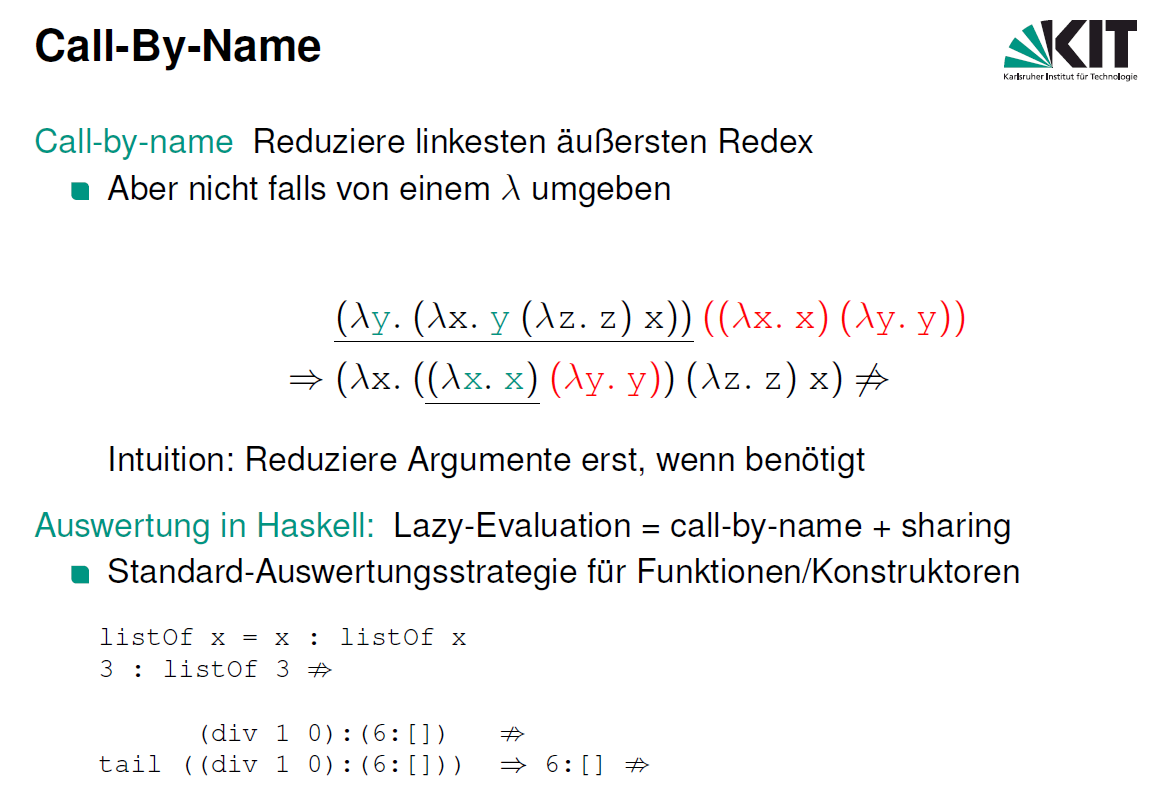


### Wert



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

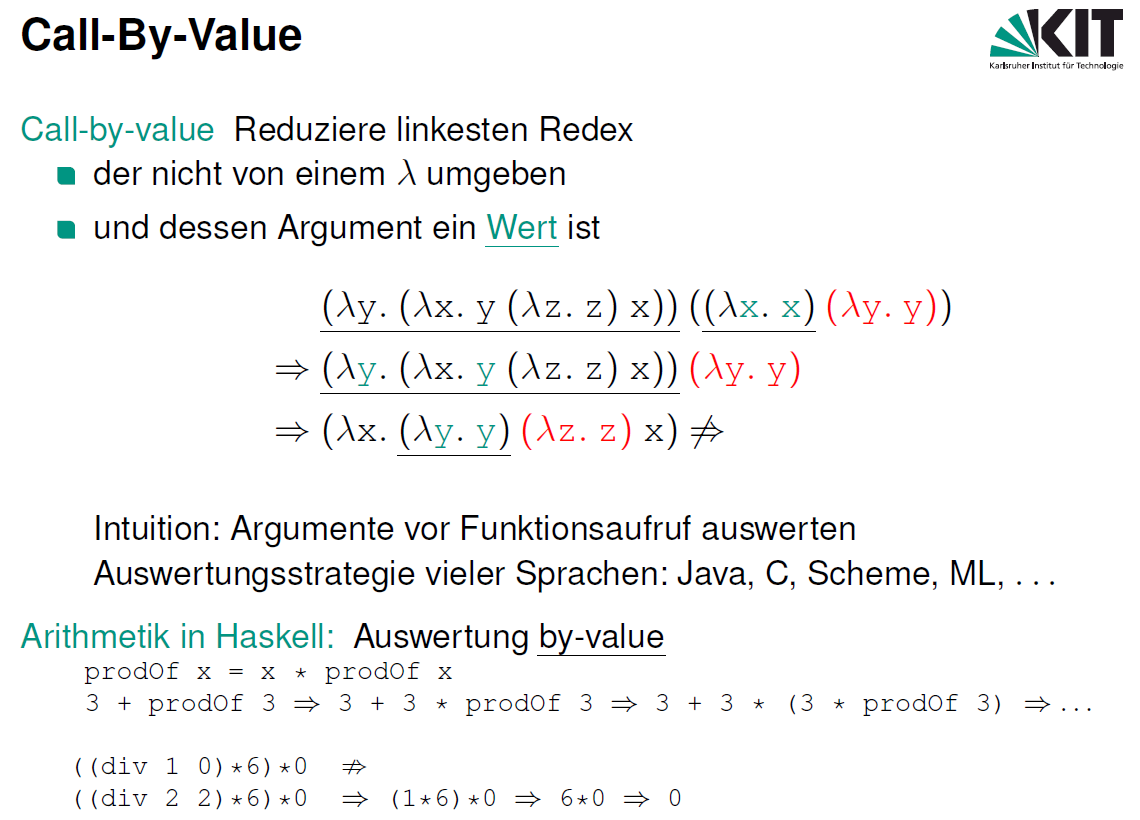


Nur die linke **\y** wird reduziert, da es keine äußere Lambda gibt.

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

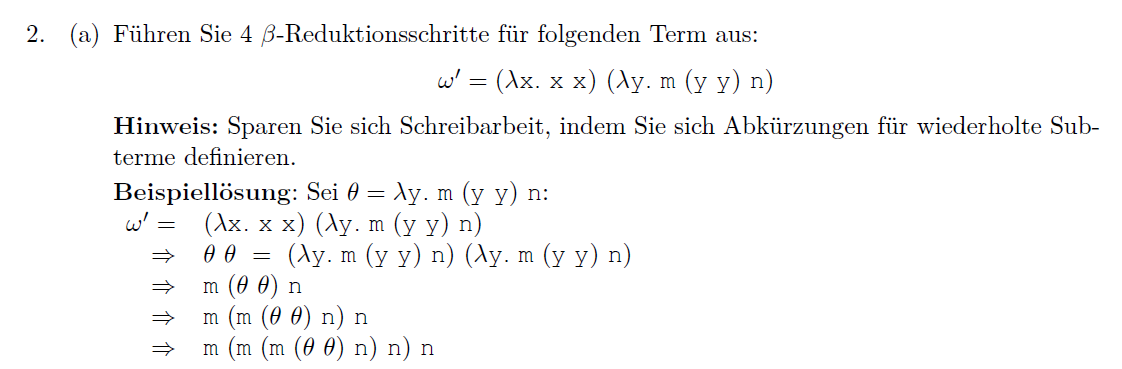
-------- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ CBN, NRF

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

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

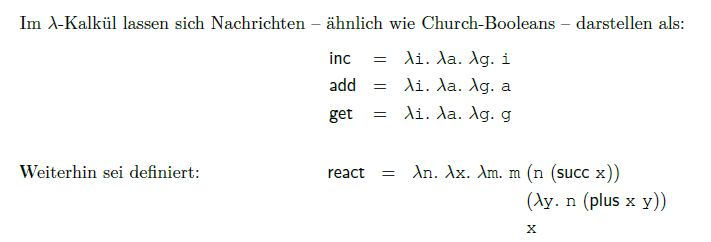
kein CBN und kein CBV (da alles von Lambda umgeben)

## Unendliche Reduktion

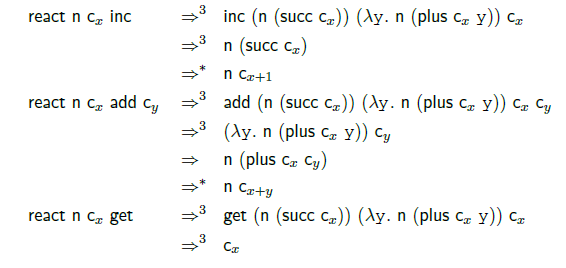


## Zähler (SS16 A4)

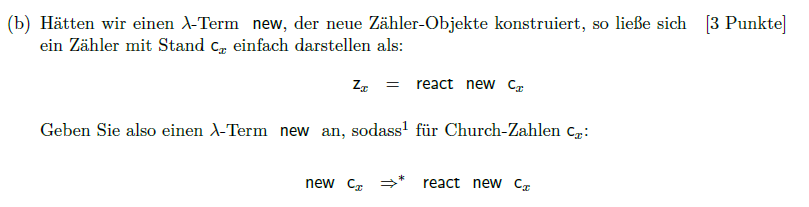
### Gegeben

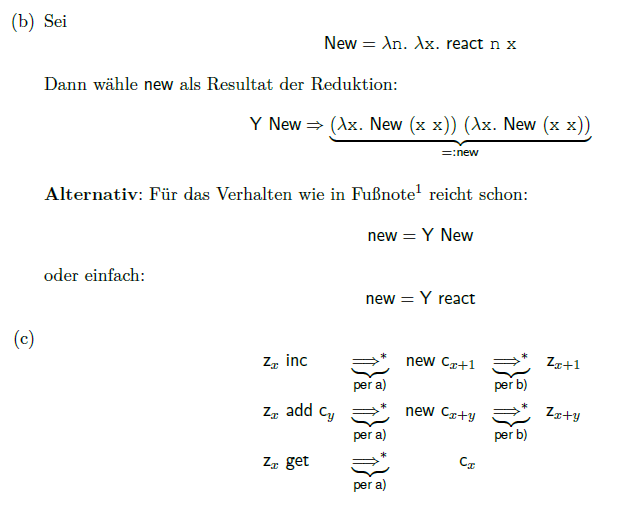


### Reduktion zeigen

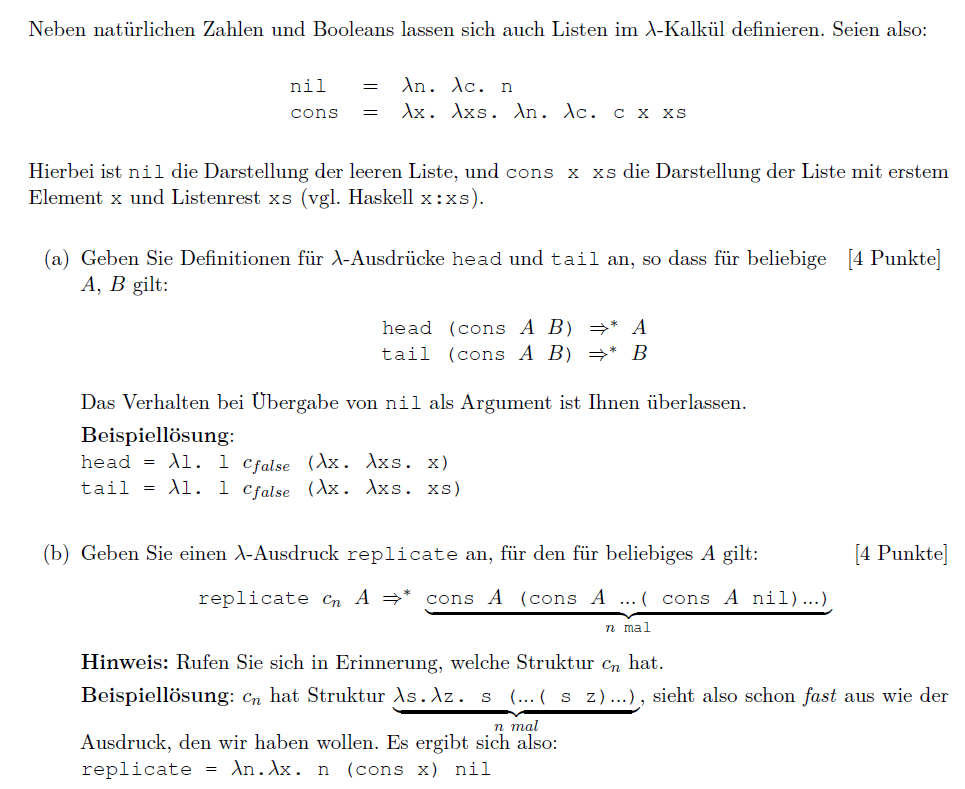


### „new“ Term mit Y Kombinator





## Listen



### Meine Lösung

head = \a. a c\_false c\_true

tail = \a. a c\_false c\_true

head (cons A B) = (cons A B) c\_false c\_true =

((\x. \xs. \n. \c. c x xs) A B) c\_false c \_true =

(\n. \c. c A B) c\_false c\_true =  
c\_true A B = A