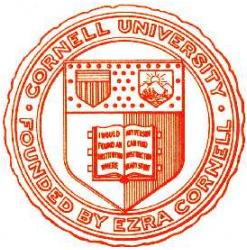


# Automatisierte Logik und Programmierung

## Einheit 5

### Der $\lambda$ -Kalkül



1. Syntax
2. Operationale Semantik
3. Schließen über Berechnung
4. Ausdruckskraft

# AUSWERTUNG VON TERMEN

- **Prädikatenlogik und Evidenzen verwenden Termsprache**
  - Prädikatenlogik: uninterpretierte Variablen- und Funktionssymbole
  - Evidenz: Terme der Prädikatenlogik und Konstrukte mit Bedeutung  
 $\lambda x.b$ ,  $f(a)$ ,  $(a, b)$ ,  $x.1$ ,  $x.2$ ,  $\text{inl}(a)$ ,  $\text{inr}(b)$ ,  $\text{case } x \text{ of } \dots$
- **Verschiedene Terme können die gleiche Bedeutung haben**
  - Für  $f = \lambda x.b$  ist  $f(a) = b[a/x]$  und  $\lambda x.f(x)$  ist dasselbe wie  $f$
  - $(a, b).1 = a$ ,  $(a, b).2 = b$  und das Paar  $(x.1, x.2)$  ist dasselbe wie  $x$
  - $\text{case } \text{inl}(a) \text{ of } \text{inl}(y) \rightarrow s \mid \text{inr}(z) \rightarrow t$  ist dasselbe wie  $s[a/y]$
  - Terme auf rechter und linker Seite haben jeweils denselben Wert
- **Termauswertung kann als Kalkül beschrieben werden**
  - Berechnungskalkül interpretiert Terme als Programmiersprache und kalkuliert Wert schrittweise durch Termersetzung (**Rewriting**)
  - Beweiskalkül beschreibt Effekte der Berechnungsschritte als Gleichheiten in einer Berechnungslogik
  - Die Grundform beider Kalküle ist der  **$\lambda$ -Kalkül**

## Logisches Schließen über den Wert von Termen

- **Einfacher mathematischer Mechanismus**
  - Funktionen werden **definiert** und **angewandt**
  - Beschreibung des Funktionsverhaltens wird zum **Namen** der Funktion
  - Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

## Logisches Schließen über den Wert von Termen

- **Einfacher mathematischer Mechanismus**
  - Funktionen werden **definiert** und **angewandt**
  - Beschreibung des Funktionsverhaltens wird zum **Namen** der Funktion
  - Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten
- **Leicht zu verstehende Basiskonzepte**
  1. Definition einer Funktion:
$$f(x) = 2*x+3$$

## Logisches Schließen über den Wert von Termen

- **Einfacher mathematischer Mechanismus**

- Funktionen werden **definiert** und **angewandt**
- Beschreibung des Funktionsverhaltens wird zum **Namen** der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

- **Leicht zu verstehende Basiskonzepte**

1. **Definition** einer Funktion:

$$f \hat{=} x \mapsto 2*x+3$$

Name der Funktion irrelevant für Beschreibung des Verhaltens

## Logisches Schließen über den Wert von Termen

- **Einfacher mathematischer Mechanismus**

- Funktionen werden **definiert** und **angewandt**
- Beschreibung des Funktionsverhaltens wird zum **Namen** der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

- **Leicht zu verstehende Basiskonzepte**

1. **Definition** einer Funktion:

$$f \hat{=} \lambda x. 2*x+3$$

**Abstraktion** von x

Name der Funktion irrelevant für Beschreibung des Verhaltens

## Logisches Schließen über den Wert von Termen

- **Einfacher mathematischer Mechanismus**

- Funktionen werden **definiert** und **angewandt**
- Beschreibung des Funktionsverhaltens wird zum **Namen** der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

- **Leicht zu verstehende Basiskonzepte**

1. **Definition** einer Funktion:

$$f \hat{=} \lambda x. 2*x+3$$

**$\lambda$ -Abstraktion**

Name der Funktion irrelevant für Beschreibung des Verhaltens

## Logisches Schließen über den Wert von Termen

- **Einfacher mathematischer Mechanismus**

- Funktionen werden **definiert** und **angewandt**
- Beschreibung des Funktionsverhaltens wird zum **Namen** der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

- **Leicht zu verstehende Basiskonzepte**

1. **Definition** einer Funktion:

$$f \hat{=} \lambda x. 2 * x + 3$$

Name der Funktion irrelevant für Beschreibung des Verhaltens

**$\lambda$ -Abstraktion**

2. **Anwendung** der Funktion (ohne Auswertung):

$$f(4) \hat{=} (\lambda x. 2 * x + 3)(4)$$

**Applikation**



## Logisches Schließen über den Wert von Termen

- **Einfacher mathematischer Mechanismus**

- Funktionen werden **definiert** und **angewandt**
- Beschreibung des Funktionsverhaltens wird zum **Namen** der Funktion
- Funktionswerte werden ausgerechnet durch **Einsetzen** von Werten

- **Leicht zu verstehende Basiskonzepte**

1. **Definition** einer Funktion:

$$f \hat{=} \lambda x. 2*x+3$$

**$\lambda$ -Abstraktion**

Name der Funktion irrelevant für Beschreibung des Verhaltens

2. **Anwendung** der Funktion (ohne Auswertung):

$$f(4) \hat{=} (\lambda x. 2*x+3)(4)$$

**Applikation**

3. **Auswertung** einer Funktionsanwendung (tatsächliches Ausrechnen):

$$(\lambda x. 2*x+3)(4) \xrightarrow{\beta} 2*4+3 \xrightarrow{*} 11$$

**Reduktion**

## Logisches Schließen über den Wert von Termen

- **Einfacher mathematischer Mechanismus**

- Funktionen werden definiert und angewandt
- Beschreibung des Funktionsverhaltens wird zum Namen der Funktion
- Funktionswerte werden ausgerechnet durch Einsetzen von Werten

- **Leicht zu verstehende Basiskonzepte**

1. Definition einer Funktion:

$$f \hat{=} \lambda x. 2 * x + 3$$

**$\lambda$ -Abstraktion**

Name der Funktion irrelevant für Beschreibung des Verhaltens

2. Anwendung der Funktion (ohne Auswertung):

$$f(4) \hat{=} (\lambda x. 2 * x + 3)(4)$$

**Applikation**

3. Auswertung einer Funktionsanwendung (tatsächliches Ausrechnen):

$$(\lambda x. 2 * x + 3)(4) \xrightarrow{\beta} 2 * 4 + 3 \xrightarrow{*} 11$$

**Reduktion**

- **Alle anderen Konstrukte können simuliert werden**

- Turingmächtige funktionale Programmiersprache  $\mapsto$  Lisp, ML, Haskell, ...

- **Einfache Programmiersprache:  $\lambda$ -Terme**

- Variablen  $x, y, z, x_0, y_0, \dots$

- $\lambda x . t$ , wobei  $x$  Variable und  $t$   $\lambda$ -Term

Vorkommen von  $x$  in  $t$  werden **gebunden**

- $f t$ , wobei  $t$  und  $f$   $\lambda$ -Terme

- $(t)$ , wobei  $t$   $\lambda$ -Term

**$\lambda$ -Abstraktion**

**Applikation**

## ● Einfache Programmiersprache: $\lambda$ -Terme

– Variablen  $x, y, z, x_0, y_0, \dots$

–  $\lambda x . t$ , wobei  $x$  Variable und  $t$   $\lambda$ -Term

Vorkommen von  $x$  in  $t$  werden **gebunden**

–  $f t$ , wobei  $t$  und  $f$   $\lambda$ -Terme

–  $(t)$ , wobei  $t$   $\lambda$ -Term

**$\lambda$ -Abstraktion**

**Applikation**

## ● Prioritätskonventionen sparen Klammern

– Applikation bindet stärker als  $\lambda$ -Abstraktion

$$\lambda x . f t \hat{=} \lambda x . (f t)$$

– Applikation ist links-assoziativ:

$$f t_1 t_2 \hat{=} (f t_1) t_2$$

## ● Einfache Programmiersprache: $\lambda$ -Terme

– Variablen  $x, y, z, x_0, y_0, \dots$

–  $\lambda x . t$ , wobei  $x$  Variable und  $t$   $\lambda$ -Term

**$\lambda$ -Abstraktion**

Vorkommen von  $x$  in  $t$  werden **gebunden**

–  $f t$ , wobei  $t$  und  $f$   $\lambda$ -Terme

**Applikation**

–  $(t)$ , wobei  $t$   $\lambda$ -Term

## ● Prioritätskonventionen sparen Klammern

– Applikation bindet stärker als  $\lambda$ -Abstraktion

$$\lambda x . f t \hat{=} \lambda x . (f t)$$

– Applikation ist links-assoziativ:

$$f t_1 t_2 \hat{=} (f t_1) t_2$$

## ● Beispiele für $\lambda$ -Terme

–  $x$

Symbole sind immer Variablen

–  $\lambda f . \lambda x . f x$

Anwendung einer Funktion

–  $\lambda f . \lambda g . \lambda x . f g (g x)$

Funktionen höherer Ordnung

–  $x x$

Selbstanwendung

–  $(\lambda x . x x) (\lambda x . x x)$

Fixpunkt

## ● Einfache Programmiersprache: $\lambda$ -Terme

– Variablen  $x, y, z, x_0, y_0, \dots$

–  $\lambda x . t$ , wobei  $x$  Variable und  $t$   $\lambda$ -Term

**$\lambda$ -Abstraktion**

Vorkommen von  $x$  in  $t$  werden **gebunden**

–  $f t$ , wobei  $t$  und  $f$   $\lambda$ -Terme

**Applikation**

–  $(t)$ , wobei  $t$   $\lambda$ -Term

## ● Prioritätskonventionen sparen Klammern

– Applikation bindet stärker als  $\lambda$ -Abstraktion

$$\lambda x . f t \hat{=} \lambda x . (f t)$$

– Applikation ist links-assoziativ:

$$f t_1 t_2 \hat{=} (f t_1) t_2$$

## ● Beispiele für $\lambda$ -Terme

(mit **impliziten** und ‘kosmetischen’ Klammern)

–  $x$

**Symbole sind immer Variablen**

–  $\lambda f . (\lambda x . (f (x)))$

**Anwendung einer Funktion**

–  $\lambda f . (\lambda g . (\lambda x . (((f g) (g x))))))$

**Funktionen höherer Ordnung**

–  $x(x)$

**Selbstanwendung**

–  $(\lambda x . x(x)) (\lambda x . x(x))$

**Fixpunkt**

- **Interpretation in Zielsprache ist zu kompliziert**

- Abbildung auf Funktionen in einfachem Universum nicht möglich
- Modellierung benötigt Theorie vollständiger Halbordnungen (CPOs)

- **Semantik von Termen wird operational erklärt**

- Entspricht Methodik beim Rechnen auf Zahlen:  $4*(3+2) = 4*5 = 20$
- $\lambda$ -Terme werden ausgewertet bis irreduzibler Term erreicht ist
- Irreduzible Terme werden als Werte angesehen

- **Berechnung ist rein syntaktischer Mechanismus**

- Auswertung ersetzt Funktionsparameter durch Funktionsargumente
- Formales Konzept: **Reduktion**  $(\lambda x . t) (b) \xrightarrow{\beta} t[b/x]$   
Ersetzen der freien Vorkommen der  $\lambda$ -Variablen  $x$  durch den  $\lambda$ -Term  $b$
- Benötigt Erweiterung des Substitutionskonzepts auf  $\lambda$ -Terme

# VORKOMMEN VON VARIABLEN IN $\lambda$ -TERMEN

- **Vorkommen der Variablen  $x$  im Term  $t$ , informal**
  - **Gebunden**:  $x$  erscheint im Bindungsbereich einer  $\lambda$ -Abstraktion  $\lambda x$
  - **Frei**:  $x$  kommt in  $t$  vor, ohne gebunden zu sein
  - $t$  heißt **geschlossen** falls  $t$  keine freien Variablen enthält
  - $t[x_1, \dots, x_n]$ : Term  $t$  hat mögliche freie Vorkommen von  $x_1, \dots, x_n$

- **Präzise, induktive Definition**

$x$  die Variable  $x$  kommt frei vor;  $y \neq x$  kommt nicht vor

$\lambda x.t$ : beliebige Vorkommen von  $x$  in  $t$  werden gebunden

Vorkommen von  $y \neq x$  in  $t$  bleiben unverändert

$f t$  freie Vorkommen von  $x$  in  $t$  bleiben frei

$(t)$  gebundene Vorkommen von  $x$  bleiben gebunden

$$\lambda f . \lambda x . \underbrace{(\lambda z . f \ x \ z)}_{x \text{ frei}} \ x$$

$x$  gebunden



# SUBSTITUTION $u[t/x]$ FORMAL

- Anwendung einer endlichen Abbildung  $\sigma = [t/x]$

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

# SUBSTITUTION $u[t/x]$ FORMAL

- Anwendung einer endlichen Abbildung  $\sigma = [t/x]$

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- Substitution und Reduktion am Beispiel

$$(\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x)$$

# SUBSTITUTION $u[t/x]$ FORMAL

- **Anwendung einer endlichen Abbildung  $\sigma = [t/x]$**

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- **Substitution und Reduktion am Beispiel**

$$\begin{aligned} & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\ = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x)) \quad (\text{mit impliziten Klammern}) \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

- **Anwendung einer endlichen Abbildung  $\sigma = [t/x]$**

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- **Substitution und Reduktion am Beispiel**

$$\begin{aligned} & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\ = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x)) \quad (\text{Reduktion ersetzt } n) \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

- Anwendung einer endlichen Abbildung  $\sigma = [t/x]$

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- Substitution und Reduktion am Beispiel

$$\begin{aligned}
 & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\
 = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x))) \quad (\text{Reduktion ersetzt } n) \\
 \longrightarrow & [\lambda f . (\lambda x . ((n f) (f x)))] [(\lambda f . (\lambda x . x)) / n] \quad (8 \text{ Substitutionsschritte})
 \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

- **Anwendung einer endlichen Abbildung  $\sigma = [t/x]$**

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- **Substitution und Reduktion am Beispiel**

$$\begin{aligned}
 & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\
 = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x)) \quad (\text{Reduktion ersetzt } n) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda f . (\lambda x . x)) f) (f x)) \quad (8 \text{ Substitutionsschritte})
 \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

- **Anwendung einer endlichen Abbildung  $\sigma = [t/x]$**

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- **Substitution und Reduktion am Beispiel**

$$\begin{aligned}
 & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\
 = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x))) \quad (\text{Reduktion ersetzt } n) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda f . (\lambda x . x)) f) (f x)) \quad (8 \text{ Substitutionsschritte})
 \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

- **Anwendung einer endlichen Abbildung  $\sigma = [t/x]$**

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- **Substitution und Reduktion am Beispiel**

$$\begin{aligned}
 & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\
 = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x))) \quad (\text{Reduktion ersetzt } n) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda f . (\lambda x . x)) f) (f x)) \quad (8 \text{ Substitutionsschritte}) \\
 \longrightarrow & \lambda f . (\lambda x . ([\lambda x . x][f / f]) (f x))
 \end{aligned}$$



# SUBSTITUTION $u[t/x]$ FORMAL

- **Anwendung einer endlichen Abbildung  $\sigma = [t/x]$**

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- **Substitution und Reduktion am Beispiel**

$$\begin{aligned}
 & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\
 = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x))) \quad (\text{Reduktion ersetzt } n) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda f . (\lambda x . x)) f) (f x)) \quad (8 \text{ Substitutionsschritte}) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda x . x) (f x)))
 \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

- **Anwendung einer endlichen Abbildung  $\sigma = [t/x]$**

$[x][t/x] = t$	$[x][t/y] = x$	$(y \neq x)$
$[\lambda x . u][t/x] = \lambda x . u$		
$[\lambda x . u][t/y] = [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y] = \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma = f\sigma u\sigma$	$[(u)]\sigma = (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- **Substitution und Reduktion am Beispiel**

$$\begin{aligned} & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\ = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x))) \quad (\text{Reduktion ersetzt } n) \\ \longrightarrow & \lambda f . (\lambda x . ((\lambda f . (\lambda x . x)) f) (f x)) \quad (8 \text{ Substitutionsschritte}) \\ \longrightarrow & \lambda f . (\lambda x . ((\lambda x . x) (f x))) \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

- **Anwendung einer endlichen Abbildung  $\sigma = [t/x]$**

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- **Substitution und Reduktion am Beispiel**

$$\begin{aligned}
 & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\
 = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x))) \quad (\text{Reduktion ersetzt } n) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda f . (\lambda x . x)) f) (f x)) \quad (8 \text{ Substitutionsschritte}) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda x . x) (f x))) \\
 \longrightarrow & \lambda f . (\lambda x . ([x] [f x / x]))
 \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

- Anwendung einer endlichen Abbildung  $\sigma = [t/x]$

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- Substitution und Reduktion am Beispiel

$$\begin{aligned}
 & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\
 = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x)) \quad (\text{Reduktion ersetzt } n) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda f . (\lambda x . x)) f) (f x)) \quad (8 \text{ Substitutionsschritte}) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda x . x) (f x))) \\
 \longrightarrow & \lambda f . (\lambda x . f x)
 \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

- **Anwendung einer endlichen Abbildung  $\sigma = [t/x]$**

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

- **Substitution und Reduktion am Beispiel**

$$\begin{aligned}
 & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\
 = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x))) \quad (\text{Reduktion ersetzt } n) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda f . (\lambda x . x)) f) (f x)) \quad (8 \text{ Substitutionsschritte}) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda x . x) (f x))) \\
 \longrightarrow & \lambda f . \lambda x . f x
 \end{aligned}$$

# SUBSTITUTION $u[t/x]$ FORMAL

## • Anwendung einer endlichen Abbildung $\sigma = [t/x]$

$[x][t/x]$	$= t$	$[x][t/y]$	$= x$	$(y \neq x)$
$[\lambda x . u][t/x]$	$= \lambda x . u$			
$[\lambda x . u][t/y]$	$= [\lambda z . u[z/x]][t/y]^*$	$[\lambda x . u][t/y]$	$= \lambda x . [u[t/y]]^{**}$	
$[f u]\sigma$	$= f\sigma u\sigma$	$[(u)]\sigma$	$= (u\sigma)$	

\*:  $y \neq x$ ,  $y$  frei in  $u$ ,  $x$  frei in  $t$ ,  $z$  neue Variable

\*\* :  $y \neq x$ ,  $y$  nicht frei in  $u$  oder  $x$  nicht frei in  $t$

## • Substitution und Reduktion am Beispiel

$$\begin{aligned}
 & (\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \\
 = & (\lambda n . (\lambda f . (\lambda x . ((n f) (f x)))) (\lambda f . (\lambda x . x))) \quad (\text{Reduktion ersetzt } n) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda f . (\lambda x . x)) f) (f x)) \quad (8 \text{ Substitutionsschritte}) \\
 \longrightarrow & \lambda f . (\lambda x . ((\lambda x . x) (f x))) \\
 \longrightarrow & \lambda f . \lambda x . f x
 \end{aligned}$$



$$(\lambda n . \lambda f . \lambda x . n f (f x)) (\lambda f . \lambda x . x) \xrightarrow{3} \lambda f . \lambda x . f x$$

# REDUKTION ALS FORMALISMUS – WICHTIGE BEGRIFFE

- **$\alpha$ -Konversion:**

- Umbenennung gebundener Variablen in Termen (erhält Bedeutung)
- Ersetze Teilterms der Gestalt  $\lambda x . u$  durch  $\lambda z . u[z/x]$  ( $z$  neue Variable)

- **Kongruenz  $t \cong u$  (“ $t$  und  $u$  sind  $\alpha$ -konvertibel”)**

- $u$  entsteht aus  $t$  durch  $\alpha$ -Konversion

- **Reduktion  $(\lambda x . u) s \xrightarrow{\beta} u[s/x]$ :**

- Ersetze Teilterm der Gestalt  $\underbrace{(\lambda x . u)}_{\text{Redex}} s$  durch  $\underbrace{u[s/x]}_{\text{Kontraktum}}$

- **Reduzierbarkeit  $t \xrightarrow{*} u$ :**

- $u$  entsteht aus  $t$  durch endlich viele Reduktionen und  $\alpha$ -Konversionen

- **Normalform:**

- Term  $u$ , der nicht weiter reduzierbar (**irreduzibel**) ist
- **Wert** von  $t$ : Normalform  $u$  mit  $t \xrightarrow{*} u$

# REGELN FÜR KONVERSION UND REDUKTION

- **$\alpha$ -Konversion:**  $\lambda x . u \cong \lambda z . u[z/x]$  ( $z$  neu)
  - $\mu$ -Konversion:  $f t \cong f u$ , falls  $t \cong u$
  - $\nu$ -Konversion:  $f t \cong g t$ , falls  $f \cong g$
  - $\xi$ -Konversion:  $\lambda x . t \cong \lambda x . u$ , falls  $t \cong u$
  - $\rho$ -Konversion:  $t \cong t$
  - $\sigma$ -Konversion:  $t \cong u$ , falls  $u \cong t$
  - $\tau$ -Konversion:  $t \cong u$ , falls  $t \cong s$  und  $s \cong u$
- **$\beta$ -Reduktion:**  $(\lambda x . u) s \longrightarrow u[s/x]$ 
  - $\mu$ -Reduktion:  $f t \longrightarrow f u$ , falls  $t \longrightarrow u$
  - $\nu$ -Reduktion:  $f t \longrightarrow g t$ , falls  $f \longrightarrow g$
  - $\xi$ -Reduktion:  $\lambda x . t \longrightarrow \lambda x . u$ , falls  $t \longrightarrow u$
  - $\tau$ -Reduktion:  $t \longrightarrow u$ , falls  $t \longrightarrow s$  und  $s \cong u$  oder  $t \cong s$  und  $s \longrightarrow u$
- **Reduzierbarkeit:**  $t \xrightarrow{*} u$ , falls  $t \xrightarrow{n} u$  für ein  $n \in \mathbb{N}$ 
  - $t \xrightarrow{0} u$ , falls  $t \cong u$
  - $t \xrightarrow{n+1} u$ , falls  $t \longrightarrow s$  und  $s \xrightarrow{n} u$



# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$(\lambda f. \lambda x. f \ x \ (f \ f)) \ (\lambda x. \lambda y. y)$

# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$$\begin{aligned} & (\lambda f. \lambda x. f \ x \ (f \ f)) \ (\lambda x. \lambda y. \ y) \\ \longrightarrow & \lambda x. \ (\lambda x. \lambda y. \ y) \ x \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y)) \end{aligned}$$

# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$(\lambda f. \lambda x. f \ x \ (f \ f)) \ (\lambda x. \lambda y. \ y)$

$\longrightarrow \lambda x. \ (\lambda x. \lambda y. \ y) \ x \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$

$\longrightarrow \lambda x. \ (\lambda y. \ y) \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$

# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$(\lambda f. \lambda x. f \ x \ (f \ f)) \ (\lambda x. \lambda y. \ y)$   
→  $\lambda x. \ (\lambda x. \lambda y. \ y) \ x \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$   
→  $\lambda x. \ (\lambda y. \ y) \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$   
→  $\lambda x. \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$

# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$(\lambda f. \lambda x. f \ x \ (f \ f)) \ (\lambda x. \lambda y. \ y)$   
→  $\lambda x. \ (\lambda x. \lambda y. \ y) \ x \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$   
→  $\lambda x. \ (\lambda y. \ y) \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$   
→  $\lambda x. \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$   
→  $\lambda x. \ \lambda y. \ y$

# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$(\lambda f . \lambda x . f \ x \ (f \ f)) \ (\lambda x . \lambda y . y)$   
→  $\lambda x . (\lambda x . \lambda y . y) \ x \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . (\lambda y . y) \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . \lambda y . y$

## Alternative Reduktionsreihenfolgen sind möglich

$(\lambda f . \lambda x . f \ x \ (f \ f)) \ (\lambda x . \lambda y . y)$

# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$(\lambda f . \lambda x . f \ x \ (f \ f)) \ (\lambda x . \lambda y . y)$   
→  $\lambda x . (\lambda x . \lambda y . y) \ x \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . (\lambda y . y) \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . \lambda y . y$

## Alternative Reduktionsreihenfolgen sind möglich

$(\lambda f . \lambda x . f \ x \ (f \ f)) \ (\lambda x . \lambda y . y)$   
→  $\lambda x . (\lambda x . \lambda y . y) \ x \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$

# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$(\lambda f. \lambda x. f \ x \ (f \ f)) \ (\lambda x. \lambda y. \ y)$   
→  $\lambda x. \ (\lambda x. \lambda y. \ y) \ x \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$   
→  $\lambda x. \ (\lambda y. \ y) \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$   
→  $\lambda x. \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$   
→  $\lambda x. \ \lambda y. \ y$

## Alternative Reduktionsreihenfolgen sind möglich

$(\lambda f. \lambda x. f \ x \ (f \ f)) \ (\lambda x. \lambda y. \ y)$   
→  $\lambda x. \ (\lambda x. \lambda y. \ y) \ x \ ((\lambda x. \lambda y. \ y) \ (\lambda x. \lambda y. \ y))$   
→  $\lambda x. \ (\lambda x. \lambda y. \ y) \ x \ (\lambda y. \ y)$   
→



# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$(\lambda f . \lambda x . f \ x \ (f \ f)) \ (\lambda x . \lambda y . y)$   
→  $\lambda x . (\lambda x . \lambda y . y) \ x \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . (\lambda y . y) \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . \lambda y . y$

## Alternative Reduktionsreihenfolgen sind möglich

$(\lambda f . \lambda x . f \ x \ (f \ f)) \ (\lambda x . \lambda y . y)$   
→  $\lambda x . (\lambda x . \lambda y . y) \ x \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . (\lambda x . \lambda y . y) \ x \ (\lambda y . y)$   
→  $\lambda x . (\lambda y . y) \ (\lambda y . y)$

# REDUKTION IST NICHT DETERMINISTISCH

## Naheliegende Reduktionsreihenfolge

$(\lambda f . \lambda x . f \ x \ (f \ f)) \ (\lambda x . \lambda y . y)$   
→  $\lambda x . (\lambda x . \lambda y . y) \ x \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . (\lambda y . y) \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . \lambda y . y$

## Alternative Reduktionsreihenfolgen sind möglich

$(\lambda f . \lambda x . f \ x \ (f \ f)) \ (\lambda x . \lambda y . y)$   
→  $\lambda x . (\lambda x . \lambda y . y) \ x \ ((\lambda x . \lambda y . y) \ (\lambda x . \lambda y . y))$   
→  $\lambda x . (\lambda x . \lambda y . y) \ x \ (\lambda y . y)$   
→  $\lambda x . (\lambda y . y) \ (\lambda y . y)$   
→  $\lambda x . \lambda y . y$

## Bekommt man immer das gleiche Ergebnis?

- **Bedeutung von  $\lambda$ -Termen ist ihr Wert**
  - **Normalform**: Term ohne keine Redizes als Teilterme
  - **$u$  Normalform von  $t$** :  $u$  in Normalform und  $t \xrightarrow{*} u$
  - **$t$  normalisierbar**: es gibt eine Normalform  $u$  von  $t$

- **Bedeutung von  $\lambda$ -Termen ist ihr Wert**
  - **Normalform**: Term ohne keine Redizes als Teilterme
  - **$u$  Normalform von  $t$** :  $u$  in Normalform und  $t \xrightarrow{*} u$
  - **$t$  normalisierbar**: es gibt eine Normalform  $u$  von  $t$
- **Hat jeder  $\lambda$ -Term eine Normalform?**

# SEMANTIK VON $\lambda$ -TERMEN PRÄZISIERT

- **Bedeutung von  $\lambda$ -Termen ist ihr Wert**
  - **Normalform**: Term ohne keine Redizes als Teilterme
  - **$u$  Normalform von  $t$** :  $u$  in Normalform und  $t \xrightarrow{*} u$
  - **$t$  normalisierbar**: es gibt eine Normalform  $u$  von  $t$
- **Hat jeder  $\lambda$ -Term eine Normalform?**
  - Nein**:  $(\lambda x. x x) (\lambda x. x x)$  ist nicht normalisierbar
  - Terminierung von  $\lambda$ -Programmen ist nicht garantiert

- **Bedeutung von  $\lambda$ -Termen ist ihr Wert**
  - **Normalform**: Term ohne keine Redizes als Teilterme
  - **$u$  Normalform von  $t$** :  $u$  in Normalform und  $t \xrightarrow{*} u$
  - **$t$  normalisierbar**: es gibt eine Normalform  $u$  von  $t$
- **Hat jeder  $\lambda$ -Term eine Normalform?**
  - Nein**:  $(\lambda x. x x) (\lambda x. x x)$  ist nicht normalisierbar
  - Terminierung von  $\lambda$ -Programmen ist nicht garantiert
- **Haben normalisierbare  $\lambda$ -Terme eindeutige Werte?**
  - Notwendig für Einsatz des  $\lambda$ -Kalküls als Programmiersprache
  - Führt jede Reduktionsfolge zu einer Normalform?
    - Wenn nein, kann man eine Normalform systematisch finden?
  - Ist die Normalform eines  $\lambda$ -Terms eindeutig?

# REDUKTION NORMALISIERBARER $\lambda$ -TERME

- **Führt jede Reduktionsfolge zu einer Normalform?**

# REDUKTION NORMALISIERBARER $\lambda$ -TERME

- **Führt jede Reduktionsfolge zu einer Normalform?**

**Nein:** Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren



# REDUKTION NORMALISIERBARER $\lambda$ -TERME

- **Führt jede Reduktionsfolge zu einer Normalform?**

**Nein:** Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

# REDUKTION NORMALISIERBARER $\lambda$ -TERME

- **Führt jede Reduktionsfolge zu einer Normalform?**

**Nein:** Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

$\longrightarrow (\lambda y. y) (\lambda x. x)$

# REDUKTION NORMALISIERBARER $\lambda$ -TERME

- **Führt jede Reduktionsfolge zu einer Normalform?**

**Nein:** Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

$\longrightarrow (\lambda y. y) (\lambda x. x)$

$\longrightarrow \lambda x. x$

- **Führt jede Reduktionsfolge zu einer Normalform?**

**Nein:** Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

→  $(\lambda y. y) (\lambda x. x)$

→  $\lambda x. x$

Eine “innermost” Strategie führt bei diesem Term nicht zur Normalform

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

# REDUKTION NORMALISIERBARER $\lambda$ -TERME

- **Führt jede Reduktionsfolge zu einer Normalform?**

**Nein:** Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

→  $(\lambda y. y) (\lambda x. x)$

→  $\lambda x. x$

Eine “innermost” Strategie führt bei diesem Term nicht zur Normalform

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

$\xrightarrow{*} (\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

# REDUKTION NORMALISIERBARER $\lambda$ -TERME

## ● Führt jede Reduktionsfolge zu einer Normalform?

**Nein:** Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

$\longrightarrow (\lambda y. y) (\lambda x. x)$

$\longrightarrow \lambda x. x$

Eine “innermost” Strategie führt bei diesem Term nicht zur Normalform

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

$\xrightarrow{*} (\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

$\xrightarrow{*} (\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

⋮

⋮

# REDUKTION NORMALISIERBARER $\lambda$ -TERME

- **Führt jede Reduktionsfolge zu einer Normalform?**

**Nein:** Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

$\longrightarrow (\lambda y. y) (\lambda x. x)$

$\longrightarrow \lambda x. x$

Eine “innermost” Strategie führt bei diesem Term nicht zur Normalform

$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

$\xrightarrow{*} (\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

$\xrightarrow{*} (\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$

⋮

⋮

- **Kann man eine Normalform immer finden?**

# REDUKTION NORMALISIERBARER $\lambda$ -TERME

## ● Führt jede Reduktionsfolge zu einer Normalform?

**Nein:** Reduktionsfolgen normalisierbarer Terme müssen nicht terminieren

$$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$$
$$\longrightarrow (\lambda y. y) (\lambda x. x)$$
$$\longrightarrow \lambda x. x$$

Eine “innermost” Strategie führt bei diesem Term nicht zur Normalform

$$(\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$$
$$\xrightarrow{*} (\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$$
$$\xrightarrow{*} (\lambda x. \lambda y. y) ((\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x)) (\lambda x. x)$$
$$\vdots$$
$$\vdots$$

## ● Kann man eine Normalform immer finden?

– **Ja:** Reduktion des äußersten Redex (“**leftmost reduction**”) führt zur Normalform, wenn es eine gibt

Beweis: Curry & Feys 1958, p142



# EINDEUTIGKEIT VON NORMALFORMEN

- **Was heißt “eindeutig”?**

- Müssen Normalformen textlich identisch sein (wie  $4 = 4$ )?
- Oder sollen sie “nur” den gleichen Wert haben (wie  $\lambda x.x$  und  $\lambda y.y$ )?
- Gleichwertigkeit zu fordern ist semantisch sinnvoller

# EINDEUTIGKEIT VON NORMALFORMEN

- **Was heißt “eindeutig”?**

- Müssen Normalformen textlich identisch sein (wie  $4 = 4$ )?
- Oder sollen sie “nur” den gleichen Wert haben (wie  $\lambda x.x$  und  $\lambda y.y$ )?
- Gleichwertigkeit zu fordern ist semantisch sinnvoller

- **Semantische Gleichheit von  $\lambda$ -Termen ist extensional**

- $t = u$ : es gibt  $v$  mit  $t \xrightarrow{*} v$  und  $u \xrightarrow{*} v$  ( $t$  und  $u$  sind **konvertierbar**)
- Auch Terme, die nicht normalisierbar sind, können gleich sein

# EINDEUTIGKEIT VON NORMALFORMEN

- **Was heißt “eindeutig”?**

- Müssen Normalformen textlich identisch sein (wie  $4 = 4$ )?
- Oder sollen sie “nur” den gleichen Wert haben (wie  $\lambda x.x$  und  $\lambda y.y$ )?
- Gleichwertigkeit zu fordern ist semantisch sinnvoller

- **Semantische Gleichheit von  $\lambda$ -Termen ist extensional**

- $t = u$ : es gibt  $v$  mit  $t \xrightarrow{*} v$  und  $u \xrightarrow{*} v$  ( $t$  und  $u$  sind **konvertierbar**)
- Auch Terme, die nicht normalisierbar sind, können gleich sein

- **Ist die Normalform eines  $\lambda$ -Terms eindeutig?**

# EINDEUTIGKEIT VON NORMALFORMEN

- **Was heißt “eindeutig”?**

- Müssen Normalformen textlich identisch sein (wie  $4 = 4$ )?
- Oder sollen sie “nur” den gleichen Wert haben (wie  $\lambda x.x$  und  $\lambda y.y$ )?
- Gleichwertigkeit zu fordern ist semantisch sinnvoller

- **Semantische Gleichheit von  $\lambda$ -Termen ist extensional**

- $t = u$ : es gibt  $v$  mit  $t \xrightarrow{*} v$  und  $u \xrightarrow{*} v$  ( $t$  und  $u$  sind **konvertierbar**)
- Auch Terme, die nicht normalisierbar sind, können gleich sein

- **Ist die Normalform eines  $\lambda$ -Terms eindeutig?**

- **Ja**: Reduktion ist **konfluent** Beweis: Barendregt 1981 §3.2

Gilt  $t \xrightarrow{*} u$  und  $t \xrightarrow{*} v$ , so folgt  $u = v$  (**Church-Rosser Theorem**)

- Alle **Normalformen** eines  $\lambda$ -Terms sind **kongruent** und können (durch Konversionen) auf denselben Term reduziert werden
- Semantisch **gleiche** Terme haben **dieselben Normalformen** oder keine
- Normalformen, die **nicht kongruent** sind, sind **nicht semantisch gleich**

# SEMANTISCHE GLEICHHEIT

- **Ausdrucksstärker als Reduzierbarkeit**

- Benötigt keine Normalform für den Vergleich von Termen
  - Es reicht, mit Reduktionen/Konversionen den gleichen Term zu erreichen
- Auch Terme, die nicht normalisierbar sind, können verglichen werden

- **Grundlage des logischen Schließens über Berechnungen**

- Im (allgemeinen)  $\lambda$ -Kalkül kann automatische Normalisierung zu nicht-terminierenden Reduktionsfolgen führen
  - Für typisierbare  $\lambda$ -Termen gibt es immer eine terminierende Reduktionsstrategie
- Schließen über semantische Gleichheit braucht terminierende Reduktion
- Beweisregeln müssen einzelne Reduktionsschritte widerspiegeln

- **Der  $\lambda$ -Kalkül ist unabhängig von der Prädikatenlogik**

- Schließen über semantische Gleichheit benötigt keine logischen Regeln
- Beweissysteme können separat beschrieben und dann integriert werden

# REGELN ZUM SCHLIESSEN ÜBER GLEICHHEIT

- **$\beta$ -Reduktion**

- Gleichheit mit einem Redex entspricht Gleichheit mit Kontraktum

$$H \vdash (\lambda x.u) s = t$$

$$H \vdash u[s/x] = t \quad \text{reduction}$$

# REGELN ZUM SCHLIESSEN ÜBER GLEICHHEIT

## ● $\beta$ -Reduktion

- Gleichheit mit einem Redex entspricht Gleichheit mit Kontraktum

$$\begin{array}{l} H \vdash (\lambda x.u) s = t \\ H \vdash u[s/x] = t \end{array} \text{reduction}$$

## ● Kongruenzregeln

- Applikationen sind gleich, wenn Funktionen und Argumente gleich sind
- Funktionen sind gleich, wenn sie auf allen Argumenten gleich sind

$$\begin{array}{l} H \vdash f t = g u \\ H \vdash f = g \\ H \vdash t = u \end{array} \text{applyEq}$$

$$\begin{array}{l} H \vdash \lambda x.t = \lambda y.u \\ H, x':\mathbb{U} \vdash t[x'/x] = u[x'/y] \end{array} \text{lambdaEq}$$

*x' ist neue Variable, die in H nicht frei vorkommt*

# REGELN ZUM SCHLIESSEN ÜBER GLEICHHEIT

## • $\beta$ -Reduktion

- Gleichheit mit einem Redex entspricht Gleichheit mit Kontraktum

$$\begin{array}{l} H \vdash (\lambda x.u) s = t \\ H \vdash u[s/x] = t \end{array} \text{reduction}$$

## • Kongruenzregeln

- Applikationen sind gleich, wenn Funktionen und Argumente gleich sind
- Funktionen sind gleich, wenn sie auf allen Argumenten gleich sind

$$\begin{array}{l} H \vdash f t = g u \\ H \vdash f = g \\ H \vdash t = u \end{array} \text{applyEq}$$

$$\begin{array}{l} H \vdash \lambda x.t = \lambda y.u \\ H, x':\mathbb{U} \vdash t[x'/x] = u[x'/y] \end{array} \text{lambdaEq}$$

*x' ist neue Variable, die in H nicht frei vorkommt*

## • Reflexivität, Symmetrie, Transitivität, Substitution

$$H \vdash t = t \text{ reflexivity}$$

$$\begin{array}{l} H \vdash t = u \\ H \vdash u = t \end{array} \text{symmetry}$$

$$\begin{array}{l} H \vdash t = u \\ H \vdash t = s \\ H \vdash s = u \end{array} \text{transitivity s}$$

$$\begin{array}{l} H \vdash C[t/x] \\ H \vdash t = u \\ H \vdash C[u/x] \end{array} \text{subst } t=u$$



# SEQUENZENBEWEIS FÜR EINE REDUKTION

## ● Beweis einer einfachen Reduktion

$\vdash (\lambda f.\lambda x.f x(f f))(\lambda x.\lambda y.y)$	BY	reduction
1. $\vdash \lambda x. (\lambda x.\lambda y.y)x((\lambda x.\lambda y.y)(\lambda x.\lambda y.y)) = \lambda x.\lambda y.y$	BY	lambdaEq
1.1. $x:\mathbb{U} \vdash (\lambda x.\lambda y.y)x((\lambda x.\lambda y.y)(\lambda x.\lambda y.y)) = \lambda y.y$	BY	reduction
1.1.1. $x:\mathbb{U} \vdash (\lambda y.y)((\lambda x.\lambda y.y)(\lambda x.\lambda y.y)) = \lambda y.y$	BY	reduction
1.1.1.1. $x:\mathbb{U} \vdash (\lambda x.\lambda y.y)(\lambda x.\lambda y.y) = \lambda y.y$	BY	reduction
1.1.1.1.1. $x:\mathbb{U} \vdash \lambda y.y = \lambda y.y$	BY	reflexivity

# SEQUENZENBEWEIS FÜR EINE REDUKTION

## ● Beweis einer einfachen Reduktion

$\vdash (\lambda f.\lambda x.f x(f f))(\lambda x.\lambda y.y) = \lambda x.\lambda y.y$	BY	reduction
1. $\vdash \lambda x.(\lambda x.\lambda y.y)x((\lambda x.\lambda y.y)(\lambda x.\lambda y.y)) = \lambda x.\lambda y.y$	BY	lambdaEq
1.1. $x:\mathbb{U} \vdash (\lambda x.\lambda y.y)x((\lambda x.\lambda y.y)(\lambda x.\lambda y.y)) = \lambda y.y$	BY	reduction
1.1.1. $x:\mathbb{U} \vdash (\lambda y.y)((\lambda x.\lambda y.y)(\lambda x.\lambda y.y)) = \lambda y.y$	BY	reduction
1.1.1.1. $x:\mathbb{U} \vdash (\lambda x.\lambda y.y)(\lambda x.\lambda y.y) = \lambda y.y$	BY	reduction
1.1.1.1.1. $x:\mathbb{U} \vdash \lambda y.y = \lambda y.y$	BY	reflexivity

## ● Beweis einer semantischen Gleichheit

$\vdash (\lambda f.\lambda x.f x)(\lambda y.y) = (\lambda x.\lambda y.y)(\lambda z.z)$	BY	reduction
1. $\vdash \lambda x.(\lambda y.y) x = (\lambda x.\lambda y.y)(\lambda z.z)$	BY	symmetry
1.1. $\vdash (\lambda x.\lambda y.y)(\lambda z.z) = \lambda x.(\lambda y.y) x$	BY	reduction
1.1.1. $\vdash \lambda y.y = \lambda x.(\lambda y.y) x$	BY	lambdaEq
1.1.1.1. $x:\mathbb{U} \vdash x = (\lambda y.y) x$	BY	symmetry
1.1.1.1.1. $x:\mathbb{U} \vdash (\lambda y.y) x = x$	BY	reduction
1.1.1.1.1.1. $x:\mathbb{U} \vdash x = x$	BY	reflexivity

# EIGENSCHAFTEN DES BEWEISKALKÜLS

## ● Korrektheit

- Wenn  $\vdash u=v$  im erweiterten Sequenzenkalkül bewiesen werden kann, dann sind  $u$  und  $v$  semantisch gleich ( $u = v$ )
- *Die neuen Regeln sind korrekt in Bezug auf semantische Gleichheit*

## ● Vollständigkeit gilt nur in eingeschränktem Sinne

Jede (konkrete) Reduktionsfolge  $u \xrightarrow{*} t$  und  $v \xrightarrow{*} t$  kann in eine Folge von Beweisschritten für  $\vdash u=v$  übertragen werden

- $\beta$ -Reduktion wird von `reduction` abgedeckt
  - $\alpha$ - und  $\xi$ -Konversionen und -Reduktionen werden von `lambdaEq` erfaßt
  - $\mu$ - und  $\nu$ -Konversionen und -Reduktionen werden von `applyEq` erfaßt
  - `reflexivity`, `symmetry`, `transitivity`, `subst` decken den Rest ab
- Achtung! Es folgt nicht: “Gilt  $u = v$ , so ist  $\vdash u=v$  beweisbar” (siehe Folie 29)

## ● Kalkül kann auch definatorische Gleichheit verarbeiten

- Sind Konzepte durch definatorische Abkürzung erklärt, so ersetzt die Regel `unfold` die linke Seite der Definition durch die rechte

# VOM $\lambda$ -KALKÜL ZU ECHTEN PROGRAMMEN

- **$\lambda$ -Kalkül ist der Basismechanismus**
  - Die *Assemblersprache* funktionaler Programme
  - Nur die elementarsten Konstrukte sind vordefiniert
- **Programm- und Datenstrukturen werden codiert**
  - Berechnung auf  $\lambda$ -Ausdrücken muß *Effekte auf Struktur simulieren*  
(Analog zu konventionellen Computern, in denen alles als Bitmuster codiert wird)
- **Alle wichtigen Strukturen sind leicht codierbar**
  - Boolesche Operationen:  $T, F, \text{if } b \text{ then } s \text{ else } t$
  - Tupel / Projektionen:  $\langle s, t \rangle, p.1, p.2$
  - Injektion / Analyse:  $\text{inl}(a), \text{inr}(b), \text{case } x \text{ of } \text{inl}(a) \rightarrow s \mid \text{inr}(b) \rightarrow t$
  - Zahlen und arithmetische Operationen
  - Iteration oder Rekursion von Funktionen

**Der  $\lambda$ -Kalkül kann alle berechenbaren Funktionen repräsentieren**

# DARSTELLUNG BOOLESCHER OPERATOREN IM $\lambda$ -KALKÜL

## Wir brauchen zwei verschiedene Objekte und einen Test

$T \equiv \lambda x. \lambda y. x$  *Term für "true"*

$F \equiv \lambda x. \lambda y. y$  *Term für "false"*

$\text{if } b \text{ then } s \text{ else } t \equiv b s t$  *Term für Konditional*

## Beweis: Die Konditional(-darstellung) ist invers zu T und F

$\vdash \text{if } T \text{ then } s \text{ else } t = s$

$\vdash T s t = s$

$\vdash (\lambda x. \lambda y. x) s t = s$

$\vdash (\lambda y. s) t = s$

$\vdash s = s$

BY *unfold cond*

BY *unfold T/F*

BY *reduction*

BY *reduction*

BY *reflexivity*

$\vdash \text{if } F \text{ then } s \text{ else } t = t$

$\vdash F s t = t$

$\vdash (\lambda x. \lambda y. y) s t = t$

$\vdash (\lambda y. y) t = t$

$\vdash t = t$

# “BEWEIS” FÜR $F \neq T$

## Beweis zeigt “nur” die Absurdität von $F = T$

- Es ist nicht möglich,  $F \neq T$  zu beweisen, aber wenn  $F = T$  wäre, dann sind alle  $\lambda$ -Terme gleich

$\vdash F = T \Rightarrow \forall s, t:U. s = t$	BY <code>impliesR THEN allR THEN allR</code>
1. $F = T, s:U, t:U \vdash s = t$	BY <code>transitivity if F then s else t</code>
1.1. $F = T, \dots \vdash s = \text{if } F \text{ then } s \text{ else } t$	BY <code>symmetry</code>
1.1.1. $F = T, \dots \vdash \text{if } F \text{ then } s \text{ else } t = s$	BY <code>subst F = T</code>
1.1.1.1. $F = T, \dots \vdash F = T$	BY <code>axiom</code>
1.1.1.2. $F = T, \dots \vdash \text{if } T \text{ then } s \text{ else } t = s$	BY <code>unfold THEN unfold</code>
1.1.1.2.1. $F = T, \dots \vdash (\lambda x. \lambda y. x) s t = s$	BY <code>reduction THEN reduction</code>
1.1.1.2.1.1... $\vdash s = s$	BY <code>reflexivity</code>
1.2. $F = T, \dots \vdash \text{if } F \text{ then } s \text{ else } t = t$	BY <code>unfold THEN unfold</code>
1.2.1. $F = T, \dots \vdash (\lambda x. \lambda y. y) s t = t$	BY <code>reduction THEN reduction</code>
1.2.1.1. $F = T, \dots \vdash t = t$	BY <code>reflexivity</code>

- Beweis auch ohne Verwendung der `subst` Regel möglich, aber länger

# BILDUNG UND ANALYSE VON PAAREN

$$\langle u, v \rangle \equiv \lambda p. p \ u \ v$$

$$pair.1 \equiv pair \ (\lambda x. \lambda y. x)$$

$$pair.2 \equiv pair \ (\lambda x. \lambda y. y)$$

$$match \ pair \ with \ \langle x, y \rangle \mapsto t \equiv pair \ (\lambda x. \lambda y. t)$$

(uniformer Analyseoperator, oft eleganter in Anwendung)

## Der Analyseoperator ist invers zur Paarbildung

$$\begin{aligned} & match \ \langle u, v \rangle \ with \ \langle x, y \rangle \mapsto t \\ \equiv & \ \langle u, v \rangle \ (\lambda x. \lambda y. t) \\ \equiv & \ (\lambda p. p \ u \ v) \ (\lambda x. \lambda y. t) \\ \longrightarrow & \ (\lambda x. \lambda y. t) \ u \ v \\ \longrightarrow & \ (\lambda y. t[u/x]) \ v \\ \longrightarrow & \ t[u, v/x, y] \quad (kurz \ für \ t[u/x][v/y]) \end{aligned}$$

# INJEKTIONEN UND FALLANALYSE

## Verallgemeinerung boolescher Operatoren

$$\text{inl}(a) \equiv \lambda x. \lambda y. x \ a$$

$$\text{inr}(b) \equiv \lambda x. \lambda y. y \ b$$

$$\text{case } z \text{ of } \text{inl}(x) \rightarrow s \mid \text{inr}(y) \rightarrow t \equiv z (\lambda x. s) (\lambda y. t)$$

## Fallanalyse ist invers zu Injektionen

$$\begin{aligned} & \text{case inl}(a) \text{ of } \text{inl}(x) \rightarrow s \mid \text{inr}(y) \rightarrow t \\ \equiv & (\text{inl}(a)) (\lambda x. s) (\lambda y. t) \\ \equiv & (\lambda x. \lambda y. x \ a) (\lambda x. s) (\lambda y. t) \\ \rightarrow & (\lambda y. (\lambda x. s) \ a) (\lambda y. t) \\ \rightarrow & (\lambda x. s) \ a \\ \rightarrow & s[a/x] \end{aligned}$$

$$\text{Analog: case inr}(b) \text{ of } \text{inl}(x) \rightarrow s \mid \text{inr}(y) \rightarrow t \xrightarrow{*} t[b/x]$$



- **Darstellung natürlicher Zahlen durch iterierte Terme**

- Semantisch: wiederholte Anwendung von Funktionen
- Repräsentiere die **Zahl**  $n$  durch den  $\lambda$ -Term  $\lambda f . \lambda x . \underbrace{f (f \dots (f x) \dots)}_{n\text{-mal}}$
- Notation:  $\bar{n} \equiv \lambda f . \lambda x . f^n x$   
(Markierung  $\bar{n}$  hilft, eine Verwechslung von Zahlen und zugehörigen  $\lambda$ -Termen zu vermeiden)
- Bezeichnung: **Church Numerals**

- **$f: \mathbb{N}^n \rightarrow \mathbb{N}$   $\lambda$ -berechenbar:**

- Es gibt einen  $\lambda$ -Term  $t$  mit der Eigenschaft

$$f(x_1, \dots, x_n) = m \Leftrightarrow t \bar{x}_1 \dots \bar{x}_n = \bar{m}$$

- **Operationen müssen “Termvielfachheit” berechnen**

- Simulation einer Funktion auf Darstellung von Zahlen muß Darstellung des Funktionsergebnisses liefern
- z.B. muß `add`  $\bar{m} \bar{n}$  als Wert immer den Term  $\overline{m+n}$  liefern

# ARITHMETISCHE OPERATIONEN IM $\lambda$ -KALKÜL

● **Nachfolgerfunktion:**  $s \equiv \lambda n. \lambda f. \lambda x. f (n f x)$

– Der Wert von  $s \bar{n}$  ist der Term  $\overline{n+1}$

$$\begin{aligned} s \bar{n} &\equiv (\lambda n. \lambda f. \lambda x. f (n f x)) (\lambda f. \lambda x. f^n x) \\ &\longrightarrow \lambda f. \lambda x. f ((\lambda f. \lambda x. f^n x) f x) \\ &\longrightarrow \lambda f. \lambda x. f ((\lambda x. f^n x) x) \\ &\longrightarrow \lambda f. \lambda x. f (f^n x) \\ &\longrightarrow \lambda f. \lambda x. f^{n+1} x \equiv \overline{n+1} \end{aligned}$$

● **Addition:**  $add \equiv \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x)$

● **Multiplikation:**  $mul \equiv \lambda m. \lambda n. \lambda f. \lambda x. m (n f) x$

● **Test auf Null:**  $zero \equiv \lambda n. n (\lambda n. F) T$

● **Vorgängerfunktion:**

$$p \equiv \lambda n. (n (\lambda f \_x. \langle s, \text{match } f \_x \text{ with } \langle f, x \rangle \mapsto f \ x \rangle) (\lambda z. \bar{0}, \bar{0})).2$$

● **Einfache Rekursion:**  $PRs[b, h] \equiv \lambda n. n h b$

– Für  $f \equiv PRs[b, h]$  gilt:  $f \bar{0} = b$  und  $f (s n) = h (f n)$

# KORREKTHEIT DER ADDITIONSFUNKTION

**Beweise:**  $\text{add } \overline{m} \ \overline{n} = \overline{m+n}$

$\vdash \text{add } \overline{m} \ \overline{n} = \overline{m+n}$	BY	unfold
$\vdash (\lambda m. \lambda n. \lambda f. \lambda x. m \ f \ (n \ f \ x)) \ \overline{m} \ \overline{n} = \overline{m+n}$	BY	reduction
$\vdash (\lambda n. \lambda f. \lambda x. \overline{m} \ f \ (n \ f \ x)) \ \overline{n} = \overline{m+n}$	BY	reduction
$\vdash \lambda f. \lambda x. \overline{m} \ f \ (\overline{n} \ f \ x) = \overline{m+n}$	BY	unfold
$\vdash \lambda f. \lambda x. (\lambda f. \lambda x. f^m \ x) \ f \ (\overline{n} \ f \ x) = \overline{m+n}$	BY	reduction
$\vdash \lambda f. \lambda x. (\lambda x. f^m \ x) \ (\overline{n} \ f \ x) = \overline{m+n}$	BY	reduction
$\vdash \lambda f. \lambda x. f^m \ (\overline{n} \ f \ x) = \overline{m+n}$	BY	unfold
$\vdash \lambda f. \lambda x. f^m \ ((\lambda f. \lambda x. f^n \ x) \ f \ x) = \overline{m+n}$	BY	reduction
$\vdash \lambda f. \lambda x. f^m \ ((\lambda x. f^n \ x) \ x) = \overline{m+n}$	BY	reduction
$\vdash \lambda f. \lambda x. f^m \ (f^n \ x) = \overline{m+n}$	BY	reduction
$\vdash \lambda f. \lambda x. f^{m+n} \ x = \overline{m+n}$	BY	unfold
$\vdash \lambda f. \lambda x. f^{m+n} \ x = \lambda f. \lambda x. f^{m+n} \ x$	BY	reflexivity

Keine Angabe der Unterzieladresse im Beweisbaum, da Beweis linear ist

Beweis ist “generisch”. Konkreter Beweis nur mit konkreter Instanz für  $m$  und  $n$

# DARSTELLUNG VON LISTEN

- **Erweiterung des Konzepts natürlicher Zahlen**

- Wiederholte Anwendung einer Funktion auf Listenelemente
- Repräsentiere  $a_1..a_n$  durch den Term  $\lambda f . \lambda x . f a_1 (f .. (f a_n x) ..)$

- **Grundkonzepte: Leere Liste, Anhängen, Rekursion**

$$[] \equiv \lambda f . \lambda x . x$$

$$t :: list \equiv \lambda f . \lambda x . f t (list f x)$$

$$list\_ind [b, h] \equiv \lambda list . list h b$$

- **Rekursionsgleichungen für  $f \equiv list\_ind [b, h]$**

$\vdash f [] = b$	BY <code>unfold</code>
$\vdash (\lambda list . list h b) [] = b$	BY <code>reduce</code>
$\vdash [] h b = b$	BY <code>unfold</code>
$\vdash (\lambda f . \lambda x . x) h b = b$	BY <code>reduce THEN reduce THEN reflexivity</code>

$\vdash f (t :: list) = h t (f list)$	BY <code>unfold THEN reduce</code>
$\vdash t :: list h b = h t (f list)$	BY <code>unfold</code>
$\vdash (\lambda f . \lambda x . f t (list f x)) h b = h t (f list)$	BY <code>reduce THEN reduce</code>
$\vdash h t (list h b) = h t (f list)$	BY <code>symmetry THEN unfold</code>
$\vdash h t ((\lambda list . list h b) list) = h t (list h b)$	BY <code>reduce THEN reflexivity</code>

# PROGRAMMIERUNG REKURSIVER FUNKTIONEN

- **Funktion, die durch Gleichung  $f(x) = t[f, x]$  definiert ist**
  - d.h. im Programmkörper  $t$  darf  $f$  sich selbst und  $x$  aufrufen
- **Darstellung: Anwendung eines Fixpunktkombinators auf  $t$** 
  - **Fixpunktkombinator**:  $\lambda$ -Term  $R$  mit Eigenschaft  $R t = t (R t)$  für alle  $t$
  - Rekursive Funktion  $f$  ist implementierbar durch  $f \equiv R (\lambda f. \lambda x. t)$ ,  
denn es gilt  $f x = (R (\lambda f. \lambda x. t)) x = (\lambda f. \lambda x. t) f x \xrightarrow{*} t[f, x]$
  - Programmiernotation für  $R(\lambda f. \lambda x. t)$  ist “function  $f(x) = t$ ”  
Notation “function  $f(x, y) = t$ ” steht für  $R(\lambda f. \lambda x. \lambda y. t)$
- **Bekanntester Fixpunktkombinator ist der Y-Kombinator**
  - $Y \equiv \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$

$\vdash Y t = t (Y t)$	BY unfold THEN reduction
$\vdash (\lambda x. t (x x)) (\lambda x. t (x x)) = t (Y t)$	BY reduction
$\vdash t (\lambda x. t (x x)) (\lambda x. t (x x)) = t (Y t)$	BY symmetry
$\vdash t (Y t) = t (\lambda x. t (x x)) (\lambda x. t (x x))$	BY applyEq
1. $\vdash t = t$	BY reflexivity
2. $\vdash Y t = (\lambda x. t (x x)) (\lambda x. t (x x))$	BY unfold THEN reduction
2.1. $\vdash (\lambda x. t (x x)) (\lambda x. t (x x)) = (\lambda x. t (x x)) (\lambda x. t (x x))$	BY reflexivity

# BEISPIELE REKURSIV PROGRAMMIERTER FUNKTIONEN

- **Fakultätsfunktion:** *Es ist  $0! = 1$  und  $n! = n * (n-1)!$  für  $n > 0$* 
  - **fak**  $\equiv$  function fak(n) = if zero(n) then  $\bar{1}$  else mul n (fak(p n))  
 $\equiv Y(\lambda \text{fak} . \lambda n . \text{if zero}(n) \text{ then } \bar{1} \text{ else mul } n \text{ (fak}(p \ n))$ )
- **Subtraktion:**  *$n - m = n$ , falls  $m = 0$ , sonst  $(n - (m - 1)) - 1$* 
  - **sub**  $\equiv$  function sub(n,m) = if zero(m) then n else p(sub n (p m))
- **Test  $n < m$ :** *Es ist  $n < m$  genau dann, wenn  $n + 1 - m = 0$* 
  - **less**  $\equiv \lambda n . \lambda m . \text{zero}(\text{sub } (s \ n) \ m)$
- **Division:** *Suche das erste  $z$  mit  $n < (z + 1) * m$ . Starte Suche bei 0*
  - **div**  $\equiv \lambda n . \lambda m . (\text{function search}(z) =$   
if less n (mul z m) then p z else search(s z))  $\bar{0}$
- **Ackermannfunktion:**
  - **A**  $\equiv$  function A(n,x) =  
if zero(x) then  $\bar{1}$   
else if zero(n) then if zero(p x) then  $\bar{2}$  else add x  $\bar{2}$   
else A (p n) (A n (p x))

## Alle $\mu$ -rekursiven Funktionen sind $\lambda$ -berechenbar

- **Nachfolgerfunktion  $s$ :**  $s \equiv \lambda n. \lambda f. \lambda x. f (n f x)$
- **Projektionsfunktionen  $pr_m^n$ :**  $pr_m^n \equiv \lambda x_1. \dots \lambda x_n. x_m$
- **Konstantenfunktion  $c_m^n$ :**  $c_m^n \equiv \lambda x_1. \dots \lambda x_n. \bar{m}$
- **Komposition  $f \circ (g_1, \dots, g_n)$ :**  
 $\circ \equiv \lambda f. \lambda g_1. \dots \lambda g_n. \lambda x. f (g_1 x) \dots (g_n x)$
- **Primitive Rekursion  $Pr[f, g]$ :**  
 $PR \equiv \lambda f. \lambda g. \text{function } h(x) = \lambda y. \text{if zero } y \text{ then } f x \text{ else } g x (p y) (h x (p y))$
- **Minimierung  $\mu[f]$ :**  
 $Mu \equiv \lambda f. \lambda x. (\text{function } \min(y) = \text{if zero}(f x y) \text{ then } y \text{ else } \min(sy)) \bar{0}$

## Ergebnisse der Berechenbarkeitstheorie übertragen sich

- **Keine nichttriviale extensionale Eigenschaft von  $\lambda$ -Termen kann automatisch geprüft werden** (Satz von Rice)
  - Terminiert die Berechnung einer Funktion  $f$  bei Eingabe  $x$ ?
  - Ist die durch einen  $\lambda$ -Term  $f$  beschriebene Funktion **total**
  - Gehört ein Wert  $y$  zum **Wertebereich** einer Funktion  $f$ ?
  - Berechnet die Funktion  $f$  bei Eingabe von  $x$  den **Wert  $y$**
  - Verhalten sich zwei  $\lambda$ -berechenbare **Funktionen gleich?**
    - Man kann nicht einmal für alle gleichen  $\lambda$ -Terme einen Gleichheitsbeweis finden
    - Das Hauptproblem ist dabei die Gleichheit nichtterminierender  $\lambda$ -Terme
- **Beweis kann direkt im  $\lambda$ -Kalkül geführt werden**
  - Umweg über Berechenbarkeitstheorie mit Church'scher These, Gödelnummern, utm- oder smn-Theorem ist nicht erforderlich



# DAS HALTEPROBLEM

## Terminierung von $\lambda$ -Termen ist unentscheidbar

- **Beweis stützt sich auf wenige Erkenntnisse**
  - $\lambda x.x$  terminiert mit Normalform  $\lambda x.x$
  - $\perp = Y(\lambda x.x)$  terminiert nicht, da  $Y(\lambda x.x) = (\lambda x.x x) (\lambda x.x x)$
  - Die Boole'schen Wahrheitswerte sind verschieden:  $\top \neq \text{F}$

## Terminierung von $\lambda$ -Termen ist unentscheidbar

- **Beweis stützt sich auf wenige Erkenntnisse**

- $\lambda x.x$  terminiert mit Normalform  $\lambda x.x$
- $\perp = Y(\lambda x.x)$  terminiert nicht, da  $Y(\lambda x.x) = (\lambda x.x x) (\lambda x.x x)$
- Die Boole'schen Wahrheitswerte sind verschieden:  $\top \neq \text{F}$

- **Einfacher Widerspruchsbeweis**

- Wir nehmen an, Terminierung von  $\lambda$ -Termen sei entscheidbar

d.h. es gibt einen  $\lambda$ -Term  $h$  mit  $h(t) = \begin{cases} \top & \text{falls } t \text{ terminiert} \\ \text{F} & \text{sonst} \end{cases}$

- Definiere  $d = \lambda x. \text{if } h(x) \text{ then } \perp \text{ else } \lambda x.x.$

- Dann  $h(Yd) = h(d(Yd)) = h(\text{if } h(Yd) \text{ then } \perp \text{ else } \lambda x.x)$

also  $h(Yd) = \begin{cases} h(\perp) = \text{F}, & \text{falls } h(Yd) = \top \\ h(\lambda x.x) = \top, & \text{falls } h(Yd) = \text{F} \end{cases}$

- Dies ist ein Widerspruch, also ist das Halteproblem nicht entscheidbar

- **Vielseitige, ausdrucksstarke Sprache**
  - Klares Berechenbarkeitsmodell
  - Wenige Vorgaben für formales Schließen
  - Mengentheoretische Semantik extrem kompliziert (**domain theory**)
- **Unkontrolliert: Kein Schließen über Datentypen möglich**
  - Wann gehört ein Objekt zu einem bestimmten Datentyp?
  - Welche Struktur hat der Datentyp eines Objekts?
  - Welche **Eigenschaften** hat ein Programm?
- **Erweiterung von Semantik / Inferenzsystem nötig**
  - Beschränke Freiheit der  $\lambda$ -Terme durch Präzisierung ihrer Eigenschaften
  - Wähle Datentypen als Beschreibung von Programmeigenschaften



**Typentheorie**