

# Inhaltsverzeichnis

<b>1</b>	<b>Terme</b>	<b>3</b>
1.1	substitution . . . . .	3
1.2	Kontext . . . . .	3
<b>2</b>	<b>Operationelle Semantik von TES</b>	<b>3</b>
<b>3</b>	<b>Terminierung</b>	<b>4</b>
<b>4</b>	<b>Reduktionsordnungen</b>	<b>5</b>
4.1	Polynomordnungen . . . . .	6
<b>5</b>	<b>Konfluenz</b>	<b>8</b>
<b>6</b>	<b>Der <math>\lambda</math>-Kalkül</b>	<b>16</b>
<b>7</b>	<b>Der ungetypte <math>\lambda</math>-Kalkül</b>	<b>16</b>
7.1	Rekursion . . . . .	19
7.2	Auswertungsstrategie . . . . .	19
<b>8</b>	<b>Der einfach getypte <math>\lambda</math>-Kalkül</b>	<b>20</b>
8.1	Typinferenz . . . . .	23
8.2	Subjektreduktion . . . . .	25
<b>9</b>	<b>Church Rosser des <math>\lambda</math>-Kalkül</b>	<b>26</b>
<b>10</b>	<b>Curry-Howard Isomorphismus</b>	<b>29</b>
<b>11</b>	<b>Induktive Datentypen</b>	<b>31</b>
11.1	Mengenkonstruktionen . . . . .	33
11.2	Initialität und Rekursion . . . . .	36

11.3 Mehrsortige Typen . . . . .	37
11.4 Strukturelle Induktion über Datentypen . . . . .	39
11.5 strukturelle Induktion über mehrsortige Datentypen . . . . .	39
11.6 Kodaten . . . . .	41

# Vorlesung 2

Alexander Mattick Kennung: qi69dube

## Kapitel 1

4. Juli 2020

### 1 Terme

$\Sigma$ -Terme  $t ::= x \mid f(t_1, \dots, t_n) \ (x \in V, f/n \in \Sigma)$

$V$  Menge von Variablen.

$T_\Sigma(V)$  = Menge der  $\Sigma$ -Terme über  $V$  (ist nicht fix, kann sich u.u. z.B. verkleinern)

$FV(t)$  = Menge der in  $t$  (frei) vorkommenden Variablen.

$FV(x) = \{x\}$

$FV(f(t_1, \dots, t_n)) = \bigcup_{i=1}^n FV(t_i)$

#### 1.1 substitution

Substitution ist eine Abbildung  $\sigma : V_0 \rightarrow T_\Sigma(V)$  für ein  $V_0 \subseteq V$ ,  $V_0$  endlich.

$$[t_1/x_1, \dots, t_n/x_n] \ V_0 = \{x_0, \dots, x_1\}, \sigma(x_i) = t_i \ t\sigma = \begin{cases} x\sigma = \sigma(x) \\ f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma) \end{cases}$$

#### 1.2 Kontext

$C(\cdot) = (\cdot) \mid f(t_1, \dots, c(\cdot), \dots, t_n), \ (f/n \in \Sigma)$

$f(t_1, \dots, C(\cdot), \dots, t_n)(g) = f(t_1, \dots, C(g), \dots, t_n)$

## 2 Operationelle Semantik von TES

$\rightarrow_0 \subseteq T_\Sigma(V) \times T_\Sigma(V)$

$R \subseteq T_\Sigma(V) \times T_\Sigma(V)$  heißt

- abgeschlossen bezüglich  $C(\cdot)$ , wenn  $\forall t, s (tRs \implies C(t)RC(s))$ 
  - Bsp.:  $x + y = y + x \implies z * (x + y) = z(y + x)$  zeigt Kontextabschluss  $C(\cdot) = z * (\cdot)$
- Kontextabgeschlossen  $\iff R$  abgeschlossen für alle  $C(\cdot)$
- stabil  $\iff \forall t, s, \sigma (tRs \implies (t\sigma)R(s\sigma))$ 
  - z.B.  $x + y = y + x \implies z^2 + xw = xw + z^2$

Einschrittreduktion  $\rightarrow \subseteq T_\Sigma(V) \times T_\Sigma(V)$  = kontextabgeschlossener und stabiler Abschluss von  $\rightarrow_0$

$\rightarrow = \{(C(s\sigma), C(t\sigma)) \mid t \rightarrow_0 s, C(\cdot) \text{ Kontext}, \sigma \text{ Substitution}\}$

Bew.: Wenn man einen Kontext von einem Kontext macht, erhält man einen Kontext (weil es nur eine Freistelle gibt).

Wenn man substituiert dann ist die substitution entweder in s/t oder im Kontext, substitution im Kontext ändert nur in einen neuen kontext (es bleibt aber kontext).

**Reduktion**  $\rightarrow^*$  (sprich "t reduziert zu s")

**Konvertierbarkeit**  $\leftrightarrow^* = (\rightarrow \cup \rightarrow^-)^*$  ist die Äquivalenz zu  $\rightarrow$ .

**t normal**  $\iff \neg \exists s (t \rightarrow s) \iff t \nrightarrow s$  Normalform von t  $\iff t \rightarrow^* s$  s normal

**Lemma 2.1.** Sei  $R \subseteq T_\Sigma(v) \times T_\Sigma(V)$

1)  $R$  kontextabg.  $\iff R$  abgeschlossen bzgl. aller  $f(t_1, \dots, t_{i-1}, (\cdot), t_{i+1}, \dots, t_n)$  (Induktion über kontexte!):

$(\cdot)$  ist trivial.

$tRs \implies C(t)RC(s) \implies f(t_1, \dots, C(t), \dots, t_n)Rf(t_1, \dots, C(s), \dots, t_n)$

2)  $R$  stabil  $\implies (R \text{ kontextabg.} \iff R \text{ abgeschlossen bezgl aller } f(x_1, \dots, (\cdot), \dots, x_n))$  (folgt direkt aus 1.)

**Beispiel 2.1.**

$\Sigma = \{+, /, 2, s/1, 0/0\}$

1)  $s(x) + y \rightarrow_0 s(x + y)$

2)  $0 + y \rightarrow_0 y$

3)  $(x + y) + z \rightarrow_0 x + (y + z)$

Es gibt versch umklammerungsmöglichkeiten:

$$\begin{aligned} (s(x) + s(y)) + z &\xrightarrow{1, C(\cdot) + z, \sigma = [s(y)/y]} s(x + s(y)) + z \xrightarrow{1, C(\cdot), \sigma = [(x + s(y))/x]} s((x + s(y)) + z) \xrightarrow{3)} s(x + (s(y) + z)) \xrightarrow{1)} s(x + s(y + z)) \\ (s(x) + s(y)) + z &\xrightarrow{3), C(\cdot), \sigma = [s(x)/x, s(y)/y]} s(x) + (s(y) + z) \xrightarrow{1)} s(x) + s(y + z) \xrightarrow{1)} s(x + s(y + z)) \end{aligned}$$

Unterschied Gleichungstheorie und TES: Gleichungstheorie ist eine **Umkehrbare** relation zwischen Termen!

### 3 Terminierung

**Definition 3.1.**

$R \subseteq X \times X$  wohlfundiert  $\iff$  es existiert keine unendliche folge  $x_0, \dots, x_n$  mit  $x_0 R x_1 R \dots$

$(\mathbb{Z}, >)$  ist nicht wohlfundiert.  $(0 > -1 > -2 > \dots)$

$(\mathbb{Q}_+, >)$  ist nicht wohlfundiert  $1 > \frac{1}{2} > \frac{1}{4} > \frac{1}{8} > \dots$

$(\mathbb{N}, >)$  ist wohlfundiert (es endet spätestens bei 0, induktion über Kettenanfänge)

**Beweis 3.1.** i.V. Die Kette  $n_1 > n_2 > \dots$  ist endlich.

Annahme: es gibt eine unendliche Kette bei  $n_0 > n_1 > \dots \implies n_1 > n_2 > \dots$  wäre auch unendlich ( $\infty - 1 = \infty$ ). Widerspruch zur Induktionsvoraussetzung!

### Definition 3.2.

- schwach normalisierend  $\iff$  t hat eine NE  $t \rightarrow \dots \rightarrow s$  normal.
- stark normalisierend  $\iff$  es gibt keine unendliche reduktionsfolge  $\neg \exists t = t_0 \rightarrow t_1 \rightarrow \dots$  (unendlich). (es gibt keine zyklen)

TES  $(\Sigma, \rightarrow_0)$  schwach/stark normalisierend (WN(SN))  $\iff$  alle t in  $(\Sigma, \rightarrow_0)$  schwach/stark normalisierend.

### Beispiel 3.1.

$f(x) \rightarrow_0 f(x)$

$g(x) \rightarrow 1$

$g(x)$  stark normalisierend einzige Reduktion  $g(x) \rightarrow 1 \dashv$

$f(x)$  nicht schwach normalisierend einzige Reduktion  $f(x) \rightarrow f(x) \rightarrow \dots$

$g(f(x))$  schwach normalisierend:  $g(f(x)) \rightarrow 1 \dashv$  (Haskell ausführung)

oder  $g(f(x)) \rightarrow g(f(x)) \rightarrow \dots$  (deshalb nicht stark normalisierend, hier ML-ausführung)

## 4 Reduktionsordnungen

$\leq$  vs  $<$ : reflexiv vs. irreflexiv:  $\forall x (\neg xRx)$ .

R ist strikte Ordnung  $\iff$  R transitiv und Irreflexiv (z.B.  $>$ ).

### Definition 4.1.

$R \subseteq T_\Sigma(V) \times T_\Sigma(V)$ .

**Reduktionsordnung**  $\iff$  R wohlfundierte, stabile, kontextabgeschlossen, strikte Ordnung.

(aus wohlfundiert folgt strikt, sonst könnte man eine unendliche Folge  $xRxRxRx\dots$ ).

**Satz 1.** Sei  $>$  Reduktionsordnung und  $\forall t, s (t \rightarrow_0 s \implies t > s) \implies \rightarrow$  SN (also in jeder Ersetzungsregel wird nach anwendung der Term kleiner).

**Beweis 4.1.**  $>$  ist stabil und kontextabgeschlossen, und  $\rightarrow_0 \subseteq > \implies \rightarrow \subseteq > \implies \rightarrow$  ist wohlfundiert, d.h.  $\rightarrow$  ist SN.

(Weil  $\rightarrow$  der kontextabg. und stabile abschluss von  $\rightarrow_0$  ist, wenn  $>$  wohlfundiert ist, dann kann es auch keine unendlichen Mengen in der Teilmenge  $\rightarrow$  geben)

### Beispiel 4.1.

$|t|$  = Größe von t.  $t > s \iff |t| > |s|$  (also die länge).

kontextabgeschlossen:  $|t| > |s| \implies |C(t)| > |C(s)|$  (freiplatz kommt einmal vor.)

stabil? nicht immer  $|x + 2y - x| > |y + y|$  aber:  $\sigma = [100x/y] \mid x + 2 * 100x - x \not> [100x + 100x]!$

aber Ok, wenn in  $t \rightarrow_0 s$  stets jede variable s höchstens so oft wie in t vorkommt.

$\emptyset$  ist eine Reduktionsordnung.

$\rightarrow \text{SN} \implies \rightarrow^+ \text{Reduktionsordnung}$ .

## 4.1 Polynomordnungen

Recall: Polynome die Menge der Polynome über  $\mathbb{N}$  d.h. mit natürlich zahligen Koeffizienten (insbesondere also keine z.B.  $-1x^2$ ).

$$\mathbb{N}[x_1, \dots, x_n] = \left\{ \sum_{i_1, \dots, i_n \in \mathbb{N}} a_{i_1, \dots, i_n} x_1^{i_1} \dots x_n^{i_n} \mid a_{i_1, \dots, i_n} \in \mathbb{N} \text{ fast immer } a_{i_1, \dots, i_n} = 0 \right\}$$

z.B.  $x^2y + 2y^2zx \in \mathbb{N}[x, y, z]$  ein Summand wird "Monom" genannt. z.B. ist  $y^2zx$  ein Monom und gehört zu  $a_{121} = 2$   
jedes  $p \in \mathbb{N}[x_1, \dots, x_n]$  definiert eine Funktion

$$\mathbb{N}^n \rightarrow \mathbb{N} \quad (k_1, \dots, k_n) \rightarrow p(k_1, \dots, k_n) \in \mathbb{N}$$

$p, q$  Polynom  $\implies p + q, p \times q$  ist Polynom (nach Zusammenfassen gleichartiger Monome)

$\implies$  für  $p \in \mathbb{N}[x_1, \dots, x_n], q_1, \dots, q_n \in \mathbb{N}[y_1, \dots, y_k]$

$\implies p(q_1, \dots, q_n) \in \mathbb{N}[y_1, \dots, y_k]$  (die eingesetzten Polynome können von jeder Art "k" sein, das "buffert" auch unten evtl. vorliegende  $(x^2)^3 = x^6$  mit  $a_{xyz} = 0$ )

$k_1, \dots, k_n \in \mathbb{N} \implies p(k_1, \dots, k_n) \in \mathbb{N}$

**Definition 4.2.** Sei  $\emptyset \neq A \subseteq \mathbb{N}$ .

$p >_A q \iff \forall k_1, \dots, k_n \in A (p(k_1, \dots, k_n) > q(k_1, \dots, k_n))$

**Beispiel 4.2.**

$x^2 >_{\mathbb{N}} x$  gilt nicht  $1^2 \not> 1$  aber schon für  $A = \{n \in \mathbb{N} \mid n \geq 2\}$

**Lemma 4.1.**  $>_A$  ist wohlfundiert.

**Beweis 4.2.** Annahme:  $p_0 >_A p_1 >_A \dots$  (unendlich) wähle  $a \in A$ ; dann  $p_0(a, \dots, a) > p_1(a, \dots, a) > \dots$  in  $\mathbb{N}$  WIDERSPRUCH ( $>_{\mathbb{N}}$  ist wohlfundiert)

**Definition 4.3.**  $p \in \mathbb{N}[x_1, \dots, x_n]$  streng monoton:

$$\forall j \exists i_1, \dots, i_n (i_j > 0 \wedge a_{i_1 \dots i_n} > 0)$$

(also wenn  $x_i$  im Polynom Struktur ist, muss es auch einen Koeffizienten geben, der  $\neq 0$  ist)

**Lemma 4.2.**

$$p \text{ streng monoton} \iff \forall k_1, \dots, k_n, k'_1, \dots, k'_n ((k_1, \dots, k_n) > (k'_1, \dots, k'_n) \implies p(k_1, \dots, k_n) > p(k'_1, \dots, k'_n))$$

$\iff 1) \forall j (k_j \geq k'_j) \text{ und } 2) \exists j (k_j > k'_j) \text{ (mindestens eins echt größer).}$

**Beweis 4.3.** “ $\implies$ ”

$a_{i_1, \dots, i_n} k_1^{i_1} \dots k_n^{i_n} \geq a_{i_1, \dots, i_n} k_1^{i'_1} \dots k_n^{i'_n}$  stets, einmal “ $>$ ”  $\square$

**Definition 4.4.** (monotone) Polynomielle Interpretation  $\mathcal{A}$  besteht aus

- zu jedem  $f/n \in \Sigma$  ein streng monotonen  $p_f \in \mathbb{N}[x_1, \dots, x_n]$
- $A \subseteq \mathbb{N}$  die unter  $p_f$  abgeschlossen ist

so dass  $k_1, \dots, k_n \in A \implies p_f(k_1, \dots, k_n) \in A$

(Eine Polynomordnung besteht aus einem Polynom für jedes Signatursymbol und einer Auswahl natürlicher Zahlen)

$\rightarrow$  Polynomordnung  $>_{\mathcal{A}} \quad t >_{\mathcal{A}} s \iff p_t >_{\mathcal{A}} p_s$

mit  $p_x = x \quad p_{f(t_1, \dots, t_n)} = p_f(p_{t_1}, \dots, p_{t_n})$

**Satz 2.**  $>_A$  ist eine Reduktionsordnung!

**Korollar 4.1.** Wenn  $t \rightarrow_0 s \implies t >_{\mathcal{A}} s$ , dann  $\rightarrow SN$ .

**Beispiel 4.3.**

$f(f(g(x))) \rightarrow_0 f(g(g(x)))$

$p_f(x) = x^2 + 1, p_g(x) = x$  also  $f(f(g(x))) \equiv (x^2 + 1)^2 + 1 >_{\mathbb{N}} f(g(g(x))) = x^2 + 1$

oder einfach  $p_f(x) = x^2, p_g(x) = x$  also, dann muss man jedoch  $A = \mathbb{N} \setminus \{1, 0\}$

**Lemma 4.3.** (Substitutionslemma):

$\sigma = [t_1/x_1, \dots, t_n/x_n], p \in \mathbb{N}[x_1, \dots, x_n] \implies p_t \sigma = p_t(p_{t_1}, \dots, p_{t_n})$

**Beweis 4.4.** Induktion über  $t$ .

-  $p_{x_i} \sigma = p_{t_i} = p_{x_i}(p_{t_1}, \dots, p_{t_n})$

$p_{f(s_1, \dots, s_k)} \sigma = p_f(p_{s_1} \sigma, \dots, p_{s_k} \sigma)$

$\stackrel{IV}{=} p_f(p_{s_1}(p_{t_1}, \dots, p_{t_n}), \dots)$

$\stackrel{\text{substitution}}{=} p_f(p_{s_1}, \dots, p_{s_k})(p_{t_1}, \dots, p_{t_n}) = p_{f(s_1, \dots, s_k)}(p_{t_1}, \dots, p_{t_n})$

**Beweis 4.5.** ( $>_{\mathcal{A}}$  ist Reduktionsordnung)

- strikte Ordnung per definition
- wohlfundiert (es gibt keine endlos absteigende Polynomfolge)
- $>_{\mathcal{A}}$  stabil: Sei  $t >_{\mathcal{A}} s, \sigma = [t_1/x_1, \dots]$

zZ.:  $t\sigma \succ_{\mathcal{A}} s\sigma$ : Seien  $k_1, \dots, k_n \in A$

$$p_{t\sigma}(k_1, \dots, k_n) \stackrel{\text{Lemma}}{=} p_t(p_{t_1}(k_1, \dots, k_n), \dots) > p_s(p_{t_1}(k_1, \dots, k_n), \dots) \stackrel{\text{Lemma}}{=} P_{s\sigma}(k_1, \dots, k_n)$$

- $\succ_{\mathcal{A}}$  kontextabgeschlossen: Sei  $t \succ_{\mathcal{A}} s$ ,  $C(\cdot) = f(x_1, \cdot, (\cdot)_i, \dots, x_n)$  (weil stabilität schon gezeigt, reicht das)

zZ.:  $C(t) \succ_{\mathcal{A}} C(s)$  Seien  $k_1, \dots, k_n \in A$

$$p_f(k_1, \dots, p_t(k_1, \dots, k_n), \dots, k_n) \stackrel{\text{streng monoton}}{<} p_f(k_1, \dots, p_s(k_1, \dots, k_n), \dots, k_n)$$

Es ist beweisbar unentscheidbar, ob es für eine gegebene reduktionsordnung eine polynomordnung die deren Terminierung beweist, gibt. (halteproblem)

#### Beispiel 4.4.

$$(x \oplus y) \oplus z \rightarrow_0 x \oplus (y \oplus z)$$

$$x \oplus (y \oplus z) \rightarrow_0 y \oplus y$$

Gesucht ist also eine poly interpretation von “ $\oplus$ ”:

Hier: Linke seite muss mehr gewichtet werden als die Rechte.

$$p_{\oplus}(x, y) = x^2 + y$$

führt zu:

$$(x^2 + y)^2 + z \succ_{\mathcal{A}} x^2 + (y^2 + z) = x^4 + 2x^2y + y^2 + z$$

$$\mathcal{A} = [1, \infty)$$

$$x^2 + y^2 + z \not\succ_{\mathcal{A}} y^2 + y$$

Geht also nicht, wenn man  $x^2 \rightarrow \infty$

Besser:

$$p_{\oplus}(x, y) = x^2 + xy$$

$$(x^2 + xy)^2 + (x^2 + xy)z = x^4 + 2x^3y + x^2y^2 + x^2z + xyz \succ_{\mathcal{A}} x^2 + x(y^2 + yz) = x^2 + xy^2 + xyz$$

$$\mathcal{A} = \mathbb{N}_{\geq 1}$$

$$x^2 + xy^2 + xyz \succ_{\mathcal{A}} y^2 + yy = 2y^2$$

$$\mathcal{A} = \mathbb{N}_{\geq 2}$$

(Wichtig, man darf keine variablen “verlieren” wenn man noch umformungsschritte hat!)

## 5 Konfluenz

#### Beispiel 5.1. Gruppen

$$x \cdot (y \cdot z) \stackrel{\neg 0}{=} (x \cdot y) \cdot z$$

$$x \cdot e \stackrel{\neg 0}{=} x$$

$$x \cdot x^{-1} \stackrel{\neg 0}{=} e$$

$$y \cdot (x \cdot x^{-1}) \rightarrow y \cdot e \rightarrow y \rightarrow \text{ist eine NF}$$

oder

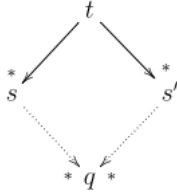
$$y \cdot (x \cdot x^{-1}) \rightarrow (y \cdot x) \cdot x^{-1} \rightarrow \text{ist eine NF}$$

(Knuth-bendix algorithmus[1] würde zur konfluenz führen: regel von einer der beiden NF zur anderen)

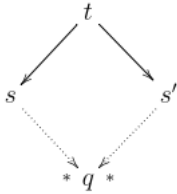


**Definition 5.1.**

- $t, s$  **zusammenführbar** (zf)  $\iff \exists u (t \rightarrow^* u \leftarrow^* s)$  (u.U auch mit null schritten)
- TES T heißt **konfluent** (CR, church/Rosser)  $\iff \forall t, s, s' (t \rightarrow^* s \wedge t \rightarrow^* s' \implies s, s' \text{ zf})$



- T heißt **lokal konfluent** (WCR, weakly Church/Rosser)  $\iff \forall t, s, s' (t \rightarrow s \wedge t \rightarrow s' \implies s, s' \text{ zf})$  (also in nur einem schritt zusammenführbar)



z.B.: ist oben 5.1 weder stark noch schwach CR

**Satz 3.** Sei  $t$  konfluent  $\implies$

- 1)  $s \leftrightarrow^* t \iff s, t \text{ zf}$
- 2)  $s, s' \text{ NF von } t \implies s = s'$

**Beweis 5.1.**

1)  $\Leftarrow$  klar  $\implies$

Haben  $s = t_0 \leftrightarrow t_1 \leftrightarrow \dots \leftrightarrow t_n = t$

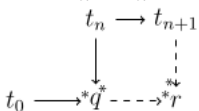
Induktion über  $n$ :

$n = 0$ :  $s = t$  klar

$n \rightarrow n + 1$ : Nach I.V.  $s = t_0 \rightarrow^* q^* \leftarrow t_n \leftrightarrow t_{n+1}$

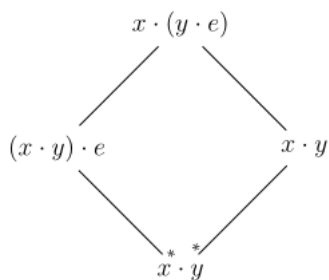
Fall 1:  $t_n \leftarrow t_{n+1}$  fertig (weil  $t_n$  über  $q$  mit  $s$  zusammenführbar)

Fall 2:  $t_n \rightarrow t_{n+1}$  dann gibt es ein  $r$ , dass über  $\rightarrow^*$  mit  $t_{n+1}$  und  $q$  erreichbar ist (wegen konfluenz)



2)  $s \leftrightarrow^* s' \implies s, s' \text{ zf}$ :  $s \rightarrow^* u^* \leftarrow s'$  weil  $s, s' \text{ NF}$ , braucht man genau 0 schritte:  $s = u = s'$

hier ein bsp für eine konfluente form:



**Satz 4.** (Newman's Lemma)

$$SN \wedge WCR \implies CR$$

(also lokale konfluenz und stark normalisierend, führt zur vollen konfluenz, starke konfluenz ist i.a. unentscheidbar (und so auch SN, deshalb widerspricht dieser Satz dem nicht...)) Beweis, später

**Beispiel 5.2.**

Regeln

$$l_1 \rightarrow r_1$$

$$l_2 \rightarrow r_2$$

Terme

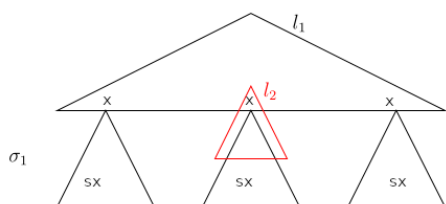
$$C_1(l_1\sigma_1) = t = C_2(l_2\sigma_2)$$

$$C_1(l_1\sigma_1) \rightarrow C_1(r_1\sigma_1)$$

$$C_2(l_2\sigma_2) \rightarrow C_2(r_2\sigma_2) \quad \omega \quad .$$

$$\begin{array}{ccccc}
 l_1 & C_1(l_1\sigma_1) & = t = & C_2(l_2\sigma_2) & l_2 \\
 \downarrow & \downarrow & & \downarrow & \downarrow \\
 r_1 & C_1(r_1\sigma_1) & & C_2(r_2\sigma_2) & r_2
 \end{array}$$

$C_1$  kann ignoriert werden wegen kontextabgeschlossen.



Weil  $l_2$  in  $l_1$  hineinragt, ist nach anwendung von  $l_2 \rightarrow r_2$  kein  $l_1$  mehr für die zweite Regel vorhanden (die eine anwendung zerschneidet die prämissen einer zweiten)

**Definition 5.2.** Unifikation

$t, s$  Terme  $t \doteq s$

$\sigma$  Unifikator von  $t, s$  ( $\sigma \in \text{Unif}(t, s)$ )  $\iff t\sigma = s\sigma$  (syntaktisch)

$t, s$  unfiz  $\iff \text{unif}(t, s) \neq \emptyset$

$\sigma$  allgemeiner als  $\sigma'$   $\iff \exists \tau (\sigma' = \sigma\tau)$

$\sigma$  allgemeinsten Unifikator (mgu) von  $t, s$   $\sigma = \text{mgu}(t, s)$   $\iff \sigma \in \text{Unif}(t, s) \wedge \forall \sigma' \in \text{Unif}(t, s) (\sigma \text{ allgemeiner als } \sigma')$

mgu existiert, wenn  $t, s$  unifizierbar, eindeutig bis auf isomorphismus (injektive umbenennung)

**Beispiel 5.3.** unifikation

$k(r(x), x) \doteq k(z, r(z))$

decomp  $r(x) \doteq z, x \doteq r(z)$

elim  $r(r(z)) \doteq z, x \doteq r(z)$

occurs.

$f(h(x), z), f(y, g(x))$

$\sigma = [h(x)/y, g(x)/z]$

**Definition 5.3.** kritisches Paar nach Knuth-Bendix

Seien  $l_1 \rightarrow_0 r_1, l_2 \rightarrow_0 r_2$ ,

$l_1 = C(t)$   $t$  nichttrivial (d.h.  $t$  keine Variable, konstanten gehen aber...)

und  $FV(l_2) \cap FV(l_1) = \emptyset$

$\sigma = \text{mgu}(t, l_2)$

also, wenn man  $r_1\sigma \leftarrow l_1\sigma = C(t)\sigma = (C\sigma)(t\sigma) = C\sigma(l_2) \rightarrow C\sigma(r_2\sigma) = C(r_2)\sigma$

$\implies (r_1\sigma, C(r_2)\sigma)$  **kritisches Paar**

**Lemma 5.1.**  $(r_1\sigma, C(r_2)\sigma)$  kritisches Paar

$\implies r_1\sigma \leftarrow l_1\sigma = C(l_2)\sigma \rightarrow C(r_2)\sigma$  (kritische Paare sind divergente Redukte eines gemeinsamen Ursprungs)

**Korollar 5.1.**  $T \text{ WCR} \implies$  alle Paare sind zf**Satz 5.** alle kritischen Paare zf  $\implies$  WCR (Critical Pair Lemma)

Aufwand ist  $O(n^3)$  (paare und dann jede Regel für Kontext  $C(t)$  einsetzen, mal die Anzahl der Schritte, die jede Reduktion selbst benötigt)

**Beispiel 5.4.** (Gruppe)

$(l_1 \rightarrow_0 r_1) = (x \cdot (y \cdot z)) \rightarrow_0 (x \cdot y) \cdot z$

(in frische Variablen umbenennen)

$(l_2 \rightarrow_0 r_2) = (x' \cdot e \rightarrow x')$

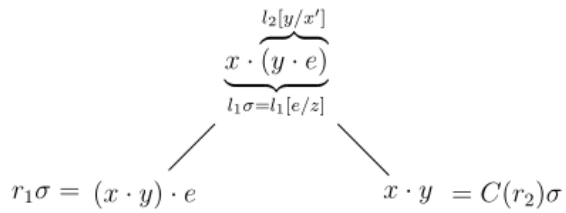
Jetzt: wähle einen Teilterm aus, und mach das "t" draus:

$$C(\cdot) = x \cdot (\cdot)$$

$$t = y \cdot z \sigma = mgu(t, l_2) = [y/x', e/z]$$

$$\rightarrow \text{kritisches Paar } (r_1 \sigma, C(r_2) \sigma) = ((x \cdot y) \cdot e, x \cdot y)$$

(die  $r_1, r_2$  sind oben definiert...)



### Beispiel 5.5.

$$l_1 \rightarrow_0 r_1 = (x \cdot (y \cdot z)) \rightarrow_0 (x \cdot y) \cdot z$$

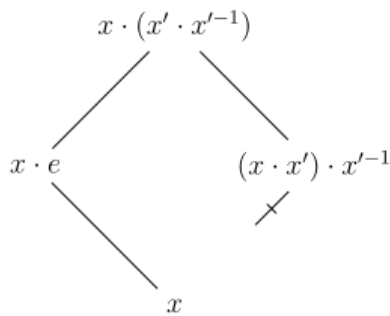
$$l_2 \rightarrow_0 r_2 = (x' \cdot x'^{-1} \rightarrow_0 e)$$

$$C(\cdot) = x \cdot (\cdot)$$

$$t = y \cdot z$$

$$\sigma = mgu(t, l_2) = [x'/y, x'^{-1}/z]$$

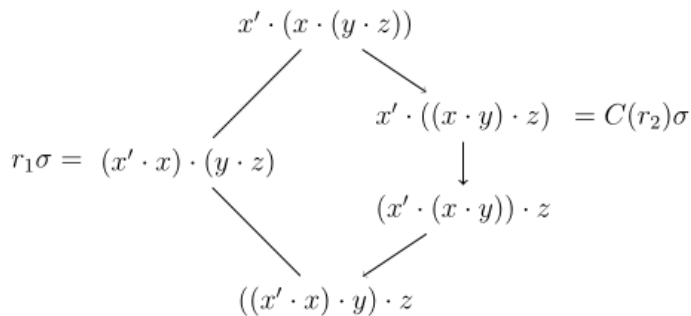
$$(x \cdot x') \cdot x'^{-1} \leftarrow x \cdot (x' \cdot x'^{-1}) \rightarrow x \cdot e \text{ nicht z.f.}$$



### Beispiel 5.6. $l_1 \rightarrow_0 r_1 = (x \cdot (y \cdot z)) \rightarrow_0 (x \cdot y) \cdot z = l_2 \rightarrow_0 r_2$

$$t = y \cdot z \sigma = mgu(t, l_2) = mgu((y \cdot z), x' \cdot (y' \cdot z')) = [y/x', y'/z'/z]$$

$$(x \cdot y) \cdot (y' \cdot z') \leftarrow x \cdot (y \cdot (y' \cdot z')) \rightarrow x \cdot ((y \cdot y') \cdot z')$$



sind z.f.

### ACHTUNG

wenn man das umbenennen der variablen vergisst, dann krigt man  $y \cdot z$  und  $x \cdot (y \cdot z) \rightarrow \perp$  occurs!!

**Beweis 5.2.** Critical Pair Lemma5

Notation  $C(\cdot) \sqsubseteq D(\cdot) \iff \exists E(\cdot)(C(\cdot) = D(E(\cdot)))$  (also C liegt unter D, wenn man in D einen weiteren kontext einführen kann, um ihn zu C zu verwandeln! Wie bei mgu auch)

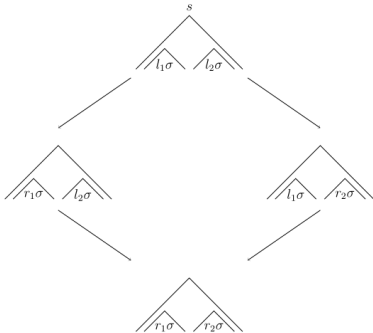


$C(\cdot) \perp D(\cdot) \iff C(\cdot) \not\sqsubseteq D(\cdot) \wedge D(\cdot) \not\sqsubseteq C(\cdot)$  (also keiner ist subset des anderen, sie sind orthogonal)



Sei  $l_1 \rightarrow_0 r_1, l_2 \rightarrow r_2$  anwendbar auf s

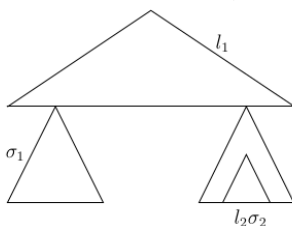
**Fall 1:**  $C_1(\cdot) \perp C_2(\cdot)$



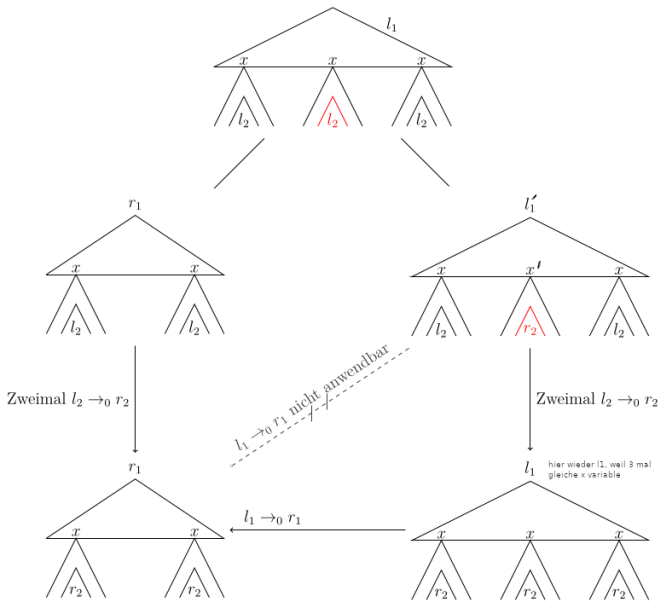
Beide Terme stören sich nicht, ich kann immer beide Regeln in beliebiger Reihenfolge anwenden

Fall 2: o.b.d.  $A C_2(\cdot) \sqsubseteq C_1(\cdot)$  mit  $C_1(\cdot) = (\cdot)$  (man kann sich den äußersten einfach wegdenken, der Teilbaum unter einem echten  $C_1 \neq (\cdot)$  ist equivalent zu einem normalen Baum mit wurzel direkt unter  $C_1$ )

ohne einschränkung  $l_2\sigma = l_2$ , weil  $l_2$  sowieso vollkommen unter unserer substitution liegt, also auch im nachhinein gemacht werden kann ( es stört den Rest des Terms nicht).

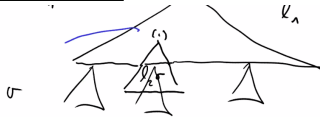


unterfall 2a)  $C_2$  ist echt unterhalb von  $l_1$



auf der rechten seite ist  $l_1 \rightarrow r_1$  nicht mehr anwendbar, weil  $l_1 \rightarrow r_1$  fordert, dass es 3 gleiche argumente gibt.

unterfall 2b)  $C_2$  ist nicht unterhalb von  $l_1$ , d.h.  $(\cdot)$  von  $C_2$  liegt in  $l_1$ :



Dies ist gleich der situation des Kritischen paares 5.1: Der einzige ort, wo es schiefgehen kann ist also, wenn das kritische paar nicht zf ist.

Es reicht also: Für  $\sigma \in Unif(t, l_2)$  ist  $(r_1\sigma, C_2(r_2)\sigma)$  zf.

Gilt nach Annahmen für  $\sigma = mgu(t, l_2)$  (alle kritischen paare sind zf), dann  $\sigma' = \sigma\tau$  für ein  $\tau$

$r_1\sigma\tau, C_2(r_2)\sigma\tau$  unsere Reduktionsrelation ist stabil, also ist  $r_1\sigma, C_2(r_2)\sigma$  zf, so auch alle substitutionen.

### Satz 6. *wohlfundiert Induktion*

$$R \subseteq X \times X \text{ wohlfundiert} \implies$$

Wenn  $\forall x(\forall y(xRy \Rightarrow P(y))) \Rightarrow P(x)$  (1)

(wenn für alle nachfolger von  $x$   $P(y)$  gilt, dann gilt auch  $P(x)$ )

dann gilt  $\forall x(P(x))$  (2)

*dies heißt wohlfundierte Induktion.*

**Beweis 5.3.** Kontraposition:

zeige  $(1) \wedge \neg(2) \Rightarrow R$  nicht wohlfundiert.

Per  $\neg(2)$  ex.  $x_0$  mit  $\neg P(x_0)$

 $\Rightarrow$ 
$$\Rightarrow$$
  
(1) ex.  $x_1$  mit  $x_0 R x_1 \neg P(x_1) \dots$ 

d.h.  $x_0 R x_1 R x_2 \dots$ , R nicht wf.

**(dependent choice, viel harmloser als ZFC's auswahlaxiom)**

**Beispiel 5.7.** 1)  $X = \mathbb{N} \ R = \{(n+1, n) | n \in \mathbb{N}\}$

wohlfundierte Relation (bzw vollständige Relation als wohlfundierte...):

$P(0)$

$\forall n(P(n) \implies P(n+1))$

zusammen liefert das  $\forall n(P(n))$

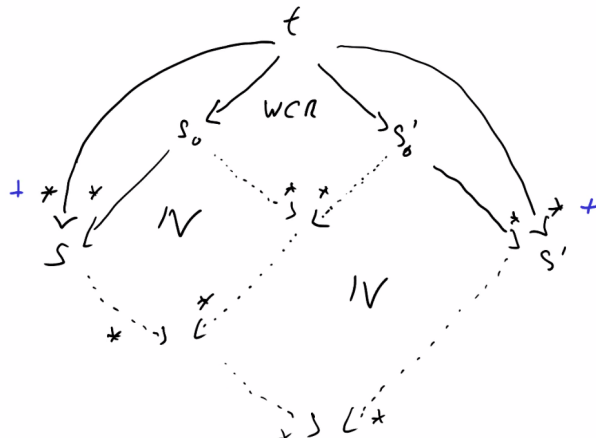
**Beispiel 5.8.**  $X = \mathbb{N} \ R \Rightarrow$ : Course-of-values-Induktion (man nimmt also für alle echt kleineren  $n$  die Aussage an)

**Beweis 5.4.** Newman's Lemma 4  $SN \& WCR \implies CR$

beweis per wohlfundierter Relation über  $\rightarrow$  (ist wf wegen SN).

o.E.  $t \rightarrow^+ s, s'$  (weil in 0 schritten reduzieren trivialerweise sofortig zf ist)

Idee: man teilt die schritte zwischen  $s$  und  $s'$  in jeweils zwei paare von beiden seiten  $(s_0, s'_0)$  dann Induktionsvoraussetzung anwenden, woraus man folgern kann, dass dies für jeden schritt möglich ist:



## 6 Der $\lambda$ -Kalkül

Beispiel haskell

```
twice f x = f $ f x
-- Eigentlich ``Schönfinkelisierung`` nach dem echten Erfinder.
map :: (a -> b) -> (List a -> List b)
map f [] = []
map f (x:xs) = (f x) : map f xs
```

ungetypter  $\lambda$ -Kalkül = LISP.

getypter  $\lambda$ -Kalküle

Church (Ein zuerst unvollständiges system), Kleene, Rosser (haben beide R, Curry)

## 7 Der ungetypte $\lambda$ -Kalkül

**Definition 7.1.**  $\lambda x.t$  "die Funktion, die x auf t abbildet  $x \mapsto t$  (wobei üblicherweise  $x \in FV(T)$  ist)"

$ts$  Anwendung von s auf t. (Applikation)

Terme t,s gegeben durch:

$$t, s ::= x \mid ts \mid \lambda x. t \quad (x \in V)$$

Wobei das Zweite als  $(\lambda)$ -Notation.

**Beispiel 7.1.**

- " $\lambda x.3 + x$ " (3 und plus ist technisch gesehen nicht definiert...)
- $\lambda x.xx$  (Haskell würde typfehler liefern, wenn man x als funktion auf sich selbst andwendet, es gibt aber sinnvolle kontexte für solche dinge: Wenn x eine berechenbare funktion ist, dann muss es einen bestimmten, Gödelnummerierbaren, funktionsraum geben. Das erste x wäre dann die interpretation als funktion, und die zweite die Gödelnummer)
- $\lambda x.\lambda y.x =: f$ , dann wäre z.B.  $fxy = (\lambda y.x)y = x$

**Definition 7.2.** Freie Variablen und konventionen

Kontexte

$$C(\cdot) = (\cdot) \mid tC(\cdot) \mid C(\cdot)s \mid \lambda x.C(\cdot)$$



**Kongruenz** = Kontextabgeschlossene Äquivalenz.

Notation  $\lambda x_1 \dots \lambda x_n. t = \lambda x_1 \dots x_n. t$

$tsu = (ts)u$

Scope von  $\lambda$  so weit wie möglich.

$\lambda x. xx = \lambda x. (xx)$  im gegensatz zu  $(\lambda x. x)x$

Freie Variablen:

$FV(x) = \{x\}$

$FV(ts) = FV(t) \cup FV(s)$

$FV(\lambda x. t) = FV(t) \setminus \{x\}$

**Definition 7.3.** Substitution

- $x\sigma = \sigma(x)$
- $(ts)\sigma = (t\sigma)(s\sigma)$
- $(\lambda x. t)\sigma = \lambda y. (t\sigma')$  wobei y eine **frische variable** ist  
(also  $y \in FV(\sigma(z)), (z \in FV(t) \setminus \{x\} \iff z \in FV(\lambda x. t))$ ), sonst könnte x "gefangen werden"  $\lambda x. y[x/y] \neq \lambda x. x!!$   
Lösung, wie bei  $\forall/\exists$  in GLOIN. (capture-avoiding substitution, liefert hier  $\lambda x. y[x/y] = \lambda t. x$ , mit  $\sigma' = \sigma[x \rightarrow t]$ )  
de-Broujin indizes.  $(\lambda x. \lambda y. xy = \lambda \lambda. 2 \ 1)$  oder nominale Mengen[3]

**Definition 7.4.**  $t =_\alpha s$  (sprich " $\alpha$ -äquivalent")

$\iff$  t geht aus s durch **Umbenennung** gebundener Variablen hervor (ohne Variableneinfang!).

Formal:  $=_\alpha$  ist die von

$$\lambda x. t =_\alpha \lambda y. t[y/x] \ (y \notin FV(t) \setminus \{x\})$$

erzeugte Kongruenz

**Beispiel 7.2.**

$$\lambda x. xy =_\alpha \lambda z. zy \not\supseteq_\alpha \lambda y. yy$$

**Lemma 7.1.**  $=_\alpha$  ist stabil

**Beweis 7.1.** Es reicht: erzeugende Relation ist stabil:

$$R = \{(\lambda x. t, \lambda y. t[y/x]) \mid y \notin FV(\lambda x. t)\}$$

Sei also  $y \notin FV(\lambda x. t)$

$$zZ: (\lambda x. t)\sigma R (\lambda y. t[y/x])\sigma$$

Daraus folgt (2)  $\lambda x'. t\sigma' \lambda y'. t[y/x]\sigma''$

Wobei  $\sigma' = \sigma[x \mapsto x']$  und  $\sigma'' = \sigma[y \mapsto y']$  und  $x', y'$  frisch

$$[y/x]\sigma'' = \sigma'[y'/x']$$

$$x \rightarrow y \rightarrow y' = x \rightarrow x' \rightarrow y'$$

somit ist die Rechte Seite  $\lambda y'. t\sigma[y'/x']$  mit  $y' \notin FV(\lambda x'. t\sigma')$  frisch.

Die Rechte Seite ist also gleich der linken in (2)

**Satz 7.**  *$\beta$ -Reduktion. Operationale Semantik ("Wie sich ein program während der Ausführung verändert")*

*Im imperativen gibt es Kontexte:  $\eta, (x := 1; c) \rightarrow \eta[x \mapsto 1]; c$  ( $\eta$  Umgebung, wie in GLOIN)*

*In  $\lambda$ -Kalkül gibt es sowas nicht:  $\beta$ -Reduktion als kontextabgeschlossene Umformung*

$$(\lambda x. 3 + x) 3 \rightarrow 3 + 4 \quad (\rightarrow \text{ wenn + bekannt ist})$$

*$\lambda$ -Kalkül ist im wesentlichen ein TES (nicht 100% wegen alpha-equiv und gebundenen Variablen)*

$$(\beta) (\lambda x. t) x \rightarrow_0 t$$

$\Rightarrow$  *Einschrittreduktion  $\rightarrow$*

$$C((\lambda x. t) s) \rightarrow C(t[s/x])$$

*$(\lambda x. t) s$  heißt  $\beta$ -Redex. [hier nicht: ( $\eta$ )  $\lambda x. yx \rightarrow_0 y$ , beliebt in theoriebetrachtung, aber nicht in programmiersprachen (wenn man Seiteneffekte/IO hat, macht ( $\eta$ ) viel kaputt, weil damit die "ausführung" von  $x$  auf  $y$  umgangen wird)]*

### Beispiel 7.3.

- $(\lambda x. xx)(yx) \rightarrow_\beta yx(yx)$
- $(\lambda xy. x(yx)) zu \rightarrow_\beta \lambda y. z(yz)u \rightarrow_\beta z(uz)$
- $\omega := \lambda x. xx, \omega\omega = (\lambda x. xx)\omega \rightarrow_\beta \omega\omega \rightarrow_\beta \dots$  Nicht terminierend.
- Booleans " $x \times x \rightarrow x$ "

$$true := \lambda xy. x \quad false := \lambda xy. y$$

- Paare: " $Paar \equiv Fkt$ "  $Bool \rightarrow x$

- $fst := \lambda p. p true$
- $snd := \lambda p. p false$
- $pair := \lambda xy. \lambda z. zxy$  wobei "z eine von true/false ist"

Dies liefert uns:

$$\begin{aligned} \underline{fst (pair x y)} &\rightarrow_\beta fst(\lambda xy. \lambda zxy) xy \rightarrow_{2 \times \beta} fst(\lambda z. zxy) = (\lambda p. p true) \lambda z. zxy \rightarrow_\beta (\lambda z. zxy) true \rightarrow_\beta true xy = \\ &(\lambda xy. x) xy \rightarrow_\beta (\lambda y. x) y \rightarrow_\beta x \end{aligned}$$

## 7.1 Rekursion

$fact = \lambda n. \text{if } n=0 \text{ then } 1 \text{ else } n * fact(n-1)$

Dieser Aufruf besteht aus einer primitiven rekursionsfunktion F und der funktion selbst.  $fact = F fact$

$F = \lambda f. \lambda n. \text{if } n=0 \text{ then } 1 \text{ else } n * f(n-1)$  F nennt man auch ein Funktional.

$fact = F fact$  nennt man Fixpunktgleichung. (rekursive Funktionen sind Fixpunktgleichungen)

Fixpunktkombinator fix:

$\text{fix } F = F(\text{fix } F)$

### Satz 8. $\lambda$ -Kalkül Fixpunktkombinator

1) Jedes  $t$  hat einen Fixpunkt  $s$ , d.h.  $s \rightarrow_{\beta} ts$  (also die Reduktion liefert wieder  $ts$  auf dem wider reduziert werden kann, ad absurdum)

2) Es existiert ein Fixpunktkombinator  $Y$ , d.h.  $Y t \rightarrow_{\beta} s$   $s$  ist Fixpunkt von  $t$

$Y t \rightarrow_{\beta} s \xrightarrow{(1)}_{\beta} ts$

### Beweis 7.2.

1)  $s = W_t W_t$ ,  $W_t = \lambda x. t(xx)$  (wie oben bei  $\omega\omega$ , bloß mit  $t$  ausenrum):

$s = W_t W_t = (\lambda x. t(xx)) W_t \rightarrow_{\beta} t(W_t W_t) = ts$

2)  $Y = \lambda f. W_f W_f$  wenn man das jetzt auf ein  $f$  anwendet erhält man genau  $fs = s$

**Beispiel 7.4.** Der Fall von oben  $\lambda x. xx = \omega$  und dann  $\omega\omega$  hat die funktion  $t = \omega$  terminiert deshalb nicht.

$\lambda x. ((\lambda y. y)(xx))$  jetzt  $t = \lambda y. y$  (also rekursion über die Identitätsfunktion) und  $s$  wäre dann  $\omega\omega \rightarrow_{\beta} t(\omega\omega) \rightarrow_{\beta} \omega\omega$

## 7.2 Auswertungsstrategie

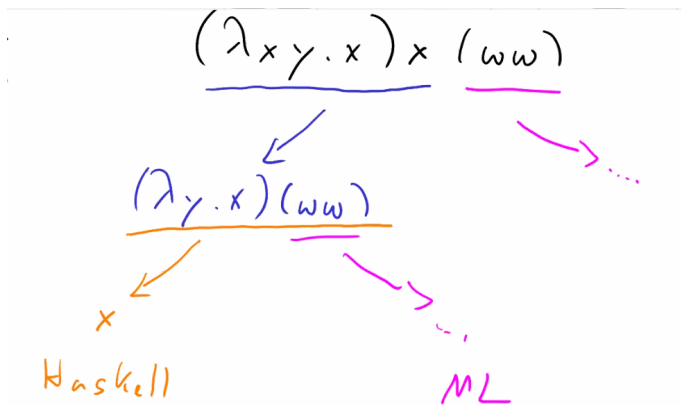
### Beispiel 7.5.

$(\lambda x y. x)x(\omega\omega)$  Wenn man probiert zuerst  $\omega\omega$  zu reduzieren, läuft man unendlich weiter.

Wenn man den rechten reduziert erhält man:

$(\lambda y. x)(\omega\omega)$  wo man entweder wieder ad absurdum ( $\omega\omega$ ) reduzieren kann (ML, leftmost-innermost, applikativ), oder das ganze zerlegen in:

$\lambda y. x\omega\omega = x$  (Haskell, leftmost-outermost, normal/standard)



**Definition 7.5.** applikative (leftmost-innermost) Reduktion  $\rightarrow_a$

induktiv definiert durch:

- 1a)  $(\lambda x. t) s \rightarrow_a t[s/x]$  **wenn**  $t, s$  normal (innermost, eager).
- 2a)  $(\lambda x. t \rightarrow_a \lambda x. t')$ , wenn  $t \rightarrow_a t'$  (eine echte prog. sprache macht aber niemals Termreduktionen unter einem lambda)
- 3a)  $ts \rightarrow_a t' s$  wenn  $t \rightarrow_a t'$
- 4a)  $ts \rightarrow_a t s'$ , wenn  $s \rightarrow_a s'$  und  $t$  normal.

**Definition 7.6.** normale (leftmost-outermost) Reduktion  $\rightarrow_n$

- 1n)  $(\lambda x. t) s \rightarrow_n t[s/x]$  immer (outermost reinziehen)
- 2n)  $\lambda x. t \rightarrow_n \lambda x. t'$ , wenn  $t \rightarrow_n t'$
- 3n)  $ts \rightarrow_n t' s$ , wenn  $t \rightarrow_n t'$  und  $t$  keine  $\lambda$ -Abstraktion. (wenn es eine wäre, dann 1. Regel)
- 4n)  $ts \rightarrow_n t s'$  wenn  $s \rightarrow_n s'$  und  $t$  normal und keine  $\lambda$ -Abstraktion.

**Beispiel 7.6.**  $(\lambda y. x)(\omega \omega)$

mit normaler Reduktion:

$$(\lambda y. x)(\omega \omega) \rightarrow_{1n} x$$

mit Applikativer Reduktion:

$$(\lambda y. x)(\omega \omega) \rightarrow_{4.a} (\lambda y. x)(\omega \omega)$$

**Satz 9.** Standardisierungssatz:

$$\text{Sei } t \rightarrow^* s \text{ s Normalform} \implies t \rightarrow_n^* s$$

(Nebenbemerkung:  $(\lambda x. f x x) t \rightarrow_n f t t \rightarrow_n f s t \rightarrow_n f s s$  und  $(\lambda x. f x x) t \rightarrow_a (\lambda x. f x x) s \rightarrow_a f s s$  also geht es mit applikativen schneller, weil man funktionen nur 1 mal evaluieren muss)

## 8 Der einfach getypte $\lambda$ -Kalkül

Typen  $\alpha, \beta$ :

$\alpha \rightarrow \beta$  Funktion von  $\alpha$  nach  $\beta$

Typvariablen  $a, b, \dots$

z.B.  $\lambda x. x : a \rightarrow a$

### Definition 8.1.

Gegebene Menge  $\mathbf{V}$  var.

Typvariablen  $\mathbf{B}$  von Basistypen.

(**bool**, **Int**, ...) sind typen  $\alpha, \beta, \dots$

definiert durch

$$\alpha, \beta ::= a | \mathbf{b} | \alpha \rightarrow \beta \quad (a \in \mathbf{V}, \mathbf{b} \in \mathbf{B})$$

z.B.  $a \rightarrow (b \rightarrow a) = (a \rightarrow b) \rightarrow a = a \rightarrow b \rightarrow a$  Terme: Church:  $\lambda x : \alpha. t$  nur typkorrekte Terme. (also termbildung und typisierung)

Curry:  $\lambda x. t$ , Term kann typisierbar sein oder nicht.

$\omega = \lambda x. x x$  z.B. nicht typisierbar ( $\lambda \rightarrow$ , weil typ voll rekursiv ist)

wir benützen curry.

$x \lambda y. y$  ? weil  $x$  unbekannt/untypisiert.

### Definition 8.2. kontexte

Ein Kontext ist eine endliche Menge  $\Gamma$  von Typisierungsannahmen  $x : \alpha$  ( $x \in V$ ) "x hat typ  $\alpha$ "

Schreibweise:  $\Gamma, x : \alpha = \Gamma \cup \{x : \alpha\}$

d.h.  $\Gamma$  ist eine endliche partielle Abbildung. (von variablennamen auf typen)

Typisierungsurteile (typing judgements)  $\Gamma \vdash t : \alpha$  "in Kontext  $\Gamma$  hat  $t$  Typ  $\alpha$ " z.B.  $f : a \rightarrow b, x : a \vdash f x : b$

Herleitbarkeit induktiv:

$$(Ax) \frac{}{\Gamma \vdash x : \alpha} \quad (x : \alpha) \in \Gamma$$

$$(\rightarrow_e) \frac{\Gamma \vdash t : \alpha \rightarrow \beta \quad \Gamma \vdash s : \alpha}{\Gamma \vdash t s : \beta}$$

$$(\rightarrow_i) \frac{\Gamma[x \mapsto \alpha] \vdash t : \beta}{\Gamma \vdash \lambda x. t : \alpha \rightarrow \beta} \quad (\text{also wenn } x \text{ schon einen typ hat, wird dieser überschrieben, shadowing})$$

Rechts: sonst könnte  $x : \alpha$  zum clash führen! (in der Realität könnte man das lösen, indem man aus  $\Gamma$  eine List statt eine Menge baut)

### Beispiel 8.1.

$$\begin{array}{c} Ax \frac{}{\vdash x : a \rightarrow b, y : a \vdash x : a \rightarrow b, x : a \rightarrow b, y : a \vdash y : a} AX \\ \rightarrow_i \frac{\vdash x : a \rightarrow b, y : a \vdash x y : b}{\vdash x : a \rightarrow b \vdash \lambda y. x y : a \rightarrow b} \\ \rightarrow_i \frac{\vdash x : a \rightarrow b \vdash \lambda y. x y : a \rightarrow b}{\vdash \lambda x y. x y : (a \rightarrow b) \rightarrow (a \rightarrow b)} \end{array}$$

$$\rightarrow_e \frac{x : a \rightarrow \vdash x : a \rightarrow x : a \rightarrow \vdash x : a}{\rightarrow_i \frac{x : \vdash xx}{\vdash \lambda x. xx :}} \text{ CIRCULAR DEPENDENCY}$$

Berechnungsprobleme:

- gilt  $\vdash t : \alpha$  ? (typcheck)
- finde (existiert?)  $\alpha$  mit  $\vdash t : \alpha$  (Typinferenz)
- finde (existiert?)  $t$  mit  $\vdash t : \alpha$  (Type inhabitation)

**Beispiel 8.2.**  $a \rightarrow a$  inhabited (Identitätsfunktion  $\lambda x.x$ , bildet typ auf sich selbst ab, nach curry-howard tautologie)

$a$  nicht inhabited (also für sich stehend hat ein wert nicht irgendeinen typ, nicht obdA gültig)

$(a \rightarrow a) \rightarrow a$  nicht inhabited (das erste ist eine Tautologie, also immer wahr, a selbst ist aber nicht immer wahr)

denn: wäre  $\vdash t : (a \rightarrow a) \rightarrow a$ , dann  $t(\lambda x.x) : a$  widerspruch! (dependent Types a'la idris/agda, Programmsynthese, automatisches Beweisen)

Eigenschaften:

$$\text{c frisch} \frac{\phi(c)}{\forall(\phi)} \forall I$$

$$\frac{\phi \vdash \psi}{\phi \rightarrow \psi} \rightarrow I$$

$$\text{c frisch} \frac{\phi(c) \vdash \psi(c)}{\forall x(\phi \rightarrow \psi)} \text{ herleitbar:}$$

Beweis:

$$\forall I \frac{\frac{\phi \vdash \psi}{\phi(c) \rightarrow \psi(c)}}{\forall x(\phi \rightarrow \psi)} \text{ c frisch}$$

Regel zulässig  $\iff$  durch ihre Hinzunahme wird nichts neu herleitbar.

**Lemma 8.1.** (Weakening)

$$(wk) \frac{\Gamma \vdash t : \alpha}{\Gamma' \vdash t : \alpha} \Gamma \subseteq \Gamma'$$

(also ein größerer Kontext ändert nichts an der Herleitbarkeit)

**Beweis 8.1.** Induktion über Herleitung von  $\Gamma \vdash t : \alpha$

$$(Ax) \Gamma \vdash x : \alpha, x : \alpha \in \Gamma \implies x : \alpha \in \Gamma' \implies \Gamma' \vdash x : \alpha \quad (\rightarrow_i) \frac{\Gamma[x \mapsto \alpha] \vdash t : \beta}{\Gamma \vdash \lambda x. t : \alpha \rightarrow \beta}$$

(\*) Nach IV. (prämissen ist kleineres Gamma als konklusion)  $\Gamma'[x \mapsto \alpha] \vdash t : \beta$

da  $\Gamma[x \mapsto \alpha] \subseteq \Gamma'[x \mapsto \alpha]$

Sei  $y : \beta \in \Gamma[x \mapsto \alpha]$  (also ein beta ist links, so muss es auch rechts sein)

Fall 1:  $y \neq x \implies y : \beta \in \Gamma \implies y : \beta \in \Gamma' \implies y : \beta \in \Gamma'[x \mapsto \alpha]$

Fall 2:  $y = x \implies \beta = \alpha \implies y : \beta = x : \alpha \in \Gamma[x \mapsto \alpha]$

per (\*)  $\rightarrow_i \Gamma' \vdash \lambda x. t : \alpha \rightarrow \beta$  (die prämissen gilt, also kann man auch die gleiche folgerung machen)

**Lemma 8.2. Inversion**

(man kann alle Regeln auch umdrehen)

- 1)  $\Gamma \vdash x : \alpha \implies (x : \alpha) \in \Gamma$  (ax inversion)
- 2)  $\Gamma \vdash ts : \beta \implies$  es existiert  $\alpha$  mit  $\Gamma \vdash t : \alpha \rightarrow \beta$  ( $\rightarrow_e$  inversion, also wenn es eine Anwendung gibt, dann muss es eine Funktion dazu gegeben haben)
- 3)  $\Gamma \vdash \lambda x. t : \gamma \implies \gamma$  hat die Form  $\alpha \rightarrow \beta$  und  $\Gamma[x \mapsto \alpha] \vdash t : \beta$  ( $\rightarrow_i$  inversion, also der type des input einer Funktion muss herleitbar sein)

**Beweis 8.2.** Regeln sind syntaxgerichtet

**8.1 Typinferenz**

$\lambda x. x : a \rightarrow a$

$\lambda x. x : (a \rightarrow b) \rightarrow (a \rightarrow b)$

Offensichtlich ist das erst besser als das zweite. Es muss also eine "Allgemeinheitshierarchie" geben (most general typing)

**Definition 8.3.** Terminologie/Notation:

$TV(\alpha)$  = Menge der in  $\alpha$  vorkommenden Typvariablen.

$$TV(\Gamma) = \bigcup_{(x:\alpha) \in \Gamma} TV(\alpha)$$

Typsubstitution = Substitution von Typen für Typvariablen  $\sigma$  Lösung von  $\Gamma \vdash t : \alpha$ , wenn  $\Gamma \sigma \vdash t : \alpha \sigma$  herleitbar.

allgemeinste Lösung (wie bei mgu  $\sigma' = \sigma \theta$  dann ist  $\sigma$  das allgemeinere, wenn das  $\forall \sigma'$  gilt dann ist  $\sigma$  die allgemeinste Lösung)

Prinzipaltyp von  $\Gamma \vdash t$  = allgemeinste Lösung von  $\Gamma \vdash t : a$  a frisch ( $a \notin TV(\Gamma)$ ) (prinzipaltyp ist allgemeinste Lösung mit frischen typen und eindeutig modulo Umbenennung)

$\Gamma \vdash t$  typisierbar  $\iff \Gamma \vdash t : a$  hat eine Lösung. (a frisch)

**Satz 10.** Algorithmus W nach HINDLEY/MILNER

Berechne zu  $\Gamma \vdash t : \alpha$  ("Ziel")

$PT(\Gamma; t; \alpha)$  Menge von Typpgleichungen  $a \doteq \beta$  mit  $PT(\Gamma; t; \alpha)$  unfizierbar  $\iff \Gamma \vdash t : \alpha$  hat Lösung.

$mgu(PT(\Gamma; t; \alpha))$  liefert allgemeinste Lösung von  $\Gamma \vdash t : \alpha$  (liefert und ist nicht gleich, weil der PT mehr variablen substituiert als notwendig)

$\implies mgu(PT(\Gamma; t; a))(a) =$  Prinzipaltyp von  $t$  (a frisch, t geschlossen)

Implizit geht man bei diesen Regeln immer von  $x \in \Gamma$  aus

- $PT(\Gamma; x; \alpha) = \{\alpha \doteq \beta \mid x : \beta \in \Gamma\}$  (nach Ax inversionslemma)
- $PT(\Gamma; ts; \alpha) = PT(\Gamma; t; a \rightarrow \alpha) \cup PT(\Gamma; s; a)$  **global** (a frisch) nach  $(\rightarrow_e)$
- $PT(\Gamma; \lambda x.t; \alpha) = PT(\Gamma[x \rightarrow a]; t, b) \cup \{a \rightarrow b \doteq \alpha\}$  mit  $a, b$  **global** frisch  $(\rightarrow_i)$  invers. (wir fitten also input auf a und output auf b)

Das global ist notwendig, um einfang bei unifikation zu vermeiden. Lösung z.b. über [3]

### Beispiel 8.3.

$\vdash \lambda xy.xy$

$$PT(\emptyset, \lambda xy.xy; a) = PT(x : b, \lambda y.xy, c) \cup \{a \doteq b \rightarrow c\} =$$

$$PT(x : b, y : d; xy; e) \cup \{a \doteq b \rightarrow c, c \doteq d \rightarrow e\}$$

$$PT(x : b, y : d; x; f \rightarrow e) \cup PT(x : b, y : d; y; y : f) \cup \{a \doteq b \rightarrow c, c \doteq d \rightarrow e\}$$

$$= \{b \doteq f \rightarrow e, y \doteq f, a \doteq b \rightarrow c, c \doteq c = d \rightarrow e\}$$

jetzt hat man gleichungen, die man unifizieren muss:

$mgu = [f/d, f \rightarrow e/c, f \rightarrow e/b, (f \rightarrow e) \rightarrow (f \rightarrow e)/a]$  wir haben oben mit a angefangen, also ist der endtyp  $\lambda xy.xy : (f \rightarrow e) \rightarrow (f \rightarrow e)$  ist Prinzipaltyp.

$$PT(x : a; x \lambda z.z; c) = PT(x : a; x; b \rightarrow c) \cup PT(x : a, \lambda z.z; b) = \{a \doteq b \rightarrow c\} \cup PT(x : a, z : d; z; e) \cup \{b \doteq d \rightarrow e\} = \{a \doteq b \rightarrow c, d \doteq e, b \doteq d \rightarrow e\}$$

$mgu = [e \rightarrow e/b, e \rightarrow e \rightarrow c/a, c/c]$  “ $e \rightarrow e \rightarrow c/a, c/c$ ” also ist Prinzipaltyp.

$$PT(\emptyset; \lambda x.xx, a) = PT(x : b; xx; c) \cup \{a \doteq b \rightarrow c\} = PT(x : b; x; d \rightarrow c) \cup PT(x : b; x; d) \cup \{a \doteq b \rightarrow c\} = \{b \doteq d \rightarrow c, b \doteq d, \dots\}$$

Unifikation liefert occurs  $\perp$  nach substitution [d/b]

**Satz 11.**  $(\Gamma, t)$  typisierbar  $\iff PT(\Gamma; t; a)$  (a frisch) unifizierbar; dann  $mgu(PT(\Gamma; t; \alpha))|_{TV(\Gamma) \cup \{a\}}$  Prinzipaltyp von  $(\Gamma; t)$

**Beweis 8.3.** Zeige allgemeiner:

$$PT(\Gamma; t; \alpha) \text{ unifizierbar} \iff \Gamma \vdash t : \alpha \text{ lösbar}$$

dann

$$mgu(PT(\Gamma; t; \alpha))|_{TV(\Gamma, \alpha)}$$

allgemeinste Lösung von  $\Gamma \vdash t : \alpha$

Zeige dazu:

$$\Gamma \sigma \vdash t : \alpha \sigma \iff \sigma \text{ ist erweiterbar zu } \sigma' \in Unif(PT(\Gamma; t; \alpha))$$

d.h.  $\sigma'|_{TV(\Gamma, \alpha)} = \sigma$  ist also erweiterbar.

per Induktion über t:

“ $\Leftarrow$ ”: per Typregeln(8.2).



z.B.  $t = \lambda x.s$ :

haben  $\sigma' \in \text{Unif}(PT(\Gamma[x \mapsto a]; s; b) \cup \{\alpha \doteq a \rightarrow b\})$

Nach IV.  $\underbrace{\Gamma[x \mapsto a]\sigma'}_{=\Gamma[x \mapsto a]\sigma} \vdash s; \underbrace{b\sigma'}_{b\sigma}$   
 Per  $(\rightarrow_e)$   $\frac{\Gamma[x \mapsto a]\sigma \quad s; b\sigma}{\Gamma\sigma \vdash \lambda x.s: a\sigma' \rightarrow b\sigma'}$

Daraus folgt  $\alpha\sigma' = \alpha\sigma$

“ $\Rightarrow$ ” Per Inversion:

1.  $\Gamma\sigma \vdash x: \alpha\sigma \xRightarrow{\text{inversion}} x: \beta \in \Gamma, \alpha\sigma = \beta\sigma$

$\Rightarrow \sigma \in \text{Unif}(PT(\Gamma; x; \alpha))$   
 $\alpha \doteq \beta$

2.  $\Gamma\sigma \vdash ts: \alpha\sigma \xRightarrow{\text{Inversion}}$

es existiert ein  $\gamma$  mit  $\Gamma\sigma \vdash t: \gamma \rightarrow \alpha\sigma, \Gamma\sigma \vdash s: \gamma$

Setze  $\sigma' = \sigma[a \mapsto \gamma]$  (a global frisch)  $\Rightarrow \Gamma\sigma' \vdash t: (a \rightarrow \alpha)\sigma', \Gamma\sigma' \vdash s: a\sigma'$

$\xRightarrow{IV}$   $\sigma'$  erweitert zu  $\underbrace{\sigma'' \in \text{Unif}(PT(\Gamma; t; a \rightarrow \alpha))}_{\text{gemeinsame TV sind nur die aus } TV(\Gamma; \alpha; a)}$  und  $\sigma'' \in \text{Unif}(PT(\Gamma; s; a))$

$\Rightarrow \sigma'' \in \text{Unif}(PT(\Gamma, ts, \alpha))$

3.  $\Gamma\sigma \vdash \lambda x.s: \alpha\sigma \xRightarrow{\text{Inversion}}$

$\alpha\sigma = \beta \rightarrow \gamma \quad \Gamma\sigma[x \mapsto \beta] \vdash s: \gamma$

Setze  $\sigma' = \sigma[a \mapsto \beta, b \mapsto \gamma]$ , a, b global frisch

$\Rightarrow \Gamma[x \mapsto a]\sigma' \vdash s: b\sigma'$

$\xRightarrow{IV}$   $\sigma'$  erweitert zu  $\sigma'' \in \text{Unif}(PT(\Gamma[x \mapsto a]; s; b))$

$(a \rightarrow b)\sigma'' = \alpha\sigma'' \Rightarrow \sigma'' \in \text{Unif}(PT(\Gamma; \lambda x.s; \alpha))$

(Das letzte folgt daraus, dass  $\alpha\sigma = \beta \rightarrow \gamma$  ist und wir  $\sigma' = \sigma[a \mapsto \beta, b \mapsto \gamma]$  haben)

## 8.2 Subjektreduktion

**Satz 12.**  $\Gamma \vdash t: \alpha, t \rightarrow_\beta s \Rightarrow \Gamma \vdash s: \alpha$

$\triangleleft$  “ $\Leftarrow$ ” gilt **NICHT**: z.B.  $t = (\lambda x.y)(\lambda x.xx) \rightarrow_\beta y$

**Lemma 8.3.** *Substitution*

$\Gamma[x \mapsto \alpha] \vdash t: \beta, \Gamma \vdash s: \alpha \Rightarrow \Gamma \vdash t[s/x]: \beta$

*Beweis: induktion über t.*

**Beweis 8.4.**  $t = C((\lambda x.u)v), s = C(v[u/x])$

Induktion über  $C(\cdot)$

z.B.:  $C(\cdot) = (\cdot)$  Per inversion:

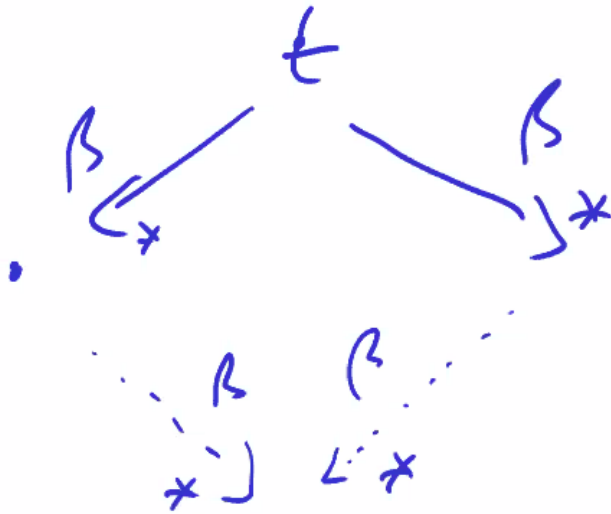
$\Gamma \vdash \lambda x.u: \beta \rightarrow \alpha, \Gamma \vdash v: \beta$

$\Gamma[x \mapsto \beta] \vdash u: \alpha$

$\xRightarrow{\text{subst-lemma}} \Gamma \vdash \underbrace{u[v/x]}_s: \alpha$

## 9 Church Rosser des $\lambda$ -Kalkül

(Dies ist der Ursprüngliche Church-Rosser beweis, der Name wird jetzt mittlerweile für alle TES benutzt).

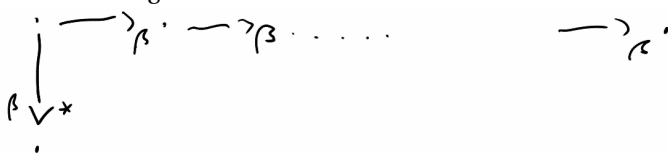


**Lemma 9.1.** *Streifenlemma (strip-lemma)*



*Effektiv ein mittelweg zwischen WCR und CR.*

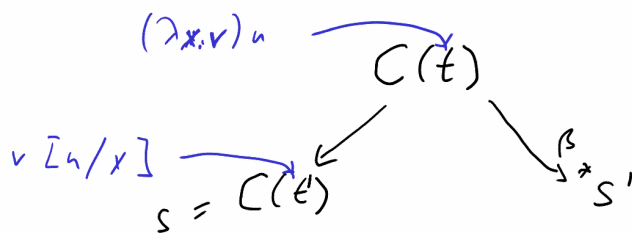
Daraus folgt CR:



oben hat man die n-beta-reduktionen. Unten macht man jeweils immer einen Schritt und kaskadiert nach links.

Dies geht für jedes TES.

Speziell für lambda sähe das so aus.



markierte Terme:  $t, s ::= x | ts | \lambda x. t | (\underline{\lambda} x. t) s$  (also genau gleich, man kann nur **beta-reduzierbare** terme markieren).

Diese unterstriche müssen wieder “unlocked” werden.

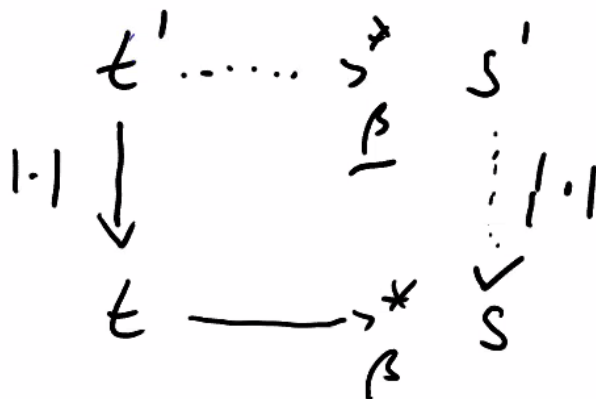
$|t|$ : Entfernt  $\_$  aus  $t$ , z.B.  $|(\underline{\lambda} x. t) s| = (\lambda x. |t|) |s|$

$\phi(t)$  Reduziere alle unterstrichenen Redexe:

- $\phi(x) = x$
- $\phi(ts) = \phi(t)\phi(s)$
- $\phi(\lambda x. t) = \lambda x. \phi(t)$
- $\phi((\underline{\lambda} x. t) s) = \phi(t)[\phi(s)/x]$

Syntactic sugar  $t \xrightarrow{| \cdot |} |t|$

**Lemma 9.2.** *Lemma A*



Dabei  $(\lambda x. t) s \rightarrow_{\beta} t[s/x]$   $(\underline{\lambda} x. t) s \rightarrow_{\beta} t[s/x]$  Es ignoriert also die Unterstriche.

**Beweis 9.1.** Reduziere auf  $t \rightarrow_{\beta} s$ , Induktion über Kontexte

Wir bekommen also nicht mehr dazu, oder verlieren ausdrucks-kraft.

**Lemma 9.3.** *Syntaktisches Substitutionslemma*

$$u[v/y][s/x] = u[s/x][v[s/x]/y]$$

nicht simultan! Das  $v$  wird vom  $x$  beeinflusst.

wenn  $y \notin FV(s)$ ,  $x \neq y$  (bei dem ersten hätte man eine doppelsubstitution, beim zweiten wird die zweite reduktion void)

**Beweis 9.2.** Induktion über u (weil in diesen Reinsubstituiert wird)

Der interessante Fall ist hier der Induktionsanfang, weil im I.S nur weitergereicht wird.

Sei n eine Variable. (LHS/RHS =Left/Right handed side)

$$u \notin \{x, y\} \checkmark$$

$$u = x : LHS = s, RHS = s \text{ weil } y \notin FV(s)$$

$$u = y : LHS = v[s/x], RHS = v[s/x]$$

**Lemma 9.4.** Lemma B

$$a) \phi(t[s/x]) = \phi(t)[\phi(s)/x]$$

b)  $\phi$  bewahrt  $\beta$

$$\begin{array}{ccc} t & \xrightarrow{\beta} & s \\ \downarrow \phi & & \downarrow \phi \\ \phi(t) & \dots\dots\dots & \phi(s) \end{array} \quad \text{„}\phi \text{ bewahrt } \beta\text{“}$$

**Beweis 9.3.** .

a) Induktion über t, interessant nur  $t = (\lambda y. u) v$

$$\begin{aligned} \phi(((\lambda y. u) v)[s/x]) &\stackrel{(1)}{=} \phi((\lambda y. u[s/x]) v[s/x]) = \phi(u[s/x])[\phi(v[s/x])/y] \\ &\stackrel{IV}{=} \phi(u)[\phi(s)/x][\phi(v)[\phi(s)/x]/y] \\ &\stackrel{\text{synt.Subst}+(1)}{=} \underbrace{\phi(u)[\phi(v)/y]}_{\phi((\lambda y. u) v)}[\phi(s)/x] \\ &= \phi((\lambda y. u) v)[\phi(s)/x] \end{aligned}$$

(1) o.E.  $y \neq x$   $y \notin FV(s)$  ( im zweifelsfall umbenennen)

b) Reduziere auf  $t \rightarrow_{\beta} s$ , Induktion über Kontexte:

$(\cdot)$  : Fälle nach Markierung:

Fall 1: markiert:

$$\left[ \begin{array}{ccc} (\lambda x. u) v & \xrightarrow{\beta} & u[v/x] \\ \phi & & \phi \\ \phi(u)[\phi(v)/x] & = & \phi(u)[\phi(v)/x] \end{array} \right]$$

Fall 2: unmarkiert:

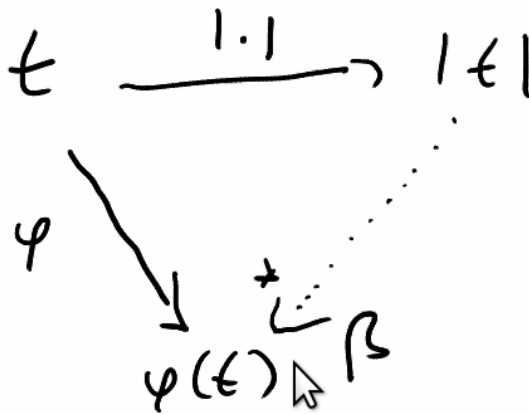
$$\left[ \begin{array}{ccc} (\lambda x. u) v & \xrightarrow{\beta} & u[v/x] \\ \phi & & \phi \\ \lambda x. \phi(u) \phi(v) & \xrightarrow{\beta} & \phi(u)[\phi(v)/x] \end{array} \right]$$

Beispiel mit nichtleerem Kontext:

Kontext  $(\lambda x. C(\cdot))s$  und  $t \rightarrow_{\beta} s$

$$\phi(\lambda x. C(t))s = \phi(C(t))[\phi(s)/x] \xrightarrow[\beta]{IV, C \text{ ist kleinerer Kontext}} \phi(C(s))[\phi(s)/x] \checkmark$$

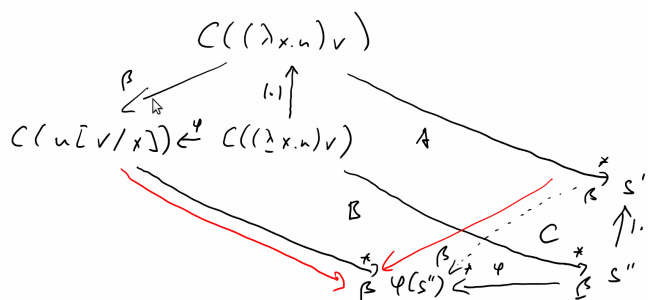
**Lemma 9.5.** Lemma C



**Beweis 9.4.** Induktion über  $t$ , interessanter Fall  $t = (\lambda x. u) v$  (sonst passiert ja nichts):

$$\left[ \begin{array}{ccc} (\lambda x. u) v & \xrightarrow{I.I} & (\lambda x. |u|) |v| \\ \phi & & \phi \\ \phi(u)[\phi(v)/x] & \beta \leftarrow & (\lambda x. \phi(u)) \phi(v) \end{array} \right]$$

$C((\lambda x. u) v)$



## 10 Curry-Howard Isomorphismus

Typen = Propositionen (=Formel) "Types are propositions".

Terme/Programme = Beweise (u.U. ist ein Typ nicht bewohnt, also nicht Beweisbar)

Coq "man hat einen Term definiert, der diesen Typ hat"

minimale Logik:

$$\phi, \psi ::= a | \phi \rightarrow \psi \quad (a \in V)$$

Deduktion:

$$(\rightarrow_E) \frac{\phi \rightarrow \psi \quad \phi}{\psi}$$

$$(\rightarrow_I) \frac{[\phi] \vdash \psi}{\phi \rightarrow \psi}$$

Sequenzenkalkül:  $\frac{\text{Annahme } \Gamma}{\text{Menge Formeln}} \vdash \underbrace{\phi}_{\text{Formel}} \quad \text{"linke Regel" } (\rightarrow_E)$

$$\begin{array}{l}
(\rightarrow_E) \frac{\Gamma \vdash \phi \rightarrow \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi} \\
(\rightarrow_I) \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \rightarrow \psi} \\
(Ax) \frac{}{\Gamma \vdash \phi}
\end{array}$$

**Satz 13.** *C-H-I:*

$$\vdash \phi \iff \phi \text{ bewohnt}$$

**Beweis 10.1.** .

$\Leftarrow$  trivial: lösche Terme aus Herleitung von  $\vdash t : \phi$  ergibt Herleitung  $\vdash \phi$  (damit erhält man die kombination der  $\phi$  als typen, dies sind trivialerweise die Herleitung)

$\Rightarrow$  Zu  $\Gamma$  definiert Kontext  $\bar{\Gamma}$ :

$$\Gamma = \{\phi_1, \dots, \phi_n\} \Rightarrow \bar{\Gamma} = \{x_1 : \phi_1, \dots, x_n : \phi_n\}$$

Also jeder Term wird als Typ gewertet und bekommt eine Variable.

$$\text{Zeige } \Gamma \vdash \psi \Rightarrow \exists t (\bar{\Gamma} \vdash t : \psi)$$

per Induktion über Herleitungen von  $\Gamma \vdash \psi$

$$(Ax): \phi \in \Gamma \Rightarrow \psi = \phi_i \text{ für ein } x_i : \phi_i \in \bar{\Gamma} \xRightarrow{(AX)} \bar{\Gamma} \vdash x_i : \phi_i$$

$(\rightarrow_E)$

$$(\rightarrow_E) \frac{\Gamma \vdash \phi \rightarrow \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}$$

I.V. haben t,s mit

$$\frac{\bar{\Gamma} \vdash t : \phi \rightarrow \psi \quad \bar{\Gamma} \vdash s : \phi}{\bar{\Gamma} \vdash ts : \psi}$$

$(\rightarrow_I)$  Beweis

$$(\rightarrow_I) \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \psi \rightarrow \phi}$$

I.V. haben (bei  $\phi = \phi_{n+1}$ )

$$(\rightarrow_I) \frac{\bar{\Gamma}, x_{i+1} : \phi_{i+1} \vdash t : \psi}{\bar{\Gamma} \vdash \lambda x_i. t : \phi_{i+1} \rightarrow \psi}$$

$\triangleleft$  “ $\rightarrow$ ” ist intuitionistisch!

z.B.  $((a \rightarrow b) \rightarrow a) \rightarrow a$  ist klassisch gültig, aber in minimaler (intuitionistischer) Logik nicht herleitbar.

$$\begin{array}{l}
(Ax) \frac{}{(a \rightarrow a) \rightarrow, a \vdash a} \\
(\rightarrow_E) \frac{(a \rightarrow a) \vdash (a \rightarrow a) \rightarrow a \quad (a \rightarrow a) \rightarrow a \vdash a \rightarrow a}{(a \rightarrow a) \rightarrow a \vdash a} \\
(\rightarrow_I) \frac{(a \rightarrow a) \rightarrow a \vdash a}{\vdash ((a \rightarrow a) \rightarrow a) \rightarrow a}
\end{array}$$

Auf Typeebene:

$$\underbrace{((a \rightarrow a) \rightarrow a) \rightarrow a}_{\lambda f. f(\lambda x. x)}$$

Man hat eine funktion, die wieder eine funktion erhält also das innere  $(a \rightarrow a) \rightarrow a$  und hat als Ergebnis wieder ein a. Dazu muss man anwenden. Die innerer funktion muss den typ  $a \rightarrow a$  haben. Das einzig mögliche hierfür ist die identitätsfunktion  $\lambda x. x$

Deshalb haben wir eine funktion f in den wir eine funktion (identität) haben.

## 11 Induktive Datentypen

```
data Nat where
  0: () -> Nat
  Suc: Nat -> Nat
```

definiert Signatur  $\Sigma_{Nat} = \{0/0, Suc/1\}$

Die Semantik von Nat ist definiert als

$\llbracket Nat \rrbracket = \{0, Suc(0), Suc(Suc(0)), \dots\}$  also ein Herbrandmodell.

**Definition 11.1.** Ein  $\Sigma$ -Modell ( $\Sigma$ -Algebra)  $\mathfrak{M}$  besteht aus

- Menge  $M$  (Träger)
- zu  $f/n \in \Sigma$
- $\mathfrak{M} \llbracket f \rrbracket : M^n \rightarrow M$

Interpretation von Termen unter Umgebung  $\eta : V \rightarrow M$ :

$$\mathfrak{M} \llbracket t \rrbracket \eta \in M$$

$$\mathfrak{M} \llbracket x \rrbracket \eta = \eta(x) \quad x \in V$$

$$\mathfrak{M} \llbracket f(t_1, \dots, t_n) \rrbracket \eta = \mathfrak{M} \llbracket f \rrbracket (\mathfrak{M} \llbracket t_1 \rrbracket \eta, \dots, \mathfrak{M} \llbracket t_n \rrbracket \eta)$$

**Beispiel 11.1.**  $\llbracket Nat \rrbracket$  ist eine  $\Sigma_{Nat}$ -Algebra per:

$$\llbracket Nat \rrbracket \llbracket 0 \rrbracket = 0$$

$$\llbracket Nat \rrbracket \llbracket succ \rrbracket (x) = Suc(x) \in \llbracket Nat \rrbracket$$

**Definition 11.2.** Seien  $\mathfrak{M}, \mathfrak{N}$   $\Sigma$ -Algebren.

$\Sigma$ -Homomorphismus = Abbildung  $h : M \rightarrow N$  (wobei  $M, N$  trägermengen des respektiven Models sind)

$\forall f/n \in \Sigma, x_1, \dots, x_n \in M$

$$h(\mathfrak{M} \llbracket f \rrbracket (x_1, \dots, x_n)) = \mathfrak{N} \llbracket f \rrbracket (h(x_1), \dots, h(x_n))$$

Erinnerung: lineare Abbildung:

$$h(\lambda \cdot x) = \lambda h(x) \wedge h(x + y) = h(x) + h(y)$$

sprich: homomorphismen sind lineare Abbildung zwischen Algebren.

**Lemma 11.1.** •  $id_n : \mathfrak{M} \rightarrow \mathfrak{M}$  ist  $\Sigma$ -Homomorphismus

•  $h : \mathfrak{M} \rightarrow \mathfrak{N}, k : \mathfrak{N} \rightarrow \mathfrak{P}$  sind  $\Sigma$ -Homomorphismus  $\implies k \cdot h : \mathfrak{M} \rightarrow \mathfrak{P}$  ist  $\Sigma$ -Homomorphismus

**Definition 11.3.**  $h$  ist Isomorphismus  $\iff h$   $\Sigma$ -Homomorphismus und bijektiv.

$\iff \exists h^{-1} (h \cdot h^{-1} = id \wedge h^{-1} \cdot h = id)$  (aber i.A. zwei verschiedene identitätsfunktionen!)

Isomorphismen sind also strukturerhaltend!!!

Dies gilt nicht in jedem Kontext: geordnete Systeme lassen das nicht einfach so zu!.

Menge  $A \{b, \gamma\}$  ohne Ordnung und eine geordnete Menge  $B (\{1, 0\}, \leq)$ .

Es gibt ein  $h : A \rightarrow B : x \leq y \implies h(x) \leq h(y)$

Aber das  $h^{-1}$  ist nicht monoton:  $0 \leq 1, h^{-1}(0) \not\leq h^{-1}(1)$

Das heißt die Abbildung die im Algebraischen sinne isomorphismen sind, sind nicht unbedingt isomorphismen im Kontext der geordneten Mengen.

**Lemma 11.2.**  $h$  Isomorphismus  $\implies h^{-1}$  ist Isomorphismus

**Beweis 11.1.**  $zZ: h^{-1}$  ist Homomorphismus (bijektivität ist ja bereits bekannt!), d.h.

$$h^{-1}(\mathfrak{N}[\![f]\!](y_1, \dots, y_n)) = \mathfrak{M}[\![f]\!](h^{-1}(y_1), \dots, h^{-1}(y_n)) \xLeftrightarrow{h \text{ injektiv}} \underbrace{h(h^{-1}(\mathfrak{N}[\![f]\!](y_1, \dots, y_n)))}_{\mathfrak{N}[\![f]\!](y_1, \dots, y_n)} = \underbrace{h(\mathfrak{M}[\![f]\!](h^{-1}(y_1), \dots, h^{-1}(y_n))))}_{=\mathfrak{N}[\![f]\!](h(h^{-1}(y_1)), \dots, h(h^{-1}(y_n)))}$$

Erinnerung  $m \equiv n \pmod{4}$  ist Äquivalenzklasse  $\mathbb{Z}/4\mathbb{Z} = \{[n]_4 | n \in \mathbb{Z}\} = \{[0]_4, [1]_4, [2]_4, [3]_4\}$

Darauf kann addition definiert werden:  $[n]_4 + [m]_4 = [n + m]_4$  ist wohldefiniert.

**Beispiel 11.2.**  $\Sigma_{Nat}$ -Algebra  $\mathfrak{M}$  mit  $M = \mathbb{Z}/4\mathbb{Z}, \mathfrak{M}[\![0]\!] = [0]_4, \mathfrak{M}[\![suc]\!]( [n]_4 ) = [n + 1]_4$

$h : \llbracket Nat \rrbracket \rightarrow \mathfrak{M}$

$Suc^n(0) \mapsto [n]_4$  ist homomorph:

$h(\llbracket Nat \rrbracket[\![0]\!]) = [0]_4 = \mathfrak{M}[\![0]\!]$

$h(\llbracket Nat \rrbracket[\![Suc]\!](Suc^n(0))) = h(Suc^{n+1}(0)) = [n+1]_4 = \mathfrak{M}[\![suc]\!]( [n]_4 ) = \mathfrak{M}[\![Suc]\!](h(Suc^n(0)))$  (sogar surjektiv homomorph, aber natürlich nicht injektiv, somit kein isomorphismus)

Allg.: Datentyp  $D = \text{Signatur } \Sigma, \llbracket D \rrbracket = \llbracket \Sigma \rrbracket = T_\Sigma(\emptyset)$  (also alle geschlossenen Terme) als Algebra, mit  $\llbracket \Sigma \rrbracket[\![f]\!](t_1, \dots, t_n) = f(t_1, \dots, t_n)$

**Definition 11.4.**  $\Sigma$ -Algebra  $\mathfrak{M}$  initial  $\iff \forall \mathfrak{N} \exists! \text{ Homomorphismus } h : \mathfrak{M} \rightarrow \mathfrak{N}$  (initiales Modell ist also allgemeinstes Modell)

**Satz 14.**  $\llbracket \Sigma \rrbracket$  ist initial genannt "Termalgebra"



**Beweis 11.2.** Sei  $\mathfrak{N}$  eine  $\Sigma$ -Sigma Algebra. z.Z:  $\exists! h$  mit

$$(*) \quad h(\llbracket \Sigma \rrbracket f \rrbracket(t_1, \dots, t_n)) = \mathfrak{N} \llbracket f \rrbracket(h(t_1), \dots, h(t_n)) \quad (f/n \in \Sigma)$$

gilt für  $h(t) = \mathfrak{N} \llbracket t \rrbracket$ ,

(\*) ist rekursive Definition von  $\mathfrak{N} \llbracket \dots \rrbracket$

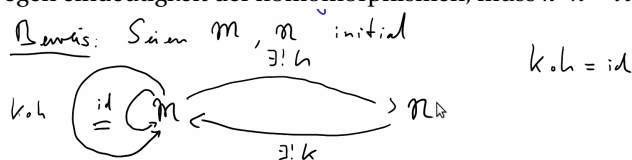
**Satz 15.** Die initiale  $\Sigma$ -Algebra ist eindeutig bis auf **eindeutige** Isomorphie (also kategorietheorie...)

**Beweis 11.3.** Seien  $\mathfrak{M}, \mathfrak{N}$  initial

$$\mathfrak{M} \xleftarrow{\exists! k} \mathfrak{N} \xrightarrow{\exists! h}$$

Zu  $\mathfrak{M}, \mathfrak{N}$  gehört jeweils eine identitätsfunktion.

Wegen Eindeutigkeit der Homomorphismen, muss  $k \cdot h = id$  sein (bzw  $h \cdot k = id$ )



technisch gesehen definiert sich damit eine Äquivalenzklasse über  $\Sigma$ -Algebren.

## 11.1 Mengenkonstruktionen

$X_1, X_2$  Mengen  $X_1 \times X_2 = \{(x_1, x_2) | x_1 \in X_1, x_2 \in X_2\}$

$\pi_i(x_1, x_2) = x_i$  (i-te Projektion,  $i=1,2$ )

$\pi_i: X_1 \times X_2 \rightarrow X_i$

Mehrere Funktionen auf gleiche Daten:

$f_i: Y \rightarrow X_i$  ( $i=1,2$ )

$\langle f_1, f_2 \rangle: Y \rightarrow X_1 \times X_2$

$y \mapsto (f_1(y), f_2(y))$

kartesisches Produkt von Projektionen. (lifting)

$g_i: Y_i \rightarrow X_i$

$g_1 \times g_2: Y_1 \times Y_2 \rightarrow X_1 \times X_2$

$(y_1, y_2) \mapsto (g_1(y_1), g_2(y_2))$

Beispiele:

$\pi_i \cdot \langle f_1, f_2 \rangle = f_i$

$\langle \pi_1, \pi_2 \rangle = id, \langle \pi_1 \cdot f, \pi_2 \cdot f \rangle = f$

$g_1 \times g_2 = \langle g_1 \cdot \pi_1, g_2 \cdot \pi_2 \rangle$

$X_1 + X_2$  (addition als analog zur Multiplikation mit kartesischen Produkt)

$$X_1 + X_2 = \{(i, x) | i \in \{1, 2\}, x \in X_i\}$$

(umbennnen, damit keine zwei elemente sich gegenseiteig in der Menge auslöschen) Duale

$$X_i \rightarrow X_1 + X_2$$

$$x \mapsto (i, x)$$

$$\text{zu } f_i : X_i \rightarrow Y \ (i = 1, 2)$$

kopaar.

$$[f_1, f_2] : X_1 + X_2 \rightarrow Y$$

$$[f_1, f_2](i, x) \mapsto f_i(x) \text{ (man wahlt also die i-te funktion aus. effektiv "case" statement)}$$

$$\text{zu } g_i : X_i \rightarrow Y_i \ (i = 1, 2)$$

$$g_1 + g_2 : X_1 + X_2 \rightarrow Y_1 + Y_2$$

$$(g_1 + g_2)(i, x) \mapsto (i, g_i(x))$$

**Lemma 11.3.** *Duale:*

$$[f_1, f_2] \circ in_i = f_i$$

$$r \circ in_1, r \circ in_2 = r$$

$$g_1 + g_2 = [in_1 \circ g_1, in_2 \circ g_2]$$

$$\text{Beweis 11.4. } [f_1, f_2](in_i(x)) = [f_1, f_2](i, x) = f_i(x)$$

$$[r \circ in_1, r \circ in_2](i, x) = r(in_i(x)) = r(i, x)$$

$$[in_1 \circ g_1, in_2 \circ g_2](i, x) = in_i(g_i(x)) = (i, g_i(x)) = (g_1 + g_2)(i, x)$$

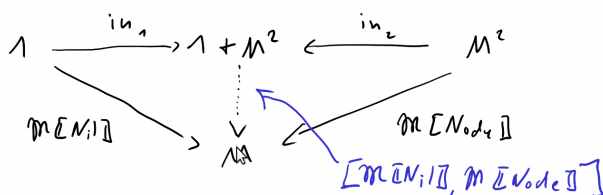
**Beispiel 11.3.** Baume

```
data Tree where
  Nil: () -> Tree
  Node: Tree*Tree -> Tree
```

Eine  $\Sigma_{Tree}$ -Algebra  $\mathfrak{M}$ :

$$\mathfrak{M}[Nil] : 1 \rightarrow M \text{ (wobei 1 unit } \{*\} \text{ ist)}$$

$$\mathfrak{M}[Node] : M^2 \rightarrow M$$



d.h.  $\mathfrak{M}$  gegeben durch  $\alpha : 1 + M^2 \rightarrow M$

$$(\mathfrak{M}[Nil] = \alpha(in_1(*)) \text{ und } \mathfrak{M}[Node] = \alpha(in_2(x, y)))$$

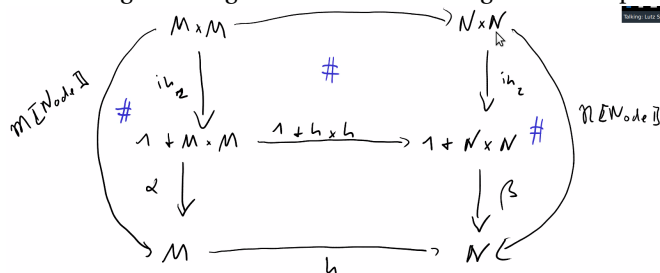
$h$  homomorph bezüglich Node  $\iff h(\mathfrak{M}[\text{Node}](x, y)) = \mathfrak{N}[\text{Node}](h(x), h(y))$

$$\begin{array}{ccc} M \times M & \xrightarrow{f \times f} & N \times N \\ \mathfrak{M}[\text{Node}] \downarrow & & \downarrow \mathfrak{N}[\text{Node}] \\ M & \xrightarrow{f} & N \end{array}$$

$$\begin{array}{ccc} 1 & \xrightarrow{1} & 1 \\ \mathfrak{M}[\text{Nil}] \downarrow & & \downarrow \mathfrak{N}[\text{Nil}] \\ M & \xrightarrow{f} & N \end{array}$$

$$\begin{array}{ccc} 1 + M \times M & \xrightarrow{1 + f \times f} & 1 + N \times N \\ [\mathfrak{M}[\text{Nil}], \mathfrak{M}[\text{Node}]] \downarrow & & \downarrow [\mathfrak{N}[\text{Nil}], \mathfrak{N}[\text{Node}]] \\ M & \xrightarrow{f} & N \end{array}$$

Erinnerung die Eckigen klammern heißen: gib mir vom paar  $[f_1, f_2](i, x)$  die  $i$ -te funktion angewandt auf  $x$ .



Da jedes Teildiagramm kommutiert, gilt homomorphie.

(Das gilt genau so für NIL (wenn man  $in_2$  verwendet))

Allgemein gilt für eine  $\Sigma$ -Algebra

$$= \alpha : \underbrace{\sum_{f/n \in \Sigma} M^n}_{=: F_{\Sigma} M} \rightarrow M$$

Dann  $h\mathfrak{M} \rightarrow \mathfrak{N}$  Homomorphismus  $\iff F_{\Sigma}$

$$\begin{array}{ccc} \sum_{f/n \in \Sigma} M^n & \xrightarrow{\sum_{f/n \in \Sigma} h^n} & \sum_{f/n \in \Sigma} N^n \\ \alpha \downarrow & & \downarrow \beta \\ M & \xrightarrow{f} & N \end{array}$$

sprich, wenn man für alle signatursymbole  $f$  ein solches signaturquadrat bauen kann, dann gilt homomorphismus.

$$(F_{\Sigma} h = \sum_{f/n \in \Sigma} h^n)$$

Man kann also alles in eine Abbildung darstellen: das  $h$  wird einfach der Tuple aller homomorphismen eines signa-

tursymbole!

## 11.2 Initialität und Rekursion

Datentyp  $\Sigma : [\Sigma]$  Initiale  $\Sigma$ -Algebra.

zu  $f/n$  wähle  $a_f : A^n \rightarrow A$  (konstruktion einer Algebra zu  $\Sigma$ )

Dan existiert ein eindeutiger homomorphismus (per def. der initialität)  $h : [\Sigma] \rightarrow A$  mit

$$h(f(\underbrace{x_1, \dots, x_n}_{\substack{\text{konstruktorargumente} \\ \text{konstruktorpattern}}})) = a_f(\underbrace{h(x_1), \dots, h(x_n)}_{\text{rekursive Aufrufe von } h})$$

Rekursive Definition von  $h$  (definiert eindeutig  $h$ ) also  $a_f$  sind irgendwelche funktionen, die etwas vom typ  $A$  liefern.

(man wechselt also von  $[\Sigma]$  auf  $A$  mithilfe eines homomorphismus)

man nennt dies das **fold-Schema**:  $h = \text{fold } (\lambda y \dots y_n. a_f)_{f \in \Sigma}$  (sprich  $a_f$  ist eine totale funktion über  $f$ .)

z.B. Tree:  $(foldc) : Tree \rightarrow a$  mit  $c : a, g : a \rightarrow a \rightarrow a$

```
fold c g Nil = c
fold c g (Node x y) = g (fold c g x) (fold c g y)
-- sprich man baut hier ein pattern auf, dass einmal eine interpretation a_NIL = c
-- und eine interpretation a_Node(x,y) = g erwartet
```

$\text{fold } 1 * = \lambda x. 1$  (also multiplikation aller blätter, wobei jedes blatt wert 1 hat)

$\text{fold } 1 + = \lambda x. \#leaves(x)$

```
fact 0 = 1
fact (suc n) = (suc n) * (fact n)
--problem man braucht auch suc n, nicht nur das ergebnis des rekursiven aufrufs fact!
```

Dies kann man jedoch lösen:

$h = \langle fact, id \rangle$  wenn man das programmieren kann, dann kann man das im fold schema schreiben, da in  $h$  ja die  $id$  drinnen ist.

```
h 0 = (1,0)
h (suc n) = (( succ (snd (h n)) ) * (fst (h n)), suc (snd (h n)))
-- snd ist das selbe wie die 2.projektion pi_2
```

Mehrstellige Funktionen:

$+: Nat \times Nat \rightarrow Nat$

$0+k=k$

$(suc\ n)+k = suc\ (n+k)$

Currying:  $+: Nat \rightarrow (Nat \rightarrow Nat)$

$+= fold(\lambda k.k) \underbrace{(\lambda f k.Suc(f\ k))}_{(Nat \rightarrow Nat) \rightarrow (Nat \rightarrow Nat)}$  primitive Rekursion:

Rekursion über Argumente verwendet rekursive Aufrufe auf Konstruktorargumente.

### 11.3 Mehrsortige Typen

```
data Tree a, Forest a where
    Leaf: a -> Tree a
    Node: Forest a -> Tree a
    Nil: () -> Forest a
    Cons: Tree a * Forest a -> Forest a
    -- bzw Cons: Tree a -> Forest a -> Forest a
```

mehrsortige Signatur, hier Sortenmenge  $S = \underbrace{\{a\}}_{S_0}, Tree\ a, Forest\ a\}$

**Definition 11.5.** Eine sortierte Signatur  $\Sigma = (S_0, S, F)$  besteht aus:

- Einer Menge  $S$  von Sorten
- Menge  $S_0 \subseteq S$  von Parametern
- Eine Menge  $F$  von Funktionssymbolen bzw. Konstruktoren  $c$  mit Profilen  $c : a_1 \times \dots \times a_n \rightarrow b$  mit  $n \geq 0$   $a_1, \dots, a_n \in S$  und  $b \in S \setminus S_0$  (sprich wir das Ziel soll nicht wieder ein Parameter sein  $Tree\ a \rightarrow Forest\ a$ )

**Definition 11.6.** .

- Ein Kontext ist eine Menge  $\Gamma = \{x_1 : a_1, \dots, x_k : a_k\}$  mit Sorten  $a_i \in S$  paarweise verschiedenen Variablen  $x_i$ . (also endliche partielle Abbildung  $\Gamma : Var \rightarrow Sorte$ )
- $\Gamma \vdash t : a$  "Term  $t$  hat im Kontext  $\Gamma$  Sorte  $a$ "

definier induktiv:

$\frac{}{\Gamma \vdash x : a} \quad (x : a \in \Gamma)$

$$\frac{\Gamma \vdash t_1 : a_1, \dots, \Gamma \vdash t_n : a_n}{\Gamma \vdash c(t_1, \dots, t_n) : b} \quad (c : a_1 \times \dots \times a_n \rightarrow b)$$

Wir schreiben  $T_\Sigma(\Gamma)_a = \{t \text{ Term} \mid \Gamma \vdash t : a\}$  für die Menge der Terme der Sorte  $a$  im Kontext  $\Gamma$ .

z.B.  $x : a \vdash \text{Node}(\text{Cons}(\text{Node Nil}) (\text{Cons}(\text{Leaf } x) \text{Nil})) : \text{Tree } a$

**Definition 11.7.** Sei  $V = (V_a)_{a \in S_0}$  eine Mengenfamilie. Eine mehrsortige  $\Sigma$ -Algebra  $\mathfrak{M}$  über  $V$  besteht aus

- für  $a \in S$  Menge  $\mathfrak{M}[[a]] = V_a$
- einer Abbildung

$$\mathfrak{M}[[c]] : \mathfrak{M}[[a_1]] \times \dots \times \mathfrak{M}[[a_n]] \rightarrow \mathfrak{M}[[b]]$$

für jedes  $c : a_1 \times \dots \times a_n \rightarrow b$

Gegeben  $\mathfrak{M}$  Umgebung  $\eta$  für  $\Gamma$  (d.h.  $\eta(x) \in \mathfrak{M}[[a]]$  für jede  $\Gamma(x) = a$ ),  $\Gamma \vdash t : a$

$$\mathfrak{M}[[t]]_\eta \in \mathfrak{M}[[a]]$$

rekursiv definiert wie bisher.

Ein  $\Sigma$ -Homomorphismus  $h : \mathfrak{M} \rightarrow \mathfrak{N}$  über  $V$  besteht aus Abbildungen (ist also selbst mehrsortig)

$$h_a : \mathfrak{M}[[a]] \rightarrow \mathfrak{N}[[a]] \quad (a \in S)$$

mit  $h_a = id$  für  $a \in S_0$  (Parameter werden in Ruhe gelassen)

sodass

$$h_b(\mathfrak{M}[[c]](x_1, \dots, x_n)) = \mathfrak{N}[[c]](h_{a_1}(x_1), \dots, h_{a_n}(x_n))$$

für  $c : a_1 \times \dots \times a_n \rightarrow b \in \Sigma$

Termalgebra

Zu  $V = (V_a)_{a \in S_0}$  setze  $\Gamma(V) = \{x : a \mid a \in S_0, x \in V_a\}$

Termalgebra  $[\Sigma]_V$  (Semantik des Datentyps  $\Sigma$  gegeben  $V$ )

$$[\Sigma]_V[[a]] = T_\Sigma(\Gamma(V))_a \quad (a \in S)$$

$$[\Sigma]_V[[c]](t_1, \dots, t_n) = c(t_1, \dots, t_n)$$

**Satz 16.** *Initialität/Fold-Prinzip*

Für jede  $\Sigma$ -Algebra  $\mathfrak{N}$  über  $V$  existiert genau ein  $\Sigma$ -Homomorphismus

$$h : [\Sigma]_V \rightarrow \mathfrak{N}$$

über  $V$ . (Termauswertung  $\eta(x) = x$ ), sonst wie vorher

$\underbrace{fold(Algebra)}_{\text{mehrere Abbildungen!}}: \llbracket \Sigma \rrbracket_V \rightarrow \dots$

**Beispiel 11.4.**  $\Sigma_{Tree}$ -Algebra  $\mathfrak{N}$  über  $V_a = \mathbb{N}$  (es gab hier ja nur eine...)  $\mathfrak{N}[\llbracket Tree\ a \rrbracket] = \mathbb{N} = \mathfrak{N}[\llbracket Forest\ a \rrbracket]$  (also hier forest und tree gleich interpretation)  $\mathfrak{N}[\llbracket Leaf \rrbracket](n) = n$

$\mathfrak{N}[\llbracket Node \rrbracket](n) = n$

$\mathfrak{N}[\llbracket Cons \rrbracket](n, m) = n + m$

$\mathfrak{N}[\llbracket Nil \rrbracket] = 0$

primitive Rekursion (gegenseitig)  $\triangleleft$  Alle interpretationen müssen auf ein mal definiert werden!

## 11.4 Strukturelle Induktion über Datentypen

```
data List a where
    Nil: () -> List a
    Cons: a -> List a -> List a
concat :: List a -> List a -> List a
concat Nil k = k
concat (Cons x l) k = Cons x (concat l k)
```

Beweisen Assoziativität:  $cc\ l\ (cc\ k\ v) = cc\ ((cc\ l\ k)\ v)$

Strukturelle Induktion: Ein Fall pro Konstruktor

I.V. jeweils für Konstruktorargumente (also immer induktion über das argument, dass auch rekursion betreibt)

hier über  $l$

-Nil:  $cc\ Nil\ (cc\ k\ v) = cc\ k\ v = cc\ (cc\ Nil\ k)\ v$

Cons:  $cc\ (Cons\ x\ l)\ (cc\ k\ v) = Cons\ x\ (cc\ l\ (cc\ k\ v)) \stackrel{IV}{=} Cons\ x\ (cc\ (cc\ l\ k)\ v) = cc\ (cc\ (cons\ x\ l)\ k)\ v = cc\ (Cons\ x\ (cc\ l\ k))\ v =$   
 $Cons\ x\ (cc\ (cc\ l\ k)\ v)$

## 11.5 strukturelle Induktion über mehrsortige Datentypen

```
mirrort :: Tree a -> Tree a
--wald an senkrechter achse spiegeln, also wald und jeden baum an einer senkrechten achse spieg
mirrorf :: Forest a -> Forest a
mirrort (Leaf x) = Leaf x
mirrort (Node f) = Node (mirrorf f)
mirrorf Nil = Nil
mirrorf (Cons t f) = concat (mirrorf f) (Cons (mirrort t) Nil)
--sortiertes flattening nach baumreihenfolge
```

```

flattent: Tree a->List a
flattenf: Forest a->List a
flattent (Leaf x) = [x]
flattent (Node f) = flattenf f
flattentf Nil = []
flattenf (Cons t f) = concat (flattent t) (flattenf f)
rev:List a->List a
rev Nil = Nil
rev (Cons x y) = concat (rev y) [x]

```

Behauptung: flattent (mirror t) = rev (flattent t)

Induktion über t:

flattent (mirror (Leaf x)) = [x] = rev (flattent (Leaf x))

Leaf geht, Node:

flattent (mirror (Node f)) = flattent (Node (mirror f)) = flattenf (mirror f)

andere seite:

rev (flattent (Node f)) = rev (flattenf f)

Induktionsbehauptung für f: Forest a

flattenf (mirror f) = rev (flattenf f)

dann gilt "rev (flattenf f) = flattenf (mirror f)" per I.V.

jetzt müssen wir die Induktionsvoraussetzung für Wälder anwenden: sprich dass flattenf (mirror f) = rev (flattenf f)

für alle Wälder gilt

flattenf (mirror Nil) = [] = rev (flattenf Nil)

gilt also für Nil, jetzt Cons

$$\begin{aligned}
 \text{flattenf (mirror (Cons t f))} &= \text{flattenf (concat (mirror f) (Cons (mirror t) Nil))} \stackrel{=}{=} \text{Lemma A concat (flattenf (mirror f))} \\
 &= \text{(flattenf (Cons (mirror t) Nil))} \stackrel{=}{=} \text{Lemma B concat } \underbrace{(\text{flattenf (mirror f)})}_{IVForest} \underbrace{(\text{flattent (mirror t)})}_{IVTree} \\
 &= \text{concat (rev (flattenf f)) (rev (flattent t))}
 \end{aligned}$$

andere seite

rev (flattenf (Cons t f)) = rev (concat (flattent t) (flattenf f))

ist gleich (Vgl Übung, bzw umgekehrte concatenation von zwei invertierten listen ist gleich der invertierten list)

Lemma A: flattenf (concat f g) = concat (flattenf f) (flattenf g)

Beweis: Induktion über f (aber hier haben wir keine Bäume!! I.A. für bäume könnte T sein, der Beweis ist unabhängig).

Trivial beweisbar via induktion.

Lemma B: flattenf [t] = flattent t (trivial, ausrechnen)

flatten einfach als fold:

fold  $\lambda x.[x]$  id [] cc

mirror als fold



fold Leaf Node Nil ( $\lambda x y. cc\ y[x]$ )

das einzige, was bei mirror “was macht” ist Cons, bei dem werden die resultate der unteren schicht in umgekehrter Reihenfolge verbunden (und das erste zum forst “casten”)

## 11.6 Kodaten

Daten werden konstruiert *Cons x Nil*

Kodaten werden destruiert/beobachtet

z.B.  $A^\omega = \{a_0, a_1, a_2, \dots \mid a_i \in A\}$  streams über A. (omega als Menge der ordinales, für uns einfach Nat)

hd (head):  $A^\omega \rightarrow A$  gibt  $a_0$  aus  $a_0 : a_1 a_2 \dots$  zurück.

tl (tail):  $A^\omega \rightarrow A^\omega$

$(a_0, a_1, \dots) \mapsto (a_1, a_2, \dots)$

```
codata Stream where
  hd: Stream -> A
  tl: Stream -> Stream
```

Im gegensatz zu Datentypen, die über konstruktoren definiert ist, sind Kodatentypen über destructoren definiert.

```
map f: Stream -> Stream
hd(map f s) = f (hd s)
tl (map f s) = map f (tl s)
```

Definition von map, allerdings terminiert map nie (soll es schließlich auch nicht).

Zunächst  $\langle hd, tl \rangle: A^\omega \rightarrow A \times A^\omega$

wir produzieren also aus  $A^\omega$  eine ding aus der Trägermenge hinaus. (Destruction)

Koalgebra: Träger M, Abb  $\alpha: M \rightarrow A \times M$

$A^\omega$  ist Koalgebra (genauer  $(A^\omega, \langle hd, tl \rangle)$  ist Koalgebra)

weiter Koalgebra:

$(A^\omega, \langle f \circ hd, tl \rangle)$ , denn  $A^\omega \xrightarrow{hd} A \xrightarrow{f} A$

$$\begin{array}{ccc}
 A^\omega & \xrightarrow{hd} & A \\
 \downarrow \langle f \circ hd, tl \rangle & \neq & \downarrow \langle hd, tl \rangle \\
 A \times A^\omega & \xrightarrow{id_A \times hd} & A \times A^\omega
 \end{array}$$

$id_A = \textcircled{A} \times hd$

explizit:

linke Komponente:  $\text{hd}(\text{h}(s)) = f(\text{hd}(s))$

rechte Komponente:  $\text{tl}(\text{h}(s)) = \text{tl}(\text{h}(s))$

Mengenoperatoren: (polynomialfunktoren)

$G ::= A | id | (G_1 + G_2) | G_1 \times G_2$  A Menge (zum eingeklammerten werden wir nicht kommen)

$GX$  rekursiv definiert:

$$(A)X = A$$

$$id(X) = X$$

$$(G_1 + G_2)X = G_1X + G_2X$$

$$(G_1 \times G_2)X = G_1X \times G_2X$$

entsprechend mit abbildungen  $Gf: GX \rightarrow GY$  ( $f: X \rightarrow Y$ )

**Definition 11.8.** Eine G-Koalgebra

besteht aus einer Menge  $M$  (von "Zuständen") und Abbildung  $\alpha: M \rightarrow GM$  (Transitionsabbildung, welche Zustände sind von  $M$  aus erreichbar).

**Beispiel 11.5.** .

$$G = A \times id$$

G-Koalgebra = Koalgebra oben.  $M \rightarrow A \times M$

## Literatur

- [1] Knuth-bendix algorithm for creating CR TES from terminating TES  
[https://en.wikipedia.org/wiki/Knuth%E2%80%93Bendix\\_completion\\_algorithm](https://en.wikipedia.org/wiki/Knuth%E2%80%93Bendix_completion_algorithm)
- [2] dependant choice  
[https://de.wikipedia.org/wiki/Axiom\\_der\\_abh%C3%A4ngigen\\_Auswahl](https://de.wikipedia.org/wiki/Axiom_der_abh%C3%A4ngigen_Auswahl)
- [3] nominale Mengen  
<https://www.tcs.ifi.lmu.de/mitarbeiter/martin-hofmann/publikationen-pdfs/c43-nominalrenamingsets.pdf>