Inhaltsverzeichnis

Vorlesung 2

Alexander Mattick Kennung: qi69dube

Kapitel 1

27. Juli 2020

1 Terme

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

V Menge von Variablen.

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

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

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

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

1.1 substitution

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

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

1.2 Kontext

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

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

2 Operationelle Semantik von TES

$\to_0 \subseteq T_\Sigma(V) \times T_\Sigma(V)$

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

$\rightarrow = \{(C(s\sigma), C(t\sigma)) | t \rightarrow_0 s, C(\cdot)$Kontext$, \sigma$ 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** $\iff \neg\exists s(t \rightarrow s) \iff t \not\rightarrow$ 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 bzg. 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 kontextabg.* $\iff$ *R 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:

$(s(x) + s(y)) + z \overset{1),C(\cdot)+z,\sigma=[s(y)/y]}{\rightarrow} s(x + s(y)) + z \overset{1),C(\cdot),\sigma=[(x+s(y))/x]}{\longrightarrow} s((x + s(y)) + z) \overset{3)}{\longrightarrow} s(x + (s(y) + z) \overset{1)}{\rightarrow} s(x + s(y + z))$

$(s(x) + s(y)) + z \overset{3),C(\cdot),\sigma=[s(x)/x,s(y)/y]}{\rightarrow} s(x) + (s(y) + z) \overset{1)}{\rightarrow} s(x) + s(y + z) \overset{1)}{\rightarrow} s(x + s(y + z))$

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 Induktionsvorraussetzung!

Definition 3.2.

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

TES $(\Sigma, \to_0)$ schwach/stark normalisierend (WN(SN)) $\iff$ alle t in $(\Sigma, \to_0)$ schwach/stark normalisierend.

Beispiel 3.1.

$f(x) \to_0 f(x)$

$g(x) \to 1$

$g(x)$ stark normalisierend einzige Reduktion $g(x) \to 1 \not\to$

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

$g(f(x))$ schwach normalisierend:$g(f(x)) \to 1 \not\to$(Haskell ausführung)

oder $g(f(x)) \to g(f(x)) \to \dots$ (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 $xRxRxRxR\dots$).

Satz 1. *Sei $>$ Reduktionsordnung und $\forall t, s(t \to_0 s \implies t > s) \implies \to$ SN (also in jeder Ersetzungsregel wird nach anwendung der Term kleiner).*

Beweis 4.1. $>$ ist stabil und kontextabgeschlossen, und $\to_0 \subseteq > \implies \to \subseteq > \implies \to$ ist wohlfundiert, d.h. $\to$ ist SN.

(Weil $\to$ der kontextabg. und stabile abschluss von $\to_0$ ist, wenn $>$ wohlfundiert ist, dann kann es auch keine unendlichen Mengen in der Teilmenge $\to$ 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: $\sigma = [100x/y]$ $|x + 2 * 100x - x| \not> |100x + 100x|$!

aber Ok, wenn in $t \to_0 s$ stets jede variable s höchstens so oft wie in t vorkommt.

$\emptyset$ ist eine Reduktionsordnung.

$\to$ SN $\implies \to^+$ Reduktionsordnung.

4.1 Polynomordnungen

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

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

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

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

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 $A = \{n \in \mathbb{N} | n \geq 2\}$

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. $p \in \mathbb{N}[x_1,\dots,x_n]$ <u>streng monoton</u>:

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

(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)* $\forall j(k_j \geq k'_j)$ *und 2)* $\exists j(k_j > k'_j)$ *(mindestens eins echt größer).*

Beweis 4.3. " $\implies$ "

$a_{i_1,\ldots,i_n} k_1^{i_1} \ldots k_n^{i_n} \geq a_{i_1,\ldots,i_n} k_1^{i'_1} \ldots k_n^{i'_n}$ stets, einmal ">"□

Definition 4.4. (monotone) Polynomielle Interpretion $\mathscr{A}$ besteht aus

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

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

(Eine polynomordnung besteht aus einem polynom für jedes signatursymbol und einer auswahl natürlicher Zahlen)

$\rightarrow$ Polynomordnung $\succ_{\mathscr{A}}$ $t \succ_{\mathscr{A}} s \iff p_t \succ_{\mathscr{A}} p_s$

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

Satz 2. $\succ_A$ *ist eine Reduktionsordnung!*

Korrolar 4.1. *Wenn* $t \rightarrow_0 s \implies t \succ_{\mathscr{A}} s$*, 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 $f(f(g(x))) \equiv (x^2+1)^2+1 \succ_{\mathbb{N}} f(g(g(x))) = x^2+1$

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):*

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

Beweis 4.4. Induktion über t.

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

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

$\underset{IV}{=} p_f(p_{s_1}(p_{t_1},\ldots,p_{t_n}),\ldots)$

$\overset{substitution}{=} p_f(p_{s_1},\ldots,p_{s_k})(p_{t_1},\ldots,p_{t_n}) = p_{f(s_1,\ldots,s_n)}(p_{t_1},\ldots,p_{t_n})$

Beweis 4.5. ($\succ_{\mathscr{A}}$ ist Reduktionsordnung)

- strikte Ordnung per definition
- wohlfundiert (es gibt keine endlos absteigende polynomfolge)
- $\succ_{\mathscr{A}}$ stabil: Sei $t \succ_{\mathscr{A}} s, \sigma = [t_1/x_1,\ldots]$

zZ.: $t\sigma >_{\mathscr{A}} s\sigma$: Seien $k_1,\dots,k_n \in A$

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

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

  zZ: $C(t) >_{\mathscr{A}} C(s)$ Seien $k_1,\dots,k_n \in A$

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

Es ist beweisbar untentscheidbar, 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 "⊕":

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

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

führt zu:

$(x^2+y)^2 + z >_{\mathscr{A}} x^2 + (y^2+z) = x^4 + 2x^2y + y^2 + z$

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

$x^2 + y^2 + z \not>_{\mathscr{A}} y^2 + y$

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

Besser:

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

$(x^2+xy)^2 + (x^2+xy)z = x^4 + 2x^3y + x^2y^2 + x^2z + xyz >_{\mathscr{A}} x^2 + x(y^2+yz) = x^2 + xy^2 + xyz$

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

$x^2 + xy^2 + xyz >_{\mathscr{A}} y^2 + yy = 2y^2$

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

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

5 Konfluenz

Beispiel 5.1. Gruppen

$x \cdot (y \cdot z) \overset{\rightarrow_0}{=} (x \cdot y) \cdot z$

$x \cdot e \overset{\rightarrow_0}{=} x$

$x \cdot x^{-1} \overset{\rightarrow_0}{=} e$

$y \cdot (x \cdot x^{-1}) \rightarrow y \cdot e \rightarrow y \not\rightarrow$ ist eine NF

oder

$y \cdot (x \cdot x^{-1}) \rightarrow (y \cdot x) \cdot x^{-1} \not\rightarrow$ 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) $\iff \exists u(t \to^* u\ {}^*\!\leftarrow s)$ (u.U auch mit null schritten)
- TES T heißt **konfluent** (CR, church/Rosser) $\iff \forall t, s, s'(t \to^* s \wedge t \to^* s' \implies s, s'\ zf)$

$$\begin{tikzcd}
& t \arrow[dl, "*"'] \arrow[dr, "*"] & \\
s \arrow[dr, dotted, "*"'] & & s' \arrow[dl, dotted, "*"] \\
& q &
\end{tikzcd}$$

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

$$\begin{tikzcd}
& t \arrow[dl] \arrow[dr] & \\
s \arrow[dr, dotted, "*"'] & & s' \arrow[dl, dotted, "*"] \\
& q &
\end{tikzcd}$$

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

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

1) $s \leftrightarrow^* t \iff s, t\ zf$

2) s, s' *NF von* $t \implies s = s'$

Beweis 5.1.

1) $\Longleftarrow$ klar $\implies$

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

Induktion über n:

$n = 0 : s = t$ klar

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

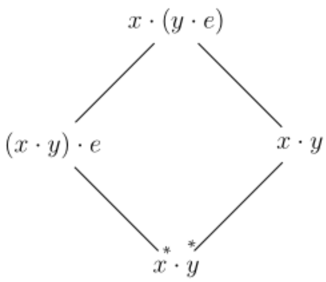
Fall 1: $t_n \leftarrow t_{n+1}$ fertig (weil t_n über q mit s zusammenführbar)

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

$$\begin{tikzcd}
& t_n \arrow[r] \arrow[d, "*"] & t_{n+1} \arrow[d, dashed, "*"] \\
t_0 \arrow[r, "*"] & q \arrow[r, dashed, "*"] & r
\end{tikzcd}$$

2) $s \leftrightarrow^* s' \implies s, s'\ zf : s \to^* u^* \leftarrow s'$ weil s, s' 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 untentscheidbar (und so auch SN, deshalb widerspricht dieser Satz dem nicht...)) Beweis, später

Beispiel 5.2.

Regeln

$l_1 \to r_1$

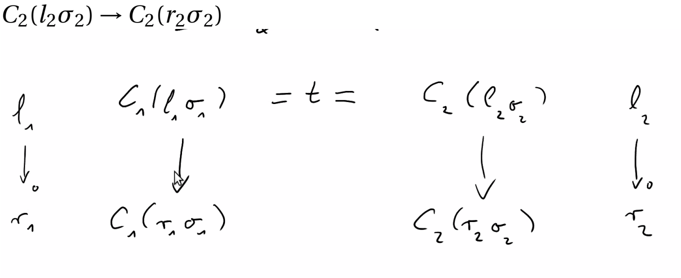
$l_2 \to r_2$

Terme

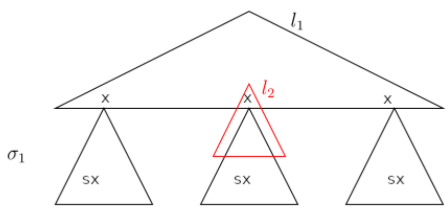
$C_1(l_1\sigma_1) = t = C_2(l_2\sigma_2)$

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

$C_2(l_2\sigma_2) \to C_2(r_2\sigma_2)$



C_1 kann ignoriert werden wegen kontextabgeschlossen.



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

Definition 5.2. Unifikation

t, s Terme $t \doteq s$

σ Unifikator von t,s ($\sigma \in Unif(t,s)$) $\iff$ $t\sigma = s\sigma$ (syntaktisch)

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

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

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

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

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 $l_1 \to_0 r_1, l_2 \to_0 r_2$,

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

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

$\sigma = mgu(t, l_2)$

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

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

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

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

Korrolar 5.1. *T $WCR \implies$ 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, den jede reduktion selbst benötigt)

Beispiel 5.4. (Gruppe)

$(l_1 \to_0 r_1) = (x \cdot (y \cdot z)) \to_0 (x \cdot y) \cdot z)$

(in frische variablen umbennenen)

$(l_2 \to_0 r_2) = (x' \cdot e \to x')$

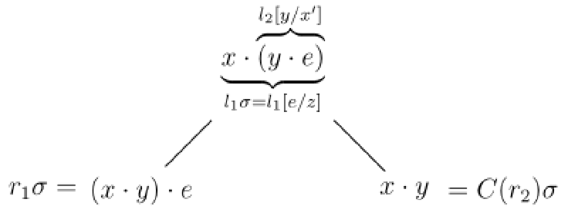
Jetzt: wähle einen Teilterm aus, und mach das "t" draus:

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

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

$\rightarrow$ 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.

$l_1 \longrightarrow_0 r_1 = (x \cdot (y \cdot z)) \longrightarrow_0 (x \cdot y) \cdot z)$

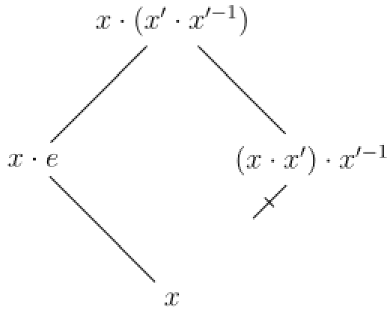
$l_2 \longrightarrow_0 r_2 = (x' \cdot x^{-1'} \longrightarrow_0 e)$

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

$t = y \cdot z$

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

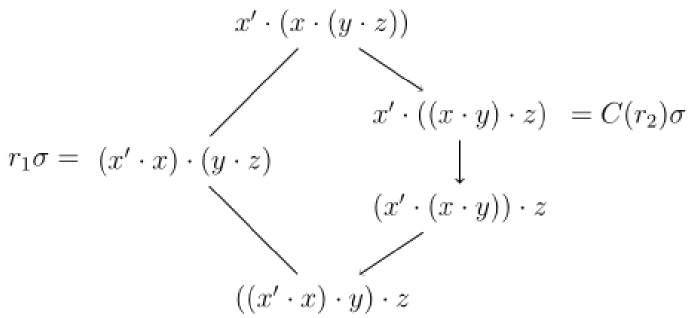
$(x \cdot x') \cdot x^{'-1} \leftarrow x \cdot (x' \cdot x^{-1'}) \rightarrow x \cdot e$ nicht z.f.



Beispiel 5.6. $l_1 \longrightarrow_0 r_1 = (x \cdot (y \cdot z) \longrightarrow_0 (x \cdot y)z) = l_2 \longrightarrow_0 r_2$

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

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



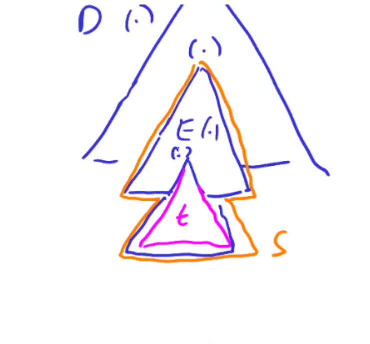
sind z.f.

<u>**ACHTUNG**</u>

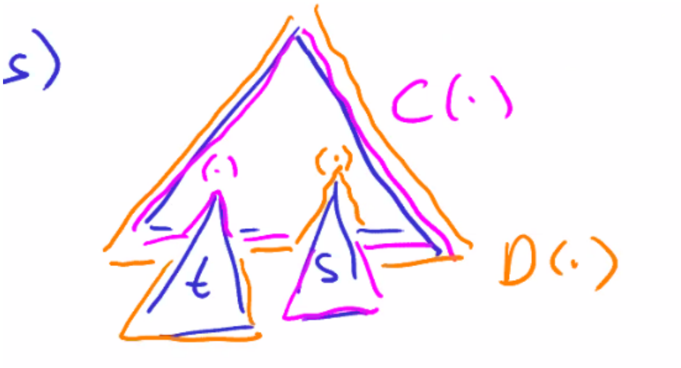
wenn man das umbennenen der variablen vergisst, dann krigt man $y \cdot z$ und $x \cdot (y \cdot z) \longrightarrow \perp$ occurs!!

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)

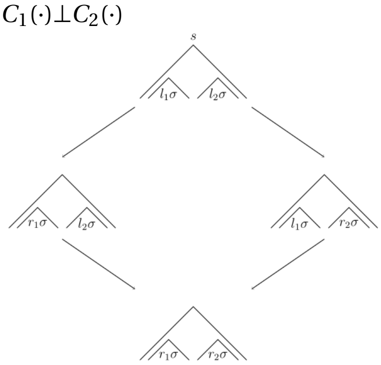


$C(\cdot) \perp D(\cdot) \iff C(\cdot) \not\sqsubseteq D(\cdot) \land D(\cdot) \not\sqsubseteq C(\cdot)$ (also keiner ist subset des anderen, sie sind orthogonal)



Sei $l_1 \to_0 r_1, l_2 \to r_2$ anwendbar auf s

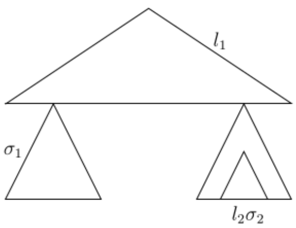
Fall 1:$C_1(\cdot) \perp C_2(\cdot)$



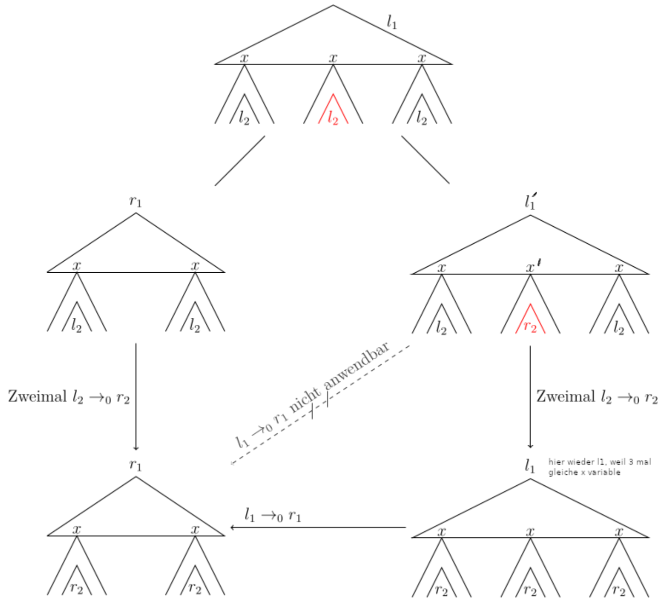
Beide Terme stören sich nicht, ich kann immer beide Regeln in beliebiger Reihenfolge anwenden

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

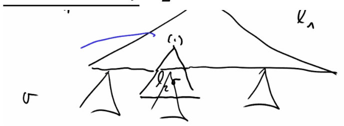


unterfall 2a) C_2 ist echt unterhalb von l_1



auf der rechten seite ist $l_1 \to r_1$ nicht mehr anwendbar, weil $l_1 \to r_1$ 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 τ

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

Satz 6. *wohlfundiert Induktion*

$R \subseteq X \times X$ wohlfundiert $\implies$

Wenn $\forall x(\forall y(xRy \implies P(y))) \implies P(x)$ (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 $(1) \land \neg(2) \implies R$ nicht wohlfundiert.

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

$\overset{\implies}{(1)}$ 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 $\forall n(P(n))$

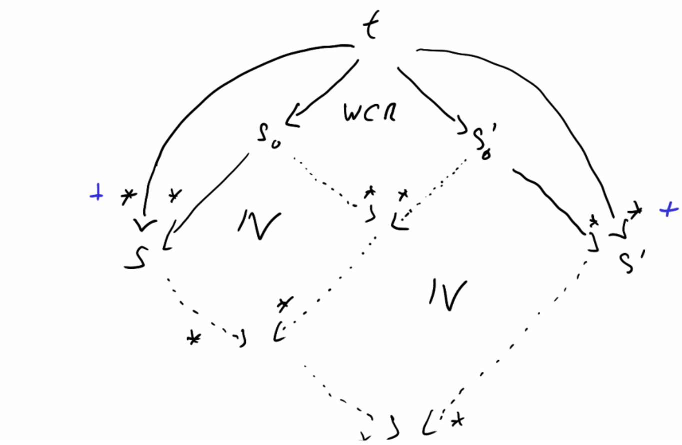
Beispiel 5.8. $X = \mathbb{N}$ R =>: Course-of-values-Induktion (man nimmt also für alle echt kleineren n die Aussage an)

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 Induktionsvorraussetzung anwenden, woraus man folgern kann, dass dies für jeden schritt möglich ist:



6 Der λ-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 λ-Kalkül = LISP.

getypter λ-Kalküle

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

7 Der ungetypte λ-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)

<u>Terme</u> t,s gegeben durch:

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

Wobei das Zweite als (λ)-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 sinnvole 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

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

Kongruenz = Kontextabgeschlossene Äquivalenz.

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

$tsu = (ts)u$

Scope von λ 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 $\sigma' = \sigma[x \to t]$)

  de-Broujin indizes. ($\lambda x.\lambda y.xy = \lambda\lambda.2\ 1$) oder nominale Mengen[3]

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

$\iff$ t geht aus s durch **Umbennenung** 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.

$\lambda x.xy =_\alpha \lambda z.zy \neq_\alpha \lambda y.yy$

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

Beweis 7.1. Es reicht: erzeugende Relation ist stabil:

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

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

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

$x \to y \to y' = x \to x' \to y'$

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

Die Rechte seite is talso gleich der linken in (2)

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

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

In λ-Kalkül gibt es sowas nicht: β-Reduktion als kontextabgeschlossene Umformung

$(\lambda x.3 + x)3 \to 3 + 4(\to$ *wenn + bekannt ist*)

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

(β) $(\lambda x.t)x \to_0 t$

$\implies$ *Einschrittreduktion* $\to$

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

$(\lambda x.t)s$ heißt β-Redex.[hier nicht: (η) $\lambda x.yx \to_0 y$, beliebt in theoriebetrachtung, aber nicht in programmiersprachen (wenn man Seiteneffekte/IO hat, macht (η) viel kaputt, weil damit die "ausführung" von x auf y umgangen wird)]

Beispiel 7.3.

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

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

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

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

  Dies liefert uns:

  $\underline{fst\ \underline{(pair\ x\ y)}} \to_\beta fst(\lambda xy.\lambda zxy)xy \to_{2\times\beta} fst(\lambda z.zxy) = (\lambda p.p\ true)\lambda z.zxy \to_\beta (\lambda z.zxy)true \to_\beta true\ xy = (\lambda xy.x)xy \to_\beta (\lambda y.x)y \to_\beta x$

7.1 Rekursion

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

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

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

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

Fixpunktkombinator fix:

fix F = F(fix F)

Satz 8. *λ-Kalkül Fixpunktkombinator*

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

2) Es existiert ein Fixpunktkombinator Y, d.h. $Yt \to_\beta s$ s ist Fixpunkt von t

$Yt \to_\beta s \overset{(1)}{\to}_\beta ts$

Beweis 7.2.

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

$s = W_t W_t = (\lambda x.t(xx))W_t \to_\beta t(W_t W_t) = ts$

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 $\omega\omega \to_\beta t(\omega\omega) \to_\beta \omega\omega$

7.2 Auswertungsstrategie

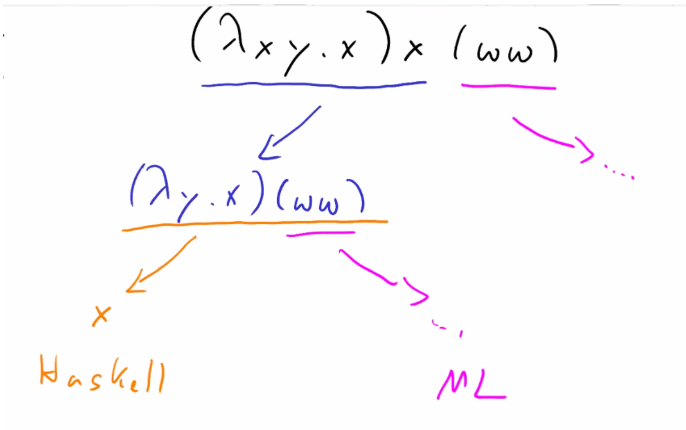
Beispiel 7.5.

$(\lambda xy.x)x(\omega\omega)$ Wenn man probiert zuerst $\omega\omega$ zu reduzieren, läuft man undendlich 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)$ts \rightarrow_a ts'$, 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 λ–Abstraktion. (wenn es eine wäre, dann 1. Regel)

4n) $ts \rightarrow_n ts'$ wenn $s \rightarrow_n s'$ und t normal und keine λ-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:*

Sei $t \rightarrow^* s$ *s Normalform* $\implies t \rightarrow_n^* s$

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

8 Der einfach getypte λ-Kalkül

Typen α, β:

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

Typvariablen a,b,...

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

Definition 8.1.

Gegebene Menge **V** var.

Typvariablen **B** von Basistypen.

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

definiert durch

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

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

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

$\omega = \lambda x.xx$ z.B. nicht typisierbar ($\lambda \to$, 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 Γ von Typisierungsannahmen $x : \alpha (x \in V)$ "x hat typ α"

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

d.h. Γ ist eine endliche partielle Abbildung. (von variablennamen auf typen)

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

Herleitbarkeit induktiv:

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

$(\to_e)$ $\dfrac{\Gamma \vdash t : \alpha \to \beta \qquad \Gamma \vdash s : \alpha}{\Gamma \vdash ts : \beta}$

$(\to_i)$ $\dfrac{\Gamma[x \mapsto \alpha] \vdash t : \beta}{\Gamma \vdash \lambda x.t : \alpha \to \beta}$ (also wenn x 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 Γ eine List statt eine Menge baut)

Beispiel 8.1.

$$\cfrac{\cfrac{Ax \quad \cfrac{}{\vdash x : a \to b, y : a \vdash x : a \to b, x : a \to b, y : a \vdash y : a} \quad AX}{\vdash x : a \to b, y : a \vdash xy : b} \to_i}{\cfrac{\vdash x : a \to b \vdash \lambda y.xy : a \to b}{\vdash \lambda xy.xy : (a \to b) \to (a \to b)} \to_i}$$

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

Berechnungsprobleme:

- gilt $\vdash t : \alpha$? (typcheck)
- finde (existiert?) α 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:

c frisch $\dfrac{\phi(c)}{\forall(\phi)}\ \forall I$

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

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

Beweis:

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

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

Lemma 8.1. *(Weakening)*

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

(also ein größerer Kontext ändert nichts an der Herleitbarkeit)

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

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

(*) Nach IV. (prämisse 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 prämisse 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* α *mit* $\Gamma \vdash t : \alpha \to \beta$ *(*$\to_e$ *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* $\alpha \to \beta$ *und* $\Gamma[x \mapsto \alpha] \vdash t : \beta$ *(*$\to_i$ *inversion, also der type des input einer Funktion muss herleitbar sein)*

Beweis 8.2. Regeln sind syntaxgerichtet

8.1 Typinferenz

$\lambda x.x : a \to a$

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

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

Definition 8.3. Terminologie/Notation:

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

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

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

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

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

$\Gamma \vdash t$ 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 Typgleichungen* $a \doteq \beta$ *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)*

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

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

- $PT(\Gamma; x; \alpha) = \{\alpha \doteq \beta | x : \beta \in \Gamma\}$ *(nach Ax inversionslemma)*
- $PT(\Gamma; ts; \alpha) = PT(\Gamma; t; a \to \alpha) \cup PT(\Gamma; s; a)$ ***global***$(a\ frisch)$ *nach* $(\to_e)$
- $PT(\Gamma; \lambda x.t; \alpha) = PT(\Gamma[x \mapsto a]; t, b) \cup \{a \to b \doteq \alpha\}$ *mit a,b* ***global*** *frisch* $(\to_i)$ *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.

$\vdash \lambda xy.xy$

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

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

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

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

jetzt hat man gleichungen, die man unifizieren muss:

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

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

$mgu = [e \to e/b, e \to e \to c/a, c/c]$ "$e \to e \to c/a, c/c$" also ist Prinzipaltyp.

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

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

Satz 11. (Γ, 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. $\sigma'|_{TV(\Gamma, \alpha} = \sigma$ sigma ist also erweiterbar.

per Induktion über t:

"$\Longleftarrow$": per Typregeln(8.2).

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

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

Nach IV. $\underbrace{\Gamma[x \mapsto a]\sigma'}_{=\Gamma[x \mapsto a]\sigma} \vdash s : \underbrace{b\sigma'}_{b\sigma}$

Per $(\to_e)$ $\dfrac{\Gamma[x \mapsto a]\sigma \qquad s : b\sigma}{\Gamma\sigma \vdash \lambda x.s : a\sigma' \to b\sigma'}$

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

" $\Longrightarrow$ " Per Inversion:

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

$\Longrightarrow \sigma \in Unif(\underbrace{PT(\Gamma; x; \alpha)}_{\alpha \doteq \beta})$

2. $\Gamma\sigma \vdash ts : \alpha\sigma \overset{Inversion}{\Longrightarrow}$

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

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

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

$\Longrightarrow \sigma'' \in Unif(PT(\Gamma, ts, \alpha))$

3. $\Gamma\sigma \vdash \lambda x.s : \alpha\sigma \overset{Inversion}{\Longrightarrow}$

$\alpha\sigma = \beta \to \gamma \; \Gamma\sigma[x \mapsto \beta] \vdash s : \gamma$

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

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

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

$\overset{(a \to b)\sigma'' = \alpha\sigma''}{\Longrightarrow} \sigma'' \in Unif(PT(\Gamma; \lambda x.s; \alpha))$

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

8.2 Subjektreduktion

Satz 12. $\Gamma \vdash t : \alpha, t \to_\beta s \Longrightarrow \Gamma \vdash s : \alpha$

⚠ *"$\Longleftarrow$" gilt **NICHT**: z.B.* $t = (\lambda x.y)(\lambda x.xx) \to_\beta y$

Lemma 8.3. *Substitution*

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

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:

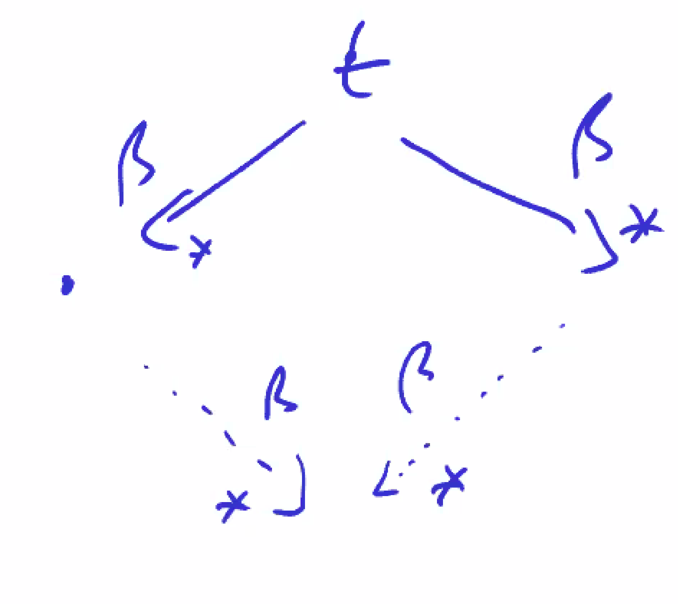
$\Gamma \vdash \lambda x.u : \beta \to \alpha, \Gamma \vdash v : \beta$

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

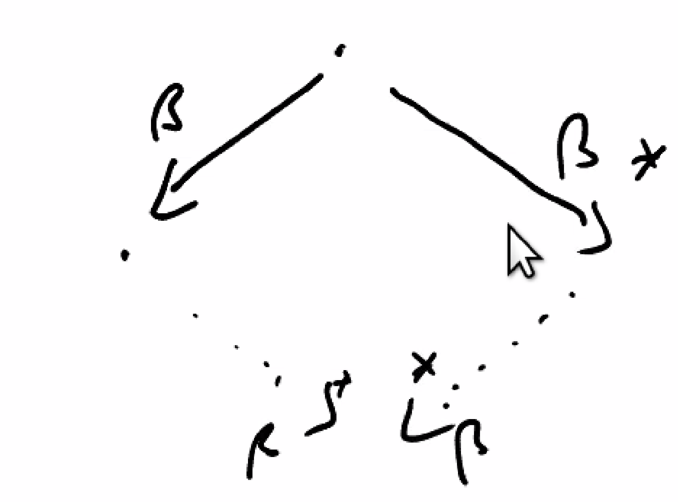
$\overset{subst-lemma}{\Longrightarrow} \Gamma \vdash \underbrace{u[v/x]}_{s} : \alpha$

9 Church Rosser des λ-Kalkül

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

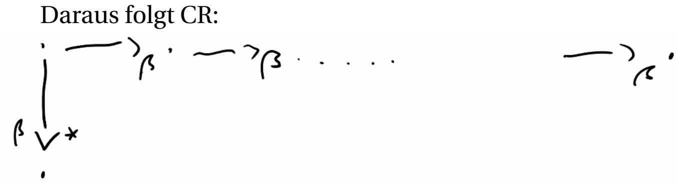


Lemma 9.1. *Streifenlemma (strip-lemma)*



Effektiv ein mittelweg zwischen WCR und CR.

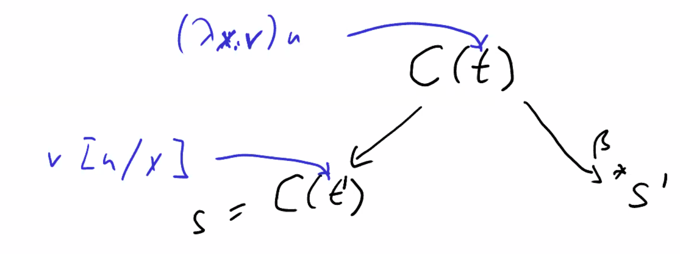
Daraus folgt CR:



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.

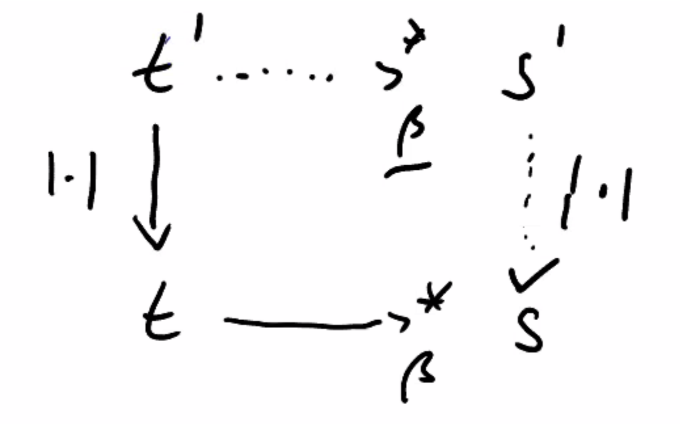
$|t|$: Entfernt _ aus t, z.B. $|(\underline{\lambda} x.t)s| = (\lambda x.|t|)|s|$

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

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

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

Lemma 9.2. *Lemma A*



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

Beweis 9.1. Reduziere auf $t \to_\beta s$, Induktion über Kontexte

Wir bekommen also nicht mehr dazu, oder verlieren ausdruckskraft.

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)

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

$u = x : LHS = s, RHS = s$ 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) ϕ *bewahrt* β

b)
$$\begin{tikzcd} t \arrow[r, "*"', "\underline{\beta}"] \arrow[d] & s \arrow[d] \\ \varphi(t) \arrow[r, dotted, "*", "\beta"'] & \varphi(s) \end{tikzcd}$$
„φ bewahrt β“

Beweis 9.3. .

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

$\phi(((\underline{\lambda} y.u)v)[s/x]) \overset{(1)}{=} \phi((\underline{\lambda} y.u[s/x])v[s/x]) = \phi(u[s/x])[\phi(v[s/x])/y]$

$\overset{IV}{=} \phi(u)[\phi(s)/x][\phi(v)[\phi(s)/x])/y]$

$\overset{synt.Subst+(1)}{=} \underbrace{\phi(u)[\phi(v)/y]}_{\phi((\underline{\lambda} y.u)v)}[\phi(s)/x]$

$= \phi((\underline{\lambda} y.u)v)[\phi(s)/x]$

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

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

$(\cdot)$: Fälle nach Markierung:

Fall 1: markiert:

$$\begin{bmatrix} (\underline{\lambda} x.u)v & \to_{\underline{\beta}} & u[v/x] \\ \phi & & \phi \\ \phi(u)[\phi(v)/x] & = & \phi(u)[\phi(v)/x] \end{bmatrix}$$

Fall 2: unmarkiert:

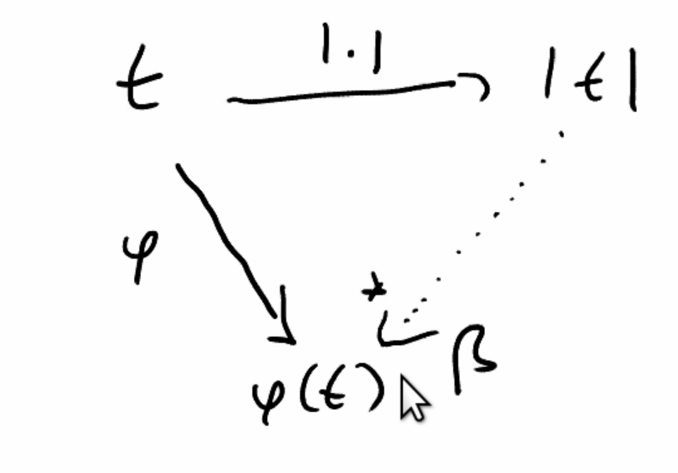
$$\begin{bmatrix} (\lambda x.u)v & \to_{\underline{\beta}} & u[v/x] \\ \phi & & \phi \\ \lambda x.\phi(u)\phi(v) & \to_{\beta} & \phi(u)[\phi(v)/x] \end{bmatrix}$$

Beispiel mit nichtleerem kontext:

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

$\phi(\underline{\lambda} x.C(t))s = \phi(C(t))[\phi(s)/x] \overset{IV,C\ ist\ kleinerer\ Kontext}{\to_{\beta}^{*}} \phi(C(s))[\phi(s)/x] \checkmark$

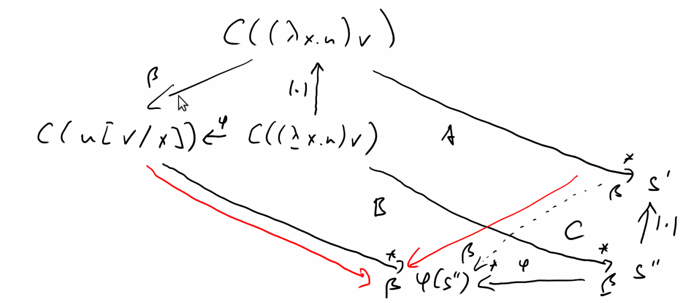
Lemma 9.5. *Lemma C*



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

$$\begin{bmatrix} (\underline{\lambda} x.u)v & \xrightarrow{|\cdot|} & (\lambda x.|u|)|v| \\ \phi & & \phi \\ \phi(u)[\phi(v)/x] & {}_{\beta}\leftarrow & (\lambda x.\phi(u))\phi(v) \end{bmatrix}$$

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



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:

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

Deduktion:

$(\to_E)\ \dfrac{\phi \to \psi \qquad \phi}{\psi}$

$(\to_I)\ \dfrac{[\phi] \vdash \psi}{\phi \to \psi}$

Sequenzenkalkül: $\underbrace{\overset{Annahme}{\Gamma}}_{MengeFormeln} \vdash \underbrace{\phi}_{Formel}$ “linke Regel” $(\to_E)$

$$(\to_E)\ \frac{\Gamma \vdash \phi \to \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi}$$
$$(\to_I)\ \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}$$
$$(Ax)\ \frac{}{\Gamma \vdash \phi}$$

Satz 13. *C-H-I:*

$\vdash \phi \iff \phi$ *bewohnt*

Beweis 10.1. .

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

$\Longrightarrow$ Zu Γ definiert Kontext $\bar{\Gamma}$:

$\Gamma = \{\phi_1, \ldots, \phi_n\} \implies \bar{\Gamma} = \{x_1 : \phi_1, \ldots, x_n : \phi_n\}$

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

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

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

(Ax): $\phi \in \Gamma \implies \psi = \phi_i$ für ein $x_i : \phi_i \in \bar{\Gamma} \overset{(AX)}{\Longrightarrow} \bar{\Gamma} \vdash x_i : \phi_i$

$(\to_E)$

$$(\to_E)\ \frac{\Gamma \vdash \phi \to \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi}$$

I.V. haben t,s mit

$$\frac{\bar{\Gamma} \vdash t : \phi \to \psi \qquad \bar{\Gamma} \vdash s : \phi}{\bar{\Gamma} \vdash ts : \psi}$$

$(\to_I)$ Beweis

$$(\to_I)\ \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \psi \to \phi}$$

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

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

⚠**"→" ist intuitionistisch!**

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

$$(\to_I)\ \dfrac{(\to_E)\ \dfrac{(\text{Ax})\ \dfrac{}{(a \to a) \vdash (a \to a) \to a} \qquad \dfrac{(\text{Ax})\ \dfrac{}{(a \to a) \to, a \vdash a}}{(a \to a) \to a \vdash a \to a}}{(a \to a) \to a \vdash a}}{\vdash ((a \to a \to) a) \to a}$$

Auf Typebene:

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

Man hat eine funktion, die wieder eine funktion erhält also das innere $(a \to a) \to a$ und hat als Ergebnis wieder ein a. Dazu muss man anwenden. Die innerer funktion muss den typ $a \to a$ 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

$[\![Nat]\!] = \{0, Suc(0), Suc(Suc(0)), \ldots\}$ also ein Herbrandmodell.

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

- Menge M (Träger)
- zu f/n∈ Σ
- $\mathfrak{M}[\![f]\!] : M^n \to M$

Interpretion von Termen unter Umgebung $\eta : V \to M$:

$$\mathfrak{M}[\![t]\!]\eta \in M$$

$$\mathfrak{M}[\![x]\!]\eta = \eta(x)\ x \in V$$

$$\mathfrak{M}[\![f(t_1, \ldots, t_n)]\!]\eta = \mathfrak{M}[\![f]\!](\mathfrak{M}[\![t_1]\!]\eta, \ldots, \mathfrak{M}[\![t_n]\!]\eta)$$

Beispiel 11.1. $[\![Nat]\!]$ ist eine Σ_{Nat}-Algebra per:

$$[\![Nat]\!][\![0]\!] = 0$$

$$[\![Nat]\!][\![succ]\!](x) = Suc(x) \in [\![Nat]\!]$$

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

Σ−Homomorphismus = Abbildung $h : M \to N$ (wobei M,N trägermengen des respektiven Models sind)

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

$$h(\mathfrak{M}[\![f]\!](x_1, \ldots, x_n)) = \mathfrak{N}[\![f]\!](h(x_1), \ldots, h(x_n))$$

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. • $id_n : \mathfrak{M} \to \mathfrak{M}$ *ist* Σ*-Homomorphismus*

- $h : \mathfrak{M} \to \mathfrak{N}, k : \mathfrak{N} \to \mathfrak{P}$ *sind* Σ*-Homomorphismus* $\Longrightarrow$ $k \cdot h : \mathfrak{M} \to \mathfrak{P}$ *ist* Σ*-Homomorphismus*

Definition 11.3. h ist isomorphismus $\iff$ h Σ-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 $h : A \to B : x \leq y \implies h(x) \leq h(y)$

Aber das h^{-1} ist nicht monoton: $0 \leq 1, h^{-1}(0) \nleq h^{-1}(1)$

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

Erinnerung $m \equiv n(mod 4)$ 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. Σ_{Nat}-Algebra $\mathfrak{M}$ mit $M = \mathbb{Z}/4\mathbb{Z}, \mathfrak{M}[\![0]\!] = [0]_4, \mathfrak{M}[\![suc]\!]([n]_4) = [n+1]_4$

$h : [\![Nat]\!] \to \mathfrak{M}$

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

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

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

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

Definition 11.4. Σ–Algebra $\mathfrak{M}$ initial $\iff \forall \mathfrak{N}\, \exists!\, Homomorphismus\ h : \mathfrak{M} \to \mathfrak{N}$ (initiales Modell ist also algemeinstes Modell)

Satz 14. $[\![\Sigma]\!]$ *ist initial genannt "Termalgebra"*

Beweis 11.2. Sei $\mathfrak{N}$ eine Σ-Sigma Algebra. zZ: $\exists! h$ mit

$$(*)\ h([\![\Sigma]\!][\![f]\!](t_1,\dots,t_n)) = \mathfrak{N}[\![f]\!](h(t_1),\dots,h(t_n))\ (f/n \in \Sigma)$$

gilt für $h(t) = \mathfrak{N}[\![t]\!]$,

(*) ist rekursive Definition von $\mathfrak{N}[\![\ldots]\!]$

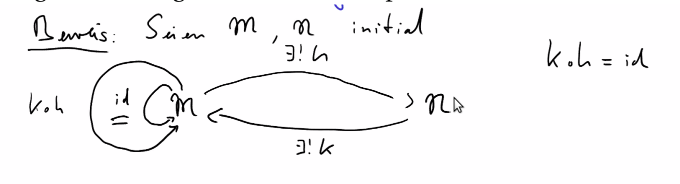
Satz 15. *Die initiale Σ-Algebra ist eindeutig bis auf* ***eindeutige*** *Isomorphie (also kategorietheorie…)*

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

$$\mathfrak{M}^{\exists! h \rightarrow}_{\leftarrow \exists! k}\mathfrak{N}$$

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 Σ-Algebraen.

11.1 Mengenkonstruktionen

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

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

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

Mehrer funktionen auf gleiche daten:

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

$< f_1, f_2 >: Y \to X_1 \times X_2$

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

kartesisches Produkt von Projektionen. (lifting)

$g_i : Y_i \to X_i$

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

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

Beispiele:

$\pi_i \cdot < f_1, f_2 >= f_i$

$< \pi_1, \pi_2 >= id, < \pi_1 \cdot f, \pi_2 \cdot f >= f$

$g_1 \times g_2 =< g_1 \cdot \pi_1, g_2 \cdot \pi_2 >$

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

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

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

$X_i \to X_1 + x_2$

$x \mapsto (i, x)$

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

kopaar.

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

$[f_1, f_2](i, x) \mapsto f_i(x)$ (man wählt also die i-te funktion aus. effektiv "case" statement)

zu $g_i : X_i \to Y_i \ (i = 1, 2)$

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

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

Lemma 11.3. *Duale:*

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

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

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

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. Bäume

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

Eine Σ_{Tree}-Algebra $\mathfrak{M}$:

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

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

$$\begin{tikzcd}
1 \arrow[r, "in_1"] \arrow[dr, "{\mathfrak{M}[\![Nil]\!]}"'] & 1 + M^2 \arrow[d, dotted, "{[\mathfrak{M}[\![Nil]\!], \mathfrak{M}[\![Node]\!]]}"] & M^2 \arrow[l, "in_2"'] \arrow[dl, "{\mathfrak{M}[\![Node]\!]}"] \\
& M &
\end{tikzcd}$$

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

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

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

$$\begin{tikzcd} M \times M \arrow[r, "f \times f"] \arrow[d, "\mathfrak{M}[\![Node]\!]"'] & N \times N \arrow[d, "\mathfrak{N}[\![Node]\!]"] \\ M \arrow[r, "f"'] & N \end{tikzcd}$$

$$\begin{tikzcd} 1 \arrow[r, "1"] \arrow[d, "\mathfrak{M}[\![Nil]\!]"'] & 1 \arrow[d, "\mathfrak{N}[\![Nil]\!]"] \\ M \arrow[r, "f"'] & N \end{tikzcd}$$

$$\begin{tikzcd} 1 + M \times M \arrow[r, "1+f \times f"] \arrow[d, "{[\mathfrak{M}[\![Nil]\!], \mathfrak{M}[\![Node]\!]]}"'] & 1 + N \times N \arrow[d, "{[\mathfrak{N}[\![Nil]\!], \mathfrak{M}[\![Node]\!]]}"] \\ M \arrow[r, "f"'] & N \end{tikzcd}$$

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

$$\begin{tikzcd} M \times M \arrow[r] \arrow[d, "in_2"'] \arrow[dd, bend right=60, "\mathfrak{M}[\![Node]\!]"'] \arrow[dr, phantom, "\#"] & N \times N \arrow[d, "in_2"] \arrow[dd, bend left=60, "\mathfrak{N}[\![Node]\!]"] \\ 1 + M \times M \arrow[r, "1 + h \times h"] \arrow[d, "\alpha"'] & 1 + N \times N \arrow[d, "\beta"] \\ M \arrow[r, "h"'] & N \end{tikzcd}$$

Da jedes Teildiagram kommutiert, gilt homomorphie.

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

Allgemein gilt für eine Σ-Algebra

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

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

$$\begin{tikzcd} \sum_{f/n \in \Sigma} M^n \arrow[r, "\sum_{f/n \in \Sigma} h^n"] \arrow[d, "\alpha"'] & \sum_{f/n \in \Sigma} N^n \arrow[d, "\beta"] \\ M \arrow[r, "f"'] & N \end{tikzcd}$$

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 signatursymbole!

11.2 Initialität und Rekursion

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

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

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

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

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= fold $(\lambda y \ldots y_n.a_f)_{f\in\Sigma}$ (sprich a_f ist eine totale funktion über f.)

z.B. Tree: $(foldcg) : Tree \to a$ mit $c : a, g : a \to a \to a$

```
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
```

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

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:

$h =< fact, id >$ 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:

$+ : Nat \times Nat \to Nat$

```
0+k=k
(suc n)+k = suc (n+k)
```

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

$+ = fold(\lambda k.k) \quad \underbrace{(\lambda f k.Suc(f\ k))}_{(Nat\to Nat)\to(Nat\to Nat)}$ primitive Rekursion:
Rekursion über Argumente verwende 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 $c : a_1 \times \cdots \times a_n \to b$ mit $n \geq 0$ $a_1, \ldots, a_n \in S$ und $b \in S \setminus S_0$ (sprich wir das ziel soll nicht wieder ein Parameter sein $Tree\ a \to Forest\ a$)

Definition 11.6. .

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

definier induktiv:

$$\frac{}{\Gamma \vdash x : a}\ (x : a \in \Gamma)$$
$$\frac{\Gamma \vdash t_1 : a_1, \ldots \Gamma \vdash t_n : a_n}{\Gamma \vdash c(t_1, \ldots, t_n) : b}\ (c : a_1 \times \cdots \times a_n \to b)$$

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

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

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

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

$\mathfrak{M}[\![c]\!] : \mathfrak{M}[\![a_1]\!] \times \cdots \times \mathfrak{M}[\![a_n]\!] \to \mathfrak{M}[\![b]\!]$

für jedes $c : a_1 \times \cdots \times a_n \to b$

Gegeben $\mathfrak{M}$ Umgebung η für Γ (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 <u>Σ-Homomorphismus</u> $h : \mathfrak{M} \to \mathfrak{N}$ über V besteht aus Abbildungen (ist also selbst mehrsortig)

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

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 $c : a_1 \times \cdots \times a_n \to b \in \Sigma$

<u>Termalgebra</u>

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

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

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

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

Satz 16. *Initialität/Fold-Prinzip*

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

$$h : [\![\Sigma]\!]_v \to \mathfrak{N}$$

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

$\underbrace{fold(Algebra):}_{mehrere\ Abbildungen!}\ [\![\Sigma]\!]_V \to \ldots$

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

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

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

$\mathfrak{N}[\![Nil]\!] = 0$

primitive Rekursion (gegenseitig) ⚠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)) $\overset{=}{IV}$ 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 (mirror 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 (mirrort t) = rev (flattent t)

Induktion über t:

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

Leaf geht, Node:

flattent (mirrort (Node f)) = flattent (Node (mirrorf f)) = flattenf (mirrorf f)

andere seite:

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

Induktionsbehauptung für f:Forest a

flattenf (mirrorf f) = rev (flattenf f)

dann gilt “rev (flattenf f) = flattenf (mirrorf f)” per I.V.

jetzt müssen wir die Induktionsvorraussetzung für Wälder anwenden: sprich dass flattenf (mirrorf f) = rev (flattenf f) für alle Wälder gilt

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

gilt also für Nil, jetzt Cons

flattenf (mirrorf (Cons t f)) = flattenf (concat (mirrorf f) (Cons (mirror t) Nil)) $\overset{=}{Lemma\ A}$ concat (flattenf (mirrorf f)) (flattenf (Cons (mirrort t) Nil)) $\overset{=}{Lemma\ B}$ concat $(\underbrace{flattenf(mirrorf\ f)}_{IVForest})$ $(\underbrace{flattent(mirrort\ t)}_{IVTree})$ = concat (rev (flattenf f)) (rev (flattent t))

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 $(\lambda xy.cc\ y[x])$

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. $A^\omega = \{a_0, a_1, a_2, \ldots | a_i \in A\}$ streams über A. (omega als Menge der ordinale, für uns einfach Nat)

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

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

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

```
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 $< hd, tl >: A^\omega \to A \times A^\omega$

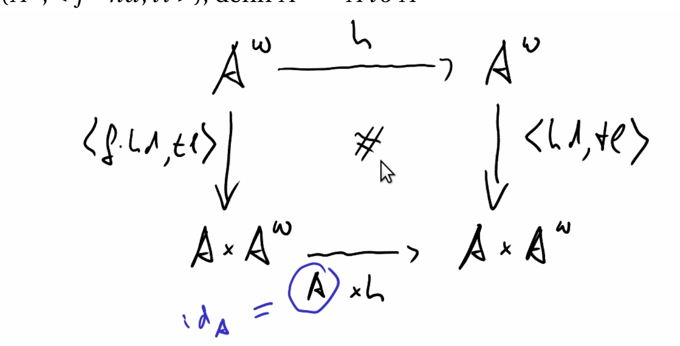
wir produzieren also aus A^ω eine ding aus der Trägermenge hinaus.(Destruction)

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

A^ω ist Koalgebra (genauer $(A^\omega, < hd, tl >)$ ist Koalgebra)

weiter Koalgebra:

$(A^\omega, < f \circ hd, tl >)$, denn $A^\omega \overset{hd}{\to} A \overset{f}{to} A$



explizit:

linke Komponente: hd(h(s)) = f(hd(s))

rechte Komponente: tl(h(s)) = tl(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: $GX \to GY$ $(f : X \to Y)$

Definition 11.8. Eine G-Koalgebra

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

Beispiel 11.5. .

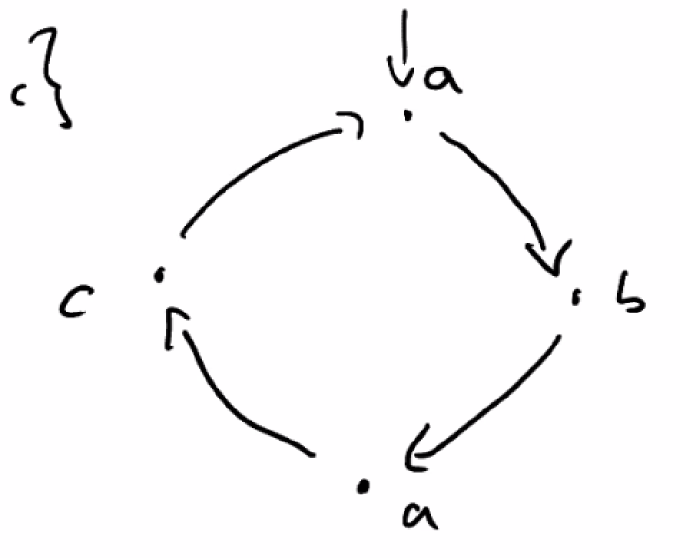
G=$A \times id$

G-Koalgebra = Koalgebra oben:

$M \to A \times M$

Eine Koalgebra besteht aus 2 sub-abbildungen: $M \to A$ ("outputabbildung") und $M \to M$ ("next-abbildung") das kreuzprodukt aus beiden ist die Hauptabbildung.

z.B. $A = \{a, b, c\}$

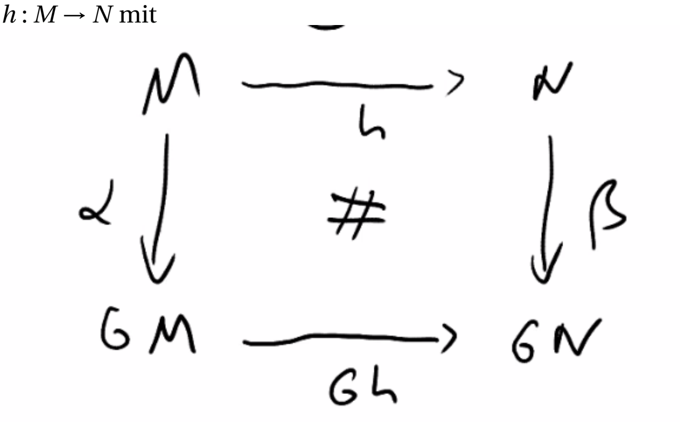


Stream abacab...

G-Koalgebraen sind Dual zu den Koalgebra (sie haben ein "unfold" statt ein "fold" prinzip)

Definition 11.9. Ein (G-Koalgebra-)Morphismus zwischen G-Koalgebren $\alpha : M \to GM$ $\beta : N \to GN$ ist eine Abbildung

$h : M \to N$ mit

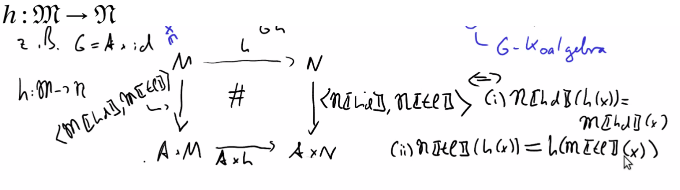


$\alpha : M \to GM = A \times M = < \underbrace{\mathfrak{M}}_{G-Koalgebra} [\![hd]\!], \mathfrak{M}[\![tl]\!] >$

Beispiel 11.6. .

z.B. $G = A \times id$

$h : \mathfrak{M} \to \mathfrak{N}$



Also das h bewahrt die heads.

h ist mit der tail-funktion verträglich.

Definition 11.10. Eine G-Koalgebra $\mathfrak{N}$ ist final $\iff$ für jede G-Koalgebra $\mathfrak{M}$ existiert genau ein Morphismus $h : \mathfrak{M} \to \mathfrak{N}$

Satz 17. *Die finale G-Koalgebra ist eindeutig modulo Isomorphismus (genauer: eindeutig bestimmter isomorphismus)*

Beweis 11.5. Dual zur Eindeutigkeit initialer Algebren (zwischen zwei initialen Algebren gibt es einen homomorphismus, wenn beide initial sind,dann gibt die verkettung der morphismen die Ursprüngliche Algebra)

Satz 18. *Existenz der finalen Algebra*

Sei $G = A \times id$. $< hd, tl >: A^{\omega} \to A \times A^{\omega}$ *ist finale G-Koalgebra.*

Beweis 11.6. (anhand eines Beispiels für einfachere Notation...)

Sei $\mathfrak{N}$ G-Koalgebra, schreibe $h_N = \mathfrak{N}[\![hd]\!] : N \to A$ $t_N = \mathfrak{N}[\![tl]\!] : N \to N$

Setze $f : N \to A^{\omega}$ $f(x) = [a_0, a_1, \ldots]$

$a_0 = h_N(x), a_1 = h_N(t_N(x))$ allg. $a_i = h_N(t_N^i(x))$

Beh.: f ist der einzige Morphismus $\mathfrak{N} \to A^{\omega}$

i) f ist ein Morphismus:

$hd(f(x)) \stackrel{!}{=} h_N(x)$

$= h_N(t_N^0(x))$

$tl(f(X)) = tl(a_0, a_1, \dots) = (a_1, \dots) =: (b_0, b_1, \dots)$

$b_i = a_{i+1} = \underbrace{h_N(t_N^{i+1}(x))}_{\text{“}beobachtungskontext''} = hn(t_N^i(\underline{t_N(x)}))$

also $(b_0, b_1, \dots) = f(t_N(x))$ d.h. f verträgt sich mit der Argumentation von tail

wir iterieren mit t_N und machen eine Observation mit h_N ii) Eindeutigkeit von f: Sei $g : \mathfrak{N} \to A^\omega$ Morphismus

zZ $f = g$.

Zeige $f(x)_i = g(x)_i$ wobei der index "an der stelle i" heißt. $\forall x \in N$

durch Induktion über $i \in \mathbb{N}$:

$i = 0 : g(x)_0 = hd(g(x)) \overset{morphismus}{=} h_N(x) = f(x)$

$i \to i+1$: $g(x)_{i+1} \overset{indexversch}{=} (tl(g(x)))_i \overset{morphismus}{=} g(t_N(x))_i \overset{IV}{=} f(t_N(x))_i = \dots = f(x)_{i+1}$ (Weil die Aussage für alle x gilt, und $t_N(x)$ ein x aus N ist)

Definition 11.11. $h : N \to A, t : N \to N$

definieren G-Koalgebra $\mathfrak{N}$, der eindeutige Morphismus $\mathfrak{N} \to A^\omega$ heißt

$$unfold\ h\ t$$

unfold h t ist definiert durch 2 Gleichungen:

```
hd(unfold h t x) = h x
tl(unfold h t x) = unfold h t (t x)
```

Korekursive Definition

Man schreibt also für jeden Konstruktor eine funktion, die im korekursiven Fall beliebig oft eine Funktion auf den gegebenes Element x ausführt.

Beispiel 11.7. ones = 1,1,...

```
ones = 1:ones

hd ones = 1
tl ones = 1
```

blink x $(a_0, a_1, \dots) = (x, a_0, x, a_1, \dots)$

```
hd (blink x s) = x
tl (blink x s) = blink x ...???
```

allgemeiner: blink: $2 \times A \times A^\omega \to A^\omega$ wobei $2 = \{0, 1\}$

Die 2 ist also ein "merker", bzw Extra Zustände in der Zustandsmaschine

blink 0 x (a0,a1,...) = (x,a0,x,a1,...)

blink 1 x (a0,a1,...) = (a0,x,a1,...)

```
blink 0 = b0
blink 1 = b1
hd (b0 x s) = x
tl (b0 x s) = b1 x s
hd (b1 x s) = hd s
tl (b1 x s) = b0 x (tl s)
```

Das tail in b1 ist wichtig, weil wir das a0 auf a1 weiterschieben müssen

11.7 Koinduktion

$even: A^\omega \to A^\omega$

even $(a_0, a_1, \ldots) = (a_0, a_2, a_4, \ldots)$

odd $(a_0, a_1, \ldots) = (a_1, a_3, a_5, \ldots)$

```
hd (even s) = hd s
tl (even s) =even $tl $tl s
hd (odd s) = hd $ tl s
tl (odd s) = odd $tl $tl $ s
```

Behauptung: odd $(b_0\ x\ s) = s$

hd (odd $(b_0\ x\ s)$) = hd (tl $(b_0\ x\ s)$) = hd (b_1 x s) = hd s

tl (odd $(b_0\ x\ s)$) = odd (tl tl $(b_0\ x\ s)$) = odd (tl (b_1 x s)) = odd (b_0 x (tl s)) $\overset{IV??????}{=}$ tl s

Problem: wir reduzieren hier "tl s→ s" aber hier wird ja nichts "abgebaut" die Streams werden also nicht kürzer!!!

KOINDUKTION!! ("Circular Solver")

Definition 11.12. Eine Relation $R \subseteq A^\omega \times A^\omega$ heißt Bisimulation, wenn für alle $(s, t) \in R$ gilt

- hd s = hd t
- (tl s) R (tl l)

Satz 19. *(Koinduktion). Wenn R eine Bisimulation ist, dann gilt R $\subseteq id$, d.h*

$$\forall s,t \in A^\omega (sRt) \implies s = t$$

Beweis 11.7. Sei $tl^n(s)Rtl^n(t)$. Per Induktion über n folgt n=0 Annahme

$n \to n+1$ per IV.

$$\underbrace{tl(tl^n(s))}_{tl^{n+1}(s)} Rtl(tl^n(t))$$

Es gilt also

$$tl^i(s)Rtl^i(t) \text{ für alle } i \geq 0$$

und damit

$$hd(tl^i(s)) = hd(tl^i(t)) \text{ für alle } i \geq 0$$

Damit folgt s=t

Die Koinduktion ist also eigentlich nur eine Induktion über Destruktorkontexte!

Beispiel 11.8. cont von oben

Setze R={$(odd(b_0\ x\ s), s) | \forall s \in A^\omega$}

Zeige: R ist Bisimulation.

Sei also $s \in A^\omega$

i) hd($odd(b_0\ x\ s)$) = hd(s). Oben schon gezeigt.

ii) tl($odd(b_0\ x\ s)$)R(tl s)

linke Seite = $odd(b_0\ x(tl\ s))$, damit haben wir die Relation ✓

Beispiel 11.9. Zip-Funktion

zip: $A^\omega \times A^\omega \to A^\omega$

zip $(x_0, x_1, \dots)$ $(y_0, y_1, \dots) = (x_0, y_0, x_1, y_1)$.

hd(zip s t) = hd s.

tl(zip s t)= zip t (tl s)

(wir drehen also t und s um und entfernen den ersten Wert)

Beh.: zip (even s) (odd s) = s.

Setze R={$(zip(even\ s)(odd\ s), s) | \forall s \in A^\omega$}

zZ.: R ist Bisimulation. Sei also $s \in A^\omega$.

i) hd (zip (even s) (odd s)) = hd (even s) = hd s ✓ wir haben die Relation

ii) tl (zip (even s) (odd s))= zip (odd s) (tl (even s)) = zip (odd s) (even (tl (tl (s)))) vs. tl s??????????

also geht R nicht (R muss abgeschlossen sein, was es hier aber nicht ist) R' = $R \cup \{(zip(odd\ s)(even\ tl^2\ s), tl\ s) | \forall s \in A^\omega\}$

R' ist Bisimulation:

Paare in R schon bewiesen.

neue Paare in R':

i) hd(zip (odd s) (...)) = hd (odd s) = hd (tl s)✓

ii) tl(zip (odd s) (even tl^2 s)) = zip (even (tl^2 s)) $\underbrace{(tl(odd s))}_{odd(tl^2\ s)}$ R'(tl(tl s))✓.

Entspricht also verstärkung des Induktionszieles.

11.8 kodatentypen mit mehreren Nachfolgeoperationen

```
codata Tree where
 out:   Tree -> A
 left:  Tree -> Tree
 right: Tree -> Tree
```

definiert die finale G-Koalgebra für $G = A \times id \times id$

Diese besteht aus unendlichen binären Bäumen mit in A beschrifteten Knoten.

$mirror : Tree \to Tree$ Korekursiv definiert durch

```
out (mirror t) = out t
left (mirror t) = mirror (right t)
right (mirror t) = mirror (left t)
```

$R \subseteq Tree \times Tree$ ist Bisimulation $\iff \forall (s,t) \in R$

i) out s = out t

ii) (right s) R (right t)

iii) (left s) R (left t)

(analog wie bei Streams, nur mit 2 "tail" funktionen)

Beispiel 11.10. Beh.: mirror (mirror t) = t.

Setze R = $\{(mirror(mirror\ t), t) | \forall t \in Tree\}$ (eig. Semantikklammern um Tree, wird hier "wegvereinfacht")

Zeige: R ist eine Bisimulation.

Sei also $t \in Tree$

i) out (mirror (mirror t)) = out(mirror t) = out t.✓

ii) left(mirror(mirror t)) = mirror (right(mirror t)) = mirror (mirror (left t)) R(left t)

weil oben in R als beliebiges t auch (left t) gewählt werden kann.

iii) analog

right(mirror(mirror t)) = mirror (left(mirror t)) = mirror (mirror(right t)) R (right t)

Nebenbemerkung: Es gibt auch Abbrechende Streams von Typ $1 + A \times X$ (gleiche System von Listen). Hier gibt es 2 optionen: Wir kriegen einen neuen Stream bei tail, den head, oder ein Ende. (vgl Skript 4.8)

12 Polymorphie

polymorphe Funktionen: anwendbar auf Argumente verschiedenen Typs.

ad-hoc-Polymorphie: +

parametrische Polymorphie

swap (x,y) = (y,x)

swap:$a \times b \to b \times a$

d.h. Das genau gleich stückchen code bekommt verschiedene typen (im gegensatz zu ad-hoc, wo im einen fall int, im anderen float addition ausgewählt wird).

concat: List a→ List a→ List a.

ist auch parametrisch polymorph.

12.1 System F

silly::Bool

silly=$(\lambda xy.x)$(id True)(id 42)

Im einfach getypten Lambda Kalkül kann man das nicht typen: $id : Bool \to Bool$ und gleichzeitig $id : Int \to Int$!

Typen in System F:

$$\alpha, \beta ::= a | \alpha \to \beta | \forall a.\alpha \quad (a \in V)$$

z.B.: $id : \lambda x.x : \forall a.a \to a$

Typregeln: $\lambda \to$ plus

$$(\forall_i)\ \frac{a \notin FV(\Gamma) \qquad \Gamma \vdash s : \alpha}{\Gamma \vdash s : \forall a.\alpha}$$

$$(\forall_e)\ \frac{\Gamma \vdash s : \forall a.\alpha}{\Gamma \vdash s : (\alpha[\beta/a])}$$

Beispiel 12.1. .

$$\forall i\ \cfrac{(\to_i)\ \cfrac{(\text{Ax})\ \cfrac{}{x : a \vdash x : a}}{\vdash \lambda x.x : a \to a}}{\vdash \lambda x.x : \forall a.a \to a}$$

NICHT $x : a | x : \forall a.a$, das a ist im Kontext

12.2 Church-Kodierung in System F

$\mathbb{N}$ ist initiale Σ_{Nat}-Algebra $\Sigma_{Nat} = \{0/0, s/1\}$

- $\mathbb{N} := \forall a.(a \to a) \to a \to a$
- $0 : \mathbb{N}$

  $0 = \lambda f x.x$ hat den typ von oben

- $s : \mathbb{N} \to \mathbb{N}$

  $s = \underbrace{\lambda n}_{wir\ kriegen\ ein\ \mathbb{N}} .\lambda\ f\ x.f(n\ f\ x)$

  Haskell mit GHC: Higher-Rank-polymorphism
- fold:$\forall a.(a \to a) \to a \to \mathbb{N} \to a$

  fold = $\lambda f\ x\ n.n\ f\ x$ also n mal f auf x anwenden.

Paare:

- $(a \times b) := \forall r\, \underbrace{(a \to b \to r)}_{f} \to r$ (sprich ich kann ein paar aus ab zu einem r machen, effektiv uncurry)
- pair:$\forall ab : a \to b \to (a \times b)$

  pair = $\lambda x y.\lambda f.f\ x\ y$
- fst:$\forall ab.(a{\times}b) \to a$ fst = $\lambda p.p \underbrace{(\lambda\ x\ y.x)}_{das\ ist\ das\ zugeordnete\ f}$ wir haben also eine pick-funktion, die das erste paarelement gibt, das zweite ignoriert

Summen:

- (a+b):$\forall r.\underbrace{(a \to r)}_{f} \to \underbrace{(b \to r)}_{g} \to r$
- inl:$\forall ab.a \to (a+b)$

  inl = $\lambda x.\lambda f\ g.f\ x$

- case: $\forall abs.(a \to s) \to (b \to s) \to (a+b) \to s$

  case = $\lambda\ f\ g\ z.z\ f\ g$

Listen:

- List a:$\forall r.r \to (a \to r \to r) \to r$

  Nil: $\forall a.List\ a$

  Nil = $\lambda u\ f.u$

  Cons: $\forall a.a \to List\ a \to List\ a$

  Cons = $\lambda u\ f.\lambda v\ g.g\ u(f\ v\ g)$

Herleitung der Typisierung von suc

$$(\text{AX})\ \dfrac{\dfrac{\text{ist typ von } \mathbb{N}}{\Gamma \vdash \forall a.(a \to a) \to a \to a}}{\Gamma \vdash n : (a \to a) \to a \to a}\ (\forall_e)$$

$$\to_e \dfrac{\text{für f und x typen aus kontext}\ \dfrac{\vdots}{\Gamma \vdash n\ f\ x : a} \qquad (\text{Ax})\ \dfrac{}{\Gamma \vdash f : a \to a}}{\underbrace{n : \mathbb{N}; f : a \to a; x : a}_{\Gamma} \vdash f\ (n\ f\ x) : a}$$

$$2\times \to_i \dfrac{}{n : \mathbb{N} \vdash \lambda\ f\ x.\ f\ (n\ f\ x) : (a \to a) \to a \to a}$$

$$\forall_i \dfrac{}{(\to_i)\ \dfrac{n : \mathbb{N} \vdash \lambda\ f\ x.\ f\ (n\ f\ x) : \mathbb{N}}{\vdash \lambda\ n\ f\ x.\ f\ (n\ f\ x) : \mathbb{N} \to \mathbb{N}}}\ \text{zum auflösen von } \mathbb{N}$$

Satz 20. *Wells 1994: Typcheck in System F ist untentscheidbar!*

Jede arithmetik 2. Stufe definierbar totale berechenbare Funktion ist in System F typisierbar!! (Reicht um die Reellen Zahlen zu typisieren!)

Satz 21. *(Normalisierend, Girard) System F ist SN, wenn $\Gamma \vdash s : \alpha$ in $\lambda 2$ dann ist s stark normalisierend (also System F nicht Turingmächtig)*

Trotzdem in System F ist definierbar

- alle primitiv rek. Funktionen
- Ackermann
- Compiler, aber nicht Interpreter

12.3 ML-Polymorphie

Einschränkungen:

1) $\forall$ nur top-level.

2) Mehrfachinstanziierung nur per let!

$(\lambda f.ff)\underbrace{(\lambda x.x)}_{id}$

ist let $\underbrace{f = \lambda x.x}_{\forall a.a \to a}$ in f f.

Typen

$$\alpha, \beta ::= a | \alpha \to \beta$$

Typschemata gebe durch die nicht rekursive Grammatik

$$S ::= \forall a_1, \dots, a_k.\alpha \text{ mit } (k \geq 0)$$

Terme

$$t, s ::= x | ts | \lambda x.t | let\ x = s\ in\ t$$

in System F sind variablen, die nicht verwendet werden implizit alquantifiziert.

Wir definieren eine Closure

$$Cl(\Gamma, \alpha) = \forall a_1, \dots, a_k.\alpha \text{ mit } \{a_1, \dots, a_k\} = FV(\alpha) \setminus FV(\Gamma)$$

Typschemata in Γ. Also alle typen sind Tyschemata, aber nicht alle Typschemata sind typen!

Die Tyschemata kommen nur im Kontext vor, man kann keine Funktion selbst polymorph machen, nur sagen, dass eine Poylmorphes "ding" im kontext steht.

Typregeln $(\to_i), (\to_e)$ gleich

$\forall_e \dfrac{}{\Gamma \vdash x : \alpha[\beta_1/a_1, \dots, \beta_k/a_k]} (x : \forall a_1, \dots, a_k.\alpha \in \Gamma)$

Effektiv wird hier also $\forall_e$ immer kombiniert mit einer Ax Regel ausgeführt. Dies liegt daran, dass wir Termen kein Typschema, sondern nur Typen zuweisen können)

$$\text{let}\ \frac{\Gamma[x \mapsto Cl(\Gamma,\alpha)] \vdash t:\beta \qquad \Gamma \vdash s:\alpha}{\Gamma \vdash \text{let x=s in t}:\beta}\ (x:\forall a_1,\dots,a_k.\alpha \in \Gamma)$$

Dies ersetzt die $\forall_i$ wir setzen also alle noch nicht vom Kontext festgelegten werte frei im Let. Wir lassen also keine funktionen mehr von polymorphen funktionen abhängen, stattdessen benutzen wir das let und passen das Typschema an.

Kontexte bestehen aus $\Gamma = \{x_1 : S_1,\dots,x_n : S_n\}$ also aus Typschemata.

Beispiel 12.2. .

$$\text{(let)}\ \dfrac{\rightarrow_e \dfrac{(\forall_e)\ \dfrac{\text{z.B. } \gamma = a \to a}{id:\forall a.a \to a \vdash id:\gamma} \qquad (\forall_e)\ \dfrac{\text{z.B. } \beta = a \to a}{id:\forall a.a \to a \vdash id:\gamma \to \beta}}{id:\forall a.a \to a \vdash id\ id:\beta} \qquad (\text{herleitung } \lambda \to)\ \dfrac{\dots}{\vdash \lambda x.x}\ \text{implizite allquantifizierung von a exp}}{\vdash \text{let id} = \lambda x.x \text{ in id id}:\beta}$$

Wenn wir den Hindley-Millner algorithmus generalisieren wollen, brauchen wir die Typinversionsregeln. Diese gelten aber in System F nicht (das macht genau die Entscheidbarkeit kaputt)

Satz 22. *Inversionslemma: Applikation,λ wie für $\lambda \to$*

- Variablen: $\Gamma \vdash x:\alpha \iff \Gamma(x) = \forall a_1,\dots,a_k.\beta$

α hat die Form $\alpha = \beta[\gamma_1/\alpha_1,\dots,\gamma_k/\alpha_k]$

- let: $\Gamma \vdash let\ x = s\ in\ t$*:* $\beta \iff \Gamma \vdash s:\alpha, \Gamma[x \mapsto Cl(\Gamma,\alpha)] \vdash t:\beta$

Recall: $PT(\Gamma,t,\alpha)$ Gleichunssystem

<u>Invariante</u>: σ löst $\Gamma \vdash t:\alpha \iff \sigma$ erweitert zu Unifikator von $PT(\Gamma,t,\alpha)$

Definition 12.1. von $PT(\Gamma,t,\alpha)$ per Rekursion über t.

Klauseln für Applikation, λ wie in $\lambda \to$

$PT(\Gamma,x,\alpha) = \{\beta[a_1'/a_1,\dots a_k'/a_k] \doteq \alpha\}$ für $\Gamma(x) = \forall a_1,\dots,a_k.\beta$ sprich wir müssen verlangen, dass die allquantifizierten variablen von β frisch sind

$PT(\Gamma, let\ x = s\ in\ t;\alpha) = PT(\Gamma\sigma[x \mapsto Cl(\Gamma\sigma,\sigma(a))];t;\alpha\sigma)$

σ =mgu($PT(\Gamma,s,a)$ (a frisch)) wir machen also Typinferenz für s und substituieren passend.

Beispiel 12.3. $PT((); let\ id\ = \lambda x.x\ in\ id\ id; a)$

zuerst das let auflösen, dafür PT von $\lambda x.x$ bestimmen

$mgu(PT((); \lambda x.x; b)); \sigma(b) = c \to c$

jetzt das Closure einsetzen für let

$PT(\underbrace{id:\forall c.c \to c}_{\Gamma}; id\ id; a)$

$= PT(\Gamma; id; d \to a) \cup PT(\Gamma, id; d)$

$= \{c \to c \doteq d \to a\} \cup \{ \underbrace{e \to e}_{global\ frisch} \doteq d\}$

der mgu ist also $a \doteq e \to e$

Beweis 12.1. Korrektheit Invariante

Variablen: wenig neues (wir machen jetzt nur ein instanziierung, die vom unifikationsalgorithmus sowieso gemacht wird)

Definition 12.2. Für $S = \forall a_1, \ldots, a_k.\alpha$

Inst(S) =$\{\alpha[\beta_1 / a_1, \ldots, \beta_k / a_k | \beta_1, \ldots, \beta_k$ Typen$\}$

S allgemeiner als $S' \iff$ Inst(S')$\subset Inst(S)$

Lemma 12.1. *Für $S = \forall a_1, \ldots, a_k.\alpha, S' = \forall b_1, \ldots, b_m.\beta$ sind äquivalent:*

i) S allgemeiner als S'

ii) $\beta \in Inst(S)$ UND $b_1 \ldots b_m \notin FV(\alpha)$

Das Liefert u.U. aber ein Problem $S = \forall a.a \to b$ $S' = \forall b.b \to b$ hier ist S nicht allgemeiner als S', obwohl man einfach a=b wählen kann

Beweis 12.2. (nur ii nach i, andere Richtung geht, brauchen wir aber nicht)

Sei $\beta = \alpha[\gamma_1 / a_1, \ldots, \gamma_k / a_k]$, $\sigma = [\delta_1 / b_1, \ldots, \delta_m / b_m]$ (d.h. $\beta\sigma \in$ Inst(S'))

zZ $\beta\sigma \in Inst(S)$

Nach Annahme

$\beta\sigma = \alpha[\gamma_1\sigma / a_1, \ldots \gamma_k\sigma / a_k] \in Inst(S)$

Damit folgt

Lemma 12.2. *$CL(\Gamma, \alpha)\sigma$ allgemeiner als $CL(\Gamma\sigma, \alpha\sigma)$*

Beweis 12.3. i) Klar ✓

ii): Quantifizierte Variablen rechts: $FV(\alpha\sigma) \setminus FV(\Gamma\sigma)$

d.h. $b \in FV(CL(\Gamma, \alpha)\sigma) \implies b \in FV(\sigma(a))$ für ein $a \in FV(CL(\Gamma, \alpha)) \subseteq FV(\Gamma)$

$\implies b \in FV(\Gamma\sigma)$,

also b rechts nicht quantifiziert.

Lemma 12.3. *Abschwächung:*

S allgemeiner als S', $\Gamma, x : S' \vdash t : \alpha$

$\implies \Gamma, x : S \vdash t : a\alpha$, *Klar.*

Invariante für let:

(nur $\implies$ weil $\impliedby$ direkt folgt, weil wir mehr unifizieren, als wir brauchen, also auch insbesondere unser let)

Sei $\Gamma\sigma \vdash let\ x = s\ in\ t : \alpha\sigma$.

Per Inversionslemma es existiert β mit $\Gamma\sigma \vdash s : \beta, \Gamma\sigma, x : CL(\Gamma\sigma, \beta) \vdash t : \alpha\sigma$

$\overset{IV}{\implies}$ σ erweiterbar zu $\sigma' \in Unif(PT(\Gamma\sigma; s; b))$ mit $\sigma'(b) = \beta$

$\tau := mgu(PT(\Gamma\sigma; s; b))$; GLOIN: o.E.$\tau\sigma' = \sigma'$

zZ: σ' erweitert zu Unifikator von $PT(\Gamma\tau, x : CL(\Gamma\tau, \tau(b)), t, \alpha\tau)$

I.V.: reicht $\Gamma\tau\sigma', x : CL(\Gamma\tau, \tau(b))\sigma' \vdash t : \alpha\tau\sigma'$

$\iff \Gamma\sigma, x : CL(\Gamma\tau, \tau(b))\sigma' \vdash t : \alpha\sigma$

$\iff \Gamma\sigma, x : CL(\Gamma\sigma, \sigma(b)) \vdash t : \alpha\sigma$

Man kann das σ' reinziehen (das geht per lemma, weil wir sogar weniger allgemein auf der rechten seite werden)

13 Automatentheorie

(Gerade Rechtzeitig. . .)

13.1 Sprachen als Kodaten

Recall: DFA A = $(Q, \Sigma, \delta, s, F)$

- Q (endliche) Menge von Zuständen
- Σ Alphabet (endlich)
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to Q$
- Anfangszustand $s \in Q$
- Menge $F \subseteq Q$ von akzeptierenden/finalen Zuständen

Definition 13.1. $\delta(q, w)$ für $w \in \Sigma^*$ induktiv

$$\delta(q, \epsilon) = q$$

$$\delta(q, wa) = \delta(\delta(q, w), a)$$

(von hinten erweitern ist einfacher, wir schalten Buchstabe für Buchstabe durch das Wort)

Von A akzeptierte Sprache

$$L(A) = \{w \in \Sigma^* | \delta(s, w) \in F\}$$

(Praxisrelevanz: Programme sind meistens automaten (u.u. endlich), d.h. es wurden schon pinautomaten geknackt, indem man automatisch den automatenuntersucht hat)

Schreibe $\delta : Q \to (\Sigma \to Q)$, $F : Q \to 2$

wir haben also $< F, \delta >: Q \to \underbrace{2 \times \overbrace{(\Sigma \to Q)}^{\cong Q \times \cdots \times Q, |\Sigma|\ mal}}_{GQ}$ wir haben also eine (rekursive) G-Koalgebra.

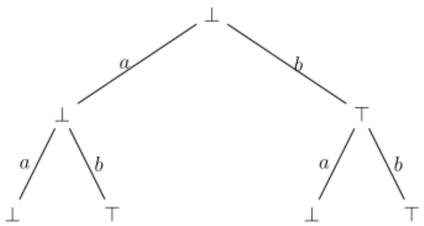
bzw. Es fehlt noch der intialzustand, wir nennen diese Vorstufe eine (deterministisches) Transitionssystem A_0

```
codata Lang Sigma where
    acc : Lang Sigma -> Bool
    der : Lang Sigma -> (Sigma -> Lang Sigma)
    -- accept/derivation
```

der sind eigentlich viele nachfolgeoperationen: Eine pro Σ.

finale Koalgebra: - unendliche Σ-verzweigenden Bäumen

- Knoten in 2 gelabled (also $T, \perp$)



Es ändert sich eigentlich nur, welche Lables in jedem schritt gesetzt werden (links bleibt a, rechts b)

Allgemein kann somit der Baum mit Worten indiziert werden. Der Baum hier entspricht also einer Teilmenge aus Σ^* ist also eine Sprache.

$$F(L)\epsilon = T \iff \epsilon \in L$$

$$\delta\ L\ a = \{u \in \Sigma | au \in L\} = L_a$$

wobei L_a Ableitung von L nach a ist. (wir schalten mit δ durch den Ableitungsbaum)

$\Lambda = P(\Sigma^*)$ bezeichnet sowohl die Menge aller Sprachen, als auch die finale G-Koalgebra

G-Koalgebramorphismen:

$$\begin{tikzcd}
Q_A \arrow[r, "f"] \arrow[d, "\langle F_A, \delta_A \rangle"'] \arrow[dr, phantom, "\#"] & Q_B \arrow[d, "\langle F_B, \delta_B \rangle"] \\
2 \times (\Sigma \to Q_A) \arrow[r, "2 \times (\Sigma \to f)"'] & 2 \times (\Sigma \to Q_B)
\end{tikzcd}$$

Sei A, B automaten und f eine funktion von $Q_A \to Q_B$

(b,g) $\mapsto (b, f \circ g)$

links: $F_B(f(q)) = F_A(q)$ d.h. $f(q)$ final $\iff$ q final.

Also wenn wir im Urbild fertig sind, dann muss auch das Bild fertig sein.

rechts: $\delta_B(f(q)(a)) = f(\delta_A(q)(a))$ wir können also eine Transition in Q_A machen, und den fortschritt in Q_B bringen.

Satz 23. *Sei $A_0 = (Q, \Sigma, \delta_A, F_A)$ Transitionssysteme.*

Der eindeutige G-Koalgebramorphismus $f : A_0 \to \Lambda$ ist

$$f : Q \to \Lambda$$

$$q \mapsto L(A_0, q) = \{w | F(\delta(q)(w)) = T\}$$

(in anderen worten: f bildet von automaten auf die akzeptierende Sprache ab)

Beweis 13.1. Reicht: f ist Morphismus

Finalität wird bewahrt: q final $\iff \epsilon \in L(A_0, q) \iff L(A_0, q)$ final in Λ

Transitionen: zZ $\overbrace{f(\delta_A(q)(a))}^{L(A_0,\delta_A(q)(a)}) = \delta_\Lambda(f(q))(a) = L(A_0, q)a$

$u \in L(A_0, \delta_A(q)(a)) \iff F_A(\underbrace{\delta_A(\delta_A(q)(a))(u)}_{=\delta_A(q)(au)}) = T \iff au \in L(A_0, q) \iff u \in L(A_0, q)_a$

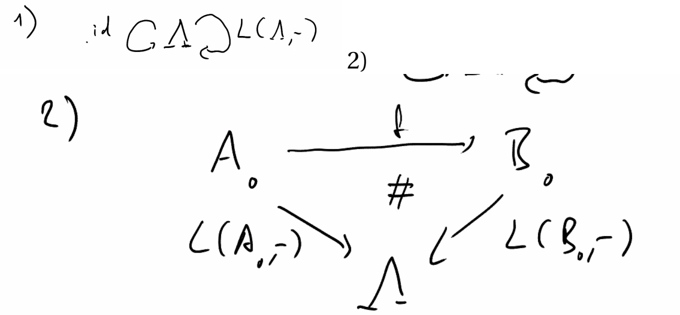
Korrolar 13.1. *1) $L \in \Lambda \implies L(\Lambda, L) = L$*

(Λ ist ein Automat, L eine Sprache in diesem (undendlichen Automat))

2) $f : A_0 \to B_0$ Morphismus von Transitionssystemen $\implies L(A_0, q) = L(B_0, f(q))$

Beweis 13.2. .

1)



13.2 Minimierung

Wir haben 2 grobe Fälle: Zustände können equivalent sein, oder Nutzlos (unerreichbar)

Definition 13.2. Sei A_0 Transitionssystem $q \in Q_0$

$$< q >= \{\delta(q)(w) | w \in \Sigma^*\}$$

Das sind alle von q aus erreichbaren Zustände. Wir nennen dies das von q erzeugte Untertransitionssystem.

Es ist Tatsächlich ein Untertransitionssystem, weil $q' \in < q >$ final $\iff q'$ final in A_0

$\underbrace{\delta_{<q>}(q')(a)}_{\delta(q)(w)} = \underbrace{\delta_{A_0}(q')(a)}_{\delta(q)(wa)\in<\delta>}$
somit unter Transition abgeschlossen.

Lemma 13.1. *Die Einbettung $< q >\hookrightarrow A_0$ Morphismus, sodass die Behauptungen aus dem Korrolar 13.1 (die Eigentliche Einbettung ist ein Noop von $q \to q$, zeigt nur eine Teilmengeneigenschaft auf der Algebra)*

Lemma 13.2. .

$$f : A_0 \to B_0$$

Morphismus $\implies$

$$f :< q >\to< f(q) >$$

Morphismus, surjektiv

(aus surjektiv folgt insbesondere $| < f(q) > | \le | < q > |$)

Beweis 13.3. .

$\{\delta_{A_0}(q)(w) | w \in \Sigma^*\}$ $\{\delta_{B_0}(q)(w) | w \in \Sigma^*\}$

f kommutiert $f(\delta_{A_0}(q)(w)) = \delta_{B_0}(f(q))(w)$

Damit sind beide Behauptungen gezeigt.

Lemma 13.3. *$L(< q >, q) = L(A_0, q)$ folgt aus 13.1*

z.B. $f = L(A_0, -)$ ist ein Morphismus:

$$| < L(A_0, q) > | \le | < q > | \le |A_0|$$

d.h. $L(A) = L \implies | < L > | \le |A| \implies L(\Lambda, L) = L(< L >, L) = L\ (< L >, L)$ ist minimaler Automat für L. (besteht aus allen von m aus erreichbaren Sprachen im Transitionssystem)

Satz 24. *L regulär $\iff$ $< L >$ endlich.*

Recall: R Äquivalent auf X $\implies X/R = \{[x]_R | x \in X\} = \{y | xRy\}$

$|x/R|$ = Index von R.

$f : X \to Y$

$Ker(f) = \{(x, y) | f(x) = f(y)\}$ ist eine Äquivalenz (Das ist eine generalisierung von Ker aus linAlg $f(x) = f(y) \iff f(x) - f(y) = 0 \iff \underbrace{f(x-y)}_{linear} = 0 \iff x - y \in Ker(f)$)

$$h : X/(Ker(f)) \to f[X] = \{f(x) | x \in X\}$$

$$[x]_{Kerf} \mapsto f(x) \text{ bijektiv}$$

h wohldefiniert: x(Kerf)y $\implies f(y) = f(x)$ DAS HIER IST WOHLDEFINIERT (die Abbildung ist von representanten unabhängig, NICHTS anderes)

also index von Kerf = $|f[x]|$

$$< L >= \{L_u | u \in \Sigma^*\} = d[\Sigma^*],$$

mit $d(u) = L_u \implies < L >$ endlich $\iff \Sigma^*/(Kerd)$ endlich.

$\sim_L := Kerd$

$$v \sim_L w \iff L_v = L_w \iff \forall u \in \Sigma^* (u \in L_v \iff u \in L_w)$$

$$\forall u \in \Sigma^* (vu \in L \iff wu \in L)$$

Satz 25. *von Myhill/Nerode: L regulär $\iff \sim_L$ hat endlichen Index.*

Bisimulation auf Λ: $R \subseteq \Lambda \times \Lambda$ Bisimulation $\iff$ L R K $\implies$

1) $\sigma \in L \iff \sigma \in K$ (also $F(L) = T \iff F(k) = T$)

2) $\forall a \in \Sigma(\underbrace{\delta(L)(a)}_{L_a} R \underbrace{\delta(K)(a)}_{K_a})$

allgemeiner: $A_0 = (Q, \Sigma, \delta, F)$ Transitionssystem

$R \subseteq Q \times Q$ Bisimulation $\iff \{(L(A_0, q), L(A_0, r)) | (q, r) \in R\}$ Bisimulation auf Λ (sprich, wenn die Sprache von q und die Sprache von r über A in Relation stehen)

d.h. $(q, r) \in R \implies$

1) $F(q) = F(r)$ 2) $\forall a \in \Sigma(\delta(q)(a) R \delta(r)(a))$

q,r <u>bisimilar</u> (ist die größte Bisimulation zwischen q und r) $\iff \exists R$ Bisimulation (qRr)

q,r bisimlar $\implies L(A_0, q) = L(A_0, r)$

ALGORITHMUS (Minimiere DFA A=$(Q, \Sigma, \delta, s, F)$) var $R \subseteq Q \times Q$

- 1. Entferne nicht erreichbare Zustände aus Q (Tiefensuche, merke gefunde Zustände, sobald ein Zustand doppelt, kann dieser Teilbaum abgebrochen werden)
- 2. Initialisiere $R = \{(q, r) \in Q \times Q | q \in F \iff r \in F\}$ (R hat erstmal 2 Äquivalenzklassen: final/nichtfinal, wir nähern die größte bisimulation von oben her an; bisimilar 1 gilt)
- 3. Pick $(q, r) \in R, a \in \Sigma$ mit

  $$(\delta(q)(a), \delta(r)(a)) \notin R$$

  if not found goto 4 else $R := R \setminus \overbrace{\{(r, q), (q, r)\}}^{symmetrie}$ goto 3
- 4. $Q := Q/R, f := F/R, \delta([q]_R)(a) = [\delta(q)(a)]_R$ wir reduzieren also den automaten auf die Äquivalenzklassen

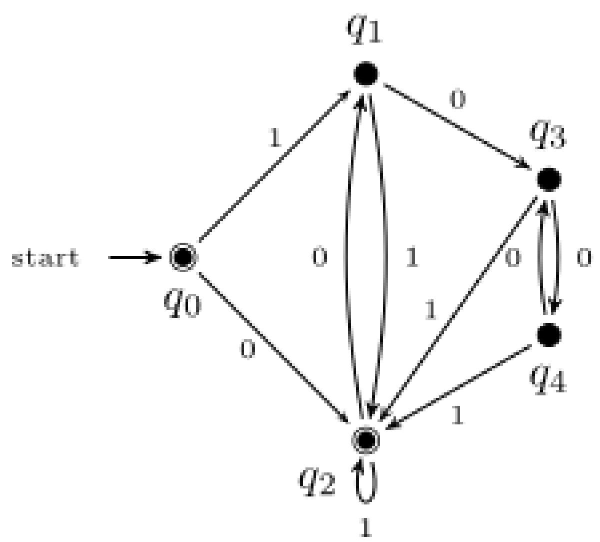
Satz 26. *Nach Schritt 4 ist A minimal (nach Quotientierung durch R)*

Beweis 13.4. Reich q, r bisimilar (im Input) $\iff qRr$ in Schritt 4.

“ $\impliedby$ ” per 2,3 ist R am ende eine Bisimulation. (in 2. werden alle finalen und nichtfinalen getrennt, in 3. werden alle nicht äquivalenten zustände getrennt)

“ $\implies$ ” Entfernte Paare sind nicht bisimilar (Invariante des Algorithmus)

Beispiel 13.1. .



q_1	0			
q_2	1	0		
q_3	0		0	
q_4	0		0	
	q_0	q_1	q_2	q_3

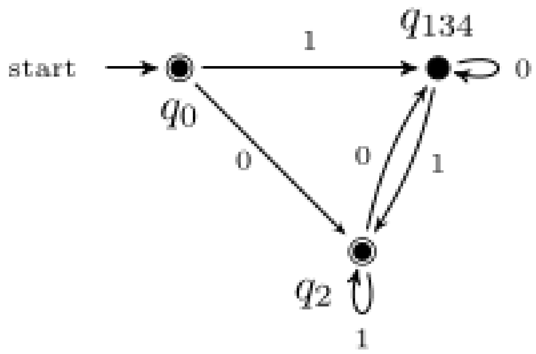
Dreieckig, weil symmetrisch, auch die diagonale kann weggelassen werden (Äq-Relation ist trivial Reflexiv) Die zahl ist die Optimierungsrunde, in der die Zustände rausfliegen.

Runde Null lässt Zustände mit unterschiedlicher finalität rausfliegen.

Erste runde z.B. q_2 hat 1 nachfolger q_2 und q_0 hat 1 nachfolger q_1. q_1 und q_2 sind nicht in relation (per schritt 0) fliegt raus.

aber z.B. q_3, q_4 haben beide als 1 nachfolger q_2 und 0 nachfolger q_3 und q_4, die beide in Relation stehen (wir checken ja gerade das, deshalb müssen sie noch drin sein)

Die leeren stellen sind die in relation stehenden zustände



die Übergänge können einfach durch einen der Nachfolger bestimmt werden (welcher ist egal, die sind ja äquivalent)

14 Bonus: SN von System F

Idee: zu Typ α definiere

$$[\![\alpha]\!]_\xi \subseteq SN = \{t \in \Lambda | t\ SN\}$$

mit $\xi : V \to SAT$

$$SAT = \{A \subseteq SN | A\ saturiert\}$$

$$A \to B = \{t \in \Lambda | \forall s \in A.\ ts \in B\}$$

mit $A, B \subseteq \Lambda$

Rekursiv 3 Fälle:

$$[\![\alpha]\!]_\xi = \xi(\alpha)$$

$$[\![\alpha \to \beta]\!]_\xi = [\![\alpha]\!] \to [\![\beta]\!]$$

$$[\![\forall a.\alpha]\!]_\xi = \bigcap_{A \in SAT} [\![\alpha]\!]_{\xi[a \mapsto A]}$$

(der letzte punkt ist ein Durchschnittstyp: das macht das ding untentscheidbar)

Lemma 14.1. *Substitutionslemma (noch eins)*

$$[\![\alpha[\beta/a]]\!]_\xi = [\![\alpha]\!]_{\xi[a \mapsto [\![\beta]\!]_\xi]}$$

Beweis: Induktion über α

Definition 14.1. $A \subseteq SN$ saturiert $\iff$

i) $xt_1 \dots t_n \in A$ $(x \in V; t_1, \dots, t_n \in SN; n \geq 0)$

ii) $t[s/x]u_1 \dots u_n \in A \implies$

$(\lambda x.t)su_1 \dots u_n \in A (s, \in SN)$ (wir müssen explizit s SN annehmen, weil x u.U. nicht in t existiert: es viele also in der prämisse weg. Die konklusion ist dann aber abhängig von reduktionsreihenfolge, also nicht mehr SN!)

Lemma 14.2. *Saturiertheitslemma*

a) $SN \in SAT$

b) $[\![\alpha]\!]_\xi \in SAT$

Beweis 14.1. a) 1. Sie $x \in V, t_1, \dots, t_n \in SN$

zZ.: $xt_1 \dots t_n \in SN \checkmark$

alle Reduktionen passieren in den $t's$ also insgesamt SN

2) Sei $s \in SN, t[s/x]u_1 \dots u_n \in SN$

zZ: $(\lambda x.t)su_1 \dots u_n \in SN$

Widerspruchsbeweis: Annahme es gibt eine unendliche Reduktionssequenz:

$(\lambda x.t)su_1 \dots u_n \to_\beta^* (\lambda x.t')s'u'_1 \dots u'_n$ wir reduzieren also alles was geht $(t, u_1 \dots u_n, s \in SN)$

$\to_\beta t'[s'/x]u'_1 \dots u'_n \to_\beta^* \dots$

Das führt jedoch zum widerspruch:

$t[s/x]u_1 \dots u_n$ reduziert in endlich vielen schritten auf die Form $t'[s'/x]u'_1 \dots u'_n$ also gäbe es eine unendliche Reduktionssequenz für $t[s/x]u_1 \dots u_n$

b)

Induktion über α

$\alpha = a\checkmark$

$\rightarrow$ IV $A = [\![\alpha]\!]_\xi$ $B = [\![\beta]\!]_\xi$ $A, B \in SAT$

zZ: $[\![\alpha \rightarrow \beta]\!]_\xi = A \rightarrow B \in SAT$

dafür zZ:

0) $A \rightarrow B \subseteq SN$: Sei $t \in A \rightarrow B$ habe $x \in A$ (wir haben so ein x, weil A saturiert und dort insbesondere alle Variablen sind) $\implies tx \in B \subseteq SN \implies t \in SN$

1) Sei $x \in V, r_1, \dots, r_n \in SN$

zZ $xr_1 \dots r_n \in A \rightarrow B$; sei also $s \in A$

zZ $xr_1 \dots r_n s \in B\checkmark$

2) Sei $s \in SN, r[s/x]r_1 \dots r_n \in A \rightarrow B$

zZ $(\lambda x.t)sr_1 \dots r_n \in A \rightarrow B$

Sei also $v \in A$, zZ $(\lambda x.t)sr_1 \dots r_n v \in B \impliedby t[s/x]r_1 \dots r_n v \in B \impliedby t[s/x]r_1 \dots r_n \in A \rightarrow B, v \in A\checkmark$

3. $\forall$ SAT ist offenbar abgeschlossen unter Durchschnitten nichtleerer Mengenfamilien. Aus a) folgt, dass im Durchschnitt mindestens ein index A, nämlich A = SN vorkommt

Definition 14.2. Erfülltheit/Konsequenz.

$\sigma, \xi \models t : \alpha$

Literatur

[1] Knuth-bendix algorithm for creating CR TES from terminating TES
https://en.wikipedia.org/wiki/Knuth%E2%80%93Bendix_completion_algorithm

[2] dependent choice
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