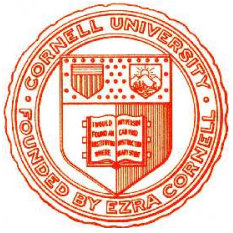
Automatisierte Logik und Programmierung

Einheit 6



Die einfache Typentheorie

1. Syntax und Semantik
2. Typechecking
3. Formales Schließen in der Typentheorie
4. Die Curry-Howard Isomorphie
5. Eigenschaften typisierbarer λ-Terme

Logisches Schliessen über Datentypen

Koppelung von Term- und Typstruktur

- **Datentypen sind Eigenschaften von Termen**
  - `1,2,3`,... sind natürliche Zahlen
  - `x+4, y-z, 4*i`,... sind Ausdrücke über Zahlen
  - `(4,5), (7,8), (3-2,4*6)`,... sind Paare von Zahlen
  - `[1;4;7;8;5]` ist eine Liste von Zahlen
  - λ`x. 4*x` ist eine Funktion auf Zahlen
- **Datentypen besitzen Struktur**
  - $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$, $\mathbb{B}$, String ... sind einfach strukturierte Typen
  - Datentypen können zusammengesetzt werden
    - λ`x. 4*x` besitzt den Typ $\mathbb{N}\to\mathbb{N}$
    - `(4,5)` ist vom Typ $\mathbb{N}\times\mathbb{N}$
    - `[1;4;7;8;5]` gehört zum Typ $\mathbb{N}$ `list`
  - Datentypen können Programmeigenschaften charakterisieren
    - $\{$`f:`$\mathbb{N}\to\mathbb{N}$ `|` $\forall$`x:`$\mathbb{N}$`. f(x)`$^2\leq$`x<(f(x)+1)`$^2\}$ spezifiziert λ`x.` $\lfloor\sqrt{\texttt{x}}\rfloor$

Datentypkalkül: was ist neu?

Logisches Schließen über Programmeigenschaften

- **Mehr Struktur in Beschreibung von Typen**
  - Präzisierung der prädikatenlogischen Bereiche
  - Typausdrücke ersetzen Bereichssymbole
  - Interpretation von Typausdrücken i.w. festgelegt
- **Regeln zur Typisierung von Termen**
  - Führt zu Einschränkung auf 'interpretierbare' Ausdrücke
    - λ-Ausdrücke müssen (sinnvolle) Funktionen darstellen
    - Keine unkontrollierte Rekursion oder Selbstanwendung
  - Minimale Begrenzung der Ausdruckskraft

$\Downarrow$

Präzisierung der Semantik formaler Ausdrücke

Typdisziplin: Beschreibungsformen

- **Getypter λ-Kalkül**
  - Typen sind Teil der Syntax von λ-Termen: $\lambda \mathtt{f}^{\mathtt{S}\to\mathtt{T}}.\lambda \mathtt{x}^{\mathtt{S}}.\,\mathtt{f}^{\mathtt{S}\to\mathtt{T}}\,\mathtt{x}^{\mathtt{S}}$
  - Datentyp aller Teilausdrücke leicht zu ermitteln
  - Untypisierbare Terme sind nicht formulierbar
- **Typentheorie**
  - Terme und Typen sind unabhängige Konzepte
  - Typen werden Termen zugeordnet: $\lambda \mathtt{f}.\lambda \mathtt{x}.\,\mathtt{f}\,\mathtt{x} \in (\mathtt{S}\to\mathtt{T})\to\mathtt{S}\to\mathtt{T}$

    Typisierung ist losgelöst von der Formulierung des Terms
  - Mehr Freiheit in der Formulierung von λ-Termen
  - Größerer Aufwand bei (nachträglicher) Bestimmung des Typs

$\Downarrow$

Kalkül zum Schließen über Typzugehörigkeit

Charakteristika eines Datentyps

Syntaktische Beschreibung von Mengen

- **Direkte Beschreibung einfacher Datentypen**
  - Festlegung eines Namens $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{B}$ ...
  - Beschreibung der Basiselemente `0,1,2,3`, ..., `0,1,-1,2`, ..., `tt`,`ff`
  - Beschreibung von Operatoren auf Elementen `-`x, x`+`y, x`-`y, x`*`y, ...
- **Konstruktoren für strukturierte Datentypen**
  - Typkonstruktoren: $T \times T'$, $T \to T'$, T `list`, ...
  - Konstruktion kanonischer Elemente: `(`a`,`b`)`, $\lambda x\,.\,t$, a_1`::`l, ...
  - Nichtkanonische Operatoren auf Elementen: p`.1`, f`(`a`)`, hd`(`l`)`, ...
- **Grundbestandteile der beschreibenden Syntax**
  - Typ-Ausdruck + kanonische Terme + nichtkanonische Terme

Einfache Typentheorie – Syntax

Minimal nötige Typkonstrukte für den λ-Kalkül

- **Alphabete für erlaubte Symbole**
  - $\mathcal{V}$: Variablensymbole
  - $\mathcal{T}$: Bereichssymbole (Typen)
- **Typ-Ausdrücke (Typen)**
  - Typsymbole $T \in \mathcal{T}$
  - $S \to T$ und (T), wobei S und T Typen
- **Objekt-Ausdrücke (λ-Terme)**
  - Variablen $x \in \mathcal{V}$
  - $\lambda x\,.\,t$, $f\ t$ und (t), wobei $x \in \mathcal{V}$ und t und f λ-Terme
- **Prioritäten**
  - Applikation bindet stärker als λ-Abstraktion
  - Applikation ist links-assoziativ: $f\ t_1\ t_2 \mathrel{\hat{=}} (f\ t_1)\ t_2$
  - $\to$ ist rechtsassoziativ: $T_1 \to T_2 \to T_3 \mathrel{\hat{=}} T_1 \to (T_2 \to T_3)$

Einfache Typentheorie – Semantik

- **Operationale Semantik für λ-Terme**
  - Bestimmung des Wertes durch Reduktion
- **Zuordnung von Typen zu λ-Termen**
  - $t \in T \equiv \iota(t)$ ist ein Element von $\iota(T)$
- **Interpretiere $\rightarrow$ als Funktionenraum**
  - $\iota(S \rightarrow T) \mathrel{\hat{=}}$ Menge der Funktionen von $\iota(S)$ nach $\iota(T)$
- **Typzugehörigkeit: Koppele Typ- und Termaufbau**
  - $\iota(x)$ kann zu $\iota(T)$ gehören, wenn x Variable und T Bereichssymbol
  - $\iota(\lambda x.t)$ gehört zu $\iota(S \rightarrow T)$, wenn $\iota(t)$ zu $\iota(T)$ gehört falls $\iota(x)$ in $\iota(S)$
  - $\iota(f\,t)$ gehört zu $\iota(T)$, wenn $\iota(t)$ zu einem $\iota(S)$ und $\iota(f)$ zu $\iota(S \rightarrow T)$ gehört

Typisierung von $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$ für beliebige S und T

$$\underbrace{\underbrace{(\mathtt{S} \rightarrow \mathtt{T}) \rightarrow (\mathtt{S} \rightarrow \mathtt{T})}_{\mathtt{T}}}_{\mathtt{S} \rightarrow \mathtt{T}}$$

Typzugehörigkeit: Vorbetrachtungen

- **Der Typ eines λ-Terms ist nicht eindeutig**
  - Naheliegend: $\lambda \mathtt{f}.\lambda \mathtt{x}.\, \mathtt{f}\,\mathtt{x} \in (\mathtt{S}\to\mathtt{T}) \to (\mathtt{S}\to\mathtt{T})$
  - Auch möglich $\lambda \mathtt{f}.\lambda \mathtt{x}.\, \mathtt{f}\,\mathtt{x} \in (\mathtt{S}\to(\mathtt{T}\to\mathtt{S})) \to (\mathtt{S}\to(\mathtt{T}\to\mathtt{S}))$

    $\lambda \mathtt{f}.\lambda \mathtt{x}.\, \mathtt{f}\,\mathtt{x} \in (\mathtt{S}\to\mathtt{S}) \to (\mathtt{S}\to\mathtt{S})$
  - $(\mathtt{S}\to\mathtt{T})\to(\mathtt{S}\to\mathtt{T})$ ist die allgemeinste Form (prinzipielles Typschema)
- **Nicht jeder λ-Term kann typisierbar sein**
  - Sonst gäbe es eine einfache Semantik des λ-Kalküls
  - Beispiel: $\lambda \mathtt{x}.\, \underbrace{\mathtt{x}}_{\mathtt{S}\to\mathtt{T}}\ \underbrace{\mathtt{x}}_{\mathtt{S}}$ für beliebige $\mathtt{S}$ und $\mathtt{T}$
  - Die Gleichung $\mathtt{S} = \mathtt{S}\to\mathtt{T}$ hat keine "natürliche" Lösung

Typzugehörigkeit und Berechnung

- **Reduktion erhält Typzugehörigkeit**
  - $\iota((\lambda x.u)\,t)$ in $\iota(T)$ impliziert $\iota(t)$ in $\iota(S)$ und $\iota(\lambda x.u)$ in $\iota(S{\to}T)$
  - $\iota(\lambda x.u)$ in $\iota(S){\to}\iota(T)$ verlangt $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$
  - $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$ und $\iota(t)$ in $\iota(S)$ impliziert $\iota(u[t/x])$ in $\iota(T)$
- **Die Umkehrung gilt nicht**
  - Typisierbarkeit von $u[t/x]$ impliziert nicht Typisierbarkeit von $(\lambda x.u)\,t$

    da x eventuell in u nicht frei vorkommt
  - Beispiel: $(\lambda \mathtt{f}.\lambda \mathtt{y}.\mathtt{y})(\lambda \mathtt{x}.\,\mathtt{x}\,\mathtt{x})$ nicht typisierbar,

    da $(\lambda \mathtt{x}.\,\mathtt{x}\,\mathtt{x})$ nicht typisierbar

    aber $\lambda \mathtt{y}.\mathtt{y} \in \mathtt{T}{\to}\mathtt{T}$

Semantik der Typentheorie präzisiert

Urteile ersetzen modelltheoretische Semantik

- **Definiere Gleichheit von λ-Termen wie zuvor**
  - Reduktions- und Konversionsregeln definieren, wann $t \xrightarrow{*} u$ gilt
- **Definiere Typzugehörigkeitsrelation $t \in T$**
  - Urteile definieren, wann diese Relation erfüllt ist
    - $x \in T$ falls $x \in \mathcal{V}$ und T ein Typ
    - $\lambda x\,.\,t \in S \to T$ falls $t \in T$ aus $x \in S$ folgt
    - $f\,t \in T$ falls $f \in S \to T$ und $t \in S$ für einen Typ S
    - $(t) \in T$ falls $t \in T$
    - $t \in (T)$ falls $t \in T$
- **Definiere Typisierbarkeit eines Terms t**
  - t typisierbar, falls $t \in T$ für einen Typ T
- **Definiere prinzipielles Typschema eines Terms t**
  - "Kleinster" (allgemeinster) Typ T, für den $t \in T$ gilt

Typzugehörigkeit harmoniert mit Berechnung

Sind t und t' semantisch gleich, so gilt $t \in T \Leftrightarrow t' \in T$

1. **Reduktion erhält den Typ: aus $t \xrightarrow{\beta} t'$ folgt $t \in T \Leftrightarrow t' \in T$**
   - Induktion über die Termstruktur von t
     - $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓
     - $t \equiv \lambda x\,.\,u$: Dann $t' \equiv \lambda x\,.\,u'$ mit $u \xrightarrow{\beta} u'$.
       Aus $t \in T$ folgt $T \equiv S_1 \to S_2$ mit $u \in S_2$ wenn $x \in S_1$.
       Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$
       Gleiches Argument für $t \in T$ wenn $t' \in T$ ✓
     - $t \equiv (\lambda x\,.\,u)\ v$, $t' \equiv u[v/x]$: Aus $t \in T$ folgt $\lambda x\,.\,u \in S \to T$ und $v \in S$, also $u \in T$, wenn $x \in S$. Damit $u[v/x] \in T$.
       Aus $t' \in T$ folgt $v \in S$ und $u \in T$ wenn $x \in S$. Damit $(\lambda x\,.\,u)\ v \in T$. ✓
     - $t \equiv f\ u$: Dann $t' \equiv f\ u'$ und $u \xrightarrow{\beta} u'$ oder $t' \equiv f'\ u$ und $f \xrightarrow{\beta} f'$.
       Gleiches Argument wie zuvor. ✓

   Achtung: t und t' müssen typisierbar sein
2. **Aus $t \xrightarrow{*} t'$ folgt $t \in T \Leftrightarrow t' \in T$** (Induktion über Reduktionslänge)
3. **t und t' sind semantisch gleich, wenn $t \xrightarrow{*} u$ und $t' \xrightarrow{*} u$**

Typ-Inferenz und -überprüfung

- **Bestimme den Typ eines beliebigen Terms**
  - Vollautomatisch für einfache Typstrukturen (ML, Haskell, Java, ...)
  - Unentscheidbar für komplexe Datentypen (Logische Spezifikationen)
- **Analyse von $t \in T$**
  - Weise t einen unbekannten Typ T zu
  - Verfeinere T anhand der Regeln der Typzugehörigkeit
  - Ergibt prinzipielles Typschema oder Nachweis der Nichttypisierbarkeit
- **Typisierung von $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$**

$$\lambda \underbrace{\mathtt{f}}_{X_1}.\lambda \underbrace{\mathtt{x}}_{X_3}.\ \underbrace{\mathtt{f}\ \mathtt{x}}_{X_1\ X_3} \in (X_3 \to X_4) \to (X_3 \to X_4)$$

- **Typisierung von** $\lambda \underbrace{\mathtt{x}}_{X_1}.\ \underbrace{\mathtt{x}}_{X_1}\ \underbrace{\mathtt{x}}_{X_1} \in X_1 \to X_2 \qquad X_1 = X_1 \to X_2$

  nicht typisierbar

Hindley-Milner Type-Checking Algorithmus

- **Eingabe**: geschlossener λ-Term t

  Ausgabe: prinzipielles Typschema von t oder Fehlermeldung

- **Start**: Setze globale Substitution σ`:=[]` und rufe `TYPE-OF([],`t`)` auf

- **Algorithmus** `TYPE-OF(`Env`,`t`)`:

  Falls $t{=}x \in \mathcal{V}$: suche Deklaration $x{:}T$ in Env und gebe T aus

  Falls $t{=}f\ u$: Setze $S_1{:=}$`TYPE-OF(`Env`,`f`)`, $S_2{:=}$`TYPE-OF(`Env`,`u`)`

  Wähle neues $\mathtt{X}_{i+1}$ und unifiziere $\sigma(S_1)$ mit $S_2{\rightarrow}\mathtt{X}_{i+1}$.

  Fehlermeldung, wenn Unifikation fehlschlägt.

  Sonst $\sigma := \sigma'{\circ}\sigma$, wobei σ' Ergebnis der Unifikation

  Ausgabe: $\sigma(\mathrm{X}_{i+1})$

  Falls $t{=}\lambda x\,.\,u$: Wähle neues $\mathtt{X}_{i+1}$

$$S_1{:=}\begin{cases} \texttt{TYPE-OF(}Env{\cdot}\texttt{[}x\texttt{:X}_{i+1}\texttt{],}u\texttt{)} & \text{falls } x \text{ nicht in } Env \\ \texttt{TYPE-OF(}Env{\cdot}\texttt{[}x'\texttt{:X}_{i+1}\texttt{],}u[x'/x]\texttt{)} & \text{sonst } (x' \in \mathcal{V} \text{ neu}) \end{cases}$$

  Ausgabe: $\sigma(\mathrm{X}_{i+1}){\rightarrow}S_1$

Typisierung von $\lambda f.\lambda x.\, f(f(f\,x))$

Env	*Aktueller Term t*	*Substitution σ*	UNIFY	*Typ T*
	$\lambda f.\lambda x.\, f(f(f\,x))$			
$f:X_0$	$\lambda x.\, f(f(f\,x))$			
$f:X_0,\ x:X_1$	$f(f(f\,x))$			
$f:X_0,\ x:X_1$	f			X_0
$f:X_0,\ x:X_1$	$f(f\,x)$			
$f:X_0,\ x:X_1$	f			X_0
$f:X_0,\ x:X_1$	$f\ x$			
$f:X_0,\ x:X_1$	f			X_0
$f:X_0,\ x:X_1$	x			X_1
$f:X_0,\ x:X_1$	$f\ x$		$X_0 = X_1 \to X_2$	X_2
$f:X_0,\ x:X_1$	$f(f\,x)$	$[X_1 \to X_2 / X_0]$	$X_1 \to X_2 = X_2 \to X_3$	X_3
$f:X_0,\ x:X_1$	$f(f(f\,x))$	$[X_3, X_3, X_3 \to X_3 / X_2, X_1, X_0]$	$X_3 \to X_3 = X_3 \to X_4$	X_4
$f:X_0$	$\lambda x.\, f(f(f\,x))$	$[X_4, X_4, X_4, X_4 \to X_4 / X_3, X_2, X_1, X_0]$		$X_4 \to X_4$
	$\lambda f.\lambda x.\, f(f(f\,x))$	$[X_4, X_4, X_4, X_4 \to X_4 / X_3, X_2, X_1, X_0]$		$(X_4 \to X_4) \to X_4 \to X_4$

Beweiskalkül für die Typentheorie

- **Umsetzung der Urteile in Inferenzregeln**
  - Urteile sind schematische Regeln zur Definition von Semantik
  - Inferenzregeln sind syntaktische Vorschriften für Beweisführung
  - Ähnlichkeit erleichtert Nachweis von Korrektheit und Vollständigkeit
- **Schließen über Gleichheit und Berechnung**
  - Regeln zur Auswertung von Termen wie im λ-Kalkül
- **Schließen über Typzugehörigkeit**
  - Dekompositionsregeln zur Zuordnung von Typen zu Termen
  - Benötigt Nachweis, daß manche Ausdrücke Datentypen darstellen
- **Schließen über Datentypen**
  - Regeln zur strukturellen Analyse von (Typ-)Ausdrücken

Typzugehörigkeit: Welche Regeln sind nötig?

- $f\,t \in T$ *falls* $f \in S \to T$ *und* $t \in S$ *für einen Typ* S
  - Direkte Umsetzung möglich $\Gamma \vdash f\,t \in T$ `BY applyI` S
    $\Gamma \vdash f \in S \to T$
    $\Gamma \vdash t \in S$
  - Typ S muß als Parameter der Regel angegeben werden
- $\lambda x\,.\,t \in S \to T$ *falls* $t \in T$ *aus* $x \in S$ *folgt*
  - Umsetzung erfordert Typdeklaration für $x \in S$ in der Teilzielsequenz
    $\Gamma \vdash \lambda x\,.\,t \in S \to T$ `BY lambdaI`
    $\Gamma, x' : S \vdash t[x'/x] \in T$
    $\Gamma \vdash S$ `Type`
  - S muß ein Typ sein, wenn es rechts in einer Deklaration auftaucht
  - Umbenennung der Variablen x ist eventuell notwendig
- $x \in T$ *falls* $x \in \mathcal{V}$ *und* T *ein Typ ist*
  - Im formalen Beweis muß die Zuordnung von T zu x deklariert sein
    $\Gamma, x : T, \Delta \vdash x \in T$ `BY declaration`

Integration von Typzugehörigkeit und Gleichheit

- **Gleichheit hängt eigentlich vom Typ ab**
  - 4 und 7 sind verschieden als Zahlen aber gleich in $\mathbb{Z}_3$
  - $(2,4)$ und $(1,2)$ sind verschieden als Zahlenpaare aber gleich in $\mathbb{Q}$
  - In der einfachen Typentheorie haben semantisch gleiche Terme allerdings immer denselben Typ
- **Kombiniere $s{=}t$ und $t \in T$ zu $s{=}t \in T$**
  - Prädikat mit fester Bedeutung: "s und t sind im Typ T gleich"
  - $s{=}t \in T$ bedingt neben der Gleichheit auch $s \in T$ und $t \in T$
  - Typzugehörigkeit $t \in T$ kann als Kurzform für $t{=}t \in T$ betrachtet werden
  - Die Regeln `applyI` und `lambdaI` werden als spezielle Form von entsprechenden Gleichheitsregeln `applyEq` und `lambdaEq` betrachtet
- **Allgemeine Reflexivitätsregel wird ungültig**
  - $t{=}t \in T$ gilt nur, wenn auch $t \in T$ gezeigt werden kann

Notwendige Erweiterungen formaler Konzepte

- **Neues festes Symbol $\mathbb{U}$ (Universum aller Typen)**
- **Deklaration von Variablen und Typen möglich**
  - $x : S$ ($x \in \mathcal{V}$, Variablendeklaration), oder $T : \mathbb{U}$ ($T \in \mathcal{T}$, Typdeklaration)
  - Konzept der Deklaration in Sequenzen bekommt erstmalig einen Sinn
- **Konklusion kann Typisierungen enthalten**
  - Typisierte Gleichheit: $s{=}t \in T$ (oder Kurzform $t \in T$)
  - Typeigenschaft: $T \in \mathbb{U}$ ("T ist ein Typ")
- **Sequenz: $H_1, \ldots, H_n \vdash C$**
  - H_i Formel oder Deklaration, C Formel (evtl. mit Typisierungen)
  - $H_1, \ldots, H_n$ ist rein, wenn jede Variable genau einmal deklariert ist und jede Typvariable einer Variablendeklarationen zuvor deklariert wurde
  - $\Gamma \vdash t \in T$ ist Initialsequenz für Typisierungsbeweise, wenn Γ rein ist und genau die in T vorkommenden Variablen deklariert

Andere Begriffe (Inferenzregel, Beweis, Theorem, ...) wie zuvor

Verfeinere Beweisregeln des λ-Kalküls

applyEq S

$$\frac{\Gamma \vdash f\,t = g\,u \in T \qquad \Gamma \vdash f = g \in S \to T \qquad \Gamma \vdash t = u \in S}{}$$

lambdaEq *

$$\frac{\Gamma \vdash \lambda x.\,t = \lambda y.\,u \in S \to T \qquad \Gamma, x' : S \vdash t[x'/x] = u[x'/y] \in T \qquad \Gamma \vdash S \in \mathbb{U}}{}$$

reduction

$$\frac{\Gamma \vdash (\lambda x.\,u)\,s = t \in T \qquad \Gamma \vdash u[s/x] = t \in T}{}$$

declaration i

$$\Gamma, x : T, \Delta \vdash x \in T \quad \hat{=} \quad x = x \in T$$

functionI

$$\frac{\Gamma \vdash S \to T \in \mathbb{U} \qquad \Gamma \vdash S \in \mathbb{U} \qquad \Gamma \vdash T \in \mathbb{U}}{}$$

*: *Die Umbenennung $[x'/x]$ kann entfallen, wenn x nicht frei in Γ vorkommt*

Abgeleitete Inferenzregeln

Regeln für Typzugehörigkeit sind Spezialfall

`applyI` S

$$\frac{\Gamma \vdash f\,t \in T}{\Gamma \vdash f \in S \to T \qquad \Gamma \vdash t \in S}$$

`lambdaI`

$$\frac{\Gamma \vdash \lambda x\,.\,t \in S \to T}{\Gamma, x' : S \vdash t[x'/x] \in T \qquad \Gamma \vdash S \in \mathbb{U}}$$

`reduction`

$$\frac{\Gamma \vdash (\lambda x\,.\,u)\,s = t \in T}{\Gamma \vdash u[s/x] = t \in T}$$

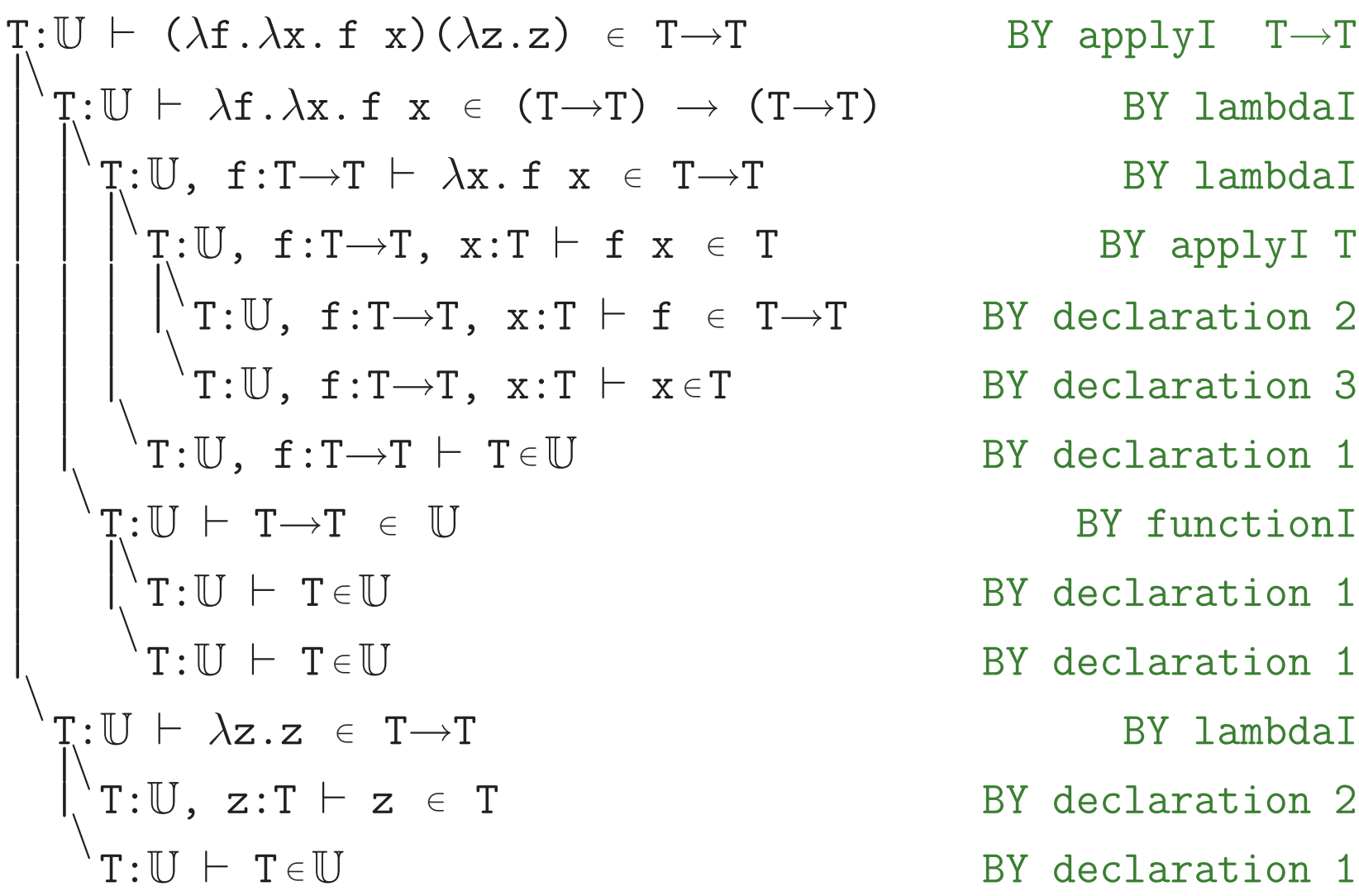
`declaration` i

$$\Gamma, x : T, \Delta \vdash x \in T$$

`functionI`

$$\frac{\Gamma \vdash S \to T \in \mathbb{U}}{\Gamma \vdash S \in \mathbb{U} \qquad \Gamma \vdash T \in \mathbb{U}}$$

Sequenzenbeweis für $(\lambda f.\lambda x.\, f\ x)(\lambda z.z) \in T \to T$



Logikregeln werden redundant

$\Gamma \vdash \lambda x.t \in S \to T$ `BY lambdaI`
$\Gamma, x' : S \vdash t[x'/x] \in T$
$\Gamma \vdash S \in \mathbb{U}$

$\Gamma \vdash A \Rightarrow B$ `BY impI`
$\Gamma, A \vdash B$

$\Gamma \vdash f\,t \in T$ `BY applyI` S
$\Gamma \vdash f \in S \to T$
$\Gamma \vdash t \in S$

$\Gamma \vdash B$ `BY modus ponens` A
$\Gamma \vdash A \Rightarrow B$
$\Gamma \vdash A$

Curry-Howard Isomorphie

Typ	Formel
Variable vom Typ S	Annahme der Formel S
Term vom Typ T (freie Variablen aus S_i)	Beweis von T (Annahmen S_i)
Typechecking	Beweisführung
(Regel der) λ-Abstraktion	$\Rightarrow$-Einführung
(Regel der) Applikation	$\Rightarrow$-Elimination

Logik als Spezialfall der Typentheorie ?

Eigenschaften typisierbarer Terme

- **Schwache Normalisierbarkeit**
  - Hat jeder Term eine Normalform?
  - Führt mindestens eine Reduktionsstrategie immer zu einem Wert?
  - Damit würden alle Funktionen total
- **Starke Normalisierbarkeit**
  - Terminiert jede beliebige Reduktionsfolge?
  - Damit könnte jede Reduktionsstrategie gewählt werden
- **Konfluenz (Church-Rosser Eigenschaft)**
  - Kann man verschiedene Reduktionsfolgen wieder zusammenführen?
  - Damit wäre die Normalform eines Terms eindeutig
- **Entscheidbarkeit**
  - Ist Typisierbarkeit oder Gleichheit von Termen entscheidbar?
  - Damit würde Korrektheit / Äquivalenz von Programmen testbar
- **Ausdruckskraft**
  - Gibt es wichtige Funktionen, die nicht typisierbar sind?

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $\quad d(S \to T) = 1 + \max(d(S), d(T))$
  - Tiefe des Redex $(\lambda x.\, t)\, u \equiv$ Tiefe des Typs von $\lambda x.\, t$.
  - Tiefe eines Terms $t \equiv$ maximale Tiefe der Redizes von t
- **Rightmost-Maxdepth Strategie:**
  - Reduziere das am weitesten rechts stehende Redex maximaler Tiefe

Beispiel: `trice(trice(λx.x))` mit `trice` $\equiv$ `λf.λx.f(f(fx))`

Ergebnis `λx. x` Keine Redices

Rightmost-Maxdepth Strategie

Die Rightmost-Maxdepth-Strategie terminiert auf allen typisierbaren λ-Termen

- Lemma: Die Strategie verringert die Anzahl der Redizes maximaler Tiefe
  - Sei $r{=}(\lambda x.u)\,v$ das rightmost-maxdepth Redex von t
  - r wird reduziert zu dem Term $u[v/x]$ ohne Redizes maximaler Tiefe
  - Redizes von t außerhalb von r bleiben unverändert ✓
- Lemma: Hat t maximal m Redizes der maximalen Tiefe d
  dann hat t eine Normalform t' mit $t \xrightarrow{n} t'$ für ein $n \in \mathbb{N}$
  - Zeige durch Doppelinduktion über d und m mit obigem Lemma, daß die Rightmost-Maxdepth-Strategie immer terminiert ✓

$$\Downarrow$$

Jeder typisierbare λ-Term ist normalisierbar

Starke Normalisierbarkeit

Führt jede Reduktionsfolge zu einer Normalform?

- **t stark normalisierbar (SN):**
  - Jede in t beginnende Reduktionsfolge terminiert
- **Nachweis extrem aufwendig**
  - Jede Reduktionsfolge müsste analysiert werden
  - Induktionsbeweis zeigt Verschärfung von starker Normalisierbarkeit
    Jeder typisierbare λ-Term ist "berechenbar"
- **Wichtige Hilfskonzepte**
  - **Berechenbare typisierte Terme**
    - $t \in T$ für $T \in \mathcal{T}$ ist berechenbar, falls t stark normalisierbar
    - $t \in S \to T$ berechenbar, falls $t\,s$ berechenbar für berechenbare $s \in S$
  - Mehr als nur SN: jede Anwendung von t ist berechenbar
  - **Neutrale Terme**
    - t ist neutral, wenn t nicht die Gestalt $\lambda x\,.\,u$ hat

Nachweis der Starken Normalisierbarkeit

Zeige durch Induktion über die Struktur des Typs T von t und t'

1. t berechenbar $\Rightarrow$ t stark normalisierbar
2. t berechenbar, $t \xrightarrow{\beta} t'$ $\Rightarrow$ t' berechenbar
3. t neutral, $t \xrightarrow{\beta} t'$ impliziert t' berechenbar $\Rightarrow$ t berechenbar
4. t neutral, in Normalform $\Rightarrow$ t berechenbar

Zeige durch Doppel-Induktion über Anzahl von Reduktionsschritten

5. $t[s/x]$ berechenbar für berechenbare s $\Rightarrow$ $\lambda x\,.\,t$ berechenbar

Zeige durch Induktion über die Struktur von $t{=}t[x_1 \ldots x_n]$

6. Sind $x_1 ... x_n$ freie in t, b_i berechenbare Terme vom Typ der x_i dann ist $t[b_1 ... b_n/x_1 ... x_n]$ berechenbar

7. Aus 6. folgt: **Jeder typisierbare λ-Term ist berechenbar**
   - Sind $x_1 .. x_n$ freie Variablen von t, so gilt $t = t[x_1 ... x_n/x_1 ... x_n]$
   - Alle x_i sind neutral und in Normalform, also berechenbar ✓

$$\Downarrow$$

Jeder typisierbare λ-Term ist stark normalisierbar

Starke Normalisierbarkeit: Beweis von 1.–4.

1. t berechenbar $\Rightarrow$ t stark normalisierbar
2. t berechenbar, $t \xrightarrow{\beta} t'$ $\Rightarrow$ t' berechenbar
3. t neutral, $t \xrightarrow{\beta} t'$ impliziert t' berechenbar $\Rightarrow$ t berechenbar
4. t neutral, in Normalform $\Rightarrow$ t berechenbar

Simultane Induktion über Struktur des Typs T von t und t'

(4. folgt jeweils aus 3., da Terme in Normalform nicht reduzierbar sind)

- **T atomar:**

  1. t berechenbar $\Leftrightarrow$ t stark normalisierbar (per Definition) ✓
  2. t berechenbar, $t \xrightarrow{\beta} t'$, dann t' SN, also t' berechenbar ✓
     (Jede Reduktionsfolge $t',t'_1,t'_2,\ldots$ terminiert, da $t,t',t'_1,t'_2,\ldots$ terminiert)
  3. t neutral, $t \xrightarrow{\beta} t'$ impliziert t' berechenbar, dann t SN und berechenbar ✓
     (Jede Reduktionsfolge $t,t',t'',t''',\ldots$ terminiert, da $t',t'',t''',\ldots$ terminiert)

Starke Normalisierbarkeit: Beweis von 1.–4. (II)

- **Sei $T = T_1 \to T_2$, Behauptung gelte für Terme in T_1 und T_2**

  1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \ldots$ Reduktionsfolge
     $\mapsto$ x neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
     $\mapsto$ $t\,x \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
     $\mapsto$ Reduktionsfolge tx, t_1x, t_2x, $\ldots$ terminiert, also auch t, t_1, t_2, $\ldots$
     $\mapsto$ t stark normalisierbar ✓

  2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
     $\mapsto$ $t\,s \in T_2$ berechenbar und $ts \xrightarrow{\beta} t's$, also $t'\,s \in T_2$ berechenbar (Induktionsannahme 2.),
     $\mapsto$ t' berechenbar (Definition) ✓

  3. Es gelte t neutral, $t \xrightarrow{\beta} t'$ impliziert t' berechenbar. Sei $s \in T_1$ berechenbar
     $\mapsto$ s SN (Induktionsannahme 1). Wir zeigen $ts \xrightarrow{\beta} t^*$ impliziert t^* berechenbar
     durch Induktion über maximale Anzahl n der Reduktionsschritte von s
     · $n{=}0$: s ist in Normalform, also $t^* = t'\,s$ mit $t \xrightarrow{\beta} t'$. Dann t', t^* berechenbar
     · $n{+}1$: t neutral, also $t\,s$ kein Redex
       · Falls $t^* = t'\,s$ mit $t \xrightarrow{\beta} t'$, dann t^* berechenbar
       · Falls $t^* = t\,s'$ mit $s \xrightarrow{\beta} s'$, dann s' SN mit maximal n Schritten und berechenbar
         $t\,s' \xrightarrow{\beta} t^+$ impliziert t^+ berechenbar (innere Annahme) und $t\,s' \in T_2$ neutral
         also $t^* \equiv t\,s'$ berechenbar (Induktionsannahme 3.)
     $\mapsto$ $t\,s \in T_2$ neutral, also berechenbar (Induktionsannahme 3.)
     $\mapsto$ t berechenbar ✓

Starke Normalisierbarkeit: Beweis von 5.

t typisierbar, $t[s/x]$ berechenbar für berechenbare s $\Rightarrow$ $\lambda x\,.\,t$ berechenbar

Sei $t[s/x]$ berechenbar für alle berechenbare s

– Zu zeigen ist $(\lambda x.t)\,s$ berechenbar für berechenbare s
  bzw., da $(\lambda x\,.\,t)\,s$ neutral: $(\lambda x\,.\,t)\,s \xrightarrow{\beta} t^* \Rightarrow t^*$ berechenbar (nach 3)

– t ist berechenbar, denn $t{=}t[x/x]$ und jedes $x \in \mathcal{V}$ ist berechenbar nach (4)

– Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Zahl der Reduktionssschritte von t und s

· Falls $t^* = t[s/x]$, dann t^* berechenbar (↦ Argument für Basisfall) ✓

· Falls $t^* = (\lambda x.t')\,s$ mit $t \xrightarrow{\beta} t'$

  Dann t' berechenbar (2) und SN (1) mit weniger Reduktionsschritten

  Also $t^*{=}(\lambda x.t')\,s \xrightarrow{\beta} \hat{t} \Rightarrow \hat{t}$ berechenbar (Induktionsannahme für t')

  Also t^* berechenbar nach (3) (↦ Äußerer Induktionsschritt für t) ✓

· Falls $t^* = (\lambda x.t)\,s'$ mit $s \xrightarrow{\beta} s'$.

  Dann s' berechenbar (2) und SN (1) mit weniger Reduktionsschritten.

  Also $t^*{=}(\lambda x.t)\,s' \xrightarrow{\beta} \hat{t} \Rightarrow \hat{t}$ berechenbar (Induktionsannahme für s')

  Also t^* berechenbar nach (3) (↦ Innere Induktionsschritte für s) ✓

Starke Normalisierbarkeit: Beweis von 6.

t typisierbar, $x_1 \ldots x_n$ (alle) freie Variablen von t,
b_i berechenbar und vom Typ der x_i. Dann $t[b_1..b_n \,/\, x_1..x_n]$ berechenbar

Induktion über die Struktur von $t{=}t[x_1..x_n]$

· Falls $t = x_i$, dann $t[b_1..b_n \,/\, x_1..x_n] = b_i$ berechenbar ✓

· Falls $t = \lambda x.u$, dann $t[b_1..b_n \,/\, x_1..x_n] = \lambda x\,.\,u[b_1..b_n \,/\, x_1..x_n]$

und $u[b_1..b_n, b \,/\, x_1..x_n, x]$ berechenbar für alle b (Induktionsannahme)

Also $t[b_1..b_n \,/\, x_1..x_n]$ berechenbar nach (5) ✓

· Falls $t = f\,u$, dann

$t[b_1..b_n \,/\, x_1..x_n] = f[b_1..b_n \,/\, x_1..x_n]\ \ u[b_1..b_n \,/\, x_1..x_n]$

und $f[b_1..b_n \,/\, x_1..x_n]$ berechenbar

und $u[b_1..b_n \,/\, x_1..x_n]$ berechenbar (Induktionsannahme)

Also $t[b_1..b_n \,/\, x_1..x_n]$ berechenbar (Definition für 'f berechenbar') ✓

Starke Normalisierbarkeit: Vorteile

- **Jede Reduktionsstrategie kann angewandt werden**
  - Terminierung ist immer gesichert
  - Es gibt effizientere ("riskantere") Strategien als Rightmost-Maxdepth
- **Rightmost (Call-by-value):**

  `trice(trice(λx.x))`

  $\xrightarrow{8}$ `λx.x`
- **Rightmost-maxdepth:**

  `trice(trice(λx.x))`

  $\xrightarrow{14}$ `λx.x`
- **Leftmost (Call-by-name):**

  `trice(trice(λx.x))`

  $\xrightarrow{19}$ `λx.x`

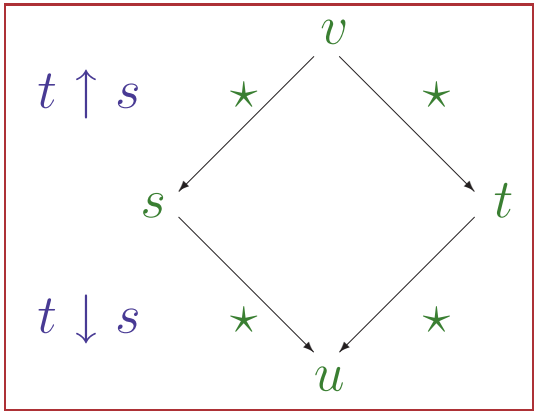
Konfluenz einer Reduktionsrelation $\xrightarrow{r}$

Ist die Normalform eines Terms eindeutig?

- **Iteration einer Reduktionsrelation** $\xrightarrow{r}$
  - $t \xrightarrow{n} s$: s ergibt sich aus t durch n Reduktionsschritte
    - $t \xrightarrow{0} s$, falls $t{=}s$, $t \xrightarrow{n+1} s$, falls es ein u gibt mit $t \xrightarrow{r} u$ und $u \xrightarrow{n} s$
  - $t \xrightarrow{*} s$: es gibt ein $n \in \mathbb{N}$ mit $t \xrightarrow{n} s$
  - $t \xrightarrow{+} s$: es gibt ein $n{>}0$ mit $t \xrightarrow{n} s$
- $t{\uparrow}s$: es gibt u mit $u \xrightarrow{*} t$ und $u \xrightarrow{*} s$

  $t{\downarrow}s$: es gibt u mit $t \xrightarrow{*} u$ und $s \xrightarrow{*} u$
- **Konfluenz:** aus $t{\uparrow}s$ folgt immer $t{\downarrow}s$

  Lokale Konfluenz: aus $u \xrightarrow{r} t$, $u \xrightarrow{r} s$ folgt $t{\downarrow}s$



Diamond Lemma

$\xrightarrow{r}$ stark normalisierbar, lokal konfluent $\Rightarrow$ $\xrightarrow{r}$ konfluent

- **Für jeden Term v gibt es ein n, so daß jede in v beginnende Reduktionsfolge in maximal n Schritten terminiert**
  - Stark normalisierbar: jede Reduktionsfolge von v terminiert
  - Endliche Verzweigung von $\xrightarrow{r}$: es gibt eine maximale Schrittzahl
- **Für alle v folgt $t{\downarrow}s$ aus $v \xrightarrow{*} t$ und $v \xrightarrow{*} s$**

  Induktion über maximale Zahl n der Reduktionsschritte

  $n{=}0$: Es folgt $v{=}t{=}s$, also $t{\downarrow}s$ trivialerweise ✓

  $n{+}1$: Falls $v \xrightarrow{0} t$ oder $v \xrightarrow{0} s$, dann $v{=}t$ oder $v{=}s$ ✓

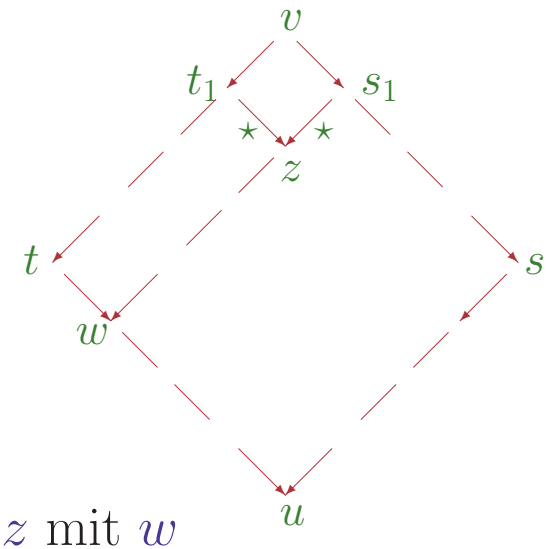
  Ansonsten $v \xrightarrow{r} t_1 \xrightarrow{*} t$ und $v \xrightarrow{r} s_1 \xrightarrow{*} s$

  Lokale Konfluenz: $t_1{\downarrow}s_1$ mit einem Term z

  Induktionsannahme für t_1: $t_1 \xrightarrow{*} t$, $t_1 \xrightarrow{*} z$ also $t{\downarrow}z$ mit w

  Induktionsannahme für s_1: $s_1 \xrightarrow{*} z \xrightarrow{*} w$, $s_1 \xrightarrow{*} s$ also $w{\downarrow}s$ mit u

  Insgesamt $t \xrightarrow{*} w \xrightarrow{*} u$ und $s \xrightarrow{*} u$, also **$t{\downarrow}s$** ✓



Church-Rosser Theorem

In der einfachen Typentheorie ist $\xrightarrow{\beta}$ konfluent

- **$\xrightarrow{\beta}$ ist lokal konfluent**

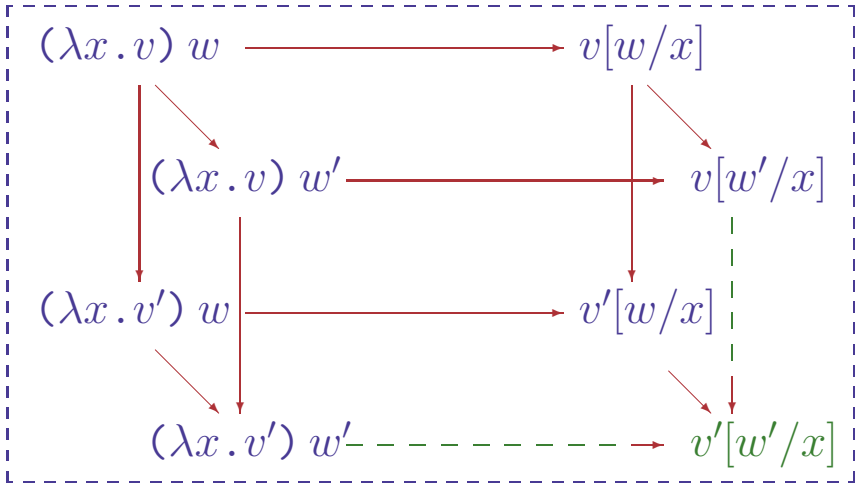
  Es gelte $u \xrightarrow{\beta} t$, $u \xrightarrow{\beta} s$ und $s \neq t$

  Betrachte Teilterm von u der Gestalt $(\lambda x\,.\,v)\,w$ oder $v\,w$, so daß

  – s entsteht durch Reduktion von v, w oder des äußeren β-Redex

  – t entsteht durch eine andere Reduktion des Teilterms

  Reduktionskette kann gemäß Diagramm zusammengeführt werden



- **Diamond Lemma + starke Normalsierbarkeit $\mapsto$ Konfluenz**

$$\Downarrow$$

Typisierbare λ-Terme haben eindeutige Normalformen

Entscheidbarkeit

Welche Eigenschaften typisierbarer Terme sind entscheidbar?

- **Typisierbarkeit**
  - Hindley-Milner Algorithmus
- **Halteproblem und Totalität von Funktionen**
  - Trivial wegen starker Normalisierbarkeit
- **Gleichheit**
  - Teste Typisierbarkeit, normalisiere im Erfolgsfall, prüfe Kongruenz
- **Korrektheit und Äquivalenz von Programmen**
  - Die Tests $f(s) = t$ und $f = g$ sind spezielle Gleichheitsprobleme
- **Programmeigenschaften, wenn als Typ codierbar**
  - Teste $t \in T$ mit Hindley-Milner Algorithmus

Wie wirkt sich das auf die Ausdruckskraft aus?

Ausdruckskraft

Welche Funktionen können durch typisierbare λ-Terme berechnet werden?

- **Church-Numerals sind typisierbar**
  - $\overline{n} \equiv \lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}^n\,\mathtt{x} \in (\mathtt{X}\to\mathtt{X}) \to \mathtt{X}\to\mathtt{X} \equiv \mathbb{N}$
  - **s** $\equiv \lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{n\ f\ (f\ x)} \in \mathbb{N}\to\mathbb{N}$
  - **add** $\equiv \lambda \mathtt{m}.\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{m\ f\ (n\ f\ x)} \in \mathbb{N}\to\mathbb{N}\to\mathbb{N}$
  - **mul** $\equiv \lambda \mathtt{m}.\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{m\ (n\ f)\ x} \in \mathbb{N}\to\mathbb{N}\to\mathbb{N}$
- **Produktbildung ist typisierbar**
  - $\langle s,t\rangle \equiv \lambda \mathtt{p}.\ \mathtt{p}\,s\,t \in (S\to T\to \mathtt{Z}) \to \mathtt{Z} \equiv S\times T$
  - $\langle s,t\rangle$**.1** $\equiv \langle s,t\rangle\ (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x}) \in S$
  - $\langle s,t\rangle$**.2** $\equiv \langle s,t\rangle\ (\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y}) \in T$

Ausdruckskraft typisierter Terme hat Grenzen

- **Typisierung boolescher Operatoren problematisch**
  - $\mathsf{T} \equiv \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x} \in \mathtt{X} \to \mathtt{Y} \to \mathtt{X}$
  - $\mathsf{F} \equiv \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y} \in \mathtt{X} \to \mathtt{Y} \to \mathtt{Y}$
  - **if** b **then** s **else** $t \equiv b\,s\,t \in \mathtt{Z}$ für $s \in \mathtt{X}$, $t \in \mathtt{Y}$, $b \in \mathbb{B} \equiv \mathtt{X} \to \mathtt{Y} \to \mathtt{Z}$
  - Aber $\mathsf{T} \in \mathbb{B}$ und $\mathsf{F} \in \mathbb{B}$ verlangt $\mathtt{X}=\mathtt{Y}=\mathtt{Z}$ und somit $\mathbb{B} \equiv \mathtt{X} \to \mathtt{X} \to \mathtt{X}$

    Das schränkt die Wahl möglicher Werte für s und t ein
- **Church Numerals brauchen polymorphes $\mathbb{N}$**
  - $\mathsf{exp} \equiv \lambda \mathtt{m}.\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{n}\ \mathtt{m}\ \mathtt{f}\ \mathtt{x} \in \mathtt{X} \to (\mathtt{X} \to \mathtt{Y} \to \mathtt{Z} \to \mathtt{T}) \to \mathtt{Y} \to \mathtt{Z} \to \mathtt{T}$
    - aber $\mathtt{m},\mathtt{n} \in \mathbb{N}$ verlangt $\mathtt{X} \to \mathtt{Y} \to \mathtt{Z} \to \mathtt{T} = \mathtt{X}$
  - $\mathsf{PRs}[\mathit{base}, h] \equiv \lambda \mathtt{n}.\ \mathtt{n}\ h\ \mathit{base} \in (\mathtt{X} \to \mathtt{Y} \to \mathtt{Z}) \to \mathtt{Z}$
    - aber $h \in \mathtt{Y}=\mathbb{N} \to \mathbb{N}$ und $\mathit{base} \in \mathtt{Z}=\mathbb{N}$ verlangt $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N} \to \mathbb{N} = \mathbb{N}$

$$\Downarrow$$

Typstruktur zu schwach für praktische Anwendungen

Schliessen über Programmierung

- **Prädikatenlogik: Analyse der logischen Struktur**
  - Bedeutung von Konzepten, Termen und Datentypen spielt keine Rolle
- **λ-Kalkül: Wert von Ausdrücken**
  - Sehr ausdrucksstark, Auswertung nur unzureichend kontrollierbar
- **Einfache Typentheorie: Programmeigenschaften**
  - Ermöglicht kontrolliertes Schließen über Wert und Eigenschaften
  - Einfache Typstruktur schränkt Ausdruckskraft zu sehr ein

$$\Downarrow$$

Verfeinere die Typstruktur der Theorie

  - Liefert erhöhte Ausdruckskraft, Integration von Logik und Berechnung
  - Verlangt komplexeres Typsystem und aufwendigere Semantik