Automatisierte Logik und Programmierung

Einheit 6

Die einfache Typentheorie

1. Syntax und Semantik
2. Type Checking und Typ-Inferenz
3. Refinement Kalkül für die Typentheorie
4. Die Curry-Howard Isomorphie
5. Eigenschaften typisierbarer Terme

Logisches Schliessen über Datentypen

- **Moderne Programmiersprachen haben Typsysteme**
  - Beschreibung der Grundstruktur von Objekten
  - Realisiert als Zuordnung eines Typs zu Programmtermen im Kontext
  - Ermöglicht statische Kontrolle elementarer Programmeigenschaften

    Viele Fehler in Programmen lassen sich als Typfehler identifizieren
- **Zwei statisch einsetzbare Kontrollmechanismen**
  - Typechecking: Überprüfung ob gegebener Typ zu einem Term passt
  - Typinferenz: Herleitung eines zum Term zugehörigen Typs
  - Beide basieren auf einem "Typzugehörigkeitskalkül"
- **Typsysteme koppeln Termstruktur an Typstruktur**
  - Terme erhalten eine Bedeutung, die über das Operationale hinausgeht
  - Ermöglicht präzise Definition der Semantik einer Programmiersprache

Typsysteme vereinigen Logik und Programmierung

- **Zwei Sichtweisen des gleichen formalen Konzepts**
  - Programmierung: Terme sind Programme, Typen ihre Eigenschaften
  - Logik: Typen sind Formeln und Terme ihre Evidenzen
  - In beiden Fällen geht es um Typzugehörigkeit
- **Zielrichtung ist unterschiedlich**
  - Typinferenz: Bestimme Datentyp eines (Programm-)terms
  - Witness finding: Finde (Evidenz-)term zu einem logischen Typ
  - Beides läßt sich in Typzugehörigkeitskalkülen beschreiben
- **Typkalkül für λ-Kalkül ist das Fundament**
  - λ-Terme sind Evidenz für Implikation und Allquantor und Elemente von (abhängigen) Funktionenräumen
  - Alle Programm- / Datenstrukturen sind durch λ-Terme beschreibbar
  - Alle Prinzipien von Typsystemen basieren auf Erkenntnissen, die mit Typkalkülen für λ-Terme gewonnen werden

Was sind eigentlich Datentypen?

- **Programmterme besitzen strukturelle Eigenschaften**
  - `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 sind syntaktische Repräsentationen dieser Eigenschaften

- **Datentypen besitzen selbst eine 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 Spezifikationen von Programmeigenschaften sein
    - $\{f{:}\mathbb{N}\to\mathbb{N} \mid \forall x{:}\mathbb{N}.\ f(x)^2 \le x < (f(x)+1)^2\}$ spezifiziert $\lambda x.\ \lfloor\sqrt{x}\rfloor$

Formale Charakteristika eines Datentyps

Syntaktische Repräsentation von Termklassen

- **Direkte Beschreibung elementarer Datentypen**
  - Festlegung eines Namens für den Typ $\mathbb{N}, \mathbb{Z}, \mathbb{B} \ldots$
  - Beschreibung der Basiselemente $\mathtt{0,1,2,\ldots}$, $\mathtt{0,1,-1,2,\ldots}$, $\mathbf{T}$,$\mathbf{F}$
  - Beschreibung von Operatoren auf Elementen $\mathtt{-}x$, $x\mathtt{+}y$, $x\mathtt{-}y$, $x\mathtt{*}y$, ...
- **Konstruktoren für strukturierte Datentypen**
  - Typkonstruktoren: $T \to T'$, $T \times T'$, $T\mathtt{+}T'$, $T\ \mathtt{list}$,...
  - Konstruktion kanonischer Elemente: $\lambda x.t$, $\mathtt{(}a\mathtt{,}b\mathtt{)}$, $\mathtt{inl(}a\mathtt{)}$, $a_1\mathtt{::}l$, ...
  - Nichtkanonische Operatoren auf Elementen: $f\mathtt{(}a\mathtt{)}$, $p\mathtt{.1}$, ... $hd\mathtt{(}l\mathtt{)}$, ...
- **Grundbestandteile der beschreibenden Syntax**
  - Typ-Ausdruck + kanonische Terme + nichtkanonische Terme
- **Alle Konzepte basieren auf Funktionstyp $T \to T'$**
  - Naheliegende Erweiterung des λ-Kalküls
  - Produkte, Summen, Abhängigkeiten, etc. kommen später hinzu

Typsysteme: Beschreibungsformen

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

    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

Einfache Typentheorie – Syntax

Minimal nötige Typkonstrukte für den λ-Kalkül

- **Erlaubte Symbole**
  - Variablen $x, y, z, x_0, y_0,, \ldots$
  - Typsymbole $S, T, S_0, T_0,, \ldots$
  - Strukturelle Symbole $\rightarrow$, λ, Punkte und Klammern
- **Typ-Ausdrücke (Typen)**
  - Typsymbole sind (atomare) Typen
  - Sind S und T Typen, dann auch $S \rightarrow T$ und (T)
- **Objekt-Ausdrücke (λ-Terme)**
  - Variablen sind Terme
  - Ist x Variable, t und f Terme, dann sind $\lambda x\,.\,t$, $f\,t$ und (t) Terme
- **Prioritätskonventionen sparen Klammern**
  - Applikation bindet stärker als λ-Abstraktion $\lambda x\,.\,f\,t \;\hat{=}\; \lambda x\,.\,(f\,t)$
  - Applikation ist links-assoziativ: $f\;t_1\,t_2 \;\hat{=}\; (f\;t_1)\,t_2$
  - $\rightarrow$ ist rechtsassoziativ: $T_1 \rightarrow T_2 \rightarrow T_3 \;\hat{=}\; T_1 \rightarrow (T_2 \rightarrow T_3)$

Einfache Typentheorie – Semantik

- **Zuordnung von Typen zu Termen**
  - $t \in T \equiv t$ ist ein Element des Datentyps T

    t beschreibt ein Objekt in der durch T beschriebenen Klasse
  - Formale Definition folgt induktivem syntaktischem Aufbau
  - Denotationelle Semantik wird gekoppelt an operationale Semantik
- **Zuordnung liefert Einschränkung "erlaubter" Terme**
  - $S \rightarrow T$ entspricht Klasse der Funktionen von S nach T
  - Typisierbare λ-Ausdrücke müssen (sinnvolle) Funktionen darstellen

    Keine unkontrollierte Rekursion oder Selbstanwendung

    Minimale Begrenzung der Ausdruckskraft des λ-Kalküls

Typzuordnung an konkrete Terme

- **Typzuordnung an Termvariablen**
  - Termvariablen sind Platzhalter für unbekannte Objekte
  - x kann zu jedem T gehören, wenn x Variable und T Typsymbol
  - Feste Zuordnung nur, wenn Annahmen über Typ von x vorliegen

Typzuordnung an konkrete Terme

- **Typzuordnung an Termvariablen**
  - Termvariablen sind Platzhalter für unbekannte Objekte
  - x kann zu jedem T gehören, wenn x Variable und T Typsymbol
  - Feste Zuordnung nur, wenn Annahmen über Typ von x vorliegen
- **Typzuordnung an λ-Abstraktionen**
  - $\lambda x.t$ muß zu einem Funktionenraum $S \rightarrow T$ gehören
  - Dabei muß t zu T gehören wenn x in S ist

Typzuordnung an konkrete Terme

- **Typzuordnung an Termvariablen**
  - Termvariablen sind Platzhalter für unbekannte Objekte
  - x kann zu jedem T gehören, wenn x Variable und T Typsymbol
  - Feste Zuordnung nur, wenn Annahmen über Typ von x vorliegen
- **Typzuordnung an λ-Abstraktionen**
  - $\lambda x.t$ muß zu einem Funktionenraum $S \rightarrow T$ gehören
  - Dabei muß t zu T gehören wenn x in S ist
- **Typzuordnung an Applikationen**
  - $f\ t$ gehört zum Bildtyp T der Funktion f, wenn der Typ S von t der Argumenttyp $f \in S \rightarrow T$ ist

Typzuordnung an konkrete Terme

- **Typzuordnung an Termvariablen**
  - Termvariablen sind Platzhalter für unbekannte Objekte
  - x kann zu jedem T gehören, wenn x Variable und T Typsymbol
  - Feste Zuordnung nur, wenn Annahmen über Typ von x vorliegen
- **Typzuordnung an λ-Abstraktionen**
  - $\lambda x.t$ muß zu einem Funktionenraum $S \rightarrow T$ gehören
  - Dabei muß t zu T gehören wenn x in S ist
- **Typzuordnung an Applikationen**
  - $f\ t$ gehört zum Bildtyp T der Funktion f, wenn der Typ S von t der Argumenttyp $f \in S \rightarrow T$ ist

Typisierung von $\lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{f}\ \mathtt{x}$

Typzuordnung an konkrete Terme

- **Typzuordnung an Termvariablen**
  - Termvariablen sind Platzhalter für unbekannte Objekte
  - x kann zu jedem T gehören, wenn x Variable und T Typsymbol
  - Feste Zuordnung nur, wenn Annahmen über Typ von x vorliegen
- **Typzuordnung an λ-Abstraktionen**
  - $\lambda x.t$ muß zu einem Funktionenraum $S \rightarrow T$ gehören
  - Dabei muß t zu T gehören wenn x in S ist
- **Typzuordnung an Applikationen**
  - $f\ t$ gehört zum Bildtyp T der Funktion f, wenn der Typ S von t der Argumenttyp $f \in S \rightarrow T$ ist

Typisierung von $\lambda \mathtt{f}.\lambda \mathtt{x}.\mathtt{f}\ \underbrace{\mathtt{x}}_{\mathtt{S}}$ für ein beliebiges $\mathtt{S}$

Typzuordnung an konkrete Terme

- **Typzuordnung an Termvariablen**
  - Termvariablen sind Platzhalter für unbekannte Objekte
  - x kann zu jedem T gehören, wenn x Variable und T Typsymbol
  - Feste Zuordnung nur, wenn Annahmen über Typ von x vorliegen
- **Typzuordnung an λ-Abstraktionen**
  - $\lambda x.t$ muß zu einem Funktionenraum $S \rightarrow T$ gehören
  - Dabei muß t zu T gehören wenn x in S ist
- **Typzuordnung an Applikationen**
  - $f\ t$ gehört zum Bildtyp T der Funktion f, wenn der Typ S von t der Argumenttyp $f \in S \rightarrow T$ ist

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

Typzuordnung an konkrete Terme

- **Typzuordnung an Termvariablen**
  - Termvariablen sind Platzhalter für unbekannte Objekte
  - x kann zu jedem T gehören, wenn x Variable und T Typsymbol
  - Feste Zuordnung nur, wenn Annahmen über Typ von x vorliegen
- **Typzuordnung an λ-Abstraktionen**
  - $\lambda x.t$ muß zu einem Funktionenraum $S \rightarrow T$ gehören
  - Dabei muß t zu T gehören wenn x in S ist
- **Typzuordnung an Applikationen**
  - $f\ t$ gehört zum Bildtyp T der Funktion f, wenn der Typ S von t der Argumenttyp $f \in S \rightarrow T$ ist

Typisierung von $\lambda \mathtt{f}\,.\,\lambda \mathtt{x}\,.\,\overbrace{\underbrace{\mathtt{f}}_{\mathtt{S}\rightarrow\mathtt{T}}\ \underbrace{\mathtt{x}}_{\mathtt{S}}}^{\mathtt{T}}$ für beliebige $\mathtt{S}$ und $\mathtt{T}$

Typzuordnung an konkrete Terme

- **Typzuordnung an Termvariablen**
  - Termvariablen sind Platzhalter für unbekannte Objekte
  - x kann zu jedem T gehören, wenn x Variable und T Typsymbol
  - Feste Zuordnung nur, wenn Annahmen über Typ von x vorliegen
- **Typzuordnung an λ-Abstraktionen**
  - $\lambda x.t$ muß zu einem Funktionenraum $S \rightarrow T$ gehören
  - Dabei muß t zu T gehören wenn x in S ist
- **Typzuordnung an Applikationen**
  - $f\ t$ gehört zum Bildtyp T der Funktion f, wenn der Typ S von t der Argumenttyp $f \in S \rightarrow T$ ist

Typisierung von $\lambda \mathtt{f}.\lambda \mathtt{x}.\underbrace{\overbrace{\underbrace{\mathtt{f}}_{\mathtt{S}\rightarrow\mathtt{T}}\ \underbrace{\mathtt{x}}_{\mathtt{S}}}^{\mathtt{T}}}_{\mathtt{S}\rightarrow\mathtt{T}}$ für beliebige $\mathtt{S}$ und $\mathtt{T}$

Typzuordnung an konkrete Terme

- **Typzuordnung an Termvariablen**
  - Termvariablen sind Platzhalter für unbekannte Objekte
  - x kann zu jedem T gehören, wenn x Variable und T Typsymbol
  - Feste Zuordnung nur, wenn Annahmen über Typ von x vorliegen
- **Typzuordnung an λ-Abstraktionen**
  - $\lambda x.t$ muß zu einem Funktionenraum $S \to T$ gehören
  - Dabei muß t zu T gehören wenn x in S ist
- **Typzuordnung an Applikationen**
  - $f\ t$ gehört zum Bildtyp T der Funktion f, wenn der Typ S von t der Argumenttyp $f \in S \to T$ ist

Typisierung von $\underbrace{\lambda \mathtt{f}.\underbrace{\lambda \mathtt{x}.\overbrace{\underbrace{\mathtt{f}}_{\mathtt{S}\to\mathtt{T}}\ \underbrace{\mathtt{x}}_{\mathtt{S}}}^{\mathtt{T}}}_{\mathtt{S}\to\mathtt{T}}}_{\overbrace{(\mathtt{S}\to\mathtt{T})\to(\mathtt{S}\to\mathtt{T})}}$ für beliebige $\mathtt{S}$ und $\mathtt{T}$

Typzugehörigkeit: wichtige Erkenntnisse

- **Nur geschlossene Terme sind vollständig typisierbar**
  - Typ offener Terme benötigt Annahmen zu Typen der freien Variablen
- **Nicht jeder (geschlossene) λ-Term ist typisierbar**
  - Sonst gäbe es eine einfache denotationelle 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
- **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)

Semantik der Typentheorie präzisiert

- **Urteile definieren Typzugehörigkeitsrelation $t \in T$**
  - $x \in T$ nur wenn $x \in T$ bekannt und x Termvariable
  - $\lambda x\,.\,t \in S \to T$ 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 gelten
  - $(t) \in T$ falls $t \in T$ gilt
  - $t \in (T)$ falls $t \in T$ gilt

  Ein Term t heißt typisierbar, falls $t \in T$ für einen Typ T gilt

  Das prinzipielles Typschema von t ist der allgemeinste Typ T mit $t \in T$

- **λ-Kalkül definiert semantische Gleichheit von Termen**
  - $t = u$, falls es einen Term v gibt mit $t \xrightarrow{*} v$ und $u \xrightarrow{*} v$
  - Reduktion erhält Typzugehörigkeit: $(t \in T \wedge t \xrightarrow{*} t') \Rightarrow t' \in T$ (Folie 11)

    Semantisch gleiche typisierbare Terme gehören zum gleichen Typ
  - Umkehrung gilt nicht: $(t' \in T \wedge t \xrightarrow{*} t') \not\Rightarrow t \in T$

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

    aber $(\lambda \mathtt{f}\,.\,\lambda \mathtt{y}\,.\,\mathtt{y})(\lambda \mathtt{x}\,.\,\mathtt{x}\,\mathtt{x}) \xrightarrow{*} \lambda \mathtt{y}\,.\,\mathtt{y} \in \mathtt{T} \to \mathtt{T}$

Typzugehörigkeit harmoniert mit Reduktion

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

1. Reduktion erhält den Typ: aus $t \xrightarrow{\beta} t'$ folgt $t \in T \Leftrightarrow t' \in T$

Beweis durch Induktion über die Termstruktur von t (Skizze)

– t Variable: t ist in Normalform, also ist t' identisch zu 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 $t' \in T$ ✓

– $t \equiv (\lambda x\,.\,u)\ v$ und $t' \equiv u[v/x]$: 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. ✓

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)

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

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** λ`f.`λ`x. f x`

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 $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$**

$$\underbrace{\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}}_{\mathtt{X}_0} \in \mathtt{X}_0$$

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 $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$**

$$\underbrace{\lambda \underbrace{\mathtt{f}}_{X_1}.\underbrace{\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}}_{X_2}}_{X_0} \in X_1 \rightarrow X_2 \qquad X_0 = X_1 \rightarrow X_2$$

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 $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$**

$$\lambda \underbrace{\mathtt{f}}_{X_1}.\underbrace{\lambda \underbrace{\mathtt{x}}_{X_3}.\ \underbrace{\mathtt{f}\ \mathtt{x}}_{X_4}}_{X_2} \in X_1 \rightarrow (X_3 \rightarrow X_4) \qquad X_2 = X_3 \rightarrow X_4$$

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 $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$**

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

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 $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$**

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

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 $\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x}$**

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** $\underbrace{\lambda \mathtt{x}.\ \mathtt{x}\ \mathtt{x}}_{X_0} \in X_0$

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 $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$**

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

- **Typisierung von** $\underbrace{\lambda \underbrace{\mathtt{x}}_{X_1}.\ \underbrace{\mathtt{x}\ \mathtt{x}}_{X_2}}_{X_0} \in X_1 \to X_2$

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** $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$

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

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

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 $\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ \mathtt{x}$**

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

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

$X_1 = X_1 \to X_2$

nicht typisierbar

Hindley-Milner Type-Checking Algorithmus

- **Eingabe**: geschlossener λ-Term t

  Ausgabe: prinzipielles Typschema von t oder Fehlermeldung

  Interne Daten: Umgebung *Env* deklarierter Typzugehörigkeiten

  Globale Substitution σ von Typvariablen durch Typen

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

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

  Falls $t{=}x$ (Variable): 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 $\mathtt{X}_{i+1}$, unifiziere $\sigma(S_1)$ mit $S_2 \rightarrow \mathtt{X}_{i+1}$.

  Setze $\sigma := \sigma' \circ \sigma$, wobei σ' Ergebnis der Unifikation

  Ausgabe: $\sigma(\mathrm{X}_{i+1})$ (Fehlermeldung, wenn Unifikation fehlschlägt)

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

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

  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
	λf.λx. f(f(f x))			

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

Env	*Aktueller Term t*	*Substitution σ*	UNIFY	*Typ T*
	λf.λx. f(f(f x))			
f:X_0	λx. f(f(f x))			

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

Env	*Aktueller Term t*	*Substitution σ*	UNIFY	*Typ T*
	λf.λx. f(f(f x))			
f:X_0	λx. f(f(f x))			
f:X_0, x:X_1	f(f(f x))			

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

Env	*Aktueller Term t*	*Substitution σ*	UNIFY	*Typ T*
	λf.λx. f(f(f x))			
f:X_0	λx. f(f(f x))			
f:X_0, x:X_1	f(f(f x))			
f:X_0, x:X_1	f			X_0

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

Env	*Aktueller Term t*	*Substitution σ*	UNIFY	*Typ T*
	λf.λx. f(f(f x))			
f:X_0	λx. 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)			

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

Env	*Aktueller Term t*	*Substitution σ*	UNIFY	*Typ T*
	λf.λx. f(f(f x))			
f:X_0	λx. 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

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

Env	*Aktueller Term t*	*Substitution σ*	UNIFY	*Typ T*
	λf.λx. f(f(f x))			
f:X_0	λx. 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			

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

Env	*Aktueller Term t*	*Substitution σ*	UNIFY	*Typ T*
	λf.λx. f(f(f x))			
f:X_0	λx. 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

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

Env	*Aktueller Term t*	*Substitution σ*	UNIFY	*Typ T*
	λf.λx. f(f(f x))			
f:X_0	λx. 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

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

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

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

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

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

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

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

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

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

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

Refinement Kalkül für die Typentheorie

- **Umsetzung der semantischen Urteile als Inferenzregeln**
  - Urteile sind schematische Regeln zur Definition von Semantik
  - Inferenzregeln sind syntaktische Vorschriften für Beweisführung
- **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 Struktur von Datentypen**
  - Regeln zur strukturellen Analyse von (Typ-)Ausdrücken

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 $t \in T$ zur Relation $s{=}t \in T$**
  - Prädikat mit fester Bedeutung: "s und t sind im Typ T gleich"
  - $s{=}t \in T$ bedingt neben der Gleichheit $s{=}t$ auch $s \in T$ und $t \in T$
  - Typzugehörigkeit $t \in T$ wird als Kurzform für $t{=}t \in T$ betrachtet
  - Inferenzregeln für Typzugehörigkeit werden als Spezialfall der entsprechenden Gleichheitsregeln angesehen
- **Allgemeine Reflexivitätsregel wird ungültig**
  - $t{=}t \in T$ gilt nur, wenn auch $t \in T$ gezeigt werden kann

Erweiterungen des logischen Refinement Kalküls

- **Sequenzen enthalten Variablen- und Typ-Deklarationen**
  - $x : S$ (Term-Variablendeklaration), oder $T : \mathbb{U}$ (Typdeklaration)
  - Konzept der Deklaration in Sequenzen bekommt zusätzlichen Sinn
- **Neues festes Symbol $\mathbb{U}$ (Universum aller Typen)**
  - $\mathbb{U}$ war bisher nur Platzhalter innerhalb von Variablendeklarationen
- **Konklusion kann Typisierungen enthalten**
  - Typisierte Gleichheit: $s{=}t \in T$ (oder Kurzform $t \in T$)
  - Typeigenschaft: $T \in \mathbb{U}$ ("T ist ein Typ")
- **Definition der Sequenz wird verfeinert: $H_1, .., H_n \vdash C$**
  - $H_i = x_1{:}A_1$ Deklaration, C (Formel mit) Typisierung(en)
  - $H_1, .., H_n$ 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 Typvariablen deklariert
- **Definition von Regel, Beweis, Theorem etc. unverändert**

Refinement-Regeln der einfachen Typentheorie

- **Regel für β-Reduktion**
  - Regel des λ-Kalküls wird erweitert um Typ der Terme in Konklusion

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

Refinement-Regeln der einfachen Typentheorie

- **Regel für β-Reduktion**
  - Regel des λ-Kalküls wird erweitert um Typ der Terme in Konklusion

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

- **Kongruenzregel für Applikation**
  - Direkte Umsetzung des Urteils erweitert Regel des λ-Kalküls um Typen

$$\frac{H \vdash f\, t = g\, u \in T}{\begin{array}{l} H \vdash f = g \in S \to T \\ H \vdash t = u \in S \end{array}} \quad \texttt{applyEq}\ S \quad \frac{H \vdash f\, t \in T}{\begin{array}{l} H \vdash f \in S \to T \\ H \vdash t \in S \end{array}}$$

Kurzform für Typzugehörigkeit

  - Argumenttyp S muß als Parameter der Regel angegeben werden

Refinement-Regeln der einfachen Typentheorie

- **Regel für β-Reduktion**
  - Regel des λ-Kalküls wird erweitert um Typ der Terme in Konklusion

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

- **Kongruenzregel für Applikation**
  - Direkte Umsetzung des Urteils erweitert Regel des λ-Kalküls um Typen

$$\frac{H \vdash f\, t = g\, u \in T}{\begin{array}{l} H \vdash f = g \in S \to T \\ H \vdash t = u \in S \end{array}} \quad \texttt{applyEq}\ S$$

  - Argumenttyp S muß als Parameter der Regel angegeben werden

- **Kongruenzregeln für Abstraktion**

$$\frac{H \vdash \lambda x.t = \lambda y.u \in S \to T}{\begin{array}{l} H, x'{:}S \vdash t[x'/x] = u[x'/y] \in T \\ H \vdash S \in \mathbb{U} \end{array}} \quad \texttt{lambdaEq} \quad (x' \text{ neue Variable, nicht frei in } H)$$

  - Teilziel muß Typdeklaration $x'{:}S$ der λ-Variablen enthalten
  - Ausdruck S muß als Typ nachgewiesen werden

Refinement-Regeln der einfachen Typentheorie (II)

- **Neue Typdekompositionsregel für Schließen über Typen**

$$\frac{H \vdash S \rightarrow T \in \mathbb{U} \qquad H \vdash S \in \mathbb{U} \qquad H \vdash T \in \mathbb{U}}{}$$ `functionI`

Refinement-Regeln der einfachen Typentheorie (II)

- **Neue Typdekompositionsregel für Schließen über Typen**

$$\frac{H \vdash S \rightarrow T \in \mathbb{U} \qquad H \vdash S \in \mathbb{U} \qquad H \vdash T \in \mathbb{U}}{}$$ `functionI`

- **Reflexivitätsregel benötigt Typdeklaration**

$$\frac{H \vdash x = x}{}$$ `declaration`

Auch geeignet für Beweis von Typdeklarationen $H, T{:}\mathbb{U}, H' \vdash T \in \mathbb{U}$

Refinement-Regeln der einfachen Typentheorie (II)

- **Neue Typdekompositionsregel für Schließen über Typen**

$$\frac{H \vdash S \rightarrow T \in \mathbb{U} \qquad H \vdash S \in \mathbb{U} \qquad H \vdash T \in \mathbb{U}}{}$$

`functionI`

- **Reflexivitätsregel benötigt Typdeklaration**

$$\frac{H, x{:}T, H' \vdash x = x \in T}{}$$

`declaration`

Auch geeignet für Beweis von Typdeklarationen $H, T{:}\mathbb{U}, H' \vdash T \in \mathbb{U}$

- **Symmetrie, Transitivität, Substitution**

$$\frac{H \vdash t = u \in T \qquad H \vdash u = t \in T}{}$$

`symmetry`

$$\frac{H \vdash t = u \in T \qquad H \vdash t = s \in T \qquad H \vdash s = u \in T}{}$$

`transitivity` s

$$\frac{H \vdash C[t/x] \qquad H \vdash t = u \in S \qquad H \vdash C[u/x] \qquad H, x{:}S \vdash C[x] \in \mathbb{U}}{}$$

`subst` $t{=}u \in S$

– Typ der ersetzten Terme muß als Parameter der Regel angegeben werden

– Ausdruck C in Konklusion muß für alle Instanzen von x wohlgeformt sein

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

$T{:}\mathbb{U} \vdash (\lambda f.\lambda x.\, fx)(\lambda z.z) \in T \to T$	BY applyI T→T
1. $T{:}\mathbb{U} \vdash \lambda f.\lambda x.\, fx \in (T \to T) \to (T \to T)$	BY lambdaI
1.1. $T{:}\mathbb{U}, f{:}T \to T \vdash \lambda x.\, fx \in T \to T$	BY lambdaI
1.1.1. $T{:}\mathbb{U}, f{:}T \to T, x{:}T \vdash fx \in T$	BY applyI T
1.1.1.1. $T{:}\mathbb{U}, f{:}T \to T, x{:}T \vdash f \in T \to T$	BY declaration
1.1.1.2. $T{:}\mathbb{U}, f{:}T \to T, x{:}T \vdash x \in T$	BY declaration
1.1.2. $T{:}\mathbb{U}, f{:}T \to T \vdash T \in \mathbb{U}$	BY declaration
1.2. $T{:}\mathbb{U} \vdash T \to T \in \mathbb{U}$	BY functionI
1.2.1. $T{:}\mathbb{U} \vdash T \in \mathbb{U}$	BY declaration
1.2.2. $T{:}\mathbb{U} \vdash T \in \mathbb{U}$	BY declaration
2. $T{:}\mathbb{U} \vdash \lambda z.z \in T \to T$	BY lambdaI
2.1. $T{:}\mathbb{U}, z{:}T \vdash z \in T$	BY declaration
2.2. $T{:}\mathbb{U} \vdash T \in \mathbb{U}$	BY declaration

Typisierungsregeln vs. logische Regeln

lambdaEq

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

impliesR

$$\frac{H \vdash A \Rightarrow B \quad \texttt{ev} = \lambda a.b \qquad H, a:A \vdash B \quad \texttt{ev} = b}{}$$

applyEq S

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

modus ponens A

$$\frac{H \vdash B \quad \texttt{ev} = f(a) \qquad H \vdash A \Rightarrow B \quad \texttt{ev} = f \qquad H \vdash A \quad \texttt{ev} = a}{}$$

Originalregel `impliesL` ist Variante von `modus ponens`, die auf Hypothesen arbeitet und $A \Rightarrow B$ explizit voraussetzt

$$\frac{H, f:A \Rightarrow B, H' \vdash C \quad \texttt{ev} = c[f(a)/b] \qquad H, f:A \Rightarrow B, H' \vdash A \quad \texttt{ev} = a \qquad H, b:B, H' \vdash C \quad \texttt{ev} = c}{}$$

Typisierungsregeln vs. logische Regeln

lambdaEq

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

impliesR

$$\frac{H \vdash A \Rightarrow B \quad \texttt{ev} = \lambda a.b \qquad H, a:A \vdash B \quad \texttt{ev} = b}{}$$

applyEq S

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

modus ponens A

$$\frac{H \vdash B \quad \texttt{ev} = f(a) \qquad H \vdash A \Rightarrow B \quad \texttt{ev} = f \qquad H \vdash A \quad \texttt{ev} = a}{}$$

Originalregel `impliesL` ist Variante von `modus ponens`, die auf Hypothesen arbeitet und $A \Rightarrow B$ explizit voraussetzt

$$\frac{H, f:A \Rightarrow B, H' \vdash C \quad \texttt{ev} = c[f(a)/b] \qquad H, f:A \Rightarrow B, H' \vdash A \quad \texttt{ev} = a \qquad H, b:B, H' \vdash C \quad \texttt{ev} = c}{}$$

Zusammenhang wurde bekannt als **Curry-Howard Isomorphie**

Typ	Formel
Variable vom Typ S	Annahme (von Evidenz für 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 (rechts)
(Regel der) Applikation	$\Rightarrow$-Analyse (links)

Curry-Howard Isomorphie lässt sich fortsetzen

Erweitere Typentheorie um Produkte und Summen

- **Zusätzliche Typ- und Objekt-Ausdrücke**
  - Sind S und T Typen, dann auch $S \times T$, $S+T$
  - Sind x, y, z Variablen, a, b, s, t, u Terme, dann sind auch (a, b), $t.1$, $t.2$, $\texttt{inl}(a)$, $\texttt{inr}(b)$, und $\texttt{case}\ u\ \texttt{of}\ \texttt{inl}(x) \to s \mid \texttt{inr}(y) \to t$ Terme
- **Zusätzliche Urteile der Typzugehörigkeitsrelation $t \in T$**
  - $(a, b) \in S \times T$ falls $a \in S$ und $b \in T$
  - $t.1 \in S$ und $t.2 \in T$ falls $t \in S \times T$
  - $\texttt{inl}(a) \in S+T$ und $\texttt{inr}(b) \in S+T$ falls $a \in S$ bzw. $b \in T$
  - $\texttt{case}\ u\ \texttt{of}\ \texttt{inl}(x) \to s \mid \texttt{inr}(y) \to t \in T'$ falls $u \in S+T$ und $s \in T'$ aus $x \in S$ sowie $t \in T'$ aus $y \in T$ folgt
- **Zusätzliche Reduktion und Inferenzregeln entsprechend**
  - Regeln für Produkttyp entsprechen denen der Konjunktion
  - Regeln für Summentyp entsprechen denen der Disjunktion

Formale Logik kann als Spezialfall der Typentheorie realisiert werden

Eigenschaften typisierbarer Terme

- **Schwache Normalisierbarkeit**
  - Hat jeder Term eine Normalform (d.h. sind alle Funktionen total)?
  - Führt mindestens eine Reduktionsstrategie immer zu einem Wert?
- **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?

Alle Erkenntnisse gelten auch für erweiterte Typentheorie

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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(f x))`

Reduziere `trice(trice(λx.x))` Tiefe 2

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\texttt{trice}\,(\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$ Nach Reduktion

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\texttt{trice}\,(\lambda \texttt{x}.\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$ Tiefe 2

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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(f x))`

Reduziere `λx. (λx.(λx.x)((λx.x)((λx.x)x)))`

`((λx.(λx.x)((λx.x)((λx.x)x)))`

`((λx.(λx.x)((λx.x)((λx.x)x))) x) )` Nur Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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(f x))`

Reduziere `λx. (λx.(λx.x)((λx.x)((λx.x)x)))`

`((λx.(λx.x)((λx.x)((λx.x)x)))`

`((λx.(λx.x)((λx.x)((λx.x)x)))x) )` Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\lambda \texttt{x.}\ (\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x}))\ \texttt{x})\ )$ Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\lambda \texttt{x.}\ (\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,\texttt{x})\ \texttt{x})\ )$ Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\lambda \texttt{x.}\ (\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x.}\,\texttt{x})\ \texttt{x})\ )$ Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\lambda \texttt{x.}\ (\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$\texttt{x}\ )$ Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f}\,\texttt{x}))$

Reduziere $\lambda \texttt{x}.\ (\lambda \texttt{x}.\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x}.\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x}))$

$\texttt{x}\ )$ Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\lambda \texttt{x.}\ (\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,\texttt{x})$

$\texttt{x}\ )$ Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\lambda \texttt{x.}\ (\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$((\lambda \texttt{x.}\ \texttt{x})$

$\texttt{x}\ )$ Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\lambda \texttt{x.}\ (\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x})))$

$\texttt{x}$ Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Reduziere $\lambda \texttt{x.}\ (\lambda \texttt{x.}\,(\lambda \texttt{x.x})\,((\lambda \texttt{x.x})\,\texttt{x}))$

$\texttt{x}$ Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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(f x))`

Reduziere `λx. (λx.(λx.x) x)`

`x` Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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(f x))`

Reduziere `λx. (λx.x)`

`x` Tiefe 1

Schwache Normalisierbarkeit

Hat jeder typisierbare λ-Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**
  - $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \to T) = 1 + \max(d(S), d(T))$
  - 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: $\texttt{trice}\,(\texttt{trice}\,(\lambda \texttt{x.x}))$ mit $\texttt{trice} \equiv \lambda \texttt{f}.\lambda \texttt{x}.\,\texttt{f}(\texttt{f}(\texttt{f x}))$

Ergebnis $\lambda \texttt{x.}\ \texttt{x}$ Keine Redices

Rightmost-Maxdepth Strategie

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

- Lemma: Die Strategie verringert Anzahl der Redizes maximaler Tiefe

Rightmost-Maxdepth Strategie

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

- Lemma: Die Strategie verringert 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 ✓

Rightmost-Maxdepth Strategie

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

- Lemma: Die Strategie verringert 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 ✓

Rightmost-Maxdepth Strategie

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

- Lemma: Die Strategie verringert 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

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"

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 $x_1 \ldots x_n$ freie Variablen in t, b_i berechenbare Terme vom Typ der x_i, dann ist $t[b_1 \ldots b_n/x_1 \ldots x_n]$ berechenbar
7. Aus 6. folgt: **Jeder typisierbare λ-Term ist berechenbar**
   - Sind $x_1 .. x_n$ freie Variablen von t, so gilt $t = t[x_1 \ldots x_n/x_1 \ldots x_n]$
   - 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)

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 $t \in T$ 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 ✓

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 $t \in T$ 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) ✓

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 $t \in T$ 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 jede Variable x ist berechenbar nach (4)

– Also t und s stark normalisierbar nach (1)

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 jede Variable x ist berechenbar nach (4)

– Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Summe der Reduktionssschritte von t und s

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 jede Variable x ist berechenbar nach (4)

– Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Summe 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=$t[x_1..x_n]$

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 ✓

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) ✓

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

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))`

Starke Normalisierbarkeit: Vorteile

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

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{1}\ & \texttt{trice}\,(\lambda\texttt{x.}\,(\lambda\texttt{x.x})\,(\,(\lambda\texttt{x.x})\,(\,(\lambda\texttt{x.x})\,\texttt{x}))) \end{aligned}$$

Starke Normalisierbarkeit: Vorteile

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

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{2}\ & \texttt{trice}\,(\lambda\texttt{x.}\,(\lambda\texttt{x.x})\,(\,(\lambda\texttt{x.x})\,\texttt{x})) \end{aligned}$$

Starke Normalisierbarkeit: Vorteile

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

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{3}\ & \texttt{trice}\,(\lambda\texttt{x.}\,(\lambda\texttt{x.x})\,\texttt{x}) \end{aligned}$$

Starke Normalisierbarkeit: Vorteile

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

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{4} & \texttt{trice(}\lambda\texttt{x.x)} \end{aligned}$$

Starke Normalisierbarkeit: Vorteile

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

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{5} & \lambda\texttt{x.}\,(\lambda\texttt{x.x})\,(\,(\lambda\texttt{x.x})\,(\,(\lambda\texttt{x.x})\,\texttt{x})) \end{aligned}$$

Starke Normalisierbarkeit: Vorteile

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

$$\texttt{trice(trice(}\lambda\texttt{x.x))}$$
$$\xrightarrow{6} \lambda\texttt{x.}\,(\lambda\texttt{x.x})\,(\,(\lambda\texttt{x.x})\,\texttt{x})$$

Starke Normalisierbarkeit: Vorteile

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

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{7} & \lambda\texttt{x.}\,(\lambda\texttt{x.x})\,\texttt{x} \end{aligned}$$

Starke Normalisierbarkeit: Vorteile

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

$$\texttt{trice(trice(}\lambda\texttt{x.x))}$$
$$\xrightarrow{8} \lambda\texttt{x.x}$$

Starke Normalisierbarkeit: Vorteile

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

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{8} \lambda\texttt{x.x}$$

- **Rightmost-maxdepth:**

$$\texttt{trice(trice(}\lambda\texttt{x.x))}$$

Starke Normalisierbarkeit: Vorteile

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

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{8} \lambda\texttt{x.x}$$

- **Rightmost-maxdepth:**

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{14} \lambda\texttt{x.x}$$

Starke Normalisierbarkeit: Vorteile

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

  $\xrightarrow{8} \lambda\texttt{x.x}$
- **Rightmost-maxdepth:**

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

Starke Normalisierbarkeit: Vorteile

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

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{8} \lambda\texttt{x.x}$$

- **Rightmost-maxdepth:**

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{14} \lambda\texttt{x.x}$$

- **Leftmost (Call-by-name):**

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{1} \lambda\texttt{x.(trice(}\lambda\texttt{x.x))((trice(}\lambda\texttt{x.x))((trice(}\lambda\texttt{x.x))x))}$$

Starke Normalisierbarkeit: Vorteile

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

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{8}\ & \lambda\texttt{x.x} \end{aligned}$$

- **Rightmost-maxdepth:**

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{14}\ & \lambda\texttt{x.x} \end{aligned}$$

- **Leftmost (Call-by-name):**

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{2}\ & \lambda\texttt{x.}\ (\lambda\texttt{x.}\ (\lambda\texttt{x.x})\ (\ (\lambda\texttt{x.x})\ (\ (\lambda\texttt{x.x})\ \texttt{x})))\ (\ (\texttt{trice}\ (\lambda\texttt{x.x})) \\ & \qquad (\ (\texttt{trice}\ (\lambda\texttt{x.x}))\ \texttt{x})) \end{aligned}$$

Starke Normalisierbarkeit: Vorteile

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

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{8}\ & \lambda\texttt{x.x} \end{aligned}$$

- **Rightmost-maxdepth:**

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{14}\ & \lambda\texttt{x.x} \end{aligned}$$

- **Leftmost (Call-by-name):**

$$\begin{aligned} & \texttt{trice(trice(}\lambda\texttt{x.x))} \\ \xrightarrow{3}\ & \lambda\texttt{x.((}\lambda\texttt{x.x)((}\lambda\texttt{x.x)((}\lambda\texttt{x.x)(trice(}\lambda\texttt{x.x)))))} \\ & \quad\texttt{((trice(}\lambda\texttt{x.x))x)} \end{aligned}$$

Starke Normalisierbarkeit: Vorteile

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

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{8} \lambda\texttt{x.x}$$

- **Rightmost-maxdepth:**

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{14} \lambda\texttt{x.x}$$

- **Leftmost (Call-by-name):**

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{4} \lambda\texttt{x.((}\lambda\texttt{x.x)((}\lambda\texttt{x.x)(trice(}\lambda\texttt{x.x))))((trice(}\lambda\texttt{x.x))x)}$$

Starke Normalisierbarkeit: Vorteile

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

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{8} \lambda\texttt{x.x}$$

- **Rightmost-maxdepth:**

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{14} \lambda\texttt{x.x}$$

- **Leftmost (Call-by-name):**

$$\texttt{trice(trice(}\lambda\texttt{x.x))} \xrightarrow{5} \lambda\texttt{x.((}\lambda\texttt{x.x)(trice(}\lambda\texttt{x.x)))((trice(}\lambda\texttt{x.x))x)}$$

Starke Normalisierbarkeit: Vorteile

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

  $\xrightarrow{8} \lambda\texttt{x.x}$
- **Rightmost-maxdepth:**

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

  $\xrightarrow{6} \lambda\texttt{x.(trice(}\lambda\texttt{x.x))((trice(}\lambda\texttt{x.x))x)}$

Starke Normalisierbarkeit: Vorteile

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

  $\xrightarrow{8} \lambda\texttt{x.x}$
- **Rightmost-maxdepth:**

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

  $\xrightarrow{7} \lambda\texttt{x.(trice(}\lambda\texttt{x.x))((trice(}\lambda\texttt{x.x))x)}$

Starke Normalisierbarkeit: Vorteile

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

  $\xrightarrow{8} \lambda\texttt{x.x}$
- **Rightmost-maxdepth:**

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

  $\xrightarrow{13} \lambda\texttt{x.(trice(}\lambda\texttt{x.x))x}$

Starke Normalisierbarkeit: Vorteile

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

  $\xrightarrow{8} \lambda\texttt{x.x}$
- **Rightmost-maxdepth:**

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

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

  $\texttt{trice(trice(}\lambda\texttt{x.x))}$

  $\xrightarrow{19} \lambda\texttt{x.x}$

Konfluenz einer Reduktionsrelation $\xrightarrow{r}$

Ist die Normalform eines Terms eindeutig?

Konfluenz einer Reduktionsrelation $\xrightarrow{r}$

Ist die Normalform eines Terms eindeutig?

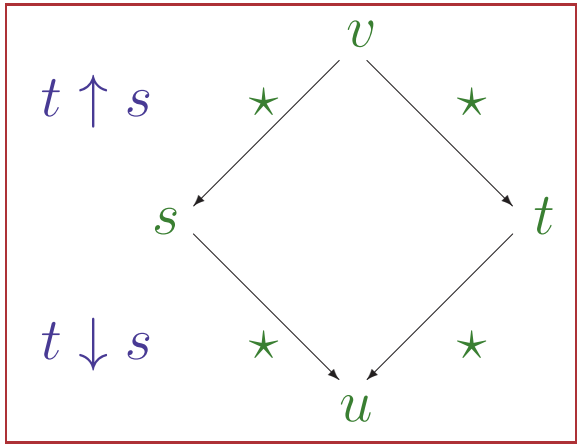
- **Iteration einer Reduktionsrelation** $\xrightarrow{r}$
  - $\boldsymbol{t \xrightarrow{n} s}$: s ergibt sich aus t durch n Reduktionsschritte
    $t \xrightarrow{0} s$, falls t=s, $t \xrightarrow{n+1} s$, falls es ein u gibt mit $t \xrightarrow{r} u$ und $u \xrightarrow{n} s$
  - $\boldsymbol{t \xrightarrow{*}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

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 $t \xrightarrow{n} s$

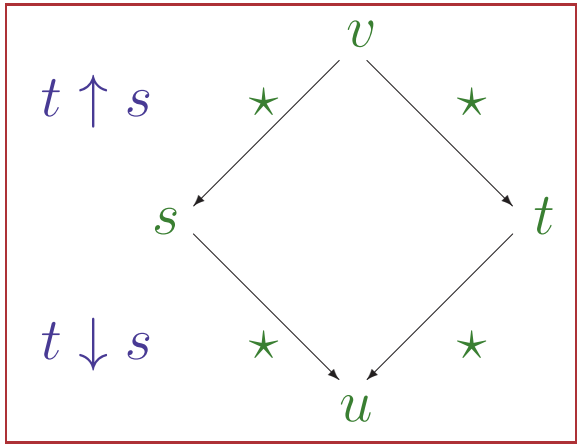
- $t \uparrow s$: es gibt u mit $u \xrightarrow{*} t$ und $u \xrightarrow{*} s$
  $t \downarrow s$: es gibt u mit $t \xrightarrow{*} u$ und $s \xrightarrow{*} u$



Konfluenz 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

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 $v \xrightarrow{*} s$**

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

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 ✓

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 ✓

Diamond Lemma

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

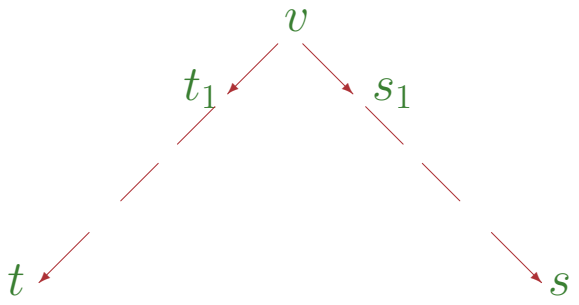
- **Für jeden Term v gibt es ein n, so daß jede in v beginnende Reduktionsfolge in maximal n Schritten terminiert**
  - Stark normalisierbar: jede Reduktionsfolge von v terminiert
  - Endliche Verzweigung von $\xrightarrow{r}$: es gibt eine maximale Schrittzahl
- **Für alle v folgt $t{\downarrow}s$ aus $v \xrightarrow{*} t$ 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 $v \xrightarrow{r} s_1 \xrightarrow{*} s$



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 $v \xrightarrow{*} s$**

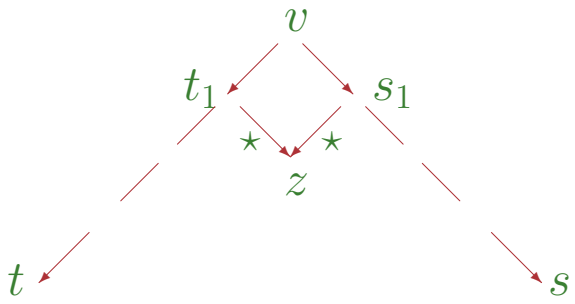
  Induktion über maximale Zahl n der Reduktionsschritte

  $n{=}0$: 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



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

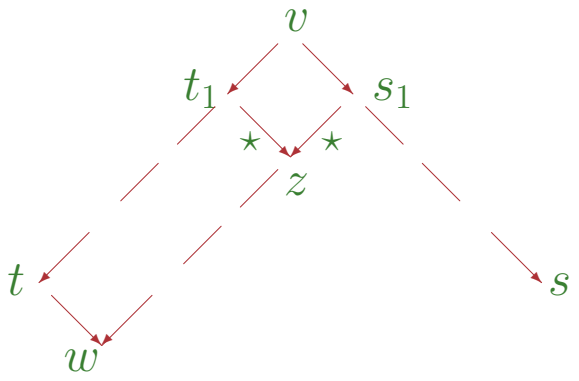
  $n{=}0$: Es folgt $v{=}t{=}s$, also $t{\downarrow}s$ trivialerweise ✓

  $n{+}1$: 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



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 ✓

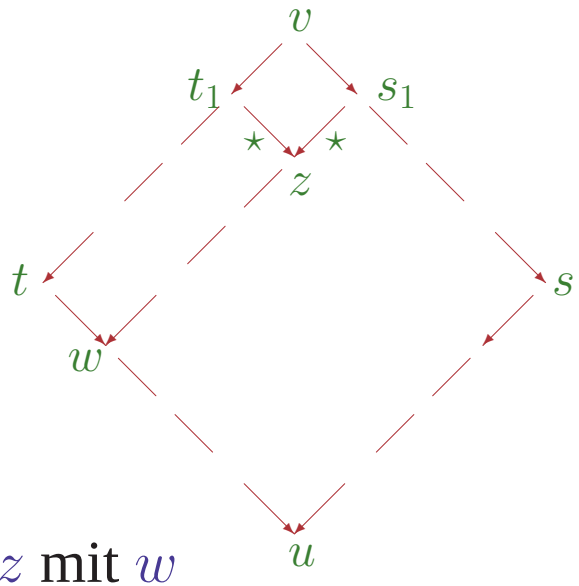
  $n{+}1$: Falls $v \xrightarrow{0} t$ oder $v \xrightarrow{0} s$, dann $v{=}t$ oder $v{=}s$ ✓

  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



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 $v{=}s$ ✓

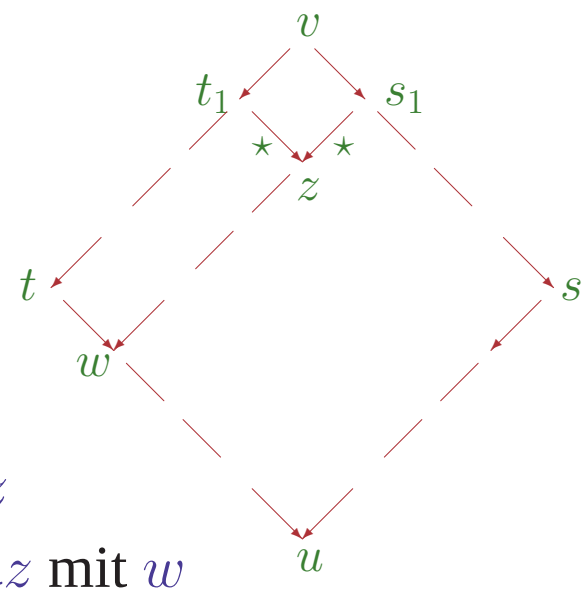
  Ansonsten $v \xrightarrow{r} t_1 \xrightarrow{*} t$ und $v \xrightarrow{r} s_1 \xrightarrow{*} s$

  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 $\boldsymbol{t{\downarrow}s}$ ✓



Church-Rosser Theorem

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

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

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

$$\begin{tikzcd}
(\lambda x\,.\,v)\;w \arrow[r] \arrow[d] \arrow[dr] & v[w/x] \\
(\lambda x\,.\,v)\;w' & \\
(\lambda x\,.\,v')\;w &
\end{tikzcd}$$

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

$$\begin{tikzcd}
(\lambda x\,.\,v)\;w \arrow[r] \arrow[d] \arrow[dr] & v[w/x] \\
(\lambda x\,.\,v)\;w' & \\
(\lambda x\,.\,v')\;w &
\end{tikzcd}$$

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

$$\begin{tikzcd}
(\lambda x\,.\,v)\;w \arrow[r] \arrow[d] \arrow[dr] & v[w/x] \arrow[dr] & \\
 & (\lambda x\,.\,v)\;w' \arrow[r] & v[w'/x] \\
(\lambda x\,.\,v')\;w & &
\end{tikzcd}$$

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

$$\begin{tikzcd}
(\lambda x\,.\,v)\;w \arrow[rr] \arrow[dd] \arrow[dr] & & v[w/x] \arrow[dr] & \\
& (\lambda x\,.\,v)\;w' \arrow[rr] \arrow[dd] & & v[w'/x] \\
(\lambda x\,.\,v')\;w \arrow[dr] & & & \\
& (\lambda x\,.\,v')\;w' & &
\end{tikzcd}$$

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

$$\begin{tikzcd}
(\lambda x\,.\,v)\;w \arrow[rr] \arrow[d] \arrow[dr] & & v[w/x] \arrow[d] \arrow[dr] & \\
& (\lambda x\,.\,v)\;w' \arrow[rr] \arrow[d] & & v[w'/x] \\
(\lambda x\,.\,v')\;w \arrow[rr] \arrow[dr] & & v'[w/x] & \\
& (\lambda x\,.\,v')\;w' \arrow[uur] & &
\end{tikzcd}$$

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

$$
\begin{tikzcd}
(\lambda x\,.\,v)\;w \arrow[rr] \arrow[d] \arrow[dr] & & v[w/x] \arrow[d] \arrow[dr] & \\
& (\lambda x\,.\,v)\;w' \arrow[rr] \arrow[d] & & v[w'/x] \arrow[d, dashed] \\
(\lambda x\,.\,v')\;w \arrow[rr] \arrow[dr] & & v'[w/x] \arrow[dr] & \\
& (\lambda x\,.\,v')\;w' \arrow[rr, dashed] & & v'[w'/x]
\end{tikzcd}
$$

Church-Rosser Theorem

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

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

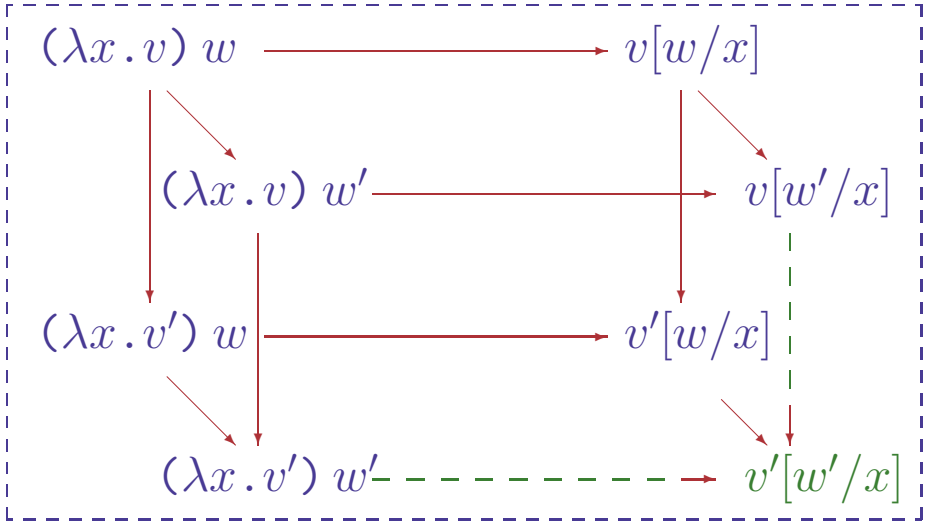
  Es gelte $u \xrightarrow{\beta} t$, $u \xrightarrow{\beta} s$ und $s \neq t$

  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**

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

$$
\begin{tikzcd}
(\lambda x\,.\,v)\;w \arrow[rr] \arrow[dr] \arrow[dd] & & v[w/x] \arrow[dr] \arrow[dd] & \\
& (\lambda x\,.\,v)\;w' \arrow[rr] \arrow[dd] & & v[w'/x] \arrow[dd, dashed] \\
(\lambda x\,.\,v')\;w \arrow[rr] \arrow[dr] & & v'[w/x] \arrow[dr] & \\
& (\lambda x\,.\,v')\;w' \arrow[rr, dashed] & & v'[w'/x]
\end{tikzcd}
$$

- **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**
  - $\overline{\boldsymbol{n}} \equiv \lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}^n\,\mathtt{x}$

Ausdruckskraft

Welche Funktionen können durch typisierbare λ-Terme berechnet werden?

- **Church-Numerals sind typisierbar**
  - $\overline{\boldsymbol{n}} \equiv \lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}^n\,\mathtt{x} \in (\mathtt{X}\to\mathtt{X})\to\mathtt{X}\to\mathtt{X} \equiv \mathbb{N}$

Ausdruckskraft

Welche Funktionen können durch typisierbare λ-Terme berechnet werden?

- **Church-Numerals sind typisierbar**
  - $\overline{n} \equiv \lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}^n\,\mathtt{x} \in (\mathtt{X}\to\mathtt{X})\to\mathtt{X}\to\mathtt{X} \equiv \mathbb{N}$
  - $\mathsf{s} \equiv \lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{n}\ \mathtt{f}\ \mathtt{x}) \in \mathbb{N}\to\mathbb{N}$
  - $\mathsf{add} \equiv \lambda \mathtt{m}.\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{m}\ \mathtt{f}\ (\mathtt{n}\ \mathtt{f}\ \mathtt{x}) \in \mathbb{N}\to\mathbb{N}\to\mathbb{N}$
  - $\mathsf{mul} \equiv \lambda \mathtt{m}.\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{m}\ (\mathtt{n}\ \mathtt{f})\ \mathtt{x} \in \mathbb{N}\to\mathbb{N}\to\mathbb{N}$

Ausdruckskraft

Welche Funktionen können durch typisierbare λ-Terme berechnet werden?

- **Church-Numerals sind typisierbar**
  - $\overline{n} \equiv \lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}^n\,\mathtt{x} \in (\mathtt{X}\to\mathtt{X})\to\mathtt{X}\to\mathtt{X} \equiv \mathbb{N}$
  - $\mathsf{s} \equiv \lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{n}\ \mathtt{f}\ \mathtt{x}) \in \mathbb{N}\to\mathbb{N}$
  - $\mathsf{add} \equiv \lambda \mathtt{m}.\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{m}\ \mathtt{f}\ (\mathtt{n}\ \mathtt{f}\ \mathtt{x}) \in \mathbb{N}\to\mathbb{N}\to\mathbb{N}$
  - $\mathsf{mul} \equiv \lambda \mathtt{m}.\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{m}\ (\mathtt{n}\ \mathtt{f})\ \mathtt{x} \in \mathbb{N}\to\mathbb{N}\to\mathbb{N}$
- **(Simulation der) Paarbildung ist typisierbar**
  - $\langle s, t\rangle \equiv \lambda \mathtt{p}.\ \mathtt{p}\, s\, t$

Ausdruckskraft

Welche Funktionen können durch typisierbare λ-Terme berechnet werden?

- **Church-Numerals sind typisierbar**
  - $\overline{n} \equiv \lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}^n\,\mathtt{x} \in (\mathtt{X}\to\mathtt{X})\to\mathtt{X}\to\mathtt{X} \equiv \mathbb{N}$
  - $\mathsf{s} \equiv \lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{f}\ (\mathtt{n}\ \mathtt{f}\ \mathtt{x}) \in \mathbb{N}\to\mathbb{N}$
  - $\mathsf{add} \equiv \lambda \mathtt{m}.\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{m}\ \mathtt{f}\ (\mathtt{n}\ \mathtt{f}\ \mathtt{x}) \in \mathbb{N}\to\mathbb{N}\to\mathbb{N}$
  - $\mathsf{mul} \equiv \lambda \mathtt{m}.\lambda \mathtt{n}.\lambda \mathtt{f}.\lambda \mathtt{x}.\ \mathtt{m}\ (\mathtt{n}\ \mathtt{f})\ \mathtt{x} \in \mathbb{N}\to\mathbb{N}\to\mathbb{N}$
- **(Simulation der) Paarbildung 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**
  - $\mathsf{T} \equiv \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x} \in \mathtt{X} \to \mathtt{Y} \to \mathtt{X}$
  - $\mathsf{F} \equiv \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y} \in \mathtt{X} \to \mathtt{Y} \to \mathtt{Y}$
  - **if** b **then** s **else** $t \equiv b\,s\,t \in \mathtt{Z}$ für $s \in \mathtt{X}$, $t \in \mathtt{Y}$, $b \in \mathbb{B} \equiv \mathtt{X} \to \mathtt{Y} \to \mathtt{Z}$

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

Ausdruckskraft typisierter Terme hat Grenzen

- **Typisierung boolescher Operatoren problematisch**
  - **T** $\equiv$ $\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x} \in \mathtt{X}\to\mathtt{Y}\to\mathtt{X}$
  - **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 **T** $\in \mathbb{B}$ und **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 $\mathbb{N}$**
  - **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 $\mathtt{X}\to\mathtt{Y}\to\mathtt{Z}\to\mathtt{T} = \mathtt{X}$
  - **PRs**$[base, h]$ $\equiv$ $\lambda \mathtt{n}.\ \mathtt{n}\ h\ base \in (\mathtt{X}\to\mathtt{Y}\to\mathtt{Z}) \to \mathtt{Z}$
    - aber $h \in \mathtt{Y}=\mathbb{N}\to\mathbb{N}$ und $base \in \mathtt{Z}=\mathbb{N}$ verlangt $(\mathbb{N}\to\mathbb{N})\to\mathbb{N}\to\mathbb{N} = \mathbb{N}$

Ausdruckskraft typisierter Terme hat Grenzen

- **Typisierung boolescher Operatoren problematisch**
  - **T** $\equiv$ $\lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x} \in \mathtt{X}\to\mathtt{Y}\to\mathtt{X}$
  - **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 **T** $\in \mathbb{B}$ und **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 $\mathbb{N}$**
  - **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 $\mathtt{X}\to\mathtt{Y}\to\mathtt{Z}\to\mathtt{T} = \mathtt{X}$
  - **PRs**$[base, h]$ $\equiv$ $\lambda \mathtt{n}.\ \mathtt{n}\ h\ base \in (\mathtt{X}\to\mathtt{Y}\to\mathtt{Z}) \to \mathtt{Z}$
    - aber $h \in \mathtt{Y}=\mathbb{N}\to\mathbb{N}$ und $base \in \mathtt{Z}=\mathbb{N}$ verlangt $(\mathbb{N}\to\mathbb{N})\to\mathbb{N}\to\mathbb{N} = \mathbb{N}$

$$\Downarrow$$

Typstruktur zu schwach für praktische Anwendungen