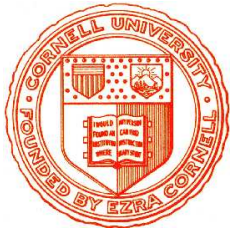


Automatisierte Logik und Programmierung

Einheit 6

Die einfache Typentheorie



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

LOGISCHES SCHLIESSEN ÜBER DATENTYPEN

Koppelung von Term- und Typstruktur

LOGISCHES SCHLIESSEN ÜBER DATENTYPEN

Koppelung von Term- und Typstruktur

- Datentypen sind **Eigenschaften von Termen**

Koppelung von Term- und Typstruktur

- Datentypen sind **Eigenschaften von Termen**

- $1, 2, 3, \dots$ sind natürliche Zahlen
- $x+4, y-z, 4*i, \dots$ sind Ausdrücke über Zahlen

Koppelung von Term- und Typstruktur

- Datentypen sind **Eigenschaften von Termen**

- $1, 2, 3, \dots$ sind natürliche Zahlen
- $x+4, y-z, 4*i, \dots$ sind Ausdrücke über Zahlen
- $(4, 5), (7, 8), (3-2, 4*6), \dots$ sind Paare von Zahlen
- $[1; 4; 7; 8; 5]$ ist eine Liste von Zahlen

Koppelung von Term- und Typstruktur

● Datentypen sind **Eigenschaften von Termen**

- $1, 2, 3, \dots$ sind natürliche Zahlen
- $x+4, y-z, 4*i, \dots$ sind Ausdrücke über Zahlen
- $(4, 5), (7, 8), (3-2, 4*6), \dots$ sind Paare von Zahlen
- $[1; 4; 7; 8; 5]$ ist eine Liste von Zahlen
- $\lambda x. 4*x$ ist eine Funktion auf Zahlen

Koppelung von Term- und Typstruktur

- Datentypen sind **Eigenschaften von Termen**

- $1, 2, 3, \dots$ sind natürliche Zahlen
- $x+4, y-z, 4*i, \dots$ sind Ausdrücke über Zahlen
- $(4, 5), (7, 8), (3-2, 4*6), \dots$ sind Paare von Zahlen
- $[1; 4; 7; 8; 5]$ ist eine Liste von Zahlen
- $\lambda x. 4*x$ ist eine Funktion auf Zahlen

- Datentypen besitzen **Struktur**

Koppelung von Term- und Typstruktur

- Datentypen sind **Eigenschaften von Termen**

- $1, 2, 3, \dots$ sind natürliche Zahlen
- $x+4, y-z, 4*i, \dots$ sind Ausdrücke über Zahlen
- $(4, 5), (7, 8), (3-2, 4*6), \dots$ sind Paare von Zahlen
- $[1; 4; 7; 8; 5]$ ist eine Liste von Zahlen
- $\lambda x. 4*x$ ist eine Funktion auf Zahlen

- Datentypen besitzen **Struktur**

- $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{B}, \text{String} \dots$ sind einfach strukturierte Typen

Koppelung von Term- und Typstruktur

● Datentypen sind **Eigenschaften von Termen**

- $1, 2, 3, \dots$ sind natürliche Zahlen
- $x+4, y-z, 4*i, \dots$ sind Ausdrücke über Zahlen
- $(4, 5), (7, 8), (3-2, 4*6), \dots$ sind Paare von Zahlen
- $[1; 4; 7; 8; 5]$ ist eine Liste von Zahlen
- $\lambda x. 4*x$ ist eine Funktion auf Zahlen

● Datentypen besitzen **Struktur**

- $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{B}, \text{String} \dots$ sind einfach strukturierte Typen
- Datentypen können zusammengesetzt werden
 - $\lambda x. 4*x$ besitzt den Typ $\mathbb{N} \rightarrow \mathbb{N}$
 - $(4, 5)$ ist vom Typ $\mathbb{N} \times \mathbb{N}$
 - $[1; 4; 7; 8; 5]$ gehört zum Typ $\mathbb{N} \text{ list}$

Koppelung von Term- und Typstruktur

● Datentypen sind **Eigenschaften von Termen**

- $1, 2, 3, \dots$ sind natürliche Zahlen
- $x+4, y-z, 4*i, \dots$ sind Ausdrücke über Zahlen
- $(4, 5), (7, 8), (3-2, 4*6), \dots$ sind Paare von Zahlen
- $[1; 4; 7; 8; 5]$ ist eine Liste von Zahlen
- $\lambda x. 4*x$ ist eine Funktion auf Zahlen

● Datentypen besitzen **Struktur**

- $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \mathbb{B}, \text{String} \dots$ sind einfach strukturierte Typen
- Datentypen können zusammengesetzt werden
 - $\lambda x. 4*x$ besitzt den Typ $\mathbb{N} \rightarrow \mathbb{N}$
 - $(4, 5)$ ist vom Typ $\mathbb{N} \times \mathbb{N}$
 - $[1; 4; 7; 8; 5]$ gehört zum Typ $\mathbb{N} \text{ list}$
- Datentypen können Programmeigenschaften charakterisieren
 - $\{f : \mathbb{N} \rightarrow \mathbb{N} \mid \forall x : \mathbb{N}. f(x)^2 \leq x < (f(x)+1)^2\}$ spezifiziert $\lambda x. \lfloor \sqrt{x} \rfloor$

DATENTYPKALKÜL: WAS IST NEU?

Logisches Schließen über Programmeigenschaften

Logisches Schließen über Programmeigenschaften

- **Mehr Struktur in Beschreibung von Typen**
 - Präzisierung der prädikatenlogischen Bereiche
 - Typausdrücke ersetzen Bereichssymbole
 - Interpretation von Typausdrücken i.w. festgelegt

Logisches Schließen über Programmeigenschaften

- **Mehr Struktur in Beschreibung von Typen**

- Präzisierung der prädikatenlogischen Bereiche
- Typausdrücke ersetzen Bereichssymbole
- Interpretation von Typausdrücken i.w. festgelegt

- **Regeln zur Typisierung von Termen**

- Führt zu Einschränkung auf ‘interpretierbare’ Ausdrücke
 - λ -Ausdrücke müssen (sinnvolle) Funktionen darstellen
 - Keine unkontrollierte Rekursion oder Selbstanwendung
- Minimale Begrenzung der Ausdruckskraft

Logisches Schließen über Programmeigenschaften

- **Mehr Struktur in Beschreibung von Typen**

- Präzisierung der prädikatenlogischen Bereiche
- Typausdrücke ersetzen Bereichssymbole
- Interpretation von Typausdrücken i.w. festgelegt

- **Regeln zur Typisierung von Termen**

- Führt zu Einschränkung auf ‘interpretierbare’ Ausdrücke
 - λ -Ausdrücke müssen (sinnvolle) Funktionen darstellen
 - Keine unkontrollierte Rekursion oder Selbstanwendung
- Minimale Begrenzung der Ausdruckskraft



Präzisierung der Semantik formaler Ausdrücke

- **Getypter λ -Kalkül**

- Typen sind Teil der Syntax von λ -Termen:

- **Getypter λ -Kalkül**

- Typen sind Teil der Syntax von λ -Termen: $\lambda f^{S \rightarrow T} . \lambda x^S . f^{S \rightarrow T} x^S$

● Getypter λ -Kalkül

- Typen sind Teil der Syntax von λ -Termen: $\lambda f^{S \rightarrow T} . \lambda x^S . f^{S \rightarrow T} x^S$
- Datentyp aller Teilausdrücke leicht zu ermitteln
- Untypisierbare Terme sind nicht formulierbar

- **Getypter λ -Kalkül**

- Typen sind Teil der Syntax von λ -Termen: $\lambda f^{S \rightarrow T} . \lambda x^S . f^{S \rightarrow T} x^S$
- Datentyp aller Teilausdrücke leicht zu ermitteln
- Untypisierbare Terme sind nicht formulierbar

- **Typentheorie**

- Terme und Typen sind unabhängige Konzepte

● Getypter λ -Kalkül

- Typen sind Teil der Syntax von λ -Termen: $\lambda f^{S \rightarrow T} . \lambda x^S . f^{S \rightarrow T} x^S$
- Datentyp aller Teilausdrücke leicht zu ermitteln
- Untypisierbare Terme sind nicht formulierbar

● Typentheorie

- Terme und Typen sind unabhängige Konzepte
- Typen werden Termen zugeordnet:

● Getypter λ -Kalkül

- Typen sind Teil der Syntax von λ -Termen: $\lambda f^{S \rightarrow T} . \lambda x^S . f^{S \rightarrow T} x^S$
- Datentyp aller Teilausdrücke leicht zu ermitteln
- Untypisierbare Terme sind nicht formulierbar

● Typentheorie

- Terme und Typen sind unabhängige Konzepte
 - Typen werden Termen zugeordnet: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow S \rightarrow T$
- Typisierung ist losgelöst von der Formulierung des Terms

● Getypter λ -Kalkül

- Typen sind Teil der Syntax von λ -Termen: $\lambda f^{S \rightarrow T} . \lambda x^S . f^{S \rightarrow T} x^S$
- Datentyp aller Teilausdrücke leicht zu ermitteln
- Untypisierbare Terme sind nicht formulierbar

● Typentheorie

- Terme und Typen sind unabhängige Konzepte
- Typen werden Termen zugeordnet: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow S \rightarrow T$
Typisierung ist losgelöst von der Formulierung des Terms
- Mehr Freiheit in der Formulierung von λ -Termen
- Größerer Aufwand bei (nachträglicher) Bestimmung des Typs

● Getypter λ -Kalkül

- Typen sind Teil der Syntax von λ -Termen: $\lambda f^{S \rightarrow T} . \lambda x^S . f^{S \rightarrow T} x^S$
- Datentyp aller Teilausdrücke leicht zu ermitteln
- Untypisierbare Terme sind nicht formulierbar

● Typentheorie

- Terme und Typen sind unabhängige Konzepte
- Typen werden Termen zugeordnet: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow S \rightarrow T$
 Typisierung ist losgelöst von der Formulierung des Terms
- Mehr Freiheit in der Formulierung von λ -Termen
- Größerer Aufwand bei (nachträglicher) Bestimmung des Typs



Kalkül zum Schließen über Typzugehörigkeit

CHARAKTERISTIKA EINES DATENTYPS

Syntaktische Beschreibung von Mengen

CHARAKTERISTIKA EINES DATENTYPS

Syntaktische Beschreibung von Mengen

- Direkte Beschreibung einfacher Datentypen

Syntaktische Beschreibung von Mengen

- **Direkte Beschreibung einfacher Datentypen**

- Festlegung eines Namens

$\mathbb{N}, \mathbb{Z}, \mathbb{B} \dots$

Syntaktische Beschreibung von Mengen

● Direkte Beschreibung einfacher Datentypen

- Festlegung eines Namens $\mathbb{N}, \mathbb{Z}, \mathbb{B} \dots$
- Beschreibung der Basiselemente $0, 1, 2, 3, \dots, 0, 1, -1, 2, \dots, \text{tt}, \text{ff}$

Syntaktische Beschreibung von Mengen

● Direkte Beschreibung einfacher Datentypen

- Festlegung eines Namens $\mathbb{N}, \mathbb{Z}, \mathbb{B} \dots$
- Beschreibung der Basiselemente $0, 1, 2, 3, \dots, 0, 1, -1, 2, \dots, \text{tt}, \text{ff}$
- Beschreibung von Operatoren auf Elementen $-x, x+y, x-y, x*y, \dots$

Syntaktische Beschreibung von Mengen

- **Direkte Beschreibung einfacher Datentypen**

- Festlegung eines Namens $\mathbb{N}, \mathbb{Z}, \mathbb{B} \dots$
- Beschreibung der Basiselemente $0, 1, 2, 3, \dots, 0, 1, -1, 2, \dots, \text{tt}, \text{ff}$
- Beschreibung von Operatoren auf Elementen $-x, x+y, x-y, x*y, \dots$

- **Konstruktoren für strukturierte Datentypen**

- Typkonstruktoren: $T \times T', T \rightarrow T', T \text{ list}, \dots$

Syntaktische Beschreibung von Mengen

● Direkte Beschreibung einfacher Datentypen

- Festlegung eines Namens $\mathbb{N}, \mathbb{Z}, \mathbb{B} \dots$
- Beschreibung der Basiselemente $0, 1, 2, 3, \dots, 0, 1, -1, 2, \dots, \text{tt}, \text{ff}$
- Beschreibung von Operatoren auf Elementen $-x, x+y, x-y, x*y, \dots$

● Konstruktoren für strukturierte Datentypen

- Typkonstruktoren: $T \times T', T \rightarrow T', T \text{ list}, \dots$
- Konstruktion kanonischer Elemente: $(a, b), \lambda x. t, a_1 :: l, \dots$

Syntaktische Beschreibung von Mengen

● Direkte Beschreibung einfacher Datentypen

- Festlegung eines Namens $\mathbb{N}, \mathbb{Z}, \mathbb{B} \dots$
- Beschreibung der Basiselemente $0, 1, 2, 3, \dots, 0, 1, -1, 2, \dots, \text{tt}, \text{ff}$
- Beschreibung von Operatoren auf Elementen $-x, x+y, x-y, x*y, \dots$

● Konstruktoren für strukturierte Datentypen

- Typkonstruktoren: $T \times T', T \rightarrow T', T \text{ list}, \dots$
- Konstruktion kanonischer Elemente: $(a, b), \lambda x. t, a_1 :: l, \dots$
- Nichtkanonische Operatoren auf Elementen: $p.1, f(a), \text{hd}(l), \dots$

Syntaktische Beschreibung von Mengen

● Direkte Beschreibung einfacher Datentypen

- Festlegung eines Namens $\mathbb{N}, \mathbb{Z}, \mathbb{B} \dots$
- Beschreibung der Basiselemente $0, 1, 2, 3, \dots, 0, 1, -1, 2, \dots, \text{tt}, \text{ff}$
- Beschreibung von Operatoren auf Elementen $-x, x+y, x-y, x*y, \dots$

● Konstruktoren für strukturierte Datentypen

- Typkonstruktoren: $T \times T', T \rightarrow T', T \text{ list}, \dots$
- Konstruktion kanonischer Elemente: $(a, b), \lambda x. t, a_1 :: l, \dots$
- Nichtkanonische Operatoren auf Elementen: $p.1, f(a), \text{hd}(l), \dots$

● Grundbestandteile der beschreibenden Syntax

- Typ-Ausdruck + kanonische Terme + nichtkanonische Terme

Minimal nötige Typkonstrukte für den λ -Kalkül

Minimal nötige Typkonstrukte für den λ -Kalkül

- **Alphabete** für erlaubte Symbole
 - $\mathcal{V}$: Variablensymbole
 - $\mathcal{T}$: Bereichssymbole (**Typen**)

Minimal nötige Typkonstrukte für den λ -Kalkül

- **Alphabete** für erlaubte Symbole
 - $\mathcal{V}$: Variablensymbole
 - $\mathcal{T}$: Bereichssymbole (Typen)
- **Typ-Ausdrücke** (Typen)
 - Typsymbole $T \in \mathcal{T}$
 - $S \rightarrow T$ und (T) , wobei S und T Typen

Minimal nötige Typkonstrukte für den λ -Kalkül

- **Alphabete** für erlaubte Symbole
 - $\mathcal{V}$: Variablensymbole
 - $\mathcal{T}$: Bereichssymbole (Typen)
- **Typ-Ausdrücke** (Typen)
 - Typsymbole $T \in \mathcal{T}$
 - $S \rightarrow T$ und (T) , wobei S und T Typen
- **Objekt-Ausdrücke** (λ -Terme)
 - Variablen $x \in \mathcal{V}$
 - $\lambda x.t$, $f t$ und (t) , wobei $x \in \mathcal{V}$ und t und f λ -Terme

Minimal nötige Typkonstrukte für den λ -Kalkül

- **Alphabete** für erlaubte Symbole

- $\mathcal{V}$: Variablensymbole
- $\mathcal{T}$: Bereichssymbole (Typen)

- **Typ-Ausdrücke** (Typen)

- Typsymbole $T \in \mathcal{T}$
- $S \rightarrow T$ und (T) , wobei S und T Typen

- **Objekt-Ausdrücke** (λ -Terme)

- Variablen $x \in \mathcal{V}$
- $\lambda x.t$, $f t$ und (t) , wobei $x \in \mathcal{V}$ und t und f λ -Terme

- **Prioritäten**

- Applikation bindet stärker als λ -Abstraktion
- Applikation ist links-assoziativ:

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

Minimal nötige Typkonstrukte für den λ -Kalkül

- **Alphabete** für erlaubte Symbole

- $\mathcal{V}$: Variablensymbole
- $\mathcal{T}$: Bereichssymbole (Typen)

- **Typ-Ausdrücke** (Typen)

- Typsymbole $T \in \mathcal{T}$
- $S \rightarrow T$ und (T) , wobei S und T Typen

- **Objekt-Ausdrücke** (λ -Terme)

- Variablen $x \in \mathcal{V}$
- $\lambda x.t$, $f t$ und (t) , wobei $x \in \mathcal{V}$ und t und f λ -Terme

- **Prioritäten**

- Applikation bindet stärker als λ -Abstraktion
- Applikation ist links-assoziativ:
- $\rightarrow$ ist rechtsassoziativ:

$$f \ t_1 \ t_2 \hat{=} (f \ t_1) \ t_2$$
$$T_1 \rightarrow T_2 \rightarrow T_3 \hat{=} T_1 \rightarrow (T_2 \rightarrow T_3)$$

- **Operationale Semantik für λ -Terme**
 - Bestimmung des Wertes durch Reduktion

- **Operationale Semantik für λ -Terme**
 - Bestimmung des Wertes durch Reduktion
- **Zuordnung von Typen zu λ -Termen**
 - $t \in T \equiv \iota(t)$ ist ein Element von $\iota(T)$

- **Operationale Semantik für λ -Terme**
 - Bestimmung des Wertes durch Reduktion
- **Zuordnung von Typen zu λ -Termen**
 - $t \in T \equiv \iota(t)$ ist ein Element von $\iota(T)$
- **Interpretiere $\rightarrow$ als Funktionenraum**
 - $\iota(S \rightarrow T) \hat{=} \text{Menge der Funktionen von } \iota(S) \text{ nach } \iota(T)$

- **Operationale Semantik für λ -Terme**
 - Bestimmung des Wertes durch Reduktion
- **Zuordnung von Typen zu λ -Termen**
 - $t \in T \equiv \iota(t)$ ist ein Element von $\iota(T)$
- **Interpretiere $\rightarrow$ als Funktionenraum**
 - $\iota(S \rightarrow T) \hat{=} \text{Menge der Funktionen von } \iota(S) \text{ nach } \iota(T)$
- **Typzugehörigkeit: Koppele Typ- und Termaufbau**

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

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

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

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

Typisierung von $\lambda f. \lambda x. f x$

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

Typisierung von $\lambda f. \lambda x. f$ 

für ein beliebiges S

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

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

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

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

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

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

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

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

TYPZUGEHÖRIGKEIT: VORBETRACHTUNGEN

- Der Typ eines λ -Terms ist nicht eindeutig

- Der Typ eines λ -Terms ist nicht eindeutig

- Naheliegend: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow (S \rightarrow T)$

- Der Typ eines λ -Terms ist nicht eindeutig

- Naheliegend: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow (S \rightarrow T)$
- Auch möglich $\lambda f . \lambda x . f x \in (S \rightarrow (T \rightarrow S)) \rightarrow (S \rightarrow (T \rightarrow S))$

- Der Typ eines λ -Terms ist nicht eindeutig

- Naheliegend: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow (S \rightarrow T)$
- Auch möglich $\lambda f . \lambda x . f x \in (S \rightarrow (T \rightarrow S)) \rightarrow (S \rightarrow (T \rightarrow S))$
 $\lambda f . \lambda x . f x \in (S \rightarrow S) \rightarrow (S \rightarrow S)$

- Der Typ eines λ -Terms ist nicht eindeutig

- Naheliegend: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow (S \rightarrow T)$
- Auch möglich $\lambda f . \lambda x . f x \in (S \rightarrow (T \rightarrow S)) \rightarrow (S \rightarrow (T \rightarrow S))$
 $\lambda f . \lambda x . f x \in (S \rightarrow S) \rightarrow (S \rightarrow S)$
- $(S \rightarrow T) \rightarrow (S \rightarrow T)$ ist die allgemeinste Form (prinzipielles Typschema)

- **Der Typ eines λ -Terms ist nicht eindeutig**

- Naheliegend: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow (S \rightarrow T)$
- Auch möglich $\lambda f . \lambda x . f x \in (S \rightarrow (T \rightarrow S)) \rightarrow (S \rightarrow (T \rightarrow S))$
 $\lambda f . \lambda x . f x \in (S \rightarrow S) \rightarrow (S \rightarrow S)$
- $(S \rightarrow T) \rightarrow (S \rightarrow T)$ ist die allgemeinste Form (**prinzipielles Typschema**)

- **Nicht jeder λ -Term kann typisierbar sein**

- Sonst gäbe es eine einfache Semantik des λ -Kalküls

- **Der Typ eines λ -Terms ist nicht eindeutig**

- Naheliegend: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow (S \rightarrow T)$
- Auch möglich $\lambda f . \lambda x . f x \in (S \rightarrow (T \rightarrow S)) \rightarrow (S \rightarrow (T \rightarrow S))$
 $\lambda f . \lambda x . f x \in (S \rightarrow S) \rightarrow (S \rightarrow S)$
- $(S \rightarrow T) \rightarrow (S \rightarrow T)$ ist die allgemeinste Form (**prinzipielles Typschema**)

- **Nicht jeder λ -Term kann typisierbar sein**

- Sonst gäbe es eine einfache Semantik des λ -Kalküls
- Beispiel: $\lambda x . x x$

TYPZUGEHÖRIGKEIT: VORBETRACHTUNGEN

- **Der Typ eines λ -Terms ist nicht eindeutig**

- Naheliegend: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow (S \rightarrow T)$
- Auch möglich $\lambda f . \lambda x . f x \in (S \rightarrow (T \rightarrow S)) \rightarrow (S \rightarrow (T \rightarrow S))$
 $\lambda f . \lambda x . f x \in (S \rightarrow S) \rightarrow (S \rightarrow S)$
- $(S \rightarrow T) \rightarrow (S \rightarrow T)$ ist die allgemeinste Form (**prinzipielles Typschema**)

- **Nicht jeder λ -Term kann typisierbar sein**

- Sonst gäbe es eine einfache Semantik des λ -Kalküls
- Beispiel: $\lambda x . x \underbrace{x}_S$ für ein beliebiges **S**

TYPZUGEHÖRIGKEIT: VORBETRACHTUNGEN

- Der Typ eines λ -Terms ist nicht eindeutig

- Naheliegend: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow (S \rightarrow T)$
- Auch möglich $\lambda f . \lambda x . f x \in (S \rightarrow (T \rightarrow S)) \rightarrow (S \rightarrow (T \rightarrow S))$
 $\lambda f . \lambda x . f x \in (S \rightarrow S) \rightarrow (S \rightarrow S)$
- $(S \rightarrow T) \rightarrow (S \rightarrow T)$ ist die allgemeinste Form (prinzipielles Typschema)

- Nicht jeder λ -Term kann typisierbar sein

- Sonst gäbe es eine einfache Semantik des λ -Kalküls
- Beispiel: $\lambda x . \underbrace{x}_{S \rightarrow T} \underbrace{x}_S$ für beliebige S und T

TYPZUGEHÖRIGKEIT: VORBETRACHTUNGEN

- Der Typ eines λ -Terms ist **nicht eindeutig**

- Naheliegend: $\lambda f . \lambda x . f x \in (S \rightarrow T) \rightarrow (S \rightarrow T)$
- Auch möglich $\lambda f . \lambda x . f x \in (S \rightarrow (T \rightarrow S)) \rightarrow (S \rightarrow (T \rightarrow S))$
 $\lambda f . \lambda x . f x \in (S \rightarrow S) \rightarrow (S \rightarrow S)$
- $(S \rightarrow T) \rightarrow (S \rightarrow T)$ ist die allgemeinste Form (**prinzipielles Typschema**)

- **Nicht jeder λ -Term kann typisierbar sein**

- Sonst gäbe es eine einfache Semantik des λ -Kalküls
- Beispiel: $\lambda x . \underbrace{x}_{S \rightarrow T} \underbrace{x}_S$ für beliebige **S** und **T**
- Die Gleichung **S** = **S** $\rightarrow$ **T** hat keine “natürliche” Lösung

- **Reduktion erhält Typzugehörigkeit**

- $\iota((\lambda x.u) t)$ in $\iota(T)$ impliziert $\iota(t)$ in $\iota(S)$ und $\iota(\lambda x.u)$ in $\iota(S \rightarrow T)$

● Reduktion erhält Typzugehörigkeit

- $\iota((\lambda x.u) t)$ in $\iota(T)$ impliziert $\iota(t)$ in $\iota(S)$ und $\iota(\lambda x.u)$ in $\iota(S \rightarrow T)$
- $\iota(\lambda x.u)$ in $\iota(S) \rightarrow \iota(T)$ verlangt $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$

● Reduktion erhält Typzugehörigkeit

- $\iota((\lambda x.u) t)$ in $\iota(T)$ impliziert $\iota(t)$ in $\iota(S)$ und $\iota(\lambda x.u)$ in $\iota(S \rightarrow T)$
- $\iota(\lambda x.u)$ in $\iota(S) \rightarrow \iota(T)$ verlangt $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$
- $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$ und $\iota(t)$ in $\iota(S)$ impliziert $\iota(u[t/x])$ in $\iota(T)$

● Reduktion erhält Typzugehörigkeit

- $\iota((\lambda x.u) t)$ in $\iota(T)$ impliziert $\iota(t)$ in $\iota(S)$ und $\iota(\lambda x.u)$ in $\iota(S \rightarrow T)$
- $\iota(\lambda x.u)$ in $\iota(S) \rightarrow \iota(T)$ verlangt $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$
- $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$ und $\iota(t)$ in $\iota(S)$ impliziert $\iota(u[t/x])$ in $\iota(T)$

● Die Umkehrung gilt nicht

- Typisierbarkeit von $u[t/x]$ impliziert nicht Typisierbarkeit von $(\lambda x.u) t$

● Reduktion erhält Typzugehörigkeit

- $\iota((\lambda x.u) t)$ in $\iota(T)$ impliziert $\iota(t)$ in $\iota(S)$ und $\iota(\lambda x.u)$ in $\iota(S \rightarrow T)$
- $\iota(\lambda x.u)$ in $\iota(S) \rightarrow \iota(T)$ verlangt $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$
- $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$ und $\iota(t)$ in $\iota(S)$ impliziert $\iota(u[t/x])$ in $\iota(T)$

● Die Umkehrung gilt nicht

- Typisierbarkeit von $u[t/x]$ impliziert nicht Typisierbarkeit von $(\lambda x.u) t$
da x eventuell in u nicht frei vorkommt

● Reduktion erhält Typzugehörigkeit

- $\iota((\lambda x.u) t)$ in $\iota(T)$ impliziert $\iota(t)$ in $\iota(S)$ und $\iota(\lambda x.u)$ in $\iota(S \rightarrow T)$
- $\iota(\lambda x.u)$ in $\iota(S) \rightarrow \iota(T)$ verlangt $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$
- $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$ und $\iota(t)$ in $\iota(S)$ impliziert $\iota(u[t/x])$ in $\iota(T)$

● Die Umkehrung gilt nicht

- Typisierbarkeit von $u[t/x]$ impliziert nicht Typisierbarkeit von $(\lambda x.u) t$
da x eventuell in u nicht frei vorkommt
- Beispiel: $(\lambda f . \lambda y . y) (\lambda x . x x)$ nicht typisierbar,
da $(\lambda x . x x)$ nicht typisierbar

● Reduktion erhält Typzugehörigkeit

- $\iota((\lambda x.u) t)$ in $\iota(T)$ impliziert $\iota(t)$ in $\iota(S)$ und $\iota(\lambda x.u)$ in $\iota(S \rightarrow T)$
- $\iota(\lambda x.u)$ in $\iota(S) \rightarrow \iota(T)$ verlangt $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$
- $\iota(u)$ in $\iota(T)$ für $\iota(x)$ in $\iota(S)$ und $\iota(t)$ in $\iota(S)$ impliziert $\iota(u[t/x])$ in $\iota(T)$

● Die Umkehrung gilt nicht

- Typisierbarkeit von $u[t/x]$ impliziert nicht Typisierbarkeit von $(\lambda x.u) t$
da x eventuell in u nicht frei vorkommt
- Beispiel: $(\lambda f . \lambda y . y) (\lambda x . x x)$ nicht typisierbar,
da $(\lambda x . x x)$ nicht typisierbar
aber $\lambda y . y \in T \rightarrow T$

SEMANTIK DER TYPENTHEORIE PRÄZISIERT

Urteile ersetzen modelltheoretische Semantik

Urteile ersetzen modelltheoretische Semantik

- Definiere Gleichheit von λ -Termen wie zuvor
 - Reduktions- und Konversionsregeln definieren, wann $t \xrightarrow{*} u$ gilt

Urteile ersetzen modelltheoretische Semantik

- Definiere **Gleichheit von λ -Termen** wie zuvor
 - Reduktions- und Konversionsregeln definieren, wann $t \xrightarrow{*} u$ gilt
- Definiere **Typzugehörigkeitsrelation $t \in T$**
 - Urteile definieren, wann diese Relation erfüllt ist

Urteile ersetzen modelltheoretische Semantik

- Definiere **Gleichheit von λ -Termen** wie zuvor
 - Reduktions- und Konversionsregeln definieren, wann $t \xrightarrow{*} u$ gilt
- Definiere **Typzugehörigkeitsrelation $t \in T$**
 - Urteile definieren, wann diese Relation erfüllt ist
 - $x \in T$ falls $x \in \mathcal{V}$ und T ein Typ

Urteile ersetzen modelltheoretische Semantik

- Definiere **Gleichheit von λ -Termen** wie zuvor
 - Reduktions- und Konversionsregeln definieren, wann $t \xrightarrow{*} u$ gilt
- Definiere **Typzugehörigkeitsrelation $t \in T$**
 - Urteile definieren, wann diese Relation erfüllt ist
 - $x \in T$ falls $x \in \mathcal{V}$ und T ein Typ
 - $\lambda x. t \in S \rightarrow T$ falls $t \in T$ aus $x \in S$ folgt

Urteile ersetzen modelltheoretische Semantik

- Definiere **Gleichheit von λ -Termen** wie zuvor
 - Reduktions- und Konversionsregeln definieren, wann $t \xrightarrow{*} u$ gilt
- Definiere **Typzugehörigkeitsrelation $t \in T$**
 - Urteile definieren, wann diese Relation erfüllt ist
 - $x \in T$ falls $x \in \mathcal{V}$ und T ein Typ
 - $\lambda x. t \in S \rightarrow T$ falls $t \in T$ aus $x \in S$ folgt
 - $f t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S

Urteile ersetzen modelltheoretische Semantik

- Definiere **Gleichheit von λ -Termen** wie zuvor
 - Reduktions- und Konversionsregeln definieren, wann $t \xrightarrow{*} u$ gilt
- Definiere **Typzugehörigkeitsrelation $t \in T$**
 - Urteile definieren, wann diese Relation erfüllt ist
 - $x \in T$ falls $x \in \mathcal{V}$ und T ein Typ
 - $\lambda x. t \in S \rightarrow T$ falls $t \in T$ aus $x \in S$ folgt
 - $f t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S
 - $(t) \in T$ falls $t \in T$
 - $t \in (T)$ falls $t \in T$

Urteile ersetzen modelltheoretische Semantik

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

Urteile ersetzen modelltheoretische Semantik

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

TYPZUGEHÖRIGKEIT HARMONIERT MIT BERECHNUNG

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

TYPZUGEHÖRIGKEIT HARMONIERT MIT BERECHNUNG

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

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

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

1. Reduktion erhält den Typ: aus $t \xrightarrow{\beta} t'$ folgt $t \in T \Leftrightarrow t' \in T$
 - Induktion über die Termstruktur von t

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

1. Reduktion erhält den Typ: aus $t \xrightarrow{\beta} t'$ folgt $t \in T \Leftrightarrow t' \in T$
 - Induktion über die Termstruktur von t
 - $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

1. Reduktion erhält den Typ: aus $t \xrightarrow{\beta} t'$ folgt $t \in T \Leftrightarrow t' \in T$
- Induktion über die Termstruktur von t
 - $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓
 - $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

- $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓
- $t \equiv \lambda x . u$: Dann $t' \equiv \lambda x . u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

TYPZUGEHÖRIGKEIT HARMONIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

- $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓
- $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

- $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓
- $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ 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 $t' \in T$

✓

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

• $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓

• $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$

Gleiches Argument für $t \in T$ wenn $t' \in T$

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

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

• $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓

• $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$

Gleiches Argument für $t \in T$ wenn $t' \in T$

• $t \equiv (\lambda x. u) v$, $t' \equiv u[v/x]$: Aus $t \in T$ folgt $\lambda x. u \in S \rightarrow T$ und $v \in S$,
also $u \in T$, wenn $x \in S$. Damit $u[v/x] \in T$. ✓

TYPZUGEHÖRIGKEIT HARMONIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

• $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓

• $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$

Gleiches Argument für $t \in T$ wenn $t' \in T$

• $t \equiv (\lambda x. u) v$, $t' \equiv u[v/x]$: Aus $t \in T$ folgt $\lambda x. u \in S \rightarrow T$ 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 $(\lambda x. u) v \in T$. ✓

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

• $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓

• $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$

Gleiches Argument für $t \in T$ wenn $t' \in T$

• $t \equiv (\lambda x. u) v$, $t' \equiv u[v/x]$: Aus $t \in T$ folgt $\lambda x. u \in S \rightarrow T$ 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 $f \xrightarrow{\beta} f'$.

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

• $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓

• $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$

Gleiches Argument für $t \in T$ wenn $t' \in T$

• $t \equiv (\lambda x. u) v$, $t' \equiv u[v/x]$: Aus $t \in T$ folgt $\lambda x. u \in S \rightarrow T$ 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. ✓

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

• $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓

• $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$

Gleiches Argument für $t \in T$ wenn $t' \in T$

• $t \equiv (\lambda x. u) v$, $t' \equiv u[v/x]$: Aus $t \in T$ folgt $\lambda x. u \in S \rightarrow T$ und $v \in S$,
also $u \in T$, wenn $x \in S$. Damit $u[v/x] \in T$. ✓

Aus $t' \in T$ folgt $v \in S$ und $u \in T$ wenn $x \in S$. Damit $(\lambda x. u) v \in T$. ✓

• $t \equiv f u$: Dann $t' \equiv f u'$ und $u \xrightarrow{\beta} u'$ oder $t' \equiv f' u$ und $f \xrightarrow{\beta} f'$. ✓

Gleiches Argument wie zuvor. ✓

Achtung: t und t' müssen typisierbar sein

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

• $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓

• $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$

Gleiches Argument für $t \in T$ wenn $t' \in T$

• $t \equiv (\lambda x. u) v$, $t' \equiv u[v/x]$: Aus $t \in T$ folgt $\lambda x. u \in S \rightarrow T$ und $v \in S$,
also $u \in T$, wenn $x \in S$. Damit $u[v/x] \in T$. ✓

Aus $t' \in T$ folgt $v \in S$ und $u \in T$ wenn $x \in S$. Damit $(\lambda x. u) v \in T$. ✓

• $t \equiv f u$: Dann $t' \equiv f u'$ und $u \xrightarrow{\beta} u'$ oder $t' \equiv f' u$ und $f \xrightarrow{\beta} f'$. ✓

Gleiches Argument wie zuvor. ✓

Achtung: t und t' müssen typisierbar sein

2. Aus $t \xrightarrow{*} t'$ folgt $t \in T \Leftrightarrow t' \in T$ (Induktion über Reduktionslänge)

TYPZUGEHÖRIGKEIT HARMONISIERT MIT BERECHNUNG

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

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

– Induktion über die Termstruktur von t

• $t \in \mathcal{V}$: t ist in Normalform, also $t' \in \mathcal{V}$. Damit $t, t' \in T$ für beliebige T ✓

• $t \equiv \lambda x. u$: Dann $t' \equiv \lambda x. u'$ mit $u \xrightarrow{\beta} u'$.

Aus $t \in T$ folgt $T \equiv S_1 \rightarrow S_2$ mit $u \in S_2$ wenn $x \in S_1$.

Per Induktionsannahme gilt $u' \in S_2$ wenn $x \in S_1$, also $t' \in T$

Gleiches Argument für $t \in T$ wenn $t' \in T$

• $t \equiv (\lambda x. u) v$, $t' \equiv u[v/x]$: Aus $t \in T$ folgt $\lambda x. u \in S \rightarrow T$ und $v \in S$,
also $u \in T$, wenn $x \in S$. Damit $u[v/x] \in T$. ✓

Aus $t' \in T$ folgt $v \in S$ und $u \in T$ wenn $x \in S$. Damit $(\lambda x. u) v \in T$. ✓

• $t \equiv f u$: Dann $t' \equiv f u'$ und $u \xrightarrow{\beta} u'$ oder $t' \equiv f' u$ und $f \xrightarrow{\beta} f'$. ✓

Gleiches Argument wie zuvor. ✓

Achtung: t und t' müssen typisierbar sein

2. Aus $t \xrightarrow{*} t'$ folgt $t \in T \Leftrightarrow t' \in T$ (Induktion über Reduktionslänge)

3. t und t' sind semantisch gleich, wenn $t \xrightarrow{*} u$ und $t' \xrightarrow{*} u$

- Bestimme den Typ eines beliebigen Terms

- **Bestimme den Typ eines beliebigen Terms**
 - Vollautomatisch für einfache Typstrukturen (ML, Haskell, Java, ...)
 - Unentscheidbar für komplexe Datentypen (Logische Spezifikationen)

- **Bestimme den Typ eines beliebigen Terms**
 - Vollautomatisch für einfache Typstrukturen (ML, Haskell, Java, ...)
 - Unentscheidbar für komplexe Datentypen (Logische Spezifikationen)
- **Analyse von $t \in T$**

- **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

- **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

- **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

- **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 $\lambda f. \lambda x. f \ x$**

- **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 $\lambda f . \lambda x . f \ x$**

$$\underbrace{\lambda f . \lambda x . f \ x}_{X_0} \in X_0$$

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 $\lambda f. \lambda x. f\ x$**

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

$$X_0 = X_1 \rightarrow X_2$$

- **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 $\lambda f. \lambda x. f x$**

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

$$X_2 = X_3 \rightarrow X_4$$

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 $\lambda f. \lambda x. f\ x$**

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

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 $\lambda f. \lambda x. f\ x$**

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

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 $\lambda f. \lambda x. f\ x$**
$$\lambda \underbrace{f}_{X_1}. \lambda \underbrace{x}_{X_3}. \underbrace{f}_{X_1} \underbrace{x}_{X_3} \in (X_3 \rightarrow X_4) \rightarrow (X_3 \rightarrow X_4)$$
- **Typisierung von $\lambda x. x\ x$**

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 $\lambda f. \lambda x. f x$**

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

- **Typisierung von $\lambda x. x x$** $\in X_0$

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 $\lambda f. \lambda x. f x$**

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

- **Typisierung von $\lambda x. x x$**

$$\lambda \underbrace{x}_{X_1}. \underbrace{x x}_{X_2} \in X_1 \rightarrow X_2$$

$\underbrace{\hspace{10em}}_{X_0}$

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 $\lambda f. \lambda x. f\ x$**

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

- **Typisierung von $\lambda x. x\ x$**

$$\lambda \underbrace{x}_{X_1}. \underbrace{\underbrace{x}_{X_1} \underbrace{x}_{X_1}}_{X_2} \in X_1 \rightarrow X_2 \quad X_1 = X_1 \rightarrow X_2$$

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 $\lambda f. \lambda x. f x$

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

● Typisierung von $\lambda \underbrace{x}_{X_1}. \underbrace{x}_{X_1} \underbrace{x}_{X_1} \in X_1 \rightarrow X_2 \quad X_1 = X_1 \rightarrow X_2$ nicht typisierbar

HINDLEY-MILNER TYPE-CHECKING ALGORITHMUS

- **Eingabe:** geschlossener λ -Term t
Ausgabe: prinzipielles Typschema von t oder Fehlermeldung

HINDLEY-MILNER TYPE-CHECKING ALGORITHMUS

- **Eingabe:** geschlossener λ -Term t
Ausgabe: prinzipielles Typschema von t oder Fehlermeldung
- **Start:** Setze globale Substitution $\sigma := []$ und rufe **TYPE-OF**($[]$, t) auf

HINDLEY-MILNER TYPE-CHECKING ALGORITHMUS

- **Eingabe:** geschlossener λ -Term t
 Ausgabe: prinzipielles Typschema von t oder Fehlermeldung
- **Start:** Setze globale Substitution $\sigma := []$ und rufe **TYPE-OF**($[]$, t) auf
- **Algorithmus** **TYPE-OF**(Env , t):

HINDLEY-MILNER TYPE-CHECKING ALGORITHMUS

- **Eingabe:** geschlossener λ -Term t
Ausgabe: prinzipielles Typschema von t oder Fehlermeldung
- **Start:** Setze globale Substitution $\sigma := []$ und rufe **TYPE-OF**($[]$, t) auf
- **Algorithmus** **TYPE-OF**(Env , t):
Falls $t = x \in \mathcal{V}$: suche Deklaration $x:T$ in Env und gebe T aus

HINDLEY-MILNER TYPE-CHECKING ALGORITHMUS

- **Eingabe:** geschlossener λ -Term t
Ausgabe: prinzipielles Typschema von t oder Fehlermeldung
- **Start:** Setze globale Substitution $\sigma := []$ und rufe **TYPE-OF**($[]$, t) auf
- **Algorithmus** **TYPE-OF**(Env , t):
 - Falls $t = x \in \mathcal{V}$: suche Deklaration $x:T$ in Env und gebe T aus
 - Falls $t = f\ u$: Setze $S_1 := \text{TYPE-OF}(Env, f)$, $S_2 := \text{TYPE-OF}(Env, u)$
Wähle neues X_{i+1} und unifiziere $\sigma(S_1)$ mit $S_2 \rightarrow X_{i+1}$.

HINDLEY-MILNER TYPE-CHECKING ALGORITHMUS

- **Eingabe:** geschlossener λ -Term t
Ausgabe: prinzipielles Typschema von t oder Fehlermeldung
- **Start:** Setze globale Substitution $\sigma := []$ und rufe **TYPE-OF**($[]$, t) auf
- **Algorithmus** **TYPE-OF**(Env , t):
 - Falls $t = x \in \mathcal{V}$: suche Deklaration $x:T$ in Env und gebe T aus
 - Falls $t = f\ u$: Setze $S_1 := \text{TYPE-OF}(Env, f)$, $S_2 := \text{TYPE-OF}(Env, u)$
Wähle neues $\mathbf{X}_{i+1}$ und unifiziere $\sigma(S_1)$ mit $S_2 \rightarrow \mathbf{X}_{i+1}$.
Fehlermeldung, wenn Unifikation fehlschlägt.

HINDLEY-MILNER TYPE-CHECKING ALGORITHMUS

- **Eingabe:** geschlossener λ -Term t
Ausgabe: prinzipielles Typschema von t oder Fehlermeldung
- **Start:** Setze globale Substitution $\sigma := []$ und rufe **TYPE-OF**($[]$, t) auf
- **Algorithmus** **TYPE-OF**(Env , t):
 - Falls $t = x \in \mathcal{V}$: suche Deklaration $x:T$ in Env und gebe T aus
 - Falls $t = f\ u$: Setze $S_1 := \text{TYPE-OF}(Env, f)$, $S_2 := \text{TYPE-OF}(Env, u)$
Wähle neues X_{i+1} und unifiziere $\sigma(S_1)$ mit $S_2 \rightarrow X_{i+1}$.
Fehlermeldung, wenn Unifikation fehlschlägt.
Sonst $\sigma := \sigma' \circ \sigma$, wobei σ' Ergebnis der Unifikation
Ausgabe: $\sigma(X_{i+1})$

HINDLEY-MILNER TYPE-CHECKING ALGORITHMUS

- **Eingabe:** geschlossener λ -Term t
Ausgabe: prinzipielles Typschema von t oder Fehlermeldung
- **Start:** Setze globale Substitution $\sigma := []$ und rufe **TYPE-OF**($[]$, t) auf
- **Algorithmus** **TYPE-OF**(Env , t):
 - Falls $t = x \in \mathcal{V}$: suche Deklaration $x:T$ in Env und gebe T aus
 - Falls $t = f\ u$: Setze $S_1 := \text{TYPE-OF}(Env, f)$, $S_2 := \text{TYPE-OF}(Env, u)$
Wähle neues X_{i+1} und unifiziere $\sigma(S_1)$ mit $S_2 \rightarrow X_{i+1}$.
Fehlermeldung, wenn Unifikation fehlschlägt.
Sonst $\sigma := \sigma' \circ \sigma$, wobei σ' Ergebnis der Unifikation
Ausgabe: $\sigma(X_{i+1})$
 - Falls $t = \lambda x. u$: Wähle neues X_{i+1}
$$S_1 := \begin{cases} \text{TYPE-OF}(Env \cdot [x:X_{i+1}], u) & \text{falls } x \text{ nicht in } Env \\ \text{TYPE-OF}(Env \cdot [x':X_{i+1}], u[x'/x]) & \text{sonst } (x' \in \mathcal{V} \text{ neu}) \end{cases}$$

Ausgabe: $\sigma(X_{i+1}) \rightarrow S_1$

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f\ x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f\ x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f\ x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
$f : X_0$	$\lambda f. \lambda x. f(f(f\ x))$ $\lambda x. f(f(f\ x))$			
	$\lambda f. \lambda x. f(f(f\ x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f\ x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f\ x))$			
$f : X_0$	$\lambda x. f(f(f\ x))$			
$f : X_0, x : X_1$	$f(f(f\ x))$			
$f : X_0$	$\lambda x. f(f(f\ x))$ $\lambda f. \lambda x. f(f(f\ x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f\ x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f\ x))$			
$f : X_0$	$\lambda x. f(f(f\ x))$			
$f : X_0, x : X_1$	$f(f(f\ x))$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f(f(f\ x))$			
$f : X_0$	$\lambda x. f(f(f\ x))$			
	$\lambda f. \lambda x. f(f(f\ x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f\ x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f\ x))$			
$f : X_0$	$\lambda x. f(f(f\ x))$			
$f : X_0, x : X_1$	$f(f(f\ x))$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f(f\ x)$			
$f : X_0, x : X_1$	$f(f(f\ x))$			
$f : X_0$	$\lambda x. f(f(f\ x))$			
	$\lambda f. \lambda x. f(f(f\ x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f(f x)$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f(f x)$			
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
	$\lambda f. \lambda x. f(f(f x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f\ x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f\ x))$			
$f : X_0$	$\lambda x. f(f(f\ x))$			
$f : X_0, x : X_1$	$f(f(f\ x))$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f(f\ x)$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f\ x$			
$f : X_0, x : X_1$	$f(f\ x)$			
$f : X_0, x : X_1$	$f(f(f\ x))$			
$f : X_0$	$\lambda x. f(f(f\ x))$			
	$\lambda f. \lambda x. f(f(f\ x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f(f x)$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f x$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f x$			
$f : X_0, x : X_1$	$f(f x)$			
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
	$\lambda f. \lambda x. f(f(f x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f(f x)$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f x$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	x			X_1
$f : X_0, x : X_1$	$f x$			
$f : X_0, x : X_1$	$f(f x)$			
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
	$\lambda f. \lambda x. f(f(f x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f(f x)$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f x$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	x			X_1
$f : X_0, x : X_1$	$f x$		$X_0 = X_1 \rightarrow X_2$	X_2
$f : X_0, x : X_1$	$f(f x)$			
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
	$\lambda f. \lambda x. f(f(f x))$			

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f x))$

<i>Env</i>	<i>Aktueller Term t</i>	<i>Substitution σ</i>	UNIFY	<i>Typ T</i>
	$\lambda f. \lambda x. f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f(f x)$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	$f x$			
$f : X_0, x : X_1$	f			X_0
$f : X_0, x : X_1$	x			X_1
$f : X_0, x : X_1$	$f x$		$X_0 = X_1 \rightarrow X_2$	X_2
$f : X_0, x : X_1$	$f(f x)$	$[X_1 \rightarrow X_2 / X_0]$	$X_1 \rightarrow X_2 = X_2 \rightarrow X_3$	X_3
$f : X_0, x : X_1$	$f(f(f x))$			
$f : X_0$	$\lambda x. f(f(f x))$			
	$\lambda f. \lambda x. f(f(f x))$			

TYPISIERUNG VON $\lambda f . \lambda x . f(f(f x))$

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

TYPISIERUNG VON $\lambda f. \lambda x. f(f(f x))$

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

TYPISIERUNG VON $\lambda f . \lambda x . f(f(f x))$

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

- **Umsetzung der Urteile in Inferenzregeln**
 - Urteile sind schematische Regeln zur Definition von Semantik
 - Inferenzregeln sind syntaktische Vorschriften für Beweisführung

● Umsetzung der Urteile in Inferenzregeln

- Urteile sind schematische Regeln zur Definition von Semantik
- Inferenzregeln sind syntaktische Vorschriften für Beweisführung
- Ähnlichkeit erleichtert Nachweis von Korrektheit und Vollständigkeit

- **Umsetzung der Urteile in Inferenzregeln**

- Urteile sind schematische Regeln zur Definition von Semantik
- Inferenzregeln sind syntaktische Vorschriften für Beweisführung
- Ähnlichkeit erleichtert Nachweis von Korrektheit und Vollständigkeit

- **Schließen über Gleichheit und Berechnung**

- Regeln zur Auswertung von Termen wie im λ -Kalkül

- **Umsetzung der Urteile in Inferenzregeln**

- Urteile sind schematische Regeln zur Definition von Semantik
- Inferenzregeln sind syntaktische Vorschriften für Beweisführung
- Ähnlichkeit erleichtert Nachweis von Korrektheit und Vollständigkeit

- **Schließen über Gleichheit und Berechnung**

- Regeln zur Auswertung von Termen wie im λ -Kalkül

- **Schließen über Typzugehörigkeit**

- Dekompositionsregeln zur Zuordnung von Typen zu Termen
- Benötigt Nachweis, daß manche Ausdrücke Datentypen darstellen

- **Umsetzung der Urteile in Inferenzregeln**

- Urteile sind schematische Regeln zur Definition von Semantik
- Inferenzregeln sind syntaktische Vorschriften für Beweisführung
- Ähnlichkeit erleichtert Nachweis von Korrektheit und Vollständigkeit

- **Schließen über Gleichheit und Berechnung**

- Regeln zur Auswertung von Termen wie im λ -Kalkül

- **Schließen über Typzugehörigkeit**

- Dekompositionsregeln zur Zuordnung von Typen zu Termen
- Benötigt Nachweis, daß manche Ausdrücke Datentypen darstellen

- **Schließen über Datentypen**

- Regeln zur strukturellen Analyse von (Typ-)Ausdrücken

TYPZUGEHÖRIGKEIT: WELCHE REGELN SIND NÖTIG?

- $f\ t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S

TYPZUGEHÖRIGKEIT: WELCHE REGELN SIND NÖTIG?

- $f\ t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S

- Direkte Umsetzung möglich $\Gamma \vdash f\ t \in T$ BY **applyI** S

$$\Gamma \vdash f \in S \rightarrow T$$

$$\Gamma \vdash t \in S$$

- Typ S muß als Parameter der Regel angegeben werden

TYPZUGEHÖRIGKEIT: WELCHE REGELN SIND NÖTIG?

- $f\ t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S
 - Direkte Umsetzung möglich $\Gamma \vdash f\ t \in T$ BY **applyI** S
$$\Gamma \vdash f \in S \rightarrow T$$
$$\Gamma \vdash t \in S$$
 - Typ S muß als Parameter der Regel angegeben werden
- $\lambda x. t \in S \rightarrow T$ falls $t \in T$ aus $x \in S$ folgt

TYPZUGEHÖRIGKEIT: WELCHE REGELN SIND NÖTIG?

- $f\ t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S
 - Direkte Umsetzung möglich $\Gamma \vdash f\ t \in T$ BY **applyI** S
$$\Gamma \vdash f \in S \rightarrow T$$
$$\Gamma \vdash t \in S$$
 - Typ S muß als Parameter der Regel angegeben werden
- $\lambda x. t \in S \rightarrow T$ falls $t \in T$ aus $x \in S$ folgt
 - Umsetzung erfordert Typdeklaration für $x \in S$ in der Teilzielsequenz
$$\Gamma \vdash \lambda x. t \in S \rightarrow T$$
 BY **lambdaI**

TYPZUGEHÖRIGKEIT: WELCHE REGELN SIND NÖTIG?

- $f\ t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S

- Direkte Umsetzung möglich $\Gamma \vdash f\ t \in T$ BY **applyI** S

$$\Gamma \vdash f \in S \rightarrow T$$

$$\Gamma \vdash t \in S$$

- Typ S muß als Parameter der Regel angegeben werden

- $\lambda x. t \in S \rightarrow T$ falls $t \in T$ aus $x \in S$ folgt

- Umsetzung erfordert Typdeklaration für $x \in S$ in der Teilzielsequenz

$$\Gamma \vdash \lambda x. t \in S \rightarrow T \text{ BY } \mathbf{lambdaI}$$

$$\Gamma, x:S \vdash t \in T$$

TYPZUGEHÖRIGKEIT: WELCHE REGELN SIND NÖTIG?

- $f\ t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S

- Direkte Umsetzung möglich $\Gamma \vdash f\ t \in T$ BY **applyI** S

$$\Gamma \vdash f \in S \rightarrow T$$

$$\Gamma \vdash t \in S$$

- Typ S muß als Parameter der Regel angegeben werden

- $\lambda x. t \in S \rightarrow T$ falls $t \in T$ aus $x \in S$ folgt

- Umsetzung erfordert Typdeklaration für $x \in S$ in der Teilzielsequenz

$$\Gamma \vdash \lambda x. t \in S \rightarrow T \text{ BY } \mathbf{lambdaI}$$

$$\Gamma, x':S \vdash t[x'/x] \in T$$

- Umbenennung der Variablen x ist eventuell notwendig

TYPZUGEHÖRIGKEIT: WELCHE REGELN SIND NÖTIG?

- $f\ t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S
 - Direkte Umsetzung möglich $\Gamma \vdash f\ t \in T$ BY **applyI** S
$$\Gamma \vdash f \in S \rightarrow T$$
$$\Gamma \vdash t \in S$$
 - Typ S muß als Parameter der Regel angegeben werden
- $\lambda x. t \in S \rightarrow T$ falls $t \in T$ aus $x \in S$ folgt
 - Umsetzung erfordert Typdeklaration für $x \in S$ in der Teilzielsequenz
$$\Gamma \vdash \lambda x. t \in S \rightarrow T$$
 BY **lambdaI**
$$\Gamma, x':S \vdash t[x'/x] \in T$$
$$\Gamma \vdash S \text{ Type}$$
 - S muß ein Typ sein, wenn es rechts in einer Deklaration auftaucht
 - Umbenennung der Variablen x ist eventuell notwendig

TYPZUGEHÖRIGKEIT: WELCHE REGELN SIND NÖTIG?

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

TYPZUGEHÖRIGKEIT: WELCHE REGELN SIND NÖTIG?

- $f\ t \in T$ falls $f \in S \rightarrow T$ und $t \in S$ für einen Typ S

- Direkte Umsetzung möglich $\Gamma \vdash f\ t \in T$ BY **applyI** S

$$\Gamma \vdash f \in S \rightarrow T$$

$$\Gamma \vdash t \in S$$

- Typ S muß als Parameter der Regel angegeben werden

- $\lambda x. t \in S \rightarrow T$ falls $t \in T$ aus $x \in S$ folgt

- Umsetzung erfordert Typdeklaration für $x \in S$ in der Teilzielsequenz

$$\Gamma \vdash \lambda x. t \in S \rightarrow T \text{ BY } \mathbf{lambdaI}$$

$$\Gamma, x':S \vdash t[x'/x] \in T$$

$$\Gamma \vdash S \text{ Type}$$

- S muß ein Typ sein, wenn es rechts in einer Deklaration auftaucht

- Umbenennung der Variablen x ist eventuell notwendig

- $x \in T$ falls $x \in \mathcal{V}$ und T ein Typ ist

- Im formalen Beweis muß die Zuordnung von T zu x deklariert sein

$$\Gamma, x:T, \Delta \vdash x \in T \text{ BY } \mathbf{declaration}$$

- **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 $\mathbb{Q}$

- **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

INTEGRATION VON TYPZUGEHÖRIGKEIT UND GLEICHHEIT

- **Gleichheit hängt eigentlich vom Typ ab**
 - 4 und 7 sind verschieden als Zahlen aber gleich in $\mathbb{Z}_3$
 - (2,4) und (1,2) sind verschieden als Zahlenpaare aber gleich in $\mathbb{Q}$
 - In der einfachen Typentheorie haben semantisch gleiche Terme allerdings immer denselben Typ
- **Kombiniere $s=t$ und $t \in T$ zu $s=t \in T$**
 - Prädikat mit fester Bedeutung: “ s und t sind im Typ T gleich”

- **Gleichheit hängt eigentlich vom Typ ab**

- 4 und 7 sind verschieden als Zahlen aber gleich in $\mathbb{Z}_3$
- (2,4) und (1,2) sind verschieden als Zahlenpaare aber gleich in $\mathbb{Q}$
- In der einfachen Typentheorie haben semantisch gleiche Terme allerdings immer denselben Typ

- **Kombiniere $s=t$ und $t \in T$ zu $s=t \in T$**

- Prädikat mit fester Bedeutung: “ s und t sind im Typ T gleich”
- $s=t \in T$ bedingt neben der Gleichheit auch $s \in T$ und $t \in T$

- **Gleichheit hängt eigentlich vom Typ ab**

- 4 und 7 sind verschieden als Zahlen aber gleich in $\mathbb{Z}_3$
- (2,4) und (1,2) sind verschieden als Zahlenpaare aber gleich in $\mathbb{Q}$
- In der einfachen Typentheorie haben semantisch gleiche Terme allerdings immer denselben Typ

- **Kombiniere $s=t$ und $t \in T$ zu $s=t \in T$**

- Prädikat mit fester Bedeutung: “ s und t sind im Typ T gleich”
- $s=t \in T$ bedingt neben der Gleichheit auch $s \in T$ und $t \in T$
- Typzugehörigkeit $t \in T$ kann als Kurzform für $t=t \in T$ betrachtet werden
- Die Regeln **applyI** und **lambdaI** werden als spezielle Form von entsprechenden Gleichheitsregeln **applyEq** und **lambdaEq** betrachtet

- **Gleichheit hängt eigentlich vom Typ ab**

- 4 und 7 sind verschieden als Zahlen aber gleich in $\mathbb{Z}_3$
- (2,4) und (1,2) sind verschieden als Zahlenpaare aber gleich in $\mathbb{Q}$
- In der einfachen Typentheorie haben semantisch gleiche Terme allerdings immer denselben Typ

- **Kombiniere $s=t$ und $t \in T$ zu $s=t \in T$**

- Prädikat mit fester Bedeutung: “ s und t sind im Typ T gleich”
- $s=t \in T$ bedingt neben der Gleichheit auch $s \in T$ und $t \in T$
- Typzugehörigkeit $t \in T$ kann als Kurzform für $t=t \in T$ betrachtet werden
- Die Regeln **applyI** und **lambdaI** werden als spezielle Form von entsprechenden Gleichheitsregeln **applyEq** und **lambdaEq** betrachtet

- **Allgemeine Reflexivitätsregel wird ungültig**

- $t=t \in T$ gilt nur, wenn auch $t \in T$ gezeigt werden kann

NOTWENDIGE ERWEITERUNGEN FORMALER KONZEPTE

- Neues festes Symbol $\mathbb{U}$ (**U**niversum aller Typen)

NOTWENDIGE ERWEITERUNGEN FORMALER KONZEPTE

- Neues festes Symbol $\mathbb{U}$ (**U**niversum aller Typen)
- **D**eklaration von Variablen und Typen möglich
 - $x:S$ ($x \in \mathcal{V}$, Variablendeklaration), oder $T:\mathbb{U}$ ($T \in \mathcal{T}$, Typdekларation)
 - Konzept der Deklaration in Sequenzen bekommt erstmalig einen Sinn

NOTWENDIGE ERWEITERUNGEN FORMALER KONZEPTE

- Neues festes Symbol $\mathbb{U}$ (**U**niversum aller Typen)
- **D**eklaration von Variablen und Typen möglich
 - $x:S$ ($x \in \mathcal{V}$, Variablendeklaration), oder $T:\mathbb{U}$ ($T \in \mathcal{T}$, Typdekларation)
 - Konzept der Deklaration in Sequenzen bekommt erstmalig einen Sinn
- **K**onklusion kann **T**ypisierungen enthalten
 - Typisierte Gleichheit: $s=t \in T$ (oder Kurzform $t \in T$)
 - Typeigenschaft: $T \in \mathbb{U}$ (“ T ist ein Typ”)

NOTWENDIGE ERWEITERUNGEN FORMALER KONZEPTE

- Neues festes Symbol $\mathbb{U}$ (**U**niversum aller Typen)
- **D**eklaration von Variablen und Typen möglich
 - $x:S$ ($x \in \mathcal{V}$, Variablendeklaration), oder $T:\mathbb{U}$ ($T \in \mathcal{T}$, Typdekларation)
 - Konzept der Deklaration in Sequenzen bekommt erstmalig einen Sinn
- **K**onklusion kann **T**ypisierungen enthalten
 - Typisierte Gleichheit: $s=t \in T$ (oder Kurzform $t \in T$)
 - Typeigenschaft: $T \in \mathbb{U}$ (“ T ist ein Typ”)
- **S**equenz: $H_1, \dots, H_n \vdash C$
 - H_i Formel oder Deklaration, C Formel (evtl. mit Typisierungen)

NOTWENDIGE ERWEITERUNGEN FORMALER KONZEPTE

- Neues festes Symbol $\mathbb{U}$ (**U**niversum aller Typen)
- **D**eklaration von Variablen und Typen möglich
 - $x:S$ ($x \in \mathcal{V}$, Variablendeklaration), oder $T:\mathbb{U}$ ($T \in \mathcal{T}$, Typdeklaration)
 - Konzept der Deklaration in Sequenzen bekommt erstmalig einen Sinn
- **K**onklusion kann **T**ypisierungen enthalten
 - Typisierte Gleichheit: $s=t \in T$ (oder Kurzform $t \in T$)
 - Typeigenschaft: $T \in \mathbb{U}$ (“ T ist ein Typ”)
- **S**equenz: $H_1, \dots, H_n \vdash C$
 - H_i Formel oder Deklaration, C Formel (evtl. mit Typisierungen)
 - $H_1, \dots, H_n$ ist **rein**, wenn jede Variable genau einmal deklariert ist und jede Typvariable einer Variablendeklaration zuvor deklariert wurde

NOTWENDIGE ERWEITERUNGEN FORMALER KONZEPTE

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

NOTWENDIGE ERWEITERUNGEN FORMALER KONZEPTE

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

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

INFERENZREGELN DER TYPENTHEORIE

Verfeinere Beweisregeln des λ -Kalküls

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

INFERENZREGELN DER TYPENTHEORIE

Verfeinere Beweisregeln des λ -Kalküls

applyEq S

$$\begin{array}{l} \Gamma \vdash f\ t = g\ u \\ \Gamma \vdash f = g \\ \Gamma \vdash t = u \end{array}$$

lambdaEq *

$$\begin{array}{l} \Gamma \vdash \lambda x. t = \lambda y. u \\ \Gamma, x':S \vdash t[x'/x] = u[x'/y] \end{array}$$

reduction

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

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

INFERENZREGELN DER TYPENTHEORIE

Verfeinere Beweisregeln des λ -Kalküls

applyEq S	$\Gamma \vdash f t = g u \in T$ $\Gamma \vdash f = g \in S \rightarrow T$ $\Gamma \vdash t = u \in S$
lambdaEq *	$\Gamma \vdash \lambda x. t = \lambda y. u$ $\Gamma, x': S \vdash t[x'/x] = u[x'/y]$
reduction	$\Gamma \vdash (\lambda x. u) s = t$ $\Gamma \vdash u[s/x] = t$

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

INFERENZREGELN DER TYPENTHEORIE

Verfeinere Beweisregeln des λ -Kalküls

applyEq S	$\Gamma \vdash f t = g u \in T$ $\Gamma \vdash f = g \in S \rightarrow T$ $\Gamma \vdash t = u \in S$
lambdaEq $*$	$\Gamma \vdash \lambda x. t = \lambda y. u \in S \rightarrow T$ $\Gamma, x': S \vdash t[x'/x] = u[x'/y] \in T$ $\Gamma \vdash S \in \mathbb{U}$
reduction	$\Gamma \vdash (\lambda x. u) s = t$ $\Gamma \vdash u[s/x] = t$

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

INFERENZREGELN DER TYPENTHEORIE

Verfeinere Beweisregeln des λ -Kalküls

applyEq S	$\Gamma \vdash f t = g u \in T$ $\Gamma \vdash f = g \in S \rightarrow T$ $\Gamma \vdash t = u \in S$
lambdaEq $*$	$\Gamma \vdash \lambda x. t = \lambda y. u \in S \rightarrow T$ $\Gamma, x': S \vdash t[x'/x] = u[x'/y] \in T$ $\Gamma \vdash S \in \mathbb{U}$
reduction	$\Gamma \vdash (\lambda x. u) s = t \in T$ $\Gamma \vdash u[s/x] = t \in T$

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

INFERENZREGELN DER TYPENTHEORIE

Verfeinere Beweisregeln des λ -Kalküls

applyEq S	$\Gamma \vdash f t = g u \in T$ $\Gamma \vdash f = g \in S \rightarrow T$ $\Gamma \vdash t = u \in S$
lambdaEq $*$	$\Gamma \vdash \lambda x. t = \lambda y. u \in S \rightarrow T$ $\Gamma, x': S \vdash t[x'/x] = u[x'/y] \in T$ $\Gamma \vdash S \in \mathbb{U}$
reduction	$\Gamma \vdash (\lambda x. u) s = t \in T$ $\Gamma \vdash u[s/x] = t \in T$
declaration i	$\Gamma, x: T, \Delta \vdash x \in T \quad \hat{=} \quad x = x \in T$

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

INFERENZREGELN DER TYPENTHEORIE

Verfeinere Beweisregeln des λ -Kalküls

applyEq S	$\Gamma \vdash f t = g u \in T$ $\Gamma \vdash f = g \in S \rightarrow T$ $\Gamma \vdash t = u \in S$
lambdaEq $*$	$\Gamma \vdash \lambda x. t = \lambda y. u \in S \rightarrow T$ $\Gamma, x': S \vdash t[x'/x] = u[x'/y] \in T$ $\Gamma \vdash S \in \mathbb{U}$
reduction	$\Gamma \vdash (\lambda x. u) s = t \in T$ $\Gamma \vdash u[s/x] = t \in T$
declaration i	$\Gamma, x: T, \Delta \vdash x \in T \quad \hat{=} \quad x = x \in T$
functionI	$\Gamma \vdash S \rightarrow T \in \mathbb{U}$ $\Gamma \vdash S \in \mathbb{U}$ $\Gamma \vdash T \in \mathbb{U}$

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

ABGELEITETE INFERENZREGELN

Regeln für Typzugehörigkeit sind Spezialfall

<code>applyEq</code>	S	$\Gamma \vdash f\ t = g\ u \in T$ $\Gamma \vdash f = g \in S \rightarrow T$ $\Gamma \vdash t = u \in S$
<code>lambdaEq</code>		$\Gamma \vdash \lambda x. t = \lambda y. u \in S \rightarrow T$ $\Gamma, x': S \vdash t[x'/x] = u[x'/y] \in T$ $\Gamma \vdash S \in \mathbb{U}$
<code>reduction</code>		$\Gamma \vdash (\lambda x. u)\ s = t \in T$ $\Gamma \vdash u[s/x] = t \in T$
<code>declaration</code>	i	$\Gamma, x: T, \Delta \vdash x \in T$
<code>functionI</code>		$\Gamma \vdash S \rightarrow T \in \mathbb{U}$ $\Gamma \vdash S \in \mathbb{U}$ $\Gamma \vdash T \in \mathbb{U}$

ABGELEITETE INFERENZREGELN

Regeln für Typzugehörigkeit sind Spezialfall

<code>applyEq</code>	S	$\Gamma \vdash f\ t = f\ t \in T$ $\Gamma \vdash f = f \in S \rightarrow T$ $\Gamma \vdash t = t \in S$
<code>lambdaEq</code>		$\Gamma \vdash \lambda x. t = \lambda y. u \in S \rightarrow T$ $\Gamma, x': S \vdash t[x'/x] = u[x'/y] \in T$ $\Gamma \vdash S \in \mathbb{U}$
<code>reduction</code>		$\Gamma \vdash (\lambda x. u)\ s = t \in T$ $\Gamma \vdash u[s/x] = t \in T$
<code>declaration</code>	i	$\Gamma, x:T, \Delta \vdash x \in T$
<code>functionI</code>		$\Gamma \vdash S \rightarrow T \in \mathbb{U}$ $\Gamma \vdash S \in \mathbb{U}$ $\Gamma \vdash T \in \mathbb{U}$

ABGELEITETE INFERENZREGELN

Regeln für Typzugehörigkeit sind Spezialfall

<code>applyI</code>	S	$\Gamma \vdash f t \in T$ $\Gamma \vdash f \in S \rightarrow T$ $\Gamma \vdash t \in S$
<code>lambdaEq</code>		$\Gamma \vdash \lambda x. t = \lambda y. u \in S \rightarrow T$ $\Gamma, x': S \vdash t[x'/x] = u[x'/y] \in T$ $\Gamma \vdash S \in \mathbb{U}$
<code>reduction</code>		$\Gamma \vdash (\lambda x. u) s = t \in T$ $\Gamma \vdash u[s/x] = t \in T$
<code>declaration</code>	i	$\Gamma, x: T, \Delta \vdash x \in T$
<code>functionI</code>		$\Gamma \vdash S \rightarrow T \in \mathbb{U}$ $\Gamma \vdash S \in \mathbb{U}$ $\Gamma \vdash T \in \mathbb{U}$

ABGELEITETE INFERENZREGELN

Regeln für Typzugehörigkeit sind Spezialfall

<code>applyI</code>	S	$\Gamma \vdash f\ t \in T$ $\Gamma \vdash f \in S \rightarrow T$ $\Gamma \vdash t \in S$
<code>lambdaI</code>		$\Gamma \vdash \lambda x. t \in S \rightarrow T$ $\Gamma, x': S \vdash t[x'/x] \in T$ $\Gamma \vdash S \in \mathbb{U}$
<code>reduction</code>		$\Gamma \vdash (\lambda x. u)\ s = t \in T$ $\Gamma \vdash u[s/x] = t \in T$
<code>declaration</code>	i	$\Gamma, x:T, \Delta \vdash x \in T$
<code>functionI</code>		$\Gamma \vdash S \rightarrow T \in \mathbb{U}$ $\Gamma \vdash S \in \mathbb{U}$ $\Gamma \vdash T \in \mathbb{U}$

SEQUENZENBEWEIS FÜR $(\lambda f.\lambda x.f\ x)(\lambda z.z) \in T \rightarrow T$

$T:\mathbb{U} \vdash (\lambda f.\lambda x.f\ x)(\lambda z.z) \in T \rightarrow T$	BY applyI $T \rightarrow T$
$T:\mathbb{U} \vdash \lambda f.\lambda x.f\ x \in (T \rightarrow T) \rightarrow (T \rightarrow T)$	BY lambdaI
$T:\mathbb{U}, f:T \rightarrow T \vdash \lambda x.f\ x \in T \rightarrow T$	BY lambdaI
$T:\mathbb{U}, f:T \rightarrow T, x:T \vdash f\ x \in T$	BY applyI T
$T:\mathbb{U}, f:T \rightarrow T, x:T \vdash f \in T \rightarrow T$	BY declaration 2
$T:\mathbb{U}, f:T \rightarrow T, x:T \vdash x \in T$	BY declaration 3
$T:\mathbb{U}, f:T \rightarrow T \vdash T \in \mathbb{U}$	BY declaration 1
$T:\mathbb{U} \vdash T \rightarrow T \in \mathbb{U}$	BY functionI
$T:\mathbb{U} \vdash T \in \mathbb{U}$	BY declaration 1
$T:\mathbb{U} \vdash T \in \mathbb{U}$	BY declaration 1
$T:\mathbb{U} \vdash \lambda z.z \in T \rightarrow T$	BY lambdaI
$T:\mathbb{U}, z:T \vdash z \in T$	BY declaration 2
$T:\mathbb{U} \vdash T \in \mathbb{U}$	BY declaration 1

LOGIKREGELN WERDEN REDUNDANT

$$\begin{array}{c} \Gamma \vdash A \Rightarrow B \text{ BY } \text{impI} \\ \Gamma, A \vdash B \end{array}$$

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY `lambdaI`

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

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

$\Gamma, A \vdash B$

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY lambdaI

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

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

$\Gamma, A \vdash B$

$\Gamma \vdash ft \in T$ BY applyI S

$\Gamma \vdash f \in S \rightarrow T$

$\Gamma \vdash t \in S$

$\Gamma \vdash B$ BY modus ponens A

$\Gamma \vdash A \Rightarrow B$

$\Gamma \vdash A$

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY lambdaI

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

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

$\Gamma, A \vdash B$

$\Gamma \vdash f t \in T$ BY applyI S

$\Gamma \vdash f \in S \rightarrow T$

$\Gamma \vdash t \in S$

$\Gamma \vdash B$ BY modus ponens A

$\Gamma \vdash A \Rightarrow B$

$\Gamma \vdash A$

Curry-Howard Isomorphie

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY lambdaI

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

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

$\Gamma, A \vdash B$

$\Gamma \vdash f t \in T$ BY applyI S

$\Gamma \vdash f \in S \rightarrow T$

$\Gamma \vdash t \in S$

$\Gamma \vdash B$ BY modus ponens A

$\Gamma \vdash A \Rightarrow B$

$\Gamma \vdash A$

Curry-Howard Isomorphie

Typ

Formel

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY lambdaI

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

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

$\Gamma, A \vdash B$

$\Gamma \vdash f t \in T$ BY applyI S

$\Gamma \vdash f \in S \rightarrow T$

$\Gamma \vdash t \in S$

$\Gamma \vdash B$ BY modus ponens A

$\Gamma \vdash A \Rightarrow B$

$\Gamma \vdash A$

Curry-Howard Isomorphie

Typ

Variable vom Typ S

Formel

Annahme der Formel S

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY **lambdaI**

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

$\Gamma \vdash A \Rightarrow B$ BY **impI**

$\Gamma, A \vdash B$

$\Gamma \vdash f t \in T$ BY **applyI** S

$\Gamma \vdash f \in S \rightarrow T$

$\Gamma \vdash t \in S$

$\Gamma \vdash B$ BY **modus ponens** A

$\Gamma \vdash A \Rightarrow B$

$\Gamma \vdash A$

Curry-Howard Isomorphie

Typ

Variable vom Typ S

Term vom Typ T (freie Variablen aus S_i)

Formel

Annahme der Formel S

Beweis von T (Annahmen S_i)

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY lambdaI

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

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

$\Gamma, A \vdash B$

$\Gamma \vdash f t \in T$ BY applyI S

$\Gamma \vdash f \in S \rightarrow T$

$\Gamma \vdash t \in S$

$\Gamma \vdash B$ BY modus ponens A

$\Gamma \vdash A \Rightarrow B$

$\Gamma \vdash A$

Curry-Howard Isomorphie

Typ

Variable vom Typ S

Term vom Typ T (freie Variablen aus S_i)

Typechecking

Formel

Annahme der Formel S

Beweis von T (Annahmen S_i)

Beweisführung

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY **lambdaI**

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

$\Gamma \vdash A \Rightarrow B$ BY **impI**

$\Gamma, A \vdash B$

$\Gamma \vdash f t \in T$ BY **applyI** S

$\Gamma \vdash f \in S \rightarrow T$

$\Gamma \vdash t \in S$

$\Gamma \vdash B$ BY **modus ponens** A

$\Gamma \vdash A \Rightarrow B$

$\Gamma \vdash A$

Curry-Howard Isomorphie

Typ

Variable vom Typ S

Term vom Typ T (freie Variablen aus S_i)

Typechecking

(Regel der) λ -Abstraktion

Formel

Annahme der Formel S

Beweis von T (Annahmen S_i)

Beweisführung

$\Rightarrow$ -Einführung

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY **lambdaI**

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

$\Gamma \vdash A \Rightarrow B$ BY **impI**

$\Gamma, A \vdash B$

$\Gamma \vdash f t \in T$ BY **applyI** S

$\Gamma \vdash f \in S \rightarrow T$

$\Gamma \vdash t \in S$

$\Gamma \vdash B$ BY **modus ponens** A

$\Gamma \vdash A \Rightarrow B$

$\Gamma \vdash A$

Curry-Howard Isomorphie

Typ

Variable vom Typ S

Term vom Typ T (freie Variablen aus S_i)

Typechecking

(Regel der) λ -Abstraktion

(Regel der) Applikation

Formel

Annahme der Formel S

Beweis von T (Annahmen S_i)

Beweisführung

$\Rightarrow$ -Einführung

$\Rightarrow$ -Elimination

LOGIKREGELN WERDEN REDUNDANT

$\Gamma \vdash \lambda x.t \in S \rightarrow T$ BY **lambdaI**

$\Gamma, x':S \vdash t[x'/x] \in T$

$\Gamma \vdash S \in \mathbb{U}$

$\Gamma \vdash A \Rightarrow B$ BY **impI**

$\Gamma, A \vdash B$

$\Gamma \vdash f t \in T$ BY **applyI** S

$\Gamma \vdash f \in S \rightarrow T$

$\Gamma \vdash t \in S$

$\Gamma \vdash B$ BY **modus ponens** A

$\Gamma \vdash A \Rightarrow B$

$\Gamma \vdash A$

Curry-Howard Isomorphie

Typ

Variable vom Typ S

Term vom Typ T (freie Variablen aus S_i)

Typechecking

(Regel der) λ -Abstraktion

(Regel der) Applikation

Formel

Annahme der Formel S

Beweis von T (Annahmen S_i)

Beweisführung

$\Rightarrow$ -Einführung

$\Rightarrow$ -Elimination

Logik als Spezialfall der Typentheorie ?

- **Schwache Normalisierbarkeit**

- Hat jeder Term eine Normalform?
- Führt mindestens eine Reduktionsstrategie immer zu einem Wert?
- Damit würden alle Funktionen total

EIGENSCHAFTEN TYPISIERBARER TERME

- **Schwache Normalisierbarkeit**

- Hat jeder Term eine Normalform?
- Führt mindestens eine Reduktionsstrategie immer zu einem Wert?
- Damit würden alle Funktionen total

- **Starke Normalisierbarkeit**

- Terminiert jede beliebige Reduktionsfolge?
- Damit könnte jede Reduktionsstrategie gewählt werden

EIGENSCHAFTEN TYPISIERBARER TERME

- **Schwache Normalisierbarkeit**

- Hat jeder Term eine Normalform?
- Führt mindestens eine Reduktionsstrategie immer zu einem Wert?
- Damit würden alle Funktionen total

- **Starke Normalisierbarkeit**

- Terminiert jede beliebige Reduktionsfolge?
- Damit könnte jede Reduktionsstrategie gewählt werden

- **Konfluenz (Church-Rosser Eigenschaft)**

- Kann man verschiedene Reduktionsfolgen wieder zusammenführen?
- Damit wäre die Normalform eines Terms eindeutig

EIGENSCHAFTEN TYPISIERBARER TERME

- **Schwache Normalisierbarkeit**

- Hat jeder Term eine Normalform?
- Führt mindestens eine Reduktionsstrategie immer zu einem Wert?
- Damit würden alle Funktionen total

- **Starke Normalisierbarkeit**

- Terminiert jede beliebige Reduktionsfolge?
- Damit könnte jede Reduktionsstrategie gewählt werden

- **Konfluenz (Church-Rosser Eigenschaft)**

- Kann man verschiedene Reduktionsfolgen wieder zusammenführen?
- Damit wäre die Normalform eines Terms eindeutig

- **Entscheidbarkeit**

- Ist Typisierbarkeit oder Gleichheit von Termen entscheidbar?
- Damit würde Korrektheit / Äquivalenz von Programmen testbar

EIGENSCHAFTEN TYPISIERBARER TERME

- **Schwache Normalisierbarkeit**

- Hat jeder Term eine Normalform?
- Führt mindestens eine Reduktionsstrategie immer zu einem Wert?
- Damit würden alle Funktionen total

- **Starke Normalisierbarkeit**

- Terminiert jede beliebige Reduktionsfolge?
- Damit könnte jede Reduktionsstrategie gewählt werden

- **Konfluenz (Church-Rosser Eigenschaft)**

- Kann man verschiedene Reduktionsfolgen wieder zusammenführen?
- Damit wäre die Normalform eines Terms eindeutig

- **Entscheidbarkeit**

- Ist Typisierbarkeit oder Gleichheit von Termen entscheidbar?
- Damit würde Korrektheit / Äquivalenz von Programmen testbar

- **Ausdruckskraft**

- Gibt es wichtige Funktionen, die nicht typisierbar sind?

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

Hat jeder typisierbare λ -Term eine Normalform?

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

Hat jeder typisierbare λ -Term eine Normalform?

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

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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 ( $\lambda x.x$ ))` mit `trice  $\equiv \lambda f.\lambda x.f(f f x)$`

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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 ( $\lambda x.x$ ))` mit `trice  $\equiv \lambda f.\lambda x.f(f f x)$`

Reduziere `trice (trice ( $\lambda x.x$ ))`

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f f x)$

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

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

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

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

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

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)))$

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

$((\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x))) x))$ Nur Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)))$

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

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

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)))$

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

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

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)))$

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

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

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x. x))$ mit $\text{trice} \equiv \lambda f. \lambda x. f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x. x) ((\lambda x. x) ((\lambda x. x) x)))$

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

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

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)))$

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

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)))$

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

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)))$

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

$x)$

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)))$

$((\lambda x. x)$

$x)$

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)))$

x

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) x))$

x

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. (\lambda x.x) x)$

x

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Reduziere $\lambda x. (\lambda x. x)$

x

Tiefe 1

SCHWACHE NORMALISIERBARKEIT

Hat jeder typisierbare λ -Term eine Normalform?

- **Tiefe $d(T)$ eines Typausdrucks T**

- $d(T) = 0$, falls $T \in \mathcal{T}$, $d(S \rightarrow T) = 1 + \max(d(S), d(T))$
- 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: $\text{trice}(\text{trice}(\lambda x.x))$ mit $\text{trice} \equiv \lambda f.\lambda x.f(f(f x))$

Ergebnis $\lambda x.x$

Keine Redices

RIGHTMOST-MAXDEPTH STRATEGIE

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

RIGHTMOST-MAXDEPTH STRATEGIE

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

- **Lemma:** Die Strategie verringert die Anzahl der Redizes maximaler Tiefe

RIGHTMOST-MAXDEPTH STRATEGIE

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

- **Lemma:** Die Strategie verringert die Anzahl der Redizes maximaler Tiefe
 - Sei $r = (\lambda x. u) v$ das rightmost-maxdepth Redex von t

RIGHTMOST-MAXDEPTH STRATEGIE

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

- **Lemma:** Die Strategie verringert die Anzahl der Redizes maximaler Tiefe
 - Sei $r = (\lambda x. u) v$ das rightmost-maxdepth Redex von t
 - r wird reduziert zu dem Term $u[v/x]$ ohne Redizes maximaler Tiefe

RIGHTMOST-MAXDEPTH STRATEGIE

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

- **Lemma:** Die Strategie verringert die Anzahl der Redizes maximaler Tiefe
 - Sei $r = (\lambda x. u) v$ das rightmost-maxdepth Redex von t
 - r wird reduziert zu dem Term $u[v/x]$ ohne Redizes maximaler Tiefe
 - Redizes von t außerhalb von r bleiben unverändert



RIGHTMOST-MAXDEPTH STRATEGIE

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

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

RIGHTMOST-MAXDEPTH STRATEGIE

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

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

RIGHTMOST-MAXDEPTH STRATEGIE

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

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



Jeder typisierbare λ -Term ist normalisierbar

STARKE NORMALISIERBARKEIT

Führt jede Reduktionsfolge zu einer Normalform?

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
 - $t \in S \rightarrow T$ berechenbar, falls $t s$ berechenbar für berechenbare $s \in S$

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
 - $t \in S \rightarrow T$ berechenbar, falls $t s$ berechenbar für berechenbare $s \in S$
 - Mehr als nur SN: jede Anwendung von t 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
 - $t \in S \rightarrow T$ 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'

NACHWEIS DER STARKEN NORMALISIERBARKEIT

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

1. t berechenbar $\Rightarrow t$ stark normalisierbar

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

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

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

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

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

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 $t=t[x_1 \dots x_n]$

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 $t=t[x_1 \dots x_n]$

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

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 $t=t[x_1 \dots x_n]$

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

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 $t = t[x_1 \dots x_n]$

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

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 $t = t[x_1 \dots x_n]$

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



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 $t = t[x_1 \dots x_n]$

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



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'

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:**

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)



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 $t', t'_1, t'_2, \dots$ terminiert, da $t, t', t'_1, t'_2, \dots$ terminiert)

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 $t', t'_1, t'_2, \dots$ terminiert, da $t, t', t'_1, t'_2, \dots$ terminiert)
3. t neutral, $t \xrightarrow{\beta} t'$ impliziert t' berechenbar, dann t SN und berechenbar ✓
(Jede Reduktionsfolge $t, t', t'', t''', \dots$ terminiert, da $t', t'', t''', \dots$ terminiert)

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

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

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

- Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2
 1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

- Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2
 1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

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

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

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

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

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

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

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

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



STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

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

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$.



STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

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

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ berechenbar und $ts \xrightarrow{\beta} t's$, also $t's \in T_2$ berechenbar (Induktionsannahme 2.),



STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

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

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar ✓
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ 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 $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar ✓
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ 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.

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

• Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 $\mapsto t$ stark normalisierbar ✓
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 $\mapsto ts \in T_2$ 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

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

• Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar 
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ 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

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

• Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar ✓
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ 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 ts kein Redex

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

• Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar ✓
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ 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 ts kein Redex
 - Falls $t^* = t' s$ mit $t \xrightarrow{\beta} t'$, dann t^* berechenbar

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

• Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar ✓
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ 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 ts 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

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

• Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar ✓
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ 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 ts 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 $ts' \xrightarrow{\beta} t^+$ impliziert t^+ berechenbar (innere Annahme) und $ts' \in T_2$ neutral also $t^* \equiv ts'$ berechenbar (Induktionsannahme 3.)

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

• Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar ✓
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ 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 ts 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
 $ts' \xrightarrow{\beta} t^+$ impliziert t^+ berechenbar (innere Annahme) und $ts' \in T_2$ neutral
 also $t^* \equiv ts'$ berechenbar (Induktionsannahme 3.)
 - $\mapsto ts \in T_2$ neutral, also berechenbar (Induktionsannahme 3.)

STARKE NORMALISIERBARKEIT: BEWEIS VON 1.–4. (II)

• Sei $T = T_1 \rightarrow T_2$, Behauptung gelte für Terme in T_1 und T_2

1. Sei t berechenbar, $x \in T_1$ Variable, $t \xrightarrow{\beta} t_1 \xrightarrow{\beta} t_2 \xrightarrow{\beta} \dots$ Reduktionsfolge
 - $\mapsto x$ neutral, in Normalform, also berechenbar (Induktionsannahme 4.)
 - $\mapsto tx \in T_2$ berechenbar (per Definition), also stark normalisierbar (Induktionsannahme 1.)
 - $\mapsto$ Reduktionsfolge $tx, t_1x, t_2x, \dots$ terminiert, also auch $t, t_1, t_2, \dots$
 - $\mapsto t$ stark normalisierbar ✓
2. Sei t berechenbar, es gelte $t \xrightarrow{\beta} t'$. Sei $s \in T_1$ berechenbar
 - $\mapsto ts \in T_2$ 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 ts 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 $ts' \xrightarrow{\beta} t^+$ impliziert t^+ berechenbar (innere Annahme) und $ts' \in T_2$ neutral also $t^* \equiv ts'$ berechenbar (Induktionsannahme 3.)
 - $\mapsto ts \in T_2$ 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

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

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)

STARKE NORMALISIERBARKEIT: BEWEIS VON 5.

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

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

– Zu zeigen ist $(\lambda x.t) s$ berechenbar für berechenbare s

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

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

STARKE NORMALISIERBARKEIT: BEWEIS VON 5.

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

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

- Zu zeigen ist $(\lambda x.t) s$ berechenbar für berechenbare s
bzw., da $(\lambda x.t) s$ neutral: $(\lambda x.t) s \xrightarrow{\beta} t^* \Rightarrow t^*$ berechenbar (nach 3)
- t ist berechenbar, denn $t = t[x/x]$ und jedes $x \in \mathcal{V}$ ist berechenbar nach (4)
- Also t und s stark normalisierbar nach (1)

STARKE NORMALISIERBARKEIT: BEWEIS VON 5.

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

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

– Zu zeigen ist $(\lambda x.t) s$ berechenbar für berechenbare s

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

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

– Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Zahl der Reduktionsschritte 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 jedes $x \in \mathcal{V}$ ist berechenbar nach (4)

– Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Zahl der Reduktionsschritte von t und s

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

STARKE NORMALISIERBARKEIT: BEWEIS VON 5.

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

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

– Zu zeigen ist $(\lambda x.t) s$ berechenbar für berechenbare s

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

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

– Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Zahl der Reduktionsschritte von t und s

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

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

STARKE NORMALISIERBARKEIT: BEWEIS VON 5.

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

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

– Zu zeigen ist $(\lambda x.t) s$ berechenbar für berechenbare s

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

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

– Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Zahl der Reduktionsschritte von t und s

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

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

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

STARKE NORMALISIERBARKEIT: BEWEIS VON 5.

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

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

- Zu zeigen ist $(\lambda x.t) s$ berechenbar für berechenbare s
bzw., da $(\lambda x.t) s$ neutral: $(\lambda x.t) s \xrightarrow{\beta} t^* \Rightarrow t^*$ berechenbar (nach 3)
- t ist berechenbar, denn $t = t[x/x]$ und jedes $x \in \mathcal{V}$ ist berechenbar nach (4)
- Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Zahl der Reduktionsschritte von t und s

- Falls $t^* = t[s/x]$, dann t^* berechenbar ($\mapsto$ 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')

STARKE NORMALISIERBARKEIT: BEWEIS VON 5.

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

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

- Zu zeigen ist $(\lambda x.t) s$ berechenbar für berechenbare s
bzw., da $(\lambda x.t) s$ neutral: $(\lambda x.t) s \xrightarrow{\beta} t^* \Rightarrow t^*$ berechenbar (nach 3)
- t ist berechenbar, denn $t = t[x/x]$ und jedes $x \in \mathcal{V}$ ist berechenbar nach (4)
- Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Zahl der Reduktionsschritte von t und s

- Falls $t^* = t[s/x]$, dann t^* berechenbar ($\mapsto$ 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) ($\mapsto$ Äußerer Induktionsschritt für t) ✓

STARKE NORMALISIERBARKEIT: BEWEIS VON 5.

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

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

– Zu zeigen ist $(\lambda x.t) s$ berechenbar für berechenbare s

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

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

– Also t und s stark normalisierbar nach (1)

Beweis durch Doppelinduktion über Zahl der Reduktionsschritte von t und s

• Falls $t^* = t[s/x]$, dann t^* berechenbar ($\mapsto$ 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) ($\mapsto$ 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) ($\mapsto$ Innere Induktionsschritte für s) ✓

STARKE NORMALISIERBARKEIT: BEWEIS VON 6.

t typisierbar, $x_1 \dots x_n$ (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, $x_1 \dots x_n$ (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, $x_1 \dots x_n$ (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 $t[b_1..b_n / x_1..x_n] = \lambda x.u[b_1..b_n / x_1..x_n]$



STARKE NORMALISIERBARKEIT: BEWEIS VON 6.

t typisierbar, $x_1 \dots x_n$ (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)

STARKE NORMALISIERBARKEIT: BEWEIS VON 6.

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

STARKE NORMALISIERBARKEIT: BEWEIS VON 6.

t typisierbar, $x_1 \dots x_n$ (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)

STARKE NORMALISIERBARKEIT: BEWEIS VON 6.

t typisierbar, $x_1 \dots x_n$ (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( $\lambda x.x$ ))`

STARKE NORMALISIERBARKEIT: VORTEILE

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

$$\begin{array}{l} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{1} \text{trice}(\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x))) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{l} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{2} \text{trice}(\lambda x. (\lambda x.x) ((\lambda x.x) x)) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{3} \text{trice}(\lambda x. (\lambda x.x) x) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{4} \text{trice}(\lambda x.x) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

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

$$\begin{array}{l} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{5} \lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x)) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

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

$$\begin{array}{l} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{6} \lambda x. (\lambda x.x) ((\lambda x.x) x) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

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

$$\begin{array}{l} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{7} \lambda x. (\lambda x.x) x \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

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

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

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

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{1} \lambda x. (\text{trice}(\lambda x.x)) ((\text{trice}(\lambda x.x)) ((\text{trice}(\lambda x.x)) x)) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{2} \lambda x. (\lambda x. (\lambda x.x) ((\lambda x.x) ((\lambda x.x) x))) ((\text{trice}(\lambda x.x)) \\ \quad ((\text{trice}(\lambda x.x)) x)) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{3} \lambda x. ((\lambda x.x) ((\lambda x.x) ((\lambda x.x) (\text{trice}(\lambda x.x))))) \\ \quad ((\text{trice}(\lambda x.x)) x) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{4} \lambda x. ((\lambda x.x) ((\lambda x.x) (\text{trice}(\lambda x.x)))) ((\text{trice}(\lambda x.x)) x) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{5} \lambda x. ((\lambda x.x) (\text{trice}(\lambda x.x))) ((\text{trice}(\lambda x.x)) x) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{6} \lambda x. (\text{trice}(\lambda x.x)) ((\text{trice}(\lambda x.x)) x) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{7} \lambda x. (\text{trice}(\lambda x.x)) ((\text{trice}(\lambda x.x)) x) \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{13} \lambda x. (\text{trice}(\lambda x.x)) x \end{array}$$

STARKE NORMALISIERBARKEIT: VORTEILE

- **Jede Reduktionsstrategie** kann angewandt werden
 - Terminierung ist immer gesichert
 - Es gibt effizientere (“riskantere”) Strategien als Rightmost-Maxdepth

- **Rightmost (Call-by-value):**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{8} \lambda x.x \end{array}$$

- **Rightmost-maxdepth:**

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{14} \lambda x.x \end{array}$$

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

$$\begin{array}{c} \text{trice(trice}(\lambda x.x)) \\ \xrightarrow{19} \lambda x.x \end{array}$$

KONFLUENZ EINER REDUKTIONSRRELATION $\xrightarrow{r}$

Ist die Normalform eines Terms eindeutig?

KONFLUENZ EINER REDUKTIONSRRELATION $\xrightarrow{r}$

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 $u \xrightarrow{n} s$

KONFLUENZ EINER REDUKTIONSRRELATION $\xrightarrow{r}$

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 $t \xrightarrow{n} s$

KONFLUENZ EINER REDUKTIONSRRELATION $\xrightarrow{r}$

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 REDUKTIONSRRELATION $\longrightarrow^r$

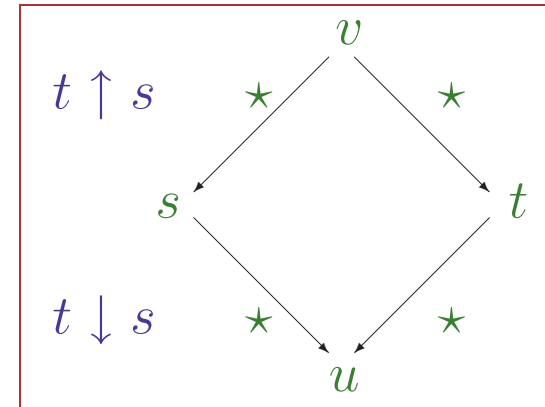
Ist die Normalform eines Terms eindeutig?

● Iteration einer Reduktionsrelation $\longrightarrow^r$

- $t \xrightarrow{n} s$: 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 REDUKTIONSRRELATION $\longrightarrow^r$

Ist die Normalform eines Terms eindeutig?

● Iteration einer Reduktionsrelation $\longrightarrow^r$

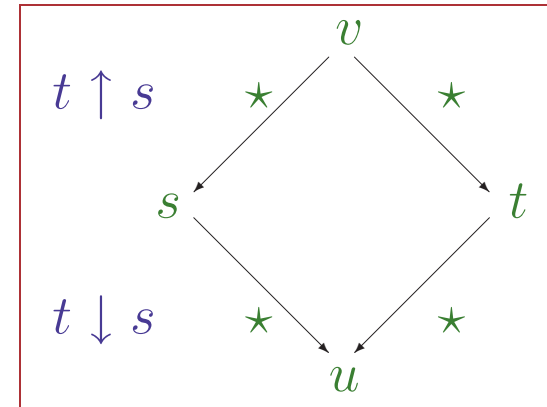
- $t \xrightarrow{n} s$: s ergibt sich aus t durch n Reduktionsschritte
 - $t \xrightarrow{0} s$, falls $t=s$, $t \xrightarrow{n+1} s$, falls es ein u gibt mit $t \xrightarrow{r} u$ und $u \xrightarrow{n} s$
- $t \xrightarrow{*} s$: es gibt ein $n \in \mathbb{N}$ mit $t \xrightarrow{n} s$
- $t \xrightarrow{+}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

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

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

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

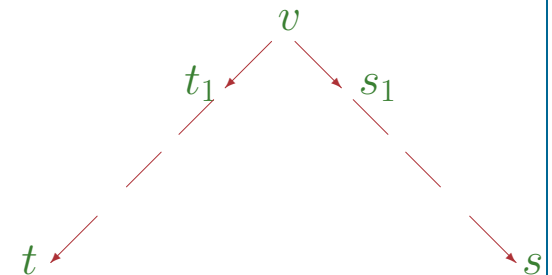
- Für alle v folgt $t \downarrow s$ aus $v \xrightarrow{*} t$ und $v \xrightarrow{*} s$

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 $v \xrightarrow{*} s$

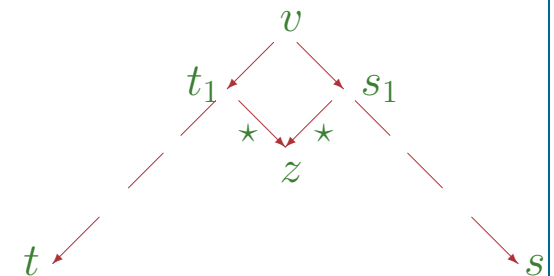
Induktion über maximale Zahl n der Reduktionsschritte

$n=0$: 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

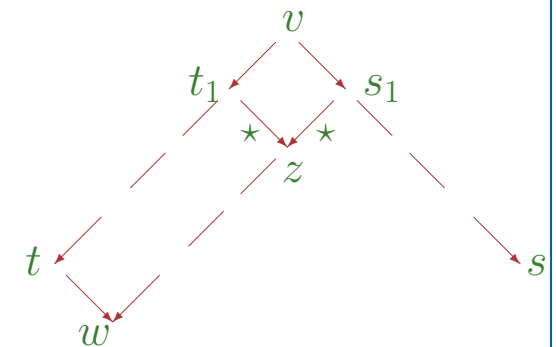
$n=0$: Es folgt $v=t=s$, also $t \downarrow s$ trivialerweise ✓

$n+1$: 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 ✓

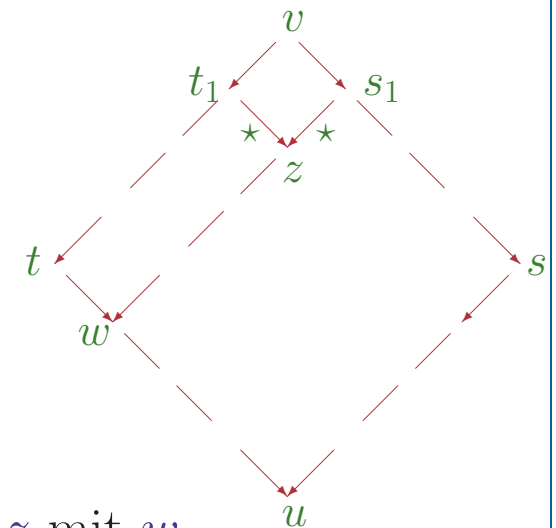
$n+1$: Falls $v \xrightarrow{0} t$ oder $v \xrightarrow{0} s$, dann $v=t$ oder $v=s$ ✓

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 $v=s$ ✓

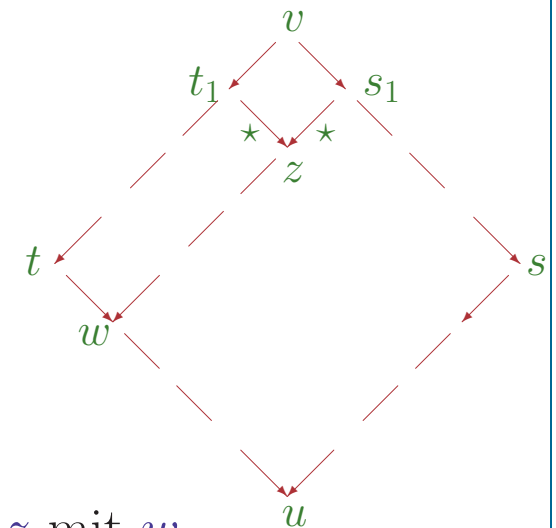
Ansonsten $v \xrightarrow{r} t_1 \xrightarrow{*} t$ und $v \xrightarrow{r} s_1 \xrightarrow{*} s$

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 $t \downarrow s$ ✓



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 $s \neq t$

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

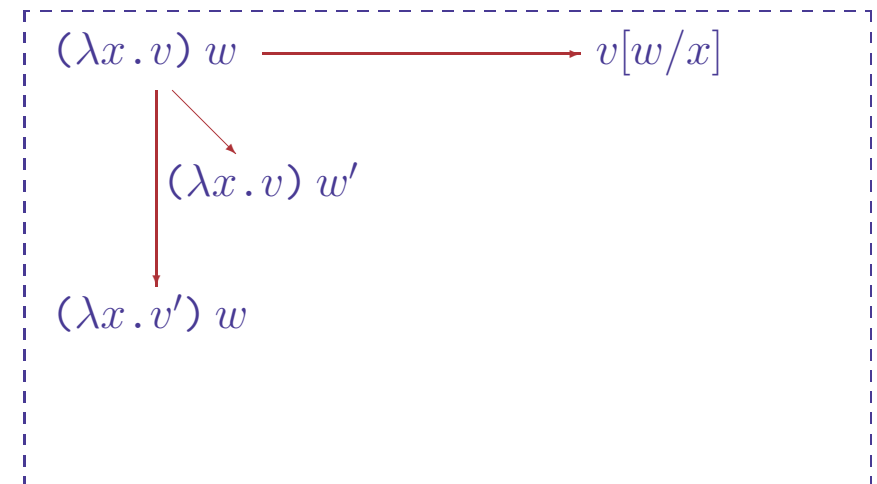
In der einfachen Typentheorie ist $\xrightarrow{\beta}$ konfluent

- $\xrightarrow{\beta}$ 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



CHURCH-ROSSER THEOREM

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

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

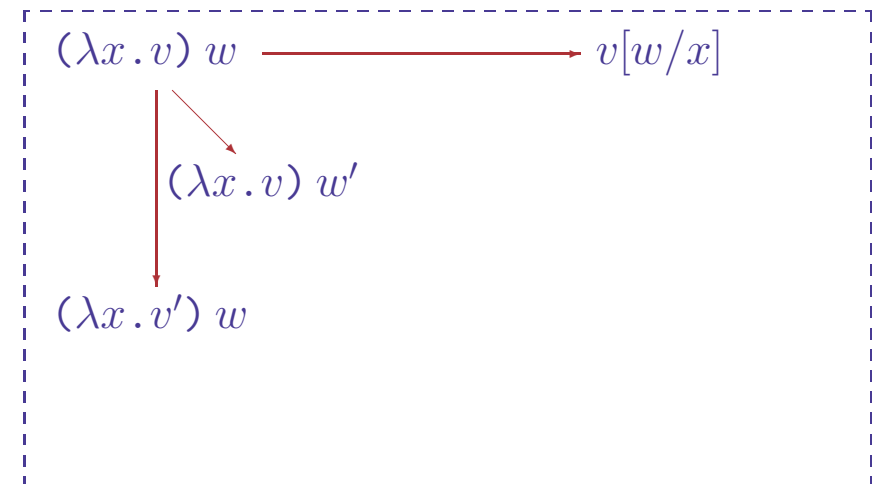
Es gelte $u \xrightarrow{\beta} t$, $u \xrightarrow{\beta} s$ und $s \neq t$

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



CHURCH-ROSSER THEOREM

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

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

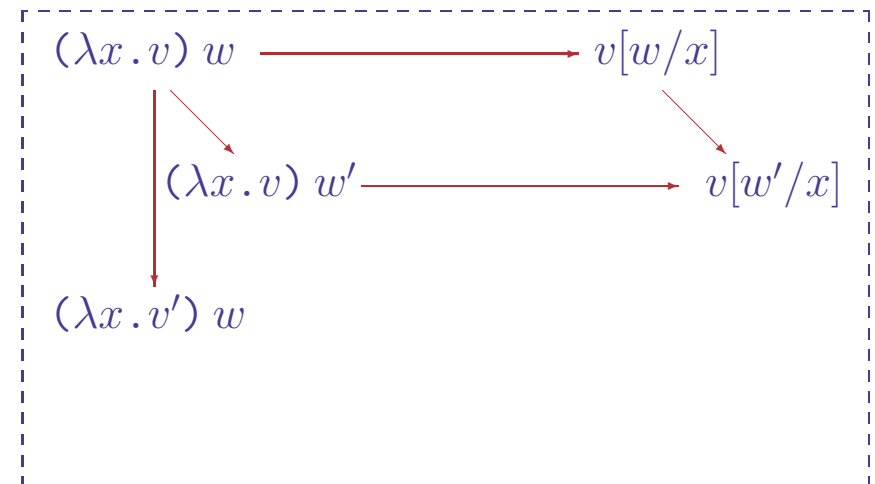
Es gelte $u \xrightarrow{\beta} t$, $u \xrightarrow{\beta} s$ und $s \neq t$

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



CHURCH-ROSSER THEOREM

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

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

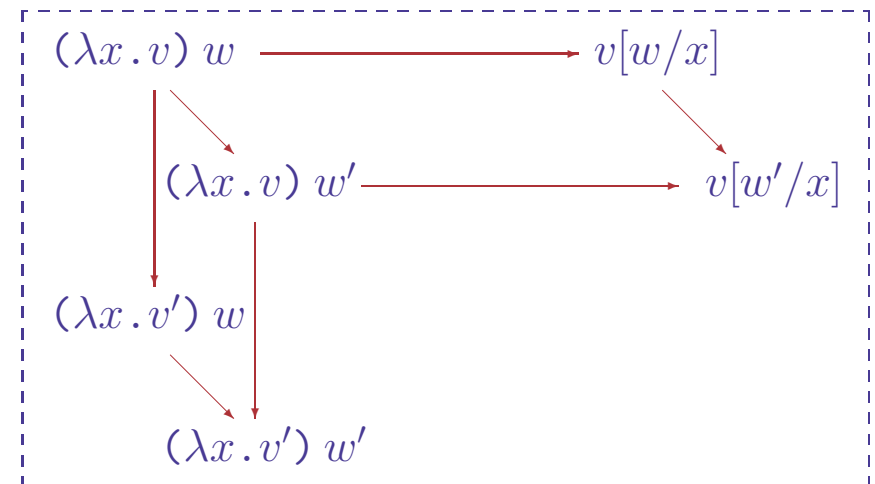
Es gelte $u \xrightarrow{\beta} t$, $u \xrightarrow{\beta} s$ und $s \neq t$

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



CHURCH-ROSSER THEOREM

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

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

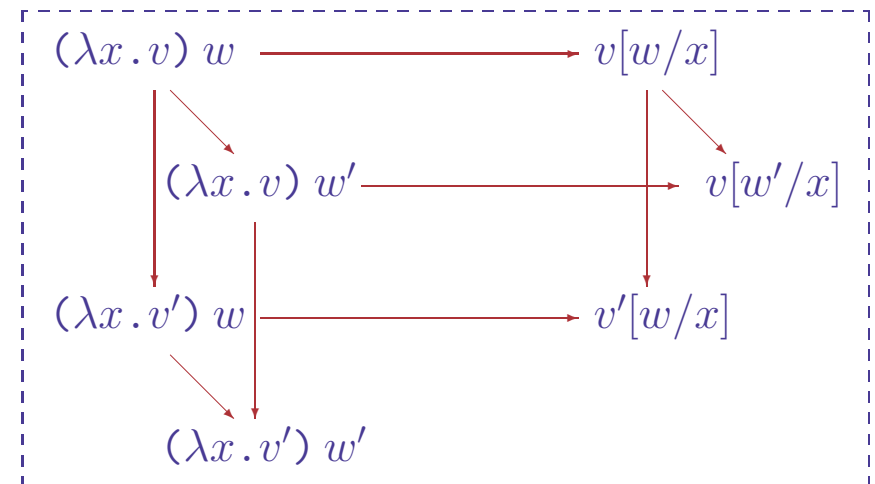
Es gelte $u \xrightarrow{\beta} t$, $u \xrightarrow{\beta} s$ und $s \neq t$

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



CHURCH-ROSSER THEOREM

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

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

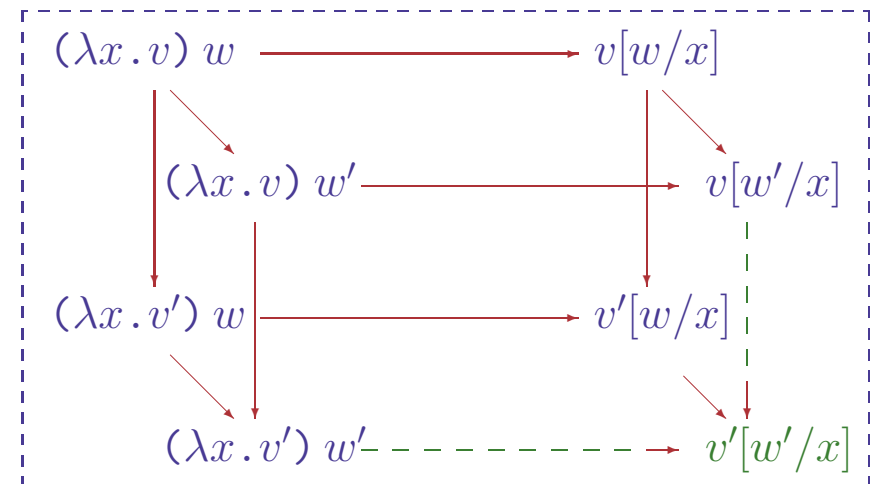
Es gelte $u \xrightarrow{\beta} t$, $u \xrightarrow{\beta} s$ und $s \neq t$

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



CHURCH-ROSSER THEOREM

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

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

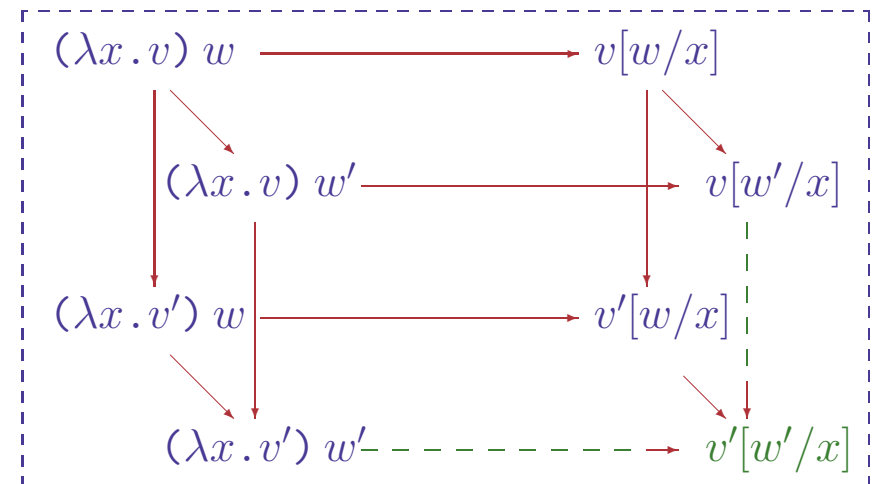
Es gelte $u \xrightarrow{\beta} t$, $u \xrightarrow{\beta} s$ und $s \neq t$

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

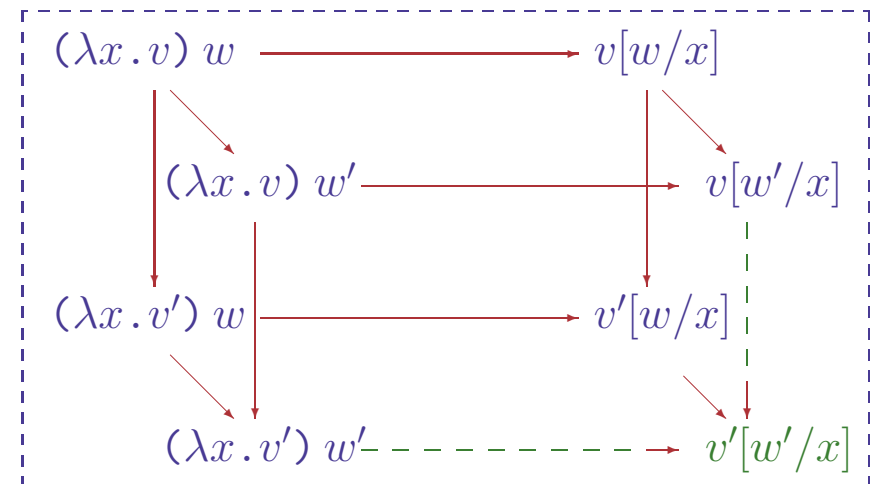
Es gelte $u \xrightarrow{\beta} t$, $u \xrightarrow{\beta} s$ und $s \neq t$

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



- Diamond Lemma + starke Normalisierbarkeit $\mapsto$ Konfluenz



Typisierbare λ -Terme haben eindeutige Normalformen

ENTSCHEIDBARKEIT

Welche Eigenschaften typisierbarer Terme sind entscheidbar?

- **Typisierbarkeit**
 - Hindley-Milner Algorithmus

ENTSCHEIDBARKEIT

Welche Eigenschaften typisierbarer Terme sind entscheidbar?

- **Typisierbarkeit**
 - Hindley-Milner Algorithmus
- **Halteproblem und Totalität von Funktionen**
 - Trivial wegen starker Normalisierbarkeit

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

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

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

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?

Welche Funktionen können durch
typisierbare λ -Terme berechnet werden?

Welche Funktionen können durch
typisierbare λ -Terme berechnet werden?

- Church-Numerals sind typisierbar

- $\overline{n} \equiv \lambda f. \lambda x. f^n x$

Welche Funktionen können durch typisierbare λ -Terme berechnet werden?

- Church-Numerals sind typisierbar

- $\overline{n} \equiv \lambda f. \lambda x. f^n x \in (X \rightarrow X) \rightarrow X \rightarrow X$

Welche Funktionen können durch
typisierbare λ -Terme berechnet werden?

- Church-Numerals sind typisierbar

$$- \overline{n} \equiv \lambda f. \lambda x. f^n x \in (X \rightarrow X) \rightarrow X \rightarrow X \equiv \mathbb{N}$$

Welche Funktionen können durch typisierbare λ -Terme berechnet werden?

- Church-Numerals sind typisierbar

- $\overline{n} \equiv \lambda f. \lambda x. f^n x \in (X \rightarrow X) \rightarrow X \rightarrow X \equiv \mathbb{N}$
- $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n f (f x) \in \mathbb{N} \rightarrow \mathbb{N}$

Welche Funktionen können durch typisierbare λ -Terme berechnet werden?

● Church-Numerals sind typisierbar

- $\overline{n} \equiv \lambda f. \lambda x. f^n x \in (X \rightarrow X) \rightarrow X \rightarrow X \equiv \mathbb{N}$
- $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n f (f x) \in \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{add} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x) \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{mul} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m (n f) x \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$

Welche Funktionen können durch typisierbare λ -Terme berechnet werden?

- Church-Numerals sind typisierbar

- $\overline{n} \equiv \lambda f. \lambda x. f^n x \in (X \rightarrow X) \rightarrow X \rightarrow X \equiv \mathbb{N}$
- $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n f (f x) \in \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{add} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x) \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{mul} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m (n f) x \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$

- Produktbildung ist typisierbar

- $\langle s, t \rangle \equiv \lambda p. p s t$

Welche Funktionen können durch typisierbare λ -Terme berechnet werden?

- Church-Numerals sind typisierbar

- $\overline{n} \equiv \lambda f. \lambda x. f^n x \in (X \rightarrow X) \rightarrow X \rightarrow X \equiv \mathbb{N}$
- $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n f (f x) \in \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{add} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x) \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{mul} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m (n f) x \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$

- Produktbildung ist typisierbar

- $\langle s, t \rangle \equiv \lambda p. p s t \in (S \rightarrow T \rightarrow Z) \rightarrow Z$

Welche Funktionen können durch typisierbare λ -Terme berechnet werden?

- Church-Numerals sind typisierbar

- $\overline{n} \equiv \lambda f. \lambda x. f^n x \in (X \rightarrow X) \rightarrow X \rightarrow X \equiv \mathbb{N}$
- $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n f (f x) \in \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{add} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x) \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{mul} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m (n f) x \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$

- Produktbildung ist typisierbar

- $\langle s, t \rangle \equiv \lambda p. p s t \in (S \rightarrow T \rightarrow Z) \rightarrow Z \equiv S \times T$

Welche Funktionen können durch typisierbare λ -Terme berechnet werden?

● Church-Numerals sind typisierbar

- $\overline{n} \equiv \lambda f. \lambda x. f^n x \in (X \rightarrow X) \rightarrow X \rightarrow X \equiv \mathbb{N}$
- $\mathbf{s} \equiv \lambda n. \lambda f. \lambda x. n f (f x) \in \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{add} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x) \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$
- $\mathbf{mul} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m (n f) x \in \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$

● Produktbildung ist typisierbar

- $\langle s, t \rangle \equiv \lambda p. p s t \in (S \rightarrow T \rightarrow Z) \rightarrow Z \equiv S \times T$
- $\langle s, t \rangle.1 \equiv \langle s, t \rangle (\lambda x. \lambda y. x) \in S$
- $\langle s, t \rangle.2 \equiv \langle s, t \rangle (\lambda x. \lambda y. y) \in T$

AUSDRUCKSKRAFT TYPISierter TERME HAT GRENZEN

- Typisierung boolescher Operatoren problematisch

AUSDRUCKSKRAFT TYPISierter TERME HAT GRENZEN

- Typisierung boolescher Operatoren problematisch
 - $\mathbf{T} \equiv \lambda x. \lambda y. x \in X \rightarrow Y \rightarrow X$

AUSDRUCKSKRAFT TYPISierter TERME HAT GRENZEN

- Typisierung boolescher Operatoren problematisch

- **T** $\equiv \lambda x. \lambda y. x \in X \rightarrow Y \rightarrow X$

- **F** $\equiv \lambda x. \lambda y. y \in X \rightarrow Y \rightarrow Y$

AUSDRUCKSKRAFT TYPISierter TERME HAT GRENZEN

- Typisierung boolescher Operatoren problematisch

- **T** $\equiv \lambda x. \lambda y. x \in X \rightarrow Y \rightarrow X$

- **F** $\equiv \lambda x. \lambda y. y \in X \rightarrow Y \rightarrow Y$

- **if** b **then** s **else** $t \equiv b s t \in Z$ für $s \in X, t \in Y, b \in \mathbb{B} \equiv X \rightarrow Y \rightarrow Z$

AUSDRUCKSKRAFT TYPISierter TERME HAT GRENZEN

- Typisierung boolescher Operatoren problematisch

- $\mathbf{T} \equiv \lambda x. \lambda y. x \in X \rightarrow Y \rightarrow X$

- $\mathbf{F} \equiv \lambda x. \lambda y. y \in X \rightarrow Y \rightarrow Y$

- $\mathbf{if\ } b \mathbf{\ then\ } s \mathbf{\ else\ } t \equiv b\ s\ t \in Z \quad \text{für } s \in X, t \in Y, b \in \mathbb{B} \equiv X \rightarrow Y \rightarrow Z$

- Aber $\mathbf{T} \in \mathbb{B}$ und $\mathbf{F} \in \mathbb{B}$ verlangt $X=Y=Z$ und somit $\mathbb{B} \equiv X \rightarrow X \rightarrow X$

Das schränkt die Wahl möglicher Werte für s und t ein

AUSDRUCKSKRAFT TYPISierter TERME HAT GRENZEN

• Typisierung boolescher Operatoren problematisch

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

Das schränkt die Wahl möglicher Werte für s und t ein

• Church Numerals brauchen polymorphes $\mathbb{N}$

- **exp** $\equiv \lambda m. \lambda n. \lambda f. \lambda x. n \ m \ f \ x \in X \rightarrow (X \rightarrow Y \rightarrow Z \rightarrow T) \rightarrow Y \rightarrow Z \rightarrow T$
 - aber $m, n \in \mathbb{N}$ verlangt $X \rightarrow Y \rightarrow Z \rightarrow T = X$

AUSDRUCKSKRAFT TYPISierter TERME HAT GRENZEN

• Typisierung boolescher Operatoren problematisch

- $\mathbf{T} \equiv \lambda x. \lambda y. x \in X \rightarrow Y \rightarrow X$
- $\mathbf{F} \equiv \lambda x. \lambda y. y \in X \rightarrow Y \rightarrow Y$
- $\mathbf{if\ } b \mathbf{\ then\ } s \mathbf{\ else\ } t \equiv b\ s\ t \in Z$ für $s \in X, t \in Y, b \in \mathbb{B} \equiv X \rightarrow Y \rightarrow Z$
- Aber $\mathbf{T} \in \mathbb{B}$ und $\mathbf{F} \in \mathbb{B}$ verlangt $X=Y=Z$ und somit $\mathbb{B} \equiv X \rightarrow X \rightarrow X$

Das schränkt die Wahl möglicher Werte für s und t ein

• Church Numerals brauchen polymorphes $\mathbb{N}$

- $\mathbf{exp} \equiv \lambda m. \lambda n. \lambda f. \lambda x. n\ m\ f\ x \in X \rightarrow (X \rightarrow Y \rightarrow Z \rightarrow T) \rightarrow Y \rightarrow Z \rightarrow T$
 - aber $m, n \in \mathbb{N}$ verlangt $X \rightarrow Y \rightarrow Z \rightarrow T = X$
- $\mathbf{PRs}[base, h] \equiv \lambda n. n\ h\ base \in (X \rightarrow Y \rightarrow Z) \rightarrow Z$
 - aber $h \in Y = \mathbb{N} \rightarrow \mathbb{N}$ und $base \in Z = \mathbb{N}$ verlangt $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N} = \mathbb{N}$

AUSDRUCKSKRAFT TYPISierter TERME HAT GRENZEN

● Typisierung boolescher Operatoren problematisch

- $\mathbf{T} \equiv \lambda x. \lambda y. x \in X \rightarrow Y \rightarrow X$
- $\mathbf{F} \equiv \lambda x. \lambda y. y \in X \rightarrow Y \rightarrow Y$
- $\mathbf{if\ } b \mathbf{\ then\ } s \mathbf{\ else\ } t \equiv b\ s\ t \in Z$ für $s \in X, t \in Y, b \in \mathbb{B} \equiv X \rightarrow Y \rightarrow Z$
- Aber $\mathbf{T} \in \mathbb{B}$ und $\mathbf{F} \in \mathbb{B}$ verlangt $X=Y=Z$ und somit $\mathbb{B} \equiv X \rightarrow X \rightarrow X$

Das schränkt die Wahl möglicher Werte für s und t ein

● Church Numerals brauchen polymorphes $\mathbb{N}$

- $\mathbf{exp} \equiv \lambda m. \lambda n. \lambda f. \lambda x. \ n\ m\ f\ x \in X \rightarrow (X \rightarrow Y \rightarrow Z \rightarrow T) \rightarrow Y \rightarrow Z \rightarrow T$
 - aber $m, n \in \mathbb{N}$ verlangt $X \rightarrow Y \rightarrow Z \rightarrow T = X$
- $\mathbf{PRs}[base, h] \equiv \lambda n. \ n\ h\ base \in (X \rightarrow Y \rightarrow Z) \rightarrow Z$
 - aber $h \in Y = \mathbb{N} \rightarrow \mathbb{N}$ und $base \in Z = \mathbb{N}$ verlangt $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N} = \mathbb{N}$



Typstruktur zu schwach für praktische Anwendungen

SCHLIESSEN ÜBER PROGRAMMIERUNG

- **Prädikatenlogik:** Analyse der logischen **Struktur**
 - Bedeutung von Konzepten, Termen und Datentypen spielt keine Rolle

SCHLIESSEN ÜBER PROGRAMMIERUNG

- **Prädikatenlogik:** Analyse der logischen **Struktur**
 - Bedeutung von Konzepten, Termen und Datentypen spielt keine Rolle
- **λ -Kalkül:** **Wert** von Ausdrücken
 - Sehr ausdrucksstark, Auswertung nur unzureichend kontrollierbar

SCHLIESSEN ÜBER PROGRAMMIERUNG

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

SCHLIESSEN ÜBER PROGRAMMIERUNG

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



Verfeinere die Typstruktur der Theorie

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