Einführung in die Funktionale Programmierung:

Typisierung

funktionaler Programme

Prof Dr. Manfred Schmidt-Schauß

WS 2024/25

Stand der Folien: 21. Januar 2025

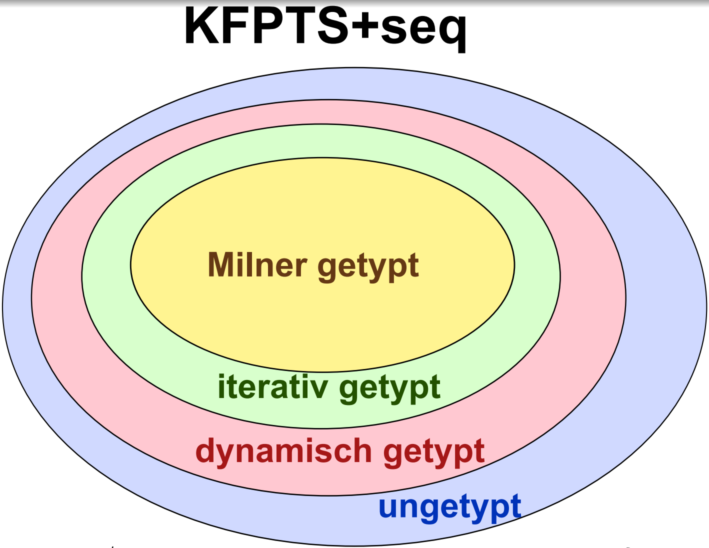
Übersicht

1. Motivation
2. Typen: Sprechweisen, Notationen und Unifikation
3. Typisierungsverfahren

Ziele des Kapitels

- Motiviation: Warum typisieren?
- Typisierungsverfahren für Haskell bzw. KFPTS+seq für parametrisch polymorphe Typen ohne Superkombinatoren
- Typisierung mit Superkombinatoren
  - Iteratives Typisierungsverfahren
  - Milnersches Typisierungsverfahren

Übersicht: Ausdrücke und Typen



Milner / iterativ getypt: syntaktische Eigenschaft
dynamisch (un-)/getypt: Laufzeit-Eigenschaft

Motivation

Warum ist ein Typsystem sinnvoll?

- Für ungetypte Programme können dynamische Typfehler auftreten
- Typfehler zur Laufzeit sind Programmierfehler
- Starkes und statisches Typsystem
  $\Longrightarrow$ keine Typfehler zu Laufzeit
- Typen als Dokumentation
- Typen bewirken besser strukturierte Programme
- Typen als Spezifikation in der Entwurfsphase

Motivation (2)

Minimalanforderungen:

- Die Typisierung sollte zur Compilezeit entschieden werden.
- Korrekt getypte Programme erzeugen keine Typfehler zur Laufzeit.

Motivation (2)

Minimalanforderungen:

- Die Typisierung sollte zur Compilezeit entschieden werden.
- Korrekt getypte Programme erzeugen keine Typfehler zur Laufzeit.

Wünschenswerte Eigenschaften:

- Typsystem schränkt wenig oder gar nicht beim Programmieren ein
- Compiler kann selbst Typen berechnen = Typinferenz

Motivation (3)

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Es gibt Typsysteme, die diese Eigenschaften nicht erfüllen:

- Z.B. Simply-typed Lambda-Calculus: Getypte Sprache ist nicht mehr Turing-mächtig, da dieses Typsystem erzwingt, dass alle Programme terminieren
- Erweiterungen in Haskells Typsystem:
  Typisierung / Typinferenz ist unentscheidbar.
  U.U. terminiert der Compiler nicht!.
  Folge: mehr Vorsicht/Anforderungen an den Programmierer.
- Typsysteme mit dependent types sind aktuell im Fokus der Forschung; und werden in Haskell-ähnlichen Programmiersprachen erprobt.
  Diese sind komplexer als polymorphe Typisierung.

Naiver Ansatz: KFPTS+seq

Naive Definition von „korrekt getypt“:

> *Ein KFPTS+seq-Programm ist korrekt getypt, wenn es keine dynamischen Typfehler zur Laufzeit erzeugt.*

Naiver Ansatz: KFPTS+seq

Naive Definition von „korrekt getypt“:

> *Ein KFPTS+seq-Programm ist korrekt getypt, wenn es keine dynamischen Typfehler zur Laufzeit erzeugt.*

Funktioniert nicht gut, denn

- Dynamische Typisierung in KFPTS+seq ist unentscheidbar!
- Dynamische Typisierung gibt wenig Information über die Struktur des Programms

Unentscheidbarkeit der dynamischen Typisierung

Sei `tmEncode` eine KFPTS+seq-Funktion, die sich wie eine universelle Turingmaschine verhält:

- Eingabe: Turingmaschinenbeschreibung und Eingabe für die TM
- Ausgabe: `True`, falls die Turingmaschine anhält

Beachte: `tmEncode` ist in KFPTS+seq definierbar und nicht dynamisch ungetypt (also dynamisch getypt)

(Haskell-Programm auf der Webseite, Archiv?)

Unentscheidbarkeit der dynamischen Typisierung (2)

Für eine TM-Beschreibung b und Eingabe e sei

$$\begin{array}{rl} s := & \texttt{if tmEncode}\ b\ e \\ & \quad \texttt{then}\ \texttt{case}_{\mathsf{Bool}}\ \texttt{Nil of}\ \{\texttt{True} \rightarrow \texttt{True}; \texttt{False} \rightarrow \texttt{False}\} \\ & \quad \texttt{else}\ \texttt{case}_{\mathsf{Bool}}\ \texttt{Nil of}\ \{\texttt{True} \rightarrow \texttt{True}; \texttt{False} \rightarrow \texttt{False}\} \end{array}$$

Es gilt:

> s *ist genau dann dynamisch ungetypt, wenn die Turingmaschine* b *auf Eingabe* e *hält.*

Daher: Wenn wir dynamische Typisierung entscheiden könnten, dann auch das Halteproblem

Satz

Die dynamische Typisierung von KFPTS+seq-Programmen ist unentscheidbar.

Typen

Syntax von polymorphen Typen:

$$\mathbf{T} ::= TV \mid TC\ \mathbf{T}_1\ \ldots\ \mathbf{T}_n \mid \mathbf{T}_1 \to \mathbf{T}_2$$

wobei TV Typvariable, TC Typkonstruktor

Sprechweisen:

- Ein Basistyp ist ein nullstelliger Typkonstruktor TC.
- Ein Grundtyp (oder alternativ monomorpher Typ) ist ein Typ, der keine Typvariablen enthält.

Beispiele:

- `Int`, `Bool` und `Char` sind Basistypen.
- `[Int]` und `Char -> Int` sind Grundtypen, aber keine Basistypen.
- `[a]` und `a -> a` sind weder Basistypen noch Grundtypen.

Typen (2)

Wir verwenden für polymorphe Typen die Schreibweise mit All-Quantoren:

- Sei τ ein polymorpher Typ mit Vorkommen der Variablen $\alpha_1, \ldots, \alpha_n$
- Dann ist $\forall \alpha_1, \ldots, \alpha_n . \tau$ der all-quantifizierte Typ für τ.
- Da die Reihenfolge der α_i egal ist, verwenden wir auch $\forall \mathcal{X}.\tau$ wobei $\mathcal{X}$ Menge von Typvariablen

Später:

Allquantifizierte Typen dürfen kopiert und umbenannt werden, Typen ohne Quantor dürfen nicht umbenannt werden!

Typsubstitutionen

Eine Typsubstitution ist eine Abbildung einer endlichen Menge von Typvariablen auf Typen, Schreibweise:
$\sigma = \{\alpha_1 \mapsto \tau_1, \ldots, \alpha_n \mapsto \tau_n\}$.
Formal: Erweiterung auf Typen: σ_E: Abbildung von Typen auf Typen

$$
\begin{array}{rcl}
\sigma_E(TV) & := & \sigma(TV), \text{ falls } \sigma \text{ die Variable } TV \text{ abbildet} \\
\sigma_E(TV) & := & TV, \text{ falls } \sigma \text{ die Variable } TV \text{ nicht abbildet} \\
\sigma_E(TC\ T_1\ \ldots\ T_n) & := & TC\ \sigma_E(T_1)\ \ldots\ \sigma_E(T_n) \\
\sigma_E(T_1 \to T_2) & := & \sigma_E(T_1) \to \sigma_E(T_2)
\end{array}
$$

Wir unterscheiden im folgenden nicht zwischen σ und der Erweiterung σ_E!

Semantik eines polymorphen Typs

Grundtypen-Semantik für polymorphe Typen:

$$\mathtt{sem}(\tau) := \{\sigma(\tau) \mid \sigma(\tau) \text{ ist Grundtyp }, \sigma \text{ ist Substitution}\}$$

Entspricht der Vorstellung von schematischen Typen:

Ein polymorpher Typ ist ein
Schema für eine Menge von Grundtypen

Typregeln

Regel für Anwendung $(s\ t)$:

$$\frac{s :: T_1 \to T_2, \quad t :: T_1}{(s\ t) :: T_2}$$

Vor Anwendung der Regel muss der Argument Teil des Typs $(\mathtt{a} \to \mathtt{b}) \to [\mathtt{a}] \to [\mathtt{b}]$ von `map` instanziert werden mit dem Typ von `not`:: `Bool` $\to$ `Bool`.

$\sigma = \{\mathtt{a} \mapsto \mathtt{Bool}, \mathtt{b} \mapsto \mathtt{Bool}\}$
und ergibt insgesamt den Typ: `map not` :: $[\mathtt{Bool}] \to [\mathtt{Bool}]$.

Statt σ zu raten, kann man σ berechnen: Unifikation

Typregeln

Regel für Anwendung $(s\ t)$:

$$\frac{s :: T_1 \to T_2, \quad t :: T_1}{(s\ t) :: T_2}$$

Problem: Man muss richtige Instanz raten, z.B. Typisierung von `map not`:

`map` :: `(a` $\to$ `b)` $\to$ `[a]` $\to$ `[b]`

`not` :: `Bool` $\to$ `Bool`

Vor Anwendung der Regel muss der Argument Teil des Typs $(\mathtt{a} \to \mathtt{b}) \to [\mathtt{a}] \to [\mathtt{b}]$ von `map` instanziiert werden mit dem Typ von `not`:: `Bool` $\to$ `Bool`.

$\sigma = \{\mathtt{a} \mapsto \mathtt{Bool}, \mathtt{b} \mapsto \mathtt{Bool}\}$

und ergibt insgesamt den Typ: `map not` :: $[\mathtt{Bool}] \to [\mathtt{Bool}]$.

Statt σ zu raten, kann man σ berechnen: Unifikation

Unifikationsproblem

Unifikation wird beim Berechnen der Typen angewendet!

Definition

Ein Unifikationsproblem auf Typen ist gegeben durch eine Menge Γ von Gleichungen der Form $\tau_1 \doteq \tau_2$, wobei τ_1 und τ_2 polymorphe Typen sind.

Eine Lösung eines Unifikationsproblem Γ auf Typen ist eine Substitution σ (bezeichnet als *Unifikator*), so dass $\sigma(\tau_1) = \sigma(\tau_2)$ für alle Gleichungen $\tau_1 \doteq \tau_2$ des Problems.

Eine allgemeinste Lösung (allgemeinster Unifikator, mgu = most general unifier) von Γ ist ein Unifikator σ, so dass gilt: Für jeden anderen Unifikator ρ von Γ gibt es eine Substitution γ so dass $\rho(x) = \gamma \circ \sigma(x)$ für alle $x \in FV(\Gamma)$.

Unifikationsalgorithmus

- Datenstruktur: Γ = Multimenge von Gleichungen

  Multimenge $\equiv$ "Menge" mit evtl. mehrfachem Vorkommen von Elementen

- $\Gamma \cup \Gamma'$ sei die disjunkte Vereinigung von zwei Multimengen
- $\Gamma[\tau/\alpha]$ ist definiert als $\{s[\tau/\alpha] \doteq t[\tau/\alpha] \mid (s \doteq t) \in \Gamma\}$.

Algorithmus: Wende Schlussregeln (s.u.) solange auf Γ an, bis

- Fail auftritt, oder
- keine Regel mehr anwendbar ist

Unifikationsalgorithmus: Schlussregeln (1)

Dekomposition:

$$\textsc{Decompose1}\ \frac{\Gamma \cup \{TC\ \tau_1\ \ldots\ \tau_n \doteq TC\ \tau'_1\ \ldots\ \tau'_n\}}{\Gamma \cup \{\tau_1 \doteq \tau'_1, \ldots, \tau_n \doteq \tau'_n\}}$$

$$\textsc{Decompose2}\ \frac{\Gamma \cup \{\tau_1 \to \tau_2 \doteq \tau'_1 \to \tau'_2\}}{\Gamma \cup \{\tau_1 \doteq \tau'_1, \tau_2 \doteq \tau'_2\}}$$

Unifikationsalgorithmus: Schlussregeln (2)

Orientierung, Elimination:

$$\textsc{Orient}\ \frac{\Gamma \cup \{\tau_1 \doteq \alpha\}}{\Gamma \cup \{\alpha \doteq \tau_1\}}$$

wenn τ_1 keine Typvariable und α Typvariable

$$\textsc{Elim}\ \frac{\Gamma \cup \{\alpha \doteq \alpha\}}{\Gamma}$$

wobei α Typvariable

Unifikationsalgorithmus: Schlussregeln (3)

Einsetzung, Occurs-Check:

$$\textsc{Solve}\ \frac{\Gamma \cup \{\alpha \doteq \tau\}}{\Gamma[\tau/\alpha] \cup \{\alpha \doteq \tau\}}$$

wenn Typvariable α nicht in τ vorkommt,
aber α kommt in Γ vor

$$\textsc{OccursCheck}\ \frac{\Gamma \cup \{\alpha \doteq \tau\}}{\mathsf{Fail}}$$

wenn $\tau \neq \alpha$ und Typvariable α kommt in τ vor

Unifikationsalgorithmus: Abbruchregeln

Fail-Regeln:

$$\textsc{Fail1}\ \frac{\Gamma \cup \{(TC_1\ \tau_1\ \ldots\ \tau_n) \doteq (TC_2\ \tau'_1\ \ldots\ \tau'_m)\}}{\mathsf{Fail}}$$

wenn $TC_1 \neq TC_2$

$$\textsc{Fail2}\ \frac{\Gamma \cup \{(TC_1\ \tau_1\ \ldots\ \tau_n) \doteq (\tau'_1 \to \tau'_2)\}}{\mathsf{Fail}}$$

$$\textsc{Fail3}\ \frac{\Gamma \cup \{(\tau'_1 \to \tau'_2) \doteq (TC_1\ \tau_1\ \ldots\ \tau_n)\}}{\mathsf{Fail}}$$

Beispiele

Beispiel 1: $\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}$:

$$\textsc{Decompose2}\ \frac{\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}}{\{a \doteq \texttt{Bool}, b \doteq \texttt{Bool}\}}$$

Beispiele

Beispiel 1: $\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}$:

$$\textsc{Decompose2}\ \frac{\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}}{\{a \doteq \texttt{Bool}, b \doteq \texttt{Bool}\}}$$

Beispiel 2: $\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}$:

$$\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}$$

Beispiele

Beispiel 1: $\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}$:

$$\textsc{Decompose2}\ \frac{\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}}{\{a \doteq \texttt{Bool}, b \doteq \texttt{Bool}\}}$$

Beispiel 2: $\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}$:

$$\textsc{Decompose2}\ \frac{\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}}{\{[d] \doteq c, a \doteq \texttt{Bool}, [a] \doteq c\}}$$

Beispiele

Beispiel 1: $\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}$:

$$\textsc{Decompose2}\ \frac{\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}}{\{a \doteq \texttt{Bool}, b \doteq \texttt{Bool}\}}$$

Beispiel 2: $\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}$:

$$\textsc{Orient}\ \frac{\textsc{Decompose2}\ \frac{\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}}{\{[d] \doteq c, a \doteq \texttt{Bool}, [a] \doteq c\}}}{\{[d] \doteq c, a \doteq \texttt{Bool}, c \doteq [a]\}}$$

Beispiele

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Beispiel 1: $\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}$:

$$\textsc{Decompose2}\ \frac{\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}}{\{a \doteq \texttt{Bool}, b \doteq \texttt{Bool}\}}$$

Beispiel 2: $\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}$:

$$\begin{array}{rc} \textsc{Decompose2} & \dfrac{\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}}{\{[d] \doteq c, a \doteq \texttt{Bool}, [a] \doteq c\}} \\ \textsc{Orient} & \dfrac{}{\{[d] \doteq c, a \doteq \texttt{Bool}, c \doteq [a]\}} \\ \textsc{Solve} & \dfrac{}{\{[d] \doteq [a], a \doteq \texttt{Bool}, c \doteq [a]\}} \end{array}$$

Beispiele

Beispiel 1: $\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}$:

$$\textsc{Decompose2}\ \frac{\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}}{\{a \doteq \texttt{Bool}, b \doteq \texttt{Bool}\}}$$

Beispiel 2: $\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}$:

$$\textsc{Decompose2}\ \frac{\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}}{\textsc{Orient}\ \frac{\{[d] \doteq c, a \doteq \texttt{Bool}, [a] \doteq c\}}{\textsc{Solve}\ \frac{\{[d] \doteq c, a \doteq \texttt{Bool}, c \doteq [a]\}}{\textsc{Solve}\ \frac{\{[d] \doteq [a], a \doteq \texttt{Bool}, c \doteq [a]\}}{\{[d] \doteq [\texttt{Bool}], a \doteq \texttt{Bool}, c \doteq [\texttt{Bool}]\}}}}}$$

Beispiele

Beispiel 1: $\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}$:

$$\textsc{Decompose2}\ \frac{\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}}{\{a \doteq \texttt{Bool}, b \doteq \texttt{Bool}\}}$$

Beispiel 2: $\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}$:

$$\textsc{Decompose1}\ \cfrac{\textsc{Solve}\ \cfrac{\textsc{Solve}\ \cfrac{\textsc{Orient}\ \cfrac{\textsc{Decompose2}\ \cfrac{\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}}{\{[d] \doteq c, a \doteq \texttt{Bool}, [a] \doteq c\}}}{\{[d] \doteq c, a \doteq \texttt{Bool}, c \doteq [a]\}}}{\{[d] \doteq [a], a \doteq \texttt{Bool}, c \doteq [a]\}}}{\{[d] \doteq [\texttt{Bool}], a \doteq \texttt{Bool}, c \doteq [\texttt{Bool}]\}}}{\{d \doteq \texttt{Bool}, a \doteq \texttt{Bool}, c \doteq [\texttt{Bool}]\}}$$

Beispiele

Beispiel 1: $\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}$:

$$\textsc{Decompose2}\ \frac{\{(a \to b) \doteq \texttt{Bool} \to \texttt{Bool}\}}{\{a \doteq \texttt{Bool}, b \doteq \texttt{Bool}\}}$$

Beispiel 2: $\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}$:

$$\begin{array}{l}
\textsc{Decompose2}\ \dfrac{\{[d] \doteq c, a \to [a] \doteq \texttt{Bool} \to c\}}{\{[d] \doteq c, a \doteq \texttt{Bool}, [a] \doteq c\}} \\
\textsc{Orient}\ \dfrac{\{[d] \doteq c, a \doteq \texttt{Bool}, [a] \doteq c\}}{\{[d] \doteq c, a \doteq \texttt{Bool}, c \doteq [a]\}} \\
\textsc{Solve}\ \dfrac{\{[d] \doteq c, a \doteq \texttt{Bool}, c \doteq [a]\}}{\{[d] \doteq [a], a \doteq \texttt{Bool}, c \doteq [a]\}} \\
\textsc{Solve}\ \dfrac{\{[d] \doteq [a], a \doteq \texttt{Bool}, c \doteq [a]\}}{\{[d] \doteq [\texttt{Bool}], a \doteq \texttt{Bool}, c \doteq [\texttt{Bool}]\}} \\
\textsc{Decompose1}\ \dfrac{\{[d] \doteq [\texttt{Bool}], a \doteq \texttt{Bool}, c \doteq [\texttt{Bool}]\}}{\{d \doteq \texttt{Bool}, a \doteq \texttt{Bool}, c \doteq [\texttt{Bool}]\}}
\end{array}$$

Der Unifikator ist $\{d \mapsto \texttt{Bool}, a \mapsto \texttt{Bool}, c \mapsto [\texttt{Bool}]\}$.

Beispiele (2)

Beispiel 3: $\{a \doteq [b], b \doteq [a]\}$

$$\textsc{OccursCheck}\ \dfrac{\textsc{Solve}\ \dfrac{\{a \doteq [b], b \doteq [a]\}}{\{a \doteq [[a]], b \doteq [a]\}}}{\text{Fail}}$$

Beispiele (2)

Beispiel 3: $\{a \doteq [b], b \doteq [a]\}$

$$\textsc{OccursCheck}\ \dfrac{\textsc{Solve}\ \dfrac{\{a \doteq [b], b \doteq [a]\}}{\{a \doteq [[a]], b \doteq [a]\}}}{\text{Fail}}$$

Beispiel 4: $\{a \to [b] \doteq a \to c \to d\}$

$$\textsc{Fail2}\ \dfrac{\textsc{Elim}\ \dfrac{\textsc{Decompose2}\ \dfrac{\{a \to [b] \doteq a \to (c \to d)\}}{\{a \doteq a, [b] \doteq c \to d\}}}{\{[b] \doteq c \to d\}}}{\text{Fail}}$$

Eigenschaften des Unifikationsalgorithmus

- Der Algorithmus endet mit Fail gdw. es keinen Unifikator für die Eingabe gibt.
- Der Algorithmus endet erfolgreich gdw. es einen Unifikator für die Eingabe gibt. Das Gleichungssystem Γ ist dann von der Form

  $$\{\alpha_1 \doteq \tau_1, \ldots, \alpha_n \doteq \tau_n\},$$

  wobei α_i paarweise verschiedene Typvariablen sind und kein α_i in irgendeinem τ_j vorkommt. Der Unifikator kann dann abgelesen werden als $\sigma = \{\alpha_1 \mapsto \tau_1, \ldots, \alpha_n \mapsto \tau_n\}$.
- Liefert der Algorithmus einen Unifikator, dann ist es ein allgemeinster Unifikator.
  σ allgemeinst bedeutet: jede andere Lösung ist abgedeckt,
  d.h ist spezieller als σ,
  genauer: kann durch weitere Einsetzung aus σ erzeugt werden.

Eigenschaften des Unifikationsalgorithmus (2)

- Man braucht keine alternativen Regelanwendungen auszuprobieren! Der Algorithmus kann deterministisch implementiert werden.
- Der Algorithmus terminiert für jedes Unifikationsproblem auf Typen.
  Ausgabe: Fail oder der (bzw. ein) allgemeinste(r) Unifikator

Eigenschaften des Unifikationsalgorithmus (3)

- Die Typen in der Resultat-Substitution können exponentiell groß werden. Aber sie sind komprimiert polynomiell groß.
- Der Unifikationsalgorithmus kann aber so implementiert werden, dass er Zeit $O(n * \log n)$ benötigt. Man muss Sharing dazu beachten; Dazu eine andere Solve-Regel benutzen.
  Die Typen in der Resultat-Substitution haben danach Darstellungsgröße $O(n)$.
- Das Unifikationsproblem (d.h. die Frage, ob eine Menge von Typgleichungen unifizierbar ist) ist P-complete. D.h. man kann im wesentlichen alle PTIME-Probleme als Unifikationsproblem darstellen:
  Interpretation ist: Unifikation ist nicht effizient parallelisierbar.

Typisierungsverfahren

Wir betrachten nun die

polymorphe Typisierung

von KFPTSP+seq-Ausdrücken

später: Typisierung von Superkombinatoren

Typisierungsverfahren

Wir betrachten nun die

polymorphe Typisierung

von KFPTSP+seq-Ausdrücken

später: Typisierung von Superkombinatoren

Typisierungsmethode

Annahme:	Superkombinatoren bereits getypt
Dann:	Typisierung von Ausdrücken
Regeln:	arbeiten rekursiv auf der Struktur der Ausdrücke

Anwendungsregel mit Unifikation

$$\frac{s :: \tau_1, \quad t :: \tau_2}{(s\ t) :: \sigma(\alpha)}$$

wenn σ allgemeinster Unifikator für die Gleichung $\tau_1 \doteq \tau_2 \to \alpha$ ist und α neue Typvariable ist.

Anwendungsregel mit Unifikation

$$\frac{s :: \tau_1, \quad t :: \tau_2}{(s\ t) :: \sigma(\alpha)}$$

wenn σ allgemeinster Unifikator für die Gleichung $\tau_1 \doteq \tau_2 \to \alpha$ ist und α neue Typvariable ist.

Beispiel: `(map not)`

$$\frac{\texttt{map} :: (a \to b) \to [a] \to [b], \quad \texttt{not} :: \texttt{Bool} \to \texttt{Bool}}{(\texttt{map not}) :: \sigma(\alpha)}$$

wenn σ allgemeinster Unifikator für die Gleichung

$$(a \to b) \to ([a] \to [b]) \doteq (\texttt{Bool} \to \texttt{Bool}) \to \alpha$$

ist, und α neue Typvariable ist.

Anwendungsregel mit Unifikation

$$\frac{s :: \tau_1, \quad t :: \tau_2}{(s\ t) :: \sigma(\alpha)}$$

wenn σ allgemeinster Unifikator für die Gleichung $\tau_1 \doteq \tau_2 \to \alpha$ ist und α neue Typvariable ist.

Beispiel: `(map not)`

$$\frac{\texttt{map} :: (a \to b) \to [a] \to [b], \quad \texttt{not} :: \texttt{Bool} \to \texttt{Bool}}{(\texttt{map}\ \texttt{not}) :: \sigma(\alpha)}$$

wenn σ allgemeinster Unifikator für die Gleichung

$$(a \to b) \to ([a] \to [b]) \doteq (\texttt{Bool} \to \texttt{Bool}) \to \alpha$$

ist, und α neue Typvariable ist.

Unifikation ergibt $\{a \mapsto \texttt{Bool}, b \mapsto \texttt{Bool}, \alpha \mapsto [\texttt{Bool}] \to [\texttt{Bool}]\}$

Daher: $\sigma(\alpha) = [\texttt{Bool}] \to [\texttt{Bool}]$

Anwendungsregel mit Unifikation

Beispiel rechnen:

`map length`

$$\frac{\texttt{map} :: (a \to b) \to [a] \to [b], \quad \texttt{length} :: [a'] \to \texttt{Nat}}{(\texttt{map length}) :: \sigma(\alpha)}$$

σ ist allgemeinster Unifikator der Gleichung

$$(a \to b) \to [a] \to [b] \doteq ([a'] \to \texttt{Nat}) \to \alpha$$

und α neue Typvariable.

Anwendungsregel mit Unifikation

Beispiel rechnen:

`map length`

$$\frac{\texttt{map} :: (a \to b) \to [a] \to [b], \quad \texttt{length} :: [a'] \to \texttt{Nat}}{(\texttt{map length}) :: \sigma(\alpha)}$$

σ ist allgemeinster Unifikator der Gleichung

$$(a \to b) \to [a] \to [b] \doteq ([a'] \to \texttt{Nat}) \to \alpha$$

und α neue Typvariable.

Dann ist der Unifikator: $\sigma = \{a \to [a']; b \to \texttt{Nat}\}$

und somit $\texttt{map length} :: [[a']] \to [\texttt{Nat}]$

Typisierung mit Bindern

Wie typisiert man eine Abstraktion $\lambda x.s$?

1. Typisiere den Rumpf s
2. Wenn x nicht in s, dann typisiere $s :: \tau$
   $\lambda x.s$ hat dann einen Funktionstyp $\alpha \to \tau$.
3. Wenn x im Rumpf s vorkommt, brauchen wir $x : \tau_1$ bei der Berechnung von τ!
   und von α!!
   Beide müssen normalerweise verfeinert werden.

Typisierung mit Bindern

Wie typisiert man eine Abstraktion $\lambda x.s$?

1. Typisiere den Rumpf s
2. Wenn x nicht in s, dann typisiere $s :: \tau$
   $\lambda x.s$ hat dann einen Funktionstyp $\alpha \to \tau$.
3. Wenn x im Rumpf s vorkommt, brauchen wir $x : \tau_1$ bei der Berechnung von τ!
   und von α!!
   Beide müssen normalerweise verfeinert werden.

Typisierung mit Bindern (2)

Informelle Regel für die Abstraktion:

$$\frac{\text{Typisierung von } s \text{ unter der Annahme "} x \text{ hat Typ } \tau_1\text{" ergibt } s :: \tau}{\lambda x.s :: \tau_1 \to \tau'}$$

Woher erhalten wir τ_1?

Typisierung mit Bindern (2)

Informelle Regel für die Abstraktion:

$$\frac{\text{Typisierung von } s \text{ unter der Annahme "}x \text{ hat Typ } \tau_1\text{" ergibt } s :: \tau}{\lambda x.s :: \tau_1 \to \tau'}$$

Woher erhalten wir τ_1?

Nehme allgemeinsten Typ an für x, danach schränke durch die Berechnung von τ den Typ ein.

Beispiel:

- $\lambda x.(x\ \texttt{True})$
- Typisiere $(x\ \texttt{True})$ beginnend mit $x :: \alpha$
- Typisierung muss liefern $\alpha = \texttt{Bool} \to \alpha'$
- Typ der Abstraktion $\lambda x.(x\ \texttt{True}) :: (\texttt{Bool} \to \alpha') \to \alpha'$.

Typisierung von Ausdrücken

Erweitertes Regelformat:

$$A \vdash s :: \tau, E$$

Bedeutung:

Gegeben eine Menge A von Typ-Annahmen.
der Form $s :: \tau$, wobei s Ausdruck, τ Typ ist.
Dann kann für den Ausdruck s der Typ τ und die Typgleichungen E hergeleitet werden.
Diese Gleichungen mittels Unifikation lösen.

- In A kommen nur Typ-Annahmen für Konstruktoren, Variablen, Superkombinatoren vor.
- In E sammeln wir Gleichungen. Diese werden sofort (oder später) unifiziert.
- $\vdash$ symbolisiert den Begriff Herleitung.

Typisierung von Ausdrücken (2)

Herleitungsregeln schreiben wir in der Form

$$\frac{\text{Voraussetzung(en)}}{\text{Konsequenz}}$$

$$\frac{A_1 \vdash s_1 :: \tau_1, E_1 \qquad \ldots \qquad A_k \vdash s_k :: \tau_k, E_K}{A \vdash s :: \tau, E}$$

Andere Lesart:

- Wenn man $A \vdash s :: \tau, E$ berechnen will, muss man erst den Teil auf dem Bruchstrich berechnen.
- Die Information kann in beide Richtungen fließen !

Typisierung von Ausdrücken (2)

Vereinfachung:

Konstruktoranwendungen $(c\ s_1\ \ldots\ s_n)$ werden
während der Typisierung
wie geschachtelte Anwendungen $(((c\ s_1)\ \ldots)\ s_n))$ behandelt.

Typisierungsregeln für KFPTS+seq Ausdrücke (1)

Axiom für Variablen:

$$(\mathrm{AxV})\ \frac{}{A \cup \{x :: \tau\} \vdash x :: \tau, \emptyset}$$

Typisierungsregeln für KFPTS+seq Ausdrücke (1)

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Axiom für Variablen:

$$(\text{AxV})\ \frac{}{A \cup \{x :: \tau\} \vdash x :: \tau, \emptyset}$$

Axiom für Konstruktoren:

$$(\text{AxK})\ \frac{}{A \cup \{c :: \forall \alpha_1 \ldots \alpha_n.\tau\} \vdash c :: \tau[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n], \emptyset}$$

wobei β_i neue Typvariablen sind

- Beachte: Bei jeder Typisierung des Konstruktors c wird ein mit neuen Typ-Variablen umbenannter Typ verwendet!

Typisierungsregeln für KFPTS+seq Ausdrücke (2)

Axiom für Superkombinatoren, deren Typ schon bekannt ist:

$$(\text{AxSK})\ \frac{}{A \cup \{SK :: \forall \alpha_1 \ldots \alpha_n.\tau\} \vdash SK :: \tau[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n], \emptyset}$$

wobei β_i neue Typvariablen sind

- Beachte: Auch hier wird jedesmal ein mit neuen Variablen umbenannter Typ verwendet!

Typisierungsregeln für KFPTS+seq Ausdrücke (3)

Regel für Anwendungen:

$$(\textsc{RApp})\ \frac{A \vdash s :: \tau_1, E_1 \quad \text{und} \quad A \vdash t :: \tau_2, E_2}{A \vdash (s\ t) :: \alpha, E_1 \cup E_2 \cup \{\tau_1 \doteq \tau_2 \to \alpha\}}$$

wobei α neue Typvariable

Typisierungsregeln für KFPTS+seq Ausdrücke (3)

Regel für Anwendungen:

$$(\textsc{RApp})\ \frac{A \vdash s :: \tau_1, E_1 \quad \text{und} \quad A \vdash t :: \tau_2, E_2}{A \vdash (s\ t) :: \alpha, E_1 \cup E_2 \cup \{\tau_1 \doteq \tau_2 \to \alpha\}}$$

wobei α neue Typvariable

Regel für `seq`:

$$(\textsc{RSeq})\ \frac{A \vdash s :: \tau_1, E_1 \quad \text{und} \quad A \vdash t :: \tau_2, E_2}{A \vdash (\texttt{seq}\ s\ t) :: \tau_2, E_1 \cup E_2}$$

Typisierungsregeln für KFPTS+seq Ausdrücke (4)

Regel für Abstraktionen:

$$(\textsc{RAbs})\ \frac{A \cup \{x :: \alpha\} \vdash s :: \tau, E}{A \vdash \lambda x.s :: \alpha \to \tau, E}$$

wobei α eine neue Typvariable

In dieser Regel werden die **Annahmen** A **erweitert** .
In E stehen die Bedingungen (Gleichungen) zu den Typvariablen.
α wird normalerweise eingeschränkt.

Typisierungsregeln für KFPTS+seq Ausdrücke (5)

Typisierung eines `case`: Prinzipien

$$\begin{pmatrix}\texttt{case}_{Typ}\ s\ \texttt{of}\ \{ \\ \quad (c_1\ x_{1,1}\ \ldots\ x_{1,\mathrm{ar}(c_1)}) \rightarrow t_1; \\ \quad \ldots; \\ \quad (c_m\ x_{m,1}\ \ldots\ x_{m,\mathrm{ar}(c_m)}) \rightarrow t_m\}\end{pmatrix}$$

- Die Pattern und der Ausdruck s haben gleichen Typ.
  Der Typ muss zum Typindex am case passen
  (Haskell hat keinen Typindex an case)
- Die Ausdrücke $t_1, \ldots, t_m$ haben gleichen Typ,
  $=$ der Typ des ganzen case-Ausdrucks.

Typisierungsregeln für KFPTS+seq Ausdrücke (6)

RCase **ist die Regel für** `case`**:**

$$
\text{(RCase)}\ \frac{\begin{array}{l} A \vdash s :: \tau, E \\ \text{für alle } i = 1,\ldots,m: \\ \quad A \cup \{x_{i,1} :: \alpha_{i,1}, \ldots, x_{i,\mathrm{ar}(c_i)} :: \alpha_{i,\mathrm{ar}(c_i)}\} \vdash (c_i\ x_{i,1}\ \ldots\ x_{i,\mathrm{ar}(ci)}) :: \tau_i, E_i \\ \text{für alle } i = 1,\ldots,m: \\ \quad A \cup \{x_{i,1} :: \alpha_{i,1}, \ldots, x_{i,\mathrm{ar}(c_i)} :: \alpha_{i,\mathrm{ar}(c_i)}\} \vdash t_i :: \tau'_i, E'_i \end{array}}{A \vdash \left(\begin{array}{l} \mathtt{case}_{Typ}\ s\ \mathtt{of}\ \{ \\ \quad (c_1\ x_{1,1}\ \ldots\ x_{1,\mathrm{ar}(c_1)}) \to t_1; \\ \quad \ldots; \\ \quad (c_m\ x_{m,1}\ \ldots\ x_{m,\mathrm{ar}(c_m)}) \to t_m\} \end{array}\right) :: \alpha, E'}
$$

wobei $E' = E \cup \bigcup_{i=1}^{m} E_i \cup \bigcup_{i=1}^{m} E'_i \cup \bigcup_{i=1}^{m} \{\tau \doteq \tau_i\} \cup \bigcup_{i=1}^{m} \{\alpha \doteq \tau'_i\}$

und $\alpha_{i,j}, \alpha$ neue Typvariablen sind

Etwas komplex: Kombination von mehreren einfachen Bedingungen

Typisierungsalgorithmus für KFPTS+seq-Ausdrücke

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Algorithmus:

Sei s ein geschlossener KFPTS+seq-Ausdruck, wobei die Typen für alle in s benutzten Superkombinatoren und Konstruktoren bekannt sind. (d.h. diese Typen sind schon berechnet oder vorgegeben)

1. Starte mit Anfangsannahme A, die Typen für die Konstruktoren und die Superkombinatoren enthält.
2. Leite $A \vdash s :: \tau, E$ mit den Typisierungsregeln her.
3. Löse E mit Unifikation.
4. Wenn die Unifikation mit Fail endet, ist s nicht typisierbar; Andernfalls: Sei σ ein allgemeinster Unifikator von E, dann gilt $s :: \sigma(\tau)$.

Optimierung

Zusätzliche Regel, zum zwischendrin Unifizieren
(Oder Abbrechen mit Fail):

Typberechnung:

$$(\textsc{RUnif})\ \frac{A \vdash s :: \tau, E}{A \vdash s :: \sigma(\tau), E_\sigma}$$

wobei E_σ das gelöste Gleichungssystem zu E ist
und σ der ablesbare Unifikator ist

Wohlgetyptheit

Definition

Ein KFPTS+seq Ausdruck s ist wohl-getypt, wenn er sich mit obigem Verfahren typisieren lässt.

(Typisierung von Superkombinatoren kommt noch)

(Schwieriger!, da diese rekursiv sein können)

Beispiele: Typisierung von (Cons True Nil)

Typisierung von `Cons True Nil`

Starte mit:

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

α: Variablen; τ: Typen und E Gleichungsmengen, die berechnet werden.

$$\text{(RApp)}\ \frac{A_0 \vdash (\texttt{Cons True}) :: \tau_1, E_1, \quad A_0 \vdash \texttt{Nil} :: \tau_2, E_2}{A_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, E_1 \cup E_2 \cup \{\tau_1 \doteq \tau_2 \to \alpha_4\}}$$

Beispiele: Typisierung von (Cons True Nil)

Typisierung von `Cons True Nil`

Starte mit:

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

α: Variablen; τ: Typen und E Gleichungsmengen, die berechnet werden.

$$\text{(RApp)}\ \frac{A_0 \vdash (\texttt{Cons True}) :: \tau_1, E_1, \quad \text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{A_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, E_1 \cup \emptyset \cup \{\tau_1 \doteq [\alpha_3] \to \alpha_4\}}$$

Beispiele: Typisierung von (Cons True Nil)

Typisierung von Cons True Nil

Starte mit:

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

α: Variablen; τ: Typen und E Gleichungsmengen, die berechnet werden.

$$\text{(RApp)}\ \dfrac{\text{(RApp)}\ \dfrac{A_0 \vdash \texttt{Cons} :: \tau_3, E_3, \quad A_0 \vdash \texttt{True} :: \tau_4, E_4}{A_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\tau_3 \doteq \tau_4 \to \alpha_2\} \cup E_3 \cup E_4}\ , \quad \text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{A_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\tau_3 \doteq \tau_4 \to \alpha_2\} \cup E_3 \cup E_4 \cup \{\alpha_2 \doteq [\alpha_3] \to \alpha_4\}}$$

Beispiele: Typisierung von (Cons True Nil)

Typisierung von `Cons True Nil`

Starte mit:

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

α: Variablen; τ: Typen und E Gleichungsmengen, die berechnet werden.

$$\text{(RApp)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Cons} :: \alpha_1 \to [\alpha_1] \to [\alpha_1], \emptyset}\ ,\quad A_0 \vdash \texttt{True} :: \tau_4, E_4}{A_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \tau_4 \to \alpha_2\} \cup E_4}\ ,\quad \text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{A_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \tau_4 \to \alpha_2\} \cup E_4 \cup \{\alpha_2 \doteq [\alpha_3] \to \alpha_4\}}$$

Beispiele: Typisierung von (Cons True Nil)

Typisierung von `Cons True Nil`

Starte mit:

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

α: Variablen; τ: Typen und E Gleichungsmengen, die berechnet werden.

$$\text{(RApp)}\;\frac{\text{(RApp)}\;\dfrac{\text{(AxK)}\;\dfrac{}{A_0 \vdash \texttt{Cons} :: \alpha_1 \to [\alpha_1] \to [\alpha_1], \emptyset}\;,\;\text{(AxK)}\;\dfrac{}{A_0 \vdash \texttt{True} :: \texttt{Bool}, \emptyset}}{A_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2\}}\;,\;\text{(AxK)}\;\dfrac{}{A_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{A_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2\} \cup \{\alpha_2 \doteq [\alpha_3] \to \alpha_4\}}$$

Beispiele: Typisierung von (Cons True Nil)

Typisierung von `Cons True Nil`

Starte mit:

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

α: Variablen; τ: Typen und E Gleichungsmengen, die berechnet werden.

$$\text{(RApp)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Cons} :: \alpha_1 \to [\alpha_1] \to [\alpha_1], \emptyset}\ ,\ \text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{True} :: \texttt{Bool}, \emptyset}}{A_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2\}}\ ,\ \text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{A_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2, \alpha_2 \doteq [\alpha_3] \to \alpha_4\}}$$

Beispiele: Typisierung von (Cons True Nil)

Typisierung von Cons True Nil

Starte mit:

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

α: Variablen; τ: Typen und E Gleichungsmengen, die berechnet werden.

$$\text{(RApp)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Cons} :: \alpha_1 \to [\alpha_1] \to [\alpha_1], \emptyset}\ ,\ \text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{True} :: \texttt{Bool}, \emptyset}}{A_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2\}}\ ,\ \text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{A_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2, \alpha_2 \doteq [\alpha_3] \to \alpha_4\}}$$

Löse $\{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2, \alpha_2 \doteq [\alpha_3] \to \alpha_4\}$ mit Unifikation

Beispiele: Typisierung von (Cons True Nil)

Typisierung von Cons True Nil

Starte mit:

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

α: Variablen; τ: Typen und E Gleichungsmengen, die berechnet werden.

$$\text{(RApp)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Cons} :: \alpha_1 \to [\alpha_1] \to [\alpha_1], \emptyset}\ ,\ \text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{True} :: \texttt{Bool}, \emptyset}}{A_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2\}}\ ,\ \text{(AxK)}\ \dfrac{}{A_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{A_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2, \alpha_2 \doteq [\alpha_3] \to \alpha_4\}}$$

Löse $\{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2, \alpha_2 \doteq [\alpha_3] \to \alpha_4\}$ mit Unifikation

Ergibt: $\sigma = \{\alpha_1 \mapsto \texttt{Bool}, \alpha_2 \mapsto ([\texttt{Bool}] \to [\texttt{Bool}]), \alpha_3 \mapsto \texttt{Bool}, \alpha_4 \mapsto [\texttt{Bool}]\}$

Beispiele: Typisierung von (Cons True Nil)

Typisierung von Cons True Nil

Starte mit:

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

α: Variablen; τ: Typen und E Gleichungsmengen, die berechnet werden.

$$\dfrac{\dfrac{\dfrac{}{A_0 \vdash \texttt{Cons} :: \alpha_1 \to [\alpha_1] \to [\alpha_1], \emptyset}\text{(AxK)}\ ,\ \dfrac{}{A_0 \vdash \texttt{True} :: \texttt{Bool}, \emptyset}\text{(AxK)}}{A_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2\}}\text{(RApp)}\ ,\ \dfrac{}{A_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}\text{(AxK)}}{A_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2, \alpha_2 \doteq [\alpha_3] \to \alpha_4\}}\text{(RApp)}$$

Löse $\{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2, \alpha_2 \doteq [\alpha_3] \to \alpha_4\}$ mit Unifikation

Ergibt: $\sigma = \{\alpha_1 \mapsto \texttt{Bool}, \alpha_2 \mapsto ([\texttt{Bool}] \to [\texttt{Bool}]), \alpha_3 \mapsto \texttt{Bool}, \alpha_4 \mapsto [\texttt{Bool}]\}$

Daher (Cons True Nil) $:: \sigma(\alpha_4) = [\texttt{Bool}]$

Beispiele: Typisierung von (Cons True Nil)

Vereinfachung für die Intuition und zum selbst nachrechnen:
Typisierung von `Cons True Nil`

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

Beispiele: Typisierung von (Cons True Nil)

Vereinfachung für die Intuition und zum selbst nachrechnen:

Typisierung von `Cons True Nil`

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

$$
\begin{tikzcd}
\texttt{Cons} : & a \arrow[r] \arrow[d, equal] & {[a]} \arrow[r] \arrow[d, equal] & {[a]} \\
& \texttt{Bool} & {[b]} & \\
& \texttt{True} & \texttt{Nil} &
\end{tikzcd}
$$

Beispiele: Typisierung von (Cons True Nil)

Vereinfachung für die Intuition und zum selbst nachrechnen:

Typisierung von `Cons True Nil`

Anfangsannahme: $A_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

$$
\begin{array}{llcl}
\texttt{Cons} : & a & \longrightarrow & [a] & \longrightarrow & [a] \\
& \| & & \| & & \\
& \texttt{Bool} & & [b] & & \\
& & & & & \\
& \texttt{True} & & \texttt{Nil} & &
\end{array}
$$

Ergibt: $a = b = \texttt{Bool}$ und

Ergebnis ist $[a]$ d.h. `(Cons True Nil)`:$[\texttt{Bool}]$

Beispiele: Typisierung von $\lambda x.x$

Typisierung von $\lambda x.x$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$\text{(RAbs)}\ \frac{A_0 \cup \{x :: \alpha\} \vdash x :: \tau, E}{A_0 \vdash (\lambda x.x) :: \alpha \to \tau, E}$$

Beispiele: Typisierung von $\lambda x.x$

Typisierung von $\lambda x.x$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$\text{(RAbs)}\ \frac{\text{(AxV)}\ \overline{A_0 \cup \{x :: \alpha\} \vdash x :: \alpha, \emptyset}}{A_0 \vdash (\lambda x.x) :: \alpha \to \alpha, \emptyset}$$

Beispiele: Typisierung von $\lambda x.x$

Typisierung von $\lambda x.x$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$\dfrac{\dfrac{}{A_0 \cup \{x :: \alpha\} \vdash x :: \alpha, \emptyset}\ (\text{AxV})}{A_0 \vdash (\lambda x.x) :: \alpha \to \alpha, \emptyset}\ (\text{RAbs})$$

Nichts zu unifizieren, daher $(\lambda x.x) :: \alpha \to \alpha$

Beispiele: Typisierung von Ω

Typisierung von $(\lambda x.(x\ x))\ (\lambda y.(y\ y))$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$\text{(RApp)}\ \frac{\emptyset \vdash (\lambda x.(x\ x)) :: \tau_1, E_1, \quad \emptyset \vdash (\lambda y.(y\ y)) :: \tau_2, E_2}{\emptyset \vdash (\lambda x.(x\ x))\ (\lambda y.(y\ y)) :: \alpha_1, E_1 \cup E_2 \cup \{\tau_1 \doteq \tau_2 \to \alpha_1\}}$$

Beispiele: Typisierung von Ω

Typisierung von $(\lambda x.(x\ x))\ (\lambda y.(y\ y))$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$\text{(RApp)}\ \dfrac{\text{(RAbs)}\ \dfrac{\{x :: \alpha_2\} \vdash (x\ x) :: \tau_6, E_1}{\emptyset \vdash (\lambda x.(x\ x)) :: \alpha_2 \to \tau_6, E_1}\ ,\quad \emptyset \vdash (\lambda y.(y\ y)) :: \tau_2, E_2}{\emptyset \vdash (\lambda x.(x\ x))\ (\lambda y.(y\ y)) :: \alpha_1, E_1 \cup E_2 \cup \{\alpha_2 \to \tau_6 \doteq \tau_2 \to \alpha_1\}}$$

Beispiele: Typisierung von Ω

Typisierung von $(\lambda x.(x\ x))\ (\lambda y.(y\ y))$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$\text{(RApp)}\ \dfrac{\text{(RAbs)}\ \dfrac{\text{(RApp)}\ \dfrac{\{x :: \alpha_2\} \vdash x :: \tau_3, E_3,\ \ \{x :: \alpha_2\} \vdash x :: \tau_4, E_4,}{\{x :: \alpha_2\} \vdash (x\ x) :: \alpha_3, \{\tau_3 \doteq \tau_4 \to \alpha_3\} \cup E_3 \cup E_4}}{\emptyset \vdash (\lambda x.(x\ x)) :: \alpha_2 \to \alpha_3, \{\tau_3 \doteq \tau_4 \to \alpha_3\} \cup E_3 \cup E_4}\ ,\ \ \emptyset \vdash (\lambda y.(y\ y)) :: \tau_2, E_2}{\emptyset \vdash (\lambda x.(x\ x))\ (\lambda y.(y\ y)) :: \alpha_1, \{\tau_3 \doteq \tau_4 \to \alpha_3\} \cup E_3 \cup E_4 \cup E_2 \cup \{\alpha_2 \to \alpha_3 \doteq \tau_2 \to \alpha_1\}}$$

Beispiele: Typisierung von Ω

Typisierung von $\color{red}{(\lambda x.(x\ x))\ (\lambda y.(y\ y))}$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$
\text{(RApp)}\ \dfrac{\text{(RAbs)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxV)}\ \dfrac{}{\{x :: \alpha_2\} \vdash x :: \alpha_2, \emptyset}\ ,\ \color{red}{\{x :: \alpha_2\} \vdash x :: \tau_4, E_4},}{\{x :: \alpha_2\} \vdash (x\ x) :: \alpha_3, \{\alpha_2 \doteq \tau_4 \to \alpha_3\} \cup E_4}}{\emptyset \vdash (\lambda x.(x\ x)) :: \alpha_2 \to \alpha_3, \{\alpha_2 \doteq \tau_4 \to \alpha_3\} \cup E_4}\ ,\ \emptyset \vdash (\lambda y.(y\ y)) :: \tau_2, E_2}{\emptyset \vdash (\lambda x.(x\ x))\ (\lambda y.(y\ y)) :: \alpha_1, \{\alpha_2 \doteq \tau_4 \to \alpha_3\} \cup E_4 \cup E_2 \cup \{\alpha_2 \to \alpha_3 \doteq \tau_2 \to \alpha_1\}}
$$

Beispiele: Typisierung von Ω

Typisierung von $(\lambda x.(x\ x))\ (\lambda y.(y\ y))$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$
\text{(RApp)}\ \dfrac{\text{(RAbs)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxV)}\ \dfrac{}{\{x :: \alpha_2\} \vdash x :: \alpha_2, \emptyset}\ ,\ \text{(AxV)}\ \dfrac{}{\{x :: \alpha_2\} \vdash x :: \alpha_2, \emptyset}\ ,}{\{x :: \alpha_2\} \vdash (x\ x) :: \alpha_3, \{\alpha_2 \doteq \alpha_2 \to \alpha_3\}}}{\emptyset \vdash (\lambda x.(x\ x)) :: \alpha_2 \to \alpha_3, \{\alpha_2 \doteq \alpha_2 \to \alpha_3\}}\ ,\ \emptyset \vdash (\lambda y.(y\ y)) :: \tau_2, E_2}{\emptyset \vdash (\lambda x.(x\ x))\ (\lambda y.(y\ y)) :: \alpha_1, \{\alpha_2 \doteq \alpha_2 \to \alpha_3\} \cup E_2 \cup \{\alpha_2 \to \alpha_3 \doteq \tau_2 \to \alpha_1\}}
$$

Beispiele: Typisierung von Ω

Typisierung von $(\lambda x.(x\ x))\ (\lambda y.(y\ y))$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$
\text{(RApp)}\ \dfrac{\text{(RAbs)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxV)}\ \dfrac{}{\{x :: \alpha_2\} \vdash x :: \alpha_2, \emptyset}\ ,\ \text{(AxV)}\ \dfrac{}{\{x :: \alpha_2\} \vdash x :: \alpha_2, \emptyset}\ ,}{\{x :: \alpha_2\} \vdash (x\ x) :: \alpha_3, \{\alpha_2 \doteq \alpha_2 \to \alpha_3\}}}{\emptyset \vdash (\lambda x.(x\ x)) :: \alpha_2 \to \alpha_3, \{\alpha_2 \doteq \alpha_2 \to \alpha_3\}}\ ,\ \dfrac{\ldots}{\emptyset \vdash (\lambda y.(y\ y)) :: \tau_2, E_2}}{\emptyset \vdash (\lambda x.(x\ x))\ (\lambda y.(y\ y)) :: \alpha_1, \{\alpha_2 \doteq \alpha_2 \to \alpha_3\} \cup E_2 \cup \{\alpha_2 \to \alpha_3 \doteq \tau_2 \to \alpha_1\}}
$$

Beispiele: Typisierung von Ω

Typisierung von $(\lambda x.(x\ x))\ (\lambda y.(y\ y))$

Starte mit: Anfangsannahme: $A_0 = \emptyset$

$$\text{(RApp)}\ \dfrac{\text{(RAbs)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxV)}\ \dfrac{}{\{x :: \alpha_2\} \vdash x :: \alpha_2, \emptyset}\ ,\ \text{(AxV)}\ \dfrac{}{\{x :: \alpha_2\} \vdash x :: \alpha_2, \emptyset}\ ,}{\{x :: \alpha_2\} \vdash (x\ x) :: \alpha_3, \{\alpha_2 \doteq \alpha_2 \to \alpha_3\}}}{\emptyset \vdash (\lambda x.(x\ x)) :: \alpha_2 \to \alpha_3, \{\alpha_2 \doteq \alpha_2 \to \alpha_3\}}\ ,\ \dfrac{\dots}{\emptyset \vdash (\lambda y.(y\ y)) :: \tau_2, E_2}}{\emptyset \vdash (\lambda x.(x\ x))\ (\lambda y.(y\ y)) :: \alpha_1, \{\alpha_2 \doteq \alpha_2 \to \alpha_3\} \cup E_2 \cup \{\alpha_2 \to \alpha_3 \doteq \tau_2 \to \alpha_1\}}$$

Man sieht schon:

Die Unifikation schlägt fehl, wegen: $\alpha_2 \doteq \alpha_2 \to \alpha_3$

Daher: $(\lambda x.(x\ x))\ (\lambda y.(y\ y))$ ist nicht typisierbar!

Beachte: $(\lambda x.(x\ x))\ (\lambda y.(y\ y))$ ist nicht dynamisch ungetypt aber nicht wohl-getypt

Beispiele: Typisierung von $\lambda x.(x\ x)$

Vereinfachung für die Intuition und zum selbst Nachrechnen

Typisierung von $\lambda x.x\ x$

Typisierung kann in $\lambda x.(x\ x)$ **das** x **nur monomorph typisieren**

Betrachte die Anwendung $(x\ x)$:

Beispiele: Typisierung von $\lambda x.(x\ x)$

Vereinfachung für die Intuition und zum selbst Nachrechnen

Typisierung von $\lambda x.x\ x$

Typisierung kann in $\lambda x.(x\ x)$ **das** x **nur monomorph typisieren**

Betrachte die Anwendung $(x\ x)$:

$$
\begin{array}{lcl}
x: & a & \longrightarrow b \\
 & \| & \\
 & (a \to b) &
\end{array}
$$

Beispiele: Typisierung von $\lambda x.(x\ x)$

Vereinfachung für die Intuition und zum selbst Nachrechnen

Typisierung von $\lambda x.x\ x$

Typisierung kann in $\lambda x.(x\ x)$ **das** x **nur monomorph typisieren**

Betrachte die Anwendung $(x\ x)$:

$$
\begin{tikzcd}
x : & a \arrow[r] \arrow[d, equal] & b \\
& (a \to b) &
\end{tikzcd}
$$

Erfordert: Lösung von $a = (a \to b)$ mit Typvariaben a, b,
aber: die Gleichung hat keine Lösung !,
da das x in $\lambda x.(x\ x)$ monomorph typisiert werden muss.

Beispiele: Typisierung von $\lambda x.(x\ x)$

Vereinfachung für die Intuition und zum selbst Nachrechnen

Typisierung von $\lambda x.x\ x$

Typisierung kann in $\lambda x.(x\ x)$ **das** x **nur monomorph typisieren**

Betrachte die Anwendung $(x\ x)$:

$$
\begin{tikzcd}
x: & a \arrow[r] \arrow[d, equal] & b \\
& (a \to b) &
\end{tikzcd}
$$

Erfordert: Lösung von $a = (a \to b)$ mit Typvariaben a, b,
aber: die Gleichung hat keine Lösung !,
da das x in $\lambda x.(x\ x)$ monomorph typisiert werden muss.

Vorsicht: `(map map)` hat Typ $[a \to b] \to [[a] \to [b]]$

Beispiele: Typisierung eines Ausdrucks mit SKs (1)

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Beispiel

Annahme:

`map` und `length` sind bereits typisierte Superkombinatoren.

Wir typisieren:

$$t := \lambda xs.\texttt{case}_{\texttt{List}}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \texttt{Nil}; (\texttt{Cons}\ y\ ys) \to \texttt{map}\ \texttt{length}\ ys\}$$

Als Anfangsannahme benutzen wir:

$$\begin{aligned} A_0 = \{&\texttt{map} :: \forall a, b.(a \to b) \to [a] \to [b], \\ &\texttt{length} :: \forall a.[a] \to \texttt{Int}, \\ &\texttt{Nil} :: \forall a.[a] \\ &\texttt{Cons} :: \forall a.a \to [a] \to [a] \\ \}& \end{aligned}$$

Beispiele: Typisierung eines Ausdrucks mit SKs (2)

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Herleitungsbaum:

$$
(\text{RAbs})\;\dfrac{(\text{RCase})\;\dfrac{(\text{AxV})\,\dfrac{}{B_3}\;,\;(\text{AxK})\,\dfrac{}{B_4}\;,\;(\text{RApp})\,\dfrac{(\text{RApp})\,\dfrac{(\text{AxK})\,\dfrac{}{B_8}\;,\;(\text{AxV})\,\dfrac{}{B_9}}{B_6}\;,\;(\text{AxV})\,\dfrac{}{B_7}}{B_5}\;,\;(\text{AxK})\,\dfrac{}{B_{10}}\;,\;(\text{RApp})\,\dfrac{(\text{RApp})\,\dfrac{(\text{AxSK})\,\dfrac{}{B_{14}}\;,\;(\text{AxSK})\,\dfrac{}{B_{15}}}{B_{12}}\;,\;(\text{AxV})\,\dfrac{}{B_{13}}}{B_{11}}}{B_2}}{B_1}
$$

Beschriftungen:

$$
\begin{aligned}
B_1 = \quad & A_0 \vdash t :: \alpha_1 \to \alpha_{13},\\
& \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7,\\
& (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \texttt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12},\\
& \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}, \}\\
B_2 = \quad & A_0 \cup \{xs :: \alpha_1\} \vdash\\
& \quad \texttt{case}_{\texttt{List}}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \texttt{Nil}; (\texttt{Cons}\ y\ ys) \to \texttt{map length}\ ys\} :: \alpha_{13},\\
& \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7,\\
& (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \texttt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12},\\
& \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}, \}
\end{aligned}
$$

Beispiele: Typisierung eines Ausdrucks mit SKs (3)

Herleitungsbaum:

$$
\text{(RAbs)}\ \dfrac{\text{(RCase)}\ \dfrac{\text{(AxV)}\ \dfrac{}{B_3}\ ,\ \text{(AxK)}\ \dfrac{}{B_4}\ ,\ \text{(RApp)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxK)}\ \dfrac{}{B_8}\ ,\ \text{(AxV)}\ \dfrac{}{B_9}}{B_6}\ ,\ \text{(AxV)}\ \dfrac{}{B_7}}{B_5}\ ,\ \text{(AxK)}\ \dfrac{}{B_{10}}\ ,\ \text{(RApp)}\ \dfrac{\text{(RApp)}\ \dfrac{\text{(AxSK)}\ \dfrac{}{B_{14}}\ ,\ \text{(AxSK)}\ \dfrac{}{B_{15}}}{B_{12}}\ ,\ \text{(AxV)}\ \dfrac{}{B_{13}}}{B_{11}}}{B_2}}{B_1}
$$

Beschriftungen:

$$
\begin{aligned}
B_3 &= A_0 \cup \{xs :: \alpha_1\} \vdash xs :: \alpha_1, \emptyset\\
B_4 &= A_0 \cup \{xs :: \alpha_1\} \vdash \texttt{Nil} :: [\alpha_2], \emptyset\\
B_5 &= A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash (\texttt{Cons}\ y\ ys) :: \alpha_7,\\
&\qquad \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7\}\\
B_6 &= A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash (\texttt{Cons}\ y) :: \alpha_6,\\
&\qquad \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6\}\\
B_7 &= A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash ys :: \alpha_4, \emptyset\\
B_8 &= A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash \texttt{Cons} :: \alpha_5 \to [\alpha_5] \to [\alpha_5], \emptyset\\
B_9 &= A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash y :: \alpha_3, \emptyset\\
B_{10} &= A_0 \cup \{xs :: \alpha_1\} \vdash \texttt{Nil} :: [\alpha_{14}], \emptyset
\end{aligned}
$$

Beispiele: Typisierung eines Ausdrucks mit SKs (4)

Herleitungsbaum:

$$
\text{(RAbs)}\dfrac{\text{(RCase)}\dfrac{\text{(AxV)}\dfrac{}{B_3},\ \text{(AxK)}\dfrac{}{B_4},\ \text{(RApp)}\dfrac{\text{(RApp)}\dfrac{\text{(AxK)}\dfrac{}{B_8},\ \text{(AxV)}\dfrac{}{B_9}}{B_6},\ \text{(AxV)}\dfrac{}{B_7}}{B_5},\ \text{(AxK)}\dfrac{}{B_{10}},\ \text{(RApp)}\dfrac{\text{(RApp)}\dfrac{\text{(AxSK)}\dfrac{}{B_{14}},\ \text{(AxSK)}\dfrac{}{B_{15}}}{B_{12}},\ \text{(AxV)}\dfrac{}{B_{13}}}{B_{11}}}{B_2}}{B_1}
$$

Beschriftungen:

$$
\begin{aligned}
B_{11} = \quad & A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash (\texttt{map length})\ ys :: \alpha_{12}, \\
& \quad \{(\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \texttt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12}\} \\
B_{12} = \quad & A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash (\texttt{map length}) :: \alpha_{11}, \\
& \quad \{(\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \texttt{Int}) \to \alpha_{11}\} \\
B_{13} = \quad & A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash ys :: \alpha_4, \emptyset \\
B_{14} = \quad & A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash \texttt{map} :: (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9], \emptyset \\
B_{15} = \quad & A_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash \texttt{length} :: [\alpha_{10}] \to \texttt{Int}, \emptyset
\end{aligned}
$$

Beispiele: Typisierung eines Ausdrucks mit SKs (5)

Beschriftung unten:

$$
\begin{aligned}
B_1 = \quad & A_0 \vdash t :: \alpha_1 \to \alpha_{13}, \\
& \quad \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7, \\
& \quad (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \mathtt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12}, \\
& \quad \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}, \}
\end{aligned}
$$

Beispiele: Typisierung eines Ausdrucks mit SKs (5)

Beschriftung unten:

$$\begin{aligned} B_1 = \quad & A_0 \vdash t :: \alpha_1 \to \alpha_{13}, \\ & \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7, \\ & (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \mathtt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12}, \\ & \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}, \} \end{aligned}$$

Löse mit Unifikation:

$$\begin{aligned} & \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7, \\ & (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \mathtt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12}, \\ & \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}\} \end{aligned}$$

Beispiele: Typisierung eines Ausdrucks mit SKs (5)

Beschriftung unten:

$$\begin{array}{rl} B_1 = & A_0 \vdash t :: \alpha_1 \to \alpha_{13}, \\ & \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7, \\ & (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \mathtt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12}, \\ & \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}, \} \end{array}$$

Löse mit Unifikation:

$$\begin{array}{l} \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7, \\ (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \mathtt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12}, \\ \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}\} \end{array}$$

Ergibt:

$$\begin{array}{rcl} \sigma & = & \{\alpha_1 \mapsto [[\alpha_{10}]], \alpha_2 \mapsto [\alpha_{10}], \alpha_3 \mapsto [\alpha_{10}], \alpha_4 \mapsto [[\alpha_{10}]], \alpha_5 \mapsto [\alpha_{10}], \\ & & \alpha_6 \mapsto [[\alpha_{10}]] \to [[\alpha_{10}]], \alpha_7 \mapsto [[\alpha_{10}]], \alpha_8 \mapsto [\alpha_{10}], \alpha_9 \mapsto \mathtt{Int}, \\ & & \alpha_{11} \mapsto [[\alpha_{10}]] \to [\mathtt{Int}], \alpha_{12} \mapsto [\mathtt{Int}], \alpha_{13} \mapsto [\mathtt{Int}], \alpha_{14} \mapsto \mathtt{Int}\} \end{array}$$

Damit erhält man $t :: \sigma(\alpha_1 \to \alpha_{13}) = [[\alpha_{10}]] \to [\mathtt{Int}]$.

zur Erinnerung:

$t := \lambda xs.\mathtt{case}_{\mathtt{List}}\ xs\ \mathtt{of}\ \{\mathtt{Nil} \to \mathtt{Nil}; (\mathtt{Cons}\ y\ ys) \to \mathtt{map}\ \mathtt{length}\ ys\}$

Vereinfachung Typisierung von (map length xs)

für die Intuition und selbst Nachrechnen:

Typisierung von `map length` xs

Starte mit:

Ann: $A_0 = \{\texttt{map} :: \forall a, b.(a \to b) \to [a] \to [b], \texttt{length} :: \forall a.[a] \to \texttt{Int}\}$

Vereinfachung Typisierung von (map length xs)

für die Intuition und selbst Nachrechnen:

Typisierung von $\texttt{map length } xs$

Starte mit:

Ann: $A_0 = \{\texttt{map} :: \forall a,b.(a \to b) \to [a] \to [b], \texttt{length} :: \forall a.[a] \to \texttt{Int}\}$

Typisiere $\texttt{map}\ \ \texttt{length}\ \ xs$

$$\begin{tikzcd}
\texttt{map}: & (a \to b) \arrow[r] \arrow[d, equal] & {[a]} \arrow[r] \arrow[d, equal] & {[b]} \arrow[d, equal] \\
& ([c] \to \texttt{Int}) & {[[c]]} & {[\texttt{Int}]}
\end{tikzcd}$$

$\texttt{length}$ $\qquad xs$ $\qquad$ (Resultat)

Vereinfachung Typisierung von (map length xs)

für die Intuition und selbst Nachrechnen:

Typisierung von `map length` xs

Starte mit:

Ann: $A_0 = \{\texttt{map} :: \forall a, b.(a \to b) \to [a] \to [b], \texttt{length} :: \forall a.[a] \to \texttt{Int}\}$

Typisiere `map length` xs

$$
\begin{tikzcd}
\texttt{map}: & (a \to b) \arrow[r] \arrow[d, equal] & {[a]} \arrow[r] \arrow[d, equal] & {[b]} \arrow[d, equal] \\
& ([c] \to \texttt{Int}) & {[[c]]} & {[\texttt{Int}]} \\
& \texttt{length} & xs & \text{(Resultat)}
\end{tikzcd}
$$

Ergibt: $a = [c]$, $b = \texttt{Int}$

$(\texttt{map length xs}) :: [\texttt{Int}]$

Bsp.: Typisierung von Lambda-geb. Variablen (1)

Beispiel:
Die Funktion `const` ist definiert als

```
const :: a -> b -> a
const x y = x
```

Typisierung von $\lambda x.\texttt{const}\ (x\ \texttt{True})\ (x\ \texttt{'A'})$

Zum Beispiel: nach Einsetzen von $x = \texttt{Id}$ wäre der Rumpf getypt.

Anfangsannahme:

$$A_0 = \{\texttt{const} :: \forall a, b.a \to b \to a, \texttt{True} :: \texttt{Bool}, \texttt{'A'} :: \texttt{Char}\}.$$

Bsp.: Typisierung von Lambda-geb. Variablen (2)

$$
{\scriptstyle(\textsc{RAbs})}\dfrac{{\scriptstyle(\textsc{RApp})}\dfrac{{\scriptstyle(\textsc{RApp})}\dfrac{{\scriptstyle(\textsc{AxK})}\dfrac{}{A_1 \vdash \texttt{const} :: \alpha_2 \to \alpha_3 \to \alpha_2, \emptyset} \;,\; {\scriptstyle(\textsc{RApp})}\dfrac{{\scriptstyle(\textsc{AxV})}\dfrac{}{A_1 \vdash x :: \alpha_1} \;,\; {\scriptstyle(\textsc{AxK})}\dfrac{}{A_1 \vdash \texttt{True} :: \texttt{Bool}}}{A_1 \vdash (x\ \texttt{True}) :: \alpha_4, E_1}}{A_1 \vdash \texttt{const}\ (x\ \texttt{True}) :: \alpha_5, E_2} \;,\; {\scriptstyle(\textsc{RApp})}\dfrac{{\scriptstyle(\textsc{AxV})}\dfrac{}{A_1 \vdash x :: \alpha_1} \;,\; {\scriptstyle(\textsc{AxK})}\dfrac{}{A_1 \vdash \texttt{'A'} :: \texttt{Char}}}{A_1 \vdash (x\ \texttt{'A'}) :: \alpha_6, E_3}}{A_1 \vdash \texttt{const}\ (x\ \texttt{True})\ (x\ \texttt{'A'}) :: \alpha_7, E_4}}{A_0 \vdash \lambda x.\texttt{const}\ (x\ \texttt{True})\ (x\ \texttt{'A'}) :: \alpha_1 \to \alpha_7, E_4}
$$

wobei $A_1 = A_0 \cup \{x :: \alpha_1\}$ und:

$$
\begin{array}{lcl}
E_1 & = & \{\alpha_1 \doteq \texttt{Bool} \to \alpha_4\} \\
E_2 & = & \{\alpha_1 \doteq \texttt{Bool} \to \alpha_4, \alpha_2 \to \alpha_3 \to \alpha_2 \doteq \alpha_4 \to \alpha_5\} \\
E_3 & = & \{\alpha_1 \doteq \texttt{Char} \to \alpha_6\} \\
E_4 & = & \{\alpha_1 \doteq \texttt{Bool} \to \alpha_4, \alpha_2 \to \alpha_3 \to \alpha_2 \doteq \alpha_4 \to \alpha_5, \alpha_1 \doteq \texttt{Char} \to \alpha_6, \\
 & & \alpha_5 \doteq \alpha_6 \to \alpha_7\}
\end{array}
$$

Bsp.: Typisierung von Lambda-geb. Variablen (2)

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

$$
\text{(RAbs)}\dfrac{\text{(RApp)}\dfrac{\text{(RApp)}\dfrac{\text{(AxK)}\dfrac{}{A_1 \vdash \mathtt{const} :: \alpha_2 \to \alpha_3 \to \alpha_2, \emptyset} \;,\; \text{(RApp)}\dfrac{\text{(AxV)}\dfrac{}{A_1 \vdash x :: \alpha_1} \;,\; \text{(AxK)}\dfrac{}{A_1 \vdash \mathtt{True} :: \mathtt{Bool}}}{A_1 \vdash (x\ \mathtt{True}) :: \alpha_4, E_1}}{A_1 \vdash \mathtt{const}\ (x\ \mathtt{True}) :: \alpha_5, E_2} \;,\; \text{(RApp)}\dfrac{\text{(AxV)}\dfrac{}{A_1 \vdash x :: \alpha_1} \;,\; \text{(AxK)}\dfrac{}{A_1 \vdash \mathtt{'A'} :: \mathtt{Char}}}{A_1 \vdash (x\ \mathtt{'A'}) :: \alpha_6, E_3}}{A_1 \vdash \mathtt{const}\ (x\ \mathtt{True})\ (x\ \mathtt{'A'}) :: \alpha_7, E_4}}{A_0 \vdash \lambda x.\mathtt{const}\ (x\ \mathtt{True})\ (x\ \mathtt{'A'}) :: \alpha_1 \to \alpha_7, E_4}
$$

wobei $A_1 = A_0 \cup \{x :: \alpha_1\}$ und:

$$
\begin{array}{lcl}
E_1 & = & \{\alpha_1 \doteq \mathtt{Bool} \to \alpha_4\} \\
E_2 & = & \{\alpha_1 \doteq \mathtt{Bool} \to \alpha_4, \alpha_2 \to \alpha_3 \to \alpha_2 \doteq \alpha_4 \to \alpha_5\} \\
E_3 & = & \{\alpha_1 \doteq \mathtt{Char} \to \alpha_6\} \\
E_4 & = & \{\alpha_1 \doteq \mathtt{Bool} \to \alpha_4, \alpha_2 \to \alpha_3 \to \alpha_2 \doteq \alpha_4 \to \alpha_5, \alpha_1 \doteq \mathtt{Char} \to \alpha_6, \\
 & & \;\alpha_5 \doteq \alpha_6 \to \alpha_7\}
\end{array}
$$

Die Unifikation schlägt fehl, da $\mathtt{Char} \neq \mathtt{Bool}$

Bsp.: Typisierung von Lambda-geb. Variablen (3)

In Haskell:

```
Main> \x -> const (x True)  (x 'A')

<interactive>:1:23:
Couldn't match expected type `Char' against inferred type `Bool'
      Expected type: Char -> b
      Inferred type: Bool -> a
 In the second argument of `const', namely `(x 'A')'
 In the expression: const (x True) (x 'A')
```

Bsp.: Typisierung von Lambda-geb. Variablen (3)

In Haskell:

```
Main> \x -> const (x True)  (x 'A')

<interactive>:1:23:
Couldn't match expected type `Char' against inferred type `Bool'
      Expected type: Char -> b
      Inferred type: Bool -> a
 In the second argument of `const', namely `(x 'A')'
 In the expression: const (x True) (x 'A')
```

- Beispiel verdeutlicht: Lambda-gebundene Variablen sind monomorph getypt!
- Das gleiche gilt für case-Pattern gebundene Variablen

Bsp.: Typisierung von Lambda-geb. Variablen (3)

In Haskell:

```
Main> \x -> const (x True)  (x 'A')

<interactive>:1:23:
Couldn't match expected type ‘Char’ against inferred type ‘Bool’
      Expected type: Char -> b
      Inferred type: Bool -> a
 In the second argument of ‘const’, namely ‘(x 'A')’
 In the expression: const (x True) (x 'A')
```

- Beispiel verdeutlicht: Lambda-gebundene Variablen sind monomorph getypt!
- Das gleiche gilt für case-Pattern gebundene Variablen
- Daher spricht man auch von `let`-Polymorphismus, da **nur `let`-gebundene Variablen (Funktionen) polymorph sind**.
- KFPTS+seq hat kein `let`, aber Superkombinatoren, die wie (ein eingeschränkten rekursives) `let` wirken