Typisierung

Prof. Dr. David Sabel

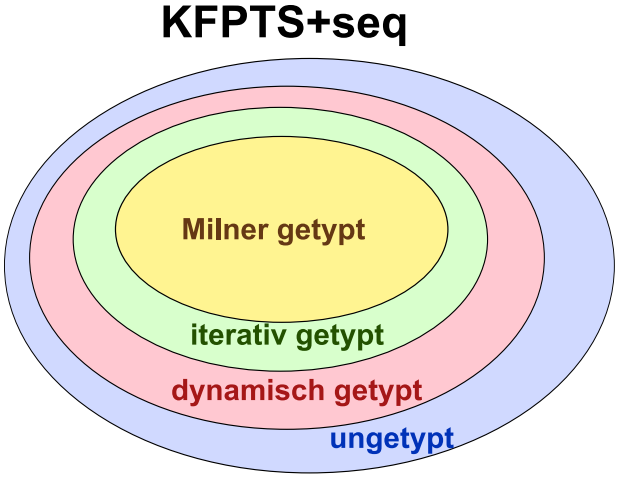
LFE Theoretische Informatik

Letzte Änderung der Folien: 15. Dezember 2021 ♦ Die Folien basieren zum Teil auf Material von Dr. Steffen Jost, dem an dieser Stelle für die Verwendungserlaubnis herzlich gedankt sei.

Ziele des Kapitels

- Warum typisieren?
- Typisierungsverfahren für Haskell bzw. KFPTS+seq für parametrisch polymorphe Typen
- Iteratives Typisierungsverfahren
- Milnersches Typisierungsverfahren

Übersicht



Motivation

Warum ist ein Typsystem sinnvoll?

- Für ungetypte Programme können dynamische Typfehler auftreten
- Fehler 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

Wünschenswerte Eigenschaften:

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

Motivation (3)

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.

Naiver Ansatz

Naive Definition von „korrekt getypt“:

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

Funktioniert nicht gut, denn

> Die dynamische Typisierung in KFPTS+seq ist unentscheidbar!

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)

Unentscheidbarkeit der dynamischen Typisierung (2)

Für eine TM-Beschreibung b und Eingabe e sei

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

Es gilt:

s ist genau dann dynamisch ungetypt, wenn 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 Typ der Form TC, wobei TC ein nullstelliger Typkonstruktor ist.
- 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 keine Basistypen aber Grundtypen.
- `[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 Reihenfolge egal, 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

Typsubstitution = 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

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

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

Bekannte Regel:

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

Problem: Man muss "richtige Instanz raten", z.B.

```
map ::  (a -> b)      -> [a] -> [b]
not :: Bool -> Bool
```

Typisierung von `map not`: Vor Anwendung der Regel muss der Typ von `map` instanziiert werden mit

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

Statt σ zu raten, kann man σ berechnen: Unifikation

Unifikationsproblem

Definition

- Ein Unifikationsproblem auf Typen ist gegeben durch eine Menge E von Gleichungen der Form $\tau_1 \doteq \tau_2$, wobei τ_1 und τ_2 polymorphe Typen sind.
- Eine Lösung eines Unifikationsproblem E 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 E ist ein Unifikator σ, so dass gilt: Für jeden anderen Unifikator ρ von E gibt es eine Substitution γ so dass $\rho(x) = \gamma \circ \sigma(x)$ für alle $x \in FV(E)$.

Unifikationsalgorithmus

- Datenstruktur: E = Multimenge von Gleichungen
  Multimenge ≡ Menge mit mehrfachem Vorkommen von Elementen
- $E \cup E'$ sei die disjunkte Vereinigung von zwei Multimengen
- $E[\tau/\alpha]$ ist definiert als $\{s[\tau/\alpha] \doteq t[\tau/\alpha] \mid (s \doteq t) \in E\}$.

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

- Fail auftritt, oder
- keine Regel mehr anwendbar ist

Unifikationsalgorithmus: Schlussregeln

Fail-Regeln:

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

wenn $TC_1 \neq TC_2$

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

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

Unifikationsalgorithmus: Schlussregeln (2)

Dekomposition:

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

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

Unifikationsalgorithmus: Schlussregeln (3)

Orientierung, Elimination:

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

wenn τ_1 keine Typvariable und α Typvariable

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

wobei α Typvariable

Unifikationsalgorithmus: Schlussregeln (4)

Einsetzung, Occurs-Check:

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

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

$$\textsc{OccursCheck}\ \frac{E \cup \{\alpha \doteq \tau\}}{\text{Fail}}$$

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

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

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

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

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

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

Beispiele (2)

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

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

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

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

Eigenschaften des Unifikationsalgorithmus

- Der Algorithmus endet mit Fail g.d.w. es keinen Unifikator für die Eingabe gibt.
- Der Algorithmus endet erfolgreich g.d.w. es einen Unifikator für die Eingabe gibt.
  Das Gleichungssystem E 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 ist dann $\sigma = \{\alpha_1 \mapsto \tau_1, \ldots, \alpha_n \mapsto \tau_n\}$.
- Liefert der Algorithmus einen Unifikator, dann ist es ein allgemeinster Unifikator.
- Man braucht keine alternativen Regelanwendungen auszuprobieren!
  Der Algorithmus kann deterministisch implementiert werden.
- Der Algorithmus terminiert für jedes Unifikationsproblem auf Typen.
  Ausgabe: Fail oder der allgemeinste Unifikator

Eigenschaften des Unifikationsalgorithmus (2)

- Die Typen in der Resultat-Substitution können exponentiell groß werden.

  Z.B. $\{\alpha_n \doteq \alpha_{n-1} \to \alpha_{n-1}, \alpha_{n-1} \doteq \alpha_{n-2} \to \alpha_{n-2}, \ldots \alpha_1 \doteq \alpha_0 \to \alpha_0\}$
  Der Unifikator bildet jedes α_i auf einen Typ ab, der $2^i - 1$ Typpfeile enthält. Z.B.
  $\sigma(\alpha_1) = \alpha_0 \to \alpha_0$,
  $\sigma(\alpha_2) = (\alpha_0 \to \alpha_0) \to (\alpha_0 \to \alpha_0)$,
  $\sigma(\alpha_3) = ((\alpha_0 \to \alpha_0) \to (\alpha_0 \to \alpha_0)) \to ((\alpha_0 \to \alpha_0) \to (\alpha_0 \to \alpha_0))$
- Der Unifikationsalgorithmus kann so implementiert werden, dass er Zeit $O(n \log n)$ benötigt. Man muss Sharing dazu beachten und 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. u.a.
  - Man kann im wesentlichen alle PTIME-Probleme als Unifikationsproblem darstellen.
  - Unifikation ist nicht effizient parallelisierbar.

Typisierungsverfahren

Wir betrachten nun die

polymorphe Typisierung von KFPTSP+seq-Ausdrücken

Wir verschieben zunächst: Typisierung von Superkombinatoren

Nächster Schritt:

Wie müssen die Typisierungsregeln aussehen?

Anwendungsregel mit Unifikation

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

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

Beispiel:

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

wenn σ allgemeinster Unifikator für
$(a \to b) \to [a] \to [b] \doteq (\texttt{Bool} \to \texttt{Bool}) \to \alpha$ ist
und α neue Typvariable ist.

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

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

Typisierung mit Bindern

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

- Typisiere den Rumpf s
- Sei $s :: \tau$
- Dann erhält $\lambda x.s$ einen Funktionstyp $\tau_1 \to \tau$
- Was hat τ_1 mit τ zu tun?
- τ_1 ist der Typ von x
- Wenn x im Rumpf s vorkommt, brauchen wir τ_1 bei der Berechnung von τ!

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:

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

Bedeutung:

Gegeben eine Menge Γ von Typ-Annahmen.
Dann kann für den Ausdruck s der Typ τ und die Typgleichungen E hergeleitet werden.

- In Γ kommen nur Typ-Annahmen für Konstruktoren, Variablen, Superkombinatoren vor.
- In E sammeln wir Gleichungen, sie werden erst später unifiziert.

Typisierung von Ausdrücken (2)

Herleitungsregeln schreiben wir in der Form

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

$$\frac{\Gamma_1 \vdash s_1 :: \tau_1, E_1 \quad \ldots \quad \Gamma_k \vdash s_k :: \tau_k, E_K}{\Gamma \vdash s :: \tau, E}$$

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:

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

Axiom für Konstruktoren:

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

wobei β_i neue Typvariablen sind

- Beachte: Jedesmal wird ein neu umbenannter Typ verwendet!

Typisierungsregeln für KFPTS+seq Ausdrücke (2)

Axiom für Superkombinatoren, deren Typ schon bekannt ist:

$$(\text{AxSK})\ \frac{}{\Gamma \cup \{SK :: \forall\alpha_1 \ldots \alpha_n.\tau\} \vdash SK :: \tau[\beta_1/\alpha_1, \ldots, \beta_n/\alpha_n], \emptyset}$$
wobei β_i neue Typvariablen sind

- Beachte: Jedesmal wird ein neu umbenannter Typ verwendet!

Typisierungsregeln für KFPTS+seq Ausdrücke (3)

Regel für Anwendungen:

$$(\text{RApp})\ \frac{\Gamma \vdash s :: \tau_1, E_1 \quad \text{und} \quad \Gamma \vdash t :: \tau_2, E_2}{\Gamma \vdash (s\ t) :: \alpha, E_1 \cup E_2 \cup \{\tau_1 \doteq \tau_2 \to \alpha\}}$$
wobei α neue Typvariable

Regel für `seq`:

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

Typisierungsregeln für KFPTS+seq Ausdrücke (4)

Regel für Abstraktionen:

$$(\text{RAbs})\ \frac{\Gamma \cup \{x :: \alpha\} \vdash s :: \tau, E}{\Gamma \vdash \lambda x.s :: \alpha \to \tau, E}$$
wobei α eine neue Typvariable

Typisierungsregeln für KFPTS+seq Ausdrücke (5)

Typisierung eines `case`: Prinzipien

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

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

Typisierungsregeln für KFPTS+seq Ausdrücke (6)

Regel für `case`**:**

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

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

Instanz der Case-Regel für Bool

$$\text{(RCase)}\frac{\Gamma \vdash s :: \tau, E \quad \Gamma \vdash \texttt{True} :: \tau_1, E_1 \quad \Gamma \vdash \texttt{False} :: \tau_2, E_2 \quad \Gamma \vdash t_1 :: \tau_1', E_1' \quad \Gamma \vdash t_2 :: \tau_2', E_2'}{\Gamma \vdash (\texttt{case}_{\texttt{Bool}}\ s\ \texttt{of}\ \{\texttt{True} \to t_1; \texttt{False} \to t_2\}) :: \alpha, E'}$$

wobei $E' = E \cup E_1 \cup E_2 \cup E_1' \cup E_2' \cup \{\tau \doteq \tau_1, \tau \doteq \tau_2\} \cup \{\alpha \doteq \tau_1', \alpha \doteq \tau_2'\}$

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

Instanz der Case-Regel für Listen

$$\text{(RCase)}\frac{\begin{array}{l}\Gamma \vdash s :: \tau, E \\ \Gamma \vdash \texttt{Nil} :: \tau_1, E_1 \\ \Gamma \cup \{x_1 :: \alpha_1, x_2 :: \alpha_2\} \vdash \texttt{Cons}\ x_1\ x_2 :: \tau_2, E_2 \\ \Gamma \vdash t_1 :: \tau_1', E_1' \\ \Gamma \cup \{x_1 :: \alpha_1, x_2 :: \alpha_2\} \vdash t_2 :: \tau_2', E_2'\end{array}}{\Gamma \vdash (\texttt{case}_{\texttt{List}}\ s\ \texttt{of}\ \{\texttt{Nil} \to t_1; (\texttt{Cons}\ x_1\ x_2) \to t_2\}) :: \alpha, E'}$$

wobei $E' = E \cup E_1 \cup E_2 \cup E_1' \cup E_2' \cup \{\tau \doteq \tau_1, \tau \doteq \tau_2\} \cup \{\alpha \doteq \tau_1', \alpha \doteq \tau_2'\}$

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

Typisierungsalgorithmus für KFPTS+seq-Ausdrücke

Sei s ein geschlossener KFPTS+seq-Ausdruck, wobei die Typen für alle in s benutzten Superkombinatoren und Konstruktoren bekannt sind.

1. Starte mit Anfangsannahme Γ, die Typen für die Konstruktoren und die Superkombinatoren enthält.
2. Leite $\Gamma \vdash s :: \tau, E$ 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:

Typberechnung:

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

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)

Beispiele: Typisierung von (Cons True Nil)

Typisierung von `Cons True Nil`

Starte mit:
Anfangsannahme: $\Gamma_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a], \texttt{True} :: \texttt{Bool}\}$

$$(\text{RApp})\ \frac{\Gamma_0 \vdash (\texttt{Cons True}) :: \tau_1, E_1, \quad \Gamma_0 \vdash \texttt{Nil} :: \tau_2, E_2}{\Gamma_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, E_1 \cup E_2 \cup \{\tau_1 \doteq \tau_2 \to \alpha_4\}}$$

$$(\text{RApp})\ \frac{\Gamma_0 \vdash (\texttt{Cons True}) :: \tau_1, E_1, \quad (\text{AxK})\ \overline{\Gamma_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{\Gamma_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, E_1 \cup \emptyset \cup \{\tau_1 \doteq [\alpha_3] \to \alpha_4\}}$$

$$(\text{RApp})\ \frac{(\text{RApp})\ \dfrac{\Gamma_0 \vdash \texttt{Cons} :: \tau_3, E_3, \Gamma_0 \vdash \texttt{True} :: \tau_4, E_4}{\Gamma_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\tau_3 \doteq \tau_4 \to \alpha_2\} \cup E_3 \cup E_4}, \quad (\text{AxK})\ \overline{\Gamma_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{\Gamma_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\tau_3 \doteq \tau_4 \to \alpha_2\} \cup E_3 \cup E_4 \cup \{\alpha_2 \doteq [\alpha_3] \to \alpha_4\}}$$

$$(\text{RApp})\ \frac{(\text{RApp})\ \dfrac{(\text{AxK})\ \overline{\Gamma_0 \vdash \texttt{Cons} :: \alpha_1 \to [\alpha_1] \to [\alpha_1], \emptyset}, \Gamma_0 \vdash \texttt{True} :: \tau_4, E_4}{\Gamma_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \tau_4 \to \alpha_2\} \cup E_4}, \quad (\text{AxK})\ \overline{\Gamma_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{\Gamma_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \tau_4 \to \alpha_2\} \cup E_4 \cup \{\alpha_2 \doteq [\alpha_3] \to \alpha_4\}}$$

$$(\text{RApp})\ \frac{(\text{RApp})\ \dfrac{(\text{AxK})\ \overline{\Gamma_0 \vdash \texttt{Cons} :: \alpha_1 \to [\alpha_1] \to [\alpha_1], \emptyset}, \quad (\text{AxK})\ \overline{\Gamma_0 \vdash \texttt{True} :: \texttt{Bool}, \emptyset}}{\Gamma_0 \vdash (\texttt{Cons True}) :: \alpha_2, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2\}}, \quad (\text{AxK})\ \overline{\Gamma_0 \vdash \texttt{Nil} :: [\alpha_3], \emptyset}}{\Gamma_0 \vdash (\texttt{Cons True Nil}) :: \alpha_4, \{\alpha_1 \to [\alpha_1] \to [\alpha_1] \doteq \texttt{Bool} \to \alpha_2\} \cup \{\alpha_2 \doteq [\alpha_3] \to \alpha_4\}}$$

Beispiele: Typisierung von $\lambda x.x$

Typisierung von $\lambda x.x$

Starte mit: Anfangsannahme: $\Gamma_0 = \emptyset$

$$(\text{RAbs})\ \frac{\Gamma_0 \cup \{x :: \alpha\} \vdash x :: \tau, E \overset{(\text{AxV})}{}}{\Gamma_0 \vdash (\lambda x.x) :: \alpha \to \tau, E} \quad (\text{RAbs})\ \frac{\overline{\Gamma_0 \cup \{x :: \alpha\} \vdash x :: \alpha, \emptyset}}{\Gamma_0 \vdash (\lambda x.x) :: \alpha \to \alpha, \emptyset}$$

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: $\Gamma_0 = \emptyset$

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

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

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

Beispiele: Typisierung eines Ausdrucks mit SKs (1)

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:

$$\begin{aligned}\Gamma_0 = \{&\texttt{map} :: \forall a, b.(a \to b) \to [a] \to [b],\\ &\texttt{length} :: \forall a.[a] \to \texttt{Int},\\ &\texttt{Nil} :: \forall a.[a]\\ &\texttt{Cons} :: \forall a.a \to [a] \to [a]\\ \}&\end{aligned}$$

Beispiele: Typisierung eines Ausdrucks mit SKs (2)

Herleitungsbaum:

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

Beschriftungen:

$$\begin{aligned}
B_1 = \quad & \Gamma_0 \vdash t :: \alpha_1 \to \alpha_{13},\\
& \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7,\\
& (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \texttt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12},\\
& \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}, \}\\
B_2 = \quad & \Gamma_0 \cup \{xs :: \alpha_1\} \vdash\\
& \quad \texttt{case}_{\texttt{List}}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \texttt{Nil}; (\texttt{Cons}\ y\ ys) \to \texttt{map}\ \texttt{length}\ ys\} :: \alpha_{13},\\
& \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7,\\
& (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \texttt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12},\\
& \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}, \}
\end{aligned}$$

Beispiele: Typisierung eines Ausdrucks mit SKs (3)

Herleitungsbaum:

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

Beschriftungen:

$$\begin{aligned}
B_3 = \quad & \Gamma_0 \cup \{xs :: \alpha_1\} \vdash\ xs :: \alpha_1, \emptyset\\
B_4 = \quad & \Gamma_0 \cup \{xs :: \alpha_1\} \vdash \texttt{Nil} :: [\alpha_2], \emptyset\\
B_5 = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash (\texttt{Cons}\ y\ ys) :: \alpha_7,\\
& \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7\}\\
B_6 = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash (\texttt{Cons}\ y) :: \alpha_6,\\
& \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6\}\\
B_7 = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash ys :: \alpha_4, \emptyset\\
B_8 = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash \texttt{Cons} :: \alpha_5 \to [\alpha_5] \to [\alpha_5], \emptyset\\
B_9 = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash y :: \alpha_3, \emptyset\\
B_{10} = \quad & \Gamma_0 \cup \{xs :: \alpha_1\} \vdash \texttt{Nil} :: [\alpha_{14}], \emptyset
\end{aligned}$$

Beispiele: Typisierung eines Ausdrucks mit SKs (4)

Herleitungsbaum:

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

Beschriftungen:

$$\begin{aligned}
B_{11} = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash (\texttt{map length})\ ys :: \alpha_{12},\\
& \{(\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \texttt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12}\}\\
B_{12} = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash (\texttt{map length}) :: \alpha_{11},\\
& \{(\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \texttt{Int}) \to \alpha_{11}\}\\
B_{13} = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash ys :: \alpha_4, \emptyset\\
B_{14} = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash \texttt{map} :: (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9], \emptyset\\
B_{15} = \quad & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_3, ys :: \alpha_4\} \vdash \texttt{length} :: [\alpha_{10}] \to \texttt{Int}, \emptyset
\end{aligned}$$

Beispiele: Typisierung eines Ausdrucks mit SKs (5)

Beschriftung unten:

$$\begin{aligned}
B_1 = \quad & \Gamma_0 \vdash t :: \alpha_1 \to \alpha_{13},\\
& \{\alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_3 \to \alpha_6, \alpha_6 \doteq \alpha_4 \to \alpha_7,\\
& (\alpha_8 \to \alpha_9) \to [\alpha_8] \to [\alpha_9] \doteq ([\alpha_{10}] \to \texttt{Int}) \to \alpha_{11}, \alpha_{11} \doteq \alpha_4 \to \alpha_{12},\\
& \alpha_1 \doteq [\alpha_2], \alpha_1 = \alpha_7, \alpha_{13} \doteq [\alpha_{14}], \alpha_{13} = \alpha_{12}, \}
\end{aligned}$$

Löse mit Unifikation:

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

Ergibt:

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

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

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

Die Funktion `const` ist definiert als

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

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

Anfangsannahme:

$\Gamma_0 = \{\texttt{const} :: \forall a, b.a \to b \to a, \texttt{True} :: \texttt{Bool}, \texttt{'A'} :: \texttt{Char}\}$.

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

$$\dfrac{\dfrac{\dfrac{\overset{(\text{AxSK})}{\overline{\Gamma_1 \vdash \texttt{const} :: \alpha_2 \to \alpha_3 \to \alpha_2, \emptyset}}, \overset{(\text{RApp})}{\dfrac{\overset{(\text{AxV})}{\overline{\Gamma_1 \vdash x :: \alpha_1}}, \overset{(\text{AxK})}{\overline{\Gamma_1 \vdash \texttt{True} :: \texttt{Bool}}}}{\Gamma_1 \vdash (x\ \texttt{True}) :: \alpha_4, E_1}}}{\Gamma_1 \vdash \texttt{const}\ (x\ \texttt{True}) :: \alpha_5, E_2}, \overset{(\text{RApp})}{\dfrac{\overset{(\text{AxV})}{\overline{\Gamma_1 \vdash x :: \alpha_1}}, \overset{(\text{AxK})}{\overline{\Gamma_1 \vdash \texttt{'A'} :: \texttt{Char}}}}{\Gamma_1 \vdash (x\ \texttt{'A'}) :: \alpha_6, E_3}}}{\Gamma_1 \vdash \texttt{const}\ (x\ \texttt{True})\ (x\ \texttt{'A'}) :: \alpha_7, E_4}\ (\text{RApp})}{\Gamma_0 \vdash \lambda x.\texttt{const}\ (x\ \texttt{True})\ (x\ \texttt{'A'}) :: \alpha_1 \to \alpha_7, E_4}\ (\text{RAbs})$$

wobei $\Gamma_1 = \Gamma_0 \cup \{x :: \alpha_1\}$ und:

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

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

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')
```

- 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 polymorph sind.
- KFPTS+seq hat kein `let`, aber Superkombinatoren, die wie (ein eingeschränktes rekursives) `let` wirken

Rekursive Superkombinatoren

Definition (direkt rekursiv, rekursiv, verschränkt rekursiv)

- Sei $\mathcal{SK}$ eine Menge von Superkombinatoren
- Für $SK_i, SK_j \in \mathcal{SK}$ sei
$$SK_i \preceq SK_j$$
g.d.w. SK_j den Superkombinator SK_i im Rumpf benutzt.
- $\preceq^+$: transitiver Abschluss von $\preceq$ ($\preceq^*$: reflexiv-transitiver Abschluss)
- SK_i ist direkt rekursiv wenn $SK_i \preceq SK_i$ gilt.
- SK_i ist rekursiv wenn $SK_i \preceq^+ SK_i$ gilt.
- $SK_1, \ldots, SK_m$ sind verschränkt rekursiv, wenn $SK_i \preceq^+ SK_j$ für alle $i, j \in \{1, \ldots, m\}$

Typisierung von nicht-rekursiven Superkombinatoren

- Nicht-rekursive Superkombinatoren kann man wie Abstraktionen typisieren
- Notation: $\Gamma \vdash_T SK :: \tau$, bedeutet: unter Annahme Γ kann man SK mit Typ τ typisieren

Typisierungsregel für (geschlossene) nicht-rekursive SK:

$$(\text{RSK1})\ \frac{\Gamma \cup \{x_1 :: \alpha_1, \ldots, x_n :: \alpha_n\} \vdash s :: \tau, E}{\Gamma \vdash_T SK :: \forall \mathcal{X}.\sigma(\alpha_1 \to \ldots \to \alpha_n \to \tau)}$$

wenn σ Lösung von E,

$SK\ x_1\ \ldots\ x_n = s$ die Definition von SK

und SK nicht rekursiv ist,

und $\mathcal{X}$ die Typvariablen in $\sigma(\alpha_1 \to \ldots \to \alpha_n \to \tau)$

Beispiel: Typisierung von `(.)`

```
(.) f g x = f (g x)
```

Γ_0 ist leer, da keine Konstruktoren oder SK vorkommen.

$$\dfrac{\dfrac{\dfrac{}{\Gamma_1 \vdash f :: \alpha_1, \emptyset}(\text{AxV}) \quad \dfrac{\dfrac{}{\Gamma_1 \vdash g :: \alpha_2, \emptyset}(\text{AxV}) \quad \dfrac{}{\Gamma_1 \vdash x :: \alpha_3, \emptyset}(\text{AxV})}{\Gamma_1 \vdash (g\ x) :: \alpha_5, \{\alpha_2 \doteq \alpha_3 \to \alpha_5\}}(\text{RApp})}{\Gamma_1 \vdash (f\ (g\ x)) :: \alpha_4, \{\alpha_2 \doteq \alpha_3 \to \alpha_5, \alpha_1 = \alpha_5 \to \alpha_4\}}(\text{RApp})}{\emptyset \vdash_T (.) :: \forall \mathcal{X}.\sigma(\alpha_1 \to \alpha_2 \to \alpha_3 \to \alpha_4)}(\text{RSK1})$$

wobei $\Gamma_1 = \{f :: \alpha_1, g :: \alpha_2, x :: \alpha_3\}$

Unifikation ergibt $\sigma = \{\alpha_2 \mapsto \alpha_3 \to \alpha_5, \alpha_1 \mapsto \alpha_5 \to \alpha_4\}$.
Daher: $\sigma(\alpha_1 \to \alpha_2 \to \alpha_3 \to \alpha_4) = (\alpha_5 \to \alpha_4) \to (\alpha_3 \to \alpha_5) \to \alpha_3 \to \alpha_4$

Jetzt kann man $\mathcal{X} = \{\alpha_3, \alpha_4, \alpha_5\}$ berechnen , und umbenennen:

$$(.) :: \forall a, b, c.(a \to b) \to (c \to a) \to c \to b$$

Typisierung von rekursiven Superkombinatoren

- Sei $SK\ x_1\ \ldots\ x_n = e$
- und SK kommt in e vor, d.h. SK ist rekursiv
- Warum kann man SK nicht ganz einfach typisieren?
- Will man den Rumpf e typisieren, so muss man den Typ von SK kennen!

Idee des Iterativen Typisierungsverfahrens

- Gebe SK zunächst den allgemeinsten Typ
  (d.h. eine Typvariable) und typisiere den Rumpf unter Benutzung dieses Typs
- Man erhält anschließend einen neuen Typ für SK
- Mache mit neuem Typ weiter
- Stoppe, wenn neuer Typ $=$ alter Typ
- Dann hat man eine konsistente Typannahme gefunden;
  Vermutung: auch eine ausreichend allgemeine (allgemeinste?)

Allgemeinster Typ: Typ T so dass $\mathrm{sem}(T) = \{\text{alle Grundtypen}\}$.
Das liefert der Typ α (bzw. quantifiziert $\forall\alpha.\alpha$)

Iteratives Typisierungsverfahren

Regel zur Berechnung neuer Annahmen:

$$(\mathrm{SKR_{EK}})\ \frac{\Gamma \cup \{x_1 :: \alpha_1, \ldots, x_n :: \alpha_n\} \vdash s :: \tau, E}{\Gamma \vdash_T SK :: \sigma(\alpha_1 \to \ldots \alpha_n \to \tau)}$$

wenn $SK\ x_1\ \ldots\ x_n = s$ die Definition von SK, σ Lösung von E

Genau wie RSK1, aber in Γ muss es eine Annahme für SK geben.

Iteratives Typisierungsverfahren: Vorarbeiten (1)

Wegen verschränkter Rekursion:

- Abhängigkeitsanalyse der Superkombinatoren
- Berechnung der starken Zusammenhangskomponenten im Aufrufgraph
- Sei $\simeq$ die Äquivalenzrelation passend zu $\preceq^*$, dann sind die starken Zusammenhangskomponenten gerade die Äquivalenzklassen zu $\simeq$.
- Jede Äquivalenzklasse wird gemeinsam typisiert

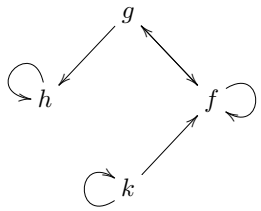
Typisierung der Gruppen entsprechend der $\preceq^*$-Ordnung modulo $\simeq$.

Iteratives Typisierungsverfahren: Vorarbeiten (2)

Beispiel:

```
f x y = if x<=1 then y else f (x-y) (y + g x)
g x   = if x==0 then (f 1 x) + (h 2) else 10
h x   = if x==1 then 0 else h (x-1)
k x y = if x==1 then y else k (x-1) (y+(f x y))
```

Der Aufrufgraph (nur bzgl. `f,g,h,k`) ist



Die Äquivalenzklassen (mit Ordnung) sind $\{h\} \preceq^+ \{f, g\} \preceq^+ \{k\}$.

Iteratives Typisierungsverfahren: Der Algorithmus

Iterativer Typisierungsalgorithmus

Eingabe: Verschränkt rekursive Superkombinatoren $SK_1, \ldots, SK_m$ (kleinere SKs schon typisiert)

1. Anfangsannahme Γ enthält Typen der Konstruktoren und der bereits bekannten SKs
2. $\Gamma_0 := \Gamma \cup \{SK_1 :: \forall\alpha_1.\alpha_1, \ldots, SK_m :: \forall\alpha_m.\alpha_m\}$ und $j = 0$.
3. Verwende für jeden Superkombinator SK_i (mit $i = 1, \ldots, m$) die Regel (SKR_{EK}) und Annahme Γ_j, um SK_i zu typisieren.
4. Wenn die m Typisierungen erfolgreich, d.h. für alle i: $\Gamma_j \vdash_T SK_i :: \tau_i$
   Dann allquantifiziere: $SK_1 :: \forall\mathcal{X}_1.\tau_1, \ldots, SK_m :: \forall\mathcal{X}_m.\tau_m$
   Setze $\Gamma_{j+1} := \Gamma \cup \{SK_1 :: \forall\mathcal{X}_1.\tau_1, \ldots, SK_m :: \forall\mathcal{X}_m.\tau_m\}$
5. Wenn $\Gamma_j \neq \Gamma_{j+1}$, dann gehe mit $j := j + 1$ zu Schritt (3).
   Anderenfalls, d.h. wenn $\Gamma_j = \Gamma_{j+1}$, war Γ_j **konsistent**.

Ausgabe: Allquantifizierte polymorphen Typen der SK_i aus der konsistenten Annahme.
Sollte irgendwann ein `Fail` in der Unifikation auftreten, dann sind $SK_1, \ldots, SK_m$ nicht typisierbar.

Eigenschaften des Algorithmus

- Die berechneten Typen pro Iterationsschritt sind eindeutig bis auf Umbenennung.
  $\Longrightarrow$ bei Terminierung liefert der Algorithmus eindeutige Typen.
- Pro Iteration werden die neuen Typen spezieller (oder bleiben gleich).
  D.h. Monotonie bzgl. der Grundtypensemantik: $\text{sem}(T_j) \supseteq \text{sem}(T_{j+1})$
- Bei Nichtterminierung gibt es keinen polymorphen Typ.
  Grund: Monotonie und man hat mit größten Annahmen begonnen.
- Das iterative Verfahren berechnet einen größten Fixpunkt (bzgl. der Grundtypensemantik): Menge wird solange verkleinert, bis sie sich nicht mehr ändert.
  D.h. es wird der *allgemeinste* polymorphe Typ berechnet

Beispiele: length (1)

$$\texttt{length}\ xs = \texttt{case}_{\texttt{List}}\ xs\ \texttt{of}\{\texttt{Nil} \to 0; (y : ys) \to 1 + \texttt{length}\ ys\}$$

Annahme:

$\Gamma = \{\texttt{Nil} :: \forall a.[a], (:) :: \forall a.a \to [a] \to [a], 0, 1 :: \texttt{Int}, (+) :: \texttt{Int} \to \texttt{Int} \to \texttt{Int}\}$

1.Iteration: $\Gamma_0 = \Gamma \cup \{\texttt{length} :: \forall\alpha.\alpha\}$

$$\text{(SKR}_{\text{EK}}\text{)}\ \frac{\text{(RC}_{\text{ASE}}\text{)}\ \dfrac{\begin{array}{ll}(a) & \Gamma_0 \cup \{xs :: \alpha_1\} \vdash xs :: \tau_1, E_1 \\ (b) & \Gamma_0 \cup \{xs :: \alpha_1\} \vdash \texttt{Nil} :: \tau_2, E_2 \\ (c) & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_4, ys :: \alpha_5\} \vdash (y : ys) :: \tau_3, E_3 \\ (d) & \Gamma_0 \cup \{xs :: \alpha_1\} \vdash 0 :: \tau_4, E_4 \\ (e) & \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_4, ys :: \alpha_5\}\} \vdash (1 + \texttt{length}\ ys) :: \tau_5, E_5\end{array}}{\Gamma_0 \cup \{xs :: \alpha_1\} \vdash (\texttt{case}_{\texttt{List}}\ xs\ \texttt{of}\{\texttt{Nil} \to 0; (y : ys) \to 1 + \texttt{length}\ xs\}) :: \alpha_3,\ E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup \{\tau_1 \doteq \tau_2, \tau_1 \doteq \tau_3, \alpha_3 \doteq \tau_4, \alpha_3 \doteq \tau_5\}}}{\Gamma_0 \vdash_T \texttt{length} :: \sigma(\alpha_1 \to \alpha_3)}$$

wobei σ Lösung von $E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup \{\tau_1 \doteq \tau_2, \tau_1 \doteq \tau_3, \alpha_3 \doteq \tau_4, \alpha_3 \doteq \tau_5\}$

Beispiele: length (2)

(a): $\dfrac{}{\Gamma_0 \cup \{xs :: \alpha_1\} \vdash xs :: \alpha_1, \emptyset}\ (\text{AxV})$
D.h $\tau_1 = \alpha_1$ und $E_1 = \emptyset$

(b): $\dfrac{}{\Gamma_0 \cup \{xs :: \alpha_1\} \vdash \texttt{Nil} :: [\alpha_6], \emptyset}\ (\text{AxK})$
D.h. $\tau_2 = [\alpha_6]$ und $E_2 = \emptyset$

(c)
$$\dfrac{\dfrac{\dfrac{}{\Gamma_0' \vdash (:) :: \alpha_9 \to [\alpha_9] \to [\alpha_9], \emptyset}\ (\text{AxK}) \quad \dfrac{}{\Gamma_0' \vdash y :: \alpha_4, \emptyset}\ (\text{AxV})}{\Gamma_0' \vdash ((:)\ y) :: \alpha_8, \{\alpha_9 \to [\alpha_9] \to [\alpha_9] \doteq \alpha_4 \to \alpha_8\}}\ (\text{RApp}) \quad \dfrac{}{\Gamma_0' \vdash ys :: \alpha_5, \emptyset}\ (\text{AxV})}{\Gamma_0' \vdash (y : ys) :: \alpha_7, \{\alpha_9 \to [\alpha_9] \to [\alpha_9] \doteq \alpha_4 \to \alpha_8, \alpha_8 \doteq \alpha_5 \to \alpha_7\}}\ (\text{RApp})$$
wobei $\Gamma_0 = \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_4, ys :: \alpha_5\}$
D.h. $\tau_3 = \alpha_7$ und $E_3 = \{\alpha_9 \to [\alpha_9] \to [\alpha_9] \doteq \alpha_4 \to \alpha_8, \alpha_8 \doteq \alpha_5 \to \alpha_7\}$

Beispiele: length (3)

(d) $\dfrac{}{\Gamma_0 \cup \{xs :: \alpha_1\} \vdash 0 :: \texttt{Int}, \emptyset}\ (\text{AxK})$
D.h. $\tau_4 = \texttt{Int}$ und $E_4 = \emptyset$

(e)
$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_0' \vdash (+) :: \texttt{Int} \to \texttt{Int} \to \texttt{Int}, \emptyset}\ (\text{AxK}) \quad \dfrac{}{\Gamma_0' \vdash 1 :: \texttt{Int}, \emptyset}\ (\text{AxK})}{\Gamma_0' \vdash ((+)\ 1) :: \alpha_{11}, \{\texttt{Int} \to \texttt{Int} \to \texttt{Int} \doteq \texttt{Int} \to \alpha_{11}\}}\ (\text{RApp}) \quad \dfrac{\dfrac{}{\Gamma_0' \vdash \texttt{length} :: \alpha_{13}, \emptyset}\ (\text{AxSK}) \quad \dfrac{}{\Gamma_0' \vdash (ys) :: \alpha_5, \emptyset}\ (\text{AxV})}{\Gamma_0' \vdash (\texttt{length}\ ys) :: \alpha_{12}, \{\alpha_{13} \doteq \alpha_5 \to \alpha_{12}\}}\ (\text{RApp})}{\Gamma_0' \vdash (1 + \texttt{length}\ ys) :: \alpha_{10}, \{\texttt{Int} \to \texttt{Int} \to \texttt{Int} \doteq \texttt{Int} \to \alpha_{11}, \alpha_{13} \doteq \alpha_5 \to \alpha_{12}, \alpha_{11} \doteq \alpha_{12} \to \alpha_{10}\}}\ (\text{RApp})$$
wobei $\Gamma_0 = \Gamma_0 \cup \{xs :: \alpha_1, y :: \alpha_4, ys :: \alpha_5\}$
D.h. $\tau_5 = \alpha_{10}$ und
$E_5 = \{\texttt{Int} \to \texttt{Int} \to \texttt{Int} \doteq \texttt{Int} \to \alpha_{11}, \alpha_{13} \doteq \alpha_5 \to \alpha_{12}, \alpha_{11} \doteq \alpha_{12} \to \alpha_{10}\}$

Beispiele: length (4)

Zusammengefasst: $\Gamma_0 \vdash_T \texttt{length} :: \sigma(\alpha_1 \to \alpha_3)$
wobei σ Lösung von

$$\{\alpha_9 \to [\alpha_9] \to [\alpha_9] \doteq \alpha_4 \to \alpha_8, \alpha_8 \doteq \alpha_5 \to \alpha_7,$$
$$\texttt{Int} \to \texttt{Int} \to \texttt{Int} \doteq \texttt{Int} \to \alpha_{11}, \alpha_{13} \doteq \alpha_5 \to \alpha_{12}, \alpha_{11} \doteq \alpha_{12} \to \alpha_{10},$$
$$\alpha_1 \doteq [\alpha_6], \alpha_1 \doteq \alpha_7, \alpha_3 \doteq \texttt{Int}, \alpha_3 \doteq \alpha_{10}\}$$

Die Unifikation ergibt als Unifikator

$$\{\alpha_1 \mapsto [\alpha_9], \alpha_3 \mapsto \texttt{Int}, \alpha_4 \mapsto \alpha_9, \alpha_5 \mapsto [\alpha_9], \alpha_6 \mapsto \alpha_9, \alpha_7 \mapsto [\alpha_9], \alpha_8 \mapsto [\alpha_9] \to [\alpha_9],$$
$$\alpha_{10} \mapsto \texttt{Int}, \alpha_{11} \mapsto \texttt{Int} \to \texttt{Int}, \alpha_{12} \mapsto \texttt{Int}, \alpha_{13} \mapsto [\alpha_9] \to \texttt{Int}\}$$

daher $\sigma(\alpha_1 \to \alpha_3) = [\alpha_9] \to \texttt{Int}$

$\Gamma_1 = \Gamma \cup \{\texttt{length} :: \forall\alpha.[\alpha] \to \texttt{Int}\}$

Da $\Gamma_0 \neq \Gamma_1$ muss man mit Γ_1 erneut iterieren.
2.Iteration: Ergibt den gleichen Typ, daher war Γ_1 konsistent.

Iteratives Verfahren ist allgemeiner als Haskell

Beispiel

```
g x = 1 : (g (g 'c'))
```

$\Gamma = \{1 :: \texttt{Int}, \texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{'c'} :: \texttt{Char}\}$
$\Gamma_0 = \Gamma \cup \{\texttt{g} :: \forall\alpha.\alpha\}$ (und $\Gamma_0' = \Gamma_0 \cup \{x :: \alpha_1\}$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_0' \vdash \texttt{Cons} :: \alpha_5 \to [\alpha_5] \to [\alpha_5], \emptyset}\ (\text{AxK}) \quad \dfrac{}{\Gamma_0' \vdash 1 :: \texttt{Int}, \emptyset}\ (\text{AxK})}{\Gamma_0' \vdash (\texttt{Cons}\ 1) :: \alpha_3, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \texttt{Int} \to \alpha_3}\ (\text{RApp}) \quad \dfrac{\dfrac{}{\Gamma_0' \vdash \texttt{g} :: \alpha_6, \emptyset}\ (\text{AxSK}) \quad \dfrac{\dfrac{}{\Gamma_0' \vdash \texttt{g} :: \alpha_8, \emptyset}\ (\text{AxSK}) \quad \dfrac{}{\Gamma_0' \vdash \texttt{'c'} :: \texttt{Char}, \emptyset}\ (\text{AxK})}{\Gamma_0' \vdash (\texttt{g}\ \texttt{'c'}) :: \alpha_7, \{\alpha_8 \doteq \texttt{Char} \to \alpha_7\}}\ (\text{RApp})}{\Gamma_0' \vdash (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_4, \{\alpha_8 \doteq \texttt{Char} \to \alpha_7, \alpha_6 \doteq \alpha_7 \to \alpha_4\}}\ (\text{RApp})}{\Gamma_0' \vdash \texttt{Cons}\ 1\ (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_2, \{\alpha_8 \doteq \texttt{Char} \to \alpha_7, \alpha_6 \doteq \alpha_7 \to \alpha_4, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \texttt{Int} \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}}\ (\text{RApp})}{\Gamma_0 \vdash_T \texttt{g} :: \sigma(\alpha_1 \to \alpha_2) = \alpha_1 \to [\texttt{Int}]}\ (\text{SKRek})$$

wobei $\sigma = \{\alpha_2 \mapsto [\texttt{Int}], \alpha_3 \mapsto [\texttt{Int}] \to [\texttt{Int}], \alpha_4 \mapsto [\texttt{Int}], \alpha_5 \mapsto \texttt{Int}, \alpha_6 \mapsto \alpha_7 \to [\texttt{Int}], \alpha_8 \mapsto \texttt{Char} \to \alpha_7\}$ die Lösung von $\{\alpha_8 \doteq \texttt{Char} \to \alpha_7, \alpha_6 \doteq \alpha_7 \to \alpha_4, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \texttt{Int} \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}$ ist.

D.h. $\Gamma_1 = \Gamma \cup \{\texttt{g} :: \forall\alpha.\alpha \to [\texttt{Int}]\}$.

Nächste Iteration zeigt: Γ_1 ist konsistent.

Iteratives Verfahren ist allgemeiner als Haskell (2)

Beachte: Für die Funktion g kann Haskell keinen Typ herleiten:

```
Prelude> let g x = 1:(g(g 'c'))

<interactive>:1:13:
Couldn't match expected type `[t]' against inferred type `Char'
      Expected type: Char -> [t]
      Inferred type: Char -> Char
    In the second argument of `(:)', namely `(g (g 'c'))'
    In the expression: 1 : (g (g 'c'))
```

Aber: Haskell kann den Typ verifizieren, wenn man ihn angibt:

```
let g::a -> [Int]; g x = 1:(g(g 'c'))
Prelude> :t g
g :: a -> [Int]
```

Grund: Wenn Typ vorhanden, führt Haskell keine Typinferenz durch, sondern verifiziert nur die Annahme. g wird im Rumpf wie bereits typisiert behandelt.

Bsp.: Mehrere Iterationen sind nötig (1)

```
g x = x : (g (g 'c'))
```

- $\Gamma = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{'c'} :: \texttt{Char}\}$.
- $\Gamma_0 = \Gamma \cup \{\texttt{g} :: \forall \alpha.\alpha\}$

$$\dfrac{\dfrac{\dfrac{\dfrac{(\text{AxK})\,\overline{\Gamma_0' \vdash \texttt{Cons} :: \alpha_5 \to [\alpha_5] \to [\alpha_5], \emptyset} \;,\; (\text{AxV})\,\overline{\Gamma_0' \vdash x :: \alpha_1, \emptyset}}{(\text{RApp})\,\Gamma_0' \vdash (\texttt{Cons}\ x) :: \alpha_3, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_1 \to \alpha_3} \;,\; \dfrac{(\text{AxSK})\,\overline{\Gamma_0' \vdash \texttt{g} :: \alpha_6, \emptyset} \;,\; \dfrac{(\text{AxSK})\,\overline{\Gamma_0' \vdash \texttt{g} :: \alpha_8, \emptyset} \;,\; (\text{AxK})\,\overline{\Gamma_0' \vdash \texttt{'c'} :: \texttt{Char}, \emptyset}}{(\text{RApp})\,\Gamma_0' \vdash (\texttt{g}\ \texttt{'c'}) :: \alpha_7, \{\alpha_8 \doteq \texttt{Char} \to \alpha_7\}}}{(\text{RApp})\,\Gamma_0' \vdash (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_4, \{\alpha_8 \doteq \texttt{Char} \to \alpha_7, \alpha_6 \doteq \alpha_7 \to \alpha_4\}}}{(\text{RApp})\,\Gamma_0' \vdash \texttt{Cons}\ x\ (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_2, \{\alpha_8 \doteq \texttt{Char} \to \alpha_7, \alpha_6 \doteq \alpha_7 \to \alpha_4, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_1 \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}}{(\text{SKRek})\,\Gamma_0 \vdash_T \texttt{g} :: \sigma(\alpha_1 \to \alpha_2) = \alpha_5 \to [\alpha_5]}$$

wobei $\sigma = \{\alpha_1 \mapsto \alpha_5, \alpha_2 \mapsto [\alpha_5], \alpha_3 \mapsto [\alpha_5] \to [\alpha_5], \alpha_4 \mapsto [\alpha_5], \alpha_6 \mapsto \alpha_7 \to [\alpha_5], \alpha_8 \mapsto \texttt{Char} \to \alpha_7\}$ die Lösung von $\{\alpha_8 \doteq \texttt{Char} \to \alpha_7, \alpha_6 \doteq \alpha_7 \to \alpha_4, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_1 \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}$ ist.

D.h. $\Gamma_1 = \Gamma \cup \{\texttt{g} :: \forall \alpha.\alpha \to [\alpha]\}$.

Bsp.: Mehrere Iterationen sind nötig (2)

Da $\Gamma_0 \neq \Gamma_1$ muss eine weitere Iteration durchgeführt werden.
Sei $\Gamma_1' = \Gamma_1 \cup \{x :: \alpha_1\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{(\text{AxK})\,\overline{\Gamma_1' \vdash \texttt{Cons} :: \alpha_5 \to [\alpha_5] \to [\alpha_5], \emptyset} \;,\; (\text{AxV})\,\overline{\Gamma_1' \vdash x :: \alpha_1, \emptyset}}{(\text{RApp})\,\Gamma_1' \vdash (\texttt{Cons}\ x) :: \alpha_3, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_1 \to \alpha_3} \;,\; \dfrac{(\text{AxSK})\,\overline{\Gamma_1' \vdash \texttt{g} :: \alpha_6 \to [\alpha_6], \emptyset} \;,\; \dfrac{(\text{AxSK})\,\overline{\Gamma_1' \vdash \texttt{g} :: \alpha_8 \to [\alpha_8], \emptyset} \;,\; (\text{AxK})\,\overline{\Gamma_1' \vdash \texttt{'c'} :: \texttt{Char}, \emptyset}}{(\text{RApp})\,\Gamma_1' \vdash (\texttt{g}\ \texttt{'c'}) :: \alpha_7, \{\alpha_8 \to [\alpha_8] \doteq \texttt{Char} \to \alpha_7\}}}{(\text{RApp})\,\Gamma_1' \vdash (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_4, \{\alpha_8 \to [\alpha_8] \doteq \texttt{Char} \to \alpha_7, \alpha_6 \to [\alpha_6] \doteq \alpha_7 \to \alpha_4\}}}{(\text{RApp})\,\Gamma_1' \vdash \texttt{Cons}\ x\ (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_2, \{\alpha_8 \to [\alpha_8] \doteq \texttt{Char} \to \alpha_7, \alpha_6 \to [\alpha_6] \doteq \alpha_7 \to \alpha_4 \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_1 \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}}{(\text{SKRek})\,\Gamma_1 \vdash_T \texttt{g} :: \sigma(\alpha_1 \to \alpha_2) = [\texttt{Char}] \to [[\texttt{Char}]]}$$

wobei $\sigma = \{\alpha_1 \mapsto [\texttt{Char}], \alpha_2 \mapsto [[\texttt{Char}]], \alpha_3 \mapsto [[\texttt{Char}]] \to [[\texttt{Char}]], \alpha_4 \mapsto [[\texttt{Char}]], \alpha_5 \mapsto [\texttt{Char}], \alpha_6 \mapsto [\texttt{Char}], \alpha_7 \mapsto [\texttt{Char}], \alpha_8 \mapsto \texttt{Char}\}$
die Lösung von $\{\alpha_8 \to [\alpha_8] \doteq \texttt{Char} \to \alpha_7, \alpha_6 \to [\alpha_6] \doteq \alpha_7 \to \alpha_4, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_1 \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}$ ist.

Daher ist $\Gamma_2 = \Gamma \cup \{\texttt{g} :: [\texttt{Char}] \to [[\texttt{Char}]]\}$.

Bsp.: Mehrere Iterationen sind nötig (3)

Da $\Gamma_1 \neq \Gamma_2$ muss eine weitere Iteration durchgeführt werden:
Sei $\Gamma_2' = \Gamma_2 \cup \{x :: \alpha_1\}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{(\text{AxK})\,\overline{\Gamma_2' \vdash \texttt{Cons} :: \alpha_5 \to [\alpha_5] \to [\alpha_5], \emptyset} \;,\; (\text{AxV})\,\overline{\Gamma_2' \vdash x :: \alpha_1, \emptyset}}{(\text{RApp})\,\Gamma_2' \vdash (\texttt{Cons}\ x) :: \alpha_3, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_1 \to \alpha_3} \;,\; \dfrac{(\text{AxSK})\,\overline{\Gamma_2' \vdash \texttt{g} :: [\texttt{Char}] \to [[\texttt{Char}]], \emptyset} \;,\; \dfrac{(\text{AxSK})\,\overline{\Gamma_2' \vdash \texttt{g} :: [\texttt{Char}] \to [[\texttt{Char}]], \emptyset} \;,\; (\text{AxK})\,\overline{\Gamma_2' \vdash \texttt{'c'} :: \texttt{Char}, \emptyset}}{(\text{RApp})\,\Gamma_2' \vdash (\texttt{g}\ \texttt{'c'}) :: \alpha_7, \{[\texttt{Char}] \to [[\texttt{Char}]] \doteq \texttt{Char} \to \alpha_7\}}}{(\text{RApp})\,\Gamma_2' \vdash (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_4, \{[\texttt{Char}] \to [[\texttt{Char}]] \doteq \texttt{Char} \to \alpha_7, [\texttt{Char}] \to [[\texttt{Char}]] \doteq \alpha_7 \to \alpha_4\}}}{(\text{RApp})\,\Gamma_2' \vdash \texttt{Cons}\ x\ (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_2, \{[\texttt{Char}] \to [[\texttt{Char}]] \doteq \texttt{Char} \to \alpha_7, [\texttt{Char}] \to [[\texttt{Char}]] \doteq \alpha_7 \to \alpha_4 \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_1 \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}}{(\text{SKRek})\,\Gamma_2 \vdash_T \texttt{g} :: \sigma(\alpha_1 \to \alpha_2)}$$

wobei σ die Lösung von
$\{[\texttt{Char}] \to [[\texttt{Char}]] \doteq \texttt{Char} \to \alpha_7, [\texttt{Char}] \to [[\texttt{Char}]] \doteq \alpha_7 \to \alpha_4, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha_1 \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}$ ist.

Unifikation:

$$\dfrac{\dfrac{[\texttt{Char}] \to [[\texttt{Char}]] \doteq \texttt{Char} \to \alpha_7, \quad \ldots}{[\texttt{Char}] \doteq \texttt{Char}, \quad [[\texttt{Char}]] \doteq \alpha_7, \quad \ldots}}{Fail}$$

g ist nicht typisierbar.

Daher gilt ...

Satz

Das iterative Typisierungsverfahren benötigt unter Umständen mehrere Iterationen, bis ein Ergebnis (untypisiert / konsistente Annahme) gefunden wurde.

Beachte: Es gibt auch Beispiele, die zeigen, dass mehrere Iterationen nötig sind, um eine konsistente Annahme zu finden (Übungsaufgabe).

Nichtterminierung des iterativen Verfahrens

```
f = [g]
g = [f]
```

Es gilt $\mathtt{f} \simeq \mathsf{g}$, d.h. das iterative Verfahren typisiert $\mathtt{f}$ und g gemeinsam.
$\Gamma = \{\mathtt{Cons} :: \forall a.a \to [a] \to [a], \mathtt{Nil} : \forall a.a\}$.
$\Gamma_0 = \Gamma \cup \{\mathtt{f} :: \forall \alpha.\alpha, \mathsf{g} :: \forall \alpha.\alpha\}$

$$\dfrac{\dfrac{\dfrac{(\text{AxK})\ \overline{\Gamma_0 \vdash \mathtt{Cons} :: \alpha_4 \to [\alpha_4] \to [\alpha_4], \emptyset}\ ,\ (\text{AxSK})\ \overline{\Gamma_0 \vdash \mathsf{g} :: \alpha_5}}{(\text{RApp})\ \Gamma_0 \vdash (\mathtt{Cons}\ \mathsf{g}) :: \alpha_3, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq \alpha_5 \to \alpha_3\}}\ ,\ (\text{AxK})\ \overline{\Gamma_0 \vdash \mathtt{Nil} :: [\alpha_2], \emptyset}}{(\text{RApp})\ \Gamma_0 \vdash [\mathsf{g}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq \alpha_5 \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}}{(\text{SKRek})\ \dfrac{\Gamma_0 \vdash [\mathsf{g}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq \alpha_5 \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}{\Gamma_0 \vdash_T \mathtt{f} :: \sigma(\alpha_1) = [\alpha_5]}}$$

$\sigma = \{\alpha_1 \mapsto [\alpha_5], \alpha_2 \mapsto \alpha_5, \alpha_3 \mapsto [\alpha_5] \to [\alpha_5], \alpha_4 \mapsto \alpha_5\}$ ist Lösung von $\{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq \alpha_5 \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}$

Nichtterminierung des iterativen Verfahrens (2)

$$\dfrac{\dfrac{\dfrac{(\text{AxK})\ \overline{\Gamma_0 \vdash \mathtt{Cons} :: \alpha_4 \to [\alpha_4] \to [\alpha_4], \emptyset}\ ,\ (\text{AxSK})\ \overline{\Gamma_0 \vdash \mathtt{f} :: \alpha_5}}{(\text{RApp})\ \Gamma_0 \vdash (\mathtt{Cons}\ \mathtt{f}) :: \alpha_3, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq \alpha_5 \to \alpha_3\}}\ ,\ (\text{AxK})\ \overline{\Gamma_0 \vdash \mathtt{Nil} :: [\alpha_2], \emptyset}}{(\text{RApp})\ \Gamma_0 \vdash [\mathtt{f}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq \alpha_5 \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}}{(\text{SKRek})\ \dfrac{\Gamma_0 \vdash [\mathtt{f}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq \alpha_5 \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}{\Gamma_0 \vdash_T \mathsf{g} :: \sigma(\alpha_1) = [\alpha_5]}}$$

$\sigma = \{\alpha_1 \mapsto [\alpha_5], \alpha_2 \mapsto \alpha_5, \alpha_3 \mapsto [\alpha_5] \to [\alpha_5], \alpha_4 \mapsto \alpha_5\}$ ist Lösung von $\{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq \alpha_5 \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}$

Daher ist $\Gamma_1 = \Gamma \cup \{\mathtt{f} :: \forall a.[a], \mathsf{g} :: \forall a.[a]\}$. Da $\Gamma_1 \neq \Gamma_0$ muss man weiter iterieren.

Nichtterminierung des iterativen Verfahrens (3)

$$\dfrac{\dfrac{\dfrac{(\text{AxK})\ \overline{\Gamma_1 \vdash \mathtt{Cons} :: \alpha_4 \to [\alpha_4] \to [\alpha_4], \emptyset}\ ,\ (\text{AxSK})\ \overline{\Gamma_1 \vdash \mathsf{g} :: [\alpha_5]}}{(\text{RApp})\ \Gamma_1 \vdash (\mathtt{Cons}\ \mathsf{g}) :: \alpha_3, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5] \to \alpha_3\}}\ ,\ (\text{AxK})\ \overline{\Gamma_1 \vdash \mathtt{Nil} :: [\alpha_2], \emptyset}}{(\text{RApp})\ \Gamma_1 \vdash [\mathsf{g}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5] \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}}{(\text{SKRek})\ \dfrac{\Gamma_1 \vdash [\mathsf{g}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5] \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}{\Gamma_1 \vdash_T \mathtt{f} :: \sigma(\alpha_1) = [[\alpha_5]]}}$$

$\sigma = \{\alpha_1 \mapsto [[\alpha_5]], \alpha_2 \mapsto [\alpha_5], \alpha_3 \mapsto [[\alpha_5]] \to [[\alpha_5]], \alpha_4 \mapsto [\alpha_5]\}$ ist Lösung von $\{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5] \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}$

$$\dfrac{\dfrac{\dfrac{(\text{AxK})\ \overline{\Gamma_1 \vdash \mathtt{Cons} :: \alpha_4 \to [\alpha_4] \to [\alpha_4], \emptyset}\ ,\ (\text{AxSK})\ \overline{\Gamma_1 \vdash \mathtt{f} :: [\alpha_5]}}{(\text{RApp})\ \Gamma_1 \vdash (\mathtt{Cons}\ \mathtt{f}) :: \alpha_3, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5] \to \alpha_3\}}\ ,\ (\text{AxK})\ \overline{\Gamma_1 \vdash \mathtt{Nil} :: [\alpha_2], \emptyset}}{(\text{RApp})\ \Gamma_1 \vdash [\mathtt{f}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5] \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}}{(\text{SKRek})\ \dfrac{\Gamma_1 \vdash [\mathtt{f}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5] \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}{\Gamma_1 \vdash_T \mathsf{g} :: \sigma(\alpha_1) = [[\alpha_5]]}}$$

$\sigma = \{\alpha_1 \mapsto [[\alpha_5]], \alpha_2 \mapsto [\alpha_5], \alpha_3 \mapsto [[\alpha_5]] \to [[\alpha_5]], \alpha_4 \mapsto [\alpha_5]\}$ ist Lösung von $\{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5] \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}$

Daher ist $\Gamma_2 = \Gamma \cup \{\mathtt{f} :: \forall a.[[a]], \mathsf{g} :: \forall a.[[a]]\}$. Da $\Gamma_2 \neq \Gamma_1$ muss man weiter iterieren.

Nichtterminierung des iterativen Verfahrens (4)

Vermutung: Terminiert nicht
Beweis: (Induktion) betrachte den i. Schritt:
$\Gamma_i = \Gamma \cup \{\mathtt{f} :: \forall a.[a]^i, \mathtt{g} :: \forall a.[a]^i\}$ wobei $[a]^i$ i-fach geschachtelte Liste

$$\dfrac{\dfrac{\dfrac{}{\Gamma_i \vdash \mathtt{Cons} :: \alpha_4 \to [\alpha_4] \to [\alpha_4], \emptyset}\,(\text{AxK}) \quad , \dfrac{}{\Gamma_i \vdash \mathtt{g} :: [\alpha_5]^i}\,(\text{AxSK})}{\Gamma_i \vdash (\mathtt{Cons}\ \mathtt{g}) :: \alpha_3, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5]^i \to \alpha_3\}}\,(\text{RApp}) \quad , \dfrac{}{\Gamma_i \vdash \mathtt{Nil} :: [\alpha_2], \emptyset}\,(\text{AxK})}{\Gamma_i \vdash [\mathtt{g}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5]^i \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}\,(\text{RApp})}{\Gamma_i \vdash_T \mathtt{f} :: \sigma(\alpha_1) = [[\alpha_5]^i]}\,(\text{SKRek})$$

$\sigma = \{\alpha_1 \mapsto [[\alpha_5]^i], \alpha_2 \mapsto [\alpha_5]^i, \alpha_3 \mapsto [[\alpha_5]^i] \to [[\alpha_5]^i], \alpha_4 \mapsto [\alpha_5]^i\}$ ist Lösung von $\{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5]^i \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}$

$$\dfrac{\dfrac{\dfrac{}{\Gamma_i \vdash \mathtt{Cons} :: \alpha_4 \to [\alpha_4] \to [\alpha_4], \emptyset}\,(\text{AxK}) \quad , \dfrac{}{\Gamma_i \vdash \mathtt{f} :: [\alpha_5]^i}\,(\text{AxSK})}{\Gamma_i \vdash (\mathtt{Cons}\ \mathtt{f}) :: \alpha_3, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5]^i \to \alpha_3\}}\,(\text{RApp}) \quad , \dfrac{}{\Gamma_i \vdash \mathtt{Nil} :: [\alpha_2], \emptyset}\,(\text{AxK})}{\Gamma_i \vdash [\mathtt{f}] :: \alpha_1, \{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5]^i \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}}\,(\text{RApp})}{\Gamma_i \vdash_T \mathtt{g} :: \sigma(\alpha_1) = [[\alpha_5]^i]}\,(\text{SKRek})$$

$\sigma = \{\alpha_1 \mapsto [[\alpha_5]^i], \alpha_2 \mapsto [\alpha_5]^i, \alpha_3 \mapsto [[\alpha_5]^i] \to [[\alpha_5]^i], \alpha_4 \mapsto [\alpha_5]^i\}$ ist Lösung von $\{\alpha_4 \to [\alpha_4] \to [\alpha_4] \doteq [\alpha_5]^i \to \alpha_3, \alpha_3 \doteq [\alpha_2] \to \alpha_1\}$

D.h. $\Gamma_{i+1} = \Gamma \cup \{\mathtt{f} :: \forall a.[a]^{i+1}, \mathtt{g} :: \forall a.[a]^{i+1}\}$.

Daher ...

Satz

Das iterative Typisierungsverfahren terminiert nicht immer.

Es gilt sogar:

Theorem

Die iterative Typisierung ist unentscheidbar.

Dies folgt aus der Unentscheidbarkeit der so genannten Semi-Unifikation von First-Order Termen.

Aufrufhierarchie

- Das iterative Verfahren benötigt die Information aus der Aufrufhierarchie nicht:
- Es liefert die gleichen Typen, unabhängig davon, in welcher Reihenfolge man die Superkombinatoren typisiert.

Type Safety

Man spricht von Type Safety wenn gilt:

- Die Typisierung bleibt unter Reduktion erhalten („Type Preservation“)
  Genauer: Für einen Grundtyp τ: $t :: \tau$ vor der Reduktion $t \to t'$, dann auch $t' :: \tau$ danach.
  D.h. polymorphe Typen können auch allgemeiner werden.
- Getypte geschlossene Ausdrücke sind reduzibel, solange sie keine WHNF sind (“Progress Lemma“).

Type Safety (2)

Lemma

Sei s ein direkt dynamisch ungetypter KFPTS+seq-Ausdruck. Dann kann das iterative Typsystem keinen Typ für s herleiten.

Beweis: s direkt dynamisch ungetypt ist, g.d.w.:

- $s = R[\texttt{case}_T\ (c\ s_1\ \ldots\ s_n)\ \texttt{of}\ Alts]$ und c ist nicht vom Typ T. Typisierung von case fügt Gleichungen hinzu, so dass der Typ von $(c\ s_1\ \ldots\ s_n)$ und Typ von Pattern gleich ist. Daher wird die Unifikation scheitern.
- $s = R[\texttt{case}_T\ \lambda x.t\ \texttt{of}\ Alts]$: Analog, Gleichungen verlangen dass $(\lambda x.t)$ einen Funktionstyp erhält, Pattern aber nie einen solchen haben.
- $R[(c\ s_1\ \ldots\ s_{ar(c)})\ t]$: Typisierung typisiert die Anwendung $((c\ s_1\ \ldots\ s_{ar(c)})\ t)$ wie eine verschachtelte Anwendung $(((c\ s_1)\ \ldots)\ s_{ar(c)})\ t)$. Es werden Gleichungen hinzugefügt, die sicherstellen, dass c höchstens $ar(c)$ Argumente verarbeiten kann.

Type Safety (3)

Lemma (Type Preservation)

Sei s ein wohl-getypter, geschlossener KFPTS+seq-Ausdruck und $s \xrightarrow{name} s'$. Dann ist s' wohl-getypt.

Beweis: Hierfür muss man die einzelnen Fälle einer (β)-, $(SK-\beta)$- und (case)-Reduktion durchgehen. Für die Typherleitung von s kann man aus der Typherleitung einen Typ für jeden Unterterm von s ablesen. Bei der Reduktion werden diese Typen einfach mitkopiert.

Type Safety (4)

Aus den letzten beiden Lemmas folgt:

Satz

Sei s ein wohl-getypter, geschlossener KFPTS+seq-Ausdruck. Dann ist s nicht dynamisch ungetypt.

Lemma (Progress Lemma)

Sei s ein wohl-getypter, geschlossener KFPTS+seq-Ausdruck. Dann gilt:

- *s ist eine WHNF, oder*
- *s ist call-by-name reduzibel, d.h. $s \xrightarrow{name} s'$.*

Beweis Betrachtet man die Fälle, wann ein geschlossener KFPTS+seq-Ausdruck irreduzibel ist, so erhält man:
s ist eine WHNF oder s ist direkt-dynamisch ungetypt. Daher folgt das Lemma.

Type Safety (5)

Theorem

Die iterative Typisierung für KFPTS+seq erfüllt die „Type-safety"-Eigenschaft.

Erzwingen der Terminierung

- $SK_1, \ldots, SK_m$ ist Gruppe verschränkt rekursiver Superkombinatoren
- $\Gamma_i \vdash_T SK_1 :: \tau_1, \ldots, \Gamma_i \vdash_T SK_m :: \tau_m$ seien die durch die i. Iteration hergeleiteten Typen

Milner-Schritt: Typisiere $SK_1, \ldots, SK_m$ auf einmal, mit der Annahme:
$\Gamma_M = \Gamma \cup \{SK_1 :: \tau_1, \ldots, SK_m :: \tau_m\}$; ohne Quantoren
und der Regel: (nächste Folie)

Erzwingen der Terminierung (2)

$$(\text{SKREKM}) \quad \frac{\begin{array}{c}\text{für } i = 1, \ldots, m: \\ \Gamma_M \cup \{x_{i,1} :: \alpha_{i,1}, \ldots, x_{i,n_i} :: \alpha_{i,n_i}\} \vdash s_i :: \tau_i', E_i\end{array}}{\Gamma_M \vdash_T \text{für } i = 1, \ldots, m \; SK_i :: \sigma(\alpha_{i,1} \to \ldots \to \alpha_{i,n_i} \to \tau_i')}$$

wenn σ Lösung von $E_1 \cup \ldots \cup E_m \cup \bigcup_{i=1}^{m} \{\tau_i \doteq \alpha_{i,1} \to \ldots \to \alpha_{i,n_i} \to \tau_i'\}$

und $\begin{array}{lll} SK_1 \; x_{1,1} \; \ldots \; x_{1,n_1} & = & s_1 \\ \ldots & & \\ SK_m \; x_{m,1} \; \ldots \; x_{m,n_m} & = & s_m \end{array}$

die Definitionen von $SK_1, \ldots, SK_m$ sind

Als zusätzliche Regel muss im Typisierungsverfahren hinzugefügt werden:

$$(\text{AXSK2}) \quad \frac{}{\Gamma \cup \{SK :: \tau\} \vdash SK :: \tau}$$

wenn τ nicht allquantifiziert ist

Erzwingen der Terminierung (3)

Unterschied zum iterativen Schritt:

- Die Typen der zu typisierenden SKs werden nicht allquantifiziert.
- Daher sind während der Typisierung keine Kopien dieser Typen möglich
- Am Ende werden die angenommenen Typen mit den hergeleiteten Typen unifiziert.

Daraus folgt:
Die neue Annahme, die man durch die (SKREKM)-Regel herleiten kann, ist stets konsistent.

Nach einem Milner-Schritt terminiert das Verfahren sofort.

Das Milner-Typisierungsverfahren

Milner-Typisierung ist analog zum iterativen Typisierungsverfahren.

Unterschiede:

- Es wird nur ein Iterationsschritt durchgeführt.
- Die aktuell zu typisierenden Superkombinatoren SK_i sind mit allgemeinstem Typ α_i (ohne Allquantor) in den Annahmen.

Haskell verwendet das Milner-Typisierungs-Verfahren.

Das Milner-Typisierungsverfahren, genauer

Milner-Typisierungsverfahren: $SK_1, \ldots, SK_m$ alle SKs einer Äquivalenzklasse bzgl. $\simeq$ wobei alle kleineren SKs bereits getypt

1. Annahme Γ enthält Typen der bereits typisierten SKs und Konstruktoren (allquantifiziert)
2. Typisiere $SK_1, \ldots, SK_m$ mit der Regel $(\mathrm{MSKR}_{\mathrm{EK}})$:

$$(\mathrm{MSKR}_{\mathrm{EK}}) \frac{\begin{array}{c}\text{für } i = 1, \ldots, m: \\ \Gamma \cup \{SK_1 :: \beta_1, \ldots, SK_m :: \beta_m\} \\ \cup \{x_{i,1} :: \alpha_{i,1}, \ldots, x_{i,n_i} :: \alpha_{i,n_i}\} \vdash s_i :: \tau_i, E_i\end{array}}{\Gamma \vdash_T \text{für } i = 1, \ldots, m \; SK_i :: \sigma(\alpha_{i,1} \to \ldots \to \alpha_{i,n_i} \to \tau_i)}$$

wenn σ Lösung von $E_1 \cup \ldots \cup E_m \cup \bigcup_{i=1}^{m} \{\beta_i \doteq \alpha_{i,1} \to \ldots \to \alpha_{i,n_i} \to \tau_i\}$

und $\begin{array}{lcl} SK_1 \; x_{1,1} \; \ldots \; x_{1,n_1} & = & s_1 \\ \ldots & & \\ SK_m \; x_{m,1} \; \ldots \; x_{m,n_m} & = & s_m \end{array}$

die Definitionen von $SK_1, \ldots, SK_m$ sind

Falls Unifikation fehlschlägt, sind $SK_1, \ldots, SK_m$ nicht Milner-typisierbar

Das Milner-Typisierungsverfahren (2)

Vereinfachung: Regel für einen rekursiven SK

$$(\mathrm{MSKR}_{\mathrm{EK}}1) \frac{\Gamma \cup \{SK :: \beta, x_1 :: \alpha_1, \ldots, x_n :: \alpha_n\} \vdash s :: \tau, E}{\Gamma \vdash_T SK :: \sigma(\alpha_1 \to \ldots \to \alpha_n \to \tau)}$$

wenn σ Lösung von $E \cup \{\beta \doteq \alpha_1 \to \ldots \to \alpha_n \to \tau\}$
und $SK \; x_1 \; \ldots \; x_n = s$ die Definition von SK ist

Eigenschaften des Milner-Typcheck

Für das Milner-Typisierungsverfahren gelten die folgenden Eigenschaften:

- Das Verfahren terminiert.
- Das Verfahren liefert eindeutige Typen (bis auf Umbenennung von Variablen)
- Die Milner-Typisierung ist entscheidbar.
- Das Problem, ob ein Ausdruck Milner-typisierbar ist, ist DEXPTIME-vollständig
- Das Verfahren liefert u.U. eingeschränktere Typen als das iterative Verfahren. Insbesondere kann ein Ausdruck iterativ typisierbar, aber nicht Milner-typisierbar sein.
- Das Milner-Typisierungsverfahren benötigt das Wissen um die Aufrufhierarchie der Superkombinatoren: Es berechnet evtl. weniger allgemeine Typen bzw. Typisierung schlägt fehl, wenn man nicht von unten nach oben typisiert.

Beispiele

Man benötigt manchmal exponentiell viele Typvariablen
(in der Größe des Ausdrucks):

```
(let x0 = \z->z in
  (let x1 = (x0,x0) in
    (let x2 = (x1,x1) in
      (let x3 = (x2,x2) in
         (let x4 = (x3,x3) in
            (let x5 = (x4,x4) in
               (let x6 = (x5,x5) in x6)))))))
```

Die Anzahl der Typvariablen ist 2^6.
Verallgemeinert man das Beispiel, dann sind es 2^n.

Beispiele: map

```
map f xs = case xs of {
             [] -> []
             (y:ys) -> (f y):(map f ys)
           }
```

$\Gamma_0 = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{Nil} :: \forall a.[a]\}$

Sei $\Gamma = \Gamma_0 \cup \{\texttt{map} :: \beta, f :: \alpha_1, xs :: \alpha_2\}$ und $\Gamma' = \Gamma \cup \{y : \alpha_3, ys :: \alpha_4\}$.

$$\text{(RCase)}\ \frac{\begin{array}{l}(a)\quad \Gamma \vdash xs :: \tau_1, E_1\\ (b)\quad \Gamma \vdash \texttt{Nil} :: \tau_2, E_2\\ (c)\quad \Gamma' \vdash (\texttt{Cons}\ y\ ys) :: \tau_3, E_3\\ (d)\quad \Gamma \vdash \texttt{Nil} :: \tau_4, E_4\\ (e)\quad \Gamma' \vdash (\texttt{Cons}\ (f\ y)\ (\texttt{map}\ f\ ys)) :: \tau_5, E_5\end{array}}{\Gamma \vdash \texttt{case}\ xs\ \texttt{of}\ \{\texttt{Nil} \to \texttt{Nil}; \texttt{Cons}\ y\ ys \to \texttt{Cons}\ y\ (\texttt{map}\ f\ ys)\} :: \alpha, E}$$

$$\text{(MSKRek1)}\ \frac{}{\Gamma \vdash_T \texttt{map} :: \sigma(\alpha_1 \to \alpha_2 \to \alpha)}\ \text{wenn } \sigma \text{ Lösung von } E \cup \{\beta \doteq \alpha_1 \to \alpha_2 \to \alpha\}$$

wobei $E = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup \{\tau_1 \doteq \tau_2, \tau_1 \doteq \tau_3, \alpha \doteq \tau_4, \alpha \doteq \tau_5\}$.

(a) bis (e) folgt

Beispiele: map (2)

(a) $\overset{\text{(AxV)}}{\dfrac{}{\Gamma \vdash xs :: \alpha_2, \emptyset}}$

D.h. $\tau_1 = \alpha_2$ und $E_1 = \emptyset$.

(b) $\overset{\text{(AxK)}}{\dfrac{}{\Gamma \vdash \texttt{Nil} :: [\alpha_5], \emptyset}}$

D.h. $\tau_2 = [\alpha_5]$ und $E_2 = \emptyset$

(c) $\dfrac{\dfrac{\overset{\text{(AxK)}}{\dfrac{}{\Gamma' \vdash \texttt{Cons} :: \alpha_6 \to [\alpha_6] \to [\alpha_6]}}\ ,\ \overset{\text{(AxV)}}{\dfrac{}{\Gamma' \vdash y :: \alpha_3, \emptyset}}}{\Gamma' \vdash (\texttt{Cons}\ y) :: \alpha_7, \{\alpha_6 \to [\alpha_6] \to [\alpha_6] \doteq \alpha_3 \to \alpha_7\}}\ \text{(RApp)}\ ,\ \overset{\text{(AxV)}}{\dfrac{}{\Gamma' \vdash ys :: \alpha_4, \emptyset}}}{\Gamma' \vdash (\texttt{Cons}\ y\ ys) :: \alpha_8, \{\alpha_6 \to [\alpha_6] \to [\alpha_6] \doteq \alpha_3 \to \alpha_7, \alpha_7 \doteq \alpha_4 \to \alpha_8\}}\ \text{(RApp)}$

D.h. $\tau_3 = \alpha_8$ und $E_3 = \{\alpha_6 \to [\alpha_6] \to [\alpha_6] \doteq \alpha_3 \to \alpha_7, \alpha_7 \doteq \alpha_4 \to \alpha_8\}$

(d) $\overset{\text{(AxK)}}{\dfrac{}{\Gamma \vdash \texttt{Nil} :: [\alpha_9], \emptyset}}$

D.h. $\tau_4 = [\alpha_9]$ und $E_4 = \emptyset$.

Beispiele: map (3)

(e)

$$\dfrac{\dfrac{\overset{\text{(AxK)}}{\dfrac{}{\Gamma' \vdash \texttt{Cons} :: \alpha_{10} \to [\alpha_{10}] \to [\alpha_{10}], \emptyset}}\ ,\ \dfrac{\overset{\text{(AxV)}}{\dfrac{}{\Gamma' \vdash f :: \alpha_1, \emptyset}}\ ,\ \overset{\text{(AxV)}}{\dfrac{}{\Gamma' \vdash y :: \alpha_3, \emptyset}}}{\Gamma' \vdash (f\ y) :: \alpha_{15}, \{\alpha_1 \doteq \alpha_3 \to \alpha_{15}\}}\ \text{(RApp)}}{\Gamma' \vdash (\texttt{Cons}\ (f\ y)) :: \alpha_{11}, \{\alpha_{10} \to [\alpha_{10}] \to [\alpha_{10}] \doteq \alpha_{15} \to \alpha_{11}, \alpha_1 \doteq \alpha_3 \to \alpha_{15}\}}\ \text{(RApp)}\ ,\ \dfrac{\overset{\text{(AxSK2)}}{\dfrac{}{\Gamma' \vdash \texttt{map} :: \beta, \emptyset}}\ ,\ \overset{\text{(AxV)}}{\dfrac{}{\Gamma' \vdash f :: \alpha_1, \emptyset}}}{\Gamma' \vdash (\texttt{map}\ f) :: \alpha_{12}, \{\beta \doteq \alpha_1 \to \alpha_{12}\}}\ \text{(RApp)}\ ,\ \overset{\text{(AxV)}}{\dfrac{}{\Gamma' \vdash ys :: \alpha_4, \emptyset}}}{\Gamma' \vdash (\texttt{Cons}\ (f\ y)\ (\texttt{map}\ f\ ys)) :: \alpha_{14}, \{\alpha_{11} \doteq \alpha_{13} \to \alpha_{14}, \alpha_{10} \to [\alpha_{10}] \to [\alpha_{10}] \doteq \alpha_{15} \to \alpha_{11}, \alpha_1 \doteq \alpha_3 \to \alpha_{15}, \beta \doteq \alpha_1 \to \alpha_{12}, \alpha_{12} \doteq \alpha_4 \to \alpha_{13}\}}\ \text{(RApp)}$$

D.h. $\tau_5 = \alpha_{14}$ und

$E_5 = \{\alpha_{11} \doteq \alpha_{13} \to \alpha_{14}, \alpha_{10} \to [\alpha_{10}] \to [\alpha_{10}] \doteq \alpha_{15} \to \alpha_{11}, \alpha_1 \doteq \alpha_3 \to \alpha_{15},$
$\beta \doteq \alpha_1 \to \alpha_{12}, \alpha_{12} \doteq \alpha_4 \to \alpha_{13}\}$

Beispiele: map (4)

Gleichungssystem $E \cup \{\beta \doteq \alpha_1 \to \alpha_2 \to \alpha\}$ durch Unifikation lösen:

$$\begin{array}{l}\{\alpha_6 \to [\alpha_6] \to [\alpha_6] \doteq \alpha_3 \to \alpha_7, \alpha_7 \doteq \alpha_4 \to \alpha_8, \alpha_{11} \doteq \alpha_{13} \to \alpha_{14},\\ \alpha_{10} \to [\alpha_{10}] \to [\alpha_{10}] \doteq \alpha_{15} \to \alpha_{11}, \alpha_1 \doteq \alpha_3 \to \alpha_{15}, \beta \doteq \alpha_1 \to \alpha_{12},\\ \alpha_{12} \doteq \alpha_4 \to \alpha_{13}, \alpha_2 \doteq [\alpha_5], \alpha_2 \doteq \alpha_8, \alpha \doteq \alpha_9, \alpha \doteq \alpha_{14},\\ \beta \doteq \alpha_1 \to \alpha_2 \to \alpha\}\end{array}$$

Die Unifikation ergibt

$$\begin{array}{l}\sigma = \{\alpha \mapsto [\alpha_{10}], \alpha_1 \mapsto \alpha_6 \to \alpha_{10}, \alpha_2 \mapsto [\alpha_6], \alpha_3 \mapsto \alpha_6, \alpha_4 \mapsto [\alpha_6], \alpha_5 \mapsto \alpha_6,\\ \alpha_7 \mapsto [\alpha_6] \to [\alpha_6], \alpha_8 \mapsto [\alpha_6], \alpha_9 \mapsto [\alpha_{10}], \alpha_{11} \mapsto [\alpha_{10}] \to [\alpha_{10}],\\ \alpha_{12} \mapsto [\alpha_6] \to [\alpha_{10}], \alpha_{13} \mapsto [\alpha_{10}], \alpha_{14} \mapsto [\alpha_{10}], \alpha_{15} \mapsto \alpha_{10},\\ \beta \mapsto (\alpha_6 \to \alpha_{10}) \to [\alpha_6] \to [\alpha_{10}],\end{array}$$

D.h. $map :: \sigma(\alpha_1 \to \alpha_2 \to \alpha) = (\alpha_6 \to \alpha_{10}) \to [\alpha_6] \to [\alpha_{10}]$.

Beispiele: erneute Betrachtung

```
g x = x : (g (g 'c'))
```

Iteratives Verfahren liefert Fail nach mehreren Iteration.
Milner: $\Gamma = \{\texttt{Cons} :: \forall a.a \to [a] \to [a], \texttt{'c'} :: \texttt{Char}\}$.
Sei $\Gamma' = \Gamma \cup \{x :: \alpha, \texttt{g} :: \beta\}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \texttt{Cons} :: \alpha_5 \to [\alpha_5] \to [\alpha_5], \emptyset}\,(\text{AxK}) \quad \dfrac{}{\Gamma \vdash x :: \alpha, \emptyset}\,(\text{AxV})}{\Gamma \vdash (\texttt{Cons}\ x) :: \alpha_3, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha \to \alpha_3}\,(\text{RApp}) \quad \dfrac{\dfrac{}{\Gamma \vdash \texttt{g} :: \beta, \emptyset}\,(\text{AxSK2}) \quad \dfrac{\dfrac{}{\Gamma \vdash \texttt{g} :: \beta, \emptyset}\,(\text{AxSK2}) \quad \dfrac{}{\Gamma \vdash \texttt{'c'} :: \texttt{Char}, \emptyset}\,(\text{AxK})}{\Gamma \vdash (\texttt{g}\ \texttt{'c'}) :: \alpha_7, \{\beta \doteq \texttt{Char} \to \alpha_7\}}\,(\text{RApp})}{\Gamma \vdash (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_4, \{\beta \doteq \texttt{Char} \to \alpha_7, \beta \doteq \alpha_7 \to \alpha_4\}}\,(\text{RApp})}{\Gamma \vdash \texttt{Cons}\ x\ (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_2, \{\beta \doteq \texttt{Char} \to \alpha_7, \beta \doteq \alpha_7 \to \alpha_4 \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}}\,(\text{MSKRek})$$

$$\Gamma \vdash_T \texttt{g} :: \sigma(\alpha \to \alpha_2)$$

wobei σ die Lösung von

$\{\beta \doteq \texttt{Char} \to \alpha_7, \beta \doteq \alpha_7 \to \alpha_4, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \alpha \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2, \beta \doteq \alpha \to \alpha_2\}$ ist.

Die Unifikation schlägt jedoch fehl, da `Char` mit einer Liste unifiziert werden soll. D.h. g ist nicht Milner-typisierbar.

Beispiele: erneute Betrachtung (2)

```
g x = 1 : (g (g 'c'))
```

Iteratives Verfahren liefert $g :: \forall \alpha.\alpha \to [\texttt{Int}]$
Milner: Sei $\Gamma' = \Gamma \cup \{x :: \alpha, \texttt{g} :: \beta\}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash \texttt{Cons} :: \alpha_5 \to [\alpha_5] \to [\alpha_5], \emptyset}\,(\text{AxK}) \quad \dfrac{}{\Gamma \vdash 1 :: \texttt{Int}, \emptyset}\,(\text{AxK})}{\Gamma \vdash (\texttt{Cons}\ 1) :: \alpha_3, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \texttt{Int} \to \alpha_3}\,(\text{RApp}) \quad \dfrac{\dfrac{}{\Gamma \vdash \texttt{g} :: \beta, \emptyset}\,(\text{AxSK2}) \quad \dfrac{\dfrac{}{\Gamma \vdash \texttt{g} :: \beta, \emptyset}\,(\text{AxSK2}) \quad \dfrac{}{\Gamma \vdash \texttt{'c'} :: \texttt{Char}, \emptyset}\,(\text{AxK})}{\Gamma \vdash (\texttt{g}\ \texttt{'c'}) :: \alpha_7, \{\beta \doteq \texttt{Char} \to \alpha_7\}}\,(\text{RApp})}{\Gamma \vdash (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_4, \{\beta \doteq \texttt{Char} \to \alpha_7, \beta \doteq \alpha_7 \to \alpha_4\}}\,(\text{RApp})}{\Gamma \vdash \texttt{Cons}\ 1\ (\texttt{g}\ (\texttt{g}\ \texttt{'c'})) :: \alpha_2, \{\beta \doteq \texttt{Char} \to \alpha_7, \beta \doteq \alpha_7 \to \alpha_4, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \texttt{Int} \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2\}}\,(\text{SKRek})$$

$$\Gamma \vdash_T \texttt{g} :: \sigma(\alpha \to \alpha_2)$$

wobei σ die Lösung von

$\{\beta \doteq \texttt{Char} \to \alpha_7, \beta \doteq \alpha_7 \to \alpha_4, \alpha_5 \to [\alpha_5] \to [\alpha_5] \doteq \texttt{Int} \to \alpha_3, \alpha_3 \doteq \alpha_4 \to \alpha_2, \beta \doteq \alpha \to \alpha_2\}$ ist.

Die Unifikation schlägt fehl, da $[\alpha_5] \doteq \texttt{Char}$ unifiziert werden soll.

Iteratives Verfahren kann allgemeinere Typen liefern

```
data Baum a = Leer | Knoten a (Baum a) (Baum a)
```

Die Typen für die Konstruktoren sind
`Leer` $:: \forall a.$ `Baum` a und
`Knoten` $:: \forall a.\ a \to$ `Baum a` $\to$ `Baum a` $\to$ `Baum a`

```
g x y = Knoten True (g x y) (g y x)
```

Milner-Typcheck `g` $:: a \to a \to$ `Baum Bool`
Iteratives Verfahren: `g` $:: a \to b \to$ `Baum Bool`
Grund:
Iteratives Verfahren erlaubt Kopien des Typs für g, Milner nicht.

Milner Typisierung und Type Safety

- Milner-getypte Programme sind immer auch iterativ typisierbar
- Daher sind Milner getypte Programme niemals dynamisch ungetypt
- Es gilt auch das Progress-Lemma: Milner getypte (geschlossene) Programme sind WHNFs oder reduzibel

Milner Typisierung und Type Safety (2)

- Type-Preservation: Gilt in KFPTSP+seq, aber vermutlich nicht in Haskell:

```
let x = (let y = \u -> z in (y [], y True, seq x True))
    z = const z x
in x
```

ist Milner-typisierbar.

Wenn man eine so genannte $(llet)$-Reduktion durchführt, erhält man:

```
let x = (y [], y True, seq x True)
    y = \u -> z
    z = const z x
in x
```

Ist nicht mehr Milner-typisierbar

Milner Typisierung und Type Safety (2)

Milner-typisierbar:

```
let x = (let y = \u -> z in (y [], y True, seq x True))
    z = const z x
in x
```

nicht Milner typisierbar (aber iterativ typisierbar):

```
let x = (y [], y True, seq x True)
    y = \u -> z
    z = const z x
in x
```

- Der Effekt kommt von der Allquantifizierung nach erfolgreicher Typisierung:

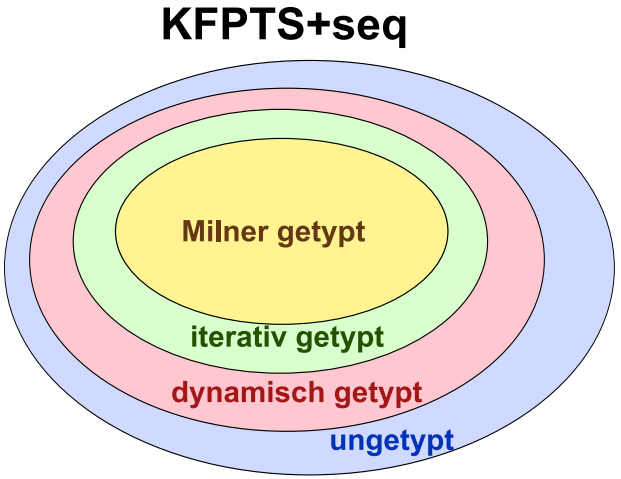
  Vorher: einmal kann allquantifiziert werden
  Nachher: alles wird auf einmal typisiert.

Milner Typisierung und Type Safety

Das Beispiel ist aber unkritisch, denn:

- Type-Preservation gilt für das iterative Verfahren;
- typisierte Programm sind dynamisch getypt;
- Milner-typisierbar impliziert iterativ typisierbar und
- Reduktion erhält iterative Typisierbarkeit

Übersicht



Prädikativ / Imprädikativ

- Prädikativer Polymorphismus: Typvariablen stehen für Grundtypen (= Haskell, KFPTSP+seq)
- Imprädikativer Polymorphismus: Typvariablen stehen auch für polymorphe Typen (mit Quantoren!)

Versuch `\x -> const (x True) (x 'A')` zu typisieren:

`x` ist eine Funktion, die für alle Eingabetypen den gleichen Ergebnistyp liefert

Mit imprädikativen Polymorphismus geht das:

```
(\x -> const (x True) (x 'A'))::(forall b.(forall a. a -> b) -> b)
```

Aber:

- Kein Haskell, sondern Erweiterung
- Typinferenz / Typisierbarkeit nicht mehr entscheidbar!

Beispiel mit Erweiterung: RankNTypes

```
*> :set -XRankNTypes

*> :t ((\x -> const (x True) (x 'A'))::(forall b.(forall a. a -> b) -> b))

((\x -> const (x True) (x 'A'))::(forall b.(forall a. a -> b) -> b)) :: (forall a. a -> b) -> b

*> :t ((\x -> const (x True) (x 'A'))::(forall b.(forall a. a -> b) -> b)) (const "Hallo")

((\x -> const (x True) (x 'A'))::(forall b.(forall a. a -> b) -> b)) (const "Hallo") :: [Char]

*> ((\x -> const (x True) (x 'A'))::(forall b.(forall a. a -> b) -> b)) (const "Hallo")
"Hallo"
```


RankNTypes

Normale Typen sind Rank-1-Typen:

- In `a -> b -> a` sind alle Typvariablen allquantifiziert. D.h. der Typ ist äquivalent zu `forall a b. a -> b -> a`
- `forall` in rechten Seiten des Funktionspfeils darf man nach oben schieben.
- D.h. z.B. `forall a. a -> (forall b. b -> a)` ist äquivalent zu `forall a b. a -> b -> a` und damit auch Rank-1

Rank-N-Typen

- haben `forall` auch **links** vom Funktionspfeil
- dies darf man **nicht** hochschieben
- N ist die Anzahl der Schachtelungen
- Z.B. `forall b.(forall a. a -> b) ->b` ist Rank-2
- `(forall a. a -> a) -> (forall b. b -> b)` ist auch Rank-2
- `((forall a. a -> a) -> Int) -> Bool -> Bool` ist Rank-3, (nicht 4, da gleich zu `((forall a. a -> a) -> Int) -> (Bool -> Bool)`)