

Einführung in die funktionale Programmierung

Prof. Dr. Manfred Schmidt-Schauß

Künstliche Intelligenz und Softwaretechnologie

4. Dezember 2007

Typregel für Anwendungen:

$$\frac{s :: \sigma \rightarrow \tau, \quad t :: \sigma}{(s \ t) :: \tau}$$

Typ-Regel für n -stellige Funktions-Anwendungen:

$$\frac{s :: \sigma_1 \rightarrow \sigma_2 \dots \rightarrow \sigma_n \rightarrow \tau \quad , \quad t_1 :: \sigma_1 \quad , \quad \dots \quad , \quad t_n :: \sigma_n}{(s \ t_1 \ \dots \ t_n) :: \tau}$$

Beispiel (+ 1 2):

$$\frac{+ :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}, \quad 1 :: \text{Int} \quad , \quad 2 :: \text{Int}}{(+ \ 1 \ 2) :: \text{Int}}$$

Typsubstitution γ : setzt Typen für Typvariablen ein

$\gamma(\tau)$ ist eine *Instanz* von τ .

Beispiel

Typsubstitution $\gamma = \{a \mapsto \text{Char}, b \mapsto \text{Float}\}$
Instanz: $\gamma([a] \rightarrow \text{Int}) = [\text{Char}] \rightarrow \text{Int}$

$$\frac{s :: \sigma \rightarrow \tau, \quad t :: \rho \text{ und } \gamma(\sigma) = \gamma(\rho)}{(s \ t) :: \gamma(\tau)}$$

wobei die Typvariablen in ρ umbenannt sind

Berechne den Typ von `map quadrat`

$\text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b]$

Typsubstitution: $\{a \mapsto \text{Int}, b \mapsto \text{Int}\}$,

Instanztyp von `map`: $(\text{Int} \rightarrow \text{Int}) \rightarrow [\text{Int}] \rightarrow [\text{Int}]$

$$\frac{\text{map} :: (\text{Int} \rightarrow \text{Int}) \rightarrow [\text{Int}] \rightarrow [\text{Int}], \quad \text{quadrat} :: (\text{Int} \rightarrow \text{Int})}{(\text{map quadrat}) :: [\text{Int}] \rightarrow [\text{Int}]}$$

$$\frac{s :: \sigma_1 \rightarrow \sigma_2 \dots \rightarrow \sigma_n \rightarrow \tau, \quad t_1 : \rho_1, \dots, t_n : \rho_n \text{ und } \forall i : \gamma(\sigma_i) = \gamma(\rho_i)}{(s \ t_1 \ \dots \ t_n) :: \gamma(\tau)}$$

Typvariablen von $\sigma_1 \rightarrow \sigma_2 \dots \rightarrow \sigma_n \rightarrow \tau$

disjunkt

zu den Typvariablen von ρ_1, \dots, ρ_n

Erforderlich: die **allgemeinste** Typsubstitution γ

Berechne Typ von `map id`

`id :: a -> a`

`id x = x`

Umbenennung: `id :: a' -> a'`

$$\frac{\text{map} :: (a \rightarrow b) \rightarrow ([a] \rightarrow [b]), \quad \text{id} :: a' \rightarrow a'}{(\text{map id}) :: \gamma([a] \rightarrow [b])}$$

mit allgemeinstem $\gamma = \{b \mapsto a, a' \mapsto a\}$.

d.h. `(map id) :: ([a] -> [a])`.

Unifikation: Berechnen der allgemeinsten Typsubstitution:

Gegeben: Typen $\delta_i, \rho_i, i = 1, \dots, n$
Berechne Typsubstitution γ so dass $\forall i : \gamma(\delta_i) = \gamma(\rho_i)$.

Verfahren: sukzessives Lösen von Typ-Gleichungen

Sei $\{\delta_i =_{\gamma} \rho_i, i = 1, \dots, n\}$ eine Multimenge von Gleichungen,

Umformungsregeln

$$\text{(Dekomposition)} \quad \frac{(TC \ \sigma_1 \dots \sigma_m) =_{\gamma} (TC \ \tau_1 \dots \tau_m), G \quad \text{und } TC \neq TC'}{\sigma_1 =_{\gamma} \tau_1, \dots, \sigma_m =_{\gamma} \tau_m, G}$$

$$\text{(Dekomp – Fail)} \quad \frac{(TC \ \sigma_1 \dots \sigma_m) =_{\gamma} (TC' \ \tau_1 \dots \tau_n), G}{\text{Abbruch}}$$

(Ersetzung)
$$\frac{a =_{\gamma} \sigma, G \quad \text{und } a \text{ kommt nicht in } \sigma \text{ vor}}{a =_{\gamma} \sigma, G[\sigma/a]}$$

(Occurs – Check)
$$\frac{a =_{\gamma} \sigma, G \quad \text{und } a \text{ kommt in } \sigma \text{ vor}}{\text{Abbruch}}$$

(Vereinfachung)
$$\frac{a =_{\gamma} a, G}{G}$$

(Vertauschung)
$$\frac{\sigma =_{\gamma} \tau, G}{\tau =_{\gamma} \sigma, G}$$

Vertauschung muss gesteuert werden:

Z.B. Vertauschung nicht erlaubt,
wenn eine Gleichung erzeugt wird,
die Vorgänger im Berechnungsbaum ist.

Gleichungen sind erfolgreich gelöst,

wenn G die Form

$a_1 =_{\gamma} \tau_1, \dots, a_k =_{\gamma} \tau_k$ hat,

wobei a_i Typvariablen sind, und

die a_i in keiner rechten Seite τ_j einer Gleichung auftreten.

Lösung ist:

$$\gamma = \{a_1 \mapsto \tau_1, \dots, a_k \mapsto \tau_k\}$$

Typisierung von `(map id)`

$$\frac{\text{map} :: (a \rightarrow b) \rightarrow [a] \rightarrow [b], \text{id} :: a' \rightarrow a'}{(\text{map id}) :: \gamma([a] \rightarrow [b])}$$

Berechnen von γ :

$$\frac{a \rightarrow b =_{\gamma} a' \rightarrow a'}{a =_{\gamma} a', b =_{\gamma} a'} \quad \text{mittels Dekomposition für } \rightarrow.$$

Erfolgreich gelöst!

berechnete Typsubstitution: $\gamma = \{a \mapsto a', b \mapsto a'\}$

Andere Reihenfolge der Regelanwendungen:

$$\frac{\frac{\frac{a \rightarrow b =_{\gamma} a' \rightarrow a'}{a =_{\gamma} a', b =_{\gamma} a'}}{a' =_{\gamma} a, b =_{\gamma} a'}}{a' =_{\gamma} a, b =_{\gamma} a}$$

Lösung:

$$\gamma = \{a' \mapsto a, b \mapsto a\}$$

ergibt äquivalenten Typ (vorher: $\{a \mapsto a', b \mapsto a'\}$)

Typ von `(concat . reverse)`

`(.)` :: $(b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c$

`concat` :: $[[a]] \rightarrow [a]$

`reverse` :: $[a] \rightarrow [a]$

$(.)$:: $(b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow (a \rightarrow c)$, $\text{concat} :: [[a']] \rightarrow [a']$ $\text{reverse} :: [a''] \rightarrow [a'']$
 $(\text{concat} . \text{reverse}) :: \gamma(a \rightarrow c)$

$$(b \rightarrow c) =_{\gamma} [[a']] \rightarrow [a'], \quad (a \rightarrow b) =_{\gamma} [a''] \rightarrow [a'']$$

$$\begin{aligned} \xrightarrow{\text{Dekomp}} \quad & b = [[a']], c = [a'], a = [a''], b = [a''] \\ \xrightarrow{\text{Ersetz}} \quad & b = [[a']], c = [a'], a = [a''], [[a']] = [a''] \\ \xrightarrow{\text{Dekomp}} \quad & b = [[a']], c = [a'], a = [a''], [a'] = a'' \\ \xrightarrow{\text{Vert}} \quad & b = [[a']], c = [a'], a = [a''], a'' = [a'] \\ \xrightarrow{\text{Ersetz}} \quad & b = [[a']], c = [a'], a = [[a']], a'' = [a'] \end{aligned}$$

(concat . reverse): $\gamma(a \rightarrow c) = [[a']] \rightarrow [a']$

Definition

γ ist **allgemeiner als** γ' bzgl. V

wenn es ein δ gibt, so dass $\forall x \in V: \delta(\gamma(x)) = \gamma'(x)$.

I.a.: $V =$ Menge der Typvariablen in der Ursprungsgleichung

(γ, γ', δ sind Typsubstitutionen)

Mit $V = \{a_1\}$,

$\gamma = \{a_1 \mapsto (a_2, b_2)\}$ ist allgemeiner als $\gamma' = \{a_1 \mapsto (\text{Int}, \text{Int})\}$

Mit $\delta = \{a_2 \mapsto \text{Int}, b_2 \mapsto \text{Int}\}$ gilt:

$$\delta(\gamma(a_1)) = (\text{Int}, \text{Int}) = \gamma'(a_1).$$

Der regelbasierte Algorithmus zur Unifikation:

- Terminiert (bei kontrollierter Vertauschung)
- Berechnet **immer ein allgemeinstes** γ zu G wenn es mindestens eine Lösung von G gibt.
- Bricht ab, wenn es keine Lösung gibt
- Kann effizient implementiert werden $O(n * \log(n))$.
Der angegebene Algorithmus hat exponentielle worst-case Komplexität.
Grund dafür: die Ersetzungsregel kann verdoppeln.
Der Algorithmus wird polynomiell, wenn man Sharing beachtet

Berechne den Typ der Liste [1]

$1 :: \text{Int}; [] :: [b]; (:) :: a \rightarrow [a] \rightarrow [a]$

zu unifizieren: $\text{Int} =_{\gamma} a; [b] =_{\gamma} [a]$

Ergibt: $\gamma = \{a \mapsto \text{Int}, b \mapsto \text{Int}\}$

Typ: $\gamma[a] = [\text{Int}]$.

$[1, 'a']$ hat **keinen Typ** :

Wir nehmen als schon berechnet an: $'a' :: [\text{Char}]$

2. **Prüfe**: $1 : ['a']$.

Zu unifizieren: $\text{Int} =_{\gamma} a; [\text{Char}] =_{\gamma} [a]$

Das ergibt nach folgenden Schritten einen Fehler:

$$\begin{array}{l} \text{Int} =_{\gamma} a; [\text{Char}] =_{\gamma} [a] \\ \xrightarrow{\text{Dekomp}} \text{Int} =_{\gamma} a; \text{Char} =_{\gamma} a \\ \xrightarrow{\text{Vert}} a =_{\gamma} \text{Int}; \text{Char} =_{\gamma} a \\ \xrightarrow{\text{Ers}} a =_{\gamma} \text{Int}; \text{Char} =_{\gamma} \text{Int} \\ \text{Abbruch mit Fehler} \end{array}$$

```
> [1,'a']  
No instance for (Num Char)  
    arising from the literal '1' at <interactive>:1:1  
...
```

Typisierung muss noch erweitert werden auf:

- (rekursiv) definierte Funktionen,
- Lambda-Ausdrücke,
- case-Ausdrücke und
- let-Ausdrücke und
- List-Komprehensionen.

In Haskell und SML: [Typcheckalgorithmus von Robin Milner](#)

schlechte worst-case Komplexität: exponentieller Platzbedarf.

Satz (wohlgetypte Programme machen keine dynamischen Typfehler:)

Sei t geschlossener und getypter Haskell-Ausdruck.
Dann wird die Auswertung des Ausdrucks t nicht mit einem Typfehler abbrechen.