GOETHE UNIVERSITÄT FRANKFURT AM MAIN

www.uni-frankfurt.de

Programmierparadigmen und Compilerbau (PPDC)

2. Polymorphismus und Unifikation

Sommersemester 2021

PD Dr. Arne Nägel

*Basierend auf Unterlagen von Prof. Dr. Manfred Schmidt-Schauß und PD Dr. David Sabel

Anwendungsregel für polymorphe Typen

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Ziel: Anwendung der Typregel für z.B. `length` oder `map`

Neuer Begriff: γ ist eine *Typsubstitution*
wenn sie Typen für Typvariablen einsetzt
$\gamma(\tau)$ nennt man Instanz von τ

Beispiel

Typsubstitution: $\gamma = \{a \mapsto \texttt{Char}, b \mapsto \texttt{Float}\}$

Instanz: $\gamma([a] \to \texttt{Int}) = [\texttt{Char}] \to \texttt{Int}$

Anwendungsregel für polymorphe Typen

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

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

Berechnet den Typ von $(s\ t)$

wenn die Typen von s, t schon bekannt sind

Hierbei ist zu beachten:

Damit alle Freiheiten erhalten bleiben:

die Typvariablen im Typ ρ müssen vorher
umbenannt werden,
so dass σ und ρ keine gemeinsamen Typvariablen haben.

Beispiel zur polymorphen Anwendungsregel

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Typ von `(map quadrat)` ?

$$\begin{array}{lllll} \texttt{map} :: & (a \to b) & \to & ([a] \to [b]) \\ \texttt{quadrat} :: & \texttt{Int} & \to & \texttt{Int} \end{array}$$

Beispiel zur polymorphen Anwendungsregel

Typ von `(map quadrat)` ?

$$\begin{array}{lll} \texttt{map} :: & (a \to b) & \to \quad ([a] \to [b]) \\ \texttt{quadrat} :: & \texttt{Int} & \to \quad \texttt{Int} \end{array}$$

Instanziiere mit: $\gamma = \{a \mapsto \texttt{Int}, b \mapsto \texttt{Int}\}$

Beispiel zur polymorphen Anwendungsregel

Typ von `(map quadrat)` ?

$$\begin{array}{lll} \texttt{map} :: & (a \to b) & \to \quad ([a] \to [b]) \\ \texttt{quadrat} :: & \texttt{Int} & \to \quad \texttt{Int} \end{array}$$

Instanziiere mit: $\gamma = \{a \mapsto \texttt{Int}, b \mapsto \texttt{Int}\}$

ergibt: $\texttt{map} :: (\texttt{Int} \to \texttt{Int}) \to ([\texttt{Int}] \to [\texttt{Int}])$

Beispiel zur polymorphen Anwendungsregel

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Typ von `(map quadrat)` ?

$$\begin{array}{lll} \texttt{map} :: & (a \to b) & \to \quad ([a] \to [b]) \\ \texttt{quadrat} :: & \texttt{Int} & \to \quad \texttt{Int} \end{array}$$

Instanziiere mit: $\gamma = \{a \mapsto \texttt{Int}, b \mapsto \texttt{Int}\}$

ergibt: $\texttt{map} :: (\texttt{Int} \to \texttt{Int}) \to ([\texttt{Int}] \to [\texttt{Int}])$

Regelanwendung ergibt:

$$\frac{\texttt{map} :: (\texttt{Int} \to \texttt{Int}) \to ([\texttt{Int}] \to [\texttt{Int}]), \quad \texttt{quadrat} :: (\texttt{Int} \to \texttt{Int})}{(\texttt{map quadrat}) :: [\texttt{Int}] \to [\texttt{Int}]}$$

Polymorphe Regel für n Argumente

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

Die Typvariablen in $\rho_1, \ldots, \rho_n$ müssen vorher umbenannt werden.

Beachte: verwende möglichst allgemeines γ
(kann berechnet werden; s.u.)

Polymorphe Regel für n Argumente

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

Die Typvariablen in $\rho_1, \ldots, \rho_n$ müssen vorher umbenannt werden.

Beachte: verwende möglichst allgemeines γ
(kann berechnet werden; s.u.)

Wie berechnet man die Typsubstitutionen?

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Unifikation: Berechnung der allgemeinsten Typsubstitution im Typberechnungsprogramm bzw Typchecker

Unifikation wird benutzt im Typchecker von Haskell!

polymorphe Typen; Unifikation

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

$$\frac{s :: \sigma \to \tau, \qquad t :: \rho \text{ und } \gamma \text{ ist } \textbf{allgemeinster Unifikator} \text{ von } \sigma \doteq \rho}{(s\ t) :: \gamma(\tau)}$$

(die Typvariablen in ρ müssen vorher umbenannt werden.)

Unifikation: Regelbasierter Algorithmus

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Man braucht 4 Regeln, die auf $(E; G)$ operieren:

E: Menge von Typgleichungen,
G: Lösung; mit Komponenten der Form $x \mapsto t$.

Beachte: x, y ist Typvariable
s_i, t_i, s, t sind Typen,
f, g sind Typkonstruktoren

Unifikation: Die Regeln

Start mit $G = \emptyset$;

E: zu lösende Gleichung(en)

(E) $$\frac{G;\ \ \{x \doteq x\} \cup E}{G; E}$$

(V) $$\frac{G;\ \ \{t \doteq x\} \cup E}{G; \{x \doteq t\} \cup E}$$ Wenn t keine Variable ist

(K) $$\frac{G;\ \ \{x \doteq t\} \cup E}{G[t/x] \cup \{x \mapsto t\}; E[t/x]}$$ Wenn x nicht in t vorkommt

(Z) $$\frac{G;\ \ \{(f\ s_1\ \ldots\ s_n) \doteq (f\ t_1\ \ldots\ t_n)\} \cup E}{G; \{s_1 \doteq t_1, \ldots, s_n \doteq t_n)\} \cup E}$$

Ersetzung: $E[t/x]$: alle Vorkommen von x werden durch t ersetzt
Effekt: $G[t/x]$: jedes $y \mapsto s$ wird durch $y \mapsto s[t/x]$ ersetzt

Unifikation: Regelbasierter Algorithmus

Fehlerabbruch, wenn:

- $x \doteq t$ in E, $x \neq t$ und x kommt in t vor. (Occurs-Check)
- $(f(\ldots) \doteq g(\ldots)$ kommt in E vor und $f \neq g$. (Clash)

Fehlerabbruch bedeutet: nicht typisierbar

Beispiel mit Typvariablen

Berechne Typ von `(map id)`

`map`::	$(a \to b)$	$\to \ [a] \to [b]$
`id`::	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E

Beispiel mit Typvariablen

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Berechne Typ von `(map id)`

`map`::	$(a \to b)$	$\to ([a] \to [b])$
`id`::	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E
$\emptyset;$	$\{(a \to b) \doteq (a' \to a')\}$

Beispiel mit Typvariablen

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Berechne Typ von `(map id)`

`map`::	$(a \to b)$	$\to ([a] \to [b])$
`id`::	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E
$\emptyset;$	$\{(a \to b) \doteq (a' \to a')\}$

Beispiel mit Typvariablen

Berechne Typ von `(map id)`

`map::`	$(a \to b)$	$\to ([a] \to [b])$
`id::`	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E
$\emptyset;$	$\{(a \to b) \doteq (a' \to a')\}$
$\emptyset;$	$\{a \doteq a', b \doteq a'\}$

Beispiel mit Typvariablen

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Berechne Typ von `(map id)`

`map`::	$(a \to b)$	$\to ([a] \to [b])$
`id`::	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E
$\emptyset;$	$\{(a \to b) \doteq (a' \to a')\}$
$\emptyset;$	$\{a \doteq a', b \doteq a'\}$

Beispiel mit Typvariablen

Berechne Typ von `(map id)`

`map`::	$(a \to b)$	$\to ([a] \to [b])$
`id`::	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E
$\emptyset;$	$\{(a \to b) \doteq (a' \to a')\}$
$\emptyset;$	$\{a \doteq a', b \doteq a'\}$
$\{a \mapsto a'\};$	$\{b \doteq a'\}$

Beispiel mit Typvariablen

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Berechne Typ von `(map id)`

`map`::	$(a \to b)$	$\to ([a] \to [b])$
`id`::	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E
$\emptyset;$	$\{(a \to b) \doteq (a' \to a')\}$
$\emptyset;$	$\{a \doteq a', b \doteq a'\}$
$\{a \mapsto a'\};$	$\{b \doteq a'\}$

Beispiel mit Typvariablen

Berechne Typ von `(map id)`

`map`::	$(a \to b)$	$\to ([a] \to [b])$
`id`::	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E
$\emptyset$;	$\{(a \to b) \doteq (a' \to a')\}$
$\emptyset$;	$\{a \doteq a', b \doteq a'\}$
$\{a \mapsto a'\}$;	$\{b \doteq a'\}$
$\{a \mapsto a', b \mapsto a'\}$;	$\emptyset$

Beispiel mit Typvariablen

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Berechne Typ von `(map id)`

`map`::	$(a \to b)$	$\to ([a] \to [b])$
`id`::	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E
$\emptyset;$	$\{(a \to b) \doteq (a' \to a')\}$
$\emptyset;$	$\{a \doteq a', b \doteq a'\}$
$\{a \mapsto a'\};$	$\{b \doteq a'\}$
$\{a \mapsto a', b \mapsto a'\};$	$\emptyset$
$\{a \mapsto a', b \mapsto a'\};$	

Beispiel mit Typvariablen

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Berechne Typ von `(map id)`

`map`::	$(a \to b)$	$\to ([a] \to [b])$
`id`::	$a' \to a'$	
Gesuchter Typ:		$\gamma([a] \to [b])$

Regelanwendung benötigt Lösung γ von $(a \to b) \doteq (a' \to a')$:

G	E
$\emptyset$;	$\{(a \to b) \doteq (a' \to a')\}$
$\emptyset$;	$\{a \doteq a', b \doteq a'\}$
$\{a \mapsto a'\}$;	$\{b \doteq a'\}$
$\{a \mapsto a', b \mapsto a'\}$;	$\emptyset$
$\{a \mapsto a', b \mapsto a'\}$;	

Einsetzen der Lösung $\gamma = \{a \mapsto a', b \mapsto a'\}$
in $[a] \to [b]$ ergibt:
`(map id)` :: $([a'] \to [a'])$.

Typisierung von Funktionen und Ausdrücken

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Typcheckalgorithmus von Robin Milner (in Haskell und ML)

- unterstützt auch komplexere Sprachkonstrukte (Lambda-Ausdrücke, Listen-Komprehensionen)
- benutzt eine optimierte Variante der Unifikation
- liefert i.a. allgemeinste polymorphe Typen (seltene Ausnahmen bei rekursiven Definitionen)
- ist i.d.R schnell, aber
- schlechte worst-case Komplexität: in seltenen Fällen exponentieller Zeitbedarf

Typisierung von Funktionen, Milner

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Satz zur Milner-Typisierung in Haskell:

Sei t ein getypter Haskell-Ausdruck, ohne freie Variablen.
Dann wird die Auswertung des Ausdrucks t
nicht mit einem Typfehler abbrechen.