Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
13. Juni 2014

8. Übung zur Vorlesung

Programmierung und Modellierung

A8-1 ***Unifikation I*** Berechnen Sie jeweils den allgemeinsten Unifikator für die folgenden Typgleichungen mit Robinson's Unifikationsalgorithmus (Folie 10-50), falls möglich:

a) $\{\texttt{Int} \to (\alpha \to \beta) = \alpha \to (\beta \to \texttt{Int})\}$

b) $\{\alpha \to (\alpha \to \texttt{Int}) = (\texttt{Int} \to \beta) \to \beta\}$

c) $\{\alpha \to (\alpha \to \texttt{Int}) = (\texttt{Int} \to \beta) \to \gamma\}$

A8-2 ***Typherleitung***

a) Erstellen Sie eine Typherleitung in Baum-Notation für folgendes Typurteil:

$$\{\,\} \vdash (\backslash x \texttt{ -> } (\backslash y \texttt{ -> } (\backslash z \texttt{ -> } (x\ z)\ y))) :: (\alpha \to \beta \to \delta) \to \beta \to \alpha \to \delta$$

Zusatzfrage: Welchen Namen hat diese Funktion in der Standardbibliothek von Haskell?

b) Erstellen Sie eine Typherleitung in linearer Notation für folgendes Typurteil:

$$\{x{::}\texttt{Bool}, y{::}\texttt{Double} \to \texttt{Int}\} \vdash (\backslash z \texttt{ -> } z\ x\ 4\ (y\ 7)) :: (\texttt{Bool} \to \texttt{Int} \to \texttt{Int} \to \beta) \to \beta$$

Hinweis: Beachten Sie dabei die übliche Klammerkonventionen für Funktionstypen und Funktionsanwendung!

c) Beweisen Sie durch eine saubere Typherleitung in der Notation Ihrer Wahl, dass die folgende Haskell Funktion den behaupteten Typ hat:

```
twice :: (a -> a) -> a -> a
twice f x = f (f x)
```

Hinweis: Da das in der Vorlesung behandelte Typsystem nicht mit benannten Funktionen umgehen kann, müssen wir zuerst den Funktionsrumpf in eine äquivalenten Term übersetzen, welcher eine anonyme Funktionsdefinition mit Lambda benutzt.

A8-3 ***Paare I*** Wir erweitern unser Haskell-Fragment nun um Paar-Bildung und das Pattern-Matching von Paaren. Dafür benötigen wir zwei neuen Programmausdrücke:

e	::=	x	Variable
	|	c	Konstante
	|	$e_1\ e_2$	Anwendung
	|	$\lambda x.e$	Funktionsabstraktion
	|	(e_1, e_2)	Paar-Introduktion
	|	`let` (x, y) `=` e_1 `in` e_2	Paar-Elimination

Weiterhin benötigen wir zwei neue Typregeln:

$$\frac{\Gamma \vdash e_1 :: A \qquad \Gamma \vdash e_2 :: B}{\Gamma \vdash (e_1, e_2) :: (A, B)}\ (\textsc{Pair-I}) \qquad \frac{\Gamma \vdash e_1 :: (A, B) \qquad \Gamma, x{::}A, y{::}B \vdash e_2 :: C}{\Gamma \vdash \texttt{let}\ (x, y)\ \texttt{=}\ e_1\ \texttt{in}\ e_2 :: C}\ (\textsc{Pair-E})$$

Diskutieren Sie diese Typregeln! Geben Sie dann für jedes folgende Typurteil eine Herleitung an, falls möglich. Ansonsten begründen Sie, warum nicht möglich ist!

Hinweis: Eine Typinferenz mit Unifikation ist hier nicht gefragt.

a) $\{u{::}\texttt{Bool}, y{::}\texttt{Int}, z{::}\texttt{Bool}\} \vdash \backslash x\ \texttt{->}\ (x\ (y, z)) :: ((\texttt{Bool}, \texttt{Int}) \to \texttt{Int}) \to \texttt{Int}$

b) $\{x{::}(\alpha, \beta), f{::}\beta \to \gamma\} \vdash \texttt{let}\ (z, y)\ \texttt{=}\ x\ \texttt{in}\ (z, f\ y) :: (\alpha, \gamma)$

c) $\{f{::}\texttt{Bool} \to \texttt{Double}\} \vdash \texttt{let}\ (x, y)\ \texttt{=}\ z\ \texttt{in}\ (\backslash z\ \texttt{->}\ f\ x) :: (\texttt{Bool}, \texttt{Int}) \to \texttt{Double}$

H8-1 ***Der richtige Kontext*** **(0 Punkte)** (Abgabeformat: Text oder PDF)
Berechnen Sie mit Papier & Bleistift für jedes der folgenden Typisierungsurteile einen konkreten *minimalen* (!) Kontext Γ, so dass das entsprechende Typurteil wahr wird.

Beispiel: Das Typurteil $\Gamma \vdash x + 1.0 :: \texttt{Double}$ ist z.B. für den Kontext $\Gamma = \{x{::}\texttt{Double}\}$ wahr (welcher auch minimal ist, d.h. keine unnötigen Variablen enthält), nicht aber jedoch für den Kontext $\Gamma = \{y{::}\texttt{Double}, x{::}\texttt{Int}\}$.

a) $\Gamma_a \vdash$ `if x then "Akzeptiert!" else f y` $::$ `String`

b) $\Gamma_b \vdash$ `\y -> \z -> x z (y z)` $::\ (\alpha \to \beta) \to \alpha \to \gamma$

c) $\Gamma_c \vdash$ `\x -> if x then \y -> y x else \z -> 1.0` $::$ `Bool` $\to$ (`Bool` $\to$ `Double`) $\to$ `Double`

GHCI's Typinferenz kann diese Aufgaben auch lösen. *Nachdem* Sie die Aufgabe von Hand berechnet haben können Sie Ihr Ergebnis mit GHCI überprüfen. Was müssen Sie dazu jedoch mit dem Programmausdruck vorher noch tun?

H8-2 ***Paare 2*** **(4 Punkte)** (Abgabeformat: Text oder PDF)
Fortsetzung von Aufgabe A8-3. Geben Sie erneut unter Verwendung der in Aufgabe A8-3 angegebenen Typregeln eine Herleitung für jedes der folgenden Typurteile an, falls möglich. Falls es nicht möglich ist, so begründen Sie warum nicht!

d) $\{f{::}(\alpha, \beta) \to \delta, g{::}\delta \to \gamma, x{::}\gamma \to \alpha\} \vdash \backslash w\ \texttt{->}\ g\ (f\ w) :: (\alpha, \beta) \to \gamma$

e) $\{g{::}\alpha \to \gamma\} \vdash \backslash x\ \texttt{->}\ (\backslash f\ \texttt{->}\ f\ (g\ x)) :: (\beta, \gamma) \to (\gamma \to \delta) \to \delta$

f) $\{\,\} \vdash \backslash x\ \texttt{->}\ (\backslash y\ \texttt{->}\ x\ y\ y) :: (\alpha \to \alpha) \to \alpha \to \alpha$

H8-3 ***Unifikation II*** **(4 Punkte)** (`.hs`-Datei als Lösung abgeben)
Implementieren Sie Robinson's Unfikationsalgorithmus in Haskell! Verwenden Sie zur Bearbeitung die auf der Vorlesungshomepage bereitgestellte Vorlage. Bearbeiten Sie nur die mit `-- TODO` markierte Stelle! Sie dürfen auch neue Hilfsfunktion definieren, und alle vorhandenen Funktion nutzen; Sie dürfen aber nicht vorhandene Definitionen, Hilfsfunktion oder importierten Module abändern! Die Vorlage enthält folgende Deklarationen:

```
data TTyp = TInt | TBool | TArrow TTyp TTyp | TVar Identifier
type Identifier = String
type TEq  = (TTyp,TTyp)
```

Ein Typ ist also entweder ein Integer; ein Bool; ein Funktionstyp, welcher aus je einem Typ für das Argument und das Ergebnis besteht; oder eine Typvariable, welche wir einfach als String kodieren. Nicht verwechseln: `TVar "a" :: TTyp` kodiert einen Typen, welcher nur aus einer Typvariablen besteht; `"a" :: Identifier` kodiert die Typvariable selbst. Eine Typgleichung modellieren wir ganz einfach als ein Paar von Typen.

Sie müssen die Funktion `unify :: [TEq] -> TSub` implementieren, welche den allgemeinsten Unifikator für eine Liste von Funktionsgleichungen berechnet, falls möglich. Die Angabe enthält bereits Testfälle zum Testen Ihrer Implementation:

```
> aufg0                                 |  > unify aufg0
[(a,Bool),(b,Int),(a,c)]                |  [Bool/a, Int/b, Bool/c]
                                        |
> aufg1                                 |  > unify aufg1
[(a -> b,Bool -> b)]                    |  [Bool/a]
                                        |
> aufg5                                 |  > unify aufg5
[(a -> a -> Bool,(Bool -> b) -> c)]     |  [Bool -> b/a, (Bool -> b) -> Bool/c]
```

Folgende bereitgestellte Hilfsfunktionen dürfen Sie verwenden, wenn Sie möchten:

```
hasTVar :: Identifier -> TTyp -> Bool   -- prüft ob Typvariable in Typ enthalten ist
                -- Typsubstitution --
idSub       :: TSub                       -- leere Substitution []
makeSub     :: (Identifier, TTyp) -> TSub -- eine Typvariable auf einen Typ abbilden
composeSubs :: TSub -> TSub -> TSub       -- Komposition 2 Subs. von links nach rechts
applySub    :: TSub -> TTyp -> TTyp       -- Substitution auf Typ anwenden
```

Typsubstitutionen werden durch den Typ `TSub` modelliert. Wie das funktioniert, müssen wir zur Lösung der Aufgabe nicht verstehen. Es reicht zu wissen, was die Hilfsfunktion machen.

Wen es jedoch interessiert, ist natürlich herzlich willkommen sich den Code anzuschauen und Tutoren/Veranstalter dazu zu befragen: `TSub` ist mithilfe von `Data.Map` implementiert.

Abgabe: Lösungen zu den Hausaufgaben können bis Dienstag, den 24.06.2014, 11:00 Uhr mit UniworX abgegeben werden.

Aufgrund des Klausurbonus müssen die Hausaufgaben von Ihnen alleine gelöst werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen.