Dr. Steffen Jost
Leah Neukirchen

Ludwig-Maximilians-Universität München
Institut für Informatik
19. Juni 2018

09. Übung zur Vorlesung

Programmierung und Modellierung

A9-1 ***Der richtige Kontext*** Berechnen Sie mit Papier & Bleistift für jedes der folgenden Typisierungsurteile einen möglichst *allgemeinen* und *minimalen* Kontext Γ, so dass das entsprechende Typurteil wahr wird. Typklassen ignorieren wir in dieser Aufgabe.

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 $\Gamma = \{y{::}\texttt{Double}, x{::}\texttt{Char}\}$.

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?

A9-2 ***Neue Typregeln I*** Wir erweitern das Haskell-Fragment von Folie 8.7 um Paare. Dazu führen wir einen neuen Typausdruck $A \times B$ (wer möchte, darf anstatt $A \times B$ auch (A, B) wie in Haskell schreiben) und zwei neue Programmausdrücke ein:

`(`e_1`,`e_2`)` `case` e_1 `of (`x`,`y`) ->` e_2

Entwickeln Sie für beide Programmausdrücke möglichst sinnvolle Typregeln! Welche Anforderungen müssen an die jeweiligen Teilausdrücke gestellt werden? Werden Variablen gebunden?

Als Hilfe geben wir hier bereits die Konklusion der jeweiligen Regel vor:

$$\frac{}{\Gamma \vdash \texttt{(}e_1\texttt{,}e_2\texttt{)} :: A \times B}\ (\textsc{Pair-I}) \qquad \frac{}{\Gamma \vdash \texttt{case } e_1 \texttt{ of (}x\texttt{,}y\texttt{) -> } e_2 :: C}\ (\textsc{Pair-E})$$

A9-3 ***Typinferenz*** Berechnen Sie jeweils mit Hilfe des Hindley-Milner-Damas Algorithmus den prinzipalen Typ für alle nachfolgenden Programmausdrücke. Identifizieren Sie zuerst alle ungebundenen Variablen und erstellen Sie einen Typkontext, der alle gebundenen Variablen eine frische Typvariable zuweist.

a) `\`x `->` $y\ x$

b) `plus` 42 69

HINWEIS: A9-4 ist wichtige Vorbereitung auf nächste Übung; ggf. alleine nacharbeiten!

A9-4 *Hello World* Kreieren Sie mit GHC (nicht GHCI) eine ausführbare Datei, welche nach dem Start eine Begrüßung ausgibt und den Benutzer dazu auffordert, in je eine Zeile nacheinander zuerst ein Lieblingstier und dann eine Lieblingseigenschaft einzugeben. Danach soll das Programm noch einen einzelnen String ausgeben, welcher zuerst die Lieblingseigenschaft und dann das Lieblingstier wiedergibt:

```
> ghc helloTier.hs
...
> ./helloTier
Hi! Gib bitte zuerst Dein Lieblingstier und dann
in die nächste Zeile Deine Lieblingseigenschaft ein:
Schildkröte
faule
Psst, willst Du faule Schildkröte kaufen?
```

Hinweis: Das Beispiel zeigt eine Linux-Konsole. Je nach Betriebssystem kann der Aufruf einer ausführbaren Datei abweichen. Die Benutzereingabe waren "Schildkröte" und "faule" in der drittletzten und vorletzten Zeile, der Rest wurde durch das Programm ausgegeben.

H9-1 *Typfehler* (2 Punkte; Abgabe: `H9-1.txt` oder `H9-1.pdf`)
Geben Sie eine Herleitung (Baum- oder lineare Notation) für jedes der folgenden Typurteile an, falls möglich. Anderfalls begründen Sie, warum eine Typherleitung nicht möglich ist.

a) $\{f{::}\alpha \to \texttt{Bool}, z{::}\beta\} \vdash \backslash y \texttt{ -> } \backslash x \texttt{ -> if } f\ x \texttt{ then } z \texttt{ else } y\ z :: (\beta \to \beta) \to \alpha \to \beta$

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

H9-2 *Typregel II* (2 Punkte; Abgabe: `H9-2.txt` oder `H9-2.pdf`)
Analog zu Aufgabe A9-2 möchten wir nun unser Haskell-Fragment um den Summentyp $A+B$ erweitern. In Haskell entspricht dies dem Typen `Either a b`, d.h. wer will darf anstatt $A+B$ auch (`Either` A B) schreiben. Wir führen drei neue Programmausdrücke ein:

$$\texttt{Left } e \qquad \texttt{Right } e \qquad \texttt{case } e_1 \texttt{ of } \{\texttt{Left } x \texttt{ -> } e_2\texttt{; Right } y \texttt{ -> } e_3\}$$

Entwickeln Sie für alle drei Programmausdrücke möglichst sinnvolle Typregeln!

Als Hilfe geben wir hier bereits die Konklusion der jeweiligen Regel vor:

$$\frac{}{\Gamma \vdash \texttt{Left } e :: A+B}\ (\textsc{Left-I}) \qquad \frac{}{\Gamma \vdash \texttt{Right } e :: A+B}\ (\textsc{Right-I})$$

$$\frac{}{\Gamma \vdash \texttt{case } e_1 \texttt{ of } \{\texttt{Left } x \texttt{ -> } e_2\texttt{; Right } y \texttt{ -> } e_3\} :: C}\ (\textsc{Either-E})$$

H9-3 ***Lambda-Terme Auswerten*** (2 Punkte; Datei `H9-3.hs` als Lösung abgeben)
Dies ist eine Fortsetzung von Aufgabe H8-1. Falls Sie H8-1 nicht gelöst haben, können Sie auf die Musterlösung der H8-1 zurückgreifen.

Implementieren Sie `eval :: Term -> Term` zum auswerten von Lambda-Termen! Die Auswertung im Lambda-Kalkül ist recht einfach, wie auf Folie 8.8 beschrieben: Alle Werte werten zu sich selbst aus, d.h. eine Variable wertet unverändert zu sich selbst aus, eine Konstante wertet unverändert zu sich selbst aus, und eine Abstraktion (also eine Funktion) wird ebenfalls komplett unverändert zurückgegeben.

Lediglich bei der Funktionsanwendung ist etwas zu tun: Zuerst wird die linke Seite ausgewertet. Liegt links eine Funktion vor, dann substituieren wir das Funktionsargument durch die unveränderte rechte Seite und setzen die Auswertung fort. Verwenden sie hierzu die Substitutionsfunktion `subst :: (Char, Term) -> Term -> Term` von Folie 8.9 bzw. H8-1.

Beispiel: In diesem Beispiel definieren wir zuerst ein paar nützliche Lambda-Terme (sind in der beiliegenden Dateivorlage enthalten) und zeigen am Schluss zwei Auswertungen:

```
> c0
Abs 'f' (Abs 'x' (Var 'x'))
> c1
Abs 'f' (Abs 'x' (App (Var 'f') (Var 'x')))
> cSUCC
Abs 'n' (Abs 'f' (Abs 'x' (App (Var 'f') (App (App (Var 'n') (Var 'f')) (Var 'x')))))

> cFALSE
Abs 'x' (Abs 'y' (Var 'y'))
> cTRUE
Abs 'x' (Abs 'y' (Var 'x'))
> cISNULL
Abs 'x' (App (App (Var 'x') (Abs 'x' (Abs 'x' (Abs 'y' (Var 'y')))))
                                                    (Abs 'x' (Abs 'y' (Var 'x'))))
> eval $ App cISNULL c0
Abs 'x' (Abs 'y' (Var 'x'))
> eval $ App cISNULL (App cSUCC c0)
Abs 'x' (Abs 'y' (Var 'y'))
```

Abgabe: Lösungen zu den Hausaufgaben können bis Samstag, den 30.6.18, mit UniWorX nur als `.zip` abgegeben werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen. Bis zu 4 Studierende können gemeinsam als Gruppe abgeben. Bitte beachten Sie auch die Hinweise zum Übungsbetrieb auf der Vorlesungshomepage (`www.tcs.ifi.lmu.de/lehre/ss-2018/promo/`).