Dr. Steffen Jost
Leah Neukirchen

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

Lösungsvorschlag zur 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`

LÖSUNGSVORSCHLAG: $\Gamma_a = \{x{::}\texttt{Bool},\ f{::}\alpha \to \texttt{String}, y{::}\alpha\}$

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

LÖSUNGSVORSCHLAG: $\Gamma_b = \{x{::}(\alpha \to \beta \to \gamma)\}$

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

LÖSUNGSVORSCHLAG: $\Gamma_c = \{\}$ – alle vorkommenden Variablen waren gebunden

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?

LÖSUNGSVORSCHLAG:

GHCI akzeptiert nur geschlossene Programmausdrücke, d.h. es dürfen keine freien Variablen vorkommen. Wir können aber einfach alle ungebunden Variablen am Anfang des Programmausdrucks mit einen Lambda binden (die Reihenfolge ist dabei egal).

Beispiel: Im Programmausdruck für Teilaufgabe a) haben drei ungebundene Variablen: `x f y`. Wir binden diese mit Lambda und fragen GHCI nach dem Typ des Ausdrucks:

```
> :type \x f y -> (if x then "Akzeptiert!" else f y)
\x f y -> (if x then "Akzeptiert!" else f y) :: Bool -> (t -> [Char]) -> t -> [Char]
```

Der Ausdruck ist also eine Funktion, welche zuerst ein Argument des Typs `Bool` verlangt. Da wir als erste Variable `x` gebunden hatten, ist dies der Typ für `x`. Der Typ für `f` entspricht dem zweiten Argument, also $\alpha \to$ `String`. Der Typ für `y` ist das dritte Argument, welches wieder die gleiche Typvariable ist, welche schon im Typ für `f` verwendet wurde, also α (GHCI nutzt Kleinbuchstaben für Typvariablen, aber wir benutzen griechische Kleinbuchstaben für Typvariablen zu besseren Unterscheidung).

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:

$$\texttt{(}e_1\texttt{,}e_2\texttt{)} \qquad\qquad \texttt{case } e_1 \texttt{ of (}x\texttt{,}y\texttt{) -> } e_2$$

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:

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

LÖSUNGSVORSCHLAG:

Pair-I: Hier werden keine Variablen gebunden, der Typkontext bleibt entsprechend unverändert. Damit ein Paar den Typ $A \times B$ hat, müssen wir lediglich fordern, dass der erste Teilausdruck des Paares den Typ A hat und der zweite Teilausdruck den Typ B.

Das I im Namen steht für Intro, da durch diese Regel Paar-Typen eingeführt werden können.

Pair-E: Zuerst fordern wir, dass der Teilausdruck für den Mustervergleich ein Paar-Typ hat.

Durch den Mustervergleich werden zwei frische Variablen gebunden, hier x und y. Diese dürfen wir als zusätzliche Annahmen mit passendem Typ dem Typkontext zur Typisierung von e_2 hinzufügen.

Das E im Namen steht für Elimination, da durch diese Regel Paar-Typen eliminiert werden können (tritt in der Konklusion nicht mehr auf).

LÖSUNGSVORSCHLAG:
Auch wenn es nicht explizit gefragt war, so ist es eine gute Übung, die vollständige Typherleitung einmal hinzuschreiben:

a)

$$\dfrac{\dfrac{}{\Gamma_a \vdash \texttt{x} :: \texttt{bool}}\,(\textsc{Var}) \qquad \dfrac{}{\Gamma_a \vdash \texttt{"Akzeptiert!"} :: \texttt{string}}\,(\textsc{Const}) \qquad \dfrac{\dfrac{}{\Gamma_a \vdash \texttt{f} :: \alpha \to \texttt{string}}\,(\textsc{Var}) \qquad \dfrac{}{\Gamma_a \vdash \texttt{y} :: \alpha}\,(\textsc{Var})}{\Gamma_a \vdash \texttt{f y} :: \texttt{string}}}{\Gamma_a \vdash \texttt{if x then "Akzeptiert!" else f y} :: \texttt{string}}\,(\textsc{Cond})$$

b)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_b, \texttt{y} : \alpha \to \beta, \texttt{z} : \alpha \vdash \texttt{x} :: \alpha \to \beta \to \gamma}\,(\textsc{Var}) \qquad \dfrac{}{\Gamma_b, \texttt{y} : \alpha \to \beta, \texttt{z} : \alpha \vdash \texttt{z} :: \alpha}\,(\textsc{Var})}{\Gamma_b, \texttt{y} : \alpha \to \beta, \texttt{z} : \alpha \vdash \texttt{x z} :: \beta \to \gamma}\,(\textsc{App}) \qquad \dfrac{\dfrac{}{\Gamma_b, \texttt{y} : \alpha \to \beta, \texttt{z} : \alpha \vdash \texttt{y} :: \alpha \to \beta}\,(\textsc{Var}) \qquad \dfrac{}{\Gamma_b, \texttt{y} : \alpha \to \beta, \texttt{z} : \alpha \vdash \texttt{z} :: \alpha}\,(\textsc{Var})}{\Gamma_b, \texttt{y} : \alpha \to \beta, \texttt{z} : \alpha \vdash \texttt{y z} :: \beta}\,(\textsc{App})}{\Gamma_b, \texttt{y} : \alpha \to \beta, \texttt{z} : \alpha \vdash \texttt{x z (y z)} :: \gamma}\,(\textsc{App})}{\Gamma_b, \texttt{y} : \alpha \to \beta \vdash \texttt{\textbackslash z -> x z (y z)} :: \alpha \to \gamma}\,(\textsc{Abs})}{\Gamma_b \vdash \texttt{\textbackslash y -> \textbackslash z -> x z (y z)} :: (\alpha \to \beta) \to \alpha \to \gamma}\,(\textsc{Abs})$$

c)

$$\dfrac{\dfrac{}{\Gamma_c, \texttt{x} : \texttt{Bool} \vdash \texttt{x} :: \texttt{Bool}}\,(\textsc{Var}) \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_c, \texttt{x} : \texttt{Bool}, \texttt{y} : \texttt{Bool} \to \texttt{Double} \vdash \texttt{y} :: \texttt{Bool} \to \texttt{Double}}\,(\textsc{Var}) \qquad \dfrac{}{\Gamma_c, \texttt{x} : \texttt{Bool}, \texttt{y} : \texttt{Bool} \to \texttt{Double} \vdash \texttt{x} :: \texttt{Bool}}\,(\textsc{Var})}{\Gamma_c, \texttt{x} : \texttt{Bool}, \texttt{y} : \texttt{Bool} \to \texttt{Double} \vdash \texttt{y x} :: \texttt{Double}}\,(\textsc{App})}{\Gamma_c, \texttt{x} : \texttt{Bool} \vdash \texttt{\textbackslash y -> y x} :: (\texttt{Bool} \to \texttt{Double}) \to \texttt{Double}}\,(\textsc{Abs}) \qquad \dfrac{\dfrac{}{\Gamma_c, \texttt{x} : \texttt{Bool}, \texttt{z} : \texttt{Bool} \to \texttt{Double} \vdash \texttt{1.0} :: \texttt{Double}}\,(\textsc{Const})}{\Gamma_c, \texttt{x} : \texttt{Bool} \vdash \texttt{\textbackslash z -> 1.0} :: (\texttt{Bool} \to \texttt{Double}) \to \texttt{Double}}\,(\textsc{Abs})}{\Gamma_c, \texttt{x} : \texttt{Bool} \vdash \texttt{if x then \textbackslash y -> y x else \textbackslash z -> 1.0} :: (\texttt{Bool} \to \texttt{Double}) \to \texttt{Double}}\,(\textsc{Cond})$$

$$\Gamma_c \vdash \texttt{\textbackslash x -> if x then \textbackslash y -> y x else \textbackslash z -> 1.0} :: \texttt{Bool} \to (\texttt{Bool} \to \texttt{Double}) \to \texttt{Double}$$

Abbildung 1: Herleitungen zur A9-1 (waren nicht gefragt)

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) $\backslash x \texttt{ -> } y\ x$

LÖSUNGSVORSCHLAG:

$$
\begin{aligned}
&\mathsf{infer}_{\{y::\alpha\}}(\backslash x \texttt{ -> } y\ x) = (\beta\sigma_0 \to B, \sigma_0) && = (\beta \to \gamma, [\beta \to \gamma/\alpha]) && (1)\\
&\mathsf{infer}_{\{y::\alpha, x::\beta\}}(y\ x) = (B, \sigma_0) && = (\gamma\sigma_3, \sigma_1\sigma_2\sigma_3) && (2)\\
&\mathsf{infer}_{\{y::\alpha, x::\beta\}}(y) = (C, \sigma_1) && = (\alpha, \mathsf{id}) && (3)\\
&\mathsf{infer}_{\{y::\alpha, x::\beta\}}(x) = (A, \sigma_2) && = (\beta, \mathsf{id}) && (4)\\
&\sigma_3 = \mathsf{unify}\{C\sigma_2 = A \to \gamma\} = \mathsf{unify}\{\alpha = \beta \to \gamma\} && = [\beta \to \gamma/\alpha] && (5)
\end{aligned}
$$

In der linken Spalte schreiben wir auf, was wir haben; in der rechten Spalte schreiben wir dann das Ergebnis auf.

Im ersten Schritt (1) sind wir in Fall 3 des Algorithmus und wählen als frische Variable β. Danach landen wir bei (2) in Fall 4. Zuerst müssen wir die beiden rekursiven Fälle (3) und (4) jeweils durch Fall 1 abhandeln. Daraus folgt $\sigma_1 = \mathsf{id}$ und $\sigma_2 = \mathsf{id}$, so wie $C = \alpha$ und $A = \beta$. Jetzt können wir den Fall 4 aus (2) durch die geforderte Unifikation komplettieren, welche uns $\sigma_3 = [\beta \to \gamma/\alpha]$. Damit kommen wir wieder zurück zu Fall 3 in (1), wo wir nur einsetzen müssen: Es gilt also $\sigma_0 = [\beta \to \gamma/\alpha]$ und $B = \gamma\sigma_3 = \gamma$.

Damit haben wir:

$$\{y::\beta \to \gamma\} \vdash \backslash x \texttt{ -> } y\ x :: \beta \to \gamma$$

b) `plus 42 69`

LÖSUNGSVORSCHLAG:

Es kommt eine freie Variable `plus` vor, welch wir den Typ α geben. Weiterhin ist es nützlich, zu Beginn die impliziten Klammern explizit hinzuschreiben, um Flüchtigkeitsfehler zu vermeiden.

$$\mathsf{infer}_{\{\texttt{plus}::\alpha\}}((\texttt{plus}\ 42)\ 69) = (\beta\sigma_3, \sigma_1\sigma_2\sigma_3) \tag{6}$$

$$\mathsf{infer}_{\{\texttt{plus}::\alpha\}}(\texttt{plus}\ 42) = (C, \sigma_1) \quad = \text{siehe (10)} \tag{7}$$

$$\mathsf{infer}_{\{\texttt{plus}::\alpha\sigma_1\}}(69) = (A, \sigma_2) \quad = (\texttt{Int}, \mathsf{id}) \tag{8}$$

$$\sigma_3 = \mathsf{unify}\{C\sigma_2 = A \to \beta\} = \mathsf{unify}\{\gamma = \texttt{Int} \to \beta\} \quad = [\texttt{Int} \to \beta/\gamma] \tag{9}$$

$$\mathsf{infer}_{\{\texttt{plus}::\alpha\sigma_1\}}(\texttt{plus}\ 42) = (\gamma\sigma_6, \sigma_4\sigma_5\sigma_6) \quad = (\gamma, [\texttt{Int} \to \gamma/\alpha]) \tag{10}$$

$$\mathsf{infer}_{\{\texttt{plus}::\alpha\sigma_1\}}(\texttt{plus}) = (E, \sigma_4) \quad = (\alpha, \mathsf{id}) \tag{11}$$

$$\mathsf{infer}_{\{\texttt{plus}::\alpha\sigma_1\}}(42) = (D, \sigma_5) \quad = (\texttt{Int}, \mathsf{id}) \tag{12}$$

$$\sigma_6 = \mathsf{unify}\{E\sigma_5 = D \to \gamma\} = \mathsf{unify}\{\alpha = \texttt{Int} \to \gamma\} \quad = [\texttt{Int} \to \gamma/\alpha] \tag{13}$$

Wir haben $\sigma_1 = [\texttt{Int} \to \gamma/\alpha]$, $\sigma_2 = \mathsf{id}$, und $\sigma_3 = [\texttt{Int} \to \beta/\gamma]$. Damit also $\sigma_1\sigma_2\sigma_3 = [\texttt{Int} \to (\texttt{Int} \to \beta)/\alpha, \texttt{Int} \to \beta/\gamma]$. Das bedeutet: Wenn die Variable `plus` den Typ $\texttt{Int} \to \texttt{Int} \to \beta$ hat, dann hat der Ausdruck `plus 42 69` den Typ β.

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.

LÖSUNGSVORSCHLAG:

```
main = do
  putStrLn "Hi! Gib bitte zuerst Dein Lieblingstier und dann"
  putStrLn "in die nächste Zeile Deine Lieblingseigenschagt ein: "
  tier        <- getLine
  eigenschaft <- getLine
  let ausgabe = "Psst, willst Du " ++ eigenschaft ++ " " ++ tier ++ " kaufen?"
  putStrLn ausgabe
```

Es ist hier unerheblich, ob der Eingangstext in einer Zeile oder in zwei verschiedenen ausgegeben wird. Es ist auch nicht von Bedeutung, ob die Ausgabe erst mit einem let-Ausdruck unter einem lokalen Bezeichner abgelegt wird oder direkt als Argument an `putStrLn` übergeben wird.

Ende der Lösungsvorschläge für die Präsenzaufgaben.
Lösungsvorschläge für die Hausaufgaben folgen nach Ende der Abgabezeit.

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

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:

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

Entwickeln Sie für alle drei Programmausdrücke möglichst sinnvolle Typregeln!

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

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

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

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/`).