Prof. Martin Hofmann, PhD
Dr. Steffen Jost

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

8. Musterlösung 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) $\{\texttt{Int} \to (\alpha \to \beta) = \alpha \to (\beta \to \texttt{Int})\}$ $[\texttt{Int}/\alpha, \texttt{Int}/\beta]$

b) $\{\alpha \to (\alpha \to \texttt{Int}) = (\texttt{Int} \to \beta) \to \beta\}$ Typfehler: Zirkulärer Typ $(\texttt{Int} \to \beta) \to \texttt{Int} = \beta$

c) $\{\alpha \to (\alpha \to \texttt{Int}) = (\texttt{Int} \to \beta) \to \gamma\}$ $[(\texttt{Int} \to \beta)/\alpha, ((\texttt{Int} \to \beta) \to \texttt{Int})/\gamma]$

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?

LÖSUNGSVORSCHLAG:

$$\dfrac{\dfrac{\dfrac{\dfrac{x{::}\alpha \to \beta \to \delta \in \Delta}{\Delta \vdash x :: \alpha \to (\beta \to \delta)}\textsc{Var} \qquad \dfrac{z{::}\alpha \in \Delta}{\Delta \vdash z :: \alpha}\textsc{Var}}{\Delta \vdash x\ z :: \beta \to \delta}\textsc{App} \qquad \dfrac{y{::}\beta \in \{x{::}\alpha \to \beta \to \delta, y{::}\beta, z{::}\alpha\}}{\{x{::}\alpha \to \beta \to \delta, y{::}\beta, z{::}\alpha\} \vdash y :: \beta}\textsc{Var}}{\{x{::}\alpha \to \beta \to \delta, y{::}\beta, z{::}\alpha\} \vdash (x\ z)\ y :: \delta}\textsc{App}}{\{x{::}\alpha \to \beta \to \delta, y{::}\beta\} \vdash \backslash z \texttt{ -> } (x\ z)\ y :: \alpha \to \delta}\textsc{Abs}}{\{x{::}\alpha \to \beta \to \delta\} \vdash \backslash y \texttt{ -> } (\backslash z \texttt{ -> } (x\ z)\ y) :: \beta \to \alpha \to \delta}\textsc{Abs}}{\{\,\} \vdash \backslash x \texttt{ -> } (\backslash y \texttt{ -> } (\backslash z \texttt{ -> } (x\ z)\ y)) :: (\alpha \to \beta \to \delta) \to \beta \to \alpha \to \delta}\textsc{Abs}$$

wobei wir aus Platzgründen die Abkürzung $\Delta := \{x{::}\alpha \to \beta \to \delta, y{::}\beta, z{::}\alpha\}$ für einen Typkontext definieren und im oberen linken Teilbaum verwenden.

Der Programmausdruck beschreibt die Funktion `flip`.

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!

LÖSUNGSVORSCHLAG:

Aus Platzgründen definieren wir

$$\Gamma := \{x{::}\texttt{Bool}, y{::}\texttt{Double} \to \texttt{Int}, z{::}\texttt{Bool} \to (\texttt{Int} \to (\texttt{Int} \to \beta))\}$$

in der folgenden Typherleitung im linearen Stil – natürlich können wir diese Abkürzung erst definieren, wenn wir später während der Konstruktion der Typherleitung wissen was wir wirklich brauchen, d.h. bei einer Herleitung mit Papier & Bleistift müssten die Abkürzung unten auf dem Blatt niederschreiben (oder anfangs oben etwas Platz lassen). Wir beginnen damit, in dem wir Programmausdruck und Typausdruck vollständig klammern – das ist nicht notwendig, macht aber klarer was zu tun ist.

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

(1)	$:: (\texttt{Bool} \to (\texttt{Int} \to (\texttt{Int} \to \beta))) \to \beta$	Abs(2)
(2)	$\Gamma \vdash ((z\ x)\ 4)\ (y\ 7) :: \beta$	App(3, 4)
(3)	$\Gamma \vdash (z\ x)\ 4 :: \texttt{Int} \to \beta$	App(5, 6)
(4)	$\Gamma \vdash y\ 7 :: \texttt{Int}$	App(9, 10)
(5)	$\Gamma \vdash z\ x :: \texttt{Int} \to (\texttt{Int} \to \beta)$	App(7, 8)
(6)	$\Gamma \vdash 4 :: \texttt{Int}$	Const
(7)	$\Gamma \vdash z :: \texttt{Bool} \to (\texttt{Int} \to (\texttt{Int} \to \beta))$	Var
(8)	$\Gamma \vdash x :: \texttt{Bool}$	Var
(9)	$\Gamma \vdash y :: \texttt{Double} \to \texttt{Int}$	Var
(10)	$\Gamma \vdash 7 :: \texttt{Double}$	Cons

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.

LÖSUNGSVORSCHLAG:

Wir müssen den Funktionsrumpf in einen geschlossenen Programmausdruck verwandeln. Die beiden Argumente der Funktion `twice` müssen also mit Lambdas gebunden werden. Dies können wir auch in Haskell so hinschreiben:

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

Damit können wir nun ganz normal unsere Typherleitung basteln. Wir wählen hier die Baum-Notation, da diese sich nach unserer Ansicht vielleicht etwas schwieriger hinschreiben läßt, dafür aber leichter zu lesen ist, so fern der Baum auf eine Seite passt.

$$\dfrac{\dfrac{\dfrac{f \in \{f{::}\alpha \to \alpha, x{::}\alpha\}}{\{f{::}\alpha \to \alpha, x{::}\alpha\} \vdash f :: \alpha \to \alpha}\,\textsc{Var} \quad \dfrac{\dfrac{f \in \{f{::}\alpha \to \alpha, x{::}\alpha\}}{\{f{::}\alpha \to \alpha, x{::}\alpha\} \vdash f :: \alpha \to \alpha}\,\textsc{Var} \quad \dfrac{x \in \{f{::}\alpha \to \alpha, x{::}\alpha\}}{\{f{::}\alpha \to \alpha, x{::}\alpha\} \vdash x :: \alpha}\,\textsc{Var}}{\{f{::}\alpha \to \alpha, x{::}\alpha\} \vdash f\ x :: \alpha}\,\textsc{App}}{\{f{::}\alpha \to \alpha, x{::}\alpha\} \vdash f\ (f\ x) :: \alpha}\,\textsc{App}}{\{f{::}\alpha \to \alpha\} \vdash \backslash x\ \texttt{->}\ f\ (f\ x) :: \alpha \to \alpha}\,\textsc{Abs}}{\{\} \vdash \backslash f\ \texttt{->}\ \backslash x\ \texttt{->}\ f\ (f\ x) :: (\alpha \to \alpha) \to (\alpha \to \alpha)}\,\textsc{Abs}$$

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
	|	$\texttt{let}\ (x, y)\ \texttt{=}\ e_1\ \texttt{in}\ e_2$	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) $\{u{::}\texttt{Bool}, y{::}\texttt{Int}, z{::}\texttt{Bool}\} \vdash \backslash x\ \texttt{->}\ (x\ (y, z)) :: ((\texttt{Bool}, \texttt{Int}) \to \texttt{Int}) \to \texttt{Int}$

LÖSUNGSVORSCHLAG:

Zur Verkürzung der Schreibweise definieren wir $\Gamma := \{u{::}\texttt{Bool}, y{::}\texttt{Int}, z{::}\texttt{Bool}\}$.

(11)	$\Gamma \vdash \backslash x\ \texttt{->}\ (x\ (y, z)) :: ((\texttt{Bool}, \texttt{Int}) \to \texttt{Int}) \to \texttt{Int}$	Abs(12)
(12)	$\Gamma, \{x{::}(\texttt{Bool}, \texttt{Int}) \to \texttt{Int}\} \vdash x\ (y, z) :: \texttt{Int}$	App(13, 14)
(13)	$\Gamma, \{x{::}(\texttt{Bool}, \texttt{Int}) \to \texttt{Int}\} \vdash x :: (\texttt{Bool}, \texttt{Int}) \to \texttt{Int}$	Var
(14)	$\Gamma, \{x{::}(\texttt{Bool}, \texttt{Int}) \to \texttt{Int}\} \vdash (y, z) :: (\texttt{Bool}, \texttt{Int})$	Pair-I(15, 16)
(15)	$\Gamma, \{x{::}(\texttt{Bool}, \texttt{Int}) \to \texttt{Int}\} \vdash y :: \texttt{Bool}$	$\lightning$
(16)	$\Gamma, \{x{::}(\texttt{Bool}, \texttt{Int}) \to \texttt{Int}\} \vdash z :: \texttt{Int}$	$\lightning$

Eine Typherleitung ist nicht möglich, da y im gegebenen Typkontext nicht den geforderten Typ `Bool` hat. Auch z hat nicht den geforderten Typ `Int` in Γ. Jedes einzelne dieser beiden Probleme verhindert eine Typherleitung. Insbesondere bei Verwendung einer Abkürzung für Typkontexte müssen wir also bei der Anwendung der Typregel Var ganz genau hinschauen, dass das geforderte Typurteil auch wirklich ein Teil des Typkontexts ist!

Es bleibt anzumerken, dass eine Typherleitung im gegebenen Typkontext für den Programmausdruck $\backslash x\ \texttt{->}\ (x\ (z, y))$ problemlos möglich wäre.

Die Variable u kommt im Programmausdruck nicht vor. Das u im gegebenen Typkontext erwähnt wird, ist ohne weitere Bedeutung.

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

LÖSUNGSVORSCHLAG:

Hier ist eine Typherleitung ohne Probleme möglich:

(17)	$\{x{::}(\alpha,\beta), f{::}\beta \to \gamma\} \vdash \texttt{let}\ (z,y)\ \texttt{=}\ x\ \texttt{in}\ (z, f\ y) :: (\alpha,\gamma)$	PAIR-E(18, 19)
(18)	$\{x{::}(\alpha,\beta), f{::}\beta \to \gamma\} \vdash x :: (\alpha,\beta)$	VAR
(19)	$\{x{::}(\alpha,\beta), f{::}\beta \to \gamma, z{::}\alpha, y{::}\beta\} \vdash (z, f\ y) :: (\alpha,\gamma)$	PAIR-I(20, 21)
(20)	$\{x{::}(\alpha,\beta), f{::}\beta \to \gamma, z{::}\alpha, y{::}\beta\} \vdash z :: \alpha$	VAR
(21)	$\{x{::}(\alpha,\beta), f{::}\beta \to \gamma, z{::}\alpha, y{::}\beta\} \vdash f\ y :: \gamma$	APP(22, 23)
(22)	$\{x{::}(\alpha,\beta), f{::}\beta \to \gamma, z{::}\alpha, y{::}\beta\} \vdash f :: \beta \to \gamma$	VAR
(23)	$\{x{::}(\alpha,\beta), f{::}\beta \to \gamma, z{::}\alpha, y{::}\beta\} \vdash y :: \beta$	VAR

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

LÖSUNGSVORSCHLAG:

Hier klemmt es gleich im zweiten Schritt: Die Variable z kommt frei im Programmausdruck vor, es ist jedoch kein Typurteil im Typkontext für z enthalten, weshalb die Herleitung zwangsläufig scheitern muss:

(24)
$\{f{::}\texttt{Bool} \to \texttt{Double}\} \vdash \texttt{let } (x, y) \texttt{ = } z \texttt{ in } (\backslash z \texttt{ -> } f\ x) :: (\texttt{Bool}, \texttt{Int}) \to \texttt{Double}$ PAIR-E(25, 26)

(25)
$\{f{::}\texttt{Bool} \to \texttt{Double}\} \vdash z :: (\texttt{Bool}, \beta)$ $\lightning$

(26)
$\{f{::}\texttt{Bool} \to \texttt{Double}, x{::}\texttt{Bool}, y{::}\beta\} \vdash \backslash z' \texttt{ -> } f\ x :: (\texttt{Bool}, \texttt{Int}) \to \texttt{Double}$ ABS(27)

(27)
$\{f{::}\texttt{Bool} \to \texttt{Double}, x{::}\texttt{Bool}, y{::}\beta, z'{::}(\texttt{Bool}, \texttt{Int})\} \vdash f\ x :: \texttt{Double}$ APP(28, 29)

(28)
$\{f{::}\texttt{Bool} \to \texttt{Double}, x{::}\texttt{Bool}, y{::}\beta, z'{::}(\texttt{Bool}, \texttt{Int})\} \vdash f :: \texttt{Bool} \to \texttt{Double}$ VAR

(29)
$\{f{::}\texttt{Bool} \to \texttt{Double}, x{::}\texttt{Bool}, y{::}\beta, z'{::}(\texttt{Bool}, \texttt{Int})\} \vdash x :: \texttt{Bool}$ VAR

Der Programmausdruck enthält zwar einen Teilausdruck $(\backslash z \texttt{ -> } f\ x)$, doch deckt nur die Vorkommen von z im Rumpf dieser anonymen Funktion ab (hier keine Vorkommen). Es ist also wichtig zu verstehen, in welcher Reihenfolge der Typausdruck zu lesen ist! Im Zweifelsfall hilft es, gebundene Variablen mit frischen Namen zu belegen. Gebunden werden hier $x, y \mathit{undauch} z$. Wenn wir diese in frische Variablen umbenennen x', y' und z' erhalten wir:

$$\texttt{let } (x', y') \texttt{ = } z \texttt{ in } (\backslash z' \texttt{ -> } (f\ x)[x'/x, y'/y, z'/z]) = \texttt{let } (x', y') \texttt{ = } z \texttt{ in } (\backslash z' \texttt{ -> } f\ x')$$

Wie wir sehen, bleibt das z im ersten Teilausdruck erhalten, da es frei vorkommt.

Alle weiteren hier gelisteten Schritte 26–29 hätten wir uns gemäß der Aufgabenstellung sparen können. Wir haben die Schritte für die Musterlösung dennoch aufgeführt, da eine Herleitung möglich wäre, wenn der am Anfang gegebene Kontext $\{f{::}\texttt{Bool} \to \texttt{Double}, z{::}(\texttt{Bool}, \beta)\}$ gewesen wäre. Eine Umbenennung des Lambda-Ausdrucks wäre dann aber wie im vorangegangenen Paragraph für z' notwendig gewesen, da der Kontext jeder Variable nur einen Typ zuweisen darf (aber nicht notwendig für x' und y', da es hier keinen Konflikt gibt).

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`

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` $::$ `Bool` $\to$ (`Bool` $\to$ `Double`) $\to$ `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 einen Lambda (man könnte auch 3 Lambdas für jede einzelne Variable angeben, aber das macht ja keinen Unterschied), und fragen GHCI nach dem Typ des Ausdrucks:

```
> :type \x f y -> (if x then "Azeptiert!" else f y)
\x f y -> (if x then "Azeptiert!" 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).

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

a)

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

b)

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

c)

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

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

Abbildung 1: Herleitungen zur H8-1 (waren nicht gefragt)

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

LÖSUNGSVORSCHLAG:

Da die Variable x aus dem gegebenen Typkontext nicht frei im Programmausdruck vorkommt, entfernen wir dieses als erste durch Anwendung der strukturellen Typregel WEAK aus dem Typkontext, um uns etwas Schreibarbeit zu sparen. Dies ist jedoch ein optionaler Schritt.

(30) $\{f{::}(\alpha,\beta)\to\delta, g{::}\delta\to\gamma, f{::}\gamma\to\alpha\} \vdash \backslash w \texttt{ -> } g\,(f\ w) :: (\alpha,\beta)\to\gamma$ WEAK(31)

(31) $\{f{::}(\alpha,\beta)\to\delta, g{::}\delta\to\gamma\} \vdash \backslash w \texttt{ -> } g\ (f\ w) :: (\alpha,\beta)\to\gamma$ ABS(32)

(32) $\{f{::}(\alpha,\beta)\to\delta, g{::}\delta\to\gamma, w{::}(\alpha,\beta)\} \vdash g\ (f\ w) :: \gamma$ APP(33, 34)

(33) $\{f{::}(\alpha,\beta)\to\delta, g{::}\delta\to\gamma, w{::}(\alpha,\beta)\} \vdash g :: \delta\to\gamma$ VAR

(34) $\{f{::}(\alpha,\beta)\to\delta, g{::}\delta\to\gamma, w{::}(\alpha,\beta)\} \vdash f\ w :: \delta$ APP(35, 36)

(35) $\{f{::}(\alpha,\beta)\to\delta, g{::}\delta\to\gamma, w{::}(\alpha,\beta)\} \vdash f :: (\alpha,\beta)\to\delta$ VAR

(36) $\{f{::}(\alpha,\beta)\to\delta, g{::}\delta\to\gamma, w{::}(\alpha,\beta)\} \vdash w :: (\alpha,\beta)$ VAR

Auch wenn in dieser Typherleitung Paar-Typen vorkommen, so brauchen wir weder die Typregel PAIR-I noch PAIR-E, da wir die entsprechenden Werte in diesem Programmausdruck nur herumreichen, aber nicht auseinander nehmen oder neu konstruieren.

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

LÖSUNGSVORSCHLAG:

(37)	$\{g{::}\alpha \to \gamma\} \vdash \backslash x \texttt{ -> } (\backslash f \texttt{ -> } f\ (g\ x)) :: (\beta, \gamma) \to (\gamma \to \delta) \to \delta$	Abs(38)
(38)	$\{g{::}\alpha \to \gamma, x{::}(\beta, \gamma)\} \vdash \backslash f \texttt{ -> } f\ (g\ x) :: (\gamma \to \delta) \to \delta$	Abs(39)
(39)	$\{g{::}\alpha \to \gamma, x{::}(\beta, \gamma), f{::}\gamma \to \delta\} \vdash f\ (g\ x) :: \delta$	App(40, 41)
(40)	$\{g{::}\alpha \to \gamma, x{::}(\beta, \gamma), f{::}\gamma \to \delta\} \vdash f :: \gamma \to \delta$	Var
(41)	$\{g{::}\alpha \to \gamma, x{::}(\beta, \gamma), f{::}\gamma \to \delta\} \vdash g\ x :: \gamma$	$\lightning$ App(42, 43) $\lightning$
(42)	$\{g{::}\alpha \to \gamma, x{::}(\beta, \gamma), f{::}\gamma \to \delta\} \vdash g :: \alpha \to \gamma$	Var
(43)	$\{g{::}\alpha \to \gamma, x{::}(\beta, \gamma), f{::}\gamma \to \delta\} \vdash x :: (\alpha, \beta)$	Var

Eine Herleitung ist nicht möglich: Die Anwendung der Typregel App in (41) erfordert entweder $g{::}\alpha \to \gamma$ und $x{::}\alpha$ oder $g{::}(\beta, \gamma) \to \gamma$ und $x{::}(\beta, \gamma)$; aber eine Mischung daraus ist nicht erlaubt. Man könnte (41) also noch retten, aber dann geht entweder (42) oder (43) schief.

Die Aufgabenstellung forderte eine Typherleitung für die gegebenen Typen. Falls eine Typinferenz gefragt wäre, dann könnte man Unifikation für $\alpha = (\beta, \gamma)$ einsetzen und eine Herleitung unter dieser Substitution bekommen. Das war jedoch hier nicht gefordert (gibt aber auch kein Punktabzug).

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

LÖSUNGSVORSCHLAG:

(44)	$\{\,\} \vdash \backslash x \texttt{ -> } (\backslash y \texttt{ -> } (x\ y)\ y) :: (\alpha \to \alpha) \to \alpha \to \alpha$	Abs(45)
(45)	$\{x{::}\alpha \to \alpha\} \vdash \backslash y \texttt{ -> } (x\ y)\ y :: \alpha \to \alpha$	Abs(46)
(46)	$\{x{::}\alpha \to \alpha, y{::}\alpha\} \vdash (x\ y)\ y :: \alpha$	App(47, 48)
(47)	$\{x{::}\alpha \to \alpha, y{::}\alpha\} \vdash (x\ y) :: \alpha \to \alpha$	App(49, 50)
(48)	$\{x{::}\alpha \to \alpha, y{::}\alpha\} \vdash y :: \alpha$	Var
(49)	$\{x{::}\alpha \to \alpha, y{::}\alpha\} \vdash x :: \alpha \to (\alpha \to \alpha)$	$\lightning$
(50)	$\{x{::}\alpha \to \alpha, y{::}\alpha\} \vdash y :: \alpha$	Var

Die Typherleitung schlägt fehl, da x eine Funktion mit Stelligkeit 1 ist, aber auf zwei Argumente angewendet wird.

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.

LÖSUNGSVORSCHLAG:

```
import Data.List
import qualified Data.Map as Map


data TTyp = TVar Identifier
          | TInt | TBool
          | TArrow TTyp TTyp
  deriving (Eq) -- Show-Instanz weiter unten von Hand deklariert

type Identifier = String

type TEq  = (TTyp,TTyp) -- Eine Typgleichung

-----------------------------------------------

unify :: [TEq]  -> TSub
-- unify = -- TODO: implementieren Sie diese Funktion gemäß Vorlesungsfolie 08-S.48!
--   error "Teilnehmer hat unify leider noch nicht implementiert!"
unify []  = idSub
unify ((ty1,ty2):tl)
  | ty1 == ty2 = unify tl
unify ((TArrow ty1a ty1b, TArrow ty2a ty2b):tl)
              = unify $ (ty1a,ty2a):(ty1b,ty2b):tl
unify (h@(TVar v1,ty2):tl)
  | hasTVar v1 ty2 = error $ "unify: Zirkulär " ++ (show $ show h)
  | otherwise       =
      let sigma1  = makeSub (v1,ty2)
          sigma2  = unify $ map (applySub2Equation sigma1) tl
      in composeSubs sigma1 sigma2
unify ((ty1,ty2@(TVar _)):tl)
  = unify $ (ty2,ty1):tl
unify (h:_)
  = error $ "unify: Typfehler " ++ (show $ show h)

-----------------------------------------------
-- Hilfsfunktionen:

hasTVar :: Identifier -> TTyp -> Bool    -- Prüft ob Typvariable im Typ vorkommt
hasTVar var = has
  where -- Abkürzung "has", um nicht bei jedem rekursiven Aufruf jedes mal var übergeben zu müssen
    has :: TTyp -> Bool
    has (TVar v1)      = v1 == var
    has (TArrow t1 t2) = has t1 || has t2
    has _othertype     = False

--------------------------------
-- All About Typesubstitutions

newtype TSub = TSub (Map.Map Identifier TTyp) -- Eine Typsubstitution
  -- newtype, damit wir eigene Show-Instanz definieren können, ansonsten würde type reichen


idSub :: TSub
idSub = TSub $ Map.empty
```

```
makeSub :: (Identifier, TTyp) -> TSub
makeSub = TSub . uncurry Map.singleton

composeSubs :: TSub -> TSub -> TSub
composeSubs (TSub s1) (TSub s2) = TSub $ Map.union (Map.map (applySub $ TSub s2) s1) s2
-- proper merging of overlapping substitution
-- s1 is applied first, then s2 is applied

applySub :: TSub -> TTyp -> TTyp
applySub (TSub sigma) = applySigma
  where
    applySigma t1@(TVar v1)   = Map.findWithDefault t1 v1 sigma
    applySigma (TArrow t1 t2) = TArrow (applySigma t1) (applySigma t2)
    applySigma othertype      = othertype

applySub2Equation :: TSub -> TEq -> TEq
applySub2Equation sigma (t1,t2) = (s t1, s t2)
  where s = applySub sigma


-------------------------------------
-- Beispiele zum Testen, z.B. mit:
-- > unify aufg1
--
alpha, beta, gamma, delta :: TTyp
alpha = TVar "a"
beta  = TVar "b"
gamma = TVar "c"
delta = TVar "d"

aufg0, aufg1, aufg2, aufg3, aufg4, aufg5, aufg6 :: [TEq]

aufg0 = [(alpha,TBool),(beta,TInt),(alpha,gamma)]

aufg1 = [(alpha `TArrow` beta, TBool `TArrow` beta)]

aufg2 =  [(alpha,beta),(beta,gamma)]

aufg3 = [(t1,t2)]
 where
   t1 = TArrow TInt  $ TArrow alpha beta
   t2 = TArrow alpha $ TArrow beta  TInt

aufg4 = [(t1,t2)] -- zirkulär
 where
   t1 = alpha `TArrow` (alpha `TArrow` TInt)
   t2 = (TInt `TArrow` beta) `TArrow` beta

aufg5 =  [(t1,t2)]
 where
   t1 = alpha `TArrow` (alpha `TArrow` TBool)
   t2 = (TBool `TArrow` beta) `TArrow` gamma

aufg6 =  [(t1,t2)] -- Typfehler
 where
```

```
   t1 = TInt `TArrow` alpha
   t2 = alpha `TArrow` TBool

-----------------------------------------------
-- Anzeige von TTyp und TSub


instance Show TTyp where -- etwas hübschere Show-Instanz als die automatisch generierte
{--- Herkömmliche Variante:
  show (TVar v)       = v
  show (TInt)         = "Int"
  show (TBool)        = "Bool"
  show (TArrow t1 t2) = "(" ++ (show t1) ++ ") -> " ++ (show t2)
  --
  -- Mit intelligenter Klammerung:
-}
  showsPrec _ (TVar v)         = showString v
  showsPrec _ (TInt)           = showString "Int"
  showsPrec _ (TBool)          = showString "Bool"
  showsPrec n (t1 `TArrow` t2) = showParen (n>0) $ (showsPrec 1 t1) . showString " -> " . showsPrec 0 t2

instance Show TSub where
  show (TSub sigma) = '[':subs ++ "]"
    where subs = intercalate ", " $ map (\(v,t) -> show t ++ '/':v) $ Map.assocs sigma
```

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.