Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
16. Juni 2015

13. Übung zur Vorlesung

Programmierung und Modellierung

Probeklausur

Auf den folgenden Seiten finden Sie eine unveränderte Klausur aus einem vergangenem Semester. Die Bearbeitungszeit betrug 120 Minuten und ist damit gleich zur Bearbeitungszeit der Klausur zur "Programmierung und Modellierung" im aktuellen Semester.

Diese Probeklausur ist natürlich nur von beispielhafter Natur. Insbesondere kann eine Klausur immer nur eine Auswahl der behandelten Themen abfragen. Weiterhin weichen auch die behandelten Themen etwas ab: in diesem Jahr wurden die Typregeln für 2-Tuple nicht behandelt; dafür haben wir in diesem Jahr mehr zum Thema Denotationelle Semantik gemacht; anstelle der Zustandsmonade hatten wir in diesem Jahr die Par-Monade betrachtet; usw.

Auch zu diesem Übungsblatt wird eine Musterlösung per UniworX herausgegeben werden.

Organisatorische Hinweise

- Eine Klausuranmeldung per UniworX ist zur Teilnahme zwingend erforderlich. Die Anmeldefirst ist abgelaufen. Unentschuldigt fehlende Klausurteilnehmer werden ans Prüfungsamt gemeldet.
- Jeder Student muss einen gültigen Lichtbildausweis und Studentenausweis mitbringen.
- Es sind keine Hilfsmittel zugelassen. Am Platz darf sich nur Schreibzeug und eventuell ein paar Nahrungsmittel befinden.
- Verwenden Sie keinen Bleistift und keine Stifte in rot oder grün! Verwendung Sie nur dokumentenechte Stifte!
- Papier wird von uns gestellt und darf nicht mitgebracht werden.
- Taschen und Jacken müssen vorne an der Tafel abgelegt werden. Sollte jemand ein Telefon, mp3-Player, oder Ähnliches am Platz haben, ist das ein Täuschungsversuch, der dem Prüfungsausschuss gemeldet wird.
- Sollte ein Telefon klingeln, ist das eine Störung des Prüfungsablaufs und hat den Ausschluss von der weiteren Teilnahme zur Folge.
- Gehen Sie rechtzeitig vor Beginn in den zugewiesenen Raum! Die Raumeinteilung wird am Montag auf der Vorlesungshomepage bekanntgegeben.

Nachklausur Eine Nachklausur ist für Anfang Oktober geplant. Der Termin wird unverzüglich auf der ProMo-Homepage bekanntgegeben, so bald die Raumbuchung erfolgte. Die Anmeldung zur Nachklausur erfolgt ca. August per UniworX. Es wird nur eine Nachklausur angeboten werden; wer sich zur Nachklausur anmeldet, ohne vorher an der Erstklausur teilgenommen zu haben, kann die Prüfung erst wieder im Sommersemester 2016 versuchen.

Aufgabe 1 (Auswertung): **(6 Punkte)**

a) Rechnen Sie aus, zu welchem Wert die folgenden Haskell-Ausdrücke vollständig auswerten. Geben Sie nur das Endergebnis an, etwaige Nebenrechnungen bitte deutlich abtrennen:

(i) `(4 == 5.0:[]`

(ii) `[[z,y]| x<-[1..4], y<-[1,3..8], let z = x-1, x==y]`

(iii) `(\x y-> y) "yes" (\z -> "no") "which?"`

---

b) Werten Sie folgenden Ausdruck schrittweise vollständig aus, unterstreichen Sie dabei den reduzierten Redex. Um die volle Punktzahl zu erreichen,

Sie dabei dürfen eine beliebige Auswertestrategie verwenden. für volle Punktzahl müssen Sie aber Call-By-Value oder Call-By-Name verwenden.

Beispiel: $(\backslash x \text{ -> } 43 + x)$ $(\underline{\texttt{negate}\ 1}) \rightsquigarrow \underline{(\backslash x \text{ -> } 43 + x)\ (-1)} \rightsquigarrow \underline{43 + (-1)} \rightsquigarrow 42$

Definitionen:

```
const x y = x
negate x  = -x
```

Ausdruck zum Auswerten:
`const const (negate 1) (negate 2) 3` $\rightsquigarrow$

c) Welche Auswertestrategie haben Sie in Aufgabenteil b verwendet?

Aufgabe 2 (Induktion): **(5 Punkte)**

Wir betrachten folgende Funktionsdefinitionen:

```
length :: [Char] -> Int
length    []  = 0                              -- (LZ)
length (x:xs) = 1 + length xs                  -- (LS)

insert :: Char -> [Char] -> [Char]
insert x    [] = [x]                           -- (IN)
insert x (y:ys) | x <= y    = x : y : ys       -- (IK)
                | otherwise = y : insert x ys  -- (IG)
```

Beweisen Sie mit Induktion über die Länge der Liste, dass für alle Zeichen c und alle Strings s die folgende Gleichung gilt:

$$\texttt{length (insert c s)} \quad = \quad \texttt{1 + length s}$$

Hinweise: Formen Sie beide Seiten der geforderten Gleichung schrittweise in den exakt gleichen Term um. Begründen Sie jeden Umformungsschritt durch Angabe des Kürzels der verwendeten Gleichung, also `(LZ)`, `(LS)`, `(IN)`, `(IK)` oder `(IG)`, und `(IH)` bei Verwendung der Induktionshypothese! Beachten Sie die Pattern-Guards bei Verwendung von `(IK)` oder `(IG)` und begründen Sie kurz, warum der jeweilig Fall eintritt.

Aufgabe 3 (Abstiegsfunktion): **(5 Punkte)**

Wir wollen mithilfe einer geeigneten Abstiegsfunktion zeigen, dass die folgende rekursive Funktion `foo :: (Int,Int,Int) -> Int`, gegeben in Haskell Notation, immer terminiert:

```
foo (x, y, z)
  | x > 10, z > 0  = 2 * foo (x-1, y+3, z+2)
  | x >  0, z > 20 = 4 * foo (x+3, y-1, z `div` 2)
  | otherwise = y
```

a) Zeigen Sie, dass die Funktion $m'(x, y, z) = \max(2x + z, 0)$ keine geeignete Abstiegsfunktion für den Terminationsbeweis von `foo` ist.

b) Finden Sie eine Abstiegsfunktion m und beweisen, dass diese eine tatsächlich eine geeignete Abstiegsfunktion ist, also dass `foo` immer terminiert.
Hinweise: Auf Auf) und Def) verzichten wir hier. Weiterhin dürfen Sie folgende Abschätzung verwenden: Wenn $z > 20$ dann gilt auch $(z \ \texttt{`div`}\ 2) < z - 10$

Aufgabe 4 (Maybe): **(5 Punkte)**

Gegeben ist folgende Datentypdeklaration aus der Standardbibliothek:

```
data Maybe a = Nothing | Just a
```

a) Implementieren Sie die Funktion `fromMaybe :: a -> Maybe a -> a` zu Fuss, d.h. ohne Verwendung von Funktionen der Standardbibliothek.

Diese Funktion liefert das erste Argument zurück, falls das zweite Argument `Nothing` war; ansonsten wird der im zweiten Argument "verpackte´´ Wert zurückgegeben.

b) Implementieren Sie eine Funktion `andMaybe :: Maybe Bool -> Maybe Bool -> Maybe Bool` zu Fuss, d.h. ohne Verwendung von Funktionen der Standardbibliothek; lediglich die Funktion `&& :: Bool -> Bool -> Bool` dürfen Sie verwenden.

Der Typkonstruktor `Maybe` kann als Monade aufgefasst werden. Die volle Punktzahl erhalten Sie für diese Aufgabe nur dann, wenn Ihre Lösung die DO-Notation sinnvoll einsetzt. Die Verwendung von `return :: a -> m a` ist erlaubt.

Aufgabe 5 (Datenstrukturen): **(5 Punkte)**

Gegeben ist folgende Datentypdeklaration zur Repräsentation von Typen:

```
data TTyp = TVar Char | TInt | TListe TTyp | TTupel [TTyp]
  deriving (Eq, Show)
```

Schreiben Sie eine Funktion `rename :: (Char -> Char) -> TTyp -> TTyp` welche alle `Char`-Zeichen in einem Wert des Typs `TTyp` gemäß einer gegebenen Funktion umbenennt.

Beispiel:
Den Typ einer Liste von Paaren aus ganzen Zahlen und eines unbekannten Typs `a`, welchen wir in Haskell mit `[(Int,a)]` bezeichnen würden, könnten wir als Wert von `TTyp` durch den Ausdruck `TListe (TTupel [TInt,TVar 'a'])` darstellen.

```
> let t = TListe (TTupel [TVar 'a', TInt, TVar 'c'])
t :: TTyp
> let s = \c -> if c=='a' then 'b' else c
s :: Char -> Char
> rename s t
TListe (TTupel [TVar 'b', TInt, TVar 'c'])
```

Aufgabe 6 (Funktionen höherer Ordnung): (5 Punkte)

a) Implementieren Sie zu Fuss, d.h. ohne Verwendung von Funktionen aus der Standardbibliothek, die Infix-Funktion zur Funktionskomposition: `(.) :: (b -> c) -> (a -> b) -> a -> c`
Beispiel: `((+1).(*2)) 3 == 7`

b) Schreiben Sie eine Funktion `compose :: [a -> a] -> a -> a` welche eine Liste von Funktion von rechts-nach-links auf einen Startwert anwendet:
Beispiel: `compose [(+1),(*2)] 3 == 7`

Sie dürfen alle Funktionen aus der Standardbibliothek einsetzen. Um die volle Punktzahl zu erhalten, dürfen Sie keine direkte Rekursion verwenden! Verwenden Sie stattdessen Funktionen der Standardbibliothek, wie z.B. `foldr :: (a -> b -> b) -> b -> [a] -> b`

Aufgabe 7 (Monaden): **(5 Punkte)**

a) Schreiben Sie eine Funktion `main :: IO ()`, welche eine Zeile von der Tastatur einliest und anschließend rückwärts ausgibt und beendet, also bei Eingabe von `"evil"` wird `"live"` ausgegeben. Sie dürfen alle Funktionen der Standardbibliothek benutzen. DO-Notation ist erlaubt.

b) Implementieren Sie die Funktion `forM :: Monad m => [a] -> (a -> m b) -> m [b]` zu Fuss, d.h. ohne Verwendung von Funktionen der Standardbibliothek. DO-Notation ist erlaubt.

Aufgabe 8 (Typen): **(7 Punkte)**

a) Geben Sie jeweils den allgemeinsten Typ des gegebenen Haskell-Ausdrucks an, inklusive etwaiger Typklassen Einschränkungen. Bitte nur das Ergebnis hinschreiben. Nebenrechnungen bitte deutlich abtrennen.

(i) `fst ('7',(()))`

_____

(ii) `[False] : []`

_____

(iii) `(\f x y -> (f y) ++ show x)`

_____

b) Geben Sie denn allgemeinsten Unifikator für folgende Typgleichung an:

$\{\alpha \to \beta \to \texttt{Int} = \beta \to \gamma\}$ _____

c) Beweisen Sie folgendes Typurteil unter Verwendung der zur Erinnerung auf Seite 9 angegeben Typregeln in einer der beiden in der Vorlesung behandelten Notationen (Herleitungsbaum oder lineare Schreibweise).

(iv) $\Gamma \vdash \texttt{let}\ (x, y) = (f, \texttt{True})\ \texttt{in}\ x\ y :: \texttt{Int}$ wobei $\Gamma = \{\texttt{f::Bool -> Int}\}$

Typregeln:

$$\frac{}{\Gamma \vdash x :: \Gamma(x)} \quad \text{(Var)}$$

Alternative Schreibweise für Var:

$$\frac{x{::}A \in \Gamma}{\Gamma \vdash x :: A} \quad \text{(Var)}$$

$$\frac{c \text{ ist eine Integer Konstante}}{\Gamma \vdash c :: \texttt{Int}} \quad \text{(Int)}$$

$$\frac{c \in \{\texttt{True}, \texttt{False}\}}{\Gamma \vdash c :: \texttt{Bool}} \quad \text{(Bool)}$$

$$\frac{\Gamma \vdash e_1 :: A \to B \qquad \Gamma \vdash e_2 :: A}{\Gamma \vdash e_1\ e_2 :: B} \quad \text{(App)}$$

$$\frac{\Gamma, x{::}A \vdash e :: B}{\Gamma \vdash \backslash x \to e :: A \to B} \quad \text{(Abs)}$$

$$\frac{\Gamma \vdash e_1 :: A \qquad \Gamma \vdash e_2 :: B}{\Gamma \vdash (e_1, e_2) :: (A, B)} \quad \text{(Pair-Intro)}$$

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

$$\frac{\Gamma \vdash e_1{::}A \qquad \Gamma,\ x{::}A \vdash e_2 :: C}{\Gamma\ \vdash\ \texttt{let}\ x\ =\ e_1\ \texttt{in}\ e_2 :: C} \quad \text{(Let)}$$

$$\frac{\Gamma \vdash e_1 :: \texttt{Bool} \qquad \Gamma \vdash e_2 :: A \qquad \Gamma \vdash e_3 :: A}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 :: A} \quad \text{(Cond)}$$