Dr. Steffen Jost
Leah Neukirchen

Ludwig-Maximilians-Universität München
Institut für Informatik
10. Juli 2018

12. 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 haben wir das Thema "Denotationelle Semantik" nicht gemacht, dementsprechend ist Aufgabe 7 nicht klausurrelevant; dafür haben wir aber in diesem Jahr Monoide, applikative Funktoren und die Par-Monade betrachtet.

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 findet am Mittwoch 26.9. von 10-13 Uhr statt. 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 2019 versuchen.

Institut für Informatik der Ludwig-Maximilians-Universität München
Prof. Martin Hofmann, PhD
Dr. Steffen Jost

WiSe 2015/16
8.10.2015

Nachklausur zur Vorlesung
Programmierung und Modellierung

Die Bearbeitungszeit beträgt 120 min. Hilfsmittel sind nicht erlaubt, auch der Besitz ausgeschalteter elektronischer Geräte wird als Betrug gewertet. Schreiben Sie Ihren vollständigen Namen und Ihre Matrikelnummer deutlich lesbar auf dieses Deckblatt, sowie Ihren Namen in die Kopfzeile auf **jedem Blatt** der Klausurangabe! Geben Sie alle Blätter ab, lassen Sie diese immer zusammengeheftet! Verwenden Sie nur dokumentenechte Stifte und weder die Farben rot noch grün.

Kontrollieren Sie, ob Sie alle Aufgabenblätter erhalten haben. Aufgabenstellungen befinden sich auf den Seiten 1–13. Sie dürfen die Rückseiten für Nebenrechnungen nutzen. Falls Sie die Rückseiten für Antworten nutzen, so markieren Sie klar, was zu welcher Aufgabe gehört und schreiben Sie in der entsprechenden Aufgabe, wo alle Teile Ihrer Antwort zu finden sind. Streichen Sie alles durch, was nicht korrigiert werden soll!

Lesen Sie die Aufgabenstellungen vollständig durch, bevor Sie mit der Bearbeitung der Aufgaben beginnen. Es gibt 8 unterschiedlich gewichtete Aufgaben zu insgesamt 50 Punkten. Mit 25 Punkten haben Sie sicherlich bestanden. Die Teilaufgaben können unabhängig voneinander bearbeitet werden. Gemäß Studienordung gilt für alle (Teil-)Aufgaben mit Antworten zum Ankreuzen (Multiplechoice), dass jede falsche Antwort genau so viel Minuspunkte bringt, wie eine richtige Antwort Pluspunkte bringt. Die Minuspunkte werden aber nicht auf andere Aufgaben übertragen, d.h. im schlimmsten Fall bekommt man 0 Punkte auf die jeweilige Aufgabe. Eine unbeantwortete Frage zählt 0 Punkte. Eventuelle Fußnoten klären mögliche Detailfragen, welche für die Aufgabe aber nicht so wichtig sind.

Mit Ihrer Unterschrift bestätigen Sie, dass Sie zu Beginn der Klausur in ausreichend guter gesundheitlicher Verfassung sind, und diese Klausurprüfung verbindlich annehmen.

Nachname:

Vorname:

Matrikelnummer:

Studiengang:

Hiermit erkläre ich die Richtigkeit der obigen Angaben: ______________________

(Unterschrift)

Aufgabe	1	2	3	4	5	6	7	8	$\sum$	Bonus	Note
Punkte	7	5	5	7	7	5	6	8	**50**		
erreicht											

Aufgabe 1 (Auswertung): (7 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) `(const $ negate $ succ $ 5) 4`

____

(ii) `let (x,y) = (\(_,a,b,_) -> (b,y,a), [1..2]) in x (3,4,5,6)`

____

(iii) `[Nothing, return 3]`

____

(iv) `if False && undefined then undefined else "Kein Fehler"`

____

b) Werten Sie folgenden Ausdruck in Einzelschritten vollständig aus. Unterstreichen Sie dabei jeweils den reduzierten Redex. Verwenden Sie die Auswertestrategie Call-By-Name um die volle Punktzahl zu erreichen.

`(\x -> (\y -> y (2 + 3))) (4 + 5) (\z -> z + z)` $\leadsto$

c) Die Auswertungsstrategien Call-By-Value und Call-By-Name können sich in der Effizienz (Anzahl Auswerteschritte) unterscheiden. Beschreiben Sie in 2–3 Sätzen ein anderes Unterscheidungmerkmal bezüglich des Resultats der Auswertung eines Ausdrucks mit diesen Strategien!

d) Wertet ein Ausdruck unter beiden Strategien zu Werten aus, so sind diese immer identisch.

☐ Ja, das stimmt. ☐ Nein, das ist falsch.

Aufgabe 2 (Induktion): (5 Punkte)

Wir betrachten folgende Funktionsdefinition:

```
foo :: (Integer,Integer) -> Integer
foo (n,m)  |  n == 0    = 0                          -- (NZ)
           |  n >  0    = m + foo (n-1, 2*m)         -- (NP)
           |  otherwise = error "negative arg."      -- (NN)
```

Beweisen Sie mit Induktion, dass für beliebige $(n, m) \in \mathbb{N} \times \mathbb{N}$ die Gleichung $\texttt{foo}(n, m) = m \cdot (2^n - 1)$ gilt. Überlegen Sie sich zuerst worüber die Induktion geführt werden soll, also z.B. ob über n oder m oder $n + m$ oder $n \cdot m$ oder $\max(n, m)$ oder ...

Induktion wird geführt über: ________

Ihr ausführlicher Beweis:

Aufgabe 3 (Abstiegsfunktion): (5 Punkte)

a) Geben Sie lediglich eine möglichst simple Abstiegsfunktion an, mit der man die Termination des folgenden Programmes auf alle Eingaben aus $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ beweisen könnte:

```
hanoi (n,i,j) | n > 1     = hanoi (n',i,ot) ++ [(i,j)] ++ hanoi (n',ot,j)
              | otherwise = [(i,j)]
  where n' = n-1
        ot = 6-i-j
```

Für den Aufruf `hanoi (n,i,j)` definieren wir als Abstiegsfunktion $m(n, i, j) :=$ ____________

b) Für den Rest dieser Aufgabe betrachten wir folgendes unsinniges Programm:

```
baz :: (String,String) -> String
baz (    [],    ys) = 'v':ys
baz ('a':xs,  y:ys) = baz ('b':'b':xs, 'w':ys)
baz ('b':xs,  y:ys) = baz (xs, baz('c':xs, 'd':'d':ys))
baz (  x:xs,    ys) = x : xs ++ ys
```

Zeigen Sie, dass die Funktion m' : `String` $\times$ `String` $\to \mathbb{N}$, welche die Anzahl der `'b'` in beiden Eingabestrings zurückgibt, keine geeignete Abstiegsfunktion für den Terminationsbeweis von `baz` ist. *Beispiel:* m'(“`aabbcc`“, “`bcde`“) $= 3$

c) Beweisen Sie mit einer geeigneten Abstiegsfunktion, dass `baz` für alle[1] Eingaben terminiert. *Hinweise:* Auf die Bedingungen Auf und Def verzichten wir hier. Weiterhin dürfen Sie annehmen, dass `l ++ r` die gleichen Elemente enthält wie die Einzellisten `l` und `r` zusammen.

[1] Für alle vollständig definierten und Typ-korrekten Eingaben, also alle Paare von beliebigen Strings.

Aufgabe 4 (Datenstrukturen): (7 Punkte)

a) Geben Sie eine benutzerdefinierte polymorphe Datentypdeklaration für gewöhnliche Listen an. (Also angenommen es gäbe den eingebauten Listentyp `[a]` nicht, wie könnte man einen dazu äquivalenten Datentyp selbst definieren? Gewöhnliche Präfix-Konstruktoren reichen uns hier.)

b) Machen Sie ihren Datentyp aus der vorherigen Teilaufgabe zu einer Instanz der Typklasse `Functor`!

Hinweis: Falls Sie die vorherige Aufgabe nicht lösen konnten, dann geben Sie die `Functor`-Instanzdeklaration für `[]` an. Sie dürfen keine Funktionen der Standardbibliothek aufrufen.

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 4:

c) Gegeben seien die Datentypen `Op` und `Reg`, deren Definition nicht weiter wichtig ist. Geben Sie eine Deklaration für einen Datentyp `Instruction` an, welcher Assembler Instruktionen modellieren soll. Jede Instruktion besteht aus einem Wert des Typs `Op` (welche die Operation identifiziert) und 0, 1 oder 2 Registerangaben des Typs `Reg`. Ihr neuer Datentyp soll keine `Instruction`-Werte erlauben, welche mehr als 2 Registerangaben enthalten!

Zusätzlich soll es möglich sein, zwei Werte von `Instruction` auf Gleichheit zu testen; dabei dürfen Sie annehmen, dass `Op` und `Reg` bereits ebenfalls vergleichbar sind. Sie dürfen diese Zusatzaufgabe auch auf dem einfachen Weg lösen, falls die hier möglich wäre.

d) Schreiben Sie eine Funktion `catMaybes :: [Maybe a] -> [a]` welche aus einer Liste von `Maybe a`-Werten nur die in `Just` verpackten Werte des Typs `a` in gleichbleibender Reihenfolge herausholt.

Aufgabe 5 (Funktionen höherer Ordnung): (7 Punkte)

Zur Lösung dieser Aufgabe dürfen keine Funktionen der Standardbibliothek verwendet werden. Jedoch alle Infix-Operatoren (`&&`, `+`, `++`, `<=`, ...) und eigene, angegebene Hilfsfunktion sind erlaubt.

a) Implementieren Sie die Funktion `curry :: ((a, b) -> c) -> a -> b -> c`

b) In einer der ersten Übungen haben wir den Sortieralgorithmus Quicksort kennengelernt. Dieser Algorithmus arbeitet nach dem allgemeinen Prinzip "Teile und herrsche" (divide and conquer). Implementieren Sie dieses abstrakte Prinzip als eine Funktion des folgenden Typs:

```
tuh :: (a -> Either (a,c,a) b) -> ((b,c,b) -> b) -> a -> b
```

Dieser Typ enthält eigentlich schon alle benötigten Informationen zum Lösen dieser Aufgabe.

Hinweise: Für ein Problem des Typs `a` soll eine Lösung des Typs `b` berechnet werden; als Hilfsmittel erhält `tuh` dazu in den ersten beiden Argumenten zwei Funktionen:

- Die Funktion des Typs `a -> Either (a,c,a) b`, oft genannt "split", bekommt ein Problem des Typs `a` und entweder spaltet es dieses in zwei (kleinere) Probleme des Typs `a` und ein Zwischenergebnis des Typs `c`; oder aber es löst das Problem direkt durch Rückgabe eines Wertes des Typs `b`.
- Die Funktion des Typs `(b,c,b) -> b`, oft genannt "join", fügt zwei (Teil-)lösungen `b` und ein Zwischenergebnis `c` zu einer Gesamtlösung des Typs `b` zusammen.

Zur Erinnerung: der Typ `Either a b` hat genau zwei Konstruktoren, `Left :: a -> Either a b` und `Right :: b -> Either a b`.

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 5:

c) Implementieren Sie Quicksort unter sinnvoller Verwendung der Funktion

```
tuh :: (a -> Either (a,c,a) b) -> ((b,c,b) -> b) -> a -> b
```

aus der vorangegangenen Teilaufgabe.

Hinweis: Dies können Sie auch tun, ohne die vorangegangene Teilaufgabe gelöst zu haben; im Wesentlichen müssen Sie die beiden Hilfsfunktionen implementieren und an `tuh` übergeben.

Aufgabe 6 (Monaden): (5 Punkte)

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

```
> mapM print [1,2]
1
2
[(),()]
```

b) Schreiben Sie eine Funktion `main :: IO ()`, welche eine Begrüßung ausgibt und danach so lange Zeilen von der Tastatur einliest, bis der Benutzer eine Leerzeile eingibt. Danach gibt das Programm die Anfangsbuchstaben aller Eingabezeilen in umgekehrter Reihenfolge aus. Sie dürfen alle Funktionen der Standardbibliothek benutzen. DO-Notation ist erlaubt.

Beispiel: Die erste und letzte Zeile wurde von dem Programm ausgegeben, alles dazwischen wurde vom Benutzer eingegeben:

```
Sag mir viel Schönes:
Tolle Klausur
Und
Gar nicht schwer!

GUT
```

Aufgabe 7 (Semantik): (6 Punkte)

a) Die Ordnungsrelation einer partiell geordnete Menge (poset) ist auf jedem Fall immer:

☐ Reflexiv	☐ Irreflexiv	☐ Keines von diesen beiden
☐ Symmetrisch	☐ Antisymmetrisch	☐ Keines von diesen beiden
☐ Transitiv	☐ Intransitiv	☐ Keines von diesen beiden
☐ 1-stellig	☐ 2-stellig	☐ 3-stellig

Hinweis: In jeder Zeile ist genau eine Aussage richtig für ein poset.

b) Welche der Nachfolgenden ist eine partiell geordnete Menge (poset):

(i) $\big(\mathbb{N} \times \mathbb{N},\ (x_1, y_1) \succcurlyeq (x_2, y_2)$ genau dann wenn $x_1 \geq x_2$ und $y_2 \geq y_1$ gilt.$\big)$

☐ Ist ein poset ☐ Kein poset

(ii) $\big(\{\diamondsuit, \heartsuit, \spadesuit, \clubsuit\},\ \{(\diamondsuit, \diamondsuit), (\diamondsuit, \heartsuit), (\heartsuit, \heartsuit), (\heartsuit, \spadesuit), (\spadesuit, \spadesuit), (\spadesuit, \clubsuit), (\clubsuit, \clubsuit), (\clubsuit, \diamondsuit)\}\big)$

☐ Ist ein poset ☐ Kein poset

c) Zeichnen Sie ein Hasse Diagramm für die vollständige Halbordnung (dcpo), welche den Datentyp `(Bool,Bool)` im Sinn der denotationellen Semantik modelliert:

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 7:

d) Wir betrachten folgende rekursive Funktionsdefinition:

```
sem (a,b) = if a then sem (b,not a) else if b then sem (not a, b) else True
```

Wir möchten nun die semantische Interpretation dieser rekursiven Definition durch Fixpunktiteration bestimmen. Dazu definieren wir:

$$\texttt{sem}_i(a,b) = \texttt{if}\ a\ \texttt{then}\ sem_{i-1}(b, \texttt{not}\ a)\ \texttt{else if}\ b\ \texttt{then}\ \texttt{sem}_{i-1}(\texttt{not}\ a, b)\ \texttt{else True}$$

Geben Sie jede dabei entstehende partielle Funktion an, bis der Fixpunkt erreicht wird! Sie können die Funktionen der Iteration in Haskell-Notation (`f0 _ = undefined`) oder in mathematischer Notation ($f_0(a,b) = \bot$) angeben.

Aufgabe 8 (Typen): (8 Punkte)

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

(i) `[tail "head", undefined, []]`

____

(ii) `\x -> (\y -> x || otherwise)`

____

(iii) `let bar _ x y z | x==z = read y in bar 27`

____

b) Geben Sie den allgemeinsten Unifikator für folgende Typgleichungen an:
$\{\alpha \to \big((\gamma \to \delta) \to \texttt{Bool}\big) = (\beta \to \gamma) \to (\beta \to \gamma)\}$

____

c) Beweisen Sie folgendes Typurteil unter Verwendung der auf Seite 13 angegeben Typregeln in einer der beiden behandelten Notationen (Herleitungsbaum oder lineare Schreibweise). Die Typregeln Pair-Intro und Pair-Elim sind neu, Sie sollten diese aber leicht lesen und anwenden können! *Hinweise:* Wenn Sie die Herleitung auf der Rückseite dieses Blattes zeichnen, dann brauchen Sie nicht ständig umblättern, um die Typregeln dabei zu sehen! Beim Zeichnen eines Baumes können Sie ggf. Platz sparen, wenn Sie den Kontext am Anfang jeder Konklusion durch eine Variable $\Gamma_1, \Gamma_2, \Gamma_3, \ldots$ abkürzen und woanders auf dem Blatt erklären, wie diese Kontexte definiert sind.

$\{\ \} \vdash \backslash x$ `->` `let` (y, z) `=` x `in` $(z$ `True`$)\ y :: (\texttt{Int}, \texttt{Bool} \to \texttt{Int} \to \texttt{Char}) \to \texttt{Char}$

Typregeln:

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

Alternative Schreibweise für Var:

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

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

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

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

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

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

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

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