Prof. Dr. Visvanathan Ramesh
PD Dr. Arne Nägel
Fachbereich Informatik und Mathematik
Institut für Informatik

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Probeklausur Programmierparadigmen und Compilerbau

Sommer 2021

13. Juli 2021

Name:

Vorname:

Matrikelnummer:

Studienfach / Abschluss:

Hinweise:

1. Füllen Sie bitte **sofort** die oben verlangten Angaben aus. Auf allen übrigen Blättern tragen Sie bitte Ihren Namen ein. Nicht in dieser Weise gekennzeichnete Blätter können von der Wertung ausgeschlossen werden.
2. Lose Blätter sind nicht erlaubt. Lassen Sie daher diesen Stapel geheftet und bearbeiten Sie die Aufgaben direkt auf den Blättern. Sollte der ausreichend eingeräumte Platz dennoch einmal nicht genügen, so **verweisen** Sie bitte auf die anhängenden Zusatzblätter.
3. Außer gewöhnlichen dokumentenechten Stiften — **nicht in rot und auch kein Bleistift**! — sind **keine Hilfsmittel** zugelassen, auch nicht Tipp-Ex o.ä. Sollen Teile der Lösung ungültig gemacht werden, müssen diese durchgestrichen werden.
4. Wer während der Klausur elektronische Geräte wie Organizer, Smartphones, oder ähnliches benutzt, wird von der Klausur ausgeschlossen.
5. Die Klausur besteht aus *nn* Aufgaben, die **nicht** nach Schwierigkeitsgrad geordnet sind. Bitte verschaffen Sie sich zunächst einen Überblick, bevor Sie die einzelnen Aufgaben bearbeiten. Die Klausur besteht aus XX Seiten. Bitte prüfen Sie, ob Sie eine vollständige Klausur haben.

   In der Klausur können insgesamt 100 Punkte erreicht werden. Mit dem Erreichen von 50 Punkten ist die Klausur sicher bestanden.
6. Die Dauer der Klausur beträgt 90 Minuten. Bitte bleiben Sie nach Ablauf dieser Zeit an Ihrem Platz, bis alle Klausuren eingesammelt sind. Sollten Sie mehr als 15 Minuten früher fertig sein, so können Sie nach Abgabe Ihrer Klausur leise den Raum verlassen.

Viel Erfolg!

Nachfolgende Tabelle bitte **nicht** ausfüllen.

Aufgabe	1	2	3	4	5	6	7	8
Punkte								
Aufgabe	9	10	11	12	13	14	15	16
Punkte								

Punkte	**Note**

Aufgabe 1: Multiple-Choice Fragen (4 Punkte)

Kreuzen Sie für jede der folgenden Aussagen an, ob diese **wahr** oder **falsch** ist.

Bewertung: Für jedes richtige Kreuz gibt es einen halben Punkt, für jedes falsche Kreuz wird ein halber Punkt abgezogen. Insgesamt gibt es nicht weniger als 0 Punkte.

1. Haskell ist eine objektorientierte Programmiersprache. ☐ wahr ☐ falsch
2. Ein Ausdruck, der bei Auswertung in applikativer Reihenfolge terminiert, terminiert auch bei Auswertung in normaler Reihenfolge. ☐ wahr ☐ falsch
3. Haskell ist eine dynamisch typisierte Sprache. ☐ wahr ☐ falsch
4. Haskell erlaubt polymorphe Typen. ☐ wahr ☐ falsch
5. Die Funktionsdefinition `f x = x (f x)` ist syntaktisch korrekt. ☐ wahr ☐ falsch
6. Im Ausdruck `let` $y =$ `\` t `->` $2 * t$ `in` $y\,t$ tritt die Variable t sowohl frei als auch gebunden auf. ☐ wahr ☐ falsch
7. Eine Baumrekursion lässt sich immer auch als Endrekursion schreiben. ☐ wahr ☐ falsch
8. Die Listenfunktion `map :: (a -> b) -> [a] -> [b]` ist eine Funktion höherer Ordnung. ☐ wahr ☐ falsch

Aufgabe 2: List Comprehensions in Haskell (8 Punkte)

a) Geben Sie die **Ergebnislisten** an, die durch die folgenden List Comprehensions erzeugt werden:

- `[x | x <- [1..3], y <- [1..2], x > y]` (2 Punkte)

- `[x | x <- ['A','B','C','D'], y <- [1..4], even y]` (3 Punkte)

b) Gegeben sei eine Liste xxs von Paaren. Definieren Sie in Haskell eine **List Comprehension**, welche die jeweiligen Summen $x + y$ nur solcher Paare (x, y) aus xxs enthält, für die gilt:
$x > y$, x ist durch 3 teilbar und y ist **nicht** durch 3 teilbar.

Z.B. lautet für $xxs =$ `[(2,3),(3,2),(4,3),(9,1)]` das Ergebnis `[5,10]`. (3 Punkte)

Aufgabe 3: Auswertung in Haskell (9s Punkte)

In dieser Aufgabe geht es um die Auswertung. In den Teilaufgaben a), b) und c) sollen Sie Auswertungsschritte angeben. Tragen Sie dabei für jeden Schritt die **verwendete Regel D**, **I** oder **A** in das danebenstehende Kästchen ein, wobei deren Bedeutung ist:

- **D** Definitionseinsetzung (δ-Reduktion)
- **I** Auswertung einer Fallunterscheidung (if-Reduktion)
- **A** Auswertung von arithmetischen Ausdrücken

Die Funktionen `f`, `g` und `h` seien definiert als:

```
f x y = h (h (g x (x+y) y))

g a b c = if c > a then (if a < c then f (a+b) (b+c) else g a (b-1) c)
                   else h b

h j = if j < 2 then g 5 j 3 else f 7 9
```

a) Führen Sie **die nächsten zwei** Auswertungsschritte in **normaler Reihenfolge** für den folgenden Ausdruck aus. `f (1+1) 7` (2 Punkte)

Erster Auswertungsschritt:
Regel: Ausdruck:

☐ _____

Zweiter Auswertungsschritt:
Regel: Ausdruck:

☐ _____

b) Führen Sie **die nächsten zwei** Auswertungsschritte in **applikativer Reihenfolge** für den folgenden Ausdruck aus. `f (h 1) (h 2)` (2 Punkte)

Erster Auswertungsschritt:
Regel: Ausdruck:

☐ _____

Zweiter Auswertungsschritt:
Regel: Ausdruck:

☐ _____

c) Führen Sie **die nächsten zwei** Auswertungsschritte in **verzögerter Reihenfolge** für den folgenden Ausdruck aus. `g 7 (h 2) (9*9)` (2 Punkte)

Erster Auswertungsschritt:
Regel: Ausdruck:

☐ _____

Zweiter Auswertungsschritt:
Regel: Ausdruck:

☐ _____

d) Geben Sie eine Funktion `fun :: Bool -> Int -> Int` an, die für den Aufruf `fun False 5` bei normaler Auswertung terminiert und beim selben Aufruf mit applikativer Auswertung nicht terminiert. Die Definition der Funktion `fun` darf höchstens 5 Zeilen lang sein. (3 Punkte)

Zur Erinnerung: Die Funktionen `f`, `g` und `h` seien definiert als:

```
f x y = h (h (g x (x+y) y))
g a b c = if c > a then (if a < c then f (a+b) (b+c) else g a (b-1) c)
                   else h b
h j = if j < 2 then g 5 j 3 else f 7 9
```

Aufgabe 4: Rekursive Listenfunktionen (7 Punkte)

a) Implementieren Sie in Haskell eine Funktion `f :: [Integer] -> [Integer]`, die eine **Liste von Ganzzahlen** erwartet und eine Liste zurückliefert, bei der alle Elemente an ungerader Listenposition (die Nummerierung der Elemente beginnt bei 1) gelöscht wurden und außerdem alle Zahlen entfernt wurden, die durch 3 teilbar sind. `f [1,3,4,5,6,7]` soll z.B. `[5,7]` liefern. (3 Punkte)

b) Die Funktion `g` sei in Haskell definiert als

```
g xs = foldl (\x y -> x-y) 8 xs
```

Welche Ausgabe erhalten Sie beim Aufruf von

```
g [1..5]
```

im GHCi? (2 Punkte)

c) Die Funktion `h` sei in Haskell definiert als

```
h [] = []
h ((True,[a1,b1],False,z1) : (False,[a2,b2],True,z2) : lst) =
    (False,[b1,a1],True,z1+z2) : h lst
h ((True,[a1,b1],False,z1) : lst)  =
    (True,[a1,b1],False,z1+1) : h lst
```

Welche Ausgabe erhalten Sie beim Aufruf von

```
h [(True,[1,2],False,3),(False,[3,4],True,5),(True,[6,7],False,8)]
```

im GHCi? (2 Punkte)

Aufgabe 5: Bäume in Haskell (6 Punkte)

Binäre Bäume mit Blatt- und Knotenmarkierungen können in Haskell durch den Datentyp `Baum` repräsentiert werden:

```
data Baum a = Blatt a | Knoten (Baum a) -- linker Teilbaum
                               a
                               (Baum a) -- rechter Teilbaum
```

a) Definieren Sie in Haskell eine Funktion `inOrder :: Baum a -> [a]`, die zu einem gegebenen Baum die Liste von Blatt- und Knotenmarkierungen bei *in-order Traversierung* berechnet.

Hinweis: Bei der *in-order Traversierung* eines binären Baumes wird zuerst der linke Teilbaum, dann die Wurzel und dann der rechte Teilbaum durchlaufen.
D.h. `inOrder (Knoten (Blatt 'B') 'A' (Blatt 'C'))` liefert als Ergebnis `['B','A','C']`.

(2 Punkte)

b) Definieren Sie eine Haskell-Funktion `g :: Baum Int -> Baum Int`, die für jeden Knoten der kein Blatt ist, die Knoten-Markierung um 1 erhöht, sofern die Markierung des linken Kindes des Knotens echt kleiner als die Markierung des rechten Kindes ist. Ansonsten soll der Eingabe-Baum nicht verändert werden. Z.B. soll `Knoten (Knoten (Blatt 4) 4 (Blatt 5)) 1 (Blatt 2)` für `g (Knoten (Knoten (Blatt 4) 3 (Blatt 5)) 1 (Blatt 2))` berechnet werden. (4 Punkte)

Aufgabe 6: Typherleitung in Haskell (8 Punkte)

Die Funktionen `f` und `g` seien wie folgt getypt:

```
f :: ((a,b) -> b) -> [a] -> [b]
g :: (Char,Int) -> c
```

Berechnen Sie **den Typ** der Anwendung `(f g)`

Hinweise:

- Die **Typregel für die Anwendung** lautet: $\dfrac{s :: \sigma \to \tau, \quad t :: \rho \text{ und } \gamma(\sigma) = \gamma(\rho)}{(s\ t) :: \gamma(\tau)}$
- Geben Sie auch die Berechnung der entsprechenden **Typsubstitution** γ mittels Unifikation an.
- Der Typpfeil `->` ist rechts-assoziativ, d.h. `a -> b -> c` entspricht `a -> (b -> c)`.

Aufgabe 7: Selbstdefinierte Datentypen in Haskell (9 Punkte)

In dieser Aufgabe betrachten wir die Häuserblöcke von Manhattan. Grundlegende Attribute eines Hochhauses seien in Haskell durch den folgenden Datentyp `Hochhaus` dargestellt. Der Datentyp `Block` stellt einen Häuserblock von Manhattan dar. Dabei sind die Hochhäuser eines Blocks als Liste dargestellt, bei der die Reihenfolge der Hochhäuser keine Bedeutung hat. Außerdem wird für jeden Block eine Liste der Namen aller angrenzenden Straßen sowie eine Liste der Namen aller unmittelbar vom Block erreichbaren Buslinien gespeichert.

```
data Hochhaus = Hochhaus Name Hoehe Nutzung Etagen Baujahr
  deriving(Eq,Show)

type Name    = String
type Hoehe   = Int    -- angegeben in Metern.
type Etagen  = Int
type Baujahr = Int
data Nutzung = Bueros | Wohnungen | Hotel | Mehrzweck
  deriving(Eq,Show)

data Block = Block [Hochhaus] [Strasse] [Buslinie]
  deriving(Eq,Show)

type Strasse  = String
type Buslinie = String
```

a) Geben Sie die Haskell-Darstellung vom Typ `Block` für den folgenden Häuserblock an:

Hochhäuser:	Chrysler Building, 319m hoch, Bürogebäude, 77 Etagen, erbaut 1930.
	Chrysler Building East, 132m hoch, Bürogebäude, 32 Etagen, erbaut 1999.
Straßen:	Lexington Ave, E 42nd St, E 43rd St, 3rd Ave
Buslinien:	M42, M101, M102, M103, X21, X63, X64, X68

(2 Punkte)

b) Definieren Sie eine Haskell-Funktion `maximaleHoehe :: Block -> Hoehe`, die für einen Block die Höhe des höchsten Hochhauses berechnet. (3 Punkte)

c) Manchmal wird in einem Block ein altes Hochhaus abgerissen und anschließend auf dem frei gewordenen Platz ein neues gebaut. Definieren Sie eine Haskell-Funktion `ersetzeHochhaus :: String -> Hochhaus -> Block -> Block`, die den Namen des abzureißenden Hochhauses, ein neu zu bauendes Hochhaus und den entsprechenden Block entgegennimmt und den Block zurückgibt, bei dem das abzureißende Hochhaus durch das neu zu bauende ersetzt wurde. Sie können davon ausgehen, dass die Namen der Hochhäuser eindeutig sind. (4 Punkte)

Leerseite

Leerseite