Karlsruher Institut für Technologie

Institut für Programmstrukturen und Datenorganisation

Prof. Dr.-Ing. Gregor Snelting
gregor.snelting@kit.edu

Prof. Dr. Ralf H. Reussner
reussner@kit.edu

Klausur Programmierparadigmen

SS18, 26. September 2018, 10:00 – 12:00 Uhr

Zugelassene Hilfsmittel: Papierbasierte Quellen (Vorlesungsfolien, Übungsblätter, eigene Aufzeichnungen, Bücher, …)
Die Verwendung von elektronischen Geräten ist verboten.

Bearbeitungszeit: 120 min

Aufgabe	**max. Punkte**	**err. Punkte**
1	15	
2	15	
3	20	
4	13	
5	15	
6	10	
7	20	
8	12	
Σ	**120**	
$\lceil\Sigma\rceil$	**120**	

Jeder Punkt entspricht ca. 1 min Bearbeitungszeit. Es ist garantiert, dass die Klausur mit 60 Punkten bestanden ist, und dass 115 Punkte für die Note 1,0 ausreichen.

Hinweis für Lehramtsstudierende: Bitte nur die Aufgaben 1 und 2 bearbeiten. Sie haben dazu 30 Minuten Zeit.

Name: ______________________________

Matrikelnummer: ______________

Studiengang: ______________

Schreiben Sie Ihre Lösungen direkt in die Klausur. Beschriften Sie **alle verwendeten Blätter** mit Ihrem Namen und Ihrer Matrikelnummer. Trennen Sie die geklammerten Blätter **nicht** auf. Weitere Blätter erhalten Sie bei Bedarf.

Aufgabe 1 (Haskell: Vollkommene Zahlen) [15 Punkte]

Eine Zahl $n \geq 2$ heißt vollkommen, wenn die Summe ihrer echten Teiler (Teiler außer n selbst) wieder n ergibt.
Beispielsweise sind 6 und 28 vollkommene Zahlen, da $1 + 2 + 3 = 6$ und $1 + 2 + 4 + 7 + 14 = 28$.

(a) Implementieren Sie die Funktionen [5 Punkte]

```
properDivisors :: Integer -> [Integer]
perfectNumber :: Integer -> Bool
```

`properDivisors n` berechnet die Liste der echten Teiler von $n \geq 2$, und `perfectNumber n` soll genau dann gelten, wenn n eine vollkommene Zahl ist.

(b) Implementieren Sie eine optimierte Variante von [10 Punkte]
`properDivisors`, die höchstens $\sqrt{n}$ Teilbarkeitsprüfungen durchführt. Dabei gilt:

- 1 ist ein echter Teiler von n.
- Ist $2 \leq i < \sqrt{n}$ und ist n durch i teilbar, so sind i und $\frac{n}{i}$ echte Teiler von n.
- Ist $i = \sqrt{n}$ eine natürliche Zahl, so ist i ein echter Teiler von n.

Sie dürfen dafür die Funktion `isqrt :: Integer -> Integer` verwenden, die die abgerundete Quadratwurzel berechnet. Was die Funktion für nicht-positive Eingaben tut, bleibt Ihnen überlassen.

Aufgabe 2 (Haskell: ListBuilder) [15 Punkte]

Auf Haskells Listen ist das Einfügen von neuen Elementen am Ende sehr ineffizient. Zum Beispiel benötigt die bekannte naive Implementierung von `reverse` deswegen Zeit in $\mathcal{O}(n^2)$.

Eine Datenstruktur ohne diese Schwäche ist `ListBuilder`:

```
data ListBuilder a
  = Nil
  | Cons a (ListBuilder a)
  | Append (ListBuilder a) (ListBuilder a)
```

`ListBuilder` erweitert Listen um einen zusätzlichen Konstruktor `Append`, der die Verkettung zweier `ListBuilder` darstellt. Hier sehen Sie zwei gültige Darstellungen der Liste `[1, 2, 3]` als `ListBuilder`:

```
Append (Cons 1 Nil) (Cons 2 (Cons 3 Nil))

Cons 1 (Cons 2 (Cons 3 Nil))
```

(a) Implementieren Sie die Funktion [3 Punkte]

```
fromList :: [a] -> ListBuilder a
```

die eine Liste in eine gültige Darstellung als `ListBuilder` überführt.

(b) Implementieren Sie die Funktion [5 Punkte]

```
rev :: ListBuilder a -> ListBuilder a
```

`rev lb` soll die umgedrehte von `lb` dargestellte Liste darstellen.

(c) Implementieren Sie die Hilfsfunktion [8 Punkte]

```
toListAcc :: ListBuilder a -> [a] -> [a]
```

sodass

```
toList :: ListBuilder a -> [a]
toList lb = toListAcc lb []
```

die die von einem `ListBuilder` dargestellte Liste zurückgibt.

Achten Sie darauf, dass die Konvertierung in Linearzeit erfolgt [1]. Ähnlich wie bei der effizienten Variante von `reverse` wird die vom `ListBuilder` dargestellte Liste der als Argument übergebenen Liste zu gegebener Zeit vorangestellt.

Beispiel:

```
> toList (rev (fromList [1, 2, 3]))
[3, 2, 1]
```

[1] Kurzum: Verzichten Sie auf `(++)` und Reimplementierung dieser Funktion.

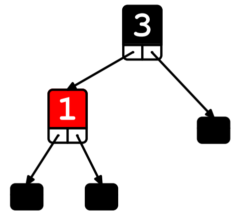
Aufgabe 3 (Prolog, Rot-Schwarz-Bäume) [20 Punkte]

Rot-Schwarz-Bäume sind gefärbte binäre Bäume mit den Eigenschaften

(i) sortiert

(ii) kein roter Knoten hat roten Kindknoten

(iii) alle vollständigen Pfade haben gleiche Anzahl schwarzer Knoten



Blätter zählen dabei als schwarze Knoten.

Gefärbte binäre Bäume lassen sich als Prolog-Terme darstellen, z.B., der abgebildete Baum `TExample` als `node(black, node(red, leaf, 1, leaf), 3, leaf).`

(a) Definieren Sie ein Prädikat `sorted(T)`, das genau dann erfüllt ist, wenn `T` Eigenschaft (i) erfüllt. [9 Punkte]

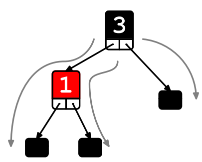
Hinweis: Es gilt: Blätter `leaf` sind sortiert, sowie: Knoten `node(Color, Left, X, Right)` sind sortiert, falls

- alle Knoten in `Left` Werte $\leq$ `X` haben,
- alle Knoten in `Right` Werte $\geq$ `X` haben, und
- die Bäume `Left` und `Right` sortiert sind.

Definieren Sie geeignete Hilfsprädikate.

(b) Definieren Sie ein Prädikat `colorPath(T,P)`, das bei Wiedererfüllung alle vollständigen Pfade durch `T` — repräsentiert als Liste von Farben — aufzählt. [5 Punkte]

```
? colorPath(TExample,P).
P = [black, red, black] ;
P = [black, red, black] ;
P = [black, black].
```



(c) Definieren Sie ein Prädikat `redRed(T)` das genau dann erfüllt ist, wenn `T` Eigenschaft (ii) *verletzt*. [6 Punkte]

Hinweis: Definieren Sie `redRed` direkt, *oder*: mittels `colorPath(T,P)` zusammen mit einem Hilfsprädikat `member2(X,L)` welches erfüllt ist, wenn `L` zwei aufeinanderfolgende Vorkommen von `X` enthält.

Aufgabe 4 (Currying im λ-Kalkül) [13 Punkte]

Paare von Werten lassen sich im λ-Kalkül mit der sogenannten Church-Kodierung darstellen. Gegeben sind folgende Ausdrücke `pair`, `fst` und `snd`, welche Church-Paare konstruieren, sowie das erste bzw. zweite Element extrahieren können:

$$\texttt{pair} = \lambda\texttt{a}.\ \lambda\texttt{b}.\ \lambda\texttt{f}.\ \texttt{f a b} \qquad \texttt{fst} = \lambda\texttt{p}.\ \texttt{p}\ (\lambda\texttt{a}.\ \lambda\texttt{b}.\ \texttt{a}) \qquad \texttt{snd} = \lambda\texttt{p}.\ \texttt{p}\ (\lambda\texttt{a}.\ \lambda\texttt{b}.\ \texttt{b})$$

Für beliebige Ausdrücke a, b gilt dann:

- $\texttt{fst}\ (\texttt{pair}\ a\ b) \Rightarrow^* a$
- $\texttt{snd}\ (\texttt{pair}\ a\ b) \Rightarrow^* b$

(a) Geben Sie zwei Ausdrücke `curry` und `uncurry` an, welche folgende Eigenschaften für beliebige Ausdrücke a, b, f, g erfüllen: [3 Punkte]

i. $(\texttt{curry}\ f)\ a\ b \Rightarrow^* f\ (\texttt{pair}\ a\ b)$
ii. $(\texttt{uncurry}\ g)\ (\texttt{pair}\ a\ b) \Rightarrow^* g\ a\ b$

(b) Rechnen Sie nach, dass folgendes für alle Ausdrücke a, b, f, g gilt (Sie dürfen hierbei die Eigenschaften aus Teilaufgabe (a) benutzen): [4 Punkte]

i. $(\texttt{curry}\ (\texttt{uncurry}\ g))\ a\ b \Rightarrow^* g\ a\ b$
ii. $(\texttt{uncurry}\ (\texttt{curry}\ f))\ (\texttt{pair}\ a\ b) \Rightarrow^* f\ (\texttt{pair}\ a\ b)$

3-Tupel können dargestellt werden als geschachtelte 2-Tupel. So stellt der Term $(\texttt{pair}\ a\ (\texttt{pair}\ b\ c))$ das Tupel (a, b, c) dar.
Gegeben seien nun zwei Funktionen `curry3` und `uncurry3` wie folgt:

$$\texttt{curry3} = \lambda\texttt{f}.\ \lambda\texttt{a}.\ \texttt{curry}\ ((\texttt{curry f})\ \texttt{a}) \qquad \texttt{uncurry3} = \lambda\texttt{g}.\ \texttt{uncurry}\ (\lambda\texttt{a}.\ \texttt{uncurry}\ (\texttt{g a}))$$

(c) Rechnen Sie nach, dass folgende Eigenschaften gelten (Sie dürfen hierfür die Eigenschaften aus allen vorherigen Teilaufgaben benutzen): [6 Punkte]

i. $\texttt{curry3}\ f\ a\ b\ c \Rightarrow^* f\ (\texttt{pair}\ a\ (\texttt{pair}\ b\ c))$
ii. $\texttt{uncurry3}\ g\ (\texttt{pair}\ a\ (\texttt{pair}\ b\ c)) \Rightarrow^* g\ a\ b\ c$

Aufgabe 5 (Typinferenz) [15 Punkte]

Diese Aufgabe beschäftigt sich mit der Typherleitung für den Lambda-Term

$$\lambda\mathtt{x}.\ \mathtt{let}\ \mathtt{f} = \lambda\mathtt{y}.\ \lambda\mathtt{z}.\ \mathtt{y}\ (\mathtt{x}\ \mathtt{z})\ \mathtt{in}\ \mathtt{f}\ (\lambda\mathtt{y}.\ \mathtt{x})$$

Gegeben ist das folgende Skelett des Herleitungsbaums

$$\textit{Abs}\ \dfrac{\textit{Let}\ \dfrac{\textit{Abs}\ \dfrac{\vdots}{\mathtt{x} : \alpha_2 \vdash \lambda\mathtt{y}.\ \lambda\mathtt{z}.\ \mathtt{y}\ (\mathtt{x}\ \mathtt{z}) : \alpha_4} \qquad \dfrac{\dfrac{①}{\Gamma' \vdash \mathtt{f} : \alpha_{13}} \qquad \dfrac{\dfrac{②}{\Gamma', \mathtt{y} : \alpha_{15} \vdash \mathtt{x} : \alpha_{16}}}{\Gamma' \vdash \lambda\mathtt{y}.\ \mathtt{x} : \alpha_{14}}\ \textit{Abs}}{\Gamma' \vdash \mathtt{f}\ (\lambda\mathtt{y}.\ \mathtt{x}) : \alpha_3}\ \textit{App}}{\mathtt{x} : \alpha_2 \vdash \mathtt{let}\ \mathtt{f} = \lambda\mathtt{y}.\ \lambda\mathtt{z}.\ \mathtt{y}\ (\mathtt{x}\ \mathtt{z})\ \mathtt{in}\ \mathtt{f}\ (\lambda\mathtt{y}.\ \mathtt{x}) : \alpha_3}}{\vdash \lambda\mathtt{x}.\ \mathtt{let}\ \mathtt{f} = \lambda\mathtt{y}.\ \lambda\mathtt{z}.\ \mathtt{y}\ (\mathtt{x}\ \mathtt{z})\ \mathtt{in}\ \mathtt{f}\ (\lambda\mathtt{y}.\ \mathtt{x}) : \alpha_1}$$

Aus dem linken (nicht dargestellten) Teilbaum ergibt sich das folgende Constraintsystem C_{let} mit allgemeinstem Unifikator σ_{let}

$$\begin{aligned} C_{let} = \{\, & \alpha_4 = \alpha_5 \to \alpha_6, \\ & \alpha_6 = \alpha_7 \to \alpha_8, \\ & \alpha_9 = \alpha_{10} \to \alpha_8, \\ & \alpha_5 = \alpha_9, \\ & \alpha_{11} = \alpha_{12} \to \alpha_{10}, \\ & \alpha_2 = \alpha_{11}, \\ & \alpha_7 = \alpha_{12} \,\} \end{aligned} \qquad \begin{aligned} \sigma_{let} = [\, & \alpha_{12} \Rightarrow \alpha_7, \\ & \alpha_{11} \Rightarrow \alpha_7 \to \alpha_{10}, \\ & \alpha_9 \Rightarrow \alpha_{10} \to \alpha_8, \\ & \alpha_6 \Rightarrow \alpha_7 \to \alpha_8, \\ & \alpha_5 \Rightarrow \alpha_{10} \to \alpha_8, \\ & \alpha_4 \Rightarrow (\alpha_{10} \to \alpha_8) \to \alpha_7 \to \alpha_8, \\ & \alpha_2 \Rightarrow \alpha_7 \to \alpha_{10} \,] \end{aligned}$$

(a) Geben Sie die Constraintmenge C_0 an, welche sich aus der Typherleitung bis zur Anwendung der *Let*-Regel ergibt. [1 Punkt]

(b) Berechnen Sie Γ' und vervollständigen Sie die Typherleitung, indem Sie ergänzen, was an den mit ① und ② markierten Stellen einzutragen ist. [6 Punkte]

(c) Geben Sie nun das vollständige Constraintsystem C zur Typherleitung von $\lambda\mathtt{x}.\ \mathtt{let}\ \mathtt{f} = \lambda\mathtt{y}.\ \lambda\mathtt{z}.\ \mathtt{y}\ (\mathtt{x}\ \mathtt{z})\ \mathtt{in}\ \mathtt{f}\ (\lambda\mathtt{y}.\ \mathtt{x})$ an (vgl. Skript Folie 333). [8 Punkte]

Aufgabe 6 (MPI) [10 Punkte]

Gegeben sei die unten stehende Methode, welche in einem MPI-Programm mit der per Präprozessor definierten Anzahl `LENGTH` an Prozessen aufgerufen wird, und welcher eine Matrix a, sowie zwei Vektoren b und c übergeben werden. Sie können davon ausgehen, dass die Initialisierung und Finalisierung von MPI vom Aufrufer übernommen wird.
Hinweis: Es wird angenommen, dass das mehrdimensionale Array a sequentiell im Speicher repräsentiert ist.

```
double* product(double a[LENGTH][LENGTH], double b[LENGTH],
            double c[LENGTH]) {
   int rank;
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   int valuesPerThread = LENGTH;
   double rowA[valuesPerThread];
   double cellB;
   double cellC;
   double fullBC[valuesPerThread];

   MPI_Scatter(a, valuesPerThread, MPI_DOUBLE,
      rowA, valuesPerThread, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Scatter(b, 1, MPI_DOUBLE,
      &cellB, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Scatter(c, 1, MPI_DOUBLE,
      &cellC, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

   double firstResult = cellB * cellC;

   MPI_Gather(&firstResult, 1, MPI_DOUBLE,
      fullBC, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   MPI_Bcast(fullBC, valuesPerThread, MPI_DOUBLE, 0, MPI_COMM_WORLD);
   double secondResult = 0;
   for (int i = 0; i < valuesPerThread; i++) {
      secondResult += rowA[i] * fullBC[i];
   }

   double* result = malloc(sizeof(double) * valuesPerThread);
   MPI_Gather(&secondResult, 1, MPI_DOUBLE,
      result, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

   return result;
}
```

(a) Erklären Sie, welche Funktion die Methode `product` erfüllt und geben Sie den Rückgabewert für folgende Eingabe an: [4,5 Punkte]
`a = {{1, 2}, {2, 4}}, b = {1, 2}, c = {3, 1}`

(b) Schreiben Sie einen Ersatz für die Zeilen 21 bis 23, der ausschließlich aus dem Aufruf einer einzigen MPI-Operation besteht. [2 Punkte]

(c) Erläutern Sie kurz, wie Sie die Zeilen 30 bis 31 *ohne* die Verwendung von kollektiven MPI-Operationen ausdrücken können. Gehen Sie insbesondere auf das Verhalten des Root-Prozesses ein. [3,5 Punkte]
Hinweis: Sie müssen nicht (dürfen aber) den nötigen Quellcode angeben, eine textuelle Erläuterung genügt.

Aufgabe 7 (Java und Design by Contract: Gefilterte Summe) [20 Punkte]

Gegeben sei folgende Klasse, die die Methode `filteredSum` implementiert und auf die Methode `filter` zurückgreift, welche einen Filter auf das übergebene Element anwendet, und entsprechend des Ergebnisses entweder den Wert zurückliefert oder ansonsten 0.

```java
public class FilteredSum {
   private static final int THREAD_COUNT = 4;

   public int filteredSum(ArrayList<Integer> elements)
         throws InterruptedException, ExecutionException {
      ExecutorService executorService =
            Executors.newFixedThreadPool(THREAD_COUNT);

      List<Future<Integer>> results = new ArrayList<>();
      for (Integer element : elements) {
         results.add(executorService.submit(() -> this.filter(element)));
      }

      int filteredSum = 0;
      for (Future<Integer> result : results) {
         filteredSum += result.get();
      }

      executorService.shutdown();
      return filteredSum;
   }

   /*@
     @ ensures \result >= 0;
    */
   private int filter(int element) {
      return checkLessTen(element) ? element : 0;
   }

   /*@
     @ requires element >= 0;
     @ ensures \result == element < 10;
    */
   private boolean checkLessTen(int element) {
      return element < 10;
   }
}
```

Parallelverarbeitung in Java (15 Punkte)

Ignorieren Sie für die folgenden Teilaufgaben zunächst die JML-Verträge der Methoden.

(a) Bei der Ausführung der Implementierung mit einer `ArrayList`, die als Testdaten alle Zahlen aus dem Intervall $[0, 1, ..., 10.000.000[$ enthält, stellen Sie fest, dass Sie gegenüber einer sequentiellen Implementierung einen „Speedup" von 0,005 erreichen. Der Speedup ist dabei der Quotient aus sequentieller und paralleler Ausführungszeit. Dieser tritt unabhängig von der verwendeten Hardware auf. Was bedeutet dieser Speedup und wie erklären Sie dies? [3 Punkte]

(b) Erwarten Sie eine Veränderung des Speedups, wenn Sie stattdessen als Eingabedaten eine `ArrayList` mit 10 Millionen Elementen aus dem Intervall [0..9] verwenden? Begründen Sie Ihre Antwort kurz. [2 Punkte]

(c) Geben Sie als Ersatz für die Zeilen 10 bis 12 eine parallelisierte Implementierung unter Verwendung des `executorService` sowie der Methode `filter` an, die einen möglichst hohen Speedup aufweist. Verwenden Sie dazu *keine* Streams. Begründen Sie außerdem kurz, warum sich der Speedup durch Ihre Implementierung erhöht. [7,5 Punkte]

Hinweis: Es wird eine im Allgemeinen gleichmäßige, aber in typischen Grenzfällen keine *optimale* Verteilung der Arbeit auf verschiedene Threads erwartet. Beispielsweise ist bei 3 Elementen und 4 Threads die Zuteilung der gesamten Arbeit auf einen Thread valide, bei 4 Elementen und 4 Threads jedoch nicht.

(d) Geben Sie eine Implementierung des gesamten Algorithmus (also der `filteredSum`-Methode) mittels parallelen Streams an. [2,5 Punkte]

Hinweis: Ein `IntStream` (siehe Foliensatz 5.5 - Folie 29) bietet die Methode `sum()` an, welche die Summe der Stream-Elemente zurückgibt.

Design by Contract (5 Punkte)

Betrachten Sie für die folgenden Teilaufgaben die JML-Verträge der Methoden.

(e) Betrachten Sie den Aufruf der `checkLessTen`-Methode durch die `filter`-Methode in Zeile 27. Wird der Vertrag hier *vom Aufrufer* erfüllt? Begründen Sie Ihre Antwort. [2,5 Punkte]

(f) Wird der Vertrag der Methode `filter` *vom Aufgerufenen* erfüllt? Begründen Sie Ihre Antwort. [2,5 Punkte]
Hinweis: Der Vertrag der Methode `checkLessTen` ist hierfür nicht relevant.

Aufgabe 8 (Syntaktische Analyse) [12 Punkte]

Gegeben ist ein Parser, der nach dem aus der Vorlesung bekannten Schema für SLL(1)-Parser konstruiert wurde. Er hat allerdings zwei mit „________“ markierte Lücken (Zeilen 34 und 35). Dort fehlt jeweils ein Token-Typ als `case`-Label.

```
1  void parseS() {
2    parseA();
3    if (token.getType() != sc) {
4      error();
5    }
6    nextToken();
7  }
8
9  void parseA() {
10   switch (token.getType()) {
11     case id:
12       nextToken();
13       parseB();
14       break;
15     case lp:
16       nextToken();
17       parseA();
18       if (token.getType() != rp) {
19         error();
20       }
21       nextToken();
22       break;
23     default:
24       error();
25   }
26 }
27
28 void parseB() {
29   switch (token.getType()) {
30     case ra:
31       nextToken();
32       parseA();
33       break;
34     case ______:
35     case ______:
36       break;
37     default:
38       error();
39   }
40 }
```

Dieser Parser implementiert eine Grammatik G mit Terminalmenge $\Sigma = \{$ **(**, **)** , **id**, **->**, **;** $\}$ und Startsymbol S. Im Code werden für die Terminale folgende Token-Typen verwendet:

`lp`, `rp`	für öffnende bzw. schließende Klammern **(**, **)**
`id`	für **id**
`ra`	für **->**
`sc`	für **;**

(a) Geben Sie die Produktionen von G an. Es genügt *nicht*, die Produktionen irgendeiner Grammatik anzugeben, die die gleiche Sprache wie G akzeptiert. Hinweis: Die Produktionenmenge ist trotz der Lücken im Parser eindeutig. [7 Punkte]

(b) Vervollständigen Sie den Parser nach dem Schema aus der Vorlesung. Tragen Sie dazu in die beiden Lücken jeweils einen Token-Typen ein. Begründen Sie kurz Ihre Auswahl. [3 Punkte]

(c) Geben Sie zwei Wörter in $L(G)$, sowie zwei Wörter in $\Sigma^+ \setminus L(G)$ an. In jedem der Wörter muss jedes Terminal mindestens einmal vorkommen. [2 Punkte]

in $L(G)$: 1.

2.

in $\Sigma^+ \setminus L(G)$: 1.

2.