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

WS17/18, 05. April 2018, 11:00 – 13:00 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	7	
2	18	
3	18	
4	15	
5	12	
6	7	
7	15	
8	8	
9	20	
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.

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) [7 Punkte]

Wenn eine große Anzahl ($> 2^{24}$) von ungefähr gleich großen Float-Werten summiert werden muss, führt das gewöhnliche Addieren (mit `fold` bzw. `sum`) zu starken Rundungs- und Auslöschungsfehlern.[1] Deshalb sollte nach dem Divide-and-Conquer-Prinzip summiert werden.

Schreiben Sie eine Haskell-Funktion `sumDQ: [Float] -> Float`, die eine Liste mit n Elementen summiert, indem für $n > 1$ die Liste in zwei Hälften geteilt und rekursiv summiert wird. Verwenden Sie dazu eine Hilfsfunktion, die zusätzlich die Listenlänge übergeben bekommt.

[1] Denken Sie *nach* der Klausur darüber nach, warum dieses Problem auftritt.

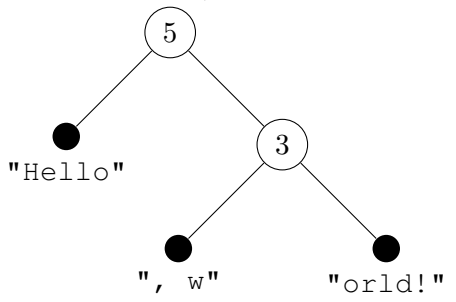
Aufgabe 2 (Haskell: Ropes) [18 Punkte]

Zur Manipulation langer Strings greifen Texteditoren häufig auf *Ropes* zurück. Ein Rope ist ein binärer Baum, der einen String repräsentiert. In den Blättern ist von links nach rechts eine Zerlegung des Strings gespeichert. Innere Knoten sind hingegen mit der Länge des vom linken Teilbaum dargestellten Teilstrings, ihrem *Gewicht*, annotiert.

Nachfolgend sehen Sie eine Datentypdefinition für Ropes in Haskell. Rechts daneben ist eine von mehreren gültigen Darstellungen des Strings `"Hello, world!"` als Rope und ihr zugehöriger Haskell-Ausdruck abgebildet.

```
data Rope
  = Leaf String
  | Inner Rope Int Rope
```

```
Inner (Leaf "Hello") 5
      (Inner (Leaf ", w") 3 (Leaf "orld!"))
```



(a) Implementieren Sie die Funktion [5 Punkte]

```
ropeLength :: Rope -> Int
```

die die Länge der durch das Rope dargestellten Zeichenkette berechnet. Nutzen Sie das Gewicht innerer Knoten, um möglichst wenige Knoten zu besuchen.

(b) Implementieren Sie die Funktion [2 Punkte]

```
ropeConcat :: Rope -> Rope -> Rope
```

die die übergebenen Ropes verkettet. Benutzen Sie `ropeLength` zur Berechnung des Gewichts.

(c) Implementieren Sie die Funktion [11 Punkte]

```
ropeSplitAt :: Int -> Rope -> (Rope, Rope)
```

`ropeSplitAt i r` zerlegt das Rope `r` der Länge n, das den String $c_0...c_{n-1}$ darstellt, an Index `i` in zwei Teilropes: Das erste Teilrope stellt den String $c_0...c_{i-1}$ dar, das zweite Teilrope $c_i...c_{n-1}$. Für Indexwerte außerhalb des Intervalls $[0, n]$ ist die Funktion unspezifiert.

Verwenden Sie das Gewicht innerer Knoten, um die Spaltposition zu finden. Die Listenfunktionen `drop, take :: Int -> [a] -> [a]` sind hilfreich, um die Strings in den Blattknoten zu zerlegen.

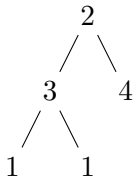
Beispiel[1]:

```
> let (l, r) = ropeSplitAt 6 (fromString "Hello, world!")
> toString l
"Hello,"
> toString r
" world!"
```

[1] Dieses Beispiel verwendet die Funktionen `fromString :: String -> Rope` und `toString :: Rope -> String`, um zwischen Strings und zugehörigen Ropes zu konvertieren. Diese dürfen Sie nicht in Ihrer Lösung verwenden.

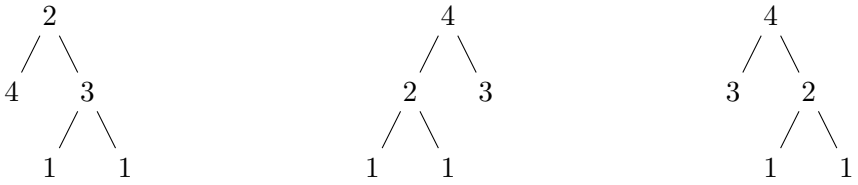
Aufgabe 3 (Prolog, gewichtete Bäume) [18 Punkte]

Es seien in Prolog binäre Bäume rekursiv durch Terme `leaf(W)` und `node(T1, W, T2)` dargestellt, wobei `W` ein ganzzahliges Knotengewicht und `T1`, `T2` Unterbäume sind:




```
node(node(leaf(1), 3, leaf(1)), 2, leaf(4))
```

Wir nennen einen solchen Baum *gewichtsbalanciert*, wenn die Summe der Gewichte auf jedem Pfad von einem Blatt zur Wurzel gleich ist. Der obige Baum ist gewichtsbalanciert, ebenso wie die folgenden Bäume mit den gleichen Gewichten 1, 1, 2, 3, 4:



Ziel dieser Aufgabe ist es, zu einer Liste von Gewichten alle gewichtsbalancierten Bäume nach dem Prinzip *Generate and Test* zu finden.

(a) Implementieren Sie ein Prolog-Prädikat `makeTree(Ws, T)`, das zu einer Liste von Gewichten `Ws` bei Reerfüllung `T` *jeden* binären Baum zuweist, dessen Inorder-Traversierung `Ws` entspricht (d.h. die Gewichte kommen in der Termdarstellung in der gleichen Reihenfolge wie in der Liste vor). [7 Punkte]

Beispiel:

```
?- makeTree([1,2,3,4,5], T).
T = node(leaf(1), 2, node(leaf(3), 4, leaf(5))) ;
T = node(node(leaf(1), 2, leaf(3)), 4, leaf(5)) ;
false.

?- makeTree([], T).
false.
```

Hinweis: Das Prädikat `append(Xs, Ys, XYs)` könnte nützlich sein!

(b) Implementieren Sie ein Prädikat `balanced(T, S)`, das `S` die Summe der Gewichte auf einem Pfad von einem Blatt zur Wurzel von `T` zuweist, wenn diese für jeden solchen Pfad übereinstimmt, und andernfalls fehlschlägt. [7 Punkte]

(c) Implementieren Sie schließlich ein Prädikat `makeBalanced(Ws, T)`, das zu einer Liste von Gewichten `Ws` bei Reerfüllung `T` jeden gewichtsbalancierten Baum zurückgibt, der genau die Gewichte aus `Ws` enthält. [4 Punkte]

Hinweis: Benutzen Sie das aus der Vorlesung bekannte Prädikat `permute(Xs, Ys)`. Sie müssen doppelte Lösungen nicht ausfiltern.

Aufgabe 4 (Typsysteme) [15 Punkte]

(a) Geben Sie zu jedem der vorgegebenen Typen je einen geschlossenen[1] λ-Term an, der mit diesem Typ typisierbar ist. [8 Punkte]

i. $(\alpha \to \gamma \to \delta) \to (\alpha \to \beta \to \gamma) \to \alpha \to \beta \to \delta$

ii. $(\gamma \to \gamma \to \gamma) \to (\alpha \to \gamma) \to (\beta \to \gamma) \to \alpha \to \beta \to \gamma$

iii. $((\alpha \to \beta) \to \gamma) \to \alpha \to \beta \to \gamma$

iv. $(\alpha \to \beta) \to (\alpha \to \gamma) \to (\beta \to \gamma \to \delta) \to \alpha \to \delta$

(b) Gegeben ist der folgende Ausschnitt eines Typherleitungsbaums: [3 Punkte]

$$\textit{Let}\ \frac{\vdash e_1 : \beta \to \alpha \to \alpha \qquad \texttt{f} : \forall\alpha.\forall\beta.\beta \to \alpha \to \alpha \vdash e_2 : \alpha \to \beta \to \gamma \to \beta}{\vdash \textbf{let}\ \texttt{f} = e_1\ \textbf{in}\ e_2 : \alpha \to \beta \to \gamma \to \beta}$$

Geben Sie λ-Terme e_1 und e_2 an, die mit den vorgegebenen Typen typisierbar sind. Dabei soll e_1 geschlossen sein und e_2 mindestens ein freies Vorkommen von `f`, ansonsten aber keine freien Variablen haben.

(c) Gegeben ist der folgende Ausschnitt eines Typherleitungsbaums: [4 Punkte]

$$\textit{Let}\ \frac{\texttt{z} : \alpha, \texttt{x} : \beta \vdash \lambda\texttt{g}.\ \texttt{g}\ \texttt{x} : \tau_f \qquad \texttt{z} : \alpha, \texttt{x} : \beta, \texttt{f} : \tau_f^{poly} \vdash \texttt{f}\ e : \alpha}{\texttt{z} : \alpha, \texttt{x} : \beta \vdash \textbf{let}\ \texttt{f} = \lambda\texttt{g}.\ \texttt{g}\ \texttt{x}\ \textbf{in}\ \texttt{f}\ e : \alpha}$$

i. Geben Sie einen allgemeinsten Typen τ_f aus dem obigen Baumausschnitt an.

ii. Geben Sie die Typabstraktion τ_f^{poly} von τ_f bezüglich der Typumgebung

$$\texttt{z} : \alpha, \texttt{x} : \beta$$

an.

iii. Geben Sie einen für den obigen Baumausschnitt passenden λ-Term e an.

[1] d.h. ohne freie Variablen

Aufgabe 5 (Ein alternativer Fixpunktkombinator) [12 Punkte]

In der Vorlesung waren die Kombinatoren S, K, I definiert als:

$S = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \lambda\mathtt{z}.\ \mathtt{x}\ \mathtt{z}\ (\mathtt{y}\ \mathtt{z}) \qquad K = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x} \qquad I = \lambda\mathtt{x}.\ \mathtt{x}$

Es gilt also (was Sie im Folgenden verwenden dürfen):
$S\ x\ y\ z \Rightarrow^3 \mathtt{x}\ \mathtt{z}\ (\mathtt{y}\ \mathtt{z})$, $K\ x\ y \Rightarrow^2 x$ und $I\ x \Rightarrow x$ für λ-Terme x, y, z

Seien weiterhin die Kombinatoren M, X, Y definiert als:

$M = S\ I\ I \qquad X = S\ (K\ (S\ I)) \qquad Y = M\ (X\ M)$

(a) Zeigen Sie [3 Punkte]

$$M\ a \Rightarrow^* a\ a$$

(b) Zeigen Sie [9 Punkte]

$$Y\ a \Rightarrow^* a\ (Y\ a)$$

Sie dürfen dazu das in (a) Gezeigte verwenden.

Aufgabe 6 (MPI: Skalarprodukt) [7 Punkte]

Gegeben ist der nachfolgende C-Code. Die Methode `scalarProduct` berechnet das Skalarprodukt zweier gegebener Vektoren, dargestellt als Arrays. Der Parameter `length` gibt die Anzahl an Elementen im Array an.

```
double scalarProduct(double a[], double b[], int length) {
    double result = 0;
    for (int i = 0; i < length; i++) {
        result += a[i] * b[i];
    }
    return result;
}
```

Füllen Sie im nachfolgenden Code die Lücken so aus, dass die oben gegebene Funktionalität mittels MPI parallel umgesetzt wird. Verteilen Sie die Daten auf die zur Verfügung stehenden Prozesse und führen Sie die lokalen Ergebnisse anschließend zusammen. Sie können davon ausgehen, dass `length` ohne Rest durch die Anzahl an MPI-Prozessen teilbar ist und die übergebenen Arrays beide die Länge `length` haben. `MPI_Init` wurde vor einem Aufruf von `scalarProduct` bereits aufgerufen. Die Parameter `a` und `b` enthalten nur für den Root-Prozess die zu verarbeitenden Daten.

```
double scalarProduct(double a[], double b[], int length) {
    int rank, procCount;
    [                          ](MPI_COMM_WORLD, &procCount);
    [                          ](MPI_COMM_WORLD, &rank);

    int valuesPerThread = [                          ];
    double localA[valuesPerThread];
    double localB[valuesPerThread];

    [                          ]([                ], [                ], MPI_DOUBLE,
        [                ], valuesPerThread, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    [                          ]([                ], [                ], MPI_DOUBLE,
        [                ], valuesPerThread, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    double localResult = 0;
    for (int i = [                ]; i < [                ]; i++) {
        localResult += [                          ];
    }

    double result = 0;
    [                          ]([                ], [                ], 1,
       MPI_DOUBLE, [                ], 0, MPI_COMM_WORLD);

    return result;
}
```

Aufgabe 7 (Java: Paralleler Filter) [15 Punkte]

Gegeben sei folgende Klasse, die eine Schablonenmethode `filter` bereitstellen soll. Diese Methode soll aus einer übergebenen Liste diejenigen Elemente filtern und zurückgeben, für die die `check`-Methode `true` zurückliefert. Die Liste soll dazu an eine gegebene Anzahl an Threads übergeben und von diesen gefiltert werden. Anschließend werden die gefilterten Listen sequentiell zu der zurückgegebenen Liste zusammengefasst.

```
1  public abstract class ParallelFiltering {
2     public List<Long> filter(ArrayList<Long> values, int threadCount) {
3        List<Future<List<Long>>> futures = new ArrayList<>();
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26       List<Long> result = new LinkedList<Long>();
27       for (Future<List<Long>> future : futures) {
28          result.addAll(future.get());
29       }
30       return result;
31    }
32
33    private List<Long> getFilteredValues(ArrayList<Long> values,
34             int startIndex, int endIndex) {
35       List<Long> result = new LinkedList<Long>();
36       for (int index = startIndex; index < endIndex; index++) {
37          if (check(values.get(index))) {
38             result.add(values.get(index));
39          }
40       }
41       return result;
42    }
43
44    protected abstract boolean check(long value);
```

```
45 }
```

(a) Vervollständigen Sie die Implementierung der `filter`-Methode unter der Verwendung eines `ExecutorService` mit der übergebenen Anzahl an Threads. Verteilen Sie die Liste zur Ausführung der `check`-Operation in nichttrivialer Weise auf die Threads. Sie können dafür die vorgegebene Methode `getFilteredValues` verwenden. Verwenden Sie `Futures`, um die gefilterten Listen zurückzugeben. Sie können *nicht* davon ausgehen, dass die Anzahl der Listenelemente ein Vielfaches der Anzahl an Threads ist. Sie müssen keine Exceptions berücksichtigen. [9 Punkte]

Hinweis: Das Interface `ExecutorService` stellt unter anderem die Methode `Future<T> submit(Callable<T>)` bereit.

(b) Erklären Sie kurz zwei weitere Möglichkeiten, mit denen Sie in Java obiges Problem parallel lösen können. Sie dürfen auch Lösungen angeben, die ohne eine explizite Angabe der Threadanzahl funktionieren. Geben Sie keine Möglichkeiten an, die explizit die Klasse `Thread` verwenden. [4 Punkte]

1.

2.

(c) Ein Kollege stellt die Behauptung auf, dass das Problem mit einer passenden Implementierung auf einer endlichen Anzahl n von Prozessoren mit einem Speedup von n gegenüber der sequentiellen Ausführung gelöst werden kann. Ist diese Aussage korrekt? Begründen Sie Ihre Antwort. [2 Punkte]

Aufgabe 8 (Design by Contract: Konto) [8 Punkte]

Gegeben sei folgende Konto-Klasse, die ein Guthaben verwaltet und einen Kreditrahmen hat, der vom Guthaben nicht unterschritten werden darf. Zur Vereinfachung seien alle Beträge ganzzahlig in Euro. Es ist bereits eine Klasseninvariante, sowie für die Methode `deposit` ein Vertrag mit JML definiert.

```
1  public class Account {
2     private int balance;
3     public final int overdraft;
4
5     /*@ private invariant balance >= -overdraft; */
6
7     public Account(int overdraft) {
8        this.overdraft = overdraft;
9        this.balance = 0;
10    }
11
12    public int getBalance() {
13       return balance;
14    }
15
16    /*@ requires amount > 0;
17      @ ensures \old(getBalance()) == getBalance() - amount;
18     */
19    public void deposit(int amount) {
20       // Einzahlungen nur bis 1000 Euro erlaubt
21       if (amount <= 1000) {
22          this.balance += amount;
23       }
24    }
25
26    public void withdraw(int amount) {
27       this.balance -= amount;
28    }
29 }
```

(a) Geben Sie alle Fehler im Vertrag der Methode `deposit`, sowie sinnvolle Korrekturen dafür an. [3 Punkte]

(b) Betrachten Sie folgende Verwendung der `Account`-Klasse. Wird der *ursprüngliche* Vertrag, ohne Ihre Korrekturvorschläge aus dem vorigen Aufgabenteil, bei dieser Verwendung vom Aufrufer verletzt? Begründen Sie ihre Antwort kurz. [2 Punkte]

```
Account account = new Account(200);
account.deposit(1100);
```

(c) Schreiben Sie unter Verwendung von JML Vor- und Nachbedingungen für die `withdraw`-Methode, die sicherstellen, dass nur positive Beträge abgehoben werden können, und dass danach der Kreditrahmen nicht verletzt ist. [3 Punkte]

Aufgabe 9 (Syntaktische Analyse) [20 Punkte]

Gegeben sei die folgende Grammatik für einen Teil der Prädikatenlogik erster Stufe mit Startsymbol *Formula*.

$$
\begin{aligned}
\textit{Formula} &\rightarrow \texttt{quantor var .}\ \textit{Formula} \mid \texttt{var (}\ \textit{Termlist}\ \texttt{)} \\
\textit{Termlist} &\rightarrow \textit{Term} \mid \textit{Term}\ \texttt{,}\ \textit{Termlist} \\
\textit{Term} &\rightarrow \texttt{var} \mid \texttt{int}
\end{aligned}
$$

(a) Geben Sie eine Linksfaktorisierung an, sodass die Grammatik SLL(1) ist. [3 Punkte]
Es genügt, geänderte bzw. neue Zeilen der Grammatik anzugeben.

(b) Geben Sie für jedes Nichtterminal einen rekursiven Abstiegsparser – ohne Erzeugung eines abstrakten Syntaxbaums – in Pseudocode an. [17 Punkte]
Dabei stehen Ihnen folgende Token-Typen zur Verfügung:

quantor	für einen Quantor **quantor**
dot	für den Punkt **.**
comma	für das Komma **,**
var	für Variablen- und Prädikatnamen **var**
int	für Zahlen **int**
lp, rp	für öffnende bzw. schließende Klammern **(**, **)**

Die globale Variable `token` enthält immer das aktuelle Token. Es besitzt die Methoden `getType()` für den Token-Typ (quantor, dot, comma, var, int, lp, rp). Die globale Methode `nextToken()` fordert das nächste Token an. Brechen Sie bei Parsefehlern durch Aufruf der globalen `error()`-Methode ohne Fehlermeldung ab.