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

SS17, 25. September 2017, 14:00 – 16:00 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	22	
2	7	
3	7	
4	12	
5	19	
6	15	
7	8	
8	8	
9	8	
10	6	
11	3	
12	5	
Σ	**120**	
⌈Σ⌉	**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, Semimagische Quadrate) [22 Punkte]

Eine Anordnung der Zahlen $1, 2, \ldots n^2$ auf einem $n \times n$ großen Feld nennt man *semimagisches Quadrat*, wenn es eine Zahl m_n gibt, so dass

- die Summe der Zahlen in jeder Zeile jeweils m_n ergibt, und
- die Summe der Zahlen in jeder Spalte jeweils m_n ergibt.

n=3, m_3=15

6	1	8
2	9	4
7	5	3

```
q::[[Int]]
q=[[6,1,8],
   [2,9,4],
   [7,5,3]]
```

Die Zahl $m_n \in \mathbb{N}$ wird *magische Zahl* genannt. Sie ist lediglich von n abhängig und kann mit der Formel $m_n = \left(\sum_{i=1}^{n^2} i\right) / n$ berechnet werden.

(a) Geben Sie eine Funktion `magicNumber :: Int -> Int` an, so dass `magicNumber n` die Zahl m_n berechnet. [2 Punkte]

(b) In einem semimagischen Quadrat darf keine Zahl doppelt vorkommen. Schreiben Sie daher eine Funktion `duplicates`, für die `duplicates xs = True` gilt gdw. in der Liste `xs` mindestens ein Element doppelt vorkommt.
Geben Sie einen möglichst allgemeinen Typ für `duplicates` an. [5 Punkte]

(c) Geben Sie eine Funktion `transpose :: [[a]] -> [[a]]` an, welche eine Liste von Listen `xs :: [[a]]` transponiert.
Es soll z.B. gelten `transpose [[1,2],[3,4]] = [[1,3],[2,4]]`
Hinweis: Sie dürfen annehmen, dass `xs` nur Listen gleicher Länge enthält. [10 Punkte]

(d) Geben Sie ein Prädikat `isMagic :: [[Int]] -> Bool` an, welches prüft, ob das gegebene Argument (eine Liste von Zeilen) ein semimagisches Quadrat ist. Sie dürfen die Funktion `all :: (a -> Bool) -> [a] -> Bool` verwenden, wobei `all f xs` angibt, ob das Prädikat `f` für alle Elemente der Liste `xs` erfüllt ist. [5 Punkte]
Beispiele:

```
> isMagic q
True
> isMagic [[1, 1], [1, 1]]
False
```

Aufgabe 2 (Haskell, Bäume) [7 Punkte]

Der Datentyp für Mehrweg-Bäume mit toten/lebendigen Blättern ist definiert als:

```
data Tree  = Node [Tree] | Leaf State                    deriving (Show, Eq)
data State = Dead | Alive                                deriving (Show, Eq)
```

Ein Baum ist lebendig, wenn er ein lebendiges Blatt besitzt. Andernfalls ist er tot.

Geben Sie eine Funktion

```
prune :: Tree -> Tree
```

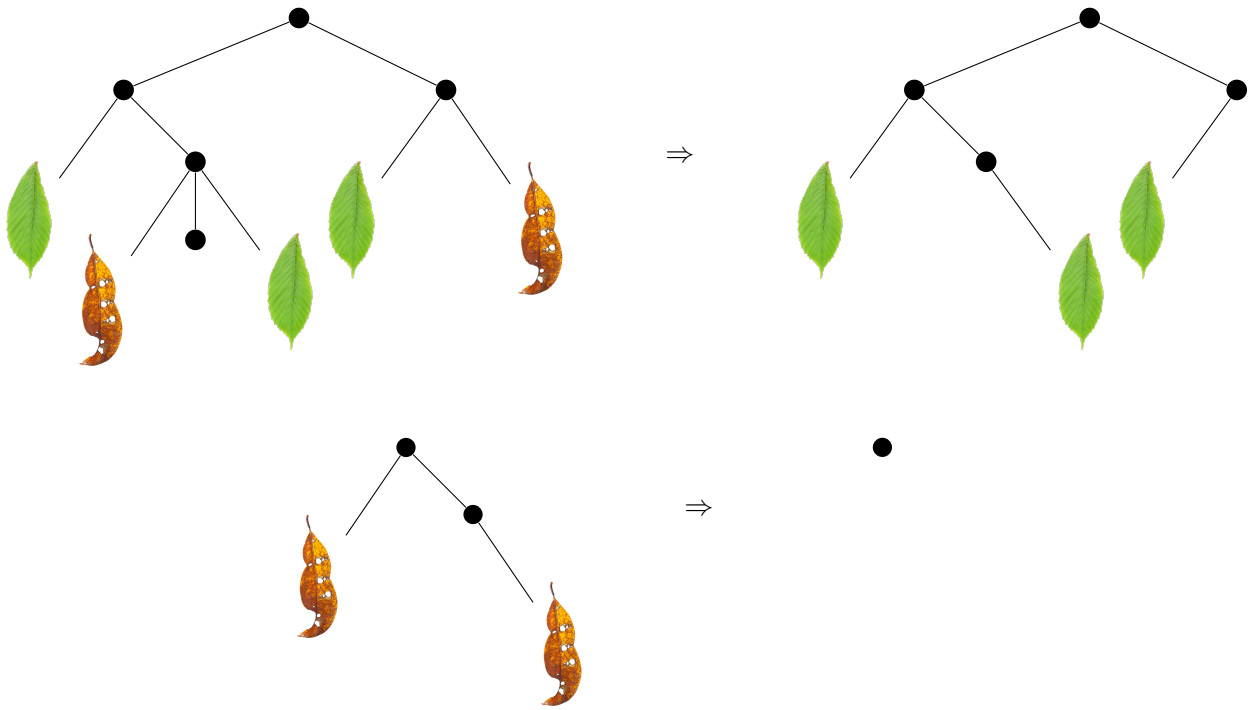
an, die alle toten Teilbäume „entfernt“

Hinweis: Verwenden Sie List-Comprehensions, oder **`filter`**, **`map`**.

Beispiele:

```
> tree1
Node [Node [Leaf Alive, Node [Leaf Dead,Node [],Leaf Alive]], Node [Leaf Alive, Leaf Dead]]
> prune tree1
Node [Node [Leaf Alive, Node [                  Leaf Alive]], Node [Leaf Alive           ]]

> prune (Node [ Leaf Dead, Node [Leaf Dead]])
Node []
```



Aufgabe 3 (Prolog, freie Variablen) [7 Punkte]

In Prolog können Formeln der Sprache[1]

$$\begin{array}{rcl} \textit{Formula} & := & \mathsf{p}\,(\textit{Term}) \mid \mathsf{q}\,(\textit{Term}, \textit{Term}) \\ & \mid & \forall\ \textit{Variable}.\ \textit{Formula} \\ & \mid & \exists\ \textit{Variable}.\ \textit{Formula} \end{array} \qquad \begin{array}{rcl} \textit{Integer} & := & \ldots \mid \mathsf{-1} \mid \mathsf{0} \mid \mathsf{1} \mid \ldots \\ \textit{Variable} & := & \mathsf{x} \mid \mathsf{y} \mid \ldots \\ \textit{Term} & := & \textit{Variable} \mid \textit{Integer} \end{array}$$

als Terme `42`, `x`, `p(T)`, `q(T1,T2)`, `exists(X,T)`, `forall(X,T)`, `...` dargestellt werden, z.B.

$$\forall\, \mathsf{x}.\ \exists\, \mathsf{y}.\ \mathsf{q}\,(\mathsf{x},\mathsf{y}) \quad \text{als} \quad \texttt{forall(x, exists(y, q(x,y)))}$$

Definieren Sie ein Prädikat `hasfree(BoundVars, Term)` das feststellt, ob ein solcher Term freie Variablen enthält! Da `forall` und `exists` Bindungskonstrukte sind, muss dieses Prädikat eine Liste der „außerhalb“ gebundenen Variablen bekommen. **Beispiel:**

```
? hasfree([], forall(x,q(42,x))).
  false.

? hasfree([], forall(x,p(y))).
  true.
```

Hinweis: Verwenden Sie **`atom`**`(X)`.

[1] hier handelt es sich um eine vereinfachte PL1-Sprache mit den Prädikatsymbolen p und q

Aufgabe 4 (Unifikation) [12 Punkte]

Gegeben sei die einelementige Menge von Gleichungen $C = \{$

$$\begin{aligned} & \texttt{node}(\texttt{node}(\texttt{T}, 2, \texttt{leaf}), 4, \texttt{R}) \\ = \; & \texttt{node}(\texttt{node}(\texttt{node}(\texttt{R}, 1, \texttt{leaf}), \texttt{V}, \texttt{T}), 4, \texttt{leaf}) \end{aligned}$$

$\}$

über den Variablen $\texttt{T}, \texttt{R}$ und $\texttt{V}$.

Führen Sie den Unifikationsalgorithmus nach Robinson (siehe Skript S. 293) zur Berechnung von $\texttt{unify}(C)$ aus. Geben Sie bei jedem rekursiven Aufruf von `unify` die erzeugte Substitution sowie die noch zu unifizierende Menge an.

Aufgabe 5 (Typsysteme) [19 Punkte]

(a) Geben Sie einen λ-Term t vom Typ [3 Punkte]

$$\tau_t = ((\alpha \to \beta) \to (\beta \to \gamma)) \to (\alpha \to \beta) \to \alpha \to \gamma$$

an.

(b) Im Folgenden betrachten wir den λ-Ausdruck [16 Punkte]

$$\texttt{let b = t } (\lambda \texttt{x. x}) \texttt{ in } \lambda \texttt{y. } \lambda \texttt{z. (b y) (b z)}$$

i. Geben Sie den polymorphen Typen $\tau_{\texttt{b}}^{poly}$ von b unter der Typannahme $\Gamma_{\texttt{t}} = \texttt{t} : \forall\alpha.\, \forall\beta.\, \forall\gamma.\, \tau_t$ an! (3 Punkte)

ii. Es seien

$$\Gamma_{\texttt{tb}} = \Gamma_{\texttt{t}}, \texttt{b} : \tau_{\texttt{b}}^{poly}$$
$$\Gamma_{\texttt{tby}} = \Gamma_{\texttt{tb}}, \texttt{y} : \alpha_2$$
$$\Gamma_{\texttt{tbyz}} = \Gamma_{\texttt{tby}}, \texttt{z} : \alpha_4$$

Unten sehen Sie einen Herleitungsbaum für einen allgemeinsten Typen von λy. λz. (b y) (b z) unter der Typannahme $\Gamma_{\texttt{tb}}$. Geben Sie das zugehörige Typgleichungssystem an und ergänzen Sie außerdem, was an den mit (A) bzw. (B) markierten Stellen einzutragen ist. (11 Punkte)

$$\textit{Abs}\ \frac{\textit{Abs}\ \dfrac{\textit{App}\ \dfrac{\textit{App}\ \dfrac{\textit{Var}\ \dfrac{\text{(A)}}{\Gamma_{\texttt{tbyz}} \vdash \texttt{b} : \alpha_7} \quad \textit{Var}\ \dfrac{\Gamma_{\texttt{tbyz}}(\texttt{y}) = \alpha_8}{\Gamma_{\texttt{tbyz}} \vdash \texttt{y} : \alpha_8}}{\Gamma_{\texttt{tbyz}} \vdash \texttt{b y} : \alpha_6} \quad \textit{App}\ \dfrac{\textit{Var}\ \dfrac{\text{(B)}}{\Gamma_{\texttt{tbyz}} \vdash \texttt{b} : \alpha_{10}} \quad \textit{Var}\ \dfrac{\Gamma_{\texttt{tbyz}}(\texttt{z}) = \alpha_{11}}{\Gamma_{\texttt{tbyz}} \vdash \texttt{z} : \alpha_{11}}}{\Gamma_{\texttt{tbyz}} \vdash (\texttt{b z}) : \alpha_9}}{\Gamma_{\texttt{tbyz}} \vdash (\texttt{b y})\ (\texttt{b z}) : \alpha_5}}{\Gamma_{\texttt{tby}} \vdash \lambda\texttt{z. (b y) (b z)} : \alpha_3}}{\Gamma_{\texttt{tb}} \vdash \lambda\texttt{y. } \lambda\texttt{z. (b y) (b z)} : \alpha_1}$$

iii. Geben Sie einen allgemeinsten Typen von λy. λz. (b y) (b z) unter der Typannahme $\Gamma_{\texttt{tb}}$ an! (2 Punkte)

Aufgabe 6 (Gleichheit auf Church-Zahlen) [15 Punkte]

In der Vorlesung haben Sie bereits Kodierungen im λ-Kalkül für boolsche Werte und die natürlichen Zahlen gesehen. isZero testet auf 0, succ bestimmt den Nachfolger.

Erinnerung:

$\mathsf{c}_{\mathsf{true}} = \lambda\mathtt{t}.\ \lambda\mathtt{f}.\ \mathtt{t}$ $\quad$ $\mathsf{c}_{\mathsf{false}} = \lambda\mathtt{t}.\ \lambda\mathtt{f}.\ \mathtt{f}$

$\mathsf{isZero}\ \mathsf{c}_0 \Rightarrow^* \mathsf{c}_{\mathsf{true}}$ $\quad$ $\mathsf{isZero}\ \mathsf{c}_n \Rightarrow^* \mathsf{c}_{\mathsf{false}}$ für alle $n > 0$ $\quad$ $\mathsf{succ}\ \mathsf{c}_n \Rightarrow^* \mathsf{c}_{n+1}$ für alle n

Ziel dieser Aufgabe ist es, einen λ-Ausdruck anzugeben, der für zwei Church-Zahlen c_n und c_m entscheidet, ob $n = m$.

Gebeben seien folgende λ-Ausdrücke:

$$\varphi = (\lambda\mathtt{x}.\ \lambda\mathtt{f}.\ \mathtt{f}\ (\mathtt{x}\ \mathsf{c}_0)\ (\mathsf{succ}\ (\mathtt{x}\ \mathsf{c}_0)))$$

$$\mathsf{pred} = \lambda\mathtt{n}.\ \mathtt{n}\ \varphi\ (\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_0\ \mathsf{c}_0)\ \mathsf{c}_{\mathsf{true}}$$

(a) Zeigen Sie durch β-Reduktion exemplarisch für c_2, dass pred die Vorgängerfunktion auf Church-Zahlen realisiert: [7 Punkte]

$$\mathsf{pred}\ \mathsf{c}_2 \Rightarrow^* \mathsf{c}_1$$

Hinweis: Reduzieren Sie geschachtelte φ-Anwendungen "von innen nach außen".
Für alle λ-Ausdrücke a, b gilt: $\mathsf{c}_0\ \mathsf{a}\ \mathsf{b} \Rightarrow^* \mathsf{b}$.

(b) Geben Sie die Rekursionsgleichung für einen λ-Ausdruck eq an, welcher folgende Eigenschaften erfüllt: [5 Punkte]

i. $\mathsf{eq}\ \mathsf{c}_m\ \mathsf{c}_m \Rightarrow^* \mathsf{c}_{\mathsf{true}}$

ii. $\mathsf{eq}\ \mathsf{c}_n\ \mathsf{c}_m \Rightarrow^* \mathsf{c}_{\mathsf{false}}$ für alle n, m mit $n \neq m$.

Hinweis: Schreiben Sie sich die notwendige Fallunterscheidung zuerst auf und benutzen Sie die in der Vorlesung bekannte Umsetzung von `if _ then _ else _`.

(c) Geben Sie nun das dazugehörige Funktional EQ, und die Definition von eq an. [3 Punkte]

Hinweis: Verwenden Sie den Fixpunktkombinator Y.

Aufgabe 7 (MPI) [8 Punkte]

Gegeben sei die unten stehende Methode, welche in einem MPI-Programm mit einer beliebigen Anzahl an Prozessen aufgerufen wird. Sie können davon ausgehen, dass die Initialisierung und Finalisierung von MPI vom Aufrufer übernommen wird. Weiterhin können Sie davon ausgehen, dass `elementCount` immer ein Vielfaches der Anzahl verfügbarer Prozesse ist.

```
1  void operate(int elementCount, int elements[elementCount]) {
2      int rank;
3      int processCount;
4      int rootRank = 0;
5
6      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
7      MPI_Comm_size(MPI_COMM_WORLD, &processCount);
8
9      int elementsPerProcess = elementCount / processCount;
10     int local[elementsPerProcess];
11     MPI_Scatter(elements, elementsPerProcess, MPI_INT, local,
12         elementsPerProcess, MPI_INT, 0, MPI_COMM_WORLD);
13
14     int offset = 0;
15     if (rank == rootRank) {
16         for (int i = 0; i < elementCount; i++) {
17             if (elements[i] < offset) offset = elements[i];
18         }
19     }
20
21     if (rank == rootRank) {
22         for (int process = 0; process < processCount; process++) {
23             if (process != rootRank) {
24                 MPI_Send(&offset, 1, MPI_INT, process, 0, MPI_COMM_WORLD);
25             }
26         }
27     } else {
28         MPI_Recv(&offset, 1, MPI_INT, rootRank, 0, MPI_COMM_WORLD, NULL);
29     }
30
31     for (int i = 0; i < elementsPerProcess; i++) {
32         local[i] = local[i] - offset;
33     }
34
35     MPI_Gather(local, elementsPerProcess, MPI_INT, elements,
36         elementsPerProcess, MPI_INT, rootRank, MPI_COMM_WORLD);
37
38     if (rank == rootRank) {
39         for (int i = 0; i < elementCount; i++) {
40             printf("%i␣", elements[i]);
41         }
42     }
43 }
```

(a) Erklären Sie, welche Funktion das Programm erfüllt und geben Sie die Ausgabe für folgende Eingabe an: `(4, [-1, 2, 9, 5])` [3 Punkte]

(b) Schreiben Sie einen Ersatz für die Zeilen 21 bis 29, der ausschließlich aus dem Aufruf einer MPI-Operation besteht. [2 Punkte]

(c) Die Zeilen 15 bis 19 werden rein sequentiell ausgeführt. Erläutern Sie die notwendigen Schritte, um diese Zeilen mittels MPI möglichst effizient parallelisieren können. Geben Sie dazu notwendige MPI-Operationen explizit an.
Hinweis: Sie müssen nicht (dürfen aber) den endgültigen Programmcode angeben, eine textuelle Erläuterung genügt. [3 Punkte]

Aufgabe 8 (Aktoren und Parallelisierbarkeit) [8 Punkte]

Es soll eine Lastverteilung für beliebige Anfragen (beispielsweise Webseiten-Aufrufe) mittels Akka-Aktoren implementiert werden. Dazu sei bereits eine Klasse `Worker` als Aktor geben, welche eine einzelne Anfrage verarbeiten kann. Der Aktor `Balancer` soll eingehende Anfragen auf eine feste Anzahl (`workerCount`) an `Worker`-Instanzen verteilen.

```
1  public class Worker extends UntypedActor {
2      @Override
3      public void onReceive(Object query) {
4          // Handle query
5      }
6  }
7
8  public class Balancer extends UntypedActor {
9      private final int workerCount;
10     private List<ActorRef> worker;
11
12
13     public Balancer(int workerCount) {
14         if (workerCount < 1) {
15             throw new IllegalArgumentException();
16         }
17         this.workerCount = workerCount;
18     }
19
20     private ActorRef createWorker() {
21         return getContext().actorOf(Props.create(Worker.class));
22     }
23
24     @Override
25     public void preStart() throws Exception {
26         super.preStart();
27         this.worker = new ArrayList<>();
28
29
30
31
32
33
34     }
35
36     @Override
37     public void onReceive(Object query) {
38
39
40
41
42
43
44     }
45 }
```

(a) Ergänzen Sie die Klasse `Balancer` durch Ausfüllen der Lücken im gegebenen Quellcode, sodass in `preStart` die `Worker`-Instanzen angelegt werden und in `onReceive` die Nachricht zu einer der `Worker`-Instanzen weitergeleitet wird. Die Anfragen sollen dabei auf alle `Worker` gleichverteilt werden (beispielsweise durch zyklische Zuweisung). [4,5 Punkte]
`Hinweis:` Sie dürfen auch außerhalb der Methoden Ergänzungen an der Klasse vornehmen.

(b) Gegeben seien folgende Voraussetzungen: [3,5 Punkte]

- Die Abarbeitung einer Anfrage durch einen `Worker` benötigt viermal soviel Zeit, wie das Verteilen einer Anfrage durch den `Balancer` auf einen `Worker`.
- Eine beliebig hohe Last (Anzahl von Anfragen pro Zeiteinheit) liegt an.

Betrachten Sie folgende zwei Szenarien:

1. Es gibt einen `Worker` (`workerCount == 1`) und nur einen (Ein-Kern-)Prozessor, auf dem sowohl der `Balancer` als auch der `Worker` ausgeführt werden.
2. Es gibt beliebig viele `Worker` (`workerCount` beliebig hoch) und beliebig viele Prozessoren, auf die die `Worker` und der `Balancer` verteilt werden.

Wie groß ist der Unterschied im Durchsatz der Anwendung (verarbeitete Anfragen pro Zeiteinheit) zwischen den beiden Szenarien? Erläutern Sie Ihre Antwort.

Aufgabe 9 (Java) [8 Punkte]

Gegeben seien folgende Fragmente einer an Petri-Netze angelehnten Implementierung. Sie besteht aus dem Interface `Token` und der Klasse `Place`, welche mit `Token`s und einer Kapazität initialisiert werden kann. Die Methode `moveTokenTo(Place)` verschiebt ein beliebiges `Token` des `this`-`Place` zu dem als Argument übergebenen `Place`. Die dafür benötigen Methoden `putToken(Token)` und `takeAnyToken()` müssen noch implementiert werden.

```
1  public interface Token {}
2
3  public class Place {
4      private final List<Token> tokens;
5      private final int capacity;
6
7      public Place(Collection<Token> initialTokens, int capacity) {
8          this.tokens = new ArrayList<>();
9          this.tokens.addAll(initialTokens);
10         this.capacity = capacity;
11     }
12
13     /**
14      * Bewegt ein Token von diesem in den übergebenen Place.
15      * Das Token kann sich in einem Übergangszustand befinden,
16      * in dem es weder in diesem noch in dem Ziel-Place vorhanden ist.
17      */
18     public void moveTokenTo(Place otherPlace) {
19         Token token = this.takeAnyToken();
20         otherPlace.putToken(token);
21     }
22
23     /**
24      * Fügt der "tokens"-Liste das übergebene Token hinzu.
25      * Falls die Kapazität der Liste aktuell erschöpft ist, wartet
26      * die Methode, bis ein Token entfernt wurde.
27      */
28     private void putToken(Token token) {
29
30
31
32
33
34
35
36
37
38
39
40
41     }
42
```

```
43      /**
44       * Gibt ein beliebiges Token aus der "tokens"-Liste zurück.
45       * Falls aktuell kein Token vorhanden ist, wartet die Methode,
46       * bis ein Token hinzugefügt wurde.
47       */
48      private Token takeAnyToken() {
49
50
51
52
53
54
55
56
57
58
59
60
61      }
62
63  }
```

(a) Implementieren Sie die Methoden `putToken(Token)` und `takeAnyToken()` entsprechend der Dokumentation durch Ausfüllen der Lücken im gegebenen Quellcode. Verwenden Sie Parallelisierungskonstrukte von Java, sodass die Methode `moveTokenTo(Place)` nebenläufig aufgerufen werden kann, ohne dass es zu einem Wettlauf kommen kann. [5 Punkte]

Hinweis: Das Interface `List<E>` stellt unter anderem folgende Methoden bereit:
- `add(E element)`
- `remove(int index): E`
- `size(): int`

(b) Angenommen die Methoden `moveTokenTo(Place)`, `putToken(Token)` und `takeAnyToken()` wären alle mit dem Keyword `synchronized` versehen. Könnte es dann beim Aufruf der Methode `moveTokenTo(Place)` zu einem Deadlock kommen? Begründen Sie Ihre Antwort mithilfe der Coffman-Bedingungen genau. Die Angabe, dass eine Bedingung erfüllt bzw. nicht erfüllt ist, ist nicht ausreichend. [3 Punkte]

Aufgabe 10 (Design by Contract) [6 Punkte]

Für eine Software soll Funktionalität zum Selektieren verschiedener Optionen erstellt werden. Hierzu wurde in Java ein Interface `Option` für die Optionen, sowie eine Klasse `Selection` mit Methoden für das Anwählen (`select`) und Abwählen (`deselect`) von Optionen entwickelt. Die Methoden wurden teilweise bereits mit Verträgen im Javadoc versehen, mit denen ihre Vor- und Nachbedingungen beschrieben werden. Die Vor- und Nachbedingungen sind als Java-Ausdrücke definiert. Zusätzlich wird mittels `\old(<expression>)` der Wert von `<expression>` vor dem Aufruf der Methode referenziert.

```
1  public interface Option {}
2
3  public class Selection {
4      private Set<Option> options = new HashSet<>();
5
6      /**
7       * Vorbedingungen:
8       *   1. option != null;
9       *
10      * Nachbedingungen:
11      *   1. options.size() == \old(options).size() + 1;
12      *   2. options.contains(option);
13      */
14     public void select(Option option) {
15         this.options.add(option);
16     }
17
18     /**
19      * Vorbedingungen:
20      *   1. option != null;
21      */
22     public void deselect(Option option) {
23         this.options.remove(option);
24     }
25 }
```

(a) Erfüllt die Methode `select` die im Javadoc spezifizierten Nachbedingungen? Begründen Sie Ihre Antwort. [2,5 Punkte]

(b) Geben Sie Nachbedingungen für die Methode `deselect` an, die das folgende, informell spezifizierte Verhalten garantieren: [3,5 Punkte]
Die Methode `deselect` entfernt die übergebene `Option` aus den `options`, falls sie darin vorhanden ist, und hat ansonsten keine weiteren Effekte.
Hinweis: Das Interface `Set<E>` stellt unter anderem folgende Methoden bereit:

- `containsAll(Collection<E> elements): boolean`
- `contains(E element): boolean`
- `size(): int`

Aufgabe 11 (First_1- und Follow_1-Mengen) [3 Punkte]

Gegeben sei die folgende Produktion einer Grammatik G:

$$S \quad \rightarrow \quad AB$$

Es gilt

$$\text{First}_1(A) \quad \subseteq \quad \text{First}_1(S).$$

Geben Sie zwei weitere gültige Teilmengenbeziehung zwischen den Mengen

$$\begin{array}{ccc} \text{First}_1(S) & \text{First}_1(A) & \text{First}_1(B) \\ \text{Follow}_1(S) & \text{Follow}_1(A) & \text{Follow}_1(B) \end{array}$$

an. Triviale Teilmengenbeziehungen, wie $\text{First}_1(S) \subseteq \text{First}_1(S)$, sind dabei ausgeschlossen.

Beachten Sie, dass die Grammatik G weitere Produktionen mit den Nichtterminalen S, A und B enthalten kann.

Aufgabe 12 (Java-Bytecode) [5 Punkte]

Gegeben sei folgender Java-Bytecode einer Methode `f`:

```
public void f(int a, int b):
  0: iconst_0
  1: istore_3
  2: iload_1
  3: iload_2
  4: if_icmplt     17
  7: iload_1
  8: iload_2
  9: isub
 10: istore_1
 11: iinc          3, 1
 14: goto          2
 17: aload_0
 18: iload_3
 19: putfield      x:I
 22: aload_0
 23: iload_1
 24: putfield      y:I
 27: return
```

Rekonstruieren Sie ein gültiges Java-Programm, für das dieser Bytecode erzeugt wird. Ergänzen Sie dazu in der rechts angegebenen Klassendefinition den Rumpf von `f`.

Sie dürfen dabei folgende Annahmen machen:

- `a` und `b` sind als lokale Variablen 1 und 2 verfügbar.
- Die lokale Variable 3 heißt `c`.

```
1  class C {
2      int x;
3      int y;
4
5      public void f(int a, int b) {
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40     }
41 }
```