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 — Beispiellösung

SS20, 06. Oktober 2020, 09:00 – 11:00 Uhr

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

Bearbeitungszeit: 120 min

Aufgabe 1 (Haskell: Socken sortieren) [12 Punkte]

Angenommen, Sie haben gerade Ihre Socken gewaschen, die nun unsortiert vor Ihnen liegen. Um sie wieder zu Paaren zusammenzustellen, verwenden Sie folgenden Algorithmus:

1. Räume eine Ablagefläche frei
2. Für jede Socke:
   - Falls eine gleiche Socke auf der Ablagefläche liegt, bilde mit dieser ein Sockenpaar und räume es weg
   - Andernfalls lege die Socke auf die Ablagefläche

Wie groß sollte die Ablagefläche sein? In dieser Aufgabe sollen Sie das mit einer Simulation in Haskell abschätzen. Socken sind vom Typ `Sock`, der die Typklasse `Eq` implementiert.

(a) Implementieren Sie eine Funktion [7 Punkte]

```
maxStorage :: [Sock] -> Int
```

Der Aufruf `maxStorage socks` soll den obenstehenden Algorithmus simulieren, wenn die Liste `socks` von vorne nach hinten abgearbeitet wird. Das Ergebnis soll die maximale Anzahl Socken sein, die während der Ausführung auf der Ablagefläche liegen.

Sie dürfen hierbei zusätzlich die Funktion `delete :: (Eq a) => a -> [a] -> [a]` benutzen: Der Aufruf `delete x list` gibt die Liste `list` ohne das erste Vorkommen von `x` zurück.

Hinweis: Neben `delete` könnte auch `elem` nützlich sein.

Beispiellösung:

```
maxStorage :: [Sock] -> Int
maxStorage socks = maximum (run socks [])
    where
      run :: [Sock] -> [Sock] -> [Int]
      -- Geht man davon aus, dass alle Socken zu Paaren zusammenpassen,
      -- reicht hier auch: run [] storage = []
      run [] storage = [length storage]
      run (s:socks) storage = length storage : nextStep
          where
            nextStep | s `elem` storage = run socks (delete s storage)
                     | otherwise        = run socks (s:storage)
```

Nun geht es darum, eine zufällige Eingabeliste zu erzeugen.

(b) Implementieren Sie eine Funktion [2 Punkte]

```
swap :: Int -> Int -> [a] -> [a]
```

Der Aufruf `swap i j list` soll `list` zurückgeben, wobei die Elemente an den Indizes `i` und `j` vertauscht sind.

Sie dürfen hierbei zusätzlich die Funktion `setAt :: Int -> a -> [a] -> [a]` benutzen: Der Aufruf `setAt i x list` gibt `list` zurück, wobei das `i`-te Element auf `x` gesetzt ist.

Beispiellösung:

```
swap :: Int -> Int -> [a] -> [a]
swap i j list = setAt i (list !! j) (setAt j (list !! i) list)
```

(c) Implementieren Sie eine Funktion [3 Punkte]

```
shuffle :: [(Int, Int)] -> [a] -> [a]
```

Das erste Argument von `shuffle` ist eine Liste von Paaren von Indizes, wobei jedes Paar eine Vertauschung codiert. `shuffle` soll auf seinem zweiten Argument diese Vertauschungen nacheinander ausführen und das Ergebnis zurückgeben.

Beispiellösung:

```
shuffle :: [(Int, Int)] -> [a] -> [a]
shuffle []            list = list
shuffle ((i, j):rest) list = shuffle rest (swap i j list)
```

Aufgabe 2 (Haskell: Pattern Matching) [18 Punkte]

In dieser Aufgabe sollen Sie Pattern Matching für binäre Bäume implementieren. Sowohl Patterns als auch die gematchten Bäume haben eine eigene Haskell-Datenstruktur:

```
data Tree                data Pattern
  = Leaf                   = PLeaf
  | Node Tree Tree         | PNode Pattern Pattern
                           | PVar String -- Variablenname als String
```

Dabei kommt keine Variable in einem `Pattern` mehrfach vor.
Durch das Pattern Matching ergibt sich eine Variablenbelegung in Form einer Assoziativliste:

```
type Subst = [(String, Tree)]
```

Beispielsweise bedeutet `[("a", Leaf), ("b", Node Leaf Leaf)]`, dass Variable `"a"` mit `Leaf` belegt ist und `"b"` mit `Node Leaf Leaf`.

(a) Implementieren Sie die Funktion [4 Punkte]

```
subst :: Subst -> Pattern -> Tree
```

`subst sub pat` substituiert alle Variablenvorkommen in `pat` durch ihre Belegung in `sub`. Die restliche Baumstruktur bleibt erhalten. Sie dürfen annehmen, dass alle in `pat` vorkommenden Variablen eine Belegung in `sub` haben.
Hinweis: `lookup :: Eq a => a -> [(a, b)] -> Maybe b` aus der Prelude könnte im Umgang mit `sub` nützlich sein.

(b) Implementieren Sie die Funktion [2 Punkte]

```
liftMaybe2 :: (a -> b -> c) -> Maybe a -> Maybe b -> Maybe c
```

`liftMaybe2 f ma mb` soll `Just c` mit geeignetem `c` zurückgeben, falls `ma` und `mb` auf `Just _` matchen, ansonsten `Nothing`.

(c) Implementieren Sie die Funktion [6 Punkte]

```
match :: Pattern -> Tree -> Maybe Subst
```

Der Aufruf `match pat tree` matcht `tree` gegen das Muster `pat`. Ist der Match erfolgreich, wird `Just sub` zurückgegeben, wobei `sub` die entstehende Variablenbelegung ist. Ansonsten ist das Ergebnis `Nothing`.

`PLeaf` matcht nur für `Leaf`. `PNode p1 p2` matcht nur für `Node n1 n2`, in welchem Fall `n1` auf `p1` und `n2` auf `p2` matchen muss. Die beiden Variablenbelegungen werden konkateniert. Jeder `tree` matcht auf `PVar v`, in welchem Fall `v` in `sub` mit `tree` belegt ist.
Hinweis: `liftMaybe2` könnte nützlich sein.

Beispiele:

```
> match PLeaf Leaf
Just []
> match (PNode (PVar "a") PLeaf) (Node Leaf Leaf)
Just [("a", Leaf)]
> match PLeaf (Node Leaf Leaf)
Nothing
```

(d) Implementieren Sie die Funktion [6 Punkte]

```
matchFirst :: [(Pattern, Subst -> a)] -> Tree -> a
```

`matchFirst alts tree` sucht die erste Alternative `(pat, f)` aus `alts`, deren Pattern `pat` `tree` matcht. Für die Rückgabe wird dann `f` auf die entstandene Variablenbelegung angewendet. Sie können davon ausgehen, dass immer eine passende Alternative existiert.

Beispiele:

```
> alts = [(PLeaf, f), (PNode (PVar "a") PLeaf, g), (PVar "b", h)]
> matchFirst alts Leaf
f []
> matchFirst alts (Node Leaf Leaf)
g [("a", Leaf)]
```

Beispiellösung:

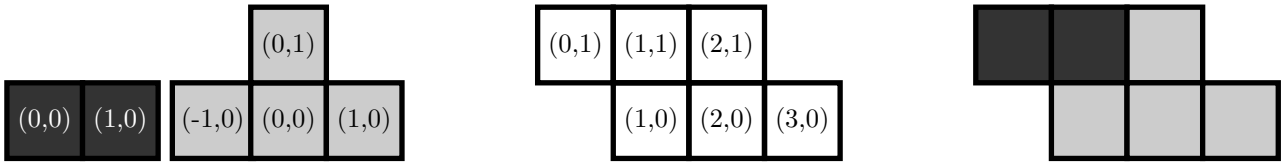
```
subst :: Subst -> Pattern -> Tree
subst s PLeaf = Leaf
subst s (PNode p1 p2) = Node (subst s p1) (subst s p2)
subst s (PVar v) = t
  where
    Just t = lookup v s

liftMaybe2 :: (a -> b -> c) -> Maybe a -> Maybe b -> Maybe c
liftMaybe2 f (Just x) (Just y) = Just (f x y)
liftMaybe2 _ _ _ = Nothing

match :: Pattern -> Tree -> Maybe Subst
match PLeaf Leaf = Just []
match (PNode l1 r1) (Node l2 r2) =
  liftMaybe2 (++) (match l1 l2) (match r1 r2)
match (PVar v) t = Just [(v, t)]
match _ _ = Nothing

matchFirst :: [(Pattern, Subst -> a)] -> Tree -> a
matchFirst ((pat, f):alts) t = go (match pat t)
  where
    go (Just sub) = f sub
    go _          = matchFirst alts t
```

Aufgabe 3 (Prolog: Legepuzzle) [18 Punkte]

Bei einem Legepuzzle muss ein Feld mit einem Vorrat von Steinen bedeckt werden. Das Feld und jeder Stein bestehen jeweils aus einzelnen Quadraten. Es gelten folgende Bedingungen:

- Alle Steine müssen verwendet werden.
- Das Feld muss vollständig bedeckt sein.
- Zwei Steine dürfen sich nicht überlappen.
- Ein Stein darf nicht über das Feld hinausragen.
- Ein Stein darf nicht gedreht oder gespiegelt werden.

Beispiel aus einem Steinvorrat mit zwei Steinen, einem Feld und einer gültigen Platzierung:



Dabei codieren wir

- einen Stein als Liste von `(X,Y)`-Koordinaten der belegten Quadrate, wobei jeder Stein die Koordinate `(0,0)` enthält. Der linke Stein wird z.B. als `[(0,0), (1,0)]` codiert.
- den Steinvorrat als Liste von Steinen, also z.B. den Steinvorrat aus den beiden obigen Steinen als

```
[
 [(0,0), (1,0)],
 [(-1,0), (0,0), (1,0), (0,1)]
].
```

- das Feld als Liste von `(X,Y)`-Koordinaten der freien Quadrate, also z.B. das obige Feld als `[(0,1), (1,1), (2,1), (1,0), (2,0), (3,0)]`.
- die Lösung als Liste von Stein-Koordinaten-Paaren, die für jeden Stein angibt, auf welches Quadrat des Feldes das `(0,0)`-Quadrat gelegt wird. Die Lösung oben wird also codiert als

```
[
 ([(0,0), (1,0)], (0,1)),
 ([(-1,0), (0,0), (1,0), (0,1)], (2,0))
].
```

(a) Definieren Sie ein vierstelliges Prädikat **`move(Stein1, X, Y, Stein2)`**, das genau dann erfüllbar ist, wenn `Stein2` aus `Stein1` durch Verschiebung um `X` in x-Richtung und `Y` in y-Richtung entsteht. Dadurch wird bei einem Stein aus dem Vorrat das Quadrat `(0,0)` auf das Quadrat `(X,Y)` geschoben. Dabei sind `Stein1`, `X` und `Y` stets instanziiert. [6 Punkte]

(b) Definieren Sie ein dreistelliges Prädikat **`remove_all(Feld1, Stein, Feld2)`**, das genau dann erfüllbar ist, wenn `Feld2` aus `Feld1` entsteht, indem daraus alle Quadrate des Steins `Stein` entfernt werden. Ist ein Quadrat aus `Stein` nicht in `Feld1` vorhanden, soll das Prädikat nicht erfüllbar sein.
Hinweis: Das Prädikat `delete(Xs, Y, Zs)` aus der Vorlesung könnte nützlich sein. [4 Punkte]

(c) Definieren Sie ein fünfstelliges Prädikat **`place(Feld1, Stein, X, Y, Feld2)`**, das genau dann erfüllbar ist, wenn `Feld2` aus `Feld1` entsteht, indem `Stein` um `X` in x-Richtung und `Y` in y-Richtung verschoben auf das Feld gelegt wird. Danach sind also die Quadrate aus `Feld1`, auf die der Stein gelegt wurde, nicht mehr frei. Sie können annehmen, dass `Feld1`, `Stein1`, `X` und `Y` stets instanziiert sind. [2 Punkte]

(d) Definieren Sie ein dreistelliges Prädikat **`solve(Feld, Steine, Platzierungen)`**, das genau dann erfüllbar ist, wenn `Platzierungen` eine gültige Lösung für das Legepuzzle mit Feld `Feld` und Steinvorrat `Steine` ist.
Hinweis: Versuchen Sie für einen Stein, die `(0,0)`-Koordinate auf ein noch freies Quadrat im Feld `Feld` zu setzen. [6 Punkte]

Beispiellösung:

```
move([], _, _, []).
move([(X1,Y1)|S1], X, Y, [(X2,Y2)|S2]):-
    X2 is X1 + X,
    Y2 is Y1 + Y,
    move(S1, X, Y, S2).

remove_all(F, [], F).
remove_all(F1, [XY|S], F2):-
    delete(F1, XY, F3), remove_all(F3, S, F2).

place(F1, S, X, Y, F2):-
    move(S, X, Y, S2),
    remove_all(F1, S2, F2).

solve([], [], []).
solve(F, [S|Ss], [(S,(X,Y))|Ps]):-
    member((X,Y), F),
    place(F, S, X, Y, F2),
    solve(F2, Ss, Ps).
```

Aufgabe 4 (λ-Kalkül: Auswertungsreihenfolgen) [12 Punkte]

In dieser Aufgabe kürzen wir Call-by-Value als CBV und Call-by-Name als CBN ab.

Wir betrachten die folgende Familie von λ-Kalkül-Termen:

$$T_0 = \lambda\texttt{x. x}$$
$$T_n = \texttt{(}\lambda\texttt{a. a a)}\ T_{n-1} \qquad \text{für } n \in \mathbb{N}, n > 0$$

Hinweis: Beachten Sie die korrekte Klammerung beim Einsetzen der Terme.

(a) Geben Sie T_3 vollständig eingesetzt an. Markieren Sie in T_3 den Redex, der unter CBV zuerst reduziert wird, und den Redex, der unter CBN zuerst reduziert wird. [2 Punkte]
Beispiellösung:

$$T_3 = \underbrace{\texttt{(}\lambda\texttt{a. a a)}\ \texttt{((}\lambda\texttt{a. a a)}\ \texttt{(}\overbrace{\texttt{(}\lambda\texttt{a. a a)}\ \texttt{(}\lambda\texttt{x. x)}}^{\text{CBV}}\texttt{))}}_{\text{CBN}}$$

Nun seien $V(n)$ und $N(n)$ die Anzahl der β-Reduktionen, die CBV bzw. CBN benötigen, um T_n in seine Normalform zu überführen. Mit diesen wollen wir die Effizienz von CBV und CBN auf T_n untersuchen.

Für alle $n \in \mathbb{N}$ gilt: $T_n \Rightarrow^* \lambda\texttt{x. x}$. Dies dürfen Sie ohne Beweis benutzen.

(b) Stellen Sie für allgemeines $n > 0$ dar, wie T_n unter CBV und CBN zu seiner Normalform reduziert wird. Fassen Sie die Reduktionen von T_{n-1} zu dessen Normalform in einem Schritt zusammen. [5 Punkte]
Geben Sie bei jedem Schritt an, wie viele β-Reduktionen in ihm passieren. Um die Anzahl auszudrücken, dürfen Sie die Funktionen V und N verwenden.
Beispiellösung:

CBV:

$$\begin{array}{lll} T_n & = & \texttt{(}\lambda\texttt{a. a a)}\ T_{n-1} \\ & \Rightarrow^{V(n-1)} & \texttt{(}\lambda\texttt{a. a a) (}\lambda\texttt{x. x)} \\ & \Rightarrow^{1} & \texttt{(}\lambda\texttt{x. x) (}\lambda\texttt{x. x)} \\ & \Rightarrow^{1} & \texttt{(}\lambda\texttt{x. x)} \end{array}$$

alternativ:

$$\begin{array}{lll} T_n & = & \texttt{(}\lambda\texttt{a. a a) (... ((}\lambda\texttt{a. a a) (}\lambda\texttt{x. x)) ...)} \\ & \Rightarrow^{1} & \texttt{(}\lambda\texttt{a. a a) (... ((}\lambda\texttt{x. x) (}\lambda\texttt{x. x)) ...)} \\ & \Rightarrow^{1} & \texttt{(}\lambda\texttt{a. a a) (... ((}\lambda\texttt{x. x)) ...)} \\ & = & T_{n-1} \\ & \Rightarrow^{V(n-1)} & \texttt{(}\lambda\texttt{x. x)} \end{array}$$

CBN:

$$\begin{array}{lll} T_n & = & \texttt{(}\lambda\texttt{a. a a)}\ T_{n-1} \\ & \Rightarrow^{1} & T_{n-1}\ T_{n-1} \\ & \Rightarrow^{N(n-1)} & \texttt{(}\lambda\texttt{x. x)}\ T_{n-1} \\ & \Rightarrow^{1} & T_{n-1} \\ & \Rightarrow^{N(n-1)} & \texttt{(}\lambda\texttt{x. x)} \end{array}$$

(c) Stellen Sie für $V(n)$ und $N(n)$ rekursive Definitionen (Rekurrenzen) auf und geben Sie ohne Begründung das asymptotische Verhalten von $V(n)$ und $N(n)$ an. [5 Punkte]
Hinweis: Wenn Sie sich einige Werte von $V(n)$ und $N(n)$ tabellieren, können Sie deren Verhalten leicht erkennen.
Beispiellösung:

Rekurrenz für V:

$$V(0) = 0$$
$$V(n) = V(n-1) + 2$$

Diese Rekurrenz wird von $V(n) = 2n$ erfüllt, also $V(n) \in \Theta(n)$.

Rekurrenz für N:

$$N(0) = 0$$
$$N(n) = 2 \cdot N(n-1) + 2$$

Diese Rekurrenz wird von $V(n) = 2 \cdot (2^n - 1) = 2^{n+1} - 2$ erfüllt, also $V(n) \in \Theta(2^n)$.

Aufgabe 5 (Typinferenz, Unifikation) [12 Punkte]

Gegeben ist der folgende Ausschnitt aus einem Herleitungsbaum, wobei t für einen nicht weiter bestimmten Term steht:

$$\dfrac{\dfrac{\dfrac{\vdots}{\mathtt{x} : \alpha_2, \mathtt{y} : \alpha_4 \vdash \lambda \mathtt{f}.\ \lambda \mathtt{z}.\ \mathtt{f}\ \mathtt{x}\ (\mathtt{f}\ \mathtt{y}\ \mathtt{z}) : \beta_1} \quad \dfrac{\dfrac{\dfrac{\textcircled{1}}{\Gamma \vdash \mathtt{g} : \alpha_7}\,\mathit{Var}}{\vdots} \quad \ldots \quad \dfrac{\dfrac{\textcircled{2}}{\Gamma \vdash \mathtt{g} : \alpha_8}\,\mathit{Var}}{\vdots}}{\Gamma \vdash t : \alpha_6}}{\mathtt{x} : \alpha_2, \mathtt{y} : \alpha_4 \vdash \mathbf{let}\ \mathtt{g} = \lambda \mathtt{f}.\ \lambda \mathtt{z}.\ \mathtt{f}\ \mathtt{x}\ (\mathtt{f}\ \mathtt{y}\ \mathtt{z})\ \mathbf{in}\ t : \alpha_5}\,\mathit{Let}}{\vdots}$$

Aus dem linken (nicht dargestellten) Teilbaum ergibt sich der allgemeinste Unifikator

$$\begin{aligned}\sigma_{let} = [\ & \\ & \beta_1 \Rightarrow (\beta_8 \to \beta_4 \to \beta_4) \to \beta_4 \to \beta_4, \\ & \beta_{10} \Rightarrow \beta_8, \\ & \beta_5 \Rightarrow \beta_4, \\ & \beta_7 \Rightarrow \beta_4, \\ & \beta_3 \Rightarrow \beta_4 \to \beta_4, \\ & \beta_6 \Rightarrow \beta_4 \to \beta_4, \\ & \beta_9 \Rightarrow \beta_4 \to \beta_4, \\ & \beta_2 \Rightarrow \beta_8 \to \beta_4 \to \beta_4, \\ & \alpha_2 \Rightarrow \beta_8, \\ & \alpha_4 \Rightarrow \beta_8 \\ &].\end{aligned}$$

(a) Geben Sie Γ an. [3 Punkte]

Beispiellösung:

$$\begin{aligned}\Gamma &= \sigma_{let}(\mathtt{x} : \alpha_2, \mathtt{y} : \alpha_4), \mathtt{g} : ta(\sigma_{let}(\beta_1), \sigma_{let}(\mathtt{x} : \alpha_2, \mathtt{y} : \alpha_4)) \\ &= \mathtt{x} : \beta_8, \mathtt{y} : \beta_8, \mathtt{g} : ta((\beta_8 \to \beta_4 \to \beta_4) \to \beta_4 \to \beta_4, (\mathtt{x} : \beta_8, \mathtt{y} : \beta_8)) \\ &= \mathtt{x} : \beta_8, \mathtt{y} : \beta_8, \mathtt{g} : \forall \beta_4.(\beta_8 \to \beta_4 \to \beta_4) \to \beta_4 \to \beta_4\end{aligned}$$

(b) Ergänzen Sie, was an den mit ① und ② markierten Stellen einzutragen ist. [3 Punkte]
Beispiellösung:

① $\Gamma(g) = \ldots,$
$\Gamma(g) \succeq (\beta_8 \to \alpha_9 \to \alpha_9) \to \alpha_9 \to \alpha_9$

② $\Gamma(g) = \ldots,$
$\Gamma(g) \succeq (\beta_8 \to \alpha_{10} \to \alpha_{10}) \to \alpha_{10} \to \alpha_{10}$

(c) Geben Sie einen allgemeinsten Unifikator für die folgende Constaintmenge in Listenform $[\ldots, \alpha_i \mapsto \tau_i, \ldots]$ an. [6 Punkte]

$$
\begin{aligned}
C = \{ & \\
& \alpha_5 = \alpha_4 \to \alpha_6, \\
& \alpha_1 = \alpha_2 \to \alpha_3 \to \alpha_4, \\
& \alpha_1 = \alpha_5 \to \alpha_6, \\
& \}
\end{aligned}
$$

Beispiellösung:

$$
\begin{aligned}
\sigma_{let} = [& \alpha_1 \mapsto (\alpha_4 \to \alpha_3 \to \alpha_4) \to \alpha_3 \to \alpha_4, \\
& \alpha_2 \mapsto \alpha_4 \to \alpha_3 \to \alpha_4, \\
& \alpha_5 \mapsto \alpha_4 \to \alpha_3 \to \alpha_4, \\
& \alpha_6 \mapsto \alpha_3 \to \alpha_4 \\
&].
\end{aligned}
$$

Aufgabe 6 (MPI) [12 Punkte]

Gegeben ist die untenstehende Methode `blockwisePartialSum`. Das im Parameter `elements` übergebene Array beinhaltet eine Sequenz an Integer-Ganzzahlen und der Parameter `elementCount` beinhaltet die Anzahl an Elementen in `elements`.
Gehen Sie bei einer Ausführung der Methode davon aus, dass die Anzahl an Elementen in `elementCount` ein Vielfaches der gestarteten Prozesse ist. Gehen Sie weiterhin davon aus, dass eine korrekte Initialisierung und Finalisierung von MPI vom Aufrufenden durchgeführt wird.

```
1  void blockwisePartialSum(int elements[], int elementCount) {
2
3      int size, rank;
4      MPI_Comm_size(MPI_COMM_WORLD, &size);
5      MPI_Comm_rank(MPI_COMM_WORLD, &rank);
6      MPI_Request request;
7      MPI_Status status;
8
9      int elementsPerProcess = elementCount / size;
10
11     int localElements[elementsPerProcess];
12     MPI_Scatter(elements, elementsPerProcess, MPI_INT, localElements,
           elementsPerProcess, MPI_INT, 0, MPI_COMM_WORLD);
13
14     int received[(rank + 1) * elementsPerProcess];
15
16     for (int i = rank; i < size; i++){
17         MPI_Isend(localElements, elementsPerProcess, MPI_INT, i, 0,
               MPI_COMM_WORLD, &request);
18     }
19
20     for (int i = 0; i <= rank; i++){
21         MPI_Recv(received + (i * elementsPerProcess), elementsPerProcess,
               MPI_INT, i, 0, MPI_COMM_WORLD, &status);
22     }
23
24     int result = 0;
25
26     for (int i = 0; i < (rank + 1) * elementsPerProcess; i++){
27         result += received[i];
28     }
29
30     int collected[size];
31     MPI_Allgather(&result, 1, MPI_INT, collected, 1, MPI_INT,
           MPI_COMM_WORLD);
32 }
```

(a) Im Folgenden finden Sie mehrere Tabellen. Jede Tabelle stellt den Inhalt eines Buffers auf allen gestartenen Prozessen *nach* Ausführung einer definierten Code-Zeile dar. Hierbei befindet sich in der ersten Spalte jeder Tabelle der `rank` der Prozesse und in der zweiten Spalte der Inhalt des Buffers auf dem zum `rank` zugehörigen Prozess. Der umschließende Kasten um eine Tabelle gibt sowohl den Namen des mit der Tabelle korrespondierenden Buffers sowie die Code-Zeile an, nach deren Ausführung der Inhalt dargestellt werden soll.
Ergänzen Sie den Inhalt jeder Tabelle auf Basis der Annahme, dass das Array `elements` die Ganzzahl-Sequenz `{1,2,3,4,5,6,7,8,9}` beinhalten und dass `elementCount` den Wert `9` beinhaltet. Gehen Sie davon aus, dass das Programm mit drei Prozessen gestartet wird. [5 Punkte]

`localElements` (Nach Zeile 12)

Prozess Nr.	**`Inhalt`** **(Lösung)**
0	1, 2, 3
1	4, 5, 6
2	7, 8, 9

`received` (Nach Zeile 22)

Prozess Nr.	**`Inhalt`** **(Lösung)**
0	1, 2, 3
1	1, 2, 3, 4, 5, 6
2	1, 2, 3, 4, 5, 6, 7, 8, 9

`result` (Nach Zeile 28)

Prozess Nr.	**`Inhalt`** **(Lösung)**
0	6
1	21
2	45

`collected` (Nach Zeile 31)

Prozess Nr.	**`Inhalt`** **(Lösung)**
0	6, 21, 45
1	6, 21, 45
2	6, 21, 45

Beispiellösung: *Siehe Tabellen*

(b) Nehmen Sie an, dass der Befehl `MPI_Recv` in Zeile 21 durch `MPI_Irecv` ersetzt wird. [3 Punkte]

i. Erklären Sie allgemein, wie sich `MPI_Recv` und `MPI_Irecv` in ihrem Verhalten unterscheiden.

ii. Wird beim Ersetzen von `MPI_Recv` in Zeile 21 durch `MPI_Irecv` das Verhalten des gesamten Programms beibehalten? Begründen Sie Ihre Antwort.

Beispiellösung:

i. `MPI_Recv` ist eine blockierende Operation, welche den Programmfortschritt so lange stoppt, bis das Empfangen (`MPI_Recv`) einer Nachricht beendet wurde. Bei der *immediate*-Variante `MPI_Irecv` wird nach dem Aufruf sofort der Programmfortschritt fortgesetzt und somit nicht gewartet bis der Empfang beendet ist. Wenn der Inhalt eines Buffers oder der entsprechende Buffer verwendet bzw. wiederverwendet werden soll, muss zunächst manuell auf das Beenden der Operationen gewartet werden oder abgefragt werden ob die Operationen beendet sind. Das explizite Warten geschieht hierbei mit dem Befehl `MPI_Wait` bzw. Varianten wie `MPI_Waitall`. Für das überprüfen, ob `MPI_Isend` bzw. `MPI_Irecv` beendet ist, kann `MPI_Test` bzw. Varianten wie `MPI_Testall` verwendet werden.

ii. Nein, das Verhalten wird nicht beibehalten. Wenn nur der Befehl `MPI_Recv` durch `MPI_Irecv` ersetzt wird ohne weitere Warte-Konstrukte einzufügen, würde `MPI_Irecv` sofort zurückkehren und mit der Berechnung der Summe beginnen bevor möglicherweise die Werte aller Prozesse empfangen wurden. Somit kann die Summenberechnung potenziell inkorrekt werden.

(c) Die Implementierung soll nun so geändert werden, dass die Größe des Eingabe-Arrays `elements` beliebig sein kann. Für diese Änderung könnte unter anderem anstatt `MPI_Scatter` in Zeile 12 die Vektor-Variante `MPI_Scatterv` verwendet werden. Erklären Sie die Bedeutung der Parameter `int* sendcounts` und `int* displacements` von `MPI_Scatterv`. Erklären Sie hierfür wie die Inhalte dieser Arrays an einem bestimmten Index i von `MPI_Scatterv` verwendet wird. [2 Punkte]

Beispiellösung: Die beiden Arrays sind für das Versenden von Blöcken verschiedener Größe notwendig. In dem Array `sendcounts` steht an jeder Array-Position mit dem Index i die Anzahl an Elementen die zu dem Prozess mit dem `rank` i gesendet werden sollen. In dem Array `displacements` steht an jeder Array-Position mit dem Index i für den Prozess mit dem `rank` i der Wert der auf die Basis-Adresse des Sendebuffers addiert werden muss, um die Adresse zu erhalten, von der die nächsten `sendcounts[i]` Elemente versandt werden sollen.

(d) Im Folgenden ist eine unvollständige Implementierung als Ersatz für die Zeilen 11 und 12 der Methode `blockwisePartialSum` gegeben, welche das Verhalten von `MPI_Scatterv` realisieren soll. [2 Punkte]
Ergänzen Sie im folgenden Quelltext den fehlenden Inhalt.
Hinweis: Beachten Sie die Kommentare in der folgendend unvollständigen Implementierung.

```
11 //Sie können die folgenden Variablen "sendcounts" and "displacements"
       in der Implementierung verwenden und annehmen, dass diese valide
     Eingaben für die korrespondierenden Parameter von MPI_Scatterv
     enthalten.
12 //int* sendcounts = ...;
13 //int* displacements = ...;
14
15 localElements[ sendcounts[rank] ];
16
17 if(rank == 0){
18     for(int i = 0; i < size; i++){
19         MPI_Isend( elements + displacements[i], sendcounts[i],
               MPI_INT, i, 0, MPI_COMM_WORLD, &request);
20     }
21 }
22
23 MPI_Recv(localElements, sendcounts[rank], MPI_INT, 0, 0,
     MPI_COMM_WORLD, &status);
```

Aufgabe 7 (Java Happens-before) [5 Punkte]

Gegeben ist untenstehende Klasse `ProducerConsumer` mit einer `main`-Methode. Hierbei erzeugt der Thread `producer` eine zufällige Ganzzahl zwischen 0 und 10 und speichert diese in der Variable `number`. Der Thread `consumer` konsumiert die von `producer` erzeugte Zahl. Auf Basis des gegebenen Programmcodes kann folgende beispielhafte Ausgabe erwartet werden:

```
Produced 3
Producer terminating
Consuming 3
Consumer terminating
```

Bei Ausführung des Programms bemerken Sie ein fehlerhaftes Verhalten, bei dem der Thread `consumer` (und somit das Programm) *nicht* terminiert und nur folgende beispielhafte Ausgabe zu sehen ist:

```
Produced 3
Producer terminating
```

```
1  public class ProducerConsumer {
2
3      public static Boolean numberAvailable = false;
4      public static int number = -1;
5
6      public static void main(String[] args){
7
8          Thread producer = new Thread(()-> {
9
10             while (numberAvailable) {/* busy waiting*/}
11
12             number = new Random().nextInt(10);
13             System.out.println("Produced " + number);
14             numberAvailable = true;
15
16             System.out.println("Producer terminating");
17         });
18
19         Thread consumer = new Thread(()-> {
20
21             while (!numberAvailable) {/* busy waiting*/}
22
23             System.out.println("Consuming " + number);
24             number = -1;
25             numberAvailable = false;
26
27             System.out.println("Consumer terminating");
28         });
29
30         producer.start();
31         consumer.start();
32     }
33 }
```

(a) Erklären Sie den Begriff der *Happens-before-Beziehung*. Nennen Sie außerdem einen Ausdruck oder ein Keyword aus der Vorlesung, welches eine *Happens-before-Beziehung* herstellt. [2 Punkte]

Beispiellösung: Eine *Happens-before-Beziehung* zwischen zwei Statements `s1` und `s2` besagt, dass ein mögliches Schreiben in `s1` immer für `s2` sichtbar ist. Statements die eine *Happens-before-Beziehung* herstellen sind z.B.

- `Thread.start`
- `Thread.join`
- `synchronized`
- `volatile`

Es gibt jedoch noch weitere Ausdrücke. Diese sind u.a. auf der Oracle Dokumentationswebseite des Pakets java.util.concurrent zu finden.

(b) In der gegebenen Implementierung kann es zu Speicherinkonsistenzen kommen, wodurch das beschriebene fehlerhafte Verhalten auftritt. Beschreiben Sie, wie Sie die gegebene Implementierung verändern müssen, damit das Problem der Speicherinkonsistenzen nicht mehr auftritt und sich hierdurch das Programm korrekt verhält. Begründen Sie unter Verwendung der *Happens-before-Beziehung* und mit Bezug auf die gegebene Implementierung, warum Ihre Änderung das Problem behebt. [3 Punkte]

Beispiellösung: Das Problem kann durch die Deklaration von `numberAvailable` als `volatile` gelöst werden. `Volatile` erzeugt eine *Happens-before-Beziehung* zwischen jedem Schreiben der deklarierten Variable, hier `numberAvailable`, und jedem folgenden Lesen über alle Threads. Hierdurch wird garantiert, dass bei jedem Lesen von `numberAvailable` der aktuellste Wert (die letzte Modifikation bzw. das letzte Schreiben) sichtbar ist. Dies kann z.B. durch das garantierte Schreiben des Inhalts der modifizierten Variable in den Hauptspeicher und dem garantierten Lesen der Variable aus dem Hauptspeicher realisiert werden. Hierdurch wird von `consumer` und `producer` immer die Änderung von `numberAvailable` gesehen wodurch der gegenseitige Fortschritt sichergestellt ist. *Hinweis:* Die Variable `number` verursacht das Problem nicht, da der Programmfortschritt durch deren Lesen und Schreiben nicht beeinflusst wird.

Aufgabe 8 (Design-By-Contract) [8 Punkte]

Gegeben sei das Fragment der Klasse `Bicycle`. Diese Klasse stellt ein Fahrrad mit zwei Schalthebeln und einem daraus resultierenden Gang dar. In `firstShift` steht der Wert des ersten Schalthebels und in `secondShift` der Wert des zweiten Schalthebels. Für den Wert von `secondShift` gilt `secondShift < maxSecondShift`. Der aktuelle Gang-Wert in `currentGear` wird durch die Formel `firstShift * maxSecondShift + secondShift` berechnet und muss stets kleiner als `maxGear` sein (`currentGear < maxGear`).

Gegeben ist die Methode `shiftUp(int increase)` mit Vor- und Nachbedingungen. Diese Methode berechnet für einen zu erhöhenden Wert `increase` des aktuellen Ganges die neuen Werte von `firstShift`, `secondShift` und `currentGear`. Zusätzlich ist eine Methode `main` gegeben, welche eine Folge von Aufrufen von `shiftUp` auf einem `Bicycle`-Objekt ausführt.

```
1  public class Bicycle {
2      private int firstShift = 0;
3      private int secondShift = 0;
4      private int currentGear = 0;
5      private final int maxGear = 21;
6      private final int maxSecondShift = 7;
7
8      /*
9      @ requires increase > 0;
10     @ requires currentGear + increase < maxGear;
11     @ ensures secondShift < maxSecondShift;
12     @ ensures increase >= maxSecondShift ==>
                    firstShift == \old(firstShift) + 1;
13     */
14     public void shiftUp(int increase) {
15
16         currentGear += increase;
17         firstShift = currentGear / maxSecondShift;
18         secondShift = currentGear % maxSecondShift;
19     }
20
21     public static void main(String[] args){
22         Bicycle bicycle = new Bicycle();
23         bicycle.shiftUp(3);
24         bicycle.shiftUp(10);
25         bicycle.shiftUp(9);
26     }
27 }
```

(a) Wird der Vertrag von `shiftUp` vom *Aufrufenden* erfüllt? Begründen Sie ihre Antwort. [2.5 Punkte]

Beispiellösung: Nein, der Aufrufende erfüllt den Vertrag nicht. Die zweite Vorbedingung besagt, dass das Ergebnis von `currentGear + increase` immer kleiner als `maxGear` sein muss. Diese Vorbedingung ist verletzt, da in der `main`-Methode beim dritten Aufruf von `shiftUp` `maxGear` 21 wäre (3 + 10 + 8 - 1).

(b) Wird der Vertrag von `shiftUp` vom *Aufgerufenen* erfüllt? Begründen Sie Ihre Antwort. [2.5 Punkte]

Beispiellösung: Nein, der Vertrag wird nicht vom Aufgerufenen erfüllt. Die zweite Nachbedingung wird nicht erfüllt, wenn in einem Aufruf von `shiftUp` der Wert von `firstShift` um zwei oder mehr erhöht wird. Diese Erhöhung hängt in der Implementierung von dem gewählten `increase` ab. Wenn gilt $increase > 2 * maxSecondShift$ wird firstGear um mindestens zwei erhöht (z.B. `shiftUp(14)`).

(c) Ergänzen Sie folgende alternative Implementierung der oben gegebenen `shiftUp`-Methode mit Assertions (`assert expr;`), so dass ihre Vor- und Nachbedingungen sichergestellt werden. Sie dürfen hierfür weitere Variablen und Kontrollstrukturen einfügen. [3 Punkte]

Beispiellösung: Das Schlüsselwort `old` beschreibt den Wert einer Variablen vor der Ausführung der Methode. Hierdurch müssen für alle Verwendungen von `old` zur Überprüfung einer Nachbedingung der Wert der Variablen zu Beginn in einer lokale Variable geschrieben werden. Die Bedingung A ==> B heißt, dass wenn A gilt auch B gelten muss. Hieraus folgt: if A then assert B.

```
public void shiftUp(int increase) {

    int oldFirstShift = firstShift;

    assert increase > 0;
    assert currentGear + increase < maxGear;

    currentGear += increase;
    firstShift = currentGear / maxSecondShift;
    secondShift = currentGear % maxSecondShift;

    assert secondShift < maxSecondShift;

    if(increase >= maxSecondShift){
        assert (firstShift == oldFirstShift + 1);
    }
}
```

Aufgabe 9 (Grammar Engineering) [5 Punkte]

Gegeben sei die folgende Grammatik mit Startsymbol T und den Terminalen **a**, **b**, **c**, **d**, **e** und **f**.

$$T \rightarrow \texttt{a} \mid \texttt{a b} \mid \texttt{c d f} \mid \texttt{c e f}$$

(a) Begründen Sie kurz, warum diese Grammatik nicht SLL(1) ist. [2 Punkte]
Wir betrachten die ersten beiden Produktionen für das Nichtterminal T:

$$\text{First}_1(\texttt{a}) \cap \text{First}_1(\texttt{a b}) = \{\texttt{a}\} \neq \emptyset \lightning$$

Dies steht im Widerspruch zur SLL(1)-Bedingung der Vorlesung.

(b) Führen Sie eine Linksfaktorisierung der Grammatik durch. [3 Punkte]

$$\begin{aligned} T &\rightarrow \texttt{a}\ \mathit{ARest} \mid \texttt{c}\ \mathit{CRest} \\ \mathit{ARest} &\rightarrow \varepsilon \mid \texttt{b} \\ \mathit{CRest} &\rightarrow \texttt{d f} \mid \texttt{e f} \end{aligned}$$

Aufgabe 10 (Syntaktische Analyse) [18 Punkte]

Gegeben sei im Folgenden eine Grammatik für Java-Generics mit Startsymbol *Generics*.

$$\begin{aligned} \mathit{Generics} &\rightarrow \texttt{<}\ \mathit{GList}\ \texttt{>} & \mathit{GAtom} &\rightarrow \texttt{id}\ \mathit{IdRest} \mid \texttt{?} \\ \mathit{GList} &\rightarrow \mathit{GAtom}\ \mathit{GRest} & \mathit{IdRest} &\rightarrow \varepsilon \mid \mathit{Generics} \\ \mathit{GRest} &\rightarrow \varepsilon \mid \texttt{,}\ \mathit{GList} \end{aligned}$$

(a) Tragen Sie die Follow_1-Menge aller Nichtterminale der Grammatik in die folgende Tabelle ein. [5 Punkte]

Nichtterminal	Follow_1-Menge
Generics	{**#**, **>**, **,**}
GList	{**>**}
GRest	{**>**}
GAtom	{**>**, **,**}
IdRest	{**>**, **,**}

(b) Begründen Sie, warum die Berechnung der Indizmengen für die Produktionen der Nichtterminale *Generics* und *GList* für die Überprüfung SLL(1)-Bedingung nicht notwendig ist. [2 Punkte]
Die SLL(1)-Bedingung betrachtet nur Paare von Produktionen für Nichtterminale. Da *Generics* und *GList* jeweils nur eine Produktion haben, muss die Bedingung nicht explizit überprüft werden.

(c) Zeigen Sie, dass die Grammatik in SLL(1) ist. Tragen Sie die hierzu notwendigen Indizmengen in die folgende Tabelle ein. [3 Punkte]

Nichterminimal	Indizmengen (in der Reihenfolge der Produktionen)
GRest	{**>**}, {**,**}
GAtom	{**id**}, {**?**}
IdRest	{**>**, **,**}, {**<**}

(d) Vervollständigen Sie den unten gegebenen SLL(1)-Parser für die Grammatik, indem Sie die Methoden für die Nichtterminale *Generics*, *GAtom* und *IdRest* implementieren. Brechen Sie bei Parserfehlern mit der globalen `error()`-Funktion ab. [8 Punkte]

Hinweis: Sie können davon ausgehen, dass die Lexer-Schnittstelle wie folgt aussieht:

- Die Token sind Werte eines `enum` namens **`Token`** und heißen **`LT`**, **`GT`**, **`COMMA`**, **`QUESTION`**, **`IDENT`** und **`EOF`** für **`<`**, **`>`**, **`,`**, **`?`**, Identifier sowie das Ende der Eingabe. Token können bspw. mit `==` verglichen werden.
- `lexer.current` gibt nichtkonsumierenden Zugriff auf das aktuelle Token.
- `lexer.lex()` konsumiert das aktuelle Token.

```
void parseGList(){...} void parseGRest(){...}

void expect(Token tok) {
  if (lexer.current != tok) { error(); }
  lexer.lex();
}
```

Beispiellösung:

```
void parseGenerics(){
  expect(LT);
  parseGList();
  expect(GT);
}

void parseGAtom(){
  switch(lexer.current) {
  case IDENT:
    lexer.lex();
    parseIdRest();
    break;
  case QUESTION:
    lexer.lex();
    break;
  default:
    error();
  }
}

void parseIdRest(){
  switch(lexer.current) {
  case COMMA:
  case GT:
    break;
  case LT:
    parseGenerics();
    break;
  default:
    error();
  }
}
```