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

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 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]

Beispiellösung:

```haskell
magicNumber :: Int -> Int
magicNumber n = sum [1..n*n] `div` n
```

(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. [5 Punkte]
Geben Sie einen möglichst allgemeinen Typ für `duplicates` an.

Beispiellösung:

```haskell
duplicates :: Eq a => [a] -> Bool
duplicates [] = False
duplicates (x:xs) = x `elem` xs || duplicates xs
```

(c) Geben Sie eine Funktion `transpose :: [[a]] -> [[a]]` an, welche eine Liste von Listen `xs :: [[a]]` transponiert. [10 Punkte]
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.

Beispiellösung:

```haskell
transpose :: [[a]] -> [[a]]
transpose [] = []
transpose ([]:xs) = transpose xs
transpose xss = [h | (h:_) <- xss] : transpose ([t | (_:t) <- xss])
```

Oder alternativ:

```haskell
transpose' :: [[a]] -> [[a]]
transpose' [] = []
transpose' [xs] = map (\x -> [x]) xs
transpose' (xs:xss) = zipWith (:) xs (transpose' xss)
```

(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
```

Beispiellösung:

```
isMagic :: [[Int]] -> Bool
isMagic s = let n = length s
                m = magicNumber n
  in (not $ duplicates (concat s))
    && all (\i -> 1 <= i && i <= n*n) (concat s)
    && all (\row -> length row == n && sum row == m) s
    && all (\col -> length col == n && sum col == m) (transpose s)
```

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
```

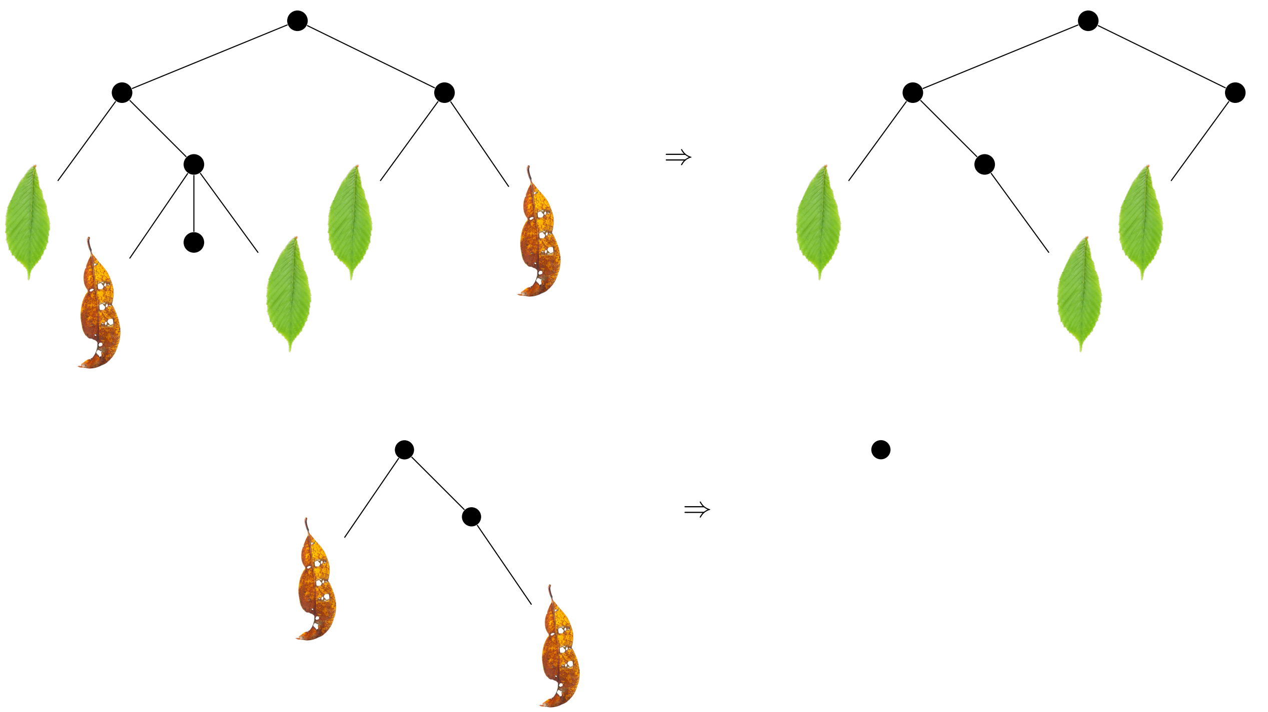
an, die alle toten Teilbäume „entfernt“.

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 []
```



Beispiellösung:

(a)
```
prune :: Tree -> Tree
prune (Leaf s)  = Leaf s
prune (Node ts) = Node removed
  where removed = filter (not . isDead) (map prune ts)
        isDead (Leaf Dead)   = True
        isDead (Node [])     = True
        isDead _             = False
```

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

In Prolog können Formeln der Sprache[1]

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

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

$\forall\ \mathsf{x}.\ \exists\ \mathsf{y}.\ \mathsf{q}\,(\mathsf{x}, \mathsf{y})$ als `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

Beispiellösung:

(a)
```
hasfree(BoundVars,X) :- atom(X), not(member(X,BoundVars)).
hasfree(BoundVars,forall(X,T)) :- hasfree([X|BoundVars],T).
hasfree(BoundVars,exists(X,T)) :- hasfree([X|BoundVars],T).
hasfree(BoundVars,p(T1))    :- hasfree(BoundVars, T1).
hasfree(BoundVars,q(T1,_)) :- hasfree(BoundVars, T1).
hasfree(BoundVars,q(_,T2)) :- hasfree(BoundVars, T2).
```

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 unify(C) aus. Geben Sie bei jedem rekursiven Aufruf von unify die erzeugte Substitution sowie die noch zu unifizierende Menge an.

Beispiellösung:

unify(C)
= unify($\{\texttt{node}(\texttt{T}, 2, \texttt{leaf}) = \texttt{node}(\texttt{node}(\texttt{R}, 1, \texttt{leaf}), \texttt{V}, \texttt{T}), 4 = 4, \texttt{R} = \texttt{leaf}\}$)
= unify($\{4 = 4, \texttt{R} = \texttt{leaf}, \texttt{T} = \texttt{node}(\texttt{R}, 1, \texttt{leaf}), 2 = \texttt{V}, \texttt{leaf} = \texttt{T}\}$)
= unify($\{\texttt{R} = \texttt{leaf}, \texttt{T} = \texttt{node}(\texttt{R}, 1, \texttt{leaf}), 2 = \texttt{V}, \texttt{leaf} = \texttt{T}\}$)
= unify($\{\texttt{T} = \texttt{node}(\texttt{leaf}, 1, \texttt{leaf}), 2 = \texttt{V}, \texttt{leaf} = \texttt{T}\}$) $\circ [\texttt{R} \Rightarrow \texttt{leaf}]$
= unify($\{2 = \texttt{V}, \texttt{leaf} = \texttt{node}(\texttt{leaf}, 1, \texttt{leaf})\}$) $\circ [\texttt{T} \Rightarrow \texttt{node}(\texttt{leaf}, 1, \texttt{leaf})] \circ [\texttt{R} \Rightarrow \texttt{leaf}]$
= unify($\{\texttt{leaf} = \texttt{node}(\texttt{leaf}, 1, \texttt{leaf})\}$) $\circ [\texttt{V} \Rightarrow 2] \circ [\texttt{T} \Rightarrow \texttt{node}(\texttt{leaf}, 1, \texttt{leaf})] \circ [\texttt{R} \Rightarrow \texttt{leaf}]$
= **fail**

C ist also nicht unifizierbar.

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.
Beispiellösung: $t = \lambda\mathtt{f}.\ \lambda\mathtt{g}.\ \lambda\mathtt{x}.\ (\mathtt{f}\ \mathtt{g})\ (\mathtt{g}\ \mathtt{x})$

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

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

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

ii. Es seien

$$\begin{aligned}\Gamma_{\mathtt{tb}} &= \Gamma_{\mathtt{t}}, \mathtt{b} : \tau_{\mathtt{b}}^{poly}\\ \Gamma_{\mathtt{tby}} &= \Gamma_{\mathtt{tb}}, \mathtt{y} : \alpha_2\\ \Gamma_{\mathtt{tbyz}} &= \Gamma_{\mathtt{tby}}, \mathtt{z} : \alpha_4\end{aligned}$$

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

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

Beispiellösung:

- Typgleichungen:

$$\begin{aligned}\alpha_1 &= \alpha_2 \to \alpha_3\\ \alpha_3 &= \alpha_4 \to \alpha_5\\ \alpha_6 &= \alpha_9 \to \alpha_5\\ \alpha_7 &= \alpha_8 \to \alpha_6\\ \alpha_7 &= (\alpha' \to \alpha') \to \alpha' \to \alpha'\\ \alpha_2 &= \alpha_8\\ \alpha_{10} &= \alpha_{11} \to \alpha_9\\ \alpha_{10} &= (\alpha'' \to \alpha'') \to \alpha'' \to \alpha''\\ \alpha_4 &= \alpha_{11}\end{aligned}$$

- A : $\Gamma_{\mathtt{tbyz}}(\mathtt{b}) = \forall\alpha.\ (\alpha \to \alpha) \to \alpha \to \alpha, \forall\alpha.\ (\alpha \to \alpha) \to \alpha \to \alpha \succeq (\alpha' \to \alpha') \to \alpha' \to \alpha'$
- B : $\Gamma_{\mathtt{tbyz}}(\mathtt{b}) = \forall\alpha.\ (\alpha \to \alpha) \to \alpha \to \alpha, \forall\alpha.\ (\alpha \to \alpha) \to \alpha \to \alpha \succeq (\alpha'' \to \alpha'') \to \alpha'' \to \alpha''$

iii. Geben Sie einen allgemeinsten Typen von $\lambda\mathtt{y}.\ \lambda\mathtt{z}.\ (\mathtt{b}\ \mathtt{y})\ (\mathtt{b}\ \mathtt{z})$ unter der Typannahme $\Gamma_{\mathtt{tb}}$ an! (2 Punkte)
Beispiellösung: $((\alpha \to \alpha) \to \alpha \to \alpha) \to (\alpha \to \alpha) \to \alpha \to \alpha$

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:

$\mathsf{c_{true}} = \lambda\mathtt{t}.\ \lambda\mathtt{f}.\ \mathtt{t}$	$\mathsf{c_{false}} = \lambda\mathtt{t}.\ \lambda\mathtt{f}.\ \mathtt{f}$	
$\mathsf{isZero}\ \mathsf{c}_0 \Rightarrow^* \mathsf{c_{true}}$	$\mathsf{isZero}\ \mathsf{c}_n \Rightarrow^* \mathsf{c_{false}}$ für alle $n > 0$	$\mathsf{succ}\ \mathsf{c}_n \Rightarrow^* \mathsf{c}_{n+1}$ für alle n

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

Gegeben seien folgende λ-Ausdrücke:

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

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

(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: $\mathsf{c}_0\ \mathsf{a}\ \mathsf{b} \Rightarrow^* \mathsf{b}$.

Beispiellösung: Wichtig ist, die Definition von c_2 zu $(\lambda\mathtt{s}.\ \lambda\mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))$ aufzufalten, sonst lässt sich der resultierende Ausdruck gar nicht erst reduzieren.

Die Reduktion lässt sich recht schnell durchführen, wenn einem auffällt, dass $\varphi\ (\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_n\ \mathsf{c}_m) \Rightarrow^* (\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_m\ \mathsf{c}_{m+1})$ gilt. Ausformuliert:

$$\begin{array}{ll}
 & \mathsf{pred}\ \mathsf{c}_2 \\
\Rightarrow & (\lambda\mathtt{s}.\ \lambda\mathtt{z}.\ \mathtt{s}\ (\mathtt{s}\ \mathtt{z}))\ \varphi\ (\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_0\ \mathsf{c}_0)\ \mathsf{c_{true}} \\
\Rightarrow^2 & \varphi(\varphi(\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_0\ \mathsf{c}_0))\ \mathsf{c_{true}} \\
= & \varphi((\lambda\mathtt{x}.\ \lambda\mathtt{f}.\ \mathtt{f}\ (\mathtt{x}\ \mathsf{c}_0)\ (\mathsf{succ}\ (\mathtt{x}\ \mathsf{c}_0)))\ (\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_0\ \mathsf{c}_0))\ \mathsf{c_{true}} \\
\Rightarrow^* & \varphi(\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_0\ (\mathsf{succ}\ \mathsf{c}_0))\ \mathsf{c_{true}} \\
\Rightarrow^* & \varphi(\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_0\ \mathsf{c}_1)\ \mathsf{c_{true}} \\
= & (\lambda\mathtt{x}.\ \lambda\mathtt{f}.\ \mathtt{f}\ (\mathtt{x}\ \mathsf{c}_0)\ (\mathsf{succ}\ (\mathtt{x}\ \mathsf{c}_0)))\ (\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_0\ \mathsf{c}_1)\ \mathsf{c_{true}} \\
\Rightarrow^* & (\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_1\ (\mathsf{succ}\ \mathsf{c}_1))\ \mathsf{c_{true}} \\
\Rightarrow^* & (\lambda\mathtt{f}.\ \mathtt{f}\ \mathsf{c}_1\ c_2)\ \mathsf{c_{true}} \\
\Rightarrow & \mathsf{c_{true}}\ \mathsf{c}_1\ c_2 \\
\Rightarrow^2 & \mathsf{c}_1
\end{array}$$

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

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

ii. $\mathsf{eq}\ \mathsf{c}_n\ \mathsf{c}_m \Rightarrow^* \mathsf{c_{false}}$ 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 _`.

Beispiellösung: Diese Aufgabe lässt sich auch ohne Rekursion mit Hilfe der (aus der Übung bekannten) Subtraktionsfunktion lösen. Hierbei muss man jedoch beachten, dass Church-Zahlen *natürliche Zahlen* sind, und Subtraktion auf natürlichen Zahlen "saturierend" ist, d.h. $0 - n = 0$.

Daher gilt $n = m$ nur dann, wenn sowohl $n - m = 0$ und $m - n = 0$ gelten. Eine funktionierende Lösung mit diesem Ansatz wäre zum Beispiel:

$$\mathsf{eq} = \lambda\mathtt{n}.\ \lambda\mathtt{m}.\ (\mathsf{isZero}\ (\mathsf{sub}\ \mathtt{n}\ \mathtt{m}))\ (\mathsf{isZero}\ (\mathsf{sub}\ \mathtt{m}\ \mathtt{n}))\ \mathsf{c}_{\mathsf{false}}$$

Da die notwendige Rekursion schon von sub durchgeführt wird, ist dies schon eine gültige Definition, und benötigt nicht einmal mehr die übliche Handhabung via Funktional + Y-Kombinator.

Die Aufgabe kann allerdings komplett ohne sub, sondern mit einer einfachen Rekursion gelöst werden. Vollständig ausformulierte Fallunterscheidung in Haskell:

```haskell
eq :: ChurchNumeral -> Bool
eq n m =
    if isZero n
    then if isZero m
        then True
        else False
    else if isZero m
        then False
        else eq (n-1) (m-1)
```

Da dies nur aus einer Kaskade von `if-then-else`-Ausdrücken besteht, ist die Umsetzung recht einfach:

$$\mathsf{eq} = \lambda\mathtt{n}.\ \lambda\mathtt{m}.\ (\mathsf{isZero}\ \mathtt{n})\ ((\mathsf{isZero}\ \mathtt{m})\ \mathsf{c}_{\mathsf{true}}\ \mathsf{c}_{\mathsf{false}})\ (\mathsf{isZero}\ \mathtt{m}\ \mathsf{c}_{\mathsf{false}}\ (\mathsf{eq}\ (\mathsf{pred}\ \mathtt{n})\ (\mathsf{pred}\ \mathtt{m})))$$

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

Hinweis: Verwenden Sie den Fixpunktkombinator Y.

Beispiellösung: Das Funktional abstrahiert lediglich das freie eq auf der rechten Seite der Rekursionsgleichung:

$$\mathsf{EQ} = \lambda\mathtt{eq}.\ \lambda\mathtt{n}.\ \lambda\mathtt{m}.\ (\mathsf{isZero}\ \mathtt{n})\ ((\mathsf{isZero}\ \mathtt{m})\ \mathsf{c}_{\mathsf{true}}\ \mathsf{c}_{\mathsf{false}})\ (\mathsf{isZero}\ \mathtt{m}\ \mathsf{c}_{\mathsf{false}}\ (\mathtt{eq}\ (\mathsf{pred}\ \mathtt{n})\ (\mathsf{pred}\ \mathtt{m})))$$

Die tatsächliche Definition von eq ist nun der Fixpunkt dieses Funktionals:

$$\mathsf{eq} = \mathsf{Y}\ \mathsf{EQ}$$

Wurde in (b) die Lösung ohne direkte Rekursion verwendet, so konnte diese Aufgabe weiterhin gelöst werden, indem für EQ eine konstante Funktion benutzt wurde, also $\mathsf{EQ} = \lambda f.\ \mathsf{eq}$ (mit eq wie in (b)). Der Y-Kombinator tut auf einem solchen konstanten Funktional das Gewünschte, da alle Eingabewerte Fixpunkte auf konstanten Funktionen sind. In diesem Fall wäre der Y-Kombinator aber nicht einmal nötig gewesen, um eine wohlgeformte Definition zu erlangen.

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.

```c
void operate(int elementCount, int elements[elementCount]) {
    int rank;
    int processCount;
    int rootRank = 0;

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &processCount);

    int elementsPerProcess = elementCount / processCount;
    int local[elementsPerProcess];
    MPI_Scatter(elements, elementsPerProcess, MPI_INT, local,
        elementsPerProcess, MPI_INT, 0, MPI_COMM_WORLD);

    int offset = 0;
    if (rank == rootRank) {
        for (int i = 0; i < elementCount; i++) {
            if (elements[i] < offset) offset = elements[i];
        }
    }

    if (rank == rootRank) {
        for (int process = 0; process < processCount; process++) {
            if (process != rootRank) {
                MPI_Send(&offset, 1, MPI_INT, process, 0, MPI_COMM_WORLD);
            }
        }
    } else {
        MPI_Recv(&offset, 1, MPI_INT, rootRank, 0, MPI_COMM_WORLD, NULL);
    }

    for (int i = 0; i < elementsPerProcess; i++) {
        local[i] = local[i] - offset;
    }

    MPI_Gather(local, elementsPerProcess, MPI_INT, elements,
        elementsPerProcess, MPI_INT, rootRank, MPI_COMM_WORLD);

    if (rank == rootRank) {
        for (int i = 0; i < elementCount; i++) {
            printf("%i ", elements[i]);
        }
    }
}
```

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

Beispiellösung: Das Programm erhöht alle Werte des Eingabe-Array `elements` um die Differenz zwischen null und dem kleinsten Wert des Arrays, falls dieser kleiner als null ist. Ansonsten tut das Programm nichts.

Ausgabe: "0 3 10 6"

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

Beispiellösung:

```
MPI_Bcast(&offset, 1, MPI_INT, rootRank, MPI_COMM_WORLD);
```

(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. [3 Punkte]
Hinweis: Sie müssen nicht (dürfen aber) den endgültigen Programmcode angeben, eine textuelle Erläuterung genügt.

Beispiellösung:

Es kann lokal in jedem Prozess für das Teilarray der Minimalwert berechnet werden und dann per `MPI_Reduce` das globale Minimum über diese lokalen Minima ermittelt werden.

```
int localOffset = 0;
for (int i = 0; i < elementsPerProcess; i++) {
    if (local[i] < localOffset) localOffset = local[i];
}
MPI_Reduce(&localOffset, &offset, 1, MPI_INT, MPI_MIN,
    rootRank, MPI_COMM_WORLD);
```

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.

```java
public class Worker extends UntypedActor {
    @Override
    public void onReceive(Object query) {
        // Handle query
    }
}

public class Balancer extends UntypedActor {
    private final int workerCount;
    private List<ActorRef> worker;

    private int nextWorker;

    public Balancer(int workerCount) {
        if (workerCount < 1) {
            throw new IllegalArgumentException();
        }
        this.workerCount = workerCount;
    }

    private ActorRef createWorker() {
        return getContext().actorOf(Props.create(Worker.class));
    }

    @Override
    public void preStart() throws Exception {
        super.preStart();
        this.worker = new ArrayList<>();

        this.nextWorker = 0;
        for (int i = 0; i < workerCount; i++) {
            this.worker.add(createWorker());
        }

    }

    @Override
    public void onReceive(Object query) {

        this.worker.get(nextWorker).tell(query, getSender());
        nextWorker = (nextWorker + 1) % workerCount;

    }
}
```

(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.

Beispiellösung: Die Lösung befindet sich innerhalb der Implementierung. Alternativ kann die Nachricht auch via `forward` weitergeleitet werden.

(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.

Beispiellösung: Jede Anfrage besteht aus einem durch den `Balancer` auszuführenden Teil und einem an einen `Worker` zugewiesen Teil im Verhältnis $\frac{1}{5}$ zu $\frac{4}{5}$. Im ersten Szenario wird das Programm vollständig sequentiell ausgeführt. Im zweiten Szenario, bei der Bearbeitung einer beliebig hohen Anzahl von Anfragen durch beliebig viele `Worker` sind die den Arbeitern zugewiesenen Teile parallel ausführbar, während nur die Verteilung sequentiell vom `Balancer` ausgeführt wird. Es lässt sich somit Amdahls Gesetz anwenden, aus welchem sich folgender maximaler Speedup ergibt: $S = \frac{T(1)}{T(n \to \text{inf})} = \frac{1}{1-\frac{4}{5}} = 5$
Der Durchsatz der Anwendung ist im zweiten Szenario dementsprechend fünfmal so hoch wie im ersten.

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 `Tokens` 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.

```java
public interface Token {}

public class Place {
    private final List<Token> tokens;
    private final int capacity;

    public Place(Collection<Token> initialTokens, int capacity) {
        this.tokens = new ArrayList<>();
        this.tokens.addAll(initialTokens);
        this.capacity = capacity;
    }

    /**
     * Bewegt ein Token von diesem in den übergebenen Place.
     * Das Token kann sich in einem Übergangszustand befinden,
     * in dem es weder in diesem noch in dem Ziel-Place vorhanden ist.
     */
    public void moveTokenTo(Place otherPlace) {
        Token token = this.takeAnyToken();
        otherPlace.putToken(token);
    }

    /**
     * Fügt der "tokens"-Liste das übergebene Token hinzu.
     * Falls die Kapazität der Liste aktuell erschöpft ist, wartet
     * die Methode, bis ein Token entfernt wurde.
     */
    private void putToken(Token token) {

        synchronized(this) {
            while (this.tokens.size() >= capacity) {
                try {
                    wait();
                } catch (InterruptedException e) {}
            }
            this.tokens.add(token);
            notifyAll();
        }

    }
```

```java
    /**
     * Gibt ein beliebiges Token aus der "tokens"-Liste zurück.
     * Falls aktuell kein Token vorhanden ist, wartet die Methode,
     * bis ein Token hinzugefügt wurde.
     */
    private Token takeAnyToken() {

        synchronized(this) {
            while (this.tokens.size() == 0) {
                try {
                    wait();
                } catch (InterruptedException e) {}
            }
            notifyAll();
            return this.tokens.remove(0);
        }

    }

}
```

(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`

Beispiellösung: *Siehe Quellcode.* Es ist jeweils eine Synchronisation auf `this` erforderlich. Solange die Liste leer ist bzw. ihre Kapazität erreicht hat, wird `wait()` aufgerufen. Wenn die jeweilige Operation durchgeführt wurde, wird `notifyAll()` aufgerufen, da sich der Zustand der Token-Liste geändert hat und andere Threads somit potentiell weiterarbeiten können.

(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]

Beispiellösung:

Ja, es kann hier zu einem Deadlock kommen.
Die Coffman-Bedingungen *mutual exclusion* und *no preemption* sind durch die Verwendung des `synchronized`-Schlüsselwortes erfüllt. Das `synchronized`-Schlüsselwort der Methode `moveTokenTo(Place)` sorgt für eine Synchronisation auf `this`. Zudem führt der Ausdruck `otherPlace.putToken(token)` zu einer Synchronisation auf `otherPlace`. Dies erfüllt bereits die *hold-and-wait*-Bedingung. Wird die Methode `moveTokenTo(Place)` nun nebenläufig für den wechselseitigen Austausch von Tokens zwischen zwei Places aufgerufen, so ist zusätzlich auch *circular wait* erfüllt (z.B. `p1.moveTokenTo(p2)` und `p2.moveTokenTo(p1)`).

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.

```java
public interface Option {}

public class Selection {
    private Set<Option> options = new HashSet<>();

    /**
     * Vorbedingungen:
     *   1. option != null;
     *
     * Nachbedingungen:
     *   1. options.size() == \old(options).size() + 1;
     *   2. options.contains(option);
     */
    public void select(Option option) {
        this.options.add(option);
    }

    /**
     * Vorbedingungen:
     *   1. option != null;
     */
    public void deselect(Option option) {
        this.options.remove(option);
    }
}
```

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

Beispiellösung:

Die Nachbedingungen werden nicht erfüllt. Ist beispielsweise die übergebene `Option` bereits vorher selektiert worden, so ändert sich die Anzahl der Elemente in `options` nicht, da dieses ein `Set` ist. Somit ist in diesem Fall die erste Nachbedingung nicht erfüllt.

(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`

Beispiellösung:

Eine Möglichkeit ist zu garantieren, dass alle nachher selektierten Optionen auch vorher selektiert waren und dass falls die übergebene `Option` vorher selektiert war die Größe von `options` um eins reduziert wird und sich ansonsten nicht ändern.

```
1.  if (\old(options).contains(option) {
        options.size() = \old(options).size() - 1;
    } else {
        options.size() = \old(options).size();
    }
2.  \old(options).containsAll(options);
```

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

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

Geben Sie zwei weitere gültige Teilmengenbeziehung zwischen den Mengen

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

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

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

Beispiellösung:

$$\begin{array}{rcl} \mathrm{First}_1(B) & \subseteq & \mathrm{Follow}_1(A) \\ \mathrm{Follow}_1(S) & \subseteq & \mathrm{Follow}_1(B) \end{array}$$

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`.

```java
class C {
    int x;
    int y;

    public void f(int a, int b) {
        int c = 0;
        while (a >= b) {
            a -= b;
            ++c;
        }
        x = c;
        y = a;
    }
}
```