Karlsruher Institut für Technologie

Institut für Programmstrukturen und Datenorganisation

Prof. Dr.-Ing. Gregor Snelting — Prof. Dr. Ralf Reussner

gregor.snelting@kit.edu — reussner@kit.edu

2 215000 006343

Klausur Programmierparadigmen

WS2011/12, 27. März 2012, 8:00 - 10:00 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	25	
2	20	
3	5	
4	20	
5	10	
6	10	
7	10	
8	20	
$\sum$	**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 Klausurangabe. Beschriften Sie **alle verwendeten Blätter** mit Ihrem Namen und Ihrer Matrikelnummer. Trennen Sie die geklammerten Blätter der Klausur nicht auf. Weitere Blätter (auch für die Lückenaufgaben) erhalten Sie bei Bedarf.

Die **Klausureinsicht** findet am 25.4.2012 um 14:00h im Raum 010, Informatik-Gebäude statt.

Aufgabe 1 (Haskell, Bäume, Funktionen höherer Ordnung) [25 Punkte]

In *Mehrweg*-Bäumen hat jeder Knoten eine beliebig lange Liste von Kindknoten. Jeder Knoten speichert zudem einen Wert. Es gibt *keine* leeren Mehrweg-Bäume. „Blätter“ sind lediglich normale Knoten, deren Kindknoten-Liste leer ist.

(a) Definieren Sie einen Datentyp [3 Punkte]

```
Tree t
```

für Mehrweg-Bäume, deren Knoten Werte vom Typ `t` speichern.

Beispiellösung:

```haskell
data Tree t = Node t [Tree t]
```

(b) Definieren Sie eine Funktion [5 Punkte]

```haskell
mapT :: (t -> s) -> Tree t -> Tree s.
```

`mapT f tree` soll den Mehrfach-Baum zurückgeben, der aus `tree` entsteht, indem auf jeden enthaltenen Wert `f` angewendet wird.

Beispiellösung:

```haskell
mapT f (Node x ts) = Node (f x) (map (mapT f) ts)
```

(c) Definieren Sie eine Funktion [9 Punkte]

```haskell
reduceT :: (t -> t -> t) -> Tree t -> t
```

welche alle Werte eines Mehrweg-Baums mit der Funktion $\oplus$ `:: (t -> t -> t)` verknüpft. Falls $x_1, \ldots, x_n$ die Werte sind, die in den Knoten des Mehrweg-Baums `tree` gespeichert sind, soll `reduceT` $\oplus$ `tree` also den Ausdruck

$$x_1 \oplus x_2 \oplus \ldots \oplus x_n$$

berechnen. Die Reihenfolge, in der die Werte verknüpft werden, sowie die Klammerung der Teilausdrücke sind beliebig.

Beispiellösung:

```haskell
reduceT f (Node x ts) =  (foldr f x (map (reduceT f) ts))
```

(d) Verwenden Sie `reduceT` zur Definition von Funktionen [3 Punkte]

- `sumT :: Tree Integer -> Integer` für Bäume mit ganzzahligen Knotenwerten. `sumT` soll die Summe aller in den Knoten des Baums gespeicherten Werte zurückliefern.
- `concatT :: Tree [t] -> [t]` für Bäume, deren Knoten Listen als Werte enthalten. Alle in den Knoten eines solchen Mehrweg-Baums gespeicherten Listen sollen aneinander gehängt und die resultierende Liste zurückgegeben werden.

Beispiellösung:

```haskell
sumT    = reduceT (+)
concatT = reduceT (++)
```

(e) Definieren Sie eine Funktion [5 Punkte]

```
toList :: Tree t -> [t]
```

die eine Liste aller im Baum gespeicherten Werte zurückgibt.

Hinweis: Verwenden Sie hierzu `concatT` und `mapT`.

Beispiellösung:

```
toList = concatT . mapT (\x -> [x])
```

Aufgabe 2 (Prolog, Nichtdeterminismus) [20 Punkte]

Ein Getränkeautomat zahlt Wechselgeld in Münzen aus. Dazu muss er mit den Münzen aus seinem Münzvorrat den auszuzahlenden Gesamtbetrag zusammensetzen. In Prolog lassen sich Wechselgeld und Münzvorrat als *Assoziativlisten* darstellen, in denen zu jedem Münzwert festgehalten wird, wieviele Münzen dieses Werts vorhanden sind. Zum Beispiel entspricht

```
[(50,2),(20,2),(10,1),(5,3)]
```

zwei 50-Cent-Münzen, zwei 20-Cent-Münzen, einer 10-Cent-Münze und drei 5-Cent-Münzen.

(a) Schreiben Sie zunächst ein Prädikat [7 Punkte]

```
remove(Coins,C,R)
```

welches besagt, dass die Assoziativliste `R` verbleibender Münzen durch Entfernung einer Münze vom Wert `C` aus der Assoziativliste `Coins` entsteht. Beispielanfrage:

```
?remove([(50,2),(20,2),(10,1),(5,3)],C,R).
  C = 50, R = [(50,1),(20,2),(10,1),(5,3)] ;
  C = 20, R = [(50,2),(20,1),(10,1),(5,3)] ;
  C = 10, R = [(50,2),(20,2),(10,0),(5,3)] ;
  C = 5,  R = [(50,2),(20,2),(10,1),(5,2)] ;
  false.
```

Hinweis: Orientieren Sie sich an dem Prädikat `delete(L,X,R)` aus der Vorlesung. Es besagt, dass die Liste `R` durch Entfernung von `X` aus `L` entsteht und wurde folgendermaßen definiert:

```
delete([X|L],X,L).
delete([X|L],Y,[X|R]) :- delete(L,Y,R).
```

(b) Nutzen Sie `remove` zur Definition eines Prolog-Prädikats [13 Punkte]

```
change(X,Av,Change)
```

das alle möglichen Kombinationen `Change` generiert, aus der Assoziativliste `Av` verfügbarer Münzen Wechselgeld vom Gesamtbetrag `X` herauszugeben. Beispielanfrage:

```
?change(100,[(50,2),(20,2),(10,1),(5,3)],Change).
  Change = [(50,2)] ;
  Change = [(10,1),(20,2),(50,1)] ;
  Change = [(5,2),(20,2),(50,1)] ;
  ...
```

Achtung: Verwenden Sie das vordefinierte Prädikat `put(Coins,C,Cnew)` zum Hinzufügen einer Münze `C` zur Assoziativliste `Coins`. Beispielanfrage mit `put`:

```
?put([(50,2),(20,3),(10,1),(5,3)],2,Cnew).
  Cnew=[(50,2),(20,3),(10,1),(5,3),(2,1)].

?put([(50,2),(20,3),(10,1),(5,3)],20,Cnew).
  Cnew=[(50,2),(20,4),(10,1),(5,3)].
```

Beispiellösung:

```
remove([(X,A)|L],X,[(X,ANew)|L]) :- A>0, ANew is A-1.
remove([X|L],Y,[X|L1]) :- remove(L,Y,L1).

change(0,_,[]).
change(X,CoinsAvailable,ChangeNew) :-
   remove(CoinsAvailable,C,CANew), C=<X, XNew is X-C,
   change(XNew,CANew,Change), put(Change,C,ChangeNew).
```

Aufgabe 3 (λ-Kalkül) [5 Punkte]

Turing fand folgenden Fixpunktkombinator:

$$\Theta = (\lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{y}\ (\texttt{x}\ \texttt{x}\ \texttt{y}))\ (\lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{y}\ (\texttt{x}\ \texttt{x}\ \texttt{y}))$$

Zeigen Sie per β-Reduktion, dass für beliebige λ-Terme F gilt[1]:

$$\Theta\ F \Rightarrow^* F\ (\Theta\ F)$$

Beispiellösung: Sei $\Theta_0 := (\lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{y}\ (\texttt{x}\ \texttt{x}\ \texttt{y}))$. Dann:

$$\begin{array}{rcl}
\Theta\ F & = & ((\lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{y}\ (\texttt{x}\ \texttt{x}\ \texttt{y}))\ \Theta_0)\ F \\
 & \Rightarrow & (\lambda\texttt{y}.\ \texttt{y}\ (\Theta_0\ \Theta_0\ \texttt{y}))\ F \\
 & \Rightarrow & F\ (\Theta_0\ \Theta_0\ F) \\
 & = & F\ (\Theta\ F)
\end{array}$$

[1] damit ist Θ in gewissem Sinn stärker als Y, denn für Y gilt nur $Y\ F \Rightarrow^* X \Leftarrow F\ (Y\ F)$ für gewissen Term X

Aufgabe 4 (λ-Kalkül, Typinferenz) [20 Punkte]

In der Vorlesung wurde gezeigt: $S\ K\ K \Rightarrow^* I$
Dabei waren S, K, I definiert als: $S = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \lambda\mathtt{z}.\ \mathtt{x\ z\ (y\ z)}$ $\quad K = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}$ $\quad I = \lambda\mathtt{x}.\ \mathtt{x}$

Zeigen Sie, dass der Term $\ \mathbf{let}\ \mathtt{k} = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}\ \mathbf{in}\ \mathtt{s\ k\ k}\ $ unter der Annahme $\Gamma_0 = \mathtt{s} : \forall\alpha,\beta,\gamma.\,(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma)$ einen allgemeinsten Typ $\ \alpha \to \alpha\ $ hat, indem Sie die unten dargestellte Typinferenz an den grau hervorgehobenen Stellen weiterführen:

(a) Bestimmen Sie einen allgemeinsten Unifikator $\sigma_{\mathtt{let}}$ für $C_{\mathtt{let}}$ sowie Γ mittels der Typabstraktion von $\sigma_{\mathtt{let}}(\alpha_0)$ [5 Punkte]

$$\dfrac{\dfrac{\dfrac{\Gamma_0, \mathtt{x}:\alpha, \mathtt{y}:\beta \vdash \mathtt{x}:\alpha_2}{\Gamma_0, \mathtt{x}:\alpha \vdash \lambda\mathtt{y}.\ \mathtt{x}:\alpha_1}}{\Gamma_0 \vdash \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}:\alpha_0} \qquad \dfrac{\dots}{\Gamma \vdash \mathtt{s\ k\ k}:\beta_0}}{\Gamma_0 \vdash \mathbf{let}\ \mathtt{k} = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}\ \mathbf{in}\ \mathtt{s\ k\ k}:\beta_0}$$

$C_{\mathtt{let}} = \{\alpha_0 = \alpha \to \alpha_1, \alpha_1 = \beta \to \alpha_2, \alpha_2 = \alpha\}$

$\sigma_{\mathtt{let}} = [\alpha_0 \Rightarrow \boxed{}, \alpha_1 \Rightarrow \boxed{}, \alpha_2 \Rightarrow \boxed{}]$

$\Gamma = \boxed{}$

(b) Vervollständigen Sie den unten dargestellten rechten Teil-Inferenzbaum sowie die dazugehörende Constraintmenge C. Vervollständigen Sie weiterhin den Unifikator σ für C. [15 Punkte]

$$\dfrac{\dots \qquad \dfrac{\dfrac{\dfrac{\Gamma(\mathtt{s}) = \forall\alpha,\beta,\gamma.\ (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma) \succeq (\beta_3 \to \beta_4 \to \beta_5) \to (\beta_3 \to \beta_4) \to (\beta_3 \to \beta_5)}{\Gamma \vdash \mathtt{s}:\beta_2 \to \beta_1 \to \beta_0} \qquad \dfrac{\Gamma(\mathtt{k}) = \boxed{} \succeq \boxed{}}{\Gamma \vdash \mathtt{k}:\beta_2}}{\Gamma \vdash \mathtt{s\ k}:\beta_1 \to \beta_0} \qquad \dfrac{\Gamma(\mathtt{k}) = \boxed{} \succeq \boxed{}}{\Gamma \vdash \mathtt{k}:\beta_1}}{\Gamma \vdash \mathtt{s\ k\ k}:\beta_0}}{\Gamma_0 \vdash \mathbf{let}\ \mathtt{k} = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}\ \mathbf{in}\ \mathtt{s\ k\ k}:\beta_0}$$

$$C = \left\{\begin{array}{lcl} \beta_2 \to \beta_1 \to \beta_0 & = & (\beta_3 \to \beta_4 \to \beta_5) \to (\beta_3 \to \beta_4) \to (\beta_3 \to \beta_5) \\ \beta_2 & = & \boxed{} \\ \beta_1 & = & \boxed{} \end{array}\right\} \qquad \sigma = \left[\begin{array}{ll} \beta_0 \Rightarrow \alpha \to \alpha, & \beta_5 \Rightarrow \alpha, \\ \beta_1 \Rightarrow \boxed{}, & \boxed{} \Rightarrow \boxed{}, \\ \beta_2 \Rightarrow \boxed{}, & \boxed{} \Rightarrow \boxed{}, \\ \beta_3 \Rightarrow \alpha, & \boxed{} \Rightarrow \boxed{}, \\ \beta_4 \Rightarrow \beta \to \alpha, & \boxed{} \Rightarrow \boxed{} \end{array}\right]$$

Aufgabe 4 (λ-Kalkül, Typinferenz) [20 Punkte]

In der Vorlesung wurde gezeigt: $S\ K\ K \Rightarrow^* I$
Dabei waren S, K, I definiert als: $S = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \lambda\mathtt{z}.\ \mathtt{x\ z\ (y\ z)}$ $\quad K = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}$ $\quad I = \lambda\mathtt{x}.\ \mathtt{x}$

Zeigen Sie, dass der Term $\mathbf{let}\ \mathtt{k} = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}\ \mathbf{in}\ \mathtt{s\ k\ k}$ unter der Annahme $\Gamma_0 = \mathtt{s} : \forall\alpha,\beta,\gamma.\,(\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma)$ einen allgemeinsten Typ $\alpha \to \alpha$ hat, indem Sie die unten dargestellte Typinferenz an den grau hervorgehobenen Stellen weiterführen:

(a) Bestimmen Sie einen allgemeinsten Unifikator $\sigma_{\mathtt{let}}$ für $C_{\mathtt{let}}$ sowie Γ mittels der Typabstraktion von $\sigma_{\mathtt{let}}(\alpha_0)$ [5 Punkte]

$$\dfrac{\dfrac{\dfrac{\Gamma_0, \mathtt{x}:\alpha, \mathtt{y}:\beta \vdash \mathtt{x}:\alpha_2}{\Gamma_0, \mathtt{x}:\alpha \vdash \lambda\mathtt{y}.\ \mathtt{x}:\alpha_1}}{\Gamma_0 \vdash \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}:\alpha_0} \qquad \dfrac{\dots}{\Gamma \vdash \mathtt{s\ k\ k}:\beta_0}}{\Gamma_0 \vdash \mathbf{let}\ \mathtt{k} = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}\ \mathbf{in}\ \mathtt{s\ k\ k}:\beta_0}$$

$$C_{\mathtt{let}} = \{\alpha_0 = \alpha \to \alpha_1, \alpha_1 = \beta \to \alpha_2, \alpha_2 = \alpha\}$$

$$\sigma_{\mathtt{let}} = [\alpha_0 \Leftarrow \alpha \to \beta \to \alpha, \alpha_1 \Leftarrow \beta \to \alpha, \alpha_2 \Leftarrow \alpha]$$

$$\Gamma = \Gamma_0, \mathtt{k} : \forall\alpha,\beta.\alpha \to \beta \to \alpha$$

(b) Vervollständigen Sie den unten dargestellten rechten Teil-Inferenzbaum sowie die dazugehörende Constraintmenge C. Vervollständigen Sie weiterhin den Unifikator σ für C. [15 Punkte]

$$\dfrac{\dots \qquad \dfrac{\dfrac{\dfrac{\Gamma(\mathtt{s}) = \forall\alpha,\beta,\gamma.\ (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma) \succeq (\beta_3 \to \beta_4 \to \beta_5) \to (\beta_3 \to \beta_4) \to (\beta_3 \to \beta_5)}{\Gamma \vdash \mathtt{s}:\beta_2 \to \beta_1 \to \beta_0} \qquad \dfrac{\Gamma(\mathtt{k}) = \forall\alpha,\beta.\alpha \to \beta \to \alpha \succeq \beta_6 \to \beta_7 \to \beta_6}{\Gamma \vdash \mathtt{k}:\beta_2}}{\Gamma \vdash \mathtt{s\ k}:\beta_1 \to \beta_0} \qquad \dfrac{\Gamma(\mathtt{k}) = \forall\alpha,\beta.\alpha \to \beta \to \alpha \succeq \beta_8 \to \beta_9 \to \beta_8}{\Gamma \vdash \mathtt{k}:\beta_1}}{\Gamma \vdash \mathtt{s\ k\ k}:\beta_0}}{\Gamma_0 \vdash \mathbf{let}\ \mathtt{k} = \lambda\mathtt{x}.\ \lambda\mathtt{y}.\ \mathtt{x}\ \mathbf{in}\ \mathtt{s\ k\ k}:\beta_0}$$

$$C = \left\{\begin{array}{lcl} \beta_2 \to \beta_1 \to \beta_0 & = & (\beta_3 \to \beta_4 \to \beta_5) \to (\beta_3 \to \beta_4) \to (\beta_3 \to \beta_5) \\ \beta_2 & = & \beta_6 \to \beta_7 \to \beta_6 \\ \beta_1 & = & \beta_8 \to \beta_9 \to \beta_8 \end{array}\right\}$$

$$\sigma = \left[\begin{array}{ll} \beta_0 \Leftarrow \alpha \to \alpha, & \beta_5 \Leftarrow \alpha, \\ \beta_1 \Leftarrow \alpha \to \beta \to \alpha, & \beta_6 \Leftarrow \alpha, \\ \beta_2 \Leftarrow \alpha \to (\beta \to \alpha) \to \alpha, & \beta_7 \Leftarrow \beta \to \alpha, \\ \beta_3 \Leftarrow \alpha, & \beta_8 \Leftarrow \alpha, \\ \beta_4 \Leftarrow \beta \to \alpha, & \beta_9 \Leftarrow \beta \end{array}\right]$$

Aufgabe 5 (C, Zeiger-Arithmetik, Arrays) [10 Punkte]

Welche Ausgabe erzeugt das folgende C-Programm? Begründen Sie Ihre Antwort kurz und machen Sie Ihren Lösungsweg deutlich, indem Sie unter die Programmzeilen jeweils Ihre Auswertung schreiben.

Hinweis: `printf("%i",i)` gibt den Zahlenwert eines Integers i aus.

```
#include <stdio.h>

int global[] = {1, 2, 3, 4, 5};

int *magic(int x[], int y) {
        printf("m");

        global[1] = *(global + y) + 3;

        return &x[y - 2];

}

int main(void) {
        printf("%i", *magic(&global[1], *(global + 1)));

        return 0;
}
```

Beispiellösung:

- Ausgabe: `m6`
- `&global[1]` liefert die Adresse von `global` ab Indexposition 1
- `*(global+1)` liefert den Wert von `global` an Indexposition 1, also 2
- Das Programm startet in main. Main ruft die Prozedur `magic` auf mit den Parametern
  `x = [2,3,4,5]` (d.h. alias von `global` ab Indexposition 1) und
  `y = 2` (d.h. Wert von `global` an Indexposition 1).
- Dort zunächst Ausgabe von "m“
- `*(global+y)` ergibt dann `*(global+2)` , d.h. Wert von `global` an Indexposition 2, also 3
- Danach wird `global` an Position 1 aktualisiert mit 3+3 = 6; d.h. `global = [1,`**`6`**`,3,4,5]`.
- Mit `&x[y-2]` → `&x[2-2]` → `&x[0]`
  wird die Adresse des ersten Elements von `x` (=6) zurückgegeben.
- `printf` dereferenziert die übergebene Adresse und gibt den Wert 6 aus.

Aufgabe 6 (MPI, kollektive Operationen) [10 Punkte]

Es sei a ein **int**-Array der Länge n, und n ein Vielfaches von 3. Die Berechnung des **Durchschnitts** von a soll auf die 3 Knoten der Communication-Group comm (Rank 0, 1, 2) verteilt werden:

$$\bar{\mathtt{a}} = \frac{1}{n}\sum_{0\leq i<n}\mathtt{a}[i] = \frac{1}{n}\left(\underbrace{\sum_{0\leq i<\frac{n}{3}}\mathtt{a}[i]}_{\textit{Knoten 0}} + \underbrace{\sum_{\frac{n}{3}\leq i<\frac{2n}{3}}\mathtt{a}[i]}_{\textit{Knoten 1}} + \underbrace{\sum_{\frac{2n}{3}\leq i<n}\mathtt{a}[i]}_{\textit{Knoten 2}} \right)$$

Auf dem Masterknoten 0 liegt a vor. Die Berechnung soll folgendermaßen ablaufen:

- Jeder Knoten (0,1,2) bekommt von Knoten 0 den Wert n mittels einer kollektiven Operation (**A**) mitgeteilt.
- Knoten 0 verteilt mit Hilfe einer anderen kollektiven Operation (**B**) an jeden Knoten (0,1,2) einen gleich großen Abschnitt des Arrays (jeweils $\frac{n}{3}$ Arrayelemente).
- Jeder Knoten (0,1,2) berechnet in Variable sum die Summe der ihm übermittelten Werte. Aus den Teilsummen wird mit einer einzigen kollektiven Operation (**C**) in Variable total auf Knoten 0 die Gesamtsumme gebildet.

Anschließend dividiert Knoten 0 die Gesamtsumme durch n.

Aufgaben:

(a) Nennen Sie für das beschriebene Verfahren geeignete kollektive MPI Operationen (**A**, **B** und **C**). Geben Sie jeweils den **Namen** der Operation an. [3 Punkte]

- A:
- B:
- C:

(b) Geben Sie für den Fall **C** zudem eine sinnvolle, vollständige **Parameterbelegung** an. [7 Punkte]

Beispiellösung:

(a)
- A: `MPI_BCAST`
- B: `MPI_SCATTER`
- C: `MPI_REDUCE`

(b) `MPI_Reduce(&sum,&total,1,MPI_INT,MPI_SUM,0,comm)`

Aufgabe 7 (X10) [10 Punkte]

Folgendes Programm gibt alle Quadratzahlen $< 100^2$ aus.

```
1  public class Map {
2          public static def map[T](array: Array[T], f:(i:T) => T):void {
3                  for (i in array) {
4                          array(i) = f(array(i));
5                  }
6          }
7
8          public static def main(argv:Array[String]) {
9                  val arr = new Array[Int](100, (i:Int) => i);
10
11                 map(arr, (i:Int) => (i*i));
12                 for(i in arr) { Console.OUT.println(arr(i)); }
13         }
14 }
```

Welche der folgenden Modifikationen von Zeile 3 sind geeignet, das Programm zu parallelisieren? Begründen Sie ihre Antwort kurz. Welche Probleme gibt es bei den nicht geeigneten Modifikationen?

(a) **async for** (i **in** array) **async** ☐ Geeignet ☐ Nicht geeignet

(b) **async for** (i **in** array) **finish** ☐ Geeignet ☐ Nicht geeignet

(c) **finish async for** (i **in** array) **async** ☐ Geeignet ☐ Nicht geeignet

(d) **finish for** (i **in** array) **async** ☐ Geeignet ☐ Nicht geeignet

(e) **for** (i **in** array) **async** ☐ Geeignet ☐ Nicht geeignet

Beispiellösung:

Ein „`finish` vor dem `async`“ ist zwingend notwendig, damit auf den Abschluss der `map`-Berechnung gewartet wird, bevor der Kontrollfluss weitergeht. Bei a), b) und e) fehlt dieses `finish`, weshalb es zur Ausgabe falscher (bzw. noch nicht vollständig berechneter) Werte kommen kann. Daher sind a), b), und e) nicht geeignet.

Ein `finish` zu Beginn stellt sicher, dass alle nachfolgenden (auch transitiv abgespaltenen) Activities beendet werden, bevor der Kontrollfluss weiter geht. Das ist bei c) und d) der Fall, daher sind sie geeignet.

Aufgabe 8 (Syntaktische Analyse) [20 Punkte]

Die Syntax für (stark vereinfachte) SQL-Anfragen ist:

$$\begin{array}{rcl} \mathit{Query} & \rightarrow & \texttt{\textbf{SELECT id FROM id WHERE}}\ \mathit{Cond} \\ \mathit{Cond} & \rightarrow & \texttt{\textbf{id}}\ \mathit{Cond'} \\ \mathit{Cond'} & \rightarrow & \texttt{\textbf{= id}} \mid \texttt{\textbf{< id}} \end{array}$$

(a) Geben Sie eine *abstrakte* Syntax für solche SQL-Anfragen an. [4 Punkte]
Verwenden Sie die Notation der Vorlesung (siehe Folie 343).

Beispiellösung:

$$\begin{array}{rcl} \mathit{Query} & :: & \texttt{\textbf{id id}}\ \mathit{Cond} \\ \\ \mathit{Cond} & = & \mathit{EqCond} \mid \mathit{LtCond} \\ \mathit{EqCond} & :: & \texttt{\textbf{id id}} \\ \mathit{LtCond} & :: & \texttt{\textbf{id id}} \end{array}$$

(b) Vervollständigen Sie den unten abgebildeten Parser für SQL-Anfragen mit rekursivem Abstieg. Geben Sie dazu fehlende Prozeduren in Pseudo-Code an. [16 Punkte]

Die Prozedur `parseQuery` soll als Ergebnis einen abstrakten Syntaxbaum zurückliefern. Verwenden Sie Klassen und Konstruktoren, die der abstrakten Syntax entsprechen. Die globale Variable `token` enthält immer das aktuelle Token. Dieses besitzt die Methode `getType()`, die einen der folgenden Tokentypen zurückgibt:

`ID`	für Namen von Spalten bzw. Tabellen
`SELECT`, `FROM` und `WHERE`	für die entsprechenden Schlüsselworte
`EQ`, `LT`	für **=** bzw. **<**

Für `ID`-Token liefert `getString()` die erkannte Zeichenkette. Die globale Prozedur `nextToken()` fordert das nächste Token an. Brechen Sie bei Parsefehlern durch Aufruf der globalen `error()`-Prozedur ohne Fehlermeldung ab.

```
Query parseQuery() {
  String col,table;
  if (token.getType() == SELECT) nextToken(); else error();
  if (token.getType() == ID) {
    col = token.getString();
    nextToken();
  } else error();
  if (token.getType() == FROM) nextToken(); else error();
  if (token.getType() == ID) {
    table = token.getString();
    nextToken();
  } else error();
  if (token.getType() == WHERE) nextToken(); else error();
  return new Query(col,table,parseCond());
}
```

Beispiellösung:

```
Cond parseCond(){
  String left;
  if (token.getType() == ID) {
    left = token.getString();
    nextToken();
  } else error();
  return parseCond'(left);
}

Cond parseCond'(String left) {
  String right;
  switch (token.getType()) {
    case EQ:
      nextToken();
      if (token.getType() == ID) {
        right = token.getString();
        nextToken();
        return new EqCond(left,right);
      } else error();
    case LT:
      nextToken();
      if (token.getType() == ID) {
        right = token.getString();
        nextToken();
        return new LtCond(left,right);
      } else error();
    default:
      error();
  }
}
```