Karlsruher Institut für Technologie

Institut für Programmstrukturen und Datenorganisation
Prof. Dr.-Ing. Gregor Snelting
gregor.snelting@kit.edu

Prof. Dr. Ralf Reussner
reussner@kit.edu

Klausur Programmierparadigmen — Beispiellösung

WS2010/11, 5. April 2011, 14:30 - 16:00 Uhr

Zugelassene Hilfsmittel: Vorlesungsfolien, Übungsblätter, eigene Aufzeichnungen
Die Verwendung von Büchern, Materialien aus dem Internet und elektronischen Geräten ist verboten.

Bearbeitungszeit: 90 min

Aufgabe 1 (Listen in Haskell, lazy evaluation) [15 Punkte]

(a) Definieren Sie eine Haskellfunktion [5 Punkte]

```
merge :: Ord t =>[t] -> [t] -> [t],
```

die zwei sortierte, möglicherweise unendliche Listen zu einer sortierten, möglicherweise unendlichen Liste vereinigt. Verwenden Sie Pattern Matching.

(b) Definieren Sie eine Haskellfunktion [10 Punkte]

```
primepowers :: Integer -> [Integer],
```

die für einen gegebenen Parameter n die unendliche Liste der ersten n Potenzen aller Primzahlen berechnet, aufsteigend sortiert. D.h., `primepowers n` enthält in aufsteigender Reihenfolge genau die Elemente der Menge

$$\{p^i \mid p \text{ Primzahl}, \ 1 \leq i \leq n\}$$

Verwenden Sie die unendliche, aufsteigend sortierte Liste `primes :: [Integer]` aller Primzahlen aus der Vorlesung (die Sie nicht angeben müssen) und `merge`.
Hinweis: `a ^ b` berechnet die `b`-te Potenz von `a` in Haskell.

Aufgabe 2 (λ-Kalkül, Church-Zahlen, Funktionen höherer Ordnung) [10 Punkte]

Bekanntlich kann man die natürlichen Zahlen im λ-Kalkül codieren als:

$$n \equiv \lambda \mathrm{s}.\ \lambda \mathrm{z}.\ \mathrm{s}^n\ z.$$

Diese Church-Zahlen haben den (polymorphen) Typ

```
type Church t = (t -> t) -> t -> t.
```

Geben Sie zwei Haskellfunktionen

```
int2church :: Integer -> Church t
church2int :: Church Integer -> Integer
```

an, die gewöhnliche natürliche Zahlen in Church-Zahlen verwandeln und umgekehrt.
Hinweis: `church2int` lässt sich ohne Rekursion definieren.

Aufgabe 3 (Typinferenz) [15 Punkte]

(a) Geben Sie eine Typisierungsregel für den Ausdruck [3 Punkte]

$$\textbf{if}\ b\ \textbf{then}\ x\ \textbf{else}\ y$$

an. b muss boolesch sein; die Typen in `then`- und `else`-Zweig müssen gleich sein und legen den Gesamttyp fest.

(b) Betrachten Sie den Ausdruck [12 Punkte]

$$\textbf{let}\ \mathtt{f} = \lambda \mathtt{x}.\ \mathtt{x}\ \textbf{in}\ \lambda \mathtt{b}.\ \mathtt{b}\ (\mathtt{f}\ 0)\ (\mathtt{f}\ \mathtt{true})$$

Ergänzen Sie in den hellgrauen Feldern die Typherleitung dieses Ausdrucks auf der nächsten Seite.
Berechnen Sie dabei den allgemeinst möglichen Typen für `f` in der **`let`**-Bindung.
Verwenden Sie als Typregel für **`let`** die Regel mit Typabstraktionen (S. 192 im Skript).

Aufgabe 4 (Logische Programmierung) [15 Punkte]

(a) Definieren Sie ein Prolog-Prädikat [7 Punkte]

```
treemember(X,Tree),
```

das zum Suchen und Einfügen eines Elements `X` in einen (sortierten) binären Suchbaum `Tree` verwendet werden kann. Binäre Suchbäume werden als Terme

```
tree(Key, LeftSon, RightSon),
```

Blätter als (paarweise verschiedene) uninstanziierte Variablen dargestellt.

Eine fehlschlagende Suche fügt `X` in den Baum an der richtigen Stelle ein.

Beispiele:

```
?- treemember(5, tree(5, L, tree(8, M, N))).
yes

?- treemember(1, tree(5, L, tree(8, M, N))).
L = tree(1, _G292, _293)
```

(b) Wenden Sie den Patterson-Wegmann-Unifikationsalgorithmus auf folgendes Unifikationsproblem an. Können Sie einen mgu angeben? [8 Punkte]

$$g(U, V, U) \stackrel{?}{=} g(h(U), h(V), V)$$

Aufgabe 5 (C) [4 Punkte]

Welche Ausgabe erzeugt das folgende C-Programm? Begründen Sie kurz Ihre Antwort.

```c
#include <stdio.h>

char p[] = "HelloWorld";
int i[] = {2,1,3,5,6}, j = 4;

int main()
{
   printf(&(p[*(i + j)]));
   return 0;
}
```

Aufgabe 6 (X10) [4 Punkte]

Gegeben sei folgendes X10-Programm zur Verdopplung der Elemente eines Arrays.

```
public class ArrayDouble {
  public static def main(args:Array[String](1)) {
    val len = 10;
    val arr = new Array[Int](len, (i:int) => i);

    for ([k] in arr) async {
      arr(k) *= 2;
    }
    if (arr(2) != 4)
      throw new RuntimeException("Expected␣4,␣was␣" + arr(2));
  }
}
```

(a) Wodurch kann es passieren, dass die Exception geworfen wird? [3 Punkte]

(b) Verändern Sie das Programm so, dass die **if**-Anweisung erst ausgeführt wird, nachdem alle Verdopplungen ausgeführt wurden. Die Verdopplungen selbst sollen weiterhin parallel ausgeführt werden. [1 Punkt]

Aufgabe 7 (MPI) [12 Punkte]

In einem Aufruf

```
int MPI_Scatter (void *sendbuf, int sendcount, MPI_Datatype sendtype,
                 void *recvbuf, int recvcount, MPI_Datatype recvtype,
                 int root, MPI_Comm comm)
```

bestimmt `sendcount`, wieviele Elemente an jeden einzelnen Prozess in `comm` gesendet werden.

`comm` sei ein Kommunikator der Größe `k`. `buf_1` sei die Anfangsadresse eines Puffers, in dem k^2 **int**-Werte abgelegt sind und zwar die zeilenweise gelesenen Elemente einer $k \times k$-Matrix. `buf_2` und `buf_3` seien hinreichend groß (jeweils $\geq k$).

Betrachten Sie das folgende Programmstück:

```
MPI_Scatter(buf_1, k, MPI_INT, buf_2, k, MPI_INT, 0, comm);
for (int i = 0; i < k; i++)
{
  MPI_Scatter(buf_2, 1, MPI_INT, &(buf_3[i]), 1, MPI_INT, i, comm);
}
```

(a) In welchem Zusammenhang stehen die Daten in `buf_1` und `buf_3` nach der Ausführung des gesamten Programmstücks? Beschreiben Sie den Zusammenhang kurz und präzise (in maximal 3 Sätzen). [?? Punkte]

(b) Ersetzen Sie die ganze **for**-Schleife durch genau einen Aufruf einer kollektiven MPI-Operation. Diese Ersetzung muss so gewählt sein, dass nach der Ausführung des gesamten veränderten Programmstücks der Inhalt von `buf_3` genau dem Inhalt nach der Ausführung der ursprünglichen Version ist. Geben Sie den genauen Aufruf mit allen notwendigen Parametern an. [?? Punkte]

Aufgabe 8 (Compilerbau) [15 Punkte]

In dieser Aufgabe soll ein Parser mit rekursivem Abstieg für Typen des λ-Kalküls entwickelt werden, der den abstrakten Syntaxbaum aufbaut. Die Syntax für die Typen ist:

$$\mathit{Type} \rightarrow \mathsf{id} \mid (\ \mathit{Type}\ \rightarrow\ \mathit{Type}\ )$$

Bemerkung: Der Funktionstyp muss hier (anders als in Haskell) in Klammern geschrieben werden, um eine LL(1)-Analyse zu ermöglichen.

Geben Sie einen Parser mit rekursivem Abstieg als Pseudo-Code an. Die Parse-Funktion soll als Ergebnis einen abstrakten Syntaxbaum zurückliefern, zu dessen Aufbau Sie auf die Konstruktorfunktionen `BaseType(name)` für Basistypen sowie `FuncType(type1,type2)` für Funktionstypen zurückgreifen können. Verwenden Sie als Tokens:

id	für Namen von Basistypen
arrow	Pfeil nach rechts in Funktionstypen $\rightarrow$
lp, rp	für öffnende bzw. schließende Klammer (,)

Die globale Variable `token` enthält immer das aktuelle Token. Es besitzt die Methoden `getType()` für den Token-Typ (id, arrow, lp, rp) und `getString()` für die erkannte Zeichenkette. Die globale Methode `nextToken()` fordert das nächste Token an. Brechen Sie bei Parsefehlern durch Aufruf der globalen `error()`-Methode ohne Fehlermeldung ab.

Aufgabe 1 (Listen in Haskell, lazy evaluation) [15 Punkte]

Beispiellösung:

```
merge :: Ord t => [t] -> [t] -> [t]
merge [] ys = ys
merge xs [] = xs
merge (x : xs) (y : ys)
  | x < y     = x : merge xs (y : ys)
  | otherwise = y : merge (x : xs) ys

primepowers :: Integer -> [Integer]
primepowers n = foldr merge [] [ map (^p) primes | p <- [1..n] ]
```

Aufgabe 2 (λ-Kalkül, Church-Zahlen, Funktionen höherer Ordnung) [10 Punkte]

Beispiellösung:

```
type Church t = (t -> t) -> t -> t

int2church :: Integer -> Church t
int2church 0 s z = z
int2church n s z = int2church (n - 1) s (s z)

church2int :: Church Integer -> Integer
church2int n = n (+1) 0
```

Aufgabe 3 (Typinferenz) [15 Punkte]

(a) Geben Sie eine Typisierungsregel für den Ausdruck [3 Punkte]

Beispiellösung:

$$\frac{\Gamma \vdash b : \mathsf{bool} \qquad \Gamma \vdash x : \tau \qquad \Gamma \vdash y : \tau}{\Gamma \vdash \mathbf{if}\ b\ \mathbf{then}\ x\ \mathbf{else}\ y : \tau}$$

(b) Betrachten Sie den Ausdruck [12 Punkte]

$$\vdash \textbf{let}\ \mathtt{f} = \lambda \mathtt{x}.\ \mathtt{x}\ \textbf{in}\ \lambda \mathtt{b}.\ \mathtt{b}\ (\mathtt{f}\ 0)\ (\mathtt{f}\ \mathtt{true}) : \ (\mathsf{int} \to \mathsf{bool} \to \beta) \to \beta$$

$$\frac{\dfrac{(\mathtt{x} : \alpha)(\mathtt{x}) = \alpha}{\mathtt{x} : \alpha \vdash \mathtt{x} : \alpha \mid \emptyset} \quad \sigma \text{ mgu für } \emptyset}{\vdash \lambda \mathtt{x}.\ \mathtt{x} : \alpha \to \alpha \mid \emptyset}$$

$\sigma = []$

$\Gamma = \mathtt{f} : ta(\sigma(\alpha \to \alpha), []) = \mathtt{f} : \forall \alpha.\ \alpha \to \alpha$

$\Gamma' = \Gamma, b : \mathsf{int} \to \mathsf{bool} \to \beta$

$$\Gamma \vdash \lambda \mathtt{b}.\ \mathtt{b}\ (\mathtt{f}\ 0)\ (\mathtt{f}\ \mathtt{true}) : \ (\mathsf{int} \to \mathsf{bool} \to \beta) \to \beta$$

$$\Gamma' \vdash \mathtt{b}\ (\mathtt{f}\ 0)\ (\mathtt{f}\ \mathtt{true}) : \ \beta$$

$$\Gamma' \vdash \mathtt{b} : \ \mathsf{int} \to \mathsf{bool} \to \beta \qquad \Gamma' \vdash \mathtt{f}\ 0 : \mathsf{int} \qquad \Gamma' \vdash \mathtt{f}\ \mathtt{true} : \ \mathsf{bool}$$

$$\Gamma'(\mathtt{b}) = \mathsf{int} \to \mathsf{bool} \to \beta$$

$$\frac{\Gamma'(\mathtt{f}) = \forall \alpha.\ \alpha \to \alpha \qquad 0 \in \mathrm{Const}}{\Gamma' \vdash \mathtt{f} : \forall \alpha.\ \alpha \to \alpha \qquad \Gamma' \vdash 0 : \mathsf{int}}$$

$$\frac{\Gamma'(\mathtt{f}) = \forall \alpha.\ \alpha \to \alpha \qquad \mathtt{true} \in \mathrm{Const}}{\Gamma' \vdash \mathtt{f} : \forall \alpha.\ \alpha \to \alpha \qquad \Gamma' \vdash \mathtt{true} : \mathsf{bool}}$$

$$\Gamma'(\mathtt{f}) = \forall \alpha.\ \alpha \to \alpha$$

$$\Gamma' \vdash \mathtt{f} : \ \mathsf{bool} \to \mathsf{bool}$$

Aufgabe 4 (Logische Programmierung) [15 Punkte]

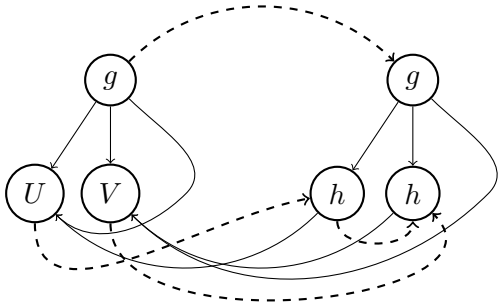
(a) Definieren Sie ein Prolog-Prädikat [7 Punkte]

Beispiellösung:

```
treemember(X,tree(K,L,R)) :- X = K.
treemember(X,tree(K,L,R)) :- X < K,treemember(X,L).
treemember(X,tree(K,L,R)) :- K < X,treemember(X,R).
```

(b) Wenden Sie den Patterson-Wegmann-Unifikationsalgorithmus auf folgendes Unifikationsproblem an. Können Sie einen mgu angeben? [8 Punkte]

Beispiellösung:



DAG-solved-form: $\mathtt{U} \mapsto \mathtt{h(V)}, \mathtt{V} \mapsto \mathtt{h(V)}$
mgu: keiner, da Zykel im Graphen

Aufgabe 5 (C) [4 Punkte]

Beispiellösung:

Ausgabe: `orld`

- `*(i+j)` ist das Element des Arrays `i` an Index-Position `j`, also 6
- `&(p[*(i+j)])` ist also ein Zeiger auf die Arrayzelle von `p` mit Index 6.
- `printf` nimmt einen Zeiger auf `char` und gibt den dort liegenden String bis zum Terminierungszeichen `0` aus.

Aufgabe 6 (X10) [4 Punkte]

Beispiellösung:

`async` wartet nicht auf die Beendigung der generierten Aktivität. Deswegen kann die **`if`**-Abfrage bereits ausgeführt werden, bevor alle Verdoppelungsaktivitäten abgeschlossen sind.

Lösung: `finish` vor die `for`-Schleife.

Aufgabe 7 (MPI) [12 Punkte]

Beispiellösung:

(a) `buf_3` des Knotens i enthält die i-te Spalte der Matrix, die ursrpünglich in `buf_1` auf dem Wurzelknoten `0` gespeichert war und immer noch ist.

Bemerkung: Laut Aufgabenbeschreibung enthalten alle Knoten anfangs in `buf_1` die gesamte Matrix, aber nur die Matrix des Wurzelknotens `0` wird verwendet.

(b) `MPI_Alltoall(buf_2, 1, MPI_INT, buf_3, 1, MPI_INT, comm);` Die Adressierung von `buf_3` braucht jetzt keine Zeigerarithmetik mehr.

Aufgabe 8 (Compilerbau) [15 Punkte]

Beispiellösung:

```
Node parseType() {
  switch (token.getType()) {
  case id:
    x = token.getString();
    nextToken();
    return new BaseType(x);
  case lp:
    nextToken();
    type1 = parseType();
    if (token.getType() == arrow) {
      nextToken();
    } else error();
    type2 = parseType();
    if (token.getType() == rp) {
        nextToken();
    } else error();
    return new FuncType(type1, type2);
  }
}
```