Karlsruher Institut für Technologie

Institut für Programmstrukturen und Datenorganisation

Prof. Dr.-Ing. Gregor Snelting gregor.snelting@kit.edu

Jun.-Prof. Dr. Oliver Hummel hummel@kit.edu

Klausur Programmierparadigmen — Beispiellösung

WS2013/14, 10. April 2014, 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, Streams) [10 Punkte]

(a) Definieren Sie eine unendliche Liste `hamming :: [Integer]` aller *Hamming*-Zahlen, also aller Zahlen h der Gestalt

$$h = 2^i \cdot 3^j \cdot 5^k \text{ mit } i, j, k \geq 0$$

Hinweis: Sie können die Definition per List Comprehensions umsetzen, *oder* folgende Eigenschaften der Hamming-Zahlen ausnutzen:

- 1 ist die erste Hamming-Zahl
- alle anderen Hamming-Zahlen sind von der Gestalt $2h$, $3h$ oder $5h$ für eine schon berechnete Hamming-Zahl h.

Verwenden Sie in diesem Fall die aus der Übung bekannte Funktion `merge :: Ord a =>[a] -> [a] -> [a]`, die zwei Streams vereinigt.

Beispiellösung:

```
hamming     = 1 : (map (2*) hamming) `merge`
                  (map (3*) hamming) `merge`
                  (map (5*) hamming)
```

Alternativ:

```
hamming = [ 2^i * 3^j * 5^k | n <- [0..],
                              i <- [0..n],
                              j <- [0..(n-i)],
                              k <- [n-i-j]
          ]
```

Alternativ:

```
hamming = 1:(concat [[2*h,3*h,5*h]| h <- hamming])
```

Aufgabe 2 (Haskell, Kombinatoren) [20 Punkte]

(a) Definieren Sie eine Funktion [8 Punkte]

```
intermsl :: (a -> b -> a) -> a -> [b] -> [a]
```

so dass `intermsl` $\oplus$ i l für eine Liste $l = [l_0, l_1, l_2, \ldots]$ die Liste

$$r = [i,\ i \oplus l_0,\ (i \oplus l_0) \oplus l_1,\ ((i \oplus l_0) \oplus l_1) \oplus l_2,\ \ldots]$$

mit $r_0 = \texttt{i}$, $r_{n+1} = r_n \oplus l_n$ berechnet. Dies soll auch für unendliche Listen funktionieren.

(b) Zur approximativen Lösung der durch f gegebenen Differentialgleichung [12 Punkte]

$$y'(x) = f(x, y(x))$$

mit dem Eulerverfahren bei Anfangswert $y(x_0) = y_0$ und Schrittweite h bildet man die Folgen $(y_n)_{n\in\mathbb{N}}$ und $(y'_n)_{n\in\mathbb{N}}$ der Funktionswerte bzw. Steigungen an den Stellen $x_n = x_0 + nh$, definiert durch:

$$\begin{aligned} y_{n+1} &= y_n + hy'_n \\ y'_n &= f(x_n, y_n) \end{aligned}$$

Hiermit hat y_n näherungsweise den Wert der gesuchten Funktion y an Stelle x_n: $y(x_n) \approx y_n$

Implementieren Sie das Eulerverfahren. Vervollständigen Sie hierzu die Funktion `euler`, indem Sie die Folgen $(x_n)_{n\in\mathbb{N}}$, $(y_n)_{n\in\mathbb{N}}$, $(y'_n)_{n\in\mathbb{N}}$ als unendliche Listen `xns, yns, yn's :: [Double]` definieren.

```
euler :: (Double -> Double -> Double) ->
          Double -> Double -> Double -> [Double]
euler f x0 y0 h = yns
  where  xns  =
         yns  =
         yn's =
```

Hinweis: Unter anderem könnten `iterate`, `zipWith` und/oder `intermsl` nützlich sein.

Beispiellösung:

```
intermsl :: (a -> b -> a) -> a -> [b] -> [a]
intermsl f i l =  i : rest l
 where
  rest []     = []
  rest (x:xs) = intermsl f (f i x) xs

euler f x0 y0 h = yns
  where  xns  = iterate (+h) x0
         yns  = intermsl (+) y0 (map (*h) yn's)
         yn's = zipWith f xns yns
```

Bemerkung: Folgende „natürliche“ Variante von `intermsl` funktioniert fast genauso wie die Beispiellösung, unterscheidet sich aber im Terminationsverhalten:

```
intermsl :: (a -> b -> a) -> a -> [b] -> [a]
intermsl f i []     = [i]
intermsl f i (x:xs) = i : intermsl f (f i x) xs
```

Anders als in der Beispiellösung muss hier – *bevor das erste Element `i` des Ergebnis berechnet werden kann* – durch das Pattern-Matching per `[]` bzw. `(x:xs)` auf dem Listen-Argument zunächst bestimmt werden, ob dieses Argument die leere Liste ist oder nicht.

Bei Definitionen wie z.B. `powersOf2 = intermsl (+) 1 powersOf2`, oder auch bei der Verwendung von `intermsl` in `euler`, bei denen ein Stream mit Hilfe von `intermsl` co-rekursiv durch sich selbst definiert werden soll, führt dies dazu, dass niemals ein Element der Ergebnisliste generiert werden kann.

Mit `intermsl` aus der Beispiellösung funktionieren solche Definitionen hingegen problemlos. Diese Variante ist auch als **`scanl`** bekannt.

Alternativ: Eine weitere schöne Lösung ist

```
intermsl :: (a -> b -> a) -> a -> [b] -> [a]
intermsl f i l = result
  where result = i : zipWith f result l
```

Aufgabe 3 (Natürliche Zahlen im λ-Kalkül, β-Reduktion) [15 Punkte]

Bekanntlich kann man im λ-Kalkül natürliche Zahlen n mittels Church-Zahlen c_n darstellen. Es gibt aber eine weitere Darstellung s_n für natürliche Zahlen n, die in gewisser Weise einfacher ist:

$$\begin{array}{rlr}
\mathsf{s}_0 = & & \lambda \mathtt{x}.\ \mathtt{x} \\
\mathsf{s}_1 = & & \langle \mathsf{c}_{\mathsf{false}}, \mathsf{s}_0 \rangle \\
\mathsf{s}_2 = & \langle \mathsf{c}_{\mathsf{false}}, \mathsf{s}_1 \rangle = & \langle \mathsf{c}_{\mathsf{false}}, \langle \mathsf{c}_{\mathsf{false}}, \mathsf{s}_0 \rangle \rangle \\
\mathsf{s}_3 = & \langle \mathsf{c}_{\mathsf{false}}, \mathsf{s}_2 \rangle = & \langle \mathsf{c}_{\mathsf{false}}, \langle \mathsf{c}_{\mathsf{false}}, \langle \mathsf{c}_{\mathsf{false}}, \mathsf{s}_0 \rangle \rangle \rangle \\
& \cdots &
\end{array}$$

Hierbei sei $\langle x, y \rangle$ eine Kurzschreibweise für Church-Paare $\lambda \mathtt{p}.\ \mathtt{p}\ x\ y$ und

$$\begin{aligned}
\mathsf{c}_{\mathsf{true}} &= \lambda \mathtt{t}.\ \lambda \mathtt{f}.\ \mathtt{t} \\
\mathsf{c}_{\mathsf{false}} &= \lambda \mathtt{t}.\ \lambda \mathtt{f}.\ \mathtt{f}
\end{aligned}$$

die bekannte Churchkodierung für Boolesche Werte. Nachfolgerfunktion und Test auf 0 lassen sich dann definieren als:

$$\begin{aligned}
\mathsf{succ} &= \lambda \mathtt{n}.\ \langle \mathsf{c}_{\mathsf{false}}, \mathtt{n} \rangle \\
\mathsf{isZero} &= \lambda \mathtt{n}.\ \mathtt{n}\ \mathsf{c}_{\mathsf{true}}
\end{aligned}$$

(a) Zeigen Sie: [6 Punkte]

$$\begin{array}{rll}
& \mathsf{isZero}\ \mathsf{s}_0 & \Rightarrow^* \mathsf{c}_{\mathsf{true}} \\
\text{und} & \mathsf{isZero}\ (\mathsf{succ}\ \mathsf{s}_0) & \Rightarrow^* \mathsf{c}_{\mathsf{false}}
\end{array}$$

(b) Geben Sie einen λ-Term pred an, so dass für alle $n \in \mathbb{N}$: [3 Punkte]

$$\mathsf{pred}\ \mathsf{s}_{n+1} \Rightarrow^* \mathsf{s}_n$$

(c) Geben Sie einen λ-Term add an, so dass für alle $m, n \in \mathbb{N}$: [6 Punkte]

$$\mathsf{add}\ \mathsf{s}_m\ \mathsf{s}_n \Rightarrow^* \mathsf{s}_{m+n}$$

Zur Erinnerung: Der λ-Kalkül erlaubt keine rekursiven Definitionen. Verwenden Sie stattdessen einen Fixpunktkombinator wie Y.

Fallunterscheidung `if` c_b `then` x `else` y für Terme c_b, die zu Church-Booleans reduzieren, schreibt sich einfach als $c_b\ x\ y$

Beispiellösung:

(a)

$$\begin{aligned}\mathsf{isZero}\ s_0 &\Rightarrow & s_0\ \mathsf{c_{true}}\\ &\Rightarrow & \mathsf{c_{true}}\end{aligned}$$

$$\begin{aligned}\mathsf{isZero}\ (\mathsf{succ}\ s_0) &\Rightarrow & \mathsf{isZero}\ \langle \mathsf{c_{false}}, \mathsf{s_0}\rangle\\ &\Rightarrow & \langle \mathsf{c_{false}}, \mathsf{s_0}\rangle\ \mathsf{c_{true}}\\ &\Rightarrow & \mathsf{c_{true}}\ \mathsf{c_{false}}\ \mathsf{s_0}\\ &\Rightarrow^2 & \mathsf{c_{false}}\end{aligned}$$

(b) $\mathsf{pred} = \lambda \mathtt{n}.\ \mathtt{n}\ \mathsf{c_{false}}$

(c) $\mathsf{Add} = \lambda \mathtt{add}.\ \lambda \mathtt{m}.\ \lambda \mathtt{n}.\ (\mathsf{isZero}\ \mathtt{m})\ \mathtt{n}\ (\mathsf{succ}\ (\mathtt{add}\ (\mathsf{pred}\ \mathtt{m})\ \mathtt{n}))$
$\mathsf{add} = \mathsf{Y}\ \mathsf{Add}$

Bemerkung: Obige Beispiellösung für (a) reicht volkommen. Eine noch ausführlichere Lösung ist:

$$\begin{aligned}&\mathsf{isZero}\ s_0 &&= (\lambda \mathtt{n}.\ \mathtt{n}\ \mathsf{c_{true}})\ s_0\\ \Rightarrow\ & s_0\ \mathsf{c_{true}} &&= (\lambda \mathtt{x}.\ \mathtt{x})\ \mathsf{c_{true}}\\ \Rightarrow\ & \mathsf{c_{true}}\end{aligned}$$

$$\begin{aligned}&\mathsf{isZero}\ (\mathsf{succ}\ s_0) &&= \mathsf{isZero}\ ((\lambda \mathtt{n}.\ \langle \mathsf{c_{false}}, \mathtt{n}\rangle)\ s_0)\\ \Rightarrow\ & \mathsf{isZero}\ \langle \mathsf{c_{false}}, \mathsf{s_0}\rangle &&= (\lambda \mathtt{n}.\ \mathtt{n}\ \mathsf{c_{true}})\ \langle \mathsf{c_{false}}, \mathsf{s_0}\rangle\\ \Rightarrow\ & \langle \mathsf{c_{false}}, \mathsf{s_0}\rangle\ \mathsf{c_{true}} &&= (\lambda \mathtt{p}.\ \mathtt{p}\ \mathsf{c_{false}}\ \mathsf{s_0})\ \mathsf{c_{true}}\\ \Rightarrow\ & \mathsf{c_{true}}\ \mathsf{c_{false}}\ \mathsf{s_0} &&= (\lambda \mathtt{t}.\ \lambda \mathtt{f}.\ \mathtt{t})\ \mathsf{c_{false}}\ \mathsf{s_0}\\ \Rightarrow\ & (\lambda \mathtt{f}.\ \mathsf{c_{false}})\ \mathsf{s_0}\\ \Rightarrow\ & \mathsf{c_{false}}\end{aligned}$$

Aufgabe 4 (Typinferenz) [10 Punkte]

Wir betrachten den Term

$$\lambda \texttt{a}.\ \texttt{a}\ \textit{true}$$

wobei die Konstante *true* den Typ $\tau_{true} = \mathsf{bool}$ hat.

(a) Geben Sie für diesen Term eine Typisierung an unter Verwendung der Regeln Var, Abs, App (Anwendungen von Const können Sie weglassen). [6 Punkte]

Stellen Sie das Gleichungssystem C für die Typvariablen auf und geben Sie einen allgemeinsten Unifikator an!

(b) Betrachten Sie nun [4 Punkte]

$$\lambda \texttt{x}.\ \textbf{let}\ \texttt{f} = \lambda \texttt{a}.\ \texttt{a}\ \textit{true}\ \textbf{in}\ \texttt{f}\ \texttt{x}$$

Was ist der polymorphe Typ von `f`? Geben Sie die Typabstraktion an. Was ist der Typ von `f x`? Warum ist der Typ von `f x` nicht polymorph (Antwort in einem Satz)?

Beispiellösung:

(a)

$$
\dfrac{\dfrac{\dfrac{(\texttt{a} : \alpha_2)\,(\texttt{a}) = \alpha_4}{\texttt{a} : \alpha_2 \vdash \texttt{a} : \alpha_4}\,\textit{Var} \qquad \texttt{a} : \alpha_2 \vdash \textit{true} : \mathsf{bool}}{\texttt{a} : \alpha_2 \vdash \texttt{a}\ \textit{true} : \alpha_3}\,\textit{App}}{\vdash \lambda\texttt{a}.\ \texttt{a}\ \textit{true} : \alpha_1}\,\textit{Abs}
$$

$$
\begin{aligned}
C &= \{\alpha_2 = \alpha_4, \quad \alpha_1 = \alpha_2 \to \alpha_3, \quad \alpha_4 = \mathsf{bool} \to \alpha_3\} \\
\sigma_C &= [\alpha_1 \Rightarrow (\mathsf{bool} \to \alpha_3) \to \alpha_3, \quad \alpha_2 \Rightarrow \mathsf{bool} \to \alpha_3, \quad \alpha_4 \Rightarrow \mathsf{bool} \to \alpha_3]
\end{aligned}
$$

(b) Der polymorphe Typ von `f` ist

$$
ta((\mathsf{bool} \to \alpha) \to \alpha, \texttt{x} : \mathsf{bool} \to \beta) \quad = \quad \forall \alpha.(\mathsf{bool} \to \alpha) \to \alpha
$$

Der Typ von `f x` ist β, wobei $\mathsf{bool} \to \beta$ der Typ von `x` sei.

Dieser ist nicht polymorph, da `x` λ-gebunden ist, also keinen polymorphen Typen hat, womit auch `f x` keinen polymorphen Typ hat.

Aufgabe 5 (Prolog, reguläre Ausdrücke) [15 Punkte]

Reguläre Ausdrücke lassen sich als Prolog-Terme in Präfixnotation darstellen. Bei Prolog-Systemen, welche die Zeichen $\cdot$, $\star$ und $\cup$ als Funktor erlauben, lassen sich z.B.

$a \cdot b \cdot c$	als	`·(a,·(b,c))`
a^*	als	`⋆(a)`
$\varepsilon \cup b$	als	`∪(ε,b)`
$a^* \cup (a \cdot b \cdot c) \cup (\varepsilon \cup b)^*$	als	`∪(⋆(a),∪(·(a,·(b,c)),⋆(∪(ε,b))))`

darstellen. Zeichen $a, b, c, \ldots \in \Sigma$ werden also als Prolog-*Atome* `a,b,c,...` dargestellt.

Ein regulärer Ausdruck α *akzeptiert* eine Folge von Zeichen, falls diese in der durch den Ausdruck definierten Sprache $L(\alpha)$ enthalten ist:

- Ausdruck ε akzeptiert die leere Zeichenfolge
- Ausdrücke a, b, c, … akzeptieren jeweils die Zeichenfolge „a“, „b“, „c“, …
- Ausdrücke $\alpha \cup \beta$ akzeptieren eine Zeichenfolge s, falls s durch α oder β akzeptiert wird
- Ausdrücke $\alpha \cdot \beta$ akzeptieren eine Zeichenfolge $s_1 \cdot s_2$, falls α die Folge s_1 und β die Folge s_2 akzeptiert
- Ausdrücke α^* akzeptieren
  - die leere Zeichenfolge, sowie
  - Zeichenfolgen $s_1 \cdot s_2$, falls s_1 nicht die leere Folge ist, α die Folge s_1 akzeptiert, und α^* die Folge s_2 akzeptiert

(a) Implementieren Sie ein Prolog-Prädikat `matches(Regexp,S)` das für Ausdrücke `Regexp` und Zeichenfolgen `S` bestimmt, ob `S` durch `Regexp` akzeptiert wird. Beispiel:

```
?matches(⋆(·(a,·(b,c))),[a,b,c]).    ?matches(⋆(·(a,·(b,c))),[]).
true .                                true .

?matches(⋆(·(a,·(b,c))),[a,b,c,d]).  ?matches(⋆(·(a,·(b,c))),[a,b,c,a,b,c]).
false.                                true .
```

Hinweis: Die Prädikate `append(S1,S2,S)`, **`not`** und `atom` könnten nützlich sein!

Beispiellösung:

```
matches(ε,[]).

matches(C,[C]) :- atom(C), !.

matches(∪(A,_),S) :- matches(A,S).
matches(∪(_,B),S) :- matches(B,S).

matches(·(A,B),S) :- append(S1,S2,S), matches(A,S1), matches(B,S2).

matches(⋆(_),[]).
matches(⋆(A),S) :- append(S1,S2,S), not(S1=[]), matches(A,S1), matches(⋆(A),S2).



% Alternative fuer zweite ⋆-Regel:
matches(⋆(A),S) :- matches(·(A,⋆(A)), S).
```

Bemerkung: Diese Implementierung ist nicht besonders effizient. Eine bessere Implementierung ergibt sich, wenn man zunächst ein Prädikat `matchesPrefix(Regexp,S,S2)` definiert, das für Ausdrücke `Regexp` und Zeichenfolen `S` erfüllt ist, falls ein Anfangsstück s_1 von $\mathtt{S} = s_1 \cdot s_2$ durch `Regexp` gematched wird. In diesem Fall enthält S2 dann die Restzeichenfolge s_2.

Aufgabe 6 (C: Precedence-Regel) [5 Punkte]

Benutzen Sie die in der Vorlesung (VL 4_4, Folie 26) vorgestellte Precedence-Regel nach Linden zum Lesen der folgenden C-Deklaration. Tragen Sie in untenstehender Tabelle jeweils ein, welches Token Sie mit Hilfe welcher Regel erkannt haben und wie es zur Benennung der Deklaration beiträgt.

```
static signed int *(*(*var [3][4])(long foo[]))[];
```

Beispiellösung:

Regel	**Bearbeitetes Token**	**Gelesener Text**
A	`var`	`var` is a
B.2.2	`[3]`	array of 3 of
B.2.2	`[4]`	array of 4
B.3	`*`	pointer to
B.2.1	`(...)`	a function taking . . . and returning
B.2.2	`long foo []`	an array of long (named foo)
B.3	`*`	pointer to
B.2.2	`[]`	an array of
B.3	`*`	pointer to
B.6	`static signed int`	a static signed int

Aufgabe 7 (MPI: Allgather & Alltoall) [10 Punkte]

Gegeben ist folgender Quelltext in MPI:

```
void foo(int argc, char *argv[])
{
    int myid, numprocs;
    int source,count;
    int buffer[1];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);

    source=0;
    count=1;
    if(myid == source){
      buffer[0] = 42;
    }

    MPI_Bcast(buffer,count,MPI_INT,source,MPI_COMM_WORLD);
    buffer[0] += myid;

    int allgath[numprocs];
    int alltoall[numprocs];

    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Allgather(buffer, count, MPI_INT, allgath, count, MPI_INT,
       MPI_COMM_WORLD);

    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Alltoall(allgath, count, MPI_INT, alltoall, count, MPI_INT,
       MPI_COMM_WORLD);

    MPI_Finalize();
}
```

a) Nehmen Sie an, dass das Programm auf einem 4-Kernsystem ausgeführt wird. Wie lautet der Inhalt der Variablen `allgath` nach dem Aufruf von MPI_AllGather und `alltoall` nach dem Aufruf von MPI_Alltoall jeweils für jeden Prozess? [6 Punkte]

Nach MPI_Allgather (**Beispiellösung**):

Prozess Nr.	Werteinhalt
1	42 43 44 45
2	42 43 44 45
3	42 43 44 45
4	42 43 44 45

Nach MPI_Alltoall (**Beispiellösung**):

Prozess Nr.	Werteinhalt
1	42 42 42 42
2	43 43 43 43
3	44 44 44 44
4	45 45 45 45

b) Füllen Sie mit Hilfe des folgenden Code-Fragments zunächst die linke Tabelle aus und fügen Sie anschließend die MPI-Aufrufe (3 mal 1 Aufruf) in den Code ein, mit denen sich das Ergebnis der rechten Tabelle realisieren lässt. Hinweis: Das System läuft mit 3 CPU-Kernen. [4 Punkte]

$\overrightarrow{data}$ (**Beispiellösung**)

P0	33	42	313	$\Rightarrow$	4	42	43	P0
P1	4	43	4711					P1
P2	87	42	43					P2

```
int numtasks, rank, tag = 11111;
int data[3];
MPI_Status status;
/* MPI initialisieren */
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks); //numtasks == 3

if (rank == 0)
{
        int i = 0;
        int input[3][3] = {{33, 42, 313},{4, 43, 4711},
        {87, 42, 43}};
        for (i = 0; i < numtasks; ++i) //numtasks == 3
        {
```

Beispiellösung:

```
                MPI_Send(input[i], 3, MPI_INT, i, tag, MPI_COMM_WORLD);

        }
}
```

Beispiellösung:

```
MPI_Recv(data, 3, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);

int result[3];
MPI_Barrier(MPI_COMM_WORLD);
```

Beispiellösung:

```
MPI_Reduce(data, result, 3, MPI_INT, MPI_MIN, 0, MPI_COMM_WORLD);

MPI_Finalize();
```

Aufgabe 8 (Scala: Binäre Exponentiation) [11 Punkte]

Der Algorithmus zur binären Exponentiation ermöglicht es natürliche Potenzen der Form x^n, $x, n \in \mathbb{N}^+$ effizient zu berechnen. Die Idee basiert darauf Multiplikationen durch Quadrieren einzusparen.
Im Folgenden finden Sie die Funktionsweise des Algorithmus und ein Beispiel dazu:

(a) Umwandlung des Exponenten n in Binärdarstellung b und Umkehrung des Strings (vereinfacht die Verarbeitung und ist beides in Scala gegeben, siehe Hinweise).

(b) Berechne $c_d = x^{2^d}$, bei allen $b_d = 1$, mit $d = 0, 1, 2, \ldots, h-1$, wobei h der Anzahl Bits der Binärdarstellung b und b_d dem d-ten Bit von b entspricht.

(c) Berechne $x^n = \prod_{d=0}^{h-1} c_d$, $\forall b_d = 1$.

Sei $x = 3$, $n = 437$, also $3^{437} \Rightarrow$ h = 9.

(a) $437 \to 110110101 \to 101011011$

(b) $c_0 = 3^{2^0}$, $(c_1 = 0)$, $c_2 = 3^{2^2}$, $(c_3 = 0)$, $c_4 = 3^{2^4}$, $c_5 = 3^{2^5}$, $(c_6 = 0)$, $c_7 = 3^{2^7}$ und $c_8 = 3^{2^8}$

(c) $x^n = 3^{437} = c_0 \cdot c_2 \cdot c_4 \cdot c_5 \cdot c_7 \cdot c_8$

a) Programmieren Sie den Algorithmus, zunächst sequentiell, in Scala. [3 Punkte]

Hinweise: Setzen Sie zur Umwandlung eines Integers in einen Binärstring die Methode `Int.toBinaryString()`, sowie zur Umkehrung des Strings `String.reverse()` ein. Gehen Sie davon aus, dass das Ergebnis nie über den Wertebereich `Long.MaxValue` hinausgehen wird. Sie brauchen also keine Fehlerbehandlung eventueller Überläufe zu implementieren.

```
def fastPow(x: Int, n: Int): Long =
{
    var result = 1L;
```

Beispiellösung:

```
    val b = n.toBinaryString.reverse;
    for (d <- 0 until b.length())
    {
        if (b.charAt(d).equals('1'))
        {
            result *= Math.pow(x,Math.pow(2, d)).toLong;
        }
    }

    return result;
}
```

b) Parallelisieren Sie nun den Algorithmus mit Hilfe von Scala Futures. [8 Punkte]

```
def fastPowParallel(x: Int, n: Int): Long =
{
    var result = 1L;
```

Beispiellösung:

```
    val b = n.toBinaryString.reverse;

    val tasks = for (d <- 0 until b.length()) yield future
    {
        var interim = 0L;
        if (b.charAt(d).equals('1'))
        {
            interim = Math.pow(x,Math.pow(2, d)).toLong;
        }

        interim;
    }

    val futureRes = awaitAll(20000L, tasks: _*);

    futureRes.foreach { res =>
      res match {
      case Some(x: Long) => if (x > 0) result *= x};
    }

    /*alternativ
        val futureRes = awaitAll(20000L, tasks: _*).asInstanceOf[Seq[Long]];

        futureRes.foreach { x => if (x > 0) res *= x};
    */

    return result;
}
```

Aufgabe 9 (Flynn's Taxonomie) [4 Punkte]

In welche Kategorien von Flynn's Taxonomie lassen sich moderne Mehrkern-Prozessoren (CPUs) & -Grafikprozessoren (GPUs) jeweils einordnen? Begründen Sie Ihre Antworten kurz.

Beispiellösung: GPU: SIMD; Ein Befehl im Programmspeicher, der auf verschiedene Daten (Pixel) angewendet wird.

Multicore-CPU: MIMD; Individuelle Befehle in den Programmspeichern möglich. Individueller Programmspeicher für jeden Kern, verschiedene Daten(speicher).

Aufgabe 10 (Parsen einer einfachen Datenaustauschsprache) [20 Punkte]

Zum Austausch hierarchischer Daten mit Schlüssel/Wert Paaren gibt es Sprachen wie *JavaScript Object Notation* (JSON). Die Grammatik einer einfachen Variante einer solchen Sprache lautet:

$$\begin{aligned}
\mathit{Value} &\rightarrow \underline{\texttt{string}} \mid \mathit{Object} \\
\mathit{Object} &\rightarrow \underline{\{}\ \mathit{Members}\ \underline{\}} \\
\mathit{Members} &\rightarrow \mathit{Pair}\ \mathit{Members'} \\
\mathit{Members'} &\rightarrow \underline{,}\ \mathit{Members} \mid \epsilon \\
\mathit{Pair} &\rightarrow \underline{\texttt{string}}\ \underline{:}\ \mathit{Value}
\end{aligned}$$

Zwei Beispiele für gültige Daten sind `{"Name" : "Helmut"}` und
`{"Name" : "Helmut", "Adresse": {"Straße" : "Zirkel", "Ort" : "Karlsruhe"}}`

Zur Implementierung eines Parsers für diese Grammatik stehen zur Verfügung: Prozeduren `nextToken()` und `error()` sowie das aktuelle Token `token` mit Methoden `String getSymbol()` und `getType()`. Die Tokentyp-Konstanten `STRING`, `LCURLY`, `RCURLY`, `COMMA` und `COLON` entsprechen dabei den Terminalen der Grammatik. Gehen Sie weiterhin von Klassen `Pair` und `JSONValue` sowie Unterklassen `JSONObject` und `JSONString` von `JSONValue` mit geeigneten Konstruktoren aus.

Bereits implementiert sind folgende zwei Prozeduren:

```
JSONValue parseValue() {
  switch (token.getType()) {
    case STRING: ... return new JSONString(...);
    case LCURLY: return parseObject();
    default: error();
  }
}

Pair parsePair() {
     ...
}
```

(a) Implementieren Sie `JSONObject parseObject()` sowie weitere noch fehlende Parser-Prozeduren in Pseudo-Code.

Hinweis: Erzeugen Sie *Listen* als Ergebnis beim Parsen von *Members*. Verwenden Sie z.B. den aus `java.util` bekannten Typ `List<Pair>`.

Beispiellösung:

```
JSONObject parseObject() {
  switch (token.getType()) {
    case LCURLY: nextToken();
                 JSONObject o = new JSONObject(parseMembers());
                 if (token.getType() != RCURLY) error();
                 nextToken();
                 return o;
    default: error();
  }
}
```

```
List<Pair> parseMembers() {
  List<Pair> l = new LinkedList<Pair>();
  l.add(parsePair());
  while (true) {
    switch (token.getType()) {
      case COMMA: nextToken();
                  l.add(parsePair());
                  continue;
      case RCURLY: break;
      default: error();
    }
    break;
  }
  return l;
}
```

Alternative für `parseMembers()`, unter Verwendung von Listen gemäß der abstrakten Syntax

$$\begin{array}{rcl} \mathit{List} & = & \mathit{Cons} \mid \mathit{Singleton} \\ \mathit{Cons} & :: & \mathit{Pair}\ \mathit{List} \\ \mathit{Singleton} & :: & \mathit{Pair} \end{array}$$

```
List parseMembers() {
  switch (token.getType()) {
    case STRING: return parseMembers_(parsePair());
    default: error();
  }
}

List parseMembers_(Pair left) {
  switch (token.getType()) {
    case COMMA:  nextToken(); return new Cons(left,parseMembers());
    case RCURLY: return new Singleton(left);
    default: error();
  }
}
```