HTWK Leipzig, Fb IMN, Prof. Dr. Waldmann
Klausur Sprachkonz. d. Par. Prog., 11. 7. 2011

Name	Vorname	Matrikelnummer	Punkte

Wir betrachten einen Prozeß P über dem Alphabet $\Sigma = \{a, b, c\}$ mit

$P = (b \to \mathrm{STOP}) \Box ((a \to P) \|_{\{b\}} (b \to c \to \mathrm{STOP}))$.

Bestimmen Sie Prozesse Q, R mit folgenden Eigenschaften. Geben Sie die dabei benutzten Inferenzregeln an.

- $P \xrightarrow{\tau}{}^{*} \cdot \xrightarrow{a} Q$, /4

- $Q \xrightarrow{\tau}{}^{*} \cdot \xrightarrow{b} R$ /6

Name	Vorname	Matrikelnummer	Punkte

Die folgenden Deklarationen modellieren einfach verkettete Listen mit Zeigern auf erste und letzte Listenzelle:

```
class Cell<A> { AtomicReference<A> item;
                AtomicReference<Cell<A>> next; }
class List<A> { AtomicReference<Cell<A>> first;
                AtommicReference<Cell<A>> last; }
```

Für `List<A> l` soll gelten `l.last.get().next.get() == null`. Benutzen Sie

```
class java.util.concurrent.atomic.AtomicReference<V> {
    boolean compareAndSet(V expect, V update); }
```

für das thread-sichere Einfügen eines Elementes am Anfang der Liste. /7

```
class List<A> { ...
    void add_front (A x) {









    }
}
```

Das Einfügen am Ende der Liste ist nicht so einfach möglich, warum? /1

Stattdessen wird ein Algorithmus mit zwei Schritten benutzt. Spezifizieren Sie den dabei auftretenden Zwischenzustand der Liste. /2

Name	Vorname	Matrikelnummer	Punkte

Das folgende Programm benutzt Transaktionsvariablen `a, b`.

```
atomically $ do
    x <- readTVar a
    writeTVar a ( x + 1 )
    y <- readTVar b
    z <- readTVar a
    writeTVar b ( y + z )
```

Beschreiben Sie die Ausführung des Programms. Benutzen Sie die Begriffe *Log, konsistent, commit*. /5

Definieren Sie den Begriff *Deadlock* eines verteilten Systems. /2

Warum kommt das in transaktionalen Programmen nicht vor? /1

Welches Problem kann trotzdem auftreten? /2

Name	Vorname	Matrikelnummer	Punkte

Die Funktion `lmp :: [ Int ] -> [ Int ]` soll den längsten monoton steigenden Präfix einer Liste bestimmen. Beispiele `lmp [1,2,2,3,0] = [1,2,2,3] ; lmp [3,1,2] = [3]`

Stellen Sie diese Funktion mittels `foldr` dar: /3

```
lmp = foldr ( \ x p -> case p of


            ) []
```

Begründen Sie, daß sich `lmp` nicht direkt als `foldl` schreiben läßt. Geben Sie dazu zwei Listen `l1, l2` sowie eine Zahl `x` an, so daß

```
lmp l1 == lmp l2  &&  not ( lmp (l1 ++ [x]) == lmp (l2 ++ [x]) )

l1 = ....     l2 = ....     lmp l1 == lmp l2 == ...

x = ...    lmp (l1 ++ [x]) == ...      lmp (l2 ++ [x]) == ...
```

/2

Mittels der Funktion `h :: [ Int ] -> ( [ Int ], Bool )` kann man `lmp` indirekt durch `foldb` implementieren. Die erste Komponente von `h xs` ist `lmp xs`, die zweite Komponente ist `True` genau dann, wenn `xs` monoton steigt. Geben Sie ein partielles Inverses für `h` an: /2

```
inv_h :: ( [ Int ], Bool ) -> [ Int ]
inv_h ( p, m ) =
```

Geben Sie für die Darstellung `h = foldb e f g` die Funktion `g` mittels `inv_h` an und vereinfachen Sie, soweit möglich. /2

Welche Laufzeitverbesserung gegenüber dem sequentiellen Fall ist theoretisch möglich, wenn man zur Ausführung des `foldb` 64 Rechenkerne zur Verfügung hat? /1