**Christian-Albrechts-Universität zu Kiel**
Institut für Informatik

Priv.-Doz. Dr. Frank Huch, Marius Rasch

**Institut für Informatik**

# 8. Übung zur Vorlesung „Concurrent and Distributed Programming“

Abgabe am Tuesday, 04. June 2019 - 12:00

---

**Aufgabe 1 - Chat using Linda** 1 Punkt

Implement a chat in Erlang using the tuple space as chat server. Make sure there is no direct communitcation between the clients.

How could you make sure, how to handle orphan messages? A simple solution might be the timeouts from the last exercise sheet.

**Aufgabe 2 - Accounts using Concurrent Haskell and STM** 1 Punkt

In the lecture we implemented some functions for bank accounts using `MVar` and `STM`.We also evaluated their behaviour. In this exercise we extend this implementation.

(1) Define a function

```
    collectedLimitedTransfer :: [Account] -> Account -> Int -> IO Bool
```

that transfers money from a list of accounts to a single account. This transfer shall only happen if there is enought money on the source accounts. Otherwise no money should be transfered. The amount to transfer shall be passed as a parameter. The return value shows, if the transfer was successful.

```
Example:

    Balance a: 50
    Balance b: 100
    Balance c: 35

    > collectedLimitedTransfer [a,b,c] d 120
    True

    Balance a: 0
    Balance b: 30
    Balacne c: 35
```

(2) Does your implementation avoid a deadlock in every case? If not, show an example which results in a deadlock.

(3) Implement `collectedLimitedTransfer` using STM. Use your example from part 2 to check, if the deadlock still exists.

## Aufgabe 3 - Buffer with two elements 1 Punkt

Implement a buffer with two elements using STM. You should be able to write into and to read from this buffer. Write should block, if the buffer is full – read should block, if the buffer is empty.

## Aufgabe 4 - Semaphore using STM 1 Punkt

1. Implement a semaphore using STM. Your implementation should provide the following operations:
   - `newSem` to create a semaphore with a given value,
   - `p` to acquire the semaphore,
   - `v` to release the semaphore and
   - `l` to get the number of processes still permitted to access the semaphore.
2. Create an example to test your implementation.