

# Software Transaction Roll Back Avoidance

## Master Proposal Talk

Three horizontal bars of different shades of green and blue, stacked on top of each other, spanning the width of the slide.

Lasse Folger

November 24, 2016

## Motivation

---

```
type Account = MVar Int
```

```
transfer :: Account -> Account -> Int -> IO ()
```

```
transfer src dst am = do
```

```
  a1 <- takeMVar src
```

```
  a2 <- takeMVar dst
```

```
  putMVar src (a1 - am)
```

```
  putMVar dst (a2 + am)
```

# MVar

---

Thread 1:

```
transfer acc1 acc2 50
```

Thread 2:

```
transfer acc2 acc1 50
```

# MVar

---

Thread 1:

```
transfer acc1 acc2 50
```

Thread 2:

```
transfer acc2 acc1 50
```

⇒ Deadlock

## Use Transactions

---

```
type Account = TVar Int

transfer :: Account -> Account -> Int -> STM ()
transfer src dst am = do
  a1 <- readTVar src
  a2 <- readTVar dst
  writeTVar src (a1 - am)
  writeTVar dst (a2 + am)
```

# TVar

---

Thread 1:

```
atomically $  
  transfer acc1 acc2 50
```

Thread 2:

```
atomically $  
  transfer acc2 acc1 50
```

# TVar

---

Thread 1:

```
atomically $  
  transfer acc1 acc2 50
```

Thread 2:

```
atomically $  
  transfer acc2 acc1 50
```

⇒ works fine, because transactions provide ACID properties

## Current Implementation (Control.Concurrent.STM)

---

- *writeTVar*, *readTVar* and *newTVar* modify TVars
- *retry* and *orElse* alter the control flow
- *atomically* executes a transaction
- composition via bind operator (or do)



# Modify Operations

---

- **newTVar**: creates a new, initialized TVar
- **readTVar**: add the TVar and its version number to the read set
- **writeTVar**: add the TVar and the value to the write set
- read set and write set serve as local cache

atomically  $:: \text{STM } a \rightarrow \text{IO } a$

---

1. compute the read set and write set
2. validate the read set
3. if valid commit the write set
4. else restart

# Validation

---

1. compare version number in read set with actual version number
2. if there is a difference return invalid
3. else return valid

# Problem

---

Thread 1:

```
a1 <- readTVar acc2  
a2 <- readTVar acc1  
writeTVar acc2 (a1 - 50)  
writeTVar acc1 (a2 + 50)
```

Thread 2:

```
a1 <- readTVar acc2  
a2 <- readTVar acc1  
writeTVar acc2 (a1 - 50)  
writeTVar acc1 (a2 + 50)
```

# Problem

---

Thread 1:

```
a1 <- readTVar acc2
a2 <- readTVar acc1
writeTVar acc2 (a1 - 50)
writeTVar acc1 (a2 + 50)
```

Thread 2:

```
a1 <- readTVar acc2
a2 <- readTVar acc1
writeTVar acc2 (a1 - 50)
writeTVar acc1 (a2 + 50)
```

⇒ either sequential or one transaction is rolled back

# Idea

---

Thread 1:

```
a1 <- readTVar acc1
writeTVar acc1 (a1 - 50)
a2 <- readTVar acc2
writeTVar acc2 (a2 + 50)
```

Thread 2:

```
a1 <- readTVar acc2
writeTVar acc2 (a1 - 50)
a2 <- readTVar acc1
writeTVar acc1 (a2 + 50)
```

# Idea

---

Thread 1:

```
a1 <- readTVar acc1
writeTVar acc1 (a1 - 50)
a2 <- readTVar acc2
writeTVar acc2 (a2 + 50)
```

Thread 2:

```
a1 <- readTVar acc2
writeTVar acc2 (a1 - 50)
a2 <- readTVar acc1
writeTVar acc1 (a2 + 50)
```

⇒ delay the evaluation of readTVar to avoid rollback, but...

## Idea does not work

---

```
limitedTransfer acc1 acc2 am = do
  a1 <- readTVar src
  if a1 < am
    then return ()
    else do a2 <- readTVar dst
             writeTVar src (a1 - am)
             writeTVar dst (a2 + am)
```

... the transaction needs the value to determine the branch condition



# Problem

---

- $(>>=) :: \text{STM } a \rightarrow (a \rightarrow \text{STM } b) \rightarrow \text{STM } b$
- Bind extracts the value from the STM context
- STM no longer controls this value.
- Need another Typeclass than Monad

# Applicative

---

- Applicative is less powerfull than Monad
- $(\langle * \rangle) :: \text{STM } (a \rightarrow b) \rightarrow \text{STM } a \rightarrow \text{STM } b$
- The value can be modified without leaving the STM context

# Project

---

- Improved a pure Haskell implementation
- Direct notification
- Explicit, ordered locking
- Optimisations

## Master Thesis until now

---

- Composition by combination of Monad and Applicative
- $\gg=$  evaluates and enables rollbacks
- $\langle * \rangle$  enables to modify values without rollbacks
- `writeTVar :: TVar a -> STM a -> STM ()`

## New Combinators

---

- $(\langle * \rangle) :: \text{STM } (a \multimap b) \multimap \text{STM } a \multimap \text{STM } b$
- $(\langle ** \rangle) :: \text{STM } a \multimap \text{STM } (a \multimap b) \multimap \text{STM } b$
- $(\langle * \rangle) :: \text{STM } a \multimap \text{STM } b \multimap \text{STM } b$
- $(\langle ** \rangle) :: \text{STM } a \multimap (\text{STM } a \multimap \text{STM } b) \multimap \text{STM } b$
- $(\langle > > = \rangle) :: \text{STM } a \multimap (a \multimap \text{STM } b) \multimap \text{STM } b$
- $(\langle > > \rangle) :: \text{STM } a \multimap \text{STM } b \multimap \text{STM } b$

## New Transfer

---

```
transfer :: Account -> Account -> Int -> STM ()
transfer src dst am = do
    readTVar src <*> pure (- am) *> writeTVar src
    readTVar dst <*> pure (+ am) *> writeTVar dst
```

# Todo

---

- Reduce the number of combinators
- ApplicativeDo might do that
- investigate other problems:
  - Branch condition is not changed by TVar modification
  - Recomputation of values which did not change

# Unnecessary Recomputation

---

```
action = do  
  transfer acc1 acc2 50  
  transfer acc3 acc4 100
```

If one transfer is invalidated, both are recomputed