Software Transactional Memory

Steve Severance – Alpha Heavy Industries

Motivation

- Innovations in concurrency has not kept pace
- Still using locks
- Immutability has helped in Haskell

STM Basics

- Transactional Concurrency
- Atomicity – Results are visible all at once
- Isolation – Action unaffected by other threads

- Everything you need is in the stm-2.4.2 package

Haskell and STM

- Exploits Three Haskell Features:
  - Purity (Transactions must be pure)
  - Monads
  - Immutability (Values are never modified)
- Utilizes the Type System
  - Anything involved in the transaction is marked (TVar, TChan, STM, etc…)

Basic Transaction Pattern

- atomically $ do …
- atomically evaluates the transaction

```
main :: IO ()
main = do
   var <- newTVarIO False
  atomically $ do
    val <- readTVar var
    if val == False then writeTVar var True else return ()
```

Primitives

- TVar (Always has a Value)
- TMVar (Analogous to Regular MVar)
- TChan (Multicast Capabilities)
- TQueue (FIFO Queue)
- TBQueue (Bounded Queue)
- TSem (Semaphore)

Retrying

- Use the retry function to retry the transaction
- The transaction will only be run when one of the inputs has changed

```
withdrawFunds :: TVar Account -> Money -> STM Money
withdrawFunds accountVar amount = do
  account <- readTVar accountVar
  if (aBalance account) > amount
   then
    writeTVar accountVar account{aBalance = (aBalance account) `subtract` amount} >> return amount
   else retry
```

Choices

- orElse runs a second option if the first retrys

```
withdrawEvil :: TVar Account -> TChan Account -> Money -> STM Money
withdrawEvil accountVar nsfChan amount = do
  withdrawFunds accountVar amount
    `orElse` withdrawNonSufficientFunds accountVar nsfChan amount
```

Using Alternative

- You always wanted to use those cool operators
- Use <|> instead of orElse

Dealing with Exceptions

- Revisiting the Atomic Guarantee
  - No need for rollback unlike MVar and Chan
- STM can throw and catch

Bounded Queues

- TBQueue limits its size
- Writers will block until space becomes available
- Useful for performance and memory management

Being Strict

- Use NFData (deepseq)
- We internally use special write functions to keep work from leaking across thread boundaries
- Transaction Time

```
writeTVar’ :: NFData a => TVar a -> a -> STM ()
{-# INLINE writeTVar' #-}
writeTVar' var val =
  rnf val `seq` writeTVar var val
```

Other Languages

- Clojure
- Scala
- C++
- C#/.Net

References

- Composable Memory Transactions
- Beautiful Concurrency
- STM Retrospective for .Net
- Gists:
  - Sample 1