Advanced concurrency

@ FProg-Spb

alexander.vershilov@tweag.io

August 31th, 2017

tweag.io

Advanced Concurrency

- "Parallel and Concurrent Programming in Haskell" S.Marlow
- Concurrency, not parallelism
- **Plan**
  - Basics
  - Exceptions
  - STM
  - Debugging (brief)

Concurrency - Threads

- `forkIO :: IO () → IO ThreadId`
  forks new green thread.
- All threads are executed on HEC (Haskell Execution Context) or Capability `N:M`
- HECs are executed on a OS threads
- Haskell threads can be bound to OS thread.
- Communication: MVar and Exceptions

Threads - 2

Possibly unexpected things

- Program exits as soon as `Thread 1` exits;
- Thread stack can't be deallocated if any ThreadId is referenced.

```
main :: IO ()
main = do
  finished <- newEmptyMVar
  forkIOFinally (...)
                (const $ putMVar finished ())
  takeMVar finished
```

- only N threads are running in parallel, where N is a number of HECs

Communication: MVar

MVar

Basic communication primitive value + lock

Shared state

`modifyMVar`

`withMVar`

`readMVar`

Communication box

`putMVar`

`takeMVar`

`readMVar`

Properties:

- can use effects inside;
- awake only one locked thread;
- fairness;.

Communication: MVar

Unexpected problems

- MVar is more lazy than one can expect; (see StrictMVar in *distributed-process*)
- Effects can escape in presence of exceptions;

Exceptions

- Asynchronous exceptions
  - Ability to stop another thread;
  - Cross-thread communication mechanism (bad).
- Working with exeptions
  - Sending exceptions

    ```
    throwTo :: Exception e => ThreadId -> e -> IO ()
    ```
  - Masking exceptions

    ```
    mask :: ((forall a. IO a -> IO a) -> IO b) -> IO b
    ```
  - Handing exceptions

    ```
    catch :: Exception e => IO a -> (e -> IO a) -> IO a
    ```

Masking example

```
modifyMVar :: MVar a -> (a -> IO (a,b)) -> IO b
modifyMVar m io =
  mask $ \restore -> do
    a      <- takeMVar m
    (a',b) <- restore (io a >>= evaluate)
                `onException` putMVar m a
    putMVar m a'
    return b
```

Exceptions are hard

- Unexpected interruptible operations;
- Exceptions in catch are masked but interriptibly
  `` `catch` ( e -> someAction `finally` someAction) ``
- not possible to distringuis between sync and async exceptions;
- SomeAsynchronousException;
- Worker processes;
- *safe-exceptions* package.

STM

- Motivating examples
  - Exchange values in a hash-tables;
  - Composable select.
- Allow transactions;
- Restrict effects that can't be replayed;
- Allow merging of effects.

STM - primitives

- `atomically` – run transaction;
- `TVar` – basic primitive transaction var;
- `retry` – block!
- `orElse` – alternative;

```
select :: Foldable f :: Int -> f a -> IO (Maybe a)
select timeout xs = do
  d <- registerDelay timeout
  atomically $ Just <$> asum xs
    `orElse` (Nothing $> readTVar d >>= check)

checkValue :: TVar a -> (a -> Bool) -> STM  a
checkValue tv f = do
  x <- readTVar tv
  check $ f x
  pure x
```

```
extract :: Int -> TVar (IntMap a) -> STM a
extract i t = do
  x <- readTVar t
  case IntMap.lookup i of
    Nothing -> retry
    Just v  -> modifyTVar t (IntMap.delete i) >> pure

tryExtract :: Int -> TVar (IntMap a) -> STM (Maybe a)
tryExtract i t = (Just <$> extract i t)
                 <|> pure Nothing

extractOrFail :: Int -> TVar (IntMap a) -> STM a
extractOrFail i t = extract i t
                    <|> throwSTM ValueNotFound
```

```
exchange :: Int
         -> TVar (IntMap a)
         -> TVar (IntMap a)
         -> STM ()
exchange i a b = do
  x <- extract i a
  y <- extract i b
  insert i a y
  insert i b x
```

Unexpected problems

- Termination is not guaranteed.
- All blocked threads are unblocked on change.
- No effects in transactions.
- Low performance under high contention.

To be done

Async

```
data Async a = Async ThreadID (STM (Either SomeException a))
```

- Swiss army knife for running asynchronous actions.
- Reuses best features of STM and MVars.
- If you'll try to implement good framework for running asynchronous computations and building process hierarchies then most likely your code will be very close or equal to *async*

debug

- *Event log*
- *Dejafu* package
- *RTS debug options* -Ds

What is missing

- Examples of efficient IPC communication.
- Lock-free operations.
- Speed comparison.

The End

Questions ?