Gtk2HS Beispiel

Sven Eric Panitz

7. Juni 2017

1 Einführung

In diesem Modul sollen Beispielsweise an dem einfachen Gtk GUI Monaden in Aktion gezeigt werden.

```
module Main where
```

Wir importieren die GUI Bibliothek:

```
import Graphics.UI.Gtk
```

Und die Bibliothek für concurrently veränderbare Variablen des Typs MVar.

```
import Control.Concurrent
```

Wir wollen auch einen Timer Thread starten, der wiederholt unseren globalen Zustand verändert.

```
import Control.Concurrent.Timer
import Control.Concurrent.Suspend.Lifted
```

2 Die GUI Komponente

Zunächst sammeln wir alle Komponenten unseres GUIs in einer Datenstruktur. Wir wollen einen Knopf und ein Label in einer vertikalen Box anordnen. Daher hat unser GUI diese drei Komponenten:

```
data GuiControls
  = GuiControls
    {mainBox :: VBox
    ,label   :: Label
    ,button  :: Button
    }
```

Für diese Datenstruktur sei eine Konstruktorfunktion definiert.

```
newGuiControls :: IO GuiControls
newGuiControls = do
  mainPanel ← vBoxNew False 10
  l         ← labelNew $ Just $ show 1
  b         ← buttonNewWithLabel "+1"
  return
    GuiControls
      { mainBox = mainPanel
      , label   = l
      , button  = b
      }
```

3 Das Layout

Ein Layout sei definiert.

```
createOverallLayout :: GuiControls → IO ()
createOverallLayout gui = do
  let mainb = mainBox gui
  let b = button gui
  let l = label gui
  boxPackStart mainb b PackGrow 0
  boxPackStart mainb l PackGrow 0
```

4 Events

Nun kommt das Spannende. Es soll eine Ereignisbehandlung hinzugefügt werden. In der Regel wird eine GUI-Applikation dazu verwendet, einen Zustand zu verändern. Aus einem Zustand kann eine Variable gelesen werden (readMVar) und mit `modifyMVar_` verändert werden.

```
addGuiEvents
  :: (Show a, Num a) ⇒
    GuiControls → MVar a → IO (ConnectId Button)
addGuiEvents gui state = do
  let b = button gui
  let l = label gui
  onClicked b $ addOne l state
```

Hier die eigentliche Funktion, die beim Click ausgeführt wird.

```
addOne l state = do
  s ← readMVar state
  labelSetText l (show (s + 1))
  modifyMVar_ state (return ∘ (λx → x + 1))
  return ()
```

5 Startfenster

Alles zusammen stellen wir in ein Fenster ein.

```
createGuiWindow
  :: (Show a, Num a) ⇒ GuiControls → MVar a → IO Window
createGuiWindow gui state = do
  window ← windowNew
  set window [containerBorderWidth := 10]
  createOverallLayout gui
  set window [containerChild := mainBox gui]
  addGuiEvents gui state
  return window
```

Und schließlich das alles zusammen geführt in einer main.

```
main = do
```

Es wird die State Variable erzeugt mit dem Zustand 1.

```
state ← newMVar 1
```

das GUI wird initalisiert

```
initGUI
gui ← newGuiControls
```

6 Nebenläufigkeit

Zusätzlich erzeugen wir einen Thread der wiederholt zeitgesteuert abläuft

```
thread ← automaticAdd (label gui) state
```

Das Fenster wird erzeugt und das ganze GUI gestartet.

```
window ← createGuiWindow gui state
window `onDestroy` do
  stopTimer thread
```

```
  mainQuit
widgetShowAll window
mainGUI
```

Und hier dier Timer zur wiederholten Ausführung. Achtung: Übersetzen in ghc mit -threaded!!!!!

```
automaticAdd l state = repeatedTimer (addOne l state) $ sDelay 1
```