Funpro Übung 01

Luminous Fennell

October 24, 2013

Übungsblätter

- Bearbeitungszeit des Übungsblatts: Dienstag — Dienstag
- **Alle** Aufgaben müssen bearbeitet werden (Prüfungsvorraussetzung)
  - falls nicht sinnvoll, dann Erklärung
  - rückmelden, falls Aufwand zu hoch (niedrig)
- Abgabe über Subversion Repo
  - **Bitte TF-Account (Pool) mailen an** `fennell@informatik.uni-freiburg.de`
  - Antwort abwarten, WWW-Passwort neu setzen, testen ...
- Korrektur wird von mir ins Repo committet
- Fragen zu den Übungen im Forum
  - http://proglang.informatik.uni-freiburg.de/forum
  - Registrierung mit Uni-Email erforderlich
- Erstes Blatt: Do — Di (dafür kurz)
- Am Schluss ein Projekt

Übungsstunden

- Besprechung des alten Übungsblattes
- Besprechung von Fragen zur Vorlesung
- Unterstützung zur aktuellen Übung

Fragen?

Installation, Stoff, ...

Aktuelles Blatt

Test-Framework

Wir haben QuickCheck Tests von einzelnen Properties gesehen

```
import Test.QuickCheck

prop_EuroUSD x              = x == 0 || euro (usd x) ~== x
prop_power_is_power' x n    = n < 0 || power x n == power' x n
```

Dann, in ghci:

```
*Main> quickCheck prop_EuroUSD
....
*Main> quickCheck prop_power_is_power'
```

Test-Framework

Was tun, wenn man 20/100/500 Properties testen will?

```
import Test.Framework
import Test.Framework.Providers.QuickCheck2
import Test.QuickCheck

prop_EuroUSD x              = x == 0 || euro (usd x) ~== x
prop_power_is_power' x n    = n < 0 || power x n == power' x n

tests =
  [ testGroup "Functions from first lecture"
      [ testProperty "euro is inverse of usd" prop_EuroUSD
      , testProperty "power == power'" prop_power_is_power'
      ]
  , testGroup "Some more tests" [ ... ]
  ]
```

Dann, in ghci:

```
*Main> defaultMain tests
```

Module

Was sind diese `import`'s?

- Jede Datei bildet ein **Modul**

```
module LabExamples01_1 where
-- ^^^ MUSS in der ersten Code Zeile stehen
  ...
```

- Module werden groß geschrieben
- Wenn man `module ...` weglässt, heißt das Modul `Main` (siehe ghci Prompt)
- `import` erlaubt das Verwenden aller (exportierten) Defininitionen aus einem Modul

Module

- Standardmäßig wird immer das Modul `Prelude` importiert
- kann problematisch sein:

```
curry :: String
curry = "yes"

travelDestination :: String
travelDestination = if curry == "yes"
                    then "India"
                    else "France"
```

Abhilfe bei Name-Clashes:

- Verstecken von Definitionen

```
import Prelude hiding (curry)
```

- Selektives Importieren

```
import Prelude (String, (==))
-- ^^^ keine gute Idee für 'Prelude'
```

- Qualifiziertes Importieren

```
import qualified Prelude -- auch nicht optimal

curry :: Prelude.String
curry = "yes"

travelDestination :: Prelude.String
travelDestination = if curry Prelude.== "yes"
                    then "India"
                    else "France"
```

Abhilfe bei Name-Clashes

- „Qualifizier" können umbenannt werden

```
import qualified Prelude as P -- auch nicht optimal

curry :: P.String
curry = "yes"
```

- Die Varianten „Verstecken" und „Qualifizieren" können kombiniert werden

```
import Prelude hiding (curry)
import qualified Prelude as P

curry :: String
curry = "yes"

preludeCurry = P.curry

travelDestination :: String
travelDestination = if curry == "yes"
                    then "India"
                    else "France"
```

Pakete

- Third-Party Module (Bibliotheken) sind in **Paketen** organisiert
- Z.B. `test-framework`
- Sind über ein Repository verfügbar: Hackage
  http://hackage.haskell.org/
  - Manuelles herunterladen des Source-Codes
  - API-Dokumentation (Module)
- Installation am besten mit `cabal` (Teil von „Haskell-Platform")

```
# einmalig updaten
cabal update
# installieren
cabal install test-framework \
              test-framework-quickcheck2
```

- Außerdem nützlich: Hoogle (http://www.haskell.org/hoogle/)
  - Suche nach Funktionen
  - Anzeige des Typs + Links zur API-Dokumentation

Die letzte Aufgabe