Real World Haskell

Projekte 2

Julian Fleischer, Alexander Steen

Donnerstag, 1. August 2013

Heute gilt es ein Projekt auszuwählen und in Gruppen von 2 bis 4 Studenten zu bearbeiten. Für alle Projekte gilt, dass sie am Montag um 10 Uhr c.t. vorzustellen sind, inklusive einer kleiner Demo-Vorführung und einem kurzen Code Review.

Ihre Projekte müssen alle als cabal-Projekt vorliegen und am Montag als Tarball-Archiv (`.tar.gz` / `.tgz`) abgegeben werden. Sie können ein solches Archiv mit Hilfe des Befehls `cabal sdist` herstellen. Orientieren Sie sich an den vorhandenen Beispielen auf der Homepage und den erfolgreichen 1. Projekten.

Sollten Sie Fragen zu `Cabal` oder `GHC` haben, konsultieren Sie bitte das entsprechende Benutzerhandbuch:

The Glorious Glasgow Haskell Compilation System User's Guide
`http://www.haskell.org/ghc/docs/7.6.3/html/users_guide/`

Cabal User Guide
`http://www.haskell.org/cabal/users-guide/`

Am Freitag findet keine Sitzung in der Uni statt. Es steht Ihnen und Ihrem Team frei einen oder mehrere Termine zu wählen und sich an einem von Ihnen gewählten Ort zu treffen. Wir erwarten durchaus, dass sie insgesamt wenigstens 8 Stunden in die Projekte investieren.

Write Yourself A Scheme in 48 Hours

Erstellen Sie unter Anleitung von Wikibooks ihren eigenen Scheme-Interpreter [1].

[1] http://en.wikibooks.org/wiki/Write_Yourself_a_Scheme_in_48_Hours

Brainfuck Interpreter und Compiler

Schreiben sie einen Brainfuck Interpreter und Compiler.

Decaffeinated Java

Schreiben Sie – ganz im Stil von CoffeeScript[2] für JavaScript – einen Transpiler für eine von ihnen erdachte Programmiersprache "Green Tea" (oder wie auch immer Sie ihr Projekt nennen wollen), die nach Java kompiliert.

Sie können für dieses Projekt das Package `language-java`[3] zu Hilfe nehmen!

[2] `http://coffeescript.org/`
[3] http://hackage.haskell.org/package/language-java

Ihre eigene Programmiersprache

Orientieren Sie sich für dieses Projekt an der heute vorgestellten WHILE-Programmiersprache.

GUI Chat

Programmieren sie einen grafischen Chatraum. Sie müssen hierfür *keinen* Server bauen sondern können den in der Vorlesung dargestellten Chat-Server verwenden. Sie müssen lediglich einen Chat-Client bauen, der die gezeichneten Formen an den Server sendet und empfangene Formen zeichnet.

Sie benötigen hierfür die Funktionen des Pakets
`Graphics.Gloss.Interface.IO.Game`
Dieses Paket bietet die Funktion `playIO` an, die es ihnen erlaubt IO-Aktionen zu benutzen um die Welt zu aktualisieren und Ereignisse zu behandeln.

Gehen Sie kleinschrittig vor:

1. Entwickeln Sie zunächst eine kleine Anwendung in der man beispielsweise Kreise durch point and click setzen kann. Verwenden Sie zunächst `play` aus dem Paket
   `Graphics.Gloss.Interface.Pure.Game`.
2. Erweitern Sie nun Ihren Canvas auf dem Sie Kreise zeichnen können indem sie `play` durch `playIO` austauschen und ihre Funktionen dergestalt modifizieren, dass sie einen monadischen return-Wert haben (siehe Signaturen von `play` vs. `playIO`).
3. Sie kennen bereits den ChatServer und den ChatClient Netcat. Sie müssen nun an Stelle von Textnachrichten Koordinaten über den ChatServer versenden. Sie müssen den Server hierfür nicht modifizieren. Es sollte sich jetzt auch erschließen, warum Ihre `playIO` Funktion monadisch ist: Sie wird Nachrichten aus dem Netzwerk lesen und empfangen.

Hinweise:

- Überlegen Sie sich welche Datentypen sie verwenden! Welcher Datentyp symbolisiert ihre Welt (Orientieren Sie sich an dem Bouncing Balls Beispiel)?
- Es ist hilfreich wenn Sie ihre Datentypen automatisch in einen String überführen können und aus einem String auslesen, um sie dann über den ChatServer (der ja Textnachrichten verarbeitet) zu verschicken und zu empfangen. Hierfür empfiehlt es sich die Typklassen `Read` und `Show` zu implenentieren. Das kann der Compiler glücklicherweise automatisch:
  `data World = World { objects :: [Object] } deriving (Eq, Show, Read)`

Schiffe versenken

Programmieren Sie eine Schiffe versenken Anwendung in der man Schiffe versenken über das Netwerk spielen kann. Implementieren Sie ein Spielfeld der Größe 10x10. Lesen Sie für dieses Projekt auch die Hinweise zu dem Projekt `GUI Chat` und `Game of Life`. Das Spiel soll ausschließlich für zwei Spieler konzipiert sein, die über eine Netzwerkverbindung miteinander spielen; ein Mehrspielermodus über einen Rechner ist nicht notwendig.

Gehen Sie kleinschrittig vor:

1. Lesen Sie die Hinweise `GUI Chat` und `Game of Life`.
2. Machen Sie sich klar, wie das Spiel grob abläuft: Welche Spielphasen sind involviert, wie startet das Spiel, wer fängt an, wann endet das Spiel? Sie können sich zum Beispiel eine kleine Grobskizze anfertigen, die Aktivitäten während einer Schiffe-Versenken- Runde visualisiert.
3. Welche Informationen beschreiben den aktuellen Spielzustand, welche Informationen müssen Sie über das Netzwerk vermitteln?[4] Welche Nachrichten müssen Sie dafür unterscheiden, wann muss ein Spieler auf den anderen warten?
4. Sie können zunächst das Schiffe-Versenken-Protokoll fertig stellen und zwischen zwei Klienten ohne eigene Gloss-Oberfläche (z.B. über das Terminal) testen.
5. Überlegen Sie sich dann, welche Informationen Sie zeichen müssen und implementieren Sie eine Gloss-Oberfläche für Ihr Spiel.

[4] Denken Sie an die Verwendung von show- und read-Funktion!

Osmos

Exception: Pacman.simplex: openFile: does not exist (No such file or directory)