Vorlesung vom 03.02.2002
Schlußbemerkungen

Inhalt der Vorlesung

- Organisatorisches
- Noch ein paar Haskell-Döntjes:
  - Web-Scripting mit Haskell
  - Concurrent Haskell
  - HTk
- Rückblick über die Vorlesung
- Ausblick

Der studienbegleitende Leistungsnachweis

- Bitte Scheinvordruck ausfüllen.
  - Siehe Anleitung.
  - Erhältlich vor FB3-Verwaltung (MZH Ebene 7)
  - Nur wer ausgefüllten Scheinvordruck abgibt, erhält auch einen.
- Bei Sylvie Rauer (MZH 8190) oder mir (MZH 8110) abgeben (oder zum Fachgespräch mitbringen)
- Nicht vergessen: in Liste eintragen!.

Das Fachgespräch

- Dient zur Überprüfung der Individualität der Leistung.
  - Insbesondere: Teilnahme an Beabeitung der Übungsblätter.
  - Keine Prüfung.
- Dauer: ca. 5–10 Min; einzeln, auf Wunsch mit Beisitzer
- Inhalt: Übungsblätter
- Bearbeitete Übungsblätter mitbringen — es werden zwei Aufgaben besprochen, die erste könnt Ihr Euch aussuchen.
- Termine:
  - Do. 06.02, Do. 13.02 — Liste vor MZH 8110
  - oder nach Vereinbarung.

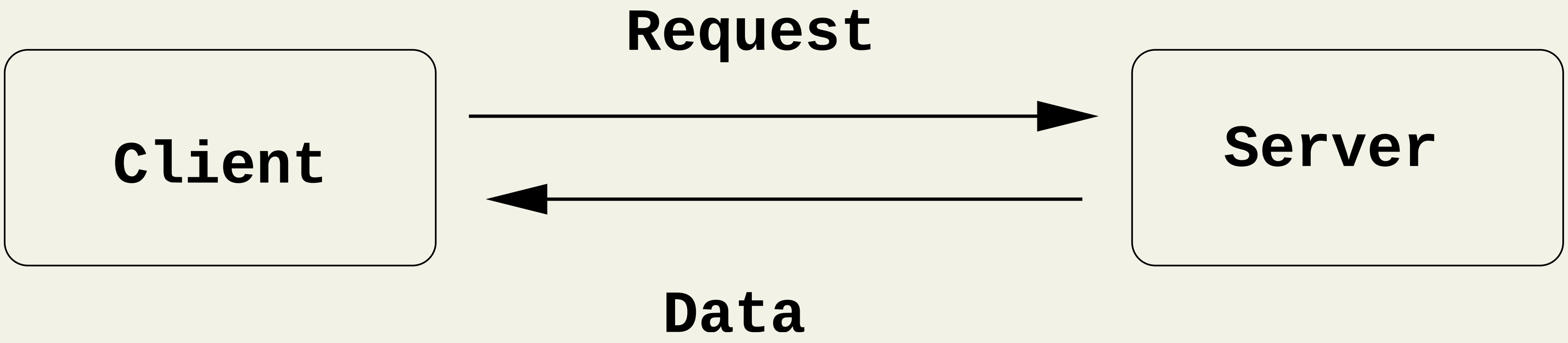
Fallbeispiel: Lehrveranstaltungsevaluation

- Ziel: Webformular zur Evaluation der Lehrveranstaltung

- Erfordert Auswertung auf Serverseite: *server side scripting*

Medieninformatik für Einsteiger

- Das Web:



- `Request` definiert durch Transferprotokoll HTTP
- Datenformat definiert durch MIME
- HTML ist ein spezielles Format

HTTP: Hypertext Transfer Protocol

Client (Webbrowser)	Server (e.g. `apache`)
`connect`	`listen` (Port 80)
`GET index.html`	
	lese Datei `index.html`
	sende Inhalt
Inhalt von `index.html`	
empfangen	
HTML als Grafik darstellen	

Verschiedene Dokumenttypen: MIME

- Dokumente sind `getypt`: MIME
  - (Multipurpose Internet Mail Extensions, `RFC1521`)
- MIME-Typ: `typ/subtyp`, wie in

  ```
  text/plain
  text/html
  image/gif
  application/postscript
  x-application/url-encoded
  ```
- Typ bestimmt Darstellung — HTML ein Typ von vielen

CGI-Skripte *(server side scripting)*

- CGI-Skripte liegen unter
  `/home/www/users/cxl/cgi-bin/<name>`,
  Aufruf als
  `http://www....de/cgi-bin/cgiwrap/cxl/<name>`
- CGI-Skripte werden auf dem Server ausgeführt,
  Ausgabe wird an Client übertragen
- Bei Ausgabe Syntax beachten:

```
Content-Type: text/html

<HTML><BODY><H1>Hello, World</H1></BODY></HTML>
```

Formulare

`Formulare` sind spezielle HTML-Elemente zum Datenaustausch zwischen Client und Server:

Element	Attribut
`FORM`	`ACTION`: `POST`, `GET`
	`METHOD`: URL
`INPUT`	`TYPE`: `TEXT`, `CHECKBOX`, `RADIO`, `SUBMIT`, `RESET` ...
	`NAME`, `VALUE`, `CHECKED`, `SIZE`, `MAXLENGTH`
`TEXTAREA`	Texteditor
	`NAME`, `ROWS`, `COLS`

Was passiert beim Abschicken *(submit)* eines Formulars?

- Client kodiert Formulardaten als URL
- Server erkennt URL als CGI-Skript und ruft CGI-Skript auf
- Kodierte Daten werden in Kommandozeile/Umgebungsvariablen übergeben
- Ergebnis wird an den Client zurückgeschickt

Ein einfaches Beispiel

```
<form method=get action=http://www...de/cgi-lokal/test>
Name: <input type=text name=name size=20 maxlength=30>
Tick: <input name=ticked value=y type=checkbox>
      <input type=submit>
</form>
```

Kodierung der Daten

- Bei `ACTION=mailto:<adress>` wird Dokument geschickt
- `GET` übergibt Argumente in URL,
  `POST` in Umgebungsvariablen
- `GET` für Anfragen, `POST` für Änderungen
- Beispiel vorherige Seite:

  Methode `GET`: `test?name=Hello+there%3f&ticked=y`

  Methode `POST`: Umgebungsvar. `name`: `Hello+there%3f`,
  `ticked`: `y`

Server Side Web Scripting in Haskell

- Ziel: Trennung von Funktionalität und Repräsentation

```
worker :: Request-> IO (CgiOut a)

wrapper :: (Request-> IO (CgiOut a)) -> IO ()
```

`wrapper` dekodiert URL, erzeugt HTML-Text

- Trennung von Funktionalität und Abstraktion
- Flexibilität: leicht auf anderes Format (e.g. ASP) anzupassen

MIME und HTML in Haskell

- MIME-Dokumente repräsentiert durch Typklasse MIME
- HTML ist ein abstrakter Datentyp HTML
  - HTML ist Instanz von MIME
  - Vordefinierte Kombinatoren:

```
prose                   :: String -> HTML
h1,h2,h3,h4,h5,h6,h7    :: String -> HTML
href                    :: URL -> [HTML] -> HTML
```

Server Side Scripting Made Very Easy

- Anfragen vom Typ `Request = [(Name, Value)]`
- Skriptfunktion liefert Wert vom Typ `CgiOut a`:

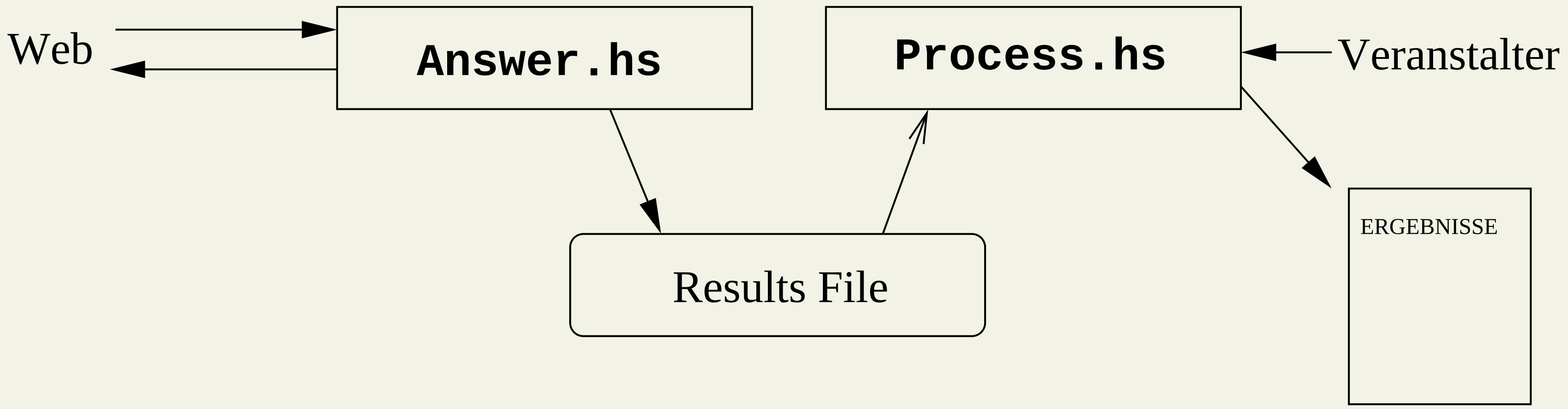
```
module Main (main) where

import CGI

helloWorldHTML :: [(Name,Value)] -> HTML
helloWorldHTML env =
   page "Hello World" [] [h1 "Hello World!"]
main :: IO ()
main = wrapper (\env ->
       do{return (Content{mime= helloWorldHTML env})})
```

Das PI3-Evaluationssystem

- Das Formular
- Gemeinsame Daten: `EvalData.hs`
- CGI-Skript: `Answer.hs`
- Auswertung: `Process.hs` Zeigen



Concurrent Haskell

- Threads in Haskell:

```
forkIO :: IO () -> IO ThreadID
killThread :: ThreadID -> IO ()
```

- Zusätzliche Primitive zur Synchronisation
- Erleichtert Programmierung reaktiver Systeme
  - Benutzerschnittstellen, Netzapplikationen, . . .
- hugs: kooperativ, ghc: präemptiv

- Beispiel: Zeigen

```
module Main where

import Concurrent

write :: Char -> IO ()
write c = putChar c >> write c

main :: IO ()
main = forkIO (write 'X') >> write '.'
```

Grafische Benutzerschnittstellen

- HTk
  - Verkapselung von Tcl/Tk in Haskell
  - Nebenläufig mit Events
  - Entwickelt an der AG BKB (Dissertation E. Karlsen)
  - Mächtig, abstrakte Schnittstelle, mittelprächtige Grafik
- GTk+HS
  - Verkapselung von GTk+ in Haskell
  - Zustandsbasiert mit call-backs
  - Entwickelt an der UNSW (M. Chakravarty)
  - Neueres Tookit, ästethischer, nicht ganz so mächtig

Grafische Benutzerschnittstellen mit HTk

- Statischer Teil: Aufbau des GUI
  - Hauptfenster öffnen

```
main:: IO ()
main =
  do main <- initHTk []
```

  - Knopf erzeugen und in Hauptfenster plazieren

```
     b <- newButton main [text "Press me!"]
     pack b []
```

- Dynamischer Teil: Verhalten während der Laufzeit
  - Knopfdruck als Event

```
click <- clicked b
```

  - Eventhandler aufsetzen

```
spawnEvent
 (forever
```

  - Eventhandler defininieren

```
   (click >>> do nu_label <- mapM randomRIO
                               (replicate 5 ('a','z'))
                 b # text nu_label))
finishHTk
```

- Zeigen. (Etwas längeres Beispiel.)

Grundlagen der funktionalen Programmierung

- Definition von Funktionen durch rekursive Gleichungen
- Auswertung durch Reduktion von Ausdrücken
- Typisierung und Polymorphie
- Funktionen höherer Ordnung
- Algebraische Datentypen
- Beweis durch strukturelle und Fixpunktinduktion

- Fortgeschrittene Features:
  - Modellierung von Zustandsabhängigkeit durch `IO`
  - Überladene Funktionen durch Typklassen
  - Unendliche Datenstrukturen und verzögerte Auswertung
- Beispiele:
  - Parserkombinatoren
  - Grafikprogrammierung
  - Animation

Zusammenfassung Haskell

Stärken:

- Abstraktion durch
  - Polymorphie und Typsystem
  - algebraische Datentypen
  - Funktionen höherer Ordnung
- Flexible Syntax
- Haskell als Meta-Sprache
- Ausgereifter Compiler
- Viele Büchereien

Schwächen:

- Komplexität
- Dokumentation
  - z.B. im Vergleich zu Java's APIs
- Büchereien
- Noch viel im Fluß
  - Tools ändern sich

Warum funktionale Programmierung lernen?

- Abstraktion
  - Denken in Algorithmen, nicht in Programmiersprachen
- FP konzentriert sich auf wesentlichen Elemente moderner Programmierung:
  - Typisierung und Spezifikation
  - Datenabstraktion
  - Modularisierung und Dekomposition
- Blick über den Tellerrand — Blick in die Zukunft
  - Studium $\neq$ Programmierkurs— was kommt in 10 Jahren?

Hat es sich gelohnt?

Hilfe!

- Haskell: primäre Entwicklungssprache an der AG BKB
  - Entwicklungsumgebung für formale Methoden (Uniform Workbench)
  - Werkzeuge für die Spezifikationssprache CASL (Tool Set CATS)
- Wir suchen studentische Hilfskräfte
  - für diese Projekte
- Wir bieten:
  - Angenehmes Arbeitsumfeld
  - Interessante Tätigkeit
- Wir suchen Tutoren für PI3
  - im WS 03/04 — meldet Euch!

Tschüß!