Vorlesung vom 07.02.2005
Schlußbemerkungen

Inhalt der Vorlesung

- Organisatorisches
- Noch ein paar Haskell-Döntjes:
  - Concurrent Haskell
  - HaXml — XML in 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 8050) 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 mindestens zwei Aufgaben besprochen, die erste könnt Ihr Euch aussuchen.
- Termine:
  - Mo. 21.02. — Liste vor MZH 8050
  - oder nach Vereinbarung.

HaXML: XML in Haskell

- Fallbeispiel: Vorteile von Typisierung, algebraische Datentypen.
- Was ist eigentlich XML?
  - Eine Notation für polynomiale Datentypen mit viel < und >
  - Eine Ansammlung darauf aufbauender Techniken und Sprachen

- XML hat mehrere Schichten:
  - Basis: Definition von semantisch strukturierten Dokumenten
  - Präsentation von strukturierten Dokumenten
  - Einheitliche Definitionen und Dialekte

XML — Eine Einführung

- Frei definierbare Elemente und Attribute
  - Vergleiche HTML
- Elemente und Attribute werden in DTD definiert
  - Heutzutage: Schemas, Relax NG
  - Entspricht einer Grammatik
- Beispiel: Ein Buch habe
  - Autor(en) — mindestens einen `[String]` (nichtleer)
  - Titel `String`
  - evtl. eine Zusammenfassung `[String]`
  - ISBN-Nummer `String`

Beispiel: DTD für Bücher

```
<!ELEMENT book (author+, title, abstract?)>
<!ATTLIST book isbn CDATA #REQUIRED>

<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT abstract (paragraph*)>
<!ELEMENT paragraph (#PCDATA)>
```

- `book`, . . . , `paragraph`: Elemente
- `isbn`: Attribut

Ein Beispiel-Buch

```
<?xml version='1.0'?>
<!DOCTYPE book SYSTEM "book.dtd">
<book isbn="3540900357">
  <author>Saunders MacLane</author>
  <title>Categories for the Working Mathematician</title>
  <abstract>
    <paragraph>The introduction to category theory
    by one of its principal inventors.
    </paragraph>
 </abstract>
</book>
```

HaXml

- HaXML: getypte Einbettung in Haskell
  - DTD $\longrightarrow$ algebraischen Datentyp
  - Klasse `XmlContent`
  - Funktionen

```
readXml :: XmlContent a => String -> Maybe a
showXml :: XmlContent a => a -> String
```

  - Jedes Element einen Typ, Instanz von `XmlContent`
- Übersetzung der DTD: `DtdToHaskell book.dtd Book.hs`

```
module Book where

data Book = Book Book_Attrs (List1 Author) Title
                (Maybe Abstract)
            deriving (Eq,Show)
data Book_Attrs = Book_Attrs { bookIsbn :: String}
                  deriving (Eq,Show)
newtype Author = Author String              deriving (Eq,Show)
newtype Title = Title String                deriving (Eq,Show)
newtype Abstract = Abstract [Paragraph] deriving (Eq,Show)
newtype Paragraph = Paragraph String    deriving (Eq,Show)

instance XmlContent Book where
    ...
```

Vorteile

- Einfache, getypte Manipulation von XML-Dokumenten
- Beispiel formatierte Ausgabe (Zeigen):

```
prt :: Book-> String
prt (Book (Book_Attrs {bookIsbn=i}) (NonEmpty as) (Title t)) =
   authors++ ": "++ t++ "\n"++ abstr++ "ISBN: "++ i where
   authors = if length nms > 3 then  head nms ++ " et al"
                else concat (intersperse ", " nms)
   nms    = map (\ (Author a)-> a) as
   abstr = case abs of
                Just (Abstract a)-> unlines (map
                                (\ (Paragraph p)-> p) a) ++ "\n"
                Nothing -> ""
```

- Noch ein Beispiel: Suche
  - . . . in Bücherei nach Stichwörtern in Titel oder Zusammenfassung:
  - Erweiterte DTD `library.dtd`

```
<!ELEMENT library (book*)>
```

  - Damit generierter Typ

```
newtype Library = Library [Book] deriving (Eq,Show)
```

  - Hilfsfunktion: Liste aller Wörter aus Titel und Zusammenfassung

```
getWords :: Book-> [String]
getWords (Book _ _ (Title t) Nothing) = words t
getWords (Book _ _ (Title t) (Just (Abstract a))) =
  words t ++ (concat (map (\ (Paragraph p)-> words p) a))
```

- Anfrage:

```
query :: Library-> String-> [Book]
query (Library bs) key =
  filter (elem key. getWords) bs
```

- Nachteil: Groß/Kleinschreibung relevant
- Deshalb alles in Kleinbuchstaben wandeln:

```
query :: Library-> String-> [Book]
query (Library bs) key =
  filter (elem (map toLower key).
               (map (map toLower).getWords)) bs
```

- Verbesserung: Suche mit regulären Ausdrücken
- Nutzt Regex-Bücherei des GHC:

```
import Text.Regex
data Regex -- abstrakt
mkRegex :: String -> Regex -- übersetzt regex
matchRegex :: Regex -> String -> Maybe [String]
```

- Damit neue Version von query (Zeigen):

```
query' (Library bs) ex =
  filter (any (isJust.matchRegex (mkRegex ex)).
            getWords) bs
```

Zusammenfassung HaXML

- Transformation in Haskell einfacher, typsicherer &c. als style sheets, XSLT, &c
- Durch XML beschriebene Datentypen sind algebraisch
- Nützlich zum Datenaustausch — nahtlose Typsicherheit, e.g.

$$\texttt{Java} \longleftrightarrow \texttt{Haskell} \longleftrightarrow \texttt{C++}$$

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 '.'
```

Der Haskell Web Server

- Ein RFC-2616 konformanter Webserver (Peyton Jones, Marlow 2000)
- Beispiel für ein
  - nebenläufiges,
  - robustes,
  - fehlertolerantes,
  - performantes System.
- Umfang: ca. 1500 LOC, „written with minimal effort“
- Performance: ca. 100 req/s bis 700 req
- `http://www.haskell.org/ simonmar/papers/web-server.ps.gz`

Grafische Benutzerschnittstellen

- HTk
  - Verkapselung von Tcl/Tk in Haskell
  - Nebenläufig mit Events
  - Entwickelt an der Uni Bremen (Dissertation E. Karlsen)
  - Mächtig, abstrakte Schnittstelle, rustikale Grafik
- wxHaskell
  - Verkapselung von wxWidgets
  - wxWidgets: abstrakte GUI-Klassenbibliothek
    - ▷ Implementiert für Linux/GTK, Windows, Mac OS X
  - Neueres Tookit, ästethischer, sehr mächtig
  - Verkapselung: Zustandsbasiert mit call-backs

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.

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
  - Zum Beispiel FFI/HGL

Andere Funktionale Sprachen

- Standard ML (SML):
  - Streng typisiert, strikte Auswertung
  - Formal definierte Semantik
  - Fünf aktiv unterstütze Compiler
  - `http://www.standardml.org/`
- Caml, O'Caml:
  - Streng typisiert, strikte Auswertung
  - Hocheffizienter Compiler, byte code & native
  - Nur ein Compiler (O'Caml)
  - `http://caml.inria.fr/`

Andere Funktionale Sprachen

- LISP & Scheme
  - Ungetypt/schwach getypt
  - Seiteneffekte
  - Viele effiziente Compiler, aber viele Dialekte
  - Auch industriell verwendet

Funktionale Programmierung in der Industrie

- Erlang
  - schwach typisiert, nebenläufig, strikt
  - Fa. Ericsson — Telekom-Anwendungen
- Nischenanwendungen
- Glasgow Haskell Compiler — Microsoft Research
- Warum nicht erfolgreicher?
  - Programmierung nur kleiner Teil
  - Unterstützung Libraries, Dokumentation etc. nicht ausgereift
  - Nicht verbreitet — Funktionale Programmierer zu teuer
  - Konservatives Management

Was lernt uns funktionale Programmierung?

- Abstraktion
  - Denken in Algorithmen, nicht in Programmiersprachen
- Konzentration auf wesentliche 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?

Hilfe!

- Haskell: primäre Entwicklungssprache an der AG BKB
  - Formale Programmentwicklung: `http://www.tzi.de/cofi/hets`
  - Benutzerschnittstellen: `http://proofgeneral.inf.ed.ac.uk/kit`
- Wir suchen studentische Hilfskräfte
  - für diese Projekte
- Wir bieten:
  - Angenehmes Arbeitsumfeld
  - Interessante Tätigkeit
- Wir suchen Tutoren für PI3
  - im WS 05/06 — meldet Euch!

Tschüß!