Fortgeschrittene Funktionale Programmierung

1. Vorlesung

Janis Voigtländer

Universität Bonn

Wintersemester 2015/16

Allgemeines zur Vorlesung

Vorkenntnisse?

- DP-Vorlesung oder äquivalent

Organisation:

- kein Skript, nicht unbedingt immer Folien
- (zumindest mitunter) Übungen mit Rechner

Technik:

- Haskell Platform, GHC
- Emacs (oder Leksah oder . . .)
- lhs2tex, literate programming

Definition „Studienleistungen“

Zu Beginn des Semesters wird eine (funktionale) Programmieraufgabe gestellt, die zeitnah gelöst werden muss.

Im Laufe des Semesters gilt: An der Hälfte der wöchentlichen Übungstermine werden zur Prüfungszulassung herangezogene theoretische oder praktische Aufgaben gestellt. Lösungen zu theoretischen Aufgaben sind im Vorfeld schriftlich abzugeben und auf Aufforderung in der Übungsstunde zu präsentieren/diskutieren. Praktische Aufgaben werden während der Übungsstunde am Rechner bearbeitet. Je Aufgabenblock (Übungsstunde mit zur Prüfungszulassung herangezogenen Aufgaben) gibt es eine binäre Entscheidung: „bestanden“ bei $>50\%$ Aufgabenerfolg. Insgesamt müssen 80% der Aufgabenblöcke in diesem Sinne erfolgreich bestanden werden.

Zum Ende der Vorlesungszeit hin gibt es ein praktisches Projekt (dafür Reduzierung der Präsenzstunden), dessen erfolgreiche Bearbeitung ebenfalls Voraussetzung für die Prüfungszulassung ist.

Zum Inhalt (etwa)

- weitere Exploration von FP-Sprachfeatures
  - laziness vs. strictness (and impact on efficiency)
  - Abstraktionsmechanismen, Typsystem
  - Monaden, Haskell als „imperative“ Sprache
  - beyond Haskell: Agda, Curry, Elm?
- functional algorithm design
  - persistence, memoization
- DSLs
  - Parserkombinatoren, QuickCheck, ...
  - Template Haskell, ...
- semantics, reasoning, verification
  - Lambda-Kalkül, formale Semantikbeschreibung
  - Beweistechniken, typbasiertes Schließen
  - Programmtransformation
- real world FP
  - interfacing, GUIs, Webprogrammierung
  - Haskell ecosystem, Parallelprogrammierung, ...?

Haskell: Laziness + Purity

Zur Erinnerung: eine DP-Klausuraufgabe

Gegeben seien die folgenden Funktionsdefinitionen:

$$\texttt{f} :: [\mathsf{Int}] \to [\mathsf{Int}]$$
$$\texttt{f}\ [\,] \quad\quad = [\,]$$
$$\texttt{f}\ (x : xs) = (\texttt{g}\ x) : (\texttt{f}\ xs)$$

$$\texttt{g} :: \mathsf{Int} \to \mathsf{Int}$$
$$\texttt{g}\ 3 = \texttt{g}\ 4$$
$$\texttt{g}\ n = n + 1$$

sowie die vordefinierten Funktionen `head` und `tail`.
Notieren Sie die einzelnen Haskell-Auswertungsschritte für folgenden Ausdruck (bis zum Endergebnis, und unter genauer Beachtung von Haskells Auswertungsstrategie!):

$$\begin{aligned} \texttt{head}\ (\texttt{tail}\ (\texttt{f}\ [3, 3+1])) &= \underline{\qquad\qquad\qquad\qquad} \\ &= \underline{\qquad\qquad\qquad\qquad} \\ &= \underline{\qquad\qquad\qquad\qquad} \\ &\vdots \end{aligned}$$

Und noch eine:

Gegeben sei die folgende Definition einer Funktion in Haskell:

$$
\begin{aligned}
&\texttt{fun} :: \mathsf{Int} \to \mathsf{Int} \to \mathsf{Int} \\
&\texttt{fun}\ x\ y \mid x < 4 \qquad\quad = x \\
&\qquad\quad\ \mid x == y \qquad = y + 1 \\
&\qquad\quad\ \mid \texttt{otherwise} = \texttt{fun}\ y\ (\texttt{fun}\ (x+1)\ (y-1))
\end{aligned}
$$

Geben Sie für die nachfolgenden Funktionsapplikationen jeweils an, ob die entsprechende Berechnung terminiert oder nicht. Geben Sie im Falle der Terminierung zudem das Resultat der Applikation an:

Applikation	terminiert (mit Resultat)	terminiert nicht
$\texttt{fun}\ 4\ 5$		
$\texttt{fun}\ 4\ 6$		

Laziness, unendliche Listen

$$\texttt{fiblist} = [\texttt{fib}\ n \mid n \leftarrow [0\,..]]$$

$$\texttt{fib}\ 0 = 1$$
$$\texttt{fib}\ 1 = 1$$
$$\texttt{fib}\ n = \texttt{fib}\ (n-2) + \texttt{fib}\ (n-1)$$

$$\Downarrow$$

$$\texttt{fiblist} = 1 : 1 : [\texttt{fiblist}\ !!\ (n-2) + \texttt{fiblist}\ !!\ (n-1) \mid n \leftarrow [2\,..]]$$

$$\Downarrow$$

$$\texttt{fiblist} = 1 : 1 : [\texttt{fiblist}\ !!\ n + \texttt{fiblist}\ !!\ (n+1) \mid n \leftarrow [0\,..]]$$

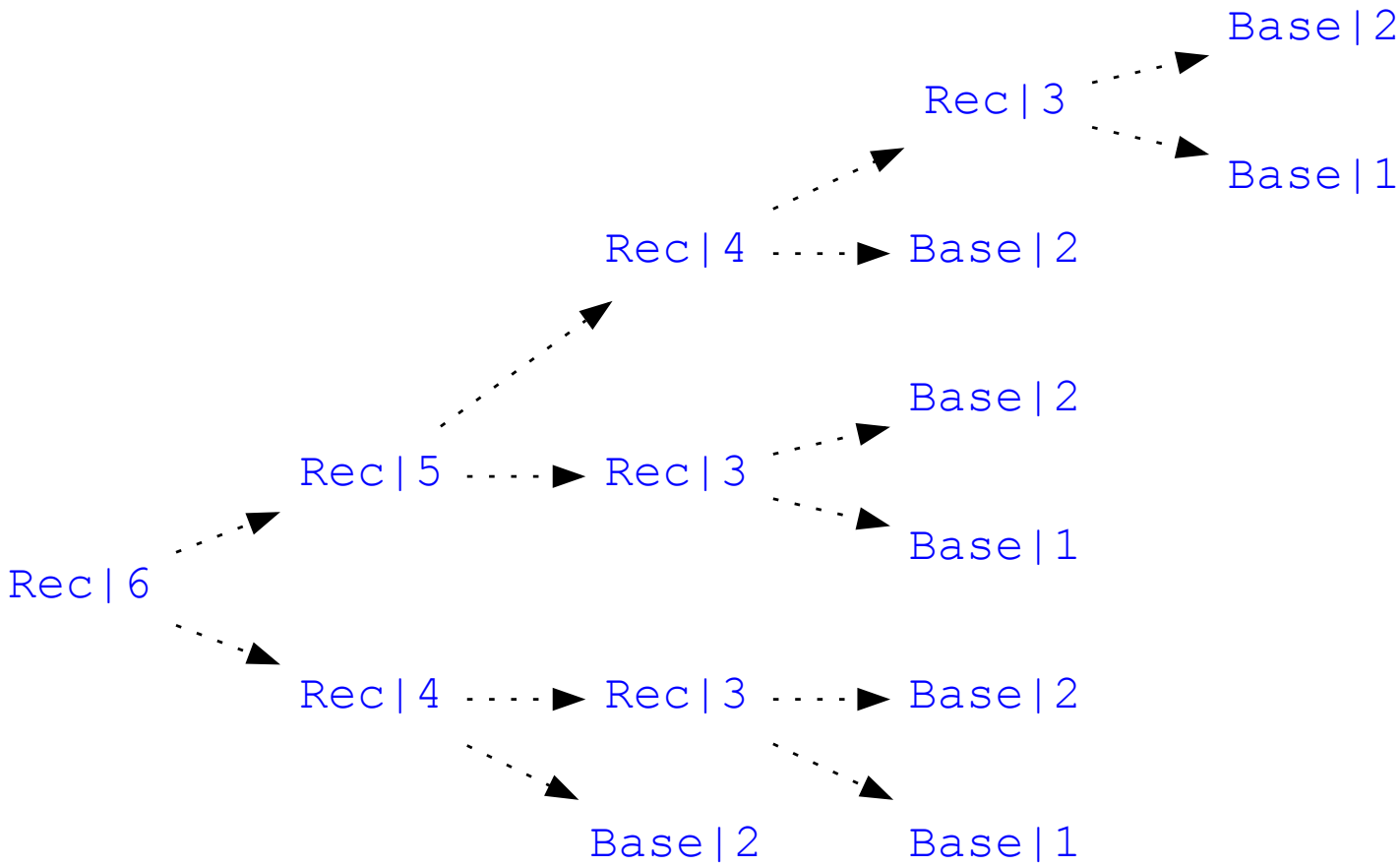
$$\Downarrow$$

$$\texttt{fiblist} = 1 : 1 : [x + y \mid (x, y) \leftarrow \texttt{zip fiblist}\ (\texttt{tail fiblist})]$$

Allgemeine Memoisierung

$$\texttt{fib} :: \mathsf{Int} \to \mathsf{Int}$$
$$\texttt{fib}\ n \mid n < 2 = 1$$
$$\texttt{fib}\ n \qquad\quad = \texttt{fib}\ (n-2) + \texttt{fib}\ (n-1)$$

Rekursiver „Aufrufgraph“, für $\texttt{fib}\ 6$:



Allgemeine Memoisierung

```
fib :: Int → Int
fib n | n < 2 = 1
fib n         = fib (n − 2) + fib (n − 1)
```

```
memo :: (Int → Int) → (Int → Int)
memo f = g
  where g n     = table !! n
        table = [f n | n ← [0 . .]]
```

```
> let mfib = memo fib
> mfib 30
1346269   -- einige Sekunden
> mfib 30
1346269   -- „sofort“
```

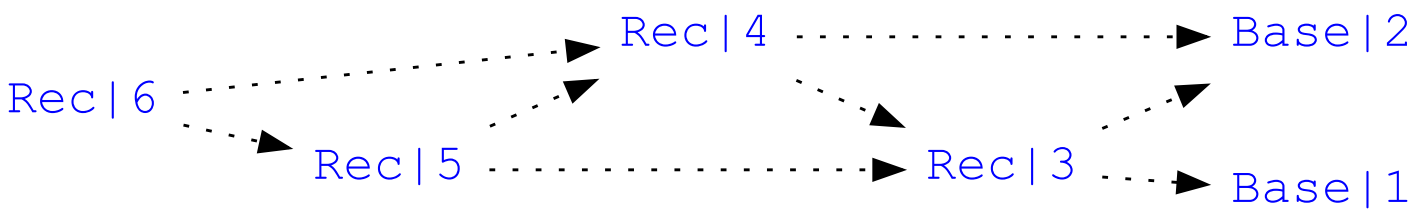
Allgemeine Memoisierung – rekursiv

```
mfib = memo fib

fib :: Int → Int
fib n | n < 2 = 1
fib n         = mfib (n − 2) + mfib (n − 1)
```

```
> fib 30
1346269   -- „sofort“
> fib 30
1346269   -- „sofort“
```

„Aufrufgraph“ jetzt:



Laziness, Kompositionalität und Re-Use

Schön kompositionell:

$$\texttt{any} :: (a \to \text{Bool}) \to [a] \to \text{Bool}$$
$$\texttt{any}\ p = \texttt{or} \circ (\texttt{map}\ p)$$

Mit:

$$\texttt{or} :: [\text{Bool}] \to \text{Bool}$$
$$\texttt{or} = \texttt{foldr}\ (||)\ \text{False}$$

wobei:

$$\texttt{foldr} :: (a \to b \to b) \to b \to [a] \to b$$
$$\texttt{foldr}\ f\ k\ [\,] \qquad = k$$
$$\texttt{foldr}\ f\ k\ (x : xs) = f\ x\ (\texttt{foldr}\ f\ k\ xs)$$

Auswertung:

$$\texttt{any even}\ [1\,..\,10\hat{}6] = \ldots$$