Heutiges Thema

Hintergrund und Grundlagen...

- Ausdrücke, Auswertung von Ausdrücken, Auswertungsstrategien
  Schlagwörter: applicative und normal order evaluation, eager und lazy evaluation, ...

- *Ausblick auf das nächste Mal*: λ-Kalkül
  ...formale Fundierung nahezu aller funktionalen Programmiersprachen

Zur Erinnerung...

fact n = if n == 0 then 1 else (n ∗ fact (n − 1))

fact 2 ⇒ if 2 == 0 then 1 else (2 ∗ fact (2 − 1))
⇒ 2 ∗ fact (2 − 1)
⇒ 2 ∗ fact 1
⇒ 2 ∗ (if 1 == 0 then 1 else (1 ∗ fact (1 − 1)))
⇒ 2 ∗ (1 ∗ fact (1 − 1))
⇒ 2 ∗ (1 ∗ fact 0)
⇒ 2 ∗ (1 ∗ (if 0 == 0 then 1 else (0 ∗ fact (0 − 1))))
⇒ 2 ∗ (1 ∗ 1)
⇒ 2 ∗ 1
⇒ 2

Im Detail...

fact n = if n == 0 then 1 else (n ∗ fact (n − 1))

fact 2 ⇒ if 2 == 0 then 1 else (2 ∗ fact (2 − 1))
⇒ 2 ∗ fact (2 − 1)

Weiter mit a)

⇒ 2 ∗ fact 1
⇒ 2 ∗ (if 1 == 0 then 1 else (1 ∗ fact (1−1)))
⇒ wie bisher...

...oder mit b)

⇒ 2 ∗ (if (2−1) == 0 then 1 else ((2−1) ∗ fact ((2−1)−1)))
⇒ analog fortgeführt...

Schließlich...

⇒ 2

Freiheitsgrade...

Betrachte...

```
fact(fact(square(2+2))) * fact(fact(square(3)))
```

Zentral...

- **Wo** im Ausdruck mit der Auswertung fortfahren?
- **Wie** mit (Funktions-) Argumenten umgehen?

Gretchenfrage...

- **Welcher** Einfluss auf das Ergebnis?

Glücklicherweise...

Theorem
Jede terminierende Auswertungsreihenfolge endet mit demselben Ergebnis.

...Alonzo Church/J. Barclay Rosser (1936)

Auswertungsstrategien...

In der Praxis...

Um den Ausdruck $f(exp)$ auszuwerten...

a) ...berechne zunächst den Wert von exp (und setze diesen Wert dann im Rumpf von f ein)
   $\rightsquigarrow$ applicative order evaluation, eager evaluation, call-by-value evaluation, leftmost-innermost evaluation

b) ...setze exp unmittelbar im Rumpf von f ein und werte den so entstehenden Ausdruck aus
   $\rightsquigarrow$ normal order evaluation, call-by-name evaluation, leftmost-outermost evaluation
   $\rightsquigarrow$ *"Intelligente" Realisierung*: lazy evaluation, call-by-need evaluation

Ein Beispiel...

Einige einfache Funktionen...

```
-- Die Funktion square zur Quadrierung einer ganzen Zahl
square :: Int -> Int
square n = n*n

-- Die Funktion first zur Projektion auf die erste Paarkomponente
first :: (Int,Int) -> Int
first (m,n) = m

-- Die Funktion infiniteInc zum "ewigen" Inkrement
infiniteInc :: Int
infiniteInc = 1 + infiniteInc
```


Auswertung gemäß...

...applicative order (leftmost-innermost):

$$\begin{array}{ll} & \mathtt{square(square(square(}2\mathtt{)))} \\ \Rightarrow & \mathtt{square(square(}2*2\mathtt{))} \\ \Rightarrow & \mathtt{square(square(}4\mathtt{))} \\ \Rightarrow & \mathtt{square(}4*4\mathtt{)} \\ \Rightarrow & \mathtt{square(}16\mathtt{)} \\ \Rightarrow & 16*16 \\ \Rightarrow & 256 \end{array}$$

...6 Schritte.

Auswertung gemäß...

...normal order (leftmost-outermost):

$$\begin{array}{ll}
& \texttt{square}(\texttt{square}(\texttt{square}(2))) \\
\Rightarrow & \texttt{square}(\texttt{square}(2)) * \texttt{square}(\texttt{square}(2)) \\
\Rightarrow & (\texttt{square}(2) * \texttt{square}(2)) * \texttt{square}(\texttt{square}(2)) \\
\Rightarrow & ((2*2) * \texttt{square}(2)) * \texttt{square}(\texttt{square}(2)) \\
\Rightarrow & (4 * \texttt{square}(2)) * \texttt{square}(\texttt{square}(2)) \\
\Rightarrow & (4 * (2*2)) * \texttt{square}(\texttt{square}(2)) \\
\Rightarrow & (4*4) * \texttt{square}(\texttt{square}(2)) \\
\Rightarrow & 16 * \texttt{square}(\texttt{square}(2)) \\
\Rightarrow & \ldots \\
\Rightarrow & 16 * 16 \\
\Rightarrow & 256
\end{array}$$

...1+6+6+1=14 Schritte.

Applicative order effizienter?

Nicht immer...

$$\texttt{first}\ (42, \texttt{square}(\texttt{square}(\texttt{square}(2))))$$

...in applicative order

$$\begin{array}{ll}
& \texttt{first}\ (42, \texttt{square}(\texttt{square}(\texttt{square}(2)))) \\
\Rightarrow & \ldots \\
\Rightarrow & \texttt{first}\ (42, 256) \\
\Rightarrow & 42
\end{array}$$

...1+6+1=8 Schritte.

...in normal order

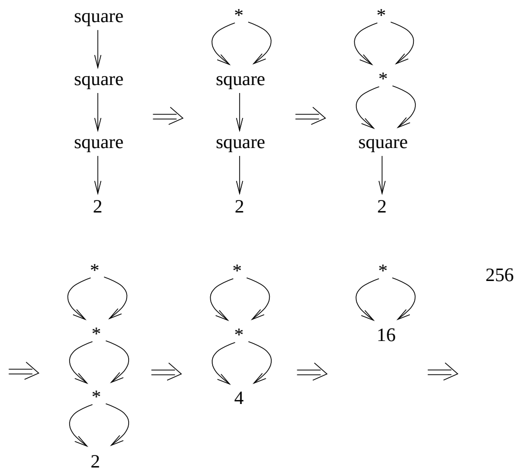
$$\begin{array}{ll}
& \texttt{first}\ (42, \texttt{square}(\texttt{square}(\texttt{square}(2)))) \\
\Rightarrow & 42
\end{array}$$

...1 Schritt.

Von Normal Order zu Lazy Evaluation...

- *Problem*: Mehrfachauswertung von Ausdrücken bei *normal order Evaluation*
- *Ziel*: Vermeidung von Mehrfachauswertungen zur Effizienzsteigerung
- *Lösung*: Graphtransformationen $\rightsquigarrow$ *Lazy Evaluation*!

Lazy evaluation (call-by-need)...



...6 Schritte, aber Graphtransformationen!

Lazy evaluation (call-by-need)...

- ...beruht (implementierungstechnisch) auf Graphtransformationen
- ...garantiert, dass Argumente höchstens einmal (möglicherweise also gar nicht) ausgewertet werden

Insgesamt...
$\leadsto$ ...effiziente Realisierung der normal order Strategie!

Zentrale Ergebnisse...

Theorem

- ...alle terminierenden Auswertungsreihenfolgen enden mit demselben Ergebnis
  $\leadsto$ Konfluenz- oder Diamanteigenschaft
- ...wenn es eine terminierende Auswertungsreihenfolge gibt, so terminiert auch die normal order Auswertungsreihenfolge

...Church/Rosser (1936)

Insbesondere:
Lazy evaluation "vergleichbar effizient" wie applicative (eager) order, falls alle Argumente benötigt werden.

Frei nach Shakespeare...

Eager or lazy evaluation: that is the question.

Quot capita, tot sensa.

Oder: Die Antworten sind verschieden...

- eager evaluation (z.B. ML, Scheme (abgesehen von Makros),...)
- lazy evaluation (z.B. Haskell, Miranda,...)

Lazy vs. Eager: Eine Abwägung... (1)

Lazy Evaluation

- Vorteile
  - terminiert mit Normalform, wenn es eine terminierende Auswertungsreihenfolge gibt
  - wertet Argumente nur aus, wenn nötig
  - elegante Behandlung potentiell unendlicher Datenstrukturen
- Nachteile
  - konzeptuell und implementierungstechnisch anspruchsvoller
    - partielle Auswertung von Ausdrücken (Seiteneffekte! Beachte: Letztere nicht in Haskell! In Scheme: Verantwortung beim Programmierer.)
    - Graphtransformationen
    - Ein-/Ausgabe
  - Volles Verständnis: Domain-Theorie und λ-Kalkül im Detail

Lazy vs. Eager: Eine Abwägung... (2)

Eager Evaluation

- Vorteile
  - Konzeptuell und implementierungstechnisch einfacher
  - Vom mathematischen Standpunkt oft "natürlicher" (Beispiel: `first (42,infiniteInc)`)
  - (Einfache(re) Integration imperativer Konzepte)

Mithin... eager oder lazy – eine Frage des Anwendungsprofils!

Zu guter Letzt...

Wäre ein Haskell-Compiler (Interpretierer) korrekt, der die Fakultätsfunktion applikativ auswertet?

Ja, weil die Funktion `fact` *strikt* in ihrem Argument ist...

$\leadsto$ eager evaluation oder auch strict evaluation!

Zur Erinnerung

Vorschau auf kommende Vorlesungstermine...

- Di, 26.10.2004 und Di, 02.11.2004 keine Vorlesung
- Ab Di, 09.11.2004, jeden Dienstag von 13-14 Uhr im Treitl-Hörsaal

Vorschau auf kommende Aufgabenblätter...
Ausgabe des...

- zweiten Aufgabenblatts: **Morgen**, Mi, den 20.10.2004
- dritten Aufgabenblatts: Mi, den 27.10.2004
- vierten Aufgabenblatts: Mi, den 03.11.2004