$$
\begin{aligned}
\mathsf{compi} &= \mathbf{fn}\ f \Rightarrow \mathbf{fn}\ g \Rightarrow \mathbf{fn}\ i \Rightarrow \mathbf{fn}\ x \Rightarrow \mathbf{let}\ y = g\ i\ x \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{in}\ \ f\ i\ y \\
\mathsf{compi}_1 &= \mathbf{fn}\ f \Rightarrow \mathbf{fn}\ g \Rightarrow \mathbf{fn}\ i \Rightarrow \mathbf{fn}\ x_1 \Rightarrow \mathbf{fn}\ x_2 \Rightarrow \\
&\qquad\qquad \mathbf{let}\ \ y\ =\ g\ i\ x_1 \\
&\qquad\qquad \mathbf{in}\ \ f\ i\ y\ x_2 \\
\mathsf{compi}_2 &= \mathbf{fn}\ f \Rightarrow \mathbf{fn}\ g \Rightarrow \mathbf{fn}\ i \Rightarrow \mathbf{fn}\ x_1 \Rightarrow \mathbf{fn}\ x_2 \Rightarrow \\
&\qquad\qquad \mathbf{let}\ \ y\ =\ g\ i\ x_2 \\
&\qquad\qquad \mathbf{in}\ \ f\ i\ x_1\ y \\
\mathsf{cmp}_1 &= \mathbf{fn}\ f \Rightarrow \mathbf{fn}\ g \Rightarrow \mathbf{fn}\ i \Rightarrow \mathbf{fn}\ x_1 \Rightarrow \mathbf{fn}\ x_2 \Rightarrow \\
&\qquad\qquad \mathbf{let}\ \ y\ =\ g\ x_2 \\
&\qquad\qquad \mathbf{in}\ \ f\ i\ x_1\ y \\
\mathsf{cmp}_2 &= \mathbf{fn}\ f \Rightarrow \mathbf{fn}\ g \Rightarrow \mathbf{fn}\ i \Rightarrow \mathbf{fn}\ x_1 \Rightarrow \mathbf{fn}\ x_2 \Rightarrow \\
&\qquad\qquad \mathbf{let}\ \ y\ =\ g\ i\ x_2 \\
&\qquad\qquad \mathbf{in}\ \ f\ x_1\ y
\end{aligned}
$$

Dann gilt:

$$
\begin{array}{lcl}
\mathsf{comp}\,(\mathsf{mapi}\ f)\ (\mathsf{map}\ g) & = & \mathsf{mapi}\,(\mathsf{comp}_2\ f\ g) \\
\mathsf{comp}\,(\mathsf{map}\ f)\ (\mathsf{mapi}\ g) & = & \mathsf{mapi}\,(\mathsf{comp}\ f\ g) \\
\mathsf{comp}\,(\mathsf{mapi}\ f)\ (\mathsf{mapi}\ g) & = & \mathsf{mapi}\,(\mathsf{compi}\ f\ g) \\
\mathsf{comp}\,(\mathsf{foldli}\ f\ a)\ (\mathsf{map}\ g) & = & \mathsf{foldli}\,(\mathsf{cmp}_1\ f\ g)\ a \\
\mathsf{comp}\,(\mathsf{foldl}\ f\ a)\ (\mathsf{mapi}\ g) & = & \mathsf{foldli}\,(\mathsf{cmp}_2\ f\ g)\ a \\
\mathsf{comp}\,(\mathsf{foldli}\ f\ a)\ (\mathsf{mapi}\ g) & = & \mathsf{foldli}\,(\mathsf{compi}_2\ f\ g)\ a \\
\mathsf{comp}\,(\mathsf{foldli}\ f\ a)\ (\mathsf{tabulate}\ g) & = & \textbf{let}\ h = \ \textbf{fn}\ a\ \Rightarrow\ \ \textbf{fn}\ i\ \Rightarrow \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{let}\ x = g\ i \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{in}\ \ f\ i\ a\ x \\
 & & \qquad\qquad\quad \textbf{in}\ \ \mathsf{loop}\ h\ a
\end{array}
$$

Diskussion:

- Achtung: index-abhängige Transformationen hängen kommutieren nicht mit rev oder filter.
- Alle unsere Regeln lassen sich nur anwenden, wenn die Funktionen id, map, mapi, foldl, foldli, filter, rev, tabulate, rev_tabulate, loop, rev_loop, ... von einer Standard-Bibliothek bereit gestellt werden: Nur dann können die algebraischen Eigenschaften garantiert werden !!!
- Ähnliche Vereinfachungsregeln lassen sich für jede Art von baumartiger Datenstruktur tree α herleiten.
- Diese stellen gegebenenfalls auch Operationen map, mapi und foldl, foldli mit den entsprechenden Regeln zur Verfügung.
- Weitere Möglichkeiten eröffnen Funktionen to_list und from_list ...

Beispiel

$$
\begin{array}{lcl}
\textbf{type}\ \mathsf{tree}\ \alpha & = & \mathsf{Leaf} \mid \mathsf{Node}\ \alpha\ (\mathsf{tree}\ \alpha)\ (\mathsf{tree}\ \alpha) \\
\mathsf{map} & = & \textbf{fn}\ f \Rightarrow \mathsf{fix}\,(m, \textbf{fn}\ t \Rightarrow \textbf{case}\ t\ \textbf{of}\ \mathsf{Leaf} : \mathsf{Leaf} \\
 & & \quad \mid \mathsf{Node}\ x\ l\ r : \textbf{let}\ l' = m\ l \\
 & & \qquad\qquad\qquad\qquad\ x' = f\ x \\
 & & \qquad\qquad\qquad\qquad\ r' = m\ r \\
 & & \qquad\qquad\qquad\quad \textbf{in}\ \mathsf{Node}\ x'\ l'\ r') \\
\mathsf{foldl} & = & \textbf{fn}\ f \Rightarrow \mathsf{fix}\,(h, \textbf{fn}\ a \Rightarrow \textbf{fn}\ t \Rightarrow \textbf{case}\ t\ \textbf{of}\ \mathsf{Leaf} : a \\
 & & \quad \mid \mathsf{Node}\ x\ l\ r : \textbf{let}\ a_1 = h\ f\ a\ l \\
 & & \qquad\qquad\qquad\ a_2 = f\ a_1\ x \\
 & & \qquad\qquad\quad \textbf{in}\ h\ a_2\ r)
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{to\_list} & = & \textbf{let}\ l = \mathsf{fix}\,(h,\ \textbf{fn}\,t\ \Rightarrow\ \textbf{fn}\,a\ \Rightarrow\ \textbf{case}\ t\ \textbf{of}\ \mathsf{Leaf}\ :\ a \\
 & & \qquad\qquad \mid\ \mathsf{Node}\ x\ t_1\ t_2\ :\ \ \textbf{let}\ \ a_1\ =\ h\ t_2\ a \\
 & & \qquad\qquad\qquad\qquad\qquad\quad z\ =\ (x, a_1) \\
 & & \qquad\qquad\qquad\qquad\qquad\quad a_2\ =\ ::z \\
 & & \qquad\qquad\qquad\qquad\quad \textbf{in}\ \ h\ t_1\ a_2 \\
 & & \textbf{in}\ \ \textbf{fn}\ t\ \rightarrow\ l\ t\ [\,] \\
\mathsf{from\_list} & = & \mathsf{fix}\,(h,\ \textbf{fn}\,l\ \Rightarrow \\
 & & \qquad \textbf{case}\ l\ \textbf{of}\ [\,]\ :\ \ \mathsf{Leaf} \\
 & & \qquad \mid\ \ ::z\ :\textbf{case}\ z\ \textbf{of}\ (x, xs)\ : \\
 & & \qquad\quad \textbf{let}\ \ r\ \ =\ \ h\ xs \\
 & & \qquad\quad \textbf{in}\ \ \mathsf{Node}\ x\ \mathsf{Leaf}\ l)
\end{array}
$$

Achtung:

Nicht jede natürlich erscheinende Gleichung ist gültig:

$$
\begin{array}{lcl}
\mathsf{comp\ to\_list\ from\_list} & = & \mathsf{id} \\
\mathsf{comp\ from\_list\ to\_list} & \neq & \mathsf{id} \\
\mathsf{comp\ to\_list\ (map}\ f\mathsf{)} & = & \mathsf{comp\ (map}\ f\mathsf{)\ to\_list} \\
\mathsf{comp\ from\_list\ (map}\ f\mathsf{)} & = & \mathsf{comp\ (map}\ f\mathsf{)\ from\_list} \\
\mathsf{comp\ (foldl}\ f\ a\mathsf{)\ to\_list} & = & \mathsf{foldl}\ f\ a \\
\mathsf{comp\ (foldl}\ f\ a\mathsf{)\ from\_list} & = & \mathsf{foldl}\ f\ a
\end{array}
$$

In diesem Fall gibt es sogar ein rev:

$$
\begin{array}{lcl}
\mathsf{rev} & = & \mathsf{fix}\,(h,\ \mathbf{fn}\,t\ \Rightarrow \\
 & & \quad \mathbf{case}\ t\ \mathbf{of}\ \mathsf{Leaf}\ :\ \mathsf{Leaf} \\
 & & \quad \mid\ \mathsf{Node}\ x\ t_1\ t_2\ :\ \ \mathbf{let}\ s_1 = h\ t_1 \\
 & & \qquad\qquad\qquad\qquad\ \ s_2 = h\ t_2 \\
 & & \qquad\qquad\qquad\qquad\ \ \mathbf{in}\ \ \mathsf{Node}\ x\ t_1\ t_2)
\end{array}
$$

$$\mathsf{comp\ to\_list\ rev} = \mathsf{comp\ rev\ to\_list}$$

$$\mathsf{comp\ from\_list\ rev} \neq \mathsf{comp\ rev\ from\_list}$$

4.6 CBN vs. CBV: Striktheitsanalyse

Problem:

- Programmiersprachen wie Haskell berechnen die Werte **let**-definierter Variablen und aktueller Parameter erst, wenn auf diese zugegriffen wird.
- Das ermöglicht den eleganten Umgang mit (potentiell) unendlichen Listen, von denen nur ein endlicher Abschnitt zur Berechnung des Ergebnisses benötigt wird :-)
- Die standard-mäßige Verzögerung von Berechnungen führt jedoch zu einem nicht akzeptablen Overhead ...

Beispiel

$$
\begin{array}{lcl}
\mathsf{from} & = & \mathbf{fn}\ n \Rightarrow \mathbf{let}\ n' = n+1 \\
& & \qquad\qquad\quad\ ns = \mathsf{from}\ n' \\
& & \qquad\qquad\quad\ z = (n, ns) \\
& & \qquad\quad \mathbf{in}\ :: z \\
\mathsf{take} & = & \mathbf{fn}\ k \Rightarrow \mathbf{fn}\ s \Rightarrow \mathbf{case}\ s\ \mathbf{of}\ [\,] : [\,] \\
& & \qquad\qquad\qquad\qquad\quad \mid\ :: z : \mathbf{case}\ z\ \mathbf{of}\ (x, xs) : \\
& & \qquad\qquad\qquad\qquad\qquad \mathbf{if}\ k \leq 0\ \mathbf{then}\ [\,] \\
& & \qquad\qquad\qquad\qquad\qquad \mathbf{else}\ \mathbf{let}\ k' = k-1 \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\quad\ ys = \mathsf{take}\ k'\ xs \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\quad\ z' = (x, ys) \\
& & \qquad\qquad\qquad\qquad\qquad\quad \mathbf{in}\ :: z'
\end{array}
$$

Dann liefert CBN:

$$\mathsf{take}\ 5\ (\mathsf{from}\ 0) = [0, 1, 2, 3, 4]$$

— während die Auswertung bei CBV nicht terminiert !!!

Dann liefert CBN:

$$\mathsf{take}\ 5\ (\mathsf{from}\ 0) = [0,1,2,3,4]$$

— während die Auswertung bei CBV nicht terminiert !!!

Andererseits benötigen bei CBN endrekursive Funktionen plötzlich nicht-konstanten Platz ???

$$\begin{array}{lcl} \mathsf{fac2} & = & \mathbf{fn}\ x \Rightarrow \mathbf{fn}\ a \Rightarrow \mathbf{if}\ x \leq 0\ \mathbf{then}\ a \\ & & \mathbf{else}\ \mathbf{let}\ a' = a \cdot x \\ & & \qquad\qquad\ x' = x - 1 \\ & & \mathbf{in}\ \mathsf{fac2}\ x'\ a' \end{array}$$

Diskussion:

- Die Multiplikationen werden durch geschachtelte Abschlüsse im akumulierenden Parameter gesammelt.
- Erst wenn auf den Wert eines Aufrufs von fac2 x 0 zugegriffen wird, wird diese dynamische Datenstruktur ausgewertet.
- Stattdessen hätten wir den akkumulierenden Parameter auch direkt stets by-value übergeben können !!!
- Das ist das Ziel der nächsten Optimierung ...

Vereinfachung:

- Wir verzichten zuerst einmal auf Datenstrukturen, höhere Funktionen und lokale Funktionsdefinitionen.
- Wir führen einen unären Operator # ein, der die Auswertung einer Variable erzwingt.
- Ziel der Transformation ist es, an möglichst vielen Stellen # einzufügen ...

Vereinfachung:

- Wir verzichten zuerst einmal auf Datenstrukturen, höhere Funktionen und lokale Funktionsdefinitionen.
- Wir führen einen unären Operator # ein, der die Auswertung einer Variable erzwingt.
- Ziel der Transformation ist es, an möglichst vielen Stellen # einzufügen ...

$$
\begin{array}{lll}
e & ::= & c \mid x \mid e_1 \,\square_2\, e_2 \mid \square_1\, e \mid f\ e_1\ \ldots\ e_k \mid \textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2 \\
 & & \mid \textbf{let}\ r_1 = e_1\ \textbf{in}\ e \\
r & ::= & x \mid \#x \\
d & ::= & f\ x_1\ \ldots\ x_k = e \\
p & ::= & \textbf{letrec}\ d_1\ \ldots\ d_n\ \textbf{in}\ e
\end{array}
$$

Idee:

- Beschreibe eine k-stellige Funktion

$$f : \mathbf{int} \to \ldots \to \mathbf{int}$$

durch eine Funktion

$$[\![f]\!]^\sharp : \mathbb{B} \to \ldots \to \mathbb{B}$$

- 0 bedeutet: Auswertung terminiert sicher nicht.
- 1 bedeutet: Auswertung terminiert möglicherweise.
- $[\![f]\!]^\sharp\ 0 = 0$ bedeutet: falls der Funktionsaufruf einen Wert liefert, dann muss auch die Auswertung des Arguments einen Wert geliefert haben

$\Longrightarrow$ f ist strikt.

Idee (Forts.):

- Wir ermitteln die abstrakte Semantik aller Funktionen :-)
- Dazu stellen wir ein Gleichungssystem auf ...

Hilfsfunktion:

$$
\begin{array}{lcl}
[\![e]\!]^\sharp & : & (Vars \to \mathbb{B}) \to \mathbb{B} \\
[\![c]\!]^\sharp\ \rho & = & 1 \\
[\![x]\!]^\sharp\ \rho & = & \rho\, x \\
[\![\Box_1\ e]\!]^\sharp\ \rho & = & [\![e]\!]^\sharp\ \rho \\
[\![e_1\ \Box_2\ e_2]\!]^\sharp\ \rho & = & [\![e_1]\!]^\sharp\ \rho \wedge [\![e_2]\!]^\sharp\ \rho \\
[\![\textbf{if}\ e_0\ \textbf{then}\ e_1\ \textbf{else}\ e_2]\!]^\sharp\ \rho & = & [\![e_0]\!]^\sharp\ \rho \wedge ([\![e_1]\!]^\sharp\ \rho \vee [\![e_2]\!]^\sharp\ \rho) \\
[\![f\ e_1\ \ldots\ e_k]\!]^\sharp\ \rho & = & [\![f]\!]^\sharp\ ([\![e_1]\!]^\sharp\ \rho)\ \ldots\ ([\![e_k]\!]^\sharp\ \rho) \\
\ldots
\end{array}
$$

$$
\begin{array}{lcl}
[\![\textbf{let } x_1 = e_1 \textbf{ in } e]\!]^\sharp\ \rho & = & [\![e]\!]^\sharp\ (\rho \oplus \{x_1 \mapsto [\![e_1]\!]^\sharp\ \rho\}) \\
[\![\textbf{let } \#x_1 = e_1 \textbf{ in } e]\!]^\sharp\ \rho & = & ([\![e_1]\!]^\sharp\ \rho) \wedge ([\![e]\!]^\sharp\ (\rho \oplus \{x_1 \mapsto 1\}))
\end{array}
$$

Gleichungssystem:

$$
[\![f_i]\!]^\sharp b_1\ \ldots\ b_k = [\![e_i]\!]^\sharp\ \{x_j \mapsto b_j \mid j = 1, \ldots, k\}, \qquad i = 1, \ldots, n, b_1, \ldots, b_k \in \mathbb{B}
$$

- Die Unbekannten des Gleichungssystems sind die Funktionen $[\![f_i]\!]^\sharp$ bzw. die einzelnen Einträge $[\![f_i]\!]^\sharp b_1\ \ldots\ b_k$ in ihre Wertetabelle.
- Sämtliche rechte Seiten sind monoton!
- Folglich existiert eine kleinste Lösung :-)
- Der vollständige Verband $\mathbb{B} \to \ldots \to \mathbb{B}$ hat Höhe $\mathcal{O}(2^k)$:-(

Beispiel:

Für fac2 erhalten wir:

$$\begin{aligned}[\![\mathsf{fac2}]\!]^\sharp\ b_1\ b_2 \quad &= \quad b_1 \wedge (b_2 \vee \\ &\qquad [\![\mathsf{fac2}]\!]^\sharp\ b_1\ (b_1 \wedge b_2))\end{aligned}$$

Die Fixpunkt-Iteration liefert:

0	$\mathbf{fn}\ x \Rightarrow \mathbf{fn}\ a \Rightarrow 0$
1	$\mathbf{fn}\ x \Rightarrow \mathbf{fn}\ a \Rightarrow x \wedge a$
2	$\mathbf{fn}\ x \Rightarrow \mathbf{fn}\ a \Rightarrow x \wedge a$

Wir schließen:

- Die Funktion fac2 ist in beiden Argumenten strikt, d.h. falls ihre Auswertung terminiert, dann auch die Berechnung ihrer Argumente.
- Entsprechend transformieren wir:

$$
\begin{array}{lcl}
\mathsf{fac2} & = & \mathbf{fn}\ x \Rightarrow \mathbf{fn}\ a \Rightarrow \quad \mathbf{if}\ x \leq 0\ \mathbf{then}\ a \\
& & \quad\ \mathbf{else}\ \mathbf{let}\ \#\,x' = x - 1 \\
& & \qquad\qquad\ \#\,a' = x \cdot a \\
& & \quad\ \mathbf{in}\ \mathsf{fac2}\ x'\ a'
\end{array}
$$

Korrektheit der Analyse:

- Das Gleichungssystem ist eine abstrakte denotationelle Semantik.
- Diese charakterisiert die Bedeutung von Funktionen als kleinste Lösung der entsprechenden Gleichung für die zugehörige Transformation.
- Für Werte benutzt sie die vollständige Halbordnung $\mathbb{Z}_\perp$.
- Für vollständige Halbordnungen gilt der Kleene'sche Fixpunktsatz :-)
- Als Beschreibungsrelation Δ verwenden wir:

$$\perp \Delta 0 \quad \text{und} \quad z \Delta 1 \quad \text{für } z \in \mathbb{Z}$$

Erweiterung: Datenstrukturen

- Funktionen können unterschiedlich große Teile einer Datenstruktur benötigen ...

$$\begin{array}{lcll} \mathsf{hd} & = & \textbf{fn}\ l & \Rightarrow \quad \textbf{case}\ l\ \textbf{of} :: z\ : \\ & & & \qquad\quad \textbf{case}\ z\ \textbf{of}\ (x, xs)\ : x \end{array}$$

- hd greift nur auf das erste Element einer Liste zu.
- length greift nur auf das Rückgrad der Argumentliste zu.
- rev verlangt die gesamte Auswertung des Arguments — sofern das Ergebnis ganz benötigt wird ...