4 Optimierung funktionaler Programme

Beispiel:

$$\begin{array}{lll} \textbf{fun}\ \ \mathsf{fac}\ x & = & \textbf{if}\ \ x \leq 1\ \ \textbf{then}\ \ 1 \\ & & \textbf{else}\ \ x \cdot \mathsf{fac}\ (x-1) \end{array}$$

- Es gibt keine Basis-Blöcke :-(
- Es gibt keine Schleifen :-(
- Viele Funktionen sind rekursiv :-((

Strategien zur Optimierung:

$\Longrightarrow$ Verbessere spezielle Ineffizienzen wie:

- Pattern Matching
- Lazy Evaluation (falls vorhanden ;-)
- Indirektionen — Unboxing / Escape-Analyse
- Zwischendatenstrukturen — Deforestation

$\Longrightarrow$ Entdecke bzw. erzeuge Schleifen mit Basis-Blöcken :-)

- Endrekursion
- Inlining
- **let**-Floating

Wende dann allgemeine Optimierungs-Techniken an!

... etwa durch Übersetzung nach C ;-)

Achtung:

Wir benötigen neue Programmanalyse-Techniken, um Informationen über funktionale Programme zu sammeln.

Beispiel: Inlining

$$
\begin{array}{lll}
\textbf{fun}\ \ \mathsf{max}\ (x, y) & = & \textbf{if}\ \ x > y\ \ \textbf{then}\ \ x \\
 & & \textbf{else}\ \ y \\
\textbf{fun}\ \ \mathsf{abs}\ z & = & \mathsf{max}\ (z, -z)
\end{array}
$$

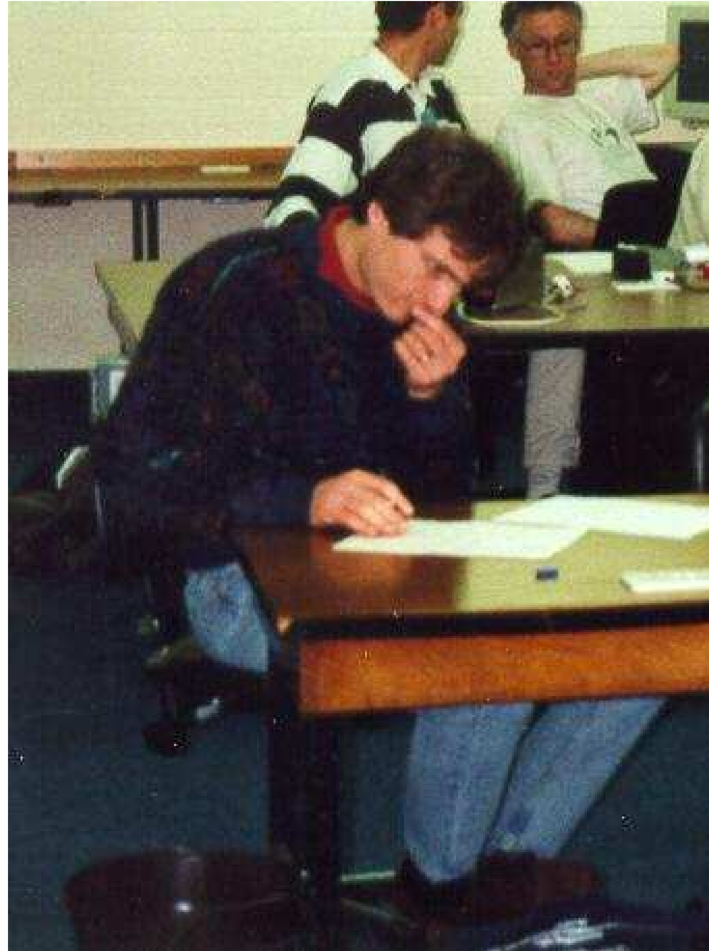
Als Ergebnis der Optimierung erwarten wir ...

```
fun max (x, y)  =  if x > y then x
                   else y
fun abs z       =  let   val x = z
                         val y = −z
                   in    if x > y then x
                         else y
                   end
```

Diskussion:

max ist zuerstmal nur ein Name. Wirt müssen herausfinden, welchen Wert er zur Laufzeit haben kann

$\Longrightarrow$ Wert-Analyse erforderlich !!



Nevin Heintze im australischen Team
des Prolog-Programmier-Wettbewerbs, 1998

Das ganze Bild:

4.1 Eine einfache Zwischensprache

Zur Vereinfachung betrachten wir:

$$\begin{array}{lcl} v & ::= & b \mid (x_1, \ldots, x_k) \mid c\ x \mid \textbf{fn}\, x \Rightarrow e \\ e & ::= & v \mid (x_1\, x_2) \mid (\Box_1\ x) \mid (x_1\, \Box_2\, x_2) \mid \\ & & \textbf{let}\ x_1 = e_1\ \ldots\ x_k = e_k\ \textbf{in}\ e_0\ \textbf{end} \mid \\ & & \textbf{letrec}\ x_1 = e_1\ \ldots\ x_k = e_k\ \textbf{in}\ e_0\ \textbf{end} \mid \\ & & \textbf{case}\ x\ \textbf{of}\ p_1 :\ e_1\ \mid \ldots \mid\ p_k :\ e_k\ \textbf{end} \\ p & ::= & v \mid x \mid c\, x \mid (x_1, \ldots, x_k) \end{array}$$

wobei b eine Konstante ist, x eine Variable, c ein (Daten-)Konstruktor und $\Box_i$ i-stellige Operatoren sind.

Diskussion:

- Konstruktoren und Funktionen sind stets ein-stellig. Dafür gibt es explizite Tupel :-)
- **if**-Ausdrücke und Fall-Unterscheidung in Funktions-Definitionen wird auf **case**-Ausdrücke zurückgeführt.
- In Fall-Unterscheidungen sind nur einfache Muster erlaubt.

  $\Longrightarrow$ Komplizierte Muster müssen zerlegt werden ...
- **let**-Definitionen entsprechen Basis-Blöcken :-)
- Typ-Annotationen an Variablen, Mustern oder Ausdrücken könnten weitere nützliche Informationen enthalten — wir verzichten aber drauf :-)

... im Beispiel:

Die Definition von max sieht dann so aus:

$$
\begin{array}{l}
\textsf{max} \;=\; \textbf{fn}\; x \Rightarrow \;\; \textbf{case}\; x\; \textbf{of}\; (x_1, x_2)\;\; : \\
\qquad\qquad\qquad\qquad \textbf{let}\; z = x_1 < x_2 \\
\qquad\qquad\qquad\qquad \textbf{in}\;\; \textbf{case}\; z \\
\qquad\qquad\qquad\qquad\qquad \textbf{of}\;\; \textsf{True}\; : \;\; x_2 \\
\qquad\qquad\qquad\qquad\qquad \mid\;\; \textsf{False}\; : \;\; x_1 \\
\qquad\qquad\qquad\qquad\qquad \textbf{end} \\
\qquad\qquad\qquad\qquad \textbf{end} \\
\qquad\qquad\qquad \textbf{end}
\end{array}
$$

Entsprechend haben wir für abs :

$$\begin{aligned}\mathsf{abs} \;=\; \mathbf{fn}\ x \Rightarrow\ & \mathbf{let}\ z_1 = -x \\ & \qquad z_2 = (x, z_1) \\ & \mathbf{in}\ (\mathsf{max}\ z_2) \\ & \mathbf{end}\end{aligned}$$

4.2 Eine einfache Wert-Analyse

Idee:

Für jeden Teilausdruck e sammeln wir die Menge $[\![e]\!]^\sharp$ der möglichen Werte von e ...

Sei V die Menge der vorkommenden Konstanten (-Klassen), Konstruktor-Anwendungen und Funktionen. Dann wählen wir als vollständigen Verband natürlich:

$$\mathbb{V} = 2^V$$

Wir stellen wir ein Ungleichungs-System auf:

- Ist e ein Wert d.h. von der Form: $b, c\,x, (x_1, \ldots, x_k)$ oder $\mathbf{fn}\, x \Rightarrow e$ erzeugen wir:

$$[\![e]\!]^\sharp \supseteq \{e\}$$

- Ist $e \equiv (x_1\; x_2)$ und $f \equiv \mathbf{fn}\; x \Rightarrow e_1$, dann

$$[\![e]\!]^\sharp \supseteq (f \in [\![x_1]\!]^\sharp)\,?\,[\![e_1]\!]^\sharp : \emptyset$$
$$[\![x]\!]^\sharp \supseteq (f \in [\![x_1]\!]^\sharp)\,?\,[\![x_2]\!]^\sharp : \emptyset$$

...

- int-Werte, die Operatoren zurück liefern, approximieren wir z.B. durch eine Konstante int.

  Operatoren, die Boolesche Werte liefern, liefern z.B. $\{$True, False$\}$:-)

- Ist $e \equiv \textbf{let } x_1 = e_1 \ldots x_k = e_k \textbf{ in } e_0 \textbf{ end}$. Dann erzeugen wir:

$$\begin{array}{lll} [\![x_i]\!]^\sharp & \supseteq & [\![e_i]\!]^\sharp \\ [\![e]\!]^\sharp & \supseteq & [\![e_0]\!]^\sharp \end{array}$$

- Analog für $e \equiv \textbf{letrec } x_1 = e_1 \ldots x_k = e_k \textbf{ in } e_0 \textbf{ end}$:

$$\begin{array}{lll} [\![x_i]\!]^\sharp & \supseteq & [\![e_i]\!]^\sharp \\ [\![e]\!]^\sharp & \supseteq & [\![e_0]\!]^\sharp \end{array}$$

- Sei $e \equiv \textbf{case}\ x\ \textbf{of}\ p_1\ :\ e_1 \mid \ldots \mid p_k\ :\ e_k\ \textbf{end}$.
  Dann erzeugen wir für $p_i \equiv b$,

$$[\![e]\!]^\sharp \supseteq (b \in [\![x]\!]^\sharp)\,?\,[\![e_i]\!]^\sharp\ :\ \emptyset$$

Ist $p_i \equiv c\,y$ und $v \equiv c\,z$ ein Wert, dann

$$\begin{array}{lll} [\![e]\!]^\sharp & \supseteq & (v \in [\![x]\!]^\sharp)\,?\,[\![e_i]\!]^\sharp\ :\ \emptyset \\ [\![y]\!]^\sharp & \supseteq & (v \in [\![x]\!]^\sharp)\,?\,[\![z]\!]^\sharp\ :\ \emptyset \end{array}$$

Ist $p_i \equiv (y_1, \ldots, y_k)$ und $v \equiv (z_1, \ldots, z_k)$ ein Wert, dann

$$\begin{array}{lll} [\![e]\!]^\sharp & \supseteq & (v \in [\![x]\!]^\sharp)\,?\,[\![e_i]\!]^\sharp\ :\ \emptyset \\ [\![y_j]\!]^\sharp & \supseteq & (v \in [\![x]\!]^\sharp)\,?\,[\![z_j]\!]^\sharp\ :\ \emptyset \end{array}$$

Ist $p_i \equiv y$, dann

$$\begin{array}{lll} [\![e]\!]^\sharp & \supseteq & [\![e_i]\!]^\sharp \\ [\![y]\!]^\sharp & \supseteq & [\![x]\!]^\sharp \end{array}$$

4.3 Eine operationelle Semantik

Idee:

Wir konstruieren eine Big-Step operationelle Semantik, die Ausdrücke auswertet :-)

Konfigurationen:

$$\begin{array}{lcl} c & ::= & (e, env) \\ vc & ::= & (v, env) \\ env & ::= & \{x_1 \mapsto vc_1, \ldots\} \end{array}$$

Werte sind Konfigurationen, in denen der Ausdruck von der Form:
$b, c\,x, (x_1, \ldots, x_k)$ oder $\mathbf{fn}\,x \Rightarrow e$ ist :-)

Umgebungen enthalten nur Werte :-))

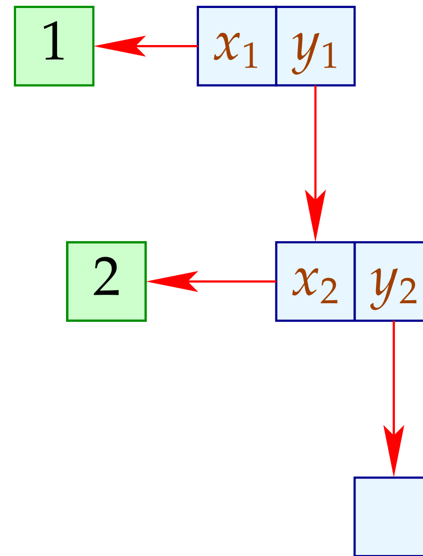
Beispiele für Werte:

$$
\begin{array}{lll}
1 & : & (1, \emptyset) \\
c\,1 & : & (c\ x, \{x \mapsto (1, \emptyset)\}) \\
[1,2] & : & ((x_1, y_1), \{x_1 \mapsto 1, \\
 & & \quad y_1 \mapsto ((x_2, y_2), \{x_2 \mapsto 2, \\
 & & \qquad y_2 \mapsto ((), \emptyset)\})\})
\end{array}
$$

Werte sehen etwas merkwürdig aus :-)

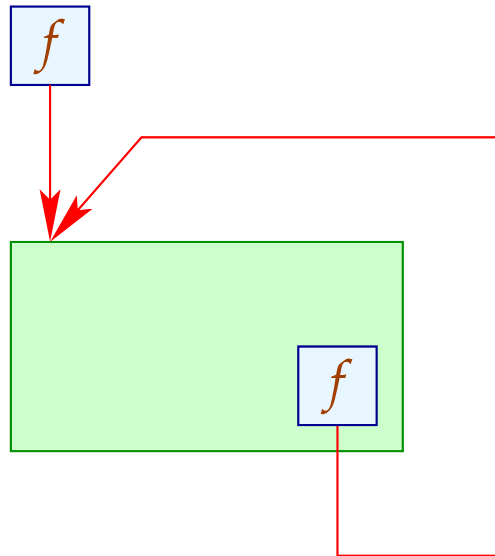
Der Grund ist, dass wir Substitutionen nie ausführen :-)

Alternativ können wir uns die Variablen in den Umgebungen als Speicherzellen vorstellen ...



Achtung:

Rekursive Funktionen führen zu zyklischen Verweis-Strukturen ;-)



Auswege:

- Rekursive Funktionen werden auf dem Toplevel definiert :-)
- Lokale Rekursive Funktionen sind stets nur selbst rekursiv.

  Für diese führen wir einen neuen Operator fix ein ...

Aus: $\textbf{letrec}\ x_1 = e_1\ \textbf{in}\ e_0\ \textbf{end}$

wird: $\textbf{let}\ x_1 = \mathsf{fix}\,(x_1, e_1)\ \textbf{in}\ e_0\ \textbf{end}$

Beispiel: Die append-Funktion

Betrachten wir die Konkatenation von zwei Listen. In ML schreiben wir einfach:

$$\begin{array}{lllcl} \textbf{fun} & \mathsf{app} & [\,] & = & \textbf{fn}\ y \Rightarrow y \\ \mid & \mathsf{app} & (x :: xs) & = & \textbf{fn}\ y \Rightarrow x :: \mathsf{app}\ xs\ y \end{array}$$

In unserer eingeschränkten Zwischensprache sieht das etwas detaillierter aus :-)

$$
\begin{array}{rcl}
\mathsf{app} & = & \mathsf{fix}\,(\mathsf{app},\ \mathbf{fn}\ x \Rightarrow \mathbf{case}\ x \\
& & \quad \mathbf{of}\ \ [\,]\ :\ \ \mathbf{fn}\ y\ \Rightarrow\ y \\
& & \quad \mid\ \ ::z\ :\ \ \mathbf{case}\ z\ \mathbf{of}\ (x_1, x_2)\ :\ \ \mathbf{fn}\ y\ \Rightarrow \\
& & \qquad\qquad \mathbf{let}\ \ a_1 = \mathsf{app}\ x_2 \\
& & \qquad\qquad\qquad a_2 = a_1\ y \\
& & \qquad\qquad\qquad z_1 = (x_1, a_2) \\
& & \qquad\qquad \mathbf{in}\ \ ::z_1 \\
& & \qquad\qquad \mathbf{end} \\
& & \qquad \mathbf{end} \\
& & \quad \mathbf{end}\ )
\end{array}
$$

Die Big-Step Semantik gibt Regeln an, zu welchem Wert sich eine Konfiguration ausrechnen lässt ...

Funktionsanwendung:

$$\frac{\eta\, x_1 = (\mathbf{fn}\; x \;\Rightarrow\; e, \eta_1) \qquad \eta\, x_2 = (v_2, \eta_2) \qquad (e, \eta_1 \oplus \{x \mapsto (v_2, \eta_2)\}) \Longrightarrow (v_3, \eta_3)}{(x_1\; x_2, \eta) \Longrightarrow (v_3, \eta_3)}$$

Lokal rekursive Funktionsanwendung:

$$\frac{\begin{array}{rcl} \eta\, x_1 & = & (\mathsf{fix}\,(f, \mathbf{fn}\; x \;\Rightarrow\; e), \eta_1) \\ \eta\, x_2 & = & (v_2, \eta_2) \\ \multicolumn{3}{l}{(e, \eta_1 \oplus \{f \mapsto (\mathsf{fix}\,(f, \mathbf{fn}\; x \;\Rightarrow\; e), \eta_1),\; x \mapsto (v_2, \eta_2)\}) \Longrightarrow (v_3, \eta_3)} \end{array}}{(x_1\; x_2, \eta) \Longrightarrow (v_3, \eta_3)}$$

Fall-Unterscheidung 1:

$$\frac{\begin{array}{lll} \eta\, x & = & (b, \eta_1) \\ (e_i, \eta) & \Longrightarrow & (v_i, \eta_i) \end{array}}{(\textbf{case}\ x\ \textbf{of}\ p_1\ :\ e_1 \mid \ldots \mid p_k\ :\ e_k\ \textbf{end}, \eta) \Longrightarrow (v_i, \eta_i)}$$

sofern $p_i \equiv b$ das erste auf b passende Muster ist :-)

Fall-Unterscheidung 2:

$$\frac{\eta\, x \;=\; (c\, z, \eta_1) \qquad (e_i, \eta \oplus \{x_i \mapsto (\eta\, z)\}) \Longrightarrow (v_i, \eta_i)}{(\textbf{case}\ x\ \textbf{of}\ p_1\ :\ e_1 \mid \ldots \mid p_k\ :\ e_k\ \textbf{end}, \eta) \Longrightarrow (v_i, \eta_i)}$$

sofern $p_i \equiv c\ x_i$ das erste auf $c\, z$ passende Muster ist :-)

Fall-Unterscheidung 3:

$$\frac{\begin{array}{l}\eta\, x \;=\; ((z_1,\ldots,z_m),\eta_1)\\ (e_i,\eta\oplus\{y_j\mapsto(\eta\, z_j)\mid j=1,\ldots,m\})\Longrightarrow(v_i,\eta_i)\end{array}}{(\textbf{case}\ x\ \textbf{of}\ p_1\ :\ e_1\mid\ldots\mid p_k\ :\ e_k\ \textbf{end},\eta)\Longrightarrow(v_i,\eta_i)}$$

für das erste passende Muster $p_i \equiv (y_1,\ldots,y_m)$:-)

Fall-Unterscheidung 4:

$$\frac{(e_i, \eta \oplus \{x_i \mapsto (\eta\, x)\}) \Longrightarrow (v_i, \eta_i)}{(\textbf{case}\ x\ \textbf{of}\ p_1\ :\ e_1 \mid \ldots \mid p_k\ :\ e_k\ \textbf{end}, \eta) \Longrightarrow (v_i, \eta_i)}$$

sofern $p_i \equiv x_i$ und alle Muster davor fehl schlugen :-)