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 :-))

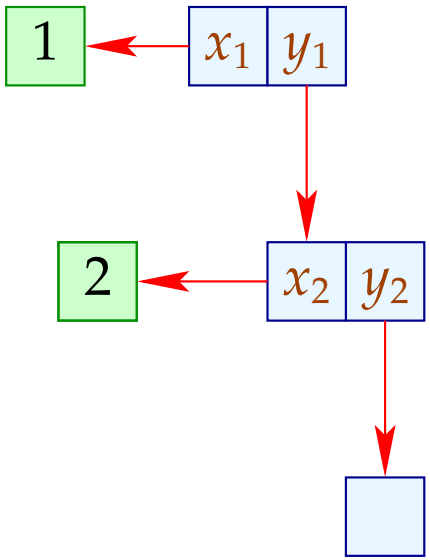
Beispiele für Werte:

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

Werte sehen etwas merkwürdig aus :-)

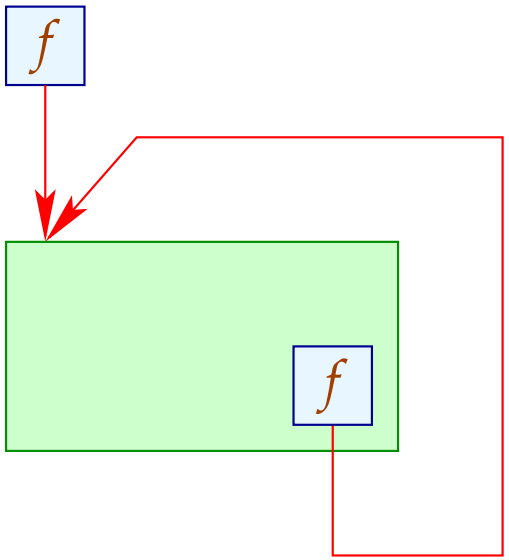
Der Grund ist, dass wir Substitutionen nie ausführen :-)

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:

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

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

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

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:

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

Fall-Unterscheidung 1:

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

sofern $p_i \equiv b$ 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 :-)

Lokale Definitionen:

$$(e_1, \eta) \Longrightarrow (v_1, \eta_1)$$
$$(e_2, \eta \oplus \{x_1 \mapsto (v_1, \eta_1)\}) \Longrightarrow (v_2, \eta_2)$$
$$\ldots$$
$$(e_k, \eta \oplus \{x_1 \mapsto (v_1, \eta_1), \ldots, x_{k-1} \mapsto (v_{k-1}, \eta_{k-1})\}) \Longrightarrow (v_k, \eta_k)$$
$$(e_0, \eta \oplus \{x_1 \mapsto (v_1, \eta_1), \ldots, x_k \mapsto (v_k, \eta_k)\}) \Longrightarrow (v_0, \eta_0)$$

$$(\textbf{let } x_1 = e_1 \ldots x_k = e_k \textbf{ in } e_0 \textbf{ end}, \eta) \Longrightarrow (v_0, \eta_0)$$

Variablen:

$$\frac{\eta\,(x)\;=\;(v_1, \eta_1)}{(x, \eta) \Longrightarrow (v_1, \eta_1)}$$

Korrektheit der Analyse:

Man zeigt für jedes (e, η), das in einer Ableitung für das Programm vorkommt:

- Falls $\eta(x) = (v, \eta_1)$, dann ist $v \in [\![x]\!]^\sharp$.
- Falls $(e, \eta) \Longrightarrow (v, \eta_1)$, dann ist $v \in [\![e]\!]^\sharp$.

Fazit:

$[\![e]\!]^\sharp$ liefert eine Obermenge der Werte, zu denen sich e möglicherweise ausrechnet :-)

4.4 Anwendung: Inlining

Probleme:

- globale Variablen. Das Programm:

$$
\begin{array}{lll}
\textbf{let} & x = 1 & \\
 & f = & \textbf{let} \quad x = 2 \\
 & & \textbf{in} \quad \textbf{fn}\ y \Rightarrow y + x \\
 & & \textbf{end} \\
\textbf{in} & f\ x & \\
\textbf{end} & &
\end{array}
$$

... berechnet offenbar etwas anderes als:

$$
\begin{array}{lll}
\textbf{let} & x = 1 & \\
 & f = & \textbf{let} \quad x = 2 \\
 & & \textbf{in} \quad \textbf{fn}\ y \ \Rightarrow \ y + x \\
 & & \textbf{end} \\
\textbf{in} & \boxed{\begin{array}{ll} \textbf{let} & y = x \\ \textbf{in} & y * x \\ \textbf{end} & \end{array}} & \\
\textbf{end} & &
\end{array}
$$

- rekursive Funktionen. In der Definition:

$$x = \mathsf{fix}\ (\mathsf{foo},\ \textbf{fn}\ y \ \Rightarrow \ \mathsf{foo}\ y)$$

sollten wir foo besser nicht substituieren :-)

Idee 1:

- → Wir machen erstmal die Namen im Programm eindeutig.
- → Dann substituieren wir nur Funktionen, die statisch im Scope der selben globalen Variablen stehn, wie die Anwendung :-)
- → Wir berechnen für jeden Ausdruck alle Funktions-Definitionen mit dieser Eigenschaft :-)

Sei $D[e]$ die Menge der Definitionen, die in e statisch ankommen.

- Für $e \equiv \textbf{let } x_1 = e_1 \ldots x_k = e_k \textbf{ in } e_0 \textbf{ end}$ haben wir:

$$\begin{aligned} D[e_1] &= D \\ &\ldots \\ D[e_k] &= D \cup \{x_1, \ldots, x_{k-1}\} \\ D[e_0] &= D \cup \{x_1, \ldots, x_k\} \end{aligned}$$

- In den anderen Fällen propagiert sich D unverändert zu den Teilausdrücken :-)

  Für $e \equiv \textbf{fn } x \Rightarrow e_1$ haben wir etwa:

$$D[e_1] = D$$

... im Beispiel:

$$
\begin{array}{lll}
\textbf{let} & x = 1 & \\
 & f = & \textbf{let} \quad x_1 = 2 \\
 & & \textbf{in} \quad \textbf{fn}\ y \Rightarrow y + x_1 \\
 & & \textbf{end} \\
\textbf{in} & f\ x & \\
\textbf{end} & &
\end{array}
$$

... steht (nach Umbenennung :-) f für $f\ x$ statisch zur Verfügung. Bei Substitution erhalten wir:

$$
\begin{array}{lll}
\textbf{let} & x = 1 & \\
 & f = & \textbf{let} \quad x_1 = 2 \\
 & & \textbf{in} \quad \textbf{fn}\ y \ \Rightarrow \ y + x_1 \\
 & & \textbf{end} \\
\textbf{in} & \boxed{\begin{array}{l}
\textbf{let} \quad y = x \\
\textbf{in} \quad \textbf{let} \quad x_1 = 2 \\
\qquad\quad \textbf{in} \quad y + x_1 \\
\qquad\quad \textbf{end} \\
\textbf{end}
\end{array}} & \\
\textbf{end} & &
\end{array}
$$

Ersetzen der Variablen-Variablen-Umbenennungen ergibt schließlich:

$$
\begin{array}{l}
\textbf{let} \quad x = 1 \\
\qquad f = \ \textbf{let} \quad x_1 = 2 \\
\qquad\qquad\ \ \textbf{in} \quad \textbf{fn}\ y \ \Rightarrow \ y + x_1 \\
\qquad\qquad\ \ \textbf{end} \\
\textbf{in} \quad \boxed{\begin{array}{l}\textbf{let} \quad x_1 = 2 \\ \textbf{in} \quad x + x_1 \\ \textbf{end}\end{array}} \\
\textbf{end}
\end{array}
$$

Idee 2:

→ Wir benutzen unsere Wert-Analyse.

→ Wir ignorieren globale Variablen :-)

→ Wir substituieren nur Funktionen ohne freie Variablen :-))

Beispiel: Die map-Funktion

```
let f = fn x => x · x
    map = fix (map, fn g => fn x => case x
                                      of [] : []
                                       | :: z : case z of (x1, x2) in
                                                  let y1 = g x1
                                                      m = map g
                                                      y2 = m x2
                                                      z1 = (y1, y2)
                                                  in :: z1
                                                  end
                                      end)
    h = map f
in h list
end
```

- Der formale Parameter g von map ist stets f :-)
- Wir können die Anwendung von f in der Definition von map ersetzen:

$$
\begin{array}{l}
\mathsf{map} = \mathsf{fix}\,(\mathsf{map},\ \mathbf{fn}\ g\ \Rightarrow\ \mathbf{fn}\ x\ \Rightarrow\ \mathbf{case}\ x \\
\qquad\qquad \mathbf{of}\quad [\,]\ :\quad [\,] \\
\qquad\qquad\quad \mid\quad :: z\ :\quad \mathbf{case}\ z\ \mathbf{of}\ (x_1, x_2)\ \mathbf{in} \\
\qquad\qquad\qquad\qquad \mathbf{let}\ y_1 = \boxed{\begin{array}{l}\mathbf{let}\ x = x_1 \\ \mathbf{in}\ x \cdot x \\ \mathbf{end}\end{array}} \\
\qquad\qquad\qquad\qquad\qquad m = \mathsf{map}\ g \\
\qquad\qquad\qquad\qquad\qquad y_2 = m\ x_2 \\
\qquad\qquad\qquad\qquad\qquad z_1 = (y_1, y_2) \\
\qquad\qquad\qquad\qquad \mathbf{in}\ :: z_1 \\
\qquad\qquad\qquad\qquad \mathbf{end} \\
\qquad\qquad \mathbf{end}) \\
h = \mathsf{map}\ \ f
\end{array}
$$

- Noch mehr könnten wir sparen, wenn wir die spezialisierte Funktion $h = \mathsf{map}\ f$ direkt definieren könnten :-)
- Dazu müssen wir überall in der Definition von map das Muster $\boxed{\mathsf{map}\ g}$ durch h ersetzen ...

$$\Longrightarrow \quad \text{fold-Transformation} \quad \text{:-)}$$

- Alle weiteren Vorkommen von g müssen durch (die Definition von) f ersetzt werden ...

  // kommt hier nicht vor :-)

$$
\begin{array}{l}
\mathsf{map} = \mathsf{fix}\,(\boxed{\mathsf{map},\ \mathbf{fn}\ g\ \Rightarrow}\ \mathbf{fn}\ x\ \Rightarrow\ \mathbf{case}\ x \\
\qquad\qquad \mathbf{of}\quad [\,]\ :\quad [\,] \\
\qquad\qquad\quad \mid\quad :: z\ :\quad \mathbf{case}\ z\ \mathbf{of}\ (x_1, x_2)\ \mathbf{in} \\
\qquad\qquad\qquad\qquad \mathbf{let}\ y_1 = \boxed{\begin{array}{l}\mathbf{let}\ x = x_1 \\ \mathbf{in}\ x \cdot x \\ \mathbf{end}\end{array}} \\
\qquad\qquad\qquad\qquad\qquad m = \boxed{\mathsf{map}\ g} \\
\qquad\qquad\qquad\qquad\qquad y_2 = m\ x_2 \\
\qquad\qquad\qquad\qquad\qquad z_1 = (y_1, y_2) \\
\qquad\qquad\qquad\qquad \mathbf{in}\ :: z_1 \\
\qquad\qquad\qquad\qquad \mathbf{end} \\
\qquad\qquad \mathbf{end}) \\
h = \mathsf{map}\ \ f
\end{array}
$$

$$
\begin{array}{l}
h = \mathsf{fix}\,(h, \mathbf{fn}\ x\ \Rightarrow\ \mathbf{case}\ x \\
\qquad\qquad \mathbf{of}\ \ [\,]\ :\ \ [\,] \\
\qquad\qquad\ \ \mid\ \ ::\, z\ :\ \ \mathbf{case}\ z\ \mathbf{of}\ (x_1, x_2)\ \mathbf{in} \\
\qquad\qquad\qquad\qquad \mathbf{let}\ y_1 = \boxed{\begin{array}{l}\mathbf{let}\ x = x_1 \\ \mathbf{in}\ x \cdot x \\ \mathbf{end}\end{array}} \\
\qquad\qquad\qquad\qquad\qquad\ m = \boxed{h} \\
\qquad\qquad\qquad\qquad\qquad\ y_2 = m\ x_2 \\
\qquad\qquad\qquad\qquad\qquad\ z_1 = (y_1, y_2) \\
\qquad\qquad\qquad\qquad \mathbf{in}\ \ ::\, z_1 \\
\qquad\qquad\qquad\qquad \mathbf{end} \\
\qquad\qquad \mathbf{end})
\end{array}
$$

Beseitigung von Variablen-Variablen-Umspeicherungen liefert:

$$
\begin{array}{l}
h = \mathsf{fix}\,(h, \mathbf{fn}\ x\ \Rightarrow\ \mathbf{case}\ x \\
\qquad\qquad \mathbf{of}\ \ [\,]\ :\ \ [\,] \\
\qquad\qquad\ \ \mid\ \ :: z :\ \ \mathbf{case}\ z\ \mathbf{of}\ (x_1, x_2)\ \mathbf{in} \\
\qquad\qquad\qquad\qquad \mathbf{let}\ y_1 = x_1 \cdot x_1 \\
\qquad\qquad\qquad\qquad\quad\ y_2 = h\ x_2 \\
\qquad\qquad\qquad\qquad\quad\ z_1 = (y_1, y_2) \\
\qquad\qquad\qquad\qquad \mathbf{in}\ \ :: z_1 \\
\qquad\qquad\qquad\qquad \mathbf{end} \\
\qquad\qquad \mathbf{end})
\end{array}
$$