Value Conversion in IL1 after Lambda Hoisting

Lee Gao (lg342)

April 16, 2013

Given the closure-conversion and then hoisted restricted language IL1 detailed below

$$\begin{aligned}
v &::= n \mid x \mid \mathsf{halt} \\
e &::= v \mid v_0 + v_1 \mid (v_0, \cdots, v_n) \mid \pi_n v \\
c &::= \mathsf{let}\, x = e \,\mathsf{in}\, c \mid v_0\ v_1\ v_2 \mid v_0\ v_1
\end{aligned}$$

we want to "lower" numbers and halt into expressions (as Val used in bindings only) and leave only variables as values, so in effect our language will now look like

$$\begin{aligned}
v &::= x \\
e &::= v \mid \mathsf{val}(n) \mid \mathsf{val}(\mathsf{halt}) v_0 + v_1 \mid (v_0, \cdots, v_n) \mid \pi_n v \\
c &::= \mathsf{let}\, x = e \,\mathsf{in}\, c \mid v_0\ v_1\ v_2 \mid v_0\ v_1
\end{aligned}$$

We want to define a set of "lowering" translation $\mathcal{LV}[\![v]\!], \mathcal{LE}[\![e]\!], \mathcal{LC}[\![c]\!]$ that binds all non-variable values (integers and halts) into their own variables. Therefore, we need to have both the value and the expressions translation be able to be abstracted as bindings.

$$\begin{aligned}
\mathcal{LV}[\![v]\!] &: (\mathsf{var} \times e)\mathsf{list} \times \mathsf{var} \\
\mathcal{LE}[\![e]\!] &: (\mathsf{var} \times e)\mathsf{list} \times \mathsf{e} \\
\mathcal{LC}[\![c]\!] &: c
\end{aligned}$$

We will use the notation

$$\mathsf{let}\, x_i = e_i \,\mathsf{in}\, x$$

to mean

$$([(x_i, e_i); \cdots (x_n, e_n)], x)$$

and respective syntax sugary for expressions.

1 Values

$$\begin{aligned}
\mathcal{LV}[\![n]\!] &= \mathsf{let}\, x = n \,\mathsf{in}\, x \\
\mathcal{LV}[\![x]\!] &= ([], x) \\
\mathcal{LV}[\![\mathsf{halt}]\!] &= \mathsf{let}\, x = \mathsf{halt} \,\mathsf{in}\, x
\end{aligned}$$

2 Expressions

Notice immediately that we only ever translate e in the context of let expressions, so we don't have to translate the case of just a value.

$$\begin{aligned}
\mathcal{LE}[\![v_0 + v_1]\!] &= \mathsf{let}(l_0, x_0) = \mathcal{LV}[\![v_0]\!] \mathsf{~in~} \mathsf{let}(l_1, x_1) = \mathcal{LV}[\![v_1]\!] \mathsf{~in~} (l_0 \cup l_1, x_0 + x_1) \\
\mathcal{LE}[\![(v_0, \cdots, v_n)]\!] &= \mathsf{let}(l_0, x_0) = \mathcal{LV}[\![v_0]\!] \mathsf{~in} \cdots \mathsf{let}(l_n, x_n) = \mathcal{LV}[\![v_n]\!] \mathsf{~in~} (\bigcup l_k, (x_1, \cdots, x_n)) \\
\mathcal{LE}[\![\pi_n v]\!] &= \mathsf{let}(l, x) = \mathcal{LV}[\![v]\!] \mathsf{~in~} (l, \pi_n x)
\end{aligned}$$

3 Commands

$$\begin{aligned}
\mathcal{LC}[\![\mathsf{let}\, x = e \mathsf{~in~} c]\!] &= \mathsf{let}(x_i = e_i, e') = \mathcal{LE}[\![e]\!] \mathsf{~in~} \qquad \underbrace{\mathsf{let}\, x_i = e_i, x = e' \mathsf{~in~} \mathcal{LC}[\![c]\!]}_{\text{this is the actual command returned}} \\
\mathcal{LC}[\![v_0\ v_1]\!] &= \mathsf{let}(x_i = e_i, x) = \mathcal{LV}[\![v_0]\!], (y_j = e'_j, y) = \mathcal{LV}[\![v_1]\!] \mathsf{~in~} \qquad \mathsf{let}\, x_i = e_i, y_j = e'_j \mathsf{~in~} x\ y
\end{aligned}$$

and without loss of generality, the same applies to the 3 argument case.