Closure conversion

February 28, 2017

Abstract

To this point we've been working with functions that have free variables, like $\lambda x.\ f\ g\ x$. In theory, instantiation of free variables is implemented by substitution, but hardware does not support this operation. Thus we now define a language, IL-Closure, in which terms explicitly carry environments defining their free variables, "closing" over them. Then, we show how to translate IL-CPS into IL-Closure.

1 A brief note on implementation

Closures are expensive, and a robust implementation should avoid creating them whenever possible. So, though this translation pass is needed to take care of higher-order usage of functions, "known" defined at the top-level should not be converted into closures.

2 Examples

Consider the term

$$\lambda x : \texttt{int}.\ x + y + z$$

Translating this to a closure, we get

$$\begin{aligned}
&\langle \lambda x : \texttt{int}.\ \lambda env : \texttt{int} \times \texttt{int}. \\
&\quad \texttt{let}\ y = \pi_0\ env\ \texttt{in} \\
&\quad \texttt{let}\ z = \pi_1\ env\ \texttt{in} \\
&\quad x + y + z \\
&, \langle y, z \rangle \rangle
\end{aligned}$$

As a consequence, for function application

$$e\ 5$$

we need to translate as well.

$$\begin{aligned}&\texttt{let } f = \pi_0\ e \texttt{ in}\\&\texttt{let } env = \pi_1\ e \texttt{ in}\\&f\ env\ 5\end{aligned}$$

But in order to get type-directed translation to go through smoothly, we will need to be somewhat clever. Consider the following "problem" case.

$$\texttt{if } b \texttt{ then } (\lambda x : \texttt{int}.\ x + y) \texttt{ else } (\lambda x : \texttt{int}.\ x + y + z)$$

We want to be able to give some τ_{env} such that

$$\overline{\texttt{int} \to \texttt{int}} = (\texttt{int} \to \tau_{env} \to \texttt{int}) \times \tau_{env}$$

But the above example shows that some terms of type $\texttt{int} \to \texttt{int}$ do not admit one correct choice τ_{env} type. The solution is to make the type existentially quantified.

$$\overline{\texttt{int} \to \texttt{int}} = \exists \alpha_{env} : \tau_{env}.\ (\texttt{int} \to \alpha_{env} \to \texttt{int}) \times \alpha_{env}$$

3 Syntax and judgements

The syntax for IL-Closure is the same as the syntax for IL-CPS. The two principal typing judgements for this language are

$$\begin{aligned}\Delta; \Gamma \vdash e : 0\\ \Delta; \Gamma \vdash v : \tau\end{aligned}$$

The rule of interest is as follows.

$$\frac{\Delta \vdash \tau \ \texttt{type} \qquad \Delta; \cdot, x : \tau \vdash e : 0}{\Delta; \Gamma \vdash \lambda x : \tau.\ e : \neg\tau}\ \text{3A}$$

4 Translation

Type translation is straightforward mapping through, except at arrow types.

$$\begin{aligned}\overline{\alpha} &= \alpha\\ &\ldots\\ \overline{\tau_1 \to \tau_2} &= \exists \tau_{env}.\ (\tau_1 \to \tau_{env} \to \tau_2) \times \tau_{env}\end{aligned}$$

The rules of interest are as follows.

4A

$$\frac{\Delta \vdash \tau : \texttt{type} \qquad \Delta; \Gamma, x : \tau \vdash e : 0 \rightsquigarrow \overline{e} \qquad \Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n}{\Delta; \Gamma \vdash \lambda x : \tau.\ e : \neg\tau \rightsquigarrow \begin{array}{l} \texttt{pack}\ [\overline{\tau_1} \times \cdots \times \overline{\tau_n}, \\ \quad \langle (\lambda y : \overline{\tau} \times (\overline{\tau_1} \times \cdots \times \overline{\tau_n}). \\ \qquad \texttt{let}\ x = \pi_1\ y\ \texttt{in} \\ \qquad \texttt{let}\ env = \pi_2\ y\ \texttt{in} \\ \qquad \texttt{let}\ x_1 = \pi_1\ env\ \texttt{in} \\ \qquad \cdots \\ \qquad \texttt{let}\ x_n = \pi_n\ env\ \texttt{in} \\ \qquad \overline{e}), \\ \quad \langle x_1, \cdots, x_n \rangle \rangle] \\ \texttt{as}\ \exists \alpha_{env} : \texttt{type}.\ \neg(\overline{\tau} \times \alpha_{env}) \times \alpha_{env} \end{array}}$$

4B

$$\frac{\Delta; \Gamma \vdash v_1 : \neg\tau \rightsquigarrow \overline{v_1} \qquad \Delta; \Gamma \vdash v_2 : \tau \rightsquigarrow \overline{v_2}}{\Delta; \Gamma \vdash v_1\ v_2 : 0 \rightsquigarrow \begin{array}{l} \texttt{unpack}\ [\alpha_{env}, x] = \overline{v_1}\ \texttt{in} \\ \quad \texttt{let}\ f = \pi_1\ x\ \texttt{in} \\ \quad \texttt{let}\ env = \pi_2\ x\ \texttt{in} \\ \quad f \langle \overline{v_2}, env \rangle \end{array}}$$