Closure Conversion in IL1

Lee Gao (lg342)

April 15, 2013

Given the language IL1 detailed below

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

with the natural CBV semantic, we want to do a closure-conversion translation so that no abstractions contain free variables. To do this, we will need a function

$$\langle \cdot \rangle : \mathsf{var} \to \mathbb{N}$$

which uniquely maps a given variable x to a natural number such that it is as compact as possible. ($\langle \cdot \rangle = \min_{f \text{ bijection}} \max_{x \in \mathsf{var}} f(x)$) We can do this easily by inspection of the program. Now, we will represent a closure environment ρ as a tuple in IL1 where we can represent

$$[\![x]\!]\rho = \pi_{\langle x \rangle}\ \rho$$

There's an immediate problem: x is in the v domain, but its translation is now in the e domain! This is unsettling, but we can overcome this.
We will define 3 closure conversion translation functions corresponding to each of the v, e, and the c domains. The point is that the closure conversion will still be embeded in IL1, from which we can just do an immediate lambda lifting to get to IL2 (with some minor reordering)

$$\begin{aligned}
&\mathcal{V}[\![v]\!]\rho : e\\
&\mathcal{E}[\![e]\!]\rho : (\mathsf{var} \times e)\ \mathsf{list} \times e\\
&\mathcal{C}[\![c]\!]\rho : c
\end{aligned}$$

In order to lift values, we will need to be able to do expression-level operations on them, so we promote them during translation. In order to lift expressions, we may need to bind to expressions, so we will need to keep track of those bindings for the commands. Commands are just business as usual. Note that because we're not allowed to have applications in values and expressions, we need to η reduce these translations! (So in OCaml, you will literally add this in as part of the translation function, not as an application in the output)

1 Values

We start with the intuitive cases

$$\begin{aligned}
\mathcal{V}[\![n]\!]\rho &= n\\
\mathcal{V}[\![x]\!]\rho &= \pi_{\langle x \rangle}\ \rho
\end{aligned}$$

next, we need to translate a λ. We will represent thses closures as a pair of environment and the lambda code:

```
$\mathcal{V}[\![\lambda k x.c]\!]\rho$ = ($\rho$, $\lambda p$. let $\rho'$ = $\pi_1$ p in
                          let y = $\pi_2$ p in
                          let $x_i$ = $\pi_i \rho'$ in  ($\forall x_i \in \mathsf{dom}(\langle\cdot\rangle)$)
                          let $x_{\langle x \rangle}$ = y in
                          let $\rho''$ = $(x_1, \cdots, x_n)$ in
                          $\mathcal{C}[\![c]\!]\rho''$)
```

convince yourself that this is okay. The case for the non-administrative lambdas are similar (add in a case for $x_{\langle k \rangle}$ too), so all we're left with is just

$$\mathcal{V}[\![\mathsf{halt}]\!]\rho = \mathsf{halt}$$

2 Expressions

We will use the notation

$$\mathsf{let}\, x_0 = e_0, \cdots, x_n = e_n \,\mathsf{in}\, e$$

to denote the pair

$$([(x_0, e_0), \cdots, (x_n, e_n)], e)$$

let's get to the translation

$$\begin{aligned} \mathcal{E}[\![v]\!]\rho &= (\varnothing, \mathcal{V}[\![v]\!]\rho) \\ \mathcal{E}[\![v_0 + v_1]\!]\rho &= \mathsf{let}\, x_0 = \mathcal{V}[\![v_0]\!]\rho, x_1 = \mathcal{V}[\![v_1]\!]\rho \,\mathsf{in}\, x_0 + x_1 \\ \mathcal{E}[\![(v_1, \cdots, v_n)]\!]\rho &= \mathsf{let}\, x_i = \mathcal{V}[\![v_i]\!]\rho \,\mathsf{in}\, (x_1, \cdots, x_n) \\ \mathcal{E}[\![\pi_n v]\!]\rho &= \mathsf{let}\, x_0 = \mathcal{V}[\![v]\!]\rho \,\mathsf{in}\, \pi_n x_0 \end{aligned}$$

3 Commands

This is the tricky one. Let's start with applications:

```
$\mathcal{C}[\![v_0\ v_1]\!]\rho$ = let $fn$ = $\mathcal{V}[\![v_0]\!]\rho$ in
             let $\rho'$ = $\pi_1 fn$ in
             let $f$ = $\pi_2 fn$ in
             let $v' = \mathcal{V}[\![v_1]\!]\rho$ in
             let $arg$ = $(\rho', v')$ in
             $f$ $arg$
```

the function call with continuation version is also similar. Next, we need to do the let case. This is the tricky one because we need to capture the new binding inside the environment as well!
Suppose that in OCaml, we already have $[\mathsf{let}\, y_i = e_i \,\mathsf{in}\, e'] = \mathcal{E}[\![e]\!]\rho$, then we can make the translation as such

```
$\mathcal{C}[\![\mathsf{let}\, x = e \,\mathsf{in}\, c]\!]\rho$ = let $y_i$ = $e_i$ in
                    let $y$ = $e'$ in
```

```
                        let $x_i = \pi_i\rho$ in
                        let $x_{\langle x\rangle} = y$ in
                        let $\rho' = (x_1, \cdots, x_n)$ in
                        $\mathcal{C}[\![c]\!]\rho'$
```

In the above, the expansion $\mathsf{let}\, y_i = e_i \,\mathsf{in} \cdots$ is required in order to compute $y = e'$. Wlog, the case for ifp is much easier.