Extended Abstract Machine for Prettyprinting Intermediate Computations

January 12, 2017

1 Compilation Scheme

$$
\begin{aligned}
\mathcal{C}(i) &= \text{INT}(i) \\
\mathcal{C}(b) &= \text{BOOL}(b) \\
\mathcal{C}(a \oplus b) &= \mathcal{C}(a); \mathcal{C}(b); \text{OP}(\oplus) \\
\mathcal{C}(a = b) &= \mathcal{C}(a); \mathcal{C}(b); \text{EQ} \\
\mathcal{C}(\underline{n}) &= \text{ACCESS}(n) \\
\mathcal{C}(\lambda a) &= \text{CLOSURE}(\mathcal{C}(a); \text{RETURN}) \\
\mathcal{C}(\texttt{let}\ a\ \texttt{in}\ b) &= \mathcal{C}(a); \text{LET}; \mathcal{C}(b); \text{ENDLET} \\
\mathcal{C}(ab) &= \mathcal{C}(a); \mathcal{C}(b); \text{APPLY} \\
\mathcal{C}(\texttt{if}\ a\ \texttt{then}\ b\ \texttt{else}\ c) &= \mathcal{C}(\lambda b); \mathcal{C}(\lambda c); \mathcal{C}(a); \text{IF}
\end{aligned}
$$

e.g `let x = 1 in let y = 2 in x + y` compiles to:

2 Evalution Scheme

Machine state before			Machine state after		
Code	Env	Stack	Code	Env	Stack
INT$(i); c$	e	s	c	e	$i.s$
BOOL$(b); c$	e	s	c	e	$b.s$
OP$(\oplus); c$	e	$i.i'.s$	c	e	$\oplus(i, i').s$
EQ$; c$	e	$i.i'.s$	c	e	$(i = i').s$
ACCESS$(n); c$	e	s	c	e	$e(n).s$
CLOSURE$(c'); c$	e	s	c	e	$c'[e].s$
LET$; c$	e	$v.s$	c	$v.e$	s
ENDLET$; c$	$v.e$	s	c	e	s
APPLY;c	e	$v.c'[e'].s$	c'	$v.e'$	$c.e.s$
RETURN;c	e	$v.c'.e'.s$	c'	e'	$v.s$
IF;c	e	$\mathsf{T}.c'[e'].c''[e''].s$	c'	e'	$c[e].s$
IF;c	e	$\mathsf{F}.c'[e'].c''[e''].s$	c''	e''	$c[e].s$

The final result is at the top of the stack when the code is empty.

3 Decompilation Scheme

We need to be able to decompile:

- Any program which has been compiled by the compilation scheme above.
- Certain incomplete evaluations under the evaluation scheme above. That is to say, given (c, e, s) we can decompile a program which represents the evaluation at that stage.

We need not be able to decompile arbitrary (c, e, s) triples.