1 Interpretation

Operational semantics: pretty-printing of the `byterun/interp.c` file.

A value is a word which represents either an integer or a pointer in the heap.

st is the stack.

ac is the accumulator.

cp is the code pointer.

en is the environment.

ea is the number of extra arguments provided by the caller.

$\mathsf{block}_{closure}(c, e)$ is a closure.

$global\_data$ is a global variable.

1.1 Basic Stack Operations

$$\frac{m[cp] = \mathtt{ACC}\ x}{\langle st, ac, cp, en, ea\rangle \to \langle st, st[x], cp+2, en, ea\rangle}$$

$$\frac{m[cp] = \mathtt{PUSH}}{\langle st, ac, cp, en, ea\rangle \to \langle ac \cdot st, ac, cp+1, en, ea\rangle}$$

$$\frac{m[cp] = \mathtt{POP}\ x}{\langle st[0]\cdots st[x-1] \cdot st, ac, cp, en, ea\rangle \to \langle st, ac, cp+2, en, ea\rangle}$$

$$\frac{m[cp] = \mathtt{ASSIGN}\ n}{\begin{array}{c}\langle st[0]\cdots st[n-1] \cdot st[n] \cdot st, ac, cp, en, ea\rangle \to \\ \langle st[0]\cdots st[n-1] \cdot ac \cdot st, ac, cp+2, en, ea\rangle\end{array}}$$

1.2 Access in Environment

$$\frac{m[cp] = \mathtt{ENVACC}\ p}{\begin{array}{c}\text{let } en = \mathsf{block}_{tag}(f_0, \ldots) \text{ in} \\ \langle st, ac, cp, en, ea\rangle \to \langle st, en[p], cp+2, en, ea\rangle\end{array}}$$

1.3 Function Application

$$\frac{m[cp] = \mathtt{APPLY}\ x}{\begin{array}{c}\langle st[0]\cdots st[x-1] \cdot st, \mathsf{block}_{closure}(c, e), cp, en, ea\rangle \to \\ \langle st[0]\cdots st[x-1] \cdot cp+2 \cdot en \cdot ea \cdot st, \mathsf{block}_{closure}(c, e), c, \mathsf{block}_{closure}(c, e), ea-1\rangle\end{array}}$$

`APPLY` instructions call the closure contained in the accumulator.

$$\frac{m[cp] = \mathtt{RETURN}\ k}{\begin{array}{c}\langle st[0]\cdots st[k-1] \cdot st[k] \cdot st[k+1] \cdot st[k+2] \cdot st, ac, cp, en, 0\rangle \to \\ \langle st, ac, st[k], st[k+1], st[k+2]\rangle\end{array}}$$

The `RETURN` instruction terminates the body of a function. Here the function has been fully applied and it simply returns to the caller.

$$\frac{m[cp] = \mathtt{RETURN}\ k}{\begin{array}{c}\langle st[0]\cdots st[k-1] \cdot st, \mathsf{block}_{closure}(c, e), cp, en, ea\rangle \to \\ \langle st, \mathsf{block}_{closure}(c, e), c, \mathsf{block}_{closure}(c, e), ea-1\rangle\end{array}}$$

There it jumps to the closure contained in the accumulator.

$$\frac{m[cp] = \mathtt{RESTART}}{\begin{array}{c}\text{let } en = \mathsf{block}_{tag}(f_0, \ldots, f_{n-1}) \text{ in} \\ \langle st, ac, cp, en, ea\rangle \to \langle en[2]\cdots en[n-1] \cdot st, ac, cp+1, en[1], ea+n-2\rangle\end{array}}$$

Restart partially applied functions.

$$\frac{m[cp] = \texttt{GRAB}\ x}{\langle st, ac, cp, en, ea \geq x\rangle \to \langle st, ac, cp+2, en, ea-x\rangle}$$

If $ea \geq x$, not all the arguments have been consumed. We go throw the body of the closure.

$$\frac{m[cp] = \texttt{GRAB}\ x}{\begin{array}{c}\langle st[0]\cdots st[ea]\cdot st[ea+1]\cdot st[ea+2], st[ea+3], ac, cp, en, ea < x\rangle \to \\ \langle st, \mathsf{block}_{closure}(cp-1, en, st[0], \ldots, st[ea]), st[ea+1], st[ea+2], st[ea+3]\rangle\end{array}}$$

If $ea < x$, all the arguments have been consumed. We save back the environment and those arguments for further execution and go back to where we come from.

`CLOSURE` instructions build a closure and put it in the accumulator.

$$\frac{m[cp] = \texttt{CLOSURE}\ 0, l}{\begin{array}{c}\langle st, ac, cp, en, ea\rangle \to \\ \langle st, \mathsf{block}_{closure}(cp+3+l), cp+3, en, ea\rangle\end{array}}$$

$$\frac{m[cp] = \texttt{CLOSURE}\ v, l}{\begin{array}{c}\langle st[0]\cdots st[v-2]\cdot st, ac, cp, en, ea\rangle \to \\ \langle st, \mathsf{block}_{closure}(cp+3+l, ac, st[0], \ldots, st[v-2]), cp+3, en, ea)\rangle\end{array}}$$

`CLOSUREREC` instructions build a closure and put it in the accumulator.

$$\frac{m[cp] = \texttt{CLOSUREREC}\ 1, 0, l}{\begin{array}{c}\langle st, ac, cp, en, ea\rangle \to \\ \text{let } cl = \mathsf{block}_{closure}(cp+3+l) \text{ in} \\ \langle cl\cdot st, cl, cp+4, en, ea\rangle\end{array}}$$

$$\frac{m[cp] = \texttt{CLOSUREREC}\ 1, v, l}{\begin{array}{c}\langle st[0]\cdots st[v-2]\cdot st, ac, cp, en, ea\rangle \to \\ let\ cl = \mathsf{block}_{closure}(cp+3+l, ac, st[0], \ldots, st[v-2]) \text{ in} \\ \langle cl\cdot st, cl, cp+4, en, ea\rangle\end{array}}$$

$$\frac{m[cp] = \texttt{OFFSETCLOSURE0}}{\begin{array}{c}let\ en = \mathsf{block}_{closure}(c, e) \text{ in} \\ \langle st, ac, cp, en, ea\rangle \to \langle st, en, cp+1, en, ea\rangle\end{array}}$$

Recursive calls.

TODO: APPTERM

TODO: mutual recursion

1.3.1 Access to Global Variables

$$\frac{m[cp] = \texttt{GETGLOBAL}\ p}{\langle st, ac, cp, en, ea\rangle \to \langle st, global\_data[p], cp+2, en, ea\rangle}$$

$$\frac{m[cp] = \texttt{SETGLOBAL}\ p}{\begin{array}{c}\langle st, ac, cp, en, ea\rangle \to \langle st, 0, cp+2, en, ea\rangle \\ global\_data[p] \leftarrow ac\end{array}}$$

1.3.2 Allocation of Blocks

$$\frac{m[cp] = \texttt{MAKEBLOCK1}\ tag}{\langle st, ac, cp, en, ea\rangle \to \langle st, \mathsf{block}_{tag}(ac), cp+2, en, ea\rangle}$$

$$\frac{m[cp] = \texttt{MAKEBLOCK2}\ tag}{\langle st[0]\cdot st, ac, cp, en, ea\rangle \to \langle st, \mathsf{block}_{tag}(ac, st[0]), cp+2, en, ea\rangle}$$

1.3.3 Access to Components of Blocks

$$\frac{m[cp] = \texttt{GETFIELD}\ p}{\begin{array}{c}\text{let } ac = \mathsf{block}_{tag}(f_0, \ldots, f_n) \text{ in} \\ \langle st, ac, cp, en, ea\rangle \to \langle st, ac[p], cp+1, en, ea\rangle\end{array}}$$

1.3.4 Branches and Conditional Branches

$$\frac{m[cp] = \texttt{BRANCH}\, l}{\langle st, ac, cp, en, ea\rangle \rightarrow \langle st, ac, cp + 1 + l, en, ea\rangle}$$

$$\frac{m[cp] = \texttt{BRANCHIF}\, l}{\langle st, 1, cp, en, ea\rangle \rightarrow \langle st, 1, cp + 1 + l, en, ea\rangle} \qquad \frac{m[cp] = \texttt{BRANCHIF}\, l}{\langle st, 0, cp, en, ea\rangle \rightarrow \langle st, 0, cp + 2, en, ea\rangle}$$

1.3.5 Calling C Functions

$$\frac{m[cp] = \texttt{C\_CALLN}\; n\;\; cfunc}{\langle a_1 \cdots a_{n-1} \cdot st, a_0, cp, en, ea\rangle \rightarrow \langle st, cfunc(a_0, \ldots, a_{n-1}), cp + 2, en, ea\rangle}$$

1.3.6 Integer Constants

$$\frac{m[cp] = \texttt{CONST}\, x}{\langle st, ac, cp, en, ea\rangle \rightarrow \langle st, x, cp + 1, en, ea\rangle}$$

1.3.7 Integer Arithmetic

$$\frac{m[cp] = \texttt{ADDINT}}{\langle st[0] \cdot st, ac, cp, en, ea\rangle \rightarrow \langle st, st[0] + ac, cp + 1, en, ea\rangle}$$

$$\frac{m[cp] = \texttt{MULINT}}{\langle st[0] \cdot st, ac, cp, en, ea\rangle \rightarrow \langle st, st[0] * ac, cp + 1, en, ea\rangle}$$

$$\frac{m[cp] = \texttt{EQ}}{\langle st[0] \cdot st, ac, cp, en, ea\rangle \rightarrow \langle st, st[0] = ac, cp + 1, en, ea\rangle}$$

$$\frac{m[cp] = \texttt{OFFSETINT}\, i}{\langle st, ac, cp, en, ea\rangle \rightarrow \langle st, ac + i, cp + 2, en, ea\rangle}$$