Semantics of $\pm$C(i.e. extended C$--$)

Villani Neven, ENS Paris-Saclay
Programmation 1 – Compilateur COCass

3 janvier 2021

Retranscribed and extended from http://www.lsv.fr/~goubault/CoursProgrammation/minic.html and http://www.lsv.fr/~goubault/CoursProgrammation/prog1_sem1.pdf.

Notation

$\mathbb{Z}_{64}$ is the set of 64-bit signed integers, in which all calculations are done when not specified otherwise.

We write $(\rho : \mathcal{S} \to \mathbb{Z}_{64}) \in \mathcal{P}$ the environment, where $\mathcal{S}$ is the set of names of variables and functions, $(\mu : \mathbb{Z}_{64} \to \mathbb{Z}_8) \in \mathcal{M}$ the memory.
μ is read by blocks of 8 bytes : $\mu^{64}(i) \triangleq \sum_{k=0}^{7} 2^{8k}\mu(i+k)$.
$\rho_g \in \mathcal{P}$ is the global environment.

A flag is defined as an element of $\mathcal{E} \triangleq S \sqcup \{\texttt{brk}, \texttt{ret}, \texttt{cnt}, \texttt{nil}\}$: either an exception string or a special control flow keyword.
Intuitively, $\rho, \mu, \chi, v \vdash c \Rightarrow \rho', \mu', \chi', v'$ means that when c is executed under the environment ρ with the memory μ, the flag χ, and the previous value v, it updates it to the new environment and memory ρ' and μ', raises χ', and changes the value to v'. Variants are used for toplevel declarations (no χ nor v but fun is added), and expressions (ρ is never modified and thus does not appear on the right side)

In addition, we write fun : $\mathbb{Z}_{64} \to \texttt{code}$, a wrapper around $\pm$C functions : $\text{fun}(a)(p_1, \cdots, p_n) = c$ updates the environment with $p_1, \cdots, p_n$ and executes the body of the function whose definition was given by the code c and stored at a. This way of considering functions allows in particular for function pointers.

For $\mu \in \mathcal{M}, v \in \mathbb{Z}_8, x \in \mathbb{Z}_{64}$ we write $\mu[x \mapsto v] : \begin{cases} x \mapsto v \\ y \mapsto \mu(y) \quad y \in \operatorname{dom}\mu \setminus \{x\} \end{cases}$
However we will usually use $\mu^{64}[x \mapsto v] \triangleq \mu[x+k \mapsto v_k \mid 0 \leqslant k < 8,\ v = \sum_{k=0}^{8} 2^{8k} v_k]$, i.e. the memory is written 8 bytes at a time.

A similar notation is used for ρ, ρ_g and fun.
$alloc^i : \mathcal{M} \to \mathcal{P}(\mathbb{Z}_{64})$ is such that if $k \in alloc^i(\mu) \neq \perp$ then $\forall 0 \leqslant j < i, k+j \notin \operatorname{dom}\mu$.
The domains as well are ommitted : "$\rho \in P$", "$\mu \in \mathcal{M}$", etc. are not explicit.

1 Expressions

1.1 Reading values

For local and global variables :

$$\frac{x \in \operatorname{dom}\rho \qquad \rho(x) \in \operatorname{dom}\mu}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{VAR}\ x \Rightarrow \mu, \texttt{nil}, \mu^{64}(\rho(x))}(\textsc{Var})$$

i.e. reading a variable returns its contents and changes nothing to the memory.

$$\frac{\chi \neq \texttt{nil}}{\rho, \mu, \chi, v \vdash^e \texttt{VAR}\ x \Rightarrow \mu, \chi, v}(\textsc{Var}^\chi)$$

For constant integers :

$$\frac{}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CST}\ n \Rightarrow \mu, \texttt{nil}, n}(\textsc{Cst})$$

$$\frac{\chi \neq \texttt{nil}}{\rho, \mu, \chi, v \vdash^e \texttt{CST}\ n \Rightarrow \mu, \chi, v}(\textsc{Cst}^\chi)$$

For strings :

$$\frac{s \text{ stored at } a \in \mathit{Addr}}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{STRING}\ s \Rightarrow \mu, \texttt{nil}, a}(\textsc{Str})$$

$$\frac{\chi \neq \texttt{nil}}{\rho, \mu, \chi, v \vdash^e \texttt{STRING}\ s \Rightarrow \mu, \chi, v}(\textsc{Cst}^\chi)$$

For arrays :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e i \Rightarrow \mu_i, \chi_i, v_i \\ \rho, \mu_i, \chi_i, i \vdash^e a \Rightarrow \mu_a, \texttt{nil}, v_a \\ v_a + v_i \times 8 \in \operatorname{dom} \mu_a\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_INDEX}, a, i) \Rightarrow \mu_a, \texttt{nil}, \mu_a^{64}(v_a + v_i \times 8)}(\textsc{Idx})$$

None of these are different from the original C—— semantics.

1.2 Unary operators without side-effects

Unary minus (same as C——) :

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \texttt{nil}, v_e}{\rho, \mu, \chi, v \vdash^e \texttt{OP1}(\texttt{M\_MINUS}, e) \Rightarrow \mu_e, \texttt{nil}, -v_e}(\textsc{Neg})$$

Unary bitwise negation (same as C——) :

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \texttt{nil}, v_e}{\rho, \mu, \chi, v \vdash^e \texttt{OP1}(\texttt{M\_NOT}, e) \Rightarrow \mu_e, \texttt{nil}, -v_e - 1}(\textsc{Not})$$

Indirection (added in ±C) :

$$\frac{x \in \operatorname{dom} \rho}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{OP1}(\texttt{M\_ADDR}, \texttt{VAR}\ x) \Rightarrow \mu, \texttt{nil}, \rho(x)}(\textsc{Var}^\&)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e i \Rightarrow \mu_i, \chi_i, v_i \\ \rho, \mu_i, \chi_i, v_i \vdash^e a \Rightarrow \mu_a, \texttt{nil}, v_a\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP1}(\texttt{M\_ADDR}, \texttt{OP2}(\texttt{S\_INDEX}, a, i)) \Rightarrow \mu_a, \texttt{nil}, v_a + v_i \times 8}(\textsc{Idx}^\&)$$

$$\frac{\rho, \mu, \chi, v \vdash^e a \Rightarrow \mu_a, \texttt{nil}, v_a}{\rho, \mu, \chi, v \vdash^e \texttt{OP1}(\texttt{M\_ADDR}, \texttt{OP1}(\texttt{M\_DEREF}, a)) \Rightarrow \mu_a, \texttt{nil}, v_a}(\textsc{Ptr}^\&)$$

Dereferencing (added in ±C) :

$$\frac{\rho, \mu, \chi, v \vdash^e a \Rightarrow \mu_a, \texttt{nil}, v_a \qquad v_a \in \operatorname{dom} \mu_a}{\rho, \mu, \chi, v \vdash^e \texttt{OP1}(\texttt{M\_DEREF}, a) \Rightarrow \mu_a, \texttt{nil}, \mu_a^{64}(v_a)}(\textsc{Ptr})$$

When the operand raises a non-`nil` flag :

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \qquad \chi_e \neq \texttt{nil}}{\rho, \mu, \chi, v \vdash^e \texttt{OP1}(op, e) \Rightarrow \mu_e, \chi_e, v_e}(\textsc{Op1}^\chi)$$

1.3 Binary operators

Multiplication (same as C——) :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_MUL}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, v_1 \times v_2}(\textsc{Mul})$$

Addition (same as C——) :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_ADD}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, v_1 + v_2}(\textsc{Add})$$

Subtraction (same as C——) :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_SUB}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, v_1 - v_2}(\textsc{Sub})$$

Division and remainder (same as C−−) :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \qquad v_2 \neq 0 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_DIV}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, v_1 \text{ div } v_2}(\textsc{Div})$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \qquad v_2 \neq 0 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_MOD}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, v_1 \text{ mod } v_2}(\textsc{Mod})$$

Shifts (added in ±C) :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_SHL}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, v_1 \times 2^{v_2}}(\textsc{Shl})$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_SHR}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, v_1 \text{ div } 2^{v_2}}(\textsc{Shr})$$

Let $\text{dec}_{64} : \{\perp, \top\}^{64} \to \mathbb{Z}_{64}$ the function

$$(b_0, \cdots, b_{63}) \mapsto \sum_{i=0}^{63} (1 \texttt{ if } b_i \texttt{ else } 0) \times 2^i$$

and $\text{bin}_{64} = {\text{dec}_{64}}^{-1}$.
We can now define bitwise operators as follows (added in ±C).

$$\frac{\begin{array}{cc}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 & \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1 \\ (b^2_0, \cdots, b^2_{63}) = \text{bin}_{64}(v_2) & (b^1_0, \cdots, b^1_{63}) = \text{bin}_{64}(v_1)\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_AND}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, \text{dec}_{64}(b^1_0 \wedge b^2_0, \cdots, b^1_{63} \wedge b^2_{63}), \mu_1}(\textsc{And})$$

$$\frac{\begin{array}{cc}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 & \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1 \\ (b^2_0, \cdots, b^2_{63}) = \text{bin}_{64}(v_2) & (b^1_0, \cdots, b^1_{63}) = \text{bin}_{64}(v_1)\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_OR}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, \text{dec}_{64}(b^1_0 \vee b^2_0, \cdots, b^1_{63} \vee b^2_{63}), \mu_1}(\textsc{Ior})$$

$$\frac{\begin{array}{cc}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 & \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1 \\ (b^2_0, \cdots, b^2_{63}) = \text{bin}_{64}(v_2) & (b^1_0, \cdots, b^1_{63}) = \text{bin}_{64}(v_1)\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(\texttt{S\_XOR}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, \text{dec}_{64}(b^1_0 \oplus b^2_0, \cdots, b^1_{63} \oplus b^2_{63}), \mu_1}(\textsc{Xor})$$

When one of the operands raises a non-`nil` flag :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \chi_1, v_1 \\ \chi_1 \neq \texttt{nil}\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP2}(op, e_1, e_2) \Rightarrow \mu_1, \chi_1, v_1}(\textsc{Op2}^\chi)$$

1.4 Comparisons

All are the same as in C−−.

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1 \\ v_1 = v_2\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CMP}(\texttt{C\_EQ}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, 1}(\textsc{Eq}^\top)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1 \\ v_1 < v_2\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CMP}(\texttt{C\_LT}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, 1}(\textsc{Lt}^\top)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1 \\ v_1 \leqslant v_2\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CMP}(\texttt{C\_LE}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, 1}(\textsc{Le}^\top)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1 \\ v_1 \neq v_2\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CMP}(\texttt{C\_EQ}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, 0}(\textsc{Eq}^\perp)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1 \\ v_1 \not< v_2\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CMP(C\_LT}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, 0}(\textsc{Lt}^\perp)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v \vdash^e e_1 \Rightarrow \mu_1, \texttt{nil}, v_1 \\ v_1 \not\leq v_2\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CMP(C\_LE}, e_1, e_2) \Rightarrow \mu_1, \texttt{nil}, 0}(\textsc{Le}^\perp)$$

For optimisation purposes mostly, the comparison operators `C_NE`, `C_GT`, `C_GE` may be introduced by the compiler (not by the parser, however).
They are defined as

$$\frac{\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_NOT}, \texttt{CMP(C\_EQ}, e_1, e_2)) \Rightarrow \mu', \chi', v'}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CMP(C\_NE}, e_1, e_2) \Rightarrow \mu', \chi', v'}(\textsc{Ne})$$

$$\frac{\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_NOT}, \texttt{CMP(C\_LE}, e_1, e_2)) \Rightarrow \mu', \chi', v'}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CMP(C\_GT}, e_1, e_2) \Rightarrow \mu', \chi', v'}(\textsc{Gt})$$

$$\frac{\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_NOT}, \texttt{CMP(C\_LT}, e_1, e_2)) \Rightarrow \mu', \chi', v'}{\rho, \mu, \texttt{nil}, v \vdash^e \texttt{CMP(C\_GE}, e_1, e_2) \Rightarrow \mu', \chi', v'}(\textsc{Ge})$$

When one of the operands raises a non-`nil` flag :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \chi_1, v_1 \\ \chi_1 \neq \texttt{nil}\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{CMP}(op, e_1, e_2) \Rightarrow \mu_1, \chi_1, v_1}(\textsc{Cmp}^\chi)$$

1.5 Assignments

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \\ \rho, \mu_e, \chi_e, v_e \vdash^e \texttt{OP1(M\_ADDR}, a) \Rightarrow \mu_a, \texttt{nil}, v_a\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{SET}(a, e) \Rightarrow \mu_a^{64}[v_a \mapsto v_e], \texttt{nil}, v_e}(\textsc{Ptr}^\leftarrow)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \\ \rho, \mu_e, \chi_e, v_e \vdash^e \texttt{OP1(M\_ADDR}, a) \Rightarrow \mu_a, \chi_a, v_a \qquad \chi_a \neq \texttt{nil}\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{SET}(a, e) \Rightarrow \mu_a, \chi_a, v_a}(\textsc{Ptr}^{\leftarrow\chi})$$

1.6 Increments

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_ADDR}, a) \Rightarrow \mu_a, \texttt{nil}, v_a \\ k = \mu_a^{64}(v_a)\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_POST\_INC}, a) \Rightarrow \mu_a^{64}[v_a \mapsto k+1], \texttt{nil}, k}(\textsc{Post}^\uparrow)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_ADDR}, a) \Rightarrow \mu_a, \texttt{nil}, v_a \\ k = \mu_a^{64}(v_a)\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_POST\_DEC}, a) \Rightarrow \mu_a^{64}[v_a \mapsto k-1], \texttt{nil}, k}(\textsc{Post}^\downarrow)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_ADDR}, a) \Rightarrow \mu_a, \texttt{nil}, v_a \\ k = \mu_a^{64}(v_a)\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_PRE\_INC}, a) \Rightarrow \mu_a^{64}[v_a \mapsto k+1], \texttt{nil}, k+1}(\textsc{Pre}^\uparrow)$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_ADDR}, a) \Rightarrow \mu_a, \texttt{nil}, v_a \\ k = \mu_a^{64}(v_a)\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OP1(M\_PRE\_DEC}, a) \Rightarrow \mu_a^{64}[v_a \mapsto k-1], \texttt{nil}, k-1}(\textsc{Pre}^\downarrow)$$

1.7 Extended assignments

Let $op \in \texttt{bin\_op} \setminus \{\texttt{S\_INDEX}\}$.

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \\ \rho, \mu_e, \chi_e, v_e \vdash^e \texttt{OP1(M\_ADDR}, a) \Rightarrow \mu_a, \chi_a, v_a \\ v_a \in \operatorname{dom} \mu_a \qquad \mu_a^{64}(v_a) = u \\ \rho, \mu_a, \chi_a, v_a \vdash^e \texttt{OP2}(op, \texttt{CST } v_a, \texttt{CST } v_e) \Rightarrow \mu', \texttt{nil}, w\end{array}}{\rho, \mu, \chi, v \vdash^e \texttt{OPSET}(op, a, e) \Rightarrow \mu'[v_a \mapsto w], \texttt{nil}, w}(\textsc{Opset})$$

$$\frac{\begin{array}{c}\rho,\mu,\chi,v\vdash^e e\Rightarrow\mu_e,\chi_e,v_e\\ \rho,\mu_e,\chi_e,v_e\vdash^e \texttt{OP1}(\texttt{M\_ADDR},a)\Rightarrow\mu_a,\chi_a,v_a\\ \chi_a\neq\texttt{nil}\end{array}}{\rho,\mu,\chi,v\vdash^e \texttt{OPSET}(op,a,e)\Rightarrow\mu_a,\chi_a,v_a}(\textsc{Opset}^\chi)$$

1.8 Ternary operator

$$\frac{\begin{array}{c}\rho,\mu,\chi,v\vdash^e e\Rightarrow\mu_e,\texttt{nil},v_e \qquad v_e=0\\ \rho,\mu_e,\texttt{nil},v_e\vdash^e e_\bot\Rightarrow\mu_\bot,\chi_\bot,v_\bot\end{array}}{\rho,\mu,\chi,v\vdash^e \texttt{EIF}(e,e_\top,e_\bot)\Rightarrow\mu_\bot,\chi_\bot,v_\bot}(\textsc{Tern}^\bot)$$

$$\frac{\begin{array}{c}\rho,\mu,\chi,v\vdash^e e\Rightarrow\mu_e,\texttt{nil},v_e \qquad v_e\neq0\\ \rho,\mu_e,\texttt{nil},v_e\vdash^e e_\top\Rightarrow\mu_\top,\chi_\top,v_\top\end{array}}{\rho,\mu,\chi,v\vdash^e \texttt{EIF}(e,e_\top,e_\bot)\Rightarrow\mu_\top,\chi_\top,v_\top}(\textsc{Tern}^\top)$$

$$\frac{\rho,\mu,\chi,v\vdash^e e\Rightarrow\mu_e,\chi_e,v_e \qquad \chi_e\neq\texttt{nil}}{\rho,\mu,\chi,v\vdash^e \texttt{EIF}(e,e_\top,e_\bot)\Rightarrow\mu_e,\chi_e,v_e}(\textsc{Tern}^\chi)$$

1.9 Sequence

$$\frac{\begin{array}{c}\rho,\mu_0,\chi_0,v_0\vdash^e e_1\Rightarrow\mu_1,\chi_1,v_1\\ \cdots\\ \rho,\mu_{n-1},\chi_{n-1},v_{n-1}\vdash^e e_n\Rightarrow\mu_n,\chi_n,v_n\end{array}}{\rho,\mu_0,\chi_0,v_0\vdash^e \texttt{ESEQ}\ [e_1;\cdots;e_n]\Rightarrow\mu_n,\chi_n,v_n}(\textsc{Seq}^n)$$

1.10 Function call

Works for both a toplevel function and a function pointer :

$$\frac{\begin{array}{c}\rho,\mu_{n+1},\chi_{n+1},v_{n+1}\vdash^e e_n\Rightarrow\mu_n,\chi_n,v_n\\ \cdots\\ \rho,\mu_2,\chi_2,v_2\vdash^e e_1\Rightarrow\mu_1,\texttt{nil},v_1\\ f\in\operatorname{dom}\rho \qquad \rho(f)\in\operatorname{dom}\mathrm{fun}\\ \rho_g,\mu_1,\texttt{nil},0\vdash^e \mathrm{fun}(\rho(f))(v_1,\cdots,v_n)\Rightarrow\rho_f,\mu_f,\chi_f,v_f\end{array}}{\rho,\mu_n,\chi_n,v_n\vdash^e \texttt{CALL}(f,[e_1;\cdots;e_n])\Rightarrow\mu_f,\chi_f,v_f}(\textsc{Call}^n)$$

$$\frac{\begin{array}{c}\rho,\mu_{n+1},\chi_{n+1},v_{n+1}\vdash^e e_n\Rightarrow\mu_n,\chi_n,v_n\\ \cdots\\ \rho,\mu_2,\chi_2,v_2\vdash^e e_1\Rightarrow\mu_1,\chi_1,v_1\\ \chi_1\neq\texttt{nil}\end{array}}{\rho,\mu_{n+1},\chi_{n+1},v_{n+1}\vdash^e \texttt{CALL}(f,[e_1;\cdots;e_n])\Rightarrow\mu_1,\chi_1,v_1}(\textsc{Call}^\chi)$$

2 Code

2.1 Expressions

An expression as statement is simply executed. If a non-`nil` flag is raised, it will be skipped anyway.

$$\frac{\rho,\mu,\chi,v\vdash^e e\Rightarrow\mu_e,\chi_e,v_e}{\rho,\mu,\chi,v\vdash^c \texttt{CEXPR}\ e\Rightarrow\rho,\mu_e,\chi_e,v_e}(\textsc{Expr})$$

2.2 Conditional branching

If only `nil` is raised after the evaluation of the condition, one of the two branches is executed.

$$\frac{\begin{array}{c}\rho,\mu,\chi,v\vdash^e e\Rightarrow\mu_e,\texttt{nil},v_e \qquad v_e=0\\ \rho,\mu_e,\texttt{nil},v_e\vdash^c c_\bot\Rightarrow\rho_\bot,\mu_\bot,\chi_\bot,v_\bot\end{array}}{\rho,\mu,\chi,v\vdash^c \texttt{CIF}(e,c_\top,c_\bot)\Rightarrow\rho,\mu_\bot,\chi_\bot,v_\bot}(\textsc{If}^\bot)$$

$$\frac{\begin{array}{c}\rho,\mu,\chi,v\vdash^e e\Rightarrow\mu_e,\texttt{nil},v_e \qquad v_e\neq0\\ \rho,\mu_e,\texttt{nil},v_e\vdash^c c_\top\Rightarrow\rho_\top,\mu_\top,\chi_\top,v_\top\end{array}}{\rho,\mu,\chi,v\vdash^c \texttt{CIF}(e,c_\top,c_\bot)\Rightarrow\rho,\mu_\top,\chi_\top,v_\top}(\textsc{If}^\top)$$

Note that the branch is allowed to modify the memory and raise flags, but not change the environment : ρ is preserved.

For all other flags, neither of the branches is executed.

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \qquad \chi_e \neq \texttt{nil}}{\rho, \mu, \chi, v \vdash^c \texttt{CIF}(e, c_\top, c_\bot) \Rightarrow \rho, \mu_e, \chi_e, v_e} (\textsc{If}^\chi)$$

2.3 Blocks

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^c c \Rightarrow \rho', \mu', \chi', v' \\ \rho', \mu', \chi', v' \vdash^c \texttt{CBLOCK}\ S \Rightarrow \rho'', \mu'', \chi'', v''\end{array}}{\rho, \mu, \chi, v \vdash^c \texttt{CBLOCK}(c :: S) \Rightarrow \rho, \mu'', \chi'', v''} (\textsc{Block}^1)$$

$$\frac{}{\rho, \mu, \chi, v \vdash^c \texttt{CBLOCK}\ [] \Rightarrow \rho, \mu, \chi, v} (\textsc{Block}^0)$$

Again for blocks, the memory may be changed and flags may be raised, but the environment is preserved.

2.4 Loops

A loop with a false condition stops :

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \texttt{nil}, v_e \qquad v_e = 0}{\rho, \mu, \chi, v \vdash^c \texttt{CWHILE}(e, c, f, \texttt{true}) \Rightarrow \rho, \mu', \texttt{nil}, v} (\textsc{While}^{\bot,\texttt{true}})$$

Except in the case of a `do-while` :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^c c \Rightarrow \rho_c, \mu_c, \chi_c, v_c \\ \rho, \mu_c, \chi_c, v_c \vdash^e f \Rightarrow \mu_f, \texttt{nil}, v_f \\ \rho, \mu_f, \texttt{nil}, v_f \vdash^c \texttt{CWHILE}(e, c, f, \texttt{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w\end{array}}{\rho, \mu, \chi, v \vdash^c \texttt{CWHILE}(e, c, f, \texttt{false}) \Rightarrow \rho, \mu_w, \chi_w, v_w} (\textsc{While}^{\texttt{false}})$$

A loop continues normally if its condition is nonzero :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \texttt{nil}, v_e \qquad v_e \neq 0 \\ \rho, \mu_e, \texttt{nil}, v_e \vdash^c c \Rightarrow \rho_c, \mu_c, \chi_c, v_c \qquad \chi_c \notin \{\texttt{brk}, \texttt{cnt}\} \\ \rho, \mu_c, \chi_c, v_c \vdash^e f \Rightarrow \mu_f, \chi_f, v_f \\ \rho, \mu_f, \chi_f, v_f \vdash^c \texttt{CWHILE}(e, c, f, \texttt{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w\end{array}}{\rho, \mu, \chi, v \vdash^c \texttt{CWHILE}(e, c, f, \texttt{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w} (\textsc{While}^{\top,\texttt{true}})$$

A flag skips the loop :

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \qquad \chi_e \neq \texttt{nil}}{\rho, \mu, \chi, v \vdash^c \texttt{CWHILE}(e, c, f, \texttt{true}) \Rightarrow \rho, \mu_e, \chi_e, v_e} (\textsc{While}^{\chi,\texttt{true}})$$

$$\frac{\chi \neq \texttt{nil}}{\rho, \mu, \chi, v \vdash^c \texttt{CWHILE}(e, c, f, \texttt{false}) \Rightarrow \rho, \mu, \chi, v} (\textsc{While}^{\chi,\texttt{false}})$$

`cnt` executes the finally clause before continuing as normal :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \texttt{nil}, v_e \qquad v_e \neq 0 \\ \rho, \mu_e, \texttt{nil}, v_e \vdash^c c \Rightarrow \rho_c, \mu_c, \texttt{cnt}, v_c \\ \rho, \mu_c, \texttt{nil}, v_c \vdash^e f \Rightarrow \mu_f, \chi_f, v_f \\ \rho, \mu_f, \chi_f, v_f \vdash^c \texttt{CWHILE}(e, c, f, \texttt{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w\end{array}}{\rho, \mu, \chi, v \vdash^c \texttt{CWHILE}(e, c, f, \texttt{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w} (\textsc{While}^{\texttt{cnt},\texttt{true}})$$

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^c c \Rightarrow \rho_c, \mu_c, \texttt{cnt}, v_c \\ \rho, \mu_c, \texttt{nil}, v_c \vdash^e f \Rightarrow \mu_f, \chi_f, v_f \\ \rho, \mu_f, \chi_f, v_f \vdash^c \texttt{CWHILE}(e, c, f, \texttt{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w\end{array}}{\rho, \mu, \chi, v \vdash^c \texttt{CWHILE}(e, c, f, \texttt{false}) \Rightarrow \rho, \mu_w, \chi_w, v_w} (\textsc{While}^{\texttt{cnt},\texttt{false}})$$

`brk` interrupts the loop but is not retransmitted :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \texttt{nil}, v_e \qquad v_e \neq 0 \\ \rho, \mu_e, \texttt{nil}, v_e \vdash^c c \Rightarrow \rho_c, \mu_c, \texttt{brk}, v_c\end{array}}{\rho, \mu, \chi, v \vdash^c \texttt{CWHILE}(e, c, f, \texttt{true}) \Rightarrow \rho, \mu_c, \texttt{nil}, v_c} (\textsc{While}^{\texttt{brk},\texttt{true}})$$

$$\frac{\rho, \mu, \chi, v \vdash^c c \Rightarrow \rho_c, \mu_c, \texttt{brk}, v_c}{\rho, \mu, \chi, v \vdash^c \texttt{CWHILE}(e, c, f, \texttt{false}) \Rightarrow \rho, \mu_c, \texttt{nil}, v_c} (\textsc{While}^{\texttt{brk},\texttt{false}})$$

2.5 Control flow

$$\frac{}{\rho, \mu, \mathtt{nil}, v \vdash^c \mathtt{CBREAK} \Rightarrow \rho, \mu, \mathtt{brk}, 0}(\textsc{Break})$$

$$\frac{\chi \neq \mathtt{nil}}{\rho, \mu, \chi, v \vdash^c \mathtt{CBREAK} \Rightarrow \rho, \mu, \chi, v}(\textsc{Break}^\chi)$$

$$\frac{}{\rho, \mu, \mathtt{nil}, v \vdash^c \mathtt{CCONTINUE} \Rightarrow \rho, \mu, \mathtt{cnt}, 0}(\textsc{Continue})$$

$$\frac{\chi \neq \mathtt{nil}}{\rho, \mu, \chi, v \vdash^c \mathtt{CCONTINUE} \Rightarrow \rho, \mu, \chi, v}(\textsc{Continue}^\chi)$$

$$\frac{}{\rho, \mu, \mathtt{nil}, v \vdash^c \mathtt{CRETURN\ None} \Rightarrow \rho, \mu, \mathtt{ret}, 0}(\textsc{Return}^{\mathtt{None}})$$

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathtt{nil}, v_e}{\rho, \mu, \mathtt{nil}, v \vdash^c \mathtt{CRETURN(Some}\ e\mathtt{)} \Rightarrow \rho, \mu_e, \mathtt{ret}, v_e}(\textsc{Return}^{\mathtt{Some}})$$

$$\frac{\chi \neq \mathtt{nil}}{\rho, \mu, \chi, v \vdash^c \mathtt{CRETURN\ None} \Rightarrow \rho, \mu, \chi, v}(\textsc{Return}^{\mathtt{None}\chi})$$

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \qquad \chi_e \neq \mathtt{nil}}{\rho, \mu, \mathtt{nil}, v \vdash^c \mathtt{CRETURN(Some}\ e\mathtt{)} \Rightarrow \rho, \mu_e, \chi_e, v_e}(\textsc{Return}^{\mathtt{Some}\chi})$$

2.6 Local variable declarations

When there are none :

$$\frac{}{\rho, \mu, \chi, v \vdash^c \mathtt{CLOCAL}\ [] \Rightarrow \rho, \mu, \chi, v}(\textsc{Local}^0)$$

When an error occurs :

$$\frac{\chi \neq \mathtt{nil}}{\rho, \mu, \chi, v \vdash^c \mathtt{CLOCAL}\ d \Rightarrow \rho, \mu, \chi, v}(\textsc{Local}^\chi)$$

Otherwise :

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathtt{nil}, v_e \\ k \in alloc^8(\mu_e) \qquad \rho' = \rho[w \mapsto k] \qquad \mu' = \mu_e[k \mapsto v_e] \\ \rho', \mu', \mathtt{nil}, v_e \vdash^c \mathtt{CLOCAL}\ S \Rightarrow \rho_s, \mu_s, \chi_s, v_s\end{array}}{\rho, \mu, \chi, v \vdash^c \mathtt{CLOCAL}((w, e) :: S) \Rightarrow \rho_s, \mu_s, \chi_s, v_s}(\textsc{Local}^1)$$

2.7 Throw

If a flag is already raised, skip the `CTHROW` :

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \qquad \chi_e \neq \mathtt{nil}}{\rho, \mu, \chi, v \vdash^c \mathtt{CTHROW}(s, e) \Rightarrow \rho, \mu_e, \chi_e, v_e}(\textsc{Throw}^\chi)$$

Otherwise raise the new exception $s \in S$:

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathtt{nil}, v_e}{\rho, \mu, \chi, v \vdash^c \mathtt{CTHROW}(s, e) \Rightarrow \rho, \mu_e, s, v_e}(\textsc{Throw})$$

2.8 Switch

$$\frac{\begin{array}{c}\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \\ \rho, \mu_e, \chi_e, v_e \vdash^c \mathtt{CBLOCK}(L(v_e)) \Rightarrow \rho, \mu_l, \chi_l, v_l\end{array}}{\rho, \mu, \chi, v \vdash^c \mathtt{CSWITCH}(e, L, c) \Rightarrow \rho, \mu_l, \chi_l, v_l}(\textsc{Switch})$$

Where for $L = [(j_1, l_1); \cdots; (j_n, l_n)]$, $L(v_e)$ is defined as follows :
Let $I_i = \{j_1, \cdots, j_i\}$ for $1 \leqslant i \leqslant n$, $I_{n+1} = \mathbb{Z}_{64}$.
$\tilde{j} \triangleq \min_{1 \leqslant i \leqslant n+1}\{i \mid v_e \in I_i\}$, finally $L(v_e) \triangleq [l_{\tilde{j}}; \cdots; l_n; c]$.

2.9 Try

Skip the block when a flag is already raised :

$$\frac{\chi \neq \texttt{nil}}{\rho, \mu, \chi, v \vdash^c \texttt{CTRY}(c, L, f) \Rightarrow \rho, \mu, \chi, v} (\textsc{Try}^{\chi})$$

For $L = [(e_1, x_1, c_1); \cdots; (e_n, x_n, c_n)]$, let $E = \{e_i | 1 \leqslant i \leqslant n\} \subset S$.
When no exception is raised :

$$\frac{\begin{array}{c}\rho, \mu, \texttt{nil}, v \vdash^c c \Rightarrow \rho_c, \mu_c, \texttt{nil}, v_c \\ \rho, \mu_c, \texttt{nil}, v_c \vdash^c f \Rightarrow \rho_f, \mu_f, \chi_f, v_f\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^c \texttt{CTRY}(c, L, f) \Rightarrow \rho, \mu_f, \chi_f, v_f} (\textsc{Try}^{\texttt{nil}})$$

When an exception is raised that is not caught by the current handler :

$$\frac{\begin{array}{c}\rho, \mu, \texttt{nil}, v \vdash^c c \Rightarrow \rho_c, \mu_c, s_c, v_c \\ s_c \notin E \qquad \_ \notin E \\ \rho, \mu, \texttt{nil}, v_c \vdash^c f \Rightarrow \rho_f, \mu_f, \texttt{nil}, v_f\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^c \Rightarrow \texttt{CTRY}(c, L, f) \Rightarrow \rho, \mu_f, s_c, v_c} (\textsc{Try}^{\texttt{nil}'})$$

$$\frac{\begin{array}{c}\rho, \mu, \texttt{nil}, v \vdash^c c \Rightarrow \rho_c, \mu_c, s_c, v_c \\ s_c \notin E \qquad \_ \notin E \\ \rho, \mu, \texttt{nil}, v_c \vdash^c f \Rightarrow \rho_f, \mu_f, \chi_f, v_f \qquad \chi_f \neq \texttt{nil}\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^c \Rightarrow \texttt{CTRY}(c, L, f) \Rightarrow \rho, \mu_f, \chi_f, v_f} (\textsc{Try}^{\chi'})$$

When the handler is able to catch the exception :

$$\frac{\begin{array}{c}\rho, \mu, \texttt{nil}, v \vdash^c c \Rightarrow \rho_c, \mu_c, s_c, v_c \\ s_c = e_{i_0} \qquad x_{i_0} \neq \_ \\ k \in \mathit{alloc}^8(\mu) \qquad \rho[x_{i_0} \mapsto k], \mu_c[k \mapsto v_c], \texttt{nil}, v_c \vdash^c c_{i_0} \Rightarrow \rho_0, \mu_0, \chi_0, v_0 \\ \rho, \mu_0, \chi_0, v_0 \vdash^c f \Rightarrow \rho_f, \mu_f, \chi_f, v_f\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^c \texttt{CTRY}(c, L, f) \Rightarrow \rho, \mu_f, \chi_f, v_f} (\textsc{Try}^{s})$$

$$\frac{\begin{array}{c}\rho, \mu, \texttt{nil}, v \vdash^c c \Rightarrow \rho_c, \mu_c, s_c, v_c \\ s_c = e_{i_0} \qquad x_{i_0} = \_ \\ \rho, \mu_c, \texttt{nil}, v_c \vdash^c c_{i_0} \Rightarrow \rho_0, \mu_0, \chi_0, v_0 \\ \rho, \mu_0, \chi_0, v_0 \vdash^c f \Rightarrow \rho_f, \mu_f, \chi_f, v_f\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^c \texttt{CTRY}(c, L, f) \Rightarrow \rho, \mu_f, \chi_f, v_f} (\textsc{Try}^{s})$$

$$\frac{\begin{array}{c}\rho, \mu, \texttt{nil}, v \vdash^c c \Rightarrow \rho_c, \mu_c, s_c, v_c \\ s_c \notin E \qquad \_ = e_{i_0} \\ \rho, \mu_c, \texttt{nil}, v_c \vdash^c c_{i_0} \Rightarrow \rho_0, \mu_0, \chi_0, v_0 \\ \rho, \mu_0, \chi_0, v_0 \vdash^c f \Rightarrow \rho_f, \mu_f, \chi_f, v_f\end{array}}{\rho, \mu, \texttt{nil}, v \vdash^c \texttt{CTRY}(c, L, f) \Rightarrow \rho, \mu_f, \chi_f, v_f} (\textsc{Try}^{s})$$

3 Declarations

3.1 Global variables

$$\frac{k \notin \operatorname{dom} \rho \qquad k \in \mathit{alloc}^8(\mu)}{\rho, \mu, \text{fun} \vdash^d \texttt{CDECL}(x, \texttt{None}) \Rightarrow \rho[x \mapsto k], \mu[k \mapsto 0], \text{fun}} (\textsc{Decl}^{\texttt{None}})$$

$$\frac{x \notin \operatorname{dom} \rho \qquad k \in \mathit{alloc}^8(\mu)}{\rho, \mu, \text{fun} \vdash^d \texttt{CDECL}(x, \texttt{Some}(\texttt{CST}\ c)) \Rightarrow \rho[x \mapsto k], \mu[k \mapsto c], \text{fun}} (\textsc{Decl}^{\texttt{CST}})$$

$$\frac{s \text{ stored at } a \in \mathit{Addr} \qquad x \notin \operatorname{dom} \rho}{\rho, \mu, \text{fun} \vdash^d \texttt{CDECL}(x, \texttt{Some}(\texttt{STRING}\ s)) \Rightarrow \rho[x \mapsto a], \mu, \text{fun}} (\textsc{Decl}^{\texttt{STRING}})$$

3.2 Functions

$$\frac{f \notin \operatorname{dom} \rho \qquad \phi_A(b) \text{ stored at } k}{\rho, \mu, \text{fun} \vdash^d \texttt{CFUN}(f, A, b) \Rightarrow \rho[f \mapsto k], \mu, \text{fun}[k \mapsto \phi_A(b)]} (\textsc{Fun})$$

Let $[a_1; \cdots; a_n] = A$, then $\phi_A(b) : \mathbb{Z}_{64}^n \to \texttt{code}$ is defined as

$$\phi_A(b)(v_1, \cdots, v_n) = \texttt{CBLOCK}(\texttt{CLOCAL}[(a_1, \texttt{Some (CST } v_1)); \cdots; (a_n, \texttt{Some (CST } v_n))] :: b)$$

4 Program

Finally, here's how the whole program executes :

$$\frac{\begin{array}{c}\rho_0, \mu_0, \mathrm{fun}_0 \vdash^d d_1 \Rightarrow \rho_1, \mu_1, \mathrm{fun}_1 \\ \cdots \\ \rho_{n-1}, \mu_{n-1}, \mathrm{fun}_{n-1} \vdash^d d_n \Rightarrow \rho_n, \mu_n, \mathrm{fun}_n \\ \rho_n, \mu_n, \texttt{nil}, 0 \vdash^e \texttt{CALL}(\texttt{main}, [\texttt{argc}; \texttt{argv}]) \Rightarrow \mu', \chi', v'\end{array}}{\rho_0, \mu_0, \mathrm{fun}_0 \vdash^\pi [d_1; \cdots; d_n] \Rightarrow \mu', \chi', v'} \text{(PROG)}$$

ρ_0, μ_0 and fun_0 initially contain some predefined globals and constants such as `NULL`, `stdout`, `EOF`, `true`, `BYTE`, `QSIZE`, ..., as well as standard library functions (`malloc`, `atol`, `rand`, `sleep`, `qsort`, ...)