

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

VILLANI Neven, ENS Paris-Saclay
PROGRAMMATION 1 – COMPILATEUR COCASS

January 7, 2021

Retranscribed and extended from <http://www.lsv.fr/~goubault/CoursProgrammation/minic.html>
and http://www.lsv.fr/~goubault/CoursProgrammation/prog1_seml.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 $(\rho : \mathcal{S} \rightarrow \mathbb{Z}_{64}) \in \mathcal{P}$ the environment, where \mathcal{S} is the set of names of variables and functions, $(\mu : \mathbb{Z}_{64} \rightarrow \mathbb{Z}_8) \in \mathcal{M}$ 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 $\mathcal{E} \triangleq \mathcal{S} \sqcup \{\mathbf{brk}, \mathbf{ret}, \mathbf{cnt}, \mathbf{nil}\}$: 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 $\mathbf{fun} : \mathbb{Z}_{64} \rightarrow \mathbf{code}$, a wrapper around $\pm C$ functions: $\mathbf{fun}(a)(p_1, \dots, p_n) = c$ updates the environment with p_1, \dots, 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 $\mu[x \mapsto v] : \begin{cases} x \mapsto v \\ y \mapsto \mu(y) & y \in \text{dom } \mu \setminus \{x\} \end{cases}$

However we will usually use $\mu^{64}[x \mapsto v] \triangleq \mu[x+k \mapsto v_k \mid 0 \leq k < 8, v = \sum_{k=0}^7 2^{8k} v_k]$, i.e. the memory is written 8 bytes at a time.

A similar notation is used for ρ, ρ_g and \mathbf{fun} .

$\mathit{alloc}^i : \mathcal{M} \rightarrow \mathcal{P}(\mathbb{Z}_{64})$ is such that if $k \in \mathit{alloc}^i(\mu) \neq \perp$ then $\forall 0 \leq j < i, k+j \notin \text{dom } \mu$.

The domains as well are omitted : " $\rho \in \mathcal{P}$ ", " $\mu \in \mathcal{M}$ ", etc. are not explicit.

1 Expressions

1.1 Reading values

For local and global variables:

$$\frac{x \in \text{dom } \rho \quad \rho(x) \in \text{dom } \mu}{\rho, \mu, \mathbf{nil}, v \vdash^e \mathbf{VAR } x \Rightarrow \mu, \mathbf{nil}, \mu^{64}(\rho(x))} (\mathbf{VAR})$$

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

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

For constant integers:

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

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

For strings:

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

$$\frac{\chi \neq \mathbf{nil}}{\rho, \mu, \chi, v \vdash^e \text{STRING } s \Rightarrow \mu, \chi, v} (\text{CST}^\chi)$$

For arrays:

$$\frac{\begin{array}{c} \rho, \mu, \chi, v \vdash^e i \Rightarrow \mu_i, \chi_i, v_i \\ \rho, \mu_i, \chi_i, i \vdash^e a \Rightarrow \mu_a, \mathbf{nil}, v_a \\ v_a + v_i \times 8 \in \text{dom } \mu_a \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\text{S_INDEX}, a, i) \Rightarrow \mu_a, \mathbf{nil}, \mu_a^{64}(v_a + v_i \times 8)} (\text{IDX})$$

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

1.2 Unary operators without side-effects

Unary minus (same as C—):

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathbf{nil}, v_e}{\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_MINUS}, e) \Rightarrow \mu_e, \mathbf{nil}, -v_e} (\text{NEG})$$

Unary bitwise negation (same as C—):

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathbf{nil}, v_e}{\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_NOT}, e) \Rightarrow \mu_e, \mathbf{nil}, -v_e - 1} (\text{NOT})$$

Indirection (added in $\pm\text{C}$):

$$\frac{x \in \text{dom } \rho}{\rho, \mu, \mathbf{nil}, v \vdash^e \text{OP1}(\text{M_ADDR}, \text{VAR } x) \Rightarrow \mu, \mathbf{nil}, \rho(x)} (\text{VAR}^\&)$$

$$\frac{\begin{array}{c} \rho, \mu, \chi, v \vdash^e i \Rightarrow \mu_i, \chi_i, v_i \\ \rho, \mu_i, \chi_i, v_i \vdash^e a \Rightarrow \mu_a, \mathbf{nil}, v_a \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_ADDR}, \text{OP2}(\text{S_INDEX}, a, i)) \Rightarrow \mu_a, \mathbf{nil}, v_a + v_i \times 8} (\text{IDX}^\&)$$

$$\frac{\rho, \mu, \chi, v \vdash^e a \Rightarrow \mu_a, \mathbf{nil}, v_a}{\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_ADDR}, \text{OP1}(\text{M_DEREF}, a)) \Rightarrow \mu_a, \mathbf{nil}, v_a} (\text{PTR}^\&)$$

Dereferencing (added in $\pm\text{C}$):

$$\frac{\rho, \mu, \chi, v \vdash^e a \Rightarrow \mu_a, \mathbf{nil}, v_a \quad v_a \in \text{dom } \mu_a}{\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_DEREF}, a) \Rightarrow \mu_a, \mathbf{nil}, \mu_a^{64}(v_a)} (\text{PTR})$$

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

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \quad \chi_e \neq \mathbf{nil}}{\rho, \mu, \chi, v \vdash^e \text{OP1}(op, e) \Rightarrow \mu_e, \chi_e, v_e} (\text{OP1}^\chi)$$

1.3 Binary operators

Multiplication (same as C—):

$$\frac{\begin{array}{c} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\text{S_MUL}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, v_1 \times v_2} (\text{MUL})$$

Addition (same as C—):

$$\frac{\begin{array}{c} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\text{S_ADD}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, v_1 + v_2} (\text{ADD})$$

Subtraction (same as C—):

$$\frac{\begin{array}{c} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\text{S_SUB}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, v_1 - v_2} (\text{SUB})$$

Division and remainder (same as C—):

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \quad v_2 \neq 0 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\mathbf{S_DIV}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, v_1 \text{ div } v_2} (\text{DIV})$$

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \quad v_2 \neq 0 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\mathbf{S_MOD}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, v_1 \text{ mod } v_2} (\text{MOD})$$

Shifts (added in $\pm C$):

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\mathbf{S_SHL}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, v_1 \times 2^{v_2}} (\text{SHL})$$

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\mathbf{S_SHR}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, v_1 \text{ div } 2^{v_2}} (\text{SHR})$$

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

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

and $\text{bin}_{64} = \text{dec}_{64}^{-1}$.

We can now define bitwise operators as follows (added in $\pm C$).

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \quad \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\ (b_0^2, \dots, b_{63}^2) = \text{bin}_{64}(v_2) \quad (b_0^1, \dots, b_{63}^1) = \text{bin}_{64}(v_1) \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\mathbf{S_AND}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, \text{dec}_{64}(b_0^1 \wedge b_0^2, \dots, b_{63}^1 \wedge b_{63}^2), \mu_1} (\text{AND})$$

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \quad \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\ (b_0^2, \dots, b_{63}^2) = \text{bin}_{64}(v_2) \quad (b_0^1, \dots, b_{63}^1) = \text{bin}_{64}(v_1) \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\mathbf{S_OR}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, \text{dec}_{64}(b_0^1 \vee b_0^2, \dots, b_{63}^1 \vee b_{63}^2), \mu_1} (\text{IOR})$$

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \quad \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\ (b_0^2, \dots, b_{63}^2) = \text{bin}_{64}(v_2) \quad (b_0^1, \dots, b_{63}^1) = \text{bin}_{64}(v_1) \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(\mathbf{S_XOR}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, \text{dec}_{64}(b_0^1 \oplus b_0^2, \dots, b_{63}^1 \oplus b_{63}^2), \mu_1} (\text{XOR})$$

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

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \chi_1, v_1 \\ \chi_1 \neq \mathbf{nil} \end{array}}{\rho, \mu, \chi, v \vdash^e \text{OP2}(op, e_1, e_2) \Rightarrow \mu_1, \chi_1, v_1} (\text{OP2}^X)$$

1.4 Comparisons

All are the same as in C—.

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\ v_1 = v_2 \end{array}}{\rho, \mu, \mathbf{nil}, v \vdash^e \text{CMP}(\mathbf{C_EQ}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, 1} (\text{EQ}^\top)$$

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\ v_1 < v_2 \end{array}}{\rho, \mu, \mathbf{nil}, v \vdash^e \text{CMP}(\mathbf{C_LT}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, 1} (\text{LT}^\top)$$

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\ v_1 \leq v_2 \end{array}}{\rho, \mu, \mathbf{nil}, v \vdash^e \text{CMP}(\mathbf{C_LE}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, 1} (\text{LE}^\top)$$

$$\frac{\begin{array}{l} \rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\ \rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\ v_1 \neq v_2 \end{array}}{\rho, \mu, \mathbf{nil}, v \vdash^e \text{CMP}(\mathbf{C_EQ}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, 0} (\text{EQ}^\perp)$$

$$\begin{array}{c}
\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\
\rho, \mu_2, \chi_2, v \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\
\hline
v_1 \not\leq v_2 \\
\rho, \mu, \mathbf{nil}, v \vdash^e \text{CMP}(\text{C_LT}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, 0 \quad (\text{LT}^\perp) \\
\\
\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\
\rho, \mu_2, \chi_2, v \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\
\hline
v_1 \not\leq v_2 \\
\rho, \mu, \mathbf{nil}, v \vdash^e \text{CMP}(\text{C_LE}, e_1, e_2) \Rightarrow \mu_1, \mathbf{nil}, 0 \quad (\text{LE}^\perp)
\end{array}$$

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

$$\begin{array}{c}
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_NOT}, \text{CMP}(\text{C_EQ}, e_1, e_2)) \Rightarrow \mu', \chi', v' \\
\rho, \mu, \mathbf{nil}, v \vdash^e \text{CMP}(\text{C_NE}, e_1, e_2) \Rightarrow \mu', \chi', v' \\
\hline
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_NOT}, \text{CMP}(\text{C_LE}, e_1, e_2)) \Rightarrow \mu', \chi', v' \\
\rho, \mu, \mathbf{nil}, v \vdash^e \text{CMP}(\text{C_GT}, e_1, e_2) \Rightarrow \mu', \chi', v' \\
\hline
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_NOT}, \text{CMP}(\text{C_LT}, e_1, e_2)) \Rightarrow \mu', \chi', v' \\
\rho, \mu, \mathbf{nil}, v \vdash^e \text{CMP}(\text{C_GE}, e_1, e_2) \Rightarrow \mu', \chi', v' \\
\hline
\end{array}
\begin{array}{l}
(\text{NE}) \\
(\text{GT}) \\
(\text{GE})
\end{array}$$

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

$$\begin{array}{c}
\rho, \mu, \chi, v \vdash^e e_2 \Rightarrow \mu_2, \chi_2, v_2 \\
\rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \chi_1, v_1 \\
\hline
\chi_1 \neq \mathbf{nil} \\
\rho, \mu, \chi, v \vdash^e \text{CMP}(op, e_1, e_2) \Rightarrow \mu_1, \chi_1, v_1 \quad (\text{CMP}^x)
\end{array}$$

1.5 Assignments

$$\begin{array}{c}
\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \\
\rho, \mu_e, \chi_e, v_e \vdash^e \text{OP1}(\text{M_ADDR}, a) \Rightarrow \mu_a, \mathbf{nil}, v_a \\
\hline
\rho, \mu, \chi, v \vdash^e \text{SET}(a, e) \Rightarrow \mu_a^{64}[v_a \mapsto v_e], \mathbf{nil}, v_e \quad (\text{PTR}^{\leftarrow}) \\
\\
\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \\
\rho, \mu_e, \chi_e, v_e \vdash^e \text{OP1}(\text{M_ADDR}, a) \Rightarrow \mu_a, \chi_a, v_a \quad \chi_a \neq \mathbf{nil} \\
\hline
\rho, \mu, \chi, v \vdash^e \text{SET}(a, e) \Rightarrow \mu_a, \chi_a, v_a \quad (\text{PTR}^{\leftarrow x})
\end{array}$$

1.6 Increments

$$\begin{array}{c}
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_ADDR}, a) \Rightarrow \mu_a, \mathbf{nil}, v_a \\
k = \mu_a^{64}(v_a) \\
\hline
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_POST_INC}, a) \Rightarrow \mu_a^{64}[v_a \mapsto k+1], \mathbf{nil}, k \quad (\text{POST}^\uparrow) \\
\\
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_ADDR}, a) \Rightarrow \mu_a, \mathbf{nil}, v_a \\
k = \mu_a^{64}(v_a) \\
\hline
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_POST_DEC}, a) \Rightarrow \mu_a^{64}[v_a \mapsto k-1], \mathbf{nil}, k \quad (\text{POST}^\downarrow) \\
\\
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_ADDR}, a) \Rightarrow \mu_a, \mathbf{nil}, v_a \\
k = \mu_a^{64}(v_a) \\
\hline
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_PRE_INC}, a) \Rightarrow \mu_a^{64}[v_a \mapsto k+1], \mathbf{nil}, k+1 \quad (\text{PRE}^\uparrow) \\
\\
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_ADDR}, a) \Rightarrow \mu_a, \mathbf{nil}, v_a \\
k = \mu_a^{64}(v_a) \\
\hline
\rho, \mu, \chi, v \vdash^e \text{OP1}(\text{M_PRE_DEC}, a) \Rightarrow \mu_a^{64}[v_a \mapsto k-1], \mathbf{nil}, k-1 \quad (\text{PRE}^\downarrow)
\end{array}$$

1.7 Extended assignments

Let $op \in \text{bin_op} \setminus \{\text{S_INDEX}\}$.

$$\begin{array}{c}
\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \\
\rho, \mu_e, \chi_e, v_e \vdash^e \text{OP1}(\text{M_ADDR}, a) \Rightarrow \mu_a, \chi_a, v_a \\
v_a \in \text{dom } \mu_a \quad \mu_a^{64}(v_a) = u \\
\rho, \mu_a, \chi_a, v_a \vdash^e \text{OP2}(op, \text{CST } v_a, \text{CST } v_e) \Rightarrow \mu', \mathbf{nil}, w \\
\hline
\rho, \mu, \chi, v \vdash^e \text{OPSET}(op, a, e) \Rightarrow \mu'[v_a \mapsto w], \mathbf{nil}, w \quad (\text{OPSET}) \\
\\
\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \\
\rho, \mu_e, \chi_e, v_e \vdash^e \text{OP1}(\text{M_ADDR}, a) \Rightarrow \mu_a, \chi_a, v_a \\
\hline
\chi_a \neq \mathbf{nil} \\
\rho, \mu, \chi, v \vdash^e \text{OPSET}(op, a, e) \Rightarrow \mu_a, \chi_a, v_a \quad (\text{OPSET}^x)
\end{array}$$

1.8 Ternary operator

$$\begin{array}{c}
\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathbf{nil}, v_e \quad v_e = 0 \\
\hline
\rho, \mu_e, \mathbf{nil}, v_e \vdash^e e_\perp \Rightarrow \mu_\perp, \chi_\perp, v_\perp \\
\rho, \mu, \chi, v \vdash^e \mathbf{EIF}(e, e_\top, e_\perp) \Rightarrow \mu_\perp, \chi_\perp, v_\perp \quad (\text{TERN}^\perp) \\
\\
\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathbf{nil}, v_e \quad v_e \neq 0 \\
\hline
\rho, \mu_e, \mathbf{nil}, v_e \vdash^e e_\top \Rightarrow \mu_\top, \chi_\top, v_\top \\
\rho, \mu, \chi, v \vdash^e \mathbf{EIF}(e, e_\top, e_\perp) \Rightarrow \mu_\top, \chi_\top, v_\top \quad (\text{TERN}^\top) \\
\\
\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \quad \chi_e \neq \mathbf{nil} \\
\hline
\rho, \mu, \chi, v \vdash^e \mathbf{EIF}(e, e_\top, e_\perp) \Rightarrow \mu_e, \chi_e, v_e \quad (\text{TERN}^\chi)
\end{array}$$

1.9 Sequence

$$\begin{array}{c}
\rho, \mu_0, \chi_0, v_0 \vdash^e e_1 \Rightarrow \mu_1, \chi_1, v_1 \\
\vdots \\
\rho, \mu_{n-1}, \chi_{n-1}, v_{n-1} \vdash^e e_n \Rightarrow \mu_n, \chi_n, v_n \\
\hline
\rho, \mu_0, \chi_0, v_0 \vdash^e \mathbf{ESEQ}[e_1; \dots; e_n] \Rightarrow \mu_n, \chi_n, v_n \quad (\text{SEQ}^n)
\end{array}$$

1.10 Function call

Works for both a toplevel function and a function pointer:

$$\begin{array}{c}
\rho, \mu_{n+1}, \chi_{n+1}, v_{n+1} \vdash^e e_n \Rightarrow \mu_n, \chi_n, v_n \\
\vdots \\
\rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \mathbf{nil}, v_1 \\
f \in \text{dom } \rho \quad \rho(f) \in \text{dom fun} \\
\rho_g, \mu_1, \mathbf{nil}, 0 \vdash^e \text{fun}(\rho(f))(v_1, \dots, v_n) \Rightarrow \rho_f, \mu_f, \chi_f, v_f \\
\hline
\rho, \mu_n, \chi_n, v_n \vdash^e \mathbf{CALL}(f, [e_1; \dots; e_n]) \Rightarrow \mu_f, \chi_f, v_f \quad (\text{CALL}^n) \\
\\
\rho, \mu_{n+1}, \chi_{n+1}, v_{n+1} \vdash^e e_n \Rightarrow \mu_n, \chi_n, v_n \\
\vdots \\
\rho, \mu_2, \chi_2, v_2 \vdash^e e_1 \Rightarrow \mu_1, \chi_1, v_1 \\
\chi_1 \neq \mathbf{nil} \\
\hline
\rho, \mu_{n+1}, \chi_{n+1}, v_{n+1} \vdash^e \mathbf{CALL}(f, [e_1; \dots; e_n]) \Rightarrow \mu_1, \chi_1, v_1 \quad (\text{CALL}^\chi)
\end{array}$$

2 Code

2.1 Expressions

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

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e}{\rho, \mu, \chi, v \vdash^c \mathbf{CEXP} e \Rightarrow \rho, \mu_e, \chi_e, v_e} (\text{EXPR})$$

2.2 Conditional branching

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

$$\begin{array}{c}
\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathbf{nil}, v_e \quad v_e = 0 \\
\hline
\rho, \mu_e, \mathbf{nil}, v_e \vdash^c c_\perp \Rightarrow \rho_\perp, \mu_\perp, \chi_\perp, v_\perp \\
\rho, \mu, \chi, v \vdash^c \mathbf{CIF}(e, c_\top, c_\perp) \Rightarrow \rho, \mu_\perp, \chi_\perp, v_\perp \quad (\text{IF}^\perp) \\
\\
\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathbf{nil}, v_e \quad v_e \neq 0 \\
\hline
\rho, \mu_e, \mathbf{nil}, v_e \vdash^c c_\top \Rightarrow \rho_\top, \mu_\top, \chi_\top, v_\top \\
\rho, \mu, \chi, v \vdash^c \mathbf{CIF}(e, c_\top, c_\perp) \Rightarrow \rho, \mu_\top, \chi_\top, v_\top \quad (\text{IF}^\top)
\end{array}$$

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.

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \quad \chi_e \neq \mathbf{nil}}{\rho, \mu, \chi, v \vdash^c \mathbf{CIF}(e, c_\top, c_\perp) \Rightarrow \rho, \mu_e, \chi_e, v_e} (\text{IF}^\chi)$$

2.3 Blocks

$$\frac{\rho, \mu, \chi, v \vdash^c c \Rightarrow \rho', \mu', \chi', v' \quad \rho', \mu', \chi', v' \vdash^c \text{CBLOCK } S \Rightarrow \rho'', \mu'', \chi'', v''}{\rho, \mu, \chi, v \vdash^c \text{CBLOCK}(c :: S) \Rightarrow \rho, \mu'', \chi'', v''} (\text{BLOCK}^1)$$

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

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:

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \text{nil}, v_e \quad v_e = 0}{\rho, \mu, \chi, v \vdash^c \text{CWHILE}(e, c, f, \text{true}) \Rightarrow \rho, \mu', \text{nil}, v} (\text{WHILE}^{\perp, \text{true}})$$

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

$$\frac{\rho, \mu, \chi, v \vdash^c c \Rightarrow \rho_c, \mu_c, \chi_c, v_c \quad \rho, \mu_c, \chi_c, v_c \vdash^e f \Rightarrow \mu_f, \text{nil}, v_f \quad \rho, \mu_f, \text{nil}, v_f \vdash^c \text{CWHILE}(e, c, f, \text{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w}{\rho, \mu, \chi, v \vdash^c \text{CWHILE}(e, c, f, \text{false}) \Rightarrow \rho, \mu_w, \chi_w, v_w} (\text{WHILE}^{\text{false}})$$

A loop continues normally if its condition is nonzero:

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \text{nil}, v_e \quad v_e \neq 0 \quad \rho, \mu_e, \text{nil}, v_e \vdash^c c \Rightarrow \rho_c, \mu_c, \chi_c, v_c \quad \chi_c \notin \{\text{brk}, \text{cnt}\} \quad \rho, \mu_c, \chi_c, v_c \vdash^e f \Rightarrow \mu_f, \chi_f, v_f \quad \rho, \mu_f, \chi_f, v_f \vdash^c \text{CWHILE}(e, c, f, \text{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w}{\rho, \mu, \chi, v \vdash^c \text{CWHILE}(e, c, f, \text{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w} (\text{WHILE}^{\top, \text{true}})$$

A flag skips the loop:

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \quad \chi_e \neq \text{nil}}{\rho, \mu, \chi, v \vdash^c \text{CWHILE}(e, c, f, \text{true}) \Rightarrow \rho, \mu_e, \chi_e, v_e} (\text{WHILE}^{\chi, \text{true}})$$

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

cnt executes the finally clause before continuing as normal:

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \text{nil}, v_e \quad v_e \neq 0 \quad \rho, \mu_e, \text{nil}, v_e \vdash^c c \Rightarrow \rho_c, \mu_c, \text{cnt}, v_c \quad \rho, \mu_c, \text{nil}, v_c \vdash^e f \Rightarrow \mu_f, \chi_f, v_f \quad \rho, \mu_f, \chi_f, v_f \vdash^c \text{CWHILE}(e, c, f, \text{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w}{\rho, \mu, \chi, v \vdash^c \text{CWHILE}(e, c, f, \text{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w} (\text{WHILE}^{\text{cnt}, \text{true}})$$

$$\frac{\rho, \mu, \chi, v \vdash^c c \Rightarrow \rho_c, \mu_c, \text{cnt}, v_c \quad \rho, \mu_c, \text{nil}, v_c \vdash^e f \Rightarrow \mu_f, \chi_f, v_f \quad \rho, \mu_f, \chi_f, v_f \vdash^c \text{CWHILE}(e, c, f, \text{true}) \Rightarrow \rho, \mu_w, \chi_w, v_w}{\rho, \mu, \chi, v \vdash^c \text{CWHILE}(e, c, f, \text{false}) \Rightarrow \rho, \mu_w, \chi_w, v_w} (\text{WHILE}^{\text{cnt}, \text{false}})$$

brk interrupts the loop but is not retransmitted:

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \text{nil}, v_e \quad v_e \neq 0 \quad \rho, \mu_e, \text{nil}, v_e \vdash^c c \Rightarrow \rho_c, \mu_c, \text{brk}, v_c}{\rho, \mu, \chi, v \vdash^c \text{CWHILE}(e, c, f, \text{true}) \Rightarrow \rho, \mu_c, \text{nil}, v_c} (\text{WHILE}^{\text{brk}, \text{true}})$$

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

2.5 Control flow

$$\begin{array}{c}
\frac{}{\rho, \mu, \mathbf{nil}, v \vdash^c \mathbf{CBREAK} \Rightarrow \rho, \mu, \mathbf{brk}, 0} (\mathbf{BREAK}) \\
\\
\frac{\chi \neq \mathbf{nil}}{\rho, \mu, \chi, v \vdash^c \mathbf{CBREAK} \Rightarrow \rho, \mu, \chi, v} (\mathbf{BREAK}^\chi) \\
\\
\frac{}{\rho, \mu, \mathbf{nil}, v \vdash^c \mathbf{CCONTINUE} \Rightarrow \rho, \mu, \mathbf{cnt}, 0} (\mathbf{CONTINUE}) \\
\\
\frac{\chi \neq \mathbf{nil}}{\rho, \mu, \chi, v \vdash^c \mathbf{CCONTINUE} \Rightarrow \rho, \mu, \chi, v} (\mathbf{CONTINUE}^\chi) \\
\\
\frac{}{\rho, \mu, \mathbf{nil}, v \vdash^c \mathbf{CRETURN} \mathbf{None} \Rightarrow \rho, \mu, \mathbf{ret}, 0} (\mathbf{RETURN}^{\mathbf{None}}) \\
\\
\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \mathbf{nil}, v_e}{\rho, \mu, \mathbf{nil}, v \vdash^c \mathbf{CRETURN}(\mathbf{Some} \ e) \Rightarrow \rho, \mu_e, \mathbf{ret}, v_e} (\mathbf{RETURN}^{\mathbf{Some}}) \\
\\
\frac{\chi \neq \mathbf{nil}}{\rho, \mu, \chi, v \vdash^c \mathbf{CRETURN} \mathbf{None} \Rightarrow \rho, \mu, \chi, v} (\mathbf{RETURN}^{\mathbf{None}^\chi}) \\
\\
\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \quad \chi_e \neq \mathbf{nil}}{\rho, \mu, \mathbf{nil}, v \vdash^c \mathbf{CRETURN}(\mathbf{Some} \ e) \Rightarrow \rho, \mu_e, \chi_e, v_e} (\mathbf{RETURN}^{\mathbf{Some}^\chi})
\end{array}$$

2.6 Local variable declarations

When there are none:

$$\frac{}{\rho, \mu, \chi, v \vdash^c \mathbf{CLOCAL} \ \square \Rightarrow \rho, \mu, \chi, v} (\mathbf{LOCAL}^0)$$

When an error occurs:

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

Otherwise:

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

2.7 Throw

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

$$\frac{\rho, \mu, \chi, v \vdash^e e \Rightarrow \mu_e, \chi_e, v_e \quad \chi_e \neq \mathbf{nil}}{\rho, \mu, \chi, v \vdash^c \mathbf{CTHROW}(s, e) \Rightarrow \rho, \mu_e, \chi_e, v_e} (\mathbf{THROW}^\chi)$$

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

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

2.8 Switch

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

Where for $L = [(j_1, l_1); \dots; (j_n, l_n)]$, $L(v_e)$ is defined as follows :

Let $I_i = \{j_1, \dots, j_i\}$ for $1 \leq i \leq n$, $I_{n+1} = \mathbb{Z}_{64}$.

$\tilde{j} \triangleq \min_{1 \leq i \leq n+1} \{i \mid v_e \in I_i\}$, finally $L(v_e) \triangleq [l_{\tilde{j}}; \dots; l_n; c]$.

2.9 Try

Skip the block when a flag is already raised:

$$\frac{\chi \neq \mathbf{nil}}{\rho, \mu, \chi, v \vdash^c \mathbf{CTRY}(c, L, f) \Rightarrow \rho, \mu, \chi, v} (\text{TRY}^\chi)$$

For $L = [(e_1, x_1, c_1); \dots; (e_n, x_n, c_n)]$, let $E = \{e_i | 1 \leq i \leq n\} \subset S$.

When no exception is raised:

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

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

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

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

When the handler is able to catch the exception:

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

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

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

3 Declarations

3.1 Global variables

$$\frac{k \notin \text{dom } \rho \quad k \in \text{alloc}^8(\mu)}{\rho, \mu, \text{fun} \vdash^d \mathbf{CDECL}(x, \mathbf{None}) \Rightarrow \rho[x \mapsto k], \mu[k \mapsto 0], \text{fun}} (\text{DECL}^{\mathbf{None}})$$

$$\frac{x \notin \text{dom } \rho \quad k \in \text{alloc}^8(\mu)}{\rho, \mu, \text{fun} \vdash^d \mathbf{CDECL}(x, \mathbf{Some}(\mathbf{CST } c)) \Rightarrow \rho[x \mapsto k], \mu[k \mapsto c], \text{fun}} (\text{DECL}^{\mathbf{CST}})$$

$$\frac{s \text{ stored at } a \in \text{Addr} \quad x \notin \text{dom } \rho}{\rho, \mu, \text{fun} \vdash^d \mathbf{CDECL}(x, \mathbf{Some}(\mathbf{STRING } s)) \Rightarrow \rho[x \mapsto a], \mu, \text{fun}} (\text{DECL}^{\mathbf{STRING}})$$

3.2 Functions

$$\frac{f \notin \text{dom } \rho \quad \phi_A(b) \text{ stored at } k}{\rho, \mu, \text{fun} \vdash^d \mathbf{CFUN}(f, A, b) \Rightarrow \rho[f \mapsto k], \mu, \text{fun}[k \mapsto \phi_A(b)]} (\text{FUN})$$

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

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

4 Program

Finally, here's how the whole program executes:

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

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`, ...)