

The Vapor and Vapor-M Operational Semantics

Jens Palsberg

October 10, 2016

1 What are Vapor and Vapor-M?

Vapor and Vapor-M are languages that we will use as intermediate languages when we compile MiniJava to MIPS. The translation steps are: $\text{MiniJava} \rightarrow \text{Vapor} \rightarrow \text{Vapor-M} \rightarrow \text{MIPS}$.

A Vapor program consists of functions that each operates on a heap, global constants, parameters, local variables, and a stack. We will specify Vapor's abstract syntax and operational semantics.

A Vapor-M program consists of functions that each operates on a heap, global constants, global registers, and a stack. We will specify Vapor-M's abstract syntax and operational semantics. Vapor and Vapor-M are closely related. One difference is that in Vapor-M, each function has no parameters, no local variables, and no return value.

2 Notation

2.1 Grammars

The grammars for Vapor and Vapor-M use the following metanotation:

- Nonterminal symbols are words written in *this font*.
- Terminal symbols are written in **this font**, except $\langle \text{STRING} \rangle$, $\langle \text{LABEL} \rangle$, $\langle \text{IDENTIFIER} \rangle$, and $\langle \text{INTEGER.LITERAL} \rangle$.
- A production is of the form $lhs ::= rhs$, where lhs is a nonterminal symbol and rhs is a sequence of nonterminal and terminal symbols, with choices separated by $|$, and some times using “...” to denote a possibly empty list.
- We will use superscripts and subscripts to distinguish metavariables.

2.2 Rules

We will use the following notation:

$$\frac{hypothesis_1 \quad hypothesis_2 \quad \dots \quad hypothesis_n}{conclusion}$$

This is a *rule* that says that if we can derive all of $hypothesis_1, hypothesis_2, \dots, hypothesis_n$, then we can also derive *conclusion*.

A special case arises when $n = 0$: we can write this case as:

$$\frac{}{conclusion}$$

or we can even omit the horizontal bar and write:

$$conclusion$$

We can say that this case is a rule with no hypotheses, or we can call it an *axiom*.

A *derivation* happens when we begin with one or more axioms, then perhaps apply some rules, and finally arrive at a conclusion. Notice that we can organize a derivation as a tree that has the axioms as leaves and the conclusion as the root. We can refer to such a tree as a *derivation tree*.

2.3 Maps

A *map* is a function with finite domain. If M is a map, then $dom(M)$ denotes the domain of M . If x_1, \dots, x_r are pairwise distinct, then $[x_1 \mapsto y_1, \dots, x_n \mapsto y_n]$ denotes a map with domain $\{x_1, \dots, x_n\}$, which maps x_i to y_i , for $i \in 1..n$. If M_1, M_2 are maps, then $M_1 \cdot M_2$ is a map:

$$(M_1 \cdot M_2)(id) = \begin{cases} M_2(id) & \text{if } id \in dom(M_2) \\ M_1(id) & \text{otherwise} \end{cases}$$

Notice that M_2 takes precedence over M_1 .

If M is a map and X is a set, then $M \setminus X$ denotes M restricted to $dom(M) \cap X$.

We define a helper function *initmap* that maps a set to a map.

$$(initmap(X))(x) = \begin{cases} 0 & x \in X \\ undefined & \text{otherwise} \end{cases}$$

If M is a map, then we define the notation M^* as follows.

$$M^*(x) = \begin{cases} M(x) & \text{if } x \in dom(M) \\ x & \text{otherwise} \end{cases} \quad \text{if } x \text{ is already defined in the map, use the mapped value. otherwise, map it to itself and put it in the new map}$$

2.4 Tuples

$$(Tuple) \quad t ::= \langle y_1, \dots, y_n \rangle$$

We define a helper function *inittuple* that maps a positive integer to a tuple.

$$inittuple(c) = \langle 0, \dots, 0 \rangle$$

where the number of 0's in the tuple is c , where $c > 0$

3 Vapor

3.1 Syntax

$(Program) \ p ::= C_1 \dots C_n \ F_1 \dots F_m$
 $(ConstDecl) \ C ::= \text{const } l \ l_1 \dots l_n$
 $(FunDecl) \ F ::= \text{func } l \ (id_1 \dots id_f) \ l_1 \ b_1 \dots l_q \ b_q$
 $(Block) \ b ::= i_1 \dots i_n \ j \text{ several instructions ending with a jump}$
 $(Instr) \ i ::= id = o \mid id = op \ (o_1 \ o_2) \mid id = m \mid m = id$
 $\quad \mid \text{if0 } o \text{ goto } l \mid id = \text{call } o \ (o_1 \dots o_f)$
 $\quad \mid id = \text{HeapAllocZ } (o) \mid \text{PrintIntS } (o) \mid \text{Error } (s)$
 $(Jump) \ j ::= \text{goto } l \mid \text{ret } o \mid \text{ret}$
 $(MemRef) \ m ::= [id + c]$
 $(Operator) \ op ::= \text{Add} \mid \text{Sub} \mid \text{MulS} \mid \text{Eq} \mid \text{LtS}$
 $(Operand) \ o ::= l \mid c \mid id$
 $(StringLiteral) \ s ::= \langle \text{STRING} \rangle$
 $(Label) \ l ::= \langle \text{LABEL} \rangle$
 $(IntegerLiteral) \ c ::= \langle \text{INTEGER_LITERAL} \rangle$
 $(Identifier) \ id ::= \langle \text{IDENTIFIER} \rangle$

3.2 Helper Functions

Defined Variables. We define a helper function *defined* that maps a block to the set of local variables that the blocks assigns. We will overload *defined* and define it also for instructions.

$$\begin{aligned}
 \text{defined}(i_1 \dots i_n \ j) &= (\text{defined}(i_1) \cup \dots \text{defined}(i_n)) \\
 \text{defined}(id = o) &= \{id\} \\
 \text{defined}(id = op \ (o_1 \ o_2)) &= \{id\} \\
 \text{defined}(id = m) &= \{id\} \\
 \text{defined}(m = id) &= \emptyset \\
 \text{defined}(\text{if0 } o \text{ goto } l) &= \emptyset \\
 \text{defined}(id = \text{call } o \ (o_1 \dots o_f)) &= \{id\} \\
 \text{defined}(\text{HeapAllocZ } (o)) &= \emptyset \\
 \text{defined}(\text{PrintIntS } (o)) &= \emptyset \\
 \text{defined}(\text{Error } (s)) &= \emptyset
 \end{aligned}$$

Initialization of Constants. We define a helper function *initconst* that maps a constant declaration to a map.

$$\text{initconst}(\text{const } l \ l_1 \dots l_n) = [l \mapsto \langle l_1 \dots l_n \rangle]$$

Initialization of Functions. We define a helper function *initfun* that maps a function declaration to a map.

$$\begin{aligned}
 \text{initfun}(F) &= [l \mapsto F, \ l_1 \mapsto b_1, \dots, \ l_q \mapsto b_q] \\
 \text{where} \\
 F &= \text{func } l \ (id_1 \dots id_f) \ l_1 \ b_1 \dots l_q \ b_q
 \end{aligned}$$

3.3 Program States

Vapor has three kinds of values: labels l , heap addresses (l, c) , and integers c . We use v to range over values.

A program state (G, H, E, b) has four components. Intuitively, G is a *global table* that represents the constants, functions, and blocks; H is the *heap*; E is an *environment* that represents the parameters and local variables; and b is the block that is executing right now.

The Global Table. The global table is a map from labels to either tuples, functions, or blocks.

$$\begin{aligned} (GlobalData) \quad d &::= t \mid F \mid b \\ (GlobalTable) \quad G &::= [l_1 \mapsto d_1, \dots, l_n \mapsto d_n] \end{aligned}$$

The Heap. A heap is a map from labels to tuples. We use H to range over heaps. A *heap address* is a pair of the form (l, c) , where l is a label and c is an integer such that $c \geq 0$ and c is divisible by 4.

The Environment. An environment represents the parameters, and local variables. An environment is a map from identifiers to values. We use E to range over environments.

The Initial Program State. Consider a program

$$C_1 \dots C_n F_1 \dots F_m$$

where

$$F_1 = \text{func } id \ (\) \ l_1 \ b_1 \ \dots \ l_q \ b_q$$

Notice that F_1 has no parameters. The initial program state is (G, H, E, b) , where:

$$\begin{aligned} G &= initconst(C_1) \cdot \dots \cdot initconst(C_n) \cdot initfun(F_1) \cdot \dots \cdot initfun(F_m) \\ H &= [] \quad \text{heap is initially empty} \\ E &= initmap((defined(b_1) \cup \dots \cup defined(b_q))) \quad \text{environment contains all} \\ b &= b_1 \quad \text{program execution starts with the} \quad \text{variables defined in all} \\ &\quad \text{first block of the first function} \quad \text{instruction blocks in the} \\ &\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{first function} \end{aligned}$$

global table tracks all global constant declarations, as well as all function declarations with the associated labels of each block.

3.4 Semantics

Single Steps.

- $(G, H, E, id = o \quad b') \mapsto (G, H, E \cdot [id \mapsto E^*(o)], b')$ (2)
- $(G, H, E, id = \text{Add } (o_1 \ o_2) \quad b') \mapsto (G, H, E \cdot [id \mapsto (c_1 + c_2)], b')$ (3)
 if $E^*(o_1) = c_1$ and $E^*(o_2) = c_2$ standard $x = a + b$ behavior
- $(G, H, E, id = \text{Add } (o_1 \ o_2) \quad b') \mapsto (G, H, E \cdot [id \mapsto (l, c_1 + c_2)], b')$ (4)
 if $E^*(o_1) = (l, c_1)$ and $E^*(o_2) = c_2$
 where $c_2 \geq 0$ and c_2 is divisible by 4 (5)
- $(G, H, E, id = \text{Sub } (o_1 \ o_2) \quad b') \mapsto (G, H, E \cdot [id \mapsto (c_1 - c_2)], b')$ (6)
 if $E^*(o_1) = c_1$ and $E^*(o_2) = c_2$ standard $x = a - b$ behavior
- $(G, H, E, id = \text{MulS } (o_1 \ o_2) \quad b') \mapsto (G, H, E \cdot [id \mapsto (c_1 \times c_2)], b')$ (7)
 if $E^*(o_1) = c_1$ and $E^*(o_2) = c_2$ standard $x = a * b$ behavior
- $(G, H, E, id = \text{Eq } (o_1 \ o_2) \quad b') \mapsto (G, H, E \cdot [id \mapsto 1], b')$ (8)
 if $E^*(o_1) = c_1$ and $E^*(o_2) = c_2$ and $c_1 = c_2$ standard $x == y$ behavior:
- $(G, H, E, id = \text{Eq } (o_1 \ o_2) \quad b') \mapsto (G, H, E \cdot [id \mapsto 0], b')$ (9)
 if $E^*(o_1) = c_1$ and $E^*(o_2) = c_2$ and $c_1 \neq c_2$ ($x == y \rightarrow 1$)
($x != y \rightarrow 0$)
- $(G, H, E, id = \text{LtS } (o_1 \ o_2) \quad b') \mapsto (G, H, E \cdot [id \mapsto 1], b')$ (10)
 if $E^*(o_1) = c_1$ and $E^*(o_2) = c_2$ and $c_1 < c_2$ standard $x < y$ behavior:
- $(G, H, E, id = \text{LtS } (o_1 \ o_2) \quad b') \mapsto (G, H, E \cdot [id \mapsto 0], b')$ (11)
 if $E^*(o_1) = c_1$ and $E^*(o_2) = c_2$ and $c_1 \geq c_2$ ($x < y \rightarrow 1$)
($x > y \rightarrow 0$)
- $(G, H, E, id = [id' + c] \quad b') \mapsto (G, H, E \cdot [id \mapsto v_{c'+c}], b')$ (12)
 where $E^*(id') = (l, c')$
 where $c \geq 0$ and c is divisible by 4 dereferencing a memory location either in the heap or in the global table:
 where $H(l) = \langle v_0, v_4, \dots, v_{c'+c}, \dots, v_n \rangle$ or else $G(l) = \langle v_0, v_4, \dots, v_{c'+c}, \dots, v_n \rangle$
 where $c' + c \leq n$
- $(G, H, E, [id + c] = id' \quad b') \mapsto (G, H[l \mapsto t'], E, b')$ (13)
 where $E^*(id) = (l, c')$ assigning a value to a variable stored in the heap
 where $c \geq 0$ and c is divisible by 4
 where $H(l) = \langle v_0, v_4, \dots, v_n \rangle$
 where $t' = \langle v_0, v_4, \dots, v_{c-4}, E^*(id'), v_{c+4}, \dots, v_n \rangle$
 where $c' + c \leq n$

$$(G, H, E, \text{if0 } o \text{ goto } l \text{ } b') \mapsto (G, H, E, b'') \quad \begin{array}{l} \text{if0 jump statement.} \\ \text{can only jump to locations in the same} \\ \text{function} \end{array} \quad (12)$$

if $E^*(o) = 0$ and $G(l) = b''$,
and $l \text{ } b''$ and $\text{if0 } o \text{ goto } l$ are in the body of the same function

$$(G, H, E, \text{if0 } o \text{ goto } l \text{ } b') \mapsto (G, H, E, b') \quad \begin{array}{l} \text{if not 0 on a if0} \\ \text{call, then don't} \\ \text{jump} \end{array} \quad (13)$$

if $E^*(o) = c$ and $c \neq 0$

$$\frac{(G, H, E^{init}, b_1) \mapsto (G, H', E', \text{ret } o')}{(G, H, E, \text{id} = \text{call } o \text{ (} o_1 \dots o_f \text{) } b') \mapsto (G, H', E \cdot [id \mapsto E'^*(o')], b')} \quad \begin{array}{l} \text{function calls} \end{array} \quad (14)$$

where $E^*(o) = l$
where $G(l) = \text{func } l \text{ (} id_1 \dots id_f \text{) } l_1 \text{ } b_1 \dots l_q \text{ } b_q$
where $E^{init} = [id_1 \mapsto E^*(o_1), \dots, id_f \mapsto E^*(o_f)] \cdot \text{initmap}(a)$
where $a = (\text{defined}(b_1) \cup \dots \cup \text{defined}(b_q)) \setminus \{id_1, \dots, id_f\}$

$$(G, H, E, \text{id} = \text{HeapAllocZ (} o \text{) } b') \mapsto (G, H \cup [l \mapsto t], E \cdot [id \mapsto (l, 0)], E, b') \quad \text{?}$$

where $l \notin \text{dom}(G, H)$ and $E^*(o)$ is a positive integer that is divisible by 4
where $t = \text{inittuple}(\frac{E^*(o)}{4})$

$$(G, H, E, \text{PrintIntS (} o \text{) } b') \mapsto (G, H, E, b') \quad \begin{array}{l} \text{printing an int doesn't modify the} \\ \text{program state} \end{array} \quad (16)$$

where $E^*(o) = c$
and display c on the screen

$$(G, H, E, \text{Error (} s \text{) } b') \quad \text{display } s \text{ on the screen and stop execution} \quad (17)$$

$$(G, H, E, \text{goto } l) \mapsto (G, H, E, b') \quad (18)$$

if $G(l) = b'$,
and $l \text{ } b'$ and $\text{goto } l$ are in the body of the same function

Multiple Steps.

$$\frac{(G, H, E, b) \mapsto (G', H', E', b') \quad (G', H', E', b') \mapsto (G'', H'', E'', b'')}{(G, H, E, b) \mapsto (G'', H'', E'', b'')} \quad (19)$$

if you can go from state A to state B and from state B to state C, then you can go from State A to state C:
"transitivity"

4 Vapor-M

4.1 Syntax

$(Program) \ p ::= C_1 \dots C_n \ F_1 \dots F_m$
 $(ConstDecl) \ C ::= \text{const } l \ l_1 \dots l_n$
 $(FunDecl) \ F ::= \text{func } l \ [\text{in } c_1, \text{out } c_2, \text{local } c_3] \ l_1 \ b_1 \dots l_q \ b_q$
 $(Block) \ b ::= i_1 \dots i_n \ j$
 $(Instr) \ i ::= id = o \mid id = op \ (\ o_1 \ o_2 \) \mid id = m \mid m = id$
 $\quad \mid \text{if0 } o \text{ goto } l \mid \text{call } o$
 $\quad \mid id = \text{HeapAllocZ} \ (\ o \) \mid \text{PrintIntS} \ (\ o \) \mid \text{Error} \ (\ s \)$
 $(Jump) \ j ::= \text{goto } l \mid \text{ret}$
 $(MemRef) \ m ::= [\ id + c \] \mid \text{in } [\ c \] \mid \text{out } [\ c \] \mid \text{local } [\ c \]$
 $(Operator) \ op ::= \text{Add} \mid \text{Sub} \mid \text{MulS} \mid \text{Eq} \mid \text{LtS}$
 $(Operand) \ o ::= l \mid c \mid id$
 $(StringLiteral) \ s ::= \langle \text{STRING} \rangle$
 $(Label) \ l ::= \langle \text{LABEL} \rangle$
 $(IntegerLiteral) \ c ::= \langle \text{INTEGER_LITERAL} \rangle$
 $(Identifier) \ id ::= \langle \text{IDENTIFIER} \rangle$

4.2 Helper Functions

Defined Variables. We define a helper function *defined* that maps a block to the set of local variables that the blocks assigns. We will overload *defined* and define it also for instructions.

$$\begin{aligned}
 \text{defined}(i_1 \dots i_n \ j) &= (\text{defined}(i_1) \cup \dots \text{defined}(i_n)) \\
 \text{defined}(id = o) &= \{id\} \\
 \text{defined}(id = op \ (\ o_1 \ o_2 \)) &= \{id\} \\
 \text{defined}(id = m) &= \{id\} \\
 \text{defined}(m = id) &= \emptyset \\
 \text{defined}(\text{if0 } o \text{ goto } l) &= \emptyset \\
 \text{defined}(\text{call } o) &= \emptyset \\
 \text{defined}(\text{HeapAllocZ} \ (\ o \)) &= \emptyset \\
 \text{defined}(\text{PrintIntS} \ (\ o \)) &= \emptyset \\
 \text{defined}(\text{Error} \ (\ s \)) &= \emptyset
 \end{aligned}$$

Initialization of Constants. We define a helper function *initconst* that maps a constant declaration to a map.

$$\text{initconst}(\text{const } l \ l_1 \dots l_n) = [l \mapsto \langle l_1 \dots l_n \rangle]$$

Initialization of Functions. We define a helper function *initfun* that maps a function declaration to a map.

$$\begin{aligned}
 \text{initfun}(F) &= [l \mapsto F, \ l_1 \mapsto b_1, \dots, \ l_q \mapsto b_q] \\
 \text{where} \\
 F &= \text{func } l \ [\text{in } c_1, \text{out } c_2, \text{local } c_3] \ l_1 \ b_1 \dots l_q \ b_q
 \end{aligned}$$

4.3 Program States

Vapor-M has three kinds of values: labels l , heap addresses (l, c) , and integers c . We use v to range over values.

A program state (G, H, R, S, b) has five components. Intuitively, G is a *global table* that represents the constants, functions, and blocks; H is the *heap*; R is the *register file*; S is the *stack*; and b is the block that is executing right now.

The Global Table. The global table is a map from labels to either tuples, functions, or blocks.

$$\begin{aligned} (\text{GlobalData}) \quad d &::= t \mid F \mid b \\ (\text{GlobalTable}) \quad G &::= [l_1 \mapsto d_1, \dots, l_n \mapsto d_n] \end{aligned}$$

The Heap. A heap is a map from labels to tuples. We use H to range over heaps. A *heap address* is a pair of the form (l, c) , where l is a label and c is an integer such that $c \geq 0$ and c is divisible by 4.

The Register File. The set *registers* consists of 23 identifiers that are akin to the names of 23 MIPS registers.

$$\text{registers} = \{\text{s0}, \dots, \text{s7}, \text{t0}, \dots, \text{t8}, \text{a0}, \dots, \text{a3}, \text{v0}, \text{v1}\}$$

Intuitively, the elements of *registers* are the global registers. A register file is a map from *registers* to values. We use R to range over register files.

The Stack.

$$(\text{Stack}) \quad S ::= \text{empty} \mid S \odot t$$

The Initial Program State. Consider a program

$$C_1 \dots C_n F_1 \dots F_m$$

where

$$F_1 = \text{func } id \text{ [in } 0, \text{ out } c_2, \text{ local } c_3] \text{ } l_1 \text{ } b_1 \dots l_q \text{ } b_q$$

Notice that F_1 has no parameters and has “[in 0]”. The initial program state is (G, H, R, S, b) , where:

$$\begin{aligned} G &= \text{initconst}(C_1) \cdot \dots \cdot \text{initconst}(C_n) \cdot \text{initfun}(F_1) \cdot \dots \cdot \text{initfun}(F_m) \\ H &= [] \\ R &= \text{initmap}(\text{registers}) \\ S &= \text{empty} \odot \text{inittuple}(c_3) \odot \text{inittuple}(c_2) \\ b &= b_1 \end{aligned}$$

4.4 Semantics

Single Steps.

$$(G, H, R, S, id = o \quad b') \mapsto (G, H, R \cdot [id \mapsto R^*(o)], S, b') \quad (20)$$

$$(G, H, R, S, id = \text{Add } (o_1 \ o_2) \quad b') \mapsto (G, H, R \cdot [id \mapsto (c_1 + c_2)], S, b') \\ \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2 \quad (21)$$

$$(G, H, R, S, id = \text{Add } (o_1 \ o_2) \quad b') \mapsto (G, H, R \cdot [id \mapsto (l, c_1 + c_2)], S, b') \\ \text{if } R^*(o_1) = (l, c_1) \text{ and } R^*(o_2) = c_2 \\ \text{where } c_2 \geq 0 \text{ and } c_2 \text{ is divisible by 4} \quad (22)$$

$$(G, H, R, S, id = \text{Sub } (o_1 \ o_2) \quad b') \mapsto (G, H, R \cdot [id \mapsto (c_1 - c_2)], S, b') \\ \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2 \quad (23)$$

$$(G, H, R, S, id = \text{MulS } (o_1 \ o_2) \quad b') \mapsto (G, H, R \cdot [id \mapsto (c_1 \times c_2)], S, b') \\ \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2 \quad (24)$$

$$(G, H, R, S, id = \text{Eq } (o_1 \ o_2) \quad b') \mapsto (G, H, R \cdot [id \mapsto 1], S, b') \\ \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2 \text{ and } c_1 = c_2 \quad (25)$$

$$(G, H, R, S, id = \text{Eq } (o_1 \ o_2) \quad b') \mapsto (G, H, R \cdot [id \mapsto 0], S, b') \\ \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2 \text{ and } c_1 \neq c_2 \quad (26)$$

$$(G, H, R, S, id = \text{LtS } (o_1 \ o_2) \quad b') \mapsto (G, H, R \cdot [id \mapsto 1], S, b') \\ \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2 \text{ and } c_1 < c_2 \quad (27)$$

$$(G, H, R, S, id = \text{LtS } (o_1 \ o_2) \quad b') \mapsto (G, H, R \cdot [id \mapsto 0], S, b') \\ \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2 \text{ and } c_1 \geq c_2 \quad (28)$$

$$(G, H, R, S, id = [id' + c] \quad b') \mapsto (G, H, R \cdot [id \mapsto v_{c'+c}], S, b') \\ \text{where } R^*(id') = (l, c') \\ \text{where } c \geq 0 \text{ and } c \text{ is divisible by 4} \\ \text{where } H(l) = \langle v_0, v_4, \dots, v_{c'+c}, \dots, v_n \rangle \text{ or else } G(l) = \langle v_0, v_4, \dots, v_{c'+c}, \dots, v_n \rangle \\ \text{where } c' + c \leq n \quad (29)$$

$$(G, H, R, S, id = \text{in } [c] \quad b') \mapsto (G, H, R \cdot [id \mapsto v_c], S, b') \\ \text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out} \\ \text{where } t_{in} = \langle v_m \dots v_c \dots v_1, v_0 \rangle \quad (30)$$

$$(G, H, R, S, id = \text{out } [c] \quad b') \mapsto (G, H, R \cdot [id \mapsto v_c], S, b') \\ \text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out} \\ \text{where } t_{out} = \langle v_m \dots v_c \dots v_1, v_0 \rangle \quad (31)$$

$$(G, H, R, S, id = \text{local } [c] \quad b') \mapsto (G, H, R \cdot [id \mapsto v_c], S, b') \\ \text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out} \\ \text{where } t_{local} = \langle v_m \dots v_c \dots v_1, v_0 \rangle \quad (32)$$

$$(G, H, R, S, [id + c] = id' \quad b') \mapsto (G, H[l \mapsto t'], R, S, b') \\ \text{where } R^*(id) = (l, c') \\ \text{where } c \geq 0 \text{ and } c \text{ is divisible by 4} \\ \text{where } H(l) = \langle v_0, v_4, \dots, v_n \rangle \\ \text{where } t' = \langle v_0, v_4, \dots, v_{c-4}, R^*(id'), v_{c+4}, \dots, v_n \rangle \\ \text{where } c' + c \leq n \quad (33)$$

$$\begin{aligned}
(G, H, R, S, \text{in } [c] = id \ b') &\mapsto (G, H, R, S \odot t'_{in} \odot t_{local} \odot t_{out}, b') \\
&\text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out} \\
&\text{where } t_{in} = \langle v_m \dots v_c \dots v_1, v_0 \rangle \\
&\text{where } t'_{in} = \langle v_m \dots R^*(id) \dots v_1, v_0 \rangle
\end{aligned} \tag{34}$$

$$\begin{aligned}
(G, H, R, S, \text{out } [c] = id \ b') &\mapsto (G, H, R, S \odot t_{in} \odot t_{local} \odot t'_{out}, b') \\
&\text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out} \\
&\text{where } t_{out} = \langle v_m \dots v_c \dots v_1, v_0 \rangle \\
&\text{where } t'_{out} = \langle v_m \dots R^*(id) \dots v_1, v_0 \rangle
\end{aligned} \tag{35}$$

$$\begin{aligned}
(G, H, R, S, \text{local } [c] = id \ b') &\mapsto (G, H, R, S \odot t_{in} \odot t'_{local} \odot t_{out}, b') \\
&\text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out} \\
&\text{where } t_{local} = \langle v_m \dots v_c \dots v_1, v_0 \rangle \\
&\text{where } t'_{local} = \langle v_m \dots R^*(id) \dots v_1, v_0 \rangle
\end{aligned} \tag{36}$$

$$\begin{aligned}
(G, H, R, S, \text{if0 } o \text{ goto } l \ b') &\mapsto (G, H, R, S, b'') \\
&\text{if } R^*(o) = 0 \text{ and } G(l) = b'', \\
&\text{and } l \ b'' \text{ and } \text{if0 } o \text{ goto } l \text{ are in the body of the same function}
\end{aligned} \tag{37}$$

$$\begin{aligned}
(G, H, R, S, \text{if0 } o \text{ goto } l \ b') &\mapsto (G, H, R, S, b') \\
&\text{if } R^*(o) = c \text{ and } c \neq 0
\end{aligned} \tag{38}$$

$$\begin{aligned}
&\frac{(G, H, R, S \odot \text{inittuple}(c_3) \odot \text{inittuple}(c_2), b_1) \mapsto (G, H', R', S' \odot t_3 \odot t_2, \text{ret})}{(G, H, R, S, \text{call } o \ b') \mapsto (G, H', R', S', b')} \\
&\text{where } R^*(o) = l \\
&\text{where } G(l) = \text{func } l \ [\text{in } c_1, \text{out } c_2, \text{local } c_3] \ l_1 \ b_1 \dots l_q \ b_q
\end{aligned} \tag{39}$$

$$\begin{aligned}
(G, H, R, S, id = \text{HeapAllocZ } (o) \ b') &\mapsto (G, H \cup [l \mapsto t], R \cdot [id \mapsto (l, 0)], S, b') \\
&\text{where } l \notin \text{dom}(G, H) \text{ and } R^*(o) \text{ is a positive integer that is divisible by 4} \\
&\text{where } t = \text{inittuple}(\frac{R^*(o)}{4})
\end{aligned} \tag{40}$$

$$\begin{aligned}
(G, H, R, S, \text{PrintIntS } (o) \ b') &\mapsto (G, H, R, S, b') \\
&\text{where } R^*(o) = c \\
&\text{and display } c \text{ on the screen}
\end{aligned} \tag{41}$$

$$\begin{aligned}
(G, H, R, S, \text{Error } (s) \ b') &\mapsto (G, H, R, S, b') \\
&\text{display } s \text{ on the screen and stop execution}
\end{aligned} \tag{42}$$

$$\begin{aligned}
(G, H, R, S, \text{goto } l) &\mapsto (G, H, R, S, b') \\
&\text{if } G(l) = b', \\
&\text{and } l \ b' \text{ and } \text{goto } l \text{ and in the body of the same function}
\end{aligned} \tag{43}$$

Multiple Steps.

$$\frac{(G, H, R, S, b) \mapsto (G', H', R', S', b') \quad (G', H', R', S', b') \mapsto (G'', H'', R'', S'', b'')}{(G, H, R, S, b) \mapsto (G'', H'', R'', S'', b'')} \tag{44}$$