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: MiniJava $\to$ Vapor $\to$ Vapor-M $\to$ 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 $\langle\mathsf{STRING}\rangle$, $\langle\mathsf{LABEL}\rangle$, $\langle\mathsf{IDENTIFIER}\rangle$, and $\langle\mathsf{INTEGER\_LITERAL}\rangle$.
- 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:

$$\frac{hypothesis_1 \qquad hypothesis_2 \quad \ldots \quad hypothesis_n}{conclusion}$$

This is a *rule* that says that if we can derive all of $hypothesis_1$, $hypothesis_2$, $\ldots$, $hypothesis_n$, 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 as 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 $x_1, \ldots, x_r$ are pairwise distinct, then $[x_1 \mapsto y_1, \ldots, x_n \mapsto y_n]$ denotes a map with domain $\{x_1, \ldots, x_n\}$, 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.

$$M^*(x) \quad = \quad \begin{cases} M(x) & \text{if } x \in dom(M) \\ x & \text{otherwise} \end{cases}$$

2.4 Tuples

$$(\textit{Tuple}) \quad t \quad ::= \quad \langle y_1, \ldots, y_n \rangle$$

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

$$inittuple(c) = \langle 0, \ldots, 0 \rangle$$
where the number of 0's in the tuple is c, where $c > 0$

3 Vapor

3.1 Syntax

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

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.

$$
\mathit{defined}(\; i_1 \ldots i_n \; j \;) \;=\; (\mathit{defined}(i_1) \cup \ldots \; \mathit{defined}(i_n))
$$

$$
\begin{array}{rcl}
\mathit{defined}(\; id \; \texttt{=} \; o \;) & = & \{id\} \\
\mathit{defined}(\; id \; \texttt{=} \; op \; \texttt{(} \; o_1 \; o_2 \; \texttt{)} \;) & = & \{id\} \\
\mathit{defined}(\; id \; \texttt{=} \; m \;) & = & \{id\} \\
\mathit{defined}(\; m \; \texttt{=} \; id \;) & = & \emptyset \\
\mathit{defined}(\; \texttt{if0} \; o \; \texttt{goto} \; l \;) & = & \emptyset \\
\mathit{defined}(\; id \; \texttt{=} \; \texttt{call} \; o \; \texttt{(} \; o_1 \; \ldots \; o_f \; \texttt{)} \;) & = & \{id\} \\
\mathit{defined}(\; \texttt{HeapAllocZ} \; \texttt{(} \; o \; \texttt{)} \;) & = & \emptyset \\
\mathit{defined}(\; \texttt{PrintIntS} \; \texttt{(} \; o \; \texttt{)} \;) & = & \emptyset \\
\mathit{defined}(\; \texttt{Error} \; \texttt{(} \; s \; \texttt{)} \;) & = & \emptyset
\end{array}
$$

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

$$
\mathit{initconst}(\texttt{const} \; l \; l_1 \; \ldots \; l_n \;) \;=\; [l \mapsto \langle l_1 \ldots \; l_n \rangle]
$$

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

$$
\begin{array}{rcl}
\mathit{initfun}(F) & = & [l \mapsto F, \; l_1 \mapsto b_1, \; \ldots, \; l_q \mapsto b_q] \\
\text{where} & & \\
F & = & \texttt{func} \; l \; (id_1 \; \ldots \; id_f) \; l_1 \; b_1 \; \ldots \; l_q \; b_q
\end{array}
$$

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.

$$
\begin{array}{rrcl}
(\mathit{GlobalData}) & d & ::= & t \mid F \mid b \\
(\mathit{GlobalTable}) & G & ::= & [l_1 \mapsto d_1, \ldots, l_n \mapsto d_n]
\end{array}
$$

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

$$C_1 \ldots\ C_n\ F_1 \ldots\ F_m$$

where

$$F_1 \quad = \quad \texttt{func}\ id\ \texttt{(}\ \texttt{)}\ l_1\ b_1\ \ \texttt{...}\ \ l_q\ b_q$$

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

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

$$
\begin{array}{rcl}
G & = & \mathit{initconst}(C_1) \cdot \ldots\ \cdot \mathit{initconst}(C_n) \cdot \mathit{initfun}(F_1) \cdot \ldots\ \cdot\ \mathit{initfun}(F_m) \\
H & = & [\,] \\
E & = & \mathit{initmap}(\ (\mathit{defined}(b_1) \cup \ldots\ \cup \mathit{defined}(b_q))\ ) \\
b & = & b_1
\end{array}
$$

heap is initially empty

environment contains all variables defined in all instruction blocks in the first function

program execution starts with the first block of the first function

3.4 Semantics

Single Steps.

$$(G, H, E, id\ \texttt{=}\ o \quad b') \longmapsto (G, H, E \cdot [id \mapsto E^*(o)], b')$$

$$\begin{array}{l}(G, H, E, id\ \texttt{=}\ \texttt{Add}\ \texttt{(}\ o_1\ \ o_2\ \texttt{)} \quad b') \longmapsto (G, H, E \cdot [id \mapsto (c_1 + c_2)], b') \\ \quad \text{if } E^*(o_1) = c_1 \text{ and } E^*(o_2) = c_2\end{array} \tag{2}$$

standard x = a + b behavior

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

$$\begin{array}{l}(G, H, E, id\ \texttt{=}\ \texttt{Sub}\ \texttt{(}\ o_1\ \ o_2\ \texttt{)} \quad b') \longmapsto (G, H, E \cdot [id \mapsto (c_1 - c_2)], b') \\ \quad \text{if } E^*(o_1) = c_1 \text{ and } E^*(o_2) = c_2\end{array} \tag{4}$$

standard x = a - b behavior

$$\begin{array}{l}(G, H, E, id\ \texttt{=}\ \texttt{MulS}\ \texttt{(}\ o_1\ \ o_2\ \texttt{)} \quad b') \longmapsto (G, H, E \cdot [id \mapsto (c_1 \times c_2)], b') \\ \quad \text{if } E^*(o_1) = c_1 \text{ and } E^*(o_2) = c_2\end{array} \tag{5}$$

standard x = a * b behavior

$$\begin{array}{l}(G, H, E, id\ \texttt{=}\ \texttt{Eq}\ \texttt{(}\ o_1\ \ o_2\ \texttt{)} \quad b') \longmapsto (G, H, E \cdot [id \mapsto 1], b') \\ \quad \text{if } E^*(o_1) = c_1 \text{ and } E^*(o_2) = c_2 \text{ and } c_1 = c_2\end{array} \tag{6}$$

$$\begin{array}{l}(G, H, E, id\ \texttt{=}\ \texttt{Eq}\ \texttt{(}\ o_1\ \ o_2\ \texttt{)} \quad b') \longmapsto (G, H, E \cdot [id \mapsto 0], b') \\ \quad \text{if } E^*(o_1) = c_1 \text{ and } E^*(o_2) = c_2 \text{ and } c_1 \neq c_2\end{array} \tag{7}$$

standard x==y behavior:
(x==y -> 1)
(x != y -> 0)

$$\begin{array}{l}(G, H, E, id\ \texttt{=}\ \texttt{LtS}\ \texttt{(}\ o_1\ \ o_2\ \texttt{)} \quad b') \longmapsto (G, H, E \cdot [id \mapsto 1], b') \\ \quad \text{if } E^*(o_1) = c_1 \text{ and } E^*(o_2) = c_2 \text{ and } c_1 < c_2\end{array} \tag{8}$$

$$\begin{array}{l}(G, H, E, id\ \texttt{=}\ \texttt{LtS}\ \texttt{(}\ o_1\ \ o_2\ \texttt{)} \quad b') \longmapsto (G, H, E \cdot [id \mapsto 0], b') \\ \quad \text{if } E^*(o_1) = c_1 \text{ and } E^*(o_2) = c_2 \text{ and } c_1 \geq c_2\end{array} \tag{9}$$

standard x<y behavior:
x<y --> 1
x>=y --> 0

$$\begin{array}{l}(G, H, E, id\ \texttt{=}\ \texttt{[}\ id'\ \texttt{+}\ \texttt{c}\ \texttt{]} \quad b') \longmapsto (G, H, E \cdot [id \mapsto v_{c'+c}], b') \\ \quad \text{where } E^*(id') = (l, c') \\ \quad \text{where } c \geq 0 \text{ and } c \text{ is divisible by } 4 \\ \quad \text{where } H(l) = \langle v_0, v_4, \ldots, v_{c'+c}, \ldots\ v_n \rangle \text{ or else } G(l) = \langle v_0, v_4, \ldots, v_{c'+c}, \ldots\ v_n \rangle \\ \quad \text{where } c' + c \leq n\end{array} \tag{10}$$

dereferencing a memory location either in the heap or in the global table:

$$\begin{array}{l}(G, H, E, \texttt{[}\ id\ \texttt{+}\ \texttt{c}\ \texttt{]}\ \texttt{=}\ id' \quad b') \longmapsto (G, H[l \mapsto t'], E, b') \\ \quad \text{where } E^*(id) = (l, c') \\ \quad \text{where } c \geq 0 \text{ and } c \text{ is divisible by } 4 \\ \quad \text{where } H(l) = \langle v_0, v_4, \ldots\ v_n \rangle \\ \quad \text{where } t' = \langle v_0, v_4, \ldots\ v_{c-4}\ E^*(id')\ v_{c+4} \ldots\ v_n \rangle \\ \quad \text{where } c' + c \leq n\end{array} \tag{11}$$

assigning a value to a variable stored in the heap

if0 jump statement.
can only jump to locations in the same function

$$(G, H, E, \texttt{if0}\ o\ \texttt{goto}\ l \quad b') \longmapsto (G, H, E, b'') \\ \quad \text{if } E^*(o) = 0 \text{ and } G(l) = b'', \\ \quad \text{and } l\ b'' \text{ and } \texttt{if0}\ o\ \texttt{goto}\ l \text{ are in the body of the same function} \tag{12}$$

$$(G, H, E, \texttt{if0}\ o\ \texttt{goto}\ l \quad b') \longmapsto (G, H, E, b') \\ \quad \text{if } E^*(o) = c \text{ and } c \neq 0 \tag{13}$$

$$\frac{(G, H, E^{init}, b_1) \longmapsto (G, H', E', \texttt{ret}\ o')}{(G, H, E, id\ \texttt{=}\ \texttt{call}\ o\ \texttt{(}\ o_1\ \texttt{...}\ o_f\ \texttt{)} \quad b') \longmapsto (G, H', E \cdot [id \mapsto E'^*(o')], b')} \\ \quad \text{where } E^*(o) = l \\ \quad \text{where } G(l) = \texttt{func}\ l\ \texttt{(}id_1\ \texttt{...}\ id_f\texttt{)}\ l_1\ b_1\ \texttt{...}\ l_q\ b_q \\ \quad \text{where } E^{init} = [id_1 \mapsto E^*(o_1), \ldots, id_f \mapsto E^*(o_f)] \cdot initmap(a) \\ \quad \text{where } a = (defined(b_1) \cup \ldots \cup defined(b_q)) \setminus \{id_1, \ldots, id_f\} \tag{14}$$

$$(G, H, E, id\ \texttt{=}\ \texttt{HeapAllocZ}\ \texttt{(}\ o\ \texttt{)} \quad b') \longmapsto (G, H \cup [l \mapsto t], E \cdot [id \mapsto (l, 0)], E, b') \\ \quad \text{where } l \notin dom(G, H) \text{ and } E^*(o) \text{ is a positive integer that is divisible by } 4 \\ \quad \text{where } t = inittuple(\tfrac{E^*(o)}{4}) \tag{15}$$

$$(G, H, E, \texttt{PrintIntS}\ \texttt{(}\ o\ \texttt{)} \quad b') \longmapsto (G, H, E, b') \\ \quad \text{where } E^*(o) = c \\ \quad \text{and display } c \text{ on the screen} \tag{16}$$

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

$$(G, H, E, \texttt{goto}\ l) \longmapsto (G, H, E, b') \\ \quad \text{if } G(l) = b', \\ \quad \text{and } l\ b' \text{ and } \texttt{goto}\ l \text{ are in the body of the same function} \tag{18}$$

Multiple Steps.

$$\frac{(G, H, E, b) \longmapsto (G', H', E', b') \qquad (G', H', E', b') \longmapsto (G'', H'', E'', b'')}{(G, H, E, b) \longmapsto (G'', H'', E'', b'')} \tag{19}$$

4 Vapor-M

4.1 Syntax

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

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.

$$defined(\ i_1 \ldots\ i_n\ j\ ) \quad = \quad (defined(i_1) \cup \ldots\ defined(i_n))$$

$$
\begin{array}{rcl}
defined(\ id\ \texttt{=}\ o\ ) & = & \{id\} \\
defined(\ id\ \texttt{=}\ op\ \texttt{(}\ o_1\ o_2\ \texttt{)}\ ) & = & \{id\} \\
defined(\ id\ \texttt{=}\ m\ ) & = & \{id\} \\
defined(\ m\ \texttt{=}\ id\ ) & = & \emptyset \\
defined(\ \texttt{if0}\ o\ \texttt{goto}\ l\ ) & = & \emptyset \\
defined(\ \texttt{call}\ o\ ) & = & \emptyset \\
defined(\ \texttt{HeapAllocZ (}\ o\ \texttt{)}\ ) & = & \emptyset \\
defined(\ \texttt{PrintIntS (}\ o\ \texttt{)}\ ) & = & \emptyset \\
defined(\ \texttt{Error (}\ s\ \texttt{)}\ ) & = & \emptyset
\end{array}
$$

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

$$initconst(\texttt{const}\ l\ l_1\ \ldots\ l_n\ ) \quad = \quad [l \mapsto \langle l_1 \ldots\ l_n \rangle]$$

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

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

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.

$$\begin{array}{rrcl} (\mathit{GlobalData}) & d & ::= & t \mid F \mid b \\ (\mathit{GlobalTable}) & G & ::= & [l_1 \mapsto d_1, \ldots, l_n \mapsto d_n] \end{array}$$

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.

$$\mathit{registers} \quad = \quad \{\texttt{s0}, \ldots, \texttt{s7}, \texttt{t0}, \ldots, \texttt{t8}, \texttt{a0}, \ldots, \texttt{a3}, \texttt{v0}, \texttt{v1}\}$$

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.

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

The Initial Program State. Consider a program

$$C_1 \ldots\ C_n\ F_1 \ldots\ F_m$$

where

$$F_1 \quad = \quad \texttt{func}\ \mathit{id}\ \texttt{[in 0, out}\ c_2\texttt{, local}\ c_3\texttt{]}\ l_1\ b_1\ \ldots\ l_q\ b_q$$

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

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

4.4 Semantics

Single Steps.

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

$$\begin{array}{l}(G, H, R, S, id \texttt{ = Add (} o_1 \ o_2 \texttt{)} \quad b') \longmapsto (G, H, R \cdot [id \mapsto (c_1 + c_2), S, b') \\ \qquad \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2\end{array} \tag{21}$$

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

$$\begin{array}{l}(G, H, R, S, id \texttt{ = Sub (} o_1 \ o_2 \texttt{)} \quad b') \longmapsto (G, H, R \cdot [id \mapsto (c_1 - c_2), S, b') \\ \qquad \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2\end{array} \tag{23}$$

$$\begin{array}{l}(G, H, R, S, id \texttt{ = MulS (} o_1 \ o_2 \texttt{)} \quad b') \longmapsto (G, H, R \cdot [id \mapsto (c_1 \times c_2), S, b') \\ \qquad \text{if } R^*(o_1) = c_1 \text{ and } R^*(o_2) = c_2\end{array} \tag{24}$$

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

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

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

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

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

$$\begin{array}{l}(G, H, R, S, id \texttt{ = in [} c \texttt{]} \quad b') \longmapsto (G, H, R \cdot [id \mapsto v_c], S, b') \\ \qquad \text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out} \\ \qquad \text{where } t_{in} = \langle v_m \ldots \ v_c \ldots \ v_1, v_0 \rangle\end{array} \tag{30}$$

$$\begin{array}{l}(G, H, R, S, id \texttt{ = out [} c \texttt{]} \quad b') \longmapsto (G, H, R \cdot [id \mapsto v_c], S, b') \\ \qquad \text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out} \\ \qquad \text{where } t_{out} = \langle v_m \ldots \ v_c \ldots \ v_1, v_0 \rangle\end{array} \tag{31}$$

$$\begin{array}{l}(G, H, R, S, id \texttt{ = local [} c \texttt{]} \quad b') \longmapsto (G, H, R \cdot [id \mapsto v_c], S, b') \\ \qquad \text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out} \\ \qquad \text{where } t_{local} = \langle v_m \ldots \ v_c \ldots \ v_1, v_0 \rangle\end{array} \tag{32}$$

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

$$(G, H, R, S, \texttt{in [} c \texttt{] = } id \quad b') \longmapsto (G, H, R, S \odot t'_{in} \odot t_{local} \odot t_{out}, b')$$
$$\text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out}$$
$$\text{where } t_{in} = \langle v_m \ldots\ v_c \ldots\ v_1, v_0 \rangle \tag{34}$$
$$\text{where } t'_{in} = \langle v_m \ldots\ R^*(id) \ldots\ v_1, v_0 \rangle$$

$$(G, H, R, S, \texttt{out [} c \texttt{] = } id \quad b') \longmapsto (G, H, R, S \odot t_{in} \odot t_{local} \odot t'_{out}, b')$$
$$\text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out}$$
$$\text{where } t_{out} = \langle v_m \ldots\ v_c \ldots\ v_1, v_0 \rangle \tag{35}$$
$$\text{where } t'_{out} = \langle v_m \ldots\ R^*(id) \ldots\ v_1, v_0 \rangle$$

$$(G, H, R, S, \texttt{local [} c \texttt{] = } id \quad b') \longmapsto (G, H, R, S \odot t_{in} \odot t'_{local} \odot t_{out}, b')$$
$$\text{where } S = S' \odot t_{in} \odot t_{local} \odot t_{out}$$
$$\text{where } t_{local} = \langle v_m \ldots\ v_c \ldots\ v_1, v_0 \rangle \tag{36}$$
$$\text{where } t'_{local} = \langle v_m \ldots\ R^*(id) \ldots\ v_1, v_0 \rangle$$

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

$$(G, H, R, S, \texttt{if0 } o \texttt{ goto } l \quad b') \longmapsto (G, H, R, S, b')$$
$$\text{if } R^*(o) = c \text{ and } c \neq 0 \tag{38}$$

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

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

$$(G, H, R, S, \texttt{PrintIntS (} o \texttt{)} \quad b') \longmapsto (G, H, R, S, b')$$
$$\text{where } R^*(o) = c \tag{41}$$
$$\text{and display } c \text{ on the screen}$$

$$(G, H, R, S, \texttt{Error (} s \texttt{)} \quad b')$$
$$\text{display } s \text{ on the screen and stop execution} \tag{42}$$

$$(G, H, R, S, \texttt{goto } l) \longmapsto (G, H, R, S, b')$$
$$\text{if } G(l) = b', \tag{43}$$
$$\text{and } l\ b' \text{ and } \texttt{goto } l \text{ and in the body of the same function}$$

Multiple Steps.

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