IR$_{\text{ES}}$: Intermediate Representation for ECMAScript Specifications

Seungmin An[*] Jihyeok Park[*] Sukyoung Ryu[*]

[*]KAIST, South Korea

1 Syntax of IR$_{\text{ES}}$

Programs	$P \ni p$	$::=$	i^+	
Instructions	$I \ni i$	$::=$	e	(expressions)
		$\mid$	$\texttt{let}\ x\ \texttt{=}\ e$	(let bindings)
		$\mid$	$r\ \texttt{:=}\ e$	(assignments)
		$\mid$	$\texttt{delete}\ r$	(deletions)
		$\mid$	$\texttt{append}\ e\ \leftarrow\ e$	(append instructions)
		$\mid$	$\texttt{prepend}\ e\ \rightarrow\ e$	(prepend instructions)
		$\mid$	$\texttt{return}\ e$	(return instructions)
		$\mid$	$\texttt{if}\ e\ i\ i$	(branches)
		$\mid$	$\texttt{while}\ e\ i$	(loops)
		$\mid$	$\texttt{\{}\ i^*\ \texttt{\}}$	(sequences)
		$\mid$	$\texttt{assert}\ e$	(assertions)
		$\mid$	$\texttt{print}\ e$	(print instructions)
		$\mid$	$\texttt{call}\ x\ \texttt{=}\ e\texttt{(}e^*\texttt{)}$	(function calls)
		$\mid$	$\texttt{access}\ x\ \texttt{=}\ e\texttt{[}e\texttt{]}$	(field accesses)
		$\mid$	$\texttt{withcont}\ x\texttt{(}x^*\texttt{)}\ \texttt{=}\ i$	(continuation bindings)
Expressions	$E \ni e$	$::=$	$d \mid n \mid s \mid b \mid \texttt{undefined} \mid \texttt{null} \mid \texttt{absent}$	(primitives)
		$\mid$	$\texttt{new}\ s\ \texttt{\{}[e \mapsto e]^*\texttt{\}}$	(maps)
		$\mid$	$\texttt{new [}e^*\texttt{]}$	(lists)
		$\mid$	$\texttt{new}\ e$	(symbols)
		$\mid$	$\texttt{pop}\ e\ e$	(pop expressions)
		$\mid$	r	(references)
		$\mid$	$\texttt{(}x^*\texttt{) =>}\ i$	(continuations)
		$\mid$	$\odot\ e$	(unary operations)
		$\mid$	$e \oplus e$	(binary operations)
		$\mid$	$\texttt{typeof}\ e$	(typeof expressions)
		$\mid$	$\texttt{is-completion}\ e$	(completion checks)
		$\mid$	$\texttt{is-instance-of}\ e\ s$	(instance checks)
		$\mid$	$\texttt{get-elems}\ e\ s$	(element getters)
		$\mid$	$\texttt{get-syntax}\ e$	(syntax getters)
		$\mid$	$\texttt{parse-syntax}\ e\ e\ e^*$	(parse expressions)
		$\mid$	$\texttt{convert}\ e\ \triangleright\ e^?$	(conversions)
		$\mid$	$\texttt{contains}\ e\ e$	(contain checks)
		$\mid$	$\texttt{copy}\ e$	(object copies)
		$\mid$	$\texttt{keys}\ e$	(key collections)
		$\mid$	$\texttt{!!!}\ e$	(not supported features)

$$
\begin{array}{llllll}
\text{References} & R & \ni & r & ::= & x & \text{(identifier references)} \\
 & & & & | & r\,\texttt{[}e\texttt{]} & \text{(field references)} \\[1em]
\text{Unary Operators} & & & \odot & ::= & \texttt{-} & \text{(negations)} \\
 & & & & | & \texttt{!} & \text{(logical NOT)} \\
 & & & & | & \texttt{\~{}} & \text{(bitwise NOT)} \\[1em]
\text{Binary Operators} & & & \oplus & ::= & \texttt{+} & \text{(additions)} \\
 & & & & | & \texttt{-} & \text{(subtractions)} \\
 & & & & | & \texttt{*} & \text{(multiplications)} \\
 & & & & | & \texttt{**} & \text{(exponentials)} \\
 & & & & | & \texttt{/} & \text{(divisions)} \\
 & & & & | & \texttt{\%\%} & \text{(unsigned modulos)} \\
 & & & & | & \texttt{\%} & \text{(modulos)} \\
 & & & & | & \texttt{eq} & \text{(strong equalities)} \\
 & & & & | & \texttt{=} & \text{(weak equalities)} \\
 & & & & | & \texttt{<} & \text{(comparisons)} \\
 & & & & | & \texttt{\&\&} & \text{(logical AND)} \\
 & & & & | & \texttt{||} & \text{(logical OR)} \\
 & & & & | & \texttt{\^{}\^{}} & \text{(logical XOR)} \\
 & & & & | & \texttt{\&} & \text{(bitwise AND)} \\
 & & & & | & \texttt{|} & \text{(bitwise OR)} \\
 & & & & | & \texttt{\^{}} & \text{(bitwise XOR)} \\
 & & & & | & \texttt{<<} & \text{(left shifts)} \\
 & & & & | & \texttt{>>} & \text{(signed right shifts)} \\
 & & & & | & \texttt{>>>} & \text{(unsigned right shifts)} \\[1em]
\text{Convert Operators} & & & \triangleright & ::= & \texttt{str2num} & \text{(strings to numbers)} \\
 & & & & | & \texttt{num2str} & \text{(numbers to strings)} \\
 & & & & | & \texttt{num2int} & \text{(numbers to integers)}
\end{array}
$$

where

$d \in \mathbb{V}_{\texttt{double}}$	double-precision 64-bit binary format IEEE 754-2008 values
$n \in \mathbb{V}_{\texttt{int}}$	mathematical integers
$s \in \mathbb{V}_{\texttt{str}}$	strings
$b \in \mathbb{V}_{\texttt{bool}}$	booleans
$x \in \mathbb{X}$	identifiers

2 Semantics of IR_{ES}

2.1 Notations

States	$(c, \bar{c}, \rho, h) = \sigma \in \mathbb{S}$	$= \mathbb{C} \times \mathbb{C}^* \times \mathbb{E} \times \mathbb{H}$
Contexts	$(x, \bar{i}, \rho) = c \in \mathbb{C}$	$= \mathbb{X} \times I^* \times \mathbb{E}$
Environments	$\rho \in \mathbb{E}$	$= \mathbb{X} \xrightarrow{\texttt{fin}} \mathbb{V}$
Heaps	$h \in \mathbb{H}$	$= \mathbb{A} \xrightarrow{\texttt{fin}} \mathbb{O}$
Values	$v \in \mathbb{V}$	
Addresses	$a \in \mathbb{A}$	
Objects	$o \in \mathbb{O}$	
Reference Values	$v^r \in \mathbb{V}_r$	

Values	$\mathbb{V}$	$\ni$ v	$::=$	$d \mid n \mid s \mid b \mid \texttt{undefined} \mid \texttt{null} \mid \texttt{absent}$	(primitives)
			$\mid$	a	(addresses)
			$\mid$	⌃	(ECMAScript ASTs)
			$\mid$	$\langle \lambda\texttt{(}x^*[, {*}x]^?\texttt{)}.\ i, \rho \rangle$	(closures)
			$\mid$	$\langle \kappa\texttt{(}x^*\texttt{)}.\ i, c, \bar{c} \rangle$	(continuations)
Objects	$\mathbb{O}$	$\ni$ o	$::=$	$s\ \texttt{\{}[v \mapsto v]^*\texttt{\}}$	(maps)
			$\mid$	$\texttt{[}v^*\texttt{]}$	(lists)
			$\mid$	$\texttt{symbol}\ v$	(symbols)
Reference Values	$\mathbb{V}_r$	$\ni$ v^r	$::=$	x	(identifiers)
			$\mid$	$a\texttt{[}v\texttt{]}$	(address fields)
			$\mid$	$s\texttt{[}v\texttt{]}$	(string fields)

2.2 Semantics of Programs

The semantics of an IR_{ES} program p is defined with a state transition system $(\mathbb{S}, \rightsquigarrow, \sigma_\iota)$. The transition relation $\rightsquigarrow \subseteq \mathbb{S} \times \mathbb{S}$ describes how states are transformed into other states as follows:

$$\frac{\begin{array}{c}\sigma = (c, \_, \_, \_) \quad c = (\_, \bar{i} = \langle i_0, i_1, \cdots, i_n \rangle, \_) \\ c' = c[\bar{i}/\langle i_1, \cdots, i_n \rangle] \quad \sigma' = \sigma[c/c'] \quad \sigma' \vdash i_0 \Rightarrow \sigma''\end{array}}{\sigma \rightsquigarrow \sigma''}$$

where $x[y/z]$ denotes substituting y in x with z. The notation $\rightsquigarrow^*$ is zero or more repetitions of the transition relation $\rightsquigarrow$. The initial state σ_ι is defined as follows:

$$\begin{aligned}
\sigma_\iota &= (c_\iota, \epsilon, \rho_\iota, h_\iota) \\
c_\iota &= (\texttt{RET}, p, \epsilon) \\
\rho_\iota &= \text{an initial global environment given by } \textsf{JISET}. \\
h_\iota &= \text{an initial heap given by } \textsf{JISET}. \\
p &= \text{a given program.} \\
\texttt{RET} &= \text{a special identifier for return instructions.}
\end{aligned}$$

The collecting semantics $[\![p]\!]$ of the program p is defined as follows:

$$[\![p]\!] = \{\sigma \mid \sigma_\iota \rightsquigarrow^* \sigma\}$$

Now, we define the operational semantics of each IR_{ES} component: (instructions in Section 2.3, expressions in Section 2.4, references in Section 2.5, and reference values in Section 2.6. We utilize several helper functions defined in Section 2.7.

2.3 Semantics of Instructions: $\boxed{\sigma \vdash i \Rightarrow \sigma}$

- expressions:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma_0}{\sigma \vdash e \Rightarrow \sigma_0}$$

- let bindings:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad \sigma_1 = \texttt{Define}(\sigma_0, x, v)}{\sigma \vdash \texttt{let}\ x\ \texttt{=}\ e \Rightarrow \sigma_1}$$

- assignments:

$$\frac{\sigma \vdash r \Rightarrow v^r,\ \sigma_0 \quad \sigma_0 \vdash e \Rightarrow v,\ \sigma_1 \quad \sigma_2 = \texttt{Updated}(\sigma_1, v^r, v)}{\sigma \vdash r\ \texttt{:=}\ e \Rightarrow \sigma_2}$$

- deletions:

$$\frac{\sigma \vdash r \Rightarrow v^r,\ \sigma_0 \quad \sigma_1 = \texttt{Deleted}(\sigma_0, v^r)}{\sigma \vdash \texttt{delete}\ r \Rightarrow \sigma_1}$$

- append instructions:

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad a = \texttt{Escape}(v_0, \sigma_0) \\ \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad v_2 = \texttt{Escape}(v_1, \sigma_1) \quad \sigma_2 = \texttt{Append}(\sigma_1, a, v_2)\end{array}}{\sigma \vdash \texttt{append}\ e_0\ \leftarrow\ e_1 \Rightarrow \sigma_2}$$

- prepend instructions:

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad v_1 = \texttt{Escape}(v_0, \sigma_0) \\ \sigma_0 \vdash e_1 \Rightarrow v_2,\ \sigma_1 \quad a = \texttt{Escape}(v_2, \sigma_1) \quad \sigma_2 = \texttt{Prepend}(\sigma_1, a, v_1)\end{array}}{\sigma \vdash \texttt{prepend}\ e_0\ \rightarrow\ e_1 \Rightarrow \sigma_2}$$

- return instructions:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad \sigma_1 = \texttt{Return}(\sigma_0, v)}{\sigma \vdash \texttt{return}\ e \Rightarrow \sigma_1}$$

- branches:

$$\frac{\begin{array}{c}\sigma_0 \vdash e \Rightarrow v,\ \sigma_0 \quad \texttt{true} = \texttt{Escape}(v, \sigma_0) \quad \sigma_0 = (c_0, \_, \_, \_) \\ c_0 = (\_, \bar{i} = \langle i_0, \cdots, i_n \rangle, \_) \quad c_1 = c_0[\bar{i}/\langle i_{\texttt{then}}, i_0, \cdots, i_n \rangle] \quad \sigma_1 = \sigma_0[c_0/c_1]\end{array}}{\sigma \vdash \texttt{if}\ e\ i_{\texttt{then}}\ i_{\texttt{else}} \Rightarrow \sigma_1}$$

$$\frac{\begin{array}{c}\sigma_0 \vdash e \Rightarrow v,\ \sigma_0 \quad \texttt{false} = \texttt{Escape}(v, \sigma_0) \quad \sigma_0 = (c_0, \_, \_, \_) \\ c_0 = (\_, \bar{i} = \langle i_0, \cdots, i_n \rangle, \_) \quad c_1 = c_0[\bar{i}/\langle i_{\texttt{else}}, i_0, \cdots, i_n \rangle] \quad \sigma_1 = \sigma_0[c_0/c_1]\end{array}}{\sigma \vdash \texttt{if}\ e\ i_{\texttt{then}}\ i_{\texttt{else}} \Rightarrow \sigma_1}$$

- loops:

$$\frac{\begin{array}{c}\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad \texttt{true} = \texttt{Escape}(v, \sigma_0) \quad \sigma_0 = (c_0, \_, \_, \_) \\ c_0 = (\_, \bar{i} = \langle i_0, \cdots, i_n \rangle, \_) \quad c_1 = c_0[\bar{i}/\langle i, \texttt{while}\ e\ i, i_0, \cdots, i_n \rangle] \quad \sigma_1 = \sigma_0[c_0/c_1]\end{array}}{\sigma \vdash \texttt{while}\ e\ i \Rightarrow \sigma_1}$$

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad \texttt{false} = \texttt{Escape}(v, \sigma_0)}{\sigma \vdash \texttt{while}\ e\ i \Rightarrow \sigma_0}$$

- sequences:

$$\frac{\begin{array}{c}\sigma = (c, \_, \_, \_) \\ c = (\_, \overline{i}' = \langle i'_0, \cdots, i'_m \rangle, \_) \quad c_0 = c[\overline{i}'/\langle i_0, \cdots, i_n, i'_0, \cdots, i'_m \rangle] \quad \sigma_0 = \sigma[c/c_0]\end{array}}{\sigma \vdash \texttt{\{ } i_0 \cdots i_n \texttt{ \}} \Rightarrow \sigma_0}$$

- assertions:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad \texttt{true} = \texttt{Escape}(v, \sigma_0)}{\sigma \vdash \texttt{assert } e \Rightarrow \sigma_0}$$

- print instructions:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad \texttt{Print}(v)}{\sigma \vdash \texttt{print } e \Rightarrow \sigma_0}$$

- function calls:

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow \langle \lambda(x_1, \cdots, x_m).\ i_{\texttt{body}}, \rho \rangle,\ \sigma_0 \\ \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \quad n < m \\ \rho_0 = \rho[x_1 \mapsto v_1, \cdots, x_n \mapsto v_n, x_{n+1} \mapsto \texttt{absent}, \cdots, x_m \mapsto \texttt{absent}] \\ \sigma_n = (c, \overline{c}' = \langle c'_0, \cdots, c'_k \rangle, \_, \_) \quad c = (x_{\texttt{ret}}, \_, \_) \\ c_0 = c[x_{\texttt{ret}}/x] \quad c_1 = (\texttt{RET}, \langle i_{\texttt{body}} \rangle, \rho_0) \quad \sigma' = \sigma_n[c/c_1][\overline{c}'/\langle c_0, c'_0, \cdots, c'_k \rangle]\end{array}}{\sigma \vdash \texttt{call } x \texttt{ = } e_0 \texttt{(} e_1, \cdots, e_n \texttt{)} \Rightarrow \sigma'}$$

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow \langle \lambda(x_1, \cdots, x_m).\ i_{\texttt{body}}, \rho \rangle,\ \sigma_0 \\ \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \quad n \geq m \\ \rho_0 = \rho[x_1 \mapsto v_1, \cdots, x_m \mapsto v_m] \\ \sigma_n = (c, \overline{c}' = \langle c'_0, \cdots, c'_k \rangle, \_, \_) \quad c = (x_{\texttt{ret}}, \_, \_) \\ c_0 = c[x_{\texttt{ret}}/x] \quad c_1 = (\texttt{RET}, \langle i_{\texttt{body}} \rangle, \rho_0) \quad \sigma' = \sigma_n[c/c_1][\overline{c}'/\langle c_0, c'_0, \cdots, c'_k \rangle]\end{array}}{\sigma \vdash \texttt{call } x \texttt{ = } e_0 \texttt{(} e_1, \cdots, e_n \texttt{)} \Rightarrow \sigma'}$$

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow \langle \lambda(x_1, \cdots, x_m, *x').\ i_{\texttt{body}}, \rho \rangle,\ \sigma_0 \\ \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \quad n < m \\ \rho_0 = \rho[x_1 \mapsto v_1, \cdots, x_n \mapsto v_n, x_{n+1} \mapsto \texttt{absent}, \cdots, x_m \mapsto \texttt{absent}] \quad \rho_1 = \rho_0[x' \mapsto \texttt{[]}] \\ \sigma_n = (c, \overline{c}' = \langle c'_0, \cdots, c'_k \rangle, \_, \_) \quad c = (x_{\texttt{ret}}, \_, \_) \\ c_0 = c[x_{\texttt{ret}}/x] \quad c_1 = (\texttt{RET}, \langle i_{\texttt{body}} \rangle, \rho_1) \quad \sigma' = \sigma_n[c/c_1][\overline{c}'/\langle c_0, c'_0, \cdots, c'_k \rangle]\end{array}}{\sigma \vdash \texttt{call } x \texttt{ = } e_0 \texttt{(} e_1, \cdots, e_n \texttt{)} \Rightarrow \sigma'}$$

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow \langle \lambda(x_1, \cdots, x_m, *x').\ i_{\texttt{body}}, \rho \rangle,\ \sigma_0 \\ \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \quad n \geq m \\ \rho_0 = \rho[x_1 \mapsto v_1, \cdots, x_m \mapsto v_m] \quad \rho_1 = \rho_0[x' \mapsto \texttt{[}v_{m+1}, \cdots, v_n\texttt{]}] \\ \sigma_n = (c, \overline{c}' = \langle c'_0, \cdots, c'_k \rangle, \_, \_) \quad c = (x_{\texttt{ret}}, \_, \_) \\ c_0 = c[x_{\texttt{ret}}/x] \quad c_1 = (\texttt{RET}, \langle i_{\texttt{body}} \rangle, \rho_1) \quad \sigma' = \sigma_n[c/c_1][\overline{c}'/\langle c_0, c'_0, \cdots, c'_k \rangle]\end{array}}{\sigma \vdash \texttt{call } x \texttt{ = } e_0 \texttt{(} e_1, \cdots, e_n \texttt{)} \Rightarrow \sigma'}$$

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow \langle \kappa(x_1, \cdots, x_m).\ i_{\texttt{body}}, c, \overline{c} \rangle,\ \sigma_0 \\ \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \quad n < m \\ \rho_0 = \rho[x_1 \mapsto v_1, \cdots, x_n \mapsto v_n, x_{n+1} \mapsto \texttt{absent}, \cdots, x_m \mapsto \texttt{absent}] \\ \sigma_n = (c', \overline{c}', \_, \_) \quad c = (\_, \overline{i}, \rho) \quad c_0 = c[\overline{i}/\langle i_{\texttt{body}} \rangle][\rho/\rho_0] \quad \sigma' = \sigma_n[c'/c_0][\overline{c}'/\overline{c}]\end{array}}{\sigma \vdash \texttt{call } x \texttt{ = } e_0 \texttt{(} e_1, \cdots, e_n \texttt{)} \Rightarrow \sigma'}$$

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow \langle \kappa(x_1, \cdots, x_m).\ i_{\texttt{body}}, c, \overline{c} \rangle,\ \sigma_0 \\ \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \quad n \geq m \\ \rho_0 = \rho[x_1 \mapsto v_1, \cdots, x_m \mapsto v_m] \\ \sigma_n = (c', \overline{c}', \_, \_) \quad c = (\_, \overline{i}, \rho) \quad c_0 = c[\overline{i}/\langle i_{\texttt{body}} \rangle][\rho/\rho_0] \quad \sigma' = \sigma_n[c'/c_0][\overline{c}'/\overline{c}]\end{array}}{\sigma \vdash \texttt{call } x \texttt{ = } e_0 \texttt{(} e_1, \cdots, e_n \texttt{)} \Rightarrow \sigma'}$$

- field accesses:

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad a = \mathtt{Escape}(v_0, \sigma_0) \quad \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad v_2 = \mathtt{Escape}(v_1, \sigma_1) \\ v' = \mathtt{GetAddrField}(\sigma_1, a, v_2) \quad \sigma_2 = \mathtt{Define}(\sigma_1, x, v')\end{array}}{\sigma \vdash \mathtt{access}\ x\ \mathtt{=}\ e_0\mathtt{[}e_1\mathtt{]} \Rightarrow \sigma_2}$$

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad s = \mathtt{Escape}(v_0, \sigma_0) \quad \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad v_2 = \mathtt{Escape}(v_1, \sigma_1) \\ v' = \mathtt{GetStringField}(s, v_2) \quad \sigma_2 = \mathtt{Define}(\sigma_1, x, v')\end{array}}{\sigma \vdash \mathtt{access}\ x\ \mathtt{=}\ e_0\mathtt{[}e_1\mathtt{]} \Rightarrow \sigma_2}$$

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad \dot{\wedge} = \mathtt{Escape}(v_0, \sigma_0) \quad \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad v_2 = \mathtt{Escape}(v_1, \sigma_1) \\ v' = \mathtt{GetASTField}(\dot{\wedge}, v_2) \quad \sigma_2 = \mathtt{Define}(\sigma_1, x, v')\end{array}}{\sigma \vdash \mathtt{access}\ x\ \mathtt{=}\ e_0\mathtt{[}e_1\mathtt{]} \Rightarrow \sigma_2}$$

- continuation bindings:

$$\frac{\sigma = (c, \bar{c}, \_, \_) \quad \sigma_0 = \mathtt{Define}(\sigma, x_0, \langle \kappa\mathtt{(}x_1, \cdots, x_n\mathtt{)}.\ i, c, \bar{c}\rangle)}{\sigma \vdash \mathtt{withcont}\ x_0\mathtt{(}x_1, \cdots, x_n\mathtt{)}\ \mathtt{=}\ i \Rightarrow \sigma_0}$$

2.4 Semantics of Expressions: $\boxed{\sigma \vdash e \Rightarrow v,\ \sigma}$

- primitives:

$$\sigma \vdash d \Rightarrow d,\ \sigma \quad \sigma \vdash n \Rightarrow n,\ \sigma \quad \sigma \vdash s \Rightarrow s,\ \sigma \quad \sigma \vdash b \Rightarrow b,\ \sigma$$

$$\sigma \vdash \texttt{undefined} \Rightarrow \texttt{undefined},\ \sigma \quad \sigma \vdash \texttt{null} \Rightarrow \texttt{null},\ \sigma \quad \sigma \vdash \texttt{absent} \Rightarrow \texttt{absent},\ \sigma$$

- maps:

$$\frac{\begin{array}{c}(a, \sigma_0) = \texttt{AllocMap}(\sigma, s)\\ \sigma_0 \vdash e_{k_1} \Rightarrow v_{k_1},\ \sigma_{k_1} \quad v'_{k_1} = \texttt{Escape}(v_{k_1}, \sigma_{k_1})\\ \sigma_{k_1} \vdash e_{v_1} \Rightarrow v_{v_1},\ \sigma_{v_1} \quad \sigma_1 = \texttt{Updated}(\sigma_{v_1}, a\texttt{[}v'_{k_1}\texttt{]}, v_{v_1})\\ \cdots\\ \sigma_{n-1} \vdash e_{k_n} \Rightarrow v_{k_n},\ \sigma_{k_n} \quad v'_{k_n} = \texttt{Escape}(v_{k_n}, \sigma_{k_n})\\ \sigma_{k_n} \vdash e_{v_n} \Rightarrow v_{v_n},\ \sigma_{v_n} \quad \sigma_n = \texttt{Updated}(\sigma_{v_n}, a\texttt{[}v'_{k_n}\texttt{]}, v_{v_n})\end{array}}{\sigma \vdash \texttt{new}\ s\ \texttt{\{}e_{k_1} \mapsto e_{v_1}, \cdots, e_{k_n} \mapsto e_{v_n}\texttt{\}} \Rightarrow a,\ \sigma_n}$$

- lists:

$$\frac{\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \quad (a, \sigma') = \texttt{AllocList}(\sigma_n, \langle v_0, \cdots, v_n\rangle)}{\sigma \vdash \texttt{new [}e_0, \cdots, e_n\texttt{]} \Rightarrow a,\ \sigma'}$$

- symbols:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad v' = \texttt{Escape}(v, \sigma_0) \quad (a, \sigma') = \texttt{AllocSymbol}(\sigma_0, v')}{\sigma \vdash \texttt{new}\ e \Rightarrow a,\ \sigma'}$$

- pop expressions:

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad a = \texttt{Escape}(v_0, \sigma_0)\\ \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad n = \texttt{Escape}(v_1, \sigma_1) \quad (v', \sigma') = \texttt{Pop}(\sigma_1, a, n)\end{array}}{\sigma \vdash \texttt{pop}\ e_0\ e_1 \Rightarrow v',\ \sigma'}$$

- references:

$$\frac{\sigma \vdash r \Rightarrow v^r,\ \sigma_0 \quad \sigma_0 \vdash v^r \Rightarrow v,\ \sigma_1}{\sigma \vdash r \Rightarrow v,\ \sigma_1}$$

- continuations:

$$\frac{\sigma = (c, \bar{c}, \_, \_)}{\sigma \vdash \texttt{(}x_0, \cdots, x_n\texttt{) =>}\ i \Rightarrow \langle \kappa\texttt{(}x_0, \cdots, x_n\texttt{)}.\ i, c, \bar{c}\rangle,\ \sigma}$$

- unary operations:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma'}{\sigma \vdash \odot\ e \Rightarrow \odot\ v,\ \sigma'}$$

- binary operations:

$$\frac{\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1}{\sigma \vdash e_0 \oplus e_1 \Rightarrow v_0 \oplus v_1,\ \sigma_1}$$

- typeof expressions:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma' \quad s = \texttt{GetType}(\sigma', v)}{\sigma \vdash \texttt{typeof}\ e \Rightarrow s,\ \sigma'}$$

- completion checks:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma' \quad b = \texttt{IsCompletion}(\sigma', v)}{\sigma \vdash \texttt{is-completion}\ e \Rightarrow b,\ \sigma'}$$

- instance checks:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma' \quad \curlywedge = \texttt{Escape}(v, \sigma') \quad b = \texttt{IsInstanceOf}(\curlywedge, s)}{\sigma \vdash \texttt{is-instance-of}\ e\ s \Rightarrow b,\ \sigma'}$$

- element getters:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad \curlywedge = \texttt{Escape}(v, \sigma_0) \quad (a, \sigma_1) = \texttt{GetElems}(\sigma_0, \curlywedge, s)}{\sigma \vdash \texttt{get-elems}\ e\ s \Rightarrow a,\ \sigma_1}$$

- syntax getters:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma' \quad \curlywedge = \texttt{Escape}(v, \sigma') \quad s = \texttt{GetSyntax}(\curlywedge)}{\sigma \vdash \texttt{get-syntax}\ e \Rightarrow s,\ \sigma'}$$

- parse expressions:

$$\frac{\begin{array}{c}\sigma \vdash e_{\texttt{code}} \Rightarrow v_{\texttt{code}},\ \sigma_0 \quad v = \texttt{Escape}(v_{\texttt{code}}, \sigma_0) \quad \sigma_0 \vdash e_{\texttt{rule}} \Rightarrow v_{\texttt{rule}},\ \sigma_1 \quad s = \texttt{Escape}(v_{\texttt{rule}}, \sigma_1) \\ \sigma_1 \vdash e_1 \Rightarrow b_1,\ \sigma_2 \quad \cdots \quad \sigma_n \vdash e_n \Rightarrow b_n,\ \sigma' \quad \curlywedge = \texttt{Parse}(v, s, \langle b_1, \cdots, b_n \rangle)\end{array}}{\sigma \vdash \texttt{parse-syntax}\ e_{\texttt{code}}\ e_{\texttt{rule}}\ e_1 \cdots e_n \Rightarrow \curlywedge,\ \sigma'}$$

- conversions:

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad v_0' = \texttt{Escape}(v_0, \sigma_0) \quad \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad v_1' = \texttt{Escape}(v_1, \sigma_1) \\ s = \texttt{Convert}(\texttt{num2str}, v_0', v_1')\end{array}}{\sigma \vdash \texttt{convert}\ e_0\ \texttt{num2str}\ e_1 \Rightarrow s,\ \sigma_1}$$

$$\frac{\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad v_1 = \texttt{Escape}(v_0, \sigma_0) \quad \triangleright \neq \texttt{num2str} \quad v = \texttt{Convert}(\triangleright, v_1, \texttt{absent})}{\sigma \vdash \texttt{convert}\ e_0\ \triangleright \Rightarrow v,\ \sigma_0}$$

- contain checks:

$$\frac{\begin{array}{c}\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad a = \texttt{Escape}(v_0, \sigma_0) \quad \sigma_0 \vdash e_1 \Rightarrow v_1,\ \sigma_1 \quad v = \texttt{Escape}(v_1, \sigma_1) \\ b = \texttt{Contains}(\sigma_1, a, v)\end{array}}{\sigma \vdash \texttt{contains}\ e_0\ e_1 \Rightarrow b,\ \sigma_1}$$

- object copies:

$$\frac{\begin{array}{c}\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad a = \texttt{Escape}(v, \sigma_0) \quad \sigma_0 = (\_, \_, \_, h) \quad a' \notin \texttt{Domain}(h) \\ h' = h[a' \mapsto h(a)] \quad \sigma' = \sigma_0[h/h']\end{array}}{\sigma \vdash \texttt{copy}\ e \Rightarrow a',\ \sigma'}$$

- key collections:

$$\frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad a = \texttt{Escape}(v, \sigma_0) \quad (a', \sigma') = \texttt{Keys}(\sigma_0, a)}{\sigma \vdash \texttt{keys}\ e \Rightarrow a',\ \sigma'}$$

2.5 Semantics of References: $\boxed{\sigma \vdash r \Rightarrow v^r,\ \sigma}$

- identifier references:

$$\sigma \vdash x \Rightarrow x,\ \sigma$$

- field references:

$$\frac{\begin{array}{c}\sigma \vdash r \Rightarrow v^r,\ \sigma_0 \quad \sigma_0 \vdash v^r \Rightarrow v_0,\ \sigma_1 \quad a = \texttt{Escape}(v_0, \sigma_1) \\ \sigma_1 \vdash e \Rightarrow v_1,\ \sigma_2 \quad v = \texttt{Escape}(v_1, \sigma_2)\end{array}}{\sigma \vdash r\texttt{[}e\texttt{]} \Rightarrow a\texttt{[}v\texttt{]},\ \sigma_2}$$

$$\frac{\begin{array}{c}\sigma \vdash r \Rightarrow v^r,\ \sigma_0 \quad \sigma_0 \vdash v^r \Rightarrow v_0,\ \sigma_1 \quad s = \texttt{Escape}(v_0, \sigma_1) \\ \sigma_1 \vdash e \Rightarrow v_1,\ \sigma_2 \quad v = \texttt{Escape}(v_1, \sigma_2)\end{array}}{\sigma \vdash r\texttt{[}e\texttt{]} \Rightarrow s\texttt{[}v\texttt{]},\ \sigma_2}$$

2.6 Semantics of Reference Values: $\boxed{\sigma \vdash v^r \Rightarrow v,\ \sigma}$

- identifiers:

$$\frac{v = \texttt{Lookup}(\sigma, x)}{\sigma \vdash x \Rightarrow v,\ \sigma}$$

- address fields:

$$\frac{v' = \texttt{GetAddrField}(\sigma, a, v)}{\sigma \vdash a\texttt{[}v\texttt{]} \Rightarrow v',\ \sigma}$$

- string fields:

$$\frac{v' = \texttt{GetStringField}(s, v)}{\sigma \vdash s\texttt{[}v\texttt{]} \Rightarrow v',\ \sigma}$$

2.7 Helper Functions

$$\mathtt{Escape}(v,\sigma) = \begin{cases} v' & \text{if } v = a \wedge \mathtt{Get}(\sigma,a) = o \wedge o = \mathtt{"Completion"}\ \mathtt{\{}\cdots, \mathtt{"Value"} \mapsto v', \cdots\mathtt{\}} \\ v & \text{otherwise} \end{cases}$$

$$\mathtt{Get}(\sigma,a) = \begin{cases} o & \text{if } \sigma = (\_,\_,\_,h) \wedge h(a) = o \\ \bot & \text{otherwise} \end{cases}$$

$$\mathtt{Set}(\sigma,a,o) = \begin{cases} \sigma' & \text{if } \sigma = (\_,\_,\_,h) \wedge h' = h[a \mapsto o] \wedge \sigma' = \sigma[h/h'] \\ \bot & \text{otherwise} \end{cases}$$

$$\mathtt{Define}(\sigma,x,v) = \sigma' \text{ where } \begin{cases} \sigma = (c,\_,\_,\_) \wedge c = (\_,\_,\rho) \wedge \\ \rho' = \rho[x \mapsto v] \wedge c' = c[\rho/\rho'] \wedge \sigma' = \sigma[c/c'] \end{cases}$$

$$\mathtt{Updated}(\sigma,v^r,v) = \begin{cases} \sigma' & \text{if } v^r = x \wedge \sigma = (\_,\_,\rho,\_) \wedge x \in \mathtt{Domain}(\rho) \wedge \rho' = \rho[x \mapsto v] \wedge \sigma' = \sigma[\rho/\rho'] \\ \sigma' & \text{if } v^r = x \wedge \sigma = (\_,\_,\rho,\_) \wedge x \notin \mathtt{Domain}(\rho) \wedge \sigma' = \mathtt{Define}(\sigma,x,v) \\ \sigma' & \text{if } v^r = a\mathtt{[}v'\mathtt{]} \wedge \sigma = (\_,\_,\_,h) \wedge h(a) = o \wedge o = s\ \mathtt{\{}\cdots\mathtt{\}} \wedge \\ & \quad o' = o[v' \mapsto v] \wedge h' = h[a \mapsto o'] \wedge \sigma' = \sigma[h/h'] \\ \bot & \text{otherwise} \end{cases}$$

$$\mathtt{Deleted}(\sigma,v^r) = \begin{cases} \sigma' & \text{if } v^r = x \wedge \sigma = (c,\_,\_,\_) \wedge c = (\_,\_,\rho) \\ & \quad \rho' = \rho - x \wedge c' = c[\rho/\rho'] \wedge \sigma' = \sigma[c/c'] \\ \sigma' & \text{if } v^r = a\mathtt{[}v\mathtt{]} \wedge \mathtt{Get}(\sigma,a) = o \wedge o = s\ \mathtt{\{}\cdots\mathtt{\}} \wedge \\ & \quad o' = o - v \wedge \sigma' = \mathtt{Set}(\sigma,a,o') \\ \bot & \text{otherwise} \end{cases}$$

$$\mathtt{Append}(\sigma,a,v) = \begin{cases} \sigma' & \text{if } \mathtt{Get}(\sigma,a) = \mathtt{[}v_1,\cdots,v_n\mathtt{]} \wedge o = \mathtt{[}v_1,\cdots,v_n,v\mathtt{]} \wedge \sigma' = \mathtt{Set}(\sigma,a,o) \\ \bot & \text{otherwise} \end{cases}$$

$$\mathtt{Prepend}(\sigma,a,v) = \begin{cases} \sigma' & \text{if } \mathtt{Get}(\sigma,a) = \mathtt{[}v_1,\cdots,v_n\mathtt{]} \wedge o = \mathtt{[}v,v_1,\cdots,v_n\mathtt{]} \wedge \sigma' = \mathtt{Set}(\sigma,a,o) \\ \bot & \text{otherwise} \end{cases}$$

$$\mathtt{Return}(\sigma,v) = \begin{cases} \sigma'' & \text{if } \sigma = (c, \overline{c} = \langle c_0,\cdots,c_n\rangle, \_,\_) \wedge c_0 = (x,\_,\_) \\ & \quad \sigma' = \mathtt{Define}(\sigma,x,v) \wedge \sigma'' = \sigma'[\overline{c}/\langle c_1,\cdots,c_n\rangle] \\ \bot & \text{otherwise} \end{cases}$$

$$\mathtt{Print}(v) = \text{print the given value } v$$

$$\mathtt{GetAddrField}(\sigma,a,v) = \begin{cases} v' & \text{if } \mathtt{Get}(\sigma,a) = o = s\ \mathtt{\{}\cdots\mathtt{\}} \wedge v \in \mathtt{Domain}(o) \wedge v' = o(v) \\ \mathtt{absent} & \text{if } \mathtt{Get}(\sigma,a) = o = s\ \mathtt{\{}\cdots\mathtt{\}} \wedge v \notin \mathtt{Domain}(o) \\ v_n & \text{if } \mathtt{Get}(\sigma,a) = o = \mathtt{[}v_0,\cdots,v_{m-1}\mathtt{]} \wedge v = n \wedge 0 \le n < m \\ \mathtt{absent} & \text{if } \mathtt{Get}(\sigma,a) = o = \mathtt{[}v_0,\cdots,v_{m-1}\mathtt{]} \wedge v = n \wedge (n < 0 \vee m \le n) \\ m & \text{if } \mathtt{Get}(\sigma,a) = o = \mathtt{[}v_0,\cdots,v_{m-1}\mathtt{]} \wedge v = \mathtt{"length"} \\ v' & \text{if } \mathtt{Get}(\sigma,a) = o = \mathtt{symbol}\ v' \wedge v = \mathtt{"Description"} \\ \bot & \text{otherwise} \end{cases}$$

$$\mathtt{GetStringField}(s,v) = \begin{cases} n & \text{if } v = \mathtt{"length"} \wedge n = (\text{the length of } s) \\ s' & \text{if } v = d \wedge n = (\text{the corresponding integer value of } d) \wedge \\ & \quad s' = (\text{a string consisting of only the } n\text{'th character of } s) \\ s' & \text{if } v = n \wedge s' = (\text{a string consisting of only the } n\text{'th character of } s) \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{GetASTField}(\curlywedge, v) = \begin{cases} v' & \text{if } v = s \land v' = (\curlywedge\text{'s member of name } s\text{, which is unique}) \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{AllocMap}(\sigma, s) = (a, \sigma') \text{ where } \begin{cases} a = (\text{a new address not in } \sigma) \\ \sigma' = \texttt{Set}(\sigma, a, s\ \texttt{\{\}}) \end{cases}$$

$$\texttt{AllocList}(\sigma, \langle v_1, \cdots, v_n \rangle) = (a, \sigma') \text{ where } \begin{cases} a = (\text{a new address not in } \sigma) \\ \sigma' = \texttt{Set}(\sigma, a, \texttt{[}v_1, \cdots, v_n\texttt{]}) \end{cases}$$

$$\texttt{AllocSymbol}(\sigma, v) = (a, \sigma') \text{ where } \begin{cases} a = (\text{a new address not in } \sigma) \\ \sigma' = \texttt{Set}(\sigma, a, \texttt{symbol}\ v) \end{cases}$$

$$\texttt{Pop}(\sigma, a, n) = \begin{cases} (v_n, \sigma') & \text{if } \texttt{Get}(\sigma, a) = o = \texttt{[}v_0, \cdots, v_{m-1}\texttt{]} \land 0 \leq n < m \land \\ & \quad o' = \texttt{[}v_0, \cdots, v_{n-1}, v_{n+1}, \cdots, v_{m-1}\texttt{]} \land \sigma' = \texttt{Set}(\sigma, a, o') \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{GetType}(\sigma, v) = \begin{cases} \texttt{"Number"} & \text{if } v = d \lor v = n \\ \texttt{"String"} & \text{if } v = s \\ \texttt{"Boolean"} & \text{if } v = b \\ \texttt{"Undefined"} & \text{if } v = \texttt{undefined} \\ \texttt{"Null"} & \text{if } v = \texttt{null} \\ \texttt{"Absent"} & \text{if } v = \texttt{absent} \\ \texttt{"Function"} & \text{if } v = \langle \lambda\texttt{(}\cdots\texttt{)}.\ i, \rho \rangle \\ \texttt{"Continuation"} & \text{if } v = \langle \kappa\texttt{(}\cdots\texttt{)}.\ i, c, \overline{c} \rangle \\ \texttt{"AST"} & \text{if } v = \curlywedge \\ s & \text{if } v = a \land \texttt{Get}(\sigma, a) = s\ \texttt{\{}\cdots\texttt{\}}) \\ \texttt{"List"} & \text{if } v = a \land \texttt{Get}(\sigma, a) = \texttt{[}\cdots\texttt{]}) \\ \texttt{"Symbol"} & \text{if } v = a \land \texttt{Get}(\sigma, a) = \texttt{symbol}\ v') \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{IsCompletion}(\sigma, v) = \begin{cases} \texttt{true} & \text{if } v = a \land \texttt{Get}(\sigma, a) = \texttt{"Completion"}\ \texttt{\{}\cdots\texttt{\}} \\ \texttt{false} & \text{otherwise} \end{cases}$$

$$\texttt{IsInstanceOf}(\curlywedge, s) = \begin{cases} \texttt{true} & \text{if } \curlywedge \text{ is the syntax element whose kind is } s \\ \texttt{false} & \text{otherwise} \end{cases}$$

$$\texttt{GetElems}(\sigma, \curlywedge, s) = (a, \sigma') \text{ where } \begin{cases} \langle \curlywedge_1, \cdots, \curlywedge_n \rangle = (\text{the list of syntax elements,} \\ \text{whose kind is } s\text{, of } \curlywedge \text{ with pre-order traversal}) \\ (a, \sigma') = \texttt{AllocList}(\sigma, \langle \curlywedge_1, \cdots, \curlywedge_n \rangle) \end{cases}$$

$$\texttt{GetSyntax}(\curlywedge) = (\text{the beautified form of string for } AST)$$

$$\texttt{Parse}(v, s, \langle b_1, \cdots, b_n \rangle) = \begin{cases} \curlywedge & \text{if } v = s_{\texttt{rule}} \land \\ & \quad \curlywedge = (\text{the parsing result of } s \text{ based on the rule } s_{\texttt{rule}} \\ & \qquad \text{with boolean arguments } \langle b_1, \cdots, b_n \rangle) \\ \curlywedge' & \text{if } v = \curlywedge \land \langle b'_1, \cdots, b'_n \rangle = (\text{boolean parameters stored in } \curlywedge) \land \\ & \quad \curlywedge' = (\text{the parsing result of } s \text{ based on the rule } s_{\texttt{rule}} \\ & \qquad \text{with boolean arguments } \langle b'_1, \cdots, b'_n \rangle) \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{Convert}(\triangleright, v, v') = \begin{cases} d & \text{if } \triangleright = \texttt{str2num} \ \wedge v = s \wedge v' = \texttt{absent} \wedge \\ & \quad d = \text{(the corresponding floating point of } s) \\ s & \text{if } \triangleright = \texttt{num2str} \ \wedge v = d \wedge v' = n \wedge \\ & \quad s = \text{(the corresponding string of } d \text{ with the radix } n) \\ n & \text{if } \triangleright = \texttt{num2int} \ \wedge v = d \wedge v' = \texttt{absent} \wedge \\ & \quad n = \text{(the corresponding integer value of } d) \wedge \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{Contains}(\sigma, a, v) = \begin{cases} \texttt{true} & \text{if } \texttt{Get}(\sigma, a) = \texttt{[}v_1, \cdots, v_n\texttt{]}) \wedge \exists 1 \le i \le n.\ v_i = v \\ \texttt{false} & \text{if } \texttt{Get}(\sigma, a) = \texttt{[}v_1, \cdots, v_n\texttt{]}) \wedge \forall 1 \le i \le n.\ v_i \ne v \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{Keys}(\sigma, a) = \begin{cases} (a', \sigma') & \text{if } \texttt{Get}(\sigma, a) = s\texttt{\{}v_1 \mapsto \_, \cdots, v_n \mapsto \_\texttt{\}}) \wedge \\ & \quad \langle v'_1, \cdots, v'_n \rangle = \text{(the list consisting of } v_1, \cdots, v_n \\ & \quad\quad \text{ordered by their creation time.)} \\ & \quad (a', \sigma') = \texttt{AllocList}(\sigma, \langle v'_1, \cdots, v'_n \rangle) \\ \bot & \text{otherwise} \end{cases}$$

$$\texttt{Lookup}(\sigma, x) = \begin{cases} \rho(x) & \text{if } \sigma = (c, \_, \_, \_) \wedge c = (\_, \_, \rho) \wedge x \in \texttt{Domain}(\rho) \\ \rho(x) & \text{if } \sigma = (\_, \_, \rho, \_) \wedge x \in \texttt{Domain}(\rho) \\ \texttt{absent} & \text{otherwise} \end{cases}$$