1 Abstract Syntax

This section explains abstract syntax of IR_{ES}.

$$
\begin{array}{rcl}
n & \in & \mathit{FloatingPoint} \\
d & \in & \mathit{Integer} \\
s & \in & \mathit{String} \\
b & \in & \mathit{Boolean} \\
r & \in & \mathit{Reference} \\
x & \in & \mathit{Identifier} \\
t & \in & \mathit{Type}
\end{array}
$$

$$
\text{Program} \quad p ::= i\texttt{;}\ \cdots\texttt{;}\ i
$$

$$
\begin{array}{rrcll}
\text{Instruction} & i & ::= & e & \text{(expression)} \\
 & & | & \texttt{let}\ x\ \texttt{=}\ e & \text{(let)} \\
 & & | & r\texttt{:=}e & \text{(assign)} \\
 & & | & \texttt{delete}\ r & \text{(delete)} \\
 & & | & \texttt{append}\ e\ \rightarrow\ e & \text{(append)} \\
 & & | & \texttt{prepend}\ e\ \rightarrow\ e & \text{(prepend)} \\
 & & | & \texttt{return}\ e & \text{(return)} \\
 & & | & \texttt{if}\ e\ i\ i & \text{(if-then-else)} \\
 & & | & \texttt{while}\ e\ i & \text{(while)} \\
 & & | & \{i^*\} & \text{(sequence)} \\
 & & | & \texttt{assert}\ e & \text{(assert)} \\
 & & | & \texttt{print}\ e & \text{(print)} \\
 & & | & \texttt{app}\ x\ \texttt{=}\ (e\ e^*) & \text{(function application)} \\
 & & | & \texttt{access}\ x\ \texttt{=}\ (e\ e) & \text{(access)} \\
 & & | & \texttt{withcont}\ x\ (x^*)\ =\ i & \text{(continuation)}
\end{array}
$$

$$
\begin{array}{rrcll}
\text{Reference} & r & ::= & x & \text{(identifier)} \\
 & & | & r\texttt{[}e\texttt{]} & \text{(reference to value of field in heap)}
\end{array}
$$

Expression	e	::=	n (number)
		|	d (integer)
		|	s (string)
		|	b (boolean)
		|	r (reference)
		|	`undefined` (undefined)
		|	`null` (null)
		|	`absent` (absent)
		|	`new` s (symbol)
		|	`new [`e^*`]` (list)
		|	`new` t `(`$e \mapsto e, \cdots, e \mapsto e$`)` (map)
		|	`pop` e e (pop)
		|	`typeof` e (typeof)
		|	`is-instance-of` e s (is-instance-of)
		|	`get-elems` e s (get-elements)
		|	`get-syntax` e (get-syntax)
		|	`parse-syntax` e e e^* (parse-syntax)
		|	`convert` $e \triangleright e^*$ (convert)
		|	`contains` e e (contains)
		|	`copy-obj` e (copy-object)
		|	`map-keys` e (map-keys)
		|	`!!!` s (not supported)
		|	$\odot$ e (unary operation)
		|	$e \oplus e$ (binary operation)
		|	`(`x^*`) [` $\Rightarrow$ `]` i (continuation)

UnaryOperator	$\odot$	::=	`-`	(negation)
		|	`!`	(boolean not)
		|	$\sim$	(bitwise not)
BinaryOperator	$\oplus$	::=	`+`	(addition)
		|	`-`	(subtraction)
		|	`*`	(multiplication)
		|	`**`	(power)
		|	`/`	(division)
		|	`%`	(modulo)
		|	`%`	(modulo)
		|	`=`	(equals)
		|	`&&`	(boolean and)
		|	`||`	(boolean or)
		|	`^^`	(boolean xor)
		|	`&`	(bitwise and)
		|	`|`	(bitwise or)
		|	`^`	(bitwise xor)
		|	`<<`	(shift left)
		|	`<`	(less-then)
		|	`>>>`	(unsigned shift right)
		|	`>>`	(shift right)
ConvertOperator	$\triangleright$	::=	`str2num`	(string to number)
		|	`num2str`	(number to string)
		|	`num2int`	(number to integer)

2 Operational Semantic

This section explains operational semantic of IR_{ES}.

2.1 Domain

Semantic domain of IR_{ES}.

$$
\begin{array}{rrcl}
\text{State} & \delta & \in & \mathit{Context} \times \mathit{ContextStack} \times \mathit{Global} \times \mathit{Heap} \\
\text{Context} & \Delta & \in & \mathit{Identifier} \times \mathit{String} \times \mathit{Instruction}^* \times \mathit{Environment} \\
\text{ContextStack} & \sqcup & \in & \mathit{Context}^* \\
\text{Global} & \mathbb{G} & \in & \mathit{Identifier} \rightarrow \mathit{Value} \\
\text{Environment} & \sigma & \in & \mathit{Identifier} \rightarrow \mathit{Value} \\
\text{Heap} & \Sigma & \in & \mathit{Address} \rightarrow \mathit{Object} \\
\text{Value} & v & \in & \mathit{Value} \\
\text{Address} & a & \in & \mathit{Address} \\
\text{Object} & o & \in & \mathit{Object}
\end{array}
$$

$$
\text{Constant} \quad c ::= n \mid d \mid s \mid b \mid \texttt{undefined} \mid \texttt{null} \mid \texttt{absent}
$$

Address	a	::= s	(named address)
		$\mid d$	(dynamic address)
Object	o	::= $\texttt{symbol}\ v$	(symbol)
		$\mid \texttt{map}\ t\ \texttt{(}v \mapsto v, \cdots, v \mapsto v\texttt{)}$	(map)
		$\mid \texttt{list}\ \texttt{[}v^*\texttt{]}$	(list)
		$\mid \texttt{not-supported}\ s$	(not supported)
Value	v	::= a	(address)
		$\mid c$	(constant)
		$\mid s\ \texttt{(}x^*, \texttt{[}x\texttt{])} \Rightarrow i$	(function)
		$\mid \Delta, \sqcup, x^*\ \texttt{[} \Rightarrow \texttt{]}\ i$	(continuation)
		$\mid \mathit{ASTVal}?$	(AST value)
		$\mid \mathit{ASTMethod}?$	(AST method)
RefValue	rv	::= x	(identifier)
		$\mid a\texttt{[}v\texttt{]}$	(reference to value of map in heap)
		$\mid s\texttt{.}v$	(reference to ???)

2.1.1 Global, Environment, Heap Description

.

[[description of initial configuration of IR_{ES}]]
Heap = (Base + Built-in) → named addr + Dynamic → dynamic addr
[[description of operators used in state, context, heap]]

2.2 Semantic of $\mathbf{IR_{ES}}$

- program : [[description of program execution]]
- instruction : $\delta \vdash i \Rightarrow \delta$
- expression : $\delta \vdash e \Rightarrow v,\ \delta$
- expression - escape completion : $\delta \vdash_{escape} e \Rightarrow v,\ \delta$
- reference : $\delta \vdash r \Rightarrow rv,\ \delta$
- reference value : $\delta \vdash rv \Rightarrow v,\ \delta$
- unary operator : $\odot v \Rightarrow v$
- binary operator : $v \oplus v \Rightarrow v$

[[description of semantic relation]]
[[description of escape completion]] : result of abstract algorithm in ECMAScript is represented by completion record. "Escape completion" means "use value of completion record($\mathrm{IR_{ES}}$ object), instead of record itself".

2.2.1 Instruction

$\boxed{\delta \vdash i \Rightarrow \delta}$

$$\frac{\delta \vdash e \Rightarrow v,\ \delta_0}{\delta \vdash e \Rightarrow \delta_0} \qquad \frac{\delta \vdash e \Rightarrow v,\ \delta_0 \quad \texttt{define}(\delta_0,\ x,\ v) = \delta_1}{\delta \vdash \texttt{let}\ x\ \texttt{=}\ e \Rightarrow \delta_1}$$

$$\frac{\delta \vdash r \Rightarrow x,\ \delta_0 \quad \delta_0 \vdash e \Rightarrow v,\ \delta_1 \quad \texttt{updated}(\delta_1,\ x,\ v) = \delta_2}{\delta \vdash r\texttt{:=}\ e \Rightarrow \delta_2}$$

$$\frac{\delta \vdash r \Rightarrow a\texttt{[}v\texttt{]},\ \delta_0 \quad \delta_0 \vdash_{escape} e \Rightarrow v_0,\ \delta_1 \quad \texttt{updated}(\delta_1,\ a\texttt{[}v\texttt{]},\ v_0) = \delta_2}{\delta \vdash r\texttt{:=}\ e \Rightarrow \delta_2}$$

$$\frac{\delta \vdash r \Rightarrow s\texttt{.}v,\ \delta_0 \quad \delta_0 \vdash_{escape} e \Rightarrow v_0,\ \delta_1 \quad \texttt{updated}(\delta_1,\ s\texttt{.}v,\ v_0) = \delta_2}{\delta \vdash r\texttt{:=}\ e \Rightarrow \delta_2}$$

$$\frac{\delta \vdash r \Rightarrow rv,\ \delta_0 \quad \texttt{deleted}(\delta_0,\ rv) = \delta_1}{\delta \vdash \texttt{delete}\ r \Rightarrow \delta_1}$$

$$\frac{\delta \vdash_{escape} e_0 \Rightarrow v_0,\ \delta_0 \quad \delta_0 \vdash_{escape} e_1 \Rightarrow v_1,\ \delta_1 \quad \texttt{append}(\delta_0\texttt{??},\ v_1,\ v_0) = \delta_2}{\delta \vdash \texttt{append}\ e_0 \rightarrow e_1 \Rightarrow \delta_2}$$

$$\frac{\delta \vdash_{escape} e_0 \Rightarrow v_0,\ \delta_0 \quad \delta_0 \vdash_{escape} e_1 \Rightarrow v_1,\ \delta_1 \quad \texttt{prepend}(\delta_0\texttt{??},\ v_1,\ v_0) = \delta_2}{\delta \vdash \texttt{prepend}\ e_0 \rightarrow e_1 \Rightarrow \delta_2}$$

$$\frac{\delta \vdash e \Rightarrow v,\ \delta_0 \quad \texttt{updateCtxStack}(\delta_0,\ v) = \delta_1}{\delta \vdash \texttt{return}\ e \Rightarrow \delta_1}$$

$$\frac{\delta \vdash_{escape} e \Rightarrow \texttt{true},\ \delta_0 \quad \texttt{updateCtx}(\delta_0,\ i_0) = \delta_1}{\delta \vdash \texttt{if}\ e\ i_0\ i_1 \Rightarrow \delta_1}$$

$$\frac{\delta \vdash_{escape} e \Rightarrow \texttt{false},\ \delta_0 \quad \texttt{updateCtx}(\delta_0,\ i_1) = \delta_1}{\delta \vdash \texttt{if}\ e\ i_0\ i_1 \Rightarrow \delta_1}$$

$$\frac{\delta \vdash_{escape} e \Rightarrow \texttt{true},\ \delta_0 \quad \texttt{updateCtx}(\delta_0,\ i) = \delta_1}{\delta \vdash \texttt{while}\ e\ i \Rightarrow \delta_1} \qquad \frac{\delta \vdash_{escape} e \Rightarrow \texttt{false},\ \delta_0}{\delta \vdash \texttt{while}\ e\ i \Rightarrow \delta_0}$$

$$\frac{\texttt{updateCtx2}(\delta,\ i^*) = \delta_0}{\delta \vdash \{i^*\} \Rightarrow \delta_0} \qquad \frac{\delta \vdash e \Rightarrow \texttt{true},\ \delta_0}{\delta \vdash \texttt{assert}\ e \Rightarrow \delta_0} \qquad \frac{\delta \vdash e \Rightarrow v,\ \delta_0 \quad \texttt{print}(v)}{\delta \vdash \texttt{print}\ e \Rightarrow \delta_0}$$

$$\frac{\Delta = \texttt{getCtx}(\delta) \quad \sqcup = \texttt{getCtxStack}(\delta) \quad v_c = \Delta,\ \sqcup,\ x^*\ \texttt{[}\Rightarrow\texttt{]}\ i \quad \texttt{define}(\delta,\ x_{id},\ v_c) = \delta_0}{\delta \vdash \texttt{withcont}\ x_{id}\ \texttt{(}x^*\texttt{)}\ \texttt{=}\ i \Rightarrow \delta_0}$$

$$\frac{\begin{array}{c}\delta \vdash e_f \Rightarrow v_f,\ \delta_f \qquad v_f = s\ \texttt{(}x^*,\ \texttt{[}x_{var}\texttt{])}\ \Rightarrow\ i \\ \delta_f \vdash e_0 \Rightarrow v_0,\ \delta_0 \quad \cdots \quad \delta_{n-1} \vdash e_n \Rightarrow v_n,\ \delta_n \\ \texttt{updateCtxRetId}(\delta_n,\ x) = \delta_\alpha \quad \texttt{createCtx}(s,\ i,\ x^*,\ v^*) = \Delta_\alpha \quad \texttt{pushCtx}(\delta_\alpha,\ \Delta_\alpha) = \delta_{next}\end{array}}{\delta \vdash \texttt{app}\ x\ \texttt{=}\ (e_f\ e^*) \Rightarrow \delta_{next}}$$

$$\frac{\delta \vdash e_f \Rightarrow v,\ \delta_0}{\delta \vdash \texttt{app}\ x\ \texttt{=}\ (e_f\ e^*) \Rightarrow \delta_0}$$

$$\frac{\begin{array}{c}\delta \vdash e_f \Rightarrow v_f,\ \delta_f \qquad v_f = \Delta,\ \sqcup,\ x^*\ \texttt{[}\Rightarrow\texttt{]}\ i \\ \delta_f \vdash e_0 \Rightarrow v_0,\ \delta_0 \quad \cdots \quad \delta_{n-1} \vdash e_n \Rightarrow v_n,\ \delta_n \\ \texttt{setCtxInst}(\Delta,\ i) = \Delta_0 \quad \texttt{updateCtxEnv}(\Delta_0,\ x^*,\ v^*) = \Delta_1 \quad \texttt{updateState}(\delta_n,\ \Delta_1,\ \sqcup) = \delta_{next}\end{array}}{\delta \vdash \texttt{app}\ x\ \texttt{=}\ (e_f\ e^*) \Rightarrow \delta_{next}}$$

2.2.2 Expression

$$\boxed{\delta \vdash e \Rightarrow v,\ \delta} \qquad \frac{x \in Domain(\sigma)}{\delta \vdash x \Rightarrow \sigma(x)} \qquad \delta \vdash n \Rightarrow n \qquad \delta \vdash b \Rightarrow b$$