

1 Abstract Syntax

This section explains abstract syntax of IR_{ES} .

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

Program $p ::= i; \dots; i$

Instruction	$i ::= e$	(expression)
	let $x = e$	(let)
	$r := e$	(assign)
	delete r	(delete)
	append $e \rightarrow e$	(append)
	prepend $e \rightarrow e$	(prepend)
	return e	(return)
	if e i i	(if-then-else)
	while e i	(while)
	$\{i^*\}$	(sequence)
	assert e	(assert)
	print e	(print)
	app $x = (e \ e^*)$	(function application)
	access $x = (e \ e)$	(access)
	withcont $x \ (x^*) = i$	(continuation)
Reference	$r ::= x$	(identifier)
	$r[e]$	(reference to value of field in heap)

Expression	$e ::=$	n	(number)
		d	(integer)
		s	(string)
		b	(boolean)
		r	(reference)
		undefined	(undefined)
		null	(null)
		absent	(absent)
		new e	(symbol)
		new $[e^*]$	(list)
		new $t (e \mapsto e, \dots, e \mapsto e)$	(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)
			~	(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-than)
			>>>	(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} .

State	δ	\in	$\text{Context} \times \text{ContextStack} \times \text{Global} \times \text{Heap}$
Context	Δ	\in	$\text{Identifier} \times \text{String} \times \text{Instruction}^* \times \text{Environment}$
ContextStack	\sqcup	\in	Context^*
Global	\mathbb{G}	\in	$\text{Identifier} \rightarrow \text{Value}$
Environment	σ	\in	$\text{Identifier} \rightarrow \text{Value}$
Heap	Σ	\in	$\text{Address} \rightarrow \text{Object}$
Value	v	\in	Value
Address	a	\in	Address
Object	o	\in	Object

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

Address $a ::= s$ (named address)
 $\mid d$ (dynamic address)

Object $o ::= \text{symbol } v$ (symbol)
 $\mid \text{map } t (v \mapsto v, \dots, v \mapsto v)$ (map)
 $\mid \text{list } [v^*]$ (list)
 $\mid \text{not-supported } s$ (not supported)

Value $v ::= a$ (address)
 $\mid c$ (constant)
 $\mid s (x^*, [x]) \Rightarrow i$ (function)
 $\mid \Delta, \sqcup, x^* [\Rightarrow] i$ (continuation)
 $\mid \text{ASTVal?}$ (AST value)
 $\mid \text{ASTMethod?}$ (AST method)

RefValue $rv ::= x$ (identifier)
 $\mid a[v]$ (reference to value of map in heap)
 $\mid s.v$ (reference to ???)

2.1.1 Global, Environment, Heap Description

[[description of initial configuration of IR_{ES}]]

Heap = (Base + Built-in) \rightarrow named addr + Dynamic \rightarrow dynamic addr

[[description of operators used in state, context, heap]]

2.2 Semantic of IR_{ES}

- program : [[description of program execution]]
- instruction : $\delta \vdash i \Rightarrow \delta$
- expression : $\delta \vdash e \Rightarrow v, \delta$
- expression - escape completion : $\delta \vdash_{\text{escape}} e \Rightarrow v, \delta$
- 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(IR_{ES} object), instead of record itself".

2.2.1 State Helper Function

[[description of state function]]

2.2.2 Instruction

$$\begin{array}{c}
\boxed{\delta \vdash i \Rightarrow \delta} \quad \frac{\delta \vdash e \Rightarrow v, \delta_0}{\delta \vdash e \Rightarrow \delta_0} \quad \frac{\delta \vdash e \Rightarrow v, \delta_0 \quad \text{define}(\delta_0, x, v) = \delta_1}{\delta \vdash \text{let } x = e \Rightarrow \delta_1} \\
\\
\frac{\delta \vdash r \Rightarrow x, \delta_0 \quad \delta_0 \vdash e \Rightarrow v, \delta_1 \quad \text{updated}(\delta_1, x, v) = \delta_2}{\delta \vdash r := e \Rightarrow \delta_2} \\
\\
\frac{\delta \vdash r \Rightarrow a[v], \delta_0 \quad \delta_0 \vdash_{\text{escape}} e \Rightarrow v_0, \delta_1 \quad \text{updated}(\delta_1, a[v], v_0) = \delta_2}{\delta \vdash r := e \Rightarrow \delta_2} \\
\\
\frac{\delta \vdash r \Rightarrow s.v, \delta_0 \quad \delta_0 \vdash_{\text{escape}} e \Rightarrow v_0, \delta_1 \quad \text{updated}(\delta_1, s.v, v_0) = \delta_2}{\delta \vdash r := e \Rightarrow \delta_2} \\
\\
\frac{\delta \vdash r \Rightarrow rv, \delta_0 \quad \text{deleted}(\delta_0, rv) = \delta_1}{\delta \vdash \text{delete } r \Rightarrow \delta_1} \\
\\
\frac{\delta \vdash_{\text{escape}} e_0 \Rightarrow v_0, \delta_0 \quad \delta_0 \vdash_{\text{escape}} e_1 \Rightarrow v_1, \delta_1 \quad \text{append}(\delta_0??, v_1, v_0) = \delta_2}{\delta \vdash \text{append } e_0 \rightarrow e_1 \Rightarrow \delta_2} \\
\\
\frac{\delta \vdash_{\text{escape}} e_0 \Rightarrow v_0, \delta_0 \quad \delta_0 \vdash_{\text{escape}} e_1 \Rightarrow v_1, \delta_1 \quad \text{prepend}(\delta_0??, v_1, v_0) = \delta_2}{\delta \vdash \text{prepend } e_0 \rightarrow e_1 \Rightarrow \delta_2} \\
\\
\frac{\delta \vdash e \Rightarrow v, \delta_0 \quad \text{updateCtxStack}(\delta_0, v) = \delta_1}{\delta \vdash \text{return } e \Rightarrow \delta_1} \\
\\
\frac{\delta \vdash_{\text{escape}} e \Rightarrow \text{true}, \delta_0 \quad \text{updateCtx}(\delta_0, i_0) = \delta_1}{\delta \vdash \text{if } e \ i_0 \ i_1 \Rightarrow \delta_1} \\
\\
\frac{\delta \vdash_{\text{escape}} e \Rightarrow \text{false}, \delta_0 \quad \text{updateCtx}(\delta_0, i_1) = \delta_1}{\delta \vdash \text{if } e \ i_0 \ i_1 \Rightarrow \delta_1} \\
\\
\frac{\delta \vdash_{\text{escape}} e \Rightarrow \text{true}, \delta_0 \quad \text{updateCtx}(\delta_0, i) = \delta_1}{\delta \vdash \text{while } e \ i \Rightarrow \delta_1} \quad \frac{\delta \vdash_{\text{escape}} e \Rightarrow \text{false}, \delta_0}{\delta \vdash \text{while } e \ i \Rightarrow \delta_0} \\
\\
\frac{\text{updateCtx2}(\delta, i^*) = \delta_0}{\delta \vdash \{i^*\} \Rightarrow \delta_0} \quad \frac{\delta \vdash e \Rightarrow \text{true}, \delta_0}{\delta \vdash \text{assert } e \Rightarrow \delta_0} \quad \frac{\delta \vdash e \Rightarrow v, \delta_0 \quad \text{print}(v)}{\delta \vdash \text{print } e \Rightarrow \delta_0} \\
\\
\frac{\Delta = \text{getCtx}(\delta) \quad \sqcup = \text{getCtxStack}(\delta) \quad v_c = \Delta, \sqcup, x^* [\Rightarrow] i \quad \text{define}(\delta, x_{id}, v_c) = \delta_0}{\delta \vdash \text{withcont } x_{id} (x^*) = i \Rightarrow \delta_0}
\end{array}$$

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

$$\begin{array}{c}
\delta \vdash e_f \Rightarrow v, \delta_0 \\
\hline
\delta \vdash \text{app } x = (e_f e^*) \Rightarrow \delta_0
\end{array}$$

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

TODO

function application - method

access

2.2.3 Expression

$$\begin{array}{c}
\boxed{\delta \vdash e \Rightarrow v, \delta} \quad \delta \vdash n \Rightarrow n, \delta \quad \delta \vdash d \Rightarrow d, \delta \quad \delta \vdash s \Rightarrow s, \delta \quad \delta \vdash b \Rightarrow b, \delta \\
\\
\delta \vdash \text{undefined} \Rightarrow \text{undefined}, \delta \quad \delta \vdash \text{null} \Rightarrow \text{null}, \delta \quad \delta \vdash \text{absent} \Rightarrow \text{absent}, \delta \\
\\
\frac{\delta \vdash_{\text{escape}} e \Rightarrow v, \delta_0 \quad \odot v \Rightarrow v_0}{\delta \vdash \odot e \Rightarrow v_0, \delta_0} \quad \frac{\delta \vdash_{\text{escape}} e_0 \Rightarrow v_0, \delta_0 \quad \delta_0 \vdash_{\text{escape}} e_1 \Rightarrow v_1, \delta_1 \quad v_0 \oplus v_1 \Rightarrow v_2}{\delta \vdash e_0 \oplus e_1 \Rightarrow v_2, \delta_1} \\
\\
\frac{\Delta = \text{getCtx}(\delta) \quad \sqcup = \text{getCtxStack}(\delta) \quad v_c = \Delta, \sqcup, x^* [\Rightarrow] i}{\delta \vdash (x^*) [\Rightarrow] i \Rightarrow v_c, \delta} \quad \frac{\delta \vdash e \Rightarrow v, \delta_0 \quad \text{getType}(v) = s_{\text{type}}}{\delta \vdash \text{typeof } e \Rightarrow s_{\text{type}}, \delta_0} \\
\\
\frac{\delta \vdash_{\text{escape}} e \Rightarrow s, \delta_0 \quad \text{allocSymbol}(\delta_0, s) = (a, \delta_1)}{\delta \vdash \text{new } e \Rightarrow a, \delta_1} \\
\\
\frac{\delta \vdash_{\text{escape}} e \Rightarrow \text{undefined}, \delta_0 \quad \text{allocSymbol}(\delta_0, \text{undefined}) = (a, \delta_1)}{\delta \vdash \text{new } e \Rightarrow a, \delta_1} \\
\\
\frac{\delta \vdash e_0 \Rightarrow v_0, \delta_0 \quad \cdots \quad \delta_{n-1} \vdash e_n \Rightarrow v_n, \delta_n \quad \text{allocList}(\delta_n, v^*) = (a, \delta_{\text{next}})}{\delta \vdash \text{new } [e^*] \Rightarrow a, \delta_{\text{next}}} \\
\\
\frac{\begin{array}{c} \text{allocMap}(\delta, t) = (a, \delta_t) \\ \delta_t \vdash_{\text{escape}} e_{k_0} \Rightarrow v_{k_0}, \delta_{0_k} \quad \delta_{0_k} \vdash e_{v_0} \Rightarrow v_{v_0}, \delta_{0_v} \quad \text{updated}(\delta_{0_v}, v_{k_0}, v_{v_0}) = \delta_0 \\ \cdots \\ \delta_{n-1} \vdash_{\text{escape}} e_{k_n} \Rightarrow v_{k_n}, \delta_{n_k} \quad \delta_{n_k} \vdash e_{v_n} \Rightarrow v_{v_n}, \delta_{n_v} \quad \text{updated}(\delta_{n_v}, v_{k_n}, v_{v_n}) = \delta_n \end{array}}{\delta \vdash \text{new } t (e_{k_0} \mapsto e_{v_0}, \cdots, e_{k_n} \mapsto e_{v_n}) \Rightarrow a, \delta_n} \\
\\
\frac{\delta \vdash_{\text{escape}} e_0 \Rightarrow a, \delta_0 \quad \delta_0 \vdash_{\text{escape}} e_1 \Rightarrow v_k, \delta_1 \quad \text{pop}(\delta_1, a, v_k) = (v, \delta_2)}{\delta \vdash \text{pop } e_0 e_1 \Rightarrow v, \delta_2} \\
\\
\frac{\delta \vdash_{\text{escape}} e \Rightarrow a_0, \delta_0 \quad \text{copyObj}(\delta_0, a_0) = (a_1, \delta_1)}{\delta \vdash \text{copy-obj } e \Rightarrow a_1, \delta_1} \quad \frac{\delta \vdash_{\text{escape}} e \Rightarrow a_0, \delta_0 \quad \text{keys}(\delta_0, a_0) = (a_1, \delta_1)}{\delta \vdash \text{map-keys } e \Rightarrow a_1, \delta_1} \\
\\
\frac{\delta \vdash_{\text{escape}} e_0 \Rightarrow a_l, \delta_0 \quad \delta_0 \vdash_{\text{escape}} e_1 \Rightarrow v, \delta_1 \quad \text{contains}(\delta_1, a_l, v) = b}{\delta \vdash \text{contains } e_0 e_1 \Rightarrow b, \delta_1}
\end{array}$$

TODO

reference

is-instance-of

get-elements

get-syntax

parse-syntax

convert

not supported

handle escapeCompletion