1 Abstract Syntax

This section explains abstract syntax of $\mathrm{IR}_{\mathrm{ES}}$.

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

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

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

Expression	e	::=	n (number)
		|	i (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}.

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

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

Address	a	::=	s	(named address)
		$\mid$	d	(dynamic address)
Object	o	::=	`symbol` v	(symbol)
		$\mid$	`map` t `(`$v \mapsto v, \cdots, v \mapsto v$`)`	(map)
		$\mid$	`list [`v^*`]`	(list)
		$\mid$	`not-supported` s	(not supported)
Value	v	::=	a	(address)
		$\mid$	c	(constant)
		$\mid$	s `(`x^*`)` $\Rightarrow$ i	(function(varparam?))
		$\mid$	$\sigma, \sqcup \vdash$ `(`x^*`) [` $\Rightarrow$ `]` i	(continuation)
		$\mid$	ASTVal?	(AST value)
		$\mid$	$\mathit{ASTMethod}$?	(AST method)

2.2 State Transition

2.2.1 Instruction

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