1 Abstract Syntax

This section explains abstract syntax of IR_{ES}.

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

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

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