1 Abstract Syntax

This section explains abstract syntax of IR_{ES}.

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

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

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

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

Expression	e	::=	n (floating point number)
		|	d (integer)
		|	s (string)
		|	b (boolean)
		|	r (reference)
		|	`undefined` (undefined)
		|	`null` (null)
		|	`absent` (absent)
		|	`new` e (symbol)
		|	`new <`e^*`>` (list)
		|	`new` t $\{[e \mapsto e]^*\}$ (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}.

$$
\begin{array}{rrcl}
\text{State} & \sigma & \in & \text{Context} \times \text{Context}^* \times \text{Environment} \times \text{Heap} \\
\text{Context} & C & \in & \text{Identifier} \times \text{String} \times \text{Instruction}^* \times \text{Environment} \\
\text{Environment} & E & \in & \text{Identifier} \to \text{Value} \\
\text{Heap} & H & \in & \text{Address} \to \text{Object} \\
\text{Value} & v & \in & \text{Value} \\
\text{Address} & a & \in & \text{Address} \\
\text{Object} & o & \in & \text{Object}
\end{array}
$$

$$
\text{State} \quad \sigma \quad ::= \quad (C,\ C^*,\ E,\ H) \qquad \text{Context} \quad C \quad ::= \quad (x,\ s,\ i^*,\ E)
$$

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

Object	o	::=	$\texttt{symbol}\ v$	(symbol)
		|	$t\ \{[v \mapsto v]^*\}$	(map)
		|	$\texttt{<}v^*\texttt{>}$	(list)
		|	$\texttt{not-supported}\ s$	(not supported)
Value	v	::=	a	(address)
		|	c	(constant)
		|	$\lambda\texttt{(}s,\ x^*,\ x,\ i\texttt{)}$	(function)
		|	$\kappa\texttt{(}C,\ C^*,\ x^*,\ i\texttt{)}$	(continuation)
		|	$ASTVal$	(AST value)
		|	$\texttt{ASTMethod}\ \lambda\texttt{(}s,\ x^*,\ x,\ i\texttt{)}\ E$	(AST method)
RefValue	rv	::=	x	(identifier)
		|	$a\texttt{[}v\texttt{]}$	(reference to value of map in heap)
		|	$s\texttt{.}v$	(reference to string field)

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

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

2.2.1 Instruction

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

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

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

$$\frac{\sigma \vdash r \Rightarrow a\texttt{[}v\texttt{]},\ \sigma_0 \quad \sigma_0 \vdash_{escape} e \Rightarrow v_0,\ \sigma_1 \quad \texttt{updated}(\sigma_1,\ a\texttt{[}v\texttt{]},\ v_0) = \sigma_2}{\sigma \vdash r\texttt{:=}\ e \Rightarrow \sigma_2}$$

$$\frac{\sigma \vdash r \Rightarrow s\texttt{.}v,\ \sigma_0 \quad \sigma_0 \vdash_{escape} e \Rightarrow v_0,\ \sigma_1 \quad \texttt{updated}(\sigma_1,\ s\texttt{.}v,\ v_0) = \sigma_2}{\sigma \vdash r\texttt{:=}\ e \Rightarrow \sigma_2}$$

$$\frac{\sigma \vdash r \Rightarrow rv,\ \sigma_0 \quad \texttt{deleted}(\sigma_0,\ rv) = \sigma_1}{\sigma \vdash \texttt{delete}\ r \Rightarrow \sigma_1}$$

$$\frac{\sigma \vdash_{escape} e_0 \Rightarrow v_0,\ \sigma_0 \quad \sigma_0 \vdash_{escape} e_1 \Rightarrow v_1,\ \sigma_1 \quad \texttt{append}(\sigma_0\texttt{??},\ v_1,\ v_0) = \sigma_2}{\sigma \vdash \texttt{append}\ e_0 \rightarrow e_1 \Rightarrow \sigma_2}$$

$$\frac{\sigma \vdash_{escape} e_0 \Rightarrow v_0,\ \sigma_0 \quad \sigma_0 \vdash_{escape} e_1 \Rightarrow v_1,\ \sigma_1 \quad \texttt{prepend}(\sigma_0\texttt{??},\ v_1,\ v_0) = \sigma_2}{\sigma \vdash \texttt{prepend}\ e_0 \rightarrow e_1 \Rightarrow \sigma_2}$$

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

$$\frac{\sigma \vdash_{escape} e \Rightarrow \texttt{true},\ \sigma_0 \quad \texttt{updateCtx}(\sigma_0,\ i_0) = \sigma_1}{\sigma \vdash \texttt{if}\ e\ i_0\ i_1 \Rightarrow \sigma_1}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow \texttt{false},\ \sigma_0 \quad \texttt{updateCtx}(\sigma_0,\ i_1) = \sigma_1}{\sigma \vdash \texttt{if}\ e\ i_0\ i_1 \Rightarrow \sigma_1}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow \texttt{true},\ \sigma_0 \quad \texttt{updateCtx}(\sigma_0,\ i) = \sigma_1}{\sigma \vdash \texttt{while}\ e\ i \Rightarrow \sigma_1} \qquad \frac{\sigma \vdash_{escape} e \Rightarrow \texttt{false},\ \sigma_0}{\sigma \vdash \texttt{while}\ e\ i \Rightarrow \sigma_0}$$

$$\frac{\texttt{updateCtx2}(\sigma,\ i^*) = \sigma_0}{\sigma \vdash \{i^*\} \Rightarrow \sigma_0} \qquad \frac{\sigma \vdash e \Rightarrow \texttt{true},\ \sigma_0}{\sigma \vdash \texttt{assert}\ e \Rightarrow \sigma_0} \qquad \frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \quad \texttt{print}(v)}{\sigma \vdash \texttt{print}\ e \Rightarrow \sigma_0}$$

$$\frac{\sigma = (C,\ C^*,\ E,\ H) \quad \texttt{define}(\sigma,\ x_{id},\ \kappa\texttt{(}C,\ C^*,\ x^*,\ i\texttt{)}) = \sigma_0}{\sigma \vdash \texttt{withcont}\ x_{id}\ \texttt{(}x^*\texttt{)}\ \texttt{=}\ i \Rightarrow \sigma_0}$$

$$\frac{\begin{array}{c}\sigma \vdash e_f \Rightarrow v_f,\ \sigma_f \qquad v_f = \lambda(s,\ x^*,\ x_{var},\ i) \\ \sigma_f \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \\ \texttt{updateCtxRetId}(\sigma_n,\ x) = \sigma_\alpha \quad \texttt{createCtx}(s,\ i,\ x^*,\ v^*) = C_\alpha \quad \texttt{pushCtx}(\sigma_\alpha,\ C_\alpha) = \sigma_{next}\end{array}}{\sigma \vdash \texttt{app}\ x\ \texttt{=}\ (e_f\ e_0\ \cdots\ e_n) \Rightarrow \sigma_{next}}$$

$$\frac{\begin{array}{c}\sigma \vdash e_f \Rightarrow v_f,\ \sigma_f \qquad v_f = \texttt{ASTMethod}\ \lambda(s,\ x^*,\ x_{var},\ i)\ E \\ \sigma_f \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \\ \texttt{updateCtxRetId}(\sigma_n,\ x) = \sigma_\alpha \quad \texttt{createCtx2}(s,\ i,\ x^*,\ v^*,\ E) = C_\alpha \quad \texttt{pushCtx}(\sigma_\alpha,\ C_\alpha) = \sigma_{next}\end{array}}{\sigma \vdash \texttt{app}\ x\ \texttt{=}\ (e_f\ e_0\ \cdots\ e_n) \Rightarrow \sigma_{next}}$$

$$\frac{\begin{array}{c}\sigma \vdash e_f \Rightarrow v_f,\ \sigma_f \qquad v_f = \kappa(C,\ C^*,\ x^*,\ i) \\ \sigma_f \vdash e_0 \Rightarrow v_0,\ \sigma_0 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \\ \texttt{setCtxInst}(C,\ i) = C_0 \quad \texttt{updateCtxEnv}(C_0,\ x^*,\ v^*) = C_1 \quad \texttt{updateState}(\sigma_n,\ C_1,\ C^*) = \sigma_{next}\end{array}}{\sigma \vdash \texttt{app}\ x\ \texttt{=}\ (e_f\ e_0\ \cdots\ e_n) \Rightarrow \sigma_{next}}$$

$$\frac{\begin{array}{c}\sigma \vdash e_b \Rightarrow a,\ \sigma_0 \quad \sigma_0 \vdash_{escape} e \Rightarrow s_p,\ \sigma_1 \\ s_p = \text{"}\texttt{length}\text{"} \quad \texttt{define}(\sigma_1,\ x,\ \texttt{length}(s)) = \sigma_2\end{array}}{\sigma \vdash \texttt{access}\ x\ \texttt{=}\ (e_b\ e) \Rightarrow \sigma_2}$$

$$\frac{\begin{array}{c}\sigma \vdash e_b \Rightarrow s,\ \sigma_0 \quad \sigma_0 \vdash_{escape} e \Rightarrow s_p,\ \sigma_1 \\ s_p = \text{"}\texttt{length}\text{"} \quad \texttt{define}(\sigma_1,\ x,\ \texttt{length}(s)) = \sigma_2\end{array}}{\sigma \vdash \texttt{access}\ x\ \texttt{=}\ (e_b\ e) \Rightarrow \sigma_2}$$

$$\frac{\sigma \vdash e_b \Rightarrow s,\ \sigma_0 \quad \sigma_0 \vdash_{escape} e \Rightarrow d,\ \sigma_1 \quad \texttt{define}(\sigma_1,\ x,\ \texttt{ToString}(d)) = \sigma_2}{\sigma \vdash \texttt{access}\ x\ \texttt{=}\ (e_b\ e) \Rightarrow \sigma_2}$$

TODO
access - addr
access - ASTVal - Lexical
access - ASTVal

2.2.2 Expression

$$\boxed{\sigma \vdash e \Rightarrow v,\ \sigma} \qquad \sigma \vdash n \Rightarrow n,\ \sigma \qquad \sigma \vdash d \Rightarrow d,\ \sigma \qquad \sigma \vdash s \Rightarrow s,\ \sigma \qquad \sigma \vdash b \Rightarrow b,\ \sigma$$

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

$$\frac{\sigma \vdash_{escape} e \Rightarrow v,\ \sigma_0 \qquad \odot\, v \Rightarrow v_0}{\sigma \vdash \odot\ e \Rightarrow v_0,\ \sigma_0} \qquad \frac{\sigma \vdash_{escape} e_0 \Rightarrow v_0,\ \sigma_0 \qquad \sigma_0 \vdash_{escape} e_1 \Rightarrow v_1,\ \sigma_1 \qquad v_0 \oplus v_1 \Rightarrow v_2}{\sigma \vdash e_0\ \oplus\ e_1 \Rightarrow v_2,\ \sigma_1}$$

$$\frac{\sigma = (C,\ C^*,\ E,\ H)}{\sigma \vdash \texttt{(}x^*\texttt{) [} \Rightarrow \texttt{]}\ i \Rightarrow \kappa(C,\ C^*,\ x^*,\ i),\ \sigma} \qquad \frac{\sigma \vdash e \Rightarrow v,\ \sigma_0 \qquad \texttt{getType}(v) = s_{type}}{\sigma \vdash \texttt{typeof}\ e \Rightarrow s_{type},\ \sigma_0}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow s,\ \sigma_0 \qquad \texttt{allocSymbol}(\sigma_0,\ s) = (a,\ \sigma_1)}{\sigma \vdash \texttt{new}\ e \Rightarrow a,\ \sigma_1}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow \texttt{undefined},\ \sigma_0 \qquad \texttt{allocSymbol}(\sigma_0,\ \texttt{undefined}) = (a,\ \sigma_1)}{\sigma \vdash \texttt{new}\ e \Rightarrow a,\ \sigma_1}$$

$$\frac{\sigma \vdash e_0 \Rightarrow v_0,\ \sigma_0 \qquad \cdots \qquad \sigma_{n-1} \vdash e_n \Rightarrow v_n,\ \sigma_n \qquad \texttt{allocList}(\sigma_n,\ v^*) = (a,\ \sigma_{next})}{\sigma \vdash \texttt{new [}e_0,\ \cdots,\ e_n\texttt{]} \Rightarrow a,\ \sigma_{next}}$$

$$\frac{\begin{array}{c}\texttt{allocMap}(\sigma,\ t) = (a,\ \sigma_t) \\ \sigma_t \vdash_{escape} e_{k_0} \Rightarrow v_{k_0},\ \sigma_{0_k} \qquad \sigma_{0_k} \vdash e_{v_0} \Rightarrow v_{v_0},\ \sigma_{0_v} \qquad \texttt{updated}(\sigma_{0_v},\ v_{k_0},\ v_{v_0}) = \sigma_0 \\ \cdots \\ \sigma_{n-1} \vdash_{escape} e_{k_n} \Rightarrow v_{k_n},\ \sigma_{n_k} \qquad \sigma_{n_k} \vdash e_{v_n} \Rightarrow v_{v_n},\ \sigma_{n_v} \qquad \texttt{updated}(\sigma_{n_v},\ v_{k_n},\ v_{v_n}) = \sigma_n\end{array}}{\sigma \vdash \texttt{new}\ t\ \texttt{(}e_{k_0}\ \mapsto\ e_{v_0},\ \cdots,\ e_{k_n}\ \mapsto\ e_{v_n}\texttt{)} \Rightarrow a,\ \sigma_n}$$

$$\frac{\sigma \vdash_{escape} e_0 \Rightarrow a,\ \sigma_0 \qquad \sigma_0 \vdash_{escape} e_1 \Rightarrow v_k,\ \sigma_1 \qquad \texttt{pop}(\sigma_1,\ a,\ v_k) = (v,\ \sigma_2)}{\sigma \vdash \texttt{pop}\ e_0\ e_1 \Rightarrow v,\ \sigma_2}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow a_0,\ \sigma_0 \qquad \texttt{copyObj}(\sigma_0,\ a_0) = (a_1,\ \sigma_1)}{\sigma \vdash \texttt{copy-obj}\ e \Rightarrow a_1,\ \sigma_1} \qquad \frac{\sigma \vdash_{escape} e \Rightarrow a_0,\ \sigma_0 \qquad \texttt{keys}(\sigma_0,\ a_0) = (a_1,\ \sigma_1)}{\sigma \vdash \texttt{map-keys}\ e \Rightarrow a_1,\ \sigma_1}$$

$$\frac{\sigma \vdash_{escape} e_0 \Rightarrow a_l,\ \sigma_0 \qquad \sigma_0 \vdash_{escape} e_1 \Rightarrow v,\ \sigma_1 \qquad \texttt{contains}(\sigma_1,\ a_l,\ v) = b}{\sigma \vdash \texttt{contains}\ e_0\ e_1 \Rightarrow b,\ \sigma_1}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow s,\ \sigma_0 \qquad \texttt{convert}(\triangleright,\ s,\ e^*) = (v,\ \sigma_1)}{\sigma \vdash \texttt{convert}\ e\ \triangleright\ e^* \Rightarrow v,\ \sigma_1}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow s_0,\ \sigma_0 \quad \texttt{equals}(s_0,\ s) = b}{\sigma \vdash \texttt{is-instance-of}\ e\ s \Rightarrow b,\ \sigma_0}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow ASTVal,\ \sigma_0 \quad \texttt{isKindOf}(ASTVal,\ s) = b}{\sigma \vdash \texttt{is-instance-of}\ e\ s \Rightarrow b,\ \sigma_0}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow ASTVal,\ \sigma_0 \quad \texttt{toString}(ASTVal) = s}{\sigma \vdash \texttt{get-syntax}\ e \Rightarrow s,\ \sigma_0}$$

$$\frac{\sigma \vdash_{escape} e \Rightarrow ASTVal,\ \sigma_0 \quad \texttt{getElems}(ASTVal) = v^* \quad \texttt{allocList}(\sigma_0,\ v^*) = (a,\ \sigma_1)}{\sigma \vdash \texttt{get-elems}\ e\ s \Rightarrow a,\ \sigma_1}$$

$$\frac{\begin{array}{c}\sigma \vdash_{escape} e_c \Rightarrow ASTVal,\ \sigma_0 \quad \sigma_0 \vdash_{escape} e_r \Rightarrow s,\ \sigma_1 \quad \texttt{assertValidParseRule}(s) \\ \texttt{getNewValue}(ASTVal,\ s) = v\end{array}}{\sigma \vdash \texttt{parse-syntax}\ e_c\ e_r\ e^* \Rightarrow v,\ \sigma_1}$$

$$\frac{\begin{array}{c}\sigma \vdash_{escape} e_c \Rightarrow s_c,\ \sigma_c \quad \sigma_c \vdash_{escape} e_r \Rightarrow s_r,\ \sigma_r \\ \sigma_r \vdash e_0 \Rightarrow b_0,\ \sigma_0 \quad \cdots \quad \sigma_{n-1} \vdash e_n \Rightarrow b_n,\ \sigma_n \quad \texttt{getNewValue}(s_c,\ s_r,\ b^*) = v\end{array}}{\sigma \vdash \texttt{parse-syntax}\ e_c\ e_r\ e_0\ \cdots\ e_n \Rightarrow v,\ \sigma_n}$$

TODO
reference + handle escapeCompletion