A Logic-preserving Compiler for JavaScript Verification - Supplemental Materials

July 27, 2016

Chapter 1

ES5 Strict Semantics

1.1 Notation

Functions and Judgements

$\mathsf{dom}(h)$	Domain of the heap.	$\mathsf{ER}(l.\mathtt{a}x)$	Environment record reference.
$(l,x) \mapsto v$	Heap cell.	$\mathsf{fun}_m(l, L, \overline{x}, \overline{s}, l')$	New function at l.
$h(l,x)$	Heap read operation.	$\mathsf{act}_m(l_s, \overline{x}, \overline{v}, s)$	New activation record at l_s.
$h[(l,x) \mapsto v]$	Heap write/add operation.	$\mathsf{defs}(s)$	Declared variables.
$h \backslash l.x$	Heap cell deallocation.	$\mathsf{SelectProto}(l)$	Determine prototype for new.
$h \uplus h'$	Disjoint heap composition.	$\mathsf{SelectThis}(l.\mathtt{a}x)$	Determine this for function call.
$\pi(h,l,x)$	Prototype resolution.	$\mathsf{FunRet}(o)$	Determine function outcome.
$\sigma(h,L,x)$	Scope resolution.	$\mathsf{ConsRet}(o)$	Determine constructor outcome.
$\gamma(h, l.\mathtt{a}x)$	Dereferencing values.	$\mathsf{False}(v)$	Determine falsity.
$\mathsf{RO}(h, l.\mathtt{a}x)$	Read-only reference.	$\mathsf{parse}(m)$	Parse m as JavaScript code.

1.2 Semantics

Prototype resolution and prototype check: $\pi(h,l,x) = v$ **and** $\pi_o(h,l,l') = b$

$$\pi(h, \mathsf{null}, x) \triangleq \bot \qquad \frac{(l,x) \in \mathsf{dom}(h)}{\pi(h,l,x) \triangleq h(l,x)} \qquad \frac{(l,x) \notin \mathsf{dom}(h) \quad l' = h(l, @proto)}{\pi(h,l,x) \triangleq \pi(h,l',x)}$$

$$\pi_o(h,l,\mathsf{null}) \triangleq \mathsf{false} \qquad \pi_o(h,l,l) \triangleq \mathsf{true} \qquad \frac{l \neq l' \quad l'' = h(l, @proto)}{\pi_o(h,l,l') \triangleq \pi_o(h,l'',l')}$$

Scope Resolution: $\sigma(h,L,x) = l$

$$\frac{(l,x) \in \mathsf{dom}(h)}{\sigma(h,(l,m)::L,x) \triangleq (l,m)} \qquad \frac{(l,x) \notin \mathsf{dom}(h)}{\sigma(h,(l,m)::L,x) \triangleq \sigma(h,L,x)} \qquad \frac{\pi(h,l_g,x) = l'}{\sigma(h,[\,],x) \triangleq l_g} \qquad \frac{\pi(h,l_g,x) = \bot}{\sigma(h,[\,],x) \triangleq \bot}$$

Dereferencing Values: $\gamma(h,r) = v$

$$\frac{v \notin \mathcal{R}}{\gamma(h,v) \triangleq v} \qquad \frac{\pi(h,l,x) = \bot}{\gamma(h, l.\mathtt{o}x) \triangleq \mathsf{undefined}} \qquad \frac{l \neq l_g}{\gamma(h, l.\mathtt{v}x) \triangleq h(l,x)} \qquad \frac{l \notin \mathcal{L}_{var} \quad \pi(h,l,x) = v}{\gamma(h, l.\mathtt{a}x) \triangleq v}$$

$$\frac{\pi(h,l_g,x) = \bot}{\gamma(h, l_g.\mathtt{v}x) \triangleq \bot} \qquad \gamma(h, \bot.\mathtt{a}x) \triangleq \bot$$

Heap update and cell deallocation: $h[r \mapsto v]$ **and** $h \backslash r$

$$\frac{(l,x) \notin \mathsf{dom}(h)}{h[(l,x) \mapsto v] \triangleq h \uplus (l,x) \mapsto v} \qquad \frac{h = h' \uplus (l,x) \mapsto v'}{h[(l,x) \mapsto v] \triangleq h \uplus (l,x) \mapsto v}$$

$$\frac{(l,x) \notin \mathsf{dom}(h)}{h \backslash l.x \triangleq h} \qquad \frac{h = h' \uplus (l,x) \mapsto v}{h \backslash l.x \triangleq h'}$$

Extended Syntax

$$\begin{array}{rcl}
v \in \mathcal{V} & ::= & \lambda \mid \lambda\overline{x}.s \mid l \\
o \in \mathcal{O} & ::= & v \mid \mathsf{ret}\, v \mid \mathsf{error} \mid \mathsf{empty} \\
e \in \mathcal{E}_{\mathrm{JS}} & ::= & \lambda \mid \mathsf{this} \mid x \mid \{\,\} \mid e_1 \oplus e_2 \mid o \oplus_1 e \mid v \oplus_2 o \mid e_1 = e_2 \mid o =_1 e \mid l.\mathtt{a}x =_2 o \mid e_1[e_2] \mid o[e]_1 \mid l[o]_2 \mid \mathsf{delete}\, e \\
 & & \mid \mathsf{del}_1\, o \mid \mathsf{function}\, (\overline{x})\{s\}^m \mid e(\overline{e}) \mid o(\overline{e})_1 \mid l.\mathtt{a}x(\overline{v})_2 \mid \mathsf{new}\; e(\overline{e}) \mid \mathsf{new}\; e(\overline{e}) \mid \mathsf{new}_1\; o(\overline{e}) \mid \mathsf{new}_2\; l(\overline{v}) \\
s \in \mathcal{S}_{\mathrm{JS}} & ::= & e \mid \mathsf{var}\, x \mid s_1;\, s_2 \mid \mathsf{seq}_1(s,o) \mid \mathsf{seq}_2(o,o') \mid \mathsf{if}(e)\,\{s_1\}\,\mathsf{else}\,\{s_2\} \mid \mathsf{if}_1(o)\,\{s_1\}\,\mathsf{else}\,\{s_2\} \\
 & & \mid \mathsf{while}(e)\,\{s\} \mid \mathsf{while}_1(e)\{s,o\} \mid \mathsf{while}_2(v,e)\{s,o\} \mid \mathsf{while}_3(e)\{s,o,o'\} \mid \mathsf{return}\, e \mid \mathsf{return}_1\, o \\
s_\lightning \in \mathcal{S}tmt_\lightning & ::= & \mathsf{error} \oplus_1 e \mid v \oplus_2 \mathsf{error} \mid \mathsf{error} =_1 e \mid r =_2 \mathsf{error} \mid \mathsf{error}[e]_1 \mid v[\mathsf{error}]_2 \mid \mathsf{del}_1\, \mathsf{error} \mid \mathsf{error}(\overline{e})_1 \\
 & & \mid \mathsf{new}_1\; \mathsf{error}(\overline{e}) \mid \mathsf{iterate}_1\{\overline{e}, L :: \mathsf{error}\} \mid \mathsf{seq}_1(s,\mathsf{error}) \mid \mathsf{seq}_2(o,\mathsf{error}) \mid \mathsf{if}_1(\mathsf{error})\,\{s_1\}\,\mathsf{else}\,\{s_2\} \\
 & & \mid \mathsf{while}_2(\mathsf{error},e)\{s,o\} \mid \mathsf{while}_3(e)\{s,o,\mathsf{error}\} \mid \mathsf{return}_1\, \mathsf{error}
\end{array}$$

PBS Semantics for Expressions: $L, l \vdash \langle h, e\rangle \Downarrow \langle h, o\rangle$

Variable
$$\frac{\sigma(h, L, x) = (l, m)}{L, \_ \vdash \langle h, x\rangle \Downarrow \langle h, l.\mathsf{v}x\rangle}$$

Object Literal
$$\frac{h' = h \uplus (l, @proto) \mapsto l_{op}}{\vdash \langle h, \{\,\}\,\rangle \Downarrow \langle h', l\rangle}$$

This
$$\frac{}{\_, l \vdash \langle h, \mathsf{this}\rangle \Downarrow \langle h, l\rangle}$$

This (Fault)
$$\frac{}{\_, \mathsf{undefined} \vdash \langle h, \mathsf{this}\rangle \Downarrow \langle h, \mathsf{error}\rangle}$$

Literal
$$\vdash \langle h, \lambda\rangle \Downarrow \langle h, \lambda\rangle$$

Binary Operator
$$\frac{L, l \vdash \langle h, e_1\rangle \Downarrow^{\gamma} \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, o_1 \oplus_1 e_2\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, e_1 \oplus e_2\rangle \Downarrow \langle h_f, o_f\rangle}$$

Binary Operator - 1
$$\frac{L, l \vdash \langle h_1, e_2\rangle \Downarrow^{\gamma} \langle h_2, o_2\rangle \quad L, l \vdash \langle h_2, v_1 \oplus_2 o_2\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h_1, v_1 \oplus_1 e_2\rangle \Downarrow \langle h_f, o_f\rangle}$$

Binary Operator - 2
$$\frac{v = v_1 \mathbin{\overline{\oplus}} v_2}{\vdash \langle h_2, v_1 \oplus_2 v_2\rangle \Downarrow \langle h_2, v\rangle}$$

Assignment
$$\frac{L, l \vdash \langle h, e_1\rangle \Downarrow \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, o_1 =_1 e_2\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, e_1 = e_2\rangle \Downarrow \langle h_f, o_f\rangle}$$

Assignment - 1
$$\frac{l \neq \bot \quad L, l' \vdash \langle h_1, e_2\rangle \Downarrow^{\gamma} \langle h_2, o_2\rangle \quad L, l' \vdash \langle h_2, l.\mathsf{a}x =_2 o_2\rangle \Downarrow \langle h_f, o_f\rangle}{L, l' \vdash \langle h_1, l.\mathsf{a}x =_1 e_2\rangle \Downarrow \langle h_f, o_f\rangle}$$

Assignment - 2
$$\frac{l \in \mathsf{locs}(h) \quad h' = h_2[(l, x) \mapsto v] \quad (x \neq \mathsf{eval} \vee \neg\mathsf{ER}(l.\mathsf{a}x))}{\vdash \langle h_2, l.\mathsf{a}x =_2 v\rangle \Downarrow \langle h', v\rangle}$$

Assignment - 2 (Fault)
$$\frac{l = \bot \vee l = \mathsf{undefined} \vee l = \mathsf{null}}{\vdash \langle h_2, l.\mathsf{a}x =_2 v\rangle \Downarrow \langle h_2, \mathsf{error}\rangle}$$

Field Access
$$\frac{L, l \vdash \langle h, e_1\rangle \Downarrow^{\gamma} \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, o_1[e_2]_1\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, e_1[e_2]\rangle \Downarrow \langle h_f, o_f\rangle}$$

Field Access - 1
$$\frac{L, l \vdash \langle h_1, e_2\rangle \Downarrow^{\gamma} \langle h_2, o_2\rangle \quad L, l \vdash \langle h_2, v_1[o_2]_2\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h_1, v_1[e_2]_1\rangle \Downarrow \langle h_f, o_f\rangle}$$

Field Access - 2
$$\vdash \langle h_2, l[x]_2\rangle \Downarrow \langle h_2, l.\mathsf{o}x\rangle$$

Field Access - 2 (Fault)
$$\frac{v_1 = \mathsf{null} \vee v_1 = \mathsf{undefined}}{\vdash \langle h_2, v_1[x]_2\rangle \Downarrow \langle h_2, \mathsf{error}\rangle}$$

Delete
$$\frac{L, l \vdash \langle h, e\rangle \Downarrow \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, \mathsf{del}_1\, o_1\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, \mathsf{delete}\, e\rangle \Downarrow \langle h_f, o_f\rangle}$$

Delete - 1 (false)
$$\frac{(l, x) \notin \mathsf{dom}(h_1)}{\vdash \langle h_1, \mathsf{del}_1\, l.\mathsf{a}x\rangle \Downarrow \langle h, \mathsf{false}\rangle}$$

Delete - 1 (true)
$$\frac{(l, x) \in \mathsf{dom}(h_1) \quad \neg\mathsf{RO}(h_1, l.\mathsf{a}x) \quad h' = h_1 \backslash l.x}{\vdash \langle h_1, \mathsf{del}_1\, l.\mathsf{a}x\rangle \Downarrow \langle h', \mathsf{true}\rangle}$$

Delete - 1 (Fault)
$$\frac{l = \mathsf{undefined} \vee \mathsf{RO}(l.\mathsf{a}x)}{\vdash \langle h_1, \mathsf{del}_1\, l.\mathsf{a}x\rangle \Downarrow \langle h_1, \mathsf{error}\rangle}$$

Function Literal
$$\frac{h' = h \uplus (l', @proto) \mapsto l_{op} \uplus \mathsf{fun}(l'', L, \overline{x}, s, l')}{L, l \vdash \langle h, \mathsf{function}\,(\overline{x})\{s\}\rangle \Downarrow \langle h', l''\rangle}$$

Function Call
$$\frac{L, l \vdash \langle h, e\rangle \Downarrow \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, o_1(\overline{e})_1\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, e(\overline{e})\rangle \Downarrow \langle h_f, o_f\rangle}$$

Function Call - 1
$$\frac{L, l' \vdash \langle h_1, \mathsf{iterate}\{\overline{e}\}\rangle \Downarrow \langle h_2, \overline{v}\rangle \quad L, l' \vdash \langle h_2, l.\mathsf{a}x(\overline{v})_2\rangle \Downarrow \langle h_f, o_f\rangle}{L, l' \vdash \langle h_1, l.\mathsf{a}x(\overline{e})_1\rangle \Downarrow \langle h_f, o_f\rangle}$$

Function Call - 2
$$\frac{l' = \gamma(h_2, l.\mathsf{a}x) \quad \lambda\overline{x}.s^m = h_2(l', @body) \quad L = h_2(l', @scope) \quad l'' = \mathsf{SelectThis}(l.\mathsf{a}x) \quad (l_s, m) :: L, l'' \vdash \langle h_2 \uplus \mathsf{act}_m(l_s, \overline{x}, \overline{v}, s), s\rangle \Downarrow \langle h', o\rangle}{\vdash \langle h_2, l.\mathsf{a}x(\overline{v})_2\rangle \Downarrow \langle h', \mathsf{FunRet}(o)\rangle}$$

Function Call - 2 (Fault)
$$\frac{\gamma(h_2, l.\mathsf{a}x) = v \quad v \notin \mathcal{L} \vee (v, @body) \notin \mathsf{dom}(h_2)}{\vdash \langle h_2, l.\mathsf{a}x(\overline{v})_2\rangle \Downarrow \langle h, \mathsf{error}\rangle}$$

Constructor Call
$$\frac{L, l \vdash \langle h, e\rangle \Downarrow \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, \mathsf{new}_1\, o_1(\overline{e})\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, \mathsf{new}\ e(\overline{e})\rangle \Downarrow \langle h_f, o_f\rangle}$$

Constructor Call - 1
$$\frac{L, l' \vdash \langle h_1, \mathsf{iterate}\{\overline{e}\}\rangle \Downarrow \langle h_2, \overline{v}\rangle \quad L, l' \vdash \langle h_2, \mathsf{new}_1\, l(\overline{v})\rangle \Downarrow \langle h_f, o_f\rangle}{L, l' \vdash \langle h_1, \mathsf{new}_1\, l(\overline{e})\rangle \Downarrow \langle h_f, o_f\rangle}$$

Constructor Call - 2 (Fault)
$$\frac{(l, @body) \notin \mathsf{dom}(h_2)}{\vdash \langle h_2, \mathsf{new}_2\, l(\overline{v})\rangle \Downarrow \langle h_2, \mathsf{error}\rangle}$$

Constructor Call - 2
$$\frac{\lambda\overline{x}.s^m = h_2(l, @body) \quad L = h_2(l, @scope) \quad v = h_2(l, prototype) \quad l' = \mathsf{SelectProto}(v) \quad h' = h_2 \uplus (l_o, @proto) \mapsto l' \uplus \mathsf{act}_m(l_s, \overline{x}, \overline{v}, s) \quad l \neq l_e \quad (l_s, m) :: L, l_o \vdash \langle h', s\rangle \Downarrow \langle h'', o\rangle}{\vdash \langle h_2, \mathsf{new}_2\, l(\overline{v})\rangle \Downarrow \langle h'', \mathsf{ConsRet}(o, l_o)\rangle}$$

PBS Semantics for Statements: $L, l \vdash \langle h, s\rangle \Downarrow \langle h, o\rangle$

Var Decl.
$$\vdash \langle h, \mathsf{var}\, x\rangle \Downarrow \langle h, \mathsf{empty}\rangle$$

Sequence
$$\frac{L, l \vdash \langle h, s_1\rangle \Downarrow^{\gamma} \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, \mathsf{seq}_1(o_1, s_2)\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, s_1; s_2\rangle \Downarrow \langle h_f, \mathsf{empty}\rangle}$$

Sequence - 1
$$\frac{o \neq \mathsf{error} \quad o \neq \mathsf{ret}\, v \quad L, l \vdash \langle h_1, s_2\rangle \Downarrow \langle h_2, o_2\rangle \quad L, l \vdash \langle h_2, \mathsf{seq}_2(o_1, o_2)\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h_1, \mathsf{seq}_1(o_1, s_2)\rangle \Downarrow \langle h_f, o_f\rangle}$$

Sequence - 1 (return)
$$\vdash \langle h_1, \mathsf{seq}_1(\mathsf{ret}\, v, s_2)\rangle \Downarrow \langle h_1, \mathsf{ret}\, v\rangle$$

Sequence - 2 (empty)
$$\vdash \langle h_2, \mathsf{seq}_2(\mathsf{empty}, o)\rangle \Downarrow \langle h_2, o\rangle$$

Sequence - 2 (non-empty)
$$\vdash \langle h_2, \mathsf{seq}_2(o, \mathsf{empty})\rangle \Downarrow \langle h_2, o\rangle$$

If

$$\frac{L, l \vdash \langle h, e\rangle \Downarrow^{\gamma} \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, \mathsf{if}_1(o_1)\,\{s_1\}\,\mathsf{else}\,\{s_2\}\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, \mathsf{if}(e)\,\{s_1\}\,\mathsf{else}\,\{s_2\}\rangle \Downarrow \langle h_f, o_f\rangle}$$

If - 1 (true)

$$\frac{\neg\mathsf{False}(v) \quad L, l \vdash \langle h_1, s_1\rangle \Downarrow^{\gamma} \langle h_f, o_f\rangle}{L, l \vdash \langle h_1, \mathsf{if}_1(v)\,\{s_1\}\,\mathsf{else}\,\{s_2\}\rangle \Downarrow \langle h_f, o_f\rangle}$$

If - 1 (false)

$$\frac{\mathsf{False}(v) \quad L, l \vdash \langle h_1, s_2\rangle \Downarrow^{\gamma} \langle h_f, o_f\rangle}{L, l \vdash \langle h_1, \mathsf{if}_1(v)\,\{s_1\}\,\mathsf{else}\,\{s_2\}\rangle \Downarrow \langle h_f, o_f\rangle}$$

While

$$\frac{L, l \vdash \langle h, \mathsf{while}_1(e)\{s, \mathsf{empty}\}\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, \mathsf{while}(e)\,\{s\}\rangle \Downarrow \langle h_f, o_f\rangle}$$

While - 1

$$\frac{L, l \vdash \langle h, e\rangle \Downarrow^{\gamma} \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, \mathsf{while}_2(o_1, e)\{s, o\}\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, \mathsf{while}_1(e)\{s, o\}\rangle \Downarrow \langle h_2, o_2\rangle}$$

While - 2 (true)

$$\frac{\neg\mathsf{False}(v) \quad L, l \vdash \langle h_1, s\rangle \Downarrow^{\gamma} \langle h_2, o_2\rangle \quad L, l \vdash \langle h_2, \mathsf{while}_3(e)\{s, o, o_2\}\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h_1, \mathsf{while}_2(v, e)\{s, o\}\rangle \Downarrow \langle h_f, o_f\rangle}$$

While - 2 (false)

$$\frac{\mathsf{False}(v)}{\vdash \langle h_1, \mathsf{while}_2(v, e)\{s, o\}\rangle \Downarrow \langle h_1, o\rangle}$$

While - 3 (value)

$$\frac{L, l \vdash \langle h_2, \mathsf{while}_1(e)\{s, v\}\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h_2, \mathsf{while}_3(e)\{s, o, v\}\rangle \Downarrow \langle h_f, o_f\rangle}$$

While - 3 (return)

$$\vdash \langle h, \mathsf{while}_3(e)\{s, o, \mathsf{ret}\,v\}\rangle \Downarrow \langle h, \mathsf{ret}\,v\rangle$$

While - 3 (empty)

$$\frac{L, l \vdash \langle h_2, \mathsf{while}_1(e)\{s, o\}\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h_2, \mathsf{while}_3(e)\{s, o, \mathsf{empty}\}\rangle \Downarrow \langle h_f, o_f\rangle}$$

Return

$$\frac{L, l \vdash \langle h, e\rangle \Downarrow^{\gamma} \langle h_1, o_1\rangle \quad L, l \vdash \langle h_1, \mathsf{return}_1\, o_1\rangle \Downarrow \langle h_f, o_f\rangle}{L, l \vdash \langle h, \mathsf{return}\, e\rangle \Downarrow \langle h_f, o_f\rangle}$$

Return - 1

$$\vdash \langle h_1, \mathsf{return}_1\, v\rangle \Downarrow \langle h_1, \mathsf{ret}\, v\rangle$$

Iterate

$$\frac{L, l \vdash \langle h, \mathsf{iterate}_1\{\overline{e}, [\,]\}\rangle \Downarrow \langle h_f, \overline{v_f}\rangle}{L, l \vdash \langle h, \mathsf{iterate}\{\overline{e}\}\rangle \Downarrow \langle h_f, \overline{v_f}\rangle}$$

Iterate - 1(non-empty)

$$\frac{L, l \vdash \langle h, e\rangle \Downarrow^{\gamma} \langle h_1, v\rangle \quad L, l \vdash \langle h_1, \mathsf{iterate}_1\{\overline{e}, \overline{v} :: v\}\rangle \Downarrow \langle h_f, \overline{v_f}\rangle}{L, l \vdash \langle h, \mathsf{iterate}_1\{e :: \overline{e}, \overline{v}\}\rangle \Downarrow \langle h_f, \overline{v_f}\rangle}$$

Iterate - 1 (empty)

$$\vdash \langle h, \mathsf{iterate}_1\{[\,], \overline{v}\}\rangle \Downarrow \langle h, \overline{v}\rangle$$

Error Propagation

$$\vdash \langle h, s_{\lightning}\rangle \Downarrow \langle h, \mathsf{error}\rangle$$

Auxiliary Functions

$$l \mapsto \{x_1 \colon v_1, \ldots, x_n \colon v_n\} \triangleq (l, x_1) \mapsto v_1 \uplus \cdots \uplus (l, x_n) \mapsto v_n$$

$$\mathsf{RO}(h, l.{}_{\mathsf{a}}x) \triangleq \mathsf{a} = \mathsf{v} \vee (x = \mathit{prototype} \wedge (l, @\mathit{body}) \in \mathsf{dom}(h))$$

$$\mathsf{ER}(l.{}_{\mathsf{a}}x) \triangleq \mathsf{a} = \mathsf{v}$$

$$\mathsf{fun}(l, L, \overline{x}, \overline{s}, l', m) \triangleq l \mapsto \{@\mathit{proto} : l_{fp}, \mathit{prototype} : l', @\mathit{scope} : L, @\mathit{body} : \lambda\overline{x}.s^m\}$$

$$\mathsf{act}(l_s, \overline{x}, \overline{v}, s) \triangleq (\uplus_{i=1}^{n}(l_s, \overline{x}_i) \mapsto \overline{v}_i) \uplus (\uplus_{i=1}^{m}(l_s, y_i) \mapsto \mathsf{undefined}),\ \text{where: } \mathsf{defs}(s) = \{y_1, ..., y_m\}$$

$$\mathsf{False}(v) \triangleq v \in \{0, \mathsf{false}, \text{“”}, \mathsf{null}, \mathsf{undefined}\}$$

$$\mathsf{SelectProto}(l) \triangleq \begin{cases} l & \text{if } l \in \mathcal{L} \\ l_g & \text{otherwise} \end{cases}$$

$$\mathsf{SelectThis}(l.{}_{\mathsf{a}}x) \triangleq \begin{cases} l & \text{if } \mathsf{a} = \mathsf{o} \\ \mathsf{undefined} & \text{otherwise} \end{cases}$$

$$\mathsf{FunRet}(o) \triangleq \begin{cases} \mathsf{undefined} & \text{if } o = v \text{ or } o = \mathsf{empty} \\ v & \text{if } o = \mathsf{ret}\, v \\ \mathsf{error} & \text{if } o = \mathsf{error} \end{cases}$$

$$\mathsf{ConsRet}(o, l) \triangleq \begin{cases} l & \text{if } o = v \text{ or } o = \mathsf{empty} \\ v & \text{if } o = \mathsf{ret}\, v \\ \mathsf{error} & \text{if } o = \mathsf{error} \end{cases}$$

$$\mathsf{defs}(s) \triangleq \begin{cases} \emptyset & \text{if } s \in \mathcal{E}_{\mathsf{JS}} \\ \{x\} & \text{if } s = \mathsf{var}\, x \\ \mathsf{defs}(s_1) \cup \mathsf{defs}(s_2) & \text{if } s = s_1; s_2 \\ \mathsf{defs}(s_1) \cup \mathsf{defs}(s_2) & \text{if } s = \mathsf{if}(e)\,\{s_1\}\,\mathsf{else}\,\{s_2\} \\ \mathsf{defs}(s) & \text{if } s = \mathsf{while}(e)\,\{s\} \\ \emptyset & \text{if } s = \mathsf{return}\, e \end{cases}$$

1.3 Closure Clarification

We begin by defining the scope clarification function constructor Φ.

Definition 1 (Scope Clarification Construction $\Phi_m(s,\psi)=\psi'$)**.**

$$\Phi_m(s,\psi)=\begin{cases}\psi & \text{if } s=\lambda \vee s=\mathsf{this} \vee s=\{\,\} \vee s\in\mathcal{X}_{\mathsf{JS}}\\ \psi & \text{if } s=\mathsf{var}\,x \wedge x\in\mathcal{X}_{\mathsf{JS}}\\ \Phi_m(e,\psi) & \text{if } s=\mathsf{delete}\,e \vee s=\mathsf{return}\,e\\ \Phi_m(e_2,\Phi_m(e_1,\psi)) & \text{if } s=e_1\oplus e_2 \vee s=(e_1=e_2) \vee s=e_1[e_2]\\ \Phi_m(e_n,\Phi_m(\ldots,\Phi_m(e_0,\psi))) & \text{if } s=e_0(e_1,...,e_n)\\ \Phi_m(e_n,\Phi_m(\ldots,\Phi_m(e_0,\psi))) & \text{if } s=\mathsf{new}\ e_0(e_1,...,e_n)\\ \Phi_m(s_2,\Phi_m(s_1,\psi)) & \text{if } s=s_1;\ s_2 \vee s=\mathsf{if}(e)\,\{s_1\}\,\mathsf{else}\,\{s_2\}\\ \Phi_m(s,\Phi_m(e,\psi)) & \text{if } s=\mathsf{while}(e)\,\{s\}\\ \Phi_{m'}(s',\hat{\psi}) & \text{if } \begin{cases} s=\mathsf{function}\,(\overline{x})\{s'\}^{m'}\\ \hat{\psi}=\psi\left[\begin{array}{l}(m,x_i)\mapsto m'\mid_{x_i\in\overline{x}\,\cup\,\mathsf{defs}(s')}\\ (m,y_j)\mapsto\psi(m,y_j)\mid_{y_j\in\mathsf{dom}(\psi(m))}\end{array}\right]\end{cases}\end{cases}$$

In the following, we use $\leq$ for function extension.

Lemma 1 (Properties of Φ)**.** *For any statement s, closure clarification functions ψ and ψ', and id m:*

- Monotonicity: $\psi\leq\psi'\Rightarrow\Phi_m(s,\psi)\leq\Phi_m(s,\psi')$
- Idempotence: $\Phi_m(s,\Phi_m(s,\psi))=\Phi_m(s,\psi)$
- Extensiveness: $\psi\leq\Phi_m(s,\psi))$

Proof. All three properties are proved by structural induction on s, assuming that there are no clashes of function identifiers. □

Definition 2 (ψ-compatibility)**.** Scope chain ψ-compatibility and heap ψ-compatibility are defined as follows:

- *Non-empty scope chain ψ-compatibility:* Given a heap h and a non-empty scope chain $L=(\_,m)::L'$, (h,L) is ψ-compatible, for a given scope clarification function ψ, if:

  $$\forall x\in\mathcal{X}_{\mathsf{JS}}.\ \psi(m,x)=m'\Leftrightarrow\exists l.\ \sigma(h,L,x)=(l,m')$$

  Empty scope chain ψ-compatibility: The empty scope chain is ψ-compatible for every heap h and scope clarification function ψ. Formally:

  $$\forall h\in\mathcal{H}.\ (h,[\,])\text{ is }\psi\text{-compatible}$$

- *Heap ψ-compatibility:* A heap h is ψ-compatible if **(1)** all non-empty scope chains in the range of h are ψ-compatible and **(2)** ψ contains the scoping information about all the lambda abstractions in the heap. Formally, for any location l, scope chain L, and lambda abstraction $\lambda\overline{x}.s^m$ such that $(l,@scope)\mapsto L\uplus(l,@body)\mapsto\lambda\overline{x}.s^m$, it follows that: **(1)** (h,L) is ψ-compatible and **(2)** $\Phi_m(s,\psi)=\psi$.

In the following lemma, we use the function $\mathsf{firstFID}$ that given a scope chain L, returns the function identifier annotation corresponding to the location of the first environment record in L; formally:

$$\mathsf{firstFID}(L)=\begin{cases}m & \text{if } L=(l,m)::L'\\ \mathsf{main} & \text{if } L=[\,]\end{cases}$$

Lemma 2 (Correctness of Φ)**.** *For any scope chain L, location l_{this}, heaps h and h_f, statement s, outcome o, and scope clarification function ψ, such that:*

- *h is ψ-compatible*
- *(h,L) is ψ-compatible*
- *$\Phi_m(s,\psi)=\psi'$, for $m=\mathsf{firstFID}(L)$*
- *$L,l_{this}\vdash\langle h,s\rangle\Downarrow\langle h_f,o\rangle$*

It holds that: h_f and (h_f,L) are ψ'-compatible.

Proof. For convenience, we name the hypotheses:

- **Hyp.** 1: h is ψ-compatible
- **Hyp.** 2: (h,L) is ψ-compatible
- **Hyp.** 3: $\Phi_m(s,\psi)=\psi'$, for $m=\mathsf{firstFID}(L)$
- **Hyp.** 4: $L,l_{this}\vdash\langle h,s\rangle\Downarrow\langle h_f,o\rangle$

We proceed by induction on the derivation of $L,l_{this}\vdash\langle h,s\rangle\Downarrow\langle h_f,o\rangle$. We only focus on the cases: [Function Call - 2], [Constructor Call - 2], and [Function Literal], because those are the only rules that create scope chains and/or store them in the heap. Furthermore, since the argumentation for [Constructor Call - 2] is analogous to that of [Function Call - 2], we will address in detail only [Function Call - 2] and [Function Literal].

[Function Call - 2] We conclude that:

- $s=l.\mathsf{a}x(\overline{v})_2$

- $l' = \gamma(h, l.\mathsf{a}x)$
- $\lambda\overline{x}.s^{m'} = h(l', @body)$
- $L' = h(l', @scope)$
- $l'' = \mathsf{SelectThis}(l.\mathsf{a}x)$
- $(l_s, m') :: L', l'' \vdash \langle h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s), s' \rangle \Downarrow \langle h_f, o' \rangle$

Where $o = \mathsf{FunRet}(o')$. Note that the four hypotheses required for applying the induction hypothesis hold, as we justify below:

1. $h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s')$ is ψ-compatible. This follows immediately from the fact that h is ψ-compatible and and $\mathsf{act}(l_s, \overline{x}, \overline{v}, s)$ does not contain any scope chains in its range.

2. $(h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s'), (l_s, m') :: L')$ is ψ-compatible. We have to prove that for every variable x, it holds that:

$$\psi(m', x) = m'' \Leftrightarrow \exists l.\ \sigma(h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s'), (l_s, m') :: L', x) = (l, m'')$$

   We start by proving the left-to-right direction of the equivalence. There are two cases to consider: $\psi(m', x) = m'$ and $\psi(m', x) \neq m'$.

   - If $\psi(m', x) = m'$, we conclude, using the definition of Φ, that $x \in \overline{x} \cup \mathsf{defs}(s')$. Hence, using the definitions of act and σ, we conclude that:

$$\sigma(h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s'), (l_s, m') :: L', x) = (l_s, m')$$

   - If $\psi(m', x) \neq m'$, we conclude, using the definition of Φ, that $x \notin \overline{x} \cup \mathsf{defs}(s')$. Hence, using the definitions of act and σ, we conclude that:

$$\sigma(h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s'), (l_s, m') :: L', x) = \sigma(h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s'), L', x)$$

     Since the same environment record cannot appear more than once in a given scope chain, we conclude that:

$$\sigma(h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s'), (l_s, m') :: L', x) = \sigma(h, L', x)$$

     Since $\psi(m', x) \neq m'$, we conclude, using the definition of Φ, that: $\psi(m', x) = \psi(\hat{m}, x)$, where $\hat{m}$ is the id of the function in which the function with id m' is defined. We know, by inspection of the semantics, that $\hat{m}$ coincides with $\mathsf{firstFID}(L')$, which implies that: $\psi(m', x) = \psi(\mathsf{firstFID}(L'), x) = m''$. Because (h, L) is ψ-compatible (which we know from **Hyp.** 1), we conclude that:

$$\exists l.\ \sigma(h, L, x) = (l, m'')$$

     which implies the result.

   For the right-to-left direction of the equivalence, there are two cases to consider:

   - If $\sigma(h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s'), (l_s, m') :: L', x) = (l_s, m')$, we conclude, using the definition of σ and act, that $x \in \overline{x} \cup \mathsf{defs}(s')$. Hence, using the definition of Φ, we conclude that $\psi(m', x) = m'$.
   - If $\sigma(h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s'), (l_s, m') :: L', x) = (l, m'') \neq (l_s, m')$, we conclude, using the definition of σ and act, that $x \notin \overline{x} \cup \mathsf{defs}(s')$ and that:

$$\sigma(h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s'), (l_s, m') :: L', x) = \sigma(h, L', x) = (l, m'')$$

     Because $x \notin \overline{x} \cup \mathsf{defs}(s')$, we conclude, using the definition of Φ, that $\psi(m', x) = \psi(\hat{m}, x)$, where $\hat{m}$ is the id of the function in which the function with id m' is defined. We know, by inspection of the semantics, that $\hat{m}$ coincides with $\mathsf{firstFID}(L')$, which implies that: $\psi(m', x) = \psi(\mathsf{firstFID}(L'), x)$. Because (h, L) is ψ-compatible (which we know from **Hyp.** 1) and recalling that $\sigma(h, L', x) = (l, m'')$, we conclude that: $\psi(m', x) = \psi(\mathsf{firstFID}(L'), x) = m''$, which implies the result.

3. $\Phi_{m'}(s', \psi) = \psi$: follows immediately from **Hyp.** 1
4. $(l_s, m') :: L', l'' \vdash \langle h \uplus \mathsf{act}(l_s, \overline{x}, \overline{v}, s), s \rangle \Downarrow \langle h_f, o' \rangle$: follows immediately from **Hyp.** 4

We can now apply the induction hypothesis and conclude that: h_f and (h_f, L) are ψ'-compatible, where $\psi' = \psi$.

[Function Literal] We conclude that:

- $s = \mathsf{function}\ (\overline{x})\{s'\}^{m'}$
- $h_f = h \uplus (l', @proto) \mapsto l_{op} \uplus \mathsf{fun}(l'', L, \overline{x}, s, l', m')$
- $o = l''$

We have to show that: h_f and (h_f, L) are ψ'-compatible, for $\psi' = \Phi_m(s, \psi)$ and $m = \mathsf{firstFID}(L)$. Because $s = \mathsf{function}\ (\overline{x})\{s'\}^{m'}$, we conclude that $\psi' = \Phi_{m'}(s', \hat{\psi})$ and:

$$\hat{\psi} = \psi \left[\begin{array}{l} (m, x_i) \mapsto m' \mid_{x_i \in \overline{x} \cup \mathsf{defs}(s')} \\ (m, y_j) \mapsto \psi(m, y_j) \mid_{y_j \in \mathsf{dom}(\psi(m))} \end{array} \right]$$

Since Φ is monotonous (Lemma 1), we know that ψ' extends ψ. Therefore, it follows that h and (h, L) are ψ'-compatible. Observe that:

- (h_f, L) is ψ'-compatible since L was not modified and (h, L) is ψ'-compatible.
- h_f is ψ'-compatible since h is ψ'-compatible and the scope chain that is added to the range of the heap (L) is also ψ'-compatible.

[Remaining Cases] The proofs of the remaining cases follow directly through the application of the induction hypothesis. □

Chapter 2

JSIL - Syntax and Semantics

The JSIL Language

Strings: $m \in \mathcal{S}tr$ Numbers: $n \in \mathcal{N}um$ Booleans: $b \in \mathcal{B}ool$

Locations: $l \in \mathcal{L}$ Variables: $\mathtt{x} \in \mathcal{X}_{\mathtt{JSIL}}$

Literals: $\lambda \in \mathcal{L}it \ ::= \ n \mid b \mid m \mid \mathsf{undefined} \mid \mathsf{null}$

References: $r \in ((\mathcal{L}it \cup \mathcal{L}) \setminus \{\mathsf{null}\}) \times \{\mathtt{v}, \mathtt{o}\} \times \mathcal{S}tr$

Types : $\mathtt{t} \in \mathtt{Type} ::= \mathtt{Num} \mid \mathtt{Bool} \mid \mathtt{Str} \mid \mathtt{Ref_o} \mid \mathtt{Ref_v} \mid \mathtt{Obj} \mid \mathtt{List}$

Values : $\mathtt{v} \in \mathcal{V}_{\mathtt{JSIL}} ::= \lambda \mid l \mid \mathsf{empty} \mid \mathsf{error} \mid \mathtt{t} \mid r \mid \overline{\mathtt{v}}$

Exprs. : $\mathtt{e} \in \mathcal{E}_{\mathtt{JSIL}} ::= \mathtt{v} \mid \mathtt{x} \mid \ominus\ \mathtt{e} \mid \mathtt{e} \oplus \mathtt{e} \mid \mathsf{typeof}\,(\mathtt{e}) \mid \mathsf{ref_v}(\mathtt{e}, \mathtt{e})$
$\mathsf{ref_o}(\mathtt{e}, \mathtt{e}) \mid \mathsf{base}\,(\mathtt{e}) \mid \mathsf{field}\,(\mathtt{e}) \mid \overline{\mathtt{e}} \mid \mathsf{elt}\,(\mathtt{e}, \mathtt{e})$

Basic Commands:

$\mathtt{bc} \in \mathtt{BCmd} ::= \mathsf{skip} \mid \mathtt{x} := \mathtt{e} \mid \mathtt{x} := \mathsf{new}\,() \mid \mathtt{x} := [\mathtt{e}, \mathtt{e}] \mid$
$[\mathtt{e}_1, \mathtt{e}_2] := \mathtt{e}_3 \mid \mathsf{delete}\,(\mathtt{e}, \mathtt{e}) \mid \mathtt{x} := \mathsf{hasField}\,(\mathtt{e}, \mathtt{e})$

Commands:

$\mathtt{c} \in \mathtt{Cmd} ::= \mathtt{bc} \mid \mathsf{goto}\ \ i \mid \mathsf{goto}\ \ [\mathtt{e}]\ i,\ j \mid$

$\mathtt{x} := \mathtt{e}(\overline{\mathtt{e}})\ \mathsf{with}\ j \mid \mathtt{x} := \phi(\overline{\mathtt{x}})$

Procedures : $\mathtt{proc} \in \mathtt{Proc} \ ::= \mathsf{proc}\ \ m(\overline{\mathtt{x}})\{\overline{\mathtt{c}}\}$

JSIL heaps : $\mathtt{h} \in \mathcal{H}_{\mathtt{JSIL}} \ : \ \mathcal{L} \times \mathcal{S}tr \rightharpoonup \mathcal{V}_{\mathtt{JSIL}}$

JSIL stores : $\rho \in \mathcal{S}to \ : \ \mathcal{X}_{\mathtt{JSIL}} \rightharpoonup \mathcal{V}_{\mathtt{JSIL}}$

Notation: $\overline{\mathtt{x}}$, $\overline{\mathtt{v}}$, $\overline{\mathtt{e}}$ denote lists of variables, values, expressions respectively. Predecessor Relation: $\mathsf{pred}(i, j) = k$ iff i is the k^{th} predecessor of j.

Semantics of JSIL Expressions: $[\![\mathtt{e}]\!]_\rho = \mathtt{v}$

Literal
$[\![\mathtt{v}]\!]_\rho \triangleq \mathtt{v}$

Variable
$[\![\mathtt{x}]\!]_\rho \triangleq \rho(\mathtt{x})$

Binary Operator
$$\frac{v_1 = [\![\mathtt{e}_1]\!]_\rho \quad v_2 = [\![\mathtt{e}_2]\!]_\rho}{[\![\mathtt{e}_1 \ \oplus \ \mathtt{e}_2]\!]_\rho \triangleq v_1 \ \oplus \ v_2}$$

Reference Constructor
$$\frac{l = [\![\mathtt{e}_1]\!]_\rho \quad x = [\![\mathtt{e}_2]\!]_\rho \quad \mathtt{t} = [\![\mathtt{e}_3]\!]_\rho}{[\![\mathsf{ref}\,(\mathtt{e}_1, \mathtt{e}_2, \mathtt{e}_3)]\!]_\rho \triangleq l._{\mathsf{Ann(t)}} x}$$

Reference Projection - Base
$$\frac{l._{\mathtt{a}} x = [\![\mathtt{e}]\!]_\rho}{[\![\mathsf{base}\,(\mathtt{e})]\!]_\rho \triangleq l}$$

Reference Projection - Field
$$\frac{l._{\mathtt{a}} x = [\![\mathtt{e}]\!]_\rho}{[\![\mathsf{field}\,(\mathtt{e})]\!]_\rho \triangleq x}$$

TypeOf
$$\frac{v = [\![\mathtt{e}]\!]_\rho \qquad \mathtt{t} = \mathsf{TypeOf}(v)}{[\![\mathsf{typeof}\,(\mathtt{e})]\!]_\rho \triangleq \mathtt{t}}$$

Semantics of Basic Commands: $[\![\mathtt{bc}]\!]_{h,\rho} = (h', \rho', \mathtt{v})$

Skip
$[\![\mathsf{skip}\,]\!]_{\mathtt{h},\rho} \triangleq (\mathtt{h}, \rho, \mathsf{empty})$

Assignment
$$\frac{[\![\mathtt{e}]\!]_\rho = \mathtt{v} \quad \rho' = \rho[\mathtt{x} \mapsto \mathtt{v}]}{[\![\mathtt{x} := \mathtt{e}]\!]_{\mathtt{h},\rho} \triangleq (\mathtt{h}, \rho', \mathtt{v})}$$

Object Creation
$$\frac{\mathtt{h}' = \mathtt{h} \uplus (l, @\mathit{proto}) \mapsto \mathsf{null} \quad \rho' = \rho[\mathtt{x} \mapsto l]}{[\![\mathtt{x} := \mathsf{new}\,()]\!]_{\mathtt{h},\rho} \triangleq (\mathtt{h}', \rho', \mathsf{true})}$$

Field Projection
$$\frac{\mathtt{v} = \mathtt{h}([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \quad \rho' = \rho[\mathtt{x} \mapsto \mathtt{v}]}{[\![\mathtt{x} := [\mathtt{e}_1, \mathtt{e}_2]]\!]_{\mathtt{h},\rho} \triangleq (\mathtt{h}, \rho', \mathtt{v})}$$

Field Assignment
$$\frac{\mathtt{v} = [\![\mathtt{e}_3]\!]_\rho \quad \mathtt{h}' = \mathtt{h}[([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \mapsto \mathtt{v}]}{[\![[\mathtt{e}_1, \mathtt{e}_2] := \mathtt{e}_3]\!]_{\mathtt{h},\rho} \triangleq (\mathtt{h}', \rho, \mathtt{v})}$$

Member Check - True
$$\frac{([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \in \mathsf{dom}(\mathtt{h}) \quad \rho' = \rho[\mathtt{x} \mapsto \mathsf{true}]}{[\![\mathtt{x} := \mathsf{hasField}\,(\mathtt{e}_1, \mathtt{e}_2)]\!]_{\mathtt{h},\rho} \triangleq (\mathtt{h}, \rho', \mathsf{true})}$$

Member Check - False
$$\frac{([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \notin \mathsf{dom}(\mathtt{h}) \quad \rho' = \rho[\mathtt{x} \mapsto \mathsf{false}]}{[\![\mathtt{x} := \mathsf{hasField}\,(\mathtt{e}_1, \mathtt{e}_2)]\!]_{\mathtt{h},\rho} \triangleq (\mathtt{h}, \rho', \mathsf{false})}$$

Deletion
$$\frac{\mathtt{h} = \mathtt{h}' \uplus ([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \mapsto \mathtt{v}}{[\![\mathsf{delete}\,(\mathtt{e}_1, \mathtt{e}_2)]\!]_{\mathtt{h},\rho} \triangleq (\mathtt{h}', \rho, \mathsf{true})}$$

Semantics of control flow commands: $\mathtt{p} \vdash \langle h, \rho, i, j\rangle \Downarrow_m \langle h', \rho', v\rangle$

Basic Command
$$\frac{\mathtt{p}_m(i) = \mathtt{c} \in \mathtt{BCmd} \quad [\![\mathtt{c}]\!]_{\mathtt{h},\rho} = (\mathtt{h}', \rho') \quad \mathtt{p} \vdash \langle \mathtt{h}', \rho', i, i+1\rangle \Downarrow_m \langle \mathtt{h}'', \rho'', \mathtt{v}\rangle}{\mathtt{p} \vdash \langle \mathtt{h}, \rho, \_, i\rangle \Downarrow_m \langle \mathtt{h}'', \rho'', \mathtt{v}\rangle}$$

Goto
$$\frac{\mathtt{p}_m(i) = \mathsf{goto}\ j \quad \mathtt{p} \vdash \langle \mathtt{h}, \rho, i, j\rangle \Downarrow_m \langle \mathtt{h}', \rho', \mathtt{v}\rangle}{\mathtt{p} \vdash \langle \mathtt{h}, \rho, \_, i\rangle \Downarrow_m \langle \mathtt{h}', \rho', \mathtt{v}\rangle}$$

COND. GOTO - TRUE

$$\frac{\mathtt{p}_m(i) = \mathsf{goto}\ [\mathtt{e}]\ j,\ k \quad [\![\mathtt{e}]\!]_\rho = \mathsf{true} \qquad \mathtt{p} \vdash \langle \mathtt{h}, \rho, i, j\rangle \Downarrow_m \langle \mathtt{h}', \rho', \mathtt{v}\rangle}{\mathtt{p} \vdash \langle \mathtt{h}, \rho, \_, i\rangle \Downarrow_m \langle \mathtt{h}', \rho', \mathtt{v}\rangle}$$

COND. GOTO - FALSE

$$\frac{\mathtt{p}_m(i) = \mathsf{goto}\ [\mathtt{e}]\ j,\ k \quad [\![\mathtt{e}]\!]_\rho = \mathsf{false} \qquad \mathtt{p} \vdash \langle \mathtt{h}, \rho, i, k\rangle \Downarrow_m \langle \mathtt{h}', \rho', \mathtt{v}\rangle}{\mathtt{p} \vdash \langle \mathtt{h}, \rho, \_, i\rangle \Downarrow_m \langle \mathtt{h}', \rho', \mathtt{v}\rangle}$$

NORMAL RETURN

$$\vdash \langle \mathtt{h}, \rho, \_, i_{ret}\rangle \Downarrow_m \langle \mathtt{h}, \rho, \rho(\mathtt{x_{ret}})\rangle$$

ERROR RETURN

$$\vdash \langle \mathtt{h}, \rho, \_, i_{err}\rangle \Downarrow_m \langle \mathtt{h}, \rho, \mathsf{error}\rangle$$

PROCEDURE CALL

$$\frac{\begin{array}{l}\mathtt{p}_m(i) = \mathtt{x} := \mathtt{e}(\mathtt{e}_1, ..., \mathtt{e}_{n_1})\ \mathsf{with}\ j \qquad [\![\mathtt{e}]\!]_\rho = m' \qquad \mathtt{p}(m') = \mathsf{proc}\ \ m'(\mathtt{y}_1, ..., \mathtt{y}_{n_2})\{\mathtt{cl}\} \\ \forall_{1\le n\le n_1} \mathtt{v}_n = [\![\mathtt{e}_n]\!]_\rho \quad \forall_{n_1<n\le n_2} \mathtt{v}_n = \mathsf{undefined} \qquad \mathtt{p} \vdash \langle \mathtt{h}, [\mathtt{y}_1 \mapsto \mathtt{v}_1, \cdots, \mathtt{v}_{n_2} \mapsto v_{n_2}], 0, 0\rangle \Downarrow_{m'} \langle \mathtt{h}', \rho', \mathtt{v}\rangle \\ \mathtt{v} \neq \mathsf{error} \Rightarrow k = i + 1 \quad \mathtt{v} = \mathsf{error} \Rightarrow k = j \qquad \mathtt{p} \vdash \langle \mathtt{h}', \rho[\mathtt{x} \mapsto \mathtt{v}], i, k\rangle \Downarrow_m \langle \mathtt{h}'', \rho'', \mathtt{v}'\rangle\end{array}}{\mathtt{p} \vdash \langle \mathtt{h}, \rho, \_, i\rangle \Downarrow_m \langle \mathtt{h}'', \rho'', \mathtt{v}'\rangle}$$

PHI-ASSIGNMENT

$$\frac{\mathtt{p}_m(j) = \mathtt{x} := \phi(\mathtt{x}_1, ..., \mathtt{x}_n) \quad \mathsf{pred}(i, j) = k \quad \mathtt{p} \vdash \langle \mathtt{h}, \rho[\mathtt{x} \mapsto \rho(\mathtt{x}_k)], j, j+1\rangle \Downarrow_m \langle \mathtt{h}', \rho', \mathtt{v}\rangle}{\mathtt{p} \vdash \langle \mathtt{h}, \rho, i, j\rangle \Downarrow_m \langle \mathtt{h}', \rho', \mathtt{v}\rangle}$$

Chapter 3

Correctly Compiling ES5 Strict to JSIL

3.1 Compiler Formalisation

The complete formalisation of the compiler for the fragment of Core ES5 Strict considered is presented in Figures 3.1, 3.2, and 3.3 below.

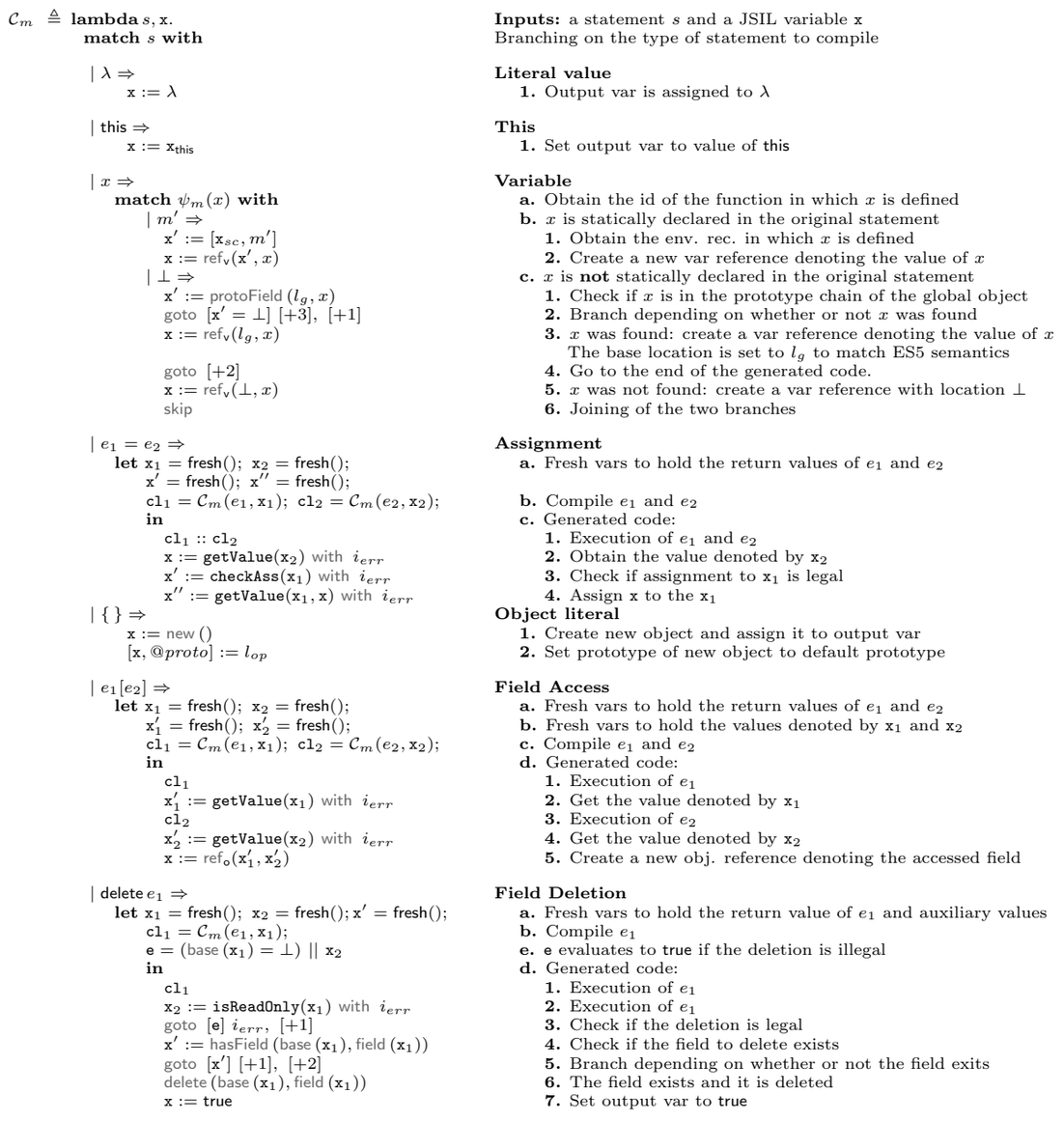

Figure 3.1: Compilation of Expressions, Part 1

$\mathcal{C}_m \triangleq$ **lambda** $s, \mathtt{x}$.
match s **with**

| function $(\overline{x})\{s\}^{m'} \Rightarrow$
let $\mathtt{x}' = \mathsf{fresh}(); \mathtt{x}'' = \mathsf{fresh}();$

in
$\mathtt{x}' :=$ `copyScopeChain(`$\mathtt{x}_{sc}$`)` with i_{err}
$\mathtt{x}'' :=$ new ()
$[\mathtt{x}'', @proto] := l_{op}$
$\mathtt{x} :=$ new ()
$[\mathtt{x}, @scope] := \mathtt{x}'$

$[\mathtt{x}, @body] := m'$

$[\mathtt{x}, prototype] := \mathtt{x}''$
$[\mathtt{x}, @proto] := l_{fp}$

Inputs: a statement s and a JSIL variable $\mathtt{x}$
Branching on the type of statement to compile

Function Literal
a. Fresh vars to hold intermediate values scope chain and assign it to $\mathtt{x}'$
b. Generated code:
1. Copy the current scope chain and assign it to $\mathtt{x}'$
2. Create prototype object
3. Set $@proto$ of $\mathtt{x}''$ to default prototype
4. Create function object
5. Set field $@scope$ of function object to copy of the current scope chain
6. Set field $@body$ of function object to the the identifier of the function literal
7. Set field prototype of function object to $\mathtt{x}''$
8. Set field $@proto$ of function object to the default function prototype

| $e_1 \oplus e_2 \Rightarrow$
let $\mathtt{x}_1 = \mathsf{fresh}();\ \mathtt{x}_2 = \mathsf{fresh}();$
$\mathtt{x}'_1 = \mathsf{fresh}();\ \mathtt{x}'_2 = \mathsf{fresh}();$
$\mathtt{cl}_1 = \mathcal{C}_m(e_1, \mathtt{x}_1);\ \mathtt{cl}_2 = \mathcal{C}_m(e_2, \mathtt{x}_2);$
in
$\mathtt{cl}_1$
$\mathtt{x}'_1 :=$ `getValue(`$\mathtt{x}_1$`)` with i_{err}
$\mathtt{cl}_2$
$\mathtt{x}'_2 :=$ `getValue(`$\mathtt{x}_2$`)` with i_{err}
$\mathtt{x} := \mathtt{x}'_1 \oplus \mathtt{x}'_2$

Binary Operator
a. Fresh vars to hold the return values of e_1 and e_2
b. Fresh vars to hold the values denoted by $\mathtt{x}_1$ and $\mathtt{x}_2$
c. Compile e_1 and e_2
d. Generated code:
1. Execution of e_1
2. Obtain the value denoted by $\mathtt{x}_1$
3. Execution of e_2
4. Obtain the value denoted by $\mathtt{x}_2$
5. Binary operation

| $e(e_1, \cdots, e_n) \Rightarrow$
let $\mathtt{x}_e = \mathsf{fresh}();\ \mathtt{x}_i = \mathsf{fresh}(), i \in \{1...n\};$
$\mathtt{x}'_e = \mathsf{fresh}(); \mathtt{x}'_i = \mathsf{fresh}(), i \in \{1...n\};$
$\mathtt{x}_l = \mathsf{fresh}(); \mathtt{x}_{sc} = \mathsf{fresh}(); \mathtt{x}_{bd} = \mathsf{fresh}();$
$\mathtt{x}' = \mathsf{fresh}(); \mathtt{x}'' = \mathsf{fresh}();$
$\mathtt{cl}_e = \mathcal{C}_m(e, \mathtt{x}_e);$
$\mathtt{cl}_i = \mathcal{C}_m(e_i, \mathtt{x}_i), i \in \{1...n\};$
$\mathtt{cl}'_i = \mathtt{x}'_i :=$ `getValue(`$\mathtt{x}_i$`)` with $i_{err}, i \in \{1...n\};$
in
$\mathtt{cl}_e :: \{\mathtt{cl}_i :: \mathtt{cl}'_i\}, i \in \{1...n\}$
goto $[\mathsf{typeof}(\mathtt{x}_e)$!= `Ref`$_\mathsf{v}]$ $[+1],\ [+4]$
$\mathtt{x}_l :=$ base $(\mathtt{x}_e)$

$\mathtt{x}'_e :=$ `getValue(`$\mathtt{x}_e$`)` with i_{err}
goto $[+2]$
$\mathtt{x}_l :=$ undefined
$\mathtt{x}_{sc} := [\mathtt{x}_e, @scope]$
$\mathtt{x}_{bd} := [\mathtt{x}_e, @body]$
$\mathtt{x}' := \mathtt{x}_{bd}(\mathtt{x}_{sc}, \mathtt{x}_l, \mathtt{x}'_1, ..., \mathtt{x}'_n)$ with i_{err}
goto $[\mathtt{x} = \bot]$ $[+1],\ [+2]$
$\mathtt{x}'' :=$ undefined
$\mathtt{x} := \phi(\mathtt{x}', \mathtt{x}'')$

Function/Method Call
a. Fresh vars to hold the return values of e and e_i, $i \in \{1...n\}$
b. Fresh auxiliary vars
c. Fresh auxiliary vars

d. Compile e
e. Compile e_i, $i \in \{1...n\}$
f. Generate the code to dereference $\mathtt{x}_i$, $i \in \{1...n\}$
g. Generated code:
1. Execution of e and e_i, dereferencing of $\mathtt{x}_i$, $i \in \{1...n\}$
2. Branch on $\mathtt{x}_e$ not being a variable reference
3. $\mathtt{x}_e$ was not a variable reference: set $\mathtt{x}_l$ to the location of $\mathtt{x}_e$
4. Dereferencing of $\mathtt{x}_e$
5. Go to the next appropriate command
6. $\mathtt{x}_e$ was a variable reference: set $\mathtt{x}_l$ to undefined
7. Set $\mathtt{x}_{sc}$ to the $@scope$ field of $\mathtt{x}_e$
8. Set $\mathtt{x}_{bd}$ to the $@body$ field of $\mathtt{x}_e$
9. Call the procedure corresponding to the function/method
10. Branch depending on whether or not $\mathtt{x}$ was found
11. $\mathtt{x}$ was not found: set $\mathtt{x}$ to undefined
12. Joining of the two branches

| new $e(e_1, \cdots, e_n) \Rightarrow$
let $\mathtt{x}_e = \mathsf{fresh}();\ \mathtt{x}_i = \mathsf{fresh}(), i \in \{1...n\};$
$\mathtt{x}'_i = \mathsf{fresh}(), i \in \{1...n\};\ \mathtt{x}_l = \mathsf{fresh}(); \mathtt{x}_{pr} = \mathsf{fresh}();$
$\mathtt{x}_{scp} = \mathsf{fresh}(); \mathtt{x}_{bd} = \mathsf{fresh}();$
$\mathtt{x}' = \mathsf{fresh}(); \mathtt{x}'' = \mathsf{fresh}();$
$\mathtt{cl}_e = \mathcal{C}_m(e, \mathtt{x}_e);$
$\mathtt{cl}_i = \mathcal{C}_m(e_i, \mathtt{x}_i), i \in \{1...n\};$
$\mathtt{cl}'_i = \mathtt{x}'_i :=$ `getValue(`$\mathtt{x}_i$`)` with $i_{err}, i \in \{1...n\};$
in
$\mathtt{cl}_e :: \{\mathtt{cl}_i :: \mathtt{cl}'_i\}, i \in \{1...n\}$
$\mathtt{x}_l :=$ new ()
$\mathtt{x}_{pr} := [\mathtt{x}_e, prototype]$
goto $[\mathsf{typeof}(\mathtt{x}_{pr})$!= `Obj`$]$ $[+1],\ [+2]$
$\mathtt{x}_{pr} := l_g$

$[\mathtt{x}_l, @proto] := \mathtt{x}_{pr}$

$\mathtt{x}_{scp} := [\mathtt{x}_e, @scope]$
$\mathtt{x}_{bd} := [\mathtt{x}_e, @body]$
$\mathtt{x}' := \mathtt{x}_{bd}(\mathtt{x}_{scp}, \mathtt{x}_l, \mathtt{x}_1, ..., \mathtt{x}_n)$ with i_{err}
goto $[\mathtt{x} = \bot]$ $[+1],\ [+2]$
skip
$\mathtt{x} := \phi(\mathtt{x}', \mathtt{x}_l)$

Constructor Call
a. Fresh vars to hold the return values of e and e_i, $i \in \{1...n\}$
b. Fresh auxiliary vars

c. Compile e
d. Compile e_i, $i \in \{1...n\}$
e. Generate the code to dereference $\mathtt{x}_i$, $i \in \{1...n\}$
f. Generated code:
1. Execution of e and e_i, $i \in \{1...n\}$
3. Create the this object
4. Set $\mathtt{x}_{pr}$ to the prototype of $\mathtt{x}_e$
5. Branch on the prototype of $\mathtt{x}_e$ not being an object
6. The prototype of $\mathtt{x}_e$ is not an object: set $\mathtt{x}_{pr}$ to the global object
7. The prototype of $\mathtt{x}_e$ is an object: set the $@proto$ field of $\mathtt{x}_l$ to the prototype of $\mathtt{x}_e$
8. Set $\mathtt{x}^f_{sc}$ to the $@scope$ field of $\mathtt{x}_e$
9. Set $\mathtt{x}_{bd}$ to the $@body$ field of $\mathtt{x}_e$
10. Call the procedure corresponding to the constructor
11. Branch depending on whether or not $\mathtt{x}$ was found
12. Set x to the this object
13. Joining of the two branches

Figure 3.2: Compilation of Expressions, Part 2

$\mathcal{C}_m \triangleq$ **lambda** $s, \mathtt{x}$.
match s **with**

| $\mathsf{var}\, x \Rightarrow$
$\mathtt{x} = \mathsf{empty}$

Inputs: a statement s and a JSIL variable $\mathtt{x}$
Branching on the type of statement to compile

Variable Declaration
a. Variable declarations evaluate to empty

| $s_1; s_2 \Rightarrow$
let $\mathtt{x}_1 = \mathsf{fresh}();\ \mathtt{x}_2 = \mathsf{fresh}();$
$\mathtt{x}'_1 = \mathsf{fresh}();\ \mathtt{x}'_2 = \mathsf{fresh}();$
$\mathtt{cl}_1 = \mathcal{C}_m(s_1, \mathtt{x}_1);\ \mathtt{cl}_2 = \mathcal{C}_m(s_2, \mathtt{x}_2);$
in
$\mathtt{cl}_1$
$\mathtt{x}'_1 := \mathtt{getValue}(\mathtt{x}_1)$ with i_{err}
$\mathtt{cl}_2$
$\mathtt{x}'_2 := \mathtt{getValue}(\mathtt{x}_2)$ with i_{err}
goto $[\mathtt{x}_2 = \mathsf{empty}]\ [+2],\ [+1]$
skip
$\mathtt{x} := \phi(\mathtt{x}_1, \mathtt{x}_2)$

Sequence Statement
a. Fresh vars to hold the return values of s_1 and s_2
b. Fresh auxiliary variables
c. Compile s_1 and s_2
d. Generated code:
1. Execution of s_1
2. Get the value denoted by $\mathtt{x}_1$
3. Execution of s_2
4. Get the value denoted by $\mathtt{x}_2$
5. Branch on $\mathtt{x}_2$ being equal to empty
6.
7. Joining of the two branches

| $\mathsf{if}(e)\,\{s_1\}\,\mathsf{else}\,\{s_2\} \Rightarrow$
let $\mathtt{x}_e = \mathsf{fresh}(); \mathtt{x}'_e = \mathsf{fresh}();$
$\mathtt{x}_1 = \mathsf{fresh}(); \mathtt{x}_2 = \mathsf{fresh}(); \mathtt{x}'_1 = \mathsf{fresh}(); \mathtt{x}'_2 = \mathsf{fresh}();$
$\mathtt{cl}_e = \mathcal{C}_m(e, \mathtt{x}_e);$
$\mathtt{cl}_1 = \mathcal{C}_m(s_1, \mathtt{x}_1);\ \mathtt{cl}_2 = \mathcal{C}_m(s_2, \mathtt{x}_2);$
$\mathtt{e} = (\mathtt{x}'_e = \mathsf{false})\ ||\ (\mathtt{x}'_e = \mathsf{undefined})\ ||$
$(\mathtt{x}'_e = \mathsf{null})\ ||\ (\mathtt{x}'_e = \text{“”})$
in
$\mathtt{cl}_e$
$\mathtt{x}'_e := \mathtt{getValue}(\mathtt{x}_e)$ with i_{err}
goto $[!\mathtt{e}]\ [+\#\mathtt{cl}'_1 + 4],\ [+1]$
$\mathtt{cl}_1$
$\mathtt{x}'_1 := \mathtt{getValue}(\mathtt{x}_1)$ with i_{err}
goto $[+\#\mathtt{cl}_2 + 3]$
$\mathtt{cl}_2$
$\mathtt{x}'_2 := \mathtt{getValue}(\mathtt{x}_2)$ with i_{err}
$\mathtt{x} := \phi(\mathtt{x}'_1, \mathtt{x}'_2)$

If Statement
a. Fresh vars to hold the return values of e, s_1, and s_2
b. Compile e and generate the code to dereference e
c. Compile s_1 and s_2
d. e evaluates to a falsy value
e. Generated code:
1. Execution of e
2. Dereferencing of $\mathtt{x}_e$
3. Branch on $\mathtt{e}$
4. $\mathtt{e}$ is not falsy: execution of s_1
5. Dereferencing of $\mathtt{x}_1$
6. Go to the end of the generated code
7. $\mathtt{e}$ is falsy: execution of s_2
8. Dereferencing of $\mathtt{x}_2$
9. Joining of the two branches

| $\mathsf{while}(e)\,\{s\} \Rightarrow$
let $\mathtt{x}_e = \mathsf{fresh}(); \mathtt{x}_s = \mathsf{fresh}();$
$\mathtt{x}'_e = \mathsf{fresh}(); \mathtt{x}'_s = \mathsf{fresh}();$
$\mathtt{x}' = \mathsf{fresh}(); \mathtt{x}'' = \mathsf{fresh}(); \mathtt{x}''' = \mathsf{fresh}();$
$\mathtt{cl}_e = \mathcal{C}_m(e, \mathtt{x}_e);$
$\mathtt{cl}_s = \mathcal{C}_m(s, \mathtt{x}_s);$
$\mathtt{e} = (\mathtt{x}'_e = \mathsf{false})\ ||\ (\mathtt{x}'_e = \mathsf{undefined})\ ||$
$(\mathtt{x}'_e = \mathsf{null})\ ||\ (\mathtt{x}'_e = \text{“”})$
in
$\mathtt{x}' := \mathsf{empty}$
$\mathtt{x}'' := \phi(\mathtt{x}', \mathtt{x}''')$
$\mathtt{cl}_e$
$\mathtt{x}'_e := \mathtt{getValue}(\mathtt{x}_e)$ with i_{err}
goto $[\mathtt{e}]\ [+\#\mathtt{cl}_s + \#\mathtt{cl}'_s + 5],\ [+1]$
$\mathtt{cl}_s$
$\mathtt{x}'_s := \mathtt{getValue}(\mathtt{x}_s)$ with i_{err}
goto $[\mathtt{x}'_s\ != \mathsf{empty}]\ [+1],\ [+2]$
skip
$\mathtt{x}''' := \phi(\mathtt{x}'', \mathtt{x}'_s)$
goto $[-\#\mathtt{cl}_e - \#\mathtt{cl}_s - 7]$
$\mathtt{x} := \mathtt{x}''$

While Statement
a. Fresh vars to hold the return values of e and s
b. Fresh auxiliary variables
c. Compile e
d. Compile s
e. e evaluates to a falsy value
f. Generated code:
1. Set output var to empty in case no iterations occur
2. Set output var to empty in case no iterations occur
3. Execution of e
4. Dereferencing of $\mathtt{x}_e$
5. Branch on $\mathtt{e}$
6. $\mathtt{e}$ is false: execution of s
7. Dereferencing of $\mathtt{x}_s$
8. Branch on $\mathtt{x}_s$ not being equal to empty
9. $\mathtt{x}_s$ is not equal to empty: set output var to $\mathtt{x}_s$
10.
11. $\mathtt{x}_s$ is equal to empty: proceed to the next iteration
12. $\mathtt{e}$ is true: exit the loop

| $\mathsf{return}\, e \Rightarrow$
let $\mathtt{x}_e = \mathsf{fresh}(); \mathtt{x}'_e = \mathsf{fresh}();$
$\mathtt{cl}_e = \mathcal{C}_m(e, \mathtt{x}_e);$
in
$\mathtt{cl}_e$
$\mathtt{x}_e := \mathtt{getValue}(\mathtt{x}'_e)$ with i_{err}
$\mathtt{x}_{\mathtt{ret}} := \mathtt{x}_e$
goto i_{ret}

Return Statement
a. Fresh var to hold the return value of e
b. Compile e
c. Generated code:
1. Execution of e
2. Dereferencing of $\mathtt{x}_e$
3. Set return value to $\mathtt{x}_e$
4. Return

Figure 3.3: Compilation of Statements. We use #cl to denote the number of commands in cl.

3.2 Correctness Proof

3.2.1 β-Indistinguishability

β-indistinguishability

On outcomes : $\sim_\beta$: $\mathcal{O} \times \mathcal{O}$

Constants and Literals
$$\frac{o \in \mathcal{L}it \cup \{\mathsf{empty}, \mathsf{error}, \bot\}}{o \sim_\beta o}$$

Locations
$$l \sim_\beta \beta(l)$$

References
$$l.\mathtt{a}x \sim_\beta \beta(l).\mathtt{a}x$$

Return values
$$\frac{v \sim_\beta v'}{\mathsf{ret}\, v \sim_\beta v'}$$

Lists of values
$$\frac{\overline{v} = \{v_i\}_{i=1}^n \quad \overline{v'} = \{v'_i\}_{i=1}^n \quad (v_i \sim_\beta v'_i)_{i=1}^n}{\overline{v} \sim_\beta \overline{v'}}$$

On heaps : $\sim_\beta$: $\mathcal{H}_{\mathtt{JS}} \times \mathcal{H}_{\mathtt{JSIL}}$

Function Objects - Scope
$$(l, @scope) \mapsto \{\{(m_i, l_i)|_{i=1}^n\}, m\} \uplus \sim_\beta (\beta(l), @scope) \mapsto l_{sc} \uplus \mathtt{SC}_\beta((m, \_) :: (m_i, l_i)|_{i=1}^n, l_{sc})$$

Function Objects - Body
$$(l, @body) \mapsto \lambda\overline{x}.s^m \sim_\beta (\beta(l), @body) \mapsto m$$

Other Properties
$$\frac{l \sim_\beta l' \quad v \sim_\beta v' \quad x \notin \{@scope, @body\}}{(l, x) \mapsto v \sim_\beta (l', x) \mapsto v'}$$

Heap Composition
$$\frac{h_1 \sim_\beta \mathtt{h}_1 \quad h_2 \sim_\beta \mathtt{h}_2}{h_1 \uplus h_2 \sim_\beta \mathtt{h}_1 \uplus \mathtt{h}_2}$$

On heaps and programs: $\simeq_\beta$: $\mathcal{H}_{\mathtt{JS}} \times (\mathtt{P} \times \mathcal{H}_{\mathtt{JSIL}})$

Heaps and Programs
$$\frac{h \sim_\beta \mathtt{h}' \quad \mathtt{h} = \mathtt{h}' \uplus \mathtt{SC}_0(l_{scp}) \quad \lambda\overline{x}.s^m \in \mathsf{funlits}(h) \Rightarrow \hat{\mathcal{C}}(\mathsf{function}\,(\overline{x})\{s\}^m) \in \mathtt{p}}{h \simeq_\beta \mathtt{p}, \mathtt{h}}$$

Full β-indistinguishability : $\simeq_\beta$: $(\mathcal{S}ch_{\mathtt{JS}} \times \mathcal{L} \times \mathcal{H}_{\mathtt{JS}}) \times (\mathtt{P} \times \mathcal{H}_{\mathtt{JSIL}} \times \mathcal{S}to)$

β-indistinguishability
$$\frac{h \simeq_\beta \mathtt{p}, \mathtt{h}' \quad \mathtt{h} = \mathtt{h}' \uplus \mathtt{SC}_\beta(L, l_{sc}) \quad \rho \geq [\mathtt{x}_{sc} \mapsto l_{sc}, \mathtt{x}_{\mathsf{this}} \mapsto \beta(l_{this})]}{L, l_{this}, h \simeq_\beta \mathtt{p}, \mathtt{h}, \rho}$$

3.2.2 Auxiliary Definitions, Lemmas and Assumptions

Definition 3 (Command List Execution)**.** The execution of a command list $\mathtt{cl}$ in the context of the program $\mathtt{p}$, written $\mathtt{p} \vdash \langle h, \rho, \mathtt{cl}\rangle \Downarrow \langle h', \rho', v\rangle$, is defined as follows:

$$\mathtt{p} \cup \{\mathsf{proc}\ \ m(\overline{\mathtt{x}})\{\mathtt{cl}\}\} \vdash \langle h, \rho, \mathsf{init}(\mathtt{cl}), \mathsf{init}(\mathtt{cl})\rangle \Downarrow_m \langle h', \rho', v\rangle,$$

where $\mathsf{init}(\mathtt{cl})$ is the first label of $\mathtt{cl}$ and $\mathtt{p}$ does not contain a procedure with identifier m.

Lemma 3 (Prototype Chain Indistinguishability)**.** *Given two heaps h and h', a location l, and a variable x such that $h \sim_\beta h'$; it holds that:*

$$\pi(h, l, x) \sim_\beta \pi(h', \beta(l), x)$$

Proof. The result follows by induction on $\pi(h, l, x) = v$. □

Assumption 3.1 (Correctness of `getValue`)**.** Given a scope chain L, a location l_{this}, two heaps h and h', a a JSIL program $\mathtt{p}$, a store ρ, two arbitrary outcomes o_0 and o'_0, and a JSIL variable $\mathtt{x} \notin \{\mathtt{x}_{sc}, \mathtt{x}_{this}\}$, such that: $L, l_{this}, h \simeq_\beta \mathtt{p}, h', \rho$, $\rho(\mathtt{x}) = o_0$, and $o_0 \sim_\beta o'_0$ for a given function β; it holds that:

$$\mathtt{getValue}(h, o_0) = o_1 \ \Leftrightarrow\ \mathtt{p} \vdash \langle h', \rho, \mathtt{x}' := \mathtt{getValue}(\mathtt{x})\ \mathsf{with}\ \ i_{err}\rangle \Downarrow \langle h', \rho_f, o'_1\rangle$$

in which case: $o_1 \sim_\beta \rho_f(\mathtt{x}) = o'_1$ and $L, l_{this}, h \simeq_\beta \mathtt{p}, h', \rho_f$.

Assumption 3.2 (Correctness of `isReadOnly`)**.** Given a JSIL program $\mathtt{p}$, a heap h, a store ρ, two JSIL variables $\mathtt{x}$ and $\mathtt{x}'$ such that: $\rho(\mathtt{x}) = r$, it holds that:

$$\mathtt{p} \vdash \langle h, \rho, \mathtt{x}' := \mathtt{isReadOnly}(\mathtt{x})\ \mathsf{with}\ \ i_{err}\rangle \Downarrow \langle h, \rho[\mathtt{x}' \mapsto v], v\rangle$$

where: $v = \mathsf{true} \Leftrightarrow \mathsf{RO}(r)\ \ \wedge\ \ v = \mathsf{false} \Leftrightarrow \neg\mathsf{RO}(r)$.

Assumption 3.3 (Correctness of `isReadOnly`)**.** Given a JSIL program $\mathtt{p}$, a heap h, a store ρ, two JSIL variables $\mathtt{x}$ and $\mathtt{x}'$ such that: $\rho(\mathtt{x}) = r$, it holds that:

$$\mathtt{p} \vdash \langle h, \rho, \mathtt{x}' := \mathtt{isReadOnly}(\mathtt{x})\ \mathsf{with}\ \ i_{err}\rangle \Downarrow \langle h, \rho[\mathtt{x}' \mapsto v], v\rangle$$

where: $v = \mathsf{true} \Leftrightarrow \mathsf{RO}(r)\ \ \wedge\ \ v = \mathsf{false} \Leftrightarrow \neg\mathsf{RO}(r)$.

Assumption 3.4 (Correctness of `copyScopeChain`)**.** Given a JSIL program $\mathtt{p}$, a heap h, a scope chain L, a store ρ, a JSIL variable $\mathtt{x} \notin \{\mathtt{x}_{sc}, \mathtt{x}_{this}\}$ such that $\mathtt{SC}_\beta(L, l_{sc}) \subseteq h$, $\rho(\mathtt{x}_{sc}) = l_{sc}$; it follows that: $\mathtt{p} \vdash \langle h, \rho, \mathtt{x}' := \mathtt{copyScopeChain}(\mathtt{x})\ \mathsf{with}\ \ i_{err}\rangle \Downarrow \langle h', \rho[\mathtt{x} \mapsto l'_{sc}], \mathsf{empty}\rangle$ and $h' = h \uplus \mathtt{SC}_\beta(L, l'_{sc})$.

3.2.3 Main Proof

Lemma 4 (Correctness of $\mathcal{C}$)**.** *Given a JavaScript statement s, a JSIL program $\mathtt{p}$, a command list $\mathtt{cl}$, a variable $\mathtt{x}$, and a procedure identifier m such that $\mathcal{C}_m(s, \mathtt{x}) = \mathtt{cl}$; a scope chain L, a location l_{this}, two heaps h and h', a program $\mathtt{p}$, and a store ρ, such that: $L, l_{this}, h \simeq_\beta \mathtt{p}, h', \rho$; it holds that:*

$$L, l_{this} \vdash \langle h, s\rangle \Downarrow \langle h_f, o\rangle \;\Leftrightarrow\; \mathtt{p} \vdash \langle h', \rho, \mathtt{cl}\rangle \Downarrow \langle h'_f, \rho_f, o'\rangle$$

in which case there exists a function β' that extends β such that:

- $L, l_{this}, h_f \simeq_{\beta'} \mathtt{p}, h'_f, \rho$,
- $o \sim_{\beta'} o'$, *and if* $o \neq \mathsf{error}$ *and* $o \neq \mathsf{ret}\, v$ *for some value v, then* $\rho_f(\mathtt{x}) = o'$.

Proof. We will prove the left-to-right implication, whereas the other direction follows suit:

$$L, l_{this} \vdash \langle h, s\rangle \Downarrow \langle h_f, o\rangle \;\Rightarrow\; \mathtt{p} \vdash \langle h', \rho, \mathtt{cl}\rangle \Downarrow \langle h'_f, \rho_f, o'\rangle$$

We proceed by induction on the derivation of $L, l_{this} \vdash \langle h, s\rangle \Downarrow \langle h_f, o\rangle$. For convenience, we name the hypotheses:

- **Hyp.** 1: $L, l_{this} \vdash \langle h, s\rangle \Downarrow \langle h_f, o\rangle$;
- **Hyp.** 2: $\mathcal{C}_m(s, \mathtt{x}) = \mathtt{cl}$;
- **Hyp.** 3: $L, l_{this}, h \simeq_\beta \mathtt{p}, h', \rho$.

We need to prove that, for some β' extending β:

- **Goal** 1: $\mathtt{p} \vdash \langle h', \rho, \mathtt{cl}\rangle \Downarrow \langle h'_f, \rho_f, o'\rangle$;
- **Goal** 2: $L, l_{this}, h_f \simeq_{\beta'} \mathtt{p}, h'_f, \rho_f$;
- **Goal** 3: $o \sim_{\beta'} o'$, and if $o \neq \mathsf{error}$, then $\rho_f(\mathtt{x}) = o'$.

Since, when it comes to expressions, the final outcome can never be $\mathsf{ret}\, v$ (expressions cannot contain `return` statements), we have chosen to omit that part of **Goal** 3 for now.

[Binary Operator] It follows that $s = e_1 \oplus e_2$. We begin with this case in order to illustrate both the induction hypothesis and the treatment of errors in the derivations and compilations, which we discuss here in detail. Our goal is to, using the following hypotheses:

- **Hyp.** H1: $L, l_{this} \vdash \langle h, e_1 \oplus e_2\rangle \Downarrow \langle h_f, o\rangle$;
- **Hyp.** H2: $\mathcal{C}_m(e_1 \oplus e_2, \mathtt{x}) = \mathtt{cl}$;
- **Hyp.** H3: $L, l_{this}, h \simeq_\beta \mathtt{p}, h', \rho$.

show that, for some β' extending β:

- **Goal** 1: $\mathtt{p} \vdash \langle h', \rho, \mathtt{cl}\rangle \Downarrow \langle h'_f, \rho_f, o'\rangle$;
- **Goal** 2: $L, l_{this}, h_f \simeq_{\beta'} \mathtt{p}, h'_f, \rho_f$
- **Goal** 3: $o \sim_{\beta'} o'$, and if $o \neq \mathsf{error}$, then $\rho_f(\mathtt{x}) = o'$.

The compiled code corresponding to the binary operator rule is:

$$\mathtt{cl} = \begin{cases} \mathtt{cl}_1 \\ \mathtt{x}'_1 := \mathtt{getValue}(\mathtt{x}_1) \text{ with } i_{err} \\ \mathtt{cl}_2 \\ \mathtt{x}'_2 := \mathtt{getValue}(\mathtt{x}_2) \text{ with } i_{err} \\ \mathtt{x} := \mathtt{x}'_1 \oplus \mathtt{x}'_2 \end{cases}$$

where $\mathtt{cl}_1 = \mathcal{C}_m(e_1, \mathtt{x}_1)$, $\mathtt{cl}_2 = \mathcal{C}_m(e_2, \mathtt{x}_2)$, for four fresh variables $\mathtt{x}_1$, $\mathtt{x}'_1$, $\mathtt{x}_2$ and $\mathtt{x}'_2$. The derivation starts with the following rule:

Binary Operator

$$\frac{L, l_{this} \vdash \langle h, e_1\rangle \Downarrow^\gamma \langle h_1, o_1\rangle \qquad L, l_{this} \vdash \langle h_1, o_1 \oplus_1 e_2\rangle \Downarrow \langle h_f, o\rangle}{L, l_{this} \vdash \langle h, e_1 \oplus e_2\rangle \Downarrow \langle h_f, o\rangle} \quad \text{i.e.} \quad \frac{L, l_{this} \vdash \langle h, e_1\rangle \Downarrow \langle h_1, \hat{o}_1\rangle,\ \mathtt{getValue}(h_1, \hat{o}_1) = o_1 \qquad L, l_{this} \vdash \langle h_1, o_1 \oplus_1 e_2\rangle \Downarrow \langle h_f, o\rangle}{L, l_{this} \vdash \langle h, e_1 \oplus e_2\rangle \Downarrow \langle h_f, o\rangle}$$

and we would like to apply the induction hypothesis (**IH**) to the first premise, which is $L, l_{this} \vdash \langle h, e_1\rangle \Downarrow \langle h_1, \hat{o}_1\rangle$. The compiled code corresponding to this premise is $\mathtt{cl}_1$. We can apply the **IH**, starting from $\langle h', \rho, \mathtt{cl}_1\rangle$, since all the necessary hypotheses are met:

- **Hyp.** H1$_1$: $L, l_{this} \vdash \langle h, e_1\rangle \Downarrow \langle h_1, \hat{o}_1\rangle$ — The premise;
- **Hyp.** H2$_1$: $\mathcal{C}_m(e_1, \mathtt{x}_1) = \mathtt{cl}_1$ — By definition of the compiler;
- **Hyp.** H3$_1$: $L, l_{this}, h \simeq_\beta \mathtt{p}, h', \rho$ — By **Hyp.** H3;

What we obtain is some β_1 extending β, such that:

- **Goal** G1$_1$: $\mathtt{p} \vdash \langle h', \rho, \mathtt{cl}_1\rangle \Downarrow \langle h'_1, \rho_1, \hat{o}'_1\rangle$;
- **Goal** G2$_1$: $L, l_{this}, h_1 \simeq_{\beta'} \mathtt{p}, h'_1, \rho_1$;

- **Goal** G3$_1$: $\hat{o}_1 \sim_{\beta_1} \hat{o}'_1$, and if $\hat{o}_1 \neq \mathsf{error}$, then $\rho_1(\mathtt{x}) = o'_1$.

First, we will consider the case in which $\hat{o}_1 = \mathsf{error}$. In that case, given the definition of β-indistinguishability, we have that $\hat{o}'_1 = \mathsf{error}$ as well. This is only possible if we have encountered a jump to the i_{err} label during the execution of $\mathtt{cl}_1$, in which case, the execution of the compiled program has terminated, and we have that $h''_f = h''_1$, $\rho_f = \rho_1$, and $o' = \mathsf{error}$. On the other hand, the derivation needs to be concluded, by first dereferencing $\hat{o}_1 = \mathsf{error}$ and obtaining $o_1 = \mathsf{error}$ and then proceeding as follows, since the error needs to be propagated through the pretty-big-step rules:

Binary Operator

$$\frac{L, l_{this} \vdash \langle h, e_1\rangle \Downarrow^{\gamma} \langle h_1, \mathsf{error}\rangle \quad L, l_{this} \vdash \langle h_1, \mathsf{error} \oplus_1 e_2\rangle \Downarrow \langle h_1, \mathsf{error}\rangle}{L, l_{this} \vdash \langle h, e_1 \oplus e_2\rangle \Downarrow \langle h_1, \mathsf{error}\rangle}$$

since **Error Propagation** $L, l_{this} \vdash \langle h_1, \mathsf{error} \oplus_1 e_2\rangle \Downarrow \langle h_1, \mathsf{error}\rangle$

As this is the only possible continuation of the derivation, we have that $h_f = h_1$ and $o = \mathsf{error}$, which means that we already have all of the three required goals (G1$_1$ – G3$_1$).

Remark on the further treatment of errors

All of the cases in which errors occur unravel precisely in the manner shown here, via the error propagation rule on the derivation side and the compiler reaching a jump to the i_{err} label, aborting the execution, and signalling an error on the compiler side. For this reason, we will not present such cases in the remainder of the proof and we will assume, without loss of generality, that the derivations that we are considering are error-free. To further justify this decision, we refer the reader to [**?**], where it is demonstrated that if we assume the pretty-big-step approach to error propagation (cf. [**?**]), then the cases in which errors occur can be fully discharged automatically.

Next, we examine the case in which $\hat{o}_1 \neq \mathsf{error}$. The next step on the derivation side is the dereferencing of $\hat{o}_1$, where we obtain $\mathtt{getValue}(h_1, \hat{o}_1) = v_1$, and the next rule to be applied is

Binary Operator - 1

$$\frac{L, l_{this} \vdash \langle h_1, e_2\rangle \Downarrow^{\gamma} \langle h_2, o_2\rangle \quad L, l_{this} \vdash \langle h_2, v_1 \oplus_2 o_2\rangle \Downarrow \langle h_f, o\rangle}{L, l_{this} \vdash \langle h_1, v_1 \oplus_1 e_2\rangle \Downarrow \langle h_f, o\rangle}$$

i.e.

$$\frac{L, l_{this} \vdash \langle h_1, e_2\rangle \Downarrow \langle h_2, \hat{o}_2\rangle,\ \mathtt{getValue}(h_2, \hat{o}_2) = o_2 \quad L, l_{this} \vdash \langle h_2, v_1 \oplus_1 o_2\rangle \Downarrow \langle h_f, o\rangle}{L, l_{this} \vdash \langle h_1, v_1 \oplus e_2\rangle \Downarrow \langle h_f, o\rangle}$$

whereas on the compiler side, we have the execution of $\mathtt{x}'_1 := \mathtt{getValue}(\mathtt{x}_1)$ with i_{err}. By Deferencing Indistinguishability, we have that $\mathtt{p} \vdash \langle h'_1, \rho_1, \mathtt{x}'_1 := \mathtt{getValue}(\mathtt{x}_1) \text{ with } i_{err}\rangle \Downarrow \langle h'_1, \rho'_1, v'_1\rangle$, $\rho'_1(\mathtt{x}_1) = \mathtt{getValue}(h'_1, \rho_1(\mathtt{x}_1)) = v'_1$, and by Dereferencing Indistinguishability, we have that

$$v_1 \sim_{\beta_1} v'_1 = \rho'_1(\mathtt{x}_1). \tag{3.1}$$

Now, we would like to apply the **IH** to $L, l_{this} \vdash \langle h_1, e_2\rangle \Downarrow \langle h_2, \hat{o}_2\rangle$. The compiled code corresponding to this premise is $\mathtt{cl}_2$. We can apply the **IH**, starting from $\langle h'_1, \rho_1, \mathtt{cl}_2\rangle$, since all the necessary hypotheses are met:

- **Hyp.** H1$_2$: $L, l_{this} \vdash \langle h_1, e_2\rangle \Downarrow \langle h_2, \hat{o}_2\rangle$ — The premise;
- **Hyp.** H2$_2$: $\mathcal{C}_m(e_2, \mathtt{x}_2) = \mathtt{cl}_2$ — By definition of the compiler;
- **Hyp.** H3$_1$: $L, l_{this}, h_1 \simeq_{\beta} \mathtt{p}, h'_1, \rho'_1$

  By **Goal** G2$_1$, we have that $L, l_{this}, h_1 \simeq_{\beta'} \mathtt{p}, h'_1, \rho_1$, so the only claim that we need to demonstrate is $\rho'_1 \geq [\mathtt{x}_{sc} \mapsto l_{sc}, \mathtt{x}_{\mathsf{this}} \mapsto \beta(l_{this})]$ given $\rho_1 \geq [\mathtt{x}_{sc} \mapsto l_{sc}, \mathtt{x}_{\mathsf{this}} \mapsto \beta(l_{this})]$, which is immediate since $\rho'_1 = \rho_1[\mathtt{x}_1 \mapsto v'_1]$.

What we obtain is some β_2 extending β_1, such that:

- **Goal** G1$_2$: $\mathtt{p} \vdash \langle h'_1, \rho'_1, \mathtt{cl}_2\rangle \Downarrow \langle h'_2, \rho_2, \hat{o}'_2\rangle$;
- **Goal** G2$_2$: $L, l_{this}, h_2 \simeq_{\beta} \mathtt{p}, h'_2, \rho_2$
- **Goal** G3$_2$: $\hat{o}_2 \sim_{\beta_2} \hat{o}'_2$, and if $\hat{o}_2 \neq \mathsf{error}$, then $\rho_2(\mathtt{x}) = o'_2$.

As noted earlier, we assume that $\hat{o}_2 \neq \mathsf{error}$. The next step on the derivation side is the dereferencing of $\hat{o}_2$, where we obtain $\gamma(h_2, \hat{o}_2) = v_2$, and the final rule to be applied is

Binary Operator - 2

$$\frac{v = v_1 \overline{\oplus} v_2}{L, l \vdash \langle h, v_1 \oplus_2 v_2\rangle \Downarrow \langle h, v\rangle}$$

whereas on the compiler side, we have the execution of $\mathtt{cl}'_2 = \mathtt{getValue}(\mathtt{x}_2)$. By Deferencing Indistinguishability, we have that $\mathtt{p} \vdash \langle h'_2, \rho_2, \mathtt{x}'_1 := \mathtt{getValue}(\mathtt{x}_1) \text{ with } i_{err}\rangle \Downarrow \langle h'_2, \rho'_2, v'_2\rangle$, $\rho'_2(\mathtt{x}_2) = \mathtt{getValue}(h'_2, \rho_2(\mathtt{x}_2)) = v'_2$, and by Dereferencing Indistinguishability, we have that

$$v_2 \sim_{\beta_2} v'_2 = \rho'_2(\mathtt{x}_2). \tag{3.2}$$

Finally, on the derivation side, we have that $h_f = h_2$ and $o = v_1 \oplus v_2$. On the compiler side, after executing the last command ($\mathtt{x} := \mathtt{x}_1 \oplus \mathtt{x}_2$), we have that $h'_f = h'_2$, $\rho_f = \rho'_2[\mathtt{x} \mapsto \rho'_2(\mathtt{x}_1) \oplus \rho'_2(\mathtt{x}_2)]$, and $o' = \rho'_2(\mathtt{x}_1) \oplus \rho'_2(\mathtt{x}_2)$. Further noting that $\beta' = \beta_2$, let us make sure that we have the required goals:

- **Goal** 1: $\mathtt{p} \vdash \langle h', \rho, \mathtt{cl}\rangle \Downarrow \langle h'_f, \rho_f, o'\rangle$

  This we have by construction:

$$\mathtt{p} \vdash \langle h', \rho, \mathtt{cl}\rangle \Downarrow \langle h'_2, \rho'_2[\mathtt{x} \mapsto \rho'_2(\mathtt{x}_1) \oplus \rho'_2(\mathtt{x}_2)], \rho'_2(\mathtt{x}_1) \oplus \rho'_2(\mathtt{x}_2)\rangle$$

- **Goal** 2: $L, l_{this}, h_f \simeq_{\beta'} \mathtt{p}, h'_f, \rho_f$, i.e. $L, l_{this}, h_2 \simeq_{\beta'} \mathtt{p}, h'_2, \rho'_2[\mathtt{x} \mapsto \rho'_2(\mathtt{x}_1) \oplus \rho'_2(\mathtt{x}_2)]$.

  By **Goal** G2$_1$, we have that $L, l_{this}, h_2 \simeq_{\beta'} \mathtt{p}, h'_2, \rho_2$, so the only claim that we need to demonstrate is $\rho'_2[\mathtt{x} \mapsto \rho'_2(\mathtt{x}_1) \oplus \rho'_2(\mathtt{x}_2)] \geq [\mathtt{x}_{sc} \mapsto l_{sc}, \mathtt{x}_{\mathsf{this}} \mapsto \beta(l_{this})]$, given that $\rho_2 \geq [\mathtt{x}_{sc} \mapsto l_{sc}, \mathtt{x}_{\mathsf{this}} \mapsto \beta(l_{this})]$. This is again immediate, given that $\rho'_2 = \rho_2[\mathtt{x}_2 \mapsto v'_2]$, and neither $\mathtt{x}_2$ (since it is fresh) nor $\mathtt{x}$ is equal to either $\mathtt{x}_{sc}$ or $\mathtt{x}_{\mathsf{this}}$.

- **Goal** 3: $o \sim_{\beta'} o'$ and if $o' \neq \mathsf{error}$, then $\rho_f(\mathtt{x}) = o'$.

  This amounts to $v_1 \oplus v_2 \sim_{\beta_2} \rho_f(\mathtt{x}_1) \oplus \rho_f(\mathtt{x}_2)$, and (as $\rho_f(\mathtt{x}_1) \oplus \rho_f(\mathtt{x}_2) \neq \mathsf{error}$) $\rho_f(\mathtt{x}) = \rho_f(\mathtt{x}_1) \oplus \rho_f(\mathtt{x}_2)$. As the latter is trivially correct, let us focus on the former, and for this, it is sufficient to show that $v_1 \sim_{\beta_2} \rho_f(\mathtt{x}_1)$ and $v_2 \sim_{\beta_2} \rho_f(\mathtt{x}_2)$. The latter holds by (3.2) and the fact that subsequent changes to ρ'_2 do not affect $\mathtt{x}_2$ (since $\mathtt{x} \neq \mathtt{x}_2$, as $\mathtt{x}_2$ is fresh). As for the former, we have from (3.1) that $v_1 \sim_{\beta_1} v'_1 = \rho'_1(\mathtt{x}_1)$. Since β_2 extends β_1, we certainly have that $v_1 \sim_{\beta_2} v'_1 = \rho'_1(\mathtt{x}_1)$. Finally, since $\mathtt{x}_1$ is fresh, we have that all the subsequent changes to the store do not affect its value at $\mathtt{x}_1$, and that, therefore, $\rho'_1(\mathtt{x}_1) = \rho_f(\mathtt{x}_1)$, and so the desired goal holds.

Remark on the further treatment of fresh variables

In this case, we have twice used the fact that if a (fresh) variable was being assigned a value once and only once during the execution, then this value in the store necessarily remains unchanged after the assignment occurs. This pattern will often be reappearing in the exact same manner in the upcoming cases, where we will denote it by VSA—Variable with a Single Assignment.

Remark on the further treatment of the induction hypothesis

In the subsequent cases, we will elaborate in detail on the use of the induction hypothesis only if there are significant differences with respect to the binary operator case shown above. Otherwise, we will simply denote its use and state the statements obtained from it.

For the remaining cases, we adopt a more schematic and succinct approach, in the style of [**?**]. We present a proof as an enumerated series of steps, where each step consists of a statement (on the left) and a sketch of how this statement was derived (on the right). For example, the step

7. $h_f \sim_\beta h'_f$, where $h''_f = h'_f \uplus \mathtt{SC}_\beta(L, h, l_{sc})$ By (1) + (5) + **Hyp.** 2

indicates that the statement $h_f \sim_\beta h'_f$, where $h''_f = h'_f \uplus \mathtt{SC}_\beta(L, h, l_{sc})$ holds, that it is the seventh statement of our derivation, and that it was derived using the first and fifth statement of the derivation, as well as the second hypothesis. The sketches will be more verbose and will be placed below the statement when there are certain details that require a more elaborate explanation. Before we proceed, let us revisit the hypotheses and goals:

- **Hyp.** 1: $L, l_{this} \vdash \langle h, s\rangle \Downarrow \langle h_f, o\rangle$,
- **Hyp.** 2: $\mathcal{C}_m(s, \mathtt{x}) = \mathtt{cl}$,
- **Hyp.** 3: $L, l_{this}, h \simeq_\beta \mathtt{p}, h', \rho$,
- **Goal** 1: $\mathtt{p} \vdash \langle h', \rho, \mathtt{cl}\rangle \Downarrow \langle h'_f, \rho_f, o'\rangle$,
- **Goal** 2: $L, l_{this}, h_f \simeq_{\beta'} \mathtt{p}, h'_f, \rho_f$,
- **Goal** 3: $o \sim_{\beta'} o'$, and if $o \neq \mathsf{error}$, then $\rho_f(\mathtt{x}) = o'$.

  for some β' extending β.

[Function Call] It follows that $s = e(e_1, \cdots, e_n)$ (**Hyp.** 4). The rules on the derivation side that are applied in this case are the following:

Function Call

$$\frac{L, l_t \vdash \langle h, e\rangle \Downarrow \langle h_0, o_0\rangle \quad L, l_t \vdash \langle h_0, o_0(\overline{e})_1\rangle \Downarrow \langle h_f, o\rangle}{L, l_t \vdash \langle h, e(\overline{e})\rangle \Downarrow \langle h_f, o\rangle}$$

Function Call - 1

$$\frac{L, l_t \vdash \langle h_0, \mathsf{iterate}\{\overline{e}\}\rangle \Downarrow \langle h_n, \overline{v}\rangle \quad L, l_t \vdash \langle h_n, l.\mathsf{a}x(\overline{v})_2\rangle \Downarrow \langle h_f, o\rangle}{L, l_t \vdash \langle h_0, l.\mathsf{a}x(\overline{e})_1\rangle \Downarrow \langle h_f, o\rangle}$$

Function Call - 2

$$\frac{\begin{array}{l} l_{\mathsf{this}} = \mathsf{SelectThis}(l.\mathsf{a}x) \quad l_f = \gamma(h_n, l.\mathsf{a}x) \\ h_n(l_f, @body) = \lambda\overline{x}.s_b^{m'} \quad h_n(l_f, @scope) = L' \\ l_s :: L', l_{\mathsf{this}} \vdash \langle h_n \uplus \mathsf{act}_{m'}(l_s, \overline{x}, \overline{v}, s_b), s_b\rangle \Downarrow \langle h_f, \hat{o}\rangle \end{array}}{L, l_t \vdash \langle h, l.\mathsf{a}x(\overline{v})_2\rangle \Downarrow \langle h_f, \mathsf{FunRet}(\hat{o})\rangle}$$

1. After the application of these rules, there exist $n+1$ heaps $h_0, h_1, \ldots, h_n$, such that:
   (a) $L, l_t \vdash \langle h, e\rangle \Downarrow \langle h_0, l.\mathsf{a}x\rangle$,
   (b) $L, l_t \vdash \langle h_i, e_i\rangle \Downarrow \langle h_{i+1}, o_i\rangle$ and $\gamma(h_i, o_i) = v_i$, for $1 \leq i \leq n$,
   (c) $\gamma(h_n, l.\mathsf{a}x) = l_f$,
   (d) $h_n(l_f, @body) = \lambda\overline{x}.s_b^{m'}$,
   (e) $h_n(l_f, @scope) = L'$,
   (f) If $\mathsf{a} = \mathsf{v}$ then $l_{\mathsf{this}} = \mathsf{undefined}$, otherwise $l_{\mathsf{this}} = l$,
   (g) $(l_s, m') :: L', l_{\mathsf{this}} \vdash \langle h_n \uplus \mathsf{act}_{m'}(l_s, \overline{x}, \overline{v}, s_b), s_b\rangle \Downarrow \langle h_f, \hat{o}\rangle$,
   (h) $o = \mathsf{FunRet}(\hat{o})$.

2. $\mathtt{cl} = \begin{cases} \mathtt{cl}_0 \\ \mathtt{cl}_i :: \mathtt{cl}'_i \ i \in \{1...n\} \\ \mathsf{goto}\ [\mathsf{typeof}\,(\mathtt{x}_0)\ \mathtt{!=}\ \mathtt{Ref_v}]\ [+1],\ [+4] \\ \mathtt{x}_t := \mathsf{base}\,(\mathtt{x}_0) \\ \mathsf{goto}\ [+2] \\ \mathtt{x}_t := \mathsf{undefined} \\ \mathtt{cl}'_0 \\ \mathtt{x}_{scp} := [\mathtt{x}_0, @scope] \\ \mathtt{x}_{bd} := [\mathtt{x}_0, @body] \\ \mathtt{x} := \mathtt{x}_{bd}(\mathtt{x}_{scp}, \mathtt{x}_l, \mathtt{x}_1, ..., \mathtt{x}_n) \\ \mathsf{goto}\ [\mathtt{x} = \bot]\ [+1],\ [+2] \\ \mathtt{x} := \mathsf{undefined} \\ \mathsf{skip} \end{cases}$

where $\mathtt{cl}_0 = \mathcal{C}_m(e, \mathtt{x}_0)$, $\mathtt{cl}_i = \mathcal{C}_m(e_i, \mathtt{x}_i)$, and $\mathtt{cl}'_i = \mathtt{x}'_i := \mathtt{getValue}(\mathtt{x}_i)$ with i_{err}.

3. By applying the induction hypothesis $n+1$ times, using (1a) and (1b), and making use of Dereferencing Indistinguishability, we obtain:

   (a) $\mathtt{p} \vdash \langle h', \rho, \mathtt{cl}_0 :: \mathtt{cl}_1 :: \mathtt{cl}'_1 :: \cdots :: \mathtt{cl}_n :: \mathtt{cl}_n \rangle \Downarrow \langle h'_n, \rho_n, o'_n \rangle$,
   (b) $L, l_t, h_n \simeq_{\beta_n} \mathtt{p}, h'_n, \rho_n$, and
   (c) $l.\mathtt{a}x \sim_{\beta_n} \rho_n(\mathtt{x}_0)$, and $v_i \sim_{\beta_n} v'_i = \rho_n(\mathtt{x}_i)$ for $1 \leq i \leq n$,

   for some β_n extending β. The induction hypothesis can be applied first due to (1a) and **Hyp.** 3, and each further application is justified by (1b) and the goals obtained from the previous application.

4. $\rho_n(\mathtt{x}_0) = \beta_n(l).\mathtt{a}x$ By (3c) and the definition of β-indistinguishability

5. $\mathtt{p} \vdash \langle h'_n, \rho, \mathtt{cl}_t \rangle \Downarrow \langle h'_n, \rho_t, \_ \rangle$ for:

$$\begin{array}{rcl} \mathtt{cl}_t & = & \mathsf{goto}\ [\mathsf{typeof}\,(\mathtt{x}_0)\ \mathtt{!=}\ \mathtt{Ref_v}]\ [+1],\ [+4] :: \mathtt{x}_t := \mathsf{base}\,(\mathtt{x}_0) \\ & & :: \mathsf{goto}\ [+2] :: \mathtt{x}_t := \mathsf{undefined} \\ \rho_t & = & \rho_n[\mathtt{x}_t \mapsto l'_{\mathsf{this}}] \end{array}$$

   where $l'_{\mathsf{this}} = \mathsf{undefined}$ if $\mathtt{a} = \mathtt{v}$ and $l'_{\mathsf{this}} = \beta_n(l)$ otherwise. From (3b), and given the definition of ρ_t, we also have that

   (a) $L, l_t, h_n \simeq_{\beta_n} \mathtt{p}, h'_n, \rho_t$.

6. $\mathtt{p} \vdash \langle h'_n, \rho_t, \mathtt{cl}'_0 :: \mathtt{x}_{scp} := [\mathtt{x}_0, @scope] :: \mathtt{x}_{bd} := [\mathtt{x}_0, @body] \rangle \Downarrow \langle h'_n, \rho'_t, \_ \rangle$, where $\rho'_t = \rho_t[\mathtt{x}_0 \mapsto \beta_n(l_f), \mathtt{x}_{scp} \mapsto l'_{sc}, \mathtt{x}_{bd} \mapsto m']$ and $\mathtt{SC}_{\beta_n}((\_, m') :: L', l'_{sc}) \in h'_n$.

   This follows from (1c-e), (5a), and the definition of β-indistinguishability for function objects. From (5a), and given the definition of ρ'_t, we also have that

   (a) $L, l_t, h_n \simeq_{\beta_n} \mathtt{p}, h'_n, \rho'_t$.

7. $\mathtt{p}$ contains the following procedure:

$$\begin{array}{l} \mathsf{proc}\, m'(\mathtt{x}_{scp}, \mathtt{x}_{this}, \mathtt{x}_1, ..., \mathtt{x}_n)\{ \\ \quad \mathtt{x} := \mathsf{new}\,() \\ \quad [\mathtt{x}, x_i] := \mathtt{x}_i, \text{ for } i \in \{1,...,n\} \\ \quad [\mathtt{x}, y_i] := \mathsf{undefined}, \text{ for } i \in \{1,...,k\} \\ \quad [\mathtt{x}_{scp}, m'] := \mathtt{x} \\ \quad \mathtt{cl}' \\ \quad \mathtt{x}_{\mathtt{ret}} := \bot \\ \quad \mathsf{goto}\ i_{ret} \\ \} \end{array}$$

   where $\mathtt{cl}' = \mathcal{C}(s_b, \_)$, $(y_1, ..., y_k) = \mathsf{defs}(s_b)$,
   and $x_1, ..., x_n$ are the arguments of $\mathsf{function}\,(\overline{x})\{s_b\}^{m'}$ By (1) + (6a)

8. $\mathtt{p} \vdash \langle h'_n, \rho_{bd}, \mathtt{cl}_{bd} \rangle \Downarrow \langle h'_{bd}, \rho'_{bd}, \_ \rangle$ where:

$$\begin{array}{rcl} \rho_{bd} & = & [\mathtt{x}_{sc} \mapsto l'_{sc}, \mathtt{x}_{this} \mapsto l'_{\mathsf{this}}, \mathtt{x}_1 \mapsto v'_1, ..., \mathtt{x}_n \mapsto v'_n] \\ \mathtt{cl}_{bd} & = & \mathtt{x} := \mathsf{new}\,() :: [\mathtt{x}, x_1] := \mathtt{x}_1 :: ... :: [\mathtt{x}, x_n] := \mathtt{x}_n \\ & & :: [\mathtt{x}, y_1] := \mathsf{undefined} :: ... :: [\mathtt{x}, y_k] := \mathsf{undefined} \\ \rho'_{bd} & = & \rho_{bd} \uplus [\mathtt{x} \mapsto l'_s] \\ h'_{bd} & = & h'_n \uplus l'_s \mapsto \{x_1 \colon v'_1, \cdots, x_n \colon v'_n, y_1 \colon \mathsf{undefined}, \cdots, y_k \colon \mathsf{undefined}\} \end{array}$$

   By definition of $\hat{\mathcal{C}}$ and $\mathcal{C}$

9. $h_{bd} = h_n \uplus \mathsf{act}_{m'}(l_s, \overline{x}, \overline{v}, s_b) \simeq_{\beta_{er}} h'_{bd}$ for $\beta_{er} = \beta_n[l_s \mapsto l'_s]$

   Since $h_n \simeq_{\beta_{er}} \mathtt{p}, h'_n$ by (6a) and the definition of β_{er}, this amounts to

$$\mathsf{act}_{m'}(l_s, \overline{x}, \overline{v}, s_b) \sim_{\beta_{er}} l'_s \mapsto \{x_1 \colon v'_1, \cdots, x_n \colon v'_n, y_1 \colon \mathsf{undefined}, \cdots, y_k \colon \mathsf{undefined}\},$$

   which we obtain directly from the definition of β-indistinguishability and (3c).

10. $(l_s, m') :: L', l_{\mathsf{this}}, h_{bd} \simeq_{\beta_{er}} \mathtt{p}, h'_{bd}, \rho'_{bd}$

    - We have that $\mathtt{SC}_{\beta}((l_s, m') :: L, l'_{sc}) \subseteq h'_n$ from (6-8).
    - We also have that $\rho \geq [\mathtt{x}'_{sc} \mapsto l_{sc}, \mathtt{x}_{\mathsf{this}} \mapsto \beta(l_{this})]$ from (8), (5) and (1f).
    - Finally, $h_{bd} \simeq_{\beta_{er}} \mathtt{p}, h'_{bd}$ from (9).

11. (a) $\mathtt{p} \vdash \langle h'_{bd}, \rho'_{bd}, \mathtt{cl}' \rangle \Downarrow \langle h'_f, \rho''_{bd}, o'_{bd} \rangle$,
    (b) $(l_s, m') :: L', l_{\mathsf{this}}, h_f \simeq_{\beta'_{er}} \mathtt{p}, h'_f, \rho''_{bd}$, and
    (c) $\hat{o} \sim_{\beta'_{er}} o'_{bd}$, and $\rho''_{bd}(\mathtt{x}) = o'_{bd}$,

    for β'_{er} that extends β_{er}

    By (1g) + (7) + (10) + **IH**

12. (G1) $\mathtt{p} \vdash \langle h', \rho, \mathtt{cl} \rangle \Downarrow \langle h'_f, \rho_f, o' \rangle$, where $\rho_f = \rho'_t[\mathtt{x} \mapsto o']$, and $o = \mathsf{FunRet}(\hat{o})$

13. (G2) $L, l_t, h_f \simeq_{\beta'_{er}} \mathtt{p}, h'_f, \rho_f$ By (11b) and the definition of β-indistinguishability

14. (G3) $o \sim_{\beta'_{er}} o'$ and $\rho_f(\mathtt{x}) = o'$

    The latter follows from (12a). As for the former, we need to show that $\mathsf{FunRet}(\hat{o}) \sim_{\beta'_{er}} o'$. This follows from (11c), the definition of FunRet, and the construction of o', i.e. the final two lines of the code in (7) and the final three lines of $\mathtt{cl}$, which make sure that in the case in which $\mathtt{cl}'$ does not return, the return value of the function is $\mathsf{undefined}$, as required by JavaScript semantics.

□

Chapter 4

JS-2-JSIL- Logic

4.1 JS and JSIL Assertions

JavaScript: Logical Values, Logical Expressions, Assertions

$$
\begin{array}{l}
V \in \mathcal{V}^L_{\mathtt{JS}} ::= \omega \mid \varnothing \\
E \in \mathcal{E}^L_{\mathtt{JS}} ::= \mathsf{this} \mid V \mid \mathrm{X} \mid \ominus\ E \mid E \oplus E \mid \mathsf{typeof}\ (E) \mid E :: E
\end{array}
$$

$$
\begin{array}{lll}
P, Q ::= E_1 = E_2 \mid E_1 \leq E_2 & & \text{Boolean} \\
\quad \mid P \vee Q \mid P \wedge Q \mid \neg P \mid \mathrm{true} \mid \mathrm{false} \mid \exists \mathtt{x}.P \mid \forall \mathtt{x}.P & & \text{Classical} \\
\quad \mid \mathtt{emp} \mid (E, E) \mapsto E \mid P * Q & & \text{JS heaps} \\
\quad \mid \sigma(x, E) \mid \pi(E_1, E_2, E_3) & & \text{Scope and proto chains}
\end{array}
$$

Semantics of JS Logical Expressions and Assertions

Logical Expressions (Semantics):

$$
\begin{array}{lcl}
[\![\mathsf{this}]\!]_{\epsilon, l_{this}} & \triangleq & l_{this} \\
[\![V]\!]_{\epsilon, l_{this}} & \triangleq & V \\
[\![\mathrm{X}]\!]_{\epsilon, l_{this}} & \triangleq & \epsilon(\mathrm{X}) \\
[\![\ominus\ E]\!]_{\epsilon, l_{this}} & \triangleq & \ominus([\![E]\!]_{\epsilon, l_{this}}) \\
[\![E_1 \oplus E_2]\!]_{\epsilon, l_{this}} & \triangleq & [\![E_1]\!]_{\epsilon, l_{this}} \oplus [\![E_2]\!]_{\epsilon, l_{this}} \\
[\![\mathsf{typeof}\ \ E]\!]_{\epsilon, l_{this}} & \triangleq & \mathsf{TypeOf}([\![E]\!]_{\epsilon, l_{this}}) \\
[\![E_1 :: E_2]\!]_{\epsilon, l_{this}} & \triangleq & [\![E_1]\!]_{\epsilon, l_{this}} :: [\![E_2]\!]_{\epsilon, l_{this}}
\end{array}
$$

Assertions (Satisfiability Relation):

$$
\begin{array}{ll}
H, L, l_{this}, \epsilon \models E_1 = E_2 & \Leftrightarrow [\![E_1]\!]_{\epsilon, l_{this}} = [\![E_2]\!]_{\epsilon, l_{this}} \\
H, L, \epsilon, l_{this} \models E_1 \leq E_2 & \Leftrightarrow [\![E_1]\!]_{\epsilon, l_{this}} \leq [\![E_2]\!]_{\epsilon, l_{this}} \\
H, L, \epsilon, l_{this} \models P * Q & \Leftrightarrow \exists H_1, H_2.\ H = H_1 \uplus H_2\ \wedge\ (H_1, \epsilon, l_{this}, \epsilon \models P)\ \wedge\ (H_2, \epsilon, l_{this} \models Q) \\
H, L, \epsilon, l_{this} \models (E_1, E_2) \mapsto E_3 & \Leftrightarrow H = ([\![E_1]\!]_{\epsilon, l_{this}}, [\![E_2]\!]_{\epsilon, l_{this}}) \mapsto [\![E_3]\!]^{\epsilon}_{\rho} \\
H, L, \epsilon, l_{this} \models \mathtt{emp} & \Leftrightarrow H = \mathsf{emp} \\
H, L, \epsilon, l_{this} \models \exists X.P & \Leftrightarrow \exists V \in \mathcal{V}^L.\ H, \epsilon, l_{this}, \epsilon[X \mapsto V] \models P \\
H, L, \epsilon, l_{this} \models \forall X.P & \Leftrightarrow \forall V \in \mathcal{V}^L.\ H, \epsilon, l_{this}, \epsilon[X \mapsto V] \models P \\
H, L, \epsilon, l_{this} \models \sigma(x, E) & \Leftrightarrow \big(L{=}l :: L' \wedge H(l, x){=}[\![E]\!]_{\epsilon, l_{this}}\big) \vee \big(L{=}[\,] \wedge H, L, \epsilon, l_{this} \models \pi(l_g, x, E)\big) \\
 & \qquad \vee \big(L = l :: L'\ \wedge\ H(l, x) = \varnothing\ \wedge\ H, L', \epsilon, l_{this} \models \sigma(x, E)\big) \\
H, L, \epsilon, l_{this} \models \pi(E_0, E_1, E_2) & \Leftrightarrow \exists l, x, l'.\ [\![E_0]\!]_{\epsilon, l_{this}} = l \wedge [\![E_1]\!]_{\epsilon, l_{this}} = x \\
 & \qquad \wedge (H(l, x){=}[\![E_2]\!]_{\epsilon}\ \vee\ (H(l, x){=}\varnothing \wedge H(l, @proto){=}l' \wedge\ H, L, \epsilon, l_{this} \models \pi(l', x, \omega)))
\end{array}
$$

JSIL: Logical Values, Logical Expressions, Assertions

$$
\begin{array}{l}
V \in \mathcal{V}^L_{\mathtt{JSIL}} ::= \omega \mid \varnothing \\
E \in \mathcal{E}^L_{\mathtt{JSIL}} ::= V \mid \mathrm{X} \mid \ominus\ E \mid E \oplus E \mid \mathsf{typeof}\ (E) \mid E :: E \mid \mathtt{x} \mid \mathsf{ref}_{\mathtt{a}}(E, E) \mid \mathsf{base}\ (E) \mid \mathsf{field}\ (E)
\end{array}
$$

$$
\begin{array}{lll}
P, Q ::= E_1 = E_2 \mid E_1 \leq E_2 & & \text{Boolean} \\
\quad \mid P \vee Q \mid P \wedge Q \mid \neg P \mid \mathrm{true} \mid \mathrm{false} \mid \exists \mathtt{x}.P \mid \forall \mathtt{x}.P & & \text{Classical} \\
\quad \mid \mathtt{emp} \mid (E, E) \mapsto E \mid P * Q & & \text{JS heaps}
\end{array}
$$

Semantics of JSIL Logical Expressions

Logical Expressions (Semantics):

$$
\begin{array}{lcl@{\qquad}lcl}
[\![V]\!]^{\epsilon}_{\rho} & \triangleq & V & [\![\mathtt{x}]\!]^{\epsilon}_{\rho} & \triangleq & \rho(\mathtt{x}) \\
[\![\mathrm{X}]\!]^{\epsilon}_{\rho} & \triangleq & \epsilon(\mathrm{X}) & [\![\mathsf{ref}_{\mathtt{a}}(E_1, E_2)]\!]^{\epsilon}_{\rho} & \triangleq & l.{}_{\mathtt{a}}x, \text{ where: } l = [\![E_1]\!]^{\epsilon}_{\rho} \text{ and } x = [\![E_2]\!]^{\epsilon}_{\rho} \\
[\![\ominus\ E]\!]^{\epsilon}_{\rho} & \triangleq & \ominus([\![E]\!]^{\epsilon}_{\rho}) & [\![\mathsf{base}\ (E)]\!]^{\epsilon}_{\rho} & \triangleq & l, \text{ where } l.{}_{\mathtt{a}}x = [\![E_1]\!]^{\epsilon}_{\rho} \\
[\![E_1 \oplus E_2]\!]^{\epsilon}_{\rho} & \triangleq & [\![E_1]\!]^{\epsilon}_{\rho} \oplus [\![E_1]\!]^{\epsilon}_{\rho} & [\![\mathsf{field}\ (E)]\!]^{\epsilon}_{\rho} & \triangleq & x, \text{ where } l.{}_{\mathtt{a}}x = [\![E_1]\!]^{\epsilon}_{\rho} \\
[\![\mathsf{typeof}\ \ E]\!]^{\epsilon}_{\rho} & \triangleq & \mathsf{TypeOf}([\![E]\!]^{\epsilon}_{\rho}) & & & \\
[\![E_1 :: E_2]\!]^{\epsilon}_{\rho} & \triangleq & [\![E_1]\!]^{\epsilon}_{\rho} :: [\![E_2]\!]^{\epsilon}_{\rho} & & &
\end{array}
$$

Translation from JS Logic Values/Expressions/Assertions to JSIL Logic Values/Expressions/Assertions

Logical Values : $\mathcal{T}_v : \mathcal{V}^L_{\mathtt{JS}} \rightharpoonup \mathcal{V}^L_{\mathtt{JSIL}}$

$$\mathcal{T}_v(\varnothing) \triangleq \varnothing \qquad \mathcal{T}_v(\lambda) \triangleq \lambda \qquad \mathcal{T}_{\beta,v}(l) \triangleq \beta(l)$$

$$\mathcal{T}_v(r) \triangleq r \qquad \mathcal{T}_v(\lambda\overline{x}.s^m) \triangleq m \qquad \mathcal{T}_v([\,]) = [\,]$$

$$\frac{\mathcal{T}_v(V) = V' \quad \mathcal{T}_v(\overline{V}) = \overline{V}'}{\mathcal{T}_v(V :: \overline{V}) \triangleq V' :: \overline{V}'}$$

Logical Environments : $\mathcal{T}_\epsilon : \mathcal{E}nv_{\mathtt{JS}} \rightharpoonup \mathcal{E}nv_{\mathtt{JSIL}}$

$$\mathcal{T}_\epsilon(\epsilon) = \{(\mathrm{X}, \mathcal{T}_v(V)) \mid (\mathrm{X}, V) \in \epsilon\}$$

Logical Expressions : $\mathcal{T}_e : \mathcal{E}^L_{\mathtt{JS}} \rightharpoonup \mathcal{E}^L_{\mathtt{JSIL}}$

$$\mathcal{T}_e(V) = \mathcal{T}_v(V) \qquad \mathcal{T}_e(\mathrm{X}) = \mathrm{X}$$

$$\frac{\mathcal{T}_e(E) = E'}{\mathcal{T}_e(\ominus E) \triangleq E'} \qquad \frac{\mathcal{T}_e(E_1) = E'_1 \quad \mathcal{T}_e(E_2) = E'_2}{\mathcal{T}_e(E_1 \oplus E_2) \triangleq E'_1 \oplus E'_2}$$

$$\frac{\mathcal{T}_e(E_1) = E'_1 \quad \mathcal{T}_e(E_2) = E'_2}{\mathcal{T}_e(E_1 :: E_2) \triangleq E'_1 :: E'_2}$$

Specifications

$$\frac{\mathcal{T}_a(P) = P' \quad \mathcal{T}_a(Q) = Q' \quad \psi^+(m) = m_1 :: \ldots :: m_n \quad P'' = \circledast_{i=1}^n(\mathtt{x}_{sc}, m_i) \mapsto Y_{m_i}}{\mathcal{T}(\{P\}\, m(\overline{\mathtt{x}})\, \{Q\}) \triangleq \{P'' * P'\}\, m(\mathtt{x}_{sc}, \mathtt{x}_{this}, \overline{\mathtt{x}})\, \{P'' * Q'\}}$$

Assertions : $\mathcal{T}_a : \mathcal{AS}_{\mathtt{JS}} \rightharpoonup \mathcal{AS}_{\mathtt{JSIL}}$

$$\mathcal{T}_a(\text{true}) = \text{true} \qquad \mathcal{T}_a(\text{false}) = \text{false} \qquad \frac{\mathcal{T}_a(P) = P'}{\mathcal{T}_a(\neg P) \triangleq \neg P'}$$

$$\frac{\mathcal{T}_a(P) = P' \quad \mathcal{T}_a(Q) = Q'}{\mathcal{T}_a(P \wedge Q) \triangleq P' \wedge Q'} \qquad \frac{\mathcal{T}_a(P) = P'}{\mathcal{T}_a(\exists X.P) \triangleq \exists X.P'}$$

$$\frac{\mathcal{T}_a(P) = P'}{\mathcal{T}_a(\forall X) \triangleq \forall X.P'} \qquad \frac{\mathcal{T}_e(E_1) = E'_1 \quad \mathcal{T}_e(E_2) = E'_2}{\mathcal{T}_a(E_1 = E_2) \triangleq E'_1 = E'_2}$$

$$\frac{\mathcal{T}_e(E_1) = E'_1 \quad \mathcal{T}_e(E_2) = E'_2}{\mathcal{T}_a(E_1 \leq E_2) \triangleq E'_1 \leq E'_2} \qquad \mathcal{T}_a(\mathsf{emp}) = \mathsf{emp}$$

$$\frac{\mathcal{T}_a(P) = P' \quad \mathcal{T}_a(Q) = Q'}{\mathcal{T}_a(P * Q) \triangleq P' * Q'}$$

$$\frac{\mathcal{T}_e(E_1) = E'_1 \quad \mathcal{T}_e(E_2) = E'_2 \quad \mathcal{T}_e(E_3) = E'_3}{\begin{array}{l}\mathcal{T}_a((E_1, E_2) \mapsto E_3) \triangleq \big(E'_2 \neq \mathtt{@}scope * (E'_1, E'_2) \mapsto E'_3\big) \\ \quad \vee \big(\exists X_i|_{i=1}^n, L_i|_{i=1}^n, X_m, X_{sc}, L_m.\ E'_2 \doteq \mathtt{@}scope \\ \qquad * (E'_1, E'_2) \mapsto X_{sc} \wedge \circledast_{i=1}^n(X_{sc}, X_i) \mapsto L_i \\ \qquad * (X_{sc}, X_m) \mapsto L_m \wedge E'_3 \doteq ((X_i, L_i)|_{i=1}^n, X_m)\big)\end{array}}$$

$$\frac{\mathcal{T}(E) = E' \quad \psi^+_m(x) = m_1 :: \ldots :: m_n}{\mathcal{T}_a(\sigma(x, E)) \triangleq \circledast_{i=1}^{n-1}(Y_{m_i}, x) \mapsto \varnothing \ * \ (Y_{m_n}, x) \mapsto E'}$$

Theorem 1 (Assertion translation correctness)**.** *For any JS and JSIL abstract heaps $H_{\mathtt{JS}}$ and $H_{\mathtt{JSIL}}$, scope chain list L, location l, JSIL program $\mathtt{p}$, and store ρ such that $L, l, H_{\mathtt{JS}} \simeq_\beta \mathtt{p}, H_{\mathtt{JSIL}}, \rho$, there exist a JSIL heap $H'_{\mathtt{JS}}$ and locations l_{scp} and l_{sc} such that $H_{\mathtt{JSIL}} = H'_{\mathtt{JSIL}} \uplus \mathtt{SC}_0(l_{scp}) \uplus \mathtt{SC}_\beta(L, l_{sc})$ and $H_{\mathtt{JS}}, L, \epsilon \models P$ if and only if $H'_{\mathtt{JSIL}}, \rho, \mathcal{T}(\epsilon) \models \mathcal{T}(P)$.*

Proof.

$$H_{\mathtt{JS}}, L, \epsilon_{\mathtt{JS}} \models P_{\mathtt{JS}} \ \Leftrightarrow\ H_{\mathtt{JSIL}}, \rho, \mathcal{T}(\epsilon_{\mathtt{JS}}) \models \mathcal{T}(P_{\mathtt{JS}})$$

when

$$H_{\mathtt{JS}}, L \sim_\beta H_{\mathtt{JSIL}}, \rho$$

By total induction on the size of P.

1. $P = \text{true}$, $P = \text{false}$, or $P = \mathsf{emp}$. Directly from the definitions of satisfiability and the translation.

2. $P = \exists X.P'$. From this, we have that there exists a $V \in \mathcal{V}^L_{\mathtt{JS}}$, such that $H, L, \epsilon[X \mapsto V] \models P'$. By the induction hypothesis, then there exist $H \in \mathcal{H}^\emptyset_{\mathtt{JSIL}}$ such that $H \sim_\beta H$, and $\rho \in \mathcal{S}to$, such that $H, \rho, \mathcal{T}_\epsilon(\epsilon)[X \mapsto \mathcal{T}_v(V)] \models \mathcal{T}_a(P')$. This, by definition of satisfiability, means that $H, \rho, \mathcal{T}_\epsilon(\epsilon) \models \mathcal{T}_a(\exists X.P')$.

3. $P = \forall X.P'$, $P = \neg P'$, and $P = P' \wedge Q'$. Same as previous, directly from the induction hypothesis.

4. $P = (E_1 = E_2)$ or $P = (E_1 \leq E_2)$. These two cases are heap-independent. By total induction on the combined size of E_1 and E_2.

5. $P = P * Q$. From this, we have that there exist H_P and H_Q, such that $H \equiv H_P \uplus H_Q$, $H_P, L, \epsilon \models P$, and $H_Q, L, \epsilon \models Q$. By the induction hypothesis, we have that there exist heaps H'_P and H'_Q, such that $H_P \sim_\beta H'_P$, $H_Q \sim_\beta H'_Q$, and for all $\rho \in \mathcal{S}to$, it holds that $H'_P, \rho, \mathcal{T}_\epsilon(\epsilon) \models \mathcal{T}_a(P)$ and $H'_Q, \rho, \mathcal{T}_\epsilon(\epsilon) \models \mathcal{T}_a(Q)$. Due to the β-indistinguishability rule for function objects, where the choice of l_{sc} is random, at this point we cannot guarantee that H'_P and H'_Q are disjoint. In the case that they are, we can take $H \equiv H'_P \uplus H'_Q$. By the β-indistinguishability rule for heap composition, we have that $H \equiv H_P \uplus H_Q \sim_\beta H'_P \uplus H'_Q \equiv H$. Finally, by the definition of satisfiability, we have that $H, \rho, \mathcal{T}_\epsilon(\epsilon) \models \mathcal{T}_a(P * Q)$. On the other hand, if H'_P and H'_Q are not disjoint, it means that there exists conflicting choices of l_{sc} for various function objects. This, however, can be remedied by renaming some of these locations to fresh ones so that all clashes. Here, it is important to note that this renaming preserves β-indistinguishability. After the renaming, we proceed as in the case when H'_P and H'_Q are disjoint.

6. $P = (E_1, E_2) \mapsto E_3$. From this, we have that

$$H \equiv (\llbracket E_1 \rrbracket_\epsilon, \llbracket E_2 \rrbracket_\epsilon) \mapsto \llbracket E_3 \rrbracket_\epsilon,$$

meaning that $\llbracket E_1 \rrbracket_\epsilon = l$, $\llbracket E_2 \rrbracket_\epsilon = x$, $\llbracket E_3 \rrbracket_\epsilon = \omega$, Note that x has to be an internal property due to the well-formedness of H. The relevant translation rule is:

$$\frac{\mathcal{T}_e(E_1) = E_1' \quad \mathcal{T}_e(E_2) = E_2' \quad \mathcal{T}_e(E_3) = E_3'}{\mathcal{T}_a((E_1, E_2) \mapsto E_3) \triangleq (E_1', E_2') \mapsto E_3'}.$$

If $x \notin \{@scope, @body\}$ then, by the correctness of H, we have that $\omega = v$, for some $v \in \mathcal{V}_{\mathtt{JS}}$.

The remaining cases follow analogously. □

Lemma 5 (Proving JS Hoare Triple in JSIL)**.** *For every JS statement s and JS assertions P and Q:*

$$\models \{P_{\mathtt{JS}}\}\, s\, \{Q_{\mathtt{JS}}\} \quad \Leftrightarrow \quad \models \{\mathcal{T}(P_{\mathtt{JS}})\}\, \mathcal{C}(s)\, \{\mathcal{T}(Q_{\mathtt{JS}})\}$$

$$\vdash \{P_{\mathtt{JSIL}}\}\, \mathtt{p}\, \{Q_{\mathtt{JSIL}}\} \quad \Rightarrow \quad \models \{P_{\mathtt{JSIL}}\}\, \mathtt{p}\, \{Q_{\mathtt{JSIL}}\}$$

$$\vdash \{\mathcal{T}(P_{\mathtt{JS}})\}\, \mathcal{C}(s)\, \{\mathcal{T}(Q_{\mathtt{JS}})\} \quad \Rightarrow \quad \models \{P_{\mathtt{JS}}\}\, s\, \{Q_{\mathtt{JS}}\}$$

4.2 JSIL Logic

Basic Commands - Inference Rules: $\{P\}\mathtt{c}\{Q\}$

SKIP
$$\{\mathtt{emp}\}\ \mathsf{skip}\ \{\mathtt{emp}\}$$

ASSIGNMENT
$$\{P\}\ \mathtt{x} := \mathtt{e}\ \{\exists \mathrm{X}.\ P[\mathrm{X}/\mathtt{x}] * \mathtt{x} \doteq \mathtt{e}[\mathrm{X}/\mathtt{x}]\}$$

Object Creation
$$\{\mathtt{emp}\}\ \mathtt{x} := \mathsf{new}\ ()\ \{(\mathtt{x}, @proto) \mapsto \mathsf{null} * \mathsf{EmptyFields}(\mathtt{x}, \{@proto\})\}$$

MEMBER CHECK - TRUE
$$\frac{P = (\mathtt{e}_1, \mathtt{e}_2) \mapsto \mathrm{V} * V \not\doteq \varnothing}{\{P\}\ \mathtt{x} := \mathsf{hasField}\ (\mathtt{e}_1, \mathtt{e}_2)\ \{P * \mathtt{x} \doteq \mathsf{true}\}}$$

MEMBER CHECK - FALSE
$$\frac{P = (\mathtt{e}_1, \mathtt{e}_2) \mapsto \varnothing}{\{P\}\ \mathtt{x} := \mathsf{hasField}\ (\mathtt{e}_1, \mathtt{e}_2)\ \{P * \mathtt{x} \doteq \mathsf{false}\}}$$

FIELD PROJECTION
$$\frac{P = (\mathtt{e}_1, \mathtt{e}_2) \mapsto \mathrm{X} * \mathrm{X} \not\doteq \varnothing}{\{P\}\ \mathtt{x} := [\mathtt{e}_1, \mathtt{e}_2]\ \{P * \mathtt{x} \doteq \mathrm{X}\}}$$

FIELD ASSIGNMENT
$$\{(\mathtt{e}_1, \mathtt{e}_2) \mapsto \_\}\ [\mathtt{e}_1, \mathtt{e}_2] := e_3\ \{(\mathtt{e}_1, \mathtt{e}_2) \mapsto \mathtt{e}_3\}$$

DELETION
$$\frac{P = (\mathtt{e}_1, \mathtt{e}_2) \mapsto V * V \not\doteq \varnothing}{\{P\}\ \mathsf{delete}\ (\mathtt{e}_1, \mathtt{e}_2)\ \{(\mathtt{e}_1, \mathtt{e}_2) \mapsto \varnothing\}}$$

Definition 4 (Weak Locality of Basic Commands)**.** A JSIL basic command $\mathtt{bc}$ is weakly local with respect to an assertion P if and only if for any assertion Q, abstract heap $H \in \mathcal{H}^{\varnothing}$, store $\rho \in \mathcal{S}to$, and logical environment $\epsilon \in \mathcal{E}nv$, such that $\{P\}\,\mathtt{bc}\,\{Q\}$ and $H, \rho, \epsilon \models P$ the following two properties hold:

- *Frame Property: for all $\hat{H}, \hat{H}_f, \rho_f$, and v, it holds that:*

$$[\![\mathtt{bc}]\!]_{\lfloor H \uplus \hat{H} \rfloor, \rho} = (\lfloor \hat{H}_f \rfloor, \rho_f, v) \implies \exists H_f.\ [\![\mathtt{bc}]\!]_{\lfloor H \rfloor, \rho} = (\lfloor H_f \rfloor, \rho_f, v) \ \wedge\ \hat{H}_f = H \uplus \hat{H}_f$$

- *Safety Monotonicity:*

$$\forall h_f, \hat{H}.\ [\![\mathtt{bc}]\!]_{\lfloor H \rfloor, \rho} = (h_f, \rho_f, v) \implies \exists \hat{h}_f.\ [\![\mathtt{bc}]\!]_{\lfloor H \uplus \hat{H} \rfloor, \rho} = (\hat{h}_f, \rho_f, v)$$

Symbolic Execution: $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$

Basic Command
$$\frac{\mathtt{p}_m(i) = \mathtt{bc} \in \mathtt{BCmd} \quad \{P\}\,\mathtt{bc}\,\{Q\}}{\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q}$$

Goto
$$\frac{\mathtt{p}_m(i) = \mathsf{goto}\ j}{\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow P}$$

Cond. Goto
$$\frac{\mathtt{p}_m(i) = \mathsf{goto}\ [\mathtt{e}]\ j,\ k}{\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow P}$$

Procedure Call
$$\frac{\mathtt{p}_m(i) = \mathtt{x} := \mathtt{e}_0(\mathtt{e}_1, ..., \mathtt{e}_k) \quad \mathtt{S}(m') = \{P\}\ m'(\mathtt{x}_1, ..., \mathtt{x}_n)\ \{Q * \mathtt{x}_{\mathtt{ret}} \doteq \mathtt{e}\} \quad \forall_{k<j\leq n}\ \mathtt{e}_j = \mathsf{undefined}}{\mathtt{p}, \mathtt{S}, i \vdash_m (P[\mathtt{e}_1/\mathtt{x}_1, ..., \mathtt{e}_k/\mathtt{x}_k] * \mathtt{e}_0 \doteq m') \rightsquigarrow (Q[\mathtt{e}_1/\mathtt{x}_1, ..., \mathtt{e}_n/\mathtt{x}_n] * \mathtt{e}_0 \doteq m' * \mathtt{x} \doteq \mathtt{e}[\mathtt{e}_1/\mathtt{x}_1, ..., \mathtt{e}_n/\mathtt{x}_n])}$$

Normal Return
$$\mathtt{p}, \mathtt{S}, i_{ret} \vdash_m P \rightsquigarrow P$$

Error Return
$$\mathtt{p}, \mathtt{S}, i_{err} \vdash_m P \rightsquigarrow P$$

Frame Rule
$$\frac{\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q}{\mathtt{p}, \mathtt{S}, i \vdash_m P * R \rightsquigarrow Q * R}$$

Elimination
$$\frac{\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q}{\mathtt{p}, \mathtt{S}, i \vdash_m \exists \mathrm{X}.\ P \rightsquigarrow \exists \mathrm{X}.\ Q}$$

Consequence
$$\frac{\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q \quad P' \Rightarrow P \quad Q \Rightarrow Q'}{\mathtt{p}, \mathtt{S}, i \vdash_m P' \rightsquigarrow Q'}$$

Lemma 6 (Soundness of Basic Commands)**.** *All basic commands satisfy soundness: for any basic command $\mathtt{bc} \in \mathtt{BCmd}$, abstract heap $H \in \mathcal{H}^{\varnothing}$, store $\rho \in \mathcal{S}to$, logical environment $\epsilon \in \mathcal{E}nv$, and assertions $P, Q \in \mathcal{AS}$ such that: $\{P\}\,\mathtt{bc}\,\{Q\}$ and $H, \rho, \epsilon \models P$, it holds that:*

$$\exists_{H', \rho'}.\ [\![\mathtt{bc}]\!]_{\lfloor H \rfloor, \rho} = (\lfloor H' \rfloor, \rho', v) \wedge H', \rho', \epsilon \models Q$$

Proof. We proceed by case analysis on the rule applied to get $\{P\}\,\mathtt{bc}\,\{Q\}$. For convenience, we name the hypotheses of the lemma:

- **Hyp.** 1: $\{P\}\,\mathtt{bc}\,\{Q\}$ and
- **Hyp.** 2: $H, \rho, \epsilon \models P$

We need to prove that there is an abstract heap H', a store ρ', and a value v such that:

- **Goal** 1: $[\![\mathtt{bc}]\!]_{\lfloor H \rfloor, \rho} = (\lfloor H' \rfloor, \rho', v)$
- **Goal** 2: $H', \rho', \epsilon \models Q$

[SKIP] It follows that $\mathtt{bc} = \mathsf{skip}$ and, applying **Hyp.** 1, that $P = Q = \mathtt{emp}$. Hence, using **Hyp.** 2, we have that $H = \mathsf{emp}$. We conclude that $H' = \mathsf{emp}$, $\rho' = \rho$, and $v = \mathsf{empty}$, because $[\![\mathsf{skip}]\!]_{\mathsf{emp}, \rho} = (\mathsf{emp}, \rho, \mathsf{empty})$. Finally, noting that $\mathsf{emp}, \rho, \epsilon \models \mathtt{emp}$, we conclude that: $H', \rho', \epsilon \models Q$.

[ASSIGNMENT] It follows that $\mathtt{bc} = \mathtt{x} := \mathtt{e}$ and, applying **Hyp.** 1, that $Q = \exists \mathrm{X}.\ P[\mathrm{X}/\mathtt{x}] * \mathtt{x} \doteq \mathtt{e}[\mathrm{X}/\mathtt{x}]$. Using the semantics of JSIL, we conclude that:

$$(\lfloor H' \rfloor, \rho', v) = [\![\mathtt{x} := \mathtt{e}]\!]_{\lfloor H \rfloor, \rho} = (\lfloor H \rfloor, \rho[\mathtt{x} \mapsto [\![\mathtt{e}]\!]_\rho], [\![\mathtt{e}]\!]_\rho)$$

Observe that:

$$\begin{array}{l} H, \rho, \epsilon \models P \\ \Rightarrow\ H, \rho[\mathtt{x} \mapsto [\![\mathtt{e}]\!]_\rho], \epsilon[\mathrm{X} \mapsto \rho(\mathtt{x})] \models P[\mathrm{X}/\mathtt{x}] \\ \Rightarrow\ H, \rho[\mathtt{x} \mapsto [\![\mathtt{e}]\!]_\rho], \epsilon[\mathrm{X} \mapsto \rho(\mathtt{x})] \models P[\mathrm{X}/\mathtt{x}] \ \wedge\ \mathtt{x} = \mathtt{e}[\mathrm{X}/\mathtt{x}] \\ \Rightarrow\ \exists V.\ H, \rho[\mathtt{x} \mapsto [\![\mathtt{e}]\!]_\rho], \epsilon[\mathrm{X} \mapsto V] \models P[\mathrm{X}/\mathtt{x}] \ \wedge\ \mathtt{x} = \mathtt{e}[\mathrm{X}/\mathtt{x}] \\ \Rightarrow\ H, \rho[\mathtt{x} \mapsto [\![\mathtt{e}]\!]_\rho], \epsilon \models \exists \mathrm{X}.\ P[\mathrm{X}/\mathtt{x}] \ \wedge\ \mathtt{x} = \mathtt{e}[\mathrm{X}/\mathtt{x}] \\ \Rightarrow\ H, \rho', \epsilon \models Q \end{array}$$

Hence, if we let $H' = H$, it follows that: $H', \rho', \epsilon \models Q$, which concludes the proof.

[Object Creation] It follows that $\mathtt{bc} = \mathtt{x} := \mathsf{new}\,()$ and, applying **Hyp.** 1, that $P = \mathsf{emp}$ and $Q = (\mathtt{x}, @\mathit{proto}) \mapsto \mathsf{null} * \mathsf{EmptyFields}(\mathtt{x}, \{@\mathit{proto}\})$. Hence, using **Hyp.** 2, we have that $H = \mathsf{emp}$. Furthermore, using the semantics of JSIL, we know that:

$$(h', \rho', v) = [\![\mathtt{x} := \mathsf{new}\,()]\!]_{\mathsf{emp},\rho} = ((l, @\mathit{proto}) \mapsto \mathsf{null}, \rho[\mathtt{x} \mapsto l], l)$$

Hence, if we let:

$$H' = \Big(\biguplus_{p \neq @\mathit{proto}} (l, p) \mapsto \varnothing \Big) \uplus (l, @\mathit{proto}) \mapsto \mathsf{null}$$

it follows that $\lfloor H' \rfloor = h'$ and $H', \rho', \epsilon \models Q$, which concludes the proof.

[Member Check - True] It follows that $\mathtt{bc} = \mathtt{x} := \mathsf{hasField}\,(\mathtt{e}_1, \mathtt{e}_2)$ and, applying **Hyp.** 1, that $P = (\mathtt{e}_1, \mathtt{e}_2) \mapsto \mathrm{V} \wedge V \neq \varnothing$ and $Q = P * \mathtt{x} \doteq \mathsf{true}$. Hence, using **Hyp.** 2, we have that:

$$H = ([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \mapsto \epsilon(\mathrm{V})$$

Note that $\lfloor H \rfloor = H$; therefore, using the semantics of JSIL, we conclude that:

$$(h', \rho', v) = [\![\mathtt{x} := \mathsf{new}\,()]\!]_{H,\rho} = (H, \rho[\mathtt{x} \mapsto \mathsf{true}], \mathsf{true})$$

Hence, letting $H' = H$ and assuming that $\mathtt{x} \notin \mathsf{vars}(\mathtt{e}_1) \cup \mathsf{vars}(\mathtt{e}_2)$, we conclude that $H', \rho', \epsilon \models Q$.

[Member Check - False] Analogous to the previous case.

[Field Projection] It follows that $\mathtt{bc} = \mathtt{x} := [\mathtt{e}_1, \mathtt{e}_2]$ and, applying **Hyp.** 1, that $P = (\mathtt{e}_1, \mathtt{e}_2) \mapsto \mathrm{V} * \mathrm{V} \not\doteq \varnothing$ and $Q = P * \mathtt{x} \doteq \mathrm{V}$, for some logical variable V. Hence, using **Hyp.** 2, we have that:

$$H = ([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \mapsto \epsilon(\mathrm{V})$$

Note that $\lfloor H \rfloor = H$; therefore, using the semantics of JSIL, we conclude that:

$$(h', \rho', v) = [\![\mathtt{x} := \mathsf{new}\,()]\!]_{H,\rho} = (H, \rho[\mathtt{x} \mapsto \epsilon(\mathrm{V})], \epsilon(\mathrm{V}))$$

Hence, letting $H' = H = h'$ and assuming that $\mathtt{x} \notin \mathsf{vars}(\mathtt{e}_1) \cup \mathsf{vars}(\mathtt{e}_2)$, we conclude that $H', \rho', \epsilon \models Q$. More concretely, we know that $H', \rho', \epsilon \models P$ from **Hyp.** 2 and we know $\mathsf{emp}, \rho', \epsilon \models \mathtt{x} \doteq \mathrm{V}$ from the definition of $\models$.

[Field Deletion] It follows that $\mathtt{bc} = \mathsf{delete}\,(\mathtt{e}_1, \mathtt{e}_2)$ and, applying **Hyp.** 1, that $P = (\mathtt{e}_1, \mathtt{e}_2) \mapsto V * V \not\doteq \varnothing$ and $Q = (\mathtt{e}_1, \mathtt{e}_2) \mapsto \varnothing$, for some logical variable V. Hence, using **Hyp.** 2, we have that:

$$H = ([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \mapsto \epsilon(\mathrm{V})$$

Note that $\lfloor H \rfloor = H$; therefore, using the semantics of JSIL, we conclude that:

$$(h', \rho', v) = [\![\mathsf{delete}\,(\mathtt{e}_1, \mathtt{e}_2)]\!]_{H,\rho} = (\mathsf{emp}, \rho[\mathtt{x} \mapsto \mathsf{true}], \mathsf{true})$$

Hence, letting $H' = ([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \mapsto \varnothing$ and assuming that $\mathtt{x} \notin \mathsf{vars}(\mathtt{e}_1) \cup \mathsf{vars}(\mathtt{e}_2)$, we conclude that $H', \rho', \epsilon \models Q$ and $\lfloor H' \rfloor = h'$.

[Field Assignment] It follows that $\mathtt{bc} = [\mathtt{e}_1, \mathtt{e}_2] := \mathtt{e}_3$ and, applying **Hyp.** 1, that $P = (\mathtt{e}_1, \mathtt{e}_2) \mapsto \_$ and $Q = (\mathtt{e}_1, \mathtt{e}_2) \mapsto \mathtt{e}_3$. Hence, using **Hyp.** 2, we have that:

$$H = ([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \mapsto v$$

for some value v. Note that $\lfloor H \rfloor = H$; therefore, using the semantics of JSIL, we conclude that:

$$(h', \rho', v) = [\![[\mathtt{e}_1, \mathtt{e}_2] := \mathtt{e}_3]\!]_{H,\rho} = (([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \mapsto [\![\mathtt{e}_3]\!]_\rho, \rho, [\![\mathtt{e}_3]\!]_\rho)$$

Hence, letting $H' = h' = ([\![\mathtt{e}_1]\!]_\rho, [\![\mathtt{e}_2]\!]_\rho) \mapsto [\![\mathtt{e}_3]\!]_\rho$, we conclude that $H', \rho', \epsilon \models Q$. □

Lemma 7 (Weak Locality of Basic Commands)**.** *All basic commands satisfy the weak locality property; i.e. they exhibit the safety monotonicity and frame properties.*

Proof. As in the proof of Lemma 6, we proceed by case analysis on the rule applied to get $\{P\}\,\mathtt{bc}\,\{Q\}$, and the proof is fully analogous. □

Theorem 2 (Soundness)**.** *For any derivation* $\mathtt{pd} \in \mathcal{D}$*, program* $\mathtt{p} \in \mathtt{P}$*, specification function* $\mathtt{S} \in \mathcal{S}tr \rightharpoonup \mathcal{S}pec$*, abstract heap* $H \in \mathcal{H}^{\varnothing}$*, store* $\rho \in \mathcal{S}to$*, logical environment* ϵ*, procedure name* m*, and command label* i *such that:*

- $\mathtt{pd} \vdash \mathtt{p}, \mathtt{S}$
- $H, \rho, \epsilon \models \mathtt{pd}(m, i)$
- $\mathtt{p} \vdash \langle \lfloor H \rfloor, \rho, i \rangle \Downarrow_m \langle h_f, \rho_f, v \rangle$

It follows that there is an abstract heap H_f *such that:* $\lfloor H_f \rfloor = h_f$ *and* $H_f, \rho_f, \epsilon \models \mathtt{pd}(m, i_{ret})$*.*

Proof. For convenience we name the hypotheses:

- **Hyp.** 1: $\mathtt{pd} \vdash \mathtt{p}, \mathtt{S}$
- **Hyp.** 2: $H, \rho, \epsilon \models \mathtt{pd}(m, i)$

- **Hyp.** 3: $\mathtt{p} \vdash \langle \lfloor H \rfloor, \rho, i \rangle \Downarrow_m \langle h_f, \rho_f, v \rangle$

We proceed by induction on the derivation of **Hyp.** 2.

[Basic Command] It follows that $\mathtt{p}_m(i) = \mathtt{bc}$ for a given basic command $\mathtt{bc}$. We conclude, using **Hyp.** 3 and the semantics of JSIL, that there is a heap h', a store ρ', and value v', such that:

$$[\![\mathtt{bc}]\!]_{\lfloor H \rfloor, \rho} = (h', \rho', v') \qquad \mathtt{p} \vdash \langle h', \rho', i+1 \rangle \Downarrow_m \langle h_f, \rho_f, v \rangle$$

Using **Hyp.** 1 and **Hyp.** 2, we conclude that there are two assertions P and Q such that: $H, \rho, \epsilon \models P$, $P \in \mathtt{pd}(m, i)$, $Q \in \mathtt{pd}(m, i+1)$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$. We now need to show that there is an abstract heap H' such that: $H', \rho', \epsilon \models Q$ and $\lfloor H' \rfloor = h'$. We prove that such an abstract heap exists by induction on the derivation of $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$. More concretely, given that $[\![\mathtt{bc}]\!]_{\lfloor H \rfloor, \rho} = (h', \rho', v')$, $H, \rho, \epsilon \models P$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$, there must exist an abstract heap H' such that $\lfloor H' \rfloor = h'$ and $H', \rho', \epsilon \models Q$.

- [Basic Command] We conclude that: $\{P\}\, \mathtt{bc}\, \{Q\}$. Applying Lemma 6, it follows that there is an abstract heap H' such that $\lfloor H' \rfloor = h'$ and $H', \rho', \epsilon \models Q$.

- [Frame Rule] We conclude that there are three assertions P', Q', and R', such that: $P = P' * R$, $Q = Q' * R$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P' \rightsquigarrow Q'$. It follows that there are two abstract heaps H_p and H_r such that $H = H_p \uplus H_r$, $H_p, \rho, \epsilon \models P'$, and $H_r, \rho, \epsilon \models R$. Using the frame property of basic commands (Lemma 7), we conclude that there is a heap h'_p such that $[\![\mathtt{bc}]\!]_{\lfloor H_p \rfloor, \rho} = (h'_p, \rho', v')$ and $h' = h'_p \uplus \lfloor H_r \rfloor$. Applying the inner induction hypothesis to $[\![\mathtt{bc}]\!]_{\lfloor H_p \rfloor, \rho} = (h'_p, \rho', v')$, $H_p, \rho, \epsilon \models P'$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P' \rightsquigarrow Q'$, we conclude that there is an abstract heap H'_p such that: $H'_p, \rho', \epsilon \models Q'$ and $\lfloor H'_p \rfloor = h'_p$. Recalling that $H_r, \rho, \epsilon \models R$, we conclude that $H'_p \uplus H_r, \rho', \epsilon \models Q * R$.

- [Consequence] We conclude that there are two assertions P' and Q', such that: $P \Rightarrow P'$, $Q' \Rightarrow Q$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P' \rightsquigarrow Q'$. Since $P \Rightarrow P'$ and $H, \rho, \epsilon \models P$, it follows that $H, \rho, \epsilon \models P'$. Applying the inner induction hypothesis to $[\![\mathtt{bc}]\!]_{\lfloor H \rfloor, \rho} = (h', \rho', v')$, $H, \rho, \epsilon \models P'$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P' \rightsquigarrow Q'$, we conclude that there is an abstract heap H' such that: $H', \rho', \epsilon \models Q'$ and $\lfloor H' \rfloor = h'$. Since $Q' \Rightarrow Q$ and $H, \rho, \epsilon \models Q'$, we conclude that $H, \rho, \epsilon \models P'$.

- [Elimination] We conclude that there are two assertions P' and Q', such that: $P = \exists \mathrm{X}.\, P'$, $Q = \exists \mathrm{X}.\, Q'$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P' \rightsquigarrow Q'$. From $H, \rho, \epsilon \models P$, it follows that there is a value $\mathtt{v}$ such that $H, \rho, \epsilon[\mathrm{X} \mapsto \mathtt{v}] \models P'$. Applying the inner induction hypothesis to $[\![\mathtt{bc}]\!]_{\lfloor H \rfloor, \rho} = (h', \rho', v')$, $H, \rho, \epsilon[\mathrm{X} \mapsto \mathtt{v}] \models P'$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P' \rightsquigarrow Q'$, we conclude that there is an abstract heap H' such that $H', \rho', \epsilon[\mathrm{X} \mapsto \mathtt{v}] \models Q'$ and $\lfloor H' \rfloor = h'$. We can therefore conclude, using the definition of satisfaction relations ($\models$) that there is an abstract heap H' such that: $H', \rho', \epsilon \models Q$.

- [All Other Cases] No other rule may have been applied in the derivation of $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$ because $\mathtt{bc}$ is a basic command. Hence, we do not have to analyse those cases.

Having established that there is an abstract heap H' such that $\lfloor H' \rfloor = h'$ and $H', \rho', \epsilon \models Q$, we conclude that there is an abstract heap H' such that $H', \rho, \epsilon \models \mathtt{pd}(m, i+1)$ and $\mathtt{p} \vdash \langle \lfloor H' \rfloor, \rho', i+1 \rangle \Downarrow_m \langle h_f, \rho_f, v \rangle$. Using **Hyp.** 1, we can apply the induction hypothesis to conclude the claim of the lemma.

[Goto] It follows that $\mathtt{p}_m(i) = \mathsf{goto}\ j$ for a given command label j. We conclude, using **Hyp.** 3 and the semantics of JSIL, that:

$$\mathtt{p} \vdash \langle \lfloor H \rfloor, \rho, j \rangle \Downarrow_m \langle h_f, \rho_f, v \rangle$$

Using **Hyp.** 1 and **Hyp.** 2, we conclude that there are two assertions P and Q such that: $H, \rho, \epsilon \models P$, $P \in \mathtt{pd}(m, i)$, $Q \in \mathtt{pd}(m, j)$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$. We now need to show that there is an abstract heap H' such that $H', \rho, \epsilon \models Q$ and $\lfloor H' \rfloor = \lfloor H \rfloor$. Like in the previous case, we prove that such an abstract heap exists by induction on the derivation of $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$. More concretely, given that $H, \rho, \epsilon \models P$ and $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$, there must exist an abstract heap H' such that $\lfloor H' \rfloor = \lfloor H \rfloor$ and $H', \rho, \epsilon \models Q$.

- [Goto] We conclude that: $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow P$. By letting $H' = H$, it immediately follows that there exists an abstract heap H' such that $\lfloor H' \rfloor = \lfloor H \rfloor$ and $H', \rho, \epsilon \models Q$.

- [Frame Rule, Consequence, Elimination] These cases are similar to those in the previous case.

- [All Other Cases] No other rule may have been applied in the derivation of $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$ because $\mathtt{p}_m(i) = \mathsf{goto}\ j$.

Having established that there is an abstract heap H' such that $\lfloor H' \rfloor = \lfloor H \rfloor$ and $H', \rho, \epsilon \models Q$, we conclude that there is an abstract heap H' such that $H', \rho, \epsilon \models \mathtt{pd}(m, j)$ and $\mathtt{p} \vdash \langle \lfloor H' \rfloor, \rho, j \rangle \Downarrow_m \langle h_f, \rho_f, v \rangle$. Using **Hyp.** 1, we can apply the induction hypothesis to conclude the claim of the lemma.

[Conditional Goto - True] It follows that $\mathtt{p}_m(i) = \mathsf{goto}\ [\mathtt{e}]\ j,\ k$ for two command labels command label j and k and a JSIL expression $\mathtt{e}$. We conclude, using **Hyp.** 3 and the semantics of JSIL, that:

$$[\![\mathtt{e}]\!]_\rho = \mathsf{true} \qquad \mathtt{p} \vdash \langle \lfloor H \rfloor, \rho, j \rangle \Downarrow_m \langle h_f, \rho_f, v \rangle$$

Using **Hyp.** 1 and **Hyp.** 2, we conclude that there are two assertions P and Q such that: $H, \rho, \epsilon \models P$, $P \in \mathtt{pd}(m, i)$, $Q \in \mathtt{pd}(m, j)$, and $\mathtt{p}, \mathtt{S}, i \vdash_m P \ \wedge\ \mathtt{e} = \mathsf{true} \rightsquigarrow Q$. Since $\mathtt{e}$ does not contain any logical variables, it follows that: $[\![\mathtt{e}]\!]_\rho = [\![\mathtt{e}]\!]_{\rho, \epsilon}$. Therefore, we conclude that $[\![\mathtt{e}]\!]_{\rho, \epsilon} = \mathsf{true}$, which, together with $H, \rho, \epsilon \models P$, implies that $H, \rho, \epsilon \models P \ \wedge\ \mathtt{e} = \mathsf{true}$. We now need to show that there is an abstract heap H' such that $H', \rho, \epsilon \models Q$ and $\lfloor H' \rfloor = \lfloor H \rfloor$. We prove that such an abstract heap exists by induction on the derivation of $\mathtt{p}, \mathtt{S}, i \vdash_m P \ \wedge\ \mathtt{e} = \mathsf{true} \rightsquigarrow Q$. More concretely, given that $H, \rho, \epsilon \models P$ and $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$, there must exist an abstract heap H' such that $\lfloor H' \rfloor = \lfloor H \rfloor$ and $H', \rho, \epsilon \models Q$.

- [Goto] We conclude that: $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow P$. By letting $H' = H$, it immediately follows that there exists an abstract heap H' such that $\lfloor H' \rfloor = \lfloor H \rfloor$ and $H', \rho, \epsilon \models Q$.

- [Frame Rule, Consequence, Elimination] These cases are similar to those in the previous case.

- [All Other Cases] No other rule may have been applied in the derivation of $\mathtt{p}, \mathtt{S}, i \vdash_m P \rightsquigarrow Q$ because $\mathtt{p}_m(i) = \mathsf{goto}\ [\mathsf{e}]\ j,\ k$.

□