Proofs: appendix for “Chaperones and Impersonators: Run-time Support for Reasonable Interposition”

Version 5.3.0.16

August 4, 2012

The definition of the approximates relation:

<e₁, s₁> ~ <e₂, s₂> = approximates[ei, s₁, e₂, s₁, ⊥]

approximates[b, s₂, s₂, ((x y) ...)] = (# ((x y) ...))
approximates[n, s₁, n, s₁, ((x y) ...)] = (# ((x y) ...))
approximates[(void), s₁, (void), s₁, ((x y) ...)] = (# ((x y) ...))
approximates[chaperone-vector, s₁, (lambda (x₁ x₂ x₃), x₂, ((x y) ...))] = (# ((x y) ...))
approximates[chaperone-vector, s₁, (loc x₁), s₁, ((x y) ...)] = (# ((x y) ...))
where (lambda (x₁ x₂ x₃) x₂) = s₁(x₁)
approximates[prim₁, s₁, prim₂, s₂, ((x y) ...)] = (# ((x y) ...))
approximates[(loc x₁), s₁, e₁, e₂, s₁, ((x y) ...)] = approximates[l, e₁, ((x y) ...)]
where (chaperone-vector ln o) = s₁(x₁)
approximates[(loc x₁), s₁, e₁, e₂, s₁, ((x y) ...)] = approximates-multi[l, (v₁ ...), s₁, (v₂ ...), s₁, ((x y) ...)]
where (immutable-vector v₁ ... ) = x₁(s₁), (immutable-vector v₂ ... ) = x₂(s₁)
approximates[(loc x₁), s₁, e₁, e₂, s₁, ((x y) ...)] = (# ((x₁, y₁) ... (x₂, y₂) ... ))
approximates[(loc x₁), s₁, e₁, e₂, s₁, ((x y) ...)] = (# ((x₁, y₁) ... (x₂, y₂) ... ))
approximates[(loc x₁), s₁, e₁, e₂, s₁, ((x y) ...)] = (# ((x₁, y₁) ... (x₂, y₂) ... ))
approximates[(loc x₁), s₁, e₁, e₂, s₁, ((x y) ...)] = (# ((x₁, y₁) ... (x₂, y₂) ... ))
approximates[(loc x₁), s₁, e₁, e₂, s₁, ((x y) ...)] = (# ((x₁, y₁) ... (x₂, y₂) ... ))
where e₁ = s₁(x₁), e₂ = s₁(x₂)
approximates[(lambda (x₁ ... ) e₁, s₁), (lambda (x₁ ... ) e₁, s₁, ((x y) ...))] = approximates-multi[l, e₁, e₁, s₁, ((x y) ...)]
approximates[(lambda (x₁ ... ) e₁, s₁), (lambda (x₁ ... ) e₁, s₁, ((x y) ...))] = (# ((x y) ... ))
approximates[(let (x₁ e₁ ... ) e₁, s₁), (let (x₁ e₁ ... ) e₁, s₁, ((x y) ...))] = (# ((x₁, y₁) ... (x₂, y₂) ... ))
approximates[(let (x₁ e₁ ... ) e₁, s₁), (let (x₁ e₁ ... ) e₁, s₁, ((x y) ...))] = (# ((x₁, y₁) ... (x₂, y₂) ... ))
approximates[(let (x₁ e₁ ... ) e₁, s₁), (let (x₁ e₁ ... ) e₁, s₁, ((x y) ...))] = (# ((x₁, y₁) ... (x₂, y₂) ... ))
approximates[(error 'variable), s₁, (error 'variable), s₁, ((x y) ...)] = (# ((x y) ... ))
approximates[(e₁, e₁, e₂, s₁, ((x y) ...))] = (# ((x y) ... ))

The definition of the immutable metawhnack:
immutable[s, (loc x)] = #t
where (vector-immutable v ... ) = s(x)
immutable[s, (loc x)] = immutable[s, l]
where (chaperone-vector l m o) = s(x)
immutable[s, v] = #f

The definition of the equal metafunction:

equal[s, v, v] = v
where (v, ((x y) ... )) = eq-tab[s, v, v, ()]
eq-tab[s, v, v, ((x y) ... )] = (# ((x y) ... ))
eq-tab[s, (loc x1), (loc y1), ((x y) ... )] = (# ((x y) ... ))
where identified[x1, y1, ((x y) ... )]
eq-tab[s, (loc x1), (loc x1), ((x y) ... )] = eq-tab[s, (loc x1), (loc x1), ((x y) ... )]
where (chaperone-vector (loc x1) v1 v2) = s(x1)
eq-tab[s, (loc x1), (loc x1), (loc x2), ((x y) ... )] = eq-tab[s, (loc x1), (loc x1), (loc x2), ((x y) ... )]
where (chaperone-vector (loc x1) v1 v2) = s(x1)
eq-tab[s, (loc x2), (loc x2), ((x y) ... )] = eq-tab[s, (loc x2), (loc x2), ((x y) ... )]
where (chaperone-vector (loc x2) v1 v2) = s(x2)
eq-tab[s, (loc x2), (loc x1), ((x y) ... )] = eqs-tab[s, ((v1 v2) ... ), ((x2 x1) (x y) ... )]
where (vector v1 ... ) = s(x1), (vector v2 ... ) = s(x2), [(v1, ..., )] = [(v2, ... )]
eq-tab[s, (loc x1), (loc x2), (x y) ... )] = eqs-tab[s, ((v1 v2) ... ), ((x1 x2) (x y) ... )]
where (vector-immutable v1 ... ) = s(x1), (vector-immutable v2 ... ) = s(x2), [(v1, ..., )] = [(v2, ... )]
eq-tab[s, v, v, ((x y) ... )] = (# ((x y) ... ))
eqs-tab[s, () , ((x y) ... )] = (# ((x y) ... ))
eqs-tab[s, ((v1 v2) ... ) , ((x y) ... )] = eqs-tab[s, ((v1 v2) ... ), ((x y) ... )]
where (# ((x1 x2) (x y) ... )) = eq-tab[s, v1, v2, ((x y) ... )]
eqs-tab[s, ((v1 v2) ... ), ((x y) ... )] = (# ((x y) ... ))

Store extension:

A store s' is an extension of a store s (that is, s' <= s) if for all locations x in the domain of s, x is in the domain of s', and for all such locations:

1) s(x) = s'(x)
2) s(x) = (vector b v_1 ... v_n) and s'(x) = (vector b v'_1 ... v'_n).

(That is, shared locations must either contain the same term or a mutable vector that cannot differ in either the boolean marker or the length, but only in the stored contents.)

Theorem 1:

For all e, if e is a user-writeable program, Eval(e) = v, and that evaluation contains no reductions where the left-hand side is of the form (s #t (vector-set! (loc x) n v_a)) where s(x) = (vector #f v_v ... ), then Eval(|e|) = v.

(|e| is defined as e[chaperone-vector |-> (lambda (v x y) v)])

(e is user-writeable means e contains no uses of set-marker or clear-marker and contains no values of the form (loc x).)
Lemma 1 (Substitution lemma):
For all $e_1, s_1, e_2, s_2, x ... e_3 ...$ and $e_4 ...$
if $<e_1, s_1> \sim <e_2, s_2>$ and $<e_3, s_1> \sim <e_4, s_4>$ ... then $<e_1[x |-> e_3, ...], s_1> \sim <e_2[x |-> e_4, ...], s_2>$.

Lemma 2 (approximations of unique decomposition):
For all $e_1, s_1, e_2, s_2$.
if $<e_1, s_1> \sim <e_2, s_2>$ and $e_2 = E_2[e_4]$,
then $e_1 = E_1[e_3], <E_1, s_1> \sim <E_2, s_2>$, and $<e_3, s_1> \sim <e_4, s_2>$.

Lemma 3 (context filling honors approximation):
For all $E_1, e_1, s_1, E_2, e_2, s_2$.
if $<e_1, s_1> \sim <e_2, s_2>$ and $<E_1, s_1> \sim <E_2, s_2>$, then $<E_1[e_1], s_1> \sim <E_2[e_2], s_2>$.

General argument for the next four lemmas: approximation ensures that the combination of a value and a store has the same graph structure (ignoring chaperones) as its approximate value/store. Thus, the traversal of that graph structure done by immutable, equal, and chaperone-of will reveal the same result on the approximate value/store as the original value/store, since addition or removal of chaperones does not affect the result of those operations.

Lemma 4 (approximations are likewise equal):
For all $v_1, v_3, s_1, e_2, v_4, s_2$.
if $<v_1, s_1> \sim <v_2, s_2>$ and $<v_3, s_1> \sim <v_4, s_4>$, then $\text{equal}[[s_1, v_1, v_3]] = \text{equal}[[s_2, v_2, v_4]]$.

Lemma 5 (approximations are likewise immutable):
For all $v_1, s_1, v_2, s_2$.
if $<v_1, s_1> \sim <v_2, s_2>$, then $\text{immutable}[[s_1, v_1]] = \text{immutable}[[s_2, v_2]]$.

Lemma 6 (approximations are likewise chaperone-of):
For all $v_1, v_3, s_1, v_2, v_4, s_2$.
if $<v_1, s_1> \sim <v_2, s_2>$ and $<v_3, s_1> \sim <v_4, s_4>$, then $\text{chaperone-of}[[s_1, v_1, v_3]] = \text{chaperone-of}[[s_2, v_2, v_4]]$.

Lemma 7 (chaperones of approximates are approximates):
For all $v_1, v_3, s_1, v_2, s_2$:
If chaperone-of[[s_1, v_3, v_1]] and $<v_1, s_1> \sim <v_2, s_2>$, then $<v_3, s_1> \sim <v_2, s_2>$.
(The $\sim$ relation strips off chaperones when it finds them when checking approximation, so adding one doesn’t change the result.)

Lemma 8 (approximates are still approximates in pure store extensions):
For all $v_1, s_1, s_1', v_2, s_2, s_2'$:
If $<v_1, s_1> \sim <v_2, s_2>$, and $s_1' \sim s_1$,
and $s_2' \sim s_2$,
then $<v_1, s_1'> \sim <v_2, s_2'>$.

(Define $\sim$ to be the same as $\leq$ except with the additional caveat that
if $s_1(x) = (\text{vector}\ #f\ v\ ...)$, then $s_1'(x) = (\text{vector}\ #f\ v\ ...)$.
That is, there are no changes in vectors allocated by the main program.
Since no vectors of the form $(\text{vector}\ #t\ v\ ...)$ are traversed by a
successful approximation, the approximation algorithm will follow
exactly the same path with the same results in the extensions.)

Lemma 9:

For all $e_2$ that do not contain set-marker, get-marker, or chaperone-vector,
Let $E_2[e_6] = e_2$.
If there exists an $s_2, v_2, s_4$.
$<E_2[e_6], #f, s_2>$ reduces to $<E_2[v_2], #f, s_4>$,
For all $e_1$ and $s_1$ such that $<e_1, s_1> \sim <e_2, s_2>$, let $E_1[e_5] = e_1$.
Also, require that the reduction of $<e_1, #f, s_1>$ contains no program
states of the form $<E[(\text{vector-set!}\ \text{(loc}\ x)\ n\ v)], #t, s>$ where
$s(x) = (\text{vector}\ #f\ v_e\ ...)$.
Either:
1) $<e_1, #f, s_1> \text{ diverges}$
2) there exists a $b, s_3$.
   $<e_1, #f, s_1>$ reduces to $<(\text{error}'\text{variable}), b, s_3>$
3) there exists an $e_3, b, s_3$.
   $<e_1, #f, s_1>$ reduces to $<e_3, b, s_3>$ and
   $e_3$ is a stuck state.
4) there exists a $v_1, s_3$.
   $<e_1, #f, s_1>$ reduces to $<E_1[v_1], #f, s_3>$ and
   $<E_1[v_1], s_3> \sim <E_2[v_2], s_4>$.

Proof:

Fix $e_2 = E_2[e_6]$. Retrieve $s_2, v_2, s_4$ from our hypothesis about
reduction (I), and fix $e_1$ and $s_1$. Since we have that $e_1$ and $e_2$ are
approximates in their respective stores (hypothesis II), we
know from Lemma 2 that $<E_1, s_1> \sim <E_2, s_2>$ and $<e_5, s_1> \sim <e_6, s_2>$.
Now we'll induct on the length of the reduction sequence from
$<E_2[e_6], #f, s_2>$ to $<E_2[v_2], #f, s_4>$ and the size of the chaperone
chain (if any) for values in $e_5$. That is, either we make progress by
taking a step in $e_2$, or we make progress by removing a chaperone from
some part of the values present in $e_5$.

$E_2[(\text{lambda}\ (y\ ...)\ e_b)], #f, s_2> \rightarrow <E_2[(\text{loc}\ z)], #f, s_2[z \rightarrow e_6]>$
Since $<e_5, s_1> \sim <e_6, s_2>$, then $e_5$ is either a lambda term
or is the operator 'chaperone-vector' (if $e_6$ is $(\text{lambda}\ (v\ x\ y)\ v)$).
If $e_5 = \text{chaperone-vector},$
$<e_1, #f, s_1> \sim <E_2[(\text{loc}\ z)], #f, s_2[z \rightarrow e_6]>$. Applying the
IH on the rest of the reduction sequence, using $e_1$ and $s_1$ and the
approximation above to discharge the hypothesis, we get our desired
result immediately.

If e_5 is a lambda, then we choose a fresh location w.

\(<E_1[e_5], \#f, s_1> \rightarrow <E_1[(\text{loc } w)], \#f, s_1[w \rightarrow e_5]>\)

By the definition of the approximation location,

\(<E_1[(\text{loc } w)], \#f, s_1[w \rightarrow e_5]> \sim <E_2[(\text{loc } z)], \#f, s_2[z \rightarrow e_6]>,\)

since in each case we’re just introducing an indirection into the store. Since the two locations are fresh, we won’t run into the case where one appears in the mapping but the other doesn’t, and the locations point to approximately equal values (the store addition doesn’t change their approximativeness). Applying the IH to the rest of the reduction sequence, using \(E_1[(\text{loc } w)]\) and \(s_1[w \rightarrow e_5]\) as our new \(e_1\) and \(s_1\) and the approximation above to discharge the hypothesis, we get the rest of the reduction sequence for our old \(e_1\) if we don’t get divergence, a stuck state, or an error. If we do, then \(e_1\) diverges, gets stuck, or errors, respectively.

\(<E_2[((\text{loc } z) v_4 \ldots)], \#f, s_2> \rightarrow <E_2[e_b2[y_2 \rightarrow v_4, \ldots]], \#f, s_2>\)

where \(s_2(z) = (\lambda (y_2 \ldots) e_b2)\)

There are two cases for \(e_5\) from the definition of approximation:

\(e_5 = ((\text{loc } w) v_3 \ldots)\)

Since \(<E_1[((\text{loc } w) v_3 \ldots)], s_1> \sim <E_2[((\text{loc } z) v_3 \ldots)], \#f, s_2>\)

and \(s_2(z) = (\lambda (y_2 \ldots) e_b2)\),

then from approximation we get \(s_1(w) = (\lambda (y_1 \ldots) e_b1)\),

where \(<(\lambda (y_1 \ldots) e_b1), s_1> \sim <(\lambda (y_2 \ldots) e_b2), s_2>\).

Since \(<v_3, s_1> \sim <v_4, s_2>\) for each \(v_3\) and \(v_4\),

\(<E_1[e_b1[y_1 \rightarrow v_3, \ldots]], s_1> \sim <E_2[e_b2[y_2 \rightarrow v_4, \ldots]], s_2>\).

Apply our IH to the rest of the reduction sequence for \(e_2\), using \(E_1[e_b1[y_1 \rightarrow v_3, \ldots]]\) and \(s_1\) as \(e_1\) and \(s_1\) and the approximation above to discharge the hypothesis, and stitch together the reduction sequence we get back with the step we took above in \(e_1\).

\(e_5 = (\text{chaperone-vector } v_3 v_5 v_7)\):

Then \(e_6 = ((\text{loc } z) v_4 v_6 v_8)\) and \(s_2(z) = (\lambda (v x y) v)\),

thus the RHS of the reduction step for \(e_2\) simplifies to

\(<E_2[v_4], \#f, s_2>\).

We know that \(<v_3, s_1> \sim <v_4, s_2>\) from hypothesis II.

Our first step in the reduction of \(e_1\) is:

\(<E_1[(\text{chaperone-vector } v_3 v_5 v_7)], \#f, s_1> \rightarrow <E_1[\text{chaperone-vector } v_3 v_5 v_7]), \#f, s_1[w \rightarrow (\text{chaperone-vector } v_3 v_5 v_7)]>\)

Since \(\sim\) ignores chaperones, \((\text{loc } w)\) points to a chaperone of \(v_3\), and the new store just adds a new mapping and doesn’t change old ones, we have that

\(<E_1[(\text{loc } w)], s_1[w \rightarrow (\text{chaperone-vector } v_3 v_5 v_7)]> \sim <E_2[v_4], s_2>\).

Apply the IH to the rest of the reduction sequence for \(e_2\), using the LHS of the approximation above as our new \(e_1\) and \(s_1\), and then stitch together the results with the single step taken above.

\(<E_2[(\text{error 'variable})], \#f, s_2> \rightarrow <(\text{error 'variable}), \#f, s_2>\)

Breaks the hypothesis that \(<e_2, \#f, s_2>\ reduces to \(<E_2[v_2], \#f, s_4>\).
\[\text{e}_1 = \text{E}_1[(\text{vector } v_3 \ldots)], \#f, s_1 \mapsto (\text{vector } #f v_3 \ldots)]\]

From the approximation hypothesis, we have that \(v_3, s_1 \sim v_4, s_2\) for all \(v_3\) and \(v_4\). By the definition of the approximation location, \(\text{E}_1[(\text{loc } w)], \#f, s_1[w \mapsto (\text{vector } #f v_3 \ldots)]\) \(\sim\) \(\text{E}_2[(\text{loc } z)], \#f, s_2[z \mapsto (\text{vector } #f v_4 \ldots)]\), since in each case we're just introducing an indirection into the store (plus adding the boolean that marks when this vector was allocated).

Since the two locations are fresh, we won't run into the case where one appears in the mapping but the other doesn't, and the locations point to approximately equal values (the store addition doesn't change their approximateness). Applying the IH to the rest of the reduction sequence, using \(\text{E}_1[(\text{loc } w)]\) and \(s_1[w \mapsto (\text{vector } #f v_3 \ldots)]\) as our new \(e_1\) and \(s_1\) and the approximation above to discharge the hypothesis, we get the rest of the reduction sequence for our old \(e_1\). As before, we stitch the reduction step above onto the one (whether divergent, erroring, stuck, or reduced to a value) we get from the IH.

\[\text{e}_1 = \text{E}_1[(\text{vector-immutable } v_3 \ldots)], \#f, s_2 \mapsto (\text{vector-immutable } v_3 \ldots)]\]

From the approximation hypothesis, we have that \(v_3, s_1 \sim v_4, s_2\) for all \(v_3\) and \(v_4\). By the definition of the approximation location, \(\text{E}_1[(\text{loc } w)], \#f, s_1[w \mapsto (\text{vector-immutable } v_3 \ldots)]\) \(\sim\) \(\text{E}_2[(\text{loc } z)], \#f, s_2[z \mapsto (\text{vector-immutable } v_4 \ldots)]\), since in each case we're just introducing an indirection into the store. Since the two locations are fresh, we won't run into the case where one appears in the mapping but the other doesn't, and the locations point to approximately equal values (the store addition doesn't change their approximateness). Applying the IH to the rest of the reduction sequence, using \(\text{E}_1[(\text{loc } w)]\) and \(s_1[w \mapsto (\text{vector-immutable } v_3 \ldots)]\) as our new \(e_1\) and \(s_1\) and the approximation above to discharge the hypothesis, we get the rest of the reduction sequence for our old \(e_1\). As before, we stitch the reduction step above onto the one (whether divergent, erroring, stuck, or reduced to a value) we get from the IH.

\[\text{e}_1 = \text{E}_1[(\text{impersonate-vector } l_2 m_2 o_2)], \#f, s_2 \mapsto (\text{impersonate-vector } l_2 m_2 o_2)]\]

From the approximation hypothesis, we have that \(l_2, m_2, o_2 \sim l_1, m_1, o_1\) for all \(l_2\) and \(l_1\), \(m_2\) and \(m_1\), and \(o_2\) and \(o_1\). By the definition of the approximation location, \(\text{E}_1[(\text{loc } w)], \#f, s_1[w \mapsto (\text{impersonate-vector } l_2 m_2 o_2)]\) \(\sim\) \(\text{E}_2[(\text{loc } z)], \#f, s_2[z \mapsto (\text{impersonate-vector } l_1 m_1 o_1)]\), since in each case we're just introducing an indirection into the store. Since the two locations are fresh, we won't run into the case where one appears in the mapping but the other doesn't, and the locations point to approximately equal values (the store addition doesn't change their approximateness). Applying the IH to the rest of the reduction sequence, using \(\text{E}_1[(\text{loc } w)]\) and \(s_1[w \mapsto (\text{impersonate-vector } l_2 m_2 o_2)]\) as our new \(e_1\) and \(s_1\) and the approximation above to discharge the hypothesis, we get the rest of the reduction sequence for our old \(e_1\). As before, we stitch the reduction step above onto the one (whether divergent, erroring, stuck, or reduced to a value) we get from the IH.
From the approximation hypothesis, we have that \(<l_1, s_1> \sim <l_2, s_2>, <m_1, s_1> \sim <m_2, s_2>, \) and \(<o_1, s_1> \sim <o_2, s_2>\).

By the definition of the approximation location,
\[
\begin{align*}
\langle E_1[(\text{loc } w)], \#f, s_1[w |-> (\text{impersonate-vector } l_1 m_1 o_1)] \rangle & \sim \\
\langle E_2[(\text{loc } z)], \#f, s_2[z |-> (\text{impersonate-vector } l_2 m_2 o_2)] \rangle,
\end{align*}
\]
since in each case we're just introducing an indirection into the store. Since the two locations are fresh, we won't run into the case where one appears in the mapping but the other doesn't, and the locations point to approximately equal values (the store addition doesn't change their approximateness). Applying the IH to the rest of the reduction sequence, using \(E_1[(\text{loc } w)]\) and \(s_1[w |-> (\text{impersonate-vector } l_1 m_1 o_1)]\) as our new \(e_1\) and \(s_1\) and the approximation above to discharge the hypothesis, we get the rest of the reduction sequence for our old \(e_1\). As before, we stitch the reduction step above onto the one (whether divergent, erroring, stuck, or reduced to a value) we get from the IH.

**immutable?, equal?, and chaperone-of? cases**

Follows from the lemmas about immutable, equal, and chaperone-of on approximations above.

\[
\langle E_2[(\text{immutable? } v_4)], \#f, s_2 \rangle \rightarrow \langle E_2[\text{immutable}[[s_2, v_4]]], \#f, s_2 \rangle
\]

\(e_1 = E_1[(\text{immutable? } v_3)],\) so we can take a step
\[
\langle E_1[(\text{immutable? } v_3)], \#f, s_1 \rangle \rightarrow \langle E_1[\text{immutable}[[s_1, v_3]]], \#f, s_1 \rangle.
\]
From approximation, we get \(<v_3, s_1> \sim <v_4, s_2>\), and from lemma 5, we get that \(\text{immutable}[[s_1, v_3]] = \text{immutable}[[s_2, v_4]]\), so
\[
\langle E_1[\text{immutable}[[s_1, v_3]]], s_1 \rangle \sim \langle E_2[\text{immutable}[[s_2, v_4]]], s_2 \rangle.
\]
Applying the IH to the rest of the reduction sequence, using \(E_1[(\text{immutable? } v_3)]\) and \(s_1\) as our new \(e_1\) and \(s_1\) and the approximation above to discharge the hypothesis, we get the rest of the reduction sequence for our old \(e_1\). As before, we stitch the reduction step above onto the one (whether divergent, erroring, stuck, or reduced to a value) we get from the IH.

**Applications of chaperone-of? and equal?**

Follows similarly using the appropriate lemma.

\[
\langle E_2[(\text{vector-ref } (\text{loc } z) n)], \#f, s_2 \rangle \rightarrow \\
\langle E_2[(m_2 l_2 n (\text{vector-ref } l_2 n))], \#f, s_2 \rangle
\]

where \(s_2(z) = (\text{impersonate-vector } l_2 m_2 o_2)\)

\(e_1 = E_1[(\text{vector-ref } (\text{loc } w) n)],\) but based on the approximation from hypothesis II, there are two possibilities for \(s_1(w)\):

\(s_1(w) = (\text{impersonate-vector } l_1 m_1 o_1)\)

Then we get the following reduction step:
\[
\langle E_1[(\text{vector-ref } (\text{loc } w) n)], \#f, s_1 \rangle \rightarrow \\
\langle E_1[(m_1 l_1 n (\text{vector-ref } l_1 n))], \#f, s_1 \rangle
\]

and \(\langle E_1[(m_1 l_1 n (\text{vector-ref } l_1 n))], s_1 \rangle \sim \\
\langle E_2[(m_2 l_2 n (\text{vector-ref } l_2 n))], \#f, s_2 \rangle\).

Applying the IH to the rest of the reduction sequence,
using $E_1[(m_1 l_1 n \text{ vector-ref } l_1 n)]$ and $s_1$ as our new $e_1$ and $s_1$
and the approximation above to discharge the hypothesis, we get the rest
of the reduction sequence for our old $e_1$. As before, we stitch the
reduction step above onto the one (whether divergent, erroring, stuck, or
reduced to a value) we get from the IH.

$s_1(w) = (\text{chaperone-vector } l_1 m_1 o_1)$
Then we get the following reduction step:
$<E_1[(\text{vector-ref } (\text{loc } w) n)], \#f, s_1> \rightarrow$
$<E_1[(\text{let } ([\text{old } (\text{vector-ref } l_1 n)])$
$(\text{let } ([\text{new } (\text{set-marker } (m_1 l_1 n \text{ old}))))$
$(\text{clear-marker } (\text{if } (\text{chaperone-of? } \text{new old})$
$\text{new}$
$(\text{error 'bad-cvref))))]),$
$\#f, s_1> \rightarrow$
Due to the approximation relation, we know that
$<l_1, s_1> \sim <(\text{loc } z), s_1>$ (since we skip through chaperones).
So what we will do is use the entire reduction for $e_6$, but use
$E_1[(\text{vector-ref } l_1 n)]$ as $e_1$ (and keep $s_1$ the same), which removes
the calculation of a chaperone. From our IH, we get that
$<E_1[(\text{vector-ref } l_1 n)], \#f, s_1>$ either:

* Diverges: then the reduction of $e_1$ diverges
* Errors: then the reduction of $e_1$ errors
* Reaches a stuck state: then the reduction of $e_1$ reaches a stuck state.
* Reduces to $<E_1[v_1], \#f, s_3'>$ for some $v_1'$ and $s_3'$
  where $<v_1', s_3'> \sim <v_2, s_4>$.

Then in reducing $e_1$, we get the same steps in the RHS of the first let:

$<E_1[(\text{vector-ref } (\text{loc } w) n)], \#f, s_1> \rightarrow$
$<E_1[(\text{let } ([\text{old } v_1'])$
$(\text{let } ([\text{new } (\text{set-marker } (m_1 l_1 n \text{ old}))))$
$(\text{clear-marker } (\text{if } (\text{chaperone-of? } \text{new old})$
$\text{new}$
$(\text{error 'bad-cvref))))]),$
$\#f, s_3'> \rightarrow$
$<E_1[(\text{let } ([\text{new } (\text{set-marker } (m_1 l_1 n v_1')))]
$(\text{clear-marker } (\text{if } (\text{chaperone-of? } \text{new v_1'})$
$\text{new}$
$(\text{error 'bad-cvref))))]),$
$\#f, s_3'> \rightarrow$
$<E_1[(\text{let } ([\text{new } (m_1 l_1 n v_1')])$
$(\text{clear-marker } (\text{if } (\text{chaperone-of? } \text{new v_1'})$
$\text{new}$
$(\text{error 'bad-cvref))))]),$
$\#t, s_3'>$

For reducing $(m_1 l_1 n v_1')$ in this context, there are several cases:
* $<E_1[(\text{let } ([\text{new } (m_1 l_1 n v_1')])$
(clear-marker (if (chaperone-of? new v_1')
   new
   (error 'bad-cvref)))]),
#t, s_3'> diverges: then reducing e_1 diverges

* <E_1[(let ([new (m_1 l_1 n v_1')])
   (clear-marker (if (chaperone-of? new v_1')
   new
   (error 'bad-cvref)))]),
#t, s_3'> errors: then reducing e_1 errors

* <E_1[(let ([new (m_1 l_1 n v_1')])
   (clear-marker (if (chaperone-of? new v_1')
   new
   (error 'bad-cvref)))]),
#t, s_3'> reaches a stuck state:
then reducing e_1 reaches a stuck state

* <E_1[(let ([new (m_1 l_1 n v_1')])
   (clear-marker (if (chaperone-of? new v_1')
   new
   (error 'bad-cvref)))]),
#t, s_3'> ->*
  <E_1[(let ([new v_1''])
   (clear-marker (if (chaperone-of? new v_1')
   v_1''
   (error 'bad-cvref)))]),
#t, s_3'''> ->
  <E_1[(if (chaperone-of? v_1'' v_1') v_1'' (error 'bad-cvref)))]),
#f, s_3''>

Now there are two cases: v_1'' is not a chaperone of v_1' or it is.

* Not a chaperone: then the reduction of e_1 errors.

  * Is a chaperone. Then we have chaperone-of[[s_3'', v_1'', v_1']] and
  <E_1[(if (chaperone-of? v_1'' v_1') v_1'' (error 'bad-cvref)))]),
  #f, s_3'''> -*
    <E_1[v_1''], #f, s_3'']>

s_3'' <= s_3', and because of the restrictions on the reduction of
e_1, s_3'' <~ s_3'. Therefore, <v_1', s_3'''> <~ <v_2, s_2> by lemma 8
and by lemma 7, <v_1'', s_3'''> <~ <v_2, s_2>. Therefore v_1' is the
v_1 we need, and s_3'' is the s_3 we need to
finish this case.
\[
\langle E_2[(\text{vector-set!} (\text{loc} \, z) \, n \, v_4)], \, \#f, \, s_2 \rangle -> \\
\langle E_2[(\text{vector-set!} \, l_2 \, n \, (o_2 \, l_2 \, n \, v_4))], \, \#f, \, s_2 \rangle
\]

where \( s_2(z) = (\text{impersonate-vector} \, l_2 \, m_2 \, o_2) \)

\( e_1 = E_1[(\text{vector-set!} \, (\text{loc} \, w) \, n \, v_3)] \), but based on the approximation from hypothesis II, there are two possibilities for \( s_1(w) \):

\( s_1(w) = (\text{impersonate-vector} \, l_1 \, m_1 \, o_1) \)

Then we get the following reduction step:

\[
\langle E_1[(\text{vector-set!} \, (\text{loc} \, w) \, n)], \, \#f, \, s_1 \rangle -> \\
\langle E_1[(\text{vector-set!} \, l_1 \, n \, (o_1 \, l_1 \, n \, v_3))], \, \#f, \, s_1 \rangle
\]

and \( \langle E_1[(\text{vector-set!} \, l_1 \, n \, (o_1 \, l_1 \, n \, v_3))], \, \#f, \, s_1 \rangle \sim \\
\langle E_2[(\text{vector-set!} \, l_2 \, n \, (o_2 \, l_2 \, n \, v_4))], \, \#f, \, s_2 \rangle \)

Applying the IH to the rest of the reduction sequence, using \( E_1[(\text{vector-set!} \, l_1 \, n \, (o_1 \, l_1 \, n \, v_3))] \) and \( s_1 \) as our new \( e_1 \) and \( s_1 \) and the approximation above to discharge the hypothesis, we get the rest of the reduction sequence for our old \( e_1 \). As before, we stitch the reduction step above onto the one (whether divergent, erroring, stuck, or reduced to a value) we get from the IH.

\( s_1(w) = (\text{chaperone-vector} \, l_1 \, m_1 \, o_1) \)

Then we get the following reduction step:

\[
\langle E_1[(\text{vector-set!} \, (\text{loc} \, w) \, n \, v_3)], \, \#f, \, s_1 \rangle -> \\
\langle E_1[(\text{let} \, ([new \, (\text{set-marker} \, (o_1 \, l_1 \, n \, v_3))])), \\
(\text{clear-marker} \, (\text{if} \, (\text{chaperone-of?} \, \text{new} \, v_3) \, \text{(vector-set!} \, l_1 \, n \, \text{new}) \, \text{(error 'bad-cvref))})),], \, \#f, \, s_1 \rangle -> \\
\langle E_1[(\text{let} \, ([new \, (o_1 \, l_1 \, n \, v_3))])), \\
(\text{clear-marker} \, (\text{if} \, (\text{chaperone-of?} \, \text{new} \, v_3) \, \text{(vector-set!} \, l_1 \, n \, \text{new}) \, \text{(error 'bad-cvref))})),], \, \#t, \, s_1 \rangle
\]

Either \( (o_1 \, l_1 \, n \, v_3) \) reduces to a value or it doesn’t (diverges, errors, gets stuck). If the latter, then the same is true for the reduction of \( e_1 \). Otherwise, the program state above reduces to

\[
\langle E_1[(\text{let} \, ([\text{new} \, v_3'])]), \\
(\text{clear-marker} \, (\text{if} \, (\text{chaperone-of?} \, \text{new} \, v_3') \, \text{(vector-set!} \, l_1 \, n \, \text{new}) \, \text{(error 'bad-cvref))})),], \, \#t, \, s_3' \rangle -> \\
\langle E_1[(\text{clear-marker} \, (\text{if} \, (\text{chaperone-of?} \, v_1' \, v_3) \, \text{(vector-set!} \, l_1 \, n \, v_3') \, \text{(error 'bad-cvref))})),], \, \#t, \, s_3' \rangle -> \\
\langle E_1[(\text{if} \, (\text{chaperone-of?} \, v_3' \, v_3) \, \text{(vector-set!} \, l_1 \, n \, v_3') \, \text{(error 'bad-cvref))})),], \, \#f, \, s_3' \rangle
\]

if \( v_3' \) is not a chaperone of \( v_3 \) in \( s_3' \), then we get an error.
Otherwise the above reduces to

\[ <E_1[(vector-set! l_1 n v_3')], #f, s_3'> \]

We have that \( \text{chaperone}_of[[s_3', v_3', v_3]] \) and \( s_3' \sim s_1 \) (since
no inappropriate mutating states are allowed), and the latter via
lemma 8 gives us \( <v_3, s_3'> \sim <v_4, s_2'> \). Using lemma 7, that means
\( <v_3', s_3'> \sim <v_4, s_2'> \). Since \( s_3' \sim s_1 \), we also have that
\( <(\text{loc } w), s_1> \sim <(\text{loc } z), s_2> \) gives us \( <(\text{loc } w), s_3'> \sim <(\text{loc } z), s_2> \)
via lemma 7. Since \( \text{loc } w \) points to a chaperone around
\( l_1 \), we also have \( <l_1, s_3'> \sim <(\text{loc } z), s_2> \), which means that
\( <E_1[(vector-set! l_1 n v_3')], #f, s_3'> \sim <E_2[(vector-set! (\text{loc } z) n v_4)], #f, s_2> \)

Thus, we use the IH on the reduction sequence of \( e_2 \), the location
corresponding to the chaperoned value (thus removing a single chaperone),
and this approximation to get the rest of the reduction sequence for
\( e_1 \), to which we prepend the above steps.

\[ <E_2[(\text{vector-ref } (\text{loc } z) n)], #f, s_2> \rightarrow\]
\[ <E_2[(\text{vector-ref } 1_2 n) (\text{vector-ref } 1_2 n)], #f, s_2> \]
where \( s_2(z) = (\text{impersonate-vector } l_2 m_2 o_2) \)

\( e_1 = E_1[(\text{vector-ref } (\text{loc } w) n)] \), but based on the approximation from
hypothesis II, there are two possibilities for \( s_1(w) \):

\( s_1(w) = (\text{impersonate-vector } l_1 m_1 o_1) \)

Then we get the following reduction step:
\[ <E_1[(\text{vector-ref } (\text{loc } w) n)], #f, s_1> \rightarrow\]
\[ <E_1[(\text{vector-ref } (\text{loc } w) n)], #f, s_1> \]
and \( <E_1[(\text{vector-ref } (\text{loc } w) n)], #f, s_1> \sim \)
\[ <E_2[(\text{vector-ref } 1_2 n)], #f, s_2> \]

Applying the IH to the rest of the reduction sequence,
using \( E_1[(\text{vector-ref } 1_1 n)] \) and \( s_1 \) as our new \( e_1 \) and \( s_1 \)
and the approximation above to discharge the hypothesis, we get the rest
of the reduction sequence for our old \( e_1 \). As before, we stitch the
reduction step above onto the one (whether divergent, erroring, stuck, or
reduced to a value) we get from the IH.

\( s_1(w) = (\text{chaperone-vector } l_1 m_1 o_1) \)

Then we get the following reduction step:
\[ <E_1[(\text{vector-ref } (\text{loc } w) n)], #f, s_1> \rightarrow\]
\[ <E_1[(\text{vector-ref } (\text{loc } w) n)], #f, s_1> \]

\( \text{Due to the approximation relation, we know that} \)
\( <l_1, s_1> \sim <(\text{loc } z), s_1> \) (since we skip through chaperones).
So what we will do is use the entire reduction for \( e_6 \), but use
\( E_1[(\text{vector-ref } l_1 n)] \) as \( e_1 \) (and keep \( s_1 \) the same), which removes
the calculation of a chaperone. From our IH, we get that
(<E_1[(vector-ref l_1 n)], #f, s_1> either:

* Diverges: then the reduction of e_1 diverges
* Errors: then the reduction of e_1 errors
* Reaches a stuck state: then the reduction of e_1 reaches a stuck state.
* Reduces to <E_1[v_1], #f, s_3'> for some v_1' and s_3'
  where <v_1', s_3'> ~ <v_2, s_4>.

Then in reducing e_1, we get the same steps in the RHS of the first let:

<E_1[(vector-ref (loc w) n)], #f, s_1> ->*
  <E_1[(let ([old v_1'])
   (let ([new (set-marker (m_1 l_1 n old))])
     (clear-marker (if (chaperone-of? new old)
                       new
                       (error 'bad-cvref)))))]
  #f, s_3'> ->
  <E_1[(let ([new (set-marker (m_1 l_1 n v_1'))])
       (clear-marker (if (chaperone-of? new v_1')
                       new
                       (error 'bad-cvref)))))]
  #f, s_3'> ->
  <E_1[(let ([new (m_1 l_1 n v_1')]
             (clear-marker (if (chaperone-of? new v_1')
                            new
                            (error 'bad-cvref)))))]
  #t, s_3'>

For reducing (m_1 l_1 n v_1') in this context, there are several cases:
* <E_1[(let ([new (m_1 l_1 n v_1')])
       (clear-marker (if (chaperone-of? new v_1')
                       new
                       (error 'bad-cvref)))))]
  #t, s_3'> diverges: then reducing e_1 diverges

* <E_1[(let ([new (m_1 l_1 n v_1')])
       (clear-marker (if (chaperone-of? new v_1')
                       new
                       (error 'bad-cvref)))))]
  #t, s_3'> errors: then reducing e_1 errors

* <E_1[(let ([new (m_1 l_1 n v_1')])
       (clear-marker (if (chaperone-of? new v_1')
                       new
                       (error 'bad-cvref)))))]
  #t, s_3'> reaches a stuck state: then reducing e_1 reaches a stuck state

* <E_1[(let ([new (m_1 l_1 n v_1')])
       (clear-marker (if (chaperone-of? new v_1')
                       new
                       (error 'bad-cvref)))))]
  #t, s_3'>
Now there are two cases: \( v_1'' \) is not a chaperone of \( v_1' \) or it is.

* Not a chaperone: then the reduction of \( e_1 \) errors.

* Is a chaperone. Then we have chaperone-of\([s_3'', v_1''', v_1']\], and

\[
\langle E_1[(\text{if (chaperone-of? } v_1'' v_1') v_1''', \text{error 'bad-cvref})]),
\#f, s_3''\rangle \rightarrow^* \langle E_1[v_1''], \#f, s_3''\rangle.
\]

\( s_3'' \leq s_3' \), and because of the restrictions on the reduction of \( e_1, s_3'' \sim s_3' \). Therefore, \( <v_1', s_3''> \sim <v_2, s_2> \) by lemma 8 and by lemma 7, \( <v_1'', s_3''> \sim <v_2, s_2> \).

Therefore \( v_1'' \) is the \( v_1 \) we need, and \( s_3'' \) is the \( s_3 \) we need to finish this case.

\[
\langle E_2[(\text{vector-set! (loc z) n v_4})], \#f, s_2\rangle \rightarrow
\langle E_2[(\text{vector-set! l_2 n (o_2 l_2 n v_4})]], \#f, s_2\rangle
\]

where \( s_2(z) = (\text{impersonate-vector l_2 m_2 o_2}) \)

\( e_1 = E_1[(\text{vector-set! (loc w) n v_3})] \), but based on the approximation from hypothesis II, there are two possibilities for \( s_1(w) \):

\( s_1(w) = (\text{impersonate-vector l_1 m_1 o_1}) \)

Then we get the following reduction step:

\[
\langle E_1[(\text{vector-set! (loc w) n})], \#f, s_1\rangle \rightarrow
\langle E_1[(\text{vector-set! l_1 n (o_1 l_1 n v_3)})], \#f, s_1\rangle
\]

and \( E_1[(\text{vector-set! l_1 n (o_1 l_1 n v_3)})], \#f, s_1\rangle \sim
\langle E_2[(\text{vector-set! l_2 n (o_2 l_2 n v_4)})], \#f, s_2\rangle
\]

Applying the IH to the rest of the reduction sequence, using \( E_1[(\text{vector-set! l_1 n (o_1 l_1 n v_3)})] \) and \( s_1 \) as our new \( e_1 \) and \( s_1 \) and the approximation above to discharge the hypothesis, we get the rest of the reduction sequence for our old \( e_1 \). As before, we stitch the reduction step above onto the one (whether divergent, erroring, stuck, or reduced to a value) we get from the IH.

\( s_1(w) = (\text{chaperone-vector l_1 m_1 o_1}) \)

Then we get the following reduction step:
Either \((o_1 \ l_1 \ n \ v_3)\) reduces to a value or it doesn’t (diverges, errors, gets stuck). If the latter, then the same is true for the reduction of \(e_1\). Otherwise, the program state above reduces to

\[
\langle E_1[(vector-set! (loc w) n v_3)], \#f, s_1\rangle \rightarrow \\
\langle E_1[(let ([new (set-marker (o_1 \ l_1 \ n \ v_3))]) \\
(clear-marker (if (chaperone-of? new v_3) \\
(vector-set! \ l_1 \ n \ new) \\
(error 'bad-cvref)))]), \\
\#f, s_1\rangle \rightarrow \\
\langle E_1[(let ([new (o_1 \ l_1 \ n \ v_3)]) \\
(clear-marker (if (chaperone-of? new v_3) \\
(vector-set! \ l_1 \ n \ new) \\
(error 'bad-cvref)))]), \\
\#t, s_1\rangle \\
\]

If \(v_3'\) is not a chaperone of \(v_3\) in \(s_3'\), then we get an error. Otherwise the above reduces to

\[
\langle E_1[(let ([new v_3'])] \\
(clear-marker (if (chaperone-of? new v_3) \\
(vector-set! \ l_1 \ n \ new) \\
(error 'bad-cvref)))]), \\
\#t, s_3'\rangle \rightarrow \\
\langle E_1[(if (chaperone-of? v_3' v_3) \\
(vector-set! \ l_1 \ n \ v_3') \\
(error 'bad-cvref)))]), \\
\#f, s_3'\rangle \\
\]

Thus, we use the IH on the reduction sequence of \(e_2\), the location corresponding to the chaperoned value (thus removing a single chaperone), and this approximation to get the rest of the reduction sequence for \(e_1\), to which we prepend the above steps.

\[
\langle E_2[(vector-ref (loc z) n)], \#f, s_2\rangle \rightarrow \\
\langle E_2[v_4n], \#f, s_2\rangle \\
\]

where \(s_2(z) = (vector \#f \ v_{40} \ ... \ v_{4n} \ ... \ v_{4k})\)

(and \(0 \leq n \leq k\), since \(e_6\) reduces to a value in the context \(E_2\))
e_1 = E_1[(vector-ref (loc w) n)], but based on the approximation from hypothesis II, there are two possibilities for s_1(w):

s_1(w) = (vector #f v_30 ... v_3n ... v_3k)
Then we get the following reduction step:
<E_1[(vector-ref (loc w) n)], #f, s_1> -> <E_1[v_3n], #f, s_1>
and <E_1[v_3n], s_1> ~ <E_2[v_4n], #f, s_2>, since the vectors were already approximate in s_1/s_2. Thus, e_5 reduces to a value (namely, v_3n).

s_1(w) = (chaperone-vector l_1 m_1 o_1)
Then we get the following reduction step:
<E_1[(vector-ref (loc w) n)], #f, s_1> ->
<E_1[(let ([old (vector-ref l_1 n)])
  (let ([new (set-marker (m_1 l_1 n old))])
   (clear-marker (if (chaperone-of? new old)
     new
     (error 'bad-cvref))))),
   #f, s_1]

Due to the approximation relation, we know that
<l_1, s_1> ~ <(loc z), s_1> (since we skip through chaperones).
So what we will do is use the entire reduction for e_6, but use
E_1[(vector-ref l_1 n)] as e_1 (and keep s_1 the same), which removes the calculation of a chaperone. From our IH, we get that
<E_1[(vector-ref l_1 n)], #f, s_1> either:

* Diverges: then the reduction of e_1 diverges
* Errors: then the reduction of e_1 errors
* Reaches a stuck state: then the reduction of e_1 reaches a stuck state.
* Reduces to <E_1[v_1], #f, s_3'> for some v_1' and s_3'
  where <v_1', s_3'> ~ <v_2, s_4>.

Then in reducing e_1, we get the same steps in the RHS of the first let:

<E_1[(vector-ref (loc w) n)], #f, s_1> ->
<E_1[(let ([old v_1'])
  (let ([new (set-marker (m_1 l_1 n old))])
   (clear-marker (if (chaperone-of? new old)
     new
     (error 'bad-cvref))))),
   #f, s_3'] ->
<E_1[(let ([new (set-marker (m_1 l_1 n v_1'))])
  (clear-marker (if (chaperone-of? new v_1')
     new
     (error 'bad-cvref))))],
   #f, s_3'] ->
<E_1[(let ([new (m_1 l_1 n v_1')])
  (clear-marker (if (chaperone-of? new v_1')
     new
     (error 'bad-cvref))))],
   #f, s_3']
For reducing \((m_1 \, l_1 \, n \, v_1')\) in this context, there are several cases:

* \(<E_1[(let ([new (m_1 \, l_1 \, n \, v_1')])
\quad (clear-marker (if (chaperone-of? new v_1')
\quad new
\quad (error 'bad-cvref)))]),
#t, s_3'>\) diverges: then reducing \(e_1\) diverges

* \(<E_1[(let ([new (m_1 \, l_1 \, n \, v_1')])
\quad (clear-marker (if (chaperone-of? new v_1')
\quad new
\quad (error 'bad-cvref)))]),
#t, s_3'>\) errors: then reducing \(e_1\) errors

* \(<E_1[(let ([new (m_1 \, l_1 \, n \, v_1')])
\quad (clear-marker (if (chaperone-of? new v_1')
\quad new
\quad (error 'bad-cvref)))]),
#t, s_3'>\) reaches a stuck state: then reducing \(e_1\) reaches a stuck state

* \(<E_1[(let ([new (m_1 \, l_1 \, n \, v_1')])
\quad (clear-marker (if (chaperone-of? new v_1')
\quad new
\quad (error 'bad-cvref)))]),
#t, s_3'>\) \(-\star\)
\(<E_1[(let ([new v_1''])]
\quad (clear-marker (if (chaperone-of? new v_1')
\quad new
\quad (error 'bad-cvref)))]),
#t, s_3'''>\) \(-\star\)
\(<E_1[(if (chaperone-of? v_1'' v_1') v_1'' (error 'bad-cvref))] ),
#f, s_3'''>

Now there are two cases: \(v_1''\) is not a chaperone of \(v_1'\) or it is.

* Not a chaperone: then the reduction of \(e_1\) errors.

* Is a chaperone. Then we have chaperone-of[[s_3'', v_1'', v_1']] and
\(<E_1[(if (chaperone-of? v_1'' v_1') v_1'' (error 'bad-cvref))] ),
#f, s_3'''> \(-\star\)
\(<E_1[v_1''], #f, s_3'''>.

\(s_3'' \leq s_3'\), and because of the restrictions on the reduction of \(e_1, s_3'' \sim s_3'. \) Therefore, \(<v_1', s_3'''> \sim <v_2, s_2>\) by lemma 8
and by lemma 7, \( <v_1'', s_3''> \sim <v_2, s_2> \).
Therefore \( v_1'' \) is the \( v_1 \) we need, and \( s_3'' \) is the \( s_3 \) we need to finish this case.

(This exactly mirrors the vector-ref of a chaperoned impersonated vector above, for good reason. I'm not going to repeat it for a chaperoned immutable vector.)

\[
\begin{align*}
\langle E_2[\text{vector-set!} (\text{loc } z) n v_4], \#f, s_2\rangle & \rightarrow \\
\langle E_2[\text{void}], \#f, s_2[z |-> (\text{vector } \#f v_{40} \ldots v_{4n} \ldots v_{4k})]\rangle
\end{align*}
\]

where \( s_2(z) = (\text{vector } \#f v_{40} \ldots v_{4n} \ldots v_{4k}) \)

\( e_1 = E_1[\text{vector-set!} (\text{loc } w) n v_3] \), but based on the approximation from hypothesis II, there are two possibilities for \( s_1(w) \):

\( s_1(w) = (\text{vector } \#f v_{30} \ldots v_{3n} \ldots v_{3k}) \)

Then we get the following reduction step:

\[
\begin{align*}
\langle E_1[\text{vector-set!} (\text{loc } w) n], \#f, s_1\rangle & \rightarrow \\
\langle E_1[\text{void}], \#f, s_1[w |-> (\text{vector } \#f v_{30} \ldots v_{3n} \ldots v_{3k})]\rangle
\end{align*}
\]

and \( \langle E_1[\text{void}], \#f, s_1[w |-> (\text{vector } \#f v_{30} \ldots v_{3n} \ldots v_{3k})]\rangle \sim \\
\langle E_2[\text{void}], \#f, s_2[z |-> (\text{vector } \#f v_{40} \ldots v_{4n} \ldots v_{4k})]\rangle
\]

(since the only change in the store is replacing the corresponding element in two approximated vectors with approximate values).

(\text{void}) is a value, so \( e_5 \) evaluates to a value (\text{void}) and the resulting expression/store is appropriately approximate to the result of reducing \( e_6 \).

\( s_1(w) = (\text{chaperone-vector } l_1 m_1 o_1) \)

Then we get the following reduction step:

\[
\begin{align*}
\langle E_1[\text{vector-set!} (\text{loc } w) n v_3], \#f, s_1\rangle & \rightarrow \\
\langle E_1[\text{let} ([\text{new } (\text{set-marker} (o_1 l_1 n v_3)))]
\quad \text{(clear-marker} \text{ if}(\text{chaperone-of? } \text{new } v_3))
\quad \text{(vector-set!} l_1 n \text{\ new})
\quad \text{(error 'bad-cvref}))]]],
\#f, s_1\rangle & \rightarrow \\
\langle E_1[\text{let} ([\text{new } o_1 l_1 n v_3])
\quad \text{(clear-marker} \text{ if}(\text{chaperone-of? } \text{new } v_3))
\quad \text{(vector-set!} l_1 n \text{\ new})
\quad \text{(error 'bad-cvref}))]]],
\#t, s_1\rangle
\end{align*}
\]

Either \( o_1 l_1 n v_3 \) reduces to a value or it doesn’t (diverges, errors, gets stuck). If the latter, then the same is true for the reduction of \( e_1 \). Otherwise, the program state above reduces to

\[
\begin{align*}
\langle E_1[\text{let} ([\text{new } v_3'])
\quad \text{(clear-marker} \text{ if}(\text{chaperone-of? } \text{new } v_3))
\quad \text{(vector-set!} l_1 n \text{\ new})
\quad \text{(error 'bad-cvref}))]]],
\#t, s_3'\rangle & \rightarrow \\
\langle E_1[\text{clear-marker} \text{ if}(\text{chaperone-of? } v_1' v_3))
\quad \text{(vector-set!} l_1 n v_3')
\end{align*}
\]
(error 'bad-cvref)))]},

#t, s_3' ->
<E_1[(if (chaperone-of? v_3' v_3)
  (vector-set! l_1 n v_3')
  (error 'bad-cvref)))]},

#f, s_3'

if v_3' is not a chaperone of v_3 in s_3', then we get an error. Otherwise the above reduces to

<E_1[(vector-set! l_1 n v_3')], #f, s_3'>

We have that chaperone_of[[s_3', v_3', v_3]] and s_3' \sim s_1 (since no inappropriate mutating states are allowed), and the latter via lemma 8 gives us <v_3, s_3'> \sim <v_4, s_2>. Using lemma 7, that means <v_3', s_3'> \sim <v_4, s_2>. Since s_3' \sim s_1, we also have that <(loc w), s_1> \sim <(loc z), s_2> gives us <(loc w), s_3'> \sim <(loc z), s_2> via lemma 7. Since (loc w) points to a chaperone around l_1, we also have <l_1, s_3'> \sim <(loc z), s_2>, which means that

<E_1[(vector-set! l_1 n v_3')], #f, s_3'> ~
<E_2[(vector-set! (loc z) n v_4)], #f, s_2'>

Thus, we use the IH on the reduction sequence of e_2, the location corresponding to the chaperoned value (thus removing a single chaperone), and this approximation to get the rest of the reduction sequence for e_1, to which we prepend the above steps.

(Again, mirrors the proof of vector-set! on a chaperoned impersonated vector.)

<E_2[(vector-ref (loc z) n), #f, s_2] ->
<E_2[v_4n], #f, s_2>

where s_2(z) = (vector-immutable v_40 ... v_4n ... v_4k)
(and 0 <= n <= k, since e_6 reduces to a value in the context E_2)

e_1 = E_1[(vector-ref (loc w) n)], but based on the approximation from hypothesis II, there are two possibilities for s_1(w):

s_1(w) = (vector-immutable v_30 ... v_3n ... v_3k)
Then we get the following reduction step:
<E_1[(vector-ref (loc w) n)], #f, s_1] -> <E_1[v_3n], #f, s_1>
and <E_1[v_3n], s_1> \sim <E_2[v_4n], #f, s_2>, since the vectors were already approximates in s_1/s_2. Thus, e_5 reduces to a value (namely, v_3n).

s_1(w) = (chaperone-vector l_1 m_1 o_1)

As before, the proof follows exactly the format of earlier vector-refs on chaperoned values, so I'm not repeating it a third time.

Lemma 10:

For all e_2 that do not contain set-marker, get-marker, or chaperone-vector and s_2,
Let E_2[e_6] = e_2.
If there exists no \( v_2 \) or \( s_4 \) such that
\[ <E_2[e_6], #f, s_2> \] reduces to \[ <E_2[v_2], #f, s_4>, \]
For all \( e_1 \) and \( s_1 \) such that \( <e_1, s_1> \sim <e_2, s_2> \), let \( E_1[e_5] = e_1 \).
Also, require that the reduction of \( <e_1, #f, s_1> \) contains no program states of the form \( E[(vector-set! (loc x) n v), #t, s> \) where \( s(x) = (vector #f v_e ...). \)

Either:
1) \( <e_1, #f, s_1> \) diverges
2) there exists a \( b, s_3 \).
   \( <e_1, #f, s_1> \) reduces to \( (error 'variable), b, s_3> \)
3) there exists an \( e_3, b, s_3 \).
   \( <e_1, #f, s_1> \) reduces to \( <e_3, b, s_3> \) and \( e_3 \) is a stuck state.

(That is, if the erased program does not reduce the current redex to a value, then the unerased program cannot.)

Proof:

If there's no initial reduction step for \( <e_2, #f, s_2> \), then we have a stuck state, and \( <e_1, #f, s_1> \) will also be a stuck state.
If there is an initial reduction step, then the proof follows the same form as Lemma 9. Most of the proof just involves stepping in both reduction sequences than inducting, so those stay pretty much the same (that is, we get the same kind of result as the hypothesis, which is that we DON'T reduce to a value). The main difference in this proof is that in the chaperone cases for \( vector-ref/vector-set!, vector-ref/vector-set! \) on the chaperoned value (the IH) does _not_ reduce to a value. However, that's fine, since that's exactly what we want! So in the vector-ref case, this is immediate, since we first vector-ref the chaperoned value. In the vector-set! case, we might either fail to reduce/diverge/error in the function from the chaperone (which is A-OK), or we fail to reduce/diverge/error from doing vector-set! on the chaperoned value.

------

Restatement of theorem 3:

For all \( e \), if \( e \) is a user-writeable program, \( Eval(e) = v \), and that evaluation contains no reductions where the left-hand side is of the form \( (s #t (vector-set! (loc x) n v_a)) \) where \( s(x) = (vector #f v_v ...), \) then \( Eval(|e|) = v \).

Proof:

Take the reduction sequence for \( <|e|, #f, {}> \). Either it diverges, ends in a stuck state, ends in an error state, or ends in a value.

Keep in mind that each reduction step in the erased program has a corresponding reduction step in the unerased programs. (Chaperones
only add reduction steps to apply the interceding function and check the returned value for chaperone-ness.

Diverges:

Ends in a stuck state:

Ends in an error state:

All these cases force the unerased program to NOT reduce to a value as shown in lemma 10. Therefore these break our initial hypothesis.

Ends in a value:

Let the value state be \(<v_2, #f, s_2>\). By lemma 9 and the fact that we know \(<e_1, #f, {}>\) reduces to \(<v_1, #f, s_1>\) for some state \(s_1\) (since \(\text{Eval}(e) = v\)), then we know that \(<v_1, s_1> \sim <v_2, s_2>\).

Now let's examine the cases of \(v_2\):

- \(v_2\) is a boolean: then \(v_1\) is the same boolean, and \(\text{Eval}(e) = \text{Eval}(|e|)\).
- \(v_2\) is a number: then \(v_1\) is the same number, and \(\text{Eval}(e) = \text{Eval}(|e|)\).
- \(v_2\) is a pointer to a lambda: then \(v_1\) must also be a pointer to a lambda, and \(\text{Eval}(e) = \text{Eval}(|e|) = 'proc'\).
- \(v_2\) is a pointer to a mutable vector, immutable vector, or impersonator: Then \(v_1\) is a pointer to the same, or a pointer to a series of chaperones that ends in the same. That is, \(v_1\) cannot contain a lambda. Since \(\text{Eval}\) only disambiguates locations on whether they contain a lambda or not, and the not case returns ‘vector’, \(\text{Eval}(e) = \text{Eval}(|e|) = 'vector'\).