CS 15-312 Assignment #3
The C Machine, References and Exceptions

Solution

1 Evaluation Rules for the C Machine

- All of the primitive operators (+, -, *, /, =) can be defined as shown in the notes on abstract machines (with M and H unchanged):

$$(M, H, K, +(e_1, e_2)) \mapsto_{\mathsf{C}} (M, H, +(\Box, e_2) \triangleright K, e_1)$$

$$(M, H, +(\Box, e_2) \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, +(v_1, \Box) \triangleright K, e_2)$$

$$(M, H, +(n_1, \Box) \triangleright K, n_2) \mapsto_{\mathsf{C}} (M, H, K, n_1 + n_2)$$

- Rules for conditionals (again, M and H are unchanged):

$$(M, H, K, \texttt{if}\, e\, \texttt{then}\, e_1\, \texttt{else}\, e_2\, \texttt{fi}) \mapsto_{\mathsf{C}} (M, H, \texttt{if}\, \Box\, \texttt{then}\, e_1\, \texttt{else}\, e_2\, \texttt{fi} \triangleright K, e)$$

$$(M, H, \texttt{if}\, \Box\, \texttt{then}\, e_1\, \texttt{else}\, e_2\, \texttt{fi} \triangleright K, \texttt{true}) \mapsto_{\mathsf{C}} (M, H, K, e_1)$$

$$(M, H, \texttt{if}\, \Box\, \texttt{then}\, e_1\, \texttt{else}\, e_2\, \texttt{fi} \triangleright K, \texttt{false}) \mapsto_{\mathsf{C}} (M, H, K, e_2)$$

- Rules for application of functions (we are NOT using an environment; substitution remains our basis for application):

$$(M, H, K, \texttt{apply}(e_1, e_2)) \mapsto_{\mathsf{C}} (M, H, \texttt{apply}(\Box, e_2) \triangleright K, e_1)$$

$$(M, H, \texttt{apply}(\Box, e_2) \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, \texttt{apply}(v_1, \Box) \triangleright K, e_2)$$

$$(M, H, \texttt{apply}(v_1, \Box) \triangleright K, v_2) \mapsto_{\mathsf{C}} (M, H, K, [v_1, v_2/f, x]e)$$

- Rules for pairs:

$$(M, H, K, \texttt{(}e_1\texttt{,}e_2\texttt{)}) \mapsto_{\mathsf{C}} (M, H, \texttt{(}\Box\texttt{,}e_2\texttt{)} \triangleright K, e_1)$$

$$(M, H, \texttt{(}\Box\texttt{,}e_2\texttt{)} \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, \texttt{(}v_1\texttt{,}\Box\texttt{)} \triangleright K, e_2)$$

$$(M, H, \texttt{(}v_1\texttt{,}\Box\texttt{)} \triangleright K, v_2) \mapsto_{\mathsf{C}} (M, H, K, \texttt{(}v_1\texttt{,}v_2\texttt{)})$$

- Rules for `bind`:

$$(M, H, K, \texttt{bind}\, p\, \texttt{to}\, e_1\, \texttt{in}\, e_2\, \texttt{end}) \mapsto_{\mathsf{C}} (M, H, \texttt{bind}\, p\, \texttt{to}\, \square\, \texttt{in}\, e_2\, \texttt{end} \triangleright K, e_1)$$

$$(M, H, \texttt{bind}\,\_\,\texttt{to}\, \square\, \texttt{in}\, e_2\, \texttt{end} \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, K, e_2)$$

$$(M, H, \texttt{bind}\,()\,\texttt{to}\, \square\, \texttt{in}\, e_2\, \texttt{end} \triangleright K, \texttt{unit}) \mapsto_{\mathsf{C}} (M, H, K, e_2)$$

$$(M, H, \texttt{bind}\, x : \tau\, \texttt{to}\, \square\, \texttt{in}\, e_2\, \texttt{end} \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, K, [v_1/x]e_2)$$

$$(M, H, \texttt{bind}\,(p_1\texttt{,}p_2)\,\texttt{to}\, \square\, \texttt{in}\, e_2\, \texttt{end} \triangleright K, (v_1\texttt{,}v_2)) \mapsto_{\mathsf{C}}$$

$$(M, H, \texttt{bind}\, p_1\, \texttt{to}\, \square\, \texttt{in}\, \texttt{bind}\, p_2\, \texttt{to}\, v_2\, \texttt{in}\, e_2\, \texttt{end}\, \texttt{end} \triangleright K, v_1)$$

- Rules for `inl/inr` (these are symmetric, of course):

$$(M, H, K, \texttt{inl}_{\tau_1\texttt{+}\tau_2}\texttt{(}e\texttt{)}) \mapsto_{\mathsf{C}} (M, H, \texttt{inl}_{\tau_1\texttt{+}\tau_2}\texttt{(}\square\texttt{)} \triangleright K, e)$$

$$(M, H, \texttt{inl}_{\tau_1\texttt{+}\tau_2}\texttt{(}\square\texttt{)} \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K, \texttt{inl}_{\tau_1\texttt{+}\tau_2}\texttt{(}v\texttt{)})$$

- Rules for `case`:

$$(M, H, K, \texttt{case}_{\tau_1\texttt{+}\tau_2}\, e\, \texttt{of}\, \texttt{inl(}x_1\texttt{:}\tau_1\texttt{+}\tau_2\texttt{)} \texttt{=>} e_1 \,\texttt{|}\, \texttt{inr(}x_2\texttt{:}\tau_1\texttt{+}\tau_2\texttt{)} \texttt{=>} e_2\, \texttt{end}) \mapsto_{\mathsf{C}}$$

$$(M, H, \texttt{case}_{\tau_1\texttt{+}\tau_2}\, \square\, \texttt{of}\, \texttt{inl(}x_1\texttt{:}\tau_1\texttt{+}\tau_2\texttt{)} \texttt{=>} e_1 \,\texttt{|}\, \texttt{inr(}x_2\texttt{:}\tau_1\texttt{+}\tau_2\texttt{)} \texttt{=>} e_2\, \texttt{end} \triangleright K, e)$$

$$(M, H, \texttt{case}_{\tau_1\texttt{+}\tau_2}\, \square\, \texttt{of}\, \texttt{inl(}x_1\texttt{:}\tau_1\texttt{+}\tau_2\texttt{)} \texttt{=>} e_1 \,\texttt{|}\, \texttt{inr(}x_2\texttt{:}\tau_1\texttt{+}\tau_2\texttt{)} \texttt{=>} e_2\, \texttt{end} \triangleright K, \texttt{inl}_{\tau_1\texttt{+}\tau_2}\texttt{(}v\texttt{)}) \mapsto_{\mathsf{C}}$$
$$(M, H, K, [v/x_1]e_1)$$

$$(M, H, \texttt{case}_{\tau_1\texttt{+}\tau_2}\, \square\, \texttt{of}\, \texttt{inl(}x_1\texttt{:}\tau_1\texttt{+}\tau_2\texttt{)} \texttt{=>} e_1 \,\texttt{|}\, \texttt{inr(}x_2\texttt{:}\tau_1\texttt{+}\tau_2\texttt{)} \texttt{=>} e_2\, \texttt{end} \triangleright K, \texttt{inr}_{\tau_1\texttt{+}\tau_2}\texttt{(}v\texttt{)}) \mapsto_{\mathsf{C}}$$
$$(M, H, K, [v/x_2]e_2)$$

- Rules for `ref`:

$$(M, H, K, \texttt{ref(}e\texttt{)}) \mapsto_{\mathsf{C}} (M, H, \texttt{ref(}\square\texttt{)} \triangleright K, e)$$

$$(M, H, \texttt{ref(}\square\texttt{)} \triangleright K, v) \mapsto_{\mathsf{C}} (M[\ell = v], H, K, l)$$

Note: ℓ here is a *new* location not in the domain of M. In practice, we have a typing store in the memory as well so that printing the type of a location in `loop` is easy. See the sample solution.

- Rules for `!`:

$$(M, H, K, \texttt{!}\,e) \mapsto_{\mathsf{C}} (M, H, \texttt{!}\square \triangleright K, e)$$

$$(M, H, \texttt{!}\square \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K, M(v))$$

- Rules for assignment:

$$(M, H, K, e_1 \texttt{:=} e_2) \mapsto_{\mathsf{C}} (M, H, \square \texttt{:=} e_2 \triangleright K, e_1)$$

$$(M, H, \square \texttt{:=} e_2 \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, v_1 \texttt{:=} \square \triangleright K, e_2)$$
$$(M, H, v_1 \texttt{:=} \square \triangleright K, v_2) \mapsto_{\mathsf{C}} (M[v_1 = v_2], H, K, v_2)$$

- Rules for `try/ow`:

$$(M, H, K, \texttt{try}\, e_1 \,\texttt{ow}\, e_2) \mapsto_{\mathsf{C}} (M, \texttt{(}K\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\square\,\texttt{ow}\, e_2 \triangleright K, e_1)$$

$$(M, \texttt{(}K'\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\square\,\texttt{ow}\, e_2 \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K, v)$$

  Note that the "try" frame doesn't really need e_2 in it (and doesn't have it in the implementation). It simply marks that we are in the context of a `try`. The handler is storing e_2 for us.

- Rules for failures:

$$(M, \texttt{(}K'\texttt{,}e'\texttt{)} \triangleright H, K, \texttt{fail}) \mapsto_{\mathsf{C}} (M, H, K', e')$$

$$(M, \bullet, K \neq \bullet, \texttt{fail}) \mapsto_{\mathsf{C}} (M, \bullet, \bullet, \texttt{fail})$$

2 Implementation

See the code solution.

3 Proof of Preservation

First we must inductively define well-formed stacks:

$$\frac{}{\vdash \bullet : \tau\,\texttt{stack}} \qquad \frac{\vdash F : (\tau,\tau')\ \texttt{frame} \quad \vdash K : \tau'\,\texttt{stack}}{\vdash F \triangleright K : \tau\,\texttt{stack}}$$

This definition depends on a proper definition of frames. Here is the relevant portion:

$$\frac{\vdash e_2 : \texttt{int}}{\vdash +(\Box, e_2) : (\texttt{int},\texttt{int})\ \texttt{frame}} \qquad \frac{v_1\ \mathsf{value} \quad \vdash v_1 : \texttt{int}}{\vdash +(v_1, \Box) : (\texttt{int},\texttt{int})\ \texttt{frame}}$$

$$\frac{\vdash e_1 : \tau \quad \vdash e_2 : \tau}{\vdash \texttt{if}\,\Box\,\texttt{then}\,e_1\,\texttt{else}\,e_2\,\texttt{fi} : (\texttt{bool},\tau)\ \texttt{frame}}$$

$$\frac{\vdash e_2 : \tau_2}{\vdash \texttt{apply}(\Box, e_2) : (\tau_2 \rightarrow \tau,\tau)\ \texttt{frame}} \qquad \frac{v_1\ \mathsf{value} \quad \vdash v_1 : \tau_2 \rightarrow \tau}{\vdash \texttt{apply}(v_1, \Box) : (\tau_2,\tau)\ \texttt{frame}}$$

$$\frac{}{\vdash \texttt{ref}(\Box) : (\tau,\tau\,\texttt{ref})\ \texttt{frame}} \qquad \frac{}{\vdash \texttt{!}\Box : (\tau\,\texttt{ref},\tau)\ \texttt{frame}}$$

$$\frac{\vdash e_2 : \tau}{\vdash \Box\texttt{:=}e_2 : (\tau\,\texttt{ref},\tau)\ \texttt{frame}} \qquad \frac{v_1\ \mathsf{value}}{\vdash v_1\texttt{:=}\Box : (\tau,\tau)\ \texttt{frame}}$$

$$\frac{\vdash e_2 : \tau}{\vdash \texttt{try}\,\Box\,\texttt{ow}\,e_2 : (\tau,\tau)\ \texttt{frame}}$$

Lemma 1: Stack Inversion
If we have a stack of type τ, consisting of a frame composed with a stack K, we can "run the rules in reverse" as with normal Inversion to get the type of the stack K.

The proof is precisely analogous to the (trivial) proof of normal Inversion.

Lemma 2: Prefix Property
If H prefix K and ($\mathsf{ok}M, H, K, e$) and $(M, H, K, e) \mapsto_{\mathsf{C}} (M', H', K', e')$, then H' prefix K'.

We note, as stated in class, that the handlers in H will always be prefixes of the stack K. We define H prefix K inductively:

$$\frac{}{\bullet\ \mathsf{prefix}\ K} \qquad \frac{\exists n : \texttt{n pops of } K = K' \quad H\ \mathsf{prefix}\ K}{(K', e') \triangleright H\ \mathsf{prefix}\ K}$$

Note that by the evaluation rules, the only time we can push a new frame on to H is when e is a `try/ow` expression. If H prefix K, then, and H remains unchanged and we push something onto K to form K', H prefix K'. If we pop K to obtain K', however, we might lose this property. Notice, however, that when a frame K, e is placed on H, a `try` frame is pushed onto K. When we pop this frame, the associated frame of H will now need 0 pops to reach the K in the handler frame. However, when we pop that frame, we also pop H (and we know inductively that the other frames of H have the prefix-property)! Thus, no pop will ever cause H to lose the prefix-property.

Now we can define what it means for a state to be ok.

$$\frac{\vdash K : \tau\ \mathtt{stack} \qquad H \ \mathsf{prefix}\ K \qquad \vdash M : \Lambda \qquad \Lambda; \bullet \vdash e : \tau}{(M, H, K, e)\ \mathsf{ok}}$$

Theorem (Preservation):
If (M, H, K, e) ok *and* $(M, H, K, e) \mapsto_{\mathsf{C}} (M', H', K', e')$, *then* (M', H', K', e') ok.

Proof:
We will use case analysis on the evaluation rules to show that this holds. We will only consider the cases resulting from additions to the language and from `if/then/else`. *(Note: we are not properly using INDUCTION because there are no "simpler" cases to consider, now that we have eliminated recursive calls and made the control stack explicit.)*

- For the `if` cases:
  - If (push rule):

    $(M, H, K, \mathtt{if}\, e\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2\, \mathtt{fi}) \mapsto_{\mathsf{C}} (M, H, \mathtt{if}\, \square\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2\, \mathtt{fi} \triangleright K, e)$

    We need to show that $(M, H, \mathtt{if}\, \square\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2\, \mathtt{fi} \triangleright K, e)$ ok. M and H remain unchanged. By Inversion, $e : \mathtt{bool}, e_1 : \tau, e_2 : \tau$. Using this and the definition of frames, we see that $\mathtt{if}\, \square\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2\, \mathtt{fi}$ is a $(\mathtt{bool}, \tau)$ frame. Thus $\mathtt{if}\, \square\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2\, \mathtt{fi} \triangleright K$ is a `bool stack` (by the definition of well-formed stacks). Thus $(M, H, \mathtt{if}\, \square\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2\, \mathtt{fi} \triangleright K, e)$ ok, which is what we wanted to show.
  - If (true rule):

    $(M, H, \mathtt{if}\, \square\, \mathtt{then}\, e_1\, \mathtt{else}\, e_2\, \mathtt{fi} \triangleright K, \mathtt{true}) \mapsto_{\mathsf{C}} (M, H, K, e_1)$

We need to show that (M, H, K, e_1) ok. M and H are again unchanged. By Inversion, $e_1 : \tau, e_2 : \tau$. By Stack Inversion, $K : \tau\,\texttt{stack}$. Therefore (M, H, K, e_1) ok.

- If (false rule):
  The proof is exactly as in the previous case.

- For the `ref` cases:

  - Ref (push rule):

$$(M, H, K, \texttt{ref}(e)) \mapsto_{\mathsf{C}} (M, H, \texttt{ref}(\Box) \triangleright K, e)$$

  We want to show $(M, H, \texttt{ref}(\Box) \triangleright K, e)$ ok. We know that $\texttt{ref}(e)$ has type $\tau\,\texttt{ref}$ (by the typing rules), and e has type τ (by normal Inversion). M and H are again unchanged (at this stage). $\texttt{ref}(\Box) \triangleright K$ is then, by the same reasoning as before, a τ stack (since $\texttt{ref}(\Box)$ is a $(\tau, \tau\,\texttt{ref})$ frame). Thus $(M, H, \texttt{ref}(\Box) \triangleright K, e)$ ok.

  - Ref (loc rule)

$$(M, H, \texttt{ref}(\Box) \triangleright K, v) \mapsto_{\mathsf{C}} (M[\ell = v], H, K, l)$$

  We want to show that $(M[\ell = v], H, K, l)$ ok. H remains unchanged. Stack Inversion gives us $K : \tau\,\texttt{ref}\,\texttt{stack}$. A consistent typing for M can be found by extending Λ to $\Lambda(\ell) = \tau$. Thus $(M[\ell = v], H, K, l)$ ok.

  - Ref (dereference push):

$$(M, H, K, \,!\,e) \mapsto_{\mathsf{C}} (M, H, \,!\,\Box \triangleright K, e)$$

  We want to show $(M, H, \,!\,\Box \triangleright K, e)$ ok. We know that $!e$ has type τ (by the typing rules) and $e : \tau\,\texttt{ref}$ (by Inversion). M and H are again unchanged. $!\,\Box \triangleright K$ is then, by the same reasoning as before, a $\tau\,\texttt{ref}$ stack (since $!\,\Box$ is a $(\tau\,\texttt{ref}, \tau)$ frame). Thus $(M, H, \,!\,\Box \triangleright K, e)$ ok.

  -

- For the `try/ow` cases:

- Try (push rule):

$$(M, H, K, \texttt{try}\, e_1\, \texttt{ow}\, e_2) \mapsto_{\mathsf{C}} (M, \texttt{(}K\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K, e_1)$$

We need to show that $(M, \texttt{(}K\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K, e_1)\ \mathsf{ok}$. M remains unchanged. Since $H\ \mathsf{prefix}\ K$, $\texttt{(}K\texttt{,}e_2\texttt{)} \triangleright H\ \mathsf{prefix}\ \texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K$ (1 pop results in K). By Inversion, $e_1 : \tau$ and $e_2 : \tau$. By Stack Inversion, $\texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K$ is a $\tau\ \texttt{stack}$, and thus $(M, \texttt{(}K\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K, e_1)\ \mathsf{ok}$.

- Try (e_1 done):

$$(M, \texttt{(}K'\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K, v)$$

M remains unchanged. By the inductive definition of $H\ \mathsf{prefix}\ K$, we know that if $\texttt{(}K'\texttt{,}e_2\texttt{)} \triangleright H\ \mathsf{prefix}\ \texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K$ then $H\ \mathsf{prefix}\ K$. By Stack Inversion, we know that K is a $\tau\ \texttt{stack}$ and $v : \tau$, thus $(M, H, K, v)\ \mathsf{ok}$.

- For the `fail` cases:

  - Fail with a handler:

$$(M, \texttt{(}K'\texttt{,}e'\texttt{)} \triangleright H, K, \texttt{fail}) \mapsto_{\mathsf{C}} (M, H, K', e')$$

Because $\texttt{(}K'\texttt{,}e'\texttt{)} \triangleright H\ \mathsf{prefix}\ K$, we know that K' can be derived from K by some number of pops. By induction on the stack and frame composition rules, we see that K' is a $\tau\ \texttt{stack}$ and $e' : \tau$. M is unchanged and $H\ \mathsf{prefix}\ K'$. Thus, $(M, H, K', e')\ \mathsf{ok}$.

  - Fail without a handler:

$$(M, \bullet, K \neq \bullet, \texttt{fail}) \mapsto_{\mathsf{C}} (M, \bullet, \bullet, \texttt{fail})$$

M remains unchanged. $(M, \bullet, \bullet, \texttt{fail})$ is "trivially" $\texttt{ok}$.

4 Proof of Progress

Lemma 3: Canonical Forms
If v is a closed well formed value of type τ, then:
(lots of things that stay the same) ...
If $\tau = \tau_1\ \texttt{ref}$ then $v = \ell$.

That's it.

Lemma 4: Substitution
If $\Lambda; \Gamma[x : \tau] \vdash e' : \tau'$ *and* $\Lambda; \Gamma \vdash e : \tau$*, then* $\Lambda; \Gamma \vdash [e/x]e' : \tau'$*.*
We modify the definition of substitution to include a few new cases:

$$[e/x]\texttt{ref(}e'\texttt{)} = \texttt{ref(}[e/x]e'\texttt{)}$$

$$[e/x]\texttt{!}e' = \texttt{!}[e/x]e'$$

$$[e/x]e_1\texttt{:=}e_2 = [e/x]e_1\texttt{:=}[e/x]e_2$$

$$[e/x]\texttt{try}\, e_1 \,\texttt{ow}\, e_2 = \texttt{try}\, [e/x]e_1 \,\texttt{ow}\, [e/x]e_2$$

$$[e/x]\texttt{fail} = \texttt{fail}$$

The proof is by induction on the structure of the derivation $\Lambda; \Gamma[x : \tau] \vdash e' : \tau'$.

- $$\frac{\Lambda; \Gamma \vdash e'' : \tau}{\Lambda; \Gamma \vdash \texttt{ref(}e''\texttt{)} : \tau\,\texttt{ref}}$$

  By subderivation, $\Lambda; \Gamma[x : \tau] \vdash e'' : \tau$. By our inductive hypothesis, $\Lambda; \Gamma \vdash [e/x]e'' : \tau$. By the typing rule $\Lambda; \Gamma \vdash \texttt{ref(}[e/x]e''\texttt{)} : \tau\,\texttt{ref}$. By the definition of substitution $\Lambda; \Gamma[x : \tau] \vdash e' : \tau'$ as required (where $\tau' = tau\,\texttt{ref}$).

- I will omit the other cases, as they are all of the same structure (we will examine the one slightly different case in the extensible exceptions version). Compare with the solution to homework #2 if you have any questions about the details.

The statement of progress is a slight modification of that given in the notes:

Theorem (Progress):
If (M, H, K, e) ok *then either*

1. $K = \bullet$ *and* e value*, or*

2. $K = \bullet$ *and* $e = \mathtt{fail}$*, or*

3. there exists M', H', K' and e' such that $(M, H, K, e) \mapsto (M', H', K', e')$.

Proof:
We prove this by case analysis on well-formed machine states and typing rules for expressions. The cases may be broken down as follows:

(M, H, K, e) ok and:

- e is a value.
  - $K = \bullet$: This satisfies one of the conditions for progress.
  - Otherwise: We know that K is a well-formed τ stack, $F \triangleright K'$ composed of a frame F and a stack K'. We now consider the possibilities for F (from the rules for frame typing and stack/frame composition). Only the cases required for your proof will be given (the others are similar):
    * If frame:

      $$\frac{\vdash e_1 : \tau \quad \vdash e_2 : \tau}{\vdash \texttt{if}\,\Box\,\texttt{then}\,e_1\,\texttt{else}\,e_2\,\texttt{fi} : \texttt{(bool,}\tau\texttt{)}\ \texttt{frame}}$$

      Since (M, H, K, e) ok, we know by Stack Inversion that e : $\texttt{bool}$. By the Canonical Forms Lemma, if e is a value of type $\texttt{bool}$, e must be either $\texttt{true}$ or $\texttt{false}$. In either case, we may make progress (to (M, H, K, e_1) or (M, H, K, e_2)) by the appropriate evaluation rule.
    * Ref frame:

      $$\frac{}{\vdash \texttt{ref(}\Box\texttt{)} : \texttt{(}\tau\texttt{,}\tau\,\texttt{ref)}\ \texttt{frame}}$$

      We make progress by the evaluation rule:

      $$(M, H, \texttt{ref(}\Box\texttt{)} \triangleright K, v) \mapsto_{\mathsf{C}} (M[\ell = v], H, K, l)$$

    * ! frame:

      $$\frac{}{\vdash \texttt{!}\Box : \texttt{(}\tau\,\texttt{ref,}\tau\texttt{)}\ \texttt{frame}}$$

      We make progress by the evaluation rule:

      $$(M, H, \texttt{!}\Box \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K, M(v))$$

* $:=$ frames

$$\frac{\vdash e_2 : \tau}{\vdash \Box\texttt{:=}e_2 : (\tau\,\texttt{ref}\,\texttt{,}\tau\texttt{)}\ \texttt{frame}}$$

We make progress by the evaluation rule:

$$(M, H, \Box\texttt{:=}e_2 \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, v_1\texttt{:=}\Box \triangleright K, e_2)$$

.

$$\frac{v_1\ \mathsf{value}}{\vdash v_1\texttt{:=}\Box : \texttt{(}\tau\texttt{,}\tau\texttt{)}\ \texttt{frame}}$$

We make progress by the evaluation rule:

$$(M, H, v_1\texttt{:=}\Box \triangleright K, v_2) \mapsto_{\mathsf{C}} (M[v_1 = v_2], H, K, v_2)$$

* try frame:

$$\frac{\vdash e_2 : \tau}{\vdash \texttt{try}\,\Box\,\texttt{ow}\,e_2 : \texttt{(}\tau\texttt{,}\tau\texttt{)}\ \texttt{frame}}$$

We make progress by the evaluation rule:

$$(M, \texttt{(}K'\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\Box\,\texttt{ow}\,e_2 \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K, v)$$

- $e : \tau$ is not a value.
  - $e = \texttt{fail}$
    We consider two major cases if $e = \texttt{fail}$:
    * $H = (K', e') \triangleright H'$
      We make progress to (M, H', K', e') by the evaluation rule

      $$(M, \texttt{(}K'\texttt{,}e'\texttt{)} \triangleright H, K, \texttt{fail}) \mapsto_{\mathsf{C}} (M, H, K', e')$$

    * $H = \bullet$
      We must consider two cases:
      · $K = \bullet$
        This satisfies our statement of progress.
      · $K \neq \bullet$
        We make progress by the evaluation rule:

        $$(M, \bullet, K \neq \bullet, \texttt{fail}) \mapsto_{\mathsf{C}} (M, \bullet, \bullet, \texttt{fail})$$

– Otherwise we analyze the typing rules for each possible derivation of $e : \tau$. If $e : \tau$, then e must appear as the conclusion of some typing rule. If we can show that for any typing rule from which we might have derived the typing for e, the machine can make progress, we have shown what we wish to show. (`fail` is considered above not because it will not fit into this framework, but because I wish to emphasize the case analysis in that case.) We will only give the cases required for this assignment (the others are very similar):

  *
  $$\frac{\Lambda; \Gamma \vdash e_1 : \texttt{bool} \quad \Lambda; \Gamma \vdash e_2 : \tau \quad \Lambda; \Gamma \vdash e_3 : \tau}{\Lambda; \Gamma \vdash \texttt{if}\, e_1 \,\texttt{then}\, e_2 \,\texttt{else}\, e_3 \,\texttt{fi} : \tau}$$

  Thus, e has the form $\texttt{if}\, e_1 \,\texttt{then}\, e_2 \,\texttt{else}\, e_3 \,\texttt{fi}$. We make progress by the evaluation rule:

  $$(M, H, K, \texttt{if}\, e \,\texttt{then}\, e_1 \,\texttt{else}\, e_2 \,\texttt{fi}) \mapsto_{\mathsf{C}} (M, H, \texttt{if}\, \Box \,\texttt{then}\, e_1 \,\texttt{else}\, e_2 \,\texttt{fi} \triangleright K, e)$$

  *
  $$\frac{\Lambda; \Gamma \vdash e : \tau}{\Lambda; \Gamma \vdash \texttt{ref(}e\texttt{)} : \tau \,\texttt{ref}}$$

  Thus, e has the form $\texttt{ref(}e\texttt{)}$. We make progress by the evaluation rule:

  $$(M, H, K, \texttt{ref(}e\texttt{)}) \mapsto_{\mathsf{C}} (M, H, \texttt{ref(}\Box\texttt{)} \triangleright K, e)$$

  *
  $$\frac{\Lambda; \Gamma \vdash e : \tau \,\texttt{ref}}{\Lambda; \Gamma \vdash \texttt{!}e : \tau}$$

  Thus, e has the form $\texttt{!}e$. We make progress by the evaluation rule:

  $$(M, H, K, \texttt{!}e) \mapsto_{\mathsf{C}} (M, H, \texttt{!}\Box \triangleright K, e)$$

  *
  $$\frac{\Lambda; \Gamma \vdash e_1 : \tau_2 \,\texttt{ref} \qquad \Lambda; \Gamma \vdash e_2 : \tau_2}{\Lambda; \Gamma \vdash e_1 \texttt{:=} e_2 : \tau_2}$$

  Thus, e has the form $e_1 \texttt{:=} e_2$. We make progress by the evaluation rule:

  $$(M, H, K, e_1 \texttt{:=} e_2) \mapsto_{\mathsf{C}} (M, H, \Box \texttt{:=} e_2 \triangleright K, e_1)$$

*

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{try}\, e_1\, \texttt{ow}\, e_2 : \tau}$$

Thus, e has the form $\texttt{try}\, e_1\, \texttt{ow}\, e_2$. We make progress by the evaluation rule:

$$(M, H, K, \texttt{try}\, e_1\, \texttt{ow}\, e_2) \mapsto_{\mathsf{C}} (M, (K\texttt{,}e_2) \triangleright H, \texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K, e_1)$$

5 Evaluation Rules for the E-C Machine

- Rules for `try/ow`:

$$(M, H, K, \texttt{try}\, e_1\, \texttt{ow}\, e_2) \mapsto_{\mathsf{C}} (M, (K\texttt{,}e_2) \triangleright H, \texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K, e_1)$$

$$(M, (K\texttt{,}e_2) \triangleright H, \texttt{try}\,\Box\,\texttt{ow}\, e_2 \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K, v)$$

- Rules for `raise`:

$$(M, H, K, \texttt{raise(}e\texttt{)}) \mapsto_{\mathsf{C}} (M, H, \texttt{raise(}\Box\texttt{)} \triangleright K, e)$$

$$(M, (k'\texttt{,}e') \triangleright H, \texttt{raise(}\Box\texttt{)} \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, k', \texttt{apply(}e', v\texttt{)})$$

$$(M, \bullet, \texttt{raise(}\Box\texttt{)} \triangleright K \neq \bullet, v) \mapsto_{\mathsf{C}} (M, \bullet, \texttt{raise(}\Box\texttt{)}, v)$$

- Rules for `letexn`:

$$(M, H, K, \texttt{letexn}\, x\, \texttt{in}\, e\, \texttt{end}) \mapsto_{\mathsf{C}} (M, H, K, [newexn_n/x]e)$$

Where $newexn$ is a *new* exception (properly we would speak of $newexn_0$, $newexn_1$, etc. Each time a `letexn` step is performed, we substitute in the next in the series for the variable in the `letexn`. The only invariant for $newexn$ is that $newexn_a = newexn_b$ iff $a = b$. We get away with not having a global list of `exn` values because we have no namespace for them, only the `letexn` construct. In truth, of course, we have introduced a new element of machine state (the n in $newexn_n$), and it would be more formally correct now to speak of a (n, M, H, K, e) state. We would then change our evaluation rules so that n is incremented when a `letexn` is evaluated (in fact, this is what we do in the implementation).

- Rules for `exnEq`:

$$(M, H, K, (e_1)\,\texttt{exnEq}\,(e_2)) \mapsto_{\mathsf{C}} (M, H, (\Box)\,\texttt{exnEq}\,(e_2) \triangleright K, e_1)$$

$$(M, H, (\Box)\,\texttt{exnEq}\,(e_2) \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, (v_1)\,\texttt{exnEq}\,(\Box) \triangleright K, e_2)$$

$$(M, H, (v_1)\,\texttt{exnEq}\,(\Box) \triangleright K, v_2) \mapsto_{\mathsf{C}} (M, H, K, v_1 = v_2 : \texttt{bool})$$

6 Extended Implementation

See the code solution.

7 Extended Proof of Preservation

I will list only the modifications from the proof above.

Our frames are somewhat changed:

$$\frac{\vdash e_2 : \texttt{exn} \to \tau}{\vdash \texttt{try}\,\Box\,\texttt{ow}\,e_2 : (\tau\texttt{,}\tau)\ \texttt{frame}}$$

$$\frac{}{\vdash \texttt{raise(}\Box\texttt{)} : (\texttt{exn,}\tau)\ \texttt{frame}}$$

$$\frac{\vdash e_2 : \tau}{\vdash (\Box)\,\texttt{exnEq}\,(e_2) : (\texttt{exn,bool})\ \texttt{frame}} \qquad \frac{v_1\ \mathsf{value}}{\vdash (v_1)\,\texttt{exnEq}\,(\Box) : (\texttt{exn,bool})\ \texttt{frame}}$$

We do not need to change the definition of `ok`.

We need to make some minor changes to our lemmas, by adding:
If $\tau = \texttt{exn}$ *then* $v = newexn_i$.
to the Canonical Forms Lemma and (in addition to the "trivial" cases that I will not elaborate upon for `exnEq` and `raise`) a case for `letexn` to the Substitution Lemma:

$$\frac{\Gamma[x' : \texttt{exn}] \vdash e' : \tau}{\Gamma \vdash \texttt{letexn}\,x'\,\texttt{in}\,e'\,\texttt{end} : \tau}$$

By subderivation, $\Gamma[x' : \texttt{exn}] \vdash e' : \tau$. We use alpha-conversion to avoid capture (choose x s.t. it does not intersect $FV(e)$, x', or $\texttt{dom}(\Gamma)$). By the inductive hypothesis, $[e/x]e' : \tau$. By definition of substitution, then, $\texttt{letexn}\,x'\,\texttt{in}\,[e/x]e'\,\texttt{end} : \tau$.

We now have what we need for the proof of preservation:

- `try/ow`:

  - Push rule:

$$(M, H, K, \texttt{try}\, e_1\, \texttt{ow}\, e_2) \mapsto_{\mathsf{C}} (M, \texttt{(}K\,\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\square\,\texttt{ow}\, e_2 \triangleright K, e_1)$$

    We need to show that $(M, \texttt{(}K\,\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\square\,\texttt{ow}\, e_2 \triangleright K, e_1)$ ok. M remains unchanged. Since H prefix K, $\texttt{(}K\,\texttt{,}e_2\texttt{)} \triangleright H$ prefix $\texttt{try}\,\square\,\texttt{ow}\, e_2 \triangleright K$ (1 pop results in K). By Inversion, $e_1 : \tau$ and $e_2 : \texttt{exn}{\rightarrow}\tau$. By Stack Inversion, $\texttt{try}\,\square\,\texttt{ow}\, e_2{\triangleright}K$ is a $\tau\,\texttt{stack}$, and thus $(M, \texttt{(}K\,\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\square\,\texttt{ow}\, e_2 \triangleright K, e$

  - Try (e_1 done):

$$(M, \texttt{(}K'\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\,\square\,\texttt{ow}\, e_2 \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K, v)$$

    M remains unchanged. By the inductive definition of H prefix K, we know that if $\texttt{(}K'\texttt{,}e_2\texttt{)} \triangleright H$ prefix $\texttt{try}\,\square\,\texttt{ow}\, e_2 \triangleright K$ then H prefix K. By Stack Inversion, we know that K is a $\tau\,\texttt{stack}$ and $v : \tau$, thus (M, H, K, v) ok.

- `raise`:

  - Push rule:

$$(M, H, K, \texttt{raise(}e\texttt{)}) \mapsto_{\mathsf{C}} (M, H, \texttt{raise(}\square\texttt{)} \triangleright K, e)$$

    By Inversion, $e : \texttt{exn}$, and then by Stack Inversion $(M, H, \texttt{raise(}\square\texttt{)} \triangleright K, e)$ ok.

  - Raise with a handler:

$$(M, \texttt{(}K'\texttt{,}e'\texttt{)} \triangleright H, \texttt{raise(}\square\texttt{)} \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K', \texttt{apply(}e', v\texttt{)})$$

    Because $\texttt{(}K'\texttt{,}e'\texttt{)} \triangleright H$ prefix K, we know that K' can be derived from K by some number of pops. By induction on the stack and frame composition rules, we see that K' is a $\tau\,\texttt{stack}$ and $e' : \texttt{exn}{\rightarrow}\tau$. M is unchanged and H prefix K'. By Stack Inversion, $v : \texttt{exn}$. Thus, $(M, H, K', \texttt{apply(}e', v\texttt{)})$ ok.

  - Raise without a handler:

$$(M, \bullet, \texttt{raise(}\square\texttt{)} \triangleright K \neq \bullet, v) \mapsto_{\mathsf{C}} (M, \bullet, \texttt{raise(}\square\texttt{)}, v)$$

    This is a trivial analysis of $(M, \bullet, \texttt{raise(}\square\texttt{)}, v)$. H, M are unchanged, and $\bullet$ can have any type as a stack.

- `letexn`:

$$(M, H, K, \texttt{letexn}\, x\, \texttt{in}\, e\, \texttt{end}) \mapsto_{\mathsf{C}} (M, H, K, [newexn_n/x]e)$$

  M, H, K are unchanged. We simply need to show that $[newexn_n/x]e : \tau$ if $\texttt{letexn}\, x\, \texttt{in}\, e\, \texttt{end} : \tau$. This follows from the Substitution Lemma.

- `exnEq`:

  - Push rule:

$$(M, H, K, (e_1)\, \texttt{exnEq}\, (e_2)) \mapsto_{\mathsf{C}} (M, H, (\Box)\, \texttt{exnEq}\, (e_2) \triangleright K, e_1)$$

    As above, we use Inversion and Stack Inversion to show that $(M, H, (\Box)\, \texttt{exnEq}\, (e_2) \triangleright K, e_1)$ ok.

  -

$$(M, H, (\Box)\, \texttt{exnEq}\, (e_2) \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, (v_1)\, \texttt{exnEq}\, (\Box) \triangleright K, e_2)$$

8 Extended Proof of Progress

Again, most of the proof is unchanged. We slightly restate progress: **Theorem (Progress):**
If (M, H, K, e) ok *then either*

1. $K = \bullet$ *and* e value*, or*

2. $K = \texttt{raise}(\Box) \triangleright \bullet$ *and* $e : \texttt{exn}$ *is a value, or*

3. *there exists* M', H', K' *and* e' *such that* $(M, H, K, e) \mapsto (M', H', K', e')$.

- e is a value.

  - $K = \bullet$: This satisfies our conditions for progress.
  - Otherwise: We know (as above) that K is a well-formed τ `stack`, $F \triangleright K'$. Again, we consider the possibly cases for the typing of F.

*

$$\frac{\vdash e_2 : \texttt{exn}{\to}\tau}{\vdash \texttt{try}\,\Box\,\texttt{ow}\,e_2 : (\tau,\tau)\ \texttt{frame}}$$

We make progress by the evaluation rule:

$$(M, (K, e_2) \triangleright H, \texttt{try}\,\Box\,\texttt{ow}\,e_2 \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, K, v)$$

*

$$\frac{}{\vdash \texttt{raise}(\Box) : (\texttt{exn},\tau)\ \texttt{frame}}$$

We have two possible evaluation rules here (for the case that there is or is not a handler). In either case, we can make progress:

$$(M, (k', e') \triangleright H, \texttt{raise}(\Box) \triangleright K, v) \mapsto_{\mathsf{C}} (M, H, k', \texttt{apply}(e', v))$$

$$(M, \bullet, \texttt{raise}(\Box) \triangleright K \neq \bullet, v) \mapsto_{\mathsf{C}} (M, \bullet, \texttt{raise}(\Box), v)$$

A third possibility is that there is no handler and $K = \bullet$. However, this is (by our notion of progress) a final result.

*

$$\frac{\vdash e_2 : \tau}{\vdash (\Box)\ \texttt{exnEq}\,(e_2) : (\texttt{exn},\texttt{bool})\ \texttt{frame}}$$

We make progress by the evaluation rule:

$$(M, H, (\Box)\ \texttt{exnEq}\,(e_2) \triangleright K, v_1) \mapsto_{\mathsf{C}} (M, H, (v_1)\ \texttt{exnEq}\,(\Box) \triangleright K, e_2)$$

*

$$\frac{v_1\ \mathsf{value}}{\vdash (v_1)\ \texttt{exnEq}\,(\Box) : (\texttt{exn},\texttt{bool})\ \texttt{frame}}$$

We make progress by the evaluation rule:

$$(M, H, (v_1)\ \texttt{exnEq}\,(\Box) \triangleright K, v_2) \mapsto_{\mathsf{C}} (M, H, K, v_1 = v_2 : \texttt{bool})$$

e is not a value:
Again, we analyze the typing rules for the derivation of $e : \tau$. As in the earlier proof of progress, we basically match each typing rule with the "push" evaluation rule for the form of the expression:

–

$$\frac{\Gamma[x : \texttt{exn}] \vdash e : \tau}{\Gamma \vdash \texttt{letexn}\, x\, \texttt{in}\, e\, \texttt{end} : \tau}$$

We make progress by the evaluation rule:

$$(M, H, K, \texttt{letexn}\, x\, \texttt{in}\, e\, \texttt{end}) \mapsto_{\mathsf{C}} (M, H, K, [newexn_n/x]e)$$

–

$$\frac{\Gamma \vdash e : \texttt{exn}}{\Gamma \vdash \texttt{raise(}e\texttt{)} : \tau}$$

Progress by:

$$(M, H, K, \texttt{raise(}e\texttt{)}) \mapsto_{\mathsf{C}} (M, H, \texttt{raise(}\Box\texttt{)} \triangleright K, e)$$

–

$$\frac{\Gamma \vdash e_1 : \tau \qquad \Gamma \vdash e_2 : \texttt{exn} \rightarrow \tau}{\Gamma \vdash \texttt{try}\, e_1\, \texttt{ow}\, e_2 : \tau}$$

Progress by:

$$(M, H, K, \texttt{try}\, e_1\, \texttt{ow}\, e_2) \mapsto_{\mathsf{C}} (M, \texttt{(}K\texttt{,}e_2\texttt{)} \triangleright H, \texttt{try}\, \Box\, \texttt{ow}\, e_2 \triangleright K, e_1)$$

–

$$\frac{\Gamma \vdash e_1 : \texttt{exn} \quad \Gamma \vdash e_2 : \texttt{exn}}{\Gamma \vdash \texttt{(}e_1\texttt{)}\, \texttt{exnEq}\, \texttt{(}e_2\texttt{)} : \texttt{bool}}$$

Progress by:

$$(M, H, K, \texttt{(}e_1\texttt{)}\, \texttt{exnEq}\, \texttt{(}e_2\texttt{)}) \mapsto_{\mathsf{C}} (M, H, \texttt{(}\Box\texttt{)}\, \texttt{exnEq}\, \texttt{(}e_2\texttt{)} \triangleright K, e_1)$$