Types and Programming Languages (15-814), Fall 2018
Assignment 6: Continuing with Continuations

Contact: 15-814 Course Staff

Due Tuesday, November 6, 2018, 11:59pm

This assignment is due by 11:59pm on the above date and it must be submitted electronically as a PDF file on Canvas. Please use the attached template to typeset your assignment and make sure to include your full name and Andrew ID. As before, problems marked "WB" are subject to the whiteboard policy; all other problems must be done individually.

Task 1 (0 points)**.** How long did you spend on this assignment? Please list the questions that you discussed with classmates using the whiteboard policy.

1 Correctness of the K machine

In class, we briefly sketched the correctness of the K machine relative to the dynamics of our language. Correctness means the following two properties hold:

Theorem 1 (Completeness)**.** If $e \mapsto^* v$ and v *val*, then $\epsilon \rhd e \mapsto^* \epsilon \lhd v$.

Theorem 2 (Soundness)**.** If $\epsilon \rhd e \mapsto^* \epsilon \lhd v$, then $e \mapsto^* v$ and v *val*.

In this section, you will complete several cases of the proof. For simplicity, we restrict our attention to the fragment consisting of functions, eager products, and recursion. We assume throughout that our continuations k are well-typed, i.e., that there exist a τ_1 and σ such that $k \div \tau_1 \Rightarrow \sigma$. We also assume that our machine states s are well-typed. For the sake of this assignment, this means:

- $k \rhd e$ is well-typed whenever $k \div \tau_1 \Rightarrow \sigma$ and $\cdot \vdash e : \tau_1$ for some τ_1 and σ;
- $k \lhd v$ is well-typed whenever $k \div \tau_1 \Rightarrow \sigma$ and $\cdot \vdash v : \tau_1$ for some τ_1 and σ and v *val*.

The associated preservation theorem states that if s is well-typed and $s \mapsto s'$, then s' is well-typed.

The development is based entirely on the proof given in the textbook [2, section 28.3]. To help you understand the proof given there, we give an "evaluation dynamics" or "big-step semantics" for our language. An evaluation dynamics is a judgment $e \Downarrow v$, which means e evaluates to the value v. One can show that $e \Downarrow v$ if and only if $e \mapsto^* v$ and v *val*.

$$\frac{}{\lambda x.e \Downarrow \lambda x.e}\ (\Downarrow\text{-}\to_1) \qquad \frac{e_1 \Downarrow \lambda x.e_1' \quad e_2 \Downarrow v_2 \quad [v_2/x]e_1' \Downarrow v}{e_1 e_2 \Downarrow v}\ (\Downarrow\text{-}\to_2)$$

$$\frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{\langle e_1, e_2\rangle \Downarrow \langle v_1, v_2\rangle}\ (\Downarrow\text{-}\otimes_1) \qquad \frac{e \Downarrow \langle v_1, v_2\rangle \quad [v_1, v_2/x_1, x_2]e' \Downarrow v}{\textbf{case}\ e\ \{\langle x_1, x_2\rangle \Rightarrow e'\} \Downarrow v}\ (\Downarrow\text{-}\otimes_2)$$

Task 2 (5 points, WB)**.** Give a rule ($\Downarrow$-FIX) for $\textbf{fix}(x.e) \Downarrow v$. It must satisfy $\textbf{fix}(x.e) \Downarrow v$ if and only if $\textbf{fix}(x.e) \mapsto^* v$ and v *val*.

1.1 Completeness

Theorem 1 is a corollary of the following lemma and the observation that $e \Downarrow v$ if and only if $e \mapsto^* v$ and v *val*. To guide you in your proofs for the missing cases, we give a few example cases.

Lemma 3 ([2, Lemma 28.2])**.** If $e \Downarrow v$, then for every stack k, $k \rhd e \mapsto^* k \lhd v$.

Proof. By induction on the derivation of $e \Downarrow v$.

Case ($\Downarrow\text{-}\to_1$). Immediate by (A.1) (see appendix).

Case ($\Downarrow\text{-}\to_2$). Consider the rule

$$\frac{e_1 \Downarrow \lambda x.e_1' \quad e_2 \Downarrow v_2 \quad [v_2/x]e_1' \Downarrow v}{e_1 e_2 \Downarrow v}\ (\Downarrow\text{-}\to_2)$$

We must show that for all k, $k \rhd e_1 e_2 \mapsto^* k \lhd v$. Fix some arbitrary k.

(a)	$e_1 \Downarrow \lambda x.e_1'$	Case rule hypothesis
(b)	$e_2 \Downarrow v_2$	Case rule hypothesis
(c)	$[v_2/x]e_1' \Downarrow v$	Case rule hypothesis
(d)	$k \rhd e_1 e_2 \mapsto k \circ (\_\, e_2) \rhd e_1$	By (A.2)
(e)	for all k, $k \rhd e_1 \mapsto^* k \lhd \lambda x.e_1'$	By IH on a
(f)	$k \circ (\_\, e_2) \rhd e_1 \mapsto^* k \circ (\_\, e_2) \lhd \lambda x.e_1'$	By e

(g) $k \circ (\_\, e_2) \lhd \lambda x.e_1' \mapsto k \circ ((\lambda x.e_1')\, \_) \rhd e_2$ By (A.3)

(h) for all k, $k \rhd e_2 \mapsto^* k \lhd v_2$ IH on b

(i) $k \circ ((\lambda x.e_1')\, \_) \rhd e_2 \mapsto^* k \circ ((\lambda x.e_1')\, \_) \lhd v_2$ By h

(j) $k \circ ((\lambda x.e_1')\, \_) \lhd v_2 \mapsto k \rhd [v_2/x]e_1'$ By (A.4)

(k) for all k, $k \rhd [v_2/x]e_1' \mapsto^* k \lhd v$ IH on c

(l) $k \rhd e_1 e_2 \mapsto^* k \lhd v$ By items d, f, g and i to k □

Task 3 (5 points, WB)**.** Complete the case (⇓-FIX) of Lemma 3.

1.2 Soundness

To show soundness, we must show that if $\epsilon \rhd e \mapsto^* \epsilon \lhd v$, then $e \mapsto^* v$ and v *val*. As explained in the notes and in the textbook, we do so by showing that we can simulate machine transitions by the language's dynamics. To do so, we first show how to "unravel" our machine states $k \rhd e$ and $k \lhd e$ back into expressions e' using a function $k \bowtie e = e'$. We then relate machine states to unravelled expressions and claim that whenever a machine state steps, so does its associated expression, and that the resulting states and expressions are still related. In this subsection, you will explore this argument in greater detail.

We write $s \twoheadrightarrow e$ (in the textbook) or $e \; R \; s$ (in the lecture notes) if s is $k \rhd e'$ and $k \bowtie e' = e$, or if s is $k \lhd v$ and $k \bowtie v = e$. Observe that $\cdot \twoheadrightarrow \cdot$ and $\cdot \bowtie \cdot = \cdot$ behave as functions: if $s \twoheadrightarrow e$ and $s \twoheadrightarrow e'$, then $e = e'$; if $k \bowtie e = d$ and $k \bowtie e = d'$, then $d = d'$. The functions they describe are also total: for all well-typed $s = k \rhd e$ or $s = k \lhd e$, there exists a d such that $k \bowtie e = d$ and $s \twoheadrightarrow d$.

We will show that $\cdot \twoheadrightarrow \cdot$ is a (weak) simulation relation. This means that if $s \twoheadrightarrow e$ and $s \mapsto s'$, then there exists an e' such that $e \mapsto^* e'$ and $s' \twoheadrightarrow e'$. We observed above that such an e' is uniquely determined for any s', so it is sufficient to show that $e \mapsto^* e'$. By transitivity of $\mapsto^*$, it then follows that if $s \twoheadrightarrow e$, $s \mapsto s'$, $s' \twoheadrightarrow e'$, and $e' \mapsto^* v$, then $e \mapsto^* v$. The next task deduces the soundness theorem from this fact.

Task 4 (5 points, WB)**.** Assume that if $s \twoheadrightarrow e$, $s \mapsto s'$, $s' \twoheadrightarrow e'$, and $e' \mapsto^* v$, then $e \mapsto^* v$. Show that

1. for all well-typed s, if $s \twoheadrightarrow e$ and $s \mapsto^* \epsilon \lhd v$, then $e \mapsto^* v$ and v *val*;
2. if $\epsilon \rhd e$ is well-typed and $\epsilon \rhd e \mapsto^* \epsilon \lhd v$, then $e \mapsto^* v$ and v *val*.

Hint. Do not forget to answer all parts of the question! If you use the preservation theorem or the fact that s is well-typed, state where you do so.

Before we can show that $\cdot \twoheadrightarrow \cdot$ is a (weak) simulation relation, we must first show that unravelling respects the transition relation:

Task 5 ([2, Lemma 28.6], 5 points, WB)**.** The following proposition can be proven by induction on $e \mapsto e'$:

> If $e \mapsto e'$, $k \bowtie e = d$, and $k \bowtie e' = d'$, then $d \mapsto d'$.

Complete the case (CI-$\otimes_1$) of the proof.

Task 6 ([2, Lemma 28.3], 10 points, WB)**.** We can show by case analysis on $s \mapsto s'$ that $\cdot \not\mapsto \cdot$ is a simulation relation, i.e., that:

> If $s \not\mapsto e$, $s \mapsto s'$, and $s' \not\mapsto e'$, then $e \mapsto^* e'$.

Show the cases of $s \mapsto s'$ by (A.10) and (A.7).

2 Programming with Continuations

So far, we have treated the control stack as an abstract notion. However, we can extend our language with constructs that let us capture the current control stack as a value—a *continuation*—and that let us restore captured control stacks. With such mechanisms, we can easily implement backtracking algorithms (e.g., for search), algorithms that "return early" (e.g., the multiplication algorithm in [2, section 30.1]), and threads (concurrently executing programs). The reader is referred to [2, Chapter 30] for an expanded development of this material.

We begin by extending our types and syntax:

$$
\begin{array}{lr}
\tau ::= \cdots \mid \tau\ \mathbf{cont} & \\
e ::= \cdots & \\
\quad \mid \mathbf{letcc}(x.e) & \text{capture current continuation} \\
\quad \mid \mathbf{throw}(e_1 \Rightarrow e_2) & \text{jump to continuation} \\
\quad \mid \mathbf{cont}(k) & \text{continuation value}
\end{array}
$$

The typing rules are given by

$$
\frac{\Gamma, x : \tau\ \mathbf{cont} \vdash e : \tau}{\Gamma \vdash \mathbf{letcc}(x.e) : \tau} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_1\ \mathbf{cont}}{\Gamma \vdash \mathbf{throw}(e_1 \Rightarrow e_2) : \tau} \qquad \frac{k \div \tau \Rightarrow \sigma}{\Gamma \vdash \mathbf{cont}(k) : \tau\ \mathbf{cont}}
$$

The dynamics are given in terms of the K machine. We begin by extending the syntax of stacks, where $v\ val$:

$$
k ::= \cdots \mid k \circ \mathbf{throw}(- \Rightarrow e_2) \mid k \circ \mathbf{throw}(v \Rightarrow -)
$$

and for all well-typed stacks k, k *val*. Where v *val*, the rules are given by

$$k \rhd \mathbf{cont}(k') \mapsto k \lhd \mathbf{cont}(k') \tag{2.1}$$
$$k \rhd \mathbf{letcc}(x.e) \mapsto k \rhd [\mathbf{cont}(k)/x]e \tag{2.2}$$
$$k \rhd \mathbf{throw}(e_1 \Rightarrow e_2) \mapsto k \circ \mathbf{throw}(- \Rightarrow e_2) \rhd e_1 \tag{2.3}$$
$$k \circ \mathbf{throw}(- \Rightarrow e_2) \lhd v \mapsto k \circ \mathbf{throw}(v \Rightarrow -) \rhd e_2 \tag{2.4}$$
$$k \circ \mathbf{throw}(v \Rightarrow -) \lhd \mathbf{cont}(k') \mapsto k' \lhd v. \tag{2.5}$$

If we interpret the proposition $\neg P$ as the type P **cont**, then the following task gives a proof of the law of the excluded middle!

We begin with a small warm-up exercise to help you understand how these constructs work. Recall that we defined τ **opt** $= \tau + \mathbf{1}$ and syntactic sugar for the introduction and eliminations in assignment 2. Let

$$g : \forall\alpha.\forall\beta.\alpha \ \mathbf{opt} \to \beta \ \mathbf{opt} \ \mathbf{cont} \to \alpha$$
$$g = \lambda t.\lambda k.\mathbf{case}\ t\ \{\mathbf{some}\ x \Rightarrow x \mid \mathbf{none} \Rightarrow \mathbf{throw}(\mathbf{none} \Rightarrow k)\}$$
$$f : \forall\alpha.\forall\beta.(\alpha \ \mathbf{opt}) \otimes (\beta \ \mathbf{opt}) \to (\alpha \otimes \beta) \ \mathbf{opt}$$
$$f = \lambda p.\mathbf{letcc}(c.\mathbf{case}\ p\ \{\langle x, y\rangle \Rightarrow \mathbf{some}\ \langle gxc, gyc\rangle\}).$$

Task 7 (10 points, WB)**.** Give a v satisfying $\epsilon \rhd f\langle \mathbf{none}, \mathbf{some}\ \mathbf{none}\rangle \mapsto^* \epsilon \lhd v$.

Note. You will receive full credit for just writing down the correct v. You are not required to write down the reduction steps. However, to receive partial credit for an incorrect v, you must provide the reduction steps.

Task 8 (10 points, WB)**.** Find a closed term e such that $\cdot \vdash e : \tau + (\tau \ \mathbf{cont})$.

Hint. You will need to use both $\mathbf{letcc}(-.-)$ and $\mathbf{throw}(- \Rightarrow -)$. Let the types guide you.

It was not actually necessary to extend our language with special constructs to capture continuations. Indeed, we can give a translation[1] from the above extended language E with continuations to our base language B without continuations:

Theorem 4. Fix an answer type ρ in B. There exist translations $\|-\|, |-|$ from types in E to types in B, and a translation $\hat{-}$ from well-typed terms in E to terms in B, all defined in terms of ρ. If $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau$ in E, then $x_1 : \|\tau_1\|, \ldots, x_n : \|\tau_n\| \vdash \hat{e} : |\tau|$ in B.

[1] The following exposition is based one by E. Cavallo [1].

We begin by defining the translations for types. For simplicity, we consider only the fragment with functions, continuations, and the unit type. Fix some "answer" or "result" type ρ. Then let

$$\begin{aligned} |\tau| &= (\|\tau\| \to \rho) \to \rho \\ \|\mathbf{1}\| &= \mathbf{1} \\ \|\tau_1 \to \tau_2\| &= \|\tau_1\| \to |\tau_2| \\ \|\tau\ \mathbf{cont}\| &= \|\tau\| \to \rho \end{aligned}$$

Intuitively, $e : |\tau| = (\|\tau\| \to \rho) \to \rho$ means that given some "result continuation" of type $\|\tau\| \to \rho$, e can produce an result of type ρ. Informally, in the K machine, the stack $k \div \tau \Rightarrow \rho$ acts as the result continuation. Indeed, our Haskell implementation [3] of the K machine represented the stack as a continuation of type `E -> E`:

```
eval :: E -> (E -> E) -> E
retn :: E -> (E -> E) -> E
```

As an aside, we note that the translation $|\tau|$ is related to the double-negation translations of classical logic into intuitionistic logic.

Next, we define our translation for terms. The translation of $\Gamma \vdash e : \tau$ is inductively defined on the derivation of the typing judgment. We define the translation via a judgment $\Gamma \vdash e : \tau \rightsquigarrow \hat{e}$. As in past assignments, we colour-code "inputs" to the judgment in blue, and the "output" in red. This judgment is intended to mean that if $\Gamma \vdash e : \tau$ in E, then the translation $\hat{e}$ satisfies $\|\Gamma\| \vdash \hat{e} : |\tau|$ in B. (Showing that the definition of $\Gamma \vdash e : \tau \rightsquigarrow \hat{e}$ satisfies this property requires proof.)

$$\frac{}{\Gamma, x : \tau \vdash x : \tau \rightsquigarrow \lambda k.kx} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2 \rightsquigarrow \hat{e}}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2 \rightsquigarrow \lambda k.k(\lambda x.\hat{e})}$$

$$\frac{\Gamma \vdash e_1 : \tau \to \tau' \rightsquigarrow \hat{e}_1 \quad \Gamma \vdash e_2 : \tau \rightsquigarrow \hat{e}_2}{\Gamma \vdash e_1 e_2 : \tau' \rightsquigarrow \lambda k.\hat{e}_1(\lambda f.\hat{e}_2(\lambda v.fvk)))} \qquad \frac{}{\Gamma \vdash \langle\,\rangle : \mathbf{1} \rightsquigarrow \lambda k.k\langle\,\rangle}$$

There is a good operational intuition for the translation. It closely mimics reduction on a stack machine and makes the order of evaluation explicit. Indeed, compare the above rules to the corresponding definitions in our implementation of the K machine:

```
eval (Lam f) k = retn (Lam f) k
eval (App e1 e2) k = eval e1 (\(Lam f) ->
                                eval e2 (\v2 -> eval (f v2) k))
eval (Unit) k = retn (Unit) k
```

To understand the rules, it is important to remember the intended meaning of the $\Gamma \vdash e : \tau \rightsquigarrow \hat{e}$ judgment described above.

Consider the rule for variables. Its intended meaning is that $\|\Gamma\|, x : \|\tau\| \vdash \lambda k.kx : (\|\tau\| \to \rho) \to \rho$. This is exactly what the rule captures: given some continuation $k : \|\tau\| \to \rho$, kx has type ρ. The rule for the empty tuple is analogous.

We consider the rule for function application $e = e_1 e_2$. If $\Gamma \vdash e_1 e_2 : \tau'$ in E, then we need the translation $\hat{e}$ to satisfy $\|\Gamma\| \vdash \hat{e} : |\tau'|$ in B. Explicitly, this means that we need $\|\Gamma\| \vdash \hat{e} : (\|\tau'\| \to \rho) \to \rho$ in B. We consider how we can combine the translations of $\hat{e}_1$ and $\hat{e}_2$ to get $\hat{e}$.

Observe that $\Gamma \vdash e_1 : \tau \to \tau' \rightsquigarrow \hat{e}_1$ means that $\|\Gamma\| \vdash \hat{e}_1 : |\tau \to \tau'|$. Expanding the definition of $|\tau \to \tau'|$, this means that $\|\Gamma\| \vdash \hat{e}_1 : (\|\tau \to \tau'\| \to \rho) \to \rho$. Intuitively, this means that $\hat{e}_1$ can produce a term of type ρ given a return continuation $\|\tau \to \tau'\| \to \rho$. The type $\|\tau \to \tau'\| = \|\tau\| \to (\|\tau'\| \to \rho) \to \rho$ intuitively says that given an argument w of the input type $\|\tau\|$ and a return continuation $k : (\|\tau'\| \to \rho)$, we can return to k the result of the applying the function on w.

$\Gamma \vdash e_2 : \tau \rightsquigarrow \hat{e}_2$ means that $\|\Gamma\| \vdash \hat{e}_2 : |\tau|$, i.e., $\|\Gamma\| \vdash \hat{e}_2 : (\|\tau\| \to \rho) \to \rho$. Intuitively, $\hat{e}_2$ can produce a term of type ρ given a return continuation for values of type $\|\tau\|$.

We must produce a term $\hat{e}$ of type $(\|\tau'\| \to \rho) \to \rho$ in B. Because it is of function type, the most sensible thing is to form an $\hat{e} = \lambda k. \cdots$, where k has type $\|\tau'\| \to \rho$. We must find a body of type ρ. The term $\hat{e}_1$ can produce something of type ρ given a term type $\|\tau \to \tau'\| \to \rho$, so we attempt to write a function of type $\|\tau \to \tau'\| \to \rho$. To do so, we assume an f of type $\|\tau \to \tau'\|$ and must produce a term of type ρ. The term $\hat{e}_2$ can produce something of type ρ given a term of type $\|\tau\| \to \rho$. We attempt to write a function of this type and assume a v of type $\|\tau\|$.

At this point, we have the following context:

$$\Gamma' = \|\Gamma\|, k : \|\tau'\| \to \rho, f : \|\tau\| \to (\|\tau'\| \to \rho) \to \rho, v : \|\tau\|.$$

Then $\Gamma' \vdash fvk : \rho$. From here, repeated applications of the abstraction and application rules give us $\|\Gamma\| \vdash \lambda k.\hat{e}_1(\lambda f.\hat{e}_2(\lambda v.fvk))) : |\tau'|$ as desired.

You are encouraged to try understand the abstraction rule for yourself.

Task 9 (10 points, WB)**.** Give translations for $\mathbf{letcc}(x.e)$ and $\mathbf{throw}(e_1 \Rightarrow e_2)$.

Hint. Don't overthink these and let the types be your guide! Make sure your solution satisfies theorem 4, i.e., that if $\Gamma \vdash e : \tau \rightsquigarrow \hat{e}$, then $\|\Gamma\| \vdash \hat{e} : |\tau|$ in B.

References

[1] E. CAVALLO, *Homework 5: Parallelism and control flow*. http://www.cs.cmu.edu/~rwh/courses/typesys/hws/hw5/hw5-handout.pdf, November 2015. CMU 15-814.

[2] R. HARPER, *Practical Foundations for Programming Languages*, Cambridge University Press, 2 ed., 2016.

[3] F. PFENNING, *Lecture Notes on the K Machine*, 15-814 Lecture Notes, 15 (2018).

A K machine rules

$$k \rhd \lambda x.e \mapsto k \lhd \lambda x.e \tag{A.1}$$
$$k \rhd e_1 e_2 \mapsto k \circ (\_\, e_2) \rhd e_1 \tag{A.2}$$
$$k \circ (\_\, e_2) \lhd v_1 \mapsto k \circ (v_1\, \_) \rhd e_2 \tag{A.3}$$
$$k \circ ((\lambda x.e_1)\, \_) \lhd v_2 \mapsto k \rhd [v_2/x]e_1 \tag{A.4}$$
$$k \rhd \langle e_1, e_2 \rangle \mapsto k \circ \langle \_, e_2 \rangle \rhd e_1 \tag{A.5}$$
$$k \circ \langle \_, e_2 \rangle \lhd v_1 \mapsto k \circ \langle v_1, \_ \rangle \rhd e_2 \tag{A.6}$$
$$k \circ \langle v_1, \_ \rangle \lhd v_2 \mapsto k \lhd \langle v_1, v_2 \rangle \tag{A.7}$$
$$k \rhd \textbf{case}\ e\ \{\langle x_1, x_2 \rangle \Rightarrow e'\} \mapsto k \circ \textbf{case}\ \_\ \{\langle x_1, x_2 \rangle \Rightarrow e'\} \rhd e \tag{A.8}$$
$$k \circ \textbf{case}\ \_\ \{\langle x_1, x_2 \rangle \Rightarrow e'\} \lhd \langle v_1, v_2 \rangle \mapsto k \rhd [v_1, v_2/x_1, x_2]e' \tag{A.9}$$
$$k \rhd \textbf{fix}(x.e) \mapsto k \rhd [\textbf{fix}(x.e)/x]e \tag{A.10}$$

We include the reductions for sums only for Task 7:

$$k \rhd l \cdot e \mapsto k \circ (l \cdot -) \rhd e \tag{A.11}$$
$$k \circ (l \cdot -) \lhd v \mapsto k \lhd l \cdot v \tag{A.12}$$
$$k \rhd \textbf{case}\ e\ \{l_i \cdot x_i \Rightarrow e_i\}_{i \in I} \mapsto k \circ \textbf{case}\ -\ \{l_i \cdot x_i \Rightarrow e_i\}_{i \in I} \rhd e \tag{A.13}$$
$$k \circ \textbf{case}\ -\ \{l_i \cdot x_i \Rightarrow e_i\}_{i \in I} \lhd l_j \cdot v_j \mapsto k \rhd [v_j/x_j]e_j \tag{A.14}$$

B Dynamics: $e \mapsto e'$ and v *val*

$$\frac{}{\lambda x.e \ \textit{val}}\ (\text{V-}\to) \qquad \frac{v_2 \ \textit{val}}{(\lambda x.\, e_1)\, v_2 \mapsto [v_2/x]e_1}\ (\text{R-}\to)$$

$$\frac{e_1 \mapsto e_1'}{e_1\, e_2 \mapsto e_1'\, e_2}\ (\text{CE-}\to_1) \qquad \frac{v_1 \ \textit{val} \quad e_2 \mapsto e_2'}{v_1\, e_2 \mapsto e_1\, e_2'}\ (\text{CE-}\to_2)$$

$$\frac{v_1 \ \textit{val} \quad v_2 \ \textit{val}}{\langle v_1, v_2\rangle \ \textit{val}}\ (\text{V-}\otimes) \quad \frac{e_1 \mapsto e_1'}{\langle e_1, e_2\rangle \mapsto \langle e_1', e_2\rangle}\ (\text{CI-}\otimes_1) \quad \frac{v_1 \ \textit{val} \quad e_2 \mapsto e_2'}{\langle v_1, e_2\rangle \mapsto \langle v_1, e_2'\rangle}\ (\text{CI-}\otimes_2)$$

$$\frac{e_0 \mapsto e_0'}{\textbf{case}\ e_0\ \{\langle x_1, x_2\rangle \Rightarrow e'\} \mapsto \textbf{case}\ e_0'\ \{\langle x_1, x_2\rangle \Rightarrow e'\}}\ (\text{CE-}\otimes)$$

$$\frac{v_1 \ \textit{val} \quad v_2 \ \textit{val}}{\textbf{case}\ \langle v_1, v_2\rangle\ \{\langle x_1, x_2\rangle \Rightarrow e'\} \mapsto [v_1/x_1, v_2/x_2]e'}\ (\text{R-}\otimes)$$

$$\frac{}{\textbf{fix}(x.e) \mapsto [\textbf{fix}(x.e)/x]e}\ (\text{R-FIX})$$