1 Source Language Syntax

2 Source Language Statics and Dynamics

3 The Target Language

4 The Translation

4.1 Translating Evidence

4.1.1 Helper Functions

With our evidence represented as tuples with integer tags, we must represent the partial functions on types in our target language. The implementation is given in Figure 7. Doing this is straightforward: if one argument is $[[?]]$, then we return the other argument. Otherwise, we check if we have simple or complex types. For simple types, either $[[Bool/\ Bool ==== Bool]]$, $[[Nat/\ Nat ==== Nat]]$. For complex types, we check that the tags agree, then recursively compute the meets of the sub-components. If neither argument is $[[?]]$, and there is a tag mismatch, then we must raise an exception, retuning the $[[error]]$ continuation.

For the partial functions decomposing types, we first check if the input is $[[?]]$, in which case we return $[[?]]$. Otherwise, we check the tag, and if it is correct, we return the relevant sub-component of the type. In all other cases, we throw an error. We give an example implementation for **dom** in Figure 8: either we are given $[[?]]$ and return $[[?]]$, we are given $[[T1- > T2]]$ and we return $[[T1]]$, or we raise an exception. We omit **cod**, $\mathbf{proj}_1$ and $\mathbf{proj}_2$, but they are implemented similarly.

5 Correctness

Lemma 5.1 (Correctness of Evidence Translation)**.** *Consider evidence $[[ep]], [[ep']]$. Then, for any $[[k]]$:*

- $[[MEET[[|ep|],[|ep'|],k]--> *k[[|ep/\ ep'|]]]]$ *if* $[[ep/\ ep']]$ *is defined.*
- *If* $[[ep/\ ep']]$ *is undefined, then* $[[MEET[[|ep|],[|ep'|]]--> *error]]$.

The same property holds for $[[domep]]$*,* $[[codep]]$*, and* $[[Proj@iep]]$.

Lemma 5.2 (Canonical Forms for Translated Values)**.** *For an irreducible* $[[v]]$*,* $[[[|v|] ==== ([|ep|],u)]]$ *for some evidence* $[[ep]]$ *and CPS-value* $[[u]]$*. Moreover, if* $[[v]]$ *is a raw irreducible, then* $[[ep]] = [[<<?>>]]$.

Proof. By inversion on the definition of $[[[|v|]]]$. $\square$

$[[n]] \in \mathbb{Z}$, $[[b]] \in \mathbb{B}$

e	$::=$		
	$\mid$	x	Variables
	$\mid$	b	Booleans
	$\mid$	n	Natural Numbers
	$\mid$	$\lambda x : T.\, e$	Functions
	$\mid$	$e_1\ e_2$	Function Application
	$\mid$	$e_1 + e_2$	Addition
	$\mid$	$e_1 \stackrel{?}{=} e_2$	Number Equality Test
	$\mid$	$\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3$	Conditionals
	$\mid$	$\langle e_1, e_2 \rangle$	Tuples
	$\mid$	$\pi_1 e$	Tuple First Projection
	$\mid$	$\pi_2 e$	Tuple Second Projection
	$\mid$	$\varepsilon\ e$	Evidence Ascription
	$\mid$	$\textbf{error}$	Runtime Type Error

O	$::=$		Observable values
	$\mid$	b	
	$\mid$	n	

v	$::=$		Irreducable (closed) terms
	$\mid$	$\varepsilon\, r$	
	$\mid$	b	
	$\mid$	n	
	$\mid$	$\lambda x : T.\, e$	Functions
	$\mid$	$\langle v_1, v_2 \rangle$	

r	$::=$		Raw Irreducable (closed) terms (i.e. not ascribed with)
	$\mid$	b	
	$\mid$	n	
	$\mid$	$\lambda x : T.\, e$	Functions
	$\mid$	$\langle v_1, v_2 \rangle$	

Figure 1: Source Language Syntax: Terms

$$
\begin{array}{lcll}
T & ::= & & \text{Types} \\
 & | & \mathsf{Nat} & \\
 & | & \mathsf{Bool} & \\
 & | & T_1 \to T_2 & \\
 & | & T_1 \times T_2 & \\
 & | & ? & \\
\end{array}
$$

$$
\begin{array}{lcl}
\varepsilon & ::= & \\
 & | & \{T\} \\
\end{array}
$$

$$
\begin{array}{lcll}
\Gamma & ::= & & \text{Environments} \\
 & | & \cdot & \\
 & | & \Gamma, (x : T) & \\
\end{array}
$$

Figure 2: Source Language Syntax: Types

$\boxed{\Gamma \vdash e : T}$ *(Typability relation)*

$$
\textsc{HastypeVar}\ \frac{(x : T) \in \Gamma}{\Gamma \vdash x : T}
\qquad
\textsc{HastypeBool}\ \frac{}{\Gamma \vdash b : \mathsf{Bool}}
\qquad
\textsc{HastypeNat}\ \frac{}{\Gamma \vdash n : \mathsf{Nat}}
\qquad
\textsc{HastypePlus}\ \frac{\Gamma \vdash e_1 : \mathsf{Nat} \quad \Gamma \vdash e_2 : \mathsf{Nat}}{\Gamma \vdash e_1 + e_2 : \mathsf{Nat}}
\qquad
\textsc{HastypeEq}\ \frac{\Gamma \vdash e_1 : \mathsf{Nat} \quad \Gamma \vdash e_2 : \mathsf{Nat}}{\Gamma \vdash e_1 \stackrel{?}{=} e_2 : \mathsf{Nat}}
$$

$$
\textsc{HastypeLam}\ \frac{\Gamma, (x : T_1) \vdash e : T_2}{\Gamma \vdash \lambda x : T_1.\, e : T_1 \to T_2}
\qquad
\textsc{HastypeApp}\ \frac{\Gamma \vdash e_1 : T_1 \to T_2 \quad \Gamma \vdash e_2 : T_1}{\Gamma \vdash e_1\, e_2 : T_2}
\qquad
\textsc{HastypeIf}\ \frac{\Gamma \vdash e : \mathsf{Bool} \quad \Gamma \vdash e_1 : T \quad \Gamma \vdash e_2 : T}{\Gamma \vdash \textbf{if}\ e\ \textbf{then}\ e_1\ \textbf{else}\ e_2 : T}
$$

$$
\textsc{HastypePair}\ \frac{\Gamma \vdash e_1 : T_1 \quad \Gamma \vdash e_2 : T_2}{\Gamma \vdash \langle e_1, e_2 \rangle : T_1 \times T_2}
\qquad
\textsc{HastypeProj}\ \frac{\Gamma \vdash e : T_1 \times T_2 \quad i \in \{1, 2\}}{\Gamma \vdash \mathsf{p}_i\, e : T}
\qquad
\textsc{HastypeAscr}\ \frac{\Gamma \vdash e : T_2 \quad \varepsilon \vdash T_1 \cong T_2}{\Gamma \vdash \varepsilon\, e : T_1}
$$

Figure 3: Source Language: Type Rules

$\boxed{\varepsilon \vdash T_1 \cong T_2}$ *(Type Consistency relative to Evidence)*

$$\textsc{ConsistentEv}\ \frac{T_3 \sqcap T_1 = T_3 \qquad T_3 \sqcap T_2 = T_3}{\{T_3\} \vdash T_1 \cong T_2}$$

$\boxed{T_1 \sqcap T_2 = T_3}$ *(Precision Meet)*

$$\textsc{MeetDynL}\ \frac{}{? \sqcap T = T} \qquad \textsc{MeetDynR}\ \frac{}{T \sqcap ? = T} \qquad \textsc{MeetRefl}\ \frac{}{T \sqcap T = T}$$

$$\textsc{MeetFun}\ \frac{T_1 \sqcap T_1' = T_1'' \qquad T_2 \sqcap T_2' = T_2''}{T_1 \to T_2 \sqcap T_1' \to T_2' = T_1'' \to T_2''}$$

$$\textsc{MeetProd}\ \frac{T_1 \sqcap T_1' = T_1'' \qquad T_2 \sqcap T_2' = T_2''}{T_1 \times T_2 \sqcap T_1' \times T_2' = T_1'' \times T_2''}$$

Figure 4: Source Language: Type Consistency and Precision

$\boxed{e_1 \longrightarrow e_2}$ *(Reduction Relation on terms)*

RedIfTrue
$$\frac{}{\textbf{if true then } e_1 \textbf{ else } e_2 \longrightarrow e_1}$$

RedIfFalse
$$\frac{}{\textbf{if false then } e_1 \textbf{ else } e_2 \longrightarrow e_2}$$

RedIfEv
$$\frac{\textbf{if } b \textbf{ then } e_1 \textbf{ else } e_2 \longrightarrow e}{\textbf{if } \varepsilon\, b \textbf{ then } e_1 \textbf{ else } e_2 \longrightarrow e}$$

RedApp
$$\frac{}{(\lambda x : T.\, e)\, v \longrightarrow [v/x]e}$$

RedAppEv
$$\frac{}{(\varepsilon_1\, (\lambda x : T.\, e))\, (\varepsilon_2\, r) \longrightarrow \textbf{cod}\, \varepsilon_2\, ([\textbf{dom}\, \varepsilon_1 \sqcap \varepsilon_2\, r/x]e)}$$

RedAppEvFail
$$\frac{\textbf{dom}\, \varepsilon_1 \sqcap \varepsilon_2 \textbf{ undefined}}{(\varepsilon_1\, (\lambda x : T.\, e))\, (\varepsilon_2\, r) \longrightarrow \textbf{error}}$$

RedAppEvPartial
$$\frac{}{(\varepsilon\, (\lambda x : T.\, e_1))\, r \longrightarrow \textbf{cod}\, \varepsilon\, ([\textbf{dom}\, \varepsilon\, e_2/x]e_1)}$$

RedPlus
$$\frac{}{n_1 + n_2 \longrightarrow n_1 + n_2}$$

RedPlusEvL
$$\frac{n_1 + v \longrightarrow e}{\varepsilon\, n_1 + v \longrightarrow e}$$

RedPlusEvR
$$\frac{n_1 + n_2 \longrightarrow e}{n_1 + \varepsilon\, n_2 \longrightarrow e}$$

RedEqT
$$\frac{}{n \stackrel{?}{=} n \longrightarrow \textbf{true}}$$

RedEqF
$$\frac{n_1 \neq n_2}{n_1 \stackrel{?}{=} n_2 \longrightarrow \textbf{false}}$$

RedEqEvL
$$\frac{n_1 \stackrel{?}{=} v \longrightarrow e}{\varepsilon\, n_1 \stackrel{?}{=} v \longrightarrow e}$$

RedEqEvR
$$\frac{n_1 \stackrel{?}{=} n_2 \longrightarrow e}{n_1 \stackrel{?}{=} \varepsilon\, n_2 \longrightarrow e}$$

RedProj
$$\frac{i \in \{1, 2\}}{\pi_i\langle e_1, e_2\rangle \longrightarrow e_i}$$

RedProjEv
$$\frac{i \in \{1, 2\}}{\pi_i(\varepsilon\, \langle e_1, e_2\rangle) \longrightarrow \textbf{Proj}_i \varepsilon\, e_i}$$

RedAscr
$$\frac{}{\varepsilon_1\, (\varepsilon_2\, e) \longrightarrow \varepsilon_1 \sqcap \varepsilon_2\, e}$$

RedAscrFail
$$\frac{\varepsilon_1 \sqcap \varepsilon_2 \textbf{ undefined}}{\varepsilon_1\, (\varepsilon_2\, e) \longrightarrow \varepsilon_1 \sqcap \varepsilon_2\, e}$$

RedContext
$$\frac{e_1 \longrightarrow e_2}{C[e_1] \longrightarrow C[e_2]}$$

RedContextFail
$$\frac{e \longrightarrow \textbf{error}}{C[e] \longrightarrow C[\textbf{error}]}$$

Figure 5: Source Language: Small-Step Operational Semantics

$$
\begin{array}{lll}
u,\ k & ::= & \\
& | & x \\
& | & n \\
& | & b \\
& | & \mathbf{fix}\, x\ u \\
& | & \lambda x_1 \ldots x_i .\, t \\
& | & \langle u_1, u_2 \rangle
\end{array}
$$

$$
\begin{array}{lll}
d & ::= & \\
& | & x := u \\
& | & x := \pi_1 u \\
& | & x := \pi_2 u \\
& | & x := u_1 + u_2 \\
& | & x := u_1 \stackrel{?}{=} u_2
\end{array}
$$

$$
\begin{array}{lll}
t & ::= & \\
& | & v \\
& | & \mathbf{let}\ d\ \mathbf{in}\ t \\
& | & u(arg) \\
& | & \mathbf{if}\ u\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \\
& | & \mathbf{halt}\ [u] \\
& | & \mathbf{error}
\end{array}
$$

Figure 6: Target Language: Syntax

$$
\begin{aligned}
[[MEET]] \quad =& [[fixself\ ty1ty2c.lettag1 := pi1ty1inletsub1 := pi2ty1inletisDyn1 := tag1 == DYNin \\
]] \quad & [[ifisDyn1c[ty2] \\
]] \quad & [[lettag2 := pi1ty2inletsub2 := pi2ty2inletisDyn2 := tag2 == DYNin \\
]] \quad & [[ifisDyn2c[ty1] \\
]] \quad & [[letisNat1 := tag1 == NATinletisNat2 := tag2 == NATin \\
]] \quad & [[ifisNat1(ifisNat2k[NAT]error) \\
]] \quad & [[letisBool1 := tag1 == BOOLinletisBool2 := tag2 == BOOLin \\
]] \quad & [[ifisBool1(ifisBool2k[BOOL]error) \\
]] \quad & [[letisArrow1 := tag1 == ARRinletisArrow2 := tag2 == ARRin \\
]] \quad & [[iifisArrow1 \\
]] \quad & [[--letdom1 := pi1sub1inletcod1 := pi2sub1in \\
]] \quad & [[--iifisArrow2 \\
]] \quad & [[----letdom2 := pi1sub2inletcod2 := pi2sub2in \\
]] \quad & [[----self[dom1, dom2, (\ meet1.self[cod1, cod2, (\ meet2.k[(ARR, (meet1, meet2))])])]]] \\
& [[--elseerror]] \\
& [[letisProduct1 := tag1 == PRODinletisProduct2 := tag2 == PRODin \\
]] \quad & [[iifisProduct1 \\
]] \quad & [[--letlhs1 := pi1sub1inletrhs1 := pi2sub1in \\
]] \quad & [[--iifisProduct2 \\
]] \quad & [[----letlhs2 := pi1sub2inletrhs2 := pi2sub2in \\
]] \quad & [[----self[lhs1, lhs2, (\ meet1.self[rhs1, rhs2, (\ meet2.k[(PROD, (meet1, meet2))])])]]] \\
& [[--elseerror]] \\
& [[elseerror]]
\end{aligned}
$$

Figure 7: CPS implementation of meet

$$
\begin{aligned}
[[DOM]] \quad =& [[\ tyc.lettag := pi1ty1inletsub := pi2ty1inletisDyn := tag == DYNin \\
]] \quad & [[ifisDync[(DYN, 0)] \\
]] \quad & [[letisArrow := tag == ARRin \\
]] \quad & [[ifisArrow(letret := pi1subink[ret])error]]
\end{aligned}
$$

Figure 8: CPS implementation of domain

$\boxed{[\![e]\!]k = t}$ *(CPS Translation of Expressions)*

TransformVar
$$\frac{}{[\![x]\!]k = k(x)}$$

TransformBool
$$\frac{}{[\![b]\!]k = k(\langle \mathtt{DYN}, b\rangle)}$$

TransformNum
$$\frac{}{[\![n]\!]k = k(\langle \mathtt{DYN}, n\rangle)}$$

TransformFun
$$\frac{[\![e]\!]c = t}{[\![(\lambda x : T.\, e)]\!]k = k(\langle \mathtt{DYN}, \lambda x\ c.\, t\rangle)}$$

TransformApp
$$\frac{\begin{array}{c} k_1 := (\lambda x_2.\, \mathbf{let}\ y_1 := \pi_1 x_1\ \mathbf{in}\ \mathbf{let}\ z_1 := \pi_2 x_1\ \mathbf{in}\ \mathbf{let}\ y_2 := \pi_1 x_2\ \mathbf{in}\ \mathbf{let}\ z_2 := \pi_2 x_2\ \mathbf{in}\ t_1) \\ t_1 := \mathtt{DOM}(y_1, \lambda y_1'.\, \mathtt{COD}(y_1, \lambda y_1''.\, \mathtt{MEET}(y_1', y_2, (\lambda y_3.\, z_1(\langle y_3, z_2\rangle, (\lambda z_3.\, \mathbf{let}\ z_3' := \pi_1 z_3\ \mathbf{in}\ \mathbf{let}\ z_3'' := \pi_2 z_3\ \mathbf{in}\ \mathtt{MEET}(y_1'', z_3', (\lambda \\ [\![e_2]\!]k_1 = t' \\ [\![e_1]\!](\lambda x_1.\, t') = t \end{array}}{[\![e_1\ e_2]\!]k = t}$$

TransformPlus
$$\frac{\begin{array}{c} k_1 := (\lambda x_2.\, \mathbf{let}\ z_1 := \pi_2 x_1\ \mathbf{in}\ \mathbf{let}\ z_2 := \pi_2 x_2\ \mathbf{in}\ \mathbf{let}\ z_3 := z_1 + z_2\ \mathbf{in}\ k(z_3)) \\ [\![e_2]\!]k_1 = t' \\ [\![e_1]\!](\lambda x_1.\, t') = t \end{array}}{[\![e_1 + e_2]\!]k = t}$$

TransformEq
$$\frac{\begin{array}{c} k_1 := (\lambda x_2.\, \mathbf{let}\ z_1 := \pi_2 x_1\ \mathbf{in}\ \mathbf{let}\ z_2 := \pi_2 x_2\ \mathbf{in}\ \mathbf{let}\ z_3 := z_1 \stackrel{?}{=} z_2\ \mathbf{in}\ k(z_3)) \\ [\![e_2]\!]k_1 = t' \\ [\![e_1]\!](\lambda x_1.\, t') = t \end{array}}{[\![e_1 + e_2]\!]k = t}$$

TransformPair
$$\frac{\begin{array}{c} [\![e_2]\!](\lambda x_2.\, k(\langle \mathtt{DYN}, \langle x_1, x_2\rangle\rangle)) = t' \\ [\![e_1]\!](\lambda x_1.\, t') = t \end{array}}{[\![\langle e_1, e_2\rangle]\!]k = t}$$

TransformIf
$$\frac{\begin{array}{c} [\![e_1]\!]k = t_1 \qquad [\![e_2]\!]k = t_2 \\ [\![e_0]\!](\lambda x_0.\, \mathbf{let}\ x := \pi_2 x_0\ \mathbf{in}\ \mathbf{if}\ x\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2) = t \end{array}}{[\![\mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]\!]k = t}$$

TransformProj
$$\frac{[\![e]\!](\lambda x.\, \mathbf{let}\ x' :=\ \pi\,@i\ x\ \mathbf{in}\ k(x')) = t}{[\![\pi_i e]\!]k = t}$$

TransformEv
$$\frac{[\![e]\!](\lambda x.\, \mathbf{let}\ x_1 := \pi_1 x\ \mathbf{in}\ \mathbf{let}\ x_2 := \pi_2 x\ \mathbf{in}\ \mathtt{MEET}([\![\varepsilon]\!], x_1, (\lambda y.\, k(\langle y, x_2\rangle)))) = t}{[\![\varepsilon\ e]\!]k = t}$$

TransformErr
$$\frac{}{[\![\mathbf{error}]\!]k_8\!= \mathbf{error}}$$

Translation: Terms and Programs

$\boxed{[\![\varepsilon]\!] = u}$ *()*

EvTransformBool

$$\frac{}{[\![\{\mathsf{Bool}\}]\!] = \langle \mathtt{BOOL}, 0\rangle}$$

EvTransformNat

$$\frac{}{[\![\{\mathsf{Nat}\}]\!] = \langle \mathtt{NAT}, 0\rangle}$$

EvTransformDyn

$$\frac{}{[\![\{?\}]\!] = \langle \mathtt{DYN}, 0\rangle}$$

EvTransformArr

$$\frac{}{[\![\{\, T_1 \rightarrow\ T_2\}]\!] = \langle \mathtt{ARROW}, \langle [\![\{\, T_1\}]\!], [\![\{\, T_2\}]\!]\rangle\rangle}$$

EvTransformProd

$$\frac{}{[\![\{\, T_1 \rightarrow\ T_2\}]\!] = \langle \mathtt{PRODUCT}, \langle [\![\{\, T_1\}]\!], [\![\{\, T_2\}]\!]\rangle\rangle}$$

Translation: Evidence

$\boxed{[\![v]\!] = u}$ *(CPS Translation of Closed Values)*

ValTransformBool

$$\frac{}{[\![b]\!] = \langle \mathtt{DYN}, b\rangle}$$

ValTransformNum

$$\frac{}{[\![n]\!] = \langle \mathtt{DYN}, n\rangle}$$

ValTransformFun

$$\frac{[\![e]\!]\, c = t}{[\![\lambda x : T.\, e]\!] = \langle \mathtt{DYN}, \lambda x\ c.\, t\rangle}$$

ValTransformPair

$$\frac{}{[\![\langle v_1, v_2\rangle]\!] = \langle \mathtt{DYN}, \langle [\![v_1]\!], [\![v_2]\!]\rangle\rangle}$$

ValTransformEv

$$\frac{[\![r]\!] = \langle \mathtt{DYN}, u\rangle}{[\![\varepsilon\, r]\!] = \langle [\![\varepsilon]\!], u\rangle}$$

Translation: Evidence

Lemma 5.3 (Value and Expression Translations Match)**.** *Let* $[[v]]$ *be an irreducible term. Then, for any* $[[k]]$, $[[v]]$, $[[[|v|]k --> *k[[|v|]]]]$.

Proof. By induction on $[[v]]$.

- Case $[[v]] = [[b]]$, $[[v]] = [[n]]$, or $[[v]] = [[\ x : T.e]]$: trivial.
- Case $[[v]] = [[(v1, v2)]]$. So $[[[|(v1, v2)|]k ==== [|v1|](\ x1.[|v2|](\ x2.k[(DYN, (x1, x2))]))]]$, which, by our hypothesis, reduces to $[[t1-- > *(\ x1.(\ x2.k[(DYN, (x1, x2))])[[|v2|]])[[|v1|]]]]$, which we can then reduce to $[[k[(DYN, ([|v1|], [|v2|]))]]]$.
- Case $[[v]] = [[epr]]$. Since all raw irreducibles are themselves irreducible, our inductive hypothesis gives that $[[[|r|](\ x.letx1 := pi1xinletx2 := pi2xinMEET[[|ep|], x1, (.k[(y, x2)])])]]$ steps to $[[(\ x.letx1 := pi1xinletx2 := pi2xinMEET[[|ep|], x1, (.k[(y, x2)])])[[|r|]]]]$. By Lemma 5.2, $[[[|r|]]]$ is of the form $[[(DYN, u)]]$ for some $[[u]]$. So we can then β-reduce and apply the let-substitutions to reach $[[MEET[[|ep|], DYN, u]]]$. By Lemma 5.1, this steps to $[[([|ep|], u)]]$. By the rule TransformEv, this means that $[[[|epr|]]]$ also steps to this value.

  TODO

$\square$

Lemma 5.4 (Translation Commutes With Substitution)**.** $[[[|[x| => v]e|]k -- > *[x| => [|v|]][|e|]k]]$.

Proof. Follows from straightforward induction on $[[e]]$, combined with Lemma 5.3 for the case where $[[e]] = [[x]]$. $\square$

Theorem 5.1 (Weak Simulation)**.** *If* $[[e1 -- > e2]]$*, then for all* $[[k]]$, $[[[|e1|]k]] \equiv [[[|e2|]k]]$.

Proof. We perform induction on the derivation tree of $[[e1 -- > e2]]$.

- RedIfTrue: then $[[e1]] = [[iftruethene2elsee3]]$. The translation $[[[|true|]k' ==== k'[(DYN, true)]]]$ for any $[[k']]$, so $[[[|iftruethene2elsee3|]k]]$ is $[[(\ x0.letx := pi2x0inifx([|e2|]k)([|e3|]k))[(I$ We can β-reduce to get $[[letx := pi2(DYN, true)inifx[|e2|]k[|e3|]k]]$, and we can substitute $[[true]]$ for $[[x]]$ and reduce the **if** to get $[[[|e2|]k]]$.
- RedIfFalse: symmetric to RedIfTrue
- RedIfEv: $[[e1]] = [[ifepbthene2'elsee3']]$. We know that $[[[|b|]k' ==== k'[(DYN, b)]]]$, so $[[[|epb|]k'' ==== (.letx1 := pi1xinletx2 := pi2xinMEET[[|ep|], x1, (.k''[(y, x2)])])[(DYN, b)]]]$. We can β-reduce, and substitute with the let-expressions, to get $[[(.MEET[[|ep|], DYN, (.k''[(y, b)])])]]$. However, $[[ep/\ \ <<?>>]] = [[<<?>>]]$, so by Lemma 5.1 this steps to $[[k''[([|ep|], b)]]]$. Since the translation of **if** ignores any evidence in the condition, we can use the same reasoning from RedIfTrue to show that it steps to $[[e2]]$ if $[[b]]$ is true, and $[[e3]]$ if $[[b]]$ is false.

- RedApp: then $[[e1]] = [[(x : T.e')v]]$ and $[[e2]] = [[[x| => v]e']]$. Let $[[([|ep|], u)]] = [[[|v|]]]$ (by Lemma 5.2). If we apply Lemma 5.3, we can see that $[[[|(x : T.e')v|]k]]$ steps to $[[(x1x2.lety1 := pi1x1in, , ,)[(DYN, (xc.[|e'|]c)), ([|ep|], u)]]]$. We can β-reduce and apply the let-substitutions to then step to
$[[DOM[DYN, 1'.COD[DYN, 1''.MEET[y1', [|ep|], (y3.(xc.[|e'|]c)[(y3, u), (3.letz3' := pi1z3inletz3'' := pi2z3inMEET[y1'', z3', (4.k[(z4, z3'')])])])]]]]]$. By applying Lemma 5.1 for $[[DOM]]$, $[[COD]]$ and $[[MEET]]$ of $[[?]]$ respectively, we can step to
$[[(xc.[|e'|]c)[([|ep|], u), (3.letz3' := pi1z3inletz3'' := pi2z3inMEET[DYN, z3', (4.k[(z4, z3'')])])]]]$.
This then β-reduces to
$[[[x| => ([|ep|], u)][|e'|](3.letz3' := pi1z3inletz3'' := pi2z3inMEET[DYN, z3', (4.k[(z4, z3'')])])]]$.
But, then, by Lemma 5.1 and η-equivalence, this is equivalent to $[[[x| => ([|ep|], u)][|e'|]k]]$.
But we know that this is $[[[x| => [|v|]][|e'|]k]]$ Finally, Lemma 5.4 gives us that this is equivalent to $[[[|[x| => v]e'|]k]]$.

- RedAppEv: then $[[e1]] = [[ep1(x : T.e')ep2v]]$ and $[[e2]] = [[codep1([x| => (domep1/ ep2)v]e')]]$.
Let $[[([|ep2|], u)]] = [[[|v|]]]$ (by Lemma 5.2). If we apply Lemma 5.3, we can see that $[[[|ep1(x : T.e')ep2v|]k]]$ steps to
$[[(x1x2.lety1 := pi1x1in, , ,)[([|ep1|], (xc.[|e'|]c)), ([|ep2|], u)]]]$. We can β-reduce and apply the let-substitutions to then step to
$[[DOM[[|ep1|], 1'.COD[[|ep1|], 1''.MEET[y1', [|ep2|], (y3.(xc.[|e'|]c)[(y3, u), (3.letz3' := pi1z3inletz3'' := pi2z3inMEET[y1'', z3', (4.k[(z4, z3'')])])])]]]]]$. By applying Lemma 5.1 for $[[DOM]]$, $[[COD]]$ and $[[MEET]]$ of $[[?]]$ respectively, we can step to
$[[(xc.[|e'|]c)[([|domep1/2|], u), (3.letz3' := pi1z3inletz3'' := pi2z3inMEET[[|codep1|], z3', (4.k[(z4, z3'')])$
This then β-reduces to
$[[[x| => ([|domep1/2|], u)][|e'|](3.letz3' := pi1z3inletz3'' := pi2z3inMEET[[|codep1|], z3', (4.k[(z4, z3'')])$
But, by the rule TransformEv, this is α-equivalent to $[[[|codep1([x| => (domep1/ ep2)v]e')|]k]]$, giving us our result.

□

6 Incorrectness