An Equivalence-Preserving CPS Translation via Multi-Language Semantics (Technical Appendix)

Amal Ahmed
Indiana University
amal@cs.indiana.edu

Matthias Blume
Google
blume@google.com

This appendix contains an extended discussion section (Section 1), an extended section on related work (Section 2), the complete set of technical definitions (Sections 3 to 7), and the proofs (Sections 8 to 13). In particular, in Section 1.3, we adopt System F without the CPS restriction as our target language and present the changes needed—to the back-translation and the proofs—to establish full abstraction in this setting.

Contents

1 Discussion and Future Work

1.1 Supporting Additional Language Features

For this paper, we chose simply-typed λ-calculus and System F as our source and target languages so that we could highlight the main ideas underlying our proof technique, in particular, the use of multi-language semantics and how back-translation can leverage partial evaluation given the right type translation. We now sketch how to extend our theorem and its proof to source and target langauges with more advanced features.

Nontermination If the source and target language have non-terminating terms, then we are faced with two difficulties: ensuring well-foundedness of the back-translation (which affects Lemma 10.1) and proving the continuation-shuffling lemma (Lemma 9.2).

To address the first problem, ensuring well-foundedness of the back-translation relation, we add a new case wherever back-translation performs computation steps: if the term to be partially evaluated is non-terminating (i.e., contextually equivalent to a term that diverges), simply make its translation a non-terminating source term of appropriate type. Notice that this definition does not yield an algorithm for back-translation but merely a relation. However, since back-translation is merely a proof device, it does not need to be an algorithm.

Specifically, let us add divergence to the source and target languages by adding the *non-value* divergent term Ω to λ^{S} as a constant, and similarly adding $\boldsymbol{\Omega}$ to λ^{T}. The grammar for evaluation contexts remains unchanged. We add the following reduction rules to the operational semantics for λ^{S} and λ^{T}, respectively:

$$\Omega \longmapsto \Omega \qquad\qquad \boldsymbol{\Omega} \longmapsto \boldsymbol{\Omega}$$

We add typing rules for Ω and $\boldsymbol{\Omega}$, such that the divergent terms have any type:

$$\frac{}{\Gamma \vdash \Omega : \sigma} \qquad\qquad \frac{}{\Delta;\Gamma \vdash \boldsymbol{\Omega} : \boldsymbol{\tau}}$$

We extend the CPS translation with the following:

$$\frac{}{\Gamma \vdash \Omega : \sigma \rightsquigarrow \boldsymbol{\lambda\,[\alpha]\,(k : \sigma^{+} \to \alpha).\,k\;\Omega}}$$

We add an additional rule to the back-translation in Figure 24 (on page 32):

$$\frac{}{\cdot;\Gamma \vdash^{+} \boldsymbol{\Omega} : \sigma^{+} \twoheadrightarrow \Omega}$$

In addition, there are two rules in Figure 24 (see page 32) where changes are needed: the $\mathbf{let\ x = \pi_i v\ in\ e}$ rule marked (†) and the $\mathbf{v_1\,[\tau']\,v_2}$ rule marked (‡). These are the rules where the back-translation performs computation steps and well-foundedness relies on the fact that the term being back-translated in the premise reduces to a value in fewer steps than the term in the conclusion.

- We add an additional premise to the rule in Figure 24 marked (†) so that we back-translate only if the partially reduced term in the premise does not diverge. The new rule is as follows:

$$\frac{\cdot;\Gamma \vdash \mathbf{v} : \boldsymbol{\tau_1 \times \tau_2} \qquad \mathbf{v} = (\mathbf{v_1}, \mathbf{v_2}) \qquad \cdot;\Gamma \vdash \mathbf{e}[\mathbf{v}_i/\mathbf{x}] \not\approx^{ctx}_{\mathrm{ST}} \boldsymbol{\Omega} : \sigma^{+} \qquad \cdot;\Gamma \vdash^{+} \mathbf{e}[\mathbf{v}_i/\mathbf{x}] : \sigma^{+} \twoheadrightarrow e}{\cdot;\Gamma \vdash^{+} \mathbf{let\ x = \pi_i v\ in\ e} : \sigma^{+} \twoheadrightarrow e}$$

  Also, we add an additional rule the applies when the partially reduced term in the premise diverges:

$$\frac{\cdot;\Gamma \vdash \mathbf{v} : \boldsymbol{\tau_1 \times \tau_2} \qquad \mathbf{v} = (\mathbf{v_1}, \mathbf{v_2}) \qquad \cdot;\Gamma \vdash \mathbf{e}[\mathbf{v}_i/\mathbf{x}] \approx^{ctx}_{\mathrm{ST}} \boldsymbol{\Omega} : \sigma^{+}}{\cdot;\Gamma \vdash^{+} \mathbf{let\ x = \pi_i v\ in\ e} : \sigma^{+} \twoheadrightarrow \Omega}$$

  Since $\mathbf{e}[\mathbf{v}_i/\mathbf{x}]$ diverges, we back-translate the entire expression to the λ^{S} term Ω.

- We similarly add an additional premise to the rule in Figure 24 marked (‡) so that we back-translate only if the partially reduced term in the premise does not diverge. The new rule is as follows:

$$\frac{\begin{array}{c}\cdot;\Gamma \vdash \mathbf{v}_1 : \forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau}_2 \rightarrow \boldsymbol{\tau} \qquad \cdot \vdash \boldsymbol{\tau}' \qquad \cdot;\Gamma \vdash \mathbf{v}_2 : \boldsymbol{\tau}_2[\boldsymbol{\tau}'/\boldsymbol{\alpha}] \\ \sigma^+ = \boldsymbol{\tau}[\boldsymbol{\tau}'/\boldsymbol{\alpha}] \qquad \mathbf{v}_1 = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{z} : \boldsymbol{\tau}_2).\,\mathbf{e} \\ \cdot;\Gamma \vdash \mathbf{e}[\boldsymbol{\tau}'/\boldsymbol{\alpha}][\mathbf{v}_2/\mathbf{z}] \not\approx^{ctx}_{\mathrm{ST}} \boldsymbol{\Omega} : \sigma^+ \qquad \cdot;\Gamma \vdash^+ \mathbf{e}[\boldsymbol{\tau}'/\boldsymbol{\alpha}][\mathbf{v}_2/\mathbf{z}] : \sigma^+ \twoheadrightarrow \mathsf{e}\end{array}}{\cdot;\Gamma \vdash^+ \mathbf{v}_1\ [\boldsymbol{\tau}']\ \mathbf{v}_2 : \sigma^+ \twoheadrightarrow \mathsf{e}}\ (\nexists \sigma_1.\ {\sigma_1}^+ = \forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau}_2 \rightarrow \boldsymbol{\tau})$$

  Also, we add an additional rule that applies when the partially reduced term in the premise diverges:

$$\frac{\begin{array}{c}\cdot;\Gamma \vdash \mathbf{v}_1 : \forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau}_2 \rightarrow \boldsymbol{\tau} \qquad \cdot \vdash \boldsymbol{\tau}' \qquad \cdot;\Gamma \vdash \mathbf{v}_2 : \boldsymbol{\tau}_2[\boldsymbol{\tau}'/\boldsymbol{\alpha}] \\ \sigma^+ = \boldsymbol{\tau}[\boldsymbol{\tau}'/\boldsymbol{\alpha}] \qquad \mathbf{v}_1 = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{z} : \boldsymbol{\tau}_2).\,\mathbf{e} \qquad \cdot;\Gamma \vdash \mathbf{e}[\boldsymbol{\tau}'/\boldsymbol{\alpha}][\mathbf{v}_2/\mathbf{z}] \approx^{ctx}_{\mathrm{ST}} \boldsymbol{\Omega} : \sigma^+\end{array}}{\cdot;\Gamma \vdash^+ \mathbf{v}_1\ [\boldsymbol{\tau}']\ \mathbf{v}_2 : \sigma^+ \twoheadrightarrow \Omega}\ (\nexists \sigma_1.\ {\sigma_1}^+ = \forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau}_2 \rightarrow \boldsymbol{\tau})$$

  Since $\mathbf{e}[\boldsymbol{\tau}'/\boldsymbol{\alpha}][\mathbf{v}_2/\mathbf{z}]$ diverges, we back-translate the entire expression to the λ^{S} term Ω.

The above changes to the back-translation rules in Figure 24 yield a well-founded back-translation relation despite the presence of non-termination. The additional rules above simply amount to additional cases of the proof of Lemma 10.1. In fact, these additional cases are trivial to prove: since we back-translate a diverging λ^{ST} term to a diverging λ^{S} term, they are clearly equivalent as required by part (2) of Lemma 10.1.

The second problem, proving the continuation-shuffling lemma, is more difficult to deal with: in the presence of non-termination, Lemma 9.2 cannot be proved by parametricity alone; parametricity only gives us an approximation relation and not an equivalence relation. However, there is an alternative, syntactic proof for the continuation-shuffling lemma in this case. The details are beyond the scope of the current work; we will present that result in a future paper.

Recursive types Recursive types give rise to non-termination, so everything said above applies here as well. In particular, though, recursive types make it somewhat trickier to define the semantics of our multi-language boundary terms. However, the technique for defining "wrapper" terms in our full abstraction proof for closure conversion [4] can be adapted to deal with this issue. Also, in the presence of recursive types we will need to switch to a step-indexed logical relation [2] for λ^{ST} to ensure well-foundedness of the logical relation. Thus, all parts of the proof that make use of the λ^{ST} logical relation will become more involved. Again, we plan to report on this result in a future paper.

Polymorphism Adding polymorphism to the source language is not difficult to deal with. The main idea is for boundary terms to not "peek under" abstraction barriers but to simply leave abstract things abstract. We have successfully applied this idea before in our work on fully abstract closure conversion [4] and do not foresee any difficulties when adapting it to the case of CPS translation.

From System F to TAL Our longer-term goal is to show that a compiler from System F (with recursion, and later, with state) to some typed assembly language is equivalence preserving. Thus far we have considered two phases of compilation, namely CPS (this paper) and typed closure conversion [4], where the type translations were precisely the key to ensuring that the target terms that compiled code interacts with are sufficiently well behaved. Thus, the type translations were the key to proving that the translations were equivalence preserving. To formulate an equivalence-preserving translation from CPS-and-closure-converted code down to assembly, we will need a target-level type system rich enough to be able to express invariants about local state, separation of state, and so on. We believe that an assembly language based on Hoare Type Theory [32] can provide the features needed for imposing the necessary well-behavedness constraints on target contexts.

1.2 Compiler Correctness Proofs

Let us compare the statement of "CPS is Semantics Preserving" (Lemma 9.4) to roughly what one would expect in a semantics-preservation (compiler correctness) proof that uses a cross-language logical relation (e.g., [7, 8, 16]) relating source terms of type σ to target terms of type σ^+ (though, relating to $\sigma^\div$ is a valid choice as well). In the absence of a combined language, the statement of the lemma there cannot mention $\mathcal{ST}$. Instead it says that the source term e must be related to its translation $\mathbf{v}$ in the cross-language relation at the type σ (which would mean that $\mathbf{v}$ would have type σ^+).

That means that to understand the definition of "relatedness" at each type, one would have to have a precise understanding of the cross-language relation. In the presence of features like recursive types (and eventually state), these logical relations get rather complicated, for instance using step-indexing and/or biorthogonality (e.g., [7, 8]), or advanced denotational models. Thus, it can be difficult for someone who simply wants to understand the statement of the semantics-preservation theorem, and not the mathematical machinery underlying the logical relation, to decipher exactly what that statement says. On the other hand, with the multi-language approach, to understand the statement of the theorem, one simply has to examine the reduction rules for boundaries. The details of the λ^{ST} logical relation are not important at this level; all one needs to know is that it is sound with respect to contextual equivalence.

Finally, defining the multi-language logical relation seems more straightforward than defining the cross-language logical relation. The multi-language logical relation is simply a combination of the logical relations for source and target; the desired relationship between source and target semantics is specified separately by defining the dynamic semantics for boundaries. The cross-language logical relation, on the other hand, must simultaneously provide a proof method *and* specify the desired relationship between source and target terms.

1.3 An Alternate Target Language (without CPS Restriction)

For this paper, following Morrisett *et al.* [31]—and in the spirit of compilers like SML/NJ [6, 39], Rabbit [41], Orbit [25], and more recently, Kennedy's SML.NET compiler [24]—we picked a target language that syntactically enforces continuation-passing style. This also allowed us to demonstrate that our proof technique works when neither the source language nor the target is a subset of the other. The reader may, however, have wondered if our CPS translation is fully abstract when the target language is full System F (i.e., without the CPS restriction). We now show that this is, in fact, the case.

Let λ^{F} denote System F without any syntactic restriction on terms. We wish to show that our CPS translation from λ^{S} to λ^{F}, utilizing locally polymorphic answer types, is equivalence preserving. Now, let λ^{A} denote System F with terms in A-normal form. We are interested in λ^{A} because it is convenient to do the proof in two steps: we first show that CPS translation from λ^{S} to λ^{A} is equivalence-preserving and then show that if two terms are equivalent in λ^{A}, then they are equivalent in λ^{F}.

Figures 1 and 2 present the syntax and operational semantics for λ^{A} (System F in A-normal form) and λ^{F} (full System F without syntactic restrictions), respectively. We use a **bold orange font with serifs** for λ^{A} and a **bold green font with serifs** for λ^{F}. We omit the typing rules for λ^{A} and λ^{F} as these are entirely standard, as is the syntax of contexts.

The operational semantics for λ^{A} is slightly complicated due to the well-known fact that A-normal form is not preserved under β-reduction. For instance, consider the λ^{A} term:

$$\mathbf{let}\ \mathbf{x} = (\lambda \mathbf{y}.\ \mathbf{let}\ \mathbf{z} = \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{in}\ \mathbf{x}_3)\ \mathbf{x}_4\ \mathbf{in}\ \mathbf{e}.$$

Now, naive β-reduction produces

$$\mathbf{let}\ \mathbf{x} = (\mathbf{let}\ \mathbf{z} = \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{in}\ \mathbf{x}_3)\ \mathbf{in}\ \mathbf{e}.$$

which is not in A-normal form. The way to fix this is to define a more complex notion of β-reduction that re-normalizes $\mathbf{let}$ constructs, in this case producing the normal form

$$\mathbf{let}\ \mathbf{z} = \mathbf{x}_1\ \mathbf{x}_2\ \mathbf{in}\ (\mathbf{let}\ \mathbf{x} = \mathbf{x}_3\ \mathbf{in}\ \mathbf{e}).$$

We define a function to do this re-normalization (middle of Figure 1): when applied to a λ^{A} term $\mathbf{e}$ surrounded by the context $\mathbf{E}$, the normalization function, written $(\,\mathbf{E}[\mathbf{e}]\,)^{\#}$, produces a valid λ^{A} term—that is, a term in A-normal form. With this re-normalization function in hand, we can define the λ^{A} reduction rules (bottom of Figure 1). These are all standard except for the rule that performs β-reduction in a $\mathbf{let}$-bound position; the latter requires that we re-normalize the result of the naive β-reduction.

Types	τ	$::=$	$\mathbf{bool} \mid \tau_1 \times \tau_2 \mid \alpha \mid \forall\,[\alpha].\tau_1 \rightarrow \tau_2$
Values	v	$::=$	$x \mid \mathbf{true} \mid \mathbf{false} \mid (v_1, v_2) \mid \lambda\,[\alpha]\,(x:\tau).\,e$
Computations	r	$::=$	$v \mid \pi_i v \mid v_1\,[\tau]\,v_2$
Terms	e	$::=$	$r \mid \mathbf{let}\ x = r\ \mathbf{in}\ e \mid \mathbf{if}\ v\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2$
Eval. Contexts	E	$::=$	$\mathbf{let}\ x = [\cdot]_A\ \mathbf{in}\ e$

$\boxed{\text{Re-normalization function: } (\,E[e]\,)^{\#}}$

$$
\begin{aligned}
(\,E[r]\,)^{\#} &= E[r] \\
(\,E[\mathbf{let}\ x = r\ \mathbf{in}\ e]\,)^{\#} &= \mathbf{let}\ x = r\ \mathbf{in}\ (\,E[e]\,)^{\#} \\
(\,E[\mathbf{if}\ v\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2]\,)^{\#} &= \mathbf{if}\ v\ \mathbf{then}\ (\,E[e_1]\,)^{\#}\ \mathbf{else}\ (\,E[e_2]\,)^{\#}
\end{aligned}
$$

$\boxed{e \longmapsto e'}$

$$
\begin{aligned}
\pi_i(v_1, v_2) &\longmapsto v_i \\
E[\pi_i(v_1, v_2)] &\longmapsto E[v_i] \\
(\lambda\,[\alpha]\,(x:\tau_1).\,e)\ [\tau]\ v &\longmapsto e[\tau/\alpha][v/x] \\
E[(\lambda\,[\alpha]\,(x:\tau_1).\,e)\ [\tau]\ v] &\longmapsto (\,E[e[\tau/\alpha][v/x]]\,)^{\#} \\
\mathbf{let}\ x = v\ \mathbf{in}\ e &\longmapsto e[v/x] \\
\mathbf{if}\ \mathbf{true}\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 &\longmapsto e_1 \\
\mathbf{if}\ \mathbf{false}\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 &\longmapsto e_2
\end{aligned}
$$

Figure 1: Alternate Intermediate Target Language: System F in A-normal Form (λ^{A})

The λ^{F} operational semantics is completely standard (see bottom of Figure 2). The statements of contextual equivalence and ciu equivalence for both λ^{A} and λ^{F} are exactly as for λ^{T} modulo color, which means that these statements involve λ^{A} (respectively λ^{F}) contexts, types, terms, and operational semantics. As for λ^{T}, it is easy to prove that contextual approximation implies ciu approximation.

Lemma 1.1 **(λ^{A}: Contextual Approx Implies CIU Approx)**
If $\Delta;\Gamma \vdash e_1 \precsim^{ctx}_{\mathrm{A}} e_2 : \tau$ *then* $\Delta;\Gamma \vdash e_1 \precsim^{ciu}_{\mathrm{A}} e_2 : \tau$.

Lemma 1.2 **(λ^{F}: Contextual Approx Implies CIU Approx)**
If $\Delta;\Gamma \vdash e_1 \precsim^{ctx}_{\mathrm{F}} e_2 : \tau$ *then* $\Delta;\Gamma \vdash e_1 \precsim^{ciu}_{\mathrm{F}} e_2 : \tau$.

Note that λ^{A} is a subset of λ^{F} (assuming that we have some standard syntactic sugar defined for λ^{F}). Therefore, whenever necessary, we will treat **orange** λ^{A} terms as if they were **green** λ^{F} terms. That is, we will skip a proper translation from **orange** terms to **green** terms since, other than the color change, the translation is simply the identity (again assuming that we have some obvious syntactic sugar for λ^{F}).

Next, we must define typed CPS translation from λ^{S} to λ^{F}. This is exactly the same as the CPS translation to λ^{T} (see Figures 16 and 17), except that the **red** target terms should now be **green**. Note that we can also color the target terms **orange** to highlight the fact that the translation only utilizes terms in the λ^{A} subset of λ^{F}.

We wish to prove that the CPS translation from λ^{S} to λ^{F} is equivalence preserving, that is, the following theorem.

Theorem 1.3
Let $\Gamma \vdash e_1 : \sigma \rightsquigarrow v_1$ *and* $\Gamma \vdash e_2 : \sigma \rightsquigarrow v_2$*. If* $\Gamma \vdash e_1 \approx^{ctx}_{\mathrm{S}} e_2 : \sigma$ *then* $\cdot;\Gamma^{+} \vdash v_1 \approx^{ctx}_{\mathrm{F}} v_2 : \sigma^{\div}$.

$$\begin{array}{llll}
\textit{Types} & \tau & ::= & \mathbf{bool} \mid \tau_1 \times \tau_2 \mid \alpha \mid \forall\,[\alpha].\tau_1 \rightarrow \tau_2 \\
\textit{Expressions} & e & ::= & x \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{if}\ e\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 \mid (e_1, e_2) \mid \pi_i e \mid \\
 & & & \lambda\,[\alpha]\,(x : \tau).\,e \mid e_1\,[\tau]\,e_2 \\
\textit{Values} & v & ::= & x \mid \mathbf{true} \mid \mathbf{false} \mid (v_1, v_2) \mid \lambda\,[\alpha]\,(x : \tau).\,e \\
\textit{Eval. Contexts} & E & ::= & [\cdot]_F \mid \mathbf{if}\ E\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 \mid (E, e) \mid (v, E) \mid \pi_i E \mid \\
 & & & E\,[\tau]\,e \mid v\,[\tau]\,E
\end{array}$$

$$\boxed{e \longmapsto e'}$$

$$\begin{array}{rcl}
\mathbf{if}\ \mathbf{true}\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 & \longmapsto & e_1 \\
\mathbf{if}\ \mathbf{false}\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 & \longmapsto & e_2 \\
\pi_i(v_1, v_2) & \longmapsto & v_i \\
(\lambda\,[\alpha]\,(x : \tau_1).\,e)\,[\tau]\,v & \longmapsto & e[\tau/\alpha][v/x]
\end{array}$$

$$\frac{e \longmapsto e'}{E[e] \longmapsto E[e']}$$

Figure 2: Alternate Target Language: System F (λ^{F})

In the above theorem, the λ^{F} terms v_1 and v_2 can also be written as λ^{A} terms v_1 and v_2 respectively, i.e., the respective terms are identical modulo color. We decompose the proof of Theorem 1.3 into two parts:

A. if $\Gamma \vdash e_1 \approx^{ctx}_{\mathrm{S}} e_2 : \sigma$ then $\cdot; \Gamma^+ \vdash v_1 \approx^{ctx}_{\mathrm{A}} v_2 : \sigma^{\div}$, where $\Gamma \vdash e_1 : \sigma \rightsquigarrow v_1$ and $\Gamma \vdash e_2 : \sigma \rightsquigarrow v_2$; and

B. if $\cdot; \Gamma^+ \vdash v_1 \approx^{ctx}_{\mathrm{A}} v_2 : \sigma^{\div}$, then $\cdot; \Gamma^+ \vdash v_1 \approx^{ctx}_{\mathrm{F}} v_2 : \sigma^{\div}$, where $v_1 = v_1$ and $v_2 = v_2$ modulo color.

To prove part (A)—that CPS translation from λ^{S} to λ^{A} is equivalence preserving—we proceed much as we did for CPS translation to λ^{T}. That is, we define a multi-language semantics that allows λ^{S} terms to interoperate with λ^{A} terms, and then split the proof into three parts as before. The proof of parts (2) and (3) are essentially unchanged. The proof of part (1) requires the only interesting change, in particular, because the back-translation from λ^{A} to λ^{S} is more involved than the back-translation from λ^{T} to λ^{S}. We present this part of the proof and the modified back-translation in Section 1.3.1.

The proof of part (B)—that if two terms are equivalent in λ^{A} then they are equivalent in λ^{F}—is fairly straightforward as we shall see. Since λ^{A} is a subset of λ^{F}, we will not need to make use of a multi-language semantics. We will need a back-translation from λ^{F} to λ^{A}, but this back-translation is just A-normalization. We describe this part of the proof in Section 1.3.2.

1.3.1 Proof: CPS Translation from λ^{S} to λ^{A} Preserves Equivalence

Our goal in this section is to prove that

(Part A) if $\Gamma \vdash e_1 \approx^{ctx}_{\mathrm{S}} e_2 : \sigma$ then $\cdot; \Gamma^+ \vdash v_1 \approx^{ctx}_{\mathrm{A}} v_2 : \sigma^{\div}$, where $\Gamma \vdash e_1 : \sigma \rightsquigarrow v_1$ and $\Gamma \vdash e_2 : \sigma \rightsquigarrow v_2$.

We start by defining the language λ^{SA} which permits λ^{S} terms and λ^{A} terms to interoperate. Figure 3 presents the λ^{SA} multi-language system which is essentially the same as λ^{ST} except for a few minor changes. Contextual and ciu equivalence for λ^{SA} are defined exactly as for λ^{ST}, and we can similarly prove that contextual approximation implies ciu approximation in λ^{SA}. We must also define a logical relation ($\precsim^{log}_{\mathrm{SA}}$, $\approx^{log}_{\mathrm{SA}}$) for λ^{SA}, but the definition of this is identical to the logical relation ($\precsim^{log}_{\mathrm{ST}}$, $\approx^{log}_{\mathrm{ST}}$) for λ^{ST}, except we change all occurrences of **red** to orange. The proof of the fundamental property for $\approx^{log}_{\mathrm{SA}}$ is almost exactly the same as the corresponding proof for $\approx^{log}_{\mathrm{ST}}$.

$$
\begin{array}{llcl}
\textit{Types} & \varphi & ::= & \sigma \mid \boldsymbol{\tau} \\
\textit{Terms} & \mathsf{e} & ::= & \ldots \mid {}^{\sigma}\mathcal{SA}\,\mathbf{e} \\
 & \mathbf{r} & ::= & \ldots \mid \mathcal{AS}^{\,\sigma}\,\mathsf{e} \\
 & \mathbf{e} & ::= & \ldots \\
 & e & ::= & \mathsf{e} \mid \mathbf{e} \\
\textit{Values} & v & ::= & \mathsf{v} \mid \mathbf{v} \\
\textit{Eval. Contexts} & \mathsf{E} & ::= & \ldots \mid {}^{\sigma}\mathcal{SA}\,\mathbf{E} \\
 & \mathbf{E} & ::= & \ldots \mid \mathcal{AS}^{\,\sigma}\,\mathsf{E} \\
 & E & ::= & \mathsf{E} \mid \mathbf{E}
\end{array}
$$

$\boxed{e \longmapsto e'}$

$$
\begin{array}{rcl}
{}^{\mathsf{bool}}\mathcal{SA}\,\mathbf{true} & \longmapsto & \mathsf{true} \\
{}^{\mathsf{bool}}\mathcal{SA}\,\mathbf{false} & \longmapsto & \mathsf{false} \\
{}^{\sigma_1 \to \sigma_2}\mathcal{SA}\,\mathbf{v} & \longmapsto & \lambda \mathsf{x} : \sigma_1.\ {}^{\sigma_2}\mathcal{SA}\,(\mathbf{let\ z} = (\mathcal{AS}^{\,\sigma_1}\,\mathsf{x})\ \mathbf{in}\ (\mathbf{v}\ [\sigma_2{}^{+}]\ (\mathbf{z}, \mathbf{id})))
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{AS}^{\,\mathsf{bool}}\,\mathsf{true} & \longmapsto & \mathbf{true} \\
\mathcal{AS}^{\,\mathsf{bool}}\,\mathsf{false} & \longmapsto & \mathbf{false} \\
\mathcal{AS}^{\,\sigma_1 \to \sigma_2}\,\mathsf{v} & \longmapsto & \lambda\,[\alpha]\ ((\mathbf{x}, \mathbf{k}) : \sigma_1{}^{+} \times (\sigma_2{}^{+} \to \alpha)). \\
 & & \quad \mathbf{let\ z} = (\mathcal{AS}^{\,\sigma_2}\,(\mathsf{v}\ ({}^{\sigma_1}\mathcal{SA}\,\mathbf{x})))\ \mathbf{in\ k\ z}
\end{array}
$$

$$
\frac{e \ \longmapsto\ e'}{E[e]\ \longmapsto\ E[e']}
$$

$$
\begin{array}{llcl}
\textit{Type Environments} & \Delta & ::= & \cdot \mid \Delta, \boldsymbol{\alpha} \\
\textit{Value Environments} & \Gamma & ::= & \cdot \mid \Gamma, \mathsf{x} : \sigma \mid \Gamma, \mathbf{x} : \boldsymbol{\tau}
\end{array}
$$

$\boxed{\Delta; \Gamma \vdash e : \varphi}$

$$
\ldots \qquad \frac{\Delta; \Gamma \vdash \mathbf{e} : \sigma^{+}}{\Delta; \Gamma \vdash {}^{\sigma}\mathcal{SA}\,\mathbf{e} : \sigma} \qquad\qquad \frac{\Delta; \Gamma \vdash \mathsf{e} : \sigma}{\Delta; \Gamma \vdash \mathcal{AS}^{\,\sigma}\,\mathsf{e} : \sigma^{+}}
$$

$$
\begin{array}{llcl}
\textit{Contexts} & \mathsf{C} & ::= & \ldots \mid {}^{\sigma}\mathcal{SA}\,\mathbf{C} \\
 & \mathbf{C} & ::= & \ldots \mid \mathcal{AS}^{\,\sigma}\,\mathsf{C} \\
 & C & ::= & \mathsf{C} \mid \mathbf{C}
\end{array}
$$

Figure 3: λ^{SA}: Syntax, Dynamic Semantics, Static Semantics, and Contexts

Now, the proof of part (A), that CPS translation from λ^{S} to λ^{A} is equivalence preserving, can be decomposed into three parts:

1. if $\Gamma \vdash \mathsf{e}_1 \approx^{ctx}_{\mathrm{S}} \mathsf{e}_2 : \sigma$ then $\Gamma \vdash \mathsf{e}_1 \approx^{ctx}_{\mathrm{SA}} \mathsf{e}_2 : \sigma$;
2. if $\Gamma \vdash \mathsf{e}_1 \approx^{ctx}_{\mathrm{SA}} \mathsf{e}_2 : \sigma$ then $\cdot; \Gamma^{+} \vdash \mathbf{v}_1 \approx^{ctx}_{\mathrm{SA}} \mathbf{v}_2 : \sigma^{\div}$, where $\Gamma \vdash \mathsf{e}_1 : \sigma \rightsquigarrow \mathbf{v}_1$ and $\Gamma \vdash \mathsf{e}_2 : \sigma \rightsquigarrow \mathbf{v}_2$; and
3. if $\cdot; \Gamma^{+} \vdash \mathbf{v}_1 \approx^{ctx}_{\mathrm{SA}} \mathbf{v}_2 : \sigma^{\div}$ then $\cdot; \Gamma^{+} \vdash \mathbf{v}_1 \approx^{ctx}_{\mathrm{A}} \mathbf{v}_2 : \sigma^{\div}$.

Once we have proved (1), (2), and (3), the proof of part (A) is immediate.

The proofs of parts (2) and (3) are essentially the same as the corresponding proofs for the translation from λ^{S} to λ^{T} so we elide the details. As before, we need to prove the boundary cancellation and continuation shuffling lemmas; the proofs are again essentially as before.

For the proof of part (1), we again need to be able to "back-translate" an arbitrary λ^{SA} context C with a hole of type $(\cdot; \Gamma \vdash \sigma)$ to an equivalent λ^{S} context C. Such a context can simply be treated as an expression $\lambda \mathsf{x} : \sigma.\, C[\mathsf{x}]$ which has type $\sigma \to \mathsf{bool}$. Thus, we need to be able to "back-translate" λ^{SA} terms of source type.

$\boxed{\cdot;\Gamma \vdash e : \sigma \twoheadrightarrow \mathsf{e}}$ $\boxed{\cdot;\Gamma \vdash^{+} e : \sigma^{+} \twoheadrightarrow \mathsf{e}}$ and where $\Gamma^{\twoheadrightarrow}$ is defined as

where $\Gamma ::= \cdot \mid \Gamma, \mathsf{x} : \sigma \mid \mathbf{y} : \sigma^{+}$
and $\mathsf{e} \in \lambda^{\mathrm{S}}$ and $\Gamma^{\twoheadrightarrow} \vdash \mathsf{e} : \sigma$

$$\begin{aligned} (\cdot)^{\twoheadrightarrow} &= \cdot \\ (\Gamma, \mathsf{x} : \sigma)^{\twoheadrightarrow} &= \Gamma^{\twoheadrightarrow}, \mathsf{x} : \sigma \\ (\Gamma, \mathbf{y} : \sigma^{+})^{\twoheadrightarrow} &= \Gamma^{\twoheadrightarrow}, \mathsf{y} : \sigma \end{aligned}$$

$$\frac{\cdot \vdash \Gamma}{\cdot;\Gamma \vdash \mathsf{true} : \mathsf{bool} \twoheadrightarrow \mathsf{true}} \qquad \frac{\cdot \vdash \Gamma}{\cdot;\Gamma \vdash \mathsf{false} : \mathsf{bool} \twoheadrightarrow \mathsf{false}}$$

$$\frac{\cdot;\Gamma \vdash \mathsf{e} : \mathsf{bool} \twoheadrightarrow \mathsf{e}' \qquad \cdot;\Gamma \vdash \mathsf{e}_1 : \sigma \twoheadrightarrow \mathsf{e}_1' \qquad \cdot;\Gamma \vdash \mathsf{e}_2 : \sigma \twoheadrightarrow \mathsf{e}_2'}{\cdot;\Gamma \vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 : \sigma \twoheadrightarrow \mathsf{if}\ \mathsf{e}'\ \mathsf{then}\ \mathsf{e}_1'\ \mathsf{else}\ \mathsf{e}_2'} \qquad \frac{\cdot \vdash \Gamma \qquad \mathsf{x} : \sigma \in \Gamma}{\cdot;\Gamma \vdash \mathsf{x} : \sigma \twoheadrightarrow \mathsf{x}}$$

$$\frac{\cdot;\Gamma, \mathsf{x} : \sigma_1 \vdash \mathsf{e} : \sigma_2 \twoheadrightarrow \mathsf{e}'}{\cdot;\Gamma \vdash \lambda \mathsf{x} : \sigma_1 . \mathsf{e} : \sigma_1 \to \sigma_2 \twoheadrightarrow \lambda \mathsf{x} : \sigma_1 . \mathsf{e}'} \qquad \frac{\cdot;\Gamma \vdash \mathsf{e}_1 : \sigma_2 \to \sigma \twoheadrightarrow \mathsf{e}_1' \qquad \cdot;\Gamma \vdash \mathsf{e}_2 : \sigma_2 \twoheadrightarrow \mathsf{e}_2'}{\cdot;\Gamma \vdash \mathsf{e}_1\ \mathsf{e}_2 : \sigma \twoheadrightarrow \mathsf{e}_1'\ \mathsf{e}_2'} \qquad \frac{\cdot;\Gamma \vdash^{+} \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{e}}{\cdot;\Gamma \vdash {}^{\sigma}\mathcal{SA}\, \mathbf{e} : \sigma \twoheadrightarrow \mathsf{e}}$$

$$\frac{\cdot \vdash \Gamma}{\cdot;\Gamma \vdash^{+} \mathbf{true} : \mathsf{bool}^{+} \twoheadrightarrow \mathsf{true}} \qquad \frac{\cdot \vdash \Gamma}{\cdot;\Gamma \vdash^{+} \mathbf{false} : \mathsf{bool}^{+} \twoheadrightarrow \mathsf{false}} \qquad \frac{\cdot \vdash \Gamma \qquad \mathbf{y} : \sigma^{+} \in \Gamma}{\cdot;\Gamma \vdash^{+} \mathbf{y} : \sigma^{+} \twoheadrightarrow \mathsf{y}}$$

$$\frac{\cdot \vdash \Gamma \qquad \cdot;\Gamma, \mathbf{y} : {\sigma_1}^{+} \vdash^{+} \mathbf{e}[{\sigma_2}^{+}/\alpha][\mathbf{id}/\mathbf{k}] : {\sigma_2}^{+} \twoheadrightarrow \mathsf{e}}{\cdot;\Gamma \vdash^{+} \lambda\,[\alpha]\,((\mathbf{y},\mathbf{k}) : ({\sigma_1}^{+} \times ({\sigma_2}^{+} \to \alpha))).\, \mathbf{e} : \forall\,[\alpha].({\sigma_1}^{+} \times ({\sigma_2}^{+} \to \alpha)) \to \alpha \twoheadrightarrow \lambda \mathsf{y} : \sigma_1 . \mathsf{e}}$$

$$\frac{\cdot;\Gamma \vdash \mathbf{v} : \sigma^{+} \times \tau_2 \qquad \mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2) \qquad \cdot;\Gamma \vdash^{+} \mathbf{v}_1 : \sigma^{+} \twoheadrightarrow \mathsf{e}}{\cdot;\Gamma \vdash^{+} \pi_1 \mathbf{v} : \sigma^{+} \twoheadrightarrow \mathsf{e}}$$

$$\frac{\cdot;\Gamma \vdash \mathbf{v} : \tau_1 \times \sigma^{+} \qquad \mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2) \qquad \cdot;\Gamma \vdash^{+} \mathbf{v}_2 : \sigma^{+} \twoheadrightarrow \mathsf{e}}{\cdot;\Gamma \vdash^{+} \pi_2 \mathbf{v} : \sigma^{+} \twoheadrightarrow \mathsf{e}}$$

$$\frac{\begin{array}{c} \cdot;\Gamma \vdash \mathbf{v}_1 : \forall\,[\alpha].\tau_2 \to \tau \qquad \cdot \vdash \tau' \qquad \cdot;\Gamma \vdash \mathbf{v}_2 : \tau_2[\tau'/\alpha] \\ \sigma^{+} = \tau[\tau'/\alpha] \qquad \mathbf{v}_1 = \lambda\,[\alpha]\,(\mathbf{z} : \tau_2).\,\mathbf{e} \qquad \cdot;\Gamma \vdash^{+} \mathbf{e}[\tau'/\alpha][\mathbf{v}_2/\mathbf{z}] : \sigma^{+} \twoheadrightarrow \mathsf{e} \end{array}}{\cdot;\Gamma \vdash^{+} \mathbf{v}_1\ [\tau']\ \mathbf{v}_2 : \sigma^{+} \twoheadrightarrow \mathsf{e}}\ (\nexists \sigma_1 .\ {\sigma_1}^{+} = \forall\,[\alpha].\tau_2 \to \tau)$$

$$\frac{\begin{array}{c} \tau' = \sigma^{+} \qquad \mathbf{v}_2 = (\mathbf{v}_a, \lambda(\mathbf{z} : {\sigma_2}^{+}).\,\mathbf{e}_k) \\ \cdot;\Gamma \vdash^{+} \mathbf{v}_1 : (\sigma_1 \to \sigma_2)^{+} \twoheadrightarrow \mathsf{e}_1 \qquad \cdot;\Gamma \vdash^{+} \mathbf{v}_a : {\sigma_1}^{+} \twoheadrightarrow \mathsf{e}_a \qquad \cdot;\Gamma, \mathbf{z} : {\sigma_2}^{+} \vdash^{+} \mathbf{e}_k : \sigma^{+} \twoheadrightarrow \mathsf{e}_k \end{array}}{\cdot;\Gamma \vdash^{+} \mathbf{v}_1\ [\tau']\ \mathbf{v}_2 : \sigma^{+} \twoheadrightarrow \mathsf{let}\ \mathsf{z} = \mathsf{e}_1\ \mathsf{e}_a\ \mathsf{in}\ \mathsf{e}_k}$$

$$\frac{\cdot;\Gamma \vdash \mathsf{e} : \sigma \twoheadrightarrow \mathsf{e}'}{\cdot;\Gamma \vdash^{+} \mathcal{AS}^{\sigma}\, \mathsf{e} : \sigma^{+} \twoheadrightarrow \mathsf{e}'}$$

$$\frac{\cdot;\Gamma \vdash^{+} \mathbf{v} : \mathsf{bool}^{+} \twoheadrightarrow \mathsf{e} \qquad \cdot;\Gamma \vdash^{+} \mathbf{e}_1 : \sigma^{+} \twoheadrightarrow \mathsf{e}_1 \qquad \cdot;\Gamma \vdash^{+} \mathbf{e}_2 : \sigma^{+} \twoheadrightarrow \mathsf{e}_2}{\cdot;\Gamma \vdash^{+} \mathbf{if}\ \mathbf{v}\ \mathbf{then}\ \mathbf{e}_1\ \mathbf{else}\ \mathbf{e}_2 : \sigma^{+} \twoheadrightarrow \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2}$$

$$\frac{\cdot;\Gamma \vdash^{+} \mathbf{r} : {\sigma_1}^{+} \twoheadrightarrow \mathsf{e}_r \qquad \cdot;\Gamma, \mathbf{x} : {\sigma_1}^{+} \vdash^{+} \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{e}}{\cdot;\Gamma \vdash^{+} \mathbf{let}\ \mathbf{x} = \mathbf{r}\ \mathbf{in}\ \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{let}\ \mathsf{x} = \mathsf{e}_r\ \mathsf{in}\ \mathsf{e}}$$

Figure 4: "Back-translation": Relating λ^{SA} terms to λ^{S} terms (part I)

The latter includes boundary terms ${}^{\sigma}\mathcal{SA}\,\mathbf{e}$, which means that we also have to be able to "back-translate" the λ^{SA} term $\mathbf{e}$ which has translation type σ^{+}.

We present the complete set of back-translation rules from λ^{SA} terms to λ^{S} terms in Figures 4 and 5. This back-translation is more involved than the back-translation from λ^{ST} to λ^{S}, essentially since λ^{A} contains more terms than λ^{T}. As before, we set up two judgments to back-translate the λ^{SA} term e to some $\mathsf{e} \in \lambda^{\mathrm{S}}$.

$$\frac{\cdot;\Gamma \vdash \mathbf{v} : \tau \qquad \cdot;\Gamma \vdash^{+} \mathbf{e}[\mathbf{v}/\mathbf{x}] : \sigma^{+} \twoheadrightarrow \mathsf{e}}{\cdot;\Gamma \vdash^{+} \mathbf{let}\ \mathbf{x} = \mathbf{v}\ \mathbf{in}\ \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{e}}\ (\nexists \sigma_r.\ {\sigma_r}^{+} = \tau)$$

$$\frac{\cdot;\Gamma \vdash \mathbf{v} : \tau_1 \times \tau_2 \qquad \mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2) \qquad \cdot;\Gamma \vdash^{+} \mathbf{e}[\mathbf{v}_i/\mathbf{x}] : \sigma^{+} \twoheadrightarrow \mathsf{e}}{\cdot;\Gamma \vdash^{+} \mathbf{let}\ \mathbf{x} = \pi_i \mathbf{v}\ \mathbf{in}\ \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{e}}\ (\nexists \sigma_r.\ {\sigma_r}^{+} = \tau_i)$$

$$\frac{\begin{array}{c}\cdot;\Gamma \vdash \mathbf{v}_1\ [\tau']\ \mathbf{v}_2 : \tau_r \qquad \cdot;\Gamma \vdash \mathbf{v}_1 : \forall\,[\alpha].\tau_2 \to \tau \qquad \cdot \vdash \tau' \qquad \cdot;\Gamma \vdash \mathbf{v}_2 : \tau_2[\tau'/\alpha] \\ \tau_r = \tau[\tau'/\alpha] \qquad \mathbf{v}_1 = \lambda\,[\alpha]\,(\mathbf{z} : \tau_2).\,\mathbf{e}_1 \\ \cdot;\Gamma \vdash^{+} (\,\mathbf{let}\ \mathbf{x} = [\,\mathbf{e}_1[\tau'/\alpha][\mathbf{v}_2/\mathbf{z}]\,]\ \mathbf{in}\ \mathbf{e}\,)^{\#} : \sigma^{+} \twoheadrightarrow \mathsf{e}\end{array}}{\cdot;\Gamma \vdash^{+} \mathbf{let}\ \mathbf{x} = \mathbf{v}_1\ [\tau']\ \mathbf{v}_2\ \mathbf{in}\ \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{e}}\ \begin{array}{l}(\nexists \sigma_r.\ {\sigma_r}^{+} = \tau_r \\ \ \nexists \sigma_1.\ {\sigma_1}^{+} = \forall\,[\alpha].\tau_2 \to \tau)\end{array}$$

$$\frac{\begin{array}{c}\cdot;\Gamma \vdash \mathbf{v}_1\ [\tau']\ \mathbf{v}_2 : \tau_r \qquad \tau_r = \tau' \qquad \mathbf{v}_2 = (\mathbf{v}_a, \lambda(\mathbf{z} : {\sigma_2}^{+}).\,\mathbf{e}_k) \qquad \cdot;\Gamma, \mathbf{z} : {\sigma_2}^{+} \vdash \mathbf{e}_k : \tau' \\ \cdot;\Gamma \vdash^{+} \mathbf{v}_1 : (\sigma_1 \to \sigma_2)^{+} \twoheadrightarrow \mathsf{e}_1 \qquad \cdot;\Gamma \vdash^{+} \mathbf{v}_a : {\sigma_1}^{+} \twoheadrightarrow \mathsf{e}_a \\ \cdot;\Gamma, \mathbf{z} : {\sigma_2}^{+} \vdash^{+} (\,\mathbf{let}\ \mathbf{x} = [\,\mathbf{e}_k\,]\ \mathbf{in}\ \mathbf{e}\,)^{\#} : \sigma^{+} \twoheadrightarrow \mathsf{e}\end{array}}{\cdot;\Gamma \vdash^{+} \mathbf{let}\ \mathbf{x} = \mathbf{v}_1\ [\tau']\ \mathbf{v}_2\ \mathbf{in}\ \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{let}\ \mathsf{z} = \mathsf{e}_1\ \mathsf{e}_a\ \mathsf{in}\ \mathsf{e}}\ (\nexists \sigma_r.\ {\sigma_r}^{+} = \tau_r)$$

Figure 5: "Back-translation": Relating λ^{SA} terms to λ^{S} terms (part II)

These have the form $\cdot;\Gamma \vdash e : \sigma \twoheadrightarrow \mathsf{e}$ (for translating σ terms) and $\cdot;\Gamma \vdash^{+} e : \sigma^{+} \twoheadrightarrow \mathsf{e}$ (for translating σ^{+} terms). Here Γ may only contain mappings of the form $\mathsf{x} : \sigma$ or $\mathbf{y} : \sigma^{+}$—that is, Γ may *only* contain variables of source type or translation type. $\Gamma^{\twoheadrightarrow}$ denotes the environment Γ with all mappings of the form $\mathbf{y} : \sigma^{+}$ replaced by $\mathsf{y} : \sigma$. $\Gamma^{\twoheadrightarrow}$ is the environment used to type-check e.

The rules for translating λ^{SA} terms $\mathsf{e} : \sigma$ are again straightforward, defined by induction on the structure of the term. The only interesting case is the boundary ${}^{\sigma}\mathcal{SA}\,\mathbf{e}$ where we switch to the other judgment ($\vdash^{+}$) which translates terms $\mathbf{e}$ that have type σ^{+}. Translating target boolean values ($\mathbf{true}$, $\mathbf{false}$), target variables $\mathbf{y}$, and $\boldsymbol{\lambda}$ terms of type ${\sigma_1 \to \sigma_2}^{+}$ proceeds exactly as before.

The back-translation of terms $\pi_i \mathbf{v}$ is interesting. When translating $\pi_1 \mathbf{v}$ of type σ^{+}, $\mathbf{v}$ must have type $\sigma^{+} \times \tau_2$. But since the latter is neither a source type nor a translation type, $\mathbf{v}$ cannot be a variable. (This is again where the restriction on the codomain of Γ plays a critical role.) Therefore, $\mathbf{v}$ must be a pair $(\mathbf{v}_1, \mathbf{v}_2)$, which means partial evaluation is possible. We extract the first component of the pair $\mathbf{v}_1$—which has type σ^{+}—and back-translate it. Note that this rule is well founded because in the premise we only back-translate a subterm of the original term. The back-translation of $\pi_2 \mathbf{v}$ is analogous.

The two rules for $\mathbf{v}_1\ [\tau']\ \mathbf{v}_2$ are exactly as before: in the first of these, $\mathbf{v}_1$ is not of translation type, while in the second, $\mathbf{v}_1$ is of translation type. Back-translating the boundary term $\mathcal{AS}^{\,\sigma}\,\mathsf{e}$ is straightforward as the subterm e has source type σ. The back-translation of $\mathbf{if}$ expressions is easy as before since all of the subterms are of translation type.

At this point we have discussed back-translation of all target terms of translation type *except* for terms of the form $\mathbf{let}\ \mathbf{x} = \mathbf{r}\ \mathbf{in}\ \mathbf{e}$. For such terms there are two different situations we must consider, namely the case where $\mathbf{r}$ is of translation type and the case where it is not.

First, consider the case where $\mathbf{r}$ is of translation type ${\sigma_1}^{+}$ (last rule in Figure 4). This case is straightforward since we can back-translate $\mathbf{r}$ and also back-translate the body of the let with $\mathbf{x}$ added to Γ. We then use the results of these back-translations to assemble the final back-translation for $\mathbf{let}\ \mathbf{x} = \mathbf{r}\ \mathbf{in}\ \mathbf{e}$.

Next, consider the back-translation of $\mathbf{let}\ \mathbf{x} = \mathbf{r}\ \mathbf{in}\ \mathbf{e}$ where $\mathbf{r}$ does *not* have translation type. All the rules relevant to this scenario are collected in Figure 5: we have rules for the case when $\mathbf{r}$ is a value $\mathbf{v}$, when $\mathbf{r}$ is a projection $\pi_i \mathbf{v}$, and when $\mathbf{r}$ is an application $\mathbf{v}_1\ [\tau']\ \mathbf{v}_2$. Note that we do not have a rule for when $\mathbf{r}$ is of the form $\mathcal{AS}^{\,\sigma_1}\,\mathsf{e}$ since the latter would have to have the type ${\sigma_1}^{+}$ which contradicts the fact that $\mathbf{r}$ does *not* have translation type. Let us consider each of the Figure 5 rules in turn.

First, when $\mathbf{r}$ is a value $\mathbf{v}$, we can perform partial evaluation: we simply substitute $\mathbf{v}$ for $\mathbf{x}$ in the body of the $\mathbf{let}$ and then back-translate the resulting term $\mathbf{e}[\mathbf{v}/\mathbf{x}]$ which has translation type σ^{+}. This rule is well founded because $\mathbf{e}[\mathbf{v}/\mathbf{x}]$ will reduce in fewer steps than $\mathbf{let}\ \mathbf{x} = \mathbf{v}\ \mathbf{in}\ \mathbf{e}$.

Next, when $\mathbf{r}$ is of the form $\pi_i\mathbf{v}$, the value $\mathbf{v}$ has type $\tau_1 \times \tau_2$ which is not a translation type. Hence, $\mathbf{v}$ cannot be a variable. Thus it must be of the form $(\mathbf{v}_1, \mathbf{v}_2)$ which means that partial evaluation is possible. We now extract the i-th component and substitute it for $\mathbf{x}$ in the body of the let. The resulting term is of translation type σ^+ so we can continue to back-translate. This rule is well founded because $\mathbf{e}[\mathbf{v}_i/\mathbf{x}]$ will reduce in fewer steps than $\mathbf{let}\ \mathbf{x} = \pi_i\mathbf{v}\ \mathbf{in}\ \mathbf{e}$.

Finally, when $\mathbf{r}$ is of the form $\mathbf{v}_1\ [\tau']\ \mathbf{v}_2$, we have two rules, one where $\mathbf{v}_1$ is *not* of translation type (third rule in Figure 5) and one where $\mathbf{v}_1$ is of translation type (last rule in Figure 5). These two rules are essentially analogous to their non-let-bound counterparts (i.e., the rules for $\mathbf{v}_1\ [\tau']\ \mathbf{v}_2$ in Figure 4) except for the fact that there we finally back-translate the terms $\mathbf{e}[\tau'/\alpha][\mathbf{v}_2/\mathbf{z}]$ and $\mathbf{e}_k$, respectively, whereas here we must back-translate let-bindings of these terms. The only subtlety is that we must re-normalize these let-bound terms before we can back-translate.

As before, our rules are exhaustive, in the sense that all possible terms of type σ and σ^+ have been covered.

At this point, we can explain why we decided to decompose the equivalence preservation proof into two parts, going from λ^{S} to λ^{A} and then to λ^{F}. Had we done a direct proof from λ^{S} to λ^{F}, we would have had to back-translate λ^{F} terms (actually, to be precise, terms of the multi-language system λ^{SF}) to λ^{S}. Since there are many more λ^{F} terms than λ^{A} terms, we would have had to consider a significantly larger number of cases, resulting in an extremely large and unwieldy set of back-translation rules. We think that such a strategy muddles the essence of the back-translation and makes it hard for one to easily have confidence that all cases have been covered by the back-translation rules. We chose λ^{A} as an intermediate point in our proof because it seems to provide a sweet spot in terms of overall proof effort while keeping the necessary back-translation rules still fairly easy to follow.

Next we show that for every term of type σ and σ^+, it is possible to construct a finite back-translation derivation, and that the back-translation e is equivalent to the original e. For the equivalence statement, on the right-hand side, we have to replace all target $\mathbf{y}$ variables with $\mathcal{AS}^{\sigma}\,\mathsf{y}$ since $\Gamma^{\twoheadrightarrow}$ contains $\mathsf{y} : \sigma$ in place of $\mathbf{y} : \sigma^+$.

Lemma 1.4 **(From λ^{SA} term : σ / σ^+ to equivalent λ^{S} term)**
Let $\Gamma ::= \cdot \mid \Gamma, \mathsf{x} : \sigma \mid \mathbf{y} : \sigma^+$

1. *If* $\cdot; \Gamma \vdash e : \sigma$ *then there exists* $\mathsf{e} \in \lambda^{\mathrm{S}}$ *s.t.* $\cdot; \Gamma \vdash e : \sigma \twoheadrightarrow \mathsf{e}$ *and* $\cdot; \Gamma^{\twoheadrightarrow} \vdash e[\overline{(\mathcal{AS}^{\Gamma^{\twoheadrightarrow}(\mathsf{y})}\,\mathsf{y})/\mathbf{y}}] \approx^{log}_{\mathrm{ST}} \mathsf{e} : \sigma$.

2. *If* $\cdot; \Gamma \vdash e : \sigma^+$ *then there exists* $\mathsf{e} \in \lambda^{\mathrm{S}}$ *s.t.* $\cdot; \Gamma \vdash^+ e : \sigma^+ \twoheadrightarrow \mathsf{e}$ *and* $\cdot; \Gamma^{\twoheadrightarrow} \vdash {}^{\sigma}\mathcal{SA}\,(e[\overline{(\mathcal{AS}^{\Gamma^{\twoheadrightarrow}(\mathsf{y})}\,\mathsf{y})/\mathbf{y}}]) \approx^{log}_{\mathrm{ST}}$ $\mathsf{e} : \sigma$.

Proof (1) and (2) are proved by simultaneous induction since the typing rules for terms of type σ and σ^+ are mutually dependent. We then proceed, as before by induction on the length of the reduction sequence for e, nested induction on the type σ, and innermost induction on the structure of the term e. □

Finally, our desired lemma, that $\Gamma \vdash \mathsf{e}_1 \approx^{ctx}_{\mathrm{S}} \mathsf{e}_2 : \sigma$ implies $\cdot; \Gamma \vdash \mathsf{e}_1 \approx^{ctx}_{\mathrm{A}} \mathsf{e}_2 : \sigma$, follows as a corollary from the lemma below.

Lemma 1.5 **(Ciu-equiv in λ^{S} implies ciu-equiv in λ^{SA})**
Let Γ *be a* λ^{S} *environment, and let* e_1 *and* e_2 *be* λ^{S} *terms.*
If $\Gamma \vdash \mathsf{e}_1 \precsim^{ciu}_{\mathrm{S}} \mathsf{e}_2 : \sigma$ *then* $\cdot; \Gamma \vdash \mathsf{e}_1 \precsim^{ciu}_{\mathrm{A}} \mathsf{e}_2 : \sigma$.

Proof Suppose $E : (\cdot; \cdot \vdash \sigma \Rightarrow (\cdot; \cdot \vdash \mathsf{bool})$, and $\gamma_{\mathrm{sa}} : \Gamma$ and $E[\gamma_{\mathrm{sa}}(\mathsf{e}_1)] \Downarrow \mathsf{v}$ where $\mathsf{v} : \mathsf{bool}$. Show: $E[\gamma_{\mathrm{sa}}(\mathsf{e}_2)] \Downarrow \mathsf{v}$.
We back-translate E (or, to be precise, $\lambda \mathsf{x} : \sigma.\, E[\mathsf{x}]$) and γ_{sa} to e_E and γ_{s}. By Lemma 1.4 these are equivalent to the original E and γ_{sa}. Hence, $E[\gamma_{\mathrm{sa}}(\mathsf{e}_1)] \approx^{ctx}_{\mathrm{SA}} \mathsf{e}_E(\gamma_{\mathrm{s}}(\mathsf{e}_1)) : \mathsf{bool}$. Hence, the latter evaluates to v. Now, we instantiate the premise with e_E (after morphing it into a valid evaluation context), and γ_{s}. Hence, $\mathsf{e}_E(\gamma_{\mathrm{s}}(\mathsf{e}_2))$ evaluates to v. Since $\mathsf{e}_E(\gamma_{\mathrm{s}}(\mathsf{e}_2)) \approx^{ctx}_{\mathrm{SA}} E[\gamma_{\mathrm{sa}}(\mathsf{e}_2)] : \mathsf{bool}$, the latter evaluates to v. □

1.3.2 Proof: Equivalence in λ^{A} Implies Equivalence in λ^{F}

Our goal in this section is to prove that

(Part B) if $\cdot;\Gamma^{+} \vdash \mathrm{v}_1 \approx^{ctx}_{\mathrm{A}} \mathrm{v}_2 : \sigma^{\div}$, then $\cdot;\Gamma^{+} \vdash \mathrm{v}_1 \approx^{ctx}_{\mathrm{F}} \mathrm{v}_2 : \sigma^{\div}$.

Note that it suffices to prove the following:

(B') if $\Delta;\Gamma \vdash \mathrm{e}_1 : \boldsymbol{\tau}$, $\Delta;\Gamma \vdash \mathrm{e}_2 : \boldsymbol{\tau}$, and $\Delta;\Gamma \vdash \mathrm{e}_1 \approx^{ctx}_{\mathrm{A}} \mathrm{e}_2 : \boldsymbol{\tau}$, then $\Delta;\Gamma \vdash \mathrm{e}_1 \approx^{ctx}_{\mathrm{F}} \mathrm{e}_2 : \boldsymbol{\tau}$.

The rest of this section focuses on proving (B').

Logical relation for λ^{F} We start by defining a logical relation for λ^{F}—see Figure 6—and proving the fundamental property of the logical relation. We then prove that this logical relation corresponds to contextual and ciu equivalence.

Theorem 1.6 **(λ^{F} Fundamental Property)**
If $\Delta;\Gamma \vdash \mathbf{e} : \boldsymbol{\tau}$ *then* $\Delta;\Gamma \vdash \mathbf{e} \precsim^{log}_{\mathrm{F}} \mathbf{e} : \boldsymbol{\tau}$.

Proof By induction on the derivation $\Delta;\Gamma \vdash \mathbf{e} : \boldsymbol{\tau}$. Each case follows from the corresponding compatibility lemma. □

Next, we prove that the λ^{F} logical relation is *sound* and *complete* with respect to contextual equivalence. Note that the following lemmas (along with the property that $\precsim^{ctx}_{\mathrm{F}}$ implies $\precsim^{ciu}_{\mathrm{F}}$, Lemma 1.2), together establish that logical approximation $\precsim^{log}_{\mathrm{F}}$, ciu-approximation $\precsim^{ciu}_{\mathrm{F}}$, and contextual approximation $\precsim^{ctx}_{\mathrm{F}}$ all coincide. Therefore, for subsequent lemmas, when proving contextual equivalence properties, we will be free to switch to whichever definition is most convenient to work with for proving the property at hand.

Theorem 1.7 **(λ^{F}: Soundness w.r.t. Contextual Approx)**
If $\Delta;\Gamma \vdash \mathbf{e}_1 \precsim^{log}_{\mathrm{F}} \mathbf{e}_2 : \boldsymbol{\tau}$ *then* $\Delta;\Gamma \vdash \mathbf{e}_1 \precsim^{ctx}_{\mathrm{F}} \mathbf{e}_2 : \boldsymbol{\tau}$.

Lemma 1.8 **(λ^{F}: CIU Approx Implies Logical Approx)**
If $\Delta;\Gamma \vdash \mathbf{e}_1 \precsim^{ciu}_{\mathrm{F}} \mathbf{e}_2 : \boldsymbol{\tau}$ *then* $\Delta;\Gamma \vdash \mathbf{e}_1 \precsim^{log}_{\mathrm{F}} \mathbf{e}_2 : \boldsymbol{\tau}$.

Theorem 1.9 **(λ^{F}: Completeness w.r.t. Contextual Approx)**
If $\Delta;\Gamma \vdash \mathbf{e}_1 \precsim^{ctx}_{\mathrm{F}} \mathbf{e}_2 : \boldsymbol{\tau}$ *then* $\Delta;\Gamma \vdash \mathbf{e}_1 \precsim^{log}_{\mathrm{F}} \mathbf{e}_2 : \boldsymbol{\tau}$.

Proof Immediate from Lemmas 1.2 and 1.8. □

Back-translation from λ^{F} to λ^{A} To prove that if two terms e_1 and e_2 are equivalent in λ^{A} then they are equivalent in λ^{F} we need to be able to back-translate an arbitrary λ^{F} term to a λ^{A} term. Figure 7 presents the back-translation; notice that this is simply A-normalization. We prove that for every λ^{F} term $\mathbf{e}$ of type $\boldsymbol{\tau}$, it is possible to back-translate (or A-normalize) to some λ^{A} term e that is equivalent to the original λ^{F} term $\mathbf{e}$.

Lemma 1.10 **(From λ^{F} term to equivalent λ^{A} term)**
If $\Delta;\Gamma \vdash \mathbf{e} : \boldsymbol{\tau}$ *then there exists* $\mathrm{e} \in \lambda^{\mathrm{A}}$ *s.t.* $(\mathbf{e})^{\mathcal{A}} = \mathrm{e}$ *and* $\Delta';\Gamma' \vdash \mathbf{e} \approx^{log}_{\mathrm{F}} \mathrm{e} : \boldsymbol{\tau}$ *(where* Δ' *and* Γ' *are identical to* Δ *and* Γ*, respectively, except that the color of type variables in* Δ' *and the color of term variables and their types in* Γ' *is* orange *instead of* green*).*

Proof By induction on the structure of the term $\mathbf{e}$. The proof requires an auxiliary lemma that the re-normalization function $(\cdot)^{\#}$ preserves equivalence—i.e., that $\Delta;\Gamma \vdash \mathrm{E}[\mathrm{e}] \approx^{log}_{\mathrm{F}} (\mathrm{E}[\mathrm{e}])^{\#} : \boldsymbol{\tau}$. □

$$
\begin{array}{rcl}
\mathrm{Atom}[\tau_1,\tau_2] & = & \{\,(e_1,e_2) \mid \;\cdot;\cdot\vdash e_1 : \tau_1 \;\wedge\; \cdot;\cdot\vdash e_2:\tau_2\,\}\\[1em]
\mathrm{Rel}[\tau_1,\tau_2] & = & \{\,R\in\mathscr{P}(\mathrm{Atom}^{\mathrm{val}}[\tau_1,\tau_2]) \mid \\
& & \quad \forall(v_1,v_2)\in R.\;\; \forall {v_2}'.\;\; v_2 \gtrsim^{ciu}_{\mathrm{ST}} {v_2}' : \tau_2 \implies (v_1,{v_2}')\in R\,\}\\[2em]
\mathcal{V}\,[\![\alpha]\!]\,\rho & = & R \qquad \text{where } \rho(\alpha)=(\tau_1,\tau_2,R)\\[1em]
\mathcal{V}\,[\![\mathbf{bool}]\!]\,\rho & = & \{\,(v,v)\in\mathrm{Atom}[\mathbf{bool},\mathbf{bool}] \mid v=\mathbf{true}\;\vee\; v=\mathbf{false}\,\}\\[1em]
\mathcal{V}\,[\![\tau\times\tau']\!]\,\rho & = & \{\,((v_1,{v_1}'),(v_2,{v_2}'))\in\mathrm{Atom}[\rho_1(\tau\times\tau'),\rho_2(\tau\times\tau')] \mid \\
& & \quad (v_1,v_2)\in\mathcal{V}\,[\![\tau]\!]\,\rho \;\wedge\; ({v_1}',{v_2}')\in\mathcal{V}\,[\![\tau']\!]\,\rho\,\}\\[1em]
\mathcal{V}\,[\![\forall\,[\alpha].\tau\!\to\tau']\!]\,\rho & = & \{\,(\lambda\,[\alpha]\,(x:\tau).\,e_1,\lambda\,[\alpha]\,(x:\tau).\,e_2)\in\mathrm{Atom}[\rho_1(\forall\,[\alpha].\tau\!\to\tau'),\rho_2(\forall\,[\alpha].\tau\!\to\tau')] \mid \\
& & \quad \forall\tau_1,\tau_2,R\in\mathrm{Rel}[\tau_1,\tau_2].\\
& & \quad \forall(v_1,v_2)\in\mathcal{V}\,[\![\tau]\!]\,\rho[\alpha\mapsto(\tau_1,\tau_2,R)].\\
& & \qquad (e_1[\tau_1/\alpha][v_1/x],e_2[\tau_2/\alpha][v_2/x])\in\mathcal{E}\,[\![\tau']\!]\,\rho[\alpha\mapsto(\tau_1,\tau_2,R)]\,\}\\[2em]
\mathcal{E}\,[\![\tau]\!]\,\rho & = & \{\,(e_1,e_2)\in\mathrm{Atom}[\rho_1(\tau),\rho_2(\tau)] \mid \\
& & \quad \forall v_1.\; e_1\longmapsto^{*} v_1 \implies \exists v_2.\; e_2\longmapsto^{*} v_2 \;\wedge\; (v_1,v_2)\in\mathcal{V}\,[\![\tau]\!]\,\rho\,\}\\[1em]
\mathcal{D}\,[\![\cdot]\!] & = & \{\,\emptyset\,\}\\
\mathcal{D}\,[\![\Delta,\alpha]\!] & = & \{\,\rho[\alpha\mapsto(\tau_1,\tau_2,R)] \mid \;\rho\in\mathcal{D}\,[\![\Delta]\!] \;\wedge\; R\in\mathrm{Rel}[\tau_1,\tau_2]\,\}\\[1em]
\mathcal{G}\,[\![\cdot]\!]\,\rho & = & \{\,\emptyset\,\}\\
\mathcal{G}\,[\![\Gamma,x:\tau]\!]\,\rho & = & \{\,\gamma[x\mapsto(v_1,v_2)] \mid \;\gamma\in\mathcal{G}\,[\![\Gamma]\!]\,\rho \;\wedge\; (v_1,v_2)\in\mathcal{V}\,[\![\tau]\!]\,\rho\,\}\\[1em]
\Delta;\Gamma\vdash e_1 \lesssim^{log}_{\mathrm{ST}} e_2 : \tau & \stackrel{\mathrm{def}}{=} & \Delta;\Gamma\vdash e_1:\tau \;\wedge\; \Delta;\Gamma\vdash e_2:\tau \;\wedge\\
& & \forall\rho,\gamma.\;\rho\in\mathcal{D}\,[\![\Delta]\!] \;\wedge\; \gamma\in\mathcal{G}\,[\![\Gamma]\!]\,\rho \implies (\rho_1(\gamma_1(e_1)),\rho_2(\gamma_2(e_2)))\in\mathcal{E}\,[\![\tau]\!]\,\rho\\[1em]
\Delta;\Gamma\vdash e_1 \approx^{log}_{\mathrm{ST}} e_2 : \tau & \stackrel{\mathrm{def}}{=} & \Delta;\Gamma\vdash e_1 \lesssim^{log}_{\mathrm{ST}} e_2 : \tau \;\wedge\; \Delta;\Gamma\vdash e_2 \lesssim^{log}_{\mathrm{ST}} e_1 : \tau
\end{array}
$$

Figure 6: System F (λ^{F}): Logical Relation

Wrapping up proof of B' Finally, our desired lemma (B'), that $\Delta;\Gamma\vdash e_1 \approx^{ctx}_{\mathrm{A}} e_2 : \tau$ implies $\Delta;\Gamma\vdash e_1 \approx^{ctx}_{\mathrm{F}} e_2 : \tau$, follows as a corollary from the lemma below.

Lemma 1.11 (Ciu-equivalence in λ^{A} implies ciu-equivalence in λ^{F})
Let Δ and Γ be λ^{A} type and value environments, respectively. Let Δ' and Γ' be λ^{F} type and value environments that are identical to Δ and Γ, except that the color of type variables in Δ' and the color of term variables and types in Γ' is green instead of orange. Finally, let e_1 and e_2 be λ^{A} terms and let τ be a λ^{A} type. (Note that these are also valid terms and types in λ^{F}.)
If $\Delta;\Gamma\vdash e_1 \precsim^{ciu}_{\mathrm{A}} e_2 : \tau$ then $\Delta';\Gamma'\vdash e_1 \precsim^{ciu}_{\mathrm{F}} e_2 : \tau$.

Proof Suppose $\mathbf{E} : (\cdot;\cdot\vdash\delta_{\mathrm{f}}(\tau)\Rightarrow(\cdot;\cdot\vdash\mathbf{bool}), \delta_{\mathrm{f}}\models\Delta', \gamma_{\mathrm{f}}:\delta_{\mathrm{f}}(\Gamma')$ and $\mathbf{E}[\delta_{\mathrm{f}}(\gamma_{\mathrm{f}}(e_1))]\Downarrow v$ where $v:\mathbf{bool}$. We are required to show: $\mathbf{E}[\delta_{\mathrm{f}}(\gamma_{\mathrm{f}}(e_2))]\Downarrow v$.
Let δ_{f} be identical to δ_{a} except that the color of type variables and types in δ_{f} is green instead of orange. We back-translate $\mathbf{E}$ (or, to be precise, $\lambda(x:\delta_{\mathrm{f}}(\tau)).\,\mathbf{E}[x]$) and γ_{f} to $\mathbf{E}$ and γ_{a}, respectively, using the back-translation (A-normalization) in Figure 7. By Lemma 1.10 $\mathbf{E}$ and γa are equivalent to the original $\mathbf{E}$ and γ_{f}. Hence, $\mathbf{E}[\delta_{\mathrm{f}}(\gamma_{\mathrm{f}}(e_1))] \approx^{ctx}_{\mathrm{F}} \mathbf{E}(\delta_{\mathrm{a}}(\gamma_{\mathrm{a}}(e_1)) : \mathbf{bool}$. Hence, it follows that $\mathbf{E}(\delta_{\mathrm{a}}(\gamma_{\mathrm{a}}(e_1))\Downarrow v$. Next, we instantate the premise with $\mathbf{E}$ (after morphing it into a valid evaluation context), δ_{a}, and γ_{a}. Hence, $\mathbf{E}[\delta_{\mathrm{a}}(\gamma_{\mathrm{a}}(e_2))]\Downarrow v$. Since $\mathbf{E}(\delta_{\mathrm{a}}(\gamma_{\mathrm{a}}(e_2)) \approx^{ctx}_{\mathrm{F}} \mathbf{E}[\delta_{\mathrm{f}}(\gamma_{\mathrm{f}}(e_2))] : \mathbf{bool}$, it follows that $\mathbf{E}[\delta_{\mathrm{f}}(\gamma_{\mathrm{f}}(e_2))]\Downarrow v$. $\square$

$\boxed{\text{A-normalization function: } (\,\mathbf{e}\,)^{\mathcal{A}}}$

$$\begin{array}{rcl}
(\,\mathbf{x}\,)^{\mathcal{A}} & = & \mathbf{x} \\
(\,\mathbf{true}\,)^{\mathcal{A}} & = & \mathbf{true} \\
(\,\mathbf{false}\,)^{\mathcal{A}} & = & \mathbf{false} \\
(\,\mathbf{if\ e\ then\ e_1\ else\ e_2}\,)^{\mathcal{A}} & = & (\,\mathbf{let\ x} = [\,(\,\mathbf{e}\,)^{\mathcal{A}}\,]\ \mathbf{in}\ (\mathbf{if\ x\ then}\ (\,\mathbf{e_1}\,)^{\mathcal{A}}\ \mathbf{else}\ (\,\mathbf{e_2}\,)^{\mathcal{A}})\,)^{\#} \\
(\,(\mathbf{e_1}, \mathbf{e_2})\,)^{\mathcal{A}} & = & (\,\mathbf{let\ x_1} = [\,(\,\mathbf{e_1}\,)^{\mathcal{A}}\,]\ \mathbf{in}\ (\,\mathbf{let\ x_2} = [\,(\,\mathbf{e_2}\,)^{\mathcal{A}}]\ \mathbf{in}\ (\mathbf{x_1}, \mathbf{x_2})\,)^{\#}\,)^{\#} \\
(\,\pi_i \mathbf{e}\,)^{\mathcal{A}} & = & (\,\mathbf{let\ x} = [\,(\,\mathbf{e}\,)^{\mathcal{A}}\,]\ \mathbf{in}\ \pi_i \mathbf{x}\,)^{\#} \\
(\,\lambda\,[\alpha]\,(\mathbf{x} : \tau).\,\mathbf{e}\,)^{\mathcal{A}} & = & \lambda\,[\alpha]\,(\mathbf{x} : \tau).\,(\,\mathbf{e}\,)^{\mathcal{A}} \\
(\,\mathbf{e_1}\ [\tau]\ \mathbf{e_2}\,)^{\mathcal{A}} & = & (\,\mathbf{let\ x_1} = [\,(\,\mathbf{e_1}\,)^{\mathcal{A}}\,]\ \mathbf{in}\ (\,\mathbf{let\ x_2} = [\,(\,\mathbf{e_2}\,)^{\mathcal{A}}\,]\ \mathbf{in\ x_1}\ [\tau]\ \mathbf{x_2}\,)^{\#}\,)^{\#}
\end{array}$$

Figure 7: “Back-translation”: A-normalizing λ^{F} terms to λ^{A} terms

2 Related Work

Fully Abstract Denotational Models The earliest work on full abstraction was done in the context of denotational models of languages (e.g., [27, 15, 23, 1]). The denotation function can be thought of as a translation from a syntactic/operational calculus into a mathematical domain. The goal there is to ensure that the denotational semantics does not expose differences that are not operationally observable [30]. As typified by parallel OR in PCF [34], in a lot of this work, full abstraction is achieved by adding certain behaviors that are possible in the denotational semantics (target) to the operational semantics of the language being modeled (source). Our work differs somewhat from that on denotational models in that we are interested in proving full abstraction of *translations* (mostly compilers); here the target language also comes with an operational semantics. In particular, we focus on type-directed and type-preserving compilation and critically require a sufficiently "clever" type translation such that the types of compiled terms can impose well-behavedness constraints on any target-level term that might interact with the result of the translation, thus ensuring that target contexts cannot violate source-level abstractions.

Translation to Continuation-Passing Style CPS has been studied extensively in the literature. Here we only discuss work that seems most closely related to ours. A number of papers have investigated full abstraction for CPS translation, but there is no prior full abstraction result for a CPS translation that uses a locally polymorphic answer type. Meyer and Riecke [28] pointed out that standard (untyped) CPS translation does not preserve observational equivalence and conjectured as to how this could be repaired.

Several papers have looked at the use of linear types to ensure that a function calls its continuation exactly once. Zdancewic and Myers [45] present a security-typed CPS target language with higher-order, imperative features where linear continuations are needed to ensure noninterference. They also give a translation from a security-typed, direct-style language into the afore-mentioned target language using linear typing of CPS and prove that the translation preserves well-typedness. Berdine *et al.* [9, 11] show that continuations are used linearly in a variety of situations, including procedure call/return, exception raising/handling, labelled jumps (`goto` statements) and process switching (coroutines). Neither Zdancewic and Myers [45] nor Berdine *et al.* [9, 11] present any full abstraction results for their CPS translation.

Berdine, O'Hearn, and Thielecke [10] use affine typing of CPS to extract the range of CPS for a call-by-value λ-calculus. Specifically, they restrict the grammar of types in the target to only those forms exercised by the CPS translation. They then define the grammar for target terms so that there is one syntactic category for terms of each type. This allows them to give a precise characterization of the range of CPS by showing that all terms in the target come from some term in the source—that is, they prove a "no junk" lemma (also known as full completeness) that states that for each term M in the target, there exists a term N in the source such that M is $\beta\eta$-equivalent to the CPS translation of N. The proof of this "no junk" lemma requires back-translating M to get N. The critical difference between Berdine *et al.*'s work [10] and ours that their target language is *exactly as expressive* as the source (i.e., every target term can be back-translated), while our target language is *more expressive* than the source (i.e., there exist well-typed target terms that cannot be back-translated, specifically, terms of type $\boldsymbol{\tau}$ where $\boldsymbol{\tau}$ is not a translation type.) Consequently, our proof framework allows for one target language to serve as the target for different source languages and compilers, and allows components written in these different source languages to interoperate at the target level (assuming the compilation strategies rely on similar representation invariants at the target level). As a concrete example, we could define a CPS translation from a second source language, System F, to our target langauge λ^{T}. Now source code written in λ^{S} can interoperate with source code written in System F *after* CPS translation to λ^{T}, as long as we have a well-typed program after "linking" the two compiled components in λ^{T}. This would not be possible if we had resorted to a strategy like Berdine *et al.*'s where back-translation requires that the target language be no more expressive than the source.

Sabry and Felleisen [37] study equational completeness of CPS based on $\beta\eta$-equality rather than observational equivalence. They present a CPS transformation and an inverse mapping (or "back-translation"). From the CPS transformation, they extract the precise language of CPS terms closed under $\beta\eta$-equality, arriving at almost exactly the same syntax (modulo "administrative" redexes) as Berdine *et al.* [10]. Sabry and Felleisen analyze the syntax of the output of CPS while Berdine *et al.* analyze the types of the output of CPS to arrive as almost exactly the same target language syntax. Hence, Berdine *et al.*'s back-translation is essentially Sabry and Felleisen's inverse mapping (from CPS to direct style).

Hasegawa [19] has proved a full completeness result for the linear CPS transformation in the setting of a simply typed λ-calculus using syntactic methods based on long $\beta\eta$-normal forms. Like Sabry and Felleisen [37], he considers only $\beta\eta$-equivalence, not observational equivalence.

Using categorical game semantics, Laird [26] showed that for call-by-value PCF one can recover full abstraction of CPS translation by imposing an *affine* typing discipline on continuations, essentially employing the idea of "linearly-used continuations" presented by Berdine *et al.* [9]. We feel that with proofs based on game semantics, it is hard for non-experts to understand even the *statements* of the main lemmas required for the proof. Therefore, a primary objective of our work has been to come up with an operational proof technique that's simpler to understand and could (plausibly) be used throughout all the stages of a compiler. The proof techniques described in this paper, combined with recent advances in step-indexed logical relations [2, 5] seem like they would scale when applied to richer languages and successive compilation phases.

Thielecke [42] seems to have been the first to study CPS transformation with a locally polymorphic answer type, though his work focuses on the role of answer type polymorphism in a language with control effects. He uses parametricity reasoning to observe the connection between linear typing of CPS and answer type polymorphism in a pure call-by-value setting, showing that functions without control effects do not impose any constraints on the answer type and so can have a locally polymorphic answer type. He essentially proves the equivalent of our continuation-shuffling lemma, a property that he calls naturality. He has also studied answer type polymorphism for the call-by-name CPS transform and used it to show that the latter satisfies the eta-law [43]. He does not, however, discuss full abstraction of CPS translations.

Danvy [17] presents a translation from CPS programs into direct-style (DS) programs in an untyped setting. His technique relies on syntactically characterizing CPS terms that can be translated back to DS. Specifically, to be back-translatable, a CPS term must satisfy certain *occurrence conditions* (see Danvy [17], Fig. 2) that ensure that the CPS term encodes a call-by-value left-to-right evaluation order, checked essentially by parsing the CPS term using a stack that holds the formal parameters of continuations. Hence, the term

$$\lambda k_1.\, k_1\ (\lambda x.\, \lambda k_2.\, g\ x\ (\lambda v_2.\, f\ x\ (\lambda v_1.\, v_1\ v_2\ (\lambda v_3.\, k\ v_3)))) \tag{1}$$

does not satisfy Danvy's occurrence conditions over formal parameters because $(g\ x)$ is computed before $(f\ x)$, but its result v_2 is used before the result v_1 of $(f\ x)$. Danvy explains that (1) must be rewritten to an equivalent term, essentially by doing an η-expansion in CPS:

$$\lambda k_1.\, k_1\ (\lambda x.\, \lambda k_2.\, g\ x\ (\lambda v_2.\, \underline{(\lambda v.\, \lambda k_3.\, f\ x\ (\lambda v_1.\, v_1\ v\ (\lambda v_4.\, k_3 v_4)))}\ v_2\ (\lambda v_3.\, k\ v_3))) \tag{2}$$

The above term (2) does satisfy the occurrence conditions and therefore can be back-translated, using Danvy's translation to DS, to $\lambda x.\, (\lambda\, v.\, f\ x)\ (g\ x)$. We, on the other hand, use types to *semantically* characterize terms that can be translated back to our direct-style λ^{S}, without requiring that all of their subterms also have back-translatable types. As a result, we can back-translate more terms; in particular, we *can* back-translate an appropriately typed version of (1) without first rewriting it to (2) as Danvy does. Finally, our CPS grammar does not distinguish between ordinary λ's and continuation λ's as Danvy's does. He does not make use of partial evaluation as we do, i.e., to deal with subterms that are not, on their own, back-translatable.

Several researchers have investigated back-and-forth translations between direct-style and CPS semantics following Meyer and Wand's [29] work on *retractions*. An embedding-retraction pair (i, j) is a pair of functions such that $j \circ i$ is the identity function. Meyer and Wand work with the typed λ-calculus and use the "standard" type translation for CPS that we showed is not fully abstract in Section 1. They write σ' to denote their type translation of σ; let $\sigma^* = (\sigma' \to \mathsf{ans}) \to \mathsf{ans}$. They show that there exist embedding-retraction pairs (i_σ, j_σ) and (I_σ, J_σ)—definable in call-by-name λ-calculus (CBN)—where $i_\sigma : \sigma \to \sigma'$, $j_\sigma : \sigma' \to \sigma$, $I_\sigma : \sigma' \to \sigma^*$, and $J_\sigma : \sigma^* \to \sigma'$. Their main result is the Retraction Theorem which says that $(j_\sigma \circ J_\sigma)$ is the inverse of the CPS transform, i.e., $M =_{CBN} j_\sigma(J_\sigma(\overline{M}))$, where $\overline{M}$ is the CPS transform of M. The boundary terms $\mathcal{TS}^{\sigma}$ and ${}^{\sigma}\mathcal{ST}$ in our multi-language semantics are similar in spirit to i_σ and j_σ, respectively; the property that $(j_\sigma \circ i_\sigma) = \mathsf{id}$ is analogous to (part 1) of our boundary cancellation property (Lemma 9.1); and their Retraction Theorem, which says that the retraction of a translation is equivalent to the original term, is analogous to our "translation is equivalent to embedding" lemma (Corollary 9.6).[1] But there are also important technical differences since our target language (System F) is more expressive than theirs (STLC), and due to the fact that we make use of a multi-language semantics. In particular, our boundary terms are "built-in" with an appropriate operational semantics; thus our embedding-retraction pairs do not have to be *definable* in the target language as is the case with retractions, and this makes our multi-language technique more general. With retractions, the source language is generally assumed to be a strict subset of the target and the Retraction Theorem proves equivalence with respect to the (larger) target language. Note that our λ^{S} is not a strict subset of λ^{T} since the latter syntactically enforces continuation-passing style. Meyer and Wand's Retraction Theorem is essentially analogous to our proof of Part (2). There is no analog to our Part (1) and no proof of full abstraction. In particular, Meyer and Riecke [28] subsequently showed that if we replace CBN $\beta\eta$-equational reasoning with call-by-value observational equivalence, then the embedding-retraction pairs defined in Meyer-Wand no longer suffice. In fact, Meyer and Riecke failed to prove a Retraction Theorem in this setting.

Later Riecke [35] and Riecke and Viswanathan [35] investigated a semantic variation of the retraction approach with the goal of isolating side-effects in sequential programs. Also, Filinski [18] generalized Meyer and Wand's Retraction Theorem to a CPS transform for the monadic metalanguage. Like Meyer-Wand, Filinski's technique does not generalize to a language with divergence.

Berger, Honda, and Yoshida have studied fully abstract translations from various languages (with recursion [12], polymorphism [13, 14], control [21], state [20], and concurrency [20]) to linear or affinely typed—and in some cases polymorphic [13, 14]—π-calculus. They prove their translations fully abstract in the case of recursion (with source language PCF) [12], polymorphism (with source language System F) [13, 14], and control [21], but only speculate about full abstraction in the case of state and concurrency [20]. Since the usual translation of the λ-calculus into the π-calculus can be seen as a form of CPS translation, it may be useful to further investigate the connections between translations into the π-calculus and translations to continuation-passing style. Like us, Berger *et al.* rely on typing in the π-calculus to ensure that the translations are fully abstract. Unlike us, they rely on game semantics to prove their translations fully abstract. In this paper, we have shown that terms of translation type are *back-translatable*. Analogously, Berger *et al.* show that terms of translation type are *definable*. Definability says that for every π-calculus term P of translation type $\sigma^{\bullet}$, there exists a well-typed source term $M : \sigma$ (where they write $\sigma^{\bullet}$ to denote the translation of a source type σ). Like us, they note that one reason definability is difficult to establish is because subterms of P may not be of translation type, which means that the proof cannot be carried out simply by induction on typing derivations. Our strategy for dealing with subterms that are not of translation type was to show that it is always possible to perform some partial evaluation that gets rid of the problematic subterm, leaving only subterms that *are* of translation type. Their strategy is to show that every finite target term P of translation type can be represented by a *finite innocent function* that can be turned into a *finite canonical form*, which in turn is easily transformed into some source term M such that P is equivalent to the translation of M. Thus, they use the notion of *innocence* [22] from game semantics to establish, in essence, that translation types at the target level are inhabited by only well-behaved computations. They are, thus, able to perform induction on the size of the corresponding innocent functions. This approach is similar to that of Laird's [26] whose proof of full abstraction also relies on game semantics. Our proof method is more elementary as it relies on operational/syntactic techniques (coupled with typing) for back-translatability; expertise in game semantics is not required to follow the details.

Full Abstraction of Other Translations Most work on proving that translations preserve equivalence has typically resorted to adding precisely those target behaviors that are problematic to the source language. For instance, Riecke [36] investigates fully abstract translations between CBN, CBV, and lazy PCF, using

[1]In technical terms, our previous work on typed closure conversion seems closer to the work on retractions since there we do not use a multi-language semantics; instead the source and target are the same language and in this language we define wrapper functions $\mathcal{W}^{+}$ and $\mathcal{W}^{-}$ that are analogous to i and j, respectively, and we prove a theorem similar to Meyer and Wand's (except that it's for closure conversion, not CPS conversion).

denotational models of the languages that include the parallel conditional. This is needed to make the models fully abstract. Also, Sanjabi and Ong [38] investigate a translation from a core calculus of additive aspects to a target language with higher-order store in the style of ML references. After showing that their original translation is not fully abstract, they weaken the source language by endowing it with the power to construct "bad labels"—the analogue of the bad references at the target that were responsible for the failure of full abstraction.

Shikuma and Igarashi [40] prove full abstraction of a translation from STLC with seal and unseal operators to STLC with base types for each sealing authority. They use a syntactic proof method, but their back-translation is only applicable to terms *all* of whose *subterms are of translation type*. Our back-translation is more general precisely because it does not impose this restriction.

3 Source Language λ^{S}

$$\begin{array}{llll}
\textit{Types} & \sigma & ::= & \mathsf{bool} \mid \sigma_1 \to \sigma_2 \\
\textit{Values} & \mathsf{v} & ::= & \mathsf{x} \mid \mathsf{true} \mid \mathsf{false} \mid \lambda \mathsf{x} : \sigma.\, \mathsf{e} \\
\textit{Expressions} & \mathsf{e} & ::= & \mathsf{v} \mid \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 \mid \mathsf{e}_1\ \mathsf{e}_2 \\
\textit{Eval. Contexts} & \mathsf{E} & ::= & [\cdot]_{\mathsf{S}} \mid \mathsf{if}\ \mathsf{E}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 \mid \mathsf{E}\ \mathsf{e} \mid \mathsf{v}\ \mathsf{E}
\end{array}$$

$\boxed{\mathsf{e} \longmapsto \mathsf{e}'}$

$$\begin{array}{rcl}
\mathsf{if}\ \mathsf{true}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 & \longmapsto & \mathsf{e}_1 \\
\mathsf{if}\ \mathsf{false}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 & \longmapsto & \mathsf{e}_2 \\
(\lambda \mathsf{x} : \sigma.\, \mathsf{e})\ \mathsf{v} & \longmapsto & \mathsf{e}[\mathsf{v}/\mathsf{x}]
\end{array}$$

$$\frac{\mathsf{e} \longmapsto \mathsf{e}'}{\mathsf{E}[\mathsf{e}] \longmapsto \mathsf{E}[\mathsf{e}']}$$

Figure 8: Source Language (STLC): Syntax and Operational Semantics

$$\textit{Value Environment} \quad \Gamma \quad ::= \quad \cdot \mid \Gamma, \mathsf{x} : \sigma$$

$\boxed{\Gamma \vdash \mathsf{e} : \sigma}$

$$\frac{\mathsf{x} : \sigma \in \Gamma}{\Gamma \vdash \mathsf{x} : \sigma} \qquad \frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}} \qquad \frac{\Gamma \vdash \mathsf{e} : \mathsf{bool} \qquad \Gamma \vdash \mathsf{e}_1 : \sigma \qquad \Gamma \vdash \mathsf{e}_2 : \sigma}{\Gamma \vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 : \sigma}$$

$$\frac{\Gamma, \mathsf{x} : \sigma_1 \vdash \mathsf{e} : \sigma_2}{\Gamma \vdash \lambda \mathsf{x} : \sigma_1.\, \mathsf{e} : \sigma_1 \to \sigma_2} \qquad \frac{\Gamma \vdash \mathsf{e}_1 : \sigma_2 \to \sigma \qquad \Gamma \vdash \mathsf{e}_2 : \sigma_2}{\Gamma \vdash \mathsf{e}_1\ \mathsf{e}_2 : \sigma}$$

Figure 9: Source Language (STLC): Static Semantics

$$\textit{Contexts} \quad \mathsf{C} ::= [\cdot]_{\mathsf{S}} \mid \mathsf{if}\ \mathsf{C}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 \mid \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{C}\ \mathsf{else}\ \mathsf{e}_2 \mid \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{C} \mid \lambda \mathsf{x}:\sigma.\,\mathsf{C} \mid \mathsf{C}\ \mathsf{e} \mid \mathsf{e}\ \mathsf{C}$$

Figure 10: Source Language (STLC) Syntax - Contexts

$$\boxed{\vdash \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma')}$$

$$\frac{\Gamma \subseteq \Gamma'}{\vdash [\cdot]_{\mathsf{S}} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma)} \qquad \frac{\vdash \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \mathsf{bool}) \quad \Gamma' \vdash \mathsf{e}_1 : \sigma' \quad \Gamma' \vdash \mathsf{e}_2 : \sigma'}{\vdash \mathsf{if}\ \mathsf{C}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma')}$$

$$\frac{\Gamma' \vdash \mathsf{e} : \mathsf{bool} \quad \vdash \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma') \quad \Gamma' \vdash \mathsf{e}_2 : \sigma'}{\vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{C}\ \mathsf{else}\ \mathsf{e}_2 : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma')}$$

$$\frac{\Gamma' \vdash \mathsf{e} : \mathsf{bool} \quad \Gamma' \vdash \mathsf{e}_1 : \sigma' \quad \vdash \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma')}{\vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma')} \qquad \frac{\vdash \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma', \mathsf{x} : \sigma_1 \vdash \sigma_2)}{\vdash \lambda \mathsf{x}:\sigma_1.\,\mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma_1 \to \sigma_2)}$$

$$\frac{\vdash \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma_2 \to \sigma') \quad \Gamma' \vdash \mathsf{e}_2 : \sigma_2}{\vdash \mathsf{C}\ \mathsf{e}_2 : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma')} \qquad \frac{\Gamma' \vdash \mathsf{e}_1 : \sigma_2 \to \sigma' \quad \vdash \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma_2)}{\vdash \mathsf{e}_1\ \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\Gamma' \vdash \sigma')}$$

Figure 11: Source Language (STLC) Static Semantics - Contexts

Definition 3.1 (Contextual Approximation & Equivalence)
Let $\Gamma \vdash \mathsf{e}_1 : \sigma$ *and* $\Gamma \vdash \mathsf{e}_2 : \sigma$.

$$\begin{aligned}\Gamma \vdash \mathsf{e}_1 \precsim^{ctx}_{\mathrm{S}} \mathsf{e}_2 : \sigma \quad &\overset{\text{def}}{=} \quad \forall \mathsf{C}, \mathsf{v}_1.\ \vdash \mathsf{C} : (\Gamma \vdash \sigma) \Rightarrow (\cdot \vdash \mathsf{bool}) \ \wedge\ \mathsf{C}[\mathsf{e}_1] \Downarrow \mathsf{v}_1 \implies \\ &\qquad\qquad \exists \mathsf{v}_2.\ \mathsf{C}[\mathsf{e}_2] \Downarrow \mathsf{v}_2 \ \wedge\ \mathsf{v}_1 = \mathsf{v}_2\end{aligned}$$

$$\Gamma \vdash \mathsf{e}_1 \approx^{ctx}_{\mathrm{S}} \mathsf{e}_2 : \sigma \quad \overset{\text{def}}{=} \quad \Gamma \vdash \mathsf{e}_1 \precsim^{ctx}_{\mathrm{S}} \mathsf{e}_2 : \sigma \ \wedge\ \Gamma \vdash \mathsf{e}_2 \precsim^{ctx}_{\mathrm{S}} \mathsf{e}_1 : \sigma$$

Definition 3.2 (CIU Approximation & Equivalence)
Let $\Gamma \vdash \mathsf{e}_1 : \sigma$ *and* $\Gamma \vdash \mathsf{e}_2 : \sigma$.

$$\begin{aligned}\Gamma \vdash \mathsf{e}_1 \precsim^{ciu}_{\mathrm{S}} \mathsf{e}_2 : \sigma \quad &\overset{\text{def}}{=} \quad \forall \mathsf{E}, \gamma, \mathsf{v}_1.\ \vdash \mathsf{E} : (\cdot \vdash \sigma) \Rightarrow (\cdot \vdash \mathsf{bool}) \ \wedge \\ &\qquad\qquad \vdash \gamma : \Gamma \ \wedge\ \mathsf{E}[\gamma(\mathsf{e}_1)] \Downarrow \mathsf{v}_1 \implies \\ &\qquad\qquad \exists \mathsf{v}_2.\ \mathsf{E}[\gamma(\mathsf{e}_2)] \Downarrow \mathsf{v}_2 \ \wedge\ \mathsf{v}_1 = \mathsf{v}_2\end{aligned}$$

$$\Gamma \vdash \mathsf{e}_1 \approx^{ciu}_{\mathrm{S}} \mathsf{e}_2 : \sigma \quad \overset{\text{def}}{=} \quad \Gamma \vdash \mathsf{e}_1 \precsim^{ciu}_{\mathrm{S}} \mathsf{e}_2 : \sigma \ \wedge\ \Gamma \vdash \mathsf{e}_2 \precsim^{ciu}_{\mathrm{S}} \mathsf{e}_1 : \sigma$$

4 Target Language λ^{T}

Types	τ	$::=$	$\mathbf{bool} \mid \tau_1 \times \tau_2 \mid \alpha \mid \forall\,[\alpha].\tau_1 \rightarrow \tau_2$
Values	$\mathbf{v}$	$::=$	$\mathbf{x} \mid \mathbf{true} \mid \mathbf{false} \mid (\mathbf{v}_1, \mathbf{v}_2) \mid \lambda\,[\alpha]\,(\mathbf{x} : \tau).\,\mathbf{e}$
Expressions	$\mathbf{e}$	$::=$	$\mathbf{v} \mid \mathbf{if\ v\ then\ e}_1\ \mathbf{else\ e}_2 \mid \mathbf{let\ x} = \pi_i \mathbf{v}\ \mathbf{in\ e} \mid \mathbf{v}_1\,[\tau]\,\mathbf{v}_2$
Eval. Contexts	$\mathbf{E}$	$::=$	$[\cdot]_{\mathbf{T}}$

$\boxed{\mathbf{e} \longmapsto \mathbf{e}'}$

$$\begin{array}{rcl}
\mathbf{if\ true\ then\ e}_1\ \mathbf{else\ e}_2 & \longmapsto & \mathbf{e}_1 \\
\mathbf{if\ false\ then\ e}_1\ \mathbf{else\ e}_2 & \longmapsto & \mathbf{e}_2 \\
\mathbf{let\ x} = \pi_1(\mathbf{v}_1, \mathbf{v}_2)\ \mathbf{in\ e} & \longmapsto & \mathbf{e}[\mathbf{v}_1/\mathbf{x}] \\
\mathbf{let\ x} = \pi_2(\mathbf{v}_1, \mathbf{v}_2)\ \mathbf{in\ e} & \longmapsto & \mathbf{e}[\mathbf{v}_2/\mathbf{x}] \\
(\lambda\,[\alpha]\,(\mathbf{x} : \tau_1).\,\mathbf{e})\ [\tau]\ \mathbf{v} & \longmapsto & \mathbf{e}[\tau/\alpha][\mathbf{v}/\mathbf{x}]
\end{array}$$

$$\frac{\mathbf{e} \longmapsto \mathbf{e}'}{\mathbf{E}[\mathbf{e}] \longmapsto \mathbf{E}[\mathbf{e}']}$$

Figure 12: Target Language (System F): Syntax and Operational Semantics

$$\begin{array}{llll} \textit{Type Context} & \Delta & ::= & \cdot \mid \Delta, \alpha \\ \textit{Value Context} & \Gamma & ::= & \cdot \mid \Gamma, x : \tau \end{array}$$

$\boxed{\Delta \vdash \tau}$

$$\frac{\alpha \in \Delta}{\Delta \vdash \alpha} \qquad \frac{}{\Delta \vdash \mathbf{bool}} \qquad \frac{\Delta \vdash \tau_1 \qquad \Delta \vdash \tau_2}{\Delta \vdash \tau_1 \times \tau_2} \qquad \frac{\Delta, \alpha \vdash \tau_1 \qquad \Delta, \alpha \vdash \tau_2}{\Delta \vdash \forall\, [\alpha].\tau_1 \rightarrow \tau_2}$$

$\boxed{\Delta \vdash \Gamma}$

$$\frac{}{\Delta \vdash \cdot} \qquad \frac{\Delta \vdash \Gamma \qquad \Delta \vdash \tau}{\Delta \vdash \Gamma, x : \tau}$$

$\boxed{\Delta; \Gamma \vdash e : \tau}$

$$\frac{\Delta \vdash \Gamma \qquad x : \tau \in \Gamma}{\Delta; \Gamma \vdash x : \tau} \qquad \frac{\Delta \vdash \Gamma}{\Delta; \Gamma \vdash \mathbf{true} : \mathbf{bool}} \qquad \frac{\Delta \vdash \Gamma}{\Delta; \Gamma \vdash \mathbf{false} : \mathbf{bool}}$$

$$\frac{\Delta; \Gamma \vdash v : \mathbf{bool} \qquad \Delta; \Gamma \vdash e_1 : \tau \qquad \Delta; \Gamma \vdash e_2 : \tau}{\Delta; \Gamma \vdash \mathbf{if}\ v\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : \tau} \qquad \frac{\Delta; \Gamma \vdash v_1 : \tau_1 \qquad \Delta; \Gamma \vdash v_2 : \tau_2}{\Delta; \Gamma \vdash (v_1, v_2) : \tau_1 \times \tau_2}$$

$$\frac{\Delta; \Gamma \vdash v : \tau_1 \times \tau_2 \qquad \Delta; \Gamma, x : \tau_1 \vdash e : \tau}{\Delta; \Gamma \vdash \mathbf{let}\ x = \pi_1 v\ \mathbf{in}\ e : \tau} \qquad \frac{\Delta; \Gamma \vdash v : \tau_1 \times \tau_2 \qquad \Delta; \Gamma, x : \tau_2 \vdash e : \tau}{\Delta; \Gamma \vdash \mathbf{let}\ x = \pi_2 v\ \mathbf{in}\ e : \tau}$$

$$\frac{\Delta \vdash \Gamma \qquad \Delta, \alpha; \Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta; \Gamma \vdash \lambda\, [\alpha]\, (x : \tau_1).\, e : \forall\, [\alpha].\tau_1 \rightarrow \tau_2} \qquad \frac{\Delta; \Gamma \vdash v_1 : \forall\, [\alpha].\tau_2 \rightarrow \tau \qquad \Delta \vdash \tau' \qquad \Delta; \Gamma \vdash v_2 : \tau_2[\tau'/\alpha]}{\Delta; \Gamma \vdash v_1\ [\tau']\ v_2 : \tau[\tau'/\alpha]}$$

Figure 13: Target Language (System F): Static Semantics

Value Contexts	$\mathbf{C^v}$	::=	$[\cdot]^{\mathbf{v}}_{\mathbf{T}} \mid (\mathbf{C^v}, \mathbf{v}_2) \mid (\mathbf{v}_1, \mathbf{C^v}) \mid \lambda\,[\alpha]\,(\mathbf{x} : \tau).\,\mathbf{C}$
Contexts	$\mathbf{C}$	::=	$[\cdot]_{\mathbf{T}} \mid \mathbf{C^v} \mid \mathbf{if}\ \mathbf{C^v}\ \mathbf{then}\ \mathbf{e}_1\ \mathbf{else}\ \mathbf{e}_2 \mid \mathbf{if}\ \mathbf{e}\ \mathbf{then}\ \mathbf{C}\ \mathbf{else}\ \mathbf{e}_2 \mid \mathbf{if}\ \mathbf{e}\ \mathbf{then}\ \mathbf{e}_1\ \mathbf{else}\ \mathbf{C} \mid$ $\mathbf{let}\ \mathbf{x} = \pi_i \mathbf{C^v}\ \mathbf{in}\ \mathbf{e} \mid \mathbf{let}\ \mathbf{x} = \pi_i \mathbf{v}\ \mathbf{in}\ \mathbf{C} \mid \mathbf{C^v}\ [\alpha]\ \mathbf{v}_2 \mid \mathbf{v}_1\ [\alpha]\ \mathbf{C^v}$

Figure 14: Target Language (System F) Syntax - Contexts

$\boxed{\vdash \mathbf{C} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau')}$

$$\frac{\Delta \subseteq \Delta' \qquad \Gamma \subseteq \Gamma'}{\vdash [\cdot]^{\mathbf{v}}_{\mathbf{T}} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau)} \qquad \frac{\vdash \mathbf{C^v} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau_1) \qquad \Delta'; \Gamma' \vdash \mathbf{v}_2 : \tau_2}{\vdash (\mathbf{C^v}, \mathbf{v}_2) : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau_1 \times \tau_2)}$$

$$\frac{\Delta'; \Gamma' \vdash \mathbf{v}_1 : \tau_1 \qquad \vdash \mathbf{C^v} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau_2)}{\vdash (\mathbf{v}_1, \mathbf{C^v}) : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau_1 \times \tau_2)}$$

$$\frac{\vdash \mathbf{C} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta', \alpha; \Gamma', \mathbf{x} : \tau_1 \vdash \tau_2)}{\vdash \lambda\,[\alpha]\,(\mathbf{x} : \tau_1).\,\mathbf{C} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \forall\,[\alpha].\tau_1 \rightarrow \tau_2)}$$

$$\frac{\Delta \subseteq \Delta' \qquad \Gamma \subseteq \Gamma'}{\vdash [\cdot]_{\mathbf{T}} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau)} \qquad \frac{\vdash \mathbf{C} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \mathbf{bool}) \qquad \Delta'; \Gamma' \vdash \mathbf{e}_1 : \tau' \qquad \Delta'; \Gamma' \vdash \mathbf{e}_2 : \tau'}{\vdash \mathbf{if}\ \mathbf{C}\ \mathbf{then}\ \mathbf{e}_1\ \mathbf{else}\ \mathbf{e}_2 : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau')}$$

$$\frac{\Delta'; \Gamma' \vdash \mathbf{e} : \mathbf{bool} \qquad \vdash \mathbf{C} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau') \qquad \Delta'; \Gamma' \vdash \mathbf{e}_2 : \tau'}{\vdash \mathbf{if}\ \mathbf{e}\ \mathbf{then}\ \mathbf{C}\ \mathbf{else}\ \mathbf{e}_2 : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau')}$$

$$\frac{\Delta'; \Gamma' \vdash \mathbf{e} : \mathbf{bool} \qquad \Delta'; \Gamma' \vdash \mathbf{e}_1 : \tau' \qquad \vdash \mathbf{C} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau')}{\vdash \mathbf{if}\ \mathbf{e}\ \mathbf{then}\ \mathbf{e}_1\ \mathbf{else}\ \mathbf{C} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau')}$$

$$\frac{\vdash \mathbf{C^v} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau_1 \times \tau_2) \qquad \Delta'; \Gamma', \mathbf{x} : \tau_i \vdash \mathbf{e} : \tau'}{\vdash \mathbf{let}\ \mathbf{x} = \pi_i \mathbf{C^v}\ \mathbf{in}\ \mathbf{e} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau')}$$

$$\frac{\Delta'; \Gamma' \vdash \mathbf{v} : \tau_1 \times \tau_2 \qquad \vdash \mathbf{C} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma', \mathbf{x} : \tau_i \vdash \tau')}{\vdash \mathbf{let}\ \mathbf{x} = \pi_i \mathbf{v}\ \mathbf{in}\ \mathbf{C} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau')}$$

$$\frac{\vdash \mathbf{C^v} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \forall\,[\alpha].\tau_2 \rightarrow \tau') \qquad \Delta'; \Gamma' \vdash \mathbf{v}_2 : \tau_2[\tau_1/\alpha]}{\vdash \mathbf{C^v}\ [\tau_1]\ \mathbf{v}_2 : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau'[\tau_1/\alpha])}$$

$$\frac{\Delta'; \Gamma' \vdash \mathbf{v}_1 : \forall\,[\alpha].\tau_2 \rightarrow \tau' \qquad \vdash \mathbf{C^v} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau_2[\tau_1/\alpha])}{\vdash \mathbf{v}_1\ [\tau_1]\ \mathbf{C^v} : (\Delta; \Gamma \vdash \tau) \Rightarrow (\Delta'; \Gamma' \vdash \tau'[\tau_1/\alpha])}$$

Figure 15: Target Language (System F) Static Semantics - Contexts

In the definition of contextual equivalence below, we must make sure that the terms $\mathbf{C}[\mathbf{e}_1]$ and $\mathbf{C}[\mathbf{e}_2]$ are syntactically well-formed target terms. This extra check in needed because the hole in a context $\mathbf{C}$ may be of the form $[\cdot]_{\mathbf{T}}$ or $[\cdot]^{\mathbf{v}}_{\mathbf{T}}$. If the hole in context $\mathbf{C}$ is of the form $[\cdot]_{\mathbf{T}}$, then any well-typed term $\mathbf{e}$ may be placed in $\mathbf{C}$. However, if the hole in $\mathbf{C}$ is of the form $[\cdot]^{\mathbf{v}}_{\mathbf{T}}$, then $\mathbf{C}[\mathbf{e}]$ will not be a well-formed term unless $\mathbf{e}$ is a value.

Definition 4.1 (Contextual Approximation & Equivalence)
Let $\Delta; \Gamma \vdash \mathbf{e}_1 : \boldsymbol{\tau}$ *and* $\Delta; \Gamma \vdash \mathbf{e}_2 : \boldsymbol{\tau}$.

$$\begin{array}{rcl}\Delta; \Gamma \vdash \mathbf{e}_1 \precsim^{ctx}_{\mathrm{T}} \mathbf{e}_2 : \boldsymbol{\tau} & \overset{\text{def}}{=} & \forall \mathbf{C}, \mathbf{v}_1.\ \vdash \mathbf{C} : (\Delta; \Gamma \vdash \boldsymbol{\tau}) \Rightarrow (\cdot; \cdot \vdash \mathbf{bool}) \ \wedge \\ & & \quad \cdot; \cdot \vdash \mathbf{C}[\mathbf{e}_1] : \mathbf{bool} \ \wedge \ \cdot; \cdot \vdash \mathbf{C}[\mathbf{e}_2] : \mathbf{bool} \ \wedge \ \mathbf{C}[\mathbf{e}_1] \Downarrow \mathbf{v}_1 \implies \\ & & \quad \exists \mathbf{v}_2.\ \mathbf{C}[\mathbf{e}_2] \Downarrow \mathbf{v}_2 \ \wedge \ \mathbf{v}_1 = \mathbf{v}_2 \end{array}$$

$$\Delta; \Gamma \vdash \mathbf{e}_1 \approx^{ctx}_{\mathrm{T}} \mathbf{e}_2 : \boldsymbol{\tau} \ \overset{\text{def}}{=} \ \Delta; \Gamma \vdash \mathbf{e}_1 \precsim^{ctx}_{\mathrm{T}} \mathbf{e}_2 : \boldsymbol{\tau} \ \wedge \ \Delta; \Gamma \vdash \mathbf{e}_2 \precsim^{ctx}_{\mathrm{T}} \mathbf{e}_1 : \boldsymbol{\tau}$$

Below $\boldsymbol{\delta}$ is a finite map from type variables $\boldsymbol{\alpha}$ to closed λ^{T} types $\boldsymbol{\tau}$. We write $\boldsymbol{\delta} \models \Delta$ whenever $\mathrm{dom}(\boldsymbol{\delta}) = \Delta$.

Definition 4.2 (CIU Approximation & Equivalence)
Let $\Delta; \Gamma \vdash \mathbf{e}_1 : \boldsymbol{\tau}$ *and* $\Delta; \Gamma \vdash \mathbf{e}_2 : \boldsymbol{\tau}$.

$$\begin{array}{rcl}\Delta; \Gamma \vdash \mathbf{e}_1 \precsim^{ciu}_{\mathrm{T}} \mathbf{e}_2 : \boldsymbol{\tau} & \overset{\text{def}}{=} & \forall \mathbf{E}, \boldsymbol{\delta}, \gamma, \mathbf{v}_1.\ \vdash \mathbf{E} : (\cdot; \cdot \vdash \boldsymbol{\delta}(\boldsymbol{\tau})) \Rightarrow (\cdot; \cdot \vdash \mathbf{bool}) \ \wedge \\ & & \quad \boldsymbol{\delta} \models \Delta \ \wedge \ \vdash \gamma : \boldsymbol{\delta}(\Gamma) \ \wedge \ \mathbf{E}[\boldsymbol{\delta}(\gamma(\mathbf{e}_1))] \Downarrow \mathbf{v}_1 \implies \\ & & \quad \exists \mathbf{v}_2.\ \mathbf{E}[\boldsymbol{\delta}(\gamma(\mathbf{e}_2))] \Downarrow \mathbf{v}_2 \ \wedge \ \mathbf{v}_1 = \mathbf{v}_2 \end{array}$$

$$\Delta; \Gamma \vdash \mathbf{e}_1 \approx^{ciu}_{\mathrm{T}} \mathbf{e}_2 : \boldsymbol{\tau} \ \overset{\text{def}}{=} \ \Delta; \Gamma \vdash \mathbf{e}_1 \precsim^{ciu}_{\mathrm{T}} \mathbf{e}_2 : \boldsymbol{\tau} \ \wedge \ \Delta; \Gamma \vdash \mathbf{e}_2 \precsim^{ciu}_{\mathrm{T}} \mathbf{e}_1 : \boldsymbol{\tau}$$

5 CPS Translation

$$\begin{array}{rcl}
(\mathsf{bool})^+ & = & \mathbf{bool} \\
(\sigma_1 \to \sigma_2)^+ & = & \forall\,[\alpha].(\sigma_1{}^+ \times (\sigma_2{}^+ \to \alpha)) \to \alpha \\
\\
\sigma^{\div} & = & \forall\,[\alpha].(\sigma^+ \to \alpha) \to \alpha \\
\\
\\
(\cdot)^+ & = & \cdot \\
(\Gamma, x : \sigma)^+ & = & \Gamma^+, \mathbf{x} : \sigma^+
\end{array}$$

Figure 16: CPS: Type Translation

$\boxed{\Gamma \vdash e : \sigma \rightsquigarrow \mathbf{v}}$ where $\cdot\,; \Gamma^+ \vdash \mathbf{v} : \sigma^{\div}$

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool} \rightsquigarrow \boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \mathsf{bool}^+ \to \alpha).\,\mathbf{k}\ \mathbf{true}} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool} \rightsquigarrow \boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \mathsf{bool}^+ \to \alpha).\,\mathbf{k}\ \mathbf{false}}$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \rightsquigarrow \mathbf{v} \qquad \Gamma \vdash e_1 : \sigma \rightsquigarrow \mathbf{v_1} \qquad \Gamma \vdash e_2 : \sigma \rightsquigarrow \mathbf{v_2}}{\begin{array}{l}\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 : \sigma \rightsquigarrow \boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \sigma^+ \to \alpha). \\ \qquad\qquad\qquad \mathbf{v}\,[\alpha]\,(\boldsymbol{\lambda}(\mathbf{x} : \mathbf{bool}).\,\mathbf{if}\ \mathbf{x}\ \mathbf{then}\ (\mathbf{v_1}\,[\alpha]\,\mathbf{k})\ \mathbf{else}\ (\mathbf{v_2}\,[\alpha]\,\mathbf{k}))\end{array}}$$

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash x : \sigma \rightsquigarrow \boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \sigma^+ \to \alpha).\,\mathbf{k}\ \mathbf{x}}$$

$$\frac{\Gamma, x : \sigma_1 \vdash e : \sigma_2 \rightsquigarrow \mathbf{v}}{\begin{array}{l}\Gamma \vdash \lambda x : \sigma_1.\,e : \sigma_1 \to \sigma_2 \rightsquigarrow \boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : (\sigma_1 \to \sigma_2)^+ \to \alpha). \\ \qquad\qquad\qquad \mathbf{k}\ (\boldsymbol{\lambda}\,[\beta]\,((\mathbf{x}, \mathbf{k}') : \sigma_1{}^+ \times (\sigma_2{}^+ \to \beta)).\,(\mathbf{v}\,[\beta]\,\mathbf{k}'))\end{array}}$$

$$\frac{\Gamma \vdash e_1 : \sigma_2 \to \sigma \rightsquigarrow \mathbf{v_1} \qquad \Gamma \vdash e_2 : \sigma_2 \rightsquigarrow \mathbf{v_2}}{\begin{array}{l}\Gamma \vdash e_1\ e_2 : \sigma \rightsquigarrow \boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \sigma^+ \to \alpha). \\ \qquad\qquad \mathbf{v_1}\,[\alpha]\,(\boldsymbol{\lambda}(\mathbf{x_1} : (\sigma_2 \to \sigma)^+).\,\mathbf{v_2}\,[\alpha]\,(\boldsymbol{\lambda}(\mathbf{x_2} : \sigma_2{}^+).\,\mathbf{x_1}\,[\alpha]\,(\mathbf{x_2}, \mathbf{k})))\end{array}}$$

Figure 17: CPS: Term Translation

6 Combined Language λ^{ST}

$$
\begin{array}{llcl}
\textit{Types} & \varphi & ::= & \sigma \mid \boldsymbol{\tau} \\
\\
\textit{Expressions} & \mathsf{e} & ::= & \ldots \mid {}^{\sigma}\mathcal{ST}\,\mathbf{e} \\
 & \mathbf{e} & ::= & \ldots \mid \mathbf{let\ x = (}\mathcal{TS}^{\,\sigma}\,\mathsf{e}\mathbf{)\ in\ e} \\
 & e & ::= & \mathsf{e} \mid \mathbf{e} \\
\\
\textit{Values} & v & ::= & \mathsf{v} \mid \mathbf{v} \\
\\
\textit{Eval. Contexts} & \mathsf{E} & ::= & \ldots \mid {}^{\sigma}\mathcal{ST}\,\mathbf{E} \\
 & \mathbf{E} & ::= & \ldots \mid \mathbf{let\ x = (}\mathcal{TS}^{\,\sigma}\,\mathsf{E}\mathbf{)\ in\ e} \\
 & E & ::= & \mathsf{E} \mid \mathbf{E}
\end{array}
$$

$\boxed{e \longmapsto e'}$

$$
\begin{array}{rcl}
{}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{true} & \longmapsto & \mathsf{true} \\
{}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{false} & \longmapsto & \mathsf{false} \\
{}^{\sigma_1 \to \sigma_2}\mathcal{ST}\,\mathbf{v} & \longmapsto & \lambda \mathsf{x} : \sigma_1.\ {}^{\sigma_2}\mathcal{ST}\,(\mathbf{let\ z = (}\mathcal{TS}^{\,\sigma_1}\,\mathsf{x}\mathbf{)\ in\ (v\ [}\sigma_2^{+}\mathbf{]\ (z, id))})
\\[1em]
\mathbf{let\ y = (}\mathcal{TS}^{\,\mathsf{bool}}\,\mathsf{true}\mathbf{)\ in\ e} & \longmapsto & \mathbf{e[true/y]} \\
\mathbf{let\ y = (}\mathcal{TS}^{\,\mathsf{bool}}\,\mathsf{false}\mathbf{)\ in\ e} & \longmapsto & \mathbf{e[false/y]} \\
\mathbf{let\ y = (}\mathcal{TS}^{\,\sigma_1 \to \sigma_2}\,\mathsf{v}\mathbf{)\ in\ e} & \longmapsto & \mathbf{e[v/y]} \\
 & & \text{where } \mathbf{v} = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,((\mathbf{x}, \mathbf{k}) : \sigma_1^{+} \times (\sigma_2^{+} \to \boldsymbol{\alpha})). \\
 & & \qquad \mathbf{let\ z = (}\mathcal{TS}^{\,\sigma_2}\,(\mathsf{v}\,({}^{\sigma_1}\mathcal{ST}\,\mathbf{x}))\mathbf{)\ in\ k\ z}
\end{array}
$$

$$
\frac{e \ \longmapsto\ e'}{E[e] \ \longmapsto\ E[e']}
$$

Figure 18: Combined Language (λ^{ST}): Syntax and Operational Semantics

$$\begin{array}{llll} \textit{Type Context} & \Delta & ::= & \cdot \mid \Delta, \boldsymbol{\alpha} \\ \textit{Value Context} & \Gamma & ::= & \cdot \mid \Gamma, \mathsf{x} : \sigma \mid \Gamma, \mathbf{x} : \boldsymbol{\tau} \end{array}$$

$\boxed{\Delta \vdash \varphi}$

$$\frac{}{\Delta \vdash \mathsf{bool}} \qquad \frac{\Delta \vdash \sigma_1 \qquad \Delta \vdash \sigma_2}{\Delta \vdash \sigma_1 \rightarrow \sigma_2}$$

$$\frac{\boldsymbol{\alpha} \in \Delta}{\Delta \vdash \boldsymbol{\alpha}} \qquad \frac{}{\Delta \vdash \mathbf{bool}} \qquad \frac{\Delta \vdash \boldsymbol{\tau}_1 \qquad \Delta \vdash \boldsymbol{\tau}_2}{\Delta \vdash \boldsymbol{\tau}_1 \times \boldsymbol{\tau}_2} \qquad \frac{\Delta, \boldsymbol{\alpha} \vdash \boldsymbol{\tau}_1 \qquad \Delta, \boldsymbol{\alpha} \vdash \boldsymbol{\tau}_2}{\Delta \vdash \forall\, [\boldsymbol{\alpha}].\boldsymbol{\tau}_1 \rightarrow \boldsymbol{\tau}_2}$$

$\boxed{\Delta \vdash \Gamma}$

$$\frac{}{\Delta \vdash \cdot} \qquad \frac{\Delta \vdash \Gamma \qquad \Delta \vdash \sigma}{\Delta \vdash \Gamma, \mathsf{x} : \sigma} \qquad \frac{\Delta \vdash \Gamma \qquad \Delta \vdash \boldsymbol{\tau}}{\Delta \vdash \Gamma, \mathbf{x} : \boldsymbol{\tau}}$$

$\boxed{\Delta; \Gamma \vdash e : \varphi}$

$$\frac{\Delta \vdash \Gamma \qquad \mathsf{x} : \sigma \in \Gamma}{\Delta; \Gamma \vdash \mathsf{x} : \sigma} \qquad \frac{\Delta \vdash \Gamma}{\Delta; \Gamma \vdash \mathsf{true} : \mathsf{bool}} \qquad \frac{\Delta \vdash \Gamma}{\Delta; \Gamma \vdash \mathsf{false} : \mathsf{bool}} \qquad \frac{\Delta; \Gamma \vdash \mathsf{e} : \mathsf{bool} \qquad \Delta; \Gamma \vdash \mathsf{e}_1 : \sigma \qquad \Delta; \Gamma \vdash \mathsf{e}_2 : \sigma}{\Delta; \Gamma \vdash \mathsf{if\ e\ then\ e_1\ else\ e_2} : \sigma}$$

$$\frac{\Delta; \Gamma, \mathsf{x} : \sigma_1 \vdash \mathsf{e} : \sigma_2}{\Delta; \Gamma \vdash \lambda \mathsf{x} : \sigma_1 .\, \mathsf{e} : \sigma_1 \rightarrow \sigma_2} \qquad \frac{\Delta; \Gamma \vdash \mathsf{e}_1 : \sigma_2 \rightarrow \sigma \qquad \Delta; \Gamma \vdash \mathsf{e}_2 : \sigma_2}{\Delta; \Gamma \vdash \mathsf{e}_1\ \mathsf{e}_2 : \sigma}$$

$$\frac{\Delta \vdash \Gamma \qquad \mathbf{x} : \boldsymbol{\tau} \in \Gamma}{\Delta; \Gamma \vdash \mathbf{x} : \boldsymbol{\tau}} \qquad \frac{\Delta \vdash \Gamma}{\Delta; \Gamma \vdash \mathbf{true} : \mathbf{bool}} \qquad \frac{\Delta \vdash \Gamma}{\Delta; \Gamma \vdash \mathbf{false} : \mathbf{bool}}$$

$$\frac{\Delta; \Gamma \vdash \mathbf{v} : \mathbf{bool} \qquad \Delta; \Gamma \vdash \mathbf{e}_1 : \boldsymbol{\tau} \qquad \Delta; \Gamma \vdash \mathbf{e}_2 : \boldsymbol{\tau}}{\Delta; \Gamma \vdash \mathbf{if\ v\ then\ e_1\ else\ e_2} : \boldsymbol{\tau}} \qquad \frac{\Delta; \Gamma \vdash \mathbf{v}_1 : \boldsymbol{\tau}_1 \qquad \Delta; \Gamma \vdash \mathbf{v}_2 : \boldsymbol{\tau}_2}{\Delta; \Gamma \vdash (\mathbf{v}_1, \mathbf{v}_2) : \boldsymbol{\tau}_1 \times \boldsymbol{\tau}_2}$$

$$\frac{\Delta; \Gamma \vdash \mathbf{v} : \boldsymbol{\tau}_1 \times \boldsymbol{\tau}_2 \qquad \Delta; \Gamma, \mathbf{x} : \boldsymbol{\tau}_1 \vdash \mathbf{e} : \boldsymbol{\tau}}{\Delta; \Gamma \vdash \mathbf{let\ x} = \boldsymbol{\pi}_1 \mathbf{v\ in\ e} : \boldsymbol{\tau}} \qquad \frac{\Delta; \Gamma \vdash \mathbf{v} : \boldsymbol{\tau}_1 \times \boldsymbol{\tau}_2 \qquad \Delta; \Gamma, \mathbf{x} : \boldsymbol{\tau}_2 \vdash \mathbf{e} : \boldsymbol{\tau}}{\Delta; \Gamma \vdash \mathbf{let\ x} = \boldsymbol{\pi}_2 \mathbf{v\ in\ e} : \boldsymbol{\tau}}$$

$$\frac{\Delta \vdash \Gamma \qquad \Delta, \boldsymbol{\alpha}; \Gamma, \mathbf{x} : \boldsymbol{\tau}_1 \vdash \mathbf{e} : \boldsymbol{\tau}_2}{\Delta; \Gamma \vdash \boldsymbol{\lambda}\, [\boldsymbol{\alpha}]\, (\mathbf{x} : \boldsymbol{\tau}_1).\, \mathbf{e} : \forall\, [\boldsymbol{\alpha}].\boldsymbol{\tau}_1 \rightarrow \boldsymbol{\tau}_2} \qquad \frac{\Delta; \Gamma \vdash \mathbf{v}_1 : \forall\, [\boldsymbol{\alpha}].\boldsymbol{\tau}_2 \rightarrow \boldsymbol{\tau} \qquad \Delta \vdash \boldsymbol{\tau}' \qquad \Delta; \Gamma \vdash \mathbf{v}_2 : \boldsymbol{\tau}_2[\boldsymbol{\tau}'/\boldsymbol{\alpha}]}{\Delta; \Gamma \vdash \mathbf{v}_1\ [\boldsymbol{\tau}']\ \mathbf{v}_2 : \boldsymbol{\tau}[\boldsymbol{\tau}'/\boldsymbol{\alpha}]}$$

$$\frac{\Delta; \Gamma \vdash \mathbf{e} : \sigma^{+}}{\Delta; \Gamma \vdash {}^{\sigma}\mathcal{ST}\, \mathbf{e} : \sigma} \qquad \frac{\Delta; \Gamma \vdash \mathsf{e} : \sigma \qquad \Delta; \Gamma, \mathbf{x} : \sigma^{+} \vdash \mathbf{e} : \boldsymbol{\tau}}{\Delta; \Gamma \vdash \mathbf{let\ x} = (\mathcal{TS}^{\,\sigma}\ \mathsf{e})\ \mathbf{in\ e} : \boldsymbol{\tau}}$$

Figure 19: Combined Language (λ^{ST}): Static Semantics

$$\begin{array}{llll} \textit{Contexts} & \mathsf{C} & ::= & \ldots \mid {}^{\sigma}\mathcal{ST}\,\mathbf{C} \\ & \mathbf{C} & ::= & \ldots \mid \mathbf{let\ x} = (\mathcal{TS}\,{}^{\sigma}\,\mathsf{C})\ \mathbf{in\ e} \mid \mathbf{let\ x} = (\mathcal{TS}\,{}^{\sigma}\,\mathsf{e})\ \mathbf{in\ C} \\ & C & ::= & \mathsf{C} \mid \mathbf{C} \end{array}$$

Figure 20: Combined Language (λ^{ST}) Syntax - Contexts

$$\boxed{\vdash C : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \varphi')}$$

$$\frac{\Delta \subseteq \Delta' \qquad \Gamma \subseteq \Gamma'}{\vdash [\cdot]_{\mathsf{S}} : (\Delta;\Gamma \vdash \sigma) \Rightarrow (\Delta';\Gamma' \vdash \sigma)} \qquad \frac{\vdash \mathsf{C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \mathsf{bool}) \qquad \Delta';\Gamma' \vdash \mathsf{e}_1 : \sigma' \qquad \Delta';\Gamma' \vdash \mathsf{e}_2 : \sigma'}{\vdash \mathsf{if\ C\ then\ e_1\ else\ e_2} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma')}$$

$$\frac{\Delta';\Gamma' \vdash \mathsf{e} : \mathsf{bool} \qquad \vdash \mathsf{C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma') \qquad \Delta';\Gamma' \vdash \mathsf{e}_2 : \sigma'}{\vdash \mathsf{if\ e\ then\ C\ else\ e_2} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma')}$$

$$\frac{\Delta';\Gamma' \vdash \mathsf{e} : \mathsf{bool} \qquad \Delta';\Gamma' \vdash \mathsf{e}_1 : \sigma' \qquad \vdash \mathsf{C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma')}{\vdash \mathsf{if\ e\ then\ e_1\ else\ C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma')}$$

$$\frac{\vdash \mathsf{C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma', \mathsf{x} : \sigma_1 \vdash \sigma_2)}{\vdash \lambda \mathsf{x} : \sigma_1 .\, \mathsf{C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma_1 \rightarrow \sigma_2)} \qquad \frac{\vdash \mathsf{C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma_2 \rightarrow \sigma') \qquad \Delta';\Gamma' \vdash \mathsf{e}_2 : \sigma_2}{\vdash \mathsf{C\ e_2} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma')}$$

$$\frac{\Delta';\Gamma' \vdash \mathsf{e}_1 : \sigma_2 \rightarrow \sigma' \qquad \vdash \mathsf{C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma_2)}{\vdash \mathsf{e_1\ C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma')}$$

$$\frac{\vdash \mathbf{C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma^{+})}{\vdash {}^{\sigma}\mathcal{ST}\,\mathbf{C} : (\Delta;\Gamma \vdash \varphi) \Rightarrow (\Delta';\Gamma' \vdash \sigma)}$$

Figure 21: Combined Language (λ^{ST}) Static Semantics - Contexts I

$\boxed{\vdash C : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\varphi')}$ (contd.)

$$\frac{\Delta\subseteq\Delta' \qquad \Gamma\subseteq\Gamma'}{\vdash [\cdot]^{\mathbf{v}}_{\mathbf{T}} : (\Delta;\Gamma\vdash\boldsymbol{\tau})\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau})} \qquad \frac{\vdash \mathbf{C}^{\mathbf{v}} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}_1) \qquad \Delta';\Gamma'\vdash\mathbf{v}_2:\boldsymbol{\tau}_2}{\vdash (\mathbf{C}^{\mathbf{v}},\mathbf{v}_2) : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}_1\times\boldsymbol{\tau}_2)}$$

$$\frac{\Delta';\Gamma'\vdash\mathbf{v}_1:\boldsymbol{\tau}_1 \qquad \vdash \mathbf{C}^{\mathbf{v}} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}_2)}{\vdash (\mathbf{v}_1,\mathbf{C}^{\mathbf{v}}) : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}_1\times\boldsymbol{\tau}_2)}$$

$$\frac{\vdash \mathbf{C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta',\boldsymbol{\alpha};\Gamma',\mathbf{x}:\boldsymbol{\tau}_1\vdash\boldsymbol{\tau}_2)}{\vdash \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{x}:\boldsymbol{\tau}_1).\,\mathbf{C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau}_1\!\rightarrow\boldsymbol{\tau}_2)}$$

$$\frac{\Delta\subseteq\Delta' \qquad \Gamma\subseteq\Gamma'}{\vdash [\cdot]_{\mathbf{T}} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau})} \qquad \frac{\vdash \mathbf{C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\mathbf{bool}) \qquad \Delta';\Gamma'\vdash\mathbf{e}_1:\boldsymbol{\tau}' \qquad \Delta';\Gamma'\vdash\mathbf{e}_2:\boldsymbol{\tau}'}{\vdash \mathbf{if\ C\ then\ e_1\ else\ e_2} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}')}$$

$$\frac{\Delta';\Gamma'\vdash\mathbf{e}:\mathbf{bool} \qquad \vdash \mathbf{C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}') \qquad \Delta';\Gamma'\vdash\mathbf{e}_2:\boldsymbol{\tau}'}{\vdash \mathbf{if\ e\ then\ C\ else\ e_2} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}')}$$

$$\frac{\Delta';\Gamma'\vdash\mathbf{e}:\mathbf{bool} \qquad \Delta';\Gamma'\vdash\mathbf{e}_1:\boldsymbol{\tau}' \qquad \vdash \mathbf{C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}')}{\vdash \mathbf{if\ e\ then\ e_1\ else\ C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}')}$$

$$\frac{\vdash \mathbf{C}^{\mathbf{v}} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}_1\times\boldsymbol{\tau}_2) \qquad \Delta';\Gamma',\mathbf{x}:\boldsymbol{\tau}_i\vdash\mathbf{e}:\boldsymbol{\tau}'}{\vdash \mathbf{let\ x}=\boldsymbol{\pi}_i\mathbf{C}^{\mathbf{v}}\ \mathbf{in\ e} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}')}$$

$$\frac{\Delta';\Gamma'\vdash\mathbf{v}:\boldsymbol{\tau}_1\times\boldsymbol{\tau}_2 \qquad \vdash \mathbf{C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma',\mathbf{x}:\boldsymbol{\tau}_i\vdash\boldsymbol{\tau}')}{\vdash \mathbf{let\ x}=\boldsymbol{\pi}_i\mathbf{v}\ \mathbf{in\ C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}')}$$

$$\frac{\vdash \mathbf{C}^{\mathbf{v}} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau}_2\!\rightarrow\boldsymbol{\tau}') \qquad \Delta';\Gamma'\vdash\mathbf{v}_2:\boldsymbol{\tau}_2[\boldsymbol{\tau}_1/\boldsymbol{\alpha}]}{\vdash \mathbf{C}^{\mathbf{v}}\ [\boldsymbol{\tau}_1]\ \mathbf{v}_2 : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}'[\boldsymbol{\tau}_1/\boldsymbol{\alpha}])}$$

$$\frac{\Delta';\Gamma'\vdash\mathbf{v}_1:\forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau}_2\!\rightarrow\boldsymbol{\tau}' \qquad \vdash \mathbf{C}^{\mathbf{v}} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}_2[\boldsymbol{\tau}_1/\boldsymbol{\alpha}])}{\vdash \mathbf{v}_1\ [\boldsymbol{\tau}_1]\ \mathbf{C}^{\mathbf{v}} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau}'[\boldsymbol{\tau}_1/\boldsymbol{\alpha}])}$$

$$\frac{\vdash \mathsf{C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\sigma) \qquad \Delta';\Gamma',\mathbf{x}:\sigma^{+}\vdash\mathbf{e}:\boldsymbol{\tau}}{\vdash \mathbf{let\ x}=(\mathcal{TS}^{\,\sigma}\,\mathsf{C})\ \mathbf{in\ e} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau})} \qquad \frac{\Delta';\Gamma'\vdash\mathsf{e}:\sigma \qquad \vdash \mathbf{C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma',\mathbf{x}:\sigma^{+}\vdash\boldsymbol{\tau})}{\vdash \mathbf{let\ x}=(\mathcal{TS}^{\,\sigma}\,\mathsf{e})\ \mathbf{in\ C} : (\Delta;\Gamma\vdash\varphi)\Rightarrow(\Delta';\Gamma'\vdash\boldsymbol{\tau})}$$

Figure 22: Combined Language (λ^{ST}) Static Semantics - Contexts II

Definition 6.1 (Contextual Approximation & Equivalence)
Let $\Delta; \Gamma \vdash e_1 : \varphi$ *and* $\Delta; \Gamma \vdash e_2 : \varphi$.

$$\begin{array}{rcl} \Delta; \Gamma \vdash e_1 \precsim^{ctx}_{\mathrm{ST}} e_2 : \varphi & \stackrel{\mathrm{def}}{=} & \forall C, v_1.\ \vdash C : (\Delta; \Gamma \vdash \varphi) \Rightarrow (\cdot; \cdot \vdash \mathsf{bool}) \ \wedge \\ & & \quad \cdot; \cdot \vdash C[e_1] : \mathsf{bool} \ \wedge \ \cdot; \cdot \vdash C[e_2] : \mathsf{bool} \ \wedge \ C[e_1] \Downarrow v_1 \implies \\ & & \exists v_2.\ C[e_2] \Downarrow v_2 \ \wedge \ v_1 = v_2 \end{array}$$

$$\Delta; \Gamma \vdash e_1 \approx^{ctx}_{\mathrm{ST}} e_2 : \varphi \quad \stackrel{\mathrm{def}}{=} \quad \Delta; \Gamma \vdash e_1 \precsim^{ctx}_{\mathrm{ST}} e_2 : \varphi \ \wedge \ \Delta; \Gamma \vdash e_2 \precsim^{ctx}_{\mathrm{ST}} e_1 : \varphi$$

Below δ is a finite map from type variables $\boldsymbol{\alpha}$ to closed λ^{ST} types $\boldsymbol{\tau}$. We write $\delta \models \Delta$ whenever $\mathrm{dom}(\delta) = \Delta$.

Definition 6.2 (CIU Approximation & Equivalence)
Let $\Delta; \Gamma \vdash e_1 : \varphi$ *and* $\Delta; \Gamma \vdash e_2 : \varphi$.

$$\begin{array}{rcl} \Delta; \Gamma \vdash e_1 \precsim^{ciu}_{\mathrm{ST}} e_2 : \varphi & \stackrel{\mathrm{def}}{=} & \forall E, \delta, \gamma, v_1.\ \vdash E : (\cdot; \cdot \vdash \delta(\varphi)) \Rightarrow (\cdot; \cdot \vdash \mathsf{bool}) \ \wedge \\ & & \quad \delta \models \Delta \ \wedge \ \vdash \gamma : \delta(\Gamma) \ \wedge \ E[\delta(\gamma(e_1))] \Downarrow v_1 \implies \\ & & \exists v_2.\ E[\delta(\gamma(e_2))] \Downarrow v_2 \ \wedge \ v_1 = v_2 \end{array}$$

$$\Delta; \Gamma \vdash e_1 \approx^{ciu}_{\mathrm{ST}} e_2 : \varphi \quad \stackrel{\mathrm{def}}{=} \quad \Delta; \Gamma \vdash e_1 \precsim^{ciu}_{\mathrm{ST}} e_2 : \varphi \ \wedge \ \Delta; \Gamma \vdash e_2 \precsim^{ciu}_{\mathrm{ST}} e_1 : \varphi$$

$$
\begin{aligned}
\mathrm{Atom}[\varphi_1, \varphi_2] &= \{ (e_1, e_2) \mid \cdot;\cdot \vdash e_1 : \varphi_1 \ \wedge\ \cdot;\cdot \vdash e_2 : \varphi_2 \} \\
\mathrm{Rel}[\boldsymbol{\tau}_1, \boldsymbol{\tau}_2] &= \{ R \in \mathscr{P}(\mathrm{Atom}^{\mathrm{val}}[\boldsymbol{\tau}_1, \boldsymbol{\tau}_2]) \mid \\
&\qquad \forall (v_1, v_2) \in R.\ \forall v_2'.\ v_2 \precsim^{ciu}_{\mathrm{ST}} v_2' : \boldsymbol{\tau}_2 \implies (v_1, v_2') \in R \}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{V}\,[\![\mathsf{bool}]\!]\,\rho &= \{ (\mathsf{v}, \mathsf{v}) \in \mathrm{Atom}[\mathsf{bool}, \mathsf{bool}] \mid \mathsf{v} = \mathsf{true} \ \vee\ \mathsf{v} = \mathsf{false} \} \\
\mathcal{V}\,[\![\mathsf{\sigma \to \sigma'}]\!]\,\rho &= \{ (\mathsf{\lambda x : \sigma.\, e_1}, \mathsf{\lambda x : \sigma.\, e_2}) \in \mathrm{Atom}[\mathsf{\sigma \to \sigma'}, \mathsf{\sigma \to \sigma'}] \mid \\
&\qquad \forall (\mathsf{v_1}, \mathsf{v_2}) \in \mathcal{V}\,[\![\mathsf{\sigma}]\!]\,\rho.\ (\mathsf{e_1}[\mathsf{v_1}/\mathsf{x}], \mathsf{e_2}[\mathsf{v_2}/\mathsf{x}]) \in \mathcal{E}\,[\![\mathsf{\sigma'}]\!]\,\rho \}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{V}\,[\![\boldsymbol{\alpha}]\!]\,\rho &= R \qquad \text{where } \rho(\boldsymbol{\alpha}) = (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R) \\
\mathcal{V}\,[\![\mathbf{bool}]\!]\,\rho &= \{ (\mathbf{v}, \mathbf{v}) \in \mathrm{Atom}[\mathbf{bool}, \mathbf{bool}] \mid \mathbf{v} = \mathbf{true} \ \vee\ \mathbf{v} = \mathbf{false} \} \\
\mathcal{V}\,[\![\boldsymbol{\tau} \times \boldsymbol{\tau}']\!]\,\rho &= \{ ((\mathbf{v}_1, \mathbf{v}_1{}'), (\mathbf{v}_2, \mathbf{v}_2{}')) \in \mathrm{Atom}[\rho_1(\boldsymbol{\tau} \times \boldsymbol{\tau}'), \rho_2(\boldsymbol{\tau} \times \boldsymbol{\tau}')] \mid \\
&\qquad (\mathbf{v}_1, \mathbf{v}_2) \in \mathcal{V}\,[\![\boldsymbol{\tau}]\!]\,\rho \ \wedge\ (\mathbf{v}_1{}', \mathbf{v}_2{}') \in \mathcal{V}\,[\![\boldsymbol{\tau}']\!]\,\rho \} \\
\mathcal{V}\,[\![\forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau} \to \boldsymbol{\tau}']\!]\,\rho &= \{ (\boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{x} : \boldsymbol{\tau}).\,\mathbf{e}_1, \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{x} : \boldsymbol{\tau}).\,\mathbf{e}_2) \in \mathrm{Atom}[\rho_1(\forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau} \to \boldsymbol{\tau}'), \rho_2(\forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau} \to \boldsymbol{\tau}')] \mid \\
&\qquad \forall \boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R \in \mathrm{Rel}[\boldsymbol{\tau}_1, \boldsymbol{\tau}_2]. \\
&\qquad \forall (\mathbf{v}_1, \mathbf{v}_2) \in \mathcal{V}\,[\![\boldsymbol{\tau}]\!]\,\rho[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]. \\
&\qquad\qquad (\mathbf{e}_1[\boldsymbol{\tau}_1/\boldsymbol{\alpha}][\mathbf{v}_1/\mathbf{x}], \mathbf{e}_2[\boldsymbol{\tau}_2/\boldsymbol{\alpha}][\mathbf{v}_2/\mathbf{x}]) \in \mathcal{E}\,[\![\boldsymbol{\tau}']\!]\,\rho[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)] \}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}\,[\![\varphi]\!]\,\rho &= \{ (e_1, e_2) \in \mathrm{Atom}[\rho_1(\varphi), \rho_2(\varphi)] \mid \\
&\qquad \forall v_1.\ e_1 \longmapsto^* v_1 \implies \exists v_2.\ e_2 \longmapsto^* v_2 \ \wedge\ (v_1, v_2) \in \mathcal{V}\,[\![\varphi]\!]\,\rho \} \\
\mathcal{D}\,[\![\cdot]\!] &= \{ \emptyset \} \\
\mathcal{D}\,[\![\Delta, \boldsymbol{\alpha}]\!] &= \{ \rho[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)] \mid \rho \in \mathcal{D}\,[\![\Delta]\!] \ \wedge\ R \in \mathrm{Rel}[\boldsymbol{\tau}_1, \boldsymbol{\tau}_2] \} \\
\mathcal{G}\,[\![\cdot]\!]\,\rho &= \{ \emptyset \} \\
\mathcal{G}\,[\![\Gamma, x : \varphi]\!]\,\rho &= \{ \gamma[x \mapsto (v_1, v_2)] \mid \gamma \in \mathcal{G}\,[\![\Gamma]\!]\,\rho \ \wedge\ (v_1, v_2) \in \mathcal{V}\,[\![\varphi]\!]\,\rho \} \\
\Delta; \Gamma \vdash e_1 \precsim^{log}_{\mathrm{ST}} e_2 : \varphi &\overset{\mathrm{def}}{=} \Delta; \Gamma \vdash e_1 : \varphi \ \wedge\ \Delta; \Gamma \vdash e_2 : \varphi \ \wedge \\
&\qquad \forall \rho, \gamma.\ \rho \in \mathcal{D}\,[\![\Delta]\!] \ \wedge\ \gamma \in \mathcal{G}\,[\![\Gamma]\!]\,\rho \implies (\rho_1(\gamma_1(e_1)), \rho_2(\gamma_2(e_2))) \in \mathcal{E}\,[\![\varphi]\!]\,\rho \\
\Delta; \Gamma \vdash e_1 \approx^{log}_{\mathrm{ST}} e_2 : \varphi &\overset{\mathrm{def}}{=} \Delta; \Gamma \vdash e_1 \precsim^{log}_{\mathrm{ST}} e_2 : \varphi \ \wedge\ \Delta; \Gamma \vdash e_2 \precsim^{log}_{\mathrm{ST}} e_1 : \varphi
\end{aligned}
$$

Figure 23: Combined Language (λ^{ST}): Logical Relation

7 “Backtranslation” from λ^{ST} to λ^{S}

$$\begin{array}{lcl} (\cdot)^{\twoheadrightarrow} & = & \cdot \\ (\Gamma, \mathsf{x} : \sigma)^{\twoheadrightarrow} & = & \Gamma^{\twoheadrightarrow}, \mathsf{x} : \sigma \\ (\Gamma, \mathbf{y} : \sigma^{+})^{\twoheadrightarrow} & = & \Gamma^{\twoheadrightarrow}, \mathsf{y} : \sigma \end{array}$$

$\boxed{\cdot; \Gamma \vdash e : \sigma \twoheadrightarrow \mathsf{e}}$ $\boxed{\cdot; \Gamma \vdash^{+} e : \sigma^{+} \twoheadrightarrow \mathsf{e}}$

where $\Gamma ::= \cdot \mid \Gamma, \mathsf{x} : \sigma \mid \mathbf{y} : \sigma^{+}$

and $\mathsf{e} \in \lambda^{\mathrm{S}}$ and $\Gamma^{\twoheadrightarrow} \vdash \mathsf{e} : \sigma$

$$\frac{\cdot \vdash \Gamma}{\cdot; \Gamma \vdash \mathsf{true} : \mathsf{bool} \twoheadrightarrow \mathsf{true}} \qquad \frac{\cdot \vdash \Gamma}{\cdot; \Gamma \vdash \mathsf{false} : \mathsf{bool} \twoheadrightarrow \mathsf{false}}$$

$$\frac{\cdot; \Gamma \vdash \mathsf{e} : \mathsf{bool} \twoheadrightarrow \mathsf{e}' \qquad \cdot; \Gamma \vdash \mathsf{e}_1 : \sigma \twoheadrightarrow \mathsf{e}_1' \qquad \cdot; \Gamma \vdash \mathsf{e}_2 : \sigma \twoheadrightarrow \mathsf{e}_2'}{\cdot; \Gamma \vdash \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2 : \sigma \twoheadrightarrow \mathsf{if}\ \mathsf{e}'\ \mathsf{then}\ \mathsf{e}_1'\ \mathsf{else}\ \mathsf{e}_2'} \qquad \frac{\cdot \vdash \Gamma \qquad \mathsf{x} : \sigma \in \Gamma}{\cdot; \Gamma \vdash \mathsf{x} : \sigma \twoheadrightarrow \mathsf{x}}$$

$$\frac{\cdot; \Gamma, \mathsf{x} : \sigma_1 \vdash \mathsf{e} : \sigma_2 \twoheadrightarrow \mathsf{e}'}{\cdot; \Gamma \vdash \lambda \mathsf{x} : \sigma_1 .\, \mathsf{e} : \sigma_1 \to \sigma_2 \twoheadrightarrow \lambda \mathsf{x} : \sigma_1 .\, \mathsf{e}'} \qquad \frac{\cdot; \Gamma \vdash \mathsf{e}_1 : \sigma_2 \to \sigma \twoheadrightarrow \mathsf{e}_1' \qquad \cdot; \Gamma \vdash \mathsf{e}_2 : \sigma_2 \twoheadrightarrow \mathsf{e}_2'}{\cdot; \Gamma \vdash \mathsf{e}_1\ \mathsf{e}_2 : \sigma \twoheadrightarrow \mathsf{e}_1'\ \mathsf{e}_2'} \qquad \frac{\cdot; \Gamma \vdash^{+} \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{e}}{\cdot; \Gamma \vdash {}^{\sigma}\mathcal{ST}\, \mathbf{e} : \sigma \twoheadrightarrow \mathsf{e}}$$

$$\frac{\cdot \vdash \Gamma}{\cdot; \Gamma \vdash^{+} \mathbf{true} : \mathsf{bool}^{+} \twoheadrightarrow \mathsf{true}} \qquad \frac{\cdot \vdash \Gamma}{\cdot; \Gamma \vdash^{+} \mathbf{false} : \mathsf{bool}^{+} \twoheadrightarrow \mathsf{false}}$$

$$\frac{\cdot; \Gamma \vdash^{+} \mathbf{v} : \mathsf{bool}^{+} \twoheadrightarrow \mathsf{e} \qquad \cdot; \Gamma \vdash^{+} \mathbf{e}_1 : \sigma^{+} \twoheadrightarrow \mathsf{e}_1 \qquad \cdot; \Gamma \vdash^{+} \mathbf{e}_2 : \sigma^{+} \twoheadrightarrow \mathsf{e}_2}{\cdot; \Gamma \vdash^{+} \mathbf{if}\ \mathbf{v}\ \mathbf{then}\ \mathbf{e}_1\ \mathbf{else}\ \mathbf{e}_2 : \sigma^{+} \twoheadrightarrow \mathsf{if}\ \mathsf{e}\ \mathsf{then}\ \mathsf{e}_1\ \mathsf{else}\ \mathsf{e}_2} \qquad \frac{\cdot \vdash \Gamma \qquad \mathbf{y} : \sigma^{+} \in \Gamma}{\cdot; \Gamma \vdash^{+} \mathbf{y} : \sigma^{+} \twoheadrightarrow \mathsf{y}}$$

$$\frac{\cdot \vdash \Gamma \qquad \cdot; \Gamma, \mathbf{y} : {\sigma_1}^{+} \vdash^{+} \mathbf{e}[{\sigma_2}^{+}/\boldsymbol{\alpha}][\mathbf{id}/\mathbf{k}] : {\sigma_2}^{+} \twoheadrightarrow \mathsf{e}}{\cdot; \Gamma \vdash^{+} \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,((\mathbf{y}, \mathbf{k}) : ({\sigma_1}^{+} \times ({\sigma_2}^{+} \to \boldsymbol{\alpha}))).\, \mathbf{e} : \forall\,[\boldsymbol{\alpha}].({\sigma_1}^{+} \times ({\sigma_2}^{+} \to \boldsymbol{\alpha})) \to \boldsymbol{\alpha} \twoheadrightarrow \lambda \mathsf{y} : \sigma_1 .\, \mathsf{e}}$$

$$(\dagger)\ \frac{\cdot; \Gamma \vdash \mathbf{v} : \boldsymbol{\tau}_1 \times \boldsymbol{\tau}_2 \qquad \mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2) \qquad \cdot; \Gamma \vdash^{+} \mathbf{e}[\mathbf{v}_i/\mathbf{x}] : \sigma^{+} \twoheadrightarrow \mathsf{e}}{\cdot; \Gamma \vdash^{+} \mathbf{let}\ \mathbf{x} = \boldsymbol{\pi}_i \mathbf{v}\ \mathbf{in}\ \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{e}}$$

$$(\ddagger)\ \frac{\begin{array}{c} \cdot; \Gamma \vdash \mathbf{v}_1 : \forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau}_2 \to \boldsymbol{\tau} \qquad \cdot \vdash \boldsymbol{\tau}' \qquad \cdot; \Gamma \vdash \mathbf{v}_2 : \boldsymbol{\tau}_2[\boldsymbol{\tau}'/\boldsymbol{\alpha}] \\ \sigma^{+} = \boldsymbol{\tau}[\boldsymbol{\tau}'/\boldsymbol{\alpha}] \qquad \mathbf{v}_1 = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{z} : \boldsymbol{\tau}_2).\, \mathbf{e} \qquad \cdot; \Gamma \vdash^{+} \mathbf{e}[\boldsymbol{\tau}'/\boldsymbol{\alpha}][\mathbf{v}_2/\mathbf{z}] : \sigma^{+} \twoheadrightarrow \mathsf{e} \end{array}}{\cdot; \Gamma \vdash^{+} \mathbf{v}_1\ [\boldsymbol{\tau}']\ \mathbf{v}_2 : \sigma^{+} \twoheadrightarrow \mathsf{e}}\ (\nexists \sigma_1.\ {\sigma_1}^{+} = \forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau}_2 \to \boldsymbol{\tau})$$

$$\frac{\begin{array}{c} \boldsymbol{\tau}' = \sigma^{+} \qquad \mathbf{v}_2 = (\mathbf{v}_a, \boldsymbol{\lambda}(\mathbf{z} : {\sigma_2}^{+}).\, \mathbf{e}_k) \\ \cdot; \Gamma \vdash^{+} \mathbf{v}_1 : (\sigma_1 \to \sigma_2)^{+} \twoheadrightarrow \mathsf{e}_1 \qquad \cdot; \Gamma \vdash^{+} \mathbf{v}_a : {\sigma_1}^{+} \twoheadrightarrow \mathsf{e}_a \qquad \cdot; \Gamma, \mathbf{z} : {\sigma_2}^{+} \vdash^{+} \mathbf{e}_k : \sigma^{+} \twoheadrightarrow \mathsf{e}_k \end{array}}{\cdot; \Gamma \vdash^{+} \mathbf{v}_1\ [\boldsymbol{\tau}']\ \mathbf{v}_2 : \sigma^{+} \twoheadrightarrow \mathsf{let}\ \mathsf{z} = \mathsf{e}_1\ \mathsf{e}_a\ \mathsf{in}\ \mathsf{e}_k}$$

$$\frac{\cdot; \Gamma \vdash \mathsf{e}_1 : \sigma_1 \twoheadrightarrow \mathsf{e}_1' \qquad \cdot; \Gamma, \mathbf{x} : {\sigma_1}^{+} \vdash^{+} \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{e}}{\cdot; \Gamma \vdash^{+} \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\sigma_1}\ \mathsf{e}_1)\ \mathbf{in}\ \mathbf{e} : \sigma^{+} \twoheadrightarrow \mathsf{let}\ \mathsf{x} = \mathsf{e}_1'\ \mathsf{in}\ \mathsf{e}}$$

Figure 24: Relating λ^{ST} terms to λ^{S} terms

8 λ^{ST} Logical Relation Corresponds to Contextual Equivalence

In this section, we layout the framework for proving that the λ^{ST} logical relation ($\precsim^{log}_{\mathrm{ST}}$) is sound and complete with respect to contextual equivalence ($\precsim^{ctx}_{\mathrm{ST}}$). We first prove the Fundamental Property of the logical relation (Section 8.1), then prove that the logical relation is sound (Section 8.2) and complete (Section 8.3) w.r.t. contextual equivalence.

8.1 λ^{ST} Logical Relation: Fundamental Property

The Fundamental Property of a logical relation says that if a term is well-typed in the language, then it is logically related to itself. Following Pitts [33], we prove a series of compatibility lemmas from which the Fundamental Property immediately follows.

The interesting compatibility lemmas are the ones involving boundaries—i.e., ${}^{\sigma}\mathcal{ST}\,\mathbf{e}$ and $\mathcal{TS}^{\,\sigma}\,\mathsf{e}$ (page 39). These cases require proving a "Bridge Lemma" which says that embedding (using boundaries) preserves equivalence (Lemma 8.16, page 35).

Proofs of the remaining compatibility lemmas are standard (as for any logical relation for System F), so we simply state the lemmas without proof.

Lemma 8.1 (Compatibility Source Var)
If $\mathsf{x} : \sigma \in \Gamma$ *then* $\Delta; \Gamma \vdash \mathsf{x} \precsim^{log}_{\mathrm{ST}} \mathsf{x} : \sigma$.

Lemma 8.2 (Compatibility Source True)
$\Delta; \Gamma \vdash \mathsf{true} \precsim^{log}_{\mathrm{ST}} \mathsf{true} : \mathsf{bool}$.

Lemma 8.3 (Compatibility Source False)
$\Delta; \Gamma \vdash \mathsf{false} \precsim^{log}_{\mathrm{ST}} \mathsf{false} : \mathsf{bool}$.

Lemma 8.4 (Compatibility Source If)
If $\Delta; \Gamma \vdash \mathsf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathsf{e}_2 : \mathsf{bool}$ *and* $\Delta; \Gamma \vdash \mathsf{e}'_1 \precsim^{log}_{\mathrm{ST}} \mathsf{e}'_2 : \sigma$ *and* $\Delta; \Gamma \vdash \mathsf{e}''_1 \precsim^{log}_{\mathrm{ST}} \mathsf{e}''_2 : \sigma$
then $\Delta; \Gamma \vdash \mathsf{if}\ \mathsf{e}_1\ \mathsf{then}\ \mathsf{e}'_1\ \mathsf{else}\ \mathsf{e}''_1 \precsim^{log}_{\mathrm{ST}} \mathsf{if}\ \mathsf{e}_2\ \mathsf{then}\ \mathsf{e}'_2\ \mathsf{else}\ \mathsf{e}''_2 : \sigma$.

Lemma 8.5 (Compatibility Source Abs)
If $\Delta; \Gamma, \mathsf{x} : \sigma \vdash \mathsf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathsf{e}_2 : \sigma'$
then $\Delta; \Gamma \vdash \lambda \mathsf{x} : \sigma.\, \mathsf{e}_1 \precsim^{log}_{\mathrm{ST}} \lambda \mathsf{x} : \sigma.\, \mathsf{e}_2 : \sigma \rightarrow \sigma'$.

Lemma 8.6 (Compatibility Source App)
If $\Delta; \Gamma \vdash \mathsf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathsf{e}_2 : \sigma \rightarrow \sigma'$ *and* $\Delta; \Gamma \vdash \mathsf{e}'_1 \precsim^{log}_{\mathrm{ST}} \mathsf{e}'_2 : \sigma$
then $\Delta; \Gamma \vdash \mathsf{e}_1\ \mathsf{e}_2 \precsim^{log}_{\mathrm{ST}} \mathsf{e}'_1\ \mathsf{e}'_2 : \sigma'$.

Lemma 8.7 (Compatibility Target Var)
If $\mathbf{x} : \boldsymbol{\tau} \in \Gamma$ *then* $\Delta; \Gamma \vdash \mathbf{x} \precsim^{log}_{\mathrm{ST}} \mathbf{x} : \boldsymbol{\tau}$.

Lemma 8.8 (Compatibility Target True)
$\Delta; \Gamma \vdash \mathbf{true} \precsim^{log}_{\mathrm{ST}} \mathbf{true} : \mathbf{bool}$.

Lemma 8.9 (Compatibility Target False)
$\Delta; \Gamma \vdash \mathbf{false} \precsim^{log}_{\mathrm{ST}} \mathbf{false} : \mathbf{bool}$.

Lemma 8.10 (Compatibility Target If)
If $\Delta; \Gamma \vdash \mathbf{v}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{v}_2 : \mathbf{bool}$ *and* $\Delta; \Gamma \vdash \mathbf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{e}_2 : \boldsymbol{\tau}$ *and* $\Delta; \Gamma \vdash \mathbf{e}_1{}' \precsim^{log}_{\mathrm{ST}} \mathbf{e}_2{}' : \boldsymbol{\tau}$
then $\Delta; \Gamma \vdash \mathbf{if}\ \mathbf{v}_1\ \mathbf{then}\ \mathbf{e}_1\ \mathbf{else}\ \mathbf{e}_1{}' \precsim^{log}_{\mathrm{ST}} \mathbf{if}\ \mathbf{v}_2\ \mathbf{then}\ \mathbf{e}_2\ \mathbf{else}\ \mathbf{e}_2{}' : \boldsymbol{\tau}$.

Lemma 8.11 (Compatibility Target Pair)
If $\Delta; \Gamma \vdash \mathbf{v}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{v}_2 : \boldsymbol{\tau}$ *and* $\Delta; \Gamma \vdash \mathbf{v}_1{}' \precsim^{log}_{\mathrm{ST}} \mathbf{v}_2{}' : \boldsymbol{\tau}'$
then $\Delta; \Gamma \vdash (\mathbf{v}_1, \mathbf{v}_1{}') \precsim^{log}_{\mathrm{ST}} (\mathbf{v}_2, \mathbf{v}_2{}') : \boldsymbol{\tau} \times \boldsymbol{\tau}'$.

Lemma 8.12 (Compatibility Target Fst)
If $\Delta;\Gamma \vdash \mathbf{v}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{v}_2 : \boldsymbol{\tau} \times \boldsymbol{\tau'}$ *and* $\Delta;\Gamma, \mathbf{x} : \boldsymbol{\tau} \vdash \mathbf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{e}_2 : \boldsymbol{\tau''}$
then $\Delta;\Gamma \vdash \mathbf{let\ x = \pi_1 v_1\ in\ e_1} \precsim^{log}_{\mathrm{ST}} \mathbf{let\ x = \pi_1 v_2\ in\ e_2} : \boldsymbol{\tau''}$.

Lemma 8.13 (Compatibility Target Snd)
If $\Delta;\Gamma \vdash \mathbf{v}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{v}_2 : \boldsymbol{\tau} \times \boldsymbol{\tau'}$ *and* $\Delta;\Gamma, \mathbf{x} : \boldsymbol{\tau'} \vdash \mathbf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{e}_2 : \boldsymbol{\tau''}$
then $\Delta;\Gamma \vdash \mathbf{let\ x = \pi_2 v_1\ in\ e_1} \precsim^{log}_{\mathrm{ST}} \mathbf{let\ x = \pi_2 v_2\ in\ e_2} : \boldsymbol{\tau''}$.

Lemma 8.14 (Compatibility Target Abs)
If $\Delta, \boldsymbol{\alpha};\Gamma, \mathbf{x} : \boldsymbol{\tau} \vdash \mathbf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{e}_2 : \boldsymbol{\tau'}$
then $\Delta;\Gamma \vdash \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{x} : \boldsymbol{\tau}).\,\mathbf{e}_1 \precsim^{log}_{\mathrm{ST}} \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{x} : \boldsymbol{\tau}).\,\mathbf{e}_2 : \forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau} \rightarrow \boldsymbol{\tau'}$.

Lemma 8.15 (Compatibility Target App)
If $\Delta \vdash \boldsymbol{\tau''}$ *and* $\Delta;\Gamma \vdash \mathbf{v}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{v}_2 : \forall\,[\boldsymbol{\alpha}].\boldsymbol{\tau} \rightarrow \boldsymbol{\tau'}$ *and* $\Delta;\Gamma \vdash \mathbf{v}_1{}' \precsim^{log}_{\mathrm{ST}} \mathbf{v}_2{}' : \boldsymbol{\tau}[\boldsymbol{\tau''}/\boldsymbol{\alpha}]$
then $\Delta;\Gamma \vdash \mathbf{v}_1\ [\boldsymbol{\tau''}]\ \mathbf{v}_1{}' \precsim^{log}_{\mathrm{ST}} \mathbf{v}_2\ [\boldsymbol{\tau''}]\ \mathbf{v}_2{}' : \boldsymbol{\tau'}[\boldsymbol{\tau''}/\boldsymbol{\alpha}]$.

Lemma 8.16 (Bridge Lemma)

(I) *If* $(\mathbf{e}_1, \mathbf{e}_2) \in \mathcal{E}[\![\sigma^+]\!]\,\emptyset$
then $({}^{\sigma}\mathcal{ST}\,\mathbf{e}_1, {}^{\sigma}\mathcal{ST}\,\mathbf{e}_2) \in \mathcal{E}[\![\sigma]\!]\,\emptyset$.

(II) *If* $(\mathsf{e}_1, \mathsf{e}_2) \in \mathcal{E}[\![\sigma]\!]\,\emptyset$
and $(\boldsymbol{\lambda}(\mathbf{x} : \sigma^+).\,\mathbf{e}_1{}', \boldsymbol{\lambda}(\mathbf{x} : \sigma^+).\,\mathbf{e}_2{}') \in \mathcal{V}[\![\sigma^+ \boldsymbol{\to} \boldsymbol{\tau}]\!]\,\rho$
then $(\mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma}\,\mathsf{e}_1)\ \mathbf{in}\ \mathbf{e}_1{}', \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma}\,\mathsf{e}_2)\ \mathbf{in}\ \mathbf{e}_2{}') \in \mathcal{E}[\![\boldsymbol{\tau}]\!]\,\rho$.

Proof

We prove parts I and II simultaneously, by induction on the structure of σ.

Proof of Part I:

Case $\sigma = \mathsf{bool}$:

(1) Have: $(\mathbf{e}_1, \mathbf{e}_2) \in \mathcal{E}[\![\mathbf{bool}]\!]\,\emptyset$.

- Must show: $({}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{e}_1, {}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{e}_2) \in \mathcal{E}[\![\mathsf{bool}]\!]\,\emptyset$:

(2) Consider arbitrary v_1^f such that ${}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{e}_1 \longmapsto^* \mathsf{v}_1^f$.

- Must show: $\exists \mathsf{v}_2^f.\,{}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{e}_2 \longmapsto^* \mathsf{v}_2^f$ and $(\mathsf{v}_1^f, \mathsf{v}_2^f) \in \mathcal{V}[\![\mathsf{bool}]\!]\,\emptyset$:
- Have: $\mathbf{e}_1 \longmapsto^* \mathbf{v}_1$ by (2) and operational semantics.
- Instantiating (1) with $\mathbf{v}_1$:
  - Have: $\mathbf{v}_2$ such that $\mathbf{e}_2 \longmapsto^* \mathbf{v}_2$ and $(\mathbf{v}_1, \mathbf{v}_2) \in \mathcal{V}[\![\mathbf{bool}]\!]\,\emptyset$.
  - Have: $\mathbf{v}_1 = \mathbf{v}_2 = \mathbf{true}$ or $\mathbf{v}_1 = \mathbf{v}_2 = \mathbf{false}$ by definition of $\mathcal{V}[\![\mathbf{bool}]\!]\,\cdot$.

  Case $\mathbf{v}_1 = \mathbf{v}_2 = \mathbf{true}$:
  - Have: ${}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{e}_1 \longmapsto^* {}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{true} \longmapsto \mathsf{true}$ by operational semantics.
  - Therefore: $\mathsf{v}_1^f = \mathsf{true}$.
  - Take $\mathsf{v}_2^f = \mathsf{true}$; we must show ${}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{e}_2 \longmapsto^* \mathsf{v}_2^f$, which follows from:
    - Have: ${}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{e}_2 \longmapsto^* {}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{true} \longmapsto \mathsf{true}$ by operational semantics.
    - Observe that: $(\mathsf{true}, \mathsf{true}) \in \mathcal{V}[\![\mathsf{bool}]\!]\,\emptyset$.

  Case $\mathbf{v}_1 = \mathbf{v}_2 = \mathbf{false}$:
  (Similar to **true** case.)

Case $\sigma = \sigma \to \sigma'$:

(3) Have: $(\mathbf{e}_1, \mathbf{e}_2) \in \mathcal{E}\left[\!\!\left[\forall\,[\boldsymbol{\alpha}].\sigma^+ \times (\sigma'^+ \boldsymbol{\to} \boldsymbol{\alpha}) \boldsymbol{\to} \boldsymbol{\alpha}\right]\!\!\right]\emptyset$.

- Must show: $({}^{\sigma\to\sigma'}\mathcal{ST}\,\mathbf{e}_1, {}^{\sigma\to\sigma'}\mathcal{ST}\,\mathbf{e}_2) \in \mathcal{E}[\![\sigma \to \sigma']\!]\,\emptyset$:

(4) Consider arbitrary v_1^f such that ${}^{\sigma\to\sigma'}\mathcal{ST}\,\mathbf{e}_1 \longmapsto^* \mathsf{v}_1^f$.

- Must show: $\exists \mathsf{v}_2^f.\,{}^{\sigma\to\sigma'}\mathcal{ST}\,\mathbf{e}_2 \longmapsto^* \mathsf{v}_2^f$ and $(\mathsf{v}_1^f, \mathsf{v}_2^f) \in \mathcal{V}[\![\sigma \to \sigma']\!]\,\emptyset$:
- Have: $\mathbf{e}_1 \longmapsto^* \mathbf{v}_1$ by (4) and operational semantics.
- Instantiating (3) with $\mathbf{v}_1$:

  (5) Have: $\mathbf{v}_2$ such that $\mathbf{e}_2 \longmapsto^* \mathbf{v}_2$ and $(\mathbf{v}_1, \mathbf{v}_2) \in \mathcal{V}\left[\!\!\left[\forall\,[\boldsymbol{\alpha}].\sigma^+ \times (\sigma'^+ \boldsymbol{\to} \boldsymbol{\alpha}) \boldsymbol{\to} \boldsymbol{\alpha}\right]\!\!\right]\emptyset$.

  - Have: ${}^{\sigma\to\sigma'}\mathcal{ST}\,\mathbf{e}_1 \longmapsto^* {}^{\sigma\to\sigma'}\mathcal{ST}\,\mathbf{v}_1 \longmapsto \lambda \mathsf{x} : \sigma.\ {}^{\sigma'}\mathcal{ST}\,\mathbf{let}\ \mathbf{z} = (\mathcal{TS}^{\,\sigma}\,\mathsf{x})\ \mathbf{in}\ \mathbf{v}_1\,[\sigma'^+]\,(\mathbf{z}, \mathbf{id})$ by operational semantics.

- Therefore: $\mathsf{v}_1^f = \lambda \mathsf{x} : \sigma.\, {}^{\sigma'}\mathcal{ST}\, \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\, \mathsf{x})\ \mathbf{in\ v_1}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id})$.
- Take
  $\mathsf{v}_2^f = \lambda \mathsf{x} : \sigma.\, {}^{\sigma'}\mathcal{ST}\, \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\, \mathsf{x})\ \mathbf{in\ v_2}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id})$; we must show ${}^{\sigma \to \sigma'}\mathcal{ST}\, \mathbf{e_2} \longmapsto^* \mathsf{v}_2^f$
  and $(\mathsf{v}_1^f, \mathsf{v}_2^f) \in \mathcal{V}[\![\sigma \to \sigma']\!]\,\emptyset$, which follows from:

  - Have:
    ${}^{\sigma \to \sigma'}\mathcal{ST}\, \mathbf{e_2}$
    $\longmapsto^* {}^{\sigma \to \sigma'}\mathcal{ST}\, \mathbf{v_2}$
    $\longmapsto \lambda \mathsf{x} : \sigma.\, {}^{\sigma'}\mathcal{ST}\, \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\, \mathsf{x})\ \mathbf{in\ v_2}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id})$
    by operational semantics.
  - Must show: $(\mathsf{v}_1^f, \mathsf{v}_2^f) \in \mathcal{V}[\![\sigma \to \sigma']\!]\,\emptyset$:
  - Consider arbitrary v_1' and v_2' such that $(\mathsf{v}_1', \mathsf{v}_2') \in \mathcal{V}[\![\sigma]\!]\,\emptyset$.
  - Must show:
    $({}^{\sigma'}\mathcal{ST}\, \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\, \mathbf{v_1}')\ \mathbf{in\ v_1}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id}),$
    ${}^{\sigma'}\mathcal{ST}\, \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\, \mathbf{v_2}')\ \mathbf{in\ v_2}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id})) \in \mathcal{E}[\![\sigma]\!]\,\emptyset$:

  (6) Now we will show:
  $(\mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\, \mathbf{v_1}')\ \mathbf{in\ v_1}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id}),$
  $\mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\, \mathbf{v_2}')\ \mathbf{in\ v_2}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id})) \in \mathcal{E}[\![\sigma^+]\!]\,\emptyset$:

  (7) Now we will show:
  $(\boldsymbol{\lambda}(\mathbf{z} : \sigma^+).\, \mathbf{v_1}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id}), \boldsymbol{\lambda}(\mathbf{z} : \sigma^+).\, \mathbf{v_2}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id})) \in \mathcal{E}[\![\sigma^+]\!]\,\emptyset$:

  - Consider arbitrary $\mathbf{v_1}''$ and $\mathbf{v_2}''$ such that $(\mathbf{v_1}'', \mathbf{v_2}'') \in \mathcal{V}[\![\sigma^+]\!]\,\emptyset$.
  - Must show: $(\mathbf{v_1}\ [\sigma'^+]\ (\mathbf{v_1}'', \mathbf{id}), \mathbf{v_2}\ [\sigma'^+]\ (\mathbf{v_2}'', \mathbf{id})) \in \mathcal{E}\left[\!\!\left[\sigma'^+\right]\!\!\right]\,\emptyset$:
  - Instantiating (5) with σ'^+, σ'^+, and $R = \mathcal{V}\left[\!\!\left[\sigma'^+\right]\!\!\right]\,\emptyset$:
    - Now we will show:
      $((\mathbf{v_1}'', \mathbf{id}), (\mathbf{v_2}'', \mathbf{id})) \in \mathcal{V}\left[\!\!\left[\sigma^+ \times \sigma'^+ \to \boldsymbol{\alpha}\right]\!\!\right]\,[\boldsymbol{\alpha} \mapsto (\sigma'^+, \sigma'^+, R)]$:
      - (Definition of $\mathcal{V}[\![\cdot \times \cdot]\!]\,\cdot$.)
    - Let $\mathbf{v_1} = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\ (\mathbf{y} : \sigma^+ \times \sigma'^+ \to \boldsymbol{\alpha}).\, \mathbf{e_1}'$.
    - Let $\mathbf{v_2} = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\ (\mathbf{y} : \sigma^+ \times \sigma'^+ \to \boldsymbol{\alpha}).\, \mathbf{e_2}'$.
    - Have:
      $(\mathbf{e_1}'[\sigma'^+/\boldsymbol{\alpha}][(\mathbf{v_1}'', \mathbf{id})/\mathbf{y}],$
      $\mathbf{e_2}'[\sigma'^+/\boldsymbol{\alpha}][(\mathbf{v_2}'', \mathbf{id})/\mathbf{y}]) \in \mathcal{E}[\![\boldsymbol{\alpha}]\!]\,[\boldsymbol{\alpha} \mapsto (\sigma'^+, \sigma'^+, R)]$
      by definition of $\mathcal{V}[\![\forall\,[\cdot].\cdot \to \cdot]\!]\,\cdot$.
    - Have: $(\mathbf{e_1}'[\sigma'^+/\boldsymbol{\alpha}][(\mathbf{v_1}'', \mathbf{id})/\mathbf{y}], \mathbf{e_2}'[\sigma'^+/\boldsymbol{\alpha}][(\mathbf{v_2}'', \mathbf{id})/\mathbf{y}]) \in \mathcal{E}[\![\boldsymbol{\alpha}]\!]\,\emptyset$.

  - Have:
    $(\mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\, \mathbf{v_1}')\ \mathbf{in\ v_1}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id}),$
    $\mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\, \mathbf{v_2}')\ \mathbf{in\ v_2}\ [\sigma'^+]\ (\mathbf{z}, \mathbf{id})) \in \mathcal{E}[\![\sigma^+]\!]\,\emptyset$
    by induction hypothesis Part II applied to (5) and (7).

  - Have: $(\mathsf{v}_1^f, \mathsf{v}_2^f) \in \mathcal{V}[\![\sigma \to \sigma']\!]\,\emptyset$ by induction hypothesis Part I applied to and (6).

Proof of Part II:

Case $\sigma = \mathsf{bool}$:

(8) Have: $(\mathsf{e}_1, \mathsf{e}_2) \in \mathcal{E}\,[\![\mathsf{bool}]\!]\,\emptyset$.

(9) Have: $(\boldsymbol{\lambda}(\mathbf{x}\,\mathbf{:}\,\mathbf{bool}).\,\mathbf{e}_1{}', \boldsymbol{\lambda}(\mathbf{x}\,\mathbf{:}\,\mathbf{bool}).\,\mathbf{e}_2{}') \in \mathcal{V}\,[\![\mathbf{bool} \rightarrow \boldsymbol{\tau}]\!]\,\rho$.

- Must show: $(\mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{e}_1)\ \mathbf{in}\ \mathbf{e}_1{}', \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{e}_2)\ \mathbf{in}\ \mathbf{e}_2{}') \in \mathcal{E}\,[\![\boldsymbol{\tau}]\!]\,\rho$:

(10) Consider arbitrary $\mathbf{v}_1^f$ such that $\mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{e}_1)\ \mathbf{in}\ \mathbf{e}_1{}' \longmapsto^* \mathbf{v}_1^f$.

- Must show: $\exists \mathbf{v}_2^f . \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{e}_2)\ \mathbf{in}\ \mathbf{e}_2{}' \longmapsto^* \mathbf{v}_2^f$ and $(\mathbf{v}_1^f, \mathbf{v}_2^f) \in \mathcal{V}\,[\![\boldsymbol{\tau}]\!]\,\rho$:
- Have: $\mathsf{e}_1 \longmapsto^* \mathsf{v}_1$ by (10) and operational semantics.
- Instantiating (8) with v_1:
  - Have: v_2 such that $\mathsf{e}_2 \longmapsto^* \mathsf{v}_2$ and $(\mathsf{v}_1, \mathsf{v}_2) \in \mathcal{V}\,[\![\mathsf{bool}]\!]\,\emptyset$.
  - Have: $\mathsf{v}_1 = \mathsf{v}_2 = \mathsf{true}$ or $\mathsf{v}_1 = \mathsf{v}_2 = \mathsf{false}$ by definition of $\mathcal{V}\,[\![\mathsf{bool}]\!]\,\cdot$.

  Case $\mathsf{v}_1 = \mathsf{v}_2 = \mathsf{true}$:

  - Have:
    $\mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{e}_1)\ \mathbf{in}\ \mathbf{e}_1{}'$
    $\longmapsto^* \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{true})\ \mathbf{in}\ \mathbf{e}_1{}'$
    $\longmapsto \mathbf{e}_1{}'[\mathbf{true}/\mathbf{x}] \longmapsto^* \mathbf{v}_1^f$
    by operational semantics.
  - Have: $\mathbf{let}\ \mathbf{x}{=}(\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{e}_2)\ \mathbf{in}\ \mathbf{e}_2{}' \longmapsto^* \mathbf{let}\ \mathbf{x}{=}(\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{true})\ \mathbf{in}\ \mathbf{e}_2{}' \longmapsto \mathbf{e}_1{}'[\mathbf{true}/\mathbf{x}]$
    by operational semantics.
  - Instantiating (9) with $\mathbf{true}$ and $\mathbf{true}$:

    (11) Have: $(\mathbf{e}_1{}'[\mathbf{true}/\mathbf{x}], \mathbf{e}_2{}'[\mathbf{true}/\mathbf{x}]) \in \mathcal{E}\,[\![\boldsymbol{\tau}]\!]\,\rho$.

    - Observe that: $\mathbf{e}_1{}'[\mathbf{true}/\mathbf{x}] \longmapsto^* \mathbf{v}_1^f$.
    - Have: $\mathbf{v}_2^f$ such that $\mathbf{e}_2{}'[\mathbf{true}/\mathbf{x}] \longmapsto^* \mathbf{v}_2^f$ and $(\mathbf{v}_1^f, \mathbf{v}_2^f) \in \mathcal{E}\,[\![\boldsymbol{\tau}]\!]\,\rho$ by (11).

  Case $\mathsf{v}_1 = \mathsf{v}_2 = \mathsf{false}$:

  (Similar to true case.)

Case $\sigma = \sigma \rightarrow \sigma'$:

(12) Have: $(\mathsf{e}_1, \mathsf{e}_2) \in \mathcal{E}\,[\![\sigma \rightarrow \sigma']\!]\,\emptyset$.

(13) Have:
$(\boldsymbol{\lambda}(\mathbf{x}\,\mathbf{:}\,\forall\,[\boldsymbol{\alpha}].\sigma^+ \times (\sigma'^+ \rightarrow \boldsymbol{\alpha}) \rightarrow \boldsymbol{\alpha}).\,\mathbf{e}_1{}',$.
$\boldsymbol{\lambda}(\mathbf{x}\,\mathbf{:}\,\forall\,[\boldsymbol{\alpha}].\sigma^+ \times (\sigma'^+ \rightarrow \boldsymbol{\alpha}) \rightarrow \boldsymbol{\alpha}).\,\mathbf{e}_2{}') \in \mathcal{V}\,[\![\sigma^+ \rightarrow \boldsymbol{\tau}]\!]\,\rho$.

- Must show: $(\mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma \rightarrow \sigma'}\ \mathsf{e}_1)\ \mathbf{in}\ \mathbf{e}_1{}', \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma \rightarrow \sigma'}\ \mathsf{e}_2)\ \mathbf{in}\ \mathbf{e}_2{}') \in \mathcal{E}\,[\![\boldsymbol{\tau}]\!]\,\rho$:
- Consider arbitrary $\mathbf{v}_1^f$ such that $\mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma \rightarrow \sigma'}\ \mathsf{e}_1)\ \mathbf{in}\ \mathbf{e}_1{}' \longmapsto^* \mathbf{v}_1^f$.
- Must show: $\exists \mathbf{v}_2^f . \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma \rightarrow \sigma'}\ \mathsf{e}_2)\ \mathbf{in}\ \mathbf{e}_2{}' \longmapsto^* \mathbf{v}_2^f$ and $(\mathbf{v}_1^f, \mathbf{v}_2^f) \in \mathcal{V}\,[\![\boldsymbol{\tau}]\!]\,\rho$:
- Have: $\mathsf{e}_1 \longmapsto^* \mathsf{v}_1$ by operational semantics.
- Instantiating (12) with v_1:
  - Have: v_2 such that $\mathsf{e}_2 \longmapsto^* \mathsf{v}_2$ and $(\mathsf{v}_1, \mathsf{v}_2) \in \mathcal{V}\,[\![\sigma \rightarrow \sigma']\!]\,\emptyset$.
  - Let $\mathbf{v}_1{}'' = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\ ((\mathbf{y}, \mathbf{k})\,\mathbf{:}\,\sigma^+ \times (\sigma'^+ \rightarrow \boldsymbol{\alpha})).\ \mathbf{let}\ \mathbf{z} = (\mathcal{TS}^{\,\sigma'}\ (\mathsf{v}_1\ {}^{\sigma}\mathcal{ST}\ \mathbf{y}))\ \mathbf{in}\ \mathbf{k}\ \mathbf{z}$.
  - Let $\mathbf{v}_2{}'' = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\ ((\mathbf{y}, \mathbf{k})\,\mathbf{:}\,\sigma^+ \times (\sigma'^+ \rightarrow \boldsymbol{\alpha})).\ \mathbf{let}\ \mathbf{z} = (\mathcal{TS}^{\,\sigma'}\ (\mathsf{v}_2\ {}^{\sigma}\mathcal{ST}\ \mathbf{y}))\ \mathbf{in}\ \mathbf{k}\ \mathbf{z}$.
  - Have:
    $\mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma \rightarrow \sigma'}\ \mathsf{e}_1)\ \mathbf{in}\ \mathbf{e}_1{}' \longmapsto^* \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma \rightarrow \sigma'}\ \mathsf{v}_1)\ \mathbf{in}\ \mathbf{e}_1{}' \longmapsto \mathbf{e}_1{}'[\mathbf{v}_1{}''/\mathbf{x}] \longmapsto^* \mathbf{v}_1^f$
    by operational semantics.

- Have: $\mathbf{let\ x} = (\mathcal{TS}^{\sigma \to \sigma'}\ e_2)\ \mathbf{in\ e_2'} \longmapsto^* \mathbf{let\ x} = (\mathcal{TS}^{\sigma \to \sigma'}\ v_2)\ \mathbf{in\ e_2'} \longmapsto \mathbf{e_2'}[\mathbf{v_2''}/\mathbf{x}]$ by operational semantics.

(14) Now we will show: $(\mathbf{v_1''}, \mathbf{v_2''}) \in \mathcal{V}\left[\!\!\left[\forall\, [\boldsymbol{\alpha}].\sigma^+ \times (\sigma'^+ \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}\right]\!\!\right] \rho$:

  - Consider arbitrary $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$, and R such that $R \in \mathrm{Rel}[\boldsymbol{\tau}_1, \boldsymbol{\tau}_2]$.

  (15) Consider arbitrary $\mathbf{v}_{y_1}$, $\mathbf{v}_{y_2}$, $\mathbf{v}_{k1}$, and $\mathbf{v}_{k2}$ such that $((\mathbf{v}_{y_1}, \mathbf{v}_{k1}), (\mathbf{v}_{y_2}, \mathbf{v}_{k2})) \in \mathcal{V}\left[\!\!\left[\sigma^+ \times (\sigma'^+ \to \boldsymbol{\alpha})\right]\!\!\right] [\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$.

  (16) Consider arbitrary $\mathbf{v}_1^{f}{}'$ such that $\mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \mathbf{v}_1\ ({}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_1}))\ \mathbf{in\ v}_{k1}\ \mathbf{z} \longmapsto^* \mathbf{v}_1^{f}{}'$.

  (17) Must show: $\exists \mathbf{v}_2^{f}{}'.\mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \mathbf{v}_2^{f}{}'\ ({}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_2}))\ \mathbf{in\ v}_{k2}\ \mathbf{z} \longmapsto^* \mathbf{v}_2^{f}{}'$ and $(\mathbf{v}_1^{f}{}', \mathbf{v}_2^{f}{}') \in \mathcal{V}[\![\boldsymbol{\alpha}]\!]\, [\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$:

  (18) Now we will show: $(v_1\ ({}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_1}), v_2\ ({}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_2})) \in \mathcal{E}[\![\sigma']\!]\, \emptyset$:

    - Must show: $(v_1\ ({}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_1}), v_2\ ({}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_2})) \in \mathcal{E}[\![\sigma']\!]\, \emptyset$:

      - Have: v_{y_1}' such that ${}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_1} \longmapsto^* v_{y_1}'$ by operational semantics on (16).

      (19) Must show: $(\mathbf{v}_{y_1}, \mathbf{v}_{y_2}) \in \mathcal{V}[\![\sigma^+]\!]\, \emptyset$:

        - Have: $(\mathbf{v}_{y_1}, \mathbf{v}_{y_2}) \in \mathcal{V}[\![\sigma^+]\!]\, \emptyset \equiv (\mathbf{v}_{y_1}, \mathbf{v}_{y_2}) \in \mathcal{V}[\![\sigma^+]\!]\, [\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$ by $\boldsymbol{\alpha} \notin \mathrm{ftv}\,\sigma^+$.

      - Have: $(v_1\ ({}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_1}), v_2\ ({}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_2})) \in \mathcal{E}[\![\sigma]\!]\, \emptyset$ by induction hypothesis Part I applied to (19).

(20) Now we will show: $(\boldsymbol{\lambda}(\mathbf{z} : \sigma'^+).\ \mathbf{v}_{k1}\ \mathbf{z}, \boldsymbol{\lambda}(\mathbf{z} : \sigma'^+).\ \mathbf{v}_{k2}\ \mathbf{z}) \in \mathcal{V}\left[\!\!\left[\sigma'^+ \to \boldsymbol{\alpha}\right]\!\!\right] [\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$:

  - Consider arbitrary $\mathbf{v}_{z1}$ and $\mathbf{v}_{z2}$ such that $(\mathbf{v}_{z1}, \mathbf{v}_{z2}) \in \mathcal{V}[\![\sigma^+]\!]\, [\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$.
  - Must show: $(\mathbf{v}_{k1}\ \mathbf{v}_{z1}, \mathbf{v}_{k2}\ \mathbf{v}_{z2}) \in \mathcal{E}[\![\boldsymbol{\alpha}]\!]\, [\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$:
  - (Immediate from (15).)

- Have: $(\mathbf{e_1'}[\mathbf{v_1''}/\mathbf{x}], \mathbf{e_2'}[\mathbf{v_2''}/\mathbf{x}]) \in \mathcal{E}[\![\boldsymbol{\tau}]\!]\, \rho$ by (14) and (13).
- Have: $\mathbf{v}_2^{f}$ such that $\mathbf{e_2'}[\mathbf{v_2''}/\mathbf{x}] \longmapsto^* \mathbf{v}_2^{f}$ and $(\mathbf{v}_1^{f}, \mathbf{v}_2^{f}) \in \mathcal{V}[\![\boldsymbol{\tau}]\!]\, \rho$.
- Have: $\exists \mathbf{v}_2^{f}{}'.\mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \mathbf{v}_2^{f}{}'\ ({}^{\sigma}\mathcal{ST}\ \mathbf{v}_{y_2}))\ \mathbf{in\ v}_{k2}\ \mathbf{z} \longmapsto^* \mathbf{v}_2^{f}{}'$
and $(\mathbf{v}_1^{f}{}', \mathbf{v}_2^{f}{}') \in \mathcal{V}[\![\boldsymbol{\alpha}]\!]\, [\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$ by induction hypothesis Part II applied to (18) and (20).

  - (This is (17)—what we are meant to show.)

$\square$

Lemma 8.17 (Compatibility Source Embed)
If $\Delta; \Gamma \vdash \mathbf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{e}_2 : \sigma^+$
then $\Delta; \Gamma \vdash {}^{\sigma}\mathcal{ST}\ \mathbf{e}_1 \precsim^{log}_{\mathrm{ST}} {}^{\sigma}\mathcal{ST}\ \mathbf{e}_2 : \sigma$.

Proof

Follows from the Part (I) of the Bridge Lemma (Lemma 8.16). ☐

Lemma 8.18 (Compatibility Target Embed)
If $\Delta; \Gamma \vdash \mathsf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathsf{e}_2 : \sigma$ *and* $\Delta; \Gamma, \mathbf{x} : \sigma^+ \vdash \mathbf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{e}_2 : \boldsymbol{\tau}$
then $\Delta; \Gamma \vdash \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma}\ \mathsf{e}_1)\ \mathbf{in}\ \mathbf{e}_1 \precsim^{log}_{\mathrm{ST}} \mathbf{let}\ \mathbf{x} = (\mathcal{TS}^{\,\sigma}\ \mathsf{e}_2)\ \mathbf{in}\ \mathbf{e}_2 : \boldsymbol{\tau}$.

Proof

Follows from Part (II) of the Bridge Lemma (Lemma 8.16). ☐

Theorem 8.19 (λ^{ST} Fundamental Property)
If $\Delta; \Gamma \vdash e : \varphi$ *then* $\Delta; \Gamma \vdash e \precsim^{log}_{\mathrm{ST}} e : \varphi$.

Proof

By induction on the derivation $\Delta; \Gamma \vdash e : \varphi$.

Each case follows from the corresponding compatibility lemma. ☐

8.2 λ^{ST} Logical Relation: Soundness w.r.t. Contextual Equivalence

Note: The proof of soundness w.r.t. contextual equivalence is fairly straightforward. It follows the same structure as the proofs in Ahmed's earlier work on step-indexed logical relations [2, 3], except that here the logical relations are not step-indexed so the proofs are even easier. Here we only give an outline of the proof. For all the details of how the proof proceeds, we refer the reader to Ahmed's earlier technical report [3], Section C.10 (pages 123 to 130).

To prove that our logical relation ($\precsim^{log}_{\mathrm{ST}}$) is sound with respect to contextual equivalence ($\precsim^{ctx}_{\mathrm{ST}}$), we first define what it means for two contexts to be logically related as follows:

Definition 8.20 (Logically Related Contexts)

$$\begin{array}{lcl} \vdash C_1 \precsim^{log}_{\mathrm{ST}} C_2 : (\Delta; \Gamma \vdash \varphi) \Rightarrow (\Delta'; \Gamma' \vdash \varphi') & \stackrel{\mathrm{def}}{=} & \forall e_1, e_2.\ \ \Delta; \Gamma \vdash e_1 \precsim^{log}_{\mathrm{ST}} e_2 : \varphi \implies \\ & & \qquad \Delta'; \Gamma' \vdash C_1[e_1] \precsim^{log}_{\mathrm{ST}} C_2[e_2] : \varphi' \end{array}$$

Next, we prove that if a context is well typed, then it is logically related to itself.

Lemma 8.21
If $\vdash C : (\Delta; \Gamma \vdash \varphi) \Rightarrow (\Delta'; \Gamma' \vdash \varphi')$, *then* $\vdash C \precsim^{log}_{\mathrm{ST}} C : (\Delta; \Gamma \vdash \varphi) \Rightarrow (\Delta'; \Gamma' \vdash \varphi')$.

Proof

By induction on the derivation of $\vdash C : (\Delta; \Gamma \vdash \varphi) \Rightarrow (\Delta'; \Gamma' \vdash \varphi')$.
Each case follows from the appropriate compatibility lemmas in Section 8.1 (i.e., Lemmas 8.1 to 8.15 plus Lemmas 8.17 and 8.18). □

Theorem 8.22 (Soundness wrt Contextual Equivalence ($\precsim^{log}_{\mathrm{ST}} \subseteq \precsim^{ctx}_{\mathrm{ST}}$))
If $\Delta; \Gamma \vdash e_1 \precsim^{log}_{\mathrm{ST}} e_2 : \varphi$ *then* $\Delta; \Gamma \vdash e_1 \precsim^{ctx}_{\mathrm{ST}} e_2 : \varphi$.

Proof

Straightforward. Follows from Lemma 8.21.
See [3] Section C.10 (page 130) for details. □

8.3 λ^{ST} Logical Relation: Completeness w.r.t. Contextual Equivalence

Note: The proof of completeness w.r.t. contextual equivalence is also quite straightforward. It follows the same structure as the proofs in Ahmed's earlier work on step-indexed logical relations [2, 3], except that here the logical relations are not step-indexed so the proofs are much simpler. Here we only sketch the main lemmas required for the proof. For all the details of how the proof proceeds, we refer the reader to Ahmed's earlier technical report [3], Sections E.2 and E.3 (pages 151 to 164).

Lemma 8.23 **($\precsim^{log}_{\mathrm{ST}}$ is Equivalence Respecting)**
Let $\rho \in \mathcal{D}[\![\Delta]\!]$ and $\Delta \vdash \varphi$.
If $(v_1, v_2) \in \mathcal{V}[\![\varphi]\!]\rho$ and $v_2 \precsim^{ciu}_{\mathrm{ST}} v_3 : \rho_2(\varphi)$,
then $(v_1, v_3) \in \mathcal{V}[\![\varphi]\!]\rho$.

Proof

By induction on the structure of the derivation $\Delta \vdash \varphi$.
When $\varphi = \boldsymbol{\alpha}$, the proof follows from the definition of Rel (since $\mathcal{V}[\![\alpha]\!]\rho = R$, where we have that $R \in \mathrm{Rel}[\rho_1(\boldsymbol{\alpha}), \rho_2(\boldsymbol{\alpha})]$).
The rest of the cases are entirely straightforward. □

Lemma 8.24 **($\precsim^{ctx}_{\mathrm{ST}}$ Congruence)**
If $\Delta; \Gamma \vdash e_1 \precsim^{ctx}_{\mathrm{ST}} e_2 : \varphi$ and $\vdash C : (\Delta; \Gamma \vdash \varphi) \Rightarrow (\Delta'; \Gamma' \vdash \varphi')$,
then $\Delta'; \Gamma' \vdash C[e_1] \precsim^{ctx}_{\mathrm{ST}} C[e_2] : \varphi'$.

Proof

Easy, standard proof. See [3] Section E.3 (page 160) for details. □

Now, the proof proceeds in two parts. First we prove that if two terms are contextually related, then they are ciu related. Then we prove that if two terms are ciu-related, then they are logically related.

Lemma 8.25 **($\precsim^{ctx}_{\mathrm{ST}} \subseteq \precsim^{ciu}_{\mathrm{ST}}$)**
If $\Delta; \Gamma \vdash e_1 \precsim^{ctx}_{\mathrm{ST}} e_2 : \varphi$ then $\Delta; \Gamma \vdash e_1 \precsim^{ciu}_{\mathrm{ST}} e_2 : \varphi$.

Proof

Straightforward. Follows from Lemma 8.24 and the fact that every evaluation context E is a program context.
See [3] Section E.3 (page 161) for details. □

Lemma 8.26 **($\precsim^{ciu}_{\mathrm{ST}} \subseteq \precsim^{log}_{\mathrm{ST}}$)**
If $\Delta; \Gamma \vdash e_1 \precsim^{ciu}_{\mathrm{ST}} e_2 : \varphi$ then $\Delta; \Gamma \vdash e_1 \precsim^{log}_{\mathrm{ST}} e_2 : \varphi$.

Proof

Follows easily from Lemmas 8.19 (Fundamental Property) and 8.23.
See [3] Section E.3 (page 162-164) for details. □

Theorem 8.27 (Completeness wrt Contextual Equivalence ($\precsim^{ctx}_{\mathrm{ST}} \subseteq \precsim^{log}_{\mathrm{ST}}$))
If $\Delta; \Gamma \vdash e_1 \precsim^{ctx}_{\mathrm{ST}} e_2 : \varphi$ *then* $\Delta; \Gamma \vdash e_1 \precsim^{log}_{\mathrm{ST}} e_2 : \varphi$.

Proof

Immediate from Lemmas 8.25 and 8.26 above. □

9 Part (2): CPS Translation Preserves Equivalence in λ^{ST}

In this section, we prove that if two terms are equivalent in λ^{ST}, then their CPS translations are equivalent in λ^{ST}.
The bulk of the work involves showing that CPS translation is equivalent to embedding using boundaries. Essentially, this follows from the proof that that CPS translation is semantics preserving (correct) and semantics reflecting.

Before we can prove that CPS translation is semantics preserving and reflecting, we need to establish two key properties.

The first is *boundary cancellation*, which essentially says that if you embed $\mathbf{e}$ into the source using $\mathcal{ST}$, and then embed that into the target using $\mathcal{TS}$, the resulting term is contextually equivalent to the original. Analogously, embedding e into the target via $\mathcal{TS}$ and then embedding the latter into the source via $\mathcal{ST}$, also results in a term that is contextually equivalent to the original.

Lemma 9.1 (Boundary Cancellation)

- *Let* $\Delta;\Gamma \vdash \mathsf{e} : \sigma$*. Then* $\Delta;\Gamma \vdash \mathsf{e} \approx^{log}_{\mathrm{ST}} {}^{\sigma}\mathcal{ST}\,(\mathcal{TS}^{\sigma}\,\mathsf{e}) : \sigma$*.*
- *Let* $\Delta;\Gamma \vdash \mathbf{e} : \sigma^{+}$*. Then* $\Delta;\Gamma \vdash \mathbf{e} \approx^{log}_{\mathrm{ST}} \mathcal{TS}^{\sigma}\,({}^{\sigma}\mathcal{ST}\,\mathbf{e}) : \sigma^{+}$*.*

Proof

By induction on the structure of σ. □

The second key property is a *free theorem* [44] regarding terms of (computation) type $\forall\,[\alpha].(\sigma^{+}\rightarrow\alpha)\rightarrow\alpha$. The following free theorem captures the essence of what we gain from switching from a CPS type translation that makes use of a global answer type, to one that makes each continuation's answer type individually abstract: namely, that a computation (or function) of the above type *must* invoke its continuation at least once, and that it does not matter (in our purely functional setting) if it invokes it more than once. A similar theorem is given in Wadler [44]. We take a notational liberty in the initial statement of this theorem, which we discuss next.

Free Theorem: Continuation Shuffling
Let $\Delta;\Gamma \vdash \mathbf{v}_f : \forall\,[\alpha].(\tau_1 \times (\tau_2\rightarrow\alpha))\rightarrow\alpha$, $\Delta;\Gamma \vdash \mathbf{v}_1 : \tau_1$, and $\Delta;\Gamma \vdash \mathbf{v}_k : \tau_2\rightarrow\tau$.
Then $\Delta;\Gamma \vdash \mathbf{v}_f\ [\tau]\ (\mathbf{v}_1, \mathbf{v}_k) \approx^{log}_{\mathrm{ST}} \mathbf{v}_k\ (\mathbf{v}_f\ [\tau_2]\ (\mathbf{v}_1, \mathbf{id})) : \tau$.

Notice that $\mathbf{v}_k\ (\mathbf{v}_f\ [\tau]\ \mathbf{id})$ is not a syntactically well-formed term in λ^{ST}! We use this essentially as shorthand to avoid a much longer (and less intuitive) statement of the theorem. Strictly speaking, the above "lemma" should be stated as follows. The intuition here is that we close off the expression $\mathbf{v}_f\ [\tau]\ \mathbf{id}$ with appropriate type and term substitutions and evaluate it to get a value $\mathbf{v}$ that we then pass to the (appropriately closed) continuation $\mathbf{v}_k$. The two clauses are required because our underlying relation $\mathcal{E}[\![\cdot]\!]$ is an approximation while what we want here is equivalence.

Lemma 9.2 (Free Theorem: Continuation Shuffling)
Let $\Delta;\Gamma \vdash \mathbf{v} : \forall\,[\alpha].(\tau_1 \times (\tau_2\rightarrow\alpha))\rightarrow\alpha$*,* $\Delta;\Gamma \vdash \mathbf{v}_1 : \tau_1$*, and* $\Delta;\Gamma \vdash \mathbf{k} : \tau_2\rightarrow\tau$*.*

$$\begin{array}{l}
\textit{Then: } \forall\rho \in \mathcal{D}[\![\Delta]\!]\,.\ \forall(\gamma_1,\gamma_2) \in \mathcal{G}[\![\Gamma]\!]\,\rho. \\
\quad \textit{if } \rho_2(\gamma_2(\mathbf{v}_f\ [\tau]\ \mathbf{id})) \longmapsto^{*} \mathbf{v} \textit{ then} \\
\quad\quad (\rho_1(\gamma_1(\mathbf{v}_f\ [\tau_k]\ \mathbf{v}_k)), (\rho_2(\gamma_2(\mathbf{v}_k)))\ \mathbf{v}) \in \mathcal{E}[\![\tau_k]\!]\,\rho \\
\quad \textit{and} \\
\quad \textit{if } \rho_1(\gamma_1(\mathbf{v}_f\ [\tau]\ \mathbf{id})) \longmapsto^{*} \mathbf{v} \textit{ then} \\
\quad\quad ((\rho_1(\gamma_1(\mathbf{v}_k)))\ \mathbf{v}, \rho_2(\gamma_2(\mathbf{v}_f\ [\tau_k]\ \mathbf{v}_k))) \in \mathcal{E}[\![\tau_k]\!]\,\rho.
\end{array}$$

See Wadler [44], Section 3.8.

With the boundary cancellation and continuation-shuffling (free) theorem in hand, we can prove that our CPS translation preserves and reflects semantics. In the statement of these lemmas, we will have a λ^{S} term e on one side and its translation, a λ^{T} term $\mathbf{v}$, on the other side. Wherever e contains a variable $\mathsf{x} : \sigma$, $\mathbf{v}$ will have the variable $\mathbf{x} : \sigma^{+}$. Therefore, we will need related (source and target) substitutions γ_{S} and γ_{T} to obtain closed terms.

Definition 9.3
Let Γ be a mapping from variables x to types σ. Let γ_{S} be a substitution mapping variables x to (closed) values v. Let γ_{T} be a substitution mapping variables $\mathbf{x}$ to (closed) values $\mathbf{v}$.
We define $\Gamma \vdash \gamma_{\mathrm{S}} \lesssim \gamma_{\mathrm{T}}$ as follows:

$$\cdot \vdash \emptyset \lesssim \emptyset \quad \text{iff} \quad \textit{(unconditionally)}$$
$$\Gamma, \mathsf{x} : \sigma \vdash \gamma_{\mathrm{S}}, \mathsf{x} \mapsto \mathsf{v} \lesssim \gamma_{\mathrm{T}}, \mathbf{x} \mapsto \mathbf{v} \quad \text{iff} \quad \Gamma \vdash \gamma_{\mathrm{S}} \lesssim \gamma_{\mathrm{T}} \ \wedge\ \vdash \mathsf{v} \precsim^{log}_{\mathrm{ST}} {}^{\sigma}\mathcal{ST}\, \mathbf{v} : \sigma$$

We define $\Gamma \vdash \gamma_{\mathrm{S}} \gtrsim \gamma_{\mathrm{T}}$ as follows:

$$\cdot \vdash \emptyset \gtrsim \emptyset \quad \text{iff} \quad \textit{(unconditionally)}$$
$$\Gamma, \mathsf{x} : \sigma \vdash \gamma_{\mathrm{S}}, \mathsf{x} \mapsto \mathsf{v} \gtrsim \gamma_{\mathrm{T}}, \mathbf{x} \mapsto \mathbf{v} \quad \text{iff} \quad \Gamma \vdash \gamma_{\mathrm{S}} \gtrsim \gamma_{\mathrm{T}} \ \wedge\ \vdash {}^{\sigma}\mathcal{ST}\, \mathbf{v} \precsim^{log}_{\mathrm{ST}} \mathsf{v} : \sigma$$

We define $\Gamma \vdash \gamma_{\mathrm{S}} \simeq \gamma_{\mathrm{T}}$ as follows:

$$\Gamma \vdash \gamma_{\mathrm{S}} \simeq \gamma_{\mathrm{T}} \quad \text{iff} \quad \Gamma \vdash \gamma_{\mathrm{S}} \lesssim \gamma_{\mathrm{T}} \ \wedge\ \Gamma \vdash \gamma_{\mathrm{S}} \gtrsim \gamma_{\mathrm{T}}$$

Notice that in the definition of $\Gamma \vdash \gamma_{\mathrm{S}} \gtrsim \gamma_{\mathrm{T}}$, we require $\vdash {}^{\sigma}\mathcal{ST}\, \mathbf{v} \precsim^{log}_{\mathrm{ST}} \mathsf{v} : \sigma$. Using boundary cancellation and the compatibility lemmas for boundaries, we can conclude that this is equivalent to $\vdash \mathbf{v} \precsim^{log}_{\mathrm{ST}} \mathcal{TS}^{\sigma}\, \mathsf{v} : \sigma^{+}$. (This observation might make it slightly easier to understand the statement of Lemma 9.5.)

Informally, the following lemma says that if e evaluates to some value v_1, then its CPS translation, when applied to the identity continuation will evaluate to some $\mathbf{v}_2 : \sigma^{+}$ that can be converted to a source value v_2 such that v_1 and v_2 are related at σ.

Lemma 9.4 (CPS is semantics preserving)
If $\Gamma \vdash \mathsf{e} : \sigma \rightsquigarrow \mathbf{v}$ and $\Gamma \vdash \gamma_{\mathrm{S}} \lesssim \gamma_{\mathrm{T}}$
then $\vdash \gamma_{\mathrm{S}}(\mathsf{e}) \precsim^{log}_{\mathrm{ST}} {}^{\sigma}\mathcal{ST}\, (\gamma_{\mathrm{T}}\,(\mathbf{v})\ \ [\sigma^{+}]\ \mathbf{id}) : \sigma$.

Proof

Proof by induction on the structure of e.

Case $\mathsf{e} = \mathsf{true}$:

- Must show: $\vdash \gamma_{\mathrm{S}}(\mathsf{true}) \precsim^{log}_{\mathrm{ST}} \gamma_{\mathrm{T}}\left({}^{\mathsf{bool}}\mathcal{ST}\,(\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \mathbf{bool} \to \alpha).\,\mathbf{k}\ \mathbf{true})\ [\mathbf{bool}]\ \mathbf{id}\right) : \mathsf{bool}$:
- Observe that: $\gamma_{\mathrm{S}}(\mathsf{true}) = \mathsf{true}$.
- Have:
  $\gamma_{\mathrm{T}}({}^{\mathsf{bool}}\mathcal{ST}\,(\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \mathbf{bool} \to \alpha).\,\mathbf{k}\ \mathbf{true})\ [\mathbf{bool}]\ \mathbf{id})\mathsf{bool}$
  $= {}^{\mathsf{bool}}\mathcal{ST}\,(\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \mathbf{bool} \to \alpha).\,\mathbf{k}\ \mathbf{true})\ [\mathbf{bool}]\ \mathbf{id}$
  $\longmapsto^{*} {}^{\mathsf{bool}}\mathcal{ST}\,\mathbf{true}$
  $\longmapsto \mathsf{true}$
  by operational semantics.

Case $\mathsf{e} = \mathsf{false}$:

(Similar to true case.)

Case $\mathsf{e} = \mathsf{if\ e\ then\ e_1\ else\ e_2}$:

- Have: $\Gamma \vdash \mathsf{e_1} : \mathsf{bool} \rightsquigarrow \mathbf{v}_1$.
- Have: $\Gamma \vdash \mathsf{e}'_1 : \sigma \rightsquigarrow \mathbf{v_2}'$.
- Have: $\Gamma \vdash \mathsf{e}''_1 : \sigma \rightsquigarrow \mathbf{v_2}''$.

(1) Have: $\vdash \gamma_S(\mathsf{e_1}) \precsim^{log}_{ST} \gamma_T({}^{\mathsf{bool}}\mathcal{ST}\ \mathbf{v_2}\ [\mathbf{bool}]\ \mathbf{id}) : \mathsf{bool}$.

(2) Have: $\vdash \gamma_S(\mathsf{e}'_1) \precsim^{log}_{ST} \gamma_T({}^{\mathsf{bool}}\mathcal{ST}\ \mathbf{v_2}'\ [\sigma^+]\ \mathbf{id}) : \sigma$.

- Have: $\vdash \gamma_S(\mathsf{e}''_1) \precsim^{log}_{ST} \gamma_T({}^{\mathsf{bool}}\mathcal{ST}\ \mathbf{v_2}''\ [\sigma^+]\ \mathbf{id}) : \sigma$.
- Must show:
  $\vdash \gamma_S(\mathsf{if\ e_1\ then\ e'_1\ else\ e''_1})$
  $\precsim^{log}_{ST} \gamma_T({}^{\sigma}\mathcal{ST}\ (\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \mathbf{bool} \to \alpha).\ \mathbf{v_2}\ [\alpha]\ (\boldsymbol{\lambda}(\mathbf{x} : \mathbf{bool}).$
  $\mathbf{if\ x\ then\ v_2'}\ [\alpha]\ \mathbf{k\ else\ v_2''}\ [\alpha]\ \mathbf{k}\ [\sigma^+]\ \mathbf{id}))) : \sigma$
- Have: $\gamma_S(\mathsf{if\ e_1\ then\ e'_1\ else\ e''_1}) \longmapsto^* \mathsf{v}^f_1$.
- Must show:
  $\exists \mathsf{v}^f_2.\gamma_T({}^{\sigma}\mathcal{ST}\ (\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \mathbf{bool} \to \alpha).\ \mathbf{v_2}\ [\alpha]\ (\boldsymbol{\lambda}(\mathbf{x} : \mathbf{bool}).$
  $\mathbf{if\ x\ then\ v_2'}\ [\alpha]\ \mathbf{k\ else\ v_2''}\ [\alpha]\ \mathbf{k}))) \ [\sigma^+]\ \mathbf{id}$
  $\longmapsto^* \mathsf{v}^f_2$:
- Have: $\exists \mathsf{v_1}.\mathsf{e_1} \longmapsto^* \mathsf{v_1}$ by operational semantics.
- Observe that: $\mathsf{v_1} = \mathsf{true}$ or $\mathsf{v_1} = \mathsf{false}$.
- Proceed by cases of $\mathsf{v_1}$:

  Case $\mathsf{v_1} = \mathsf{true}$:

  - Have: $\gamma_T(\mathbf{v_2})\ [\mathbf{bool}]\ \mathbf{id} \longmapsto^* \mathbf{true}$ by (1).
  - Have:
    $\gamma_T({}^{\sigma}\mathcal{ST}\ ((\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \mathbf{bool} \to \alpha).\ \mathbf{v_2}\ [\alpha]\ \boldsymbol{\lambda}(\mathbf{x} : \mathbf{bool}).$
    $\mathbf{if\ x\ then\ v_2'}\ [\alpha]\ \mathbf{k\ else\ v_2''}\ [\alpha]\ \mathbf{k})\ [\sigma^+]\ \mathbf{id}))$
    $\longmapsto {}^{\sigma}\mathcal{ST}\ (\boldsymbol{\lambda}(\mathbf{x} : \mathbf{bool}).\ \mathbf{if\ x\ then}\ \gamma_T(\mathbf{v_2}')\ [\sigma^+]\ \mathbf{id\ else}\ \gamma_T(\mathbf{v_2}'')\ [\sigma^+]\ \mathbf{id}\ )\mathbf{true}$
    $\longmapsto^* {}^{\sigma}\mathcal{ST}\ \gamma_T(\mathbf{v_2}')\ [\sigma^+]\ \mathbf{id}$
    by operational semantics.
  - Have: $\exists \mathsf{v}^f_2.{}^{\sigma}\mathcal{ST}\ \gamma_T(\mathbf{v_2}')\ [\sigma^+]\ \mathbf{id} \longmapsto^* \mathsf{v}^f_2$ and $(\mathsf{v}^f_1, \mathsf{v}^f_2) \in \mathcal{V}\,[\![\sigma]\!]\,\emptyset$ by (2).

  Case $\mathsf{v_1} = \mathsf{false}$:

  - (Similar to $\mathsf{v_1} = \mathsf{true}$ case.)

Case $\mathsf{e} = \mathsf{x}$:

- Have: $\Gamma \vdash \mathsf{x} : \sigma \rightsquigarrow \boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \sigma^+ \to \alpha).\ \mathbf{k\ x}$.
- Must show: $\vdash \gamma_S(\mathsf{x}) \precsim^{log}_{ST} {}^{\sigma}\mathcal{ST}\ ((\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \sigma^+ \to \alpha).\ \mathbf{k}\ (\gamma_T(\mathbf{x})))\ [\sigma^+]\ \mathbf{id}) : \sigma$:
- Observe that: $\gamma_S(\mathbf{x})$ is a value.
- Have: ${}^{\sigma}\mathcal{ST}\ ((\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \sigma^+ \to \alpha).\ \mathbf{k}\ (\gamma_T(\mathbf{x})))\ [\sigma^+]\ \mathbf{id}) \longmapsto^* {}^{\sigma}\mathcal{ST}\ \gamma_T(\mathbf{x})$ by operational semantics.
- Have: $\vdash \gamma_S(\mathsf{x}) \precsim^{log}_{ST} {}^{\sigma}\mathcal{ST}\ (\gamma_T(\mathbf{x})) : \sigma$ by construction of γ_S and γ_T.

Case $\mathsf{e} = \lambda \mathsf{x} : \sigma.\ \mathsf{e}$:

- Let $\Gamma' = \Gamma, \mathsf{x} : \sigma$.
- Have: $\Gamma' \vdash \mathsf{e_1} : \sigma' \rightsquigarrow \mathbf{v_2}$.

(3) Have: $\forall \gamma'_S, \gamma'_T.\Gamma' \vdash \gamma'_S \precsim \gamma'_T$ implies $\vdash \gamma'_S(\mathsf{e_1}) \precsim^{log}_{ST} {}^{\sigma'}\mathcal{ST}\ ((\gamma'_T(\mathbf{v_1}))\ [\sigma'^+]\ \mathbf{id}) : \sigma'$.

- Must show:
  $\vdash \lambda x:\sigma.\,\gamma_S(e_1)$
  $\precsim^{log}_{ST}\ {}^{\sigma\to\sigma'}\mathcal{ST}\,(\boldsymbol{\lambda\,[\alpha]\,(k:(\sigma\to\sigma')^+).}$
  $\mathbf{k}\ (\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).\ v_2\ [\beta]\ k'})\ [(\sigma\to\sigma')^+]\ \mathbf{id}) : \sigma\to\sigma'$:
- Observe that: $\lambda x:\sigma.\,\gamma_S(e_1) = v_1^f$ is a value.
- Must show:
  $\exists v_2^f.$
  ${}^{\sigma\to\sigma'}\mathcal{ST}\,(\boldsymbol{\lambda\,[\alpha]\,(k:(\sigma\to\sigma')^+).\ k\ (\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).\ v_2\ [\beta]\ k')}\ [(\sigma\to\sigma')^+]\ \mathbf{id})$
  $\longmapsto^* v_2^f$
  and $(v_1^f, v_2^f) \in \mathcal{V}\,[\![\sigma\to\sigma']\!]\,\emptyset$:
- Have:
  ${}^{\sigma\to\sigma'}\mathcal{ST}\,(\boldsymbol{\lambda\,[\alpha]\,(k:(\sigma\to\sigma')^+).\ k\ (\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).\ v_2\ [\beta]\ k')}\ [(\sigma\to\sigma')^+]\ \mathbf{id})$.
  $\longmapsto {}^{\sigma\to\sigma'}\mathcal{ST}\,\mathbf{id}\ (\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}\ (\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'})$
  $\longmapsto {}^{\sigma\to\sigma'}\mathcal{ST}\,(\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}\ (\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'})$
  $\longmapsto \lambda y:\sigma.\ {}^{\sigma'}\mathcal{ST}\ \mathbf{let}\ \mathbf{z} = (\mathcal{TS}^{\,\sigma}\ y)\ \mathbf{in}$
  $(\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}\ (\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'})\ [\sigma'^+]\ (\mathbf{z},\mathbf{id})$
  $= v_2^f$
- Must show: $(v_1^f, v_2^f) \in \mathcal{V}\,[\![\sigma\to\sigma']\!]\,\emptyset$:
- Consider arbitrary v_1' and v_2' such that $(v_1', v_2') \in \mathcal{V}\,[\![\sigma]\!]\,\emptyset$.
- Must show:
  $(\gamma_S(e)[v_1'/x],$
  $({}^{\sigma'}\mathcal{ST}\ \mathbf{let}\ \mathbf{z} = (\mathcal{TS}^{\,\sigma}\ y)\ \mathbf{in}\ (\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}$
  $(\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'}\ [\sigma'^+]\ (\mathbf{z},\mathbf{id}))[v_2'/y]) \in \mathcal{E}\,[\![\sigma']\!]\,\emptyset$:
- Have: $\gamma_S(e)[v_1'/x] = (\gamma_S[x\mapsto v_1'])(e) \longmapsto^* v_1^{f\prime}$.
- Must show:
  $\exists v_2^{f\prime}.({}^{\sigma'}\mathcal{ST}\ \mathbf{let}\ \mathbf{z} = (\mathcal{TS}^{\,\sigma}\ y)\ \mathbf{in}\ (\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}$
  $(\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'}\ [\sigma'^+]\ (\mathbf{z},\mathbf{id}))[v_2'/y])$
  $\longmapsto^* v_2^{f\prime}$
  and $(v_1^{f\prime}, v_2^{f\prime}) \in \mathcal{V}\,[\![\sigma']\!]\,\emptyset$:
- Have: $\exists \widehat{\mathbf{v_2}'}.\mathcal{TS}^{\,\sigma}\ v_2' \longmapsto \widehat{\mathbf{v_2}'}$ by operational semantics.
- Have:
  $({}^{\sigma'}\mathcal{ST}\ \mathbf{let}\ \mathbf{z} = (\mathcal{TS}^{\,\sigma}\ y)\ \mathbf{in}\ (\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}$.
  $(\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'}\ [\sigma'^+]\ (\mathbf{z},\mathbf{id}))[v_2'/y])$
  $\longmapsto^* v_2^{f\prime}$
- Have:
  $({}^{\sigma'}\mathcal{ST}\ \mathbf{let}\ \mathbf{z} = (\mathcal{TS}^{\,\sigma}\ y)\ \mathbf{in}\ (\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}$.
  $(\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'}\ [\sigma'^+]\ (\mathbf{z},\mathbf{id}))[v_2'/y])$
  $= {}^{\sigma'}\mathcal{ST}\ \mathbf{let}\ \mathbf{z} = (\mathcal{TS}^{\,\sigma}\ v_2')\ \mathbf{in}$
  $(\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}\ (\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'})\ [\sigma'^+]\ (\mathbf{z},\mathbf{id})$
  $\longmapsto {}^{\sigma'}\mathcal{ST}\ ((\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}\ (\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'})\ [\sigma'^+]\ (\mathbf{z},\mathbf{id}))[\widehat{\mathbf{v_2}'}/\mathbf{z}]$
  $= {}^{\sigma'}\mathcal{ST}\ ((\boldsymbol{\lambda\,[\beta]\,((x,k'):\sigma^+\times(\sigma'^+\to\beta)).}\ (\gamma_T(\mathbf{v_2}))\ [\beta]\ \mathbf{k'})\ [\sigma'^+]\ (\widehat{\mathbf{v_2}'},\mathbf{id}))$
  $\longmapsto (\gamma_T(\mathbf{v_2}[\widehat{\mathbf{v_2}'}/\mathbf{x}]))\ [\sigma'^+]\ \mathbf{id}$
  $= (\gamma_T[\mathbf{x}\mapsto\widehat{\mathbf{v_2}'}])(\mathbf{v_2})\ [\sigma'^+]\ \mathbf{id}$
- Let $\gamma_S' = \gamma_S[x\mapsto v_1']$.

- Let $\gamma'_{\mathrm{T}} = \gamma_{\mathrm{T}}[\mathbf{x} \mapsto \widehat{\mathbf{v_2}'}]$.
- Observe that: $\Gamma' \vdash \gamma'_{\mathrm{S}} \lesssim \gamma'_{\mathrm{T}}$.
- Instantiating (3) with γ'_{S} and γ'_{T}:
- Have: $\vdash \gamma'_{\mathrm{S}}(\mathsf{e}_1) \lesssim^{log}_{\mathrm{ST}} {}^{\sigma'}\mathcal{ST}\,((\gamma'_{\mathrm{T}}(\mathbf{v}_1))\ [\sigma'^{+}]\ \mathbf{id}) : \sigma'$.

Case $\mathsf{e} = \mathsf{e}_1\ \mathsf{e}_2$:

- Have: $\Gamma \vdash \mathsf{e}_1 : \sigma \rightarrow \sigma' \rightsquigarrow \mathbf{v}_2$.
- Have: $\Gamma \vdash \mathsf{e}'_1 : \sigma \rightsquigarrow \mathbf{v_2}'$.
- Have: $\vdash \gamma_{\mathrm{S}}(\mathsf{e}_1) \lesssim^{log}_{\mathrm{ST}} {}^{\sigma \rightarrow \sigma'}\mathcal{ST}\,(\gamma_{\mathrm{T}}(\mathbf{v}_2))\ [(\sigma \rightarrow \sigma')^{+}]\ \mathbf{id} : \sigma \rightarrow \sigma'$.
- Have: $\vdash \gamma_{\mathrm{S}}(\mathsf{e}'_1) \lesssim^{log}_{\mathrm{ST}} {}^{\sigma \rightarrow \sigma'}\mathcal{ST}\,(\gamma_{\mathrm{T}}(\mathbf{v_2}'))\ [\sigma^{+}]\ \mathbf{id} : \sigma$.
- Have:
  $\mathbf{v} = \boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \sigma'^{+} \rightarrow \alpha).\ \mathbf{v}_2\ [\alpha]\ (\boldsymbol{\lambda}(\mathbf{x}_1 : (\sigma \rightarrow \sigma')^{+}).\ \mathbf{v_2}'\ [\alpha]\ (\boldsymbol{\lambda}(\mathbf{x}_2 : \sigma^{+}).\ \mathbf{x}_1\ [\alpha]\ (\mathbf{x}_2, \mathbf{k})))$.
- Must show: $\vdash \gamma_{\mathrm{S}}(\mathsf{e}_1\ \mathsf{e}'_1) \lesssim^{log}_{\mathrm{ST}} {}^{\sigma'}\mathcal{ST}\,\gamma_{\mathrm{T}}(\mathbf{v})\ [\sigma'^{+}]\ \mathbf{id} : \sigma'$:
- Rewriting the right hand side:

$$\begin{aligned}
&{}^{\sigma'}\mathcal{ST}\,\gamma_{\mathrm{T}}(\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \sigma'^{+} \rightarrow \alpha).\ \mathbf{v}_2\ [\alpha]\ (\boldsymbol{\lambda}(\mathbf{x}_1 : (\sigma \rightarrow \sigma')^{+}).\\
&\quad \mathbf{v_2}'\ [\alpha]\ (\boldsymbol{\lambda}(\mathbf{x}_2 : \sigma^{+}).\ \mathbf{x}_1\ [\alpha]\ (\mathbf{x}_2, \mathbf{k}))))\ [\sigma'^{+}]\ \mathbf{id}\\
&\equiv {}^{\sigma'}\mathcal{ST}\,(\boldsymbol{\lambda}\,[\alpha]\,(\mathbf{k} : \sigma'^{+} \rightarrow \alpha).\ \gamma_{\mathrm{T}}(\mathbf{v}_2)\ [\alpha]\ (\boldsymbol{\lambda}(\mathbf{x}_1 : (\sigma \rightarrow \sigma')^{+}).\\
&\qquad \gamma_{\mathrm{T}}(\mathbf{v_2}')\ [\alpha]\ (\boldsymbol{\lambda}(\mathbf{x}_2 : \sigma^{+}).\ \mathbf{x}_1\ [\alpha]\ (\mathbf{x}_2, \mathbf{k}))))\ [\sigma'^{+}]\ \mathbf{id}\\
&\equiv {}^{\sigma'}\mathcal{ST}\,\gamma_{\mathrm{T}}(\mathbf{v}_2)\ [\sigma'^{+}]\ (\boldsymbol{\lambda}(\mathbf{x}_1 : (\sigma \rightarrow \sigma')^{+}). && \text{(beta)}\\
&\qquad \gamma_{\mathrm{T}}(\mathbf{v_2}')\ [\sigma'^{+}]\ (\boldsymbol{\lambda}(\mathbf{x}_2 : \sigma^{+}).\ \mathbf{x}_1\ [\sigma'^{+}]\ (\mathbf{x}_2, \mathbf{id})))\\
&\equiv {}^{\sigma'}\mathcal{ST}\,\gamma_{\mathrm{T}}(\mathbf{v}_2)\ [\sigma'^{+}]\ (\boldsymbol{\lambda}(\mathbf{x}_1 : (\sigma \rightarrow \sigma')^{+}). && \text{(free cont. theorem, beta)}\\
&\qquad \mathbf{x}_1\ [\sigma'^{+}]\ (\gamma_{\mathrm{T}}(\mathbf{v_2}')\ [\sigma^{+}]\ \mathbf{id}, \mathbf{id}))\\
&\equiv {}^{\sigma'}\mathcal{ST}\,(\gamma_{\mathrm{T}}(\mathbf{v}_2)\ [(\sigma \rightarrow \sigma')^{+}]\ \mathbf{id})\ [\sigma'^{+}]\ (\gamma_{\mathrm{T}}(\mathbf{v_2}')\ [\sigma^{+}]\ \mathbf{id}, \mathbf{id}) && \text{(free cont. theorem, beta)}\\
&\equiv {}^{\sigma'}\mathcal{ST}\,(\gamma_{\mathrm{T}}(\mathsf{v}_2)\ [(\sigma \rightarrow \sigma')^{+}]\ \mathbf{id})\ [\sigma'^{+}] && \text{(cancellation lemma)}\\
&\qquad \mathcal{TS}^{\,\sigma}\ ({}^{\sigma}\mathcal{ST}\,(\gamma_{\mathrm{T}}(\mathbf{v_2}')\ [\sigma^{+}]\ \mathbf{id}, \mathbf{id}))\\
&\equiv (\lambda \mathsf{x} : \sigma.\ {}^{\sigma'}\mathcal{ST}\,(\mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\ \mathsf{x})\ \mathbf{in} && \text{(beta)}\\
&\qquad (\gamma_{\mathrm{T}}(\mathbf{v}_2)\ [(\sigma \rightarrow \sigma')^{+}]\ \mathbf{id})\ [\sigma'^{+}]\ (\mathbf{z}, \mathbf{id}))\\
&\quad ({}^{\sigma}\mathcal{ST}\,\gamma_{\mathrm{T}}(\mathsf{v}'_2)\ [\sigma^{+}]\ \mathbf{id})\\
&\equiv {}^{\sigma \rightarrow \sigma'}\mathcal{ST}\,\gamma_{\mathrm{T}}(\mathbf{v}_2)\ [(\sigma \rightarrow \sigma')^{+}]\ \mathbf{id}\ {}^{\sigma}\mathcal{ST}\,\gamma_{\mathrm{T}}(\mathbf{v_2}')\ [\sigma^{+}]\ \mathbf{id} && \text{(operational semantics)}
\end{aligned}$$

- Have: $\vdash \gamma_{\mathrm{S}}(\mathsf{e}_1)\ \gamma_{\mathrm{S}}(\mathsf{e}'_1) \lesssim^{log}_{\mathrm{ST}} {}^{\sigma \rightarrow \sigma'}\mathcal{ST}\,\gamma_{\mathrm{T}}(\mathbf{v}_2)\ [(\sigma \rightarrow \sigma')^{+}]\ \mathbf{id}\ {}^{\sigma}\mathcal{ST}\,\gamma_{\mathrm{T}}(\mathsf{v}'_2)\ [\sigma^{+}]\ \mathbf{id} : \sigma'$
  $= \vdash \gamma_{\mathrm{S}}(\mathsf{e}_1\ \mathsf{e}'_1) \lesssim^{log}_{\mathrm{ST}} {}^{\sigma'}\mathcal{ST}\,\gamma_{\mathrm{T}}(\mathbf{v})\ [\sigma'^{+}]\ \mathbf{id} : \sigma'$ by rewriting.

$\square$

Lemma 9.5 (CPS is semantics reflecting)
If $\Gamma \vdash \mathsf{e} : \sigma \rightsquigarrow \mathbf{v}$ *and* $\Gamma \vdash \gamma_S \gtrsim \gamma_T$
then $\vdash \gamma_T(\mathbf{v}) \precsim^{log}_{ST} \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{k} : \sigma^+ \!\to \boldsymbol{\alpha}).\, \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\ \gamma_S(\mathsf{e}))\ \mathbf{in\ k\ z} : \sigma^{\div}$.

Proof

- Consider arbitrary $\boldsymbol{\tau}_1$, $\boldsymbol{\tau}_2$, R, $\mathbf{k_1}$, and $\mathbf{k_2}$ such that $R \in \mathrm{Rel}[\boldsymbol{\tau}_1, \boldsymbol{\tau}_2]$ and $(\mathbf{k_1}, \mathbf{k_2}) \in \mathcal{V}[\![\sigma^+ \!\to \boldsymbol{\alpha}]\!]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$.
- Must show: $(\gamma_T(\mathbf{v})\ [\boldsymbol{\tau}_1]\ \mathbf{k_1}, \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\ \gamma_S(\mathsf{e}))\ \mathbf{in\ k_2\ z}) \in \mathcal{E}[\![\boldsymbol{\alpha}]\!]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$:
- Proceed by induction on the structure of e:

  Case $\mathsf{e} = \mathsf{true}$:

  - Observe that: $\mathbf{v} = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{k} : \mathbf{bool} \to \boldsymbol{\alpha}).\, \mathbf{k\ true}$.
  - Must show: $(\mathbf{k_1}\ \gamma_T(\mathbf{true}), \mathbf{let\ z} = (\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{true})\ \mathbf{in\ k_2\ z}) \in \mathcal{E}[\![\boldsymbol{\alpha}]\!]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$:
  - Rewriting:
    $\mathbf{k_1}\ \gamma_T(\mathbf{true})$
    $\equiv \mathbf{k_1\ true}$ (subst. on closed term)
  - Rewriting:
    $\mathbf{let\ z} = (\mathcal{TS}^{\,\mathsf{bool}}\ \mathsf{true})\ \mathbf{in\ k_2\ z}$
    $\equiv \mathbf{k_2\ true}$ (subst. on closed term; beta)
  - Observe that: Related continuations applied to related arguments are related.

  Case $\mathsf{e} = \mathsf{false}$:

  - (Similar to true case)

  Case $\mathsf{e} = \mathsf{if}\ \mathsf{e}_2\ \mathsf{then}\ \mathsf{e}_2'\ \mathsf{else}\ \mathsf{e}_2''$:

  - Observe that:
    $\mathbf{v} = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{k} : \sigma^+ \!\to \boldsymbol{\alpha}).\, \mathbf{v}_1\ [\boldsymbol{\alpha}]\ \boldsymbol{\lambda}(\mathbf{x} : \mathbf{bool}).\, \mathbf{if\ x\ then\ v_1'}\ [\boldsymbol{\alpha}]\ \mathbf{k\ else\ v_1''}\ [\boldsymbol{\alpha}]\ \mathbf{k}$.
  - Have: $\Gamma \vdash \mathsf{e}_2 : \mathsf{bool} \rightsquigarrow \mathbf{v}_1$.
  - Have: $\Gamma \vdash \mathsf{e}_2' : \sigma \rightsquigarrow \mathbf{v_1'}$.
  - Have: $\Gamma \vdash \mathsf{e}_2'' : \sigma \rightsquigarrow \mathbf{v_1''}$.

  (1) Have: $\vdash \gamma_T(\mathbf{v}_1) \precsim^{log}_{ST} \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{k} : \mathbf{bool} \to \boldsymbol{\alpha}).\, \mathbf{let\ z} = (\mathcal{TS}^{\,\mathsf{bool}}\ \gamma_S(\mathsf{e}_2))\ \mathbf{in\ k\ z} : \mathsf{bool}^{\div}$.

  (2) Have: $\vdash \gamma_T(\mathbf{v_1'}) \precsim^{log}_{ST} \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{k} : \sigma^+ \!\to \boldsymbol{\alpha}).\, \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\ \gamma_S(\mathsf{e}_2'))\ \mathbf{in\ k\ z} : \sigma^{\div}$.

  - Have: $\vdash \gamma_T \mathbf{v_1''} \precsim^{log}_{ST} \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\,(\mathbf{k} : \sigma^+ \!\to \boldsymbol{\alpha}).\, \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\ \gamma_S(\mathsf{e}_2''))\ \mathbf{in\ k\ z} : \sigma^{\div}$.
  - Must show:
    $(\gamma_T(\mathbf{v}_1)\ [\boldsymbol{\tau}_1]\ \boldsymbol{\lambda}\,[\mathbf{x}]\,(\mathbf{bool} : \mathbf{if\ x\ then}\ \gamma_T(\mathbf{v_1'})\ [\boldsymbol{\tau}_1]\ \mathbf{k_1\ else}\ \gamma_T(\mathbf{v_1''})\ [\boldsymbol{\tau}_1]\ \mathbf{k_1}).,$
    $\mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\ \mathsf{if}\ \gamma_S(\mathsf{e}_2)\ \mathsf{then}\ \gamma_S(\mathsf{e}_2')\ \mathsf{else}\ \gamma_S(\mathsf{e}_2''))\ \mathbf{in}\ ) \in \mathcal{E}[\![\boldsymbol{\alpha}]\!]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$:
  - Rewriting:
    $\gamma_T(\mathbf{v}_1)\ [\boldsymbol{\tau}_1]\ (\boldsymbol{\lambda}(\mathbf{x} : \mathbf{bool}).\ )$
    $\mathbf{if\ x\ then}\ \gamma_T(\mathbf{v_1'})\ [\boldsymbol{\tau}_1]\ \mathbf{k_1\ else}\ \gamma_T(\mathbf{v_1''})\ [\boldsymbol{\tau}_1]\ \mathbf{k_1}$
    $\equiv (\boldsymbol{\lambda}(\mathbf{x} : \mathbf{bool}).$ (free cont. theorem)
    $\quad \mathbf{if\ x\ then}\ \gamma_T(\mathbf{v_1'})\ [\boldsymbol{\tau}_1]\ \mathbf{k_1\ else}\ \gamma_T(\mathbf{v_1''})\ [\boldsymbol{\tau}_1]\ \mathbf{k_1})$
    $\quad (\gamma_T(\mathbf{v}_1)\ [\mathbf{bool}]\ \mathbf{id})$
  - Have: $(\gamma_T(\mathbf{v}_1)\ [\mathbf{bool}]\ \mathbf{id}, \mathbf{let\ z} = (\mathcal{TS}^{\,\mathsf{bool}}\ \gamma_S(\mathsf{e}_2))\ \mathbf{in\ id\ z}) \in \mathcal{E}[\![\mathbf{bool}]\!]\,\emptyset$ by (1).
  - Consider arbitrary $\mathbf{v}_1^f$ such that $\gamma_T(\mathbf{v}_1)\ [\mathbf{bool}]\ \mathbf{id} \longmapsto^* \mathbf{v}_1^f$.
    Proceed by cases of $\mathbf{v}_1^f$:

Case $\mathbf{v}_1^f = \mathbf{true}$:

(3) Have: $\gamma_S(\mathsf{e}_2) \longmapsto^* \mathsf{true}$ by operational semantics.

- Rewriting:
  $\gamma_T(\mathbf{v}_1)\ [\boldsymbol{\tau}_1]\ (\boldsymbol{\lambda}(\mathbf{x} : \mathbf{bool}).$
  $\quad \mathbf{if\ x\ then}\ \gamma_T(\mathbf{v}_1')\ [\boldsymbol{\tau}_1]\ \mathbf{k_1}\ \mathbf{else}\ \gamma_T(\mathbf{v}_1'')\ [\boldsymbol{\tau}_1]\ \mathbf{k_1})$
  $\equiv \gamma_T(\mathbf{v}_2')\ [\boldsymbol{\tau}_1]\ \mathbf{k_1}$ (operational semantics)

- Rewriting:
  $\mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\ \mathsf{if}\ \gamma_S(\mathsf{e}_2)\ \mathsf{then}\ \gamma_S(\mathsf{e}_2')\ \mathsf{else}\ \gamma_S(\mathsf{e}_2''))\ \mathbf{in\ k_2\ z}$
  $\equiv \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\ \gamma_S(\mathsf{e}_2'))\ \mathbf{in\ k_2\ z}$ (operational semantics; (3))
  $\equiv (\boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\ (\mathbf{k} : \sigma^+ \!\to \boldsymbol{\alpha}).\ \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma}\ \gamma_S(\mathsf{e}_2'))\ \mathbf{in\ k\ z})\ [\boldsymbol{\tau}_2]\ \mathbf{k_2}$ (beta)

- (By (2), these are related as required.)

Case $\mathbf{v}_1^f = \mathbf{false}$:

- (Similar to **true** case)

Case $\mathsf{e} = \mathsf{x}$:

- Observe that: $\mathbf{v} = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\ (\mathbf{k} : \sigma^+ \!\to \boldsymbol{\alpha}).\ \mathbf{k\ x}$.
- Must show: $(\mathbf{k_1}\ (\gamma_T(\mathbf{x})), \mathcal{TS}^{\,\sigma}\ \gamma_S(\mathsf{x})) \in \mathcal{E}\,[\![\boldsymbol{\alpha}]\!]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$:
- Observe that: $\gamma_S(\mathsf{x})$ is a value of type σ.

(4) Have: $\exists \mathbf{v}_1 . \mathcal{TS}^{\,\sigma}\ \gamma_S(\mathsf{x}) \longmapsto \mathbf{v}_1$ by operational semantics.

(5) Have: $({}^{\sigma}\mathcal{ST}\ \gamma_T(\mathbf{x}), \gamma_S(\mathsf{x})) \in \mathcal{E}\,[\![\sigma]\!]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$ from $\Gamma \vdash \gamma_S \gtrsim \gamma_T$.

- Have: $(\gamma_T(\mathbf{x}), \mathbf{v}_1) \in \mathcal{V}\,[\![\sigma^+]\!]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$ from (5), (4) and cancellation lemma.
- Must show: $(\mathbf{k_1}\ (\gamma_T(\mathbf{x})), \mathbf{k_2\ v_2'}) \in \mathcal{E}\,[\![\boldsymbol{\alpha}]\!]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$:
- Observe that: Related continuations applied to related arguments are related.

Case $\mathsf{e} = \lambda \mathsf{x} : \sigma.\ \mathsf{e}_2$:

- Observe that:
  $\mathbf{v} = \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\ (\mathbf{k} : (\sigma \to \sigma')^+ \!\to \boldsymbol{\alpha}).\ \mathbf{k}\ (\boldsymbol{\lambda}\,[\boldsymbol{\beta}]\ ((\mathbf{x}, \mathbf{k}') : \sigma^+ \times (\sigma'^+ \!\to \boldsymbol{\beta})).\ \mathbf{v}_1\ [\boldsymbol{\beta}]\ \mathbf{k}')$.
- Let $\Gamma' = \Gamma, \mathsf{x} : \sigma$.
- Have: $\Gamma' \vdash \mathsf{e}_2 : \sigma' \rightsquigarrow \mathbf{v}_1$.

(6) Have:
$\forall \gamma_S', \gamma_T' . \Gamma' \vdash \gamma_S' \gtrsim \gamma_T'$
implies $\vdash \gamma_T'(\mathbf{v}_1) \precsim^{log}_{ST} \boldsymbol{\lambda}\,[\boldsymbol{\alpha}]\ (\mathbf{k} : \sigma'^+ \!\to \boldsymbol{\alpha}).\ \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma'}\ \gamma_S'(\mathsf{e}_2))\ \mathbf{in\ k\ z} : \sigma'^{\div}$.

- Must show:
  $(\mathbf{k_1}\ (\boldsymbol{\lambda}\,[\boldsymbol{\beta}]\ ((\mathbf{x}, \mathbf{k}') : \sigma^+ \times (\sigma'^+ \!\to \boldsymbol{\beta})).\ \gamma_T(\mathbf{v}_1)\ [\boldsymbol{\beta}]\ \mathbf{k}'),$
  $\mathbf{let\ z} = (\mathcal{TS}^{\,\sigma \to \sigma'}\ \lambda \mathsf{x} : \sigma.\ \gamma_S(\mathsf{e}_2))\ \mathbf{in\ k_2}\ tz) \in \mathcal{E}\,[\![\boldsymbol{\alpha}]\!]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R)]$
- Rewriting:
  $\mathbf{let\ z} = (\mathcal{TS}^{\,\sigma \to \sigma'}\ \lambda \mathsf{x} : \sigma.\ \gamma_S(\mathsf{e}_2))\ \mathbf{in\ k_2}\ tz$
  $\equiv \mathbf{k_2}\ (\boldsymbol{\lambda}\,[\boldsymbol{\beta}]\ ((\mathbf{x}, \mathbf{k}) : \sigma^+ \times (\sigma'^+ \!\to \boldsymbol{\beta})).$ (beta)
  $\quad \mathbf{let\ z} = (\mathcal{TS}^{\,\sigma'}\ (\lambda \mathsf{x} : \sigma.\ \gamma_S(\mathsf{e}))\ ({}^{\sigma}\mathcal{ST}\ \mathbf{x}))\ \mathbf{in\ k\ z})$
- Consider arbitrary $\boldsymbol{\tau}_1'$, $\boldsymbol{\tau}_2'$, R' such that $R' \in \mathrm{Rel}[\boldsymbol{\tau}_1', \boldsymbol{\tau}_2']$.
- Consider arbitrary $\widehat{\mathbf{v}_1}$, $\widehat{\mathbf{v}_1'}$, $\widehat{\mathbf{v}_2}$, and $\widehat{\mathbf{v}_2'}$ such that
  $((\widehat{\mathbf{v}_1}, \widehat{\mathbf{v}_1'}), (\widehat{\mathbf{v}_2}, \widehat{\mathbf{v}_2'})) \in \mathcal{V}\left[\!\!\left[\sigma^+ \times (\sigma'^+ \!\to \boldsymbol{\beta})\right]\!\!\right]\,[\boldsymbol{\alpha} \mapsto (\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, R), \boldsymbol{\beta} \mapsto (\boldsymbol{\tau}_1', \boldsymbol{\tau}_2', R')]$.
- Have: $\exists \mathsf{v}_2^f . {}^{\sigma}\mathcal{ST}\ \widehat{\mathbf{v}_2} \longmapsto \mathsf{v}_2^f$ by operational semantics.

- Have: $(\widehat{\mathbf{v}_1}, \mathcal{TS}^{\sigma}\, \mathsf{v}_2^f) \in \mathcal{V}[\![\sigma^+]\!]\,[\alpha \mapsto (\tau_1, \tau_2, R)]$ by cancellation lemma.
- Must show:
  $(\gamma_T[\mathbf{x} \mapsto \widehat{\mathbf{v}_1}](\mathbf{v}_1)\ [\tau_1']\ \widehat{\mathbf{v}_1{}'},$
  $\mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ (\lambda \mathsf{x} : \sigma.\ \gamma_S(\mathsf{e}))\ ({}^{\sigma}\mathcal{ST}\ \widehat{\mathbf{v}_2}))\ \mathbf{in}\ \widehat{\mathbf{v}_2{}'}\ \mathbf{z})$
  $\quad \in \mathcal{V}[\![\beta]\!]\,[\alpha \mapsto (\tau_1, \tau_2, R), \beta \mapsto (\tau_1', \tau_2', R')]$:
- Rewriting:
  $$\begin{aligned}
  &\mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ (\lambda \mathsf{x} : \sigma.\ \gamma_S(\mathsf{e}))\ ({}^{\sigma}\mathcal{ST}\ \widehat{\mathbf{v}_2}))\ \mathbf{in}\ \widehat{\mathbf{v}_2{}'}\ \mathbf{z} \\
  &\equiv \mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \gamma_S[\mathsf{x} \mapsto \widehat{\mathsf{v}_2}](\mathsf{e}))\ \mathbf{in}\ \widehat{\mathbf{v}_2{}'}\ \mathbf{z} && \text{(operational semantics)} \\
  &\equiv \lambda\,[\alpha]\ (\mathbf{k} : \sigma'^{+} \to \alpha). && \text{(beta)} \\
  &\qquad \mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \gamma_S[\mathsf{x} \mapsto \widehat{\mathsf{v}_2}](\mathsf{e}))\ \mathbf{in\ k\ z}\ [\tau_2']\ \widehat{\mathbf{v}_2{}'}
  \end{aligned}$$
- Must show:
  $(\gamma_T[\mathbf{x} \mapsto \widehat{\mathbf{v}_1}](\mathbf{v}_1)\ [\tau_1']\ \widehat{\mathbf{v}_1{}'},$
  $(\lambda\,[\alpha]\ (\mathbf{k} : \sigma'^{+} \alpha).\ \mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \gamma_S[\mathsf{x} \mapsto \widehat{\mathsf{v}_2}](\mathsf{e}))\ \mathbf{in\ k\ z})\ [\tau_2']\ \widehat{\mathbf{v}_2{}'})$
  $\quad \in \mathcal{V}[\![\beta]\!]\,[\alpha \mapsto (\tau_1, \tau_2, R), \beta \mapsto (\tau_1', \tau_2', R')]$:
- Observe that: $\Gamma' \vdash \gamma_S[\mathsf{x} \mapsto \widehat{\mathsf{v}_2}] \gtrsim \gamma_T[\mathbf{x} \mapsto \widehat{\mathbf{v}_1}]$.
- (This follows from (6).)

Case $\mathsf{e} = \mathsf{e}_2\ \mathsf{e}_2'$:

- Observe that:
  $\mathbf{v} = \lambda\,[\alpha]\ (\mathbf{k} : \sigma'^{+} \to \alpha).\ \mathbf{v}_1\ [\alpha]\ (\lambda(\mathbf{x}_1 : (\sigma \to \sigma')^{+}).\ \mathbf{v}_1{}'\ [\alpha]\ (\lambda(\mathbf{x}_2 : \sigma^{+}).\ \mathbf{x}_1\ [\alpha]\ (\mathbf{x}_2, \mathbf{k}))).$
- Have: $\Gamma \vdash \mathsf{e}_2 : \sigma \to \sigma' \rightsquigarrow \mathbf{v}_1$.
- Have: $\Gamma \vdash \mathsf{e}_2' : \sigma \rightsquigarrow \mathbf{v}_1{}'$.

(7) Have:
$(\gamma_T(\mathbf{v}_1), \lambda\,[\alpha]\ (\mathbf{k} : (\sigma \to \sigma')^{+} \to \alpha).\ \mathbf{let\ z} = (\mathcal{TS}^{\sigma}\ \gamma_S(\mathsf{e}_2))\ \mathbf{in\ k\ z}) \in \mathcal{V}\left[\!\!\left[(\sigma \to \sigma')^{\div}\right]\!\!\right] \emptyset.$

(8) Have: $(\gamma_T(\mathbf{v}_1{}'), \lambda\,[\alpha]\ (\mathbf{k} : \sigma^{+} \to \alpha).\ \mathbf{let\ z} = (\mathcal{TS}^{\sigma}\ \gamma_S(\mathsf{e}_2'))\ \mathbf{in\ k\ z}) \in \mathcal{V}[\![\sigma^{\div}]\!]\,\emptyset$.

- Must show:
  $(\gamma_T(\mathbf{v}_1)\ [\tau_1]\ (\lambda(\mathbf{x}_1 : (\sigma \to \sigma')^{+}).\ \gamma_T(\mathbf{v}_1{}')\ [\tau_1]\ (\lambda(\mathbf{x}_2 : \sigma^{+}).\ \mathbf{x}_1\ [\tau_1]\ (\mathbf{x}_2, \mathbf{k}_1))),$
  $\mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \gamma_S(\mathsf{e}_2)\ \gamma_S(\mathsf{e}_2'))\ \mathbf{in\ k_2\ z}) \in \mathcal{E}[\![\alpha]\!]\,[\alpha \mapsto (\tau_1, \tau_2, R)]$:
- Rewriting the left-hand side:
  $$\begin{aligned}
  &\gamma_T(\mathbf{v}_1)\ [\tau_1]\ (\lambda(\mathbf{x}_1 : (\sigma \to \sigma')^{+}). \\
  &\quad \gamma_T(\mathbf{v}_1{}')\ [\tau_1]\ (\lambda(\mathbf{x}_2 : \sigma^{+}).\ \mathbf{x}_1\ [\tau_1]\ (\mathbf{x}_2, \mathbf{k}_1))) \\
  &\equiv (\lambda(\mathbf{x}_1 : (\sigma \to \sigma')^{+}).\ \gamma_T(\mathbf{v}_1{}')\ [\tau_1] && \text{(free cont. theorem)} \\
  &\qquad (\lambda(\mathbf{x}_2 : \sigma^{+}).\ \mathbf{x}_1\ [\tau_1]\ (\mathbf{x}_2, \mathbf{k}_1)))\ (\gamma_T(\mathbf{v}_1)\ [(\sigma \to \sigma')^{+}]\ \mathbf{id}) \\
  &\equiv \gamma_T(\mathbf{v}_1{}')\ [\tau_1]\ (\lambda(\mathbf{x}_2 : \sigma^{+}).\ ) && \text{(beta)} \\
  &\qquad (\gamma_T(\mathbf{v}_1)\ [(\sigma \to \sigma')^{+}]\ \mathbf{id})\ [\tau_1]\ (\mathbf{x}_2, \mathbf{k}_1) \\
  &\equiv (\lambda(\mathbf{x}_2 : \sigma^{+}).\ (\gamma_T(\mathbf{v}_1)\ [(\sigma \to \sigma')^{+}]\ \mathbf{id})\ [\tau_1]\ \mathbf{k}_1) && \text{(free cont. theorem)} \\
  &\qquad (\gamma_T(\mathbf{v}_1{}')\ [\sigma^{+}]\ \mathbf{id}) \\
  &\equiv (\gamma_T(\mathbf{v}_1)\ [(\sigma \to \sigma')^{+}]\ \mathbf{id})\ [\tau_1]\ (\gamma_T(\mathbf{e}_1)\ [(\sigma \to \sigma')^{+}]\ \mathbf{id}, \mathbf{k}_1) && \text{(beta)}
  \end{aligned}$$
- Rewriting the right-hand side:
  $$\begin{aligned}
  &\mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \gamma_S(\mathsf{e}_2)\ \gamma_S(\mathsf{e}_2'))\ \mathbf{in\ k_2\ z} \\
  &\equiv \mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \gamma_S(\mathsf{e}_2)\ ({}^{\sigma}\mathcal{ST}\ (\mathcal{TS}^{\sigma}\ \gamma_S(\mathsf{e}_2'))))\ \mathbf{in\ k_2\ z} && \text{(cancellation lemma)} \\
  &\equiv (\lambda\,[\alpha]\ ((\mathbf{x}, \mathbf{k}) : \sigma^{+} \times (\sigma'^{+} \to \alpha)). && \text{(beta)} \\
  &\qquad \mathbf{let\ z} = (\mathcal{TS}^{\sigma'}\ \gamma_S(\mathsf{e}_2)\ ({}^{\sigma}\mathcal{ST}\ \mathsf{x}))\ \mathbf{in\ k\ z}) \\
  &\quad (\mathcal{TS}^{\sigma}\ \gamma_S(\mathsf{e}_2'), \mathbf{k}_2) \\
  &\equiv (\mathcal{TS}^{\sigma \to \sigma'}\ \gamma_S(\mathsf{e}_2))\ [\tau_2]\ (\mathcal{TS}^{\sigma}\ \gamma_S(\mathsf{e}_2{}'), \mathbf{k}_2) && \text{(operational semantics)}
  \end{aligned}$$

- Now we will show:
  $(\gamma_T(\mathbf{v}_1)\ [(\sigma \to \sigma')^+]\ \mathbf{id}, \mathcal{TS}^{\sigma \to \sigma'}\ \gamma_S(e_2)) \in \mathcal{E}\left[\!\left[(\sigma \to \sigma')^+\right]\!\right] [\alpha \mapsto (\tau_1, \tau_2, R)]$:

  - Rewriting:
    $\mathcal{TS}^{\sigma \to \sigma'}\ \gamma_S(e_2)$
    $\equiv (\lambda\,[\alpha]\ (\mathbf{k} : (\sigma \to \sigma')^+ \to \alpha).$ (beta)
    $\quad \mathbf{let\ z} = (\mathcal{TS}^{\sigma \to \sigma'}\ \gamma_S(e_2))\ \mathbf{in\ k\ z})\ [(\sigma \to \sigma')^+]\ \mathbf{id}$
  - (This follows from (7).)

- Now we will show: $(\gamma_T(\mathbf{v}_1')\ [\sigma^+]\ \mathbf{id}, \mathcal{TS}^{\sigma}\ \gamma_S(\mathbf{e}_2')) \in \mathcal{E}[\![\sigma^+]\!]\,[\alpha \mapsto (\tau_1, \tau_2, R)]$:

  - Rewriting:
    $\mathcal{TS}^{\sigma}\ \gamma_S(\mathbf{e}_2')$
    $\equiv (\lambda\,[\alpha]\ (\mathbf{k} : \sigma^+ \to \alpha).\ \mathbf{let\ z} = (\mathcal{TS}^{\sigma}\ \gamma_S(e_2'))\ \mathbf{in\ k\ z})\ [\sigma^+]\ \mathbf{id}$ (beta)
  - (This follows from (8).)

- (This follows from the definition of $\mathcal{V}[\![\forall\,[\cdot].\cdot \to \cdot]\!]\,\cdot$)

$\square$

The following is a corollary of semantics preservation and reflection. Notice that by boundary cancellation, the conclusion is equivalent to $\vdash \mathcal{TS}^{\sigma}(\gamma_S(e)) \approx^{log}_{ST} (\gamma_T\,(\mathbf{v})\ [\sigma^+]\ \mathbf{id}) : \sigma^+$. (This explains why we call it "translation is equivalent to embedding.")

Corollary 9.6 (Translation is Equivalent to Embedding)
Let $\Gamma \vdash e : \sigma \rightsquigarrow \mathbf{v}$ *and* $\Gamma \vdash \gamma_S \simeq \gamma_T$.
Then $\vdash \gamma_S(e) \approx^{log}_{ST} {}^{\sigma}\mathcal{ST}\,(\gamma_T\,(\mathbf{v})\ [\sigma^+]\ \mathbf{id}) : \sigma$.

Proof

Immediate from Lemmas 9.4 (CPS is semantics preserving), 9.5 (CPS is semantics reflecting), and 9.1 (boundary cancellation). $\square$

Theorem 9.7 (Translation Preserves Equivalence in λ^{ST})
Let e_1 *and* e_2 *be* λ^S *terms.*
If $\Gamma \vdash e_1 \precsim^{log}_{ST} e_2 : \sigma$, $\Gamma \vdash e_1 : \sigma \rightsquigarrow \mathbf{v}_1$, *and* $\Gamma \vdash e_2 : \sigma \rightsquigarrow \mathbf{v}_2$, *then* $\Gamma \vdash \mathbf{v}_1 \precsim^{log}_{ST} \mathbf{v}_2 : \sigma^{\div}$.

Proof

- (Follows from Lemma 9.4, Lemma 9.5, and transitivity of $\cdot \precsim^{log}_{ST} \cdot$).

$\square$

10 Part (1): Equivalence in λ^{S} Implies Equivalence in λ^{ST}

Lemma 10.1 **(λ^{ST} term of type σ/σ^{+} to equivalent λ^{S} term)**
Let $\Gamma ::= \cdot \mid \Gamma, x : \sigma \mid y : \sigma^{+}$

1. *If* $\cdot;\Gamma \vdash e : \sigma$
then $\exists e \in \lambda^{\mathrm{S}}.\ \cdot;\Gamma \vdash e : \sigma \twoheadrightarrow e\ \wedge\ \cdot;\Gamma^{\twoheadrightarrow} \vdash e[\overline{(\mathcal{TS}^{\Gamma^{\twoheadrightarrow}(y)}\, y)/y}] \approx^{log}_{\mathrm{ST}} e : \sigma.$

2. *If* $\cdot;\Gamma \vdash e : \sigma^{+}$
then $\exists e \in \lambda^{\mathrm{S}}.\ \cdot;\Gamma \vdash^{+} e : \sigma^{+} \twoheadrightarrow e\ \wedge\ \cdot;\Gamma^{\twoheadrightarrow} \vdash {}^{\sigma}\mathcal{ST}\,(e[\overline{(\mathcal{TS}^{\Gamma^{\twoheadrightarrow}(y)}\, y)/y}]) \approx^{log}_{\mathrm{ST}} e : \sigma.$

Proof

The proof is relatively straightforward. (1) and (2) are proved by simultaneous induction since the σ and σ^{+} translation rules are mutually dependent. We then proceed by induction on the length of the reduction sequence for e, nested induction on the type σ, and innermost induction on the structure of the term e.
Proving all the cases for (1) is especially straightforward: the results follow trivially or from the induction hypotheses.
Some of the cases for (2) are a bit more involved.
When $e = \mathbf{true}$ and when $e = \mathbf{false}$, the proof is immediate from the reduction rules for ${}^{\mathsf{bool}}\mathcal{ST}\cdot$ and $\mathcal{TS}^{\mathsf{bool}}\cdot$.
When $e = \mathbf{if\ v\ then\ e_1\ else\ e_2}$, the proof simply follows from the induction hypotheses.
When $e = \mathbf{y}$, we need to make use of the boundary cancellation lemma.
When e is a lambda term, the proof follows easily from the reduction rule for ${}^{\sigma_1 \to \sigma_2}\mathcal{ST}\cdot$ together with the induction hypothesis.
When projecting from a pair, the proof follows easily from the induction hypothesis after applying a single reduction step to e.
The case for application, where the function being applied *is not* of translation type, also follows easily from the induction hypothesis after applying a single reduction step to e.
The case for application where the function being applied *is* of translation type requires use of the continuation shuffling free theorem together with the induction hypotheses. $\square$

Theorem 10.2 **(Ciu-equiv in λ^{S} implies ciu-equiv in λ^{ST})**
If $\Gamma \vdash e_1 \precsim^{ciu}_{\mathrm{S}} e_2 : \sigma$ *then* $\cdot;\Gamma \vdash e_1 \precsim^{ciu}_{\mathrm{ST}} e_2 : \sigma$.

Proof

Suppose $E : (\cdot;\cdot \vdash \sigma \Rightarrow (\cdot;\cdot \vdash \mathsf{bool})$, and $\gamma_{\mathrm{st}} : \Gamma$ and $E[\gamma_{\mathrm{st}}(e_1)] \Downarrow v$ where $v : \mathsf{bool}$. Show: $E[\gamma_{\mathrm{st}}(e_2)] \Downarrow v$.
We back-translate E (or, to be precise, $\lambda x : \sigma.\, E[x]$) and γ_{st} to e_E and γ_{s}. By Lemma 10.1 these are equivalent to the original E and γ_{st}. Hence, $E[\gamma_{\mathrm{st}}(e_1)] \approx^{ctx}_{\mathrm{ST}} e_E(\gamma_{\mathrm{s}}(e_1)) : \mathsf{bool}$. Hence, the latter evaluates to v. Now, we instantate the premise with e_E (after morphing it into a valid evaluation context), and γ_{s}. Hence, $e_E(\gamma_{\mathrm{s}}(e_2))$ evaluates to v. Since $e_E(\gamma_{\mathrm{s}}(e_2)) \approx^{ctx}_{\mathrm{ST}} E[\gamma_{\mathrm{st}}(e_2)] : \mathsf{bool}$, the latter evaluates to v. $\square$

Note that contextual approximation implies ciu approximation in λ^{S}:

Lemma 10.3
If $\Gamma \vdash \mathtt{e}_1 \precsim^{ctx}_{\mathrm{S}} \mathtt{e}_2 : \sigma$ *then* $\Gamma \vdash \mathtt{e}_1 \precsim^{ciu}_{\mathrm{S}} \mathtt{e}_2 : \sigma$.

Proof

Straightforward. Follows from the fact that $\precsim^{ctx}_{\mathrm{S}}$ is a congruence, and the fact that every evaluation context E is a program context.
See [3] Section E.3 (page 161) for details. □

Our desired lemma now easily follows:

Theorem 10.4 (Equivalence in λ^{S} implies equivalence in λ^{ST})
If $\Gamma \vdash \mathtt{e}_1 \precsim^{ctx}_{\mathrm{S}} \mathtt{e}_2 : \sigma$ *then* $\cdot; \Gamma \vdash \mathtt{e}_1 \precsim^{ctx}_{\mathrm{ST}} \mathtt{e}_2 : \sigma$.

Proof

Immediate from Lemmas 10.2, 10.3, and the fact that $\precsim^{ciu}_{\mathrm{ST}}$ implies $\precsim^{ctx}_{\mathrm{ST}}$ (which in turn follows from Lemmas 8.26 and 8.22). □

11 Part (3): Equivalence in λ^{ST} Implies Equivalence in λ^{T}

For the final (bottom) layer of our proof of equivalence preservation, we must show that if $\cdot;\Gamma^{+} \vdash \mathbf{v}_1 \approx^{ctx}_{\mathrm{ST}} \mathbf{v}_2 : \sigma^{\div}$ then $\cdot;\Gamma^{+} \vdash \mathbf{v}_1 \approx^{ctx}_{\mathrm{T}} \mathbf{v}_2 : \sigma^{\div}$. The latter is immediate from the following more general lemma.

Lemma 11.1 (Equivalence in λ^{ST} implies equivalence in λ^{T})
Let $\mathbf{e}_1$ and $\mathbf{e}_2$ be λ^{T} terms.
If $\Delta;\Gamma \vdash \mathbf{e}_1 \precsim^{ctx}_{\mathrm{ST}} \mathbf{e}_2 : \boldsymbol{\tau}$ then $\Delta;\Gamma \vdash \mathbf{e}_1 \precsim^{ctx}_{\mathrm{T}} \mathbf{e}_2 : \boldsymbol{\tau}$.

Proof

The proof is straightforward, intuitively, because λ^{T} contexts are a subset of λ^{ST} contexts.
Given an arbitrary λ^{T} context $\mathbf{C}$ of the appropriate type, we must show that if $\mathbf{C}[\mathbf{e}_1]$ evaluates to $\mathbf{v}$ (which will be of type **bool**) then so does $\mathbf{C}[\mathbf{e}_1]$.
We can instantiate the premise with the context ${}^{\mathsf{bool}}\mathcal{ST}\,[\mathbf{C}[\cdot]]$.
The rest easily follows from the fact that there is a one-to-one correspondence between the evaluation of a λ^{T} expression in λ^{ST} and in λ^{T}, and from noting that the reduction rule for ${}^{\mathsf{bool}}\mathcal{ST}\cdot$ simply converts **true** to true and **false** to false. □

12 CPS Translation is Equivalence Preserving

Theorem 12.1 (CPS Translation is Equivalence Preserving)
If $\Gamma \vdash \mathsf{e}_1 : \sigma \rightsquigarrow \mathbf{v}_1$, $\Gamma \vdash \mathsf{e}_2 : \sigma \rightsquigarrow \mathbf{v}_2$, and $\Gamma \vdash \mathsf{e}_1 \precsim^{ctx}_{\mathrm{S}} \mathsf{e}_2 : \sigma$, then $\cdot;\Gamma^{+} \vdash \mathbf{v}_1 \precsim^{ctx}_{\mathrm{T}} \mathbf{v}_2 : \sigma^{\div}$.

Proof

Immediate from Lemmas 10.4 (part 1), 9.7 (part 2), and 11.1 (part 3). □

13 CPS Translation is Equivalence Reflecting

Equivalence reflection is a direct consequence of semantics preservation. Semantics preservation states that source programs (closed λ^{S} terms of base type, i.e., bool) and their translations in λ^{T} behave analogously:

Lemma 13.1 (Semantics preservation)
Let $\cdot \vdash \mathsf{e} : \mathsf{bool} \rightsquigarrow \mathbf{v}$. *If* $\mathsf{e} \longmapsto^* \mathsf{true}$ *then* $\mathbf{v}\,[\mathbf{bool}]\,\mathbf{id} \longmapsto^* \mathbf{true}$ *and if* $\mathsf{e} \longmapsto^* \mathsf{false}$ *then* $\mathbf{v}\,[\mathbf{bool}]\,\mathbf{id} \longmapsto^* \mathbf{false}$.

Proof

Immediate from Lemma 9.6. □

The next observation we need concerns the structural behavior of the CPS translation:

Lemma 13.2 (Context translation)
Let $\Gamma \vdash \mathsf{C}[\mathsf{e}_1] : \sigma$ *and* $\Gamma \vdash \mathsf{C}[\mathsf{e}_2] : \sigma$. *Then there exist* $\mathbf{C}$, $\mathbf{v}_1$, $\mathbf{v}_2$ *such that* $\Gamma \vdash \mathsf{C}[\mathsf{e}_1] : \sigma \rightsquigarrow \mathbf{C}[\mathbf{v}_1]$ *and* $\Gamma \vdash \mathsf{C}[\mathsf{e}_2] : \sigma \rightsquigarrow \mathbf{C}[\mathbf{v}_2]$.

Proof

By induction on the structure of C, using the definition of the CPS translation relation. □

Equivalence reflection now follows almost immediately from Lemmas 13.1 and 13.2:

Theorem 13.3 (Equivalence reflection)
Let $\Gamma \vdash \mathsf{e}_1 : \sigma \rightsquigarrow \mathbf{v}_1$ *and* $\Gamma \vdash \mathsf{e}_2 : \sigma \rightsquigarrow \mathbf{v}_2$. *If* $\cdot; \Gamma^+ \vdash \mathbf{v}_1 \approx^{ctx}_{\mathrm{T}} \mathbf{v}_2 : \sigma^{\div}$ *then* $\Gamma \vdash \mathsf{e}_1 \approx^{ctx}_{\mathrm{S}} \mathsf{e}_2 : \sigma$.

Proof

By contradiction: Suppose the conclusion does not hold, which means there exist some C such that $\cdot \vdash \mathsf{C}[\mathsf{e}_1] : \mathsf{bool}$ and $\cdot \vdash \mathsf{C}[\mathsf{e}_2] : \mathsf{bool}$ where (w.l.o.g.) $\mathsf{C}[\mathsf{e}_1] \longmapsto^* \mathsf{true}$ while $\mathsf{C}[\mathsf{e}_2] \longmapsto^* \mathsf{false}$. By Lemma 13.2 there must exist a $\mathbf{C}$ such that $\cdot \vdash \mathsf{C}[\mathsf{e}_1] : \mathsf{bool} \rightsquigarrow \mathbf{C}[\mathbf{v}_1]$ and $\cdot \vdash \mathsf{C}[\mathsf{e}_2] : \mathsf{bool} \rightsquigarrow \mathbf{C}[\mathbf{v}_2]$. At this point Lemma 13.1 tells us that $(\mathbf{C}[\mathbf{v}_1])\,[\mathbf{bool}]\,\mathbf{id} \longmapsto^* \mathbf{true}$ while $(\mathbf{C}[\mathbf{v}_2])\,[\mathbf{bool}]\,\mathbf{id} \longmapsto^* \mathbf{false}$. Thus, $(\mathbf{C}[\cdot])\,[\mathbf{bool}]\,\mathbf{id}$ is a context that discriminates between $\mathbf{v}_1$ and $\mathbf{v}_2$, which is a contradiction. □

Acknowledgements

The first author would like to thank Greg Morrisett for suggesting the problem of fully abstract compilation to her in Spring 2005. We thank Kyle Ross for helping with the proofs of Lemmas 8.16, 8.17, 8.18, 9.4, and 9.5 in Spring 2010. We are also grateful to Amr Sabry and several anonymous reviewers for many helpful suggestions on earlier versions of this paper.

References

[1] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.

[2] Amal Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *European Symposium on Programming (ESOP)*, pages 69–83, March 2006.

[3] Amal Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. Technical Report TR-01-06, Harvard University, January 2006. Available at `http://www.cs.indiana.edu/∼amal/papers/lr-recquant-techrpt.pdf`.

[4] Amal Ahmed and Matthias Blume. Typed closure conversion preserves observational equivalence. In *International Conference on Functional Programming (ICFP), Victoria, British Columbia, Canada*, pages 157–168, September 2008.

[5] Amal Ahmed, Derek Dreyer, and Andreas Rossberg. State-dependent representation independence. In *ACM Symposium on Principles of Programming Languages (POPL), Savannah, Georgia*, January 2009.

[6] Andrew W. Appel. *Compiling with Continuations.* Cambridge University Press, 1992.

[7] Nick Benton and Chung-Kil Hur. Biorthogonality, step-indexing and compiler correctness. In *International Conference on Functional Programming (ICFP), Edinburgh, Scotland*, September 2009.

[8] Nick Benton and Chung-Kil Hur. Realizability and compositional compiler correctness for a polymorphic language. Technical Report MSR-TR-2010-62, Microsoft Research, April 2010.

[9] Josh Berdine, Peter O'Hearn, Uday Reddy, and Hayo Thielecke. Linear continuation-passing. *Higher Order Symbol. Comput.*, 15(2-3):181–208, 2002.

[10] Josh Berdine, Peter O'Hearn, and Hayo Thielecke. Extracting the range of cps from affine typing: Extended abstract. In *Workshop on Linear Logic*, 2002.

[11] Joshua Berdine. Linear and affine typing of continuation-passing style. Technical Report RR-04-04, Queen Mary, Univ. of London, January 2004.

[12] Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In *Proceedings of the 5th international conference on Typed lambda calculi and applications*, TLCA'01, pages 29–45, 2001.

[13] Martin Berger, Kohei Honda, and Nobuko Yoshida. Genericity and the π-calculus. In *Proceedings of the 6th International conference on Foundations of Software Science and Computation Structures and joint European conference on Theory and practice of software*, FOSSACS'03/ETAPS'03, pages 103–119, 2003.

[14] Martin Berger, Kohei Honda, and Nobuko Yoshida. Genericity and the π-calculus. *Acta Informatica*, 42:83–141, November 2005.

[15] Robert Cartwright and Matthias Felleisen. Observable sequentiality and full abstraction. In *ACM Symposium on Principles of Programming Languages (POPL), Albuquerque, New Mexico*, pages 328–342, 1992.

[16] Adam Chlipala. A certified type-preserving compiler from lambda calculus to assembly language. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), San Diego, California*, June 2007.

[17] Olivier Danvy. Back to direct style. *Science of Computer Programming*, 22(3):183–195, 1994.

[18] Andrzej Filinski. Representing monads. In *ACM Symposium on Principles of Programming Languages (POPL), Portland, Oregon*, January 1994.

[19] Masahito Hasegawa. Linearly used effects: Monadic and CPS transformations into the linear lambda calculus. In *International Symposium on Functional and Logic Programming (FLOPS), Aizu, Japan*, pages 167–182, 2002.

[20] Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. In *ACM Symposium on Principles of Programming Languages (POPL), Portland, Oregon*, January 2002.

[21] Kohei Honda, Nobuko Yoshida, and Martin Berger. Control in the π-calculus. In *Fourth ACM-SIGPLAN Continuations Workshop (CW '04)*, January 2004.

[22] J. M. E. Hyland and C. H. Luke Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000.

[23] Alan Jeffrey. A fully abstract semantics for a concurrent functional language with monadic types. In *IEEE Symposium on Logic in Computer Science (LICS), San Diego, California*, 1995.

[24] Andrew Kennedy. Compiling with continuations, continued. In *International Conference on Functional Programming (ICFP), Freiburg, Germany*, October 2007.

[25] David A. Kranz, Richard A. Kelsey, Jonathan A. Rees, Paul Hudak, and James Philbin. ORBIT: an optimizing compiler for Scheme. In *Proceedings of the ACM Symposium on Compiler Construction*, pages 219–233, June 1986.

[26] J. Laird. Game semantics and linear CPS interpretation. *Theor. Comput. Sci.*, 333(1-2):199–224, 2005.

[27] A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables. In *ACM Symposium on Principles of Programming Languages (POPL), San Diego, California*, pages 191–203, 1988.

[28] Albert Meyer and Jon G. Riecke. Continuations may be unreasonable. In *Conf. on LISP and functional programming, LFP '88*, pages 63–71, 1988.

[29] Albert R. Meyer and Mitchell Wand. Continuation semantics in typed lambda-calculi. In R. Parikh, editor, *Logics of Programs (Brooklyn, June, 1985)*, volume 193 of *Lecture Notes in Computer Science*, pages 219–224. Springer-Verlag, 1985.

[30] Robin Milner. Fully abstract models of typed lambda calculi. *Theoretical Computer Science*, 4(1), 1977.

[31] Greg Morrisett, David Walker, Karl Crary, and Neal Glew. From System F to typed assembly language. *ACM Transactions on Programming Languages and Systems*, 21(3):527–568, May 1999.

[32] Aleksandar Nanevski, Amal Ahmed, Greg Morrisett, and Lars Birkedal. Abstract predicates and mutable adts in hoare type theory. In *European Symposium on Programming (ESOP)*, pages 189–204, March 2007.

[33] Andrew M. Pitts. Typed operational reasoning. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*. MIT Press, 2005.

[34] Gordon D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[35] Jon Riecke and Ramesh Viswanathan. Isolating side effects in sequential languages. In *ACM Symposium on Principles of Programming Languages (POPL), San Francisco, California*, January 1995.

[36] Jon G. Riecke. Fully abstract translations between functional languages. In *ACM Symposium on Principles of Programming Languages (POPL), Orlando, Florida*, pages 245–254, 1991.

[37] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In *Conf. on LISP and functional programming, LFP '92*, 1992.

[38] S. B. Sanjabi and C.-H. L. Ong. Fully abstract semantics of additive aspects by translation. In *Proceedings of the 6th international conference on Aspect-oriented software development (AOSD)*, pages 135–148, 2007.

[39] Zhong Shao and Andrew W. Appel. A type-based compiler for Standard ML. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), La Jolla, California*, pages 116–129. ACM Press, 1995.

[40] Naokata Shikuma and Atsushi Igarashi. Proving noninterference by a fully complete translation to the simply typed lambda-calculus. *Logical Methods in Computer Science*, 4(3:10):1–31, 2008.

[41] Guy L. Steele. RABBIT: A compiler for SCHEME. Technical Report AI-TR-474, MIT, May 1978.

[42] Hayo Thielecke. From control effects to typed continuation passing. In *ACM Symposium on Principles of Programming Languages (POPL), New Orleans, Louisiana*, 2003.

[43] Hayo Thielecke. Answer type polymorphism in call-by-name continuation passing. In *European Symposium on Programming (ESOP)*, March 2004.

[44] Philip Wadler. Theorems for free! In *ACM Symposium on Functional Programming Languages and Computer Architecture (FPCA)*, September 1989.

[45] Steve Zdancewic and Andrew C. Myers. Secure information flow and CPS. In *European Symposium on Programming (ESOP)*, pages 46–61, April 2001.