Type-Preserving CPS Translation of Σ and Π-Types is Not Not Possible (Technical Appendix)

WILLIAM J. BOWMAN, Northeastern University, USA and Inria Paris, France
YOUYOU CONG, Ochanomizu University, Japan
NICK RIOUX, Northeastern University, USA
AMAL AHMED, Northeastern University, USA and Inria Paris, France

TECHNICAL APPENDIX

This document includes extended figures, proofs, and discussion for the paper "Type-Preserving CPS Translation of Σ and Π-types is Not Not Possible". It excludes introductory materials, including only extended versions of Sections 2, 4, 5, and 6 from the paper. It also includes a discussion of extending the work to type-level strong dependent pairs, instead of only term-level pairs.

1 THE CALCULUS OF CONSTRUCTIONS (CC)

Our source language is an extension of the intensional Calculus of Constructions (CC) with strong dependent pairs (Σ types) and dependent let. We typeset this language in a non-bold, blue, sans-serif font. We adapt this presentation from the model of the Calculus of Inductive Constructions (CIC) given in the Coq reference manual [The Coq Development Team 2017, Chapter 4].

We present the syntax of CC in Figure 1 in the style of a Pure Type System (PTS) with no syntactic distinction between terms, which are run-time computations, types, which statically describe terms and compute during type checking, and kinds, which describe types. We use the phrase "expression" to refer to a term, type, or kind in the PTS syntax. We usually use the meta-variable e to evoke a term expression and A or B to evoke a type expression. Similarly, we use x to evoke term variables and α for type variables; note that we have no kind-level computation in this language. We use t for an expression to be explicitly ambiguous about its nature as a term, type, or kind.

The language includes one impredicative *universe*, or *sort*, $*$, and its type, $\Box$. The syntax of expressions includes the universe $*$, variables x or α, Π types $\Pi\, x : A.\, B$, functions $\lambda\, x : A.\, e$, application $e_1\ e_2$, dependent let $\mathsf{let}\ x = e : A\ \mathsf{in}\ e'$, Σ types $\Sigma\, x : A.\, B$, dependent pairs $\langle e_1, e_2 \rangle\ \mathsf{as}\ \Sigma\, x : A.\, B$, and first and second projections $\mathsf{fst}\ e$ and $\mathsf{snd}\ e$. Note that we cannot write $\Box$ in source programs—it is only used by the type system. The environment Γ includes assumptions $x : A$ and definitions $x = e : A$. Definitions, introduced while type checking let, allow us to convert a variable x to its definition e, called δ-reduction, and provides additional definitional equivalences compared to application.

For brevity, we omit the type annotations on pairs, $\langle e_1, e_2 \rangle$, and let expressions, $\mathsf{let}\ x = e\ \mathsf{in}\ e'$, when they are clear from context. We use the notation $A \to B$ for a function type whose result B does not depend on the input.

In Figure 2 we present the convertibility and equivalence relations for CC. These relations are defined over *untyped* expressions and are used to decide equivalences between types during type checking. The conversion relation can also be seen as the dynamic semantics of programs in CC. It does not fix an evaluation order, but this is not important since CC is effect-free.

We start with the small-step reductions $\Gamma \vdash e \rhd e'$. Note that we label each individual reduction rule with an appropriate subscript, such as $\rhd_\beta$ for β-reduction. When we refer to the undecorated transition $\Gamma \vdash e \rhd e'$ we mean that e reduces to e' using *some* reduction rule— *i.e.*, using one of $\rhd_\delta$, $\rhd_\beta$, $\rhd_\zeta$, $\rhd_{\pi_1}$, or $\rhd_{\pi_2}$. This relation requires the environment Γ for δ-reduction as mentioned previously. For brevity, we usually write this relation as $e \rhd e'$, with the environment Γ as an implicit parameter. This reduction relation is completely standard, although δ-reduction may be surprising to readers unfamiliar with dependent type theory. We can δ-reduce any variable x to its definition e, written $x \rhd_\delta e$.

We define the relation $\Gamma \vdash e \rhd^\star e'$ as the reflexive, transitive, compatible closure of the small-step relation $\Gamma \vdash e \rhd e'$. This relation can apply the small-step relation any number of times to any sub-expression in any order. We usually

Authors' addresses: William J. Bowman, Northeastern University, USA , Inria Paris, France, wjb@williamjbowman.com; Youyou Cong, Ochanomizu University, Japan, so.yuyu@is.ocha.ac.jp; Nick Rioux, Northeastern University, USA, rioux.n@husky.neu.edu; Amal Ahmed, Northeastern University, USA , Inria Paris, France, amal@ccs.neu.edu.

$$
\begin{array}{llll}
\textit{Universes} & U & ::= & * \mid \Box \\
\textit{Expressions} & t, e, A, B & ::= & * \mid x \mid \Pi\, x : A.\, e \mid \lambda\, x : A.\, e \mid e\ e \mid \mathsf{let}\ x = e : A\ \mathsf{in}\ e \mid \Sigma\, x : A.\, B \\
& & \mid & \langle e_1, e_2 \rangle\ \mathsf{as}\ \Sigma\, x : A.\, B \mid \mathsf{fst}\ e \mid \mathsf{snd}\ e \\
\textit{Environments} & \Gamma & ::= & \cdot \mid \Gamma, x : A \mid \Gamma, x = e : A
\end{array}
$$

Fig. 1. CC Syntax

$\boxed{\Gamma \vdash e \rhd e'}$

$$
\begin{array}{rcll}
x & \rhd_\delta & e & \text{where } x = e : A \in \Gamma \\
(\lambda\, x : A.\, e_1)\ e_2 & \rhd_\beta & e_1[e_2/x] & \\
\mathsf{let}\ x = e_2 : A\ \mathsf{in}\ e_1 & \rhd_\zeta & e_1[e_2/x] & \\
\mathsf{fst}\ \langle e_1, e_2 \rangle & \rhd_{\pi_1} & e_1 & \\
\mathsf{snd}\ \langle e_1, e_2 \rangle & \rhd_{\pi_2} & e_2 &
\end{array}
$$

$\boxed{\Gamma \vdash e \equiv e'}$

$$
\frac{\Gamma \vdash e \rhd^\star e_1 \qquad \Gamma \vdash e' \rhd^\star e_1}{\Gamma \vdash e \equiv e'}\ [\equiv] \qquad \frac{\Gamma \vdash e \rhd^\star \lambda\, x : A.\, e_1 \qquad \Gamma \vdash e' \rhd^\star e_2 \qquad \Gamma, x : A \vdash e_1 \equiv e_2\ x}{\Gamma \vdash e \equiv e'}\ [\equiv\text{-}\eta_1]
$$

$$
\frac{\Gamma \vdash e \rhd^\star e_1 \qquad \Gamma \vdash e' \rhd^\star \lambda\, x : A.\, e_2 \qquad \Gamma, x : A \vdash e_1\ x \equiv e_2}{\Gamma \vdash e \equiv e'}\ [\equiv\text{-}\eta_2]
$$

Fig. 2. CC Convertibility and Equivalence

omit the Γ and write $e \rhd^\star e'$ for brevity, but note that the compatible closure rule for let introduces a new definition into Γ, as follows.

$$
\frac{\Gamma, x = e : A \vdash e_1 \rhd^\star e_2}{\Gamma \vdash \mathsf{let}\ x = e : A\ \mathsf{in}\ e_1 \rhd^\star \mathsf{let}\ x = e : A\ \mathsf{in}\ e_2}
$$

We define definitional equivalence $\Gamma \vdash e \equiv e'$ as reduction in the $\rhd^\star$ relation to the same expression, up to η-equivalence. This algorithmic presentation induces symmetry and transitivity of $\equiv$ without explicit symmetry and transitivity rules, but requires two symmetric versions of η-equivalence. We usually abbreviate this judgment as $e \equiv e'$, leaving Γ implicit.

The typing rules for CC, Figure 3, are completely standard. The judgment $\vdash \Gamma$ checks that the environment Γ is well formed; it is defined by mutual recursion with the typing judgment. The typing judgment $\Gamma \vdash e : A$ checks that expressions are well typed. The rule [Prod-*] implicitly allows impredicativity in $*$, since the domain A could be in the higher universe $\Box$. The rule [Lam] for functions $\lambda\, x : A.\, e$ gives this function the type $\Pi\, x : A.\, B$, binding the function's variable x in the result type B. The rule [App] is the standard dependent application rule. When applying a dependent function $e : \Pi\, x : A.\, B$ to an argument e', the argument is substituted into the result type B yielding an expression $e\ e' : B[e'/x]$. The rule [Let] is similar; however, when checking the body of $\mathsf{let}\ x = e' : A\ \mathsf{in}\ e$, we also adds a definition $x = e' : A$ to the environment. This provides strictly more type expressivity than the application rule, since the body e is typed with respect to a particular value for x while a function is typed with respect to an arbitrary value. The rule [Sigma] ensures we do not allow impredicative strong Σ types, which are inconsistent [Coquand 1986; Hook and Howe 1986]. Note that the type of a dependent pair $\Sigma\, x : A.\, B$ may have the first component x free in the type of the second component B. The rule [Snd] for the second projection of a dependent pair, $\mathsf{snd}\ e$, replaces the free variable x by the first projection, giving $\mathsf{snd}\ e$ the dependent type $B[(\mathsf{fst}\ e)/x]$. Finally, as we have computation in types, the rule [Conv] allows typing an expression $e : A$ as $e : B$ when $A \equiv B$. Note that while the equivalence relation is untyped, we ensure decidability by only using equivalence in [Conv] after type checking both A and B.

$\boxed{\vdash \Gamma}$

$$\frac{}{\vdash \cdot}\ [\text{W-Empty}] \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A : U}{\vdash \Gamma, x : A}\ [\text{W-Assum}] \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash e : A \qquad \Gamma \vdash A : U}{\vdash \Gamma, x = e : A}\ [\text{W-Def}]$$

$\boxed{\Gamma \vdash e : A}$

$$\frac{\vdash \Gamma}{\Gamma \vdash * : \Box}\ [\text{Ax-}*] \qquad \frac{(x : A \in \Gamma \text{ or } x = e : A \in \Gamma) \qquad \vdash \Gamma}{\Gamma \vdash x : A}\ [\text{Var}] \qquad \frac{\Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi\, x : A.\, B : *}\ [\text{Prod-}*]$$

$$\frac{\Gamma, x : A \vdash B : \Box}{\Gamma \vdash \Pi\, x : A.\, B : \Box}\ [\text{Prod-}\Box] \qquad \frac{\Gamma, x : A \vdash e : B \qquad \Gamma \vdash \Pi\, x : A.\, B : U}{\Gamma \vdash \lambda\, x : A.\, e : \Pi\, x : A.\, B}\ [\text{Lam}]$$

$$\frac{\Gamma \vdash e : \Pi\, x : A'.\, B \qquad \Gamma \vdash e' : A'}{\Gamma \vdash e\ e' : B[e'/x]}\ [\text{App}] \qquad \frac{\Gamma \vdash e' : A \qquad \Gamma, x = e' : A \vdash e : B}{\Gamma \vdash \text{let } x = e' : A \text{ in } e : B[e'/x]}\ [\text{Let}]$$

$$\frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Sigma\, x : A.\, B : *}\ [\text{Sigma}] \qquad \frac{\Gamma \vdash e_1 : A \qquad \Gamma \vdash e_2 : B[e_1/x]}{\Gamma \vdash \langle e_1, e_2 \rangle \text{ as } \Sigma\, x : A.\, B : \Sigma\, x : A.\, B}\ [\text{Pair}] \qquad \frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \text{fst } e : A}\ [\text{Fst}]$$

$$\frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \text{snd } e : B[\text{fst } e/x]}\ [\text{Snd}] \qquad \frac{\Gamma \vdash e : A \qquad \Gamma \vdash B : U \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash e : B}\ [\text{Conv}]$$

Fig. 3. CC Typing

Kinds	κ	::=	$* \mid \Pi\, \alpha : \kappa.\, \kappa \mid \Pi\, x : A.\, \kappa$
Types	A, B	::=	$\alpha \mid \lambda\, x : A.\, B \mid \lambda\, \alpha : \kappa.\, B \mid A\ e \mid A\ B \mid \Pi\, x : A.\, B \mid \Pi\, \alpha : \kappa.\, B \mid \text{let } x = e : A \text{ in } B$
		|	$\text{let } \alpha = A : \kappa \text{ in } B \mid \Sigma\, x : A.\, B$
Terms	e	::=	$x \mid \lambda\, x : A.\, e \mid \lambda\, \alpha : \kappa.\, e \mid e\ e \mid e\ A \mid \text{let } x = e : A \text{ in } e \mid \text{let } \alpha = A : \kappa \text{ in } e$
		|	$\langle e_1, e_2 \rangle \text{ as } \Sigma\, x : A.\, B \mid \text{fst } e \mid \text{snd } e$
Environment	Γ	::=	$\cdot \mid \Gamma, x : A \mid \Gamma, x = e : A, \mid \Gamma, \alpha : \kappa \mid \Gamma, \alpha = A : \kappa$

Fig. 4. CC Explicit Syntax

Note that for simplicity, we include only term-level pairs of type $\Sigma\, x : A.\, B : *$. Type-level pairs $\Sigma\, x : A.\, B : \Box$ introduce numerous minor difficulties. For instance, we can write pairs of terms and types $\langle e, A \rangle$ or $\langle A, e \rangle$. In some cases, it is unclear how this expression should be CPS translated. We discuss type-level pairs further in Section 5.

To make our upcoming CPS translation easier to follow, we present a second version of the syntax for CC in which we make the distinction between terms, types, and kinds explicit (see Figure 4). The two presentations are equivalent [Barthe et al. 1999]. Distinguishing terms from types and kinds is useful since we only want to CPS translate *terms*, because our goal is to internalize only *run-time* evaluation contexts.

$$
\begin{array}{llll}
\textit{Universes} & \mathbf{U} & ::= & * \mid \Box \\
\textit{Terms/Types} & \mathbf{e}, \mathbf{A}, \mathbf{B} & ::= & \mathbf{x} \mid \lambda\, \mathbf{x} : \mathbf{A}.\, \mathbf{e} \mid \mathbf{e}\ \mathbf{e} \mid \Pi\, \mathbf{x} : \mathbf{A}.\, \mathbf{e} \mid * \mid \mathbf{e}\ @\ \mathbf{A}\ \mathbf{e} \mid \mathsf{let}\, \mathbf{x} = \mathbf{e} : \mathbf{A}\, \mathsf{in}\, \mathbf{e} \mid \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B} \\
 & & \mid & \mathsf{fst}\, \mathbf{e} \mid \mathsf{snd}\, \mathbf{e} \mid \langle \mathbf{e}_1, \mathbf{e}_2 \rangle\, \mathsf{as}\, \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B} \\
\textit{Local Environment} & \Gamma & ::= & \cdot \mid \Gamma, \mathbf{x} : \mathbf{A} \mid \Gamma, \mathbf{x} = \mathbf{e} : \mathbf{A}
\end{array}
$$

Fig. 5. CC^k Syntax

2 THE CALCULUS OF CONSTRUCTIONS WITH CPS AXIOMS (CC^K)

Our target language CC^k is CC extended with a syntax for parametric reasoning about computations in CPS form. . We add the form $\mathbf{e}\ @\ \mathbf{A}\ \mathbf{e}'$ to the syntax of CC^k, Figure 5. This form represents a computation $\mathbf{e}$ applied to the answer type $\mathbf{A}$ and the continuation $\mathbf{e}'$. The dynamic semantics, Figure 6, are the same as standard application. The equivalence rule [≡-Cont] states that a computation $\mathbf{e}$ applied to its continuation $\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{e}'$ is equivalent to the application of that continuation to the underlying value of $\mathbf{e}$. We extract the underlying value by applying $\mathbf{e}$ to the "halt continuation", encoded as the identity function in our system. The rule [T-Cont] is used to type check applications that use our new @ syntax. This typing (Figure 7) rule internalizes the fact that a continuation will be applied to one particular input, rather than an arbitrary value. It tells the type system that the application of a computation to a continuation $\mathbf{e}\ @\ \mathbf{A}\ (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{e}')$ jumps to the continuation $\mathbf{e}'$ after evaluating $\mathbf{e}$ to a value and binding the result to $\mathbf{x}$. We check the body of the continuation $\mathbf{e}'$ under the assumption that $\mathbf{x} = \mathbf{e}\ \mathbf{B}\ \mathbf{id}$, *i.e.*, with the equality that the name $\mathbf{x}$ refers to the underlying value in the computation $\mathbf{e}$, which we access using the interface given by the polymorphic answer type.

The rule [≡-Cont] (Figure 6) is a declarative rule that requires explicit symmetry and transitivity rules to complete the definition. We give a declarative presentation of this rule for clarity. The algorithmic versions might look something like the following.

$$\frac{\Gamma \vdash \mathbf{e} \rhd^\star (\mathbf{e}_1\ @\ \mathbf{A}\ (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{e}_2)) \qquad \Gamma \vdash (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{e}_2)\ (\mathbf{e}_1\ \mathbf{B}\ \mathbf{id}) \equiv \mathbf{e}'}{\Gamma \vdash \mathbf{e} \equiv \mathbf{e}'}\ [\equiv\text{-Cont}_1]$$

$$\frac{\Gamma \vdash \mathbf{e}' \rhd^\star (\mathbf{e}_1\ @\ \mathbf{A}\ (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{e}_2)) \qquad \Gamma \vdash \mathbf{e} \equiv (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{e}_2)\ (\mathbf{e}_1\ \mathbf{B}\ \mathbf{id})}{\Gamma \vdash \mathbf{e} \equiv \mathbf{e}'}\ [\equiv\text{-Cont}_2]$$

We have not shown that these rules induce symmetry and transitivity.

Note that [≡-Cont] and [T-Cont] internalize a specific "free theorem" that we need to prove type preservation of the CPS translation. In particular, [≡-Cont] only holds when the CPS'd term $\mathbf{e}_1$ has the expected parametric type $\Pi\, \alpha : *.\, (\mathbf{A} \to \alpha) \to \alpha$ given in [T-Cont]. Notice, however, that our statement of [≡-Cont] does not put any requirements on the type of $\mathbf{e}_1$. This is because we use an untyped equivalence based on the presentation of CIC in Coq [The Coq Development Team 2017, Chapter 4], and this untyped equivalence is necessary in our type-preservation proof (see Section 3.1). Therefore, we cannot simply add typing assumptions directly to [≡-Cont]. Instead, we rely on the fact that the term $\mathbf{e}\ @\ \mathbf{A}\ \mathbf{e}'$ has only one introduction rule, [T-Cont]. Since there is only one applicable typing rule, anytime $\mathbf{e}\ @\ \mathbf{A}\ \mathbf{e}'$ appears in our type system, $\mathbf{e}$ has the required parametric type. Furthermore, while our equivalence is untyped, we never appeal to equivalence with ill-typed terms; we only refer to the equivalence $\mathbf{A}' \equiv \mathbf{B}'$ in [Conv] *after* checking that both $\mathbf{A}'$ and $\mathbf{B}'$ are well-typed. For example, suppose the term $\mathbf{e}\ @\ \mathbf{A}\ \mathbf{e}'$ occurs in type $\mathbf{A}'$, and to prove that $\mathbf{A}' \equiv \mathbf{B}'$ requires our new rule [≡-Cont]. Because $\mathbf{A}'$ is well-typed, we know that its subterms, including $\mathbf{e}\ @\ \mathbf{A}\ \mathbf{e}'$, are well-typed. Since $\mathbf{e}\ @\ \mathbf{A}\ \mathbf{e}'$ can only be well-typed by [T-Cont], we know $\mathbf{e}$ has the required parametric type.

Finally, notice that in [T-Cont] and [≡-Cont] we use standard application syntax for the term $\mathbf{e}\ \mathbf{B}\ \mathbf{id}$. We only use the @ syntax in our CPS translation when we require one of our new rules. The type of the identity function doesn't depend on any value, so we never need [T-Cont] to type-check the identity continuation. In a sense, $\mathbf{e}\ \mathbf{B}\ \mathbf{id}$ is the normal form of a CPS'd "value" so we never need [≡-Cont] to rewrite this term—*i.e.*, using [≡-Cont] to rewrite $\mathbf{e}\ \mathbf{B}\ \mathbf{id}$ to $\mathbf{id}\ (\mathbf{e}\ \mathbf{B}\ \mathbf{id})$ would just evaluate the original term.

$\boxed{\Gamma \vdash e \rhd e'}$

$$
\begin{array}{rcll}
x & \rhd_\delta & e & \text{where } x = e \in \Gamma \\
(\lambda\, x : A.\, e_1)\; e_2 & \rhd_\beta & e_1[e_2/x] & \\
\text{let } x = e_2 : A \text{ in } e_1 & \rhd_\zeta & e_1[e_2/x] & \\
\lambda\, \alpha : *.\, e_1 \;@\; A\; e_2 & \rhd_@ & (e_1[A/\alpha])\; e_2 &
\end{array}
$$

$\boxed{\Gamma \vdash e \equiv e'}$

$$\frac{\Gamma \vdash e \rhd^\star e_1 \qquad \Gamma \vdash e' \rhd^\star e_1}{\Gamma \vdash e \equiv e'}\ [\equiv] \qquad \frac{\Gamma \vdash e \rhd^\star \lambda\, x : A.\, e_1 \qquad \Gamma \vdash e' \rhd^\star e_2 \qquad \Gamma, x : A \vdash e_1 \equiv e_2\; x}{\Gamma \vdash e \equiv e'}\ [\equiv\text{-}\eta_1]$$

$$\frac{\Gamma \vdash e \rhd^\star e_1 \qquad \Gamma \vdash e' \rhd^\star \lambda\, x : A.\, e_2 \qquad \Gamma, x : A \vdash e_1\; x \equiv e_2}{\Gamma \vdash e \equiv e'}\ [\equiv\text{-}\eta_2]$$

$$\frac{}{\Gamma \vdash (e_1\; @\; A\; (\lambda\, x : B.\, e_2)) \equiv (\lambda\, x : B.\, e_2)\; (e_1\; B\; \mathrm{id})}\ [\equiv\text{-Cont}] \qquad \frac{\Gamma \vdash e' \equiv e}{\Gamma \vdash e \equiv e'}\ [\equiv\text{-Sym}]$$

$$\frac{\Gamma \vdash e \equiv e_1 \qquad \Gamma \vdash e_1 \equiv e'}{\Gamma \vdash e \equiv e'}\ [\equiv\text{-Trans}]$$

Fig. 6. CC^k Convertibility and Equivalence

2.1 Consistency of CC^k

We prove that CC^k is consistent by giving a model of CC^k in the extensional Calculus of Constructions. Boulier et al. [2017] provide a detailed explanation of this standard technique.

The idea behind the model is that we can translate each use of [≡-Cont] in CC^k to a propositional equivalence in extensional CC. Next, we translate any term that is typed by [T-Cont] into a dependent let. Finally, we establish that if there were a proof of **False** in CC^k, our translation would construct a proof of *False* in extensional CC. But since extensional CC is consistent, there can be no proof of **False** in CC^k. We construct the model in three parts.

(1) produce proofs (in extensional CC) of all propositional equivalences introduced by our translation of [≡-Cont],
(2) show **False** in CC^k is translated to *False* in extensional CC,
(3) show our translation is type preserving, *i.e.*, it translates a proof of **A** to a proof of its translation $\mathbf{A}^\circ$.

As our model is in the extensional CC, it is not clear that type checking in CC^k is decidable. We believe that type checking should be decidable for all programs produced by our compiler, since type checking in our source language CC is decidable. In the worst case, to ensure decidability we could change our translation to use a propositional version of [≡-Cont]. The definitional presentation is simpler, but it should be possible to change the translation so that, in any term that currently relies on [≡-Cont], we insert type annotations that compute type equivalence using a propositional version of [≡-Cont]. We leave the issue of decidability of type checking in CC^k for future work.

2.1.1 Modeling [≡-Cont]. The extensional Calculus of Constructions differs from our source language CC in only one way: it allows using the existence of a propositional equivalence as a definitional equivalence, as shown in Figure 8. The syntax and typing rules are exactly the same as in CC presented in Section 1. We write terms in extensional CC using a *italic, black, serif font.*

In extensional CC we can model each use of the definitional equivalence [≡-Cont] by [≡-Ext], as long as there exists a proof $p : (e\ A\ k) = (k\ (e\ B\ id))$, *i.e.*, a propositional proof of [≡-Cont]; we prove this propositional proof always exists by using the parametricity translation of Keller and Lasson [2012]. This translation gives a parametric model of CC in itself. This translation is based on prior translations that apply to all Pure Type Systems [Bernardy et al. 2012], but includes an impredicative universe and provides a Coq implementation that we use.

The translation of a type A, written $[\![A]\!]$, essentially transforms the type into a relation on terms of that type. On terms e of type A, the translation $[\![e]\!]$ produces a proof that e is related to itself in the relation given by $[\![A]\!]$. For

$\boxed{\vdash \Gamma}$

$$\frac{}{\vdash \cdot}\ [\text{W-Empty}] \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A : U}{\vdash \Gamma, x : A}\ [\text{W-Assum}] \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash e : A \qquad \Gamma \vdash A : U}{\vdash \Gamma, x = e : A}\ [\text{W-Def}]$$

$\boxed{\Gamma \vdash e : A}$

$$\frac{\vdash \Gamma}{\Gamma \vdash * : \square}\ [\text{Ax-*}] \qquad \frac{(x : A \in \Gamma \text{ or } x = e : A \in \Gamma) \qquad \vdash \Gamma}{\Gamma \vdash x : A}\ [\text{Var}] \qquad \frac{\Gamma, x : A \vdash B : *}{\Gamma \vdash \Pi\, x : A.\, B : *}\ [\text{Prod-*}]$$

$$\frac{\Gamma, x : A \vdash B : \square}{\Gamma \vdash \Pi\, x : A.\, B : \square}\ [\text{Prod-}\square] \qquad \frac{\Gamma, x : A \vdash e : B \qquad \Gamma \vdash \Pi\, x : A.\, B : U}{\Gamma \vdash \lambda\, x : A.\, e : \Pi\, x : A.\, B}\ [\text{Lam}]$$

$$\frac{\Gamma \vdash e : \Pi\, x : A'.\, B \qquad \Gamma \vdash e' : A'}{\Gamma \vdash e\ e' : B[e'/x]}\ [\text{App}] \qquad \frac{\Gamma \vdash e : A \qquad \Gamma, x = e : A \vdash e' : B}{\Gamma \vdash \text{let}\, x = e : A\, \text{in}\, e' : B[e/x]}\ [\text{Let}]$$

$$\frac{\Gamma \vdash A : * \qquad \Gamma, x : A \vdash B : *}{\Gamma \vdash \Sigma\, x : A.\, B : *}\ [\text{Sigma}] \qquad \frac{\Gamma \vdash e_1 : A \qquad \Gamma \vdash e_2 : B[e_1/x]}{\Gamma \vdash \langle e_1, e_2 \rangle\, \text{as}\, \Sigma\, x : A.\, B : \Sigma\, x : A.\, B}\ [\text{Pair}] \qquad \frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \text{fst}\, e : A}\ [\text{Fst}]$$

$$\frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \text{snd}\, e : B[\text{fst}\, e/x]}\ [\text{Snd}] \qquad \frac{\Gamma \vdash e : A \qquad \Gamma \vdash B : U \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash e : B}\ [\text{Conv}]$$

$$\frac{\Gamma \vdash e : \Pi\, \alpha : *.\, (B \to \alpha) \to \alpha \qquad \Gamma \vdash A : * \qquad \Gamma, x = e\ B\ \text{id} \vdash e' : A}{\Gamma \vdash e\ @\ A\ (\lambda\, x : B.\, e') : A}\ [\text{T-Cont}]$$

Fig. 7. CC^k Typing

$\boxed{\Gamma \vdash e_1 \equiv e_2}$

$$\ldots \qquad \frac{\Gamma \vdash p : e_1 = e_2}{\Gamma \vdash e_1 \equiv e_2}\ [\equiv\text{-Ext}]$$

Fig. 8. Additional Equivalence Rule for Extensional CC

example, a type $*$ is translated to the relation $[\![*]\!] = \lambda\,(x, x' : *).\, x \to x' \to *$. The translation of a polymorphic function type $[\![\Pi\, \alpha : *.\, A]\!]$ is the following.

$$\lambda\,(f, f' : (\Pi\, \alpha : *.\, A)).\, \Pi\,(\alpha, \alpha' : *).\, \Pi\, \alpha_r : [\![*]\!]\ \alpha\ \alpha'.\, ([\![A]\!]\ (f\ \alpha)\ (f'\ \alpha'))$$

This relation produces a proof that the bodies of functions f and f' are related when provided a relation α_r for the two types of α and α'. This captures the idea that functions at this type must behave parametrically in the abstract type α. This translation gives us Theorem 2.1 (Parametricity for extensional CC), *i.e.*, that every expression in extensional CC is related to itself in the relation given by its type.

Theorem 2.1 (Parametricity for extensional CC). *If* $\Gamma \vdash t : t'$ *then* $[\![\Gamma]\!] \vdash [\![t]\!] : [\![t']\!]\ t\ t$

We apply Theorem 2.1 to our CPS type $\Pi\, \alpha : *.\, (B \to \alpha) \to \alpha$ to prove Lemma 2.2. Since a CPS'd term is a polymorphic function, we get to provide a relation α_r for the type α. The translation then gives us a proof that $e\ A\ k$ and $e\ B\ id$ are related by α_r, so we simply choose α_r to be a relation that guarantees $e\ A\ k = k\ (e\ B\ id)$. We formalize part of the proof in Coq in our supplementary materials [Bowman et al. 2017]. By [≡-Ext], Theorem 2.1, and the relation just described, we arrive at a proof of Lemma 2.2 for CPS'd computations encoded in the extensional CC.

Lemma 2.2 (Continuation Shuffling). *If* $\Gamma \vdash A : *$, $\Gamma \vdash B : *$, $\Gamma \vdash e : \Pi\, \alpha : *.\, (B \to \alpha) \to \alpha$, $\Gamma \vdash k : B \to A$, *and* $\Gamma \vdash e : \Pi\, \alpha : *.\, (B \to \alpha) \to \alpha$ *then* $\Gamma \vdash e\ A\ k \equiv k\ (e\ B\ id)$

$\boxed{\Gamma \vdash \mathbf{e} : \mathbf{A} \rightsquigarrow_\circ e}$

$$\ldots \qquad \frac{\Gamma \vdash \mathbf{e} : \_ \rightsquigarrow_\circ e \qquad \Gamma \vdash \mathbf{B} : \_ \rightsquigarrow_\circ B \qquad \Gamma \vdash \mathbf{A} : \_ \rightsquigarrow_\circ A \qquad \Gamma, \mathbf{x} = \mathbf{e}\ \mathbf{B}\ \mathbf{id} \vdash \mathbf{e}' : \mathbf{A} \rightsquigarrow_\circ e'}{\Gamma \vdash \mathbf{e}\ @\ \mathbf{A}\ (\boldsymbol{\lambda}\, \mathbf{x} : \mathbf{B}.\ \mathbf{e}') : \mathbf{A} \rightsquigarrow_\circ \mathit{let}\ x = e\ B\ \mathit{id} : B\ \mathit{in}\ e'}\ [\textsc{Un-Cont}]$$

Fig. 9. Translation from CC^k to Extensional CC (excerpt)

Note that this lemma relies on the type of the term e. We must only appeal to this lemma, and the equivalence justified by it, when e has the right type. In CC^k, this is guaranteed by the typing rule [T-Cont], as discussed earlier in this section.

2.1.2 Modeling [T-Cont]. In Figure 9 we present the key translation rule for modeling CC^k in extensional CC. All other rules are inductive on the structure of typing derivations. Note that since we only need to justify the additional typing rule [T-Cont], this is the only rule that is changed by the translation. This translation rule is essentially the same rule from the inverse CPS given by Flanagan et al. [1993], although we do not necessarily produce output in A-normal form (ANF) since we only translate uses of this one typing rule.

For brevity in our proofs, we define the following notation for the translation of terms and types from CC^k into extensional CC.

$$\mathbf{e}^\circ \stackrel{\text{def}}{=} e \text{ where } \Gamma \vdash \mathbf{e} : \mathbf{A} \rightsquigarrow_\circ e$$

By writing $\mathbf{e}^\circ$, we refer to the term produced by the translation with the typing derivation $\Gamma \vdash \mathbf{e} : \mathbf{A}$ as an implicit parameter.

First, we show that the definition of False is preserved. We define $\mathbf{False}$ as $\mathbf{\Pi}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\alpha}$, *i.e.*, the function that accepts any proposition and returns a proof that the proposition holds. It is simple to see that this type has type $*$ in CC^k by the rule [Prod-*]. Note that $\mathbf{\Pi}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\alpha}$ is translated to $\Pi\, \alpha : *.\ \alpha$ of type $*$, *i.e.*, $\mathbf{False}$ is translated to *False*.

Lemma 2.3 (False Preservation). $\Gamma \vdash (\mathbf{\Pi}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\alpha}) : * \rightsquigarrow_\circ \Pi\, \alpha : *.\ \alpha$

Next, to show type preservation, we must first show that equivalence is preserved since the type system appeals to equivalence. A crucial lemma to both equivalence preservation and type preservation is *compositionality*, which says that the translation commutes with substitution.

Lemma 2.4 (Compositionality). $(\mathbf{e}[\mathbf{e}'/\mathbf{x}])^\circ \equiv \mathbf{e}^\circ[\mathbf{e}'^\circ/x]$

Proof. By induction on the typing derivation of $\mathbf{e}$. There is one interesting case.

Case [T-Cont] $\mathbf{e} = \mathbf{e}_1\ @\ \mathbf{B}\ (\boldsymbol{\lambda}\, \mathbf{x}' : \mathbf{A}.\ \mathbf{e}_2)$ and $\mathbf{e}^\circ = \mathit{let}\ x' = (\mathbf{e}_1^\circ\ \mathbf{A}^\circ\ \mathit{id})\ \mathit{in}\ \mathbf{e}_2^\circ$

Without loss of generality, assume $\mathbf{x} \neq \mathbf{x}'$.

It suffices to show that

$(\mathbf{e}_1[\mathbf{e}'/\mathbf{x}]\ @\ \mathbf{B}[\mathbf{e}'/\mathbf{x}]\ (\boldsymbol{\lambda}\, \mathbf{x}' : \mathbf{A}.\ \mathbf{e}_2)[\mathbf{e}'/\mathbf{x}])^\circ = (\mathit{let}\ x' = (\mathbf{e}_1^\circ\ \mathbf{A}^\circ\ \mathit{id})\ \mathit{in}\ \mathbf{e}_2^\circ)[\mathbf{e}'^\circ/x]$

$$\begin{aligned}
&(\mathbf{e}_1[\mathbf{e}'/\mathbf{x}]\ @\ \mathbf{B}[\mathbf{e}'/\mathbf{x}]\ (\boldsymbol{\lambda}\, \mathbf{x}' : \mathbf{A}.\ \mathbf{e}_2)[\mathbf{e}'/\mathbf{x}])^\circ && && (1)\\
&= (\mathbf{e}_1[\mathbf{e}'/\mathbf{x}]\ @\ \mathbf{B}[\mathbf{e}'/\mathbf{x}]\ (\boldsymbol{\lambda}\, \mathbf{x}' : \mathbf{A}[\mathbf{e}'/\mathbf{x}].\ \mathbf{e}_2[\mathbf{e}'/\mathbf{x}]))^\circ && \text{by definition of substitution} && (2)\\
&= (\mathit{let}\ x' = ((\mathbf{e}_1[\mathbf{e}'/\mathbf{x}])^\circ\ (\mathbf{A}[\mathbf{e}'/\mathbf{x}])^\circ\ \mathit{id})\ \mathit{in}\ (\mathbf{e}_2[\mathbf{e}'/\mathbf{x}])^\circ) && \text{by definition of the translation} && (3)\\
&= (\mathit{let}\ x' = (\mathbf{e}_1^\circ[\mathbf{e}'^\circ/x]\ \mathbf{A}^\circ[\mathbf{e}'^\circ/x]\ \mathit{id})\ \mathit{in}\ \mathbf{e}_2^\circ[\mathbf{e}'^\circ/x]) && \text{by the induction hypothesis} && (4)\\
&= (\mathit{let}\ x' = \mathbf{e}_1^\circ\ \mathbf{A}^\circ\ \mathit{id}\ \mathit{in}\ \mathbf{e}_2^\circ)[\mathbf{e}'^\circ/x] && \text{by definition of substitution} && (5)
\end{aligned}$$

□

The equivalence rules of extensional CC with the addition of Lemma 2.2 (Continuation Shuffling) are the same as CC^k. Therefore, to show that equivalence is preserved, it suffices to show that reduction sequences are preserved. We first show that single-step reduction is preserved, Lemma 2.5, which easily implies preservation of reduction sequences, Lemma 2.6.

Lemma 2.5 (Preservation of One-Step Reduction). *If* $\mathbf{e}_1 \triangleright \mathbf{e}_2$, *then* $\mathbf{e}_1^\circ \triangleright^\star e'$ *and* $\mathbf{e}_2^\circ \equiv e'$

Proof. By cases on the reduction step $\mathbf{e}_1 \triangleright \mathbf{e}_2$. There is one interesting case.

Case $\mathbf{e} = (\lambda\, \alpha : *.\, \mathbf{e}_1)\ @\ \mathbf{B}\ (\lambda\, \mathbf{x}' : \mathbf{A}.\, \mathbf{e}_2) \rhd_{@} (\mathbf{e}_1[\mathbf{B}/\alpha])\ (\lambda\, \mathbf{x}' : \mathbf{A}.\, \mathbf{e}_2)$

By definition $\mathbf{e}^\circ = (let\ x' = ((\lambda\, \alpha : *.\, \mathbf{e}_1)^\circ\ \mathbf{A}^\circ\ id)\ in\ \mathbf{e}_2^\circ) \rhd_{\zeta} \mathbf{e}_2^\circ[((\lambda\, \alpha : *.\, \mathbf{e}_1)^\circ\ \mathbf{A}^\circ\ id)/x']$

We must show $((\mathbf{e}_1[\mathbf{B}/\alpha])\ (\lambda\, \mathbf{x}' : \mathbf{A}.\, \mathbf{e}_2))^\circ \equiv \mathbf{e}_2^\circ[((\lambda\, \alpha : *.\, \mathbf{e}_1)^\circ\ \mathbf{A}^\circ\ id)/x']$.

$$((\mathbf{e}_1[\mathbf{B}/\alpha])\ (\lambda\, \mathbf{x}' : \mathbf{A}.\, \mathbf{e}_2))^\circ \tag{6}$$

$$\equiv ((\mathbf{e}_1^\circ[\mathbf{B}^\circ/\alpha])\ (\lambda\, x' : \mathbf{A}^\circ.\, \mathbf{e}_2^\circ)) \qquad \text{by Lemma 2.4 and definition of } ^\circ \tag{7}$$

$$\equiv (\lambda\, \alpha : *.\, \mathbf{e}_1^\circ)\ \mathbf{B}^\circ\ (\lambda\, x' : \mathbf{A}^\circ.\, \mathbf{e}_2^\circ) \qquad \text{by } [\equiv] \text{ and } \rhd_\beta \tag{8}$$

$$\equiv (\lambda\, x' : \mathbf{A}^\circ.\, \mathbf{e}_2^\circ)\ ((\lambda\, \alpha : *.\, \mathbf{e}_1^\circ)\ \mathbf{A}^\circ\ id) \qquad \text{by Lemma 2.2 (Continuation Shuffling)} \tag{9}$$

$$\equiv \mathbf{e}_2^\circ[((\lambda\, \alpha : *.\, \mathbf{e}_1^\circ)\ \mathbf{A}^\circ\ id)/x'] \qquad \text{by } [\equiv] \text{ and } \rhd_\beta \tag{10}$$

$$\equiv \mathbf{e}_2^\circ[((\lambda\, \alpha : *.\, \mathbf{e}_1)^\circ\ \mathbf{A}^\circ\ id)/x'] \qquad \text{by Lemma 2.4} \tag{11}$$

□

Lemma 2.6 (Preservation of Reduction Sequences). *If* $\mathbf{e}_1 \rhd^\star \mathbf{e}_2$, *then* $\mathbf{e}_1^\circ \rhd^\star e'$ *and* $\mathbf{e}_2^\circ \equiv e'$.

Proof. By induction on the length n of the reduction sequence $\mathbf{e}_1 \rhd^n \mathbf{e}_2$. Follows from Lemma 2.5 (Preservation of One-Step Reduction). □

Lemma 2.7 (Equivalence Preservation). *If* $\mathbf{e}_1 \equiv \mathbf{e}_2$, *then* $\mathbf{e}_1^\circ \equiv \mathbf{e}_2^\circ$

Finally, we can show type preservation, which completes our proof of consistency. Since the translation is homomorphic on all typing rules except [T-Cont], there is only one interesting case in the proof of Lemma 2.8. We must show that [Un-Cont] is type preserving. Note that the case for [Conv] appeals to Lemma 2.7.

Lemma 2.8 (Type Preservation). *If* $\Gamma \vdash \mathbf{e} : \mathbf{A}$ *then* $\Gamma^\circ \vdash \mathbf{e}^\circ : \mathbf{A}^\circ$

Proof. By induction on the derivation $\Gamma \vdash \mathbf{e} : \mathbf{A}$. There is one interesting case.

Case [T-Cont]

We have the following.

$$\frac{\Gamma \vdash \mathbf{e}_1 : \Pi\, \alpha : *.\, (\mathbf{B} \to \alpha) \to \alpha \qquad \Gamma \vdash \mathbf{A} : * \qquad \Gamma, \mathbf{x}' = \mathbf{e}_1\ \mathbf{B}\ \mathbf{id} \vdash \mathbf{e}_2 : \mathbf{A}}{\Gamma \vdash \mathbf{e}_1\ @\ \mathbf{A}\ (\lambda\, \mathbf{x}' : \mathbf{B}.\, \mathbf{e}_2) : \mathbf{A}}$$

We must show $\Gamma^\circ \vdash let\ x' = (\mathbf{e}_1^\circ\ \mathbf{B}^\circ\ id)\ in\ \mathbf{e}_2^\circ : \mathbf{A}^\circ$.

By [Let], it suffices to show

- $\Gamma^\circ \vdash (\mathbf{e}_1^\circ\ \mathbf{B}^\circ\ id) : \mathbf{B}^\circ$, which follows easily by the induction hypothesis applied to the premises of [T-Cont].
- $\Gamma^\circ, x' = (\mathbf{e}_1^\circ\ \mathbf{B}^\circ\ id) : \mathbf{B}^\circ \vdash \mathbf{e}_2^\circ : \mathbf{A}^\circ$, which follows immediately by the induction hypothesis. □

Theorem 2.9 (Consistency of CC^k). *There does not exist a closed term* $\mathbf{e}$ *such that* $\cdot \vdash \mathbf{e} : \mathbf{False}$.

3 CALL-BY-NAME CPS TRANSLATION OF CC

We now present our call-by-name CPS translation (CPS^n) of CC. The main differences between our translation and the one by Barthe and Uustalu [2002] are that we use a locally polymorphic answer type instead of a fixed answer type, which enables our type-preservation proof of snd e, and that we use a domain-full target language, which supports decidable type-checking.

We need a computation translation and a value translation on types. But in addition to types, we will need to translate universes, kinds, and terms as well. All of our translations are defined by induction on the typing derivations. This is important when translating to a domain-full target language, since the domain annotations we generate come from the type of the term we are translating. However, we find it useful to abbreviate these with $t^\div$ (for computation) and t^+ (for value translation). Below we give abbreviations for all of the translation judgments we define for our CPS translation. Note that anywhere we use this notation, we require the typing derivation as an implicit parameter.

$$A^\div \stackrel{\text{def}}{=} \mathbf{A} \text{ where } \Gamma \vdash A : * \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad e^\div \stackrel{\text{def}}{=} \mathbf{e} \text{ where } \Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{e}$$

$$U^+ \stackrel{\text{def}}{=} \mathbf{U} \text{ where } \Gamma \vdash U \rightsquigarrow^n_U \mathbf{U} \qquad \kappa^+ \stackrel{\text{def}}{=} \boldsymbol{\kappa} \text{ where } \Gamma \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad A^+ \stackrel{\text{def}}{=} \mathbf{A} \text{ where } \Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A}$$

The CPS^n translations on universes, kinds, and types are defined in Figure 10. We define the translation for kinds CPS$^n_\kappa$ and universe CPS^n_U, which we abbreviate with $^+$. There is no separate computation translation for kinds or universes. We only have separate computation and value translations for *types* since we only internalize the concept of evaluation at the *term*-level, and *types* describe term-level computations and term-level values. Recall that this is the call-by-name translation, so function arguments, even type-level functions, are still computations. Note, therefore, that the rule [CPS$^n_\kappa$-ProdA] uses the computation translation on the domain annotation A of $\Pi\, x : A.\, \kappa$—*i.e.*, the kind describing a type-level function that abstracts over a term of type A.

For types, we define a value translation CPS^n_A and a computation translation CPS$^n_{A^\div}$. Most rules are straightforward. We translate type-level variables α in-place in rule [CPS^n_A-Var]. Again, since this is the CBN translation, we use the computation translation on domain annotations. The rule [CPS^n_A-Constr] for the value translation of type-level functions that abstract over a term, $\lambda\, x : A.\, B$, translates the domain annotation A using the computation translation. The rule for the value translation of a function type, [CPS^n_A-Prod], translates the domain annotation A using the computation translation. This means that a function is a value when it accepts a computation as an argument. The rule [CPS^n_A-Sigma] produces the value translation of a pair type by translating both components of a pair using the computation translation. This means we consider a pair a value when it contains computations as components. Note that since our translation is defined on typing derivations, we have an explicit translation of the conversion rule [CPS^n_A-Conv].

There is only one rule for the computation translation of a type, [CPS$^n_{A^\div}$-Comp], which is the polymorphic answer type translation. Notice that [CPS$^n_{A^\div}$-Comp] is defined only for types of kind $*$, since only types of kind $*$ have inhabitants. For example, we cannot apply [CPS$^n_{A^\div}$-Comp] to type-level function since no term inhabits a type-level function.

The CPS^n translation on terms is defined in Figure 11 and Figure 12. Intuitively, we translate each term e of type A to $\mathbf{e}$ of type $\mathbf{\Pi\, \alpha : *.\, (A \to \alpha) \to \alpha}$, where $\mathbf{A}$ is the *value translation* of A. This type represents a computation that, when given a continuation $\mathbf{k}$ that expects a value of type $\mathbf{A}$, promises to call $\mathbf{k}$ with a value of type $\mathbf{A}$. Since we have only two value forms in the call-by-name translation, we do not explicitly define a separate value translation, but inline that translation. Note that the value cases, [CPS^n_e-Fun] and [CPS^n_e-Pair], feature the same pattern: produce a computation $\boldsymbol{\lambda\, \alpha.\, \lambda\, \mathbf{k}.\, \mathbf{k}\ \mathbf{v}}$ that expects a continuation and then immediately calls that continuation on the value $\mathbf{v}$. In the case of [CPS^n_e-Fun], the value $\mathbf{v}$ is the function $\boldsymbol{\lambda\, \mathbf{x} : \mathbf{A}.\, \mathbf{e}}$ produced by translating the source function $\lambda\, x : A.\, e$ using the computation type translation from $A \rightsquigarrow^n_{A^\div} \mathbf{A}$ and the computation term translation $e \rightsquigarrow^n_e \mathbf{e}$. In the case of [CPS$^n_e$-Pair], the value we produce $\langle \mathbf{e}_1, \mathbf{e}_2 \rangle$ contains computations, not values.

The rest of the translation rules are for computations. Notice that while all terms produced by the term translation have a computation type, all continuations take a value type. Since this is a CBN translation, we consider variables as computations in [CPS^n_e-Var]. We translate term variables as an η-expansion of a CPS'd computation. We must η-expand the variable case to guarantee CBN evaluation order, as we discuss shortly. In [CPS^n_e-App] we encode the CBN evaluation order for function application $e\ e'$ in the usual way. We translate the computations $e \rightsquigarrow^n_e \mathbf{e}$

and $e' \rightsquigarrow^n_e \mathbf{e}'$. First we evaluate $\mathbf{e}$ to a value $\mathbf{f}$, then apply $\mathbf{f}$ to the *computation* $\mathbf{e}'$. The application $\mathbf{f}\ \mathbf{e}'$ is itself a computation, which we call with the continuation $\mathbf{k}$.

Notice that only the translation rules [CPS^n_e-Fst] and [CPS^n_e-Snd] use the new @ form. To type check the translation of snd e produced by [CPS^n_e-Snd], we require the rule [T-Cont] when type checking the continuation that performs the second projection. While type checking the continuation, we know that the value $\mathbf{y}$ that the continuation receives is equivalent to $e^{\div}\ \boldsymbol{\alpha}\ \mathbf{id}$. Now, the reason we must use the @ syntax in the in the translation [CPS^n_e-Fst] is so that we can apply the [≡-Cont] rule to resolve the equivalence of the two *first projections* in the type of the second projection. That is type preservation fails because we must show equivalence between $(\mathsf{fst}\ e)^{\div}$ and $\mathbf{fst}\ \mathbf{y}$. Since these are the only two cases that require our new rules, these are the only cases where we use the @ form in our translation; all other translation rules use standard application. In Section 4, we will see that the CBV translation must use the @ form much more frequently since, intuitively, our new equivalence rule recovers a notion of "value" in our CPS'd language, and in call-by-*value* types can only depend on values.

Our CPS translation encodes the CBN evaluation order explicitly so that the evaluation order of compiled terms is independent of the target language's evaluation order. This property is not immediately obvious since the [CPS^n_e-Let] rule binds a variable $\mathbf{x}$ to an expression $\mathbf{e}$, making it seem like there are two possible evaluation orders: either evaluate $\mathbf{e}$ first, or substitute $\mathbf{e}$ for $\mathbf{x}$ first. Note, however, that our CBN translation always produces a $\boldsymbol{\lambda}$ term—even in the variable case since [CPS^n_e-Var] employs η-expansion as noted above. Therefore, in the [CPS^n_e-Let] rule $\mathbf{e}$ will always be a value, which means it doesn't evaluate in either CBN or CBV. Therefore, there is no ambiguity in how to evaluate the translation of let.

The translation rule [CPS^n_e-Conv] is deceptively simple. We could equivalently write this translation as follows, which makes its subtlety apparent.

$$\frac{\Gamma \vdash e : B \qquad \Gamma \vdash A \equiv B \qquad \Gamma \vdash e : B \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash B : \kappa \rightsquigarrow^n_A \mathbf{B}}{\Gamma \vdash e : A \rightsquigarrow^n_e \boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\,\mathbf{k} : \mathbf{A} \to \boldsymbol{\alpha}.\, \mathbf{e}\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{x} : \mathbf{B}.\, \mathbf{k}\ \mathbf{x})}\ [\text{CPS}^n_e\text{-Conv}]$$

Notice now that while the continuation $\mathbf{k}$ expects a term of type $\mathbf{A}$, we call $\mathbf{k}$ with a term of type $\mathbf{B}$. Intuitively, this should be fine since $\mathbf{A}$ and $\mathbf{B}$ should be equivalent, but formally this introduces a subtlety in the staging of our proof of type preservation, which we discuss next in Section 3.1.

We lift the translations to environments in the usual way. Since this is the CBN translation, we recur over the environment applying the *computation* translation.

3.1 Proof of Type Preservation for CPS^n

In a dependent type system, type preservation requires *coherence*, which essentially tells us that the translation preserves definitional equivalence. Since equivalence is defined by reduction, we first have to show that reduction sequences are preserved. Since reduction relies on substitution, we first must show *compositionality*, *i.e.*, that the translation commutes with substitution.

However, there is a problem with the proof architecture for CPS. Typed CPS for a *domain-full* target language inserts the type of every term into the output as a type annotation on the continuation. For example, $e : A$ is compiled to $\boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\,\mathbf{k} : A^+ \to \boldsymbol{\alpha}.\, (\ldots)$. Therefore, the translation is defined on typing derivations, not on syntax. This introduces a problem in the case of the translation of the typing rule [Conv]. As alluded to above when describing [CPS^n_e-Conv], in order to preserve typing, we must first show coherence, *i.e.*, that we preserve equivalence. Working with the second definition we gave for the [CPS^n_e-Conv] rule, we need to show that the following term is well-typed.

$$\boldsymbol{\lambda}\,\mathbf{k} : A^+ \to \boldsymbol{\alpha}.\, e^{\div}\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{x} : B^+.\, \mathbf{k}\ \mathbf{x})$$

Note that this term seems to only make sense when $A^+ \equiv B^+$. While we have $A \equiv B$ from the source typing derivation, we don't know that $A^+ \equiv B^+$ unless we have coherence. But if equivalence is only defined on well-typed terms, as is the case in some dependently typed languages, we must first prove type preservation to know that A^+ and B^+ are well typed before we can prove coherence. So we have a circularity: type preservation requires coherence, but coherence requires type preservation.

A similar problem arises in other dependent typing rules, like the translation of application. In the case of application, to show type preservation we must show compositionality, *i.e.*, that the translation commutes with substitution

up to equivalence. Again we have a circularity: we need type preservation to prove compositionality, but to prove compositionality we need type preservation.

Barthe et al. [1999] explain this in detail, and their solution is to use a domain-free target language. This avoids the circularity because when there are no type annotations to generate, the translation can be defined on the syntax instead of typing derivations.

We solve the problem by using an untyped equivalence, which is based on the equivalence for the Calculus of Inductive Constructions from the Coq reference manual [The Coq Development Team 2017, Chapter 4]. Since the equivalence is untyped, we can show equivalence between terms with different domain annotations as long as their behavior is equivalent. This allows us to stage the proof: we can prove compositionality before coherence, and prove coherence before type preservation. While it seems surprising that we can prove equivalence between possibly ill-typed terms, recall that in the type system, we only appeal to the equivalence *after* checking that the terms we wish to prove equivalent are well typed. We can think of this equivalence as proving a stronger equivalence than we provide for well-typed terms, allowing us to prove a stronger form of coherence: in addition to preserving all well-typed equivalences, we also preserve certain equivalences that are valid according to their dynamic behavior, but conservatively ruled out by the strong type system. From this version of coherence, we are able to prove type preservation.

The proofs in this section are staged as follows. First we show compositionality (Lemma 3.1), since reduction is defined in terms of substitution. Then we show that the translation preserves reduction sequences (Lemma 3.2 and Lemma 3.3), which allows us to show coherence (Lemma 3.4). Using compositionality and coherence, we prove that the translation is type preserving (Lemma 3.5).

We now show Lemma 3.1, which states that the CPS^n translation commutes with substitution. The formal statement of the lemma is somewhat complicated since we have the cross product of four syntactic categories and two translations. However, the intuition is simple: first substituting and then translating is equivalent to translating and then substituting.

This lemma is critical to our proofs. Since reduction is essentially defined by substitution, this lemma does most of the work in showing that the translation preserves reduction. However, it is also necessary in type preservation when showing that a dependent type is preserved. A dependent type, such as $B[e'/x]$ produced by the rule [App], occurs when we perform substitution into a type. We want to show, for example, that if $e : B[e'/x]$ then the translation of e has the type translation $(B[e'/x])^\div$. Since our translation is compositional, *i.e.*, commutes with substitution, we know how to translate $B[e'/x]$ by simply translating B and e'.

Lemma 3.1 (CPS^n Compositionality).

(1) $(\kappa[A/\alpha])^+ \equiv \kappa^+[A^+/\boldsymbol{\alpha}]$
(2) $(\kappa[e/x])^+ \equiv \kappa^+[e^\div/\mathbf{x}]$
(3) $(A[B/\alpha])^+ \equiv A^+[B^+/\boldsymbol{\alpha}]$
(4) $(A[e/x])^+ \equiv A^+[e^\div/\mathbf{x}]$
(5) $(A[B/\alpha])^\div \equiv A^\div[B^+/\boldsymbol{\alpha}]$
(6) $(A[e/x])^\div \equiv A^\div[e^\div/\mathbf{x}]$
(7) $(e[A/\alpha])^\div \equiv e^\div[A^+/\boldsymbol{\alpha}]$
(8) $(e[e'/x])^\div \equiv e^\div[e'^\div/\mathbf{x}]$

Proof. In the PTS syntax, we represent source expressions as $t[t'/x]$. The proof is by induction on the typing derivations for t. Note that our $^\div$ and $^+$ notation implicitly require the typing derivations as premises. The proof is completely straightforward. Part 6 follows immediately by part 3. Part 7 follows immediately by part 4. We give representative cases for the other parts.

Case [Ax-$*$], parts 1 and 2. Trivial, since no free variables appear in $*$.

Case [Prod-$*$] and [Prod-$\Box$]: $t = \Pi\, x : B.\, \kappa'$

Sub-Case Part 1. We must show that $((\Pi\, x : B.\, \kappa')[A/\alpha])^+ = (\Pi\, x : B.\, \kappa')^+[A^+/\boldsymbol{\alpha}]$.

$$
\begin{aligned}
& ((\Pi\, x : B.\, \kappa')[A/\alpha])^+ && && (12)\\
&= (\Pi\, x : B[A/\alpha].\, \kappa'[A/\alpha])^+ && \text{by definition of substitution} && (13)\\
&= \boldsymbol{\Pi}\, \mathbf{x}' : (B[A/\alpha])^\div.\, (\kappa'[A/\alpha])^+ && \text{by definition of the translation} && (14)\\
&= \boldsymbol{\Pi}\, \mathbf{x}' : B^\div[A^+/\alpha].\, \kappa'^+[A^+/\alpha] && \text{by parts 1 and 5 of the induction hypothesis} && (15)\\
&= (\boldsymbol{\Pi}\, \mathbf{x}' : B^\div.\, \kappa'^+)[A^+/\boldsymbol{\alpha}] && \text{by definition of substitution} && (16)\\
&= (\Pi\, x' : B.\, \kappa')^+[A^+/\boldsymbol{\alpha}] && \text{by definition of the translation} && (17)
\end{aligned}
$$

Sub-Case Part 2. Similar to the previous subcase.
Case [Var]
Sub-Case Part 3 $t = \alpha'$. Part 4 is trivial since x is not free in α.
We must show that $(\alpha'[A/\alpha])^+ = \boldsymbol{\alpha}'[A^+/\boldsymbol{\alpha}]$.
Sub-Sub-Case $\alpha = \alpha'$. It suffices to show that $A^+ = A^+$, which is trivial.
Sub-Sub-Case $\alpha \neq \alpha'$. $\alpha^+ = \boldsymbol{\alpha}$ by definition.
Sub-Case x, parts 7 and 8. Similar to previous case.
Case [App]
Sub-Case $t = e_1\ e_2$, Part 7
We must show $((e_1\ e_2)[A'/\alpha'])^\div = (e_1\ e_2)^\div[A'^+/\boldsymbol{\alpha}']$.

$$
\begin{aligned}
&((e_1\ e_2)[A'/\alpha'])^\div && && (18)\\
&= (e_1[A/\alpha']\ e_2[A'/\alpha'])^\div && \text{by definition of substitution} && (19)\\
&= \boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\,\mathbf{k} : ((B[A'/\alpha'])^+[(e_2[A'/\alpha'])^\div/\mathbf{x}]) \to \boldsymbol{\alpha}.\\
&\quad (e_1[A'/\alpha'])^\div\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{f} : \boldsymbol{\Pi}\,\mathbf{x} : (A[A'/\alpha'])^\div.\ (B[A'/\alpha'])^\div.\\
&\qquad (\mathbf{f}\ (e_2[A'/\alpha'])^\div)\ \boldsymbol{\alpha}\ \mathbf{k}) && \text{by def. of translation} && (20)\\
&= \boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\,\mathbf{k} : (B^+[A'^+/\boldsymbol{\alpha}'][e_2^\div[A'^+/\boldsymbol{\alpha}']/\mathbf{x}]) \to \boldsymbol{\alpha}.\\
&\quad e_1^\div[A'^+/\boldsymbol{\alpha}']\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{f} : \boldsymbol{\Pi}\,\mathbf{x} : A^\div[A'^+/\boldsymbol{\alpha}'].\ B^\div[A'^+/\boldsymbol{\alpha}'].\\
&\qquad (\mathbf{f}\ e_2^\div[A'^+/\boldsymbol{\alpha}'])\ \boldsymbol{\alpha}\ \mathbf{k}) && \text{by part 3, 5, and 7 of IH} && (21)\\
&= (\boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\,\mathbf{k} : (B^+[e_2^\div/\mathbf{x}]) \to \boldsymbol{\alpha}.\\
&\quad e_1^\div\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{f} : \boldsymbol{\Pi}\,\mathbf{x} : A^\div.\ B^\div.\\
&\qquad (\mathbf{f}\ e_2^\div)\ \boldsymbol{\alpha}\ \mathbf{k}))[A'^+/\boldsymbol{\alpha}'] && \text{by definition of substitution} && (22)\\
&= (e_1\ e_2)^\div[A'^+/\boldsymbol{\alpha}'] && \text{by definition of translation} && (23)
\end{aligned}
$$

Sub-Case Part 8
We must show $((e_1\ e_2)[e/x])^\div = (e_1\ e_2)^\div[e^\div/\mathbf{x}]$.
Similar to previous case.
Case [Conv]. The proof is trivial, now that we have staged the proof appropriately. We give part 8 as an example.

$$\frac{\Gamma \vdash e : B \qquad \Gamma \vdash A : U \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash e : A}$$

We must show that $(e[e'/x])^\div \equiv e^\div[e'^\div/\mathbf{x}]$ (at type A). Note that by part 8 of the induction hypothesis, we know that $(e[e'/x])^\div \equiv e^\div[e'^\div/\mathbf{x}]$ (at the smaller derivation for type B). But recall that our equivalence cares nothing for types, so the proof is complete. □

We next prove that the translation preserves reduction sequences, Lemma 3.2 and Lemma 3.3. Note that kinds do not take steps in the one step reduction, but can in the $\triangleright^\star$ relation since it reduces under all contexts. As mentioned earlier, this is necessary to show equivalence is preserved, since equivalence is defined in terms of reduction. Note that we can only preserve reduction up to equivalence, in particular η-equivalence. This intuition for this is simple. The computation translation of a term $e'^\div$ always produce a lambda expression $\boldsymbol{\lambda}\,\boldsymbol{\alpha}.\,\boldsymbol{\lambda}\,\mathbf{k}.\,\mathbf{e}''$. However, when $e^\div \triangleright^\star \mathbf{e}'$, we do not know that the term $\mathbf{e}'$ is equal to a lambda expression, although it is η equivalent to one.

Lemma 3.2 (CPS^n Preserves One-Step Reduction).

- *If* $\Gamma \vdash e : A$ *and* $e \triangleright e'$ *then* $e^\div \triangleright^\star \mathbf{e}'$ *and* $\mathbf{e}' \equiv e'^\div$
- *If* $\Gamma \vdash A : \kappa$ *and* $A \triangleright A'$ *then* $A^+ \triangleright^\star \mathbf{A}'$ *and* $\mathbf{A}' \equiv A'^+$
- *If* $\Gamma \vdash A : *$ *and* $A \triangleright A'$ *then* $A^\div \triangleright^\star \mathbf{A}'$ *and* $\mathbf{A}' \equiv A'^\div$

Proof. The proof is straightforward by cases on the $\triangleright$ relation. We give some representative cases.

Case $x \triangleright_\delta e'$ where $x = e' : A' \in \Gamma$
It suffices to show that $x^\div \triangleright_\delta e'^\div$ where $x^\div = e'^\div : A'^\div \in \Gamma^+$, which follows by [CPS$^n_e$-Def].
Case $(\lambda\, x : \_.\ e_1)\ e_2 \triangleright_\beta e_1[e_2/x]$

We must show that $((\lambda\, x : \_.\ e_1)\ e_2)^\div \rhd^\star \mathbf{e'}$ and $\mathbf{e'} \equiv (e_1[e_2/x])^\div$.

$$((\lambda\, x : \_.\ e_1)\ e_2) \tag{24}$$
$$= \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\, \mathbf{k} : \_.\ (\boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\, \mathbf{k} : \_.\ \mathbf{k}\ (\boldsymbol{\lambda}\, \mathbf{x} : \_.\ e_1^\div))\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \_.\ (\mathbf{f}\ e_2^\div)\ \boldsymbol{\alpha}\ \mathbf{k}) \quad \text{by definition of translation} \tag{25}$$
$$\rhd^\star \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\, \mathbf{k} : \_.\ (((\boldsymbol{\lambda}\, \mathbf{x} : \_.\ e_1^\div)\ e_2^\div)\ \boldsymbol{\alpha}\ \mathbf{k}) \quad \text{by } \rhd_\beta \tag{26}$$
$$\rhd^\star \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\, \mathbf{k} : \_.\ (e_1^\div[e_2^\div/\mathbf{x}])\ \boldsymbol{\alpha}\ \mathbf{k} \tag{27}$$
$$\equiv e_1^\div[e_2^\div/\mathbf{x}] \quad \text{by } [\equiv\text{-}\eta] \tag{28}$$
$$= (e_1[e_2/x])^\div \quad \text{by Lemma 3.1} \tag{29}$$

Case $\mathsf{snd}\ \langle e_1, e_2\rangle \rhd_{\pi_2} e_2$

We must show that $(\mathsf{snd}\ \langle e_1, e_2\rangle)^\div \rhd^\star \mathbf{e'}$ and $\mathbf{e'} \equiv e_2^\div$.

$$(\mathsf{snd}\ \langle e_1, e_2\rangle)^\div \tag{30}$$
$$= \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\, \mathbf{k} : \_.\ (\boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\, \mathbf{k} : \_.\ \mathbf{k}\ \langle e_1^\div, e_2^\div\rangle)\ @\ \boldsymbol{\alpha}\ \boldsymbol{\lambda}\, \mathbf{y} : \_.\ \mathbf{let\ z = snd\ y\ in\ z}\ \boldsymbol{\alpha}\ \mathbf{k} \tag{31}$$
$$\rhd^\star \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\, \mathbf{k} : \_.\ \mathbf{let\ z = snd}\ \langle e_1^\div, e_2^\div\rangle\ \mathbf{in\ z}\ \boldsymbol{\alpha}\ \mathbf{k} \tag{32}$$
$$\rhd^\star \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\, \mathbf{k} : \_.\ e_2^\div\ \boldsymbol{\alpha}\ \mathbf{k} \tag{33}$$
$$\equiv e_2^\div \quad \text{by } [\equiv\text{-}\eta] \tag{34}$$

□

Lemma 3.3 (CPS^n Preserves Reduction Sequences).

- *If* $\Gamma \vdash e : A$ *and* $e \rhd^\star e'$ *then* $e^\div \rhd^\star \mathbf{e'}$ *and* $\mathbf{e'} \equiv e'^\div$
- *If* $\Gamma \vdash A : \kappa$ *and* $A \rhd^\star A'$ *then* $A^+ \rhd^\star \mathbf{A'}$ *and* $\mathbf{A'} \equiv A'^+$
- *If* $\Gamma \vdash A : *$ *and* $A \rhd^\star A'$ *then* $A^\div \rhd^\star \mathbf{A'}$ *and* $\mathbf{A'} \equiv A'^\div$
- *If* $\Gamma \vdash \kappa : U$ *and* $\kappa \rhd^\star \kappa'$ *then* $\kappa^+ \rhd^\star \boldsymbol{\kappa'}$ *and* $\boldsymbol{\kappa'} \equiv \kappa'^+$

Proof. The proof is straightforward by induction on the length of the reduction sequence. The base case is trivial and the inductive case follows by Lemma 3.2 and the inductive hypothesis. □

Lemma 3.4 (CPS^n Coherence).

- *If* $e \equiv e'$ *then* $e^\div \equiv e'^\div$
- *If* $A \equiv A'$ *then* $A^+ \equiv A'^+$
- *If* $A \equiv A'$ *then* $A^\div \equiv A'^\div$
- *If* $\kappa \equiv \kappa'$ *then* $\kappa^+ \equiv \kappa'^+$

Proof. The proof is by induction on the derivation of $e \equiv e'$. The base case follows by Lemma 3.3, and the cases of η-equivalence follow from Lemma 3.3, the induction hypothesis, and the fact that we have the same η-equivalence rules in the CC^k. □

We first prove type preservation, Lemma 3.5, using the explicit syntax on which we defined CPS^n. This proof is our central contribution to the call-by-name translation. In this lemma, proving that the translation of snd e preserves typing requires both the new typing rule [T-Cont] and the equivalence rule [≡-Cont]. The rest of the proof is straightforward.

Lemma 3.5 (CPS^n is Type Preserving (Explicit Syntax)).

(1) *If* $\vdash \Gamma$ *then* $\vdash \Gamma^+$
(2) *If* $\Gamma \vdash e : A$ *then* $\Gamma^+ \vdash e^\div : A^\div$
(3) *If* $\Gamma \vdash A : \kappa$ *then* $\Gamma^+ \vdash A^+ : \kappa^+$
(4) *If* $\Gamma \vdash A : *$ *then* $\Gamma^+ \vdash A^\div : *^+$
(5) *If* $\Gamma \vdash \kappa : U$ *then* $\Gamma^+ \vdash \kappa^+ : U^+$

Proof. All cases are proven simultaneously by mutual induction on the type derivation and well-formedness derivation. Part 4 follows easily by part 3 in every case, so we elide its proof. Most cases follow easily from the induction hypotheses. We give proofs of the cases where reasoning is not simple, including those that require subtle reasoning about the universe hierarchy and the cases related to pairs.

Case Part 5, [Ax-*]: $\Gamma \vdash * : \Box$

We must show that $\Gamma^+ \vdash *^+ : \Box^+$, which follows by part 1 of the induction hypothesis and by definition of the translation, since $*^+ = *$ and $\Box^+ = \Box$.

Case [Prod-*]: $\Gamma \vdash \Pi\, x : e_1.\, e_2 : \kappa$

There are two sub-cases: either e_2 is a type, or a kind.

Sub-Case Part 3, $e_2 = B$, *i.e.*, is a type.

There are two sub-cases: either e_1 is a type or a kind.

Sub-Sub-Case $e_1 = A$, *i.e.*, is a type.

We have $\Gamma \vdash \Pi\, x : A.\, B : *$.

We must show that $\Gamma^+ \vdash (\Pi\, x : A.\, B)^+ : *^+$

By definition, must show $\Gamma^+ \vdash \mathbf{\Pi}\, \mathbf{x} : A^\div.\, B^\div : *$, which follows by the part 4 of the induction hypothesis applied to A and B.

Sub-Sub-Case $e_1 = \kappa$, *i.e.*, is a kind.

We have $\Gamma \vdash \Pi\, \alpha : \kappa.\, B : *$.

We must show that $\Gamma^+ \vdash (\Pi\, \alpha : \kappa.\, B)^+ : *^+$

By definition, must show $\Gamma^+ \vdash \mathbf{\Pi}\, \boldsymbol{\alpha} : \kappa^+.\, B^\div : *$, which follows by the part 4 of the induction hypothesis applied to B, and part 5 of the induction hypothesis applied to κ.

Sub-Case Part 5, $e_2 = \kappa'$, *i.e.*, is a kind.

There are two sub-cases: either e_1 is a type or a kind.

Sub-Sub-Case $e_1 = A$, *i.e.*, is a type.

We have $\Gamma \vdash \Pi\, x : A.\, \kappa' : U$.

We must show that $\Gamma^+ \vdash (\Pi\, x : A.\, \kappa')^+ : *^+$

By definition, must show $\Gamma^+ \vdash \mathbf{\Pi}\, \mathbf{x} : A^\div.\, \kappa'^+ : *$, which follows by the part 4 of the induction hypothesis applied to A and part 5 of the induction hypothesis applied to κ.

Sub-Sub-Case $e_1 = \kappa$, *i.e.*, is a kind.

We have $\Gamma \vdash \Pi\, \alpha : \kappa.\, \kappa' : *$.

We must show that $\Gamma^+ \vdash (\Pi\, \alpha : \kappa.\, \kappa')^+ : *^+$

By definition, must show $\Gamma^+ \vdash \mathbf{\Pi}\, \boldsymbol{\alpha} : \kappa^+.\, \kappa'^+ : *$, which follows by the part 5 of the induction hypothesis applied to κ and κ'^+.

Case [Prod-$\Box$] Similar to the previous case, except with $*$ replaced by $\Box$; there are two fewer cases since this must be a kind.

Case [Sigma]: $\Gamma \vdash \Sigma\, x : A.\, B : *$

We must show that $\Gamma^+ \vdash \mathbf{\Sigma}\, \mathbf{x} : A^\div.\, B^\div : *$, which follows easily by the part 4 of the induction hypothesis applied to A and B.

Case [Pair] $\Gamma \vdash \langle e_1, e_2 \rangle : \Sigma\, x : A.\, B$

By definition of the translation, we must show that

$\Gamma^+ \vdash \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : (\mathbf{\Sigma}\, \mathbf{x} : A^\div.\, B^\div \to \boldsymbol{\alpha}).$

$\quad \mathbf{k}\, \langle e_1^\div, e_2^\div \rangle\, \mathbf{as}\, \mathbf{\Sigma}\, \mathbf{x} : A^\div.\, B^\div : \mathbf{\Pi}\, \boldsymbol{\alpha} : *.\, (\mathbf{\Sigma}\, \mathbf{x} : A^\div.\, B^\div \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$

It suffices to show that $\Gamma^+ \vdash \langle e_1^\div, e_2^\div \rangle\, \mathbf{as}\, \mathbf{\Sigma}\, \mathbf{x} : A^\div.\, B^\div : \mathbf{\Sigma}\, \mathbf{x} : A^\div.\, B^\div$, which follows easily by part 2 of the induction hypothesis applied to $\Gamma \vdash e_1 : A$ and $\Gamma \vdash e_2 : B[e_1/x]$.

Case [Snd] $\Gamma \vdash \mathsf{snd}\, e : B[\mathsf{fst}\, e/x]$

We must show that $\boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : B^+[(\mathsf{fst}\, e)^\div/\mathbf{x}] \to \boldsymbol{\alpha}.$

$\quad e^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{y} : \mathbf{\Sigma}\, \mathbf{x} : A^\div.\, B^\div.\, \mathbf{let}\, \mathbf{z} = \mathbf{snd}\, \mathbf{y}\, \mathbf{in}\, \mathbf{z}\ \boldsymbol{\alpha}\ \mathbf{k})$

has type $(B[\mathsf{fst}\, e/x])^\div$.

By part 6 of Lemma 3.1, and definition of the translation, this type is equivalent to

$\mathbf{\Pi}\, \boldsymbol{\alpha} : *.\, (B^+[(\mathsf{fst}\, e)^\div/\mathbf{x}] \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$

By [Lam], it suffices to show that

$$\Gamma^+, \boldsymbol{\alpha} : *, \mathbf{k} : B^+[(\mathsf{fst}\, e)^\div/\mathbf{x}] \to \boldsymbol{\alpha} \vdash e^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{y} : \mathbf{\Sigma}\, \mathbf{x} : A^\div.\, B^\div.\, \mathbf{let}\, \mathbf{z} = \mathbf{snd}\, \mathbf{y}\, \mathbf{in}\, \mathbf{z}\ \boldsymbol{\alpha}\ \mathbf{k}) : \boldsymbol{\alpha}$$

This is the key difficulty in the proof. The term $\mathbf{z}\ \boldsymbol{\alpha}$ has type $(B^+[\mathbf{fst}\, \mathbf{y}/\mathbf{x}] \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$ while the term $\mathbf{k}$ has type $B^+[(\mathsf{fst}\, e)^\div/\mathbf{x}] \to \boldsymbol{\alpha}$. To show that $\mathbf{z}\ \boldsymbol{\alpha}\ \mathbf{k}$ is well-typed, we must show that $(\mathsf{fst}\, e)^\div \equiv \mathbf{fst}\, \mathbf{y}$. We proceed by our new typing rule [T-Cont], which will help us prove this.

First, note that $e^\div\ (\Sigma\,\mathbf{x} : A^\div.\ B^\div)\ \mathbf{id}$ is well-typed. By part 4 of the induction hypothesis we know that $\Gamma^+ \vdash A^\div : *$ and $\Gamma^+, \mathbf{x} : A^\div \vdash B^\div : *$. By part 2 of the induction hypothesis applied to $\Gamma \vdash e : \Sigma\, x : A.\ B$, we know $\Gamma^+ \vdash e^\div : \Pi\, \boldsymbol{\alpha} : *.\ (\Sigma\,\mathbf{x} : A^\div.\ B^\div \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$.

Now, by [T-Cont], it suffices to show that

$$\Gamma^+, \boldsymbol{\alpha} : *, \mathbf{k} : B^+[(\mathbf{fst}\ e)^\div/\mathbf{x}] \to \boldsymbol{\alpha}, \mathbf{y} = e^\div\ \Sigma\,\mathbf{x} : A^\div.\ B^\div \mathbf{id} \vdash \mathbf{let\ z = snd\ y\ in\ z}\ \boldsymbol{\alpha}\ \mathbf{k} : \boldsymbol{\alpha}$$

Note that we now have the definitional equivalence $\mathbf{y} = e^\div\ (\Sigma\,\mathbf{x} : A^\div.\ B^\div)\ \mathbf{id}$. By [Let] it suffices to show

$$\Gamma^+, \boldsymbol{\alpha} : *, \mathbf{k} : B^+[(\mathbf{fst}\ e)^\div/\mathbf{x}] \to \boldsymbol{\alpha}, \mathbf{y} = e^\div\ \Sigma\,\mathbf{x} : A^\div.\ B^\div\ \mathbf{id}, \mathbf{z} = \mathbf{snd\ y} : B^\div[\mathbf{fst\ y}/\mathbf{x}] \vdash \mathbf{z}\ \boldsymbol{\alpha}\ \mathbf{k} : \boldsymbol{\alpha}$$

Note that

$$\begin{aligned}
&\mathbf{z} : B^\div[\mathbf{fst\ y}/\mathbf{x}] && && (35)\\
&= \Pi\,\boldsymbol{\alpha} : *.\ (B^+[\mathbf{fst\ y}/\mathbf{x}] \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha} && \text{by definition of } B^\div && (36)\\
&\equiv \Pi\,\boldsymbol{\alpha} : *.\ (B^+[\mathbf{fst}\,(e^\div\ _\ \mathbf{id})/\mathbf{x}] \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha} && \text{by } \delta \text{ reduction on } \mathbf{y} && (37)
\end{aligned}$$

The Equation (37) above, in which we δ reduce $\mathbf{y}$, is impossible without [T-Cont].

By [Conv], and since $\mathbf{k} : B^+[(\mathbf{fst}\ e)^\div/\mathbf{x}] \to \boldsymbol{\alpha}$, to show $\mathbf{z}\ \boldsymbol{\alpha}\ \mathbf{k} : \boldsymbol{\alpha}$ it suffices to show that $(\mathbf{fst}\ e)^\div \equiv \mathbf{fst}\,(e^\div\ \_\ \mathbf{id})$.

Note that $(\mathbf{fst}\ e)^\div = \boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\,\mathbf{k}' : (A^+ \to \boldsymbol{\alpha}).$

$$e^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{y} : \Sigma\,\mathbf{x} : A^\div.\ B^\div.\ \mathbf{let\ z' = fst\ y} : A^\div\ \mathbf{in\ z'}\ \boldsymbol{\alpha}\ \mathbf{k}')$$

by definition of the translation.

By [$\equiv$-η], it suffices to show that

$$\begin{aligned}
&e^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{y} : \Sigma\,\mathbf{x} : A^\div.\ B^\div.\ \mathbf{let\ z' = fst\ y} : A^\div\ \mathbf{in\ z'}\ \boldsymbol{\alpha}\ \mathbf{k}') && && (38)\\
&\equiv (\boldsymbol{\lambda}\,\mathbf{y} : \Sigma\,\mathbf{x} : A^\div.\ B^\div.\ \mathbf{let\ z' = fst\ y\ in\ z'}\ \boldsymbol{\alpha}\ \mathbf{k}')\ (e^\div\ _\ \mathbf{id}) && [\equiv\text{-Cont}] && (39)\\
&\equiv (\mathbf{fst}\,(e^\div\ _\ \mathbf{id}))\ \boldsymbol{\alpha}\ \mathbf{k}' && \text{by reduction} && (40)
\end{aligned}$$

Notice that Equation (39) requires our new equivalence rule applied to the translation of the fst.

Case [Lam]

We give proofs for only the term-level functions; the type-level functions follow exactly the same structure as type-level function types. There are two subcases.

Sub-Case The function abstracts over a term, $\Gamma \vdash \lambda\, x : A.\ e : \Pi\, x : A.\ B$

We must show

$\Gamma^+ \vdash (\lambda\, x : A.\ e)^\div : (\Pi\, x : A.\ B)^\div$.

By definition of the translation, we must show

$\Gamma^+ \vdash \boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\,\mathbf{k} : (\Pi\,\mathbf{x} : A^\div.\ B^\div) \to \boldsymbol{\alpha}.$

$\quad (\mathbf{k}\ (\boldsymbol{\lambda}\,\mathbf{x} : A^\div.\ e^\div)) : \Pi\,\boldsymbol{\alpha} : *.\ (\Pi\,\mathbf{x} : A^\div.\ B^\div \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$

It suffices to show that

$\Gamma^+, \boldsymbol{\alpha} : *, \mathbf{k} : \Pi\,\mathbf{x} : A^\div.\ B^\div \to \boldsymbol{\alpha} \vdash \mathbf{k}\ (\boldsymbol{\lambda}\,\mathbf{x} : A^\div.\ e^\div) : \boldsymbol{\alpha}$.

By [App], it suffices to show that

$\Gamma^+, \boldsymbol{\alpha} : *, \mathbf{k} : \Pi\,\mathbf{x} : A^\div.\ B^\div \to \boldsymbol{\alpha} \vdash (\boldsymbol{\lambda}\,\mathbf{x} : A^\div.\ e^\div) : \Pi\,\mathbf{x} : A^\div.\ B^\div$

By part 2 of the induction hypothesis applied to $\Gamma, x : A \vdash e : B$, we know that

$\Gamma^+, \mathbf{x} : A^\div \vdash e^\div : B^\div$

It suffices to show that

$\vdash \Gamma^+, \boldsymbol{\alpha} : *, \mathbf{k} : \Pi\,\mathbf{x} : A^\div.\ B^\div \to \boldsymbol{\alpha}$ which follows easily by part 4 of the induction hypothesis applied to the typing derivations for A and B.

Sub-Case The function abstracts over a type, $\Gamma \vdash \lambda\,\alpha : \kappa.\ e : \Pi\,\alpha : \kappa.\ B$

We must show $\Gamma^+ \vdash (\lambda\,\alpha : \kappa.\ e)^\div : (\Pi\,\alpha : \kappa.\ B)^\div$.

By definition of the translation, we must show that

$\Gamma^+ \vdash \boldsymbol{\lambda}\,\boldsymbol{\alpha}_{ans} : *.\ \boldsymbol{\lambda}\,\mathbf{k} : (\Pi\,\boldsymbol{\alpha} : \kappa^+.\ B^\div) \to \boldsymbol{\alpha}.$

$\quad (\mathbf{k}\ (\boldsymbol{\lambda}\,\boldsymbol{\alpha} : \kappa^+.\ e^\div)) : \Pi\,\boldsymbol{\alpha}_{ans} : *.\ (\Pi\,\boldsymbol{\alpha} : \kappa^+.\ B^\div \to \boldsymbol{\alpha}_{ans}) \to \boldsymbol{\alpha}_{ans}$

It suffices to show that

$\Gamma^+, \boldsymbol{\alpha}_{ans} : *, \mathbf{k} : \Pi\,\boldsymbol{\alpha} : \kappa^+.\ B^\div \to \boldsymbol{\alpha}_{ans} \vdash \mathbf{k}\ (\boldsymbol{\lambda}\,\boldsymbol{\alpha} : \kappa^+.\ e^\div) : \boldsymbol{\alpha}_{ans}$.

By [App], it suffices to show that

$\Gamma^+, \boldsymbol{\alpha}_{ans} : *, \mathbf{k} : \Pi\,\boldsymbol{\alpha} : \kappa^+.\ B^\div \to \boldsymbol{\alpha}_{ans} \vdash (\boldsymbol{\lambda}\,\mathbf{x} : A^\div.\ e^\div) : \Pi\,\boldsymbol{\alpha} : \kappa^+.\ B^\div$

By part 2 of the induction hypothesis applied to $\Gamma, \alpha : \kappa \vdash e : B$, we know that
$\Gamma^+, \boldsymbol{\alpha} : \kappa^+ \vdash e^\div : B^\div$
It suffices to show that $\vdash \Gamma^+, \boldsymbol{\alpha}_{ans} : *, \mathbf{k} : \Pi\, \boldsymbol{\alpha} : \kappa^+. B^\div \to \boldsymbol{\alpha}_{ans}$ which follows easily by parts 5 and 4 of the induction hypothesis applied to the typing derivations for κ and B.

Case [App]

Sub-Case A term-level function applied to a term $\Gamma \vdash e_1\ e_2 : B[e_2/x]$

We must show that
$\Gamma^+ \vdash (e_1\ e_2)^\div : (B[e_2/x])^\div$
By definition of the translation, we must show:
$\Gamma^+ \vdash \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *. \boldsymbol{\lambda}\, \mathbf{k} : (B[e_2/x])^+ \to \boldsymbol{\alpha}. e_1^\div\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \Pi\, \mathbf{x} : A^\div. B^\div. (\mathbf{f}\ e_2^\div)\ \boldsymbol{\alpha}\ \mathbf{k}) : (B[e_2/x])^\div$
By part 6 of Lemma 3.1 and definition of $B^\div$, we must show:
$\Gamma^+ \vdash \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *. \boldsymbol{\lambda}\, \mathbf{k} : (B^+[e_2^\div/\mathbf{x}]) \to \boldsymbol{\alpha}.$
$e_1^\div\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \Pi\, \mathbf{x} : A^\div. B^\div. (\mathbf{f}\ e_2^\div)\ \boldsymbol{\alpha}\ \mathbf{k}) : \Pi\, \boldsymbol{\alpha} : *. (B^+[e_2^\div/\mathbf{x}] \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$
It suffices to show that

- $\Gamma^+ \vdash B^+[e_2^\div/\mathbf{x}] : \boldsymbol{\kappa}$ By the part 3 of the induction hypothesis we know that $\Gamma^+, \mathbf{x} : A^\div \vdash B^+ : \boldsymbol{\kappa}$, and by part 2 of the induction hypothesis we know that $\Gamma^+ \vdash e_2^\div : A^\div$, hence the goal follows by substitution.
- $\Gamma^+ \vdash e_1^\div : \Pi\, \boldsymbol{\alpha} : *. (\Pi\, \mathbf{x} : A^\div. B^\div \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$, which follows by part 2 of the induction hypothesis and by definition of $(\Pi\, x : A.\ B)^\div$.
- $\Gamma^+, \boldsymbol{\alpha} : *, \mathbf{k} : (B^+[e_2^\div/\mathbf{x}]) \to \boldsymbol{\alpha} \vdash (\boldsymbol{\lambda}\, \mathbf{f} : \Pi\, \mathbf{x} : A^\div. B^\div. (\mathbf{f}\ e_2^\div)\ \boldsymbol{\alpha}\ \mathbf{k}) : \Pi\, \mathbf{x} : A^\div. B^\div \to \boldsymbol{\alpha}$, which follows since by part 2 of the induction hypothesis $e_2^\div : A^\div$ we know $(\mathbf{f}\ e_2^\div) : B^\div[e_2^\div/\mathbf{x}]$ and by definition $B^\div[e_2^\div/\mathbf{x}] = \Pi\, \boldsymbol{\alpha} : *. (B^+[e_2^\div/\mathbf{x}] \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$.

Sub-Case A term-level function applied to a type $\Gamma \vdash e_1\ A : B[A/\alpha]$

The proof is similar to the previous case, but relies on showing that $\Gamma^+ \vdash A^+ : \kappa^+$, which follows by part 3 of the induction hypothesis.

Sub-Case A type-level function applied to a term $\Gamma \vdash A\ e : \kappa[e/x]$

This case is straightforward by the part 3 and part 2 of the induction hypothesis.

Sub-Case A type-level function applied to a type $\Gamma \vdash A\ B : \kappa[B/\alpha]$

This case is straightforward by the part 3 of the induction hypothesis.

Case [Conv] $\Gamma \vdash e : A$ such that $\Gamma \vdash e : B$ and $A \equiv B$.

We must show that $e^\div$ has type $A^\div = \Pi\, \boldsymbol{\alpha} : *. (A^+ \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$.
By the induction hypothesis, we know that $e^\div : B^\div = \Pi\, \boldsymbol{\alpha} : *. (B^+ \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$. By [Conv] it suffices to show that $A^+ \equiv B^+$, which follows by Lemma 3.4. □

To recover a simple statement of the type-preservation theorem over the PTS syntax, we define two meta-functions for translating terms and types depending on their use. We define CPS $[\![t]\!]$ to translate a PTS expression in "term" position, *i.e.*, when used on the left side of a type annotation as in $t : t'$, and we define CPST $[\![t']\!]$ to translate an expression in "type" position, *i.e.*, when used on the right side of a type annotation. We define these in terms of the translation shown above, noting that for every $t : t'$ in the PTS syntax, one of the following is true: t is a term e and t' is a type A in the explicit syntax; t is a type A and t' is a kind κ in the explicit syntax; or t is a kind κ and t' is a universe U in the explicit syntax.

$$\begin{array}{llll} \text{CPS } [\![t]\!] & \stackrel{\text{def}}{=} & e^\div \text{ when } t \text{ is a term} & \text{CPST } [\![t']\!] \stackrel{\text{def}}{=} A^\div \text{ when } t' \text{ is a type} \\ \text{CPS } [\![t]\!] & \stackrel{\text{def}}{=} & A^+ \text{ when } t \text{ is a type} & \text{CPST } [\![t']\!] \stackrel{\text{def}}{=} \kappa^+ \text{ when } t' \text{ is a kind} \\ \text{CPS } [\![t]\!] & \stackrel{\text{def}}{=} & \kappa^+ \text{ when } t \text{ is a kind} & \text{CPST } [\![t']\!] \stackrel{\text{def}}{=} U^+ \text{ when } t' \text{ is a universe} \end{array}$$

This notation is based on Barthe and Uustalu [2002].

Theorem 3.6 (CPS^n is Type Preserving (PTS syntax)). *$\Gamma \vdash t : t'$ then $\Gamma^+ \vdash$ CPS $[\![t]\!]$: CPST $[\![t']\!]$*

3.2 Proof of Correctness for CPS^n

Since type preservation in a dependently typed language requires preserving reduction sequences, we have already done most of the work to prove two other compiler correctness properties: correctness of separate compilation, and whole-program compiler correctness. To specify compiler correctness, we need an independent specification that tells

us how source values—or, more generally, observations—are related to target values. For instance, when compiling to C we might specify that the number 5 is related to the bits $0x101$. Without a specification, independent of the compiler, there is no definition that the compiler can be correct with respect to. In our setting, such an independent specification is simple to construct. We can add ground values such as booleans to our language with the obvious cross-language relation: $\mathsf{True} \approx \mathbf{True}$ and $\mathsf{False} \approx \mathbf{False}$.

Next, to define separate compilation, we need a definition of linking. We can define linking as substitution, and define valid closing substitutions γ as follows.

$$\Gamma \vdash \gamma \stackrel{\text{def}}{=} \forall x : A \in \Gamma.\ \vdash \gamma(x) : \gamma(A)$$

We extend the compiler in a straightforward way to compile closing substitutions, written $\gamma^\div$, and allow compiled code to be linked with the compilation of any valid closing substitution γ. This definition supports a separate compilation theorem that allows linking with the output of our compiler, but not with the output of other compilers.

Now we can show that the compiler is correct with respect to separate compilation: if we first link and run to a value, we get a related value when we compile and then link with the compiled closing substitution. Since our target language is in CPS form, we should apply the halt continuation, $\mathbf{id}$, and compare the ground values.

Theorem 3.7 (Separate Compilation Correctness). *If* $\Gamma \vdash e : A$ *where* A *is ground, and* $\gamma(e) \rhd^\star v$ *then* $\gamma^\div(e^\div)\ A^+\ \mathbf{id} \rhd^\star \mathbf{v}$ *and* $v \approx \mathbf{v}$.

Proof. Since reduction implies equivalence, we reason in terms of equivalence. By Lemma 3.3, $(\gamma(e))^\div \rhd^\star \mathbf{e}$ and $v^\div \equiv \mathbf{e}$. By Lemma 3.1, $(\gamma(e))^\div \equiv \gamma^\div(e^\div)$, hence $\gamma^\div(e^\div) \rhd^\star \mathbf{e}$ and $v^\div \equiv \mathbf{e}$. Since the translation on all ground values is $v^\div = \boldsymbol{\lambda}\,\boldsymbol{\alpha}.\,\boldsymbol{\lambda}\,\mathbf{k}.\,\mathbf{k}\ \mathbf{v}$, where $v \approx \mathbf{v}$, we know $v^\div\ A^+\ \mathbf{id} \rhd^\star \mathbf{v}$ such that $v \approx \mathbf{v}$. Since $v^\div \equiv \mathbf{e} \equiv \gamma^\div(e^\div)$, we also know that $\gamma^\div(e^\div)\ A^+\mathbf{id} \rhd^\star \mathbf{v}'$ and $\mathbf{v}' \equiv \mathbf{v}$. Since $\mathbf{v}$ is ground, $\mathbf{v}' = \mathbf{v}$ and $v \approx \mathbf{v}'$. □

Corollary 3.8 (Whole-Program Compiler Correctness). *If* $\vdash e : A$ *and* $e \rhd^\star v$ *then* $e^\div\ A^+\ \mathbf{id} \rhd^\star \mathbf{v}$ *and* $v \approx \mathbf{v}$.

Our separate-compilation correctness theorem is similar to the guarantees provided by SepCompCert [Kang et al. 2016] in that it supports linking with only the output of the same compiler. We could support more flexible notions of linking—such as linking with code produced by different compilers, from different source languages, or code written directly in the target language—by defining an independent specification for when closing substitutions are related across languages (*e.g.*, [Ahmed and Blume 2011; Neis et al. 2015; New et al. 2016; Perconti and Ahmed 2014]).

$\boxed{\Gamma \vdash U \rightsquigarrow^n_U \mathbf{U}}$

$$\frac{}{\Gamma \vdash * \rightsquigarrow^n_U *}\ [\mathrm{CPS}^n_U\text{-Star}] \qquad \frac{}{\Gamma \vdash \Box \rightsquigarrow^n_U \Box}\ [\mathrm{CPS}^n_U\text{-Box}]$$

$\boxed{\Gamma \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa}}$ Lemma 3.5 will show $\Gamma^+ \vdash \kappa^+ : U^+$

$$\frac{}{\Gamma \vdash * : \Box \rightsquigarrow^n_\kappa *}\ [\mathrm{CPS}^n_\kappa\text{-Ax}] \qquad \frac{\Gamma \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash \kappa' : U' \rightsquigarrow^n_\kappa \boldsymbol{\kappa'}}{\Gamma \vdash \Pi\, \alpha : \kappa.\, \kappa' : U' \rightsquigarrow^n_\kappa \mathbf{\Pi}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \boldsymbol{\kappa'}}\ [\mathrm{CPS}^n_\kappa\text{-ProdK}]$$

$$\frac{\Gamma \vdash A : \kappa' \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa}}{\Gamma \vdash \Pi\, x : A.\, \kappa : U \rightsquigarrow^n_\kappa \mathbf{\Pi}\, \mathbf{x} : \mathbf{A}.\, \boldsymbol{\kappa}}\ [\mathrm{CPS}^n_\kappa\text{-ProdA}]$$

$\boxed{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A}}$ Lemma 3.5 will show $\Gamma^+ \vdash A^+ : \kappa^+$

$$\frac{}{\Gamma \vdash \alpha : \kappa \rightsquigarrow^n_A \boldsymbol{\alpha}}\ [\mathrm{CPS}^n_A\text{-Var}] \qquad \frac{\Gamma \vdash A : \kappa' \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : \kappa \rightsquigarrow^n_A \mathbf{B}}{\Gamma \vdash \lambda\, x : A.\, B : \Pi\, x : A.\, \kappa \rightsquigarrow^n_A \boldsymbol{\lambda}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Constr}]$$

$$\frac{\Gamma \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash B : \kappa' \rightsquigarrow^n_A \mathbf{B}}{\Gamma \vdash \lambda\, \alpha : \kappa.\, B : \Pi\, \alpha : \kappa.\, \kappa' \rightsquigarrow^n_A \boldsymbol{\lambda}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Abs}]$$

$$\frac{\Gamma \vdash A : \Pi\, x : B.\, \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash e : B \rightsquigarrow^n_e \mathbf{e}}{\Gamma \vdash A\ e : \kappa[e/x] \rightsquigarrow^n_A \mathbf{A}\ \mathbf{e}}\ [\mathrm{CPS}^n_A\text{-AppConstr}]$$

$$\frac{\Gamma \vdash A : \Pi\, \alpha : \kappa'.\, \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash B : \kappa' \rightsquigarrow^n_A \mathbf{B}}{\Gamma \vdash A\ B : \kappa[B/\alpha] \rightsquigarrow^n_A \mathbf{A}\ \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Inst}]$$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : \kappa' \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \Pi\, x : A.\, B : \kappa' \rightsquigarrow^n_A \mathbf{\Pi}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Prod}]$$

$$\frac{\Gamma \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, x : A \vdash B : \kappa' \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \Pi\, \alpha : \kappa.\, B : \kappa' \rightsquigarrow^n_A \mathbf{\Pi}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-ProdK}]$$

$$\frac{\Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x = e : A \vdash B : \kappa' \rightsquigarrow^n_A \mathbf{B}}{\Gamma \vdash \mathsf{let}\, x = e : A\, \mathsf{in}\, B : \kappa' \rightsquigarrow^n_A \mathsf{let}\, \mathbf{x} = \mathbf{e} : \mathbf{A}\, \mathsf{in}\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Let}]$$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha = A : \kappa \vdash B : \_ \rightsquigarrow^n_A \mathbf{B}}{\Gamma \vdash \mathsf{let}\, \alpha = A : \kappa\, \mathsf{in}\, B : \_ \rightsquigarrow^n_A \mathsf{let}\, \boldsymbol{\alpha} = \mathbf{A} : \boldsymbol{\kappa}\, \mathsf{in}\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-LetK}]$$

$$\frac{\Gamma \vdash A : * \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : * \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \Sigma\, x : A.\, B : * \rightsquigarrow^n_A \mathbf{\Sigma}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Sigma}]$$

$$\frac{\Gamma \vdash A : \kappa' \qquad \Gamma \vdash \kappa \equiv \kappa' \qquad \Gamma \vdash A : \kappa' \rightsquigarrow^n_A \mathbf{A}}{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A}}\ [\mathrm{CPS}^n_A\text{-Conv}]$$

$\boxed{\Gamma \vdash A : * \rightsquigarrow^n_{A^\div} \mathbf{A}}$ Lemma 3.5 will show $\Gamma^+ \vdash A^\div : *^+$

$$\frac{\Gamma \vdash A : * \rightsquigarrow^n_A \mathbf{A}}{\Gamma \vdash A : * \rightsquigarrow^n_{A^\div} \mathbf{\Pi}\, \boldsymbol{\alpha} : *.\, (\mathbf{A} \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}}\ [\mathrm{CPS}^n_A\text{-Comp}]$$

Fig. 10. CPS^n of Universes, Kinds, and Types

$\boxed{\Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{A}}$ Lemma 3.5 will show $\Gamma^+ \vdash e^\div : A^\div$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A}}{\Gamma \vdash x : A \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{A} \to \boldsymbol{\alpha}.\, \mathbf{x}\ \boldsymbol{\alpha}\ \mathbf{k}}\ [\mathrm{CPS}^n_e\text{-Var}]$$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : \kappa' \rightsquigarrow^n_{A^\div} \mathbf{B} \qquad \Gamma, x : A \vdash e : B \rightsquigarrow^n_e \mathbf{e}}{\Gamma \vdash \lambda\, x : A.\, e : \Pi\, x : A.\, B \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : (\Pi\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}) \to \boldsymbol{\alpha}.\, \mathbf{k}\ (\boldsymbol{\lambda}\, \mathbf{x} : \mathbf{A}.\, \mathbf{e})}\ [\mathrm{CPS}^n_e\text{-Fun}]$$

$$\frac{\Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B} \qquad \Gamma, \alpha : \kappa \vdash e : B \rightsquigarrow^n_e \mathbf{e}}{\Gamma \vdash \lambda\, \alpha : \kappa.\, e : \Pi\, \alpha : \kappa.\, B \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : (\Pi\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}) \to \boldsymbol{\alpha}_{ans}.\, \mathbf{k}\ (\boldsymbol{\lambda}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{e})}\ [\mathrm{CPS}^n_e\text{-Abs}]$$

$$\frac{\begin{array}{c}\Gamma \vdash e : \Pi\, x : A.\, B \rightsquigarrow^n_e \mathbf{e} \\ \Gamma \vdash A : \kappa \rightsquigarrow^n_{A^\div} \mathbf{A}^\div \qquad \Gamma, x : A \vdash B : \kappa' \rightsquigarrow^n_{A^\div} \mathbf{B}^\div \qquad \Gamma, x : A \vdash B : \kappa' \rightsquigarrow^n_A \mathbf{B}^+ \qquad \Gamma \vdash e' : A \rightsquigarrow^n_e \mathbf{e}'\end{array}}{\begin{array}{l}\Gamma \vdash e\ e' : B[e'/x] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : (\mathbf{B}^+[\mathbf{e}'/\mathbf{x}]) \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e}\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \Pi\, \mathbf{x} : \mathbf{A}^\div.\, \mathbf{B}^\div.\, (\mathbf{f}\ \mathbf{e}')\ \boldsymbol{\alpha}\ \mathbf{k})\end{array}}\ [\mathrm{CPS}^n_e\text{-App}]$$

$$\frac{\Gamma \vdash e : \Pi\, \alpha : \kappa.\, B \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B}^\div \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^n_A \mathbf{B}^+ \qquad \Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A}}{\begin{array}{l}\Gamma \vdash e\ A : B[A/\alpha] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : (\mathbf{B}^+[\mathbf{A}/\boldsymbol{\alpha}]) \to \boldsymbol{\alpha}_{ans}. \\ \qquad \mathbf{e}\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \Pi\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}^\div.\, (\mathbf{f}\ \mathbf{A})\ \boldsymbol{\alpha}_{ans}\ \mathbf{k})\end{array}}\ [\mathrm{CPS}^n_e\text{-Inst}]$$

$$\frac{\Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x = e : A \vdash B : \kappa' \rightsquigarrow^n_A \mathbf{B} \qquad \Gamma, x = e : A \vdash e' : B \rightsquigarrow^n_e \mathbf{e}'}{\Gamma \vdash \mathsf{let}\ x = e : A\ \mathsf{in}\ e' : B[e/x] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{B}[\mathbf{e}/\mathbf{x}] \to \boldsymbol{\alpha}.\, \mathsf{let}\ \mathbf{x} = \mathbf{e} : \mathbf{A}\ \mathsf{in}\ \mathbf{e}'\ \boldsymbol{\alpha}\ \mathbf{k}}\ [\mathrm{CPS}^n_e\text{-Let}]$$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha = A : \kappa \vdash B : \kappa' \rightsquigarrow^n_A \mathbf{B} \qquad \Gamma, \alpha = A : \kappa \vdash e : B \rightsquigarrow^n_e \mathbf{e}}{\Gamma \vdash \mathsf{let}\ \alpha = A : \kappa\ \mathsf{in}\ e : B[A/\alpha] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{B}[\mathbf{A}/\mathbf{x}] \to \boldsymbol{\alpha}_{ans}.\, \mathsf{let}\ \boldsymbol{\alpha} = \mathbf{A} : \boldsymbol{\kappa}\ \mathsf{in}\ \mathbf{e}\ \boldsymbol{\alpha}_{ans}\ \mathbf{k}}\ [\mathrm{CPS}^n_e\text{-LetK}]$$

$$\frac{\Gamma \vdash e : B \rightsquigarrow^n_e \mathbf{e}}{\Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{e}}\ [\mathrm{CPS}^n_e\text{-Conv}]$$

Fig. 11. CPS^n of Terms

$$\frac{\Gamma \vdash e_1 : A \rightsquigarrow^n_e \mathbf{e}_1 \qquad \Gamma \vdash e_2 : B[e_1/x] \rightsquigarrow^n_e \mathbf{e}_2 \qquad \Gamma \vdash A : * \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : * \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \langle e_1, e_2 \rangle : \Sigma\, x : A.\, B \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B} \to \boldsymbol{\alpha}.\, \mathbf{k}\ \langle \mathbf{e}_1, \mathbf{e}_2 \rangle\ \mathsf{as}\ \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^n_e\text{-Pair}]$$

$$\frac{\Gamma \vdash A : * \rightsquigarrow^n_{A^\div} \mathbf{A}^\div \qquad \Gamma, x : A \vdash B : * \rightsquigarrow^n_{A^\div} \mathbf{B}^\div \qquad \Gamma \vdash A : * \rightsquigarrow^n_A \mathbf{A}^+ \qquad \Gamma \vdash e : \Sigma\, x : A.\, B \rightsquigarrow^n_e \mathbf{e}}{\begin{array}{l}\Gamma \vdash \mathsf{fst}\ e : A \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{A}^+ \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e}\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{y} : \Sigma\, \mathbf{x} : \mathbf{A}^\div.\, \mathbf{B}^\div.\, \mathsf{let}\ \mathbf{z} = \mathsf{fst}\ \mathbf{y}\ \mathsf{in}\ \mathbf{z}\ \boldsymbol{\alpha}\ \mathbf{k})\end{array}}\ [\mathrm{CPS}^n_e\text{-Fst}]$$

$$\frac{\begin{array}{c}\Gamma \vdash A : * \rightsquigarrow^n_{A^\div} \mathbf{A}^\div \qquad \Gamma, x : A \vdash B : * \rightsquigarrow^n_{A^\div} \mathbf{B}^\div \\ \Gamma, x : A \vdash B : * \rightsquigarrow^n_A \mathbf{B}^+ \qquad \Gamma \vdash (\mathsf{fst}\ e) : A \rightsquigarrow^n_e (\mathsf{fst}\ e)^\div \qquad \Gamma \vdash e : \Sigma\, x : A.\, B \rightsquigarrow^n_e \mathbf{e}\end{array}}{\begin{array}{l}\Gamma \vdash \mathsf{snd}\ e : B[(\mathsf{fst}\ e)/x] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{B}^+[(\mathsf{fst}\ e)^\div/\mathbf{x}] \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e}\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{y} : \Sigma\, \mathbf{x} : \mathbf{A}^\div.\, \mathbf{B}^\div.\, \mathsf{let}\ \mathbf{z} = \mathsf{snd}\ \mathbf{y}\ \mathsf{in}\ \mathbf{z}\ \boldsymbol{\alpha}\ \mathbf{k})\end{array}}\ [\mathrm{CPS}^n_e\text{-Snd}]$$

Fig. 12. CPS^n of Terms (pairs)

$\boxed{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma}}$ Lemma 3.5 will show $\vdash \Gamma^+$

$$\frac{}{\vdash \cdot \rightsquigarrow^n \cdot}\ [\mathrm{CPS}^n_\Gamma\text{-Empty}] \qquad \frac{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^n_{A^\div} \mathbf{A}}{\vdash \Gamma, x : A \rightsquigarrow^n \mathbf{\Gamma}, \mathbf{x} : \mathbf{A}}\ [\mathrm{CPS}^n_\Gamma\text{-AssumT}]$$

$$\frac{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma} \qquad \Gamma \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa}}{\vdash \Gamma, \alpha : \kappa \rightsquigarrow^n \mathbf{\Gamma}, \boldsymbol{\alpha} : \boldsymbol{\kappa}}\ [\mathrm{CPS}^n_\Gamma\text{-AssumK}] \qquad \frac{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{e}}{\vdash \Gamma, x = e : A \rightsquigarrow^n \mathbf{\Gamma}, \mathbf{x} = \mathbf{e} : \mathbf{A}}\ [\mathrm{CPS}^n_\Gamma\text{-Def}]$$

$$\frac{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa}}{\vdash \Gamma, \alpha = A : \kappa \rightsquigarrow^n \mathbf{\Gamma}, \boldsymbol{\alpha} = \mathbf{A} : \boldsymbol{\kappa}}\ [\mathrm{CPS}^n_\Gamma\text{-DefT}]$$

Fig. 13. CPS^n of Environments

4 CALL-BY-VALUE CPS TRANSLATION OF CC

In this section, we present the call-by-value CPS translation (CPS^{v}) of CC. First, we redefine our short-hand from Section 3 to refer to call-by-value translation.

$$A^{\div} \stackrel{\text{def}}{=} \mathbf{A} \text{ where } \Gamma \vdash A : * \rightsquigarrow^{v}_{A^{\div}} \mathbf{A} \qquad U^{+} \stackrel{\text{def}}{=} \mathbf{U} \text{ where } \Gamma \vdash U \rightsquigarrow^{v}_{U} \mathbf{U}$$
$$e^{\div} \stackrel{\text{def}}{=} \mathbf{e} \text{ where } \Gamma \vdash e : A \rightsquigarrow^{v}_{e} \mathbf{e} \qquad \kappa^{+} \stackrel{\text{def}}{=} \mathbf{e} \text{ where } \Gamma \vdash \kappa : U \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa}$$
$$A^{+} \stackrel{\text{def}}{=} \mathbf{A} \text{ where } \Gamma \vdash A : \kappa \rightsquigarrow^{v}_{A} \mathbf{A}$$

Unlike CBN, the CBV translation forces every computation to a value. Therefore, every dependent elimination requires our new [T-Cont] typing rule. Moreover, all substitutions of a term into a type must substitute *values* instead of computations so all dependent type annotations must explicitly convert computations to values by supplying the identity continuation.

We present our call-by-value translation CPS^{v} in Figures 14 and 15. In general, CPS^{v} differs from CPS^{n} in two ways. First, all term variables must have value types, so the translation rules for all binding constructs now use the value translation for type annotations. Second, we change the definition of value types so that functions must receive values and pairs must contain values.

The universe translation is unchanged from CPS^{n}. The kind translation (Figure 14) has changed in only one place. Now the translation rule [CPS^{v}_{κ}-ProdA] translates the kind of type-level functions $\Pi\, x : A.\, \kappa$ to accept a *value* as argument $\mathbf{x} : A^{+}$.

The type translation (Figure 14) has multiple rules with variable annotations that have changed from CBN. The computation translation of types is unchanged. In the value translation, similar to the kind translation, dependent function types that abstract over terms now translate the argument annotation $x : A$ using the value translation. Dependent pair types $\Sigma\, x : A.\, B$ now translate to pairs of values $\Sigma\, \mathbf{x} : A^{+}.\, B^{+}$. When terms appear in the type language, such as in [CPS^{v}_{A}-AppConstr] and [CPS^{v}_{A}-Let], we must explicitly convert the computation to a value to maintain the invariant that all term variables refer to term values. Hence in [CPS^{v}_{A}-AppConstr] we translate a type-level application with a term argument $A\ e$ to $A^{+}\ (e^{\div}\ B^{+}\ \mathbf{id})$. We similarly translate let-bound terms [CPS^{v}_{A}-Let] by casting the computation to a value. While expressions of the form $A^{+}\ (e^{\div}\ B^{+}\ \mathbf{id})$ are not in CPS form, this expression is a type and will be evaluated during type checking. Terms that evaluate at run time are always in proper CPS form and do not return.

The term translation (Figure 15 and Figure 16) changes in three major ways. As in Section 3, we implicitly have a computation and a value translation on term values, with the latter inlined into the former. First, unlike in CPS^{n}, variables are values, whereas the translation must produce a computation. Therefore, we translate x by "value η-expansion" into $\boldsymbol{\lambda}\, \boldsymbol{\alpha}.\, \boldsymbol{\lambda}\, \mathbf{k}.\, \mathbf{k}\ \mathbf{x}$, a computation that immediately applies the continuation to the value. Second, as discussed above, we change the translation of application [CPS^{v}_{e}-App] to force the evaluation of the function argument. Third and finally, in the translation of pairs [CPS^{v}_{e}-Pair], we force the evaluation of the components of the pair and produce a pair of values for the continuation. Note that in cases of the translation where we have types with dependency—[CPS^{v}_{e}-App], [CPS^{v}_{e}-Let], [CPS^{v}_{e}-Pair], and [CPS^{v}_{e}-Snd]—we cast computations to values in the types by applying the identity continuation, and require the @ form to use our new typing rule.

Interestingly, because typing the application of a continuation is essentially a dependent let, we can simplify the translation of pairs. We present this in the rule [CPS^{v}_{e}-Pair-Alt]. Instead of explicitly substituting the value of $\mathbf{e}_1$ into the type $\mathbf{B}$, we simply use the same variable name $\mathbf{x}$ to bind the value of $\mathbf{e}_1$ in both the term and the type. Since that variable is free in the type annotation $\mathbf{B}$ on the variable $\mathbf{x}_2$, we implicitly substitute its value into $\mathbf{B}$ rather than being so explicitly. This is rather subtle so we prefer the more direct and explicit translation, [CPS^{v}_{e}-Pair].

Given the translation of binding constructs in the language, the translation of the environment (Figure 17) should be unsurprising. Since all variables are values, we translate term variables $x : A$ using the value translation on types to produce $\mathbf{x} : A^{+}$ instead of $\mathbf{x} : A^{\div}$. We must also translate term definitions $x = e : A$ by casting the computation to a value, producing $\mathbf{x} = e^{\div}\ A^{+}\ \mathbf{id} : A^{+}$.

4.1 Proof of Type-Preservation for CPS^{v}

The structure of the type-preservation proof is the same as in Section 3. First we prove that the translation commutes with substitution, then that reduction sequences are preserved, then that equivalence is preserved, and finally that

typing is preserved. Except for Lemma 4.2 (CPS^{v} Compositionality), the proofs of the supporting lemmas are essentially the same as in Section 3.

We begin with a technical lemma that is essentially an η principle for CPS'd computations. In the CPS^{v} setting, we must frequently reason about normal forms of CPS'd computations. This lemma provides a useful abstraction for doing so.[1] The lemma states that any CPS'd computation $e^{\div}$ is equivalent to a new CPS'd computation that accepts a continuation $\mathbf{k}$ simply applies $e^{\div}$ to $\mathbf{k}$. The proof is straightforward. Note the type annotations are mismatched, as in our explanation of coherence in Section 3.1, but the behaviors of the terms are the same and equivalence is untyped.

Lemma 4.1 (CPS^{v} Computation η). $e^{\div} \equiv \lambda\, \alpha : *.\, \lambda\, \mathbf{k} : \mathbf{A} \to \alpha.\, e^{\div}\ @\ \alpha\ (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{k}\ \mathbf{x})$

Proof. Note that $e^{\div} \equiv \lambda\, \alpha : *.\, \lambda\, \mathbf{k} : \mathbf{A} \to \alpha.\, e^{\div}\ \alpha\ (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{k}\ \mathbf{x})$, by η-equivalence. By transitivity, it suffices to show that

$\lambda\, \alpha : *.\, \lambda\, \mathbf{k} : \mathbf{A} \to \alpha.\, e^{\div}\ \alpha\ (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{k}\ \mathbf{x}) \equiv \lambda\, \alpha : *.\, \lambda\, \mathbf{k} : \mathbf{A} \to \alpha.\, e^{\div}\ @\ \alpha\ (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{k}\ \mathbf{x})$

Intuitively, this is true since @ dynamically behaves exactly like application, only changing which typing rule is used. Since our equivalence is untyped, the semantics of @ as far as equivalence is concerned is no different than normal application.

Formally, note that by definition of the translation, $e^{\div}$ must be of the form $\lambda\, \alpha.\, \lambda\, \mathbf{k}.\, \mathbf{e}'$.

The goal follows since

$\lambda\, \alpha : *.\, \lambda\, \mathbf{k} : \mathbf{A} \to \alpha.\, (\lambda\, \alpha.\, \lambda\, \mathbf{k}.\, \mathbf{e}')\ \alpha\ (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{k}\ \mathbf{x}) \rhd^{\star} \lambda\, \alpha : *.\, \lambda\, \mathbf{k} : \mathbf{A} \to \alpha.\, \mathbf{e}'$

and

$\lambda\, \alpha : *.\, \lambda\, \mathbf{k} : \mathbf{A} \to \alpha.\, (\lambda\, \alpha.\, \lambda\, \mathbf{k}.\, \mathbf{e}')\ @\ \alpha\ (\lambda\, \mathbf{x} : \mathbf{B}.\, \mathbf{k}\ \mathbf{x}) \rhd^{\star} \lambda\, \alpha : *.\, \lambda\, \mathbf{k} : \mathbf{A} \to \alpha.\, \mathbf{e}'$ □

Since variables are values in call-by-value, we adjust the statement of Lemma 4.2 to cast computations to values. Proving this lemma now requires our new equivalence rule [≡-Cont] for cases involving substitution of terms. By convention, all terms being translated have an implicit typing derivation, so the omitted types are easy to reconstruct.

[1] The proofs for the CBN setting only require a specialized instance of this property although the general form holds.

Lemma 4.2 (CPS$^{\upsilon}$ Compositionality).

(1) $(\kappa[A/\alpha])^+ \equiv \kappa^+[A^+/\boldsymbol{\alpha}]$
(2) $(\kappa[e/x])^+ \equiv \kappa^+[e^{\div}\ \_\ \mathbf{id}/\mathbf{x}]$
(3) $(A[B/\alpha])^+ \equiv A^+[B^+/\boldsymbol{\alpha}]$
(4) $(A[e/x])^+ \equiv A^+[e^{\div}\ \_\ \mathbf{id}/\mathbf{x}]$
(5) $(A[B/\alpha])^{\div} \equiv A^{\div}[B^+/\boldsymbol{\alpha}]$
(6) $(A[e/x])^{\div} \equiv A^{\div}[e^{\div}\ \_\ \mathbf{id}/\mathbf{x}]$
(7) $(e[A/\alpha])^{\div} \equiv e^{\div}[A^+/\boldsymbol{\alpha}]$
(8) $(e[e'/x])^{\div} \equiv e^{\div}[e'^{\div}\ \_\ \mathbf{id}/\mathbf{x}]$

Proof. The proof is straightforward by induction on the typing derivation of the expression t being substituted into. Part 6 follows immediately by part 3 of the induction hypothesis. Part 7 follows immediately by part 4 of the induction hypothesis. We give representative cases for the other parts. In most cases, it suffices to show that the two terms are syntactically identical.

Case [Ax-*] $t = U$, parts 1 and 2. Trivial, since no free variables appear in U.
Case [Prod-*] $t = \Pi\, x : B.\, \kappa'$
Sub-Case Part 1. We must show that $((\Pi\, x : B.\, \kappa')[A/\alpha])^+ = (\Pi\, x : B.\, \kappa')^+[A^+/\boldsymbol{\alpha}]$.

$$
\begin{aligned}
& ((\Pi\, x : B.\, \kappa')[A/\alpha])^+ && && (41)\\
&= (\Pi\, x : B[A/\alpha].\, \kappa'[A/\alpha])^+ && \text{by definition of substitution} && (42)\\
&= \mathbf{\Pi}\, \mathbf{x}' : (B[A/\alpha])^+.\, (\kappa'[A/\alpha])^+ && \text{by definition of the translation} && (43)\\
&= \mathbf{\Pi}\, \mathbf{x}' : B^+[A^+/\alpha].\, \kappa'^+[A^+/\alpha] && \text{by parts 1 and 3 of the induction hypothesis} && (44)\\
&= (\mathbf{\Pi}\, \mathbf{x}' : B^+.\, \kappa'^+)[A^+/\boldsymbol{\alpha}] && \text{by definition of substitution} && (45)\\
&= (\Pi\, x' : B.\, \kappa')^+[A^+/\boldsymbol{\alpha}] && \text{by definition of the translation} && (46)
\end{aligned}
$$

Sub-Case Part 2. Similar to the previous subcase.
Case [Var]
Sub-Case $t = \alpha'$, part 3. Part 4 is trivial since x is not free in α.
We must show that $(\alpha'[A/\alpha])^+ = \boldsymbol{\alpha}'[A^+/\boldsymbol{\alpha}]$.
Sub-Case $\alpha = \alpha'$. It suffices to show that $A^+ = A^+$, which is trivial.
Sub-Case $\alpha \neq \alpha'$. Trivial.
Sub-Case $t = x$, part 7 is trivial.
Sub-Case Part 8
We must show $(x[e/x'])^{\div} = x^{\div}[e^{\div}\ \_\ \mathbf{id}/\mathbf{x}']$.
W.l.o.g., assume $x = x'$.

$$
\begin{aligned}
& (x[e/x])^{\div} && && (47)\\
&= e^{\div} && \text{by definition of substitution} && (48)\\
&\equiv \boldsymbol{\lambda}\, \boldsymbol{\alpha}.\, \boldsymbol{\lambda}\, \mathbf{k}.\, (e^{\div}\ @\ \boldsymbol{\alpha}\ \boldsymbol{\lambda}\, \mathbf{x}.\, \mathbf{k}\ \mathbf{x}) && \text{by Lemma 4.1} && (49)\\
&\equiv \boldsymbol{\lambda}\, \boldsymbol{\alpha}.\, \boldsymbol{\lambda}\, \mathbf{k}.\, (\boldsymbol{\lambda}\, \mathbf{x}.\, \mathbf{k}\ \mathbf{x})\ (e^{\div}\ \_\ \mathbf{id}) && \text{by [}\equiv\text{-Cont]} && (50)\\
&= (\boldsymbol{\lambda}\, \boldsymbol{\alpha}.\, \boldsymbol{\lambda}\, \mathbf{k}.\, (\boldsymbol{\lambda}\, \mathbf{x}.\, \mathbf{k}\ \mathbf{x})\ \mathbf{x})[(e^{\div}\ \_\ \mathbf{id})/\mathbf{x}] && \text{by substitution} && (51)\\
&\equiv (\boldsymbol{\lambda}\, \boldsymbol{\alpha}.\, \boldsymbol{\lambda}\, \mathbf{k}.\, \mathbf{k}\ \mathbf{x})[(e^{\div}\ \_\ \mathbf{id})/\mathbf{x}] && \text{by } \triangleright_{\beta} && (52)\\
&= x^{\div}[(e^{\div}\ \_\ \mathbf{id})/\mathbf{x}] && \text{by definition of translation} && (53)
\end{aligned}
$$

Case [App] $e_1\ e_2$
Sub-Case Part 7

We must show $((e_1\ e_2)[A'/\alpha'])^\div = (e_1\ e_2)^\div[A'^+/\boldsymbol{\alpha}']$.

$$
\begin{aligned}
&((e_1\ e_2)[A'/\alpha'])^\div && && (54)\\
&= (e_1[A/\alpha']\ e_2[A'/\alpha'])^\div && \text{by substitution} && (55)\\
&= \boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\,\mathbf{k} : ((B[A'/\alpha'])^+[(e_2[A'/\alpha'])^\div/\mathbf{x}]) \to \boldsymbol{\alpha}. && \text{by translation} && (56)\\
&\quad (e_1[A'/\alpha'])^\div\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{f} : \boldsymbol{\Pi}\,\mathbf{x} : (A[A'/\alpha'])^+.\ (B[A'/\alpha'])^\div.\\
&\qquad\qquad (e_2[A'/\alpha'])^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{x} : (A[A'/\alpha'])^+.\ (\mathbf{f}\ \mathbf{x})\ \boldsymbol{\alpha}\ \mathbf{k}))\\
&= \boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\,\mathbf{k} : (B^+[A'^+/\boldsymbol{\alpha}'][e_2^\div[A'^+/\boldsymbol{\alpha}']/\mathbf{x}]) \to \boldsymbol{\alpha}. && \text{by IH (3,5,7)} && (57)\\
&\quad e_1^\div[A'^+/\boldsymbol{\alpha}']\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{f} : \boldsymbol{\Pi}\,\mathbf{x} : A^+[A'^+/\alpha'].\ B^\div[A'^+/\boldsymbol{\alpha}'].\\
&\qquad\qquad e_2^\div[A'^+/\boldsymbol{\alpha}']\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{x} : A^+[A'^+/\boldsymbol{\alpha}'].\ (\mathbf{f}\ \mathbf{x})\ \boldsymbol{\alpha}\ \mathbf{k}))\\
&= (\boldsymbol{\lambda}\,\boldsymbol{\alpha} : *.\ \boldsymbol{\lambda}\,\mathbf{k} : (B^+[e_2^\div/\mathbf{x}]) \to \boldsymbol{\alpha}. && \text{by substitution} && (58)\\
&\quad e_1^\div\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{f} : \boldsymbol{\Pi}\,\mathbf{x} : A^\div.\ B^\div.\\
&\qquad\qquad e_2^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{x} : A^+.\ (\mathbf{f}\ \mathbf{x})\ \boldsymbol{\alpha}\ \mathbf{k})))[A'^+/\boldsymbol{\alpha}']\\
&= (e_1\ e_2)^\div[A'^+/\boldsymbol{\alpha}'] && \text{by translation} && (59)
\end{aligned}
$$

□

Lemma 4.3 (Translation Preserves One-Step Reduction).

- *If* $\Gamma \vdash e : A$ *and* $e \rhd e'$ *then* $e^\div \rhd^\star \mathbf{e}'$ *and* $\mathbf{e}' \equiv e'^\div$
- *If* $\Gamma \vdash A : \kappa$ *and* $A \rhd A'$ *then* $A^+ \rhd^\star \mathbf{A}'$ *and* $\mathbf{A}' \equiv A'^+$
- *If* $\Gamma \vdash A : *$ *and* $A \rhd A'$ *then* $A^\div \rhd^\star \mathbf{A}'$ *and* $\mathbf{A}' \equiv A'^\div$

Proof. The proof is straightforward by cases on the $\rhd$ relation. We give some representative cases.

Case $x \rhd_\delta e'$ where $x = e' : A' \in \Gamma$
We must show that $x^\div \rhd_\delta \mathbf{e}$ such that $\mathbf{e} \equiv e'^\div\ \_\ \mathbf{id}$ where $x^\div = e'^\div\ \_\ \mathbf{id} : A'^+ \in \Gamma^+$, which follows by the same argument as Sub-Case Part 8 of the x case of Lemma 4.3.

Case $(\lambda\, x : \_.\ e_1)\ e_2 \rhd_\beta e_1[e_2/x]$
We must show that $((\lambda\, x : \_.\ e_1)\ e_2)^\div \rhd^\star \mathbf{e}'$ and $\mathbf{e}' \equiv (e_1[e_2/x])^\div$.

$$
\begin{aligned}
&((\lambda\, x : \_.\ e_1)\ e_2) && && (60)\\
&= (\boldsymbol{\lambda}\,\boldsymbol{\alpha}.\ \boldsymbol{\lambda}\,\mathbf{k}. && \text{by translation} && (61)\\
&\quad (\boldsymbol{\lambda}\,\boldsymbol{\alpha}.\ \boldsymbol{\lambda}\,\mathbf{k}.\ \mathbf{k}\ (\boldsymbol{\lambda}\,\mathbf{x}.\ e_1^\div))\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{f}.\ e_2^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{x}.\ (\mathbf{f}\ \mathbf{x})\ \boldsymbol{\alpha}\ \mathbf{k})))\\
&\rhd^\star (\boldsymbol{\lambda}\,\boldsymbol{\alpha}.\ \boldsymbol{\lambda}\,\mathbf{k}.\ e_2^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{x}.\ ((\boldsymbol{\lambda}\,\mathbf{x}.\ e_1^\div)\ \mathbf{x})\ \boldsymbol{\alpha}\ \mathbf{k})) && \text{by } \rhd_\beta && (62)\\
&\rhd^\star (\boldsymbol{\lambda}\,\boldsymbol{\alpha}.\ \boldsymbol{\lambda}\,\mathbf{k}.\ e_2^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\,\mathbf{x}.\ e_1^\div\ \boldsymbol{\alpha}\ \mathbf{k})) && \text{by } \rhd_\beta && (63)\\
&\equiv (\boldsymbol{\lambda}\,\boldsymbol{\alpha}.\ \boldsymbol{\lambda}\,\mathbf{k}.\ (\boldsymbol{\lambda}\,\mathbf{x}.\ e_1^\div\ \boldsymbol{\alpha}\ \mathbf{k})\ (e_2^\div\ \_\ \mathbf{id})) && \text{by [}\equiv\text{-Cont]} && (64)\\
&\rhd^\star (\boldsymbol{\lambda}\,\boldsymbol{\alpha}.\ \boldsymbol{\lambda}\,\mathbf{k}.\ (e_1^\div\ \boldsymbol{\alpha}\ \mathbf{k})[(e_2^\div\ \_\ \mathbf{id})/\mathbf{x}]) && \text{by } \rhd_\beta && (65)\\
&= (\boldsymbol{\lambda}\,\boldsymbol{\alpha}.\ \boldsymbol{\lambda}\,\mathbf{k}.\ (e_1^\div\ \boldsymbol{\alpha}\ \mathbf{k}))[(e_2^\div\ \_\ \mathbf{id})/\mathbf{x}] && \text{by substitution} && (66)\\
&\equiv e_1^\div[e_2^\div\ \_\ \mathbf{id}/\mathbf{x}] && \text{by [}\equiv\text{-}\eta\text{]} && (67)\\
&= (e_1[e_2/x])^\div && \text{by Lemma 4.2} && (68)
\end{aligned}
$$

□

Note that kinds do not take steps in the one step reduction, but can in the $\rhd^\star$ relation (since it is the compatible closure).

Lemma 4.4 (Translation Preserves $\rhd^\star$ Relation).

- *If* $\Gamma \vdash e : A$ *and* $e \rhd^\star e'$ *then* $e^\div \rhd^\star \mathbf{e}'$ *and* $\mathbf{e}' \equiv e'^\div$
- *If* $\Gamma \vdash A : \kappa$ *and* $A \rhd^\star A'$ *then* $A^+ \rhd^\star \mathbf{A}'$ *and* $\mathbf{A}' \equiv A'^+$
- *If* $\Gamma \vdash A : *$ *and* $A \rhd^\star A'$ *then* $A^\div \rhd^\star \mathbf{A}'$ *and* $\mathbf{A}' \equiv A'^\div$
- *If* $\Gamma \vdash \kappa : U$ *and* $\kappa \rhd^\star \kappa'$ *then* $\kappa^+ \rhd^\star \boldsymbol{\kappa}'$ *and* $\boldsymbol{\kappa}' \equiv \kappa'^+$

Proof. The proof is straightforward by induction on the length of the reduction sequence. The base case is trivial and the inductive case follows by Lemma 4.3 and the inductive hypothesis. □

Lemma 4.5 (CPS^v Coherence).

- *If* $e \equiv e'$ *then* $e^\div \equiv e'^\div$
- *If* $A \equiv A'$ *then* $A^+ \equiv A'^+$
- *If* $A \equiv A'$ *then* $A^\div \equiv A'^\div$
- *If* $\kappa \equiv \kappa'$ *then* $\kappa^+ \equiv \kappa'^+$

Lemma 4.6 (CPS^v is Type Preserving (Explicit Syntax)).

(1) *If* $\vdash \Gamma$ *then* $\vdash \Gamma^+$
(2) *If* $\Gamma \vdash e : A$ *then* $\Gamma^+ \vdash e^\div : A^\div$
(3) *If* $\Gamma \vdash A : \kappa$ *then* $\Gamma^+ \vdash A^+ : \kappa^+$
(4) *If* $\Gamma \vdash A : *$ *then* $\Gamma^+ \vdash A^\div : *^+$
(5) *If* $\Gamma \vdash \kappa : U$ *then* $\Gamma^+ \vdash \kappa^+ : U^+$

Proof. All cases are proven simultaneously by mutual induction on the typing derivation and well-formedness derivation. Part 4 follows easily by part 3 in every case, so we elide its proof. Most cases follow easily from the induction hypotheses.

Case [W-Assum] $\vdash \Gamma, x : A$
There are two sub-cases: either A is a type or a kind.
Sub-Case A is a type
We must show $\vdash \Gamma^+, \mathbf{x} : A^+$.
It suffices to show that $\Gamma^+ \vdash A^+ : \boldsymbol{\kappa}$, which follows by part 3 of the induction hypothesis.
Sub-Case A is a kind; similar to the previous case, except the goal follows by part 5 of the induction hypothesis.
Case [W-Def] $\vdash \Gamma, x = e : A$
We give the case for when A is a type; the case when A is a kind is similar.
We must show $\vdash \Gamma^+, \mathbf{x} = e^\div\ A^+\ \mathbf{id} : A^+$.
It suffices to show that $\Gamma^+ \vdash e^\div\ A^+\ \mathbf{id} : A^+$.
By part 2 of the induction hypothesis and definition of the translation, we know that $\Gamma^+ \vdash e^\div : \Pi\, \boldsymbol{\alpha} : *. (A^+ \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$, easily which implies the goal.
Case [Var] $\Gamma \vdash x : A$
We give the case for when A is a type; the case when A is a kind is simple since the translation on type variables is the identity.
We must show that $\Gamma^+ \vdash \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *. \boldsymbol{\lambda}\, \mathbf{k} : A^+ \to \boldsymbol{\alpha}. \mathbf{k}\ \mathbf{x} : A^\div$
By the part 1 of the induction hypothesis, we know $\Gamma^+ \vdash \mathbf{x} : A^+$, which implies the goal.
Case [App] $\Gamma \vdash e_1\ e_2 : B[e_2/x]$.
There are 4 sub-cases: e_1 can be either a term or a type, and e_2 can be either a term or a type. The interesting case is when both are terms, since this is the case most affected by the CPS translation.
Sub-Case [CPS^v_e-App], both e_1 and e_2 are terms.
We must show that

$$\Gamma^+ \vdash \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *. \boldsymbol{\lambda}\, \mathbf{k} : (B^+[(e_2^\div\ A^+\ \mathbf{id})/\mathbf{x}]) \to \boldsymbol{\alpha}. \quad e_1^\div\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \Pi\, \mathbf{x} : A^+. B^\div. e_2^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{x} : A^+. (\mathbf{f}\ \mathbf{x})\ \boldsymbol{\alpha}\ \mathbf{k})) \quad : (B[e_2/x])^\div$$

Note that,

$$(B[e_2/x]) \tag{69}$$
$$\equiv B^\div[e_2^\div\ A^+\ \mathbf{id}/\mathbf{x}] \quad \text{by Lemma 4.2} \tag{70}$$
$$\equiv \Pi\, \boldsymbol{\alpha} : *. ((B^+[e_2^\div\ A^+\ \mathbf{id}/\mathbf{x}]) \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha} \quad \text{by translation} \tag{71}$$

Hence it suffices to show that
$\Gamma^+, \boldsymbol{\alpha} : *, \mathbf{k} : (B^+[(e_2^\div\ A^+\ \mathbf{id})/\mathbf{x}]) \to \boldsymbol{\alpha} \vdash e_1^\div\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \Pi\, \mathbf{x} : A^+. B^\div. e_2^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{x} : A^+. (\mathbf{f}\ \mathbf{x})\ \boldsymbol{\alpha}\ \mathbf{k})) : \boldsymbol{\alpha}$
By part 2 of the induction hypothesis, we know that $\Gamma^+ \vdash e_1^\div : \Pi\, \boldsymbol{\alpha} : *. ((\Pi\, \mathbf{x} : A^+. B^\div) \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}$,
hence it suffices to show that
$\Gamma^+, \boldsymbol{\alpha} : *, \mathbf{k} : (B^+[(e_2^\div\ A^+\ \mathbf{id})/\mathbf{x}]) \to \boldsymbol{\alpha}, \mathbf{f} : \Pi\, \mathbf{x} : A^+. B^\div \vdash e_2^\div\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{x} : A^+. (\mathbf{f}\ \mathbf{x})\ \boldsymbol{\alpha}\ \mathbf{k}) : \boldsymbol{\alpha}$

By [T-Cont], we must show
$\Gamma^+, \alpha : *, \mathbf{k} : (B^+[(e_2^\div\ A^+\ \mathbf{id})/\mathbf{x}]) \to \alpha, \mathbf{f} : \Pi\, \mathbf{x} : A^+.\ B^\div, \mathbf{x} = e_2^\div\ A^+\ \mathbf{id}, \vdash (\mathbf{f}\ \mathbf{x})\ \alpha\ \mathbf{k} : \alpha$
Note that $\mathbf{f}\ \mathbf{x} : B^\div[\mathbf{x}/\mathbf{x}]$ and $B^\div[\mathbf{x}/\mathbf{x}] = \Pi\, \alpha : *.\ (B^+[\mathbf{x}/\mathbf{x}]) \to \alpha \to \alpha$.
But $\mathbf{k} : (B^+[(e_2^\div\ A^+\ \mathbf{id})/\mathbf{x}]) \to \alpha$.
Hence it suffices to show that $(B^+[\mathbf{x}/\mathbf{x}]) \equiv (B^+[(e_2^\div\ A^+\ \mathbf{id})/\mathbf{x}])$, which follows by δ reduction on $\mathbf{x}$ since we have $\mathbf{x} = e_2^\div\ A^+\ \mathbf{id}$ by our new typing rule [T-Cont].
Note that without our new typing rule, we would be here stuck. However, thanks to [T-Cont], we have the equality that $\mathbf{x} = e_2^\div\ A^+\ \mathbf{id}$, and we are able to complete the proof.

Sub-Case e_1 is a term but e_2 is a type A'. This case is similar to the application case of the CBN translation. It does not require the new typing rule [T-Cont], as the argument is a type, the argument is not CPS translated.

Sub-Case e_1 is a type and e_2 is a term. This case is simple; note that the translate [CPS_A^υ-AppConstr] translates the argument e_2 into $e_2^\div\ A^+\ \mathbf{id}$ since the term variable must have a value type.

Sub-Case Both e_1 and e_2 are types. This case is trivial by the induction hypothesis.

Case [Let] $\Gamma \vdash \mathsf{let}\ x = e_1 : A\ \mathsf{in}\ e_2 : B[e_1/x]$
There are 4 subcases, as in the case of application, and the proofs are nearly identical. This should be unsurprising, since our new typing rule [T-Cont] essentially gives the typing of application of a continuation the same expressive power as dependent let. We give the case for both e_1 and e_2 are terms, since this is the most interesting case.

Sub-Case [CPS_e^υ-Let]
We must show that $\Gamma^+ \vdash \lambda\, \alpha : *.\ \lambda\, \mathbf{k} : B^+[e_2^\div\ A^+\ \mathbf{id}/\mathbf{x}] \to \alpha.\quad : (B[e_2/x])^\div$
$e_1^\div\ @\ \alpha\ (\lambda\, \mathbf{x} : A^+.\ e_2^\div\ \alpha\ \mathbf{k})$
By Lemma 4.2 and the definition of the translation, it suffices to show
$\Gamma^+, \alpha : *, \mathbf{k} : B^+[e_2^\div\ A^+\ \mathbf{id}/\mathbf{x}] \to \alpha \vdash e_1^\div\ @\ \alpha\ (\lambda\, \mathbf{x} : A^+.\ e_2^\div\ \alpha\ \mathbf{k}) : \alpha$
By [T-Cont], we must show
$\Gamma^+, \alpha : *, \mathbf{k} : B^+[e_2^\div\ A^+\ \mathbf{id}/\mathbf{x}] \to \alpha, \mathbf{x} = e_1^\div\ A^+\ \mathbf{id} \vdash e_2^\div\ \alpha\ \mathbf{k} : \alpha$
Note that by the induction hypothesis, $\Gamma^+, \mathbf{x} = e_1^\div\ A^+\ \mathbf{id} \vdash e_2^\div : \Pi\, \alpha : *.\ (B^+ \to \alpha) \to \alpha$
Hence by δ reduction and [Conv],
$\Gamma^+, \mathbf{x} = e_1^\div\ A^+\ \mathbf{id} \vdash e_2^\div : \Pi\, \alpha : *.\ (B^+[(e^\div\ A^+\ \mathbf{id})/\mathbf{x}] \to \alpha) \to \alpha$ which implies the goal.

Case [Pair]
Note that the translation of pair values, [CPS_e^υ-Pair], also requires a use of the rule [T-Cont]. Since our source language allows pairs of expressions, but our target language for the CBV translation should not, we must evaluate both components of the pair before calling the continuation. However, since the type of second component depends on the value of the first component, we must apply [T-Cont] when typing the application of the continuation to the first component so that we have $\mathbf{x}_1 = e_1^\div\ A^+\ \mathbf{id}$ when typing the continuation for the second component.
The proof is similar to the application case.

Case [Snd] The proof is exactly like the case for the CBN translation.

□

Theorem 4.7 (CPS^υ is Type Preserving (PTS syntax)). *If* $\Gamma \vdash e : A$ *then* $\Gamma^+ \vdash \mathrm{cps}\ [\![e]\!] : \mathrm{cpsT}\ [\![A]\!]$.

4.2 Proof of Correctness for CPS^υ

To prove correctness of separate compilation for CPS^υ, we follow the same recipe as in Section 3.2. We use the same cross-language relation $\approx$ on values of ground type. However, note that in CBV we should only link with values, so we restrict closing substitutions γ to values and use the value translation on substitutions γ^+. The proofs follow exactly the same structure as in Section 3.2.

Theorem 4.8 (Separate Compilation Correctness). *If* $\Gamma \vdash e : A$ *where* A *is ground, and* $\gamma(e) \rhd^\star v$ *then* $\gamma^+(e^\div)\ A^+\ \mathbf{id} \rhd^\star \mathbf{v}$ *and* $v \approx \mathbf{v}$.

Corollary 4.9 (Whole-Program Compiler Correctness). *If* $\vdash e : A$ *and* $e \rhd^\star v$ *then* $e^\div\ A^+\ \mathbf{id} \rhd^\star \mathbf{v}$ *and* $v \approx \mathbf{v}$.

$\boxed{\Gamma \vdash U \rightsquigarrow^{v}_{U} \mathbf{U}}$

$$\frac{}{\Gamma \vdash * \rightsquigarrow^{v}_{U} *}\ [\mathrm{CPS}^{v}_{U}\text{-Star}] \qquad \frac{}{\Gamma \vdash \Box \rightsquigarrow^{v}_{U} \Box}\ [\mathrm{CPS}^{v}_{U}\text{-Box}]$$

$\boxed{\Gamma \vdash \kappa : U \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa}}$ Lemma 4.6 will show $\Gamma^{+} \vdash \kappa^{+} : U^{+}$

$$\frac{}{\Gamma \vdash * : \Box \rightsquigarrow^{v}_{\kappa} *}\ [\mathrm{CPS}^{v}_{\kappa}\text{-Ax}] \qquad \frac{\Gamma \vdash \kappa : U \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash \kappa' : U \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa'}}{\Gamma \vdash \Pi\, \alpha : \kappa.\, \kappa' : U \rightsquigarrow^{v}_{\kappa} \mathbf{\Pi}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \boldsymbol{\kappa'}}\ [\mathrm{CPS}^{v}_{\kappa}\text{-ProdK}]$$

$$\frac{\Gamma \vdash A : \kappa' \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x : A \vdash \kappa : U \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa}}{\Gamma \vdash \Pi\, x : A.\, \kappa : U \rightsquigarrow^{v}_{\kappa} \mathbf{\Pi}\, \mathbf{x} : \mathbf{A}.\, \boldsymbol{\kappa}}\ [\mathrm{CPS}^{v}_{\kappa}\text{-ProdA}]$$

$\boxed{\Gamma \vdash A : \kappa \rightsquigarrow^{v}_{A} \mathbf{A}}$ Lemma 4.6 will show $\Gamma^{+} \vdash A^{+} : \kappa^{+}$

$$\frac{}{\Gamma \vdash \alpha : \kappa \rightsquigarrow^{v}_{A} \boldsymbol{\alpha}}\ [\mathrm{CPS}^{v}_{A}\text{-Var}] \qquad \frac{\Gamma \vdash A : \kappa' \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x : A \vdash B : \kappa \rightsquigarrow^{v}_{A} \mathbf{B}}{\Gamma \vdash \lambda\, x : A.\, B : \Pi\, x : A.\, \kappa \rightsquigarrow^{v}_{A} \boldsymbol{\lambda}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^{v}_{A}\text{-Constr}]$$

$$\frac{\Gamma \vdash \kappa : U \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash B : \kappa' \rightsquigarrow^{v}_{A} \mathbf{B}}{\Gamma \vdash \lambda\, \alpha : \kappa.\, B : \Pi\, \alpha : \kappa.\, \kappa' \rightsquigarrow^{v}_{A} \boldsymbol{\lambda}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}}\ [\mathrm{CPS}^{v}_{A}\text{-Abs}]$$

$$\frac{\Gamma \vdash A : \Pi\, x : B.\, \kappa \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x : A \vdash B : \kappa' \rightsquigarrow^{v}_{A} \mathbf{B} \qquad \Gamma \vdash e : B \rightsquigarrow^{v}_{e} \mathbf{e}}{\Gamma \vdash A\ e : \kappa[e/x] \rightsquigarrow^{v}_{A} \mathbf{A}\ (\mathbf{e}\ \mathbf{B}\ \mathbf{id})}\ [\mathrm{CPS}^{v}_{A}\text{-AppConstr}]$$

$$\frac{\Gamma \vdash A : \Pi\, \alpha : \kappa'.\, \kappa \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma \vdash B : \kappa' \rightsquigarrow^{v}_{A} \mathbf{B}}{\Gamma \vdash A\ B : \kappa[B/\alpha] \rightsquigarrow^{v}_{A} \mathbf{A}\ \mathbf{B}}\ [\mathrm{CPS}^{v}_{A}\text{-Inst}]$$

$$\frac{\Gamma \vdash A : \kappa' \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x : A \vdash B : \kappa \rightsquigarrow^{v}_{A^{\div}} \mathbf{B}}{\Gamma \vdash \Pi\, x : A.\, B : \kappa \rightsquigarrow^{v}_{A} \mathbf{\Pi}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^{v}_{A}\text{-Prod}]$$

$$\frac{\Gamma \vdash \kappa : U' \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa} \qquad \Gamma, x : A \vdash B : U \rightsquigarrow^{v}_{A^{\div}} \mathbf{B}}{\Gamma \vdash \Pi\, \alpha : \kappa.\, B : U \rightsquigarrow^{v}_{A} \mathbf{\Pi}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}}\ [\mathrm{CPS}^{v}_{A}\text{-ProdK}]$$

$$\frac{\Gamma \vdash e : A \rightsquigarrow^{v}_{e} \mathbf{e} \qquad \Gamma \vdash A : \kappa' \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x = e : A \vdash B : \kappa \rightsquigarrow^{v}_{A} \mathbf{B}}{\Gamma \vdash \mathsf{let}\, x = e : A\, \mathsf{in}\, B : \kappa \rightsquigarrow^{v}_{A} \mathbf{let}\, \mathbf{x} = \mathbf{e}\ \mathbf{A}\ \mathbf{id} : \mathbf{A}\, \mathbf{in}\, \mathbf{B}}\ [\mathrm{CPS}^{v}_{A}\text{-Let}]$$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \vdash \Gamma \rightsquigarrow^{v} \mathbf{\Gamma} \qquad \mathbf{\Gamma} \vdash \mathbf{A} : \boldsymbol{\kappa'} \qquad \Gamma, \alpha = A : \kappa \vdash B : \_ \rightsquigarrow^{v}_{e} \mathbf{B}}{\Gamma \vdash \mathsf{let}\, \alpha = A : \kappa\, \mathsf{in}\, B : \_ \rightsquigarrow^{v}_{A} \mathbf{let}\, \boldsymbol{\alpha} = \mathbf{A} : \boldsymbol{\kappa'}\, \mathbf{in}\, \mathbf{B}}\ [\mathrm{CPS}^{v}_{A}\text{-LetK}]$$

$$\frac{\Gamma \vdash A : * \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x : A \vdash B : * \rightsquigarrow^{v}_{A} \mathbf{B}}{\Gamma \vdash \Sigma\, x : A.\, B : * \rightsquigarrow^{v}_{A} \mathbf{\Sigma}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^{v}_{A}\text{-Sigma}]$$

$$\frac{\Gamma \vdash A : \kappa' \qquad \Gamma \vdash \kappa \equiv \kappa' \qquad \Gamma \vdash A : \kappa' \rightsquigarrow^{n}_{A} \mathbf{A}}{\Gamma \vdash A : \kappa \rightsquigarrow^{n}_{A} \mathbf{A}}\ [\mathrm{CPS}^{n}_{A}\text{-Conv}]$$

$\boxed{\Gamma \vdash A : * \rightsquigarrow^{v}_{A^{\div}} \mathbf{A}}$ Lemma 4.6 will show $\Gamma^{+} \vdash A^{\div} : *^{+}$

$$\frac{\Gamma \vdash A : * \rightsquigarrow^{v}_{A} \mathbf{A}}{\Gamma \vdash A : * \rightsquigarrow^{v}_{A^{\div}} \mathbf{\Pi}\, \boldsymbol{\alpha} : *.\, (\mathbf{A} \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}}\ [\mathrm{CPS}^{v}_{A}\text{-Comp}]$$

Fig. 14. CPS^{v} of Universes, Kinds, and Types

$\boxed{\Gamma \vdash e : A \rightsquigarrow^{v}_{e} \mathbf{e}}$ Lemma 4.6 will show $\Gamma^{+} \vdash e^{\div} : A^{\div}$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^{v}_{A} \mathbf{A}}{\Gamma \vdash x : A \rightsquigarrow^{v}_{e} \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{A} \to \boldsymbol{\alpha}.\, \mathbf{k}\ \mathbf{x}}\ [\mathrm{CPS}^{v}_{e}\text{-Var}]$$

$$\frac{\Gamma \vdash A : \kappa' \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x : A \vdash B : \kappa \rightsquigarrow^{v}_{A^{\div}} \mathbf{B} \qquad \Gamma, x : A \vdash e : B \rightsquigarrow^{v}_{e} \mathbf{e}}{\Gamma \vdash \lambda\, x : A.\, e : \Pi\, x : A.\, B \rightsquigarrow^{v}_{e} \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : (\Pi\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}) \to \boldsymbol{\alpha}.\, \mathbf{k}\ (\boldsymbol{\lambda}\, \mathbf{x} : \mathbf{A}.\, \mathbf{e})}\ [\mathrm{CPS}^{v}_{e}\text{-Fun}]$$

$$\frac{\Gamma \vdash \kappa : \_ \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^{v}_{A^{\div}} \mathbf{B} \qquad \Gamma, \alpha : \kappa \vdash e : B \rightsquigarrow^{v}_{e} \mathbf{e}}{\begin{array}{l}\Gamma \vdash \lambda\, \alpha : \kappa.\, e : \Pi\, \alpha : \kappa.\, B \rightsquigarrow^{v}_{e} \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : (\Pi\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}) \to \boldsymbol{\alpha}_{ans}.\\ \qquad \mathbf{k}\ (\boldsymbol{\lambda}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{e})\end{array}}\ [\mathrm{CPS}^{v}_{e}\text{-Abs}]$$

$$\frac{\begin{array}{c}\Gamma \vdash e : \Pi\, x : A.\, B \rightsquigarrow^{v}_{e} \mathbf{e}\\ \Gamma, x : A \vdash B : \kappa \rightsquigarrow^{v}_{A^{\div}} \mathbf{B}^{\div} \qquad \Gamma, x : A \vdash B : \kappa \rightsquigarrow^{v}_{A} \mathbf{B}^{+} \qquad \Gamma \vdash e' : A \rightsquigarrow^{v}_{e} \mathbf{e}' \qquad \Gamma \vdash A : \kappa' \rightsquigarrow^{v}_{A} \mathbf{A}\end{array}}{\begin{array}{l}\Gamma \vdash e\ e' : B[e'/x] \rightsquigarrow^{v}_{e} \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : (\mathbf{B}^{+}[(\mathbf{e}'\ \mathbf{A}\ \mathrm{id})/\mathbf{x}]) \to \boldsymbol{\alpha}.\\ \qquad \mathbf{e}\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \Pi\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}^{\div}.\\ \qquad\quad \mathbf{e}'\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{x} : \mathbf{A}.\, (\mathbf{f}\ \mathbf{x})\ \boldsymbol{\alpha}\ \mathbf{k}))\end{array}}\ [\mathrm{CPS}^{v}_{e}\text{-App}]$$

$$\frac{\Gamma \vdash e : \Pi\, \alpha : \kappa.\, B \rightsquigarrow^{v}_{e} \mathbf{e} \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^{v}_{A^{\div}} \mathbf{B} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^{v}_{e} \mathbf{A}}{\begin{array}{l}\Gamma \vdash e\ A : \mathsf{let}\ x = A\ \mathsf{in}\ B \rightsquigarrow^{v}_{e} \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : (\mathbf{B}[\mathbf{A}/\boldsymbol{\alpha}]) \to \boldsymbol{\alpha}_{ans}.\\ \qquad \mathbf{e}\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \Pi\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}.\\ \qquad\quad (\mathbf{f}\ \mathbf{A})\ \boldsymbol{\alpha}_{ans}\ \mathbf{k})\end{array}}\ [\mathrm{CPS}^{v}_{e}\text{-Inst}]$$

$$\frac{\Gamma \vdash e : A \rightsquigarrow^{v}_{e} \mathbf{e} \qquad \Gamma \vdash A : \kappa' \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma \vdash B : \kappa \rightsquigarrow^{v}_{A} \mathbf{B} \qquad \Gamma, x = e : A \vdash e' : B \rightsquigarrow^{v}_{e} \mathbf{e}'}{\begin{array}{l}\Gamma \vdash \mathsf{let}\ x = e : A\ \mathsf{in}\ e' : B[e/x] \rightsquigarrow^{v}_{e} \boldsymbol{\lambda}\, \boldsymbol{\alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{B}[(\mathbf{e}\ \mathbf{A}\ \mathrm{id})/\mathbf{x}] \to \boldsymbol{\alpha}.\\ \qquad \mathbf{e}\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{x} : \mathbf{A}.\, \mathbf{e}'\ \boldsymbol{\alpha}\ \mathbf{k})\end{array}}\ [\mathrm{CPS}^{v}_{e}\text{-Let}]$$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma \vdash \kappa : U \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa} \qquad \Gamma, \alpha = A : \kappa \vdash B : \kappa' \rightsquigarrow^{v}_{A} \mathbf{B} \qquad \Gamma, \alpha = A : \kappa \vdash e : B \rightsquigarrow^{v}_{e} \mathbf{e}}{\Gamma \vdash \mathsf{let}\ \alpha = A : \kappa\ \mathsf{in}\ e : B[A/\alpha] \rightsquigarrow^{v}_{e} \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{B}[\mathbf{A}/\boldsymbol{\alpha}] \to \boldsymbol{\alpha}_{ans}.\, \mathsf{let}\ \boldsymbol{\alpha} = \mathbf{A} : \boldsymbol{\kappa}\ \mathsf{in}\ \mathbf{e}\ \boldsymbol{\alpha}_{ans}\ \mathbf{k}}\ [\mathrm{CPS}^{v}_{e}\text{-LetK}]$$

$$\frac{\Gamma \vdash e : B \rightsquigarrow^{v}_{e} \mathbf{e}}{\Gamma \vdash e : A \rightsquigarrow^{v}_{e} \mathbf{e}}\ [\mathrm{CPS}^{v}_{e}\text{-Conv}]$$

Fig. 15. CPS^{v} of Terms

$$\frac{\Gamma \vdash e_1 : A \rightsquigarrow^{v}_{e} \mathbf{e_1} \qquad \Gamma \vdash e_2 : B[e_1/x] \rightsquigarrow^{v}_{e} \mathbf{e_2} \qquad \Gamma \vdash A : * \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x : A \vdash B : * \rightsquigarrow^{v}_{A} \mathbf{B}}{\begin{array}{l}\Gamma \vdash \langle e_1, e_2 \rangle : \Sigma\, x : A.\, B \rightsquigarrow^{v}_{e} \boldsymbol{\lambda\, \alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B} \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e_1}\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{x_1} : \mathbf{A}.\, \mathbf{e_2}\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{x_2} : \mathbf{B}[(\mathbf{e_1}\ \mathbf{A}\ \mathbf{id})/\mathbf{x}]. \\ \qquad\qquad \mathbf{k}\ \langle \mathbf{x_1}, \mathbf{x_2} \rangle\ \mathbf{as}\ \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}))\end{array}}\ [\mathrm{CPS}^{v}_{e}\text{-Pair}]$$

$$\frac{\Gamma \vdash A : * \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma \vdash e : \Sigma\, x : A.\, B \rightsquigarrow^{v}_{e} \mathbf{e}}{\begin{array}{l}\Gamma \vdash \mathsf{fst}\ e : A \rightsquigarrow^{v}_{e} \boldsymbol{\lambda\, \alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{A}^{+} \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e}\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{y} : \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}.\, \mathbf{let}\ \mathbf{z} = \mathbf{fst}\ \mathbf{y}\ \mathbf{in}\ \mathbf{k}\ \mathbf{z})\end{array}}\ [\mathrm{CPS}^{v}_{e}\text{-Fst}]$$

$$\frac{\Gamma \vdash A : * \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x : A \vdash B : * \rightsquigarrow^{v}_{A} \mathbf{B} \qquad \Gamma \vdash (\mathsf{fst}\ e) : A \rightsquigarrow^{v}_{e} (\mathsf{fst}\ e)^{\dot{+}} \qquad \Gamma \vdash e : \Sigma\, x : A.\, B \rightsquigarrow^{v}_{e} \mathbf{e}}{\begin{array}{l}\Gamma \vdash \mathsf{snd}\ e : B[\mathsf{fst}\ e/x] \rightsquigarrow^{v}_{e} \boldsymbol{\lambda\, \alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{B}[((\mathsf{fst}\ e)^{\dot{+}}\ \mathbf{A}\ \mathbf{id})/\mathbf{x}] \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e}\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{y} : \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}.\, \mathbf{let}\ \mathbf{z} = \mathbf{snd}\ \mathbf{y}\ \mathbf{in}\ \mathbf{k}\ \mathbf{z})\end{array}}\ [\mathrm{CPS}^{v}_{e}\text{-Snd}]$$

$$\frac{\Gamma \vdash e_1 : A \rightsquigarrow^{v}_{e} \mathbf{e_1} \qquad \Gamma \vdash e_2 : B[e_1/x] \rightsquigarrow^{v}_{e} \mathbf{e_2} \qquad \Gamma \vdash A : * \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma, x : A \vdash B : * \rightsquigarrow^{v}_{A} \mathbf{B}}{\begin{array}{l}\Gamma \vdash \langle e_1, e_2 \rangle : \Sigma\, x : A.\, B \rightsquigarrow^{v}_{e} \boldsymbol{\lambda\, \alpha} : *.\, \boldsymbol{\lambda}\, \mathbf{k} : \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B} \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e_1}\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{x} : \mathbf{A}. \\ \qquad\qquad \mathbf{e_2}\ @\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{x_2} : \mathbf{B}.\, \mathbf{k}\ \langle \mathbf{x}, \mathbf{x_2} \rangle\ \mathbf{as}\ \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}))\end{array}}\ [\mathrm{CPS}^{v}_{e}\text{-Pair-Alt}]$$

Fig. 16. CPS^{v} of Terms (pairs)

$\boxed{\vdash \Gamma \rightsquigarrow^{v} \mathbf{\Gamma}}$ Lemma 4.6 will show $\vdash \Gamma^{+}$

$$\frac{}{\vdash \cdot \rightsquigarrow^{v} \cdot}\ [\mathrm{CPS}^{v}_{\Gamma}\text{-Empty}] \qquad \frac{\vdash \Gamma \rightsquigarrow^{v} \mathbf{\Gamma} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^{v}_{A} \mathbf{A}}{\vdash \Gamma, x : A \rightsquigarrow^{v} \mathbf{\Gamma}, \mathbf{x} : \mathbf{A}}\ [\mathrm{CPS}^{v}_{\Gamma}\text{-AssumT}]$$

$$\frac{\vdash \Gamma \rightsquigarrow^{v} \mathbf{\Gamma} \qquad \Gamma \vdash \kappa : U \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa}}{\vdash \Gamma, \alpha : \kappa \rightsquigarrow^{v} \mathbf{\Gamma}, \boldsymbol{\alpha} : \boldsymbol{\kappa}}\ [\mathrm{CPS}^{v}_{\Gamma}\text{-AssumK}] \qquad \frac{\vdash \Gamma \rightsquigarrow^{v} \mathbf{\Gamma} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma \vdash e : A \rightsquigarrow^{v}_{e} \mathbf{e}}{\vdash \Gamma, x = e : A \rightsquigarrow^{v} \mathbf{\Gamma}, \mathbf{x} = \mathbf{e}\ \mathbf{A}\ \mathbf{id} : \mathbf{A}}\ [\mathrm{CPS}^{v}_{\Gamma}\text{-Def}]$$

$$\frac{\vdash \Gamma \rightsquigarrow^{v} \mathbf{\Gamma} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^{v}_{A} \mathbf{A} \qquad \Gamma \vdash \kappa : U \rightsquigarrow^{v}_{\kappa} \boldsymbol{\kappa}}{\vdash \Gamma, \alpha = A : \kappa \rightsquigarrow^{v} \mathbf{\Gamma}, \boldsymbol{\alpha} = \mathbf{A} : \boldsymbol{\kappa}}\ [\mathrm{CPS}^{v}_{\Gamma}\text{-DefT}]$$

Fig. 17. CPS^{v} of Environments

Universes	U	::=	$\mathsf{Type}_i \mid \mathsf{Set} \mid \mathsf{Prop}$
Kinds	κ	::=	$U \mid \Pi\, \alpha : \kappa. \kappa \mid \Pi\, x : A. \kappa \mid \Sigma\, x : A. \kappa \mid \Sigma\, \alpha : \kappa. \kappa'$
Types	A, B	::=	$\alpha \mid \lambda\, x : A. B \mid \lambda\, \alpha : \kappa. B \mid A\ e \mid A\ B \mid \Pi\, x : A. B \mid \Pi\, \alpha : \kappa. B \mid \mathsf{let}\ x = e : A\ \mathsf{in}\ B$
		|	$\mathsf{let}\ \alpha = A : \kappa\ \mathsf{in}\ B \mid \Sigma\, x : A. B \mid \Sigma\, x : A. \kappa \mid \Sigma\, \alpha : \kappa. B$
		|	$\langle e, B \rangle\ \mathsf{as}\ \Sigma\, x : A. \kappa \mid \langle A, B \rangle\ \mathsf{as}\ \Sigma\, \alpha : \kappa'. \kappa \mid \langle A, e \rangle\ \mathsf{as}\ \Sigma\, \alpha : \kappa'. B \mid \mathsf{fst}\ A \mid \mathsf{snd}\ A$
Terms	e	::=	$x \mid \lambda\, x : A. e \mid \lambda\, \alpha : \kappa. e \mid e\ e \mid e\ A \mid \mathsf{let}\ x = e : A\ \mathsf{in}\ e \mid \mathsf{let}\ \alpha = A : \kappa\ \mathsf{in}\ e \mid \mathsf{fst}\ e$
		|	$\mathsf{snd}\ e \mid \langle e_1, e_2 \rangle\ \mathsf{as}\ \Sigma\, x : A. B \mid \langle e_1, B \rangle\ \mathsf{as}\ \Sigma\, x : A. \kappa \mid \langle A, e_2 \rangle\ \mathsf{as}\ \Sigma\, \alpha : \kappa. B$
Local Environment	Γ	::=	$\cdot \mid \Gamma, x : A \mid \Gamma, x = e : A, \mid \Gamma, \alpha : \kappa \mid \Gamma, \alpha = A : \kappa$

Fig. 18. ECC Explicit Syntax

$\boxed{\Gamma \vdash \kappa : U \rightsquigarrow^n_\kappa \boldsymbol{\kappa}}$

$$\frac{}{\Gamma \vdash U : U' \rightsquigarrow^n_\kappa \mathbf{U}}\ [\mathrm{CPS}^n_\kappa\text{-Ax}] \qquad \frac{\Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash \kappa' : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa'}}{\Gamma \vdash \Pi\, \alpha : \kappa. \kappa' : U \rightsquigarrow^n_\kappa \mathbf{\Pi}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}. \boldsymbol{\kappa'}}\ [\mathrm{CPS}^n_\kappa\text{-ProdK}]$$

$$\frac{\Gamma \vdash A : \_ \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma, x : A \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa}}{\Gamma \vdash \Pi\, x : A. \kappa : U \rightsquigarrow^n_\kappa \mathbf{\Pi}\, \mathbf{x} : \mathbf{A}. \boldsymbol{\kappa}}\ [\mathrm{CPS}^n_\kappa\text{-ProdA}]$$

$$\frac{\Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash \kappa' : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa'}}{\Gamma \vdash \Sigma\, \alpha : \kappa. \kappa' : U \rightsquigarrow^n_\kappa \mathbf{\Sigma}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}. \boldsymbol{\kappa'}}\ [\mathrm{CPS}^n_\kappa\text{-SigmaK}]$$

$$\frac{\Gamma \vdash A : \_ \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma, x : A \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa}}{\Gamma \vdash \Sigma\, x : A. \kappa : U \rightsquigarrow^n_\kappa \mathbf{\Sigma}\, \mathbf{x} : \mathbf{A}. \boldsymbol{\kappa}}\ [\mathrm{CPS}^n_\kappa\text{-SigmaA}_1]$$

$$\frac{\Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^n_A \mathbf{B}}{\Gamma \vdash \Sigma\, \alpha : \kappa. B : U \rightsquigarrow^n_\kappa \mathbf{\Sigma}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}. \mathbf{B}}\ [\mathrm{CPS}^n_\kappa\text{-SigmaA}_2]$$

Fig. 19. CBN CPS Translation of Kinds

5 EXTENDING THE CALL-BY-NAME CPS TRANSLATION TO THE CALCULUS OF INDUCTIVE CONSTRUCTIONS (CIC)

CIC introduces additional challenges to the translation. For one, the infinite universe hierarchy creates a problem with the locally polymorphic answer type.

First, It's not clear what the type of the answer types $\boldsymbol{\alpha}$ should be, since that is determined by the context. One approach, which we've sketched here, is to use a fixed answer universe. This introduces several challenges of its own. As this universe is really determined by the calling context, we should probably use universe polymorphism instead. However, that may prove to introduces other challenges.

Second, we need to accommodate universes constructing functions, like $\boldsymbol{\lambda}\, \mathbf{x} : \mathbf{Type}_1. \mathbf{Set}$. It seems that this is a type-level function, and that universe can be both types and kinds. This does not seem difficult to solve, but we must be check the details carefully.

Third and finally, we need to accommodate type-level pairs. We've sketch the translations of type-level pairs, but have yet to solve all the issues. In CC, some of these combination are invalid, but become valid with the infinite hierarchy of universes. It's not clear what some of the translations, such as a pair of a term and a universe, should be.

$\boxed{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A}}$

$$\frac{}{\Gamma \vdash \alpha : \kappa \rightsquigarrow^n_A \boldsymbol{\alpha}}\ [\mathrm{CPS}^n_A\text{-Var}] \qquad \frac{\Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \lambda\, x : A.\, B : \kappa \rightsquigarrow^n_A \boldsymbol{\lambda}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Constr}]$$

$$\frac{\Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \lambda\, \alpha : \kappa.\, B : \kappa' \rightsquigarrow^n_A \boldsymbol{\lambda}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Abs}]$$

$$\frac{\Gamma \vdash A : \Pi\, x : B.\, \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash e : B \rightsquigarrow^n_e \mathbf{e}}{\Gamma \vdash A\ e : \kappa[e/x] \rightsquigarrow^n_A \mathbf{A}\ \mathbf{e}}\ [\mathrm{CPS}^n_A\text{-AppConstr}]$$

$$\frac{\Gamma \vdash A : \Pi\, \alpha : \kappa'.\, \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash B : \kappa' \rightsquigarrow^n_A \mathbf{B}}{\Gamma \vdash A\ B : \kappa[B/\alpha] \rightsquigarrow^n_A \mathbf{A}\ \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Inst}]$$

$$\frac{\Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \Pi\, x : A.\, B : \kappa \rightsquigarrow^n_A \boldsymbol{\Pi}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Prod}]$$

$$\frac{\Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, x : A \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \Pi\, \alpha : \kappa.\, B : \_ \rightsquigarrow^n_A \boldsymbol{\Pi}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-ProdK}]$$

$$\frac{\Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x = e : A \vdash B : \_ \rightsquigarrow^n_A \mathbf{B}}{\Gamma \vdash \mathsf{let}\ x = e : A\ \mathsf{in}\ B : \_ \rightsquigarrow^n_A \mathsf{let}\ \mathbf{x} = \mathbf{e} : \mathbf{A}\ \mathsf{in}\ \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Let}]$$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \vdash \Gamma \rightsquigarrow^n \boldsymbol{\Gamma} \qquad \boldsymbol{\Gamma} \vdash \mathbf{A} : \boldsymbol{\kappa}' \qquad \Gamma, \alpha = A : \kappa \vdash B : \_ \rightsquigarrow^n_e \mathbf{B}}{\Gamma \vdash \mathsf{let}\ \alpha = A : \kappa\ \mathsf{in}\ B : \_ \rightsquigarrow^n_A \mathsf{let}\ \boldsymbol{\alpha} = \mathbf{A} : \boldsymbol{\kappa}'\ \mathsf{in}\ \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-LetK}]$$

$$\frac{\Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \Sigma\, x : A.\, B : \_ \rightsquigarrow^n_A \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-Sigma}]$$

$\boxed{\Gamma \vdash A : \kappa \rightsquigarrow^n_{A^\div} \mathbf{A}}$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A}}{\Gamma \vdash A : \kappa \rightsquigarrow^n_{A^\div} \boldsymbol{\Pi}\, \boldsymbol{\alpha} : \mathbf{U}_i.\, (\mathbf{A} \to \boldsymbol{\alpha}) \to \boldsymbol{\alpha}}\ [\mathrm{CPS}^n_A\text{-Comp}]$$

Fig. 20. CBN CPS Translation of Types

$$\frac{\Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \Sigma\, \alpha : \kappa.\, B : \_ \rightsquigarrow^n_A \Sigma\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}}\ [\mathrm{CPS}^n_A\text{-SigmaK}_1]$$

$$\frac{\Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa}}{\Gamma \vdash \Sigma\, x : A.\, \kappa : \_ \rightsquigarrow^n_A \Sigma\, \mathbf{x} : \mathbf{A}.\, \boldsymbol{\kappa}}\ [\mathrm{CPS}^n_A\text{-SigmaK}_2]$$

$$\frac{\Gamma \vdash A : \_ \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash B : \_ \rightsquigarrow^n_A \mathbf{B} \qquad \vdash \Gamma \rightsquigarrow^n \boldsymbol{\Gamma} \qquad \boldsymbol{\Gamma} \vdash \mathbf{A} : \boldsymbol{\kappa}'_1 \qquad \boldsymbol{\Gamma} \vdash \mathbf{B} : \boldsymbol{\kappa}'_2[\mathbf{A}/\boldsymbol{\alpha}]}{\Gamma \vdash \langle A, B\rangle \text{ as } \Sigma\, \alpha : \kappa_1.\, \kappa_2 : \_ \rightsquigarrow^n_A \langle \mathbf{A}, \mathbf{B}\rangle \text{ as } \Sigma\, \boldsymbol{\alpha} : \boldsymbol{\kappa}'_1.\, \boldsymbol{\kappa}'_2}\ [\mathrm{CPS}^n_A\text{-PairK}]$$

$$\frac{\Gamma \vdash e : \_ \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : \_ \rightsquigarrow^n_A \mathbf{B} \qquad \vdash \Gamma \rightsquigarrow^n \boldsymbol{\Gamma} \qquad \boldsymbol{\Gamma} \vdash \mathbf{B} : \boldsymbol{\kappa}'[\mathbf{e}/\mathbf{x}]}{\Gamma \vdash \langle e, B\rangle \text{ as } \Sigma\, x : A.\, \kappa : \_ \rightsquigarrow^n_A \langle \mathbf{e}, \mathbf{B}\rangle \text{ as } \Sigma\, \mathbf{x} : \mathbf{A}.\, \boldsymbol{\kappa}'}\ [\mathrm{CPS}^n_A\text{-PairA}_1]$$

$$\frac{\Gamma \vdash B : \_ \rightsquigarrow^n_A \mathbf{B} \qquad \Gamma \vdash e : \_ \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma, \alpha : \kappa \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \vdash \Gamma \rightsquigarrow^n \boldsymbol{\Gamma} \qquad \boldsymbol{\Gamma} \vdash \mathbf{B} : \boldsymbol{\kappa}'}{\Gamma \vdash \langle B, e\rangle \text{ as } \Sigma\, \alpha : \kappa.\, A : \_ \rightsquigarrow^n_A \langle \mathbf{B}, \mathbf{e}\rangle \text{ as } \Sigma\, \boldsymbol{\alpha} : \boldsymbol{\kappa}'.\, \mathbf{A}}\ [\mathrm{CPS}^n_A\text{-PairA}_2]$$

$$\frac{\Gamma \vdash A : \Sigma\, \alpha : \kappa_1.\, \kappa_2 \rightsquigarrow^n_A \mathbf{A}}{\Gamma \vdash \mathrm{fst}\, A : \kappa_1 \rightsquigarrow^n_A \mathbf{fst}\, \mathbf{A}}\ [\mathrm{CPS}^n_A\text{-Fst}_1] \qquad \frac{\Gamma \vdash A : \Sigma\, \alpha : \kappa_1.\, B \rightsquigarrow^n_A \mathbf{A}}{\Gamma \vdash \mathrm{fst}\, A : \kappa_1 \rightsquigarrow^n_A \mathbf{fst}\, \mathbf{A}}\ [\mathrm{CPS}^n_A\text{-Fst}_2]$$

$$\frac{\Gamma \vdash A : \Sigma\, \alpha : \kappa_1.\, \kappa_2 \rightsquigarrow^n_A \mathbf{A}}{\Gamma \vdash \mathrm{snd}\, A : \kappa_2[\mathrm{fst}\, A/\alpha] \rightsquigarrow^n_A \mathbf{snd}\, \mathbf{A}}\ [\mathrm{CPS}^n_A\text{-Snd}_1] \qquad \frac{\Gamma \vdash A : \Sigma\, x : B.\, \kappa_2 \rightsquigarrow^n_A \mathbf{A}}{\Gamma \vdash \mathrm{snd}\, A : \kappa_2[\mathrm{fst}\, A/x] \rightsquigarrow^n_A \mathbf{snd}\, \mathbf{A}}\ [\mathrm{CPS}^n_A\text{-Snd}_2]$$

Fig. 21. CBN CPS Translation of Type-Level Pairs

$\boxed{\Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{e}}$

$$\frac{}{\Gamma \vdash x : A \rightsquigarrow^n_e \mathbf{x}}\ [\mathrm{CPS}^n_e\text{-Var}] \qquad \frac{\Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B} \qquad \Gamma, x : A \vdash e : B \rightsquigarrow^n_e \mathbf{e}}{\begin{array}{l}\Gamma \vdash \lambda\, x : A.\, e : \Pi\, x : A.\, B \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : (\boldsymbol{\Pi}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}) \to \boldsymbol{\alpha}. \\ \qquad \mathbf{k}\ (\boldsymbol{\lambda}\, \mathbf{x} : \mathbf{A}.\, \mathbf{e})\end{array}}\ [\mathrm{CPS}^n_e\text{-Fun}]$$

$$\frac{\Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B} \qquad \Gamma, \alpha : \kappa \vdash e : B \rightsquigarrow^n_e \mathbf{e}}{\begin{array}{l}\Gamma \vdash \lambda\, \alpha : \kappa.\, e : \Pi\, \alpha : \kappa.\, B \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : (\boldsymbol{\Pi}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}) \to \boldsymbol{\alpha}_{ans}. \\ \qquad \mathbf{k}\ (\boldsymbol{\lambda}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{e})\end{array}}\ [\mathrm{CPS}^n_e\text{-Abs}]$$

$$\frac{\Gamma \vdash e : \Pi\, x : A'.\, B \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma, x : A \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B} \qquad \Gamma \vdash e' : A' \rightsquigarrow^n_e \mathbf{e}'}{\begin{array}{l}\Gamma \vdash e\ e' : B[e'/x] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : (\mathbf{B}[\mathbf{e}'/\mathbf{x}]) \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e}\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \boldsymbol{\Pi}\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}. \\ \qquad\quad (\mathbf{f}\ \mathbf{e}')\ \boldsymbol{\alpha}\ \mathbf{k})\end{array}}\ [\mathrm{CPS}^n_e\text{-App}]$$

$$\frac{\Gamma \vdash e : \Pi\, \alpha : \kappa.\, B \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B} \qquad \Gamma \vdash A : \kappa \rightsquigarrow^n_e \mathbf{A}}{\begin{array}{l}\Gamma \vdash e\ A : B[A/\alpha] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : (\mathbf{B}[\mathbf{A}/\boldsymbol{\alpha}]) \to \boldsymbol{\alpha}_{ans}. \\ \qquad \mathbf{e}\ \boldsymbol{\alpha}\ (\boldsymbol{\lambda}\, \mathbf{f} : \boldsymbol{\Pi}\, \boldsymbol{\alpha} : \boldsymbol{\kappa}.\, \mathbf{B}. \\ \qquad\quad (\mathbf{f}\ \mathbf{A})\ \boldsymbol{\alpha}_{ans}\ \mathbf{k})\end{array}}\ [\mathrm{CPS}^n_e\text{-Inst}]$$

$$\frac{\Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{e} \qquad \Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x = e : A \vdash e' : B \rightsquigarrow^n_e \mathbf{e}'}{\Gamma \vdash \mathsf{let}\, x = e : A\, \mathsf{in}\, e' : B[e/x] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{B}[\mathbf{e}/\mathbf{x}] \to \boldsymbol{\alpha}.\, \mathsf{let}\, \mathbf{x} = \mathbf{e} : \mathbf{A}\, \mathsf{in}\, \mathbf{e}'\ \boldsymbol{\alpha}\ \mathbf{k}}\ [\mathrm{CPS}^n_e\text{-Let}]$$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa} \qquad \Gamma, \alpha = A : \kappa \vdash e : \_ \rightsquigarrow^n_e \mathbf{e}}{\Gamma \vdash \mathsf{let}\, \alpha = A : \kappa\, \mathsf{in}\, e : B[A/\alpha] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{B}[\mathbf{A}/\mathbf{x}] \to \boldsymbol{\alpha}_{ans}.\, \mathsf{let}\, \boldsymbol{\alpha} = \mathbf{A} : \boldsymbol{\kappa}\, \mathsf{in}\, \mathbf{e}\ \boldsymbol{\alpha}_{ans}\ \mathbf{k}}\ [\mathrm{CPS}^n_e\text{-LetK}]$$

Fig. 22. CBN CPS Translation of Terms

$$\frac{\Gamma \vdash e_1 : A \rightsquigarrow^n_e \mathbf{e}_1 \qquad \Gamma \vdash e_2 : B[e_1/x] \rightsquigarrow^n_e \mathbf{e}_2 \qquad \Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma, x : A \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B}}{\Gamma \vdash \langle e_1, e_2 \rangle : \Sigma\, x : A.\, B \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B} \to \boldsymbol{\alpha}.\, \mathbf{k}\ \langle \mathbf{e}_1, \mathbf{e}_2 \rangle\ \mathsf{as}\, \Sigma\, \mathbf{x} : \mathbf{A}.\, \mathbf{B}}\ [\mathrm{CPS}^n_e\text{-Pair}]$$

$$\frac{\Gamma \vdash A : \kappa \rightsquigarrow^n_A \mathbf{A} \qquad \Gamma \vdash e_2 : B[A/x] \rightsquigarrow^n_e \mathbf{e}_2 \qquad \Gamma, \alpha : \kappa \vdash B : \_ \rightsquigarrow^n_{A^\div} \mathbf{B} \qquad \vdash \Gamma \rightsquigarrow^n \Gamma \qquad \Gamma \vdash \mathbf{A} : \boldsymbol{\kappa}'}{\Gamma \vdash \langle A, e_2 \rangle : \Sigma\, \alpha : \kappa.\, B \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : \Sigma\, \boldsymbol{\alpha} : \boldsymbol{\kappa}'.\, \mathbf{B} \to \boldsymbol{\alpha}.\, \mathbf{k}\ \langle \mathbf{A}, \mathbf{e}_2 \rangle\ \mathsf{as}\, \Sigma\, \boldsymbol{\alpha} : \boldsymbol{\kappa}'.\, \mathbf{B}}\ [\mathrm{CPS}^n_e\text{-PairA}_1]$$

$$\frac{\Gamma \vdash e_1 : A \rightsquigarrow^n_e \mathbf{e}_1 \qquad \Gamma \vdash B : \kappa \rightsquigarrow^n_A \mathbf{B} \qquad \Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \vdash \Gamma \rightsquigarrow^n \Gamma \qquad \Gamma \vdash \mathbf{B} : \boldsymbol{\kappa}'[\mathbf{e}_1/\mathbf{x}]}{\Gamma \vdash \langle e_1, B \rangle : \Sigma\, x : A.\, \kappa \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha}_{ans} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : \Sigma\, \mathbf{x} : \mathbf{A}.\, \boldsymbol{\kappa}' \to \boldsymbol{\alpha}.\, \mathbf{k}\ \langle \mathbf{e}_1, \mathbf{B} \rangle\ \mathsf{as}\, \Sigma\, \mathbf{x} : \mathbf{A}.\, \boldsymbol{\kappa}'}\ [\mathrm{CPS}^n_e\text{-PairA}_2]$$

$$\frac{\Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A}^\div \qquad \Gamma \vdash A : \_ \rightsquigarrow^n_A \mathbf{A}^+ \qquad \Gamma \vdash e : \Sigma\, x : A.\, B \rightsquigarrow^n_e \mathbf{e}}{\begin{array}{l}\Gamma \vdash \mathsf{fst}\, e : A \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{A}^+ \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e}\ \boldsymbol{\alpha}\ \boldsymbol{\lambda}\, \mathbf{y} : \Sigma\, \mathbf{x} : \mathbf{A}^\div.\, \mathbf{B}^\div. \\ \qquad\quad \mathsf{let}\, \mathbf{z} = \mathsf{fst}\, \mathbf{y}\, \mathsf{in}\, \mathbf{z}\ \boldsymbol{\alpha}\ \mathbf{k}\end{array}}\ [\mathrm{CPS}^n_e\text{-Fst}]$$

$$\frac{\Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A}^\div \qquad \Gamma, x : A \vdash B : \_ \rightsquigarrow^n_A \mathbf{B}^+ \qquad \Gamma \vdash (\mathsf{fst}\, e) : A \rightsquigarrow^n_e (\mathsf{fst}\, e)^\div \qquad \Gamma \vdash e : \Sigma\, x : A.\, B \rightsquigarrow^n_e \mathbf{e}}{\begin{array}{l}\Gamma \vdash \mathsf{snd}\, e : B[(\mathsf{fst}\, e)/x] \rightsquigarrow^n_e \boldsymbol{\lambda}\, \boldsymbol{\alpha} : \mathbf{U}_i.\, \boldsymbol{\lambda}\, \mathbf{k} : \mathbf{B}^+[(\mathsf{fst}\, e)^\div/\mathbf{x}] \to \boldsymbol{\alpha}. \\ \qquad \mathbf{e}\ \boldsymbol{\alpha}\ \boldsymbol{\lambda}\, \mathbf{y} : \Sigma\, \mathbf{x} : \mathbf{A}^\div.\, \mathbf{B}^\div. \\ \qquad\quad \mathsf{let}\, \mathbf{z} = \mathsf{snd}\, \mathbf{y}\, \mathsf{in}\, \mathbf{z}\ \boldsymbol{\alpha}\ \mathbf{k}\end{array}}\ [\mathrm{CPS}^n_e\text{-Snd}]$$

Fig. 23. CBN CPS Translation of Terms (pairs)

$\boxed{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma}}$

$$\frac{}{\vdash \cdot \rightsquigarrow^n \cdot}\ [\mathrm{CPS}^n_\Gamma\text{-}\textsc{Empty}] \qquad \frac{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma} \qquad \Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A}}{\vdash \Gamma, x : A \rightsquigarrow^n \mathbf{\Gamma}, \mathbf{x} : \mathbf{A}}\ [\mathrm{CPS}^n_\Gamma\text{-}\textsc{AssumT}]$$

$$\frac{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma} \qquad \Gamma \vdash \kappa : \_ \rightsquigarrow^n_\kappa \boldsymbol{\kappa}}{\vdash \Gamma, \alpha : \kappa \rightsquigarrow^n \mathbf{\Gamma}, \boldsymbol{\alpha} : \boldsymbol{\kappa}}\ [\mathrm{CPS}^n_\Gamma\text{-}\textsc{AssumK}] \qquad \frac{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma} \qquad \Gamma \vdash A : \_ \rightsquigarrow^n_{A^\div} \mathbf{A} \qquad \Gamma \vdash e : A \rightsquigarrow^n_e \mathbf{e}}{\vdash \Gamma, x = e : A \rightsquigarrow^n \mathbf{\Gamma}, \mathbf{x} = \mathbf{e} : \mathbf{A}}\ [\mathrm{CPS}^n_\Gamma\text{-}\textsc{Def}]$$

$$\frac{\vdash \Gamma \rightsquigarrow^n \mathbf{\Gamma} \qquad \Gamma \vdash A : \_ \rightsquigarrow^n_A \mathbf{A} \qquad \mathbf{\Gamma} \vdash \mathbf{A} : \boldsymbol{\kappa'}}{\vdash \Gamma, \alpha = A : \kappa \rightsquigarrow^n \mathbf{\Gamma}, \boldsymbol{\alpha} = \mathbf{A} : \boldsymbol{\kappa'}}\ [\mathrm{CPS}^n_\Gamma\text{-}\textsc{DefT}]$$

Fig. 24. CBN CPS Translation of Environments

REFERENCES

Amal Ahmed and Matthias Blume. 2011. An Equivalence-preserving CPS Translation Via Multi-language Semantics. In *International Conference on Functional Programming (ICFP)*. https://doi.org/10.1145/2034773.2034830

Gilles Barthe, John Hatcliff, and Morten Heine B. Sørensen. 1999. CPS Translations and Applications: The Cube and Beyond. *Higher-Order and Symbolic Computation* 12, 2 (Sept. 1999). https://doi.org/10.1023/a:1010000206149

Gilles Barthe and Tarmo Uustalu. 2002. CPS Translating Inductive and Coinductive Types. In *Workshop on Partial Evaluation and Semantics-based Program Manipulation (PEPM)*. https://doi.org/10.1145/509799.503043

Jean-philippe Bernardy, Patrik Jansson, and Ross Paterson. 2012. Proofs for Free: Parametricity for Dependent Types. *Journal of Functional Programming* 22, 02 (March 2012). https://doi.org/10.1017/S0956796812000056

Simon Boulier, Pierre-marie Pédrot, and Nicolas Tabareau. 2017. The Next 700 Syntactical Models of Type Theory. In *Conference on Certified Programs and Proofs (CPP)*. https://doi.org/10.1145/3018610.3018620

William J. Bowman, Youyou Cong, Nick Rioux, and Amal Ahmed. 2017. Type-Preserving CPS Translation of Σ and Π Types Is Not Not Possible (Supplementary Materials). (Oct. 2017). https://williamjbowman.com/resources/cps-sigma.tar.gz

Thierry Coquand. 1986. An Analysis of Girard's Paradox. In *Symposium on Logic in Computer Science (LICS)*. https://hal.inria.fr/inria-00076023

Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. 1993. The Essence of Compiling with Continuations. In *International Conference on Programming Language Design and Implementation (PLDI)*. https://doi.org/10.1145/155090.155113

James G. Hook and Douglas J. Howe. 1986. *Impredicative Strong Existential Equivalent to Type:type.* Technical Report. Cornell University. http://hdl.handle.net/1813/6600

Jeehoon Kang, Yoonseung Kim, Chung-kil Hur, Derek Dreyer, and Viktor Vafeiadis. 2016. Lightweight Verification of Separate Compilation. In *Symposium on Principles of Programming Languages (POPL)*. https://doi.org/10.1145/2837614.2837642

Chantal Keller and Marc Lasson. 2012. Parametricity in an Impredicative Sort. In *International Workshop on Computer Science Logic (CSL)*. https://hal.inria.fr/hal-00730913

Georg Neis, Chung-kil Hur, Jan-oliver Kaiser, Craig Mclaughlin, Derek Dreyer, and Viktor Vafeiadis. 2015. Pilsner: A Compositionally Verified Compiler for a Higher-order Imperative Language. In *International Conference on Functional Programming (ICFP)*. https://doi.org/10.1145/2784731.2784764

Max S. New, William J. Bowman, and Amal Ahmed. 2016. Fully Abstract Compilation Via Universal Embedding. In *International Conference on Functional Programming (ICFP)*. https://doi.org/10.1145/2951913.2951941

James T. Perconti and Amal Ahmed. 2014. Verifying an Open Compiler Using Multi-language Semantics. In *European Symposium on Programming (ESOP)*. https://doi.org/10.1007/978-3-642-54833-8_8

The Coq Development Team. 2017. The Coq Proof Assistant Reference Manual. https://coq.inria.fr/doc/Reference-Manual006.html