Interconnecting Between CPS Terms and Non-CPS Terms *

Jung-taek Kim and Kwangkeun Yi †
ROPAS ‡
Department of Computer Science
KAIST §

Abstract

We present a type-based partial CPS transformation (CPS-transforming only a sub-part of a program) and its correctness proof. A program's sub-parts which need to be CPS-transformed are initially annotated as such. Partial CPS-transformation scans the expressions and selectively apply the transformation depending on the annotation. Wherever an interface between CPS-transformed and direct-style expressions is needed, proper conversion code is padded based on expression types. The correctness of the partial CPS transformation is proven similarly to that of Plotkin's simulation theorem[Plo75].

1 This Work

Is it possible to CPS-transform only a sub-part of a program, leaving others in direct-style? Such partial CPS-transformation is needed, for example, to link CPS-transformed modules with non-CPS modules or to minimize the performance overhead by the transformation[KYD98].

In this article we present a type-based partial CPS transformation. The partial CPS transformation transforms only parts of a program and connects the results to the rest, direct-style parts. A program's sub-parts which need to be CPS-transformed is initially annotated as such. Partial CPS-transformation scans the expressions and selectively apply the transformation depending on the annotation. Wherever an interface between CPS-transformed and direct-style expressions is needed, proper conversion code is padded based on expression types.

Let's consider an example code:

$$((\lambda x.x)\,1) + 1$$

Assume that we want to CPS-transform only the function part - $\lambda x.x$. A CPS transformed version of the function part is: $\lambda K.K\ (\lambda x.\lambda K.K\ x)$. To use the CPS-transformed function inside the non-CPS context $([]\,1) + 1$, we wrap the CPS-function position with a conversion code as follows:

$$\lambda y.([](\lambda x.x)\,y\ (\lambda x.x))$$

where the first identity continuation is for the function expression itself and the second identity continuation is for the function's result. By merging all components, we get a partially CPS transformed code :

$$((\lambda y.(\lambda K.K\ (\lambda x.\lambda K.K\ x))\ (\lambda x.x)\ y\ (\lambda x.x))1) + 1.$$

2 Source Language and CPS Transformation

We consider ML's core language with annotations. Its abstract syntax reads as follows. (κ denotes a data constructor)

$$\begin{array}{llcl}
\textit{expressions} & e & ::= & t^a \\
 & t & ::= & \mathtt{1} \\
 & & \mid & \lambda x.e \\
 & & \mid & \mathtt{fix}\ f\ \lambda x.e \\
 & & \mid & x \\
 & & \mid & e_1 @ e_2 \\
 & & \mid & \mathtt{con}\ \kappa\ e \\
 & & \mid & \mathtt{decon}\ e \\
 & & \mid & \mathtt{case}\ e_1\ \kappa\ e_2\ e_3 \\
\textit{annotation} & a & ::= & N \mid C
\end{array}$$

Annotation C denotes that the expression needs CPS transformation and N otherwise. Value $\kappa\cdot v$ is constructed by "$\mathtt{con}\,\kappa\,e$" where evaluating e yields v. Symmetrically, value $\kappa\cdot v$ denoted by e is deconstructed into v by evaluating "$\mathtt{decon}\,e$". Evaluating the case expression "$\mathtt{case}\,e_1\ \kappa\ e_2\ e_3$" yields the value of e_2 if the value of e_1 is $\kappa\cdot v$; otherwise, it yields the value of e_3.

We assume that the source programs are type-checked before processing. Types range over τ :

$$\begin{array}{llcl}
\textit{types} & \tau & ::= & \iota \\
 & & \mid & \tau \to \tau \\
 & & \mid & \alpha \\
 & & \mid & \mu\,\alpha.\tau_{\kappa_1}+\tau_{\kappa_2}
\end{array}$$

ι denotes basic types, $\tau \to \tau$ denotes function types and $\mu\,\alpha.\tau_{\kappa_1}+\tau_{\kappa_2}$ denotes user-defined recursive types where data constructor κ_1 has type $\tau_{\kappa_1} \to (\mu\,\alpha.\tau_{\kappa_1}+\tau_{\kappa_2})$ and κ_2 has type $\tau_{\kappa_2} \to (\mu\,\alpha.\tau_{\kappa_1}+\tau_{\kappa_2})$. And we use the conventional mono-morphic type system for the ML core language.

Figure 1 shows the conventional call-by-value CPS transformation [Plo75] which transforms the whole program(we ignore the annotations).

*This work is supported by Creative Research Initiatives of Korean Ministry of Science and Technology.

† email:{judaigi,kwang}@ropas.kaist.ac.kr
‡ Research On Program Analysis System (http://ropas.kaist.ac.kr)
§ Korea Advanced Institute of Science and Technology (http://www.kaist.ac.kr)

$$
\begin{aligned}
[\![1]\!] &= \lambda K.K@1\\
[\![x]\!] &= \lambda K.K@x\\
[\![\texttt{con}\,\kappa\, e]\!] &= \lambda K.[\![e]\!]@(\lambda v.K@(\texttt{con}\,\kappa\, v))\\
[\![\texttt{decon}\, e]\!] &= \lambda K.[\![e]\!]@(\lambda v.K@(\texttt{decon}\, v))\\
[\![\lambda x.e]\!] &= \lambda K.K@(\lambda x.[\![e]\!])\\
[\![\texttt{fix}\, f\, \lambda x.e]\!] &= \lambda K.K@(\texttt{fix}\, f\, \lambda x.[\![e]\!])\\
[\![e_1@e_2]\!] &= \lambda K.[\![e_1]\!]@(\lambda f.[\![e_2]\!]\; (\lambda v.f@v@K))\\
[\![\texttt{case}\, e_1\, \kappa\, e_2\, e_3]\!] &= \\
&\lambda K.[\![e_1]\!]@(\lambda v.\texttt{case}\; v\; \kappa\; ([\![e_2]\!]@K)\; ([\![e_3]\!]@K))
\end{aligned}
$$

Figure 1: Base CPS transformation $[\![\cdot]\!]$

3 Conversion Functions and Partial CPS Transformation

Partial CPS transformation receives an annotated program and outputs a partially CPS transformed version of it. C-annotated expressions are targets for CPS transformation.

Not all annotations lead to correct translations. If a variable occurs at multiple positions, they all should have the same annotation. In case that a variable is bound, its annotation should match with that of its binding expression. Annotated programs satisfying these constraints are called consistent:

Definition 1 (Consistency of Annotations) *Annotations in an expression e is called consistent whenever the following holds :*

- *If x^a and $x^{a'}$ are free variables in e then a and a' are equal.*
- *If x^a is bounded by a sub-expression $(\lambda x.e)^{a'}$, $(\texttt{fix} f \lambda x.e)^{a'}$, or $(\texttt{fix}\, x\, \lambda y.e)^{a'}$ in e then a and a' are equal.*

In partial CPS transformation, we need conversion codes that interface CPS transformed expressions with non-CPS contexts and vice versa. We have to consider three cases according to the types of the expressions to be converted:

- For expressions of base type ι :

  No conversion is necessary. CPS version of a base-typed expression just passes to the continuation the value computed by the non-CPS version.

- For expressions of function type $\tau_1 \to \tau_2$:

  We eta-expand them to set up conversion codes around their arguments and results. For example, let's assume a CPS function is used in a non-CPS part. Then its argument should be a non-CPS value. So, we convert the argument to a CPS expression and apply it to the function. Because function's result is a CPS expression, we convert the result to a non-CPS expression.

- For expressions of user-defined type $\mu\, \alpha.\tau_{\kappa_1}+\tau_{\kappa_2}$:

  An expression of this user-defined type will be evaluated to $\kappa_1 \cdot v$ or $\kappa_2 \cdot v$. In order to convert the argument value v, because the argument value v can also be $\kappa_1 \cdot v'$ or $\kappa_2 \cdot v'$ (due to recursive type), we should make a recursive conversion function. This conversion function deconstructs the value, converts the argument value recursively, and reconstructs.

The conversion codes are generated by two mutually recursive functions `c2n` (CPS to non-CPS) and `n2c` (non-CPS to CPS) which are inductively defined over the types as follows:

- The `c2n` function, given the type of a non-CPS expression, generates a conversion code that converts the values of the type into their CPS versions, to be used inside a CPS context:

$$
\begin{aligned}
\texttt{c2n}(\iota) &= \lambda v.v\\
\texttt{c2n}(\tau_1 \to \tau_2) &= \lambda v.\lambda x.v@(\texttt{n2c}(\tau_1)@x)@\texttt{c2n}(\tau_2)\\
\texttt{c2n}(\mu\, \alpha.\tau_{\kappa_1}+\tau_{\kappa_2}) &= \\
&(\texttt{fix}\, f\, \lambda y.\, \texttt{case}\; y\; \kappa_1\\
&\quad (\texttt{con}\, \kappa_1\; (\texttt{c2n}([\alpha^f/\alpha]\tau_{\kappa_1})@(\texttt{decon}\, y)))\\
&\quad (\texttt{con}\, \kappa_2\; (\texttt{c2n}([\alpha^f/\alpha]\tau_{\kappa_2})@(\texttt{decon}\, y))))\\
\texttt{c2n}(\alpha^f) &= f
\end{aligned}
$$

  Note that the conversion code for a recursive type $\mu\, \alpha.\tau_{\kappa_1}+\tau_{\kappa_2}$ is a recursive function "$\texttt{fix}\, f\, \lambda x.-$." The code for a recursive call to f is generated when `n2c` is called with the recursive type α^f.

- The `n2c` receives the original type of CPS expression. The output is a conversion code that will convert the CPS values into their non-CPS versions, to be used inside a non-CPS context.

$$
\begin{aligned}
\texttt{n2c}(\iota) &= \lambda v.v\\
\texttt{n2c}(\tau_1 \to \tau_2) &= \\
&\lambda v.\lambda x.\lambda K.K@(\texttt{n2c}(\tau_2)@(v@(\texttt{c2n}(\tau_1)@x)))\\
\texttt{n2c}(\mu\, \alpha.\tau_{\kappa_1}+\tau_{\kappa_2}) &= \\
&(\texttt{fix}\, f\, \lambda y.\, \texttt{case}\; y\; \kappa_1\\
&\quad (\texttt{con}\, \kappa_1\; (\texttt{n2c}([\alpha^f/\alpha]\tau_{\kappa_1})@(\texttt{decon}\, y)))\\
&\quad (\texttt{con}\, \kappa_2\; (\texttt{n2c}([\alpha^f/\alpha]\tau_{\kappa_2})@(\texttt{decon}\, y))))\\
\texttt{n2c}(\alpha^f) &= f
\end{aligned}
$$

The conversion codes can be generated at compile time because sizes of types are finite.

The partial CPS transformation $[\![\cdot]\!]_N$ and $[\![\cdot]\!]_C$ by using the two conversion-code-generators (`c2n` and `n2c`) are defined in Figure 2. For a term t, we write τ_t for its type.

4 Correctness of Partial CPS transformation

To prove the correctness of our partial CPS transformation, we extend our annotated language to have annotation sequences, not just a single annotation. These annotation sequences are used to trace the changes of annotations (hence the conversions between CPS and non-CPS styles) during the evaluation.

$$
\begin{aligned}
[\![\mathtt{1}^N]\!]_N &= \mathtt{1}\\
[\![(\lambda x.e)^N]\!]_N &= \lambda x.[\![e]\!]_N\\
[\![(\mathtt{fix}\ f\ \lambda x.e)^N]\!]_N &= \mathtt{fix}\ f\ \lambda x.[\![e]\!]_N\\
[\![x^N]\!]_N &= x\\
[\![(e_1@e_2)^N]\!]_N &= [\![e_1]\!]_N@[\![e_2]\!]_N\\
[\![(\mathtt{con}\,\kappa\, e)^N]\!]_N &= \mathtt{con}\,\kappa\,[\![e]\!]_N\\
[\![(\mathtt{decon}\, e)^N]\!]_N &= \mathtt{decon}\,[\![e]\!]_N\\
[\![(\mathtt{case}\ e_1\ \kappa\ e_2\ e_3)^N]\!]_N &= \mathtt{case}\ [\![e_1]\!]_N\ \kappa\ [\![e_2]\!]_N\ [\![e_3]\!]_N\\
[\![t^C]\!]_N &= [\![t^C]\!]_C@\mathtt{c2n}(\tau_t)
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathtt{1}^C]\!]_C &= \lambda K.K@\mathtt{1}\\
[\![(\lambda x.e)^C]\!]_C &= \lambda K.K@\lambda x.[\![e]\!]_C\\
[\![(\mathtt{fix}\ f\ \lambda x.e)^C]\!]_C &= \lambda K.K@\mathtt{fix}\ f\ \lambda x.[\![e]\!]_C\\
[\![x^C]\!]_C &= \lambda K.K@x\\
[\![(e_1@e_2)^C]\!]_C &= \lambda K.[\![e_1]\!]_C@\\
&\qquad (\lambda f.[\![e_2]\!]_C@(\lambda v.f@v@K))\\
[\![(\mathtt{con}\,\kappa\, e)^C]\!]_C &= \lambda K.[\![e]\!]_C@(\lambda v.K@(\mathtt{con}\,\kappa\, v))\\
[\![(\mathtt{decon}\, e)^C]\!]_C &= \lambda K.[\![e]\!]_C@(\lambda v.K@(\mathtt{decon}\, v))\\
[\![(\mathtt{case}\ e_1\ \kappa\ e_2\ e_3)^C]\!]_C &= \lambda K.[\![e_1]\!]_C@(\lambda v.\mathtt{case}\ v\ \kappa\\
&\qquad ([\![e_2]\!]_C@K)\ ([\![e_2]\!]_C@K))\\
[\![t^N]\!]_C &= \lambda K.K@(\mathtt{n2c}(\tau_t)@[\![t^N]\!]_N)
\end{aligned}
$$

Figure 2: Partial CPS transformation

$$
\begin{array}{lcl}
\textit{annotated expressions} & & \\
e & ::= & t^a \mid e^a\\
t & ::= & \mathtt{1}\\
 & \mid & \lambda x.e\\
 & \mid & \mathtt{fix}\ f\ \lambda x.e\\
 & \mid & \kappa\cdot v\\
 & \mid & x\\
 & \mid & e_1@e_2\\
 & \mid & \mathtt{con}\,\kappa\, e\\
 & \mid & \mathtt{decon}\, e\\
 & \mid & \mathtt{case}\ e_1\ \kappa\ e_2\ e_3\\
\textit{annotated values} & & \\
v & ::= & w^a \mid v^a\\
w & ::= & \mathtt{1}\\
 & \mid & \lambda x.e\\
 & \mid & \mathtt{fix}\ f\ \lambda x.e\\
 & \mid & \kappa\cdot v\\
\textit{annotation} & & \\
a & ::= & N \mid C
\end{array}
$$

For convenience, we use the following notation:

$$
\begin{array}{lccl}
\textit{annotation sequence} & l & ::= & a \mid la\\
\textit{annotated expressions} & t^{la} & = & (t^l)^a\\
\textit{reversal of annotations} & \overline{a} & = & a\\
 & \overline{al} & = & \overline{l}a
\end{array}
$$

We now define the semantics of annotated expressions. This semantics is equivalent to that of ML core language except that we book-keep the annotations during reduction. The annotated evaluation context E is defined by the following grammar:

$$
\begin{array}{lcl}
\textit{annotated contexts} & & \\
E & ::= & D^a\\
D & ::= & [\,]\\
 & \mid & E@e\\
 & \mid & v@E\\
 & \mid & \mathtt{con}\,\kappa\, E\\
 & \mid & \mathtt{decon}\, E\\
 & \mid & \mathtt{case}\ E\ \kappa\ e_1\ e_2
\end{array}
$$

This context defines a left-to-right, call-by-value reduction. As usual, we write $E[e]$ if the hole in context E is filled with e. We use this context to define the reduction rules for arbitrary expressions :

$$
\frac{e \mapsto e'}{E[e] \to E[e']}
$$

The single reduction step $e \mapsto e'$ for a redex e is defined as :

$$
\begin{array}{rcll}
((\lambda x.e)^l@v)^a & \mapsto & [v^{a\overline{l}}/x]e^{la} & \\
((\mathtt{fix}\ f\ \lambda x.e)^l@v)^{a'} & \mapsto & [(\mathtt{fix}\ f\ \lambda x.e)^a/f][v^{a'\overline{l}}/x]e^{la'} & \\
 & & & (\text{head of } l \text{ is } a)\\
(\mathtt{con}\,\kappa\, v)^a & \mapsto & (\kappa\cdot v)^a & \\
(\mathtt{decon}\ (\kappa\cdot v)^l)^a & \mapsto & (v)^{la} & \\
(\mathtt{case}\ (\kappa\cdot v)^l\ \kappa\ e_1\ e_2)^a & \mapsto & e_1^a & \\
(\mathtt{case}\ (\kappa\cdot v)^l\ \kappa'\ e_1\ e_2)^a & \mapsto & e_2^a & (\kappa \neq \kappa')
\end{array}
$$

Notation $[v/x]e$ denotes, as usual, the new expression that results from substituting v for every free occurrence of x in e.

Definition 2 *The semantics of a closed annotated expression e is defined to be the sequence of reduction steps*

$$
e \to e_1 \to e_2 \to \cdots.
$$

If the sequence terminates with a value v after zero or more reductions, we write

$$
e \overset{*}{\to} v
$$

Now we can define a correctness theorem for our partial CPS transformation similarly to the simulation theorem in [Plo75].

Theorem 1 (Correctness of Partial CPS transformation) *For any closed expression e with consistent annotations,*

$$
e \overset{*}{\to} v \iff [\![e]\!]_N \overset{*}{\to} \Psi_N(v)
$$

where the auxiliary functions Ψ_N and Ψ_C coerces direct-style values to partial CPS values:

$$
\begin{array}{rcll}
\Psi_N(\mathtt{1}^N) & = & \mathtt{1} & \\
\Psi_N((\lambda x.e)^N) & = & \lambda x.[\![e]\!]_N & \\
\Psi_N((\mathtt{fix}\ f\ \lambda x.e)^N) & = & \mathtt{fix}\ f\ \lambda x.[\![e]\!]_N & \\
\Psi_N((\kappa\cdot v)^N) & = & \kappa\cdot\Psi_N(v) & \\
\Psi_N(v^N) & = & \Psi_N(v) & \\
\Psi_N(w^C) & = & w' & (\mathtt{c2n}(\tau_w)@\Psi_C(w^C) \overset{*}{\to} w')\\
\Psi_N(v^C) & = & v' & (\mathtt{c2n}(\tau_v)@\Psi_C(v^C) \overset{*}{\to} v')
\end{array}
$$

$$
\begin{array}{rcll}
\Psi_C(\mathtt{1}^C) & = & \mathtt{1} & \\
\Psi_C((\lambda x.e)^C) & = & \lambda x.[\![e]\!]_C & \\
\Psi_C((\mathtt{fix}\ f\ \lambda x.e)^C) & = & \mathtt{fix}\ f\ \lambda x.[\![e]\!]_C & \\
\Psi_C((\kappa\cdot v)^C) & = & \kappa\cdot\Psi_C(v) & \\
\Psi_C(v^C) & = & \Psi_C(v) & \\
\Psi_C(w^N) & = & w' & (\mathtt{n2c}(\tau_w)@\Psi_N(w^N) \overset{*}{\to} w')\\
\Psi_C(v^N) & = & v' & (\mathtt{n2c}(\tau_v)@\Psi_N(v^N) \overset{*}{\to} v')
\end{array}
$$

$$
\begin{array}{rcl}
\llbracket v \rrbracket_N & = & \Psi_N(v) \\
\llbracket (\lambda x.e)^N \rrbracket_N & = & \lambda x.\llbracket e \rrbracket_N \\
\llbracket (\mathtt{fix}\ f\ \lambda x.e)^N \rrbracket_N & = & \mathtt{fix}\ f\ \lambda x.\llbracket e \rrbracket_N \\
\llbracket x^N \rrbracket_N & = & x \\
\llbracket (e_1 @ e_2)^N \rrbracket_N & = & \llbracket e_1 \rrbracket_N @ \llbracket e_2 \rrbracket_N \\
\llbracket (\mathtt{con}\ \kappa\ e)^N \rrbracket_N & = & \mathtt{con}\ \kappa\ \llbracket e \rrbracket_N \\
\llbracket (\mathtt{decon}\ e)^N \rrbracket_N & = & \mathtt{decon}\ \llbracket e \rrbracket_N \\
\llbracket (\mathtt{case}\ e_1\ \kappa\ e_2\ e_3)^N \rrbracket_N & = & \mathtt{case}\ \llbracket e_1 \rrbracket_N\ \kappa\ \llbracket e_2 \rrbracket_N\ \llbracket e_3 \rrbracket_N \\
\llbracket e^N \rrbracket_N & = & \llbracket e \rrbracket_N \\
\llbracket t^C \rrbracket_N & = & \llbracket t^C \rrbracket_C @ \mathtt{c2n}(\tau_t) \\
\llbracket e^C \rrbracket_N & = & \llbracket e^C \rrbracket_C @ \mathtt{c2n}(\tau_e) \\
\\
\llbracket v \rrbracket_C & = & \lambda K.K @ \Psi_C(v) \\
\llbracket (\lambda x.e)^C \rrbracket_C & = & \lambda K.K @ \lambda x.\llbracket e \rrbracket_C \\
\llbracket (\mathtt{fix}\ f\ \lambda x.e)^C \rrbracket_C & = & \lambda K.K @ \mathtt{fix}\ f\ \lambda x.\llbracket e \rrbracket_C \\
\llbracket x^C \rrbracket_C & = & \lambda K.K @ x \\
\llbracket (e_1 @ e_2)^C \rrbracket_C & = & \lambda K.\llbracket e_1 \rrbracket_C @ \\
 & & \quad (\lambda f.\llbracket e_2 \rrbracket_C @ (\lambda v.f @ v @ K)) \\
\llbracket (\mathtt{con}\ \kappa\ e)^C \rrbracket_C & = & \lambda K.\llbracket e \rrbracket_C @ (\lambda v.K @ (\mathtt{con}\ \kappa\ v)) \\
\llbracket (\mathtt{decon}\ e)^C \rrbracket_C & = & \lambda K.\llbracket e \rrbracket_C @ (\lambda v.K @ (\mathtt{decon}\ v)) \\
\llbracket (\mathtt{case}\ e_1\ \kappa\ e_2\ e_3)^C \rrbracket_C & = & \lambda K.\llbracket e_1 \rrbracket_C @ (\lambda v.\mathtt{case}\ v\ \kappa \\
 & & \quad (\llbracket e_2 \rrbracket_C @ K)\ (\llbracket e_2 \rrbracket_C @ K)) \\
\llbracket e^C \rrbracket_C & = & \llbracket e \rrbracket_C \\
\llbracket t^N \rrbracket_C & = & \lambda K.K @ (\mathtt{n2c}(\tau_t) @ \llbracket t^N \rrbracket_N) \\
\llbracket e^N \rrbracket_C & = & \lambda K.K @ (\mathtt{n2c}(\tau_e) @ \llbracket e^N \rrbracket_N)
\end{array}
$$

Figure 3: Extended Partial CPS transformation

$$
\begin{array}{rcl}
v{:} & = & \Psi_N(v) \\
(v_1 @ v_2)^N{:} & = & (\Psi_N(v_1)) @ (\Psi_N(v_2)) \\
(v @ e)^N{:} & = & (\Psi_N(v)) @ (e{:}) \\
(e_1 @ e_2)^N{:} & = & (e_1{:}) @ \llbracket e_2 \rrbracket_N \\
(\mathtt{con}\ \kappa\ v)^N{:} & = & \mathtt{con}\ \kappa\ (\Psi_N(v)) \\
(\mathtt{con}\ \kappa\ e)^N{:} & = & \mathtt{con}\ \kappa\ (e{:}) \\
(\mathtt{decon}\ v)^N{:} & = & \mathtt{decon}\ (\Psi_N(v)) \\
(\mathtt{decon}\ e)^N{:} & = & \mathtt{decon}\ (e{:}) \\
(\mathtt{case}\ v\ \kappa\ e_1\ e_2)^N{:} & = & \mathtt{case}\ (\Psi_N(v))\ \kappa\ \llbracket e_1 \rrbracket_N\ \llbracket e_2 \rrbracket_N \\
(\mathtt{case}\ e_1\ \kappa\ e_2\ e_3)^N{:} & = & \mathtt{case}\ (e_1{:})\ \kappa\ \llbracket e_2 \rrbracket_N\ \llbracket e_3 \rrbracket_N \\
e^N{:} & = & e{:} \\
t^C{:} & = & t^C{:}\mathtt{c2n}(\tau_t) \\
e^C{:} & = & e^C{:}\mathtt{c2n}(\tau_e) \\
\\
v{:}K & = & K @ \Psi_C(v) \\
(v_1 @ v_2)^C{:}K & = & \Psi_C(v_1) @ \Psi_C(v_2) @ K \\
(v @ e)^C{:}K & = & e{:}\lambda v.\Psi_C(v) @ v @ K \\
(e_1 @ e_2)^C{:}K & = & e_1{:}\lambda f.\llbracket e_2 \rrbracket_C @ (\lambda v.f @ v @ K) \\
(\mathtt{con}\ \kappa\ v)^C{:}K & = & K @ (\mathtt{con}\ \kappa\ (\Psi_C(v))) \\
(\mathtt{con}\ \kappa\ e)^C{:}K & = & e{:}\lambda v.K @ (\mathtt{con}\ \kappa\ v) \\
(\mathtt{decon}\ v)^C{:}K & = & K @ (\mathtt{decon}\ (\Psi_C(v))) \\
(\mathtt{decon}\ e)^C{:}K & = & e{:}\lambda v.K @ (\mathtt{decon}\ v) \\
(\mathtt{case}\ v\ \kappa\ e_1\ e_2)^C{:}K & = & \mathtt{case}\ (\Psi_C(v))\ \kappa \\
 & & \quad (\llbracket e_1 \rrbracket_C @ K)\ (\llbracket e_2 \rrbracket_C @ K) \\
(\mathtt{case}\ e_1\ \kappa\ e_2\ e_3)^C{:}K & = & e_1{:}(\lambda v.\mathtt{case}\ v\ \kappa \\
 & & \quad (\llbracket e_2 \rrbracket_C @ K)(\llbracket e_3 \rrbracket_C @ K)) \\
e^C{:}K & = & e{:}K \\
t^N{:}K & = & K @ (\mathtt{n2c}(\tau_t) @ (t^N{:})) \\
e^N{:}K & = & K @ (\mathtt{n2c}(\tau_e) @ (e^N{:}))
\end{array}
$$

Figure 4: The Colon Operators

We use a similar approach to [Plo75] to prove the above theorem.

At first, we extend the partial CPS transformation to be defined also for the intermediate expressions during reduction. Then we define two colon operators $:$ and $:K$ to represent the expressions after the administrative reduction. Operator $:$ is used for $\llbracket\cdot\rrbracket_N$ and $:K$ for $\llbracket\cdot\rrbracket_C$. In partially CPS-transforming the intermediate expressions, computed values (closed expressions without redices) are transformed by a special transformers Ψ_N and Ψ_C. The extended partial CPS transformation is defined in Figure 3.

Figure 4 defines the two colon operators ($:$ and $:K$) which represent the transformed terms (respectively from the $\llbracket\cdot\rrbracket_N$ and $\llbracket\cdot\rrbracket_C$) after the administrative reductions.

Value transformation preserves the structure of the values:

Lemma 1

$$\Psi_N((\kappa \cdot v)^l) = \kappa \cdot \Psi_N(v^l) \quad \textit{and} \quad \Psi_C((\kappa \cdot v)^l) = \kappa \cdot \Psi_C(v^l)$$

Proof. Proof by induction on the length of l. The details of the proof are in Appendix A. □

Following three Lemmas correspond to the Plotkin's ones [Plo75]. Substitution Lemma enables substitions to penetrate $\llbracket\cdot\rrbracket_N$ and $\llbracket\cdot\rrbracket_C$:

Lemma 2 (Substitution) *For any expression e with consistent annotation and any value v, if free variable x is annotated as N then*

$$
\begin{array}{rcl}
[\Psi_N(v)/x]\llbracket e \rrbracket_N & = & \llbracket [v/x]e \rrbracket_N \\
[\Psi_N(v)/x]\llbracket e \rrbracket_C & = & \llbracket [v/x]e \rrbracket_C
\end{array}
$$

and if free variable x is annotated as C then

$$
\begin{array}{rcl}
[\Psi_C(v)/x]\llbracket e \rrbracket_N & = & \llbracket [v/x]e \rrbracket_N \\
[\Psi_C(v)/x]\llbracket e \rrbracket_C & = & \llbracket [v/x]e \rrbracket_C
\end{array}
$$

Proof. Proof by mutual structural induction on e with a case for type of expression. The details of the proof are in Appendix B. □

Static reduction Lemma establishes the relation between partial CPS transformation and colon operator:

Lemma 3 (Static Reduction) *For any closed expression e with consistent annotation and any closed value K,*

$$
\begin{array}{rcl}
\llbracket e \rrbracket_N & \stackrel{*}{\to} & e{:} \\
\llbracket e \rrbracket_C @ K & \stackrel{*}{\to} & e{:}K
\end{array}
$$

Proof. Proof by mutual structural induction on e with a case for type of expression. The details of the proof are in Appendix C. □

Single-Step Simulation Lemma shows that a reduction from an original expression corresponds to reductions between partially CPS transformed expressions:

Lemma 4 (Single-Step Simulation) *For any closed expression e with consistent annotations and any closed value K,*

$$e \to e' \implies e{:} \xrightarrow{+} e'{:} \quad \text{and} \quad e{:}K \xrightarrow{+} e'{:}K$$

Proof. Proof by mutual structural induction on the reduction $e \to e'$ with a case for type of reduction rules. The details of the proof are in Appendix D. □

Proof of Theorem 1. Using the lemmas, we can prove the correctness theorem easily. If the original program evaluates to a value, that is , $e \xrightarrow{*} v$ then

$$\begin{array}{lll} [\![e]\!]_N & & \\ \xrightarrow{*} & e{:} & \text{(Lemma 3)} \\ \xrightarrow{*} & v{:} & \text{(repeated use of Lemma 4)} \\ = & \Psi_N(v) & \text{(def. of :)} \end{array}$$

Note that non-termination is also preserved by the single step simulation.

□

5 Conclusion

We presented a type-based partial CPS transformation. The partial CPS transformation transforms only parts of a program and connects the results to the rest, direct-style parts. A program's sub-parts which need to be CPS-transformed is initially annotated as such. Partial CPS-transformation scans the expressions and selectively apply the transformation depending on the annotation. Wherever an interface between CPS-transformed and direct-style expressions is needed, proper conversion code is padded based on expression types. The correctness of the partial CPS transformation is proven similarly to that of Plotkin's simulation theorem[Plo75].

There are analogous works in literature. Our `n2c` and `c2n` functions can be seen as the retraction pairs in the setting of Meyer and Wand[MW85]. Danvy, Dussart, and Hatcliff [DD95, DH93a, DH93b]'s works are analogous to ours. They switch between call-by-value and call-by-name CPS transformations interconnecting the two transformations by "delay" and "force" operators. Another analogous one is Leroy's "box" and "unbox" operators[Ler92] that are used to handle mixed representations of values.

References

[DD95] Olivier Danvy and Dirk Dussart. CPS transformation after binding-time analysis. Unpublished note, Department of Computer Science, University of Aarhus, April 1995.

[DH93a] Olivier Danvy and John Hatcliff. CPS transformation after strictness analysis. *ACM Letters on Programming Languages and Systems*, 1(3):195–212, 1993.

[DH93b] Olivier Danvy and John Hatcliff. On the transformation between direct and continuation semantics. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Proceedings of the 9th Conference on Mathematical Foundations of Programming Semantics*, number 802 in Lecture Notes in Computer Science, pages 627–648, New Orleans, Louisiana, April 1993. Springer-Verlag.

[KYD98] Jung-taek Kim, Kwangkeun Yi, and Olivier Danvy. Assessing the overhead of ML exceptions by selective CPS transformation. In Greg Morrisett, editor, *Record of the 1998 ACM SIGPLAN Workshop on ML and its Applications*, Baltimore, Maryland, September 1998. Also appears as BRICS technical report RS-98-15.

[Ler92] Xavier Leroy. Unboxed objects and polymorphic typing. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 177–188, January 1992.

[MW85] Albert R. Meyer and Mitchell Wand. Continuation semantics in typed lambda-calculi. In R. Parikh, editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 219–224. Springer, Brooklyn, June 1985.

[Plo75] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

A Proof of Lemma 1

Lemma 1

$$\Psi_N((\kappa{\cdot}v)^l) = \kappa{\cdot}\Psi_N(v^l) \quad \text{and} \quad \Psi_C((\kappa{\cdot}v)^l) = \kappa{\cdot}\Psi_C(v^l)$$

Proof.

Proof by induction on the length of l.

In proof, we abbreviate some expressions as "$(\cdots)$" when they are omittable.

For Ψ_N cases,

- base case $(\kappa{\cdot}v)^N$

$$\begin{array}{lll} \Psi_N((\kappa{\cdot}v)^N) & & \\ = & \kappa{\cdot}\Psi_N(v) & \text{(def. of } \Psi_N) \\ = & \kappa{\cdot}\Psi_N(v^N) & \text{(def. of } \Psi_N) \end{array}$$

- base case $(\kappa{\cdot}v)^C$

$$\begin{array}{ll} \multicolumn{2}{l}{\texttt{c2n}(\mu\,\alpha.\tau_\kappa{+}\tau_{\kappa'})@\Psi_C((\kappa{\cdot}v)^C)} \\ = & (\texttt{fix}\ f\ \lambda y.\texttt{case}\ y\ \kappa \\ & \quad (\texttt{con}\ \kappa\ \texttt{c2n}([\alpha^f/\alpha]\tau_\kappa)@(\texttt{decon}\ y))\ (\cdots)) \\ & @(\kappa{\cdot}\Psi_C(v)) \quad \text{(def. of } \texttt{n2c} \text{ and } \Psi_N) \\ \to & \texttt{case}\ (\kappa{\cdot}\Psi_C(v))\ \kappa \\ & \quad (\texttt{con}\ \kappa\ (\texttt{c2n}([\alpha^{f'}/\alpha]\tau_\kappa) \\ & \qquad @(\texttt{decon}\ (\kappa{\cdot}\Psi_C(v)))))\ (\cdots) \\ & \quad \text{(where } f' \text{ is } \texttt{fix}\ f\ \cdots) \\ \xrightarrow{+} & \texttt{con}\ \kappa\ (\texttt{c2n}([\alpha^{f'}/\alpha]\tau_\kappa)@(\Psi_C(v))) \\ = & \texttt{con}\ \kappa\ (\texttt{c2n}([\alpha^{f'}/\alpha]\tau_\kappa)@(\Psi_C(v^C))) \\ & \text{(def. of } \Psi_N) \\ \xrightarrow{*} & \texttt{con}\ \kappa\ (\Psi_N(v^C)) \\ & \quad \text{(Because } [\alpha^{f'}/\alpha]\tau_\kappa \text{ is indeed the type of } v, \\ & \quad \texttt{c2n}([\alpha^{f'}/\alpha]\tau_\kappa)@\Psi_C(v^C) \xrightarrow{*} v' = \Psi_N(v^C) \\ & \quad \text{by definition of } \Psi_N.) \\ \to & \kappa{\cdot}(\Psi_N(v^C)) \end{array}$$

By above and definition of Ψ_N,

$$\begin{array}{ll} \Psi_N((\kappa{\cdot}v)^C) & \\ = & \kappa{\cdot}\Psi_N(v^C) \end{array}$$

- induction step case $(\kappa\cdot v)^{lN}$

$$\begin{array}{lll} \Psi_N((\kappa\cdot v)^{lN}) & & \\ \quad = & \Psi_N((\kappa\cdot v)^{l}) & \text{(def. of } \Psi_N) \\ \quad = & \kappa\cdot\Psi_N((v)^{l}) & \text{(I.H.)} \\ \quad = & \kappa\cdot\Psi_N((v)^{lN}) & \text{(def. of } \Psi_N) \end{array}$$

- induction step case $(\kappa\cdot v)^{lC}$

$$\begin{array}{lll} \mathtt{c2n}(\mu\,\alpha.\tau_\kappa{+}\tau_{\kappa'})@\Psi_C((\kappa\cdot v)^{lC}) & & \\ \quad = & \mathtt{c2n}(\mu\,\alpha.\tau_\kappa{+}\tau_{\kappa'})@\Psi_C((\kappa\cdot v)^{l}) & \text{(def. of } \Psi_C) \\ \quad = & (\mathtt{fix}\ f\ \lambda y.\mathtt{case}\ y\ \kappa & \\ & \quad\ (\mathtt{con}\ \kappa\ \mathtt{c2n}([\alpha^f/\alpha]\tau_\kappa)@(\mathtt{decon}\ y))\ (\cdots)) & \\ & @(\kappa\cdot\Psi_C(v^l)) & \text{(def. of } \mathtt{n2c} \text{ and I.H.)} \\ \quad \to & \mathtt{case}\ (\kappa\cdot\Psi_C(v^l))\ \kappa & \\ & \quad\ (\mathtt{con}\ \kappa\ (\mathtt{c2n}([\alpha^{f'}/\alpha]\tau_\kappa) & \\ & \quad\quad\ @(\mathtt{decon}\ (\kappa\cdot\Psi_C(v^l)))))\ (\cdots) & \\ & \text{(where } f' \text{ is } \mathtt{fix}\ f\ \cdots) & \\ \quad \xrightarrow{+} & \mathtt{con}\ \kappa\ (\mathtt{c2n}([\alpha^{f'}/\alpha]\tau_\kappa)@(\Psi_C(v^l))) & \\ \quad = & \mathtt{con}\ \kappa\ (\mathtt{c2n}([\alpha^{f'}/\alpha]\tau_\kappa)@(\Psi_C(v^{lC}))) & \\ & & \text{(def. of } \Psi_C) \\ \quad \xrightarrow{*} & \mathtt{con}\ \kappa\ (\Psi_N(v^{lC})) & \\ & \text{(Because } [\alpha^{f'}/\alpha]\tau_\kappa \text{ is indeed the type of } v, & \\ & \ \mathtt{c2n}([\alpha^{f'}/\alpha]\tau_\kappa)@\Psi_C(v^{lC}) \xrightarrow{*} v' = \Psi_N(v^{lC}) & \\ & \ \text{by definition of } \Psi_N.) & \\ \quad \to & \kappa\cdot(\Psi_N(v^{lC})) & \end{array}$$

By above and definition of Ψ_N,

$$\begin{array}{lll} \Psi_N((\kappa\cdot v)^{lC}) & & \\ \quad = & \kappa\cdot\Psi_N(v^{lC}) & \text{(def. of } \Psi_N) \end{array}$$

Because Ψ_T and Ψ_N are duals, we can prove Ψ_T cases similarly to Ψ_N cases. □

B Proof of Substitution Lemma

Lemma 2 (Substitution) *For any expression e with consistent annotation and any value v, if free variable x is annotated as N then*

$$\begin{array}{rcl} [\Psi_N(v)/x][\![e]\!]_N & = & [\![[v/x]e]\!]_N \\ [\Psi_N(v)/x][\![e]\!]_C & = & [\![[v/x]e]\!]_C \end{array}$$

and if free variable x is annotated as C then

$$\begin{array}{rcl} [\Psi_C(v)/x][\![e]\!]_N & = & [\![[v/x]e]\!]_N \\ [\Psi_C(v)/x][\![e]\!]_C & = & [\![[v/x]e]\!]_C \end{array}$$

Proof.

Proof by mutual structural induction on e, firstly taking the $[\![\cdot]\!]_N$ for the N annotated expressions and $[\![\cdot]\!]_C$ for the C annotated expressions.

For $[\![\cdot]\!]_N$ cases,

- case v, $(\lambda x.e_1)^N$, $(\mathtt{fix}\ f\ \lambda x.e_1)^N$, or $(\mathtt{fix}\ x\ \lambda y.e_1)^N$

$$\begin{array}{lll} [\Psi_a(v)/x][\![e]\!]_N & & \\ \quad = & [\![e]\!]_N & (x \text{ is not free}) \\ \quad = & [\![[v/x]e]\!]_N & (x \text{ is not free}) \end{array}$$

a is N or C according to the annotation of the free variable x.

- case $(\lambda y.e)^N \quad (x \neq y)$

$$\begin{array}{lll} [\Psi_a(v)/x][\![(\lambda y.e)^N]\!]_N & & \\ \quad = & [\Psi_a(v)/x]\lambda y.[\![e]\!]_N & \text{(def. of } [\![\cdot]\!]_N) \\ \quad = & \lambda y.[\Psi_a(v)/x][\![e]\!]_N & \text{(def. of } [/]) \\ \quad = & \lambda y.[\![[v/x]e]\!]_N & \text{(I.H.)} \\ \quad = & [\![(\lambda y.[v/x]e)^N]\!]_N & \text{(def. of } [\![\cdot]\!]_N) \\ \quad = & [\![[v/x](\lambda y.e)^N]\!]_N & \text{(def. of } [/]) \end{array}$$

a is N or C according to the annotation of the free variable x.

- case $(\mathtt{fix}\ f\ \lambda y.e)^N \quad (x \neq y, x \neq f)$

We can prove similarly to the above $(\lambda y.e)^N$ case.

- case x^N

Because the variable x is annotated as N, we need to prove only for Ψ_N.

$$\begin{array}{lll} [\Psi_N(v)/x][\![x^N]\!]_N & & \\ \quad = & [\Psi_N(v)/x]x & \text{(def. of } [\![\cdot]\!]_N) \\ \quad = & \Psi_N(v) & \text{(def. of } [/]) \\ \quad = & \Psi_N(v^N) & \text{(def. of } \Psi_N) \\ \quad = & [\![v^N]\!]_N & \text{(def. of } [\![\cdot]\!]_N) \\ \quad = & [\![[v/x]x^N]\!]_N & \text{(def. of } [/]) \end{array}$$

- case $(e_1@e_2)^N$

The annotations of the free variable x in both e_1 and e_2 are equal.

$$\begin{array}{lll} [\Psi_a(v)/x][\![(e_1@e_2)^N]\!]_N & & \\ \quad = & [\Psi_a(v)/x]([\![e_1]\!]_N@[\![e_2]\!]_N) & \text{(def. of } [\![\cdot]\!]_N) \\ \quad = & ([\Psi_a(v)/x][\![e_1]\!]_N)@([\Psi_a(v)/x][\![e_2]\!]_N) & \\ & & \text{(def. of } [/]) \\ \quad = & [\![[v/x]e_1]\!]_N@[\![[v/x]e_2]\!]_N & \text{(I.H. twice)} \\ \quad = & [\![(([v/x]e_1)@([v/x]e_2))^N]\!]_N & \text{(def. of } [\![\cdot]\!]_N) \\ \quad = & [\![[v/x](e_1@e_2)^N]\!]_N & \text{(def. of } [/]) \end{array}$$

a is N or C according to the annotation of the free variable x.

- case $(\mathtt{con}\ \kappa\ e)^N$, $(\mathtt{decon}\ e)^N$, $(\mathtt{case}\ e_1\ \kappa\ e_2\ e_3)^N$, or e^N

We can prove similarly to the above $(e_1@e_2)^N$ case.

- case e^C

$$\begin{array}{lll} [\Psi_a(v)/x][\![e^C]\!]_N & & \\ \quad = & [\Psi_a(v)/x]([\![e^C]\!]_C@\mathtt{c2n}(\tau_e)) & \text{(def. of } [\![\cdot]\!]_N) \\ \quad = & ([\Psi_a(v)/x][\![e^C]\!]_C)@\mathtt{c2n}(\tau_e) & \text{(def. of } [/]) \\ \quad = & [\![[v/x]e^C]\!]_C@\mathtt{c2n}(\tau_e) & \text{(I.H. about } [\![\cdot]\!]_C) \\ \quad = & [\![[v/x]e^C]\!]_N & \text{(def. of } [\![\cdot]\!]_N) \end{array}$$

- case t^C

We can prove similarly to the above e^C case.

For $[\![\cdot]\!]_C$ cases,

- case v, $(\lambda x.e_1)^C$, $(\mathtt{fix}\ f\ \lambda x.e_1)^C$, or $(\mathtt{fix}\ x\ \lambda y.e_1)^C$

$$\begin{array}{lll} [\Psi_a(v)/x][\![e]\!]_C & & \\ \quad = & [\![e]\!]_C & (x \text{ is not free}) \\ \quad = & [\![[v/x]e]\!]_C & (x \text{ is not free}) \end{array}$$

a is N or C according to the annotation of the free variable x.

- case $(\lambda y.e)^C \quad (x \neq y)$

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\Psi_a(v)/x][\![(\lambda y.e)^C]\!]_C} & \\
= & [\Psi_a(v)/x]\lambda K.K@\lambda x.[\![e]\!]_C & (\text{def. of } [\![\cdot]\!]_C) \\
= & \lambda K.K@\lambda x.[\Psi_a(v)/x][\![e]\!]_C & (\text{def. of } [/]) \\
= & \lambda K.K@\lambda x.[\![[v/x]e]\!]_C & (\text{I.H.}) \\
= & [\![(\lambda y.[v/x]e)^C]\!]_C & (\text{def. of } [\![\cdot]\!]_C) \\
= & [\![[v/x](\lambda y.e)^C]\!]_C & (\text{def. of } [/])
\end{array}
$$

a is N or C according to the annotation of the free variable x.

- case $(\texttt{fix}\, f\, \lambda y.e)^C \quad (x \neq y, x \neq f)$

  We can prove similarly to the above $(\lambda y.e)^C$ case.

- case x^C

  Because the variable x is annotated as C, we need to prove only for Ψ_C.

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\Psi_C(v)/x][\![x^C]\!]_C} & \\
= & [\Psi_C(v)/x]\lambda K.K@x & (\text{def. of } [\![\cdot]\!]_C) \\
= & \lambda K.K@\Psi_C(v) & (\text{def. of } [/]) \\
= & \lambda K.K@\Psi_C(v^C) & (\text{def. of } \Psi_C) \\
= & [\![v^C]\!]_C & (\text{def. of } [\![\cdot]\!]_C) \\
= & [\![[v/x]x^C]\!]_C & (\text{def. of } [/])
\end{array}
$$

- case $(e_1@e_2)^C$

  The annotations of the free variable x in both e_1 and e_2 are equal.

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\Psi_a(v)/x][\![(e_1@e_2)^C]\!]_C} & \\
= & [\Psi_a(v)/x]\lambda K.[\![e_1]\!]_C@(\lambda f.[\![e_2]\!]_C@(\lambda v.f@v@K)) & \\
 & & (\text{def. of } [\![\cdot]\!]_C) \\
= & \lambda K.([\Psi_a(v)/x][\![e_1]\!]_C)@ & \\
 & \quad (\lambda f.([\Psi_a(v)/x][\![e_2]\!]_C)@(\lambda v.f@v@K)) & \\
 & & (\text{def. of } [/]) \\
= & \lambda K.[\![[v/x]e_1]\!]_C@(\lambda f.[\![[v/x]e_2]\!]_C@(\lambda v.f@v@K)) & \\
 & & (\text{I.H.}) \\
= & [\![(([v/x]e_1)@([v/x]e_2))^C]\!]_C & (\text{def. of } [\![\cdot]\!]_C) \\
= & [\![[v/x](e_1@e_2)^C]\!]_C & (\text{def. of } [/])
\end{array}
$$

a is N or C according to the annotation of the free variable x.

- case $(\texttt{con}\,\kappa\, e)^C$, $(\texttt{decon}\, e)^C$, $(\texttt{case}\, e_1\, \kappa\, e_2\, e_3)^C$, or e^C

  We can prove similarly to the above $(e_1@e_2)^C$ case.

- case e^N

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\Psi_a(v)/x][\![e^N]\!]_C} & \\
= & [\Psi_a(v)/x](\lambda K.K@(\texttt{n2c}(\tau_e)@[\![e^N]\!]_N)) & \\
 & & (\text{def. of } [\![\cdot]\!]_C) \\
= & \lambda K.K@(\texttt{n2c}(\tau_e)@([\Psi_a(v)/x][\![e^N]\!]_N)) & \\
 & & (\text{def. of } [/]) \\
= & \lambda K.K@(\texttt{n2c}(\tau_e)@[\![[v/x]e^N]\!]_N) & \\
 & & (\text{I.H. about } [\![\cdot]\!]_N) \\
= & [\![[v/x]e^N]\!]_C & (\text{def. of } [\![\cdot]\!]_C)
\end{array}
$$

a is N or C according to the annotation of the free variable x.

- case t^N

  We can prove similarly to the above e^N case.

□

C Proof of Static Reduction Lemma

Lemma 3 (Static Reduction) *For any closed expression e with consistent annotation and any closed value K,*

$$
\begin{array}{lll}
[\![e]\!]_N & \stackrel{*}{\to} & e{:} \\
[\![e]\!]_C@K & \stackrel{*}{\to} & e{:}K
\end{array}
$$

Proof.

Proof by mutual structural induction on e, firstly taking the $[\![\cdot]\!]_N$ for the N annotated expressions and $[\![\cdot]\!]_C$ for the C annotated expressions.

For $[\![\cdot]\!]_N$ cases,

- case v

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\![v]\!]_N} & \\
= & \Psi_N(v) & (\text{def. of } [\![\cdot]\!]_N) \\
= & v{:} & (\text{def. of } :)
\end{array}
$$

- case $(v_1@v_2)^N$

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\![(v_1@v_2)^N]\!]_N} & \\
= & [\![v_1]\!]_N@[\![v_2]\!]_N & (\text{def. of } [\![\cdot]\!]_N) \\
= & \Psi_N(v_1)@\Psi_N(v_2) & (\text{def. of } [\![\cdot]\!]_N) \\
= & (v_1@v_2)^N{:} & (\text{def. of } :)
\end{array}
$$

- case $(v@e)^N$

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\![(v@e)^N]\!]_N} & \\
= & [\![v]\!]_N@[\![e]\!]_N & (\text{def. of } [\![\cdot]\!]_N) \\
= & (\Psi_N(v))@[\![e]\!]_N & (\text{def. of } [\![\cdot]\!]_N) \\
\stackrel{*}{\to} & (\Psi_N(v))@(e{:}) & (\text{I.H.}) \\
= & (v@e)^N{:} & (\text{def. of } :)
\end{array}
$$

- case $(e_1@e_2)^N$, $(\texttt{con}\,\kappa\, v)^N$, $(\texttt{con}\,\kappa\, e)^N$, $(\texttt{decon}\, v)^N$, $(\texttt{decon}\, e)^N$, $(\texttt{case}\, v\, \kappa\, e_1\, e_2)^N$, $(\texttt{case}\, e_1\, \kappa\, e_2\, e_3)^N$, or e^N

  We can prove similarly to one of the above two cases - $(v_1@v_2)^N$, $(v@e)^N$.

- case e^C

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\![e^C]\!]_N} & \\
= & [\![e^C]\!]_C@\texttt{c2n}(\tau_e) & (\text{def. of } [\![\cdot]\!]_N) \\
\stackrel{*}{\to} & [\![e^C]\!]_C{:}\texttt{c2n}(\tau_e) & (\text{I.H. about } [\![\cdot]\!]_C) \\
= & e^C{:} & (\text{def. of } :)
\end{array}
$$

- case t^C

  We can prove similarly to the above e^C case.

For C-annotated expressions and $[\![\cdot]\!]_C$ cases,

- case v

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\![v]\!]_C@K} & \\
\to & K@\Psi_C(v) & (\text{def. of } [\![\cdot]\!]_C) \\
= & v{:}K & (\text{def. of } {:}K)
\end{array}
$$

- case $(v_1@v_2)^C$

$$
\begin{array}{lll}
\multicolumn{2}{l}{[\![(v_1@v_2)^C]\!]_C@K} & \\
\to & [\![v_1]\!]_C@(\lambda f.[\![v_2]\!]_C@(\lambda v.f@v@K)) & \\
 & & (\text{def. of } [\![\cdot]\!]_C) \\
= & (\lambda K.K@\Psi_C(v_1))@(\lambda f.[\![v_2]\!]_C@(\lambda v.f@v@K)) & \\
 & & (\text{def. of } [\![\cdot]\!]_C) \\
\stackrel{+}{\to} & [\![v_2]\!]_C@(\lambda v.\Psi_C(v_1)@v@K) & \\
= & (\lambda K.K@\Psi_C(v_2))@(\lambda v.\Psi_C(v_1)@v@K) & \\
 & & (\text{def. of } [\![\cdot]\!]_C) \\
\stackrel{+}{\to} & \Psi_C(v_1)@\Psi_C(v_2)@K & \\
= & (v_1@v_2)^C{:}K & (\text{def. of } {:}K)
\end{array}
$$

- case $(v@e)^C$

$$
\begin{array}{lll}
\multicolumn{3}{l}{[\![(v@e)^C]\!]_C@K} \\
\to & [\![v]\!]_C@(\lambda f.[\![e]\!]_C@(\lambda v.f@v@K)) & (\text{def. of } [\![\cdot]\!]_C) \\
= & (\lambda K.K@\Psi_C(v))@(\lambda f.[\![e]\!]_C@(\lambda v.f@v@K)) & \\
 & & (\text{def. of } [\![\cdot]\!]_C) \\
\xrightarrow{+} & [\![e]\!]_C@(\lambda v.\Psi_C(v)@v@K) & \\
\xrightarrow{*} & e{:}\lambda v.\Psi_C(v)@v@K & (\text{I.H.}) \\
= & (v@e)^C{:}K & (\text{def. of } {:}K)
\end{array}
$$

- case $(e_1@e_2)^C$, $(\texttt{con}\ \kappa\ v)^C$, $(\texttt{con}\ \kappa\ e)^C$, $(\texttt{decon}\ v)^C$, $(\texttt{decon}\ e)^C$, $(\texttt{case}\ v\ \kappa\ e_1\ e_2)^C$, $(\texttt{case}\ e_1\ \kappa\ e_2\ e_3)^C$, or$e^C$

We can prove similarly to one of the above two cases - $(v_1@v_2)^C$, $(v@e)^C$.

- case e^N

$$
\begin{array}{lll}
\multicolumn{3}{l}{[\![e^N]\!]_C@K} \\
\to & K@(\texttt{n2c}(\tau_e)@[\![e^N]\!]_N) & (\text{def. of } [\![\cdot]\!]_C) \\
\xrightarrow{*} & K@(\texttt{n2c}(\tau_e)@(e^N{:})) & (\text{I.H. about } [\![\cdot]\!]_N) \\
= & e^N{:}K & (\text{def. of } {:}K)
\end{array}
$$

- case t^N

We can prove similarly to the above e^N case.

□

D Proof of Single Step Simulation Lemma

Lemma 4 (Single Step Simulation) *For any closed expression e with consistent annotations and any closed value K,*

$$e \to e' \implies e{:} \xrightarrow{+} e'{:} \quad \textit{and} \quad e{:}K \xrightarrow{+} e'{:}K$$

Proof.

Proof by mutual structural induction on e, firstly taking : for the N annotated expressions and $:K$ for the C annotated expressions.

For : cases,

- $((\lambda x.e)^l@v)^N \to [v^{N\bar{l}}/x]e^{lN}$

Prove this case by induction on the length of l.

  - $((\lambda x.e)^N@v)^N \to [v^{NN}/x]e^{NN}$

  By the consistency, free variable x in e should be annotated as N.

$$
\begin{array}{lll}
\multicolumn{3}{l}{((\lambda x.e)^N@v)^N{:}} \\
= & \Psi_N((\lambda x.e)^N)@(\Psi_N(v)) & (\text{def. of } {:}) \\
= & (\lambda x.[\![e]\!]_N)@(\Psi_N(v)) & (\text{def. of } \Psi_N) \\
\to & [\Psi_N(v)/x][\![e]\!]_N & \\
= & [\Psi_N(v^{NN})/x][\![e^{NN}]\!]_N & \\
 & & (\text{def. of } \Psi_N \text{ and } [\![\cdot]\!]_N) \\
= & [\![[v^{NN}/x]e^{NN}]\!]_N & (\text{Lemma 2}) \\
\xrightarrow{*} & [v^{NN}/x]e^{NN}{:} & (\text{Lemma 3})
\end{array}
$$

  - $((\lambda x.e)^C@v)^N \to [v^{NT}/x]e^{CN}$

  By the consistency, free variable x in e should be annotated as C. And $\lambda x.e$'s type should be $\tau_1 \to \tau_2$.

$$
\begin{array}{lll}
\multicolumn{3}{l}{((\lambda x.e)^C@v)^N{:}} \\
= & \Psi_N((\lambda x.e)^C)@\Psi_N(v) & (\text{def. of } {:}) \\
= & (\lambda x.\Psi_C((\lambda x.e)^C)@(\texttt{n2c}(\tau_1)@x)@ & \\
 & \quad \texttt{c2n}(\tau_2))@\Psi_N(v^N) & (\text{def. of } \Psi_N) \\
\to & \Psi_C((\lambda x.e)^C)@(\texttt{n2c}(\tau_1)@\Psi_N(v^N))@\texttt{c2n}(\tau_2) & \\
\xrightarrow{*} & \Psi_C((\lambda x.e)^C)@\Psi_C(v^N)@\texttt{c2n}(\tau_2) & \\
 & & (\text{def. of } \Psi_C) \\
= & (\lambda x.[\![e]\!]_C)@\Psi_C(v^N)@\texttt{c2n}(\tau_2) & (\text{def. of } \Psi_C) \\
\to & ([\Psi_C(v^N)/x][\![e]\!]_C)@\texttt{c2n}(\tau_2) & \\
= & ([\Psi_C(v^{NC})/x][\![e^C]\!]_C)@\texttt{c2n}(\tau_2) & \\
 & & (\text{def. of } \Psi_C \text{ and } [\![\cdot]\!]_C) \\
= & [\![[v^{NC}/x]e^C]\!]_C@\texttt{c2n}(\tau_2) & (\text{Lemma 2}) \\
= & [\![[v^{NC}/x]e^C]\!]_N & (\text{def. of } [\![\cdot]\!]_N) \\
\xrightarrow{*} & [v^{NC}/x]e^C{:} & (\text{Lemma 3}) \\
= & [v^{NC}/x]e^{CN}{:} & (\text{def. of } {:})
\end{array}
$$

  - $((\lambda x.e)^{lN}@v)^N \to [v^{NN\bar{l}}/x]e^{lNN}$

$$
\begin{array}{lll}
\multicolumn{3}{l}{((\lambda x.e)^{lN}@v)^N{:}} \\
= & \Psi_N((\lambda x.e)^{lN})@(\Psi_N(v)) & (\text{def. of } {:}) \\
= & \Psi_N((\lambda x.e)^l)@(\Psi_N(v^N)) & (\text{def. of } \Psi_N) \\
= & ((\lambda x.e)^l@v^N)^N{:} & (\text{def. of } {:}) \\
\xrightarrow{+} & [v^{NN\bar{l}}/x]e^{lN}{:} & \\
 & & (\text{I.H. about the length of } l) \\
= & [v^{NN\bar{l}}/x]e^{lNN}{:} & (\text{def. of } {:})
\end{array}
$$

  - $((\lambda x.e)^{lT}@v)^N \to [v^{NT\bar{l}}/x]e^{lTN}$

  $\lambda x.e$'s type should be $\tau_1 \to \tau_2$.

$$
\begin{array}{lll}
\multicolumn{3}{l}{((\lambda x.e)^{lT}@v)^N{:}} \\
= & \Psi_N((\lambda x.e)^{lT})@(\Psi_N(v)) & (\text{def. of } {:}) \\
= & (\lambda x.\Psi_C((\lambda x.e)^{lT})@(\texttt{n2c}(\tau_1)@x)@\texttt{c2n}(\tau_2)) & \\
 & \quad @(\Psi_N(v^N)) & (\text{def. of } \Psi_N) \\
\to & \Psi_C((\lambda x.e)^{lT}) & \\
 & \quad @(\texttt{n2c}(\tau_1)@\Psi_N(v^N))@\texttt{c2n}(\tau_2) & \\
\xrightarrow{*} & \Psi_C((\lambda x.e)^l)@\Psi_C(v^N)@\texttt{c2n}(\tau_2) & \\
 & & (\text{def. of } \Psi_C) \\
= & ((\lambda x.e)^l@v^N)^C{:}\texttt{c2n}(\tau_2) & (\text{def. of } {:}K) \\
\xrightarrow{+} & ([v^{NT\bar{l}}/x]e^{lT}){:}\texttt{c2n}(\tau_2) & \\
 & & (\text{I.H. about the length of } l) \\
= & ([v^{NT\bar{l}}/x]e^{lT}){:} & \\
= & ([v^{NT\bar{l}}/x]e^{lTN}){:} & (\text{def. of } {:})
\end{array}
$$

By induction, $(((\lambda x.e)^l@v)^N{:}) \xrightarrow{+} ([v^{N\bar{l}}/x]e^{lN}{:})$

- $((\texttt{fix}\ f\ \lambda x.e)^{al}@v)^N \to [(\texttt{fix}\ f\ \lambda x.e)^a/f][v^{N\bar{l}a}/x]e^{alN}$ (head of l is a)

We can prove similarly to the above case except that f should annotated as the head of l.

- $(\texttt{con}\ \kappa\ v)^N \to (\kappa{\cdot}v)^N$

$$
\begin{array}{lll}
\multicolumn{3}{l}{(\texttt{con}\ \kappa\ v)^N{:}} \\
= & \texttt{con}\ \kappa\ (\Psi_N(v)) & (\text{def. of } {:}) \\
\to & \kappa{\cdot}(\Psi_N(v)) & \\
= & \Psi_N(\kappa{\cdot}v) & (\text{def. of } \Psi_N) \\
= & (\kappa{\cdot}v){:} & (\text{def. of } {:}) \\
= & (\kappa{\cdot}v)^N{:} & (\text{def. of } {:})
\end{array}
$$

- $(\texttt{decon}\ (\kappa{\cdot}v)^l)^N \to v^{lN}$

$$\begin{array}{lll}
\multicolumn{2}{l}{(\mathtt{decon}\ (\kappa{\cdot}v)^l)^N{:}} & \\
= & \mathtt{decon}\ (\Psi_N((\kappa{\cdot}v)^l)) & \text{(def. of :)}\\
= & \mathtt{decon}\ (\kappa{\cdot}\Psi_N(v^l)) & \text{(Property 1)}\\
\to & \Psi_N(v^l) & \\
= & v^l{:} & \text{(def. of :)}\\
= & v^{lN}{:} & \text{(def. of :)}
\end{array}$$

- $(\mathtt{case}\ (\kappa{\cdot}v)^l\ \kappa\ e_1\ e_2)^N \to e_1^N$

$$\begin{array}{lll}
\multicolumn{2}{l}{(\mathtt{case}\ (\kappa{\cdot}v)^l\ \kappa\ e_1\ e_2)^N{:}} & \\
= & \mathtt{case}\ (\Psi_N((\kappa{\cdot}v)^l))\ \kappa\ [\![e_1]\!]_N\ [\![e_2]\!]_N & \text{(def. of :)}\\
= & \mathtt{case}\ (\kappa{\cdot}\Psi_N(v^l))\ \kappa\ [\![e_1]\!]_N\ [\![e_2]\!]_N & \text{(Property 1)}\\
\to & [\![e_1]\!]_N & \\
= & [\![e_1^N]\!]_N & \text{(def. of } [\![\cdot]\!]_N)\\
\stackrel{*}{\to} & e_1^N{:} & \text{(Lemma 3)}
\end{array}$$

- $(\mathtt{case}\ (\kappa{\cdot}v)^l\ \kappa'\ e_1\ e_2)^N \to e_2^N \quad (\kappa \neq \kappa')$

  We can prove similarly to the above case.

- $(e_1@e_2)^N \to (e_1'@e_2)^N \qquad (e_1 \to e_1')$

$$\begin{array}{lll}
\multicolumn{2}{l}{(e_1@e_2)^N{:}} & \\
= & (e_1{:})@[\![e_2]\!]_N & \text{(def. of :)}\\
\stackrel{+}{\to} & (e_1'{:})@[\![e_2]\!]_N & \text{(I.H.)}\\
= & (e_1'@e_2)^N{:} & \text{(def. of :)}
\end{array}$$

- $(v@e)^N \to (v@e')^N$, $(\mathtt{con}\,\kappa\ e)^N \to (\mathtt{con}\,\kappa\ e')^N$, $(\mathtt{decon}\ e)^N \to (\mathtt{decon}\ e')^N$, $(\mathtt{case}\ e_1\ \kappa\ e_2\ e_3)^N \to (\mathtt{case}\ e_1'\ \kappa\ e_2\ e_3)^N$,

  We can prove similarly to the above case.

- $e^N \to e'^N$

  Then $e \to e'$.

$$\begin{array}{lll}
\multicolumn{2}{l}{e^N{:}} & \\
= & e{:} & \text{(def. of :)}\\
\stackrel{+}{\to} & e'{:} & \text{(I.H.)}\\
= & e'^N{:} & \text{(def. of :)}
\end{array}$$

- $e^C \to e'^C$

$$\begin{array}{lll}
\multicolumn{2}{l}{e^C{:}} & \\
= & e^C{:}\mathtt{c2n}(\tau_e) & \text{(def. of :)}\\
\stackrel{+}{\to} & e'^C{:}\mathtt{c2n}(\tau_e) & \text{(I.H. about :}K)\\
= & e'^C{:} & \text{(def. of :)}
\end{array}$$

- $t^C \to t'^C$

  We can prove similarly to the above e^C case.

For $:K$ cases,

- $((\lambda x.e)^l@v)^C \to [v^{C\bar{l}}/x]e^{lC}$

  Prove this case by induction on the length of l.

  - $((\lambda x.e)^C@v)^C \to [v^{CC}/x]e^{CC}$

    By the consistency, free variable x in e should be annotated as C.

$$\begin{array}{lll}
\multicolumn{2}{l}{((\lambda x.e)^C@v)^C{:}K} & \\
= & \Psi_C((\lambda x.e)^C)@\Psi_C(v)@K & \text{(def. of :}K)\\
= & (\lambda x.[\![e]\!]_C)@\Psi_C(v)@K & \text{(def. of } \Psi_C)\\
\to & ([\Psi_C(v)/x][\![e]\!]_C)@K & \\
= & ([\Psi_C(v^{CC})/x][\![e^{CC}]\!]_C)@K & \\
 & & \text{(def. of } \Psi_C \text{ and } [\![\cdot]\!]_C)\\
= & ([\![[v^{CC}/x]e^{CC}]\!]_C)@K & \text{(Lemma 2)}\\
\stackrel{*}{\to} & [v^{CC}/x]e^{CC}{:}K & \text{(Lemma 3)}
\end{array}$$

  - $((\lambda x.e)^N@v)^C \to [v^{CN}/x]e^{NC}$

    By the consistency, free variable x in e should be annotated as N. And $\lambda x.e$'s type should be $\tau_1 \to \tau_2$.

$$\begin{array}{lll}
\multicolumn{2}{l}{((\lambda x.e)^N@v)^C{:}K} & \\
= & \Psi_C((\lambda x.e)^N)@\Psi_C(v)@K & \text{(def. of :}K)\\
= & (\lambda x.\lambda K.K@(\mathtt{n2c}(\tau_2)@(\Psi_N((\lambda x.e)^N) & \\
 & \quad @(\mathtt{c2n}(\tau_1)@x))))@\Psi_C(v^C)@K & \\
 & & \text{(def. of } \Psi_C)\\
\stackrel{+}{\to} & K@(\mathtt{n2c}(\tau_2)@(\Psi_N((\lambda x.e)^N) & \\
 & \quad @(\mathtt{c2n}(\tau_1)@\Psi_C(v^C)))) & \\
\stackrel{*}{\to} & K@(\mathtt{n2c}(\tau_2)@(\Psi_N((\lambda x.e)^N)@\Psi_N(v^C))) & \\
 & & \text{(def. of } \Psi_N)\\
= & K@(\mathtt{n2c}(\tau_2)@((\lambda x.[\![e]\!]_N)@\Psi_N(v^C))) & \\
 & & \text{(def. of } \Psi_N)\\
\to & K@(\mathtt{n2c}(\tau_2)@([\Psi_N(v^C)/x][\![e]\!]_N)) & \\
= & K@(\mathtt{n2c}(\tau_2)@([\Psi_N(v^{CN})/x][\![e^N]\!]_N)) & \\
 & & \text{(def. of } \Psi_N \text{ and } [\![\cdot]\!]_N)\\
= & K@(\mathtt{n2c}(\tau_2)@([\![[v^{CN}/x]e^N]\!]_N)) & \text{(Lemma 2)}\\
\stackrel{*}{\to} & K@(\mathtt{n2c}(\tau_2)@([v^{CN}/x]e^N{:})) & \text{(Lemma 3)}\\
= & [v^{CN}/x]e^N{:}K & \text{(def. of :}K)\\
= & [v^{CN}/x]e^{NC}{:}K & \text{(def. of :}K)
\end{array}$$

  - $((\lambda x.e)^{lC}@v)^C \to [v^{CC\bar{l}}/x]e^{lCC}$

$$\begin{array}{lll}
\multicolumn{2}{l}{((\lambda x.e)^{lC}@v)^C{:}K} & \\
= & \Psi_C((\lambda x.e)^{lC})@\Psi_C(v)@K & \text{(def. of :}K)\\
= & \Psi_C((\lambda x.e)^l)@\Psi_C(v^C)@K & \text{(def. of } \Psi_C)\\
= & ((\lambda x.e)^l@v^C)^C{:}K & \text{(def. of :}K)\\
\stackrel{+}{\to} & [v^{CC\bar{l}}/x]e^{lC}{:}K & \\
 & & \text{(I.H. about the length of } l)\\
= & [v^{CC\bar{l}}/x]e^{lCC}{:}K & \text{(def. of :}K)
\end{array}$$

  - $((\lambda x.e)^{lN}@v)^C \to [v^{CN\bar{l}}/x]e^{lNC}$

    $\lambda x.e$'s type should be $\tau_1 \to \tau_2$.

$$\begin{array}{lll}
\multicolumn{2}{l}{((\lambda x.e)^{lN}@v)^C{:}K} & \\
= & \Psi_C((\lambda x.e)^{lN})@\Psi_C(v)@K & \text{(def. of :}K)\\
= & (\lambda x.\lambda K.K@(\mathtt{n2c}(\tau_2)@(\Psi_N((\lambda x.e)^{lN}) & \\
 & \quad @(\mathtt{c2n}(\tau_1)@x))))@\Psi_C(v)@K & \\
 & & \text{(def. of } \mathtt{n2c}())\\
\stackrel{+}{\to} & K@(\mathtt{n2c}(\tau_2)@(\Psi_N((\lambda x.e)^{lN}) & \\
 & \quad @(\mathtt{c2n}(\tau_1)@\Psi_C(v)))) & \\
\stackrel{*}{\to} & K@(\mathtt{n2c}(\tau_2)@(\Psi_N((\lambda x.e)^l)@\Psi_N(v^C))) & \\
 & & \text{(def. of } \Psi_N)\\
= & K@(\mathtt{n2c}(\tau_2)@(((\lambda x.e)^l@v^C)^N{:})) & \text{(def. of :)}\\
= & ((\lambda x.e)^l@v^C)^N{:}K & \text{(def. of :}K)\\
\stackrel{+}{\to} & [v^{CN\bar{l}}/x]e^{lN}{:}K & \text{(I.H. about the length of } l)\\
= & [v^{CN\bar{l}}/x]e^{lNC}{:}K & \text{(def. of :}K)
\end{array}$$

By induction, $((\lambda x.e)^l@v)^C{:}K \stackrel{+}{\to} [v^{C\bar{l}}/x]e^{lC}{:}K$

- $((\mathtt{fix}\ f\ \lambda x.e)^l@v)^C \to [(\mathtt{fix}\ f\ \lambda x.e)^a/f][v^{C\bar{l}}/x]e^{lC}$ (head of l is a)

  We can prove similarly to the above case except that f should annotated as the head of l.

- $(\mathtt{con}\ \kappa\ v)^C \to (\kappa{\cdot}v)^C$

$$
\begin{array}{lll}
(\mathtt{con}\,\kappa\,v)^C{:}K & & \\
\quad = & K@(\mathtt{con}\,\kappa\,\Psi_C(v)) & \text{(def. of :}K) \\
\quad \to & K@(\kappa{\cdot}\Psi_C(v)) & \\
\quad = & K@(\Psi_C((\kappa{\cdot}v)^C)) & \text{(def. of }\Psi_C) \\
\quad = & (\kappa{\cdot}v)^C{:}K & \text{(def. of :}K)
\end{array}
$$

- $(\mathtt{decon}\;(\kappa{\cdot}v)^l)^C \to v^{lC}$

$$
\begin{array}{lll}
(\mathtt{decon}\;(\kappa{\cdot}v)^l)^C{:}K & & \\
\quad = & K@(\mathtt{decon}\;\Psi_C((\kappa{\cdot}v)^l)) & \text{(def. of :}K) \\
\quad = & K@(\mathtt{decon}\;(\kappa{\cdot}\Psi_C(v^l))) & \text{(Property 1)} \\
\quad \to & K@\Psi_C(v^l) & \\
\quad = & v^l{:}K & \text{(def. of :}K) \\
\quad = & v^{lC}{:}K & \text{(def. of :}K)
\end{array}
$$

- $(\mathtt{case}\;(\kappa{\cdot}v)^l\;\kappa\;e_1\;e_2)^C \to e_1^C$

$$
\begin{array}{lll}
(\mathtt{case}\;(\kappa{\cdot}v)^l\;\kappa\;e_1\;e_2)^C{:}K & & \\
\quad = & \mathtt{case}\;(\Psi_C((\kappa{\cdot}v)^l))\;\kappa\;([\![e_1]\!]_C@K)\;([\![e_2]\!]_C@K) & \\
 & & \text{(def. of :)} \\
\quad = & \mathtt{case}\;(\kappa{\cdot}\Psi_C(v^l))\;\kappa\;([\![e_1]\!]_C@K)\;([\![e_2]\!]_C@K) & \\
 & & \text{(Property 1)} \\
\quad \to & [\![e_1]\!]_C@K & \\
\quad = & [\![e_1^C]\!]_C@K & \text{(def. of }[\![\cdot]\!]_C) \\
\quad \xrightarrow{*} & e_1^C{:}K & \text{(Lemma 3)}
\end{array}
$$

- $(\mathtt{case}\;(\kappa{\cdot}v)^l\;\kappa'\;e_1\;e_2)^C \to e_1^C \quad (\kappa \neq \kappa')$

  We can prove similarly to the above case.

- $(e_1@e_2)^C \to (e_1'@e_2)^C \qquad (e_1 \to e_1')$

$$
\begin{array}{lll}
(e_1@e_2)^C{:}K & & \\
\quad = & e_1{:}\lambda f.[\![e_2]\!]_C@(\lambda v.f@v@K) & \text{(def. of :)} \\
\quad \xrightarrow{+} & e_1'{:}\lambda f.[\![e_2]\!]_C@(\lambda v.f@v@K) & \text{(I.H.)} \\
\quad = & (e_1'@e_2)^C{:}K & \text{(def. of :)}
\end{array}
$$

- $(v@e)^C \to (v@e')^C$, $(\mathtt{con}\,\kappa\,e)^C \to (\mathtt{con}\,\kappa\,e')^C$, $(\mathtt{decon}\,e)^C \to (\mathtt{decon}\,e')^C$, $(\mathtt{case}\;e_1\;\kappa\;e_2\;e_3)^C \to (\mathtt{case}\;e_1'\;\kappa\;e_2\;e_3)^C$,

  We can prove similarly to the above case.

- $e^C \to e'^C$

  Then $e \to e'$.

$$
\begin{array}{lll}
e^C{:}K & & \\
\quad = & e{:}K & \text{(def. of :}K) \\
\quad \xrightarrow{+} & e'{:}K & \text{(I.H.)} \\
\quad = & e'^C{:}K & \text{(def. of :}K)
\end{array}
$$

- $e^N \to e'^N$

$$
\begin{array}{lll}
e^N{:}K & & \\
\quad = & K@(\mathtt{n2c}(\tau_e)@(e^N{:})) & \text{(def. of :}K) \\
\quad \xrightarrow{+} & K@(\mathtt{n2c}(\tau_e)@(e'^N{:})) & \text{(I.H.)} \\
\quad = & e'^N{:}K & \text{(def. of :}K)
\end{array}
$$

- $t^N \to t'^N$

  We can prove similarly to the above e^N case.

$\square$