

# Eager Normal Form Bisimulation

Soren Lassen  
Google, Inc.  
soren@google.com

## Abstract

*This paper describes two new bisimulation equivalences for the pure untyped call-by-value  $\lambda$ -calculus, called enf bisimilarity and enf bisimilarity up to  $\eta$ . They are based on eager reduction of terms to eager normal form (enf), analogously to co-inductive bisimulation characterizations of Lévy-Longo tree equivalence and Böhm tree equivalence (up to  $\eta$ ). We argue that enf bisimilarity is the call-by-value analogue of Lévy-Longo tree equivalence. Enf bisimilarity (up to  $\eta$ ) is the congruence on source terms induced by the call-by-value CPS transform and Böhm tree equivalence (up to  $\eta$ ) on target terms. Enf bisimilarity and enf bisimilarity up to  $\eta$  enjoy powerful bisimulation proof principles which, among other things, can be used to establish a retraction theorem for the call-by-value CPS transform.*

## 1. Introduction

This paper adapts underlying principles from Böhm tree and Lévy-Longo tree equivalences to the call-by-value  $\lambda$ -calculus. Böhm trees and Lévy-Longo trees are two kinds of infinitary normal forms of  $\lambda$ -terms. They are used extensively in the study of conventional call-by-name  $\lambda$ -calculus for local structure characterization of  $\lambda$ -models and for constructing syntactical  $\lambda$ -models [2, 14, 19]. The induced term congruences, Böhm tree equivalence (up to  $\eta$ ) and Lévy-Longo tree equivalence, relate terms with identical trees (up to infinite  $\eta$  expansion).

The calculation of the Böhm tree of a  $\lambda$ -term can be described as a recursive process that head reduces the term to head normal form (hnf) and recursively calculates the Böhm trees of the subterms of the hnf. The calculation of the Lévy-Longo tree is similar, using weak head reduction to weak head normal form (whnf) instead of head reduction. The recursive calculation of trees and the recursive process of testing the identity or equivalence of trees can be combined, “deforested” as it were, into bisimulation characterizations of the induced term congruences with no explicit reference to trees [26, 13]. These bisimulation characteri-

zations are based on recursive head reduction or weak head reduction of terms and matching hnfs or whnfs. In this paper we will refer to these term equivalences collectively as “normal form bisimulation” equivalences. The enf bisimulation equivalences introduced in this paper are new members of this family of equivalences. Contrary to the other normal form bisimulation equivalences, there are no obvious tree representations of the enf bisimulation equivalence classes.

Whereas conventional call-by-name  $\lambda$ -calculus is based on the  $\beta$  notion of reduction, the call-by-value  $\lambda$ -calculus is based on the more restricted  $\beta_v$  notion on reduction and, instead of head reduction or weak head reduction (normal order reduction), an appropriate reduction strategy is eager reduction (applicative order reduction) which reduces a function’s argument to a value before application. (In effect, we will use “call-by-value” and “eager” synonymously.) We will describe the grammar of the eager normal forms (enfs), that is, the terms without any eager redex. Our description of the eager normal forms appear to be new.

We will define normal form bisimulation congruences based on eager reduction to enf, called enf bisimilarity and enf bisimilarity up to  $\eta$ . They include the usual  $\beta_v$  and  $\eta_v$  equations for the call-by-value  $\lambda$ -calculus and also Moggi’s untyped computational  $\lambda$ -calculus theory [17]. Furthermore, they equate all terms without enf, that is, terms whose eager reduction sequences diverge.

Enf bisimilarity (up to  $\eta$ ) is defined co-inductively and their associated bisimulation proof principles are very useful for proving the equivalence of recursive functions and higher-order data structures, some times more useful than other operationally defined term equivalences such as applicative bisimilarity [1] or CIU equivalence [15], because enf bisimulation does not involve any quantification over all possible closed function arguments or “closed instantiations” or “uses”.

The enf bisimulation and enf bisimulation up to  $\eta$  proof principles lead to straightforward proofs of program equivalences in the call-by-value  $\lambda$ -calculus that are difficult to establish by other means. We demonstrate this by proving the equivalence of different call-by-value fixed point combi-

nators, the syntactic minimal invariance equation, and a new retraction theorem for the call-by-value CPS transform.

Contrary to applicative bisimilarity and CIU equivalence (but like Lévy-Longo equivalence and Böhm tree equivalence), enf bisimilarity and enf bisimilarity up to  $\eta$  are not “operationally extensional” and do not coincide with contextual equivalence.

We claim that the enf bisimulation congruences are call-by-value analogues to Lévy-Longo tree equivalence. In support of this we show that they are the induced term congruences on source terms from the call-by-value CPS transform and from Böhm tree equivalence and Böhm tree equivalence up to  $\eta$  on target terms. This result is analogous to the relationship between Lévy-Longo tree equivalence and the call-by-name CPS transform [3]. Moreover, Lévy-Longo tree equivalence and enf bisimilarity coincide on terms in the target of the CPS transforms.

## 1.1. Related work

The definitions of the enf bisimilarities are modeled over the bisimulation characterizations of Lévy-Longo and Böhm tree equivalences in [26, 13].

The CPS translation theorems for the enf bisimilarities are call-by-value versions of Boudol’s translation theorem for Lévy-Longo tree equivalence and a call-by-name CPS transform [3]. One difference is that our proofs are co-inductive, utilizing the bisimulation definitions/characterizations of enf bisimilarity (up to  $\eta$ ) and Böhm tree equivalence (up to  $\eta$ ).

We believe our retraction theorem for the call-by-value CPS transform and enf bisimilarity up to  $\eta$  is new. Earlier retraction theorems [16, 7, 10] are for simply typed  $\lambda$ -calculi. Kucan [10] describes the difficulties of extending type-based proofs to fixed point recursion and recursive types.

Our CPS transform is a slight modification of Plotkin’s call-by-value CPS transform [23]. We show that the modified CPS transform satisfies a translation theorem that relates, one-to-one, the equational theory of Moggi’s untyped computational  $\lambda$ -calculus [17] on source terms, without (the *let* construct and) the  $\eta_v$  equation, to  $\beta$  equivalence between target terms. The theorem improves Plotkin’s call-by-value translation theorem which states that  $\beta_v$  equations are preserved by the CPS transform. (There’s a curious, incidental parallel with the way Hatcliff and Danvy [8] added an  $\eta$  redex to Plotkin’s call-by-name CPS transform to fix the call-by-name translation theorem. Our modification of the call-by-value CPS transform adds an  $\eta_v$  redex.) Our proof is structured similarly to the equational correspondence proof for Fischer’s call-by-value CPS transform by Sabry and Felleisen [24]. Compared to *op.cit.*, we prove a translation theorem not only for  $\beta\eta$  equivalence between target terms

but also for  $\beta$  equivalence. Sabry and Wadler [25] prove that an optimized form of Plotkin’s CPS transform to a refined CPS calculus forms a reflection (a strong correspondence between reductions in the source calculus and the target calculus) but their CPS calculus is not a sub-calculus of the  $\lambda$ -calculus, because it uses a tailored notion of reduction, so their results do not directly yield a CPS translation theorem for the CPS transform into the  $\lambda$ -calculus. In our proof of the CPS translation theorem for the equational theories of the computational  $\lambda$ -calculus and  $\beta$  equivalence, we use an inverse CPS transform that is an “unoptimized” variant of Danvy’s [4] and Sabry and Wadler’s [25].

**Outline** The paper is organized as follows. §2 specifies the syntax and eager operational semantics (on open terms) of the pure untyped call-by-value  $\lambda$ -calculus. §3 defines enf bisimilarity. §4 introduces a call-by-value CPS transform, extends Plotkin’s indifference and simulation results to open terms, and presents a CPS translation theorem for enf bisimilarity and Böhm tree equivalence. §5 extends enf bisimilarity up to  $\eta$  and the CPS translation theorem to Böhm tree equivalence up to  $\eta$ . §6 establishes a CPS retraction theorem. §7 suggests future work. Appendix A improves Plotkin’s call-by-value CPS translation theorem.

## 2. The pure untyped call-by-value $\lambda$ -calculus

The terms of the pure untyped  $\lambda$ -calculus are variables ( $x, y, z$ ),  $\lambda$ -abstractions, and applications:

$$\text{Terms } t ::= x \mid \lambda x.t \mid t_1 t_2$$

The scope of a  $\lambda$ -abstraction extends as far to the right as possible and application associates to the left, so  $\lambda x.t_1 t_2 t_3$  means  $\lambda x.((t_1 t_2) t_3)$ . Notation  $\lambda x.y.t$  is short for  $\lambda x.\lambda y.t$ .

As is usual, we identify terms up to renaming of bound variables. For any syntactic phrase  $\phi$ , let  $\text{FV}(\phi)$  denote its set of free variables.  $\text{FV}(\phi_1, \phi_2)$  is shorthand for  $\text{FV}(\phi_1) \cup \text{FV}(\phi_2)$ . We say  $\phi$  is closed if  $\text{FV}(\phi)$  is empty.

To define the call-by-value operational semantics we introduce the syntactic categories of values and evaluation contexts.

$$\text{Values } v ::= x \mid \lambda x.t$$

$$\text{Evaluation contexts } E ::= [] \mid Et \mid vE$$

An evaluation context is a term with a single hole  $[]$ . We write  $E[t]$  for the term obtained by substituting of  $t$  for  $[]$  in  $E$ . Every term  $t$  is either a value or has a unique decomposition  $t = E[v_1 v_2]$ .

We write  $t[v/x]$  for the capture-free substitution of value  $v$  for all free occurrences of variable  $x$  in term  $t$ .

Let the call-by-value operational semantics be specified in terms of the following “eager” reduction relation:

$$E[(\lambda x. t) v] \rightarrow_e E[t[v/x]]$$

Let  $\rightarrow_e$  denote the reflexive transitive closure of  $\rightarrow_e$ .

The eager reduction is defined for arbitrary terms, not just closed terms. This is important for the definition and theory of “normal form bisimulation” in the sequel.

**Lemma 2.1.**  $t[v/x] \rightarrow_e t'[v/x]$  and  $E[t] \rightarrow_e E[t']$  if  $t \rightarrow_e t'$ .

An eager normal form (enf) is a terms that is a normal form for eager reduction. The enfs are the terms generated by the grammar:

$$\text{Eager normal forms (enfs)} \quad e ::= v \mid E[xv]$$

If  $t \rightarrow_e e$  we say that  $t$  has enf  $e$ . Otherwise there is an infinite reduction sequence  $t \rightarrow_e^\omega$ .

### 3. Enf bisimulation

Let an enf bisimulation be a relation  $R$  between terms such that, whenever  $t R t'$ , one of the following is true:

(enf.1)  $t$  and  $t'$  have no enf.

(enf.2)  $t \rightarrow_e E[xv]$  and  $t' \rightarrow_e E'[xv']$  for some  $E, E'$ ,  $v$  and  $v'$  such that  $v R v'$  and  $E[z] R E'[z]$  for some  $z \notin \text{FV}(E) \cup \text{FV}(E')$ .

(enf.3)  $t \rightarrow_e x$  and  $t' \rightarrow_e x$  for some variable  $x$ .

(enf.4)  $t \rightarrow_e \lambda y. t_0$  and  $t' \rightarrow_e \lambda y. t'_0$  for some  $y, t_0$  and  $t'_0$  such that  $t_0 R t'_0$ .

Enf bisimilarity, written  $\approx_e$ , is the largest enf bisimulation. It exists because the union of enf bisimulations is again an enf bisimulation.

This is a co-inductive definition. The associated co-induction proof principle says that you can prove terms enf bisimilar by exhibiting an enf bisimulation that relates the terms.

Observe that the definition of enf bisimilarity does not involve any quantification over function arguments, substitutions, or evaluation contexts. Rather it is defined in terms of recursive evaluation to enf, similarly to the definitions of Böhm tree equivalence and Lévy-Longo tree equivalence.

**Example 3.1.** We can prove the equivalence of call-by-value versions of Curry’s and Turing’s fixed point combinators:

$$Y_v = \lambda x. \Delta \Delta, \text{ where } \Delta = \lambda z. x \lambda y. z z y, \\ \Theta_v = (\lambda z x. x \lambda y. z z x y) (\lambda z x. x \lambda y. z z x y),$$

by checking that the relation

$$R = \{(Y_v, \Theta_v), (\Delta \Delta, x \lambda y. \Theta_v x y), \\ (\lambda y. \Delta \Delta y, \lambda y. \Theta_v x y), (z, z), \\ (\Delta \Delta y, \Theta_v x y), (z y, z y)\}$$

is an enf bisimulation. This bisimulation relation is found by starting with the relation  $\{(Y_v, \Theta_v)\}$  and repeatedly reducing related terms to enfs and adding to the relation any terms needed to satisfy clauses (enf.2) through (enf.4).

This is an example of an equivalence that cannot readily be established using other operational methods—such as applicative bisimulation or the CIU theorem—but, as illustrated, is straightforward to prove using normal form bisimulation.

*Remark 1.* Rather than  $Y_v v$  or  $\Theta_v v$ , a more convenient fixed point encoding for the symbolic manipulations in the sequel is  $\text{fix}[v] = \lambda x. \Theta_v v x$ , because  $\text{fix}[v] x \rightarrow_e v \text{fix}[v] x$ .

Enf bisimilarity includes the  $\beta_v$  equation,

$$(\lambda x. t) v \approx_e t[v/x] \quad (\beta_v)$$

because  $\{((\lambda x. t) v, t[v/x])\} \cup Id$  is an enf bisimulation, where  $Id = \{(t, t) \mid t \text{ is a term}\}$  is the identity relation.

Enf bisimilarity also includes the equations of Moggi’s untyped computational  $\lambda$ -calculus (a variant without  $let$ ), apart from  $\eta_v$ ,

$$(\lambda x. x) t \approx_e t \quad (id)$$

$$(\lambda x_1. t) ((\lambda x_2. t_1) t_2) \approx_e (\lambda x_2. (\lambda x_1. t) t_1) t_2, \\ \text{if } x_2 \notin \text{FV}(t) \quad (comp)$$

$$t_1 t_2 \approx_e (\lambda x_1. x_1 t_2) t_1, \text{ if } x_1 \notin \text{FV}(t_2) \quad (let.1)$$

$$t_1 t_2 \approx_e (\lambda x_2. t_1 x_2) t_2, \text{ if } x_2 \notin \text{FV}(t_1) \quad (let.2)$$

(*id*) holds because  $\{((\lambda x. x) t, t) \mid t \text{ is a term}\} \cup Id$  is an enf bisimulation, which follows by analysis of the eager reduction behaviour of  $t$ . The proofs of the other equations are analogous.

Let  $\beta_c$  denote the union of  $\beta_v$ , *id*, *comp*, *let.1* and *let.2*. Then  $\beta_c \eta_v$  are the equations of the untyped computational  $\lambda$ -calculus.

It is easy to show that enf bisimilarity is an equivalence relation (reflexive, transitive, and symmetric) by co-induction. In the next section we will show that enf bisimilarity is a congruence.

Contextual equivalence for the call-by-value  $\lambda$ -calculus can be defined as the largest congruence relation which relates two closed terms  $t$  and  $t'$  only if either both  $t$  and  $t'$  have no enf or both  $t$  and  $t'$  have enfs (cf. [12]). Since enf bisimilarity is a congruence, it is immediate from its definition that it is included in contextual equivalence. The inclusion is strict, because enf bisimilarity does not share the

“operational extensionality” property of contextual equivalence: terms  $t$  and  $t'$  are contextually equivalent if, for all closed values  $v$ ,  $t[v/x]$  and  $t'[v/x]$  are contextually equivalent (see [15, 21, 6]).

**Example 3.2.** Let  $\Omega = (\lambda x. x x) (\lambda x. x x)$  and  $I = \lambda x. x$ . For every closed value  $v$ ,

- (1)  $v \simeq_e \lambda y. v y$ , but  $x \not\simeq_e \lambda y. x y$ ,
- (2)  $\Omega \simeq_e (\lambda y. \Omega) (v I)$ , but  $\Omega \not\simeq_e (\lambda y. \Omega) (x I)$ ,
- (3)  $v I \simeq_e (\lambda y. v I) (v I)$ , but  $x I \not\simeq_e (\lambda y. x I) (x I)$ .

## 4. CPS transform

In this section we will show that enf bisimilarity is the equivalence relation induced by the call-by-value CPS transform and Böhm tree equivalence, that is, terms are enf bisimilar if and only if their CPS transforms are Böhm tree equivalent. We will use this result to derive that enf bisimilarity is a congruence from the congruence of Böhm tree equivalence.

The call-by-value CPS transform  $T$  maps terms into  $\lambda$ -abstractions:

$$\begin{aligned} T(x) &= \lambda k. k x, \\ T(\lambda x. t) &= \lambda k. k \lambda x. T(t), \\ T(t_1 t_2) &= \lambda k. T(t_1) \lambda x_1. T(t_2) \lambda x_2. x_1 x_2 \lambda x. k x, \end{aligned}$$

where  $k$ ,  $x_1$  and  $x_2$  are distinct, fresh variables that do not occur free in  $x$ ,  $\lambda x. t$  and  $t_1 t_2$ .

*Remark 2.*  $T$  is Plotkin’s call-by-value CPS transform [23], modified by an added  $\eta_v$  redex  $\lambda x. k x$  around  $k$  in the last clause of the definition. This modification is necessary to ensure  $T((\lambda x. x) t) \simeq_e T(t)$ , which is necessary for theorem 4.10 because  $(\lambda x. x) t \simeq_e t$ . The extra  $\eta_v$  redex also “fixes” Plotkin’s call-by-value CPS translation theorem—the modified  $T$  satisfies  $t =_{\beta_c} t'$  iff  $T(t) =_{\beta_v} T(t')$  iff  $T(t) =_{\beta_c} T(t')$  iff  $T(t) =_{\beta} T(t')$ . See appendix A.

We first extend Plotkin’s indifference and simulation theorems to our modified call-by-value CPS transform. We also extend the indifference theorem and simulation lemmas to open terms for use in the proofs of the CPS translation and retraction theorems in the sequel.

### 4.1. Indifference

The image of the CPS transform consists of lambda terms that have values in the argument position in every application. (The same is true of the call-by-name CPS transform.) In other words, the target of the CPS transform(s) is

the following evaluation order independent (or continuation passing style)  $\lambda$ -terms:

$$\begin{aligned} \text{EOI Terms } s &::= u \mid s u \\ \text{EOI Values } u &::= x \mid \lambda x. s \end{aligned}$$

The evaluation order independent (EOI) evaluation contexts are applicative contexts with value arguments:

$$\text{EOI Evaluation contexts } D ::= [] \mid D u$$

The EOI terms are closed under substitution of EOI values for free variables and are closed under  $\beta$  reduction and, in particular, eager reduction.

Moreover, observe that weak head reduction coincides with eager reduction on EOI terms and that the EOI enfs coincide with the EOI weak head normal forms (whnfs), which we call the EOI weak normal forms:

$$\text{EOI Weak normal forms (wnfs) } d ::= \lambda x. s \mid D[x]$$

*Remark 3.* The EOI syntax is a rather loose description of the target of the call-by-value CPS transform. It also includes the target of other call-by-value and call-by-name CPS transforms from the literature. One drawback of the EOI syntax is that it is not closed under  $\eta$  reduction. In appendix A, we refine the EOI syntax to a smaller CPS syntax that more accurately describes the target of our call-by-value CPS transform.

**Theorem 4.1 (Indifference).** *Eager reduction coincides with weak head reduction on  $T(t)u$ , for arbitrary terms  $t$  and EOI values  $u$ .*

**Corollary 4.2.** *Enf bisimilarity coincides with Lévy-Longo tree equivalence on EOI terms.*

*Proof.* If we compare the definition of enf bisimulation with the definition of whnf bisimulation in [13] (called open applicative bisimulation in [26]), we see that the two coincide on EOI terms. Therefore the greatest enf bisimulation on EOI terms is also the greatest whnf bisimulation on EOI terms. The former is enf bisimilarity, restricted to EOI terms, and the latter is Lévy-Longo tree equivalence, restricted to EOI terms [26, 13].  $\square$

### 4.2. Simulation

We now extend Plotkin’s simulation theorem to open terms. The definitions and proofs are similar to Plotkin’s.

We use an auxiliary transformation,  $\Psi$ , which maps values into EOI values,

$$\Psi(x) = x, \quad \Psi(\lambda x. t) = \lambda x. T(t).$$

**Lemma 4.3.**  $T(t[v/x]) = T(t)[\Psi(v)/x]$ .

*Proof.* By structural induction on  $t$ .  $\square$

We also define a ternary relation,  $t : v : t'$ , between terms, values, and terms. It relates a direct-style term  $t$  and a “continuation”  $v$  to a continuation-passing style term  $t'$ , where  $t'$  is obtained from  $T(t)v$  by, on one hand, reducing some “administrative” redexes (like Plotkin’s “colon translation”) and, on the other hand, adding some  $\eta_v$  redexes around “continuations”. These  $\eta_v$  redexes are necessary to capture the propagation of the extra  $\eta_v$  redex in the definition of  $T$  (compared to Plotkin’s CPS transform). The ternary relation is defined inductively by the rules:

$$\frac{v' \eta_v^* v}{v_0 : v : v' \Psi(v_0)}$$

$$\frac{v' \eta_v^+ v \quad x \notin \text{FV}(v)}{(v_1 v_2) : v : \Psi(v_1) \Psi(v_2) v'}$$

$$\frac{(t_1 t_2) : \lambda x_2. \Psi(v_1) x_2 v' : t \quad v' \eta_v^+ v \quad x_2 \notin \text{FV}(v_1, v)}{(v_1 (t_1 t_2)) : v : t}$$

$$\frac{(t_1 t_2) : \lambda x_1. T(t_3) \lambda x_2. x_1 x_2 v' : t \quad v' \eta_v^+ v \quad x_1 \notin \text{FV}(t_3, v) \quad x_2 \notin \text{FV}(v)}{((t_1 t_2) t_3) : v : t}$$

where  $\eta_v^+$  and  $\eta_v^*$  denote the transitive closure and the reflexive, transitive closure, respectively, of the relation

$$\eta_v = \{(\lambda x. v x, v) \mid x \notin \text{FV}(v)\}.$$

If  $t : u : t'$  and  $u$  is an EOI value then  $t'$  is an EOI term.

**Lemma 4.4.** *For all  $t$  and  $v$ , there exists  $t'$  such that  $t : v : t'$  and  $T(t)v \rightarrow_e t'$ .*

*Proof.* By structural induction on  $t$ .  $\square$

**Lemma 4.5.**

- (1)  $t \rightarrow_e^\omega$  implies  $T(t)v \rightarrow_e^\omega$ .
- (2)  $t \rightarrow_e v'$  implies  $T(t)v \rightarrow_e v \Psi(v')$ .
- (3)  $t \rightarrow_e E[xv']$  implies there exist  $v''$  and  $\lambda z. t'$  such that  $v'' \eta_v^* \lambda z. t'$ ,  $z \notin \text{FV}(E, v)$ ,  $E[z] : v : t'$ , and  $T(t)v \rightarrow_e x \Psi(v') v''$ .

**Theorem 4.6 (Simulation).** *If  $t$  is closed then  $T(t)I \rightarrow_e u$  iff there exists  $v$  such that  $t \rightarrow_e v$  and  $\Psi(v) = u$ .*

*Proof.* Follows from lemma 4.5 and the fact that eager reduction is deterministic.  $\square$

### 4.3. Translation

**Lemma 4.7.**  $t \approx_e t'$  implies  $T(t) \approx_e T(t')$ .

*Proof.* Let  $R = R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5$ , where

$$R_1 = \{(T(t), T(t')) \mid t \approx_e t'\},$$

$$R_2 = \{(t, t') \mid (\lambda k. t, \lambda k. t') \in R_1\},$$

$$R_3 = \{(\Psi(v), \Psi(v')) \mid v \approx_e v'\},$$

$$R_4 = \{(v, v') \mid t_0 \approx_e t'_0, v \eta_v^* \lambda z. t, t_0 : k : t, v' \eta_{v'}^* \lambda z. t', t'_0 : k : t'\},$$

$$R_5 = \{(t, t') \mid (\lambda z. t, \lambda z. t') \in R_4\}.$$

Check that  $R$  is an enf bisimulation by analysis of the eager reduction behaviour of terms, using lemma 4.4 and 4.5.  $\square$

**Corollary 4.8.**  $t \approx_e t'$  implies  $T(t)$  and  $T(t')$  are Lévy-Longo tree equivalent and  $T(t)$  and  $T(t')$  are Böhm tree equivalent.

*Proof.* Enf bisimilarity coincides with Lévy-Longo tree equivalence on EOI terms and Lévy-Longo tree equivalence is included in Böhm tree equivalence.  $\square$

**Lemma 4.9.**  $t \approx_e t'$  if  $T(t)$  and  $T(t')$  are Böhm tree equivalent.

*Proof outline.* Check that

$$\{(t, t') \mid T(t) \text{ and } T(t') \text{ are Böhm tree equivalent}\} \cup \{(t_0, t'_0) \mid t_0 : k : t, t'_0 : k : t', t \text{ and } t' \text{ are Böhm tree equivalent}\}$$

is an enf bisimulation using theorem 4.1 and lemma 4.5.  $\square$

Altogether we have the following translation result:

**Theorem 4.10 (Translation,  $\approx_e$ ).**

$$t \approx_e t' \text{ iff } T(t) \approx_e T(t') \\ \text{iff } T(t) \text{ and } T(t') \text{ are Lévy-Longo tree equivalent} \\ \text{iff } T(t) \text{ and } T(t') \text{ are Böhm tree equivalent.}$$

The theorem says that enf bisimilarity is the induced equivalence by the CPS transform from Lévy-Longo tree equivalence and it is also the induced equivalence from Böhm tree equivalence.

Finally, we derive that enf bisimilarity is a congruence:

**Theorem 4.11.** *Enf bisimilarity is a congruence.*

*Proof.* This is a consequence of (i) that the CPS transform is compositional, (ii) that Böhm tree equivalence is a congruence [2, 13], and (iii) the fact from theorem 4.10 that  $t \approx_e t'$  iff  $T(t)$  and  $T(t')$  are Böhm tree equivalent. (Or replace Böhm tree equivalence with Lévy-Longo tree equivalence in (ii) and (iii), because Lévy-Longo tree equivalence is also a congruence [18].)  $\square$

Recall that  $\approx_e$  includes the  $\beta_v$  and  $\beta_c$  equations. Since  $\approx_e$  is a congruence, we can conclude that  $\approx_e$  includes the equational theories generated by these equations (that is, their congruence closure), which we denote by  $=_{\beta_v}$  and  $=_{\beta_c}$ , respectively.

## 5. $\eta$ equivalence

Example 3.2(1) shows that enf bisimilarity does not include the  $\eta_v$  equation,  $x \not\approx_e \lambda y. x y$ . We shall now define a larger (more coarse-grained) variant of enf bisimilarity that includes  $\eta_v$ . Rather than just relate  $x$  to  $\lambda y. x y$ , we relate  $x$  to any  $\lambda$ -abstraction  $\lambda y. t$  where  $t \rightarrow_e E[x v]$  for some evaluation context  $E$  equivalent to  $[]$  and some value  $v$  equivalent to  $y$ .

Let an enf bisimulation up to  $\eta$  be defined the same as an enf bisimulation, but with two extra clauses:

(enf.5)  $t \rightarrow_e x$  and  $t' \rightarrow_e \lambda y. t_0$  for some  $x, y$  and  $t_0$  such that  $t_0 \rightarrow_e E[x v]$  and  $y R v$  and  $z R E[z]$ , for some  $E, v$  and  $z \notin \text{FV}(E)$ .

(enf.6)  $t \rightarrow_e \lambda y. t_0$  and  $t' \rightarrow_e x$  for some  $x, y$  and  $t_0$  such that  $t_0 \rightarrow_e E[x v]$  and  $v R y$  and  $E[z] R z$ , for some  $E, v$  and  $z \notin \text{FV}(E)$ .

Let enf bisimilarity up to  $\eta$ , written  $\approx_{e\eta}$ , be the greatest enf bisimulation up to  $\eta$ .

Clearly, every enf bisimulation is also an enf bisimulation up to  $\eta$ . Therefore  $\approx_e \subseteq \approx_{e\eta}$ .

Like for  $\approx_e$ , it is easy to show, by co-induction, that  $\approx_{e\eta}$  is a preorder. In theorem 5.2 we will show that  $\approx_{e\eta}$  is also a congruence, which implies that it is included in contextual equivalence. Again the inclusion is strict, because  $\Omega \not\approx_{e\eta} (\lambda y. \Omega) (x I)$  and  $x I \not\approx_{e\eta} (\lambda y. x I) (x I)$ ; see example 3.2(2)–(3).

**Example 5.1.**  $\lambda y. x y \approx_{e\eta} x$ . To see this, check that  $R = \{(\lambda y. x y, x), (y, y)\}$  is an enf bisimulation up  $\eta$  by observing that  $x y \rightarrow_e x y = E[x v]$ , where  $E = []$  and  $v = y$ , and that  $v = y R y$  and  $E[y] = y R y$ .

The next example illustrates the full generality of the definition of enf bisimulation up to  $\eta$ . It is an example of infinite  $\eta_v$  expansion, both of the function argument and result. (Wadsworth's example of infinite  $\eta$  expansion,  $\lambda x. x =_{\eta_\infty} Y \lambda j. x y. x (j y)$ , only  $\eta$  expands the argument.)

**Example 5.2.**  $\lambda x. x \approx_{e\eta} P$ , where  $P$  is the function given by the recursive equation  $P x x' = P (x (P x'))$ , that is,  $P = \text{fix}[\lambda p x x'. p (x (p x'))]$ . To show  $\lambda x. x \approx_{e\eta} P$ , check that the relation

$$\begin{aligned} & \{ (\lambda x. x, P), (x, (\Theta_v \lambda p x x'. p (x (p x')))) x), \\ & (x', \lambda x. P (x' (P x))), (x, P x) \\ & \mid x \text{ and } x' \text{ are any two distinct variables} \} \end{aligned}$$

is an enf bisimulation.

This example is the syntactic minimal invariance equation from [22, 11]. Our normal form bisimulation proof is significantly simpler than the proofs in *op.cit.*

Let us call terms Böhm tree equivalent “up to  $\eta$ ” if their Böhm trees are identical up to possibly infinite  $\eta$  expansion. This equivalence is the maximum sensible  $\lambda$ -theory [2].

**Theorem 5.1** (Translation,  $\approx_{e\eta}$ ).  $t \approx_{e\eta} t'$  iff  $T(t)$  and  $T(t')$  are Böhm tree equivalent up to  $\eta$ .

*Proof outline.* By co-induction. The ‘if’ and ‘only if’ implications are similar to the proofs of lemma 4.9 and lemma 4.7, respectively.  $\square$

Contrary to the Translation Theorem 4.10 for  $\approx_e$ , it is not the case that  $t \approx_{e\eta} t'$  iff  $T(t) \approx_{e\eta} T(t')$ . In general,  $t \approx_{e\eta} t'$  does not imply  $T(t) \approx_{e\eta} T(t')$ .

**Example 5.3.**  $x \approx_{e\eta} \lambda y. x y$  but  $T(x) \not\approx_{e\eta} T(\lambda y. x y)$ . In fact,  $T(x)$  and  $T(\lambda y. x y)$  are not even contextually equivalent: Let  $C = (\lambda x. []) (\lambda y. \Omega) (\lambda x. x I)$ , then  $C[T(x)] \rightarrow_e^\omega$  and  $C[T(\lambda y. x y)] \rightarrow_e \lambda z. (\lambda y. \Omega) I z$ .

**Theorem 5.2.** *Enf bisimilarity up to  $\eta$  is a congruence.*

*Proof.* Follows from theorem 5.1 and the congruence of Böhm tree equivalence up to  $\eta$  [9, 27, 13].  $\square$

Recall that  $=_{\beta_c}$  is included in enf bisimilarity. Since enf bisimilarity up to  $\eta$  is a congruence and includes enf bisimilarity and the  $\eta_v$  equation, it includes  $=_{\beta_c \eta_v}$ , the equational theory of the untyped computational  $\lambda$ -calculus.

## 6. CPS Retraction Theorem

The CPS retraction theorem [16, 10] says that the inverse CPS transform is  $\lambda$ -definable. It can also be read as an equational formulation of the CPS simulation theorem.

Let  $F$  and  $G$  be given by the mutually recursive equations  $F x x' k = k (F (x (G x')))$  and  $G y y' = y (F y') G$ , that is,

$$\begin{aligned} F &= \text{fix}[\lambda f x x' k. k (f (x (\text{fix}[\lambda g y y'. y (f y') g] x')))] \\ G &= \text{fix}[\lambda g y y'. y (\text{fix}[\lambda f x x' k. k (f (x (g x')))] y') g] \end{aligned}$$

$F$  translates DS (direct style) values to CPS values and  $G$  translates CPS values to DS values.

The proofs below use of the following notation:

$$\phi[v] = \lambda x' k. k (F (v (G x'))), \quad \gamma[v] = \lambda y'. v (F y') G,$$

so that  $F v \rightarrow_e \phi[v]$  and  $G v \rightarrow_e \gamma[v]$ , for any value  $v$ .

**Lemma 6.1.**  $G(F(x)) \approx_{e\eta} x$ .

*Proof.* The relation

$$R_1 = \{(G(Fx), x), (\gamma[\phi[x]], x) \mid x \text{ is a variable}\}$$

is an enf bisimulation up to  $\eta$ , because

$$G(Fx) \rightarrow_e \gamma[\phi[x]] = \lambda y. \phi[x](Fy)G,$$

and  $\phi[x](Fy)G \rightarrow_e E[xv]$ , where  $E = G(F[])$  and  $v = \gamma[\phi[y]]$ , so  $G(Fx)$  and  $\gamma[\phi[x]]$  are both related to  $x$  by (enf.6).

The proof is similar to the proof of syntactic minimal invariance in example 5.2.  $\square$

**Theorem 6.2** (CPS Retraction Theorem). *If  $t$  is closed,*

$$T(t)G \approx_{\text{enf}} t.$$

More generally, if  $n \geq 0$  and  $\text{FV}(t) \subseteq \{x_1, \dots, x_n\}$ ,

$$(\lambda x_1 \dots x_n. T(t))(Fx_1) \dots (Fx_n)G \approx_{\text{enf}} t.$$

*Proof.* If  $X = \{x_1, \dots, x_n\}$ , let

$$\sigma_X[t] = t[\phi[x_1]/x_1] \dots [\phi[x_n]/x_n].$$

Then  $(\lambda x_1 \dots x_n. T(t))(Fx_1) \dots (Fx_n) \rightarrow_e \sigma_X[T(t)]$ . So to prove the theorem it suffices to show  $\sigma_X[T(t)]G \approx_{\text{enf}} t$ . We prove this by establishing that the relation

$$R = R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6$$

is an enf bisimulation up to  $\eta$ , where  $R_1$  is the relation from the proof of lemma 6.1 and

$$\begin{aligned} R_2 &= \{(\sigma_X[T(t)]G, t) \mid \text{FV}(t) \subseteq X\}, \\ R_3 &= \{(\sigma_X[t'], t) \mid t : G : t', \text{FV}(t) \subseteq X\}, \\ R_4 &= \{(\sigma_X^y[T(t)]G, t) \mid \text{FV}(t) \subseteq X \uplus \{y\}\}, \\ R_5 &= \{(\sigma_X^z[t'], t) \mid t : G : t', \text{FV}(t) \subseteq X \uplus \{z\}\}, \\ R_6 &= \{(\lambda y. \sigma_X^y[T(t)]G, \lambda y. t) \mid \text{FV}(t) \subseteq X \uplus \{y\}\}. \end{aligned}$$

and  $\sigma_X^y[t] = (\lambda y. \sigma_X[T(t)])(Fy)$ , and  $X$  ranges over arbitrary finite sets of variables.

We need to check that all pairs of terms in  $R$  satisfy one of (enf.1)–(enf.6). From the proof of lemma 6.1 we know  $R_1$  is OK. Furthermore,  $R_6$  satisfies (enf.4) because  $R_4 \subseteq R$ . Finally, by lemma 2.1 and 4.4,  $\sigma_X[T(t)]G \rightarrow_e \sigma_X[t']$ , for some  $t'$  such that  $t : G : t'$ , and  $\sigma_X^y[t'] \rightarrow_e \sigma_{X \uplus \{y\}}[t']$ , so checking  $R_2$ – $R_5$  boils down to checking  $R_3$ .

So consider terms  $(\sigma_X[t'], t) \in R_3$  such that  $t : G : t'$  and  $\text{FV}(t) \subseteq X$ . Proceed by analysis of the eager reduction behaviour of  $t$ . There are three cases.

Case 1:  $t$  has no enf,  $t \rightarrow_e^\omega$ . Then, by lemma 4.5 and 2.1,  $\sigma_X[t'] \rightarrow_e^\omega$ .

Case 2:  $t$  eager reduces to a value,  $t \rightarrow_e v$ . Then, by lemma 4.5 and 2.1,  $\sigma_X[t'] \rightarrow_e G\sigma_X[\Psi(v)]$ . Furthermore,  $G\sigma_X[\Psi(v)] \rightarrow_e \gamma[\sigma_X[\Psi(v)]]$ .

- If  $v$  is a variable,  $v = x$ , then  $x \in X$  and

$$\gamma[\sigma_X[\Psi(v)]] = \gamma[\phi[x]].$$

As in the proof of lemma 6.1, we deduce that  $\sigma_X[t']$  is related to  $t$  by (enf.6).

- If  $v$  is a  $\lambda$ -abstraction,  $v = \lambda y. t_0$ , then

$$\begin{aligned} \gamma[\sigma_X[\Psi(v)]] &= \gamma[\lambda y. \sigma_X[T(t_0)]] \\ &= \lambda y. (\lambda y. \sigma_X[T(t_0)])(Fy)G \end{aligned}$$

and, since  $(t_0, (\lambda y. \sigma_X[T(t_0)])(Fy)G) \in R'$ , this shows that  $\sigma_X[t']$  is related to  $t$  by (enf.4).

Case 3:  $t$  eager reduces to a non-value enf,  $t \rightarrow_e E[xv]$ . Then  $x \in X$  and, by lemma 4.5,

$$t' \rightarrow_e x \Psi(v) \lambda z. t'',$$

where  $z \notin \text{FV}(E)$  and  $E[z] : G : t''$ . Therefore

$$\sigma_X[t'] \rightarrow_e \phi[x] \sigma_X[\Psi(v)] \lambda z. \sigma_X[t''] \rightarrow_e E'[xv'],$$

where  $E' = (\lambda z. \sigma_X[t''])(F[])$  and  $v' = \gamma[\sigma_X[\Psi(v)]]$ . By the same argument as in case 2, we see that

$$\begin{aligned} (v', v) &= (\gamma[\phi[v]], v) \in R_1, \text{ if } v = x, \\ (v', v) &= (\lambda y. (\lambda y. \sigma_X[T(t_0)])(Fy)G, \lambda y. t_0) \in R_6, \\ &\quad \text{if } v = \lambda y. t_0. \end{aligned}$$

And  $(E'[z], E[z]) = (\sigma_X^z[t''], E[z]) \in R_5$ .  $\square$

## 7. Future work

The work presented in this paper is syntactic. One question to ask is whether the analogies between enf bisimilarity (up to  $\eta$ ) and Lévy-Longo tree equivalence extend to model constructions. Are there models for enf bisimilarity or enf bisimilarity up to  $\eta$  that are call-by-value analogues of the lazy PSE models [19] or game models [20] for Lévy-Longo tree equivalence?

Normal form bisimulation can be extended to  $\lambda$ -calculi with control operators and non-deterministic choice operators and these extensions are conservative with regard to normal form bisimilarity. This makes a virtue, as it were, of the fact that most normal form bisimilarities are not operationally extensional because, in general, these extensions are not conservative with regard to operationally extensional equivalences. Lévy-Longo tree equivalence coincides with contextual equivalence for some extended  $\lambda$ -calculi, see [5]. Is there an extension of the call-by-value  $\lambda$ -calculus for which enf bisimilarity or enf bisimilarity up to  $\eta$  coincides with contextual equivalence?

We derived the congruence proof for enf bisimilarity (up to  $\eta$ ) from the congruence of Böhm tree equivalence (up

to  $\eta$ ) via the CPS transform. It is also possible to prove congruence of enf bisimilarity and enf bisimilarity up to  $\eta$  directly like the congruence proofs for other normal form bisimilarities (tree equivalences) in [13], although the congruence proofs for  $\simeq_e$  and  $\simeq_{e\eta}$  require non-trivial changes to the relational substitutive context closure operation in *op.cit.* Such direct congruence proofs are needed to extend normal form bisimulation to  $\lambda$ -calculi that are not amenable to CPS transform (for instance, some non-deterministic  $\lambda$ -calculi). Moreover, from the direct congruence proofs, we can derive bisimulation “up to context” proof principles like those for other normal form bisimilarities in *op.cit.*. For example, bisimulation up to context would simplify the proof of  $Y_v \simeq_e \Theta_v$  in example 3.1 by trimming the relation  $R$  to  $\{(Y_v, \Theta_v), (\Delta \Delta, \Theta_v x)\}$ . This is another example of how normal form bisimulation proof principles can be more powerful than applicative bisimulation. The validity of applicative bisimulation up to context is an open problem [11].

**Acknowledgments** I thank the referees for helpful suggestions to improve the presentation.

## References

- [1] S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.
- [2] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Number 103 in Studies in Logic and the Foundations of Mathematics. North-Holland, revised edition, 1984.
- [3] G. Boudol. On the semantics of the call-by-name CPS transform. *Theoretical Computer Science*, 234:309–321, 2000.
- [4] O. Danvy. Back to direct style. *Science of Computer Programming*, 22(3):183–195, 1994.
- [5] M. Dezani-Ciancaglini, J. Tiuryn, and P. Urzyczyn. Discrimination by parallel observers. In *12th LICS*, pages 396–407. IEEE, 1997.
- [6] L. Egidi, F. Honsell, and S. Ronchi della Rocca. Operational, denotational and logical descriptions: a case study. *Fundamenta Informaticae*, 16(2):149–169, 1992.
- [7] A. Filinski. Representing monads. In *21st POPL*, pages 446–457. ACM, 1994.
- [8] J. Hatcliff and O. Danvy. Thunks and the  $\lambda$ -calculus. *Journal of Functional Programming*, 7(3):303–319, 1997.
- [9] J. M. E. Hyland. A syntactic characterisation of the equality in some models for the lambda calculus. *Journal of the London Mathematical Society*, 12(3):361–370, 1976.
- [10] J. Kucan. Retraction approach to CPS transform. *Higher-Order and Symbolic Computation*, 11(2):145–175, 1998.
- [11] S. B. Lassen. Relational reasoning about contexts. In A. D. Gordon and A. M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 91–135. Cambridge University Press, 1998.
- [12] S. B. Lassen. *Relational Reasoning about Functions and Nondeterminism*. PhD thesis, Dept. of Computer Science, Univ. of Aarhus, 1998. BRICS Dissertation Series DS-98-2.
- [13] S. B. Lassen. Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context. In *MFPS XV*, volume 20 of *ENTCS*, 1999.
- [14] G. Longo. Set-theoretical models of lambda calculus: Theories, expansions and isomorphisms. *Annals of Pure and Applied Logic*, 24:153–188, 1983.
- [15] I. A. Mason and C. L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):297–327, 1991.
- [16] A. R. Meyer and M. Wand. Continuation semantics in typed lambda-calculi. In *3rd Workshop on Logics of Programs*, volume 193 of *LNCS*, pages 219–224, 1985.
- [17] E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-86, Univ. of Edinburgh, 1988.
- [18] C.-H. L. Ong. *The Lazy Lambda Calculus: An Investigation into the Foundations of Functional Programming*. PhD thesis, Univ. of London, 1988.
- [19] C.-H. L. Ong. Lazy lambda calculus: Theories, models and local structure characterization. In *19th ICALP*, volume 623 of *LNCS*, pages 487–498, 1992.
- [20] C.-H. L. Ong and P. D. Gianantonio. Games characterizing Lévy-Longo trees. *Theoretical Computer Science*, 312(1):121–142, 2004.
- [21] R. P. Perez. An extensional partial combinatory algebra based on  $\lambda$ -terms. In *MFCS*, volume 520 of *LNCS*, pages 387–396, 1991.
- [22] A. M. Pitts. Some notes on inductive and co-inductive techniques in the semantics of functional programs (draft version). BRICS Notes Series NS-94-5, BRICS, Dept. of Computer Science, Univ. of Aarhus, 1994.
- [23] G. D. Plotkin. Call-by-name, call-by-value and the  $\lambda$ -calculus. *Theoretical Computer Science*, 1:125–159, 1975.
- [24] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3/4):289–360, 1993.
- [25] A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems*, 19(6):916–941, 1997.
- [26] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1):120–153, 1994.
- [27] C. P. Wadsworth. The relation between computational and denotational properties for Scott’s  $D_\infty$ -models of the lambda-calculus. *SIAM Journal on Computing*, 5(3):488–521, 1976.

## A. CPS translation theorem for equational theories

This appendix shows how our modification of Plotkin’s call-by-value CPS transform “fixes” Plotkin’s call-by-value CPS translation theorem.

Let  $P$  denote Plotkin’s original call-by-value CPS transform. Plotkin [23] showed that

$$t =_{\beta_v} t' \text{ implies } P(t) =_{\beta_v} P(t'), \quad (\text{A.1})$$

$$P(t) =_{\beta_v} P(t') \text{ iff } P(t) =_{\beta} P(t'). \quad (\text{A.2})$$

We are going to show that we can strengthen the implication in (A.1) to a bi-implication, if we replace  $P$  by  $T$  and  $\beta_v$  by  $\beta_c$ , and that (A.2) also holds for  $T$  in place of  $P$ .

Moreover, we show that

$$t =_{\beta_c \eta_v} t' \text{ iff } T(t) =_{\beta_\eta} T(t'). \quad (\text{A.3})$$

(And, since  $P$  and  $T$  are identical up to  $\eta$  conversion, (A.3) also holds for  $P$  in place of  $T$ .)

The structure of our proofs is adapted from the work by Sabry and Felleisen [24] who showed (A.3) for Fischer's call-by-value CPS transform. We

- (1) show that  $T$  maps  $\beta_c$  and  $\beta_c \eta_v$  equivalences into  $\beta_v$  and  $\beta_\eta$  equivalences, respectively,
- (2) describe a target CPS calculus of the CPS transform that is closed under  $\beta$  and  $\eta$  reductions,
- (3) define an inverse CPS transform  $T^{-1}$  such that  $T^{-1} \circ T$  is the identity transform up to  $=_{\beta_v}$ ,
- (4) show that  $T^{-1}$  maps  $\beta$  and  $\eta$  reductions in the target CPS calculus into  $\beta_c$  and  $\beta_c \eta_v$  equivalences, respectively, in the source calculus, and
- (5) use the Church-Rosser property of  $\beta$  and  $\beta_\eta$  reduction to conclude that  $\beta$  and  $\beta_\eta$  equivalence on target terms of  $T$  induce  $\beta_c$  and  $\beta_c \eta_v$  equivalence, respectively, on source terms.

The proofs are simpler than Sabry and Felleisen's because we eschew any auxiliary results about correspondences between reductions in the source and target calculi and we do not establish a full equational correspondence between  $T$  and  $T^{-1}$ .

### A.1. CPS transformation of equations

Recall from §4 the call-by-value CPS transform  $T$ , that maps terms into  $\lambda$ -abstractions, and the auxiliary transform  $\Psi$ , that maps values into values.

**Lemma A.1.**  $\text{FV}(T(t)) = \text{FV}(t)$  and  $\text{FV}(\Psi(v)) = \text{FV}(v)$ .

*Proof.* By structural induction on  $t$ . The second equation follows from the first.  $\square$

**Lemma A.2.**  $\lambda k. T(t) \lambda x. k x =_{\beta_v} T(t)$ .

*Proof.* By inspection of the definition of  $T(t)$ , we see that  $T(t)$  is a  $\lambda$ -abstraction,  $T(t) = \lambda k. t'$  and that every occurrence of the parameter  $k$  in the body  $t'$  is applied to a value. Therefore  $\lambda k. T(t) \lambda x. k x =_{\beta_v} \lambda k. (T(t')[\lambda x. k x/k]) =_{\beta_v} \lambda k. (T(t')[k/k]) = T(t)$ .  $\square$

**Lemma A.3.**  $t =_{\beta_c} t'$  implies  $T(t) =_{\beta_v} T(t')$ .

*Proof.* Since  $T$  is compositional, it suffices to prove the lemma for the five  $\beta_c$  equations. For illustration, we prove that (*id*) satisfies the implication:

$$\begin{aligned} T((\lambda x. x) t) &= \lambda k. ((\lambda k. k \lambda x. \lambda k. k x) \\ &\quad \lambda x_1. T(t) \lambda x_2. x_1 x_2 \lambda x. k x) \\ &=_{\beta_v} \lambda k. T(t) \lambda x_2. k x_2 \\ (\text{lemma A.2}) &=_{\beta_v} T(t). \end{aligned} \quad \square$$

**Lemma A.4.**  $t =_{\beta_c \eta_v} t'$  implies  $T(t) =_{\beta_\eta} T(t')$ .

The proof is similar to that of lemma A.3.

### A.2. Call-by-value CPS calculus

We now describe a subset of the  $\lambda$ -calculus that contains the target terms of the call-by-value CPS transform and, furthermore, is closed under arbitrary  $\beta$  and  $\eta$  reductions.

We partition the variables into two kinds, value variables  $x$  and continuation variables  $k$ . The syntax is specified by the grammar:

Thunks	$q ::= \lambda k. a \mid p_1 p_2$
CPS Values	$p ::= x \mid \lambda x. q$
Answers	$a ::= c p \mid q c$
Continuations	$c ::= k \mid \lambda x. a$

The syntax is further constrained by the side condition that CPS values and thunks have no free occurrences of continuation variables. Given this side condition, every answer has exactly one free continuation variable, namely from the first production in the grammar for continuations. The bound variable  $k$  in a thunk  $\lambda k. a$  is the free continuation variable in  $a$ .

An inspection of the definitions of  $\Psi$  and  $T$  shows that  $\Psi$  maps values into CPS values and  $T$  maps terms into thunks.

*Remark 4.* We name the first syntactic category ‘thunks’ (somewhat inaccurately), because  $\lambda k. a$  delays evaluation of answer  $a$ , awaiting a continuation (or ‘context’)  $k$ .

CPS values and continuations are syntactic values (variables or  $\lambda$ -abstractions), so every function argument in the CPS calculus is a value. Therefore every  $\beta$  redex is a  $\beta_v$  redex. (But not every  $\eta$  redex is an  $\eta_v$  redex—for instance, the thunk  $\lambda k. x y k$  is an  $\eta$  redex but not an  $\eta_v$  redex.)

The following key property of the CPS syntax can be verified by a straightforward examination of all possible forms of  $\beta$  and  $\eta$  redexes expressible in the CPS syntax.

**Lemma A.5.** *The CPS syntax, including the side conditions on free occurrences of continuation variables, is closed under  $\beta$  and  $\eta$  reductions.*

### A.3. Inverse CPS transform

We will now define an inverse CPS transform  $T^{-1}$  that maps CPS thunks back into terms (cf. [4]). We define  $T^{-1}$  together with three auxiliary functions  $\Psi^{-1}$ ,  $A^{-1}$ , and  $K^{-1}$  that map the other syntactic categories of CPS terms into terms.

$$\begin{aligned} T^{-1}(\lambda k. a) &= A^{-1}(a), \quad T^{-1}(p_1 p_2) = \Psi^{-1}(p_1) \Psi^{-1}(p_2), \\ \Psi^{-1}(x) &= x, \quad \Psi^{-1}(\lambda x. q) = \lambda x. T^{-1}(q), \\ A^{-1}(c p) &= K^{-1}(c) \Psi^{-1}(p), \quad A^{-1}(q c) = K^{-1}(c) T^{-1}(q), \\ K^{-1}(k) &= \lambda x. x, \quad K^{-1}(\lambda x. a) = \lambda x. A^{-1}(a). \end{aligned}$$

where  $\Psi^{-1}$  maps CPS values into values,  $A^{-1}$  maps answers into terms, and  $K^{-1}$  maps continuations into  $\lambda$ -abstractions.

**Lemma A.6.**  $T^{-1}(T(t)) =_{\beta_c} t$  and  $\Psi^{-1}(\Psi(v)) =_{\beta_c} v$ .

*Proof.* The second equation follows easily from the first. Prove the first equation by structural induction on  $t$ .  $\square$

### A.4. CPS reductions

We now show that  $T^{-1}$  maps  $\beta$  and  $\eta$  reductions in the target CPS calculus into  $\beta_c$  and  $\beta_c \eta_v$  equivalences, respectively, in the source calculus.

The proofs use the following two substitution lemmas.

**Lemma A.7.**  $T^{-1}(q[p'/x]) = T^{-1}(q)[\Psi^{-1}(p')/x]$ ,  
 $\Psi^{-1}(p[p'/x]) = \Psi^{-1}(p)[\Psi^{-1}(p')/x]$ ,  
 $A^{-1}(a[p'/x]) = A^{-1}(a)[\Psi^{-1}(p')/x]$ ,  
 $K^{-1}(c[p'/x]) = K^{-1}(c)[\Psi^{-1}(p')/x]$ .

*Proof.* By simultaneous structural induction on  $q$ ,  $p$ ,  $a$ , and  $c$ .  $\square$

**Lemma A.8.**  $A^{-1}(a[c/k]) =_{\beta_c} K^{-1}(c) A^{-1}(a)$ .

*Proof.* By structural induction on  $a$ . The proof utilizes that  $k$  does not occur free in any CPS value  $p$  or thunk  $q$ .  $\square$

**Lemma A.9.**  $q \rightarrow_{\beta} q'$  implies  $T^{-1}(q) =_{\beta_c} T^{-1}(q')$ .

*Proof.* By examination of every form of  $\beta$  redex in the CPS syntax. There are three cases.

- (1)  $(\lambda x. q) p \rightarrow_{\beta} q[p/x]$ . Use lemma A.7 to show  
 $T^{-1}((\lambda x. q) p) = (\lambda x. T^{-1}(q)) \Psi^{-1}(p) =_{\beta_c}$   
 $T^{-1}(q)[\Psi^{-1}(p)/x] = T^{-1}(q[p/x]).$
- (2)  $(\lambda x. a) p \rightarrow_{\beta} a[p/x]$ . Similar to case 1.
- (3)  $(\lambda k. a) c \rightarrow_{\beta} a[c/k]$ . Use lemma A.8 to show  
 $A^{-1}((\lambda k. a) c) = K^{-1}(c) T^{-1}(\lambda k. a) =$   
 $K^{-1}(c) A^{-1}(a) =_{\beta_c} A^{-1}(a[c/k]).$   $\square$

**Lemma A.10.**  $q \rightarrow_{\eta} q'$  implies  $T^{-1}(q) =_{\beta_c \eta_v} T^{-1}(q')$ .

*Proof.* By examination of every form of  $\eta$  redex in the CPS syntax. There are three cases.

- (1)  $\lambda k. q k \rightarrow_{\eta} q$ . (The side condition  $k \notin \text{FV}(q)$  is superfluous because  $k$  cannot appear free in the thunk  $q$  according to the CPS syntax.)  
 $T^{-1}(\lambda k. q k) = (\lambda x. x) T^{-1}(q) =_{\beta_c} T^{-1}(q).$
- (2)  $\lambda x. p x \rightarrow_{\eta} p$ , if  $x \notin \text{FV}(p)$ . Since  $\Psi^{-1}(p)$  is a value,  
 $\Psi^{-1}(\lambda x. p x) = \lambda x. \Psi^{-1}(p) x =_{\eta_v} \Psi^{-1}(p).$
- (3)  $\lambda x. c x \rightarrow_{\eta} c$ , if  $x \notin \text{FV}(c)$ . Since  $K^{-1}(c)$  is a  $\lambda$ -abstraction,  
 $K^{-1}(\lambda x. c x) = \lambda x. K^{-1}(c) x =_{\beta_v} K^{-1}(c).$   $\square$

### A.5. CPS translation theorem

**Theorem A.11.**  $t =_{\beta_c} t'$  iff  $T(t) =_{\beta_v} T(t')$   
iff  $T(t) =_{\beta_c} T(t')$   
iff  $T(t) =_{\beta} T(t')$ .

*Proof.* Lemma A.3 says that  $t =_{\beta_c} t'$  implies  $T(t) =_{\beta_v} T(t')$  which, in turn, implies  $T(t) =_{\beta_c} T(t')$  and  $T(t) =_{\beta} T(t')$  because  $=_{\beta_v} \subseteq =_{\beta_c} \subseteq =_{\beta}$ . Finally, because of the Church-Rosser property of  $\beta$  reduction,  $T(t) =_{\beta} T(t')$  implies there exists a term  $q$  such that  $T(t) \rightarrow_{\beta} q$  and  $T(t') \rightarrow_{\beta} q$ , so, by repeated applications of lemma A.9,

$$T^{-1}(T(t)) =_{\beta_c} \dots =_{\beta_c} T^{-1}(q) =_{\beta_c} \dots =_{\beta_c} T^{-1}(T(t'))$$

and then  $t =_{\beta_c} t'$  follows from lemma A.6.  $\square$

**Theorem A.12.**  $t =_{\beta_c \eta_v} t'$  iff  $T(t) =_{\beta \eta} T(t')$ .

*Proof.* The ‘only if’ direction comes from lemma A.4. The reverse implication follows from the Church-Rosser property of  $\beta \eta$  reduction, lemma A.6, lemma A.9, and lemma A.10, analogously to the proof of theorem A.11.  $\square$

*Remark 5.* Plotkin used an extended  $\lambda$ -calculus with interpreted constants in [23]. If we extend the CPS calculus appropriately, we expect theorem A.11 and its proof can be extended to Plotkin’s calculus. Theorem A.12 cannot be because the  $\eta_v$  and  $\eta$  equations are unsound for the extended calculus.