Normal Form Bisimulation For Typed Calculi

Soren B. Lassen
Google, Inc.
soren@google.com

Paul Blain Levy
University of Birmingham, U.K.
pbl@cs.bham.ac.uk

Abstract

Normal form bisimulation is a powerful theory of program equivalence, originally developed to characterize Lévy-Longo tree equivalence and Böhm tree equivalence. It has been adapted to a range of untyped, higher-order calculi, but types have presented a difficulty. In this paper, we present an account of normal form bisimulation for types, including recursive types. We develop our theory both for a continuation-passing style calculus, jump-with-argument (JWA), where normal form bisimilarity takes a very simple form, and for a general direct-style calculus, call-by-push-value (CBPV), that subsumes both call-by-name and call-by-value. A stack-passing transform from CBPV to JWA connects the two. We give a novel congruence proof, based on insights from game semantics. A notable feature is the seamless treatment of eta-expansion.

1 Introduction

1.1 Normal Form Bisimulation

Suppose we have two functions $V, V' : A \to B$. When should they be deemed equivalent? Here are two answers:

- when VW and $V'W$ behave the same for every *closed* $W : A$
- when $V\mathtt{y}$ and $V'\mathtt{y}$ behave the same for *fresh* $\mathtt{y} : A$.

The first answer leads to the theory of *applicative bisimulation* [1], the second to that of *normal form bisimulation*. The first answer requires us to run closed terms only, the second to run non-closed terms.

To illustrate the difference, let $G(p, q)$ be the following

$$\lambda\mathtt{x}.\mathtt{if}\ (\mathtt{x}p) \begin{cases} \mathtt{then}\ (\mathtt{if}\ (\mathtt{x}q)\ \mathtt{then}\ \mathtt{true}\ \mathtt{else}\ \mathtt{false}) \\ \mathtt{else}\ (\mathtt{if}\ (\mathtt{x}q)\ \mathtt{then}\ \mathtt{false}\ \mathtt{else}\ \mathtt{true}) \end{cases}$$

and let V be $G(\mathtt{true}, \mathtt{false})$ and V' be $G(\mathtt{false}, \mathtt{true})$, both of type $(\mathtt{bool} \to \mathtt{bool}) \to \mathtt{bool}$. Assuming the language is free of effects besides divergence, they cannot be distinguished by applying to closed arguments. But let us apply them to a fresh identifier $\mathtt{y}$. Then the first stops when it tries to apply $\mathtt{y}$ to $\mathtt{true}$, the second when it tries to apply $\mathtt{y}$ to $\mathtt{false}$, in each case with a frame still on the stack.

Normal form bisimilarity requires that two (nondivergent) programs end up *either* applying the same identifier to equivalent arguments with equivalent continuations, *or* returning equivalent values. In our example, the arguments are different ($\mathtt{true}$ and $\mathtt{false}$ respectively), so V and V' are not normal form bisimilar.

The game semantics in [2, 7] contains a similar idea; readers familiar with it will see that V and V' denote different strategies. But normal form bisimulation describes *operationally* the distinction between V and V'; it is not defined compositionally and must be proved *a posteriori* to be a congruence.

1.2 Contributions

Normal form bisimulation has been developed for a variety of calculi e.g. [8, 13, 14], but they are all untyped. Adapting it to typed calculi has presented difficulties. For example, in call-by-value, if A is an empty type, such as $\mu\mathtt{X}.(\mathtt{bool} \times \mathtt{X})$, then all functions of type $A \to B$ should be equivalent—without appealing to the absence of any closed arguments, as we would for applicative bisimulation.

Another problem is the treatment of the η-law. Existing congruence proofs generally prove substitutivity for η-expanded terms, and justify η-expansion separately. This is reminiscent of operational analyses of game semantics [5, 9] that are (explicitly) valid for η-long terms only.

The three key contributions of this paper are:

1. the extension of normal form bisimulation to types, including sum types, recursive types and empty types
2. a new, lucid congruence proof that highlights connections with game semantics
3. a seamless treatment of η-expansion—we do not need to mention it in our definitions or proofs.

Because stacks play such a central role in the theory, we develop our theory first in a continuation-passing style calculus Jump-With-Argument (JWA), before transferring the ideas to direct-style. For the latter we use call-by-push-value (CBPV), a general calculus that subsumes call-by-name and call-by-value. The stack-passing style (StkPS) transform from CBPV to JWA (an extension of CPS) is used to obtain a notion of bisimulation for CBPV from that already developed for JWA, and an immediate congruence proof. As an example application, we prove that at each computation type of CBPV, there is a unique fixpoint combinator with respect to normal form bisimilarity.

We expect these contributions to play significant roles in applying normal form bisimulation to reason about programming languages as well as in furthering our understanding of the semantical structure of normal form bisimulation, especially connections with game semantics. Our normal form bisimulation theory for CBPV is general and straightforwardly induces normal form bisimulation theories for, e.g., PCF and FPC via their embeddings into CBPV. Moreover, we believe our treatment of types can be readily combined with existing untyped normal form bisimulation theories for control, and state [8, 14].

1.3 Related Work

The main related work is the game semantics of [7, 2], which is adapted to JWA and CBPV in [10]. We state without proof (though it is rather obvious from Sect. 3.1) that terms are normal form bisimilar iff they denote the same strategy. Like this paper, [8] mentions game semantics as motivation for a congruence proof using a transition sytem.

Abramsky's applicative bisimulation [1] introduced the general idea of bisimulation equivalences between functional programs; this was adapted to a CPS calculus in [12]. Normal form bisimulation generalizes Sangiorgi's open applicative bisimulation [13] and is closely connected to Böhm tree equivalence [3]. In the typed setting, Böhm trees are studied compositionally in [4], closely linked to game semantics.

2 Jump-With-Argument

2.1 Syntax and Semantics

Jump-With-Argument is a continuation-passing style calculus, extending the CPS calculus in [16]. Its types are given (including recursive types) by

$$A ::= \neg A \mid \textstyle\sum_{i\in I} A_i \mid 1 \mid A \times A \mid \mathtt{X} \mid \mu\mathtt{X}.A$$

where I is any finite set. The type $\neg A$ is the type of functions that take an argument of type A and do not return.

$$\frac{}{\Gamma \vdash^{\mathsf{v}} \mathtt{x} : A}\ (\mathtt{x} : A) \in \Gamma \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{n}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M}$$

$$\frac{\hat{\imath} \in I \quad \Gamma \vdash^{\mathsf{v}} V : A_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{v}} \langle \hat{\imath}, V\rangle : \sum_{i\in I} A_i} \qquad \frac{\begin{array}{c}\Gamma \vdash^{\mathsf{v}} V : \sum_{i\in I} A_i \\ \Gamma, \mathtt{x}_i : A_i \vdash^{\mathsf{n}} M_i \ \ (\forall i \in I)\end{array}}{\Gamma \vdash^{\mathsf{n}} \mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x}_i\rangle.M_i\}_{i\in I}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} \langle V, V'\rangle : A \times A'} \qquad \frac{\begin{array}{c}\Gamma \vdash^{\mathsf{v}} V : A \times A' \\ \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{n}} M\end{array}}{\Gamma \vdash^{\mathsf{n}} \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.M}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{n}} M}{\Gamma \vdash^{\mathsf{v}} \lambda\mathtt{x}.M : \neg A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : \neg A \quad \Gamma \vdash^{\mathsf{v}} W : A}{\Gamma \vdash^{\mathsf{n}} VW}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A[\mu\mathtt{X}.A/\mathtt{X}]}{\Gamma \vdash^{\mathsf{v}} \mathtt{fold}\ V : \mu\mathtt{X}.A} \qquad \frac{\begin{array}{c}\Gamma \vdash^{\mathsf{v}} V : \mu\mathtt{X}.A \\ \Gamma, \mathtt{x} : A[\mu\mathtt{X}.A/\mathtt{X}] \vdash^{\mathsf{n}} M\end{array}}{\Gamma \vdash^{\mathsf{n}} \mathtt{pm}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M}$$

Figure 1. Syntax of JWA, with type recursion

JWA has two judgements: *values* written $\Gamma \vdash^{\mathsf{v}} V$ and *non-returning commands* written $\Gamma \vdash^{\mathsf{n}} M$. The syntax[1] is shown in Fig. 1. We write `pm` as an abbreviation for "pattern-match", and write `let` to make a binding. We omit typing rules, etc., for 1, since 1 is analogous to $\times$.

From the cpo viewpoint, a JWA type denotes an (unpointed) cpo. In particular, $\neg A$ denotes $[\![A]\!] \rightarrow R$, where R is a chosen pointed cpo.

Operational semantics To evaluate a command $\Gamma \vdash^{\mathsf{n}} M$, simply apply the transitions (β-reductions) in Fig. 2 until a terminal command is reached. Every command M is either a redex or terminal; by determinism, either $M \leadsto^* T$ for unique terminal T, or else $M \leadsto^\infty$. This operational semantics is called the *C-machine*.

2.2 Ultimate Pattern Matching

To describe normal form bisimulation, we need to decompose a value into an *ultimate pattern* (the tags) and a value sequence (the rest). Take, for example, the value

$$\langle i_0, \langle\langle\langle \lambda\mathtt{w}.M, \mathtt{x}\rangle, \mathtt{y}\rangle, \langle i_1, \mathtt{x}\rangle\rangle\rangle$$

Provided x and y have $\neg$ type, we can decompose this as the ultimate pattern

$$\langle i_0, \langle\langle\langle -_{\neg A}, -_{\neg B}\rangle, -_{\neg C}\rangle, \langle i_1, -_{\neg B}\rangle\rangle\rangle$$

[1]In earlier works e.g. [9], $\gamma\mathtt{x}.M$ was written for $\lambda\mathtt{x}.M$ and $V \nearrow W$ for VW.

Transitions

$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.M$	$\rightsquigarrow$	$M[V/\mathtt{x}]$
$\mathtt{pm}\ \langle \hat{\imath}, V\rangle\ \mathtt{as}\ \{\langle i,\mathtt{x}\rangle.M_i\}_{i\in I}$	$\rightsquigarrow$	$M_{\hat{\imath}}[V/\mathtt{x}]$
$\mathtt{pm}\ \langle V, V'\rangle\ \mathtt{as}\ \langle \mathtt{x},\mathtt{y}\rangle.M$	$\rightsquigarrow$	$M[V/\mathtt{x}, V'/\mathtt{y}]$
$(\lambda\mathtt{x}.M)V$	$\rightsquigarrow$	$M[V/\mathtt{x}]$
$\mathtt{pm}\ \mathtt{fold}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M$	$\rightsquigarrow$	$M[V/\mathtt{x}]$

Terminal Commands

$\mathtt{z}V$
$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\langle i,\mathtt{x}\rangle.\ M_i\}_{i\in I}$
$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{x},\mathtt{y}\rangle.\ M$
$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M$

Figure 2. C-machine

(for appropriate types A, B, C), where the holes are filled with the value sequence

$$\lambda\mathtt{w}.M, \mathtt{x}, \mathtt{y}, \mathtt{x}$$

As this example shows, an ultimate pattern is built up out of tags and holes; the holes are to be filled by values of $\neg$ type. For each type A, we define the set $\mathsf{ult}^{\mathsf{v}}(A)$ of ultimate patterns of type A, by mutual induction:

- $-_{\neg A} \in \mathsf{ult}^{\mathsf{v}}(\neg A)$
- if $p \in \mathsf{ult}^{\mathsf{v}}(A)$ and $p' \in \mathsf{ult}^{\mathsf{v}}(A')$ then $\langle p, p'\rangle \in \mathsf{ult}^{\mathsf{v}}(A \times A')$
- if $\hat{\imath} \in I$ and $p \in \mathsf{ult}^{\mathsf{v}}(A_{\hat{\imath}})$ then $\langle \hat{\imath}, p\rangle \in \mathsf{ult}^{\mathsf{v}}(\sum_{i\in I} A_i)$
- if $p \in \mathsf{ult}^{\mathsf{v}}(A[\mu\mathtt{X}.A/\mathtt{X}])$ then $\mathtt{fold}\ p \in \mathsf{ult}^{\mathsf{v}}(\mu\mathtt{X}.A)$.

For $p \in \mathsf{ult}^{\mathsf{v}}(A)$, we write $H(p)$ for the list of types (all $\neg$ types) of holes of p. Given a value sequence $\Gamma \vdash^{\mathsf{v}} \overrightarrow{V} : H(p)$, we obtain a value $\Gamma \vdash^{\mathsf{v}} p(\overrightarrow{V}) : A$ by filling the holes of p with $\overrightarrow{V}$. We can now state our decomposition theorem.

Proposition 1 *Let $\overrightarrow{\mathtt{x} : \neg A} \vdash V : B$ be a value. Then there is a unique ultimate pattern $p \in \mathsf{ult}^{\mathsf{v}}(B)$ and value sequence $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{W} : H(p)$ such that $V = p(\overrightarrow{W})$.*

Proof Induction on V. □

3 Normal Form Bisimulation

In this section, we define normal form bisimulation. Some readers may like to see this as a way of characterizing when two terms have the same Böhm tree[2], or (equivalently) denote the same innocent strategy, but neither of these concepts will be used in the paper.

Notation For any $n \in \mathbb{N}$, we write $\$n$ for the set $\{0, \ldots, n-1\}$. For a sequence $\overrightarrow{a}$, we write $|\overrightarrow{a}|$ for its length.

Any terminal command $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{n}} M$ must be of the form $\mathtt{x}_i p(\overrightarrow{V})$. The core of our definition is that we regard (i, p) as an observable action, so we write

$$M \overset{ip}{\leadsto} \overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{v}} \overrightarrow{V} : H(p)$$

(This action is called a "Proponent move".) Thus, for any command $\overrightarrow{\mathtt{x} : A} \vdash N$, we have either

$$N \rightsquigarrow^{*} \overset{ip}{\leadsto} \overrightarrow{V}$$

for unique $i, p, \overrightarrow{V}$, or else $N \rightsquigarrow^{\infty}$.

Suppose we are given a value sequence $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{V : \neg B}$. For each $j \in \$|\overrightarrow{\neg B}|$ and $q \in \mathsf{ult}^{\mathsf{v}}(B_j)$, we define $(\overrightarrow{V : \neg B}) : jq$ to be the command

$$\overrightarrow{\mathtt{x} : \neg A}, \overrightarrow{\mathtt{y}} : H(q) \vdash^{\mathsf{n}} V_j q(\overrightarrow{y})$$

where $\overrightarrow{\mathtt{y}}$ are fresh. (We call this operation an "Opponent move".)

Definition 1 *Let $\mathcal{R}$ be a set of pairs of commands $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{n}} M, M'$.*

1. *Let $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{V}, \overrightarrow{V'} : \overrightarrow{\neg B}$ be two value sequences. We say $\overrightarrow{V}\, \mathcal{R}^{\mathsf{v}}\, \overrightarrow{V'}$ when for any $j \in \$|\overrightarrow{\neg B}|$ and $q \in \mathsf{ult}^{\mathsf{v}}(B_j)$, we have $(\overrightarrow{V} : jq)\, \mathcal{R}\, (\overrightarrow{V'} : jq)$.*
2. *$\mathcal{R}$ is a* normal form bisimulation *when $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{n}} N\, \mathcal{R}\, N'$ implies either*
   - *$N \rightsquigarrow^{\infty}$ and $N' \rightsquigarrow^{\infty}$, or*
   - *$N \rightsquigarrow^{*} \overset{ip}{\leadsto} \overrightarrow{V}$ and $N' \rightsquigarrow^{*} \overset{ip}{\leadsto} \overrightarrow{V'}$ and $\overrightarrow{V}\, \mathcal{R}^{\mathsf{v}}\, \overrightarrow{V'}$.*

We write $\eqsim$ for normal form bisimilarity, *i.e. the greatest normal form bisimulation.*

A variant formulation is often useful:

Proposition 2 *Let $\mathcal{R}$ be a set of pairs of commands $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{n}} M, M'$, and $\mathcal{S}$ a set of pairs of value sequences $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{V}, \overrightarrow{V'} : \overrightarrow{\neg B}$. Suppose that*

- *whenever $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{V}\ \mathcal{S}\ \overrightarrow{V'} : \overrightarrow{\neg B}$, we have $(\overrightarrow{V} : jq)\, \mathcal{R}\, (\overrightarrow{V'} : jq)$ for every $j \in \$|\overrightarrow{\neg B}|$ and $q \in \mathsf{ult}^{\mathsf{v}}(B_j)$*
- *whenever $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{n}} N\, \mathcal{R}\, N'$ we have either*

[2]The Böhm trees for JWA are given by the following classes of commands and values, defined coinductively:

$$\begin{array}{rcl} \overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{n}} M & ::= & \mathtt{diverge} \mid \mathtt{x}_i p(\overrightarrow{V}) \\ \overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} V : \neg A & ::= & \lambda\{p(\overrightarrow{\mathtt{y}}).M_p\}_{p\in\mathsf{ult}^{\mathsf{v}}(A)} \end{array}$$

These trees are not actually (infinite) JWA terms, because ultimate pattern matching is not part of the syntax of JWA.

- $N \rightsquigarrow^{\infty}$ *and* $N' \rightsquigarrow^{\infty}$*, or*
- $N \rightsquigarrow^{*} \overset{ip}{\rightsquigarrow} \overrightarrow{V}$ *and* $N' \rightsquigarrow^{*} \overset{ip}{\rightsquigarrow} \overrightarrow{V'}$ *and* $\overrightarrow{V}\, \mathcal{S}\, \overrightarrow{V'}$*.*

Then $\mathcal{R}$ *is contained in* $\eqsim$*, and* $\mathcal{S}$ *is contained in* $\eqsim^{\mathsf{v}}$*.*

Proof $\mathcal{S} \subseteq \mathcal{R}^{\mathsf{v}}$ and $\mathcal{R}$ is a normal form bisimulation. □

Proposition 3 *(preservation under renaming) For any renaming* $\overrightarrow{\mathtt{x} : \neg A} \xrightarrow{\theta} \overrightarrow{\mathtt{y} : \neg B}$*, we have* $\overrightarrow{\mathtt{y} : \neg B} \vdash M \eqsim M'$ *implies* $M\theta \eqsim M'\theta$*, and* $\overrightarrow{\mathtt{x} : \neg B} \vdash^{\mathsf{v}} \overrightarrow{V} \eqsim^{\mathsf{v}} \overrightarrow{V'} : \overrightarrow{\neg C}$ *implies* $\overrightarrow{V\theta} \eqsim^{\mathsf{v}} \overrightarrow{V'\theta}$*.*

Proof Let $\mathcal{R}$ to be the set of pairs $(M\theta, M'\theta)$ where $\overrightarrow{\mathtt{y} : \neg B} \vdash M \eqsim M'$ and $\overrightarrow{\mathtt{x} : \neg A} \xrightarrow{\theta} \overrightarrow{\mathtt{y} : \neg B}$ is a renaming. Let $\mathcal{S}$ be the set of pairs $(\overrightarrow{V\theta}, \overrightarrow{V'\theta})$ such that $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{V}, \overrightarrow{V'} \eqsim^{\mathsf{v}} \overrightarrow{\neg B}$ and $\overrightarrow{\mathtt{x} : \neg A} \xrightarrow{\theta} \overrightarrow{\mathtt{y} : \neg B}$ is a renaming. It is easy to show $(\mathcal{R}, \mathcal{S})$ satisfy the conditions of Prop. 2. □

Proposition 4 *(preservation under substitution) Suppose* $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{W} \eqsim^{\mathsf{v}} \overrightarrow{W'} : \overrightarrow{\neg B}$*. Then* $\overrightarrow{\mathtt{y} : \neg B} \vdash^{\mathsf{n}} M \eqsim M'$ *implies* $M[\overrightarrow{W/\mathtt{y}}] \eqsim M'[\overrightarrow{W'/\mathtt{y}}]$ *and* $\overrightarrow{\mathtt{x} : \neg B} \vdash^{\mathsf{v}} \overrightarrow{V} \eqsim^{\mathsf{v}} \overrightarrow{V'} : \overrightarrow{\neg C}$ *implies* $V[\overrightarrow{W/\mathtt{x}}] \eqsim^{\mathsf{v}} V'[\overrightarrow{W'/\mathtt{x}}]$*.*

This is proved in the next section.

Next we have to extend normal form bisimilarity to arbitrary commands and values (conceptually following the categorical construction in [2]).

Definition 2
- *Given commands* $\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{n}} M, M'$*, we say* $M \eqsim M'$ *when for each* $\overrightarrow{p \in \mathsf{ult}^{\mathsf{v}}(A)}$ *we have*

$$\overrightarrow{\overrightarrow{\mathtt{y}} : H(p)} \vdash^{\mathsf{n}} M[\overrightarrow{p(\overrightarrow{\mathtt{y}})/\mathtt{x}}] \eqsim M'[\overrightarrow{p(\overrightarrow{\mathtt{y}})/\mathtt{x}}]$$

- *Given values* $\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{v}} V, V' : B$*, we say* $V \eqsim V'$ *when for each* $\overrightarrow{p \in \mathsf{ult}^{\mathsf{v}}(A)}$*, decomposing* $V[\overrightarrow{p(\overrightarrow{\mathtt{y}})/\mathtt{x}}]$ *as* $q(\overrightarrow{W})$ *and* $V'[\overrightarrow{p(\overrightarrow{\mathtt{y}})/\mathtt{x}}]$ *as* $q'(\overrightarrow{W'})$*, we have* $q = q'$ *and*

$$\overrightarrow{\overrightarrow{\mathtt{y}} : H(p)} \vdash^{\mathsf{v}} \overrightarrow{W} \eqsim^{\mathsf{v}} \overrightarrow{W'} : H(q)$$

It is easy to see that normal form bisimulation for JWA is an equivalence relation and validates all the β and η laws [9].

Proposition 5 $\eqsim$ *is a substitutive congruence.*

Proof Substitutivity follows from Prop. 4. For each term constructor, we prove it preserves $\eqsim$ using subsitutivity, as in [14]. □

3.1 Alternating Substitution

To prove Prop. 4, it turns out to be easier to show preservation by a more general operation on terms, *alternating substitution*, which is applied to an *alternating table* of terms. They are defined in Fig. 3. A table provides the following information:

- the context of each term is the identifiers to the left of it
- the type of each identifier, and of each term, is given in the top row.

For example, the table

$\overrightarrow{\neg A_{\mathsf{out}}}$	$\overrightarrow{\neg B_{\mathsf{out}}}$	$\overrightarrow{\neg A_0}$	$\overrightarrow{\neg B_0}$	$\overrightarrow{\neg A_1}$	$\overrightarrow{\neg B_1}$	$\vdash^{\mathsf{n}}$
$\overrightarrow{\mathtt{x}}_{\mathsf{out}}$		$\overrightarrow{V}_0$	$\overrightarrow{\mathtt{x}}_0$	$\overrightarrow{V}_1$	$\overrightarrow{\mathtt{x}}_1$	M
	$\overrightarrow{\mathtt{z}}_{\mathsf{out}}$	$\overrightarrow{\mathtt{z}}_0$	$\overrightarrow{W}_0$	$\overrightarrow{\mathtt{z}}_1$	$\overrightarrow{W}_1$	

consists of the following value-sequences and commands:

$$\begin{array}{rcl}
\overrightarrow{\mathtt{x}}_{\mathsf{out}} : \overrightarrow{\neg A_{\mathsf{out}}} & \vdash^{\mathsf{v}} & \overrightarrow{V}_0 : \overrightarrow{\neg A_0} \\
\overrightarrow{\mathtt{z}}_{\mathsf{out}} : \overrightarrow{\neg B_{\mathsf{out}}}, \overrightarrow{\mathtt{z}}_0 : \overrightarrow{\neg A_0} & \vdash^{\mathsf{v}} & \overrightarrow{W}_0 : \overrightarrow{\neg B_0} \\
\overrightarrow{\mathtt{x}}_{\mathsf{out}} : \overrightarrow{\neg A_{\mathsf{out}}}, \overrightarrow{\mathtt{x}}_0 : \overrightarrow{\neg B_0} & \vdash^{\mathsf{v}} & \overrightarrow{V}_1 : \overrightarrow{\neg A_1} \\
\overrightarrow{\mathtt{z}}_{\mathsf{out}} : \overrightarrow{\neg B_{\mathsf{out}}}, \overrightarrow{\mathtt{z}}_0 : \overrightarrow{\neg A_0}, \overrightarrow{\mathtt{z}}_1 : \overrightarrow{\neg A_1} & \vdash^{\mathsf{v}} & \overrightarrow{W}_1 : \overrightarrow{\neg B_1} \\
\overrightarrow{\mathtt{x}}_{\mathsf{out}} : \overrightarrow{\neg A_{\mathsf{out}}}, \overrightarrow{\mathtt{x}}_0 : \overrightarrow{\neg B_0}, \overrightarrow{\mathtt{x}}_1 : \overrightarrow{\neg B_1} & \vdash^{\mathsf{n}} & M
\end{array}$$

We define transitions between tables in Fig. 4. The key result is the following:

Proposition 6
1. *There is no infinite chain of consecutive switching transitions*

$$T_0 \rightsquigarrow_{\mathrm{switch}} T_1 \rightsquigarrow_{\mathrm{switch}} T_2 \rightsquigarrow_{\mathrm{switch}} \cdots$$

2. *If* $T_0 \rightsquigarrow_{\mathrm{inner}} T_1$*, then* $\mathrm{subst}\ T_0 \rightsquigarrow \mathrm{subst}\ T_1$.
3. *If* $T_0 \rightsquigarrow_{\mathrm{switch}} T_1$*, then* $\mathrm{subst}\ T_0 = \mathrm{subst}\ T_1$.
4. *If* $T_0 \overset{ip}{\rightsquigarrow} T_1$ *then* $\mathrm{subst}\ T_0 \overset{ip}{\rightsquigarrow} \mathrm{subst}\ T_1$.
5. *Let* R *be a value-sequence table of type* $\overrightarrow{\neg C}$ *(in the rightmost column), and let* $j \in \$|\overrightarrow{\neg C}|$ *and* $q \in \mathsf{ult}^{\mathsf{v}}(C_j)$*. Then* $\mathrm{subst}\ (R : jq) = (\mathrm{subst}\ R) : jq$*.*

Proof (1) In a command-table T that is *inner-terminal* (i.e. the command is terminal), the command will be of the form $\mathtt{x}p(\overrightarrow{U})$, where $\mathtt{x}$ is declared in some column of T. We call this column $\mathrm{col}(T)$. If $T_0 \rightsquigarrow_{\mathrm{switch}} T_1$, and T_1 is itself inner-terminal, then $\mathrm{col}(T_1)$ must be to the left of $\mathrm{col}(T_0)$. To see this, suppose that T_0 is of the form (1). Then the command of T_0 is $\mathtt{x}_{m,i}p(\overrightarrow{V}_n)$, and the command of T_1 is $W_{m,i}p(\overrightarrow{\mathtt{z}}_n)$. Since T_1 is inner-terminal, $W_{m,i}$ must be an

A *command table* T is a collection of terms, either

of the form (1)

$\overrightarrow{\neg A}_{\mathsf{out}}$	$\overrightarrow{\neg B}_{\mathsf{out}}$	$\overrightarrow{\neg A}_0$	$\overrightarrow{\neg B}_0$	$\cdots$	$\overrightarrow{\neg A}_{n-1}$	$\overrightarrow{\neg B}_{n-1}$	$\vdash^{\mathsf{n}}$
$\vec{\mathsf{x}}_{\mathsf{out}}$		$\vec{V}_0$	$\vec{\mathsf{x}}_0$	$\cdots$	$\vec{V}_{n-1}$	$\vec{\mathsf{x}}_{n-1}$	M
	$\vec{\mathsf{z}}_{\mathsf{out}}$	$\vec{\mathsf{z}}_0$	$\vec{W}_0$	$\cdots$	$\vec{\mathsf{z}}_{n-1}$	$\vec{W}_{n-1}$	

or of the form (2)

$\overrightarrow{\neg A}_{\mathsf{out}}$	$\overrightarrow{\neg B}_{\mathsf{out}}$	$\overrightarrow{\neg A}_0$	$\overrightarrow{\neg B}_0$	$\cdots$	$\overrightarrow{\neg A}_{n-1}$	$\overrightarrow{\neg B}_{n-1}$	$\overrightarrow{\neg A}_n$	$\vdash^{\mathsf{n}}$
$\vec{\mathsf{x}}_{\mathsf{out}}$		$\vec{V}_0$	$\vec{\mathsf{x}}_0$	$\cdots$	$\vec{V}_{n-1}$	$\vec{\mathsf{x}}_{n-1}$	$\vec{V}_n$	
	$\vec{\mathsf{z}}_{\mathsf{out}}$	$\vec{\mathsf{z}}_0$	$\vec{W}_0$	$\cdots$	$\vec{\mathsf{z}}_{n-1}$	$\vec{W}_{n-1}$	$\vec{\mathsf{z}}_n$	M

We define the command $\vec{\mathsf{x}}_{\mathsf{out}} : \overrightarrow{\neg A}_{\mathsf{out}}, \vec{\mathsf{z}}_{\mathsf{out}} : \overrightarrow{\neg B}_{\mathsf{out}} \vdash^{\mathsf{n}} \text{subst}\ T$ to be

$$M[\vec{\mathsf{x}}_{n-1}\backslash\vec{W}_{n-1}][\vec{\mathsf{z}}_{n-1}\backslash\vec{V}_{n-1}]\cdots[\vec{\mathsf{x}}_0\backslash\vec{W}_0][\vec{\mathsf{z}}_0\backslash\vec{V}_0] \quad \text{in case (1)}$$
$$M[\vec{\mathsf{z}}_n\backslash\vec{V}_n][\vec{\mathsf{x}}_{n-1}\backslash\vec{W}_{n-1}][\vec{\mathsf{z}}_{n-1}\backslash\vec{V}_{n-1}]\cdots[\vec{\mathsf{x}}_0\backslash\vec{W}_0][\vec{\mathsf{z}}_0\backslash\vec{V}_0] \quad \text{in case (2)}$$

A *value-sequence table* R is a collection of terms, either

of the form (3)

$\overrightarrow{\neg A}_{\mathsf{out}}$	$\overrightarrow{\neg B}_{\mathsf{out}}$	$\overrightarrow{\neg A}_0$	$\overrightarrow{\neg B}_0$	$\cdots$	$\overrightarrow{\neg A}_{n-1}$	$\overrightarrow{\neg B}_{n-1}$	$\vdash^{\mathsf{v}} \overrightarrow{\neg C}$
$\vec{\mathsf{x}}_{\mathsf{out}}$		$\vec{V}_0$	$\vec{\mathsf{x}}_0$	$\cdots$	$\vec{V}_{n-1}$	$\vec{\mathsf{x}}_{n-1}$	$\vec{U}$
	$\vec{\mathsf{z}}_{\mathsf{out}}$	$\vec{\mathsf{z}}_0$	$\vec{W}_0$	$\cdots$	$\vec{\mathsf{z}}_{n-1}$	$\vec{W}_{n-1}$	

or of the form (4)

$\overrightarrow{\neg A}_{\mathsf{out}}$	$\overrightarrow{\neg B}_{\mathsf{out}}$	$\overrightarrow{\neg A}_0$	$\overrightarrow{\neg B}_0$	$\cdots$	$\overrightarrow{\neg A}_{n-1}$	$\overrightarrow{\neg B}_{n-1}$	$\overrightarrow{\neg A}_n$	$\vdash^{\mathsf{v}} \overrightarrow{\neg C}$
$\vec{\mathsf{x}}_{\mathsf{out}}$		$\vec{V}_0$	$\vec{\mathsf{x}}_0$	$\cdots$	$\vec{V}_{n-1}$	$\vec{\mathsf{x}}_{n-1}$	$\vec{V}_n$	
	$\vec{\mathsf{z}}_{\mathsf{out}}$	$\vec{\mathsf{z}}_0$	$\vec{W}_0$	$\cdots$	$\vec{\mathsf{z}}_{n-1}$	$\vec{W}_{n-1}$	$\vec{\mathsf{z}}_n$	$\vec{U}$

We define the value-sequence $\vec{\mathsf{x}}_{\mathsf{out}} : \overrightarrow{\neg A}_{\mathsf{out}}, \vec{\mathsf{z}}_{\mathsf{out}} : \overrightarrow{\neg B}_{\mathsf{out}} \vdash^{\mathsf{v}} \text{subst}\ R : \overrightarrow{\neg C}$ just as for command tables.

Figure 3. Alternating tables and substitution

identifier, declared in the context of $W_{m,i}$ i.e. somewhere to the left of column $\text{col}(T_0)$. Similarly if T_0 is of the form (2).

Hence, if there are N columns to the left of $\text{col}(T_0)$ (counting the two "outer" columns as one) then there are at most N switching transitions from T_0.

(2)–(5) are trivial. □

We say that two tables are *componentwise bisimilar* when they have the same types and identifiers, the corresponding value sequences are related by $\eqsim^{\mathsf{v}}$, and the commands (if they are command tables) are related by $\eqsim$.

Proposition 7 *If T, T' are command tables that are componentwise bisimilar, then* $\text{subst}\ T \eqsim \text{subst}\ T'$. *If R, R' are value-sequence tables that are componentwise bisimilar, then* $\text{subst}\ R \eqsim^{\mathsf{v}} \text{subst}\ R'$.

Proof Let $\mathcal{R}$ be the set of pairs $(\text{subst}\ T, \text{subst}\ T')$, where T, T' are command tables that are componentwise bisimilar. Let $\mathcal{S}$ be the set of pairs $(\text{subst}\ R, \text{subst}\ R')$, where R, R' are value-sequence tables that are componentwise bisimilar. We wish to show that $(\mathcal{R}, \mathcal{S})$ satisfy the premises of Prop. 2.

We show, by induction on n, that if $\text{subst}\ T \rightsquigarrow^n \overset{ip}{\rightsquigarrow} U$, then $T(\rightsquigarrow_{\text{inner}} \cup \rightsquigarrow_{\text{switch}})^* \overset{ip}{\rightsquigarrow} R$ where $\text{subst}\ R = U$. For this, Prop. 6 gives us $T \rightsquigarrow^*_{\text{switch}} T_1 \not\rightsquigarrow_{\text{switch}}$ with $\text{subst}\ T = \text{subst}\ T_1$ and the rest is straightforward.

We next show, by induction on n, that if $T\,\mathcal{R}\,T'$ and $T(\rightsquigarrow^*_{\text{inner}}\rightsquigarrow_{\text{switch}})^n \rightsquigarrow^*_{\text{inner}} \overset{ip}{\rightsquigarrow} R$ then $T'(\rightsquigarrow^*_{\text{inner}}\rightsquigarrow_{\text{switch}})^n \rightsquigarrow^*_{\text{inner}} \overset{ip}{\rightsquigarrow} R'$ for some $R\,\mathcal{S}\,R'$; and hence $\text{subst}\ T' \rightsquigarrow^* \overset{ip}{\rightsquigarrow} \text{subst}\ R'$. The inductive step uses Prop. 3.

These two facts give us the required property of $\mathcal{R}$. The required property of $\mathcal{S}$ is trivial. □

The first part of Prop. 4 is now given by

$$M[\overrightarrow{W/\mathsf{y}}] = \text{subst}\ \begin{array}{|c|c||c||c|}\hline \epsilon & \overrightarrow{\neg A} & \overrightarrow{\neg B} & \vdash^{\mathsf{n}} \\ \hline & & \vec{\mathsf{y}} & M \\ & \vec{\mathsf{x}} & \vec{W} & \\ \hline \end{array}$$
$$\eqsim \text{subst}\ \begin{array}{|c|c||c||c|}\hline \epsilon & \overrightarrow{\neg A} & \overrightarrow{\neg B} & \vdash^{\mathsf{n}} \\ \hline & & \vec{\mathsf{y}} & M' \\ & \vec{\mathsf{x}} & \vec{W'} & \\ \hline \end{array} = M'[\overrightarrow{W'/\mathsf{y}}]$$

and the second part is similar.

Let T be of the form (1). There are 3 possibilities for M:

- If $M \rightsquigarrow M'$, we have an *inner transition*

$T \quad \rightsquigarrow_{\text{inner}}$

$\overrightarrow{\neg A}_{\mathsf{out}}$	$\overrightarrow{\neg B}_{\mathsf{out}}$	$\overrightarrow{\neg A}_0$	$\overrightarrow{\neg B}_0$	$\cdots$	$\overrightarrow{\neg A}_{n-1}$	$\overrightarrow{\neg B}_{n-1}$	$\vdash^{\mathsf{n}}$
$\vec{\mathtt{x}}_{\mathsf{out}}$		$\vec{V}_0$	$\vec{\mathtt{x}}_0$	$\cdots$	$\vec{V}_{n-1}$	$\vec{\mathtt{x}}_{n-1}$	M'
	$\vec{\mathtt{z}}_{\mathsf{out}}$	$\vec{\mathtt{z}}_0$	$\vec{W}_0$	$\cdots$	$\vec{\mathtt{z}}_{n-1}$	$\vec{W}_{n-1}$	

- If $M = \mathtt{x}_{m,i}p(\vec{U})$, we have a *switching transition*

$T \quad \rightsquigarrow_{\text{switch}}$

$\overrightarrow{\neg A}_{\mathsf{out}}$	$\overrightarrow{\neg B}_{\mathsf{out}}$	$\overrightarrow{\neg A}_0$	$\overrightarrow{\neg B}_0$	$\cdots$	$\overrightarrow{\neg A}_{n-1}$	$\overrightarrow{\neg B}_{n-1}$	$H(p)$	$\vdash^{\mathsf{n}}$
$\vec{\mathtt{x}}_{\mathsf{out}}$		$\vec{V}_0$	$\vec{\mathtt{x}}_0$	$\cdots$	$\vec{V}_{n-1}$	$\vec{\mathtt{x}}_{n-1}$	$\vec{U}$	
	$\vec{\mathtt{z}}_{\mathsf{out}}$	$\vec{\mathtt{z}}_0$	$\vec{W}_0$	$\cdots$	$\vec{\mathtt{z}}_{n-1}$	$\vec{W}_{n-1}$	$\vec{\mathtt{z}}_n$	$W_{m,i}p(\vec{\mathtt{z}}_n)$

- If $M = \mathtt{x}_{\mathsf{out},i}p(\vec{U})$, we have an *outer Proponent move*

$T \quad \overset{ip}{\rightsquigarrow}$

$\overrightarrow{\neg A}_{\mathsf{out}}$	$\overrightarrow{\neg B}_{\mathsf{out}}$	$\overrightarrow{\neg A}_0$	$\overrightarrow{\neg B}_0$	$\cdots$	$\overrightarrow{\neg A}_{n-1}$	$\overrightarrow{\neg B}_{n-1}$	$\vdash^{\mathsf{v}} H(p)$
$\vec{\mathtt{x}}_{\mathsf{out}}$		$\vec{V}_0$	$\vec{\mathtt{x}}_0$	$\cdots$	$\vec{V}_{n-1}$	$\vec{\mathtt{x}}_{n-1}$	$\vec{U}$
	$\vec{\mathtt{z}}_{\mathsf{out}}$	$\vec{\mathtt{z}}_0$	$\vec{W}_0$	$\cdots$	$\vec{\mathtt{z}}_{n-1}$	$\vec{W}_{n-1}$	

The case where T is of the form (2) is similar, with $T \overset{ip}{\rightsquigarrow}$ replaced by $T \overset{(|\vec{\mathtt{x}}|+i)p}{\rightsquigarrow}$.
Let R be of the form (3). For $j \in \$|\overrightarrow{\neg C}|$ and $q \in \mathsf{ult}^{\mathsf{v}}(E_j)$, we define

$R : jq \quad \stackrel{\text{def}}{=}$

$\overrightarrow{\neg A}_{\mathsf{out}}, H(q)$	$\overrightarrow{\neg B}_{\mathsf{out}}$	$\overrightarrow{\neg A}_0$	$\overrightarrow{\neg B}_0$	$\cdots$	$\overrightarrow{\neg A}_{n-1}$	$\overrightarrow{\neg B}_{n-1}$	$\vdash^{\mathsf{n}}$
$\vec{\mathtt{x}}_{\mathsf{out}}, \vec{\mathtt{y}}$		$\vec{V}_0$	$\vec{\mathtt{x}}_0$	$\cdots$	$\vec{V}_{n-1}$	$\vec{\mathtt{x}}_{n-1}$	$U_j q(\vec{\mathtt{y}})$
	$\vec{\mathtt{z}}_{\mathsf{out}}$	$\vec{\mathtt{z}}_0$	$\vec{W}_0$	$\cdots$	$\vec{\mathtt{z}}_{n-1}$	$\vec{W}_{n-1}$	

where $\vec{\mathtt{y}}$ is fresh (this is called an *outer Opponent move*). The case where R is of the form (4) is similar.

Figure 4. Transitions between alternating tables

4 Call-By-Push-Value

We now turn to the study of direct-style computation. For this we use call-by-push-value (CBPV) [11, 9] a general calculus subsuming both call-by-value (CBV) and call-by-name (CBN). It has two kinds of types, *value types* A and *computation types* $\underline{B}$. They are given (including recursive types) by

$$\begin{array}{rcl} A & ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \mid \mathtt{X} \mid \mu\mathtt{X}.A \\ \underline{B} & ::= & FA \mid \prod_{i\in I}\underline{B}_i \mid A \to \underline{B} \mid \underline{\mathtt{Y}} \mid \mu\underline{\mathtt{Y}}.\underline{B} \end{array}$$

where I is any finite set. The syntax is shown in Fig. 5. There are two kinds of terms: values, written $\Gamma \vdash^{\mathsf{v}} V : A$, and computations, written $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$. We write application in operand-first order, using the ‘ symbol. A function in CBPV is a computation, by contrast with CBV.

From the cpo viewpoint, a value type denotes an unpointed cpo and a computation type denotes a pointed cpo. In particular F denotes lift, and $U\underline{B}$ denotes the same as $\underline{B}$.

Operational semantics We present operational semantics using a stack, in the "CK-machine" style of [6]. To evaluate M `to` $\mathtt{x}.\ N$, we first evaluate M until it becomes `return` V, then proceed to evaluate $N[V/\mathtt{x}]$. During the first part, we do not require the second piece of the term `to` $\mathtt{x}.\ N$, so we place it on the stack. Similarly, to evaluate V‘M, we first evaluate M until it becomes $\lambda\mathtt{x}.N$, then proceed to evaluate $N[V/\mathtt{x}]$. During the first part, we do not require the operand V, so we place it on the stack.

In the course of evaluating a computation $\Gamma \vdash^{\mathsf{c}} M_0 : \underline{C}$, the machine at any time has the form M, K. If $\underline{B}$ is the type of M, we write $\Gamma|\underline{B} \vdash^{\mathsf{k}} K : \underline{C}$ to mean that K is well-typed in this situation. A stack from F type is called a *continuation*. (For CBV, continuations are the only stacks used.) Finally, we write $\Gamma \vdash^{\mathsf{config}} R : \underline{C}$ when R is such a configuration M, K. The CK-machine and its typing rules are shown in Fig. 6

CBV and CBN fragments Both CBV and CBN calculi are

$$\frac{}{\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A}{\Gamma \vdash^{\mathsf{c}} \mathtt{return}\ V : FA} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : FA \quad \Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} N : \underline{B}}{\Gamma \vdash^{\mathsf{c}} M\ \mathtt{to}\ \mathtt{x}.\ N : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{v}} \mathtt{thunk}\ M : U\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : U\underline{B}}{\Gamma \vdash^{\mathsf{c}} \mathtt{force}\ V : \underline{B}}$$

$$\frac{\hat{\imath} \in I \quad \Gamma \vdash^{\mathsf{v}} V : A_{\hat{\imath}}}{\Gamma \vdash^{\mathsf{v}} \langle \hat{\imath}, V \rangle : \sum_{i \in I} A_i} \qquad \frac{\begin{array}{c}\Gamma \vdash^{\mathsf{v}} V : \sum_{i \in I} A_i \\ \Gamma, \mathtt{x} : A_i \vdash^{\mathsf{c}} M_i : \underline{B}\ \ (\forall i \in I)\end{array}}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x} \rangle . M_i\}_{i \in I} : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{v}} V' : A'}{\Gamma \vdash^{\mathsf{v}} \langle V, V' \rangle : A \times A'} \qquad \frac{\begin{array}{c}\Gamma \vdash^{\mathsf{v}} V : A \times A' \\ \Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{c}} M : \underline{B}\end{array}}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle . M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M_i : \underline{B}_i\ \ (\forall i \in I)}{\Gamma \vdash^{\mathsf{c}} \lambda\{i.M_i\}_{i \in I} : \prod_{i \in I} \underline{B}_i} \qquad \frac{\hat{\imath} \in I \quad \Gamma \vdash^{\mathsf{c}} M : \prod_{i \in I} \underline{B}_i}{\Gamma \vdash^{\mathsf{c}} \hat{\imath}\text{‘}M : \underline{B}_{\hat{\imath}}}$$

$$\frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B}}{\Gamma \vdash^{\mathsf{c}} \lambda \mathtt{x}.M : A \to \underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{c}} M : A \to \underline{B}}{\Gamma \vdash^{\mathsf{c}} V\text{‘}M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{v}} V : A[\mu \mathtt{X}.A/\mathtt{X}]}{\Gamma \vdash^{\mathsf{v}} \mathtt{fold}\ V : \mu \mathtt{X}.A} \qquad \frac{\begin{array}{c}\Gamma \vdash^{\mathsf{v}} V : \mu \mathtt{X}.A \\ \Gamma, \mathtt{x} : A[\mu \mathtt{X}.A/\mathtt{X}] \vdash^{\mathsf{c}} M : \underline{B}\end{array}}{\Gamma \vdash^{\mathsf{c}} \mathtt{pm}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M : \underline{B}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}[\mu \underline{\mathtt{Y}}.\underline{B}/\underline{\mathtt{Y}}]}{\Gamma \vdash^{\mathsf{c}} \mathtt{fold}\ M : \mu \underline{\mathtt{Y}}.\underline{B}} \qquad \frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B}[\mu \underline{\mathtt{Y}}.\underline{B}/\underline{\mathtt{Y}}]}{\Gamma \vdash^{\mathsf{c}} \mathtt{unfold}\ M : \mu \underline{\mathtt{Y}}.\underline{B}}$$

Figure 5. Syntax of CBPV, with type recursion

Initial configuration to evaluate $\Gamma \vdash^{\mathsf{c}} M_0 : \underline{C}$

$M_0,$	$\mathtt{nil}$

Transitions

$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M,$	K	$\leadsto$	$M[V/\mathtt{x}],$	K
$M\ \mathtt{to}\ \mathtt{x}.\ N,$	K	$\leadsto$	$M,$	$\mathtt{to}\ \mathtt{x}.\ N :: K$
$\mathtt{return}\ V,$	$\mathtt{to}\ \mathtt{x}.\ N :: K$	$\leadsto$	$N[V/\mathtt{x}],$	K
$\mathtt{force}\ \mathtt{thunk}\ M,$	K	$\leadsto$	$M,$	K
$\mathtt{pm}\ \langle \hat{\imath}, V \rangle\ \mathtt{as}\ \{\langle i, \mathtt{x} \rangle . M_i\}_{i \in I},$	K	$\leadsto$	$M_{\hat{\imath}}[V/\mathtt{x}],$	K
$\mathtt{pm}\ \langle V, V' \rangle\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle . M,$	K	$\leadsto$	$M[V/\mathtt{x}, V'/\mathtt{y}],$	K
$\hat{\imath}\text{‘}M,$	K	$\leadsto$	$M,$	$\hat{\imath} :: K$
$\lambda\{i.M_i\}_{i \in I},$	$\hat{\imath} :: K$	$\leadsto$	$M_{\hat{\imath}},$	K
$V\text{‘}M,$	K	$\leadsto$	$M,$	$V :: K$
$\lambda \mathtt{x}.M,$	$V :: K$	$\leadsto$	$M[V/\mathtt{x}],$	K
$\mathtt{pm}\ \mathtt{fold}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M,$	K	$\leadsto$	$M[V/\mathtt{x}],$	K
$\mathtt{unfold}\ M,$	K	$\leadsto$	$M,$	$\mathtt{unfold} :: K$
$\mathtt{fold}\ N,$	$\mathtt{unfold} :: K$	$\leadsto$	$N,$	K

Terminal configurations

$\mathtt{return}\ V,$	$\mathtt{nil}$
$\lambda\{i.M_i\}_{i \in I},$	$\mathtt{nil}$
$\lambda \mathtt{x}.M,$	$\mathtt{nil}$
$\mathtt{fold}\ N,$	$\mathtt{nil}$
$\mathtt{force}\ \mathtt{z},$	K
$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \{\langle i, \mathtt{x} \rangle .\ M_i\}_{i \in I},$	K
$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y} \rangle .\ M,$	K
$\mathtt{pm}\ \mathtt{z}\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M,$	K

Typing rules for stacks and configurations

$$\frac{}{\Gamma | \underline{C} \vdash^{\mathsf{k}} \mathtt{nil} : \underline{C}} \qquad \frac{\Gamma, \mathtt{x} : A \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma | \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma | FA \vdash^{\mathsf{k}} \mathtt{to}\ \mathtt{x}.\ M :: K : \underline{C}}$$

$$\frac{\hat{\imath} \in I \quad \Gamma | \underline{B}_{\hat{\imath}} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma | \prod_{i \in I} \underline{B}_i \vdash^{\mathsf{k}} \hat{\imath} :: K : \underline{C}} \qquad \frac{\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma | \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma | A \to \underline{B} \vdash^{\mathsf{k}} V :: K : \underline{C}}$$

$$\frac{\Gamma | \underline{B}[\mu \underline{\mathtt{Y}}.\underline{B}/\underline{\mathtt{Y}}] \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma | \mu \underline{\mathtt{Y}}.\underline{B} \vdash^{\mathsf{k}} \mathtt{unfold} :: K : \underline{C}}$$

$$\frac{\Gamma \vdash^{\mathsf{c}} M : \underline{B} \quad \Gamma | \underline{B} \vdash^{\mathsf{k}} K : \underline{C}}{\Gamma \vdash^{\mathsf{config}} M, K : \underline{C}}$$

Figure 6. CK-machine for CBPV

fragments of CBPV[3]. A CBV type is a value type, whilst a CBN type is a computation type. In particular,

$$\begin{array}{ll} \mathtt{bool}_{\mathbf{CBV}} = 1 + 1, & A \to_{\mathbf{CBV}} B = U(A \to FB), \\ \mathtt{bool}_{\mathbf{CBN}} = F(1+1), & \underline{A} \to_{\mathbf{CBN}} \underline{B} = U\underline{A} \to \underline{B}. \end{array}$$

A CBV term $\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1} \vdash M : B$ is represented in CBPV as $\mathtt{x}_0 : A_0, \ldots, \mathtt{x}_{n-1} : A_{n-1} \vdash^{\mathsf{c}} M : FB$, whilst a CBN term $\mathtt{x}_0 : \underline{A}_0, \ldots, \mathtt{x}_{n-1} : \underline{A}_{n-1} \vdash M : \underline{B}$ is represented as $\mathtt{x}_0 : U\underline{A}_0, \ldots, \mathtt{x}_{n-1} : U\underline{A}_{n-1} \vdash^{\mathsf{c}} M : \underline{B}$.

4.1 StkPS transform

We can translate CBPV into JWA by regarding a computation as a nonreturning command with an extra

[3]This is a very strong claim, which is justified by the fact that the embeddings of CBV and CBN in CBPV preserve a wide range of operational and denotational semantics [9].

parameter—its stack $\mathtt{k}$. This is called the *stack-passing style* transform, which generalizes the CBV CPS transform (as well as the CBN StkPS transform in [15]), and is presented in Fig. 7.

The key notion is that a computation type $\underline{B}$ is translated into the set of stacks from $\underline{B}$. For example, a stack from $A \to \underline{B}$ is $V :: K$, i.e. a value V of type A and a stack K from $\underline{B}$, so we translate $A \to \underline{B}$ as $\overline{A} \times \overline{\underline{B}}$, and translate the stack $V :: K$ as just a pair.

The transform relates the CK-machine to the C-machine in lockstep:

Proposition 8
- *The StkPS transform preserves substitution.*
- *Let $\Gamma \vdash^{\mathsf{config}} R : \underline{C}$ be a configuration. If R is terminal, then $\overline{R}$ is terminal. If $R \leadsto R'$ then $\overline{R} \leadsto \overline{R'}$.*

In Sect. 4.2–4.3, we wish to pull back the definition of normal form bisimulation from JWA to CBPV.

4.2 Ultimate Pattern-Matching For CBPV

All of Sect. 2.2, in particular Prop. 1, holds for CBPV, with $\neg A$ replaced throughout by $U\underline{A}$. But there is also a decomposition theorem for stacks. We write

$$\Gamma \vdash^{\mathsf{v}} \overrightarrow{V} : \overrightarrow{A}; \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$$

as shorthand for $\Gamma \vdash^{\mathsf{v}} \overrightarrow{V : A}$ and $\Gamma | \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$, and we call $\overrightarrow{V}; K$ a *value sequence with stack*. We want to decompose a stack as an ultimate stack-pattern applied to a value sequence with stack. For example, consider the stack

$$\langle i_0, \langle \lambda\mathtt{x}.M, \mathtt{y}\rangle\rangle :: i_1 :: \langle i_2, \mathtt{y}\rangle :: \mathtt{to}\,\mathtt{z}.M :: i_3 :: \mathtt{to}\,\mathtt{w}.N :: \mathtt{nil}$$

Assuming that $\mathtt{y}$ has U type, the stack decompses into the ultimate stack-pattern

$$\langle i_0, \langle -_{U\underline{A}}, -_{U\underline{B}}\rangle\rangle :: i_1 :: \langle i_2, -_{U\underline{B}}\rangle ::=_{FC}$$

for appropriate types $\underline{A}, \underline{B}, C$, filled with the value sequence with stack

$$\lambda\mathtt{x}.M, \mathtt{y}, \mathtt{y};\ \mathtt{to}\,\mathtt{z}.M :: i_3 :: \mathtt{to}\,\mathtt{w}.N :: \mathtt{nil}$$

The values fill the $-$ holes, and the stack fills the $=$ hole.

For each type $\underline{B}$ we define the set $\mathsf{ult}^{\mathsf{k}}(\underline{B})$ of ultimate stack-patterns from $\underline{B}$, by mutual induction:

- $=_{FA} \in \mathsf{ult}^{\mathsf{k}}(FA)$
- if $p \in \mathsf{ult}^{\mathsf{v}}(A)$ and $p' \in \mathsf{ult}^{\mathsf{k}}(\underline{B})$ then $p :: p' \in \mathsf{ult}^{\mathsf{k}}(A \to \underline{B})$
- if $\hat{\imath} \in I$ and $p \in \mathsf{ult}^{\mathsf{k}}(\underline{B}_{\hat{\imath}})$ then $\hat{\imath} :: p \in \mathsf{ult}^{\mathsf{k}}(\prod_{i\in I}\underline{B}_i)$

$\overrightarrow{\mathtt{X}}, \overrightarrow{\underline{\mathtt{Y}}} \vdash A$ val. type	$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{Y}} \vdash \overline{A}$ type
$U\underline{B}$	$\neg\overline{\underline{B}}$
$\sum_{i\in I}A_i$	$\sum_{i\in I}\overline{A_i}$
$A \times A'$	$\overline{A} \times \overline{A'}$
$\mathtt{X}$	$\mathtt{X}$
$\mu\mathtt{X}.A$	$\mu\mathtt{X}.\overline{A}$

$\overrightarrow{\mathtt{X}}, \overrightarrow{\underline{\mathtt{Y}}} \vdash \underline{B}$ comp. type	$\overrightarrow{\mathtt{X}}, \overrightarrow{\mathtt{Y}} \vdash \overline{\underline{B}}$ type
FA	$\neg\overline{A}$
$\prod_{i\in I}\underline{B}_i$	$\sum_{i\in I}\overline{\underline{B}_i}$
$A \to \underline{B}$	$\overline{A} \times \overline{\underline{B}}$
$\underline{\mathtt{Y}}$	$\mathtt{Y}$
$\mu\underline{\mathtt{Y}}.\underline{B}$	$\mu\mathtt{Y}.\overline{\underline{B}}$

$\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{v}} V : B$	$\overrightarrow{\mathtt{x} : \overline{A}} \vdash^{\mathsf{v}} \overline{V} : \overline{B}$
$\mathtt{x}$	$\mathtt{x}$
$\langle \hat{\imath}, V\rangle$	$\langle \hat{\imath}, \overline{V}\rangle$
$\langle V, V'\rangle$	$\langle \overline{V}, \overline{V'}\rangle$
$\mathtt{thunk}\ M$	$\lambda\mathtt{k}.\overline{M}$
$\mathtt{fold}\ V$	$\mathtt{fold}\ \overline{V}$

$\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{c}} M : \underline{B}$	$\overrightarrow{\mathtt{x} : \overline{A}}, \mathtt{k} : \overline{\underline{B}} \vdash^{\mathsf{n}} \overline{M}$
$\mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M$	$\mathtt{let}\ \overline{V}\ \mathtt{be}\ \mathtt{x}.\ \overline{M}$
$\mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\ M_i\}_{i\in I}$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ \{\langle i, \mathtt{x}\rangle.\overline{M_i}\}_{i\in I}$
$\mathtt{pm}\ V\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ M$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{y}\rangle.\ \overline{M}$
$\mathtt{return}\ V$	$\mathtt{k}\overline{V}$
$M\ \mathtt{to}\ \mathtt{x}.\ N$	$\mathtt{let}\ \lambda\mathtt{x}.\overline{N}\ \mathtt{be}\ \mathtt{k}.\ \overline{M}$
$\lambda\{i.M_i\}_{i\in I}$	$\mathtt{pm}\ \mathtt{k}\ \mathtt{as}\ \{\langle i, \mathtt{k}\rangle.\ \overline{M_i}\}_{i\in I}$
$\hat{\imath}\text{`}M$	$\mathtt{let}\ \langle \hat{\imath}, \mathtt{k}\rangle\ \mathtt{be}\ \mathtt{k}.\ \overline{M}$
$\lambda\mathtt{x}.M$	$\mathtt{pm}\ \mathtt{k}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{k}\rangle.\overline{M}$
$V\text{`}M$	$\mathtt{let}\ \langle \overline{V}, \mathtt{k}\rangle\ \mathtt{be}\ \mathtt{k}.\ \overline{M}$
$\mathtt{force}\ V$	$\overline{V}\mathtt{k}$
$\mathtt{pm}\ V\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ M$	$\mathtt{pm}\ \overline{V}\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}.\ \overline{M}$
$\mathtt{fold}\ M$	$\mathtt{pm}\ \mathtt{k}\ \mathtt{as}\ \mathtt{fold}\ \mathtt{k}.\ \overline{M}$
$\mathtt{unfold}\ M$	$\mathtt{let}\ \mathtt{fold}\ \mathtt{k}\ \mathtt{be}\ \mathtt{k}.\ \overline{M}$

$\overrightarrow{\mathtt{x} : A} \mid \underline{B} \vdash^{\mathsf{k}} K : \underline{C}$	$\overrightarrow{\mathtt{x} : \overline{A}}, \mathtt{nil} : \overline{\underline{C}} \vdash^{\mathsf{v}} \overline{K} : \overline{\underline{B}}$
$\mathtt{nil}$	$\mathtt{nil}$
$\mathtt{to}\ \mathtt{x}.\ M :: K$	$\lambda\mathtt{x}.(\overline{M}[\overline{K}/\mathtt{k}])$
$\hat{\imath} :: K$	$\langle \hat{\imath}, \overline{K}\rangle$
$V :: K$	$\langle \overline{V}, \overline{K}\rangle$
$\mathtt{unfold}\ :: K$	$\mathtt{fold}\ \overline{K}$

$\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{config}} R : \underline{B}$	$\overrightarrow{\mathtt{x} : \overline{A}}, \mathtt{nil} : \overline{\underline{B}} \vdash^{\mathsf{n}} \overline{R}$
M, K	$\overline{M}[\overline{K}/\mathtt{k}]$

Figure 7. StkPS transform from CBPV to JWA

- if $p \in \mathsf{ult}^{\mathsf{k}}(\underline{B}[\mu\underline{\mathtt{Y}}.\underline{B}/\underline{\mathtt{Y}}])$ then $\mathtt{unfold} :: p \in \mathsf{ult}^{\mathsf{k}}(\mu\underline{\mathtt{Y}}.\underline{B})$.

For $p \in \mathsf{ult}^{\mathsf{k}}(\underline{B})$, we write $H^{\mathsf{v}}(p)$ fir the list of types (all U types) of the $-$ holes of p, and $H^{\mathsf{k}}(p)$ for the type (an F type) of the $=$ hole. Given a value sequence with stack $\Gamma \vdash^{\mathsf{v}} \overrightarrow{V} : H^{\mathsf{v}}(p); H^{\mathsf{k}}(p) \vdash^{\mathsf{k}} K : \underline{C}$, we define the stack $\Gamma|\underline{B} \vdash^{\mathsf{k}} p(\overrightarrow{V}; K) : \underline{C}$ by filling the $-$ holes with $\overrightarrow{V}$ and the $=$ hole with K. We can now state our decomposition theorem for stacks.

Proposition 9 *Let $\overrightarrow{\mathtt{x} : U\underline{A}}|\underline{B} \vdash^{\mathsf{k}} K : FC$ be a stack. Then there is a unique ultimate stack-pattern $p \in \mathsf{ult}^{\mathsf{k}}(\underline{B})$ and value sequence with stack $\overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{v}} \overrightarrow{W} : H^{\mathsf{v}}(p); H^{\mathsf{k}}(p) \vdash^{\mathsf{k}} L : FC$ such that that $K = p(\overrightarrow{W}; L)$.*

In the obvious way, we define the StkPS transform on patterns and show that it commutes with hole-filling and is bijective at each CBPV type.

4.3 Normal Form Bisimulation For CBPV

Any configuration $\overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{config}} R : FC$ that is terminal must either be of the form $\mathtt{force}\ \mathtt{x}_i, p(\overrightarrow{V}; K)$ or of the form $\mathtt{return}\ p(\overrightarrow{V}), \mathtt{nil}$. We regard i (in the first case) and p as observable, so we write

$$R \overset{\mathsf{Q}ip}{\rightsquigarrow} \quad \overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{v}} \overrightarrow{V} : H^{\mathsf{v}}(p); H^{\mathsf{k}}(p) \vdash^{\mathsf{k}} K : FC$$

in the first case (the action is called a "Proponent Question move"), and

$$R \overset{\mathsf{A}p}{\rightsquigarrow} \quad \overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{v}} \overrightarrow{V} : H(p)$$

in the second case (called a "Proponent Answer move").

Suppose we are given a value sequence with stack $\overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{v}} \overrightarrow{V} : \overrightarrow{U\underline{B}} : FB \vdash^{\mathsf{k}} K : FC$.

- For $j \in \$|\overrightarrow{U\underline{B}}|$ and $q \in \mathsf{ult}^{\mathsf{k}}(\underline{B}_j)$ we write $(\overrightarrow{V}; K) : \mathsf{Q}jq$ for the configuration

  $$\begin{array}{l}\overrightarrow{\mathtt{x} : U\underline{A}}, \overrightarrow{\mathtt{y}} : H^{\mathsf{v}}(q) \vdash^{\mathsf{config}} \\ \mathtt{force}\ V_j, q(\overrightarrow{\mathtt{y}}; \mathtt{nil}\ ) : H^{\mathsf{k}}(q)\end{array}$$

  where $\overrightarrow{\mathtt{y}}$ are fresh. ("Opponent Question move".)

- For $q \in \mathsf{ult}^{\mathsf{v}}(B)$ we write $(\overrightarrow{V}; K) : \mathsf{A}q$ for the configuration

  $$\overrightarrow{\mathtt{x} : U\underline{A}}, \overrightarrow{\mathtt{y}} : H(q) \vdash^{\mathsf{config}} \mathtt{return}\ q(\overrightarrow{\mathtt{y}}), K : FC$$

  where $\overrightarrow{\mathtt{y}}$ are fresh. ("Opponent Answer move".)

Suppose we are given a value sequence $\overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{v}} \overrightarrow{V : U\underline{B}}$. For $j \in \$|\overrightarrow{U\underline{B}}|$ and $q \in \mathsf{ult}^{\mathsf{k}}(\underline{B}_j)$ we write $\overrightarrow{V} : \mathsf{Q}jq$ for the configuration

$$\begin{array}{l}\overrightarrow{\mathtt{x} : U\underline{A}}, \overrightarrow{\mathtt{y}} : H^{\mathsf{v}}(q) \vdash^{\mathsf{config}} \\ \mathtt{force}\ V_j, q(\overrightarrow{\mathtt{y}}; \mathtt{nil}\ ) : H^{\mathsf{k}}(q)\end{array}$$

where $\overrightarrow{\mathtt{y}}$ are fresh. ("Opponent Question move".)

Definition 3 *Let $\mathcal{R}$ associate to each context $\overrightarrow{\mathtt{x} : U\underline{A}}$ and type FB a binary relation on configurations $\overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{config}} R : FB$.*

1. *Let $\overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{v}} \overrightarrow{V}, \overrightarrow{V'} : \overrightarrow{U\underline{B}}$ be two value sequences. We say $\overrightarrow{V}\, \mathcal{R}^{\mathsf{v}}\, \overrightarrow{V'}$ when, for any $j \in \$|\overrightarrow{U\underline{B}}|$ and $q \in \mathsf{ult}^{\mathsf{k}}(\underline{B}_j)$, we have $\overrightarrow{V} : \mathsf{Q}jq\, \mathcal{R}\, \overrightarrow{V'} : \mathsf{Q}jq$*

2. *Let $\overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{v}} \overrightarrow{V}, \overrightarrow{V'} : \overrightarrow{U\underline{B}}; FB \vdash^{\mathsf{k}} K, K' : FC$ be two value sequences with stack. Then we say $\overrightarrow{V}; K\, \mathcal{R}^{\mathsf{vk}}\, \overrightarrow{V'}; K'$ when*

   - *for any $j \in \$|\overrightarrow{U\underline{B}}|$ and $q \in \mathsf{ult}^{\mathsf{k}}(\underline{B}_j)$, we have*

     $$(\overrightarrow{V}; K) : \mathsf{Q}jq\, \mathcal{R}\, (\overrightarrow{V'}; K') : \mathsf{Q}jq$$

   - *for any $q \in \mathsf{ult}^{\mathsf{v}}(B)$ we have*

     $$(\overrightarrow{V}; K) : \mathsf{A}q\, \mathcal{R}\, (\overrightarrow{V'}; K') : \mathsf{A}q$$

3. *$\mathcal{R}$ is a* normal form bisimulation *when $\overrightarrow{\mathtt{x} : U\underline{A}} \vdash^{\mathsf{n}} N\, \mathcal{R}\, N'$ implies either*

   - *$N \rightsquigarrow^{\infty}$ and $N' \rightsquigarrow^{\infty}$, or*
   - *$N \rightsquigarrow^{*} \overset{\mathsf{Q}ip}{\rightsquigarrow} \overrightarrow{V}; K$ and $N' \rightsquigarrow^{*} \overset{\mathsf{Q}ip}{\rightsquigarrow} \overrightarrow{V'}; K'$ and $\overrightarrow{V}; K\, \mathcal{R}^{\mathsf{vk}}\, \overrightarrow{V'}; K'$, or*
   - *$N \rightsquigarrow^{*} \overset{\mathsf{A}p}{\rightsquigarrow} \overrightarrow{V}$ and $N' \rightsquigarrow^{*} \overset{\mathsf{A}p}{\rightsquigarrow} \overrightarrow{V'}$ and $\overrightarrow{V}\, \mathcal{R}^{\mathsf{v}}\, \overrightarrow{V'}$.*

We write $\approx$ for *normal form bisimilarity*, i.e. the greatest normal form bisimulation.

Now we have to extend this relation to arbitrary computations and values.

Definition 4 • *Given computations $\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{c}} M, M' : \underline{B}$, we say $M \approx M'$ when for each $\overrightarrow{p \in \mathsf{ult}^{\mathsf{v}}(A)}$ and $q \in \mathsf{ult}^{\mathsf{k}}(\underline{B})$, we have*

$$\begin{array}{l}\overrightarrow{\mathtt{y} : H(p)}, \overrightarrow{\mathtt{z}} : H^{\mathsf{v}}(q) \vdash^{\mathsf{config}} M[\overrightarrow{p(\overrightarrow{\mathtt{y}})/\mathtt{x}}], q(\overrightarrow{\mathtt{z}}; \mathtt{nil}) \\ \quad \approx M'[\overrightarrow{p(\overrightarrow{\mathtt{y}})/\mathtt{x}}], q(\overrightarrow{\mathtt{z}}; \mathtt{nil}) : H^{\mathsf{k}}(q)\end{array}$$

- *Given values $\overrightarrow{\mathtt{x} : A} \vdash^{\mathsf{v}} V, V' : B$, we say $V \approx V'$ when for each $\overrightarrow{p \in \mathsf{ult}^{\mathsf{v}}(A)}$, decomposing $V[\overrightarrow{p(\overrightarrow{\mathtt{y}})/\mathtt{x}}]$ as $q(\overrightarrow{W})$ and $V'[\overrightarrow{p(\overrightarrow{\mathtt{y}})/\mathtt{x}}]$ as $q'(\overrightarrow{W'})$, we have $q = q' \in \mathsf{ult}^{\mathsf{v}}(B)$ and $\overrightarrow{\mathtt{y} : H(p)} \vdash^{\mathsf{v}} \overrightarrow{W} \approx^{\mathsf{v}} \overrightarrow{W'} : H(q)$*

Proposition 10 *Computations (resp. values) of CBPV are bisimilar precisely when their StkPS transforms are.*

This follows from Prop. 8 and the last sentence of Sect. 4.2.

Corollary 11 $\approx$ *is a substitutive congruence, validating the CBPV equational theory [11].*

5 Fixed point combinators are unique

To illustrate the use of normal form bisimulation, we now prove that at each CBPV computation type, there is a unique fixpoint combinator up to normal form bisimilarity. The proof is similar to the classical result that all λ-calculus fixed point combinators have the same Böhm tree [3].

Theorem 12 *All solutions* $\vdash^{\mathsf{c}} M : U(U\underline{A} \to \underline{A}) \to \underline{A}$ *to the fixed point equation*

$$\mathtt{f} : U(U\underline{A} \to \underline{A}) \vdash^{\mathsf{c}} \mathtt{f}`M \approx (\mathtt{thunk}(\mathtt{f}`M))`\mathtt{force}\,\mathtt{f} : \underline{A}$$

are normal form bisimilar.

Proof Suppose M_1 and M_2 are both solutions. Let R relate

$$\begin{aligned}&\mathtt{f} : U(U\underline{A} \to \underline{A}), \overrightarrow{\mathtt{x} : U\underline{B}}, \overrightarrow{\mathtt{y}} : H^{\mathsf{v}}(q) \vdash^{\mathsf{config}}\\ &\quad N_1, q(\overrightarrow{\mathtt{y}}; \mathtt{nil})\ \mathcal{R}\ N_2, q(\overrightarrow{\mathtt{y}}; \mathtt{nil}) : H^{\mathsf{k}}(q)\end{aligned}$$

for all $q \in \mathsf{ult}^{\mathsf{k}}(\underline{A})$ and Γ, N_1, N_2 such that, for $i \in \{1, 2\}$,

$$\mathtt{f} : U(U\underline{A} \to \underline{A}), \overrightarrow{\mathtt{x} : U\underline{B}} \vdash^{\mathsf{c}} N_i \approx \mathtt{f}`M_i : \underline{A}. \qquad (5)$$

Observe that $\mathtt{f} : U(U\underline{A} \to \underline{A}) \vdash^{\mathsf{c}} \mathtt{f}`M_1\ \mathcal{R}\ \mathtt{f}`M_2 : \underline{A}$. Now we show that $\mathcal{R} \cup \approx$ is a normal form bisimulation. Since $\approx$ is itself a bisimulation, we only need to consider configurations $N_i, q(\overrightarrow{\mathtt{y}}; \mathtt{nil})$ related by $\mathcal{R}$. By (5) and the assumption that $\mathtt{f}`M_i$ is a fixed point,

$$\begin{aligned}&\mathtt{f} : U(U\underline{A} \to \underline{A}), \overrightarrow{\mathtt{x} : U\underline{B}}, \overrightarrow{\mathtt{y}} : H^{\mathsf{v}}(q) \vdash^{\mathsf{config}}\\ &\quad N_i, q(\overrightarrow{\mathtt{y}}; \mathtt{nil}) \approx\\ &\quad (\mathtt{thunk}(\mathtt{f}`M_i))`\mathtt{force}\,\mathtt{f}, q(\overrightarrow{\mathtt{y}}; \mathtt{nil}) : \underline{A}.\end{aligned}$$

The right hand side evaluates to

$$\mathtt{force}\,\mathtt{f}, \mathtt{thunk}(\mathtt{f}`M_i) :: q(\overrightarrow{\mathtt{y}}; \mathtt{nil}),$$

so $N_i, q(\overrightarrow{\mathtt{y}}; \mathtt{nil})$ evaluates to $\mathtt{force}\,\mathtt{f}, W_i :: q(\overrightarrow{V_i}; K_i)$, where

$$\begin{aligned}&\Gamma \vdash^{\mathsf{v}} W_i \approx \mathtt{thunk}(\mathtt{f}`M_i) : U\underline{A},\\ &\Gamma \vdash^{\mathsf{v}} \overrightarrow{V_i} \approx \overrightarrow{\mathtt{y}} : H^{\mathsf{v}}(q), &(6)\\ &\Gamma \mid H^{\mathsf{k}}(q) \vdash^{\mathsf{k}} K_i \approx \mathtt{nil} : H^{\mathsf{k}}(q), &(7)\end{aligned}$$

and $\Gamma \overset{\text{def}}{=} \mathtt{f} : U(U\underline{A} \to \underline{A}), \overrightarrow{\mathtt{x} : U\underline{B}}, \overrightarrow{\mathtt{y}} : H^{\mathsf{v}}(q)$. Hence

$$\Gamma \vdash^{\mathsf{c}} \mathtt{force}\,W_i \approx \mathtt{f}`M_i : \underline{A}$$

and thus, by the definition of $\mathcal{R}$ and $\mathcal{R}^{\mathsf{v}}$,

$$\Gamma \vdash^{\mathsf{v}} W_1\ \mathcal{R}^{\mathsf{v}}\ W_2 : U\underline{A}. \qquad (8)$$

From (8), (6), and (7) we conclude that $\mathcal{R} \cup \approx$ is a normal form bisimulation, thus

$$\mathtt{f} : U(U\underline{A} \to \underline{A}) \vdash^{\mathsf{c}} \mathtt{f}`M_1 \approx \mathtt{f}`M_2 : \underline{A}$$

and, by congruence and η-conversion, we conclude

$$\vdash^{\mathsf{c}} \lambda\mathtt{f}.\,\mathtt{f}`M_1 \approx \lambda\mathtt{f}.\,\mathtt{f}`M_2 : U(U\underline{A} \to \underline{A}) \to \underline{A}$$

and $\vdash^{\mathsf{c}} M_1 \approx M_2 : U(U\underline{A} \to \underline{A}) \to \underline{A}$. □

References

[1] S. Abramsky. The lazy λ-calculus. In D. Turner, editor, *Research Topics in Functional Prog.* Addison-Wesley, 1990.

[2] S. Abramsky and G. McCusker. Call-by-value games. In M. Nielsen and W. Thomas, editors, *Comp. Sci. Logic: 11th International Workshop Proc.*, LNCS. Springer, 1998.

[3] 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.

[4] P.-L. Curien and H. Herbelin. Computing with abstract Böhm trees. In *Fuji International Symposium on Functional and Logic Programming*, pages 20–39, 1998.

[5] V. Danos, H. Herbelin, and L. Regnier. Game semantics and abstract machines. In *Proc., 11th Annual IEEE Symposium On Logic In Computer Science*, 1996.

[6] M. Felleisen and D. P. Friedman. Control operators, the SECD-machine, the λ-calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts III*, pages 193–217. North-Holland, 1986.

[7] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[8] R. Jagadeesan, C. Pitcher, and J. Riely. Open bisimulation for aspects. In *Intl. Conf. on Aspect-Oriented Software Development*. ACM Press, 2007.

[9] P. B. Levy. *Call-By-Push-Value. A Functional/Imperative Synthesis*. Semantic Struct. in Computation. Springer, 2004.

[10] P. B. Levy. Adjunction models for call-by-push-value with stacks. *Theory and Applications of Categories*, 14, 2005.

[11] P. B. Levy. Call-by-push-value: decomposing call-by-value and call-by-name. *Higher-Order & Symb. Comp.*, 19(4), 2006.

[12] M. Merro and C. Biasi. On the observational theory of the CPS calculus. In *Proc., 22nd Conf. on the Mathematical Foundations of Programming Semantics*, volume 158 of *ENTCS*, 2006.

[13] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1), 1994.

[14] K. Støvring and S. B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *Proc. 34th ACM Symp., Principles of Programming Languages*, 2007.

[15] T. Streicher and B. Reus. Classical logic, continuation semantics and abstract machines. *Journal of Functional Programming*, 8(6):543–572, 1998.

[16] H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997.