Typed Normal Form Bisimulation

Soren B. Lassen[1] and Paul Blain Levy[2]

[1] Google, Inc.
soren@google.com
[2] 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 Boehm 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 for a continuation-passing style calculus, Jump-With-Argument (JWA), where normal form bisimilarity takes a very simple form. We give a novel congruence proof, based on insights from game semantics. A notable feature is the seamless treatment of eta-expansion. We demonstrate the normal form bisimulation proof principle by using it to establish a syntactic minimal invariance result and the uniqueness of the fixed point operator at each type.

1 Introduction

1.1 Background

Normal form bisimulation—also known as open (applicative) bisimulation—originated as a coinductive way of describing Lévy-Longo tree equivalence for the lazy λ-calculus [1], and has subsequently been extended to call-by-name, call-by-value, nondeterminism, aspects, storage, and control [2–7].

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* [8, 9], 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[3], let $G(p, q)$ be the following function

$$\lambda\mathtt{x}.\mathtt{if}\ (\mathtt{x}\,p)\ \mathtt{then}\ \mathtt{x}\,q\ \mathtt{else}\ \mathtt{not}(\mathtt{x}\,q)$$

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,

[3] This example works in both call-by-value and call-by-name.

they cannot be distinguished by applying to closed arguments. But let us apply them to a fresh identifier `y` and evaluate the resulting open terms, symbolically. Then the first begins by applying `y` to `true` with the continuation `if − then x false else not(x true)`. The second applies `y` to `false` with the continuation `if − then x true else not(x false)`.

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 (`true` and `false` respectively) and so are the continuations. Therefore V and V' are not normal form bisimilar.

This is a situation where normal form bisimulation gives an equivalence that is finer than contextual equivalence, in contrast with applicative bisimulation. However, the addition of state and suitable control effects makes normal form bisimulation coincide with contextual equivalence [2, 7].

Compared to applicative bisimulation, the absence of the universal quantification over closed arguments to functions in the definition of normal form bisimulation makes certain program equivalence proofs possible that have not been accomplished using applicative bisimulation. An example is our proof of syntactic minimal invariance in Section 4. See also the examples in [2, 4, 7].

But all the results on normal form bisimulation are in untyped settings only. The adaptation 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 that has slowed progress on normal form bisimulation is that the congruence proofs given in the literature (especially [7]) are complicated: they establish congruence for η-long terms, and separately prove the validity of the η-law.

1.2 Contributions

This paper makes three important contributions to the development of normal form bisimulation. First, we extend normal form bisimulation to types, inclusive product, sum, and higher-order types, empty types and recursive types. Second, we give a new, lucid congruence proof that highlights connections with game semantics. Third, we present a seamless treatment of η-expansion—we do not need to mention it in our definitions or proofs.

To illustrate the power of our theory, we use it to prove

- uniqueness of the fixpoint operator at each type
- syntactic minimal invariance.

It is not known how to prove the latter using applicative bisimulation, so this shows a definite advantage of normal form bisimulation as an operational reasoning technique.

Our work is part of a larger programme to explore the scope of normal form bisimulation, both (1) as a syntactic proof principle for reasoning about program

equivalence and (2) as an operational account of pointer game semantics. Space constraints prevent us from giving a formal account of the second point but we mention some connections in our discussion of related work below and in the technical exposition of our theory we give some hints to the benefit of readers familiar with game semantics.

For simplicity, we develop our theory in a continuation-passing style calculus, called Jump-With-Argument (JWA), where it takes a very simple form. It is then trivial to adapt it to a direct-style calculus (with or without control operators), by applying the appropriate CPS transform. The pointer game semantics for JWA, and the relationship with direct-style programs, is given in [10, 11]; but here we do not assume familiarity with pointer games.

In this paper, we look at JWA without storage. In game terminology, the strategies we are studying are innocent. It is clear from the game literature that, when we extend JWA with storage, normal form bisimilarity will coincide with contextual equivalence, but we leave such an analysis to future work.

1.3 Related Work

In Section 1.1 we descibed the origins of and previous work on normal form bisimulation.

We refer the reader to [7] for a survey of other syntactic theories for reasoning about program equivalence: equational theories, context lemmas, applicative bisimulation, environmental bisimulation, and syntactic logical relations. Given that our calculus, JWA, is a CPS calculus, note that applicative bisimulation has been studied recently in a CPS setting by Merro and Biasi [12].

On the other hand, let us discuss some relevant literature on game semantics, because its relationship with normal form bisimulation has not been surveyed before.

Pointer game semantics is a form of denotational model in which a term denotes a strategy for a game with pointers between moves. It was introduced in [13], and has been used to model typed and untyped languages, call-by-name, call-by-value, recursion, storage, control operators and much more [14–19]. In general, denotational equality is finer than contextual equivalence, but in the presence of storage and control, they coincide [20]. The simplest models, for terms without local state, use *innocent* strategies, which correspond to Böhm trees [21], or variants thereof such as PCF trees, Nakajima trees and Lévy-Longo trees.

Pointer games are analyzed operationally in [22, 10], relating a term's denotation to its behaviour in an abstract machine. But these abstract machines are complex to describe and reason about. Moreover, the only terms studied in these accounts are η-long: thus [22] states "in the sequel, we will consider terms up to η-equality". This is a limitation, especially in the presence of recursive types at which terms cannot be fully η-expanded.

In [23], Sect. 1–2, a quite different operational account of a game model is given, without abstract machines or η-expansion—albeit in the limited setting of

a first-order language. It is defined in terms of an operational semantics for *open* terms (unlike traditional operational semantics, defined on closed terms only).

Clearly there are many similarities between normal form bisimulation and pointer game semantics: the connection with Böhm trees and Lévy-Longo trees, the completeness in the presence of storage and control, the use of operational semantics for open terms. Readers familiar with game semantics will immediately see that V and V' in the example in Section 1.1 denote different strategies.

An immediate precursor for our definition of typed normal form bisimulation is the labelled transition system and pointer game semantics in [24]. This work also describes an extension to mutable references, which is something we plan to explore in future work. Recently and independently, Laird [25] developed a very similar labelled transition system semantics for a typed functional language with mutable references.

The representation of strategies as π-calculus processes by Hyland, Ong, Fiore, and Honda [26, 27] leads to a bisimulation approach to equality of strategies that is analogous to normal form bisimulation. However, because of the detour via π-calculus encodings, the resulting π-calculus-based bisimulation proof principles for program equivalence are not as direct as normal form bisimulation, for proving specific program equivalences between terms.

2 Jump-With-Argument

2.1 Syntax and Semantics

Jump-With-Argument is a continuation-passing style calculus, extending the CPS calculus in [28]. Its types are given (including recursive types) by

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

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. JWA has two judgements: *values* written $\Gamma \vdash^{\mathtt{v}} V$ and *nonreturning commands* written $\Gamma \vdash^{\mathtt{n}} M$. The syntax[4] 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 $[\![A]\!] \to R$, where R is a chosen pointed cpo.

Operational semantics To evaluate a command $\Gamma \vdash^{\mathtt{n}} M$, 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 $M \rightsquigarrow^* T$ for unique terminal T, or else $M \rightsquigarrow^\infty$. This operational semantics is called the *C-machine*.

We define a fixed point combinator Y as follows

$$Y \stackrel{\text{def}}{=} \Phi(\mathtt{fold}\ \lambda\langle \mathtt{x}, \langle \mathtt{u}, \mathtt{f}\rangle\rangle.\mathtt{f}\langle \mathtt{u}, \lambda \mathtt{v}.\Phi(\mathtt{x})\langle \mathtt{v}, \mathtt{f}\rangle\rangle)$$

$$\Phi(x) \stackrel{\text{def}}{=} \lambda \mathtt{z}.\mathtt{pm}\ x\ \mathtt{as}\ \mathtt{fold}\ \mathtt{y}.\mathtt{y}\langle \mathtt{fold}\ \mathtt{y}, \mathtt{z}\rangle$$

[4] In earlier works e.g. [10], $\gamma\mathtt{x}.M$ was written for $\lambda\mathtt{x}.M$ and $W \nearrow V$ for VW.

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

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

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

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

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

Fig. 1. Syntax of JWA, with type recursion

Transitions

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

Terminal Commands

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

Fig. 2. C-machine

where notation $\lambda\langle \mathtt{x}, \langle \mathtt{u}, \mathtt{f} \rangle \rangle . M$ is short for $\lambda \mathtt{t}.\mathtt{pm\ t\ as}\ \langle \mathtt{x}, \mathtt{p} \rangle . \mathtt{pm\ p\ as}\ \langle \mathtt{u}, \mathtt{f} \rangle . M$.

Using type recursion, we assign type $\neg(A \times \neg(A \times \neg A))$ to Y, for every type A, by giving the argument to Φ type $\mu \mathtt{X}.\neg(\mathtt{X} \times (A \times \neg A))$.

Let $V = \lambda\langle \mathtt{x}, \langle \mathtt{u}, \mathtt{f} \rangle \rangle . \mathtt{f}\langle \mathtt{u}, \lambda \mathtt{v}.\Phi(\mathtt{x})\langle \mathtt{v}, \mathtt{f} \rangle \rangle$, then we calculate:

$$\begin{array}{rl}
Y\langle \mathtt{u}, \mathtt{f} \rangle & \rightsquigarrow\ \mathtt{pm\ fold}\ V\ \mathtt{as\ fold\ y}.\mathtt{y}\langle \mathtt{fold\ y}, \langle \mathtt{u}, \mathtt{f} \rangle \rangle \\
& \rightsquigarrow\ V\langle \mathtt{fold}\ V, \langle \mathtt{u}, \mathtt{f} \rangle \rangle \\
& \rightsquigarrow^3\ \mathtt{f}\langle \mathtt{u}, \lambda \mathtt{v}.\Phi(\mathtt{fold}\ V)\langle \mathtt{v}, \mathtt{f} \rangle \rangle \\
& =\ \mathtt{f}\langle \mathtt{u}, \lambda \mathtt{v}.Y\langle \mathtt{v}, \mathtt{f} \rangle \rangle
\end{array}$$

That is, Y is a solution to the fixed point equation

$$\vdash^{\mathsf{v}} Y =_\beta \lambda\langle \mathtt{u}, \mathtt{f} \rangle . \mathtt{f}\langle \mathtt{u}, \lambda \mathtt{v}.Y\langle \mathtt{v}, \mathtt{f} \rangle \rangle : \neg(A \times \neg(A \times \neg A))$$

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

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

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

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

(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[5], or (equivalently) denote the same innocent strategy, but neither of these concepts will be used in the paper.

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

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

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

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

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

(This action is called a "Proponent move". Interchangeably, we write it as $\mathtt{x}_i p$ when it is more convenient to name the identifier than its index.) Thus, for any command $\overrightarrow{\mathtt{x} : A} \vdash N$, we have either

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

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*
   - *$N \rightsquigarrow^* \overset{ip}{\rightsquigarrow} \overrightarrow{V}$ and $N' \rightsquigarrow^* \overset{ip}{\rightsquigarrow} \overrightarrow{V'}$ and $\overrightarrow{V}\, \mathcal{R}^{\mathsf{v}}\, \overrightarrow{V'}$.*

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

A *renaming* $\Gamma \xrightarrow{\theta} \Delta$ maps each identifier $\mathtt{x} : A \in \Gamma$ to an identifier $\theta(\mathtt{x}) : A \in \Delta$. This induces a map $M \mapsto M\theta$ from terms in context Γ to terms in context Δ.

Proposition 2. *(preservation under renaming)*
For any renaming $\overrightarrow{\mathtt{x} : \neg A} \xrightarrow{\theta} \overrightarrow{\mathtt{y} : \neg B}$, we have $\overrightarrow{\mathtt{x} : \neg A} \vdash M \approx M'$ implies $M\theta \approx M'\theta$, and $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{V} \approx^{\mathsf{v}} \overrightarrow{V'} : \overrightarrow{\neg C}$ implies $\overrightarrow{V\theta} \approx^{\mathsf{v}} \overrightarrow{V'\theta}$.

Proof. Let $\mathcal{R}$ be the set of pairs $(M\theta, M'\theta)$ where $\overrightarrow{\mathtt{x} : \neg A} \vdash M \approx M'$ and $\overrightarrow{\mathtt{x} : \neg A} \xrightarrow{\theta} \overrightarrow{\mathtt{y} : \neg B}$ is a renaming. Then $\overrightarrow{V\theta}\, \mathcal{R}^{\mathsf{v}}\, \overrightarrow{V'\theta}$ whenever $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{V} \approx^{\mathsf{v}} \overrightarrow{V'} : \overrightarrow{\neg C}$. It is easy to show $\mathcal{R}$ is a normal form bisimulation.

Proposition 3. *(preservation under substitution)*
Suppose $\overrightarrow{\mathtt{y} : \neg B} \vdash^{\mathsf{v}} \overrightarrow{W} \mathrel{\approx^{\mathsf{v}}} \overrightarrow{W'} : \overrightarrow{\neg A}$. *Then* $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{n}} M \approx M'$ *implies* $M[\overrightarrow{W/\mathtt{x}}] \approx M'[\overrightarrow{W'/\mathtt{x}}]$ *and* $\overrightarrow{\mathtt{x} : \neg A} \vdash^{\mathsf{v}} \overrightarrow{V} \mathrel{\approx^{\mathsf{v}}} \overrightarrow{V'} : \overrightarrow{\neg C}$ *implies* $V[\overrightarrow{W/\mathtt{x}}] \mathrel{\approx^{\mathsf{v}}} V'[\overrightarrow{W'/\mathtt{x}}]$.

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 [14]).

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

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

– *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* $V[\overrightarrow{p(\mathtt{y})/\mathtt{x}}]$ *as* $q(\overrightarrow{W})$ *and* $V'[\overrightarrow{p(\mathtt{y})/\mathtt{x}}]$ *as* $q'(\overrightarrow{W'})$, *we have* $q = q'$ *and*

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

It is easy to see that normal form bisimilarity for JWA is an equivalence relation and validates all the β and η laws [10].

Proposition 4. $\approx$ *is a substitutive congruence.*

Proof. Substitutivity follows from Prop. 3. For each term constructor, we prove it preserves $\approx$ using substitutivity, as in [7].

As a corollary, we get that normal form bisimilar terms are contextually equivalent, for any reasonable definition of contextual equivalence. The opposite is not true. For example, this equation holds for contextual equivalence:

$$\mathtt{x} : \neg\neg 1, \mathtt{y} : \neg 1 \vdash^{\mathsf{n}} \mathtt{x}\,\mathtt{y} \cong_{\mathrm{ctx}} \mathtt{x}\,(\lambda \mathtt{z}.\mathtt{x}\,\mathtt{y}) \tag{1}$$

The intuition is that either $\mathtt{x}$ ignores its argument, and then the equivalence holds trivially, or else $\mathtt{x}$ invokes its argument at some point, and from that point onward $\mathtt{x}\,(\lambda \mathtt{z}.\mathtt{x}\,\mathtt{y})$ emulates $\mathtt{x}\,\mathtt{y}$ from the beginning.[6] But $\mathtt{x}\,\mathtt{y}$ and $\mathtt{x}\,(\lambda \mathtt{z}.\mathtt{x}\,\mathtt{y})$ are not normal form bisimilar: $\mathtt{x}\,\mathtt{y} \overset{\mathtt{x}(-_{\neg 1})}{\rightsquigarrow} \mathtt{y}$ and $\mathtt{x}\,(\lambda \mathtt{z}.\mathtt{x}\,\mathtt{y}) \overset{\mathtt{x}(-_{\neg 1})}{\rightsquigarrow} \lambda \mathtt{z}.\mathtt{x}\,\mathtt{y}$ but $\mathtt{y}$ and $\lambda \mathtt{z}.\mathtt{x}\,\mathtt{y}$ are clearly not bisimilar since, given an argument $\mathtt{z}$, the labelled transitions $\mathtt{y}\,\mathtt{z} \overset{\mathtt{y}(-_1)}{\rightsquigarrow} \mathtt{z}$ and $(\lambda \mathtt{z}.\mathtt{x}\,\mathtt{y})\,\mathtt{z} \overset{\mathtt{x}(-_{\neg 1})}{\rightsquigarrow} \mathtt{y}$ mismatch.

[6] We omit both the definition of contextual equivalence and the proof of (1) but the reasoning is analogous to Thielecke's proof of Filinski's equation $M \cong_{\mathrm{ctx}} M; M$ in a direct-style calculus with exception and continuations but without state [29]. Indeed, (1) is essentially derived from Filinski and Thielecke's example by a CPS transform.

3.1 Alternating Substitution

To prove Prop. 3, 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:

$$\begin{aligned}
\overrightarrow{\mathtt{x}}_{\mathsf{out}} : \overrightarrow{\neg A}_{\mathsf{out}} &\vdash^{\mathsf{v}} \overrightarrow{V}_0 : \overrightarrow{\neg A}_0 \\
\overrightarrow{\mathtt{z}}_{\mathsf{out}} : \overrightarrow{\neg B}_{\mathsf{out}}, \overrightarrow{\mathtt{z}}_0 : \overrightarrow{\neg A}_0 &\vdash^{\mathsf{v}} \overrightarrow{W}_0 : \overrightarrow{\neg B}_0 \\
\overrightarrow{\mathtt{x}}_{\mathsf{out}} : \overrightarrow{\neg A}_{\mathsf{out}}, \overrightarrow{\mathtt{x}}_0 : \overrightarrow{\neg B}_0 &\vdash^{\mathsf{v}} \overrightarrow{V}_1 : \overrightarrow{\neg A}_1 \\
\overrightarrow{\mathtt{z}}_{\mathsf{out}} : \overrightarrow{\neg B}_{\mathsf{out}}, \overrightarrow{\mathtt{z}}_0 : \overrightarrow{\neg A}_0, \overrightarrow{\mathtt{z}}_1 : \overrightarrow{\neg A}_1 &\vdash^{\mathsf{v}} \overrightarrow{W}_1 : \overrightarrow{\neg B}_1 \\
\overrightarrow{\mathtt{x}}_{\mathsf{out}} : \overrightarrow{\neg A}_{\mathsf{out}}, \overrightarrow{\mathtt{x}}_0 : \overrightarrow{\neg B}_0, \overrightarrow{\mathtt{x}}_1 : \overrightarrow{\neg B}_1 &\vdash^{\mathsf{n}} M
\end{aligned}$$

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

Proposition 5. *1. There is no infinite chain of consecutive switching transitions* $T_0 \rightsquigarrow_{\text{switch}} T_1 \rightsquigarrow_{\text{switch}} T_2 \rightsquigarrow_{\text{switch}} \cdots$
2. If $T_0 \rightsquigarrow_{\text{inner}} T_1$*, then* subst $T_0 \rightsquigarrow$ subst T_1*.*
3. If $T_0 \rightsquigarrow_{\text{switch}} T_1$*, then* subst $T_0 =$ subst T_1*.*
4. If $T_0 \overset{ip}{\rightsquigarrow} T_1$ *then* subst $T_0 \overset{ip}{\rightsquigarrow}$ 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* subst $(R : jq) = (\text{subst } R) : jq$*.*

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 $\text{col}(T)$. If $T_0 \rightsquigarrow_{\text{switch}} T_1$, and T_1 is itself inner-terminal, then $\text{col}(T_1)$ must be to the left of $\text{col}(T_0)$. To see this, suppose that T_0 is of the form (2). 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 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 (3).

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.

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

of the form

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

(2)

or of the form

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

(3)

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

$$M[\overrightarrow{\mathtt{x}}_{n-1}\backslash\overrightarrow{W}_{n-1}][\overrightarrow{\mathtt{z}}_{n-1}\backslash\overrightarrow{V}_{n-1}]\cdots[\overrightarrow{\mathtt{x}}_0\backslash\overrightarrow{W}_0][\overrightarrow{\mathtt{z}}_0\backslash\overrightarrow{V}_0] \quad \text{in case (2)}$$
$$M[\overrightarrow{\mathtt{z}}_n\backslash\overrightarrow{V}_n][\overrightarrow{\mathtt{x}}_{n-1}\backslash\overrightarrow{W}_{n-1}][\overrightarrow{\mathtt{z}}_{n-1}\backslash\overrightarrow{V}_{n-1}]\cdots[\overrightarrow{\mathtt{x}}_0\backslash\overrightarrow{W}_0][\overrightarrow{\mathtt{z}}_0\backslash\overrightarrow{V}_0] \quad \text{in case (3)}$$

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

of the form

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

(4)

or of the form

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

(5)

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

Fig. 3. Alternating tables and substitution

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

Proof. Let $\mathcal{R}$ be the set of pairs (subst T, subst T'), where T, T' are command tables that are componentwise bisimilar. Then (subst R, subst R') $\in \mathcal{R}^{\mathsf{v}}$ whenever R, R' are value-sequence tables that are componentwise bisimilar. We wish to show that $\mathcal{R}$ is a normal form bisimulation.

We show, by induction on n, that if subst $T \rightsquigarrow^n \overset{ip}{\mathcal{R}}$ U, then $T(\rightsquigarrow_{\text{inner}} \cup \rightsquigarrow_{\text{switch}})^* \overset{ip}{\mathcal{R}}$ R where subst $R = U$. For this, Prop. 5 gives us $T \rightsquigarrow^*_{\text{switch}} T_1 \not\rightsquigarrow_{\text{switch}}$ with subst $T =$ subst T_1 and the rest is straightforward.

Let T be of the form (2). There are 3 possibilities for M:

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

$$T \quad \rightsquigarrow_{\text{inner}} \quad \begin{array}{|c|c||c|c|c|c|c||c|} \hline \overrightarrow{\neg A}_{\mathsf{out}} & \overrightarrow{\neg B}_{\mathsf{out}} & \overrightarrow{\neg A}_0 & \overrightarrow{\neg B}_0 & \cdots & \overrightarrow{\neg A}_{n-1} & \overrightarrow{\neg B}_{n-1} & \vdash^{\mathsf{n}} \\ \hline \overrightarrow{\mathtt{x}}_{\mathsf{out}} & & \overrightarrow{V}_0 & \overrightarrow{\mathtt{x}}_0 & \cdots & \overrightarrow{V}_{n-1} & \overrightarrow{\mathtt{x}}_{n-1} & M' \\ & \overrightarrow{\mathtt{z}}_{\mathsf{out}} & \overrightarrow{\mathtt{z}}_0 & \overrightarrow{W}_0 & \cdots & \overrightarrow{\mathtt{z}}_{n-1} & \overrightarrow{W}_{n-1} & \\ \hline \end{array}$$

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

$$T \quad \rightsquigarrow_{\text{switch}} \quad \begin{array}{|c|c||c|c|c|c|c|c||c|} \hline \overrightarrow{\neg A}_{\mathsf{out}} & \overrightarrow{\neg B}_{\mathsf{out}} & \overrightarrow{\neg A}_0 & \overrightarrow{\neg B}_0 & \cdots & \overrightarrow{\neg A}_{n-1} & \overrightarrow{\neg B}_{n-1} & H(p) & \vdash^{\mathsf{n}} \\ \hline \overrightarrow{\mathtt{x}}_{\mathsf{out}} & & \overrightarrow{V}_0 & \overrightarrow{\mathtt{x}}_0 & \cdots & \overrightarrow{V}_{n-1} & \overrightarrow{\mathtt{x}}_{n-1} & \overrightarrow{U} & \\ & \overrightarrow{\mathtt{z}}_{\mathsf{out}} & \overrightarrow{\mathtt{z}}_0 & \overrightarrow{W}_0 & \cdots & \overrightarrow{\mathtt{z}}_{n-1} & \overrightarrow{W}_{n-1} & \overrightarrow{\mathtt{z}}_n & W_{m,i}p(\overrightarrow{\mathtt{z}}_n) \\ \hline \end{array}$$

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

$$T \quad \overset{ip}{\rightsquigarrow} \quad \begin{array}{|c|c||c|c|c|c|c||c|} \hline \overrightarrow{\neg A}_{\mathsf{out}} & \overrightarrow{\neg B}_{\mathsf{out}} & \overrightarrow{\neg A}_0 & \overrightarrow{\neg B}_0 & \cdots & \overrightarrow{\neg A}_{n-1} & \overrightarrow{\neg B}_{n-1} & \vdash^{\mathsf{v}} H(p) \\ \hline \overrightarrow{\mathtt{x}}_{\mathsf{out}} & & \overrightarrow{V}_0 & \overrightarrow{\mathtt{x}}_0 & \cdots & \overrightarrow{V}_{n-1} & \overrightarrow{\mathtt{x}}_{n-1} & \overrightarrow{U} \\ & \overrightarrow{\mathtt{z}}_{\mathsf{out}} & \overrightarrow{\mathtt{z}}_0 & \overrightarrow{W}_0 & \cdots & \overrightarrow{\mathtt{z}}_{n-1} & \overrightarrow{W}_{n-1} & \\ \hline \end{array}$$

The case where T is of the form (3) is similar, with $T \overset{ip}{\rightsquigarrow}$ replaced by $T \overset{(|\overrightarrow{\mathtt{x}}|+i)p}{\rightsquigarrow}$.
Let R be of the form (4). For $j \in \$|\overrightarrow{\neg C}|$ and $q \in \mathsf{ult}^{\mathsf{v}}(E_j)$, we define

$$R : jq \quad \stackrel{\text{def}}{=} \quad \begin{array}{|c|c||c|c|c|c|c||c|} \hline \overrightarrow{\neg A}_{\mathsf{out}}, H(q) & \overrightarrow{\neg B}_{\mathsf{out}} & \overrightarrow{\neg A}_0 & \overrightarrow{\neg B}_0 & \cdots & \overrightarrow{\neg A}_{n-1} & \overrightarrow{\neg B}_{n-1} & \vdash^{\mathsf{n}} \\ \hline \overrightarrow{\mathtt{x}}_{\mathsf{out}}, \overrightarrow{\mathtt{y}} & & \overrightarrow{V}_0 & \overrightarrow{\mathtt{x}}_0 & \cdots & \overrightarrow{V}_{n-1} & \overrightarrow{\mathtt{x}}_{n-1} & U_j q(\overrightarrow{\mathtt{y}}) \\ & \overrightarrow{\mathtt{z}}_{\mathsf{out}} & \overrightarrow{\mathtt{z}}_0 & \overrightarrow{W}_0 & \cdots & \overrightarrow{\mathtt{z}}_{n-1} & \overrightarrow{W}_{n-1} & \\ \hline \end{array}$$

where $\overrightarrow{\mathtt{y}}$ is fresh (this is called an *outer Opponent move*). The case where R is of the form (5) is similar.

Fig. 4. Transitions between alternating tables

We next show, by induction on n, that if $(\text{subst } T, \text{subst } T') \in \mathcal{R}$ 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' componentwise bisimilar to R; and hence subst $T' \rightsquigarrow^* \overset{ip}{\rightsquigarrow}$ subst R'. The inductive step uses Prop. 2.

These two facts give us the required property of $\mathcal{R}$.

The first part of Prop. 3 is now given by

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

and the second part is similar.

4 Examples

4.1 Fixed point combinators are unique

To illustrate the use of normal form bisimulation, we now prove that at each 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 [30].

Theorem 1. *All solutions* $\vdash^{\mathsf{v}} U : \neg(A \times \neg(A \times \neg A))$ *to the fixed point equation*

$$\vdash^{\mathsf{v}} U \approx^{\mathsf{v}} \lambda\langle \mathtt{u}, \mathtt{f}\rangle.\mathtt{f}\langle \mathtt{u}, \lambda \mathtt{v}.U\langle \mathtt{v}, \mathtt{f}\rangle\rangle : \neg(A \times \neg(A \times \neg A))$$

are normal form bisimilar.

Proof. Suppose U_1 and U_2 are both solutions. We show they are bisimilar

$$\vdash^{\mathsf{v}} U_1 \approx^{\mathsf{v}} U_2 : \neg(A \times \neg(A \times \neg A)) \tag{6}$$

by exhibiting a bisimulation relation that relates

$$\overrightarrow{\mathtt{y}} : H(q) \vdash^{\mathsf{n}} U_1 q(\overrightarrow{\mathtt{y}}), U_2 q(\overrightarrow{\mathtt{y}}) \tag{7}$$

for all $q \in \mathsf{ult}^{\mathsf{v}}(A \times \neg(A \times \neg A))$, that is, all $q = \langle p, -_{\neg(A\times\neg A)}\rangle$ where $p \in \mathsf{ult}^{\mathsf{v}}(A)$.

We define $\mathcal{R}$ to be the relation between all commands $\Gamma \vdash^{\mathsf{n}} M_1, M_2$ where

$$\Gamma \vdash^{\mathsf{n}} M_i \approx U_i\langle p(\overrightarrow{\mathtt{x}}), \mathtt{f}\rangle, \quad \text{for } i \in \{1,2\},$$

and $p \in \mathsf{ult}^{\mathsf{v}}(A)$ and $\Gamma = \overrightarrow{\mathtt{x}} : H(p), \mathtt{f} : \neg(A \times \neg A), \overrightarrow{\mathtt{y} : \neg B}$. Then $\mathcal{R} \cup \approx$ is a normal form bisimulation. To see this, let $p' = \langle p, -_{\neg A}\rangle$ and observe that since $\Gamma \vdash^{\mathsf{n}} M_i \approx U_i\langle p(\overrightarrow{\mathtt{y}}), \mathtt{f}\rangle$ and U_i is a fixed point, there exist $\overrightarrow{W_i}, V_i$ such that

$$M_i \rightsquigarrow^* \mathtt{f}\langle p(\overrightarrow{W_i}), V_i\rangle = \mathtt{f}p'(\overrightarrow{W_i}, V_i) \overset{\mathtt{f}p'}{\rightsquigarrow} \overrightarrow{W_i}, V_i$$
$$\Gamma \vdash^{\mathsf{v}} \overrightarrow{W_i} \approx^{\mathsf{v}} \overrightarrow{\mathtt{x}} : H(p)$$
$$\Gamma \vdash^{\mathsf{v}} V_i \approx^{\mathsf{v}} \lambda \mathtt{v}.U_i\langle \mathtt{v}, \mathtt{f}\rangle : \neg A$$

for $i \in \{1,2\}$. Hence $\Gamma \vdash^{\mathsf{v}} \overrightarrow{W_1} \approx^{\mathsf{v}} \overrightarrow{W_2} : H(p)$ and $\Gamma \vdash^{\mathsf{v}} V_1 \ \mathcal{R}^{\mathsf{v}} \ V_2 : \neg A$. We conclude that $\mathcal{R} \cup \approx$ is a bisimulation and thus (6), since $\mathcal{R}$ relates (7).

4.2 Syntactic minimal invariance

Finally, we prove a syntactic version of the domain-theoretic minimal invariance property [31] for JWA. Our proof is greatly facilitated by the normal form bisimulation proof principle and is simpler than other syntactic minimal invariance proofs in the literature for typed and untyped calculi [32, 33].

For every closed type A, we define the type $A^\dagger \stackrel{\text{def}}{=} A \times \neg A$ and we will define a closed term $\vdash^{\mathsf{v}} h(A) : \neg A^\dagger$. More generally, to deal with recursive types, we define in Figure 4.2, by structural induction on A, open terms:

$$\Gamma \vdash^{\mathsf{v}} h(\Gamma \vdash A) : \neg A[\Gamma]^\dagger,$$

where

– $\Gamma = \overrightarrow{\mathtt{X} : \neg B^\dagger}$ (we take the liberty to use $\overrightarrow{\mathtt{X}}$ as term identifiers in Γ and $h(\Gamma \vdash A)$ and as type identifiers in A),
– the types in $\overrightarrow{B}$ are closed,
– A is an open type: $\overrightarrow{\mathtt{X}} \vdash A$ type, and
– $[\Gamma]$ denotes the type substitution $[\overrightarrow{B/\mathtt{X}}]$.

When A is closed, $h(A) \stackrel{\text{def}}{=} h(\vdash A)$.

$$
\begin{aligned}
h(\Gamma \vdash \neg A_0) &= \lambda\langle \mathtt{x}, \mathtt{k}\rangle.\mathtt{k}\, \lambda \mathtt{x}_0.h(\Gamma \vdash A_0)\langle \mathtt{x}_0, \mathtt{x}\rangle \\
h(\Gamma \vdash \textstyle\sum_{i\in I} A_i) &= \lambda\langle \mathtt{x}, \mathtt{k}\rangle.\mathtt{pm}\ \mathtt{x}\ \mathtt{as}\ \{\langle i, \mathtt{x}_i\rangle.h(\Gamma \vdash A_i)\langle \mathtt{x}_i, \lambda \mathtt{y}_i.\mathtt{k}\langle i, \mathtt{y}_i\rangle\rangle\}_{i\in I} \\
h(\Gamma \vdash 1) &= \lambda\langle \mathtt{x}, \mathtt{k}\rangle.\mathtt{k}\,\mathtt{x} \\
h(\Gamma \vdash A_1 \times A_2) &= \lambda\langle \mathtt{x}, \mathtt{k}\rangle.\mathtt{pm}\ \mathtt{x}\ \mathtt{as}\ \langle \mathtt{x}_1, \mathtt{x}_2\rangle. \\
&\qquad\qquad h(\Gamma \vdash A_1)\langle \mathtt{x}_1, \lambda \mathtt{y}_1.h(\Gamma \vdash A_2)\langle \mathtt{x}_2, \lambda \mathtt{y}_2.\mathtt{k}\langle \mathtt{y}_1, \mathtt{y}_2\rangle\rangle\rangle \\
h(\Gamma \vdash \mathtt{X}) &= \mathtt{X} \\
h(\Gamma \vdash \mu \mathtt{X}.A) &= \lambda \mathtt{u}.Y\langle \mathtt{u}, \lambda\langle \mathtt{v}, \mathtt{X}\rangle. \\
&\qquad\qquad \mathtt{pm}\ \mathtt{v}\ \mathtt{as}\ \langle \mathtt{x}, \mathtt{k}\rangle. \\
&\qquad\qquad \mathtt{pm}\ \mathtt{x}\ \mathtt{as}\ \mathtt{fold}\ \mathtt{x}_0. \\
&\qquad\qquad h(\Gamma, \mathtt{X} : \neg(\mu \mathtt{X}.A)^\dagger \vdash A)\langle \mathtt{x}_0, \lambda \mathtt{y}_0.\mathtt{k}(\mathtt{fold}\ \mathtt{y}_0)\rangle\rangle
\end{aligned}
$$

Fig. 5. Definition of $h(\Gamma \vdash A)$

Proposition 7. *$h(A[\Gamma]) = h(\Gamma \vdash A)[\overrightarrow{h(B)/\mathtt{X}}]$, if $\Gamma = \overrightarrow{\mathtt{X} : \neg B^\dagger}$.*

Proof. By structural induction on A.

In particular, if $\mu \mathtt{X}.A$ is closed,

$$h(A[\mu \mathtt{X}.A/\mathtt{X}]) = h(\mathtt{X} : \neg(\mu \mathtt{X}.A)^\dagger \vdash A)[h(\mu \mathtt{X}.A)/\mathtt{X}] \tag{8}$$

Lemma 1. *Let $g(V : \neg A) = \lambda \mathtt{y}.h(A)\langle \mathtt{y}, V\rangle$ and, by extension, let $g(\overrightarrow{V : \neg A})$ be the value sequence where $g(\overrightarrow{V : \neg A})_i = g(V_i : \neg A_i)$. Then, for all closed A and $p \in \mathsf{ult}^{\mathsf{v}}(A)$, $h(A)\langle p(\overrightarrow{V}), \mathtt{k}\rangle \rightsquigarrow^* \mathtt{k}(p(g(\overrightarrow{V : H(p)})))$.*

Proof. By structural induction on p. For illustration, we show the induction step for the case when $A = \mu \mathtt{X}.A_0$ and $p = \mathtt{fold}\ p_0$. Let $K = \lambda \mathtt{y}_0.\mathtt{k}(\mathtt{fold}\ \mathtt{y}_0)$.

$$
\begin{aligned}
h(A)\langle p(\overrightarrow{V}), \mathtt{k}\rangle &\rightsquigarrow^* h(\mathtt{X} : \neg A^\dagger \vdash A_0)[h(A)/\mathtt{X}]\langle p_0(\overrightarrow{V}), K\rangle \\
&= h(A_0[h(A)/\mathtt{X}])\langle p_0(\overrightarrow{V}), K\rangle, \text{ by (8)} \\
&\rightsquigarrow^* K(p_0(\overrightarrow{V : H(p_0)})), \text{ by the induction hypothesis} \\
&\rightsquigarrow \mathtt{k}(\mathtt{fold}\ p_0(\overrightarrow{V : H(p_0)})) = \mathtt{k}(p(\overrightarrow{V : H(p)}))
\end{aligned}
$$

Theorem 2 (Syntactic minimal invariance). *For all closed types A,* $\vdash^{\mathsf{v}} h(A) \approx^{\mathsf{v}} \lambda\langle \mathtt{x}, \mathtt{k}\rangle.\mathtt{k}\,\mathtt{x} : \neg A^{\dagger}$.

Proof. We need to show, for all $p \in \mathsf{ult}^{\mathsf{v}}(A)$,

$$\mathtt{k} : \neg A, \overrightarrow{\mathtt{x} : H(p)} \vdash^{\mathsf{n}} h(\Gamma \vdash A)\langle p(\overrightarrow{\mathtt{x}}), \mathtt{k}\rangle \approx (\lambda\langle \mathtt{x}, \mathtt{k}\rangle.\mathtt{k}\,\mathtt{x})\langle p(\overrightarrow{\mathtt{x}}), \mathtt{k}\rangle$$

By Lemma 1, the left hand side β-reduces to $\mathtt{k}(p(g(\overrightarrow{\mathtt{x} : H(p)})))$ and the right hand side β-reduces to $\mathtt{k}(p(\overrightarrow{\mathtt{x}}))$. It remains to show that

$$\mathtt{k} : \neg A, \overrightarrow{\mathtt{x} : H(p)} \vdash^{\mathsf{v}} g(\overrightarrow{\mathtt{x} : H(p)}) \approx^{\mathsf{v}} \overrightarrow{\mathtt{x}} : H(p)$$

This follows because the relation that relates, for all closed A and $p \in \mathsf{ult}^{\mathsf{v}}(A)$,

$$\overrightarrow{\mathtt{z} : \neg B}, \mathtt{x} : \neg A, \overrightarrow{\mathtt{y} : H(p)} \vdash^{\mathsf{n}} g(\mathtt{x} : \neg A)(p(\overrightarrow{\mathtt{y}})), \mathtt{x}(p(\overrightarrow{\mathtt{y}}))$$

is a bisimulation, which is immediate from the calculation (using Lemma 1)

$$g(\mathtt{x} : \neg A)(p(\overrightarrow{\mathtt{y}})) \rightsquigarrow h(A)\langle p(\overrightarrow{\mathtt{y}}), \mathtt{x}\rangle \rightsquigarrow^{*} \mathtt{x}(p(g(\overrightarrow{\mathtt{y} : H(p)})))$$

Acknowledgements We thank Radha Jagadeesan and Corin Pitcher for their comments on an earlier version of the paper and we thank the anonymous reviewers for pointing out some errors.

References

1. Sangiorgi, D.: The lazy lambda calculus in a concurrency scenario. Information and Computation **111**(1) (1994) 120–153
2. Jagadeesan, R., Pitcher, C., Riely, J.: Open bisimulation for aspects. In: Intl. Conf. on Aspect-Oriented Software Development, ACM (2007) 209–224
3. Lassen, S.B.: Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context. In: MFPS XV. Volume 20 of ENTCS., Elsevier (1999) 346–374
4. Lassen, S.B.: Eager normal form bisimulation. In: 20th LICS, IEEE (2005) 345–354
5. Lassen, S.B.: Normal form simulation for McCarty's amb. In: MFPS XXI. Volume 155 of ENTCS., Elsevier (2005) 445–465
6. Lassen, S.B.: Head normal form bisimulation for pairs and the $\lambda\mu$-calculus (extended abstract). In: 21st LICS, IEEE (2006) 297–306
7. Støvring, K., Lassen, S.B.: A complete, co-inductive syntactic theory of sequential control and state. In: 34th POPL, ACM (2007) 63–74
8. Abramsky, S.: The lazy λ-calculus. In Turner, D., ed.: Research Topics in Functional Programming. Addison-Wesley (1990) 65–116
9. Gordon, A.D.: Functional Programming and Input/Output. CUP (1994)
10. Levy, P.B.: Call-By-Push-Value. A Functional/Imperative Synthesis. Semantic Struct. in Computation. Springer (2004)
11. Levy, P.B.: Adjunction models for call-by-push-value with stacks. Theory and Applications of Categories **14**(5) (2005) 75–110
12. Merro, M., Biasi, C.: On the observational theory of the CPS calculus. In: MFPS XXII. Volume 158 of ENTCS., Elsevier (2006) 307–330

13. Hyland, J.M.E., Ong, C.H.L.: On full abstraction for PCF: I, II, and III. Information and Computation **163**(2) (2000) 285–408
14. Abramsky, S., McCusker, G.: Call-by-value games. In: 11th CSL. Volume 1414 of LNCS., Springer (1997) 1–17
15. Abramsky, S., Honda, K., McCusker, G.: A fully abstract game semantics for general references. In: 13th LICS, IEEE (1998) 334–344
16. Laird, J.: Full abstraction for functional languages with control. In: 12th LICS, IEEE (1997) 58–67
17. Ker, A.D., Nickau, H., Ong, C.H.L.: Innocent game models of untyped lambda-calculus. Theoretical Computer Science **272**(1-2) (2002) 247–292
18. Ker, A.D., Nickau, H., Ong, C.H.L.: Adapting innocent game models for the Böhm tree λ-theory. Theoretical Computer Science **308**(1-3) (2003) 333–366
19. Ong, C.H.L., Gianantonio, P.D.: Games characterizing Lévy-Longo trees. Theoretical Computer Science **312**(1) (2004) 121–142
20. Laird, J.: A categorical semantics of higher-order store. In: 9th Conference on Category Theory and Computer Science. Volume 69 of ENTCS., Elsevier (2003)
21. Curien, P.L., Herbelin, H.: Computing with abstract Böhm trees. In: Fuji International Symposium on Functional and Logic Programming. (1998) 20–39
22. Danos, V., Herbelin, H., Regnier, L.: Game semantics and abstract machines. In: 11th LICS, IEEE (1996) 394–405
23. Levy, P.B.: Infinite trace equivalence. In: MFPS XXI. Number 155 in ENTCS, Elsevier (2006) 467–496
24. Levy, P.B.: Game semantics using function inventories. Talk given at *Geometry of Computation 2006*, Marseille, 2006
25. Laird, J.: A fully abstract trace semantics for general references. In: 34th ICALP. Volume 4596 of LNCS., Springer (2007)
26. Hyland, J.M.E., Ong, C.H.L.: Pi-calculus, dialogue games and PCF. In: 7th FPCA, ACM (1995) 96–107
27. Fiore, M.P., Honda, K.: Recursive types in games: Axiomatics and process representation. In: 13th LICS, IEEE (1998) 345–356
28. Thielecke, H.: Categorical Structure of Continuation Passing Style. PhD thesis, University of Edinburgh (1997)
29. Thielecke, H.: Contrasting exceptions and continuations. Unpublished (October 2001)
30. Barendregt, H.P.: The Lambda Calculus: Its Syntax and Semantics. Revised edn. North-Holland (1984)
31. Pitts, A.M.: Relational properties of domains. Information and Computation **127** (1996) 66–90
32. Birkedal, L., Harper, R.: Operational interpretations of recursive types in an operational setting (summary). In: TACS. Volume 1281 of LNCS., Springer (1997)
33. Lassen, S.B.: Relational reasoning about contexts. In Gordon, A.D., Pitts, A.M., eds.: Higher Order Operational Techniques in Semantics. CUP (1998) 91–135