A Complete, Co-Inductive Syntactic Theory of Sequential Control and State

Kristian Støvring
BRICS, Dept. of Comp. Science, University of Aarhus
kss@brics.dk

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

Abstract

We present a new co-inductive syntactic theory, *eager normal form bisimilarity*, for the untyped call-by-value lambda calculus extended with continuations and mutable references.

We demonstrate that the associated bisimulation proof principle is easy to use and that it is a powerful tool for proving equivalences between recursive imperative higher-order programs.

The theory is modular in the sense that eager normal form bisimilarity for each of the calculi extended with continuations and/or mutable references is a fully abstract extension of eager normal form bisimilarity for its sub-calculi. For each calculus, we prove that eager normal form bisimilarity is a congruence and is sound with respect to contextual equivalence. Furthermore, for the calculus with both continuations and mutable references, we show that eager normal form bisimilarity is complete: it coincides with contextual equivalence.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features—Control structures; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Operational semantics; F.4.1 [*Mathematical Logic and Formal Languages*]: Mathematical Logic—Lambda calculus and related systems

General Terms Languages, Theory

Keywords Bisimulation, Continuations, Mutable References

1. Introduction

Program equivalence is a fundamental concept in programming language semantics, and new and better frameworks and techniques for reasoning about program equivalence are continually being developed. Nonetheless, there are still no general and easy to use methods that capture the features and subtleties of actual programs in languages that combine general recursion, higher-order functions and objects, mutable state, and non-local control flow.

Denotational semantics and domain theory cover many programming language features but straightforward models fail to capture certain important aspects of program equivalence, especially concerning mutable state. The solutions to these "full abstraction" problems, including game semantics, are complex.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

POPL'07 January 17–19, 2007, Nice, France.
Copyright © 2007 ACM 1-59593-575-4/07/0001... $5.00.

Syntactic reduction calculi and equational theories are easy to use but they exclude many important program equivalences.

The broadest notion of program equivalence is Morris-style contextual equivalence which equates two terms if they behave the same in all program contexts. The quantification over all program contexts makes it impractical to use the definition directly to prove programs contextually equivalent.

Syntactic methods based on operational semantics—context lemmas, applicative bisimulation, and operationally-based logical relations—generally incur modest "mathematical overhead" and are easy to use for certain classes of program equivalences. For instance, applicative bisimulation is very useful for proving the equivalence of programs that output infinite data structures. However, all these proof principles are weak for program equivalences involving general higher-order functions because, somewhat like the definition of contextual equivalence, they involve universal quantifications over all continuations, stores, and/or function arguments.

For example, fixed-point combinators are higher-order functions that make essential use of higher-order arguments. What does it take to prove the equivalence of two different fixed-point combinators? A proof obligation that involves a universal quantification over all possible arguments to the fixed-point combinators is about as difficult as proving that the fixed-point combinators are contextually equivalent from first principles.

This example is easily solved using a different class of syntactic theories which originate from the theories of Böhm tree equivalence and Lévy–Longo tree equivalence. They can be presented as bisimulation theories, called *normal form bisimulation* (originally introduced by Sangiorgi under the name "open applicative bisimulation"), without explicit reference to trees. Normal form bisimulation is based on symbolic evaluation of open terms to normal forms. It does not involve any universal quantification over function arguments and is therefore, in some respects, a more powerful proof principle for proving equivalences between recursive higher-order functions than other operationally-based syntactic methods. However, normal form bisimulation has only been developed for state-less λ-calculi and is, in general, not fully abstract.

In this article we address these shortcomings by extending eager normal form bisimulation, a variant of normal form bisimulation for the call-by-value λ-calculus. We present new syntactic bisimulation theories for the untyped call-by-value λ-calculus extended with continuations and mutable references.

1. The theories all extend eager normal form (enf) bisimulation for the pure call-by-value λ-calculus [19].
2. The extension with continuations, namely an untyped call-by-value version of Parigot's $\lambda\mu$-calculus [26], is based on the second author's normal form bisimulation theory for the untyped $\lambda\mu$-calculus [21].

3. The extension with mutable references, which we call the $\lambda\rho$-calculus (essentially Felleisen and Hieb's λ-calculus with state [8]; their "ρ-application" is a primitive in our calculus hence we name it "$\lambda\rho$"), is based on bisimulations as sets of relations. This idea of "relation-sets bisimulation" is adapted from bisimulation theories for imperative calculi [13, 16] and existential types [32].
4. Finally, we extend the theories to a combined $\lambda\mu\rho$-calculus.

The resulting bisimulation proof principle for proving semantical equivalences between terms inherits the best properties of normal form bisimulation and relation-sets bisimulation, namely

- like other kinds of normal form bisimulation, the enf bisimulation proof obligations for continuations and mutable references require no universal quantifications over function arguments or continuations or stores, and
- the relation-set structure represents the "possible worlds" necessary to capture the behaviour of mutable references.

We demonstrate the power and ease of use of the resulting enf bisimulation proof principle for continuations and mutable references by proving the correctness of Friedman and Haynes's encoding of call/cc in terms of "one-shot" continuations [9]. Despite the subtlety of their encoding and the mix of higher-order functions, first-class continuations, and mutable references, the bisimulation proof is remarkably straightforward, as we hope the reader will appreciate.

The enf bisimulation theories for the pure λ-calculus and the extensions with continuations and/or mutable references are modular: enf bisimilarity for each of the extended calculi is a fully abstract extension of enf bisimilarity for its sub-calculi. This is similar to the relationship between Felleisen and Hieb's syntactic theories for control and state [8] but contrasts the situation for contextual equivalence because each language extension makes contextual equivalence more discriminative on terms of the sub-calculi.

One of the main technical contributions of the work behind this article is a proof that enf bisimilarity for the calculus extended with continuations and/or mutable references is a congruence. As an immediate consequence of congruence, enf bisimilarity is included in contextual equivalence for each calculus. For the pure λ-calculus as well as the two extensions with only continuations and only mutable references, enf bisimilarity is strictly smaller than contextual equivalence, that is, enf bisimulation is a sound but incomplete method for proving contextual equivalence. However, for the full calculus with both continuations and mutable references, we prove that enf bisimilarity is fully abstract in the sense that it coincides with contextual equivalence.

In summary, we present a complete, co-inductive syntactic theory for a calculus with higher-order functions, continuations, and mutable references, and we demonstrate the power and ease-of-use of the bisimulation proof method for proving equivalences between recursive programs.

Our results provide further illustration of the promise of normal form bisimulation as a basis for syntactic theories and proof principles, demonstrated by earlier results for other pure and extended λ-calculi in the literature (Sangiorgi [31] and Lassen [18, 20, 21]). However, we note one caveat: Although our theory for the combined $\lambda\mu\rho$-calculus captures key functional and imperative aspects of the programming language Scheme, it lacks constants such as nil, cons, numerals, and arithmetic operators. These constants need to be encoded in our calculus, e.g., using standard λ-calculus encodings [4], but such encodings are in general not faithful to the constants' equational properties. For instance, addition of values should be commutative, up to contextual equivalence—that is, the representations of the Scheme terms (lambda $(x\ y)$ $(+\ x\ y)$) and (lambda $(x\ y)$ $(+\ y\ x)$) in the $\lambda\mu\rho$-calculus should be equivalent—but this fails for encodings of arithmetic in the $\lambda\mu\rho$-calculus, hence the resulting proof principles are only sound, not complete. There does not seem to be a satisfactory direct definition of normal form bisimulation (or Böhm-tree equivalence) for untyped calculi with constants. In future joint work with Paul Blain Levy we plan, instead, to address this shortcoming in extensions of normal form bisimulation to typed calculi with recursive types. This work is related to recent game models by Levy [22].

1.1 Related work

There exists a large body of work on syntactic theories and semantic models (domains and games) for λ-calculi with continuations and mutable references. We only survey a few works on syntactic theories most closely related to the results in this article.

As mentioned in the introduction, our results build directly on recent work on normal form bisimulation for call-by-value [19] and the $\lambda\mu$-calculus [21] and on relation-sets bisimulation for existential types [32] and untyped imperative λ-calculus [13, 16].

One particular inspiration for the work presented in this article is the seminal research by Felleisen *et al.* on syntactic theories for sequential control and state [8]. The calculi in *op.cit.* are enriched with constants and δ-reduction but otherwise the state calculus is essentially what we call the $\lambda\rho$-calculus in this article. The control calculus differs from the $\lambda\mu$-calculus but they are comparable. (Their relationship is analyzed by de Groote [12] and by Ariola and Herbelin [3]. We found that it was easiest to define eager reduction on open terms, enfs, and enf bisimilarity for the $\lambda\mu$-calculus.) The syntactic theories of successive λ-calculus extensions by Felleisen et al. [8] are modular (conservative extensions), like our syntactic theories. An important difference is that the syntactic theories in *op.cit.* are *inductive* in the sense that all equations are derived inductively from equational axioms and inference rules, whereas our bisimulation theories are *co-inductive* and therefore equate many more programs.

Another body of related work is Mason and Talcott's CIU ("closed instantiations of uses") characterizations of contextual equivalence for functional languages with mutable references and continuations [23, 33]. (The context lemmas for the $\lambda\mu$-calculus by Bierman [5] and by David and Py [6] are essentially CIU characterizations.) The CIU equivalences are complete syntactic theories but the resulting proof methods are in many cases weaker than normal form bisimulation.

Most co-inductive syntactic programming language theories in the literature are variants and extensions of Abramsky's applicative bisimulation [1]. However, there are no fully abstract applicative bisimulation theories for general λ-calculi with continuations and/or mutable references.

Ritter and Pitts [30] define a form of applicative bisimilarity for a functional language with mutable references. It is sound but not complete. In fact, it does not equate many of the well-known, subtle contextual equivalences between programs with state [25].

Wand and Sullivan [34] define a CPS language with mutable references and show that applicative bisimilarity is both sound and complete. They use the CPS language as a semantic meta-language and CPS translate a source language with state into the CPS language. But they do not give an independent characterization of the induced syntactic theory on source terms via the CPS transform.

Koutavas and Wand's relation-sets bisimulation theory [13] is complete for a general "direct-style" imperative calculus. However, it involves a universal quantification over closed function arguments, unlike our normal form bisimulation theories.

Merro and Biasi [24] present a complete bisimulation theory for a CPS calculus. It can be viewed as a kind of applicative bisimulation, presented as a labelled transition system in the style of Gordon [10], and also leads to a context lemma.

Pitts and Stark [28, 29] develop syntactic theories based on operationally-based logical relations that address many of the subtleties of contextual equivalences between programs with mutable references. The relation-sets bisimulation theories for mutable state, in general, are alternative approaches with a very different meta-theory. For logical relations the key proof obligation is existence, whereas the key proof obligation for the bisimulation theories is congruence.

Finally, we note that the modularity of the enf bisimilarity theories for control and state resembles the modularity of game semantics for control and state [2, 14].

2. Eager normal form bisimulation

Let us briefly reintroduce the definition of enf bisimulation for the pure call-by-value λ-calculus [19]. Consider a variant of the call-by-value λ-calculus in which computations must be explicitly sequenced by means of a let-construct:

VARIABLES x, y, z
VALUES $v ::= x \mid \lambda x.\, t$
TERMS $t ::= v \mid \mathsf{let}\, x{=}t_1 \,\mathsf{in}\, t_2 \mid v_1\, v_2$

We identify terms up to renaming of bound variables.

Reduction is defined by means of evaluation contexts:

EVALUATION CONTEXTS $E ::= [\,] \mid E[\mathsf{let}\, x{=}[\,] \,\mathsf{in}\, t]$
EAGER NORMAL FORMS (ENFS) $e ::= v \mid E[x\, v]$

(R1) $E[\mathsf{let}\, x{=}v \,\mathsf{in}\, t] \mapsto E[t[v/x]]$
(R2) $E[(\lambda x.\, t)\, v] \mapsto E[t[v/x]]$

The reflexive-transitive closure of the reduction relation $\mapsto$ is written $\mapsto^*$. For every term t, there are two possibilities: either t *diverges* in the sense that there is an infinite reduction sequence starting from t, or else t *converges* in the sense that $t \mapsto^* e$ for some (unique) eager normal form e. The notation $t \mapsto^\omega$ means that t diverges. Eager normal forms are truly normal forms with respect to reduction: they do not reduce to anything.

For a syntactic phrase ϕ, let $\mathrm{FV}(\phi)$ denote the set of free variables of ϕ (the formal definitions are omitted).

Definition 1. A binary relation S on terms is an *enf bisimulation* if $S \subseteq B(S)$, where

$$\begin{aligned}
B(S) &= \{(t,t') \mid \text{either } t \mapsto^\omega \text{ and } t' \mapsto^\omega, \\
&\qquad\quad \text{or } t \mapsto^* e \text{ and } t' \mapsto^* e' \text{ where } (e,e') \in M(S)\} \\
M(S) &= \{(v,v') \mid (v,v') \in V(S)\} \\
&\quad \cup \{(E[x\,v], E'[x\,v']) \mid (E,E') \in K(S)\ \& \\
&\qquad\qquad (v,v') \in V(S)\} \\
V(S) &= \{(x,x)\} \cup \{(v,v') \mid \exists y \notin \mathrm{FV}(v) \cup \mathrm{FV}(v'). \\
&\qquad\qquad (v \star y, v' \star y) \in S\} \\
K(S) &= \{([\,],[\,])\} \cup \{(E,E') \mid \exists y \notin \mathrm{FV}(E) \cup \mathrm{FV}(E'). \\
&\qquad\qquad (E \star y, E' \star y) \in S\}
\end{aligned}$$

with $x \star y = x\,y$, $(\lambda y.\, t) \star x = t[x/y]$, $[\,] \star y = y$, and $E[\mathsf{let}\, y{=}[\,] \,\mathsf{in}\, t] \star x = E[t[x/y]]$.

The intuition behind enf bisimulation is that two related open terms either (1) both diverge, or (2) reduce to matching eager normal forms whose components are again related. As an example, define the Curry call-by-value fixed-point combinator $\mathsf{Y_v}$:

$$\begin{aligned}
\Psi[f] &= \lambda g.\, f\,(\lambda x.\, \mathsf{let}\, z{=}g\,g \,\mathsf{in}\, z\,x) \\
\mathsf{Y_v} &= \lambda f.\, \Psi[f]\, \Psi[f]
\end{aligned}$$

and the Turing call-by-value fixed-point combinator Θ_v:

$$\begin{aligned}
\Xi &= \lambda g.\lambda f.f\,(\lambda x.\, \mathsf{let}\, z_1{=}g\,g \,\mathsf{in}\, \mathsf{let}\, z_2{=}z_1\,f \,\mathsf{in}\, z_2\,x) \\
\Theta_\mathsf{v} &= \Xi\,\Xi.
\end{aligned}$$

These two fixed-point combinators are enf bisimilar, i.e., there exists an enf bisimulation S such that $(\mathsf{Y_v}, \Theta_\mathsf{v}) \in S$ [19]. We invite the reader to try to prove this equivalence by constructing such an S: one starts with the singleton $\{(\mathsf{Y_v}, \Theta_\mathsf{v})\}$ and then iteratively adds pairs in order to satisfy the definition of an enf bisimulation above. (In Section 5, a similar, but more complicated, equivalence between $\mathsf{Y_v}$ and a store-based fixed-point combinator is shown.)

Remark. The following construction, derived from the Turing call-by-value fixed-point combinator, is convenient for defining functions by recursion: For all values v, v_1, and v_2, define

$$\begin{aligned}
\mathsf{D}[v_1, v_2] &= \mathsf{let}\, z_1{=}\Theta_\mathsf{v} \,\mathsf{in}\, \mathsf{let}\, z_2{=}z_1\,v_1 \,\mathsf{in}\, z_2\,v_2 \\
\mathsf{fix}[v] &= \lambda x.\mathsf{D}[v, x]
\end{aligned}$$

Then $\mathsf{fix}[v]\,x \mapsto^* \mathsf{let}\, z{=}v\,\mathsf{fix}[v] \,\mathsf{in}\, z\,x$.

Contextual equivalence is defined in the standard way. Informally, two terms t and t' are contextually equivalent if for every many-holed term context $C[\,]$ such that $C[t]$ and $C[t']$ are closed terms, $C[t]$ converges if and only if $C[t']$ converges.

Theorem 2 ([19]). *If $(t,t') \in S$ for some enf bisimulation S, then t and t' are contextually equivalent.*

Remark. The definition of an enf bisimulation is slightly different from the one in the original presentation [19]. In particular, the variant defined here is equivalent to what is called an enf bisimulation up to η in the original presentation.

In the sequel we omit the "enf" qualifier for bisimulations and instead qualify them by calculi. We will refer to the bisimulations for the pure λ-calculus in Definition 1 as "λ-bisimulations".

3. The $\lambda\mu$-calculus

We now extend enf bisimulation to the $\lambda\mu$-calculus. This extension is new, but based on head normal form bisimulation for the $\lambda\mu$-calculus [21].

VARIABLES x, y, z
NAMES a, b
VALUES $v ::= x \mid \lambda x.\, t$
NAMED TERMS $nt ::= [a]t$
TERMS $t ::= v \mid \mathsf{let}\, x{=}t_1 \,\mathsf{in}\, t_2 \mid v_1\, v_2 \mid \mu a.\, nt$

We identify syntactic phrases up to renaming of bound variables and names. For a syntactic phrase ϕ, let $\mathrm{FN}(\phi)$ denote the set of free names of ϕ.

Names in the $\lambda\mu$-calculus represent continuations. Names are not first-class, but we will represent a name a as the first-class value $\hat{a} = \lambda x.\, \mu b.\, [a]x$. The familiar *call/cc* control operator can be encoded in the $\lambda\mu$-calculus as

$$\mathsf{call/cc} = \lambda f.\, \mu a.\, [a] f\, \hat{a}.$$

The operational semantics of the $\lambda\mu$-calculus is defined by a reduction relation on named terms:

NAMED EVAL. CONTEXTS $NE ::= [a][\,] \mid NE[\mathsf{let}\, x{=}[\,] \,\mathsf{in}\, t]$
NAMED ENFS $ne ::= [a]v \mid NE[x\, v]$

(Rμ1) $NE[\mathsf{let}\, x{=}v \,\mathsf{in}\, t] \mapsto NE[t[v/x]]$
(Rμ2) $NE[(\lambda x.\, t)\, v] \mapsto NE[t[v/x]]$
(Rμ3) $NE[\mu a.\, nt] \mapsto nt[NE/a]$

Here $\phi[NE/a]$ denotes capture-avoiding substitution of named evaluation contexts for names: for example, if $b \notin \mathrm{FN}(NE)$, then $(\mu b.\, [a]t)[NE/a] = \mu b.\, NE[t]$.

Definition 3. A binary relation S on named $\lambda\mu$-terms is a $\lambda\mu$-*bisimulation* if $S \subseteq B_\mu(S)$, where

$$\begin{aligned}
B_\mu(S) &= \{(nt, nt') \mid \text{either } nt \mapsto^\omega \text{ and } nt' \mapsto^\omega, \\
&\qquad \text{or } nt \mapsto^* ne \text{ and } nt' \mapsto^* ne' \\
&\qquad \text{where } (ne, ne') \in M_\mu(S)\} \\
M_\mu(S) &= \{([a]v, [a]v') \mid (v, v') \in V_\mu(S)\} \\
&\quad \cup \{(NE[x\,v], NE'[x\,v']) \mid (NE, NE') \in K_\mu(S) \;\&\; (v, v') \in V_\mu(S)\} \\
V_\mu(S) &= \{(x, x)\} \\
&\quad \cup \{(v, v') \mid \exists y \notin \mathrm{FV}(v) \cup \mathrm{FV}(v').\ (v \star y, v' \star y) \in T_\mu(S)\} \\
K_\mu(S) &= \{([a][\,], [a][\,])\} \\
&\quad \cup \{(NE, NE') \mid \exists y \notin \mathrm{FV}(NE) \cup \mathrm{FV}(NE').\ (NE \star y, NE' \star y) \in T_\mu(S)\} \\
T_\mu(S) &= \{(t, t') \mid \exists a \notin \mathrm{FN}(t) \cup \mathrm{FN}(t').\ ([a]t, [a]t') \in S\}
\end{aligned}$$

with $[a][\,] \star y = [a]y$ and $NE[\mathsf{let}\, x{=}[\,]\ \mathsf{in}\ t] \star y = NE[t[y/x]]$.

Definition 4. Say that t and t' are $\lambda\mu$-*bisimilar*, written $t \approx_\mu t'$, if there exists a $\lambda\mu$-bisimulation S such that $(t, t') \in T_\mu(S)$.

We show in Section 10 that $\lambda\mu$-bisimilar terms are contextually equivalent.

Recall that $\hat{a} = \lambda x.\, \mu b.\, [a]x$. To illustrate $\lambda\mu$-bisimilarity we define the term $\psi = \mathsf{fix}[\mathsf{P}]$, where

$$\mathsf{P} = \lambda f.\, \lambda x.\, \mu a.\, [a]\, \mathsf{let}\, y{=}x\,\hat{a}\ \mathsf{in}\ f\,y.$$

The term ψ takes a function x as argument and applies x to successive arguments

$$x\,\hat{a_1}\,\hat{a_2}\ldots$$

until x applies one of the $\hat{a_i}$ to an argument v, in which case v is returned as the result of $\psi\,x$. On the other hand, $\psi\,x$ diverges if x never applies any of its arguments, e.g., if $x = \lambda y.\,\Omega$ or $x = \mathsf{fix}[\lambda f.\,\lambda y.\,f]$.

Remark. A term with the behavior of ψ cannot be expressed in the pure call-by-value λ-calculus. To see this, consider the two functions

$$v = \lambda y.\ \mathsf{let}\, z{=}y\,y\ \mathsf{in}\ \Omega \quad \text{and} \quad v' = \lambda y.\,\Omega.$$

where $\Omega = (\lambda x.x\,x)(\lambda x.x\,x)$. They are contextually equivalent in the pure call-by-value λ-calculus. (This can be established using the operational extensionality property of the pure call-by-value λ-calculus [7, 27], because the term $\mathsf{let}\, z{=}v_0\,v_0\ \mathsf{in}\ \Omega$ diverges if v_0 is any closed pure value.) But ψ can tell them apart: $\psi\,v$ converges while $\psi\,v'$ diverges.

A potential optimization of ψ is the following variant ψ' which returns straight to its final "return address" when x applies an argument (rather than returning from all the recursive invocations of the recursive function): $\psi' = \lambda x.\,\mu a.\,[a]\,\mathsf{fix}[\mathsf{P}']\,x$, where

$$\mathsf{P}' = \lambda f.\,\lambda x.\ \mathsf{let}\, y{=}x\,\hat{a}\ \mathsf{in}\ f\,y$$

The optimization is correct up to enf bisimilarity, that is, $\psi \approx_\mu \psi'$, because

$$\begin{aligned}
S = \{&([a]\psi, [a]\psi'),\ ([a]\mathsf{D}[\mathsf{P}, x], [a]\mu a.\,[a]\mathsf{fix}[\mathsf{P}']\,x), \\
&([b]\mu b.\,[a]x, \mu b.\,[a]x),\ ([a]\mathsf{fix}[\mathsf{P}]\,y, [a]\mathsf{fix}[\mathsf{P}']\,y)\}
\end{aligned}$$

is a $\lambda\mu$-bisimulation.

4. The $\lambda\rho$-calculus

The $\lambda\rho$-calculus is obtained from the pure call-by-value λ-calculus by adding constructs for allocating a number of new reference cells, for storing a value in a reference cell, and for fetching the value from a reference cell.

VARIABLES x, y, z
REFERENCES $\imath, \jmath$
VALUES $v ::= x \mid \lambda x.\, t$
TERMS $t ::= v \mid \mathsf{let}\, x{=}t_1\ \mathsf{in}\ t_2 \mid v_1\,v_2 \mid \rho s.\, t \mid \imath{:=}v; t \mid\ !\imath$
STORES $s ::= \{\imath_1{:=}v_1, \ldots, \imath_n{:=}v_n\}$ $\quad(\imath_1, \ldots, \imath_n$ are distinct)

Stores are identified up to reordering, and therefore a store can be considered as a finite map from references to values. Terms are identified up to renaming of bound variables and references: in the term $\rho s.\, t$, the references in the domain of s are considered bound in the range of s and in t. For a syntactic phrase ϕ, let $\mathrm{FR}(\phi)$ be the set of references occurring free in ϕ. A syntactic phrase is *reference-closed* if it contains no free references. Write $dom(s)$ for the domain of the store s. If s and s' have disjoint domains, $s{\cdot}s'$ denotes their disjoint union. If $s = \{\imath{:=}v\}{\cdot}s'$, let $s(\imath) = v$ and $s[\imath{:=}v'] = \{\imath{:=}v'\}{\cdot}s'$.

Reduction is defined on *configurations*, which are pairs (s, t) of stores and terms such that $\mathrm{FR}(t) \subseteq dom(s)$. (Configurations are not identified up to renaming of the domains of the stores, hence a configuration (s, t) should not be thought of as a term $\rho s.\, t$.)

EVALUATION CONTEXTS $E ::= [\,] \mid E[\mathsf{let}\, x{=}[\,]\ \mathsf{in}\ t]$
EAGER NORMAL FORMS (ENFS) $e ::= v \mid E[x\,v]$

(Rρ1) $(s, E[\mathsf{let}\, x{=}v\ \mathsf{in}\ t]) \mapsto (s, E[t[v/x]])$
(Rρ2) $(s, E[(\lambda x.\, t)\, v]) \mapsto (s, E[t[v/x]])$
(Rρ3) $(s, E[\rho s'.\, t]) \mapsto (s{\cdot}s', E[t])$,
if $(dom(s) \cup \mathrm{FR}(s) \cup \mathrm{FR}(E)) \cap dom(s') = \emptyset$
(Rρ4) $(s, E[\imath{:=}v; t]) \mapsto (s[\imath{:=}v], E[t])$ $\quad$ if $\imath \in dom(s)$
(Rρ5) $(s, E[!\imath]) \mapsto (s, E[s(\imath)])$ $\quad$ if $\imath \in dom(s)$

Eager normal form bisimulation for the $\lambda\rho$-calculus is based on the relation-sets bisimulation idea [13, 16, 32]. Briefly, instead of defining a bisimulation as a single binary relation on terms, one defines a bisimulation as a *set* of such relations, each associated with a "world": here, a pair of stores. The requirement is that if two terms are related in a certain world, then the eager normal forms (if any) of these two terms are related in a "future world" where the two stores may have changed. Moreover, everything that was related in the old world must still be related in the new world.

Now for the formal definitions. Let X, Y, Z range over finite sets of variables and let J range over finite sets of references. We write $X{\cdot}Y$ for the disjoint union of X and Y. When the meaning is clear from the context, we write a singleton set $\{x\}$ as just x. We use the same notational conventions for finite sets of references.

Notation $X, J \vdash \phi, \phi', \ldots$ means the syntactic phrases $\phi, \phi', \ldots$ have free variables in X and free references in J. We omit X and/or J on the left of $\vdash$ if it is empty.

Let R range over sets of triples $(X|t, t')$, more specifically subsets of $Rel(Y, J, J')$ for some Y, J and J', where

$$\begin{aligned}
&Rel(Y, J, J') = \\
&\quad \{(X|t, t') \mid X \cap Y = \emptyset \;\&\; X{\cdot}Y, J \vdash t \;\&\; X{\cdot}Y, J' \vdash t'\}
\end{aligned}$$

We identify triples that differ only up to renaming of the variables from the first component X: in the triple $(X|t, t')$, the variables in X are considered bound in t and t'. A triple $(\emptyset|t, t')$ where the first component is empty is also written $(|t, t')$.

A *term relation tuple* is a quadruple $(X|s, s', R)$ where $X \vdash s, s'$ and $R \subseteq Rel(X, dom(s), dom(s'))$. We identify term relation tuples that differ only up to renaming of the variables from the first component X and up to renaming of references. Let Q range over *term relation sets*, that is, sets of term relation tuples.

Definition 5. Q is a *$\lambda\rho$-bisimulation* iff $Q \subseteq B_\rho(Q)$, where

$$\begin{aligned}
B_\rho(Q) = \{&(X|s_0, s_0', R_0) \mid \\
&\text{for all } (Y|t,t') \in R_0, \text{ either} \\
&(s_0,t) \mapsto^\omega \ \& \ (s_0',t') \mapsto^\omega, \text{ or} \\
&\exists s_1, s_1', e, e', R_1 \supseteq R_0, X_1 \supseteq X{\cdot}Y. \\
&\quad (s_0,t) \mapsto^* (s_1,e) \ \& \ (s_0',t') \mapsto^* (s_1',e') \ \& \\
&\quad (e,e') \in M_\rho(R_1) \ \& \ (X_1|s_1,s_1',R_1) \in Q\} \\
M_\rho(R) = \{&(v,v'), (E[x\,v], E'[x\,v']) \mid \\
&(v,v') \in V_\rho(R) \ \& \ (E,E') \in K_\rho(R)\} \\
V_\rho(R) = \{&(x,x)\} \\
\cup \{&(v,v') \mid \exists y \notin \text{FV}(v) \cup \text{FV}(v'). \\
&\quad (y|v \star y, v' \star y) \in R\} \\
K_\rho(R) = \{&([\,],[\,])\} \\
\cup \{&(E,E') \mid \exists y \notin \text{FV}(E) \cup \text{FV}(E'). \\
&\quad (y|E \star y, E' \star y) \in R\}
\end{aligned}$$

Definition 6. Reference-closed $\lambda\rho$-terms t and t' are *$\lambda\rho$-bisimilar*, written $t \approx_\rho t'$, iff there exists a $\lambda\rho$-bisimulation Q which contains a quadruple $(X|\{\}, \{\}, R)$ with $(|t,t') \in R$.

We show in Section 9 that $\lambda\rho$-bisimilarity is a congruence. Therefore, as explained in Section 10, $\lambda\rho$-bisimilar terms are contextually equivalent.

5. Example: imperative fixed-point combinator

It is well-known that a store that may contain functional values can be used to define functions by recursion. Abbreviate

$$\Pi[f,\imath] = \lambda x.\ \mathsf{let}\, z_1{=}!\imath \ \mathsf{in}\ \mathsf{let}\, z_2{=}f\,z_1 \ \mathsf{in}\ z_2\,x$$

and consider the term:

$$\mathsf{Y}_\rho = \lambda f.\, \rho\{\imath{:=}\Pi[f,\imath]\}.\, f\,\Pi[f,\imath].$$

Y_ρ can be used to define functions by recursion in the $\lambda\rho$-calculus. The technique of defining recursive functions by means of a "circular store" is due to Landin [15].

We now show that the fixed-point combinator Y_ρ is $\lambda\rho$-bisimilar to the Curry call-by-value fixed-point combinator Y_v (defined in Section 2 above). This equivalence can be shown directly from the definition of a $\lambda\rho$-bisimulation, but it is more convenient to apply the following general lemma:

Lemma 7. *Define $\hat\rho s.\,t = \rho s.\,t$ for $s \neq \{\}$, and $\hat\rho\{\}.\,t = t$. Assume that there exists a $\lambda\rho$-bisimulation containing a tuple $(X|s,s',R)$ where $(|t,t') \in R$, and let $x_1,\ldots,x_n \in X$. Then $\lambda x_1 \ldots \lambda x_n.\,\hat\rho s.\,t \approx_\rho \lambda x_1 \ldots \lambda x_n.\,\hat\rho s'.\,t'$.*

The lemma follows from Corollary 36 in Section 9.

Proposition 8. $\mathsf{Y}_\rho \approx_\rho \mathsf{Y}_\mathsf{v}$.

Proof. By definition, $\mathsf{Y}_\rho = \lambda f.\,\rho\{\imath{:=}\Pi[f,\imath]\}.\,f\,\Pi[f,\imath]$ and $\mathsf{Y}_\mathsf{v} = \lambda f.\,\Psi[f]\,\Psi[f]$. The proof therefore consists of constructing a $\lambda\rho$-bisimulation Q containing a tuple $(\{f\}|\{\imath{:=}\Pi[f,\imath]\}, \{\}, R)$ where $(|f\,\Pi[f,\imath],\ \Psi[f]\,\Psi[f]) \in R$, and then using Lemma 7.

Instead of specifying Q right away, we show how one would in practice construct Q: by starting from the two configurations $(\{\imath{:=}\Pi[f,\imath]\}, f\,\Pi[f,\imath])$ and $(\{\}, \Psi[f]\,\Psi[f])$ and iteratively adding tuples in order to satisfy the conditions in the definition of a $\lambda\rho$-bisimulation. In that way, the main part of the equivalence proof consists in a number of calculations of reduction sequences.

Abbreviate $D[f] = \lambda x.\ \mathsf{let}\, z{=}\Psi[f]\,\Psi[f] \ \mathsf{in}\ z\,x$. Now calculate:

$$\begin{aligned}
(\{\imath{:=}\Pi[f,\imath]\}, f\,\Pi[f,\imath]) &\mapsto^* (\{\imath{:=}\Pi[f,\imath]\}, f\,\Pi[f,\imath]) \\
(\{\}, \Psi[f]\,\Psi[f]) &\mapsto^* (\{\}, f\,D[f]).
\end{aligned}$$

The two resulting eager normal forms are $f\,\Pi[f,\imath]$ and $f\,D[f]$. The variables in function position match (both are f), so consider the arguments, $\Pi[f,\imath]$ and $D[f]$. Since

$$\Pi[f,\imath] = \lambda x.\ \mathsf{let}\, z_1{=}!\imath \ \mathsf{in}\ \mathsf{let}\, z_2{=}f\,z_1 \ \mathsf{in}\ z_2\,x$$

and

$$D[f] = \lambda x.\ \mathsf{let}\, z{=}\Psi[f]\,\Psi[f] \ \mathsf{in}\ z\,x,$$

the definition of a $\lambda\rho$-bisimulation indicates that one should continue by reducing the bodies of these two λ-abstractions:

$$\begin{aligned}
&(\{\imath{:=}\Pi[f,\imath]\}, \mathsf{let}\, z_1{=}!\imath \ \mathsf{in}\ \mathsf{let}\, z_2{=}f\,z_1 \ \mathsf{in}\ z_2\,x) \\
&\quad \mapsto^* (\{\imath{:=}\Pi[f,\imath]\}, \mathsf{let}\, z_2{=}f\,\Pi[f,\imath] \ \mathsf{in}\ z_2\,x)
\end{aligned}$$

and

$$\begin{aligned}
(\{\}, \mathsf{let}\, z{=}\Psi[f]\,\Psi[f] \ \mathsf{in}\ z\,x) &\mapsto^* (\{\}, \mathsf{let}\, z{=}f\,D[f] \ \mathsf{in}\ z\,x) \\
&= (\{\}, \mathsf{let}\, z_2{=}f\,D[f] \ \mathsf{in}\ z_2\,x)
\end{aligned}$$

The resulting eager normal forms are

$$\mathsf{let}\, z_2{=}f\,\Pi[f,\imath] \ \mathsf{in}\ z_2\,x \quad \text{and} \quad \mathsf{let}\, z_2{=}f\,D[f] \ \mathsf{in}\ z_2\,x.$$

Again, the variables in function position match (both are f), and the evaluation contexts are identical (both are $\mathsf{let}\, z_2{=}[\,] \ \mathsf{in}\ z_2\,x$). The function arguments, $\Pi[f,i]$ and $D[f]$, are λ-abstractions, and therefore one should continue reducing the bodies of these two λ-abstractions. But this is exactly what was already done in the previous two reduction sequences.

Using the results of these calculations it is possible to construct the required bisimulation Q. First, define

$$\begin{aligned}
R = \{&(|f\,\Pi[f,\imath],\ \Psi[f]\,\Psi[f]), \\
&(x|\,\mathsf{let}\, z_1{=}!\imath \ \mathsf{in}\ \mathsf{let}\, z_2{=}f\,z_1 \ \mathsf{in}\ z_2\,x, \\
&\quad \mathsf{let}\, z{=}\Psi[f]\,\Psi[f] \ \mathsf{in}\ z\,x)\}.
\end{aligned}$$

Let $x_1, x_2, \ldots$ be distinct variables, and define, for every $n \geq 0$,

$$S_n = \{(z_2|z_2\,x_k,\ z_2\,x_k) \mid 1 \leq k \leq n\}.$$

Finally, define Q as the set of all tuples

$$(\{f, x_1, \ldots, x_n\}|\{\imath{:=}\Pi[f,\imath]\}, \{\}, R \cup S_n)$$

where $n \geq 0$. Then Q is a $\lambda\rho$-bisimulation, as can be verified using the calculations above.

Note that Q contains the tuple $(\{f\}|\{\imath{:=}\Pi[f,\imath]\}, \{\}, R)$ where $(|f\,\Pi[f,\imath],\ \Psi[f]\,\Psi[f]) \in R$. Therefore, Lemma 7 implies that $\mathsf{Y}_\rho \approx_\rho \mathsf{Y}_\mathsf{v}$. □

6. The $\lambda\mu\rho$-calculus

The $\lambda\mu\rho$-calculus combines the control aspects of the $\lambda\mu$-calculus with the state aspects of the $\lambda\rho$-calculus. The definition of $\lambda\mu\rho$-bisimilarity is a natural combination of the definitions of $\lambda\mu$-bisimilarity and of $\lambda\rho$-bisimilarity. However, unlike the cases for the calculi considered previously in the article, $\lambda\mu\rho$-bisimilarity is not only contained in contextual equivalence, it coincides with contextual equivalence, as will be shown in Section 10.

VARIABLES x, y, z
NAMES a, b
REFERENCES $\imath, \jmath$
VALUES $v \ ::= \ x \mid \lambda x.\,t$
NAMED TERMS $nt \ ::= \ [a]t$
TERMS $t \ ::= \ v \mid \ \mathsf{let}\, x{=}t_1 \ \mathsf{in}\ t_2 \mid v_1\,v_2 \mid \mu a.\,nt \mid$
$\qquad \rho s.\,t \mid \imath{:=}v; t \mid\ !\imath$
STORES $s \ ::= \ \{\imath_1{:=}v_1, \ldots, \imath_n{:=}v_n\}$

Reduction is defined on *configurations*, which are now pairs (s, nt) of stores and named terms such that $\text{FR}(nt) \subseteq \mathit{dom}(s)$.

NAMED EVAL. CONTEXTS $NE ::= [a][\,] \mid NE[\mathsf{let}\, x{=}[\,]\ \mathsf{in}\ t]$
NAMED ENFS $ne ::= [a]v \mid NE[x\,v]$

$(\mathrm{R}\mu\rho1)\ (s, NE[\mathsf{let}\, x{=}v\ \mathsf{in}\ t]) \mapsto (s, NE[t[{}^{v}\!/\!{}_{x}]])$
$(\mathrm{R}\mu\rho2)\ (s, NE[(\lambda x.\, t)\, v]) \mapsto (s, NE[t[{}^{v}\!/\!{}_{x}]])$
$(\mathrm{R}\mu\rho3)\ (s, NE[\mu a.\, nt]) \mapsto (s, nt[{}^{NE}\!/\!{}_{a}])$
$(\mathrm{R}\mu\rho4)\ (s, NE[\rho s'.\, t]) \mapsto (s{\cdot}s', NE[t])$,
if $(\mathit{dom}(s) \cup \mathrm{FR}(s) \cup \mathrm{FR}(NE)) \cap \mathit{dom}(s') = \emptyset$
$(\mathrm{R}\mu\rho5)\ (s, NE[\imath{:=}v; t]) \mapsto (s[\imath{:=}v], NE[t])$ if $\imath \in \mathit{dom}(s)$
$(\mathrm{R}\mu\rho6)\ (s, NE[!\imath]) \mapsto (s, NE[s(\imath)])$ if $\imath \in \mathit{dom}(s)$

Now X, Y, Z range over finite sets of variables and names. Let NR range over sets of triples $(X|nt, nt')$, more specifically subsets of $\mathit{NRel}(Y, J, J')$ for some Y, J and J', where

$$\mathit{NRel}(Y, J, J') = \{(X|nt, nt') \mid X \cap Y = \emptyset \,\&\, X{\cdot}Y, J \vdash nt \,\&\, X{\cdot}Y, J' \vdash nt'\}$$

We identify triples that differ only up to renaming of the variables and names from the first component X.

A *named term relation tuple* is a quadruple $(X|s, s', NR)$ where $X \vdash s, s'$ and $NR \subseteq \mathit{NRel}(X, \mathit{dom}(s), \mathit{dom}(s'))$. We identify named term relation tuples that differ only up to renaming of the variables and names from the first component X and up to renaming of references. A *named term relation set* is a set of named term relation tuples. Let NQ range over named term relations sets.

Definition 9. NQ is a *$\lambda\mu\rho$-bisimulation* iff $NQ \subseteq B_{\mu\rho}(NQ)$, where

$$\begin{aligned}
B_{\mu\rho}(NQ) = \{(X|s_0, s_0', NR_0) \mid\ & \text{for all } (Y|nt, nt') \in NR_0,\ \text{either} \\
& (s_0, nt) \mapsto^{\omega}\ \&\ (s_0', nt') \mapsto^{\omega},\ \text{or} \\
& \exists s_1, s_1', ne, ne', NR_1 \supseteq NR_0,\ X_1 \supseteq X{\cdot}Y. \\
& \quad (s_0, nt) \mapsto^{*} (s_1, ne)\ \& \\
& \quad (s_0', nt') \mapsto^{*} (s_1', ne')\ \& \\
& \quad (ne, ne') \in M_{\mu\rho}(NR_1)\ \& \\
& \quad (X_1|s_1, s_1', NR_1) \in NQ\} \\
M_{\mu\rho}(NR) = \{([a]v, [a]v'), (NE[x\,v], NE'[x\,v']) \mid\ & (v, v') \in V_{\mu\rho}(NR)\ \&\ (NE, NE') \in K_{\mu\rho}(NR)\} \\
V_{\mu\rho}(NR) = \{(x, x)\}\ & \\
\cup\ \{(v, v') \mid\ & \exists y \notin \mathrm{FV}(v) \cup \mathrm{FV}(v'). \\
& \exists a \notin \mathrm{FN}(v) \cup \mathrm{FN}(v'). \\
& \quad (a{\cdot}y|[a](v \star y), [a](v' \star y)) \in NR\} \\
K_{\mu\rho}(NR) = \{([a][\,], [a][\,])\}\ & \\
\cup\ \{(NE, NE') \mid\ & \exists y \notin \mathrm{FV}(NE) \cup \mathrm{FV}(NE'). \\
& \quad (y|NE \star y, NE' \star y) \in NR\}
\end{aligned}$$

Definition 10. Reference-closed named terms nt and nt' are *$\lambda\mu\rho$-bisimilar*, written $nt \approx_{\mu\rho} nt'$, iff there exists a $\lambda\mu\rho$-bisimulation NQ which contains a quadruple $(X|\{\}, \{\}, NR)$ with $(|nt, nt') \in NR$. Reference-closed terms t and t' are *$\lambda\mu\rho$-bisimilar*, written $t \approx_{\mu\rho} t'$, iff there exists a $\lambda\mu\rho$-bisimulation NQ which contains a quadruple $(X|\{\}, \{\}, NR)$ with $(t, t') \in T_{\mu\rho}(NR)$, where

$$T_{\mu\rho}(NR) = \{(t, t') \mid \exists a \notin \mathrm{FN}(t) \cup \mathrm{FN}(t').\ (a|[a]t, [a]t') \in NR\}.$$

We show in Section 9 that $\lambda\mu\rho$-bisimilarity is a congruence.

7. Example: one-shot continuations

As an extended example, we show the correctness of Friedman and Haynes's encoding of call/cc in terms of "one-shot continuations" [9].

A one-shot continuation is a continuation which may be applied at most once. Friedman and Haynes showed that, perhaps surprisingly, call/cc can be encoded in terms of its restricted one-shot variant. They did this by exhibiting an "extraordinarily difficult program" [9, p.248] together with an informal equivalence argument. We confirm the correctness of this program by a formal proof using the enf bisimulation method. The equivalence proof below can be viewed as a formalization of Friedman and Haynes's informal argument.

One cannot directly use the $\lambda\mu\rho$-calculus to prove correctness of this encoding of call/cc, since the $\lambda\mu\rho$-calculus does not contain one-shot continuations as a primitive. Instead, we define one-shot continuations in terms of unrestricted continuations using another, but simpler, construction due to Friedman and Haynes. We then show the correctness of the encoding of call/cc by means of one-shot continuations relative to this encoding of one-shot continuations.

First, we need to encode a conditional operator in the $\lambda\mu\rho$-calculus. Since the evaluation order in the $\lambda\mu\rho$-calculus is call-by-value, the encoding is done using "thunks":

$$\mathsf{T} = \lambda x.\, \lambda y.\, x\, \mathsf{I}$$
$$\mathsf{F} = \lambda x.\, \lambda y.\, y\, \mathsf{I}$$

$$\begin{aligned}
\mathsf{if}[t_1, t_2, t_3] = &\ \mathsf{let}\, z_1{=}t_1 \mathsf{in} \\
&\ \mathsf{let}\, z_2{=}z_1\, (\lambda z.\, t_2) \mathsf{in} \\
&\ z_2 (\lambda z.\, t_3)
\end{aligned}$$

where $\mathsf{I} = \lambda x.\, x$, and where z_1 and z_2 are not free in t_1, t_2, or t_3.

Recall the definition of call/cc:

$$\mathsf{call/cc} = \lambda f.\, \mu a.\, [a] f\, \hat{a}$$

where $\hat{a} = \lambda x.\, \mu b.\, [a]x$. Now define the one-shot variant of call/cc:

$$\begin{aligned}
\mathsf{call/cc1} = \lambda f.\, (&\mathsf{call/cc} \\
&(\lambda k.\, \rho\{\imath{:=}\mathsf{T}\}.\, f\, (\lambda x.\, \mathsf{if}[!\imath, (\imath{:=}\mathsf{F}; k\, x), \Omega])))
\end{aligned}$$

The requirement that every captured continuation k is applied at most once is enforced by means of the local reference $\imath$.

Now for the encoding of unrestricted continuations by means of one-shot continuations. For every reference $\jmath$, define

$$\begin{aligned}
\Phi_\jmath = \lambda g.\, \lambda f.\, \mathsf{let}\, y{=}&\mathsf{call/cc1} \\
&(\lambda k.\, (\jmath{:=}k; f\, (\lambda x.\, \mathsf{let}\, y{=}!\jmath \\
&\qquad\qquad \mathsf{in}\ y\, x))) \\
\mathsf{in}\ &\mathsf{call/cc1}\, (\lambda k'.\, g\, (\lambda k. k' y)).
\end{aligned}$$

Then define

$$\mathsf{call/cc}{*} = \lambda f.\, \rho\{\jmath{:=}\mathsf{I}\}.\, \mathsf{fix}[\Phi_\jmath]\, f.$$

(See the original presentation of the encoding [9] for an informal explanation of how it works.)

The aim of this section is to show that

$$\mathsf{call/cc} \approx_{\mu\rho} \mathsf{call/cc}{*}.$$

It follows that call/cc and call/cc$*$ are contextually equivalent, and hence that call/cc$*$ is as an encoding of call/cc by means of one-shot continuations.

As in Section 5, the equivalence could be shown directly from the definition of a bisimulation, but it is more convenient to use the following generalization of Lemma 7 to the $\lambda\mu\rho$-calculus:

Lemma 11. *Define $\hat{\rho} s.\, t = \rho s.\, t$ for $s \neq \{\}$, and $\hat{\rho}\{\}.\, t = t$. Assume that there exists a $\lambda\mu\rho$-bisimulation containing a tuple $(X|s, s', NR)$ where $(|[a]t, [a]t') \in NR$, and let $x_1, \ldots, x_n \in X$. If $a \in X$ does not occur free in any of s, s', t, and t', then $\lambda x_1 \ldots \lambda x_n.\, \hat{\rho} s.\, t \approx_{\mu\rho} \lambda x_1 \ldots \lambda x_n.\, \hat{\rho} s'.\, t'$.*

The lemma follows from Corollary 36 in Section 9.

Proposition 12. call/cc $\approx_{\mu\rho}$ call/cc$*$.

Proof. By definition, $\mathsf{call/cc} = \lambda f.\, \mu a.\, [a] f\, \hat{a}$ and $\mathsf{call/cc}{*} = \lambda f.\, \rho\{\jmath{:=}\mathsf{I}\}.\, \mathsf{fix}[\Phi_\jmath]\, f$. We therefore construct a bisimulation containing a tuple

$$(f{\cdot}a|\{\}, \{\jmath{:=}\mathsf{I}\}, \mathit{NR})$$

where $(|[a]\mu a.\,[a]f\,\hat{a},\ \ [a]\mathsf{fix}[\Phi_\jmath]\,f) \in \mathit{NR}$. The conclusion then follows from Lemma 11.

The main part of the proof consists in a number of calculations of reduction sequences. One starts from the two configurations $(\{\},\ [a]\mu a.\,[a]f\,\hat{a})$ and $(\{\jmath{:=}\mathsf{I}\},\ [a]\mathsf{fix}[\Phi_\jmath]\,f)$ and iteratively tries to add tuples in order to satisfy the conditions in the definition of a $\lambda\mu\rho$-bisimulation.

First, define the named evaluation context

$$\mathit{NE}_0 = [a]\,\mathsf{let}\,x{=}[\,]\,\mathsf{in}\ \mathsf{call/cc1}\,(\lambda k'.\,\mathsf{fix}[\Phi_\jmath]\,(\lambda k.k'x))$$

and for every reference $\imath$, define the term

$$\mathsf{C}[\imath] = \lambda x.\,\mathsf{if}[!\imath,(\imath{:=}F;(\lambda x.\,\mu b.\,\mathit{NE}_0[x])\,x),\Omega].$$

Now calculate, for any store s and any value v:

(1) $(s{\cdot}\{\jmath{:=}v\}, [a]\mathsf{fix}[\Phi_\jmath]\,f)$
$\mapsto^*$
$(s{\cdot}\{\jmath{:=}\mathsf{C}[\imath], \imath{:=}\mathsf{T}\}, \mathit{NE}_0[f\,(\lambda x.\ \mathsf{let}\,y{=}!\jmath\ \mathsf{in}\ y\,x)])$.

(2) $(s{\cdot}\{\jmath{:=}\mathsf{C}[\imath], \imath{:=}\mathsf{T}\}, [b]\,\mathsf{let}\,y{=}!\jmath\ \mathsf{in}\ y\,x)$
$\mapsto^*$
$(s{\cdot}\{\jmath{:=}\mathsf{C}[\imath], \imath{:=}\mathsf{F}\}, [a]\mathsf{call/cc1}\,(\lambda k'.\,\mathsf{fix}[\Phi_\jmath]\,(\lambda k.k'x)))$.

(3) $(s{\cdot}\{\jmath{:=}\mathsf{C}[\imath]\}, [a]\mathsf{call/cc1}\,(\lambda k'.\,\mathsf{fix}[\Phi_\jmath]\,(\lambda k.k'x)))$
$\mapsto^*$
$(s{\cdot}\{\jmath{:=}\mathsf{C}[\imath'], \imath_0{:=}\mathsf{F}, \imath'{:=}\mathsf{T}\}, [a]x)$.

These calculations dictate the following construction of a $\lambda\mu\rho$-bisimulation: let

$$\begin{aligned}\mathit{NR}_0 =\{&(|[a]\mu a.\,[a]f\,\hat{a},\ [a]\mathsf{fix}[\Phi_\jmath]\,f),\\ &(y\mid[a]y,\ [a]\mathsf{call/cc1}\,(\lambda k'.\,\mathsf{fix}[\Phi_\jmath]\,(\lambda k.k'y))),\\ &(y{\cdot}b\mid[b]\mu b.[a]y,\ [b]\,\mathsf{let}\,z{=}!\jmath\ \mathsf{in}\ z\,y)\}\end{aligned}$$

and let NQ consist of the tuple

$$(f{\cdot}a|\{\}, \{\jmath{:=}\mathsf{I}\}, \{(|[a]\mu a.\,[a]f\,\hat{a},\ [a]\mathsf{fix}[\Phi_\jmath]\,f)\})$$

together with all named term relation tuples of the form

$$(X|\{\}, s, \mathit{NR}_0)$$

where $\{f, a\} \subseteq X$, where s is a store such that $\jmath \in \mathit{dom}(s)$, and where there exists an $\imath \in \mathit{dom}(s)$ such that

$$s(\jmath) = C[\imath] \quad \text{and} \quad s(\imath) = \mathsf{T}.$$

Then NQ is a $\lambda\mu\rho$-bisimulation, as can be verified using the calculations (1)-(3) above. By Lemma 11, $\mathsf{call/cc} \approx_{\mu\rho} \mathsf{call/cc*}$. □

8. Enf bisimulation for terms with free references

So far in this article, eager normal form bisimulation has been used as a proof principle for proving equivalence of *reference-closed* terms. In this section it is shown how to extend eager normal form bisimulation to terms which may contain free references. Besides allowing one to prove equivalences about terms with free references, this extension is also used in the congruence proof for enf bisimilarity in Section 9. As a part of that proof, it must be shown that the following holds: If $t \approx_{\mu\rho} t'$ and $v \approx_{\mu\rho} v'$, then $\rho\{\imath{:=}v\}.\,t \approx_{\mu\rho} \rho\{\imath{:=}v'\}.\,t'$ and $\imath{:=}v;t \approx_{\mu\rho} \imath{:=}v';t'$. Here the reference $\imath$ will in general occur free in the terms t, t', v, and v', and, of course, in the terms $\imath{:=}v;t$ and $\imath{:=}v';t'$.

The modification needed to take free references into account can be explained as follows. Suppose that the free references of the terms t and t' are contained in J, and that one wants to prove that t and t' are equivalent. According to the previous definition, one requirement is that $[a]t$ and $[a]t'$ should either both diverge, or reduce to matching named eager normal forms. But one cannot reduce $[a]t$ and $[a]t'$ without providing values for the references in J, i.e., the references which are free in t and t'. The solution is to initialize the references in J with a number of fresh variables $z_\jmath{}^{\jmath\in J}$. This initialization takes care of the "input" aspect of the free references; the "output" aspect is taken care of by an extra requirement: if both $(\{\jmath{:=}z_\jmath{}^{\jmath\in J}\}, [a]t)$ and $(\{\jmath{:=}z_\jmath{}^{\jmath\in J}\}, [a]t')$ reduce to named eager normal forms, then in the two resulting stores, the references from J must contain values which are pairwise related.

Now for the formal definitions. Named term relation sets are generalized as follows: let

$$\begin{aligned}\mathit{NU}_J = \{&(X|s, s', \mathit{NR}) \mid\\ &X, J \vdash s, s'\ \&\\ &\mathit{NR} \subseteq \mathit{NRel}(X, J{\cdot}\mathit{dom}(s), J{\cdot}\mathit{dom}(s'))\}.\end{aligned}$$

We identify quadruples that differ only up to renaming of the variables and names from the first component X and up to renaming of references from $\mathit{dom}(s)$ and $\mathit{dom}(s')$. Notice that $\mathit{NU}_\emptyset = \mathit{NU}$.

Definition 13. *$\mathit{NQ} \subseteq \mathit{NU}_J$ is a J-bisimulation* iff $\mathit{NQ} \subseteq B_J(\mathit{NQ})$, where

$$\begin{aligned}&B_J(\mathit{NQ}) =\\ &\quad\{(X|s_0, s_0', \mathit{NR}_0) \in \mathit{NU}_J \mid\\ &\quad\ \text{for all distinct variables } z_\imath{}^{\imath\in J}\\ &\quad\ \text{and all } (Y|nt, nt') \in \mathit{NR}_0,\ \text{either}\\ &\quad\ (\{\imath{:=}z_\imath{}^{\imath\in J}\}{\cdot}s_0, nt) \mapsto^\omega\ \&\ (\{\imath{:=}z_\imath{}^{\imath\in J}\}{\cdot}s_0', nt') \mapsto^\omega,\ \text{or}\\ &\quad\ \exists ne, ne', (v_\imath, v_\imath')^{\imath\in J}, s_1, s_1', \mathit{NR}_1 \supseteq \mathit{NR}_0,\ X_1 \supseteq X{\cdot}Y{\cdot}z_\imath{}^{\imath\in J}.\\ &\qquad (\{\imath{:=}z_\imath{}^{\imath\in J}\}{\cdot}s_0, nt) \mapsto^* (\{\imath{:=}v_\imath{}^{\imath\in J}\}{\cdot}s_1, ne)\ \&\\ &\qquad (\{\imath{:=}z_\imath{}^{\imath\in J}\}{\cdot}s_0', nt') \mapsto^* (\{\imath{:=}v_\imath'{}^{\imath\in J}\}{\cdot}s_1', ne')\ \&\\ &\qquad (ne, ne') \in M_{\mu\rho}(\mathit{NR}_1)\ \&\\ &\qquad \forall\imath \in J.\ (v_\imath, v_\imath') \in V_{\mu\rho}(\mathit{NR}_1)\ \&\\ &\qquad (X_1|s_1, s_1', \mathit{NR}_1) \in \mathit{NQ}\}\end{aligned}$$

Say that two terms t and t' are *J-bisimilar* if there exists a J-bisimulation containing a tuple $(X|\{\},\{\},\mathit{NR})$ where $(t, t') \in T_{\mu\rho}(\mathit{NR})$.

We now generalize the previously given definition of enf bisimilarity for reference-closed terms:

Definition 14. Let t and t' be $\lambda\mu\rho$-terms. Say that t and t' are *$\lambda\mu\rho$-bisimilar*, written $t \approx_{\mu\rho} t'$, if there exists a finite set J of references such that t and t' are J-bisimilar.

Example 15. It is easy to show that

$$\mathsf{let}\,z{=}!\jmath\ \mathsf{in}\ (\jmath{:=}\mathsf{I};\jmath{:=}z;\,f\,x) \approx_{\mu\rho} f\,x$$

while on the other hand

$$\mathsf{let}\,z{=}!\jmath\ \mathsf{in}\ (\jmath{:=}\mathsf{I};\ \mathsf{let}\,y{=}f\,x\ \mathsf{in}\ (\jmath{:=}z;\ y)) \not\approx_{\mu\rho} f\,x.$$

The proofs of this equivalence and this non-equivalence illustrate a basic sequentiality property of the calculi considered in this article: in order for two terms to be equivalent, it is enough that the contents of the free references are equivalent at certain "synchronization points", but in-between these points the contents of the free references can be modified arbitrarily.

Proposition 16. *Let J_0 and J be finite sets of references such that $J_0 \subseteq J$. Any two terms which are J_0-bisimilar are also J-bisimilar.*

9. Congruence

This section contains an outline of the proof that $\lambda\mu\rho$-bisimilarity is a congruence: it is an equivalence relation which is furthermore compatible. A binary relation S on terms and named terms of the $\lambda\mu\rho$-calculus is *compatible* if it is closed under the term formation rules of the $\lambda\mu\rho$-calculus. For example, if $t_1\ S\ t_1'$ and $t_2\ S\ t_2'$, then also $(\mathsf{let}\,x{=}t_1\ \mathsf{in}\ t_2)\ S\ (\mathsf{let}\,x{=}t_1'\ \mathsf{in}\ t_2')$, and if $nt\ S\ nt'$, then $\mu a.\,nt\ S\ \mu a.\,nt'$. The straightforward formal definition is omitted.

Proposition 17. *For every finite set J of references, there exists a greatest J-bisimulation $\mathcal{B}_J$.*

Proof. The definition of B_J immediately implies that the union of an arbitrary family of J-bisimulations is also a J-bisimulation. In particular, the union of all J-bisimulations is the greatest J-bisimulation. $\square$

At this point it is useful to change the definitions of a $\lambda\mu\rho$-bisimulation and of a J-bisimulation slightly: in those definitions, replace the operators $V_{\mu\rho}$ and $K_{\mu\rho}$ with $V'_{\mu\rho}$ and $K'_{\mu\rho}$:

$$\begin{aligned}V'_{\mu\rho}(NR) = \{(v,v') \mid\ & \exists y \notin \text{FV}(v) \cup \text{FV}(v'). \\ & \exists a \notin \text{FN}(v) \cup \text{FN}(v'). \\ & (a{\cdot}y|[a]v\,y, [a]v'\,y) \in NR\}.\\ K'_{\mu\rho}(NR) = \{(NE,NE') \mid\ & \exists y \notin \text{FV}(NE) \cup \text{FV}(NE'). \\ & (y|NE[y], NE'[y]) \in NR\}.\end{aligned}$$

These modifications do not change the relation of $\lambda\mu\rho$-bisimilarity; in fact, the greatest J-bisimulation is unchanged. The two operators $V'_{\mu\rho}$ and $K'_{\mu\rho}$ are more convenient in the congruence proof below, while the other two operators are more convenient when using $\lambda\mu\rho$-bisimulation as a proof principle.

We first show that $\lambda\mu\rho$-bisimilarity is an equivalence relation.

Definition 18. Let $NQ \subseteq NU_J$.

1. NQ is *closed under weakening* if whenever $(X_0|s,s',NR) \in NQ$ and $X_0 \subseteq X$ for some finite set X of names and variables, also $(X|s,s',NR) \in NQ$.
2. NQ is *closed under context extrusion* if $(X|s,s',NR) \in NQ$ and $(Z_1{\cdot}Z_2|nt,nt') \in NR$ imply that there exists $NR' \supseteq NR \cup \{(Z_2|nt,nt')\}$ such that $(X{\cdot}Z_1|s,s',NR') \in NQ$.

Lemma 19. *The greatest J-bisimulation is closed under weakening and context extrusion.*

Lemma 20. *$\lambda\mu\rho$-bisimilarity is an equivalence relation.*

Proof sketch. Reflexivity and symmetry follow easily from the definition of B_J. As for transitivity, assume that $t \approx_{\mu\rho} t'$ and that $t' \approx_{\mu\rho} t''$; we must show that $t \approx_{\mu\rho} t''$ (and similarly for named terms). Proposition 16 implies that there exists some J such that t and t' are J-bisimilar and t' and t'' are J-bisimilar. Now consider a general composition construction on named term relation sets. Given $NR_1 \subseteq NRel(Y,J,J_1)$ and $NR_2 \subseteq NRel(Y,J,J_2)$, define their composition as

$$\begin{aligned}NR_1;NR_2 = \{(X|nt_1,nt_2) \mid\ & \exists nt.(X|nt_1,nt) \in NR_1\ \& \\ & (X|nt,nt_2) \in NR_2\},\end{aligned}$$

and given $NQ_1, NQ_2 \subseteq NU_J$, define

$$\begin{aligned}NQ_1;NQ_2 = \{(X|s_1,s_2,NR_1;NR_2) \mid\ & \\ \exists s.(X|s_1,s,NR_1) \in NQ_1\ & \& \\ (X|s,s_2,NR_2) \in NQ_2\}&.\end{aligned}$$

Then the following property holds: if NQ_1 and NQ_2 are J-bisimulations closed under weakening, then so is $NQ_1;NQ_2$. $\square$

It remains to show that $\lambda\mu\rho$-bisimilarity is compatible. The proof of this fact is structured as follows:

- First, we show that a restricted variant of $\lambda\mu\rho$-bisimilarity is *substitutive* in a sense defined below. (The restricted variant does not validate certain common extensionality rules for call-by-value calculi.)
- Second, we use a syntactic translation to show that full $\lambda\mu\rho$-bisimilarity is substitutive. It follows that $\lambda\mu\rho$-bisimilarity is compatible.

9.1 Substitutions

A *substitution* is a finite map σ with a domain consisting of variables and names, and such that σ maps each variable in its domain to a $\lambda\mu\rho$-calculus value, and each name in its domain to a $\lambda\mu\rho$-calculus named evaluation context. Let σ range over substitutions. When ϕ is a syntactic phrase (store, value, term, or named term), $\phi\sigma$ denotes the result of "carrying out the substitution" σ on ϕ (we omit the formal definitions). Also, define

$$NR(\sigma,\sigma') = \{(Z|nt\sigma, nt'\sigma') \mid (Z|nt,nt') \in NR\}$$

(where the variables and names occurring free in the ranges of σ and σ' are not in Z).

Let $dom(\sigma)$ denote the domain of σ. Say that

$$X \vdash \sigma\ \Sigma(NR)\ \sigma' : Y$$

when $dom(\sigma) = dom(\sigma') = Y$, and:

1. For every variable $x \in Y$, $(\sigma(x), \sigma'(x)) \in V'_{\mu\rho}(NR)$.
2. For every name $a \in Y$, $(\sigma(a), \sigma'(a)) \in K'_{\mu\rho}(NR)$.
3. The free variables and names in the ranges of σ and σ' are contained in X.

Say that two substitutions σ and σ' are $\lambda\mu\rho$-bisimilar (notation: $\sigma \approx_{\mu\rho} \sigma'$) if there exists a J-bisimulation containing a tuple $(X \cdot Y|\{\},\{\},NR)$ such that $X \vdash \sigma\ \Sigma(NR)\ \sigma' : Y$. In the next sections we show that $\lambda\mu\rho$-bisimilarity is *substitutive* in the following sense:

1. If $t \approx_{\mu\rho} t'$ and $\sigma \approx_{\mu\rho} \sigma'$, then $t\sigma \approx_{\mu\rho} t'\sigma'$.
2. If $nt \approx_{\mu\rho} nt'$ and $\sigma \approx_{\mu\rho} \sigma'$, then $nt\sigma \approx_{\mu\rho} nt'\sigma'$.

9.2 Non-η bisimulation

In order to show that $\lambda\mu\rho$-bisimilarity is substitutive, we first show the analogous result for a certain restricted variant of $\lambda\mu\rho$-bisimilarity. The variation consists in a change in the definition of the operators V and K (which are used to define relations on values and named evaluation contexts, respectively).

Definition 21. Let NR be a named term relation.

$$\begin{aligned}M^\dagger(NR) = \{([a]v,[a]v'), (NE[x\,v], NE'[x\,v']) \mid\ & \\ (v,v') \in V^\dagger(NR)\ \&\ (NE,NE') \in K^\dagger(NR)\}&\\ V^\dagger(NR) = \{(x,x) \mid x \text{ is a variable}\}\ & \\ \cup\ \{(\lambda x.\,t, \lambda x.\,t') \mid \exists a \notin \text{FN}(t) \cup \text{FN}(t').\ & \\ (x{\cdot}a|[a]t,[a]t') \in NR\}&\\ K^\dagger(NR) = \{([a][\,],[a][\,]) \mid a \text{ is a name}\}\ & \\ \cup\ \{(NE[\mathsf{let}\,x{=}[\,]\,\mathsf{in}\,t], NE'[\mathsf{let}\,x{=}[\,]\,\mathsf{in}\,t']) \mid\ & \\ x \notin \text{FV}(NE) \cup \text{FV}(NE')\ \&\ & \\ (x|NE[t],NE'[t']) \in NR\}&\end{aligned}$$

Definition 22.

1. For every named term relation set $NQ \subseteq NU_J$, the named term relation set $B^\dagger_J(NQ)$ is defined in the same way as $B_J(NQ)$, except that $M^\dagger$ and $V^\dagger$ are used instead of $M_{\mu\rho}$ and $V_{\mu\rho}$.
2. NQ is a *non-η J-bisimulation* if $NQ \subseteq B^\dagger_J(NQ)$.
3. Two reference-closed $\lambda\mu\rho$-terms t and t' are *non-η bisimilar* (notation: $t \approx^\dagger t'$) if there exist a finite set of references J and a non-η J-bisimulation containing a tuple $(X|\{\},\{\},NR)$ such that $(t,t') \in T_{\mu\rho}(NR)$. Non-$\eta$ bisimilarity of named terms is defined similarly.

Remark. The reason for the name "non-η" is that non-η bisimilarity does not satisfy two common extensionality rules for call-by-value calculi, namely the η_v-rule and the let_η-rule: $\lambda x.\, y\, x \not\approx^\dagger y$ and $(\mathsf{let}\, x{=}y\, z\ \mathsf{in}\ x) \not\approx^\dagger y\, z$.

Let $\mathcal{B}_J^\dagger$ be the greatest non-η J-bisimulation. The key to showing that non-η bisimilarity is substitutive is to show that $\mathcal{B}_J^\dagger$ is *closed under substitutions* in the sense defined next.

Definition 23.

1. For every $NQ \subseteq NU_J$, let

$$F^\dagger(NQ) = \{(X|s\sigma, s'\sigma', NR(\sigma,\sigma')) \mid \exists Y.\ (X{\cdot}Y|s,s',NR) \in NQ\ \&\ X \vdash \sigma\ \Sigma^\dagger(NR)\ \sigma' : Y\}$$

where $\Sigma^\dagger(NR)$ is defined in the same way as $\Sigma(NR)$, except that $V^\dagger$ and $K^\dagger$ are used in place of $V'_{\mu\rho}$ and $K'_{\mu\rho}$ in the definition.

2. A named term relation set $NQ \subseteq NU_J$ is *closed under substitutions* if $F^\dagger(NQ) \subseteq NQ$.

We now proceed to show that for every J, the greatest non-η J-bisimulation is closed under substitutions. Define the *substitutive closure* of NQ as

$$S^\dagger(NQ) = \bigcup_{n<\omega} (F^\dagger)^n(NQ).$$

It is the least fixed point of $F^\dagger$ containing NQ.

Main Lemma. *Let $NQ \subseteq NU_J$ be a non-η J-bisimulation which is closed under context extrusion. Let $(X|s,s',NR) \in (F^\dagger)^n(NQ)$ and $(Z|nt,nt') \in NR$ and $(v_\jmath, v'_\jmath) \in V^\dagger(NR)$ for all $\jmath \in J$.*

1. *Assume that $(\{\jmath{:=}v_\jmath{}^{\jmath\in J}\}{\cdot}s, nt) \mapsto^\star (\{\jmath{:=}w_\jmath{}^{\jmath\in J}\}{\cdot}s_1, ne_1)$ in m or fewer steps. Then there exist $X_1 \supseteq X{\cdot}Z$, s'_1, ne'_1, $w'_\jmath{}^{\jmath\in J}$, and $NR_1 \supseteq NR$ such that*

$$(\{\jmath{:=}v'_\jmath{}^{\jmath\in J}\}{\cdot}s', nt') \mapsto^\star (\{\jmath{:=}w'_\jmath{}^{\jmath\in J}\}{\cdot}s'_1, ne'_1),$$

$(X_1|s_1,s'_1,NR_1) \in S^\dagger(NQ)$, $(ne_1,ne'_1) \in M^\dagger(NR_1)$, and $(w_\jmath, w'_\jmath) \in V^\dagger(NR_1)$ for all $\jmath \in J$.

2. *Conversely, assume that*

$$(\{\jmath{:=}v'_\jmath{}^{\jmath\in J}\}{\cdot}s', nt') \mapsto^\star (\{\jmath{:=}w'_\jmath{}^{\jmath\in J}\}{\cdot}s'_1, ne'_1)$$

in m or fewer steps. Then there exist $X_1 \supseteq X{\cdot}Z$, s_1, ne_1, $w_\jmath{}^{\jmath\in J}$, and $NR_1 \supseteq NR$ such that

$$(\{\jmath{:=}v_\jmath{}^{\jmath\in J}\}{\cdot}s, nt) \mapsto^\star (\{\jmath{:=}w_\jmath{}^{\jmath\in J}\}{\cdot}s_1, ne_1)$$

etc.

Proof sketch. By induction on the pairs (m,n), ordered lexicographically. □

Corollary 24. *The greatest non-η J-bisimulation $\mathcal{B}_J^\dagger$ is closed under substitutions.*

Proof sketch. The Main Lemma implies that

$$(F^\dagger)^n(\mathcal{B}_J^\dagger) \subseteq B_J^\dagger(S^\dagger(\mathcal{B}_J^\dagger))$$

for all $n \geq 0$. By definition of $S^\dagger$ and the fact that non-η bisimulations are closed under unions, $S^\dagger(\mathcal{B}_J^\dagger) \subseteq B_J^\dagger(S^\dagger(\mathcal{B}_J^\dagger))$. This means that $S^\dagger(\mathcal{B}_J^\dagger)$ is a non-η J-bisimulation, and therefore $F^\dagger(\mathcal{B}_J^\dagger) \subseteq S^\dagger(\mathcal{B}_J^\dagger) \subseteq \mathcal{B}_J^\dagger$, since $\mathcal{B}_J^\dagger$ is the largest non-η J-bisimulation. □

9.3 Non-η bisimilarity is substitutive

In order to show that non-η bisimilarity is substitutive, one needs the following construction for combining named term relation sets:

Definition 25. Given $NQ_1, NQ_2 \subseteq NU_J$, define

$$NQ_1 + NQ_2 = \{(X|s_1{\cdot}s_2, s'_1{\cdot}s'_2, NR_1 \cup NR_2) \mid (X|s_1,s'_1,NR_1) \in NQ_1\ \&\ (X|s_2,s'_2,NR_2) \in NQ_2\ \&\ dom(s_1) \cap dom(s_2) = dom(s'_1) \cap dom(s'_2) = \emptyset\}.$$

Lemma 26. *If NQ_1 and NQ_2 are non-η J-bisimulations closed under weakening, then so is $NQ_1 + NQ_2$.*

Corollary 27. *The greatest non-η J-bisimulation $\mathcal{B}_J^\dagger$ satisfies that $\mathcal{B}_J^\dagger = \mathcal{B}_J^\dagger + \mathcal{B}_J^\dagger$.*

Finally, non-η bisimilarity is substitutive:

Theorem 28.

1. *If $t \approx^\dagger t'$ and $\sigma \approx^\dagger \sigma'$, then $t\sigma \approx^\dagger t'\sigma'$.*
2. *If $nt \approx^\dagger nt'$ and $\sigma \approx^\dagger \sigma'$, then $nt\sigma \approx^\dagger nt'\sigma'$.*

Proof sketch. We show the second implication—the first is completely similar. Assume that $nt \approx^\dagger nt'$ and $\sigma \approx^\dagger \sigma'$, and let J be the set of free references in nt, nt', σ, and σ'. Then the greatest non-η J-bisimulation $\mathcal{B}_J^\dagger$ contains a tuple $(X_1|\{\},\{\},NR_1)$ such that $(|nt,nt') \in NR_1$ and a tuple $(X_2 \cdot Y|\{\},\{\},NR_2)$ such that $X_2 \vdash \sigma\ \Sigma^\dagger(NR_2)\ \sigma' : Y$. Then by Corollary 27, $\mathcal{B}_J^\dagger$ also contains the tuple $(X_1 \cup X_2 \cup Y|\{\},\{\},NR_1 \cup NR_2)$. Finally, since $\mathcal{B}_J^\dagger$ is closed under substitutions, it also contains the tuple $((X_1 \setminus Y) \cup X_2|\{\},\{\},NR_1(\sigma,\sigma') \cup NR_2(\sigma,\sigma'))$ where $(|nt\sigma, nt\sigma') \in NR_1(\sigma,\sigma')$. Hence $nt\sigma \approx^\dagger nt'\sigma'$. □

9.4 $\lambda\mu\rho$-bisimilarity is substitutive

The fact that $\lambda\mu\rho$-bisimilarity is substitutive can be derived from the analogous result for non-η bisimilarity, Theorem 28, by means of a syntactic translation involving an "infinite η-expansion" combinator H.

Fix a finite set of references $J = \{\jmath_1, \ldots \jmath_n\}$. For every value v and every term t, define the term

$$\begin{aligned}\mathsf{app}[v,t] = {} & \mathsf{let}\, x_1{=}!\jmath_1\ \mathsf{in}\ \ldots \mathsf{let}\, x_n{=}!\jmath_n\ \mathsf{in} \\ & \mathsf{let}\, y_1{=}v\, x_1\ \mathsf{in}\ \ldots \mathsf{let}\, y_n{=}v\, x_n\ \mathsf{in} \\ & (\jmath_1{:=}y_1; \ldots \jmath_n{:=}y_n;\ t)\end{aligned}$$

(where $x_1, \ldots x_n, y_1, \ldots, y_n$ are not free in v or t). The operational behavior of $\mathsf{app}[v,t]$ is to "apply v to every reference in J" and then continue according to t. Now define

$$\begin{aligned}\mathsf{H}_0 &= \lambda z.\lambda f.\lambda x.\mathsf{let}\, y_1{=}z\, x\ \mathsf{in} \\ &\qquad \mathsf{app}[z,\ \mathsf{let}\, y_2{=}f\, y_1\ \mathsf{in}\ \mathsf{app}[z, z\, y_2]] \\ \mathsf{H} &= \mathsf{fix}[\mathsf{H}_0].\end{aligned}$$

The combinator H originates from a generalization of a "syntactic minimal invariance" equation [17, 29].

Also, for every value v and every named evaluation context NE, define

$$\begin{aligned}\mathsf{G}[v] &= \lambda x.\, \mathsf{let}\, y_1{=}\mathsf{H}\, x\ \mathsf{in} \\ &\qquad \mathsf{app}[\mathsf{H}, \mathsf{let}\, y_2{=}v\, y_1\ \mathsf{in}\ \mathsf{app}[\mathsf{H}, \mathsf{H}\, y_2]] \\ \mathsf{G}[NE] &= NE[\mathsf{let}\, x{=}[\,]\ \mathsf{in}\ \mathsf{app}[\mathsf{H}, \mathsf{H}\, x]].\end{aligned}$$

Definition 29.

1. For every term t, let $t^\ddagger$ be the result of substituting $\mathsf{G}[x]$ for every free variable x in t, and substituting $\mathsf{G}[[a][\,]]$ for every free name a in t. For every named term nt, define $nt^\ddagger$ similarly.
2. For every term t, define

$$t^\dagger = \mathsf{app}[\mathsf{H},\ \mathsf{let}\, x{=}t^\ddagger\ \mathsf{in}\ \mathsf{app}[\mathsf{H}, \mathsf{H}\, x]].$$

3. For every named term $nt = [a]t$, define $nt^\dagger = [a]t^\dagger$.

Using the above syntactic constructs, $\lambda\mu\rho$-bisimilarity can be characterized in terms of non-η bisimilarity:

Proposition 30. *Let the free references of t, t', v, and v' be contained in J.*

1. $t \eqsim_{\mu\rho} t'$ *iff* $t^{\dagger} \eqsim^{\dagger} t'^{\dagger}$.
2. $v \eqsim_{\mu\rho} v'$ *iff* $\mathsf{G}[v^{\ddagger}] \eqsim^{\dagger} \mathsf{G}[v'^{\ddagger}]$.
3. $nt \eqsim_{\mu\rho} nt'$ *iff* $nt^{\dagger} \eqsim^{\dagger} nt'^{\dagger}$.

If v is a value such that the free references of v are contained in J, then $v \eqsim_{\mu\rho} \mathsf{G}[v]$ (but in general $v \not\eqsim^{\dagger} \mathsf{G}[v]$). As will be shown next, a more general version of that property holds.

Definition 31. The binary relation R on terms, named evaluation contexts, and named terms is defined inductively by means of the inference rules in Figure 1.

$$\frac{}{t\ R\ t} \qquad \frac{v\ R\ v'}{v\ R\ \mathsf{G}[v']} \qquad \frac{t\ R\ t'}{\lambda x.\,t\ R\ \lambda x.\,t'}$$

$$\frac{v_1\ R\ v_1' \qquad v_2\ R\ v_2'}{v_1\,v_2\ R\ v_1'\,v_2'} \qquad \frac{v\ R\ v' \qquad t\ R\ t'}{(\imath{:=}v;t)\ R\ (\imath{:=}v';t')}$$

$$\frac{t\ R\ t' \qquad \forall\jmath \in J_0.s(\jmath)\ R\ s'(\jmath)}{\rho s.\,t\ R\ \rho s'.\,t'}\ \begin{array}{l} \mathit{dom}(s) = \mathit{dom}(s') = J_0, \\ J_0 \cap J = \emptyset \end{array}$$

$$\frac{nt\ R\ nt'}{\mu a.\,nt\ R\ \mu a.\,nt'} \qquad \frac{}{\mathit{NE}\ R\ \mathit{NE}} \qquad \frac{\mathit{NE}\ R\ \mathit{NE}'}{\mathit{NE}\ R\ \mathsf{G}[\mathit{NE}']}$$

$$\frac{t\ R\ t' \qquad \mathit{NE}\ R\ \mathit{NE}'}{\mathit{NE}[\mathsf{let}\,x{=}[\,]\ \mathsf{in}\ t]\ R\ \mathit{NE}'[\mathsf{let}\,x{=}[\,]\ \mathsf{in}\ t']}$$

$$\frac{\mathit{NE}\ R\ \mathit{NE}' \qquad t\ R\ t'}{\mathit{NE}[t]\ R\ \mathit{NE}'[t']}$$

Figure 1. The relation R.

Two stores s and s' are related by R if they have the same domain J_0 and if $s(\jmath)\ R\ s'(\jmath)$ for all $\jmath \in J_0$.

Proposition 32. *Let $J' \subseteq J$. The named term relation set*

$$\begin{array}{l}\{(X|s, s', \mathit{NR}) \in \mathit{NU}_{J'} \mid \\ \quad s\ R\ s'\ \&\ \mathit{NR} \subseteq \{(Z|nt, nt') \mid nt\ R\ nt'\}\}\end{array}$$

is a J'-bisimulation.

In particular, taking $J' = J$:

Corollary 33. *Let the free references of t, t', nt, and nt' be contained in J.*

1. $t\ R\ t'$ *implies* $t \eqsim_{\mu\rho} t'$.
2. $nt\ R\ nt'$ *implies* $nt \eqsim_{\mu\rho} nt'$.

It follows that $\lambda\mu\rho$-bisimilarity is substitutive:

Theorem 34.

1. *If $t \eqsim_{\mu\rho} t'$ and $\sigma \eqsim_{\mu\rho} \sigma'$, then $t\sigma \eqsim_{\mu\rho} t'\sigma'$.*
2. *If $nt \eqsim_{\mu\rho} nt'$ and $\sigma \eqsim_{\mu\rho} \sigma'$, then $nt\sigma \eqsim_{\mu\rho} nt'\sigma'$.*

Proof sketch. As a simple example, assume that $t \eqsim_{\mu\rho} t'$ and $v \eqsim_{\mu\rho} v'$; it must be shown that $t[{}^{v}/_{x}] \eqsim_{\mu\rho} t'[{}^{v'}/_{x}]$. By Corollary 33, $t[{}^{v}/_{x}] \eqsim_{\mu\rho} t[{}^{\mathsf{G}[\mathsf{G}[v]]}/_{x}]$. Hence by Proposition 30 and the fact that non-η bisimilarity is substitutive:

$$\begin{aligned}(t[{}^{v}/_{x}])^{\dagger} \eqsim^{\dagger} (t[{}^{\mathsf{G}[\mathsf{G}[v]]}/_{x}])^{\dagger} &= t^{\dagger}[{}^{\mathsf{G}[v^{\ddagger}]}/_{x}] \\ &\eqsim^{\dagger} t'^{\dagger}[{}^{\mathsf{G}[v'^{\ddagger}]}/_{x}] \\ &\eqsim^{\dagger} (t'[{}^{v'}/_{x}])^{\dagger}.\end{aligned}$$

By Proposition 30 again, $t[{}^{v}/_{x}] \eqsim_{\mu\rho} t'[{}^{v'}/_{x}]$. □

9.5 $\lambda\mu\rho$-bisimilarity is a congruence

Now it is shown that $\lambda\mu\rho$-bisimilarity is compatible, using the fact that it is substitutive.

Proposition 35. $\mathcal{B}_J = \mathcal{B}_J + \mathcal{B}_J$.

Corollary 36. *Let $\mathrm{FR}(nt) \cup \mathrm{FR}(nt') \subseteq J$, and let ${y_\jmath}^{\jmath\in J}$ be distinct variables not free in nt or nt'. Suppose that*

$$\begin{aligned}(\{\jmath{:=}{y_\jmath}^{\jmath\in J}\}, nt) &\mapsto^{*} (\{\jmath{:=}{v_\jmath}^{\jmath\in J}\}{\cdot}s_1, ne_1), \\ (\{\jmath{:=}{y_\jmath}^{\jmath\in J}\}, nt') &\mapsto^{*} (\{\jmath{:=}{v_\jmath'}^{\jmath\in J}\}{\cdot}s_1', ne_1'),\end{aligned}$$

and $(X_1|s_1, s_1', \mathit{NR}_1) \in \mathcal{B}_J$ with $(ne_1, ne_1') \in M_{\mu\rho}(\mathit{NR}_1)$ and $(v_\jmath, v_\jmath') \in V_{\mu\rho}(\mathit{NR}_1)$ for all $\jmath \in J$. Then $nt \eqsim_{\mu\rho} nt'$.

Theorem 37. *$\lambda\mu\rho$-bisimilarity is compatible.*

Proof sketch. The most complicated case to show is that $\lambda\mu\rho$-bisimilarity is closed under ρ-abstraction: if $t \eqsim_{\mu\rho} t'$ and also $v_\jmath \eqsim_{\mu\rho} v_\jmath'$ for all $\jmath \in J_0$, then

$$\rho\{\jmath{:=}{v_\jmath}^{\jmath\in J_0}\}.\,t \eqsim_{\mu\rho} \rho\{\jmath{:=}{v_\jmath'}^{\jmath\in J_0}\}.\,t'.$$

Here one proceeds in three steps:

1. If $z \notin \mathrm{FV}(t)$, then $\rho s.\,t \eqsim_{\mu\rho} \mathsf{let}\,x{=}\rho s.\,\lambda z.\,t\ \mathsf{in}\ x\,\mathsf{I}$.
2. If $v \eqsim_{\mu\rho} v'$ and $v_\jmath \eqsim_{\mu\rho} v_\jmath'$ for all $\jmath \in J_0$, and if the free references of all these values are contained in $J \supseteq J_0$, then
$$\rho\{\jmath{:=}\mathsf{G}[v_\jmath^{\ddagger}]^{\jmath\in J_0}\}.\,\mathsf{G}[v^{\ddagger}] \ \eqsim_{\mu\rho}\ \rho\{\jmath{:=}\mathsf{G}[v_\jmath'^{\ddagger}]^{\jmath\in J_0}\}.\,\mathsf{G}[v'^{\ddagger}].$$
3. If the free references of v and ${v_\jmath}^{\jmath\in J_0}$ are contained in $J \supseteq J_0$, then $\rho\{\jmath{:=}{v_\jmath}^{\jmath\in J_0}\}.\,v \eqsim_{\mu\rho} \rho\{\jmath{:=}\mathsf{G}[v_\jmath^{\ddagger}]^{\jmath\in J_0}\}.\,\mathsf{G}[v^{\ddagger}]$.

The third part follows from Proposition 32 and Corollary 36. The proof of the second part uses Corollary 36 and the following construction: for every $\mathit{NQ} \subseteq \mathit{NU}_J$ with $J_0 \subseteq J$, let $\mathit{NQ}\backslash J_0$ be the subset of $\mathit{NU}_{J\backslash J_0}$ defined by

$$\begin{aligned}\mathit{NQ}\backslash J_0 = \{(X_1|\{\jmath := {w_\jmath}^{\jmath\in J_0}\}{\cdot}s,\ \{\jmath := {w_\jmath'}^{\jmath\in J_0}\}{\cdot}s',\ \mathit{NR}) \mid \\ (X|s, s', \mathit{NR}) \in \mathit{NQ}\ \& \\ X \subseteq X_1\ \& \\ \forall\jmath \in J_0.\ (w_\jmath, w_\jmath') \in V^{\dagger}(\mathit{NR})\}.\end{aligned}$$

The Main Lemma implies that $\mathcal{B}_J^{\dagger}\backslash J_0 \subseteq \mathcal{B}_{J\backslash J_0}^{\dagger}$.

The other cases of the proof are simpler and use Theorem 34 and Corollary 36. □

In summary, the main result of this section:

Theorem 38. *$\lambda\mu\rho$-bisimilarity is a congruence: an equivalence relation which is furthermore compatible.*

Corollary 39. *Each of λ-bisimilarity, $\lambda\mu$-bisimilarity, and $\lambda\rho$-bisimilarity is a congruence.*

Proof. It is easy to see that two $\lambda\mu$-terms are $\lambda\mu$-bisimilar if and only if they are $\lambda\mu\rho$-bisimilar, and similarly for the other inclusions between the four calculi considered in this article. (Each extension is "fully abstract"). The statement of the corollary immediately follows. Suppose for example that $v_1 \eqsim_{\mu} v_1'$ and $v_2 \eqsim_{\mu} v_2'$. Then $v_1 \eqsim_{\mu\rho} v_1'$ and $v_2 \eqsim_{\mu\rho} v_2'$. Therefore, since $\lambda\mu\rho$-bisimilarity is a congruence, $v_1\,v_2 \eqsim_{\mu\rho} v_1'\,v_2'$. Finally, since $v_1\,v_2$ and $v_1'\,v_2'$ are $\lambda\mu$-terms, $v_1\,v_2 \eqsim_{\mu} v_1'\,v_2'$. □

Remark. Non-η bisimilarity is also a congruence. The relation between non-η bisimilarity and $\lambda\mu\rho$-bisimilarity is analogous to the relation between Böhm tree equivalence and Böhm tree equivalence up to η for the pure λ-calculus.

10. Full abstraction

In this section we show that $\lambda\mu\rho$-bisimilarity coincides with contextual equivalence for the $\lambda\mu\rho$-calculus.

First, let us say that a variable-closed and reference-closed named term nt *terminates*, written $nt\Downarrow$, iff $\exists s, ne.\ (\{\}, nt) \mapsto^* (s, ne)$. Then we define that terms t and t' are *contextually equivalent*, written $t \cong_{\mu\rho} t'$, iff for all names a and term contexts C such that $C[t]$ and $C[t']$ are variable-closed and reference-closed, $[a]C[t]\Downarrow \Leftrightarrow [a]C[t']\Downarrow$. It is easy to see that $\cong_{\mu\rho}$ is a congruence and, moreover, it is the largest congruence relation which satisfies that $t \cong_{\mu\rho} t'$ implies $[a]t\Downarrow \Leftrightarrow [a]t'\Downarrow$ for all names a and variable-closed and reference-closed terms t and t'. Since $\lambda\mu\rho$-bisimilarity is a congruence, it is immediate from its definition that it is included in contextual equivalence, viz. that $\lambda\mu\rho$-bisimilarity is sound with respect to contextual equivalence.

Theorem 40 (Soundness). $\precapprox_{\mu\rho} \subseteq \cong_{\mu\rho}$.

Similarly, λ-bisimilarity, $\lambda\mu$-bisimilarity, and $\lambda\rho$-bisimilarity are included in contextual equivalence for their respective calculi.

To prove the converse of Theorem 40, we will form a bisimulation which relates all contextually equivalent terms. The task is similar to the Böhm-out proof of the separability theorem in the call-by-name λ-calculus. For readers familiar with the Böhm-out proof, we briefly compare our approach: The Böhm-out proof substitutes, for each free variable, a function that takes many arguments. This makes it possible to control the function's behaviour separately each time it is called. We need the same level of control over the behaviour of the functions and continuations that are substituted for the free variables and names in the contextually equivalent terms we want to prove bisimilar. However, instead of using functions that take many arguments, we use stateful functions and continuations. They use mutable references to execute pre-programmed strategies that specify how they will behave each time they are invoked. Moreover, we use the expressive power of the $\lambda\mu$-calculus to capture not only the argument when a function substituted for a free variable is invoked but also the continuation. The presence of mutable references introduces one complication, namely it requires us to store every argument and continuation we see so that we can invoke them multiple times to expose stateful behaviour.

We will need to accumulate values in lists and random access list entries. We encode the empty list as the identity function I and we encode the list v with elements $v_1, \ldots, v_n$ appended as:

$$\begin{aligned}&\langle v|v_1, \ldots, v_n\rangle = \\ &\quad \lambda z.\ \mathsf{let}\, x_0{=}v\, z \ \mathsf{in}\ \mathsf{let}\, x_1{=}x_0\, v_1 \ \mathsf{in} \ldots \mathsf{in}\ x_{n-1}\, v_n\end{aligned}$$

where $z, x_0, \ldots, x_{n-1}$ are not free in $v, v_1, \ldots, v_n$. When $v = \mathsf{I}$, we write just $\langle v_1, \ldots, v_n\rangle$. We access the q'th element in the list v with $v\#q = \mu a.\ [a]v\ \lambda x_1 \ldots \lambda x_{q-1}.\ \hat{a}$, where a is not free in v.

We use a designated reference $\jmath_0$ to store a list with all the arguments v and continuations *NE* we see along the way (each *NE* is stored as the value $\lambda x.\ \mu a.\ \mathit{NE}[x]$ with x, a not free in *NE*). We let w range over both pairs of values and pairs of named evaluation contexts. Given such a pair, let w^{V} be the pair of values and let w^{NR} be the singleton named term relation defined as:

$$w^{\mathsf{V}} = \begin{cases} (v_1, v_2) & \text{if } w = (v_1, v_2) \\ (\lambda x.\ \mu a.\ \mathit{NE}_1[x], \lambda x.\ \mu a.\ \mathit{NE}_2[x]) & \text{if } w = (\mathit{NE}_1, \mathit{NE}_2) \end{cases}$$

$$w^{\mathsf{NR}} = \begin{cases} (a{\cdot}x|[a](v_1 \star x), [a](v_2 \star x)) & \text{if } w = (v_1, v_2) \\ (x|\mathit{NE}_1 \star x, \mathit{NE}_2 \star x) & \text{if } w = (\mathit{NE}_1, \mathit{NE}_2) \end{cases}$$

where x and a are not free in $v_1, v_2, \mathit{NE}_1, \mathit{NE}_2$.

For each free variable x_i and name a_j, we associate a reference $\imath_i$ and $\jmath_j$, respectively. Each reference stores a *strategy* p which is a sequence of *moves* m, one for each successive invocation of the variable x_i or name a_j, ending in "success" $\top$ or "failure" $\bot$:

$$\begin{aligned}&\textsc{Strategies}\ p \ ::= \ \top \mid \bot \mid m; p \\ &\textsc{Moves}\ m \ ::= \ \mathsf{move}(q, p_1, p_2) \qquad (q \geq 1)\end{aligned}$$

A move $\mathsf{move}(q, p_1, p_2)$ "plays" the q'th value or continuation from the list stored in $\jmath_0$ and associates the strategy p_1 with a variable, which is used as argument, and p_2 with a name, which is used as continuation. These ideas are expressed in the following encodings. Given a reference $\imath$ and a strategy p, we define the function $[\![p]\!](\imath)$, which takes a function argument x and a continuation y as arguments, and for every move m, we define the term $[\![m]\!]$, as follows.

$$\begin{aligned}&[\![\top]\!](\imath) = \lambda x.\ \lambda y.\ \mathsf{I} \\ &[\![\bot]\!](\imath) = \lambda x.\ \lambda y.\ \Omega \\ &[\![m; p]\!](\imath) = \lambda x.\ \lambda y.\ \mathsf{let}\, z{=}!\jmath_0 \ \mathsf{in}\ \jmath_0{:=}\langle z|x, y\rangle; \imath{:=}[\![p]\!](\imath); [\![m]\!] \\ &[\![\mathsf{move}(q, p_1, p_2)]\!] = \\ &\quad \mathsf{let}\, z{=}!\jmath_0 \ \mathsf{in}\ \mathsf{let}\, x{=}z\#q \ \mathsf{in}\ \rho\{\imath{:=}[\![p_1]\!](\imath), \jmath{:=}[\![p_2]\!](\jmath)\}.\ \check{\jmath}\,[x\,\hat{\imath}]\end{aligned}$$

where $\hat{\imath}$ and $\check{\jmath}$ denote the value and the evaluation context

$$\begin{aligned}\hat{\imath} &= \lambda x.\ \mu a.\ [a_0]\ \mathsf{let}\, y{=}!\imath \ \mathsf{in}\ \mathsf{let}\, z{=}y\, x \ \mathsf{in}\ z\, \hat{a} \\ \check{\jmath} &= \mathsf{let}\, x{=}[\,] \ \mathsf{in}\ \mathsf{let}\, y{=}!\jmath \ \mathsf{in}\ \mathsf{let}\, z{=}y\, x \ \mathsf{in}\ z\, \hat{a_0}\end{aligned}$$

and where a_0 is a designated "top-level" name.

These are the building blocks we use to "separate" terms. As in [21], our separation proof is co-inductive. We define:

$$\begin{aligned}\mathcal{W} = \{&(x_1, \ldots, x_m, a_1, \ldots, a_n|s_1, s_2, \{{w_1}^{\mathsf{NR}}, \ldots, {w_q}^{\mathsf{NR}}\}) \\ &\mid \forall \text{ strategies } p_1, \ldots, p_m, p'_1, \ldots, p'_n. \\ &\quad \forall \text{ distinct references } \imath_1, \ldots, \imath_m, \jmath_1, \ldots, \jmath_n \\ &\qquad\qquad \notin \mathit{dom}(s_1) \cup \mathit{dom}(s_2). \\ &\quad \forall \text{ moves } m. \\ &\quad [a_0]\rho s{\cdot}s'_1\sigma.\ [\![m]\!]\Downarrow \Leftrightarrow [a_0]\rho s{\cdot}s'_2\sigma.\ [\![m]\!]\Downarrow \\ &\quad \text{where } s = \{\imath_1{:=}[\![p_1]\!](\imath_1), \ldots, \imath_m{:=}[\![p_m]\!](\imath_m), \\ &\qquad\qquad \jmath_1{:=}[\![p'_1]\!](\jmath_1), \ldots, \jmath_n{:=}[\![p'_n]\!](\jmath_n)\}, \\ &\qquad \sigma = [\hat{\imath}_1/x_1, \ldots, \hat{\imath}_m/x_m, [a_0]\check{\jmath}_1/a_1, \ldots, [a_0]\check{\jmath}_n/a_n], \\ &\qquad s'_i = s_i{\cdot}\{\jmath_0{:=}\langle v_{i1}, \ldots, v_{iq}\rangle\}, \\ &\qquad (v_{11}, v_{21}) = {w_1}^{\mathsf{V}}, \ldots, (v_{1q}, v_{2q}) = {w_q}^{\mathsf{V}}\}\end{aligned}$$

Lemma 41. *$\mathcal{W}$ is a $\lambda\mu\rho$-bisimulation.*

Proof. By detailed analysis of the possible operational behaviours of the named terms in each ${w_i}^{\mathsf{NR}}$ triple. □

Lemma 42. $t_1 \cong_{\mu\rho} t_2$ *implies* $t_1 \precapprox_{\mu\rho} t_2$ *if* $\mathrm{FR}(t_1) \cup \mathrm{FR}(t_2) = \emptyset$.

Proof. Suppose t_1 and t_2 are reference-closed terms, $t_1 \cong_{\mu\rho} t_2$, and $X \vdash t_1, t_2$. Then, by the definition of $\mathcal{W}$ and contextual equivalence, $t_1 \cong_{\mu\rho} t_2$ implies $(X|\{\}, \{\}, \{w^{\mathsf{NR}}\}) \in \mathcal{W}$ where $w = (\lambda x.\ t_1, \lambda x.\ t_2)$ and $x \notin X$. Observe that $w^{\mathsf{NR}} = (a{\cdot}x|[a]t_1, [a]t_2)$. We conclude $t_1 \precapprox_{\mu\rho} t_2$ by Lemmas 41 and 19. □

To extend this result to general terms, we define a term context L_J that "translates" any term t with $\mathrm{FV}(t) \subseteq J = \{\imath_1, \ldots \imath_n\}$ to the reference-closed closed term

$$\begin{aligned}L_J[t] = \lambda x.\ \rho\{&\imath_1{:=}\mathsf{I}, \ldots, \imath_n{:=}\mathsf{I}\}. \\ &\langle \lambda x.\ t, \mathsf{get}(\imath_1), \mathsf{set}(\imath_1), \ldots, \mathsf{get}(\imath_n), \mathsf{set}(\imath_n)\rangle\end{aligned}$$

where $x \notin \mathrm{FV}(t)$, $\mathsf{get}(\imath_i) = \lambda x.\ !\imath_i$, and $\mathsf{set}(\imath_i) = \lambda x.\ (\imath_i{:=}x; \mathsf{I})$.

Lemma 43. $t_1 \precapprox_{\mu\rho} t_2$ *iff* $L_J[t_1] \precapprox_{\mu\rho} L_J[t_2]$.

Theorem 44 (Completeness). $\precapprox_{\mu\rho} \supseteq \cong_{\mu\rho}$.

Proof. Suppose $J = \text{FR}(t_1) \cup \text{FR}(t_2)$. Then

$$\begin{aligned} t_1 \cong_{\mu\rho} t_2 &\Rightarrow L_J[t_1] \cong_{\mu\rho} L_J[t_2] && \cong_{\mu\rho} \text{ is a congruence} \\ &\Rightarrow L_J[t_1] \precsim_{\mu\rho} L_J[t_2] && \text{Lemma 42} \\ &\Rightarrow t_1 \precsim_{\mu\rho} t_2 && \text{Lemma 43.} \quad \square \end{aligned}$$

Acknowledgments

We thank Olivier Danvy and Andrzej Filinski for guidance and comments on this work. We also thank Matthias Felleisen and the anonymous referees for comments and suggestions, and Radha Jagadeesan, Paul Levy, and Corin Pitcher for discussions.

The first author is supported by BRICS (Basic Research in Computer Science, funded by the Danish National Research Foundation).

References

[1] S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.

[2] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proc. 13th Annual IEEE Symposium on Logic in Computer Science*, pages 334–344, 1998.

[3] Z. M. Ariola and H. Herbelin. Minimal classical logic and control operators. In *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 871–885. Springer-Verlag, 2003.

[4] 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.

[5] G. M. Bierman. A computational interpretation of the lambda-mu calculus. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proc. 23rd Mathematical Foundations of Computer Science, Brno, Czech Republic*, volume 1450, pages 336–345. Springer-Verlag, 1998.

[6] R. David and W. Py. $\lambda\mu$-calculus and Böhm's theorem. *Journal of Symbolic Logic*, 66(1):407–413, 2001.

[7] L. Egidi, F. Honsell, and S. Ronchi della Rocca. Operational, denotational and logical descriptions: a case study. *Fundamenta Informaticae*, 16(2):149–169, 1992.

[8] M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103:235–271, 1992.

[9] D. P. Friedman and C. T. Haynes. Constraining control. In *Proc. 12th ACM Symposium on Principles of Programming Languages*, pages 245–254, 1985.

[10] A. D. Gordon. Bisimilarity as a theory of functional programming. *Theoretical Computer Science*, 228(1–2):5–47, 1999.

[11] A. D. Gordon and A. M. Pitts, editors. *Higher Order Operational Techniques in Semantics*. Publications of the Newton Institute. Cambridge University Press, 1998.

[12] P. de Groote. On the relation between the lambda-mu-calculus and the syntactic theory of sequential control. In *LPAR '94*, volume 822 of *Lecture Notes in Artificial Intelligence*, pages 31–43. Springer-Verlag, 1994.

[13] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *Proc. 33rd ACM Symposium on Principles of Programming Languages*, pages 141–152, 2006.

[14] J. Laird. Full abstraction for functional languages with control. In *Proc. 12th Annual IEEE Symposium on Logic in Computer Science*, pages 58–67, 1997.

[15] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[16] S. B. Lassen. Bisimulation up to context for imperative lambda calculus. Unpublished note. Presented at *The Semantic Challenge of Object-Oriented Programming*, Schloss Dagstuhl, 1998.

[17] S. B. Lassen. Relational reasoning about contexts. In Gordon and Pitts [11], pages 91–135.

[18] S. B. Lassen. Bisimulation in untyped lambda calculus: Böhm trees and bisimulation up to context. In *MFPS XV*, volume 20 of *Electronic Notes in Theoretical Computer Science*, pages 346–374. Elsevier, 1999.

[19] S. B. Lassen. Eager normal form bisimulation. In *Proc. 20th Annual IEEE Symposium on Logic in Computer Science*, pages 345–354, 2005.

[20] S. B. Lassen. Normal form simulation for McCarty's amb. In *MFPS XXI*, volume 155 of *Electronic Notes in Theoretical Computer Science*, pages 445–465. Elsevier, 2005.

[21] S. B. Lassen. Head normal form bisimulation for pairs and the $\lambda\mu$-calculus (extended abstract). In *Proc. 21th Annual IEEE Symposium on Logic in Computer Science*, 2006.

[22] P. B. Levy. Game semantics using function inventories. Talk given at *Geometry of Computation 2006*, Marseille, 2006.

[23] I. A. Mason and C. L. Talcott. Equivalence in functional languages with effects. *Journal of Functional Programming*, 1(3):297–327, 1991.

[24] M. Merro and C. Biasi. On the observational theory of the CPS-calculus (extended abstract). In *Proc. 22nd Conference on Mathematical Foundations of Programming Semantics*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 307–330. Elsevier, 2006.

[25] A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables: Preliminary report. In *Proc. 15th ACM Symposium on Principles of Programming Languages, San Diego, CA*, 1988.

[26] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In A. Voronkov, editor, *Proceedings Intl. Conf. on Logic Programming and Automated Reasoning, LPAR'92, St Petersburg*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.

[27] R. P. Perez. An extensional partial combinatory algebra based on λ-terms. In A. Tarlecki, editor, *Proc. Mathematical Foundations of Computer Science*, volume 520 of *Lecture Notes in Computer Science*, pages 387–396. Springer-Verlag, 1991.

[28] A. M. Pitts. Reasoning about local variables with operationally-based logical relations. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-Like Languages*, volume 2, chapter 17, pages 173–193. Birkhauser, 1997. Reprinted from *Proceedings Eleventh Annual IEEE Symposium on Logic in Computer Science*, 1996, pp 152–163.

[29] A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. In Gordon and Pitts [11], pages 227–273.

[30] E. Ritter and A. M. Pitts. A fully abstract translation between a λ-calculus with reference types and Standard ML. In *Proc. 2nd International Conference on Typed Lambda Calculus and Applications, Edinburgh*, volume 902 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[31] D. Sangiorgi. The lazy lambda calculus in a concurrency scenario. *Information and Computation*, 111(1):120–153, 1994.

[32] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. In *Proc. 32nd ACM Symposium on Principles of Programming Languages*, pages 63–74, 2005.

[33] C. Talcott. Reasoning about functions with effects. In Gordon and Pitts [11], pages 347–390.

[34] M. Wand and G. T. Sullivan. Denotational semantics using an operationally-based term model. In *Proc. 24th ACM Symposium on Principles of Programming Languages*, pages 386–399, 1997.