Higher Order Symb Comput manuscript No.
(will be inserted by the editor)

A Mechanized Bisimulation for the Nu-Calculus

Nick Benton · **Vasileios Koutavas**

12 September 2012 (preprint)

Abstract We introduce a Sumii-Pierce-Koutavas-Wand-style bisimulation for the nu-calculus of Pitts and Stark, a simply-typed lambda calculus with fresh name generation. This bisimulation coincides with contextual equivalence and provides a usable and elementary method for establishing all the subtle equivalences given by Stark. We also describe the formalisation of soundness and of the examples in the Coq proof assistant.

Keywords Functional programming with effects · Name generation · Contextual equivalence · Bisimulation · The Coq proof assistant

1 Introduction

Generative local names are ubiquitous: objects (as in Java), exceptions, references (as in ML), channels (as in the π-calculus), cryptographic keys (as in the spi-calculus or cryptographic lambda calculus) are all first-class things-with-identity that can be generated freshly within some scope. The ν-calculus of Pitts and Stark [33,41] is a simply-typed, call-by-value lambda calculus over the base types of names, ν, and booleans, o, that captures the essence of this kind of situation in a deceptively minimal way. Names can be generated freshly, tested for equality and passed around, but that is all; there are no other effects (not even divergence) in the language. Though austere, the ν-calculus can express many important aspects of generativity, locality and independence, and has proved to have a remarkably complex theory. The central problem is to find models and reasoning principles for establishing contextual equivalence of ν-calculus terms.

Names are an *abstract* type in the ν-calculus: contextual equivalence only involves observations on booleans, and the only way to examine a name is to test it for equality with

Koutavas was partially supported by SFI project SFI 06 IN.1 1898.

N. Benton (✉)
Microsoft Research, 7 J J Thomson Avenue, Cambridge, CB3 0FB, United Kingdom
E-mail: nick@microsoft.com

V. Koutavas
School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, Ireland
E-mail: Vasileios.Koutavas@scss.tcd.ie

another one. An elementary example of the kind of equivalence we wish to be able to establish is the following:

$$\nu n.\, \nu n'.\, e \quad \cong \quad \nu n'.\, \nu n.\, e \tag{1}$$

The LHS generates a fresh name, n, then another, n', both of which are bound in the expression e. The RHS is similar, but generates the two names in the opposite order; the equivalence thus expresses that the order in which names are generated is non-observable. Equation (1) matches one's intuitions about names, but already requires a modicum of sophistication to model: the obvious (adequate) denotational semantics that models names as naturals and interprets expressions in the state monad $TX = \mathbb{N} \to \mathbb{N} \times X$, passing around and incrementing 'the next free name', for example, fails to validate it. Moving to the functor category $Set^{\mathscr{I}}$, where $\mathscr{I}$ is the category of finite sets and injections, yields a model that validates (1), but is still far from fully abstract.

The subtle interaction of generativity with higher-order functions, and the restricted nature of contexts, lead to the ν-calculus satisfying various more complex equivalences that challenge both intuition and our ability to reason formally. The canonical 'hard' example is the following:

$$\nu n.\, \nu n'.\, \lambda f{:}\nu \to o.\, (f\ n = f\ n') \quad \cong \quad \lambda f{:}\nu \to o.\, \mathtt{true} \tag{2}$$

The LHS generates two fresh names, n and n', and yields an abstraction that accepts a function f from names to booleans and returns the result of comparing $f\ n$ with $f\ n'$. One's first thought might be that this equivalence holds because any f that is passed to the result of evaluating the LHS cannot 'know' either of the fresh names n and n', and must therefore treat them the same; thus the equality test must always return `true`. But this explanation is far too naïve, as the following program, resulting from plugging the LHS of (2) into a cunningly-designed context, reveals:

$$\begin{array}{l} \mathtt{let}\ F = \nu n.\, \nu n'.\, \lambda f{:}\nu \to o.\, (f\ n = f\ n') \\ \mathtt{in\ let}\ G = \lambda x{:}\nu.\, F\ (\lambda y{:}\nu.\, (x = y)) \\ \quad \mathtt{in}\ F\ G \end{array} \tag{3}$$

The whole program does return indeed `true`, as we would expect if the equivalence (2) is to hold, but the calls to F that occur inside G actually return `false`! Furthermore, naïve intuition might lead one to believe that the following *in*equivalence is actually an equivalence:

$$\nu n.\, \lambda f{:}\nu \to o.\, \nu n'.\, (f\ n = f\ n') \quad \ncong \quad \lambda f{:}\nu \to o.\, \mathtt{true}$$

The context (3), where the term bound to F is replaced by each of the terms above, can be used to distinguish them. In the LHS case, the calls of F inside G will this time return different values.

Pitts and Stark have established many equivalences using logical relations, both directly over the operational semantics and denotationally, further refining the functor category model mentioned above. Zhang and Nowak [49] define a Kripke logical relation over a similar functor category model. None of these techniques is complete, however, failing in particular to prove equivalences such as (2) above. Stark [41] achieved a proof of (2) using a technique based on operational logical relations that is tailored to fit this particular example, without an obvious completeness result.

Jeffrey and Rathke [19] define a sound and complete bisimulation for an extension of the ν-calculus with assignment (for which (2) is *not* a valid equivalence) and observe that

their analysis "illuminates the difficulties involved in finding fully abstract models for ν-calculus proper". More recently, the problem has been attacked using game semantics. Laird [24] constructs a game model using automorphisms of names that is fully abstract for a language like that of Jeffrey and Rathke. Abramsky et al. [3] use games in the topos of FM-sets to construct the first model of ν-calculus proper that is both fully abstract and can be used to validate (2). Discrepancies in that model were recently rectified by Tzevelekos [46]. The proofs of (2) using Stark's logical relations, the amended games semantics model of Abramsky et al., and the bisimulation technique we present in this paper are summarised and discussed by Tzevelekos [47].

In this paper we provide a sound and complete theory for reasoning about contextual equivalence in the ν-calculus using bisimulation, which is rather more elementary than games in nominal sets. The form of bisimulation we use was introduced by Sumii and Pierce for proving equivalences in lambda calculi with cryptographic operations [43] and existential and recursive types [44] and later developed by Koutavas and Wand for reasoning about untyped imperative higher-order [22] and object [21] calculi. Instead of just being a binary relation on terms, Sumii and Pierce's bisimulations are *sets* of relations, each element of which intuitively corresponds to a different 'state of knowledge' of the surrounding context. We too will work with sets, $\mathscr{X}$, of typed relations, R, each of which is annotated by two sets of (generated) names, s and s'. We remark that, although this kind of bisimulations has already been shown sound and complete for other languages with 'difficult' features, such as higher-order references, it certainly does not automatically follow that similar results will hold for the, apparently simpler, ν-calculus. The very paucity of contexts in the simpler language actually yields a more complex equational theory than is the case for richer ones; it is easier to achieve completeness for extensions of the calculus (vide supra).

The theoretical development broadly follows that of previous work by Koutavas and Wand [22,21,23]. We start by defining when a set of relations is *adequate* — a restatement of the conditions for being contained in contextual equivalence that is arranged to be establishable by induction. We then investigate the class of all such inductive proofs by abstracting over the actual contents of the sets and attempting a *proof construction scheme*. By this process we find proof obligations that the sets should satisfy in order to be adequate. Our main theorem says that if a set satisfies exactly these conditions, then it is adequate, and, by soundness, all terms related under the empty stores in this set are contextually equivalent.

Having a provably sound and complete reasoning principle is good, but we also want something that is usable in practice. A further contribution, beyond the development of the general metatheory, is that we show that our bisimulation really does give an elementary method for establishing interesting equivalences, including the tricky (2) above. The proof of (2) is particularly interesting in making *two* uses of our technique: the adequacy of an initial relation is established via that of another. The second relation is used to show that any argument $v : \nu \to o$ created by the context outside of the dynamic extent of the functions—i.e. before n and n' are revealed to the context—cannot distinguish between the two private names, even if these names are given as arguments to v. In this way we deduce the crucial operational fact about how the expression $v\,n = v\,n'$ will evaluate via bisimulation-based reasoning about the *equivalence* of $v\,n$ and $v\,n'$.

The third contribution is a formalisation of the soundness of the metatheory and of the examples in the Coq theorem prover. We discuss the formalisation in Section 8; the proof script is available on the web via the authors' homepages.

$$
\begin{array}{rrll}
\textsc{Type}: & T & ::= & o \quad \text{Boolean} \\
& & \mid & \nu \quad \text{Name} \\
& & \mid & T \to T \quad \text{Function} \\
\textsc{Expression}: & e,d & ::= & x \quad \text{Identifier} \\
& & \mid & n \quad \text{Name} \\
& & \mid & \texttt{true} \mid \texttt{false} \quad \text{Boolean Constants} \\
& & \mid & \lambda x{:}T.\, e \quad \text{Abstraction} \\
& & \mid & e\ e \quad \text{Application} \\
& & \mid & \texttt{new} \quad \text{Fresh Name Generation} \\
& & \mid & (e = e) \quad \text{Name Equality} \\
& & \mid & \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \quad \text{Conditional} \\
\textsc{Value}: & u,v,w & ::= & n \mid \texttt{true} \mid \texttt{false} \\
& & \mid & \lambda x{:}T.\, e \\
\textsc{Name}: & n & & \\
\textsc{Nameset}: & s,t & \in & \mathscr{P}_{\mathsf{fin}}(\textsc{Name})
\end{array}
$$

Fig. 1 Syntactic domains of the ν-calculus

$\boxed{s;\Gamma \vdash e : T}$

$$
\textsc{Typ-Var}\ \frac{x:T \in \Gamma}{s;\Gamma \vdash x:T} \qquad \textsc{Typ-Name}\ \frac{n \in s}{s;\Gamma \vdash n:\nu} \qquad \textsc{Typ-Bool}\ \frac{b \in \{\texttt{true},\texttt{false}\}}{s;\Gamma \vdash b:o}
$$

$$
\textsc{Typ-Abs}\ \frac{s;\Gamma,x:T_1 \vdash e:T_2}{s;\Gamma \vdash \lambda x{:}T_1.\, e : T_1 \to T_2} \qquad \textsc{Typ-App}\ \frac{s;\Gamma \vdash e_0 : T_1 \to T_2 \qquad s;\Gamma \vdash e_1 : T_1}{s;\Gamma \vdash e_0\ e_1 : T_2}
$$

$$
\textsc{Typ-Cond}\ \frac{s;\Gamma \vdash e_0 : o \qquad s;\Gamma \vdash e_1 : T \qquad s;\Gamma \vdash e_2 : T}{s;\Gamma \vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : T}
$$

$$
\textsc{Typ-New}\ \frac{}{s;E \vdash \texttt{new} : \nu} \qquad \textsc{Typ-Eq}\ \frac{s;E \vdash e_1 : \nu \qquad s;E \vdash e_2 : \nu}{s;E \vdash (e_1 = e_2) : o}
$$

Fig. 2 Typing rules for the ν-calculus

2 The ν-calculus

The ν-calculus is a simply-typed lambda calculus over base types of names and booleans, extended with a conditional construct and operations for generating and comparing names. The expression `new` generates a fresh name, and $(n_1 = n_2)$ returns `true` when n_1 and n_2 are the same name. We often write $\nu x.\, e$ as an abbreviation of the expression $(\lambda x{:}\nu.\, e)\ \texttt{new}$,[1]

[1] Pitts and Stark take $\nu x.\, e$ as primitive and define `new` as $\nu x.\, x$, which results to evaluation trees of different size. The proof technique we develop here is based on an induction on the size of evaluation trees, but it is not affected by this difference since the evaluation of all constructs is of size at least one. Therefore the induction hypotheses are equally applicable in example equivalences in both settings.

$\boxed{s \vdash e \Downarrow^k (t)\, w}$

$$\textsc{Eval-Val}\ \frac{k > 0}{s \vdash v \Downarrow^k (\emptyset)\, v} \qquad \textsc{Eval-New}\ \frac{n \notin s \qquad k > 0}{s \vdash \mathtt{new} \Downarrow^k (\{n\})\, n}$$

$$\textsc{Eval-Cond}\ \frac{s \vdash e_0 \Downarrow^{k_0} (t_0)\, b \qquad s \oplus t_0 \vdash e_i \Downarrow^k (t)\, w \qquad (i,b) \in \{(1,\mathtt{true}),(2,\mathtt{false})\}}{s \vdash \mathtt{if}\ e_0\ \mathtt{then}\ e_1\ \mathtt{else}\ e_2 \Downarrow^{1+k_0+k} (t_0 \oplus t)\, w}$$

$$\textsc{Eval-Eq1}\ \frac{s \vdash e_1 \Downarrow^{k_1} (t_1)\, n \qquad s \oplus t_1 \vdash e_2 \Downarrow^{k_2} (t_2)\, n \qquad n \in s}{s \vdash (e_1 = e_2) \Downarrow^{1+k_1+k_2} (t_1 \oplus t_2)\, \mathtt{true}}$$

$$\textsc{Eval-Eq2}\ \frac{s \vdash e_1 \Downarrow^{k_1} (t_1)\, n_1 \qquad s \oplus t_1 \vdash e_2 \Downarrow^{k_2} (t_2)\, n_2 \qquad n_1, n_2 \text{ distinct}}{s \vdash (e_1 = e_2) \Downarrow^{1+k_1+k_2} (t_1 \oplus t_2)\, \mathtt{false}}$$

$$\textsc{Eval-App}\ \frac{s \vdash e_0 \Downarrow^{k_0} (t_0)\, \lambda x{:}T_1.\, e_2 \qquad s \oplus t_0 \vdash e_1 \Downarrow^{k_1} (t_1)\, w_1 \qquad s \oplus t_0 \oplus t_1 \vdash e_2[w_1/x] \Downarrow^{k_2} (t_2)\, w}{s \vdash e_0\ e_1 \Downarrow^{1+k_0+k_1+k_2} (t_0 \oplus t_1 \oplus t_2)\, w}$$

Fig. 3 Operational semantics for the ν-calculus

and $(e = e')$, when e and e' have type o, as syntactic sugar for

$$(\lambda x{:}o.\, \lambda y{:}o.\, \mathtt{if}\ x\ \mathtt{then}\ y\ \mathtt{else}\ (\mathtt{if}\ y\ \mathtt{then}\ \mathtt{false}\ \mathtt{else}\ \mathtt{true}))\ e\ e'$$

Furthermore, we use an overbar to denote sequences. Names are drawn from an infinite set NAME, of which finite subsets are called *namesets*. We write $s \oplus t$ for the disjoint union of namesets s and t. All syntactic domains of the ν-calculus are shown in Figure 1.

The typing judgement $s; \Gamma \vdash e : T$ states that the expression e has type T under the nameset s and typing environment Γ. The typing rules are standard and shown in Figure 2. We write $\lambda \overline{x{:}T}.\, e$ for the abstraction $\lambda x_1{:}T_1. \ldots \lambda x_n{:}T_n.\, e$ and $\overline{T} \to T$ for the type $T_1 \to \ldots \to T_n \to T$.

The evaluation judgement $s \vdash e \Downarrow^k (t)\, w$ states that the closed, well-typed expression e, under the nameset s, terminates with the value w, generating the set of fresh names t in the process. The judgement further records that the *size* of the evaluation tree is less than k. We write $s \vdash e \Downarrow (t)\, w$ when there exists some k for which $s \vdash e \Downarrow^k (t)\, w$, and $s \vdash e \Downarrow$ when $s \vdash e \Downarrow (t)\, w$, for some t and w. Figure 3 shows the evaluation rules of the ν-calculus.

Typing is stable under the addition of names. Evaluation preserves types, and is stable under the addition and removal of unused names. It is also total and deterministic, modulo fresh name generation.

Lemma 2.1 *If $s; \Gamma \vdash e : T$ and $s \cap s_0 = \emptyset$ then $s \oplus s_0; \Gamma \vdash e : T$.*

Proof By induction on $s; \Gamma \vdash e : T$. □

Lemma 2.2 (Type Preservation) *If $s; \cdot \vdash e : T$ and $s \vdash e \Downarrow (t)\, v$ then $s \oplus t; \cdot \vdash v : T$.*

Proof By induction on k. □

Lemma 2.3 (Garbage Addition) *If* $s \vdash e \Downarrow^k (t)\, w$ *and* $s \cap s_0 = t \cap s_0 = \emptyset$ *then* $s \oplus s_0 \vdash e \Downarrow^k (t)\, w$.

Proof By induction on $s \vdash e \Downarrow^k (t)\, w$. □

Lemma 2.4 (Garbage Collection) *If* $s \oplus s_0 \vdash e \Downarrow^k (t)\, w$ *and* $s_0 \cap \mathit{names}(e) = \emptyset$ *then* $s \vdash e \Downarrow^k (t)\, w$.

Proof We prove this lemma by proving a more general property. Namely that if $s_1 \vdash e \Downarrow^k (t)\, w$ and $s_0 \cap \mathit{names}(e) = \emptyset$ then for all s such that $s_1 = s \oplus s_0$,

$$s \vdash e \Downarrow^k (t)\, w \qquad s_0 \cap \mathit{names}(v) = \emptyset$$

We prove the property by induction on $s \vdash e \Downarrow^k (t)\, w$. □

Lemma 2.5 (Totality) *If* $s; \cdot \vdash e : T$ *then* $s \vdash e \Downarrow$.

Lemma 2.6 (Determinacy of Evaluation at o-Type) *If* $s \vdash e \Downarrow (t_1)\, b_1$, $s \vdash e \Downarrow (t_2)\, b_2$, *and* $\emptyset; \cdot \vdash b_i : o$ $(i = 1,2)$ *then* $b_1 = b_2$.

We prove both of the above lemmas simultaneously as in Chapter 2 of Stark's thesis [41]. We construct the following unary logical relation:

$$\begin{aligned}
\mathbb{V}(s, o) &= \{\mathtt{true}, \mathtt{false}\} \\
\mathbb{V}(s, \nu) &= s \\
\mathbb{V}(s, T_1 \to T_2) &= \{\lambda x{:}T_1.\, e \mid s; \cdot \vdash \lambda x{:}T.\, e : T_1 \to T_2 \wedge \\
&\qquad \forall s', v \in \mathbb{V}(s \oplus s', T_1).\ e[v/x] \in \mathbb{E}(s \oplus s', T_2)\} \\
\mathbb{E}(s, T) &= \{e \mid s; \cdot \vdash e : T \wedge \\
&\qquad \exists t, w.\ s \vdash e \Downarrow (t)\, w \wedge t, w \text{ are unique up to renaming of names in } t\}
\end{aligned}$$

Lemmas 2.5 and 2.6 are consequences of the following totality lemma for the logical relation.

Lemma 2.7 *If* $s; \overline{x : T} \vdash e : T$ *and* $s; \cdot \vdash \overline{v : T}$ *then* $e[\overline{v/x}] \in \mathbb{E}(s, T)$.

Proof By induction on $s; \overline{x : T} \vdash e : T$. □

3 Overview of the Technical Development

In Sections 4, 5, and 6 we develop our theory of equivalence for the ν-calculus. We begin with the standard definition of contextual equivalence ($\equiv$) for the language (Definition 4.1). This relates two possibly open expressions e and e' if, when placed in the (single) hole of any variable-capturing context C and evaluated under any store s, they evaluate to the same boolean constant:

$$(\exists t.\ s \vdash C[e] \Downarrow (t)\, b) \iff (\exists t.\ s \vdash C[e'] \Downarrow (t)\, b)$$

The quantification over possible contexts, stores, and evaluation trees makes it awkward to use the definition of contextual equivalence directly in proofs of non-trivial equivalences. Amongst the problems one runs into are the following:

1. The capturing of free variables makes substitution into contexts hard to work with.

2. Evaluation may duplicate, or generate more, related values, thus leading one to have to consider multi-hole contexts.
3. During evaluation the "Kripke world" of the equivalence changes: related expressions may generate different number of names, some of which may be revealed to the context and added to its "knowledge", while others remain hidden from the context.
4. A proof of equivalence would have to reason about intermediate computations.

As in previous work [44,43,22,21,23], we will take an approach that can be seen as having a component that addresses each of these issues. The complexity of variable-capturing substitution is dealt with by restricting our attention to closed values. This suffices because, as we will prove in Theorem 4.2, one can close open expressions under appropriate number of abstractions and reason about the closed values instead:

$$s;\overline{x:T} \vdash e \equiv e' : T \iff s;\cdot \vdash \lambda\overline{x{:}T}.\, e \equiv \lambda\overline{x{:}T}.\, e' : \overline{T} \to T$$

Given a relation R over closed values, we use the construction R^{cxt} (Definition 4.4) to generate all multi-hole contexts with related closed values in corresponding holes. This addresses the second complication.

The construction of $(-)^{\mathsf{cxt}}$ uses name-free contexts, which can only access names related in R via holes of type ν. Hence, we make an important distinction between names in the knowledge of the context (those related in R) and the rest, known only to the related terms. Moreover, we annotate relations with the namesets under which the related terms can be evaluated, forming tuples of the form (s,s',R) which we call *annotated relations*. These tuples represent the Kripke worlds of the equivalence and address the third complication.

With the above, we give a more convenient definition of equivalence which we call *Pre-Adequacy* (Definition 4.6) and prove it sound and complete with respect to contextual equivalence. This equivalence is the set of all *pre-adequate* annotated relations (Definition 4.5); i.e., the set of all annotated relations (s,s',R) for which whenever e and e' are program contexts of type o related in R^{cxt},

$$(\exists t.\ s \vdash e \Downarrow (t)\, b) \iff (\exists t'.\ s' \vdash e' \Downarrow (t')\, b)$$

Pre-Adequacy is a stepping stone towards the definition of *Adequacy* (Definition 4.10), our main equivalence, which addresses the final complication for an effective proof technique.

Compared to Pre-Adequacy, Adequacy considers program contexts of arbitrary type, again drawn from R^{cxt}, and requires final values and stores of related computations to be in extensions of the starting relation R. Thus, it enables reasoning about intermediate computations that do not necessarily evaluate to constants. Because of this condition, Adequacy can be seen as a *big-step bisimulation*.

We show that Adequacy coincides with Pre-Adequacy, and therefore is sound and complete with respect to contextual equivalence. As we discuss in Section 5, this gives us an effective proof technique of equivalence. Roughly, to prove two terms equivalent, it suffices to construct a set $\mathscr{X}$ of annotated relations that relates the terms in question under all stores, and show that this set is adequate (and therefore included in Adequacy).

The proof that $\mathscr{X}$ is adequate can always be organised as an induction on the sizes of the evaluations of related expression contexts: if an expression context evaluates to a value, then it must do so with an evaluation tree of a finite size k, giving rise to a mathematical induction principle. We therefore identify a pair of induction hypotheses, $IH_{\mathscr{X}}(k)$ and $IH_{\mathscr{X}^{-1}}(k)$, for the forward and reverse conditions of the definition for adequate sets, such that

$$\mathscr{X} \text{ is adequate} \iff \forall k.\ IH_{\mathscr{X}}(k) \wedge IH_{\mathscr{X}^{-1}}(k)$$

Although usable, the proof technique based on Adequacy relies greatly on intuition to construct a convenient set that can be shown adequate. We make this task easier by giving in Section 6 smaller proof obligations for a set of annotated relations to be adequate. These conditions are necessary and sufficient for any set $\mathscr{X}$ to be adequate (Theorem 6.1), and provide a better guide for the construction of adequate sets. With these conditions we prove two example equivalences in Section 7, including the difficult one discussed in the introduction.

4 Equivalence and Adequacy

Here we define contextual equivalence in the usual way and develop a theory of adequate relations. We then show that the largest adequate relation coincides with contextual equivalence.

4.1 Contextual Equivalence

Contextual Equivalence is a typed binary relation on open terms, indexed by a nameset, type environment, and type

$$(\equiv) \in \textsc{Nameset} \times \textsc{TypeEnv} \to \mathscr{P}(\textsc{Expression} \times \textsc{Expression} \times \textsc{Type})$$

We write $s;\Gamma \vdash e \equiv e' : T$ when $(e,e',T) \in ((\equiv)\, s\,\Gamma)$ and similarly for other typed relations. We also leave implicit the assumption that both terms do actually have the type at which they are related (i.e. $s;\Gamma \vdash e : T$, and similarly for e') in such judgements.

To define contextual equivalence we first need to define typed contexts. We write $s;\Gamma \vdash C[\cdot]^{T'}_{\Gamma'} : T$ to mean that $C[\cdot]$ is a single-hole context such that whenever $s;\Gamma' \vdash e : T'$ then $s;\Gamma \vdash C[e] : T$, where $C[e]$ is the capturing substitution of e for the hole in $C[\cdot]$.

The ν-calculus is normalising, hence, as in [41], we take as our notion of observation the (in-)equality of final values at type o.

Definition 4.1 (Contextual Equivalence $(\equiv)$**)** Write $s;\Gamma \vdash e \equiv e' : T$ if and only if for all contexts C with $s;\cdot \vdash C[\cdot]^{T}_{\Gamma} : o$ and boolean values b:

$$(\exists t.\ s \vdash C[e] \Downarrow (t)\, b) \iff (\exists t.\ s \vdash C[e'] \Downarrow (t)\, b).$$

Note that the generation of fresh names is *not* directly observable.

Part of the difficulty of reasoning about contextual equivalence is the capturing of the free variables of terms by the context. The following theorem simplifies the situation, by allowing us to consider only closed values:

Theorem 4.2 (Expression Closedness) *For any two expressions e and e' with $s;\overline{x:T} \vdash e, e' : T$*

$$s;\overline{x:T} \vdash e \equiv e' : T \quad \iff \quad s;\cdot \vdash \lambda\overline{x{:}T}.\, e \equiv \lambda\overline{x{:}T}.\, e' : \overline{T} \to T.$$

The proof of Theorem 4.2 is presented in Appendix A.

4.2 Pre-Adequacy

Reasoning about intermediate states of ν-calculus programs will require us to consider related values that allocate different sets of names. So, although the definition of contextual equivalence only involves one nameset, our development of the theory of (pre-) adequate relations is based on typed relations on closed values, annotated by *two* namesets

$$(s,s',R) \in \textsc{Nameset} \times \textsc{Nameset} \times \mathscr{P}(\textsc{Value}_{\emptyset} \times \textsc{Value}_{\emptyset} \times \textsc{Type})$$

Each annotated relation will be used to describe part of our semantic equivalences (i.e. pre-adequacy and adequacy), since contextual equivalence is defined at every nameset whereas an annotated relation is defined only at a specific pair of namesets.

We write $s,s';\cdot \vdash v\ R\ v' : T$ when (s,s',R) is an annotated relation and $(v,v',T) \in R$. The metavariable $\mathscr{X}$ ranges over *sets* of such relations:

$$\mathscr{X} \subseteq \textsc{Nameset} \times \textsc{Nameset} \times \mathscr{P}(\textsc{Value}_{\emptyset} \times \textsc{Value}_{\emptyset} \times \textsc{Type})$$

Definition 4.3 If $\mathscr{X}$ is a set of annotated relations, the *inverse* of $\mathscr{X}$, written $\mathscr{X}^{-1}$, is defined as

$$(s',s,R^{-1}) \in \mathscr{X}^{-1} \quad \textit{iff} \quad (s,s',R) \in \mathscr{X}$$

An important construction on annotated relations is *context closure*, substituting related values into identical, name-free contexts. The context closure of a relation R only allows contexts direct access to names that are R-related, via substitution into holes of type ν. This makes a crucial distinction between names in the knowledge of the context (i.e. those that are related in R) and names that are private to the terms.

Definition 4.4 (Context Closure of Annotated Relations) If (s,s',R) is an annotated relation on closed values, then (s,s',R^{cxt}) is the relation defined by

$$\frac{\emptyset;\overline{x:T} \vdash d : T \qquad s,s';\cdot \vdash \overline{u}\ R\ \overline{u'} : \overline{T}}{s,s';\cdot \vdash d[\overline{u/x}]\ R^{\mathsf{cxt}}\ d[\overline{u'/x}] : T}$$

Using the context closure of annotated relations we give our definition of pre-adequacy for the ν-calculus, which closely resembles the standard definition of contextual equivalence. In fact, we show that the open extension of pre-adequacy coincides with contextual equivalence.

Definition 4.5 (Pre-Adequate Annotated Relations) An annotated relation, (s,s',R), is *pre-adequate* if and only if for all expressions e and e', such that $s,s';\cdot \vdash e\ R^{\mathsf{cxt}}\ e' : o$, we have

$$(\exists t.\ s \vdash e \Downarrow (t)\, b) \iff (\exists t'.\ s' \vdash e' \Downarrow (t')\, b).$$

Definition 4.6 (Pre-Adequacy ($\cong$)) Write $(\cong)$ for the set of all pre-adequate annotated relations.

To provide a connection between sets of annotated relations and contextual equivalence, we extend such sets to indexed relations on open expressions.

Definition 4.7 (Open Extension of Sets of Annotated Relations) If $\mathscr{X}$ is a set of annotated relations, then $\mathscr{X}^\circ$ is an indexed relation on open expressions such that $s;\overline{x:T} \vdash e\ \mathscr{X}^\circ\ e' : T$ if and only if there exists R such that

$$\begin{array}{c}(s,s,R) \in \mathscr{X} \\ s,s;\cdot \vdash (\lambda\overline{x{:}T}.\,e)\ R\ (\lambda\overline{x{:}T}.\,e') : \overline{T} \to T \\ \forall n \in s.\ \ s,s;\cdot \vdash n\,R\,n : \nu\end{array}$$

Note that the contexts in the definition of contextual equivalence and the contexts involved in the definition of pre-adequate relations differ in their treatment of names: the former may contain any name in the corresponding nameset, while the latter are name-free and have access only to related names via substitution. The definition above reconciles the two notions of context by requiring R to be the identity on all names in the namesets.

Theorem 4.8 (Soundness and Completeness of $(\cong)$) *The open extension of pre-adequacy coincides with contextual equivalence:* $(\cong)^\circ = (\equiv)$.

Proof Let $\overline{x:T}$ be a non-empty type environment, s a nameset with $s = \{\overline{n}\}$, and e and e' expressions with $s;\overline{x:T} \vdash e,e' : T$. Then

$$s;\overline{x:T} \vdash e \equiv e' : T$$

if and only if, by Theorem 4.2,

$$s;\cdot \vdash \lambda\overline{x{:}T}.\,e \equiv \lambda\overline{x{:}T}.\,e' : \overline{T} \to T$$

if and only if, by the definition of $(\equiv)$,

$$\begin{array}{l}\forall C,b.\ \ s;\cdot \vdash C[\cdot]^{\overline{T}\to T}_{\emptyset} : o \\ \qquad \Longrightarrow\ (\exists t.\ \ s \vdash C[\lambda\overline{x{:}T}.\,e] \Downarrow (t)\,b) \iff (\exists t.\ \ s \vdash C[\lambda\overline{x{:}T}.\,e'] \Downarrow (t)\,b)\end{array}$$

if and only if, by choosing the appropriate C for the forward direction (and the appropriate d for the reverse), such that $\emptyset;\overline{y:\nu},z:\overline{T}\to T \vdash d : o$ and $s;z:\overline{T}\to T \vdash C[z] = d[\overline{n/y}] : o$, and because capturing substitution of a closed term coincides with capture-avoiding substitution of the same term,

$$\begin{array}{l}\forall \overline{y},z,d.\ \ \emptyset;\overline{y:\nu},z:\overline{T}\to T \vdash d : o \\ \qquad \Longrightarrow\ (\exists t.\ \ s \vdash d[\overline{n/y},\lambda\overline{x{:}T}.\,e/z] \Downarrow (t)\,b) \\ \qquad\qquad \iff (\exists t.\ \ s \vdash d[\overline{n/y},\lambda\overline{x{:}T}.\,e'/z] \Downarrow (t)\,b)\end{array}$$

if and only if, by choosing $R = \{(\lambda\overline{x{:}T}.\,e, \lambda\overline{x{:}T}.\,e', \overline{T}\to T), \overline{(n,n,\nu)}\}$ for the forward direction,

$$\begin{array}{l}\exists R.\ \ s,s;\cdot \vdash \lambda\overline{x{:}T}.\,e\ R\ \lambda\overline{x{:}T}.\,e' : \overline{T}\to T \\ \quad \wedge\ \forall n \in s.\ \ s,s;\cdot \vdash n\,R\,n : \nu \\ \quad \wedge\ \forall e_d,e'_d.\ \ s,s;\cdot \vdash e_d\ R^{\mathsf{cxt}}\ e'_d : o \ \Longrightarrow\ (\exists t.\ \ s \vdash e_d \Downarrow (t)\,b) \\ \qquad\qquad\qquad\qquad\qquad\qquad \iff (\exists t.\ \ s \vdash e'_d \Downarrow (t)\,b)\end{array}$$

if and only if, by Definition 4.5 and the definition of $(\cong)$,

$$\begin{array}{l}\exists R.\ \ s,s;\cdot \vdash \lambda\overline{x{:}T}.\,e\ R\ \lambda\overline{x{:}T}.\,e' : \overline{T}\to T \\ \quad \wedge\ \forall n \in s.\ \ s,s;\cdot \vdash n\,R\,n : \nu \\ \quad \wedge\ (s,s,R) \in (\cong)\end{array}$$

if and only if, by Definition 4.7,

$$s;\overline{x:T} \vdash e\ (\cong)^{\circ}\ e' : T \qquad \square$$

4.3 Adequacy

Our main technical tool for reasoning about equivalence in the ν-calculus is the definition of adequate sets of annotated relations. This definition permits the use of an induction in the proofs of equivalence.

Definition 4.9 (Adequate Sets of Annotated Relations) A set of annotated relations $\mathscr{X}$ is *adequate* if and only if for all $(s,s',R) \in \mathscr{X}$ we have

$$\begin{array}{l}\forall e,e',t,w.\ \ s,s';\cdot \vdash e\ R^{\mathsf{cxt}}\ e' : T \\ \qquad \wedge\ s \vdash e \Downarrow (t)\,w \\ \qquad\quad \Longrightarrow\ \exists t',w',Q.\ \ s' \vdash e' \Downarrow (t')\,w' \\ \qquad\qquad\qquad \wedge\ (T = o)\ \Longrightarrow\ (w = w') \\ \qquad\qquad\qquad \wedge\ s{\oplus}t, s'{\oplus}t';\cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T \\ \qquad\qquad\qquad \wedge\ (s{\oplus}t, s'{\oplus}t', Q) \in \mathscr{X} \\ \qquad\qquad\qquad \wedge\ R \subseteq Q\end{array}$$

and similarly for all $(s,s',R) \in \mathscr{X}^{-1}$.

It is easy to see that the union of adequate sets is an adequate set. Thus, the union of all adequate sets is the largest adequate set.

Definition 4.10 (Adequacy $(\approx)$) Write $(\approx)$ for the largest adequate set of annotated relations.

We now show that adequacy is sound and complete with respect to contextual equivalence by showing that it coincides with pre-adequacy.

Theorem 4.11 (Soundness of Adequate Sets) *If $\mathscr{X}$ is adequate then it is included in pre-adequacy.*

Proof Immediate by the definitions of pre-adequate annotated relations and adequate sets of annotated relations. $\square$

Theorem 4.12 (Completeness of Adequate Sets) *Pre-adequacy, $(\cong)$, is adequate.*

Proof Let $(s,s',R) \in (\cong)$ and $s,s';\cdot \vdash e\ R^{\mathsf{cxt}}\ e' : T$. We will show that

$$\begin{array}{l}\forall t,w.\ \ s \vdash e \Downarrow (t)\,w \\ \qquad\ \Longrightarrow\ \exists t',w',Q.\ \ s' \vdash e' \Downarrow (t')\,w' \\ \qquad\qquad\qquad \wedge\ (T = o)\ \Longrightarrow\ (w = w') \\ \qquad\qquad\qquad \wedge\ s{\oplus}t, s'{\oplus}t';\cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T \\ \qquad\qquad\qquad \wedge\ (s{\oplus}t, s'{\oplus}t', Q) \in (\cong) \\ \qquad\qquad\qquad \wedge\ R \subseteq Q\end{array}$$

By the definition of pre-adequate annotated relations (Definition 4.5) and the totality of evaluation (Lemma 2.5), it suffices to show that

$$\begin{array}{rl} \forall t,t',w,w'. & s\vdash e\Downarrow(t)\,w \\ \wedge & s'\vdash e'\Downarrow(t')\,w' \\ \Longrightarrow & \exists Q.\ \ s\oplus t,s'\oplus t';\cdot\vdash w\;Q^{\mathsf{cxt}}\;w':T \\ & \wedge\ (s\oplus t,s'\oplus t',Q)\in(\cong) \\ & \wedge\ R\subseteq Q \end{array}$$

Let $s\vdash e\Downarrow(t)\,w$ and $s'\vdash e'\Downarrow(t')\,w'$; we will show that

$$(s\oplus t,s'\oplus t',R\cup\{(w,w')\})\in(\cong).$$

For any $\overline{x,T,v,v'}$, y, T_0, b, and d such that

$$\emptyset;\overline{x:T},y:T_0\vdash d:o\qquad\text{and}\qquad s\oplus t,s'\oplus t';\cdot\vdash\overline{v}\;R\;\overline{v'}:\overline{T}$$

we have

$$\begin{array}{rll} & \exists t_1.\ s\oplus t\vdash d[\overline{v/x},w/y]\Downarrow(t_1)\,b & \\ \Longleftrightarrow & \exists t_1.\ s\vdash\lambda y{:}T.\,d[\overline{v/x}]\;e\Downarrow(t\oplus t_1)\,b & \text{(by properties of evaluation)} \\ \Longleftrightarrow & \exists t_1'.\ s'\vdash\lambda y{:}T.\,d[\overline{v'/x}]\;e'\Downarrow(t'\oplus t_1')\,b & ((s,s',R)\in(\cong)) \\ \Longleftrightarrow & \exists t_1'.\ s'\oplus t'\vdash d[\overline{v'/x},w'/y]\Downarrow(t_1')\,b & \text{(by properties of evaluation.)} \end{array}$$

Therefore, by Definitions 4.5 and 4.6

$$(s\oplus t,s'\oplus t',R\cup\{(w,w')\})\in(\cong)\qquad\square$$

Theorem 4.13 *Pre-adequacy coincides with adequacy:* $(\cong)=(\approx)$.

Proof By Theorem 4.11 we have $(\approx)\subseteq(\cong)$ and by Theorem 4.12 we have $(\cong)\subseteq(\approx)$. $\square$

From the above we conclude that the open extension of adequacy coincides with the standard definition of contextual equivalence.

Theorem 4.14 $(\approx)^\circ=(\equiv)$.

Proof By Theorems 4.8 and 4.13. $\square$

5 Inductive Proofs of Equivalence

We now have a proof method for showing that $s;\overline{x:T}\vdash e\equiv e':T$:

1. Find a set $\mathscr{X}$ containing (s,s,R) such that

$$\begin{array}{c} s,s;\cdot\vdash(\lambda\overline{x{:}T}.\,e)\;R\;(\lambda\overline{x{:}T}.\,e'):\overline{T}\to T \\ \forall n\in s.\ \ s,s;\cdot\vdash n\;R\;n:\nu \end{array}$$

2. show that $\mathscr{X}$ is adequate, and
3. invoke Theorem 4.14 to conclude

$$s;\overline{x:T}\vdash e\equiv e':T.$$

We can always organise the proof of a set $\mathscr{X}$ being adequate (step (2) above) as an *inductive proof* with the following induction hypothesis:

Definition 5.1

$$\begin{array}{l} IH_{\mathscr{X}}(k) \stackrel{\text{def}}{=} \forall (s,s',R) \in \mathscr{X}. \\ \quad \forall e,e',t,w.\ \ s,s';\cdot \vdash e\ R^{\mathsf{cxt}}\ e' : T \\ \quad\quad \wedge\ s \vdash e \Downarrow^k (t)\, w \\ \quad\quad\quad \Longrightarrow\ \exists t',w',Q.\ \ s' \vdash e' \Downarrow (t')\, w' \\ \quad\quad\quad\quad\quad \wedge\ (T = o) \implies (w = w') \\ \quad\quad\quad\quad\quad \wedge\ s \oplus t, s' \oplus t';\cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T \\ \quad\quad\quad\quad\quad \wedge\ (s \oplus t, s' \oplus t', Q) \in \mathscr{X} \\ \quad\quad\quad\quad\quad \wedge\ R \subseteq Q \end{array}$$

The measure of the induction is the size k of the evaluation $s \vdash e \Downarrow^k (t)\, w$. Hence, proving a set of annotated relations $\mathscr{X}$ adequate amounts to proving that for all k, $IH_{\mathscr{X}}(k)$ and $IH_{\mathscr{X}^{-1}}(k)$ hold. For $k = 0$ it is trivial; for $k > 0$ it can be organised as an induction:

$$\boxed{\begin{array}{l} \forall k.\ IH_{\mathscr{X}}(k-1) \implies IH_{\mathscr{X}}(k) \\ \forall k.\ IH_{\mathscr{X}^{-1}}(k-1) \implies IH_{\mathscr{X}^{-1}}(k) \end{array}}$$

6 Deriving Smaller Proof Obligations for Adequate Sets

By using a *proof construction scheme*, as in [22], we factor out the common parts of the two inductions at the end of the previous section, and discover necessary and sufficient proof obligations for adequacy. Thus, we arrive at the following adequacy theorem.

Theorem 6.1 *A set of annotated relations $\mathscr{X}$ is adequate if and only if for all k and all $(s,s',R) \in \mathscr{X}$, assuming that $IH_{\mathscr{X}}(k-1)$ holds, the following conditions hold:*

1. *For all $s,s';\cdot \vdash b\ R\ b' : o$ it must be that $b = b'$.*
2. *For all $s,s';\cdot \vdash \lambda x{:}T_0.\, e\ R\ \lambda x{:}T_0.\, e' : T_0 \to T$, and all $s,s';\cdot \vdash v\ R^{\mathsf{cxt}}\ v' : T_0$, t, and w, such that $s \vdash (\lambda x{:}T_0.\, e)\ v \Downarrow^k (t)\, w$, there exist t', w', and $Q \supseteq R$ such that*

$$\begin{array}{c} s' \vdash (\lambda x{:}T_0.\, e')\ v' \Downarrow (t')\, w' \qquad s \oplus t, s' \oplus t';\cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T \\ (s \oplus t, s' \oplus t', Q) \in \mathscr{X} \end{array}$$

3. *For all $n \notin s$ there exist $n' \notin s'$ and $Q \supseteq R$ such that*

$$s \oplus \{n\}, s' \oplus \{n'\};\cdot \vdash n\ Q\ n' : \nu \qquad (s \oplus \{n\}, s' \oplus \{n'\}, Q) \in \mathscr{X}$$

4. *For all $s,s';\cdot \vdash n_1\ R\ n_1' : \nu$ and $s,s';\cdot \vdash n_2\ R\ n_2' : \nu$*

$$n_1 = n_2 \iff n_1' = n_2'$$

Moreover, the same conditions hold for $\mathscr{X}^{-1}$.

The first condition requires that any annotated relation R in $\mathscr{X}$ is a partial identity at type o and the fourth condition that R is a partial bijection at type ν. These conditions reveal the essential observations of the context at base types.

The second condition requires that (s,s',R)-related functions, applied to (s,s',R^{cxt})-related values, return results that are related in an extended annotated relation $(s\oplus t, s'\oplus t', Q)$ from $\mathscr{X}$, where t and t' are the namesets generated by the two applications. This condition is reminiscent to the condition of applicative bisimulations [2], but asks for functions to be applied to *related* and not identical arguments, a condition which is more characteristic of logical relations.

The third condition requires that $\mathscr{X}$ is closed under allocation of fresh names by the context. This requires related functions to still return related results when applied to arguments that involve arbitrarily many new names, a condition which is again characteristic of Kripke logical relations [12].

Proof (Theorem 6.1) As we argued in Section 5, $\mathscr{X}$ is adequate iff

$$\begin{array}{l}\forall k.\ IH_{\mathscr{X}}(k-1) \implies IH_{\mathscr{X}}(k)\\ \forall k.\ IH_{\mathscr{X}^{-1}}(k-1) \implies IH_{\mathscr{X}^{-1}}(k)\end{array}$$

Hence it suffices to prove that conditions 1–4 are satisfied for $\mathscr{X}$ iff

$$\forall k.\ IH_{\mathscr{X}}(k-1) \implies IH_{\mathscr{X}}(k)$$

and similarly for $\mathscr{X}^{-1}$.

Forward direction: We assume that $\mathscr{X}$ satisfies conditions 1–4 of the theorem and $IH_{\mathscr{X}}(k-1)$, for any k, and consider s,s',R,e,e',t, and w such that

$$(s,s',R)\in\mathscr{X} \qquad s,s';\cdot\vdash e\ R^{\mathsf{cxt}}\ e' : T \qquad s\vdash e\Downarrow^{k}(t)\,w$$

We need to show that there exist t',w', and Q such that

$$\begin{array}{c}s'\vdash e'\Downarrow(t')\,w'\\ (T=o) \implies (w=w')\\ s\oplus t, s'\oplus t';\cdot\vdash w\ Q^{\mathsf{cxt}}\ w' : T\\ (s\oplus t, s'\oplus t', Q)\in\mathscr{X}\\ R\subseteq Q\end{array}$$

By expanding the definition of $(\ )^{\mathsf{cxt}}$ (Definition 4.4) in $s,s';\cdot\vdash e\ R^{\mathsf{cxt}}\ e' : T$, we get that there exist $d,\overline{x},\overline{T},\overline{u}$, and $\overline{u'}$ such that

$$e=d[\overline{u/x]} \qquad e'=d[\overline{u'/x]} \qquad \emptyset;\overline{x:T}\vdash d:T \qquad s,s';\cdot\vdash \overline{u}\ R\ \overline{u'} : \overline{T}$$

We proceed by cases on d. The case where $(d=x_i)$ and $(\emptyset;\overline{x:T}\vdash x_i:o)$ follows by condition 1. The case where $(d=(x_i\,d_1))$, $(\emptyset;\overline{x:T}\vdash x_i:T_1\to T)$, and $(\emptyset;\overline{x:T}\vdash d_1:T_1)$ follows by condition 2 and $IH_{\mathscr{X}}(k-1)$. The case where $(d=\mathtt{new})$ follows by condition 3. The case where $(d=(d_1=d_2))$, $(\emptyset;\overline{x:T}\vdash d_1:\nu)$, and $(\emptyset;\overline{x:T}\vdash d_2:\nu)$ follows by condition 4 and $IH_{\mathscr{X}}(k-1)$. The rest of the cases follow directly by $IH_{\mathscr{X}}(k-1)$.

Reverse direction: We assume that for an all k, $IH_{\mathscr{X}}(k-1)$ holds and for all s, s', R, d, $\overline{x}, \overline{T}, \overline{u}, \overline{u'}, t$, and w such that

$$(s, s', R) \in \mathscr{X} \qquad s, s'; \cdot \vdash \overline{u}\, R\, \overline{u'} : \overline{T} \qquad s \vdash d[\overline{u/x}] \Downarrow^k (t)\, w$$

there exist t', w', and Q such that

$$s' \vdash d[\overline{u'/x}] \Downarrow (t')\, w'$$
$$(T = o) \implies (w = w')$$
$$s \oplus t, s' \oplus t'; \cdot \vdash w\ Q^{\mathsf{cxt}}\ w' : T$$
$$(s \oplus t, s' \oplus t', Q) \in \mathscr{X}$$
$$R \subseteq Q$$

We need to show that $\mathscr{X}$ satisfies conditions 1–4. The first condition follows by considering $(d = x_1)$, $(u_1 = b)$, and $(u'_1 = b')$. The second condition follows by considering $(d = (x_1\ d_2))$, $(u_1 = \lambda x{:}T_0.\, e)$, $(u'_1 = \lambda x{:}T_0.\, e')$, $(d_2[\overline{u/x}] = v)$, and $(d_2[\overline{u'/x}] = v')$. The third condition follows by considering $(d = \texttt{new})$ and the final condition follows by considering $(d = \texttt{if}\ (x_1 = x_2)\ \texttt{then true else false})$, $(u_1 = n_1)$, $(u_2 = n_2)$, $(u'_1 = n'_1)$, and $(u'_2 = n'_2)$. □

7 Examples

Using the preceding theorem, we are able to prove all equivalences in the ν-calculus from [41]. In this section we start with the proof of a straightforward equivalence and then present the proof of the 'hard' equivalence (2) that we gave in the introduction.

7.1 A Simple Example: Local Names

This equivalence demonstrates that the context cannot provide names that are freshly generated and then kept local within a closure.

Theorem 7.1

$$\emptyset; \cdot \vdash \nu n.\, \lambda x{:}\nu.\, (x = n) \quad \equiv \quad \lambda x{:}\nu.\, \texttt{false} : \nu \to o.$$

Proof The theorem concerns the equivalence of two closed terms, but we want to work with closed values. By Theorem 4.2, it suffices to show that the following two closed values are equivalent:

$$N \stackrel{\text{def}}{=} \lambda y{:}o.\, \nu n.\, \lambda x{:}\nu.\, (x = n)$$
$$N' \stackrel{\text{def}}{=} \lambda y{:}o.\, \lambda x{:}\nu.\, \texttt{false}$$

To prove $\emptyset; \cdot \vdash N \equiv N' : o \to \nu \to o$ we need to construct an adequate set of annotated relations, $\mathscr{X}$, such that there exists R with

$$(\emptyset, \emptyset, R) \in \mathscr{X} \qquad \emptyset, \emptyset; \cdot \vdash N\, R\, N' : o \to \nu \to o$$

We start the inductive construction of an adequate $\mathscr{X}$ by the first two rules shown in Figure 4. Rule $\mathscr{X}$-2 fulfils Condition 3 of Theorem 6.1. Conditions 1 and 4 are trivially satisfied.

$$\frac{}{(\emptyset,\emptyset,\{(N,N',o\to\nu\to o)\})\in\mathscr{X}}\ \mathscr{X}\text{-}1$$

$$\frac{(s,s',R)\in\mathscr{X}\qquad n\notin s\qquad n'\notin s'}{(s\oplus\{n\},s'\oplus\{n'\},R\cup\{(n,n',\nu)\})\in\mathscr{X}}\ \mathscr{X}\text{-}2$$

$$\frac{(s,s',R)\in\mathscr{X}\qquad n\notin s}{(s\oplus\{n\},s',R\cup\{(\lambda x{:}\nu.\,(x{=}n),\lambda x{:}\nu.\,\mathtt{false},\nu\to o)\})\in\mathscr{X}}\ \mathscr{X}\text{-}3$$

Fig. 4 Construction of adequate set of annotated relations for proving the equivalence of a simple equivalence in the ν-calculus.

We need to establish Condition 2 of Theorem 6.1 for any $(s,s',R)\in\mathscr{X}$ with $s,s';\cdot\vdash N\ R\ N':o\to\nu\to o$. Let $s,s';\cdot\vdash b\ R^{\mathsf{cxt}}\ b:o$. We have

$$\begin{array}{ll} s\vdash N\ b\Downarrow(\{n\})\,\lambda x{:}\nu.\,(x{=}n) & n\notin s\\ s'\vdash N'\ b\Downarrow(\emptyset)\,\lambda x{:}\nu.\,\mathtt{false} & \end{array}$$

Thus we add Rule $\mathscr{X}$-3 of Figure 4 to the construction of $\mathscr{X}$, so we now have

$$(s\oplus\{n\},s',R\cup\{(\lambda x{:}\nu.\,(x{=}n),\lambda x{:}\nu.\,\mathtt{false},\nu\to o)\})\in\mathscr{X}\qquad(s,s',R)\in\mathscr{X}\qquad n\notin s$$

We now have to check that Condition 2 holds for any $(s\oplus\{n\},s',R)\in\mathscr{X}$ with

$$s\oplus\{n\},s';\cdot\vdash\lambda x{:}\nu.\,(x{=}n)\ R\ \lambda x{:}\nu.\,\mathtt{false}:\nu\to o$$

So, let

$$s\oplus\{n\},s';\cdot\vdash n_0\ R^{\mathsf{cxt}}\ n_0':\nu$$

By the definition of $(\ )^{\mathsf{cxt}}$ we have

$$s\oplus\{n\},s';\cdot\vdash n_0\ R\ n_0':\nu$$

Because (s,s',R) is a typed relation, $n_0\in s$; moreover, because $n\notin s$, we have $n\neq n_0$. Hence

$$\begin{array}{c} s\oplus\{n\}\vdash(\lambda x{:}\nu.\,(x{=}n))\ n_0\Downarrow(\emptyset)\,\mathtt{false}\\ s'\vdash(\lambda x{:}\nu.\,\mathtt{false})\ n_0'\Downarrow(\emptyset)\,\mathtt{false}\\ s\oplus\{n\},s';\cdot\vdash\mathtt{false}\ R^{\mathsf{cxt}}\ \mathtt{false}:o\\ (s\oplus\{n\},s',R)\in\mathscr{X}\end{array}$$

This concludes the proof of adequacy of $\mathscr{X}$, and by Theorem 4.14

$$\emptyset;\cdot\vdash N\equiv N':\nu\to o\qquad\square$$

$$\frac{}{(\emptyset,\emptyset,\{(N,N',o\to(\nu\to o)\to o)\})\in\mathscr{X}}\ \mathscr{X}\text{-1}$$

$$\frac{(s,s',R)\in\mathscr{X}\qquad n\notin s\qquad n'\notin s'}{(s\oplus\{n\},s'\oplus\{n'\},R\cup\{(n,n',\nu)\})\in\mathscr{X}}\ \mathscr{X}\text{-2}$$

$$\frac{(s,s',R)\in\mathscr{X}\qquad \{n_1,n_2\}\cap s=\emptyset}{(s\oplus\{n_1,n_2\},s',R\cup\{(U(n_1,n_2),M',(\nu\to o)\to o)\})\in\mathscr{X}}\ \mathscr{X}\text{-3}$$

$$\frac{(s,s',R)\in\mathscr{X}\qquad s_0\cap s=0}{(s\oplus s_0,s',R)\in\mathscr{X}}\ \mathscr{X}\text{-4}$$

Fig. 5 Construction of the primary adequate set of annotated relations for proving the canonical 'hard' equivalence in the ν-calculus.

7.2 The 'Hard' Equivalence

Here we prove the canonical 'hard' equivalence (2) of the ν-calculus, discussed in the introduction. This has previously only been validated with the use of game semantics [3], and a logical relation designed particularly for this example [41]. Our proof here uses operational semantics and two adequate sets of annotated relations.

Theorem 7.2 *If we define*

$$M\stackrel{\text{def}}{=}\nu n_1.\,\nu n_2.\,U(n_1,n_2)\qquad U(n_1,n_2)\stackrel{\text{def}}{=}\lambda f{:}\nu\to o.\,((f\ n_1)=(f\ n_2))$$
$$M'\stackrel{\text{def}}{=}\lambda f{:}\nu\to o.\,\texttt{true}$$

then $\emptyset;\cdot\vdash M\equiv M':(\nu\to o)\to o$.

This example is tricky because, when the result of evaluating M is applied to an argument f that is supplied by the context, the names n_1 and n_2 are *not* kept entirely private: they are passed as arguments to f. However, the fact that the names cannot be communicated between subsequent applications of f (the language has no store) suffices to ensure that the outermost application of $U(n_1,n_2)$ will return `true`.

In the evaluation tree of the application $f\ n_1$ (and $f\ n_2$), the abstraction $U(n_1,n_2)$ may be applied again to an abstraction g that knows the name n_1 (respectively n_2) and cause the inner application of $U(n_1,n_2)$ to return `false`. Such an example is the application of $U(n_1,n_2)$ to the abstraction

$$F\stackrel{\text{def}}{=}\lambda x{:}\nu.\,(U(n_1,n_2)\ (\lambda y{:}\nu.\,(x=y)))$$

where the inner applications of $U(n_1,n_2)$ will return `false`. Nonetheless, due to symmetry of the evaluation trees of $F\ n_1$ and $F\ n_2$, they both return the same boolean value and thus the overall computation returns `true`.

Proof From Theorem 4.2, it suffices to show that the following two values are equivalent.

$$N\stackrel{\text{def}}{=}\lambda y{:}o.\,\nu n_1.\,\nu n_2.\,U(n_1,n_2)$$
$$N'\stackrel{\text{def}}{=}\lambda y{:}o.\,\lambda f{:}\nu\to o.\,\texttt{true}$$

To prove $(\emptyset,\emptyset;\cdot \vdash N\ R\ N' : o\to(\nu\to o)\to o)$ we will construct *two sets* of annotated relations $\mathscr{X}$ and $\mathscr{Y}$ and prove that both are adequate. The first set $\mathscr{X}$ will be the main set of our proof and, as required by Theorem 4.14, will contain an annotated relation $(\emptyset,\emptyset,R)\in\mathscr{X}$ such that

$$\emptyset,\emptyset;\cdot \vdash N\ R\ N' : o\to(\nu\to o)\to o$$

This set will essentially specify the possible worlds (s,s',R) under which the abstractions $U(n_1,n_2)$ and M' will need to be applied to related arguments and satisfy the second condition of Theorem 6.1. Hence, to prove adequacy of $\mathscr{X}$ we will need to show that in any such possible world (s,s',R), and for any values u and u' of type $(\nu\to o)$ related in this world, the applications $(U(n_1,n_2)\ u)$ and $(M'\ u')$ will both return `true`. As one would expect, the interesting part of this proof is showing that under the nameset s, the applications $(u\ n_1)$ and $(u\ n_2)$ will evaluate to the same boolean constant.

We will prove this by making use of the second set of annotated relations $\mathscr{Y}$. First we establish a correlation between $\mathscr{X}$ and $\mathscr{Y}$: we show that for all worlds $(s,s',R)\in\mathscr{X}$, there exists P such that the world (s,s,P) is in $\mathscr{Y}$ and

$$\forall v,v'.\ \ s,s';\cdot \vdash v\ R\ v' : T \implies s,s;\cdot \vdash v\ P\ v : T$$

This means that when we pick a possible world (s,s',R) from $\mathscr{X}$ in which we perform the applications $(U(n_1,n_2)\ u)$ and $(M'\ u')$, there is a world (s,s,P) in $\mathscr{Y}$ where

$$s,s;\cdot \vdash U(n_1,n_2)\ P\ U(n_1,n_2) : (\nu\to o)\to o$$
$$s,s;\cdot \vdash u\ P^{\mathsf{cxt}}\ u : \nu\to o$$

We will construct $\mathscr{Y}$ in such a way that when $s,s;\cdot \vdash U(n_1,n_2)\ P\ U(n_1,n_2) : (\nu\to o)\to o$ then

$$s,s;\cdot \vdash n_1\ P\ n_2 : \nu \qquad s,s;\cdot \vdash n_2\ P\ n_1 : \nu$$

Therefore we have that the applications of interest are related in P^{cxt}:

$$s,s;\cdot \vdash (u\ n_1)\ P^{\mathsf{cxt}}\ (u\ n_2) : o$$

By proving that $\mathscr{Y}$ is adequate we will prove that these two applications will result to the same boolean constant, which completes the proof of the example.

We start the construction of $\mathscr{X}$ by the first two rules of Figure 5. Rule $\mathscr{X}$-2 ensures Condition 3 of Theorem 6.1. Conditions 1 and 4 are trivially satisfied. We need to establish Condition 2 of Theorem 6.1 for any $(s,s',R)\in\mathscr{X}$ with $s,s';\cdot \vdash N\ R\ N' : o\to(\nu\to o)\to o$. Let $s,s';\cdot \vdash b\ R^{\mathsf{cxt}}\ b : o$. We have

$$s\vdash N\ b\Downarrow(\{n_1,n_2\})\,U(n_1,n_2) \qquad \{n_1,n_2\}\cap s=\emptyset$$
$$s'\vdash N'\ b\Downarrow(\emptyset)\,M'$$

which leads us to add Rule $\mathscr{X}$-3 to the construction of $\mathscr{X}$. Hence we now have

$$(s\oplus\{n_1,n_2\},s',R\cup\{(U(n_1,n_2),M',(\nu\to o)\to o)\})\in\mathscr{X}$$

It remains to establish Condition 2 for any $(s\oplus\{n_1,n_2\},s',R)\in\mathscr{X}$ with $s\oplus\{n_1,n_2\},s';\cdot \vdash U(n_1,n_2)\ R\ M' : (\nu\to o)\to o$. Let

$$s\oplus\{n_1,n_2\},s';\cdot \vdash u_f\ R^{\mathsf{cxt}}\ u'_f : \nu\to o$$
$$s\oplus\{n_1,n_2\}\vdash U(n_1,n_2)\ u_f\Downarrow^k(t)\,w$$

$$\frac{}{(\emptyset,\emptyset,\{(N,N,o\to(\nu\to o)\to o)\})\in\mathscr{Y}}\ \mathscr{Y}\text{-}1$$

$$\frac{(s,s,R)\in\mathscr{Y}\qquad n\notin s}{(s\oplus\{n\},s\oplus\{n\},R\cup\{(n,n,\nu)\})\in\mathscr{Y}}\ \mathscr{Y}\text{-}2$$

$$\frac{\begin{array}{c}(s,s,R)\in\mathscr{Y}\qquad \{n_1,n_2\}\cap s=\emptyset\\ Q=R\cup\{(n_1,n_2,\nu),(n_2,n_1,\nu),(U(n_1,n_2),U(n_1,n_2),(\nu\to o)\to o)\}\end{array}}{(s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\},Q)\in\mathscr{Y}}\ \mathscr{Y}\text{-}3$$

$$\frac{(s,s,R)\in\mathscr{Y}\qquad s_0\cap s=0}{(s\oplus s_0,s\oplus s_0,R)\in\mathscr{Y}}\ \mathscr{Y}\text{-}4$$

Fig. 6 Construction of the auxiliary adequate set of annotated relations for proving the canonical 'hard' equivalence in the ν-calculus.

We need to show that there exist Q, t', and w' such that

$$\begin{array}{rl} s'\vdash M'\ u'_f\Downarrow(t')\,w' & w=w'\\ (s\oplus\{n_1,n_2\}\oplus t,s'\oplus t',Q)\in\mathscr{X} & R\subseteq Q\end{array}$$

But we have

$$s'\vdash M'\ u'_f\Downarrow(\emptyset)\,\mathtt{true}$$

Thus $t'=\emptyset$, $w'=\mathtt{true}$, and $Q=R$. It remains to show that for some t,

$$s\oplus\{n_1,n_2\}\vdash U(n_1,n_2)\ u_f\Downarrow(t)\,\mathtt{true}\qquad(s\oplus\{n_1,n_2\}\oplus t,s',R)\in\mathscr{X}$$

We add Rule $\mathscr{X}$-4 to the construction of $\mathscr{X}$ in Figure 5 which discharges the second proof obligation. It only remains to show that the first application evaluates to `true`. By the properties of evaluation, it suffices to show that there exist b, t_1, and t such that

$$\begin{array}{c}s\oplus\{n_1,n_2\}\vdash u_f\ n_1\Downarrow(t_1)\,b\\ s\oplus\{n_1,n_2\}\oplus t_1\vdash u_f\ n_2\Downarrow(t)\,b\end{array}$$

or, from Lemma 2.3, it suffices to show that there exist b, t_1, and t such that

$$\begin{array}{c}s\oplus\{n_1,n_2\}\vdash u_f\ n_1\Downarrow(t_1)\,b\\ s\oplus\{n_1,n_2\}\vdash u_f\ n_2\Downarrow(t)\,b\end{array}$$

We show this by a *second* use of our proof technique: constructing an auxiliary adequate set $\mathscr{Y}$ of annotated relations such that there exists a relation P with

$$\begin{array}{c}s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\};\cdot\vdash(u_f\ n_1)\ P^{\mathsf{cxt}}\ (u_f\ n_2):o\\ (s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\},P)\in\mathscr{Y}.\end{array}$$

The construction of $\mathscr{Y}$ is shown in Figure 6. A correlation between the two sets of annotated relations $\mathscr{X}$ and $\mathscr{Y}$ is established by the following lemma:

Lemma 7.3 *For all $(s,s',R) \in \mathscr{X}$, there exists P such that*

$$(s,s,P) \in \mathscr{Y}$$
$$\forall v,v'.\ s,s';\cdot \vdash v\ R\ v' : T \implies s,s;\cdot \vdash v\ P\ v : T$$

Proof We proceed by induction on the construction of $\mathscr{X}$.

CASE $\mathscr{X}$-1: $\dfrac{}{(\emptyset,\emptyset,\{(N,N',o\to(\nu\to o)\to o)\}) \in \mathscr{X}}$
This case is trivial because, by $\mathscr{Y}$-1:

$$(\emptyset,\emptyset,\{(N,N,o\to(\nu\to o)\to o)\}) \in \mathscr{Y}.$$

CASE $\mathscr{X}$-2: $\dfrac{(s,s',R) \in \mathscr{X} \qquad n \notin s \qquad n' \notin s'}{(s\oplus\{n\},s'\oplus\{n'\},R\cup\{(n,n',\nu)\}) \in \mathscr{X}}$
By the induction hypothesis at $(s,s',R) \in \mathscr{X}$ we get that there exists P such that

$$(s,s,P) \in \mathscr{Y} \tag{4}$$
$$\forall v,v'.\ s,s';\cdot \vdash v\ R\ v' : T \implies s,s;\cdot \vdash v\ P\ v : T \tag{5}$$

By $\mathscr{Y}$-2 and (4) we get that

$$(s\oplus\{n\},s\oplus\{n\},P\cup\{(n,n,\nu)\}) \in \mathscr{Y}$$

Now let $s\oplus\{n\},s'\oplus\{n'\};\cdot \vdash v\ (R\cup\{(n,n',\nu)\})\ v' : T$. We have two cases:

1. $s,s';\cdot \vdash v\ R\ v' : T$. By (5) we get $s,s;\cdot \vdash v\ P\ v : T$, and thus
$$s\oplus\{n\},s\oplus\{n\};\cdot \vdash v\ (P\cup\{(n,n,\nu)\})\ v : T$$
2. $v = n$, $v = n'$, $T = \nu$. It is immediate that
$$s\oplus\{n\},s\oplus\{n\};\cdot \vdash n\ (P\cup\{(n,n,\nu)\})\ n : \nu.$$

CASE $\mathscr{X}$-3: $\dfrac{(s,s',R) \in \mathscr{X} \qquad \{n_1,n_2\}\cap s = \emptyset}{(s\oplus\{n_1,n_2\},s',R\cup\{(U(n_1,n_2),M',(\nu\to o)\to o)\}) \in \mathscr{X}}$
By the induction hypothesis at $(s,s',R) \in \mathscr{X}$ we get that there exists P such that

$$(s,s,P) \in \mathscr{Y} \tag{6}$$
$$\forall v,v'.\ s,s';\cdot \vdash v\ R\ v' : T \implies s,s;\cdot \vdash v\ P\ v : T \tag{7}$$

Let

$$Q = P\cup\{(n_1,n_2,\nu),(n_2,n_1,\nu),(U(n_1,n_2),U(n_1,n_2),(\nu\to o)\to o)\}$$

By $\mathscr{Y}$-3 and (6) we get that

$$(s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\},Q) \in \mathscr{Y}$$

Let

$$s\oplus\{n_1,n_2\},s';\cdot \vdash v\ (R\cup\{(U(n_1,n_2),M',(\nu\to o)\to o)\})\ v' : T$$

We have two cases:

1. $s,s';\cdot \vdash v\ R\ v' : T$. By (7) we get $s,s;\cdot \vdash v\ P\ v : T$, and thus
$$s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\};\cdot \vdash v\ Q\ v : T$$
2. $v = U(n_1,n_2)$, $v' = M'$, $T = (\nu \to o)\to o$. It is immediate that
$$s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\};\cdot \vdash U(n_1,n_2)\ Q\ U(n_1,n_2) : (\nu\to o)\to o.$$

CASE $\mathscr{X}$-4: $\dfrac{(s,s',R)\in\mathscr{X} \qquad s_0\cap s = 0}{(s\oplus s_0,s',R)\in\mathscr{X}}$

Immediate by the induction hypothesis at $(s,s',R)\in\mathscr{X}$ and $\mathscr{Y}$-4. □

(Continuing proof from page 19.) We have that
$$\begin{gathered}(s\oplus\{n_1,n_2\},s',R)\in\mathscr{X}\\ s\oplus\{n_1,n_2\},s';\cdot \vdash U(n_1,n_2)\ R\ M' : (\nu\to o)\to o\\ s\oplus\{n_1,n_2\},s';\cdot \vdash u_f\ R^{\mathsf{cxt}}\ u'_f : \nu\to o\end{gathered}$$

Therefore, by Lemma 7.3, there exists P such that
$$\begin{gathered}(s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\},P)\in\mathscr{Y}\\ s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\};\cdot \vdash U(n_1,n_2)\ P\ U(n_1,n_2) : (\nu\to o)\to o\end{gathered}$$

By construction, the value relations in the tuples of $\mathscr{Y}$ relate names to names, the term N to itself (by rule $\mathscr{Y}$-1), and values of the form $U(n,m)$ to themselves for any n and m (by rule $\mathscr{Y}$-3). Hence, by an easy induction on the construction of $\mathscr{Y}$, we derive that because $U(n_1,n_2)$ is related to itself we also have
$$\begin{gathered}s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\};\cdot \vdash n_1\ P\ n_2 : \nu\\ s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\};\cdot \vdash n_2\ P\ n_1 : \nu\end{gathered}$$

By construction of $\mathscr{X}$, there exist $\overline{x}$, $\overline{y}$, d, $\overline{n,n'}$, and $\overline{n_1,n_2}$ such that
$$\begin{gathered}\emptyset;\overline{x:(\nu\to o)\to o},\overline{y:\nu}\vdash d:\nu\to o\\ s\oplus\{n_1,n_2\},s';\cdot \vdash \overline{U(n_1,n_2)}\ R\ \overline{M'} : \overline{(\nu\to o)\to o}\\ s\oplus\{n_1,n_2\},s';\cdot \vdash \overline{n}\ R\ \overline{n'} : \overline{\nu}\\ u_f = d[\overline{U(n_1,n_2)/x},\overline{n/y}]\\ u'_f = d[\overline{M'/x},\overline{n'/y}]\end{gathered}$$

Thus, by Lemma 7.3,
$$\begin{gathered}s\oplus\{n_1,n_2\};\cdot \vdash \overline{U(n_1,n_2)}\ P\ \overline{U(n_1,n_2)} : \overline{(\nu\to o)\to o}\\ s\oplus\{n_1,n_2\};\cdot \vdash \overline{n}\ P\ \overline{n} : \overline{\nu}\end{gathered}$$

and therefore
$$s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\};\cdot \vdash u_f\ P^{\mathsf{cxt}}\ u_f : \nu\to o$$

From the above we get
$$s\oplus\{n_1,n_2\},s\oplus\{n_1,n_2\};\cdot \vdash (u_f\ n_1)\ P^{\mathsf{cxt}}\ (u_f\ n_2) : o$$

It remains to show that $\mathscr{Y}$ is adequate by showing that it satisfies the conditions of Theorem 6.1. $\mathscr{Y}$ trivially satisfies Conditions 1 and 4 of Theorem 6.1. Condition 3 of the theorem is fulfilled by Rule $\mathscr{Y}$-2. It remains to prove Condition 2 for all related abstractions.

Let $(s,s',R) \in \mathscr{Y}$. It is the case that $\mathscr{Y}$ is the identity modulo the crosswise renaming of some names. Thus $s = s'$ and for some names $\overline{n_1,n_2}$ and values $\overline{v}$ we have that $R = \{\overline{(v,v)}, \overline{(n_1,n_2)}, \overline{(n_2,n_1)}\}$.

Therefore, we consider $(s,s,R) \in \mathscr{Y}$, and prove Condition 2 for the following cases:

CASE 1: $s,s;\cdot \vdash N\ R\ N : o \to (\nu \to o) \to o$

Let $s,s;\cdot \vdash b\ R^{\mathsf{cxt}}\ b : o$ and $s \vdash N\ b \Downarrow^k (\{n_1,n_2\})\,U(n_1,n_2)$. By Rule $\mathscr{Y}$-3, there exists Q such that

$$Q = R \cup \{(n_1,n_2,\nu),(n_2,n_1,\nu),(U(n_1,n_2),U(n_1,n_2),(\nu\to o)\to o)\}$$
$$s \vdash N\ b \Downarrow (\{n_1,n_2\})\,U(n_1,n_2)$$
$$s\oplus\{n_1,n_2\}, s\oplus\{n_1,n_2\};\cdot \vdash U(n_1,n_2)\ Q^{\mathsf{cxt}}\ U(n_1,n_2) : (\nu\to o)\to o$$
$$(s\oplus\{n_1,n_2\}, s\oplus\{n_1,n_2\}, Q) \in \mathscr{Y}$$

CASE 2: $s,s;\cdot \vdash U(n_1,n_2)\ R\ U(n_1,n_2) : (\nu \to o) \to o$

Let $s,s;\cdot \vdash v\ R^{\mathsf{cxt}}\ v' : \nu \to o$ and $s \vdash U(n_1,n_2)\ v \Downarrow^k (t)\,b$. By the properties of evaluation we have $t = s_1 \oplus s_2$ and

$$s \vdash v\ n_1 \Downarrow^{k-1} (s_1)\,b_1 \tag{8}$$
$$s\oplus s_1 \vdash v\ n_2 \Downarrow^{k-1} (s_2)\,b_1 \tag{9}$$

By Lemma 2.4,

$$s \vdash v\ n_2 \Downarrow^{k-1} (s_2)\,b_1 \tag{10}$$

Because

$$s,s;\cdot \vdash (v\ n_1)\ R^{\mathsf{cxt}}\ (v'\ n_2) : o \qquad s,s;\cdot \vdash (v\ n_2)\ R^{\mathsf{cxt}}\ (v'\ n_1) : o$$

and by $IH_{\mathscr{Y}}(k-1)$, (8), and (10) we get that there exist Q_1 and Q_2 such that

$$s \vdash v'\ n_2 \Downarrow (s_1')\,b_1' \quad s\oplus s_1, s\oplus s_1';\cdot \vdash b_1\ Q_1{}^{\mathsf{cxt}}\ b_1' : o \quad (s\oplus s_1, s\oplus s_1', Q_1) \in \mathscr{Y}$$
$$s \vdash v'\ n_1 \Downarrow (s_2')\,b_2' \quad s\oplus s_2, s\oplus s_2';\cdot \vdash b_2\ Q_2{}^{\mathsf{cxt}}\ b_2' : o \quad (s\oplus s_2, s\oplus s_2', Q_2) \in \mathscr{Y}$$

By the definition of $(\ )^{\mathsf{cxt}}$ we get that $b_1 = b_1'$ and $b_2 = b_2'$. Because each relation in $\mathscr{Y}$ is annotated with identical stores, $s_1 = s_1'$ and $s_2 = s_2'$. Therefore

$$s \vdash v'\ n_2 \Downarrow (s_1)\,b_1$$
$$s \vdash v'\ n_1 \Downarrow (s_2)\,b_2$$

By (9) we get that $s_1 \cap s_2 = s \cap s_2 = \emptyset$. Thus, by Lemma 2.3 we get

$$s\oplus s_2 \vdash v'\ n_2 \Downarrow (s_1)\,b_1$$

and by the properties of evaluation,

$$s \vdash ((v'\ n_1) = (v'\ n_2)) \Downarrow (s_1 \oplus s_2)\,b$$

Furthermore,

$$s\oplus s_1, s\oplus s_1;\cdot \vdash b\ R^{\mathsf{cxt}}\ b : o$$

and by Rule $\mathscr{Y}$-4 we get

$$(s\oplus s_1, s\oplus s_1, R) \in \mathscr{Y}.$$

Therefore, $\mathscr{Y}$ is adequate. □

8 The Formalization in Coq

As should now be clear, both the metatheory and application of our bisimulation, although elementary, involve lengthy and rather fiddly calculations, the correctness of some of which one could be forgiven for doubting. We have formalised the semantics of the ν-calculus and the soundness[2] (though not the completeness) of our theory in the Coq theorem prover [8]. We have also formalized the full proofs of the example equivalences presented in the previous section.

There are still two axioms in our development, concerning well known basic properties of ν-calculus evaluation that are proved in Stark's thesis [41]. These are the determinacy lemma (Lemma 2.6) and the totality lemma (Lemma 2.5). These are entirely standard results and proofs, the mechanisation of which is not especially interesting.

8.1 Semantics of the ν-calculus

There has been much recent research effort expended on reducing the pain of doing mechanised reasoning about syntax involving binders, most notably under the umbrella of the POPLmark challenge [5]. We were pleased to find that this effort is paying off: our formalisation uses a Coq framework for 'locally nameless' reasoning about binding due to Aydemir et al. [6, 10], which worked very well.

The locally nameless style uses de Bruijn indices for bound identifiers and names for free variables. The benefit of this representation is that each alpha equivalence class has a unique representation. A further feature of the framework is the use of cofinite quantification for free variables; the definitions and tactics provided by Aydemir et al. make it very convenient to generate fresh variable names whenever they are required in proofs.

Following this framework we define an inductive set of *pre-terms* that contains the encodings of all valid terms of the ν-calculus, as well as some invalid ones (e.g. terms with wrong de Bruijn indices):

```
Inductive trm : Set :=
  ...
  | bvar : nat -> trm
  | fvar : var -> trm
  | abs : typ -> trm -> trm
```

This set of pre-terms is sufficient for many of our lemmas, usually the ones that require induction over terms. For others, as well as for the definition of the typing relation, one needs to exclude the illegal terms, which is done by the following inductive predicate:

```
Inductive term : trm -> Prop :=
  ...
  | term_var : forall x, term (fvar x)
  | term_nam : forall (n : nam), term (name n)
  | term_abs : forall L t1 U,
      (forall x, x \notin L -> term (t1 ^ x))
      -> term (abs U t1)
```

Top-level de Bruijn indices are not valid terms; they can only appear under binders. Even then there should not be any dangling indices. The rule for abstractions excludes such terms.

[2] We prove soundness with respect to the characterization of contextual equivalence given by the Context Lemma of [41].

It states that the abstraction is valid when its body, with all references to the abstraction's binder replaced with a fresh variable (`t1 ^ x`), is a valid term. Freshness here is expressed by requiring to provide a finite set of names, `L`, for which all names *not* in that set prove the premise. This *co-finite quantification* establishes stronger induction hypotheses than just requiring `x` to be disjoint from the free variables in `t1`. A similar co-finite quantification is used in defining the typing relation:

```
Inductive typing: nameset -> env -> trm -> typ -> Prop:=
  ...
  | typing_abs: forall L s E U T t1,
      (forall x, x \notin L
      ->  (typing s  (E & x ~ U)  (t1 ^ x)  T))
      -> typing s E (abs U t1) (arrow U T)
```

Here `E & E'` concatenates two environments (or substitutions), and `x ~ U` is the singleton environment that binds `x` to the type `U`.

For our formalisation of bisimulations we needed multiple substitutions, which we got by instantiating the polymorphic library for environments from [6,10] to give finite maps from identifiers to `trm`s and then defining a fold function to actually apply the substitution.

8.2 Relations

We encode typed relations as sets of tuples of nameset, type environment, terms, and type:

```
Definition TRel := nameset -> env -> trm -> trm -> typ -> Prop.
```

We similarly encode generalised typed relations that contain two namesets.

```
Definition GTRel := nameset -> nameset -> env -> trm -> trm -> typ -> Prop.
```

An annotated relation of Section 4 is encoded as a generalised typed relation where all tuples have the same two namesets, empty type environments, and the terms are values of the type in the tuple, under the corresponding namesets. The following predicate encodes these conditions:

```
Definition isAnnotRel (R : GTRel) : Prop :=
  trcRel R
  /\ (exists s1, exists s2, nonempty R s1 s2 empty)
  /\ (forall s1 s2 E t1 t2 T,
       (R s1 s2 E t1 t2 T) -> (can s1 t1 T) /\ (can s2 t2 T))
  /\ (forall s1 s2 E t1 t2 T,
       forall s1' s2' E' t1' t2' T',
         (R s1 s2 E t1 t2 T /\ R s1' s2' E' t1' t2' T')
         -> (s1 = s1') /\ (s2 = s2')).
```

The first conjunct `trcRel R` encodes the requirement that `R` is type-respecting and contains only closed terms. The third conjunct encodes the requirement that all terms in `R` are values (`can`onical forms), and the last conjunct that all tuples in `R` contain the same two namesets. The second conjunct requires that `R` contains at least one tuple with namesets `s1` and `s2` and is a way of expressing that `R` is always annotated with two namesets.

We encode Context Closure of `GTRel`s (Definition 4.4) in two parts. First we construct the $[\overline{v/x}]$ and $[\overline{v'/x}]$ of Definition 4.4 by defining an inductive relation on 'synchronised' environments and substitutions containing closed expressions from a value relation R.

```
Inductive InSync (R:GTRel) (s1 s2:nameset)
      : env -> substitution -> substitution -> Prop :=
  | insync_empty:
      nonempty R s1 s2 empty
        -> InSync R s1 s2 empty empty empty
  | insync_push:
      forall E sub1 sub2 x T t1 t2,
        InSync R s1 s2 E sub1 sub2
        -> R s1 s2 empty t1 t2 T
        -> closed_subst (sub1 & x ~ t1)
        -> closed_subst (sub2 & x ~ t2)
        -> InSync R s1 s2 (E & x ~ T) (sub1 & x ~ t1)
                                      (sub2 & x ~ t2).
```

For a relation `R` annotated with namesets s and s', the empty environment and the empty substitutions are synchronised. When `E`, `sub1`, and `sub2` are synchronised under the relation `R`, and the stores `s1` and `s2`, then their extension with a single mapping from a variable `x` to, respectively, a type `T`, a term `t1`, and a term `t2` from `R` is also synchronised. The predicate `closed_subst` ensures that the resulting substitutions are valid. `R` is normally type-respecting, thus the constructed `sub1` and `sub2` can be used to close any term typeable under `E`.

We then define an operation `substClosure` that combines two relations, one for contexts (`R`) and one for terms (`Q`), into a new relation using substitutions. By giving the identity relation as the first argument and `R` as the second, one obtains the context closure R^{cxt}. The definition of `substClosure` is as follows:

```
Definition substClosure (R:GTRel) (Q:GTRel) : GTRel :=
  fun (s1 s2:nameset) (E:env) (t1 t2:trm) (T:typ) =>
    (E = empty)
    /\ (exists sr1, exists sr2, exists sq1, exists sq2,
         s1 = (sr1 (U) sq1)
         /\ s2 = (sr2 (U) sq2)
         /\ (exists sub1, exists sub2, exists td1,
             exists td2, exists E,
              R sr1 sr2 E td1 td2 T
              /\ InSync Q sq1 sq2 E sub1 sub2
              /\ t1 = <[ sub1 ]> td1
              /\ t2 = <[ sub2 ]> td2)).
```

where `s1 (U) s2` is the syntax for union of namesets. This construction unions the namesets from the two relations, but in the case of R^{cxt}, defined to be `(substClosure IdCxtRel R)`, `sr1` and `sr2` are always empty, thus all names come from the second relation. Here `IdCxtRel` is the identity type-respecting `GTRel` with empty namesets and non-empty typing environments:

```
Definition IdRel : GTRel :=
  fun (s1 s2 : nameset) (E : env) (t1 t2 : trm) (T : typ) =>
    (s1 ; E |= t1 ~: T)  /\  (s2 ; E |= t2 ~: T)
    /\ (t1 = t2)  /\  (s1 = s2).

Definition IdCxtRel : GTRel :=
  fun (s1 s2 : nameset) (E : env) (t1 t2 : trm) (T : typ) =>
    emptyns = s1  /\  emptyns = s2
    /\ IdRel emptyns emptyns E t1 t2 T.
```

The proof of soundness, as well as the proofs for particular equivalences, are fairly lengthy, but manageable. Having made the essential representation choices for terms and relations, as sketched above, the formalization broadly follows an (unusually pedantic) on-paper development. The line counts of different sections of the Coq development are currently as follows:

Section	Lines
Library from UPenn	3148
Semantics, general lemmas, multiple substitutions	3188
Infrastructure about relations	2132
Soundness proof	2301
Simple example	811
Hard example	2740
Total	14320

9 Related Work on Bisimulations and Equivalence

Bisimulation originated as a technique to characterize the behavior of non-deterministic systems [31,15,16]. Abramsky [2] adapted this idea to deterministic languages, creating what is known as *applicative bisimulations*, and used it to reason about an untyped lazy lambda-calculus. Gordon and Rees [14] applied applicative bisimilarity to one of the stateless, typed, object calculi of Abadi and Cardelli [1] and proved that it coincides with contextual equivalence. Applicative bisimulations have also been used in continuation-passing style languages to prove contextual equivalence. For example, Tiuryn and Wand [45] presented a continuation-passing model of an untyped lambda-calculus with input and output, and proved that applicative approximation coincides with contextual approximation. Recently, Koutavas, Levy and Sumii [20] showed that the simple condition of applicative bisimulations at function type is unsound for sequential higher-order languages with local state or existential types.

Sumii and Pierce [44,43] made possible the use of bisimulations in sequential higher-order languages with existential types and dynamic sealing of values. They replaced a single bisimulation with a set of partial bisimulations, each corresponding to a "world", representing the conditions of knowledge, or the state, in which it holds. Similar ideas have previously been used in process calculi (e.g., [11,32]) and suggested for imperative languages [25]. Their method, as presented, has limited applicability when dealing with some equivalences of higher-order procedures [27].

Koutavas and Wand, whose work we continue here, extended Sumii and Pierce's bisimulations to a lambda calculus with general store, an object calculus, and a class-based calculus [22,21,23] where parts of the store are hidden from the context. They used an up-to context technique [35,36,39] to account for large parts of the relations and reduce their size, and introduced an induction hypothesis in the definition that can be used to immediately reason about smaller related computations, including computations that "leak" from the context into the procedures; a similar suggestion of conditions using up-to context and induction was originally made by Sumii and Pierce [44] but was not developed in detail.

Recently, Sangiorgi, Kobayashi, and Sumii [38,37] presented an alternative formulation of Sumii-Pierce-Koutavas-Wand bisimulations dubbed *environmental bisimulations*. The development starts by defining a bare-bones bisimulation à la Sumii and Pierce on a small-step semantics and then builds on top of that a number of up-to techniques. Their formulation uses an up-to expansion and reduction technique instead of the induction hypotheses we use here. Moreover, due to the use of small-step semantics, their technique can be used to characterize reduction barbed congruence in concurrent languages, where the branching structure of the terms is important. The big-step bisimulations we use here and environmental bisimulations seem to be equally effective in proofs of equivalence for the languages on which they have been defined.

Also recently, Støvring and Lassen [42] presented an *eager normal form bisimulation* for a language with control operators and general references. This is an extension of previous work from Lassen [26], where, roughly, two expressions are bisimilar if they have bisimilar normal forms. They make use of possible-worlds relations, similar to the ones we use here.

Of course, there is also a vast literature on contextual equivalence and full abstraction using techniques other than bisimulation, most notably game semantics and various kinds of logical relation (defined over an operational or a denotational semantics). We have already mentioned much of the previous work on applying such approaches to modelling and reasoning the ν-calculus in the introduction. Game semantics have provided fully abstract models of the ν-calculus [3,46], though the precise nature of the connection between such models and Kripke-style relations of the sort used here seems both unclear and deserving of further investigation. It would also be interesting to see if game-based techniques for fully automated equivalence-checking [17] could be applied to tricky ν-calculus equivalences.

Stark [41,40] gives an adequate model of ν-calculus in the functor category $Set^{\mathscr{I}}$, interpreting types as 'variable sets', indexed by a finite set of generated names. The type of names is interpreted by the inclusion functor $\mathscr{I} \hookrightarrow Set$, with generation of fresh names accounted for via a computational monad [28], δ on $Set^{\mathscr{I}}$, the functor part of which is given by $\delta X(n) = X(n+1)$. These functor category models can then be refined by parametric logical relations to get closer to (but not, so far, provably achieve) full abstraction. Similar uses of functor categories were previously made in giving models for local variables in Algol-like languages [34,30], and these were also refined with logical relations [29] to yield proof techniques that could cope with 'tricky' equivalences involving encapsulated state; the 'Meyer-Sieber examples' being a canonical list [27].

The full subcategory of $Set^{\mathscr{I}}$ with pullback-preserving functors as objects is equivalent to the category of *nominal sets* (also the Schanuel topos). Objects of this category are sets X equipped with a permutation action $perm(A) \times X \to X$ for a countably infinite set A of atoms (names), such that every element has *finite support* (a finite subset of A such that the element is fixed by every permutation that fixes the subset). The fully-abstract game semantics of Abramsky et al. [3] is built over nominal sets, which have also been used in modelling references in ML-like languages [7,9]. Nominal sets have been extensively investigated and exploited in the context of formal manipulation of syntax with variable binding [13,48], an issue that arose in our Coq formalization, and which we addressed by using the 'locally nameless' library [6,10].

The current state of the art in non-bisimulation reasoning about contextual equivalence in ML-like languages uses quite sophisticated step-indexed Kripke logical relations defined directly over an operational semantics [12]. Step-indexing [4], which has been widely used in recent years, involves stratifying predicates and relations by a number of evaluation steps. Whilst this may seem superficially similar to the use we make of sizes of evaluation trees in inductive proofs of adequacy (Section 5), step-indexing is really a technical device to deal with various forms of circularity that arise when dealing with recursion, recursive types, higher-order store or recursively-defined possible worlds. None of these issues are relevant for the ν-calculus and steps do not occur in our definition of Adequate sets of relations, nor in the construction of such sets, so these two uses of the natural numbers do not really seem to be connected. However, there *are* many suggestive similarities between the different approaches to reasoning about generativity and encapsulation (such as the explicitly game-like structure of the parameters in the logical relations of Dreyer et al. [12], and the Kripke-like structure of our bisimulations noted in Section 6), and a better understanding of their relationships is long overdue. Recently, relation transition systems [18] have been proposed

as a method for proving equivalence of ML-like programs, which combines aspects of the two techniques.

10 Conclusions

We have introduced a bisimulation for the ν-calculus that is both sound and complete for contextual equivalence, and can be used in practice to establish contextual equivalences that were previously only provable in rather sophisticated game semantic models or by logical relations designed specifically for these examples. Moreover we formalized the relation, its soundness and the proofs of these equivalences in Coq.

The Coq development is a little on the large side, perhaps leading one to question the viability of this technology. However, there is no use of automation beyond that in the library from UPenn and the majority of the development was carried out in around 3 months by someone with no previous experience of mechanical theorem proving. The framework and library for locally nameless reasoning was extremely useful. We remain convinced of the value of mechanizing this style of reasoning, not just for metatheory but also for specific examples, which tend to involve long error-prone calculations that are inherently less interesting than those required to establish general facts about the language.

Acknowledgments We are grateful to Nikos Tzevelekos and the anonymous reviewers for pointing out errors in earlier versions of the paper, and for their many helpful comments and suggestions.

References

1. Abadi, M., Cardelli, L.: A Theory of Objects. Springer-Verlag, Berlin, Heidelberg, and New York (1996)
2. Abramsky, S.: The lazy lambda calculus. In: D.A. Turner (ed.) Research Topics in Functional Programming, pp. 65–116. Addison-Wesley, Boston, MA, USA (1990)
3. Abramsky, S., Ghica, D.R., Murawski, A.S., Ong, C.H.L., Stark, I.D.B.: Nominal games and full abstraction for the nu-calculus. In: Proc. 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004), pp. 150–159. IEEE Computer Society, Washington, DC, USA (2004)
4. Appel, A.W., McAllester, D.: An indexed model of recursive types for foundational proof-carrying code. ACM Transactions on programming languages and systems (TOPLAS 2001) **23**(5), 657–683 (2001)
5. Aydemir, B.E., Bohannon, A., Fairbairn, M., Foster, J.N., Pierce, B.C., Sewell, P., Vytiniotis, D., Washburn, G., Weirich, S., Zdancewic, S.: Mechanized metatheory for the masses: The POPLmark Challenge. In: Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics (TPHOLs), *Lecture Notes in Computer Science*, vol. 3603. Springer-Verlag (2005)
6. Aydemir, B.E., Charguéraud, A., Pierce, B.C., Pollack, R., Weirich, S.: Engineering formal metatheory. In: G.C. Necula, P. Wadler (eds.) Proceeding of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), pp. 3–15. ACM (2008)
7. Benton, N., Leperchey, B.: Relational reasoning in a nominal semantics for storage. In: Proc. 7th International Conference on Typed Lambda Calculi and Applications (TLCA 2005), *Lecture Notes in Computer Science*, vol. 3461, pp. 86–101. Springer, Berlin, Heidelberg, and New York (2005)
8. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer-Verlag (2004)
9. Bohr, N., Birkedal, L.: Relational reasoning for recursive types and references. In: N. Kobayashi (ed.) Proc. 4th Asian Symposium on Programming Languages and Systems (APLAS 2006), *Lecture Notes in Computer Science*, vol. 4279, pp. 79–96. Springer, Berlin, Heidelberg, and New York (2006)
10. Charguéraud, A.: The locally nameless representation. Journal of Automated Reasoning pp. 1–46 (2011). 10.1007/s10817-011-9225-2
11. Deng, Y., Sangiorgi, D.: Towards an algebraic theory of typed mobile processes. In: Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP 2004), *Lecture Notes in Computer Science*, vol. 3142, pp. 445–456. Springer, Berlin, Heidelberg, and New York (2004)

12. Dreyer, D., Neis, G., Birkedal, L.: The impact of higher-order state and control effects on local relational reasoning. Journal of Functional Programming **22**(Special Issue 4-5), 477–528 (2012). (Extended abstract appeared in ICFP 2010.)
13. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing **13**, 341–363 (2002)
14. Gordon, A.D., Rees, G.D.: Bisimilarity for a first-order calculus of objects with subtyping. In: Proc. 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 1996), pp. 386–395. ACM, New York, NY, USA (1996)
15. Hennessy, M., Milner, R.: On observing nondeterminism and concurrency. In: J.W. de Bakker, J. van Leeuwen (eds.) Proc. 7th International Colloquium Automata, Languages and Programming (ICALP 1980), *Lecture Notes in Computer Science*, vol. 85, pp. 299–309. Springer, Berlin, Heidelberg, and New York (1980)
16. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. Journal of the ACM **32**, 137–161 (1985)
17. Hopkins, D., Ong, C.H.L.: HOMER: A higher-order observational equivalence model checker. In: Computer Aided Verification, 21st International Conference (CAV 2009), *LNCS*, vol. 6174. Springer (2009)
18. Hur, C.K., Dreyer, D., Neis, G., Vafeiadis, V.: The marriage of bisimulations and Kripke logical relations. In: Proc. 39th ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL 2012), pp. 59–72. ACM, New York, NY, USA (2012)
19. Jeffrey, A., Rathke, J.: Towards a theory of bisimulation for local names. In: Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS). IEEE (1999)
20. Koutavas, V., Levy, P.B., Sumii, E.: From applicative to environmental bisimulation. Electronic Notes in Theoretical Computer Science **276**(0), 215 – 235 (2011). Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVII)
21. Koutavas, V., Wand, M.: Bisimulations for untyped imperative objects. In: P. Sestoft (ed.) Proc. 15th European Symposium on Programming (ESOP 2006), Programming Languages and Systems, *Lecture Notes in Computer Science*, vol. 3924, pp. 146–161. Springer-Verlag, Berlin, Heidelberg, and New York (2006)
22. Koutavas, V., Wand, M.: Small bisimulations for reasoning about higher-order imperative programs. In: Proc. 33rd ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL 2006), pp. 141–152. ACM Press, New York, NY, USA (2006)
23. Koutavas, V., Wand, M.: Reasoning about class behavior. Appeared in FOOL/WOOD 2007 Workshop (2007)
24. Laird, J.: A game semantics of names and pointers. In: Foundations of Software Science and Computation Structures (FoSSaCS), *Lecture Notes in Computer Science*, vol. 2987. Springer-Verlag (2004)
25. Lassen, S.B.: Bisimulation up to context for imperative lambda calculus (1998). Part of a presentation "Bisimulation up to Context for Sequential Higher-Order Languages" at The Semantic Challenge of Object-Oriented Programming, Dagstuhl Seminar 98261. Schloss Dagstuhl, Wadern, Germany. June 28 - July 3
26. Lassen, S.B.: Eager normal form bisimulation. In: Proc. 20th Annual IEEE Symposium on Logic in Computer Science (LICS 2005), pp. 345–354. IEEE Computer Society, Washington, DC, USA (2005)
27. Meyer, A.R., Sieber, K.: Towards fully abstract semantics for local variables. In: Proc. 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1988), pp. 191–203. ACM, New York, NY, USA (1988)
28. Moggi, E.: Notions of computation and monads. Inf. and Comp. **93**(1), 55–92 (1991)
29. O'Hearn, P.W., Tennent, R.D.: Relational parametricity and local variables. J. ACM **42**(3), 658–709 (1995)
30. Oles, F.J.: Type algebras, functor categories and block structure. In: M. Nivat, J.C. Reynolds (eds.) Algebraic Methods in Semantics, pp. 543–573. Cambridge University Press (1985)
31. Park, D.M.R.: Concurrency and automata on infinite sequences. In: Theoretical Computer Science, 5th GI-Conference, *Lecture Notes in Computer Science*, vol. 104, pp. 167–183. Springer (1981)
32. Pierce, B.C., Sangiorgi, D.: Behavioral equivalence in the polymorphic pi-calculus. In: Proc. 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL 1997), pp. 242–255. ACM Press, New York, NY, USA (1997)
33. Pitts, A.M., Stark, I.D.B.: Observable properties of higher order functions that dynamically create local names, or: What's new? In: Proc. 18th International Symposium on Mathematical Foundations of Computer Science (MFCS 1993), *Lecture Notes in Computer Science*, vol. 711, pp. 122–141. Springer-Verlag, Berlin, Heidelberg, and New York (1993)
34. Reynolds, J.C.: The essence of Algol. In: J.W. de Bakker, J.C. van Vliet (eds.) Algorithmic Languages, pp. 345–372. North-Holland (1981)

35. Sangiorgi, D.: Locality and non-interleaving semantics in calculi for mobile processes. Theoretical Computer Science **155**, 39–83 (1996)
36. Sangiorgi, D.: On the bisimulation proof method. Mathematical Structures in Computer Science **8**, 447–479 (1998)
37. Sangiorgi, D., Kobayashi, N., Sumii, E.: Logical bisimulations and functional languages. In: Proc. International Symposium on Fundamentals of Software Engineering (FSEN 2007), *Lecture Notes in Computer Science*, vol. 4767, pp. 364–379. Springer, Berlin, Heidelberg, and New York (2007)
38. Sangiorgi, D., Kobayashi, N., Sumii, E.: Environmental bisimulations for higher-order languages. ACM Trans. Program. Lang. Syst. **33**(1), 5:1–5:69 (2011). (Extended abstract appeared in LICS 2007.)
39. Sangiorgi, D., Milner, R.: The problem of "Weak Bisimulation up to". In: W. Cleveland (ed.) Proc. 3rd International Conference on Concurrency Theory (CONCUR 1992), *Lecture Notes in Computer Science*, vol. 630, pp. 32–46. Springer, Berlin, Heidelberg, and New York (1992)
40. Stark, I.: Categorical models for local names. LISP and Symbolic Computation **9**(1), 77–107 (1996)
41. Stark, I.D.B.: Names and higher-order functions. Ph.D. thesis, University of Cambridge, Cambridge, UK (1994). Also published as Technical Report 363, University of Cambridge Computer Laboratory
42. Støvring, K., Lassen, S.B.: A complete, co-inductive syntactic theory of sequential control and state. In: Proc. 34th ACM SIGPLAN-SIGACT symposium on principles of programming languages (POPL 2007), pp. 161–172. ACM, New York, NY, USA (2007)
43. Sumii, E., Pierce, B.C.: A bisimulation for dynamic sealing. Theoretical Computer Science **375**(1–3), 169–192 (2007). (Extended abstract appeared in POPL 2004.)
44. Sumii, E., Pierce, B.C.: A bisimulation for type abstraction and recursion. Journal of the ACM **54**(5), 1–43 (2007). (Extended abstract appeared in POPL 2005.)
45. Tiuryn, J., Wand, M.: Untyped lambda-calculus with input-output. In: H. Kirchner (ed.) Proc. 21st International Colloquium on Trees in Algebra and Programming (CAAP 1996), *Lecture Notes in Computer Science*, vol. 1059, pp. 317–329. Springer, Berlin, Heidelberg, and New York (1996)
46. Tzevelekos, N.: Nominal game semantics. Ph.D. thesis, University of Oxford, Oxford, UK (2009). Also published as OUCL Technical Report RR-09-18
47. Tzevelekos, N.: Program equivalence in a simple language with state. Computer Languages, Systems & Structures **38**(2), 181 – 198 (2012)
48. Urban, C.: Nominal techniques in Isabelle/HOL. J. Automated Reasoning **40**(4), 327–356 (2008)
49. Zhang, Y., Nowak, D.: Logical relations for dynamic name creation. In: Proceedings of the 17th International Workshop on Computer Science Logic (CSL), *Lecture Notes in Computer Science*, vol. 2803. Springer-Verlag (2003)

A Omitted Proofs

Theorem 4.2, reducing general contextual equivalence to equivalence of closed values, is established via an auxiliary congruence relation $(\preccurlyeq)$, shown in Figure 7.

We write $s;\Gamma \vdash e \preccurlyeq^m e' : T$ when there is some derivation tree of height less than m for the relatedness judgement. The following lemma states several useful properties of the $(\preccurlyeq)$ relation.

Lemma A.1 (Properties of $(\preccurlyeq)$)
1. *If* $s;\Gamma \vdash b \preccurlyeq b' : o$ *then* $b = b'$.
2. *If* $s;\Gamma \vdash e \preccurlyeq e' : T$ *then* $s \oplus s';\Gamma \vdash e \preccurlyeq e' : T$.
3. *If* $s;\Gamma \vdash e \preccurlyeq e' : T$ *and* $s;\Gamma_0 \vdash C[\cdot]^T_\Gamma : T_0$ *then* $s;\Gamma_0 \vdash C[e] \preccurlyeq C[e'] : T_0$.
4. *If* $s;\overline{x:T} \vdash e \preccurlyeq e' : T_0$ *and* $s;\cdot \vdash \overline{v} \preccurlyeq \overline{v'} : \overline{T}$ *then* $s;\cdot \vdash e[\overline{v/x}] \preccurlyeq e'[\overline{v/x}] : T_0$.
5. *If* $s;\Gamma \vdash \lambda x{:}T_1.\, e \preccurlyeq \lambda x{:}T_1.\, e' : T_1 \to T_2$ *then* $s;\Gamma, x:T_1 \vdash e \preccurlyeq e' : T_2$.

Proof The first property is immediate by construction of $(\preccurlyeq)$ and the second by Lemma 2.1. We prove the third property by induction on the typing derivation $s;\Gamma_0 \vdash C[\cdot]^T_\Gamma : T_0$ and the fourth property by induction on the sequence $\overline{x:T}$.

We prove the last property by induction on the height of the derivation $s;\Gamma \vdash \lambda x{:}T_1.\, e \preccurlyeq \lambda x{:}T_1.\, e' : T_1 \to T_2$ using the following induction hypothesis:

$$\begin{aligned} IH(m) = \forall s,\Gamma,x,T_1,T_2,e,e'.\ & \\ & s;\Gamma \vdash \lambda x{:}T_1.\, e \preccurlyeq^m \lambda x{:}T_1.\, e' : T_1 \to T_2 \\ & \implies s;\Gamma, x:T_1 \vdash e \preccurlyeq e' : T_2 \end{aligned}$$

The base case is trivial. In the inductive case we assume

$$s;\Gamma \vdash \lambda x{:}T_1.\, e \preccurlyeq^m \lambda x{:}T_1.\, e' : T_1 \to T_2$$

and take cases on this derivation. There are three cases that apply:

$$\frac{}{s;\Gamma,x:T\vdash x\preccurlyeq x:T} \qquad \frac{}{s;\Gamma\vdash \texttt{true}\preccurlyeq \texttt{true}:o} \qquad \frac{}{s;\Gamma\vdash \texttt{false}\preccurlyeq \texttt{false}:o} \qquad \frac{n\in s}{s;\Gamma\vdash n\preccurlyeq n:\nu}$$

$$\frac{}{s;\Gamma\vdash \texttt{new}\preccurlyeq \texttt{new}:\nu} \qquad \frac{s;\Gamma,x:T_1\vdash e\preccurlyeq e':T_2}{s;\Gamma\vdash(\lambda x{:}T_1.\,e)\preccurlyeq(\lambda x{:}T_1.\,e'):T_1\to T_2} \qquad \frac{\begin{array}{c}s;\Gamma\vdash e_1\preccurlyeq e_1':T_1\to T_2\\ s;\Gamma\vdash e_2\preccurlyeq e_2':T_1\end{array}}{s;\Gamma\vdash e_1\ e_2\preccurlyeq e_1'\ e_2':T_2}$$

$$\frac{s;\Gamma\vdash e_0\preccurlyeq e_0':o \qquad s;\Gamma\vdash e_1\preccurlyeq e_1':T \qquad s;\Gamma\vdash e_2\preccurlyeq e_2':T}{s;\Gamma\vdash \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2\preccurlyeq \texttt{if}\ e_0'\ \texttt{then}\ e_1'\ \texttt{else}\ e_2':T} \qquad \frac{\begin{array}{c}s;\Gamma\vdash e_1\preccurlyeq e_1':\nu\\ s;\Gamma\vdash e_2\preccurlyeq e_2':\nu\end{array}}{s;\Gamma\vdash(e_1=e_2)\preccurlyeq(e_1'=e_2'):o}$$

$$\frac{s;\Gamma,x:T_0\vdash e\preccurlyeq e':T}{s;\Gamma,x:T_0\vdash e\preccurlyeq(\lambda x{:}T_0.\,e')\ x:T} \qquad \frac{s;\Gamma,x:T_0\vdash e\preccurlyeq e':T \qquad s;\cdot\vdash v\preccurlyeq v':T_0}{s;\Gamma\vdash e[v/x]\preccurlyeq e'[v'/x]:T}$$

Fig. 7 The congruence relation $(\preccurlyeq)$.

CASE $\dfrac{s;\Gamma,y:T_1\to T_2\vdash y\preccurlyeq^{m_1} y:T_1\to T_2 \qquad s;\cdot\vdash\lambda x{:}T_1.\,e\preccurlyeq^{m_2}\lambda x{:}T_1.\,e':T_1\to T_2}{s;\Gamma\vdash y[\lambda x{:}T_1.\,e/y]\preccurlyeq^{m} y[\lambda x{:}T_1.\,e'/y]:T_1\to T_2}$, $m_1<m$, $m_2<m$: This case follows by $IH(m_2)$ and $s;\cdot\vdash\lambda x{:}T_1.\,e\preccurlyeq^{m_2}\lambda x{:}T_1.\,e':T_1\to T_2$.

CASE $\dfrac{s;\Gamma,y:T_0\vdash\lambda x{:}T_1.\,e_0\preccurlyeq^{m_1}\lambda x{:}T_1.\,e_0':T_1\to T_2 \qquad s;\cdot\vdash v\preccurlyeq^{m_2} v':T_0}{s;\Gamma\vdash\lambda x{:}T_1.\,e_0[v/y]\preccurlyeq^{m}\lambda x{:}T_1.\,e_0'[v'/y]:T_1\to T_2}$, $m_1<m$, $m_2<m$, $e=e_0[v/y]$, $e'=e_0'[v'/y]$: By $IH(m_1)$ and $s;\Gamma,y:T_0\vdash\lambda x{:}T_1.\,e_0\preccurlyeq^{m_1}\lambda x{:}T_1.\,e_0':T_1\to T_2$ we get

$$s;\Gamma,y:T_0,x:T_1\vdash e_0\preccurlyeq e_0':T_2$$

and therefore

$$s;\Gamma,x:T_1\vdash e_0[v/y]\preccurlyeq e_0'[v/y]:T_2$$

CASE $\dfrac{s;\Gamma,x:T_1\vdash e\preccurlyeq^{m-1} e':T_2}{s;\Gamma\vdash\lambda x{:}T_1.\,e\preccurlyeq^{m}\lambda x{:}T_1.\,e':T_1\to T_2}$: This case follows by $IH(m-1)$. □

The following lemma says that (in a suitably generalised sense) evaluation preserves $(\preccurlyeq)$:

Lemma A.2 *If $s;\overline{x:T}\vdash e\preccurlyeq e':T$ and $s;\cdot\vdash\overline{v}\preccurlyeq\overline{v'}:\overline{T}$ then if $s\vdash e[\overline{v/x}]\Downarrow(t)\,w$, there exists w' such that*

$$s\vdash e'[\overline{v'/x}]\Downarrow(t)\,w' \qquad s\oplus t;\cdot\vdash w\preccurlyeq w':T$$

and similarly for any t,w' such that $s\vdash e'[\overline{v'/x}]\Downarrow(t)\,w'$.

Proof We give the proof for the forward direction; the converse is similar. We define the lexicographic ordering $(\prec_{\text{lex}})$ of pairs of natural numbers by

$$(i,j)\prec_{\text{lex}}(k,l) \quad \textit{if} \quad i<k \textit{ or } (i=k \textit{ and } j<l)$$

and proceed by lexicographic induction on the pair of sizes (k,m) of the derivations $s\vdash e[\overline{v/x}]\Downarrow^k(t)\,w$ and $s;\overline{x:T}\vdash e\preccurlyeq^m e':T$. The induction hypothesis is the following.

$$\begin{aligned}IH(k,m)\stackrel{\text{def}}{=}\forall\overline{x,v,v'},e,e',s,w,t.&\\ &(s;\overline{x:T}\vdash e\preccurlyeq^m e':T)\wedge(s;\cdot\vdash\overline{v}\preccurlyeq\overline{v'}:\overline{T})\\ &\wedge(s\vdash e[\overline{v/x}]\Downarrow^k(t)\,w)\\ &\Longrightarrow\exists w'.\ (s\vdash e'[\overline{v'/x}]\Downarrow(t)\,w')\wedge(s\oplus t;\cdot\vdash w\preccurlyeq w':T)\end{aligned}$$

We will prove that, for all k and m, $IH(k,m)$ holds by proving that for any k and m

$$(\forall\, j,n.\ (j,n) \prec_{\text{lex}} (k,m) \implies IH(j,n)) \implies IH(k,m)$$

Assume

$$\forall\, j,n.\ (j,n) \prec_{\text{lex}} (k,m) \implies IH(j,n)$$

and $s;\overline{x:T} \vdash e \preccurlyeq^m e' : T$, $s;\cdot \vdash \overline{v} \preccurlyeq \overline{v'} : \overline{T}$, and $s \vdash e[\overline{v/x}] \Downarrow^k (t)\, w$. We proceed by cases on $s;\overline{x:T} \vdash e \preccurlyeq^m e' : T$. Most cases easily follow from the induction hypothesis. The interesting cases are the ones for lambda abstraction and application, and the beta and substitution rules shown above.

CASE $\dfrac{s;\overline{x:T}, y:T_1 \vdash e_0 \preccurlyeq^{m-1} e_0' : T_2}{s;\overline{x:T} \vdash \lambda y{:}T_1.\, e_0 \preccurlyeq^m \lambda y{:}T_1.\, e_0' : T_1 \to T_2}$: We have $s \vdash \lambda y{:}T_1.\, e_0[\overline{v/x}] \Downarrow^k (\emptyset)\, \lambda y{:}T_1.\, e_0[\overline{v/x}]$ and $s \vdash \lambda y{:}T_1.\, e_0'[\overline{v'/x}] \Downarrow (\emptyset)\, \lambda y{:}T_1.\, e_0'[\overline{v'/x}]$ and by Lemma A.1 (4):

$$s;\cdot \vdash \lambda y{:}T_1.\, e_0[\overline{v/x}] \preccurlyeq \lambda y{:}T_1.\, e_0'[\overline{v'/x}] : T_1 \to T_2$$

CASE $\dfrac{s;\overline{x:T_0} \vdash e_1 \preccurlyeq^{m_1} e_1' : T_2 \to T \quad s;\overline{x:T_0} \vdash e_2 \preccurlyeq^{m_2} e_2' : T_2}{s;\overline{x:T_0} \vdash (e_1\ e_2) \preccurlyeq^m e_1'\ e_2' : T}$, $m_1 < m$, $m_2 < m$: We have $s \vdash (e_1\ e_2)[\overline{v/x}] \Downarrow^k (t)\, w$, thus for some s_1, s_2, s_3, $\lambda y{:}T_2.\, e_3$, w_2, and $\overline{k} < k$

$$\begin{aligned} s &\vdash e_1[\overline{v/x}] \Downarrow^{k_1} (s_1)\, \lambda y{:}T_2.\, e_3 \\ s \oplus s_1 &\vdash e_2[\overline{v/x}] \Downarrow^{k_2} (s_2)\, w_2 \\ s \oplus s_1 \oplus s_2 &\vdash e_3[w_2/y] \Downarrow^{k_3} (s_3)\, w \end{aligned}$$

and $t = s \oplus s_1 \oplus s_2 \oplus s_3$. By $IH(k_1,m_1)$ and $IH(k_2,m_2)$, there exist $\lambda y{:}T_2.\, e_3'$ and w_2', such that

$$\begin{aligned} s &\vdash e_1'[\overline{v'/x}] \Downarrow (s_1)\, \lambda y{:}T_2.\, e_3' \qquad & s \oplus s_1;\cdot &\vdash \lambda y.\, e_3 \preccurlyeq \lambda y.\, e_3' : T_2 \to T \\ s_1 &\vdash e_2'[\overline{v'/x}] \Downarrow (s_2)\, w_2' & s \oplus s_1 \oplus s_2;\cdot &\vdash w_2 \preccurlyeq w_2' : T_2 \end{aligned}$$

Therefore, by Lemma A.1 (2),

$$s \oplus s_1 \oplus s_2;\cdot \vdash \lambda y.\, e_3 \preccurlyeq \lambda y.\, e_3' : T_2 \to T$$

and by Lemma A.1 (5), for some m_3,

$$s \oplus s_1 \oplus s_2; y:T_2 \vdash e_3 \preccurlyeq^{m_3} e_3' : T$$

Because $k_3 < k$, it is $(k_3,m_3) \prec_{\text{lex}} (k,m)$. Thus, by $IH(k_3,m_3)$ we get that there exists w', such that

$$s \oplus s_1 \oplus s_2 \vdash e_3'[w_2'/y] \Downarrow (s_3)\, w' \qquad s \oplus s_1 \oplus s_2 \oplus s_3;\cdot \vdash w \preccurlyeq w' : T$$

and therefore, by the evaluation rule of application,

$$s \vdash (e_1'\ e_2')[\overline{v'/x}] \Downarrow (t)\, w'.$$

CASE $\dfrac{s;\overline{x:T}, y:T_0 \vdash e \preccurlyeq^{m-1} e' : T}{s;\overline{x:T}, y:T_0 \vdash e \preccurlyeq^m (\lambda y{:}T_0.\, e')\ y : T}$: We have $s \vdash e[\overline{v/x}, u/y] \Downarrow^k (t)\, w$. By $IH(k,m-1)$ we get that for any u' with $s;\cdot \vdash u \preccurlyeq u' : T_0$ there exists w' such that

$$s \vdash e'[\overline{v'/x}, u'/y] \Downarrow (t)\, w' \qquad s \oplus t;\cdot \vdash w \preccurlyeq w' : T$$

and therefore $s \vdash ((\lambda y{:}T_0.\, e')\ y)[\overline{v'/x}, u'/y] \Downarrow (t)\, w'$.

CASE $\dfrac{s;\overline{x:T}, y:T_0 \vdash e \preccurlyeq^{m_1} e' : T \quad s;\cdot \vdash u \preccurlyeq^{m_2} u' : T_0}{s;\overline{x:T} \vdash e[u/y] \preccurlyeq^m e'[u'/y] : T}$, $m_1 < m$, $m_2 < m$: We have $s \vdash e[\overline{v/x}, u/y] \Downarrow^k (t)\, w$. By $IH(k,m_1)$ we get that there exists w' such that

$$s \vdash e'[\overline{v'/x}, u'/y] \Downarrow (t)\, w' \qquad s \oplus t;\cdot \vdash w \preccurlyeq w' : T$$

which concludes this proof. □

It is immediate by the above lemma and Lemma A.1(1) that $(\preccurlyeq)$-related expressions at type o evaluate to the same value:

Corollary A.3 *If $s;\cdot \vdash e \preccurlyeq e' : o$, t is a nameset, and b a boolean value then*

$$s \vdash e \Downarrow (t)\, b \iff s \vdash e' \Downarrow (t)\, b.$$

We can now give the proof of Theorem 4.2.

Proof (Theorem 4.2) The forward direction follows directly by the definition of $(\equiv)$. For the converse direction we need to show that for all contexts C with $s;\cdot \vdash C[\cdot]^{T}_{\Gamma} : o$ and boolean values b,

$$(\exists t.\ s \vdash C[e] \Downarrow (t)\, b) \iff (\exists t.\ s \vdash C[e'] \Downarrow (t)\, b)$$

assuming $s;\vdash \lambda\overline{x{:}T}.\, e \equiv \lambda\overline{x{:}T}.\, e' : \overline{T} \to T$ and $\overline{x:T} \in \Gamma$. By construction of $(\preccurlyeq)$ and Lemma A.1(3),

$$s;\cdot \vdash C[e] \preccurlyeq C[(\lambda\overline{x{:}T}.\, e)\, x_1 \ldots x_n] : o \tag{11}$$

$$s;\cdot \vdash C[e'] \preccurlyeq C[(\lambda\overline{x{:}T}.\, e')\, x_1 \ldots x_n] : o \tag{12}$$

Hence

$$\begin{aligned}
& \exists t.\ s \vdash C[e] \Downarrow (t)\, b \\
\textit{iff}\ & \exists t.\ s \vdash C[(\lambda\overline{x{:}T}.\, e)\, x_1 \ldots x_n] \Downarrow (t)\, b && \text{(by Corollary A.3 and (11))} \\
\textit{iff}\ & \exists t.\ s \vdash C[(\lambda\overline{x{:}T}.\, e')\, x_1 \ldots x_n] \Downarrow (t)\, b && \text{(by Definition 4.1)} \\
\textit{iff}\ & \exists t.\ s \vdash C[e'] \Downarrow (t)\, b && \text{(by Corollary A.3 and (12))}
\end{aligned}$$

□