A Mechanized Bisimulation for the Nu-Calculus

Nick Benton
Microsoft Research
Cambridge

Vasileios Koutavas
Northeastern University
Boston

September 2008

Technical Report
MSR-TR-2008-129

We introduce a Sumii-Pierce-Koutavas-Wand-style bisimulation for Pitts and Stark's nu-calculus, 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 [11]. We also describe the formalization of soundness and of the examples in the Coq proof assistant.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
http://www.research.microsoft.com

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 [10, 11] is a simply typed 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. The interaction of generativity with higher-order functions and the restricted nature of contexts lead to various subtle and hard-to-prove equivalences, of which the canonical 'hard' example is the following:

$$\nu n.\nu n'.\lambda f{:}\nu \to o.(f\ n{=}f\ n') \quad \cong \quad \lambda f{:}\nu \to o.\mathtt{true} \tag{1}$$

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 $f\ n'$. The intuition here is that the two names 'leak' into f but they never escape its dynamic extent. The subtlety of the ν-calculus is indicated by the following *in*equivalence, which similar intuitions might lead one to believe to be an equivalence.

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

Pitts and Stark have used logical relations to establish many equivalences, both directly over the operational semantics and denotationally, refining a model in the functor category $Set^{\mathcal{I}}$. Yang and Nowak [14] 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 (1) above. Jeffrey and Rathke [5] define a sound and complete bisimulation for an extension of the ν-calculus with assignment (for which (1) 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 [9] constructs a game model using automorphisms of names that is fully abstract for a language like that of Jeffrey and Rathke. Abramsky et al [1] use games in the topos of FM-sets to construct the first fully abstract model of ν-calculus proper (and the first to validate (1)).

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 [12] and existential and recursive types

[13] and later developed by Koutavas and Wand for reasoning about untyped imperative higher-order [7] and object [6] 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, $\mathcal{X}$, of typed relations, R, each of which is annotated by two sets of (generated) names, s and s'.

The theoretical development broadly follows that of previous work by Koutavas and Wand [7, 6, 8]. 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 (1) above. The proof of (1) is particularly interesting in making two uses of our technique: the adequacy of an initial relation is established via that of another. The third contribution is a formalization of the metatheory and of the examples in the Coq theorem prover. We discuss the formalization in Section 7; the proof script is also available via the authors' homepages.

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 $(\lambda x{:}\nu.\, e)$ `new`,[1] and $(e = e')$, when e and e' have type o, as syntactic sugar for

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

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.

[1] Pitts and Stark take $\nu x.\, e$ as primitive and define `new` as $\nu x.\, x$—the presentations are entirely equivalent.

TYPE:	T	$::= o$	Boolean
		$\mid \nu$	Name
		$\mid T \to T$	Function
EXPRESSION:	e, d	$::= x$	Identifier
		$\mid n$	Name
		$\mid$ `true` $\mid$ `false`	Boolean Constants
		$\mid \lambda x{:}T.\, e$	Abstraction
		$\mid e\ e$	Application
		$\mid$ `new`	Fresh Name Generation
		$\mid (e = e)$	Name Equality
		$\mid$ `if` e `then` e `else` e	Conditional
VALUE:	u, v, w	$::= n \mid$ `true` $\mid$ `false`	
		$\mid \lambda x{:}T.\, e$	
NAME:	n		
NAMESET:	s, t	$\in \mathscr{P}_{\mathsf{fin}}(\text{NAME})$	

Figure 1: Syntactic domains of the ν-calculus

The typing judgment $s; E \vdash e{:}T$ says that the expression e has type T under the nameset s and typing environment E. 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 judgment $s \vdash e \Downarrow^k (t)\, w$ says that the closed, well-typed expression e, under the nameset s, terminates with the value w producing a set of fresh names t. 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.

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

Lemma 2.1 (Type Preservation) *If* $s; \cdot \vdash e : T$ *and* $s \vdash e \Downarrow (t)\, v$ *then* $s \oplus t; \cdot \vdash v : T$.

Lemma 2.2 (Strong Normalization) *If* $s; \cdot \vdash e : T$ *then* $s \vdash e \Downarrow$.

Lemma 2.3 (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* $b_1 = b_2$.

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

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

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

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

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

Figure 2: Typing rules for the ν-calculus

Lemma 2.4 (Garbage Addition) *If* $s \vdash e \Downarrow^k (t)\, w$ *and* $s \cap s_0 = t \cap s_0 = \emptyset$ *then:*

$$s \oplus s_0 \vdash e \Downarrow^k (t)\, w.$$

Lemma 2.5 (Garbage Collection) *If* $s \oplus s_0 \vdash e \Downarrow^k (t)\, w$ *and* $s_0 \cap names(e) = \emptyset$ *then:*

$$s \vdash e \Downarrow^k (t)\, w.$$

3 Equivalence and Adequacy

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

3.1 Contextual Equivalence

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

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

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 the terms do

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

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

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

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

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

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

Figure 3: Operational semantics for the ν-calculus

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: under the nameset s and type environment Γ, C is a context of type T when it contains a hole that can be filled with terms that, under the same nameset and environment Γ', have type T'. We write $s; \Gamma \vdash C[\cdot]^{T'}_{\Gamma'} : T$ for such a context.

The ν-calculus is strongly normalizing, hence, as in [11], we take as our notion of observation the (in-)equality of final values at type o.

Definition 3.1 (Contextual Equivalence ($\equiv$)) *$s; \Gamma \vdash e \equiv e' : T$ 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. We simplify this situation by making use of the following theorem, which allows us to work only with closed values.

Theorem 3.2 (Expression Closedness) *For any two expressions e and e' with $s; \overline{x : T} \vdash e, e' : T$*

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

To prove this theorem we define the relation $(\preccurlyeq)$ as the smallest congruence on expressions that admits the following rules.

$$\frac{s; \Gamma, x : T_0 \vdash e \preccurlyeq e' : T}{s; \Gamma, x : T_0 \vdash e \preccurlyeq (\lambda x{:}T_0.\, e')\ x : T}$$

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

The theorem follows from the next bisimulation lemma for $(\preccurlyeq)$.

Lemma 3.3 *If $s; \overline{x : T} \vdash e \preccurlyeq e' : T$ and $s; \cdot \vdash \overline{v} \preccurlyeq \overline{v'} : \overline{T}$ and $s \vdash e[\overline{v/x}] \Downarrow (t)\, w$ then 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 vice versa.

Proof. Here we show the proof the forward direction; the proof of the converse is similar. We proceed by induction on the height of the derivations $s \vdash e[\overline{v/x}] \Downarrow (t)\, w$ and $s; \overline{x : T} \vdash e \preccurlyeq e' : T$, ordered lexicographically. The induction hypothesis is the following.

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

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

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

We 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 by the induction hypothesis. The interesting cases are the ones for application, and the beta and substitution rules shown above.

Case $\dfrac{\begin{array}{c} s;\overline{x:T_0} \vdash e_1 \preccurlyeq^{m_1} e_1' : T_2 \to T \\ s;\overline{x:T_0} \vdash e_2 \preccurlyeq^{m_2} e_2' : T_2 \end{array}}{s;\overline{x:T_0} \vdash (e_1\ e_2) \preccurlyeq^{m} (e_1'\ e_2') : T} \qquad m_1 < m,\ \ m_2 < m$

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 $\overline{k} < k$

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

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

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

By construction of $(\preccurlyeq)$ we get that $s{\oplus}s_1{\oplus}s_2; \cdot \vdash \lambda y.\, e_3 \preccurlyeq \lambda y.\, e_3' : T_2 \to T$ and $s{\oplus}s_1{\oplus}s_2; y : T_2 \vdash e_3 \preccurlyeq e_3' : T$ and $s{\oplus}s_1{\oplus}s_2; \cdot \vdash e_3[w_2/y] \preccurlyeq e_3'[w_2'/y] : T$. Thus, by $IH(k_3, m_3)$, for any m_3, we get that there exist t' and 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, $s \vdash (e_1'\ e_2')[\overline{v'/z}] \Downarrow (t)\, w'$.

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 $\dfrac{\begin{array}{c} s;\overline{x:T}, y : T_0 \vdash e \preccurlyeq^{m_1} e' : T \\ s; \cdot \vdash u \preccurlyeq^{m_2} u' : T_0 \end{array}}{s;\overline{x:T} \vdash e[u/y] \preccurlyeq^{m} e'[u'/y] : T} \qquad m_1 < m,\ m_2 < m$

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 from the above lemma and the construction of $(\preccurlyeq)$ that $\preccurlyeq$-related expressions at type o evaluate to the same value.

Corollary 3.4 *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 3.2. *Proof.*[Theorem 3.2] 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 $(\preccurlyeq)$,

$$s; \cdot \vdash C[e] \preccurlyeq C[(\lambda\overline{x{:}T}.\, e)\ x_1 \ldots x_n] : o \tag{2}$$

$$s; \cdot \vdash C[e'] \preccurlyeq C[(\lambda\overline{x{:}T}.\, e')\ x_1 \ldots x_n] : o \tag{3}$$

Hence

$$\begin{array}{lll} & \exists t.\ s \vdash C[e] \Downarrow (t)\, b & \\ \textit{iff} & \exists t.\ s \vdash C[(\lambda\overline{x{:}T}.\, e)\ x_1 \ldots x_n] \Downarrow (t)\, b & \text{(by Lemma 3.4 and (2))} \\ \textit{iff} & \exists t.\ s \vdash C[(\lambda\overline{x{:}T}.\, e')\ x_1 \ldots x_n] \Downarrow (t)\, b & (s; \cdot \vdash \lambda\overline{x{:}T}.\, e \equiv \lambda\overline{x{:}T}.\, e' : \overline{T} \to T) \\ \textit{iff} & \exists t.\ s \vdash C[e'] \Downarrow (t)\, b & \text{(by Lemma 3.4 and (3))} \end{array}$$

□

3.2 Pre-Adequacy

Reasoning about intermediate states of ν-calculus programs will require us to consider relate 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:

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

We write $s, s'; \cdot \vdash v\ R\ v' : T$ when $(v, v', T) \in R$.

We reason about sets of such relations:

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

The inverse of a set of annotated relations is defined as follows.

Definition 3.5 *If $\mathcal{X}$ is a set of annotated relations, the inverse of $\mathcal{X}$, written $\mathcal{X}^{-1}$, is defined as:*

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

We close annotated relations under name-free, identical contexts. Therefore, we allow contexts to access only related names that can be substituted in holes. This is an important distinction between *public* (i.e. related) names and names that are private to the terms.

Definition 3.6 (Context Closure of Annotated Relations) *If (s, s', R) is an annotated relation on closed values, then $(s, s', R^{\mathsf{cxt}})$ 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 3.7 (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 3.8 (Pre-Adequacy $(\cong)$) *$(\cong)$ is 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 3.9 (Open Extension of Sets of Annotated Relations) *If $\mathcal{X}$ is a set of annotated relations, then $\mathcal{X}^\circ$ is an indexed relation on open expressions such that $s; \overline{x : T} \vdash e\ \mathcal{X}^\circ\ e' : T$ if and only if there exists R such that:*

$$(s, s, R) \in \mathcal{X} \qquad s, s; \cdot \vdash \lambda\overline{x{:}T}.e\ R\ \lambda\overline{x{:}T}.e' : \overline{T} \to T \qquad \forall n \in s.\ s, s; \cdot \vdash n\ R\ n : \nu$$

The contexts in the definition of contextual equivalence and the contexts in the definition of pre-adequate relations are slightly different: 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. Hence, in the above definition, we reconcile the two notions of contexts by requiring R to be the identity on all names in the namesets.

Theorem 3.10 (Soundness and Completeness of $(\cong)$) $(\cong)^\circ = (\equiv)$

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 3.2,

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

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

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

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

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

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

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

if and only if, by Definition 3.7 and the definition of $(\cong)$,

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

if and only if, by Definition 3.9,

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

3.3 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 3.11 (Adequate Sets of Annotated Relations) *A set of annotated relations $\mathcal{X}$ is* adequate *if an only if for all $(s, s', R) \in \mathcal{X}$ we have:*

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

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

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 3.12 (Adequacy ($\approx$)) ($\approx$) *is the largest adequate set of annotated relations.*

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

Theorem 3.13 (Soundness of Adequate Sets) *If $\mathcal{X}$ is adequate then it is included in pre-adequacy.*

Proof. Trivial by the definitions of pre-adequate annotated relations and adequate sets of annotated relations. □

Theorem 3.14 (Completeness of Adequate Sets) ($\cong$) *is adequate.*

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

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

By the definition of pre-adequate annotated relations (Definition 3.7) and the determinacy of the semantics, it suffices to show that

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

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

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

we have:

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

Therefore, by Definitions 3.7 and 3.8:

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

Theorem 3.15 $(\cong) = (\approx)$.

Proof. By Theorem 3.13 we have $(\approx) \subseteq (\cong)$ and by Theorem 3.14 we have $(\cong) \subseteq (\approx)$. Thus $(\cong) = (\approx)$. $\square$

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

Theorem 3.16 $(\approx)^\circ = (\equiv)$.

Proof. By Theorems 3.10 and 3.15. $\square$

4 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 $\mathcal{X}$ containing (s, s, R) such that

$$s; \cdot \vdash \lambda \overline{x{:}T}.\, e \; R \; \lambda \overline{x{:}T}.\, e' : \overline{T} \to T$$
$$\forall n \in s.\ s; \cdot \vdash n \; R \; n : \nu$$

2. show that $\mathcal{X}$ is adequate, and

3. invoke Theorem 3.16 to show $s; \overline{x : T} \vdash e \equiv e' : T$.

We can show a set $\mathcal{X}$ adequate *by induction*. The induction hypothesis we use is the following.

Definition 4.1

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

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 $\mathcal{X}$ adequate amounts to proving that for all k, $IH_{\mathcal{X}}(k)$ and $IH_{\mathcal{X}^{-1}}(k)$ hold. For $k = 0$ it is trivial; for $k > 0$ it can be shown by induction:

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

5 Deriving Smaller Proof Obligations for Adequate Sets

By using a *proof construction scheme*, as in [7], 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 5.1 *A set of annotated relations $\mathcal{X}$ is adequate if and only if for all k and all $(s, s', R) \in \mathcal{X}$, assuming that $IH_{\mathcal{X}}(k-1)$ 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:*

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

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

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

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

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

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

Proof. We prove the forward direction by showing that if one of the conditions is not satisfied, then $\mathcal{X}$ is not adequate. The converse direction is immediate by the proof construction scheme. □

$$\frac{}{([],[],\{(N,N',o\to\nu\to o)\})\in\mathcal{X}}\ \mathcal{X}\text{-1}$$

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

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

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

6 Examples

Using the preceding theorem, we are able to prove all equivalences in the ν-calculus from [11]. In this section we start with the proof of a straightforward equivalence and then show the proof of the most interesting of these equivalences. The only other method for proving the latter equivalence is by using game semantics [1].

6.1 A Simple Example: Local Names

This equivalence demonstrates that the context can not provide names that are local to the terms.

$$M \stackrel{\text{def}}{=} \nu n.\,\lambda x{:}\nu.\,(x{=}n)$$
$$M' \stackrel{\text{def}}{=} \lambda x{:}\nu.\,\mathtt{false}$$

Proof.

From Theorem 3.2, it suffices to show that the following two values are equivalent:

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

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

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

We start the construction of an adequate $\mathcal{X}$ by the first two rules of Figure 4. Rule $\mathcal{X}$-2 fulfills Condition 3 of Theorem 5.1. Conditions 1 and 4 are trivially satisfied.

We need to prove Condition 2 of Theorem 5.1 for any $(s, s', R) \in \mathcal{X}$ 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:

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

To prove this case we add Rule $\mathcal{X}$-3 of Figure 4 in the construction of $\mathcal{X}$. Hence we have:

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

It remains to prove Condition 2 for any $(s \oplus \{n\}, s', R) \in \mathcal{X}$ with $s \oplus \{n\}, s'; \cdot \vdash \lambda x{:}\nu.(x{=}n)\ R\ \lambda x{:}\nu.\mathtt{false} : \nu \to o$. Let:

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

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

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

By construction of $\mathcal{X}$ we have that $n \neq n_0$. Hence:

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

This concludes the proof of adequacy of $\mathcal{X}$, and by Theorem 3.16:

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

6.2 The 'Hard' Equivalence

Here we show the proof of what is considered the canonical hard-to-prove equivalence of the ν-calculus. This has only been validated with the use of game semantics [1]. Our proof here uses operational semantics and two adequate sets of annotated relations.

$$\begin{array}{ll} M \stackrel{\text{def}}{=} \nu n_1.\nu n_2.U(n_1, n_2) & U(n_1, n_2) \stackrel{\text{def}}{=} \lambda f{:}\nu \to o.((f\ n_1){=}(f\ n_2)) \\ M' \stackrel{\text{def}}{=} \lambda f{:}\nu \to o.\mathtt{true} & \end{array}$$

The equivalence here means that although the names n_1 and n_2 are revealed to the context (by passing them as arguments to f), they cannot be stored between applications of f. Thus the outermost application of $U(n_1, n_2)$ will

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

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

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

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

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

return `true`. While the body of f is evaluated, though, internal applications of $U(n_1,n_2)$ may return `false`.

Proof.

From Theorem 3.2, 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 $\emptyset;\cdot\vdash N\equiv N':o\to\nu\to o$ we need to construct an adequate set of annotated relations, $\mathcal{X}$, such that there exists R with:

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

We start the construction of an adequate $\mathcal{X}$ by the first two rules of Figure 5. Rule $\mathcal{X}$-2 fulfills Condition 3 of Theorem 5.1. Conditions 1 and 4 are trivially satisfied.

We need to prove Condition 2 of Theorem 5.1 for any $(s,s',R)\in\mathcal{X}$ 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:

$$\begin{array}{ll} s\vdash N\ b\Downarrow(\{n_1,n_2\})\,U(n_1,n_2) & \{n_1,n_2\}\cap s=\emptyset\\ s'\vdash N'\ b\Downarrow(\emptyset)\,M' & \end{array}$$

To prove this case we add Rule $\mathcal{X}$-2 in the construction of $\mathcal{X}$. Hence we have:

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

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

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

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

$$s' \vdash M'\ u' \Downarrow (t')\, w' \qquad w = w' \qquad (s \oplus \{n_1, n_2\} \oplus t, s' \oplus t', Q) \in \mathcal{X} \qquad R \subseteq Q$$

But we have:

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

Thus $t' = \emptyset$, $w' = \mathtt{true}$, and $Q = R$. We add Rule $\mathcal{X}$-4 to the construction of $\mathcal{X}$ in Figure 5 and get:

$$(s \oplus \{n_1, n_2\} \oplus t, s', R) \in \mathcal{X}$$

It only remains to show that the first application evaluates to `true`; i.e.:

$$s \oplus \{n_1, n_2\} \vdash U(n_1, n_2)\ u \Downarrow (t)\, \mathtt{true}$$

To prove this we need to reason about the operational behavior of the application. We do by re-using our method.

By the properties of evaluation, it suffices to show that there exist b, t_1, and t_2 such that:

$$s \oplus \{n_1, n_2\} \vdash u\ n_1 \Downarrow (t_1)\, b$$
$$s \oplus \{n_1, n_2\} \oplus t_1 \vdash u\ n_2 \Downarrow (t_2)\, b$$

or, from Theorem 2.4, it suffices to show that there exist b, t_1, and t_2 such that:

$$s \oplus \{n_1, n_2\} \vdash u\ n_1 \Downarrow (t_1)\, b$$
$$s \oplus \{n_1, n_2\} \vdash u\ n_2 \Downarrow (t_2)\, b$$

We show this by constructing an auxiliary adequate set $\mathcal{Y}$ of annotated relations such that there exists a relation P with:

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

The construction of $\mathcal{Y}$ is shown in Figure 6.

We establish a correlation between the sets of annotated relations $\mathcal{X}$ and $\mathcal{Y}$ by the following lemma and corollary.

Lemma 6.1 *For all $(s, s', R) \in \mathcal{X}$, there exists P such that:*

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

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

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

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

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

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

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

Case $\mathcal{X}$-1

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

This case is trivial because by $\mathcal{Y}$-1

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

Case $\mathcal{X}$-2

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

By the induction hypothesis at $(s, s', R) \in \mathcal{X}$ we get that there exists P such that

$$(s, s, P) \in \mathcal{Y} \tag{4}$$

$$\forall v, v'.\ s, s'; \cdot \vdash v\ R\ v' : T \implies s; \cdot \vdash v\ P\ v : T \tag{5}$$

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

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

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 $s; \cdot \vdash v\ P\ v : T$, and thus

$$s \oplus \{n\}; \cdot \vdash v\ (P \cup \{(n, n, \nu)\})\ v : T$$

2. $v = n$, $v = n'$, $T = \nu$. It is immediate that

$$s \oplus \{n\}; \cdot \vdash n\ (P \cup \{(n, n, \nu)\})\ n : \nu$$

Case $\mathcal{X}$-3

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

By the induction hypothesis at $(s, s', R) \in \mathcal{X}$ we get that there exists P such that

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

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

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

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 $s; \cdot \vdash v\ P\ v : T$, and thus

$$s \oplus \{n_1, n_2\}; \cdot \vdash v\ Q\ v : T$$

2. $v = U(n_1, n_2)$, $v' = M'$, $T = (\nu \to o) \to o$. It is immediate that

$$s \oplus \{n_1, n_2\}; \cdot \vdash U(n_1, n_2)\ Q\ U(n_1, n_2) : (\nu \to o) \to o$$

Case $\mathcal{X}$-4

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

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

Corollary 6.2 *For all $(s, s', R) \in \mathcal{X}$, there exists P such that:*

$$(s, s, P) \in \mathcal{Y}$$
$$s, s'; \cdot \vdash v\ R^{\mathsf{cxt}}\ v' : T \implies s; \cdot \vdash v\ P^{\mathsf{cxt}}\ v : T$$

(Continuing proof from page 17.) From the above corollary and because

$$(s \oplus \{n_1, n_2\}, s', R) \in \mathcal{X} \qquad s \oplus \{n_1, n_2\}, s'; \cdot \vdash (U(n_1, n_2)\ u)\ R^{\mathsf{cxt}}\ (M'\ u') : o$$

we have that there exists P such that:

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

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

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

Because $U(n_1, n_2)$ is related to itself only in rule $\mathcal{X}$-3, we also have that:

$$s \oplus \{n_1, n_2\}; \cdot \vdash n_1\ P\ n_2 : o$$

Therefore:

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

It remains to show that $\mathcal{Y}$ is adequate by showing that it satisfies the conditions of Theorem 5.1.

$\mathcal{Y}$ trivially satisfies Conditions 1 and 4 of Theorem 5.1. Condition 3 of the theorem is fulfilled by Rule $\mathcal{Y}$-2. It remains to prove Condition 2 for all related abstractions.

Let $(s, s', R) \in \mathcal{Y}$. It is the case that $\mathcal{Y}$ 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 $(s, s, R) \in \mathcal{Y}$, and prove Condition 2 for the following cases:

Case $s; \cdot \vdash N\ R\ N : o \to (\nu \to o) \to o$

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

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

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

Let $s; \cdot \vdash v\ R^{\mathsf{cxt}}\ v' : \nu \to o$ 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 2.5:

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

Because

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

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

$$\begin{array}{llll} s \vdash v'\ n_2 \Downarrow (s'_1)\, b'_1 & s{\oplus}s_1, s{\oplus}s'_1;\cdot \vdash b_1\ {Q_1}^{\mathsf{cxt}}\ b'_1 : o & (s{\oplus}s_1, s{\oplus}s'_1, Q_1) \in \mathcal{Y} \\ s \vdash v'\ n_1 \Downarrow (s'_2)\, b'_2 & s{\oplus}s_2, s{\oplus}s'_2;\cdot \vdash b_2\ {Q_2}^{\mathsf{cxt}}\ b'_2 : o & (s{\oplus}s_2, s{\oplus}s'_2, Q_2) \in \mathcal{Y} \end{array}$$

By the definition of $(\ )^{\mathsf{cxt}}$ we get that $b_1 = b'_1$ and $b_2 = b'_2$. Because each relation in $\mathcal{Y}$ is annotated with identical stores, $s_1 = s'_1$ and $s_2 = s'_2$. Therefore:

$$\begin{array}{l} s \vdash v'\ n_2 \Downarrow (s_1)\, b_1 \\ s \vdash v'\ n_1 \Downarrow (s_2)\, b_2 \end{array}$$

By (9) we get that $s_1 \cap s_2 = s \cap s_2 = \emptyset$. Thus, by Theorem 2.4 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 $\mathcal{X}$-4 we get:

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

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

7 The Formalization in Coq

We have formalized the semantics of the ν-calculus and our bisimulation theory in the Coq theorem prover [4]. The mechanized development covers the soundness of our method given in Section 3 and the examples given in Section 6.

There are still two axioms in our development, concerning well known basic properties of ν-calculus evaluation that are proved in Stark's thesis [11]. These are the determinacy lemma (Lemma 2.3) and the totality lemma (Lemma 2.2). These are entirely standard results and proofs, the mechanization of which is not especially interesting.

7.1 Semantics of the ν-calculus

There has been much recent research effort expended on reducing the pain of doing mechanized reasoning about syntax involving binders, most notably under the umbrella of the POPLmark challenge [3]. We were pleased to find that this effort is paying off: our formalization uses a Coq framework for 'locally nameless' reasoning about binding due to Aydemir et al.[2], which worked very well.

The locally nameless style uses de Bruijin 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 Bruijin indices). Due to space limitations we show only part of this construction here:

```
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 Bruijin 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. 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 at 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 unary environment that binds `x` to the type `U`.

For our formalization of bisimulations we needed multiple substitutions, which we got by instantiating the polymorphic library for environments from [2] to give finite maps from identifiers to `trm`s and then defining a fold function to actually apply the substitution.

7.2 Relations

We encode in Coq all definitions of Section 3. Most of them are straightforwardly transcribed. The most interesting one is the context closure of Definition 3.6. We encode it in two parts.

First we construct the $[\overline{v/x}]$ and $[\overline{v'/x}]$ of Definition 3.6 by defining an inductive relation on 'synchronized' 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 non-empty `R`, containing namesets s and s', the empty environment and the empty substitutions are synchronized. When `E`, `sub1`, are `sub2` are synchronized 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 synchronized. 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 typable under `E`.

We then define a constructor that combines two relations, using substitutions. By giving the identity relation as the first argument and `R` as the second we get the context closure R^{cxt}. This constructor is the following function that accepts only the tuples that satisfy the predicate in it.

```
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}, `sr1` and `sr2` are always empty, thus all names come from the second relation.

The proof of soundness as well as the proofs for particular equivalences are fairly long as they stand, but manageable. The approximate line counts of different sections of the Coq development are currently as follows:

Section	Lines
Library from UPenn	3500
Semantics, general lemmas, multiple substitutions	3500
Infrastructure about relations	1900
Soundness proof	2800
Simple example	1000
Hard example	3000
Total	15700

8 Conclusions

We have introduced a bisimulation for the ν-calculus that can be used to establish contextual equivalences that were previously only provable in rather sophisticated game semantic models. Moreover we formalized its metatheory 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 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.

References

[1] S. Abramsky, D. Ghica, A. Murawski, L. Ong, and I. Stark. Nominal games and full abstraction for the nu-calculus. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society Press, 2004.

[2] B. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. In *Proceedings 35th Annual ACM Symposium on Principles of Programming Languages (POPL)*, 2008.

[3] B. E. Aydemir, A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, and S. Zdancewic. Mechanized metatheory for the masses: The POPLmark Challenge. In *Proceed-*

ings of the 18th International Conference on Theorem Proving in Higher Order Logics (TPHOLs), volume 3603 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[4] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.

[5] A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1999.

[6] V. Koutavas and M. Wand. Bisimulations for untyped imperative objects. In *Proceedings of the 15th European Symposium on Programming (ESOP)*, volume 3924 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.

[7] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *Proceedings 33rd Annual ACM Symposium on Principles of Programming Languages (POPL)*. ACM Press, 2006.

[8] V. Koutavas and M. Wand. Reasoning about class behavior. Appeared in the FOOL/WOOD 2007 workshop, 2007.

[9] J. Laird. A game semantics of names and pointers. In *Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 2987 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[10] A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *Proceedings of the 18th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 711 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[11] I. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, 1994. Also available as Technical Report 363, University of Cambridge Computer Laboratory.

[12] E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. In *Proceedings 31st Annual ACM Symposium on Principles of Programming Languages (POPL)*. ACM Press, 2004.

[13] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. In *Proceedings 32nd Annual ACM Symposium on Principles of Programming Languages (POPL)*. ACM Press, 2005.

[14] Y. Zhang and D. Nowak. Logical relations for dynamic name creation. In *Proceedings of the 17th International Workshop on Computer Science Logic (CSL)*, volume 2803 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.