A Mechanized Bisimulation for the Nu-Calculus

Nick Benton[1] and Vasileios Koutavas[2]

[1] Microsoft Research, Cambridge
[2] Northeastern University, Boston

Abstract. 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 implies contextual equivalence and provides a usable and elementary method for establishing all the subtle equivalences given by Stark. We also describe the formalization of the metatheory and of the examples in 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 freshly generated 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:

$$\nu n.\, \nu n'.\, \lambda f{:}\,\nu \to o.\, f\,n{=}f\,n' \quad \cong \quad \lambda f{:}\,\nu \to o.\,\texttt{true} \tag{1}$$

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.

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

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 reason 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 [6] and object [7] 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, $\mathbb{X}$, of typed relations, R, each of which is a set of 6-tuples; each tuple relates two sets of (generated) names, s and s', and two terms e and e' of the same type T under the same typing environment E; we'll write

$$s, s', E \vdash e \; R \; e' : T$$

for such a tuple in R.

The theoretical development broadly follows that of previous work by Koutavas and Wand [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 even 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 5; the proof script is also available via the authors' homepages.

$$\begin{array}{rll} T & ::= o \mid \nu \mid T \to T & \textsc{Types} \\ e & ::= x \mid n \mid \texttt{true} \mid \texttt{false} \mid \lambda x{:}T.\, e \mid e\, e & \textsc{Expressions} \\ & \mid\ \texttt{new} \mid e{=}e \mid \texttt{if}\, e\, \texttt{then}\, e\, \texttt{else}\, e & \end{array}$$

$$\boxed{s; E \vdash e{:}\,T}$$

$$\frac{n \in s}{s; E \vdash n{:}\,\nu} \qquad \frac{}{s; E \vdash \texttt{new}{:}\,\nu} \qquad \frac{s; E \vdash e_1{:}\,\nu \qquad s; E \vdash e_2{:}\,\nu}{s; E \vdash e_1{=}e_2{:}\,o}$$

$$\boxed{s_1 \vdash e \Downarrow_T (s_2)\, v}$$

$$\frac{n \notin s}{s \vdash \texttt{new} \Downarrow_\nu (\{n\})\, n} \qquad \frac{s \vdash e_1 \Downarrow_\nu (s_1)\, n_1 \qquad s{\cup}s_1 \vdash e_2 \Downarrow_\nu (s_2)\, n_2}{s \vdash e_1{=}e_2 \Downarrow_o (s_1{\cup}s_2)\, b}$$

Fig. 1. Syntax and some typing and evaluation rules of 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 $(\lambda x{:}\,\nu.\, e)\, \texttt{new}$.[3] We also use an overbar notation to denote sequences.

The typing judgment $s; E \vdash e{:}\,T$ says that the expression e has type T under the finite set of names ('nameset') s and typing environment E. The evaluation judgment $s \vdash e \Downarrow_T (s')\, w$, says that the closed expression e of type T, under the nameset s, terminates to the value w producing a set of fresh names s'. We write $s \vdash e \Downarrow_T^k (s')\, w$ when the evaluation tree $s \vdash e \Downarrow_T (s')\, w$ has *size* less than k. Figure 1 shows the syntax, typing and evaluation rules that involve names.

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

Lemma 2.1. *If $s \vdash e \Downarrow_o (s_0)\, w$ and $s \vdash e \Downarrow_o (s_0')\, w'$ then $w = w'$.*

Lemma 2.2. *If $s; \emptyset \vdash e{:}\,T$ then there exist s_0 and w such that $s \vdash e \Downarrow_T (s_0)\, w$.*

Lemma 2.3 (Garbage Addition). *If $s_1 \vdash e \Downarrow_T^k (s_3)\, w$ and $s_2 \cap s3 = \emptyset$ then*

$$s_1 \cup s_2 \vdash e \Downarrow_T^k (s_3)\, w.$$

Lemma 2.4 (Garbage Collection). *If $s_1 {\cup} s_2 {\vdash} e {\Downarrow_T^k} (s_3)\, w$ and $s_2 {\cap} names(e) = \emptyset$ then*

$$s_1 \vdash e \Downarrow_T^k (s_3)\, w.$$

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

3 Equivalence and Adequacy

We develop a sound equivalence theory with respect to contextual equivalence. All the definitions and proofs in this section, with the exception (at the time of writing) of the context lemma (Lemma 3.12), which was proved by Stark [11], have been mechanized in the Coq theorem prover.

We build our theory on type-respecting relations from the domain:

$$\textsc{Ns} \times \textsc{Ns} \times \textsc{Env} \times \textsc{Exp} \times \textsc{Exp} \times \textsc{Typ},$$

and write $s, s', E \vdash e \ R \ e' : T$ when $(s, s', E, e, e', T) \in R$, and $s, s' \vdash e \ R \ e' : T$ when $(s, s', \emptyset, e, e', T) \in R$.

Contextual Equivalence is such a relation: each 6-tuple contains two identical namesets and two terms of the same type under the same typing environment.

Definition 3.1 (Contextual Equivalence ($\cong$)). Contextual equivalence *is a type-respecting relation, such that* $s, s, E \vdash e \cong e' : T$ *iff for all contexts* $C[\ ]$ *with names in* s*, which bind* $dom(E)$ *at their hole, and for all boolean values* b*:*

$$(\exists\, s_1.\ s \vdash C[e] \Downarrow_o (s_1)\, b) \iff (\exists\, s_1'.\ s \vdash C[e'] \Downarrow_o (s_1')\, b)$$

For proving some equivalence we relax the per-tuple restriction for identical namesets allowing us to reason about *related* namesets. We do impose, though, a per-relation restriction of all tuples having the same pair of namesets. Such a relation—which we call a *generalized typed relation (GTRel)*—can be seen as representing the part of the equivalence that holds under that particular pair of namesets. We therefore need to handle sets of GTRels to describe the whole equivalence. We further require that GTRels contain closed expressions.

Definition 3.2 (Generalized Typed Relations (GTRel)). *The type-respecting, non-empty relation* R *is a GTRel iff for any* $s_1, s_1', E_1 \vdash e_1 \ R \ e_1' : T_1$ *and* $s_2, s_2', E_2 \vdash e_2 \ R \ e_2' : T_2$*:*

$$s_1 = s_2 \quad \wedge \quad s_1' = s_2' \quad \wedge \quad E_1 = E_2 = \emptyset$$

We write $R_{s,s'}$ for a GTRel with tuples containing the namesets s and s'. A GTRel extends another when the former preserves the equivalences of the latter under extended stores.

Definition 3.3 (GTRel Extension ($\sqsubseteq$)). *We say that* Q extends R*, and write* $R \sqsubseteq Q$*, iff for all* $s, s' \vdash e \ R \ e' : T$ *there exist fresh* s_1 *and* s_1'*, such that*

$$s \uplus s_1, s' \uplus s_1' \vdash e \ Q \ e' : T$$

The following constructor projects all equivalences of a GTRel to extended stores:

Definition 3.4 (Projection).

$$R^{s \cup s_0, s \cup s_0'} = \{(s \cup s_0, s' \cup s_0', e, e', T) \mid (s, s' \vdash e \ R \ e' : T)\}$$

We define the GTRel of contexts with R-related values in their holes:

Definition 3.5 **($Id^{\emptyset}[R]$).** *$s, s' \vdash d[\overline{u/x}]\ Id^{\emptyset}[R]\ d[\overline{u'/x}] : T$ iff there exist $\overline{T_x}$ such that:*

$$\emptyset; \overline{x \colon T_x} \vdash d \colon T \quad \wedge \quad s, s', \vdash \overline{u}\ R\ \overline{u'} : \overline{T_x}$$

Lemma 3.6. *If R is a GTRel then $Id^{\emptyset}[R]$ is also a GTRel.*

Lemma 3.7. *If R is a GTRel, then $R \subseteq Id^{\emptyset}[R]$.*

As mentioned earlier, we prove an equivalence by reasoning about sets of GTRels. We will show that the equivalences in such a set are valid iff the set is *adequate*.

Definition 3.8 (Adequacy). *A set of GTRels, $\mathbb{X}$, is adequate iff for any $R \in \mathbb{X}$, and $s, s' \vdash e\ Id^{\emptyset}[R]\ e' : T$ the following holds:*

$$\begin{aligned} &\forall\, k, s_1, w.\ s \vdash e \Downarrow^k_T (s_1)\, w \\ &\qquad (\exists\, s'_1, w', Q\ .\ s' \vdash e' \Downarrow_T (s'_1)\, w' \\ &\qquad\qquad \wedge (T = o) \Longrightarrow (w = w') \\ &\qquad\qquad \wedge s \uplus s_1, s' \uplus s'_1 \vdash w\ Id^{\emptyset}[Q]\ w' : T \\ &\qquad\qquad \wedge R \sqsubseteq Q \\ &\qquad\qquad \wedge Q \in \mathbb{X}) \end{aligned}$$

and so does the converse.

Adequacy is a closure condition for a set $\mathbb{X}$ provable by induction on k. Following previous work [6–8], we find 'smaller' closure conditions for $\mathbb{X}$ via a *proof construction scheme*: we investigate the class of all such inductive proofs by abstracting over the actual contents of the sets and attempting a hypothetical proof. At the places where the proof can not proceed we find necessary proof obligations for $\mathbb{X}$. Our main theorem states that if $\mathbb{X}$ satisfies exactly these conditions, then it is adequate, and thus, by soundness, all terms related in $\mathbb{X}$ under same stores are contextually equivalent.

Definition 3.9 (Induction Hypotheses).

$$\begin{aligned} IH^{\triangleright}_{\mathbb{X}}(k) = \forall\, & s, s', e, e', T, R, s_1, w. \\ &\qquad (s, s' \vdash e\ Id^{\emptyset}[R]\ e' : T) \\ &\qquad \wedge (s \vdash e \Downarrow^k_T (s_1)\, w) \\ &\qquad \exists\, s'_1, w', Q\ .\ (s' \vdash e' \Downarrow_T (s'_1)\, w') \\ &\qquad\qquad \wedge (T = o) \Longrightarrow (w = w') \\ &\qquad\qquad \wedge (s \uplus s_1, s' \uplus s'_1 \vdash w\ Id^{\emptyset}[Q]\ w' : T) \\ &\qquad\qquad \wedge R \sqsubseteq Q \\ &\qquad\qquad \wedge Q \in \mathbb{X} \end{aligned}$$

Similarly we define $IH^{\triangleleft}_{\mathbb{X}}(k)$ for the induction hypothesis of the reverse direction.

Theorem 3.10 (Adequacy Conditions). *A set $\mathbb{X}$ of GTRels is adequate iff for all $R \in \mathbb{X}$, the following conditions are satisfied:*

1. *if $s, s' \vdash v \; R \; v' : o$, then $v = v'$,*
2. *if $s, s' \vdash n_1 \; R \; n_1' : \nu$, and $s, s' \vdash n_2 \; R \; n_2' : \nu$, then*
$$n_1 = n_1' \iff n_2 = n_2'$$
3. *if $n \notin s$ then there exists $n' \notin s'$ and Q, such that*
$$(s \uplus \{n\}, s' \uplus \{n'\} \vdash n \; Q \; n' : \nu) \quad \wedge \quad R \sqsubseteq Q \quad \wedge \quad Q \in \mathbb{X}$$
and the converse,
4. *if $s, s' \vdash e \; R \; e' : T$, $IH^{\triangleright}_{\mathbb{X}}(k)$ holds, and*
$$s \vdash e \Downarrow^{k+1}_{T} (s_1) \, w$$
then there exist s_1', w', and Q such that
$$(s' \vdash e' \Downarrow_T (s_1') \, w') \wedge (s \uplus s_1, s' \uplus s_1' \vdash w \; Id^{\emptyset}[Q] \; w' : T) \wedge (R \sqsubseteq Q) \wedge (Q \in \mathbb{X})$$
and the converse.
5. *if $s, s' \vdash (\lambda x{:}\, U.\, e) \; R \; (\lambda x{:}\, U.\, e') : U \to T$, $IH^{\triangleright}_{\mathbb{X}}(k)$ holds, and*
$$s, s' \vdash v \; Id^{\emptyset}[R] \; v' : U \quad \wedge \quad s \vdash e[v/x] \Downarrow^{k}_{T} (s_1) \, w$$
then there exist s_1', w', and Q such that
$$(s' \vdash e'[v'/x] \Downarrow_T (s_1') \, w') \wedge (s \uplus s_1, s' \uplus s_1' \vdash w \; Id^{\emptyset}[Q] \; w' : T) \wedge (R \sqsubseteq Q) \wedge (Q \in \mathbb{X})$$
and the converse.

Proof. The direction proving that the Adequacy Conditions imply Adequacy is done by induction on the evaluation measure k in the definition of Adequacy.

The other direction is done by inspecting that if one of the conditions is not satisfied for some $R \in \mathbb{X}$ then there is a pair of contexts in $Id^{\emptyset}[R]$ that would invalidate the definition of Adequacy.

Adequacy is defined on sets of GTRels, which contain closed expressions and related stores. We define *adequate equivalence* on open expressions and identical stores and show that it is contained in contextual equivalence.

Definition 3.11 (Adequate Equivalence $(\equiv)$). *Adequate equivalence is the largest type-respecting relation for which $s, s, \overline{x{:}\, T_x} \vdash e \equiv e' : T$ iff for all s_0, $\overline{v}$, with $s \cup s_0; \emptyset \vdash \overline{v{:}\, T_x}$, there exist adequate $\mathbb{X}$ and $R \in \mathbb{X}$ such that:*

$$s \cup s_0, s \cup s_0 \vdash e[\overline{v/x}] \; R \; e'[\overline{v/x}] : T \quad \wedge \quad \forall n \in s \cup s_0.\ s \cup s_0, s \cup s_0 \vdash n \; R \; n : \nu$$

For our proof of soundness we make use of the following context lemma, proved by Stark in [11].

Lemma 3.12 (Context Lemma). *$s, s, \overline{x{:}\, T_x} \vdash e \cong e' : T$ iff for any nameset s_0, disjoint from s, function f, and values $\overline{v}$ with:*

$$s \uplus s_0; \emptyset \vdash f{:}\, T \to o \quad \wedge \quad s \uplus s_0; \emptyset \vdash \overline{v{:}\, T_x}$$

and any boolean value b, the following holds:

$$(\exists \, s_1.\ s \uplus s_0 \vdash f \; e[\overline{v/x}] \Downarrow_o (s_1) \, b) \iff (\exists \, s_1'.\ s \uplus s_0 \vdash f \; e'[\overline{v/x}] \Downarrow_o (s_1') \, b)$$

$$\frac{}{(\emptyset,\emptyset,\emptyset,M,N,\nu\to o)\in\mathbb{X}}$$

$$\frac{R_{s,s'}\in\mathbb{X}\qquad n\notin s\qquad n'\notin s'}{R^{s\uplus\{n\},s'\uplus\{n'\}}\cup\{(s\uplus\{n\},s'\uplus\{n'\},\emptyset,n,n',\nu)\}\in\mathbb{X}}$$

$$\frac{R_{s,s'}\in\mathbb{X}\qquad s,s'\vdash M\ R\ N:\nu\to o\qquad n\notin s}{R^{s\uplus\{n\},s\cup\emptyset}\cup\{(s\uplus\{n\},s',\emptyset,\lambda x\nu x{=}n,N,\nu\to o)\}\in\mathbb{X}}$$

Fig. 2. Construction of the set $\mathbb{X}$ for the first example.

Theorem 3.13 (Soundness).

$$s,s,E\vdash e\equiv e':T\quad\Longrightarrow\quad s,s,E\vdash e\cong e':T$$

Proof. By using the context lemma 3.12 and showing that when $s{\cup}s_0,s{\cup}s_0\vdash e[\overline{v/x}]\ R\ e'[\overline{v/x}]:T$, then $s{\cup}s_0,s{\cup}s_0\vdash f\ e[\overline{v/x}]\ Id^{\emptyset}[R]\ f\ e'[\overline{v/x}]:o$.

We conjecture that adequate equivalence is also complete with respect to contextual equivalence, but have not so far succeeded in proving it due to technical complications.

4 Examples

We first show the proof of a more straightforward equivalence before proving the equivalence discussed in the introduction.

4.1 A Simple Equivalence

$$M\overset{def}{=}\nu n.\,\lambda x{:}\,\nu.\,x{=}n$$
$$N\overset{def}{=}\lambda x{:}\,\nu.\,\texttt{false}$$

The intent here is to show that a fresh name is indeed fresh; i.e. the context has no way of knowing it and provide it to the abstractions. To prove this equivalence we have to create an adequate set of GTRels that relates M and N (at least) under identical namesets, and then show that this set is closed under the conditions of Theorem 3.10.

Writing such a set, $\mathbb{X}$, here is easy—the construction is shown in Figure 2. First M and N are added to $\mathbb{X}$, under the empty namesets, by the first rule of the Figure. Then the set is closed with arbitrary related fresh locations by the second rule, satisfying condition 3 of the Theorem. Moreover this rule guarantees that the two expressions are related under any pairs of identical namesets.

Finally the values that M and N evaluate to are added to the set by the last inductive rule, satisfying condition 4. The evaluation of M produces a new

$$\frac{}{(\emptyset, \emptyset, \emptyset, M, N, (\nu \to o) \to o) \in \mathbb{X}}$$

$$\frac{R_{s,s'} \in \mathbb{X} \qquad n \notin s \qquad n' \notin s'}{R^{s \uplus \{n\}, s' \uplus \{n'\}} \cup \{(s \uplus \{n\}, s' \uplus \{n'\}, \emptyset, n, n', \nu)\} \in \mathbb{X}}$$

$$\frac{R_{s,s'} \in \mathbb{X} \qquad s, s' \vdash M \ R \ N : (\nu \to o) \to o \qquad n_1 \notin s \qquad n_2 \notin s \uplus \{n_1\}}{R^{s \uplus \{n_1, n_2\}, s \cup \emptyset} \cup \{(s \uplus \{n_1, n_2\}, s', \emptyset, U(n_1, n_2), N, (\nu \to o) \to o)\} \in \mathbb{X}}$$

$$\frac{R_{s,s'} \in \mathbb{X} \qquad s \cap s_0 = \emptyset}{R^{s \uplus s_0, s'} \in \mathbb{X}}$$

Fig. 3. Construction of the set $\mathbb{X}$ for the second example.

name, which is then added in the left-hand side store of the tuple relating the two values.

We now have to satisfy condition 5 for these values. This is immediate because the freshly generated name does not appear in related-expression position in any $R \in \mathbb{X}$, and thus in any $Id^{\emptyset}[R]$. Therefore it can never be an argument to the left-hand side procedure, hence both of them will always return false.

4.2 The 'Hard' Equivalence

Here we revisit the equivalence discussed in the introduction and prove it using the bisimulation method.

The two equivalent expressions are the following:

$$M \overset{def}{=} \nu n_1. \nu n_2. U(n_1, n_2)$$
$$N \overset{def}{=} \lambda f{:}\nu \to o. \mathtt{true}$$

where $U(x, y) \overset{def}{=} \lambda f{:}\nu \to o. (f\,x){=}(f\,y)$, and the boolean equal operator is syntactic sugar for the equivalent expression using nested conditional statements.

We start our proof as before by inductively constructing a set of GTRels, which we have to show adequate. This set is shown in Figure 3.

Lemma 4.1. *If $R \in \mathbb{X}$, then R is an HGTRel.*

Proof. By induction on the structure of $\mathbb{X}$.

Lemma 4.2. *If $R \in \mathbb{X}$, and $s, s' \vdash e \ R \ e' : T$, then one of the following is true:*

- *$e = M$, $e' = N$, and $T = (\nu \to o) \to o$, or*
- *there exist n and n', such that $e = n$, $e' = n'$, $T = \nu$, $n \in s$, and $n' \in s'$, or*
- *there exist n_1 and n_2, such that $e = U(n_1, n_2)$, $e' = N$, $T = (\nu \to o) \to o$, $n_1 \in s$, and $n_2 \in s$.*

$$\frac{}{(\emptyset,\emptyset,\emptyset,M,M,(\nu\to o)\to o)\in\mathbb{Y}}$$

$$\frac{R_{s,s}\in\mathbb{Y}\qquad n\notin s}{R^{s\uplus\{n\},s\uplus\{n\}}\cup\{(s\uplus\{n\},s\uplus\{n\},\emptyset,n,n,\nu)\}\in\mathbb{Y}}$$

$$\frac{R_{s,s}\in\mathbb{Y}\qquad s,s\vdash M\ R\ M:(\nu\to o)\to o\qquad n_1\notin s\qquad n_2\notin s\uplus\{n_1\}}{\begin{array}{l}R^{s\uplus\{n_1,n_2\},s\uplus\{n_1,n_2\}}\cup\\ \{(s\uplus\{n_1,n_2\},s\uplus\{n_1,n_2\},\emptyset,n_1,n_2,\nu)\}\cup\\ \{(s\uplus\{n_1,n_2\},s\uplus\{n_1,n_2\},\emptyset,n_2,n_1,\nu)\}\cup\\ \{(s\uplus\{n_1,n_2\},s\uplus\{n_1,n_2\},\emptyset,U(n_1,n_2),U(n_1,n_2),(\nu\to o)\to o)\}\in\mathbb{Y}\end{array}}$$

$$\frac{R_{s,s}\in\mathbb{Y}\qquad s\cap s_0=\emptyset}{R^{s\uplus s_0,s\uplus s_0}\in\mathbb{Y}}$$

Fig. 4. Construction of the auxiliary set $\mathbb{Y}$ for the second example.

Proof. By induction on the structure of $\mathbb{X}$.

Lemma 4.3. *If $R\in\mathbb{X}$, $s,s'\vdash n_1\ R\ n_1':\nu$, and $s,s'\vdash n_2\ R\ n_2':\nu$, then*

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

Proof. By induction on the structure of $\mathbb{X}$, using Lemmas 4.1 and 4.2.

To prove the equivalence of M and N we have to show that $\mathbb{X}$ satisfies the conditions for adequacy of Theorem 3.10. We take any $R\in\mathbb{X}$ and invoke Lemma 4.2. Condition 1 is trivially satisfied; condition 2 is satisfied by Lemma 4.3. Conditions 3 and 4 are obviously satisfied by construction of $\mathbb{X}$.

The interesting part of this proof is to show that when $R_{s,s'}\in\mathbb{X}$, condition 5 holds for the tuple

$$s,s'\vdash U(n_1,n_2)\ R\ N:(\nu\to o)\to o$$

We have to show that if $s,s'\vdash f\ Id^{\emptyset}[R]\ f':\nu\to o$, and $s\vdash f\,n_1{=}f'\,n_2\Downarrow_o(s_0)\,w$ then, there exists Q, such that:

$$s\uplus s_0,s'\vdash w\ Id^{\emptyset}[Q]\ \texttt{true}:o\quad\wedge\quad R\sqsubseteq Q\quad\wedge\quad Q\in\mathbb{X}$$

Proving $R\sqsubseteq Q$ and $Q\in\mathbb{X}$ for any s_0 is easily done by construction of $\mathbb{X}$. What remains to be shown is that $w=\texttt{true}$. This we do by an interesting re-use of our method.

We form an auxiliary set of GTRels, $\mathbb{Y}$, shown in Figure 4. The intuition behind the construction of $\mathbb{Y}$ is that we want for the above s, f, n_1, and n_2 to exist some $P\in\mathbb{Y}$, such that:

$$s,s\vdash(f\,n_1)\ Id^{\emptyset}[P]\ (f\,n_2):o$$

Then, by showing that $\mathbb{Y}$ is adequate, we prove that the two applications will evaluate to the same final values, and thus the boolean test $f\,n_1 = f\,n_2$ will evaluate to `true`.

We first show that for each GTRel in $\mathbb{X}$ there exists another in $\mathbb{Y}$ that contains all the left-hand side expressions and namesets of the former.

Lemma 4.4. *For all $R \in \mathbb{X}$, there exists $P \in \mathbb{Y}$, such that, for all s, s', e, e', T:*

$$s, s' \vdash e\ R\ e' : T \implies s, s \vdash e\ P\ e : T$$

Proof. By induction on the structure of $\mathbb{X}$.

This property is extended to the context closures of the relations.

Corollary 4.5. *For all $R \in \mathbb{X}$, there exists $P \in \mathbb{Y}$, such that, for all s, s', e, e', T:*

$$s, s' \vdash e\ Id^{\emptyset}[R]\ e' : T \implies s, s \vdash e\ Id^{\emptyset}[P]\ e : T$$

We can now write the property of the applications $f\,n_1$ and $f\,n_2$ discussed earlier.

Lemma 4.6. *For all $R \in \mathbb{X}$ with*

$$s, s' \vdash U(n_1, n_2)\ R\ N : (\nu \to o) \to o \quad \wedge \quad s, s' \vdash f\ Id^{\emptyset}[R]\ f' : \nu \to o$$

there exists $P \in \mathbb{Y}$, such that

$$s, s \vdash (f\,n_1)\ Id^{\emptyset}[P]\ (f\,n_2) : o$$

Proof. Trivial by Lemmas 4.4, 4.5, and 3.7.

What remains to be shown is that $\mathbb{Y}$ is adequate and then use this property to reason about the final values of the two applications.

Lemma 4.7. *If $P \in \mathbb{Y}$, then P is an GTRel.*

Proof. By induction on the structure of $\mathbb{Y}$.

Lemma 4.8. *If $P \in \mathbb{Y}$, and $s, s' \vdash e\ P\ e' : T$, then $s = s'$, and one of the following is true:*

- *$e = M$, $e' = M$, and $T = (\nu \to o) \to o$, or*
- *there exist n and n', such that $e = n$, $e' = n'$, $T = \nu$, $n \in s$, and $n' \in s'$, or*
- *there exist n_1 and n_2, such that $e = U(n_1, n_2)$, $e' = U(n_1, n_2)$, $n_1, n_2 \in s$, $T = (\nu \to o) \to o$, and*

$$s, s' \vdash n_1\ P\ n_2 : \nu \quad \wedge \quad s, s' \vdash n_2\ P\ n_1 : \nu$$

Proof. By induction on the structure of $\mathbb{Y}$.

Lemma 4.9. *If $P \in \mathbb{Y}$, $s, s' \vdash n_1\ P\ n_1' : \nu$, and $s, s' \vdash n_2\ P\ n_2' : \nu$, then*

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

Proof. By induction on the structure of $\mathbb{Y}$, using Lemmas 4.7 and 4.8.

Lemma 4.10. $\mathbb{Y}$ *is adequate.*

Proof. By showing that $\mathbb{Y}$ satisfies the conditions of Theorem 3.10, using Lemmas 4.7, 4.8, and 4.9.

We can now complete the proof of adequacy for $\mathbb{X}$. For any $R_{s,s'} \in \mathbb{X}$, with

$$s, s' \vdash U(n_1, n_2)\ R\ N : (\nu \to o) \to o \quad \wedge \quad s, s' \vdash f\ Id^{\emptyset}[R]\ f' : \nu \to o$$

we know that there exist s_0, w, such that

$$s \vdash f\, n_1 {=} f\, n_2 \Downarrow_o (s_0)\, w$$

which implies that there exist s_1,s_2, w_1, and w_2 such that $s_0 = s_1 \uplus s_2$ and

$$s \vdash f\, n_1 \Downarrow_o (s_1)\, w_1 \tag{2}$$
$$s \uplus s_1 \vdash f\, n_2 \Downarrow_o (s_2)\, w_2 \tag{3}$$

and by Lemma 2.4 and (3) we get

$$s \vdash f\, n_2 \Downarrow_o (s_2)\, w_2 \tag{4}$$

Moreover, by Lemma 4.6, there exists $P \in \mathbb{Y}$, such that

$$s, s \vdash (f\, n_1)\ Id^{\emptyset}[P]\ (f\, n_2) : o$$

and from Lemma 4.10, Definition 3.8, (2), and (4) we get $w_1 = w_2$. Thus, the result of the boolean equality test, w, is `true`, and the terms are contextually equivalent.

5 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 both the metatheory of Section 3 and the examples given in Section 4.

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 totality Lemma (2.2) and the determinacy Lemma (2.1). These are entirely standard results and proofs, the mechanization of which is not especially interesting, though we do intend to do it in due course just so as to have a complete self-contained development.

5.1 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.

5.2 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 3.5. We encode it in two parts.

First we construct the $[\overline{v/x}]$ and $[\overline{v'/x}]$ of Definition 3.5 by defining an inductive relation on 'synchronized' environments and substitutions containing closed expressions from a GTRel 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` GTRel, 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 $Id^{\emptyset}[R]$. 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 $Id^{\emptyset}[R]$, `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

6 Conclusions and Future Work

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.

On the theoretical side, the most obvious next step is investigate completeness of our method. Previous work on similar bisimulations for related calculi gives us good reason to believe that the method is complete, but some technical difficulties have so far prevented us from a straightforward adaptation of previous proofs.

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. Samson Abramsky, Dan Ghica, Andrzej Murawski, Luke Ong, and Ian Stark. Nominal games and full abstraction for the nu-calculus. In *Proceedings of the Nineteenth Annual IEEE Symposium on Logic in Computer Science*, volume 00, pages 150–159, Los Alamitos, CA, USA, 2004. IEEE Computer Society Press.
2. Brian Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. Engineering formal metatheory. To appear in POPL 2008, July 2007.
3. Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The POPLmark Challenge. In *Proceedings of TPHOLs 2005: the 18th International Conference on Theorem Proving in Higher Order Logics (Oxford), LNCS 3603*, pages 50–65, August 2005.
4. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. Springer-Verlang, 2004.
5. A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *LICS '99: Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 1999.
6. V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *Proceedings 33rd ACM Symposium on Programming Languages*, pages 141–152, January 2006.
7. Vasileios Koutavas and Mitchell Wand. Bisimulations for untyped imperative objects. In Peter Sestoft, editor, *Proc. ESOP 2006*, volume 3924 of *Lecture Notes in Computer Science*, pages 146–161, Berlin, Heidelberg, and New York, 2006. Springer-Verlag.
8. Vasileios Koutavas and Mitchell Wand. Reasoning about class behavior. Appeared in the FOOL/WOOD 2007 workshop, January 2007.
9. J. Laird. A game semantics of names and pointers. In *FoSSaCS '04: Foundations of Software Science and Computation Structures*, 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*, volume 711 of *Lecture Notes in Computer Science*, pages 122–141. Springer-Verlag, 1993.
11. Ian Stark. *Names and Higher-Order Functions.* PhD thesis, University of Cambridge, December 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*, pages 161–172, New York, NY, USA, 2004. ACM Press.
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*, pages 63–74, New York, NY, USA, 2005. ACM Press.
14. Y. Zhang and D. Nowak. Logical relations for dynamic name creation. In *CSL '03: Proceedings of 17th International Workshop on Computer Science Logic*, volume 2803 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.