Program equivalence with names

Nikos Tzevelekos

Oxford University Computing Laboratory

Abstract. The nu-calculus of Pitts and Stark was introduced as a paradigmatic functional language with a very basic local-state effect: references of unit type. These were called names, and the motto of the new language went as follows.
"Names are created with local scope, can be tested for equality, and are passed around via function application, but that is all."
Because of this limited framework, the hope was that fully abstract models and complete proof techniques could be obtained. However, it was soon realised that the behaviour of nu-calculus programs is quite intricate, and program equivalence in particular is surprisingly difficult to capture. Here we focus on the following "hard" equivalence,

$$\mathsf{new}\ x, y\ \mathsf{in}\ \lambda f.(fx = fy) \quad \equiv \quad \lambda f.\,\mathsf{true}$$

where x, y are names, and $f : \mathsf{Name} \to \mathsf{Bool}$. We examine attempts and proofs of the above, explain the advantages and disadvantages of the proof methods and discuss why program equivalence in this simple language remains to date a mystery.

1 Introduction

Names constitute a pervasive feature in programming languages. They appear in every computational scenario where entities of specific kinds can be created at will and, moreover, in such a manner that newly created entities are always *fresh* — distinct from any other created thus far. They are used for expressing a large variety of features, e.g. references and exceptions in languages like ML or C. The behaviour of languages which feature names is in general very subtle due to issues of privacy, visibility and flow of names, and the ensuing notion of local state.

In order to study this behaviour in higher-order languages, and in isolation from other computational effects, Pitts and Stark introduced in the early 90's the nu-calculus [6]: a simple higher-order call-by-value functional language with references of unit type which cannot be assigned values nor be dereferenced. Those entities, following a long tradition in computer science, were called *names*.

> *[...] names are created with local scope, can be tested for equality and can be passed around via function application, but that is all.*

The above motto [6] is the basis of computation with names, on which further effects can be built (e.g. by allowing names to be assigned values, dereferenced, raised as exceptions, etc.).

Dagstuhl Seminar Proceedings 10351
Modelling, Controlling and Reasoning About State
http://drops.dagstuhl.de/opus/volltexte/2010/2809

Although the language was designed with simplicity in mind, it was soon realised that it incorporated genuinely intricate effects which in turn resulted in quite delicate behaviours [7]:

> *Functions may have local names that remain private and persist from one use of the function to the next; alternatively, names may be passed out of their original scope and can even outlive their creator. It is precisely this mobility of names that allows the nu-calculus to model issues of locality, privacy and non-interference.*

Research focused primarily on the notion of observational equivalence, which revealed important and perhaps unexpected properties of nominal computation.

- The nu-calculus has a type `Name` for names. In empty context there is only one equivalence class in `Name`, represented by the term that produces a fresh name. However, this simple behaviour changes drastically when one moves to the type $\texttt{Name} \to \texttt{Name}$: here there are infinitely many observationally distinct terms, represented by (closed or open) *name-chains* of arbitrary size.
- Standard proof methods for observational equivalence did not seem to migrate to this new setting, and in particular low-order equivalences like

$$\mathsf{new}\ x, y\ \mathsf{in}\ \lambda f.(fx = fy) \quad \equiv \quad \lambda f.\,\mathsf{true} \tag{$*$}$$

  turned out to be remarkably difficult to prove.

Hence, this seemingly plain language became of great importance. Here we focus on attempts and proofs of $(*)$ that have appeared in the literature, each of which hailing from different traditions in semantics. We present Stark's original proof [7], based on logical relations for names; we demonstrate a complete proof method developed by Benton and Koutavas [2], based on environmental bisimulations; and finally we present a proof by Abramsky, Ghica, Murawski, Ong and Stark using game semantics [1], we show it not to be entirely correct and provide its rectification.

2 The nu-calculus

The nu-calculus is a simply-typed lambda-calculus built over base types for names and booleans. Types are given by:

$$T \ ::= \ \texttt{Name} \mid \texttt{Bool} \mid T \to T$$

Assume distinct countably infinite sets $\mathbb{V}$ and $\mathbb{A}$ containing variables and names respectively, and let $\mathbb{B} = \{\texttt{true}, \texttt{false}\}$ be the set of boolean constants. Terms are given by:

$$t \ ::= \ x \mid a \mid \texttt{b} \mid \lambda x.t \mid t\,t \mid \texttt{if}\ t\ \texttt{then}\ t\ \texttt{else}\ t \mid t == t \mid \nu a.t$$

where $x \in \mathbb{V}$, $a \in \mathbb{A}$ and $\texttt{b} \in \mathbb{B}$.

The typing rules are just like in the simply-typed lambda-calculus with conditionals, with additional rules for names. The contexts are now pairs (S, Γ), where Γ is a finite variable context of the form $\{x_1 : T_1, \ldots, x_n : T_n\}$ and S is a finite subset of $\mathbb{A}$. Some of the typing rules, including the non-standard ones, are given below.

$$\frac{(x : T) \in \Gamma}{S; \Gamma \vdash x : T} \qquad \frac{a \in S}{S; \Gamma \vdash a : \mathtt{Name}} \qquad \frac{S \uplus \{a\}; \Gamma \vdash t : T}{S; \Gamma \vdash \nu a.t : T}$$

$$\frac{S; \Gamma, x : T \vdash t : T'}{S; \Gamma \vdash \lambda x.t : T \to T'} \qquad \frac{S; \Gamma \vdash t_1 : \mathtt{Name} \quad S; \Gamma \vdash t_2 : \mathtt{Name}}{S; \Gamma \vdash t_1 == t_2 : \mathtt{Bool}}$$

Note that there are two notions of abstraction: variable- and name-abstraction (via λ and ν respectively). We write $\mathsf{fn}(t)$ for the set of free names in term t. Terms are equated up to α-equivalence for both notions.

The operational semantics is given by means of a small-step transition relation with elements of the form:

$$(S, t) \to (S', t')$$

with S, S' being finite subsets of $\mathbb{A}$, and such that $S \subseteq S'$, $\mathsf{fn}(t) \subseteq S$ and $\mathsf{fn}(t') \subseteq S'$. The semantics is call-by-value, with values given by:

$$v ::= \; x \mid a \mid \mathtt{true} \mid \mathtt{false} \mid \lambda x.t$$

We write $Val_T(S)$ for the set of values v which can be typed as $S; \emptyset \vdash v : T$. The reduction rules are as follows (plus rules for conditionals),

$$(S, (\lambda x.t)v) \to (S, t[v/x]) \qquad (S, a == a) \to (S, \mathtt{true}) \qquad \frac{(S, t) \to (S', t')}{(S, E[t]) \to (S', E[t'])}$$
$$(S, \nu a.t) \to (S \uplus \{a\}, t) \qquad (S, a == a') \to (S, \mathtt{false})$$

with a, a' being distinct names, and evaluation contexts given by:

$$E \; ::= \; \_t \mid (\lambda x.t)\_ \mid \_ == t \mid a == \_ \mid \cdots$$

Note that, due to α-equivalence, the rule for fresh-name creation is non-deterministic: the ν-abstraction can create any name $a' \notin S$.

$$(S, \nu a.t) \to (S \uplus \{a'\}, t[a'/a])$$

For any pair (S, t) with $\mathsf{fn}(t) \subseteq S$ we write $(S, t) \Downarrow v$ if there exists S' such that $(S, t) \twoheadrightarrow (S', v)$. Finally, program equivalence is defined by means of *observational (or contextual) equivalence*.

Definition 1. We say that typed term $S; \Gamma \vdash t_1 : T$ is ***equivalent*** to $S; \Gamma \vdash t_2 : T$ (written $S; \Gamma \vdash t_1 \cong t_2 : T$) if

$$(S, C[t_1]) \Downarrow \mathtt{true} \iff (S, C[t_2]) \Downarrow \mathtt{true}$$

for any context $C[-]$ such that $S; \emptyset \vdash C[t_1], C[t_2] : \mathtt{Bool}$.

Example 2 (Sample equivalences I). The following equivalences express some very basic nominal features.

$$\nu a.t \;\cong\; t : T \;\text{ if } a \notin \mathsf{fn}(t) \tag{1}$$

$$\nu a.\nu a'.t \;\cong\; \nu a'.\nu a.t : T \tag{2}$$

$$\nu a.\lambda x.\, a \;\ncong\; \lambda x.\nu a.\, a : \mathtt{Bool} \to \mathtt{Name} \tag{3}$$

In particular, creating a fresh name that is not used is equivalent to not creating it at all and, on the order hand, order is not important in name creation. Moreover, λ- and ν-abstractions do not commute. For example, the context

$$(\lambda f.\, (f\mathtt{true}) == (f\mathtt{true})) -$$

separates the terms in (3). (1) and (2) are proven in [7].

Example 3 (Sample equivalences II). More interesting equivalences are the following. Here "=" is boolean equality, expressed by use of conditionals.

$$\nu a.\lambda x.\, (x == a) \;\cong\; \lambda x.\, \mathtt{false} : \mathtt{Name} \to \mathtt{Bool} \tag{4}$$

$$\nu a_1, a_2.\, \lambda f.\, (f a_1 = f a_2) \;\cong\; \lambda f.\, \mathtt{true} : (\mathtt{Name} \to \mathtt{Bool}) \to \mathtt{Bool} \tag{5}$$

These are also proven in [7] but they are significantly more intricate. The former expresses the fact that a context environment can never guess an unrevealed private name. The latter is more subtle; in rough terms it states that a context function (of name-less output type) cannot distinguish between different private names fed to it. This rough argument may seem convincing at first, but it turns out that things are a bit more complicated: f may not be able to distinguish between a_1 and a_2 when it is first called, but it may do it within a recursive call. The equivalence holds nonetheless, because overall there is perfect symmetry between the LHS and RHS of the comparison $f a_1 = f a_2$. This will be made precise in the following sections. For now, the reader may want to note the difference between (5) and:

$$\nu a_1.\, \lambda f.\, \nu a_2.\, (f a_1 = f a_2) \;\ncong\; \lambda f.\, \mathtt{true} : (\mathtt{Name} \to \mathtt{Bool}) \to \mathtt{Bool} \tag{6}$$

These are separated e.g. by the context (note the recursive call of G):

$$(\lambda G.\, G(\lambda x.\, G(\lambda y.\, x == y))) - \tag{7}$$

The rest of the paper examines three different methods which have been developed for proving program equivalences in the nu-calculus, and how in particular do they prove (5).

3 Stark's proof

The first proof of (5) was given in Stark's PhD thesis [7], which presents a meticulous study of higher-order computation with names. Chapter 4 of the

thesis examines operational logical relations for deciding observational equivalence. It presents two such formulations: logical relations and predicated logical relations. Logical relations are shown to be sound but not generally complete: completeness is proven only for terms of first-order type. Therefore, they capture (4) but fail to verify (5). Stark then proceeds by extending his setting to predicated logical relations, which successfully capture the equivalence although they do not lead to a better completeness result.

3.1 Logical Relations

Because names are created dynamically, equivalent terms may evaluate to terms with distinct names which, however, have equivalent roles inside the terms. Hence, logical relations between terms with names are built around the notion of *span*: a bijective mapping between names appearing inside the terms. Formally, a span R between sets of names S_1, S_2, written

$$R : S_1 \rightleftharpoons S_2$$

is a relation $R \subseteq S_1 \times S_2$ such that, $\forall (a_1, a_2), (a_1', a_2') \in R.\ a_1 = a_1' \iff a_2 = a_2'$. We may write $a_1\, R\, a_2$ instead of $(a_1, a_2) \in R$. From the notion of span we move to logical relations as follows.

Definition 4. For each $R : S_1 \rightleftharpoons S_2$ define the ***logical relation***

$$R_T \subseteq Val_T(S_1) \times Val_T(S_2)$$

by induction on T as follows.

$$\begin{aligned}
\mathtt{b}_1\ R_{\mathtt{Bool}}\ \mathtt{b}_2 &\equiv \mathtt{b}_1 = \mathtt{b}_2 \\
a_1\ R_{\mathtt{Name}}\ a_2 &\equiv a_1\ R\ a_2 \\
(\lambda x.t_1)\ R_{T\to T'}\ (\lambda x.t_2) &\equiv \forall R' : S_1' \rightleftharpoons S_2',\ v_i \in Val(S_i \uplus S_i').\\
&\qquad v_1\ (R \uplus R')_T\ v_2 \implies t_1[v_1/x]\ (R \uplus R')^*_{T'}\ t_2[v_2/x]
\end{aligned}$$

Here R^*_T is the *closure* of R_T to general closed terms of type T, given by:

$$t_1\, R^*_T\, t_2 \equiv \exists R' : S_1' \rightleftharpoons S_2', v_i \in Val(S_i \uplus S_i').\, (S_i, t_i) \twoheadrightarrow (S_i', v_i) \wedge v_1\, (R \uplus R')_T\, v_2$$

Stark shows that logical relations are complete at first-order types. Below we write $\mathsf{id} : S \rightleftharpoons S$ for the span which coincides with the identity function $\mathsf{id} : S \to S$.

Theorem 5 ([7]). *For any $S; \emptyset \vdash t_1, t_2 : T$, if $t_1\, \mathsf{id}^*_T\, t_2$ then $S; \emptyset \vdash t_1 \cong t_2 : T$. Moreover, if T is of order at most 1 and all types in Γ are of order 0 then the converse also holds.*

In particular, (note that types are omitted for economy)

$$\text{we have}\ \ \nu a.\lambda x.\, (x == a)\ \mathsf{id}^*\ \lambda x.\, \mathtt{false} \tag{8}$$

$$\text{but not}\ \ \nu a_1, a_2.\, \lambda f.\, (f a_1 = f a_2)\ \mathsf{id}^*\ \lambda f.\, \mathtt{true}\,. \tag{9}$$

Note that in both cases the underlying spans are empty. In order to show (8), by definition, it suffices to show that $\lambda x.\,(x == a)\ R\ \lambda x.\,\texttt{false}$, where $R : \{a\} \rightleftharpoons \emptyset$ is the empty span. For this to hold, it must be that the two functions evaluate to the same boolean value once they are fed R-related inputs. But note that those inputs would not include a, and therefore $\lambda x.\,(x == a)$ would return `false`. On the other hand, for (9) to hold it would be necessary to have $\lambda f.\,(fa_1 = fa_2)\ R\ \lambda f.\,\texttt{true}$, for $R : \{a_1, a_2\} \rightleftharpoons \emptyset$ the empty span. But note now that, reasoning as above, we have $\lambda x.\,(x == a_1)\ R\ \lambda x.\,\texttt{false}$, and if we use the latter as arguments to $\lambda f.\,(fa_1 = fa_2)$ and $\lambda f.\,\texttt{true}$ respectively then we get different values.

3.2 Predicated Logical Relations

The reason why (5) holds is that there is a symmetry between the names a_1 and a_2 on the LHS of the equivalence. Stark extended the notion of span in such a way that symmetries, and all predicated relations on the sets of names of related terms, are captured. An *augmented span* $\hat{R} : S_1 \rightleftharpoons S_2$ is a triple of ordinary spans (R_1, R, R_2) of the form:

$$\hat{R} = (S_1 \overset{R_1}{\rightleftharpoons} S_1 \overset{R}{\rightleftharpoons} S_2 \overset{R_2}{\rightleftharpoons} S_2)$$

Augmented spans are the basis of predicated logical relations.

Definition 6. For each $\hat{R} : S_1 \rightleftharpoons S_2$ define the ***predicated logical relation***

$$\hat{R}_T \subseteq \mathit{Val}_T(S_1) \times \mathit{Val}_T(S_2)$$

by induction on T as follows.

$$\begin{aligned}
\mathtt{b}_1\ \hat{R}_{\texttt{Bool}}\ \mathtt{b}_2 &\equiv \mathtt{b}_1 = \mathtt{b}_2 \\
a_1\ \hat{R}_{\texttt{Name}}\ a_2 &\equiv a_1\ R_1\ a_1\ R\ a_2\ R_2\ a_2 \\
(\lambda x.t_1)\ \hat{R}_{T\to T'}\ (\lambda x.t_2) &\equiv (\lambda x.t_i)\,(R_i)_{T\to T'}\,(\lambda x.t_i) \wedge \forall \hat{R}' : S_1' \rightleftharpoons S_2', v_i \in \mathit{Val}(S_i \uplus S_i').\\
&\qquad v_1\,(\hat{R} \uplus \hat{R}')_T\,v_2 \implies t_1[v_1/x]\,(\hat{R} \uplus \hat{R}')^*_{T'}\,t_2[v_2/x]
\end{aligned}$$

Here $\hat{R}^*_T$ is the *closure* of $\hat{R}_T$ to general closed terms of type T, given by:

$$t_1\,\hat{R}^*_T\,t_2 \equiv \exists \hat{R}' : S_1' \rightleftharpoons S_2', v_i \in \mathit{Val}(S_i \uplus S_i').\,(S_i, t_i) \twoheadrightarrow (S_i', v_i) \wedge v_1\,(\hat{R} \uplus \hat{R}')_T\,v_2$$

Theorem 7 ([7]). *For any $S; \emptyset \vdash t_1, t_2 : T$, if $t_1\,\hat{\mathsf{id}}^*_T\,t_2$ then $S; \emptyset \vdash t_1 \equiv t_2 : T$. Moreover, if T is of order at most 1 and all types in Γ are of order 0 then the converse also holds.*

We can now present the first proof of equivalence (5).

Proof 8 (Stark) By the previous theorem, it suffices to show

$$\nu a_1,a_2.\,\lambda f.\,(fa_1 = fa_2)\ \hat{\mathsf{id}}^*\ \lambda f.\,\texttt{true}$$

where $\mathsf{id} : \emptyset \rightleftharpoons \emptyset$ the empty span, and for the latter it suffices to show that $\lambda f.\,(fa_1 = fa_2)\ \hat{R}\ \lambda f.\,\mathtt{true}$, where now we choose $\hat{R} : \{a_1, a_2\} \rightleftharpoons \emptyset$ to be given by the triple:

$$R = R_2 = \emptyset\,, \quad R_1 = \{(a_1, a_2), (a_2, a_1)\}$$

that is, an empty span with a symmetry at the left hand. But, by definition 6 (case for λ-abstractions), any relevant input v_1 for $\lambda f.\,(fa_1 = fa_2)$ must be R_1-related to itself and therefore $v_1 a_1$ and $v_1 a_2$ would be deemed to give the same value. Thus,

$$v_1\,(\hat{R} \uplus \hat{R}')\,v_2 \implies (v_1 a_1 = v_1 a_2) \Downarrow \mathtt{true}$$

which implies $\lambda f.\,(fa_1 = fa_2)\ \hat{R}\ \lambda f.\,\mathtt{true}$, as required. □

The proof method just presented is advantageous in that it is notionally simple and robust. Moreover, it is constructive in the sense that predicated relations can be derived from the spans without much guessing: the only step in the definition when we are required to guess is in the case of $*$-closure but this can be overcome by first evaluating and then considering all the possible spans for the yielded name-sets. On the other hand, the method is limited in that it is complete only up to first-order types, and its extension to predicated relations seems to be very specific to cases like (5) and does not give better completeness.

4 Benton & Koutavas' proof

Another proof with operational flavour, but with full completeness power, was originally presented in 2007 by Benton and Koutavas [2]. Their method invokes *environmental bisimulations* [8, 5], a notion of bisimulation defined not as a single relation but rather as a set of relations. This expresses the fact that the information available to the environment changes during the bisimulation test. In particular, the bisimulations considered are annotated with sets of names and relate terms containing names from those sets. As the knowledge of names available to the environment increases during computation, the bisimulation sets invoke larger bisimulations.

4.1 Adequate bisimulation sets and full completeness

Consider relations annotated with sets of names. Formally, an *annotated relation* is a triple

$$(S_1, S_2, R)$$

where R is an infinite product of relations, one for each type T:

$$R = \langle\, R_T \,\rangle_{\text{all types } T} \qquad R_T \subseteq \mathit{Val}_T(S_1) \times \mathit{Val}_T(S_2)$$

relating closed values of T. An annotated relation R is extended to closed terms in the following way (note that $\overline{T}$ and T are unrelated notations).

$$\frac{\emptyset; \overline{x : T} \vdash s : T \qquad \overline{v_1\ R_T\ v_2}}{s[\overline{v_1/x}]\ \ R_T^*\ \ s[\overline{v_2/x}]}$$

Observe that here closures are purely contextual, a formulation quite different to that of Stark, which reduces terms to related values. We now consider sets $\mathcal{X}$ of annotated relations. The notion which allows us to capture observational equivalence is the following.

Definition 9. A set of annotated relations $\mathcal{X}$ is ***semi-adequate*** if, for all $(S_1, S_2, R) \in \mathcal{X}$ and all t_1, t_2, S_1', v_1:

$$\begin{aligned} t_1\, R_T^*\, t_2 \;\wedge\; (S_1, t_1) \twoheadrightarrow (S_1', v_1) \implies \exists S_2', v_2, Q\,.\;\; & R \subseteq Q \wedge v_1\, Q_T^*\, v_2 \\ & \wedge\; (S_1 \uplus S_1', S_2 \uplus S_2', Q) \in \mathcal{X} \\ & \wedge\; (S_2, t_2) \twoheadrightarrow (S_2', v_2) \\ & \wedge\; (T = \mathtt{Bool}) \implies (v_1 = v_2) \end{aligned}$$

The inverse of $\mathcal{X}$ is given by $\mathcal{X}^{-1} = \{(S_1, S_2, R) \mid (S_2, S_1, R^{-1}) \in \mathcal{X}\}$. We say that $\mathcal{X}$ is ***adequate*** if both $\mathcal{X}$ and $\mathcal{X}^{-1}$ are semi-adequate.

The above formulation is reminiscent of Stark's logical relations but there are substantial differences. First, we have moved from single relations R to sets of relations $\mathcal{X}$. Moreover, the (sets of) relations are not constructed inductively from their base specifications (their spans), but are rather specified coinductively.

From a set of relations $\mathcal{X}$ we can can obtain a relation between open terms by 'flattening' $\mathcal{X}$ in the following sense.

Definition 10. Given a set $\mathcal{X}$ of annotated relations we define a relation $(\mathcal{X})^\circ$ on typed open terms by setting:

$$S; \overline{x : T} \vdash t_1\, (\mathcal{X})^\circ\, t_2 : T \;\equiv\; \exists (S, S, R) \in \mathcal{X}.\; \lambda\overline{x}.t_1\, R_{\overline{T} \to T}\, \lambda\overline{x}.t_2 \wedge \forall a \in S.\, a\, R_{\mathtt{Name}}\, a$$

Benton and Koutavas show that the latter notion applied to adequate $\mathcal{X}$ is sound and complete with respect to observational equivalence at all types.

Theorem 11 ([2]). *For all typed terms $S; \Gamma \vdash t_1, t_2 : T$:*

$$t_1 \cong t_2 \iff \exists \mathcal{X}.\; \mathcal{X} \text{ adequate} \wedge t_1 (\mathcal{X})^\circ t_2$$

4.2 A simpler proof method

Theorem 11 yields a proof method for verifying equivalences. In order to show that $S; \Gamma \vdash t_1 \cong t_2 : T$, where $\Gamma = \overline{x : T}$, we follow the steps below.

1. Find a set $\mathcal{X}$ containing (S, S, R) such that $\lambda\overline{x}.t_1\, R_{\overline{T} \to T}\, \lambda\overline{x}.t_2$ and $a\, R_{\mathtt{Name}}\, a$ for all $a \in S$.
2. Show that $\mathcal{X}$ is adequate.

The former step requires some guessing. However, the really difficult part is the latter step: we need to verify the condition of Definition 9 for all t_1, t_2 related by the closure of R. Benton and Koutavas produce a simpler proof method by parameterising the condition of Definition 9 by the number of reduction steps in the evaluation $(S_1, t_1) \twoheadrightarrow (S_1', v_1)$, which allows one to prove adequacy by induction on this parameter. The induction hypothesis for the induction is given as follows.

Definition 12. For each set $\mathcal{X}$ of annotated relations and each $k \in \omega$ define $\mathsf{IH}_{\mathcal{X}}(k)$ to be the following condition. For all $(S_1, S_2, R) \in \mathcal{X}$ and all t_1, t_2, S_1', v_1:

$$\begin{aligned} t_1 \, R_T^* \, t_2 \;\wedge\; (S_1, t_1) \twoheadrightarrow_k (S_1', v_1) \implies \exists S_2', v_2, Q \,.\; & R \subseteq Q \wedge v_1 \, Q_T^* \, v_2 \\ & \wedge\; (S_1 \uplus S_1', S_2 \uplus S_2', Q) \in \mathcal{X} \\ & \wedge\; (S_2, t_2) \twoheadrightarrow (S_2', v_2) \\ & \wedge\; (T = \mathtt{Bool}) \implies (v_1 = v_2) \end{aligned}$$

where $\twoheadrightarrow_k$ is the reflexive transitive closure of $\rightarrow$ with transitivity bounded at k steps.

Thus, in order to prove that $\mathcal{X}$ is adequate it suffices to show that $\mathsf{IH}_{\mathcal{X}}(k)$ and $\mathsf{IH}_{\mathcal{X}^{-1}}(k)$ hold, by induction on k. The base cases are trivial, so all we are left to prove is the induction steps. Spelling out explicitly what the latter means, and removing some clear cases, one can prove the following.

Theorem 13 ([2]). *A set of annotated relations $\mathcal{X}$ is adequate if and only if for all $k \in \omega$ and all $(S_1, S_2, R) \in \mathcal{X}$ if $\mathsf{IH}_{\mathcal{X}}(k-1)$ holds then the following conditions are satisfied.*

1. *For all $\mathtt{b}_1 \, R_{\mathtt{Bool}} \, \mathtt{b}_2$, $\mathtt{b}_1 = \mathtt{b}_2$.*
2. *For all $\lambda x.t_1 \; R_{T \to T'} \; \lambda x.t_2$ and all v_1, v_2, S_1', v_1' with $(S_1, (\lambda x.t_1)v_1) \twoheadrightarrow_k (S_1', v_1')$ and $v_1 \, R_T^* \, v_2$, there exist S_2', v_2', Q such that:*

   $$R \subseteq Q \;\wedge\; v_1' \, Q_{T'}^* \, v_2' \;\wedge\; (S_1 \uplus S_1', S_2 \uplus S_2', Q) \in \mathcal{X} \;\wedge\; (S_1, (\lambda x.t_2)v_2) \twoheadrightarrow (S_2', v_2')$$

3. *For all $a_1 \notin S_1$ there exist $a_2 \notin S_2$ and Q such that:*

   $$R \subseteq Q \;\wedge\; a_1 \, Q_{\mathtt{Name}} \, a_2 \;\wedge\; (S_1 \uplus \{a_1\}, S_2 \uplus \{a_2\}, Q) \in \mathcal{X}$$

4. *For all $a_1 \, R_{\mathtt{Name}} \, a_2$ and $a_1' \, R_{\mathtt{Name}} \, a_2'$, $a_1 = a_1' \iff a_2 = a_2'$.*

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

We proceed to the proof of (5). We take a shortcut with respect to the proof presented in [2] by using a lemma regarding order of evaluation in the nu-calculus. Let us set

$$U_{a_1 a_2} \equiv \; \lambda f. \, (f a_1 = f a_2)$$

with $f : \mathtt{Name} \to \mathtt{Bool}$.

Lemma 14. *For all $a_1, a_2 \in \mathbb{A}$, $U_{a_1 a_2} \cong U_{a_2 a_1}$.*

Proof. It suffices to show that $t[a_1/x] = t[a_2/x] \cong t[a_2/x] = t[a_1/x]$, for any term t, which is straightforward. □

Proof 15 (Benton & Koutavas) It suffices to relate $\lambda y.\tau_1$ with $\lambda y.\tau_2$, where

$$\tau_1 \;\equiv\; \nu a_1, a_2 . \, U_{a_1 a_2} \,, \qquad \tau_2 \;\equiv\; \lambda f. \, \mathtt{true} \,.$$

and y is a dummy variable of type $\mathtt{Bool}$. Define $\mathcal{X}$ to be the set of annotated relations given by the following rules.

$$\frac{}{(\emptyset,\emptyset,\{\,(\lambda y.\tau_1\,,\lambda y.\tau_2)\,\})\in\mathcal{X}}\ \mathcal{X}_1 \qquad \frac{(S_1,S_2,R)\in\mathcal{X} \qquad S_1\cap\{a_1,a_2\}=\emptyset}{(S_1\uplus\{a_1,a_2\},S_2,R\cup\{\,(U_{a_1a_2},\tau_2)\,\})\in\mathcal{X}}\ \mathcal{X}_3$$

$$\frac{(S_1,S_2,R)\in\mathcal{X} \quad R':S_1'\rightleftharpoons S_2' \quad S_i\cap S_i'=\emptyset}{(S_1\uplus S_1',S_2\uplus S_2',R\uplus R')\in\mathcal{X}}\ \mathcal{X}_{2,4}$$

We proceed to show that $\mathcal{X}$ is adequate. It will suffice to show that the conditions of Theorem 13 hold for $\mathcal{X}$. Conditions 1 and 4 are trivially satisfied, and condition 3 is taken care of by rule $\mathcal{X}_{2,4}$. We are left with condition 2. Note that the terms which appear in $\mathcal{X}$ are determined by rules $\mathcal{X}_1$ and $\mathcal{X}_3$; we examine each case separately. Suppose $(S_1,S_2,R)\in\mathcal{X}$ and that $\mathsf{IH}_{\mathcal{X}}(k-1)$ holds.

• Let $\lambda y.\tau_1\,R_T\,\lambda y.\tau_2$, with $T=\mathtt{Bool}\to(\mathtt{Name}\to\mathtt{Bool})\to\mathtt{Bool}$, and let $\mathtt{b}_1\,R^*_{\mathtt{Bool}}\,\mathtt{b}_2$. By construction of $\mathcal{X}$, $\mathtt{b}_1=\mathtt{b}_2$. Moreover,

$$\begin{aligned}(S_1,(\lambda y.\tau_1)\,\mathtt{b}_1)&\twoheadrightarrow(S_1\uplus\{a_1,a_2\},U_{a_1a_2})\\(S_2,(\lambda y.\tau_2)\,\mathtt{b}_2)&\twoheadrightarrow(S_2,\tau_2)\end{aligned}$$

and now observe that $(S_1\uplus\{a_1,a_2\},S_2,R\cup\{\,(U_{a_1a_2},\tau_2)\,\})\in\mathcal{X}$ by $\mathcal{X}_3$ so we can take $Q=R\cup\{\,(U_{a_1a_2},\tau_2)\,\}$.

• Let $U_{a_1a_2}\,R_{T\to T'}\,\tau_2$, with $T=\mathtt{Name}\to\mathtt{Bool}$ and $T'=\mathtt{Bool}$, and let $v_1\,R^*_T\,v_2$. Then, $a_1,a_2\in S_1$ and there are some $S',\mathtt{b}$ such that:

$$\begin{aligned}(S_1,U_{a_1a_2}\,v_1)&\twoheadrightarrow(S_1\uplus S',\mathtt{b})\\(S_2,\tau_2\,v_2)&\twoheadrightarrow(S_2,\mathtt{true})\end{aligned}$$

Because of rule $\mathcal{X}_{2,4}$, it suffices to show that $\mathtt{b}=\mathtt{true}$. Note that $v_1\,R^*_T\,v_2$ implies that $v_1\equiv\lambda z.\,s[\overline{u_1/x}]$ for some $\emptyset;\overline{x:T}\vdash\lambda z.s:T$ and $\overline{u_1\,R\,u_2}$. Hence,

$$(S_1,U_{a_1a_2}\,v_1)\twoheadrightarrow(S_1,\ s[\overline{u_1/x}][a_1/z]=s[\overline{u_1/x}][a_2/z]\,)$$

and, using the notation $(a_1\ a_2)\cdot t$ for the term obtained from t by permuting all occurrences of a_1 and a_2 inside it,

$$(S_1,\ s[\overline{u_1/x}][a_1/z]\,)\Downarrow\mathtt{b}_1\implies(S_1,(a_1\,a_2)\cdot s[\overline{u_1/x}][a_1/z]\,)\Downarrow\mathtt{b}_1\,.$$

But the only way that a_1,a_2 may appear in $\overline{u_1}$ is by some u_{1j} being $U_{a_1a_2}$. Thus,

$$(a_1\,a_2)\cdot s[\overline{u_1/x}][a_1/z]\ \equiv\ s[\overline{u_1/x}][a_2/z][U_{a_2a_1}/U_{a_1a_2}]$$

and therefore, by Lemma 14, we get $(S_1,s[\overline{u_1/x}][a_2/z]\,)\Downarrow\mathtt{b}_1$ and thus $\mathtt{b}=\mathtt{true}$. □

Note that, in order to show that $\mathtt{b}=\mathtt{true}$, the original proof of [2] uses the bisimulation method again: it constructs an appropriate auxiliary set $\mathcal{Y}$ and proves it adequate by use of Theorem 13.

The proof of Benton and Koutavas is the only one based on a proper complete method. Stark's logical relations are not complete while the AGMOS approach provides a proof of (5) but not a general method. Moreover, the setting of the method is such that proofs can be automatically checked — and (5) in particular is checked in [2]. The downside of the method is that some guessing is needed in choosing $\mathcal{X}$ and, most importantly, that showing adequacy is by no means automatic, although the passage from Theorem 11 to Theorem 13 is a definite simplification.

5 AGMOS' proof

Abramsky, Ghica, Murawski, Ong and Stark [1] presented in 2004 a fully abstract model for the nu-calculus using *nominal game semantics* — game semantics with names. The paper is a remarkable achievement, providing the first, and only up to date, fully abstract model for the language. Using the model, the authors demonstrated a remarkably concise proof of (5) which, however, turns out to be flawed. As we show below, the flaw is easily fixable.

5.1 Nominal game semantics

Game semantics models computation as a *game*: a dialogue between the program and its environment which follows their exchange of data during program execution. Thus, a game has two players: a *Proponent* (or simply P) representing the program, and an *Opponent* (or O) representing the environment. A game is formally specified by *plays*, that is, sequences of moves played in alternation by the two players. Moves represent computational data and are chosen from appropriate *game arenas*. Arenas represent types and can be seen as game specifications: they provide the moves of the game, and its rules.

Definition 16. An ***arena*** A is a tuple $A = \langle M_A, I_A, \vdash_A \rangle$ where:

- M_A is a set of *moves*, partitioned into *questions* and *answers* by $M_A = M_A^Q \uplus M_A^A$, and to *O-moves* and *P-moves* by $M_A = M_A^O \uplus M_A^P$.
- $I_A \subseteq M_A^A \cap M_A^P$ is a set of *initial* moves.
- $\vdash_A \subseteq M_A \times (M_A \setminus I_A)$ is a *justification relation*, satisfying the condition: $\vdash_A \subseteq ((M_A^P \times M_A^O) \cup (M_A^O \times M_A^P)) \cap ((M_A^Q \times M_A) \cup (M_A^A \times M_A^Q))$.

The justification relation specifies the causality between the moves in an arena. For example, a question representing a function input justifies an answer representing the function's value. The justification condition stipulates that O-moves justify P-moves, and viceversa, and that answers can only justify questions. Note that initial moves are always P-answers and are not justified by anything.

For example, the arenas corresponding to the types `Bool` and `Name` are:

$$[\![\texttt{Bool}]\!] = \langle \mathbb{B}, \mathbb{B}, \emptyset \rangle\,, \quad [\![\texttt{Name}]\!] = \langle \mathbb{A}, \mathbb{A}, \emptyset \rangle$$

These are "flat" arenas in the sense that all moves are initial. Things lift up in higher-order types. First, for each set of moves M_A we define the *flipped* set $\overline{M_A}$ to be the set with the same elements, albeit with O- and P-polarities swapped, and with initial moves being changed to O-questions. Put formally:

$$\overline{M_A^O} = M_A^P,\ \overline{M_A^P} = M_A^O,\ \overline{M_A^Q} = M_A^Q \cup I_A,\ \overline{M_A^A} = M_A^A \setminus I_A$$

Now, for arenas A and B we define the arena $A \to B$ by setting:

$$\begin{aligned} M_{A\to B} &= \{*_F\} \uplus \overline{M_A} \uplus M_B \\ I_{A\to B} &= \{*_F\} \\ \vdash_{A\to B} &= \vdash_A \cup \vdash_B \cup \{(*_F, i_A), (i_A, i_B) \mid i_A \in I_A \wedge i_B \in I_B\} \end{aligned}$$

Intuitively, arrows are formed by justifying the initial moves of B from the initial moves of A, and the latter from the initial move $*_F$. The moves of B are otherwise left untouched, while those of A are flipped. For example:

$$[\![\mathtt{Bool} \to \mathtt{Bool}]\!] = [\![\mathtt{Bool}]\!] \to [\![\mathtt{Bool}]\!] = \langle \{*_F\} \uplus \overline{\mathbb{B}} \uplus \mathbb{B}, \{*_F\}, \{(*_F, \mathtt{b}_l), (\mathtt{b}_l, \mathtt{b}_r) \mid \mathtt{b}_l, \mathtt{b}_r \in \mathbb{B}\} \rangle$$

where the two copies of $\mathbb{B}$ are distinguished by using l- and r-tags. Observe that $\mathtt{b}_l$'s are O-questions while $\mathtt{b}_r$'s are P-answers.

Given arenas A and B we define the product arena $A \times B$ by setting:

$$\begin{aligned} M_{A\times B} &= (I_A \times I_B) \uplus (M_A \setminus I_A) \uplus (M_B \setminus I_B) \\ I_{A\times B} &= I_A \times I_B \\ \vdash_{A\times B} &= (\vdash_A \restriction (M_A \setminus I_A)) \cup (\vdash_B \restriction (M_B \setminus I_B)) \cup \{((i_A, i_B), m) \mid i_A \vdash_A m \vee i_B \vdash_B m\} \end{aligned}$$

Intuitively, products are formed by putting the component arenas side-by-side and gluing them together at their initial moves.

Remark 17. What distinguishes the games we define here from ordinary call-by-value games of [4] is the presence of names, which we have kept as inconspicuous as possible. In fact, all the game constructions we present are formally conducted within *strong nominal sets* [3, 9]. This means that there is a canonical notion of applying *name-permutations* to elements of our constructions, and any such element may only involve finitely many names — all other names are *fresh* for it. In particular, sets of moves are closed under name-permutations, and so do sets of initial moves, sets of plays, and strategies below. Moreover, all functions and relations defined are *nominal*: they commute with name-permutations.

Games are not played in arenas but *between* arenas, in structures defined below.

Definition 18. Given arenas A, B define the ***prearena*** $A \longrightarrow B$ to be a triple consisting of a set of moves, a set of initial moves and a justification relation:

$$A \longrightarrow B = \langle\ \overline{M_A} \uplus M_B, \overline{I_A}, \vdash_A \cup \vdash_B \cup \{(i_A, i_B) \mid i_A \in I_A \wedge i_B \in I_B\}\ \rangle$$

where moves are partitioned according to the partitions of $\overline{M_A}$ and M_B.

Thus, prearenas are almost identical to arenas only that their initial moves are O-questions, appearing on the left of the arrow. The difference between the arena $A \to B$ and the prearena $A \longrightarrow B$ is the extra initial move of the former.

Prearenas model types-in-context: each $S; \Gamma \vdash T$ with $|S| = n$ and $\Gamma = \{x_1 : T_1, \ldots, x_m : T_m\}$, is mapped to a prearena:

$$[\![\texttt{Name}]\!]^n \times [\![T_1]\!] \times \cdots \times [\![T_m]\!] \longrightarrow [\![T]\!]$$

with its initial move being an O-question opening the context on the LHS. This reflects the way games are played: the first move of a play is played by the environment and provides the context of the modelled program. For example, the prearena corresponding to the typing context $\emptyset; \emptyset \vdash \texttt{Bool} \to \texttt{Bool}$ is:

$$1 \longrightarrow [\![\texttt{Bool}]\!] \to [\![\texttt{Bool}]\!]$$

where $1 = \langle \{*\}, \{*\}, \emptyset \rangle$ is the one-move arena. Valid plays in this prearena are of the form:

$$\begin{array}{c} * \; *_F \; \mathtt{b}_l \; \mathtt{b}'_r \; \mathtt{b}''_l \; \mathtt{b}'''_r \cdots \\ O \;\; P \;\; O \;\; P \;\; O \;\; P \cdots \end{array}$$

These start with the initial O-move $*$ which provides the context ("the context is empty"), to which P answers by playing $*_F$ ("the result of the computation is a function"). From that point on, O may ask (possibly repeatedly) the value of the function for specific inputs $\mathtt{b}_l$, to which P answers with values $\mathtt{b}'_r$ according to the function which P represents.

Observe that we use pointers representing the causality relation between moves in the play. Pointers follow the justification relation and, for example, allow us to distinguish between different function calls. In the plays we consider all moves apart from the first one have pointers to preceding moves. For any sequence s of moves with full pointers which is *alternating* (moves alternate between O- and P-moves) we define its *view*, $\ulcorner s \urcorner$, inductively as follows.

$$\begin{aligned} \ulcorner m \urcorner &= m \\ \ulcorner s\, n\, m\, s'\, n' \urcorner &= \ulcorner s\, n \urcorner\, m\, n' \end{aligned}$$

For such a sequence s and a move m in it, say $s = s_1\, m\, s_2$, the *view of* m is $\ulcorner s_1 \urcorner$.

Definition 19. A ***legal sequence*** s in a prearena $A \longrightarrow B$ is an alternating sequence of moves from $A \longrightarrow B$ attached with explicit *justification pointers*, satisfying the conditions:

- The first move of s is an initial move (i.e. taken from $I_{A \to B}$).
- Apart from the first move, each move m in s is justified by some move n preceding it (by means of a justification pointer) such that $n \vdash_{A \longrightarrow B} m$.
- Each answer move in s is justified by its closest preceding question.
- Each move in s is justified by a move in its view.

The above are the standard conditions of *well-opening*, *justification*, *well-bracketing* and *visibility*. Plays are justified sequences attached with names. Formally, let us write $\mathbb{A}^{\circledast}$ for the set of finite lists of distinct names. A ***move-with-names*** is a pair $m^{\bar{a}}$, where m a move and $\bar{a}$ is a *name-list*, i.e. an element of $\mathbb{A}^{\circledast}$. We set $\underline{m^{\bar{a}}} = m$.

Definition 20. A ***play*** s in a prearena $A \longrightarrow B$ is a sequence of moves-with-names such that $\underline{s}$ is a legal sequence and the following conditions are satisfied.

- If $m^{\bar{a}}$ is a P-move in s then $\bar{a}$ contains as a prefix the name-list of the move preceding it. It possibly contains some other names, all of which are fresh for the whole sequence up to $m^{\bar{a}}$.
- If $m^{\bar{a}}$ is a P-move in s, and $a \in \mathbb{A}$ appears in m but is fresh for (i.e. it does not appear in) the view of $m^{\bar{a}}$, then a is contained in $\bar{a}$.
- The name-list of an O-move in s is that of the move justifying it, if the move is non-initial, otherwise it is empty.

The set of plays i n a $A \longrightarrow B$ is denoted by $P_{A,B}$.

Terms are modelled by sets of plays that represent specific *strategies*: instructions for P on how to play the game.

Definition 21. A strategy σ on a prearena $A \longrightarrow B$ is a prefix-closed set of plays in $P_{A,B}$ satisfying:

- If $s \in \sigma$ and s' is obtained from s by some name-permutation then $s' \in \sigma$.
- If even-length $s \in \sigma$ and $sm^{\bar{a}}$ is a play then $sm^{\bar{a}} \in \sigma$.
- If even-length $sm_1^{\bar{a}_1}, sm_2^{\bar{a}_2} \in \sigma$ then $m_2^{\bar{a}_2}$ is obtained from $m_1^{\bar{a}_1}$ by means of permuting names which are fresh for s.
- If even-length $s_1 m_1^{\bar{a}_1} \in \sigma$ and odd-length $s_2 \in \sigma$ have $\ulcorner s_1 \urcorner = \ulcorner s_2 \urcorner$ then there exists $s_2 m_2^{\bar{a}_2} \in \sigma$ where $m_2^{\bar{a}_2}$ is obtained from $m_1^{\bar{a}_1}$ by means of permuting names which are fresh for $\ulcorner s_1 \urcorner$.

We write $\sigma : A \longrightarrow B$.

The third condition stipulates that strategies are deterministic up to permutation of fresh names. The last condition, called *innocence*, expresses the fact that, excluding names, the nu-calculus exhibits purely functional behaviour: a program behaves in the same manner in different calls — the only thing that may change is the choice of generated names.

Example 22. Consider $\emptyset; \emptyset \vdash \lambda x, y.\, x == y : \mathtt{Name} \to \mathtt{Name} \to \mathtt{Bool}$. Its denotation is the strategy for the prearena

$$1 \longrightarrow [\![\ (\mathtt{Name}_x \to (\mathtt{Name}_y \to \mathtt{Bool}^z)^G)^F\ ]\!]$$

with typical play:

$$* \frown *_F \frown a_x \frown *_G \frown a_y \frown \mathtt{b}_z$$

$$O \quad P \quad O \quad P \quad O \quad P$$

where $\mathtt{b}_z$ is $\mathtt{true}$ iff $a_x = a_y$. Note that since only names introduced by P make it to the name-lists, here the name-lists are empty. Consider now the following.

$$[\![\nu a.\lambda x.\, a]\!] : 1 \longrightarrow [\![\mathtt{Bool}]\!] \to [\![\mathtt{Name}]\!] \quad \left\{ \begin{array}{l} * \; * \; \mathtt{b} \; a^a \; \mathtt{b}' \; a^a \; \cdots \\ O \;\; P \;\; O \;\; P \;\; O \;\; P \end{array} \right.$$

$$[\![\lambda x.\nu a.\, a]\!] : 1 \longrightarrow [\![\mathtt{Bool}]\!] \to [\![\mathtt{Name}]\!] \quad \left\{ \begin{array}{l} * \; * \; \mathtt{b} \; a^a \; \mathtt{b}' \; a'^{a'} \; \cdots \\ O \;\; P \;\; O \;\; P \;\; O \;\; P \end{array} \right.$$

The former term is a one-name generator: it generates a fresh name a and returns it whenever it is called. The latter is a proper name-generator: it returns a fresh name *each time* it is called.

Composition of strategies is defined by playing one against the other:

$$\text{if } \sigma : A \longrightarrow B \text{ and } \tau : B \longrightarrow C$$

then σ and τ have opposite O/P polarities in their B components, and therefore we can play them in parallel synchronising them at their B-moves (taking some extra care for name-lists). By subsequently hiding the moves from B from this parallel play (and redirecting pointers from initial moves of C to initial moves of A) we obtain a strategy:

$$\sigma; \tau : A \longrightarrow C$$

Theorem 23 ([1]). *Nominal games form a category, with objects being arenas and arrows being strategies on prearenas.*

5.2 Full abstraction

Nominal games provide a sound model for the nu-calculus which, moreover, satisfies finitary definability: any strategy with finite representation is the denotation of some nu-calculus term. Full abstraction is then obtained via the following notion of *intrinsic equivalence*.

Definition 24. Suppose $\sigma_1, \sigma_2 : 1 \longrightarrow A$. We define $\sigma_1 \approx \sigma_2$ to hold if:

$$\forall \rho : A \longrightarrow [\![\mathtt{Bool}]\!].\ \sigma_1; \rho = \{ * \; \mathtt{true} \} \iff \sigma_2; \rho = \{ * \; \mathtt{true} \}$$

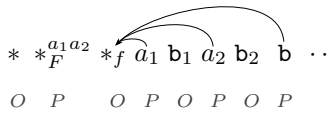
Theorem 25 (Full abstraction [1]). *For all typed terms $\emptyset; \emptyset \vdash t_1, t_2 : A$:*

$$t_1 \cong t_2 \iff [\![t_1]\!] \approx [\![t_2]\!]$$

We proceed to prove (5). The semantics of the simple term is given as follows (we write $\mathtt{t}$ as short for $\mathtt{true}$).

$$[\![\lambda f.\, \mathtt{true}]\!] : 1 \longrightarrow [\![(\mathtt{Name} \to \mathtt{Bool}) \to \mathtt{Bool}]\!] = \left\{ \begin{array}{l} * \; *_F \; *_f \; \mathtt{t} \; \cdots \\ O \;\; P \;\; O \;\; P \end{array} \right.$$

The semantics of the other term, $[\![\nu a_1, a_2.\lambda f.\, fa_1 = fa_2]\!]$, is given below (where we have omitted some name-lists and some pointers pointing to the preceding move),



with $\mathtt{b} = (\mathtt{b}_1 = \mathtt{b}_2)$. By full abstraction, it suffices to consider plays which can be played by a counter-strategy ρ in the prearena

$$[\![(\mathtt{Name} \to \mathtt{Bool}) \to \mathtt{Bool}]\!] \longrightarrow [\![\mathtt{Bool}]\!]\,.$$

Since the roles of O and P are reversed in ρ, the moves $\mathtt{b}_1$ and $\mathtt{b}_2$ are P-moves for it. Moreover, ρ contains the plays:

$$*_F\ *_f\ a_1\,\mathtt{b}_1\,, \quad *_F\ *_f\ a_1\,\mathtt{b}_1\,a_2\,\mathtt{b}_2$$

(note that ρ cannot see the name-lists a_1a_2, while the initial move $*$ is not in ρ's part of the board). Since ρ is closed under permutation of names, it contains the plays:

$$*_F\ *_f\ a_1\,\mathtt{b}_1\,, \quad *_F\ *_f\ a_2\,\mathtt{b}_1\,a_1\,\mathtt{b}_2\,.$$

Moreover, the view at $\mathtt{b}_1, \mathtt{b}_2$ is the same:

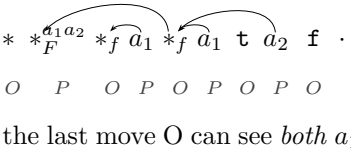
$$\ulcorner *_F\ *_f\ a_1 \urcorner = *_F\ *_f\ a_1 = \ulcorner *_F *_f\ a_2\,\mathtt{b}_1\,a_1 \urcorner$$

and therefore, by innocence, $\mathtt{b}_1 = \mathtt{b}_2$ and thus $\mathtt{b} = \mathtt{t}$. We then have that the two strategies cannot be distinguished by ρ because

$$\ulcorner *_F\ *_f\ \mathtt{t} \urcorner = *_F\ *_f\ \mathtt{t} = \ulcorner *_F\ *_f\ a_2\,\mathtt{b}_1\,a_1\,\mathtt{b}_2\,\mathtt{b} \urcorner$$

and ρ is innocent.

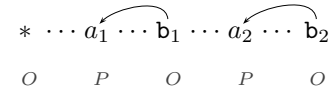
The above argument was presented in [1]. It is flawed in that in does not take into account the possibility of O not playing immediately $\mathtt{b}_1$ after a_1 is played, but rather opening a recursive call of f by playing $*_f$ again. Once the latter is considered, the argument fails. For example, the following is a valid play for the strategy.



Note that in the view of the last move O can see *both* a_1 *and* a_2 and, for example reply `f` (`false`): the two names are not the same. In fact, such an Opponent precisely corresponds to the counter-strategy given by the context (7).

There is no reason why O should stop in one recursive call of f, and therefore O is able to see more than two names in his view, call f recursively again after P closes one of its call, etc. Thus, a direct argument as the above does not go through. The following lemma solves the problem.

Lemma 26. *For any play s in $[\![\nu a_1, a_2.\lambda f.\, f a_1 = f a_2]\!]$ of the form*



in which O plays innocently if $\ulcorner * \cdots a_1 \urcorner$ can be obtained by $\ulcorner * \cdots a_2 \urcorner$ by permuting the names a_1 and a_2 then $\mathtt{b}_1 = \mathtt{b}_2$.

Proof. By induction on the length of s. Let $s = *\ *_F^{a_1 a_2}\ s_1\, a_1\, s_1'\, \mathtt{b}_1\, s_2\, a_2\, s_2'\, \mathtt{b}_2$. If $s_1' = \epsilon$ then, by O-innocence, $\mathtt{b}_2 = \mathtt{b}_1$. Otherwise, s_1' starts with $*_f$ and therefore, by O-innocence, so does s_2'. Thus,

$$s_1' = *_f\, a_1 \cdots \mathtt{b}_{11}\, a_2 \cdots \mathtt{b}_{12} \cdots$$
$$s_2' = *_f\, a_1 \cdots \mathtt{b}_{21}\, a_2 \cdots \mathtt{b}_{22} \cdots$$

with a_1, a_2 justified by $*_f$, $\mathtt{b}_{11}$ justified by a_1, $\mathtt{b}_{12}$ by a_2, and so on. We have:

$$s = *\ *_F^{a_1 a_2}\ s_1\, a_1\ *_f\ a_1^1 \cdots \mathtt{b}_{11}\, a_2^1 \cdots \mathtt{b}_{12} \cdots \mathtt{b}_1\, s_2\, a_2\ *_f\ a_1^2 \cdots \mathtt{b}_{21}\, a_2^2 \cdots \mathtt{b}_{22} \cdots \mathtt{b}_2$$

where we have tagged different occurrences of a_1, a_2. By hypothesis $\ulcorner * \cdots a_1 \urcorner = (a_1\ a_2) \cdot \ulcorner * \cdots a_2 \urcorner$ and therefore:

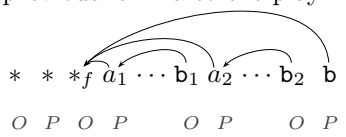
$$\ulcorner * \cdots a_1\ *_f\ a_1^1 \urcorner = (a_1\ a_2) \cdot \ulcorner * \cdots a_2\ *_f\ a_1^2 \cdots \mathtt{b}_{21}\, a_2^2 \urcorner$$
$$\ulcorner * \cdots a_1\ *_f\ a_1^1 \cdots \mathtt{b}_{11}\, a_2^1 \urcorner = (a_1\ a_2) \cdot \ulcorner * \cdots a_2\ *_f\ a_1^2 \urcorner$$

By applying the IH to the subsequence ending in $\mathtt{b}_{22}$ we obtain $\mathtt{b}_{11} = \mathtt{b}_{22}$, and by applying it to the one ending in $\mathtt{b}_{21}$ we get $\mathtt{b}_{12} = \mathtt{b}_{21}$. Thus,

$$s_1' = *_f\, a_1 \cdots \mathtt{b}_{11}\, a_2 \cdots \mathtt{b}_{12}\, \mathtt{b}\, s_1''$$
$$s_2' = *_f\, a_1 \cdots \mathtt{b}_{21}\, a_2 \cdots \mathtt{b}_{22}\, \mathtt{b}\, s_2''$$

with $\mathtt{b}$ justified by $*_f$. If $s_1'' = \epsilon$ then by O-innocence $s_2'' = \epsilon$ and $\mathtt{b}_1 = \mathtt{b}_2$. Otherwise, $s_1'' = *_f \cdots$ and $s_2'' = *_f \cdots$, and we repeat the same argument. □

Proof 27 Applying the previous lemma to the play



with its last move omitted we obtain that $\mathtt{b}_1 = \mathtt{b}_2$ and therefore $\mathtt{b} = \mathtt{true}$, and the argument proceeds as in [1]. □

The use of nominal games is advantageous in that it comes with a powerful full abstraction result. Moreover, it is constructive in the sense that strategies are derived from the syntax compositionally. On the other hand, though, the amount of notions (some of them very technical) one needs to digest in order to understand the model is quite substantial and the model is not particularly good for proving observational equivalences. It was possible to prove (5), but we have no general method for deriving such proofs.

6 Concluding remarks

We presented research in several directions instigated by the surprising complicacy involved in proving program equivalences of the nu-calculus. The methods have their advantages and disadvantages, and probably none of them is fully satisfactory: the language is simply difficult. In particular none of these, or any other, attempts have been able to answer the question:

Is program equivalence in the nu-calculus decidable?

We conjecture it is decidable.

Acknowledgements

Many thanks to Vasilis Koutavas, Paul-Andre Mellies, Andrzej Murawski and Ian Stark for fruitful discussions, explanations and suggestions.

References

1. S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. Nominal games and full abstraction for the nu-calculus. In *LICS*, pages 150–159, 2004.
2. N. Benton and V. Koutavas. A mechanized bisimulation for the nu-calculus, 2009. Symposium in Honor of Mitchell Wand, August 2009. Submitted to Higher Order and Symbolic Computation.
3. M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3-5):341–363, 2002.
4. K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation. *Theoretica Computer Science*, 221(1-2):393–456, 1999.
5. V. Koutavas. *Reasoning about Imperative and Higher-Order Programs*. PhD thesis, Northeastern University, 2008.
6. A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or what's new? In *MFCS*, pages 122–141, 1993.
7. I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, 1994.
8. E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54(5), 2007.
9. N. Tzevelekos. Full abstraction for nominal general references. *Logical Methods in Computer Science*, 5(3), 2009.