Program equivalence in a simple language with state

Nikos Tzevelekos

Queen Mary, University of London

Abstract

We examine different approaches to reasoning about program equivalence in a higher-order language which incorporates a basic notion of state: references of unit type (names). In particular, we present three such methods stemming from logical relations, bisimulation techniques and game semantics respectively. The methods are tested against a surprisingly difficult example equivalence at second order which exploits the intricacies of the language with respect to privacy and flow of names, and the ensuing notion of local state.

1. Introduction

Program equivalence constitutes one of the most expressive properties examined in program analysis.[1] Its expressiveness arises from the fact that it is a *contextual* notion: it regards program phrases under any possible context. As such, it subsumes other important properties like, for example, reachability or termination, and has straightforward applications in software verification and compiler optimisation. On the other hand, program equivalence is typically undecidable and even reasoning about specific such equivalences presents a demanding task. This is even more so when it comes to examining languages which combine local state with higher-order computation.

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 effects, e.g. references, objects and exceptions in languages like ML or Java. 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 [39]: a basic higher-order call-by-value functional language with references of unit type. The new language was accompanied by the following slogan.

> *[...] 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 lines capture the basic specification of computation with names. One can then go forward and build further "nominal" effects on top, for example by allowing names to be assigned values, dereferenced, raised as exceptions, etc.

[1]Two program phrases are called *observationally equivalent*, or simply *equivalent*, if they can be used interchangeably in any program context without affecting the observable behaviour of the latter.

Although the nu-calculus was designed with simplicity in mind, it was soon realised that it incorporated genuinely intricate effects which in turn resulted in quite delicate behaviours [45]:

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 following have been identified as characteristic such examples.

- The nu-calculus has a type Name for names. In empty (typing) context there is only one equivalence class in Name, represented by the term which produces a fresh name. This simple behaviour changes drastically when one moves to the type $\mathsf{Name} \to \mathsf{Name}$, where there are infinitely many observationally distinct terms. An infinite such class of terms is represented by *name-chains* of arbitrary size (Example 3).

- Standard proof methods for observational equivalence do not seem to migrate to this new setting, and in particular low-order equivalences like

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

  with $f : \mathsf{Name} \to \mathsf{Bool}$, have turned out to be remarkably difficult to prove.

Hence, this seemingly plain language attracted the attention of researchers and with time acquired the status of a riddle. The nu-calculus is a basic, strongly normalising language with finite base types and store of unit type (which amounts to no store). Nonetheless, apart from the first-order case [39], it is not known whether program equivalence is decidable, and even reasoning about equivalences like ($*$) presents great difficulties.

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 [45], based on logical relations for names; we demonstrate a complete proof method developed by Benton and Koutavas [8], based on environmental bisimulations; and finally we present a proof by Abramsky, Ghica, Murawski, Ong and Stark using game semantics [2], we show it to be flawed 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:

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

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

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

where $x \in \mathbb{V}$, $a \in \mathbb{A}$ and $\mathsf{b} \in \mathbb{B}$. Note that, in contrast to the standard formulation [39, 45] and in order to avoid introducing an extra notion of syntactic abstraction and α-equivalence for names, we have replaced the usual $\nu a.t$ construct by the constant new. It is easy to see that this change is harmless and the formulations are equivalent.[2] As usually, terms are considered modulo α-equivalence for λ-binding.

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 $\mathbb{A}$. The typing rules are given below.[3]

$$\frac{}{S;\Gamma \vdash \mathsf{b} : \mathsf{Bool}} \qquad \frac{(x:T)\in\Gamma}{S;\Gamma \vdash x : T} \qquad \frac{a \in S}{S;\Gamma \vdash a : \mathsf{Name}}$$

$$\frac{S;\Gamma \vdash s : T \to T' \quad S;\Gamma \vdash t : T}{S;\Gamma \vdash st : T'} \qquad \frac{S;\Gamma, x : T \vdash t : T'}{S;\Gamma \vdash \lambda x.t : T \to T'} \qquad \frac{}{S;\Gamma \vdash \mathsf{new} : \mathsf{Name}}$$

$$\frac{S;\Gamma \vdash t_1, t_2 : \mathsf{Name}}{S;\Gamma \vdash t_1 == t_2 : \mathsf{Bool}} \qquad \frac{S;\Gamma \vdash t : \mathsf{Bool} \quad S;\Gamma \vdash t_1, t_2 : T}{S;\Gamma \vdash \mathsf{if}\ t\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 : T}$$

To keep compatible with tradition, instead of new we shall be using in practice the following ν-constructor and its corresponding derived typing rule.

$$\nu x.t \;\equiv\; (\lambda x.t)\,\mathsf{new} \qquad\qquad \frac{S;\Gamma, x : \mathsf{Name} \vdash t : T}{S;\Gamma \vdash \nu x.t : T}$$

We shall also use the notation: $\mathsf{let}\ x = t\ \mathsf{in}\ t' \;\equiv\; (\lambda x.t')t$.

A pair (S, t) is called *valid* if S contains all names appearing in t. The operational semantics is given by means of a small-step transition relation with elements of the form:

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

with $(S, t), (S', t')$ valid pairs and $S \subseteq S'$. The semantics is call-by-value, with values given by:

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

We write $\mathit{Val}_T(S)$ for the set of values v which can be typed as $S;\emptyset \vdash v : T$. The reduction rules are as follows,

$$\begin{array}{ll}
(S, (\lambda x.t)v) \to (S, t[v/x]) & (S, a == a) \to (S, \mathsf{true}) \\
(S, \mathsf{new}) \to (S \uplus \{a\}, a) & (S, a == b) \to (S, \mathsf{false}) \\
(S, \mathsf{if\ true\ then}\ t_1\ \mathsf{else}\ t_2) \to (S, t_1) & \dfrac{(S,t) \to (S',t')}{(S, E[t]) \to (S', E[t'])} \\
(S, \mathsf{if\ false\ then}\ t_1\ \mathsf{else}\ t_2) \to (S, t_2) &
\end{array}$$

[2]Technically speaking, there is a difference between the two formulations. In particular, the original one seems to allow for name-binding in contexts, e.g. $C[a] \equiv \nu a.a$ is in theory a valid instantiation of $C[–] \equiv \nu a.[–]$ in [39, 45], whereas this in not possible here. But such instantiations are not intended by the definitions in [39, 45], and are not taken into account in practice (e.g. see proof of Context Lemma [45, Theorem 2.8]).
Note also that new is customised notation for the ML-like constant $\mathsf{ref}()$.

[3]Note that, with the advent of nominal sets [12], the use of S in the typing rules has become redundant as the latter can be deduced from the *support* of the term in question.

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

$$E[-] \ ::= \ [-]\,t \ \mid \ (\lambda x.t)[-] \ \mid \ [-] == t \ \mid \ a == [-] \ \mid \mathsf{if}\ [-]\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$$

We write $\twoheadrightarrow$ for the reflexive transitive closure of $\rightarrow$.

The use of disjoint union ($S \uplus \{a\}$) in the rule for fresh-name creation stipulates that $a \notin S$.[4] Moreover, the rule is non-deterministic, as any such a can be created. The derived rule for the ν constructor is:

$$(S, \nu x.t) \rightarrow (S \uplus \{a\}, t[a/x])$$

which, again, can create any name $a \notin S$.

Let us borrow from [12] the notation $(a\ b) \cdot t$ for the term obtained from t by permuting all occurrences of names a and b inside it, and similarly for the set $(a\ b) \cdot S$. We write $(a\ b) \cdot (S, t)$ for $((a\ b) \cdot S, (a\ b) \cdot t)$. We proceed with a first set of properties [45, 12]. For any valid pair (S, t), the following hold.

Weakening If $(S, t) \twoheadrightarrow (S', t')$ then $\forall S''.\ (S \uplus S'', t) \twoheadrightarrow (S' \uplus S'', t')$.

Equivariance If $(S, t) \twoheadrightarrow (S', t')$ then $\forall a, b.\ (a\ b) \cdot (S, t) \twoheadrightarrow (a\ b) \cdot (S', t')$.

Nominal Determinacy If $(S, t) \twoheadrightarrow (S_1, t_1)$ and $(S, t) \twoheadrightarrow (S_2, t_2)$ then there are $a_1, b_1, ..., a_n, b_n \notin S$ such that $(S_2, t_2) = (a_1\ b_1) \cdot \ldots \cdot (a_n\ b_n) \cdot (S_1, t_1)$, i.e. (S_2, t_2) can be obtained from (S_1, t_1) by permuting names not in S.

Strong Normalisation There is no infinite reduction sequence from (S, t).

Program equivalence is defined by means of *observational (or contextual) equivalence*. For any valid pair (S, t) and value v, we write $(S, t) \Downarrow v$ if there exists S' such that $(S, t) \twoheadrightarrow (S', v)$.

Definition 1. We say that a 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

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

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

Stark proves a Context Lemma [45, Theorem 2.8] which narrows down the kinds of context that need to be considered for establishing equivalences. In particular, $S; \Gamma \vdash t_1 \cong t_2 : T$ iff for all name-sets S', test functions $\lambda x.t \in \mathit{Val}_{T \to \mathsf{Bool}}(S \uplus S')$ and values $v_i \in \mathit{Val}_{T_i}(S \uplus S')$, $1 \le i \le n$,

$$(S \uplus S', (\lambda x.t)(t_1[\vec{v}/\vec{y}])) \Downarrow \mathsf{true} \iff (S \uplus S', (\lambda x.t)(t_2[\vec{v}/\vec{y}])) \Downarrow \mathsf{true}$$

where $\Gamma = \{y_1 : T_1, \ldots, y_n : T_n\}$. This is proved using operational reasoning from first principles.

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

$$\nu x.t \ \cong \ t : T \ \ \text{(if } x \text{ is not free in } t\text{)} \tag{1}$$

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

$$\nu x.\lambda y.\, x \ \ncong \ \lambda y.\nu x.\, x : \mathsf{Bool} \rightarrow \mathsf{Name} \tag{3}$$

[4]This is a convention we follow more generally in the paper.

Put in prose, 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

$$(\lambda f.\,(f\mathsf{true}) == (f\mathsf{true}))[-]$$

separates the terms in (3). (1) and (2) are proven in [45, Chapter 3] using a customised notion of applicative equivalence.

Example 3 (Chains and lassos). Using nested conditionals, one can derive a case constructor for the nu-calculus. Then, for each $n \in \omega$ we define a closed term $\mathsf{chain}_n : \mathsf{Name} \to \mathsf{Name}$ as follows [45].

$$\begin{aligned}\mathsf{chain}_n \;\equiv\; \nu x_1.\cdots\nu x_n.\lambda x.\,\mathsf{case}\,x\,\mathsf{of}\;x_1 &\;\Rightarrow\; x_2,\\ \vdots\; &\;\vdots\;\;\vdots\\ x_{n-1} &\;\Rightarrow\; x_n,\\ x_n &\;\Rightarrow\; x_1,\\ \mathsf{other} &\;\Rightarrow\; x_1.\end{aligned}$$

Hence, chain_n evaluates to a function which, the first time that it is called, will return a fresh name a_1. When called again, this time with input a_1, it will return another fresh name a_2, and so on until a_n. On input a_n, the function returns a_1 again, thus closing the “chain” of produced names.

We can easily check that, for each $n \neq m$, $\mathsf{chain}_n \ncong \mathsf{chain}_m$: a context repeatedly calling its argument function $\max(n,m)$-times, each time feeding to the function the previous output of the latter, will notice that only the smaller chain closes. Thus, there are infinitely many equivalence classes of terms in $\mathsf{Name} \to \mathsf{Name}$. Note that the terms we defined above do not provide a complete representation of $\mathsf{Name} \to \mathsf{Name}$. One can, for example, also consider open chains (by replacing the penultimate case with $x_n \;\Rightarrow\; \mathsf{new}$) or lassos (replacing with $x_n \;\Rightarrow\; x_i$ for $i > 1$).

Example 4 (Sample equivalences II). More interesting equivalences are the following. Here “$=$” is boolean equality, expressed by use of conditionals.

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

$$\nu x_1.\nu x_2.\,\lambda f.\,(f x_1 = f x_2) \;\cong\; \lambda f.\,\mathsf{true} : (\mathsf{Name} \to \mathsf{Bool}) \to \mathsf{Bool} \tag{5}$$

These are also proven in [45] but 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 intuitive 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 so within a nested call. The equivalence holds nonetheless, because overall there is perfect symmetry between the LHS and RHS of the comparison $fa_1 = fa_2$. This will be made precise in the following sections. For now, the reader may want to note the difference between (5) and:

$$\nu x_1.\,\lambda f.\,\nu x_2.\,(f x_1 = f x_2) \;\ncong\; \lambda f.\,\mathsf{true} : (\mathsf{Name} \to \mathsf{Bool}) \to \mathsf{Bool} \tag{6}$$

These are separated e.g. by the context:

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

where note that we use a nested call to G.

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).

Remark 5. In the presence of proper local state, i.e. in the extension of the calculus with ground store, equivalence (5) fails. A context can easily distinguish between the two terms: the first one calls its argument function f twice, while the second does not call it at all. As this comparison essentially misses the essence of (5), let us consider instead the following terms, in a language where names are integer references.[5]

$$\begin{aligned}&\mathsf{let}\ x_1, x_2 = \mathsf{ref}(0)\ \mathsf{in}\ \lambda f.(f x_1 = f(x_2 := !x_1;\ x_2))\\ &\mathsf{let}\ x = \mathsf{ref}(0)\ \mathsf{in}\ \lambda f.(f x = f x)\end{aligned}$$

The former term creates two names a_1, a_2 and feeds to f first a_1 and then a_2 (after updating the value of a_2 to that of a_1), and compares the two results. The latter creates a single name a and compares the results of calling f twice on input a. This equivalence query is also answered in the negative. For example, the terms can be distinguished by an integer-type context like (7) which in addition uses a flag to spot distinct names:

$$(\mathsf{let}\ z = \mathsf{ref}(0)\ \mathsf{in}\ \lambda G.\, G(\lambda x.\, G(\lambda y.\, \mathsf{if}\ x == y\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ z := 1; \mathsf{false})); !z)[-]$$

3. Stark's proof

The first proof of (5) was given in Stark's PhD thesis [45], 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 tackle (4) but fail to verify (5). Stark then proceeds by extending his setting to predicated logical relations, which successfully capture that specific equivalence although they do not lead to a better completeness result.

3.1. Logical Relations

Logical relations are based on the principle of relating functions just in case the latter map related inputs to related outputs. Because in the nu-calculus names are created dynamically, in relating terms one should also take into account that related terms may expand their name-sets with different names which, however, have equivalent roles in each of them. Hence, logical relations between terms with names are built around the notion of *span*: a bijective mapping between names inside name-sets. Formally, a span R between finite sets of names S_1, S_2, written

$$R : S_1 \rightleftharpoons S_2$$

[5]The language we imply here is Reduced ML [45, 33]. The constructor $t := s$ stands for value assignment, while $!t$ is for dereferencing. We write $s; t$ for $(\lambda x.t)s$, for some x not in t.

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 $(a_1,a_2) \in R$. We let $\mathsf{id} : S \rightleftharpoons S$ be the span which coincides with the identity function $\mathsf{id} : S \to S$.

Note that a span R can be equivalently seen as an inclusion-bijection-inclusion triple:

$$S_1 \;\supseteq\; \mathsf{dom}(R) \;\xrightarrow{\cong}\; \mathsf{cod}(R) \;\subseteq\; S_2$$

with $\mathsf{dom}(R) = \{a \mid \exists b.(a,b) \in R\}$ and $\mathsf{cod}(R) = \mathsf{dom}(R^{-1})$. Thus, R contains information about: (a) which names of S_1 are private to S_1 (the set $S_1 \setminus \mathsf{dom}(R)$); (b) which names of S_2 are private to S_2 (the set $S_2 \setminus \mathsf{cod}(R)$); (c) how are the public names of S_1 related to those of S_2 (the bijection $\mathsf{dom}(R) \cong \mathsf{cod}(R)$).

From the notion of span we move to logical relations as follows.

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

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

by induction on T as follows.

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

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

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

Stark shows that logical relations are complete at first-order types.[6]

Theorem 7 ([45])**.** *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 then the converse also holds.*

In particular, we can show that (8) below holds, and thus prove (4). However, we cannot show (9).

$$\nu x.\lambda y.\,(y == x) \;\mathsf{id}^*\; \lambda y.\,\mathsf{false} \tag{8}$$

$$\nu x_1.\nu x_2.\,\lambda f.\,(f x_1 = f x_2) \;\mathsf{id}^*\; \lambda f.\,\mathsf{true} \tag{9}$$

Note that in both cases the underlying spans are empty. In order to show (8), by definition, it suffices to show that $\lambda y.\,(y == a)\; R \;\lambda y.\,\mathsf{false}$, 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 $\lambda y.\,(y == a)$ would return false.

On the other hand, for (9) to hold it would be necessary to have $\lambda f.\,(f a_1 = f a_2)\; R\; \lambda f.\,\mathsf{true}$, with $R : \{a_1,a_2\} \rightleftharpoons \emptyset$ the empty span (i.e. $R = \emptyset$). But note now that, reasoning as above, we have $\lambda y.\,(y == a_1)\; R\; \lambda y.\,\mathsf{false}$, and if we use the latter functions as arguments to $\lambda f.\,(f a_1 = f a_2)$ and $\lambda f.\,\mathsf{true}$ respectively then we obtain different values.

A possibly unexpected limitation of logical relations is lack of transitivity.

[6]Type-order is given by: $\mathsf{ord}(\mathsf{Bool}) = \mathsf{ord}(\mathsf{Name}) = 0, \mathsf{ord}(T{\to}T') = \max\{\mathsf{ord}(T){+}1, \mathsf{ord}(T')\}$. Note that Stark's result is expressed for open terms, but equivalence of open terms is reducible to equivalence of closed ones.

Lemma 8. *For any $S;\emptyset \vdash t : T$, we have $t\,\mathsf{id}^*_T\,t$. Moreover, there exist $S;\emptyset \vdash t_1, t_2, t_3 : T$ such that $t_1\,\mathsf{id}^*_T\,t_2$ and $t_2\,\mathsf{id}^*_T\,t_3$, but not $t_1\,\mathsf{id}^*_T\,t_3$.*

Proof. Reflexivity is by [45, Proposition 4.7]. The proof of failure of transitivity we present below is again due to Stark [private communication]. Consider the term

$$t_1 \;\equiv\; \nu x_1.\nu x_2.\lambda f.\,(fx_1 == x_2) \wedge (fx_2 == x_1) \;:\; (\mathsf{Name} \to \mathsf{Name}) \to \mathsf{Bool}$$

and take also $t_2 \equiv \nu x.\lambda f.\mathsf{false}$ and $t_3 \equiv \lambda f.\mathsf{false}$, all of the same type, with logical conjunction defined e.g. by $t \wedge t' \equiv \mathsf{if}\ t\ \mathsf{then}\ t'\ \mathsf{else\ false}$. Moreover, let $t'_1 \equiv \lambda f.\,(fa_1 == a_2) \wedge (fa_2 == a_1)$ and $t'_2 \equiv t'_3 \equiv \lambda f.\mathsf{false}$.
To show $t_1\,\mathsf{id}^*\,t_2$ it suffices to show $t'_1\ R\ t'_2$ for some $R : \{a_1, a_2\} \rightleftharpoons \{a\}$. We choose $R = \{(a_1, a)\}$. Then, for all R-related $v_1, v_2 : \mathsf{Name} \to \mathsf{Name}$, it must be that $v_1 a_1 \not\Downarrow a_2$ (as a_2 is not R-related to any name). Therefore, if we use v_1, v_2 as arguments to t_1, t_2 respectively then we evaluate to false in both cases. This establishes $t_1\,\mathsf{id}^*\,t_2$. For $t_2\,\mathsf{id}^*\,t_3$ we use a much simpler argument.
On the other hand, to show $t_1\,\mathsf{id}^*\,t_3$ we need to show $t'_1\ R\ t'_3$ with $R : \{a_1, a_2\} \rightleftharpoons \emptyset$ the empty span. But note that we have $\mathsf{swp}\ R\ \lambda x.x$, where swp is the term:

$$\lambda x.\,\mathsf{if}\ x == a_1\ \mathsf{then}\ a_2\ \mathsf{else}\ (\mathsf{if}\ x == a_2\ \mathsf{then}\ a_1\ \mathsf{else}\ x) \;:\; \mathsf{Name} \to \mathsf{Name}$$

and feeding $\mathsf{swp}, \lambda x.x$ as arguments to t'_1, t'_3 respectively yields different values. Thus, t_1 and t_3 are not id^*-related. □

3.2. Predicated Logical Relations

The reason why (5) holds is that there is a symmetry between x_1 and x_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 examined terms, be 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 9. 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.

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

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

$$t_1\ \hat{R}^*_T\ t_2 \;\equiv\; \exists \hat{R}' : S'_1 \rightleftharpoons S'_2, v_i \in \mathit{Val}(S_i \uplus S'_i).\,(S_i, t_i) \twoheadrightarrow (S_i \uplus S'_i, v_i) \wedge v_1\ (\hat{R} \uplus \hat{R}')_T\ v_2$$

Theorem 10 ([45]). *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 then the converse also holds.*

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

PROOF (STARK). By the previous theorem, it suffices to show

$$\nu x_1.\nu x_2.\,\lambda f.\,(fx_1 = fx_2)\ \hat{\mathsf{id}}^*\ \lambda f.\,\mathsf{true}$$

where $\mathsf{id} : \emptyset \rightleftharpoons \emptyset$ the empty span, and for the latter it suffices to show that $\lambda f.\,(fa_1 = fa_2)\ \hat{R}\ \lambda f.\,\mathsf{true}$, 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 on the left. Note that $\lambda f.\mathsf{true}$ is R_2-related to itself by reflexivity, and it is also easy to see that $\lambda f.(fa_1 = fa_2)$ is R_1-related to itself. Thus, to show that $\lambda f.(fa_1 = fa_2)\ \hat{R}\ \lambda f.\mathsf{true}$, consider input functions $v_1, v_2 : \mathsf{Name} \to \mathsf{Bool}$ such that $v_1\,(\hat{R} \uplus \hat{R}')\,v_2$. By Definition 9 (case for λ-abstractions), v_1 must be $(R_1 \uplus R_1')$-related to itself and therefore v_1a_1 and v_1a_2 are deemed to give the same value. Thus,

$$v_1\,(\hat{R} \uplus \hat{R}')\,v_2 \implies (v_1a_1 = v_1a_2) \Downarrow \mathsf{true}$$

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

Note that, although the use of predicated logical relations allows us to show (5), we still do not obtain a complete method.

Lemma 12. *For any $S;\emptyset \vdash t : T$, we have $t\,\hat{\mathsf{id}}^*_T\,t$. Moreover, there exist $S;\emptyset \vdash t_1, t_2, t_3 : T$ such that $t_1\,\hat{\mathsf{id}}^*_T\,t_2$ and $t_2\,\hat{\mathsf{id}}^*_T\,t_3$, but not $t_1\,\hat{\mathsf{id}}^*_T\,t_3$.*

Proof. Reflexivity is in [45, Proposition 4.31]. To prove failure of transitivity we adapt the example terms from Lemma 8. Consider the term

$$t_1 \equiv \nu x_1.\nu x_2.\lambda f.\,(fx_1 == x_2) \vee (fx_2 == x_1) \,:\, (\mathsf{Name} \to \mathsf{Name}) \to \mathsf{Bool}$$

and take also $t_2 \equiv \nu x.\lambda f.\mathsf{false}$ and $t_3 \equiv \lambda f.\mathsf{false}$, all of the same type, with e.g. $t \vee t' \equiv \mathsf{if}\ t\ \mathsf{then\ true\ else}\ t'$, and let $t_1' \equiv \lambda f.\,(fa_1 == a_2) \vee (fa_2 == a_1)$ and $t_2' \equiv t_3' \equiv \lambda f.\mathsf{false}$.
For $t_1\,\hat{\mathsf{id}}^*\,t_2$ we show $t_1'\ \hat{R}\ t_2'$ for $\hat{R} : \{a_1, a_2\} \rightleftharpoons \{a\}$ given by $R_1 = \{(a_1, a_2), (a_2, a_1)\}$, $R = \{(a_1, a)\}$ and $R_2 = \{(a, a)\}$. Note first that $t_2'\ R_2\ t_2'$ by reflexivity and, moreover, $t_1'\ R_1\ t_1'$. The latter holds because, for any $(R_1 \uplus R_1')$-related inputs v_1, v_2 and $i \in \{1, 2\}$, $v_1a_i \Downarrow a_{3-i}$ iff $v_2a_{3-i} \Downarrow a_i$, and thus $t_1'v_1$ and $t_1'v_2$ must give the same value. Now take $v_1, v_2 : \mathsf{Name} \to \mathsf{Name}$ such that $v_1(\hat{R} \uplus \hat{R}')v_2$. We want to show that $t_1'v_1 \Downarrow \mathsf{false}$. Indeed, suppose that $v_1a_1 \Downarrow a_2$. Then, $v_1(\hat{R} \uplus \hat{R}')v_2$ would imply that $v_2a \Downarrow a'$ for some a' such that $a_2\ R\ a'$, which is not possible. On the other hand, if $v_1a_2 \Downarrow a_1$ then $v_1(R_1 \uplus R_1')v_1$ would imply that $v_1a_2 \Downarrow a_1$. This establishes $t_1\,\hat{\mathsf{id}}^*\,t_2$. It is easy to show $t_2\,\hat{\mathsf{id}}^*\,t_3$.
On the other hand, $t_1\,\hat{\mathsf{id}}^*\,t_3$ holds iff $t_1'\ \hat{R}\ t_3'$ for some $\hat{R} : \{a_1, a_2\} \rightleftharpoons \emptyset$. In this case, $R = R_2 = \emptyset$, while there are several choices for R_1. We show that for none of the choices do we get $t_1\,\hat{\mathsf{id}}^*\,t_3$.
If $R_1 = \emptyset, \mathsf{id}, \{(a_1, a_2), (a_2, a_1)\}$ then $\mathsf{swp}\ R_1\ \mathsf{swp}$ so the same argument as in Lemma 8 goes

through.
If $R_1 = \{(a_1, a_1)\}$ then consider the term:

$$\mathsf{2to1} \;\equiv\; \lambda x.\, \mathsf{if}\ x == a_2\ \mathsf{then}\ a_1\ \mathsf{else}\ x$$

We have that $\mathsf{2to1}\ R_1\ \mathsf{2to1}$ and $\mathsf{2to1}\ \hat{R}\ \lambda x.x$, but $t_1'(\mathsf{2to1})$ and $t_3'(\lambda x.x)$ yield different values. Dually for $R_1 = \{(a_2, a_2)\}$.
Finally, for $R_1 = \{(a_1, a_2)\}$ we have that $t_1'\ R_1\ t_1'$ fails since $\mathsf{2to1}\ R_1\ \lambda x.x$, while $t_1'(\mathsf{2to1})$ and $t_1'(\lambda x.x)$ give different values. Dually for $R_1 = \{(a_2, a_1)\}$. $\square$

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 the usefulness of predicated relations seems to be very specific to cases like (5).

4. Benton & Koutavas' proof

Another proof with operational flavour, but with full completeness power, was presented by Benton and Koutavas [8]. Their method invokes *environmental bisimulations* [47, 24], a notion of bisimulation defined not as a single relation but rather as a set of relations. This is dictated by the fact that the information available to the environment changes during the bisimulation test. Thus, the bisimulations considered are themselves annotated with sets of names and relate terms containing names from those sets. As the knowledge of names available to the environment may increase during computation, the bisimulation sets may grow and 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, relating closed values of T.

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

An annotated relation R is extended to closed terms in the following way.

$$\frac{\emptyset; \{x_1 : T_1, \ldots, x_m : T_m\} \vdash s : T \qquad v_{11}\, R_T\, v_{21} \quad \cdots \quad v_{1m}\, R_T\, v_{2m}}{s[\overline{v_1}/\overline{x}]\; R_T^*\; s[\overline{v_2}/\overline{x}]}$$

Observe that here closures are purely contextual, in contrast to the notion of closure in logical relations, 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 13. 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:

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

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.[7]

Definition 14. Given a set $\mathcal{X}$ of annotated relations we define a relation $(\mathcal{X})^\circ$ on typed closed terms by setting:

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

where z is a dummy variable of type Bool.

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 15 ([8])**.** *For all typed terms $S; \Gamma \vdash t_1, t_2 : T$:*

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

4.2. *A simpler proof method*

Theorem 15 yields a proof method for verifying equivalences. In order to show that $S; \emptyset \vdash t_1 \cong t_2 : T$, one follows the steps below.

1. Find a set $\mathcal{X}$ containing (S, S, R) such that $\lambda z.t_1 \, R_{\mathsf{Bool} \to T} \, \lambda z.t_2$ and $a \, R_{\mathsf{Name}} \, a$ 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: one needs to verify the condition of Definition 13 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 13 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.

[7] Again, in contrast to [8], we give the definition just for closed terms. The dummy abstraction λz is necessary, in accordance to [8], because R is defined only on values.

Definition 16. 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:

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

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 are the induction steps. Spelling out explicitly what the latter mean, and removing some clear cases, one can prove the following.

Theorem 17 ([8])**.** *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 $\mathsf{b}_1\, R_{\mathsf{Bool}}\, \mathsf{b}_2$, $\mathsf{b}_1 = \mathsf{b}_2$.*
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 $(S_1, (\lambda x.t_1)v_1) \twoheadrightarrow_k (S_1 \uplus S_1', v_1')$ and $v_1\, R_T^*\, v_2$, there exist S_2', v_2', Q such that:*

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

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

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

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

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

We proceed with the proof of (5). We take a shortcut with respect to the proof presented in [8] by using a lemma specific to evaluation in the nu-calculus. Let us set

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

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

Lemma 18. *For all $a_1, a_2 \in \mathbb{A}$, $U_{a_1 a_2} \cong U_{a_2 a_1}$.*

Proof. We can use the proof method described above but, in fact, this equivalence is rather simple and we can use e.g. Stark's logical relations and check that $U_{a_1 a_2}$ id $U_{a_2 a_1}$. ☐

PROOF (BENTON & KOUTAVAS). It suffices to relate $\lambda z.\tau_1$ with $\lambda z.\tau_2$, where

$$\tau_1 \;\equiv\; \nu x_1.\nu x_2.\,\lambda f.(f x_1 = f x_2)\,, \qquad \tau_2 \;\equiv\; \lambda f.\,\mathsf{true}\,.$$

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

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

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

We proceed to show that $\mathcal{X}$ is adequate. It will suffice to show that the conditions of Theorem 17 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 $\lambda z.\tau_1\,R_T\,\lambda z.\tau_2$, with $T=\mathsf{Bool}\to(\mathsf{Name}\to\mathsf{Bool})\to\mathsf{Bool}$, and let $\mathsf{b}_1\,R^*_{\mathsf{Bool}}\,\mathsf{b}_2$. By construction of $\mathcal{X}$, $\mathsf{b}_1=\mathsf{b}_2$. Moreover,

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

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 $T=\mathsf{Name}\to\mathsf{Bool}$ and $T'=\mathsf{Bool}$, and let $v_1\,R^*_T\,v_2$. Then, $a_1,a_2\in S_1$ and there are some S',b such that:

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

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

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

and therefore (using also Equivariance):

$$\begin{aligned}(S_1,\ s[\overline{u_1}/\overline{x}][a_1/y]\,)\Downarrow\mathsf{b}_1 &\implies (a_1\ a_2)\cdot(S_1,s[\overline{u_1}/\overline{x}][a_1/y]\,)\Downarrow\mathsf{b}_1\\ &\implies (S_1,s[(a_1\ a_2)\cdot\overline{u_1}/\overline{x}][a_2/y]\,)\Downarrow\mathsf{b}_1\end{aligned}$$

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, $(a_1\ a_2)\cdot\overline{u_1}$ is $\overline{u_1}$ with each occurrence of $U_{a_1a_2}$ replaced by $U_{a_2a_1}$. Therefore, by Lemma 18, we get $(S_1,s[\overline{u_1}/\overline{x}][a_2/y]\,)\Downarrow\mathsf{b}_1$ and thus $\mathsf{b}=\mathsf{true}$. □

Note that, in order to show that $\mathsf{b}=\mathsf{true}$, the original proof of [8] uses the bisimulation method again: it constructs an appropriate auxiliary set $\mathcal{Y}$ and proves it adequate by use of Theorem 17.

The proof of Benton and Koutavas is the only one based on a complete method. Stark's logical relations are not complete while the AGMOS approach (see next section) provides a proof of (5) but not a proper method. Moreover, the method is such that proofs can be automatically checked — that of (5) in particular was checked in [8]. The downside is that some guessing is needed for choosing $\mathcal{X}$ and, most importantly, that showing adequacy is by no means automatic, although the passage from Theorem 15 to Theorem 17 is a definite simplification.

5. AGMOS' proof

A proof based on a denotational approach was presented by Abramsky, Ghica, Murawski, Ong and Stark in [2]. The paper, combined with rectifications of [50], provides the first (and currently the only) fully abstract denotational model for the language.[8] Concretely, the model is constructed using *game semantics*, a generalisation of functional semantics which embodies intentional structure in the form of formal interactions (games) between two players: the program and its environment. Using the model, the authors demonstrated a concise proof of (5) which, however, turns out to be flawed. In this section, we give a description of the model as presented in [50], present the proof of (5) given in [2], and then provide a fix for it.

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, along with rules about which moves can be played at each stage of a play.

Definition 20. 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 moves of an arena. For example, a question which represents a function input will justify answers representing function return values. The justification condition stipulates that O-moves justify P-moves, and viceversa, and that answers may only justify questions. Initial moves are always P-answers and are not justified by anything.

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

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

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

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

[8]A denotational model is *fully abstract* if it matches observational equivalence with equality in the model.

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

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

where $+$ is a coproduct on sets, i.e. a disjoint union construction which imposes some indexing where necessary (such indices will remain invisible unless there is risk of confusion). Here $*$ is a constant representing the abstract value which corresponds to evaluated but yet unexplored functions. Since it is initial, it is a P-answer. Thus, arrows are formed by justifying the initial moves of B from the initial moves of A, and the latter from the overall initial move $*$. The moves of B are otherwise left untouched, while those of A are flipped. For example:

$$\begin{aligned} [\![\mathsf{Bool} \to \mathsf{Bool}]\!] &= [\![\mathsf{Bool}]\!] \to [\![\mathsf{Bool}]\!] \\ &= \langle\, \{*\} + \overline{\mathbb{B}} + \mathbb{B},\ \{*\},\ \{(*, \mathsf{b}_l), (\mathsf{b}_l, \mathsf{b}_r) \mid \mathsf{b}_l, \mathsf{b}_r \in \mathbb{B}\}\, \rangle \end{aligned}$$

where the two copies of $\mathbb{B}$ are distinguished by indices l and r. Observe that b_l's are O-questions while b_r's are P-answers. Drawing arenas as directed graphs, with moves as nodes and justification pairs as (inverse) edges, $[\![\mathsf{Bool} \to \mathsf{Bool}]\!]$ looks as follows.

$$\begin{tikzcd}
 & \mathsf{true}_l \arrow[dl] & \mathsf{true}_r \arrow[l] \arrow[ddl] \\
* & & \\
 & \mathsf{false}_l \arrow[ul] & \mathsf{false}_r \arrow[l] \arrow[uul] \\
P & O & P
\end{tikzcd}$$

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

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

where $\vdash_A\restriction M = \{(m,n) \in \vdash_A \mid m \in M\}$. Intuitively, products are formed by putting the component arenas side-by-side and gluing them together at their initial moves. For example, $[\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool}]\!] = \langle \mathbb{B} \times \mathbb{B},\, \mathbb{B} \times \mathbb{B},\, \emptyset \rangle$ whereas $[\![\mathsf{Bool}]\!] \times [\![\mathsf{Bool} \to \mathsf{Bool}]\!]$ is given by:

$$\langle\, (\mathbb{B} \times \{*\}) + \overline{\mathbb{B}} + \mathbb{B},\ \mathbb{B} \times \{*\},\ \{((\mathsf{b}, *), \mathsf{b}_l), (\mathsf{b}_l, \mathsf{b}_r) \mid \mathsf{b}, \mathsf{b}_l, \mathsf{b}_r \in \mathbb{B}\}\, \rangle$$

Remark 21 (Nominal games). What distinguishes the games we define here from ordinary call-by-value games of [19] 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* [12, 50]. This means that there is a canonical notion of applying *name-permutations* to elements of our constructions,[9] 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-permutation, and so do sets of initial moves, sets of plays, and strategies below. Moreover, all functions and relations defined are *nominal*: they are closed under name-permutation.[10]

[9]Formally, for any nominal set X, $x \in X$ and $\pi : \mathbb{A} \xrightarrow{\cong} \mathbb{A}$, we write $\pi \cdot x$ for the element of X obtained from x be application of π. We saw this formalism before, on page 4, when we denoted by $(a\ b) \cdot t$ the term obtained from t by swapping all occurrences of a and b inside it.

[10]I.e. if $R \subseteq X \times Y$ then, for all $x \in X$, $y \in Y$ and $\pi : \mathbb{A} \xrightarrow{\cong} \mathbb{A}$, $(x, y) \in R \implies (\pi \cdot x, \pi \cdot y) \in R$.

Games are not played in single arenas but rather *between two arenas*: one representing the context, and another one representing the term type.

Definition 22. Given arenas A, B define the ***prearena*** $A \rightharpoonup B$ to be a triple consisting of a set of moves, a set of initial moves and a justification relation:

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

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

Thus, prearenas are similar to arenas but their initial moves are O-questions. The difference between the arena $A \to B$ and the prearena $A \rightharpoonup B$ is the extra initial move of the former. For example:

$$[\![\mathsf{Bool}]\!] \rightharpoonup [\![\mathsf{Bool}]\!] = \langle\, \overline{\mathbb{B}} + \mathbb{B},\ \overline{\mathbb{B}},\ \{(\mathsf{b}_l, \mathsf{b}_r) \mid \mathsf{b}_l, \mathsf{b}_r \in \mathbb{B}\}\,\rangle$$

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

$$[\![S; \Gamma \vdash T]\!] \;=\; [\![\mathsf{Name}]\!]^{\#n} \times [\![T_1]\!] \times \cdots \times [\![T_m]\!] \;\rightharpoonup\; [\![T]\!]$$

where $[\![\mathsf{Name}]\!]^{\#n} = \langle \mathbb{A}^{\#n}, \mathbb{A}^{\#n}, \emptyset \rangle$ and $\mathbb{A}^{\#n} = \{(a_1, ..., a_n) \in \mathbb{A}^n \mid \forall i \neq j.\, a_i \neq a_j\}$.

The initial move of $[\![S; \Gamma \vdash T]\!]$ is 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 $\emptyset; \emptyset \vdash \mathsf{Bool} \to \mathsf{Bool}$ is:

$$1 \;\rightharpoonup\; [\![\mathsf{Bool}]\!] \to [\![\mathsf{Bool}]\!]$$

Here $1 = \langle \{\star\},\ \{\star\},\ \emptyset \rangle$, the *one-move arena*, is the unit of the product operation on arenas ($\times$). Valid plays in $[\![\emptyset; \emptyset \vdash \mathsf{Bool} \to \mathsf{Bool}]\!]$ are of the form:

$$\begin{tikzcd}
\star & * \arrow[l, bend right] & \mathsf{b}_l \arrow[l, bend right] & \mathsf{b}'_r \arrow[l, bend right] & \mathsf{b}''_l \arrow[lll, bend right] & \mathsf{b}'''_r \arrow[l, bend right] & \cdots \\
O & P & O & P & O & P & \cdots
\end{tikzcd}$$

These start with the initial O-move $\star$ which provides the context ("the context is empty"), to which P answers by playing $*$ ("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 b_l, to which P answers with values b'_r according to the function that P represents.

The format of plays is that of sequences of moves attached with justification pointers which specify the dependency between different moves in the play. Pointers follow the justification relation of the prearena and, in particular, every move in a play apart from the first one has a unique pointer to a preceding move. With moves having more than one occurrence inside a play, e.g. because of different calls of the same function, the role of pointers is essential. For example, move m may have two occurrences inside play s and move n one occurrence, with $m \vdash_A n$ in the prearena A. A pointer from n to one of the occurrences of m allows us to clarify which move does n depend on.

When examining a point in computation where different copies of the same function have been opened, the relevant copy is the one that is *active* in the current function environment. Usually, this is just the last pending function call. However, in a setting where different functions can be called in a mutual-dependent fashion, determining the active functions is more elaborate.

In game semantics, the active function environment corresponds to a notion of *partial view* of the play: at each point in a play, only some of the earlier moves are relevant to the next move to be played. Formally, for any sequence s of moves with justification pointers, we define its ***view***, $\ulcorner s \urcorner$, inductively as follows.

$$\ulcorner sm \urcorner = m \quad \text{if } m \text{ has no outgoing pointer}$$
$$\ulcorner s\, \overgroup{m\, s'\, n} \urcorner = \ulcorner s \urcorner\, \overgroup{m n}$$

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

We next stipulate some basic conditions on the legality of plays.

Definition 23. A ***legal sequence*** s in a prearena $A \rightharpoonup B$ is a sequence of moves from $A \rightharpoonup B$ attached with explicit *justification pointers*, satisfying the conditions:

- If $s = s_1 m n s_2$ then $m \in M^O_{A \rightharpoonup B}$ iff $n \in M^P_{A \rightharpoonup B}$ (*alternation*).
- If $s = m s_1 n s_2$ then $m \in I_{A \rightharpoonup B}$ and $n \notin I_{A \rightharpoonup B}$ (*well-opening*).
- If $s = s_1 n s_2$ with $|s_1| > 0$ then there is m *justifying* n in s, that is, $s = s_1' \overgroup{m s_1'' n} s_2$ and $m \vdash_{A \rightharpoonup B} n$ (*justification*).
- If $s = s_1 \overgroup{m s_2 n} s_3$ with $n \in M^A_{A \rightharpoonup B}$ then each $m' \in M^Q_{A \rightharpoonup B}$ in s_2 is *answered* in s_2, that is, there is $n' \in M^A_{A \rightharpoonup B}$ such that $s = s_1 \overgroup{m s_2' \overgroup{m' s_2'' n'} s_2''' n} s_4$ (*well-bracketing*).
- If $s = s_1 \overgroup{m s_2 n} s_3$ then m appears in $\ulcorner s_1 m s_2 \urcorner$ (*visibility*).

We therefore concentrate on sequences of moves with pointers, s, such that: s is alternating between O- and P-moves; the first move of s is an initial move, and no other move in s is initial; apart from the first move, each move n in s is justified by some move m preceding it (by means of a justification pointer) such that $m \vdash_{A \rightharpoonup B} n$; each answer move n in s *answers* (i.e. it is justified by) its closest preceding question m that is still unanswered; each move in s is justified by a move in its view.

Plays are legal sequences in which moves are attached with an additional component: a list of names. The purpose of the latter is to collect names produced by the program during its execution. More precisely, the names which appear in the name-list of a move are precisely the names *introduced* (i.e. played first, allocated) by P in the current view (i.e. in the view of the examined move). Formally, let us write $\mathbb{A}^{\circledast}$ for the set of all 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*,[11] i.e. an element of $\mathbb{A}^{\circledast}$. We set $\underline{m^{\bar{a}}} = m$ and say that a name a is ***fresh*** for a sequence s if it does not appear in any move or name-list in it. Plays are given as follows.

Definition 24. A ***play*** s in a prearena $A \rightharpoonup B$ is a sequence of moves-with-names such that $\underline{s}$ is a legal sequence and the following conditions are satisfied.

- If $s = s_1 n^{\bar{a}'} m^{\bar{a}} s_2$ with $m \in M^P_{A \rightharpoonup B}$ then $\bar{a}'$ is a prefix of $\bar{a}$ and, for all $a \in \mathbb{A}$, we have that $a \in \bar{a} \setminus \bar{a}'$ iff a appears in $m^{\bar{a}}$ but is fresh for $s_1 n^{\bar{a}'}$.

[11] Originally, moves-with-names contained finite *sets* of names [2] but, as shown in [50], the latter choice leads to flaws in strategy composition. The choice of lists (and, more generally, the choice of a setting with *strong support*) rectifies the problem. Note in particular that the order of appearance of names in name-lists corresponds to their order of introduction.

- If $s = s_1 \overset{\curvearrowleft}{n^{\bar{a}'} s_2\, m^{\bar{a}}} s_3$ with $m \in M^O_{A \rightharpoonup B}$ then $\bar{a} = \bar{a}'$; if $s = m^{\bar{a}} s_1$ then $\bar{a} = \epsilon$.
- If $s = s_1 m^{\bar{a}} s_2$ with $m \in M^P_{A \rightharpoonup B}$ and a appears in $m^{\bar{a}}$ but not in $\ulcorner s_1 \urcorner$ then a is fresh for s_1.

The set of plays in a $A \rightharpoonup B$ is denoted by $P_{A,B}$.

The above conditions stipulate that, inside a play: each P-move contains in its name-list (as a prefix) the name-list of the O-move preceding it, augmented with any extra names just introduced; the name-list of each O-move coincides with that of the P-move justifying it (or is empty if the O-move is initial); whenever a P-move introduces a name in its view, it must be introducing it in the whole play[12].

Terms are modelled by sets of plays that represent specific *strategies*: instructions for P on how to play the game. In particular, a strategy σ should satisfy three properties: (a) it should be closed under name-permutation; (b) it should be deterministic up to choice of fresh names; (c) it should behave in a functional manner (i.e. *innocently*) up to choice of fresh names: the choice of next move should not depend on the whole preceding play, but solely on the current view. The latter condition reflects the fact that, names excluded, the nu-calculus exhibits purely functional behaviour: a program behaves in the same manner in different calls of it — the only thing that may change is the choice of generated names. Since the active call environment is captured by the view of the play, pure functional behaviour is captured by innocence (cf. [21]): strategies make moves each time judging just from what is currently in the view.

Definition 25. A strategy σ on a prearena $A \rightharpoonup B$ is an even-prefix-closed set of even-length plays from $P_{A,B}$ satisfying:[13]

- If $s \in \sigma$ and s' is obtained from s by some name-permutation then $s' \in \sigma$.
- If $sm^{\bar{a}}, sm'^{\bar{a}'} \in \sigma$ then $sm'^{\bar{a}'}$ is obtained from $sm^{\bar{a}}$ by means of permuting names which are fresh for s.
- If $s_1 m_1^{\bar{a}_1}, s_2 \in \sigma$ and $s_2 m_2^{\bar{a}_2} \in P_{A,B}$ are such that $\ulcorner s_1 \urcorner = \ulcorner s_2 m_2^{\bar{a}_2} \urcorner$ then there exists $s_2 m_2^{\bar{a}_2} m_1'^{\bar{a}_1'} \in \sigma$ such that $\ulcorner s_2 m_2^{\bar{a}_2} m_1'^{\bar{a}_1'} \urcorner$ is obtained from $\ulcorner s_1 m_1^{\bar{a}_1} \urcorner$ by means of permuting names which are fresh for s_1.

We write $\sigma : A \longrightarrow B$ to denote that σ is a strategy on $A \rightharpoonup B$.

Example 26. Consider $\emptyset; \emptyset \vdash \lambda x. \lambda y.\, x == y : \mathsf{Name} \to \mathsf{Name} \to \mathsf{Bool}$. Its denotation is a strategy $[\![\lambda x. \lambda y.\, x == y]\!]$ for the prearena:[14]

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

[12]i.e. P cannot use any name name that appears in the play but not in the current view.

[13]Using notation from nominal sets and exploiting the fact that, for all s, π, we have $\pi \cdot s = s$ iff π fixes all names in s, the conditions can be rewritten as:

- if $s \in \sigma$ then $(\pi \cdot s) \in \sigma$ for all permutations π;
- if $sm^{\bar{a}}, sm'^{\bar{a}'} \in \sigma$ then $sm'^{\bar{a}'} = \pi \cdot (sm^{\bar{a}})$ for some permutation π;
- if $s_1 m_1^{\bar{a}_1}, s_2 \in \sigma$, $s_2 m_2^{\bar{a}_2} \in P_{A,B}$ and $\ulcorner s_1 \urcorner = \ulcorner s_2 m_2^{\bar{a}_2} \urcorner$ then there exists $s_2 m_2^{\bar{a}_2} m_1'^{\bar{a}_1'} \in \sigma$ such that $\ulcorner s_2 m_2^{\bar{a}_2} m_1'^{\bar{a}_1'} \urcorner = \pi \cdot \ulcorner s_1 m_1^{\bar{a}_1} \urcorner$ for some permutation π.

[14]notice we use variables to index subtypes (and corresponding subarenas) of the main type.

In particular, the typical plays of $[\![\lambda x.\lambda y.\, x == y]\!]$ are of the form:

$$\begin{tikzcd}[row sep=small]
\star & *_F \arrow[l, bend right] & a_x \arrow[l, bend right] & *_G \arrow[l, bend right] & a'_y \arrow[l, bend right] & \mathsf{b}_z \arrow[l, bend right] \\
O & P & O & P & O & P
\end{tikzcd}$$

where $\mathsf{b}_z = \mathsf{true}$ 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 characteristic plays for the following terms.

$$[\![\nu x.\lambda y.\, x]\!] : 1 \longrightarrow [\![(\mathsf{Bool}_y \to \mathsf{Name}^x)^F]\!] \;=\; \left\{ \begin{tikzcd}[row sep=small]
\star & *_F^a \arrow[l, bend right] & \mathsf{b}_y^a \arrow[l, bend right] & a_x^a \arrow[l, bend right] & \mathsf{b}'_y \arrow[lll, bend right] & a_x^a \arrow[l, bend right] & \cdots \\
O & P & O & P & O & P &
\end{tikzcd} \right\}$$

$$[\![\lambda y.\nu x.\, x]\!] : 1 \longrightarrow [\![(\mathsf{Bool}_y \to \mathsf{Name}^x)^F]\!] \;=\; \left\{ \begin{tikzcd}[row sep=small]
\star & *_F \arrow[l, bend right] & \mathsf{b}_y \arrow[l, bend right] & a_x^a \arrow[l, bend right] & \mathsf{b}'_y \arrow[lll, bend right] & {a'}_x^{a'} \arrow[l, bend right] & \cdots \\
O & P & O & P & O & P &
\end{tikzcd} \right\}$$

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 (i.e. $a \neq a'$ etc.).

In the previous examples we specified strategies by providing "characteristic" plays. We can make this representation fully formal. Because of the innocence condition, strategies are determined by their behaviour on views. Hence, in defining a strategy it suffices to specify just the views of its plays. In cases like the above where such views have bounded length, it suffices to provide the views of maximal length.

Put formally, for each strategy $\sigma : A \longrightarrow B$ we define:

$$\ulcorner \sigma \urcorner = \{ \ulcorner s \urcorner \mid s \in \sigma \}$$

We call σ *finitary* if $\ulcorner \sigma \urcorner$ is finite modulo permutations of names.[15] If σ is finitary, we collect the elements of maximal length from $\ulcorner \sigma \urcorner$ in the set:

$$\hat{\sigma} = \{\, s \in \ulcorner \sigma \urcorner \mid \forall t \in \ulcorner \sigma \urcorner .\, s \sqsubseteq t \implies s = t \,\}$$

It is not difficult to see that, for all finitary σ_1, σ_2, $\;\sigma_1 = \sigma_2 \iff \hat{\sigma}_1 = \hat{\sigma}_2\;$ and therefore that finitary strategies are uniquely determined by this representation.

Games model programs denotationally and, hence, a focal aspect of the formalism is *composition*. Composition of strategies is defined by playing one against the other: 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 and synchronise them at their B-moves (taking some extra care for name-lists). By subsequently hiding the moves from B in this extended play, and redirecting pointers from initial moves of C to initial moves of A, we obtain a strategy:[16]

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

[15]That is, if the set $\{\{\pi \cdot \ulcorner s \urcorner \mid \pi : \mathbb{A} \xrightarrow{\cong} \mathbb{A}\} \mid s \in \sigma\}$ is finite.

[16]This is an informal, simplified description of composition. See [50] for the formal definition.

By virtue of strategy composition we obtain a *category* of nominal games.[17] Using the structure present in the category (formally, that of a λ_c-model [32]), we can effectively construct a sound model for the ν-calculus in nominal games. That is, for each typed term $S;\Gamma \vdash t : T$ with $\Gamma = \{x : T_1, \cdots, x : T_m\}$ and $|S| = n$, one can define a strategy:

$$[\![t]\!] \ : \ [\![\mathsf{Name}]\!]^{\#n} \times [\![T_1]\!] \times \cdots \times [\![T_m]\!] \longrightarrow [\![T]\!]$$

The above translation is effective and in particular compositional in the structure of t. Moreover, it is such that, for all t_1, t_2, if $[\![t_1]\!] = [\![t_2]\!]$ then $t_1 \cong t_2$.

Theorem 27 ([2, 50])**.** *Nominal games form a category, with objects being arenas and arrows being strategies on prearenas. Moreover, they yield a sound model for the nu-calculus.*

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 translation of some nu-calculus term. Full abstraction is then obtained via quotienting, using the following notion of *intrinsic equivalence*.

Definition 28. Suppose $\sigma_1, \sigma_2 : [\![\mathsf{Name}]\!]^{\#n} \longrightarrow A$. We define $\sigma_1 \approx \sigma_2$ to hold if:

$$\forall \rho : [\![\mathsf{Name}]\!]^{\#n} \times A \longrightarrow [\![\mathsf{Bool}]\!].\ \sigma_1 \mathbin{⨾} \rho = \{\, * \,\mathsf{true}\,\} \iff \sigma_2 \mathbin{⨾} \rho = \{\, * \,\mathsf{true}\,\}$$

where ⨾ is the canonical[18] notion of composition which projects along the common component $[\![\mathsf{Name}]\!]^{\#n}$:

$$\sigma_i \mathbin{⨾} \rho = \ [\![\mathsf{Name}]\!]^{\#n} \xrightarrow{\langle \mathsf{id}, \sigma_i \rangle} [\![\mathsf{Name}]\!]^{\#n} \times A \xrightarrow{\rho} [\![\mathsf{Bool}]\!]$$

with $\langle \mathsf{id}, \sigma_i \rangle = \{ \bar{a}\,(\bar{a}, m)^{\bar{b}} s \mid \bar{a}\, m^{\bar{b}} s \in \sigma_i \}$.

Theorem 29 (Full abstraction [2])**.** *For all typed terms $S; \emptyset \vdash t_1, t_2 : A$:*

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

Since quotienting is necessary for proving full abstraction, it is also important in proofs of equivalences. The need for quotienting stems from two inaccuracies of the (unquotiented) game model:

(a) Opponent is allowed to play names which have been introduced in the play by P but appear solely in name-lists and are therefore not really available to O.[19] More generally, even if O doesn't play such private names explicitly, he may follow a strategy that depends on them.

(b) Opponent is allowed to have non-innocent behaviours, e.g. by giving different answers when asked the same P-question consecutively. More generally, strategies are strictly more intentional than the terms they model. For example, the strategy $[\![\lambda f.\,\mathsf{if}\ fa\ \mathsf{then}\ fa\ \mathsf{else}\ fa]\!]$ will invoke f twice (i.e. play the P-question a under $*_f$ twice), in contrast to the equivalent strategy $[\![\lambda f.fa]\!]$ (which does this once). In fact, the latter is a problem pertinent more generally to the innocent modelling of functional behaviour. It is inherited to this model from the prototypical game

[17]For basic notions of category theory (not necessary for following this paper) see e.g. [6].

[18]technically speaking, this is composition in the co-Kleisli category of the comonad $[\![\mathsf{Name}]\!]^{\#n} \times \_$.

[19]These correspond to names allocated privately by the program but not revealed to its environment.

model for the purely functional language PCF [3, 21, 19]. In the case of PCF, though, semantic quotienting has been shown to be unavoidable: from the undecidability of program equivalence in (finitary) PCF, proved by Loader [31], it follows that the fully abstract game model cannot be effectively made quotient-free. Decidability of program equivalence in the nu-calculus, on the other hand, remains an open problem.

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

$$[\![\widehat{\lambda f.\,\mathsf{true}}]\!] : 1 \longrightarrow [\![((\mathsf{Name}^x \to \mathsf{Bool}_y)_f \to \mathsf{Bool}^z)^F]\!] = \left\{ \begin{tikzcd}[row sep=tiny, column sep=small] \star & *_F \arrow[l, bend right] & *_f \arrow[l, bend right] & \mathsf{t}_z \arrow[l, bend right] \\ O & P & O & P \end{tikzcd} \right\}$$

We set $s \equiv \star *_F *_f \mathsf{t}_z$. The semantics of the other term is given below, where for economy we omit name-lists of moves after $*_F^{a_1 a_2}$ (all these moves have name-list $a_1 a_2$),

$$[\![\nu x_1.\nu x_2.\widehat{\lambda f.\,f x_1 = f x_2}]\!] \;=\; \left\{ \begin{tikzcd}[row sep=tiny, column sep=small] \star & *_F^{a_1 a_2} \arrow[l, bend right] & *_f \arrow[l, bend right] & a_{1\,x} \arrow[l, bend right] & \mathsf{b}_{1\,y} \arrow[l, bend right] & a_{2\,x} \arrow[lll, bend right] & \mathsf{b}_{2\,y} \arrow[l, bend right] & \mathsf{b}_z \arrow[lllll, bend right] \\ O & P & O & P & O & P & O & P \end{tikzcd} \right\}$$

with $\mathsf{b} = (\mathsf{b}_1 = \mathsf{b}_2)$. Let us set $s' \equiv \star *_F *_f a_1 \,\mathsf{b}_1\, a_2\, \mathsf{b}_2\, \mathsf{b}_z$ (we drop the indices x, y for more economy). By full abstraction, in order to show the terms are equivalent, it suffices to focus on plays which can be played by a counter-strategy ρ in the prearena

$$[\![(\mathsf{Name} \to \mathsf{Bool}) \to \mathsf{Bool}]\!] \rightharpoonup [\![\mathsf{Bool}]\!]\,.$$

Since the roles of O and P are reversed in ρ, the moves b_1 and b_2 are P-moves for it. Moreover, ρ contains the plays:[20]

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

As ρ is closed under permutation of names, it also contains the plays:

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

In the latter, the view at b_1 and b_2 is the same:

$$\ulcorner *_F\ *_f\ a_1 \urcorner = *_F\ *_f\ a_1 = \ulcorner \begin{tikzcd}[column sep=tiny] *_F & *_f & a_2 & \mathsf{b}_1 & a_1 \arrow[lll, bend right] \end{tikzcd} \urcorner$$

and therefore, by innocence, $\mathsf{b}_1 = \mathsf{b}_2$ and thus $\mathsf{b} = \mathsf{t}$. Therefore, the two strategies cannot be distinguished by ρ because the latter is innocent, the views of s and s' are the same,

$$\ulcorner *_F\ *_f\ \mathsf{t} \urcorner = *_F\ *_f\ \mathsf{t} = \ulcorner \begin{tikzcd}[column sep=tiny] *_F & *_f & a_2 & \mathsf{b}_1 & a_1 & \mathsf{b}_2 & \mathsf{b} \arrow[lllll, bend right] \end{tikzcd} \urcorner$$

and the moves hidden in the view of s' cannot be revisited.

The above argument was presented in [2]. It is flawed in that in does not take into account the possibility that, after a_1 is played, O does not play b_1 but rather opens a nested call of f by playing $*_f$ again. Once this scenario is considered, the argument fails. For example,

[20]note that the plays of ρ do not contain the name-lists $a_1 a_2$ because the names a_1, a_2 are introduced by $*_F$, which is an O-move for ρ.

the following is a valid play for the strategy (we omit some pointers to immediately preceding moves).

$$\begin{tikzcd}[row sep=small, column sep=small]
\star & *_F^{a_1 a_2} & *_f \arrow[l, bend right] & a_1 \arrow[l, bend right] & *_f \arrow[lll, bend right] & a_1 \arrow[l, bend right] & \mathsf{t}_z & a_2 \arrow[lll, bend right] & \mathsf{f}_z & \cdots \\
O & P & O & P & O & P & O & P & O &
\end{tikzcd}$$

Note that in the argument of [2] it was essential to assume that there was ever only one name in the view of the play and therefore O had no option but play in the same way no matter what the name in the view was. This is no longer the case here, as the above play testifies: in the view of the last move, O can see *both* a_1 *and* a_2 and, for example reply f (the two names are not the same), where two moves before he had replied t. In fact, such an Opponent corresponds to the counter-strategy given by context (7).

There is no reason why O should stop at one nested call of f. This means that O may potentially see more than two names in his view, call f again after P closes one of his calls, etc. Thus, a direct argument in the style of [2] cannot go through. The following lemma solves the problem.

Lemma 30. *For any play s in $[\![\nu x_1.\nu x_2.\lambda f.\, f x_1 = f x_2]\!]$ of the form*

$$\begin{tikzcd}[row sep=small, column sep=small]
\star & \cdots & a_1 & \cdots & \mathsf{b}_1 \arrow[ll, bend right] & \cdots & a_2 & \cdots & \mathsf{b}_2 \arrow[ll, bend right] \\
O & & P & & O & & P & & O
\end{tikzcd}$$

in which O plays innocently, if $\ulcorner \star \cdots a_1 \urcorner = (a_1\ a_2) \cdot \ulcorner \star \cdots a_2 \urcorner$ then $\mathsf{b}_1 = \mathsf{b}_2$.[21]

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

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

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

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

where we attach superscripts to distinguish different occurrences of a_1, a_2. By hypothesis, $\ulcorner \star \cdots a_1 \urcorner = (a_1\ a_2) \cdot \ulcorner \star \cdots a_2 \urcorner$ and therefore:

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

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

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

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

[21] Recall that $(a_1\ a_2) \cdot s$ is the sequence obtained from s by swapping all occurrences of the names a_1 and a_2 inside it.

We can now prove (5).

PROOF. Applying the previous lemma to the play

$$\begin{tikzcd}[column sep=small]
\star & *_F & *_f & a_1 \arrow[l, bend right] & \cdots & \mathsf{b}_1 \arrow[ll, bend right] & a_2 \arrow[llll, bend right] & \cdots & \mathsf{b}_2 \arrow[ll, bend right] & \mathsf{b}_z \arrow[lllllll, bend right] \\
O & P & O & P & & O & P & & O & P
\end{tikzcd}$$

with its last move omitted we obtain $\mathsf{b}_1 = \mathsf{b}_2$ and therefore $\mathsf{b} = \mathsf{t}$, and the argument proceeds as in [2]. □

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. Related work and concluding remarks

The proofs we presented herein embody three different approaches to program reasoning in languages with state.

Logical relations. Although the first use of such (unary) relations probably goes back to Tait [48], the notion was formally introduced by Plotkin in the early 70's [41]. Later on, Sieber [44] and O'Hearn and Riecke [38] attacked the full-abstraction problem for PCF by considering models with continuous functions invariant under logical relations. Based on previous work by Pitts and Stark [39], Stark [45] developed the method we presented in Section 3 with the aim to construct logical relations fully capturing observational equivalence in the nu-calculus. Logical relations for the nu-calculus were revisited from a denotational viewpoint by Zhang and Nowak [52], without improving on completeness or proving (5), themselves building on the relational reasoning technique for monadic effects of Goubault-Larrecq, Lasota and Nowak [17]. Extensions of the nu-calculus with proper local state (i.e. where names are used as references of integer or higher-order type) have been examined extensively via relational methods. The first major contribution in this direction was a fully abstract relational model for a language with integer references, constructed by Pitts and Stark [40]. Subsequent works have managed to capture substantial parts of realistic languages involving features such as higher-order references, polymorphism, recursive types and control. The work of Ahmed, Birkedal, Dreyer and coauthors [7, 10, 11] (see also references therein) has been instrumental in these latter developments.

Bisimulation techniques. Bisimulations were introduced by Hennessy and Milner [18] for characterising program equivalence in the presence of non-determinism and concurrency. They nowadays constitute a dominant reasoning tool in process calculi [43]. The passage to general programs was made with the introduction of the lazy lambda-calculus and its theory of applicative bisimulations by Abramsky [1]. Gordon and Rees [16] devised a bisimulation equivalence fully abstract for a calculus with objects and subtyping, while Tiuryn and Wand [49] did the same for an untyped lambda-calculus with input/output. Extending the trace bisimulation technique for lambda-calculus terms introduced by Gordon [15] and Bernstein and Stark [9], Jeffrey and

Rathke [22] built bisimulation models for lambda-calculi with local names. These captured adequately and completely the nu-calculus with pointers (i.e. names store names), but soundness failed for the nu-calculus itself. The authors subsequently extended the method to a concurrent calculus with dynamically generated higher-order channels and threads [23]. In the work of Sumii and Pierce [46, 47] we see the introduction of environmental bisimulations, that is, sets of bisimulations with each relation defined at a particular stage in a dynamically changing state. In subsequent work of Koutavas and Wand [27, 26, 24] and Sangiorgi, Kobayashi and Sumii [42] the technique took off and captured a wider variety of effects, including higher-order store and objects. It is in this strand of work that the method of Benton and Koutavas [8] of Section 4 is classified. In recent work, Koutavas, Levy and Sumii [25] study in depth the additional expressiveness that environments bring to bisimulations and provide specific examples which classify the limitations of applicative versus environmental bisimulation.

Game semantics. The use of game-related ideas in semantics of programs was instigated by the problem of full abstraction for the purely functional language PCF, and the realisation that modelling of sequential computation required functions with intentional content. Thus, in the early 90's, in order to solve the problem a denotational theory of game semantics was deployed, independently by Abramsky, Jagadeesan and Malacaria [3], Hyland and Ong [21], and Nickau [37]. What followed was a first wave of results providing fully abstract semantics for a wide variety of programming paradigms, such as local state, non-determinism, control and general references. By the end of the 90's, applications to control-flow analysis started to emerge, accompanied by the development of algorithmic methods for reasoning about game models.[22] However, those approaches were limited in their treatment of names, and in effect circumvented names altogether by modelling them as products.[23] The characteristic example of this modus operandi was the game model of Idealized Algol, a language with ground local state, by Abramsky and McCusker [4]. Shortcomings in the treatment of names were addressed by Abramsky, Ghica, Murawski, Ong and Stark [2] and Laird [28] with the introduction of nominal game semantics. The latter involved games with a built-in notion of names as atomic computational data, formally constructed inside the universe of nominal sets, as (re)introduced[24] by Gabbay and Pitts [12]. The AGMOS model, with rectifications of [50], captured the nu-calculus while Laird's model captured its extension with pointers. Since then, a series of nominal game models have emerged, devised by Laird, Murawski and the author [29, 50, 33, 30, 35], capturing dynamic effects including local state, higher-order concurrency and higher-order store. As exemplified in the work of Ghica, McCusker, Murawski and collaborators [14, 5, 20], game models combine full abstraction with concrete representations which allow for algorithmic descriptions in terms of abstract machines. In applying algorithmic reasoning on nominal games, and nominal computation in general, the machines one is led to consider operate over infinite alphabets [36, 51], as names are infinite in number. In recent work with Murawski [34], a fragment of a language with finite ground local-state is captured in terms of such machines, thus providing an automated method for deciding program equivalence in such a setting.

[22] See [13] for an overview of these accomplishments and extensive references.

[23] In particular, names would be modelled *extensionally*, as products of type dictated by the modelled behaviour. For example, ground references corresponded to pairs of read/write functions. In order to achieve completeness, constructs of reference type which would not behave as actual references (so-called *bad variables*), e.g. due to their read/write methods being unrelated, needed to be included in the language. The latter heavily affect observational equivalence, with basic things like $x{:=}1; !x \cong x{:=}1; 1$ failing, and in practice disallow simple features like reference-equality tests.

[24] Nominal sets trace back to the 20's and in particular to Fraenkel-Mostowski permutation models of set theory with atoms (ZFA).

Conclusion. In this paper, we examined several approaches to proving program equivalences of the nu-calculus. Although very powerful, the methods presented have their limitations, and none of them is fully satisfactory. In particular, we are yet unable to answer the question:

Is program equivalence in the nu-calculus decidable?

We conjecture it is decidable. We envisage that one should attack the problem by combining techniques. Environmental bisimulations give a full proof method which, however, potentially leads to examining an infinity of contexts in order to form relation closures and relate function terms. As we saw in the proof of (5), though, in proving specific equivalences not all these contexts are needed. In making the choice of the right contexts systematic, one could use game semantics as a precise representation of the observable behaviour of the terms in question. This is left for future work.

Acknowledgements

Many thanks to Vasilis Koutavas, Paul-Andre Mellies, Andrzej Murawski and Ian Stark for fruitful discussions, explanations and suggestions. This work was supported by a Royal Academy of Engineering research fellowship.

References

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

[2] 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.

[3] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000.

[4] S. Abramsky and G. McCusker. Linearity, Sharing and State: a fully abstract game semantics for Idealized Algol. In P. O'Hearn and R. D. Tennent, editors, *Algol-like languages*, volume 2, pages 297–329. Birkhäuser, Boston, 1997.

[5] S. Abramsky and C.-H. L. Ong. Algorithmic game semantics and its applications: Final report, 2006.

[6] S. Abramsky and N. Tzevelekos. Introduction to categories and categorical logic. In *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*, pages 3–94. Springer, 2011. Available on the arXiv.

[7] A. Ahmed, D. Dreyer, and A. Rossberg. State-dependent representation independence. In *POPL*, pages 340–353, 2009.

[8] 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.

[9] K. L. Bernstein and E. W. Stark. Operational semantics of a focusing debugger. In *MFPS*, volume 1 of *Electronic Notes in Theoretical Computer Science*, pages 13 – 31, 1995.

[10] L. Birkedal, K. Støvring, and J. Thamsborg. Realisability semantics of parametric polymorphism, general references and recursive types. *Mathematical Structures in Computer Science*, 20(4):655–703, 2010.

[11] D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *ICFP*, pages 143–156, 2010.

[12] 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.

[13] D. R. Ghica. Applications of game semantics: From program analysis to hardware synthesis. In *LICS*, pages 17–26, 2009.

[14] D. R. Ghica and G. McCusker. Reasoning about Idealized Algol using regular languages. In *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 103–115, 2000.

[15] A. D. Gordon. Bisimilarity as a theory of functional programming. In *MFPS*, volume 1 of *Electronic Notes in Theoretical Computer Science*, pages 232 – 252, 1995.

[16] A. D. Gordon and G. D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *POPL*, pages 386–395, 1996.

[17] J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. *Mathematical Structures in Computer Science*, 18(6):1169–1217, 2008.

[18] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309, 1980.

[19] K. Honda and N. Yoshida. Game-theoretic analysis of call-by-value computation. *Theoretica Computer Science*, 221(1-2):393–456, 1999.

[20] D. Hopkins, A. S. Murawski, and C.-H. L. Ong. A fragment of ML decidable by visibly pushdown automata. In *ICALP*, volume 6756 of *Lecture Notes in Computer Science*, pages 149–161, 2011.

[21] J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000.

[22] A. Jeffrey and J. Rathke. Towards a theory of bisimulation for local names. In *LICS*, pages 56–66, 1999.

[23] A. Jeffrey and J. Rathke. A theory of bisimulation for a fragment of concurrent ML with local names. *Theoretical Computer Science*, 323(1-3):1–48, 2004.

[24] V. Koutavas. *Reasoning about Imperative and Higher-Order Programs*. PhD thesis, Northeastern University, 2008.

[25] V. Koutavas, P. B. Levy, and E. Sumii. From applicative to environmental bisimulation. In *MFPS*, volume 276 of *Electronic Notes in Theoretical Computer Science*, pages 215 – 235, 2011.

[26] V. Koutavas and M. Wand. Bisimulations for untyped imperative objects. In *ESOP*, volume 3924 of *Lecture Notes in Computer Science*, pages 146–161, 2006.

[27] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative programs. In *POPL*, pages 141–152, 2006.

[28] J. Laird. A game semantics of local names and good variables. In *FoSSaCS*, volume 2987 of *Lecture Notes in Computer Science*, pages 289–303, 2004.

[29] J. Laird. Game semantics for higher-order concurrency. In *FSTTCS*, volume 4337 of *Lecture Notes in Computer Science*, pages 417–428, 2006.

[30] J. Laird. A game semantics of names and pointers. *Annals of Pure and Applied Logic*, 151(2-3):151–169, 2008.

[31] R. Loader. Finitary PCF is not decidable. *Theoretical Computer Science*, 266:342–364, 2001.

[32] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[33] A. S. Murawski and N. Tzevelekos. Full abstraction for Reduced ML. In *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 32–47, 2009.

[34] A. S. Murawski and N. Tzevelekos. Algorithmic nominal game semantics. In *ESOP*, volume 6602 of *Lecture Notes in Computer Science*, pages 419–438, 2011.

[35] A. S. Murawski and N. Tzevelekos. Game semantics for good general references. In *LICS*, pages 75–84, 2011.

[36] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transanctions on Computational Logic*, 5(3):403–435, 2004.

[37] H. Nickau. *Hereditarily Sequential Functionals: A Game-Theoretic Approach to Sequentiality*. PhD thesis, University of Siegen, 1996.

[38] P. W. O'Hearn and J. G. Riecke. Kripke logical relations and PCF. *Information and Computation*, 120(1):107–116, 1995.

[39] 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.

[40] A. M. Pitts and I. D. B. Stark. *Operational reasoning for functions with local state*, pages 227–274. Cambridge University Press, 1998.

[41] G. D. Plotkin. λ-definability and logical relations. Technical Report SAI-RM-4, School of Artificial Intelligence, University of Edinburgh, 1973.

[42] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-order languages. *ACM Trans. Program. Lang. Syst.*, 33(1):5, 2011.

[43] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[44] K. Sieber. Reasoning about sequential functions via logical relations. In P. T. J. M. P. Fourman and A. M. Pitts, editors, *Applications of Categories in Computer Science*. Cambridge University Press, 1992.

[45] I. D. B. Stark. *Names and Higher-Order Functions*. PhD thesis, University of Cambridge, 1994.

[46] E. Sumii and B. C. Pierce. A bisimulation for dynamic sealing. In *POPL*, pages 161–172, 2004.

[47] E. Sumii and B. C. Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54(5), 2007.

[48] W. W. Tait. Intensional interpretations of functionals of finite type I. *The Journal of Symbolic Logic*, 32(2):198–212, 1967.

[49] J. Tiuryn and M. Wand. Untyped lambda-calculus with input-output. In *CAAP*, volume 1059 of *Lecture Notes in Computer Science*, pages 317–329, 1996.

[50] N. Tzevelekos. Full abstraction for nominal general references. *Logical Methods in Computer Science*, 5(3), 2009.

[51] N. Tzevelekos. Fresh-register automata. In *POPL*, pages 295–306, 2011.

[52] Y. Zhang and D. Nowak. Logical relations for dynamic name creation. In *CSL*, volume 2803 of *Lecture Notes in*

Computer Science, pages 575–588, 2003.