Denotational Semantics with Nominal Scott Domains

STEFFEN LÖSCH and ANDREW M. PITTS, University of Cambridge

When defining computations over syntax as data, one often runs into tedious issues concerning α-equivalence and semantically correct manipulations of binding constructs. Here we study a semantic framework in which these issues can be dealt with automatically by the programming language. We take the user-friendly 'nominal' approach in which bound objects are named. In particular, we develop a version of Scott domains within nominal sets and define two programming languages whose denotational semantics are based on those domains. The first language, $\lambda\nu$-PCF, is an extension of Plotkin's PCF with names that can be swapped, tested for equality and locally scoped; although simple, it already exposes most of the semantic subtleties of our approach. The second language, PNA, extends the first with name abstraction and concretion so that it can be used for metaprogramming over syntax with binders.

For both languages, we prove a full abstraction result for nominal Scott domains analogous to Plotkin's classic result about PCF and conventional Scott domains: two program phrases have the same observable operational behaviour in all contexts if and only if they denote equal elements of the nominal Scott domain model. This is the first full abstraction result we know of for languages combining higher-order functions with some form of locally scoped names which uses a domain theory based on ordinary extensional functions, rather than using the more intensional approach of game semantics.

To obtain full abstraction, we need to add two functionals, one for existential quantification over names and one for 'definite description' over names. Only adding one of them is not enough, as we give counter examples to full abstraction in both cases.

Categories and Subject Descriptors: F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Denotational semantics*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Lambda calculus and related systems*

General Terms: Languages, Theory

Additional Key Words and Phrases: Metaprogramming, denotational semantics, domain theory, full abstraction, nominal sets, symmetry

ACM Reference Format:
Steffen Lösch and Andrew M. Pitts. 2014. Denotational Semantics with Nominal Scott Domains. *J. ACM* X, X, Article XX (2014), 45 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

If one wants to express computations not just with numbers, but also with structures representing the programs of a programming language, or the formulas of a logic, then one is in a setting called *metaprogramming*. It arises for example in mechanised theorem proving, or in domain specific languages. We distinguish between an *object-language* and a *meta-language*: we write algorithms in the meta-language for

This work is supported by a Gates Cambridge Scholarship and the ERC Advanced Grant *Events, Causality and Symmetry* (ECSYM).
Authors' address: S. Lösch and A. M. Pitts, University of Cambridge Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom.

DOI: http://dx.doi.org/10.1145/0000000.0000000

manipulating object-language syntax. For example, if we are using the `Coq` system [coq.inria.fr] to prove theorems about `C` programs, then `C` is the object-language and `Coq` is the meta-language.

Whenever the object-language has name-binding constructs (and most programming languages do), we run into tedious issues regarding α-equivalence (renaming of bound names) that programmers like to gloss over, but that have to be dealt with in the design of a meta-language. The informal approach is to use the first-order representation of binding as (bound name, body)-pairs and not carry out tedious formal checks that α-equivalence is respected by computations. A common formal approach to representing object-language binders is to use function expressions in the meta-language to represent binding operations. However, the notion of 'function' is not absolute in the way that the notion of 'pair' is. As a result it requires some ingenuity (for example, distinguishing functions-as-data from functions-as-computation [Licata and Harper 2009]) to ensure that a meta-language using this form of representation for binders can conveniently express the wide range of first-order syntax-manipulating algorithms commonly employed in informal practice; see Savary-Belanger et al. [2013], for example.

An alternative approach to this problem, that we follow here, is to devise meta-languages that represent object-language binders using the permutation-based notion of *name abstraction* [Pitts 2013, Chapter 4]. This approach lies somewhere between the first-order and the functional representation and turns out to have good computational properties [Shinwell et al. 2003; Pitts 2011]. In this paper we will investigate the denotational semantics of name abstraction using a 'symmetry-aware' version of Scott domains [Scott 1982].

Various forms of symmetry are used in many branches of mathematics and computer science. The results in this paper have to do with using symmetry to extend the reach of computation theory from finite data structures and algorithms to ones that, although they are infinite, *become finite when quotiented by a suitable notion of symmetry*. Our motivating example of such data is the abstract syntax trees for an object-language involving name-binding constructs, since infinitely many abstract syntax trees represent the same expression modulo permuting their bound names. This way of viewing α-equivalence via symmetry was the initial stimulus for the development of *nominal sets*, introduced by Gabbay and Pitts [2002]. They provide a theory for mathematical structures involving atomic names[1] based on name permutations and the notion of *finite support*. We review the concept of nominal set briefly in Section 2, referring the reader to Pitts [2013] for a comprehensive account.

Nominal sets have been used to develop the semantic properties of binders and locally scoped names, with applications to functional and logic programming, to equational logic and rewriting, to type theory and to interactive theorem proving; see Gabbay [2011] and Pitts [2013] for recent surveys. The work by Montanari and Pistore [2000] on the π-calculus and HD-automata provides a somewhat different application of nominal sets (an independent one, since it uses a notion of 'named set' that only subsequently was shown to be equivalent to nominal sets [Gadducci et al. 2006; Staton 2007]). The use of symmetries of names (of fresh communication channels in this case) to get finite representations of infinitely many states is at the forefront in their work. It has recently been subsumed and generalised in a programme of what one might call 'orbit-finite' automata theory [Tzevelekos 2011; Bojańczyk et al. 2011; Bojańczyk and Lasota 2012; Gabbay and Ciancia 2011; Murawski and Tzevelekos 2012].

In this paper we bring together the 'names and binders' and the 'orbit-finite state space' aspects of nominal sets. We observe that a key concept underlying the automata-theoretic research programme of Bojańczyk et al. [2012], that of being an *orbit-finite*

[1]Names whose only attribute is their identity; Harper [2013, part XII] calls them 'symbols'.

subset, turns out to subsume a notion of topological compactness introduced, for quite different purposes, by Turner and Winskel [2009] in their work on nominal domain theory for concurrency. We explain the connection and use it to develop a version of the classic notion of Scott domain within nominal sets. (Previous work on denotational semantics with nominal sets [Shinwell and Pitts 2005; Turner 2009] has focussed on less sophisticated notions of domain, analogous either to ω-chain complete posets, or to algebraic lattices.) The well-known result of Plotkin [1977] proves that PCF with parallel-or is *fully abstract* with respect to conventional Scott domains, in the sense that two expressions have equal denotations if and only if they the have the same observable operational behaviour in all contexts. We obtain an analogous result for nominal Scott domains, through adding functionals for existential quantification over names and 'definite description' over names to a programming language for recursively defined higher-order functions with name abstractions and Odersky [1994] style locally scoped names.

Outline of the main results in this paper

— We show that the notion of finiteness used in Turner-Winskel nominal domain theory coincides with the notion of orbit-finite used by Bojańczyk et al. [2012]. Specifically, we prove (Theorem 4.5) that a finitely supported subset of a nominal set is compact with respect to unions that are uniform-directed in the sense of Turner and Winskel if and only if it is orbit-finite.

— We use orbit-finite subsets to generalise the notion of Scott domain from ordinary sets to nominal sets (Definition 5.1). We prove that the category of nominal Scott domains is cartesian closed (Theorem 5.5), has least fixed points and is closed under forming domains of name abstractions (Theorem 10.1). Although there are infinitely many names, the nominal Scott domain of names has some strong finiteness properties. In particular, we show that the functionals for existential quantification over names and definite description of names denotationally are uniform-compact elements of their function domains (Examples 5.7 and 5.8) and can be given a structural operational semantics.

— We define a language $\lambda\nu$-PCF that extends the well-known language PCF of Plotkin [1977] with names that can be locally scoped, swapped, and tested for equality. $\lambda\nu$-PCF's operational semantics is inspired by [Pitts 2011; Lösch and Pitts 2011]. In particular, it makes use of Odersky [1994] style local names, which have a simple denotational semantics using nominal Scott domains. We show that this semantics is computationally adequate, in the sense that equality of denotations implies that two programs are contextually equivalent for the operational semantics (Theorem 7.5).

— We show by counter examples that the reverse implication of computational adequacy can fail, that is, the denotational semantics is not fully abstract (Theorem 8.4), even if $\lambda\nu$-PCF is extended with either existential quantification over names or definite descriptions of names. However, extending $\lambda\nu$-PCF with both new constructs together, we prove that the nominal Scott domain model is fully abstract for the resulting language: any two expressions are contextually equivalent if and only if they have equal denotations in the model (Theorem 8.5).

— We illustrate how $\lambda\nu$-PCF can be the basis for a metaprogramming language for object-languages with binding operations by adding facilities for computing with name abstractions, together with a type representing λ-calculus terms. The resulting language, PNA (Programming with Name Abstractions), is given a denotational semantics using nominal Scott domains that is computationally adequate (Theorem 11.2). Furthermore, this denotational semantics is shown to be fully abstract once PNA is extended in the same way that $\lambda\nu$-PCF was extended, that is, with existential quantification over names and definite descriptions of names (Theorem 12.2).

There are full abstraction results for higher-order functions with local names using the intensional approach of game semantics [Abramsky et al. 2004; Tzevelekos 2008; Laird 2008], but our full abstraction Theorems 8.5 and 12.2 are the first such results we know of that are based on ordinary extensional functions. There is no similar result known for FreshML [Shinwell et al. 2003], which uses generative rather than Odersky-style local names to implement the features that PNA provides for programming with name abstractions; and yet we believe that PNA (extended with recursive types) is in principle as expressive as FreshML, in view of our previous work [Lösch and Pitts 2011]. Our proof of the full abstraction theorems seems novel compared with other proofs of similar full abstraction results in the literature [Curien 2007]. On the other hand it gives rise to some open problems that we discuss at the end of Section 12, together with a number of possibilities for future work exploiting the use of orbit-finite subsets within nominal domain theory.

Note

This is a revised and extended version of Lösch and Pitts [2013]. Compared with that article, here we show that most of the difficulty with our original full abstraction result (Theorem 12.2) is already present for the simple extension of PCF with locally scoped names rather than with name abstraction as well (Theorem 8.5). We thank the POPL reviewers for indirectly suggesting this approach and for other helpful comments.

In contrast to that article, here we do not extend $\lambda\nu$-PCF$^+$ explicitly with parallel-or, because it can in fact be expressed in terms of existential quantification over names (Remark 8.3). In addition we resolved the open problem of showing that the nominal Scott domain models of $\lambda\nu$-PCF+`the` and $\lambda\nu$-PCF+`ex` fail to be fully abstract, using examples suggested by Tzevelekos (Theorem 8.4), whom we thank for discussions on the topic of the paper.

2. FINITE SUPPORT

We are interested in the denotational semantics of programs written in languages featuring names that can be tested for equality and locally scoped by binding constructs. To take symmetry into account, we fix some countably infinite set $\mathbb{A}$, whose elements we call *atomic names*, and consider finite permutations of $\mathbb{A}$, that is, bijections $\pi : \mathbb{A} \cong \mathbb{A}$ with the property that $\pi\, a = a$ holds for all but finitely many $a \in \mathbb{A}$. Recall that an *action* of such permutations π on the elements x of a set X is a binary operation, written $(\pi, x) \mapsto \pi \cdot x$ and satisfying $\mathrm{id} \cdot x = x$ (where id is the identity permutation) and $\pi' \cdot (\pi \cdot x) = (\pi' \circ \pi) \cdot x$ (where $\_ \circ \_$ is composition), for all $x \in X$.

Actions of finite permutations of $\mathbb{A}$ on sets X and Y can be extended to their cartesian product $X \times Y$ by defining for each $x \in X$ and $y \in Y$

$$\pi \cdot (x, y) \triangleq (\pi \cdot x, \pi \cdot y). \tag{1}$$

More interestingly, given actions on X and Y, we get an action on the set of functions Y^X by defining for each $f \in Y^X$

$$\pi \cdot f \triangleq \lambda x \in X \to \pi \cdot (f(\pi^{-1} \cdot x)) \tag{2}$$

where π^{-1} is the inverse of the permutation π. In particular, taking $Y = 2 = \{\mathrm{true}, \mathrm{false}\}$, a two-element set with trivial action ($\pi \cdot \mathrm{true} = \mathrm{true}$, $\pi \cdot \mathrm{false} = \mathrm{false}$), we get an action on 2^X and hence on subsets of X. This action of finite permutations on subsets has a simple description: for each $S \subseteq X$

$$\pi \cdot S = \{\pi \cdot x \mid x \in S\}. \tag{3}$$

Programs, being finite syntactic objects, only involve finitely many atomic names in their construction; whereas the elements of a set X used to denote program behaviours

may well be infinite mathematical objects. We wish to limit our attention to infinite behaviours that depend only upon finitely many atomic names, as doing so yields a richer and better behaved theory. We can make precise what it means to 'only depend upon finitely many atomic names' entirely in terms of symmetry, that is, in terms of a given permutation action. An element $x \in X$ is *supported* by a set $A \subseteq \mathbb{A}$ of atomic names if every permutation π that preserves each name in A also preserves x:

$$((\forall a \in A)\ \pi\, a = a) \Rightarrow \pi \cdot x = x\,. \tag{4}$$

Definition 2.1 (*nominal set*). We say that a set X equipped with an action of finite permutations of $\mathbb{A}$ is a *nominal set* if each of its elements is supported by some finite set of atomic names. In this case one can show (see Pitts [2013, Proposition 2.3] for example) that for each $x \in X$ there is a smallest finite subset of $\mathbb{A}$ supporting x, which we write as $\operatorname{supp} x$. The *freshness relation* $a \# x$ is defined by:

$$a \# x \ \triangleq\ a \notin \operatorname{supp} x\,. \tag{5}$$

Note that since $\operatorname{supp} x$ is a finite set and $\mathbb{A}$ is not, given x we can always find some $a \in \mathbb{A}$ satisfying $a \# x$.

Given a nominal set X, the subsets that possess a finite support with respect to the action in (3) are called *finitely supported subsets* of X. Not every subset is finitely supported. For example, when $X = \mathbb{A}$ (with action $\pi \cdot a = \pi\, a$), the only finitely supported subsets are those $S \subseteq \mathbb{A}$ for which either S, or $\mathbb{A} - S$ is finite [Pitts 2013, Proposition 2.9]. Similarly, not every function $f \in Y^X$ between nominal sets X and Y is finitely supported with respect to the action in (2). Functions with empty support are called *equivariant* and all such functions satisfy for any finite permutation π and $x \in X$ that $f(\pi \cdot x) = \pi \cdot (f\, x)$.

We write $\mathrm{P}_{\mathrm{fs}} X$ for the collection of all finitely supported subsets of X. With the action in (3), this is a nominal set; indeed it is the power object (in the sense of topos theory [Johnstone 2002]) for a model of higher-order logic based on nominal sets. The main difference between this model and the classical one is that it fails to satisfy the Axiom of Choice: see Pitts [2013, Section 2.7].[2] As we discuss next, this difference causes nominal domain theory to be something more than just 'classical domain theory carried out in the nominal model of higher-order logic'.

3. UNIFORM-DIRECTED LEAST UPPER BOUNDS

In this section we recall the nominal domain theory introduced by Turner and Winskel [2009]. A key idea behind domain theory in general is to give a denotation to a program with potentially infinite behaviour as a limit of approximations. For domain theory based on approximation via a partial order (rather than a metric), limits are least upper bounds of chains (totally ordered subsets), or more generally, least upper bounds of subsets that are directed (every finite set of elements in the subset has an upper bound in the subset). So long as one considers chains of arbitrary (ordinal) length, classically there is no difference between using least upper bounds of chains and using least upper bounds of directed subsets [Markowsky 1976]. However, the equivalence of the two approaches relies on the Axiom of Choice and, as we noted above, that fails to hold for nominal sets. Thus in a nominal version of domain theory, formulating limits in terms of least upper bounds of chains leads to a different notion of domain than does the use of least upper bounds of arbitrary directed subsets (of course, both the chains and the directed subsets should be finitely supported, to make sense nominally).

[2]The associated model of set theory goes back to work in the 1930s by Fraenkel and Mostowski, who devised it specifically to negate the Axiom of Choice; see Gabbay [2011, Remark 2.22].

Turner and Winskel provide a compelling reason for preferring the former kind of limit: they show by example that a key notion provided by the nominal approach, the operation of name abstraction, does not always preserve least upper bounds of finitely supported, directed subsets. Example 3.2 gives a simplified version of the Winskel-Turner example of the failure of name abstraction to preserve least upper bounds of all finitely supported, directed subsets.

Definition 3.1 (*name abstraction*). A *nominal poset* is a nominal set D equipped with a partial order $\sqsubseteq$ that is respected by the action of permutations: $d \sqsubseteq d' \Rightarrow \pi \cdot d \sqsubseteq \pi \cdot d'$. Given such a D, we get a pre-order on $\mathbb{A} \times D$ by defining $(a, d) \sqsubseteq (a', d')$ to hold whenever we have $(a\ a'') \cdot d \sqsubseteq (a'\ a'') \cdot d'$ in D for some (or indeed, for any) $a'' \# (a, a', d, d')$. (As usual, $(a\ a')$ denotes the permutation that swaps a and a', leaving all other atomic names fixed.) We write $[\mathbb{A}]D$ for the poset obtained by quotienting $\mathbb{A} \times D$ by the equivalence relation associated with this pre-order, and $\langle a \rangle d$ for the equivalence class of (a, d). Defining a permutation action by $\pi \cdot \langle a \rangle d = \langle \pi\, a \rangle (\pi \cdot d)$, one can show that $[\mathbb{A}]D$ is also a nominal poset, with $\mathrm{supp}\langle a \rangle d = (\mathrm{supp}\, d) - \{a\}$. We call the elements of $[\mathbb{A}]D$ *name abstractions*. They are a generalized form of α-equivalence class for elements of D – generalized, because D itself may not consist of concrete syntactic data (we just need to know how name permutations act on its elements).

Example 3.2. For any nominal set X, partially ordering the elements of the nominal set $\mathrm{P}_{\mathrm{fs}}X$ of finitely supported subsets of X by inclusion, we get a nominal poset. Consider the case when $X = \mathbb{A}$. Given $a \in \mathbb{A}$, the name abstraction function $\mathrm{P}_{\mathrm{fs}}\mathbb{A} \to [\mathbb{A}](\mathrm{P}_{\mathrm{fs}}\mathbb{A})$ mapping each $S \in \mathrm{P}_{\mathrm{fs}}\mathbb{A}$ to $\langle a \rangle S$ does not preserve all least upper bounds of finitely supported, directed subsets. For example, consider the directed subset $\mathcal{F} \in \mathrm{P}_{\mathrm{fs}}(\mathrm{P}_{\mathrm{fs}}\mathbb{A})$ consisting of all finite sets of atomic names. $\mathcal{F}$ has empty support and its least upper bound $\bigsqcup \mathcal{F}$ is equal to $\mathbb{A}$. However, fixing upon $a \neq a'$ in $\mathbb{A}$, one has $\langle a \rangle (\bigsqcup \mathcal{F}) = \langle a \rangle \mathbb{A} = \langle a' \rangle \mathbb{A} \not\sqsubseteq \langle a' \rangle (\mathbb{A} - \{a\})$ and one can check that $\bigsqcup \{\langle a \rangle F \mid F \in \mathcal{F}\} \sqsubseteq \langle a' \rangle (\mathbb{A} - \{a\})$ (see Pitts [2013, Proposition 11.5]). So $\langle a \rangle (\bigsqcup \mathcal{F}) \neq \bigsqcup \{\langle a \rangle F \mid F \in \mathcal{F}\}$.

Despite this failure of name abstraction to preserve least upper bounds of arbitrary (finitely supported) directed subsets, we will see (Theorem 10.1) that it does preserve least upper bounds of finitely supported chains. Preservation of limits of approximations by a semantic operation is crucial if it is to be used for giving the denotational semantics of some linguistic construct. So one is naturally led to consider a nominal domain theory based upon the use of least upper bounds of finitely supported chains. However, this does not mean that one has to give up entirely the convenience of using directed sets. This is because closure under least upper bounds of finitely supported chains turns out to be equivalent to closure under least upper bounds of directed sets that are not merely finitely supported, but are uniformly supported in the sense of Turner and Winskel [2009, Definition 1]:

Definition 3.3 (*uniform support*). Let X be a nominal set. A subset $S \subseteq X$ is *uniformly supported* if there exists a finite set A of atomic names that supports each $x \in S$. Note that if this is the case, then for all finite permutations π satisfying $(\forall a \in A)\ \pi\, a = a$ we have $\pi \cdot S = \{\pi \cdot x \mid x \in S\} = \{x \mid x \in S\} = S$; so as well as supporting each element of S, A supports S itself. Thus a uniformly supported subset of X is in particular a finitely supported one.

For example, any finite $A \subseteq_{\mathrm{f}} \mathbb{A}$ is a uniformly supported subset (since A itself is a common finite support for each $a \in A$), whereas $\mathbb{A} - A$ is finitely supported (by A), but not uniformly supported (since any support for each $b \in \mathbb{A} - A$ has to contain b).

The crucial observation is that finitely supported chains in nominal posets have to be uniformly supported (as usual, a *chain* in D is a subset that is totally ordered by $\sqsubseteq$):

LEMMA 3.4. *[Turner 2009, Lemma 3.4.2.1] In a nominal poset D, every finitely supported chain C is necessarily uniformly supported.*

PROOF. Suppose that C is supported by the finite set $A \subseteq_{\mathrm{f}} \mathbb{A}$. We will show that A supports each $d \in C$.

For any finite permutation π, if $(\forall a \in A)\ \pi\, a = a$, then $\pi \cdot C = C$ (since A supports C). So if $d \in C$, then $\pi \cdot d \in C$. But C is totally ordered, so either $d \sqsubseteq \pi \cdot d$, or $\pi \cdot d \sqsubseteq d$. In the first case, since π is a finite permutation, for sufficiently large $n \in \mathbb{N}$ we have $\pi^n = \mathrm{id}$ and hence $d \sqsubseteq \pi \cdot d \sqsubseteq \pi^2 \cdot d \sqsubseteq \cdots \sqsubseteq \pi^n \cdot d = \mathrm{id} \cdot d = d$; and therefore $d = \pi \cdot d$ by antisymmetry of $\sqsubseteq$. Similarly, if $\pi \cdot d \sqsubseteq d$, then we also get $\pi \cdot d = d$. So in either case π preserves d. Since this holds for all π satisfying $(\forall a \in A)\ \pi\, a = a$, we have that A supports d. □

Definition 3.5 (*udcpo*). A *uniform-directed* subset of a nominal poset D is a subset $S \subseteq D$ that is both uniformly supported and directed (that is, each finite subset of S possesses an upper bound in S). A *uniform-directed complete partial order* (*udcpo*) is a nominal poset that has a least upper bound $\bigsqcup S$ for all uniform-directed subsets S.

As Turner [2009] points out, using Lemma 3.4, the classic result of Markowsky [1976] can be extended to show that *a nominal poset is a udcpo if and only if it has least upper bounds for all finitely supported chains*. So in effect udcpos give us a domain theory within the higher-order logic of nominal sets based on chain-completeness. As we will see in Section 10, they also give us access to the name abstraction construct.

We model potentially infinite program behaviours in languages with names using denotations that are uniform-directed least upper bounds of approximations to the behaviour. Each approximation should be finite in a suitable sense. For classical domain theory this amounts to being compact (also known as 'finite' or 'isolated') with respect to directed least upper bounds. By analogy, we have:

Definition 3.6 (*uniform compactness*). An element $u \in D$ of a udcpo D is *uniform-compact* if for all uniform-directed subsets $S \subseteq D$ it is the case that $u \sqsubseteq \bigsqcup S \Rightarrow (\exists d \in S)\ u \sqsubseteq d$. We write $\mathrm{K}D$ for the set of uniform-compact elements of D. We say that D is an *algebraic* udcpo if each of its elements is the least upper bound of a uniform-directed subset of $\mathrm{K}D$. D is *ω-algebraic* if in addition the underlying set of $\mathrm{K}D$ is countable.

Recall that a subset of a set is compact with respect to directed least upper bounds (unions) of subsets if and only if it is a finite set. Here we are restricting attention to a smaller class of least upper bounds, the uniform-directed ones. Therefore, one should expect uniform-compactness to be a more liberal notion of finiteness. Indeed, we show in the next section that it corresponds precisely to the notion of orbit-finite subset introduced by Bojańczyk et al. [2012, Section 3].[3]

4. ORBIT-FINITE SUBSETS

The action of finite permutations of $\mathbb{A}$ on the elements of a nominal set X partitions them into *orbits*: two elements x and x' are in the same orbit if $x' = \pi \cdot x$ for some finite permutation π. For example $\mathbb{A}$ itself has just one orbit; $\mathbb{A} \times \mathbb{A}$ has two, namely $\{(a, a) \mid a \in \mathbb{A}\}$ and $\{(a, a') \in \mathbb{A}^2 \mid a \neq a'\}$; and in general $\mathbb{A}^n$ has finitely many orbits, corresponding to equivalence relations on the finite set $\{0, 1, \ldots, n-1\}$. Contrastingly, the nominal set $\mathbb{A}^*$ of finite lists of atomic names has infinitely many orbits, since lists of different length cannot be in the same orbit.

[3]The term 'finitary subset' is used in that paper, but in subsequent work the authors use the terminology we have adopted here.

Remark 4.1 (*orbit-finite = finitely presentable*). The category-theoretic generalisation of the order-theoretic notion of directed least upper bound is the notion of *filtered colimit*; and compactness with respect to directed least upper bounds generalises to the notion of an object being *finitely presentable* (fp): an object X in a (locally small) category $\mathbf{C}$ with filtered colimits is fp if the hom-functor $\mathbf{C}(X, \_) : \mathbf{C} \to \mathbf{Set}$ preserves filtered colimits. $\mathbf{C}$ is called *locally finitely presentable* (lfp) if every object is the filtered colimit of fp objects [Gabriel and Ulmer 1971]. The category $\mathbf{Nom}$ of nominal sets and equivariant functions, being a Grothendieck topos, is lfp. Although we will not need the characterisation here, it is worth remarking that Petrişan [2011, Proposition 2.3.7] shows that *a nominal set is an fp object of* $\mathbf{Nom}$ *if and only if it is orbit-finite, that is, its set of orbits is finite.* See Pitts [2013, Section 5.3].

Definition 4.2 (*orbit-finite subsets*). A finitely supported subset $S \in \mathrm{P}_{\mathrm{fs}}X$ of a nominal set X is said to be *orbit-finite* if it is contained in the union of finitely many orbits of X. We write $\mathrm{P}_{\mathrm{of}}X$ for the collection of orbit-finite subsets of X.

Bojańczyk *et al.* investigate orbit-finite data structures and algorithms (for a generalised version of nominal sets over any 'Fraïssé symmetry'). Note that an orbit-finite subset may well have infinitely many different elements. For example, $\mathbb{A}$ is an orbit-finite subset of itself. Therefore, in order to compute with orbit-finite subsets one needs an effective presentation of them and of operations upon them. The following notion turns out to give an alternative characterisation of orbit-finite subsets that is suitable for calculation. It was introduced independently by Turner [2009, Definition 3.4.3.2], Gabbay [2009, Section 3.3; 2011, Definition 3.1; Gabbay and Ciancia 2011, Definition 3.1] and Bojańczyk et al. [2012, Section 8], whose 'hull' terminology we adopt here. (See also Ciancia and Montanari [2010, Definition 6.10], whose 'closures' are hulls of the form $\mathrm{hull}_{\mathrm{supp}\, x - \{a\}}\{x\}$.)

Definition 4.3 (*hulls*). Let X be a nominal set. Given finite subsets $A \subseteq_{\mathrm{f}} \mathbb{A}$ and $F \subseteq_{\mathrm{f}} X$, define $\mathrm{hull}_A F \triangleq \{\pi \cdot x \mid \pi \# A \wedge x \in F\}$, where $\pi \# A$ as usual means that π and A have disjoint support, which in this case means that the finite permutation π and the finite set of atomic names A satisfy $(\forall a \in A)\, \pi\, a = a$. We call such sets *hulls*.

LEMMA 4.4. *A subset of a nominal set X is orbit-finite if and only if it is a hull. For all finite subsets $A \subseteq_{\mathrm{f}} \mathbb{A}$ and $F \subseteq_{\mathrm{f}} X$, we have $\mathrm{hull}_A F \in \mathrm{P}_{\mathrm{of}}X$. Conversely, every $S \in \mathrm{P}_{\mathrm{of}}X$ is of the form $\mathrm{hull}_{\mathrm{supp}\, S} F$ for some $F \subseteq_{\mathrm{f}} X$.*

PROOF. We sketch the proof and refer the reader to Pitts [2013, Proposition 5.25] for the details. It is not hard to see that $\mathrm{hull}_A F$ is supported by A and is contained in a finite union of orbits of X, namely the orbits of each $x \in F$. What is less obvious is that every orbit-finite subset is of this form. This follows from a key technical property of hulls, proved independently by Turner [2009, Lemma 3.4.3.5] and Bojańczyk et al. [2012, Lemma 3]:

$$(\forall A \subseteq A' \subseteq_{\mathrm{f}} \mathbb{A})(\forall F \subseteq_{\mathrm{f}} X)(\exists F' \subseteq_{\mathrm{f}} X)\ \mathrm{hull}_A F = \mathrm{hull}_{A'} F'. \tag{6}$$

Now if S is orbit-finite, it is finitely supported and contained in the union of the orbits of the elements of some finite set $F \subseteq_{\mathrm{f}} X$; in other words $S \subseteq \mathrm{hull}_\emptyset F$. By (6), there exists $F' \subseteq_{\mathrm{f}} X$ with $\mathrm{hull}_\emptyset F = \mathrm{hull}_{\mathrm{supp}\, S} F'$ and from this it follows that $S = \mathrm{hull}_{\mathrm{supp}\, S}(F' \cap S)$. □

This lemma can be used to prove the following theorem that makes the connection between orbit-finite subsets and the notion of uniform-compactness from the previous section. Consider the nominal poset $\mathrm{P}_{\mathrm{fs}}X$ of finitely supported subsets of a nominal

set X, the partial order being subset inclusion. It possesses least upper bounds for all finitely supported subsets, given by union, and hence in particular it is a udcpo.

THEOREM 4.5. *An element of the udcpo $\mathrm{P}_{\mathrm{fs}}X$ is uniform-compact if and only if it is an orbit-finite subset of X. Every $S \in \mathrm{P}_{\mathrm{fs}}X$ is the uniform-directed least upper bound of the orbit-finite subsets contained in and with the same support as S. Thus $\mathrm{P}_{\mathrm{fs}}X$ is an algebraic udcpo in the sense of Definition 3.6.*

PROOF. Note that for every $S \in \mathrm{P}_{\mathrm{fs}}X$, since any $F \subseteq_{\mathrm{f}} S$ satisfies $F \subseteq \mathrm{hull}_{\mathrm{supp}\, S}F \subseteq S$, we have

$$S = \bigcup\{\mathrm{hull}_{\mathrm{supp}\, S}F \mid F \subseteq_{\mathrm{f}} S\} \tag{7}$$

and the right-hand side is a directed union of orbit-finite subsets that have common support $\mathrm{supp}\, S$. Therefore to prove the theorem it just suffices to prove that any S is uniform-compact if and only if it is orbit-finite.

Suppose $S \in \mathrm{P}_{\mathrm{of}}X$ and $S \subseteq \bigcup\mathcal{S}$ with $\mathcal{S}$ a uniform-directed subset of $\mathrm{P}_{\mathrm{fs}}X$. By Lemma 4.4, $S = \mathrm{hull}_A F$ for some $A \subseteq_{\mathrm{f}} \mathbb{A}$ and $F \subseteq_{\mathrm{f}} X$; and by (6) we may assume that A supports each element of $\mathcal{S}$. Since F is finite, $F \subseteq \bigcup\mathcal{S}$ and $\mathcal{S}$ is directed, we have $F \subseteq S'$ for some $S' \in \mathcal{S}$; and since A supports S', $F \subseteq S'$ implies that $\mathrm{hull}_A F \subseteq S'$, that is, $S \subseteq S'$. So S is uniform-compact.

Conversely, if $S \in \mathrm{P}_{\mathrm{fs}}X$ is uniform-compact, then in view of (7) we have $S \subseteq \mathrm{hull}_{\mathrm{supp}\, S}F$ for some $F \subseteq_{\mathrm{f}} S$. Hence $S = \mathrm{hull}_{\mathrm{supp}\, S}F$ is orbit-finite by Lemma 4.4. □

5. NOMINAL SCOTT DOMAINS

In view of Theorem 4.5 there is the following analogy

$$\frac{\text{finite}}{\text{directed}}\text{sets} \;\sim\; \frac{\text{orbit-finite}}{\text{uniform-directed}}\text{nominal sets}$$

which we apply to transfer to nominal sets the classical notion of *Scott domain* that arose in the denotational semantics of higher-order functional programming languages [Plotkin 1977, Lemma 4.4].

Definition 5.1. A *nominal Scott domain* D is an ω-algebraic udcpo with a least element and least upper bounds for all finitely supported subsets that have upper bounds (or equivalently, by Theorem 4.5, least upper bounds for all orbit-finite subsets of $\mathrm{K}D$ that have upper bounds). Functions between nominal Scott domains are called *uniform-continuous* if they are finitely supported, monotone, and preserve all uniform-directed least upper bounds. The category **Nsd** has nominal Scott domains for its objects and for its morphisms it has functions $f : D \to D'$ that are both equivariant and uniform-continuous.

Definition 5.2 (*flat domains*). If X is a nominal set, the *flat* nominal poset $X_\perp$ is given by $X \uplus \{\perp\}$, with partial order $d \sqsubseteq d' \Leftrightarrow d = \perp \vee d = d'$ and permutation action extending that on X by $\pi \cdot \perp = \perp$. It is easily seen to be a nominal Scott domain, with $\mathrm{K}(X_\perp) = X_\perp$, provided the underlying set of X is countable.

Remark 5.3 (*Turner-Winskel domain theory*). The *nominal domain theory for concurrency* of Turner and Winskel [2009] introduces the notion of 'uniform-directed least upper bound' and contains a characterisation of uniform-compact elements in terms of the hull construct from Definition 4.3. However, their domains are more specific than ours since they are based on path sets (downwards-closed subsets of preorders), which form prime-algebraic complete lattices. Modulo countability, their category $\mathbf{FMCts}_\emptyset$ is a full subcategory of **Nsd**.

LEMMA 5.4. *Let D be a udcpo. If an orbit-finite subset of $\mathrm{K}D$ possesses a least upper bound in D, then that least upper bound is also in $\mathrm{K}D$.*

PROOF. Suppose $K \in \mathrm{P}_{\mathrm{of}}(\mathrm{K}D)$ possesses a least upper bound and that $\bigsqcup K \sqsubseteq \bigsqcup S$ for some uniform-directed subset $S \subseteq D$. By Lemma 4.4 and (6) we can assume $K = \mathrm{hull}_A F$ where $F \subseteq_{\mathrm{f}} \mathrm{K}D$ and $A \subseteq_{\mathrm{f}} \mathbb{A}$ supports each $d \in S$. Since $F \subseteq \mathrm{hull}_A F = K$, we get $(\forall u \in F)\ u \sqsubseteq \bigsqcup K \sqsubseteq \bigsqcup S$; so since u is uniform-compact, F is finite and S is directed, there is some $d \in S$ satisfying $(\forall u \in F)\ u \sqsubseteq d$. Because A supports d, if follows that $(\forall u \in \mathrm{hull}_A F)\ u \sqsubseteq d$ and therefore $\bigsqcup K = \bigsqcup \mathrm{hull}_A F \sqsubseteq d$. So we do indeed have $\bigsqcup K \in \mathrm{K}D$. □

THEOREM 5.5. **Nsd** *is cartesian closed.*

PROOF. The terminal object is given by the trivial flat domain $\emptyset_\perp$. The product of D_1 and D_2 is given by their cartesian product, with permutation action as in (1) and partial order $(d_1, d_2) \sqsubseteq (d_1', d_2') \triangleq d_1 \sqsubseteq d_1' \wedge d_2 \sqsubseteq d_2'$. Exponentials $D_1 \to D_2$ have an underlying set consisting of all functions $f \in D_2^{D_1}$ that are uniform-continuous. The partial order on such functions is also given as usual, argument-wise: $f \sqsubseteq f' \triangleq (\forall d \in D_1)\ f\,d \sqsubseteq f'd$. That $D_1 \to D_2$ is a udcpo, has a least element, has least upper bounds of all finitely supported bounded subsets, and has the correct universal property for an exponential are all easy to verify. Less straightforward is its ω-algebraicity with respect to uniform-directed least upper bounds: for ordinary Scott domains, compact elements of the exponential are given by least upper bounds of finite, bounded sets of step-functions; here we use orbit-finite rather than finite sets. Given uniform-compact elements $u_i \in \mathrm{K}D_i$ $(i = 1, 2)$, consider the *step-function*

$$(u_1 \searrow u_2) \triangleq \lambda d \in D_1 \to \text{if } u_1 \sqsubseteq d \text{ then } u_2 \text{ else } \perp . \tag{8}$$

It is easily seen to be uniform-continuous (because u_1 is uniform-compact) and uniform-compact (because u_2 is uniform-compact). Let $(D_1 \to_{\mathrm{step}} D_2) \subseteq \mathrm{K}(D_1 \to D_2)$ be the set of all such step-functions, and note that this is a countable nominal set with $\pi \cdot (u_1 \searrow u_2) = (\pi \cdot u_1 \searrow \pi \cdot u_2)$. For algebraicity of $D_1 \to D_2$ we show that for each $f \in D_1 \to D_2$ we have $f = \bigsqcup S_f$, where

$$S_f \triangleq \{\textstyle\bigsqcup \mathrm{hull}_{\mathrm{supp}\, f} F \mid F \subseteq_{\mathrm{f}} K_f\}$$
$$K_f \triangleq \{(u_1 \searrow u_2) \in (D_1 \to_{\mathrm{step}} D_2) \mid (u_1 \searrow u_2) \sqsubseteq f\}.$$

Every $\mathrm{hull}_{\mathrm{supp}\, f} F$ is bounded by f and finitely supported by $\mathrm{supp}\, f$, so $\bigsqcup \mathrm{hull}_{\mathrm{supp}\, f} F$ exists and is uniform-compact by Lemma 5.4. This shows that S_f consists of uniform-compact elements, is bounded by f, is uniformly supported by $\mathrm{supp}\, f$ and is directed (as $\mathrm{hull}_A(\_)$ preserves inclusions), so its least upper bound exists and satisfies $\bigsqcup S_f \sqsubseteq f$.

To prove $f = \bigsqcup S_f$ we still need $f \sqsubseteq \bigsqcup S_f$, which by algebraicity of D_1 holds if $(\forall u_1 \in \mathrm{K}D_1)\ f\,u_1 \sqsubseteq (\bigsqcup S_f)\,u_1$. For each $u_1 \in \mathrm{K}D_1$, since D_2 is algebraic, we have $f\,u_1 = \bigsqcup U_2$ for some uniform-directed subset $U_2 \subseteq \mathrm{K}D_2$. For each $u_2 \in U_2$, $u_2 \sqsubseteq \bigsqcup U_2 = f\,u_1$ and hence $(u_1 \searrow u_2) \sqsubseteq f$. Therefore $(u_1 \searrow u_2) \in K_f$; and since $(u_1 \searrow u_2) \in \mathrm{hull}_{\mathrm{supp}\, f}\{(u_1 \searrow u_2)\}$ we get $(u_1 \searrow u_2) \sqsubseteq \bigsqcup \mathrm{hull}_{\mathrm{supp}\, f}\{(u_1 \searrow u_2)\} \in S_f$. Hence $(u_1 \searrow u_2) \sqsubseteq \bigsqcup S_f$ and thus $u_2 = (u_1 \searrow u_2)\,u_1 \sqsubseteq (\bigsqcup S_f)\,u_1$. Since this is true for each $u_2 \in U_2$, we get $f\,u_1 = \bigsqcup U_2 \sqsubseteq (\bigsqcup S_f)\,u_1$, as required.

Knowing $f \sqsubseteq \bigsqcup S_f$ also allows us to characterise the uniform-compact elements of $D_1 \to D_2$ by

$$\mathrm{K}(D_1 \to D_2) = \{\textstyle\bigsqcup S \mid S \in \mathrm{P}_{\mathrm{of}}(D_1 \to_{\mathrm{step}} D_2) \wedge S \text{ is bounded}\} \tag{9}$$

and this shows that $D_1 \to D_2$ is ω-algebraic, because $\mathrm{P}_{\mathrm{of}} X$ is countable if X is. □

We give some examples of uniform-compact elements of exponential objects in Nsd associated with the flat domain of atomic names, $\mathbb{A}_\perp$. The examples show that although $\mathbb{A}_\perp$ has a countably infinite underlying set, it has very different uniform-compactness properties from the flat domain of natural numbers, $\mathbb{N}_\perp$, for which the permutation action is *discrete*: $\pi \cdot n \triangleq n$. It turns out that these examples are important for the full abstraction results developed later in this paper.

Example 5.6 (*name equality test*). Let $2 = \{\mathrm{true}, \mathrm{false}\}$ be a two-element, discrete nominal set. For each atomic name $a \in \mathbb{A}$, consider the function $eq_a : \mathbb{A}_\perp \to 2_\perp$ given by

$$eq_a\, d \triangleq \begin{cases} \mathrm{true} & \text{if } d = a \\ \mathrm{false} & \text{if } d \in \mathbb{A} - \{a\} \\ \perp & \text{if } d = \perp \end{cases} \tag{10}$$

for each $d \in \mathbb{A}_\perp$. This function can be expressed in terms of hulls (Definition 4.3) and step-functions (8):

$$eq_a = \bigsqcup \mathrm{hull}_{\{a\}}\{(a \searrow \mathrm{true}), (a' \searrow \mathrm{false})\}$$

where a' is any atomic name not equal to a. Thus from (9) we have $eq_a \in \mathrm{K}(\mathbb{A}_\perp \to 2_\perp)$.

Example 5.7 (*exists name*). The function $exists_{\mathbb{A}} : (\mathbb{A}_\perp \to 2_\perp) \to 2_\perp$ defined by

$$exists_{\mathbb{A}} f \;\triangleq\; \begin{cases} \mathrm{true} & \text{if } (\exists a \in \mathbb{A})\ f\, a = \mathrm{true} \\ \mathrm{false} & \text{if } (\forall a \in \mathbb{A})\ f\, a = \mathrm{false} \\ \perp & \text{otherwise.} \end{cases} \tag{11}$$

is not only uniform-continuous, it is in fact uniform-compact. To see this, first note that picking any $a \in \mathbb{A}$, the least upper bound $\bigsqcup \mathrm{hull}_\emptyset\{(a \searrow \mathrm{false})\}$ exists and by (9) it is an element of $\mathrm{K}(\mathbb{A}_\perp \to 2_\perp)$. Observe that this least upper bound is exactly the function $k_{\mathrm{false}} \triangleq \lambda d \in \mathbb{A}_\perp \to \text{if } d = \perp \text{ then } \perp \text{ else false}$ and that

$$(\forall f \in \mathbb{A}_\perp \to 2_\perp)\ k_{\mathrm{false}} \sqsubseteq f \;\Leftrightarrow\; (\forall a \in \mathbb{A})\ f\, a = \mathrm{false}\,. \tag{12}$$

Furthermore, for any $g : (\mathbb{A}_\perp \to 2_\perp) \to 2_\perp$ we have $exists_{\mathbb{A}} \sqsubseteq g$ if and only if $(\forall f \in \mathbb{A}_\perp \to 2_\perp)\ exists_{\mathbb{A}}\, f \sqsubseteq g\, f$ if and only if

$$\begin{aligned}(\forall f \in \mathbb{A}_\perp \to 2_\perp)\ ((\exists a \in \mathbb{A})\ f\, a = \mathrm{true}) \Rightarrow g\, f = \mathrm{true} \\ \wedge\ ((\forall a \in \mathbb{A})\ f\, a = \mathrm{false}) \Rightarrow g\, f = \mathrm{false}\end{aligned}$$

which by definition of step-functions (8) and by (12) is equivalent to

$$(\forall f \in \mathbb{A}_\perp \to 2_\perp)(\forall a \in \mathbb{A})\ ((a \searrow \mathrm{true}) \sqsubseteq f \Rightarrow g\, f = \mathrm{true}) \wedge (k_{\mathrm{false}} \sqsubseteq f \Rightarrow g\, f = \mathrm{false})$$

or more simply, to $(\forall a \in \mathbb{A})\ ((a \searrow \mathrm{true}) \searrow \mathrm{true}) \sqsubseteq g \ \wedge\ (k_{\mathrm{false}} \searrow \mathrm{false}) \sqsubseteq g$. Thus picking some $a \in \mathbb{A}$, we have $exists_{\mathbb{A}} = \bigsqcup \mathrm{hull}_\emptyset\{((a \searrow \mathrm{true}) \searrow \mathrm{true}), (k_{\mathrm{false}} \searrow \mathrm{false})\}$ and by (9) we have $exists_{\mathbb{A}} \in \mathrm{K}((\mathbb{A}_\perp \to 2_\perp) \to 2_\perp)$.

Example 5.8 (*definite name description*). The name equality tests $eq_a : \mathbb{A}_\perp \to 2_\perp$ from Example 5.6 satisfy

$$\begin{aligned} \pi \cdot eq_a &= eq_{\pi\, a} \\ eq_a = eq_{a'} &\Rightarrow a = a' \\ eq_a \sqsubseteq f &\Rightarrow eq_a = f\,. \end{aligned}$$

From these properties and the uniform-compactness of eq_a it follows that the function $the_{\mathbb{A}} : (\mathbb{A}_\perp \to 2_\perp) \to \mathbb{A}_\perp$ defined by

$$the_{\mathbb{A}}\, f \triangleq \begin{cases} a & \text{if } f = eq_a \text{ for some } a \in \mathbb{A} \\ \perp & \text{otherwise} \end{cases} \tag{13}$$

satisfies for all $g : (\mathbb{A}_\perp \to 2_\perp) \to \mathbb{A}_\perp$ that $the_{\mathbb{A}} \sqsubseteq g$ holds if and only if $(\forall a \in \mathbb{A})\ (eq_a \searrow a) \sqsubseteq g$. Therefore picking any $a \in \mathbb{A}$, we have $the_{\mathbb{A}} = \bigsqcup \mathrm{hull}_\emptyset\{(eq_a \searrow a)\}$ and hence by (9) that $the_{\mathbb{A}} \in \mathrm{K}((\mathbb{A}_\perp \to 2_\perp) \to \mathbb{A}_\perp)$.

Remark 5.9. The analogue of the equality test function of Example 5.6 for natural numbers, $eq_n : \mathbb{N}_\perp \to 2_\perp$, is not uniform-compact. To see this, note that it can be expressed as the least upper bound of a finitely supported chain of functions $eq_n = \bigsqcup\{eq_{n,m} \mid m \in \mathbb{N}\}$, where $eq_{n,m}$ is defined by

$$eq_{n,m}\, d \triangleq \begin{cases} eq_n\, d & \text{if } d = m' \wedge m' \leq m \\ \perp & \text{otherwise} \end{cases}$$

If eq_n was uniform-compact, then $eq_n \sqsubseteq eq_{n,m}$ would hold for some $m \in \mathbb{N}$, which is not the case. We cannot use the same kind of argument for eq_a, because there is no finitely supported total ordering of the elements of $\mathbb{A}$.

The functionals for existential quantification and definite description of natural numbers $exists_{\mathbb{N}}$ and $the_{\mathbb{N}}$ not only fail to be uniform-compact, they are also not uniform-continuous. For $the_{\mathbb{N}}$, eq_n from above gives a counter-example to uniform-continuity, because $the_{\mathbb{N}}(eq_n) = n$ as well as $(\forall m \in \mathbb{N})\ the_{\mathbb{N}}(eq_{n,m}) = \perp$ hold. For $exists_{\mathbb{N}}$, observe that k_{false} from Example 5.7 (for numbers instead of names, so $k_{\mathrm{false}} : \mathbb{N}_\perp \to 2_\perp$) can be characterised by $k_{\mathrm{false}} = \bigsqcup\{k_{\mathrm{false},m} \mid m \in \mathbb{N}\}$, where

$$k_{\mathrm{false},m}\, d \triangleq \begin{cases} \mathrm{false} & \text{if } d = m' \wedge m' \leq m \\ \perp & \text{otherwise.} \end{cases}$$

This contradicts uniform-continuity of $exists_{\mathbb{N}}$, as $exists_{\mathbb{N}}(k_{\mathrm{false}}) = \mathrm{false}$ and $(\forall m \in \mathbb{N})\ exists_{\mathbb{N}}(k_{\mathrm{false},m}) = \perp$.

Remark 5.10 (*least fixed points*). If $D \in \mathbf{Nsd}$ and $f \in D \to D$, then $\{\perp, f\,\perp, f^2\,\perp, \ldots\}$ is a uniform-directed subset of D (each element is supported by $\mathrm{supp}\, f$) whose least upper bound is $\mathrm{fix}\, f$, the least fixed point of f, by the usual Tarskian argument. Indeed for each nominal Scott domain D, the function assigning least fixed points gives us a morphism in $\mathbf{Nsd}$

$$\mathrm{fix} : (D \to D) \to D. \tag{14}$$

6. SEMANTICS OF LOCALLY SCOPED NAMES

As for any cartesian closed category, Theorem 5.5 allows one to model the typed λ-calculus using the category $\mathbf{Nsd}$; and in view of Remark 5.10, $\mathbf{Nsd}$ also supports the usual domain-theoretic interpretation of recursively defined terms via least fixed points. For example, nominal Scott domains subsume the usual Scott semantics of Plotkin's language PCF [Plotkin 1977] for recursively defined higher-order numerical functions. At the same time $\mathbf{Nsd}$ can give denotational semantics for programming language features involving atomic names, such as local scoping and abstraction; furthermore this denotational semantics is pleasingly simple compared with the use of functor categories, for example by Stark [1994, Chapter 3]. We postpone discussing name abstraction until Section 9 and concentrate first on the $\mathbf{Nsd}$ semantics of locally scoped names.

$\tau \in \mathrm{Typ} ::=$	*types*
$\mathtt{bool} \mid \mathtt{nat} \mid \tau \times \tau \mid \tau \to \tau \mid \mathtt{name}$	
$e \in \mathrm{Exp} ::=$	*expressions*
x	variable $(x \in \mathbb{V})$
$\mathtt{T}$	truth
$\mathtt{F}$	falsity
$\mathtt{if}\, e\, \mathtt{then}\, e\, \mathtt{else}\, e$	conditional
$\mathtt{0}$	number zero
$\mathtt{S}\, e$	successor
$\mathtt{pred}\, e$	predecessor
$\mathtt{zero}\, e$	zero test
$(e\,, e)$	pair
$\mathtt{fst}\, e$	first projection
$\mathtt{snd}\, e$	second projection
$\lambda x : \tau \to e$	function abstraction
$e\, e$	function application
$\mathtt{fix}\, e$	fixed-point recursion
a	atomic name $(a \in \mathbb{A})$
$\nu a.\, e$	locally scoped name
$e = e$	name equality test
$(e \rightleftharpoons e)\, e$	name swapping
$c \in \mathrm{Can} ::=$	*canonical forms*
$\mathtt{T} \mid \mathtt{F} \mid \mathtt{0} \mid \mathtt{S}\, c \mid (e\,, e) \mid \lambda x : \tau \to e \mid a$	

Fig. 1. Syntax of $\lambda\nu$-PCF

To do so, we consider a combination of Plotkin's PCF with Odersky's $\lambda\nu$-calculus, a functional theory of local names [Odersky 1994]. The resulting programming language, which we call $\lambda\nu$-PCF, not only has arithmetic constructs and (call-by-name, higher-order) recursive functions, but also has a type of atomic names that can be tested for equality, locally scoped and explicitly swapped (in expressions of any type).

Figure 1 gives the syntax of $\lambda\nu$-PCF. In the grammar for expressions, the part below the dotted line is what is added to PCF. There are two kinds of identifier in the language: *variables* $x, y, z, f, \ldots \in \mathbb{V}$ and *atomic names* $a, b, c, \ldots \in \mathbb{A}$. The sets $\mathbb{V}$ of variables and $\mathbb{A}$ of atomic names are disjoint and countably infinite. Both kinds of identifier may be bound; binding forms are function abstraction $\lambda x : \tau \to \_$ (for variables) and local scoping $\nu a.\, \_$ (for atomic names). We identify expressions up to α-equivalence of bound identifiers. For any expression e, we write $\mathrm{fn}(e)$ for its finite set of free atomic names and $\mathrm{fv}(e)$ for its finite set of free variables.

The reason for making a syntactic distinction between variables x and atomic names a is that they behave differently: an occurrence of x in an expression stands for an unknown expression; whereas an occurrence of a denotes an entity whose identity is definitely different from the other atomic names that occur in the expression. Thus if x and y are two different variables, the meaning of the boolean expression $x = y$ is indeterminate, whereas if a and b are two different atomic names, then the meaning of $a = b$ is the boolean value false. For this reason various properties of $\lambda\nu$-PCF, such as its typing judgement, are preserved by the operation of substitution of expressions for variables, but are only preserved by permutations of atomic names rather than more general forms of substitution for names. The operation of simultaneous substitution

$$\frac{(x:\tau)\in\Gamma}{\Gamma\vdash x:\tau} \qquad \frac{c\in\{\mathtt{T},\mathtt{F}\}}{\Gamma\vdash c:\mathtt{bool}} \qquad \frac{\Gamma\vdash e_1:\mathtt{bool} \qquad \Gamma\vdash e_2:\tau \qquad \Gamma\vdash e_3:\tau}{\Gamma\vdash \mathtt{if}\, e_1\, \mathtt{then}\, e_2\, \mathtt{else}\, e_3:\tau} \qquad \frac{}{\Gamma\vdash \mathtt{0}:\mathtt{nat}}$$

$$\frac{\Gamma\vdash e:\mathtt{nat}}{\Gamma\vdash \mathtt{S}\, e:\mathtt{nat}} \qquad \frac{\Gamma\vdash e:\mathtt{nat}}{\Gamma\vdash \mathtt{pred}\, e:\mathtt{nat}} \qquad \frac{\Gamma\vdash e:\mathtt{nat}}{\Gamma\vdash \mathtt{zero}\, e:\mathtt{bool}} \qquad \frac{\Gamma\vdash e_1:\tau_1 \qquad \Gamma\vdash e_2:\tau_2}{\Gamma\vdash (e_1\,, e_2):\tau_1\times\tau_2}$$

$$\frac{\Gamma\vdash e:\tau_1\times\tau_2}{\Gamma\vdash \mathtt{fst}\, e:\tau_1} \qquad \frac{\Gamma\vdash e:\tau_1\times\tau_2}{\Gamma\vdash \mathtt{snd}\, e:\tau_2} \qquad \frac{\Gamma, x:\tau\vdash e:\tau'}{\Gamma\vdash \lambda x:\tau\to e:\tau\to\tau'}$$

$$\frac{\Gamma\vdash e_1:\tau\to\tau' \qquad \Gamma\vdash e_2:\tau}{\Gamma\vdash e_1\, e_2:\tau'} \qquad \frac{\Gamma\vdash e:\tau\to\tau}{\Gamma\vdash \mathtt{fix}\, e:\tau}$$

$$\frac{a\in\mathbb{A}}{\Gamma\vdash a:\mathtt{name}} \qquad \frac{a\in\mathbb{A} \qquad \Gamma\vdash e:\tau}{\Gamma\vdash \nu a.\, e:\tau} \qquad \frac{\Gamma\vdash e_1:\mathtt{name} \qquad \Gamma\vdash e_2:\mathtt{name}}{\Gamma\vdash e_1 = e_2:\mathtt{bool}}$$

$$\frac{\Gamma\vdash e_1:\mathtt{name} \qquad \Gamma\vdash e_2:\mathtt{name} \qquad \Gamma\vdash e_3:\tau}{\Gamma\vdash (e_1\rightleftharpoons e_2)\, e_3:\tau}$$

Fig. 2. $\lambda\nu$-PCF typing rules

of expressions $e_1,\ldots,e_n$ for distinct variables $x_1,\ldots,x_n$ in an expression e is written as $e[e_1/x_1,\ldots,e_n/x_n]$, where the substitution avoids capture of both free variables and free atomic names by the language's binding forms. The operation of applying a finite permutation $\pi:\mathbb{A}\cong\mathbb{A}$ to an expression e is written $\pi\cdot e$. It is defined by recursing into all sub-expressions and applying π to all occurrences of atomic names. This is a permutation action in the sense of Section 2 and makes the set Exp of $\lambda\nu$-PCF expressions into a nominal set. Since the elements of Exp are expressions up to α-equivalence, the smallest support of an element e of this nominal set of expressions is given by its free names $\mathrm{fn}(e)$; see Pitts [2013, Proposition 8.10].

The language $\lambda\nu$-PCF contains one syntactic form absent from Odersky's $\lambda\nu$-calculus: explicit swapping expressions, $(e_1\rightleftharpoons e_2)\, e$. Recall that every finite permutation is a composition of *transpositions* $(a\ a')$, the permutation that swaps a and a' while leaving every other atomic name fixed. It follows that the operation on expressions mapping e to $\pi\cdot e$ can be expressed in terms of the operations $e\mapsto(a\ a')\cdot e$ for all $a,a'\in\mathbb{A}$. The language $\lambda\nu$-PCF internalizes this name-swapping operation with expressions of the form $(e_1\rightleftharpoons e_2)\, e$, where e_1 and e_2 are expressions of type `name` that need not be specific atomic names (for example, they could be variables of type `name`). These expressions seem quite natural in the context of semantics based on nominal sets and we use them in the full abstraction proof in Section 8.

Definition 6.1 ($\lambda\nu$-PCF *typing*). $\lambda\nu$-PCF is a simply typed language. The grammar for types (Figure 1) extends that for PCF (in a version with products $\tau_1\times\tau_2$) with a type `name` of atomic names. The inductively defined typing judgement $\Gamma\vdash e:\tau$ (read as 'in the environment Γ the expression e has type τ') is defined in Figure 2 by the usual rules for PCF and, below the dotted line, rules concerning names. The *typing environments* $\Gamma=\{x_1:\tau_1,\ldots,x_n:\tau_n\}$ are finite functions from variables to types that track occurrences of free variables in e. For simplicity, $\lambda\nu$-PCF only features one sort of atomic name (many-sorted versions are possible; cf. Pitts [2013, Section 4.7]).

$$\frac{c \in \{\mathtt{T}, \mathtt{F}, \mathtt{0}, (e_1\,, e_2), \lambda x : \tau \to e\}}{c \Downarrow c} \qquad \frac{e \Downarrow c}{\mathtt{S}\, e \Downarrow \mathtt{S}\, c} \qquad \frac{e_1 \Downarrow \mathtt{T} \qquad e_2 \Downarrow c}{\mathtt{if}\, e_1\, \mathtt{then}\, e_2\, \mathtt{else}\, e_3 \Downarrow c}$$

$$\frac{e_1 \Downarrow \mathtt{F} \qquad e_3 \Downarrow c}{\mathtt{if}\, e_1\, \mathtt{then}\, e_2\, \mathtt{else}\, e_3 \Downarrow c} \qquad \frac{e \Downarrow \mathtt{S}\, c}{\mathtt{pred}\, e \Downarrow c} \qquad \frac{e \Downarrow \mathtt{0}}{\mathtt{zero}\, e \Downarrow \mathtt{T}} \qquad \frac{e \Downarrow \mathtt{S}\, c}{\mathtt{zero}\, e \Downarrow \mathtt{F}}$$

$$\frac{e \Downarrow (e_1\,, e_2) \qquad e_1 \Downarrow c}{\mathtt{fst}\, e \Downarrow c} \qquad \frac{e \Downarrow (e_1\,, e_2) \qquad e_2 \Downarrow c}{\mathtt{snd}\, e \Downarrow c} \qquad \frac{\begin{array}{c} e_1 \Downarrow \lambda x : \tau \to e \\ e[e_2/x] \Downarrow c \end{array}}{e_1\, e_2 \Downarrow c} \qquad \frac{e\,(\mathtt{fix}\, e) \Downarrow c}{\mathtt{fix}\, e \Downarrow c}$$

$$\frac{a \in \mathbb{A}}{a \Downarrow a} \qquad \frac{e \Downarrow c \qquad a \backslash\backslash c := c'}{\nu a.\, e \Downarrow c'} \qquad \frac{e_1 \Downarrow a \qquad e_2 \Downarrow a}{e_1 = e_2 \Downarrow \mathtt{T}} \qquad \frac{e_1 \Downarrow a \qquad e_2 \Downarrow a' \qquad a \neq a'}{e_1 = e_2 \Downarrow \mathtt{F}}$$

$$\frac{e_1 \Downarrow a_1 \qquad e_2 \Downarrow a_2 \qquad e_3 \Downarrow c}{(e_1 \rightleftharpoons e_2)\, e_3 \Downarrow (a_1\ a_2) \cdot c}$$

Fig. 3. $\lambda\nu$-PCF evaluation rules

Therefore the type of free atomic names is always `name` and we do not need to add a component to Γ tracking the types of free atomic names in e.

Definition 6.2 ($\lambda\nu$-PCF *evaluation*). In Figure 3 we extend PCF's usual rules for an inductively defined big-step evaluation relation with the rules below the dotted line that concern atomic names. Thus the relation $e \Downarrow c$ defines when a $\lambda\nu$-PCF expression e evaluates to *canonical form* c. (See Figure 1 for the grammar of canonical forms.) As for PCF, we only evaluate expressions that are *variable-closed* in the sense that $\mathrm{fv}(e) = \emptyset$. However, expressions for evaluation may contain free atomic names; this is because, unlike variables, atomic names are canonical forms.

The $\lambda\nu$-PCF evaluation relation is easily seen to be deterministic ($e \Downarrow c \wedge e \Downarrow c' \Rightarrow c = c'$) and type-sound ($\emptyset \vdash e : \tau \wedge e \Downarrow c \Rightarrow \emptyset \vdash c : \tau$). It is also equivariant:

$$e \Downarrow c \Rightarrow \pi \cdot e \Downarrow \pi \cdot c. \tag{15}$$

The evaluation rule in Figure 3 for local names makes use of the auxiliary definition in Figure 4, which implements the characteristic features of Odersky's functional theory of local names [Odersky 1994]: scopes intrude in a type-directed fashion. The operation of name restriction on canonical forms, $a, c \mapsto a\backslash\backslash c$, is partial because $a\backslash\backslash a := c$ holds for no c. Thus, unlike Pitts [2011], we choose to follow Odersky [1994] and make $\nu a.\, a$ a stuck expression that does not evaluate to any canonical form and whose denotation is $\bot$.

Remark 6.3 (*generative names*). The use of Odersky-style local names means that the operational semantics of $\lambda\nu$-PCF is stateless, unlike the operational semantics of the more usual, *generative* version of $\nu a.\, \_$ used in the ν-calculus [Pitts and Stark 1993] and most practical languages. However, Odersky-style local names are known to be as expressive as generative ones, at least in a simply typed setting. This follows from the existence of an adequate continuation-passing style translation from ν-calculus into the $\lambda\nu$-calculus [Lösch and Pitts 2011]. Indeed here we do not escape the subtle

$$\frac{c \in \{\mathtt{T}, \mathtt{F}, \mathtt{0}\}}{a \backslash\backslash c := c} \qquad \frac{a \backslash\backslash c := c'}{a \backslash\backslash \mathtt{S}\, c := \mathtt{S}\, c'} \qquad \frac{a \neq a'}{a \backslash\backslash a' := a'} \qquad \frac{}{a \backslash\backslash (e_1\,, e_2) := (\nu a.\, e_1\,, \nu a.\, e_2)}$$

$$\frac{}{a \backslash\backslash \lambda x : \tau \to e := \lambda x : \tau \to \nu a.\, e}$$

Fig. 4. $\lambda\nu$-PCF partial operation of name restriction

properties of generative names modulo contextual equivalence, but encounter them higher up the type hierarchy—see (27) in Example 6.5 below.

We adopt the usual definitions of PCF contextual preorder and contextual equivalence (observing the termination behaviour of expressions of higher type in contexts of ground type), arriving at the following definition for $\lambda\nu$-PCF:

Definition 6.4 (*contextual equivalence*). As usual, a $\lambda\nu$-PCF *context* $C[\_]$ is an expression with a single sub-expression replaced by the place-holder '_', and $C[e]$ is the expression that results from replacing _ by an expression e (possibly capturing free variables and atomic names in e). Given well-typed expressions $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$, we write $\Gamma \vdash e \leq_{\lambda\nu\text{-PCF}} e' : \tau$ and say that e and e' are in the *contextual preorder* if for all contexts $C[\_]$ for which $\emptyset \vdash C[e] : \mathtt{bool}$ and $\emptyset \vdash C[e'] : \mathtt{bool}$ hold, it is the case that $C[e] \Downarrow \mathtt{T}$ implies $C[e'] \Downarrow \mathtt{T}$. The equivalence relation generated by this preorder is written $\cong_{\lambda\nu\text{-PCF}}$ and is called *contextual equivalence*.

Example 6.5. Although $\lambda\nu$-PCF contains the ν-calculus [Pitts and Stark 1993] as a subset, the two languages have very different semantics for local names – Odersky-style for $\lambda\nu$-PCF versus generative for the ν-calculus. This affects properties of contextual equivalence in the two languages. For example, if $\Gamma, x : \tau \vdash e : \tau'$, then

$$\Gamma \vdash \nu a.\, \lambda x : \tau \to e \cong_{\lambda\nu\text{-PCF}} \lambda x : \tau \to \nu a.\, e : \tau \to \tau' \tag{16}$$

is valid. Indeed, this is a 'Kleene equivalence' for $\lambda\nu$-PCF (one expression evaluates to a canonical form if and only if the other does and in that case the canonical forms are equal); and all such identities can be proved by checking that they hold in the denotational model developed in the next section and then appealing to the computational adequacy result proved there (Theorem 7.5). However, (16) does not always hold in the ν-calculus [Pitts and Stark 1993, Example 2]. On the other hand, analogues of some ν-calculus equivalences are also true for $\lambda\nu$-PCF, once one takes into account the fact that, like PCF, $\lambda\nu$-PCF is call-by-name, whereas the ν-calculus is call-by-value. For example, here are call-by-name analogues of two equivalences in Pitts and Stark [1993, Example 4]:

$$\emptyset \vdash \nu a.\, \lambda x : \mathtt{name} \to (x = a) \cong_{\lambda\nu\text{-PCF}} \lambda x : \mathtt{name} \to \mathtt{if}\ x = x\ \mathtt{then}\ \mathtt{F}\ \mathtt{else}\ \mathtt{F} : \mathtt{name} \to \mathtt{bool} \tag{17}$$

$$\emptyset \vdash \nu a.\, \nu a'.\, \lambda(f : \mathtt{name} \to \mathtt{bool}) \to \mathtt{eq}\,(f\, a)\,(f\, a') \cong_{\lambda\nu\text{-PCF}} \lambda(f : \mathtt{name} \to \mathtt{bool}) \to \nu a.\ \mathtt{if}\ f\, a\ \mathtt{then}\ \mathtt{T}\ \mathtt{else}\ \mathtt{T} : (\mathtt{name} \to \mathtt{bool}) \to \mathtt{bool} \tag{18}$$

where $\mathtt{eq} : \mathtt{bool} \to \mathtt{bool} \to \mathtt{bool}$ is an abbreviation for a boolean-equality test, defined using conditionals. In contrast to the ν-calculus, where it takes some effort to prove equivalences like (17) and (18) (see Tzevelekos [2012]), for $\lambda\nu$-PCF these properties are easily seen to hold in the straightforward and computationally adequate denotational semantics that we describe in the next section; see Example 7.6. However, this is not always the case: there are valid instances of the $\lambda\nu$-PCF contextual preorder and

equivalence that cannot be established by calculating denotations in our model. Here are two examples of this phenomenon, suggested by Tzevelekos [private communication]. Consider the following variable-closed $\lambda\nu$-PCF expressions:

$$\mathtt{bot}_\tau \triangleq \mathtt{fix}\,(\lambda x : \tau \to x) \tag{19}$$

$$\mathtt{eqBot}_a \triangleq \lambda x : \mathtt{name} \to \mathtt{if}\; x = a \;\mathtt{then}\; \mathtt{T}\; \mathtt{else}\; \mathtt{bot}_{\mathtt{bool}} \tag{20}$$

$$\mathtt{kBot} \triangleq \lambda x : \mathtt{name} \to \mathtt{bot}_{\mathtt{bool}} \tag{21}$$

$$\mathtt{F}_1 \triangleq \lambda(f : (\mathtt{name} \to \mathtt{bool}) \to \mathtt{bool}) \to \nu a.\, f\,\mathtt{eqBot}_a \tag{22}$$

$$\mathtt{F}_2 \triangleq \lambda(f : (\mathtt{name} \to \mathtt{bool}) \to \mathtt{bool}) \to f\,\mathtt{kBot} \tag{23}$$

$$\mathtt{eq}_a \triangleq \lambda x : \mathtt{name} \to (x = a) \tag{24}$$

$$\mathtt{G}_1 \triangleq \lambda(g : (\mathtt{name} \to \mathtt{bool}) \to \mathtt{name}) \to \nu a.\, (g\,\mathtt{eq}_a = a) \tag{25}$$

$$\mathtt{G}_2 \triangleq \lambda(g : (\mathtt{name} \to \mathtt{bool}) \to \mathtt{name}) \to \mathtt{F}\,. \tag{26}$$

Thus $\mathtt{eqBot}_a$, $\mathtt{kBot}$ and $\mathtt{eq}_a$ are of type $\mathtt{name}\to\mathtt{bool}$; $\mathtt{F}_1$ and $\mathtt{F}_2$ are of type $((\mathtt{name}\to\mathtt{bool})\to \mathtt{bool})\to\mathtt{bool}$; and $\mathtt{G}_1$ and $\mathtt{G}_2$ are of type $((\mathtt{name}\to\mathtt{bool})\to\mathtt{name})\to\mathtt{bool}$. We claim that

$$\emptyset \vdash \mathtt{F}_1 \cong_{\lambda\nu\text{-}\mathsf{PCF}} \mathtt{F}_2 : ((\mathtt{name}\to\mathtt{bool})\to\mathtt{bool})\to\mathtt{bool} \tag{27}$$

$$\emptyset \vdash \mathtt{G}_1 \leq_{\lambda\nu\text{-}\mathsf{PCF}} \mathtt{G}_2 : ((\mathtt{name}\to\mathtt{bool})\to\mathtt{name})\to\mathtt{bool}\,. \tag{28}$$

The intuitive justification for (27) is that whatever argument of type $(\mathtt{name}\to\mathtt{bool})\to\mathtt{bool}$ is supplied for f by a context, it cannot have a free occurrence of a (because substitution for f in $\nu a.\,(f\,\mathtt{eq}_a)$ is capture-avoiding) and hence, it is claimed, cannot distinguish $\mathtt{eq}_a$ from $\mathtt{k}_\mathtt{F}$. Similarly for (28), whatever argument of type $(\mathtt{name}\to\mathtt{bool})\to\mathtt{name}$ is supplied for g by a context will not contain a free and hence, it is claimed, cannot produce a when applied to $\mathtt{eq}_a$. (Since that application may diverge, in (28) we only have $\leq_{\lambda\nu\text{-}\mathsf{PCF}}$ rather than $\cong_{\lambda\nu\text{-}\mathsf{PCF}}$.) Formal proofs of this intuition are sketched in Appendix A.

Once we add existential quantification over names (Example 5.7) to the programming language, one does gain the ability to distinguish $\mathtt{eq}_a$ from $\mathtt{k}_\mathtt{F}$ with a test not involving a free; and adding definite name description (Example 5.8) allows one to regain a from $\mathtt{eq}_a$ by applying an expression not containing a free. We use these examples to establish results about full abstraction in Section 8.

7. DENOTATIONAL SEMANTICS OF $\lambda\nu$-PCF

For each $\lambda\nu$-PCF type τ, we define a nominal Scott domain $[\![\tau]\!]$ by recursion on the structure of τ as follows:

— $[\![\mathtt{bool}]\!] = 2_\perp$, the flat domain (*cf.* Definition 5.2) on a discrete nominal set with two elements, $2 = \{\mathrm{true}, \mathrm{false}\}$.

— $[\![\mathtt{nat}]\!] = \mathbb{N}_\perp$, the flat domain on the discrete nominal set of natural numbers, $\mathbb{N} = \{0, 1, 2, \ldots\}$.

— $[\![\tau \times \tau']\!] = [\![\tau]\!] \times [\![\tau']\!]$, the product of nominal Scott domains.

— $[\![\tau \to \tau']\!] = [\![\tau]\!] \to [\![\tau']\!]$, the nominal Scott domain of uniform-continuous functions (the exponential in the cartesian closed category **Nsd**, Theorem 5.5).

— $[\![\mathtt{name}]\!] = \mathbb{A}_\perp$, the flat domain on the nominal set of atomic names, $\mathbb{A} = \{a, b, c, \ldots\}$.

Typing environments are interpreted as finite cartesian products:

$$[\![\{x_1 : \tau_1, \ldots, x_n : \tau_n\}]\!] = [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!]\,.$$

Finite tuples $\rho \in [\![\Gamma]\!]$ can be interpreted as partial functions from variables to domains such that $\mathrm{dom}(\rho) = \mathrm{dom}(\Gamma)$ and $\rho(x) \in [\![\Gamma(x)]\!]$ for all $x \in \mathrm{dom}(\Gamma)$. We call such partial

$$[\![x]\!]\rho = \rho\, x \qquad [\![\mathtt{T}]\!]\rho = \text{true} \qquad [\![\mathtt{F}]\!]\rho = \text{false} \qquad [\![\mathtt{0}]\!]\rho = 0$$

$$[\![\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3]\!]\rho = \begin{cases} [\![e_2]\!]\rho & \text{if } [\![e_1]\!]\rho = \text{true} \\ [\![e_3]\!]\rho & \text{if } [\![e_1]\!]\rho = \text{false} \\ \bot & \text{otherwise} \end{cases} \qquad [\![\mathtt{S}\, e]\!]\rho = \begin{cases} n+1 & \text{if } [\![e]\!]\rho = n \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\mathtt{pred}\, e]\!]\rho = \begin{cases} n & \text{if } [\![e]\!]\rho = n+1 \\ \bot & \text{otherwise} \end{cases} \qquad [\![\mathtt{zero}\, e]\!]\rho = \begin{cases} \text{true} & \text{if } [\![e]\!]\rho = 0 \\ \text{false} & \text{if } [\![e]\!]\rho = n+1 \\ \bot & \text{otherwise} \end{cases}$$

$$[\![(e_1\, , e_2)]\!]\rho = ([\![e_1]\!]\rho, [\![e_2]\!]\rho) \qquad [\![\mathtt{fst}\, e]\!]\rho = \pi_1\,([\![e]\!]\rho) \qquad [\![\mathtt{snd}\, e]\!]\rho = \pi_2\,([\![e]\!]\rho)$$

$$[\![\lambda x : \tau \to e]\!]\rho = \lambda d \in [\![\tau]\!] \to [\![e]\!]\rho[x \mapsto d] \qquad [\![e_1\, e_2]\!]\rho = [\![e_1]\!]\rho\,([\![e_2]\!]\rho) \qquad [\![\mathtt{fix}\, e]\!]\rho = \mathrm{fix}\,([\![e]\!]\rho)$$

. . . . . .

$$[\![a]\!]\rho = a \qquad [\![\nu a.\, e]\!]\rho = (a\backslash[\![e]\!])\rho \qquad [\![e_1 \mathtt{=} e_2]\!]\rho = \begin{cases} \text{true} & \text{if } [\![e_i]\!]\rho = a_i \text{ and } a_1 = a_2 \\ \text{false} & \text{if } [\![e_i]\!]\rho = a_i \text{ and } a_1 \neq a_2 \\ \bot & \text{otherwise} \end{cases}$$

$$[\![(e_1 \rightleftharpoons e_2)\, e_3]\!]\rho = \begin{cases} (a_1\ a_2) \cdot ([\![e_3]\!]\rho) & \text{if } [\![e_i]\!]\rho = a_i \\ \bot & \text{otherwise} \end{cases}$$

Fig. 5. $\lambda\nu$-PCF denotational semantics

functions Γ-*valuations*. If $\rho \in [\![\Gamma]\!]$, $x \notin \mathrm{dom}(\Gamma)$ and $d \in [\![\tau]\!]$, then we write $\rho[x \mapsto d]$ for the $(\Gamma, x : \tau)$-valuation that maps x to d and otherwise acts like ρ.

The denotation of each well-typed expression $\Gamma \vdash e : \tau$ is an element

$$[\![e]\!] \in [\![\Gamma]\!] \to [\![\tau]\!]$$

of the exponential domain $[\![\Gamma]\!] \to [\![\tau]\!]$; in other words, $[\![e]\!]$ is a uniform-continuous function mapping Γ-valuations $\rho \in [\![\Gamma]\!]$ to elements $[\![e]\!]\rho$ of the nominal Scott domain $[\![\tau]\!]$. Figure 5 gives the definition of these functions, by recursion over the structure[4] of e. The denotations of constructs from PCF are analogous to the standard denotational semantics of PCF in Scott domains. The functions π_1 and π_2 in the clauses for $\mathtt{fst}\, e$ and $\mathtt{snd}\, e$ are the first and second projection functions for pairs; and fix in the clause for $\mathtt{fix}\, e$ is the least fixed point function (14). The clauses below the dotted line in Figure 5 are for the syntactic constructs that $\lambda\nu$-PCF adds to PCF. Atomic names are their own denotation. The semantics of explicit swapping expressions, $(e_1 \rightleftharpoons e_2)\, e_3$, make use of the action of finite permutations that each nominal Scott domain possesses. Finally, to interpret expressions with locally scoped names, $\nu a.\, e$, we use uniform-continuous functions

$$a\backslash(\_) \in ([\![\Gamma]\!] \to [\![\tau]\!]) \to ([\![\Gamma]\!] \to [\![\tau]\!])$$

that are name restriction operations [Pitts 2013, Definition 9.2]:

Definition 7.1 (*uniform-continuous name restriction*). A *uniform-continuous name restriction operation* on a nominal Scott domain D is a morphism $(\_)\backslash(\_) : \mathbb{A}_\bot \times D \to D$

[4]Strictly speaking, it is by α-structural recursion [Pitts 2006], since we identify expressions up to α-equivalence of bound identifiers.

in **Nsd** satisfying for all $a, a' \in \mathbb{A}$ and $d \in D$

$$\bot\backslash d = \bot \tag{29}$$
$$a \# (a\backslash d) \tag{30}$$
$$a \# d \;\Rightarrow\; a\backslash d = d \tag{31}$$
$$a\backslash(a'\backslash d) = a'\backslash(a\backslash d) \tag{32}$$

(where as usual $\_ \# \_$ is the freshness relation (5) for D). Since $(\_)\backslash(\_)$ is equivariant, (30) is equivalent to asking that $a\backslash(\_)$ be a binding operation, that is, respect the generalized version of α-equivalence that any nominal set supports: $a' \# (a, d) \Rightarrow a\backslash d = a'\backslash((a\ a') \cdot d)$. (31) and (32) are basic structural properties that one expects any notion of local scoping to have; see the discussion at the start of Section 9.1 of Pitts [2013].

THEOREM 7.2. *Every flat nominal Scott domain $X_\bot$ has a uniform-continuous name restriction operation satisfying*

$$a\backslash d = \begin{cases} d & \textit{if } a \# d \\ \bot & \textit{if } a \in \operatorname{supp} d \end{cases} \tag{33}$$

*for all $a \in \mathbb{A}$ and $d \in X_\bot$. If $D_1, D_2 \in$ **Nsd** have uniform-continuous name restriction operations, their product $D_1 \times D_2$ has one satisfying*

$$a\backslash(d_1, d_2) = (a\backslash d_1, a\backslash d_2) \tag{34}$$

*for all $a \in \mathbb{A}$, $d_1 \in D_1$ and $d_2 \in D_2$. If $D_1, D_2 \in$ **Nsd** and D_2 has a uniform-continuous name restriction operation, then whether or not D_1 has one as well, the exponential $D_1 \to D_2$ has such an operation, satisfying*

$$(a\backslash f)\, d = a\backslash(f\, d) \quad \textit{if } a \# d \tag{35}$$

for all $d \in D_1$ and $f \in D_1 \to D_2$.

PROOF. It is easy to see that (33) and (34) determine uniform-continuous name restriction operations on flat and product domains respectively. The case for exponential domains is less straightforward. (35) at first seems like only a partial specification for the function $a\backslash f : D_1 \to D_2$, but in fact determines it uniquely, since it implies that for all $d \in D_1$, $f \in D_1 \to D_2$ and $a \in \mathbb{A}$

$$(a\backslash f)\, d = a'\backslash(((a\ a') \cdot f)\, d) \quad \text{for some, or indeed any } a' \# (f, d). \tag{36}$$

This and (29)–(32) follow from the results for nominal sets in Pitts [2013, Section 9.3]. One also has to check that each function $a\backslash f$ is uniform-continuous and that $f \mapsto a\backslash f$ is uniform-continuous, but these are routine calculations. □

Using this theorem we get a uniform-continuous name restriction operation on the denotation $[\![\tau]\!]$ of each $\lambda\nu$-PCF type τ, by recursion on the structure of τ; and then we use the theorem again to lift the name restriction operation on $[\![\tau]\!]$ to one on the exponential domain $[\![\Gamma]\!] \to [\![\tau]\!]$, for each typing environment Γ. It is this operation which is used in Figure 5 to give the denotation of locally scoped expressions $\nu a.\, e$. From (35) we have

$$[\![\nu a.\, e]\!]\rho = a\backslash([\![e]\!]\rho) \qquad \text{if } a \# \rho \tag{37}$$

where the name restriction operation on the right-hand side is that for $[\![\tau]\!]$. Note that the side condition $a \# \rho$ is always satisfiable, since we identify expressions up to α-equivalence of ν-bound atomic names.

Notation 7.3. For the empty typing environment $\emptyset$, $[\![\emptyset]\!] = 1$ contains a unique $\emptyset$-valuation, ρ_0. Given a variable-closed expression $\emptyset \vdash e : \tau$, we simply write $[\![e]\!]$ for $[\![e]\!]\rho_0$.

LEMMA 7.4 ($\lambda\nu$-PCF SOUNDNESS). *If e is a well-typed, variable-closed $\lambda\nu$-PCF expression and $e \Downarrow c$, then $[\![e]\!] = [\![c]\!]$.*

PROOF. The proof is by induction on the derivation of $e \Downarrow c$ from the rules in Figure 3 and most of the induction steps are routine. As for PCF, to establish closure under the rule for evaluating function applications we need the following substitution property, which can be proved by induction on the structure of expressions:

$$\begin{array}{l}\textit{Substitution Lemma.}\ \text{If } \Gamma \vdash e : \tau \text{ and } \Gamma, x : \tau \vdash e' : \tau', \text{ then } \Gamma \vdash e'[e/x] : \tau' \text{ holds}\\ \text{and for all } \rho \in [\![\Gamma]\!] \text{ we have } [\![e'[e/x]]\!]\rho = [\![e']\!]\rho[x \mapsto [\![e]\!]\rho].\end{array} \tag{38}$$

For closure under the rule for evaluating explicit swapping expressions, we use the fact that the denotational semantics is equivariant:

$$\begin{array}{l}\textit{Equivariance Lemma.}\ \text{For all } \Gamma \vdash e : \tau \text{ and all finite permutations } \pi \text{ it holds}\\ \text{that } \pi \cdot [\![e]\!] = [\![\pi \cdot e]\!].\end{array} \tag{39}$$

Thus we also have $(a_1\ a_2) \cdot [\![c]\!] = [\![(a_1\ a_2) \cdot c]\!]$ for all variable-closed canonical forms c. Finally, for closure under the rule for evaluating locally scoped expressions, we need the following property of the name restriction operations $a\backslash(\_) \in [\![\tau]\!] \to [\![\tau]\!]$:

$$\begin{array}{l}\textit{Restriction Lemma.}\ \text{If } c \text{ and } c' \text{ are variable-closed canonical forms of type } \tau\\ \text{satisfying } a \backslash\backslash c := c' \text{ (see Figure 4), then } a\backslash[\![c]\!] = [\![c']\!] \text{ in } [\![\tau]\!].\end{array} \tag{40}$$

This follows from the definition in Figure 4, by induction on the structure of c; for λ-abstractions one needs the fact, noted in the proof of Theorem 7.2, that (35) uniquely determines the name restriction operation on functions defined by (36). □

The following result allows one to establish $\lambda\nu$-PCF contextual equivalences by proving equality of denotations.

THEOREM 7.5 ($\lambda\nu$-PCF COMPUTATIONAL ADEQUACY). *For any $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$, if $[\![e]\!] \sqsubseteq [\![e']\!] \in [\![\Gamma]\!] \to [\![\tau]\!]$, then $\Gamma \vdash e \leq_{\lambda\nu\text{-PCF}} e' : \tau$. Consequently, if $[\![e]\!] = [\![e']\!]$, then $\Gamma \vdash e \cong_{\lambda\nu\text{-PCF}} e' : \tau$.*

PROOF. It is evident from Figure 5 that the denotational semantics is compositional, in the sense that for any suitably typed context $C[\_]$ one has

$$[\![e]\!] \sqsubseteq [\![e']\!] \Rightarrow [\![C[e]]\!] \sqsubseteq [\![C[e']]\!]\,.$$

So in view of Lemma 7.4 it suffices to show that if e is a variable-closed expression of type `bool`, then

$$[\![e]\!] = \mathrm{true} \Rightarrow e \Downarrow \mathtt{T}. \tag{41}$$

As for PCF, this can be proved by defining a suitable logical relation

$$d \lhd_\tau e \quad (d \in [\![\tau]\!],\ \emptyset \vdash e : \tau) \tag{42}$$

between the semantics and the syntax of $\lambda\nu$-PCF. See Streicher [2006, Chapter 4] for a good exposition of this method of proving computational adequacy for the Scott domain model of PCF. We outline the method, concentrating on how the move to nominal Scott domains allows us to deal with the new features of $\lambda\nu$-PCF.

The definition of (42) is by recursion on the structure of the type τ:

$$\begin{aligned} d \lhd_\gamma e &\triangleq d = \bot \vee (\exists c)\, e \Downarrow c \wedge [\![c]\!] = d \qquad \text{for } \gamma \in \{\mathtt{bool}, \mathtt{nat}, \mathtt{name}\} \\ (d_1, d_2) \lhd_{\tau_1 \times \tau_2} e &\triangleq d_1 \lhd_{\tau_1} \mathtt{fst}\, e \wedge d_2 \lhd_{\tau_2} \mathtt{snd}\, e \\ d \lhd_{\tau_1 \to \tau_2} e &\triangleq (\forall d_1, e_1)\, d_1 \lhd_{\tau_1} e_1 \Rightarrow d\, d_1 \lhd_{\tau_2} e\, e_1\,. \end{aligned}$$

The relation is extended to typing environments Γ by:

$$\rho \lhd_\Gamma s \triangleq (\forall x \in \mathrm{dom}(\Gamma))\, \rho\, x \lhd_{\Gamma x} s\, x\,. \tag{43}$$

Here $\rho \in [\![\Gamma]\!]$ and s ranges over Γ-*substitutions*, which by definition are functions mapping each variable $x \in \mathrm{dom}(\Gamma)$ to a variable-closed expression $s\, x$ of type $\Gamma\, x$. This extension allows us to state the

> *Fundamental Property of the Logical Relation.* For every $\Gamma \vdash e : \tau$, $\rho \in [\![\Gamma]\!]$ and Γ-substitution s
>
> $$\rho \lhd_\Gamma s \;\Rightarrow\; [\![e]\!]\rho \lhd_\tau e\, s \tag{44}$$
>
> where $e\, s$ is the variable-closed expression of type τ obtained by simultaneous substitution; thus $e\, s \triangleq e[e_1/x_1, \ldots, e_n/x_n]$ if $\mathrm{dom}(\Gamma)$ consists of the distinct variables $x_1, \ldots, x_n$ and $s\, x_i = e_i$ for $i = 1, \ldots, n$.

The fundamental property (44) is proved by induction on the derivation of $\Gamma \vdash e : \tau$ from the rules in Figure 2. As for PCF, the proof makes use of the 'Kleene preorder' defined by

$$e \leq^k e' \triangleq (\forall c)\, e \Downarrow c \Rightarrow e' \Downarrow c \tag{45}$$

together with the fact, established by induction on the structure of τ, that the logical relation is closed under this preorder:

$$d \lhd_\tau e \;\wedge\; e \leq^k e' \Rightarrow d \lhd_\tau e'. \tag{46}$$

The induction steps in the proof of (44) for PCF constructs are standard, except that when it comes to fixed point recursion one uses the facts (easily verified by induction on τ) that each $\_ \lhd_\tau e$ satisfies $\bot \lhd_\tau e$ and $((\forall d \in S)\; d \lhd_\tau e) \Rightarrow \bigsqcup S \lhd_\tau e$ for all uniform-directed subsets $S \subseteq [\![\tau]\!]$. The induction step for name swapping $(e_1 \rightleftharpoons e_2)\, e_3$ depends on the equivariance of the operational semantics (15), of the denotational semantics (39), and also of the logical relation

$$d \lhd_\tau e \;\Rightarrow\; \pi \cdot d \lhd_\tau \pi \cdot e \tag{47}$$

which can be proved by induction on τ. The induction step for locally scoped names $\nu a.\, e$ follows directly from the fact that the logical relation is closed under name restriction:

$$d \lhd_\tau e \;\Rightarrow\; a \backslash d \lhd_\tau \nu a.\, e\,. \tag{48}$$

This in turn can be shown by induction on τ, where we use some easily checked facts about the Kleene preorder:

$$\begin{gathered} \nu a.\, \mathtt{fst}\, e \leq^k \mathtt{fst}\, \nu a.\, e \\ \nu a.\, \mathtt{snd}\, e \leq^k \mathtt{snd}\, \nu a.\, e \\ a \# e_1 \Rightarrow \nu a.\, (e\, e_1) \leq^k (\nu a.\, e)\, e_1 \end{gathered}$$

together with (46) and the fact that $\nu a.\, \_$ is a binder.

As a corollary of the fundamental property (44) we have that every variable-closed expression e of type $\mathtt{bool}$ satisfies $[\![e]\!] \lhd_{\mathtt{bool}} e$. Consequently, if $[\![e]\!] = \mathrm{true}$, then by definition of $\lhd_{\mathtt{bool}}$ we must have $e \Downarrow c$ for some c with $[\![c]\!] = \mathrm{true}$, that is, $e \Downarrow \mathtt{T}$. So we do indeed have (41) and this completes the proof of the theorem. □

$$
\begin{array}{ll}
e \in \mathrm{Exp} ::= & \textit{expressions} \\
\vdots & \text{(as for } \lambda\nu\text{-PCF)} \\
\mathtt{ex}\, x.\, e & \text{existential name quantification} \\
\mathtt{the}\, x.\, e & \text{definite name description}
\end{array}
$$

$$
\frac{\Gamma, x : \mathtt{name} \vdash e : \mathtt{bool}}{\Gamma \vdash \mathtt{ex}\, x.\, e : \mathtt{bool}} \qquad \frac{\Gamma, x : \mathtt{name} \vdash e : \mathtt{bool}}{\Gamma \vdash \mathtt{the}\, x.\, e : \mathtt{name}}
$$

$$
\frac{a \in \mathbb{A} \qquad e[a/x] \Downarrow \mathtt{T}}{\mathtt{ex}\, x.\, e \Downarrow \mathtt{T}} \qquad \frac{b \# e \qquad (\forall a' \in \mathrm{fn}(e) \cup \{b\})\ e[a'/x] \Downarrow \mathtt{F}}{\mathtt{ex}\, x.\, e \Downarrow \mathtt{F}}
$$

$$
\frac{e[a/x] \Downarrow \mathtt{T} \qquad b \# (e, a) \qquad (\forall a' \in (\mathrm{fn}(e) - \{a\}) \cup \{b\})\ e[a'/x] \Downarrow \mathtt{F}}{\mathtt{the}\, x.\, e \Downarrow a}
$$

$$
[\![\mathtt{ex}\, x.\, e]\!]\rho = \mathit{exists}_{\mathbb{A}}([\![\lambda x : \mathtt{name} \to e]\!]\rho) \qquad [\![\mathtt{the}\, x.\, e]\!]\rho = \mathit{the}_{\mathbb{A}}([\![\lambda x : \mathtt{name} \to e]\!]\rho)
$$

Fig. 6. Syntax and semantics of $\lambda\nu$-PCF$^+$

Example 7.6. Using Theorem 7.5 one can prove many contextual equivalences in $\lambda\nu$-PCF, such as (16)–(18), in a straightforward manner via the denotational semantics. We prove (17) in detail. Since we identify expressions up to α-equivalence, for any given $a' \in \mathbb{A}$ we can pick a representative expression $\nu a.\, \lambda x : \mathtt{name} \to (x = a)$ such that $a \neq a'$, then

$$
\begin{array}{ll}
[\![\nu a.\, \lambda x : \mathtt{name} \to (x = a)]\!]\, a' & \\
= (a \backslash [\![\lambda x : \mathtt{name} \to (x = a)]\!])\, a' & \text{using the definition in Figure 5} \\
= a \backslash ([\![\lambda x : \mathtt{name} \to (x = a)]\!]\, a') & \text{by (35), since } a \neq a' \\
= a \backslash \mathrm{false} & \text{as } a \neq a' \\
= \mathrm{false} & \text{by (33)} \\
= [\![\lambda x : \mathtt{name} \to \mathtt{if}\ x = x\ \mathtt{then}\ \mathtt{F}\ \mathtt{else}\ \mathtt{F}]\!]\, a'. &
\end{array}
$$

Similarly $[\![\nu a.\, \lambda x : \mathtt{name} \to (x = a)]\!]\, \bot = \bot = [\![\lambda x : \mathtt{name} \to \mathtt{if}\ x = x\ \mathtt{then}\ \mathtt{F}\ \mathtt{else}\ \mathtt{F}]\!]\, \bot$. Hence $[\![\nu a.\, \lambda x : \mathtt{name} \to (x = a)]\!] = [\![\lambda x : \mathtt{name} \to \mathtt{if}\ x = x\ \mathtt{then}\ \mathtt{F}\ \mathtt{else}\ \mathtt{F}]\!]$ and so (17) holds by Theorem 7.5.

To prove (18) one can combine the definition in Figure 5 with the fact that if $a, a' \# f \in (\mathbb{A}_\bot \to 2_\bot) \to 2_\bot$, then $f\, a = f((a\ a') \cdot a') = (a\ a') \cdot (f\, a') = f\, a'$ (since $f\, a' \in 2_\bot$).

8. FULL ABSTRACTION

Plotkin [1977] famously proves that the Scott domain model of PCF is computationally adequate, but not fully abstract: equality of denotations implies, but is not implied by, PCF contextual equivalence. Furthermore, he shows that the Scott model becomes fully abstract once one extends PCF with a parallel-or construct.

A similar story unfolds for $\lambda\nu$-PCF. We prove in Theorem 8.5 that the nominal Scott domain model becomes fully abstract once we add the $\mathtt{ex}\, x.\, e$ and $\mathtt{the}\, x.\, e$ constructs from Figure 6 to $\lambda\nu$-PCF$^+$. If we leave one of these constructs out, then full abstraction fails, as Theorem 8.4 shows.

Definition 8.1. The language $\lambda\nu$-PCF$^+$ is obtained by extending $\lambda\nu$-PCF with expressions for existentially quantifying over $\mathtt{name}$ ('there exists some $x : \mathtt{name}$ such

that...') and for forming definite descriptions over $\mathtt{name}$ ('the unique $x : \mathtt{name}$ such that...'). The syntax, typing, evaluation rules and denotational semantics for this extension are given in Figure 6, where the functions $exists_{\mathbb{A}} \in \mathrm{K}((\mathbb{A}_\perp \to 2_\perp) \to 2_\perp)$ and $the_{\mathbb{A}} \in \mathrm{K}((\mathbb{A}_\perp \to 2_\perp) \to \mathbb{A}_\perp)$ are defined in Examples 5.7 and 5.8. Both $\mathtt{ex}\, x.\, e$ and $\mathtt{the}\, x.\, e$ bind the variable x in e. Contextual equivalence for the extended language $\Gamma \vdash e \cong_{\lambda\nu\text{-}\mathsf{PCF}^+} e' : \tau$ is defined in the same way as it is for $\lambda\nu$-PCF in Definition 6.4.

Similarly we define $\lambda\nu$-PCF+ex to be $\lambda\nu$-PCF with only existential quantification added, and define $\lambda\nu$-PCF+the to be $\lambda\nu$-PCF with only definite description added.

Remark 8.2. The addition of existential quantification and definite description is mainly motivated by the need for them in our proof of full abstraction (Theorem 8.5). A less strict form of existential quantification for numbers (rather than, as here, for names) occurs in Plotkin's original PCF paper [Plotkin 1977]. Definite description has a long history in logic, but it is harder to motivate from a programming language perspective. Note that the analogue for numbers of our definite description and existential quantification functionals are not computable, as indicated already in Remark 5.9. The computability of $\mathtt{ex}\, x.\, e$ and $\mathtt{the}\, x.\, e$ provide an example of the phenomenon of 'finite modulo symmetry' mentioned in the introduction. For example, to prove $\mathtt{ex}\, x.\, e \Downarrow \mathtt{F}$, we just have to show $e[a'/x] \Downarrow \mathtt{F}$ for each of the finitely many atomic names a' that occur free in e and then pick any one of the infinitely many atomic names b that do not occur free in e and show $e[b/x] \Downarrow \mathtt{F}$; equivariance of evaluation (15) ensures that if $e[b/x] \Downarrow \mathtt{F}$, then $e[b'/x] \Downarrow \mathtt{F}$ holds for any other b' not occurring free in e.

Remark 8.3 (*parallel-or*). The parallel-or construct is central in Plotkin's classical work on PCF [Plotkin 1977, Table 1], because PCF fails to be fully abstract without it, but becomes fully abstract with it. It is a boolean-valued operation that evaluates two boolean expressions 'in parallel' and is true whenever one of the expressions evaluates to true, no matter if the other one diverges. In the presence of $\mathtt{ex}\, x.\, e$ we can define parallel-or as syntactic sugar by

$$e\ \mathtt{por}\ e' \triangleq \nu a.\, \nu a'.\, \mathtt{ex}\, x.\ \mathtt{if}\ x = a\ \mathtt{then}\ e\ \mathtt{else}\ \mathtt{if}\ x = a'\ \mathtt{then}\ e'\ \mathtt{else}\ \mathtt{F}$$

where a, a' and x do not occur free in e or e'.

THEOREM 8.4 (FAILURE OF FULL ABSTRACTION). *The denotational semantics of neither $\lambda\nu$-PCF+ex, nor $\lambda\nu$-PCF+the is fully abstract, in the sense that there are expressions in the contextual preorder relations of those languages which are not in the denotational partial order.*

PROOF. We start with $\lambda\nu$-PCF+the. Consider the two expressions $\mathtt{F}_1$ and $\mathtt{F}_2$ defined in (22) and (23). Their denotations are (equivariant and uniform-continuous) functions from $(\mathbb{A}_\perp \to 2_\perp) \to 2_\perp$ to $2_\perp$. Applying $exists_{\mathbb{A}}$ from Example 5.7 to them, we get

$$[\![\mathtt{F}_1]\!](exists_{\mathbb{A}}) = a\backslash(exists_{\mathbb{A}}\, [\![\mathtt{eqBot}_a]\!]) = a\backslash\mathrm{true} = \mathrm{true} \not\sqsubseteq \perp = exists_{\mathbb{A}}\, [\![\mathtt{kBot}]\!] = [\![\mathtt{F}_2]\!](exists_{\mathbb{A}})$$

where $\mathtt{eqBot}_a$ and $\mathtt{kBot}$ are defined in (20) and (21). This shows that $[\![\mathtt{F}_1]\!] \not\sqsubseteq [\![\mathtt{F}_2]\!]$. Nevertheless, in Appendix A we sketch the proof that

$$\emptyset \vdash \mathtt{F}_1 \leq_{\lambda\nu\text{-}\mathsf{PCF}+\mathtt{the}} \mathtt{F}_2 : ((\mathtt{name} \to \mathtt{bool}) \to \mathtt{bool}) \to \mathtt{bool} \tag{49}$$

which shows that $\mathtt{F}_1$ and $\mathtt{F}_2$ are counter examples to full abstraction for $\lambda\nu$-PCF+the. We explained informally why (49) holds for $\lambda\nu$-PCF in Example 6.5; and adding $\mathtt{the}\, x.\, e$ does not add any ability to distinguish the evaluation behaviour of the two expressions in any boolean context. The formal proof of (49) in Appendix A works via an extensionality result (Proposition A.3) that is of interest in its own right.

For $\lambda\nu$-PCF+ex, recall the expressions $\mathtt{G}_1$ and $\mathtt{G}_2$ from (25) and (26). They are in the contextual preorder for $\lambda\nu$-PCF+ex

$$\emptyset \vdash \mathtt{G}_1 \leq_{\lambda\nu\text{-}\mathsf{PCF}+\mathtt{ex}} \mathtt{G}_2 : ((\mathtt{name} \to \mathtt{bool}) \to \mathtt{name}) \to \mathtt{bool}\,. \tag{50}$$

and the proof sketch of this is in Appendix A. This provides a counter example to full abstraction for $\lambda\nu$-PCF+ex, because through applying $the_{\mathbb{A}}$ from Example 5.8

$$[\![\mathtt{G}_1]\!](the_{\mathbb{A}}) = a\backslash(eq_a(the_{\mathbb{A}}\,[\![\mathtt{eq}_a]\!])) = a\backslash(eq_a\,a) = a\backslash\mathrm{true} = \mathrm{true} \not\sqsubseteq \mathrm{false} = [\![\mathtt{G}_2]\!](the_{\mathbb{A}})$$

(using the function eq_a from (10) and the expression $\mathtt{eq}_a$ from (24)), we obtain that $[\![\mathtt{G}_1]\!] \not\sqsubseteq [\![\mathtt{G}_2]\!]$. □

More positively, we now show that $\lambda\nu$-PCF$^+$ is fully abstract with respect to the nominal Scott domain model.

THEOREM 8.5 (FULL ABSTRACTION FOR $\lambda\nu$-PCF$^+$). *For all well-typed expressions $\Gamma \vdash e : \tau$ and $\Gamma \vdash e' : \tau$ in $\lambda\nu$-PCF$^+$, we have*

$$[\![e]\!] \sqsubseteq [\![e']\!] \in [\![\Gamma]\!] \to [\![\tau]\!] \Leftrightarrow \Gamma \vdash e \leq_{\lambda\nu\text{-}\mathsf{PCF}^+} e' : \tau. \tag{FA_τ}$$

and consequently $[\![e]\!] = [\![e']\!] \in [\![\Gamma]\!] \to [\![\tau]\!] \Leftrightarrow \Gamma \vdash e \cong_{\lambda\nu\text{-}\mathsf{PCF}^+} e' : \tau$ holds.

The outline of the proof of this theorem occupies the rest of this section, with some technical details relegated to the appendices. First note that the extension of Theorem 7.5 to $\lambda\nu$-PCF$^+$ is straightforward and gives us the left-to-right implication in (FA_τ). Establishing the reverse implication leads to an investigation of the uniform-compact elements (Definition 3.6) of nominal Scott domains of the form $[\![\tau]\!]$ for some type $\tau \in \mathrm{Typ}$.

Definition 8.6 (*definability*). An element in the nominal Scott domain model $d \in [\![\tau]\!]$ is $\lambda\nu$-PCF$^+$*-definable* if there is a variable-closed $\lambda\nu$-PCF$^+$ expression $\emptyset \vdash e : \tau$ that denotes it: $d = [\![e]\!]$. We say uniform-compact definability holds at a type τ if all uniform-compact elements of $[\![\tau]\!]$ are definable:

$$(\forall u \in \mathrm{K}[\![\tau]\!])\,(\exists e)\ \emptyset \vdash e : \tau \ \wedge\ u = [\![e]\!]. \tag{DEF_τ}$$

As Curien [2007] surveys, knowing (DEF_τ) for all types τ is enough to show the right-to-left implication in (FA_τ). Our proof follows this proof pattern. However, we only show uniform-compact definability at certain types that avoid the use of function types $\tau_1 \to \tau_2$ in which the nominal Scott domain $[\![\tau_2]\!]$ might contain elements with non-empty support. So $\tau_2 = \mathtt{nat}$ is OK, but $\tau_2 = \mathtt{name}$ is not, for example. This leads us to make the following definition:

Definition 8.7 (*simple types*). Let $\mathrm{Styp} \subseteq \mathrm{Typ}$ be the subset of types (Figure 1) given by the following grammar

$$\sigma \in \mathrm{Styp} ::= \mathtt{nat} \mid \mathtt{name} \mid \sigma \times \sigma \mid \sigma \to \mathtt{nat}$$

and call types of that grammar *simple types*.

Note that $\sigma \to \mathtt{name}$ is not in Styp. The following lemma is the key to the usefulness of simple types. It is where the presence of '$\mathtt{the}\,x.\,e$' expressions in $\lambda\nu$-PCF$^+$ gets used.

LEMMA 8.8. *For any type $\tau \in \mathrm{Typ}$, there is a simple type $\sigma \in \mathrm{Styp}$ such that there is a $\lambda\nu$-PCF$^+$-definable embedding-projection pair from τ to σ, meaning that there are closed $\lambda\nu$-PCF$^+$ expressions $\emptyset \vdash e : \tau \to \sigma$ and $\emptyset \vdash p : \sigma \to \tau$ with $[\![\lambda x : \tau \to p\,(e\,x)]\!] = id_{[\![\tau]\!]}$ and $[\![\lambda x : \sigma \to e\,(p\,x)]\!] \sqsubseteq id_{[\![\sigma]\!]}$.*

PROOF. It is standard that identity functions are embedding-projections, that embedding-projections compose, and that products and functions give covariant functors with respect to embedding-projections; and all these constructs yield $\lambda\nu$-PCF$^+$-definable embedding-projections when fed them. Furthermore, using '$\mathtt{the}\, x.\, e$' expressions, we get a $\lambda\nu$-PCF$^+$-definable embedding-projection pair from $\mathtt{name}$ to $\mathtt{name} \to \mathtt{nat}$, via

$$e \triangleq \lambda x : \mathtt{name} \to \lambda y : \mathtt{name} \to \mathtt{if}\ x = y\ \mathtt{then}\ \mathtt{0}\ \mathtt{else}\ \mathtt{S}\,\mathtt{0}$$

$$p \triangleq \lambda(f : \mathtt{name} \to \mathtt{nat}) \to \mathtt{the}\, x.\, (\mathtt{if}\ \mathtt{zero}\,(f\, x)\ \mathtt{then}\ \mathtt{T}\ \mathtt{else}\ \mathtt{if}\ \mathtt{zero}\,(\mathtt{pred}\,(f\, x))\ \mathtt{then}\ \mathtt{F}\ \mathtt{else}\ \mathtt{fix}\,(\lambda x : \mathtt{bool} \to x)).$$

It is also not hard to see that there is a $\lambda\nu$-PCF$^+$-definable embedding-projection pair from $\mathtt{bool}$ to $\mathtt{nat}$.

Using these facts, it follows by induction on the structure of $\sigma_2 \in \mathrm{Styp}$ that for all $\sigma_1 \in \mathrm{Styp}$, there is a $\lambda\nu$-PCF$^+$-definable embedding-projection pair from $\sigma_1 \to \sigma_2$ to some $\sigma \in \mathrm{Styp}$. (For example, when $\sigma_2 = \mathtt{name}$, then we can take $\sigma = \sigma_1 \times \mathtt{name} \to \mathtt{nat}$.) Using this, one can proceed by induction on the structure of $\tau \in \mathrm{Typ}$ to show that there is a $\lambda\nu$-PCF$^+$-definable embedding-projection pair from τ to some simple type. □

LEMMA 8.9. *If there is a $\lambda\nu$-PCF$^+$-definable embedding-projection pair from τ to σ, then uniform-compact definability at σ implies the same at τ, i.e.* $(\mathrm{DEF}_\sigma) \Rightarrow (\mathrm{DEF}_\tau)$.

PROOF. Let e and p be as in Lemma 8.8 and suppose we are given $u \in \mathrm{K}[\![\tau]\!]$. By Abramsky and Jung [1994, Proposition 3.1.16] we get that $[\![e]\!]u \in \mathrm{K}[\![\sigma]\!]$ and by (DEF_σ) we know that there is a $\emptyset \vdash e' : \sigma$ such that $[\![e']\!] = [\![e]\!]u$. It follows that $p\, e'$ defines u, since $[\![p\, e']\!] = [\![p]\!]\,[\![e']\!] = [\![p]\!]([\![e]\!]u) = u$. □

Combining Lemma 8.8 and Lemma 8.9 with the argument in the survey by Curien [2007, Criterion 2.2], we know that to prove full abstraction for $\lambda\nu$-PCF$^+$ (Theorem 8.5) it suffices to show that (DEF_σ) holds for all simple types $\sigma \in \mathrm{Styp}$.

The proof of (DEF_σ) in principle follows the structure of the traditional proof of uniform-compact definability by Plotkin [1977, Lemma 4.5]. A modern account of this proof can be found in Streicher [2006, Theorem 13.9]. However, in our nominal setting many uses of finite subsets in the traditional proof are replaced by uses of orbit-finite subsets and their presentation as hulls (Theorem 4.5). The definition of $\mathrm{hull}_A F$ involves existential quantification over finite permutations of $\mathbb{A}$, and for the definability proof we need to reduce this to existential quantification over elements of $\mathbb{A}$. This is where the presence of $\mathtt{ex}\, x.\, e$ expressions in $\lambda\nu$-PCF$^+$ gets used, in order to prove the following two crucial lemmas. Neither is trivial to prove; the arguments can be found in Appendices B and C. In particular Lemma 8.11 works by a subtle case distinction over all the different ways the atomic names in the supports of u and u' can overlap.

LEMMA 8.10. *Recall from* (8) *the definition of step-functions. For each $\tau \in \mathrm{Typ}$, $u \in \mathrm{K}[\![\tau]\!]$ and $A \subseteq_{\mathrm{f}} \mathbb{A}$, if the step-function $(u \searrow \mathrm{true}) \in [\![\tau \to \mathtt{bool}]\!]$ is $\lambda\nu$-PCF$^+$-definable, then so is $\bigsqcup \mathrm{hull}_A\{(u \searrow \mathrm{true})\}$. (Note that the orbit-finite subset $\mathrm{hull}_A\{(u \searrow \mathrm{true})\}$ of the nominal Scott domain $[\![\tau \to \mathtt{bool}]\!] = [\![\tau]\!] \to 2_\perp$ is bounded above by the constantly-*true *function and so has a least upper bound.)*

PROOF. See Appendix B. □

LEMMA 8.11. *Suppose that $\tau \in \mathrm{Typ}$, $u, u' \in \mathrm{K}[\![\tau]\!]$ and $A \subseteq_{\mathrm{f}} \mathbb{A}$ satisfy:*

— for all uniform-compact elements $v, v' \in \mathrm{K}[\![\tau]\!]$ that do not have an upper bound in $[\![\tau]\!]$, the least upper bound $(v \searrow \mathrm{true}) \sqcup (v' \searrow \mathrm{false})$ (exists and) is $\lambda\nu$-PCF$^+$-definable;

— *for all finite permutations* $\pi : \mathbb{A} \cong \mathbb{A}$ *satisfying* $\pi \# A$ *(see Definition 4.3), it holds that* u *and* $\pi \cdot u'$ *do not have an upper bound in* $[\![\tau]\!]$.

Then the least upper bound $\bigsqcup \mathrm{hull}_A\{(u \searrow \mathrm{true}), (u' \searrow \mathrm{false})\}$ *(exists and) is* $\lambda\nu$-PCF^+-*definable.*

PROOF. See Appendix C. □

Using these two lemmas one can complete the proof of (DEF_σ) for all simple types $\sigma \in \mathrm{Styp}$ (and hence complete the proof of Theorem 8.5) by showing by simultaneous induction on the structure of σ that:

— u and $(u \searrow \mathrm{true})$ are definable for all uniform-compact elements $u \in \mathrm{K}[\![\sigma]\!]$;
— $(u \searrow \mathrm{true}) \sqcup (u' \searrow \mathrm{false})$ is definable whenever $u, u' \in \mathrm{K}[\![\sigma]\!]$ are uniform-compact elements that do not have an upper bound in $[\![\sigma]\!]$.

Most of the work involved in the proof of these two properties lies in the case for functions types, which for simple types are of the form, $\sigma \to \mathtt{nat}$. By Theorem 4.5 and 5.5 each uniform-compact element u of $[\![\sigma \to \mathtt{nat}]\!]$ can be expressed as $u = \bigsqcup \mathrm{hull}_A F$ for some $A \subseteq_{\mathrm{f}} \mathbb{A}$, $F = \{(u_1 \searrow n_1), \ldots, (u_k \searrow n_k)\}$, $u_1, \ldots, u_k \in \mathrm{K}[\![\sigma]\!]$ and $n_1, \ldots, n_k \in \mathbb{N}$. One has to perform another induction on the size of F and make a case distinction based on the existence of $(u' \searrow n'), (u'' \searrow n'') \in F$ such that for all $\pi \# A$ the uniform-compact elements u' and $\pi \cdot u''$ have no upper bound in $[\![\sigma]\!]$. Using Lemmas 8.10 and 8.11 the proof goes through following the structure of Streicher [2006, Theorem 13.9], thereby showing full abstraction for $\lambda\nu$-PCF^+.

Remark 8.12 (*generative names*). It is worth noting that the full abstraction result in this section depends crucially upon the fact that $\lambda\nu$-PCF^+ uses Odersky-style local names, rather than generative ones. In Remark 6.3 we noted that an extension of PCF with the kind of generative local names used for example in the ν-calculus [Pitts and Stark 1993] can be translated into $\lambda\nu$-PCF using continuations. Denotationally, generative names can be modelled adequately in Nsd with a continuation monad, but full abstraction fails in this model, because of the results of Stark [1994, Page 66]. So far the only denotational model of this combination of generative names with higher-order functions that is known to be fully abstract makes use of game semantics [Abramsky et al. 2004; Tzevelekos 2008] and is not extensional, that is, quotienting of game strategies by an equivalence relation is needed to make equality in the model coincide with contextual equivalence.

9. PNA: PROGRAMMING WITH NAME ABSTRACTIONS

As mentioned in the Introduction, our motivation for studying nominal Scott domains is to explore the denotational semantics of meta-languages that represent object-language binders using the nominal sets notion of *name abstraction* [Pitts 2013, Chapter 4]. We have seen in Section 7 that there is a straightforward denotational semantics of locally scoped names combined with arithmetic and higher-order recursive functions using nominal Scott domains and the notion of name restriction operation (Definition 7.1). Now we wish to extend this to cover name abstraction.

To do so, we extend $\lambda\nu$-PCF with constructs for name abstraction $\alpha a.\, e$ and concretion $e_1 \mathbin{@} e_2$, so that it can serve as meta-language. The resulting programming language is called PNA (Programming with Name Abstractions). To exercise the use of name abstraction for expressing object-level binding PNA also features a representative 'nominal algebraic datatype', namely a type for α-equivalence classes of λ-terms, called `term`. This datatype comes with three constructors (`V` for variables, `A` for applications and `L`

$\tau \in \mathrm{Typ} ::=$	*types*
$\vdots$	(as for $\lambda\nu$-PCF)
$\delta\,\tau$	type of name abstractions
$\mathtt{term}$	type of λ-terms
$e \in \mathrm{Exp} ::=$	*expressions*
$\vdots$	(as for $\lambda\nu$-PCF)
$\mathtt{V}\,e$	variable term
$\mathtt{A}\,e\,e$	application term
$\mathtt{L}\,e$	lambda term
$\mathtt{case}\,e\,\mathtt{of}\,(\mathtt{V}\,x \to e \mid \mathtt{A}\,x\,x \to e \mid \mathtt{L}\,x \to e)$	term case
$\alpha a.\,e$	name abstraction
$e\,@\,e$	name concretion
$c \in \mathrm{Can} ::=$	*canonical forms*
$\ldots$(as for $\lambda\nu$-PCF)$\ldots \mid \mathtt{V}\,c \mid \mathtt{A}\,c\,c \mid \mathtt{L}\,c \mid \alpha a.\,c$	

Fig. 7. Syntax of PNA

for λ-abstractions) and a pattern matching construct

$$\mathtt{case}\,e\,\mathtt{of}\,(\mathtt{V}\,x_1 \to e_1 \mid \mathtt{A}\,x_2\,x_2' \to e_2 \mid \mathtt{L}\,x_3 \to e_3)\,.$$

Using these new constructs, computation over α-equivalence classes of λ-terms can be expressed directly in PNA. For example, when $subst$ is the PNA expression

$$\begin{array}{l} subst \triangleq \lambda y : \mathtt{term} \to \lambda x : \mathtt{name} \to \\ \quad \mathtt{fix}\,(\lambda(f : \mathtt{term} \to \mathtt{term}) \to \lambda y' : \mathtt{term} \to \\ \qquad \mathtt{case}\,y'\,\mathtt{of} \\ \qquad\quad \mathtt{V}\,x_1 \to \mathtt{if}\,x_1 = x\,\mathtt{then}\,y\,\mathtt{else}\,y' \\ \qquad \mid \mathtt{A}\,y_2\,y_2' \to \mathtt{A}\,(f\,y_2)\,(f\,y_2') \\ \qquad \mid \mathtt{L}\,z \to \mathtt{L}\,(\alpha a.\,f(z\,@\,a))) \end{array}$$

then $subst\,e_1\,a\,e_2$ computes the λ-term obtained by capture-avoiding substitution of the λ-term represented by e_1 for all free occurrences of the variable named a in the λ-term represented by e_2.

The syntax and typing rules for PNA, extending those for $\lambda\nu$-PCF, are given in Figures 7 and 8. The language's binding forms are

$$\lambda x : \tau \to \_ \qquad \nu a.\,\_ \qquad \mathtt{case}\,e\,\mathtt{of}\,(\mathtt{V}\,x \to \_ \mid \mathtt{A}\,x\,x \to \_ \mid \mathtt{L}\,x \to \_) \qquad \alpha a.\,\_$$

and as before, we identify PNA expressions up to α-equivalence of bound identifiers, be they variables (x), or atomic names (a).

The operational semantics of PNA is given in Figures 9 and 10. As for $\lambda\nu$-PCF, we choose $\nu a.\,a$ to be a stuck expression that does not evaluate to any canonical form and whose denotation is $\bot$. We also choose to evaluate under name abstractions, so that $\alpha a.\,e$ is in canonical form if and only if e is.[5] These two choices permit a representation of α-equivalence classes of λ-terms in PNA that is as simple as PCF's representation of numbers: they are in bijection with variable-closed canonical forms of type $\mathtt{term}$.

[5]It is certainly possible to give a different operational semantics in which one does not evaluate under α-abstractions. The corresponding denotational semantics would make more use of lifting than does the one in Section 11.

$$\ldots\text{(as for } \lambda\nu\text{-PCF)}\ldots$$

$$\frac{\Gamma \vdash e : \mathtt{name}}{\Gamma \vdash \mathtt{V}\, e : \mathtt{term}} \qquad \frac{\Gamma \vdash e_1 : \mathtt{term} \qquad \Gamma \vdash e_2 : \mathtt{term}}{\Gamma \vdash \mathtt{A}\, e_1\, e_2 : \mathtt{term}} \qquad \frac{\Gamma \vdash e : \delta\,\mathtt{term}}{\Gamma \vdash \mathtt{L}\, e : \mathtt{term}}$$

$$\frac{\begin{array}{c}\Gamma \vdash e : \mathtt{term} \\ \Gamma, x_1 : \mathtt{name} \vdash e_1 : \tau \qquad \Gamma, x_2 : \mathtt{term}, x_2' : \mathtt{term} \vdash e_2 : \tau \qquad \Gamma, x_3 : \delta\,\mathtt{term} \vdash e_3 : \tau\end{array}}{\Gamma \vdash \mathtt{case}\, e\, \mathtt{of}\, (\mathtt{V}\, x_1 \to e_1 \mid \mathtt{A}\, x_2\, x_2' \to e_2 \mid \mathtt{L}\, x_3 \to e_3) : \tau}$$

$$\frac{a \in \mathbb{A} \qquad \Gamma \vdash e : \tau}{\Gamma \vdash \alpha a.\, e : \delta\,\tau} \qquad \frac{\Gamma \vdash e_1 : \delta\,\tau \qquad \Gamma \vdash e_2 : \mathtt{name}}{\Gamma \vdash e_1\, @\, e_2 : \tau}$$

Fig. 8. PNA typing rules

$$\ldots\text{(as for } \lambda\nu\text{-PCF)}\ldots$$

$$\frac{e \Downarrow c}{\mathtt{V}\, e \Downarrow \mathtt{V}\, c} \qquad \frac{e_1 \Downarrow c_1 \qquad e_2 \Downarrow c_2}{\mathtt{A}\, e_1\, e_2 \Downarrow \mathtt{A}\, c_1\, c_2} \qquad \frac{e \Downarrow c}{\mathtt{L}\, e \Downarrow \mathtt{L}\, c} \qquad \frac{e \Downarrow \mathtt{V}\, c \qquad e_1[c/x_1] \Downarrow c'}{\mathtt{case}\, e\, \mathtt{of}\, (\mathtt{V}\, x_1 \to e_1 \mid \cdots) \Downarrow c'}$$

$$\frac{e \Downarrow \mathtt{A}\, c\, c' \qquad e_2[c/x_2, c'/x_2'] \Downarrow c''}{\mathtt{case}\, e\, \mathtt{of}\, (\cdots \mid \mathtt{A}\, x_2\, x_2' \to e_2 \mid \cdots) \Downarrow c''} \qquad \frac{e \Downarrow \mathtt{L}\, c \qquad e_3[c/x_3] \Downarrow c'}{\mathtt{case}\, e\, \mathtt{of}\, (\cdots \mid \mathtt{L}\, x_3 \to e_3) \Downarrow c'}$$

$$\frac{e \Downarrow c}{\alpha a.\, e \Downarrow \alpha a.\, c} \qquad \frac{e_1 \Downarrow \alpha a.\, c \qquad e_2 \Downarrow a' \qquad a \neq a' \qquad \nu a.\, (a \rightleftharpoons a')\, c \Downarrow c'}{e_1\, @\, e_2 \Downarrow c'}$$

Fig. 9. PNA evaluation rules

$$\ldots\text{(as for } \lambda\nu\text{-PCF)}\ldots$$

$$\frac{a \backslash\backslash c := c'}{a \backslash\backslash \mathtt{V}\, c := \mathtt{V}\, c'} \qquad \frac{a \backslash\backslash c_1 := c_1' \qquad a \backslash\backslash c_2 := c_2'}{a \backslash\backslash \mathtt{A}\, c_1\, c_2 := \mathtt{A}\, c_1'\, c_2'} \qquad \frac{a \backslash\backslash c := c'}{a \backslash\backslash \mathtt{L}\, c := \mathtt{L}\, c'} \qquad \frac{a \backslash\backslash c := c' \qquad a \neq a'}{a \backslash\backslash \alpha a'.\, c := \alpha a'.\, c'}$$

Fig. 10. PNA partial operation of name restriction

Computing with name abstractions involves deconstructing them, and to do so PNA features a concretion operation $e\, @\, e'$ whose operational behaviour is given by the last rule in Figure 9. This rule might be surprising at first, but it will make sense once we consider the denotational semantics of name abstraction and concretion in the next section, in particular the second case of (52).

10. ABSTRACTION AND CONCRETION

The results in this section are the basis for giving denotational semantics using nominal Scott domains to languages with name abstraction operations, such as PNA from Section 9.

THEOREM 10.1. *If D is a nominal Scott domain, then so is the nominal poset $[\mathbb{A}]D$ from Definition 3.1. The operation of name abstraction $(a, d) \mapsto \langle a \rangle d$ extends to a morphism $\mathbb{A}_\perp \times D \to [\mathbb{A}]D$ in* **Nsd** *once we define $\langle \perp \rangle d \triangleq \perp$.*

PROOF. If $S \in \mathrm{P}_{\mathrm{fs}}([\mathbb{A}]D)$ is uniform-directed, then so is $\{d \in D \mid \langle a\rangle d \in S\} \in \mathrm{P}_{\mathrm{fs}}D$, for any $a \in \mathbb{A}$. The same holds if S is finitely supported and bounded from above. In all cases, picking any $a \# S$, one finds that the least upper bound of S in $[\mathbb{A}]D$ is $\langle a\rangle(\bigsqcup\{d \in D \mid \langle a\rangle d \in S\})$. Thus, $[\mathbb{A}]D$ has uniform-directed least upper bounds and least upper bounds of bounded finitely supported subsets, and its least element is $\langle a\rangle\bot$ (for any $a \in \mathbb{A}$). The above description of uniform-directed least upper bounds in $[\mathbb{A}]D$ implies that $\langle a\rangle u \in \mathrm{K}(\langle\mathbb{A}\rangle D)$ if and only if $u \in \mathrm{K}D$ and (hence) that $[\mathbb{A}]D$ is ω-algebraic. It also implies that each $\lambda d \in D \to \langle a\rangle d$ is uniform-continuous. □

Given a nominal Scott domain D and $d \in [\mathbb{A}]D$, for each $a \in \mathbb{A}$ with $a \# d$ there is a unique element $d @ a \in D$ satisfying $d = \langle a\rangle(d @ a)$, called the *concretion* [Pitts 2013, Section 4.3] of the name abstraction d at the atomic name a. Note that this operation is partially defined: to form $d @ a$ we require that a not be in the support of d. For flat domains we can make concretion a total, uniform-continuous function simply by mapping the pairs where concretion is undefined to $\bot \in D$. However, for non-flat domains this is not possible, because in general it does not give a monotone function. For example in $[\mathbb{A}](\mathrm{P}_{\mathrm{fs}}\mathbb{A})$, $a' \in \mathrm{supp}(\langle a\rangle\{a, a'\})$ (assuming $a \neq a'$), but we cannot define the concretion of $\langle a\rangle\{a, a'\}$ at a' to be the least element $\emptyset$ of $\mathrm{P}_{\mathrm{fs}}\mathbb{A}$ since $\langle a\rangle\{a\} \sqsubseteq \langle a\rangle\{a, a'\}$ and $(\langle a\rangle\{a\}) @ a' = \{a'\} \neq \emptyset$.

One way to deal with this partiality of concretion in a programming language is to enhance its type system with 'freshness assumptions' to ensure statically that name abstractions are only concreted at fresh atomic names. This is the solution adopted by the original version of FreshML [Pitts and Gabbay 2000] and is the one chosen by Winskel and Turner in their language Nominal HOPLA [Turner and Winskel 2009; Turner 2009]. Later versions of FreshML use a conventional type system and enforce freshness conditions dynamically via the use of generative local names in expressions [Shinwell et al. 2003]—at the expense of purity [Pottier 2007]. We do the same with the language PNA, but achieve purity via the use of Odersky-style local names [Odersky 1994] rather than generative ones. The meaning of this form of local name construct is type-directed and we used Theorem 7.2 to give its denotational semantics for $\lambda\nu$-PCF's types. This extends to PNA types because of the following result.

THEOREM 10.2. *If $D \in$ Nsd possesses a uniform-continuous name restriction operation (Definition 7.1), then the name abstraction domain $[\mathbb{A}]D$ also has one and it satisfies*

$$a\backslash(\langle a'\rangle d) = \langle a'\rangle(a\backslash d) \qquad \textit{if } a \neq a' \tag{51}$$

for all $a, a' \in \mathbb{A}$ and $d \in D$.

PROOF. First note that (51) uniquely defines name restriction for name abstraction domains because given $a \in \mathbb{A}$, we can always choose a representative for the equivalence class $\langle a'\rangle d$ with $a \neq a'$. The rest of the proof, that is showing that (29)–(32) hold, is a straightforward generalisation to nominal Scott domains of the corresponding result for the name abstraction operation on nominal sets [Pitts 2013, Theorem 9.18]. □

Following Pitts [2011, Corollary 2.14] and assuming $D \in$ Nsd has a uniform-continuous name restriction operation, we can extend the partial operation of concretion to a *total* equivariant function $\_ @^t \_ : [\mathbb{A}]D \times \mathbb{A}_\bot \to D$ satisfying

$$\left.\begin{array}{lcl} (\langle a\rangle d) @^t a & = & d \\ (\langle a\rangle d) @^t a' & = & a\backslash(a\ a') \cdot d \quad \text{if } a \neq a' \\ (\langle a\rangle d) @^t \bot & = & \bot. \end{array}\right\} \tag{52}$$

The fact that this is uniform-continuous and hence determines a morphism in Nsd follows from the description of uniform-directed least upper bounds in $[\mathbb{A}]D$ in the

$$\ldots(\text{as for } \lambda\nu\text{-PCF})\ldots$$

$$[\![\mathtt{V}\, e]\!]\rho = \begin{cases} [a]_\alpha & \text{if } [\![e]\!]\rho = a \\ \bot & \text{otherwise} \end{cases} \qquad [\![\mathtt{A}\, e_1\, e_2]\!]\rho = \begin{cases} [t_1\, t_2]_\alpha & \text{if } [\![e_i]\!]\rho = [t_i]_\alpha \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\mathtt{L}\, e]\!]\rho = \begin{cases} [\lambda a.t]_\alpha & \text{if } [\![e]\!]\rho = \langle a\rangle[t]_\alpha \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\mathtt{case}\, e\, \mathtt{of}\, (\mathtt{V}\, x_1 \to e_1 \mid \mathtt{A}\, x_2\, x_2' \to e_2 \mid \mathtt{L}\, x_3 \to e_3)]\!]\rho = \begin{cases} [\![e_1]\!]\rho[x_1 \mapsto a] & \text{if } [\![e]\!]\rho = [a]_\alpha \\ [\![e_2]\!]\rho[x_2 \mapsto [t]_\alpha, x_2' \mapsto [t']_\alpha] & \text{if } [\![e]\!]\rho = [t\, t']_\alpha \\ [\![e_3]\!]\rho[x_3 \mapsto \langle a\rangle[t]_\alpha] & \text{if } [\![e]\!]\rho = [\lambda a.t]_\alpha \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\alpha a.\, e]\!]\rho = \langle a\rangle([\![e]\!]\rho) \quad \text{if } a \# \rho \qquad [\![e_1 \mathbin{@} e_2]\!]\rho = ([\![e_1]\!]\rho) \mathbin{@^t} ([\![e_2]\!]\rho)$$

Fig. 11. PNA denotational semantics

proof of Theorem 10.1. For fresh names, the total concretion operation agrees with the 'usual' concretion

$$(\forall d \in [\mathbb{A}]D)\ a \# d \ \Rightarrow\ d \mathbin{@^t} a = d \mathbin{@} a\,.$$

We will use the total concretion morphism to interpret name concretion expressions in the denotational semantics of PNA.

11. DENOTATIONAL SEMANTICS OF PNA

As for $\lambda\nu$-PCF, types in PNA denote nominal Scott domains and well-typed expressions $\Gamma \vdash e : \tau$ denote uniform-continuous functions $[\![e]\!] \in [\![\Gamma]\!] \to [\![\tau]\!]$. Denotations for typing environments Γ, booleans `bool`, numbers `nat`, product types $\tau_1 \times \tau_2$ and function types $\tau_1 \to \tau_2$ are the same as the $\lambda\nu$-PCF definitions in Section 7. The type of λ-terms `term` and the type of name abstractions $\delta\,\tau$ are denoted as follows:

— $[\![\mathtt{term}]\!] = (\Lambda_\alpha)_\bot$, the flat domain on the nominal set of α-equivalence classes of λ-terms [Pitts 2013, Section 4.1],

$$\Lambda_\alpha \triangleq \{t ::= a \mid \lambda a.t \mid t\, t\}/{=_\alpha} \quad \text{(where } a \in \mathbb{A}\text{).} \tag{53}$$

— $[\![\delta\,\tau]\!] = [\mathbb{A}][\![\tau]\!]$, the domain of name abstractions of the nominal Scott domain $[\![\tau]\!]$ (Theorem 10.1).

Figure 11 gives the additional definitions for the denotations of PNA expressions. In the clauses involving expressions of type `term`, we use $[t]_\alpha$ to denote the α-equivalence class of the syntax tree t of a λ-term. By Theorems 7.2 and 10.2 the denotation of every PNA-type has a uniform-continuous name restriction operation. This operation is used in the definitions of $[\![\nu a.\, e]\!]$ and $e_1 \mathbin{@} e_2$, where for the latter we use the total concretion morphism from (52). Note that the side condition in the clause for $\alpha a.\, e$ can always be satisfied, because we identify expression up to α-equivalence. The $\lambda\nu$-PCF results from Section 7 carry over to PNA.

LEMMA 11.1 (PNA SOUNDNESS). *If $e \Downarrow c$, then $[\![e]\!] = [\![c]\!]$.*

PROOF. The Substitution Lemma, Equivariance Lemma and Restriction Lemma mentioned in the proof of Lemma 7.4 extend from $\lambda\nu$-PCF to PNA. Soundness follows then by rule induction on $e \Downarrow c$. □

THEOREM 11.2 (PNA COMPUTATIONAL ADEQUACY). *The* PNA *contextual preorder* ($\Gamma \vdash e \leq_{\mathsf{PNA}} e' : \tau$) *and contextual equivalence* ($\Gamma \vdash e \cong_{\mathsf{PNA}} e' : \tau$) *are defined as for* $\lambda\nu$-PCF *(Definition 6.4). For any* $\Gamma \vdash e : \tau$ *and* $\Gamma \vdash e' : \tau$, *if* $[\![e]\!] \sqsubseteq [\![e']\!] \in [\![\Gamma]\!] \to [\![\tau]\!]$, *then* $\Gamma \vdash e \leq_{\mathsf{PNA}} e' : \tau$, *and hence we have also that* $[\![e]\!] = [\![e']\!] \in [\![\Gamma]\!] \to [\![\tau]\!]$ *implies* $\Gamma \vdash e \cong_{\mathsf{PNA}} e' : \tau$.

PROOF. The overall structure of the proof is the same as in Theorem 7.5, but the presence of name abstraction and concretion creates complications: see Appendix D. □

12. FULL ABSTRACTION FOR PNA^+

In this section we show that the full abstraction result from Section 8 can be extended to include computation with name abstractions. With some tweaks and extensions, the proof goes through as for $\lambda\nu$-PCF^+. We extended $\lambda\nu$-PCF with existential name quantification and definite name description in order to gain enough expressivity for full abstraction. As Theorem 12.2 shows, the same extensions suffice for PNA.

Definition 12.1. PNA^+ is the extension of PNA with constructs for existential name quantification and definite name description, as they are given in Figure 6: $\mathsf{PNA}^+ = \mathsf{PNA} + \mathtt{ex} + \mathtt{the}$. Similarly we define the languages PNA+ex and PNA+the. The contextual preorder relations $\leq_{\mathsf{PNA}^+}$, $\leq_{\mathsf{PNA+ex}}$ and $\leq_{\mathsf{PNA+the}}$ are given as in Definition 6.4.

The results of Appendix A can be extended to PNA, giving us the equivalents of (49) and (50)

$$\emptyset \vdash \mathtt{F}_1 \leq_{\mathsf{PNA+the}} \mathtt{F}_2 : ((\mathtt{name} \to \mathtt{bool}) \to \mathtt{bool}) \to \mathtt{bool}$$
$$\emptyset \vdash \mathtt{G}_1 \leq_{\mathsf{PNA+ex}} \mathtt{G}_2 : ((\mathtt{name} \to \mathtt{bool}) \to \mathtt{name}) \to \mathtt{bool}\,.$$

Arguing as in the proof of Theorem 8.4, this shows that PNA+ex and PNA+the fail to be fully abstract. More positively, Theorem 8.5 can be adjusted to the current setting, showing that PNA^+ is fully abstract.

THEOREM 12.2 (FULL ABSTRACTION FOR PNA^+). *For all well-typed expressions* $\Gamma \vdash e : \tau$ *and* $\Gamma \vdash e' : \tau$ *in* PNA^+, *we have*

$$[\![e]\!] \sqsubseteq [\![e']\!] \in [\![\Gamma]\!] \to [\![\tau]\!] \;\Leftrightarrow\; \Gamma \vdash e \leq_{\mathsf{PNA}^+} e' : \tau \qquad (\mathsf{PNA}^+\text{-}\mathrm{FA}_\tau)$$

and hence also $[\![e]\!] = [\![e']\!] \in [\![\Gamma]\!] \to [\![\tau]\!] \Leftrightarrow \Gamma \vdash e \cong_{\mathsf{PNA}^+} e' : \tau$.

In the rest of this section we sketch how the proof of Theorem 8.5 can be extended to a proof of Theorem 12.2.

As for $\lambda\nu$-PCF^+, we reduce the full abstraction property (PNA^+-FA_τ) at arbitrary PNA^+ types to this property for simple types, using the same definition of 'simple type' as before (Definition 8.7). However, the analogue of Lemma 8.8 for PNA^+ fails: it is not the case that there is a definable embedding-projection pair from each PNA^+ type to some simple type (see the example given below in Open Problem 12.5). However, the property holds for definable retracts instead of definable embedding-projection pairs and this suffices for our proof.

LEMMA 12.3. *A type* τ *is a* PNA^+*-definable retract of a type* σ *if there are closed* PNA^+*-expressions* $\emptyset \vdash i : \tau \to \sigma$ *and* $\emptyset \vdash r : \sigma \to \tau$ *so that* $[\![\lambda x : \tau \to r\,(i\,x)]\!] = id_{[\![\tau]\!]}$. *It holds that every* PNA^+ *type* $\tau \in \mathrm{Typ}$ *is a is a* PNA^+*-definable retract of some* $\sigma \in \mathrm{Styp}$.

PROOF. Since name abstraction satisfies a form of η-expansion ($[\![\alpha a.\,(e\,@\,a)]\!] = [\![e]\!]$, if $a \notin \mathrm{fn}(e)$), each type $\delta\,\tau$ is a PNA^+-definable retract of $\mathtt{name} \to \tau$ via

$$\begin{aligned} i &\triangleq \lambda(x : \delta\,\tau) \to \lambda y : \mathtt{name} \to x\,@\,y \\ r &\triangleq \lambda(f : \mathtt{name} \to \tau) \to \alpha a.\, f\,a \end{aligned} \qquad (54)$$

(*cf.* Pitts [2011, Theorem 2.13]). It follows that if τ is a PNA^+-definable retract of a simple type, then so is $\delta\,\tau$ (using the fact implicit in the proof of Lemma 8.8 that if σ_1 and σ_2 are simple, then $\sigma_1 \to \sigma_2$ is a definable retract of a simple type).

Using a suitable Gödel-numbering of λ-terms in the presence of an environment consisting of a finite list of distinct atomic names (giving the free variables of the λ-term) one can exhibit $\mathtt{term}$ as a PNA^+-definable retract of the simple type $\sigma \times \mathtt{nat}$, where the second component $\mathtt{nat}$ is used to codes the term's syntax tree and the first component $\sigma \triangleq \mathtt{nat} \times (\mathtt{name} \to \mathtt{nat})$ is used to code environments as a pair consisting of list-length and a function (taking value 0 at all but finitely many arguments) giving positions in the list.

Every embedding-projection pair forms a retract, so we can combine these observations about $\delta\,\tau$ and $\mathtt{term}$ with the argument in the proof of Lemma 8.8 to conclude that every PNA^+-type is a definable retract of a simple type. □

We can use the above lemma to show that definability at simple types implies full abstraction at all types.

LEMMA 12.4. *It holds that* $(\forall\sigma \in \mathrm{Styp})\ (\mathrm{DEF}_\sigma)$ *implies* $(\forall\tau \in \mathrm{Typ})\ (\mathsf{PNA}^+\text{-}\mathrm{FA}_\tau)$.

PROOF. The left-to-right direction of $(\mathsf{PNA}^+\text{-}\mathrm{FA}_\tau)$ of course holds already by computational adequacy (Theorem 11.2). We prove the right-to-left direction first for closed expressions of simple type, that is $[\![e]\!] \not\sqsubseteq [\![e']\!] \in [\![\sigma]\!] \Rightarrow \emptyset \vdash e \not\leq_{\mathsf{PNA}^+} e' : \sigma$. This proof is straight-forward by following the classical definability argument as surveyed by Curien [2007, Criterion 2.2].

For showing the property for open expressions of simple type $[\![e]\!] \not\sqsubseteq [\![e']\!] \in [\![\Gamma]\!] \to [\![\sigma]\!] \Rightarrow \Gamma \vdash e \not\leq_{\mathsf{PNA}^+} e' : \sigma$, we need to take into account that the environment $\Gamma = \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$ may contain non-simple types. By Lemma 12.3 we know that there is a $\sigma \in \mathrm{Styp}$ such that $\tau_1 \times \ldots \times \tau_n$ is a definable retract of σ, say with expressions i and r. It follows from $[\![e]\!] \not\sqsubseteq [\![e']\!]$ that there must be a $u \in \mathrm{K}[\![\sigma]\!]$ (which is definable) such that $[\![e]\!]([\![r]\!]\,u_1) \not\sqsubseteq [\![e]\!]([\![r]\!]\,u_1) \in [\![\sigma]\!]$ and with that we can use the above argument for closed expressions to show $\Gamma \vdash e \not\leq_{\mathsf{PNA}^+} e' : \sigma$.

What is left to prove is that if τ is a definable retract of σ, then $(\mathsf{PNA}^+\text{-}\mathrm{FA}_\sigma)$ implies $(\mathsf{PNA}^+\text{-}\mathrm{FA}_\tau)$. Suppose $(\mathsf{PNA}^+\text{-}\mathrm{FA}_\sigma)$ holds and that $\Gamma \vdash e \leq_{\mathsf{PNA}^+} e' : \tau$. Then we have $\Gamma \vdash i\,e \leq_{\mathsf{PNA}^+} i\,e' : \sigma$ by compositionality of $\leq_{\mathsf{PNA}^+}$. Thus by (FA_σ), for any $\rho \in [\![\Gamma]\!]$ we have $[\![i]\!]([\![e]\!]\rho) = [\![i\,e]\!]\rho \sqsubseteq [\![i\,e']\!]\rho = [\![i]\!]([\![e']\!]\rho)$. We know that $[\![i]\!]$ has a monotone left inverse $[\![r]\!]$, so $[\![e]\!]\rho \sqsubseteq [\![e']\!]\rho$. As ρ was chosen arbitrarily we get $[\![e]\!] \sqsubseteq [\![e']\!] \in [\![\Gamma]\!] \to [\![\tau]\!]$. □

$\lambda\nu\text{-}\mathsf{PCF}^+$ is included in PNA^+, so the proof of uniform-compact definability at simple types $(\forall\sigma \in \mathrm{Styp})\ (\mathrm{DEF}_\sigma)$ for $\lambda\nu\text{-}\mathsf{PCF}^+$ in Section 8 directly translates to PNA^+. This completes the proof of full abstraction of PNA^+ by Lemma 12.4.

We conclude this section with the discussion of some open problems.

Open Problem 12.5. *For an arbitrary* PNA^+ *type* τ*, are the uniform-compact elements of* $[\![\tau]\!]$ PNA^+*-definable?*

The definable retract (54) is not in general an embedding-projection pair. For example, when $\tau = \mathtt{bool}$ one can calculate that $[\![r]\!] \in (\mathbb{A}_\perp \to 2_\perp) \to [\mathbb{A}]2_\perp$ maps the element $eq_a \in \mathbb{A}_\perp \to 2_\perp$ from Example 5.6 to $\langle a\rangle\mathrm{false}$; and that $[\![i]\!]$ maps this name abstraction to $k_{\mathrm{false}} \triangleq \lambda d \in \mathbb{A}_\perp \to \mathrm{if}\ d = \perp\ \mathrm{then}\ \perp\ \mathrm{else\ false}$. Since $eq_a \not\sqsubseteq k_{\mathrm{false}}$, we do not have that $[\![\lambda(x : \mathtt{name} \to \mathtt{bool}) \to i\,(r\,x)]\!] \sqsubseteq id_{[\![\mathtt{name}\to\mathtt{bool}]\!]}$. So it seems that PNA^+-types are not definably embeddable into simple types. As a result, we cannot deduce the definability property of Definition 8.6 at arbitrary types from the special case for simple types.

Open Problem 12.6. *Is there a fully abstract model of* PNA *based on games in nominal sets?*

Just as PCF is of more interest from a programming point of view than PCF+por, we regard PNA (suitably extended with recursive types) as a 'pure' version of FreshML that is potentially useful for functional programming with syntactical data involving binders. Game semantics provided an interesting solution for the original full abstraction problem for PCF [Abramsky et al. 2000; Hyland and Ong 2000], and its nominal version has provided computationally useful, fully abstract models of generative local state [Abramsky et al. 2004; Tzevelekos 2008; Laird 2008; Murawski and Tzevelekos 2012]. Can nominal game semantics provide a similar thing for PNA?

Open Problem 12.7. *Is there a nominal Scott domain semantics for the form of nominal computation embodied by the* Nλ *language [Bojańczyk et al. 2012]?*

With Nλ, Bojańczyk *et al.* extend the simply-typed λ-calculus with a collection type representing orbit-finite subsets (Section 2) via a syntax for hulls (Definition 4.3).[6] It is natural to consider adding fixed point recursion to this language, with a denotational semantics using nominal domains rather than nominal sets. The denotational semantics of such an extension of Nλ will require the development of *orbit-finite* power domains $\mathsf{F}_{\mathrm{n}}D$ in **Nsd**, whose uniform-compact elements are orbit-finite subsets of the uniform-compact elements of D.

Open Problem 12.8. *What recursive domain equations can be solved in* **Nsd***?*

In his thesis, Shinwell [2005, Section 4.5] shows that the traditional method for constructing minimally invariant solutions for locally continuous functors of mixed variance can be applied to the simple notion of nominal domain given by nominal posets with least upper bounds of finitely supported ω-chains. This can be extended to udcpos: see Pitts [2013, Section 11.4]. An interesting alternative approach is to develop a nominal version of Scott's information systems [Scott 1982] and construct solutions for recursive domain equations via inductively defined nominal sets of information tokens. We have begun to develop such a theory of *nominal Scott information systems* in which the role of finite sets is replaced by orbit-finite nominal sets. From a logical point of view [Abramsky 1991], nominal information systems are presentations of non-trivial nominal posets *with all orbit-finite meets*, rather than just finite meets. We expect this machinery can be used to good effect for the orbit-finite power domain construct mentioned above, as well as for a version for nominal Scott domains of Moggi's monad for dynamic allocation [Moggi 1989, Section 4.14] featuring a *freely generated* uniform-continuous name restriction operation (cf. Pitts [2013, Section 9.5]).

13. CONCLUSION

The results in this paper provide further evidence for how a semantic theory (domain theory in this case) is enhanced by using nominal sets: we gain the ability to model constructs involving names and their symmetries while preserving most aspects of the classical theory. The complications arising from the use of nominal sets are feasible and somehow orthogonal to the other developments. At the same time, their use gives access to new constructs that are far from trivial. This is the case for the notion of *orbit-finite subset*, which formalizes the important idea of finiteness modulo symmetry within nominal sets. We agree with Bojańczyk et al. [2012] that this is an important notion with many potential applications. Here we have used it to develop a nominal domain theory that, via our full abstraction result, has a good fit with higher-type computation involving local names and name abstractions.

[6] Their paper [Bojańczyk et al. 2012] is concerned with general 'Fraïssé nominal sets'. Here we restrict our attention to the 'equality symmetry' and nominal sets in the original sense.

APPENDIX

A. PROOFS OF (49) AND (50)

We first establish some extensionality results for the $\lambda\nu$-PCF$^+$ contextual preorder (Proposition A.3). These results specialise to $\lambda\nu$-PCF+the and $\lambda\nu$-PCF+ex and hence can be used as tools for proving (49) and (50).

Notation A.1. In order to make a better distinction between the programming languages in use, for this section we tag syntactic sets with the languages they belong to. So $\mathrm{Exp}^{\lambda\nu\text{-}\mathsf{PCF}^+}$ is the set of expressions that belong to $\lambda\nu$-PCF$^+$; ditto $\mathrm{Exp}^{\lambda\nu\text{-}\mathsf{PCF}+\mathtt{the}}$ and $\mathrm{Exp}^{\lambda\nu\text{-}\mathsf{PCF}+\mathtt{ex}}$, as well as $\mathrm{Can}^{\lambda\nu\text{-}\mathsf{PCF}^+}$ and $\mathrm{Typ}^{\lambda\nu\text{-}\mathsf{PCF}^+}$. We also extend the logical relation (42) to open expressions $\Gamma \vdash e : \tau$ by defining

$$d \lhd_{\Gamma|\tau} e \;\triangleq\; (\forall \rho, s)\; \rho \lhd_\Gamma s \;\Rightarrow\; d\,\rho \lhd_\tau e\,s$$

where $d \in [\![\Gamma]\!] \to [\![\tau]\!]$, ρ is a Γ-valuation, s is a Γ-substitution and $\lhd_\Gamma$ is defined in (43).

LEMMA A.2. *All expressions $e_1, e_2 \in \mathrm{Exp}^{\lambda\nu\text{-}\mathsf{PCF}^+}$ satisfy*

$$\Gamma \vdash e_1 \leq_{\lambda\nu\text{-}\mathsf{PCF}^+} e_2 : \tau \;\Leftrightarrow\; [\![e_1]\!] \lhd_{\Gamma|\tau} e_2.$$

PROOF. As noted in Theorem 8.5, the adequacy results in Section 7, in particular the fundamental property of the logical relation (44), extend from $\lambda\nu$-PCF to $\lambda\nu$-PCF$^+$. We use these results in this and the forthcoming proofs.

Suppose $[\![e_1]\!] \lhd_{\Gamma|\tau} e_2$ and let a context $C[\_]$ be given with $\emptyset \vdash C[e_1] : \mathtt{bool}$, $\emptyset \vdash C[e_2] : \mathtt{bool}$ and $C[e_1] \Downarrow \mathtt{T}$. Lemma 7.4 implies $[\![C[e_1]]\!] = \mathrm{true}$. We can show by induction on the structure of $C[\_]$ that $[\![e_1]\!] \lhd_{\Gamma|\tau} e_2 \Rightarrow [\![C[e_1]]\!] \lhd_{\mathtt{bool}} C[e_2]$. Therefore $\mathrm{true} \lhd_{\mathtt{bool}} C[e_2]$; and hence $C[e_2] \Downarrow \mathtt{T}$, by definition of $\lhd_{\mathtt{bool}}$. Thus we obtain $\Gamma \vdash e_1 \leq_{\lambda\nu\text{-}\mathsf{PCF}^+} e_2 : \tau$.

To show the converse, one proves by induction on τ that

$$(d \lhd_{\Gamma|\tau} e_1 \;\wedge\; \Gamma \vdash e_1 \leq_{\lambda\nu\text{-}\mathsf{PCF}^+} e_2 : \tau) \;\Rightarrow\; d \lhd_{\Gamma|\tau} e_2\,. \tag{55}$$

Then we can apply this to $[\![e_1]\!] \lhd_{\Gamma|\tau} e_1$, which holds by the fundamental property (44). □

PROPOSITION A.3 (EXTENSIONALITY). *The contextual preorder for $\lambda\nu$-PCF$^+$ is extensional at functions and ground types for variable-closed expressions. That is for all $e_1, e_2 \in \mathrm{Exp}^{\lambda\nu\text{-}\mathsf{PCF}^+}$ we have that if $\gamma \in \{\mathtt{bool}, \mathtt{nat}, \mathtt{name}\}$, then $\emptyset \vdash e_1 \leq_{\lambda\nu\text{-}\mathsf{PCF}^+} e_2 : \gamma$ holds if and only if*

$$(\forall c)\; e_1 \Downarrow c \Rightarrow e_2 \Downarrow c\,.$$

At function types $\emptyset \vdash e_1 \leq_{\lambda\nu\text{-}\mathsf{PCF}^+} e_2 : \tau_1 \to \tau_2$ holds if and only if

$$(\forall e)\; \emptyset \vdash e : \tau_1 \;\Rightarrow\; \emptyset \vdash e_1\,e \leq_{\lambda\nu\text{-}\mathsf{PCF}^+} e_2\,e : \tau_2\,.$$

PROOF. The only-if directions are immediate from the definition of $\leq_{\lambda\nu\text{-}\mathsf{PCF}^+}$. Conversely, Lemma A.2 tells us that it is enough to show $[\![e_1]\!] \lhd_\tau e_2$. In the ground type case $\tau = \gamma$, if $[\![e_1]\!] = \bot$ then $[\![e_1]\!] \lhd_\gamma e_2$ follows directly. Otherwise we must have $[\![e_1]\!] = [\![c]\!]$ which implies $e_1 \Downarrow c$ by the fundamental property (44) at e_1. This implies $e_2 \Downarrow c$ by assumption, and hence $[\![e_1]\!] \lhd_\gamma e_2$, as required. In the function case $\tau = \tau_1 \to \tau_2$, let any $d \lhd_{\tau_1} e$ be given. With the fundamental property we get $[\![e_1]\!] \lhd_{\tau_1\to\tau_2} e_1$, so by definition $[\![e_1]\!]\,d \lhd_{\tau_2} e_1\,e$. By assumption we know $\emptyset \vdash e_1\,e \leq_{\lambda\nu\text{-}\mathsf{PCF}^+} e_2\,e : \tau_2$ and with (55) this gives us $[\![e_1]\!]\,d \lhd_{\tau_2} e_2\,e$. This shows $[\![e_1]\!] \lhd_{\tau_1\to\tau_2} e_2$, by the definition of $\lhd_{\tau_1\to\tau_2}$; and then we can apply Lemma A.2 to get $\emptyset \vdash e_1 \leq_{\lambda\nu\text{-}\mathsf{PCF}^+} e_2 : \tau_1 \to \tau_2$. □

Extensionality holds with the same proofs also for the contextual preorders of $\lambda\nu$-PCF+the and $\lambda\nu$-PCF+ex. This allows us to deduce (49) from the following property of the $\lambda\nu$-PCF+the evaluation relation

$$a \# e \ \wedge \ (\forall c)\, e[\mathtt{eqBot}_a/x] \Downarrow c \ \Rightarrow \ e[\mathtt{kBot}/x] \Downarrow c \tag{56}$$

that holds for any expression $e \in \mathrm{Exp}^{\lambda\nu\text{-}\mathsf{PCF}+\mathtt{the}}$ of type $x : \mathtt{name} \to \mathtt{bool} \vdash e : \mathtt{bool}$; $\mathtt{eqBot}_a$ and $\mathtt{kBot}$ are defined in (20) and (21). In the same way we can use extensionality to deduce (50) from the following property of the $\lambda\nu$-PCF+ex evaluation relation

$$a \# e \ \wedge \ e[\mathtt{eq}_a/x] \Downarrow a' \ \Rightarrow \ a \neq a' \tag{57}$$

where $e \in \mathrm{Exp}^{\lambda\nu\text{-}\mathsf{PCF}+\mathtt{ex}}$ with $x : \mathtt{name} \to \mathtt{bool} \vdash e : \mathtt{name}$ and $\mathtt{eq}_a$ is defined in (24).

We prove (56) and (57) by reformulating the evaluation relation $\Downarrow$ in terms of a small-step transition relation in the style of Felleisen and Hieb [1992], but with evaluation contexts formulated as stacks of evaluation frames. We call this type of operational semantics *frame-stack semantics*, it also known as abstract machine semantics in the literature. (See Pitts [2002] for a discussion of this technique for proving contextual equivalences.)

The frame-stack evaluation relation for $\lambda\nu$-PCF$^+$

$$\langle F \,,\, e\rangle \to \langle F' \,,\, e'\rangle$$

is defined in Figure 12. We use the name restriction operation from Figure 4 in the clause for $\nu a.\, e$. For brevity, the cases for $\mathtt{ex}\, x.\, e$ and $\mathtt{the}\, x.\, e$ use some syntactic sugar: $\mathtt{bot}_\tau$ from (19) and a pattern matching construct $\mathtt{case}_{\,\mathtt{bool}}$ for boolean tuples, which can be defined by conditionals for any tuple length.

The evaluation relation is formulated in terms of a transition system between *configurations*. However, a configurations $\langle F \,,\, e\rangle$ is not just a pair built from a *frame-stack* $F \in \mathit{Stack}^{\lambda\nu\text{-}\mathsf{PCF}^+}$ and an expression $e \in \mathrm{Exp}^{\lambda\nu\text{-}\mathsf{PCF}^+}$: We identify configurations by a form of α-equivalence where a binder in a frame (for $\lambda\nu$-PCF$^+$ only the frame $\nu a.\,\cdot$) can bind free identifiers in later frames or the expression. For example $\langle \mathrm{Id} \circ (\nu a.\,\cdot) \circ (\cdot = a) \,,\, a\rangle$ is α-equivalent to $\langle \mathrm{Id} \circ (\nu a'.\,\cdot) \circ (\cdot = a') \,,\, a'\rangle$, but it is not α-equivalent to $\langle \mathrm{Id} \,,\, \nu a.\, a = a\rangle$. This identification is implicit and results in the following equality for configurations

$$\langle F \,,\, e\rangle = \langle F' \,,\, e'\rangle \ \Leftrightarrow \ |F| = |F'| \ \wedge \ F[e] = F'[e']$$

where $|F|$ is the length of the frame-stack F. Let $\mathrm{Config}^{\lambda\nu\text{-}\mathsf{PCF}^+}$ be the set of all such configurations for $\lambda\nu$-PCF$^+$. The notions of permutation action, free names, free variables and capture-avoiding substitution extend to configurations in the obvious way, taking into account this form of α-equivalence for configurations.

Definition A.4. For each $n \in \mathbb{N}$, let the relation $\to^n$ be defined by

$$\begin{aligned} \langle F \,,\, e\rangle \to^0 \langle F' \,,\, e'\rangle \ &\triangleq \ \langle F \,,\, e\rangle = \langle F' \,,\, e'\rangle \\ \langle F \,,\, e\rangle \to^{n+1} \langle F' \,,\, e'\rangle \ &\triangleq \ \langle F \,,\, e\rangle \to \langle F'' \,,\, e''\rangle \ \wedge \ \langle F'' \,,\, e''\rangle \to^n \langle F' \,,\, e'\rangle \end{aligned}$$

and let $\langle F,e\rangle \to^* \langle F',e'\rangle$ be defined to hold if there is a $n \in \mathbb{N}$ such that $\langle F,e\rangle \to^n \langle F',e'\rangle$.

The operational semantics from Section 6 and the frame-stack semantics agree:

PROPOSITION A.5. *For all* $e \in \mathrm{Exp}^{\lambda\nu\text{-}\mathsf{PCF}^+}$ *and* $c \in \mathrm{Can}^{\lambda\nu\text{-}\mathsf{PCF}^+}$ *we have* $e \Downarrow c \Leftrightarrow \langle \mathrm{Id} \,,\, e\rangle \to^* \langle \mathrm{Id} \,,\, c\rangle$.

PROOF. We prove $e \Downarrow c \Rightarrow (\forall F \in \mathit{Stack}^{\lambda\nu\text{-}\mathsf{PCF}^+})\, \langle F \,,\, e\rangle \to^* \langle F \,,\, c\rangle$ by rule induction on $e \Downarrow c$ for the left-to-right direction. The right-to-left direction is a consequence of the more general statement $(\forall n \in \mathbb{N})\, \langle F \,,\, e\rangle \to^n \langle \mathrm{Id} \,,\, c\rangle \Rightarrow F[e] \Downarrow c$ (proved by induction on

$$F \in \mathit{Stack}^{\mathsf{PNA}^+} ::= \mathrm{Id} \mid F \circ E$$
$$E \in \mathit{Frame}^{\mathsf{PNA}^+} ::= \mathtt{if}\cdot\,\mathtt{then}\; e\;\mathtt{else}\; e \mid \mathtt{S}\cdot \mid \mathtt{pred}\cdot \mid \mathtt{zero}\cdot \mid \mathtt{fst}\cdot \mid \mathtt{snd}\cdot \mid \cdot\, e \mid \nu a.\cdot \mid (\cdot \rightleftharpoons e)\, e \mid (a \rightleftharpoons \cdot)\, e \mid (a \rightleftharpoons a)\cdot \mid \cdot\, \mathtt{=}\, e \mid a\, \mathtt{=}\, \cdot$$

$$\langle F\,,\mathtt{if}\, e_1\,\mathtt{then}\, e_2\,\mathtt{else}\, e_3\rangle \to \langle F \circ \mathtt{if}\cdot\,\mathtt{then}\, e_2\,\mathtt{else}\, e_3\,, e_1\rangle$$

$$\langle F \circ \mathtt{if}\cdot\,\mathtt{then}\, e_2\,\mathtt{else}\, e_3\,,\mathtt{T}\rangle \to \langle F\,, e_2\rangle \qquad \langle F \circ \mathtt{if}\cdot\,\mathtt{then}\, e_2\,\mathtt{else}\, e_3\,,\mathtt{F}\rangle \to \langle F\,, e_3\rangle$$

$$\langle F\,,\mathtt{S}\, e\rangle \to \langle F \circ \mathtt{S}\cdot\,, e\rangle \qquad \langle F \circ \mathtt{S}\cdot\,, c\rangle \to \langle F\,,\mathtt{S}\, c\rangle \qquad \langle F\,,\mathtt{pred}\, e\rangle \to \langle F \circ \mathtt{pred}\cdot\,, e\rangle$$

$$\langle F \circ \mathtt{pred}\cdot\,,\mathtt{S}\, c\rangle \to \langle F\,, c\rangle \qquad \langle F\,,\mathtt{zero}\, e\rangle \to \langle F \circ \mathtt{zero}\cdot\,, e\rangle \qquad \langle F \circ \mathtt{zero}\cdot\,,\mathtt{0}\rangle \to \langle F\,,\mathtt{T}\rangle$$

$$\langle F \circ \mathtt{zero}\cdot\,,\mathtt{S}\, c\rangle \to \langle F\,,\mathtt{F}\rangle \qquad \langle F\,,\mathtt{fst}\, e\rangle \to \langle F \circ \mathtt{fst}\cdot\,, e\rangle \qquad \langle F \circ \mathtt{fst}\cdot\,,(e_1\,, e_2)\rangle \to \langle F\,, e_1\rangle$$

$$\langle F\,,\mathtt{snd}\, e\rangle \to \langle F \circ \mathtt{snd}\cdot\,, e\rangle \qquad \langle F \circ \mathtt{snd}\cdot\,,(e_1\,, e_2)\rangle \to \langle F\,, e_2\rangle \qquad \langle F\,, e_1\, e_2\rangle \to \langle F \circ \cdot\, e_2\,, e_1\rangle$$

$$\langle F \circ \cdot\, e_2\,, \lambda x : \tau \to e\rangle \to \langle F\,, e[e_2/x]\rangle \qquad \langle F\,,\mathtt{fix}\, e\rangle \to \langle F\,, e\,(\mathtt{fix}\, e)\rangle$$

$$\langle F\,, \nu a.\, e\rangle \to \langle F \circ \nu a.\cdot\,, e\rangle \qquad \langle F \circ \nu a.\cdot\,, c\rangle \to \langle F\,, c'\rangle \quad \text{if } a \backslash\backslash c := c'$$

$$\langle F\,,(e_1 \rightleftharpoons e_2)\, e_3\rangle \to \langle F \circ (\cdot \rightleftharpoons e_2)\, e_3\,, e_1\rangle \qquad \langle F \circ (\cdot \rightleftharpoons e_2)\, e_3\,, a_1\rangle \to \langle F \circ (a_1 \rightleftharpoons \cdot)\, e_3\,, e_2\rangle$$

$$\langle F \circ (a_1 \rightleftharpoons \cdot)\, e_3\,, a_2\rangle \to \langle F \circ (a_1 \rightleftharpoons a_2)\cdot\,, e_3\rangle \qquad \langle F \circ (a_1 \rightleftharpoons a_2)\cdot\,, c\rangle \to \langle F\,,(a_1\; a_2)\cdot c\rangle$$

$$\langle F\,, e_1\,\mathtt{=}\, e_2\rangle \to \langle F \circ \cdot\,\mathtt{=}\, e_2\,, e_1\rangle \qquad \langle F \circ \cdot\,\mathtt{=}\, e_2\,, a_1\rangle \to \langle F \circ a_1\,\mathtt{=}\,\cdot\,, e_2\rangle$$

$$\langle F \circ a_1\,\mathtt{=}\,\cdot\,, a_2\rangle \to \langle F\,,\mathtt{T}\rangle \quad \text{if } a_1 = a_2 \qquad \langle F \circ a_1\,\mathtt{=}\,\cdot\,, a_2\rangle \to \langle F\,,\mathtt{F}\rangle \quad \text{if } a_1 \neq a_2$$

$$\langle F\,,\mathtt{ex}\, x.\, e\rangle \to \langle F\,,\mathtt{if}\, e[a/x]\,\mathtt{then}\,\mathtt{T}\,\mathtt{else}\,\mathtt{bot}_{\mathtt{bool}}\rangle \quad \text{for all } a \in \mathbb{A}$$

$$\langle F\,,\mathtt{ex}\, x.\, e\rangle \to \langle F\,,\mathtt{case}_{\,\mathtt{bool}}(e[a_1/x],\ldots,e[a_n/x])\,\mathtt{of}\,((\mathtt{F},\ldots,\mathtt{F}) \to \mathtt{F} \mid \_ \to \mathtt{bot}_{\mathtt{bool}})\rangle$$
where $\{a_1,\ldots,a_n\} = \mathrm{fn}\, e \cup \{b\}$ and $b \# e$

$$\langle F\,,\mathtt{the}\, x.\, e\rangle \to \langle F\,, \nu b.\,\mathtt{case}_{\,\mathtt{bool}}(e[a_1/x], e[a_2/x],\ldots, e[a_n/x], e[b/x])\mathtt{of}\; ((\mathtt{T},\mathtt{F},\ldots,\mathtt{F},\mathtt{F}) \to a_1 \mid (\mathtt{F},\mathtt{T},\ldots,\mathtt{F},\mathtt{F}) \to a_2 \mid \ldots \mid (\mathtt{F},\mathtt{F},\ldots,\mathtt{T},\mathtt{F}) \to a_n \mid \_ \to \mathtt{bot}_{\mathtt{name}})\rangle$$
where $\{a_1,\ldots,a_n\} = \mathrm{fn}\, e$ and $b \# e$

Fig. 12. Frame-stack operational semantics of $\lambda\nu$-PCF$^+$

n), whose proof uses $\langle F\,, e\rangle \to \langle F'\,, e'\rangle \wedge F'[e'] \Downarrow c \Rightarrow F[e] \Downarrow c$ (proved by case analysis of $\to$) and $F[e] \Downarrow c \Leftrightarrow e \Downarrow c' \wedge F[c'] \Downarrow c$ (proved by induction on F). □

The frame-stack semantics specialises to $\lambda\nu$-PCF+the and $\lambda\nu$-PCF+ex in the obvious way (by dismissing the rules for $\mathtt{ex}\, x.\, e$ or $\mathtt{the}\, x.\, e$), and Proposition A.5 remains to be valid for those languages. The fine-grainedness of the frame-stack rules allows us to formalise the intuitive justification of (27) in Section 6.

LEMMA A.6. *Define for* $n \in \mathbb{N}$ *and* $a \in \mathbb{A}$

$$\begin{aligned}\mathtt{eqBot}_{a,n} &\triangleq \lambda x : \mathtt{name} \to \nu b_1. \ldots \nu b_n.\ \mathtt{if}(x = a)\ \mathtt{then}\ \mathtt{T}\ \mathtt{else}\ \mathtt{bot}_{\mathtt{bool}}\\ EqBot_a &\triangleq \{\mathtt{eqBot}_{a,n} \mid n \in \mathbb{N}\}\\ \mathtt{kBot}_n &\triangleq \lambda x : \mathtt{name} \to \nu b_1. \ldots \nu b_n.\, \mathtt{bot}_{\mathtt{bool}}\\ KBot &\triangleq \{\mathtt{kBot}_n \mid n \in \mathbb{N}\}\end{aligned}$$

where $b_1, \ldots, b_n$ *are distinct and fresh for* a*. Note that the definitions of* $\mathtt{eqBot}_{a,n}$ *and* $\mathtt{kBot}_n$ *are independent of the choice of* $b_1, \ldots, b_n$ *due to* α*-equivalence. It then holds that*

$$\begin{aligned}&(\forall i \in \mathbb{N})(\forall a \in \mathbb{A})(\forall j \in \mathbb{N})(\forall x_1, \ldots, x_j \in \mathbb{V})(\forall e_1, \ldots, e_j \in EqBot_a)(\forall \gamma \in \{\mathtt{bool}, \mathtt{nat}, \mathtt{name}\})\\ &(\forall \langle F\,, e\rangle \in \mathrm{Config}^{\lambda\nu\text{-}\mathsf{PCF}+\mathtt{the}})(\forall c \in \mathrm{Can}^{\lambda\nu\text{-}\mathsf{PCF}+\mathtt{the}})\ a \notin \mathrm{fn}(\langle F\,, e\rangle)\ \wedge\\ &x_1 : \mathtt{name} \to \mathtt{bool}, \ldots, x_j : \mathtt{name} \to \mathtt{bool} \vdash F[e] : \gamma\ \wedge\ \langle F\,, e\rangle[e_1/x_1, \ldots, e_j/x_j] \to^i \langle \mathrm{Id}\,, c\rangle\\ &\ \Rightarrow\ (\forall e'_1, \ldots, e'_j \in KBot)\ \langle F\,, e\rangle[e'_1/x_1, \ldots, e'_j/x_j] \to^* \langle \mathrm{Id}\,, c\rangle\,.\end{aligned}$$

PROOF. The proof works by induction on i. The $i = 0$ case is obvious, because then $F = \mathrm{Id}$ and $e = c$. For the inductive step we proceed by case distinction on the first transition. Special care needs to be taken in the case where e is a variable, as this is where we need the generality of having multiple substitutions and name restrictions in the proof statement. □

It is straight-forward to see that (56) is an instance of Lemma A.6. Similarly, we can use the frame-stack semantics to give a lemma that proves (57).

LEMMA A.7. *Define for any* $n \in \mathbb{N}$

$$\begin{aligned}\mathtt{eq}_{a,n} &\triangleq \lambda x : \mathtt{name} \to \nu b_1. \ldots \nu b_n.\, (x = a)\\ Eq_a &\triangleq \{\mathtt{eq}_{a,n} \mid n \in \mathbb{N}\}\end{aligned}$$

where $b_1, \ldots, b_n$ *are distinct and not equal to* a*. Note that the definition of* $\mathtt{eq}_{a,n}$ *is independent of the choice of* $b_1, \ldots, b_n$ *due to* α*-equivalence. It follows that:*

$$\begin{aligned}&(\forall i \in \mathbb{N})(\forall a \in \mathbb{A})(\forall j \in \mathbb{N})(\forall x_1, \ldots, x_j \in \mathbb{V})(\forall e_1, \ldots, e_j \in Eq_a)(\forall \langle F\,, e\rangle \in \mathrm{Config}^{\lambda\nu\text{-}\mathsf{PCF}+\mathtt{ex}})\\ &(\forall a' \in \mathbb{A})\ x_1 : \mathtt{name} \to \mathtt{bool}, \ldots, x_j : \mathtt{name} \to \mathtt{bool} \vdash F[e] : \mathtt{name}\ \wedge\ a \notin \mathrm{fn}(\langle F\,, e\rangle)\ \wedge\\ &\langle F\,, e\rangle[e_1/x_1, \ldots, e_j/x_j] \to^i \langle \mathrm{Id}\,, a'\rangle\ \Rightarrow\ a \neq a'\,.\end{aligned}$$

PROOF. We proceed by induction on i and a case distinction on the first transition in the inductive step. The argument is very similar to the one in the proof of Lemma A.6, it only simplifies in many cases due to the simpler induction hypothesis. □

B. PROOF OF LEMMA 8.10

We suppose given $\tau \in \mathrm{Typ}$, $u \in \mathrm{K}[\![\tau]\!]$, $A \subseteq_{\mathrm{f}} \mathbb{A}$ and a $\lambda\nu$-PCF^+-expression $\emptyset \vdash e : \tau \to \mathtt{bool}$ such that $[\![e]\!] = (u \searrow \mathrm{true})$. We have to find $\emptyset \vdash e' : \tau \to \mathtt{bool}$ such that

$$[\![e']\!] = \bigsqcup \mathrm{hull}_A\{(u \searrow \mathrm{true})\}\,. \tag{58}$$

Let Perm be the set of all finite permutations on $\mathbb{A}$, and define for each $f \in \mathrm{Perm} \to 2_\perp$ the analogue of (11) for permutations:

$$exists_{\mathrm{Perm}} f \ \triangleq\ \begin{cases}\mathrm{true} & \text{if } (\exists \pi \in \mathrm{Perm})\ f\,\pi = \mathrm{true}\\ \mathrm{false} & \text{if } (\forall \pi \in \mathrm{Perm})\ f\,\pi = \mathrm{false}\\ \perp & \text{otherwise.}\end{cases} \tag{59}$$

This allows us to characterise $\bigsqcup \mathrm{hull}_A\{(u \searrow \mathrm{true})\}$ by

$$\bigsqcup \mathrm{hull}_A\{(u \searrow \mathrm{true})\} = \lambda d \in [\![\tau]\!] \to \begin{cases} \mathrm{true} & \text{if } (\exists \pi \in \mathrm{Perm})\, \pi \# A \wedge \pi \cdot u \sqsubseteq d \\ \bot & \text{otherwise} \end{cases}$$
$$= \lambda d \in [\![\tau]\!] \to \mathit{exists}_{\mathrm{Perm}}(\lambda \pi \in \mathrm{Perm} \to \pi \# A \wedge (\pi \cdot u \searrow \mathrm{true})\, d)\,. \quad (60)$$

Notation B.1. We write vectors of atomic names as $\vec{a} = a_1, \ldots, a_n$ and define the swapping of two vectors of the same length by

$$(\vec{a}\ \vec{b}) \triangleq (a_1\ b_1) \circ \cdots \circ (a_n\ b_n).$$

Let $\mathbb{A}^{\#n}$ be the set of vectors of length n that contain distinct atomic names. When convenient we confuse notation for vectors and sets, and write for example $\mathrm{supp}\, x = \vec{a}$.

The quantification over permutations and the permutation action in (60) can be encoded with quantification over atomic names and swappings:

LEMMA B.2. *Suppose $A \subseteq_{\mathrm{f}} \mathbb{A}$, $\pi \in \mathrm{Perm}$ and $\pi \# A$. For any element u of a nominal set, suppose $\mathrm{supp}\, u - A = \vec{a}$ with $\vec{a} \in \mathbb{A}^{\#n}$, and let $b_i = \pi(a_i)$ for $i = 1, \ldots, n$. Note that $\vec{b} \in \mathbb{A}^{\#n}$ (because π is a permutation) and $\vec{b} \# A$ (because $\pi \# A$ and $a_i \notin A$). Picking any $\vec{c} \in \mathbb{A}^{\#n}$ with $\vec{c} \# A, \vec{a}, \vec{b}$, we get $\pi \cdot u = (\vec{b}\ \vec{c}) \circ (\vec{a}\ \vec{c}) \cdot u$.*

PROOF. It is easy to check that $(\pi^{-1} \circ (\vec{b}\ \vec{c}) \circ (\vec{a}\ \vec{c}))\, a = a$ holds for all $a \in \mathrm{supp}\, u$. Then the defining property (4) of $\mathrm{supp}\, u$ gives us $(\pi^{-1} \circ (\vec{b}\ \vec{c}) \circ (\vec{a}\ \vec{c})) \cdot u = u$, from which the desired property follows by applying π to both sides of the equation. □

Applying this Lemma B.2 to (60), together with the semantics of locally scoped names, it follows that (58) holds with

$$e' \triangleq \lambda x : \tau \to \mathtt{ex}\, y_1.\ \ldots \mathtt{ex}\, y_n.\, (\mathtt{distinct}\, \vec{y})\ \mathtt{and}\ (\vec{y}\ \mathtt{freshfor}\ A)$$
$$\mathtt{and}\ \nu b_1.\ \ldots \nu b_n.\, (((\vec{y} \rightleftharpoons \vec{b})\, (\vec{a} \rightleftharpoons \vec{b})\, e)\, x)$$

where $[\![e]\!] = (u \searrow \mathrm{true})$ as above, $\vec{a} \triangleq \mathrm{supp}\ u - A$ and $\vec{b}$ are distinct atomic names satisfying $\vec{b} \# u, A$. For better readability, we used some syntactic sugar in the above expression:

— swapping vectors of expressions, $(\vec{e} \rightleftharpoons \vec{e'}) \triangleq (e_1 \rightleftharpoons e'_1)\ \ldots (e_n \rightleftharpoons e'_n)$
— boolean conjunction, $e_1\ \mathtt{and}\ e_2 \triangleq \mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ \mathtt{F}$
— the expression $\mathtt{distinct}\, \vec{e}$ stands for the expression that tests if all atomic names that $\vec{e}$ evaluates to are distinct
— $\vec{e}\ \mathtt{freshfor}\ A$ expresses that none of the expressions in the vector $\vec{e}$ evaluate to elements of the finite set $A \subseteq_{\mathrm{f}} \mathbb{A}$.

The last two constructs can be defined for any length of $\vec{e}$ by using conditionals and name equality tests.

C. PROOF OF LEMMA 8.11

Suppose we are given $\tau \in \mathrm{Typ}$ such that

$$(\forall v, v' \in \mathrm{K}[\![\tau]\!])\, v \not\asymp v' \Rightarrow (\exists e \in \mathrm{Exp})\, \emptyset \vdash e : \tau \to \mathtt{bool} \wedge [\![e]\!] = (v \searrow \mathrm{true}) \sqcup (v' \searrow \mathrm{false}) \quad (61)$$

where $v \not\asymp v'$ is notation for v and v' not having an upper bound in $[\![\tau]\!]$. We also suppose given $u, u' \in \mathrm{K}[\![\tau]\!]$ and $A \subseteq_{\mathrm{f}} \mathbb{A}$ with

$$(\forall \pi \in \mathrm{Perm})\, \pi \# A \Rightarrow u \not\asymp \pi \cdot u'\,. \quad (62)$$

We have to find $\emptyset \vdash e : \tau \to \mathtt{bool}$ such that

$$[\![e]\!] = \bigsqcup \mathrm{hull}_A\{(u \searrow \mathrm{true}), (u' \searrow \mathrm{false})\}\,. \tag{63}$$

Without loss of generality we may assume

$$(\mathrm{supp}\, u \cap \mathrm{supp}\, u') - A = \emptyset \tag{64}$$

because otherwise we can take $u'' \triangleq (\vec{a}\ \vec{b})\cdot u$ instead of u', where $\vec{a} \triangleq (\mathrm{supp}\, u \cap \mathrm{supp}\, u') - A$ and $\vec{b}$ are some distinct and fresh atomic names. (We use notation for swapping lists of atomic names as in Notation B.1.) It is easy to show that $(\mathrm{supp}\, u \cap \mathrm{supp}\, u'') - A = \emptyset$. We can replace u' with u'' because $(\vec{a}\ \vec{b}) \# A$ implies that $\bigsqcup \mathrm{hull}_A\{(u \searrow \mathrm{true}), (u' \searrow \mathrm{false})\} = \bigsqcup \mathrm{hull}_A\{(u \searrow \mathrm{true}), (u'' \searrow \mathrm{false})\}$ and that (61) as well as (62) hold for u''.

Note that (62) entails that whenever there is a $\pi \# A$ with $\pi \cdot u \sqsubseteq x$, then for all $\pi' \# A$ it is the case that $\pi' \cdot u \not\sqsubseteq x$ (and symmetrically with u and u' interchanged). Therefore we get:

$$\begin{aligned}
&\bigsqcup \mathrm{hull}_A\{(u \searrow \mathrm{true}), (u' \searrow \mathrm{false})\}\\
&= \lambda d \in [\![\tau]\!] \to \begin{cases} \mathrm{true} & \text{if } (\exists \pi \in \mathrm{Perm})\, \pi \# A \wedge \pi \cdot u \sqsubseteq d \\ \mathrm{false} & \text{if } (\exists \pi' \in \mathrm{Perm})\, \pi' \# A \wedge \pi' \cdot u' \sqsubseteq d \\ \bot & \text{otherwise} \end{cases}\\
&= \lambda d \in [\![\tau]\!] \to\\
&\qquad \begin{cases} \mathrm{true} & \text{if } (\exists \pi \in \mathrm{Perm})\, \pi \# A \wedge \pi \cdot u \sqsubseteq d \wedge (\forall \pi' \in \mathrm{Perm})\, \pi' \# A \Rightarrow \pi' \cdot u' \not\sqsubseteq d \\ \mathrm{false} & \text{if } (\exists \pi' \in \mathrm{Perm})\, \pi' \# A \wedge \pi' \cdot u' \sqsubseteq d \wedge (\forall \pi \in \mathrm{Perm})\, \pi \# A \Rightarrow \pi \cdot u \not\sqsubseteq d \\ \bot & \text{otherwise} \end{cases}\\
&= \lambda d \in [\![\tau]\!] \to \mathit{exists}_{\mathrm{Perm}}(\lambda \pi \in \mathrm{Perm} \to \pi \# A \wedge \mathit{all}_{\mathrm{Perm}}(\lambda \pi' \in \mathrm{Perm} \to \pi' \# A\\
&\qquad \Rightarrow ((\pi \cdot u \searrow \mathrm{true}) \sqcup (\pi' \cdot u' \searrow \mathrm{false}))\, d))
\end{aligned} \tag{65}$$

where

$$\mathit{all}_{\mathrm{Perm}} f \ \triangleq\ \begin{cases} \mathrm{true} & \text{if } (\forall \pi \in \mathrm{Perm})\ f\,\pi = \mathrm{true} \\ \mathrm{false} & \text{if } (\exists \pi \in \mathrm{Perm})\ f\,\pi = \mathrm{false} \\ \bot & \text{otherwise} \end{cases}$$

is the dual of (59).

As in Appendix B, we can replace the quantifications over π and π' in (65) by multiple quantifications over atomic names. Suppose $\mathrm{supp}\, u - A$ consists of the distinct atomic names $\vec{b} = (b_1, \ldots, b_n) \in \mathbb{A}^{\#n}$ and $\mathrm{supp}\, u' - A$ consists of the distinct atomic names $\vec{c} = (c_1, \ldots, c_m) \in \mathbb{A}^{\#m}$; so from (64) we have $\vec{b} \# \vec{c}$. Given $\pi, \pi' \# A$, as in Lemma B.2 we have

$$\begin{aligned}
\pi \cdot u &= (\vec{b'}\ \vec{d}) \circ (\vec{b}\ \vec{d}) \cdot u\\
\pi' \cdot u' &= (\vec{c'}\ \vec{d'}) \circ (\vec{c}\ \vec{d'}) \cdot u'
\end{aligned}$$

with $\vec{b'} = \pi \cdot \vec{b}$, $\vec{c'} = \pi' \cdot \vec{c}$ and $\vec{d}, \vec{d'}$ chosen suitably fresh. Therefore in view of (65), to solve (63) we can take e to be

$$\begin{aligned}
e \triangleq\ & \lambda x : \tau \to \mathtt{ex}\, x_1.\ \ldots \mathtt{ex}\, x_n.\, (\mathtt{distinct}\, \vec{x})\ \mathtt{and}\ (\vec{x}\ \mathtt{freshfor}\ A)\\
& \mathtt{and}\ \mathtt{not}\, (\mathtt{ex}\, y_1.\ \ldots \mathtt{ex}\, y_m.\, (\mathtt{distinct}\, \vec{y})\ \mathtt{and}\ (\vec{y}\ \mathtt{freshfor}\ A)\ \mathtt{and}\ \mathtt{not}(e'\, x))
\end{aligned} \tag{66}$$

provided we can find an expression e' with typing $x_1 : \mathtt{name}, \ldots, x_n : \mathtt{name}, y_1 : \mathtt{name}, \ldots, y_m : \mathtt{name} \vdash e' : \tau \to \mathtt{bool}$ that satisfies for each valuation ρ that

$$[\![e']\!]\rho = (([\![\vec{x}]\!]\rho\ \vec{d}) \circ (\vec{b}\ \vec{d}) \cdot u \searrow \mathrm{true}) \sqcup (([\![\vec{y}]\!]\rho\ \vec{d'}) \circ (\vec{c}\ \vec{d'}) \cdot u' \searrow \mathrm{false}) \tag{67}$$

for some/any fresh $\vec{d}, \vec{d'}$. In (66) we use the syntactic sugar from Appendix B as well as $\texttt{not}\, e \triangleq \texttt{if}\, e\, \texttt{then F else T}$.

Giving such an e' might seem easy at first, since in view of (61) and (62), $(\pi \cdot u \searrow \mathrm{true}) \sqcup (\pi' \cdot u' \searrow \mathrm{false})$ is $\lambda\nu$-PCF-definable for any particular $\pi, \pi' \# A$. However, it is not so easy: the problem is that e' has $\vec{x}, \vec{y}$ as free variables and we need to be parametric with respect to whatever atomic names those free variables get assigned by a given valuation ρ. Specifically, to define e' we need to consider all ways in which atomic names assigned to $\vec{x}$ and $\vec{y}$ may overlap. Fortunately each way corresponds to a partial bijection from $\{1, \ldots, n\}$ to $\{1, \ldots, m\}$ and there are only finitely many of them, N say. Thus we take

$$\begin{aligned} e' \triangleq\ & \texttt{if not}(x_1 = y_1) \texttt{ and not}(x_1 = y_2) \texttt{ and } \ldots \texttt{ and not}(x_n = y_m) \texttt{ then } e_1 \\ & \texttt{else if } x_1 = y_1 \texttt{ and not}(x_2 = y_2) \texttt{ and } \ldots \texttt{ and not}(x_n = y_m) \texttt{ then } e_2 \\ & \vdots \\ & \texttt{else if } x_1 = y_1 \texttt{ and } x_2 = y_2 \texttt{ and } \ldots \texttt{ and } x_n = y_m \texttt{ then } e_N \\ & \texttt{else } \lambda(x : \tau) \to \texttt{F} \end{aligned} \tag{68}$$

and show how to define the expressions $e_1, \ldots, e_N$ in such a way that for each ρ, $[\![e']\!]\rho = [\![e_i]\!]\rho$ for the i corresponding to the overlap conditions ρ induces between $\vec{x}$ and $\vec{y}$. Then (67) holds because the `if`-clauses in (68) are exhaustive; in particular the $\lambda(x : \tau) \to \texttt{F}$ case will never be reached. Let f_i be the partial bijection corresponding to e_i, and define $\vec{b_i} \triangleq \{b_j \mid j \in \mathrm{dom} f_i\}$, $\vec{c_i} \triangleq \{c_{f_i(j)} \mid j \in \mathrm{dom} f_i\}$, $\vec{c'_i} \triangleq \{c_j \mid j \notin \mathrm{im} f_i\}$, $\vec{x_i} \triangleq \{x_j \mid j \in \mathrm{dom} f_i\}$, $\vec{y_i} \triangleq \{y_{f_i(j)} \mid j \in \mathrm{dom} f_i\}$, $\vec{y'_i} \triangleq \{y_j \mid j \notin \mathrm{im} f_i\}$. By (61) let $\emptyset \vdash e'_i : \tau \to \texttt{bool}$ be given such that $[\![e'_i]\!] = (u \searrow \mathrm{true}) \sqcup ((\vec{c_i}\ \vec{b_i}) \cdot u' \searrow \mathrm{false})$. This allows us to define

$$e_i \triangleq \nu d_1. \ldots \nu d_n. \nu d'_1. \ldots \nu d'_{m-|\mathrm{dom} f_i|}. (\vec{y'_i} \rightleftharpoons \vec{d'})\, (\vec{x} \rightleftharpoons \vec{d})\, (\vec{c'_i} \rightleftharpoons \vec{d'})\, (\vec{b} \rightleftharpoons \vec{d})\, e'_i \tag{69}$$

where $\vec{d}, \vec{d'}$ consist of distinct atomic names that satisfy $\vec{d}, \vec{d'} \# A, \vec{b}, \vec{c}$; through α-renaming they may also be chosen to satisfy $\vec{d}, \vec{d'} \# [\![\vec{x}]\!]\rho, [\![\vec{y}]\!]\rho$ once we are given a particular valuation ρ. Define also $\vec{d_i} = \{d_j \mid j \in \mathrm{dom} f_i\}$. Finally, with $\pi \cdot ((u \searrow \mathrm{true}) \sqcup (u' \searrow \mathrm{false})) = (\pi \cdot u \searrow \mathrm{true}) \sqcup (\pi \cdot u' \searrow \mathrm{false})$, property (31),

$$\begin{aligned} & ([\![\vec{y'_i}]\!]\rho\ \vec{d'}) \circ ([\![\vec{x}]\!]\rho\ \vec{d}) \circ (\vec{c'_i}\ \vec{d'}) \circ (\vec{b}\ \vec{d}) \cdot u \\ &= ([\![\vec{y'_i}]\!]\rho\ \vec{d'}) \circ ([\![\vec{x}]\!]\rho\ \vec{d}) \circ (\vec{b}\ \vec{d}) \cdot u && \text{as } (\vec{c'_i}\ \vec{d'}) \# \vec{d} \\ &= ([\![\vec{x}]\!]\rho\ \vec{d}) \circ (\vec{b}\ \vec{d}) \cdot u && \text{as } ([\![\vec{y'_i}]\!]\rho\ \vec{d'}) \# [\![\vec{x}]\!]\rho \end{aligned}$$

and

$$\begin{aligned} & ([\![\vec{y'_i}]\!]\rho\ \vec{d'}) \circ ([\![\vec{x}]\!]\rho\ \vec{d}) \circ (\vec{c'_i}\ \vec{d'}) \circ (\vec{b}\ \vec{d}) \circ (\vec{c_i}\ \vec{b_i}) \cdot u' \\ &= ([\![\vec{y'_i}]\!]\rho\ \vec{d'}) \circ ([\![\vec{x}]\!]\rho\ \vec{d}) \circ (\vec{c'_i}\ \vec{d'}) \circ (\vec{c_i}\ \vec{d_i}) \circ (\vec{b}\ \vec{d}) \cdot u' && \text{as } \pi \circ (a\ b) = (\pi\ a\ \pi\ b) \circ \pi \\ &= ([\![\vec{y'_i}]\!]\rho\ \vec{d'}) \circ ([\![\vec{x}]\!]\rho\ \vec{d}) \circ (\vec{c'_i}\ \vec{d'}) \circ (\vec{c_i}\ \vec{d_i}) \cdot u' && \text{as } (\vec{b}\ \vec{d}) \# u' \\ &= ([\![\vec{y'_i}]\!]\rho\ \vec{d'}) \circ ([\![\vec{x_i}]\!]\rho\ \vec{d_i}) \circ (\vec{c'_i}\ \vec{d'}) \circ (\vec{c_i}\ \vec{d_i}) \cdot u' && \text{as } \vec{x} = \vec{x_i} \cup \vec{x'_i} \\ &= ([\![\vec{y'_i}]\!]\rho\ \vec{d'}) \circ ([\![\vec{y_i}]\!]\rho\ \vec{d_i}) \circ (\vec{c'_i}\ \vec{d'}) \circ (\vec{c_i}\ \vec{d_i}) \cdot u' && \text{as } [\![\vec{x_i}]\!]\rho = [\![\vec{y_i}]\!]\rho \\ &= ([\![\vec{y}]\!]\rho\ \vec{d''}) \circ (\vec{c}\ \vec{d''}) \cdot u' && \text{as } \vec{y} = \vec{y_i} \cup \vec{y'_i}, \text{ with } \vec{d''} = \vec{d'} \cup \vec{d_i} \end{aligned}$$

we obtain that $[\![e_i]\!]\rho = [\![e']\!]\rho$ as required.

This concludes the proof of Lemma 8.11. Since the last part of the proof is combinatorially complicated, we illustrate the constructions for a simple instance.

Example C.1. Suppose we are in the special case of Lemma 8.11 where $A = \{a\}$, $\operatorname{supp} u = \{b_1, b_2\}$ and $\operatorname{supp} u' = \{c_1, c_2\}$. In this setting, the expression e satisfying $[\![e]\!] = \bigsqcup \operatorname{hull}_{\{a\}}\{(u \searrow \mathrm{true}), (u' \searrow \mathrm{false})\}$ is defined by

$$\begin{aligned} e \triangleq \lambda x : \tau \to \mathtt{ex}\, x_1.\, \mathtt{ex}\, x_2.\, \mathtt{not}(x_1 = x_2)\ \mathtt{and}\ \mathtt{not}(x_1 = a)\ \mathtt{and}\ \mathtt{not}(x_2 = a) \\ \mathtt{and}\ \mathtt{not}(\mathtt{ex}\, y_1.\, \mathtt{ex}\, y_2.\, \mathtt{not}(x_1 = x_2)\ \mathtt{and}\ \mathtt{not}(y_1 = a)\ \mathtt{and}\ \mathtt{not}(y_2 = a)\ \mathtt{and}\ \mathtt{not}(e'\, x)) \end{aligned}$$

where the expression e' reads

$$\begin{aligned} e' \triangleq\ & \mathtt{if}\ \mathtt{not}(x_1 = y_1)\ \mathtt{and}\ \mathtt{not}(x_2 = y_2)\ \mathtt{and}\ \mathtt{not}(x_1 = y_2)\ \mathtt{and}\ \mathtt{not}(x_n = y_m)\ \mathtt{then}\ e_1 \\ & \mathtt{else\ if}\ x_1 = y_1\ \mathtt{and}\ \mathtt{not}(x_2 = y_2)\ \mathtt{then}\ e_2\ \mathtt{else\ if}\ \mathtt{not}(x_1 = y_1)\ \mathtt{and}\ x_2 = y_2\ \mathtt{then}\ e_3 \\ & \mathtt{else\ if}\ x_1 = y_2\ \mathtt{and}\ \mathtt{not}(x_2 = y_1)\ \mathtt{then}\ e_4\ \mathtt{else\ if}\ \mathtt{not}(x_1 = y_2)\ \mathtt{and}\ x_2 = y_1\ \mathtt{then}\ e_5 \\ & \mathtt{else\ if}\ x_1 = y_1\ \mathtt{and}\ x_2 = y_2\ \mathtt{then}\ e_6\ \mathtt{else\ if}\ x_1 = y_2\ \mathtt{and}\ x_2 = y_1\ \mathtt{then}\ e_7 \\ & \mathtt{else}\ \lambda x : \tau \to \mathtt{F}. \end{aligned}$$

Compare this to (66) and (68). Let us now define e_3 explicitly as in (69). The corresponding partial bijection is $f_3 = \{2 \mapsto 2\}$ and by (61) we may assume the existence of e_3' with $[\![e_3']\!] = (u \searrow \mathrm{true}) \sqcup ((c_2\ b_2) \cdot u' \searrow \mathrm{false})$. Then we can define

$$e_3 \triangleq \nu d_1.\, \nu d_2.\, \nu d_1'.\, (y_1 \rightharpoonup d_1')\, (x_1 \rightharpoonup d_1)\, (x_2 \rightharpoonup d_2)\, (c_1 \rightharpoonup d_1')\, (b_1 \rightharpoonup d_1)\, (b_2 \rightharpoonup d_2)\, e_3'$$

for which $[\![e_3]\!]\rho = (([\![x_1]\!]\rho\ d_1) \circ ([\![x_2]\!]\rho\ d_2) \circ (b_1\ d_1) \circ (b_2\ d_2) \cdot u \searrow \mathrm{true}) \sqcup (([\![y_1]\!]\rho\ d_1') \circ ([\![y_2]\!]\rho\ d_2) \circ (c_1\ d_1') \circ (c_2\ d_2) \cdot u' \searrow \mathrm{false})$ holds for any ρ under the conditions that $[\![x_1]\!]\rho \neq [\![y_1]\!]\rho$ and $[\![x_2]\!]\rho = [\![y_2]\!]\rho$.

D. PROOF OF THEOREM 11.2

The overall structure of the proof of Theorem 11.2 is the same as in Theorem 7.5. We extend the logical relation to cover the type of λ-terms and abstraction types:

$$\begin{aligned} d \lhd_{\mathtt{term}} e &\triangleq d = \bot \vee (\exists c)\ e \Downarrow c \wedge [\![c]\!] = d \\ d \lhd_{\delta\,\tau} e &\triangleq (И a)\ d \mathbin{@} a \lhd_\tau e \mathbin{@} a. \end{aligned}$$

In the second clause we use the *freshness quantifier* $(И a)$ of nominal logic [Pitts 2013, Section 3.2]; thus $d \lhd_{\delta\,\tau} e$ holds if and only if $d \mathbin{@} a \lhd_\tau e \mathbin{@} a$ holds for some $a \# (d, e)$, or equivalently, for any $a \# (d, e)$.

The proof that this logical relation is closed under restriction, abstraction and concretion

$$d \lhd_\tau e \ \Rightarrow\ (\forall a)\ a \backslash d \lhd_\tau \nu a.\, e \tag{70}$$

$$d \lhd_\tau e \Rightarrow (\forall a)\ \langle a \rangle d \lhd_{\delta\,\tau} \alpha a.\, e \tag{71}$$

$$d \lhd_{\delta\,\tau} e \Rightarrow (\forall a)\ d \mathbin{@} a \lhd_\tau e \mathbin{@} a \tag{72}$$

is not straightforward and occupies the rest of this appendix. Using these three properties one can then prove the fundamental property of the logical relation as in Theorem 7.5, and computational adequacy follows from that in the usual way.

To prove (70)–(72) we introduce a weaker form of 'Kleene preorder' between expressions:

Definition D.1. For every PNA type $\tau \in \mathrm{Typ}$, let $\mathrm{Exp}(\tau) \triangleq \{e \in \mathrm{Exp} \mid \emptyset \vdash e : \tau\}$ and $\mathrm{Can}(\tau) \triangleq \{c \in \mathrm{Can} \mid \emptyset \vdash c : \tau\}$, and define the *weak Kleene preorders*

$$\begin{aligned} \leq_\tau^{\mathrm{wk}} &\subseteq \mathrm{Exp}(\tau) \times \mathrm{Exp}(\tau) \\ \leq_\tau^{\mathrm{wk}} &\subseteq \mathrm{Can}(\tau) \times \mathrm{Can}(\tau) \end{aligned}$$

by simultaneously structural recursion over τ:

$$
\begin{aligned}
e \leq^{\mathrm{wk}}_{\tau} e' &\triangleq (\forall c)\, e \Downarrow c \;\Rightarrow\; (\exists c')\, e' \Downarrow c' \;\wedge\; c \leq^{\mathrm{wk}}_{\tau} c' \\
c \leq^{\mathrm{wk}}_{\gamma} c' &\triangleq c = c' \qquad (\gamma \in \{\mathtt{name}, \mathtt{bool}, \mathtt{nat}, \mathtt{term}\}) \\
c \leq^{\mathrm{wk}}_{\tau_1 \times \tau_2} c' &\triangleq (\exists e_1, e_2, e_1', e_2')\, c = (e_1\,, e_2) \;\wedge\; c' = (e_1'\,, e_2') \;\wedge\; e_1 \leq^{\mathrm{wk}}_{\tau_1} e_1' \;\wedge\; e_2 \leq^{\mathrm{wk}}_{\tau_2} e_2' \\
c \leq^{\mathrm{wk}}_{\tau_1 \to \tau_2} c' &\triangleq (\exists x, e, e')\, c = \lambda x : \tau_1 \to e \;\wedge\; c' = \lambda x : \tau_1 \to e' \\
&\qquad \wedge\; (\forall e_1 \in \mathrm{Exp}(\tau_1))\, e[e_1/x] \leq^{\mathrm{wk}}_{\tau_2} e'[e_1/x] \\
c \leq^{\mathrm{wk}}_{\delta\,\tau} c' &\triangleq (\mathit{N}a)(\exists c_1, c_1')\, c = \alpha a.\, c_1 \;\wedge\; c' = \alpha a.\, c_1' \;\wedge\; c_1 \leq^{\mathrm{wk}}_{\tau} c_1'\,.
\end{aligned}
$$

The next lemma collects together the properties of these preorders that we need.

LEMMA D.2. *The following properties hold:*

$$((\forall c)\, e \Downarrow c \Rightarrow e' \Downarrow c) \;\Rightarrow\; e \leq^{\mathrm{wk}}_{\tau} e'. \tag{73}$$

$$c \leq^{\mathrm{wk}}_{\tau} c' \;\wedge\; a \backslash\backslash c := c_1 \;\Rightarrow\; (\exists c_1')\, a \backslash\backslash c' := c_1' \;\wedge c_1 \leq^{\mathrm{wk}}_{\tau} c_1' \tag{74}$$

$$e \leq^{\mathrm{wk}}_{\tau} e' \;\Rightarrow\; \nu a.\, e \leq^{\mathrm{wk}}_{\tau} \nu a.\, e' \tag{75}$$

$$e \leq^{\mathrm{wk}}_{\delta\,\tau} e' \;\Rightarrow\; (\forall a)\, e \mathbin{@} a \leq^{\mathrm{wk}}_{\tau} e' \mathbin{@} a \tag{76}$$

$$e \leq^{\mathrm{wk}}_{\tau_1 \times \tau_2} e' \;\Rightarrow\; \mathtt{fst}\; e \leq^{\mathrm{wk}}_{\tau_1} \mathtt{fst}\; e' \;\wedge\; \mathtt{snd}\; e \leq^{\mathrm{wk}}_{\tau_2} \mathtt{snd}\; e' \tag{77}$$

$$e \leq^{\mathrm{wk}}_{\tau_1 \to \tau_2} e' \;\Rightarrow\; (\forall e_1 \in \mathrm{Exp}(\tau_1))\, e\, e_1 \leq^{\mathrm{wk}}_{\tau_2} e\, e_1 \tag{78}$$

$$a' \backslash\backslash c := c_1 \;\wedge\; a \backslash\backslash c_1 := c_2 \;\Rightarrow\; (\exists c_1', c_2')\, a \backslash\backslash c := c_1' \;\wedge a' \backslash\backslash c_1' := c_2' \;\wedge\; c_2 \leq^{\mathrm{wk}}_{\tau} c_2' \tag{79}$$

$$\nu a.\, \nu a'.\, e \leq^{\mathrm{wk}}_{\tau} \nu a'.\, \nu a.\, e \tag{80}$$

$$a \# c \;\Rightarrow\; (\exists c')\, a \backslash\backslash c := c' \;\wedge\; c \leq^{\mathrm{wk}}_{\tau} c' \;\wedge\; c' \leq^{\mathrm{wk}}_{\tau} c \tag{81}$$

$$a \# e \;\Rightarrow\; \nu a.\, e \leq^{\mathrm{wk}}_{\tau} e \;\wedge\; e \leq^{\mathrm{wk}}_{\tau} \nu a.\, e \tag{82}$$

$$a \neq a' \;\Rightarrow\; \nu a.\, (e \mathbin{@} a') \leq^{\mathrm{wk}}_{\tau} (\nu a.\, e) \mathbin{@} a' \tag{83}$$

$$\nu a.\, \mathtt{fst}\; e \leq^{\mathrm{wk}}_{\tau} \mathtt{fst}\; (\nu a.\, e) \tag{84}$$

$$\nu a.\, \mathtt{snd}\; e \leq^{\mathrm{wk}}_{\tau} \mathtt{snd}\; (\nu a.\, e) \tag{85}$$

$$a \# e_1 \;\Rightarrow\; \nu a.\, (e\, e_1) \leq^{\mathrm{wk}}_{\tau} (\nu a.\, e)\, e_1\,. \tag{86}$$

PROOF. (73) follows from the definition of $\leq^{\mathrm{wk}}_{\tau}$ and the easily verified fact that $\leq^{\mathrm{wk}}_{\tau}$ is reflexive. (74) can be proved by induction on the structure of τ from the definition of $\leq^{\mathrm{wk}}$; and then (75) and (76) follow from this. The proof of (76) additionally uses that $\leq^{\mathrm{wk}}$ and $\leq^{\mathrm{wk}}$ are equivariant, which is straight-forward to prove. (77) and (78) are direct consequences of the definition of $\leq^{\mathrm{wk}}$. (79) can be proved from the definition of $\leq^{\mathrm{wk}}$ by induction on the structure of τ; and then (80) follow from this. Similarly for and (81) and (82). (83) follows from (79) and (80). (84)–(86) are immediate by using (73). □

COROLLARY D.3.

$$e \leq^{\mathrm{wk}}_{\tau} (\alpha a.\, e) \mathbin{@} a \tag{87}$$

$$a' \# (a, e) \;\Rightarrow\; \nu a'.\, (a\; a') \cdot (e \mathbin{@} a') \leq^{\mathrm{wk}}_{\tau} e \mathbin{@} a \tag{88}$$

PROOF. For showing that (88) implies (87), we use transitivity of $\leq^{\mathrm{wk}}_{\tau}$ and prove $e \leq^{\mathrm{wk}}_{\tau} \nu a'.\, (a\; a') \cdot (\alpha a.\, e \mathbin{@} a') \leq^{\mathrm{wk}}_{\tau} \alpha a.\, e \mathbin{@} a$ for some/any $a' \# a, e$. The second preorder relation is a consequence of (88), whereas the first one can be proved directly.

For (88), suppose $a' \# (a, e)$ and

$$\nu a'.\,(a\ a') \cdot (e \mathbin{@} a') \Downarrow c. \tag{89}$$

We have to show $e \mathbin{@} a \Downarrow c'$, for some c' with $c \leq^{\mathrm{wk}}_{\tau} c'$. But (89) can only hold because for some c_1 we have

$$(a\ a') \cdot (e \mathbin{@} a') \Downarrow c_1 \tag{90}$$

$$a' \backslash\backslash c_1 := c. \tag{91}$$

From (90) we get (by equivariance of $\Downarrow$) that $e \mathbin{@} a' \Downarrow (a\ a') \cdot c_1$; and hence we must have for some $a'' \# (a, a', e, c, c_1)$ and c_2 that

$$e \Downarrow \alpha a''.\, c_2 \tag{92}$$

$$a'' \backslash\backslash (a''\ a') \cdot c_2 := (a\ a') \cdot c_1. \tag{93}$$

Since $a' \# e$, from (92) we get $a' \# c_2$ and hence $a'' \# (a''\ a') \cdot c_2$. Then by (81) and (93) we have $(a\ a') \cdot c_1 \leq^{\mathrm{wk}}_{\tau} (a''\ a') \cdot c_2$ and hence also

$$c_1 \leq^{\mathrm{wk}}_{\tau} (a\ a')(a''\ a') \cdot c_2. \tag{94}$$

Applying (74) to (91) and (94), there exists c' with

$$a' \backslash\backslash (a\ a')(a''\ a') \cdot c_2 := c' \ \wedge\ c \leq^{\mathrm{wk}}_{\tau} c'. \tag{95}$$

Note that since $a' \# c_2$, $\alpha a''.\, c_2 = \alpha a'.\, (a''\ a') \cdot c_2$ and therefore from (92), $e \Downarrow \alpha a'.\, (a''\ a') \cdot c_2$. Combining this with (95) we get $e \mathbin{@} a \Downarrow c'$ and $c \leq^{\mathrm{wk}}_{\tau} c'$, as required. □

We can now complete the proof of properties (70)–(72) of the logical relation:

— (70) follows by structural induction on τ, using (83) for name abstraction types, (84) and (85) for product types, and (86) for function types.

— (71) follows from (87).

— Finally, for (72), suppose $d \lhd_{\delta\,\tau} e$. Given any $a \in \mathbb{A}$, pick $a' \# (a, d, e)$. Then putting $d' \triangleq d \mathbin{@} a'$, we have $d = \langle a' \rangle d'$ and $d \mathbin{@} a = a' \backslash (a\ a') \cdot d'$. By definition of $\lhd_{\delta\,\tau}$ we have $d' \lhd_{\tau} e \mathbin{@} a'$ and hence by equivariance of the logical relation, $(a\ a') \cdot d' \lhd_{\tau} (a\ a') \cdot (e \mathbin{@} a')$. So by (70) and (88), $d \mathbin{@} a = a' \backslash (a\ a') \cdot d' \lhd_{\tau} \nu a'.\, (a\ a') \cdot (e \mathbin{@} a') \leq^{\mathrm{wk}}_{\tau} e \mathbin{@} a$. So it just remains to see that the logical relation is closed under composition with $\leqq^{\mathrm{wk}}$

$$d \lhd_{\tau} e \ \wedge\ e \leqq^{\mathrm{wk}}_{\tau} e' \ \Rightarrow\ d \lhd_{\tau} e'$$

which follows by combining (76)–(78) with the definitions of $\leqq^{\mathrm{wk}}$ and $\lhd$.

REFERENCES

S. Abramsky. 1991. Domain Theory In Logical Form. *Annals of Pure and Applied Logic* 51 (1991), 1–77.

S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, and I. D. B. Stark. 2004. Nominal Games and Full Abstraction for the Nu-Calculus. In *Proc. LICS 2004*. IEEE Computer Society Press, 150–159.

S. Abramsky, R. Jagadeesan, and P. Malacaria. 2000. Full Abstraction for PCF. *Information and Computation* 163, 2 (2000), 409–470.

S. Abramsky and A. Jung. 1994. Domain Theory. In *Handbook of Logic in Computer Science*, S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum (Eds.). Vol. 3. Clarendon Press, 1–168.

M. Bojańczyk, L. Braud, B. Klin, and S. Lasota. 2012. Towards Nominal Computation. In *Proc. POPL 2012*. ACM Press, 401–412.

M. Bojańczyk, B. Klin, and S. Lasota. 2011. Automata with Group Actions. In *Proc. LICS 2011*. IEEE Computer Society Press, 355–364.

M. Bojańczyk and S. Lasota. 2012. A Machine-Independent Characterization of Timed Languages. In *Proc. ICALP 2012, Part II (Lecture Notes in Computer Science)*, A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer (Eds.), Vol. 7392. Springer-Verlag, 92–103.

V. Ciancia and U. Montanari. 2010. Symmetries, Local Names and Dynamic (De)-allocation of Names. *Information and Computation* 208, 12 (2010), 1349–1367.

P.-L. Curien. 2007. Definability and Full Abstraction. In *Computation, Meaning and Logic, Articles dedicated to Gordon Plotkin*, L. Cardelli, M. Fiore, and G. Winskel (Eds.). Electronic Notes in Theoretical Computer Science, Vol. 172. Elsevier, 301–310.

M. Felleisen and R. Hieb. 1992. The Revised Report on the Syntactic Theories of Sequential Control and State. *Theoretical Computer Science* 103 (1992), 235–271.

M. J. Gabbay. 2009. A Study of Substitution, Using Nominal Techniques and Fraenkel-Mostowski Sets. *Theoretical Computer Science* 410, 12–13 (2009), 1159–1189.

M. J. Gabbay. 2011. Foundations of Nominal Techniques: Logic and Semantics of Variables in Abstract Syntax. *Bulletin of Symbolic Logic* 17, 2 (2011), 161–229.

M. J. Gabbay and V. Ciancia. 2011. Freshness and Name-Restriction in Sets of Traces with Names. In *Proc. FOSSACS 2011 (LNCS)*, Vol. 6604. Springer-Verlag, 365–380.

M. J. Gabbay and A. M. Pitts. 2002. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing* 13 (2002), 341–363.

P. Gabriel and F. Ulmer. 1971. *Lokal Präsentierbare Kategorien*. Lecture Notes in Mathematics, Vol. 221. Springer-Verlag.

F. Gadducci, M. Miculan, and U. Montanari. 2006. About Permutation Algebras, (Pre)sheaves and Named Sets. *Higher-Order Symb. Computation* 19 (2006), 283–304.

R. Harper. 2013. *Practical Foundation for Programming Languages*. Cambridge University Press.

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

P. T. Johnstone. 2002. *Sketches of an Elephant, A Topos Theory Compendium, Volumes 1 and 2*. Number 43–44 in Oxford Logic Guides. Oxford University Press.

J. Laird. 2008. A Game Semantics of Names and Pointers. *Annals of Pure and Applied Logic* 151, 2 (2008), 151–169.

D. R. Licata and R. Harper. 2009. A Universe of Binding and Computation. In *Proc. ICFP 2009*. ACM, New York, NY, USA, 123–134.

S. Lösch and A. M. Pitts. 2011. Relating Two Semantics of Locally Scoped Names. In *Proc. CSL 2011*, Vol. 12. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 396–411.

S. Lösch and A. M. Pitts. 2013. Full Abstraction for Nominal Scott Domains. In *Proc. POPL 2013*. ACM Press, 3–14.

G. Markowsky. 1976. Chain-Complete Posets and Directed Sets with Applications. *Algebra Universalis* 6 (1976), 53–68.

E. Moggi. 1989. *An Abstract View of Programming Languages*. Technical Report ECS-LFCS-90-113. Department of Computer Science, University of Edinburgh.

U. Montanari and M. Pistore. 2000. π-Calculus, Structured Coalgebras and Minimal HD-Automata. In *Proc. MFCS 2000 (LNCS)*, Vol. 1893. Springer-Verlag, 569–578.

A. S. Murawski and N. Tzevelekos. 2012. Algorithmic Games for Full Ground References. In *Proc. ICALP 2012, Part II (LNCS)*, Vol. 7392. Springer-Verlag, 312–324.

M. Odersky. 1994. A Functional Theory of Local Names. In *Proc. POPL 1994*. ACM Press, 48–59.

D. L. Petrişan. 2011. *Investigations into Algebra and Topology over Nominal Sets*. Ph.D. Dissertation. Department of Computer Science, University of Leicester.

A. M. Pitts. 2002. Operational Semantics and Program Equivalence. In *Applied Semantics, Advanced Lectures, International Summer School, APPSEM 2000, Caminha, Portugal*, G. Barthe, P. Dybjer, and J. Saraiva (Eds.). Lecture Notes in Computer Science, Tutorial, Vol. 2395. Springer-Verlag, 378–412.

A. M. Pitts. 2006. Alpha-Structural Recursion and Induction. *Journal of the ACM* 53 (2006), 459–506.

A. M. Pitts. 2011. Structural Recursion with Locally Scoped Names. *Journal of Functional Programming* 21, 3 (2011), 235–286.

A. M. Pitts. 2013. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science, Vol. 57. Cambridge University Press.

A. M. Pitts and M. J. Gabbay. 2000. A Metalanguage for Programming with Bound Names Modulo Renaming. In *Proc. MPC 2000 (LNCS)*, Vol. 1837. Springer-Verlag, 230–255.

A. M. Pitts and I. D. B. Stark. 1993. Observable Properties of Higher Order Functions That Dynamically Create Local Names, or: What's new?. In *Proc. MFCS 1993 (LNCS)*, Vol. 711. Springer-Verlag, 122–141.

G. D. Plotkin. 1977. LCF Considered as a Programming Language. *Theoretical Computer Science* 5 (1977), 223–255.

F. Pottier. 2007. Static Name Control for FreshML. In *Proc. LICS 2007*. IEEE Computer Society Press, 356–365.

O. Savary-Belanger, S. Monnier, and B. Pientka. 2013. Programming Type-Safe Transformations Using Higher-Order Abstract Syntax. In *Certified Programs and Proofs*, G. Gonthier and M. Norrish (Eds.). Lecture Notes in Computer Science, Vol. 8307. Springer-Verlag, 243–258.

D. S. Scott. 1982. Domains for Denotational Semantics. In *Proc. ICALP 1982 (LNCS)*, Vol. 140. Springer-Verlag, 577–610.

M. R. Shinwell. 2005. *The Fresh Approach: Functional Programming with Names and Binders*. Ph.D. Dissertation. University of Cambridge. Available as University of Cambridge Computer Laboratory Technical Report UCAM-CL-TR-618.

M. R. Shinwell and A. M. Pitts. 2005. On a Monadic Semantics for Freshness. *Theoretical Computer Science* 342 (2005), 28–55.

M. R. Shinwell, A. M. Pitts, and M. J. Gabbay. 2003. FreshML: Programming with Binders Made Simple. In *Proc. ICFP 2003*. ACM Press, 263–274.

I. D. B. Stark. 1994. *Names and Higher-Order Functions*. Ph.D. Dissertation. University of Cambridge. Available as University of Cambridge Computer Laboratory Technical Report Number UCAM-CL-TR-363.

S. Staton. 2007. *Name-Passing Process Calculi: Operational Models and Structural Operational Semantics*. Ph.D. Dissertation. University of Cambridge. Available as University of Cambridge Computer Laboratory Technical Report Number UCAM-CL-TR-688.

T. Streicher. 2006. *Domain-Theoretic Foundations of Functional Programming*. World Scientific, Singapore.

D. C. Turner. 2009. *Nominal Domain Theory for Concurrency*. Ph.D. Dissertation. University of Cambridge. Available as University of Cambridge Computer Laboratory Technical Report UCAM-CL-TR-751.

D. C. Turner and G. Winskel. 2009. Nominal Domain Theory for Concurrency. In *Proc. CSL 2009*, E. Grädel and R. Kahle (Eds.). Lecture Notes in Computer Science, Vol. 5771. Springer-Verlag, 546–560.

N. Tzevelekos. 2008. *Nominal Game Semantics*. Ph.D. Dissertation. University of Oxford. Available as Oxford University Computing Laboratory Technical Report RR-09-18.

N. Tzevelekos. 2011. Fresh-Register Automata. In *Proc. POPL 2011*. ACM Press, 295–306.

N. Tzevelekos. 2012. Program Equivalence in a Simple Language with State. *Computer Languages, Systems and Structures* 38, 2 (2012), 181–198.

Received August 2013; revised February 2014; accepted April 2014