Supported Sets – A New Foundation for Nominal Sets and Automata

Thorsten Wißmann
Radboud University, Nijmegen, the Netherlands

Abstract

The present work proposes and discusses the category of supported sets which provides a uniform foundation for nominal sets of various kinds, such as those for equality symmetry, for the order symmetry, and renaming sets. We show that all these differently flavoured categories of nominal sets are monadic over supported sets. Thus, supported sets provide a canonical finite way to represent nominal sets and the automata therein, e.g. register automata and coalgebras in general. Name binding in supported sets is modelled by a functor following the idea of de Bruijn indices. This functor lifts to the well-known abstraction functor in nominal sets. Together with the monadicity result, this gives rise to a transformation process from finite coalgebras in supported sets to orbit-finite coalgebras in nominal sets. One instance of this process transforms the finite representation of a register automaton in supported sets into its configuration automaton in nominal sets.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Nominal Sets, Monads, LFP-Category, Supported Sets, Coalgebra

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.38

Related Version *Full Version*: https://arxiv.org/abs/2201.09825

Funding *Thorsten Wißmann*: Supported by the NWO TOP project 612.001.852.

Acknowledgements The author thanks Jurriaan Rot, Joshua Moerman, and Frits Vaandrager for inspiring discussions. The definition of supported sets originated from discussions with Lutz Schröder, Dexter Kozen, and Stefan Milius.

1 Introduction

Nominal sets provide an elegant framework to reason about structures that involve names, the permutation of names, and name binding. Originally used 100 years ago by Fraenkel and Mostowski for the entirely different purpose of axiomatic set theory, Gabbay and Pitts [19] re-discovered them in 1999 to define λ-expressions modulo α-equivalence as an inductive data type. Its associated structural recursion principle allowed it to define capture avoiding substitution as a total function. Since then, a plethora of results using nominal sets were established in many areas, including proof assistants [42, 5], calculi [18, 41], and automata models [7, 39, 29]. Nominal automata are capable of processing words over infinite alphabets (*data alphabets*), while having good computational properties. Their expressiveness is similar (and sometimes identical) to that of register automata [27, 13, 8], which are automata with a *finite* description processing infinite alphabets. This finiteness condition translates into the notion of *orbit-finiteness* in the nominal world, which requires extra work to obtain a finite description of nominal automata, because orbit-finite objects are infinite in general.

In recent years, more general concepts of nominal sets were considered that generalize from the permutation of names to other operations. One branch is called *renaming sets* [41, 21, 34] and allows that distinct names are mapped to the same identifier; in another branch, symmetries on other data alphabets [7, 46] were considered, for example monotone bijections on rational numbers $\mathbb{Q}$, called the *total order symmetry*.

Nominal sets are not the only categorical approach to name binding. Multiple presheaf-based approaches to name binding were developed over the years [16, 17] which are strongly related to nominal sets [23]. Also, the theory of *named sets* were introduced for the study of history dependent automata [15, 36], they were shown to be equivalent (in the categorical sense) to nominal sets [23], and they also feature a name binding construction [9]. In contrast to nominal sets, the name binding in presheaf approaches and named sets follow the method of *de Bruijn indices* [11].

Despite their rich categorical structure, it is known that the category of nominal sets (in all above-mentioned flavours) is not *monadic* over sets, that is, their theory is not an algebraic theory that can be described by a monad on sets – unlike groups, rings, vector spaces, etc, which can be described by algebraic theories.

In order to still make algebraic methods applicable, we introduce the category of *supported sets* and show that nominal sets are monadic over those. Thus, we make nominal sets applicable to results and methods known from algebraic categories, such as the generalized powerset construction [40, 6] or results about the representation of algebras [1] that provide a canonical finite representation of orbit-finite nominal sets. Moreover, supported sets do not require symmetries on the data alphabet, so they can also serve as a categorical foundation for data alphabets that are not described by symmetries. In fact, we discovered supported sets while working on a learning algorithm for register automata for general data alphabets.

Structure of the paper. After recalling the basic definitions for nominal sets (Section 3), we introduce supported sets and discuss their basic categorical properties (Section 4). Having established the monadicity of nominal sets (Section 5), we show that supported sets have a functor for name binding that lifts to nominal sets (Section 6). This yields a coalgebraic construction from (register) automata in supported sets to nominal automata (Section 7). Full proofs and also additional explanations of definitions and examples can be found in the appendix of the full version: `https://arxiv.org/abs/2201.09825`.

2 A Quick Overview of Supported Sets

Let us first take a more higher-level look at the properties of supported sets. The proposed category has such a simple definition that we can state it already here in full detail:

▶ **Definition.** *For a fixed set A, the category* $\mathsf{Supp}(A)$ *contains the following data:*

- *a supported set* (X, s_X) *is a set* X *and a map* $\mathsf{s}_X\colon X \to \mathcal{P}_\mathsf{f}(A)$*, called the* support *(where* $\mathcal{P}_\mathsf{f}$ *denotes finite powerset),*
- *a supported map* $f\colon (X, \mathsf{s}_X) \to (Y, \mathsf{s}_Y)$ *is a map* $f\colon X \to Y$ *with the property that* $\mathsf{s}_Y(f(x)) \subseteq \mathsf{s}_X(x)$ *for all* $x \in X$*.*

This definition reflects the basic principles when working with data alphabets A: the terms or structures $x \in X$ of interest store finitely many data values $\mathsf{s}_X(x) \subseteq A$. The computations f on such a structure x can not invent new data values but may decide to drop data values, e.g. by clearing a register: $\mathsf{s}_Y(f(x)) \subseteq \mathsf{s}_X(x)$.

In contrast to nominal sets, the map s_X is a structural property of a supported set X and not a derived notion. Thus, we can define supported sets by simply specifying X and s_X without any notion of permutation or renaming of data values.

$\mathsf{Supp}(A)$ has nice categorical properties: it is complete, cocomplete, and cartesian closed. $\mathsf{Supp}(A)$ is locally finite, that is, a supported set is *finitely presentable* iff it is finite, and so it is easy to represent supported sets for algorithmic purposes.

Despite the little structure of supported sets, *name binding* exists as a functor on $\mathsf{Supp}(A)$ (for countably infinite A) which is reminiscent of de Bruijn indices [11], and thus different to the permutation based *abstraction functor* of nominal sets. Surprisingly, the binding functor of $\mathsf{Supp}(A)$ *lifts* to the abstraction functor in nominal sets (up to natural isomorphism).

Different kinds of *nominal theories* can be modelled as monads on supported sets. If A is a countable infinite set, the following categories are then *monadic* over $\mathsf{Supp}(A)$:

1. **Nominal Sets** (for the equality symmetry) [19, 38].
2. **Nominal renaming sets** in which atoms can be merged together [41, 21, 34].
3. **Ordered nominal sets** in which the atoms are rational numbers and can only be renamed in a monotone way [7, 46].

These monadic adjunctions make general results about monadic adjunctions applicable:

1. Finite representation [1]: a (possibly infinite) nominal set is orbit-finite iff it can be described by a finite supported set of *generators* and finitely many *equations*.
2. Generalized Powerset Construction [40, 6]: a register automaton lives as a finite coalgebra in supported sets. The construction transforms it into an orbit-finite nominal coalgebra, whose state-space is then the configuration space of the original register automaton.

These applications show that supported sets should not be understood as a competitor to nominal sets, but as a common foundation for different nominal symmetries, and it may serve as a categorical framework for data alphabets that do not admit symmetries at all.

3 Preliminaries on Nominal Sets

Before introducing supported sets in detail, we first recall some notations and basic concepts of nominal sets. We assume that the reader is familiar with basic category theory, but we restrict to set-theoretic definitions wherever possible.

▶ **Notation 3.1.** *Given sets X, Y, the set of all maps $X \to Y$ is written Y^X. Injective maps are denoted by $\rightarrowtail$, surjective maps by $\twoheadrightarrow$, and $\cdot$ denotes map composition. We fix sets $1 = \{0\}$, $2 = \{0, 1\}$. The cycle notation $(a_0\ a_1 \cdots a_{n-1})$ for elements of a set A denotes the bijection $A \to A$ sending $a_0 \mapsto a_1$, $a_1 \mapsto a_2$, …, $a_{n-1} \mapsto a_0$, and fixing all other elements of A.*

Nominal sets and various flavours thereof are built around the notion of monoid actions, which specifies how atoms nested in some structure can be permuted or renamed.

▶ **Definition 3.2.** *Given a monoid $(M, \cdot, e)$, an M-set is a set X together with a map "$\cdot$": $M \times X \to X$, called the* action *and written infix $m \cdot x$ for $x \in X$, $m \in M$, such that $e \cdot x = x$ and $(m \cdot m') \cdot x = m \cdot (m' \cdot x)$ for all $m, m' \in M$, $x \in X$. Thus, we use "$\cdot$" both for the monoid multiplication and the action. A map $f\colon X \to Y$ between M-sets $(X, \cdot)$, $(Y, \star)$ is* equivariant *if $f(m \cdot x) = m \star f(x)$ for all $m \in M$, $x \in X$.*

Throughout the paper, M will be a submonoid of $(A^A, \cdot, \mathsf{id}_A)$ for a set A, written $M \leq A^A$, in which most of the results are parametric. The set A is called the set of *atoms*, which can intuitively be understood as names of registers or data-values that appear in the input of an automaton or as names used for binding operations. The submonoid M determines a subset of maps of interest, closed under composition. Thus, we use $\cdot$ both for map composition and monoid-multiplication, and moreover, the unit of the monoid M is simply id_A.

▶ **Example 3.3.** The main instances of monoids M for nominal techniques are as follows:

1. For a set A, let $\mathfrak{S}_\mathsf{f}(A)$ be the bijections on A that modify only finitely many elements, i.e.

$$\mathfrak{S}_\mathsf{f}(A) := \{\phi\colon A \to A \mid \phi \text{ bijective and } \{a \in A \mid \phi(a) \neq a\} \text{ is finite}\}$$

For nominal sets (over the equality symmetry), one fixes a countably infinite set $\mathbb{A}$ (understood as *names*) and fixes $M := \mathfrak{S}_\mathsf{f}(\mathbb{A})$ [19, 38].

2. For the set $\mathbb{Q}$ of rational numbers, let $\mathsf{Aut}(\mathbb{Q},<)$ be the set of bijective and monotone maps $\mathbb{Q} \to \mathbb{Q}$. For nominal sets over the total order symmetry, one considers $M := \mathsf{Aut}(\mathbb{Q},<)$ [7].
3. Let $\mathsf{Fin}(A) \subseteq A^A$ be the set of maps $f\colon A \to A$ for which $\{f(a) \neq a \mid a \in A\}$ is finite. For *nominal renaming sets*, one considers $A := \mathbb{A}$ and $M := \mathsf{Fin}(\mathbb{A})$ [21].

An element $x \in X$ of an M-set $(X, \cdot)$ can be understood as a structure with elements from A embedded. We can alter these embedded elements according to $m \in M$, yielding $m \cdot x \in X$.

▶ **Example 3.4.** For every set A and $M \leq A^A$, the following sets are M-sets:
1. The set A itself with the action $m \cdot a := m(a)$, for $m \in M$, $a \in A$.
2. If X is an M-set, then so is $\mathcal{P}_\mathsf{f}(X)$, the set of finite subsets of X, where the action is defined point-wise: $m \cdot S := \{m \cdot x \mid x \in S\}$ for $m \in M, S \in \mathcal{P}_\mathsf{f}X$.
3. Every set D can be equipped with the *discrete* action: $m \cdot d = d$ for all $m \in M, d \in D$.
4. The set M itself is an M-set with monoid multiplication $M \times M \to M$ as the action.

The outcome of $m \cdot x$ only depends on what m does on the atoms $S \subseteq A$ buried in x:

▶ **Definition 3.5.** *We say that $m, m' \in M$ are* identical on $S \subseteq A$, *written $m \approx_S m'$, if $m(a) = m'(a)$ for all $a \in S$.*

With the intuition that the action of an M-set X renames the atoms A in an element $x \in X$, we can derive from the action which atoms an element $x \in X$ carries. Then, an M-set is *nominal*, if each $x \in X$ carries only finitely many atoms.

▶ **Definition 3.6** [21, Def. 7]**.** *A set $S \subseteq A$* supports $x \in X$ *of an M-set X, if for all $m \approx_S m'$ (in M), we have $m \cdot x = m' \cdot x$. An M-set X is* nominal *if every element of X has some finite support, i.e. is supported by a finite set $S \subseteq A$.*

If M happens to be a group, then the definition of support can be simplified slightly [19, 38].

▶ **Example 3.7.** Most of the M-sets from Example 3.4 are nominal:
1. A is nominal: every $a \in A$ is supported by $S := \{a\}$, and also by any superset of S.
2. If X is a nominal M-set, then so is $\mathcal{P}_\mathsf{f}(X)$. A set $E \in \mathcal{P}_\mathsf{f}(X)$ of elements is supported by the union of finite supports of the elements $x \in E$. This union is finite because E is so.
3. Every discrete M-set D is nominal because every $x \in D$ is supported by the empty set.
4. For all monoids M of interest for nominal techniques, M considered as an M-set is not nominal: whenever M is a nominal M-set, then *every* M-set is nominal. For example, $\mathfrak{S}_\mathsf{f}(\mathbb{A})$ is not nominal because no $\sigma \in \mathfrak{S}_\mathsf{f}(\mathbb{A})$ has finite support.

▶ **Definition 3.8.** *Let $\mathsf{Nom}(M)$ be the category of nominal M-sets and equivariant maps.*

For our main instances (Example 3.3), we obtain the following categories:
1. The category of *nominal sets* is denoted by $\mathsf{Nom} = \mathsf{Nom}(\mathfrak{S}_\mathsf{f}(\mathbb{A}))$ (slightly overloading notation), that is for $A := \mathbb{A}$ and $M := \mathfrak{S}_\mathsf{f}(\mathbb{A})$. This is called *the equality symmetry.*
2. *Ordered nominal sets* are $\mathsf{OrdNom} = \mathsf{Nom}(\mathsf{Aut}(\mathbb{Q},<))$, i.e. for $A := \mathbb{Q}$, $M := \mathsf{Aut}(\mathbb{Q},<)$.
3. *Nominal renaming sets* are $\mathsf{RnNom} = \mathsf{Nom}(\mathsf{Fin}(\mathbb{A}))$, i.e. for $A := \mathbb{A}$, $M := \mathsf{Fin}(\mathbb{A})$.

▶ **Definition 3.9** [7, Definition 4.11]**.** *A monoid $M \leq A^A$* admits least supports *if each element of a nominal M-set has a least finite support. If so, we write $\mathsf{supp}_X\colon X \to \mathcal{P}_\mathsf{f}(A)$ for the map that sends elements of the M-set X to their least finite support.*

In all running examples of nominal set flavours, the monoid admits least supports, i.e. Nom [20, 38], OrdNom [7] and RnNom [21]. Intuitively, $\mathsf{supp}(x) \subseteq A$ can be understood as precisely the atoms that appear in x. For example, $\mathsf{supp}_A\colon A \to \mathcal{P}_\mathsf{f}(A)$ sends a to $\{a\}$, and $\mathsf{supp}_{\mathcal{P}_\mathsf{f}(X)}\colon \mathcal{P}_\mathsf{f}(X) \to \mathcal{P}_\mathsf{f}(A)$ is $\mathsf{supp}_{\mathcal{P}_\mathsf{f}(X)}(E) = \bigcup\{\mathsf{supp}_X(x) \mid x \in E\}$. The opposite of an atom $a \in A$ in the support is:

▶ **Definition 3.10.** *An atom $a \in A$ is* fresh *for $x \in X$ in a nominal set X (notated $a \# x$), if $a \notin \mathsf{supp}_X(x)$. For multiple elements, we write $a \# x, y$ to denote that a is fresh for x and y.*

Both freshness and support are preserved by equivariant maps. Intuitively, equivariant maps can possibly forget about atoms, but can never introduce new atoms:

▶ **Lemma 3.11.** *For an equivariant map, if S supports x, then S also supports $f(x)$. If X and Y have least finite supports, then $\mathsf{supp}_Y(f(x)) \subseteq \mathsf{supp}_X(x)$.*

The set of elements in an M-set that can be reached from an element x is called the orbit:

▶ **Definition 3.12.** *Given a group M, the* orbit *of an element x in an M-set is the subset $\{m \cdot x \mid m \in M\} \subseteq X$. A nominal M-set is* orbit-finite *if it consists of finitely many orbits.*

In this definition, we assume M to be a group, because then, every nominal set X is a disjoint union of orbits. For $M := \mathfrak{S}_\mathsf{f}(\mathbb{A})$, the *orbit-finite* nominal sets are precisely the finitely presentable objects in Nom [37, Proposition 2.3.7]. E.g. $\mathbb{A}$ is orbit-finite, $\mathcal{P}_\mathsf{f}\mathbb{A}$ is not. In nominal automata, one requires the state set to be orbit-finite, as opposed to finiteness in classical automata theory.

4 Supported Sets

The central notion of the present paper are supported sets, which are parametric in the set A of atoms or data symbols of interest:

▶ **Definition 4.1.** *For a set A, the category of* supported sets $\mathsf{Supp}(A)$ *contains the following:*

1. *a supported set X is a set X together with a map $\mathsf{s}_X \colon X \to \mathcal{P}_\mathsf{f}(A)$.*
2. *a supported map $f \colon (X, \mathsf{s}_X) \to (Y, \mathsf{s}_Y)$ is a map $f \colon X \to Y$ with $\mathsf{s}_Y(f(x)) \subseteq \mathsf{s}_X(x)$ for all $x \in X$. This means, f makes the following triangle weakly commute.*

$$\begin{tikzcd}
X \arrow[rr, "f"] \arrow[dr, "\mathsf{s}_X"'] & \supseteq & Y \arrow[dl, "\mathsf{s}_Y"] \\
& \mathcal{P}_\mathsf{f}(A) &
\end{tikzcd}$$

Whenever clear from the context, we simply speak of supported sets $X, Y \in \mathsf{Supp}(A)$ and supported maps $f \colon X \to Y$, leaving s_X and s_Y implicit.

The intuition for supported sets comes from the least finite support map supp of nominal sets. Hence, in a supported set X, the map $\mathsf{s}_X(x)$ tells which atoms are carried by $x \in X$, but we can not permute or rename them in general.

The definition of the morphisms in supported sets comes from the property of equivariant maps: they possibly forget about atoms in the support, but they can never introduce new atoms (Lemma 3.11). This observation becomes the defining property of supported maps.

▶ **Example 4.2.** The set A itself is a support set with $\mathsf{s}_A(a) = \{a\}$. However, the singleton subset $\{a\} \subseteq A$ is also a supported set with $\mathsf{s}_{\{a\}}(a) = \{a\}$. In general, every nominal M-set X (for M admitting least supports) yields a supported set by putting $\mathsf{s}_X := \mathsf{supp}_X$.

The difference between nominal and supported sets is that supp in a nominal set X is a derived notion since it is implicit in the M-set action. On the other hand, for supported sets, s_X is part of the syntactical structure. For the sake of clarity, we use different mathematical symbols for the semantical supp and the structural s.

▶ **Lemma 4.3.** *For every $M \leq A^A$, there is a faithful functor $U\colon \mathsf{Nom}(M) \to \mathsf{Supp}(A)$ sending $(X, \cdot)$ to a supported set (X, s_X), where $\mathsf{s}_X(x)$ is the intersection of all finite supports of x in $(X, \cdot)$. If $M \leq A^A$ admits least supports, then $\mathsf{s}_{UX} = \mathsf{supp}_X$. Equivariant maps in $\mathsf{Nom}(M)$ are sent to their underlying map by U.*

Later, we show that this forgetful functor $\mathsf{Nom}(M) \to \mathsf{Supp}(A)$ is right-adjoint and even monadic, so one can consider nominal sets as algebras on supported sets. Before, we establish some categorical properties of $\mathsf{Supp}(A)$ (generic in the choice of A) that will come in handy.

4.1 Universal Constructions and Finiteness

Unsurprisingly, supported sets have a very set-like nature. In the present Section 4.1, we let V denote the forgetful functor $V\colon \mathsf{Supp}(A) \to \mathsf{Set}$ that forgets the support map and sends (X, s_X) to the plain set X. Conversely, every set can be equipped with trivial support:

▶ **Lemma 4.4.** *The inclusion functor $J\colon \mathsf{Set} \hookrightarrow \mathsf{Supp}(A)$ defined by $JX = (X, \mathsf{s}_X)$ with $\mathsf{s}_X(x) = \emptyset$ is right-adjoint to the forgetful $V\colon \mathsf{Supp}(A) \to \mathsf{Set}$ ($V \dashv J$) and $VJ = \mathsf{Id}_{\mathsf{Set}}$.*

In other words, Set is a reflective subcategory of $\mathsf{Supp}(A)$. Similarly to sets, universal constructions such as limits and colimits all exist in supported sets and are (almost) set-like. Given a diagram $D\colon \mathcal{D} \to \mathsf{Set}$, we write $\mathsf{pr}_X\colon \lim D \to DX$ for the limit projection map of $X \in \mathcal{D}$ and $\mathsf{in}_X\colon DX \to \mathsf{colim} D$ for the colimit injections.

▶ **Proposition 4.5.** *All colimits in $\mathsf{Supp}(A)$ exist: Given a diagram $D\colon \mathcal{D} \to \mathsf{Supp}(A)$, the colimit is formed in Set and then equipped with the support $\mathsf{s}\colon \mathsf{colim}\, VD \to \mathcal{P}_\mathsf{f}(A)$:*

$$\mathsf{s}(c) = \bigcap \{\mathsf{s}_{DY}(y) \mid Y \in \mathcal{D}, y \in DY, \mathsf{in}_Y(y) = c\}.$$

The intersection handles the case where elements of possibly different support are identified in the colimit. Thus, the intersection vanishes if there are no morphisms in the diagram:

▶ **Example 4.6.** The coproduct of supported sets X, Y is given by their disjoint union, $X + Y$ equipped with support $\mathsf{s}_{X+Y}(\mathsf{in}_X(x)) = \mathsf{s}_X(x)$ and $\mathsf{s}_{X+Y}(\mathsf{in}_Y(y)) = \mathsf{s}_Y(y)$.

Limits on the other hand are formed differently because of the finite support map:

▶ **Proposition 4.7.** *$\mathsf{Supp}(A)$ is complete. The limit of a diagram $D\colon \mathcal{D} \to \mathsf{Supp}(A)$ is the subset of finitely supported elements in the limit $\lim VD$ in Set:*

$$\lim D = \{x \in \lim VD \mid \bigcup_{Y \in \mathcal{D}} \mathsf{s}_{DY}(\mathsf{pr}_Y(x)) \text{ is finite}\} \tag{1}$$

The process of restricting to finitely supported elements also happens for limits in nominal sets. For finite limits, this side-condition disappears:

▶ **Corollary 4.8.** *$V\colon \mathsf{Supp}(A) \to \mathsf{Set}$ preserves all finite limits.*

$\mathsf{Supp}(A)$ is cartesian closed, that is, we have an internal hom object. Similarly to nominal set, this internal hom object X^E in $\mathsf{Supp}(A)$ contains more than just the hom set $\mathsf{hom}(E, X)$:

▶ **Definition 4.9.** *For supported sets X and E, let X^E contain those maps $f\colon E \to X$ for which $\mathsf{s}_{X^E}(f) = \bigcup_{e \in E} \mathsf{s}_X(f(e)) \setminus \mathsf{s}_E(e)$ is finite.*

Note that "*map*" really means ordinary maps between sets. Intuitively, a map $f \in X^E$ may introduce finitely many atoms when mapping elements of E to X. For example, if E is a finite supported set, then X^E contains all maps $E \to X$. In contrast, a *supported* map may not introduce any new atoms at all, hence, a map $f\colon E \to X$ is a supported map if and only if $\mathsf{s}_{X^E}(f) = \emptyset$. In particular, $\mathsf{hom}(E, X) \subseteq X^E$.

▶ **Proposition 4.10.** *$(-) \times E$ is left adjoint to $(-)^E$, for all supported sets E.*

In contrast to nominal sets, categorical finiteness notions in $\mathsf{Supp}(A)$ express actual finiteness. For the present paper, we do not need the precise definition of finitely presentable objects and locally finitely presentable (lfp) categories [3, 22]. For the present purposes, it is enough to mention that $\mathsf{Supp}(A)$ is lfp and that the finitely presentable objects are the finite supported sets, i.e. $\mathsf{Supp}(A)$ is locally finite. Detailed definitions can be found in the proof.

▶ **Proposition 4.11.** *$\mathsf{Supp}(A)$ is lfp and finite presentability is actual finiteness.*

4.2 Injectivity, Surjectivity, Quasitopoi

Notions of surjective and injective maps generalize from Set:

▶ **Lemma 4.12.** *Monomorphisms in $\mathsf{Supp}(A)$ are precisely the injective supported maps and epimorphisms are precisely the surjective supported maps.*

However, not all bijective supported maps are isomorphisms in $\mathsf{Supp}(A)$, because they may drop atoms. That is, the difference between bijections and isomorphisms is the following:

▶ **Definition 4.13.** *A map $f\colon X \to Y$ is called* support-reflecting *if $\mathsf{s}_Y \cdot f = \mathsf{s}_X$.*

▶ **Lemma 4.14.** *The isomorphisms in $\mathsf{Supp}(A)$ are the support-reflecting bijective maps.*

▶ **Example 4.15.** The unit $\eta_A\colon A \to JVA$ of the above adjunction $V \dashv J$ to Set is a bijective supported map but not support-reflecting, because $\mathsf{s}_A(a) = \{a\}$ and $\mathsf{s}_{JVA}(a) = \emptyset$. Thus, it is not an isomorphism in $\mathsf{Supp}(A)$.

This shows that $\mathsf{Supp}(A)$ is not a topos – in contrast to Set and Nom. As we will see, it is a *quasitopos* [26, Def. 2.6.1], which entails that $\mathsf{Supp}(A)$ is locally cartesian closed and that it has a subobject classifier for *regular* monomorphisms. The precise definition of regularity is not relevant here (but cf. [2, Rem 7.76(2)]), since it can be nicely characterized via support-reflection:

▶ **Lemma 4.16.** *A monomorphism is regular iff it is support-reflecting. Moreover, $t\colon 1 \to 2$, $0 \mapsto 1$ is a regular-subobject classifier, i.e. for every supported set X, the support-reflecting monomorphisms $m\colon S \to X$ are in correspondence to characteristic maps $\chi_S\colon X \to 2$.*

In summary, for every set A, $\mathsf{Supp}(A)$ is (co)complete, cartesian closed, and locally finite, and we can express many of the above-mentioned properties in one line:

▶ **Theorem 4.17.** *$\mathsf{Supp}(A)$ is a quasitopos (for both [26, Def. 2.6.1] and [2, Def. 28.7]).*

Thus, $\mathsf{Supp}(A)$ constitutes a rich basis to study algebraic theories on it, such as nominal sets.

5 Monadicity of Nominal M-Sets

If a category is *monadic*, then intuitively, it is a well-behaved class of algebras. If so, it becomes applicable for many results and constructions, e.g. regarding representation and the generalized powerset construction. Algebraic theories have been studied extensively throughout the decades and are in correspondence to monads: given a monad T on a category $\mathcal{C}$, e.g. Set, its *Eilenberg-Moore category* $\mathsf{EM}(T)$ contains the models of the algebraic theory defined by T (see e.g. [4, 10.3]). The forgetful functor $U\colon \mathsf{EM}(T) \to \mathcal{C}$ is a right-adjoint functor and its left-adjoint $F\colon \mathcal{C} \to \mathsf{EM}(T)$ sends an object $X \in \mathcal{C}$ (e.g. a set of generators) to the *free algebra* on X:

$$F\colon \mathsf{Supp}(A) \to \mathsf{EM}(T) \quad \dashv \quad U\colon \mathsf{EM}(T) \to \mathsf{Supp}(A) \qquad (\text{for } \mathcal{C} = \mathsf{Supp}(A)) \tag{2}$$

The category of M-sets is monadic over Set, but $\mathsf{Nom}(M)$ fails to be monadic over Set in the instances of interest (Example 3.3). The reason for this is that infinite products in $\mathsf{Nom}(M)$ are different than in Set [38], so the forgetful functor $\mathsf{Nom}(M) \to \mathsf{Set}$ is not right-adjoint and therefore not monadic. As we will show, $\mathsf{Nom}(M)$ is still monadic, but *over supported sets.*

▶ **Definition 5.1.** *A functor $U\colon \mathcal{D} \to \mathcal{C}$ is called* monadic (over $\mathcal{C}$) *if there is a monad T on $\mathcal{C}$ such that $\mathcal{D}$ is $\mathsf{EM}(T)$ and U is the forgetful functor.*

We first show that $U\colon \mathsf{Nom}(M) \to \mathsf{Supp}(A)$ is right-adjoint and then that it is monadic. Then, we can view nominal sets as algebras over $\mathsf{Supp}(A)$. There, the algebraic operations come from the following (supported) set:

▶ **Definition 5.2.** *For $M \le A^A$ and a set $S \subseteq A$, put $[m]_S$ for the $\approx_S$-equivalence class of m (cf. Definition 3.5) and $M/S = \{[m]_S \mid m \in M\}$ for the supported set of equivalence classes with $\mathsf{s}_{M/S}([m]_S) := m[S] = \{m(a) \mid a \in S\}$.*

We can consider M/S either as equivalence classes of M-elements that are identical on S or alternatively as special maps $S \to A$ that are obtained by restricting some $m \in M \subseteq A^A$ to $S \subseteq A$. Intuitively, M/S is the free nominal set on one generator with support S. Hence, the free nominal M-set over a supported set X is the union of multiple M/S:

▶ **Definition 5.3.** *Fix the functor $M\bullet\colon \mathsf{Supp}(A) \to \mathsf{Supp}(A)$ by $M\bullet X = \coprod_{x\in X} M/\mathsf{s}(x)$.*

$M\bullet$ is the monad that will define nominal M-sets as an algebraic theory. Written with elements, we have $M\bullet X = \{([m]_{\mathsf{s}_X(x)}, x) \mid x \in X\}$ with $\mathsf{s}_{M\bullet X}([m]_{\mathsf{s}(x)}, x) = m[\mathsf{s}(x)]$. For a supported set X, every such element $([m]_{\mathsf{s}(x)}, x)$ of the nominal set $M\bullet X$ is equivalently an element $x \in X$ together with an $|\mathsf{s}(x)|$-tuple of atoms. Of course, such an equivalence class $[m]_{\mathsf{s}(x)}$ is not an arbitrary tuple of elements of A but only one that is obtained from restricting some $m \in M \subseteq (A \to A)$ to $\mathsf{s}_X(x) \subseteq A$. For example, for the equality symmetry $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$, such a $t \in M/\mathsf{s}_X(x)$ is a tuple of *distinct* elements of A.

▶ **Proposition 5.4.** *$M\bullet X$ with the M-set action $\ell \cdot ([m]_{\mathsf{s}(x)}, x) = ([\ell \cdot m]_{\mathsf{s}(x)}, x)$ is nominal and gives rise to a functor $F\colon \mathsf{Supp}(A) \to \mathsf{Nom}(M)$, $FX = M\bullet X$.*

Note that there is no statement about the least finite support of $([m]_{\mathsf{s}(x)}, x) \in M\bullet X$. In the proof, we show that it is supported by $m[\mathsf{s}(x)]$, but the least support might be smaller. Nevertheless, the existence of finite supports in $M\bullet X$ suffices to show the adjunction $F \dashv U$:

▶ **Proposition 5.5.** *If $M \le A^A$ admits least supports, then $F\colon \mathsf{Supp}(A) \to \mathsf{Nom}(M)$ is left-adjoint to $U\colon \mathsf{Nom}(M) \to \mathsf{Supp}(A)$ with unit $\eta_X\colon X \to UFX$, $\eta_X(x) = ([\mathsf{id}_A]_{\mathsf{s}(x)}, x)$.*

$$\begin{tikzcd}
\overbrace{\mathsf{Nom}}^{\mathsf{Nom}(\mathfrak{S}_{\mathsf{f}}(\mathbb{A}))} \arrow[dr, bend left, "U"] \arrow[dr, phantom, "\dashv"] & \overbrace{\mathsf{RnNom}}^{\mathsf{Nom}(\mathsf{Fin}(\mathbb{A}))} \arrow[d, bend left, "U"] \arrow[d, phantom, "\dashv"] & \overbrace{\mathsf{OrdNom}}^{\mathsf{Nom}(\mathsf{Aut}(\mathbb{Q},<))} \arrow[dl, bend left, "U"] \arrow[dl, phantom, "\dashv"] \\
& \boxed{\mathsf{Supp}(A) \text{ for countably infinite } A} \arrow[ul, bend left, "\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet(-)"] \arrow[u, bend left, "\mathsf{Fin}(\mathbb{A})\bullet(-)"] \arrow[ur, bend left, "\mathsf{Aut}(\mathbb{Q},<)\bullet(-)"] &
\end{tikzcd}$$

Figure 1 Monadic adjunctions (Ex. 5.10).

$$\begin{tikzcd}
\boxed{\begin{matrix}\textbf{Monadic}\\ \mathsf{Nom}(\mathfrak{S}_{\mathsf{f}}(\mathbb{A}))\\ \\ \mathsf{Supp}(\mathbb{A})\end{matrix}} & & \boxed{\begin{matrix}\textbf{Monadic}\\ [\mathbb{I},\mathsf{Set}]\\ \\ [|\mathbb{I}|,\mathsf{Set}]\end{matrix}}
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\mathsf{Nom}(\mathfrak{S}_{\mathsf{f}}(\mathbb{A})) \arrow[rr, bend left, "I"] \arrow[rr, phantom, "\top"] \arrow[d, bend left, "U"] \arrow[d, phantom, "\dashv"] \arrow[rrd, bend left, "D\cdot U"] \arrow[rrd, phantom, "\top"] & & {[\mathbb{I},\mathsf{Set}]} \arrow[ll, bend left, "I^*"] \arrow[d, bend left, "U^{\mathbb{I}}"] \arrow[d, phantom, "\dashv"] \\
\mathsf{Supp}(\mathbb{A}) \arrow[u, bend left, "FX=\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet X"] \arrow[rr, bend left, "D"] \arrow[rr, phantom, "\top"] & & {[|\mathbb{I}|,\mathsf{Set}]} \arrow[ll, bend left, "\Sigma"] \arrow[u, bend left, "F^{\mathbb{I}}"] \arrow[llu, bend left, "F\cdot\Sigma"]
\end{tikzcd}$$

Figure 2 Relation to presheaves for $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$.

This adjunction shows that every nominal M-set is an Eilenberg-Moore algebra and that every supported set generates a free nominal M-set satisfying a universal mapping property. The remaining direction to prove is that every Eilenberg-Moore algebra for $M\bullet$ is indeed a nominal set, or concretely, that the adjunction is monadic. For doing so, we impose a property on the monoid M of interest, which will not only be used in the monadicity proof but also help us to characterize the least finite supports of the free nominal sets:

▶ **Definition 5.6** [7, Def. 9.6]**.** *A monoid $M \le A^A$ is called* fungible *if for all finite $R \subseteq A$ and $a \in A \setminus R$, there is some $\ell \in M$ with $\ell \approx_R \mathsf{id}_A$ and $\ell(a) \neq a$.*

Intuitively, the condition expresses that the atoms in the support can be renamed independently of each other: if an element is supported by $R \cup \{a\}$, we can always rename a to something fresh while keeping the rest of the support fixed.

▶ **Example 5.7.** All the leading examples (Ex. 3.3) have fungible monoids.

- For $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$ and $M := \mathsf{Fin}(\mathbb{A})$, consider a finite $R \subseteq \mathbb{A}$ and $a \mathrel{\#} R$. Then for $b \mathrel{\#} a, R$, the permutation $\ell = (a\,b)$ fulfils the desired property $(a\,b) \approx_R \mathsf{id}_{\mathbb{A}}$.
- For $M := \mathsf{Aut}(\mathbb{Q},<)$, the verification uses a notion called homogeneity [47, Lemma 5.2].

▶ **Lemma 5.8.** *If M is fungible, then $M\bullet X$ is a nominal M-set with $\mathsf{supp}([m]_{\mathsf{s}(x)}, x) = m[\mathsf{s}(x)]$ for every $m \in M$ and $x \in X$.*

We prove the adjunction to be monadic via Beck's theorem [32, Sec. VI.7], which will provide us with the monadicity of the leading examples of nominal sets by instantiating M.

▶ **Theorem 5.9.** *$U\colon \mathsf{Nom}(M) \to \mathsf{Supp}(A)$ is monadic and $\mathsf{Nom}(M) = \mathsf{EM}(M\bullet(-))$, for every fungible M admitting least supports.*

▶ **Example 5.10.** The following categories are monadic over $\mathsf{Supp}(A)$, for A being countably infinite. The monadic adjunctions are visualized in Figure 1, and the monads listed below.

1. The category Nom of nominal sets (with equality symmetry) is monadic over $\mathsf{Supp}(\mathbb{A})$. The operations on a generator x in the corresponding theory are injective maps $\mathsf{s}(x) \rightarrowtail \mathbb{A}$:

$$\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet X = \big\{(\pi, x) \mid x \in X, \pi\colon \mathsf{s}_X(x) \rightarrowtail \mathbb{A}\big\}$$

2. The category RnNom of nominal renaming sets is monadic over $\mathsf{Supp}(\mathbb{A})$. The operations on a generator x are arbitrary maps $\mathsf{s}(x) \to \mathbb{A}$ (not necessarily injective):

$$\mathsf{Fin}(\mathbb{A})\bullet X = \big\{(\pi, x) \mid x \in X, \pi\colon \mathsf{s}_X(x) \to \mathbb{A}\big\}$$

3. The category OrdNom of nominal sets for the total order symmetry is monadic over $\mathsf{Supp}(\mathbb{Q})$ (using $\mathbb{Q} \cong A$). The operations on x are monotone injective maps $\mathsf{s}(x) \rightarrowtail \mathbb{Q}$:

$$\mathsf{Aut}(\mathbb{Q},<)\bullet X = \big\{(\pi,x) \mid x \in X, \pi\colon \mathsf{s}_X(x) \rightarrowtail \mathbb{Q}\ \forall q,p \in \mathsf{s}_X(x) : q < p \Rightarrow \pi(q) < \pi(p)\big\}$$

As a direct application of the monadicity, we can characterize orbit-finite nominal sets and finitely presentable objects in $\mathsf{Nom}(M)$ using a general result about algebraic categories [1, Thm. 3.7]:

▶ **Example 5.11.** A nominal M-set X is finitely presentable iff it can be described by a finite supported set G of *generators* and a finite subset $E \subseteq (M\bullet G) \times (M\bullet G)$ of *equations*. This characterization means that given such finite G and E, we obtain projection maps

$$\begin{tikzcd} E \arrow[r, shift left, "\ell"] \arrow[r, shift right, "r"'] & M\bullet G & \text{in } \mathsf{Supp}(A) & \iff & M\bullet E \arrow[r, shift left, "\bar\ell"] \arrow[r, shift right, "\bar r"'] & M\bullet G & \text{in } \mathsf{Nom}(M) \end{tikzcd}$$

and their coequalizer in $\mathsf{Nom}(M)$ is X.

For $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$, 1. every orbit-finite nominal set can be described by such finite supported sets G and E, and 2. any such finite system of equations E on G presents an orbit-finite nominal set. For example, the nominal set of unordered pairs of atoms can be described by one generator g encoded as a supported set $G = \{g\}, \mathsf{s}_G(g) := \{a,b\}$ (for fixed $a,b \in \mathbb{A}$) and one equation $E := \big\{(a\,b)\cdot g = \mathsf{id}\cdot g\big\}$. Here, we use intuitive notation to denote

$$E := \Big\{\big(\,([(a\,b)]_{\{a,b\}}, g),\ ([\mathsf{id}]_{\{a,b\}}, g)\,\big)\Big\} \subseteq (M\bullet G) \times (M\bullet G).$$

With the projections $\ell, r\colon E \to M\bullet G$ (in $\mathsf{Supp}(A)$) and their extensions $\bar\ell, \bar r$ to Nom, we have the unordered pairs as a coequalizer diagram in Nom:

$$\begin{tikzcd} M\bullet E \arrow[r, shift left, "\bar\ell"] \arrow[r, shift right, "\bar r"'] & M\bullet G \arrow[r, two heads] & \big\{\{c,d\} \mid c,d \in \mathbb{A}, c \neq d\big\}. \end{tikzcd}$$

▶ Remark 5.12 (Relation to Presheaves). The category of supported sets $\mathsf{Supp}(\mathbb{A})$ nicely fits into an existing diagram of Kurz, Petrisan, and Velebil [31] relating nominal sets Nom (for the equality symmetry $\mathfrak{S}_{\mathsf{f}}(\mathbb{A})$) with two presheaf categories:

1. $[\mathbb{I}, \mathsf{Set}]$ is the category of functors $P\colon \mathbb{I} \to \mathsf{Set}$ (*sets in context*), where the objects of $\mathbb{I}$ are finite subsets of $\mathbb{A}$ (i.e. $|\mathbb{I}| = \mathcal{P}_{\mathsf{f}}\mathbb{A}$) and the morphisms are injective maps (i.e. $\mathbb{I} \neq (\mathcal{P}_{\mathsf{f}}\mathbb{A}, \subseteq)$).
2. $[|\mathbb{I}|, \mathsf{Set}]$ is the category of functors $P\colon |\mathbb{I}| \to \mathsf{Set}$, from the set $\mathcal{P}_{\mathsf{f}}(\mathbb{A})$ to Set, i.e. P is a $\mathcal{P}_{\mathsf{f}}(\mathbb{A})$-indexed family of sets.

Figure 2 shows the result when extending the diagram of Kurz et al. [31] by $\mathsf{Supp}(\mathbb{A})$. Let us go through the functors and adjunctions relating the categories:

1. The left adjoint $\Sigma\colon [|\mathbb{I}|, \mathsf{Set}] \to \mathsf{Supp}(\mathbb{A})$ sends a family $X\colon \mathcal{P}_{\mathsf{f}}(\mathbb{A}) \to \mathsf{Set}$ to the coproduct $\Sigma(X) = \coprod_{S \in \mathcal{P}_{\mathsf{f}}(\mathbb{A})} X(S)$, where the component for $S \in \mathcal{P}_{\mathsf{f}}(\mathbb{A})$ has support S.
2. The right-adjoint $D\colon \mathsf{Supp}(\mathbb{A}) \to [|\mathbb{I}|, \mathsf{Set}]$ forms down-sets: for a supported set X, the family $DX\colon \mathcal{P}_{\mathsf{f}}\mathbb{A} \to \mathsf{Set}$ is given by $DX(S) = \{x \in X \mid \mathsf{s}_X(x) \subseteq S\}$.
3. Since adjunctions compose, we have the adjunction $F\cdot\Sigma \dashv D\cdot U$, which is *not monadic* [31], so the monad $DUFR$ does not have Nom as its Eilenberg-Moore category.
4. Instead, $[\mathbb{I}, \mathsf{Set}] = \mathsf{EM}(DUFR)$ [31], and $F^{\mathbb{I}} \vdash U^{\mathbb{I}}$ is the corresponding adjunction.
5. The induced comparison functor from Nom is itself adjoint: Nom is (equivalent to) the full reflective subcategory of pullback preserving functors (cf. [23, 19]).

Note that $\mathsf{Supp}(\mathbb{A})$ is the only category in Figure 2 that is not a topos, and therefore not a presheaf category.

▶ Remark 5.13 (Relation to named sets). Another notion to reason about names are *named sets* [15], which have been defined in slightly different ways over the years. For a fixed countably infinite set of names $\mathbb{A} = \{v_1, v_2, \ldots\}$ the definitions are as follows:

1. In the definition by Ferrari, Montanari, Pistore [15, Def. 2], every element x of a named set X does not have an explicit support, but is only associated with a natural number $n \in \mathbb{N}$, and so its support is implicitly considered to be $\{v_1, \ldots, v_n\} \subseteq \mathbb{A}$. Moreover, every element is equipped with a subgroup of $\mathfrak{S}_\mathsf{f}(n)$ that describes how the atoms $v_1, \ldots, v_n$ in the support may be permuted. In later works, this natural number $n \in \mathbb{N}$ denoting only the size was replaced with a set of atoms:
2. Montanari and Pistore [36] define a named set to be a set X together with a map $n_X\colon X \to \mathcal{P}(\mathbb{A})$ (without any further information about permutations). A *named function* $m\colon (X, n_X) \to (Y, n_Y)$ is a map on the sets $m\colon X \to Y$ and for each $x \in X$ an injective map $m[x]\colon n_Y(f(x)) \rightarrowtail n_X(x)$. A named set (X, n_X) is called *finitely named* if $n_X(x)$ is finite for every $x \in X$.

   The category of all named sets and named functions is quite different from $\mathsf{Supp}(\mathbb{A})$. For instance, the finitely named set $(\mathbb{A}, n_\mathbb{A})$ with $n_\mathbb{A}(a) = \{a\}$ has infinitely many automorphisms $(\mathbb{A}, n_\mathbb{A}) \to (\mathbb{A}, n_\mathbb{A})$, whereas both in $\mathsf{Nom}(\mathfrak{S}_\mathsf{f}(\mathbb{A}))$ and in $\mathsf{Supp}(\mathbb{A})$, there is only one morphism $\mathbb{A} \to \mathbb{A}$, the identity. Without the finiteness restriction, infinite limits in the category of named sets are Set-like and not Nom-like. More precisely, the infinite product $\prod_{n \in \mathbb{N}}(\mathbb{A}, n_\mathbb{A})$ contains streams making use of infinitely many names.
3. Gaducci, Miculan, and Montanari [23] restrict to *finitely* named sets and additionally equip every element $x \in X$ of a named set (X, n_X) with a subgroup $G_X(x)$ of the group $\mathfrak{S}_\mathsf{f}(n_X(x))$ of all permutations on the names $n_X(x)$. Here, a named function $f\colon (X, n_X, G_X) \to (Y, n_Y, G_Y)$, is a map $f\colon X \to Y$ and additionally for each $x \in X$ a non-empty set of injections $f_x \subseteq (n_Y(f(x)) \rightarrowtail n_X(x))$ satisfying an additional coherence condition involving G_X. Details are not relevant here, because the resulting category NSet was shown to be equivalent to $\mathsf{Nom}(\mathfrak{S}_\mathsf{f}(\mathbb{A}))$ in the same work [23, Prop. 29]. This has a couple of immediate implications regarding the relationship to supported sets:
   - NSet is monadic over $\mathsf{Supp}(\mathbb{A})$.
   - The right-adjoint of this monadic adjunction is a faithful functor $\mathsf{Supp}(\mathbb{A}) \to \mathsf{NSet}$, which is not full.

Having discussed categorical properties of supported sets and its relation to other name-aware notions, we now continue to see how name binding can be accomplished.

6 Name Abstraction and de Bruijn indices

In λ-calculus, the computational steps (β-reduction) essentially consist of a substitution rule $(\lambda x.\,T)\,P \longrightarrow_\beta T[x := P]$ which requires some bound variable names in T to be sufficiently fresh. Thus, it might be necessary to rename those bound variables ($\to_\alpha$) in T before a β step can be performed. But even in such a renaming step $\to_\alpha$, a similar side-condition needs taken care of, because otherwise, the reduction would lead to wrong results.

Thus, there are several approaches to define λ-expressions directly modulo α-equivalence, making substitution total and β-reduction always applicable to λ-expressions containing a reducible expression (redex). In 1972, de Bruijn [11] invented a technique of replacing the variable names with an index (the *de Bruijn index*) that counts the number binders between the variable and its corresponding binder. In 1999, Gabbay and Pitts [19] presented another way to define λ-expressions directly as α-equivalence classes, in which λ-abstraction is a functor on *nominal sets* (called *FM-sets* back then). From now on, we stick to their setting of equality symmetry:

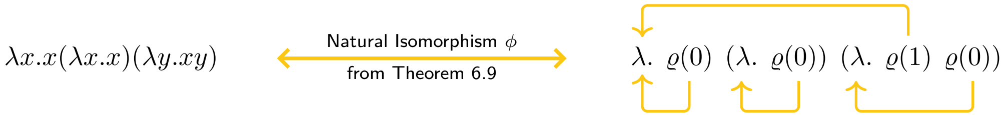

Figure 3 Visualization of de Bruijn indices.

▶ **Assumption 6.1.** *For the rest of the paper, fix $A := \mathbb{A}$ and $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$. Hence, we may assume a bijection $\varrho\colon \mathbb{N} \to \mathbb{A}$ and we simply write* Nom *for the M-nominal sets.*

For the name abstraction functor, the notion of α-equivalence is first defined for arbitrary nominal sets, which is then used in the definition of the abstraction functor:

▶ **Definition 6.2** [19]**.** *For $X \in \mathsf{Nom}$, the equivalence relation $\sim_\alpha$ on $\mathbb{A} \times X$ is defined by*

$$(a, x) \sim_\alpha (b, y) \quad :\Leftrightarrow \quad \exists c \# (a, b, x, y)\colon (c\, a) \cdot x = (c\, b) \cdot y$$

The abstraction functor *$[\mathbb{A}]X\colon \mathsf{Nom} \to \mathsf{Nom}$ is given by $[\mathbb{A}]X = (A \times X)/{\sim_\alpha}$, where $\langle a \rangle x$ denotes the equivalence class of $(a, x) \in \mathbb{A} \times X$.*

In the equivalence class $\langle a \rangle x$, a disappears from the support: $\mathsf{supp}_{[\mathbb{A}]X}(\langle a \rangle x) = \mathsf{supp}_X(x) \setminus \{a\}$.

▶ **Example 6.3** [19]**.** The initial algebra of the functor $\Lambda X = \mathbb{A} + [\mathbb{A}]X + X \times X$ is carried by the nominal set of λ-expressions modulo α-equivalence. The first summand $\mathbb{A}$ describes variables $a \in \mathbb{A}$, the second summand $[\mathbb{A}]X$ describes λ-abstractions $\lambda x.T$, where $x \in \mathbb{A}$ and T is a λ-expression, and the third summand $X \times X$ describes the application $T\,S$ of one λ-expression to one other.

In the present paper, we define an abstraction functor on supported sets for which we require the set of atoms to be a countably infinite set. This functor in fact has a lifting to nominal sets and the lifting is (naturally isomorphic to) the above abstraction functor $[\mathbb{A}]\colon \mathsf{Nom} \to \mathsf{Nom}$ on nominal sets.

The definition of $[\mathbb{A}]$ makes use of the $\mathfrak{S}_{\mathsf{f}}(\mathbb{A})$-action to capture α-equivalence. When introducing abstraction as a functor directly on supported sets, we do not have renaming available, and so we use de Bruijn indices via the bijection $\varrho\colon \mathbb{N} \to \mathbb{A}$ (Assumption 6.1).

▶ **Definition 6.4.** *The* de Bruijn functor *$\mathcal{B}\colon \mathsf{Supp}(\mathbb{A}) \to \mathsf{Supp}(\mathbb{A})$ sends a supported set X to the same set $\mathcal{B}X = X$ but with a different support function. To distinguish elements of X and $\mathcal{B}X$, we write $\lambda.x \in \mathcal{B}X$ for $x \in X$ (i.e. λ is a nameless binder). The support on $\mathcal{B}X$ is*

$$\mathsf{s}_{\mathcal{B}X}\colon \mathcal{B}X \to \mathcal{P}_{\mathsf{f}}(\mathbb{A}) \qquad \mathsf{s}_{\mathcal{B}X}(\lambda.x) := \{\varrho(k) \mid \varrho(k+1) \in \mathsf{s}_X(x), k \in \mathbb{N}\}. \tag{3}$$

A supported map $f\colon X \to Y$ is sent to the same map $\mathcal{B}f\colon \mathcal{B}X \to \mathcal{B}Y$, $\mathcal{B}f(\lambda.x) = \lambda.f(y)$.

The definition of $\mathsf{s}_{\mathcal{B}X}$ captures the idea of de Bruijn indices: we can think of the notation $\lambda.x$ as a lambda abstraction of nameless binder $\lambda.$ of a lambda term x (visualized in Figure 3):

- The variables referring to $\lambda.$ have the de Bruijn index of 0 (at the level of x), because there is no other binder between x and "$\lambda.$". Since $\varrho(0)$ is bound, $\mathsf{s}_{\mathcal{B}X}(x)$ does not depend on whether $\varrho(0) \in \mathsf{s}_X(x)$.
- All other variables $\varrho(k+1) \in \mathsf{s}_X(x)$ refer to variables that are free in $\lambda.x$ and so refer to binders "more above" than $\lambda.x$. A variable $\varrho(k) \in \mathsf{s}_{\mathcal{B}X}(\lambda.x)$ refers to the same binder as the variable $\varrho(k+1) \in \mathsf{s}_X(x)$ under "$\lambda.$", because the latter is one level further down.

▶ **Lemma 6.5.** *$\mathcal{B}\colon \mathsf{Supp}(\mathbb{A}) \to \mathsf{Supp}(\mathbb{A})$ is a functor.*

$$\begin{tikzcd}
\mathsf{EM}(T) \arrow[r, "G"] \arrow[d, "U"'] & \mathsf{EM}(T) \arrow[d, "U"] \\
\mathcal{C} \arrow[r, "H"] & \mathcal{C}
\end{tikzcd}$$

Figure 4 Diagram for Definition 6.6.

$$\begin{tikzcd}
\mathsf{Nom} \arrow[r, "{[\mathbb{A}]}"] \arrow[d, "U"'] & \mathsf{Nom} \arrow[d, "U"] \\
\mathsf{Supp}(\mathbb{A}) \arrow[r, "\mathcal{B}"] & \mathsf{Supp}(\mathbb{A})
\end{tikzcd}$$

Figure 5 Diagram for Theorem 6.9.

We use a slightly generalized notion of a *lifting* of the functor $\mathcal{B}$ to nominal sets:

▶ **Definition 6.6.** *For a monad T on a category $\mathcal{C}$ and a functor $H\colon \mathcal{C} \to \mathcal{C}$, a functor $G\colon \mathsf{EM}(T) \to \mathsf{EM}(T)$ is called a* lifting of H *if HU and UG are naturally isomorphic functors (see Figure 4 for the diagram). We say that a lifting is* strict *if $HU = UG$.*

▶ Remark 6.7. Usually, not just a natural isomorphism but identity is required. Strict liftings $G\colon \mathsf{EM}(T) \to \mathsf{EM}(T)$ are in one-to-one correspondence to distributive laws $TH \to HT$ [25].

This generalization to natural isomorphisms is sound, because they induce strict liftings:

▶ **Lemma 6.8.** *For every natural isomorphism $\phi\colon HU \to UG$, there is a unique strict lifting $\bar{H}\colon \mathsf{EM}(T) \to \mathsf{EM}(T)$ such that $\phi\colon H \to G$ is a natural isomorphism in $\mathsf{EM}(T)$.*

This means that for $(C, \gamma) \in \mathsf{EM}(T)$, $\phi_{(C,\gamma)}$ is a T-algebra isomorphism $\bar{H}(C, \gamma) \to G(C, \gamma)$.

▶ **Theorem 6.9.** *The abstraction functor $[\mathbb{A}]\colon \mathsf{Nom} \to \mathsf{Nom}$ is a lifting of the de Bruijn functor $\mathcal{B}\colon \mathsf{Supp}(\mathbb{A}) \to \mathsf{Supp}(\mathbb{A})$ (see Figure 5 for the diagram). That is, there is a natural isomorphism $\phi\colon \mathcal{B}U \longrightarrow U[\mathbb{A}]$.*

Intuitively, ϕ translates between the de Bruijn indices and the nominal abstraction functor: in the term $\lambda.x \in \mathcal{B}X$, we have $\varrho(0)$ implicitly bound, like it is in $\langle\varrho(0)\rangle(x) \in [\mathbb{A}]X$. However, every $\varrho(k+1)$ in x appears as $\varrho(k)$ in the support of $\lambda.x$, so ϕ essentially renames $\varrho(\ell+1) \mapsto \varrho(\ell)$ in x for all $\ell \geq 1$.

▶ Remark 6.10 (Abstraction in named sets). A de Bruijn-style abstraction functor can also defined directly on Nom [9, Def. 3.3]. The defining property (3) of abstraction on $\mathsf{Supp}(\mathbb{A})$ then reappears as a theorem of the support function supp in nominal sets [9, Thm. 4.1]. This adheres to the principle that in $\mathsf{Supp}(\mathbb{A})$, the support is part of the data, whereas in nominal sets, supp is a derived notion. Since named sets are equivalent to nominal sets, we have name abstraction there, too, and an explicit definition is provided by Ciancia and Montanari [9, Sect. 7.1].

▶ Remark 6.11 (Abstraction in presheaves). Fiore et al. [16] also use de Bruijn indices in their abstraction functor, but treat support in a different way. They consider presheaves $X \in \mathsf{Set}^{\mathbb{F}}$ where $\mathbb{F}$ is the full subcategory of Set containing only natural numbers as objects (considering natural numbers as finite cardinals). Thus for every natural number $n \in \mathbb{N}$, the set $X(n)$ intuitively denotes the elements that make only use of the atoms $\varrho(0), \dots, \varrho(n-1) \in \mathbb{A}$. For each map $\pi\colon n \to m$, the map $X(\pi)\colon X(n) \to X(m)$ renames embedded variables. Their name abstraction functor $\delta\colon \mathsf{Set}^{\mathbb{F}} \to \mathsf{Set}^{\mathbb{F}}$ sends a presheaf X to the presheaf $\delta(X)(n) = X(n+1)$, implicitly binding $\varrho(0)$, and "shifting" the role of the other atoms. Despite this similarity, it is unclear whether there is a formal categorical connection between $\mathsf{Supp}(A)$ and $\mathsf{Set}^{\mathbb{F}}$ or between $\mathcal{B}$ and δ.

With name binding in supported sets, we can now study automata in supported sets and relate them to nominal automata.

7 Register Automata to Nominal Automata

The well-known powerset construction for automata is an instance of a more general principle called *generalized determinization* [40, 6] that *internalizes side-effects* modelled by a monad. The input of the standard powerset construction is a *non-deterministic* finite automaton (NFA), which can be understood as a deterministic automaton extended with non-deterministic branching as a side-effect. With Q as the set of states of the NFA, the construction returns a deterministic automaton with states $\mathcal{P}_{\mathsf{f}}Q$, which happens to be precisely the set of *configurations* of the original NFA. This construction generalizes from the (finite) powerset monad $\mathcal{P}_{\mathsf{f}}$ to arbitrary monads $T\colon \mathcal{C} \to \mathcal{C}$ and from automata to state-based systems modelled via coalgebras for a functor $H\colon \mathcal{C} \to \mathcal{C}$. We apply this principle to the monad from the monadicity result in Section 5.

A *coalgebra* (for $H\colon \mathcal{C} \to \mathcal{C}$) is an object $Q \in \mathcal{C}$ together with a morphism $Q \to HQ$. E.g. an H-coalgebra for $H\colon \mathsf{Set} \to \mathsf{Set}$, $HX = 2 \times X^{\Sigma}$, is a deterministic automaton, i.e. a set Q together with a map $d\colon Q \to 2 \times Q^{\Sigma}$. For a state $q \in Q$, the first component $\mathsf{pr}_1(d(q)) \in 2$ specifies the finality of q, and the second component $\mathsf{pr}_2(d(q)) \in Q^{\Sigma}$ sends input symbols $a \in \Sigma$ to successor states in Q (the initial state $q_0 \in Q$ is not important here).

On the other hand, a non-deterministic automaton is simply a coalgebra for the composed functor $H\mathcal{P}_{\mathsf{f}}\colon \mathsf{Set} \to \mathsf{Set}$, i.e. a map $c\colon Q \to 2 \times (\mathcal{P}_{\mathsf{f}}Q)^{\Sigma}$. The generalized determinization assumes that $T\colon \mathcal{C} \to \mathcal{C}$ is a monad (with unit η and multiplication μ) and that $H\colon \mathcal{C} \to \mathcal{C}$ is a functor that lifts to the Eilenberg-Moore category of T, i.e. that we have a functor $G\colon \mathsf{EM}(T) \to \mathsf{EM}(T)$ with $\phi\colon HU \xrightarrow{\cong} UG$ (Definition 6.6) – for example $HX = 2 \times X^{\Sigma}$ and $TX = \mathcal{P}_{\mathsf{f}}X$. Then the generalized determinization turns an HT-coalgebra on Q into a G-coalgebra on TQ, using the adjunction $F \dashv U$ between $\mathcal{C}$ and $\mathsf{EM}(T)$ (cf (2)):

$$c\colon Q \to HTQ \quad \text{in } \mathcal{C} \quad \xmapsto{\text{Internalization}} \quad d\colon FQ \to GFQ \quad \text{in } \mathsf{EM}(T)$$

In the instance $TX = \mathcal{P}_{\mathsf{f}}X$ of the powerset construction, $\mathsf{EM}(T)$ is the category of join-semilattices, and every non-deterministic automaton on Q is turned into a deterministic automaton on $\mathcal{P}_{\mathsf{f}}Q$. Since we internalized the side-effect of non-deterministic branching in the states, the resulting state space $\mathcal{P}_{\mathsf{f}}Q$ is the configuration space of the non-deterministic automaton and the induced transition structure d preserves joins. Instantiating the generalized determinization to supported sets and nominal sets yields a construction that internalizes the side-effect of rearranging atoms or registers. Here, we stick to the equality symmetry $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$, $TX := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet X$, i.e. we stick to Assumption 6.1.

▶ **Construction 7.1.** *Fix $M := \mathfrak{S}_{\mathsf{f}}(\mathbb{A})$. For functors $H\colon \mathsf{Supp}(\mathbb{A}) \to \mathsf{Supp}(\mathbb{A})$ and $G\colon \mathsf{Nom} \to \mathsf{Nom}$ with $HU \cong UG$, we have the internalization process*

$$c\colon Q \to H(M\bullet Q) \quad \textit{in } \mathsf{Supp}(\mathbb{A}) \quad \xmapsto{\text{Internalization}} \quad d\colon M\bullet Q \to G(M\bullet Q) \quad \textit{in } \mathsf{Nom}.$$

Here, the nominal set $M\bullet Q$ is the configuration space (states + register assignments) of the original automaton (Q, c). The configuration $\mathsf{id}_A \cdot q$ in $M\bullet Q$ behaves like $q \in Q$ in c, and the resulting transition structure d is equivariant. We can apply this construction to many different system types H and G that arise from the following functors. Here, $\mathcal{P}_{\mathsf{ufs}}X$ (for a nominal set X) contains those subsets $S \subseteq X$ for which $\bigcup\{\mathsf{supp}(x) \mid x \in S\}$ is finite [39].

▶ **Proposition 7.2.** *The functors G on Nom that are the lifting of a functor H on $\mathsf{Supp}(\mathbb{A})$ contain $\mathcal{P}_{\mathsf{f}}$, $\mathcal{P}_{\mathsf{ufs}}$, $[\mathbb{A}]$, and all constant functors and are closed under all (possibly infinite) products and coproducts.*

▶ **Example 7.3.** All Nom-functors arising from binding signatures [30, 16] are such liftings. In particular, $\Lambda X = \mathbb{A} + [\mathbb{A}]X + X \times X$ (Example 6.3) is the lifting of $HX = \mathbb{A} + \mathcal{B}X + X \times X$. The coalgebras of Λ are possibly infinite λ-trees modulo α-equivalence [30]. Such a λ-tree can then be represented by a supported set X and a supported map

$$f\colon X \to \mathbb{A} \;+\; \mathcal{B}(\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet X) \;+\; (\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet X) \times (\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet X)$$

▶ **Example 7.4.** Nominal automata with name binding considered by Schröder et al. [39] are coalgebras for the Nom-functor $KX = 2 \times \mathcal{P}_{\mathsf{ufs}}([\mathbb{A}]X + \mathbb{A} \times X)$ which is a lifting of $HX = 2 \times \mathcal{P}_{\mathsf{ufs}}(\mathcal{B}X + \mathbb{A} \times X)$ on supported sets. Thus, these nominal automata can be represented by finite coalgebras in $\mathsf{Supp}(\mathbb{A})$.

Interpreting $\mathbb{A}$ as the names of registers, register automata in the style of Cassel et al. [8] straightforwardly adapt to HT-coalgebras in supported sets:

▶ **Example 7.5.** Let $\mathcal{G}$ be a nominal set of *guards*, where we understand $g \in \mathcal{G}$ as a predicate involving register names $\mathsf{supp}_G(g) \in \mathcal{P}_{\mathsf{f}}(\mathbb{A})$, e.g. $\mathsf{iszero}(a)$, $\mathsf{divides}(a, b)$, $\mathsf{plus}(a, b, c)$. It is important that the atoms $a \in \mathbb{A}$ are the register names, and not the data itself. So e.g. $\mathsf{iszero}(a_3)$ is satisfied if the data value stored in register a_3 is zero – this is the *symbolic* semantics of register automata [44, 45]. Using the forgetful $U\colon \mathsf{Nom} \to \mathsf{Supp}(\mathbb{A})$, we can also use $\mathcal{G}$ as a supported set where the support of $g \in \mathcal{G}$ is the set of register names $a \in \mathbb{A}$ appearing in g. Now, a register automaton is a HT-coalgebra in $\mathsf{Supp}(A)$

$$c\colon Q \to 2 \times \mathcal{B}\mathcal{P}_{\mathsf{f}}(U\mathcal{G} \times \mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet Q) \qquad \text{for } HX = 2 \times \mathcal{B}\mathcal{P}_{\mathsf{f}}(\mathcal{G} \times X) \text{ and } TX = \mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet X.$$

As before, the first component of $c(q)$ defines the finality of a state $q \in Q$. While in state q, the registers $\mathsf{s}(q) \subseteq \mathbb{A}$ are filled with data and the second component of $c(q)$ binds the next input data symbol into a fresh register ($\mathcal{B}$) and then provides a finite number of transitions of the form $q \xrightarrow{g,\pi} q'$. In such a transition, the guard $g \in \mathcal{G}$, specifies when this transition can be taken, depending on the register contents of q and the freshly read input symbol. When a transition is taken, the registers are rearranged via the permutation π before state q' is entered. The map $\pi\colon \mathsf{s}(q') \rightarrowtail \mathbb{A}$ specifies for each register $r \in \mathsf{s}(q')$ of the target set, where the data symbol is drawn from. The map c being a supported map ensures that whenever we transfer to state q', then all registers $\mathsf{s}(q')$ can be filled with data, coming for the support of the previous state q or from the "input register" in $\mathcal{B}$ (a concrete register automaton is discussed in Example 7.6). The internalization process turns c into an equivariant map $d\colon \bar{Q} \to 2 \times [\mathbb{A}]\mathcal{P}_{\mathsf{f}}(\mathcal{G} \times \bar{Q})$, which is the nominal automaton of configurations $\bar{Q} = \mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet Q$.

The construction also nicely interacts with *initial* states. An initial state of a register automaton is simply a supported map $i\colon 1 \to Q$, that is, an element of Q with empty support. Applying the functor $\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet$ to i yields an equivariant map, $\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet i\colon \mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet 1 \to \bar{Q}$. Since the only element of $1 = \{0\}$ has empty support, we have $\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet 1 \cong 1$, so $\mathfrak{S}_{\mathsf{f}}(\mathbb{A})\bullet i$ is equivalent to an equivariant map $i'\colon 1 \to \bar{Q}$.

▶ **Example 7.6.** An example register automaton is visualized in Figure 6: the upper half of the nodes provide the state names $Q = \{q_0, q_1, q_2\}$, the lower half specifies their support $\mathsf{s}_Q(q_0) = \emptyset$, $\mathsf{s}_Q(q_1) = \{\ell\}$, $\mathsf{s}_Q(q_2) = \{s\}$, where $\ell, s \in \mathbb{A}, \ell \neq s$ are arbitrary (standing for ℓast and $s$econd last). The initial state is q_0, i.e. the supported map $i\colon 1 \to Q$ is $i(0) = q_0$, and all states are final.We mimic existing register automata notation [8, 44] by defining $p := \varrho(0) \in \mathbb{A}$ (note that we do *not* require ℓ, s to be distinct from p). Also, we use $\mathsf{old}\colon \mathbb{A} \to \mathbb{A}$ defined by $\mathsf{old}(\varrho(k)) = \varrho(k + 1)$, which satisfies $a \in \mathsf{s}_{\mathcal{B}X}(\lambda.x)$ iff $\mathsf{old}(a) \in \mathsf{s}_X(x)$ for all supported sets X and $x \in X$. The nominal set of guards is defined by

$$\mathcal{G} := \{\top\} + \{\mathsf{plus}\} \times \mathbb{A}^3,$$

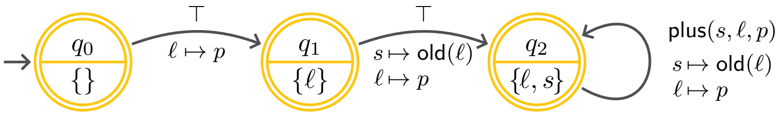

Figure 6 Example of a (symbolic) register automaton.

so the constant $\top$ represents “true” and the ternary relation symbol $\mathsf{plus}(a,b,c)$ represents that the sum of the register contents of a and b equals the register content of c. Since we use symbolic semantics, the concrete data domain does not need to be specified here; but of course one can interpret plus over rational numbers for example. The supported map $c\colon Q \to 2 \times \mathcal{BP}_{\mathsf{f}}(U\mathcal{G} \times \mathfrak{S}_{\mathsf{f}}[\mathbb{A}]Q)$ representing the automaton is sincerely visualized in Figure 6:

- The transition $q_0 \xrightarrow{g,\pi} q_1$ has the guard $g = \top$ and the register reassignment $\pi\colon \mathsf{s}(q_1) \rightarrowtail \mathbb{A}$ is defined by $\pi(\ell) = p$, meaning that when entering state q_1, the register ℓ will be filled with what was in the input p before: $c(q_0) = (1, \lambda.\{(\top,(\pi,q_1))\})$. This satisfies support preservation because $\mathsf{s}(\top,(\pi,q_1)) = \pi[\mathsf{s}(q_1)] = \{p\}$ and so $\mathsf{s}(\lambda.(\top,(\pi,q_1))) = \emptyset \subseteq \mathsf{s}(q_0)$.
- The transition $q_1 \xrightarrow{g,\sigma} q_2$ has the same guard $g = \top$ and $\sigma\colon \mathsf{s}(q_2) \rightarrowtail \mathbb{A}$ is defined by $\sigma(s) = \mathsf{old}(\ell)$ and $\sigma(\ell) = p$. Again, $c(q_1) = (1, \lambda.\{(\top,(\sigma,q_2))\})$ preserves support because
$$\mathsf{s}(\top,(\sigma,q_2)) = \sigma[\mathsf{s}(q_2)] = \sigma[\{\ell,s\}] = \{p,\mathsf{old}(\ell)\} \quad\text{and}\quad \mathsf{s}(\lambda.\{(\top,(\sigma,q_2))\}) = \{\ell\} \subseteq \mathsf{s}(q_1).$$
- The loop $q_2 \xrightarrow{g',\sigma} q_2$ has the guard $g' = \mathsf{plus}(s,\ell,p)$ and literally the same $\sigma\colon \mathsf{s}(q_2) \rightarrowtail \mathbb{A}$ as in the previous transition. Support preservation holds for the mapping $c(q_1) = (1, \lambda.\{(\top,(\sigma,q_2))\})$, because
$$\mathsf{s}(\mathsf{plus}(s,\ell,p),(\sigma,q_2)) = \{\mathsf{old}(s),\mathsf{old}(\ell),p\} \text{ and } \mathsf{s}(\lambda.\{(\mathsf{plus}(s,\ell,p),(\sigma,q_2))\}) = \{s,\ell\} \subseteq \mathsf{s}(q_2).$$

The coherence axioms of register automata naturally translate into c being a supported map. Construction 7.1 transforms this finite coalgebra in $\mathsf{Supp}(\mathbb{A})$ into an nominal automaton, in the style of symbolic semantics [44] of register automata.

Not only in this example, but also in general, Construction 7.1 preserves finite presentability. So every finite coalgebra in $\mathsf{Supp}(\mathbb{A})$ is turned into an orbit-finite coalgebra in Nom. The semantics of orbit-finite coalgebras can be characterized for many functors [33] that arise from the examples listed here (Proposition 7.2).

8 Conclusions and Future Work

We have seen that by going from the base category of sets to supported sets, nominal sets for various symmetries surprisingly turn out to be monadic. Supported sets have a functor for name binding, which even lifts to the abstraction functor in nominal sets. It remains for future work whether a similar name binding functor can be found for other data alphabets, most notably for the total order symmetry on $\mathbb{Q}$, and whether multiple atoms can be bound simultaneously, as it is possible in nominal sets [10]. It can be conjectured that such generalizations are not possible on the level of supported sets.

On the positive side, due to the little structure of supported sets, it provides a common foundation for the nominal sets for different symmetries, in the sense of being described by monads on supported sets. The monadicity can be used to relate nominal automata with register automata, which have a natural definition in supported sets. It remains for future investigation how the *data semantics* of register automata can be phrased in supported sets. We are optimistic that it helps to develop a categorical semantics for register automata for data alphabets and signatures beyond symmetries (e.g. those for priority queues [8]). When developing algorithms, in particular learning algorithms and minimization algorithms, for register or nominal automata [8, 43, 35], supported sets directly yield a finite representation that can help in the implementation and complexity analysis. Also for other algorithmic tasks such as efficient coalgebraic minimization [12, 14, 24, 28, 48, 49], supported sets can help to canonically represent the nominal coalgebras subject to minimization.

References

1 Jiří Adámek, Stefan Milius, Lurdes Sousa, and Thorsten Wißmann. Finitely presentable algebras for finitary monads. *Theory and Applications of Categories*, 34(37):1179–1195, November 2019. URL: `http://www.tac.mta.ca/tac/volumes/34/37/34-37abs.html`.

2 Jiří Adámek, Horst Herrlich, and George E. Strecker. Abstract and concrete categories. the joy of cats, 2004.

3 Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994.

4 Steve Awodey. *Category Theory*. Oxford Logic Guides. OUP Oxford, 2010.

5 Brian E. Aydemir, Aaron Bohannon, and Stephanie Weirich. Nominal reasoning techniques in coq: (extended abstract). *Electron. Notes Theor. Comput. Sci.*, 174(5):69–77, 2007. `doi:10.1016/j.entcs.2007.01.028`.

6 Falk Bartels. *On generalized coinduction and probabilistic specification formats: Distributive laws in coalgebraic modelling*. PhD thesis, Vrije Universiteit Amsterdam, 2004.

7 Mikolaj Bojanczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10, 2014. `doi:10.2168/LMCS-10(3:4)2014`.

8 Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Extending automata learning to extended finite state machines. In Amel Bennaceur, Reiner Hähnle, and Karl Meinke, editors, *Machine Learning for Dynamic Software Analysis: Potentials and Limits – International Dagstuhl Seminar 16172*, volume 11026 of *LNCS*, pages 149–177. Springer, 2018. `doi:10.1007/978-3-319-96562-8_6`.

9 Vincenzo Ciancia and Ugo Montanari. A name abstraction functor for named sets. In Jirí Adámek and Clemens Kupke, editors, *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008*, volume 203, 5 of *Electronic Notes in Theoretical Computer Science*, pages 49–70. Elsevier, 2008. `doi:10.1016/j.entcs.2008.05.019`.

10 Ranald Clouston. Generalised name abstraction for nominal sets. In Frank Pfenning, editor, *Foundations of Software Science and Computation Structures – 16th International Conference, FOSSACS 2013*, volume 7794 of *Lecture Notes in Computer Science*, pages 434–449. Springer, 2013. `doi:10.1007/978-3-642-37075-5_28`.

11 N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972. `doi:10.1016/1385-7258(72)90034-0`.

12 Hans-Peter Deifel, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Generic partition refinement and weighted tree automata. In *Formal Methods – The Next 30 Years, Proc. 3rd World Congress on Formal Methods (FM 2019)*, volume 11800 of *LNCS*, pages 280–297. Springer, October 2019.

13 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. `doi:10.1145/1507244.1507246`.

14 Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Efficient coalgebraic partition refinement. In *Proc. 28th International Conference on Concurrency Theory (CONCUR 2017)*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017.

15 Gian Luigi Ferrari, Ugo Montanari, and Marco Pistore. Minimizing transition systems for name passing calculi: A co-algebraic formulation. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2002. `doi:10.1007/3-540-45931-6_10`.

16 Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding (extended abstract). In *Proc. 14$^{\text{th}}$ LICS Conf.*, pages 193–202. IEEE, Computer Society Press, 1999.

17 Marcelo P. Fiore and Sam Staton. Comparing operational models of name-passing process calculi. *Inf. Comput.*, 204(4):524–560, 2006. doi:10.1016/j.ic.2005.08.004.

18 Murdoch Gabbay and James Cheney. A sequent calculus for nominal logic. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 139–148. IEEE Computer Society, 2004. doi:10.1109/LICS.2004.1319608.

19 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 214–224. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782617.

20 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects Comput.*, 13(3-5):341–363, 2002. doi:10.1007/s001650200016.

21 Murdoch James Gabbay and Martin Hofmann. Nominal renaming sets. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008*, volume 5330 of *LNCS*, pages 158–173. Springer, 2008. doi:10.1007/978-3-540-89439-1_11.

22 Peter Gabriel and Friedrich Ulmer. *Lokal präsentierbare Kategorien*, volume 221 of *Lecture Notes Math.* Springer-Verlag, 1971.

23 Fabio Gadducci, Marino Miculan, and Ugo Montanari. About permutation algebras, (pre)sheaves and named sets. *Higher Order Symbol. Comput.*, 19(2-3):283–304, September 2006. doi:10.1007/s10990-006-8749-3.

24 Jules Jacobs and Thorsten Wißmann. Fast coalgebraic bisimilarity minimization. In *Principles of Programming Languages, POPL '23.* ACM, 2023. to appear. URL: https://arxiv.org/abs/2204.12368.

25 Peter T. Johnstone. Adjoint Lifting Theorems for Categories of Algebras. *Bull. London Math. Soc.*, 7(3):294–297, November 1975.

26 Peter T Johnstone. *Sketches of an elephant: a Topos theory compendium.* Oxford logic guides. Oxford Univ. Press, New York, NY, 2002.

27 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

28 Barbara König and Sebastian Küpper. Generic partition refinement algorithms for coalgebras and an instantiation to weighted automata. In *Theoretical Computer Science, IFIP TCS 2014*, volume 8705 of *LNCS*, pages 311–325. Springer, 2014. doi:10.1007/978-3-662-44602-7.

29 Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan, and Alexandra Silva. Nominal kleene coalgebra. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *ICALP 2015, Proceedings*, volume 9135 of *LNCS*, pages 286–298. Springer, 2015. doi:10.1007/978-3-662-47666-6_23.

30 Alexander Kurz, Daniela Petrisan, Paula Severi, and Fer-Jan de Vries. Nominal coalgebraic data types with applications to lambda calculus. *Logical Methods in Computer Science*, 9(4), 2013. doi:10.2168/LMCS-9(4:20)2013,http://arxiv.org/abs/1311.1395.

31 Alexander Kurz, Daniela Petrisan, and Jiri Velebil. Algebraic theories over nominal sets. *CoRR*, abs/1006.3027, 2010. URL: http://arxiv.org/abs/1006.3027, arXiv:1006.3027.

32 Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer New York, 1998. URL: http://books.google.de/books?id=eBvhyc4z8HQC.

33 Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Regular behaviours with names. *Applied Categorical Structures*, 24(5):663–701, 2016. doi:10.1007/s10485-016-9457-8.

34 Joshua Moerman and Jurriaan Rot. Separation and Renaming in Nominal Sets. In Maribel Fernández and Anca Muscholl, editors, *CSL 2020*, volume 152 of *LIPIcs*, pages 31:1–31:17, Dagstuhl, Germany, 2020. LIPIcs. doi:10.4230/LIPIcs.CSL.2020.31.

35 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In Giuseppe Castagna and Andrew D. Gordon, editors, *POPL 2017*, pages 613–625. ACM, 2017. doi:10.1145/3009837.3009879.

36 Ugo Montanari and Marco Pistore. History-dependent automata: An introduction. In Marco Bernardo and Alessandro Bogliolo, editors, *SFM-Moby 2005: Formal Methods for Mobile Computing*, volume 3465 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2005. doi:10.1007/11419822_1.

37 Daniela Petrişan. *Investigations into Algebra and Topology over Nominal Sets*. dissertation, University of Leicester, 2011.

38 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.

39 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. In Javier Esparza and Andrzej Murawski, editors, *FoSSaCS 2017*, volume 10203 of *LNCS*, pages 124–142. Springer, 2017. doi:10.1007/978-3-662-54458-7_8.

40 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Log. Methods Comput. Sci.*, 9(1), 2013. doi:10.2168/LMCS-9(1:9)2013.

41 Sam Staton. Name-passing process calculi: operational models and structural operational semantics. Technical Report UCAM-CL-TR-688, University of Cambridge, Computer Laboratory, June 2007. doi:10.48456/tr-688.

42 Christian Urban and Christine Tasson. Nominal techniques in isabelle/hol. In Robert Nieuwenhuis, editor, *CADE-20*, volume 3632 of *LNCS*, pages 38–53. Springer, 2005. doi:10.1007/11532231_4.

43 Henning Urbat and Lutz Schröder. Automata learning: An algebraic approach. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 900–914. ACM, 2020. doi:10.1145/3373718.3394775.

44 Frits W. Vaandrager and Abhisek Midya. A myhill-nerode theorem for register automata and symbolic trace languages. In Violet Ka I Pun, Volker Stolz, and Adenilso Simão, editors, *ICTAC 2020*, volume 12545 of *LNCS*, pages 43–63. Springer, 2020. doi:10.1007/978-3-030-64276-1_3.

45 Frits W. Vaandrager and Abhisek Midya. A myhill-nerode theorem for register automata and symbolic trace languages. *Theor. Comput. Sci.*, 912:37–55, 2022. doi:10.1016/j.tcs.2022.01.015.

46 David Venhoek, Joshua Moerman, and Jurriaan Rot. Fast computations on ordered nominal sets. In Bernd Fischer and Tarmo Uustalu, editors, *ICTAC 2018*, volume 11187 of *LNCS*, pages 493–512. Springer, 2018. doi:10.1007/978-3-030-02508-3_26.

47 David Venhoek, Joshua Moerman, and Jurriaan Rot. Fast computations on ordered nominal sets. *Theor. Comput. Sci.*, 935:82–104, 2022. doi:10.1016/j.tcs.2022.09.002.

48 Thorsten Wißmann, Hans-Peter Deifel, Stefan Milius, and Lutz Schröder. From generic partition refinement to weighted tree automata minimization. *Formal Aspects of Computing*, pages 1–33, March 2021. doi:10.1007/s00165-020-00526-z.

49 Thorsten Wißmann, Ulrich Dorsch, Stefan Milius, and Lutz Schröder. Efficient and modular coalgebraic partition refinement. *Logical Methods in Computer Science*, 16:1:8:1–8:63, January 2020. doi:10.23638/LMCS-16(1:8)2020.