Fast Computations on Ordered Nominal Sets

David Venhoek, Joshua Moerman, and Jurriaan Rot

Institute for Computing and Information Sciences,
Radboud Universiteit, Nijmegen, The Netherlands
david@venhoek.nl, joshua.moerman@cs.ru.nl, jrot@cs.ru.nl

Abstract. We show how to compute efficiently with nominal sets over the total order symmetry, by developing a direct representation of such nominal sets and basic constructions thereon. In contrast to previous approaches, we work directly at the level of orbits, which allows for an accurate complexity analysis. The approach is implemented as the library Ons (Ordered Nominal Sets).
Our main motivation is nominal automata, which are models for recognising languages over infinite alphabets. We evaluate Ons in two applications: minimisation of automata and active automata learning. In both cases, Ons is competitive compared to existing implementations and outperforms them for certain classes of inputs.

1 Introduction

Automata over infinite alphabets are natural models for programs with unbounded data domains. Such automata, often formalised as *register automata*, are applied in modelling and analysis of communication protocols, hardware, and software systems (see [4,10,15,16,22,26] and references therein). Typical infinite alphabets include sequence numbers, timestamps, and identifiers. This means one can model data flow in such automata beside the basic control flow provided by ordinary automata. Recently, it has been shown in a series of papers that such models are amenable to learning [1,6,7,11,21,29] with the verification of (closed source) TCP implementations as a prominent example [13].

A foundational approach to infinite alphabets is provided by the notion of *nominal set*, originally introduced in computer science as an elegant formalism for name binding [14,25]. Nominal sets have been used in a variety of applications in semantics, computation, and concurrency theory (see [24] for an overview). Bojańczyk et al. introduce *nominal automata*, which allow one to model languages over infinite alphabets with different symmetries [4]. Their results are parametric in the structure of the data values. Important examples of data domains are ordered data values (e.g., timestamps) and data values that can only be compared for equality (e.g., identifiers). In both data domains, nominal automata and register automata are equally expressive [4].

Important for applications of nominal sets and automata are implementations. A couple of tools exist to compute with nominal sets. Notably, Nλ [17] and Lois [18,19] provide a general purpose programming language to manipulate

infinite sets.[1] Both tools are based on SMT solvers and use logical formulas to represent the infinite sets. These implementations are very flexible, and the SMT solver does most of the heavy lifting, which makes the implementations themselves relatively straightforward. Unfortunately, this comes at a cost as SMT solving is in general PSPACE-hard. Since the formulas used to describe sets tend to grow as more calculations are done, running times can become unpredictable.

In the current paper, we use a direct representation, based on symmetries and orbits, to represent nominal sets. We focus on the *total order symmetry*, where data values are rational numbers and can be compared for their order. Nominal automata over the total order symmetry are more expressive than automata over the equality symmetry (i.e., traditional register automata [16]). A key insight is that the representation of nominal sets from [4] becomes rather simple in the total order symmetry; each orbit is presented solely by a natural number, intuitively representing the number of variables or registers.

Our main contributions include the following.

- We develop the *representation theory* of nominal sets over the total order symmetry. We give concrete representations of nominal sets, their products, and equivariant maps.
- We provide *time complexity bounds* for operations on nominal sets such as intersections and membership. Using those results we give the time complexity of Moore's minimisation algorithm (generalised to nominal automata) and prove that it is polynomial in the number of orbits.
- Using the representation theory, we are able to *implement nominal sets in a C++ library* ONS. The library includes all the results from the representation theory (sets, products, and maps).
- We *evaluate the performance* of ONS and compare it to Nλ and LOIS, using two algorithms on nominal automata: minimisation [5] and automata learning [21]. We use randomly generated automata as well as concrete, logically structured models such as FIFO queues. For random automata, our methods are drastically faster than the other tools. On the other hand, LOIS and Nλ are faster in minimising the structured automata as they exploit their logical structure. In automata learning, the logical structure is not available a-priori, and ONS is faster in most cases.

The structure of the paper is as follows. Section 2 contains background on nominal sets and their representation. Section 3 describes the concrete representation of nominal sets, equivariant maps and products in the total order symmetry. Section 4 describes the implementation ONS with complexity results, and Section 5 the evaluation of ONS on algorithms for nominal automata. Related work is discussed in Section 6, and future work in Section 7.

[1] Other implementations of nominal techniques that are less directly related to our setting (Mihda, Fresh OCaml, and Nominal Isabelle) are discussed in Section 6.

2 Nominal sets

Nominal sets are infinite sets that carry certain symmetries, allowing a finite representation in many interesting cases. We recall their formalisation in terms of group actions, following [4,24], to which we refer for an extensive introduction.

Group actions. Let G be a group and X be a set. A *(right) G-action* is a function $\cdot\colon X \times G \to X$ satisfying $x \cdot 1 = x$ and $(x \cdot g) \cdot h = x \cdot (gh)$ for all $x \in X$ and $g, h \in G$. A set X with a G-action is called a *G-set* and we often write xg instead of $x \cdot g$. The *orbit* of an element $x \in X$ is the set $\{xg \mid g \in G\}$. A G-set is always a disjoint union of its orbits (in other words, the orbits partition the set). We say that X is *orbit-finite* if it has finitely many orbits, and we denote the number of orbits by $N(X)$.

A map $f\colon X \to Y$ between G-sets is called *equivariant* if it preserves the group action, i.e., for all $x \in X$ and $g \in G$ we have $f(x)g = f(xg)$. If an equivariant map f is bijective, then f is an *isomorphism* and we write $X \cong Y$. A subset $Y \subseteq X$ is equivariant if the corresponding inclusion map is equivariant. The *product* of two G-sets X and Y is given by the Cartesian product $X \times Y$ with the pointwise group action on it, i.e., $(x, y)g = (xg, yg)$. Union and intersection of X and Y are well-defined if the two actions agree on their common elements.

Nominal sets. A *data symmetry* is a pair $(\mathcal{D}, G)$ where $\mathcal{D}$ is a set and G is a subgroup of $\mathrm{Sym}(\mathcal{D})$, the group of bijections on $\mathcal{D}$. Note that the group G naturally acts on $\mathcal{D}$ by defining $xg = g(x)$. In the most studied instance, called the *equality symmetry*, $\mathcal{D}$ is a countably infinite set and $G = \mathrm{Sym}(\mathcal{D})$. In this paper, we will mostly focus on the *total order symmetry* given by $\mathcal{D} = \mathbb{Q}$ and $G = \{\pi \mid \pi \in \mathrm{Sym}(\mathbb{Q}), \pi \text{ is monotone}\}$.

Let $(\mathcal{D}, G)$ be a data symmetry and X be a G-set. A set of data values $S \subseteq \mathcal{D}$ is called a *support* of an element $x \in X$ if for all $g \in G$ with $\forall s \in S : sg = s$ we have $xg = x$. A G-set X is called *nominal* if every element $x \in X$ has a finite support.

Example 1. We list several examples for the total order symmetry. The set $\mathbb{Q}^2$ is nominal as each element $(q_1, q_2) \in \mathbb{Q}^2$ has the finite set $\{q_1, q_2\}$ as its support. The set has the following three orbits: $\{(q_1, q_2) \mid q_1 < q_2\}$, $\{(q_1, q_2) \mid q_1 > q_2\}$ and $\{(q_1, q_2) \mid q_1 = q_2\}$.

For a set X, the set of all subsets of size $n \in \mathbb{N}$ is denoted by $\mathcal{P}_n(X) = \{Y \subseteq X \mid \#Y = n\}$. The set $\mathcal{P}_n(\mathbb{Q})$ is a single-orbit nominal set for each n, with the action defined by direct image: $Yg = \{yg \mid y \in Y\}$. The group of monotone bijections also acts by direct image on the full power set $\mathcal{P}(\mathbb{Q})$, but this is *not* a nominal set. For instance, the set $\mathbb{Z} \in \mathcal{P}(\mathbb{Q})$ of integers has no finite support.

If $S \subseteq \mathcal{D}$ is a support of an element $x \in X$, then any set $S' \subseteq \mathcal{D}$ such that $S \subseteq S'$ is also a support of x. A set $S \subseteq \mathcal{D}$ is a *least support* of $x \in X$ if it is a support of x and $S \subseteq S'$ for any support S' of x. The existence of least supports

is crucial for representing orbits. Unfortunately, even when elements have a finite support, in general they do not always have a least support. A data symmetry $(\mathcal{D}, G)$ is said to *admit least supports* if every element of every nominal set has a least support. Both the equality and the total order symmetry admit least supports. (See [4] for other (counter)examples of data symmetries admitting least supports.) Having least supports is useful for a finite representation.

Given a nominal set X, the size of the least support of an element $x \in X$ is denoted by $\dim(x)$, the *dimension* of x. We note that all elements in the orbit of x have the same dimension. For an orbit-finite nominal set X, we define $\dim(X) = \max\{\dim(x) \mid x \in X\}$. For a single-orbit set O, observe that $\dim(O) = \dim(x)$ where x is any element $x \in O$.

2.1 Representing nominal orbits

We represent nominal sets as collections of single orbits. The finite representation of single orbits is based on the theory of [4], which uses the technical notions of *restriction* and *extension*. We only briefly report their definitions here. However, the reader can safely move to the concrete representation theory in Section 3 with only a superficial understanding of Theorem 2 below.

The *restriction* of an element $\pi \in G$ to a subset $C \subseteq \mathcal{D}$, written as $\pi|_C$, is the restriction of the function $\pi : \mathcal{D} \to \mathcal{D}$ to the domain C. The restriction of a group G to a subset $C \subseteq \mathcal{D}$ is defined as $G|_C = \{\pi|_C \mid \pi \in G, C\pi = C\}$. The *extension* of a subgroup $S \leq G|_C$ is defined as $\mathrm{ext}_G(S) = \{\pi \in G \mid \pi|_C \in S\}$. For $C \subseteq \mathcal{D}$ and $S \leq G|_C$, define $[C, S]^{ec} = \{\{sg \mid s \in \mathrm{ext}_G(S)\} \mid g \in G\}$, i.e., the set of right cosets of $\mathrm{ext}_G(S)$ in G. Then $[C, S]^{ec}$ is a single-orbit nominal set.

Using the above, we can formulate the representation theory from [4] that we will use in the current paper. This gives a finite description for all single-orbit nominal sets X, namely a finite set C together with some of its symmetries.

Theorem 2. *Let X be a single-orbit nominal set for a data symmetry $(\mathcal{D}, G)$ that admits least supports and let $C \subseteq \mathcal{D}$ be the least support of some element $x \in X$. Then there exists a subgroup $S \leq G|_C$ such that $X \cong [C, S]^{ec}$.*

The proof [4] uses a bit of category theory: it establishes an equivalence of categories between single-orbit sets and the pairs (C, S). We will not use the language of category theory much in order to keep the paper self-contained.

3 Representation in the total order symmetry

This section develops a concrete representation of nominal sets over the total order symmetry, as well as their equivariant maps and products. It is based on the abstract representation theory from Section 2.1. From now on, by *nominal set* we always refer to a nominal set over the total order symmetry. Hence, our data domain is $\mathbb{Q}$ and we take G to be the group of monotone bijections.

3.1 Orbits and nominal sets

From the representation in Section 2.1, we find that any single-orbit set X can be represented as a tuple (C, S). Our first observation is that the finite group of 'local symmetries', S, in this representation is always trivial, i.e., $S = I$, where $I = \{1\}$ is the trivial group. This follows from the following lemma and $S \leq G|_C$.

Lemma 3. *For every finite subset $C \subset \mathbb{Q}$, we have $G|_C = I$.*

Immediately, we see that $(C, S) = (C, I)$, and hence that the orbit is fully represented by the set C. A further consequence of Lemma 3 is that each *element* of an orbit can be uniquely identified by its least support. This leads us to the following characterisation of $[C, I]^{ec}$.

Lemma 4. *Given a finite subset $C \subset \mathbb{Q}$, we have $[C, I]^{ec} \cong \mathcal{P}_{\#C}(\mathbb{Q})$.*

By Theorem 2 and the above lemmas, we can represent an orbit by a single integer n, the size of the least support of its elements. This naturally extends to (orbit-finite) nominal sets with multiple orbits by using a multiset of natural numbers, representing the size of the least support of each of the orbits. These multisets are formalised here as functions $f\colon \mathbb{N} \to \mathbb{N}$.

Definition 5. *Given a function $f\colon \mathbb{N} \to \mathbb{N}$, we define a nominal set $[f]^o$ by*

$$[f]^o = \bigcup_{\substack{n \in \mathbb{N} \\ 1 \leq i \leq f(n)}} \{i\} \times \mathcal{P}_n(\mathbb{Q}).$$

Proposition 6. *For every orbit-finite nominal set X, there is a function $f\colon \mathbb{N} \to \mathbb{N}$ such that $X \cong [f]^o$ and the set $\{n \mid f(n) \neq 0\}$ is finite. Furthermore, the mapping between X and f is one-to-one up to isomorphism of X when restricting to $f\colon \mathbb{N} \to \mathbb{N}$ for which the set $\{n \mid f(n) \neq 0\}$ is finite.*

The presentation in terms of a function $f\colon \mathbb{N} \to \mathbb{N}$ enforces that there are only finitely many orbits of any given dimension. The first part of the above proposition generalises to arbitrary nominal sets by replacing the codomain of f by the class of all sets and adapting Definition 5 accordingly. However, the resulting correspondence will no longer be one-to-one.

As a brief example, let us consider the set $\mathbb{Q} \times \mathbb{Q}$. The elements (a, b) split in three orbits, one for $a < b$, one for $a = b$ and one for $a > b$. These have dimension 2, 1 and 2 respectively, so the set $\mathbb{Q} \times \mathbb{Q}$ is represented by the multiset $\{1, 2, 2\}$.

3.2 Equivariant maps

We show how to represent equivariant maps, using two basic properties. Let $f : X \to Y$ be an equivariant map. The first property is that the direct image of an orbit (in X) is again an orbit (in Y), that is to say, f is defined 'orbit-wise'. Second, equivariant maps cannot introduce new elements in the support (but they can drop them). More precisely:

Lemma 7. *Let $f\colon X \to Y$ be an equivariant map, and $O \subseteq X$ a single orbit. The direct image $f(O) = \{f(x) \mid x \in O\}$ is a single-orbit nominal set.*

Lemma 8. *Let $f\colon X \to Y$ be an equivariant map between two nominal sets X and Y. Let $x \in X$ and let C be a support of x. Then C supports $f(x)$.*

Hence, equivariant maps are fully determined by associating two pieces of information for each orbit in the domain: the orbit on which it is mapped and a string denoting which elements of the least support of the input are preserved. These ingredients are formalised in the first part of the following definition. The second part describes how these ingredients define an equivariant function. Proposition 10 then states that every equivariant function can be described in this way.

Definition 9. *Let $H = \{(I_1, F_1, O_1), \ldots, (I_n, F_n, O_n)\}$ be a finite set of tuples where the I_i's are disjoint single-orbit nominal sets, the O_i's are single-orbit nominal sets with $\dim(O_i) \leq \dim(I_i)$, and the F_i's are bit strings of length $\dim(I_i)$ with exactly $\dim(O_i)$ ones.*

Given a set H as above, we define $f_H\colon \bigcup I_i \to \bigcup O_i$ as the unique equivariant function such that, given $x \in I_i$ with least support C, $f_H(x)$ is the unique element of O_i with support $\{C(j) \mid F_i(j) = 1\}$, where $F_i(j)$ is the j-th bit of F_i and $C(j)$ is the j-th smallest element of C.

Proposition 10. *For every equivariant map $f\colon X \to Y$ between orbit-finite nominal sets X and Y there is a set H as in Definition 9 such that $f = f_H$.*

Consider the example function $\min\colon \mathcal{P}_3(\mathbb{Q}) \to \mathbb{Q}$ which returns the smallest element of a 3-element set. Note that both $\mathcal{P}_3(\mathbb{Q})$ and $\mathbb{Q}$ are single orbits. Since for the orbit $\mathcal{P}_3(\mathbb{Q})$ we only keep the smallest element of the support, we can thus represent the function min with $\{(\mathcal{P}_3(\mathbb{Q}), 100, \mathbb{Q})\}$.

3.3 Products

The product $X \times Y$ of two nominal sets is again a nominal set and hence, it can be represented itself in terms of the dimension of each of its orbits as shown in Section 3.1. However, this approach has some disadvantages.

Example 11. We start by showing that the orbit structure of products can be non-trivial. Consider the product of $X = \mathbb{Q}$ and the set $Y = \{(a, b) \in \mathbb{Q}^2 \mid a < b\}$. This product consists of five orbits, more than one might naively expect from the fact that both sets are single-orbit:

$$\{(a,(b,c)) \mid a,b,c \in \mathbb{Q}, a<b<c\}, \quad \{(a,(a,b)) \mid a,b \in \mathbb{Q}, a<b\},$$
$$\{(b,(a,c)) \mid a,b,c \in \mathbb{Q}, a<b<c\}, \quad \{(b,(a,b)) \mid a,b \in \mathbb{Q}, a<b\},$$
$$\{(c,(a,b)) \mid a,b,c \in \mathbb{Q}, a<b<c\}.$$

We find that this product is represented by the multiset $\{2, 2, 3, 3, 3\}$. Unfortunately, this is not sufficient to accurately describe the product as it abstracts

away from the relation between its elements with those in X and Y. In particular, it is not possible to reconstruct the projection maps from such a representation.

The essence of our representation of products is that each orbit O in the product $X \times Y$ is described entirely by the dimension of O together with the two (equivariant) projections $\pi_1 : O \to X$ and $\pi_2 : O \to Y$. This combination of the orbit and the two projection maps can already be represented using Propositions 6 and 10. However, as we will see, a combined representation for this has several advantages. For discussing such a representation, let us first introduce what it means for tuples of a set and two functions to be isomorphic:

Definition 12. *Given nominal sets X, Y, Z_1 and Z_2, and equivariant functions $l_1 \colon Z_1 \to X$, $r_1 \colon Z_1 \to Y$, $l_2 \colon Z_2 \to X$ and $r_2 \colon Z_2 \to Y$, we define $(Z_1, l_1, r_1) \cong (Z_2, l_2, r_2)$ if there exists an isomorphism $h : Z_1 \to Z_2$ such that $l_1 = l_2 \circ h$ and $r_1 = r_2 \circ h$.*

Our goal is to have a representation that, for each orbit O, produces a tuple (A, f_1, f_2) isomorphic to the tuple (O, π_1, π_2). The next lemma gives a characterisation that can be used to simplify such a representation.

Lemma 13. *Let X and Y be nominal sets and $(x, y) \in X \times Y$. If C, C_x, and C_y are the least supports of (x, y), x, and y respectively, then $C = C_x \cup C_y$.*

With Proposition 10 we represent the maps π_1 and π_2 by tuples (O, F_1, O_1) and (O, F_2, O_2) respectively. Using Lemma 13 and the definitions of F_1 and F_2, we see that at least one of $F_1(i)$ and $F_2(i)$ equals 1 for each i.

We can thus combine the strings F_1 and F_2 into a single string $P \in \{L, R, B\}^*$ as follows. We set $P(i) = L$ when only $F_1(i)$ is 1, $P(i) = R$ when only $F_2(i)$ is 1, and $P(i) = B$ when both are 1. The string P fully describes the strings F_1 and F_2. This process for constructing the string P gives it two useful properties. The number of Ls and Bs in the string gives the size dimension of O_1. Similarly, the number of Rs and Bs in the string gives the dimension of O_2. We will call strings with that property *valid*. In conclusion, to describe a single orbit of the product $X \times Y$, a valid string P together with the images of π_1 and π_2 is sufficient.

Definition 14. *Let $P \in \{L, R, B\}^*$, and $O_1 \subseteq X$, $O_2 \subseteq Y$ be single-orbit sets. Given a tuple (P, O_1, O_2), where the string P is valid, define*

$$[(P, O_1, O_2)]^t = (\mathcal{P}_{|P|}(\mathbb{Q}), f_{H_1}, f_{H_2}),$$

where $H_i = \{(\mathcal{P}_{|P|}(\mathbb{Q}), F_i, O_i)\}$ and the string F_1 is defined as the string P with Ls and Bs replaced by 1s and Rs by 0s. The string F_2 is similarly defined with the roles of L and R swapped.

Proposition 15. *There exists a one-to-one correspondence between the orbits $O \subseteq X \times Y$, and tuples (P, O_1, O_2) satisfying $O_1 \subseteq X$, $O_2 \subseteq Y$, and where P is a valid string, such that $[(P, O_1, O_2)]^t \cong (O, \pi_1|_O, \pi_2|_O)$.*

From the above proposition it follows that we can generate the product $X \times Y$ simply by enumerating all valid strings P for all pairs of orbits (O_1, O_2) of X and Y. Given this, we can calculate the multiset representation of a product from the multiset representations of both factors.

Theorem 16. *For $X \cong [f]^o$ and $Y \cong [g]^o$ we have $X \times Y \cong [h]^o$, where*

$$h(n) = \sum_{\substack{0 \leq i,j \leq n \\ i+j \geq n}} f(i)g(j)\binom{n}{j}\binom{j}{n-i}.$$

Example 17. To illustrate some aspects of the above representation, let us use it to calculate the product of Example 11. First, we observe that both $\mathbb{Q}$ and $S = \{(a, b) \in \mathbb{Q}^2 \mid a < b\}$ consist of a single orbit. Hence any orbit of the product corresponds to a triple $(P, \mathbb{Q}, S)$, where the string P satisfies $|P|_L + |P|_B = \dim(\mathbb{Q}) = 1$ and $|P|_R + |P|_B = \dim(S) = 2$. We can now find the orbits of the product $\mathbb{Q} \times S$ by enumerating all strings satisfying these equations. This yields:

- LRR, corresponding to the orbit $\{(a, (b, c)) \mid a, b, c \in \mathbb{Q}, a < b < c\}$,
- RLR, corresponding to the orbit $\{(b, (a, c)) \mid a, b, c \in \mathbb{Q}, a < b < c\}$,
- RRL, corresponding to the orbit $\{(c, (a, b)) \mid a, b, c \in \mathbb{Q}, a < b < c\}$,
- RB, corresponding to the orbit $\{(b, (a, b)) \mid a, b \in \mathbb{Q}, a < b\}$, and
- BR, corresponding to the orbit $\{(a, (a, b)) \mid a, b \in \mathbb{Q}, a < b\}$.

Each product string fully describes the corresponding orbit. To illustrate, consider the string BR. The corresponding bit strings for the projection functions are $F_1 = 10$ and $F_2 = 11$. From the lengths of the string we conclude that the dimension of the orbit is 2. The string F_1 further tells us that the left element of the tuple consists only of the smallest element of the support. The string F_2 indicates that the right element of the tuple is constructed from both elements of the support. Combining this, we find that the orbit is $\{(a, (a, b)) \mid a, b \in \mathbb{Q}, a < b\}$.

3.4 Summary

We summarise our concrete representation in the following table. Propositions 6, 10 and 15 correspond to the three rows in the table.

Object	*Representation*
Single orbit O	Natural number $n = \dim(O)$
Nominal set $X = \bigcup_i O_i$	Multiset of these numbers
Map from single orbit $f \colon O \to Y$	The orbit $f(O)$ and a bit string F
Equivariant map $f \colon X \to Y$	Set of tuples $(O, F, f(O))$, one for each orbit
Orbit in a product $O \subseteq X \times Y$	The corresponding orbits of X and Y, and a string P relating their supports
Product $X \times Y$	Set of tuples (P, O_X, O_Y), one for each orbit

Notice that in the case of maps and products, the orbits are inductively represented using the concrete representation. As a base case we can represent single orbits by their dimension.

4 Implementation and Complexity of ONS

The ideas outlined above have been implemented in the C++ library Ons.[2] The library can represent orbit-finite nominal sets and their products, (disjoint) unions, and maps. A full description of the possibilities is given in the documentation included with Ons.

As an example, the following program computes the product from Example 11. Initially, the program creates the nominal set A, containing the entirety of $\mathbb{Q}$. Then it creates a nominal set B, such that it consists of the orbit containing the element $(1, 2) \in \mathbb{Q} \times \mathbb{Q}$. For this, the library determines to which orbit of the product $\mathbb{Q} \times \mathbb{Q}$ the element $(1, 2)$ belongs, and then stores a description of the orbit as described in Section 3. Note that this means that it internally never needs to store the element used to create the orbit. The function `nomset_product` then uses the enumeration of product strings mentioned in Section 3.3 to calculate the product of A and B. Finally, it prints a representative element for each of the orbits in the product. These elements are constructed based on the description of the orbits stored, filled in to make their support equal to sets of the form $\{1, 2, \ldots, n\}$.

```
nomset<rational> A = nomset_rationals();
nomset<pair<rational, rational>> B({rational(1),rational(2)});

auto AtimesB = nomset_product(A, B);   // compute the product
for (auto orbit : AtimesB)
        cout << orbit.getElement() << "␣";
```

Running this gives the following output ('`/1`' signifies the denominator):

```
(1/1,(2/1,3/1)) (1/1,(1/1,2/1)) (2/1,(1/1,3/1))
(2/1,(1/1,2/1)) (3/1,(1/1,2/1))
```

Internally, `orbit` is implemented following the theory presented in Section 3, storing the dimension of the orbit it represents. It also contains sufficient information to reconstruct elements given their least support, such as the product string for orbits resulting from a product. The class `nomset` then uses a standard set data structure to store the collection of orbits contained in the nominal set it represents.

In a similar way, `eqimap` stores equivariant maps by associating each orbit in the domain with the image orbit and the string representing which of the least support to keep. This is stored using a map data structure. For both nominal sets and equivariant maps, the underlying data structure is currently implemented using trees.

[2] Ons can be found at `https://github.com/davidv1992/ONS`

4.1 Complexity of operations

Using the concrete representation of nominal sets, we can determine the complexity of common operations. To simplify such an analysis, we will make the following assumptions:

- The comparison of two orbits takes $O(1)$.
- Constructing an orbit from an element takes $O(1)$.
- Checking whether an element is in an orbit takes $O(1)$.

These assumptions are justified as each of these operations takes time proportional to the size of the representation of an individual orbit, which in practice is small and approximately constant. For instance, the orbit $\mathcal{P}_n(\mathbb{Q})$ is represented by just the integer n and its type.

Theorem 18. *If nominal sets are implemented with a tree-based set structure (as in* Ons*), the complexity of the following set operations is as follows. Recall that* $\mathrm{N}(X)$ *denotes the number of orbits of* X*. We use* p *and* f *to denote functions implemented in whatever way the user wants, which we assume to take* $O(1)$ *time. The software assumes these are equivariant, but this is not verified.*

Operation	*Complexity*
Test $x \in X$	$O(\log \mathrm{N}(X))$
Test $X \subseteq Y$	$O(\min(\mathrm{N}(X) + \mathrm{N}(Y), \mathrm{N}(X) \log \mathrm{N}(Y)))$
Calculate $X \cup Y$	$O(\mathrm{N}(X) + \mathrm{N}(Y))$
Calculate $X \cap Y$	$O(\mathrm{N}(X) + \mathrm{N}(Y))$
Calculate $\{x \in X \mid p(x)\}$	$O(\mathrm{N}(X))$
Calculate $\{f(x) \mid x \in X\}$	$O(\mathrm{N}(X) \log \mathrm{N}(X))$
Calculate $X \times Y$	$O(\mathrm{N}(X \times Y)) \subseteq O(3^{\dim(X)+\dim(Y)} \mathrm{N}(X) \mathrm{N}(Y))$

Proof. Since most parts are proven similarly, we only include proofs for the first and last item.

Membership. To decide $x \in X$, we first construct the orbit containing x, which is done in constant time. Then we use a logarithmic lookup to decide whether this orbit is in our set data structure. Hence, membership checking is $O(\log(\mathrm{N}(X)))$.

Products. Calculating the product of two nominal sets is the most complicated construction. For each pair of orbits in the original sets X and Y, all product orbits need to be generated. Each product orbit itself is constructed in constant time. By generating these orbits in-order, the resulting set takes $O(\mathrm{N}(X \times Y))$ time to construct.

We can also give an explicit upper bound for the number of orbits in terms of the input. Recall that orbits in a product are represented by strings of length at most $\dim(X) + \dim(Y)$. (If the string is shorter, we pad it with one of the symbols.) Since there are three symbols (L, R and B), the product of X and Y will have at most $3^{\dim(X)+\dim(Y)} \mathrm{N}(X) \mathrm{N}(Y)$ orbits. It follows that taking products has time complexity of $O(3^{\dim(X)+\dim(Y)} \mathrm{N}(X) \mathrm{N}(Y))$. □

5 Results and evaluation in automata theory

In this section we consider applications of nominal sets to automata theory. As mentioned in the introduction, nominal sets are used to formalise languages over infinite alphabets. These languages naturally arise as the semantics of register automata. The definition of register automata is not as simple as that of ordinary finite automata. Consequently, transferring results from automata theory to this setting often requires non-trivial proofs. Nominal automata, instead, are defined as ordinary automata by replacing finite sets with orbit-finite nominal sets. The theory of nominal automata is developed in [4] and it is shown that many, but not all, algorithms from automata theory transfer to nominal automata.

As an example we consider the following language on rational numbers:

$$\mathcal{L}_{\text{int}} = \{a_1 b_1 \cdots a_n b_n \mid a_i, b_i \in \mathbb{Q}, a_i < a_{i+1} < b_{i+1} < b_i \text{ for all } i\}.$$

We call this language the *interval language* as a word $w \in \mathbb{Q}^*$ is in the language when it denotes a sequence of nested intervals. This language contains arbitrarily long words. For this language it is crucial to work with an infinite alphabet as for each finite set $C \subset \mathbb{Q}$, the restriction $\mathcal{L}_{\text{int}} \cap C^*$ is just a finite language. Note that the language is equivariant: $w \in \mathcal{L}_{\text{int}} \iff wg \in \mathcal{L}_{\text{int}}$ for any monotone bijection g, because nested intervals are preserved by monotone maps.[3] Indeed, $\mathcal{L}_{\text{int}}$ is a nominal set, although it is not orbit-finite.

Informally, the language $\mathcal{L}_{\text{int}}$ can be accepted by the automaton depicted in Figure 1. Here we allow the automaton to store rational numbers and compare them to new symbols. For example, the transition from q_2 to q_3 is taken if any value c between a and b is read and then the currently stored value a is replaced by c. For any other value read at state q_2 the automaton transitions to the sink state q_4. Such a transition structure is made precise by the notion of nominal automata.

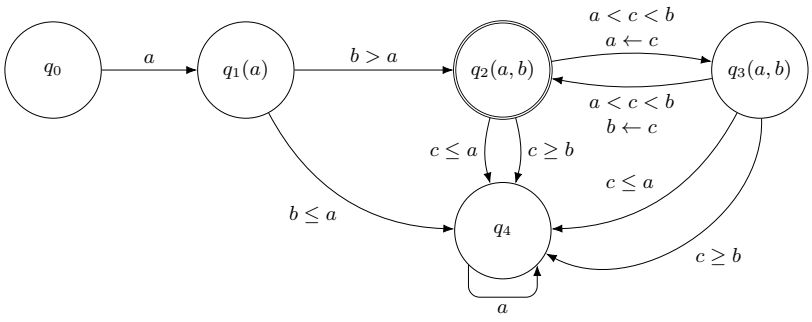

Fig. 1. Example automaton that accepts the language $\mathcal{L}_{\text{int}}$.

[3] The G-action on words is defined point-wise: $(w_1 \dots w_n)g = (w_1 g) \dots (w_n g)$.

Definition 19. *A* nominal language *is an equivariant subset $L \subseteq A^*$ where A is an orbit-finite nominal set.*

Definition 20. *A* nominal deterministic finite automaton *is a tuple (S, A, F, δ), where S is an orbit-finite nominal set of states, A is an orbit-finite nominal set of symbols, $F \subseteq S$ is an equivariant subset of final states, and $\delta\colon S \times A \to S$ is the equivariant transition function.*

Given a state $s \in S$, we define the usual acceptance condition: a word $w \in A^*$ is accepted *if w denotes a path from s to a final state.*

The automaton in Figure 1 can be formalised as a nominal deterministic finite automaton as follows. Let $S = \{q_0, q_4\} \cup \{q_1(a) \mid a \in \mathbb{Q}\} \cup \{q_2(a,b) \mid a < b \in \mathbb{Q}\} \cup \{q_3(a,b) \mid a < b \in \mathbb{Q}\}$ be the set of states, where the group action is defined as one would expect. The transition we described earlier can now be formally defined as $\delta(q_2(a,b), c) = q_3(c,b)$ for all $a < c < b \in \mathbb{Q}$. By defining δ on all states accordingly and defining the final states as $F = \{q_2(a,b) \mid a < b \in \mathbb{Q}\}$, we obtain a nominal deterministic automaton $(S, \mathbb{Q}, F, \delta)$. The state q_0 accepts the language $\mathcal{L}_{\text{int}}$.

Testing. We implement two algorithms on nominal automata, minimisation and learning, to benchmark Ons. The performance of Ons is compared to two existing libraries for computing with nominal sets, Nλ and Lois. The following automata will be used.

Random automata. As a primary test suite, we generate random automata as follows. The input alphabet is always $\mathbb{Q}$ and the number of orbits and dimension k of the state space S are fixed. For each orbit in the set of states, its dimension is chosen uniformly at random between 0 and k, inclusive. Each orbit has a probability $\frac{1}{2}$ of consisting of accepting states.

To generate the transition function δ, we enumerate the orbits of $S \times \mathbb{Q}$ and choose a target state uniformly from the orbits S with small enough dimension. The bit string indicating which part of the support is preserved is then sampled uniformly from all valid strings. We will denote these automata as $\text{rand}_{N(S),k}$. The choices made here are arbitrary and only provide basic automata. We note that the automata are generated orbit-wise and this may favour our tool.

Structured automata. Besides random automata we wish to test the algorithms on more structured automata. We define the following automata.

- FIFO(n) Automata accepting valid traces of a finite FIFO data structure of size n. The alphabet is defined by two orbits: $\{\text{Put}(a) \mid a \in \mathbb{Q}\}$ and $\{\text{Get}(a) \mid a \in \mathbb{Q}\}$.
- $ww(n)$ Automata accepting the language of words of the form ww, where $w \in \mathbb{Q}^n$.
- $\mathcal{L}_{\max}$ The language $\mathcal{L}_{\max} = \{wa \in \mathbb{Q}^* \mid a = \max(w_1, \ldots, w_n)\}$ where the last symbol is the maximum of previous symbols.
- $\mathcal{L}_{\text{int}}$ The language accepting a series of nested intervals, as defined above.

In Table 1 we report the number of orbits for each automaton. The first two classes of automata were previously used as test cases in [21]. These two classes are also equivariant w.r.t. the equality symmetry. The extra bit of structure allows the automata to be encoded more efficiently, as we do not need to encode a transition for each orbit in $S \times A$. Instead, a more symbolic encoding is possible. Both Lois and Nλ allow to use this more symbolic representation. Our tool, Ons, only works with nominal sets and the input data needs to be provided orbit-wise. Where applicable, the automata listed above were generated using the same code as used in [21], ported to the other libraries as needed.

5.1 Minimising nominal automata

For languages recognised by nominal DFAs, a Myhill-Nerode theorem holds which relates states to right congruence classes [4]. This guarantees the existence of unique minimal automata. We say an automaton is *minimal* if its set of states has the least number of orbits and each orbit has the smallest dimension possible.[4] We generalise Moore's minimisation algorithm to nominal DFAs (Algorithm 1) and analyse its time complexity using the bounds from Section 4.

Algorithm 1 Moore's minimisation algorithm for nominal DFAs

Require: Nominal automaton (S, A, F, δ).
1: $i \leftarrow 0$, $\equiv_{-1} \leftarrow S \times S$, $\equiv_0 \leftarrow F \times F \cup (S \backslash F) \times (S \backslash F)$
2: **while** $\equiv_i \neq \equiv_{i-1}$ **do**
3: $\quad \equiv_{i+1} = \{(q_1, q_2) \mid (q_1, q_2) \in \equiv_i \wedge \forall a \in A, (\delta(q_1, a), \delta(q_2, a)) \in \equiv_i\}$
4: $\quad i \leftarrow i + 1$
5: **end while**
6: $E \leftarrow S/_{\equiv_i}$
7: $F_E \leftarrow \{e \in E \mid \forall s \in e, s \in F\}$
8: Let δ_E be the map such that, if $s \in e$ and $\delta(s, a) \in e'$, then $\delta_E(e, a) = e'$.
9: **return** (E, A, F_E, δ_E).

Theorem 21. *The runtime complexity of Moore's algorithm on nominal deterministic automata is* $O(3^{5k} k \,\mathrm{N}(S)^3 \,\mathrm{N}(A))$*, where* $k = \dim(S \cup A)$.

Proof. This is shown by counting operations, using the complexity results of set operations stated in Theorem 18. We first focus on the while loop on lines 2 through 5. The runtime of an iteration of the loop is determined by line 3, as this is the most expensive step. Since the dimensions of S and A are at most k, computing $S \times S \times A$ takes $O(\mathrm{N}(S)^2 \,\mathrm{N}(A) 3^{5k})$. Filtering $S \times S$ using that then takes $O(\mathrm{N}(S)^2 3^{2k})$. The time to compute $S \times S \times A$ dominates, hence each iteration of the loop takes $O(\mathrm{N}(S)^2 \,\mathrm{N}(A) 3^{5k})$.

[4] Abstractly, an automaton is minimal if it has no proper quotients. Minimal deterministic automata are unique up to isomorphism.

Next, we need to count the number of iterations of the loop. Each iteration of the loop gives rise to a new partition, which is a refinement of the previous partition. Furthermore, every partition generated is equivariant. Note that this implies that each refinement of the partition does at least one of two things: distinguish between two orbits of S previously in the same element(s) of the partition, or distinguish between two members of the same orbit previously in the same element of the partition. The first can happen only $\mathrm{N}(S) - 1$ times, as after that there are no more orbits lumped together. The second can only happen $\dim(S)$ times per orbit, because each such a distinction between elements is based on splitting on the value of one of the elements of the support. Hence, after $\dim(S)$ times on a single orbit, all elements of the support are used up. Combining this, the longest chain of partitions of S has length at most $O(k\,\mathrm{N}(S))$.

Since each partition generated in the loop is unique, the loop cannot run for more iterations than the length of the longest chain of partitions on S. It follows that there are at most $O(k\,\mathrm{N}(S))$ iterations of the loop, giving the loop a complexity of $O(k\,\mathrm{N}(S)^3\,\mathrm{N}(A)3^{5k})$

The remaining operations outside the loop have a lower complexity than that of the loop, hence the complexity of Moore's minimisation algorithm for a nominal automaton is $O(k\,\mathrm{N}(S)^3\,\mathrm{N}(A)3^{5k})$.

The above theorem shows in particular that minimisation of nominal automata is fixed-parameter tractable (FPT) with the dimension as fixed parameter. The complexity of Algorithm 1 for nominal automata is very similar to the $O((\#S)^3\#A)$ bound given by a naive implementation of Moore's algorithm for ordinary DFAs. This suggest that it is possible to further optimise an implementation with similar techniques used for ordinary automata.

Implementations. We implemented the minimisation algorithm in Ons. For Nλ and Lois we used their implementations of Moore's minimisation algorithm [17,18,19]. For each of the libraries, we wrote routines to read in an automaton from a file and, for the structured test cases, to generate the requested automaton. For Ons, all automata were read from file. The output of these programs was manually checked to see if the minimisation was performed correctly.

Results. The results (shown in Table 1) for random automata show a clear advantage for Ons, which is capable of running all supplied testcases in less than one second. This in contrast to both Lois and Nλ, which take more than 2 hours on the largest random automata.

The results for structured automata show a clear effect of the extra structure. Both Nλ and Lois remain capable of minimising the automata in reasonable amounts of time for larger sizes. In contrast, Ons benefits little from the extra structure. Despite this, it remains viable: even for the larger cases it falls behind significantly only for the largest FIFO automaton and the two largest ww automata.

Type	N(S)	N($S^{\min}$)	Ons		Nλ	Lois
				Gen.		
$\text{rand}_{5,1}$ (x10)	5	n/a	0.02s	n/a	0.82s	3.14s
$\text{rand}_{10,1}$ (x10)	10	n/a	0.03s	n/a	17.03s	1m 32s
$\text{rand}_{10,2}$ (x10)	10	n/a	0.09s	n/a	35m 14s	> 60m
$\text{rand}_{15,1}$ (x10)	15	n/a	0.04s	n/a	1m 27s	10m 20s
$\text{rand}_{15,2}$ (x10)	15	n/a	0.11s	n/a	55m 46s	> 60m
$\text{rand}_{15,3}$ (x10)	15	n/a	0.46s	n/a	> 60m	> 60m
FIFO(2)	13	6	0.01s	0.01s	1.37s	0.24s
FIFO(3)	65	19	0.38s	0.09s	11.59s	2.4s
FIFO(4)	440	94	39.11s	1.60s	1m 16s	14.95s
FIFO(5)	3686	635	> 60m	39.78s	6m 42s	1m 11s
$ww(2)$	8	8	0.00s	0.00s	0.14s	0.03s
$ww(3)$	24	24	0.19s	0.02s	0.88s	0.16s
$ww(4)$	112	112	26.44s	0.25s	3.41s	0.61s
$ww(5)$	728	728	> 60m	6.37s	10.54s	1.80s
$\mathcal{L}_{\max}$	5	3	0.00s	0.00s	2.06s	0.03s
$\mathcal{L}_{\text{int}}$	5	5	0.00s	0.00s	1.55s	0.03s

Table 1. Running times for Algorithm 1 implemented in the three libraries. $N(S)$ is the size of the input and $N(S^{\min})$ the size of the minimal automaton. For Ons, the time used to generate the automaton is reported separately (in grey).

5.2 Learning nominal automata

Another application that we implemented in Ons is *automata learning*. The aim of automata learning is to infer an unknown regular language $\mathcal{L}$. We use the framework of active learning as set up by Dana Angluin [2] where a learning algorithm can query an oracle to gather information about $\mathcal{L}$. Formally, the oracle can answer two types of queries:

1. *membership queries*, where a query consists of a word $w \in A^*$ and the oracle replies whether $w \in \mathcal{L}$, and
2. *equivalence queries*, where a query consists of an automaton $\mathcal{H}$ and the oracle replies positively if $\mathcal{L}(\mathcal{H}) = \mathcal{L}$ or provides a counterexample if $\mathcal{L}(\mathcal{H}) \neq \mathcal{L}$.

With these queries, the $\text{L}^\star$ algorithm can learn regular languages efficiently [2]. In particular, it learns the unique minimal automaton for $\mathcal{L}$ using only finitely many queries. The $\text{L}^\star$ algorithm has been generalised to $\nu\text{L}^\star$ in order to learn *nominal* regular languages [21]. In particular, it learns a nominal DFA (over an infinite alphabet) using only finitely many queries. We implement $\nu\text{L}^\star$ in the presented library and compare it to its previous implementation in Nλ. The algorithm is not polynomial, unlike the minimisation algorithm described above. However, the authors conjecture that there is a polynomial algorithm.[5] For the correctness, termination, and comparison with other learning algorithms see [21].

[5] See `joshuamoerman.nl/papers/2017/17popl-learning-nominal-automata.html` for a sketch of the polynomial algorithm.

Implementations. Both implementations in Nλ and Ons are direct implementations of the pseudocode for $\nu L^\star$ with no further optimisations. The authors of Lois implemented $\nu L^\star$ in their library as well.[6] They reported similar performance as the implementation in Nλ (private communication). Hence we focus our comparison on Nλ and Ons. We use the variant of $\nu L^\star$ where counterexamples are added as columns instead of prefixes.

The implementation in Nλ has the benefit that it can work with different symmetries. Indeed, the structured examples, FIFO and ww, are equivariant w.r.t. the equality symmetry as well as the total order symmetry. For that reason, we run the Nλ implementation using both the equality symmetry and the total order symmetry on those languages. For the languages $\mathcal{L}_{\max}$, $\mathcal{L}_{\text{int}}$ and the random automata, we can only use the total order symmetry.

To run the $\nu L^\star$ algorithm, we implement an external oracle for the membership queries. This is akin to the application of learning black box systems [29]. For equivalence queries, we constructed counterexamples by hand. All implementations receive the same counterexamples. We measure CPU time instead of real time, so that we do not account for the external oracle.

Results. The results (Table 2) for random automata show an advantage for Ons. Additionally, we report the number of membership queries, which can vary for each implementation as some steps in the algorithm depend on the internal ordering of set data structures.

In contrast to the case of minimisation, the results suggest that Nλ cannot exploit the logical structure of FIFO(n), $\mathcal{L}_{\max}$ and $\mathcal{L}_{\text{int}}$ as it is not provided a priori. For $ww(2)$ we inspected the output on Nλ and saw that it learned some logical structure (e.g., it outputs $\{(a,b) \mid a \neq b\}$ as a single object instead of two orbits $\{(a,b) \mid a < b\}$ and $\{(a,b) \mid b < a\}$). This may explain why Nλ is still competitive. For languages which are equivariant for the equality symmetry, the Nλ implementation using the equality symmetry can learn with much fewer queries. This is expected as the automata themselves have fewer orbits. It is interesting to see that these languages can be learned more efficiently by choosing the right symmetry.

6 Related work

As stated in the introduction, Nλ [17] and Lois [18] use first-order formulas to represent nominal sets and use SMT solvers to manipulate them. This makes both libraries very flexible and they indeed implement the equality symmetry as well as the total order symmetry. As their representation is not unique, the efficiency depends on how the logical formulas are constructed. As such, they do not provide complexity results. In contrast, our direct representation allows for complexity results (Section 4) and leads to different performance characteristics (Section 5).

[6] Can be found on `github.com/eryxcc/lois/blob/master/tests/learning.cpp`

			Ons		Nλ^{ord}		Nλ^{eq}	
Model	N(S)	dim(S)	time	MQs	time	MQs	time	MQs
$\text{rand}_{5,1}$	4	1	2m 7s	2321	39m 51s	1243		
$\text{rand}_{5,1}$	5	1	0.12s	404	40m 34s	435		
$\text{rand}_{5,1}$	3	0	0.86s	499	30m 19s	422		
$\text{rand}_{5,1}$	5	1	$>$ 60m	n/a	$>$ 60m	n/a		
$\text{rand}_{5,1}$	4	1	0.08s	387	34m 57s	387		
FIFO(1)	3	1	0.04s	119	3.17s	119	1.76s	51
FIFO(2)	6	2	1.73s	2655	6m 32s	3818	40.00s	434
FIFO(3)	19	3	46m 34s	298400	$>$ 60m	n/a	34m 7s	8151
$ww(1)$	4	1	0.42s	134	2.49s	77	1.47s	30
$ww(2)$	8	2	4m 26s	3671	3m 48s	2140	30.58s	237
$ww(3)$	24	3	$>$ 60m	n/a	$>$ 60m	n/a	$>$ 60m	n/a
$\mathcal{L}_{\max}$	3	1	0.01s	54	3.58s	54		
$\mathcal{L}_{\text{int}}$	5	2	0.59s	478	1m 23s	478		

Table 2. Running times and number of membership queries for the $\nu L^\star$ algorithm. For Nλ we used two version: Nλ^{ord} uses the total order symmetry Nλ^{eq} uses the equality symmetry.

A second big difference is that both Nλ and Lois implement a "programming paradigm" instead of just a library. This means that they overload natural programming constructs in their host languages (Haskell and C++ respectively). For programmers this means they can think of infinite sets without having to know about nominal sets.

It is worth mentioning that an older (unreleased) version of Nλ implemented nominal sets with orbits instead of SMT solvers [3]. However, instead of characterising orbits (e.g., by its dimension), they represent orbits by a representative element. The authors of Nλ have reported that the current version is faster [17].

The theoretical foundation of our work is the main representation theorem in [4]. We improve on that by instantiating it to the total order symmetry and distil a concrete representation of nominal sets. As far as we know, we provide the first implementation of the representation theory in [4].

Another tool using nominal sets is Mihda [12]. Here, only the equality symmetry is implemented. This tool implements a translation from π-calculus to history-dependent automata (HD-automata) with the aim of minimisation and checking bisimilarity. The implementation in OCaml is based on *named sets*, which are finite representations for nominal sets. The theory of named sets is well-studied and has been used to model various behavioural models with local names. For those results, the categorical equivalences between named sets, nominal sets and a certain (pre)sheaf category have been exploited [8,9]. The total order symmetry is not mentioned in their work. We do, however, believe that similar equivalences between categories can be stated. Interestingly, the product of named sets is similar to our representation of products of nominal sets: pairs of elements together with data which denotes the relation between data values.

Fresh OCaml [27] and Nominal Isabelle [28] are both specialised in name-binding and α-conversion used in proof systems. They only use the equality symmetry and do not provide a library for manipulating nominal sets. Hence they are not suited for our applications.

On the theoretical side, there are many complexity results for register automata [15,23]. In particular, we note that problems such as emptiness and equivalence are NP-hard depending on the type of register automaton. This does not easily compare to our complexity results for minimisation. One difference is that we use the total order symmetry, where the local symmetries are always trivial (Lemma 3). As a consequence, all the complexity required to deal with groups vanishes. Rather, the complexity is transferred to the input of our algorithms, because automata over the equality symmetry require more orbits when expressed over the total order symmetry. Another difference is that register automata allow for duplicate values in the registers. In nominal automata, such configurations will be encoded in different orbits. An interesting open problem is whether equivalence of unique-valued register automata is in PTIME [23].

Orthogonal to nominal automata, there is the notion of symbolic automata [10,20]. These automata are also defined over infinite alphabets but they use predicates on transitions, instead of relying on symmetries. Symbolic automata are finite state (as opposed to infinite state nominal automata) and do not allow for storing values. However, they do allow for general predicates over an infinite alphabet, including comparison to constants.

7 Conclusion and Future Work

We presented a concrete finite representation for nominal sets over the total order symmetry. This allowed us to implement a library, ONS, and provide complexity bounds for common operations. The experimental comparison of ONS against existing solutions for automata minimisation and learning show that our implementation is much faster in many instances. As such, we believe ONS is a promising implementation of nominal techniques.

A natural direction for future work is to consider other symmetries, such as the equality symmetry. Here, we may take inspiration from existing tools such as Mihda (see Section 6). Another interesting question is whether it is possible to translate a nominal automaton over the total order symmetry which accepts an equality language to an automaton over the equality symmetry. This would allow one to efficiently move between symmetries. Finally, our techniques can potentially be applied to timed automata by exploiting the intriguing connection between the nominal automata that we consider and timed automata [5].

Acknowledgement We would like to thank Szymon Toruńczyk and Eryk Kopczyński for their prompt help when using the LOIS library. For general comments and suggestions we would like to thank Ugo Montanari and Niels van der Weide. At last, we want to thank the anonymous reviewers for their comments.

References

1. Aarts, F., Fiterau-Brostean, P., Kuppens, H., Vaandrager, F.W.: Learning register automata with fresh value generation. In: Theoretical Aspects of Computing - ICTAC 2015. pp. 165–183 (2015). https://doi.org/10.1007/978-3-319-25150-9_11
2. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6
3. Bojańczyk, M., Braud, L., Klin, B., Lasota, S.: Towards nominal computation. In: Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012. pp. 401–412 (2012). https://doi.org/10.1145/2103656.2103704
4. Bojańczyk, M., Klin, B., Lasota, S.: Automata theory in nominal sets. Logical Methods in Computer Science **10**(3) (2014). https://doi.org/10.2168/LMCS-10(3:4)2014
5. Bojańczyk, M., Lasota, S.: A machine-independent characterization of timed languages. In: Automata, Languages, and Programming, ICALP 2012, Part II. pp. 92–103 (2012). https://doi.org/10.1007/978-3-642-31585-5_12
6. Bollig, B., Habermehl, P., Leucker, M., Monmege, B.: A fresh approach to learning register automata. In: Developments in Language Theory, DLT 2013. pp. 118–130 (2013). https://doi.org/10.1007/978-3-642-38771-5_12
7. Cassel, S., Howar, F., Jonsson, B., Steffen, B.: Active learning for extended finite state machines. Formal Asp. Comput. **28**(2), 233–263 (2016). https://doi.org/10.1007/s00165-016-0355-5
8. Ciancia, V., Kurz, A., Montanari, U.: Families of symmetries as efficient models of resource binding. Electr. Notes Theor. Comput. Sci. **264**(2), 63–81 (2010). https://doi.org/10.1016/j.entcs.2010.07.014
9. Ciancia, V., Montanari, U.: Symmetries, local names and dynamic (de)-allocation of names. Inf. Comput. **208**(12), 1349–1367 (2010). https://doi.org/10.1016/j.ic.2009.10.007
10. D'Antoni, L., Veanes, M.: The power of symbolic automata and transducers. In: Computer Aided Verification, CAV 2017, Part I. pp. 47–67 (2017). https://doi.org/10.1007/978-3-319-63387-9_3
11. Drews, S., D'Antoni, L.: Learning symbolic automata. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2017, Part I. pp. 173–189 (2017). https://doi.org/10.1007/978-3-662-54577-5_10
12. Ferrari, G.L., Montanari, U., Tuosto, E.: Coalgebraic minimization of hd-automata for the pi-calculus using polymorphic types. Theor. Comput. Sci. **331**(2-3), 325–365 (2005). https://doi.org/10.1016/j.tcs.2004.09.021
13. Fiterau-Brostean, P., Janssen, R., Vaandrager, F.W.: Combining model learning and model checking to analyze TCP implementations. In: Computer Aided Verification, CAV 2016, Part II. pp. 454–471 (2016). https://doi.org/10.1007/978-3-319-41540-6_25
14. Gabbay, M., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Asp. Comput. **13**(3-5), 341–363 (2002). https://doi.org/10.1007/s001650200016
15. Grigore, R., Tzevelekos, N.: History-register automata. Logical Methods in Computer Science **12**(1) (2016). https://doi.org/10.2168/LMCS-12(1:7)2016
16. Kaminski, M., Francez, N.: Finite-memory automata. Theor. Comput. Sci. **134**(2), 329–363 (1994). https://doi.org/10.1016/0304-3975(94)90242-9

17. Klin, B., Szynwelski, M.: SMT solving for functional programming over infinite structures. In: Proceedings 6th Workshop on Mathematically Structured Functional Programming, MSFP 2016. pp. 57–75 (2016). https://doi.org/10.4204/EPTCS.207.3
18. Kopczynski, E., Toruńczyk, S.: LOIS: an application of SMT solvers. In: Proceedings of the 14th International Workshop on Satisfiability Modulo Theories, SMT 2016. pp. 51–60 (2016), http://ceur-ws.org/Vol-1617/paper5.pdf
19. Kopczynski, E., Toruńczyk, S.: LOIS: syntax and semantics. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017. pp. 586–598 (2017), http://dl.acm.org/citation.cfm?id=3009876
20. Maler, O., Mens, I.: A generic algorithm for learning symbolic automata from membership queries. In: Models, Algorithms, Logics and Tools. pp. 146–169 (2017). https://doi.org/10.1007/978-3-319-63121-9_8
21. Moerman, J., Sammartino, M., Silva, A., Klin, B., Szynwelski, M.: Learning nominal automata. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017. pp. 613–625 (2017), http://dl.acm.org/citation.cfm?id=3009879
22. Montanari, U., Pistore, M.: An introduction to history dependent automata. Electr. Notes Theor. Comput. Sci. **10**, 170–188 (1998). https://doi.org/10.1016/S1571-0661(05)80696-6
23. Murawski, A.S., Ramsay, S.J., Tzevelekos, N.: Bisimilarity in fresh-register automata. In: 30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015. pp. 156–167 (2015). https://doi.org/10.1109/LICS.2015.24
24. Pitts, A.M.: Nominal Sets: Names and Symmetry in Computer Science. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press (2013)
25. Pitts, A.M.: Nominal techniques. SIGLOG News **3**(1), 57–72 (2016). https://doi.org/10.1145/2893582.2893594
26. Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Computer Science Logic, CSL 2006. pp. 41–57 (2006). https://doi.org/10.1007/11874683_3
27. Shinwell, M.R., Pitts, A.M.: Fresh objective Caml user manual. Tech. rep., University of Cambridge, Computer Laboratory (2005)
28. Urban, C., Tasson, C.: Nominal techniques in isabelle/hol. In: Automated Deduction - CADE-20. pp. 38–53 (2005). https://doi.org/10.1007/11532231_4
29. Vaandrager, F.W.: Model learning. Commun. ACM **60**(2), 86–95 (2017). https://doi.org/10.1145/2967606