Learning Nominal Automata *

Joshua Moerman
Radboud University, The Netherlands
joshua.moerman@cs.ru.nl

Matteo Sammartino
Alexandra Silva
University College London, UK
{m.sammartino,alexandra.silva}@ucl.ac.uk

Bartek Klin
Michał Szynwelski
University of Warsaw, Poland
{klin,szynwelski}@mimuw.edu.pl

Abstract

We present an Angluin-style algorithm to learn nominal automata, which are acceptors of languages over infinite (structured) alphabets. The abstract approach we take allows us to seamlessly extend known variations of the algorithm to this new setting. In particular we can learn a subclass of nominal non-deterministic automata. An implementation using a recently developed Haskell library for nominal computation is provided for preliminary experiments.

Categories and Subject Descriptors D.1.1 [*Software*]: Programming Techniques; F.4.3 [*Mathematical Logic and Formal Languages*]: Formal Languages; I.3.2 [*Artificial Intelligence*]: Learning

Keywords Active Learning, (Non)Deterministic Finite Automata, Nominal Automata, Functional Programming

1. Introduction

Automata are a well established computational abstraction with a wide range of applications, including modelling and verification of (security) protocols, hardware, and software systems. In an ideal world, a model would be available before a system or protocol is deployed in order to provide ample opportunity for checking important properties that must hold and only then the actual system would be synthesized from the verified model. Unfortunately, this is not at all the reality: Systems and protocols are developed and coded in short spans of time and if mistakes occur they are most likely found after deployment. In this context, it has become popular to infer or learn a model from a given system just by observing its behaviour or response to certain queries. The learned model can then be used to ensure the system is complying to desired properties or to detect bugs and design possible fixes.

Automata learning, or regular inference [3], is a widely used technique for creating an automaton model from observations. The original algorithm [3], by Dana Angluin, works for deterministic finite automata, but since then has been extended to other types of automata [1, 4, 35], including Mealy machines and I/O automata, and even a special class of context-free grammars. Angluin's algorithm is sometimes referred to as *active learning*, because it is based on direct interaction of the learner with an oracle ("the Teacher") that can answer different types of queries. This is in contrast with *passive* learning, where a fixed set of positive and negative examples is given and no interaction with the system is possible.

In this paper, staying in the realm of active learning, we will extend Angluin's algorithm to a richer class of automata. We are motivated by situations in which a program model, besides control flow, needs to represent basic data flow, where data items are compared for equality (or for other theories such as total ordering). In these situations, values for individual symbols are typically drawn from an infinite domain and automata over *infinite alphabets* become natural models, as witnessed by a recent trend [2, 9, 12, 15, 17].

One of the foundational approaches to formal language theory for infinite alphabets uses the notion of nominal sets [9]. The theory of nominal sets originates from the work of Fraenkel in 1922, and they were originally used to prove the independence of the axiom of choice and other axioms. They have been rediscovered in Computer Science by Gabbay and Pitts [36], as an elegant formalism for modeling name binding, and since then they form the basis of many research projects in the semantics and concurrency community. In a nutshell, nominal sets are infinite sets equipped with symmetries which make them finitely representable and tractable for algorithms. We make crucial use of this feature in the development of a learning algorithm.

Our main contributions are the following.

- A generalization of Angluin's original algorithm to nominal automata. The generalization follows a generic pattern for transporting computation models from finite sets to nominal sets, which leads to simple correctness proofs and opens the door to further generalizations. The use of nominal sets with different symmetries also creates potential for generalization, e.g. to languages with time features [7] or data dependencies represented as graphs [33].
- An extension of the algorithm to nominal non-deterministic automata (nominal NFAs). To the best of our knowledge, this is the first learning algorithm for non-deterministic automata over infinite alphabets. It is important to note that, in the nominal setting, NFAs are strictly more expressive than DFAs. We learn a subclass of the languages accepted by nominal NFAs, which includes all the languages accepted by nominal DFAs. The main advantage of learning NFAs directly is that they can provide exponentially smaller automata when compared to their deterministic counterpart. This can be seen both as a generalization and as an optimization of the algorithm.
- An implementation using our recently developed Haskell library tailored to nominal computation – NLambda [26]. Our implementation is the first non-trivial application of a novel programming paradigm of functional programming over infinite

* Work partially supported by the Polish National Science Centre (NCN) grant 2012/07/E/ST6/03026.

POPL'17, January 15–21, 2017, Paris, France
© 2017 ACM. 978-1-4503-4660-3/17/01...$15.00
http://dx.doi.org/10.1145/3009837.3009879

L* LEARNER

1 $S, E \leftarrow \{\epsilon\}$
2 **repeat**
3 **while** (S, E) is not closed or not consistent
4 **if** (S, E) is not closed
5 find $s_1 \in S$, $a \in A$ such that $row(s_1 a) \neq row(s)$, for all $s \in S$
6 $S \leftarrow S \cup \{s_1 a\}$
7 **if** (S, E) is not consistent
8 find $s_1, s_2 \in S$, $a \in A$, and $e \in E$ such that $row(s_1) = row(s_2)$ and $\mathcal{L}(s_1 ae) \neq \mathcal{L}(s_2 ae)$
9 $E \leftarrow E \cup \{ae\}$
10 Make the conjecture $M(S, E)$
11 **if** the Teacher replies **no**, with a counter-example t
12 $S \leftarrow S \cup \texttt{prefixes}(t)$
13 **until** the Teacher replies **yes** to the conjecture $M(S, E)$.
14 **return** $M(S, E)$

Figure 1. Angluin's algorithm for deterministic finite automata [3]

structures, which allows the programmer to rely on convenient intuitions of searching through infinite sets in finite time.

The paper is organized as follows. In Section 2, we present an overview of our contributions (and the original algorithm) highlighting the challenges we faced in the various steps. In Section 3, we revise some basic concepts of nominal sets and automata. Section 4 contains the core technical contributions of our paper: The new algorithm and proof of correctness. In Section 5, we describe an algorithm to learn nominal non-deterministic automata. Section 6 contains a description of NLambda, details of the implementation, and results of preliminary experiments. Section 7 contains a discussion of related work. We conclude the paper with a discussion section where also future directions are presented.

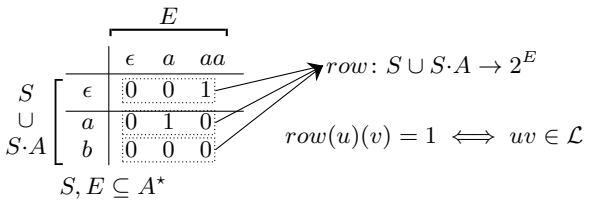
2. Overview of the Approach

In this section, we give an overview of the work developed in the paper through examples. We will start by explaining the original algorithm for regular languages over finite alphabets, and then explain the challenges in extending it to nominal languages.

Angluin's algorithm L* provides a procedure to learn the minimal DFA accepting a certain (unknown) language $\mathcal{L}$. The algorithm has access to a *teacher* which answers two types of queries:

- *membership queries*, consisting of a single word $w \in A^\star$, to which the teacher will reply whether $w \in \mathcal{L}$ or not;
- *equivalence queries*, consisting of a hypothesis DFA H, to which the teacher replies **yes** if $\mathcal{L}(H) = \mathcal{L}$, and **no** otherwise, providing a counterexample $w \in \mathcal{L}(H) \triangle \mathcal{L}$ ($\triangle$ denotes the symmetric difference of two languages).

The learning algorithm works by incrementally building an *observation table*, which at each stage contains partial information about the language $\mathcal{L}$. The algorithm is able to fill the table with membership queries. As an example, and to set notation, consider the following table (over $A = \{a, b\}$).



This table indicates that $\mathcal{L}$ contains at least aa and definitely does not contain the words $\epsilon, a, b, ba, baa, aaa$. Since row is fully determined by the language $\mathcal{L}$, we will from now on refer to an observation table as a pair (S, E), leaving the language $\mathcal{L}$ implicit.

Given an observation table (S, E) one can construct a deterministic automaton $M(S, E) = (Q, q_0, \delta, F)$ where

- $Q = \{row(s) \mid s \in S\}$ is a finite set of states;
- $F = \{row(s) \mid s \in S, row(s)(\epsilon) = 1\} \subseteq Q$ is the set of final states;
- $q_0 = row(\epsilon)$ is the initial state;
- $\delta\colon Q \times A \to Q$ is the transition function given by $\delta(row(s), a) = row(sa)$.

For this to be well-defined, we need to have $\epsilon \in S$ (for the initial state) and $\epsilon \in E$ (for final states), and for the transition function there are two crucial properties of the table that need to hold: Closedness and consistency. An observation table (S, E) is *closed* if for all $t \in S{\cdot}A$ there exists an $s \in S$ such that $row(t) = row(s)$. An observation table (S, E) is *consistent* if, whenever s_1 and s_2 are elements of S such that $row(s_1) = row(s_2)$, for all $a \in A$, $row(s_1 a) = row(s_2 a)$. Each time the algorithm constructs an automaton, it poses an equivalence query to the teacher. It terminates when the answer is **yes**, otherwise it extends the table with the counterexample provided.

2.1 Simple Example of Execution

Angluin's algorithm is displayed in Figure 1. Throughout this section, we will consider the language(s)

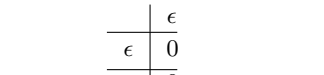
$$\mathcal{L}_n = \{ww \mid w \in A^\star, |w| = n\}$$

If the alphabet A is finite then $\mathcal{L}_n$ is regular for any $n \in \mathbb{N}$, and there is a finite DFA accepting it.

The language $\mathcal{L}_1 = \{aa, bb\}$ looks trivial, but the minimal DFA recognizing it has as many as 5 states. Angluin's algorithm will terminate in (at most) 5 steps. We illustrate some relevant ones.

Step 1**.** We start from $S, E = \{\epsilon\}$, and we fill the entries of the table below by asking membership queries for ϵ, a and b. The table is closed and consistent, so we construct the hypothesis $\mathcal{A}_1$.

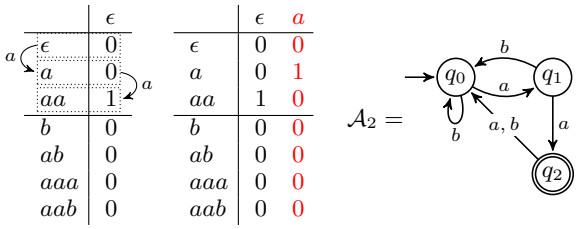
	ϵ
ϵ	0
a	0
b	0



$q_0 = row(\epsilon) = \{\epsilon \mapsto 0\}$

The Teacher replies **no** and gives the counterexample aa, which is in $\mathcal{L}_1$ but it is not accepted by $\mathcal{A}_1$. Therefore, line 12 of the algorithm is triggered and we set $S \leftarrow S \cup \{a, aa\}$.

Step 2**.** The table becomes the one on the left below. It is closed, but not consistent: Rows ϵ and a are identical, but appending a leads to different rows, as depicted. Therefore, line 9 is triggered and an extra column a, highlighted in red, is added. The new table is closed and consistent and a new hypothesis $\mathcal{A}_2$ is constructed.

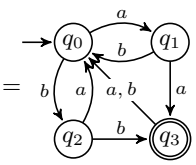


The Teacher again replies **no** and gives the counterexample bb, which should be accepted by $\mathcal{A}_2$ but it is not. Therefore we put $S \leftarrow S \cup \{b, bb\}$.

Step 3**.** The new table is the one on the left. It is closed, but ϵ and b violate consistency, when b is appended. Therefore we add the column b and we get the table on the right, which is closed and consistent. The new hypothesis is $\mathcal{A}_3$.

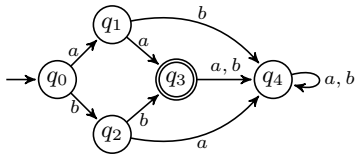
	ϵ	a
ϵ	0	0
a	0	1
aa	1	0
b	0	0
bb	1	0
ab	0	0
aaa	0	0
aab	0	0
ba	0	0
bba	0	0
bbb	0	0

	ϵ	a	b
ϵ	0	0	0
a	0	1	0
aa	1	0	0
b	0	0	1
bb	1	0	0
ab	0	0	0
aaa	0	0	0
aab	0	0	0
ba	0	0	0
bba	0	0	0
bbb	0	0	0

$\mathcal{A}_3 =$ 

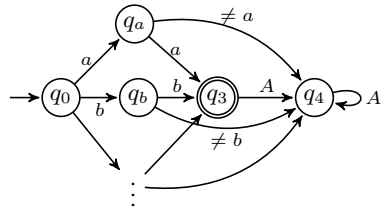
The Teacher replies **no** and provides the counterexample $babb$, so $S \leftarrow S \cup \{ba, bab\}$.

Step 4**.** One more step brings us to the correct hypothesis $\mathcal{A}_4$ (details are omitted).

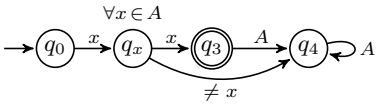
$\mathcal{A}_4 =$ 

2.2 Learning Nominal Languages

Consider now an infinite alphabet $A = \{a, b, c, d, \dots\}$. The language $\mathcal{L}_1$ becomes $\{aa, bb, cc, dd, \dots\}$. Classical theory of finite automata does not apply to this kind of languages, but one may draw an infinite deterministic automaton that recognizes $\mathcal{L}_1$ in the standard sense:

$\mathcal{A}_5 =$ 

where $\xrightarrow{A}$ and $\xrightarrow{\neq a}$ stand for the infinitely-many transitions labelled by elements of A and $A \setminus \{a\}$, respectively. This automaton is infinite, but it can be finitely presented in a variety of ways, for example:



(1)

One can formalize the quantifier notation above (or indeed the "dots" notation above that) in several ways. A popular solution is to consider finite *register automata* [18, 25], i.e., finite automata equipped with a finite number of registers where alphabet letters can be stored and later compared for equality. Our language $\mathcal{L}_1$ is recognized by a simple automaton with four states and one register. The problem of learning registered automata has been successfully attacked before [21].

In this paper, however, we will consider nominal automata [9] instead. These automata ostensibly have infinitely many states, but the set of states can be finitely presented in a way open to effective manipulation. More specifically, in a nominal automaton the set of states is subject to an action of permutations of a set $\mathbb{A}$ of *atoms*, and it is finite up to that action. For example, the set of states of $\mathcal{A}_5$ is:

$$\{q_0, q_3, q_4\} \cup \{q_a \mid a \in A\}$$

and it is equipped with a canonical action of permutations $\pi \colon \mathbb{A} \to \mathbb{A}$ that maps every q_a to $q_{\pi a}$, and leaves q_0, q_3 and q_4 fixed. Technically speaking, the set of states has four *orbits* (one infinite orbit and three fixed points) of the action of the group of permutations of $\mathbb{A}$. Moreover, it is required that in a nominal automaton the transition relation is *equivariant*, i.e., closed under the action of permutations. The automaton $\mathcal{A}_5$ has this property: For example, it has a transition $q_a \xrightarrow{a} q_3$, and for any $\pi \colon \mathbb{A} \to \mathbb{A}$ there is also a transition $\pi(q_a) = q_{\pi(a)} \xrightarrow{\pi(a)} q_3 = \pi(q_3)$.

Nominal automata with finitely many orbits of states are equiexpressive with finite register automata [9], but they have an important theoretical advantage: They are a direct reformulation of the classical notion of finite automaton, where one replaces finite sets with orbit-finite sets and functions (or relations) with equivariant ones. A research programme advocated in [8, 9] is to transport various computation models, algorithms and theorems along this correspondence. This can often be done with remarkable accuracy, and our paper is a witness to this. Indeed, as we shall see, nominal automata can be learned with an algorithm that is almost a verbatim copy of the classical Angluin's one.

Indeed, consider applying Angluin's algorithm to our new language $\mathcal{L}_1$. The key idea is to change the basic data structure: Our observation table (S, E) will be such that *S and E are equivariant subsets of $A^\star$*, i.e., they are closed under the canonical action of atom permutations. In general, such a table has *infinitely many rows and columns*, so the following aspects of the algorithm seem problematic:

line 3: closedness and consistency tests range over infinite sets;

line 5 and 8: finding witnesses for closedness or consistency violations potentially require checking all infinitely many rows;

line 12: every counterexample t has only finitely many prefixes, so it is not clear how one would construct an infinite set S in finite time. However, an infinite S is necessary for the algorithm to ever succeed, because no finite automaton recognizes $\mathcal{L}_1$.

At this stage, we need to observe that due to equivariance of S, E and $\mathcal{L}_1$, the following crucial properties hold:

(P1) the sets S, $S{\cdot}A$ and E admit a *finite* representation up to permutations;

(P2) the function row is such that $row(\pi(s))(\pi(e)) = row(s)(e)$, for all $s \in S$ and $e \in E$, so the observation table admits a finite symbolic representation.

Intuitively, checking closedness and consistency, and finding a witness for their violations, can be done effectively on the representations up to permutations **(P1)**. This is sound, as row is invariant w.r.t. permutations **(P2)**.

We now illustrate these points through a few steps of the algorithm for $\mathcal{L}_1$.

Step 1′**:** We start from $S, E = \{\epsilon\}$. We have $S{\cdot}A = A$, which is infinite but admits a finite representation. In fact, for any $a \in A$, we have $A = \{\pi(a) \mid \pi \text{ is a permutation}\}$. Then, by **(P2)**, $row(\pi(a))(\epsilon) = row(a)(\epsilon) = 0$, for all π, so the first table can be written as:

	ϵ
ϵ	0
a	0

$\mathcal{A}'_1 =$ 

It is closed and consistent. Our hypothesis is $\mathcal{A}'_1$, where $\delta_{\mathcal{A}'_1}(row(\epsilon), x) = row(x) = q_0$, for all $x \in A$. As in **Step** 1, the Teacher replies with the counterexample aa.

Step 2′**.** By equivariance of $\mathcal{L}_1$, the counterexample tells us that *all* words of length 2 with two repeated letters are accepted. Therefore

we extend S with the (infinite!) set of such words. The new symbolic table is:

	ϵ
ϵ	0
a	0
aa	1
ab	0
aaa	0
aab	0

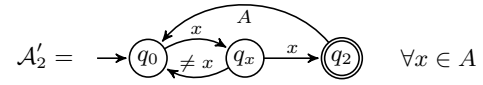
The lower part stands for elements of $S{\cdot}A$. For instance, ab stands for words obtained by appending a fresh letter to words of length 1 (row a). It can be easily verified that all cases are covered. Notice that the table is different from that of **Step** 2: A single b is not in the lower part, because it can be obtained from a via a permutation. The table is closed.

Now, for consistency we need to check $row(\epsilon x) = row(ax)$, for all $a, x \in A$. Again, by **(P2)**, it is enough to consider rows of the table above. Consistency is violated, because $row(a) \neq row(aa)$. We found a "symbolic" witness a for such violation. In order to fix consistency, while keeping E equivariant, we need to add columns for all $\pi(a)$. The resulting table is

	ϵ	a	b	c	...
ϵ	0	0	0	0	...
a	0	1	0	0	...
aa	1	0	0	0	...
ab	0	0	0	0	...
aaa	0	0	0	0	...
aab	0	0	0	0	...

where non-specified entries are 0. Only finitely many entries of the table are relevant: $row(s)$ is fully determined by its values on letters in s and on just one letter not in s. For instance, we have $row(a)(a) = 1$ and $row(a)(a') = 0$, for all $a' \in A \setminus \{a\}$. The table is trivially consistent.

Notice that this step encompasses both **Step** 2 and 3, because the rows b and bb added by **Step** 2 are already represented by a and aa. The hypothesis automaton is



This is again incorrect, but one additional step will give the correct hypothesis automaton, shown earlier in (1).

2.3 Generalization to Non-Deterministic Automata

Since our extension of Angluin's $\mathtt{L}^\star$ algorithm stays close to her original development, exploring extensions of other variations of $\mathtt{L}^\star$ to the nominal setting can be done in a systematic way. We will show how to extend the algorithm $\mathtt{NL}^\star$ for learning NFAs by Bollig et al. [11]. This has practical implications: It is well-known that NFAs are exponentially more succinct than DFAs. This is true also in the nominal setting. However, there are challenges in the extension that require particular care.

- Nominal NFAs are strictly more expressive than nominal DFAs. We will show that the nominal version of $\mathtt{NL}^\star$ terminates for all nominal NFAs that have a corresponding nominal DFA and, more surprisingly, that it is capable of learning some languages that are not accepted by nominal DFAs.
- Language equivalence of nominal NFAs is undecidable. This does not affect the correctness proof, as it assumes a teacher which is able to answer equivalence queries accurately. For our implementation, we will describe heuristics that produce correct results in many cases.

For the learning algorithm the power of non-determinism means that we can make some shortcuts during learning: If we want to make the table closed, we were previously required to find an equivalent row in the upper part; now we may find a sum of rows which, together, are equivalent to an existing row. This means that in some cases fewer rows will be added for closedness.

3. Preliminaries

We recall the notions of nominal sets, nominal automata and nominal regular languages (see [9] for a detailed account).

Let $\mathbb{A}$ be a countable set and let $Perm(\mathbb{A})$ be the set of *permutations on* $\mathbb{A}$, i.e., the bijective functions $\pi\colon \mathbb{A} \to \mathbb{A}$. Permutations form a group where the identity permutation id is the unit element, inverse is functional inverse and multiplication is function composition.

A *nominal set* [36] is a set X plus a function $\cdot\colon Perm(\mathbb{A}) \times X \to X$, interpreting permutations over X. Such function must be a *group action* of $Perm(\mathbb{A})$, i.e., it must satisfy $id \cdot x = x$ and $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$. We say that a finite $A \subseteq \mathbb{A}$ *supports* $x \in X$ whenever, for all π acting as the identity on A, we have $\pi \cdot x = x$. In other words, permutations that only move elements outside A do not affect x. The support of $x \in X$, denoted $\mathtt{supp}(x)$, is the smallest finite set supporting x. We require nominal sets to have *finite support*, meaning that $\mathtt{supp}(x)$ exists for all $x \in X$.

The *orbit* $\mathtt{orb}(x)$ of $x \in X$ is the set of elements in X reachable from x via permutations, explicitly

$$\mathtt{orb}(x) = \{\pi \cdot x \mid \pi \in Perm(\mathbb{A})\}$$

Then X is *orbit-finite* whenever it is a union of finitely many orbits.

Given a nominal set X, a subset $Y \subseteq X$ is *equivariant* if it is preserved by permutations, i.e., $\pi \cdot y \in Y$, for all $y \in Y$. In other words, Y is the union of orbits of X. This definition extends to the notion of an equivariant relation $R \subseteq X \times Y$, by setting $\pi \cdot (x, y) = (\pi \cdot x, \pi \cdot y)$, for $(x, y) \in R$; similarly for relations of greater arity. The *dimension* of nominal set X is the maximal size of $\mathtt{supp}(x)$, for any $x \in X$. Every orbit-finite set has finite dimension.

We define $\mathbb{A}^{(k)} = \{(a_1, \ldots, a_k) \mid a_i \neq a_j \text{ for } i \neq j\}$. For every single-orbit nominal set X with dimension k, there is a surjective equivariant map

$$f_X\colon \mathbb{A}^{(k)} \to X \ .$$

This map can be used to get an upper bound for the number of orbits of $X_1 \times \cdots \times X_n$, for X_i a nominal set with l_i orbits and dimension k_i. Suppose O_i is an orbit of X_i. Then we have a surjection

$$\mathbb{A}^{(k_i)} \times \cdots \times \mathbb{A}^{(k_n)} \xrightarrow{f_{O_1} \times \cdots \times f_{O_n}} O_1 \times \cdots \times O_n$$

stipulating that the codomain cannot have more orbits than the domain. Let $f_{\mathbb{A}}(\{k_i\})$ denote the number of orbits of $\mathbb{A}^{(k_1)} \times \cdots \times \mathbb{A}^{(k_n)}$, for any finite sequence of natural numbers $\{k_i\}$. We can form at most $l = l_1 l_2 \ldots l_n$ tuples of the form $O_1 \times \cdots \times O_n$, so $X_1 \times \cdots \times X_n$ has at most $l f_{\mathbb{A}}(k_1, \ldots, k_n)$ orbits.

For X single-orbit, the *local symmetries* are defined by the group $\{g \in S_k \mid f(x_1, \ldots, x_k) = f(x_{g(1)}, \ldots, x_{g(k)}) \text{ for all } x_i \in X\}$, where k is the dimension of X and S_k is the *symmetric group* of permutations over k distinct elements.

NFAs on sets have a finite state space. We can define *nominal* NFAs, with the requirement that the state space is orbit-finite and the transition relation is equivariant. A nominal NFA is a tuple (Q, A, Q_0, F, δ), where:

- Q is an orbit-finite nominal set of *states*;
- A is an orbit-finite nominal alphabet;
- $Q_0, F \subseteq Q$ are equivariant subsets of *initial* and *final states*;
- $\delta \subseteq Q \times A \times Q$ is an equivariant *transition relation*.

A nominal DFA is a special case of nominal NFA where $Q_0 = \{q_0\}$ and the transition relation is an equivariant function $\delta\colon Q \times A \to Q$.

Equivariance here can be rephrased as requiring $\delta(\pi \cdot q, \pi \cdot a) = \pi \cdot \delta(q, a)$. In most examples we take the alphabet to be $A = \mathbb{A}$, but it can be any orbit-finite nominal set. For instance, $A = Act \times \mathbb{A}$, where Act is a finite set of actions, represents actions $act(x)$ with one parameter $x \in \mathbb{A}$ (actions with arity n can be represented via n-fold products of $\mathbb{A}$).

A language $\mathcal{L}$ is *nominal regular* if it is recognized by a nominal DFA. The theory of nominal regular languages recasts the classical one using nominal concepts. A nominal Myhill-Nerode-style *syntactic congruence* is defined: $w, w' \in A^\star$ are equivalent w.r.t. $\mathcal{L}$, written $w \equiv_{\mathcal{L}} w'$, whenever

$$wv \in \mathcal{L} \iff w'v \in \mathcal{L}$$

for all $v \in A^\star$. This relation is equivariant and the set of equivalence classes $[w]_{\mathcal{L}}$ is a nominal set.

Theorem 1 (Myhill-Nerode theorem for nominal sets [9])**.** *Let $\mathcal{L}$ be a regular nominal language. The following conditions are equivalent:*

1. *the set of equivalence classes of $\equiv_{\mathcal{L}}$ is orbit-finite;*
2. *$\mathcal{L}$ is recognized by a nominal DFA.*

Unlike what happens for ordinary regular languages, nominal NFAs and nominal DFAs *are not equi-expressive*. Here is an example of a language accepted by a nominal NFA, but not by a nominal DFA:

$$\mathcal{L}_{eq} = \{a_1 \dots a_n \mid a_i = a_j, \text{for some } i < j \in \{1, \dots, n\}\}$$

In the theory of nominal regular languages, several problems are decidable: Language inclusion and minimality test for nominal DFAs. Moreover, orbit-finite nominal sets can be finitely-represented, and so can be manipulated by algorithms. This is the key idea underpinning our implementation.

3.1 Different Atom Symmetries

An important advantage of nominal set theory as considered in [9] is that it retains most of its properties when the structure of atoms $\mathbb{A}$ is replaced with an arbitrary infinite relational structure subject to a few model-theoretic assumptions. An example alternative structure of atoms is the total order of rational numbers $(\mathbb{Q}, <)$, with the group of monotone bijections of $\mathbb{Q}$ taking the role of the group of all permutations. The theory of nominal automata remains similar, and an example nominal language over the atoms $(\mathbb{Q}, <)$ is:

$$\{a_1 \dots a_n \mid a_i \leq a_j, \text{for some } i < j \in \{1, \dots, n\}\}$$

which is recognized by a nominal DFA over those atoms.

To simplify the presentation, in this paper we concentrate on the "equality atoms" only. Also our implementation of nominal learning algorithms is restricted to equality atoms. However, both the theory and the implementation can be generalized to other atom structures, with the "ordered atoms" $(\mathbb{Q}, <)$ as the simplest other example. We leave the details of this for a future extended version of this paper.

4. Angluin's Algorithm for Nominal DFAs

In our algorithm, we will assume a teacher as described at the start of Section 2. In particular, the teacher is able to answer membership queries and equivalence queries, now in the setting of nominal languages. We fix a target language $\mathcal{L}$, which is assumed to be a nominal regular language.

The learning algorithm for nominal automata, $\nu\mathtt{L}^\star$, will be very similar to $\mathtt{L}^\star$ in Figure 1. In fact, we only change the following lines:

$$\begin{array}{ll} 6' & S \leftarrow S \cup \mathtt{orb}(sa) \\ 9' & E \leftarrow E \cup \mathtt{orb}(ae) \\ 12' & S \leftarrow S \cup \mathtt{prefixes}(\mathtt{orb}(t)) \end{array} \tag{2}$$

The basic data structure is an *observation table* (S, E, T) where S and E are orbit-finite subsets of $A^\star$ and $T : S \cup S{\cdot}A \times E \to 2$ is an equivariant function defined by $T(se) = \mathcal{L}(se)$ for each $s \in S \cup S{\cdot}A$ and $e \in E$. Since T is determined by $\mathcal{L}$ we omit it from the notation. Let $row : S \cup S{\cdot}A \to 2^E$ denote the curried counterpart of T. Let $u \sim v$ denote the relation $row(u) = row(v)$.

Definition 1. The table is called *closed* if for each $t \in S{\cdot}A$ there is a $s \in S$ with $t \sim s$. The table is called *consistent* if for each pair $s_1, s_2 \in S$ with $s_1 \sim s_2$ we have $s_1 a \sim s_2 a$ for all $a \in A$.

The above definitions agree with the abstract definitions given in [24] and we may use some of their results implicitly. The intuition behind the definitions is as follows. Closedness assures us that for each state we have a successor state for each input. Consistency assures us that each state has at most one successor for each input. Together it allows us to construct a well-defined minimal automaton from the observations in the table.

The algorithm starts with a trivial observation table and tries to make it closed and consistent by adding orbits of rows and columns, filling the table via membership queries. When the table is closed and consistent it constructs a hypothesis automaton and poses an equivalence query.

The pseudocode for the nominal version is the same as listed in Figure 1, modulo the changes displayed in (2). However, we have to take care to ensure that all manipulations and tests on the (possibly) infinite sets S, E and A terminate in finite time. We refer to [9] and [36] for the full details on how to represent these structures and provide a brief sketch here. The sets S, E, A and $S{\cdot}A$ can be represented by choosing a representative for each orbit. The function T in turn can be represented by cells $T_{i,j} : \mathtt{orb}(s_i) \times \mathtt{orb}(e_j) \to 2$ for each representative s_i and e_j. Note, however, that the product of two orbits may consist of several orbits, so that $T_{i,j}$ is not a single boolean value. Each cell is still orbit-finite and can be filled with only finitely many membership queries. Similarly the curried function row can be represented by a finite structure.

To check whether the table is closed, we observe that if we have a corresponding row $s \in S$ for some $t \in S{\cdot}A$, this holds for any permutation of t. Hence it is enough to check the following: For all representatives $t \in S{\cdot}A$ there is a representative $s \in S$ with $row(t) = \pi \cdot row(s)$ for some permutation π. Note that we only have to consider finitely many permutations, since the support is finite and so we can decide this property. Furthermore if the property does not hold, we immediately find a witness represented by t.

Consistency is a bit more complicated, but it is enough to consider the set of inconsistencies, $\{(s_1, s_2, a, e) \mid row(s_1) = row(s_2) \wedge row(s_1 a)(e) \neq row(s_2 a)(e)\}$. It is an equivariant subset of $S \times S \times A \times E$ and so it is orbit-finite. Hence we can decide emptiness and obtain representatives if it is non-empty.

Constructing the hypothesis happens in the same way as before (Section 2), where we note the state space is orbit-finite since it is a quotient of S. Moreover the function row is equivariant, so all structure (Q_0, F and δ) is equivariant as well.

The representation given above is not the only way to represent nominal sets. For example, first-order definable sets can be used as well [26]. From now on we assume to have set theoretic primitives so that each line in Figure 1 is well defined.

4.1 Correctness

To prove correctness we only have to prove that the algorithm terminates, that is, only finitely many hypotheses will be produced. Correctness follows trivially from termination since the last step of the algorithm is an equivalence query to the teacher inquiring whether an hypothesis automaton accepts the target language. We start out by listing some facts about observation tables.

Lemma 1. *The relation $\sim$ is an equivariant equivalence relation. Furthermore, for all $u, v \in S$ we have that $u \equiv_{\mathcal{L}} v$ implies $u \sim v$.*

Lemma 1 implies that at any stage of the algorithm the number of orbits of $S/\sim$ does not exceed the number of orbits of the minimal acceptor with state space $A^\star/\equiv_{\mathcal{L}}$ (recall that $\equiv_{\mathcal{L}}$ is the nominal Myhill-Nerode equivalence relation). Moreover, the following lemma shows that the dimension of the state space never exceeds the dimension of the minimal acceptor. Recall that the dimension is the maximal size of the support of any state, which is different than the number of orbits.

Lemma 2. *We have $\mathtt{supp}([u]_\sim) \subseteq \mathtt{supp}([u]_{\equiv_{\mathcal{L}}}) \subseteq \mathtt{supp}(u)$ for all $u \in S$.*

Lemma 3. *The automaton constructed from a closed and consistent table is minimal.*

Proof. Follows from the categorical perspective given in [24]. □

We note that the constructed automaton is consistent with the table (we use that the set S is prefix-closed and E is suffix-closed [3]). The following lemma shows that there are no strictly "smaller" automata consistent with the table. So the automaton is not just minimal, it is minimal w.r.t. the table.

Lemma 4. *Let H be the automaton associated with a closed and consistent table (S, E). If M' is an automaton consistent with (S, E) (meaning that $se \in \mathcal{L}(M') \iff se \in \mathcal{L}(H)$ for all $s \in S \cup S{\cdot}A$ and $e \in E$) and M' has at most as many orbits as H, then there is a surjective map $f : Q_{M'} \to Q_H$. If moreover*

- *M's dimension is bounded by the dimension of H, i.e. $\mathtt{supp}(m) \subseteq \mathtt{supp}(f(m))$ for all Q'_M, and*
- *M' has no fewer local symmetries than H, i.e. $\pi \cdot f(m) = f(m)$ implies $\pi \cdot m = m$ for all $m \in Q'_M$,*

then f defines an isomorphism $M' \cong H$ of nominal DFAs.

Proof. (All maps in this proof are equivariant.) Define a map $row' : Q'_M \to 2^E$ by restricting the language map $Q'_M \to 2^{A^\star}$ to E. First, observe that $row'(\delta'(q'_0, s)) = row(s)$ for all $s \in S \cup S{\cdot}A$, since $\epsilon \in E$ and M' is consistent with the table. Second, we have $\{row'(\delta'(q'_0, s))|s \in S\} \subseteq \{row'(q)|q \in M'\}$.

Let n be the number of orbits of H. The former set has n orbits by the first observation, the latter set has at most n orbits by assumption. We conclude that the two sets (both being equivariant) must be equal. That means that for each $q \in M'$ there is a $s \in S$ such that $row'(q) = row(s)$. We see that $row' : M' \twoheadrightarrow \{row'(\delta'(q'_0, s))|s\} = H$ is a surjective map. Since a surjective map cannot increase the dimensions of orbits and the dimensions of M' are bounded, we note that the dimensions of the orbits in H and M' have to agree. Similarly, surjective maps preserve local symmetries. This map must hence be an isomorphism of nominal sets. Note that $row'(q) = row'(\delta'(q'_0, s))$ implies $q = \delta'(q'_0, s)$.

It remains to prove that it respects the automaton structures. It preserve the initial state: $row'(q'_0) = row(\delta'(q'_0, \epsilon)) = row(\epsilon)$. Now let $q \in M'$ be a state and $s \in S$ such that $row'(q) = row(s)$. It preserves final states: $q \in F' \iff row'(q)(\epsilon) = 1 \iff row(s)(\epsilon) = 1$. Finally, it preserves the transition structure:

$$\begin{aligned} row'(\delta'(q, a)) &= row'(\delta'(\delta'(q'_0, s), a)) = row'(\delta'(q'_0, sa)) \\ &= row(sa) = \delta(row(s), a) \end{aligned}$$

□

The above proof is an adaptation of Angluin's proof for automata over sets. We will now prove termination of the algorithm by proving that all steps are productive.

Theorem 2. *The algorithm terminates and is hence correct.*

Proof. Provided that the if-statements and set operations terminate, we are left proving that the algorithm adds (orbits of) rows and columns only finitely often. We start by proving that a table can be made closed and consistent in finite time.

If the table is not closed, we find a row $s_1 \in S{\cdot}A$ such that $row(s_1) \neq row(s)$ for all $s \in S$. The algorithm then adds the orbit containing s_1 to S. Since s_1 was nonequivalent to all rows, we find that $S \cup \mathtt{orb}(t)/\sim$ has strictly more orbits than $S/\sim$. Since orbits of $S/\sim$ cannot be more than those of $A^\star/\equiv_{\mathcal{L}}$, this happens finitely often.

Columns are added in case of an inconsistency. Here the algorithm finds two elements $s_1, s_2 \in S$ with $row(s_1) = row(s_2)$ but $row(s_1ae) \neq row(s_2ae)$ for some $a \in A$ and $e \in E$. Adding ae to E will ensure that $row'(s_1) \neq row'(s_2)$ (row' is the function belonging to the updated observation table). If the two elements $row'(s_1), row'(s_2)$ are in different orbits, the number of orbits is increased. If they are not in the same orbit, we have $row'(s_2) = \pi \cdot row'(s_1)$ for some permutation π. Using $row(s_1) = row(s_2)$ and $row'(s_1) \neq row'(s_2)$ we have:

$$row(s_1) = \pi \cdot row(s_1) \qquad row'(s_1) \neq \pi \cdot row'(s_1)$$

Consider all such π and suppose there is a π and $x \in \mathtt{supp}(row(s_1))$ such that $\pi \cdot x \notin \mathtt{supp}(row(s_1))$. Then we find that $\pi \cdot x \in \mathtt{supp}(row'(s_1))$, and so the support of the row has grown. By Lemma 2 this happens finitely often. Suppose such π and x do not exist, then we consider the finite group $R = \{\rho|_{\mathtt{supp}([s_1]_\sim)} \,|\, row(s_1) = \rho \cdot row(s_1)\}$. We see that $\{\rho|_{\mathtt{supp}([s_1]_\sim)} \,|\, row'(s_1) = \rho{\cdot}row'(s_1)\}$ is a proper subgroup of R. So, adding a column in this case decreases the size of the group R, which can happen only finitely often. In this case a local symmetry is removed.

In short, the algorithm will succeed in producing a hypothesis in each round. It remains to prove that it needs only finitely many equivalence queries.

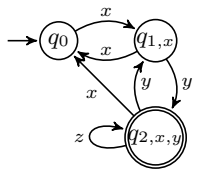
Let (S, E) be the closed and consistent table and H its corresponding hypothesis. If it is incorrect a second hypothesis H' will be constructed which is consistent with the old table (S, E). The two hypotheses are nonequivalent, as H' will handle the counter example correctly and H does not. Therefore, H' will have at least one orbit more, one local symmetry less, or one orbit will have strictly bigger dimension (Lemma 4), all of which can only happen finitely often. □

We remark that all the lemmas and proofs as above are close to the original ones of Angluin. However, two things are crucially different. First, adding a column does not always increase the number of (orbits of) states. It can happen that by adding a column a bigger support is found or that a local symmetry is broken. Second, the new hypothesis does not necessarily have more states, again it might have bigger dimensions or less local symmetries.

From the proof Theorem 2 we observe moreover that the way we handle counterexamples is not crucial. Any other method which ensures a non-equivalent hypothesis will work. In particular our algorithm is easily adapted to include optimizations such as the ones in [37] and [31], where counterexamples are added as columns.

4.2 Example

Consider the target automaton in Figure 2 and an observation table T_1 at some stage during the algorithm. We remind the reader that the table is represented in a symbolic way: The sequences in the rows and columns stand for whole orbits and the cells denote functions from the product of the orbits to 2. Since the cells can consist of multiple orbits, where each orbit is allowed to have a different value, we use a formula to specify which orbits have a 1.



T_1	ϵ
ϵ	0
a	0
ab	1
aa	0
aba	0
abb	0
abc	1

T_2	ϵ	a'
ϵ	0	0
a	0	$\begin{cases}1 & a' \neq a \\ 0 & \text{else}\end{cases}$
ab	1	$\begin{cases}1 & a' \neq a,b \\ 0 & \text{else}\end{cases}$
aa	0	0
aba	0	0
abb	0	$\begin{cases}1 & a' \neq a \\ 0 & \text{else}\end{cases}$
abc	1	$\begin{cases}1 & a' \neq a,b \\ 0 & \text{else}\end{cases}$

T_3	$b'a'$
ϵ	1
a	$\begin{cases}1 & a \neq a',b' \\ 0 & \text{else}\end{cases}$
ab	$\begin{cases}1 & (b' \neq a,b \wedge a' \neq a,b) \vee (b' = b \wedge a' \neq a) \\ 0 & \text{else}\end{cases}$
aa	1
aba	1
abb	$\begin{cases}1 & a \neq a',b' \\ 0 & \text{else}\end{cases}$
abc	$\begin{cases}1 & (b' \neq a,b \wedge a' \neq a,b) \vee (b' = b \wedge a' \neq a) \\ 0 & \text{else}\end{cases}$

Figure 2. Example automaton to be learnt and three subsequent tables computed by $\nu\mathtt{L}^\star$. In the automaton, x, y, z denote distinct atoms. In T_3 we only show a relevant column.

The table T_1 at some stage of the algorithm has to be checked for closedness and consistency. We note that it is definitely closed. For consistency we check the rows $row(\epsilon)$ and $row(a)$ which are equal. Observe, however, that $row(\epsilon b)(\epsilon) = 0$ and $row(ab)(\epsilon) = 1$, so we have an inconsistency. The algorithm adds the orbit $\mathtt{orb}(b)$ as column and extends the table, obtaining T_2. We note that, in this process, the number of orbits did grow, as the two rows are split. Furthermore we see that both $row(a)$ and $row(ab)$ have empty support in T_1, but not in T_2, because $row(a)(a')$ depends on a' being equal or different from a, similarly for $row(ab)(a')$.

The table T_2 is still not consistent as we see that $row(ab) = row(ba)$ but $row(abb)(c) = 1$ and $row(bab)(c) = 0$. Hence the algorithm adds the columns $\mathtt{orb}(bc)$, obtaining table T_3. We note that in this case, no new orbits are obtained and no support has grown. In fact, the only change here is that the local symmetry between $row(ab)$ and $row(ba)$ is removed. This last table, T_3, is closed and consistent and will produce the correct hypothesis.

4.3 Query Complexity

In this section, we will analyse the number of queries made by the algorithm in the worst case. Let M be the minimal target automaton with n orbits and of dimension k. We will use log in base two.

Lemma 5. *The number of equivalence queries $E_{n,k}$ is $O(nk \log k)$.*

Proof. By Lemma 4 each hypothesis will be either 1) bigger in the number of orbits, which is bounded by n, or 2) bigger in the dimension of an orbit, which is bounded by k or 3) smaller in local symmetries of an orbit. For the last part we want to know how long a subgroup series of the permutation group S_k can be. This is bounded by the number of divisors of $k!$, as each subgroup divides the order of the group. We can easily bound the number of divisors of any m by $\log m$ and so one can at take a subgroup at most $k \log k$ times when starting with S_k.

Since the hypothesis will grow monotonically in the number of orbits and for each orbit will grow monotonically w.r.t. the remaining two dimensions, the number of equivalence queries is bound by $n + n(k + k \log k)$. □

Next we will give a bound for the size of the table.

Lemma 6. *The table has at most $n + mE_{n,k}$ orbits in S with sequences of at most length $n + m$, where m is the length of the longest counter example given by the teacher. The table has at most $n(k + k \log k + 1)$ orbits in E of at most length $n(k + k \log k + 1)$*

Proof. In the termination proof we noted that rows are added at most n times. In addition (all prefixes of) counter examples are added as rows which add another $mE_{n,k}$ rows. Obviously counter examples are of length at most m and are extended at most n times, making the length at most $m + n$ in the worst case.

For columns we note that one of three dimensions approaches a bound similarly to the proof of Lemma 5. So at most $n(k+k \log k+1)$ columns are added. Since they are suffix closed, the length is at most $n(k + k \log k + 1)$. □

Let p and l denote respectively the dimension and the number of orbits of A.

Lemma 7. *The number of orbits in the lower part of the table, $S{\cdot}A$, is bounded by $(n + mE_{n,k})lf_{\mathbb{A}}(p(n + m), p)$.*

Proof. Any sequence in S is of length at most $n+m$, so it contains at most $p(n+m)$ distinct atoms. When we consider $S{\cdot}A$, the extension can either reuse atoms from those $p(n+m)$, or none at all. Since the extra letter has at most p distinct atoms, the set $\mathbb{A}^{(p(n+m))} \times \mathbb{A}^{(p)}$ gives a bound $f_{\mathbb{A}}(p(n+m), p)$ for the number of orbits of $O_S \times O_A$, with O_X an orbit of X. Multiplying by the number of such ordered pairs, namely $(n + mE_{n,k})l$, gives a bound for $S{\cdot}A$. □

Let $C_{n,k,m} = (n + mE_{n,k})(lf_{\mathbb{A}}(p(n + m), p) + 1)n(k + k \log k + 1)$ be the maximal number of cells in the table. We note that this number is polynomial in k, l, m and n but not in p.

Corollary 1. *The number of membership queries is bounded by $C_{n,k,m} f_{\mathbb{A}}(p(n + m), pn(k + k \log k + 1))$.*

5. Learning Non-Deterministic Nominal Automata

In this section, we introduce a variant of $\nu\mathtt{L}^\star$, which we call $\nu\mathtt{NL}^\star$, where the learnt automaton is non-deterministic. It will be based on $\mathtt{NL}^\star$[11], an Angluin-style algorithm for learning NFAs. The algorithm is shown in Figure 3. We first illustrate $\mathtt{NL}^\star$, then we discuss its extension to nominal automata.

$\mathtt{NL}^\star$ crucially relies on the use of *residual finite-state automata* (RFSA) [19], which are NFAs admitting *unique minimal canonical representatives*. The states of this automaton correspond to Myhill-Nerode right-congruence classes, but can be exponentially smaller than the corresponding minimal DFA: *Composed* states, language-equivalent to sets of other states, can be dropped. The algorithm $\mathtt{NL}^\star$ equips the observation table (S, E) with a union operation, allowing for the detection of *composed* and *prime* rows.

Definition 2. Let $(row(s_1) \sqcup row(s_2))(e) = row(s_1)(e) \vee row(s_1)(e)$ (regarding cells as booleans). This operation induces an ordering between rows: $row(s_1) \sqsubseteq row(s_2)$ whenever $row(s_1)(e) = 1$ implies $row(s_2)(e) = 1$, for all $e \in E$.

$\mathtt{NL}^\star$ LEARNER

1 $S, E \leftarrow \{\epsilon\}$
2 **repeat**
3 **while** (S, E) is not RFSA-closed or not RFSA-consistent
4 **if** (S, E) is not RFSA-closed
5 find $s \in S, a \in A$ such that
$row(sa) \in PR(S, E) \setminus PR^\top(S, E)$
6 $S \leftarrow S \cup \{sa\}$
7 **if** (S, E) is not RFSA-consistent
8 find $s_1, s_2 \in S$, $a \in A$, and $e \in E$ such that
$row(s_1) \sqsubseteq row(s_2)$ and
$\mathcal{L}(s_1 ae) = 1, \mathcal{L}(s_2 ae) = 0$
9 $E \leftarrow E \cup \{ae\}$
10 Make the conjecture $N(S, E)$
11 **if** the Teacher replies **no**, with a counter-example t
12 $E \leftarrow E \cup \mathtt{suffixes}(t)$
13 **until** the Teacher replies **yes** to the conjecture $N(S, E)$.
14 **return** $N(S, E)$

Figure 3. Bollig et al.'s algorithm for learning NFAs [11]

A row $row(s)$ is composed if $row(s) = row(s_1) \sqcup \ldots \sqcup row(s_n)$, for $row(s_i) \neq row(s)$. Otherwise it is prime. We denote by $PR^\top(S, E)$ the rows in the top part of the table (ranging over S) which are prime w.r.t. the whole table (not only w.r.t. the top part). We write $PR(S, E)$ for all the prime rows of (S, E).

As in $\mathtt{L}^\star$, states of hypothesis automata will be rows of (S, E) but, as the aim is to construct a minimal RFSA, only prime rows are picked. New notions of closedness and consistency are introduced, to reflect features of RFSAs.

Definition 3. A table (S, E) is:
- *RFSA-closed* if, for all $t \in S{\cdot}A$, $row(t) = \bigsqcup\{row(s) \in PR^\top(S, E) \mid row(s) \sqsubseteq row(t)\}$;
- *RFSA-consistent* if, for all $s_1, s_2 \in S$ and $a \in A$, $row(s_1) \sqsubseteq row(s_2)$ implies $row(s_1 a) \sqsubseteq row(s_2 a)$.

If (S, E) is not RFSA-closed, then there is a row in the bottom part of the table which is prime, but not contained in the top part. This row is then added to S (line 5). If (S, E) is not RFSA-consistent, then there is a suffix which does not preserve the containment of two existing rows, so those rows are actually incomparable. A new column is added to distinguish those rows (line 8). Notice that counterexamples supplied by the teacher are added *to columns* (line 12). Indeed, in [11] it is shown that treating the counterexamples as in the original $\mathtt{L}^\star$, namely adding them to rows, does not lead to a terminating algorithm.

Definition 4. Given a RFSA-closed and RFSA-consistent table (S, E), the conjecture automaton is $N(S, E) = (Q, Q_0, F, \delta)$, where:
- $Q = PR^\top(S, E)$;
- $Q_0 = \{r \in Q \mid r \sqsubseteq row(\epsilon)\}$;
- $F = \{r \in Q \mid r(\epsilon) = 1\}$;
- the transition relation $\delta \subseteq Q \times A \times Q$ is given by $\delta(row(s), a) = \{r \in Q \mid r \sqsubseteq row(sa)\}$.

As observed in [11], $N(S, E)$ is not necessarily a RFSA, but it is a canonical RFSA if it is consistent with (S, E). If the algorithm terminates, then $N(S, E)$ must be consistent with (S, E), which ensures correctness. The termination argument is more involved than $\mathtt{L}^\star$, but still it relies on the minimal DFA.

Developing an algorithm to learn nominal NFAs is not an obvious extension of $\mathtt{NL}^\star$: Non-deterministic nominal languages strictly contain nominal regular languages, so it is not clear what the developed algorithm should be able to learn. To deal with this, we introduce a nominal notion of RFSAs. They are a *proper subclass* of nominal NFAs, because they recognize nominal regular languages. Nonetheless, they are more succinct than nominal DFAs.

5.1 Nominal Residual Finite-State Automata

Let $\mathcal{L}$ be a nominal regular language and let u be a finite string. The *derivative* of $\mathcal{L}$ w.r.t. u is $u^{-1}\mathcal{L} = \{v \in A^\star \mid uv \in \mathcal{L}\}$. A set $\mathcal{L}' \subseteq \mathbb{A}^*$ is a *residual* of $\mathcal{L}$ if there is u with $\mathcal{L}' = u^{-1}\mathcal{L}$. Note that a residual might not be equivariant, but it does have a finite support. We write $R(\mathcal{L})$ for the set of residuals of $\mathcal{L}$. Residuals form an orbit-finite nominal set: They are in bijection with the state-space of the minimal nominal DFA for $\mathcal{L}$.

A *nominal residual finite-state automaton* for $\mathcal{L}$ is a nominal NFA whose states are subsets of such minimal automaton. Given a state q of an automaton, we write $\mathcal{L}(q)$ for the set of words leading from q to a set of states containing a final one.

Definition 5. A *nominal residual finite-state automaton* (nominal RFSA) is a nominal NFA $\mathcal{A}$ such that $\mathcal{L}(q) \in R(\mathcal{L}(\mathcal{A}))$, for all $q \in Q_\mathcal{A}$.

Intuitively, all states of a nominal RSFA recognize residuals, but not all residuals are recognized by a single state: There may be a residual $\mathcal{L}'$ and a set of states Q' such that $\mathcal{L}' = \bigcup_{q \in Q'} \mathcal{L}(q)$, but no state q' is such that $\mathcal{L}(q') = \mathcal{L}'$. A residual $\mathcal{L}'$ is called *composed* if it is equal to the union of the components it strictly contains, explicitly

$$\mathcal{L}' = \bigcup\{\mathcal{L}'' \in R(\mathcal{L}) \mid \mathcal{L}'' \subsetneq \mathcal{L}'\} \ ;$$

otherwise it is called *prime*. In an ordinary RSFA, composed residuals have finitely-many components. This is not the case in a nominal RFSA. However, the set of components of $\mathcal{L}'$ always has a finite support, namely $\mathtt{supp}(\mathcal{L}')$.

The set of prime residuals $PR(\mathcal{L})$ is an orbit-finite nominal set, and can be used to define a *canonical* nominal RFSA for $\mathcal{L}$, which has the minimal number of states and the maximal number of transitions. This can be regarded as obtained from the minimal nominal DFA, by removing composed states and adding all initial states and transitions that do not change the recognized language. This automaton is necessarily unique.

Lemma 8. *Let the* canonical *nominal RSFA of $\mathcal{L}$ be (Q, Q_0, F, δ) such that:*
- $Q = PR(\mathcal{L})$;
- $Q_0 = \{\mathcal{L}' \in Q \mid \mathcal{L}' \subseteq \mathcal{L}\}$;
- $F = \{\mathcal{L}' \in Q \mid \epsilon \in \mathcal{L}'\}$;
- $\delta(\mathcal{L}_1, a) = \{\mathcal{L}_2 \in Q \mid \mathcal{L}_2 \subseteq a^{-1}\mathcal{L}_1\}$.

It is a well-defined nominal NFA accepting $\mathcal{L}$.

5.2 $\nu\mathtt{NL}^\star$

Our nominal version of $\mathtt{NL}^\star$ again makes use of an observation table (S, E) where S and E are equivariant subsets of A^* and row is an equivariant function. As in the basic algorithm, we equip (S, E) with a union operation $\sqcup$ and row containment relation $\sqsubseteq$, defined as in Definition 2. It is immediate to verify that $\sqcup$ and $\sqsubseteq$ are equivariant.

Our algorithm is a simple modification of the algorithm in Figure 3, where a few lines are replaced:

$6'$ $S \leftarrow S \cup \mathtt{orb}(sa)$
$9'$ $E \leftarrow E \cup \mathtt{orb}(ae)$
$12'$ $E \leftarrow E \cup \mathtt{suffixes}(\mathtt{orb}(t))$

Switching to nominal sets, several decidability issues arise. The most critical one is that rows may be the union of infinitely many component rows, as happens for residuals of nominal languages,

so finding all such components can be challenging. We adapt the notion of composed to rows: $row(t)$ is composed whenever

$$row(t) = \bigsqcup\{row(s) \mid row(s) \sqsubset row(t)\} \ .$$

where $\sqsubset$ is *strict* row inclusion; otherwise $row(t)$ is prime.

We now check that relevant parts of our algorithm terminate.

Row Containment Check. The basic containment check $row(s) \sqsubseteq row(t)$ is decidable, as $row(s)$ and $row(t)$ are supported by the finite supports of s and t respectively.

Line 3: RFSA-Closedness and RFSA-Consistency Checks. We first show that prime rows form orbit-finite nominal sets.

Lemma 9. *$PR(S,E)$, $PR^{\top}(S,E)$ and $PR(S,E) \setminus PR^{\top}(S,E)$ are orbit-finite nominal sets.*

Consider now RFSA-closedness. It requires computing the set $C(row(t))$ of components of $row(t)$ contained in $PR^{\top}(S,E)$ (possibly including $row(t)$). This may not be equivariant under permutations $Perm(\mathbb{A})$, but it is if we pick a subgroup.

Lemma 10. *The set $C(row(t))$ has the following properties:*

1. $\mathtt{supp}(C(row(t))) \subseteq \mathtt{supp}(row(t))$.
2. *it is equivariant and orbit-finite under the action of the group*

$$G_t = \{\pi \in Perm(\mathbb{A}) \mid \pi{\restriction}_{\mathtt{supp}(row(t))} = id\}$$

of permutations fixing $\mathtt{supp}(row(t))$.

We established that $C(row(t))$ can be effectively computed, and the same holds for $\bigsqcup C(row(t))$. In fact, $\bigsqcup$ is equivariant w.r.t the whole $Perm(\mathbb{A})$ and then, in particular, w.r.t. G_t, so it preserves orbit-finiteness. Now, to check $row(t) = \bigsqcup C(row(t))$, we can just pick one representative of every orbit of $S{\cdot}A$, because we have $C(\pi \cdot row(t)) = \pi \cdot C(row(t))$ and permutations distribute over $\sqcup$, so permuting both sides of the equation gives again a valid equation.

For RFSA-consistency, consider the two sets:

$$N = \{(s_1, s_2) \in S \times S \mid row(s_1) \sqsubseteq row(s_2)\}$$
$$M = \{(s_1, s_2) \in S \times S \mid \forall a \in A : row(s_1 a) \sqsubseteq row(s_2 a)\}$$

They are both orbit-finite nominal sets, by equivariance of row, $\sqsubseteq$ and A. We can check RFSA-consistency in finite time by picking orbit representatives from N and M. For each representative $n \in N$, we look for a representative $m \in M$ and a permutation π such that $n = \pi \cdot m$. If no such m and π exist, then n does not belong to any orbit of M, so it violates RFSA-consistency.

Lines 5 and 8: Finding Witnesses for Violations. We can find witnesses by comparing orbit representatives of orbit-finite sets, as we did with RFSA-consistency. Specifically, we can pick representatives in $S \times A$ and $S \times S \times A \times E$ and check them against the following orbit-finite nominal sets:

- $\{(s,a) \in S \times A \mid row(sa) \in PR(S,E) \setminus PR^{\top}(S,E)\}$;
- $\{(s_1, s_2, a, e) \in S \times S \times A \times E \mid row(s_1 a)(e) = 1, row(s_2 a)(e) = 0, row(s_1) \sqsubseteq row(s_2)\}$;

5.3 Correctness

Now we prove correctness and termination of the algorithm. First, we prove that hypothesis automata are nominal NFAs.

Lemma 11. *The hypothesis automaton $N(S,E)$ (see Definition 4) is a nominal NFA.*

$N(S,E)$, as in ordinary $\mathtt{NL}^\star$, is not always a nominal RFSA. However, we have the following.

Theorem 3. *If the table (S,E) is RFSA-closed, RFSA-consistent and $N(S,E)$ is consistent with (S,E), then $N(S,E)$ is a canonical nominal RFSA.*

This is proved in [10] for ordinary RFSAs, using the standard theory of regular languages. The nominal proof is exactly the same, using derivatives of nominal regular languages and nominal RFSAs as defined in Section 5.1.

Lemma 12. *The table (S,E) cannot have more than n orbits of distinct rows, where n is the number of orbits of the minimal nominal DFA for the target language.*

Proof. Rows are residuals of $\mathcal{L}$, which are states of the minimal nominal DFA for $\mathcal{L}$, so orbits cannot be more than n. □

Theorem 4. *The algorithm $\nu\mathtt{NL}^\star$ terminates and returns the canonical nominal RFSA for $\mathcal{L}$.*

Proof. If the algorithm terminates, then it must return the canonical nominal RFSA for $\mathcal{L}$ by Theorem 3. We prove that a table can be made RFSA-closed and RFSA-consistent in finite time. This is similar to the proof of Theorem 2 and is inspired by the proof [10, Theorem 3].

If the table is not RFSA-closed, we find a row $s \in S{\cdot}A$ such that $row(s) \in PR(S,E) \setminus PR^{\top}(S,E)$. The algorithm then adds $\mathtt{orb}(s)$ to S. Since s was nonequivalent to all upper prime rows, and thus from all the rows indexed by S, we find that $S \cup \mathtt{orb}(t)/{\sim}$ has strictly more orbits than $S/{\sim}$ (recall that $s \sim t \iff row(s) = row(t)$). This addition can only be done finitely-many times, because the number of orbits of $S/{\sim}$ is bounded, by Lemma 12.

Now, the case of RFSA-consistency needs some additional notions. Let R be the (orbit-finite) nominal set of all rows, and let $I = \{(r, r') \in R \times R \mid r \sqsubset r'\}$ be the set of all inclusion relations among rows. The set I is orbit-finite. In fact, consider

$$J = \{(s,t) \in (S \cup S{\cdot}A) \times (S \cup S{\cdot}A) \mid row(s) \sqsubset row(t)\}$$

This set is an equivariant, thus orbit-finite, subset of $(S \cup S{\cdot}A) \times (S \cup S{\cdot}A)$. The set I is the image of J via $row \times row$, which is equivariant, so it preserves orbit-finiteness.

Now, suppose the algorithm finds two elements $s_1, s_2 \in S$ with $row(s_1) \sqsubseteq row(s_2)$ but $row(s_1 a)(e) = 1$ and $row(s_2 a)(e) = 0$ for some $a \in A$ and $e \in E$. Adding a column to fix RFSA-consistency may: **C1)** increase orbits of $(S \cup S{\cdot}A)/{\sim}$, or; **C2)** decrease orbits of I, or; **C3)** decrease local symmetries/increase dimension of one orbit of rows. In fact, if no new rows are added (**C1**), we have two cases.

- If $row(s_1) \sqsubset row(s_2)$, i.e., $(row(s_1), row(s_2)) \in I$, then $row'(s_1) \not\sqsubset row'(s_2)$, where row' is the new table. Therefore the orbit of $(row'(s_1), row'(s_2))$ is not in I. Moreover, $row'(s) \sqsubset row'(t)$ implies $row(s) \sqsubset row(t)$ (as no new rows are added), so no new pairs are added to I. Overall, I has less orbits (**C2**).
- If $row(s_1) = row(s_2)$, then we must have $row(s_1) = \pi \cdot row(s_1)$, for some π, because line 5 forbids equal rows in different orbits. In this case $row'(s_1) \neq \pi \cdot row'(s_1)$ and we can use part of the proof of Theorem 2 to see that the orbit of $row'(s_1)$ has bigger dimension or less local symmetries than that of $row(s_1)$ (**C3**).

Orbits of $(S \cup S{\cdot}A)/{\sim}$ and of I are finitely-many, by Lemma 12 and what we proved above. Moreover, local symmetries can decrease finitely-many times, and the dimension of each orbit of rows is bounded by the dimension of the minimal DFA state-space. Therefore all the above changes can happen finitely-many times.

We have proved that the table eventually becomes RFSA-closed and RFSA-consistent. Now we prove that a finite number of equivalence queries is needed to reach the final hypothesis automaton. To do this, we cannot use a suitable version of Lemma 4, because this relies on $N(S,E)$ being consistent with (S,E), which in general is not true (see [10] for an example of this). We can, however,

use an argument similar to that for RFSA-consistency, because the algorithm adds columns in response to counterexamples. Let w the counterexample provided by the teacher. When $12'$ is executed, the table must change. In fact, by [10, Lemma 2], if it does not, then w is already correctly classified by $N(S, E)$, which is absurd. We have the following cases. **E1)** orbits of $(S \cup S{\cdot}A)/{\sim}$ increase (**C1**). Or, **E2)** either: Orbits in $PR(S, E)$ increase, or any of the following happens: Orbits in I decrease (**C2**), local symmetries/ dimension of an orbit of rows change (**C3**). In fact, if **E1** does not happen and $PR(S, E)$, I and local symmetries/dimension of orbits of rows do not change, the automaton $\mathcal{A}$ for the new table coincides with $N(S, E)$. But $N(S, E) = \mathcal{A}$ is a contradiction, because $\mathcal{A}$ correctly classifies w (by [10, Lemma 2], as w now belongs to columns), whereas $N(S, E)$ does not. Both **E1** and **E2** can only happen finitely-many times. □

5.4 Query Complexity

We now give bounds for the number of equivalence and membership queries needed by $\nu\mathtt{NL}^\star$. Let n be the number of orbits of the minimal DFA M for the target language and let k be the dimension (i.e., the size of the maximum support) of its nominal set of states.

Lemma 13. *The number of equivalence queries $E'_{n,k}$ is $O(n^2 f_{\mathbb{A}}(k, k) + nk \log k)$.*

Proof. In the proof of Theorem 4, we saw that equivalence queries lead to more orbits in $(S \cup S{\cdot}A)/{\sim}$, in $PR(S, E)$, less orbits in I or less local symmetries/bigger dimension for an orbit. Clearly the first two ones can happen at most n times. We now estimate how many times I can decrease. Suppose $(S \cup S{\cdot}A)/{\sim}$ has d orbits and h orbits are added to it. Recall that, given an orbit O of rows of dimension at most m, $f_{\mathbb{A}}(m, m)$ is an upper bound for the number of orbits in the product $O \times O$. Since the support of rows is bounded by k, we can give a bound for the number of orbits added to I: $dhf_{\mathbb{A}}(k, k)$, for new pairs $r \sqsubset r'$ with r in a new orbit of rows and r' in an old one (or viceversa); plus $(h(h-1)/2)f_{\mathbb{A}}(k, k)$, for r and r' both in (distinct) new orbits; plus $hf_{\mathbb{A}}(k, k)$, for r and r' in the same new orbit. Notice that, if $PR(S, E)$ grows but $(S \cup S{\cdot}A)/{\sim}$ does not, I does not increase. By Lemma 12, $h, d \leq n$, so I cannot decrease more than $(n^2 + n(n-1)/2 + n)f_{\mathbb{A}}(k, k)$ times.

Local symmetries of an orbit of rows can decrease at most $k \log k$ times (see proof of Lemma 5), and its dimension can increase at most k times. Therefore $n(k + \log k)$ is a bound for all the orbits of rows, which are at most n, by Lemma 12. Summing up, we get the main result. □

Lemma 14. *Let m be the length of the longest counterexample given by the teacher. Then the table has:*

- *at most n orbits in S, with words of length at most n;*
- *at most $mE'_{n,k}$ orbits in E, with words of length at most $mE'_{n,k}$.*

Proof. By Lemma 12, the number of orbits of rows indexed by S is at most n. Now, notice that line 5 does not add $\mathtt{orb}(sa)$ to S if $sa \in S$, and lines 12 and 9 cannot identify rows, so S has at most n orbits. The length of the longest word in S must be at most n, as $S = \{\epsilon\}$ when the algorithm starts, and line $6'$ adds words with one additional symbol than those in S.

For columns, we note that both fixing RFSA-consistency and adding counterexamples increase the number of columns, but this can happen at most $E'_{n,k}$ times (see proof of Lemma 13). Each time at most m suffixes are added to E. □

We compute the maximum number of cells as in Section 4.3.

Lemma 15. *The number of orbits in the lower part of the table, $S{\cdot}A$, is bounded by $nlf_{\mathbb{A}}(pn, p)$.*

Then $C'_{n,k,m} = n(lf_{\mathbb{A}}(pn, p) + 1)mE'_{n,k}$ is the maximal number of cells in the table. This bound is polynomial in n, m and l, but not in k and p.

Corollary 2. *The number of membership queries is bounded by $C'_{n,k,m} f_{\mathbb{A}}(pn, pmE'_{n,k})$.*

6. Implementation and Preliminary Experiments

Our algorithms for learning nominal automata operate on infinite sets of rows and columns, and hence it is not immediately clear how to actually implement them on a computer. We have used NLambda [26], our recently developed Haskell library designed to allow direct manipulation of infinite (but orbit-finite) nominal sets, within the functional programming paradigm. The semantics of NLambda is based on [8], and the library itself is inspired by Fresh O'Caml [39], a language for functional programming over nominal data structures with binding.

6.1 NLambda

NLambda extends Haskell with a new type `Atoms`. Values of this type are atomic values that can be compared for equality and have no other discernible structure. They correspond to the elements of the infinite alphabet $\mathbb{A}$ described in Section 3.

Furthermore, NLambda provides a unary type constructor `Set`. This appears similar to the the `Data.Set` type constructor from the standard Haskell library, but its semantics is markedly different: Whereas the latter is used to construct finite sets, the former has *orbit-finite* sets as values. The new constructor `Set` can be applied to a range of equality types that include `Atoms`, but also the tuple type $(\mathtt{Atoms}, \mathtt{Atoms})$, the list type $[\mathtt{Atoms}]$, the set type `Set Atoms`, and other types that provide basic infrastructure necessary to speak of supports and orbits. All these are instances of a type class `NominalType` specified in NLambda for this purpose.

NLambda, in addition to all the standard machinery of Haskell, offers primitives to manipulate values of any nominal types τ, σ:

- $\mathtt{empty{:}Set}\ \tau$, returns the empty set of any type;
- $\mathtt{atoms{:}Set\ Atoms}$, returns the (infinite but single-orbit) set of all atoms;
- $\mathtt{insert} : \tau \to \mathtt{Set}\ \tau \to \mathtt{Set}\ \tau$, adds an element to a set;
- $\mathtt{map} : (\tau \to \sigma) \to (\mathtt{Set}\ \tau \to \mathtt{Set}\ \sigma)$, applies a function to every element of a set;
- $\mathtt{sum} : \mathtt{Set\ Set}\ \tau \to \mathtt{Set}\ \tau$, computes the union of a family of sets;
- $\mathtt{isEmpty} : \mathtt{Set}\ \tau \to \mathtt{Formula}$, checks whether a set is empty.

The type `Formula` takes the role of a Boolean type. For technical reasons it is distinct from the standard Haskell type `Bool`, but it provides standard logical operations such as

$$\mathtt{not} : \mathtt{Formula} \to \mathtt{Formula}$$
$$\mathtt{or} : \mathtt{Formula} \to \mathtt{Formula} \to \mathtt{Formula},$$

as well as a conditional operator $\mathtt{ite} : \mathtt{Formula} \to \tau \to \tau \to \tau$ that mimics the standard `if` construction. It is also the result type of a built-in equality test on atoms, $\mathtt{eq} : \mathtt{Atoms} \to \mathtt{Atoms} \to \mathtt{Formula}$.

Using these primitives, one builds more functions to operate on orbit-finite sets, such as a function to build singleton sets:

$$\mathtt{singleton} : \tau \to \mathtt{Set}\ \tau$$
$$\mathtt{singleton\ x} = \mathtt{insert\ x\ empty}$$

or a filtering function to select elements that satisfy a given predicate:

$$\mathtt{filter} : (\tau \to \mathtt{Formula}) \to \mathtt{Set}\ \tau \to \mathtt{Set}\ \tau$$
$$\mathtt{filter\ p\ s} = \mathtt{sum}\ (\mathtt{map}(\lambda \mathtt{x.ite}\ (\mathtt{p\ x})\ (\mathtt{singleton\ x})\ \mathtt{empty})\ \mathtt{s})$$

or functions to quantify a predicate over a set:

$$\texttt{exists}, \texttt{forall} : (\tau \to \texttt{Formula}) \to \texttt{Set}\ \tau \to \texttt{Formula}$$

$$\texttt{exists p s} = \texttt{not}\ (\texttt{isEmpty}\ (\texttt{filter p s}))$$

$$\texttt{forall p s} = \texttt{isEmpty}\ (\texttt{filter}\ (\lambda x.\texttt{not}\ (\texttt{p x}))\ \texttt{s})$$

and so on. Note that these functions are written in exactly the same way as they would be for finite sets and the standard `Data.Set` type. This is not an accident, and indeed the programmer can use the convenient set-theoretic intuition of NLambda primitives. For example, one could conveniently construct various orbit-finite sets such as the set of all pairs of atoms:

$$\texttt{atomPairs} = \texttt{sum}\ (\texttt{map}\ (\lambda \texttt{x.map}\ (\lambda \texttt{y.}(\texttt{x}, \texttt{y}))\ \texttt{atoms})\ \texttt{atoms}),$$

the set of all pairs of *distinct atoms*:

$$\texttt{distPairs} = \texttt{filter}\ (\lambda(\texttt{x}, \texttt{y}).\texttt{not}(\texttt{eq x y}))\ \texttt{atomPairs}$$

and so on.

It should be stressed that all these constructions terminate in finite time, even though they formally involve infinite sets. To achieve this, values of orbit-finite set types $\texttt{Set}\ \tau$ are internally not represented as lists or trees of elements of type τ. Instead, they are stored and manipulated symbolically, using first-order formulas over variables that range over atom values. For example, the value of `distPairs` above is stored as the formal expression:

$$\{(a, b) \mid a, b \in \mathbb{A},\ a \neq b\}$$

or, more specifically, as a triple:

- a pair (a, b) of "atom variables",
- a list $[a, b]$ of those atom variables that are bound in the expression (in this case, the expression contains no free variables),
- a formula $a \neq b$ over atom variables.

All the primitives listed above, such as `isEmpty`, `map` and `sum`, are implemented on this internal representation. In some cases, this involves checking the satisfiability of certain formulas over atoms. In the current implementation of NLambda, an external SMT solver Z3 [34] is used for that purpose. For example, to evaluate the expression `isEmpty distPairs`, NLambda makes a system call to the SMT solver to check whether the formula $a \neq b$ is satisfiable in the first-order theory of equality and, after receiving the affirmative answer, returns the value `False`.

For more details about the semantics and implementation of NLambda, see [26]. The library itself can be downloaded from [40].

6.2 Implementation of $\nu\texttt{L}^\star$ and $\nu\texttt{NL}^\star$

Using NLambda we implemented the algorithms from Sections 4 and 5. We note that the internal representation is slightly different than the one discussed in Section 4. Instead of representing the table (S, E) with actual representatives of orbits, the sets are represented logically as described above. Furthermore the control flow of the algorithm is adapted to fit in the functional programming paradigm. In particular, recursion is used instead of a while loop. In addition to the nominal adaptation of Angluin's algorithm $\nu\texttt{L}^\star$, we implemented a variant, $\nu\texttt{L}^\star_{col}$ which adds counterexamples to the columns instead of rows.

Target automata are defined using NLambda as well, using the automaton data type provided by the library. Membership queries are already implemented by the library. Equivalence queries are implemented by constructing a bisimulation (recall that bisimulation implies language equivalence), where a counterexample is obtained when two DFAs are not bisimilar. For nominal NFAs, however, we cannot implement a complete equivalence query as their language equivalence is undecidable. We approximated the equivalence by bounding the depth of the bisimulation for nominal NFAs. As an optimization, we use bisimulation up to congruence [13]. Having an approximate teacher is a minor issue since in many applications no complete teacher can be implemented and one relies on testing [2, 12]. For the experiments listed here the bound was chosen large enough for the learner to terminate with the correct automaton.

	DFA		$\nu\texttt{L}^\star$ (s)	$\nu\texttt{L}^\star_{col}$ (s)	RFSA		$\nu\texttt{NL}^\star$ (s)
$FIFO_0$	2	0	1.9	1.9	2	0	2.4
$FIFO_1$	3	1	12.9	7.4	3	1	17.3
$FIFO_2$	5	2	45.6	22.6	5	2	70.3
$FIFO_3$	10	3	189	107	10	3	476
$FIFO_4$	25	4	370	267	25	4	1230
$FIFO_5$	77	5	1337	697	∞	∞	∞
$\mathcal{L}_0$	2	0	1.3	1.4	2	0	1.4
$\mathcal{L}_1$	4	1	29.6	4.7	4	1	8.9
$\mathcal{L}_2$	7	2	229	23.1	7	2	84.7
$\mathcal{L}'_0$	3	1	4.4	4.9	3	1	11.3
$\mathcal{L}'_1$	5	1	15.4	15.4	4	1	66.4
$\mathcal{L}'_2$	9	1	46.3	40.5	5	1	210
$\mathcal{L}'_3$	17	1	89.0	66.8	6	1	566
$\mathcal{L}_{eq}$	n/a	n/a	n/a	n/a	3	1	16.3

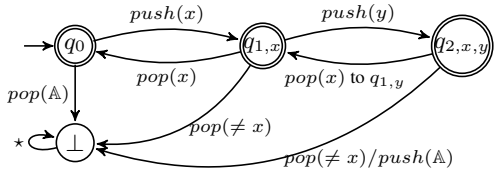
Table 1. Results of experiments. The column DFA (resp. RFSA) shows the number of orbits (left sub-column) and dimension (right sub-column) of the learnt minimal DFA (resp. canonical RFSA). We use ∞ when the running time is too high.

We remark that our algorithms seamlessly merge with teachers written in NLambda, but the current version of the library does not allow generating concrete membership queries for external teachers. We are currently working on a new version of the library in which this will be possible.

6.3 Test Cases

To provide a benchmark for future improvements, we tested our algorithms on a few simple automata described below. We report results in Table 1. The experiments were performed on a machine with an Intel Core i5 (Skylake, 2.4 GHz) and 8 GB RAM.

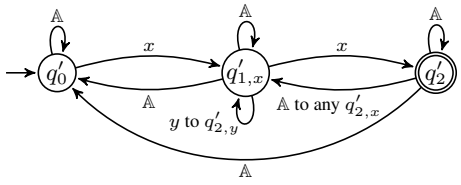
Queue Data Structure. A queue is a data structure to store elements which can later be retrieved in a first-in, first-out order. It has two operations: `push` and `pop`. We define the alphabet $\Sigma_{FIFO} = \{push(a), pop(a) \mid a \in \mathbb{A}\}$. The language $FIFO_n$ contains all valid traces of push and pop using a bounded queue of size n. The minimal nominal DFA for $FIFO_2$ is



The state reached from $q_{1,x}$ via $\xrightarrow{push(x)}$ is omitted: Its outgoing transitions are those of $q_{2,x,y}$, where y is replaced by x. Similar benchmarks appear in [2, 23].

Double Word. $\mathcal{L}_n = \{ww \mid w \in \mathbb{A}^n\}$ from Section 2.

NFA. Consider the language $\mathcal{L}_{eq} = \bigcup_{a \in \mathbb{A}} \mathbb{A}^\star a \mathbb{A}^\star a \mathbb{A}^\star$ of words where some letter appears twice. This is accepted by an NFA which guesses the position of the first occurrence of a repeated letter a and then waits for the second a to appear. The language is not accepted by a DFA [9]. Despite this $\nu\texttt{NL}^\star$ is able to learn the automaton:



where the transition from q'_2 to $q'_{1,x}$ is defined as $\delta(q'_2, a) = \{q'_{1,b} \mid b \in \mathbb{A}\}$.

n-last Position. A prototypical example of regular languages which are accepted by very small NFAs is the set of words where a distinguished symbol a appears on the n-last position [11]. We define a similar nominal language $\mathcal{L}'_n = \bigcup_{a\in\mathbb{A}} a\mathbb{A}^\star a\mathbb{A}^n$. To accept such words non-deterministically, one simply guesses the n-last position. This language is also accepted by a much larger deterministic automaton.

7. Related Work

This section compares $\nu\mathtt{L}^\star$ with other algorithms from the literature. We stress that no comparison is possible for $\nu\mathtt{NL}^\star$, as it is the first learning algorithm for non-deterministic automata over infinite alphabets.

The first one to consider learning automata over infinite alphabets was Sakamoto [38]. In his work the problem is reduced to $\mathtt{L}^\star$ with some finite sub-alphabet. The sub-alphabet grows in stages and $\mathtt{L}^\star$ is rerun at every stage, until the alphabet is big enough to capture the whole language. In Sakamoto's approach, any learning algorithm can be used as a back-end. This, however, comes at a cost: It has to be rerun at every stage, and each symbol is treated in isolation, which might require more queries. Our algorithm $\nu\mathtt{L}^\star$, instead, works with the whole alphabet from the very start, and it exploits its symmetry. An example is in Sections 2.1 and 2.2: The ordinary learner uses four equivalence queries, whereas the nominal one, using the symmetry, only needs three. Moreover, our algorithm is easier to generalize to other alphabets and computational models, such as non-determinism.

More recently papers appeared on learning register automata [15, 21]. Their register automata are as expressive as our deterministic nominal automata. The state-space is similar to our orbit-wise representation: It is formed by finitely many locations with registers. Transitions are defined symbolically using propositional logic. We remark that the most recent paper [15] generalizes the algorithm to alphabets with different structures (which correspond to different atom symmetries in our work), but at the cost of changing Angluin's framework. Instead of membership queries the algorithm requires more sophisticated tree queries. In our approach, using a different symmetry does not affect neither the algorithm nor its correctness proof. Tree queries can be reduced to membership queries by enumerating all n-types for some n (n-types in logic correspond to orbits in the set of n-tuples). Keeping that in mind, their complexity results are roughly the same as ours, although this is hard to verify, as they do not give bounds on the length of individual tree queries. Finally, our approach lends itself better to be extended to other variations on $\mathtt{L}^\star$ (of which many exist), as it is closer to Angluin's original work.

Another class of learning algorithms for systems with large alphabets is based on abstraction and refinement, which is orthogonal to the approach in the present paper but connections and possible transference of techniques are worth exploring in the future. In [2], the alphabet is reduced to a finite alphabet of abstractions, and $\mathtt{L}^\star$ for ordinary DFAs over such finite alphabet is used. Abstractions are refined by counterexamples. Other similar approaches are [20, 22], where global and local per-state abstractions of the alphabet are used, and [30, 32], where the alphabet can also have additional structure (e.g., an ordering relation). We can also mention [14], a framework for learning symbolic models of software behavior.

In [5, 6], authors cope with an infinite alphabet by running $\mathtt{L}^\star$ (adapted to Mealy machines) using a finite approximation of the alphabet, which may be augmented when equivalence queries are answered. A smaller symbolic model is derived subsequently. Their approach, unlike ours, does not exploit the symmetry over the full alphabet. The symmetry allows our algorithm to reduce queries and to produce the smallest possible automaton at every step.

Finally we compare with results on session automata [12]. Session automata are defined over finite alphabets just like the work by Sakamoto. However, session automata are more restrictive than deterministic nominal automata. For example, the model cannot capture an acceptor for the language of words where consecutive data values are distinct. This language can be accepted by a three orbit nominal DFA, which can be learned by our algorithm.

We implemented our algorithms in the nominal library NLambda as sketched before. Other implementation options include Fresh O'Caml [39], a functional programming language designed for programming over nominal data structures with binding, and LOIS [27, 28], a C++ library for imperative nominal programming. We chose NLambda for its convenient set-theoretic primitives, but the other options remain to be explored, in particular the low-level LOIS could be expected to provide more efficient implementations.

8. Discussion and Future Work

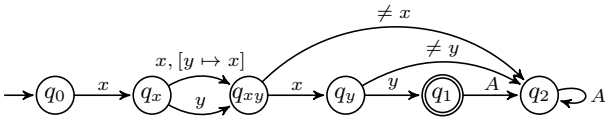
In this paper we defined and implemented extensions of several versions of $\mathtt{L}^\star$ and of $\mathtt{NL}^\star$ for nominal automata.

We highlight two features of our approach:

- it has strong theoretical foundations: The *theory of nominal languages*, covering different alphabets and symmetries (see Section 3.1); *category theory*, where nominal automata have been characterized as *coalgebras* [16, 29] and many properties and algorithms (e.g., minimization) have been studied at this abstract level.
- it follows a generic pattern for transporting computation models and algorithms from finite sets to nominal sets, which leads to simple correctness proofs.

These features pave the way to several extensions and improvements.

Future work includes a general version of $\nu\mathtt{NL}^\star$, parametric in the notion of side-effect (an example is non-determinism). Different notions will yield models with different degree of succinctness w.r.t. deterministic automata. The key observation here is that many forms of non-determinism and other side effects can be captured via the categorical notion of *monad*, i.e., an algebraic structure, on the state-space. Monads allow generalizing the notion of composed and prime state: A state is composed whenever it is obtained from other states via an algebraic operation. Our algorithm $\nu\mathtt{NL}^\star$ is based on the *powerset* monad, representing classical non-determinism. We are currently investigating a *substitution* monad, where the operation is "applying a (possibly non-injective) substitution of atoms in the support". A minimal automaton over this monad, akin to a RFSA, will have states that can generate all the states of the associated minimal DFA via a substitution, but cannot be generated by other states (they are prime). For instance, we can give an automaton over the substitution monad that recognizes $\mathcal{L}_2$ from Section 2:



Here $[y \mapsto x]$ means that, if that transition is taken, q_{xy} (hence its language) is subject to $y \mapsto x$. In general, the size of the minimal DFA for $\mathcal{L}_n$ grows more than exponentially with n, but an automaton with substitutions on transitions, like the one above, only needs $O(n)$ states.

In principle, thanks to the generic approach we have taken, all our algorithms should work for various kinds of atoms with more structure than just equality, as advocated in [9]. Details, such as precise assumptions on the underlying structure of atoms necessary for proofs to go through, remain to be checked. For an implementation

of automata learning over other kinds of atoms without compromising the generic approach, an extension of NLambda to those atoms will be needed, as the current version of the library only supports equality and totally ordered atoms.

The efficiency of our current implementation, as measured in Section 6.3, leaves much to be desired. There is plenty of potential for running time optimization, ranging from improvements in the learning algorithms itself, to optimizations in the NLambda library (such as replacing the external and general-purpose SMT solver with a purpose-built, internal one, or a tighter integration of nominal mechanisms with the underlying Haskell language as it was done in [39]), to giving up the functional programming paradigm for an imperative language such as LOIS [27, 28].

Acknowledgements

We thank Frits Vaandrager and Gerco van Heerdt for useful comments and discussions. We also thank the anonymous reviewers.

References

[1] Fides Aarts and Frits W. Vaandrager. Learning I/O automata. In *CONCUR*, pages 71–85, 2010. doi: 10.1007/978-3-642-15375-4_6.

[2] Fides Aarts, Paul Fiterau-Brostean, Harco Kuppens, and Frits W. Vaandrager. Learning register automata with fresh value generation. In *ICTAC*, pages 165–183, 2015. doi: 10.1007/978-3-319-25150-9_11.

[3] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. doi: 10.1016/0890-5401(87)90052-6.

[4] Dana Angluin and Miklós Csürös. Learning markov chains with variable memory length from noisy output. In *COLT*, pages 298–308, 1997. doi: 10.1145/267460.267517.

[5] Therese Berg, Bengt Jonsson, and Harald Raffelt. Regular inference for state machines with parameters. In *FASE*, pages 107–121, 2006. doi: 10.1007/11693017_10.

[6] Therese Berg, Bengt Jonsson, and Harald Raffelt. Regular inference for state machines using domains with equality tests. In *FASE*, pages 317–331, 2008. doi: 10.1007/978-3-540-78743-3_24.

[7] Mikołaj Bojańczyk and Sławomir Lasota. A machine-independent characterization of timed languages. In *ICALP*, pages 92–103, 2012. doi: 10.1007/978-3-642-31585-5_12.

[8] Mikołaj Bojańczyk, Laurent Braud, Bartek Klin, and Sławomir Lasota. Towards nominal computation. In *POPL*, pages 401–412, 2012.

[9] Mikołaj Bojańczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *LMCS*, 10(3), 2014. doi: 10.2168/LMCS-10(3:4)2014.

[10] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of nfa. Technical Report LSV-08-28, ENS de Cachan, 2008.

[11] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *IJCAI*, pages 1004–1009, 2009.

[12] Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A fresh approach to learning register automata. In *DLT*, pages 118–130, 2013. doi: 10.1007/978-3-642-38771-5_12.

[13] Filippo Bonchi and Damien Pous. Hacking nondeterminism with induction and coinduction. *Commun. ACM*, 58(2):87–95, 2015. doi: 10.1145/2713167.

[14] Matko Botincan and Domagoj Babic. Sigma*: symbolic learning of input-output specifications. In *POPL*, pages 443–456, 2013. doi: 10.1145/2429069.2429123.

[15] Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Active learning for extended finite state machines. *Formal Asp. Comput.*, 28(2):233–263, 2016. doi: 10.1007/s00165-016-0355-5.

[16] Vincenzo Ciancia and Ugo Montanari. Symmetries, local names and dynamic (de)-allocation of names. *Inf. Comput.*, 208(12):1349–1367, 2010. doi: 10.1016/j.ic.2009.10.007.

[17] Loris D'Antoni and Margus Veanes. Minimization of symbolic automata. In *POPL*, pages 541–554, 2014. doi: 10.1145/2535838.2535849.

[18] Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009. doi: 10.1145/1507244.1507246.

[19] François Denis, Aurélien Lemay, and Alain Terlutte. Residual finite state automata. *Fundam. Inform.*, 51(4):339–368, 2002.

[20] Falk Howar, Bernhard Steffen, and Maik Merten. Automata learning with automated alphabet abstraction refinement. In *VMCAI*, pages 263–277, 2011. doi: 10.1007/978-3-642-18275-4_19.

[21] Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring canonical register automata. In *VMCAI*, pages 251–266, 2012. doi: 10.1007/978-3-642-27940-9_17.

[22] Malte Isberner, Falk Howar, and Bernhard Steffen. Inferring automata with state-local alphabet abstractions. In *NFM*, pages 124–138, 2013. doi: 10.1007/978-3-642-38088-4_9.

[23] Malte Isberner, Falk Howar, and Bernhard Steffen. Learning register automata: from languages to program structures. *Machine Learning*, 96(1-2):65–98, 2014. doi: 10.1007/s10994-013-5419-7.

[24] Bart Jacobs and Alexandra Silva. Automata learning: A categorical perspective. In *Horizons of the Mind*, pages 384–406, 2014. doi: 10.1007/978-3-319-06880-0_20.

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

[26] Bartek Klin and Michał Szynwelski. SMT solving for functional programming over infinite structures. In *MFSP*, volume 207, pages 57–75, 2016. doi: 10.4204/EPTCS.207.3.

[27] Eryk Kopczyński and Szymon Toruńczyk. LOIS: an application of SMT solvers. In *SMT*, volume 1617, pages 51–60, 2016.

[28] Eryk Kopczyński and Szymon Toruńczyk. LOIS: syntax and semantics. In *POPL*, 2017. This volume.

[29] Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan, and Alexandra Silva. Nominal kleene coalgebra. In *ICALP*, pages 286–298, 2015. doi: 10.1007/978-3-662-47666-6_23.

[30] Oded Maler and Irini-Eleftheria Mens. Learning regular languages over large alphabets. In *TACAS*, pages 485–499, 2014. doi: 10.1007/978-3-642-54862-8_41.

[31] Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Inf. Comput.*, 118(2):316–326, 1995. doi: 10.1006/inco.1995.1070.

[32] Irini-Eleftheria Mens and Oded Maler. Learning regular languages over large ordered alphabets. *LMCS*, 11(3), 2015. doi: 10.2168/LMCS-11(3:13)2015.

[33] Ugo Montanari and Matteo Sammartino. A network-conscious π-calculus and its coalgebraic semantics. *Theor. Comput. Sci.*, 546:188–224, 2014. doi: 10.1016/j.tcs.2014.03.009.

[34] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *TACAS*, pages 337–340, 2008. doi: 10.1007/978-3-540-78800-3_24.

[35] Oliver Niese. *An integrated approach to testing complex systems*. PhD thesis, Universität Dortmund, 2003.

[36] Andrew M Pitts. *Nominal sets: Names and symmetry in computer science*. Cambridge University Press, 2013.

[37] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993. doi: 10.1006/inco.1993.1021.

[38] Hiroshi Sakamoto. Learning simple deterministic finite-memory automata. In *ALT*, pages 416–431, 1997. doi: 10.1007/3-540-63577-7_58.

[39] Mark R. Shinwell. Fresh O'Caml: Nominal abstract syntax for the masses. *ENTCS*, 148(2):53–77, 2006. doi: 10.1016/j.entcs.2005.11.040.

[40] Michał Szynwelski. Nλ. Available from `http://www.mimuw.edu.pl/~szynwelski/nlambda/`.