On Learning Nominal Automata with Binders

Yi Xiao
Department of Informatics
University of Leicester (UK)
yx104@leicester.ac.uk

Emilio Tuosto
Gran Sasso Science Institute (IT) and
Department of Informatics, University of Leicester (UK)
emilio.tuosto@gssi.it

We investigate a learning algorithm in the context of *nominal automata*, an extension of classical automata to alphabets featuring names. This class of automata captures *nominal regular languages*; analogously to the classical language theory, nominal automata have been shown to characterise *nominal regular expressions with binders*. These formalisms are amenable to abstract modelling resource-aware computations.

We propose a learning algorithm on nominal regular languages with binders. Our algorithm generalises Angluin's $L^\star$ algorithm with respect to nominal regular languages with binders. We show the correctness and study the theoretical complexity of our algorithm.

1 Introduction

This paper combines *nominal languages* and *learning automata* to abstractly model computations connected with *resource awareness*. Here, we do not restrict ourselves to a specific type of resources; rather we think of resources in a very abstract and general sense. We use *names* as models of resources and (abstract) operations on names as developed in *nominal* languages (see Section 2 for an overview) as mechanisms to capture basic properties of resources; in particular we focus on the *dynamic allocation and deallocation* of resources. More precisely, we take inspiration from binders with dynamic scoping of nominal languages in an operational context based on finite state *nominal* automata. The states of these automata have transitions to explicitly (i) allocate names, corresponding to *scope extrusion* of nominal languages, and (ii) to deallocate names corresponding to *garbage collection* of (unused) names. Our theory sets in the context of *nominal regular expressions* that transfer the traditional Kleene theorem to the nominal framework adopted here. In fact, the class of nominal languages that we consider can be characterised as those accepted by nominal automata or, equivalently, that can be generated by *nominal regular expressions*. The latter algebraic presentation (that we borrow from the literature and review in Section 2) features, besides the usual operations of regular expressions (union, concatenation, Kleene-star), a name binding mechanism and a special *resource-aware* complementation operation. Our results rely on the closure properties of these class of languages that has been already demonstrated in the literature.

In this context, we develop a learning algorithm for nominal automata. We take inspiration from the $\mathsf{L}^\star$ algorithm of Angluin (also reviewed in Section 2). As we will see, the design of the algorithm requires some ingenuity and opens up the possibility of interesting investigations due to richer structure brought in by names and name binding.

Nominal languages and learning The pioneering work on languages on infinite alphabet is [21]. And, the theory of nominal languages has been advocated as a suitable abstraction for computations *with resources* emerging from the so-called nominal calculi which bred after the seminal work introducing the π-calculus [29, 28, 35]. Abstract theories capturing the computational phenomena in this context have been developed in [16, 17, 15] in parallel with a theory of nominal automata [30, 14, 33]. The formal

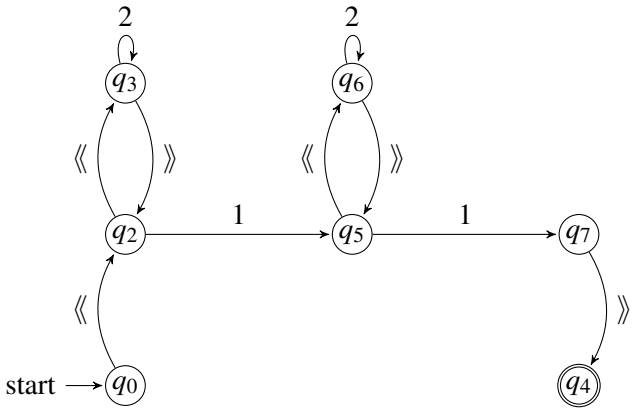
M. Bartoletti, L. Henrio, A. Mavridou, A. Scalas (Eds.):
12th Interaction and Concurrency Experience (ICE 2019).
EPTCS 304, 2019, pp. 137–155, doi:10.4204/EPTCS.304.9

connections between these theories have been unveiled in [18]. Later, [25] proposed the notion of nominal regular languages and the use of nominal automata as acceptors of such languages. As observed in [25, 4] are not suitable to handle name binding as registers are 'global'; the nominal model in [4] is instead closer (see also the comment below Example 11.4 of [4]) to history dependent automata [33], which are also the inspiration for the model of automata in [25]. The nominal automata in [4] are (abstractions of) deterministic HD-automata (which can be seen as 'implementation' of orbit-finite nominal automata following the connection between nominal and named set of [18]). This class of automata is more expressive than the classes of automata capturing nominal regular expressions as ours or nominal Kleene algebras [24]. In fact, as noted in [24] this automata accept languages with words having arbitrarely deep nesting of binders. However, orbit-finite nominal automata are not closed under any reasonable notion of complementation [4]. Note that the resource-sensitive complementation operation of [25] is essential in our context. On the other hand, the use of symmetries to capture binding offers a more flexible mechanism to express patterns or words that escape the constraints that the use of 'nested scoping' imposes in our language.

A main motivation for this proposal is the abstract characterisation of basic features of computations *with resources*. For instance, nominal automata have applications to the verification of protocols and systems [12, 13]. Other approaches to verify resource-aware computations have also been based on automata models [3, 10, 11] employ usage automata (UA) to express and model check patterns of resource-usage. A distinguishing feature of the approach in [25, 26] is that allocation and deallocation of resources is abstracted away with *binders*. Inspired by the *scope extrusion* mechanism of the π-calculus, the allocation of a resource corresponds to an (explicit) operation that introduces a fresh name; likewise, the deallocation of a resource corresponds to an (explicit) operation to "free" names. We illustrate this idea with an example. Consider the following expression

$$\hat{E} = \langle n.\langle m.m\rangle^{\star} n\langle k.k^{\star}\rangle n\rangle$$

which is a nominal regular expression where n,m,k are names, $\_^{\star}$ is the usual Kleene-star operation, and subexpressions of the form $\langle n.E\rangle$ represent the binding mechanism whereby name n is bound (that is "local") to expression E. Intuitively, $\hat{E}$ describes a language of words starting with the allocation of a freshly generated name, conventionally denoted n, followed by the words generated by the subexpression $\langle m.m\rangle^{\star}$ post-fixed by n, and so on. Note that in $\hat{E}$ name m occurs in a nested binder for name n. According to [25], $\hat{E}$ corresponds to the following nominal automaton:



which from the initial state q_0 allocates a fresh name through the transition labelled $\langle\!\langle$ to state q_2. Notice how bound names are rendered in the nominal automaton: they are concretely represented as (strictly

positive) natural numbers. This allows us to abstract away from the identities of bound names. In fact, the identity of bound names is immaterial and can be *alpha-converted*, that is replace with any other name provided that the name has not been used already. The use of numbers enables a simple "implementation" of alpha-conversion. More precisely, think of numbers as being addresses of registers of states. For instance, q_3 has 2 registers addressed by 1 and 2 respectively. Then the self-loop transition in state q_3 can consume any name n, provided that n is different than the name (currently) stored in register 1. Finally, note that the content of registers is local to states; once a deallocation transition $\rangle\!\rangle$ is fired, the content in last allocated register is disregarded.

For a practical example, we consider a scenario based on servers to show how nominal regular languages with binders can suitably specify usage policies of servers $S_1,\ldots,S_k$, such that, $\forall 1 \leq h \leq k$ S_h offers operations $\{o_{h_1},\ldots,o_{h_k}\} = O_h$. Given an alphabet

$$\Sigma = \bigcup_{h=1}^{k} \{l_{i_h}, l_{o_h}\} \cup O_h \qquad \text{where symbols } l_{i_h}, l_{o_h} \text{ represent basic input and output activities}$$

$$\text{consider the regular expressions on } \Sigma \qquad E_h = (l_{i_h}\langle s.e_h l_{o_h}\rangle)^* \qquad e_h = (\sum_{o\in O_h} o.e_o)^*$$

Name s is a fresh session identifier which the symbol $\langle$ allocates when the session starts and the symbol $\rangle$ deallocates when the session ends. Note that s may occur in e_h to e.g., avoid re-authentication. Resources (activities, sessions, operations, etc.) can be abstracted as letters and names, and the (de)allocation represent the binding and freshness conditions. Intuitively, we give the following nominal regular expressions with $\Sigma = \bigcup_{h=1}^{k}\{l_{i_h}, l_{o_h}\} \cup \{\text{readFead}, \text{updateProfile}\}$ and $n_1 \neq n_2$ be distinct names. Operations readFeed, updateProfile allow users read a feed and to update a profile.

$$E_1 = (l_{i_1}\langle n_1.e_1 l_{o_1}\rangle)^* \qquad e_1 = (n\ \text{readFeed} + \text{updateProfile}\ E_2)^* \qquad E_2 = (l_{i_2}\langle n_2.e_2 l_{o_2}\rangle)^*$$

From the above equations, we see clearly the binders delimit the scope of n_1 and n_2. And, n_2 is nested in n_1. Intuitively, the approach above relaxes the condition of classical language theory that the alphabet of a language is constant. Binder allow us to extend the alphabet "dynamically". This is strongly related to other approaches in the literature, where languages over infinite alphabets are considered. A form of regular expressions, called UB-expressions, for languages on infinite alphabets investigated in [22]. In [37] pebble automata are compared to register automata. This class of languages are not suitable for our purposes as they do not account for freshness.

Alternative approaches investigating languages over infinite alphabets are those in [4, 36]. A finite representation of nominal sets and automata with data symmetries and permutations is given in [4]; this presentation differs from the one in [26]. Using the equivalence in [18], the nominal regular expressions with binders of [26] can be transferred into the context as the style of [4]. In fact, permutations permit to encode name binding and give an implicit representation of name scoping in nominal words. On the other hand, [36] defines *bar strings* and considers another representation of nominal sets and automata with binders, regular expressions, and non-deterministic nominal automata over them. An example of bar string with names a, b, and c is $ab|ccb$ that represents a word where name c following a is bound in the rest of the string. A key observation is that in [4] the scope of binders models load freshness and it is fixed: once stated, the scope extends as far as possible "to the right". In our setting, this would account to allocate a resource and never deallocate, that is, $|b$ in [4] corresponds to the notation $\langle\!\langle b.b$ of [26] where the angled bracket opens the scope of the binder restricting the occurrences of name b after the dot symbol; in this notation $\rangle\!\rangle$ are used to close the scope opened by $\langle\!\langle$. To illustrate the differences from [26], we consider

the example under local freshness semantics [4, p. 5]. Let $\mathbb{A}$ be a set of names, and a, b, and c be names in $\mathbb{A}$, $\{|a|b, |a|a\}$ is alpha-equivalent to $\{|a|a\}$, unlike in [26] where is $\{\langle\!\langle a.a\rangle\!\rangle\langle\!\langle b.b\rangle\!\rangle, \langle\!\langle a.a\rangle\!\rangle\langle\!\langle a.a\rangle\!\rangle\}$ is alpha-equivalent to $\{\langle\!\langle a.a\rangle\!\rangle\langle\!\langle b.b\rangle\!\rangle\}$ while $\{\langle\!\langle a.a\langle\!\langle b.b\rangle\!\rangle\rangle\!\rangle, \langle\!\langle a.a\langle\!\langle a.a\rangle\!\rangle\rangle\!\rangle\}$ is alpha-equivalent to $\{\langle\!\langle a.a\langle\!\langle b.b\rangle\!\rangle\rangle\!\rangle\}$. Thus, without the closing scope, the two kinds of context are not easily transferred to each other. Bar string $|a|ba$ could be corresponding to $\langle\!\langle a.a\langle\!\langle b.b\rangle\!\rangle a\rangle\!\rangle$ or $\langle\!\langle a.a\langle\!\langle b.ba\rangle\!\rangle\rangle\!\rangle$ under open conditions. However, if $|a|ba$ is a word in the language $\{cdc \in \mathbb{A}^3 \mid c \neq d\}$, $|a|ba$ is corresponding to $\langle\!\langle a.a\langle\!\langle b.b\rangle\!\rangle a\rangle\!\rangle$.

One of the most known and used learning algorithm is $\mathsf{L}^\star$ introduced by Angluin [1]. As surveyed in Section 2, given a regular language, $\mathsf{L}^\star$ creates a deterministic automaton that accepts the language. This is done by mimicking the "dialogue" between a *learner* and a *teacher*; the former poses questions about the language to the latter. In $\mathsf{L}^\star$ there are two types of queries the learner can ask the teacher: *membership* queries allow the learner to check whether a word belongs to the input language while with *equivalence* queries the learner checks if an automaton accepts or not the language. The automaton is "guessed" by the learner according to the answers the teacher provides to queries. The outcome of an equivalence query may be a *counterexample* selected by the teacher to exhibit that the automaton does not accept the language.

The $\mathsf{L}^\star$ algorithm has been extended to several classes of languages [6, 31, 32]. Using a categorical approach, the $\mathsf{L}^\star$ algorithm has been generalised to other classes of automata such as Moore and Mealy [20]. An interesting line of research is the one explored in [6] which applies learning automata to distributed systems based on message-passing communications to learn communicating finite-state machines [7] from message-sequence charts. Applications of learning automata are in [32] and [31]. The former defines a framework based on $\mathsf{L}^\star$ to fully automatise an incremental assume-guarantee verification technique and the latter proposes an optimised approach for integrated testing of complex systems.

Variants of Angluin's algorithm for languages over infinite alphabets have attracted researchers' attention. An $\mathsf{L}^\star$ algoritm for register automata is given in [5] where so-called *session automata* support the notion of fresh data values. Session automata are defined over pairs of finite-infinite alphabets. Interestingly, session automata also have a canonical form to decide equivalence queries.

Like [5], [8] works on register automata and data language but the latter aims to the application of dynamic black-box analysis. The key point is that [8] uses a tree queries instead of membership queries and ensures the observation tables closeness and register-consistency. Further, [8] defines a new version of equivalence to achieve the correctness and termination of the algorithm.

Recently, [9] proposes a learning algorithm which extends $\mathsf{L}^\star$ to nominal automata. The main difference between our approach and the one in [9] is the representation of nominal languages and nominal automata. Language theories for infinite alphabets [4, 34] are used in [9], handling names through finitely-supported permutations. Accordingly, in [9] observation tables and the states of nominal automata are orbit-finite. Another difference is the operation on counterexamples. Unlike in $\mathsf{L}^\star$, [9] adds the counterexamples into columns. It is an interesting research direction to optimise our work on the operations of the counterexamples in the future.

Main contributions Our main objective is to develop a learning algorithm for nominal languages, with a focus on resources. The baseline to tackle this objective is the use of binders and languages on infinite alphabets. Section 3 collects our main results.

Our first achievement is the design of a learning algorithm that generalises Angluin's $\mathsf{L}^\star$ algorithm to nominal regular languages with binders. We call our algorithm $\mathsf{nL}^\star$ (after *nominal* $\mathsf{L}^\star$), as a tribute to Angluin's work. This is attained by retaining the basic scheme of $\mathsf{L}^\star$ (query / response dialogue been a learner and a teacher) and ideas of $\mathsf{L}^\star$ (the represenation of a finite state automaton with specific

a *observation table*). Technically, this requires a revision of the main concepts of Angluin's theory. In particular, the type of queries and answers now have to account for names and the allocation and deallocation operations on them. Consequently, we have to reconsider the data structure to represent observation tables and hence the notions of *closedness* and *consistency*.

Interestingly, this revision culminates in Theorem 4 and Theorem 5 respectively showing how the learned nominal automata associated behave and the correctness of $\mathsf{nL}^\star$. Finally, we discuss the complexity of $\mathsf{nL}^\star$.

2 Background

We survey the principal concepts need in the rest of the paper. In particular, we review basics of formal language theory, its nominal counterpart, and the learning algorithm of Angluin's $\mathsf{L}^\star$ [1].

2.1 Regular Languages

Regular Languages, regular expressions and finite automata have a well-known relationship established by the Kleene theorem [23]. A regular language can be represented by regular expressions and accepted by a finite state automaton. In this section, we introduce necessary notions and definitions for these concepts.

An *alphabet* is a set (whose elements are often called *letters* or *symbols*). We denote a finite alphabet as Σ. A *word* is a sequence of symbols of an alphabet. Let w be a word, we denote the length of w as $|w|$. The word of length zero is called *empty word* and denoted by ε. The concatenation of two words is denoted as $\_\cdot\_$. We define $\Sigma^\star = \bigcup_{n=0}^{\infty} \Sigma^n$, where $\Sigma^0 = \{\varepsilon\}$ and for each $n > 0$, $\Sigma^n = \{w \cdot w' \mid w \in \Sigma \text{ and } w' \in \Sigma^{n-1}\}$. A *language* L is a set of words over an alphabet Σ, that is, $L \subseteq \Sigma^\star$.

Other language operations we use are concatenation, union, *Kleene-star* and complementation. Assuming that L *and* L' are languages over Σ, we have the following standard definitions:

- concatenation $L \cdot L' = \{w \cdot w' \mid w \in L \text{ and } w' \in L'\}$,
- union $L \cup L' = \{w \mid w \in L \text{ or } w \in L'\}$,
- *Kleene-star* $L^\star = \bigcup_{n=0}^{\infty} L^n = \begin{cases} \{\varepsilon\} & n = 0 \\ L^{n-1} \cdot L & n \neq 0 \end{cases}$,
- complementation $L^C = \{w \in \Sigma^\star \mid w \notin L\}$.

A *regular expressions over* Σ is a term derived from the grammar where $a \in \Sigma$:

$$re ::= \varepsilon \mid \emptyset \mid a \mid re + re \mid re \cdot re \mid re^\star$$

(where operators are listed in inverse order of precedence). Given two regular expressions re and re', $re + re'$ denotes the union of re and re', $re \cdot re'$ denotes the concatenation of re and re', and $re^\star$ denotes the Kleene-star of re.

A *finite automaton over alphabet* Σ is a five-tuple $M = \langle Q, q_0, F, \delta \rangle$ such that Q is a finite set of states, q_0 is the initial state, $F \subseteq Q$ is the finite set of final states, $\delta \subseteq Q \times \Sigma \times Q$ is a relation from states and alphabet symbols to states. The automaton M is *deterministic* when δ is a function on $Q \times \Sigma \to Q$. We extend the transition relation δ to $\Sigma^\star$ in the obvious way, define the language of an automaton M as usual, and denote it as $\mathfrak{L}(M)$.

It is well-known that regular expressions denote regular languages; we let $\mathscr{L}(re)$ denote the language of a regular expression re.

Theorem 1 ([19])**.** *A language L is regular iff there exists a finite automaton M such that $L = \mathfrak{L}(M)$. Moreover, there exists a minimal finite automaton M accepting L and M is unique.*

2.2 Nominal Languages

We use the nominal regular expressions introduced in [26, 27]. Languages over infinite alphabets are generalised to nominal automata and nominal expressions [4, 26, 27, 36, 37]. The approach in [26] is distinguished by the use of names and binders in the expressions. In the following, we recall the basic notions first and then we survey nominal languages [26]. Hereafter, we fix a countably infinite set of names $\mathscr{N}$.

A *nominal language over* $\mathscr{N}$ *and* Σ is a set of nominal words w over $\mathscr{N}$ and Σ, that is terms derived by the grammar

$$w ::= \varepsilon \mid a \mid \mathsf{n} \mid w \cdot w \mid \langle\!\langle \mathsf{n}.w \rangle\!\rangle \qquad \text{where } \mathsf{n} \in \mathscr{N} \text{ and } a \in \Sigma$$

A name is *bound* in a word when it occurs in the scope of a binder. Occurrences of names not bound are called *free*. For example, n is bound in word $\langle\!\langle \mathsf{n}.\mathsf{n}\, a \rangle\!\rangle\, \mathsf{m}$ while m is free.

A *nominal regular expression* is a term derived from the grammar

$$ne ::= \varepsilon \mid \emptyset \mid a \mid \mathsf{n} \mid ne + ne \mid ne \cdot ne \mid ne^{\star} \mid \langle \mathsf{n}.ne \rangle \qquad \text{where } \mathsf{n} \in \mathscr{N} \text{ and } a \in \Sigma$$

In nominal expressions, binders are represented as $\langle \mathsf{n}._{-} \rangle$ for $\mathsf{n} \in \mathscr{N}$. If the names in a nominal expression are all bound, the nominal expression is *closed*. Nominal regular expressions denote *nominal languages*.

Definition 1. *[26] The* nominal language $\mathscr{L}(ne)$ of a nominal regular expression *ne is defined as*

- $\mathscr{L}(\varepsilon) = \{\varepsilon\} \qquad \mathscr{L}(\emptyset) = \emptyset \qquad \mathscr{L}(a) = \{a\} \qquad \mathscr{L}(\mathsf{n}) = \{\mathsf{n}\}$
- $\mathscr{L}(ne_1 + ne_2) = \mathscr{L}(ne_1) \cup \mathscr{L}(ne_2)$
- $\mathscr{L}(ne_1 \cdot ne_2) = \mathscr{L}(ne_1) \cdot \mathscr{L}(ne_2) = \{w \cdot v \mid w \in \mathscr{L}(ne_1), v \in \mathscr{L}(ne_2)\}$
- $\mathscr{L}(ne^{\star}) = \bigcup_{k \in \mathbb{N}} \mathscr{L}(ne)^k$, *where* $\mathscr{L}(ne)^k = \begin{cases} \{\varepsilon\} & k = 0 \\ \mathscr{L}(ne) \cdot \mathscr{L}(ne)^{k-1} & k \neq 0 \end{cases}$
- $\mathscr{L}(\langle \mathsf{n}.ne \rangle) = \{\langle\!\langle \mathsf{n}.w \rangle\!\rangle \mid w \in \mathscr{L}(ne)\}$.

The closure properties of nomianl regular languages are stated below:

Theorem 2. *[26] Nominal regular languages are closed under union, intersection, and* resource sensitive *complementation.*

The main difference with respect to classical regular expressions is on complementation. Since, the complement of $\mathscr{L}(ne)$ is not a nominal regular language, the classical complementation does not work in the nominal case. Therefore, [26] give the following definition:

Definition 2. *[26] Let ne be a nominal regular expression. The* resource sensitive complement of $\mathscr{L}(ne)$ *is the set* $\{w \notin \mathscr{L}(ne) \mid \theta(w) \leq \theta(ne)\}$ *where*

- $ne \in \{\varepsilon, \emptyset\} \cup \mathscr{N} \cup \Sigma \implies \theta(ne) = 0$
- $ne = ne_1 + ne_2$ *or* $ne_1 \cdot ne_2 \implies \theta(ne) = max(\theta(ne_1), \theta(ne_2))$
- $ne = \langle \mathsf{n}.ne \rangle \implies 1 + \theta(ne)$
- $ne = ne^{\star} \implies \theta(ne)$.

and the depth θ *of a word is defined as the depth the corresponding expression.*

We now define the notions of nominal automata adopted here. Let $\mathbb{N}$ be the set of natural numbers and define $\underline{n} = \{1, \cdots, n\}$ for each $n \in \mathbb{N}$. Considering a set of states Q paired with a map $\|\_\| : Q \to \mathbb{N}$, let us define the local registers of $q \in Q$ to be $\underline{\|q\|}$. We use a definition of nominal automata [25] as Definition 3. Moreover, we describe how to allocate names via maps $\sigma : \underline{\|q\|} \to \mathscr{N}$.

Definition 3 (Nominal Automata [25]). *Let $\mathcal{N}_{fin} \subset \mathcal{N}$ be a finite set of names. A* nominal automaton with binders over Σ and $\mathcal{N}_{fin}$, $(\Sigma, \mathcal{N}_{fin})$*-automaton for short, is a tuple $M = \langle Q, q_0, F, \delta \rangle$ such that*

- *Q is a finite set of states equipped with a map $\|\_\| : Q \to \mathbb{N}$*
- *q_0 is the initial state and $\|q_0\| = 0$*
- *F is the finite set of final states and $\|q\| = 0$ for each $q \in F$*
- *for each $q \in Q$ and $\alpha \in \Sigma \cup \mathcal{N}_{fin} \cup \{\varepsilon, \langle\!\langle, \rangle\!\rangle\}$, we have a set $\delta(q,\alpha) \subseteq Q$ such that for all $q' \in \delta(q,\alpha)$ must hold:*
  - *$\alpha = \langle\!\langle \implies \|q'\| = \|q\| + 1$*
  - *$\alpha = \rangle\!\rangle \implies \|q'\| = \|q\| - 1$*
  - *otherwise $\implies \|q'\| = \|q\|$*

  A transition is a triple (q, α, q') such that $q' \in \delta(q,\alpha)$.

A nominal automaton M is *deterministic* if, for each $q \in Q$,

$$\begin{cases} |\delta(q,\alpha)| = 0, & \text{if } (\alpha = \langle\!\langle \text{ and } \|q\| = max\{\|q'\| \mid q' \in Q\}) \text{ or } (\alpha = \rangle\!\rangle \text{ and } \|q\| = 0) \\ |\delta(q,\alpha)| = 1, & \text{otherwise} \end{cases}.$$

Let $M = \langle Q, q_0, F, \delta \rangle$ be a nominal automata over Σ and $\mathcal{N}_{fin}$, we denote the image of a map σ by $Im(\sigma)$ and the empty map by $\emptyset$. Let q be a state, w be a word whose free names are in $\mathcal{N}_{fin} \cup Im(\sigma)$ and $\sigma : \underline{\|q\|} \to \mathcal{N}$ be a map, a configuration of M is denoted by $\langle q, w, \sigma \rangle$. A configuration $\langle q, w, \sigma \rangle$ is *initial* if $q = q_0$, w is a word whose free names are in $\mathcal{N}_{fin}$, and $\sigma = \emptyset$; a configuration $\langle q, w, \sigma \rangle$ is accepting if $q \in F$, $w = \varepsilon$, and $\sigma = \emptyset$. Given $q, q' \in Q$ and two configurations $t = \langle q, w, \sigma \rangle$ and $t' = \langle q', w', \sigma' \rangle$, M *moves from t to t'* if there is $s \in \Sigma \cup \mathcal{N} \cup \{\varepsilon, \langle\!\langle, \rangle\!\rangle\} \cup \mathbb{N}$ such that $q' \in \delta(q, s)$ and

$$\begin{cases} s \in \underline{\|q\|}, & w = \sigma(s)w',\ \sigma' = \sigma \text{ and } \forall n > s :\ \sigma(s) \neq \sigma(n) \\ s \in \mathcal{N}_{fin} \setminus Im(\sigma) & w = aw',\ \sigma' = \sigma \\ s \in \Sigma & w = aw',\ \sigma' = \sigma \\ s = \varepsilon & w = w',\ \sigma' = \sigma \\ s = \langle\!\langle & w = \langle\!\langle w',\ \sigma' = \sigma[\|q'\| \mapsto n] \\ s = \rangle\!\rangle & w = \rangle\!\rangle w',\ \sigma' = \sigma_{|\underline{\|q'\|}} \end{cases}$$

where $\sigma[\|q'\| \mapsto n]$ extends σ by allocating the maximum index in $\underline{\|q\|}$ to n and $\sigma_{|\underline{\|q'\|}}$ is restriction on $\underline{\|q'\|}$ of σ. The language accepted by M is the set of nominal words w such that M moves from the initial configuration $\langle q, w, \sigma \rangle$ to an accepting configuration. (For more details see [25, 27]).

Theorem 3. *[26] Every language recognised by a nominal automaton is representable by a nominal regular expression. Conversely, every language represented by a nominal regular expression is acceptable by a nominal automaton.*

2.3 Angluin's Algorithm $\mathsf{L}^\star$

The algorithm $\mathsf{L}^\star$ was introduced in [1] to learn a finite automaton accepting a given regular language L over an alphabet Σ. The basic idea of the algorithm is to implement a dialogue between a *learner* and a *teacher*. The learner may ask the teacher for *membership* queries "$w \in L$?" to check whether a word w is in the given language. Moreover, the learner may submit an automaton M to the teacher who replies "yes"

if $\mathfrak{L}(M) = L$, or provides a counter-example showing that $\mathfrak{L}(M) \neq L$. The teacher is assumed to answer all the learner's questions correctly.

Key data structures of $\mathsf{L}^\star$ are *observation tables* representing finite predicates of words over Σ classifying them as members of L or not.

Definition 4 (Observation Tables [1])**.** *An* observation table (S,E,T) *consists of nonempty finite languages* $S,E \subseteq \Sigma^\star$ *such that* S *is prefix-closed and* E *is suffix-closed, and a function* $T : (S \cup S\cdot\Sigma)\cdot E \to \{0,1\}$.

The rows of an observation table are labelled by elements of $S \cup S\cdot\Sigma$, and the columns are labelled by elements of E with the entry for row s and column e given by $T(s\cdot e)$. A row of the table can be represented by a function $row(s) : E \to \{0,1\}$ such that $row(s)(e) = T(s\cdot e)$. A word $s\cdot e$ is a member of L of (S,E,T) iff $T(s\cdot e) = 1$. An observation table (S,E,T) is *closed* when

$$\forall w \in S\cdot\Sigma\,.\,\exists s \in S\,.\,row(w) = row(s)$$

An observation table (S,E,T) is *consistent* when for all $a \in \Sigma$ and all $s,s' \in S$

$$row(s) = row(s') \implies row(sa) = row(s'a)$$

A closed and consistent observation table (S,E,T) has an associated finite automaton $M = (Q,\delta,q_0,F)$ given by

- $Q = \{row(s) \mid s \in S\}$,
- $q_0 = row(\varepsilon)$,
- $F = \{row(s) \mid row(s)(\varepsilon) = 1, s \in S\}$,
- $\delta(row(s),a) = row(s\cdot a),\ a \in \Sigma$.

To see that this is a well-defined automaton, note that the initial state is defined since S is prefix-closed and must contain ε. Similarly, E is suffix-closed and must contain ε. And, if $s,s' \in S, row(s) = row(s')$, then $T(s) = T(s\cdot\varepsilon)$ and $T(s') = T(s'\cdot\varepsilon)$ are equal as defined. The transition function is well-defined since the table is closed and consistent. Suppose s and s' are elements of S such that $row(s) = row(s')$. Since the table (S,E,T) is consistent, $\forall a \in \Sigma, row(sa) = row(s'a)$. And the value of $row(sa)$ is equal to such a $row(s'')$ for an $s'' \in S$, since the table is closed.

The learning process of $\mathsf{L}^\star$ is shown in Figure 1. Let L be the input regular language over an alphabet Σ. Initially, the observation table $(\{\varepsilon\},\{\varepsilon\},T)$ is such that T is initialised by asking for membership queries about ε and each element in Σ (line 2). Then the algorithm enters into the main loop (lines 3-23). Inside of the main loop, a while loop tests the current observation table (S,E,T) for closedness (line 5) and consistency (line 11).

If the current observation table (S,E,T) is not closed, the algorithm finds s' in $S\cdot\Sigma$ such that $row(s')$ is different from $row(s)$ for all $s \in S$. Then the word s' is added into S and new rows are added for words $s'\cdot a$ for all $a \in \Sigma$. Thus, T is extended to $(S\cup S\cdot\Sigma)\cdot E$ by asking for membership queries about missing elements.

Similarly, if (S,E,T) is not consistent, the algorithm finds $s_1,s_2 \in S$,$e \in E$, and $a \in \Sigma$ such that $row(s_1) = row(s_2)$ but $row(s_1\cdot a)(e) \neq row(s_2\cdot a)(e)$. The word $a\cdot e$ is added into E. That is, each row in the table has a new column $a\cdot e$. T is extended to $(S\cup S\cdot\Sigma)\cdot E$ by asking for missing elements $row(s)(a\cdot e)$ for all $s \in (S\cup S\cdot\Sigma)$.

An associated automaton M is constructed when the observation table (S,E,T) is closed and consistent. And then, an equivalence query about M is asked for. The algorithm terminates and outputs M when the teacher replies "yes" to the query. If the teacher replies with a counterexample c, the word c and all its prefixes are added into S, and then T is extended by asking membership queries about new entries in $(S\cup S\cdot\Sigma)\cdot E$. Then, a new round for the main loop of closedness and consistency starts.

Initialisation: $S = \{\varepsilon\}, E = \{\varepsilon\}$.
Construct the initial observation table (S, E, T) by asking for membership queries about $(S \cup S \cdot \Sigma) \cdot E$.
repeat
  while (S, E, T) is not closed or not consistent **do**
    if (S, E, T) is not closed **then**
      find $s' \in S \cdot \Sigma$ such that
      $row(s) \neq row(s')$ for all $s \in S$,
      add s' into S,
      extend T to $(S \cup S \cdot \Sigma) \cdot E$ using membership queries.
    end if
    if (S, E, T) is not consistent **then**
      find $s_1, s_2 \in S$,$e \in E$ and $a \in \Sigma$ such that
      $row(s_1) = row(s_2)$ and $row(s_1 \cdot a)(e) \neq row(s_2 \cdot a)(e)$,
      Add $a \cdot e$ into E,
      extend T to $(S \cup S \cdot \Sigma) \cdot E$ using membership queries.
    end if
  end while
  Construct an automaton M from table (S, E, T) and ask teacher an equivalence query.
  if teacher replies a counterexample c **then**
    add c and all its prefixes into S.
    extend T to $(S \cup S \cdot \Sigma) \cdot E$ using membership queries.
  end if
until teacher replies yes to equivalence query M.
Halt and output M.

Figure 1: The learner in $\mathsf{L}^\star$.

3 Nominal Learning

In this section, we introduce our learning algorithm based on nominal automata. Our teacher still answers two kinds of queries: membership queries and equivalence queries regarding a target nominal regular language.

3.1 Preliminaries

Before introducing our learning algorithm, some auxiliary notions are necessary to give a concrete representation of nominal languages and automata. In fact, binders yield infinitely many equivalent representation of nominal words due to alpha-conversion. For instance, $\langle\!\langle \mathsf{n.n} \rangle\!\rangle$ and $\langle\!\langle \mathsf{m.m} \rangle\!\rangle$ are the same nominal word *up-to* renaming of their bound name. We introduce *canonical expressions* to give a finitary representation of nominal regular languages.

Definition 5 (Canonical expressions)**.** *Let* $1 \leq n \in \mathbb{N}$ *a natural number and ne a closed nominal regular expression. The* n-canonical representation $\chi(ne, n)$ *of ne is defined as follows*

- $ne \in \{\varepsilon, \emptyset\} \cup \Sigma \implies \chi(ne, n) = ne$
- $\chi(ne + ne', n) = \chi(ne, n) + \chi(ne', n)$
- $\chi(ne \cdot ne', n) = \chi(ne, n) \cdot \chi(ne', n)$,

- $\chi(ne^*,n) = (\chi(ne,n))^*$
- $ne = \langle \mathsf{n}.ne' \rangle \implies \chi(ne,n) = \langle n.\chi(ne'[n/\mathsf{n}], n+1) \rangle$

where $ne'[n/\mathsf{n}]$ *is the capture-avoiding substitution of* n *for* n *in* ne'. *The* canonical representation *of* ne *is the term* $\chi(ne,1)$.

Note that the map $\chi(\_,\_)$ does not change the structure of the nominal regular expression ne. Basically, $\chi(\_,\_)$ maps nominal regular expressions to terms where names are concretely represented as positive numbers.

Example 1. *Given* $\Sigma = \{a,b\}$, *we give some examples of canonical representations of nominal expressions.*

- aba *is the canonical representations of itself; indeed* $\chi(aba,1) = aba$
- $\chi(\langle \mathsf{n}.a\,\mathsf{n} \rangle, 1) = \chi(\langle \mathsf{m}.a\,\mathsf{m} \rangle, 1) = \langle 1.a\,1 \rangle$ *is the canonical representation of both* $\langle \mathsf{n}.a\,\mathsf{n} \rangle$ *and* $\langle \mathsf{m}.a\,\mathsf{m} \rangle$
- *the canonical representation of* $ne = \langle \mathsf{n}.a\,\mathsf{n} \langle \mathsf{m}.\mathsf{n}\,b\,\mathsf{m} \rangle \rangle \langle \mathsf{m}.\mathsf{m} \rangle$ *is* $\chi(ne,1) = \langle 1.a1\chi((\langle \mathsf{m}.1\,b\,\mathsf{m} \rangle), 2) \langle 1.1 \rangle$ $= \langle 1.a\,1 \langle 2.1\,b\,2 \rangle \rangle \langle 1.1 \rangle$.

Note that the map $\chi(\_,\_)$ *replaces names with numbers so that alpha-equivalent expressions are mapped to the same term (second example above).*

Canonical expressions are the linguistic counter part of the mechanism used in the definition of $(\Sigma, \mathscr{N}_{fin})$-automaton give in [27], where transitions with indexes can consume bound names of words. Thus, we use canonical expressions in order to represent nominal languages concretely.

Fix a nominal regular language L, in our algorithm, the learner asks for membership queries about legal words. If a word w is not legal, the learner marks it as $\perp$ in the observation table. The membership query consists of a legal word w and it has the following possible answers:

- if $w \in L$, the answer is "1",
- if w is a prefix of a word in L, the answer is "P",
- otherwise, the answer is "0".

As in Angluin's $\mathsf{L}^\star$ algorithm, only the teacher knows L. Unlike in $\mathsf{L}^\star$, the learner in our algorithm does not know the whole alphabet A_n. The learner knows Σ initially and learns names via counterexamples. We will see that the learner knows the whole alphabet when the algorithm terminates.

Remark. The answer "P" is used for efficiency. In fact, the teacher could answer "0" instead of "P". However, this would require the learner to ask more membership or equivalence queries. This is confirmed by some experimental results that are not in scope of this paper.

3.2 Nominal observation tables

Observation tables are pivotal data structure to ensure the algorithm's functionalities. A closed and consistent observation table allows us to construct a minimal automaton. We extend Angluin's observation tables to *nominal observation tables*, n-observation tables for short.

Definition 6 (n-observation table)**.** *A* legal word *is a prefix of a nominal word; the* depth $\|w\|$ *of a legal word* w *is the highest number of nested binders in* w. *Let*

$$\mathbb{A}_0 = \Sigma \qquad \textit{and} \qquad \mathbb{A}_n = \{\langle\!\langle, \rangle\!\rangle\} \cup \Sigma \cup \underline{n} \quad \textit{for } 0 < n \in \mathbb{N}$$

A tuple $(S,E,T,\mathbb{A}_n)$ *is an* n-observation table *if*

- $S \subseteq {\mathbb{A}_n}^*$ *is a prefix-closed set of legal strings, for all* $s \in S, \|s\| \leq n$,
- $E \subseteq {\mathbb{A}_n}^*$ *is suffix-closed,*
- $T : (S \cup S \cdot \mathbb{A}_n) \cdot E \to \{0, 1, P, \bot\}$.

As in Angluin's definition, an n-observation table $(S, E, T, \mathbb{A}_n)$ consists of rows labelled by legal words in $S \cup S \cdot \mathbb{A}_n$ and columns labelled by words in E:

$$row : (S \cup S \cdot \mathbb{A}_n) \to (E \to \{0, 1, P, \bot\})$$
$$row(s)(e) = T(s \cdot e)$$

In order to reflect the layers of nominal automata, we use $\|\_\|$ to distinguish rows. Therefore, we need the following auxiliary notion of equivalence of rows: in an n-observation table $(S, E, T, \mathbb{A}_n)$, for all $s, s' \in S \cup S \cdot \mathbb{A}_n$,

$$row(s) \doteq row(s') \iff row(s) = row(s') \text{ and } \|s\| = \|s'\|.$$

Accordingly, the definition of closed and consistent table changes as follows.

Definition 7 (Closed and Consistent Tables)**.** *An* n-observation table $(S, E, T, \mathbb{A}_n)$ *is* closed *when*

$$\forall s' \in S \cdot \mathbb{A}_n . \exists s \in S.\ row(s') \doteq row(s).$$

An n-observation table $(S, E, T, \mathbb{A}_n)$ *is* consistent *when*

$$\forall \alpha \in \mathbb{A}_n . \forall s, s' \in S\ \ row(s) \doteq row(s') \implies row(s\alpha) \doteq row(s'\alpha).$$

3.3 From n-observation tables to nominal automata

Analogously to Angluin's theory, closed and consistent n-observation tables correspond to deterministic finite nominal automata.

Definition 8. *The* $(\Sigma, \mathcal{N}_{fin})$*-automaton* $M = (Q, q_0, F, \delta)$ *associated with a closed and consistent* n-observation table $(S, E, T, \mathbb{A}_n)$ *is defined as*

- $\Sigma = \mathbb{A}_n \setminus \{\{\langle\!\langle, \rangle\!\rangle\} \cup \underline{n}\}$, $\mathcal{N}_{fin} = \underline{n}$,
- *a set of states* $Q = \{(row(s), \|s\|) \mid s \in S\}$ *with a map* $\|\_\|_M$, *and* $\|q\|_M = \|s\|$ *for each* $q = (row(s), \|s\|) \in Q$,
- *an initial state* $q_0 = (row(\varepsilon), \|\varepsilon\|)$,
- *a set of final states* $F = \{(row(s), \|s\|) \mid row(s)(\varepsilon) = 1, \|s\| = 0 \text{ and } s \in S\}$,
- *A transition function is a partial function* $\delta : Q \times \mathbb{A}_n \to Q$*: for all* $s \in S, \alpha \in \mathbb{A}_n$, $\delta((row(s), \|s\|), \alpha) = (row(s\alpha), \|s\alpha\|)$ *if* $s\alpha \in S \cup S \cdot \mathbb{A}_n$.

Accordingly, we define a partial function $\delta^* : Q \times {\mathbb{A}_n}^* \to Q$ inductively as follows

$$\delta^*(q, \varepsilon) = q$$
$$\delta^*(q, aw) = \delta^*(\delta(q, a), w)$$

for all $a \in \mathbb{A}_n$, $w \in {\mathbb{A}_n}^*$, $q \in Q$. Note that $\delta^*(q, a) = \delta^*(q, a \cdot \varepsilon) = \delta^*(\delta(q, a), \varepsilon) = \delta(q, a)$.

Lemma 1. *Let* $M = (Q, q_0, F, \delta)$ *be the automaton associated with a closed and consistent n-observation table* $(S, E, T, \mathbb{A}_n)$. *Suppose* $w, u \in {\mathbb{A}_n}^*$. *We have* $\delta^*(q, w \cdot u) = \delta^*(\delta^*(q, w), u)$ *for all* $q \in Q$.

Proof. By induction on length of w. □

Theorem 4. *Assume that $M = (Q, q_0, F, \delta)$ is the automaton associated with a closed and consistent n-observation table $(S, E, T, \mathbb{A}_n)$.*

- *For all w in $S \cup S \cdot \mathbb{A}_n$, $\delta^*(q_0, w) = (row(w), \|w\|)$.*
- *For all w in $S \cup S \cdot \mathbb{A}_n$ and u in E, $\delta^*(q_0, w \cdot u)$ in F if and only if*

$$row(w)(u) = 1.$$

Proof. Let $w = w'a$ in $S \cup S \cdot \mathbb{A}_n$ and $u = a \cdot u'$ in E.

Since S is prefix-closed, all prefixes of w are in S, that is, w' is in S. We know:

$$\begin{aligned}
\delta^*(q_0, w) &= \delta^*(q_0, w'a) \\
&= \delta^*(\delta^*(q_0, w'), a) && \text{by Lemma 1} \\
&= \delta^*((row(w'), \|w'\|), a) && \text{by induction hypothesis} \\
&= \delta((row(w'), \|w'\|), a) && \text{by the definition of } \delta^* \\
&= (row(w'a), \|w'a\|) && \text{by the definition of } \delta \\
&= (row(w), \|w\|)
\end{aligned}$$

Since E is suffix-closed, all suffixes of u are in E. Depending on the length of u, we have two situations.

- When $u = \varepsilon$, $row(w)(u) = row(w)(\varepsilon)$ and $\delta^*(q_0, w \cdot u) = \delta^*(q_0, w)$. From preceding proof, $\delta^*(q_0, w) = (row(w), \|w\|)$. Because the table is closed, there is a $w' \in S$ such that $row(w') \doteq row(w)$. $\delta^*(q_0, w \cdot u)$ is in F if and only if $row(w')$ is in F from the definition of F. Thus $row(w')(\varepsilon) = row(w)(\varepsilon) = 1$, that is, $row(w)(u) = 1$.
- Assume that when the length of $u' \in E$ is n, we have that for all w in $S \cup S \cdot \mathbb{A}_n$ and u in E, $\delta^*(q_0, w \cdot u)$ in F if and only if $row(w)(u) = 1$. Let $u = au'$ and $u \in E$. Because the table is closed, there is a $w' \in S$ such that $row(w') \doteq row(w)$.

$$\begin{aligned}
\delta^*(q_0, w \cdot u) &= \delta^*(\delta^*(q_0, w), u) && \text{by Lemma 1} \\
&= \delta^*((row(w), \|w\|), u) && \text{by preceding proof} \\
&= \delta^*((row(w'), \|w'\|), u) && \text{since } row(w') = row(w) \\
&= \delta^*((row(w'), \|w'\|), au') && u = au' \\
&= \delta^*(row(w' \cdot a), u') && \text{by closedness and definition of } \delta \\
&= \delta^*(\delta^*(q_0, w' \cdot a), u') && \text{by preceding proof} \\
&= \delta^*(q_0, w' \cdot a \cdot u')
\end{aligned}$$

By induction hypothesis on u', $\delta^*(q_0, w' \cdot a \cdot u')$ is in F if only if $row(w' \cdot a)(u') = 1$. Because $row(w) \doteq row(w')$ and $u = au'$, $row(w' \cdot a)(u') = T(w' \cdot a \cdot u') = row(w')(a \cdot u') = row(w')(u) = row(w)(u)$. Therefore $\delta^*(q_0, w \cdot u)$ in F if and only if $row(w)(u) = 1$. □

We are now ready to introduce a learning algorithm for our nominal automata.

1: Initialisation: $S = \{\varepsilon\}, E = \{\varepsilon\}$, $n = 0$
2: Asking for membership queries about ε and each $a \in \mathbb{A}_n$ build the initial observation table $(S, E, T, \mathbb{A}_n)$.
3: **repeat**
4: **while** $(S, E, T, \mathbb{A}_n)$ is not closed or consistent **do**
5: **if** $(S, E, T, \mathbb{A}_n)$ is not closed **then**
6: **find** $s' \in S \cdot \mathbb{A}_n$ such that $row(s) \doteq row(s')$ is not satisfied for all $s \in S$
7: **add** s' into S
8: **extend** T to $(S \cup S \cdot \mathbb{A}_n) \cdot E$ using membership queries.
9: **end if**
10: **if** $(S, E, T, \mathbb{A}_n)$ is not consistent **then**
11: **find** $s_1, s_2 \in S$,$e \in E$ and $a \in \mathbb{A}_n$ such that $row(s_1) \doteq row(s_2)$ but $row(s_1 \cdot a)(e) \neq row(s_2 \cdot a)(e)$.
12: **Add** $a \cdot e$ into E
13: **extend** T to $(S \cup S \cdot \mathbb{A}_n) \cdot E$ using membership queries.
14: **end if**
15: **end while**
16: Construct an automata M associated to $(S, E, T, \mathbb{A}_n)$.
17: Ask equivalence query about M.
18: **if** teacher replies a counterexample c **then**
19: add c and all its prefixes into S.
20: **extend** $\mathbb{A}_n$ with $\|s\|$ for all $s \in S$, $\|s\| > 0$.
21: **extend** T to $(S \cup S \cdot \mathbb{A}_n) \cdot E$ using membership queries.
22: **end if**
23: **until** teacher replies yes to M.
24: Halt and output M.

Figure 2: The learner of Learning Algorithm for Nominal Automaton

4 The nL* Algorithm

We dubbed our algorithm nL*, after *nominal* L*. The algorithm is shown in Figure 2. The learner in nL* is similar to the one in L*. Basically, our learner modifies the initial n-observation table until it becomes closed and consistent in the nominal sense (according to notions introduced before). When the current n-observation table $(S, E, T, \mathbb{A}_n)$ is closed and consistent, the learner would ask the teacher if the automaton associated with $(S, E, T, \mathbb{A}_n)$ accepts the input language L. If this is the case, the teacher will reply "yes" and the learning process halts. Otherwise, the learning process continues after the teacher has produced a counterexample.

Because of the new definitions of closedness and consistency, we refine some actions about checking closedness (line 6) and consistency (line 11). If the current n-observation table $(S, E, T, \mathbb{A}_n)$ is not closed, the learner finds a row s' such that for no $s \in S$ we have $row(s') \doteq row(s)$. If $(S, E, T, \mathbb{A}_n)$ is not consistent, the learner finds a word $a \cdot e$, with $a \in \mathbb{A}_n$ and $e \in E$, such that for some $s_1 \in S$ and $s_2 \in S$ with $row(s_1) \doteq row(s_2)$, we have $row(s_1 \cdot a)(e) \neq row(s_2 \cdot a)(e)$.

The main difference with respect to the algorithm of Angluin is that the learner has partial knowledge of the alphabet. The alphabet $\mathbb{A}_n$ is enlarged by adding names (line 20) during the learning process. More precisely, the learner expands the alphabet if the counterexample requires to allocate fresh names. When names are required, the operators of allocations and deallocations are added into $\mathbb{A}_n$. When the algorithm terminates, the learner's alphabet $\mathbb{A}_n$ is the alphabet of the given language. Like in the original algorithm, our algorithm terminates when the teacher replies "yes" to an equivalence query.

We now show that nL* is correct. That is, that eventually the teacher replies "yes" to an equivalence

query. In other word, $\mathsf{nL}^\star$ terminates with a "yes" answer to an equivalence query. Hence, the automaton submitted in the query accepts the input language.

Theorem 5. *The algorithm terminates, hence it is correct.*

Proof. We show that the if- and the while-statements terminate. Let us consider the if-statements first. It is easy to check that closedness and consistency are decidable because these properties require just the inspection of the n-observation table (which is finite). Hence, the if-statements starting at lines 5 and 10 never diverge because their guards do not diverge and their then-branch is a finite sequence of assignments:

- The if-statement for closedness (line 5) terminates directly if the table is closed. Otherwise, in case of making a table closed, we find a row $s' \in S \cdot \mathbb{A}_n$ such that $row(s') \doteq row(s)$ is not satisfied for all $s \in S$. The algorithm adds s' into S. Since $\mathbb{A}_n$ is finite and bounded by n, the sets S and E are both finite. Thus, $S \cdot \mathbb{A}_n$ is a finite set and there are finitely many choices for s'. That is, line 7 can only be executed finitely times. Besides, the content of rows is one of the permutations and combinations of $\{0, 1, P, \bot\}$ which also has finite possibilities. So we conclude that the branch terminates.
- Similarly, the if-statement for consistency (line 10) terminates directly if the table is consistent. Otherwise, to make the table consistent the algorithm searches for two rows $s_1, s_2 \in S$ satisfying the condition at line 11. As in the previous case for closedness, to add elements into E there are only finitely many possibilities s_1, s_2, a, and e (line 12). Thus, the branch of the if-statement terminates.

Therefore, the while-statement (line 4) terminates in finite repetitions, since the algorithm makes a table closed and consistent in finite operations. Then, the algorithm will succeed in construct an automata M associated to a closed and consistent table. Next, the learner asks for an equivalence query. The teacher replies a counterexample c (line 18) or yes (line 23). It remains to prove that the learner only asks finitely many equivalence queries.

Let M be the nominal automaton associated to the current n-observation table $(S, E, T, \mathbb{A}_n)$. Assume that the equivalence query about M fails. The teacher has to find a counterexample c; this is finitely computable since the nominal regular expressions are closed under the operations of Kleene algebra and under resource complementation. Hence, the if-statement on line 18 goes the branch extending the table (line 19). Then, the algorithm will start a new loop (line 4) for the modified table by the counterexamples. We can prove that a closed and consistent table builds a minimal finite automaton (omitted for space reasons). A new automaton M' will be constructed when the extended table $(S, E, T, \mathbb{A}_n)$ is closed and consistent. Since M' handles the counterexample, M' has more equivalent states to the minimal automaton accepted the given language, compared with M. Repeating this process, the automaton associated with a closed and consistent table has the same number of the minimal automaton which accepted the given language. Since the minimal automaton is unique, the two minimal automata are equal. The teacher relies yes to the automaton at such a point. Therefore, with respect to the number of the states of the minimal automaton accepted the given language, equivalence queries are finite and the algorithm terminates finally. □

In the following, we analyse the number of queries and the execution time of $\mathsf{nL}^\star$ in the worst case. Let M be the minimal automaton accepting the given language and let M have $\mathfrak{s}$ states. Let $\mathfrak{b}$ be a bound on the maximum length of the counterexamples presented by the teacher.

From the Figure 2 (line 1), we know that S and E contain one element λ initially. As the algorithm runs, it will add one element to S when $(S, E, T, \mathbb{A}_n)$ is not closed (line 8). And it will add one element to

E when $(S,E,T,\mathbb{A}_n)$ is not consistent (line13). For each counterexample of length at most $\mathfrak{b}$ presented by the teacher, the algorithm will add at most $\mathfrak{b}$ elements to S (line 19).

Thus, the cardinality of S is depends on $\mathfrak{s}$ and $\mathfrak{b}$. In detail, S is at most

$$1+(\mathfrak{s}-1)+\mathfrak{b}(\mathfrak{s}-1)=\mathfrak{s}+\mathfrak{b}(\mathfrak{s}-1)$$

because $(S,E,T,\mathbb{A}_n)$ can be not closed at most $\mathfrak{s}-1$ times. As the same as the teacher replies counterexamples at most $\mathfrak{s}-1$ times. And each time the teacher replies with a counterexample of length $\mathfrak{b}$, S will be increased by at most $\mathfrak{b}$ elements.

The cardinality of E is at most $\mathfrak{s}$, because $(S,E,T,\mathbb{A}_n)$ can be not consistent at most $\mathfrak{s}-1$ times.

The cardinality of $S\cdot\mathbb{A}_n$ could calculate from two parts: the cardinality of S and the cardinality of $\mathbb{A}_n$. We already know the cardinality of S is at most $\mathfrak{s}+\mathfrak{b}(\mathfrak{s}-1)$. As the definition of $\mathbb{A}_n$, let $\mathfrak{k}$ be the cardinality of Σ. Therefore the cardinality of $\mathbb{A}_n$ is $\mathfrak{k}+n+2$ and the cardinality of $S\cdot\mathbb{A}_n$ is $(\mathfrak{k}+n+2)(\mathfrak{s}+\mathfrak{b}(\mathfrak{s}-1))$ at most.

Therefore, the maximum cardinality of $(S\cup S\cdot\mathbb{A}_n)\cdot E$ is at most

$$(\mathfrak{k}+n+2)(\mathfrak{s}+\mathfrak{b}(\mathfrak{s}-1))\mathfrak{s}=O((\mathfrak{k}+n)\mathfrak{b}\mathfrak{s}^2).$$

The minimal automaton M has $\mathfrak{s}$ states and the table $(S,E,T,\mathbb{A}_n)$ has one row initially. In the worst case, the table $(S,E,T,\mathbb{A}_n)$ adds only a distinguished row by every counterexample. The algorithm produces at most $\mathfrak{s}-1$ equivalence queries.

Running nL*: An Example Given a finite alphabet $\Sigma=\{a,b\}$, we have an example of learning a language L representing as canonical nominal regular expression $cne=ab\langle 1^*\rangle$.

In the first step, we initialize $S_1=\{\varepsilon\}$, $E_1=\{\varepsilon\}$, $n=0$ and $\mathbb{A}_0=\Sigma$, and construct T_1 as follows.

Step 1

$T_1=$

$\|\_\|$		ε
0	ε	P
0	a	P
0	b	0

$(S_1,E_1,T_1,\mathbb{A}_0)$ consistent? There is only one row in S, thus the table is consistent.
$(S_1,E_1,T_1,\mathbb{A}_0)$ closed? No, $row(b)\neq row(\varepsilon)$.
So, $S_2\leftarrow S_1\cup\{b\}$ and we go to step 2.

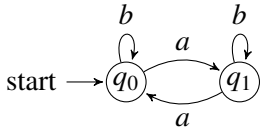
Step 2

Let $S_2=S\cup\{b\}$ and $E_2=E$ and then construct a new observation table $(S_2,E_2,T_2,\mathbb{A}_0)$ through membership queries.

$T_2=$

$\|\_\|$		ε
0	ε	P
0	b	0
0	a	P
0	aa	0
0	ab	P

$(S_2,E_2,T_2,\mathbb{A}_0)$ closed? $\surd$
$(S_2,E_2,T_2,\mathbb{A}_0)$ consistent? $\surd$
Then, we compute the automaton M:



Teacher replies no and a counterexample, say, $ab\langle\!\langle 1.\rangle\!\rangle$. It is in L not in M. And we go to step 3.

Step 3

Let $S_3\leftarrow S_2\cup\{a,ab,ab\langle\!\langle 1.,ab\langle\!\langle 1.\rangle\!\rangle\}$, $E_3\leftarrow E_2$, and $n=1$, and then, the alphabet is extended to $\mathbb{A}_1=\Sigma\cup\underline{n}\cup\{\langle\!\langle,\rangle\!\rangle\}$. We should construct new observation tables sequentially through membership queries. Then we check the new table for closeness and consistency.

T_3=

$\|\_\|$		ε
0	ε	P
0	b	0
0	a	P
0	ab	P
1	$ab\langle\!\langle 1.$	P
0	$ab\langle\!\langle 1.\rangle\!\rangle$	1
1	$\langle\!\langle 1.$	0
0	a	P
0	$ab\langle\!\langle 1.\rangle\!\rangle a$	0
0	$ab\langle\!\langle 1.\rangle\!\rangle b$	0
1	$ab\langle\!\langle 1.1$	P
1	$ab\langle\!\langle 1.a$	0
...	...	...

$(S_3, E_3, T_3, \mathbb{A}_1)$ consistent?
No, $row(ab) = row(\varepsilon)$ but $row(ab\langle\!\langle 1.) \neq row(\langle\!\langle 1.)$.
$(S_3, E_3, T_3, \mathbb{A}_1)$ closed?
No, $row(\langle\!\langle 1.)$ with $\|\langle\!\langle 1.\| = 1$ has a fresh content.

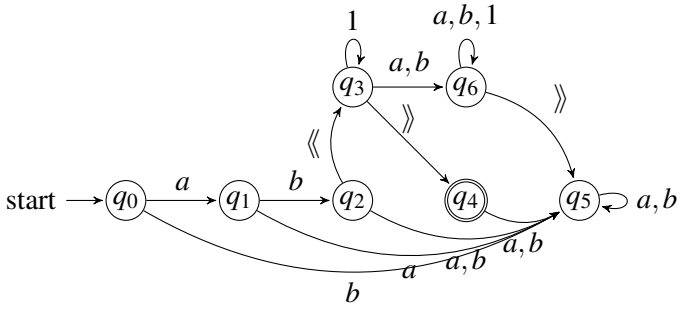
Step 4

Let $S_4 \leftarrow S_3 \cup \{\langle\!\langle 1.\}$,$E_4 \leftarrow E_3 \cup \{\langle\!\langle 1.\}$, we should construct a new observation table $(S_4, E_4, T_4, \mathbb{A}_1)$ and check the new table for closeness and consistency.

T_4=

$\|\_\|$		ε	$\langle\!\langle 1.$
0	ε	P	0
0	b	0	0
0	a	P	0
0	ab	P	P
1	$ab\langle\!\langle 1.$	P	$\perp$
0	$ab\langle\!\langle 1.\rangle\!\rangle$	1	0
1	$\langle\!\langle 1.$	0	$\perp$
0	ba	0	0
0	bb	0	0
1	$ab\langle\!\langle 1.1$	P	$\perp$
1	$ab\langle\!\langle 1.a$	0	$\perp$
...	...	...	...

Once the table is closed and consistent, we ask an equivalence query.
Finally, the teacher replies "yes" to an equivalence query about. The learning progress terminates. The learner automaton is as below.



5 Conclusions

The learning algorithm $\mathsf{L}^\star$ was introduced more than thirty years ago and has been intensively extended to many types of models in following years. This algorithm continues to attract the attention of many researchers [2, 36, 9].

We designed a learning algorithm for a class of languages over infinite alphabet; more precisely, we have considered nominal regular languages with binders [25, 27]. We have tackled the finitary representations of the alphabets, words and automata for retaining the basic scheme and ideas of $\mathsf{L}^\star$. Hence, we revised and added definitions for the nominal words and automata. Further, accounting for names and the allocation and deallocation operations, we revised the data structures and notions in $\mathsf{L}^\star$. Accordingly, we have proposed the learning algorithm, $\mathsf{nL}^\star$, to stress the progress of learning a nominal language with binders. We have proved the correctness and analysed the complexities of $\mathsf{nL}^\star$.

As for $\mathsf{L}^\star$, a key factor for the effectiveness of and $\mathsf{nL}^\star$ is the selection of counterexamples. Due to possibly infinite number of candidate counterexamples, the selection of the counterexamples is non-deterministic in $\mathsf{nL}^\star$. We are developing an implementation of $\mathsf{nL}^\star$ to study effective mechanisms to resolve this non-determinism. Interestingly, the rich structure of nominal automata offer different directions to solve this problem. In fact, we started to investigate this issue and defined two different strategies used by the teacher to generate counterexamples based on the "size" of the counterexamples and on the preference of the teacher for counterexamples with maximal or minimal number of fresh names. Initial experiments show how the strategies impact on the convergence of $\mathsf{nL}^\star$. This immediately suggest that $\mathsf{nL}^\star$ could be improved by designing different strategies to generate "better" counterexamples, that is counterexamples that allow the learner to learn "more quickly".

References

[1] Dana Angluin (1987): *Learning Regular Sets from Queries and Counterexamples*. Inf. Comput. 75(2), pp. 87–106, doi:10.1016/0890-5401(87)90052-6.

[2] Dana Angluin & Tyler Dohrn (2017): *The Power of Random Counterexamples*. In: International Conference on Algorithmic Learning Theory, ALT 2017, 15-17 October 2017, Kyoto University, Kyoto, Japan, pp. 452–465. Available at `http://proceedings.mlr.press/v76/angluin17a.html`.

[3] Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari & Roberto Zunino (2009): *Local policies for resource usage analysis*. ACM Trans. Program. Lang. Syst. 31(6), doi:10.1145/1552309.1552313.

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

[5] Benedikt Bollig, Peter Habermehl, Martin Leucker & Benjamin Monmege (2013): *A Fresh Approach to Learning Register Automata*. In: Developments in Language Theory - 17th International Conference, DLT2013, Marne-la-Vallée, France, June 18-21, 2013. Proceedings, pp. 118–130, doi:10.1007/978-3-642-38771-5_12.

[6] Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern & Martin Leucker (2010): *Learning Communicating Automata from MSCs*. IEEE Trans. Software Eng. 36(3), pp. 390–408, doi:10.1109/TSE.2009.89.

[7] Daniel Brand & Pitro Zafiropulo (1983): *On Communicating Finite-State Machines*. JACM 30(2), pp. 323–342, doi:10.1145/322374.322380.

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

[9] Giuseppe Castagna & Andrew D. Gordon, editors (2017): *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. ACM, doi:10.1145/3009837.

[10] Pierpaolo Degano, Gian Luigi Ferrari & Gianluca Mezzetti (2012): *Nominal Automata for Resource Usage Control*. In: Implementation and Application of Automata - 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings, pp. 125–137, doi:10.1007/978-3-642-31606-7_11.

[11] Pierpaolo Degano, Gian Luigi Ferrari & Gianluca Mezzetti (2013): *Towards Nominal Context-Free Model-Checking*. In: *Implementation and Application of Automata - 18th International Conference, CIAA 2013, Halifax, NS, Canada, July 16-19, 2013. Proceedings*, pp. 109–121, doi:10.1007/978-3-642-39274-0_11.

[12] Gianluigi Ferrari, Giovanni Ferro, Stefania Gnesi, Ugo Montanari, Marco Pistore & Gioia Ristori (1997): *An Automata Based Verification Environment for Mobile Processes*. In Ed Brinksma, editor: *TACAS*, *LNCS* 1217, Springer, pp. 275–289.

[13] Gianluigi Ferrari, Stefania Gnesi, Ugo Montanari, Marco Pistore & Gioia Ristori (1998): *Verifying Mobile Processes in the HAL Environment*. In: *Proc. 10th International Computer Aided Verification Conference*, pp. 511–515, doi:10.1007/BFb0028772.

[14] Gianluigi Ferrari, Ugo Montanari & Marco Pistore (2002): *Minimizing Transition Systems for Name Passing Calculi: A Co-algebraic Formulation*. In Mogens Nielsen & Uffe Engberg, editors: *Foundations of Software Science and Computation Structures*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 129–143, doi:10.1007/3-540-45931-6_10.

[15] Murdoch J. Gabbay (2001): *A Theory of Inductive Definitions with alpha-Equivalence*. phdthesis, University of Cambridge, UK. Available at `http://www.gabbay.org.uk/papers.html#thesis`.

[16] Murdoch J. Gabbay & Andrew M. Pitts (1999): *A New Approach to Abstract Syntax Involving Binders*. In Giuseppe Longo, editor: *LICS*, IEEE, Trento, Italy, pp. 214–224, doi:10.1109/LICS.1999.782617.

[17] Murdoch J. Gabbay & Andrew M. Pitts (2002): *A New Approach to Abstract Syntax with Variable Binding*. *J. of Formal Aspects of Computing* 13(3-5), pp. 341–363, doi:10.1007/s001650200016.

[18] Fabio Gadducci, Marino Miculan & Ugo Montanari (2006): *About permutation algebras, (pre)sheaves and named sets*. *Higher-Order and Symbolic Computation* 19(2-3), pp. 283–304, doi:10.1007/s10990-006-8749-3.

[19] John E. Hopcroft, Rajeev Motwani & Jeffrey D. Ullman (2001): *Introduction to Automata Theory, Languages, and Computation, 2Nd Edition*. *SIGACT News* 32(1), pp. 60–65, doi:10.1145/568438.568455.

[20] Bart Jacobs & Alexandra Silva (2014): *Automata learning: A categorical perspective*. In: *Horizons of the Mind. A Tribute to Prakash Panangaden*, Springer, pp. 384–406, doi:10.1007/978-3-319-06880-0_20.

[21] Michael Kaminski & Nissim Francez (1994): *Finite-Memory Automata*. *Theor. Comput. Sci.* 134(2), pp. 329–363, doi:10.1016/0304-3975(94)90242-9.

[22] Michael Kaminski & Tony Tan (2006): *Regular Expressions for Languages over Infinite Alphabets*. *Fundam. Inform.* 69(3), pp. 301–318, doi:10.1007/978-3-540-27798-9_20.

[23] Stephen C. Kleene (1956): *Representation of Events in Nerve Nets and Finite Automata*. In John Shannon, Claude E. McCarthy, editor: *Automata Studies*, Princeton University Press, pp. 3–42.

[24] Dexter Kozen, Konstantinos Mamouras, Daniela Petrisan & Alexandra Silva (2015): *Nominal Kleene Coalgebra*. In: *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pp. 286–298, doi:10.1007/978-3-662-47666-6_23.

[25] Alexander Kurz, Tomoyuki Suzuki & Emilio Tuosto (2012): *A Characterisation of Languages on Infinite Alphabets with Nominal Regular Expressions*. In: *Theoretical Computer Science - 7th IFIP TC 1/WG 2.2 International Conference, TCS 2012, Amsterdam, The Netherlands, September 26-28, 2012. Proceedings*, pp. 193–208, doi:10.1007/978-3-642-33475-7_14.

[26] Alexander Kurz, Tomoyuki Suzuki & Emilio Tuosto (2012): *On Nominal Regular Languages with Binders*. In Lars Birkedal, editor: *Foundations of Software Science and Computational Structures*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 255–269, doi:10.1007/978-3-642-28729-9_17.

[27] Alexander Kurz, Tomoyuki Suzuki & Emilio Tuosto (2013): *Nominal Regular Expressions for Languages over Infinite Alphabets. Extended Abstract*. *CoRR* abs/1310.7093. Available at `http://arxiv.org/abs/1310.7093`.

[28] Robin Milner (1999): *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press.

[29] Robin Milner, Joachim Parrow & David Walker (1992): *A Calculus of Mobile Processes, I*. *Inf. Comput.* 100(1), pp. 1–40, doi:10.1016/0890-5401(92)90008-4.

[30] Ugo Montanari & Marco Pistore (2000): *pi-Calculus, Structured Coalgebras, and Minimal HD-Automata*. In: *Mathematical Foundations of Computer Science 2000, 25th International Symposium, MFCS 2000, Bratislava, Slovakia, August 28 - September 1, 2000, Proceedings*, pp. 569–578, doi:10.1007/3-540-44612-5_52.

[31] Oliver Niese (2003): *An integrated approach to testing complex systems*. Ph.D. thesis, Technical University of Dortmund, Germany. Available at `http://eldorado.uni-dortmund.de:8080/0x81d98002_0x0007b62b`.

[32] Corina S. Pasareanu, Dimitra Giannakopoulou, Mihaela Gheorghiu Bobaru, Jamieson M. Cobleigh & Howard Barringer (2008): *Learning to divide and conquer: applying the $L^\star$ algorithm to automate assume-guarantee reasoning*. *Formal Methods in System Design* 32(3), pp. 175–205, doi:10.1007/s10703-008-0049-6.

[33] Marco Pistore (1999): *History Dependent Automata*. Ph.D. thesis, Dipartimento di Informatica, Università di Pisa.

[34] Andrew M. Pitts (2015): *Names and Symmetry in Computer Science (Invited Tutorial)*. In: *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pp. 21–22, doi:10.1109/LICS.2015.12.

[35] Davide Sangiorgi & David Walker (2001): *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press.

[36] Lutz Schröder, Dexter Kozen, Stefan Milius & Thorsten Wißmann (2017): *Nominal Automata with Name Binding*. In: *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, pp. 124–142, doi:10.1007/978-3-662-54458-7_8.

[37] Luc Segoufin (2006): *Automata and Logics for Words and Trees over an Infinite Alphabet*. In: *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, pp. 41–57, doi:10.1007/11874683_3.