A Kleene Theorem for Nominal Automata

Paul Brunet
University College London, UK
paul.brunet-zamansky.fr
paul@brunet-zamansky.fr

Alexandra Silva
University College London, UK
www.alexandrasilva.org
alexandra.silva@ucl.ac.uk

Abstract

Nominal automata are a widely studied class of automata designed to recognise languages over infinite alphabets. In this paper, we present a Kleene theorem for nominal automata by providing a syntax to denote regular nominal languages. We use regular expressions with explicit binders for creation and destruction of names and pinpoint an exact property of these expressions – namely memory-finiteness – identifying a subclass of expressions denoting exactly regular nominal languages.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Formal languages and automata theory

Keywords and phrases Kleene Theorem, Nominal automata, Bracket Algebra

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.107

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version A full version of the paper is available at http://hal.inria.fr/hal-02112892.

Supplement Material A Coq library is available on GitHub.

Funding ERC Starting Grant ProFoundNet (grant code 679127)
Paul Brunet: EPSRC project EP/R006865/1
Alexandra Silva: Leverhulme Prize (PLP-2016-129)

1 Introduction

Languages over infinite alphabets have been studied in a variety of contexts: query-based languages [8], XML processing [19], URLs [1], process calculi [5], etc. Accordingly, a number of automata models have been introduced for these languages, either register-based, where the state space is finite but registers are available for storing data, or based on nominal sets, where the state space is infinite but can be represented finitely due to symmetries. The most general classes of such automata are Kaminski and Francez's finite-memory automata (FMA) [8], in the register-based style, and Bojańczyk, Klin and Lasota's nondeterministic orbit-finite automata (NOFA) [4], in the nominal style. These two kinds of automata have been shown to have the same expressivity [4], and equivalence is known to be undecidable [8, 16].

While automata are useful to process and compare languages, to specify languages it is often more natural to use regular expressions; this is for instance the standard way of denoting a path in an XML tree. To that effect, many classes of expressions have been proposed [9, 12, 11, 18, 14]. The expressions from [14] capture the full class of languages recognised by either FMA of NOFA, but having been developed for FMAs they are not straightforwardly suitable to describe NOFA languages. Some of the other formalisms are more natural in the context of nominal automata, but all fail to capture the full class, and instead coincide with some (usually decidable) sub-classes.


© Paul Brunet and Alexandra Silva;
licensed under Creative Commons License CC-BY
46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 107; pp. 107:1–107:13


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We define in this paper a new class of regular expressions for data languages, originally motivated by applications to program verification, as part of larger framework called *bracket algebra*. These expressions feature explicit allocation $\langle_a$ and deallocation $_a\rangle$ binders, and may be used to generate nominal languages. We prove in this paper that they are in fact able to describe every language recognisable by a NOFA.

Let us illustrate our syntax on simple examples. To make this discussion simpler, we assume for now that our alphabet is an infinite set of names $\mathbb{A}$. The first notion we present is that of α-equivalence of words with binders. Here we choose to define α-equivalence as the smallest congruence stable by permutation of bound or fresh names. For instance the following pair of words is equivalent: $\langle_a\, a\langle_b\, b\,{}_a\rangle\langle_a\, a\,{}_b\rangle\langle_b\, b\,{}_a\rangle\,{}_b\rangle =_\alpha \langle_b\, b\langle_c\, c\,{}_b\rangle\langle_d\, d\,{}_c\rangle\langle_a\, a\,{}_d\rangle\,{}_a\rangle$. Indeed, we may derive this as follows (we underline the redex at each step):

$$\begin{aligned}\langle_a\, a\underline{\langle_b\, b\,{}_a\rangle\langle_a\, a\,{}_b\rangle}\langle_b\, b\,{}_a\rangle\,{}_b\rangle &=_\alpha \langle_a\, a\langle_c\, c\,{}_a\rangle\underline{\langle_a\, a\,{}_c\rangle\langle_b\, b\,{}_a\rangle}\,{}_b\rangle \\ &=_\alpha \underline{\langle_a\, a\langle_c\, c\,{}_a\rangle}\langle_d\, d\,{}_c\rangle\langle_b\, b\,{}_d\rangle\,{}_b\rangle =_\alpha \langle_b\, b\langle_c\, c\,{}_b\rangle\langle_d\, d\,{}_c\rangle\langle_a\, a\,{}_d\rangle\,{}_a\rangle.\end{aligned}$$

We then define well-formed words to be those without name capture, i.e. for every prefix $u\langle_a$, every $\langle_a$ in u must be matched with a corresponding $_a\rangle$. For instance $\langle_a\, a\langle_b\, b\,{}_b\rangle\,{}_a\rangle$ is well-formed, but $\langle_a\, a\langle_a\, a\,{}_a\rangle\,{}_a\rangle$ is not, even though the two are equivalent. Now, consider regular expressions over an alphabet composed of names from $\mathbb{A}$ and binders $\langle_a$ and $_a\rangle$. We associate to such an expression e a nominal language $(\!|e|\!)$ in several steps:

1) take the regular language $[\![e]\!]$ denoted by e;
2) compute its closure by α-equivalence $[\![e]\!]^\alpha$, adding every word that is equivalent to some word in the initial language;
3) restrict this language to its well-formed members;
4) erase the brackets.

Here are some examples:

L1 $:= (\!|\langle_a\, a\,{}_a\rangle|\!) = \mathbb{A}$: the set of all atoms;
L2 $:= (\!|\langle_a\, \langle_b\, ab\,{}_b\rangle\,{}_a\rangle|\!) = \{ab \mid a \neq b\}$: two letter words made of different letters;
L3 $:= (\!|\langle_a\, a\,{}_a\rangle^\star|\!) = \mathbb{A}^\star$: the set of all words;
L4 $:= (\!|\langle_a\, a\langle_a\, a\,{}_a\rangle^\star\,{}_a\rangle|\!) = \{a_1 \dots a_n \mid n > 0, \forall 1 < i, a_i \neq a_1\}$: the set of words such that the first letter is different from all others;
L5 $:= (\!|\langle_a\, \langle_a\, a\,{}_a\rangle^\star a\,{}_a\rangle|\!) = \{a_1 \dots a_n \mid n > 0, \forall i < n, a_i \neq a_n\}$: the set of words such that the last letter is different from all others;
L6 $:= (\!|\langle_a\, a\,(\langle_b\, b\,{}_a\rangle\langle_a\, a\,{}_b\rangle)^\star\,(1 + \langle_b\, b\,{}_b\rangle)\,{}_a\rangle|\!) = \{a_1a_2 \dots a_n \mid n > 0, \forall i, a_i \neq a_{i+1}\}$: the set of non-empty words such that two consecutive letters are different;
L7 $:= (\!|(\langle_a\, x)^\star\,(y\,{}_a\rangle)^\star|\!) = \{x\} \cup \{x^n y^m \mid n \leqslant m\}$.

As one can see, this technique allows for the definition of a large class of nominal languages. In fact this class is in some sense "too large" and contains languages that are not regular, like for instance L7. To get a Kleene theorem, we therefore introduce a tractability condition: we ask regular expressions to have a memory-finite language. Intuitively this means there should be a number N such that any prefix of a word in the language has less than N unmatched brackets. This condition is decidable by induction on expressions, and such expressions generate exactly the class of languages recognisable by NOFAs. The main result of the paper is an exact correspondence between memory-finite nominal languages and NOFA:

▶ **Theorem 1** (Kleene Theorem). *Let L be a nominal language. The following are equivalent:*
(i) *L is rational, that is $L = (\!|e|\!)$ for some memory-finite regular expression e.*
(ii) *L is regular nominal, that is recognisable by a NOFA.*

The paper is structured as follows. In Section 2, we define our notations, and recall some elements of nominal automata theory. We introduce in Section 3 words with explicit binders and define an α-equivalence relation for these words. To recognise this relation we construct in Section 4 a nominal transducer. We then present in Section 5 our syntax for regular expressions with binders, and prove in Section 6 our main result, a Kleene theorem for NOFA. We briefly discuss related work in Section 7. We omit some proofs in this paper, but a longer version is available on HAL.

This paper is part of a larger research program developing a framework to reason about programs with explicit resource (de)allocation. A companion paper describing the algebraic framework of *bracket algebra* and a hierarchy of nominal languages can be found online, as well as a Coq formalisation of the framework.

2 Preliminaries

The set of finite subsets of a set A is denoted by $\mathcal{P}_f(A)$. If A is finite, its cardinal is denoted by $\#A \in \mathbb{N}$. The set of words over an alphabet Σ is written $\Sigma^\star$. The empty word is denoted by ε, concatenation of words u and v is written uv, and $|u|$ is the length of word u. For $w \in \Sigma^\star$ and $x \in \Sigma$, $|w|_x$ is the number of occurrences of x in w. We write $\lfloor w \rfloor$ for the set of letters appearing in the word w, i.e. $\lfloor w \rfloor := \{x \in \Sigma \mid |w|_x > 0\}$. We denote the i^{th} letter of a word u by u_i, for $0 < i \leqslant |u|$. The set of prefixes of a language is defined as: $\mathbf{pref}(L) := \{u \in \Sigma^\star \mid \exists v : uv \in L\}$.

Given a set A, and $B \subseteq A$, the set $B^{\bowtie}$ of shuffles of B consists of the lists without repetitions of elements from B: $B^{\bowtie} := \{l \in B^\star \mid |l| = \#B \wedge (\forall 0 < i < j \leqslant |l|\,,\ l_i \neq l_j)\}$. Observe that if $w \in B^{\bowtie}$, then $\{a \mid \exists 0 < i \leqslant |w| : w_i = a\} = B$. We say that a list $l \in A^\star$ is duplication-free, written $l \in A^{(\star)}$ when $l \in \{a \mid \exists 0 < i \leqslant |l| : l_i = a\}^{\bowtie}$.

Rational expressions over an alphabet Σ are terms generated by the following grammar: $e, f \in \mathrm{Rat}\,\langle\Sigma\rangle ::= 0 \mid 1 \mid l \mid e + f \mid e \cdot f \mid e^\star$, where l ranges over the alphabet Σ. Such a term e denotes a language $[\![e]\!]$, defined in the usual way:

$$[\![0]\!] := \emptyset, \qquad [\![1]\!] := \{\varepsilon\}\,, \quad [\![e \cdot f]\!] := \{uv \mid u \in [\![e]\!] \wedge v \in [\![f]\!]\}\,,$$
$$[\![e + f]\!] := [\![e]\!] \cup [\![f]\!]\,, \quad [\![l]\!] := \{l\}\,, \qquad [\![e^\star]\!] := [\![e]\!]^\star = \{u_1 \dots u_n \mid n \in \mathbb{N} \wedge \forall i, u_i \in [\![e]\!]\}\,.$$

2.1 Nominal sets

We fix an infinite set $\mathbb{A}$ of atoms (also called names), and write $\mathfrak{S}_{\mathbb{A}}$ the set of finitely supported permutations over $\mathbb{A}$. These are bijections π such that there is a finite set $\bar{a} \subseteq \mathbb{A}$ such that $a \notin \bar{a} \Rightarrow \pi(a) = a$. In the following we let $a, b, \dots$ range over $\mathbb{A}$ and $\bar{a}, \bar{b}, \dots$ range over finite sets of atoms. The inverse of a permutation π is written π^{-1} . The permutation exchanging a and b, and leaving every other name unchanged, is written $(a\ b)$. We say that a permutation π fixes a finite set $\bar{a} \subseteq \mathbb{A}$, written $\pi \perp \bar{a}$, when $\forall a \in \bar{a},\ \pi(a) = a$.

A set X is called nominal if it can be equipped with two functions, respectively action $- \cdot - : \mathfrak{S}_{\mathbb{A}} \times X \to X$ and support $\mathbf{supp}(-) : X \to \mathcal{P}_f(\mathbb{A})$, satisfying $\forall x \in X,\ \forall \pi, \pi' \in \mathfrak{S}_{\mathbb{A}}$,:

$$\pi \perp \mathbf{supp}(x) \Rightarrow \pi \cdot x = x. \tag{$\dagger_1$}$$
$$\mathbf{supp}(\pi \cdot x) = \left\{a \in \mathbb{A} \;\middle|\; \pi^{-1}(a) \in \mathbf{supp}(x)\right\}. \tag{$\dagger_2$}$$
$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x. \tag{$\dagger_3$}$$

Intuitively, this means that we may replace a name by another in any element of X, and that each element of X only depends on a finite number of names. We say that a permutation π fixes a subset $Y \subseteq X$, also written $\pi \perp Y$ if $\forall y \in Y, \pi \cdot y = y$. This enables use to state $(\dagger_1)$ as $\pi \perp \mathbf{supp}\,(x) \Rightarrow \pi \perp x$. We say that the name a is fresh for x, and write $a \# x$, whenever $a \notin \mathbf{supp}\,(x)$. We will also use the notation $X \restriction_{\bar{a}}$ to mean $\{x \in X \mid \mathbf{supp}\,(x) \subseteq \bar{a}\}$.

▶ Remark 2. In Pitts' book [17] a nominal set is defined as a $\mathfrak{S}_{\mathbb{A}}$-action such that every element has some finite support. From conditions $(\dagger_1)$ and $(\dagger_3)$ we infer that X is a nominal set as in [17]. Furthermore, condition $(\dagger_2)$ enforces that $\mathbf{supp}\,(x)$ is the least finite set that supports x, so our notion of support coincides with the one introduced in [17]. For Coq implementation considerations, we chose to include the support function in the definition.

For the rest of this section, we fix a nominal set X. Given $x, y \in X$, we say that x and y are in the same orbit, written $x \sim_{\mathcal{O}} y$, if there exists $\pi \in \mathfrak{S}_{\mathbb{A}}$ such that $x = \pi \cdot y$. This is an equivalence relation, and its equivalence classes are called orbits. A subset $Y \subseteq X$ is called:

- strict if it has no symmetries, i.e. $(\dagger_1)$ holds as an equivalence: $\pi \perp \mathbf{supp}\,(y) \Leftrightarrow \pi \perp y$;
- equivariant if for every permutation $\pi \in \mathfrak{S}_{\mathbb{A}}$, we have $\pi \cdot Y = Y$, meaning

$$\forall \pi \in \mathfrak{S}_{\mathbb{A}}, \forall y \in X,\ y \in Y \Leftrightarrow \pi \cdot y \in Y;$$

- finitely supported if there is a finite $\bar{a} \subseteq \mathbb{A}$ such that $\pi \perp \bar{a}$ entails $\pi \cdot Y = Y$;
- orbit-finite if Y only intersects finitely many orbits;
- tractable if it is both orbit-finite and finitely supported.

▶ Remark 3. In Bojańczyk [2, 3] terminology, what we call tractable sets are simply called orbit-finite, even though these are sets that are both orbit-finite and finitely supported. We chose a different name to avoid confusion as in other papers orbit-finite sets are not necessarily finitely supported.

In the following, we will use the following results adapted from [3]:

▶ **Lemma 4** (Simple extension of Lemma 3.5 in [3]). *Every tractable set can be expressed as the image of a tractable set of words from $\mathbb{A}^\star$ by some equivariant function.*

▶ **Lemma 5** (Fact 3.6 in [3]). *Tractable sets are closed under finite unions and products, and under finitely supported subsets.*

2.2 Nominal automata

Let Σ be an orbit-finite nominal alphabet. A nominal automaton (NOFA) over Σ is a structure $\mathcal{A} = \langle Q, \Sigma, \Delta, I, F \rangle$ where Q is a tractable state space, $I, F \subseteq Q$ are finitely supported sets of respectively initial and final states, and $\Delta \subseteq Q \times \Sigma \times Q$ is a finitely supported transition relation. This definition corresponds to Bojańczyk, Klin, and Lasota's "orbit-finite automata" [4]. We define the automaton's path relation in the usual way, by saying that $p \xrightarrow{\varepsilon}_{\mathcal{A}} p$ and whenever $p \xrightarrow{w}_{\mathcal{A}} q'$ and $\langle q', x, q \rangle \in \Delta$ then we also have $p \xrightarrow{wx}_{\mathcal{A}} q$. Notice that since Δ is finitely supported, so is the path relation. The language recognised by such an automaton is defined as usual as the set of traces leading from an initial state to a final state:

$$\mathcal{L}_{\mathcal{A}} := \left\{ w \in \Sigma^\star \;\middle|\; \exists \langle q_i, q_f \rangle \in I \times F : q_i \xrightarrow{w}_{\mathcal{A}} q_f \right\}.$$

Nominal regular languages are those recognised by nominal automata.

▶ Remark 6. In the literature, the name "Nominal automaton" is sometimes used to refer to a different class of automata, where the tractability requirement is replaced by orbit-finite and equivariant. These two classes define the same languages: an equivariant automaton is a particular case of a tractable one, and any tractable automaton with support $\bar{a}$ might be seen as an equivariant automaton by replacing the set of atoms $\mathbb{A}$ with the set $\mathbb{A} \setminus \bar{a}$. However, we feel that our approach leads to more intuitive encoding of some natural languages. Consider for instance the language $a \cdot \mathbb{A}^\star$ of words over $\mathbb{A}$ starting with the letter $a \in \mathbb{A}$. This language is not equivariant, therefore to represent it with an equivariant automaton one needs to remove the name a from the set of names, considering instead the alphabet $\mathbb{A}$ as a nominal set over the set of names $\mathbb{A} \setminus \{a\}$. We feel this is a bit counter-intuitive. However, it may be represented by a simple tractable automaton with two states p and q, with p initial, q final, a transition $p \xrightarrow{a} q$, and transitions $q \xrightarrow{b} q$ for every name $b \in \mathbb{A}$.

We will later on rely on the following properties of nominal automata.

▶ **Lemma 7.** *Every nominal automaton is language equivalent to a nominal automaton whose state space is strict.*

▶ **Lemma 8.** *Nominal automata enjoy ε-elimination.*

A nominal automaton is called deterministic (DOFA) if it has a single initial state and its transition relation is a deterministic function, i.e. if we have two transitions $\langle p, x, q\rangle \in \Delta \wedge \langle p, x, q'\rangle \in \Delta$, then $q = q'$. Languages recognised by DOFA form a strict subclass of the regular nominal languages. E.g. the language over $\mathbb{A}$ of words with the last letter distinct from all others is regular nominal but cannot be recognised by a DOFA: intuitively to check for membership one needs to guess what will be the last letter before reading the word. There is also a significant complexity difference: equivalence of DOFA is decidable in polynomial time [15], the corresponding problem for NOFA is undecidable [16].

▶ **Lemma 9.** *Regular languages can be recognised by deterministic nominal automata.*

Proof. Regular languages can be recognised by deterministic finite state automata. Being finite, such automata are also tractable, thus deterministic nominal automata. ◀

2.3 Nominal transductions

We will make intensive use of transductions in this paper. A nominal transducer is a nominal automaton over an alphabet of the shape $(\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})$. For a nominal transducer $\mathcal{T}$, we may define its path relation $-[-/-]\!\to_{\mathcal{T}}$ and the binary relation $\mathcal{R}_{\mathcal{T}}$ it recognises:

$$\frac{}{p \,-[\varepsilon/\varepsilon]\!\to_{\mathcal{T}} p} \qquad \frac{p \,-[w/w']\!\to_{\mathcal{T}} q' \quad \langle q', \langle x, x'\rangle, q\rangle \in \Delta}{p \,-[wx/w'x']\!\to_{\mathcal{T}} q}$$

$$\mathcal{R}_{\mathcal{T}} := \{\langle u, v\rangle \in \Sigma^\star \times \Gamma^\star \mid \exists \langle q_i, q_f\rangle \in I \times F : q_i \,-[u/v]\!\to_{\mathcal{T}} q_f\}\,.$$

A binary relation $R \subseteq \Sigma^\star \times \Gamma^\star$ is called a nominal transduction if it is recognised by some nominal transducer. For a transduction R, we will sometimes see R as either a function $\Sigma^\star \to \mathcal{P}(\Gamma^\star)$ or a function $\mathcal{P}(\Sigma^\star) \to \mathcal{P}(\Gamma^\star)$, writing:

$$u \in \Sigma^\star,\ R(u) := \{v \in \Gamma^\star \mid u\ R\ v\} \qquad L \subseteq \Sigma^\star,\ R(L) := \{v \in \Gamma^\star \mid \exists u \in L : u\ R\ v\}\,.$$

This should not introduce any ambiguity, thanks to typing considerations.

▶ **Lemma 10.** *Nominal regular languages are stable under nominal transductions.*

Proof. Let Σ, Δ be two tractable alphabets, and $\Sigma' := (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})$. Consider a nominal automaton $\mathcal{A} = \langle Q_1, \Sigma, \Delta_1, I_1, F_1 \rangle$ and a nominal transducer $\mathcal{T} = \langle Q_2, \Sigma', \Delta_2, I_2, F_2 \rangle$. We want to show that the language $\mathcal{R}_{\mathcal{T}}(\mathcal{L}_{\mathcal{A}})$ is regular nominal, by building a nominal automaton $\mathcal{T}(\mathcal{A})$ with ε-transitions. Its states are in $Q_1 \times Q_2$, with initial and final states respectively $I_1 \times I_2$ and $F_1 \times F_2$. Its transition relation is given by:

$$\Delta := \{\langle \langle p_1, p_2 \rangle, x, \langle q_1, q_2 \rangle \rangle \mid \exists y : \langle p_1, y, q_1 \rangle \in \Delta_1 \wedge \langle p_2, \langle y, x \rangle, q_2 \rangle \in \Delta_2\}$$
$$\cup \{\langle \langle p_1, p_2 \rangle, x, \langle p_1, q_2 \rangle \rangle \mid \langle p_2, \langle \varepsilon, x \rangle, q_2 \rangle \in \Delta_2\}.$$ ◀

3 Words over an alphabet with binders

For the rest of the paper, we fix an orbit-finite nominal set $\mathbb{X}$ of variables, to represent our alphabet. We consider words built out of variables, left and right binders, respectively written $\langle_a$ and $_a\rangle$. These binders are meant to represent the creation and destruction of names.

We now introduce a notion of α-equivalence for these words. This relation will be a congruence stable under substitution of "local" names: for instance the words $\langle_a\ {}_a\rangle$ and $\langle_b\ {}_b\rangle$ are equivalent. The definitions in this section are straightforward adaptations from [7].

Formally, we define our alphabet by $\mathbb{\Sigma} := \mathbb{X} \cup \{\langle_a \mid a \in \mathbb{A}\} \cup \{{}_a\rangle \mid a \in \mathbb{A}\}$. This alphabet can be endowed with a nominal structure in the obvious way, by setting $\pi \cdot \langle_a = \langle_{\pi(a)}$, $\pi \cdot {}_a\rangle = {}_{\pi(a)}\rangle$, and $\mathbf{supp}(\langle_a) = \mathbf{supp}({}_a\rangle) = \{a\}$. In the following, a word with binders will be an element of $\mathbb{\Sigma}^\star$, that is a finite sequence of letters from the alphabet $\mathbb{\Sigma}$. Words with binders come with a natural nominal structure: the action is defined by applying the alphabet action letter by letter, and the support of a word is the union of the supports of its letters.

Before we define α-equivalence, we need to introduce the notion of *binding power* of a word with binders. The purpose of this notion is to keep track of the occurrences of each name along a word, and enable us to decide whether a particular name is local to the word, and more generally to get a precise account of the way the name is used in the word, from the point of view of the context. The binding monoid $\mathcal{B}$ is defined as the free monoid over the three element set $\{\mathbf{c}, \mathbf{f}, \mathbf{d}\}$, quotiented by the identities: $\mathbf{f} \cdot \mathbf{f} = \mathbf{f}$, $\mathbf{c} \cdot \mathbf{f} = \mathbf{c}$, $\mathbf{f} \cdot \mathbf{d} = \mathbf{d}$, and $\mathbf{c} \cdot \mathbf{d} = \varepsilon$. The letters $\mathbf{c}$, $\mathbf{f}$, $\mathbf{d}$ are meant to represent that a name might be *created*, *free* or *destroyed*. An important property of this monoid is the following, as noticed in [7]: every element of $\mathcal{B}$ can be uniquely represented in the form $\mathbf{d}^m\mathbf{f}^n\mathbf{c}^p$, with $\langle m, n, p \rangle \in \mathbb{N} \times \{0, 1\} \times \mathbb{N}$. We use this remark to define the *size*[1] of a binding element $b \in \mathcal{B}$ as $|\mathbf{d}^m\mathbf{f}^n\mathbf{c}^p| = m + p$.

The binding power of a letter $l \in \mathbb{\Sigma}$ with respect to a name $a \in \mathbb{A}$, written $\mathcal{F}_a(l)$, is computed as follows:

$$\mathcal{F}_a(\langle_b) := \begin{cases} \mathbf{c} & (a = b) \\ \varepsilon & (a \neq b) \end{cases} \qquad \mathcal{F}_a({}_b\rangle) := \begin{cases} \mathbf{d} & (a = b) \\ \varepsilon & (a \neq b) \end{cases} \qquad \mathcal{F}_a(x) := \begin{cases} \mathbf{f} & (a \in \mathbf{supp}(x)) \\ \varepsilon & (a \# x) \end{cases}$$

The function $\mathcal{F}$ may be extended to words naturally as a monoid homomorphism, by setting $\mathcal{F}_a(\varepsilon) = \varepsilon$ and $\mathcal{F}_a(lw) = \mathcal{F}_a(l) \cdot \mathcal{F}_a(w)$. If $\mathcal{F}_a(u) = \mathbf{d}^m\mathbf{f}^n\mathbf{c}^p$ with $n \in \{0, 1\}$, we define $d_a(u) := m$, $f_a(u) := n$, and $c_a(u) := p$. This is well defined thanks to the uniqueness of such representations. This function is equivariant, in the sense that $\mathcal{F}_{\pi(a)}(\pi \cdot u) = \mathcal{F}_a(u)$.

The weight of a word u is the sum of the sizes of its binding powers: $\|u\| := \sum_{a \in \mathbb{A}} |\mathcal{F}_a(u)|$. This sum is finite, since for every name a outside the finite set $\mathbf{supp}(u)$ we know that the binding power of u with respect to a is ε, so $|\mathcal{F}_a(u)| = 0$. The memory of a word u is the maximum weight of a prefix of u, i.e. $\mathbf{m}(u) := \max\{\|v\| \mid \exists w \in \mathbb{\Sigma}^\star : vw = u\}$.

[1] Since the size of a Boolean is constant, we do not count n in the size of $\mathbf{d}^m\mathbf{f}^n\mathbf{c}^p$. This simplifies a number of computations.

Table 1 Alpha-equivalence.

$$\frac{}{\varepsilon =_\alpha \varepsilon}\ (\alpha\varepsilon) \qquad \frac{u =_\alpha v \quad v =_\alpha w}{u =_\alpha w}\ (\alpha\mathbf{t}) \qquad \frac{w_1 =_\alpha w_2}{w_1 l =_\alpha w_2 l}\ (\alpha\mathbf{r}) \qquad \frac{w_1 =_\alpha w_2}{l w_1 =_\alpha l w_2}\ (\alpha\mathbf{l})$$

$$\frac{a \diamond u \quad b \#_\alpha u}{\langle_a u\, {}_a\rangle =_\alpha \langle_b (a\ b) \cdot u\, {}_b\rangle}\ (\alpha\alpha).$$

(a) Definition of Alpha-equivalence.

$$u =_\alpha v \Rightarrow v =_\alpha u \quad (1) \qquad u =_\alpha v \Rightarrow \forall \pi,\ \pi \cdot u =_\alpha \pi \cdot v \quad (4)$$
$$u =_\alpha v \wedge u' =_\alpha v' \Rightarrow uu' =_\alpha vv' \quad (2) \qquad u =_\alpha v \Rightarrow |u| = |v| \quad (5)$$
$$u =_\alpha v \Rightarrow \forall a,\ \mathcal{F}_a(u) = \mathcal{F}_a(v) \quad (3)$$

(b) Properties of Alpha-equivalence.

We use the binding power to define the following: a is balanced in the word w, written $a \diamond w$, if $\mathcal{F}_a(w) \in \{\mathbf{f}, \varepsilon\}$; a is α-fresh in w, written $a \#_\alpha w$, if $\mathcal{F}_a(w) = \varepsilon$; the α-support of w, written $\mathbf{supp}_\alpha(w)$, is the set of names a such that $\mathcal{F}_a(w) \neq \varepsilon$. Notice that $\mathbf{supp}_\alpha(w) \subseteq \mathbf{supp}(w)$. Therefore, we get that $\pi(a) \#_\alpha \pi \cdot u$ if and only if $a \#_\alpha u$, and similarly for $\pi(a) \in \mathbf{supp}_\alpha(\pi \cdot u)$ and $\pi(a) \diamond \pi \cdot u$.

We may now define the α-equivalence relation over words. It is the smallest congruence such that applying the transposition $(a\ b)$ to a word where a and b are α-fresh yields an equivalent word. We give the formal definition of $=_\alpha$ in Table 1a and list some of its properties in Table 1b. The propositions (1) and (2) state that $=_\alpha$ is symmetric and that concatenation is compatible with $=_\alpha$, which together with $(\alpha\varepsilon)$and $(\alpha\mathbf{t})$establishes $=_\alpha$ as a congruence, while (3), (4), and (5) are necessary preservation properties of $=_\alpha$. The proofs of these results follow a simple induction of proof trees.

Note that the deduction system we provided for $=_\alpha$ is not a priori equivalent to the informal description we gave before. However, the correspondence can be proved in the sense that the same relation is obtained if we replace rule $(\alpha\alpha)$ with the following rule:

$$\frac{a \#_\alpha u \quad b \#_\alpha u}{u =_\alpha (a\ b) \cdot u}\ (\alpha\alpha')$$

However, this proof is not straightforward: $(\alpha\alpha')$ obviously implies $(\alpha\alpha)$ (as the latter may be seen as an instance of the former), but the converse direction is more subtle. Unfortunately, this is the most interesting direction, as it is necessary to show that words quotiented by $=_\alpha$ form a nominal set, with the support function $\mathbf{supp}_\alpha()$. This property may however be established using the transducer presented in the next section.

We say that a word u is well-formed when for every decomposition $u = u_1 \langle_a u_2$, we have $c_a(u_1) = 0$. Intuitively, this means that there is no name capture for bound variables. The set of well-formed words is written $\mathcal{WF}$, and we define $\mathbf{wf}(u) := \{v \mid u =_\alpha v \wedge v \in \mathcal{WF}\}$.

4 A transducer for α-equivalence-checking

The problem that arises when trying to prove statements like $(\alpha\alpha)$is that α-equivalence is not preserved in the inductive calls: the property $ux =_\alpha vy$ does not entail $u =_\alpha v$. In this section we introduce a nominal transducer recognising the relation $=_\alpha$. The reachability

relation in this transducer will give us more powerful proof techniques, allowing us to perform proofs by induction. This transducer serves several purposes: it provides us with a decision procedure for $=_\alpha$, enables us to show that $(\alpha\alpha')$ is admissible, and will be used here as a bridge between nominal automata and rational expressions over $\mathbb{\Sigma}$.

4.1 Stacks

The states of this transducer will consist of lists of pairs of atoms, called stacks in the following. Before we define the transducer, we introduce some useful notations. Stacks are generated by the following grammar: $s \in \mathbb{S} ::= [] \mid s :: \langle a, b\rangle$, where a, b range over names. Hence $\mathbb{S}$ is isomorphic to $(\mathbb{A} \times \mathbb{A})^\star$. We will also use the notation $s :: t$ for the concatenation of the two stacks $s, t \in \mathbb{S}$. We write $\mathfrak{p}_1(s)$ for the word over $\mathbb{A}$ obtained by erasing the second components of every pair in s, and symmetrically $\mathfrak{p}_2(s)$ when we erase the first components. For instance $\mathfrak{p}_1([] :: \langle a, b\rangle :: \langle c, d\rangle) = ac$, and $\mathfrak{p}_2([] :: \langle a, b\rangle :: \langle c, d\rangle) = bd$.

Stacks can be endowed with a canonical nominal structure defined by:

$$\pi \cdot [] := [] \qquad \pi \cdot (s :: \langle a, b\rangle) := \pi \cdot s :: \langle \pi(a), \pi(b)\rangle$$
$$\mathbf{supp}([]) := \emptyset \qquad \mathbf{supp}(s :: \langle a, b\rangle) := \mathbf{supp}(s) \cup \{a, b\}.$$

Note that $\mathbf{supp}(s) = \mathbf{supp}(\mathfrak{p}_1(s)) \cup \mathbf{supp}(\mathfrak{p}_2(s)) = \lfloor \mathfrak{p}_1(s) \rfloor \cup \lfloor \mathfrak{p}_2(s) \rfloor$.

The pivotal notions for stacks are the validates predicate and the pop function. We say that a stack s validates the pair $\langle a, b\rangle$, written $s \models \langle a, b\rangle$, when either $a = b$ and $a \# s$, or s can be decomposed as $s = s' :: \langle a, b\rangle :: s''$ in such a way that $a \notin \lfloor \mathfrak{p}_1(s'') \rfloor$ and $b \notin \lfloor \mathfrak{p}_2(s'') \rfloor$. When s validates $\langle a, b\rangle$, we may pop the pair from s, yielding the stack $s \ominus \langle a, b\rangle$ defined by:

$$\frac{a \notin \mathbf{supp}(s)}{s \ominus \langle a, a\rangle := s} \qquad \frac{a \notin \lfloor \mathfrak{p}_1(s') \rfloor \quad b \notin \lfloor \mathfrak{p}_2(s') \rfloor}{(s :: \langle a, b\rangle :: s') \ominus \langle a, b\rangle := s :: s'}.$$

4.2 Equivalence transducer

We now define the equivalence transducer $\mathcal{T}_\alpha$, recognising $=_\alpha$. Strictly speaking, this will not be a nominal transducer, as we will discuss later on. Its state space is $\mathbb{S}$, with initial state $[]$, and the set of accepting states $\mathbb{S}^{acc}$ consists of all stacks s containing only reflexive pairs, i.e. such that $\mathfrak{p}_1(s) = \mathfrak{p}_2(s)$. The transition relation $-[-/-]\to_{\mathcal{T}_\alpha}$ is defined by:

$$\begin{array}{rcl} & & s \,-[\langle_a / \langle_b]\to_{\mathcal{T}_\alpha}\, s :: \langle a, b\rangle \\ s \models \langle a, b\rangle & \Rightarrow & s \,-[\,_a\rangle / \,_b\rangle]\to_{\mathcal{T}_\alpha}\, s \ominus \langle a, b\rangle \\ \forall a \in \mathbf{supp}(x),\ s \models \langle a, \pi(a)\rangle & \Rightarrow & s \,-[x/\pi \cdot x]\to_{\mathcal{T}_\alpha}\, s \end{array}$$

Note that this relation is functional, in the sense that for every triple $\langle s, l, l'\rangle \in \mathbb{S} \times \mathbb{\Sigma} \times \mathbb{\Sigma}$ there exists at most one stack s' such that $s \,-[l/l']\to_{\mathcal{T}_\alpha}\, s'$. This transducer over an infinite state space is equivariant, as one can easily check that $s \,-[u/v]\to_{\mathcal{T}_\alpha}\, s'$ entails $\pi \cdot s \,-[\pi \cdot u/\pi \cdot v]\to_{\mathcal{T}_\alpha}\, \pi \cdot s'$. However, it is not orbit finite. This seems to be unavoidable since there are infinitely many α-equivalence classes (in particular, words of different length cannot be equivalent).

▶ **Theorem 11.** *The relation $\mathcal{R}_{\mathcal{T}_\alpha}$ is exactly $=_\alpha$.*

The full proof has been done in Coq. The following technical lemma allows one to relate the binding power of a word with the stack contents:

▶ **Lemma 12.** *Whenever* $s \xrightarrow{-[u/v]\to_{\mathcal{T}_\alpha}} s'$ *the following identities hold:*

$$|\mathfrak{p}_1(s')|_a = (|\mathfrak{p}_1(s)|_a \dot{-} d_a(u)) + c_a(u) \qquad |\mathfrak{p}_2(s')|_a = (|\mathfrak{p}_2(s)|_a \dot{-} d_a(v)) + c_a(v).$$

(Where $\dot{-}$ *is the truncated subtraction.)*

This lemma has the following corollaries:

▶ **Corollary 13.** *If* $[] \xrightarrow{-[u/v]\to_{\mathcal{T}_\alpha}} s \xrightarrow{-[u'/v']\to_{\mathcal{T}_\alpha}} s'$ *then* $|s| \leqslant \mathbf{m}(uu')$.

Proof. By Lemma 12, and since $[] \xrightarrow{-[u/v]\to_{\mathcal{T}_\alpha}} s$, we have $|s| = \sum_a c_a(u) \leqslant \|u\|$. Since $\|u\| \leqslant \mathbf{m}(uu')$, the result follows. ◀

▶ **Corollary 14.** *For any words* u, v *of length* n*, the following are equivalent:*

(i) $u =_\alpha v$ *and* $v \in \mathcal{WF}$*;*

(ii) *there are stacks* $s_0 \dots s_n$ *such that* $s_0 = []$*,* $s_n \in \mathbb{S}^{acc}$*, for every index* $0 \leqslant i < n$ *we have* $s_i \xrightarrow{-[u_{i+1}/v_{i+1}]\to_{\mathcal{T}_\alpha}} s_{i+1}$*, and for any index* $0 \leqslant i \leqslant n$ *and name* a *we have* $|\mathfrak{p}_2(s_i)|_a \leqslant 1$.

These results allow us to show that the following are nominal transductions:

$$=_\alpha^{\leqslant n} := \{\langle u, v\rangle \mid u =_\alpha v \wedge \mathbf{m}(u) \leqslant n\}$$
$$\mathbf{wf}^n := \{\langle u, v\rangle \mid u =_\alpha v \wedge \mathbf{m}(u) \leqslant n \wedge v \in \mathcal{WF}\}.$$

▶ **Theorem 15.** *For any* $n \in \mathbb{N}$*, both* $=_\alpha^{\leqslant n}$ *and* $\mathbf{wf}^n$ *are nominal transductions.*

Proof. Thanks to Corollary 13, we know that $=_\alpha^{\leqslant n}$ is recognised by $\mathcal{T}_\alpha$ restricted to states $\mathbb{S}^{\leqslant n}$, made up of stacks of length less that n. This is a tractable set, by Lemma 5. Combined with Corollary 14, this proves that $\mathbf{wf}^n$ is recognised by $\mathcal{T}_\alpha$ restricted to stacks such that $|s| \leqslant n$ and $\forall a, |\mathfrak{p}_2(s)|_a \leqslant 1$. This set of stacks being an equivariant subset of $\mathbb{S}^{\leqslant n}$, by Lemma 5 it is also tractable. ◀

5 Memory-finite rational languages

In this section we consider regular languages over $\mathbb{\Sigma}$, i.e. languages $[\![e]\!]$ for some $e \in \mathrm{Rat}\langle\mathbb{\Sigma}\rangle$. We may lift α-equivalence to languages by first defining the α-closure of a language L as:

$$L^\alpha := \{u \in \mathbb{\Sigma}^\star \mid \exists v \in L,\ u =_\alpha v\}.$$

Now we say that two languages are equivalent if their α-closures are equal.

We lift the support function from $\mathbb{\Sigma}$ to $\mathrm{Rat}\langle\mathbb{\Sigma}\rangle$ in the canonical way: for letters in $\mathbb{\Sigma}$ we use the $\mathbf{supp}(-)$ function from the nominal structure of the alphabet, the support of 0 and 1 is the empty set, the support of $e^\star$ is that of e and the support of both $e + f$ and $e \cdot f$ is $\mathbf{supp}(e) \cup \mathbf{supp}(f)$. This definition is an over approximation of the pointwise lifting of the support function on words: indeed $\bigcup_{u \in [\![e]\!]} \mathbf{supp}(u) \subseteq \mathbf{supp}(e)$. Note that $\mathbf{supp}(e)$ is always finite, and supports $[\![e]\!]$ in the sense that whenever $\pi \perp \mathbf{supp}(e)$, we have $\pi \cdot [\![e]\!] = [\![e]\!]$.

A language $L \subseteq \mathbb{\Sigma}^\star$ is called memory-finite if there exists a bound N such that $\forall u \in L, \mathbf{m}(u) \leqslant N$. A rational expression is memory-finite if its language is memory-finite.

▶ **Lemma 16.** *For any rational expression* e*, the following are equivalent:*

(i) e *is memory-finite;*

(ii) *the set* $\{\mathcal{F}_a(u) \mid u \in [\![e]\!],\ a \in \mathbb{A}\}$ *is finite;*

(iii) $\forall u \in [\![e]\!], \mathbf{m}(u) \leqslant 2 \times |e|$.

(Where $|e|$ *is the number of occurrences of letters in* e*.)*

This lemma was proved in Coq. The following result is of independent interest:

▶ **Theorem 17.** *If e is memory-finite, then $[\![e]\!]^\alpha$ is recognisable by DOFA.*

Proof. Let N be the memory of $[\![e]\!]$. By definition, this means that $[\![e]\!]^\alpha$ is equal to the language $=_\alpha^{\leqslant N}([\![e]\!])$. However, the automaton built by applying the construction from Lemma 10 does not yield a deterministic automaton, even if the input automaton is deterministic. Fortunately, in the present case we can determinise the resulting automaton. To do so, we will rely on the following technical result about $\mathcal{T}_\alpha$, which was established using Coq: for every word $u \in \mathbb{\Sigma}^\star$, there is a word $tr(u) \in \mathbb{A}^\star$ such that for any stack s and word v:

$$[] -[u/v]\to_{\mathcal{T}} s \Rightarrow \mathfrak{p}_1(s) = tr(u) \qquad [] -[v/u]\to_{\mathcal{T}} s \Rightarrow \mathfrak{p}_2(s) = tr(u).$$

Notice that this implies that $\mathbf{supp}(tr(u)) \subseteq \mathbf{supp}(u)$: indeed since $u =_\alpha u$ there is a stack s such that $[] -[u/u]\to_{\mathcal{T}_\alpha} s$, so $tr(u) = \mathfrak{p}_1(s)$, and according to Lemma 12 whenever $a \in \mathfrak{p}_1(s)$ we have $c_a(u) \neq 0$ which implies $a \in \mathbf{supp}(u)$.

Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ be some deterministic finite-state automaton for $[\![e]\!]$, with $\Sigma \in \mathcal{P}_f(\mathbb{\Sigma})$. We write $\bar{a}$ for the finite set of names mentioned in the finite alphabet Σ: $\bar{a} := \bigcup_{l\in\Sigma} \mathbf{supp}(l) \subseteq \mathbf{supp}(e)$. Notice that this means that $\pi \perp \bar{a} \Rightarrow \pi \perp \Sigma^\star$. Without loss of generality, we assume that δ is a partial function $Q \times \Sigma \to Q$ and that $\mathcal{A}$ has no sink-state: for any state $q \in Q$, there exists a word $u \in \Sigma^\star$ such that $\delta(q,u) \in F$. If we look back at the proof of Lemma 10, we see that the states in the automaton we get for $=_\alpha^{\leqslant N}(\mathcal{A})$ are pairs of a state from Q and a stack from $\mathbb{S}^{\leqslant N} := \left(\mathbb{A}^2\right)^{\leqslant N}$. Now, let us do the standard powerset construction on this automaton: we get an automaton $\mathcal{A}' := \langle Q', \mathbb{\Sigma}, \delta', q_0', F'\rangle$ where:

$$Q' = \mathcal{P}\left(Q \times \mathbb{S}^{\leqslant N}\right); \qquad q_0' = \{\langle q_0, []\rangle\}; \qquad F' = \{\bar{q} \in Q' \mid \bar{q} \cap (F \times \mathbb{S}^{acc}) \neq \emptyset\};$$
$$\delta'(\bar{q}, l) = \{\langle q', s'\rangle \mid \exists \langle q, s\rangle \in \bar{q}, \exists l' \in \Sigma : q' = \delta(q,l) \wedge s -[l'/l]\to_{\mathcal{T}_\alpha} s'\}.$$

Unfortunately, the state space Q' is not tractable, since it is not orbit-finite. However, as we will now prove, the subset of reachable states is tractable. Therefore if we restrict $\mathcal{A}'$ to its reachable part we get a language-equivalent DOFA. A state $\bar{q} \in Q'$ is reachable if there exists a word v such that $\delta'(q_0', v) = \bar{q}$. By unfolding the definitions, we can see that $\bar{q}$ is reachable by the word v when the following equivalence is satisfied:

$$\forall q, s : \langle q, s\rangle \in \bar{q} \Leftrightarrow \exists u : q = \delta(q_0, u) \wedge [] -[u/v]\to_{\mathcal{T}_\alpha} s.$$

This implies that $\forall \langle q, s\rangle \in \bar{q}$, $\mathfrak{p}_2(s) = tr(v)$, and $\mathfrak{p}_1(s) = tr(u)$ for some $u \in \mathbf{pref}([\![e]\!])$. This second condition tells us that $\mathfrak{p}_1(s) \in \bar{a}^{\leqslant N}$ which is a finite set. Hence the set of reachable states is contained (modulo isomorphism) in the set: $\mathcal{Q} := \mathcal{P}\left(Q \times \bar{a}^{\leqslant N}\right) \times \mathbb{A}^{\leqslant N}$. This set being the product of a finite set with a tractable one, it is tractable. Notice that the set of reachable states is supported by the finite set $\bar{a}$: indeed if $\pi \perp \bar{a}$, then we already know that $\pi \perp \Sigma^\star$ so if $\bar{q}$ is reachable by the word v, then $\pi \cdot \bar{q}$ is reachable by $\pi \cdot v$ since:

$$\begin{aligned}\langle q, s\rangle \in \pi \cdot \bar{q} \Leftrightarrow \left\langle q, \pi^{-1} \cdot s\right\rangle \in \bar{q} &\Leftrightarrow \exists u : q = \delta(q_0, u) \wedge [] -[u/v]\to_{\mathcal{T}_\alpha} \pi^{-1} \cdot s \\ &\Leftrightarrow \exists u : q = \delta(q_0, u) \wedge \pi \cdot [] -[\pi \cdot u/\pi \cdot v]\to_{\mathcal{T}_\alpha} s \\ &\Leftrightarrow \exists u : q = \delta(q_0, u) \wedge [] -[u/\pi \cdot v]\to_{\mathcal{T}_\alpha} s.\end{aligned}$$

We conclude that the set of reachable states is tractable by applying Lemma 5, which tells us that a finitely supported subset of a tractable set is tractable. ◀

We may use expressions over $\mathbb{\Sigma}$ to generate languages over $\mathbb{X}$ as follows: the language generated by a term $e \in \mathrm{Rat}\langle\mathbb{\Sigma}\rangle$, written $(\!|e|\!)$, is the set of words obtained by erasing the brackets from the well-formed words from $[\![e]\!]^\alpha$. In other words, if we denote by η the monoid homomorphism defined by $\eta(\langle_a) = \eta(\,_a\rangle) = \varepsilon$ and $\eta(x) = x$, we have $(\!|e|\!) := \eta(\mathbf{wf}([\![e]\!]))$.

6 Kleene Theorem

In this section, we show that regular nominal languages over $\mathbb{X}$ are exactly those generated by memory-finite rational expressions. To that end, we call a language L rational if there is some memory-finite expression e such that $L = (\!|e|\!)$. One direction is immediate:

▶ **Lemma 18.** *For any memory-finite expression e, $(\!|e|\!)$ is regular nominal.*

Proof. Since e is memory-finite, according to Lemma 16, every word in e has memory less than $2 \times |e|$. Therefore, $\mathbf{wf}\left([\![e]\!]\right) = \mathbf{wf}^{2\times|e|}\left([\![e]\!]\right)$. By the classic Kleene theorem $[\![e]\!]$ is regular and thanks to Theorem 15 we know that $\mathbf{wf}^{2\times|e|}$ is a nominal transduction. Since we may also see easily that η is a nominal transduction, the statement follows from Lemma 10. ◀

We now show that nominal regular languages are rational. We fix a nominal automaton $\mathcal{A} = \langle Q, \mathbb{X}, \Delta, I, F\rangle$, and assume without loss of generality that its state space is strict, equivariant and orbit-finite. We also fix a finite set $\bar{\mathbf{a}}_0 \subseteq \mathbb{A}$ that supports I, F and Δ. As a first step, we will find a finite sub-automaton of $\mathcal{A}$ that is "large enough" to describe the language of $\mathcal{A}$. We do this by picking a finite set $\bar{\mathbf{a}} \subseteq \mathbb{A}$ such that:

$$\forall \alpha \in I \cup F \cup \Delta, \exists \beta \in I \cup F \cup \Delta : \mathbf{supp}\left(\beta\right) \subseteq \bar{\mathbf{a}} \wedge \exists \pi : \pi \perp \bar{\mathbf{a}}_0 \wedge \pi \cdot \beta = \alpha.$$

Such a set always exists: we just need to pick a representative per orbit, and take the union of their supports. As a shorthand, we write $\mathfrak{S}_0$ for the set of permutations over $\mathbb{A} \setminus \bar{\mathbf{a}}_0$, i.e. the permutations $\pi \in \mathfrak{S}_{\mathbb{A}}$ such that π fixes $\bar{\mathbf{a}}_0$. We then define the finite automaton $\mathcal{A} \restriction_{\bar{\mathbf{a}}} := \langle Q \restriction_{\bar{\mathbf{a}}}, \mathbb{X} \restriction_{\bar{\mathbf{a}}}, \Delta \restriction_{\bar{\mathbf{a}}}, I \restriction_{\bar{\mathbf{a}}}, F \restriction_{\bar{\mathbf{a}}}\rangle$. We can relate the runs of $\mathcal{A} \restriction_{\bar{\mathbf{a}}}$ to those in $\mathcal{A}$ as follows.

▶ **Lemma 19.** *For any letters $(x_i)_{1,\dots,n}$ and any states $(q_i)_{0,\dots,n}$, t.f.a.e.:*

(i) *there is a run $p_0 \xrightarrow{x_1}_{\mathcal{A}} p_1 \dots \xrightarrow{x_n}_{\mathcal{A}} p_n$*

(ii) *there is a run $q_0 \xrightarrow{y_1}_{\mathcal{A}\restriction_{\bar{\mathbf{a}}}} q_1 \cdots \xrightarrow{y_n}_{\mathcal{A}\restriction_{\bar{\mathbf{a}}}} q_n$ and a sequence $(\pi_i)_{0,\dots,n}$ from $\mathfrak{S}_0$ such that $\pi_0 \cdot q_0 = p_0$ and $\forall i > 0$ we have $\pi_i \cdot \langle q_{i-1}, y_i, q_i\rangle = \langle p_{i-1}, x_i, p_i\rangle$.*

We now define a finite automaton $\mathcal{A}'$ over the alphabet $\mathbb{\Sigma} \restriction_{\bar{\mathbf{a}}}^{\star}$. The state space of this automaton will be $Q' := Q \restriction_{\bar{\mathbf{a}}} \cup \{q_0, q_f\}$, with q_0 and q_f fresh states, respectively the initial and final states. We build its transitions as follows:

1. we have $q_0 \xrightarrow{\langle a_1 \cdots \langle a_n}$ $q \in \Delta'$ for any $q \in I \restriction_{\bar{\mathbf{a}}}$, and any word $a_1 \dots a_n \in \left(\bar{\mathbf{a}}_0 \cup \mathbf{supp}\left(q\right)\right)^{\bowtie}$;
2. we have $q \xrightarrow{a_1 \rangle \cdots a_n \rangle}$ $q_f \in \Delta'$ for any $q \in F \restriction_{\bar{\mathbf{a}}}$, and any word $a_1 \dots a_n \in \left(\mathbf{supp}\left(q\right) \setminus \bar{\mathbf{a}}_0\right)^{\bowtie}$;
3. we have $p \xrightarrow{\langle a_1 \cdots \langle a_n\, x\, b_1 \rangle \cdots b_m \rangle}$ $q \in \Delta'$ for every transition $p \xrightarrow{x}_{\mathcal{A}\restriction_{\bar{\mathbf{a}}}} q$ and any pair of words:

$$a_1 \dots a_n \in \left(\left(\mathbf{supp}\left(q\right) \cup \mathbf{supp}\left(x\right)\right) \setminus \left(\mathbf{supp}\left(p\right) \cup \bar{\mathbf{a}}_0\right)\right)^{\bowtie}$$
$$b_1 \dots b_m \in \left(\left(\mathbf{supp}\left(p\right) \cup \mathbf{supp}\left(x\right)\right) \setminus \left(\mathbf{supp}\left(q\right) \cup \bar{\mathbf{a}}_0\right)\right)^{\bowtie}.$$

Since we have only a finite number of transitions, we know that this automaton may be transformed into a finite state automaton over $\mathbb{\Sigma} \restriction_{\bar{\mathbf{a}}}$, therefore thanks to Kleene's theorem there is a rational expression $e \in \mathrm{Rat}\,\langle \mathbb{\Sigma} \rangle$ such that $[\![e]\!] = \mathcal{L}_{\mathcal{A}'}$. We now need to check that e is memory-finite and that $(\!|e|\!) = \mathcal{L}_{\mathcal{A}}$. For the first property, we show the following lemma:

▶ **Lemma 20.** *For every run $q_0 \xrightarrow{w}_{\mathcal{A}'} q \in Q \restriction_{\bar{\mathbf{a}}}$, the word $w \in \mathcal{WF}$, $\mathbf{m}\left(w\right) \leqslant \#\bar{\mathbf{a}}$ and either $a \in \mathbf{supp}\left(q\right) \cup \bar{\mathbf{a}}_0$ and $\mathcal{F}_a\left(w\right) = \mathbf{c}$, or $a \notin \mathbf{supp}\left(q\right) \cup \bar{\mathbf{a}}_0$ and $\mathcal{F}_a\left(w\right) = \varepsilon$.*

This entails that $[\![e]\!] \subseteq \mathcal{WF}$ and $\mathbf{m}\left(e\right) \leqslant \#\bar{\mathbf{a}}$. Lemma 20 will also serve in the next proof.

▶ **Lemma 21.** *For any $w \in \mathbb{\Sigma}^\star$, the word w belongs to $\mathbf{wf}\left(\mathcal{L}_{\mathcal{A}'}\right)$ if and only if there is a sequence of permutations $\pi_0 \dots \pi_{n+1} \in \mathfrak{S}_0$ and a run $q_0 \xrightarrow{u_0}_{\mathcal{A}'} q_1 \xrightarrow{u_1}_{\mathcal{A}'} \cdots \xrightarrow{u_n}_{\mathcal{A}'} q_{n+1} \xrightarrow{u_{n+1}}_{\mathcal{A}'} q_f$ such that $w = (\pi_0 \cdot u_0) \dots (\pi_{n+1} \cdot u_{n+1})$ and $\forall 0 < i \leqslant n,\ \pi_{i-1} \cdot q_i = \pi_i \cdot q_i$.*

From Lemmas 19 and 21 it is not hard to see that our construction is correct, thus proving that every regular nominal language is rational.

▶ **Theorem 1** (Kleene Theorem)**.** *Let L be a nominal language. The following are equivalent:*

(i) *L is rational, that is $L = (\!|e|\!)$ for some memory-finite regular expression e.*

(ii) *L is regular nominal, that is recognisable by a NOFA.*

7 Related work

Schröder at al.'s *regular bar-expressions* [18] enjoy a Kleene-like theorem. Regular bar-expressions add an operator $|_a$ to the alphabet, intuitively writing an a on the right-hand side of the bar, and hiding it from the left-hand side. These expressions are equipped with two semantics, called "local" and "global" freshness. Under "local" freshness, the class of automata represented by these expressions is a strict subset of the class of nominal automata, where no name may be guessed (i.e. for every transition $p \xrightarrow{x} q$ we have $\mathbf{supp}\,(q) \subseteq \mathbf{supp}\,(p) \cup \mathbf{supp}\,(x)$), and where a policy of "name dropping" is enforced: a name may be in the support of a state only if it will appear later. For instance, this precludes recognising the languages L_2 and L_5 from the introduction. Under "global" freshness however the situation is more contrasted. With this semantics, the expressive power of bar-expressions is incomparable with that of memory-finite expressions. Indeed, they can denote the language of words where all the letters are different by $|a^\star$, but cannot denote $L_3 := (\!|\langle_a\, a\, {}_a\rangle^\star|\!) = \mathbb{A}^\star$. However if we drop the memory-finite requirement, one can translate bar-expressions into regular expressions over $\mathbb{\Sigma}$ by replacing every occurrence of $|a$ with $\langle_a\, a$ and suffixing the expression with $\left(\sum_{a \in \mathbf{supp}(e)} {}_a\rangle\right)^\star$. For instance the term $|a^\star$ is sent to the expression $(\langle_a\, a)^\star\ {}_a\rangle^\star$. In this case, our well-formed predicate corresponds to the *clean* predicate used to define the global freshness semantics, and this transformation preserves languages. This means that unrestricted expressions with brackets are strictly more expressive than bar-expressions.

In a study of Nominal Kleene Algebra [11, 10, 6], *NKA expressions* were introduced, and half a Kleene theorem for NOFA was proved. These expressions feature a unary $\nu_a(e)$ operator to make a name a local to an expression e. These expressions do not allow the interleaving of scopes, thus failing to capture languages such as 5 from the introduction.

Kurz et al. [13] considered regular expressions with binders. However, their framework only accounts for well nested brackets, thus not covering many of the languages we consider. They present a Kleene theorem for history-dependent automata that incorporates a bound on the nesting depth of binding, rejecting words that exceed this depth, which is the analogue restriction at the automaton level of our memory-finiteness property at the language level. It is unclear whether HD-automata could be generalised to accommodate interleaving of scopes.

On the other hand Libkin and Vrgoč's *regular expressions with memory* [14] enjoy a full Kleene theorem with register automata. Since register automata and nominal automata are equi-expressive, this means that regular expressions with memory are as expressive as our memory-finite expressions. They are however quite different in style. The point of view they choose is that of data words: they assume a finite alphabet Σ and an infinite set of data values $\mathcal{D}$, and consider languages over the alphabet $\Sigma \times \mathcal{D}$, i.e. each letter carries a data value. The key feature of their syntax is to use annotation on letters. They fix a number of variables $x_1 \dots x_k$, and use regular expressions over an alphabet made of elements of the shape $a[c]{\downarrow}I$ where a is a letter from Σ, I is a subset of the variables, and c is a boolean

formula that may use atomic predicates $x_i^{=}$ and $x_i^{\neq}$. These expressions are then interpreted as ternary relations, linking two k-tuples of data values with data words. In effect, this amounts to simulating the run of a register automaton where the k-tuples of data values represent the content of the registers.

References

1 Michal Bielecki, Jan Hidders, Jan Paredaens, Jerzy Tyszkiewicz, and Jan Van den Bussche. Navigating with a Browser. In *ICALP*, pages 764–775, 2002. doi:10.1007/3-540-45465-9_65.

2 Mikołaj Bojańczyk. Nominal Monoids. *Theory of Computing Systems*, 53(2):194–222, 2013. doi:10.1007/s00224-013-9464-1.

3 Mikołaj Bojańczyk. Slightly Infinite Sets. A draft of a book, 2017. URL: https://www.mimuw.edu.pl/~bojan/paper/atom-book.

4 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3):1–44, 2014. doi:10.2168/LMCS-10(3:4)2014.

5 Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A Robust Class of Data Languages and an Application to Learning. *Logical Methods in Computer Science*, 10, 2014. doi:10.2168/LMCS-10(4:19)2014.

6 Paul Brunet and Damien Pous. A Formal Exploration of Nominal Kleene Algebra. In *MFCS*, 2016. doi:10.4230/LIPIcs.MFCS.2016.22.

7 Jamie Gabbay, Dan R. Ghica, and Daniela Petrişan. Leaving the Nest: Nominal Techniques for Variables with Interleaving Scopes. In *CSL*, volume 41, 2015. doi:10.4230/LIPIcs.CSL.2015.374.

8 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

9 Michael Kaminski and Tony Tan. Regular Expressions for Languages over Infinite Alphabets. In *Computing and Combinatorics*, 2004. doi:10.1007/978-3-540-27798-9_20.

10 Dexter Kozen, Konstantinos Mamouras, and Alexandra Silva. Completeness and Incompleteness in Nominal Kleene Algebra. In *RAMiCS*, 2015. doi:10.1007/978-3-319-24704-5_4.

11 Dexter Kozen, Konstantinos Mamouras, Alexandra Silva, and Daniela Petrişan. Nominal Kleene Coalgebra. In *ICALP*, volume 9135, pages 290–302, 2015. doi:10.1007/978-3-662-47666-6.

12 Alexander Kurz, Tomoyuki Suzuki, and Emilio Tuosto. A Characterisation of Languages on Infinite Alphabets with Nominal Regular Expressions. In *TCS*, pages 193–208, 2012.

13 Alexander Kurz, Tomoyuki Suzuki, and Emilio Tuosto. On Nominal Regular Languages with Binders. In *FoSSaCS*, pages 255–269, 2012.

14 Leonid Libkin, Tony Tan, and Domagoj Vrgoč. Regular Expressions for Data Scientists. *Journal of Computer and System Sciences*, 81(7):1278–1287, 2015. doi:10.1016/j.jcss.2015.03.005.

15 Andrzej S Murawski, Steven J Ramsay, and Nikos Tzevelekos. Polynomial-Time Equivalence Testing for Deterministic Fresh-Register Automata. In *MFCS*, 2018. doi:10.4230/LIPIcs.MFCS.2018.72.

16 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004. doi:10.1145/1013560.1013562.

17 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, New York, NY, USA, 2013.

18 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal Automata with Name Binding. In *FoSSaCS*, pages 124–142, 2017. doi:10.1007/978-3-662-54458-7_8.

19 Thomas Schwentick. Automata for XML – A survey. *Journal of Computer and System Sciences*, 73(3):289–315, 2007. doi:/10.1016/j.jcss.2006.10.003.