A Kleene theorem for nominal automata*

Paul Brunet[1] and Alexandra Silva[2]

[1] University College London paul@brunet-zamansky.fr
[2] University College London 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 – that identifies a subclass of expressions denoting exactly regular nominal languages.

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 some 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 their 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.

We use in this paper a new class of regular expressions for data languages, originally motivated by applications to program verification, as part of larger

* This work was partially supported by the ERC Starting Grant ProFoundNet (grant code 679127), a Leverhulme Prize (PLP-2016-129) and the EPSRC project EP/R006865/1.

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 may 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:

$L_1 := (\!|\langle_a\, a\ _a\rangle|\!) = \mathbb{A}$: the set of all atoms;
$L_2 := (\!|\langle_a\, \langle_b\, ab\ _b\rangle\ _a\rangle|\!) = \{ab \mid a \neq b\}$: two letter words made of different letters;
$L_3 := (\!|\langle_a\, a\ _a\rangle^\star|\!) = \mathbb{A}^\star$: the set of all words;
$L_4 := (\!|\langle_a\, a \langle_a\, a\ _a\rangle^\star\ _a\rangle|\!) = \{a_1 \dots a_n \mid n > 0, \forall 1 < i, a_i \neq a_1\}$: the set of words such that the first letter is different from all others;
$L_5 := (\!|\langle_a\, \langle_a\, a\ _a\rangle^\star a\ _a\rangle|\!) = \{a_1 \dots a_n \mid n > 0, \forall i < n, a_i \neq a_n\}$: the set of words such that the last letter is different from all others;
$L_6 := (\!|\langle_a\, a\, (\langle_b\, b\ _a\rangle \langle_a\, a\ _b\rangle)^\star\, (1 + \langle_b\, b\ _b\rangle)\ _a\rangle|\!) = \{a_1 a_2 \dots a_n \mid n > 0, \forall i, a_i \neq a_{i+1}\}$: the set of non-empty words such that two consecutive letters are different;
$L_7 := (\!|(\langle_a\, x)^\star\, (y\ _a\rangle)^\star|\!) = \{x\} \cup \{x^n y^m \mid n \leqslant m\}$.

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 L_7. 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 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 the expressions, and such expressions generate exactly the class of languages recognisable by NOFAs.

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. All proofs are provided in an appendix.

This paper is part of a larger research program on developing a framework to reason about programs with explicit resource allocation and deallocation. A companion paper describing the algebraic framework of *bracket algebra* and a hierarchy of nominal languages can be found in http://paul.brunet-zamansky.fr/Brackets/, 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, the concatenation of the words u and v is written uv, and $|u|$ is the length of the word u. If w is a word over the alphabet Σ and $x \in \Sigma$, then $|w|_x$ is the number of occurrences of x in w. We sometimes identify words with the set of letters occurring in them, writing $x \in w$ to denote the fact that the letter x appears in w, i.e. $|w|_x \neq 0$. We will extend this analogy by writing $w \approx X$ for $w \in \Sigma^\star$ and $X \subseteq \Sigma$ when $\forall x, (x \in X) \Leftrightarrow (|w|_x \neq 0)$. 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 a finite subset $B \subseteq A$, 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:

$$[\![0]\!] := \emptyset \qquad [\![1]\!] := \{\varepsilon\} \qquad [\![l]\!] := \{l\} \qquad [\![e + f]\!] := [\![e]\!] \cup [\![f]\!]$$

$$[\![e \cdot f]\!] := \{uv \mid u \in [\![e]\!] \wedge v \in [\![f]\!]\} \qquad [\![e^\star]\!] := [\![e]\!]^\star.$$

2.1 Nominal sets

We fix an infinite set $\mathbb{A}$ of names, and write $\mathfrak{S}_{\mathbb{A}}$ the set of finitely supported permutations over $\mathbb{A}$. In the following we let $a, b, \ldots$ range over $\mathbb{A}$ and $\bar{a}, \bar{b}, \ldots$ range over finite sets of atoms. These are bijections π such that there is a finite set $\bar{a} \subseteq \mathbb{A}$ such that $a \notin \bar{a} \Rightarrow \pi(a) = a$. 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 there are functions $- \cdot - : \mathfrak{S}_{\mathbb{A}} \times X \to X$ and $\mathbf{supp}(-) : X \to \mathcal{P}_f(\mathbb{A})$, respectively called the *action* and the *support*, such that $\forall x \in X,\ \forall \pi, \pi' \in \mathfrak{S}_{\mathbb{A}}$, we have:

$$\pi \perp \mathbf{supp}(x) \Rightarrow \pi \cdot x = x. \tag{$\dagger_1$}$$

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

$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x. \tag{$\dagger_3$}$$

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 the variable x, and write $a \# x$, whenever $a \notin \mathbf{supp}(x)$. We will also use the notation $X_{\lfloor \bar{a}}$ to mean $\{x \in X \mid \mathbf{supp}(x) \subseteq \bar{a}\}$.

Remark 2.1. 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 the one introduced in [17]. For `Coq` implementation considerations, we opted for explicitly including the support function in the definition.

Let us fix for the remainder of this section a nominal set X. Given two elements $x, y \in X$, we say that x and y are in the same orbit, written $x \sim_{\mathcal{O}} y$, if there is 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, meaning that axiom $(\dagger_1)$ now holds as an equivalence: $\pi \perp \mathbf{supp}(x) \Leftrightarrow \pi \perp x$;
- *equivariant* if for every permutation $\pi \in \mathfrak{S}_{\mathbb{A}}$, we have $\pi \cdot Y = Y$;
- *finitely supported* if there exists a finite set $\bar{a} \subseteq \mathbb{A}$ such that for any permutation $\pi \in \mathfrak{S}_{\mathbb{A}}$ that fixes $\bar{a}$ we have $\pi \cdot Y = Y$;
- *orbit-finite* if Y only intersects finitely many orbits;
- *tractable* if it is both orbit-finite and finitely supported.

Remark 2.2. In Bojańczyk [2,3] terminology, what we call tractable sets are simply called orbit-finite, even though they are sets that are both orbit-finite and finitely supported. We chose a different name to avoid confusion as in most other papers, as expected, orbit-finite sets are not necessarily finitely supported.

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

Lemma 2.3. *Every tractable set can be expressed as the image of a tractable set of words from $\mathbb{A}^\star$ by some equivariant function.*

Proof. Simple extension of [3, Lemma 3.5]. □

Lemma 2.4. *Tractable sets are closed under finite unions and products, finitely supported subsets and images under finitely supported functions.*

Proof. [3, Fact 3.6]. □

2.2 Nominal automata

Let Σ be an orbit-finite nominal alphabet. A *nominal automaton* over Σ is a structure $\mathscr{A} = \langle Q, \Sigma, \Delta, I, F \rangle$ where:

- Q is a tractable state space;
- $I, F \subseteq Q$ are finitely supported sets of respectively initial and final states;
- $\Delta \subseteq Q \times \Sigma \times Q$ is a finitely supported transition relation.

This definition corresponds to Bojaczyk's "orbit-finite automata" [3]. We define the automaton's path relation in the usual way, by saying that $p \xrightarrow{\varepsilon}_{\mathscr{A}} p$ and whenever $p \xrightarrow{w}_{\mathscr{A}} q'$ and $\langle q', x, q \rangle \in \Delta$ then we also have $p \xrightarrow{wx}_{\mathscr{A}} q$. 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:

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

Nominal regular languages are those recognised by nominal automata.

Remark 2.5. 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 of words over $\mathbb{A}$ where the name $a \in \mathbb{A}$ does not appear. 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, which seems a bit counter-intuitive. However, it may be represented by a simple tractable automaton with a single state q both initial and final and transitions $q \xrightarrow{b} q$ for every name $b \neq a$.

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

Lemma 2.6. *Every nominal automaton is language equivalent to a nominal automaton whose state space is strict.*

Proof. Corollary of Lemma 2.3. □

Lemma 2.7. *Nominal automata enjoy ε-elimination.*

A nominal automaton is called deterministic if it has a single initial state and its transition relation is a partial function, meaning:

$$\forall p, q, q' \in Q, \forall x \in \Sigma,\ \langle p, x, q\rangle \in \Delta \wedge \langle p, x, q'\rangle \in \Delta \Rightarrow q = q'.$$

It is well-known that the languages recognised by deterministic nominal automata form a strict subclass of the regular nominal languages. For instance the language over $\mathbb{A}$ of words where the last letter is distinct from all others is regular nominal but cannot be recognised by a deterministic automaton: 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: while equivalence of deterministic nominal automata is decidable in polynomial time [15], the corresponding problem for non-deterministic automata is undecidable [16].

The following simple remark will be used later:

Lemma 2.8. *Regular languages can be recognised by deterministic nominal automata.*

Proof. We know that 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 $\mathscr{T}$, we may define its path relation $-[-/-]\!\to_{\mathscr{T}}$ and the binary relation $\mathcal{R}_{\mathscr{T}}$ it recognises:

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

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

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, since what we mean will always be clear from typing considerations.

Lemma 2.9. *Nominal regular languages are stable under nominal transductions.*

Proof. Let Σ, Δ be two tractable alphabets, we write $\Sigma' = (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})$. Consider a nominal automaton $\mathscr{A} = \langle Q_1, \Sigma, \Delta_1, I_1, F_1 \rangle$ and a nominal transducer $\mathscr{T} = \langle Q_2, \Sigma', \Delta_2, I_2, F_2 \rangle$. We want to show that the language $\mathcal{R}_{\mathscr{T}}(\mathcal{L}_{\mathscr{A}})$ is regular nominal. We define a nominal automaton $\mathscr{T}(\mathscr{A})$ 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:

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

Correction of this construction can be checked as a matter of routine. □

3 Words over an alphabet with binders

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

In this section we introduce a notion of α-equivalence for these sequences. This relation will be a congruence stable by substitution of "local" names: for instance the sequence $\langle_a\ _a\rangle$ should be equivalent to the sequence $\langle_b\ _b\rangle$. The definitions in this section are straightforward adaptations from [7].

Formally, we define our alphabet by $\mathbb{\Sigma}_{\mathbb{X}}^{\mathbb{A}} := \mathbb{X} \cup \{\langle_a \mid a \in \mathbb{A}\} \cup \{_a\rangle \mid a \in \mathbb{A}\}$. If the sets of atoms and variables are clear from the context, we simply write $\mathbb{\Sigma}$. 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 naturally support a 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*[3] of a binding element $b \in \mathcal{B}$ as $|\mathbf{d}^m\mathbf{f}^n\mathbf{c}^p| = m + p$.

[3] 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.

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 $\not{d}_a(u) := m$, $f_a(u) := n$, and $\not{c}_a(u) := p$. Notice that 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. $\mathbf{m}(u) := \max_{vw=u} \|v\|$.

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 $a \#_\alpha w$, is $\mathcal{F}_a(w) = \varepsilon$; the *$\alpha$-support* of w, written $\mathbf{supp}_\alpha(w)$, is the set of names a such that $\mathcal{F}_a(w) \neq \varepsilon$. Notice that $\mathbf{supp}_\alpha(w) \subseteq \mathbf{supp}(w)$. Using the previous remark, 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. Formally, we define $=_\alpha$ by:

$$\varepsilon =_\alpha \varepsilon \qquad (\alpha\varepsilon)$$

$$u =_\alpha v \ \wedge\ v =_\alpha w \Rightarrow \quad u =_\alpha w \qquad (\alpha\mathbf{t})$$

$$w_1 =_\alpha w_2 \Rightarrow \quad w_1 l =_\alpha w_2 l \qquad (\alpha\mathbf{r})$$

$$w_1 =_\alpha w_2 \Rightarrow \quad l w_1 =_\alpha l w_2 \qquad (\alpha\mathbf{l})$$

$$a \diamond u \ \wedge\ b \#_\alpha u \Rightarrow \quad \langle_a u\, _a\rangle =_\alpha \langle_b (a\ b) \cdot u\, _b\rangle. \qquad (\alpha\alpha)$$

The following statements hold:

$$u =_\alpha v \Rightarrow v =_\alpha u \qquad (1)$$

$$u =_\alpha v \wedge u' =_\alpha v' \Rightarrow uu' =_\alpha vv' \qquad (2)$$

$$u =_\alpha v \Rightarrow \forall a,\ \mathcal{F}_a(u) = \mathcal{F}_a(v) \qquad (3)$$

$$u =_\alpha v \Rightarrow \forall \pi,\ \pi \cdot u =_\alpha \pi \cdot v \qquad (4)$$

$$u =_\alpha v \Rightarrow |u| = |v| \qquad (5)$$

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:

$$a \#_\alpha u \wedge b \#_\alpha u \Rightarrow u =_\alpha (a\ b) \cdot u. \qquad (\alpha\alpha')$$

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.

In the following, we will need the $\mathcal{WF}$ predicate: a word u is called *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. We will use the notation $\mathbf{wf}(u) := \{v \mid u =_\alpha v \wedge v \in \mathcal{WF}\}$.

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 $\mathbf{supp}(s) = \mathbf{supp}(\mathfrak{p}_1(s)) \cup \mathbf{supp}(\mathfrak{p}_2(s)) = \mathfrak{p}_1(s) \cup \mathfrak{p}_2(s)$, the last identity using our shorthand identifying words with the set of letters occurring in them.

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 $a \notin \mathfrak{p}_1(s'')$ and $b \notin \mathfrak{p}_2(s'')$. 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:

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

4.2 Equivalence transducer

We now define the *equivalence transducer* $\mathscr{T}_\alpha$, recognising $=_\alpha$. This will not strictly speaking 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 $-[-/-]\!\to_{\mathscr{T}_\alpha}$ is defined by:

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

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 $s -[l/l']\!\to_{\mathscr{T}_\alpha} s'$. This transducer over an infinite state space is equivariant, as one can easily check that $s -[u/v]\!\to_{\mathscr{T}_\alpha} s'$ entails $\pi \cdot s -[\pi \cdot u/\pi \cdot v]\!\to_{\mathscr{T}_\alpha} \pi \cdot s'$. 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 4.1. *The relation $\mathcal{R}_{\mathscr{T}_\alpha}$ 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 4.2. *Whenever $s -[u/v]\!\to_{\mathscr{T}_\alpha} s'$ 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 4.3. *If $[] -[u/v]\!\to_{\mathscr{T}_\alpha} s -[u'/v']\!\to_{\mathscr{T}_\alpha} s'$ then $|s| \leqslant \mathbf{m}(uu')$.*

Proof. By Lemma 4.2, and since $[] -[u/v]\!\to_{\mathscr{T}_\alpha} s$, we have $|s| = \sum_a c_a(u) \leqslant \|u\|$. Since $\|u\| \leqslant \mathbf{m}(uu')$, the result follows. □

Corollary 4.4. *For any words u, v of length n, the following are equivalent:*

(i) $u =_\alpha v$ *and* $v \in \mathcal{WF}$*;*
(ii) there are stack $s_0 \dots s_n$ *such that* $s_0 = []$, $s_n \in \mathbb{S}^{acc}$*, for every index* $0 \leqslant i < n$ *we have* $s_i \xrightarrow{-[u_{i+1}/v_{i+1}]\to_{\mathscr{T}_\alpha}} s_{i+1}$*, 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 4.5. *For any* $n \in \mathbb{N}$*, both* $=_\alpha^{\leqslant n}$ *and* $\mathbf{wf}^n$ *are nominal transductions.*

Proof. Thanks to Corollary 4.3, we know that $=_\alpha^{\leqslant n}$ is recognised by $\mathscr{T}_\alpha$ restricted to states $\mathbb{S}^{\leqslant n}$, made up of stacks of length less that n. This is a tractable set, by Lemma 2.4. Combined with Corollary 4.4, this proves that $\mathbf{wf}^n$ is recognised by $\mathscr{T}_\alpha$ 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 2.4 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 expression $e \in \mathrm{Rat}\langle\mathbb{\Sigma}\rangle$. We may lift α-equivalence to languages as follows. First, the *α-closure* of a language L is defined 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 5.1. *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`. As a result of independent interest, we state here the following fact:

Theorem 5.2. *For any memory-finite expression* e*,* $[\![e]\!]^\alpha$ *is recognised by some deterministic nominal automaton.*

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 2.9 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 $\mathscr{T}_\alpha$, 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:

$$[] -[u/v]\to_{\mathscr{T}} s \Rightarrow \mathfrak{p}_1(s) = tr(u) \tag{6}$$
$$[] -[v/u]\to_{\mathscr{T}} s \Rightarrow \mathfrak{p}_2(s) = tr(u). \tag{7}$$

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 $[] -[u/u]\to_{\mathscr{T}_\alpha} s$, so $tr(u) = \mathfrak{p}_1(s)$, and according to Lemma 4.2 whenever $a \in \mathfrak{p}_1(s)$ we have $c_a(u) \neq 0$ which implies $a \in \mathbf{supp}(u)$.

Let $\mathscr{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$ 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 $\mathscr{A}$ 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 2.9, we see that the states in the automaton we get for $=_\alpha^{\leqslant N}(\mathscr{A})$ 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 $\mathscr{A}' := \langle Q', \mathbb{\Sigma}, \delta', q_0', F'\rangle$ where:

$$Q' = \mathcal{P}\left(Q \times \mathbb{S}^{\leqslant N}\right); \qquad q_0' = \{\langle q_0, []\rangle\}; \qquad F' = \{\bar{q} \in Q' \mid \bar{q} \cap (F \times \mathbb{S}^{acc}) \neq \emptyset\};$$

$$\delta'(\bar{q}, l) = \{\langle q', s'\rangle \mid \exists \langle q, s\rangle \in \bar{q}, \exists l' \in \Sigma : q' = \delta(q,l) \wedge s -[l'/l]\to_{\mathscr{T}_\alpha} s'\}.$$

Unfortunately, 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 $\mathscr{A}'$ to its reachable part we get a language-equivalent deterministic nominal automaton. 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:

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

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:

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

We conclude that the set of reachable states is tractable by applying Lemma 2.4, which tells us that a finitely supported subset of a tractable set is tractable. □

We may use rational expressions over Σ to generate languages over $\mathbb{X}$ as follows: given a term $e \in \mathrm{Rat}\,\langle\Sigma\rangle$, the language *generated* by e, 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 define η to be the monoid homomorphism defined by $\eta(\langle_a) = \eta(\,_a\rangle) = \varepsilon$ and $\forall x \in \mathbb{X},\ \eta(x) = x$, 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 $L = (\!|e|\!)$.

From what we know so far, one direction is immediate:

Lemma 6.1. *For any memory-finite expression e, $(\!|e|\!)$ is regular nominal.*

Proof. Since e is memory-finite, according to Lemma 5.1, every word in e has memory less than $2 \times |e|$. Therefore, $\mathbf{wf}([\![e]\!]) = \mathbf{wf}^{2\times|e|}([\![e]\!])$. By the classic Kleene theorem $[\![e]\!]$ is regular and thanks to Theorem 4.5 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 2.9. □

We now show that nominal regular languages are rational. We fix a nominal automaton $\mathscr{A} = \langle Q, \mathbb{X}, \Delta, I, F\rangle$, 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 $\mathscr{A}$ that is "large enough" to describe the language of $\mathscr{A}$. We do this by picking a finite set $\bar{\mathbf{a}} \subseteq \mathbb{A}$ such that:

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

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 $\mathscr{A}\!\restriction_{\bar{\mathbf{a}}} := \langle Q\!\restriction_{\bar{\mathbf{a}}}, \mathbb{X}\!\restriction_{\bar{\mathbf{a}}}, \Delta\!\restriction_{\bar{\mathbf{a}}}, I\!\restriction_{\bar{\mathbf{a}}}, F\!\restriction_{\bar{\mathbf{a}}}\rangle$. The runs of this automaton are related to those in $\mathscr{A}$ as follows.

Fact 1. *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* $p_0 \xrightarrow{x_1}_{\mathscr{A}} p_1 \dots \xrightarrow{x_n}_{\mathscr{A}} p_n$ *(ii) there is a run* $q_0 \xrightarrow{y_1}_{\mathscr{A}\lfloor_{\bar{\mathbf{a}}}} q_1 \cdots \xrightarrow{y_n}_{\mathscr{A}\lfloor_{\bar{\mathbf{a}}}} q_n$ *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 $\mathscr{A}'$ over the alphabet $\mathbb{\Sigma}\lfloor_{\bar{\mathbf{a}}}^\star$. The state space of this automaton will be $Q' := Q\lfloor_{\bar{\mathbf{a}}} \cup \{q_0, q_f\}$, with q_0 and q_f fresh states, respectively the initial and final states. We build its transitions as follows:

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

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

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

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

Fact 2. *For every run* $q_0 \xrightarrow{w}_{\mathscr{A}'} q \in Q\lfloor_{\bar{\mathbf{a}}}$*, the word* $w \in \mathcal{WF}$*,* $\mathbf{m}(w) \leqslant \#\bar{\mathbf{a}}$ *and either* $a \in \mathbf{supp}(q) \cup \bar{\mathbf{a}}_0$ *and* $\mathcal{F}_a(w) = \mathbf{c}$*, or* $a \notin \mathbf{supp}(q) \cup \bar{\mathbf{a}}_0$ *and* $\mathcal{F}_a(w) = \varepsilon$*.*

This fact entails that $[\![e]\!] \subseteq \mathcal{WF}$ and $\mathbf{m}(e) \leqslant \#\bar{\mathbf{a}}$. Fact 2 will also serve to prove the following correspondence.

Fact 3. *For any* $w \in \mathbb{\Sigma}^\star$*, the word* w *belongs to* $\mathbf{wf}\left(\mathcal{L}_{\mathscr{A}'}\right)$ *if and only if there is a sequence of permutations* $\pi_0 \dots \pi_{n+1} \in \mathfrak{S}_0$ *and a run*

$$q_0 \xrightarrow{u_0}_{\mathscr{A}'} q_1 \xrightarrow{u_1}_{\mathscr{A}'} \cdots \xrightarrow{u_n}_{\mathscr{A}'} q_{n+1} \xrightarrow{u_{n+1}}_{\mathscr{A}'} q_f$$

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 Facts 1 and 3 it is not hard to see that our construction is correct, thus proving that every regular nominal language is rational.

Theorem 6.2. *The class of regular nominal languages coincides with the class of rational languages.*

7 Related work

Schröder at al.'s *regular bar-expressions* [18] enjoy such a Kleene 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 Σ by replacing every occurrence of $|a$ with $\langle_a\, a$ add to the expression a suffix $\left(\sum_{a\in\mathbf{supp}(e)} {}_a\rangle\right)^\star$. 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 regular 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. Because of this, these expressions do not allow the interleaving of scopes, thus failing to capture languages such as L_5 from the introduction.

Kurz et al. [13] considered regular expressions with binders. However, their framework only accounts for well nested brackets which means they cannot recognise many of the languages we consider. They present a Kleene theorem for history-dependent automata that incorporate a bound on the nesting depth of binding, rejecting words that exceed this depth, which is a restriction at the automaton level to our memory-finiteness property at the language level. It is not clear whether their HD-automata exploration 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 as 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. Bielecki, M., Hidders, J., Paredaens, J., Tyszkiewicz, J., Van den Bussche, J.: Navigating with a browser. In: Automata, Languages and Programming (2002)
2. Bojańczyk, M.: Nominal monoids. Theory of Computing Systems **53**(2), 194–222 (2013). https://doi.org/10.1007/s00224-013-9464-1
3. Bojańczyk, M.: Slightly infinite sets (2017), https://www.mimuw.edu.pl/~bojan/paper/atom-book, a draft of a book
4. Bojańczyk, M., Klin, B., Lasota, S.: Automata theory in nominal sets. Logical Methods in Computer Science **10**(3), 1–44 (2014). https://doi.org/10.2168/LMCS-10(3:4)2014
5. Bollig, B., Habermehl, P., Leucker, M., Monmege, B.: A robust class of data languages and an application to learning. Logical Methods in Computer Science **10** (2014). https://doi.org/10.2168/LMCS-10(4:19)2014
6. Brunet, P., Pous, D.: A Formal Exploration of Nominal Kleene Algebra. In: MFCS (2016). https://doi.org/10.4230/LIPIcs.MFCS.2016.22
7. Gabbay, J., Ghica, D.R., Petrişan, D.: Leaving the Nest: Nominal Techniques for Variables with Interleaving Scopes. In: CSL. vol. 41 (2015). https://doi.org/10.4230/LIPIcs.CSL.2015.374
8. Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science **134**(2), 329 – 363 (1994). https://doi.org/10.1016/0304-3975(94)90242-9
9. Kaminski, M., Tan, T.: Regular expressions for languages over infinite alphabets. In: Computing and Combinatorics (2004). https://doi.org/10.1007/978-3-540-27798-9_20
10. Kozen, D., Mamouras, K., Silva, A.: Completeness and incompleteness in nominal kleene algebra. In: Relational and Algebraic Methods in Computer Science (2015). https://doi.org/10.1007/978-3-319-24704-5_4
11. Kozen, D., Mamouras, K., Silva, A., Petrişan, D.: Nominal Kleene Coalgebra. In: ICALP. vol. 9135, pp. 290–302 (2015). https://doi.org/10.1007/978-3-662-47666-6
12. Kurz, A., Suzuki, T., Tuosto, E.: A characterisation of languages on infinite alphabets with nominal regular expressions. In: Theoretical Computer Science (2012)
13. Kurz, A., Suzuki, T., Tuosto, E.: On nominal regular languages with binders. In: Foundations of Software Science and Computational Structures (2012)
14. Libkin, L., Tan, T., Vrgoč, D.: Regular Expressions for Data Scientists. Journal of Computer and System Sciences **81**(7), 1278–1287 (2015). https://doi.org/10.1016/j.jcss.2015.03.005
15. Murawski, A.S., Ramsay, S.J., Tzevelekos, N.: Polynomial-Time Equivalence Testing for Deterministic Fresh-Register Automata. In: MFCS (2018). https://doi.org/10.4230/LIPIcs.MFCS.2018.72
16. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Transactions on Computational Logic **5**(3), 403–435 (2004). https://doi.org/10.1145/1013560.1013562
17. Pitts, A.M.: Nominal Sets: Names and Symmetry in Computer Science. Cambridge University Press, New York, NY, USA (2013)
18. Schröder, L., Kozen, D., Milius, S., Wißmann, T.: Nominal Automata with Name Binding. In: FoSSaCS. pp. 124–142 (2017). https://doi.org/10.1007/978-3-662-54458-7_8
19. Schwentick, T.: Automata for xmla survey. Journal of Computer and System Sciences **73**(3), 289 – 315 (2007). https://doi.org//10.1016/j.jcss.2006.10.003

A Proofs of section 2

Lemma 2.3. *Every tractable set can be expressed as the image of a tractable set of words from $\mathbb{A}^\star$ by some equivariant function.*

Proof. Let $Y \subseteq X$ be a tractable set. Let I be the set of orbits intersected by Y, i.e. $I = \{[x]_{\sim_\mathcal{O}} \mid x \in Y\} \subseteq X$. By definition of tractability I is finite, let $n = \#I$. We may therefore index the orbits in I with numbers $1 \leqslant i \leqslant n$. For an orbit $o_i \in I$, we define $Y_i = o_i \cap Y$. Hence Y may be expressed as the finite union:

$$Y = \bigcup_{1 \leqslant i \leqslant n} Y_i.$$

Since Y is finitely supported, so are the Y_i. Indeed, $\pi \cdot Y = Y$ entails $\pi \cdot Y_i = Y_i$, so any set supporting Y also supports Y_i.

According to [3, Lemma 3.5], for each Y_i there exist an equivariant function f_i and a word $w_i \in \mathbb{A}^\star$ such that:

$$Y_i = \{\pi \cdot w_i \mid \pi \in \mathfrak{S}_\mathbb{A} : \pi \perp \mathbf{supp}(Y_i)\}.$$

Now, we encode the numbers $1 \leqslant i \leqslant n$ into words as follows: let $a, b \in \mathbb{A}$ be two distinct names, $\underline{i} = a^{n-i+1}b^i$. Each of those words have length $n+1$, and there is an equivariant function $idx : \mathbb{A}^\star \to \{0, \dots, n\}$ such that: $idx\,(\underline{i}) = i$.

We finally define:

$$W_Y := \{\underline{i} w_i \mid 1 \leqslant i \leqslant n\}$$

$$f(w) := \begin{cases} f_1(w) \text{ if } |w| < n+1 \\ f_{idx(w_1 \dots w_{n+1})}\left(w_{n+2} \dots w_{|w|}\right) \text{ otherwise.} \end{cases}$$

□

Lemma 2.7. *Nominal automata enjoy ε-elimination.*

Proof. Let $\mathscr{A} = \langle Q, \mathbb{X}, \Delta, I, F\rangle$ be an automaton with ε-transitions, meaning Δ is a finitely supported subset of $Q \times (\mathbb{X} \cup \{\varepsilon\}) \times Q$.

We may define $\mathscr{A}' = \langle Q, \mathbb{X}, \Delta', I, F'\rangle$ where:

$$\Delta' := \left\{\langle p, x, q\rangle \in Q \times \mathbb{X} \times Q \;\middle|\; \exists p' : \langle p', x, q\rangle \in \Delta \wedge p \xrightarrow{\varepsilon}_\mathscr{A} p'\right\}$$

$$F' := \left\{q \in Q \;\middle|\; \exists q' \in F : q \xrightarrow{\varepsilon}_\mathscr{A} q'\right\}.$$

Clearly this automaton is language-equivalent to $\mathscr{A}$. What we need to show is that F' and Δ' are finitely supported. This is a consequence of the fact that F and Δ were finitely supported. Let $\bar{a}$ be a finite set of names supporting both F and Δ. By definition of the path relation, we have that:

$$\forall \pi \in \mathfrak{S}_\mathbb{A}, \pi \perp \bar{a} \Rightarrow \forall p, w, q : p \xrightarrow{w}_\mathscr{A} q \Leftrightarrow \pi \cdot p \xrightarrow{\pi \cdot w}_\mathscr{A} \pi \cdot q.$$

Therefore, since $\pi \cdot \varepsilon = \varepsilon$, we can deduce that $\bar{a}$ supports both Δ' and F'. □

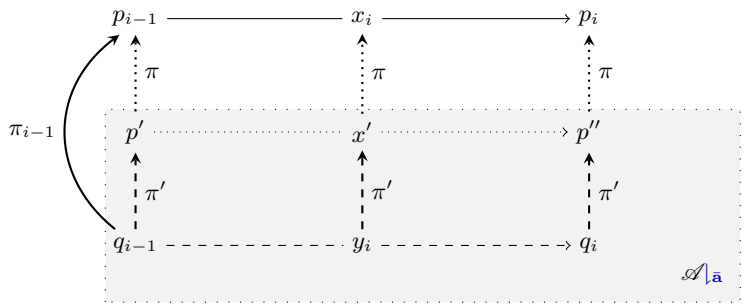

Fig. 1. Illustration of the inductive case for $(i) \Rightarrow (ii)$ of Fact 1.

B Proof of Section 4

Corollary 4.4. *For any words u, v of length n, the following are equivalent:*

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

Proof. By Theorem 4.1, we know that $u =_\alpha v$ if and only if there are stack $s_0 \dots s_n$ such that $s_0 = []$, $s_n \in \mathbb{S}^{acc}$, for every index $0 \leqslant i < n$ we have $s_i \xrightarrow{-[u_{i+1}/v_{i+1}]\to_{\mathscr{T}_\alpha}} s_{i+1}$. We need to show that in this situation, $v \in \mathcal{WF}$ if and only if for any index $0 \leqslant i \leqslant n$ and name a we have $|\mathfrak{p}_2(s_i)|_a \leqslant 1$. Assume v is well-formed, and let $a \in \mathbb{A}$. We prove the property by recursion on i. Since $|[]|_a = 0$, the initialisation is trivial. Now, assume $|\mathfrak{p}_2(s_i)|_a \leqslant 1$. We proceed by case analysis on the letter v_{i+1}.

- If v_{i+1} is a closing bracket or a variable, then s_{i+1} is a subword of s_i (either it is equal to s_i or it was obtained by removing a pair from s_i). In this case $|\mathfrak{p}_2(s_{i+1})|_a \leqslant |\mathfrak{p}_2(s_i)|_a \leqslant 1$.
- If $v_{i+1} = \langle_b$ then $u_{i+1} = \langle_c$ for some c and $s_{i+1} = s_i :: \langle c, b\rangle$, thus $\mathfrak{p}_2(s_{i+1}) = \mathfrak{p}_2(s_i)\, b$.
  - If $b \neq a$, then $|\mathfrak{p}_2(s_{i+1})|_a = |\mathfrak{p}_2(s_i)|_a \leqslant 1$.
  - If on the other hand $a = b$, then by definition of $\mathcal{WF}$ we know that $c_a(v_1 \dots v_i) = 0$, thus by Lemma 4.2 we get that $|\mathfrak{p}_2(s_i)|_a = 0$. Therefore $|\mathfrak{p}_2(s_{i+1})|_a = |\mathfrak{p}_2(s_i)|_a + 1 = 1 \leqslant 1$. □

C Proofs of Section 6

Fact 1. *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 $p_0 \xrightarrow{x_1}_{\mathscr{A}} p_1 \dots \xrightarrow{x_n}_{\mathscr{A}} p_n$ (ii) there is a run $q_0 \xrightarrow{y_1}_{\mathscr{A}\lfloor_{\bar{\mathbf{a}}}} q_1 \cdots \xrightarrow{y_n}_{\mathscr{A}\lfloor_{\bar{\mathbf{a}}}} q_n$ 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 $\pi_i \cdot \langle q_{i-1}, y_i, q_i\rangle = \langle p_{i-1}, x_i, p_i\rangle$.*

Proof. We proceed to prove both directions of the equivalence.

(ii)⇒(i): This is immediate since for every $\pi \in \mathfrak{S}_0, \pi \cdot \Delta = \Delta$.

(i)⇒(ii): We will prove this direction by induction on n. If $n = 0$, then by definition of $\bar{\mathbf{a}}$ there is some state $p_0 \in Q\lfloor_{\bar{\mathbf{a}}}$ and some permutation $\pi_0 \in \mathfrak{S}_0$ such that $\pi_0 \cdot q_0 = p_0$. The inductive case is illustrated in Figure 1. Assume we have $\pi_{i-1} \cdot q_{i-1} = p_{i-1}$ with $q_{i-1} \in Q\lfloor_{\bar{\mathbf{a}}}$ and $\pi_{i-1} \in \mathfrak{S}_0$. These hypotheses are represented by solid arrows in the diagram. By definition of $\bar{\mathbf{a}}$ we can find a permutation $\pi \in \mathfrak{S}_0$ such that $\pi^{-1} \cdot \langle p_{i-1}, x_i, p_i \rangle \in \Delta\lfloor_{\bar{\mathbf{a}}}$. On the figure, this gives us the dotted arrows. We write $p' = \pi^{-1} \cdot p_{i-1}$. Since:

$$\pi^{-1} \circ \pi_{i-1} \cdot q_{i-1} = p' \qquad \pi^{-1} \circ \pi_{i-1} \in \mathfrak{S}_0 \qquad \mathbf{supp}(q_{i-1}) \cup \mathbf{supp}(p') \subseteq \bar{\mathbf{a}}$$

there must be some permutation $\pi' \in \mathfrak{S}_0$ such that $\pi' \cdot \bar{\mathbf{a}} = \bar{\mathbf{a}}$ and $\pi' \cdot q_{i-1} = p'$. We choose $\pi_i = \pi \circ \pi'$, $y_i = \pi_i^{-1} \cdot x_i$ and $q_i = \pi_i^{-1} \cdot p_i$. This step is materialised by dashed arrows on the figure. Since $\pi, \pi' \in \mathfrak{S}_0$, we have $\pi_i \in \mathfrak{S}_0$, and since we already know that $\pi_i \cdot q_{i-1} = \pi \cdot \pi' \cdot q_{i-1} = \pi \cdot p' = p_{i-1}$, we get that $\pi_i \cdot \langle q_{i-1}, y_i, q_i \rangle = \langle p_{i-1}, x_i, p_i \rangle$. To conclude, notice that $\mathbf{supp}(y_i) = \pi_i^{-1} \cdot \mathbf{supp}(x_i) = \pi'^{-1} \cdot \mathbf{supp}\big(\pi^{-1} \cdot x_i\big)$. Since $\mathbf{supp}\big(\pi^{-1} \cdot x_i\big) \subseteq \bar{\mathbf{a}}$ and $\pi' \cdot \bar{\mathbf{a}} = \bar{\mathbf{a}}$, we conclude that $\mathbf{supp}(y_i) \subseteq \bar{\mathbf{a}}$, meaning $y_i \in \mathbb{X}\lfloor_{\bar{\mathbf{a}}}$. The same argument shows that $q_i \in Q\lfloor_{\bar{\mathbf{a}}}$. □

Fact 2. *For every run $q_0 \xrightarrow{w}_{\mathscr{A}'} q \in Q\lfloor_{\bar{\mathbf{a}}}$, the word $w \in \mathcal{WF}$, $\mathbf{m}(w) \leqslant \#\bar{\mathbf{a}}$ and either $a \in \mathbf{supp}(q) \cup \bar{\mathbf{a}}_0$ and $\mathcal{F}_a(w) = \mathbf{c}$, or $a \notin \mathbf{supp}(q) \cup \bar{\mathbf{a}}_0$ and $\mathcal{F}_a(w) = \varepsilon$.*

Proof. To prove that w is well-formed and that its memory is bounded by $\#\bar{\mathbf{a}}$, we actually show that for every word u prefix of w and any name a, $\mathcal{F}_a(u) \in \{\mathbf{c}, \varepsilon\}$. Since by construction $\mathbf{supp}(u) \subseteq \bar{\mathbf{a}}$, this entails that $\|u\| \leqslant \#\bar{\mathbf{a}}$, and since this holds for every prefix of w it means that $\mathbf{m}(w) \leqslant \#\bar{\mathbf{a}}$. It also means that for any decomposition $w = w_1 \langle_a w_2$, since both $\mathcal{F}_a(w_1)$ and $\mathcal{F}_a(w_1 \langle_a)$ belong to the set $\{\mathbf{c}, \varepsilon\}$, we have $\mathcal{F}_a(w_1) = \varepsilon$ and $\mathcal{F}_a(w_1 \langle_a) = \mathbf{c}$, therefore $w \in \mathcal{WF}$.

We proceed by induction on the path. The initial case is:

$$q_0 \xrightarrow{\langle_{a_1} \cdots \langle_{a_n}}_{\mathscr{A}'} q \text{ where } q \in I\lfloor_{\bar{\mathbf{a}}} \text{ and } a_1 \dots a_n \in (\mathbf{supp}(q) \cup \bar{\mathbf{a}}_0)^{\bowtie}.$$

Since $w = \langle_{a_1} \dots \langle_{a_n}$ is made exclusively of $\langle\_$ with no duplication, we have:

$$\mathcal{F}_a\left(\langle_{a_1} \dots \langle_{a_i}\right) = \begin{cases} \mathbf{c} & \text{if } a = a_j \text{ for some } j \leqslant i \\ \varepsilon & \text{otherwise} \end{cases}$$

This shows that for every prefix $\mathcal{F}_a(u) \in \{\mathbf{c}, \varepsilon\}$. For w itself, we get the value $\mathbf{c}$ for precisely the names in $\{a_1 \dots a_n\} = \mathbf{supp}(q) \cup \bar{\mathbf{a}}_0$.

Now, for the inductive step, assume that we have

$$q_0 \xrightarrow{w}_{\mathscr{A}'} p \xrightarrow{\langle_{a_1} \cdots \langle_{a_n}\, x\, {}_{b_1}\rangle \cdots {}_{b_m}\rangle}_{\mathscr{A}'} q \text{ where: } p \xrightarrow{x} q \in \Delta\lfloor_{\bar{\mathbf{a}}} \text{ and}$$

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

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

By induction hypothesis we know that we have $\mathcal{F}_a(u) \in \{\mathbf{c}, \varepsilon\}$ for every prefix u of w and that:

$$\mathcal{F}_a(w) = \begin{cases} \mathbf{c} & \text{if } a \in \mathbf{supp}(p) \cup \bar{\mathbf{a}}_0, \\ \varepsilon & \text{otherwise.} \end{cases}$$

We write $u = \langle_{a_1} \dots \langle_{a_n}$ and $v = {}_{b_1}\rangle \dots {}_{b_m}\rangle$. By definition we gather that:

$$\mathcal{F}_a(u) = \begin{cases} \mathbf{c} \text{ if } a \in (\mathbf{supp}(q) \cup \mathbf{supp}(x)) \setminus \mathbf{supp}(p), \\ \varepsilon \text{ otherwise.} \end{cases}$$

$$\mathcal{F}_a(x) = \begin{cases} \mathbf{f} \text{ if } a \in \mathbf{supp}(x), \\ \varepsilon \text{ otherwise.} \end{cases}$$

$$\mathcal{F}_a(v) = \begin{cases} \mathbf{d} \text{ if } a \in (\mathbf{supp}(p) \cup \mathbf{supp}(x)) \setminus \mathbf{supp}(q), \\ \varepsilon \text{ otherwise.} \end{cases}$$

To conclude on $\mathcal{F}_a(wuxv)$, we check the 16 possible boolean combinations of:

$$a \overset{?}{\in} \bar{\mathbf{a}}_0 \qquad a \overset{?}{\in} \mathbf{supp}(p) \qquad a \overset{?}{\in} \mathbf{supp}(q) \qquad a \overset{?}{\in} \mathbf{supp}(x)$$

and using the expressions above we compute $\mathcal{F}_a(wuxv)$ to verify that indeed this evaluates to $\mathbf{c}$ when $a \in \mathbf{supp}(q)$ and to ε otherwise. See Table 1 for a detailed analysis. To show that for every prefix w' of $wuxv$ and any name a the

$a \in \bar{\mathbf{a}}_0$	$a \in \mathbf{supp}(q)$	$a \in \mathbf{supp}(p)$	$a \in \mathbf{supp}(x)$	$\mathcal{F}_a(w)$	$\mathcal{F}_a(u)$	$\mathcal{F}_a(x)$	$\mathcal{F}_a(v)$	$\mathcal{F}_a(wuxv)$
✗	✗	✗	✗	ε	ε	ε	ε	ε
✗	✗	✗	✓	ε	$\mathbf{c}$	$\mathbf{f}$	$\mathbf{d}$	ε
✗	✗	✓	✗	$\mathbf{c}$	ε	ε	$\mathbf{d}$	ε
✗	✗	✓	✓	$\mathbf{c}$	ε	$\mathbf{f}$	$\mathbf{d}$	ε
✗	✓	✗	✗	ε	$\mathbf{c}$	ε	ε	$\mathbf{c}$
✗	✓	✗	✓	ε	$\mathbf{c}$	$\mathbf{f}$	ε	$\mathbf{c}$
✗	✓	✓	✗	$\mathbf{c}$	ε	ε	ε	$\mathbf{c}$
✗	✓	✓	✓	$\mathbf{c}$	ε	$\mathbf{f}$	ε	$\mathbf{c}$
✓	–	–	✗	$\mathbf{c}$	ε	ε	ε	$\mathbf{c}$
✓	–	–	✓	$\mathbf{c}$	ε	$\mathbf{f}$	ε	$\mathbf{c}$

Table 1. Case analysis for $\mathcal{F}_a(wuxv)$.

binding power of w' is either $\mathbf{c}$ or ε, we need to look into four cases:

1. w' is a prefix of w: in this case, we conclude using the induction hypothesis;
2. $w' = w\langle_{a_1} \dots \langle_{a_i}$: in this case,
   (a) if $a \notin \{a_1, \dots, a_i\}$, then $\mathcal{F}_a(w') = \mathcal{F}_a(w) \in \{\mathbf{c}, \varepsilon\}$,
   (b) otherwise we know that $a \notin \mathbf{supp}(p) \cup \bar{\mathbf{a}}_0$ so $\mathcal{F}_a(w) = \varepsilon$, hence we have: $\mathcal{F}_a(w') = \varepsilon \cdot \mathbf{c} = \mathbf{c}$;
3. $w' = wux$: in this case we way conclude by looking at Table 1;
4. $w' = wux\,{}_{b_1}\rangle \dots {}_{b_i}\rangle$: in this case,

(a) if $a \notin \{b_1, \dots, b_i\}$, then $\mathcal{F}_a(w') = \mathcal{F}_a(wux)$, which we covered in the previous case,
(b) otherwise we get $\mathcal{F}_a\left({}_{b_1}\rangle \dots {}_{b_i}\rangle\right) = \mathbf{d}$, and it implies that $a \in \mathbf{supp}(p) \cup \mathbf{supp}(x)$ hence $\mathcal{F}_a(w') = \mathcal{F}_a(wux) \cdot \mathbf{d} = \mathbf{c} \cdot \mathbf{d} = \varepsilon$. □

Lemma C.1. *$[\![e]\!] \subseteq \mathcal{WF}$ and $\mathbf{m}(e) \leqslant \#\bar{\mathbf{a}}$.*

Proof. Let $u \in [\![e]\!]$. By definition, there is a run:

$$q_0 \xrightarrow{w}_{\mathscr{A}'} q \xrightarrow{{}_{a_1}\rangle \dots {}_{a_n}\rangle}_{\mathscr{A}'} q_f \text{ where:}$$

$$u = w\,{}_{a_1}\rangle \dots {}_{a_n}\rangle \qquad q \in F\lfloor_{\bar{\mathbf{a}}} \qquad a_1 \dots a_n \in (\mathbf{supp}(q) \setminus \bar{\mathbf{a}}_0)^{\bowtie}.$$

By Fact 2, the word w is well-formed, $\mathbf{m}(w) \leqslant \#\bar{\mathbf{a}}$ and

$$\mathcal{F}_a(w) = \begin{cases} \mathbf{c} & \text{if } a \in \mathbf{supp}(q) \cup \bar{\mathbf{a}}_0 \\ \varepsilon & \text{otherwise.} \end{cases}$$

Clearly u is well-formed since a prefix of u ending with ${}_a\rangle$ must be a prefix of w. To conclude, notice that $\forall i$ we have:

$$\begin{aligned} \mathcal{F}_a\left(w\,{}_{a_1}\rangle \dots {}_{a_i}\rangle\right) &= \mathcal{F}_a(w) \cdot \mathcal{F}_a\left({}_{a_1}\rangle \dots {}_{a_i}\rangle\right) \\ &= \begin{cases} \mathbf{c} \cdot \varepsilon & \text{if } a \in \mathbf{supp}(q) \cup \bar{\mathbf{a}}_0 \text{ and } \forall j \leqslant i, a_j \neq a \\ \mathbf{c} \cdot \mathbf{d} & \text{if } a \in \mathbf{supp}(q) \cup \bar{\mathbf{a}}_0 \text{ and } \exists j \leqslant i, a_j = a \\ \varepsilon & \text{otherwise.} \end{cases} \\ &= \begin{cases} \mathbf{c} & \text{if } a \in \mathbf{supp}(q) \cup \bar{\mathbf{a}}_0 \text{ and } \forall j \leqslant i, a_j \neq a \\ \varepsilon & \text{otherwise.} \end{cases} \end{aligned}$$

This entails that for any name a, $\left|\mathcal{F}_a\left(w\,{}_{a_1}\rangle \dots {}_{a_i}\rangle\right)\right| \leqslant 1$, and since the support of $w\,{}_{a_1}\rangle \dots {}_{a_i}\rangle$ is contained in $\bar{\mathbf{a}}$ this means that $\left\|w\,{}_{a_1}\rangle \dots {}_{a_i}\rangle\right\| \leqslant \#\bar{\mathbf{a}}$. Now we conclude:

$$\begin{aligned} \mathbf{m}(u) &= \max\{\|v\| \mid v \in \mathbf{pref}(u)\} \\ &= \max\left(\max\{\|v\| \mid v \in \mathbf{pref}(w)\}, \max\left\{\left\|w\,{}_{a_1}\rangle \dots {}_{a_i}\rangle\right\| \mid 1 \leqslant i \leqslant n\right\}\right) \\ &= \max\left(\mathbf{m}(w), \max\left\{\left\|w\,{}_{a_1}\rangle \dots {}_{a_i}\rangle\right\| \mid 1 \leqslant i \leqslant n\right\}\right) \\ &\leqslant \max(\#\bar{\mathbf{a}}, \#\bar{\mathbf{a}}) \\ &= \#\bar{\mathbf{a}}. \end{aligned}$$

□

Fact 3. *For any $w \in \mathbb{\Sigma}^\star$, the word w belongs to $\mathbf{wf}\left(\mathcal{L}_{\mathscr{A}'}\right)$ if and only if there is a sequence of permutations $\pi_0 \dots \pi_{n+1} \in \mathfrak{S}_0$ and a run*

$$q_0 \xrightarrow{u_0}_{\mathscr{A}'} q_1 \xrightarrow{u_1}_{\mathscr{A}'} \cdots \xrightarrow{u_n}_{\mathscr{A}'} q_{n+1} \xrightarrow{u_{n+1}}_{\mathscr{A}'} q_f$$

such that $w = (\pi_0 \cdot u_0) \dots (\pi_{n+1} \cdot u_{n+1})$ and $\forall 0 < i \leqslant n,\ \pi_{i-1} \cdot q_i = \pi_i \cdot q_i$.

Before getting to the proof of this result, we establish the following technical lemmas.

Lemma C.2. *Let $q_0 \xrightarrow{u}_{\mathscr{A}'} p \xrightarrow{v}_{\Delta'} q$ be a run in $\mathscr{A}'$, and $[] -[u/u']\to_{\mathscr{T}_\alpha} s$ a run in $\mathscr{T}_\alpha$ with $u' \in \mathcal{WF}$. For every permutation such that $\pi \cdot \mathfrak{p}_1(s) = \mathfrak{p}_2(s)$, we have $u'(\pi \cdot v) \in \mathcal{WF}$ and there exists s' such that:*

$$s -[v/\pi \cdot v]\to_{\mathscr{T}} s' \qquad\qquad \pi \cdot \mathfrak{p}_1(s') = \mathfrak{p}_2(s').$$

Proof. Let us write $v = \langle_{a_1} \dots \langle_{a_n} x \, {}_{b_k}\rangle \dots {}_{b_1}\rangle$, with

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

$$b_k \dots b_1 \in ((\mathbf{supp}(p) \cup \mathbf{supp}(x)) \setminus (\mathbf{supp}(q) \cup \bar{\mathbf{a}}_0))^{\bowtie}.$$

We star by showing that $u'(\pi \cdot v) \in \mathcal{WF}$. Since u' is well-formed, it amounts to showing that:

$$\forall 1 < i \leqslant n, c_{\pi(a_i)}\left(u'\langle_{\pi(a_1)} \dots \langle_{\pi(a_{i-1})}\right) = 0. \tag{$\dagger$}$$

Notice the following identities:

$$d_{\pi(a_i)}\left(\langle_{\pi(a_1)} \dots \langle_{\pi(a_{i-1})}\right) = 0 \tag{$*$}$$

$$c_{\pi(a_i)}\left(\langle_{\pi(a_1)} \dots \langle_{\pi(a_{i-1})}\right) = 0 \tag{$**$}$$

The fact that $(*)$ holds is due to the fact that there are no closing brackets in the word considered, and $(**)$ stems from the fact that $\forall j \neq i, a_i \neq a_j$, which entails $\pi(a_i) \neq \pi(a_j)$. Hence we may simplify the expression we had before:

$$\begin{aligned} & c_{\pi(a_i)}\left(u'\langle_{\pi(a_1)} \dots \langle_{\pi(a_{i-1})}\right) \\ = \; & \left(c_{\pi(a_i)}(u') \dot{-} d_{\pi(a_i)}\left(\langle_{\pi(a_1)} \dots \langle_{\pi(a_{i-1})}\right)\right) + c_{\pi(a_i)}\left(\langle_{\pi(a_1)} \dots \langle_{\pi(a_{i-1})}\right) \\ = \; & \left(c_{\pi(a_i)}(u') \dot{-} 0\right) + 0 = c_{\pi(a_i)}(u'). \end{aligned}$$

Using Lemma 4.2 and the fact that $\mathfrak{p}_2(s) = \pi \cdot \mathfrak{p}_1(s)$, we may further obtain that:

$$c_{\pi(a_i)}(u') = |\mathfrak{p}_2(s)|_{\pi(a_i)} = |\pi \cdot \mathfrak{p}_1(s)|_{\pi(a_i)} = |\mathfrak{p}_1(s)|_{a_i}.$$

Finally, we know by Fact 2 that since $a_i \notin \mathbf{supp}(p) \cup \bar{\mathbf{a}}_0$, $|\mathfrak{p}_1(s)|_{a_i} = 0$. This means that $(\dagger)$ holds, hence that $u'm \in \mathcal{WF}$.

By defining $s_0 = s$ and $\forall 0 \leqslant i < n, s_{i+1} = s_i :: \langle a_i, \pi(a_i)\rangle$, we get a run:

$$s_0 -[\langle_{a_1} \dots \langle_{a_n} / \langle_{\pi(a_1)} \dots \langle_{\pi(a_n)}]\to_{\mathscr{T}} s_n.$$

Notice that this implies $\forall i, \mathfrak{p}_2(s_i) = \pi \cdot \mathfrak{p}_1(s_i)$.

The next step consists in checking that $s_n -[x/\pi \cdot x]\to_{\mathscr{T}} s_n$. To do so, we establish that if $a \in \mathbf{supp}(x)$, then $s_n \models \langle a, \pi(a)\rangle$. Let $a \in \mathbf{supp}(x)$.

- If $a \in \mathbf{supp}(p) \cup \bar{\mathbf{a}}_0$, then by Fact 2 we know that $a \in \mathfrak{p}_1(s)$. As $\mathfrak{p}_2(s) = \pi \cdot \mathfrak{p}_1(s)$, this means that $s \models \langle a, \pi(a)\rangle$. Because $a \in \mathbf{supp}(p)$, $a \notin \{a_1, \ldots, a_n\}$, so $\pi(a) \notin \{\pi(a_1), \ldots, \pi(a_n)\}$. Therefore $s_n \models \langle a, \pi(a)\rangle$.
- If $a \notin \mathbf{supp}(p) \cup \bar{\mathbf{a}}_0$, there must be some j such that $a_j = a$, hence

$$s_j = s_{j-1} :: \langle a_j, \pi(a_j)\rangle \models \langle a_j, \pi(a_j)\rangle .$$

  Since $\forall j < i, a_i \neq a_j \wedge \pi(a_i) \neq \pi(a_j)$, this ensures that $s_n \models \langle a, \pi(a)\rangle$.

We now define $t_{k+1} = s_n$, and $t_i = t_{i+1} \ominus \langle b_i, \pi(b_i)\rangle$. Notice that this preserves the fact that $\mathfrak{p}_2(t_i) = \pi \cdot \mathfrak{p}_1(t_i)$. We now show that $\forall j,\ i < j \Rightarrow t_j \models \langle b_i, \pi(b_i)\rangle$.

- If $b_i \in \mathbf{supp}(p) \cup \bar{\mathbf{a}}_0$, then by Fact 2 we know that $a \in \mathfrak{p}_1(s)$. As $\mathfrak{p}_2(s) = \pi \cdot \mathfrak{p}_1(s)$, this means that $s \models \langle b_i, \pi(b_i)\rangle$. By construction, and since all of the a_j are different from b_i, we know that $s_n = t_{k+1} \models \langle b_i, \pi(b_i)\rangle$. From there, we remove pairs $\langle b_l, \pi(b_l)\rangle$ from t_{k+1} to get to t_j, but since all of these are different from $\langle b_i, \pi(b_i)\rangle$ in the end we still have $t_j \models \langle b_i, \pi(b_i)\rangle$.
- If $b_i \notin \mathbf{supp}(p) \cup \bar{\mathbf{a}}_0$, then there must be some i' such that $a_{i'} = b_i$ hence:

$$s_{i'} = s_{i'-1} :: \langle a_{i'}, \pi(a_{i'})\rangle \models \langle a_{i'}, \pi(a_{i'})\rangle = \langle b_i, \pi(b_i)\rangle .$$

  Using the same reasoning as before, we get that $s_n = t_{k+1} \models \langle b_i, \pi(b_i)\rangle$, and thus $t_j \models \langle b_i, \pi(b_i)\rangle$.

This ensures that $t_{i+1} \models \langle b_i, \pi(b_i)\rangle$, so we get $t_{i+1} \; {-}[\,{}_{b_i}\rangle / {}_{\pi(b_i)}\rangle]{\to}_{\mathscr{T}} \; t_i$. We may now conclude since we know that $\mathfrak{p}_2(t_1) = \pi \cdot \mathfrak{p}_1(t_1)$ and we have a run: $s \; {-}[v/\pi \cdot v]{\to}_{\mathscr{T}} \; t_1$. □

Lemma C.3. *Let* $q_0 \xrightarrow{u}_{\mathscr{A}'} p \xrightarrow{v}_{\Delta'} q$ *be a run in* $\mathscr{A}'$*, and*

$$[] \; {-}[u/u']{\to}_{\mathscr{T}_\alpha} \; s \; {-}[v/v']{\to}_{\mathscr{T}_\alpha} \; s'$$

a run in $\mathscr{T}_\alpha$ *with* $u'v' \in \mathcal{WF}$*. There exists a permutation such that:*

$$\pi \cdot \mathfrak{p}_1(s) = \mathfrak{p}_2(s), \qquad \pi \cdot \mathfrak{p}_1(s') = \mathfrak{p}_2(s'), \qquad (\pi \cdot v) = v'.$$

Proof. Let us write $v = \langle_{a_1} \ldots \langle_{a_n} x \, {}_{b_k}\rangle \cdots {}_{b_1}\rangle$, with

$$a_1 \ldots a_n \in ((\mathbf{supp}(q) \cup \mathbf{supp}(x)) \setminus (\mathbf{supp}(p) \cup \bar{\mathbf{a}}_0))^{\bowtie}$$

$$b_k \ldots b_1 \in ((\mathbf{supp}(p) \cup \mathbf{supp}(x)) \setminus (\mathbf{supp}(q) \cup \bar{\mathbf{a}}_0))^{\bowtie}.$$

In order to find the appropriate permutations, we decompose the run of the transducer as follows:

$$\begin{aligned} s = s_0 \; {-}[\langle_{a_1} / \langle_{a'_1}]{\to}_{\mathscr{T}} \; s_1 \cdots {-}[\langle_{a_n} / \langle_{a'_n}]{\to}_{\mathscr{T}} \; s_n \; {-}[x/x']{\to}_{\mathscr{T}} \; s_n \\ s_n = t_{k+1} \; {-}[\,{}_{b_k}\rangle / {}_{b'_k}\rangle]{\to}_{\mathscr{T}} \; t_k \cdots {-}[\,{}_{b_1}\rangle / {}_{b'_1}\rangle]{\to}_{\mathscr{T}} \; t_1 = s'. \end{aligned}$$

This means that $v' = \langle_{a'_1} \dots \langle_{a'_n} x' \, {}_{b'_1}\rangle \cdots \, {}_{b'_k}\rangle$. It is straightforward to see that:

$$\forall 0 \leqslant i \leqslant n, s_i = s :: \langle a_1, a'_1\rangle \cdots :: \langle a_i, a'_i\rangle .$$

Since $u'v'$ is well-formed, we know that for every i, $c_{a'_i}\left(w' \langle_{a'_1} \dots \langle_{a'_{i-1}}\right) = 0$. According to Lemma 4.2, this implies that $a'_i \notin \mathfrak{p}_2(s_{i-1})$, which in turn proves that for every i, $\mathfrak{p}_2(s_i)$ has no duplicates. For the same reason $\mathfrak{p}_1(s_i)$ has no duplicates, so there exists a permutation π_i such that $\mathfrak{p}_2(s_i) = \pi_i \cdot \mathfrak{p}_1(s_i)$. By definition of s_i, this implies that $\forall i, a'_i = \pi_i(a_i)$. We also get that $\mathfrak{p}_2(s) = \pi_i \cdot \mathfrak{p}_1(s)$.

Since $s_n \, {-}[x/x']{\to}_{\mathscr{T}} \, s_n$ there must be a permutation π_x such that $x' = \pi_x \cdot x$ and $\forall a \in \mathbf{supp}(x), s_n \models \langle a, \pi_x(a)\rangle$. As we have just shown that $\mathfrak{p}_2(s_n) = \pi_n \cdot \mathfrak{p}_1(s_n)$, and because whenever $a \in \mathbf{supp}(x)$ either $a \in \mathbf{supp}(p) \cup \bar{\mathbf{a}}_0$ or $a \in \{a_1, \dots, a_n\}$, so in both cases $a \in \mathfrak{p}_1(s_n)$, we know that $\forall a \in \mathbf{supp}(x), \pi_x(a) = \pi_n(a)$, which implies $x' = \pi_n \cdot x$.

For every step $t_{i+1} \, {-}[{}_{b_i}\rangle / {}_{b'_i}\rangle]{\to}_{\mathscr{T}} \, t_i$, we know that $t_{i+1} \models \langle b_i, b'_i\rangle$ and $t_i = t_{i+1} \ominus \langle b_i, b'_i\rangle$. From the later remark, a simple induction shows that $\forall i, \mathfrak{p}_2(t_i) = \pi_n \cdot \mathfrak{p}_1(t_i)$. We may show from Fact 2 that

$$a \in \mathfrak{p}_1(t_i) \Leftrightarrow (a \in \mathbf{supp}(p) \cup \mathbf{supp}(q) \cup \mathbf{supp}(x) \cup \bar{\mathbf{a}}_0) \setminus \{b_k, \dots, b_i\} .$$

Therefore $b_i \in \mathfrak{p}_1(t_{i+1})$, meaning that $t_{i+1} \models \langle b_i, b'_i\rangle$ entails $\langle b_i, b'_i\rangle \in t_{i+1}$. Since $\mathfrak{p}_2(t_{i+1}) = \pi_n \cdot \mathfrak{p}_1(t_{i+1})$, this in turn ensures that $b'_i = \pi_n(b_i)$.

Summing up, if we write $\pi := \pi_n$, we have $\pi \cdot \mathfrak{p}_1(s) = \mathfrak{p}_2(s)$, $\pi \cdot \mathfrak{p}_1(s') = \mathfrak{p}_2(s')$, and:

$$\pi \cdot v = \langle_{\pi(a_1)} \dots \langle_{\pi(a_n)} (\pi \cdot x) \; {}_{\pi(b_k)}\rangle \cdots \, {}_{\pi(b_1)}\rangle = \langle_{a'_1} \dots \langle_{a'_n} x' \, {}_{b'_k}\rangle \cdots \, {}_{b'_1}\rangle = v'.$$

This means that π satisfies the requirements we wanted. □

We may now prove Fact 3.

Proof (Fact 3). Assume $w \in \mathbf{wf}\left(\mathcal{L}_{\mathscr{A}'}\right)$. Then there is are runs:

$$q_0 \xrightarrow{u_0}_{\mathscr{A}'} q_1 \cdots \xrightarrow{u_n}_{\mathscr{A}'} q_{n+1} \xrightarrow{u_{n+1}}_{\mathscr{A}'} q_f$$

$$[] \, {-}[u_0/v_0]{\to}_{\mathscr{T}_\alpha} \, s_1 \cdots {-}[u_n/v_n]{\to}_{\mathscr{T}_\alpha} \, s_{n+1} \, {-}[u_{n+1}/v_{n+1}]{\to}_{\mathscr{T}_\alpha} \, s_f$$

Such that:

$$q_1, \dots, q_n \in Q\lfloor_{\bar{\mathbf{a}}} \qquad q_1 \in I\lfloor_{\bar{\mathbf{a}}} \qquad q_n \in F\lfloor_{\bar{\mathbf{a}}} \qquad s_f \in \mathbb{S}^{acc} \qquad w = v_0 \dots v_{n+1}.$$

Using a similar argument as in the proof of Lemma C.3, we find a permutation $\pi_0 \in \mathfrak{S}_{\mathbb{A}}$ such that

$$\pi_0 \cdot \mathfrak{p}_1(s_1) = \mathfrak{p}_2(s_1) \qquad\qquad \pi_0 \cdot u_0 = v_0.$$

We may then apply Lemma C.3 recursively, since $\forall i, v_0 \dots v_i \in \mathcal{WF}$, and get a sequence of permutations $\pi_1, \dots, \pi_n \in \mathfrak{S}_{\mathbb{A}}$ such that:

$$\pi_i \cdot \mathfrak{p}_1(s_i) = \mathfrak{p}_2(s_i) \qquad \pi_i \cdot \mathfrak{p}_1(s_{i+1}) = \mathfrak{p}_2(s_{i+1}) \qquad \pi_i \cdot u_i = v_i.$$

Using a again a similar argument as in the proof of Lemma C.3, we find a permutation $\pi_{n+1} \in \mathfrak{S}_{\mathbb{A}}$ such that

$$\pi_{n+1} \cdot \mathfrak{p}_1(s_{n+1}) = \mathfrak{p}_2(s_{n+1}) \qquad \pi_{n+1} \cdot \mathfrak{p}_1(s_f) = \mathfrak{p}_2(s_f) \qquad \pi_{n+1} \cdot u_{n+1} = v_{n+1}.$$

By Fact 2, we know that $\mathfrak{p}_1(s_{n+1}) \approx \mathbf{supp}(q_{n+1}) \cup \bar{\mathbf{a}}_0$. By construction, we know $u_{n+1} = {}_{a_1}\rangle \dots {}_{a_k}\rangle$, for $a_1 \dots a_k \in (\mathbf{supp}(q_{n+1}) \setminus \bar{\mathbf{a}}_0)^{\bowtie}$. According to Lemma 4.2, we deduce that $\mathfrak{p}_1(s_f) \approx \bar{\mathbf{a}}_0$. Since $s_f \in \mathbb{S}^{acc}$, we know that $\langle a, b\rangle \in s_f \Rightarrow a = b$. This means that since $\pi_{n+1} \cdot \mathfrak{p}_1(s_f) = \mathfrak{p}_2(s_f)$, we have:

$$a \in \bar{\mathbf{a}}_0 \Leftrightarrow a \in \mathfrak{p}_1(s_f) \Leftrightarrow \langle a, \pi_{n+1}(a)\rangle \in s_f \Rightarrow \pi_{n+1}(a) = a.$$

Since $\forall 0 < i \leqslant n+1, \pi_i \cdot \mathfrak{p}_1(s_i) = \mathfrak{p}_2(s_i) = \pi_{i-1} \cdot \mathfrak{p}_1(s_i)$, and since we know by Fact 2 and Lemma 4.2 that $\bar{\mathbf{a}}_0 \subseteq \mathbf{supp}(q_i) \approx \mathfrak{p}_1(s_i)$, we can conclude by descending recursion that $\forall i, \pi_i \in \mathfrak{S}_0$.

For the other direction, assume we have

$$q_0 \xrightarrow{u_0}_{\mathscr{A}'} q_1 \cdots \xrightarrow{u_n}_{\mathscr{A}'} q_{n+1} \xrightarrow{u_{n+1}}_{\mathscr{A}'} q_f$$

$$\pi_0, \dots, \pi_{n+1} \in \mathfrak{S}_0 \qquad \pi_{i-1} \cdot q_i = \pi_i \cdot q_i$$

Using a similar argument as Lemma C.2, we show that $\pi_0 \cdot u_0 \in \mathcal{WF}$ and find a stack s_1 such that:

$$\pi_0 \cdot \mathfrak{p}_1(s_1) = \mathfrak{p}_2(s_1) \qquad [] \; -[u_0/\pi_0 \cdot u_0]\rightarrow_{\mathscr{T}_\alpha} s_1.$$

We apply Lemma C.2 recursively, and get stacks such that:

$$\pi_i \cdot \mathfrak{p}_1(s_i) = \mathfrak{p}_2(s_i) \qquad \pi_i \cdot \mathfrak{p}_1(s_{i+1}) = \mathfrak{p}_2(s_{i+1}) \qquad s_i \; -[u_i/\pi_i \cdot u_i]\rightarrow_{\mathscr{T}_\alpha} s_{i+1}.$$

We also show along the way that $(\pi_0 \cdot u_0) \dots (\pi_i \cdot u_i) \in \mathcal{WF}$.

Since $\pi_{n+1} \cdot q_{n+1} = \pi_n \cdot q_{n+1}$, and since we have assumed that Q is strict, we know that $\forall a \in \mathbf{supp}(q_{n+1}), \pi_n(a) = \pi_{n+1}(a)$. Since we know that

$$\mathfrak{p}_1(s_{n+1}) \approx \mathbf{supp}(q_{n+1}) \cup \bar{\mathbf{a}}_0 \qquad \pi_n, \pi_{n+1} \in \mathfrak{S}_0 \qquad \pi_n \cdot \mathfrak{p}_1(s_{n+1}) = \mathfrak{p}_2(s_{n+1})$$

this entails $\pi_{n+1} \cdot \mathfrak{p}_1(s_{n+1}) = \mathfrak{p}_2(s_{n+1})$. Again, with a similar argument as Lemma C.2, we show that

$$(\pi_0 \cdot u_0) \dots (\pi_n \cdot u_n)(\pi_{n+1} \cdot u_{n+1}) \in \mathcal{WF}$$

and find a stack s_f such that:

$$\pi_{n+1} \cdot \mathfrak{p}_1(s_f) = \mathfrak{p}_2(s_f) \qquad s_{n+1} \; -[u_{n+1}/\pi_{n+1} \cdot u_{n+1}]\rightarrow_{\mathscr{T}_\alpha} s_f.$$

s_{n+1} is accepting, since it only contains pairs $\langle a, \pi_{n+1}(a)\rangle$ with $a \in \bar{\mathbf{a}}_0$. This means that $(\pi_0 \cdot u_0) \dots (\pi_n \cdot u_n)(\pi_{n+1} \cdot u_{n+1})$ is a well-formed word that is α-equivalent to a word in $\mathcal{L}_{\mathscr{A}'}$, meaning it belongs to $\mathbf{wf}(\mathcal{L}_{\mathscr{A}'})$. □