Regular Expressions with Dynamic Name Binding

Dexter Kozen¹, Stefan Milius², Lutz Schröder², and Thorsten Wißmann²

¹ Cornell University
² Friedrich-Alexander-Universität Erlangen-Nürnberg

Abstract. Nominal Kleene algebra (NKA) is a formalism to specify trace languages with name generation; it extends standard regular expressions with a name binding construct. NKA has been proved complete over a natural nominal language model. Moreover, it has been shown that NKA expressions can be translated into a species of nondeterministic nominal automata, thus providing one half of a Kleene theorem. The other half is known to fail, i.e. there are nominal languages that can be accepted by a nominal automaton but are not definable in NKA. In the present work, we introduce a calculus of regular expressions with dynamic name binding. It satisfies the full Kleene theorem, i.e. it is equivalent to a natural species of nominal automata, and thus strictly more expressive than NKA. We show that containment checking in our calculus is decidable in \textsf{EXPSpace}, and in fact has polynomial fixed-parameter space complexity. The known \textsf{EXPSpace} bound for containment of NKA expressions follows.

1 Introduction

The concept of name plays a key role for many aspects of the syntax and semantics of processes and computations. In particular, the generation of fresh resources (e.g. channel names as used in the π-calculus) involves name binding operators, which declare names that are available only within a given scope. Traces of such computations are naturally based on suitable generalizations of the notion of word that take into account bound names. Nominal Kleene algebra (NKA) has been introduced by Gabbay and Ciancia as a foundation for trace languages with bound names in this spirit [6]. It extends the usual calculus of regular expressions with a locally scoped binding construct \( \nu \); e.g. \((\nu a.a)^*\) denotes a process that generates and uses any finite number of fresh names. Like many recent developments in the semantics of calculi with name binding (e.g. [4]), NKA is cast in the framework of nominal sets [17], which incorporates names and their renaming into the underlying set theory.

The semantics of NKA turns out to be a subtle issue. The semantics originally defined by Gabbay and Ciancia was given in terms of languages over the infinite alphabet of names, and interpreted \( \nu \) as generating a genuinely fresh name; e.g. the above-mentioned expression \((\nu a.a)^*\) defines, in this semantics, the language of all words that do not repeat any letter. It was subsequently shown by Kozen et al. [13] that the equational calculus for NKA [6] is incomplete over this semantics but complete over an alternative more fine-grained semantics, which was later [12] shown to be equivalent to a language semantics over words that keep explicit binders, so-called \( \nu \)-strings; we refer to such languages as \( \nu \)-languages. This amounts to a stricter distinction between free and bound
names; e.g. the terms \( a + \nu a.a \) and \( \nu a.a \) become inequivalent in \( \nu \)-language semantics. This behavior is similar to that of some forms of trace semantics for the \( \pi \)-calculus, which also retain binding operators in traces [9]. Finitely supported \( \nu \)-languages are final for a natural species of deterministic nominal automata, to wit, they form the final coalgebra for the functor \( K \) taking a nominal set \( X \) to
\[
KX = 2 \times X^A \times [A]X
\]
where \( A \) is the set of names and \([A]\) denotes the name abstraction functor [17]. Such automata have two types of transitions, consuming free and bound names, respectively.

It has been shown [12] that NKA expressions can be converted into equivalent finite nondeterministic automata of a similar type, thus establishing one direction of a Kleene theorem. The other direction fails [12], essentially because automata have no consideration for nesting of scopes and therefore can accept \( \nu \)-languages that require unbounded nesting of name bindings. (The conclusion of [12] mentions a Kleene theorem for deterministic nominal automata and a language with explicit recursion.)

Here, we introduce a simple language of regular bar expressions, which extend standard regular expressions by introducing a second sort of letters \( \bar{a} \), understood as simultaneously binding and consuming the name \( a \). Unlike for the \( \nu \)-binders of NKA, the scope of such a bound name never ends after the name is bound, and crosses the borders of other syntactic constructs such as Kleene star; hence our use of the term dynamic. This construct is similar to the binders in regular expressions with memory, introduced by Libkin et al. as a formalism for data languages [15]. Such binders have a more imperative than declarative flavor but are necessary to obtain equivalence results with automata [15]. Indeed we establish a full Kleene theorem for regular bar expressions, showing that they are polynomially equivalent to nondeterministic nominal automata that are regular, i.e. orbit finite and, up to \( \alpha \)-equivalence, finitely branching. The translation of NKA expressions into nondeterministic nominal automata [12] actually produces regular automata, so that our Kleene theorem implies that regular bar expressions are (strictly) more expressive than NKA.

We show moreover that containment of regular bar expressions can be checked in exponential space, and in fact more precisely in fixed-parameter polynomial space, with the parameter being essentially the number of distinct bound names in the expressions. As the translation from NKA to regular bar expressions incurs an exponential blowup in expression size but leaves the bound names unchanged, the known EXPSPACE bound for containment of NKA expressions [12] follows; the fixed-parameter version, however, does not carry over from regular bar expressions to NKA expressions.

**Related Work** There is extensive work on expression calculi for data languages, i.e. languages over infinite alphabets (often more specifically over products of a finite alphabet and an infinite data component; here we consider only infinite alphabets of names), which goes back at least as far as Kaminski and Francez’s finite memory automata [10] (also called register automata). A complete summary of results on data languages is out of scope of the current work, which is primarily concerned with \( \nu \)-languages; surveys are found, e.g., in [18,8]). Nevertheless we do briefly discuss a data language semantics of regular bar expressions (coarser than the semantics in terms of
ν-languages), with a view to relating regular nondeterministic nominal automata to nondeterministic orbit-finite automata as studied by Bojanczyk et al. [2]; see Section 6. Generally, expression calculi and automata models for data words tend to be much more expressive than regular bar expressions, and often have undecidable containment problems [16]. In terms of data languages, regular bar expressions can be embedded into regular expressions with memory [15], which are equivalent to register automata (in particular have an undecidable containment problem). Closer to regular bar expressions, in terms of expressive power, are Kaminski and Tan’s regular expressions over infinite alphabets [11], which are equivalent to finite-state unification-based automata and hence have a decidable containment problem [20]. These expressions effectively feature an only mildly restricted form of name binding without immediate use of the name; we do not know whether this amounts to a strict increase in expressive power but expect that it implies higher complexity (about which there do not appear to be any claims in the literature).

We emphasize that our main interest in this work is in languages of ν-strings rather than in data languages. Besides the mentioned work on NKA [6,12], there is previous work on regular expressions for languages over words with explicit binding by Kurz et al. [14]. The words used in their semantics are similar to ν-strings but differ technically in being taken only modulo α-equivalence, not the other equations of NKA (in particular concerning scope extension of binders). They satisfy a Kleene theorem for a species of automata that incorporate an explicit bound on the nesting depth of bindings, and reject words that exceed this depth.

2 Preliminaries

\textbf{G-sets.} Recall that a group action of a group \( G \) on a set \( X \) is a map \( G \times X \to X \), denoted by juxtaposition or infix \( \cdot \), such that \( \pi(\rho x) = (\pi \rho)x \) and \( 1x = x \) for every \( \pi, \rho \in G \) and every \( x \in X \). A \( G \)-set is a set \( X \) equipped with a group action of the group \( G \) on \( X \). The orbit of \( x \in X \) is the set \{\( \pi x \mid \pi \in G \}\}. A function \( f : X \to Y \) between \( G \)-sets \( X, Y \) is equivariant if \( f(\pi x) = \pi(f(x)) \) for all \( \pi \in G, x \in X \). Given a \( G \)-set \( X \), \( G \) acts on subsets \( A \subseteq X \) by \( \pi A = \{\pi x \mid x \in A\} \). For \( A \subseteq X \ni x \), we put \( \text{fix} x = \{\pi \in G \mid \pi x = x\} \) and \( \text{Fix} A = \bigcap_{x \in A} \text{fix} x \), i.e. elements of fix \( A \) and Fix \( A \) fix \( A \) setwise and pointwise, respectively.

\textbf{Nominal sets.} Fix a countably infinite set \( \mathcal{A} \) of names, and denote by \( \mathcal{G} \) the group of finite permutations on \( \mathcal{A} \), i.e. the subgroup of the symmetric group \( \text{Sym}(\mathcal{A}) \) generated by the transpositions. The set \( \mathcal{A} \) is a \( \mathcal{G} \)-set under the group action given by \( \pi a = \pi(a) \). Given a \( \mathcal{G} \)-set \( X \) we say that \( A \subseteq \mathcal{A} \) supports \( x \in X \) if \( \text{Fix} A \subseteq \text{fix} x \). We say that \( x \in X \) has finite support if there exists a finite \( A \subseteq \mathcal{A} \) that supports \( x \). If \( x \) has finite support, then there exists a smallest set supporting \( x \), denoted \( \text{supp}(x) \). For \( a \in \mathcal{A} \), we write \( a \sharp x \) for \( a \notin \text{supp}(x) \). A nominal set is a \( \mathcal{G} \)-set \( X \) all whose elements have finite support. For every equivariant function \( f \) between nominal sets we have \( \text{supp}(f(x)) \subseteq \text{supp}(x) \). The function \( \text{supp} \) itself is equivariant, thus for any \( \pi \in G \), \( \text{supp}(\pi x) = \pi(\text{supp}(x)) \). It
follows that if \( x_1, x_2 \) are in the same orbit of a nominal set, then \( \sharp \text{supp}(x_1) = \sharp \text{supp}(x_2) \) (we use \( \sharp \) for cardinalities, avoiding overuse of '|\(^\#\)'). A subset \( S \subseteq X \) is \textit{finitely supported} (fs) if \( S \) has finite support w.r.t. the above-mentioned action of \( G \) on subsets; \textit{equivariant} if \( \pi x \in S \) for all \( \pi \in G \) and \( x \in S \) (which implies \( \text{supp}(S) = \emptyset \)); and \textit{uniformly finitely supported} (ufs) if \( \bigcup_{x \in S} \text{supp}(x) \) is finite \([12]\). We have

**Lemma 2.1** ([5], Theorem 2.29). If \( S \) is ufs, then \( \text{supp}(S) = \bigcup_{x \in S} \text{supp}(x) \).

For a nominal set \( X \), we denote by \( P_{\text{fs}}(X) \), \( P_{\text{ufs}}(X) \), and \( P_{\omega}(X) \) the sets of fs, ufs, and finite subsets of \( X \), respectively. Note that any ufs \( S \subseteq X \) is finitely supported, but not conversely (e.g. the set \( \mathbb{A} \) is fs but not ufs). Moreover, any finite subset of \( X \) is ufs but not conversely (e.g. the set of all words \( a^n \) for fixed \( a \in \mathbb{A} \) is ufs but not finite). A nominal set \( X \) is \textit{orbit-finite} if the action of \( G \) on it has only finitely many orbits.

**Lemma 2.2.** For any orbit-finite \( X \), \( P_{\text{ufs}}(X) = P_{\omega}(X) \).

Nominal sets and equivariant maps form a category, \( \text{Nom} \). On \( \text{Nom} \), we have the \textit{abstraction functor} \( [\mathbb{A}]_\sim \), that takes a nominal set \( X \) to the set \( [\mathbb{A}]X = (\mathbb{A} \times X)/\sim \). Here, the relation \( \sim \) abstracts \( \alpha \)-equivalence: \( (a, x) \sim (b, y) \) iff \( (ca) \cdot x = (cb) \cdot y \) for any fresh \( c \). We write \( (a) x \) for the equivalence class of \( (a, x) \) under \( \sim \).

**Coalgebra** An \textit{\( F \)-coalgebra} \((C, \gamma)\) for an endofunctor \( F : C \rightarrow C \) on a category \( C \) consists of a \( C \)-object \( C \) of \textit{states} and a morphism \( \gamma : C \rightarrow FC \); here, we are interested in the case \( C = \text{Nom} \). A \textit{coalgebra morphism} \( f : (C, \gamma) \rightarrow (D, \delta) \) is a morphism \( f : C \rightarrow D \) such that \( Ff \gamma = \delta f \). An \( F \)-coalgebra \((C, \gamma)\) is \textit{final} if for each \( F \)-coalgebra \((D, \delta)\), there exists a unique coalgebra morphism \((D, \delta) \rightarrow (C, \gamma)\). A pointed coalgebra is a coalgebra with a distinguished \textit{initial state}.

E.g., \( F \)-coalgebras for the functor \( F \mathbb{X} = \mathbb{A} \times X \) on \( \text{Nom} \) consist of two equivariant maps \( X \rightarrow \mathbb{A} \) (output) and \( X \rightarrow X \) (next state), and hence produce, at each state \( x \), a stream of names; equivariance and finite support of \( x \) imply that this stream has finite support, i.e. only finitely many distinct names appear in it. Consequently, the final \( F \)-coalgebra in this case is the set of finitely supported streams over \( \mathbb{A} \).

**Nominal Kleene Algebra** We recall that expressions \( r, s \) of nominal Kleene algebra (NKA) \([6]\), briefly \textit{NKA expressions}, are defined by the grammar

\[
    r, s ::= 0 \mid 1 \mid a \mid r + s \mid rs \mid r^\ast \mid va. r \quad (a \in \mathbb{A}).
\]

We adopt the notational convention that sequential composition is associative and the scope of \( \nu \) extends as far to the right as possible. The semantics that we use here, introduced by Kozen et al. \([13,12]\), is defined in terms of languages over words with binding, so called \textit{\( \nu \)-strings}, which are either 1 or \( \nu \)-regular expressions formed using only names \( a \in \mathbb{A} \), sequential composition, and name binding \( \nu \), taken modulo associativity of sequential composition. We write \( \mathbb{A}^\nu \) for the set of \( \nu \)-strings, and \( M_\nu \) for \( \mathbb{A}^\nu \) modulo the congruence \( \equiv \) generated by \( \alpha \)-equivalence and the laws

\[
    \nu a. vb. r \equiv \nu b. \nu a. r \quad (\nu a. r)s \equiv \nu a.(rs) \quad s(\nu a. r) \equiv \nu a.(sr) \quad \nu a.s \equiv s \quad \text{for } a^\ast s.
\]
We refer to finitely supported subsets of \( M_r \) as \( \nu \)-languages. The \( \nu \)-language \( L_\alpha(r) \) defined by a \( \nu \)-regular expression is then in fact a ufs subset of \( M_r \) given by

\[
L_\alpha(\nu a. r) = \{ [\nu a. w] \equiv [w] \equiv \in L_\alpha(r) \}
\]

and the usual clauses for the other operators (to be definite, \( L_\alpha(r + s) = L_\alpha(r) \cup L_\alpha(s) \); \( L_\alpha(rs) = \{ [\nu w] \equiv [w] \equiv \in L_\alpha(r), [\nu v] \equiv \in L_\alpha(s) \} \); \( L_\alpha(a) = \{ [\nu a] \equiv \} \); \( L_\alpha(1) = \{ [1] \equiv \} \); \( L_\alpha(0) = \emptyset \).

We regard \( \nu a \) as binding the name \( a \) in \( \nu a.r \), inducing standard notions of free and bound names as well as \( \alpha \)-equivalence for NKA expressions. We write \( FN(r) \) for the set of free names in \( r \). We say that an NKA expression is clean if its bound variables are mutually distinct and distinct from all its free variables. Clearly, every expression is \( \alpha \)-equivalent to a clean one. We write \( ub(r) \) (for unbind) for the term arising from \( r \) by removing all \( \nu \)-bindings, i.e. \( ub \) is defined by \( ub(\nu a.r) = ub(r) \), \( ub(a) = a \) for \( a \in \alpha \), and by commutation with all other operators. Gabbay and Ciancia’s original semantics is given in terms of languages over the infinite alphabet \( \alpha \), where \( \nu \) is regarded as generating fresh names without possibility of deallocation; i.e. an NKA expression is interpreted as the language \( N(L_\alpha(r)) \) where

\[
N(L) = \{ ub(w) \mid w \text{ clean}, [w] \equiv \in L \} \subseteq \alpha^*
\]

for a \( \nu \)-language \( L \). It has been shown [13] that the NKA equations postulated by Gabbay and Ciancia [6] are incomplete over the semantics \( N(L_\alpha(\cdot)) \) but complete over \( L_\alpha(\cdot) \) (the different formulation of \( L_\alpha(\cdot) \) in [13] is isomorphic to the one given here [12]).

A simple separating example is that \( N(L_\alpha(a + \nu a.a)) = \alpha = N(L_\alpha(\nu a.a)) \) but \( L_\alpha(a + \nu a.a) = \{ a, \nu a.a \} \neq \{ \nu a.a \} = L_\alpha(\nu a.a) \) (eliding equivalence classes); that is, the semantics \( L_\alpha(\cdot) \) distinguishes more strictly between free and bound names.

The \( \nu \)-languages form the final coalgebra for the endofunctor \( K \) on Nom given by

\[
KX = 2 \times X^\alpha \times [\alpha]\times X,
\]

whose coalgebras are deterministic nominal automata [12]: they convey, at each state \( s \), information about whether \( s \) is final (component 2), which states are reached via a free transition for a given name \( a \in \alpha \) (component \( X^\alpha \)), and which states are reached via a bound transition (component \([\alpha]\times X\)), where bound transitions are invariant under \( \alpha \)-equivalence. The description of the final \( K \)-coalgebra confirms that bound transitions in such automata should be read as not only allocating but also as immediately consuming a new name, and that bound names should be treated as strictly distinct from free ones.

Every NKA expression can be converted into an orbit-finite nondeterministic nominal automaton of a quite similar flavor, which amounts to one direction of a Kleene theorem [12]; we discuss details in Remark 4.21. The other direction of the Kleene theorem is known to fail even for orbit-finite deterministic nominal automata:

**Example 2.3.** [12] It is easy to construct an orbit-finite deterministic nominal automaton accepting the \( \nu \)-language

\[
\{ \epsilon, \nu b. a, \nu b. b (\nu a. b), \nu b. b (\nu a. b (\nu b. b)), \nu b. b (\nu a. b (\nu b. b (\nu a. b))), \ldots \},
\]

which however can be shown to require unbounded nesting depth of \( \nu \) and hence cannot be defined by an NKA expression.
3 Regular Bar Expressions

To remedy the failure of the Kleene theorem for NKA, we introduce a modified calculus that differs from NKA in two respects: First, we restrict binders to appear only immediately before the first actual use of the bound name; and second, we give up explicit scoping of binders, letting the scope of every binder effectively reach up to the end of the whole expression (unless shadowing occurs in between). We write the binding and use of $a$ as $|a|$, with the vertical bar thought of as belonging to an opening bracket '(' with infinite radius. As indicated earlier, similar constructs have appeared in calculi for data languages (here, we remain interested primarily in $\nu$-languages). E.g. in regular expressions with memory [15], the expression $a \downarrow x$ will consume the letter $a$ and simultaneously bind the encountered data value to the variable $x$. Kaminski and Tan’s regular expressions over infinite alphabets [11] have a construct that binds variables, without consuming them immediately, for the remainder of the expression, again up to shadowing. In dynamic sequences [7], there are two dynamically scoped constructs $\langle a \rangle$ and $\langle a \rangle$ for dynamic allocation and deallocation, respectively, of a name $a$; in this notation, our $a$ corresponds to $\langle aa \rangle$. Formal definitions are as follows.

**Definition 3.1.** Regular bar expressions $r, s$ (over $\check{A}$) are generated by the grammar

$$r, s ::= a \mid |a| \mid 1 \mid 0 \mid r + s \mid rs \mid r^* \quad (a \in \check{A});$$

in other words, regular bar expressions are just regular expressions over the alphabet $\check{\check{A}} = \check{A} \cup \{a \mid a \in \check{A}\}$.

We understand $|a|$ as binding and consuming the name $a$. The degree $\deg(r)$ of a regular bar expression $r$ is the number of distinct bound names in $r$, i.e. $\deg(r) = \#\{a \mid |a| \text{ occurs in } r\}$.

A bar string is a word over $\check{A}$, i.e. an element of $\check{\check{A}}^*$. The set $\check{\check{A}}^*$ is made into a nominal set by the letter-wise action of $G$. The free names occurring in a bar string $w$ are those names $a$ that occur in $w$ to the left of any occurrence of $|a|$. We write $FN(w)$ for the set of free names of $w$, and say that $w$ is closed of $FN(w) = \emptyset$. We define $\alpha$-equivalence $\equiv_\alpha$ on bar strings as the equivalence (not: congruence) generated by $w|av = w|b(\pi_{ab} \cdot v)$ where $b \notin FN(v)$ and $\pi_{ab}$ is the transposition $(ab)$. The set $FN(w)$ is clearly invariant under $\alpha$-equivalence, so we have a well-defined notion of free names of bar strings modulo $\equiv_\alpha$.

Note that standard regular expressions are just regular bar expressions of degree 0.

**Remark 3.2.** Sequential composition of bar strings is not capture-avoiding, so $\alpha$-equivalence fails to be a congruence. E.g. $|a| \equiv_\alpha b$ but $|oa| \not\equiv_\alpha |ba|$. For similar reasons, there is no straightforward notion of $\alpha$-equivalence on regular bar expressions; e.g. renaming the bound name $a$ in $(a|a)^*$ will require first expanding the expression to $1 + a|a(a|a)^*$ (only then renaming to, say, $1 + ab(b|b)^*$).

We will define the semantics of regular bar expressions in terms of bar languages. These are fs sets of bar strings modulo $\alpha$-equivalence, i.e. fs subsets of $\check{\check{M}} := \check{\check{A}}^*/\equiv_\alpha$. 

6
We can convert every bar string into a $\nu$-string by replacing any occurrence of $ta$ with $\nu a.a$, with the scope of the binder extending to the end of the string. By the equational laws of NKA, this map induces an equivariant bijection

$$\bar{M} = \bar{h}^*/\equiv_{\alpha} \cong M_{\nu}. \quad (1)$$

That is, bar languages are essentially the same as the $\nu$-languages used in the semantics of NKA we consider here; in particular, $P_{\bar{h}}(M)$ carries a final coalgebra for the functor $K$ defined in Section 2.

For technical purposes, we will consider two types of languages generated by an expression or automaton $X$ in the sequel:

- **Bar languages** (equivalently, $\nu$-languages) are sets of bar strings modulo $\alpha$-equivalence, denoted $L_{\alpha}(X)$.
- **Literal languages** are subsets of $\bar{h}^*$, denoted $L_0(X)$ and inducing bar languages via $L_{\alpha}(X) = \{[w]_{\alpha} \mid w \in L_0(X)\}$.

In Section 6, we will discuss a third type of language for purposes of comparison with formalisms for data languages.

**Definition 3.3.** The literal language $L_0(r) \subseteq \bar{h}^*$ defined by regular bar expression $r$ is just the language defined by $r$ as a regular expression over $\bar{h}$. The bar language $L_{\alpha}(r)$ defined by $r$ then consists of the $\alpha$-equivalence classes of the words in $L_0(r)$.

**Lemma 3.4.** For every regular bar expression $r$, $L_{\alpha}(r)$ is ufs.

**Remark 3.5.** The previous lemma implies that neither $\bar{M}$ nor the set $\bar{M}_0 \subseteq \bar{M}$ of all closed bar strings modulo $\equiv_{\alpha}$ are definable by a regular bar expression. Consequently, no regular bar expression can be complemented within either $\bar{M}$ or $\bar{M}_0$.

**Remark 3.6.** The semantics of regular bar expressions is inherently non-compositional; specifically, there is no way to calculate $L_{\alpha}(rs)$ from $L_{\alpha}(r)$ and $L_{\alpha}(s)$ since $r$ may bind names that are free in $s$ but due to formation of $\alpha$-equivalence classes, $L_{\alpha}(r)$ contains no information about these names. However, the semantics can be defined by coinduction; see Appendix B.

**Example 3.7.** Under the isomorphism (1), the $\nu$-language of Example 2.3 translates to the bar language

$$\{\epsilon, \bar{ba}, \bar{ba}ab, \bar{ba}ab\bar{ba}, \bar{ba}ab\bar{ba}ab\ldots\}$$

(again eliding equivalence classes), which can be defined by the regular bar expression

$$(\bar{ba}ab)^*(1 + \bar{ba}).$$

**Remark 3.8.** Our Kleene theorem will imply that regular bar expressions are (by the previous example: strictly) more expressive than NKA. It is possible to define a direct recursive translation of NKA expressions to regular bar expressions but it seems surprisingly hard to do this without doubly exponential blowup, while the translation via automata (see Remark 4.21 for details) has only singly exponential blowup.
4 Regular Nondeterministic Nominal Automata

We next define the nominal automaton model that corresponds to regular bar expressions. We introduce the model directly with all requisite finiteness conditions, refraining for the time being from discussing the ramifications of weakening these conditions.

**Definition 4.1.** A regular nondeterministic nominal automaton (RNNA) is a pointed orbit-finite coalgebra for the functor $N$ on $\text{Nom}$ given by

$$NX = 2 \times \mathcal{P}_{ufs}(\mathbb{A} \times X) \times \mathcal{P}_{ufs}([\mathbb{A}]X),$$

i.e. for the variant of the functor $K$ defining deterministic nominal automata that arises by adding ufs-branching nondeterminism.

Recall that ufs subsets of orbit-finite sets are finite (Lemma 2.2). Explicitly, an RNNA can thus be described as a tuple $A = (Q, \rightarrow, s, F)$ consisting of

- an orbit-finite set $Q$ of states;
- an equivariant subset $\rightarrow$ of $Q \times \mathbb{A} \times X$, the transition relation, where we write $q \xrightarrow{a} q'$ for $(q, a, q') \in \rightarrow$;
- an equivariant subset $F \subseteq Q$ of final states; and
- an initial state $s \in Q$,

satisfying the following conditions:

- The relation $\rightarrow$ is $\alpha$-invariant, i.e. $q \xrightarrow{b} q'$ and $(\alpha)q' = (\beta)q''$ imply $q \xrightarrow{\beta} q''$.
- The relation $\rightarrow$ is finitely branching up to $\alpha$-equivalence, i.e. for each state $q$ the sets $\{(a, q') | q \xrightarrow{a} q'\}$ and $\{(a)q' | q \xrightarrow{a} q'\}$ are finite.

Transitions of type $q \xrightarrow{a} q'$ are called free, and those of type $q \xrightarrow{a} q'$ bound. Given a bar string $w = \alpha_1 \cdots \alpha_n \in \mathbb{A}^*$, we write $q \xrightarrow{w} q'$ if there is a path $q = q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} q_n = q'$ in $A$. We say that $A$ accepts $w$ if $s \xrightarrow{w} q$ for some $q \in F$. Acceptance of bar strings is clearly invariant under $\alpha$-equivalence. The bar language accepted by $A$ is the set $L_{\alpha}(A)$ of bar strings accepted by $A$, modulo $\alpha$-equivalence. More generally, given a state $q$ in $A$ we write $L_{\alpha}(q)$ for the language accepted by the automaton obtained by making $q$ the initial state of $A$.

**Lemma 4.2.** The map $q \mapsto L_{\alpha}(q)$ is equivariant.

**Remark 4.3.** There is an expressivity gap between deterministic and nondeterministic nominal automata [12, Example 4.13].

The key property of RNNAs is that supports of states evolve in the expected way along transitions (cf. [12, Lemma 4.6] for the deterministic case):

**Lemma 4.4.** Let $A$ be an RNNA. Then the following hold.

1. If $q \xrightarrow{a} q'$ in $A$ then $\text{supp}(q') \cup \{a\} \subseteq \text{supp}(q)$.
2. If $q \xrightarrow{\alpha} q'$ in $A$ then $\text{supp}(q') \subseteq \text{supp}(q) \cup \{a\}$.

In fact, the properties in the lemma are clearly also sufficient for ufs branching. From Lemma 4.4, we immediately have

**Corollary 4.5.** Let $A$ be a RNNA. Then $L_{\alpha}(A)$ is ufs; specifically, if $s$ is the initial state of $A$ and $w \in L_{\alpha}(A)$, then $\text{supp}(w) \subseteq \text{supp}(s)$. 
4.1 From Bar Automata to Nominal Automata

Although orbit-finite, RNNAs remain infinite objects. We next provide a finite representation of RNNAs in terms of ordinary nondeterministic finite automata (NFAs) over $\tilde{\mathbb{A}}$.

**Definition 4.6.** A nondeterministic finite bar automaton, or bar NFA for short, over $\mathbb{A}$ is an NFA $A$ over $\tilde{\mathbb{A}}$ that is live, i.e. every state in $A$ accepts some word (this clearly does not restrict expressivity). We call transitions of type $q \xrightarrow{a} q$ in $A$ free, and transitions of type $q \xrightarrow{a} q'$ bound. The literal language $L_0(A)$ of $A$ is the language accepted by $A$ qua NFA over $\tilde{\mathbb{A}}$. The bar language $L_\alpha(A)$ accepted by $A$ is defined as

$$L_\alpha(A) = L_0(A)/\equiv_\alpha.$$  

Generally, we denote by $L_0(q)$ the $\tilde{\mathbb{A}}$-language accepted by the state $q$ in $A$, and by $L_\alpha(q)$ the quotient of $L_0(q)$ by $\alpha$-equivalence. The degree of $A$ is the number of $a \in \mathbb{A}$ such that $q \xrightarrow{a} q'$ for some $q, q'$ in $A$.

**Lemma 4.7.** Let $q$ be a state in a bar NFA; then $L_\alpha(q)$ is ufs.

We first present the construction of an RNNA $\tilde{A}$ from a given bar NFA $A$. We let

$$H_q = \text{Fix}(\text{supp}(L_\alpha(q))) \quad \text{and} \quad K_q = G/H_q$$

where by $G/H_q$ we denote the set of left cosets of $H_q$. Note that left cosets for $H_q$ can be identified with injective renamings $\text{supp}(L_\alpha(q)) \to \tilde{\mathbb{A}}$. The states of $\tilde{A}$ are pairs

$$(q, \pi H_q)$$

consisting of a state $q$ in $A$ and a left coset $\pi H_q \in K_q$. We let $G$ act on left cosets by left multiplication, and on states by $\pi_1 \cdot (q, \pi_2 H_q) = (q, \pi_1 \pi_2 H_q)$. The initial state of $\tilde{A}$ is $(s, H_s)$ where $s$ is the initial state of $A$; a state $(q, \pi H_q)$ is final in $\tilde{A}$ iff $q$ is final in $A$.

Free transitions in $\tilde{A}$ are of the form

$$(q, \pi H_q) \xrightarrow{\pi(a)} (q', \pi H_{q'}) \quad \text{where} \quad q \xrightarrow{a} q',$$

and bound transitions are of the form

$$(q, \pi H_q) \xrightarrow{b} (q', \pi' H_{q'}) \quad \text{where} \quad q \xrightarrow{b} q' \quad \text{and} \quad \langle a \rangle \pi' H_{q'} = \langle (\pi(b)) \rangle H_{q'}.$$

**Lemma 4.8.** $\tilde{A}$ is an RNNA, and has as many orbits as $A$ has states.

**Example 4.9.** The language $(lb \mid a \mid b)^*(1 + lb)$ from Example 3.7 can be accepted by the bar NFA $A$ with four states $s, t, u, v$, where $s$ is initial and $s$ and $u$ are final, and transitions $s \xrightarrow{b} t \xrightarrow{a} u \xrightarrow{b} v \xrightarrow{b} s$. The above construction produces an RNNA very similar to the one shown for this example in [12]: By the above description of left cosets for $H_q$, we annotate every state $q$ with $\sharp \text{supp}(L_\alpha(q))$ distinct names in the transition.
from $A$ to $\bar{A}$, i.e. one name for $s$ and $u$, and two names for $t$ and $v$. We can draw the orbits of the resulting RNNA in the form

$$s(c) \xleftarrow{b} t(c, b) \xrightarrow{c} u(b) \xleftarrow{v(b, c)} v(c, b)$$

(for $b \neq c$), with $s(c)$ and $u(b)$ final for all $b, c \in A$, and $s(a)$ the initial state. The automaton given in [12] has fewer orbits, being obtained from the above by identifying $s(c)$ with $u(c)$ and $t(c, b)$ with $v(c, b)$.

We need to show that $\bar{A}$ accepts $L_\alpha(A)$. We have an evident notion of $\alpha$-equivalence of paths in an RNNA, defined analogously as for bar strings. Of course, $\alpha$-equivalent paths always start in the same state. The following normalization result for paths is crucial.

**Definition 4.10.** A path in $\bar{A}$ is $\pi$-literal for $\pi \in G$ if all states in it are of the form $(q, \pi H_q)$. Intuitively, a $\pi$-literal path is one that uses the same pattern of name reusage for free and bound names as the underlying path in $A$, up to a joint renaming $\pi$ of the free and bound names.

**Lemma 4.11.** Let $P$ be a path in $\bar{A}$ beginning at $(q_0, \pi_0 H_{q_0})$. Then $P$ is $\alpha$-equivalent to a $\pi_0$-literal path.

**Corollary 4.12.** The RNNA $\bar{A}$ accepts the bar language $L_\alpha(A)$.

**Proof.** The inclusion $L_\alpha(A) \subseteq L_\rho(A)$ is clear as $A$ is contained in $\bar{A}$ via $q \mapsto (q, H_q)$. For the reverse inclusion, note that by Lemma 4.11, every accepting path of $\bar{A}$ is $\alpha$-equivalent to an $id$-literal accepting path starting at the initial state $(s, H_s)$ of $\bar{A}$. ⊓ ⊔

**Definition 4.13.** Let $A$ be an RNNA, and $A_0$ a bar NFA contained in $A$. Then $A_0$ generates $A$ if $A_0$ is not contained in a proper sub-RNNA of $A$; that is, all states of $A$ are of the form $\pi \cdot q$ for some state $q$ in $A_0$, similarly for the free transitions, and all bound transitions of $A$ are $\alpha$-equivalent to transitions of the form $\pi \cdot q \xrightarrow{a} \pi \cdot q'$ where $q \xrightarrow{a} q'$ in $A_0$. We say that $A$ has degree at most $d$ if it can be generated from a bar NFA of degree $d$ in this sense.

### 4.2 From Nominal Automata to Bar Automata

We next present the reverse construction, i.e. given an RNNA $A$ we extract a bar NFA $A_0$ (a subautomaton of $A$) such that $L_\alpha(A_0) = L_\alpha(A)$.

Since $A$ is orbit-finite, the supports of the states in $A$ are of bounded size, say at most $k$. We fix a set $\mathcal{A}_0 \subseteq A$ of size $|\mathcal{A}_0| = k$ such that $\text{supp}(s) \subseteq \mathcal{A}_0$ for the initial state $s$ of $A$, and a name $* \in A - \mathcal{A}_0$. The states of $A_0$ are those states $q$ in $A$ such that $\text{supp}(q) \subseteq \mathcal{A}_0$. 

10
As this implies that the set \( Q_0 \) of states in \( A_0 \) is ufs, \( Q_0 \) is finite by Lemma 2.2. Note that \( s \in Q_0 \). For \( q, q' \in Q_0 \), the free transitions \( q \xrightarrow{a} q' \) in \( A_0 \) are the same as in \( A \) (hence have \( a \in A_0 \) by Lemma 4.4.1). The bound transitions \( q \xrightarrow{b} q' \) in \( A_0 \) are those bound transitions \( q \xrightarrow{b} q' \) in \( A \) such that \( a \in A_0 \cup \{*\} \). A state is final in \( A_0 \) iff it is final in \( A \).

The initial state of \( A_0 \) is \( s \).

**Theorem 4.14.** We have \( L_\alpha(A_0) = L_\alpha(A) \).

**Proof (Sketch).** The key observation is that \( Q_0 \) is closed under free transitions in \( A \), and closed up to \( \alpha \)-equivalence under bound transitions \( q \xrightarrow{b} q' \). The latter is shown by a case distinction on whether \( b \in A_0 \) (the positive case being clear) and, in case \( b \notin A_0 \), additionally on whether \( b \in \text{supp}(q') \). In case \( b \notin A_0 \) and \( b \in \text{supp}(q') \), we have \( |\text{supp}(q') \cap A_0| < k \), so that we can pick a name \( a \in A_0 \) that is fresh for \( q' \) and rename \( b \) into \( a \).

**Example 4.15.** Recall that after identification of \( s(b) \) with \( u(b) \) and \( t(c, b) \) with \( v(b, c) \), the RNNA constructed in Example 4.9 has states \( s(b) \) and \( t(c, b) \) for distinct \( b, c \in A \), with \( s(b) \) final for all \( b \) and \( s(a) \) initial. So we pick \( A_0 = \{a, b\} \) and obtain by the above construction a bar NFA with four states \( s(a), s(b), t(a, b), \) and \( t(b, a) \), with \( s(a) \) initial and \( s(b) \) final. Up to renaming \( v \) into \( t \), \( u \) into \( s \), and \( c \) into \( a \), the graphical display of this automaton is as in Example 4.9 but is now to be understood as showing all states of a bar NFA instead of just representatives of the orbits of an RNNA.

In combination with the construction from the previous section, we obtain

**Theorem 4.16.** RNNA are expressively equivalent to bar NFAs.

While it might seem that we can now just give up nominal automata and use bar NFAs instead, it turns out that our decision procedure for containment will actually use both concepts, essentially running a bar NFA in synchrony with an RNNA.

**Remark 4.17.** The number of states in the bar NFA \( A_0 \) constructed from an RNNA \( A \) as above is linear in the number of orbits of \( A \) and exponential (only) in the maximal size \( k \) of supports of states in \( A \).

### 4.3 A Kleene Theorem

From the equivalence of RNNA and bar NFAs (Theorem 4.16) and the classical Kleene theorem, the announced Kleene theorem for regular bar expressions is now immediate:

**Corollary 4.18 (Kleene theorem for regular bar expressions).** Regular bar expressions and RNNA are expressively equivalent, i.e. a bar language can be accepted by some RNNA iff it is definable by a regular bar expression.

**Definition 4.19.** Bar languages satisfying the equivalent conditions of Corollary 4.18 are called regular.

**Corollary 4.20.** The class of regular bar languages is closed under finite intersections.
Remark 4.21. It has been shown by Kozen et al. [12] that a given NKA expression \( r \) can be translated into a nondeterministic nominal automaton whose states are the so-called spines of \( r \). The nominal set of spines is orbit-finite, with exponentially many orbits. The transitions of the automaton are given by a form of Antimirov derivative; in particular, the automata are, up to \( \alpha \)-equivalence, finitely branching. In summary, NKA expressions can be translated into RNNAs; this shows that regular bar expressions are (strictly) more expressive than NKA expressions.

5 Decidability and Upper Complexity Bounds

For any formalism defining \( \nu \)-languages or, equivalently, bar languages, we refer to the problem of checking whether \( L_\alpha(X) \subseteq L_\alpha(Y) \) for given expressions or automata \( X, Y \) as the containment problem. We proceed to show that the containment problem for bar regular expressions is in \( \text{EXPSPACE} \), the same as the best known upper bound for containment of NKA expressions [12]. This reduces immediately to a corresponding result about the containment problem for bar NFAs.

Theorem 5.1. The containment problem for bar NFAs is in \( \text{EXPSPACE} \); more precisely, the containment \( L_\alpha(A_1) \subseteq L_\alpha(A_2) \) can be checked using space polynomial in the size of \( A_1 \) and \( A_2 \), and exponential in \( \deg(A_2) \log(p) \) where \( p \) is the number of names occurring literally in \( A_1 \) (i.e. \( p = \deg(A_1) + 2\text{FN}(A_1) \)).

The theorem can be rephrased as saying that bar language containment of NFA is in parameterized polynomial space (para-PSpace) [19], with the parameter being essentially the degree. In particular, we obtain the classical PSPACE bound on containment of NFAs as the special case for degree 0.

Proof. Let \( A_1, A_2 \) be bar NFAs. We exhibit a nondeterministic exponential space procedure to check that \( L_\alpha(A_1) \) is not contained in \( L_\alpha(A_2) \), which implies the claimed bound by Savitch’s theorem. The algorithm is inspired by the spine-based algorithm from [12]. It maintains a state \( q \) of \( A_1 \) and a set \( \Xi \) of states in the RNN \( \hat{A}_2 \) generated by \( A_2 \) as described in Section 4.1, with \( q \) initialized to the initial state of \( A_1 \), and \( \Xi \) to the singleton set containing the initial state of \( A_2 \). It then iterates the following:

- Guess a transition \( q \xrightarrow{\alpha} q' \) in \( A_1 \).
- Update \( q \) to \( q' \).
- Compute the set \( \Xi' \) of all states of \( \hat{A}_2 \) reachable from states in \( \Xi \) via \( \alpha \)-transitions (literally, i.e. not up to \( \alpha \)-equivalence).
- Update \( \Xi \) to \( \Xi' \).

The algorithm terminates successfully (reporting that \( L_\alpha(A_1) \not\subseteq L_\alpha(A_2) \)) if it reaches a final state \( q \) of \( A_1 \) while \( \Xi \) contains only non-final states.

Correctness of the algorithm is clear by Lemma 4.11 and Corollary 4.12. To see that it uses only exponential space, note that by Lemma 4.4, \( \Xi \) will only ever contain states \( (q, \pi H_q) \) (where \( H_q = \text{Fix}(\text{supp}(L_\alpha(q))) \)) such that \( \pi \) fixes all names in \( \text{supp}(L_\alpha(q)) \) that do not appear bound anywhere in \( A_2 \), and renames the others (at most \( \deg(A_2) \) many) into names occurring literally in \( A_1 \). But \( \hat{A}_2 \) only has exponentially many such states, specifically at most \( k \cdot 2^\deg(A_2) = k \cdot 2^{\deg(A_2) \log(p)} \) where \( k \) is the number of orbits in \( A_2 \) and \( p \) is as in the claim. \( \square \)
Corollary 5.2. Language containment of regular bar expressions is in \( \text{EXPSPACE} \); more precisely, the containment \( L_\alpha(r_1) \subseteq L_\alpha(r_2) \) can be checked using space polynomial in the size of \( r_1 \) and \( r_2 \), and exponential in \( \deg(r_2) \log(e) \) where \( e \) is the number of names occurring literally in \( r_1 \).

**Proof.** Bar regular expressions can be polynomially translated into bar NFAs, preserving free and bound names. \( \square \)

**Remark 5.3.** The translation from NKA expressions to regular bar expressions that arises via our Kleene theorem has exponential blowup in terms of expression size (Remarks 4.17 and 4.21). The spines of an NKA expression \( r \) arise by \( \alpha \)-renaming and subsequent deletion of some binders from expressions that use only bound variables occurring already in \( r \); therefore, the RNNA formed by the spines can be generated from a bar NFA whose degree is at most that of \( r \). It follows that the translation does not increase the degree. Therefore, the \( \text{EXPSPACE} \) upper bound on containment for NKA expressions proved in [12] follows, via their translation into regular bar expressions, from Corollary 5.2. Note however that for NKA expressions, space usage is exponential already in the size of the expression (here as well as in [12]).

### 6 Data Languages

We emphasize again that we intend all formalisms discussed so far primarily for the definition of bar languages (equivalently \( \nu \)-languages). We can however use them also to define languages over the infinite alphabet \( A \), which for brevity we will call \( A \)-languages in the following. Specifically, we can extract from a bar language \( L \) the \( A \)-language

\[
D(L) = \{ \text{ub}(w) \mid [w]_\alpha \in L \}.
\]

That is, \( D(L) \) is obtained by taking all representatives of \( \alpha \)-equivalence classes in \( L \) and removing all bars. In fact, the operator \( N \) defining the original semantics of NKA (Section 2) also yields a (different) \( A \)-language; we discuss the comparison in Remark 6.8.

Note that the semantics \( D(L_\alpha(\cdot)) \) is coarser than \( L_\alpha(\cdot) \); e.g. we have, eliding equivalence classes, \( L_\alpha(a + [a]) = \{a, [a]\} \neq \{[a]\} = L_\alpha([a]) \) but \( D(L_\alpha(a + [a])) = A = D(L_\alpha([a])) \).

**Example 6.1.** Under \( D(L_\alpha(\cdot)) \), the regular expression \((lbabab)^*(1 + lbab)\) (Example 3.7) defines the \( A \)-language consisting of all even-length words over \( A \) that contain a in the second position (if any) and repeat every letter in an odd position three positions later.

As discussed in the introduction, there are numerous formalisms for \( A \)-languages and, more generally, data languages. Under the semantics \( D(L_\alpha(\cdot)) \), regular bar expressions can be seen as a sublanguage of regular expressions with memory [15] and of Kaminski and Tan’s regular expressions over infinite alphabets [11]. In the following, we discuss in more detail how, under the semantics \( D(L_\alpha(\cdot)) \), RNNA compare to non deterministic orbit-finite automata (NOFA) in the sense of Bojańczyk et al. [2] (considered there in the setting of arbitrary symmetries). NOFAs are orbit-finite coalgebras for the functor \( G \) on \( \text{Nom} \) given by

\[
GX = 2 \times \mathcal{P}_a(A \times X),
\]
equipped with an equivariant subset of initial states. They thus differ from RNNA in the following respects:

– Instead of a single initial state, NOFAs have an equivariant set of initial states (and hence accept equivariant languages, i.e. equivariant subsets of the set of all strings).
– NOFAs have no bound transitions (so do not meaningfully accept bar languages).
– In NOFAs, there is no requirement that the sets \( \{(a,q') \mid q \xrightarrow{a} q'\} \) be ufs.
– The alphabet of a NOFA can be any orbit-finite set.

We regard the differences concerning initial states and, to some degree, alphabets as relatively minor; to enable a comparison, we restrict for the rest of this section to RNNA that are closed, i.e. whose initial state has empty support, and to NOFAs over the alphabet \( \mathcal{A} \). Notice that closed RNNA accept equivariant \( \mathcal{A} \)-languages. Every NOFA \( A \) has an underlying classical (although infinite) nondeterministic automaton \( U(A) \) (with a set of initial states), and the \( \mathcal{A} \)-language accepted by \( A \) is just that accepted by \( U(A) \). We refer to a NOFA whose transition relation is deterministic as a DOFA.

We can convert a closed RNNA \( A \) into a NOFA \( D(A) \) accepting the same \( \mathcal{A} \)-language by simply replacing every transition \( q \xrightarrow{a} q' \) with a transition \( q \xrightarrow{a} q' \). We show that the image of this translation is a natural class of NOFAs:

**Definition 6.2.** A NOFA is non-spontaneous if

\[ \text{supp}(q') \subseteq \text{supp}(a,q) \text{ whenever } q \xrightarrow{a} q'. \]

That is, a NOFA is non-spontaneous if its \( a \)-transitions do not create new names other than \( a \) in the target state. It is easy to see that non-spontaneous NOFAs are coalgebras for the subfunctor of \( G \) taking \( X \) to \( 2 \times \mathcal{P}_{ufs}(X)^{\mathcal{A}} \).

**Proposition 6.3.** A NOFA is of the form \( D(B) \) for some RNNA \( B \) iff it is non-spontaneous.

For distinction from containment of bar languages as discussed in Section 5, we refer to containment of \( \mathcal{A} \)-languages as weak containment. We next extend the algorithm from Section 5 to weak containment; to this end, we need the following characterization of weak containment.

**Definition 6.4.** For \( \alpha, \beta \in \mathcal{A} \), we write \( \alpha \sqsubseteq \beta \) if \( \alpha = \beta \) or \( \alpha = a, \beta = |a| \) for some \( a \in \mathcal{A} \). We extend \( \sqsubseteq \) to words \( w_1, w_2 \in \mathcal{A}^* \) by putting \( w_1 \sqsubseteq w_2 \) if \( w_1 \) and \( w_2 \) have the same length and are letterwise related by \( \sqsubseteq \).

**Lemma 6.5.** Let \( L_1, L_2 \) be regular bar languages. Then \( D(L_1) \subseteq D(L_2) \) iff for each \( [w]_\alpha \in L_1 \) there exists \( w' \sqsubseteq w \) such that \( [w']_\alpha \in L_2 \).

**Corollary 6.6.** Weak containment for regular bar expressions is decidable in EXPSPACE; more precisely, the containment \( D(L_\alpha(r_1)) \subseteq D(L_\alpha(r_2)) \) can be checked using space polynomial in the size of \( r_1 \) and \( r_2 \), and exponential in \( \deg(r_2) \log(e) \) where \( e \) is the number of names occurring literally in \( r_1 \).

**Proof.** Equivalently consider bar NFAs. By Lemma 6.5, we can use essentially the same algorithm as in Section 5. The only modification is to let \( \Xi' \) additionally contain also states of \( \mathcal{A} \) reachable from states in \( \Xi \) via \( 0a \)-transitions in case \( \alpha \) is a free name \( a \).
Remark 6.7. Again, the above result says that data language containment of regular bar expressions is in para-\textsc{pspace}. Contrastingly, (weak) containment and even universality of unrestricted NOFAs are undecidable – this follows from their equivalence to finite memory automata [2] and the undecidability of the corresponding problems for the latter [16]. This implies that in terms of data languages, NOFAs are strictly more expressive than RNNAs. Moreover, RNNAs are strictly more expressive than DOFAs (which, in turn, are strictly more expressive than orbit-finite nominal monoids [?]) – every DOFA is non-spontaneous, by equivariance of the transition map; and the data language 'the last letter has been seen before' is defined by the regular bar expression \((b)^*la(lb)^*a\) but not accepted by any DOFA. The decidability part of Corollary 6.6, but not as far as we can see the upper complexity bound, follows from decidability of containment for Kaminski and Tan’s regular expressions over infinite alphabets [11].

Remark 6.8. The original semantics of NKA, \(N(L_\alpha(\cdot))\), is finer than the \(A\)-language semantics \(D(L_\alpha(\cdot))\) considered above, that is, for regular bar expressions \(r, s\) we have that \(N(L_\alpha(r)) = N(L_\alpha(s))\) implies \(D(L_\alpha(r)) = D(L_\alpha(s))\) but not conversely. We prove the former claim in the appendix. For the latter, \(r = \lambda a b\) and \(s = \lambda a b + c c\) provide a counterexample: we have \(D(L_\alpha(r)) = \mathcal{A}_2 = D(L_\alpha(s)) = N(L_\alpha(s))\) but \(N(L_\alpha(r)) = \{ab \in \mathcal{A}_2 \mid a \neq b\}\), recalling that \(N\) imposes genuine freshness of bound names.
Observe that under \(N\), the NKA expression \((\nu a.a)^*\) defines the \(A\)-language \(L\) consisting of all words that do not repeat any letter, a language presumably not recognizable in any automaton model with finite memory. In particular, \(L\) is not recognizable by a NOFA [1]; incidentally, the complement of \(L\) is, under \(D(L_\alpha(\cdot))\), definable by a regular bar expression (and even in NKA), \((b)^*la(lb)^*a(lb)^*\).

7 Conclusions and Future Work

We have introduced regular bar expressions, which extend standard regular expressions with a construct for fresh name allocation. We have shown these expressions to be equivalent to a natural nominal automaton model with binding transitions, so-called regular nondeterministic nominal automata (RNNAs). The key technical ingredient is a representation of RNNAs in terms of standard nondeterministic finite automata (NFAs) over an extended alphabet. Regular bar expressions are comparatively well-behaved computationally, and in particular admit containment checking in parametrized polynomial space.

We leave the implementation of our calculus, possibly transferring efficient methods for equivalence checking of NFAs using bisimulation up to congruence [3] to the nominal setting, as future work. Another challenge is to add support for deallocation operators in the spirit of dynamic sequences [7] to the framework.

Acknowledgements We wish to thank Charles Paperman for useful discussions.

References


A Omitted Proofs

Proof of Lemma 2.2
Firstly, any finite set \( S \subseteq X \) is ufs, because \( \bigcup_{y \in S} \text{supp}(y) \) is a finite union of finite sets. Secondly, for any ufs \( S \subseteq X \), we have \( \text{supp}(S) = \bigcup_{y \in S} \text{supp}(y) \), which is a finite union (because \( X \) is orbit-finite) of again finite sets.

Proof of Lemma 3.4
Only the names that appear free in \( r \) can appear free in words in \( L_\alpha(r) \).

Proof of Lemma 4.2
Let \( \pi \in G \), \( w = \alpha_1 \cdots \alpha_n \in M \) (eliding equivalence classes in the notation), and \( q \in Q \). Then \( w \in \pi(L_\alpha(q)) \) iff \( \pi^{-1}w \in L_\alpha(q) \) iff there is a sequence of transitions
\[
q = q_0 \xrightarrow{\pi^{-1}\alpha_1} q_1 \xrightarrow{\pi^{-1}\alpha_2} \cdots \xrightarrow{\pi^{-1}\alpha_{n-1}} q_{n-1} \xrightarrow{\pi^{-1}\alpha_n} q_n
\]
with \( q_n \) accepting, iff (by equivariance of the transition relation)
\[
\pi q = \pi q_0 \xrightarrow{\alpha_1} \pi q_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{n-1}} \pi q_{n-1} \xrightarrow{\alpha_n} \pi q_n
\]
with \( \pi q_n \) accepting, iff \( w \in L_\alpha(\pi q) \). This implies that \( L \) is equivariant.

Proof of Lemma 4.4
1. Since \( Z = \{(a,q') \mid q \xrightarrow{\alpha} q'\} \) is ufs, we have \( \text{supp}(q') \cup \{a\} = \text{supp}(a,q') \subseteq \text{supp}(Z) \subseteq \text{supp}(q) \).
2. Since \( Z = \{[a]q' \mid q \xrightarrow{\alpha} q'\} \) is ufs, we have \( \text{supp}(q') \subseteq \text{supp}([a]q') \cup \{a\} \subseteq \text{supp}(Z) \cup \{a\} \subseteq \text{supp}(q) \cup \{a\} \).

Proof of Lemma 4.7
The finitely many transitions of \( A \) only mention letters from a finite subset of \( \bar{A} \), and \( \bigcup_{w \in L_\alpha(q)} \text{supp}(w) \) is contained in that finite subset.

Proof of Lemma 4.8
Remark A.1. (1) Note that by Lemma 2.1, \( \text{supp}(L_\alpha(q)) = \bigcup_{w \in L_\alpha(q)} \text{supp}(w) \), and this is the set of all those names that appear free in some word \( w \in L_\alpha(q) \).
(2) The coset \( \pi H_q \) has the finite support \( \pi \text{supp}(L_\alpha(q)) \), and thus \( K_q \) is a nominal set.
(3) For future use we observe that if \( \pi \) and \( \pi' \) yield the same left coset, i.e. \( \pi H_q = \pi' H_q \), then \( \pi(v) = \pi'(v) \) for all \( v \in \text{supp}(L_\alpha(q)) \). Indeed, there is some \( \rho \in H_q \) with \( \pi = \pi' \circ \rho \). By the definition of \( H_q \) we have \( \rho(v) = v \) whence \( \pi(v) = \pi'(v) \) for all \( v \in \text{supp}(L_\alpha(q)) \).
Proof (Lemma 4.11). The free and bound transitions are equivariant, and the bound transitions $\alpha$-invariant, by construction of the transition relation on $\tilde{A}$. The claim on the number of orbits of $\tilde{A}$ is immediate from the construction of $\tilde{A}$, and implies that $\tilde{A}$ is orbit finite. Moreover, whenever $q \xrightarrow{a} q'$ in $\tilde{A}$ then $a \in \text{supp}(\alpha(q))$ (otherwise, $\alpha(q') = \emptyset$, contradicting liveness of $\tilde{A}$). Therefore, a state $(q, \pi H_q)$ in $\tilde{A}$ only has as many free transitions as $q$ has transitions for free letters $a$ in $A$, for every transition $q \xrightarrow{\alpha} q'$ yields only one transition $(q, \pi) \xrightarrow{\pi(a)} (q', \pi)$; indeed, if $(q, \pi) = (q, \pi')$ then we have $\pi(a) = \pi'(a)$ by Remark A.1(3) since $a \in \text{supp}(\alpha(q)).$

Similarly, $q$ only has as many bound transitions modulo $\alpha$-equivalence as $q$ has transitions for bound letters $\bar{a}$ in $A$.

\begin{proof}[Proof of Lemma 4.11]
We make the definition of $\alpha$-equivalence on paths in an RNNA precise:

**Definition A.2.**

$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n$ is $\alpha$-equivalent to $q_0' \xrightarrow{a_1'} q_1' \xrightarrow{a_2'} \ldots \xrightarrow{a_n'} q_n'$, and

$q_0 \xrightarrow{\bar{a}} q_1 \xrightarrow{\bar{a}_2} \ldots \xrightarrow{\bar{a}_n} q_n$ is $\alpha$-equivalent to $q_0' \xrightarrow{\bar{a}_1'} q_1' \xrightarrow{\bar{a}_2'} \ldots \xrightarrow{\bar{a}_n'} q_n'$,

if $\langle a \rangle [q_1 \xrightarrow{\alpha_2} q_2 \xrightarrow{\alpha_3} \ldots \xrightarrow{\alpha_n} q_n] = \langle b \rangle [q_1' \xrightarrow{\alpha_2'} q_2' \xrightarrow{\alpha_3'} \ldots \xrightarrow{\alpha_n'} q_n']$, where we use $[-]_\alpha$ to denote $\alpha$-equivalence classes of paths.

**Lemma A.3.** Let $P = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n$ be a path in an RNNA $A$, and let $\langle a \rangle q_1 = \langle b \rangle q_1'$. Then there exists a path in $A$ of the form $q_0 \xrightarrow{\psi} q_1' \xrightarrow{\bar{a}_2'} \ldots \xrightarrow{\bar{a}_n'} q_n'$ that is $\alpha$-equivalent to $P$.

**Proof.** Since $A$ is an RNNA, the support of the $\alpha$-equivalence class of $q_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} q_n$ is $\text{supp}(q_1)$ (Remark A.4), so we obtain an $\alpha$-equivalent path $q_0 \xrightarrow{\psi} q_1' \xrightarrow{\bar{a}_2'} \ldots \xrightarrow{\bar{a}_n'} q_n'$ by renaming $a$ into $b$ in $q_1 \xrightarrow{\bar{a}_2} \ldots \xrightarrow{\bar{a}_n} q_n$. \hfill $\square$

**Remark A.4.** Note that the support of an $\alpha$-equivalences class of paths in an RNNA $A$ is the support of its starting state. Indeed, let $\text{supp}(P) = \{q(\alpha(x)) \mid \alpha \in \text{supp}(P)\}$ where $\alpha$ is the (equivariant) structure map of the initial algebra for $FX \cong Q \times (1 + \Lambda \times X + [\Lambda]X_1)$ and $\text{supp}(q, x) = \text{supp}(q) \cup \text{supp}(x)$. The converse equation follows from this using Lemma 4.4 and induction.

**Proof (Lemma 4.11).** We prove the statement by induction over the path length. The base case is trivial. For the inductive step, let $P = (q_0, \pi_0 H_{q_0}) \xrightarrow{\alpha_1} (q_1, \pi_1 H_{q_1}) \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} (q_n, \pi_n H_{q_n})$ be a path of length $n + 1$. If $\alpha_1$ is a free name then $\pi_1 H_{q_1} = \pi_0 H_{q_0}$ by construction of $\tilde{A}$; by induction hypothesis, we can assume that the length-$n$ path from
We have to show that every accepting path in \((q_1, \pi_0 H_{q_1})\) onward is \(\pi_0\)-literal, and hence the whole path is \(\pi_0\)-literal. If \(\alpha_1 = |a\) then we have \((a, (q_1, \pi_1 H_{q_1})) \equiv_{\alpha} (\pi_0(b), (q_1, \pi_0 H_{q_1}))\) for some transition \(q_0 \xrightarrow{b} q_1\) in \(A\).

By Lemma A.3, this induces an \(\alpha\)-equivalence of \(P\) with a path \((q_0, \pi_0 H_{q_0}) \xrightarrow{\pi_0(b)} (q_1, \pi_0 H_{q_1}) \rightarrow \ldots\); by the induction hypothesis, we can transform the length-\(n\) path from \((q_1, \pi_0 H_{q_1})\) onward into a \(\pi_0\)-literal one, so that the whole path becomes \(\pi_0\)-literal as desired.

**Full Proof of Theorem 4.14**

We have to show that every accepting path in \(A\) is \(\alpha\)-equivalent to an accepting path in \(A_0\). Note that \(Q_0\) is closed under free transitions in \(A\), so by Lemma A.3, it suffices to show that for every bound transition \(q \xrightarrow{a} q'\) in \(A\) with \(q \in Q_0\) we find an \(\alpha\)-equivalent transition \(q \xrightarrow{a} q''\) in \(A_0\). We distinguish the following cases.

- If already \(b \in A_0\) then \(\text{supp}(q') \subseteq \text{supp}(q) \cup \{b\} \subseteq A_0\), so \(q' \in Q_0\) and we are done.
- If \(b \notin A_0\) and \(b \notin \text{supp}(q')\) then \(\text{supp}(q') \subseteq \text{supp}(q) \subseteq A_0\). In particular, \(q'\) is already in \(Q_0\) and \(\ast\) is fresh for \(q'\), so we can rename \(b\) into \(\ast\) and obtain an \(\alpha\)-equivalent transition \(q \xrightarrow{\ast} q'\) in \(A_0\).
- If \(q \notin A_0\) and \(b \in \text{supp}(q')\) then \(|\text{supp}(q') \cap A_0| < k\), so that we can pick a name \(a \in A_0\) that is fresh for \(q'\). We put \(q'' = (ab)q'\); then \((b)q' = (a)q''\), and \(q'' \in Q_0\) because \(\text{supp}(q'') = \{a\} \cup (\text{supp}(q') \setminus \{b\}) \subseteq \{a\} \cup \text{supp}(q) \subseteq A_0\); thus, \(q \xrightarrow{a} q''\) is a transition in \(A_0\).

**Proof of Corollary 4.20**

By a standard construction of product automata. The only point to note beyond the classical case is that cartesian products of orbit-finite sets are again orbit-finite [2].

**Non-Spontaneous NOFAs are \(2 \times \mathcal{P}_{ufs}\)-Coalgebras**

This claim is immediate from the following:

**Lemma A.5.** A NOFA is non-spontaneous iff for each state \(q\) and each \(a \in \mathcal{A}\), the set \(\{q' \mid q \xrightarrow{a} q'\}\) is ufs.

‘Only if’: non-spontaneity states that \(\text{supp}(a, q)\) is a uniform finite support of \(\{q' \mid q \xrightarrow{a} q'\}\).

‘If’: The map \((q, a) \mapsto \{q' \mid q \xrightarrow{a} q'\}\) is equivariant, hence \(\text{supp}\{q' \mid q \xrightarrow{a} q'\} \subseteq \text{supp}(q, a)\). Since \(\text{supp}\{q' \mid q \xrightarrow{a} q'\}\) is ufs, this implies using Lemma 2.1 \(\text{supp}(q') \subseteq \text{supp}(a, q)\) whenever \(q \xrightarrow{a} q'\).
Proof of Proposition 6.3

‘Only if’ is immediate by Lemma 4.4. To see ‘if’, let $A$ be a non-spontaneous NOFA. We construct an RNNA $B$ with the same states as $A$, as follows.

- $q \xrightarrow{a} q'$ in $B$ iff $q \xrightarrow{a} q'$ in $A$ and $a \in \text{supp}(q)$.

- $q \xrightarrow{\text{wa}} q'$ in $B$ iff $q \xrightarrow{\text{wa}} q''$ in $A$ for all $b,q''$ such that $\langle b \rangle q'' = \langle a \rangle q'$ and $b \sharp q$.

The transition relation thus defined is clearly equivariant and $\alpha$-invariant, so $B$ is an NNA. Regularity of $B$ is immediate from non-spontaneity of $A$.

It remains to verify that $D(B) = A$, i.e. that

$$q \xrightarrow{a} q' \text{ in } A \iff (q \xrightarrow{a} q' \text{ or } q \xrightarrow{\text{wa}} q' \text{ in } B).$$

Here, ‘if’ is immediate from the construction of $B$. To see ‘only if’, let $q \xrightarrow{a} q'$ in $A$, $\langle b \rangle q'' = \langle a \rangle q'$, and $b \sharp q$; we have to show that $q \xrightarrow{b} q''$ in $A$. But this is immediate from equivariance of $\to$ in $A$. $lacksquare$

Proof of Lemma 6.5

We shortly write $D(w) = D(L_\alpha(w)) = \{ub(w') \mid w' \equiv_\alpha w\}$ for $w \in \bar{\mathbb{A}}^\ast$. The lemma is immediate from the following:

Lemma A.6. Let $L$ be a regular bar language, and let $w \in \bar{\mathbb{A}}^\ast$. Then $D(w) \subseteq D(L)$ iff there exists $w' \sqsupseteq w$ such that $[w]_\alpha \in L$.

Proof. ‘If’ is clear; we prove ‘only if’. We generalize the claim to state that whenever

$$D(w) \subseteq \bigcup_{i \in I} D(L_\alpha(q_i))$$

for states $q_i$ in an RNNA $A$ and a finite index set $I$, then there exist $i$ and $w' \sqsupseteq w$ such that $[w']_\alpha \in L_\alpha(q_i)$. We prove the generalized claim by induction over $w$, with trivial base case.

Induction step for words $aw$: Let $D(aw) \subseteq \bigcup_{i=1}^n D(L_\alpha(q_i))$. Since $D(aw) = aD(w)$, we have

$$D(w) \subseteq \bigcup_{i \in I, q_i \xrightarrow{\alpha} q', \alpha \in \{a,aw\}} D(L_\alpha(q'))$$

where the union is finite because $A$ is an RNNA. By induction, we thus have $i \in I$, $\alpha \in \{a,aw\}$, $q_i \xrightarrow{\alpha} q'$, and $w' \sqsupseteq w$ such that $[w']_\alpha \in L_\alpha(q_i)$. Then $\alpha w \equiv aw$ and $[\alpha w]_\alpha \in L_\alpha(q_i)$, as required.

Induction step for words $|aw$: Let $D(|aw) \subseteq \bigcup_{i=1}^n D(L_\alpha(q_i))$. Notice that

$$D(|aw) = \bigcup_{b=a\vee b\sharp[w]_\alpha} bD(\pi_{ab} \cdot w)$$

(where $\cdot$ denotes the permutation group action and $\pi_{ab} = (a \ b)$ the transposition of $a$ and $b$). Now pick $b \in \mathbb{A}$ such that $b\sharp[w]_\alpha$ and none of the $q_i$ has a $b$-transition (such a $b$
exists because the set of free transitions of each $q_i$ is finite, as $A$ is an RNNA). Then the assumption $D(la w) \subseteq \bigcup_{i=1}^{n} D(L_{\alpha}(q_i))$ implies that necessarily

$$bD(\pi_{ab} \cdot w) \subseteq \bigcup_{i \in I, q_i \rightarrow q'} bD(L_{\alpha}(q')), \quad \text{and hence}$$

$$D(\pi_{ab} \cdot w) \subseteq \bigcup_{i \in I, q_i \rightarrow q'} D(L_{\alpha}(q')).$$

By induction, we thus have $i \in I$, $q_i \rightarrow b$ and $w' \sqsubseteq \pi_{ab} \cdot w$ such that $[w']_{\alpha} \in L_{\alpha}(q_i)$. It follows that $[bw'] \sqsupseteq [b(\pi_{ab} \cdot w)]$ and $[bw']_{\alpha} \in L_{\alpha}(q_i)$. Now we have $a[\pi_{ab} \cdot w]_{\alpha}$ (because $b[\pi_{ab} \cdot w]_{\alpha}$), and therefore $a[w']_{\alpha}$ because $\pi_{ab} \cdot w \sqsubseteq w'$; it follows that $[a[w']]_{\alpha} = [bw']_{\alpha} \in L_{\alpha}(q_i)$, as required.

**Details for Remark 6.7**

We show that the data language

$$L = \{ wava \mid w, v \in A^*, a \in A \}$$

is not accepted by any DOFA. Assume for a contradiction that $A$ is a DOFA that accepts $L$. Let $n$ be the maximal size of a support of a state in $A$. Let $w = a_1 \ldots a_{n+1}$ for distinct $a_i$, and let $q$ be the state reached by $A$ after consuming $w$. Then there is $i \in \{1, \ldots, n+1\}$ such that $a_i \notin \text{supp}(q)$. Pick a fresh name $b$. Then $\delta(a_i, q)$ is final and $\delta(b, q)$ is not; but since $\delta(a_i, q) = (a_i b) \cdot \delta(b, q)$, this is in contradiction to equivariance of the set of final states.

**Details for Remark 6.8**

We have to show that $D(L_1) \subseteq D(L_1)$ for regular bar languages $L_1, L_2$ whenever $N(L_1) \subseteq N(L_2)$. We generalize $N$ to an operator $N(B, L)$ taking as an argument a finite set $B$ of names to be avoided in $\alpha$-renaming bound names; that is, $N(B, L)$ consists of all $ub(w)$ such that $[w]_{\alpha} \in L$ and $w$ is clean (i.e. its bound names are mutually distinct and distinct from its free names) and does not use bound names from $B$. Note that in this notation, $N(L) = N(\emptyset, L)$. Moreover, we introduce a variant of the relation $\sqsubseteq$: for bar strings $w, w'$ we write

$$w \sqsubseteq_B w'$$

if $w'$ arises from $w$ by adding bars $|$ to occurrences of names $a$ in $w$ such that $a \notin B$ and neither $a$ nor $|a$ occurs further to the left in $w$. We then have the following variant of Lemma A.6:
Lemma A.7. Let $L$ be a regular bar language, and let $w \in \bar{A}^*$. If $N(w) \subseteq N(L)$ then there exists $w' \supseteq_B w$ such that $[w']_\alpha \in L$.

Proof. We generalize the claim to state that whenever

$$N(B, w) \subseteq \bigcup_{i \in I} N(B, L_\alpha(q_i))$$

for states $q_i$ in an RNNA $A$, a finite index set $I$, and a finite set $B$ of names, then there exist $i$ and $w' \supseteq_B w$ such that $[w']_\alpha \in L_\alpha(q_i)$. We prove the generalized claim by induction over $w$, with trivial base case.

Induction step for words $aw$: Let $N(B, aw) \subseteq \bigcup_{i=1}^n N(B, L_\alpha(q_i))$. Since $N(B, aw) = aN(B \cup \{a\}, w)$ by using cleanliness of $aw$, we have

$$N(B \cup \{a\}, w) \subseteq \bigcup_{i \in I, q_i \triangleright_{\alpha, q'} \alpha \in \{a, aw\}} N(B \cup \{a\}, L_\alpha(q'))$$

where the union is finite because $A$ is an RNNA. By induction, we thus have $i \in I$, $\alpha \in \{a, |a\}$, $q_i \triangleright_{\alpha, q'} q'$, and $w' \supseteq_{B \cup \{a\}} w$ such that $[w']_\alpha \in L_\alpha(q')$. Then $aw' \supseteq_B aw$ and $[\alpha w']_\alpha \in L_\alpha(q_i)$, as required.

Induction step for words $aw$: Let $N(B, aw) \subseteq \bigcup_{i=1}^n N(B, L_\alpha(q_i))$. Notice that

$$N(B, aw) = \bigcup_{b=a\lor b[w]_\alpha} bN(B \cup \{b\}, \pi_{ab} \cdot w)$$

(where $\cdot$ denotes the permutation group action and $\pi_{ab} = (a \ b)$ the transposition of $a$ and $b$). Now pick $b \in \bar{A}$ such that $b[w]_\alpha$, $b \notin B$, and none of the $q_i$ has a $b$-transition (such a $b$ exists because the set of free transitions of each $q_i$ is finite, as $A$ is an RNNA). Then the assumption $N(B, aw) \subseteq \bigcup_{i=1}^n N(B, L_\alpha(q_i))$ implies that necessarily

$$bN(B \cup \{b\}, \pi_{ab} \cdot w) \subseteq \bigcup_{i \in I, q_i \triangleright_{\alpha, q'} q'} bN(B \cup \{b\}, L_\alpha(q'))$$

and hence

$$N(B \cup \{b\}, \pi_{ab} \cdot w) \subseteq \bigcup_{i \in I, q_i \triangleright_{\alpha, q'} q'} N(B \cup \{b\}, L_\alpha(q'))$$

By induction, we thus have $i \in I$, $q_i \triangleright_{\alpha, q'} q'$ and $w' \supseteq_{B \cup \{b\}} \pi_{ab} \cdot w$ such that $[w']_\alpha \in L_\alpha(q')$. It follows that

$$[bw']_\alpha \in L_\alpha(q_i) \quad \text{and} \quad [bw']_\alpha \in L_\alpha(q_i).$$

Now we have $\pi_{ab} \cdot w \supseteq_{B \cup \{b\}} w'$ (because $b[w]_\alpha$, and therefore $\pi_{ab} \cdot w \supseteq_{B \cup \{b\}} w'$; it follows that $[bw']_\alpha \in L_\alpha(q)$, and hence $[\alpha \pi_{ab} \cdot w']_\alpha \in L_\alpha(q_i)$. Moreover, since $\subseteq_B$ is clearly equivariant as a relation on words and finite sets of names, we obtain from $\pi_{ab} \cdot w \subseteq_{B \cup \{b\}} w'$ that

$$[\alpha \pi_{ab} \cdot w'] \subseteq_{B \cup \{a\} \cup \{b \mid a \in B\}} \text{law},$$

which implies $[\alpha \pi_{ab} \cdot w'] \supseteq_B \text{law}$ as required. □
Corollary A.8. If \( N(L_1) \subseteq N(L_2) \) for regular bar languages \( L_1, L_2 \), then \( D(L_1) \subseteq D(L_2) \).

Proof. Let \( w \in L_1 \); it suffices to show that \( D(w) \subseteq D(L_2) \). We have \( N([w]_\alpha) \subseteq N(L_1) \subseteq N(L_2) \). By Lemma A.7, we obtain \( w' \parallel_\emptyset w \) such that \([w']_\alpha \in L_2\). Then also \( w' \parallel w \), so \( D(w) \subseteq D(L_2) \) by Lemma A.6. \( \square \)

B Coinductive Semantics of Regular Bar Expressions

Although the semantics of bar expressions is not compositional, it can be defined coinductively. Let \( \text{RBExp} \) denote the nominal set of bar expressions, with an action of \( G \) defined by \( \pi \cdot [a] = [\pi(a)] \), \( \pi \cdot a = \pi(a) \) and commutation of the action with all other connectives. The semantics is defined by specifying an equivariant coalgebra \( \text{RBExp} \rightarrow K \text{RBExp} \) in terms of its three components, and then applying the finality of the \( K \)-coalgebra \( \mathcal{P}_\omega(M) \). The first component is \( e : \text{RBExp} \rightarrow 2 \), telling whether a expression contains the empty bar string:

\[
e(1) = 1, \quad e(0) = e(a) = e([a]) = 0, \quad e(r + s) = \max(e(r), e(s)), \quad e(rs) = \min(e(r), e(s)), \quad e(r^*) = 1.
\]

All cases are invariant under permutation of atoms, so \( e \) is equivariant. The second map is \( (-)_a : \text{RBExp} \rightarrow \text{RBExp} \) for \( a \in A \), namely the left derivation.

\[
a_a = 1, \quad x_a = 0, \quad (r + s)_a = r_a + s_a, \quad (rs)_a = r_a s + e(r)s_a, \quad r_a^* = r_a r_a^*
\]

with \( x \in \{1, 0, \emptyset, b, c\} \) and \( c \neq a \). Again, all cases (in particular \( c \neq a \)) are invariant under permutation of atoms, so \( r, a \mapsto r_a \) is equivariant. Note that \( e(r) \in \{0, 1\} \) is to be read as a bar expression here. For the third map, we first need a helper function \( g : \text{RBExp} \rightarrow \mathcal{P}_\omega(\mathbb{A} \times \text{RBExp}) \). Intuitively, \( g(r) \) returns all ways to reach a bar, namely returns a pair consisting of the letter under the bar and the expression after the bar.

\[
g(1) = \emptyset, \quad g(0) = \emptyset, \quad g(b) = \emptyset, \quad g([b]) = \{(b, 1)\},
\]

\[
g(rs) = \{ (b, r's) \mid (b, r') \in g(r) \} \cup \{ (b, s') \mid (b, s') \in g(s), e(r) = 1 \}
\]

\[
g(r + s) = g(r) \cup g(s), \quad g(r^*) = \{ (b, r'r^*) \mid (b, r') \in g(r) \}
\]

Using \( g \), we can define the left derivation under \( [a] \) or the bound transitions as

\[
r_{1a} = \sum_{(b, r') \in g(r)} (a \cdot b) \cdot r'
\]

Lemma B.1. The maps \( e, (r, a) \mapsto r_a, g, \) and \( (r, a) \mapsto r_{1a} \) are equivariant.

Proof. The induction is trivial for \( e \) and \( r_a \), because the case matching is invariant under the permutation of atoms. It remains to show that \( g \) is equivariant, by induction on \( r \). For \( x \in \{0, 1, b\} \) we have

\[
\sigma \cdot [g(x)] = \sigma \cdot \emptyset = \emptyset = g(\sigma \cdot x).
\]
For the case \( \emptyset \), we have: \( \sigma \cdot g(\emptyset) = \{ \sigma \cdot (b, 1) \} = \{ (\sigma \cdot b, 1) \} = g(\sigma(\emptyset)) \).

Summation is obvious: \( \sigma \cdot g(r + s) = \sigma \cdot g(r) \cup \sigma \cdot g(s) \).

For composition:

\[
\sigma \cdot g(rs) = \{ \sigma(b), \sigma \cdot r's \mid (b, r') \in g(r) \} \cup \{ \sigma(b), \sigma \cdot s' \mid (b, s') \in g(s), e(r) = 1 \}
= \{ (\bar{b}, \bar{r}(\sigma \cdot s)) \mid (\bar{b}, \bar{r}) \in \sigma \cdot g(r) \} \cup \{ (\bar{b}, \bar{s}) \mid (\bar{b}, \bar{s}) \in \sigma \cdot g(s), e(r) = 1 \}
\]

\[
= g(\sigma \cdot r \cdot s)
\]

The last case \( r^* \) is just a combination of \( + \) and composition:

\[
\sigma \cdot g(r^*) = \{ (\sigma(b), \sigma \cdot r' r^*) \mid (b, r') \in g(r) \} = \{ \bar{b}, \bar{r}(\sigma \cdot r^*) \mid (\bar{b}, \bar{r}) \in \sigma \cdot g(r) \}
\]

\[
= g((\sigma \cdot r^*)) = g(\sigma \cdot r^*) \quad \square
\]

This means that we have a \( K \) coalgebra structure on \( \text{RBExp} \) in \( \text{Nom} \) and thus obtain a unique \( K \) coalgebra homomorphism into the final \( K \) coalgebra on \( \mathcal{P}_\text{fs}(M) \).

Our previously defined semantics consists in first interpreting a regular bar expression like a standard regular expression, then obtaining a literal language, and then forming \( \alpha \) equivalence classes. Denote the literal semantics by \( L_0 : \text{RBExp} \to \mathcal{P}_\text{fs}(\mathbb{A}^*) \), and the quotient map \( \mathbb{A}^* \to M \) by \( [-]_\alpha \). Thus, our original semantics is \( \mathcal{P}_\text{fs}[\cdot]_\alpha \circ L_0 \); we have to show that this agrees with the coinductive definition. We have a \( K \) coalgebra structure on \( \mathcal{P}_\text{fs}(\text{RBExp}) \), which is the usual for emptiness and free transitions. Bound transitions are defined as:

\[
X_{\alpha} = \{ (a b) \cdot v \mid bv \in X \} \quad \text{for } X \in \mathcal{P}_\text{fs}(\mathbb{A}^*)
\]

**Lemma B.2.** The map \( L_0 : \text{RBExp} \to \mathcal{P}_\text{fs}(\mathbb{A}^*) \) is a \( K \) coalgebra morphism.

**Proof.** First, \( L_0 \) is equivariant, because the nominal structure on \( \mathcal{P}_\text{fs}(\mathbb{A}^*) \) is just the restricted nominal structure of \( \text{RBExp} \), applied point-wise. Second, \( L_0 \) is a \( K \) coalgebra morphism, using that \( L_0 \) is compositional, i.e. \( L_0(rs) = L_0(r)L_0(s) \):

\begin{itemize}
  \item The definition of \( e \) precisely tells whether an ordinary regular expression accepts the empty word.
  \item For free transitions, we have to show that \( L_0(r_a) = L_0(r)_a \), by induction on \( r \). But that is easy, because all the cases of \( (\cdot)_a \) are identities on ordinary regular expressions. E.g. composition:
    \[
    L_0((rs)_a) = L_0(r_a s + e(r)s_a) = L_0(r_a)L_0(s) \cup L_0(e(r)L_0(s)_a)
    \]
    \[
    = L_0(r_a)L_0(s) \cup \{ x \in L_0(s) \mid e \in L_0(r) \} = L_0(rs)_a
    \]
  \item For bound transitions, the first thing to prove is that for arbitrary \( r, b \):
    \[
    bv \in L_0(r) \quad \text{if and only if} \quad (b, r') \in g(r) \quad \text{and} \quad v \in L_0(r') \quad (2)
    \]
\end{itemize}
by induction on \( r \). The cases 1, 0, \( a \in A \) are trivial, because both statements are false for such an \( r \). For composition, we have:

\[
\begin{align*}
|bv| &\in L_0(rs) \overset{\text{Def}}{=} \bigvee_{L_0} \{ |bw| \in L_0(r), y \in L_0(s), v = wy \mid \varepsilon \in L_0(r), |bv| \in L_0(s) \} \\
&\overset{\text{IH}}{=} \bigvee_{L_0} \{ (b, r') \in g(r), w \in L_0(r'), y \in L_0(s), v = wy \mid \varepsilon \in L_0(r), (b, s') \in g(s), v \in L_0(s') \} \\
&\overset{\text{IH}}{=} \bigvee_{L_0} \{ (b, r') \in g(r), v \in L_0(r's') \mid e(r) = 1, (b, s') \in g(s), v \in L_0(s') \} \\
&\overset{\text{Def}}{=} \bigvee_{g(r')} (b, z) \in g(rs), v \in L_0(z)
\end{align*}
\]

The case for Kleene star is similar:

\[
\begin{align*}
|bv| &\in L_0(r^*) \overset{\text{Def}}{=} \overline{|bv| \in L_0(rr^*)} \lor \overline{|bv| \in L_0(\varepsilon)} \\
&\overset{\text{Def}}{=} \bigvee_{L_0} \{ |bu| \in L_0(r), w \in L_0(r^*), v = uw \} \\
&\overset{\text{IH}}{=} \bigvee_{L_0} \{ (b, r') \in g(r), w \in L_0(r^*), v \in L_0(r') \mid (b, r') \in g(r), v \in L_0(r^*) \} \\
&\overset{\text{Def}}{=} \bigvee_{g(r^*)} (b, r'r^*) \in g(r^*), v \in L_0(r'r^*)
\end{align*}
\]

Note that the case for \( r + s \) is trivial, because both \( L_0 \) and \( g \) are just defined as the union of their recursive calls on \( r \) and \( s \), so in total we have (2).

Now, we easily verify:

\[
L_0(r)_a = \{ (a \ b) \cdot v \mid |bv| \in L_0(r) \} = \{ (a \ b) \cdot v \mid (b, r') \in g(r), v \in L_0(r') \} = \bigcup_{(b, r') \in g(r)} (a \ b) \cdot L_0(r') = L_0(\prod_{a} r)
\]

**Lemma B.3.** The map \( \prod_{a} [\cdot]_a : \prod_{a} (A^*) \to \prod_{a} (M) \) is a \( K \)-coalgebra homomorphism.

**Proof.** The coalgebra structure on \( \prod_{a} (M) \) [12] has been defined on representatives in such a way that it is compatible with \( \alpha \)-equivalence.

In total, the originally defined semantics \( L = \prod_{a} [\cdot]_a \cdot L_0 \) on \( \text{RBExp} \) is a \( K \)-coalgebra homomorphism \( \text{RBExp} \to \prod_{a} (M) \), and by the finality of \( \prod_{a} (M) \), it is identical to the coinductively defined semantics.