A Characterisation of Languages on Infinite Alphabets with Nominal Regular Expressions

Alexander Kurz, Tomoyuki Suzuki, Emilio Tuosto

A Characterisation of Languages on Infinite Alphabets with Nominal Regular Expressions

Alexander Kurz, Tomoyuki Suzuki, and Emilio Tuosto

Department of Computer Science, University of Leicester, UK

Abstract. We give a characterisation of languages on infinite alphabets in a variant of nominal regular expressions with permutations (p-NREs). We also introduce automata with fresh name generations and permutations (fp-automata), inspired by history-dependent automata (HDAs) and fresh-register automata. Noteworthy, permutations require to deal with dynamic context-dependent expressions. Finally, we give a Kleene theorem for p-NREs and fp-automata to formally characterise languages on infinite alphabets.

1 Introduction

The study of languages on infinite alphabets has been pushed by the need of formalising data structures built on top of infinite domains of values [16]. In this context, it is natural to appeal to the theory of automata [11, 19, 3, 22] operating on a countably infinite alphabet $\mathcal{N}$ of *names* to express languages of interest such as $\mathcal{L}_1$ below, see [22],

$$\mathcal{L}_1 = \{n_1 \dots n_k \in \mathcal{N}^* \;\mid\; \forall i \,.\, 1 \leq i < k \,.\, n_i \neq n_{i+1}\}$$

In [13], we extended this line of investigations to languages where words do not only consist of names but also of binders (e.g. lambda-calculus terms). In particular, [13] studies a notion of regular expression for words with binders and the associated notion of finite automata, aiming at applications to the design and analysis of programming languages (as in [20] or [18]) or to verification and testing.

Here, we move back to languages of words without binders and apply the techniques of [13] in order to obtain a novel notion of regular expression for languages on infinite alphabets. While being built from concatenation, sum, and Kleene star in the usual way, the nominal regular expressions introduced in this paper may also contain a binder

$$\langle_n \mathsf{ne} \rangle_n^m \tag{1}$$

Intuitively, $\langle_n \mathsf{ne}$ allocates a fresh resource within ne, whereas $\rangle_n^m$ deallocates it. The crucial new ingredient, which allows us to capture for example the language $\mathcal{L}_1$ above, is that $\rangle_n^m$ permutes n and m before deallocating n. For example, we can read

$$\langle_m \, (\langle_n n \rangle_n^m)^* \, \rangle_m^m \tag{2}$$

as follows. First $\langle_m$ allocates a name m, then $\langle_n$ allocates a fresh (i.e. different) name n. Now the permutation specified by $\rangle_n^m$ makes sure that it is the first name m that is

deallocated and the second name n that is retained, before looping back to the beginning as specified by the Kleene star *.

Note that permutations have a slightly subtle effect on concatenation. For example, in $\langle_m \langle_n n\rangle_n^m \circ \mathsf{ne}\rangle_m^m$, due to the permutation specified by $\rangle_n^m$, the name m in ne semantically represents n. So it will take some care, in § 3, to describe how nominal regular expressions give rise to nominal regular languages. In § 4 we introduce the corresponding notion of automata with fresh-name generation and permutations (fp-automata). Like HD-automata they have allocation transitions, corresponding to $\langle_n$, but unlike HD-automata permutations only appear in special deallocation transitions corresponding to $\rangle_n^m$. In § 5 we prove a Kleene-type theorem stating that the languages accepted by fp-automata are precisely the nominal regular languages.

Finally, we argue that the work nominal languages with binders also sheds new light on well-established work on languages on infinite alphabets.

Related Work Apart from languages on infinite alphabets, our research draws on HD-automata [17, 14] and nominal sets [9], as well as on the recent confluence of these three areas in [6, 22, 3, 8, 4, 13]. Moreover, the way we deal with scope and binders is related to nested words [1] and automata working on lambda-calculus terms [21].

Although our paper does neither require nor use the theory of nominal sets, the work on nominal sets does suggest to view automata as (co)algebras in the category of nominal sets. As nominal sets are themselves permutation algebras one would expect regular expressions, as it is the case in our work, to contain permutations, see [15].

A form of regular expressions, called UB-expressions, for languages on infinite alphabets investigated in [10]. There it is shown that UB-expressions are as expressive as finite-state unification based automata (FSUBA), which are somewhat weaker than the finite-memory automata (FMA) of [11]. In a nutshell, FSUBA do not account for freshness (they can, for example, accept the first but not the second language of Example 6 in § 3.2). Moreover, UB-expressions do not have permutations, although their device of labelling the Kleene-star allows them to accept $\mathcal{L}_1$ in a similar fashion to ours.

2 Motivating examples: languages on infinite alphabets

Languages on infinite alphabets can suitably formalise data structures on infinite domains (take data words in [16, 5] as an example). In this context, typical languages consist of finite words — that is finite sequences of symbols — whose symbols are possibly drawn from an infinite set. This is illustrated in the next example.

Example 1 ([22]). Assume that $a, a', a_1, \ldots$ range on an infinite set A. The language

$$\mathcal{L}_1 \stackrel{\text{def}}{=} \{a_1 \cdots a_k \;\mid\; k \geq 0 \wedge \forall i \in \{1, \ldots, k-1\}.\, a_i \neq a_{i+1}\}$$

consists of finite sequences on A in which all two consecutive letters are different. $\diamond$

In general, the infinite alphabet has a very simple structure that permits to test just for equality or inequality of symbols (see e.g., [16]) as in Example 1 (in recent work this is being reconsidered, see § 6 for a discussion). For practical reasons, it is sometimes

convenient to consider words whose letters can be taken either from an infinite alphabet or from a disjoint finite set, as in Example 2.

The next example shows how languages on infinite alphabets could be used to represent computations where resources have to be acquired and released before their usage.

Example 2. Assume that $a, a', a_1, \ldots$ range on an infinite set A while α (for "acquire") and ρ ("for release") are two distinguished symbols not in A. The language

$$\mathcal{L}_{\alpha\rho} \stackrel{\text{def}}{=} \alpha A^* \rho \cup \bigcup_{a \neq a'} \{\alpha \underbrace{a \cdots a}_{i \geq 0 \text{ times}} \alpha\, a_1 \cdots a_j\, \rho \underbrace{a' \cdots a'}_{h \geq 0 \text{ times}} \rho \mid j \geq 0 \wedge \forall 1 \leq r \leq j . a_r \in \{a, a'\}\}$$

represents the executions of a process that acquires, uses, and then releases either one or two resources. ◇

3 Nominal regular expressions with permutations

In order to characterise languages on infinite alphabets, we extend regular expressions with name-abstraction and permutation. Let $\mathcal{N}$ be an infinite set of names and $\mathcal{S}$ a finite set of letters. We assume that $\mathcal{N}$ and $\mathcal{S}$ are disjoint. A *language on* $\mathcal{N} \cup \mathcal{S}$ is a subset of $(\mathcal{N} \cup \mathcal{S})^*$, namely a collection of *words on infinite alphabets*.

The nominal regular expressions with permutations (*p-NREs*) are given by

$$\mathsf{ne} ::= \mathit{1} \mid \mathit{0} \mid n \mid s \mid \mathsf{ne} + \mathsf{ne} \mid \mathsf{ne} \circ \mathsf{ne} \mid \mathsf{ne}^* \mid \langle_n \mathsf{ne} \rangle^m_n$$

where *1* denotes the singleton with the empty word, *0* denotes the empty language, n ranges over $\mathcal{N}$, s ranges over $\mathcal{S}$; the operators $+$, $\circ$, and $\_^*$ are as the classical operators of regular expressions, while $\langle_n \mathsf{ne} \rangle^m_n$ is a binder. Note that the superscript m on the closing bracket is not bound unless it is the same as the subscript. Closed and bound occurrences of names are defined in the natural way and we call *closed* any p-NRE with no occurrence of free names.

Intuitively, $\langle_n \mathsf{ne}$ allocates a *fresh resource* within ne; as in nominal calculi, this is rendered by declaring a *fresh local name* n. Novel is here that $\rangle^m_n$ specifies a permutation when disposing the resource denoted by n, to the effect that to the right of $\rangle^m_n$ every syntactic occurrence of m is semantically read as n. One can think of n and m as registers whose contents are swapped when n is deallocated. To make this work, we assume that the superscript m on $\rangle^m_n$ is in the syntactic scope of some $\langle_m$. For example, the p-NRE $\langle_n \langle_m nm \rangle^n_m n \rangle^n_n$ is acceptable while $\langle_n \langle_m nm \rangle^l_m m \rangle^n_n$ is not, because the subexpression $\langle_m nm \rangle^l_m$ is not within a scope of a $\langle_l$, and the rightmost m is outside of the scope of $\langle_m$.

We aim to contribute to a foundational theory of interactions based on nominal calculi. This requires to consider interactions with an *environment* that can be seen as a resource handler for requiring and releasing resources. In this respect, the role of the execution environment can be suitably represented using *contexts*. In our theory, contexts capture two fundamental notions: one notion is *the fresh name generation* (read the environment) and the other is *the permutation action* (change the environment). To generate a fresh local name, we have to know which names are already in the environment. And, to leave the permutation result, we must update the environment which could be different from the original environment.

Our contexts are finite lists of names in $\mathcal{N}$. For a p-NRE ne, we call a triple $L \ddagger \mathsf{ne} \ddagger R$ an *expression-in-contexts*, where L and R are respectively called *pre-* and *post-context*. Intuitively, $L \ddagger \mathsf{ne} \ddagger R$ is an expression ne that is interpreted in the pre-context L and modifies it to the post-context R.

3.1 Preliminaries

Given a function f, the update $f_{[a \mapsto b]}$ extends $dom(f)$ to $dom(f) \cup \{a\}$ with $f(a) = b$; $\perp$ is the empty map. Let L, M, ... range over lists of $\mathcal{N}$ and let $lth(L)$ be the length of the list L. The empty list is denoted as $[]$. For $n \in \mathcal{N}$, write $n\#L$ (read "n is fresh for L,") when $n \neq l$ for any element l in L and write $L@n$ to be the list appending n to the tail of L. We consider only lists with no repeated elements. Given a list L we may abuse the notation and denote its underlying set by L.

The transposition of n and m, denoted by $(m\,n)$, is the permutation that swaps m and n and is the identity on any other names. Given two lists of the same length k, say $N = [n_1, \ldots, n_k]$ and $M = [m_1, \ldots, m_k]$, let $N \triangleright M$ be the bijection from N to M such that

$$N \triangleright M : n_i \mapsto m_i \qquad \text{for each } i \in \{1, \ldots, k\}$$

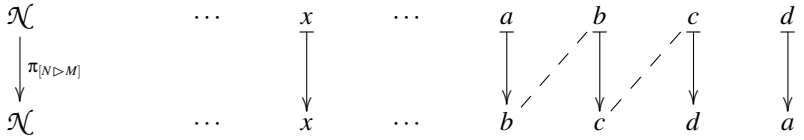
and define the bijection $\pi_{[N \triangleright M]}$ on $\mathcal{N}$ as

$$\pi_{[N \triangleright M]}(x) \stackrel{\text{def}}{=} \begin{cases} N \triangleright M(x), & \text{if } x \in N \\ N \triangleleft M(x), & \text{if } x \in M \setminus N \\ x, & \text{if otherwise} \end{cases}$$

where $N \triangleleft M(x)$ is a function from $M \setminus N$ to $N \setminus M$ recursively defined as follows:

$$N \triangleleft M(m_i) \stackrel{\text{def}}{=} \begin{cases} (N \triangleright M)^{-1}(m_i), & \text{if } (N \triangleright M)^{-1}(m_i) \notin N \setminus M \\ N \triangleleft M(m_j), & \text{if } (N \triangleright M)^{-1}(m_i) = m_j \text{ for some } j \neq i \end{cases}$$

For example, if $M = [b, c, d]$ and $N = [a, b, c]$, we have $\pi_{[N \triangleright M]}$ as follows:



where the target of d (defined by $N \triangleleft M$) is traced by going backwards along $\longmapsto$ and the dashed lines.

We define the action of a permutation π on p-NREs and on lists as follows. For a p-NRE ne, the permutation action of π on a p-NRE ne, denoted as $\pi \cdot \mathsf{ne}$, is

1. $\pi \cdot 1 = 1$; $\quad \pi \cdot 0 = 0$; $\quad \pi \cdot n = \pi(n)$; $\quad \pi \cdot s = s$
2. $\pi \cdot (\mathsf{ne}_1 + \mathsf{ne}_2) = (\pi \cdot \mathsf{ne}_1) + (\pi \cdot \mathsf{ne}_2)$
3. $\pi \cdot (\mathsf{ne}_1 \circ \mathsf{ne}_2) = (\pi \cdot \mathsf{ne}_1) \circ (\pi \cdot \mathsf{ne}_2)$
4. $\pi \cdot (\mathsf{ne}^*) = (\pi \cdot \mathsf{ne})^*$
5. $\pi \cdot (\langle_n \mathsf{ne} \rangle_n^m) = \langle_{\pi(n)} (\pi \cdot \mathsf{ne}) \rangle_{\pi(n)}^{\pi(m)}$

while, the permutation action of π on $L = [l_1, \ldots, l_k]$ is $\pi \cdot L = [\pi(l_1), \ldots, \pi(l_k)]$.

3.2 From p-NREs to languages on infinite alphabets

The interpretations of p-NREs depend on pre- and post-contexts, therefore we introduce the set of rules in Fig. 1.

$$\frac{L \ddagger \mathsf{ne}_1 + \mathsf{ne}_2 \ddagger R}{L \ddagger \mathsf{ne}_1 \ddagger R}\ (\hat{+}_1) \qquad \frac{L \ddagger \mathsf{ne}_1 + \mathsf{ne}_2 \ddagger R}{L \ddagger \mathsf{ne}_2 \ddagger R}\ (\hat{+}_2)$$

$$\frac{L \ddagger \mathsf{ne}_1 \circ \mathsf{ne}_2 \ddagger R}{L \ddagger \mathsf{ne}_1 \ddagger L \qquad L \ddagger \mathsf{ne}_2 \ddagger R}\ (\hat{\circ}) \qquad \frac{L \ddagger \mathsf{ne}^* \ddagger R}{L \ddagger \underbrace{\mathsf{ne} \circ \cdots \circ \mathsf{ne}}_{k \text{ times}} \ddagger R}\ (\hat{*})$$

$$\frac{L \ddagger \langle_n \mathsf{ne}\rangle^m_n \ddagger R \qquad m \neq n}{(L@\star) \ddagger (n\ \star) \cdot \mathsf{ne} \ddagger (((m\ \star) \cdot R)\ @m)}\ (\hat{\Diamond}_{\neq}) \qquad \frac{L \ddagger \langle_n \mathsf{ne}\rangle^m_n \ddagger R \qquad m = n}{(L@\star) \ddagger (n\ \star) \cdot \mathsf{ne} \ddagger (R@\star)}\ (\hat{\Diamond}_{=})$$

Fig. 1. Rules computing expressions-in-contexts

Given a closed p-NRE ne, we start applying the rules to the *expression-in-contexts* $[] \ddagger \mathsf{ne} \ddagger []$. In $(\hat{*})$ in Fig. 1, k is a natural number; if $k = 0$, the conclusion of the rule is $L \ddagger \mathit{1} \ddagger R$. Also, in $(\hat{\Diamond}_{=})$ and $(\hat{\Diamond}_{\neq})$, $\star$ denotes a name fresh for L and for R.

Fact 1 *For any derivation of $L \ddagger \mathsf{ne}' \ddagger R$ from an expression-in-contexts $[] \ddagger \mathsf{ne} \ddagger []$ using the rules in Fig. 1 we have that*

- *there is a permutation π such that $R = \pi \cdot L$ (and hence $lth(L) = lth(R)$)*
- *names in L are pairwise disjoint (and similarly for R)*

Example 3. Application of the rules in Fig. 1 to $\langle_m \langle_n m\rangle^m_n \langle_n nm\rangle^n_n\rangle^m_m$ gives:

$$(\hat{\Diamond}_{\neq})\ \frac{(\hat{\Diamond}_{=})\ \dfrac{[] \ddagger \langle_m \langle_n m\rangle^m_n \langle_n nm\rangle^n_n\rangle^m_m \ddagger []}{[a] \ddagger \langle_n a\rangle^a_n \langle_n na\rangle^n_n \ddagger [a]}}{[a] \ddagger \langle_n a\rangle^a_n \ddagger [a]}\ (\hat{\circ}) \qquad \frac{[a] \ddagger \langle_n na\rangle^n_n \ddagger [a]}{[a,c] \ddagger ca \ddagger [a,c]}\ (\hat{\Diamond}_{=})}{[a,b] \ddagger a \ddagger [b,a]} \qquad \frac{}{[a,c] \ddagger c \ddagger [a,c]} \qquad \frac{}{[a,c] \ddagger a \ddagger [a,c]}\ (\hat{\circ})$$

the derivation tree is read from top to bottom

Note that b and c are distinct from a, but b may be the same as c (Fact 1). Also, in the first step of the derivation, we can take n instead of a as a fresh name, yielding $[n] \ddagger \langle_m n\rangle^n_m \langle_m mn\rangle^m_m \ddagger [n]$ as the conclusion. ◇

By the rules of Fig. 1, there may be more than one *derivation tree for* $[] \ddagger \mathsf{ne} \ddagger []$ (one can choose either of the branches in a sum or unfold any number of times a Kleene-star). We associate a language to each derivation tree T. This is done by applying the rules in Fig. 2, starting from the leaves of T and going upwards to the root. Finally, we define *the language of* ne to be the union of the languages of all derivation trees for ne, and call such languages *nominal regular*.

To define the rules in Fig. 2, we extend the notation of expressions-in-contexts to languages and write e.g., $L \ddagger \mathbf{L}(\mathsf{ne}) \ddagger R$. Rules *(1)*, *(0)*, *(n)* and *(s)* yield the natural

$$\frac{L \ddagger 1 \ddagger R}{L \ddagger \{\varepsilon\} \ddagger R}\ (1) \qquad \frac{L \ddagger 0 \ddagger R}{L \ddagger \emptyset \ddagger R}\ (0) \qquad \frac{L \ddagger n \ddagger R}{L \ddagger \{n\} \ddagger R}\ (n) \qquad \frac{L \ddagger s \ddagger R}{L \ddagger \{s\} \ddagger R}\ (s)$$

$$\frac{L \ddagger \mathbf{L}(\mathsf{ne}_1) \ddagger M \qquad M' \ddagger \mathbf{L}(\mathsf{ne}_2) \ddagger R}{L \ddagger \{w \circ \left(\pi_{[M' \rhd M]} \cdot v\right) \mid w \in \mathbf{L}(\mathsf{ne}_1), v \in \mathbf{L}(\mathsf{ne}_2)\} \ddagger \left(\pi_{[M' \rhd M]} \cdot R\right)}\ (\check{\circ})$$

$$\frac{(L@n) \ddagger \mathbf{L}(\mathsf{ne}) \ddagger (R@m)}{L \ddagger \{(n\star) \cdot w \mid \star \in \mathcal{N}, w \in \mathbf{L}(\mathsf{ne}) \text{ and } \star \# L\} \ddagger ((n\star) \cdot R)}\ (\check{\Diamond})$$

Fig. 2. Rules computing languages

interpretation for basic expressions. Rule ($\check{\circ}$) deals with the concatenation of languages; note that, since permutations may change the post-context, it is necessary to rename everything by a permutation $\pi_{[M' \rhd M]}$ before combining them (recall $\pi_{[M' \rhd M]}$ from § 3.1). Also, the fact that the rule is applied on proof trees obtained by rules in Fig. 1 implies that $lth(M') = lth(M)$.

Rule ($\check{\Diamond}$) deallocates n. If $n \in R$, then $(n\star) \cdot R$ remembers the fresh name $\star$ in the new post-context. This rule maintains the invariant that the set of names in the pre-context is in bijection to the set of names in the post-context.

For simplicity, in Fig. 2 we do not explicitly consider e.g., rules for the $+$ operator; when such kind of nodes are reached, the computed language is just the language of the branch and similarly for the Kleene-star (cf. Example 5 below).

Example 4. Starting from the tree in Example 3, we calculate the language of the expression $\langle_m \langle_n m \rangle^m_n \langle_n nm \rangle^n_n \rangle^m_m$

$$\dfrac{\dfrac{\dfrac{(a)\ \dfrac{[a,b] \ddagger a \ddagger [b,a]}{[a,b] \ddagger \{a\} \ddagger [b,a]}}{\dfrac{[a] \ddagger \{(b\star_1) \cdot a \mid \star_1 \in \mathcal{N}, \star_1 \neq a\} \ddagger [\star_1]}{[a] \ddagger \{a \mid \star_1 \in \mathcal{N}, \star_1 \neq a\} \ddagger [\star_1]}\ (\check{\Diamond})} \qquad \dfrac{\dfrac{(c)\ \dfrac{[a,c] \ddagger c \ddagger [a,c]}{[a,c] \ddagger \{c\} \ddagger [a,c]} \qquad (a)\ \dfrac{[a,c] \ddagger a \ddagger [a,c]}{[a,c] \ddagger \{a\} \ddagger [a,c]}}{[a,c] \ddagger \{ca\} \ddagger [a,c]}\ (\check{\circ})}{\dfrac{[a] \ddagger \{(c\star_2) \cdot ca \mid \star_2 \in \mathcal{N}, \star_2 \neq a\} \ddagger [a]}{[a] \ddagger \{\star_2 a \mid \star_2 \in \mathcal{N}, \star_2 \neq a\} \ddagger [a]}\ (\check{\Diamond})}}{\dfrac{[a] \ddagger \{a \circ ((a\star_1) \cdot \star_2 a) \mid \star_1, \star_2 \in \mathcal{N}, \star_1 \neq a, \star_2 \neq a\} \ddagger [\star_1]}{\dfrac{[a] \ddagger \{a\star_2\star_1 \mid \star_1, \star_2 \in \mathcal{N}, \star_1 \neq a, \star_2 \neq \star_1\} \ddagger [\star_1]}{\dfrac{[] \ddagger \{(a\star_3) \cdot a\star_2\star_1 \mid \star_1, \star_2, \star_3 \in \mathcal{N}, \star_1 \neq a, \star_2 \neq \star_1\} \ddagger []}{[] \ddagger \{\star_3\star_2\star_1 \mid \star_1, \star_2, \star_3 \in \mathcal{N}, \star_1 \neq \star_3, \star_2 \neq \star_1\} \ddagger []}\ (\check{\Diamond})}}\ (\check{\circ})$$

(where the dashed lines are just simplifications of expressions) and we obtain that $\mathbf{L}(\langle_m \langle_n m \rangle^m_n \langle_n nm \rangle^n_n \rangle^m_m) \stackrel{\text{def}}{=} \{acb \mid a,b,c \in \mathcal{N}. b \neq a \text{ and } c \neq b\}$.

Interestingly, the application of rule ($\check{\Diamond}$) in the left branch of the above derivation corresponds to the deallocation of m (which yields the name a) and the contextual renaming of the content of n with the new name chosen for n (that is $\star_1$) due to the permutation dictated by the subexpression $\langle_n m \rangle^m_n$. Instead, the application of ($\check{\Diamond}$) in the right branch, simply cuts out the last names of both the left and right contexts because the subexpression $\langle_n nm \rangle^n_n$ does not involve any permutation. $\diamond$

Example 5 below shows that the language $\mathcal{L}_1$ in Example 1 is nominal regular.

Example 5. To show how the rules in Fig. 2 apply when the expressions contain a Kleene-star, we consider $\langle_m(\langle_n n\rangle^m_n)^*\rangle^m_m$ with the derivation tree corresponding to a three-fold unfolding of the Kleene-star.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[]\ddagger\langle_m(\langle_n n\rangle^m_n)^*\rangle^m_m\ddagger[]}(\hat{\Diamond}_=)}{[a]\ddagger(\langle_n n\rangle^a_n)^*\ddagger[a]}(\circledast)}{[a]\ddagger\langle_n n\rangle^a_n\langle_n n\rangle^a_n\langle_n n\rangle^a_n\ddagger[a]}(\hat{\diamond})}{\dfrac{[a]\ddagger\langle_n n\rangle^a_n\ddagger[a]}{[a,b]\ddagger b\ddagger[b,a]}(\hat{\Diamond}_\neq)} \quad \dfrac{\dfrac{[a]\ddagger\langle_n n\rangle^a_n\langle_n n\rangle^a_n\ddagger[a]}{\dfrac{[a]\ddagger\langle_n n\rangle^a_n\ddagger[a]}{[a,c]\ddagger c\ddagger[c,a]}(\hat{\Diamond}_\neq)} \quad \dfrac{[a]\ddagger\langle_n n\rangle^a_n\ddagger[a]}{[a,d]\ddagger d\ddagger[d,a]}(\hat{\Diamond}_\neq)}{}(\hat{\diamond})$$

$$\dfrac{\dfrac{\dfrac{\dfrac{[a,b]\ddagger b\ddagger[b,a]}{[a,b]\ddagger\{b\}\ddagger[b,a]}(b)}{[a]\ddagger\{\star_1\mid\star_1\neq a\}\ddagger[\star_1]}(\check{\Diamond}) \quad \dfrac{\dfrac{[a,c]\ddagger c\ddagger[c,a]}{[a,c]\ddagger\{c\}\ddagger[c,a]}(c)}{[a]\ddagger\{\star_2\mid\star_2\neq a\}\ddagger[\star_2]}(\check{\Diamond}) \quad \dfrac{\dfrac{[a,d]\ddagger d\ddagger[d,a]}{[a,d]\ddagger\{d\}\ddagger[d,a]}(d)}{[a]\ddagger\{\star_3\mid\star_3\neq a\}\ddagger[\star_3]}(\check{\Diamond})}{[a]\ddagger\{\star_2\star_3\mid\star_2\neq a,\star_3\neq\star_2\}\ddagger[\star_3]}(\check{\diamond})}{[a]\ddagger\{\star_1\star_2\star_3\mid\star_1\neq a,\star_2\neq\star_1,\star_3\neq\star_2\}\ddagger[\star_3]}(\check{\Diamond})}{[]\ddagger\{\star_1\star_2\star_3\mid\star_1\neq\star_0,\star_2\neq\star_1,\star_3\neq\star_2\}\ddagger[]}$$
$$[]\ddagger\{\star_1\star_2\star_3\mid\star_2\neq\star_1,\star_3\neq\star_2\}\ddagger[]$$

Generalising to a k-fold unfolding, we have the language $L^k=\{a_1\cdots a_k\mid a_1\neq a_0,\ldots,a_k\neq a_{k-1}\}$ which yields

$$\mathbf{L}(\langle_m(\langle_n n\rangle^m_n)^*\rangle^m_m)\stackrel{\text{def}}{=}\bigcup_{k\in\mathbb{N}}L^k=\{a_1\cdots a_k\mid\forall k\in\mathbb{N},\forall i\in\{1,\ldots,k-1\}.a_i\neq a_{i+1}\}\quad\diamond$$

We can also express the following languages taken from [11]. Note that the second is not quasi-regular in the sense of [11], that is, it cannot be accepted by FMAs.

Example 6. Define $\mathsf{N}\stackrel{\text{def}}{=}\langle_n n\rangle^n_n$. Note that $\mathbf{L}(\mathsf{N})=\mathcal{N}$. We have

$\mathbf{L}(\langle_n\mathsf{N}^*n\mathsf{N}^*n\rangle^n_n\mathsf{N}^*)=\{a_1\cdots a_k\mid\exists i,j.\,1\leq i<j\leq k\ \&\ a_i=a_j\}$,

$\mathbf{L}(\langle_n\mathsf{N}^*n\rangle^n_n)=\{a_1\cdots a_k\mid\forall i.\,1\leq i<k\ \Rightarrow\ a_i\neq a_k\}$. $\diamond$

The language $\mathcal{L}_{\alpha\rho}$ in Example 2 is nominal regular as discussed in the next example.

Example 7. Calculations similar to those in Examples 4 and 5 can be done to show that the p-NRE $\alpha\langle_m((m^*\langle_n(m+n)\,\rho\rangle^m_n m^*)+m^*)\,\rho\rangle^m_m$ (where $\alpha,\rho\in\mathcal{S}$) defines $\mathcal{L}_{\alpha\rho}$. $\diamond$

α-conversion on p-NREs. Permutations can be used to define α-conversion on p-NREs. Let S be a finite set of $\mathcal{N}$ and let π_S be a bijection on $\mathcal{N}$ which fixes each element in S. We say that $L\ddagger\pi_L\cdot\mathsf{ne}\ddagger R$ is an α-conversion of an expression-in-contexts $L\ddagger\mathsf{ne}\ddagger R$. For a p-NRE ne, ne' is an α-conversion of ne, if ne' is obtained by α-converting a subexpression of ne which appears in a derivation tree from $[]\ddagger\mathsf{ne}\ddagger[]$ using the rules Fig. 1 (taking care of unfolding Kleene-stars only once and of avoiding any renamings in name abstractions). For example, a p-NRE $\langle_m\langle_n m\rangle^m_n\langle_n nm\rangle^n_n\rangle^m_m$ is α-converted to $\langle_l\langle_m l\rangle^l_m\langle_n nl\rangle^n_n\rangle^l_l$.

$$(m\ n)\ \dfrac{\dfrac{\dfrac{[a]\ddagger\langle_n a\rangle^a_n\ddagger[a]}{[a]\ddagger\langle_m a\rangle^a_m\ddagger[a]} \quad [a]\ddagger\langle_n na\rangle^n_n\ddagger[a]}{[a]\ddagger\langle_m a\rangle^a_m\langle_n na\rangle^n_n\ddagger[a]}(\hat{\diamond})}{[]\ddagger\langle_a\langle_m a\rangle^a_m\langle_n na\rangle^n_n\rangle^a_a\ddagger[]}(\hat{\Diamond})}{[]\ddagger\langle_l\langle_m l\rangle^l_m\langle_n nl\rangle^n_n\rangle^l_l\ddagger[]}(l\ a)$$

Proposition 1. *α-conversion on p-NREs is an equivalence relation on p-NREs.*

Theorem 1. *For p-NREs* ne_1 *and* ne_2*, if they are* α*-equivalent, they define the same nominal regular languages, i.e.* $\mathbf{L}(\mathsf{ne}_1) = \mathbf{L}(\mathsf{ne}_2)$.

The above results are due to the fact that a completely fresh name is always available when applying the rules $(\hat{\Diamond}_{\neq})$ and $(\hat{\Diamond}_{=})$ in Fig. 1

4 Automata with fresh-name generations and permutations

To handle *binders (fresh names)* and *permutations*, we extend *automata on binders over* $\mathcal{S}$ *and* $\mathcal{N}_{fin}$ in [13]. Denote the set of natural numbers with $\mathbb{N}$ and define $\underline{i}$ as $\{1,\dots,i\}$ for each $i \in \mathbb{N}$.

The automata in Definition 1 below have a set (of states) Q and a map $\|\_\| : Q \to \mathbb{N}$ which yield the *local registers of* $q \in Q$ as $\underline{\|q\|}$. Also, given $q \in Q$,

$$\mathfrak{L}(q) \stackrel{\text{def}}{=} \mathcal{S} \cup \underline{\|q\|} \cup \{\star\} \cup \{\circlearrowleft_i \mid i \in \underline{\|q\|}\},$$

is the set of possible labels of q.

Definition 1. *An* automaton with fresh-name generation and permutations over $\mathcal{S}$*, an* fp-automaton *for short , is a tuple* $\mathcal{H} = \langle Q, q_0, F, tr\rangle$ *such that*

- Q *is a finite set (of states) equipped with a map* $\|\_\| : Q \to \mathbb{N}$
- $q_0 \in Q$ *is* the initial state *and* $\|q_0\| = 0$
- $F \subseteq Q$ *is the set of* final states *and* $\|q\| = 0$ *for each* $q \in F$
- *for each* $q \in Q$ *and* $\alpha \in \mathfrak{L}(q) \cup \{\varepsilon\}$*, the set* $tr(q,\alpha) \subseteq Q$ *contains the* successor states *of* q*; for all* $q' \in tr(q,\alpha)$*, the following conditions must hold:*

$$\begin{aligned} \alpha = \star &\implies \|q'\| = \|q\| + 1 \\ \alpha = \circlearrowleft_i \text{ for } i \in \underline{\|q\|} &\implies \|q'\| = \|q\| - 1 \\ \alpha \in \mathcal{S} \cup \underline{\|q\|} \text{ or } \alpha = \varepsilon &\implies \|q'\| = \|q\| \end{aligned}$$

An fp-automaton is deterministic*, if for each* $q \in Q$ *and each label* $\alpha \in \mathfrak{L}(q)$

$$\begin{cases} |tr(q,\varepsilon)| = 0, \\ |tr(q,\alpha)| = 1, \quad \textit{otherwise}. \end{cases}$$

Finally, the i-th layer of $\mathcal{H}$ *is the subset* $Q^i \stackrel{\text{def}}{=} \{q \in Q \mid \|q\| = i\}$ *of* Q.

In an fp-automaton, the i-th layer is connected only by $\star$ to the $(i+1)$-th layer, and only by $\{\circlearrowleft_j \mid j \in \underline{i}\}$ to the $(i-1)$-th layer. Note that the i-th layer forms an automaton over $\mathcal{S} \cup \underline{i} \cup \{\varepsilon\}$ in the classical sense. Note that each state on the 0-th layer cannot have any $\circlearrowleft_{\_}$ transition, by definition; similarly, states in the highest layer cannot have $\star$-transitions. For a technical reason, we assume every fp-automaton is accessible in the usual sense.

Hereafter we fix an fp-automaton as $\mathcal{H} = \langle Q, q_0, F, tr\rangle$. A *configuration* of $\mathcal{H}$ is a triple $\langle q, w, \sigma\rangle$ consisting of a state q, a map $\sigma\colon \underline{\|q\|} \to \mathcal{N}$ assigning names to the local registers in q and a word w. The following definition is almost the same as the one in [13]. The only exceptions are $\star$-transitions and $\circlearrowleft_i$-transitions.

Definition 2. *Given $q,q' \in Q$ and two configurations $t = \langle q,w,\sigma\rangle$ and $t' = \langle q',w',\sigma'\rangle$, an fp-automaton $\mathcal{H}$* moves from *t to t' (written $t \xrightarrow{\mathcal{H}} t'$) if there is $\alpha \in \mathfrak{L}(q) \cup \{\varepsilon\}$ such that $q' \in tr(q,\alpha)$ and*

$$\begin{cases} \alpha \in \underline{\|q\|}, & w = \sigma(\alpha)\, w' \text{ and } \sigma' = \sigma \\ \alpha \in \mathcal{S}, & w = \alpha\, w' \text{ and } \sigma' = \sigma \\ \alpha = \varepsilon, & w = w' \text{ and } \sigma' = \sigma \\ \alpha = \star, & w = w', n \in \mathcal{N} \setminus Im(\sigma) \text{ and } \sigma' = \sigma_{[\|q'\| \mapsto n]} \\ \alpha = \circlearrowleft_i, & w = w' \text{ and } \sigma' = (\sigma \cdot (\|q\|\ i))_{\underline{\|q'\|}} \end{cases}$$

where $(\sigma \cdot (\|q\|\ i))_{\underline{\|q'\|}}$ is the restriction on $\underline{\|q'\|}$ of the function $\sigma \cdot (\|q\|\ i)$, i.e. σ permuted by $(\|q\|\ i)$. A configuration $\langle q,w,\sigma\rangle$ is initial *if $q = q_0$, w is a word and $\sigma = \bot$, and it is* accepting *if $q \in F$, $w = \varepsilon$ and $\sigma = \bot$.*

The set $reach_{\mathcal{H}}(t)$ of states reached by $\mathcal{H}$ from the configuration t is defined as

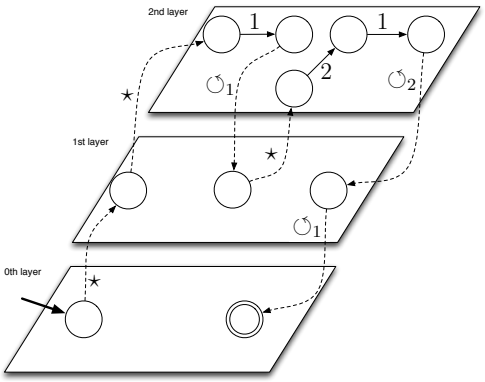
$$reach_{\mathcal{H}}(t) \stackrel{\text{def}}{=} \begin{cases} \{q\} & \text{if } t = \langle q,\varepsilon,\sigma\rangle \\ \bigcup_{t \xrightarrow{\mathcal{H}} t'} reach_{\mathcal{H}}(t') & \text{otherwise} \end{cases}$$

A run of $\mathcal{H}$ on a word *w is a sequence of moves of $\mathcal{H}$ from $\langle q_0,w,\bot\rangle$.*

Intuitively, $\star$ means "generate a *fresh* name" and store it in the highest register. Transitions labelled by $\circlearrowleft_i$ are meant to permute the value in highest register with the one in the i-th register and dispose the highest register. The $\star$ and $\circlearrowleft_i$ transitions are performed independently of the word w and introduce some non-determinism even to deterministic fp-automata.

Definition 3. *The fp-automaton $\mathcal{H}$* accepts*, or* recognises*, a word w on $\mathcal{S} \cup \mathcal{N}$ when $F \cap reach_{\mathcal{H}}(\langle q_0,w,\bot\rangle) \neq \emptyset$. The language of $\mathcal{H}$ is the set $\mathcal{L}_{\mathcal{H}}$ of words accepted by $\mathcal{H}$.*

Example 8. The fp-automaton below



accepts the language $\mathbf{L}(\langle_m\langle_n m\rangle_n^m\langle_n nm\rangle_n^n\rangle_m^m)$, see Examples 3 and 4. ◇

5 A Kleene Theorem

We show a Kleene theorem for nominal regular languages: Every nominal regular language is recognised by an fp-automaton (Theorem 2) and, vice versa, every language accepted by an fp-automaton is nominal regular (Theorems 3 and 4).

The interpretation of p-NREs via the rules of Fig. 1 has to be extended to expressions-in-contexts and to *languages-in-contexts*. For example, for $[a] \ddagger \langle_n n\rangle^a_n \langle_n n\rangle^a_n \ddagger [a]$, the language-in-contexts is $[a] \ddagger \{cd \mid \forall c \neq a, \forall d \neq c\} \ddagger [a]$.

5.1 From p-NREs to fp-automata

Given a p-NRE ne, we shall inductively construct an fp-automaton:

Theorem 2. *Given a p-NRE* ne*, there exists an fp-automaton $\mathcal{H}$ which accepts the nominal language* $\mathbf{L}(\mathsf{ne})$*, i.e.* $\mathbf{L}(\mathsf{ne}) = \mathcal{L}_{\mathcal{H}}$.

As seen in § 3, our expressions are *context-dependent* and the contexts are *dynamic*. Similarly, we construct automata-in-contexts $L \ddagger \mathcal{H} \ddagger R$, that is generalised fp-automata where initial and final states may have $lth(L) = lth(M)$ registers equipped with a function η mapping the h-th register of the initial state to the h-th name of L. Abusing notation, we let $\mathcal{H}$ to range over automata-in-contexts.

Base cases. Let $L \ddagger \mathsf{ne} \ddagger R$ be an expression-in-contexts. By Fact 1, we can assume that L and R have the same elements (hence $lth(L) = lth(R)$). Since L and R are in general non-empty, we equip fp-automata with a function η that maps the local registers of the initial state to names (in L).

When $\mathsf{ne} = \mathit{1}$ or $\mathsf{ne} = \mathit{0}$, we define

$\mathcal{H}_{(\!|1|\!)} = \langle Q, q_0, tr, F, \eta\rangle$ as follows

- $Q \stackrel{\text{def}}{=} \{q_0\}$ with $\|q_0\| = lth(L)$;
- $tr(q_0, \alpha) \stackrel{\text{def}}{=} \emptyset$ for each $\alpha \in \mathfrak{L}(q_0)$;
- $F \stackrel{\text{def}}{=} \{q_0\}$;
- $\eta(k) \stackrel{\text{def}}{=} l_k$, for each $k \in \{1, \ldots, lth(L)\}$.

$\mathcal{H}_{(\!|0|\!)} = \langle Q, q_0, tr, F, \eta\rangle$ as follows:

- $Q \stackrel{\text{def}}{=} \{q_0\}$ with $\|q_0\| = lth(L)$;
- $tr(q_0, \alpha) \stackrel{\text{def}}{=} \emptyset$ for each $\alpha \in \mathfrak{L}(q_0)$;
- $F \stackrel{\text{def}}{=} \emptyset$;
- $\eta(k) \stackrel{\text{def}}{=} l_k$ for each $k \in \{1, \ldots, lth(L)\}$.

When $\mathsf{ne} = n$, we let $\mathcal{H}_{(\!|n|\!)} = \langle Q, q_0, tr, F, \eta\rangle$ as follows:

- $Q \stackrel{\text{def}}{=} \{q_0, q_1\}$ with $\|q_0\| = lth(L)$ and $\|q_1\| = lth(L)$;
- $tr(q_0, \alpha) \stackrel{\text{def}}{=} \begin{cases} \{q_1\} & \alpha = k \text{ and } l_k = n \\ \emptyset & \text{otherwise} \end{cases}$ and $tr(q_1, \alpha) \stackrel{\text{def}}{=} \emptyset$ for each $\alpha \in \mathfrak{L}(q_1)$;
- $F \stackrel{\text{def}}{=} \{q_1\}$; $\eta(k) \stackrel{\text{def}}{=} l_k$ for each $k \in \{1, \ldots, lth(L)\}$.

When $\mathsf{ne} = s$, we let $\mathcal{H}_{(\!|s|\!)} = \langle Q, q_0, tr, F, \eta\rangle$ as follows:

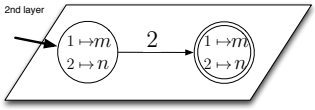
- $Q \stackrel{\text{def}}{=} \{q_0, q_1\}$ with $\|q_0\| = lth(L)$ and $\|q_1\| = lth(L)$;
- $tr(q_0, \alpha) \stackrel{\text{def}}{=} \begin{cases} \{q_1\} & \alpha = s \\ \emptyset & \text{otherwise} \end{cases}$ and $tr(q_1, \alpha) \stackrel{\text{def}}{=} \emptyset$ for each $\alpha \in \mathfrak{L}(q_1)$;

– $F \stackrel{\text{def}}{=} \{q_1\}$; $\eta(k) \stackrel{\text{def}}{=} l_k$ for each $k \in \{1,\ldots,lth(L)\}$.

Proposition 2. *The automata-in-contexts $L \ddagger \mathcal{H}_{(\!|l|\!)} \ddagger R$, $L \ddagger \mathcal{H}_{(\!|0|\!)} \ddagger R$, $L \ddagger \mathcal{H}_{(\!|n|\!)} \ddagger R$ and $L \ddagger \mathcal{H}_{(\!|s|\!)} \ddagger R$ accept the languages-in-contexts $L \ddagger \{\varepsilon\} \ddagger R$, $L \ddagger \emptyset \ddagger R$, $L \ddagger \{n\} \ddagger R$ and $L \ddagger \{s\} \ddagger R$, respectively. Furthermore, for every final state q in $L \ddagger \mathcal{H}_{(\!|l|\!)} \ddagger R$, $L \ddagger \mathcal{H}_{(\!|0|\!)} \ddagger R$, $L \ddagger \mathcal{H}_{(\!|n|\!)} \ddagger R$ and $L \ddagger \mathcal{H}_{(\!|s|\!)} \ddagger R$, we have $\|q\| = lth(R)$.*

For automata-in-contexts we consider configurations and runs as in Definition 2, with the exception that the η in the initial and final configurations $\langle q, w, \eta\rangle$ takes into account the names in the pre- and post-contexts.

Example 9. The automaton-in-contexts $[m,n] \ddagger \mathcal{H}_{(\!|n|\!)} \ddagger [n,m]$ below



is constructed from $[m,n] \ddagger n \ddagger [n,m]$ ◇

Union. Let $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1|\!)} \ddagger R$ and $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_2|\!)} \ddagger R$ be automata-in-contexts, where $\mathcal{H}_{(\!|\mathsf{ne}_1|\!)} = \langle Q_1, q_{1,0}, tr_1, F_1, \eta_1\rangle$ and $\mathcal{H}_{(\!|\mathsf{ne}_2|\!)} = \langle Q_2, q_{2,0}, tr_2, F_2, \eta_2\rangle$ for the corresponding expressions-in-contexts $L \ddagger \mathsf{ne}_1 \ddagger R$ and $L \ddagger \mathsf{ne}_2 \ddagger R$. Therefore, η_1 and η_2 are identical. Then, we let $\mathcal{H}_{(\!|\mathsf{ne}_1+\mathsf{ne}_2|\!)} = \langle Q^+, q_0^+, tr^+, F^+, \eta^+\rangle$ as follows:

– $Q^+ \stackrel{\text{def}}{=} \{q_0^+\} \uplus Q_1 \uplus Q_2$ with $\|q_0^+\| = lth(L)$;
– $tr^+(q_0^+, \alpha) \stackrel{\text{def}}{=} \begin{cases} \{q_{1,0}, q_{2,0}\} & \alpha = \varepsilon \\ \emptyset & \text{otherwise} \end{cases}$ and the others are the same as before;
– $F^+ \stackrel{\text{def}}{=} F_1 \uplus F_2$; $\eta \stackrel{\text{def}}{=} \eta_1 \ (= \eta_2)$.

Proposition 3. *The automaton-in-contexts $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1+\mathsf{ne}_2|\!)} \ddagger R$ accepts the language-in-contexts $L \ddagger \mathbf{L}(\mathsf{ne}_1 + \mathsf{ne}_2) \ddagger R$. Furthermore, for each final state q in $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1+\mathsf{ne}_2|\!)} \ddagger R$, we have $\|q\| = lth(R)$.*

Concatenation. By the context calculus of Fig. 1, the post-context of the first expression must be the same as the pre-context of the second expression. Let $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1|\!)} \ddagger L$ and $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_2|\!)} \ddagger R$ be automata-in-contexts with $\mathcal{H}_{(\!|\mathsf{ne}_1|\!)} = \langle Q_1, q_{1,0}, tr_1, F_1, \eta_1\rangle$ and $\mathcal{H}_{(\!|\mathsf{ne}_2|\!)} = \langle Q_2, q_{2,0}, tr_2, F_2, \eta_2\rangle$, and $L \ddagger \mathsf{ne}_1 \ddagger L$ and $L \ddagger \mathsf{ne}_2 \ddagger R$ the corresponding expressions-in-contexts. By the definition of the context calculus, the post-context of the first expression must be the same as the pre-context of the second expression. We let $\mathcal{H}_{(\!|\mathsf{ne}_1 \circ \mathsf{ne}_2|\!)} = \langle Q^\circ, q_0^\circ, tr^\circ, F^\circ, \eta^\circ\rangle$ as follows:

– $Q^\circ \stackrel{\text{def}}{=} Q_1 \uplus Q_2$; $q_0^\circ \stackrel{\text{def}}{=} q_{1,0}$; $F^\circ \stackrel{\text{def}}{=} F_2$; $\eta^\circ \stackrel{\text{def}}{=} \eta_1$;
– $tr^\circ(q,\alpha) \stackrel{\text{def}}{=} \begin{cases} tr_1(q,\alpha) \cup \{q_{2,0}\} & q \in F_1 \text{ and } \alpha = \varepsilon \\ tr_1(q,\alpha) & q \in Q_1 \text{ and either } q \notin F_1 \text{ or } \alpha \neq \varepsilon \\ tr_2(q,\alpha) & q \in Q_2 \end{cases}$.

Proposition 4. *The automaton-in-contexts $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1 \circ \mathsf{ne}_2|\!)} \ddagger R$ accepts the language-in-contexts $L \ddagger \mathbf{L}(\mathsf{ne}_1 \circ \mathsf{ne}_2) \ddagger R$. Furthermore, for each final state q in $L \ddagger \mathcal{H}_{(\!|\mathsf{ne}_1 \circ \mathsf{ne}_2|\!)} \ddagger R$, we have $\|q\| = lth(R)$.*

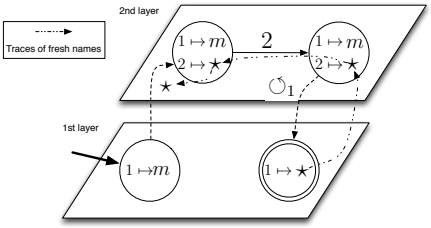
Name-abstraction. Let $(L@n) \ddagger \mathcal{H}_{\langle\!|\mathsf{ne}|\!\rangle} \ddagger (R@m)$ be an automaton-in-contexts, where $\mathcal{H}_{\langle\!|\mathsf{ne}|\!\rangle} = \langle Q, q_0, tr, F, \eta \rangle$, and $(L@n) \ddagger \mathsf{ne} \ddagger (R@m)$ the expression-in-contexts. We let $\mathcal{H}_{\langle\!|\langle_n \mathsf{ne} \rangle_n^m|\!\rangle} = \langle Q^\Diamond, q_0^\Diamond, tr^\Diamond, F^\Diamond, \eta^\Diamond \rangle$ as follows:

- $Q^\Diamond \stackrel{\text{def}}{=} Q \uplus \{q_s, q_t\}$ with $\|q_s\| = lth(L)$ and $\|q_t\| = lth(L)$;
- $q_0^\Diamond \stackrel{\text{def}}{=} q_s$; $F^\Diamond \stackrel{\text{def}}{=} \{q_t\}$; $\eta^\Diamond \stackrel{\text{def}}{=} \eta$;
- $tr^\Diamond(q,\alpha) \stackrel{\text{def}}{=} \begin{cases} \{q_0\} & q = q_s \text{ and } \alpha = \star \\ \emptyset & q = q_s \text{ and } \alpha \neq \star \\ \emptyset & q = q_t \\ \{q_t\} & q \in F \text{ and } \alpha = \circlearrowleft_k \text{ for } k \text{ with } r_k = n \\ \emptyset & q \in F \text{ and } \alpha = \circlearrowleft_k \text{ for } k \text{ with } r_k \neq n \\ tr(q,\alpha) & \text{otherwise} \end{cases}$;

where $(R@m) = [r_1, \ldots, r_{lth(R)+1}]$ (so $r_{lth(R)+1} = m$).

Proposition 5. *The automaton-in-contexts $L \ddagger \mathcal{H}_{\langle\!|\langle_n \mathsf{ne} \rangle_n^m|\!\rangle} \ddagger R$ recognises the language-in-contexts $L \ddagger \mathbf{L}(\langle_n \mathsf{ne} \rangle_n^m) \ddagger R$. Furthermore, for the final state q_t in $L \ddagger \mathcal{H}_{\langle\!|\langle_n \mathsf{ne} \rangle_n^m|\!\rangle} \ddagger R$, we have $\|q_t\| = lth(R)$.*

Example 10. For $[m,n] \ddagger \mathcal{H}_{\langle\!|n|\!\rangle} \ddagger [n,m]$, the fp-automaton below



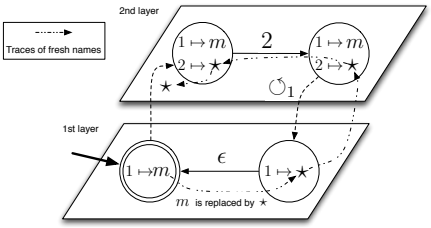
is constructed according to the name-abstraction in contexts $[m] \ddagger \mathcal{H}_{\langle\!|\langle_n n \rangle_n^m|\!\rangle} \ddagger [m]$. ◇

Kleene star. For an automaton-in-contexts $L \ddagger \mathcal{H}_{\langle\!|\mathsf{ne}|\!\rangle} \ddagger R$ with $\mathcal{H}_{\langle\!|\mathsf{ne}|\!\rangle} = \langle Q, q_0, tr, F, \eta \rangle$ and the expression-in-contexts $L \ddagger \mathsf{ne} \ddagger R$, let $\mathcal{H}_{\langle\!|\mathsf{ne}^*|\!\rangle} = \langle Q^*, q_0^*, tr^*, F^*, \eta^* \rangle$ as follows:

- $Q^* \stackrel{\text{def}}{=} Q$; $q_0^* \stackrel{\text{def}}{=} q_0$; $F^* \stackrel{\text{def}}{=} \{q_0^*\}$; $\eta^* \stackrel{\text{def}}{=} \eta$;
- $tr^*(q,\alpha) \stackrel{\text{def}}{=} \begin{cases} tr(q,\alpha) \cup \{q_0^*\} & q \in F \text{ and } \alpha = \varepsilon \\ tr(q,\alpha) & \text{otherwise} \end{cases}$.

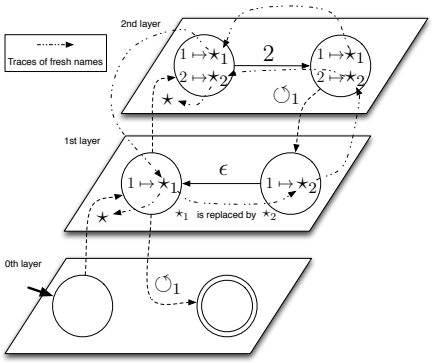
Proposition 6. *The automaton-in-contexts $L \ddagger \mathcal{H}_{\langle\!|\mathsf{ne}^*|\!\rangle} \ddagger R$ recognises the language-in-contexts $L \ddagger \mathbf{L}(\mathsf{ne}^*) \ddagger R$. Furthermore, for the final state q_0 in $L \ddagger \mathcal{H}_{\langle\!|\mathsf{ne}^*|\!\rangle} \ddagger R$, we have $\|q_0\| = lth(R)$.*

Example 11. For $[m] \ddagger \mathcal{H}_{\langle\!|\langle_n n \rangle_n^m|\!\rangle} \ddagger [m]$, the fp-automaton



is the Kleene star construction for $[m] \ddagger \mathcal{H}_{(\!|(\langle_n n\rangle_n^m)^*|\!)} \ddagger [m]$. ◇

From the fp-automaton in Example 11 we build an fp-automaton that accepts the language $\mathcal{L}_1$ in Example 1 by name-abstraction of $[m] \ddagger \mathcal{H}_{(\!|(\langle_n n\rangle_n^m)^*|\!)} \ddagger [m]$. This yields the following fp-automaton



5.2 From fp-automata to p-NREs

Deterministic and non-deterministic fp-automata are equivalent.

Theorem 3. *Given an fp-automaton $\mathcal{H}$, there is a deterministic fp-automaton which accepts the same language as $\mathcal{L}_{\mathcal{H}}$.*

Proof (Sketch). The main proof technique is a "layer-wise" powerset construction. Since the i-th layer is basically a classical automaton over $\mathcal{S} \cup \underline{i} \cup \{\varepsilon\}$, the powerset construction allows us to make each layer deterministic. The only thing we have to care about is how to connect these deterministic layers by $\star$ and $\{\circlearrowleft_{i'} \mid i' \in \underline{i}\}$ in a deterministic way. This is shown below.

For each subset $\{q_1^i, \ldots, q_k^i\}$ of the i-th layer Q^i, we let

$$tr(\{q_1^i, \ldots, q_k^i\}, \star) \stackrel{\text{def}}{=} \{q^{i+1} \in Q^{i+1} \mid \exists j \in \underline{k}.\ q^{i+1} \in tr(q_j^i, \star)\}$$
$$tr(\{q_1^i, \ldots, q_k^i\}, \circlearrowleft_{i'}) \stackrel{\text{def}}{=} \{q^{i-1} \in Q^{i-1} \mid \exists j \in \underline{k}.\ q^{i-1} \in tr(q_j^i, \circlearrowleft_{i'})\}$$

for each $i' \in \underline{i}$. Hence we obtain a deterministic automaton of $\mathcal{H}$. □

Note that the powerset construction in the proof above has to be performed layer-wise due to the presence of local registers.

Theorem 4. *Any language accepted by a deterministic fp-automaton $\mathcal{H}$ is a nominal regular language. That is, there exists a p-NRE* ne *such that $\mathcal{L}_{\mathcal{H}} = \mathbf{L}(\mathsf{ne})$.*

Proof (Sketch). The states Q of a deterministic fp-automaton $\mathcal{H}$ can be decomposed into $h = \max_{q \in Q} \|q\|$ layers (where h is the highest layer of $\mathcal{H}$):

$$Q^0 = \{q_1^0, \ldots, q_{m_0}^0\}, \qquad Q^1 = \{q_1^1, \ldots, q_{m_1}^1\}, \qquad \cdots \qquad Q^h = \{q_1^h, \ldots, q_{m_h}^h\} \tag{3}$$

Note that $q_0 \in Q^0$ (we assume $q_1^0 = q_0$) and $F \subseteq Q^0$. We fix an arbitrary order on states given by their index in (3), let ${}^sR^k_{i,j}$ denote the set of paths from q_i^s to q_j^s which visit only states on layers higher than s or states $q_r^s \in Q^s$ with $r \leq k$, and let $E_{i,j} \stackrel{\text{def}}{=} \emptyset$ if $i \neq j$ and $E_{i,i} \stackrel{\text{def}}{=} \{\varepsilon\}$. Then, ${}^sR^k_{i,j}$ is defined by

$$ {}^hR^0_{i,j} \stackrel{\text{def}}{=} \{\alpha \mid q_j^h \in tr(q_i^h, \alpha)\} \cup E_{i,j} \qquad {}^hR^k_{i,j} \stackrel{\text{def}}{=} {}^hR^{k-1}_{i,k} \left({}^hR^{k-1}_{k,k}\right)^* {}^hR^{k-1}_{k,j} \cup {}^hR^{k-1}_{i,j} $$

on the highest layer h. On the other layers ($s < h$), it is defined by

$$ {}^sR^0_{i,j} \stackrel{\text{def}}{=} \{\alpha \mid q_j^s \in tr(q_i^s, \alpha)\} \cup \bigcup_{s' \in \underline{s+1}} \left\langle_{s+1} \bigcup_{(i',j') \in \Gamma^{s,s'}_{i,j}} {}^{s+1}R^{m_{s+1}}_{i',j'} \right\rangle^{s'}_{s+1} \cup E_{i,j} $$

$$ {}^sR^k_{i,j} \stackrel{\text{def}}{=} {}^sR^{k-1}_{i,k} \left({}^sR^{k-1}_{k,k}\right)^* {}^sR^{k-1}_{k,j} \cup {}^sR^{k-1}_{i,j} \cup \bigcup_{s' \in \underline{s+1}} \left\langle_{s+1} \bigcup_{(i',j') \in \Gamma^{s}_{i,j}} {}^{s+1}R^{m_{s+1}}_{i',j'} \right\rangle^{s'}_{s+1} $$

where $\Gamma^{s,s'}_{i,j} \stackrel{\text{def}}{=} \{(i',j') \mid q^{s+1}_{i'} \in tr(q^s_i, \star) \,\&\, q^s_j \in tr(q^{s+1}_{j'}, \circlearrowleft_{s'})\}$ for each $s' \in \underline{s+1}$. Hence, $\bigcup_{s' \in \underline{s+1}} \left\langle_{s+1} \bigcup_{(i',j') \in \Gamma^{s,s'}_{i,j}} {}^{s+1}R^{m_s}_{i',j'} \right\rangle^{s'}_{s+1}$ is the collection of all paths from q_i^s to q_j^s visiting only states on the higher layers. Finally, we translate all paths from the initial state to final states into a nominal regular expression, but this is analogous to the classical theory. The only distinction is how to choose fresh names for binders. However, this is done by reserving names for fresh names as a distinct subset $\{n_1, \ldots, n_h\}$ of $\mathcal{N}$, with the $\left\langle_{s+1}\right.$ and $\left.\right\rangle^{s'}_{s+1}$ indicating how to generate expressions for the binding construct. □

Therefore, by the above theorems, we conclude that every fp-automaton $\mathcal{H}$ has a p-NRE ne such that $\mathcal{L}_{\mathcal{H}} = \mathbf{L}(\mathsf{ne})$.

6 Conclusion

We have extended the nominal regular expressions and automata presented in [13] with permutations in order to provide a notion of regular expression for languages on infinite alphabets (without binders). Our main technical contribution is a Kleene theorem that establishes an equivalence between nominal regular expressions with permutations and fp-automata.

A novelty of our approach is how to handle the environments and how permutations change the local views of the environment. This is done with the help of the context calculus in Fig. 1, which represents the views on the environments by "contexts" similar to Hoare triples. The language construction of Fig. 2 then explains how this information flow generates nominal regular languages.

Yet another delicate aspect of our theory is the subtle non-deterministic behaviour present even in deterministic automata. As highlighted by the first language of Example 6, Definition 2 does not require the automaton to consume a letter if moving on

an allocation or deallocation transition. These moves are non-deterministic in the sense that they are not controlled by the word to be recognised. This is crucial to the equivalence established in Theorem 4. Indeed, non-deterministic models are more expressive than deterministic ones when considering languages on infinite alphabets [11, 16].

The natural next step to take in our research is to exploit the results presented here to compare the expressiveness of nominal regular expressions with other models featuring languages on infinite alphabets. We note that our nominal regular languages are closed under union, intersection, concatenation and Kleene-star, but not under complement. Whereas the regular languages of [13] are closed under resource-sensitive complement, this is no longer the case here, since allocation and deallocation transitions are no longer controlled by explicit binders in the words. This situation is similar to the FMA of [11] although FMA do not accept the second language of Example 6. A precise comparison with FMA and related models such as those of [16] or [22] is left for future work.

Further investigations should reveal the categorical and (co)algebraic nature of our automata. In particular, the fact that the automata work level-wise suggests a many-sorted approach via presheaves (see also the two-sorted coalgebras of [7]). It would also be interesting to combine the work of this paper with [13] along the lines suggested by [2], which investigates how the implicit scope of names in words without binders interacts with binders having explicit scope. In another direction, we plan to extend our approach towards Kleene algebras (with tests) [12] and possible applications to verification. Other interpretations of the binders in the style of the research programme devised in [4] will also be of interest.

Acknowledgements We would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

1. R. Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3), 2009.
2. M. Bartoletti, P. Degano, G. L. Ferrari, and R. Zunino. Hard life with weak binders. *Electr. Notes Theor. Comput. Sci.*, 242(1):49–72, 2009.
3. M. Bojanczyk. Data monoids. In *STACS*, pages 105–116, 2011.
4. M. Bojanczyk, B. Klin, and S. Lasota. Automata with group actions. In *IEEE Symposium on Logic in Computer Science*, pages 355–364, 2011.
5. P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, May 2003.
6. V. Ciancia and E. Tuosto. A novel class of automata for languages on infinite alphabets. Technical Report CS-09-003, Leicester, 2009.
7. V. Ciancia and Y. Venema. Stream automata are coalgebras. In *11th International Workshop on Coalgebraic Methods in Computer Science (CMCS 2012)*, 2012.
8. M. Gabbay and V. Ciancia. Freshness and name-restriction in sets of traces with names. In *FoSSaCS*, volume 6604 of *LNCS*, pages 365–380. Springer, 2011.
9. M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In *Symbolic on Logics in Comput Science*, pages 214–224, 1999.
10. M. Kaminski and T. Tan. Regular expressions for languages over infinite alphabets. *Fundam. Inform.*, 69(3):301–318, 2006.
11. N. Kaminski, Michael Francez. Finite-memory automata. *TCS*, 134(2):329–363, 94.

12. D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000.
13. A. Kurz, T. Suzuki, and E. Tuosto. On nominal regular languages with binders. In *FoSSaCS*, volume 7213 of *LNCS*, pages 255–269, 2012.
14. U. Montanari and M. Pistore. π-Calculus, Structured Coalgebras, and Minimal HD-Automata. In M. Nielsen and B. Roman, editors, *MFCS*, volume 1983 of *LNCS*. Springer, 2000.
15. R. Myers. *Rational Coalgebraic Machines in Varieties: Languages, Completeness and Automatic Proofs.* PhD thesis, Imperial College London, 2011.
16. F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.
17. M. Pistore. *History Dependent Automata*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1999.
18. N. Pouillard and F. Pottier. A fresh look at programming with names and binders. In *Proceeding of the 15th ACM SIGPLAN international conference on Functional programming*, pages 217–228, 2010.
19. L. Segoufin. Automata and Logics for Words and Trees over an Infinite Alphabet. In *Computer Science Logic*, volume 4207 of *LNCS*, pages 41–57. Springer, 2006.
20. M. Shinwell, A. Pitts, and M. Gabbay. Freshml: programming with binders made simple. In *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming*, pages 263–274, 2003.
21. C. Stirling. Dependency tree automata. In *FoSSaCS'09*, pages 92–106, 2009.
22. N. Tzevelekos. Fresh-Register Automata. In *Symposium on Principles of Programming Languages*, pages 295–306. ACM, 2011.