?

Abstract

Process calculi for service-oriented computing often feature generation of fresh resources. So-called nominal automata have been studied both as semantic models for such calculi, and as acceptors of languages of finite words over infinite alphabets. In this paper we investigate nominal automata that accept infinite words. These automata are a generalisation of deterministic Muller automata to the setting of nominal sets. We prove decidability of complement, union, intersection, emptiness and equivalence, and determinacy by ultimately periodic words. The key to obtain such results is to use finite representations of the (otherwise infinite-state) defined class of automata. The definition of such operations enables model checking of process calculi featuring infinite behaviours, and resource allocation, to be implemented using classical automata-theoretic methods.

1. Introduction

This paper aims at contributing to the theory of formal verification of *global computing* systems, by extending the theory of Muller automata to the case of infinite alphabets, while retaining decidability. In this way, it is possible to adapt the classical automata-theoretic approach to formal specification and verification [**?**] to systems with resource generation capabilities, where the number of possible resources is infinite, provided that these systems enjoy a *finite memory* property.

Transition structures, in the form of automata, are used to represent logic formalisms interpreted over finite and infinite words, dating back to [**? ?**]. The possibility of translating modal logic formulas to automata led to the development of *model checking*. Systems that feature resource allocation (e.g. [**?**]), typically in the form of *name allocation*, pose specific challenges. For instance, they have ad-hoc notions of bisimulation, which cannot be captured by standard set-theoretic models. Transition structures that correctly model name allocation have been proposed in various forms, including coalgebras over presheaf categories [**? ? ? ? ?**], history-dependent automata [**?**], and automata over nominal sets [**?**]. Equivalence of these models has been established both at the level of base categories [**? ? ?**] and of coalgebras [**?**]. More recently, the field of *nominal automata* has essentially used the same structures, no longer as semantic models, but rather as acceptors of languages of finite words (see e.g., [**? ? ? ?**]). In particular, the obtained languages are based on infinite alphabets, but still

enjoy finite memory (in fact, the well known *register automata* of Francez and Kaminski can be regarded as nominal automata, see [**?**]).

The case of infinite words over nominal alphabets is more problematic, as an infinite word over an infinite alphabet is generally not *finitely supported*.[1] Consider a machine that reads any symbol from an infinite, countable alphabet, and never stores it. Clearly, such a machine has finite (empty) memory. The set of its traces is simply described as the set of all infinite words over the alphabet. However, in the language we have various species of words. Some of them are finitely supported, e.g. words that consist of the infinite repetition of a finite word. Some others are not finitely supported, such as the word enumerating all the symbols of the alphabet. Such words lay inherently out of the realm of nominal sets. However, the existence of these words does not give infinite memory to the language. More precisely, words without finite support can not be "singled out" by a finite memory machine; if a machine accepts one of them, then it will accept infinitely many others, including finitely supported words.

This work aims at translating the above intuitions into precise mathematical terms, in order to define a class of languages made of infinite words over infinite alphabets, enjoying finite-memory properties. We extend automata over nominal sets to handle infinite words, by imposing a (Muller-style) acceptance condition over the *orbits* (not the states!) of automata. By doing so, it turns out that our languages not only are finite-memory, but they retain computational properties, such as closure under boolean operations and decidability of emptiness (thus, containment and equivalence), which we prove by providing finite representations, and effective constructions. As in the case of standard ω-automata, the shift to infinite words requires these results to be proved from scratch, as it is not possible to merely extend proofs from the finite words case. These results enable automata-theoretic model checking to be performed on systems with infinite resources, using traditional model-checking algorithms.

Furthermore, we prove that the defined languages are determined by their ultimately-periodic fragments. This theorem is fundamental for *learning* logical properties (see e.g., [**?**], or [**?**]) and has been used to provide a complete minimisation procedure for equivalent representations of Muller automata [**?**]. Establishing this theoretical result in the nominal case is an important step towards the application of such techniques to global computing scenarios.

2. Example: peer-to-peer system

In order to introduce the presented topic, in this section we discuss an application of nominal automata to distributed systems. Consider an idealized *peer-to-peer* system where each peer receives queries from an arbitrary, unbounded number of other peers, represented by an infinite set of unique identifiers. Each peer buffers requests in a finite queue. Then one query is selected, among the

[1] The notion of finite support, coming from the theory of nominal sets, will be clarified later; roughly, finitely supported elements just use a finite set of names.

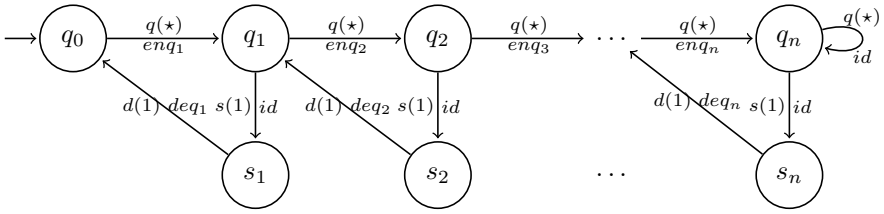

Figure 1: Automaton for the FCFS policy.

buffered ones, and is served by establishing a temporary connection with the target. Peers are not normally supposed to terminate, thus their relevant properties ought to predicate on infinite words. On the other hand, actions executed by peers carry information about other peers, which are drawn from an infinite set, therefore the symbols constituting words are infinite. Finally, each peer has finite memory. This is the key to maintain decidability, and is mathematically modeled by the notion of *finite support* in nominal automata. Once established that our languages are made of infinite words over an infinite alphabet, but retain a finite memory property, we can use automata to characterize properties of local peers in a global environment. We assume three kinds of observable actions for peers: arrival of a new query from p, written $q(p)$; selection of a query to serve and connection to its sender p, written $s(p)$; disconnection from p, written $d(p)$.

Variants of a communication protocol in this setting may have very different behaviors. Policies characterizing desired ones, for instance fairness requirements, can be specified by automata. For decidability reasons, policies should be *deterministic*: they should always consider all possible actions from a given state, even if not all of them will be accepted, and each action should have a unique outcome. Our main example will be the specification of a "first come first serve" peer selection policy for queries, named *FCFS fair policy*; we shall also discuss a policy that takes into account a number of locally identified "friend peers" taking priority over the others, that we call *friend policy*.

FCFS fair policy. We model query selection by a *fair* FCFS discipline: queries from already buffered peers are discarded. We assume that the buffer has size n. The automaton is shown in Figure 1. Each state q_i, s_i is equipped with i registers, for $i = 1, \ldots, n$, and q_0 has no registers. Registers can store identifiers, and are local to states (we will discuss this aspect throughout the paper), thus each transition is equipped with a function expressing how registers in the target state take values from those of the source state. We have two such functions: $enq_i(x)$, mapping i to $\star$ and $x < i$ to x, and deq_i, mapping x to $x + 1$. The intuition is that transitions from q_i to q_{i+1} labeled with $\star$ correspond to buffering a "fresh" query, from a peer whose identiy is not already known, and thus it is stored in register $i + 1$. The loop on q_n discards new peers when the buffer is full. The transition from q_i to s_i picks the query p from register 1, that is always the oldest one, and establishes a connection to p; the transition from s_i to q_{i-1} removes p from the buffer, and shifts the registers' content so that register 1

contains the query that arrived right after the one of p. Our automaton should (1) be deterministic and (2) have a Muller-style accepting condition. For (1), we assume each state has all possible outgoing transitions: those not shown in Figure 1 are assumed to go to a sink state. For (2), we take all subsets of the states, excluding the sink one, as Muller sets; that is: behaviors that go through states and transitions depicted in Figure 1 are all accepted.

Friend queries. A *friend query* is a query coming from a "friend" peer, which should be served as soon as possible, that is: after the current query has been served. To model such scenarios, one can introduce an action $q_f(p)$ to model a friend query from p. An automaton that correctly handles such queries can be obtained from the one of Figure 1 as follows: we add a transition from q_i to q_{i+1}, for each $i = 0, \ldots, n-1$, labelled with $q_f(\star)$ and with the map top_i, sending 1 to $\star$ and $x > 1$ to $x - 1$; furthermore, a looping transition on q_n is added with label $q_f(\star)$ and map id, which discards friend queries when the buffer is full. Intuitively, top_i always stores the friend query in register 1, so that transitions $s(1)$ will always pick it, and shifts the priority of all the other peers.

3. Background

Notation.. Throughout the paper: $f\colon X \to Y$ is a total function, $f\colon X \rightarrowtail Y$ is total and injective, $f\colon X \rightharpoonup Y$ is partial; $\mathrm{dom}(f)$ is the subset of X where f is defined, $Im(f)$ its image. Symbol ω denotes the set of natural numbers. For s a sequence, we let s_i or $s(i)$ denote its i^{th} element. R^* is the symmetric, transitive, reflexive closure of binary relation R. We use $\circ$ for (partial) function composition and also for "relational" composition, as usual, by seeing functions as relations.

We shall now briefly introduce nominal sets; we refer the reader to [**?**] for more details on the subject. We assume a countable set of *names* $\mathcal{N}$, and we write $\mathbb{P}$ for the group of finite-kernel permutations of $\mathcal{N}$, namely those bijections $\pi\colon \mathcal{N} \to \mathcal{N}$ such that the set $\{a \mid \pi(a) \neq a\}$ is finite.

Definition 3.1. A *nominal set* is a set X along with an action for $\mathbb{P}$, that is a function $\cdot\colon \mathbb{P} \times X \to X$ such that, for all $x \in X$ and $\pi, \pi' \in \mathbb{P}$, $id_{\mathcal{N}} \cdot x = x$ and $(\pi \circ \pi') \cdot x = \pi \cdot (\pi' \cdot x)$. Also, it is required that each $x \in X$ has *finite support*, meaning that there exists a finite $S \subseteq \mathcal{N}$ such that, for all $\pi \in \mathbb{P}$, $\pi|_S = id_S$ implies $\pi \cdot x = x$. We denote the least[2] such S with $supp(x)$. An *equivariant function* from nominal set X to nominal set Y is a function $f : X \to Y$ such that, for all π and x, $f(\pi \cdot x) = \pi \cdot f(x)$.

Definition 3.2. Given $x \in X$, the *orbit* of x, denoted by $orb(x)$, is the set $\{\pi \cdot x \mid \pi \in \mathbb{P}\} \subseteq X$. For $S \subseteq X$, we write $orb(S)$ for $\{orb(x) \mid x \in S\}$. We call X *orbit-finite* when $orb(X)$ is finite.

Note that $orb(X)$ is a partition of X. The prototypical nominal set is $\mathcal{N}$ with $\pi \cdot a = \pi(a)$ for each $a \in \mathcal{N}$; we have $supp(a) = \{a\}$, and $orb(a) = \mathcal{N}$.

[2]It is a theorem that whenever there is a finite support, there is also a least support.

4. Nominal ω-regular languages

In the following, we extend *Muller automata* to the case of nominal alphabets. Traditionally, automata can be deterministic or non-deterministic. In the case of finite words, non-deterministic nominal automata are not closed under complementation, whereas the deterministic ones are; similar considerations apply to the infinite words case. Thus, we adopt the deterministic setting in order to retain complementation.

Definition 4.1. A *nominal deterministic Muller automaton* (nDMA) is a tuple $(Q, \longrightarrow, q_0, \mathcal{A})$ where:

- Q is an orbit-finite nominal set of *states*, with $q_0 \in Q$ the *initial state*;
- $\mathcal{A} \subseteq \mathcal{P}(orb(Q))$ is a set of sets of orbits, intended to be used as an acceptance condition in the style of Muller automata.
- $\longrightarrow$ is the *transition relation*, made up of triples $q_1 \xrightarrow{a} q_2$, having *source* q_1, *target* q_2, *label* $a \in \mathcal{N}$;
- the transition relation is *deterministic*, that is, for each $q \in Q$ and $a \in \mathcal{N}$ there is exactly one transition with source q and label a;
- the transition relation is *equivariant*, that is, invariant under permutation: there is a transition $q_1 \xrightarrow{a} q_2$ if and only if, for all π, also the transition $\pi \cdot q_1 \xrightarrow{\pi(a)} \pi \cdot q_2$ is present.

In nominal sets terminology, the transition relation is an *equivariant function* of type $Q \times \mathcal{N} \to Q$. Notice that nDMA are infinite state, infinitely branching machines, even if orbit finite. For effective constructions we employ equivalent finite structures (see Section 5). Definition 4.1 induces a simple definition of acceptance, very close to the classical one. In the following, fix a nDMA $A = (Q, \longrightarrow, q_0, \mathcal{A})$.

Definition 4.2. An infinite *word* $\alpha \in \mathcal{N}^\omega$ is an infinite sequence of symbols in $\mathcal{N}$. Words have *point-wise* permutation action, namely $(\pi \cdot \alpha)_i = \pi(\alpha_i)$, making a word finitely supported if and only if it contains finitely many different symbols.

Definition 4.3. Given a word $\alpha \in \mathcal{N}^\omega$, a *run* of α from $q \in Q$ is a sequence of states $r \in Q^\omega$, such that $r_0 = q$, and for all i we have $r_i \xrightarrow{\alpha_i} r_{i+1}$. By determinism (see Definition 4.1), for each infinite word α, and each state q, there is exactly one run of α from q, that we call $r^{\alpha,q}$, or simply r^α when $q = q_0$.

Definition 4.4. For $r \in Q^\omega$, let $Inf(r)$ be the set of *orbits* that r traverses infinitely often, i.e., $orb(q) \in Inf(r)$ iff. for all i, there is $j > i$ s.t. $r_j \in orb(q)$.

Definition 4.5. A word α is *accepted* by state q whenever $Inf(r^{\alpha,q}) \in \mathcal{A}$. We let $\mathcal{L}_{A,q}$ be the set of all accepted words by q in A; we omit A when clear from the context, and q when it is q_0, thus $\mathcal{L}_A$ is the language of the automaton A. We say that $\mathcal{L} \subseteq \mathcal{N}^\omega$ is a *nominal ω-regular language* if it is accepted by a nDMA.

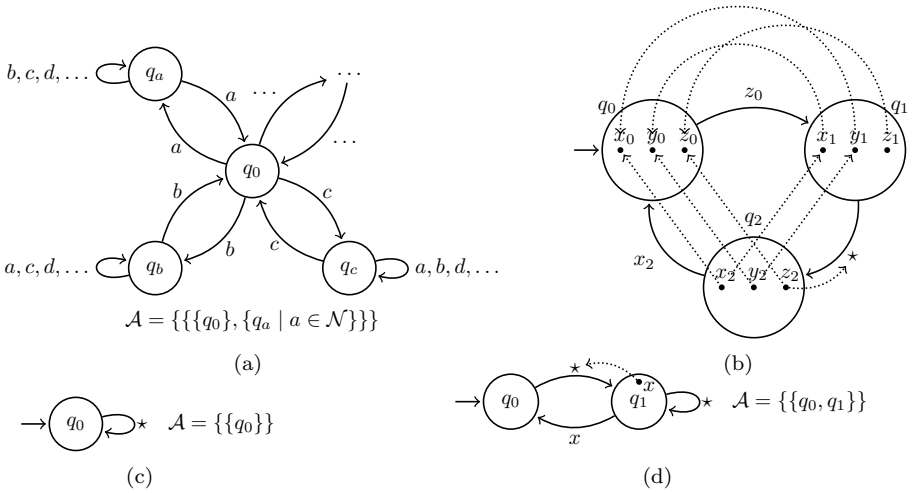

Figure 2: Some automata, together with their accepting conditions.

Remark 4.6. We use $\mathcal{N}$ as alphabet. One can chose any orbit-finite nominal set; the definitions of automata and acceptance are unchanged, and finite representations are similar. Using $\mathcal{N}$ simplifies the presentation, especially in Section 5.

Example 4.7. Consider the nDMA in Figure 2(a). We have $Q = \{q_0\} \cup \{q_a \mid a \in \mathcal{N}\}$. For all π, we let $\pi \cdot q_0 = q_0$, $\pi \cdot q_a = q_{\pi(a)}$. We have $supp(q_0) = \emptyset$, and $supp(q_a) = \{a\}$. For all a, let $q_0 \xrightarrow{a} q_a$, $q_a \xrightarrow{a} q_0$, and for $b \neq a$, $q_a \xrightarrow{b} q_a$. Each of the infinite "legs" of the automaton rooted in q_0 remembers a different name, and returns to q_0 when the same name is encountered again. There are two orbits, namely $orb_0 = \{q_0\}$ and $orb_1 = \{q_a \mid a \in \mathcal{N}\}$. We let $\mathcal{A} = \{\{orb_0, orb_1\}\}$. For acceptance, a word needs to cross both orbits infinitely often. Thus, $\mathcal{L}_{q_0} = \{aua \mid a \in \mathcal{N}, u \in (\mathcal{N} \setminus \{a\})^*\}^\omega$. This is an idealized version of a service, where each in a number of potentially infinite users (represented by names) may access the service, reference other users, and later leave. Infinitely often, an arbitrary symbol occurs, representing an "access"; the next occurrence of the same symbol denotes a "leave". One could use an alphabet with two infinite orbits to distinguish the two kinds of action (see Remark 4.6), or reserve two distinguished names of $\mathcal{N}$ to be used as "brackets" before the different occurrences of other names, adding more states.

Accepted words may fail to be finitely supported. However, languages are. This adheres to the intuition that a machine running forever may read an unbounded amount of different pieces of data, but still have finite memory.

Theorem 4.8. *For $\mathcal{L}$ a language, and $\pi \in \mathbb{P}$, let $\pi \cdot \mathcal{L} = \{\pi \cdot \alpha \mid \alpha \in \mathcal{L}\}$. For each state q of an nDMA, $\mathcal{L}_q$ is finitely supported.*

5. Finite automata

In this section, we introduce finite representations of nDMAs. These are similar to classical finite-state automata, but each state is equipped with local registers. There is a notion of assignment to registers, and it is possible to accept, and eventually store, *fresh* symbols. Technically, these structures extend *history-dependent automata* (see [**?**]), introducing acceptance of infinite words.

Definition 5.1. A *history-dependent deterministic Muller automaton* (hDMA) is a tuple $(Q, |-|, q_0, \rho_0, \longrightarrow, \mathcal{A})$ where:

- Q is a finite set of *states*;
- for $q \in Q$, $|q|$ is a finite set of *local names* (or *registers*) of state q;
- $q_0 \in Q$ is the *initial state*;
- $\rho_0 : |q_0| \rightarrowtail \mathcal{N}$ is the *initial assignment*;
- $\mathcal{A} \subseteq \mathcal{P}(Q)$ is the *accepting condition*, in the style of *Muller automata*;
- $\longrightarrow$ is the *transition relation*, made up of quadruples $q_1 \xrightarrow[\sigma]{l} q_2$, having *source* q_1, *target* q_2, label $l \in |q_1| \uplus \{\star\}$, and *history* $\sigma : |q_2| \rightarrowtail |q_1| \cup \{l\}$;
- the transition relation is *deterministic* in the following sense: for each $q_1 \in Q$, there is exactly one transition with source q_1 and label $\star$, and exactly one transition with source q_1 and label x for each $x \in |q_1|$.

Remark 5.2. To keep the notation lightweight, we do not use a *symmetry* attached to states of an hDMA. It is well known (see [**?**]) that symmetries are needed for existence of canonical representatives; we consider this aspect out of the scope of this work. Note that (classical) Muller automata do not have canonical representatives up-to language equivalence. To obtain those, one can use two-sorted structures as in [**?**]. Even though this idea could be applied to hDMAs, this is not straightforward, and requires further investigation.

In the following we fix an hDMA $A = (Q, |-|, q_0, \rho_0, \longrightarrow, \mathcal{A})$. We overload notation (e.g., for the inf-set of a word) from section 4, as it will be always clear from the context whether we are referring to an nDMA or to an hDMA. Acceptance of $\alpha \in \mathcal{N}^\omega$ is defined using the *configuration graph* of A.

Definition 5.3. The set $\mathcal{C}(A)$ of *configurations* of A consists of the pairs (q, ρ) such that $q \in Q$ and $\rho : |q| \rightarrowtail \mathcal{N}$ is an injective *assignment* of names to registers.

Definition 5.4. The *configuration graph* of A is a graph with edges of the form $(q_1, \rho_1) \xrightarrow{a} (q_2, \rho_2)$ where the source and destination are configurations, and $a \in \mathcal{N}$. There is one such edge iff there is a transition $q_1 \xrightarrow[\sigma]{l} q_2$ in A and either $l \in |q_1|$, $\rho_1(l) = a$, and $\rho_2 = \rho_1 \circ \sigma$, or $l = \star$, $a \notin Im(\rho_1)$, $\rho_2 = (\rho_1 \circ \sigma)[^a/_{\sigma^{-1}(\star)}]$.

The definition deserves some explanation. Fix a configuration (q_1, ρ_1). Say that name $a \in \mathcal{N}$ is *assigned to* the register $x \in |q_1|$ if $\rho_1(x) = a$. When a is not assigned to any register, it is *fresh* for a given configuration. Then the transition $q_1 \xrightarrow[\sigma]{l} q_2$, under the assignment ρ_1, consumes a symbol as follows: either $l \in |q_1|$ and a is the name assigned to register l, or l is $\star$ and a is fresh. The destination assignment ρ_2 is defined using σ as a binding between local registers of q_2 and local registers of q_1, therefore composing σ with ρ_1 and eventually adding a freshly received name, whenever $\star$ is in the image of σ. For readability, we assume that the functional update $[a/\sigma^{-1}(\star)]$ is void when $\star \notin Im(\sigma)$. The following lemma clarifies the notion of determinism that we use.

Lemma 5.5. *For each configuration (q_1, ρ_1) and symbol $a \in \mathcal{N}$, there is exactly one configuration (q_2, ρ_2) such that $(q_1, \rho_1) \xrightarrow{a} (q_2, \rho_2)$.*

We use the notation $(q_1, \rho_1) \overset{v}{\Longrightarrow} (q_2, \rho_2)$ to denote a path that spells v in the configuration graph. Furthermore, we define runs of infinite words.

Definition 5.6. A *run* r of an infinite word $\alpha \in \mathcal{N}^\omega$ from configuration (q, ρ) is a sequence (q_i, ρ_i) of configurations, indexed by ω, such that $(q_0, \rho_0) = (q, \rho)$ and for all i, in the configuration graph, we have $(q_i, \rho_i) \xrightarrow{\alpha_i} (q_{i+1}, \rho_{i+1})$.

Finally, after a simple corollary of Lemma 5.5, we define acceptance of hDMAs.

Proposition 5.7. *Given $(q_1, \rho_1) \in \mathcal{C}(A)$ and $v \in \mathcal{N}^\omega$, there exists a unique path $(q_1, \rho_1) \overset{v}{\Longrightarrow} (q_2, \rho_2)$ in the configuration graph of A. Similarly, for each word α and configuration (q, ρ), there is a unique run $r^{\alpha,q,\rho}$ from (q, ρ). We omit q and ρ from the notation, when dealing with the* initial configuration *(q_0, ρ_0).*

Definition 5.8. Consider the unique run r of an infinite word α from configuration (q, ρ). Let $Inf(r)$ denote the set of states that appear infinitely often in the first component of r. By finiteness of Q, $Inf(r)$ is not empty. The automaton A accepts α whenever $Inf(r) \in \mathcal{A}$. In this case, we speak of the *language* $\mathcal{L}_A$ of words accepted by the automaton.

As an example, the language $\mathcal{N}^\omega$ of all infinite words over $\mathcal{N}$ is recognised by the hDMA in Figure 2(c); the initial assignment ρ_0 is necessarily empty, and so is the history σ along the transition. Differently from nDMAs, hDMAs have finite states. Finite representations are useful for effective operations on languages, as we shall see later. The similarity between configuration graphs of hDMAs, and nDMAs, is deep and is the essence of the proof of Proposition 5.9 below. These are similar to the categorical equivalence results in [**?** **?**]; however, notice that representing infinite branching systems using "allocating transitions" requires further machinery, similar to what is studied in [**?**].

Proposition 5.9. *For each (orbit-finite) nDMA there is a finite hDMA accepting the same language, and vice versa.*

Example 5.10. Consider the hDMA in Figure 2(d), where the labelled dot within q_1 represents its register, and the dashed line depicts the history from q_1 to q_0 (we omit empty histories). This automaton accepts the language of Example 4.7. In fact, q_0 is the only element in the orbit of the initial state of the nDMA, and q_1 canonically represents all q_a, $a \in \mathcal{N}$. This notation for hDMAs will be used throughout the paper.

6. Synchronized product and boolean operations

The product of two finite automata uses the well-known *synchronized product* construction. In this section we define this operation on the underlying *transition structures* of hDMAs, i.e. on tuples $\mathcal{T} = (Q, |-|, q_0, \rho_0, \longrightarrow)$ (we want to be parametric w.r.t. the accepting condition). One should be careful in handling registers. When forming pairs of states, some of these registers could be constrained to have the same value. Thus, states have the form (q_1, q_2, R), where R is a relation linking registers of q_1 and q_2 that represent the same register in the synchronized product. This is implemented by quotienting registers w.r.t. the equivalence R^* induced by R; the construction is similar to the case of register automata, and to the construction of products in named sets given in [**?**].

Given two transition structures $\mathcal{T}_i = (Q_i, |-|_i, q_0^i, \rho_0^i, \longrightarrow_i)$, $i = 1, 2$, we define their synchronized product $\mathcal{T}_1 \otimes \mathcal{T}_2$. Given $q_1 \in Q_1$,$q_2 \in Q_2$, $Reg(q_1, q_2)$ is the set of relations that are allowed to appear in states of the form (q_1, q_2, R), namely those $R \subseteq |q_1|_1 \times |q_2|_2$ such that, for each $(x, y) \in R$, there is no other $(x', y') \in R$ with $x' = x$ or $y' = y$. This avoids inconsistent states where the individual assignment for q_1 or q_2 would not be injective. In the following we assume $[x]_{R^*}$ (the canonical representative of the equivalence class of x in R^*) to be $\{x\}$ when x does not appear in any pair of R.

Definition 6.1. $\mathcal{T}_1 \otimes \mathcal{T}_2$ is the tuple $(Q_\otimes, |-|_\otimes, q_0^\otimes, \rho_0^\otimes, \longrightarrow_\otimes)$ where:

- $Q_\otimes := \{(q_1, q_2, R) \mid q_1 \in Q_1, q_2 \in Q_2, R \in Reg(q_1, q_2)\}$;
- $|(q_1, q_2, R)|_\otimes := (|q_1|_1 \cup |q_2|_2)_{/R^*}$, for $(q_1, q_2, R) \in Q_\otimes$;
- $q_0^\otimes := (q_0^1, q_0^2, R_0)$, where $R_0 := \{(x_1, x_2) \in |q_0^1|_1 \times |q_0^2|_2 \mid \rho_0^1(x_1) = \rho_0^2(x_2)\}$;
- $\rho_0([x]_{R_0^*}) = \rho_0^i(x)$ whenever $x \in |q_0^i|_i$, $i \in \{1, 2\}$;
- transitions are generated by the following rules

(Reg)

$$\frac{q_1 \xrightarrow[\sigma_1]{l_1}_1 q_1' \qquad q_2 \xrightarrow[\sigma_2]{l_2}_2 q_2' \qquad \exists i \in \{1,2\} : l_i \in \mathcal{N} \wedge [l_i]_{R^*} = \{l_1, l_2\} \cap \mathcal{N}}{(q_1, q_2, R) \xrightarrow[\sigma_R]{[l_i]_{R^*}}_\otimes (q_1', q_2', S)}$$

(Alloc)

$$\frac{q_1 \xrightarrow[\sigma_1]{l_1}_1 q_1' \quad q_2 \xrightarrow[\sigma_2]{l_2}_2 q_2' \quad l_1, l_2 = \star}{(q_1, q_2, R) \xrightarrow[\sigma_A]{\star}_\otimes (q_1', q_2', S)}$$

where $S := \sigma_2^{-1} \circ R \cup \{(l_1, l_2)\} \circ \sigma_1$ and

$$\sigma_\tau([x]_{S^*}) := \begin{cases} [\sigma_i(x)]_{R^*} & x \in |q_i'|_i \wedge \sigma_i(x) \neq \star \\ [l_{3-i}]_{R^*} & x \in |q_i'|_i \wedge \sigma_i(x) = \star \wedge \tau = R \\ \star & x \in |q_i'|_i \wedge \sigma_i(x) = \star \wedge \tau = A \end{cases}$$

Before explaining in detail the formal definition, we remark that the relation S is well defined, i.e. it belongs to $Reg(q_1', q_2')$: the addition of $\{(l_1, l_2)\}$ to R is harmless, as will be explained in the following, and σ_1 and σ_2^{-1} can never map the same value to two different values (as they are functions) or vice versa (as they are injective). The definition of $q_0^\otimes$ motivates the presence of relations in states: R_0-related registers are the ones that are assigned the same value by ρ_0^1 and ρ_0^2; these form the same register of $q_0^\otimes$, so $\rho_0^\otimes$ is well-defined. The synchronization mechanism is implemented by rules (Reg) and (Alloc): they compute transitions of $(q_1, q_2, R) \in Q_\otimes$ from those of q_1 and q_2 as follows.

Rule (Reg) handles two cases. First, if the transitions of q_1 and q_2 are both labelled by registers, say l_1 and l_2, and these registers correspond to the same one in (q_1, q_2, R) (condition $[l_i]_{R^*} = \{l_1, l_2\}$, recalling that $[l_i]_{R^*}$ cannot contain more labels due to injectivity of register maps), then (Reg) infers a transition labelled with $[l_i]_{R^*}$ (the specific i is not relevant). The target state of this transition is made of those of the transitions from q_1 and q_2, plus a relation S obtained by translating R-related registers to S-related registers via σ_1 and σ_2. In this case, adding the pair (l_1, l_2) to R in the definition of S has no effect, as it is already in R. The inferred history σ_R just combines σ_1 and σ_2, consistently with S^*.

The other case for (Reg) is when a fresh name is consumed from just one state, e.g. q_2. This name must coincide with the value assigned to the register l_1 labelling the transition of q_1. Therefore the inferred label is $[l_1]_{R^*}$. The target relation S changes slightly. Suppose there are $l_1' \in |q_1'|$ and $l_2' \in |q_2'|$ such that $\sigma_1(l_1') = l_1$ and $\sigma_2(l_2') = \star$; after q_1 and q_2 perform their transitions, both these registers are assigned the same value, so we require $(l_1', l_2') \in S$. This pair is forced to be in S by adding $(l_1, \star)$ to R when computing S. This does not harm well-definedness of S, because $[l_1]_{R^*}$ is a singleton (rule premise $[l_1]_{R^*} = \{l_1, \star\} \cap \mathcal{N} = \{l_1\}$), so no additional, inconsistent identifications are added to S^* due to transitivity. If either l_1 or $\star$ is not in the image of the corresponding history map, then augmenting R has no effect, as the relational composition discards $(l_1, \star)$. The history σ_R should map $[l_2']_{S^*}$ to $[l_1]_{R^*}$: this is treated by the second case of its definition; all the other values are mapped as before.

Transitions of q_1 and q_2 consuming a fresh name on both sides are turned by (Alloc) into a unique transition with freshness: S is computed by adding $(\star, \star)$ to R, thus the registers to which the fresh name is assigned (if any) form one register in the overall state; the inferred history σ_A gives the freshness status to this register, and acts as usual on other registers.

Remark 6.2. $\mathcal{T}_1 \otimes \mathcal{T}_2$ is finite-state and deterministic. In fact, every set in the definition of $Q_\otimes$ is finite. As for determinism, given $(q_1, q_2, R) \in Q_\otimes$, each

$l \in |(q_1, q_2, R)|_\otimes \cup \{\star\}$ uniquely determines which labels l_1 and l_2 should appear in the rule premises (e.g. if $l = \{l_1\}$, with $l_1 \in |q_1|_1$, then $l_2 = \star$), and by determinism each q_i can do a unique transition labeled by l_i.

We shall now relate the configuration graphs of $\mathcal{T}_1 \otimes \mathcal{T}_2$, $\mathcal{T}_1$ and $\mathcal{T}_2$.

Definition 6.3. Let $((q_1, q_2, R), \rho) \in \mathcal{C}(\mathcal{T}_1 \otimes \mathcal{T}_2)$. Its i-th projection, denoted π_i, is defined as $\pi_i((q_1, q_2, R), \rho) = (q_i, \rho_i)$ with $\rho_i := \lambda x \in |q_i|_i.\rho([x]_{R^*})$

Projections always produce valid configurations in $\mathcal{C}(\mathcal{T}_1)$ and $\mathcal{C}(\mathcal{T}_2)$: injectivity of ρ_i follows from the definition of $Reg(q_1, q_2)$, ensuring that two different $x_1, x_2 \in |q_i|_i$ cannot belong to the same equivalence class of R^*, i.e. cannot have the same image through ρ_i. The correspondence between edges is formalized as follows.

Proposition 6.4. *Given $C \in \mathcal{C}(\mathcal{T}_1 \otimes \mathcal{T}_2)$: if $C \xrightarrow{a} C'$ then $\pi_i(C) \xrightarrow{a} \pi_i(C')$, $i = 1, 2$; (i) if $\pi_i(C) \xrightarrow{a}_i C_i$, $i = 1, 2$, then $\exists C' : C \xrightarrow{a} C'$ and $\pi_i(C) = C_i$.*

Corollary 6.5. *Let $C_0 = (q_0^\otimes, \rho_0)$. We have a path $C_0 \xrightarrow{a_0} \dots \xrightarrow{a_{n-1}} C_n$ in the configuration graph of $\mathcal{T}_1 \otimes \mathcal{T}_2$ if and only if we have paths $\pi_i(C_0) \xrightarrow{a_0} \dots \xrightarrow{a_{n-1}} \pi_i(C_n)$ in the configuration graphs of $\mathcal{T}_i$, for $i = 1, 2$. The correspondence clearly holds also for infinite paths, i.e. runs.*

This result allows us to relate the *Inf* of runs in the defined transition structures.

Theorem 6.6. *Given $\alpha \in \mathcal{N}^\omega$, let r be a run for α in the configuration graph of $\mathcal{T}_1 \otimes \mathcal{T}_2$, and let r_1 and r_2 be the corresponding runs for $\mathcal{T}_1$ and $\mathcal{T}_2$, according to Corollary 6.5. Then $\pi_1(Inf(r)) = Inf(r_1)$ and $\pi_2(Inf(r)) = Inf(r_2)$.*

Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be ω-regular nominal languages, and let $A_1 = (\mathcal{T}_1, \mathcal{A}_1)$ and $A_2 = (\mathcal{T}_2, \mathcal{A}_2)$ be automata for these languages, where $\mathcal{T}_1$ and $\mathcal{T}_2$ are the underlying transition structures. By Theorem 6.6 above, we are now able to show that constructing the automaton for a boolean combination of $\mathcal{L}_1$ and $\mathcal{L}_2$ amounts to defining an appropriate accepting set for $\mathcal{T}_1 \otimes \mathcal{T}_2$.

Theorem 6.7. *Using the transition structure $\mathcal{T}_1 \otimes \mathcal{T}_2$, define the accepting conditions $\mathcal{A}_\cap = \{S \subseteq Q_\otimes \mid \pi_1(S) \in \mathcal{A}_1 \wedge \pi_2(S) \in \mathcal{A}_2\}$, $\mathcal{A}_\cup = \{S \subseteq Q_\otimes \mid \pi_1(S) \in \mathcal{A}_1 \vee \pi_2(S) \in \mathcal{A}_2\}$ and $\mathcal{A}_{\overline{\mathcal{L}_1}} = \mathcal{P}(Q_1) \setminus \mathcal{A}_1$, where Q_1 are the states of A_1. The obtained hDMAs accept, respectively, $\mathcal{L}_1 \cap \mathcal{L}_2$, $\mathcal{L}_1 \cup \mathcal{L}_2$, and $\overline{\mathcal{L}_1}$.*

Theorem 6.8. *Emptiness and, as a corollary, equality of ω-regular nominal languages are decidable.*

7. Ultimately-periodic words

An *ultimately periodic* word is a word of the form uv^ω, with u, v finite words. Given a language of infinite words $\mathcal{L}$, let $UP(\mathcal{L})$ be its *ultimately periodic fragment* $\{\alpha \in \mathcal{L} \mid \alpha = uv^\omega \wedge u, v \text{ are finite}\}$. It has been proven in [**? ?**] that, for

every two ω-regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$, $UP(\mathcal{L}_1) = UP(\mathcal{L}_2)$ implies $\mathcal{L}_1 = \mathcal{L}_2$, i.e. ω-regular languages are characterised by their ultimately periodic fragments. In this section we aim to extend this result to the nominal setting.

The preliminary result to establish, as in the classical case, is that every non-empty nominal ω-regular language $\mathcal{L}$ contains at least one ultimately periodic word. For ω-regular languages, this involves finding a loop through accepting states in the automaton and iterating it. For hDMAs, freshness constraints could forbid consuming the same name in consecutive traversals of the same transition. We first show that, given a loop in a hDMA, there always is a path induced by consecutive traversals of the loop, such that its initial and final configurations coincide. Thus, such path can be taken an arbitrary number of times.

Fix a loop $L := p_0 \xrightarrow[\sigma_0]{l_0} p_1 \xrightarrow[\sigma_1]{l_1} \dots \xrightarrow[\sigma_{n-1}]{l_{n-1}} p_0$ (the specific hDMA is not relevant). We write $\underline{i}$ for $i \mod n$. For all $i = 0, \dots, n-1$, let $\widehat{\sigma}_i \colon |p_{\underline{i+1}}| \rightharpoonup |p_i|$ be the partial maps telling the history of old registers and ignoring the new ones, formally $\widehat{\sigma}_i := \sigma_i \setminus \{(x, y) \in \sigma_i \mid y = \star\}$, and let $\widehat{\sigma} \colon |p_0| \rightharpoonup |p_0|$ be their composition $\widehat{\sigma}_0 \circ \widehat{\sigma}_1 \cdots \circ \widehat{\sigma}_{n-1}$. We define the set I as the greatest subset of $\mathrm{dom}(\widehat{\sigma})$ such that $\widehat{\sigma}(I) = I$, i.e. I are the registers of p_0 that "survive" along L. We denote by T all the other registers, namely $T := |p_0| \setminus I$. These are registers whose content is eventually discarded (not necessarily within a single loop traversal), as the following lemma states.

Lemma 7.1. *Given any $x \in T$, let $\{x_j\}_{j \in J_x}$ be the smallest sequence that satisfies the following conditions: $x_0 = x$ and $x_{j+1} = \sigma_{\underline{j}}^{-1}(x_j)$, where $j + 1 \in J_x$ only if $\sigma_{\underline{j}}^{-1}(x_j)$ is defined. Then J_x has finite cardinality.*

Now, consider any assignment $\hat{\rho}_0 \colon |p_0| \to \mathcal{N}$. We give some lemmata about paths that start from $(p_0, \hat{\rho}_0)$ and are induced by consecutive traversals of L. The first one says that the assignment for I given by $\hat{\rho}_0$ is always recovered after a fixed number of traversals of L, regardless of which symbols are consumed. In the following, given a sequence of transitions P, we write $(q_1, \rho_1) \overset{v}{\Longrightarrow}_P (q_2, \rho_2)$ whenever $(q_1, \rho_1) \overset{v}{\Longrightarrow} (q_2, \rho_2)$ and such path is induced by P.

Lemma 7.2. *There is $\theta \geq 1$ such that, for all $v_0, \dots, v_{\theta-1}$ satisfying $(p_0, \hat{\rho}_0) \overset{v_0}{\Longrightarrow}_L (p_0, \hat{\rho}_1) \overset{v_1}{\Longrightarrow}_L \dots \overset{v_{\theta-1}}{\Longrightarrow}_L (p_0, \hat{\rho}_\theta)$ we have $\hat{\rho}_\theta|_I = \hat{\rho}_0|_I$.*

The second one says that, after a minimum number of traversals of L, a configuration can be reached where the initial values of T, namely those assigned by $\hat{\rho}_0$, cannot be found in any of the registers.

Lemma 7.3. *There is $\epsilon \geq 1$ s.t., for all $\gamma \geq \epsilon$,there are $v_0, \dots, v_{\gamma-1}$ satisfying $(p_0, \hat{\rho}_0) \overset{v_0}{\Longrightarrow}_L (p_0, \hat{\rho}_1) \overset{v_1}{\Longrightarrow}_L \dots \overset{v_{\gamma-1}}{\Longrightarrow}_L (p_0, \hat{\rho}_\gamma)$, with $Im(\hat{\rho}_\gamma) \cap \hat{\rho}_0(T) = \varnothing$.*

We give the dual of the previous lemma: if we start from a configuration where registers are not assigned values in $\hat{\rho}_0(T)$, then these values can be assigned back to T in a fixed number of traversals of L, regardless of the initial assignment.

Lemma 7.4. *There is $\zeta \geq 1$ such that, for any $\tilde{\rho}_0 \colon |p_0| \to \mathcal{N}$ with $Im(\tilde{\rho}_0) \cap \hat{\rho}_0(T) = \varnothing$, there are $v_0, \ldots, v_{\zeta-1}$ satisfying $(p_0, \tilde{\rho}_0) \overset{v_0}{\Longrightarrow}_L (p_0, \tilde{\rho}_1) \overset{v_1}{\Longrightarrow}_L \cdots \overset{v_{\zeta-1}}{\Longrightarrow}_L (p_0, \tilde{\rho}_\zeta)$, with $\tilde{\rho}_\zeta|_T = \hat{\rho}_0|_T$.*

Finally, we combine the above lemmata. We construct a path where: (1) the values assigned to T are forgotten and then recovered (2) the values assigned to I are swapped, but the initial assignment is periodically regained. Therefore, the length of such path should allow (1) and (2) to "synchronize", so that the final assignment is again $\hat{\rho}_0$.

Theorem 7.5. *For each loop L with initial state p_0, and assignment $\hat{\rho}_0 \colon |p_0| \to \mathcal{N}$, there are $v_0, \ldots, v_n$ such that $(p_0, \hat{\rho}_0) \overset{v_0}{\Longrightarrow}_L (p_0, \hat{\rho}_1) \overset{v_1}{\Longrightarrow}_L \cdots \overset{v_n}{\Longrightarrow}_L (p_0, \hat{\rho}_0)$.*

Example 7.6. We justify the construction on the hDMA of Figure 2(b), with initial assignment $\rho_0(x_0) = a$,$\rho_0(y_0) = b$ and $\rho_0(z_0) = c$. Consider the loop L formed by all the depicted transitions. We have $I = \{x_0, y_0\}$ and $T = \{z_0\}$. Look at the path $(q_0, [a/x_0, b/y_0, c/z_0]) \xrightarrow{c} (q_1, [b/x_1, a/y_1, c/z_1]) \xrightarrow{d} (q_2, [b/x_2, a/y_2, d/z_2]) \xrightarrow{b}$ $(q_0, [b/x_0, a/y_0, d/z_0])$ where $d \neq a, b, c$. The values of x_0 and y_0 are swapped according to the permutation $(a\ b)$, and d is assigned to z_0. Our aim is to recover ρ_0 again. According to Lemma 7.2, x_0 and y_0 get their assignment back in $\theta = 2$ traversals of L (in fact $(a\ b)^2 = (a\ b)$). As for z_0, its assignment is established in the second transition, but c should not have been assigned to any register of q_1 in order for it to be consumed during this transition. This is where Lemma 7.3 comes into play: it says that in at least $\epsilon = 1$ traversals of L the name c is discarded. This is exactly what happens in the path shown above. Then we can assign c to z_0 in another $\zeta = 1$ traversal of L, according to Lemma 7.4. Since $\epsilon + \zeta = \theta = 2$, traversing L twice is enough (e.g., consider the path *cdbdca*).

Finally we introduce the main results of this section.

Theorem 7.7. *When $\mathcal{L}$ is a non-empty nominal ω-regular language, $UP(\mathcal{L}) \neq \emptyset$.*

Theorem 7.8. *For $\mathcal{L}_1, \mathcal{L}_2$ nominal ω-regular, $UP(\mathcal{L}_1) = UP(\mathcal{L}_2) \implies \mathcal{L}_1 = \mathcal{L}_2$.*

Note that a similar result could not be achieved in the presence of so-called *global freshness* [**?**], e.g. the one-state automaton accepting only globally fresh symbols would have empty ultimately periodic fragment, just like the empty language. As a concluding remark, we note that, by Theorem 7.8, every ω-regular language is characterized by a sublanguage of finitely supported words (the support of uv^ω just contains the finitely many symbols in u and v). We find this result appealing, given the central role of the notion of support in the nominal setting.

8. Conclusions

This work is an attempt to provide a simple definition that merges the theories of nominal automata and ω-regular languages, retaining effective closure

under boolean operations, and decidability of emptiness, and language equivalence. We sketch some possible future directions. A very relevant application of formal verification in the presence of fresh resources could be model-checking of nominal process calculi. However, the presented theory only accommodates the deterministic case; undecidability issues arise for non-deterministic systems. Future work will be directed to identify (fragments of) nominal calculi that retain decidability. For this, one needs to limit not only non-determinism, but also parallel composition (again, decidability may be an issue otherwise). A calculus that could be handled by the current theory is a deterministic, finite-control variant of the π-calculus; capturing analogous versions of more recent calculi, e.g., ψ-calculi [**?**], should be possible, as they are based on nominal structures with notions of permutation action, support, orbits. As mentioned in section 2, we argue that deterministic behavior is enough to specify meaningful policies. Furthermore, recall that automata correspond to logic formulae: hDMAs could be used to represent logic formulae with binders; it would also be interesting to investigate the relation with first-order logic on nominal sets [**?**]. There may be different logical interpretations of hDMAs, where causality or dependence [**?** **?**] between events are made explicit. Finally, extending the two-sorted coalgebraic representation of Muller automata introduced in [**?**] to hDMAs would yield canonical representative of automata up to language equivalence. *Related work.* Automata over infinite data words have been introduced to prove decidability of satisfiability for many kinds of logic: LTL with freeze quantifier [**?**]; safety fragment of LTL [**?**]; FO with two variables, successor, and equality and order predicates [**?**]; EMSO with two variables, successor and equality [**?**]; generic EMSO [**?**]; EMSO with two variables and LTL with additional operators for data words [**?**]. The main result for these papers is decidability of nonemptiness. These automata are ad-hoc, and often have complex acceptance conditions, while we aim to provide a simple and seamless nominal extension of a well-known class of automata. We can also cite variable finite automata (VFA) [**?**], that recognize patterns specified through ordinary finite automata, with variables on transitions. Their version for infinite words (VBA) relies on Büchi automata. VBA are not closed under complementation and determinism is not a syntactic property. For our automata, determinism is easily checked and we have closure under complementation. On the other hand, VBA can express "global" freshness, i.e. symbols that are different from all the others.

Acknowledgements.. The authors thank Nikos Tzevelekos, Emilio Tuosto and Gianluca Mezzetti for several fruitful discussions related to nominal automata.