Nominal deterministic omega automata*

Vincenzo Ciancia[1] and Matteo Sammartino[2]

[1] ISTI-CNR, Pisa
[2] Dipartimento di Informatica, Università di Pisa, Pisa

Abstract. Nominal sets, presheaf categories, and named sets have successfully served as models of the state space of process calculi featuring resource generation. More recently, automata built in such categories have been studied as acceptors of languages of finite words over infinite alphabets. In this paper we investigate automata whose state spaces are nominal sets, and 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. This is shown using finite representations of the (otherwise infinite-state) defined class of automata. The definition of such operations enables model checking of systems featuring infinite behaviours, and resource allocation, to be implemented using automata-theoretic methods, like in the classical case.

1 Introduction

The behavior of processes has traditionally been represented using models with transition structures, such as labelled transition systems or automata. On the other hand, transition structures, in the form of automata, are also used to represent logic formalisms interpreted over finite and infinite words, dating back to [1,2]. The possibility of translating modal logic formulas to automata led to model-checking (see e.g. [3]). Processes that feature resource allocation (e.g. [4]), 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 [5,6,7,8,9], history-dependent automata [10], that are coalgebras over named sets [11], and automata over nominal sets [12]. Equivalence of these models has been established both at the level of base categories [13,14,15] and of coalgebras [11]. 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., [16,17,18,12]). In particular, the obtained languages are based on infinite alphabets, but still feature some form of finite memory (in fact, the well known *register automata* of Francez and Kaminski can be regarded as nominal automata, see [12]).

* Research partially funded by projects EU ASCENS (nr. 257414), EU QUANTICOL (nr. 600708), IT MIUR CINA and PAR FAS 2007-2013 Regione Toscana TRACE-IT.

The case of infinite words over nominal alphabets is more problematic, as an infinite word over an infinite alphabet is generally not *finitely supported*.[3] 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 to the language infinite memory. More precisely, words without finite support can not be "singled out" by a finite memory machine; if a machine accepts one of those, then it will accept infinitely many others, including finitely supported words.

The aim of this work is to translate the intuitions in the previous paragraphs into precise mathematical terms, in order to define a class of languages 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.

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 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 attempt to use automata to characterize properties of local peers in a global environment. We assume peers have three kinds of observable actions:

[3] The notion of finite support comes from the theory of nominal sets, and will be clarified in the paper.

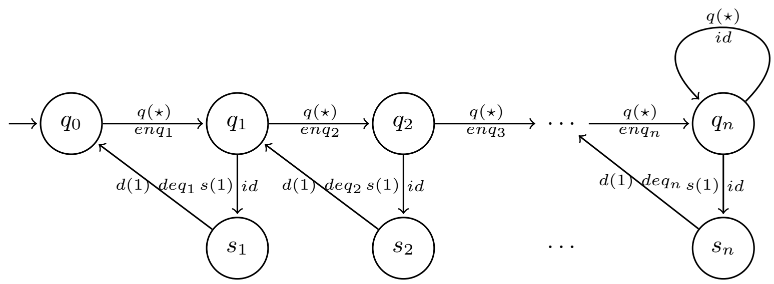

Fig. 1. Automaton for the FCFS policy.

- arrival of a new query from p, denoted $q(p)$;
- selection of a query to serve and connection to its sender p, denoted $s(p)$;
- disconnection from p, denoted $d(p)$.

Automata executing variants of a communication protocol in this setting may have very different behaviors. For example, a system could give preference to a small set of peers, thus making the execution of a global peer-to-peer algorithm less effective. Using automata, we can specify policies (for example, fairness requirements) that characterise desired behaviors. Our main example will be the automata-theoretic 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 want to model query selection in a FCFS style. Moreover, we want to capture fairness: 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.

$$enq_i(x) = \begin{cases} \star & x = i \\ x & x < i \end{cases} \qquad\qquad deq_i(x) = x + 1$$

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 now 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. All these notions will be clearer after we formally introduce nominal Muller automata.

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. For X, Y sets, we let $f \colon X \to Y$ be a total function from X to Y, $f \colon X \rightarrowtail Y$ be a total injective function and $f \colon X \rightharpoonup Y$ a partial function. We write $\mathrm{dom}(f)$ for the subset of X on which f is defined, and $Im(f)$ for the image of f. For f injective, the expression $f^{-1} \colon Y \rightharpoonup X$ denotes the the partial inverse function $\{(y,x) \mid f(x) = y\}$. We let $f|_{X'}$, with $X' \subseteq X$, be the domain restriction of f to X'. (Partial) function composition is written $f \circ g$: it maps x to $f(g(x))$ only if $x \in \mathrm{dom}(g)$; f^n is the n-fold composition of f with itself. We denote the natural numbers with ω. For s a sequence, we let s_i or $s(i)$ denote its i^{th} element, for $i \in \omega$. Given a binary relation R, we denote by R^* its symmetric, transitive and reflexive closure. We say that x and y are *R-related* whenever $(x,y) \in R$. We use $\circ$ also for "relational" composition, as usual, by seeing functions as relations.

We shall now briefly introduce nominal sets; we refer the reader to [19] 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 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*[4] *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, $f(\pi \cdot x) = \pi \cdot f(x)$.*

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

Definition 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 $orb(a) = \mathcal{N}$.

4 Nominal regular ω-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 3. *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 3 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. *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 5. *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 3), 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 6. *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 7. *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.*

Remark 1. 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 1. 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 1), 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 1. *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 [20]), introducing acceptance of infinite words.

Definition 8. *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*;*

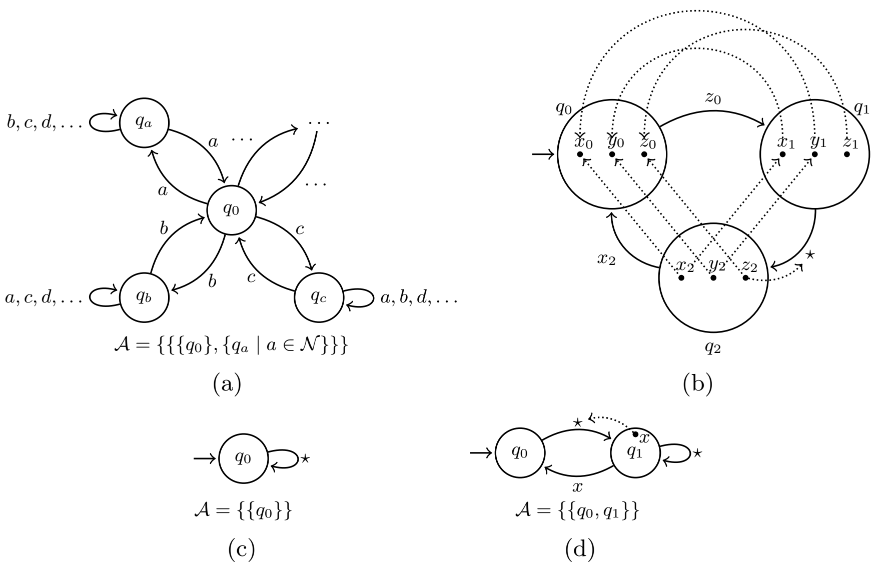

Fig. 2. Some automata, together with their accepting conditions. (a) is the nDMA of Example 1 and (d) is the corresponding hDMA; (b) is an example hDMA where we have omitted the accepting condition and unrelevant transitions, all assumed to end up in a sink state without registers; (c) is the hDMA recognizing $\mathcal{N}^\omega$.

– $\rho_0 : |q_0| \rightarrowtail \mathcal{N}$ *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 $\sigma : |q_2| \rightarrowtail |q_1| \uplus \{l\}$*;*
– *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 2. To keep the notation lightweight, we do not use a *symmetry* attached to states of an hDMA. It is well known (see [10]) 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 [21]. 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 or the unique run 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 9. *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 10. *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 if and only if there is a transition* $q_1 \xrightarrow[\sigma]{l} q_2$ *in* A *and either of the following happens:*

- $l \in |q_1|$, $\rho_1(l) = a$, *and* $\rho_2 = \rho_1 \circ \sigma$;
- $l = \star$, $a \notin Im(\rho_1)$, $\rho_2 = (\rho_1 \circ \sigma)[^a/_{\sigma^{-1}(\star)}]$.

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 1. *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 the configuration graph. Furthermore, we define runs of infinite words.

Definition 11. *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* $(q_i, \rho_i) \xrightarrow{\alpha_i} (q_{i+1}, \rho_{i+1})$.

The following is a simple corollary of Lemma 1.

Proposition 1. *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).

Finally, we define acceptance of hDMAs.

Definition 12. *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, as stated in the following propositions. These are similar to the categorical equivalence results in [13,14]; however, notice that representing infinite branching systems using "allocating transitions" requires further machinery, similar to what is studied in [11]. See also Remark 2 about symmetry.

Proposition 2. *The configuration graph of A, equipped with the permutation action $\pi \cdot (q, \rho) = (q, \pi \circ \rho)$ forms the transition structure of an nDMA. The orbits of the obtained nDMA are in one to one correspondence with states in Q; thus the acceptance condition on states can be used as an acceptance condition on the orbits of the configuration graph. When the configuration (q_0, ρ_0) is chosen as initial state, the obtained nDMA accepts the same language as A.*

Proposition 3. *Fix an nDMA $(Q, \longrightarrow, q_0, \mathcal{A})$. For $q \in Q$, let o_q be a chosen canonical representative of $orb(q)$, $o_{S \subseteq X} = \{o_q \mid q \in S\}$ and ρ_q be a chosen permutation such that $\rho_q \cdot o_q = q$. Construct the hDMA $(o_Q, |-|, \longrightarrow, o_{q_0}, \rho_{q_0}|_{|o_{q_0}|}, \{\{o_q \mid q \in A\} \mid A \in \mathcal{A}\})$, with $|o_q| = supp(o_q)$. For each nDMA transition $o_{q_1} \xrightarrow{a} o_{q_2}$, if $a \in supp(o_{q_1})$, let $o_{q_1} \xrightarrow[\sigma]{a} o_{q_2}$; otherwise, let $o_{q_1} \xrightarrow[\sigma_\star]{\star} o_{q_2}$, where $\sigma = \rho_{q_2}|_{|o_{q_2}|}$ and $\sigma_\star = \sigma_{q_2}[{}^*/_a]|_{|o_{q_2}|}$. The defined hDMA accepts the same language as the original nDMA.*

Example 2. 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 1. 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

The product of finite automata is a well-known operation: in the binary case, it produces an automaton whose states are pairs (q_1, q_2) of states of the original automata and transitions are those both states do. In this section we define a similar 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 telling which registers of q_1 and q_2 contain the same value, representing the same register in the composite state. This is implemented by quotienting registers w.r.t. the equivalence relation

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 [11].

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 13. *$\mathcal{T}_1 \otimes \mathcal{T}_2$ is the transition structure $(Q_\otimes, |-|_\otimes, q_0^\otimes, \rho_0^\otimes, \longrightarrow_\otimes)$ defined as follows:*

- $Q_\otimes := \{(q_1, q_2, R) \mid q_1 \in Q_1, q_2 \in Q_2, R \in Reg(q_1, q_2)\}$;
- $|(q_1, q_2, R)|_\otimes := (|q_1|_1 \cup |q_2|_2)_{/R^*}$, *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 the relation S and the mappings σ_τ, for $\tau \in \{A, R\}$, are as follows

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

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

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 $[l_i]_{R^*} = \{l_1, l_2\}$), 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 3. $\mathcal{T}_1 \otimes \mathcal{T}_2$ 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 14. *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 4. *Given $C \in \mathcal{C}(\mathcal{T}_1 \otimes \mathcal{T}_2)$,*

(i) if $C \xrightarrow{a} C'$ then $\pi_i(C) \xrightarrow{a} \pi_i(C')$, $i = 1, 2$;
(ii) if $\pi_i(C) \xrightarrow{a}_i C_i$, $i = 1, 2$, then there is C' s.t. $C \xrightarrow{a} C'$ and $\pi_i(C) = C_i$.

Corollary 1. *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 2. *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 1. Then $\pi_1(Inf(r)) = Inf(r_1)$ and $\pi_2(Inf(r)) = Inf(r_2)$.*

7 Boolean operations and decidability

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. The crucial tool is Theorem 2: constructing an 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 3. *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 4. *Emptiness, and, as a corollary, equality of languages are decidable.*

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, again, decidability may be an issue otherwise). A calculus that can be handled by the current theory is the determinstic, finite-control pi-calculus. It is nowadays well known that nominal sets correspond to presheaves over finite sets and injections that are *sheaves* with respect to the atomic topology (the so-called *Schanuel topos*), and that HD-automata correspond to coalgebras on such sheaves. By changing the index category of sheaves one obtains different kinds of nominal sets [15], and different classes of HD-automata. Since hDMAs are based on HD-automata, this correspondence seems relevant also for our work. For instance, by taking sheaves over graphs [22], one could express complex relations among symbols in the alphabet, and require that, infinitely often, one encounters a symbol which is related in a certain way to a number of its predecessors. 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 [23]. There may be different logical interpretations of hDMAs, where causality or dependence [24,25] between events are made explicit. Finally, extending the two-sorted coalgebraic representation of Muller automata introduced in [21] 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 [26]; safety fragment of LTL [27]; FO with two variables, successor, and equality and order predicates [28]; EMSO with two variables, successor and equality [29]; generic EMSO [30]; EMSO with two variables and LTL with additional operators for data words [31]. 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) [32], 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.

References

1. Büchi, J.R.: Weak second-order arithmetic and finite automata. Z. Math. Logik Grundl. Math. **6** (1960) 66–92
2. Elgot, C.C.: Decision problems of finite automata design and related arithmetics. Trans. Amer. Math. Soc. (98) (1961) 21–51
3. Clarke, E.M., Schlingloff, B.H.: Model checking. In: Handbook of Automated Reasoning. Elsevier (2001) 1635–1790
4. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I/II. Inf. Comput. **100**(1) (1992) 1–77
5. Fiore, M.P., Turi, D.: Semantics of name and value passing. In: 16th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society (2001) 93–104
6. Bonchi, F., Buscemi, M.G., Ciancia, V., Gadducci, F.: A presheaf environment for the explicit fusion calculus. J. Autom. Reasoning **49**(2) (2012) 161–183
7. Miculan, M.: A categorical model of the fusion calculus. Electr. Notes Theor. Comput. Sci. **218** (2008) 275–293
8. Ghani, N., Yemane, K., Victor, B.: Relationally staged computations in calculi of mobile processes. Electr. Notes Theor. Comput. Sci. **106** (2004) 105–120
9. Montanari, U., Sammartino, M.: A network-conscious π-calculus and its coalgebraic semantics. To appear in Theor. Comput. Sci. (2014)
10. Montanari, U., Pistore, M.: Structured coalgebras and minimal hd-automata for the π-calculus. Theor. Comput. Sci. **340**(3) (2005) 539–576

11. Ciancia, V., Montanari, U.: Symmetries, local names and dynamic (de)-allocation of names. Inf. Comput. **208**(12) (2010) 1349 – 1367
12. Bojanczyk, M., Klin, B., Lasota, S.: Automata with group actions. In: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, IEEE Computer Society (2011) 355–364
13. Gadducci, F., Miculan, M., Montanari, U.: About permutation algebras, (pre)sheaves and named sets. Higher-Order and Symbolic Computation **19**(2-3) (2006) 283–304
14. Fiore, M.P., Staton, S.: Comparing operational models of name-passing process calculi. Inf. Comput. **204**(4) (2006) 524–560
15. Ciancia, V., Kurz, A., Montanari, U.: Families of symmetries as efficient models of resource binding. Electr. Notes Theor. Comput. Sci. **264**(2) (2010) 63–81
16. Tzevelekos, N.: Fresh-register automata. In: Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, ACM (2011) 295–306
17. Kurz, A., Suzuki, T., Tuosto, E.: On nominal regular languages with binders. In: Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Springer (2012) 255–269
18. Gabbay, M.J., Ciancia, V.: Freshness and name-restriction in sets of traces with names. In: FOSSACS. (2011) 365–380
19. Gabbay, M., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Asp. Comput. **13**(3-5) (2002) 341–363
20. Pistore, M.: History Dependent Automata. PhD thesis, University of Pisa (1999)
21. Ciancia, V., Venema, Y.: Stream automata are coalgebras. In: Coalgebraic Methods in Computer Science. Volume 7399 of LNCS. Springer (2012) 90–108
22. Sammartino, M.: A Network-Aware Process Calculus for Global Computing and its Categorical Framework. PhD thesis, University of Pisa (2013)
23. Bojanczyk, M.: Modelling infinite structures with atoms. In Libkin, L., Kohlenbach, U., de Queiroz, R.J.G.B., eds.: WoLLIC. Volume 8071 of Lecture Notes in Computer Science., Springer (2013) 13–28
24. Väänänen, J.A.: Dependence Logic - A New Approach to Independence Friendly Logic. Volume 70 of London Mathematical Society student texts. Cambridge University Press (2007)
25. Galliani, P.: The Dynamics of Imperfect Information. PhD thesis, University of Amsterdam (September 2012)
26. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. ACM Trans. Comput. Log. **10**(3) (2009)
27. Lazic, R.: Safety alternating automata on data words. ACM Trans. Comput. Log. **12**(2) (2011) 10
28. Bojanczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data words. ACM Trans. Comput. Log. **12**(4) (2011) 27
29. Kara, A., Schwentick, T., Tan, T.: Feasible automata for two-variable logic with successor on data words. In Dediu, A.H., Martín-Vide, C., eds.: LATA. Volume 7183 of Lecture Notes in Computer Science., Springer (2012) 351–362
30. Bollig, B.: An automaton over data words that captures EMSO logic. In Katoen, J.P., König, B., eds.: CONCUR. Volume 6901 of Lecture Notes in Computer Science., Springer (2011) 171–186
31. Kara, A., Tan, T.: Extending Büchi automata with constraints on data values. CoRR **abs/1012.5439** (2010)

32. Grumberg, O., Kupferman, O., Sheinvald, S.: Variable automata over infinite alphabets. In Dediu, A.H., Fernau, H., Martín-Vide, C., eds.: LATA. Volume 6031 of Lecture Notes in Computer Science., Springer (2010) 561–572

A Proofs

Proof (of Theorem 1). By properties of nominal sets, for x finitely supported and f equivariant, $f(x)$ is finitely supported with $supp(f(x)) \subseteq supp(x)$. Let $h : Q \to \mathcal{P}(\mathcal{N}^\omega)$ be the function mapping each q to $\mathcal{L}_q$. We need to show that h is equivariant, that is, $h(\pi \cdot q) = \{\pi \cdot \alpha \mid \alpha \in \mathcal{L}_q\}$. Without loss of generality, we shall prove the right-to-left inclusion. Then, since π and q are arbitrary, one can prove the left-to-right inclusion starting from the state $\pi \cdot q$ and the permutation π^{-1}. Let $\alpha \in \mathcal{L}_q$. We shall prove that $\pi \cdot \alpha \in \mathcal{L}_{\pi \cdot q}$. Consider the unique (accepting) run r of α from q, and the unique run r' of $\pi \cdot \alpha$ from $\pi \cdot q$. By equivariance of the transition function, and definition of run, for all i, we have $r'_i = \pi \cdot r_i$, thus $orb(r'_i) = orb(r_i)$, therefore $Inf(r') = Inf(r)$. □

Proof (of Lemma 1). For each a, if $a \in Im(\rho_1)$, recalling that ρ_1 is injective, there is $l \in |q_1|$ with $\rho_1(l) = a$. By definition of hDMA, there is exactly one transition labelled with l, let it be $q_1 \xrightarrow[\sigma]{l} q_2$. Then by definition of configuration graph, we have $(q_1, \rho_1) \xrightarrow{a} (q_2, \rho_1 \circ \sigma)$. Since ρ_1 is injective, there can not be other transitions labelled with a in the configuration graph. If $a \notin Im(\rho_1)$, consider the only transition with label $\star$ from q_1, namely $q_1 \xrightarrow[\sigma]{\star} q_2$. Then we have $(q_1, \rho_1) \xrightarrow{a} (q_2, (\rho_1 \circ \sigma)[a/\sigma^{-1}(\star)])$ in the configuration graph; this transition is unique by definition.

Proof (of Proposition 2). A run r in the configuration graph clearly is also a run in the obtained automaton. As $orb(q, \rho) = \{(q, \rho') \mid \rho' : |q| \rightarrowtail \mathcal{N}\}$, also acceptance is the same on both sides. By Lemma 1 we get determinism. The proof is completed by noting that the obtained transition function is equivariant. For this, chose an edge $(q_1, \rho_1) \xrightarrow{a} (q_2, \rho_2)$ in the configuration graph, and look at Definition 10, thus consider a corresponding hDMA transition $q_1 \xrightarrow[\sigma]{l} q_2$. The case when $l \in |q_1|$ is straightforward. When $l = \star$, thus $(q_1, \rho_1) \xrightarrow{a} (q_2, (\rho_1 \circ \sigma)[a/\sigma^{-1}(\star)])$ consider the permuted configuration $(q_1, \pi \circ \rho_1)$, for any permutation π. Since $a \notin Im(\rho_1)$, also $\pi(a) \notin Im(\pi \circ \rho_1)$, thus we have a transition $(q_1, \pi \circ \rho_1) \xrightarrow{\pi(a)} (q_2, (\pi \circ \rho_1 \circ \sigma)[\pi(a)/\sigma^{-1}(*)])$, which is precisely the required permuted transition. □

Proof (of Proposition 3). The proof is similar to the equivalence results between categories of coalgebras given in [11]. First, we need to show that, for each transition $q_1 \xrightarrow{a} q_2$ in the original nDMA, there is an edge $(o_{q_1}, \rho_{q_1}|_{|q_1|}) \xrightarrow{a} (o_{q_2}, \rho_{q_2}|_{|q_2|})$ in the configuration graph of the derived hDMA. We look at the case $a \in supp(q_1)$; the case with allocation is similar, even though technically more involved. By equivariance, from $q_1 \xrightarrow{a} q_2$, we have $o_{q_1} \xrightarrow{\rho_{q_1}^{-1}(a)} \rho_{q_1}^{-1}(q_2)$. Then we have an hDMA transition $o_{q_1} \xrightarrow[\sigma]{\rho_{q_1}^{-1}(a)} o_{q_2}$ where $\sigma = \rho_{\rho_{q_1}^{-1}(q_2)}\Big|_{|o_2|}$. By looking at the used permutations, we have $\sigma = \rho_{q_1}^{-1} \circ \rho_{q_2}\big|_{|o_2|}$. Then, in the configuration graph, we have $(o_{q_1}, id) \xrightarrow{\rho_{q_1}^{-1}(a)} (o_{q_2}, \sigma)$, thus by equivariance, we have $(o_{q_1}, \rho_{q_1}\big|_{|q_1|}) \xrightarrow{a} (o_{q_2}, \rho_{q_1} \circ \rho_{q_1}^{-1} \circ \rho_{q_2}\big|_{|q_2|})$, thus $(o_{q_1}, \rho_{q_1}\big|_{|q_1|}) \xrightarrow{a} (o_{q_2}, \rho_{q_2}\big|_{q_2})$. Accordance of the accepting conditions is straightforward. ☐

Proof (of Proposition 4). Let $C = ((q_1, q_2, R), \rho)$ and $\pi_i(C) = (q_i, \rho_i)$, $i = 1, 2$.

Part (i). Let $C' = ((q_1', q_2', R'), \rho')$ and let

$$(q_1, q_2, R) \xrightarrow[\sigma]{l}_{\otimes} (q_1', q_2', S)$$

be the transition inducing $C \xrightarrow{a} C'$. We proceed by cases on the rule used to infer this transition:

– (Reg): then the transition is inferred from $q_i \xrightarrow[\sigma_i]{l_i}_i q_i'$, $i = 1, 2$, such that either l_1 or l_2 is in $\mathcal{N}$. Suppose, w.l.o.g., $l_1 \in \mathcal{N}$. Then $l = [l_1]_{R^*}$ and $\rho_i(l_1) = \rho([l_1]_{R^*}) = a$, so there is an edge $(q_1, \rho_1) \xrightarrow{a}_1 (q_1', \rho_1')$ in the configuration graph of $\mathcal{T}_1$. The following chain of equations shows that $\pi_1(C') = (q_1', \rho_1')$:

$$\begin{aligned} \rho_1'(x) &= \rho_1(\sigma_1(x)) \\ &= \rho([\sigma_1(x)]_{R^*}) \\ &= \rho(\sigma_r([x]_{S^*})) \\ &= \rho'([x]_{S^*}) \end{aligned} \tag{†}$$

To prove the existence of an edge $(q_2, \rho_2) \xrightarrow{a}_2 (q_2', \rho_2')$ in the configuration graph of $\mathcal{T}_2$, we have to consider the following two cases:
- If $l_2 \in \mathcal{N}$, then $\rho_2(l_2) = \rho([l_2]_{R^*}) = \rho([l_1]_{R^*}) = a$, by the rule premise $[l_2]_{R^*} = \{l_1, l_2\}$;
- If $l_2 = \star$, then a should be fresh, so we have to check $a \notin Im(\rho_2)$. Suppose, by contradiction, that there is $x \in |q_2|_2$ such that $\rho_2(x) = a$, then $\rho([x]_{R^*}) = a = \rho([l_1]_{R^*})$, by definition of ρ, which implies $[x]_{R^*} = [l_1]_{R^*}$, by injectivity of ρ, i.e. $\{l_1, x\} \in [l_1]_{R^*}$, but the premise of the rule states $[l_1]_{R^*} = \{l_1, \star\} \cap \mathcal{N} = \{l_1\}$, so we have a contradiction.

Now we have to check $\pi_2(C') = (q_2', \rho_2')$. Since we have $\rho_2'(x) = (\rho_2 \circ \sigma_2)[^a/_{\sigma_2^{-1}(\star)}](x)$, for $x \neq \sigma_2^{-1}(\star)$ the equations (†) hold. For $x = \sigma_2^{-1}(\star)$ we have:

$$\begin{aligned} \rho_2'(x) &= (\rho_2 \circ \sigma_2)[^a/_x](x) \\ &= a \\ &= \rho([l_1]_{R^*}) \\ &= (\rho \circ \sigma_r)([x]_{S^*}) \\ &= \rho'([x]_{S^*}) \end{aligned}$$

– (Alloc): then we have $l = \star$ and the transition is inferred from $q_i \xrightarrow[\sigma_i]{\star}_i q_i'$, $i = 1, 2$. Since $a \notin Im(\rho)$, we also have $a \notin Im(\rho_i)$, so there are $(q_i, \rho_i) \xrightarrow{a}_i (q_i', \rho_i')$ with $\rho_i' = (\rho_i \circ \sigma_i)[{}^a/_{\sigma_i^{-1}(\star)}]$, for $i = 1, 2$. Finally, we have to check that each $\rho_i'(x)$ is as required: if $x \neq \sigma_i^{-1}(\star)$ equations (†) hold; for $x = \sigma_i^{-1}(\star)$ we have

$$\begin{aligned}
\rho_i'(x) &= (\rho_i \circ \sigma_i)[{}^a/_x](x) \\
&= a \\
&= (\rho \circ \sigma_a)[{}^a/_{\sigma_a^{-1}(\star)}](\sigma_a^{-1}(\star)) \\
&= (\rho \circ \sigma_a)[{}^a/_{[x]_{S^*}}]([x]_{S^*}) \\
&= \rho'([x]_{S^*})
\end{aligned}$$

Part (ii). Since $\mathcal{T}_1 \otimes \mathcal{T}_2$ is deterministic, there certainly is $C \xrightarrow{a} C'$, for any $a \in \mathcal{N}$. This edge, by the previous part of the proof, has a corresponding edge $\pi_i(C) \xrightarrow{a}_i \pi_i(C')$, for each $i = 1, 2$. But then $\pi_i(C') = C_i$, by determinism of $\mathcal{T}_i$.

□

Proof (of Theorem 3). We just consider $\mathcal{L}_1 \cap \mathcal{L}_2$, the other cases are analogous. Let $A_\cap$ be $(\mathcal{T}_1 \otimes \mathcal{T}_2, \mathcal{A}_\cap)$; this is a proper hDMA, thanks to Remark 3. Given $\alpha \in \mathcal{N}^\omega$, let $r_\cap$,r_1 and r_2 be the runs for α in the configuration graphs of $A_\cap, A_1$ and A_2, respectively. Then, by Theorem 2, we have $\pi_i(Inf(r_\cap)) = Inf(r_i)$, for each $i = 1, 2$. From this, and the definition of $\mathcal{A}_\cap$, we have that $Inf(r_\cap) \in \mathcal{A}_\cap$ if and only if $Inf(r_1) \in \mathcal{A}_1$ and $Inf(r_2) \in \mathcal{A}_2$, i.e. $\alpha \in \mathcal{L}_{A_\cap}$ if and only if $\alpha \in \mathcal{L}_{A_1}$ and $\alpha \in \mathcal{L}_{A_2}$. □

Proof (of Theorem 4). Let $A = (Q, | - |, q_0, \rho_0, \longrightarrow, \mathcal{A})$ be a hDMA for $\mathcal{L}$. Consider the set $\Sigma_A = \{(l, \sigma) \mid \exists q, q' \in Q : q \xrightarrow[\sigma]{l} q'\}$. This is finite, so we can use it as the alphabet of an ordinary deterministic Muller automaton $M_A = (Q \cup \{\delta\}, q_0, \longrightarrow_s, \mathcal{A})$, where $\delta \notin Q$ is a dummy state, and the transition function is defined as follows: $q \xrightarrow{(l,\sigma)}_s q'$ if and only if $q \xrightarrow[\sigma]{l} q'$, and $q \xrightarrow{(l,\sigma)}_s \delta$ for all other pairs $(l, \sigma) \in \Sigma_A$. Clearly $\mathcal{L}_{M_A} = \varnothing$ if and only if $\mathcal{L} = \varnothing$, as words in $\mathcal{L}_{M_A}$ are sequence of transitions of A that go through accepting states infinitely often, and thus produce a word in $\mathcal{L}$, and vice versa. The claim follows by decidability of emptiness for ordinary deterministic Muller automata. Finally, to check equality of languages, observe that the language $(\mathcal{L}_1 \cup \mathcal{L}_2) \setminus (\mathcal{L}_1 \cap \mathcal{L}_2)$ is ω-regular nominal, thanks to Theorem 3. Then we just have to check its emptiness, which is decidable. □