Nominal Automata and Register Automata with Permutations

Mrudula Balachander[1], Emmanuel Filiot[2], Raffaella Gentilini[3] and Nikos Tzevelekos[4]

[1]University of Birmingham, UK
[2]Université Libre de Bruxelles, Belgium
[3]University of Perugia, Italy
[4]Queen Mary University of London, UK

Abstract

We consider two deterministic automata models over infinite alphabets: nominal automata and register automata with permutations (pDRA). We prove that a data language is recognizable by a nominal automaton with n orbits and dimension k if and only if it is recognizable by a pDRA having n states and k registers. The equivalence builds a bridge between abstract algebraic models and concrete algorithmic implementations in data language theory, ultimately allowing the use of pDRAs to efficiently write down nominal automata. It also enables the transfer of algorithmic complexity bounds such as memorability and language equivalence from pDRAs into nominal automata.

1. Introduction

Automata over infinite alphabets have been studied extensively[1, 2, 3]. Among the proposed models, *register automata*, introduced as *finite-memory automata* by Kaminski and Francez [4], are perhaps the most widely studied. They model systems that process data words using only finitely many remembered data values: a register automaton over an infinite data domain $\mathcal{D}$ is a finite-state automaton equipped with finitely many registers, whose contents can be compared with incoming data and updated along transitions. Deterministic register automata (DRAs) enjoy closure under complement and a decidable equivalence problem; the languages they recognise are the *regular data languages*.

Two robust theories of regular data languages have emerged, of rather different flavours. On the foundations side, *nominal automata* [5, 6] recast finite-state automata inside the universe of *nominal sets* [7], generalising finiteness of the state space to *orbit-finiteness*; this yields a clean Myhill–Nerode characterization and neat generalisations of classical constructions, such as automata learning [8]. On the concrete side, register automata give an operational, implementable description of the same languages, and the two are equi-expressive [9]. Nonetheless, the gap between them is more than cosmetic: the inner workings of nominal automata rely on nominal-set reasoning (e.g. equivalence under data permutation), which adds a level of indirection, and direct register-automata algorithms tend to be faster in theory and practice [10, 11, 12]. Conversely, as observed in [6, Sec. 6.2], extending finite automata with registers implicitly imposes a *linear order* on the stored data, which is incompatible with minimality in the sense of Myhill–Nerode.

Register automata with permutations (pDRAs) [13] reconcile the two views: a pDRA is a deterministic register automaton in which every transition also applies a *permutation* of the registers. This removes the linear-order obstruction — pDRAs admit canonical, minimal representatives based on a finite-index word equivalence — while retaining a polynomial-time equivalence test from their permutation-free fragment [10]. Our main result (Theorem 14) makes the bridge precise: a data language L is recognised by a deterministic nominal automaton with n orbits and dimension k if and only if it is recognised by a pDRA with n states and k registers, with minimal models corresponding on both sides. Consequently, pDRAs are a finite, concrete syntax for (minimal) nominal automata — orbits become states, the dimension becomes the number of registers — so a nominal automaton can be *written down* as a finite

ICTCS 2026: Italian Conference on Theoretical Computer Science, September 7–9, 2026, Udine, Italy

labelled graph (Proposition 15) and algorithmic results transfer from pDRAs to nominal automata, notably polynomial-time language equivalence and memorability, as does succinctness. The connection between pDRAs and nominal automata was suggested by an anonymous reviewer of [13], where pDRAs were introduced; here we isolate and develop it. Proofs of our results are in the appendix.

2. Preliminaries

For $i, j \in \mathbb{N}$, $[i, j] = \{k \mid i \leq k \leq j\}$ and $\mathcal{P}(X)$ is the powerset of X. For a relation R we write $\mathsf{dom}(R), \mathsf{rng}(R)$ for its domain and range, $R|_{X'}$ for its restriction, $R_1; R_2$ for composition, and $R[i \mapsto j] = \{(i, j)\} \cup (R \setminus (\{i\} \times Y))$ for update. We fix an infinite alphabet $\mathcal{D}$ of *data*, ranged over by $d, a, b, \ldots$; a *data word* is $w \in \mathcal{D}^*$, with ϵ the empty word, $w[i]$ its i-th datum and $\hat{w} : [1, |w|] \to \mathcal{D}$ its functional view. A *data permutation* is a bijection $\tau : \mathcal{D} \to \mathcal{D}$ fixing all but finitely many data; $\mathrm{Perm}(\mathcal{D})$ is the set of data permutations and $(d\ d')$ the transposition of d, d'.

A *nominal set* [7] is a set X with a group action $\cdot : \mathrm{Perm}(\mathcal{D}) \times X \to X$. A set $S \subseteq \mathcal{D}$ *supports* $x \in X$ if $\tau \cdot x = x$ for every τ fixing S pointwise; we require every element to have finite support and write $\mathsf{supp}(x)$ for its least one, calling x *equivariant* if $\mathsf{supp}(x) = \varnothing$. Supports are closed under intersection, so $\mathsf{supp}(x)$ is well defined. Then $\mathcal{D}^*$ is a nominal set under $\tau \cdot w = \hat{w}; \tau$, with $\mathsf{supp}(w) = \{w[i] \mid i \in [1, |w|]\}$, and $L \subseteq \mathcal{D}^*$ is a *data language* if it is equivariant ($\tau \cdot L = L$ for all τ). Finally, a *register assignment* is a partial injection $\rho : [1, r] \rightharpoonup \mathcal{D}$; $\mathcal{S}_n$ is the group of permutations of $[1, n]$, id the identity, $(i\ j)$ a transposition; the *canonical extension* $\hat{\tau} \in \mathcal{S}_n$ of a partial permutation τ on $[1, n]$ closes its maximal paths and is the identity elsewhere.

3. Automata over Infinite Alphabets

We recall the two deterministic models over infinite alphabets that we relate: register automata with permutations, which are concrete and based on memory; and nominal automata, which use elements of nominal set theory.

Register automata with permutations. We recall here the relevant definitions from [13].

Definition 1. A *deterministic permutation register automaton (pDRA)* is a tuple $\mathcal{A} = (r, Q, \mu, q_0, F, \delta)$ where $[1, r]$ is a set of *registers* ($r \geq 0$); Q is a finite set of *states* with *initial* state q_0 and *final* states $F \subseteq Q$; $\mu : Q \to \mathcal{P}([1, r])$ is an *availability function* with $\mu(q_0) = \varnothing$; and $\delta = (\delta_= \cup \delta_\neq)$ is a pair of partial *transition functions*:

$$\delta_= : Q \times [1, r] \rightharpoonup Q \times \mathcal{S}_r \quad \text{and} \quad \delta_\neq : Q \rightharpoonup Q \times [1, r+1] \times \mathcal{S}_r\,.$$

We write $p \xrightarrow{k,\pi} q$ for $\delta_=(p, k) = (q, \pi)$ and $p \xrightarrow{k^\bullet,\pi} q$ for $\delta_\neq(p) = (q, k, \pi)$. Transitions obey the *availability condition*: if $p \xrightarrow{x,\pi} q$ with $x \in [1, r]$ then $x \in \mu(p)$ and $\mu(q) \subseteq \pi(\mu(p))$; and if $x = k^\bullet$ with $k \in [1, r+1]$ then $\mu(q) \subseteq \pi(\mu(p) \cup \{k\})$. $\mathcal{A}$ is *complete* if $\delta_\neq$ is total and $\mathsf{dom}(\delta_=(p, \cdot)) = \mu(p)$ for all p.

Thus, each state p carries a set $\mu(p) \subseteq [1, r]$ of *available* registers. While examining an input word, the automaton keeps a register assignment $\rho : \mu(p) \to \mathcal{D}$. On datum d: if $d = \rho(x)$ for available x and $\delta_=(p, x) = (q, \pi)$, it moves to q, permutes registers by π and deletes the contents outside $\mu(q)$; if d is *locally fresh* ($d \notin \mathsf{rng}(\rho)$) and $\delta_\neq(p) = (q, k, \pi)$, it stores d in register k (a dummy when $k = r + 1$, so d is discarded), permutes by π and deletes unavailable contents. Formally, a *configuration* is (p, ρ) with $\rho : \mu(p) \to \mathcal{D}$ injective; $\mathbb{C}_\mathcal{A}$ is the set of configurations and $\to_\mathcal{A}$ has $(p, \rho_1) \xrightarrow{d}_\mathcal{A} (q, \rho_2)$ whenever:

- $\rho_1(k) = d$, $\delta_=(p, k) = (q, \pi)$ and $\rho_2 = (\pi^{-1}; \rho_1)|_{\mu(q)}$ (*Equality*),
- or $d \notin \mathsf{rng}(\rho_1)$, $\delta_\neq(p) = (q, k, \pi)$ and $\rho_2 = (\pi^{-1}; \rho_1[k \mapsto d])|_{\mu(q)}$ (*Disequality*).

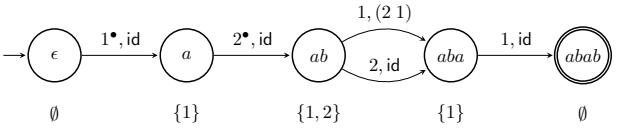

Figure 1: A pDRA recognising $L = \{abab\} \cup \{abba\}$ (Example 3); available registers are shown below each state and the rejecting sink $[aa]$ is omitted. It uses two registers and the transposition $(2\ 1)$, and is in fact the minimal pDRA for L, with six states.

A *run* of $w = d_1 \cdots d_n$ starts from $\kappa_0 = (q_0, \varnothing)$; determinism gives at most one transition per datum. Writing $(q,\rho) \xrightarrow{w}_{\mathcal{A}} (q',\rho')$ for such a run, $\mathcal{L}(q,\rho) = \{w \mid (q,\rho) \xrightarrow{w}_{\mathcal{A}} (q',\rho'),\ q' \in F\}$ and $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\kappa_0)$. A *regular data language* is one recognised by a pDRA, and the *residual* of L by u is $u^{-1}L = \{w \mid uw \in L\}$ (which need not be equivariant, but $\mathsf{supp}(u^{-1}L) \subseteq \mathsf{supp}(u)$).

The permutation-free pDRAs — using only id — are exactly the MRT-automata of [10], which are equi-expressive with register automata [4]. Permutations add no expressive power, but do add succinctness: e.g. for $L_n = \{d_1 \cdots d_n d_{\pi(1)} \cdots d_{\pi(n)} \mid d_i \text{ pairwise distinct},\ \pi \in \mathcal{S}_n\}$ a pDRA needs $2n+1$ states whereas any MRT-automaton needs exponentially many, to hard-code the order in which data are read back.

Lemma 2 ([13, Lemma 5])**.** *pDRAs are exponentially more succinct than MRT-automata.*

Example 3. Let $L = \{abab \mid a \neq b \in \mathcal{D}\} \cup \{abba \mid a \neq b \in \mathcal{D}\}$; a pDRA for L is shown in Figure 1. After storing distinct a, b in registers $1, 2$ (state ab), it accepts ab (test registers 1 then 2) or ba (test 2 then 1). To merge both continuations into one state aba, the transition reading the third datum on register 1 applies $(2\ 1)$, so that the datum still to be read is always in register 1 — exactly what a permutation-free model cannot do without splitting the state.

A pDRA also has a semantic reading: its configuration graph is a nominal automaton whose state space is a *strong nominal set* [14, 9]. We now recall nominal automata, and in Section 4 make this connection exact.

Nominal automata. Let X be a nominal set and $x \in X$. The *orbit* of x is $\mathsf{orb}(x) = \{\tau \cdot x \mid \tau \in \mathrm{Perm}(\mathcal{D})\}$; orbits are equal or disjoint, so $\mathsf{orbits}(X) = \{\mathsf{orb}(x) \mid x \in X\}$ partitions X. X is *orbit-finite* if $\mathsf{orbits}(X)$ is finite; its *size* $N(X)$ is the number of orbits and its *dimension* is $\dim(X) = \max\{|\mathsf{supp}(x)| \mid x \in X\}$.

Definition 4. A *(deterministic) nominal automaton* over $\mathcal{D}$ is a tuple $\mathcal{N} = (Q, q_0, F, \delta)$ with Q an orbit-finite nominal set of states, $q_0 \in Q$ and $F \subseteq Q$ equivariant, and $\delta : (Q \times \mathcal{D}) \to Q$ equivariant, i.e. $\delta(\tau \cdot q, \tau(d)) = \tau \cdot \delta(q,d)$ and $\mathsf{supp}(\delta(q,d)) \subseteq \mathsf{supp}(q,d)$. Its *dimension* is that of Q, and it *has* n *orbits* if Q has n orbits.

For $L \subseteq \mathcal{D}^*$, the *Myhill–Nerode* relation $\equiv_L^{\mathsf{MN}}$ is given by $u \equiv_L^{\mathsf{MN}} v$ iff $uw =_L vw$ for all w (where $u =_L v$ means $u \in L \Leftrightarrow v \in L$). It is known that L is recognised by a nominal automaton iff $\mathcal{D}^*/\equiv_L^{\mathsf{MN}}$ is orbit-finite, and then the minimal such automaton has state space $\mathcal{D}^*/\equiv_L^{\mathsf{MN}}$ [5, 6].

4. A Correspondence Between the Two Models

We connect the two models through *memorable data* and a word equivalence $\equiv_L$ characterising pDRA-recognisability. Throughout, $L \subseteq \mathcal{D}^*$ is a data language.

The *memorable data* of a word with respect to L capture what any automaton accepting L must keep in memory after reading said word [15]: d is *L-memorable* in u if $d \in \mathsf{supp}(u^{-1}L)$, and one lets $\mathsf{mem}_L(u) = \mathsf{supp}(u^{-1}L)$. A pDRA is *frugal* if $\mathsf{rng}(\rho) = \mathsf{supp}(\mathcal{L}(q,\rho))$ for every configuration, and *minimal* if it is complete, frugal and *state-minimal*.

Lemma 5 ([13, Lemma 9]). *Let d' be fresh in u. Then d is L-memorable in u iff $u^{-1}L \neq ((d\ d') \cdot u)^{-1}L$. Hence if a pDRA recognising L reaches (q, ρ) on reading u, then $\mathsf{mem}_L(u) \subseteq \mathsf{rng}(\rho)$.*

Definition 6. Words u, v are *L-equivalent*, written $u \equiv_L v$, if $u^{-1}L = (\tau \cdot v)^{-1}L$ for some data permutation τ; we write $u \equiv_L^\tau v$ to record τ (this is not an equivalence relation in general).

Proposition 7 ([13, Proposition 13 and Lemma 14]). *$\equiv_L$ is an equivalence relation, and if $u \equiv_L^\tau v$ then $\mathsf{mem}_L(u) = \tau \cdot \mathsf{mem}_L(v)$; in particular $|\mathsf{mem}_L(u)| = |\mathsf{mem}_L(v)|$.*

Example 8. For $L = \{abab\} \cup \{abba\}$ from Example 3, $\equiv_L$ has six classes, $[\epsilon]$, $[a]$, $[aa]$, $[ab]$, $[aba]$, $[abab]$, where $[aa]$ collects every non-empty word that is not a prefix of a word in L (e.g. $aba \equiv_L^\tau abb$ for τ swapping a, b, since both residuals equal $\{b\}$, while $abab \equiv_L^{\mathsf{id}} abba$). The memorable data are $\mathsf{mem}_L(\epsilon) = \mathsf{mem}_L(abab) = \varnothing$, $\mathsf{mem}_L(a) = \{a\}$, $\mathsf{mem}_L(ab) = \{a, b\}$, $\mathsf{mem}_L(aba) = \{b\}$, so the maximal memory is 2, matching the two registers of Figure 1.

Theorem 9 ([13, Theorem 16]). *L is recognisable by a pDRA iff $\equiv_L$ has finite index.*

The right-to-left direction is witnessed by a *canonical pDRA* $\mathcal{A}_L$ [13, Section 4]: its states are the $\equiv_L$-classes, each equipped with an assignment of its memorable data, and its transitions permute registers so that residual languages match up to $\equiv_L$. This automaton is minimal and exhibits the exact resources used.

Lemma 10 ([13, Lemma 18]). *$\mathcal{A}_L$ recognises L and is minimal, with $|\mathcal{D}^*/\equiv_L|$ states and $\max_u |\mathsf{mem}_L(u)|$ registers; moreover any pDRA recognising L has at least as many of each.*

We now relate $\equiv_L$ to $\equiv_L^{\mathsf{MN}}$, and mem_L to least supports. The key point is that $\equiv_L$ is $\equiv_L^{\mathsf{MN}}$ taken up to a data permutation. We start with the following lemma.

Lemma 11. *For all $u \in \mathcal{D}^*$ and data permutations τ, $\tau \cdot [u]_{\equiv_L^{\mathsf{MN}}} = [\tau \cdot u]_{\equiv_L^{\mathsf{MN}}}$.*

Proof. By nominal-set reasoning, using that L is equivariant. □

Lemma 12. *For $u, v \in \mathcal{D}^*$, $u \equiv_L v$ iff $u \equiv_L^{\mathsf{MN}} \tau \cdot v$ for some τ; equivalently, iff $[u]_{\equiv_L^{\mathsf{MN}}}$ and $[v]_{\equiv_L^{\mathsf{MN}}}$ lie in the same orbit.*

Thus the orbits of $\mathcal{D}^*/\equiv_L^{\mathsf{MN}}$ are in bijection with the $\equiv_L$-classes. The memories match too:

Lemma 13. *For all $u \in \mathcal{D}^*$, $\mathsf{supp}([u]_{\equiv_L^{\mathsf{MN}}}) = \mathsf{mem}_L(u)$.*

Combining these with the canonical pDRA of Lemma 10 and the nominal Myhill–Nerode theorem yields the correspondence.

Theorem 14. *Given $L \subseteq \mathcal{D}^*$, the following are equivalent:*

1. *$\mathcal{D}^*/\equiv_L^{\mathsf{MN}}$ has n orbits and nominal dimension k;*
2. *$\mathcal{D}^*/\equiv_L$ has size n and $k = \sup\{|\mathsf{mem}_L(v)| \mid v \in \mathcal{D}^*\}$;*
3. *L is recognised by a deterministic nominal automaton with n orbits and dimension k; moreover every deterministic nominal automaton recognising L has $n' \geq n$ orbits and dimension $k' \geq k$;*
4. *L is recognised by a deterministic pDRA with n states and k registers; moreover every deterministic pDRA recognising L has $n' \geq n$ states and $k' \geq k$ registers.*

Proof. By Lemmas 11 and 12, $\mathcal{D}^*/\equiv_L^{\mathsf{MN}}$ has n orbits iff $\mathcal{D}^*/\equiv_L$ has size n, while Lemma 13 ensures that the nominal dimension of $\mathcal{D}^*/\equiv_L^{\mathsf{MN}}$ is $k = \sup\{|mem_L(v)| \mid v \in \mathcal{D}^*\}$. Therefore, the first two statements are equivalent. The equivalence between the first and the third statement was proven in [6]. Finally, Lemma 10 yields the equivalence between the second statement and the last one. □

Hence the minimal nominal automaton and the minimal pDRA for L carry the same data: orbits correspond to states, and the dimension to the number of registers. For the running example, $\equiv_L$ has six classes and maximal memory 2, so the minimal pDRA of Figure 1 has six states and two registers, and Theorem 14 shows the minimal nominal automaton for L has six orbits and dimension two.

The correspondence is effective in both directions. From a pDRA one gets a nominal automaton by casting its configuration graph as one; the reverse turns a nominal automaton into a concrete pDRA. Call a nominal set X *strong* [14] if $\tau \cdot x = x \iff \forall d \in \mathsf{supp}(x).\, \tau(d) = d$; not all nominal sets are strong (e.g. $\{\{a, b\} \mid a \neq b\}$). Intuitively, the construction below collapses each orbit of states of nominal automaton $\mathcal{N}$ into a single state of a language-equivalent pDRA $\mathcal{A}_\mathcal{N}$ whose registers store the support of a chosen representative, and reads the transitions of $\mathcal{A}_\mathcal{N}$ off those of $\mathcal{N}$; the two are then bisimilar, and even isomorphic when Q is strong.

Proposition 15. *Given a nominal automaton $\mathcal{N} = (Q, q_0, F, \delta)$ of dimension r, one can construct a pDRA $\mathcal{A}_\mathcal{N} = (r, Q', \mu, q_0', F', \delta')$ such that*

- $Q' = \mathsf{orbits}(Q)$, $q_0' = \mathsf{orb}(q_0) = \{q_0\}$, $F' = \mathsf{orbits}(F)$;
- *for each $q \in Q$, $\mu(\mathsf{orb}(q)) = [1, |\mathsf{supp}(q)|]$;*
- *there is a relation $R \subseteq Q \times \mathbb{C}_{\mathcal{A}_\mathcal{N}}$ such that $(q_0, (q_0', \varnothing)) \in R$ and, for each $(q, (c, \rho)) \in R$, $c = \mathsf{orb}(q)$ and $\mathsf{rng}(\rho) = \mathsf{supp}(q)$, and R is a bisimulation (with respect to $\mathcal{N}$ and the transition graph of $\mathcal{A}_\mathcal{N}$).*

Moreover, if Q is strong, then $\mathcal{N}$ and the configuration graph of $\mathcal{A}_\mathcal{N}$ are isomorphic.

Proposition 15 lets one *write down* a nominal automaton as a finite labelled graph with permutations on its edges, of size its number of orbits and using as many registers as its dimension; for minimal strong-support automata this is faithful up to isomorphism. Conversely, algorithmic results for pDRAs transfer to nominal automata: language equivalence and bisimilarity of pDRAs are decidable in polynomial time [13, Theorem 8], as is testing memorability of a datum in a word [13, Theorem 11], so the same bounds hold for nominal automata once presented as pDRAs — without explicit nominal-set machinery. Finally, since pDRAs are exponentially more succinct than MRT-automata, i.e. register automata (Lemma 2), so are nominal automata in their number of orbits. The same correspondence should help transfer further pDRA results, such as active learning [13, Section 6].

Declaration on Generative AI

During the preparation of this work, the authors used Claude (Opus 4.8) for: Drafting content. Furthermore, the authors used GPT-5.5 for: Peer review simulation, Paraphrase and reword, Formatting assistance. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] L. D'Antoni, In the maze of data languages, CoRR abs/1208.5980 (2012). URL: http://arxiv.org/abs/1208.5980. arXiv:1208.5980.

[2] L. Segoufin, Automata and logics for words and trees over an infinite alphabet, in: Z. Ésik (Ed.), Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings, volume 4207 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 41–57. doi:10.1007/11874683_3.

[3] T. Schwentick, Automata for XML - A survey, J. Comput. Syst. Sci. 73 (2007) 289–315. doi:10.1016/J.JCSS.2006.10.003.

[4] M. Kaminski, N. Francez, Finite-memory automata, Theoretical Computer Science 134 (1994) 329–363. doi:https://doi.org/10.1016/0304-3975(94)90242-9.

[5] M. Bojańczyk, B. Klin, S. Lasota, Automata with group actions, in: Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada, IEEE Computer Society, 2011, pp. 355–364. doi:10.1109/LICS.2011.48.

[6] M. Bojańczyk, B. Klin, S. Lasota, Automata theory in nominal sets, Log. Methods Comput. Sci. 10 (2014). doi:10.2168/LMCS-10(3:4)2014.

[7] A. M. Pitts, Nominal Sets: Names and Symmetry in Computer Science, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2013. doi:10.1017/CBO9781139084673.

[8] J. Moerman, M. Sammartino, A. Silva, B. Klin, M. Szynwelski, Learning nominal automata, in: G. Castagna, A. D. Gordon (Eds.), Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017, ACM, 2017, pp. 613–625. doi:10.1145/3009837.3009879.

[9] M. Bojańczyk, Slightly infinite sets, 2019. URL: https://www.mimuw.edu.pl/~bojan/paper/atom-book, draft version.

[10] A. S. Murawski, S. J. Ramsay, N. Tzevelekos, Polynomial-Time Equivalence Testing for Deterministic Fresh-Register Automata, in: I. Potapov, P. Spirakis, J. Worrell (Eds.), 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018), volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2018, pp. 72:1–72:14. doi:10.4230/LIPIcs.MFCS.2018.72.

[11] A. S. Murawski, S. J. Ramsay, N. Tzevelekos, DEQ: equivalence checker for deterministic register automata, in: Y. Chen, C. Cheng, J. Esparza (Eds.), Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings, volume 11781 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 350–356. doi:10.1007/978-3-030-31784-3_20.

[12] M. H. Bandukara, N. Tzevelekos, On-the-fly bisimulation equivalence checking for fresh-register automata, J. Syst. Archit. 145 (2023) 103010. doi:10.1016/J.SYSARC.2023.103010.

[13] M. Balachander, E. Filiot, R. Gentilini, N. Tzevelekos, Register automata with permutations, in: P. Gawrychowski, F. Mazowiecki, M. Skrzypczak (Eds.), 50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025, Warsaw, Poland, August 25-29, 2025, volume 345 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, pp. 14:1–14:18. doi:10.4230/LIPICS.MFCS.2025.14.

[14] N. Tzevelekos, Full abstraction for nominal general references, Log. Methods Comput. Sci. 5 (2009). doi:10.2168/LMCS-5(3:8)2009.

[15] M. Benedikt, C. Ley, G. Puppis, What you must remember when processing data words, in: A. H. F. Laender, L. V. S. Lakshmanan (Eds.), Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management, Buenos Aires, Argentina, May 17-20, 2010, volume 619 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2010. URL: https://ceur-ws.org/Vol-619/paper11.pdf.

[16] M. Gabbay, A. M. Pitts, A new approach to abstract syntax with variable binding, Formal Aspects Comput. 13 (2002) 341–363. doi:10.1007/S001650200016.

A. Omitted Proofs from Section 4

We use the following equality, for all data permutations τ and words v,

$$(\tau \cdot v)^{-1}L = \{w \mid (\tau \cdot v)w \in L\} = \{w \mid v(\tau^{-1} \cdot w) \in L\} = \{\tau \cdot w \mid vw \in L\} = \tau \cdot (v^{-1}L). \quad (1)$$

Proof of Lemma 12. The first equivalence is immediate from Definition 6, as $u^{-1}L = (\tau \cdot v)^{-1}L$ means exactly $u \equiv_L^{\mathsf{MN}} \tau \cdot v$. For the second, $u \equiv_L^{\mathsf{MN}} \tau \cdot v$ iff $[u]_{\equiv_L^{\mathsf{MN}}} = [\tau \cdot v]_{\equiv_L^{\mathsf{MN}}} = \tau \cdot [v]_{\equiv_L^{\mathsf{MN}}}$ by Lemma 11, i.e. iff $[u]_{\equiv_L^{\mathsf{MN}}}$ and $[v]_{\equiv_L^{\mathsf{MN}}}$ are in the same orbit. □

Proof of Lemma 13. We recall that $\mathsf{mem}_L(u)$ is defined as $\mathsf{supp}(u^{-1}L)$. First, note that the equality

above holds true for any representative of $[u]_{\equiv_L^{\mathsf{MN}}}$. Indeed, if $u \equiv_L^{\mathsf{MN}} v$, then $u^{-1}L = v^{-1}L$ and so, we get $\mathsf{mem}_L(u) = \mathsf{supp}(u^{-1}L) = \mathsf{supp}(v^{-1}L) = \mathsf{mem}_L(v)$.

We use the following characterization (†) of least supports [16]: for any nominal set X, element $x \in X$ and datum d, it holds that $d \in supp(x)$ iff the set of data d' such that $(d\ d') \cdot x \neq x$ is infinite.

For brevity, we write $[u]_L$ instead of $[u]_{\equiv_L^{\mathsf{MN}}}$. Then, we prove that $\mathsf{supp}([u]_L) \subseteq \mathsf{mem}_L(u)$. Let $d \in \mathsf{supp}([u]_L)$. By (†), there are infinitely many d' such that $(d\ d') \cdot [u]_L \neq [u]_L$. Take such a datum d'. The inequality $(d\ d') \cdot [u]_L \neq [u]_L$ implies the existence of some data word v such that $v \equiv_L^{\mathsf{MN}} u$ and $(d\ d') \cdot v \not\equiv_L^{\mathsf{MN}} u$. By definition of $\equiv_L^{\mathsf{MN}}$, this is equivalent to $u^{-1}L = v^{-1}L$ and $u^{-1}L \neq ((d\ d') \cdot v)^{-1}L$. As $((d\ d') \cdot v)^{-1}L = (d\ d') \cdot (v^{-1}L)$ by Equation (1) and $u^{-1}L = v^{-1}L$, we get $u^{-1}L \neq (d\ d') \cdot (u^{-1}L)$. This inequality is true for infinitely many d', so by (†), it implies that $d \in \mathsf{supp}(u^{-1}L) = \mathsf{mem}_L(u)$.

Conversely, let $d \in \mathsf{mem}_L(u) = \mathsf{supp}(u^{-1}L)$. By (†), we get that $(d\ d') \cdot (u^{-1}L) \neq u^{-1}L$ for infinitely many data d'. Take such a datum d'. By Equation (1), we get $((d\ d') \cdot u)^{-1}L \neq u^{-1}L$ and so by definition of $\equiv_L^{\mathsf{MN}}$, $(d\ d') \cdot u \not\equiv_L^{\mathsf{MN}} u$. So, $[(d\ d') \cdot u]_L \neq [u]_L$. By Lemma 11, we have $[(d\ d') \cdot u]_L = (d\ d') \cdot [u]_L$. It implies that $[u]_L \neq (d\ d') \cdot [u]_L$. Since it holds for infinitely many d', by (†), it is equivalent to $d \in \mathsf{supp}([u]_L)$. □

Proof of Proposition 15. Given $\mathcal{N}$ as above, we pick for each orbit $c \in Q'$ a representative q_c (i.e. $\mathsf{orb}(q_c) = c$) and an assignment $\rho_c : [1, |\mathsf{supp}(q_c)|] \to \mathsf{supp}(q_c)$. Moreover, for each $q \in \mathsf{orb}(q_c)$, we fix some data permutation τ_q such that $q = \tau_q \cdot q_c$. We define δ' as follows. For each $c \in Q'$ we include in δ' precisely the following transitions. For each $q_c \xrightarrow{d} q'$ in δ, setting $c' = \mathsf{orb}(q')$:

- if $d \in \mathsf{supp}(q_c)$ then add $c \xrightarrow{i,\pi} c'$ in δ', where $i = \rho_c^{-1}(d)$ and π is the canonical extension to $[1, r]$ of $\rho_c; \tau_{q'}^{-1}; \rho_{c'}^{-1}$;
- if $d \notin \mathsf{supp}(q_c)$ then add $c \xrightarrow{i^\bullet,\pi} c'$ in δ', where $i = \min(\{i \mid \rho_c(i) \notin \mathsf{supp}(q_{c'})\} \cup \{|\mathsf{supp}(q_c)|+1\})$ and π is the canonical extension to $[1, r]$ of $\rho_c[i \mapsto d]; \tau_{q'}^{-1}; \rho_{c'}^{-1}$.

We next show that the relation

$$R = \{(q, (c, \rho_c; \tau)) \mid q \in Q \wedge c = \mathsf{orb}(q) \wedge q = \tau \cdot q_c\}$$

is a bisimulation. Note first that $\mathsf{supp}(\rho_c; \tau) = \mathsf{supp}(\tau \cdot \rho_c) = \tau \cdot \mathsf{supp}(\rho_c) = \tau \cdot \mathsf{supp}(q_c) = \mathsf{supp}(q)$, where the last equality follows from $q = \tau \cdot q_c$. Now suppose $(q, (c, \rho_c; \tau)) \in R$.

- If $q \xrightarrow{d} q'$ then $q_c \xrightarrow{\tau^{-1}(d)} q'' = \tau^{-1} \cdot q'$ and, by definition: if $d \in \mathsf{supp}(q)$ then $c \xrightarrow{i,\pi} c'$, and if $d \notin \mathsf{supp}(q)$ then $c \xrightarrow{i^\bullet,\pi} c'$, where $c' = \mathsf{orb}(\tau^{-1} \cdot q') = \mathsf{orb}(q')$. In either case, $(c, \rho_c; \tau) \xrightarrow{d} (c', \rho')$. We need to show that $(q', (c', \rho')) \in R$ and in particular that $\rho' = \rho_{c'}; \tau'$ for some $q' = \tau' \cdot q_{c'}$. If $d \in \mathsf{supp}(q)$ then:

  $$\rho' = (\pi^{-1}; \rho_c; \tau)|_{\mu(c')} = \mathsf{id}_{\mu(c')}; \rho_{c'}; \tau_{q''}; \rho_c^{-1}; \rho_c; \tau = \rho_{c'}; \tau_{q''}; \tau$$

  and $(\tau_{q''}; \tau) \cdot q_{c'} = \tau \cdot \tau_{q''} \cdot q_{c'} = \tau \cdot q'' = q'$. On the other hand, if $d \notin \mathsf{supp}(q)$ then:

  $$\begin{aligned}\rho' &= (\pi^{-1}; (\rho_c; \tau)[i \mapsto d])|_{\mu(c')} = \mathsf{id}_{\mu(c')}; \rho_{c'}; \tau_{q''}; \rho_c[i \mapsto \tau^{-1}(d)]^{-1}; (\rho_c; \tau)[i \mapsto d] \\ &= \rho_{c'}; \tau_{q''}; \rho_c[i \mapsto \tau^{-1}(d)]^{-1}; \rho_c[i \mapsto \tau^{-1}(d)]; \tau = \rho_{c'}; \tau_{q''}; \tau\end{aligned}$$

  and we conclude as above.
- If $c \xrightarrow{i,\pi} c'$ or $c \xrightarrow{i^\bullet,\pi} c'$ then let $q_c \xrightarrow{d} q'$ be the corresponding transition in δ. We can repeat the argument from the previous case above to find a τ' such that $(q', (c', \rho_{c'}; \tau')) \in R$.

Note that R is injective: if $(q_1, (c, \rho_c; \tau_1)), (q_2, (c, \rho_c; \tau_2)) \in R$ with $\rho_c; \tau_1 = \rho_c; \tau_2$ then $\tau_1|_{\mathsf{supp}(q_c)} = \tau_2|_{\mathsf{supp}(q_c)}$ and, thus, $q_1 = \tau_1 \cdot q_c = \tau_2 \cdot q_c = q_2$. If Q is a strong nominal set then we show that R is a function and, therefore, an isomorphism. Suppose that Q is strong and let $(q, (c_1, \rho_{c_1}; \tau_1)), (q, (c_2, \rho_{c_2}; \tau_2)) \in R$. By definition, $c_1 = c_2 = \mathsf{orb}(q)$, so let us write $c = c_1 = c_2$. We have that $q = \tau_1 \cdot q_c = \tau_2 \cdot q_c$, therefore $q = (\tau_1; \tau_2^{-1}) \cdot q$. By strong support, the latter implies that $(\tau_1; \tau_2^{-1})(d) = d$ for all $d \in \mathsf{supp}(q_c) = \mathsf{rng}(\rho_c)$, therefore $\rho_c; \tau_1; \tau_2^{-1} = \rho_c$, from which we obtain $\rho_c; \tau_1 = \rho_c; \tau_2$ by multiplying both sides with τ_2. □