



Nominal Tree Automata with Name Allocation

Simon Prucker  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Lutz Schröder  

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

Data trees serve as an abstraction of structured data, such as XML documents. A number of specification formalisms for languages of data trees have been developed, many of them adhering to the paradigm of register automata, which is based on storing data values encountered on the tree in registers for subsequent comparison with further data values. Already on word languages, the expressiveness of such automata models typically increases with the power of control (e.g. deterministic, non-deterministic, alternating). Language inclusion is typically undecidable for non-deterministic or alternating models unless the number of registers is radically restricted, and even then often remains non-elementary. We present an automaton model for data trees that retains a reasonable level of expressiveness, in particular allows non-determinism and any number of registers, while admitting language inclusion checking in elementary complexity, in fact in parametrized exponential time. We phrase the description of our automaton model in the language of nominal sets, building on the recently introduced paradigm of explicit name allocation in nominal automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Data languages, tree automata, nominal automata, inclusion checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.35

Related Version *Full Version:* <https://arxiv.org/abs/2405.14272> [32]

Funding Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) -- Projektnummer 517924115.

1 Introduction

Letters from infinite alphabets generally serve as an abstraction of data values in formalisms for the specification and verification of structured data such as data words or data trees (e.g. [29]). They have variously been used to represent data values in XML documents [29], object identities [14], parameters of method calls [17], or nonces in cryptographic protocols [24]. There are, by now, quite a number of automata models for data languages, including register automata [20], data walking automata [27], and data automata [3]. A typical phenomenon in such models is that expressiveness increases strictly with the power of control (ranging from deterministic to alternating models). In such models, the key reasoning problem of inclusion checking tends to be either undecidable or computationally very hard unless stringent restrictions are imposed on either the power of control or on key parameters such as the number of registers. For instance, inclusion checking of nondeterministic register automata is undecidable unless one restricts either to unambiguous automata [28, 7] or to automata with at most two registers [20] (no complexity bound being given in the latter case); inclusion checking for alternating register automata is undecidable unless one restricts to only one register, and even then fails to be primitive recursive [11]; the inclusion problem of data walking automata is decidable but at least as hard as Petri net reachability [8], which by recent results is Ackermann-complete [25, 10, 26]; non-emptiness of data automata [3] is decidable but, again, at least as hard as Petri net reachability; and emptiness of variable automata [15] is undecidable unless one restricts to the (less expressive) deterministic fragment.



© Simon Prucker and Lutz Schröder;

licensed under Creative Commons License CC-BY 4.0

35th International Conference on Concurrency Theory (CONCUR 2024).

Editors: Rupak Majumdar and Alexandra Silva; Article No. 35; pp. 35:1–35:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Register-based automata models are often essentially equivalent to automata models in nominal sets [31]; for instance, register automata with nondeterministic reassignment [22] are equivalent to *nondeterministic orbit-finite automata* [4]. In the nominal setting, one way to ameliorate the mentioned degree of computational hardness while retaining a reasonable level of expressiveness is provided by the paradigm of explicit *name allocation* in nominal automata, which first appeared in *regular nondeterministic nominal automata (RNNA)* [34] and has subsequently been used in Büchi RNNA [40] and in a linear-time nominal μ -calculus [16]. In this paradigm, notions of freshness are based on α -equivalence, i.e. renaming of bound names as in λ -calculus, which blocks renaming of bound names into names that have free occurrences later in the word; in terms of the equivalent register-based setting, this amounts to a lossiness condition saying that at every point, register contents may nondeterministically be erased (thus freeing the register). At the same time, name-allocating models impose finite branching. Inclusion checking in these models is typically elementary, and in fact has low parametrized complexities with the parameter being the *degree*, which in the register-based setting corresponds to the number of registers.

Our present contribution is a nominal non-deterministic top-down tree automaton model that follows the name allocation paradigm. As our main result, we show that our model of *regular nominal tree automata (RNTA)* admits inclusion checking in doubly exponential time, more precisely in parametrized singly exponential time with the degree as the parameter (recall that the problem is already EXPTIME-complete for finite-alphabet non-deterministic top-down tree automata). We thus obtain an efficiently decidable formalism for the specification data words that admits full non-determinism and unboundedly many registers.

Omitted proofs can be found in the full version [32].

Related Work. We have already mentioned work on register word automata. Kaminski and Tan [21] introduce top-down and bottom-up register tree automata with or without non-deterministic reassignment; computational hardness of inclusion and universality is inherited from register word automata (while membership and emptiness are decidable in elementary complexity NP and EXPTIME, respectively [36]). Without non-deterministic reassignment, the top-down model and the bottom-up model have incomparable expressiveness. Our model of regular nominal tree automata (RNTA) relates, along the usual correspondence [4], to non-deterministic top-down register tree automata without non-deterministic reassignment. Van Heerdt et al. [41] treat bottom-up nominal tree automata with name allocation in a general algebraic setting, describing minimization and determinization constructions (without considering inclusion checking; determinization produces orbit-infinite automata). Since the finite-branching condition for the bottom-up variant differs from top-down finite branching as imposed in RNTAs, we expect similar incomparability as in the register-based setting; we leave the investigation of this point to future work. A register automata model for data trees with a different navigation pattern has been introduced by Jurdziński and Lazić [18, 19] and extended by Figueira [12]; in this model, the automaton moves downwards or rightwards on the tree instead of passing copies of itself to child nodes. The emptiness problem of the alternating model is decidable but not primitive recursive when the number of registers is restricted to 1; expressiveness is incomparable to Kaminski and Tan’s model [19]. One formalism for data trees that does allow inclusion checking in elementary complexity is the logic $FO^2(+1, \sim)$ [5], whose satisfiability problem is in 3NEXPTIME. Register tree automata have been extended to cover an ordering on the data values [37, 38].

2 Preliminaries

We assume basic familiarity with category theory (e.g. [1]). We briefly recall some background on nominal sets (see [31] for a textbook treatment), as well as on tree automata (e.g. [9]), register automata [20], and nominal automata [4].

Group Actions and Nominal sets. We fix a countably infinite set \mathbb{A} of *names*. Throughout, we let G denote the group of finite permutations on \mathbb{A} , which is generated by the permutations (ab) that swap two names $a, b \in \mathbb{A}$; we write id for the identity permutation, and $(-)\cdot(-)$ for the group operation, i.e. applicative-order composition of permutations. A G -set consists of a set X and a left action $(-)\cdot(-): G \times X \rightarrow X$ of G on X (subject to $id \cdot x = x$ and $\pi \cdot (\pi' \cdot x) = (\pi \cdot \pi') \cdot x$). Given G -sets X, Y , a map $f: X \rightarrow Y$ is *equivariant* if $f(\pi \cdot x) = \pi \cdot f(x)$ for all x, y , and a subset $S \subseteq X$ is *equivariant* if $\pi \cdot s \in S$ for all $\pi \in G$ and $s \in S$, i.e. S is closed under the group action. The *orbit* of $x \in X$ is $G \cdot x = \{\pi \cdot x \mid \pi \in G\}$. The orbits form a disjoint decomposition of X ; the G -set X is *orbit-finite* if it has only finitely many orbits.

We write $\text{fix}(x) = \{\pi \in G \mid \pi(x) = x\}$ for $x \in X$, and $\text{Fix}(A) = \bigcap_{x \in A} \text{fix}(x)$ for $A \subseteq X$. The set \mathbb{A} itself is a G -set in a canonical manner. A set $S \subseteq \mathbb{A}$ is a *support* of an element $x \in X$ if

$$\text{Fix}(S) \subseteq \text{fix}(x),$$

that is, if every permutation that fixes all names in S also fixes x , which we understand as x depending only on the names in S .

Then, a G -set X is a *nominal set* if every element of X has a finite support. It follows that every $x \in X$ has a *least* finite support $\text{supp}(x)$, also just called the *support* of x . A name $a \in \mathbb{A}$ is *fresh* for x if $a \notin \text{supp}(x)$, in which case we write $a \# x$. We write Nom for the category of nominal sets and equivariant maps. Examples of nominal sets include \mathbb{A} itself (with $\text{supp}(a) = \{a\}$ for $a \in \mathbb{A}$); the product $X \times Y$ of nominal sets X, Y (with $\text{supp}(x, y) = \text{supp}(x) \cup \text{supp}(y)$); and the finitely supported powerset $\mathcal{P}_{\text{fs}}X$ of a nominal set X , which consists of the subsets of X that have finite support under the pointwise action of G on the full powerset. A set $A \subseteq X$ is *uniformly finitely supported* if $\bigcup_{x \in A} \text{supp}(x)$ is finite; we write $\mathcal{P}_{\text{ufs}}X = \{A \subseteq X \mid A \text{ uniformly finitely supported}\}$ for the *uniformly finitely supported powerset* of X . If X is orbit-finite, then the uniformly finitely supported subsets of X are precisely the finite subsets.

An important role in the technical development is played by the *abstraction functor* $[\mathbb{A}](-)$ on Nom . For a nominal set X , $[\mathbb{A}]X$ is defined as the quotient $\mathbb{A} \times X / \sim$ of $\mathbb{A} \times X$ by the equivalence relation \sim defined by

$$(a, x) \sim (b, y) \quad \text{iff} \quad (ca) \cdot x = (cb) \cdot y \quad \text{for } c \# (a, x, b, y).$$

Equivalently, $(a, x) \sim (b, y)$ iff either $(a, x) = (b, y)$ or $y = (ab) \cdot x$ and $b \# x$. We write $\langle a \rangle x$ for the equivalence class of (a, x) under \sim . Thus, $\langle a \rangle x$ may be read as arising from x by binding the name a . The equivalence \sim then captures α -equivalent renaming of the bound name a ; in particular, note that by the alternative description of \sim , renaming a into b in $\langle a \rangle x$ is *blocked* when $b \in \text{supp}(x)$.

Nominal automata and register automata. The classical notion of nondeterministic finite automaton can be transferred canonically to the category of nominal sets, where finiteness corresponds to orbit-finiteness; this gives rise to the notion of *nondeterministic orbite-finite automaton (NOFA)* [4]. For simplicity, we use the set \mathbb{A} of names as the alphabet (more

generally, one can work with any orbit-finite alphabet). Then, a NOFA consists of an orbit-finite set Q of states; an equivariant transition relation $\Delta \subseteq Q \times \mathbb{A} \times Q$; an equivariant initial state q_0 (or more generally a set of initial states); and an equivariant set $F \subseteq Q$ of final states. NOFAs accept finite words over \mathbb{A} , with the notions of run and acceptance defined exactly as in the classical case.

NOFAs are equivalent to a flavour of *nondeterministic register automata with nondeterministic reassignment*, roughly described as follows (see [4] for details). A register automaton has a finite set Q of control states; a fixed finite number of registers, which at any point in time can be either empty or contain a letter (i.e. a name from \mathbb{A}); an initial control state q_0 ; a set F of final control states; and a transition relation δ consisting of triples (q, ϕ, q') where $q, q' \in Q$ and where ϕ is a boolean combination of equality constraints concerning register contents before and after the transition and the current input letter. For simplicity, we require that all registers are initially empty. A *configuration* of the automaton consists of a control state and an assignment of letters to some of the registers (the others are empty). A *run* of the automaton is a sequence of configurations starting in the initial configuration (consisting of q_0 and all registers empty) such that every next configuration is justified by some transition (q, ϕ, q') in the sense that the boolean combination ϕ of equality constraints is satisfied by the register contents in the configurations before and after the transition and by the current input letter. A run is accepting if it ends in a configuration over a final control state. The language of the automaton is the set of accepted words. What this means is that the automaton can, in each transition step, perform any combination of the following actions as specified by the transition constraint ϕ : store the current letter in a register; copy content among registers; delete content from a register; nondeterministically store any name in a register (nondeterministic reassignment). These actions are conditioned on tests for equality or inequality among the registers and the input letter. In the correspondence between NOFAs and register automata, a configuration c of a register automaton becomes a state in the corresponding NOFA, whose support contains precisely the letters stored in the registers in c .

Both nondeterminism and nondeterministic reassignment strictly increase expressive power. For instance, the language L_2 “some letter occurs twice” can be accepted by a nondeterministic register automaton but not by a deterministic one; and the language “the last letter has not been seen before” can only be accepted using nondeterministic reassignment. Also, the nondeterministic model is not closed under complement: The complement of the above-mentioned language L_2 is the language “all letters are distinct”, which cannot be accepted by a nondeterministic register automaton.

Tree automata. Throughout, we fix a finite *ranked alphabet* (or *signature*) Σ , consisting of finitely many (*function*) *symbols*, each equipped with an assigned finite *arity*; we write $f/n \in \Sigma$ to indicate that f is a symbol of arity n in Σ . We assume that Σ contains at least one *constant*, i.e. a symbol of arity 0. The set $\mathcal{T}(\Sigma)$ of (ground) Σ -*Terms* is defined inductively by stipulating that whenever $f/n \in \Sigma$ and $t_1, \dots, t_n \in \mathcal{T}(\Sigma)$, then $f(t_1, \dots, t_n)$ in Σ . Terms are regarded as a representation of trees, with each node of the tree labelled with a symbol from Σ whose arity determines the number of child nodes.

A (classical, finite-alphabet) *nondeterministic top-down tree automaton (NFTA)* (e.g. [9]) over Σ is a tuple $A = (Q, q_0, \Delta)$ (we elide the fixed signature Σ) where Q is a finite set of *states*, $q_0 \in Q$ is the *initial state*, and Δ is a set of *rewrite rules* or *transitions* of the form

$$q(f(x_1, \dots, x_n)) \rightarrow f(q_1(x_1), \dots, q_n(x_n))$$

where $f/n \in \Sigma$, $q, q_1, \dots, q_n \in Q$, and the x_i are variables; these rules thus manipulate *extended terms* containing automata states as unary symbols (with at most one occurrence of such a symbol per tree branch). Much as usual (e.g. [2, 9]), such rewrite rules are applied within a surrounding context, and with the variables x_i substituted with ground terms; we continue to write \rightarrow for the arising rewrite relation on extended terms. When $f/n \in \Sigma$ has arity $n > 1$, then a rewrite rule of the above shape transforms a single automaton state q into several automaton states q_1, \dots, q_n . One may view this phenomenon as the automaton creating copies of itself, which continue to operate independently on the children of the present node. The NFTA A *accepts* a term t if $q_0(t) \rightarrow^* t$. The *language* $L(A)$ is the set of terms (trees) accepted by A . NFTAs have the same expressiveness as *bottom-up tree automata*, in which the rewrite rules propagate automata states from the leaves to the root. Deterministic top-down tree automata, on the other hand, are strictly less expressive than NFTA. In the present work, our focus is on nondeterministic top-down models. The *inclusion problem* for NFTA, i.e. to decide whether $L(A) \subseteq L(B)$ for given NFTAs A, B , is EXPTIME-complete [35].

3 A Nominal View on Data Tree Languages

We proceed to introduce the relevant notion of nominal tree language, viewed as representing a data tree language. Trees are represented as a form of algebraic terms, and carry data values on their nodes. Throughout the technical development, we fix a finite algebraic *signature* Σ , i.e. a finite set of operation symbols f, g, \dots , each equipped with a finite *arity*. We write $f/n \in \Sigma$ to indicate that f is an operation symbol of arity n in Σ .

► **Definition 3.1** (Terms). We define the set $\mathcal{T}_{\mathbb{A}}(\Sigma)$ of *nominal Σ -terms*, or just *(Σ -)terms*, t by the grammar

$$t ::= a.f(t_1, \dots, t_n) \mid \nu a.f(t_1, \dots, t_n) \quad (t_1, \dots, t_n \in \mathcal{T}_{\mathbb{A}}(\Sigma), f/n \in \Sigma, a \in \mathbb{A}) \quad (3.1)$$

For uniformity of notation, we occasionally write $\bar{\mathbb{A}}$ for the set $\mathbb{A} \cup \{\nu a \mid a \in \mathbb{A}\}$; in particular, all terms then have the form $\gamma.f(t_1, \dots, t_n)$ with $\gamma \in \bar{\mathbb{A}}$.

► **Remark 3.2.** Like in the finite-alphabet case as recalled in Section 2, the base case of the above definition is the one where f is a constant (i.e. has arity 0). The case of words is recovered by taking Σ to consist of unary operations and an end-of-word constant. When viewing terms as trees, we see the operations of Σ as spanning the tree structure, with every node being labelled with an element of $\bar{\mathbb{A}}$. We understand terms of the form $a.f(t_1, \dots, t_n)$ as being labelled with a free name a , and terms of the form $\nu a.f(t_1, \dots, t_n)$ as allocating a new name a with scope $f(t_1, \dots, t_n)$. In the latter case, the name a is bound by the ν -operator, in the same style as in the π -calculus or in nominal Kleene algebra [13], with formal definitions to be provided presently. (In work on nominal word automata with name allocation [34], the notation $|a$ has been used in place of νa .)

► **Remark 3.3.** For brevity of presentation, we have opted for a setup where every node either binds a name or carries a free name. A generalization that allows nodes not labelled with any name is easily encoded into the present setup by means of a fixed free dummy name that appears in place of the absent name. We thus do use this generalization in the examples. In particular, it allows for trees not containing any name, while terms according to Definition 3.1 always contain at least one name.

The notion of free name informally used above is formally defined in the expected way:

► **Definition 3.4.** The set $\text{FN}(t)$ of *free names* of a term t is defined recursively by the clauses

$$\begin{aligned}\text{FN}(a.f(t_1, \dots, t_n)) &= \text{FN}(t_1) \cup \dots \cup \text{FN}(t_n) \cup \{a\} \\ \text{FN}(\nu a.f(t_1, \dots, t_n)) &= (\text{FN}(t_1) \cup \dots \cup \text{FN}(t_n)) \setminus \{a\}.\end{aligned}$$

Contrastingly, we refer to the name a in a term $\nu a.f(t_1, \dots, t_n)$ as a *bound name*. A term t is *closed* if $\text{FN}(t) = \emptyset$. The sets $\overline{\mathbb{A}}$ and $\mathcal{T}_{\mathbb{A}}(\Sigma)$ become nominal sets under the expected actions of G , defined on $\overline{\mathbb{A}}$ by $\pi \cdot a = \pi(a)$ and $\pi \cdot \nu a = \nu \pi(a)$, and on $\mathcal{T}_{\mathbb{A}}(\Sigma)$ recursively by

$$\pi \cdot (\delta.f(t_1, \dots, t_n)) = \pi(\delta).f(\pi \cdot t_1, \dots, \pi \cdot t_n).$$

From this view, we obtain the expected notion of α -equivalence of terms: α -*equivalence* \equiv_{α} is the least congruence on terms such that $\nu a.f(t_1, \dots, t_n) \equiv_{\alpha} \nu b.f(t'_1, \dots, t'_n)$ whenever $\langle a \rangle(t_1, \dots, t_n) = \langle b \rangle(t'_1, \dots, t'_n)$, which means that in $\nu a.f(t_1, \dots, t_n)$, a can be renamed into b provided that b does not occur in t_1, \dots, t_n (by temporary renaming of inner bound names, one can then also rename a into b if b is not free in t_1, \dots, t_n). We write $[t]_{\alpha}$ for the equivalence class of a term t under α -equivalence. A term t is *clean* if all its bound names are mutually distinct and not free in t , and *non-shadowing* if on every branch of t , all bound names are mutually distinct and not free in t (in particular, no bound name is ever shadowed in t). Clearly, every term is α -equivalent to a clean one (hence, a fortiori, to a non-shadowing one).

► **Example 3.5.** Under the extension where names are made optional (Remark 3.3), we can represent λ -terms as terms for the signature $\{\text{app}/2, \lambda/1, \text{var}/0\}$. For instance, the λ -term $\lambda a.aa$ is represented as the Σ -term $\nu a.\lambda(\text{app}(a.\text{var}, a.\text{var}))$. (Of course, there are Σ -terms not corresponding to any λ -term, such as $\nu a.\text{var}$; in our automaton model, it will be no problem to exclude such terms by just letting them get stuck.) Similarly, we can represent π -calculus expressions as terms for a suitable signature; we will return to this in Example 4.7. The notion of α -equivalence defined above is exactly the standard one in these examples.

As indicated in the introduction, the notion of α -equivalence determines how we interpret allocating a new name as “reading a name” in a paradigm where we see bound names as placeholders for arbitrary free names. For this purpose, we distinguish between *literal tree languages*, which consist of terms and hence are subsets of $\mathcal{T}_{\mathbb{A}}(\Sigma)$, and *alphatic tree languages*, which consist of α -equivalence classes of terms and hence are subsets of $\mathcal{T}_{\mathbb{A}}(\Sigma) / \equiv_{\alpha}$. (The latter generalize the *bar languages* used in work on nominal word automata [34] but in the absence of the bar notation \bar{a} , that term seems no longer appropriate.) In both cases, we say that a language is *closed* if it consists of closed (equivalence classes of) words only. For brevity, we restrict the main line of the technical treatment to closed languages. The notion of bound names representing free names is captured by the function $d\nu$, which removes all occurrences of ν from a term, and is recursively defined by

$$d\nu(\nu a.f(t_1, \dots, t_n)) = d\nu(a.f(t_1, \dots, t_n)) = a.f(d\nu(t_1), \dots, d\nu(t_n)).$$

Thus, $d\nu$ returns terms without name allocation νa . We refer to such terms as *data trees*; they correspond essentially to the notion of data tree found in the literature (which is often restricted to binary trees for simplicity, e.g. [21, 18]), with \mathbb{A} serving as the infinite alphabet of data values. We capture notions of freshness by imposing different disciplines on variable management in α -renaming and subsequently applying $d\nu$. Specifically, a *global freshness semantics*, a *branchwise freshness semantics*, and a *local freshness semantics* for alphatic tree languages L are embodied in the operators \mathbf{N} , \mathbf{B} , and \mathbf{D} , respectively, defined by

$$\begin{aligned} \mathbf{N}(L) &= \{d\nu(t) \mid t \text{ clean}, [t]_\alpha \in L\} & \mathbf{B}(L) &= \{d\nu(t) \mid t \text{ non-shadowing}, [t]_\alpha \in L\} \\ \mathbf{D}(L) &= \{d\nu(t) \mid [t]_\alpha \in L\}. \end{aligned}$$

That is, N and B insist on clean or non-shadowing α -renaming, respectively, while D allows unrestricted α -renaming. The languages $\mathbf{N}(L)$, $\mathbf{B}(L)$, and $\mathbf{D}(L)$ are what we term *data tree languages*, i.e. languages consisting only of data trees. This makes one additional semantics in comparison to the case of words [34] where, of course, N and B coincide.

► **Example 3.6.** Consider $\Sigma = \{f/2, k/0\}$ and the term

$$t = \nu a.f(\nu b.f(a.k, b.k), \nu b.f(b.k, b.k)).$$

Then

$$\begin{aligned} \mathbf{N}(\{t\}) &= \{a.f(b.f(a.k, b.k), c.f(c.k, c.k)) \mid a, b, c \in \mathbb{A}, a \neq b, a \neq c, b \neq c\} \\ \mathbf{B}(\{t\}) &= \{a.f(b.f(a.k, b.k), c.f(c.k, c.k)) \mid a, b, c \in \mathbb{A}, a \neq b, a \neq c\} \\ \mathbf{D}(\{t\}) &= \{a.f(b.f(a.k, b.k), c.f(c.k, c.k)) \mid a, b, c \in \mathbb{A}, a \neq b\}. \end{aligned}$$

Thus, N instantiates bound names by free names that are fresh w.r.t. the entire tree, while B only enforces freshness w.r.t. the current branch, and allows siblings to instantiate bound names by the same free name (i.e. allows $b = c$ in $\mathbf{B}(\{t\})$). Finally, D , enforces freshness only where α -renaming is blocked. In the case of t , renaming b into a is blocked in the left-hand subterm $\nu b.f(a.k, b.k)$ because of the free occurrence of a , while there is no such occurrence in the right-hand subterm $\nu b.f(b.k, b.k)$ so that b can be renamed into a in that subterm; this is why $\mathbf{D}(\{t\})$ requires $a \neq b$ but not $a \neq c$. Thus, N and B are variants of global freshness as found, for instance, in session automata [6], while D is indeed a notion of local freshness: We will see later (Lemma 4.2) that the presence of the free name a in $\nu b.f(a.k, b.k)$ implies that at the time of reading νb in a run of a regular nominal tree automaton, the name a is still kept in memory, so D essentially enforces freshness w.r.t. currently stored names in the spirit of register automata models. As indicated in the introduction, we trade some of the expressiveness of register automata models for computational tractability. In our example, this is apparent in the right-hand subterm $\nu b.f(b.k, b.k)$, in which freshness of b w.r.t. a cannot be enforced under D because there is no free occurrence of a ; as a slogan, D enforces freshness w.r.t. stored names only if these are still expected to be seen again. In work on nominal word automata [34], it is shown that this phenomenon relates to a lossiness property stating that register contents may be non-deterministically lost at any time.

It turns out that both variants of global freshness remain essentially equivalent to the original alphatic language, in the sense that they do not affect language inclusion:

► **Lemma 3.7.** *Both N and B preserve and reflect language inclusion: For alphatic languages L_1, L_2 , we have $L_1 \subseteq L_2$ iff $\mathbf{N}(L_1) \subseteq \mathbf{N}(L_2)$ iff $\mathbf{B}(L_1) \subseteq \mathbf{B}(L_2)$.*

That is, for purposes of checking language inclusion, it does not matter whether we consider alphatic languages, their global freshness semantics, or their branchwise freshness semantics.

4 Regular Nominal Tree Automata

We cast our model of *regular nominal tree automata* (RNTAs) as an extension of *regular nondeterministic nominal automata* (RNNAs) [34], which differ from NOFAs (Section 2) in essentially two ways: Branching is restricted to be finite (in NOFAs, the set of successors of a state only needs to be finitely supported, as implied by equivariance of the transition relation); this is partially made up for by including *bound transitions* which read bound names. RNTAs natively accept alphabetic tree languages, which as discussed in Section 3 may be seen as representing languages of data trees in a number of ways differing w.r.t. notions of freshness: Under global or branchwise freshness semantics, RNTAs may be seen as a generalization of session automata [6], while under local freshness semantics, they will be seen to correspond to a subclass of nondeterministic register tree automata [21] characterized by a lossiness condition (Remark 5.6). As indicated in the introduction and in Example 3.6, we thus incur a decrease in expressiveness in comparison to the register model, which however buys elementary complexity of inclusion checking.

► **Definition 4.1** (Regular nominal tree automata). A *regular nominal tree automaton* (RNTA) over our fixed signature Σ is a tuple

$$A = (Q, \Delta, q_0)$$

where Q is a orbit-finite nominal set of *states*, $q_0 \in Q$ is the *initial state* and Δ is an equivariant set of *rewrite rules* or *transitions* of the form

$$q(\gamma.f(x_1, \dots, x_n)) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))$$

where $q_1, \dots, q_n \in Q$, $\gamma \in \overline{\mathbb{A}}$, $f/n \in \Sigma$, and the x_i are variables. When no confusion is likely, we just write $q(\gamma.f(x_1, \dots, x_n)) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))$ to indicate that $(q(\gamma.f(x_1, \dots, x_n)) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))) \in \Delta$. We impose two properties on Δ :

■ *α -invariance*: For $q, q_1, \dots, q_n, q'_1, \dots, q'_n \in Q$, if $\langle a \rangle(q_1, \dots, q_n) = \langle b \rangle(q'_1, \dots, q'_n)$, then

$$\begin{aligned} q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q_1(x_1), \dots, q_n(x_n)) &\text{ implies} \\ q(\nu b.f(x_1, \dots, x_n)) \rightarrow \nu b.f(q'_1(x_1), \dots, q'_n(x_n)). \end{aligned}$$

■ *Finite branching up to α -equivalence*: For all $q \in Q$ and $f/n \in \Sigma$, the sets $\{(a, (q_1, \dots, q_n)) \mid q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q_1(x_1), \dots, q_n(x_n))\}$ and $\{\langle a \rangle(q_1, \dots, q_n) \mid q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q_1(x_1), \dots, q_n(x_n))\}$ are finite.

Like in the classical case (Section 2), the rewrite rules in Δ may be applied within a surrounding context and with variables substituted by (ground) terms. A state $q \in Q$ *accepts* a term t if there exists a sequence of applications of the rewrite rules in Δ , called a *run*, that starts from $q(t)$ and ends in t , symbolized as $q(t) \xrightarrow{*} t$. We define the *literal tree language* $L(q)$ and the *alphabetic tree language* $L_\alpha(q)$ accepted by q by

$$L(q) = \{t \in \mathcal{T}_{\mathbb{A}}(\Sigma) \mid q \text{ accepts } t\} \quad L_\alpha(q) = \{[t]_\alpha \mid t \in L(q)\}$$

We put $L(A) = L(q_0)$ and $L_\alpha(A) = L_\alpha(q_0)$, i.e. the RNTA A *accepts* t if its initial state accepts t . Moreover, A *accepts* a data tree s under *global*, *branchwise*, or *local freshness semantics* if $s \in \mathbf{N}(L_\alpha(A))$, $s \in \mathbf{B}(L_\alpha(A))$, or $s \in \mathbf{D}(L_\alpha(A))$, respectively (cf. Section 3).

The *degree* of A is the maximal cardinality of supports of states in A .

We think of the support of an RNTA state as consisting of finitely many stored names. In examples, we typically write states in the form

$$q(a_1, \dots, a_n)$$

where q indicates the orbit and a_1, \dots, a_n are stored names. Thus, the degree of an RNTA corresponds morally to the number of registers. As an important consequence of finite branching, stored names can come about only by either inheriting them from a predecessor state or by reading (i.e. binding) them:

► **Lemma 4.2.** *In an RNTA, we have the following properties:*

1. *If $q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q_1(x_1), \dots, q_n(x_n))$, then $\text{supp}(q_i) \cup \{a\} \subseteq \text{supp}(q)$ for $i = 1, \dots, n$.*
2. *If $q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q_1(x_1), \dots, q_n(x_n))$, then $\text{supp}(q_i) \subseteq \text{supp}(q) \cup \{a\}$ for $i = 1, \dots, n$.*

► **Corollary 4.3.** *If state q accepts term t , then $\text{FN}(t) \subseteq \text{supp}(q)$.*

► **Remark 4.4.** For brevity, we restrict the further technical treatment to the case where *the initial state has empty support*, which by Corollary 4.3 implies that *the accepted language is closed*. We also assume this without further mention in the examples, with the possible exception of the dummy name needed in examples with unlabelled nodes (cf. Remark 3.3), in which the dummy name is assumed to be in the support of all states.

► **Example 4.5.** Let $\Sigma = \{f/2, k/0\}$ (so Σ -terms are just $\bar{\mathbb{A}}$ -labelled binary trees).

1. The universal data tree language, i.e. the language consisting of all (non-empty, cf. Remark 3.3) data trees, is accepted under local freshness semantics by the RNTA with only one state q and transitions $q(\nu a.f(x, y)) \rightarrow \nu a.f(q(x), q(y))$, $q(\nu a.k) \rightarrow \nu a.k$. On the other hand, it is easy to see that the universal data tree language cannot be accepted by an RNTA under global or branchwise freshness semantics. Under the latter semantics, the above RNTA accepts the language of all data trees in which all names are distinct or in which all names found on the same branch are distinct, respectively.
2. The data tree language “the letter at the root of the tree (which moreover is not a leaf) reappears in all leaves, but not in any other node of the tree” is accepted under local freshness semantics by the RNTA with states $q_0, q_1(a)$ ($a \in \mathbb{A}$) and transitions

$$\begin{aligned} q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_1(a)(x), q_1(a)(y)) \\ q_1(a)(\nu b.f(x, y)) &\rightarrow \nu b.f(q_1(a)(x), q_1(a)(y)) \quad q_1(a)(a.k) \rightarrow a.k \end{aligned}$$

where we mean this and all further examples to be implicitly closed under equivariance and α -invariance. (Regarding notation, read $q_1(a)(y)$ as “state $q_1(a)$ processing term y ”.)

3. The data tree language “there is some letter that appears twice on the same branch” (which, in analogy to the word case [4, 34], cannot be accepted by a deterministic register tree automaton) is accepted under local freshness semantics by the RNTA with states $q_0, q_1(a), q_2$ ($a \in \mathbb{A}$), transitions

$$\begin{aligned} q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_0(x), q_2(y)) & q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_2(x), q_0(y)) \\ q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_1(a)(x), q_2(y)) & q_0(\nu a.f(x, y)) &\rightarrow \nu a.f(q_2(x), q_1(a)(y)) \\ q_1(a)(\nu b.f(x, y)) &\rightarrow \nu b.f(q_1(a)(x), q_2(y)) & q_1(a)(\nu b.f(x, y)) &\rightarrow \nu b.f(q_2(x), q_1(a)(y)) \\ q_1(a)(a.f(x, y)) &\rightarrow a.f(q_2(x), q_2(y)), \end{aligned}$$

and transitions ensuring that q_2 accepts the universal data tree language as in item 1. It is easy to see that the complement of this language, while acceptable under branchwise freshness semantics as seen in item 1., cannot be accepted by an RNTA under local freshness semantics.

35:10 Nominal Tree Automata with Name Allocation

► **Example 4.6** (Structured data). We use $\Sigma = \{\text{!elem}/2, \text{\#data}/1, \text{eof}/0\}$ to support an XML-like syntax for structured data. We want to recognize the language of Σ -trees where every occurrence of **!elem** is properly closed by **eof** in the subtree below it, at a leaf as far to the left in the tree as possible under the policy that later occurrences of **!elem** are closed further to the left. Occurrences of **eof** and **!elem** are matched by binding a name at **!elem** and labelling the corresponding **eof** with this name (moreover, one unlabelled **eof** closes the entire term), as in the term

```

νa.!elem(
  νb.#data(
    νc.!elem(
      νd.#data(
        νd.#data(
          νd.#data(
            c.eof))),
      νb.#data(
        a.eof),
    eof)))

```

Under local freshness semantics, the data elements b in the **#data** fields of this term can be any names except a , similarly for d and c . This language is accepted by the RNTA with states $q_0, q_1(a), q_1(c)$ ($a, c \in \mathbb{A}$) and transitions

$$\begin{aligned}
q_0(\nu a.!\text{elem}(x_1, x_2)) &\rightarrow \nu a.!\text{elem}(q_1(a)(x_1), q_0(x_2)) \\
q_0(\nu b.\#\text{data}(x_1)) &\rightarrow \nu b.\#\text{data}(q_0(x_1)) & q_0(\text{eof}) &\rightarrow \text{eof} \\
q_1(a)(\nu c.!\text{elem}(x_1, x_2)) &\rightarrow \nu c.!\text{elem}(q_1(c)(x_1), q_1(a)(x_2)) \\
q_1(a)(\nu d.\#\text{data}(x_1)) &\rightarrow \nu d.\#\text{data}(q_1(a)(x_1)) & q_1(a)(a.\text{eof}) &\rightarrow a.\text{eof}.
\end{aligned}$$

Notice here that although every state stores at most one name, the automaton is able to track an unbounded number of **!elem** markers as it effectively creates copies of itself when reading an input tree.

► **Example 4.7** (π -Calculus expressions). We use $\Sigma = \{\text{par}/2, \text{rw}/1, \text{ch}/1, 0/0\}$, $\mathbb{A} = \{\perp, a, b, c, \dots\}$ to represent the syntax (only!) of a small fragment of the π -calculus, with *par* standing for parallel composition, and with *ch* and *rw* working in combination to represent writing or reading on a channel. Here, we model reading and allocation of channel names natively as name binding: $a.ch$ communicates on an existing channel a , and $\nu a.ch$ on a newly allocated channel a . Similarly, $a.rw$ writes a , while $\nu a.rw$ reads a value a . E.g., $\nu a.ch(\nu b.rw(x))$ reads b from a newly allocated channel a , and continues with x . Let L be the language of all Σ -terms that are parallel composites of $k \geq 1$ processes that each read a name b from a newly allocated channel a and then may, maybe repeatedly, read a new name from b and use that name as the input channel in the next round, before terminating (0). This language is accepted by the RNTA with states $q_0, q_1, q_2(a)$ ($a \in \mathbb{A}$) and transitions

$$\begin{aligned}
q_0(\text{par}(x, y)) &\rightarrow \text{par}(q_0(x), q_0(y)) & q_0(\nu a.ch(x)) &\rightarrow \nu a.ch(q_1(x)) \\
q_1(\nu a.rw(x)) &\rightarrow \nu a.rw(q_2(a)(x)) & q_2(a)(a.ch(x)) &\rightarrow a.ch(q_1(x)) \\
q_2(a)(0) &\rightarrow 0.
\end{aligned}$$

(Notice that the right hand transitions forget the channel name once the channel command has been processed; the name is no longer needed, as every channel is used only once.)

► **Remark 4.8.** In the paradigm of universal coalgebra [33], RNTAs may be viewed as coalgebras for the functor F given by

$$FX = \mathcal{P}_{\text{ufs}}(\sum_{f/n \in \Sigma} (\mathbb{A} \times X^n + [\mathbb{A}]X^n)). \quad (4.1)$$

5 Name Dropping

The key to the algorithmic tractability of name-allocating automata models in general [34, 40, 16] is to ensure that the literal language of an automaton is closed under α -equivalence, so that only boundedly many names need to be considered in inclusion checking. The problem to be overcome here is that this property does not hold in general, and needs to be enforced in a modification of the automaton that preserves the alphabetic language. Specifically, the problem comes about by extraneous names that do not occur in the remainder of a given word to be processed but do still occur in the relevant successor state, thus blocking the requisite α -renaming. As a simple example, when an automaton state q is processing $\nu a.f(t)$ for $f/1 \in \Sigma$ and we have a matching transition $q(\nu a.f(x)) \rightarrow \nu a.f(q'(x))$, then it may happen that $b \notin \text{FN}(t)$, so that a may be α -equivalently renamed into b in $\nu a.f(t)$, but $a \in \text{supp}(q')$ so that a cannot be α -equivalently renamed into b in $\nu a.f(q'(x))$. The solution to this is to extend the automaton by states that come about by dropping some of the names from the support of previous states [34]; in the example, a state q'' that has b removed from its support but otherwise behaves like q' will allow for the requisite α -renaming of the transition into $\nu a.f(q''(x))$, and will still be able to accept the remaining term t since $b \notin \text{FN}(t)$. We proceed to lay out the details of this construction, which we dub the *name-dropping modification*.

Following work on nominal Büchi automata [40], we first transform the automaton into one whose state set Q forms a *strong* nominal set; we do not need the original definition of strong nominal set [39] but instead use the equivalent description [30] of strong nominal sets as being those of the form $\sum_{i \in I} \mathbb{A}^{\#X_i}$ where the X_i are finite sets and $\mathbb{A}^{\#X_i}$ denotes the nominal set of total injective maps $X_i \rightarrow \mathbb{A}$. We generally write elements of sums like the above as pairs (i, r) , in this case consisting of $i \in I$ and $r \in \mathbb{A}^{\#X_i}$. Strong nominal sets thus materialize the intuition that the states of a nominal automaton consist of a control state (the index i in the above sum) and a store configuration assigning names to registers in a duplicate-free manner.

► **Lemma 5.1.** *For every RNTA A , there exists an RNTA A' whose states form a strong nominal set such that A and A' accept the same literal language.*

Our name-dropping modification will now come about by dropping the requirement that each register is necessarily occupied. This amounts to working with *partial* injective maps $r: X_i \rightarrow \mathbb{A}$, with undefinedness (denoted as $r(x) = \perp$) indicating that a register is currently empty. We first introduce notation for restricting such partial maps by dropping some of the names:

► **Definition 5.2.** Let X be a finite set. We write $\mathbb{A}^{\text{s}X}$ for the set of partial injective maps $X \rightarrow \mathbb{A}$. Let $r \in \mathbb{A}^{\text{s}X}$, and let $N \subseteq \text{supp}(r) = r[X]$. Then the partial injective map $r|_N \in \mathbb{A}^{\text{s}X}$ defined by $r|_N(x) = r(x)$ if $r(x) \in N$ and $r|_N(x) = \perp$ otherwise is the *restriction* of r to N (and r is an *extension* of $r|_N$).

► **Definition 5.3** (Name-dropping modification). Let $A = (Q, \Delta, q_0)$ be an RNTA such that $Q = \sum_{i \in I} \mathbb{A}^{\#X_i}$ is a strong nominal set. For $q = (i, r) \in Q$, we write $q|_N = (i, r|_N)$. Then the *name-dropping modification* of A is the RNTA $A_{\perp} = (Q_{\perp}, \Delta_{\perp}, q_0)$ where

1. $Q_{\perp} = \sum_{i \in I} \mathbb{A}^{\otimes X_i}$;
2. for all $q, q'_1, \dots, q'_n \in Q$, $N \subseteq \text{supp}(q)$, $N_i \subseteq \text{supp}(q'_i) \cap N$ ($i = 1, \dots, n$), and $a \in N$, whenever $q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q'_1(x_1), \dots, q'_n(x_n))$ in A , then $q|_N(a.f(x_1, \dots, x_n)) \rightarrow a.f(q'_1|_{N_1}(x_1), \dots, q'_n|_{N_n}(x_n))$ in A_{\perp} ; and
3. for all $q, q'_1, \dots, q'_n \in Q$, $N \subseteq \text{supp}(q)$, $a \in \mathbb{A}$, and $N_i \subseteq \text{supp}(q'_i) \cap (N \cup \{a\})$ ($i = 1, \dots, n$), whenever $q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q'_1(x_1), \dots, q'_n(x_n))$ in A and $\langle a \rangle q'_i|_{N_i} = \langle b \rangle q''_i$, $i = 1, \dots, n$, then $q|_N(\nu b.f(x_1, \dots, x_n)) \rightarrow \nu b.f(q''_1(x_1), \dots, q''_n(x_n))$ in A_{\perp} .

Notice that clauses defining the transition relation on A_{\perp} are only implications: A_{\perp} inherits transitions from A as long as these are consistent with Corollary 4.3, and bound transitions in A_{\perp} are subsequently closed under α -invariance. The arising transitions are characterized as follows.

► **Lemma 5.4.** *Let $A = (Q, \Delta, i)$ be an RNTA with Q strong, and let $A_{\perp} = (Q_{\perp}, \Delta_{\perp}, i)$ be its name-dropping modification.*

1. *If $q|_N(a.f(x_1, \dots, x_n)) \rightarrow a.f(q_1(x_1), \dots, q_n(x_n))$ in A_{\perp} for some $q, q_1, \dots, q_n \in Q$ and $N \subseteq \text{supp}(q)$, then each q_i has the form $q_i = q'_i|_{N_i}$ for some $q'_i \in Q$, $N_i \subseteq \text{supp}(q) \cap N$ such that $q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q'_1(x_1), \dots, q'_n(x_n))$ in A .*
2. *If $q|_N(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu b.f(q_1(x_1), \dots, q_n(x_n))$ in A_{\perp} for some $q, q_1, \dots, q_n \in Q$ and $N \subseteq \text{supp}(q)$, then for each q_i there is q'_i and $N_i \subseteq \text{supp}(q'_i) \cap (N \cup \{b\})$ such that $q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu b.f(q'_1(x_1), \dots, q'_n(x_n))$ in A and $\langle b \rangle q'_i|_{N_i} = \langle a \rangle q_i$.*

In both claims, the given conditions are, by definition, also sufficient; the key point of the lemma is that all transitions of a given state $q|_N$ come from transitions of q even when one also has $q|_N = q'|_N$ for a different q' .

The degree of the name-dropping modification remains the same because the new states arise by deleting names from the support of previous states; the number of orbits increases only by a factor 2^d , where d is the degree, because there are only 2^d ways to delete names from a support of size d . Moreover, one can show that as per the intention of the construction, the name dropping modification closes an RNTA under α -equivalence; summing up:

► **Theorem 5.5.** *For each RNTA A , the name-dropping modification A_{\perp} of A is an RNTA that accepts the closure of the literal tree language of A under α -equivalence, and hence the same alphabetic tree language as A . Moreover, A_{\perp} has the same degree d as A , and the number of orbits of A_{\perp} exceeds that of A by at most a factor 2^d .*

► **Remark 5.6 (Lossiness).** It is apparent from the construction of the name-dropping modification that, in the usual correspondence between nominal automata models and register-based models [4, 34], it establishes a lossiness property saying that during any transition, letters may nondeterministically be lost from the registers. Intuitively speaking, the effect of losing a letter from a register is on the one hand that one escapes freshness requirements against that letter in successor states, but on the other hand progress may later be blocked when the lost name is required to be seen in the word; the overall consequence of this phenomenon is that distinctness of the current letter b from a letter a seen previously in the word can only be enforced if a is expected to be seen again, as already illustrated in Examples 3.6 and 4.5.

6 Inclusion Checking

We conclude by showing that language inclusion of RNTAs is decidable in elementary complexity, in sharp contrast to the typical situation in register-based models as discussed in Section 1. The algorithm is based on reducing the problem to language inclusion of classical NFTAs over finite signatures (Section 2), using the name-dropping modification

to ensure closure of literal tree languages under α -equivalence (Section 5): Using closure under α -equivalence, we can restrict to a finite set of names that is large enough to represent every relevant α -equivalence class. Using this set of names, we cut out a finite part of the RNTA in which only the specified names appear. For these restricted automata, which are just NFTAs, we can decide language inclusion in EXPTIME. A key step in this programme is thus the following:

► **Definition 6.1.** *Given a finite set $S \subseteq \mathbb{A}$, we write $\mathcal{T}_S(\Sigma) = \{t \in \mathcal{T}_{\mathbb{A}}(\Sigma) \mid \text{supp}(t) \subseteq S\}$.*

(That is, a term is in $\mathcal{T}_S(\Sigma)$ if all its free and bound names are in S .)

► **Lemma 6.2.** *Let A be an RNTA of degree d_A , and let n_{ar} be the maximal arity of symbols in Σ . Pick $S \subseteq \mathbb{A}$ such that $|S| = d_A \cdot n_{ar} + 1$. If A accepts a term t , then A also accepts some term $t' \in \mathcal{T}_S(\Sigma)$ such that $t' \equiv_{\alpha} t$.*

(Recall that initial states have empty support by our running assumption; otherwise, S would also need to contain the support of the initial state.)

► **Theorem 6.3.** *Alphabetic tree language inclusion $L_{\alpha}(A) \subseteq L_{\alpha}(B)$ of RNTAs A, B of degrees d_A, d_B , respectively, over the fixed signature Σ is decidable in doubly exponential time, and in fact in parametrized singly exponential time with the degree as the parameter, i.e. exponential in a function that depends exponentially on $d_A + d_B$ and polynomially on the size of A, B .*

Here, we understand the size of A and B in terms of standard finitary representations of orbit-finite nominal sets, which essentially just enumerate the support sizes and symmetry groups of the orbits (e.g. [40]). In the complexity analysis, we assume the signature to be fixed; if the signature is made part of the input, then the parameter includes also the maximal arity n_{ar} of symbols in Σ as in Lemma 6.2.

Proof. The proof uses reduction to a finite-alphabet [34, 40]. Again, let n_{ar} be the maximal arity of symbols in Σ , and pick $S \subseteq \mathbb{A}$ such that $|S| = d_A \cdot n_{ar} + 1$ as required in Lemma 6.2. Put $\bar{S} = S \cup \{\nu a \mid a \in S\}$, and let B_{\perp} be the name-dropping modification of B as per Theorem 5.5. Let $(Q_A, \Delta_A, q_0^A) = A$ and $(Q_B, \Delta_B, q_0^B) = B$.

1. Show that $L_{\alpha}(A) \subseteq L_{\alpha}(B)$ iff $L(A) \cap \mathcal{T}_S(\Sigma) \subseteq L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$.
 - “ \Rightarrow ”: By Theorem 5.5, $L(B_{\perp})$ is the closure of $L(B)$ under α -equivalence. Thus, $L(A) \subseteq L(B_{\perp})$, and hence $L(A) \cap \mathcal{T}_S(\Sigma) \subseteq L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$.
 - “ \Leftarrow ”: Let $[t]_{\alpha} \in L_{\alpha}(A)$; we have to show that $[t]_{\alpha} \in L_{\alpha}(B)$. By definition of $L_{\alpha}(A)$, we have $t' \in L(A)$ such that $t' \equiv_{\alpha} t$, so by Lemma 6.2 there exists $t'' \in L(A) \cap \mathcal{T}_S(\Sigma)$ such that $t'' \equiv_{\alpha} t$. Then $t'' \in L(B_{\perp})$ by hypothesis, and hence $[t]_{\alpha} \in L_{\alpha}(B_{\perp})$. By Theorem 5.5, we obtain $[t]_{\alpha} \in L_{\alpha}(B)$ as required.
2. By 1, we are left to decide whether $L(A) \cap \mathcal{T}_S(\Sigma) \subseteq L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$. Observe that $L(A) \cap \mathcal{T}_S(\Sigma)$ and $L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$ are effectively just tree languages over the finite signature $\bar{S} \times \Sigma$. We construct top-down NFTAs A_S and B_S over $\bar{S} \times \Sigma$ that restrict A and B_{\perp} , respectively, to S and accept $L(A) \cap \mathcal{T}_S(\Sigma)$ and $L(B_{\perp}) \cap \mathcal{T}_S(\Sigma)$, respectively: Put $A_S = (Q_{A,S}, \Delta_{A,S}, q_0^A)$ where $Q_{A,S} = \{q \in Q_A \mid \text{supp}(q) \subseteq S\}$ and $\Delta_{A,S} = \{(q(\gamma.f(x_1, \dots, x_n) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))) \in \Delta_A \mid q, q_1, \dots, q_n \in Q_{A,S}, \gamma \in \bar{S})\}$. The construction of $B_S = (Q_{B_{\perp},S}, \Sigma \times \bar{S}, \Delta_{B_{\perp},S}, i_{B_{\perp}})$ is analogous. The automata A_S and B_S are finite because A and B_{\perp} are orbit-finite and each orbit of a nominal set contains only finitely many elements with a given finite support.

We verify that A_S accepts $L(A) \cap \mathcal{T}_S(\Sigma)$, i.e. $L(A_S) = L(A) \cap \mathcal{T}_S(\Sigma)$; the corresponding claim for B_{\perp} is analogous. Since all states and rewrite rules of A_S are inherited from A , it is immediate that $L(A_S) \subseteq L(A) \cap \mathcal{T}_S(\Sigma)$; we show the reverse inclusion. So let

$t \in L(A) \cap \mathcal{T}_S(\Sigma)$; we have to show that A_S accepts t . We show more generally that every state q of A_S accepts all terms $t \in \mathcal{T}_S(\Sigma)$ that q accepts in A , and proceed via induction on the length of an accepting run.

For the base case, let q accept $t = \gamma.c$ in A , where $\gamma \in \bar{S}$ because $t \in \mathcal{T}_S(\Sigma)$. That is, we have $\delta = (q(\gamma.c) \rightarrow \gamma.c) \in \Delta_A$. The $\delta \in \Delta_{A,S}$ by construction, so q accepts t in A_S .

For the inductive step, let q accept $t = \gamma.f(t_1, \dots, t_n)$ in A . Again, $\gamma \in \bar{S}$ because $t \in \mathcal{T}_S(\Sigma)$. Thus, we have $\delta = (q(\gamma.f(x_1, \dots, x_n)) \rightarrow \gamma.f(q_1(x_1), \dots, q_n(x_n))) \in \Delta_A$ such that q_i accepts t_i for $i = 1, \dots, n$. We distinguish between bound and free transitions:

- $\gamma = a$: By Lemma 4.2, $\text{supp}(q_i) \subseteq \text{supp}(q) \subseteq S$, so $q_i \in Q_S$, and by induction, q_i accepts t_i in A_S for $i = 1, \dots, n$. Since, $\gamma \in S$, we thus have $\delta \in \Delta_{A,S}$; it follows that q accepts t in A_S .
 - $\gamma = \nu a$: By Lemma 4.2, $\text{supp}(q_i) \subseteq \text{supp}(q) \cup \{a\}$. Since $a \in S$ and $\text{supp}(q) \subseteq S$, this implies $\text{supp}(q_i) \subseteq S$, i.e. $q_i \in Q_S$ for $i = 1, \dots, n$. By induction, q_i accepts t_i in A_S , and again, $\delta \in \Delta_{A,S}$ by construction because $\gamma \in S$, implying that q accepts t in A_S .
3. So far, we have reduced the problem to deciding language inclusion of NFTAa, which is in EXPTIME [9]; it remains to analyse the size of the NFTAs A_S, B_S constructed in step 2., where we have first constructed the name-dropping modification B_\perp of the RNTA B and have then extracted A_S and B_S from A and B_\perp , respectively, by restricting to the finite set S of names. We assume for simplicity that the state spaces of A and B are given as strong nominal sets, so that the size of A and B is essentially the respective number of orbits. When estimating the size of A_S and B_S , it suffices to consider the number of states, since the size of the signature $\bar{S} \times \Sigma$ is linear in d_A (as Σ is assumed to be fixed) so that the number of transitions of NFTAs over $\bar{S} \times \Sigma$ is polynomial in their number of states and d_A . It thus suffices to show that the number of states in A_S and B_S , respectively, is the number of orbits of A or B , respectively, multiplied by a factor that is singly exponential in the degree. Now by Theorem 5.5, the name-dropping modification step for B increases the number of orbits by an exponential factor in the degree d_B but leaves the degree itself unchanged. Moreover, we generally have that every orbit of a given nominal set with support size m has at most $m!$ elements with a given fixed support, so the step from A, B to A_S, B_S indeed incurs only an exponential factor in the degree, which proves the claim. \blacktriangleleft

From Lemma 3.7, it is immediate that the same complexity bound as in Theorem 6.3 holds also for inclusion checking of RNTAs under global and branchwise freshness semantics, respectively (i.e. for checking whether $\mathbf{N}(L_\alpha(A)) \subseteq \mathbf{N}(L_\alpha(B))$ or $\mathbf{B}(L_\alpha(A)) \subseteq \mathbf{B}(L_\alpha(B))$, respectively). We conclude by showing that this remains true under local freshness semantics. The following observation is key:

► **Definition 6.4.** We define an ordering \leq on $\bar{\mathbb{A}}$ by $a \leq \nu a$ for all $a \in \mathbb{A}$. We then define the ordering \sqsubseteq on $\mathcal{T}_{\bar{\mathbb{A}}}(\Sigma)$ recursively by $t \sqsubseteq s$ iff t, s have the form $t = \gamma.f(t_1, \dots, t_n)$ and $s = \delta.f(s_1, \dots, s_n)$ where $\gamma \leq \delta$ and $t_i \sqsubseteq s_i$ for $i = 1, \dots, n$. For a literal language L , $\downarrow L = \{t \in \mathcal{T}_{\bar{\mathbb{A}}}(\Sigma) \mid \exists t' \in L. t \sqsubseteq t'\}$ denotes the downward closure of L with respect to \sqsubseteq .

That is, $t \sqsubseteq t'$ if t arises from t' by removing zero or more occurrences of ν ; e.g. $\nu a.f(a.f(a.f(a.k))) \sqsubseteq \nu a.f(\nu a.f(a.k))$.

► **Lemma 6.5.** For closed alphatic languages L_1, L_2 , we have $D(L_1) \subseteq D(L_2)$ iff for all $[t] \in L_1$ there exists $t' \sqsubseteq t'$ such that $[t'] \in L_2$.

► **Theorem 6.6.** *Language inclusion $D(L_\alpha(A)) \subseteq D(L_\alpha(B))$ under local freshness semantics of RNTAs A, B of degrees d_A, d_B , respectively, over the fixed signature Σ is decidable in doubly exponential time, and in fact in parametrized singly exponential time with the degree as the parameter, i.e. exponential in a function that depends exponentially on $d_A + d_B$ and polynomially on the size of A, B .*

Proof. The proof is largely analogous to that of Theorem 6.3. In step 1., one shows using Lemma 6.5 (and recalling Remark 4.4) that $D(L_\alpha(A)) \subseteq D(L_\alpha(B))$ iff $L(A) \cap \mathcal{T}_S(\Sigma) \subseteq \downarrow(L(B_\perp) \cap \mathcal{T}_S(\Sigma))$. In step 2., the NFTA accepting $\downarrow(L(B_\perp) \cap \mathcal{T}_S(\Sigma))$ is constructed as in Theorem 6.3 and then closed downwards under \sqsubseteq by adding a transition $q(a.f(x_1, \dots, x_n)) \rightarrow a.f(q_1(x_1), \dots, q_n(x_n))$ for every transition $q(\nu a.f(x_1, \dots, x_n)) \rightarrow \nu a.f(q_1(x_1), \dots, q_n(x_n))$. ◀

7 Conclusions

We have introduced the model of *regular nominal tree automata (RNTA)*, a species of non-deterministic top-town nominal tree automata. RNTAs can be equipped with different data tree semantics ranging from global freshness as found in session automata [6] to local freshness. Under the latter, RNTAs correspond, via the usual equivalence of nominal automata and register-based automata [4, 34], to a subclass of register tree automata [21]. As such, they are less expressive than the full register model, but in return admit inclusion checking in elementary complexity (parametrized exponential time); this in a model that allows unboundedly many registers and unrestricted nondeterminism (cf. Section 1). RNTAs feature a native notion of name allocation, allowing them to process terms in languages with name binding such as the λ - and the π -calculus.

In future research, we aim to work towards a notion of nominal automata with name allocation for infinite trees, in particular with a view to applications in reasoning over name-allocating fragments of the nominal μ -calculus [23]. Also, it will be of interest to develop the theory in coalgebraic generality [33], aiming to support automata with effects such as probabilistic or weighted branching.

References

- 1 Jiří Adámek, Horst Herrlich, and George E. Strecker. *Abstract and Concrete Categories*. John Wiley and Sons, 1990. Reprint: <http://www.tac.mta.ca/tac/reprints/articles/17/tr17abs.html>.
- 2 Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.
- 3 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403.
- 4 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.
- 5 Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):13:1–13:48, 2009. doi:10.1145/1516512.1516515.
- 6 Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A robust class of data languages and an application to learning. *Log. Meth. Comput. Sci.*, 10(4), 2014. doi:10.2168/LMCS-10(4:19)2014.

- 7 Thomas Colcombet. Unambiguity in automata theory. In *Descriptive Complexity of Formal Systems, DCFS 2015*, volume 9118 of *LNCS*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3.
- 8 Thomas Colcombet and Amaldev Manuel. μ -calculus on data words. *CoRR*, 2014. arXiv:1404.4827.
- 9 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. HAL Portal Inria, 2008. hal-03367725. URL: <https://hal.inria.fr/hal-03367725/document>.
- 10 Wojciech Czerwiński and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. *CoRR*, 2021. arXiv:2104.13866.
- 11 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.
- 12 Diego Figueira. Forward-XPath and extended register automata on data-trees. In Luc Segoufin, editor, *Database Theory, ICDT 2010*, pages 231–241. ACM, 2010. doi:10.1145/1804669.1804699.
- 13 Murdoch James Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *Foundations of Software Science and Computation Structures, FOSSACS 2011*, volume 6604 of *LNCS*, pages 365–380. Springer, 2011. doi:10.1007/978-3-642-19805-2.
- 14 Radu Grigore, Dino Distefano, Rasmus Petersen, and Nikos Tzevelekos. Runtime verification based on register automata. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2013*, volume 7795 of *LNCS*, pages 260–276. Springer, 2013. doi:10.1007/978-3-642-36742-7_19.
- 15 Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. Model checking systems and specifications with parameterized atomic propositions. In *Automated Technology for Verification and Analysis, ATVA 2012*, volume 7561 of *LNCS*, pages 122–136. Springer, 2012. doi:10.1007/978-3-642-33386-6_11.
- 16 Daniel Hausmann, Stefan Milius, and Lutz Schröder. A linear-time nominal μ -calculus with name allocation. In Filippo Bonchi and Simon J. Puglisi, editors, *Mathematical Foundations of Computer Science, MFCS 2021*, volume 202 of *LIPICs*, pages 58:1–58:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.MFCS.2021.58.
- 17 Falk Howar, Bengt Jonsson, and Frits Vaandrager. Combining black-box and white-box techniques for learning register automata. In *Computing and Software Science – State of the Art and Perspectives*, volume 10000 of *LNCS*, pages 563–588. Springer, 2019. doi:10.1007/978-3-319-91908-9_26.
- 18 Marcin Jurdziński and Ranko Lazić. Alternation-free modal μ -calculus for data trees. In *Logic in Computer Science, LICS 2007*, pages 131–140. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.11.
- 19 Marcin Jurdziński and Ranko Lazić. Alternating automata on data trees and XPath satisfiability. *ACM Trans. Comput. Log.*, 12(3):19:1–19:21, 2011. doi:10.1145/1929954.1929956.
- 20 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.
- 21 Michael Kaminski and Tony Tan. Tree automata over infinite alphabets. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *LNCS*, pages 386–423. Springer, 2008. doi:10.1007/978-3-540-78127-1_21.
- 22 Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *Int. J. Found. Comput. Sci.*, 21(5):741–760, 2010. doi:10.1142/S0129054110007532.
- 23 Bartek Klin and Mateusz Łętyk. Scalar and vectorial μ -calculus with atoms. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:5)2019.
- 24 Klaas Kürtz, Ralf Küsters, and Thomas Wilke. Selecting theories and nonce generation for recursive protocols. In *Formal methods in security engineering, FMSE 2007*, pages 61–70. ACM, 2007. doi:10.1145/1314436.1314445.

- 25 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. *CoRR*, 2021. [arXiv:2104.12695](https://arxiv.org/abs/2104.12695).
- 26 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Logic in Computer Science, LICS 2019*, pages 1–13. IEEE, 2019. [doi:10.1109/LICS.2019.8785796](https://doi.org/10.1109/LICS.2019.8785796).
- 27 Amaldev Manuel, Anca Muscholl, and Gabriele Puppis. Walking on data words. *Theory Comput. Sys.*, 59(2):180–208, 2016. [doi:10.1007/s00224-014-9603-3](https://doi.org/10.1007/s00224-014-9603-3).
- 28 Antoine Mottet and Karin Quaas. The containment problem for unambiguous register automata. In *Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPICs*, pages 53:1–53:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. [doi:10.4230/LIPICs.STACS.2019.53](https://doi.org/10.4230/LIPICs.STACS.2019.53).
- 29 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004. [doi:10.1145/1013560.1013562](https://doi.org/10.1145/1013560.1013562).
- 30 Daniela Petrişan. *Investigations into Algebra and Topology over Nominal Sets*. PhD thesis, University of Leicester, 2011.
- 31 Andrew Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.
- 32 Simon Prucker and Lutz Schröder. Nominal tree automata with name allocation. *CoRR*, abs/2405.14272, 2024. [doi:10.48550/arXiv.2405.14272](https://doi.org/10.48550/arXiv.2405.14272).
- 33 Jan Rutten. Universal coalgebra: A theory of systems. *Theor. Comput. Sci.*, 249:3–80, 2000.
- 34 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. In *Foundations of Software Science and Computation Structures, FOSSACS 2017*, volume 10203 of *LNCS*, pages 124–142, 2017. [doi:10.1007/978-3-662-54458-7_8](https://doi.org/10.1007/978-3-662-54458-7_8).
- 35 Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, 19(3):424–437, 1990. [doi:10.1137/0219027](https://doi.org/10.1137/0219027).
- 36 Ryoma Senda, Yoshiaki Takata, and Hiroyuki Seki. Complexity results on register context-free grammars and register tree automata. In *Theoretical Aspects of Computing, ICTAC 2018*, volume 11187 of *LNCS*, pages 415–434. Springer, 2018. [doi:10.1007/978-3-030-02508-3_22](https://doi.org/10.1007/978-3-030-02508-3_22).
- 37 Tony Tan. Extending two-variable logic on data trees with order on data values and its automata. *ACM Trans. Comput. Log.*, 15(1):8:1–8:39, 2014. [doi:10.1145/2559945](https://doi.org/10.1145/2559945).
- 38 Szymon Torunczyk and Thomas Zeume. Register automata with extrema constraints, and an application to two-variable logic. *Log. Methods Comput. Sci.*, 18(1), 2022. [doi:10.46298/LMCS-18\(1:42\)2022](https://doi.org/10.46298/LMCS-18(1:42)2022).
- 39 Nikos Tzevelekos. *Nominal Game Semantics*. PhD thesis, University of Oxford, 2008.
- 40 Henning Urbat, Daniel Hausmann, Stefan Milius, and Lutz Schröder. Nominal büchi automata with name allocation. In Serge Haddad and Daniele Varacca, editors, *Concurrency Theory, CONCUR 2021*, volume 203 of *LIPICs*, pages 4:1–4:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. [doi:10.4230/LIPICs.CONCUR.2021.4](https://doi.org/10.4230/LIPICs.CONCUR.2021.4).
- 41 Gerco van Heerdt, Tobias Kappé, Jurriaan Rot, Matteo Sammartino, and Alexandra Silva. Tree automata as algebras: Minimisation and determinisation. In *Algebra and Coalgebra in Computer Science, CALCO 2019*, volume 139 of *LIPICs*, pages 6:1–6:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. [doi:10.4230/LIPICs.CALCO.2019.6](https://doi.org/10.4230/LIPICs.CALCO.2019.6).