Towards Nominal Context-Free Model-Checking ⋆

Pierpaolo Degano, Gian-Luigi Ferrari, and Gianluca Mezzetti

{degano,giangi,mezzetti}@di.unipi.it
Dipartimento di Informatica — Università di Pisa

A Philippe Darondeau, amico schivo e carissimo, P.

Abstract. Two kinds of automata are introduced, for recognising regular and context-free nominal languages. We compare their expressive power with that of analogous proposals in the literature. Some properties of our languages are proved, in particular that emptiness of a context-free nominal language L is decidable, and that the intersection of L with a regular nominal language is still context-free. This paves the way for model-checking systems against access control properties in the nominal case, which is our main objective.

Introduction

Languages over an infinite alphabet are receiving growing interest, see e.g. [3, 21]. They are used to formalise different aspects of many real computational systems, in particular of those concerning the usage of potentially unboundedly many resources that are dynamically created, accessed, and disposed. Examples can be found in XML schemas, in the Web (URLs), in security protocols (e.g. nonces and time-stamps), in virtualised resources of Cloud systems, in mobile and ubiquitous scenarios, etc [23, 8, 4]. Also foundational calculi for concurrent and distributed systems faced the similar problem of handling unboundedly many fresh (or restricted) names, under the term *nominal* languages [22, 5, 13, 17, 18].

The literature mainly reports on various kinds of regular languages over infinite alphabets and on their recognizers [16, 2, 7]. Also context-free languages over infinite alphabets have been investigated [9, 5, 19, 20]. Indeed, even the following simple example, as well as the recursive patterns of PCRE [15], shows that the patterns of dynamical resource usage have an intrinsic context-free nature.

```
let rec exec() =
    if(...)
      let socket = newsocketfromenv();
      send(socket);
      exec();
      release(socket);
    else ...
```

⋆ This work has been partially supported by the MIUR project *Security Horizons*, and by IST-FP7-FET open-IP project ASCENS

The ML-like script above is an abstract dispatcher of tasks on sockets. The execution environment yields a new socket that is *fresh*, so to guarantee exclusive access. Then a sequence of actions `exec` occurs (we omit them below), and eventually the socket is released. An example of a trace generated during a run is `new(s1)new(s2)new(s3)...release(s3)release(s2)release(s1)`. Now, forgetting the actions `new`, `release` and only keeping the names of the sockets (taken from an infinite alphabet), we get a word ww^R, where the symbols in w are all different. For the sake of simplicity, we omit below actions and we only consider resources; actions, hence data words are dealt with in Examples 2 and 7 showing that only simple extensions are required in the general case.

In the line of [10], we pursue here our investigation on developing a foundational model for nominal languages. Our main interest is in statically verifying nominal regular properties of systems, in particular safety properties. Systems are modelled as nominal context-free languages, and verification is done via model-checking. In this paper, we propose an effective model that characterises a novel class of nominal languages, including the one of [9] and expressing, e.g. the traces ww^R above — because of this ability in dealing with "balanced" parenthesis, we shall call these nominal languages *context-free*, and *regular* those that cannot. Additionally, our model can express traces ww^R where the restriction that the resources of w are different is relaxed at wish, yet keeping freshness. This makes the binding name-resource more flexible, in the spirit of dynamic allocation and garbage collection of variable-location typical of programming languages.

To make our model-checking procedure effective, we need to establish the conditions under which the emptiness of the intersection of a nominal context-free language with a nominal regular one is decidable. Preliminary to that is defining a group of nominal automata with increasing expressive power. Besides regular vs. context free, we also consider a disposal mechanism that supports "garbage collection" of symbols thus allowing to re-use disposed symbols, so having, e.g., the above nominal languages of words ww^R with/without replication of symbols.

We proved that regular nominal languages are not closed under complement and (full) concatenation, but they are under union and also under intersection, provided that symbol re-use is forbidden. In that case the language of *all* the strings over an infinite alphabet is however *not* regular, while it is in the general case. We also establish relations with proposals in the literature: without disposal our class of regular languages includes that of Usage Automata (UA) [2] and is incomparable with Variable Automata (VFA) [14] and Finite-memory Automata (FMA) [16]. With the disposal mechanism instead, VFA languages are included in our class, and we conjecture that the same holds for FMA.

Our class of context-free nominal languages is only closed under union and a restricted form of concatenation. Without disposal, ours are equivalent to Usages [2] and become more powerful than these and than *quasi context-free languages* (QCFL) [9] with the possibility of re-using disposed symbols.

As said above, our main goal is proving nominal properties of nominal models, in particular regular safety properties expressing secure access to resources [10]. Standard and efficient (automata-based) model-checking techniques require that

Math	
$i, j \in \underline{r} = \{i \mid 1 \leq i \leq r\}$	set of indices in $\mathbb{N}$, the natural numbers
$L \lneqq L'$	incomparable sets: $L \not\subseteq L'$ and $L \not\supseteq L'$
Words	
$\Sigma = \Sigma_s \cup \Sigma_d$,	alphabet with Σ_s *finite* set of *static* symbols and Σ_d disjoint
$(\{?, \top\} \cup \mathbb{N}) \cap \Sigma = \emptyset$	*countably infinite* set of *dynamic* symbols.
$a, b \in \Sigma$; $w \in \Sigma^*$	symbols in Σ and words, where ε is the empty string
$w[i], \lvert w \rvert, w^R, \lVert w \rVert$	i-th symbol, length, reverse, and set of symbols of w
Automata	
$q \in Q$	state of an automaton
$\sigma \in \Sigma_s \cup \underline{r} \cup \{\varepsilon, \top\}$	input symbol in a transition label
$Z \in \Sigma_s \cup \underline{r} \cup \{\varepsilon, ?\}$	stack read symbol in a transition label
$\zeta \in (\Sigma_s \cup \underline{r})^*$; $z \in \Sigma_s \cup \underline{r}$	stack write symbols in a transition label
$\Delta \in \{i+, i-\} \cup \{\varepsilon\}$	m-register update in a transition label
S	a stack, _ is the empty one
N, M	m-registers, _ is the empty one
$\lVert N \rVert$	set of (dynamic) symbols in N
C, ρ	a configuration, and a run $C_1 \to \cdots \to C_k$
R, A; $L(R), L(A)$	a FSNA,PSNA automaton and their language
$\mathcal{L}(\text{FSNA}), \mathcal{L}(\text{VFA}), \ldots$	set of languages accepted by FSNA, VFA, ... automata

Table 1. Notation

the emptiness of context-free nominal languages is decidable, and that the intersection of a property and a model is still context-free. We here consider the nominal languages accepted by automata without disposal, and prove that both requirements above hold. We also conjecture that intersecting a regular and a context-free language results in a context-free one, provided that at most one of their recognizers uses a disposal mechanism.

The paper is organised as follows. Notation and abbreviations are summarised in Tab. 1. For lack of space, we occasionally only give an intuition of why our results hold; all the proofs are in `www.di.unipi.it/~mezzetti`. Sect. 1 introduces Finite State Nominal Automata that express nominal properties richer than those considered in [10]; also, we compare these regular nominal languages (with and without disposal) with other proposals in the literature. Then Sect. 2 defines Pushdown Nominal Automata, studies some language theoretic properties and compares them with existing formalisms. Sect. 3 contains our main achievements: decidability of the emptiness of a context-free nominal language (without disposal) and feasibility of model-checking them against regular nominal properties.

1 Finite State Nominal Automata

Our automata accept languages over infinite alphabets, partitioned in a finite set of *static* symbols and an infinite set of *dynamic* symbols, representing resp. the resources known before program execution and those created at run-time.

An *m-register* is a pair $N = \langle x \in \{0,1\}, S\rangle$, where x is the *activation state* of the stack S

$\text{s-push}(a, \langle x, S\rangle) = \langle 1, push(a, S)\rangle$ $\text{s-top}(\langle 1, S\rangle) = top(S)$	$\text{s-pop}(\langle x, S\rangle) = \begin{cases} \langle 0, pop(S)\rangle \text{ if } S \neq \_ \\ \langle 0, S\rangle \text{ if } S = \_ \end{cases}$

Table 2. Operations s-push, s-pop, s-top on m-registers.

A finite nominal automaton is a *non-deterministic finite state automaton* (FSA) enriched with a fixed number of *mindful registers*. These registers are components of the configurations that store dynamic symbols, and are accessed and manipulated while recognizing a string. (See Tab. 1 for notation used below.)

Definition 1 (Finite State Nominal Automata).
A finite state nominal automaton *(FSNA) is* $R = \langle Q, q_0, \Sigma, \delta, r, F\rangle$ *where:*

- Q *is a finite set of states,* $q_0 \in Q$ $\quad -F \subseteq Q$ *is the set of final states*
- $\Sigma = \Sigma_s \cup \Sigma_d$ *is a partitioned alphabet (*Σ_s *is finite and* Σ_d *denumerable)*
- r *is the number of* mindful registers *(*m-registers *for short)*
- δ *is a relation between pairs* (q, σ) *and* (q', Δ)*, with* $\sigma \neq \top$*. For brevity, we write* $q \xrightarrow[\Delta]{\sigma} q' \in \delta$ *whenever* $(q, \sigma, q', \Delta) \in \delta$

A configuration is a triple $C = \langle q, w, [N_1, \ldots, N_r]\rangle$ *where* q *is the current state,* w *is the word to be read and* $[N_1, \ldots, N_r]$ *is an array of* r m-registers *with symbols in* Σ*. The configuration is* final *if* $\langle q_f \in F, \varepsilon, [N_1, \ldots, N_r]\rangle$*.*

An m-register N is actually a stack plus a tag, recording if it is either active or not. The operations on N are almost standard (see Tab. 2), but additionally N becomes inactive after a s-pop, while a s-push makes it active (note that s-popping an empty m-register results in a no-operation). The operation s-top yields a value only if the m-register is active, otherwise it is undefined, as well as when the m-register is empty.

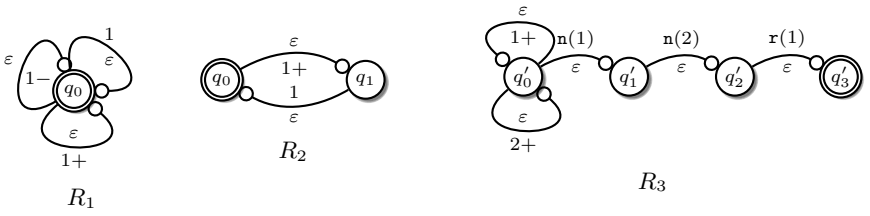
If $\sigma \in \Sigma_s$, just as in FSA, a transition $q \xrightarrow[\Delta]{\sigma} q'$ checks whether the current symbol in the input string is σ. Instead, if $\sigma \in \underline{\mathtt{r}}$, the symbol to be read is $\text{s-top}(N_\sigma)$, i.e. the top of the σ^{th} m-registers (analogously to [16]).

An m-register is at the same time updated according to Δ. There are three cases: $\Delta = \varepsilon$ then nothing has to be done; $\Delta = i+$ then a *fresh* dynamic symbol is s-pushed on the i-th m-register; $\Delta = i-$ then the i-th m-register is s-popped. A symbol is *fresh* if it does not appear in any of the r m-registers.

The application of a transition is detailed in the following definition:

Definition 2 (Recognizing Step).
Given an FSNA R*, a step* $\langle q, w, [N_1, \ldots, N_r]\rangle \to \langle q', w', [N'_1, \ldots, N'_r]\rangle$ *occurs iff there exists a transition* $q \xrightarrow[\Delta]{\sigma} q' \in \delta$ *s.t. both conditions hold:*

1. $\begin{cases} \sigma & = \varepsilon \Rightarrow w = w' \text{ and} \\ \sigma & = i \Rightarrow w = \text{s-top}(N_i)w' \text{ and} \\ \sigma & \in \Sigma_s \Rightarrow w = \sigma w' \end{cases}$



$\rho_1 = \langle q_0, aax, [\_]\rangle \xrightarrow{\varepsilon} \langle q_0, aax, [\boxed{\text{a}}]\rangle \xrightarrow{a} \langle q_0, ax[\boxed{\text{a}}]\rangle \xrightarrow{a} \langle q_0, x, [\_]\rangle \xrightarrow{\varepsilon} \langle q_0, x, [\boxed{\text{x}}]\rangle \xrightarrow{x} \langle q_0, \varepsilon, [\boxed{\text{x}}]\rangle$

$\rho_2 = \langle q_0, abc, [\_]\rangle \xrightarrow{\varepsilon} \langle q_1, abc, [\boxed{\text{a}}]\rangle \xrightarrow{a} \langle q_0, bc, [\boxed{\text{a}}]\rangle \xrightarrow{\varepsilon} \langle q_1, bc, [\begin{smallmatrix}\boxed{\text{b}}\\ \boxed{\text{a}}\end{smallmatrix}]\rangle \xrightarrow{b} \langle q_0, c, [\begin{smallmatrix}\boxed{\text{b}}\\ \boxed{\text{a}}\end{smallmatrix}]\rangle \xrightarrow{\varepsilon} \langle q_1, c, [\begin{smallmatrix}\boxed{\text{c}}\\ \boxed{\text{b}}\\ \boxed{\text{a}}\end{smallmatrix}]\rangle \xrightarrow{c} \langle q_0, \varepsilon, [\begin{smallmatrix}\boxed{\text{c}}\\ \boxed{\text{b}}\\ \boxed{\text{a}}\end{smallmatrix}]\rangle$

$\rho_3 = \langle q'_0, \mathtt{n}(a)\mathtt{n}(b)\mathtt{r}(a), [\_, \_]\rangle \xrightarrow{\varepsilon} \langle q'_0, \mathtt{n}(a)\mathtt{n}(b)\mathtt{r}(a), [\boxed{\text{a}}, \_]\rangle \xrightarrow{\varepsilon} \langle q'_0, \mathtt{n}(a)\mathtt{n}(b)\mathtt{r}(a), [\boxed{\text{a}}, \boxed{\text{b}}]\rangle \xrightarrow{\mathtt{n}(a)} \langle q'_1, \mathtt{n}(b)\mathtt{r}(a), [\boxed{\text{a}}, \boxed{\text{b}},]\rangle \xrightarrow{\mathtt{n}(b)} \langle q'_2, \mathtt{r}(a), [\boxed{\text{a}}, \boxed{\text{b}}]\rangle \xrightarrow{\mathtt{r}(a)} \langle q'_3, \varepsilon, [\boxed{\text{a}}, \boxed{\text{b}}]\rangle$

Fig. 1. Three examples of FSNA R_i and of their runs ρ_i. The automaton R_1 accepts Σ^*; note that the dynamic symbol x can be any symbol in Σ_d, even a, because the m-register is empty when x is s-pushed and there is no restriction on its freshness. The automaton R_2 accepts L_0 in Ex. 4; and R_3 accepts strings aba ($a \neq b$).

$$2.\ \begin{cases} \Delta & = i+ \Rightarrow N'_i = \textit{s-push}(b, N_i) \wedge \forall j.b \notin \|N_j\| \wedge \forall j\ (j \neq i).N_j = N'_j \textit{ and} \\ \Delta & = i- \Rightarrow N'_i = \textit{s-pop}(N_i) \wedge \forall j\ (j \neq i).N_j = N'_j \textit{ and} \\ \Delta & = \varepsilon \Rightarrow \forall j.N_j = N'_j \end{cases}$$

Finally, the language accepted by R is

$$L(R) = \{w \in \Sigma^* \mid \langle q_0, w, [\_, \ldots, \_]\rangle \to^* C_k, \textit{with } C_k \textit{ final}\}$$

Some examples follow.

Example 1. The FSNA R_1 in Fig. 1 non-deterministically accepts $\Sigma^* \in \mathcal{L}(\text{FSNA})$.

Example 2. The FSNA R_3 in Fig. 1 on data words represents a property for the `new-release` traces of the introduction, in the default-accept paradigm [1]. (The symbols $\mathtt{new}(a), \mathtt{release}(a)$ are abbreviated by $\mathtt{n}(a), \mathtt{r}(a)$.) The automaton accepts the unwanted traces where a second socket is created ($\mathtt{n}(2)$) before having released the last one created ($\mathtt{r}(1)$). Dealing with data words, σ assumes the form $\mathtt{n}(u), \mathtt{r}(u), u \in \underline{\mathtt{r}} \cup \Sigma_s$. For readability, we only mention the sockets in the m-registers, omitting the actions $\mathtt{n}, \mathtt{r}$.

Example 3. Let $L_r = \{ww^R \in \Sigma_d^* \mid |w| = r \text{ and } \forall i, j.\, w[i] \neq w[j]\}$ then no FSNA R with less that r states and r m-registers accepts L_r. Indeed, a standard argument on FSA proves that r states are required. Assume now that R has less than r registers and accepts ww^R. By the pigeonhole principle, there is at least one symbol of w, say a, s.t. $\forall i.\ a \neq \text{s-top}(N_i)$ when w has been read. Since $a \in \|w\|$, a needs to be s-pushed while traversing w^R, but it is fresh so it can be

replaced by any other (fresh) different symbol, which makes R to accept also ww', where $w' \neq w^R$: contradiction.

We now define a sub-class of FSNA where no transitions can s-pop any m-registers.

Definition 3 (FSNA$_+$). *An FSNA$_+$ is a FSNA with no edge* $q \xrightarrow[i-]{\sigma} q'$.

Example 4. The FSNA$_+$ R_2 in Fig. 1 accepts $L_0 = \{w \in \Sigma_d^* \mid \forall i \neq j.\, w[i] \neq w[j]\}$.

Example 5. Σ^* is not accepted by any FSNA$_+$, as Σ_d is infinite. Indeed, if there is one with r m-registers, accepting ww with $|w| = |\|w\|| = r + 1$, the word ww is not accepted if $a \in \|w\|$ and, after having read w, $\forall i.\ a \neq \text{s-top}(N_i)$.

The two kinds of automata above enjoy a few closure properties with respect to standard language operations. We conjecture that FSNA are also closed under intersection; concatenation also preserves regularity, if the two languages do not share any dynamic symbol.

Theorem 1 (Closures).

	$\cup$	$\cap$	$\overline{\cdot}$	$\bullet$	$*$
FSNA	✓	*?*	×	✓	✓
FSNA$_+$	✓	✓	×	×	×

We now compare the expressive power of FSNA and of FSNA$_+$ with that of other models for regular nominal languages, namely VFA [14], FMA [16] and UA [2].

Theorem 2 (Comparison).

- $\mathcal{L}(FSNA) \supset \mathcal{L}(VFA) \supset \mathcal{L}(UA)$
- $\mathcal{L}(FSNA) \supset \mathcal{L}(FMA)$
- $\mathcal{L}(FSNA_+) \supset \mathcal{L}(UA)$
- $\mathcal{L}(FSNA_+) \not\subseteq \mathcal{L}(VFA)$
- $\mathcal{L}(FSNA_+) \not\subseteq \mathcal{L}(FMA)$

2 Pushdown Nominal Automata

We presented in the introduction a simple program showing a behaviour that is conveniently abstracted as a context-free nominal language. This is often the case when defining static analysis, typically type and effect systems, through which proving program properties, e.g. on resource usage [2]. Nominal context-free languages have therefore both a theoretical and a practical relevance, and received some attention [9, 6, 5, 19]. Below, we extend FSNA with a stack, so obtaining Pushdown Nominal Automata: nominal context-free automata.

Definition 4 (Pushdown Nominal Automata).
A Pushdown Nominal Automata *(PSNA) is* $A = \langle Q, q_0, \Sigma, \delta, r, F\rangle$ *where:*

- Q, q_0, r, F *are as in FSNA (Def. 1)*
- δ *is a relation between triples* (q, σ, Z) *and* (q', Δ, ζ)*, with* $(q, \sigma, Z, q', \Delta, \zeta) \in \delta$ *written* $q \xrightarrow[\Delta,\zeta]{\sigma,Z} q' \in \delta$

Pushreg is an operation taking ζ and S and returning a new stack as follows:

$$\begin{array}{l} Pushreg(z\,\zeta', S) = Pushreg(\zeta', push(\sigma, S)) \\ Pushreg(\varepsilon, S) = S \end{array} \qquad \text{where } \sigma = \begin{cases} z & \text{if } z \in \Sigma_s \\ \text{s-top}(N_z) & \text{if } z \in \underline{\mathtt{r}} \end{cases}$$

Table 3. The *Pushreg* operation

A configuration is a tuple $C = \langle q, w, [N_1, \ldots, N_r], S\rangle$ *where* $q, w, [N_1, \ldots, N_r]$ *are as in FSNA and* S *is a stack with symbols in* Σ.
The configuration $\langle q, w, [N_1, \ldots, N_r], S\rangle$ *is* final *if* $q \in F, w = \varepsilon$ *and* $S = \_$.

Since the label ζ of a transition refers to m-registers, the *push* on the stack S requires an operation (illustrated in Tab. 3) able to also push the s-top of the mentioned m-registers. E.g., $\zeta = a\,3\,b$ causes the string $a\,\text{s-top}(N_3)\,b$ to be pushed.

Definition 5 (Recognizing Step).
Given a PSNA A, *the step* $\langle q, w, [N_1, \ldots, N_r], S\rangle \to \langle q', w', [N'_1, \ldots, N'_r], S'\rangle$ *occurs iff* $q \xrightarrow[\Delta,\zeta]{\sigma,Z} q' \in \delta$ *and the following hold*

1. *condition 1 of Def. 2 and* $\sigma = \top \Rightarrow w = top(S)w'$ *and*
2. *condition 2 of Def. 2 and*
3. $\begin{cases} Z = \varepsilon \Rightarrow S' = Pushreg(\zeta, S) \text{ and} \\ Z = i \Rightarrow S' = Pushreg(\zeta, pop(S)) \wedge top(S) = s\text{-}top(N_i) \text{ and} \\ Z = ? \Rightarrow S' = Pushreg(\zeta, pop(S)) \text{ and} \end{cases}$

Finally, the language accepted by R *is*
$L(A) = \{w \in \Sigma^* \mid \langle q_0, w, [\_, \ldots, \_], \_\rangle \to^* C_k, \text{with } C_k \text{ final}\}$

The definition above extends that for FSNA in handling the stack. In item (1) we add the possibility of checking if the current symbol equals the top of the stack, written $\top$. The top of the stack, say a, is popped if $Z = ?$, as well as if $Z = i$, provided that the s-top of the i^{th} m-register equals a. Finally, if $Z = \varepsilon$ the only action is pushing the string obtained from ζ, as explained above.

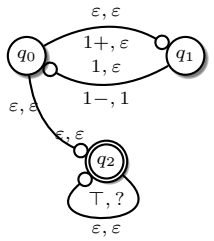
Example 6. Fig. 2(a) shows a PSNA accepting $L_p = \{ww^R \mid w \in \Sigma_d^*\}$, and a run accepting *aabbaa* (for brevity, configurations are without strings, the symbols read being in the labels of steps).

As done for FSNA we present a sub-class of PSNA without delete transitions.

Definition 6 (PSNA$_+$). *A PSNA*$_+$ *is a PSNA with no edges* $q \xrightarrow[i-,\zeta]{\sigma,Z} q'$.

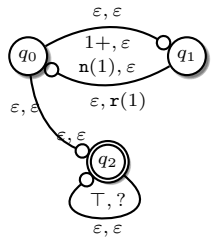
Example 7. Consider again the `new-release` (abbreviated $\mathtt{n}, \mathtt{r}$) language on data words of the introduction, with all sockets different. The PSNA$_+$ accepting this language is in Fig. 2(b). The labels of transitions, but Δ, contain $\mathtt{n}(u), \mathtt{r}(u), u \in \underline{\mathtt{r}} \cup \Sigma_s$. Fig. 2(b) also shows the run for $\mathtt{n}(a)\,\mathtt{n}(b)\,\mathtt{r}(b)\,\mathtt{r}(a)$; here we omit the strings in configurations and we only mention the sockets in the m-registers. Note that, only keeping the names of the sockets, we get $\cup_{r\in\mathbb{N}} L_r$, for L_r of Ex. 3.

(a)



$\langle q_0, [\_], \_\rangle \xrightarrow{\varepsilon} \langle q_1\, [\boxed{\text{a}}], \_\rangle \xrightarrow{a} \langle q_0, [\_], \boxed{\text{a}}\rangle \xrightarrow{\varepsilon} \langle q_1, [\boxed{\text{a}}], \boxed{\text{a}}\rangle \xrightarrow{a}$
$\langle q_0, [\_], \boxed{\begin{matrix}\text{a}\\\text{a}\end{matrix}}\rangle \xrightarrow{\varepsilon} \langle q_1, [\boxed{\text{b}}], \boxed{\begin{matrix}\text{a}\\\text{a}\end{matrix}}\rangle \xrightarrow{b} \langle q_0, [\_], \boxed{\begin{matrix}\text{b}\\\text{a}\\\text{a}\end{matrix}}\rangle \xrightarrow{\varepsilon}$
$\langle q_2, [\_], \boxed{\begin{matrix}\text{b}\\\text{a}\\\text{a}\end{matrix}}\rangle \xrightarrow{b} \langle q_2, [\_], \boxed{\begin{matrix}\text{a}\\\text{a}\end{matrix}}\rangle \xrightarrow{a} \langle q_2, [\_], \boxed{\text{a}}\rangle \xrightarrow{a} \langle q_2, [\_], \_\rangle$

(b)



$\langle q_0, [\_], \_\rangle \xrightarrow{\varepsilon} \langle q_1\, [\boxed{\text{a}}], \_\rangle \xrightarrow{\mathtt{n}(a)} \langle q_0, [\boxed{\text{a}}], \boxed{\mathtt{r}(\text{a})}\rangle \xrightarrow{\varepsilon}$
$\langle q_1, [\boxed{\begin{matrix}\text{b}\\\text{a}\end{matrix}}], \boxed{\mathtt{r}(\text{a})}\rangle \xrightarrow{\mathtt{n}(b)} \langle q_0, [\boxed{\begin{matrix}\text{b}\\\text{a}\end{matrix}}], \boxed{\begin{matrix}\mathtt{r}(\text{b})\\\mathtt{r}(\text{a})\end{matrix}}\rangle \xrightarrow{\varepsilon} \langle q_2, [\boxed{\begin{matrix}\text{b}\\\text{a}\end{matrix}}], \boxed{\begin{matrix}\mathtt{r}(\text{b})\\\mathtt{r}(\text{a})\end{matrix}}\rangle \xrightarrow{\mathtt{r}(b)}$
$\langle q_2, [\boxed{\begin{matrix}\text{b}\\\text{a}\end{matrix}}], \boxed{\mathtt{r}(\text{a})}\rangle \xrightarrow{\mathtt{r}(a)} \langle q_2, [\boxed{\begin{matrix}\text{b}\\\text{a}\end{matrix}}], \_\rangle$

Fig. 2. (a) A PSNA accepting $\{ww^R \mid w \in \Sigma_d^*\}$, and a run on *aabbaa*. (b) A PSNA$_+$ for the data word language of the introduction and a run on $\mathtt{n}(a)\,\mathtt{n}(b)\,\mathtt{r}(b)\,\mathtt{r}(a)$ (`n` and `r` stand for `new` and `release`). Strings are omitted in configurations.

A few properties of closure with respect to language operations follow.

Theorem 3 (Closures).

	$\cup$	$\cap$	$\overline{\cdot}$	$\bullet$	$*$
PSNA	✓	×	×	✓	✓
PSNA$_+$	✓	×	×	×	×

We now relate our proposal to *quasi context-free languages* (QCFL) [9] and Usages [2], that are an automata-like and a process calculi-like models for nominal context-free languages, respectively.

Theorem 4 (Comparison).

- $\mathcal{L}(PSNA) \supset \mathcal{L}(QCFL)$
- $\mathcal{L}(PSNA_+) \not\gtrless \mathcal{L}(QCFL)$
- $\mathcal{L}(PSNA) \supset \mathcal{L}(Usages)$
- $\mathcal{L}(PSNA_+) = \mathcal{L}(Usages)$

3 Towards Nominal Model Checking

This section studies the feasibility of model-checking models expressed by nominal context-free languages A against regular nominal properties R, similarly to, e.g., [8, 12, 2, 11]. We give a first positive answer to this problem, which is a main result of this paper.

The standard *automata based model-checking* procedure [24] requires to verify the emptiness of the intersection of the two languages: $A \cap \overline{R} = \emptyset$. A first point seems to arise in our case, because $\overline{R}$ is not necessarily regular, by Thm. 1.

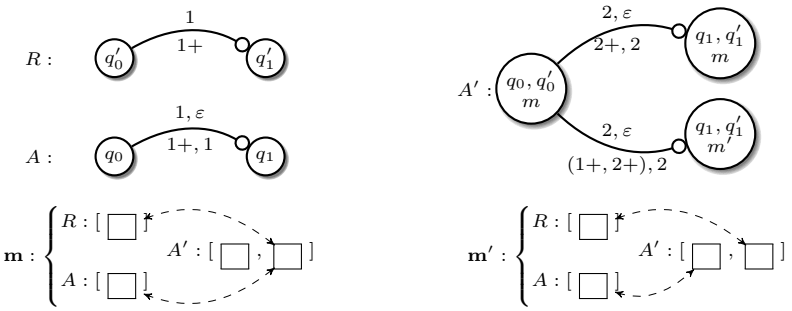


Fig. 3. A' is a portion of the automaton recognizing the intersection of the languages of R and A, the diagrams at the bottom represent the *merges* m, m'. Note that we are using the equivalent version of automata that allows for updating two m-registers at the same time.

Luckily, this is not a problem, since our main concern is now verifying access control policies expressed within the *default-accept* paradigm, where *unwanted* traces are defined, rather than those obeying the required properties [1]. We are therefore left to prove that the intersection above is still a context-free language and that its emptiness is decidable. Indeed, both properties hold for our classes of languages, when their recognizing automata have no transitions that s-pop any m-registers.

Our first theorem states that the emptiness problem is decidable for PSNA_+.

Theorem 5. *Given a PSNA_+ A, it is decidable whether $L(A) = \emptyset$.*

Proof. (Sketch) The proof relies on a restricted form of the pumping lemma: roughly, there exists a constant n depending on A, s.t. any string $w \in L(A), |w| > n$ can be decomposed into $w = uvxyz$, s.t. also $w = u'x'z'$ belongs to $L(A)$ with u', x', z' obtained from u, x, z by carefully substituting (distinguished) dynamic symbols. By repeatedly applying this kind of pumping lemma, $L(A)$ is non empty if it contains a word w', made of distinguished symbols, and s.t. $|w'| \leq n$.

Our next theorem guarantees that model-checking is feasible in our case: PSNA_+ are closed under intersection with FSNA_+.

Theorem 6. *Let A be a PSNA_+ and R be a FSNA_+. Then, $L(A) \cap L(R)$ is a PSNA_+ A'.*

Lack of space prevents us from detailing the proof, and we only present below some ingredients of the above sketched construction, ignoring a few mild conditions and assuming all the registers to be active in the initial configuration. To illustrate our construction, consider the automata A and R in Fig. 3 and the portion of the PSNA_+ A' accepting their intersection. A' is obtained by the standard construction that builds the new states as the product of the old ones. Additionally, each pair $\langle q, q' \rangle$ is enriched with a *merge* function m. Intuitively, m

describes how the m-registers of the two automata are mapped into those of A' — as a matter of fact, we shall use a variant of our automata (called PSNA_{+2}), the transitions of which can update two m-registers at the same time; the equivalence of the extended automata with the ones used so far is easily shown by rendering the extended transition $q \xrightarrow[\Delta_1,\Delta_2,\zeta]{\sigma,Z} q'$ updating the m-registers Δ_1 and Δ_2 with two sequentialized "standard" transitions.

The idea underlying the definition of a merge m is to guarantee the following invariant $\mathcal{I}$ along the runs: if A and R are in configurations $\langle q_0, w, [\![\boxed{\mathrm{y}}]\!], S\rangle$ and $\langle q_0', w, [\![\boxed{\mathrm{x}}]\!]\rangle$ then A' will be in configuration $\langle\langle q_0, q_0', m\rangle, w, [\![\boxed{\mathrm{h}}], \boxed{\mathrm{z}}]\!], S'\rangle$ and if two m-registers have the same s-tops then they are merged by m (and vice versa). This is illustrated in the three left-most configurations of Fig. 4: if $x = y = a$ then m maps the two registers to one register of A' (here the second one), and $z = a$. The edges of the automaton are also defined in the standard way. However, the m-registers mentioned in σ, Δ, ζ, Z of A' are those merged by m, provided that R and A agree on both σ and Δ under m.

Consider again Fig. 3. The transition $t : \langle\langle q_0, q_0'\rangle, m\rangle \xrightarrow[2+,2]{2,\epsilon} \langle\langle q_1, q_1'\rangle, m\rangle$ is present because there are $q_0' \xrightarrow[1+]{1} q_1'$ and $q_0 \xrightarrow[1+,1]{1,\epsilon} q_1$ and m maps the first m-register of R and that of A to the second of A'. Instead, the state $\langle\langle q_0, q_0'\rangle, m'\rangle$ (omitted in the figure) has no outgoing edges, because the symbols read by R and A are kept apart by m'.

There are transitions that only differ for the merge function in their target state. Not all the possible merges can however be taken, but only those "compatible" with the update Δ, in order to keep the invariant mentioned above. For example, the transition $\langle\langle q_0, q_0'\rangle, m\rangle \xrightarrow[(1+,2+),2]{2,\epsilon} \langle\langle q_1, q_1'\rangle, m'\rangle$ permits the recognizing step $C \xrightarrow{a} C'$, where the m-register of R now has got a d, while that of A has got c, and m' keeps them apart. Instead, if both m-registers store the same dynamic symbol c, the merge is still m, and the transition t above enables the step $C \xrightarrow{a} C''$ and guarantees the invariant.

Definition 7 (Merge function). *The function $m : \{1,2\} \times \underline{\mathtt{r}} \to \underline{2\mathtt{r}}$ is a* merge *iff $m_1(x) = m(1,x), m_2(x) = m(2,x)$ are injective. The following are shorthands:*
- two m-registers i, j are merged *($i \overset{\mathtt{m}}{\longleftrightarrow} j$) iff $m_1(i) = m_2(j)$, or are* taken apart *($i \overset{\mathtt{m}}{\not\longleftrightarrow} j$), otherwise;*
- $m(\zeta), m(Z)$ homomorphically apply m to ζ, Z, leaving untouched $\sigma \in \Sigma_s \cup \{?\}$;
- $m[N_1^1, \ldots, N_r^1, N_1^2, \ldots, N_r^2] = [M_1, \ldots, M_{2r}]$, iff the s-tops of the merged m-registers match and additionally $\bigcup_{i\in\underline{\mathtt{r}}} \|N_i^1\| \cup \bigcup_{i\in\underline{\mathtt{r}}} \|N_i^2\| = \bigcup_{i\in\underline{2\mathtt{r}}} \|M_i\|$.

A further notion is in order. As shown in Fig. 4, from C one reaches both C' and C'', the states of which however have different merge functions (m' and m, resp.). To account for that, we explicitly represent the changes made by the transition in the registers of the automata to be intersected, call them Δ_1, Δ_2. We then say that a merge m *evolves* into m' ($m \overset{\Delta_1,\Delta_2}{\frown} m'$), provided that m' only differs on the updated m-registers Δ_1, Δ_2, possibly merging either of them with

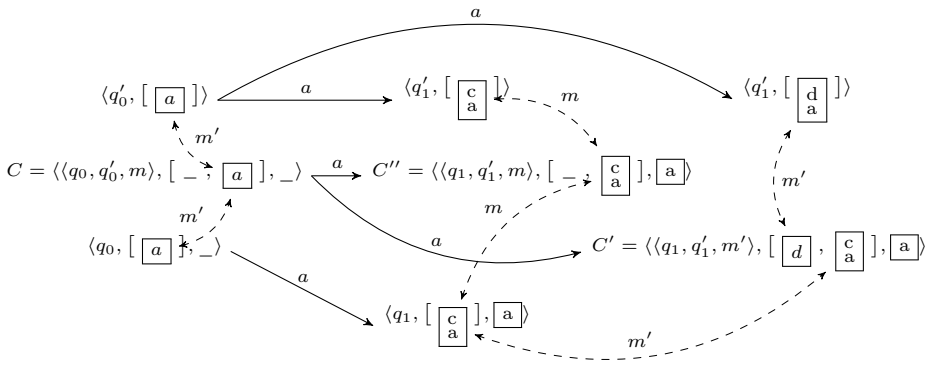


Fig. 4. Two recognizing steps of A' (middle), built from steps of R (top), and steps of A (bottom) (see Fig. 3). The step $C \xrightarrow{a} C'$ simultaneously updates two m-registers.

other m-registers by m' or taking them apart from some to which they were associated with by m. The registers updated in the intersection automata are actually computed by the suitable function $\overset{m}{m'}(\Delta_1, \Delta_2)$, the definition of which we omit. It guarantees that the invariant $\mathcal{I}$ discussed above, and the compatibility between the actual content of m-registers and m'.

The intersection of PSNA_+ with FSNA_+ is given as follows:

Definition 8 (Intersection Automaton).

Given the PSNA$_+$ $\langle Q_1, q_0^1, \Sigma, \delta_1, r, F_1\rangle$ and the FSNA$_+$ $\langle Q_2, q_0^2, \Sigma, \delta_2, r, F_2\rangle$, their intersection automaton *(of type PSNA$_{+2}$) is $\langle \overline{Q}, \overline{q_0}, \Sigma, \overline{\delta}, 2r, \overline{F}\rangle$, where*

- $\overline{Q} = Q_1 \times Q_2 \times M$, *with M set of merge functions*
- $\overline{q_0} = \langle q_0^1, q_0^2, \langle id_{\underline{\mathtt{r}}}, id_{\underline{\mathtt{r}}}\rangle\rangle$ $\quad$ $\overline{F} = \{\langle q_1, q_2, m\rangle \mid q_1 \in F_1, q_2 \in F_2, m \in M\}$
- $\langle q_1, q_2, m\rangle \xrightarrow[\overline{\Delta_1}, \overline{\Delta_2}, \overline{\zeta}]{\overline{\sigma}, \overline{Z}} \langle q_1', q_2', m'\rangle \in \overline{\delta}$ *iff* $m \overset{\Delta_1, \Delta_2}{\curvearrowright} m'$ *and*

  $q_1 \xrightarrow[\Delta_1, \zeta]{\sigma_1, Z} q_1' \in \delta_1$ *and* $q_2 \xrightarrow[\Delta_2]{\sigma_2} q_2' \in \delta_2$ *and* $(\sigma_1, \sigma_2 \in \underline{\mathtt{r}}$ *or* $\sigma_1, \sigma_2 \in \Sigma_s)$ *and*
  - *if* $\sigma_1, \sigma_2 \in \underline{\mathtt{r}}$ *then* $\overline{\sigma} = m_1(\sigma_1) = m_2(\sigma_2)$ *and*
  - *if* $\sigma_1, \sigma_2 \in \Sigma_s$ *then* $\overline{\sigma} = \sigma_1 = \sigma_2$ *and*
  - $(\overline{\Delta_1}, \overline{\Delta_2}) = \overset{m}{m'}(\Delta_1, \Delta_2)$, $\overline{Z} = m_1(Z), \overline{\zeta} = m_1(\zeta)$

  or $q_1 \xrightarrow[\Delta_1, \zeta]{\top, Z} q_1' \in \delta_1$ *and* $q_2 \xrightarrow[\Delta_2]{\sigma_2} q_2' \in \delta_2$ *and* $\sigma_2 \in \underline{\mathtt{r}}, \overline{\sigma} = m_2(\sigma_2)$ *and* $\overline{\zeta} = m_1(\zeta)$

  and either $Z = k \in \underline{\mathtt{r}}$ *implies* $k \overset{m}{\longleftrightarrow} \sigma_2$, $\overline{Z} = m_2(\sigma_2), (\overline{\Delta_1}, \overline{\Delta_2}) = \overset{m}{m'}(\Delta_1, \Delta_2)$,

  or $Z =?$ *implies* $\overline{Z} = m_2(\sigma_2), (\overline{\Delta_1}, \overline{\Delta_2}) = \overset{m}{m'}(\Delta_1, \Delta_2)$

  or $\quad q_1 \xrightarrow[\Delta_1 \zeta]{\varepsilon, Z} q_1' \in \delta_1$ *and* $\overline{\sigma} = \varepsilon$, $(\overline{\Delta_1}, \overline{\Delta_2}) = \overset{m}{m'}(\Delta_1, \varepsilon)$, $\overline{Z} = m_1(Z), \overline{\zeta} = m_1(\zeta)$

  or $\quad q_2 \xrightarrow[\Delta_2]{\varepsilon} q_2' \in \delta_2$ $\overline{\sigma} = \varepsilon$, $(\overline{\Delta_1}, \overline{\Delta_2}) = \overset{m}{m'}(\Delta_1, \varepsilon)$, $\overline{Z} = \varepsilon, \overline{\zeta} = \varepsilon$

4 Conclusions

We introduced novel kinds of automata that recognise new classes of regular and of context-free nominal languages. We studied their closure properties and we related their expressive power to that of the models in the literature. A main result of ours is that the problem of checking a model expressed by a nominal context-free language against a regular nominal property is decidable, under mild assumptions. Our contribution addresses therefore the shortcoming of standard automata based model-checking approaches in the nominal setting, that only considered regular languages. Ours is a further step towards developing methods for formally verifying computational systems that handle an unbounded number of resources. This is more and more the case nowadays, e.g. XML schemas, web and cloud system, security protocols.

Further investigation is required on the impact that a disposal mechanism has on the feasibility of model-checking. Preliminary results suggest us that intersecting a regular and a context-free language results in a context-free one, when at most one of them has a disposal mechanism.

Future research is needed to fill in the gap between our foundational results and the concrete case studies and prototypal implementation of our abstract model-checking procedure. Another line of research will study a logical characterization of our nominal regular automata, as done e.g. by [7, 11].

References

1. Bartoletti, M., Degano, P., Ferrari, G.L.: Planning and verifying service composition. JCS 17(5), 799–837 (2009)
2. Bartoletti, M., Degano, P., Ferrari, G.L., Zunino, R.: Model checking usage policies To appear in *Math. Stuct. Comp. Sci.*, abridged version in TGC 2008, vol 5474 LNCS (2009)
3. Benedikt, M., Ley, C., Puppis, G.: Automata vs. logics on data words. In: Dawar, A., Veith, H. (eds.) CSL. LNCS, vol. 6247, pp. 110–124. Springer (2010)
4. Bojanczyk, M.: Data monoids. In: Dürr, C., Wilke, T. (eds.) STACS 2011. vol. 9, pp. 105–116. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2011)
5. Bojańczyk, M., Klin, B., Lasota, S.: Automata theory in nominal sets (2011), `http://www.mimuw.edu.pl/~sl/PAPERS/lics11full.pdf`
6. Bojanczyk, M., Klin, B., Lasota, S.: Automata with group actions. pp. 355–364. LICS, IEEE Computer Society, Washington, DC, USA (2011)
7. Bollig, B.: An automaton over data words that captures EMSO logic. In: Katoen, J.P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 171–186. Springer (2011)
8. Bollig, B., Cyriac, A., Gastin, P., Kumar, K.N.: Model checking languages of data words. In: Birkedal, L. (ed.) FOSSACS 2012. LNCS, vol. 7213, pp. 391–405. Springer (2012)
9. Cheng, E.Y.C., Kaminski, M.: Context-free languages over infinite alphabets. Acta Inf. 35(3), 245–267 (1998)
10. Degano, P., Ferrari, G.L., Mezzetti, G.: Nominal automata for resource usage control. In: Moreira, N., Reis, R. (eds.) CIAA 2012. LNCS, vol. 7381, pp. 125–137. Springer (2012)

11. Demri, S., Lazić, R., Nowak, D.: On the freeze quantifier in constraint ltl: Decidability and complexity. Information and Computation 205(1), 2–24 (2007)
12. Ferrari, G., Gnesi, S., Montanari, U., Pistore, M.: A model-checking verification environment for mobile processes. TOSEM 12(4), 440–473 (2003)
13. Gordon, A.D.: Notes on nominal calculi for security and mobility. In: Focardi, R., Gorrieri, R. (eds.) FOSAD 2000. LNCS, vol. 2171, pp. 262–330. Springer (2001)
14. Grumberg, O., Kupferman, O., Sheinvald, S.: Variable automata over infinite alphabets. In: Dediu, A.H., Fernau, H., Martín-Vide, C. (eds.) LATA. LNCS, vol. 6031, pp. 561–572. Springer (2010)
15. Hazel, P.: Pcre: Perl compatible regular expressions (2005), `http://www.pcre.org/pcre.txt`
16. Kaminski, M., Francez, N.: Finite-memory automata. TCS 134(2), 329–363 (1994)
17. Kurz, A., Suzuki, T., Tuosto, E.: On nominal regular languages with binders. In: Birkedal, L. (ed.) FOSSACS 2012. LNCS, vol. 7213, pp. 255–269. Springer (2012)
18. Montanari, U., Pistore, M.: π-calculus, structured coalgebras, and minimal hd-automata. In: Mogens, N., Branislav, R. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 569–578. Springer (2000)
19. Parys, P.: Higher-order pushdown systems with data. In: Faella, M., Murano, A. (eds.) GandALF. EPTCS, vol. 96, pp. 210–223 (2012)
20. Perrin, D., Pin, J.: Infinite words: automata, semigroups, logic and games, Pure and Applied Mathematics, vol. 141. Elsevier (2004)
21. Pitts, A.M.: Nominal sets names and symmetry in computer science: Names and symmetry in computer science. Cambridge Tracts in Theoretical Computer Science, vol. 57. Cambridge University Press
22. Pitts, A.M., Stark, I.D.B.: Observable properties of higher order functions that dynamically create local names, or what's new? In: Andrzej, M.B., Stefan, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 122–141. Springer (1993)
23. Segoufin, L.: Automata and logics for words and trees over an infinite alphabet. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 41–57. Springer (2006)
24. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: LICS. pp. 332–344. IEEE Computer Society (1986)