On the coverability and reachability languages of monotonic extensions of Petri Nets

Fernando Rosa-Velardo[a,1], Giorgio Delzanno[b]

[a] *Universidad Complutense de Madrid, Spain*
[b] *Università di Genova, Italy*

Abstract

We apply language theory to compare the expressive power of infinite-state models that extend Petri nets with features like colored tokens and/or whole place operations. Specifically, we consider extensions of Petri nets in which tokens carry pure names dynamically generated with special ν-transitions (ν-PN) and compare their expressiveness with transfer and reset nets with black indistinguishable tokens (Affine Well-Structured Nets), and nets in which tokens carry data taken from a linearly ordered domain (Data nets and CMRS). All these models are well-structured transitions systems. In order to compare these models we consider the families of languages they recognize, using coverability as accepting condition. With this criterion, we prove that ν-PNs are in between AWNs and Data Nets/CMRS, but equivalent to an extension of ν-PN with whole-place operations. These results extend the currently known classification of the expressive power of well-structured transition systems. Finally, we study several problems regarding (coverability) languages of AWN and ν-PN.

Keywords: Language Theory, Petri Nets, Expressive Power, Well Structured Transition Systems

1. Introduction

Dynamic name generation has been thoroughly studied in the past decade, mainly in the field of security and mobility [13]. Paradigmatic examples of nominal calculi are the π-calculus and the Ambient Calculus [13]. Along this research line, in previous works we have studied an extension of Petri nets, that we called ν-PN [24]. Tokens in ν-PNs are pure names, that can be created fresh, moved along the net and used to restrict the firing of transitions with name matching. Names can be seen as process identifiers [22], so that ν-PN

Email addresses: `fernandorosa@sip.ucm.es` (Fernando Rosa-Velardo), `giorgio@disi.unige.it` (Giorgio Delzanno)

[1]Author partially supported by the MEC Spanish project DESAFIOS10 TIN2009-14599-C03-01, and Comunidad de Madrid program PROMETIDOS S2009/TIC-1465.

can serve as the basis of models in which an unbounded number of components (which are in turn unbounded) synchronize, as in resource-constrained workflow nets, an extension of workflow nets in which an arbitrary number of instances of the workflow can be executed concurrently [15], or in [7], where they are used to give a semantics to an extension of BPEL with instance isolation.

In previous works we studied the decidability and complexity of ν-PN [24]. In the first place, we have seen that reachability for them is undecidable. However, the transition system produced by a ν-PN belongs to the class of (strictly) Well Structured Transition Systems (WSTS) [24]. This means that the problems of termination (whether every execution is finite), coverability (whether a marking which is *greater* than a given one is reachable) and boundedness (whether the set of reachable states is finite) are all decidable. However, most of the refinements of the notion of boundedness yield undecidability. For instance, place-boundedness (whether every reachable marking contains a bounded number of tokens in a given place) is undecidable. Finally, we proved that all the decidable problems have a non-primitive recursive complexity.

In this paper we compare ν-PN with other extensions of Petri nets that are also WSTS. Among these models, we highlight *Affine Well-structured Nets* (AWN) [10], a well-structured extension of Petri nets in which whole-place operations (as transfers and resets) are allowed; Data nets [18], an extension of AWNs in which tokens are no longer indistinguishable, but taken from a linearly ordered domain; and CMRS [4], a fragment of Data nets without whole-place operations. All above mentioned models are well-structured transition systems in which the reachability problem is undecidable.

To compare the expressive power of different models, it comes natural to study the class of languages generated by associating labels to transitions: a finite firing sequence defines a word. The standard notions of acceptance is based on reachability of a configuration. Other acceptance notions used in the literature are termination, coverability and no condition.

We first compare several variants of ν-PN with each acceptance condition, which will provide us with useful techniques in the rest of the paper. We prove that ν-PN are equivalent (with any accepting condition) to a variation of ν-PN allowing to check for inequality of names, which we denote by $\nu_{\neq}$-PN. Moreover, we prove that if we forbid name matching in ν-PN, then its expressive power boils down to that of Petri nets for any accepting condition.

Moreover, we prove that the class of languages accepted by ν-PN with reachability or termination is RE, the class of Recursively Enumerable languages. This is typical for Well Structured Transitions Systems in which reachability is undecidable. Therefore, though reachability is the more standard acceptance condition, we need finer-grain criteria to distinguish Petri nets extended with whole place operations and colored tokens. More specifically, we consider well-structured languages [11], in which the acceptance condition is defined using coverability of a given configuration.

In [2] such comparison is done for Petri nets, AWNs, and Data Nets, and the following is proved:

$$\mathcal{L}(\text{Petri nets}) \subset \mathcal{L}(\text{AWN}) \subset \mathcal{L}(\text{Data nets})$$

Moreover, the authors proved that Data nets are equivalent (they generate the same family of languages) to the so called Petri Data nets, Data nets for which no whole-place operation is allowed, and equivalent to CMRS. We plan to put ν-PN in that picture, by studying the family of coverability languages recognized by them. We prove that ν-PN are strictly above AWN. Moreover, we prove that they are strictly below Data Nets. This last result relies on [5], in which a framework to prove non inclusions between families of WSTS coverability languages is defined. Moreover, we prove that ν-PN are equivalent to an extension allowing whole-place operations, that we call $w\nu$-PN.

We then study closure and decidability properties of the languages accepted by ν-PN, and also for AWN. We prove that the class of coverability languages of ν-PN satisfy a good number of closure properties. However, closure under iteration remains open. Then, we study several decidability results of the coverability languages of AWN and ν-PN. It is immediate to see that emptyness and membership are decidable. The rest of the problems we will consider are undecidable. For instance, we prove undecidability of the universality and the regularity problems. Then, we prove that it is undecidable whether a given AWN accepts the language of some Petri net and, analogously, whether a ν-PN accepts the language of some AWN. Finally, we prove that we cannot compute a regular expression that generates the downward-closure of the language of an AWN or a ν-PN (for any accepting condition), even if such regular expression always exists. This is the case even if we consider injective and ϵ-free labellings of transitions. This contrasts with the situation for Petri nets [14], in which such regular expression is always computable.

The rest of the paper is organized as follows: Section 2 defines some basic concepts that we use throughout the paper. In Sect. 3 we define ν-PN and compare the languages they accept with different accepting conditions. In Section 4 we compare ν-PN and Data Nets. Section 5 considers ν-PN with whole-place operations. In Section 6 we compare the languages recognized by AWN and ν-PN. In Section 7 we study closure properties of the class of ν-PN languages, while its decision properties are studied in Sect. 8. Finally, in Section 9 we draw some conclusions.

Note. Some of the results in this paper already appear in a preliminary version in [23].

2. Preliminaries

We write $[n] = \{1, ..., n\}$ for any $n \in \mathbb{N}$.

Languages. Given a (finite) alphabet Σ, any $w = \mathbf{a}_1 \cdots \mathbf{a}_n$ with $n \geqslant 0$ and $\mathbf{a}_i \in \Sigma$, for all $i \in [n]$, is a (finite) *word* on Σ. If $n = 0$ then w is the empty word, denoted by ϵ. We denote by Σ^* the set of words on Σ and $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$. A *language* on Σ is a set of words on Σ. If we denote by $\cdot$ the word concatenation, then $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1,\ w_2 \in L_2\}$ is the *concatenation* of L_1 and L_2. If we denote by L^i the language $L \overset{i}{\cdots} L$, the *iteration* L^+ of the language L is $\bigcup_{i>0} L^i$. A function $h : \Sigma^* \to \Sigma^*$ is a *homomorphism* if $h(w_1 \cdot w_2) = h(w_1) \cdot h(w_2)$. Given a homomorphism h and a language L, we can define $h(L) = \{h(w) \mid w \in L\}$ and $h^{-1}(L) = \{w \mid h(w) \in L\}$.

A *semi-full abstract family of languages* (semi-full AFL) [12] is a family of languages closed under union, intersection with regular languages, homomorphism and inverse homomorphism. A semi-full AFL is a *full AFL* if it is closed under concatenation and iteration.

Well Structured Transition Systems. A *quasi-order* $\leqslant$ is a reflexive and transitive binary relation on a set X. A quasi order is a *well quasi-order* (wqo) [9], if for every infinite sequence $s_0, s_1, \ldots$ there are i and j with $i < j$ such that $s_i \leqslant s_j$.

A *labelled transition system* is a tuple $(S, \Sigma, \to, s_0, s_f)$ with set of states S, set of labels Σ, initial and final states $s_0 \in S$ and $s_f \in S$, respectively, and transition relation $\to \subseteq S \times \Sigma_\epsilon \times S$. We write $s \xrightarrow{\mathbf{a}} s'$ instead of $(s, \mathbf{a}, s') \in \to$. Moreover, if $s \xrightarrow{\mathbf{a}} s'$ does not hold for any $\mathbf{a} \in \Sigma_\epsilon$ and $s' \in S$, we write $s \not\to$. For $w \in \Sigma^*$ we write $s \xrightarrow{w} s'$ if s' can be reached from s and the concatenation of the labels of the transitions used (some of which may be ϵ) is the word w.

A *labelled well structured transition system* (WSTS for short) is a tuple $(S, \Sigma, \to, s_0, s_f, \leqslant)$, where $(S, \Sigma, \to, s_0, s_f)$ is a labelled transition system, and $(S, \leqslant)$ is a wqo satisfying the following monotonicity condition: $s_1 \leqslant s_2$ and $s_1 \xrightarrow{w} s_1'$ implies the existence of s_2' such that $s_2 \xrightarrow{w} s_2'$ and $s_1' \leqslant s_2'$.

In the classic theory of Petri net languages [20] three types of labelling functions are considered: injective, ϵ-free and arbitrary. In this work we concentrate on arbitrary labelling functions, which lead to better closure properties. Moreover, four acceptance conditions can be considered: reachability, coverability, deadlock and no condition.[2]

Definition 1. Given a labelled transition system $\mathcal{S} = (S, \Sigma, \to, s_0, s_f)$ endowed with a quasi-order $\leqslant$, we define:

- $\mathcal{L}^L(\mathcal{S}) = \{w \in \Sigma^* \mid s_0 \xrightarrow{w} s_f\}$,
- $\mathcal{L}^G(\mathcal{S}) = \{w \in \Sigma^* \mid s_0 \xrightarrow{w} s,\ s \geqslant s_f\}$,
- $\mathcal{L}^T(\mathcal{S}) = \{w \in \Sigma^* \mid s_0 \xrightarrow{w} s,\ s \not\to\}$,
- $\mathcal{L}^P(\mathcal{S}) = \{w \in \Sigma^* \mid s_0 \xrightarrow{w} s\}$,

Notice that conditions T and P do not make use of the final state s_f. For any of the models $\mathbf{M}$ we consider in this paper, we denote by $\mathcal{L}^R(\mathbf{M})$ the class of languages $\{\mathcal{L}^R(\mathcal{S}) \mid \mathcal{S} \in \mathbf{M}\}$. We will sometimes refer to $\mathcal{L}^R(\mathbf{M})$ as the class

[2]We use a notation borrowed from [20].

of R-languages of $\mathbf{M}$. A *Well Structured Language* (WSL) is the G-language of some WSTS [11].

Given a WSTS $\mathcal{S}$, a *lossy version* of $\mathcal{S}$ is obtained from $\mathcal{S}$ by adding some transitions $s \xrightarrow{\epsilon} s'$ with $s' \leqslant s$. It holds [2] that if $\mathcal{S}'$ is a lossy version of $\mathcal{S}$ then $\mathcal{L}^G(\mathcal{S}) = \mathcal{L}^G(\mathcal{S}')$. A WSTS is *lossy* if it contains every such transition. For a lossy WSTS $\mathcal{S}$, we also have $\mathcal{L}^G(\mathcal{S}) = \mathcal{L}^L(\mathcal{S})$. For all the usual classes of WSTS, and certainly for all the classes considered in this paper, given $\mathcal{S}$ it is possible to find a lossy version of $\mathcal{S}$, which is lossy, in the same class. This means that G-languages are in particular L-languages for them. As a consequence, we have the following result.

Proposition 1. *Let* $\mathbf{M}$ *be any class of WSTS appearing in this paper. Then,* $\mathcal{L}^G(\mathbf{M}) \subseteq \mathcal{L}^L(\mathbf{M})$.

For instance, the classical proof [20] of the result above for Petri nets is a particular case of the previous comments about lossiness.

Multisets. A (finite) *multiset* m over a set A is a mapping $m : A \to \mathbb{N}$ with finite support, that is, such that $supp(m) = \{a \in A \mid m(a) > 0\}$ is finite. We denote by $A^{\oplus}$ the set of finite multisets over A. When needed, we identify each set with the multiset defined by its characteristic function, and use set notation for multisets when convenient, with repetitions to account for multiplicities greater than one. We take $|m| = \sum_{a \in supp(m)} m(a)$ the *cardinality* of m. We denote by $m_1 + m_2$, $m_1 \subseteq m_2$ and $m_1 - m_2$ the multiset addition, inclusion, and subtraction, respectively. If $f : A \to B$ is a mapping and $m \in A^{\oplus}$ then we can define $f(m) \in B^{\oplus}$ by $f(m)(b) = \sum_{f(a)=b} m(a)$.

Every quasi-order $\leqslant$ defined over A induces a quasi-order $\leqslant^{\oplus}$ in the set $A^{\oplus}$, given by $\{a_1, \ldots, a_n\} \leqslant^{\oplus} \{b_1, \ldots, b_m\}$ if there is an injection $h : [n] \to [m]$ such that $a_i \leqslant b_{h(i)}$ for all $i \in [n]$. It is well known that the multiset order induced by any wqo is a wqo [16].

Petri Nets. A *labelled Petri Net* (PN) [21] is a tuple $N = (P, T, F, \lambda)$, where P is a finite set of places, T is a finite set of transitions (disjoint with P), $\lambda : T \to \Sigma_\epsilon$ is the labelling of transitions and $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the flow function. A *marking* of N is an element of $P^{\oplus}$. For a transition t we define $pre(t) \in P^{\oplus}$ as $pre(t)(p) = F(p, t)$. Analogously, we take $post(t)(p) = F(t, p)$. A marking m *enables* a transition $t \in T$ if $pre(t) \subseteq m$. Then t can be *fired*, reaching the marking $m' = (m - pre(t)) + post(t)$, in which case we write $m \xrightarrow{\lambda(t)} m'$. Any Petri net induces a labelled transitions system, once endowed with an initial and a final marking, as defined in the preliminaries. This will be the case for every model defined in the paper. The labelled transition system induced by any PN, with multiset inclusion, is a WSTS [9].

3. Nets in which tokens carry pure names

We now define the class ν-PN, an extension of Petri nets in which tokens are not indistinguishable, but pure names, that can only be compared by the

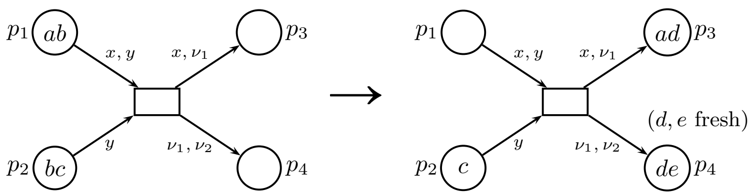

Figure 1: A ν-PN and the firing of its only transition

equality predicate. We consider a set Id of names, a set Var of variables and a subset of special variables $\Upsilon \subset Var$ used for fresh name creation.

Definition 2 (ν-Petri Nets). A *labelled ν-Petri Net* (ν-PN) is a tuple $N = (P, T, F, \lambda)$, where P and T are finite disjoint sets of elements called places and transitions, respectively, $\lambda : T \to \Sigma_\epsilon$ is the labelling of transitions, and

$$F : (P \times T) \cup (T \times P) \to Var^{\oplus}$$

is such that for every $t \in T$, $pre(t) \cap \Upsilon = \varnothing$ and $post(t) \backslash \Upsilon \subseteq pre(t)$, where $pre(t) = \bigcup_{p \in P} supp(F(p,t))$ and $post(t) = \bigcup_{p \in P} supp(F(t,p))$. We also take $Var(t) = pre(t) \cup post(t)$.

The mapping F labels every pair (p, t) and (t, p) by a multiset of variables. These variables specify how tokens flow from preconditions to postconditions. Variables in Υ can only be instantiated to names that do not occur in the current marking, so that they formalize fresh name creation. We are assuming that these variables only appear in post-arcs, that is, labelling pairs of the form (t, p). Moreover, these are the only variables that can appear only in post-arcs.

Definition 3 (Markings). A *marking* of a ν-PN $N = (P, T, F, \lambda)$ is a mapping $M : P \to Id^{\oplus}$. We take $Id(M) = \bigcup_{p \in P} supp(M(p))$, the set of names in M.

Thus, a marking M assigns to each place a multiset of names. We will often refer to an occurrence of $a \in M(p)$ as an a-token in p. Given a transition $t \in T$, a *mode* of t is a mapping $\sigma : Var(t) \to Id$ such that $\sigma(\nu_1) \neq \sigma(\nu_2)$ for each different $\nu_1, \nu_2 \in \Upsilon$.

Definition 4 (Enabling and firing). A transition t is *enabled* with mode σ for a marking M if for all $p \in P$, $\sigma(F(p,t)) \subseteq M(p)$ and $\sigma(\nu) \notin Id(M)$ for all $\nu \in \Upsilon$. Then t can be *fired* with mode σ, reaching the marking M' given by $M'(p) = (M(p) - \sigma(F(p,t))) + \sigma(F(t,p))$ for all $p \in P$. In that case we write $M \xrightarrow{\lambda(t)} M'$.

Example 1. Figure 1 depicts a simple ν-PN with four places and a single transition. This transition moves one token from p_1 to p_3 (because of variable x labelling both arcs), removes a token from p_1 and p_2 provided they carry the same name (variable y appears in both incoming arcs but it does not appear in any outgoing arc), and two different names are created: one appears both in p_3 and p_4 (because of $\nu_1 \in \Upsilon$) and the other appears only in p_4 (because of $\nu_2 \in \Upsilon$).

We will assume that $\bullet$ is a name in Id, in order to use ordinary black tokens in ν-PN as in PN.

In the previous example, if we replace in the initial marking every occurrence of b by a, the transition could also have been fired, since modes can instantiate different variables with the same name. In other words, in ν-PNs we cannot check for inequality. We consider a variation of ν-PNs, that we call $\nu_{\neq}$-PNs, in which we can check for inequality, which can be simply formalized by taking modes to be injections. Then, if the net in Fig. 1 is actually a $\nu_{\neq}$-PN, its transition becomes disabled when b is replaced by a.

We define $M_1 \sqsubseteq M_2$ if there is an injection $\iota : Id(M_1) \to Id(M_2)$ such that $\iota(M_1(p)) \subseteq M_2(p)$, for all $p \in P$. The relation $\sqsubseteq$ is a wqo and the transition system generated by ν-PNs and $\nu_{\neq}$-PNs are WSTS with that order [24]. Notice that if $M_1 \sqsubseteq M_2$ and $M_2 \sqsubseteq M_1$ then M_2 can be obtained from M_1 by renaming. Let us denote by $\equiv$ the kernel of $\sqsubseteq$, that is, $M_1 \equiv M_2$ iff $M_1 \sqsubseteq M_2$ and $M_2 \sqsubseteq M_1$. We remark that transitions are enabled up to renaming, that is, if t is enabled according to M_1 and $M_1 \equiv M_1'$ then t is also enabled in M_1'.

Though the order $\sqsubseteq$ allows renaming, in the definition of $\mathcal{L}^L(\nu\text{-PN})$ we are requiring that we reach exactly the final marking M_f, not a renaming of it. We could think that by allowing renaming we could end up with a different class of languages, though we will see this is not the case in Lemma 1. Next we define the class of reachability languages, up to renaming.

Definition 5 (L_α-languages). Given a ν-PN N, with initial and final marking M_0 and M_f, respectively, we define $\mathcal{L}^{L_\alpha}(N) = \{w \in \Sigma^* \mid M_0 \xrightarrow{w} M \equiv M_f\}$, and $\mathcal{L}^{L_\alpha}(\nu\text{-PN}) = \{\mathcal{L}^{L_\alpha}(N) \mid N \in \nu\text{-PN}\}$.

We will refer to that class as the class of L_α-languages of ν-PN.

Example 2. Consider the ν-PN N in the left of Fig. 4, with initial marking M_0 given by $M_0(p_1) = \{a\}$ and $M_0(p_2) = \varnothing$. Assume that the labelling of the transitions is the identity, so that we are using $\{t\}$ as alphabet. If we consider as the final marking that with only an a-token in p_2, we cannot reach the final marking, because t produces in p_2 a name that must be different from a. Hence, $\mathcal{L}^L(N) = \varnothing$ with that empty marking. However, the reached marking is a renaming of the final marking, so that $\mathcal{L}^{L_\alpha}(N) = \{t\}$. If we consider as final marking that with only a b-token in p_2, it is true that $\mathcal{L}^L(N) = \mathcal{L}^{L_\alpha}(N) = \{t\}$.

Before relating L-languages and L_α-languages in Lemma 1, let us study the relation between the R-languages of ν-PN and $\nu_{\neq}$-PN. We will obtain in Cor. 1 thay both classes of nets are equivalent, with any accepting condition. First, let us see that being able to check inequality gives us at least as much expressive power:

Proposition 2. *$\mathcal{L}^R(\nu\text{-PN}) \subseteq \mathcal{L}^R(\nu_{\neq}\text{-PN})$ for $R \in \{L, L_\alpha, G, T, P\}$.*

Proof. We simulate a ν-PN N by means of a $\nu_{\neq}$-PN N', with the same places. Let us see how we simulate a transition t of N. For any partition

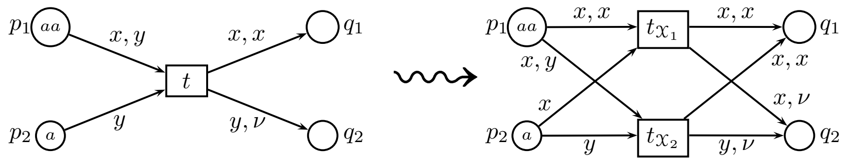

Figure 2: Simulation of ν-PN (left) by means of a $\nu_{\neq}$-PN (right)

$\mathcal{X} = \{X_1, ..., X_k\}$ of $Var(t) \backslash \Upsilon$, we choose variables $x_1, \dots, x_k$ so that $x_i \in X_i$. Then, for every such partition we consider in N' a transition $t_{\mathcal{X}}$ (with the same label as t). Intuitively, in $t_{\mathcal{X}}$, variables in the same set are instantiated to the same name. Therefore, we obtain $F(p, t_{\mathcal{X}})$ from $F(p, t)$ by replacing every variable $x \in X_i$ by x_i, for every $i \in [k]$. Analogously, we define $F(t_{\mathcal{X}}, p)$. Finally, the initial (final) marking of N' is just the initial (final) marking of N. □

In Fig. 2 the case with $Var(t) \backslash \Upsilon = \{x, y\}$ is shown, that has two possible partitions, $\mathcal{X}_1 = \{\{x, y\}\}$ (for which x and y are the same, so that y is replaced by x) and $\mathcal{X}_2 = \{\{x\}, \{y\}\}$ (for which x and y are different). In this case, $t_{\mathcal{X}_1}$ can be fired, but not $t_{\mathcal{X}_2}$. The converse of the previous result is also true, that is, considering checks for inequalities does not give us more expressive power.

Proposition 3. *$\mathcal{L}^R(\nu_{\neq}$-PN$) \subseteq \mathcal{L}^R(\nu$-PN$)$ for $R \in \{L, L_\alpha, G, T, P\}$.*

Proof. We have to simulate a $\nu_{\neq}$-PN by means of a ν-PN. We simply add a new place *all* that contains at each time a single copy of every name that has appeared along the current execution (see Fig. 3). It initially contains a single copy of the names in the initial marking, and every name that is created is also put in *all*. Then, for every $t \in T$, we add $F(all, t) = Var(t) \backslash \Upsilon$ and $F(t, all) = Var(t)$. Since *all* contains a single copy of all names, two different variables are necessarily instantiated to different names. This is enough for $R \in \{G, T, P\}$, considering for $R = G$ that the new place *all* is empty in the final marking. For $R \in \{L, L_\alpha\}$ we also take *all* as empty in the final marking, but we add a new transition that can always remove tokens from *all*. □

Corollary 1. *$\mathcal{L}^R(\nu$-PN$) = \mathcal{L}^R(\nu_{\neq}$-PN$)$ for $R \in \{L, L_\alpha, G, T, P\}$.*

Therefore, we can use ν-PN or $\nu_{\neq}$-PN indifferently, so that we will use the most convenient in each case. However, notice that ν-PN allow an exponentially more succinct description of our systems. Indeed, in the proof of Prop. 2 each transition t is simulated by $B_{|Var(t)|}$ transitions, where B_n is the n-th Bell number.

Let us see that the class of L-languages and L_α-languages of ν-PN coincide. In order to simplify the proof, we will work with ν-PN for which every transition can create at most one fresh name by means of the special variable $\nu \in \Upsilon$, that is, such that $Var(t) \cap \Upsilon \subseteq \{\nu\}$ for every $t \in T$. This will be assumed several times in the paper. Indeed, if $|Var(t) \cap \Upsilon| = n$, we can replace t by the sequential firing of n new transitions t_1 (labelled as t), $t_2, ...t_n$ (labelled by ϵ). The first transition, t_1, has the same effect as t, except because it creates only one fresh

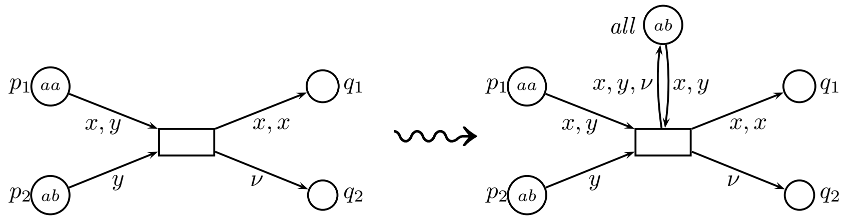

Figure 3: Simulation of $\nu_{\neq}$-PN (left) by means of a ν-PN (right)

name, by means of ν. Moreover, each transition in $t_2, ..., t_n$ also creates a single fresh name by means of ν.

Lemma 1. $\mathcal{L}^{L_\alpha}(\nu\text{-}PN) = \mathcal{L}^{L}(\nu\text{-}PN)$

Proof. We work with $\nu_{\neq}$-PN (Cor. 1). First, let us see that $\mathcal{L}^{L_\alpha}(\nu\text{-PN}) \subseteq \mathcal{L}^{L}(\nu\text{-PN})$. Let N be a $\nu_{\neq}$-PN with final marking M_f. We build N' by adding to N an ϵ-labelled transition that removes M_f (or any renaming of it) and puts a black token in a new place *accept*. Then, if the final marking of N' is that with a black token in *accept*, $\mathcal{L}^{L_\alpha}(N) = \mathcal{L}^{L}(N')$.

Conversely, let $N = (P, T, F, \lambda)$ be a $\nu_{\neq}$-PN with initial and final markings M_0 and M_f, respectively. Let us build $N' = (P', T', F', \lambda')$ with initial and final markings M_0' and M_f', respectively, such that $\mathcal{L}^{L}(N) = \mathcal{L}^{L_\alpha}(N')$.

The main idea is to keep track (despite renamings) of names in $I = Id(M_0)$, and to distinguish them from all other names that are dynamically generated during a computation (names that may or may not appear in the final marking M_f). In other words, we want to make those names conspicuous. We do that by adding for each $a \in I$ a new place p^a. Intuitively, this place is meant to contain a. Furthermore, we use a place *other* for all other names. Therefore, p^a initially contains a. Furthermore, we add a place del^a to denote the fact that the name a has been (temporarily) deleted. The place del^a maintains the name active, so that it cannot be generated again. Creation of fresh names are either reactivations of the name in place del^a or creation of a new name (that is necessarily different from that in p^a or del^a). Every other freshly created name is put in the special place *other*.

The transformation is defined as follows. For each transition t, we generate the least set of transitions t' such that the definition of F in t is extended to the new set of places in order to satisfy the following conditions:

- if x is copied from the precondition to the postcondition of t, i.e., $x \in F(p, t) \cap F(t, p)$, then
  - either $x \in F(other, t')$ and $x \in F(t', other)$, i.e., x selects a name from *other*,
  - or $x \in F(p^a, t')$ and $x \in F(t', p^a)$ for some $a \in I$, i.e., x selects the initial name a;
- if x is removed from the application of t, i.e., $x \in F(p, t) \backslash F(t, p)$, then

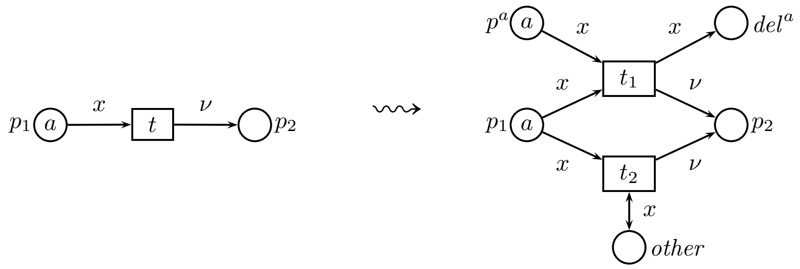

Figure 4: Construction in Lemma 1

- either $x \in F(other, t')$ and $x \notin F(t', other)$, i.e., x selects and removes a name from place *other*;
- or $x \in F(p^a, t')$ and $x \in F(t', del^a)$ for some $a \in I$, i.e., x selects and temporarily disactivate one of the initial names;

- if $x \in F(p, t) \cap \Upsilon$, then
  - either $x \in F(del^a, t')$ and $x \in F(t', p^a)$ for some $a \in I$, i.e., x reactivates an initial name (i.e. x is removed from Υ)
  - or $x \in F(t', other)$, i.e., x puts the fresh name into *other*.

The initial marking M'_0 coincides with M_0 in places in P. Moreover, $M'_0(p^a) = \{a\}$ if $a \in I$, and is empty elsewhere. As final marking M'_f, we extend M_f by requiring that a name $a \in I$ in M_f is contained into place p^a, and each name $b \in M_f$ but not in I is contained into place *other*, and is empty elsewhere. Since every newly introduced (distinct from those in I) can be renamed, in M'_f we can select exactly those in M_f. Then, $\mathcal{L}^L(N) = \mathcal{L}^{L_\alpha}(N')$ holds. □

Example 3. As an example, consider the net N in the left of Fig. 4. Furthermore, consider the initial marking M_0 with the sole name a in p_1. The marking with a in p_2 is unreachable in N, whereas the marking with $b \neq a$ in p_2 is reachable.

We build N' (in the right of Fig. 4) by adding the places $p^a, del^a, other$ together with transition t_1 s.t. $F(p_1, t_1) = F(p^a, t_1) = \{x\}$ and $F(t_1, del^a) = \{x\}$, and $F(t_1, p_2) = \{\nu\}$, and transition t_2 s.t. $F(p_1, t_2) = F(other, t_2) = \{x\}$ and $F(t_2, other) = F(t_2, p_2) = \{\nu\}$. Transition t_1 models the removal of name a and creation of a fresh name, necessarily distinct from a (following the semantics of ν). The name a is kept in the place del^a ready to be reused in later steps. Transition t_2 models the removal of a name distinct from a and the creation of a fresh name, necessarily distinct from a in accord to the semantics of ν. As in N, neither t_1 nor t_2 can be applied to obtain a marking in which a occurs in p_2. However both transitions can be applied to obtain a marking in which a distinct name, say b, occurs in p_2.

Consider now the net N with places p_1, p_2, p_3, a transition t_1 that removes a name in p_1 and puts a black token in p_2, and a transition t_2 that removes the black token from p_2 and creates a fresh name in p_3. If we start from name a in p_1, we may end up in a marking with any name (including a) in p_3. In our encoding we obtain the same effect because we can first copy a from p^a to del^a, and then reuse it by moving it back to p^a.

Thus, we may use L-languages or L_α-languages indifferently. Informally, we may define a final marking for instance by saying that it has two different tokens in a given place, without specifying which names carry those tokens.

Now, let us see that with reachability or termination, we reach the expressiveness of Turing machines. First we prove the following lemma.

Lemma 2. $\mathcal{L}^L(\nu\text{-}PN) \subseteq \mathcal{L}^T(\nu\text{-}PN)$

Proof. We work with $\mathcal{L}^{L_\alpha}(\nu\text{-PN})$ (Lemma 1). Given a ν-PN N, we build N' such that $\mathcal{L}^{L_\alpha}(N) = \mathcal{L}^T(N')$. Let M_0 and M_f be the initial and final markings of N, respectively. We add a new place *run*, initially marked, which is a precondition/postcondition of every transition in N. We also add a new transition $\bar{t}_1$ (labelled with ϵ), with *run* both as precondition and postcondition, and a new transition $\bar{t}_2$ (labelled with ϵ), with M_f and *run* as precondition, and a new place *stop* as postcondition. Thus, when M_f is covered, $\bar{t}$ can move the token from *run* to *stop* and remove M_f. Finally, for each place $p \in P$, we add a transition t_p (labelled with ϵ), that has both *stop* and p as precondition and as postcondition. Then, when *stop* and p are marked, t_p can be fired infinitely often. Therefore, the only dead marking in N' is that with a token in *stop* and empty elsewhere.

It holds that $\mathcal{L}^{L_\alpha}(N) = \mathcal{L}^T(N')$. Indeed, if N reaches M_f through w, then N' can reach through w the marking given by M_f in the places of N, and a token in *run*. Then, it can fire $\bar{t}_2$, reaching the marking with a single token in *stop*. Since this marking is dead, w is in $\mathcal{L}^T(N')$. Conversely, if N' reaches the marking with a token in *stop* and empty elsewhere (its only dead marking), then necessarily $\bar{t}_2$ has been fired exactly from the marking given by M_f in the places of N, and $w \in \mathcal{L}^{L_\alpha}(N)$. □

Instead of working with Turing machines, we consider *Inhibitor nets*, which extend Petri nets with the so called inhibitor arcs. An inhibitor arc (p, t) restricts the firing of t to happen only when p is empty. It is well known that inhibitor nets with at least two inhibitor arcs are Turing complete [19]. In particular, inhibitor nets have the expressive power of Turing machines, and the class of L-languages of inhibitor nets is RE, the class of recursively enumerable languages.

Proposition 4. $\mathcal{L}^L(\nu\text{-}PN) = \mathcal{L}^T(\nu\text{-}PN) = RE$

Proof. By the two previous lemmas, it is enough to see that $\mathcal{L}^{L_\alpha}(\nu\text{-PN}) = RE$. If L is a recursively enumerable language then there is an inhibitor net that has L as L-language. By using the standard technique of removing the final marking, we can assume that the inhibitor net has the empty marking as final marking. Let us see that we can weakly simulate it by means of a ν-PN, in

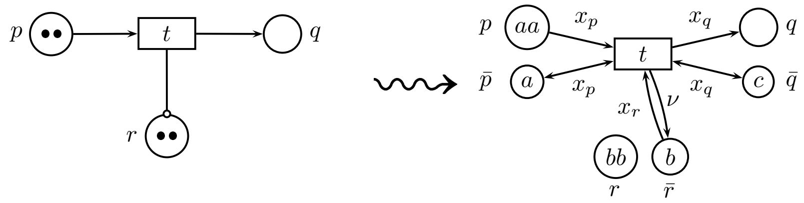

Figure 5: Weak simulation of Petri nets with inhibitor arcs

a way that preserves reachability languages. We reuse the construction that proves undecidability of reachability for ν-PNs [24]. For each place p we consider a new place $\overline{p}$ and a different variable x_p. Each $\overline{p}$ will contain at any time a single token, that identifies the "legal" tokens in p. Initially, each $\overline{p}$ contains a different name. Every transition is fired so that the name that is used in p coincides with that in $\overline{p}$. Moreover, if there is an inhibitor arc (p, t), then the firing of t replaces the current name in $\overline{p}$ by a fresh one. See Fig. 5 to illustrate the construction. The arc ending in a circle represents an inhibitor arc.

Our simulation can cheat, firing t even when there are legal tokens in p, though in that case garbage tokens remain in p (those that were legal tokens before the firing of t, but not after). As final marking we consider that with a different name in each $\overline{p}$, and empty elsewhere (we are specifying final markings modulo renaming). Then the L_α-language of the ν-PN is L. Indeed, any transition sequence in the inhibited net can be reproduced in the ν-PN. Moreover, cheating transition sequences cannot empty every place, so that they do not lead to the accepting marking. □

Now we are ready to establish the relations between all the accepting conditions considered.

Proposition 5. $\mathcal{L}^P(\nu\text{-}PN) \subset \mathcal{L}^G(\nu\text{-}PN) \subset \mathcal{L}^L(\nu\text{-}PN) = \mathcal{L}^T(\nu\text{-}PN)$

Proof. For the first inclusion it is enough to consider the empty marking as acceptance. To see that it is strict, notice that P-languages are always prefix-closed, and it is trivial to devise non prefix-closed languages in $\mathcal{L}^G(\nu\text{-PN})$. The second inclusion follows from Prop. 1. Moreover, it is strict because there are recursively enumerable languages that are not WSL, such as $L = \{a^n b^n \mid n > 0\}$, which can be easily seen using a pumping lemma for WSL proved in [11]: If L is a WSL and $(w_k)_{k=1}^{\infty} \subseteq L$ with $w_k = B_k \cdot E_k$ for every $k \geqslant 1$, then there are $i < j$ such that $B_j \cdot E_i \in L$. In our case, it is enough to take $B_k = a^k$ and $E_k = b^k$, which satisfy the hypothesis of the lemma, though no $a^j b^i \in L$ with $i < j$. The last equality is the previous proposition. □

To conclude this section, let us see that if we forbid name matching in ν-PN, then its expressive power boils down to that of PN.

Definition 6 ($\nu_=$-PN). A $\nu_=$-PN is a ν-PN $N = (P, T, F, \lambda)$ such that for each transition $t \in T$, $\sum_{p \in P} F(p,t)(x) \leqslant 1$ for every $x \in Var(t)$.

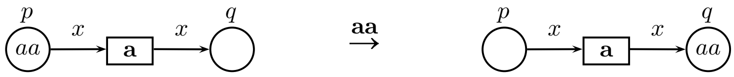

Figure 6: $\nu_=$-PN with final marking $M(p) = \varnothing$ and $M(q) = \{a, b\}$

Therefore, in $\nu_=$-PNs, variables in pre-arcs appear at most once. The intuitive idea is that, without matching, the specific nature of named tokens, that is, the identifiers carried by tokens, does not play any role in the firing of transitions. Therefore, we could flatten the given $\nu_=$-PN to the PN with the same places, transitions and flow relation, by removing variables in arcs and replacing each name in M_0 by a black token. This would be enough if we were considering T or P as accepting conditions, but this is not the case for G or L. To see it, it is enough to consider the net depicted in Fig. 6, using $M(p) = \varnothing$ and $M(q) = \{a, b\}$ as final marking. That net can fire its only transition twice, reaching a marking with the identifier a twice in place q, which does not cover M. Therefore, it generates the empty language, though the sketched construction would generate the language $\{\mathbf{aa}\}$. In other words, the accepting condition does allow us to retrieve some information about the involved tokens, even though that information was not relevant in the enabling and firing of transitions. However, that information is finite (about tokens in the initial and the final marking), so that we can control it with some special places.

Proposition 6. *$\mathcal{L}^R(PN) = \mathcal{L}^R(\nu_=\text{-}PN)$ for $R \in \{L, G, T, P\}$.*

Proof. The inclusion $\mathcal{L}^R(\text{PN}) \subseteq \mathcal{L}^R(\nu_=\text{-PN})$ for $R \in \{L, G, T, P\}$ follows from the fact that any PN can be seen as a $\nu_=$-PN (labelling all its arcs with variables in a legal way, so that all the pre-arcs of every transition are labelled by a different variable), and the two equalities for $R \in \{T, P\}$ derive from the fact that the acceptance conditions P and T are independent of the reached markings, so that a $\nu_=$-PN can be simulated by a PN just by erasing all variables, and replacing names in M_0 by black tokens.

Let us now see that $\mathcal{L}^R(\text{PN}) \supseteq \mathcal{L}^R(\nu_=\text{-PN})$, for $R \in \{L, G\}$. Let $N = (P, T, F, \lambda)$ be a $\nu_=$-PN N with initial and final markings M_0 and M_f. We assume that each transition can at most create a fresh name by means of $\nu \in \Upsilon$. Take $Id(M_0) = \{a_1, ..., a_k\}$ and $Id(M_f) = \{b_1, ..., b_l\}$. By Lemma 1, we can assume that M_f is given up to renaming for L-acceptance. Furthermore, by definition of $\sqsubseteq$ (based on an injection from names to names) we can also reason modulo renaming in the case of G-acceptance.

Let us define a PN $N' = (P', T', F', \lambda')$ such that $\mathcal{L}^R(N) = \mathcal{L}^R(N')$, for $R \in \{L, G\}$. For each $p \in P$, N' has places $p_0, p_1, ..., p_l$, so that a token in p_i with $i > 0$ represents a b_i-token in p, and a token in p_0 stands for some b-token in p, with $b \notin Id(M_f)$. We also consider places $f_1, ..., f_l$, initially marked, whose purpose will be explained later.

First, N' decides non-deterministically, what names in $Id(M_0)$ correspond to names in $Id(M_f)$. It starts its executions with k choices. The i-th choice decides if a_i is mapped to some name in $Id(M_f)$, and if it is, to which one. For

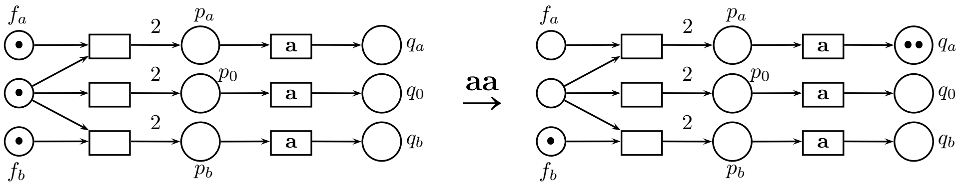

Figure 7: Simulation of the $\nu_=$-PN in Fig. 6 by means of a Petri net

that purpose we add transitions t_i^j (labelled by ϵ), whose firing represents that a_i is mapped to b_j if $j > 0$, or that it is not mapped to any name in $Id(M_f)$, if $j = 0$. Therefore, for $j > 0$, t_i^j removes a token from f_j, and puts a token in p_j for each a_i-token in $M_0(p)$; on the other hand, t_i^0 puts a token in p_0 for each a_i token in $M_0(p)$.

Notice that after this preliminary phase, if f_j is marked then b_j has not still been assigned any name, so that some fresh name during the execution of N must be assigned to b_j. On the contrary, if f_j is unmarked then b_j has already been assigned a name in the initial marking.

After the firing of the previous k transitions, N' starts to simulate N. For that purpose, for each $t \in T$ and each $\sigma : Var(t) \to \{0, ..., l\}$ we consider a transition t_σ (labelled by $\lambda(t)$). Intuitively, it simulates t, assuming that x is instantiated to b_i if $\sigma(x) = i$, or to some other name if $\sigma(x) = 0$. Therefore, we take $F'(p_{\sigma(x)}, t_\sigma) = F(p,t)(x)$ and $F'(t_\sigma, p_{\sigma(x)}) = F(t,p)(x)$. Notice that if $\sigma(\nu) = j$, then we are assigning to b_j the name that t is creating, so that we add f_j as precondition of such t_σ.

Finally, we consider as final marking M'_f given by $M'_f(p_i) = M_f(p)(b_i)$ for $i > 0$, and empty elsewhere. In particular, the places $f_1, ..., f_l$ are empty. It holds that $\mathcal{L}^{L_\alpha}(N) = \mathcal{L}^L(N')$ and $\mathcal{L}^G(N) = \mathcal{L}^G(N')$, and we are done. $\square$

Fig. 7 depicts the construction in the previous proof for the $\nu_=$-PN in Fig. 6. For a better readability, we write p_a and f_a instead of p_1 and f_1 (and analogously for p_b and f_b). The final marking is that with a token in q_a and a token in q_b, which is not reachable from the initial marking (shown in the figure). Hence, $\mathcal{L}^R(N') = \varnothing$ for $R \in \{L, G\}$ (as for N).

In the following sections, namely Sect. 4 to Sect. 6, we will compare ν-PN with other well-structured extensions of Petri nets. Hence, because the class of L-languages is RE, we will now focus on their G-languages.

4. Pure Names vs Ordered Data

In this section we compare ν-PNs with two extensions of Petri nets in which tokens carry data taken from an ordered domain, namely Data nets [18] and CMRS [4]. In [2] it is proved that $\mathcal{L}^G(\text{Data nets}) = \mathcal{L}^G(CMRS)$, so that we will work with CMRS only.

We assume a set $\mathbb{V}$ of variables which range over $\mathbb{N}$, and a set $\mathbb{P}$ of unary predicate symbols. In CMRS we write multisets as lists, so $[1, 5, 5, 1, 1]$ represents a multiset with three occurrences of 1 and two occurrences of 5. For a set

$V \subseteq \mathbb{V}$, a *valuation* *Val* of V is a mapping from V to $\mathbb{N}$. A *condition* is a finite conjunction of *gap order* formulas of the forms: $x <_c y$, $x \leqslant y$, $x = y$, $x < c$, $x > c$, $x = c$, where $x, y \in \mathbb{V}$ and $c \in \mathbb{N}$. Here $x <_c y$ stands for $x + c < y$. We use $x < y$ instead of $x <_0 y$. We use *true* to indicate an empty set of conditions. A *term* is of the form $p(x)$ where $p \in \mathbb{P}$ and $x \in \mathbb{V}$. A *ground term* is of the form $p(c)$ where $p \in \mathbb{P}$ and $c \in \mathbb{N}$.

A *constrained multiset rewriting system (CMRS)* $\mathcal{S}$ consists of a finite set of *rules* each of the form $L \rightsquigarrow R : \psi$, where L and R are multisets of terms, and ψ is a condition. Each rule ρ is labelled by some $\lambda(\rho) \in \Sigma_\epsilon$. For a valuation *Val*, we use $Val(\psi)$ to denote the result of substituting each variable x in ψ by $Val(x)$. We use $Val \models \psi$ to denote that $Val(\psi)$ evaluates to *true*. For a multiset T of terms we define $Val(T)$ as the multiset of ground terms obtained from T by replacing each variable x by $Val(x)$. A *configuration* is a multiset of ground terms. Each rule $\rho = L \rightsquigarrow R : \psi \in \mathcal{S}$ defines a relation between configurations. More precisely, we write $\gamma \xrightarrow{\lambda(\rho)} \gamma'$ if and only if there is a valuation *Val* such that: *(i)* $Val \models \psi$, *(ii)* $\gamma \supseteq Val(L)$, and *(iii)* $\gamma' = \gamma - Val(L) + Val(R)$. CMRS is well-structured when configurations are compared via an ordering $\leqslant$ that abstracts from concrete values and only considers equality and relative gaps between constants. For instance, $[p(1), q(1), q(3)] \leqslant [p(4), q(4), q(8), q(9)]$ since the first two atoms have the same value in both configurations and the gaps between the constants occurring in the former are less or equal than those occurring in the latter. To be more formal, let us assume that CMRS rules have only conditions of the form $x + c < y$ $(c \geqslant 0)$ or $x = y$ (we can simulate comparisons with a finite number of constants by using equalities with variables stored in special predicates). Let $Values(\gamma)$ be the set of values occurring in configuration γ (e.g. $Values([p(1), q(1), q(3)]) = \{1, 3\}$). For two configurations $\gamma = [p_1(a_1), \ldots, p_n(a_n)]$ and γ', we say that $\gamma \leqslant \gamma'$ if and only if $\gamma' = [p_1(b_1), \ldots, p_n(b_n)] + \gamma''$. and there exists an injection ι from $Values(\gamma)$ to $Values([p_1(b_1), \ldots, p_n(b_n)])$ s.t.

- $a_i = a_j$ iff $\iota(a_i) = \iota(a_j)$,
- $a_i + c < a_j$ implies $\iota(a_i) + d < \iota(a_j)$ where $c \leqslant d$.

In other words, configurations can symbolically be represented as multisets of terms with variables (instead of values) and gap order constraints defined over them (to keep track of equalities and gaps). Alternatively, configurations can be represented as strings (to represent relative ordering of values induced by gaps) of multisets of predicate symbols (to collect all the predicates with the same argument). Gaps can be represented as special substrings of singleton multisets (as many multisets as the minimal gaps between predicate arguments). For instance, $[p(1), q(1), q(3)]$ can be seen as the string $[p, q] \cdot [u] \cdot [q]$, where u is used to denote a 1-unit gap in between the predicates with argument 1 and that with argument 3.

By composing string and multiset inclusion we obtain a wqo ordering over configurations [3, 4] that makes CMRS a wsts. Next we show an example of CMRS.

Example 4. Consider the CMRS rule:

$$\rho = [p(x),\ q(y)] \quad \rightsquigarrow \quad [q(z),\ r(x),\ r(w)] \quad : \quad x + 2 < y \wedge x + 4 < z \wedge z < w$$

A valuation which satisfies the condition is $Val(x) = 1$, $Val(y) = 4$, $Val(z) = 8$, and $Val(w) = 10$. Then $[p(1), p(3), q(4)] \xrightarrow{\lambda(\rho)} [p(3), q(8), r(1), r(10)]$.

We can prove the following property.

Proposition 7. $\mathcal{L}^G(\nu\text{-}PN) \subseteq \mathcal{L}^G(CMRS)$

Proof. We have to simulate a ν-PN N by means of a CMRS N'. We assume that every transition of N can create at most one fresh name by means of the special variable $\nu \in \Upsilon$. Moreover, by using the standard technique of removing the final marking, we can assume that the final marking of N has a single token in a new place *accept*. We add a special predicate *next* to identify the next new identifier. At any point, the name in *next* contains a number which is greater than any other number used. For each t we have the rule with the same label as t:

$$[next(\nu)] + \sum_{p \in P} \sum_{x \in F(p,t)} [p(x)] \rightsquigarrow [next(\nu')] + \sum_{p \in P} \sum_{x \in F(t,p)} [p(x)] : \nu' > \nu$$

Notice that in N' we are recording the order in which the different identifiers have been created, though we do not record such order in N (that is, if a is created before b then $a < b$). However, since the final marking consists of a single name, such order is irrelevant.

Finally, we set the initial marking as follows. For every name $a \in Id(M_0)$, we consider a different $c_a \in \mathbb{N}$. Then, for every a-token in p, we consider a predicate $p(c_a)$. Moreover, we add $next(c)$, where c is greater than any c_a. The final marking is $[accept(c)]$, where $c \in \mathbb{N}$ is an arbitrary natural. $\square$

If we apply the previous construction to the ν-PN in Fig. 2, we obtain the following rule:

$$[p_1(x), p_1(y), p_2(y), next(\nu)] \rightsquigarrow [q_1(x), q_1(x), q_2(y), q_2(\nu), next(\nu')] : \nu' > \nu$$

In [5] it is proved that $\mathcal{L}^G(\nu\text{-PN}) \not\supseteq \mathcal{L}^G(\text{Data Nets})$. Then we can conclude the following.

Corollary 2. $\mathcal{L}^G(\nu\text{-}PN) \subset \mathcal{L}^G(\text{CMRS})$

5. Pure names with whole-place operations

In this section we extend ν-PN to deal with whole-place operations, like transfers, copies or resets. The consequence of adding whole-place operations varies between different models. For instance, adding whole-place operations to PN (thus obtaining Affine Well-Structured Nets, or AWN for short) yields a model which is strictly more expressive [11]. However, in the case of Data Nets, the models with and without such operations are equivalent [3]. In this section we prove that this is also the case for ν-PN. In the following, we denote by $\mathbf{0}$ the null tuple in any $\mathbb{N}^k$, and the null matrix in any $\mathbb{N}^{n\times m}$. We denote by $+$, $-$ and $\leqslant$ the component-wise sum, difference and order in any $\mathbb{N}^k$, respectively, and by $*$ the matrix multiplication.

Definition 7 ($w\nu$-PN). A *labelled $w\nu$-PN* is a tuple $N = (P, T, F, G, H, \lambda)$, where:

- P and T are finite disjoint sets of places and transitions, respectively;
- For each $t \in T$, there is a finite set $Var(t) \subseteq Var \backslash \Upsilon$ such that:
  - $F_t : Var(t) \to \mathbb{N}^P$ is the subtraction function,
  - $H_t : Var(t) \cup \Upsilon \to \mathbb{N}^P$ is the addition function, with finite support, that is, such that $H_t(\nu) \neq \mathbf{0}$ for finitely many $\nu \in \Upsilon$.
  - $G_t : Var(t) \times Var(t) \to \mathbb{N}^{P\times P}$ is the whole-place operations matrix.
- $\lambda : T \to \Sigma_\epsilon$ is the labelling function.

Notice that in $w\nu$-PN, $Var(t)$ does not contain variables in Υ, unlike for ν-PN. For each t, the subtraction function F_t is responsible of the removal of tokens. More precisely, when t is fired, $F_t(x)(p)$ tokens to which x is instantiated are removed from p. Similarly, H_t is responsible of the addition of tokens, but also of the creation of fresh names. Finally, G_t performs whole-place operations (after the removal of tokens). More precisely, for every x-token in p, $G_t(x, y)(p, q)$ y-tokens are put in q.

Definition 8 (Marking of $w\nu$-PN). A *marking* of N is an element of $(\mathbb{N}^P)^\oplus$.

Instead of considering names and allowing renaming, as we did for ν-PN, we are directly abstracting away from them, considering for every name a mapping in $\mathbb{N}^P$.[3] Assuming an arbitrary order in $P = \{p_1, ..., p_{|P|}\}$, we treat mappings in $\mathbb{N}^P$ as tuples in $\mathbb{N}^{|P|}$, and mappings in $\mathbb{N}^{P\times P}$ as matrices. Finally, we identify markings up to the addition/removal of $\mathbf{0}$, so that $M = M + \mathbf{0}$.[4]

Let us now define the behavior of a $w\nu$-PN.

Definition 9 (Enabling and firing in $w\nu$-PN). A *mode* for a transition t is any mapping $\sigma : Var(t) \to \mathbb{N}^P$. We say a transition t is *enabled* at a marking M if $M = \{\sigma(x) \mid x \in Var(t)\} + \overline{M}$ for some marking $\overline{M}$, so that $F_t(x) \leqslant \sigma(x)$

[3] In other words, markings of ν-PN modulo renaming can also be seen as elements in $(\mathbb{N}^P)^\oplus$.

[4] We do it in order to simplify our definitions. Alternatively, we could have just considered multisets over $\mathbb{N}^P \backslash \{\mathbf{0}\}$.

for each $x \in Var(t)$. In that case, t can be *fired*, reaching the marking $M' = \{M_x \mid x \in Var(t)\} + \sum_{\nu\in\Upsilon} H_t(\nu) + \overline{M}$, where

$$M_x = \sum_{y\in Var(t)} (\sigma(y) - F_t(y)) * G_t(x, y) + H_t(x)$$

As always, if M' is reached from M by firing t we write $M \stackrel{\lambda(t)}{\rightarrow} M'$. In particular, if for every $t \in T$, $G_t(x, y) = \mathbf{0}$ when $x \neq y$ and $G_t(x, x)$ is the identity matrix, then M_x boils down to $(\sigma(x) - F_t(x)) + H_t(x)$, thus obtaining a $\nu_{\neq}$-PN, so that they are subsumed by $w\nu$-PN. This means, by Prop. 4, that $\mathcal{L}^L(w\nu\text{-}PN) = \mathcal{L}^T(w\nu\text{-}PN) = RE$. However, a $w\nu$-PN can perform whole-place operations. For instance, if $G_t(x, y)(p, q) = 0$ for every $y \in Var(t)$ and every $p \in P$, the firing of t removes from q every name to which y is instantiated.

Example 5. Let us consider the $w\nu$-PN $N = (\{p_1, p_2\}, \{t\}, F, G, H, \lambda)$ defined as follows: $Var(t) = \{x_1, x_2\}$, $F_t(x_1) = F_t(x_2) = (1, 0)$, $H_t(x_1) = (0, 1)$, $H_t(x_2) = (0, 0)$, and $H_t(\nu) = (0, 1)$. We define G_t as follows:

$$G_t(x_1, x_1) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad G_t(x_1, x_2) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$G_t(x_2, x_1) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad G_t(x_2, x_2) = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

The value of $\lambda(t)$ is irrelevant for this example. Let $M = \{(1, 0), (3, 0)\}$ be a marking of N, containing two different tokens, one appearing once in p_1, and the other appearing three times in p_1. The transition t can be fired with the mode σ given by $\sigma(x_1) = (1, 0) \geqslant F_t(x_1)$ and $\sigma(x_2) = (3, 0) \geqslant F_t(x_2)$ (notice that in this case $\overline{M}$ in the previous definition is the empty marking). Then, the marking M' reached by the firing of t with that mode is $M' = \{M_{x_1}, M_{x_2}\} + H_t(\nu)$.

$$M_{x_1} = (\sigma(x_1) - F_t(x_1)) * G_t(x_1, x_1) + (\sigma(x_2) - F_t(x_2)) * G_t(x_1, x_2) + H_t(x_1) = (0, 1)$$

$$M_{x_2} = (\sigma(x_1) - F_t(x_1)) * G_t(x_2, x_1) + (\sigma(x_2) - F_t(x_2)) * G_t(x_2, x_2) + H_t(x_2) = (0, 0)$$

Therefore, $M' = \{(0, 1), (0, 0), (0, 1)\}$ which is equal to $\{(0, 1), (0, 1)\}$ (because we can always remove the empty tuple). Notice that t can also be fired with mode σ' given by $\sigma'(x_1) = (3, 0)$ and $\sigma'(x_2) = (1, 0)$, which produces a different result, namely $\{(2, 1), (0, 1)\}$.

The class of $w\nu$-PN belongs to WSTS with the canonical order in $(\mathbb{N}^P)^{\oplus}$, which is the multiset order induced by the component-wise order in $\mathbb{N}^P$, given by $m \leqslant m'$ iff $m(p) \leqslant m'(p)$ for every $p \in P$. Indeed, $w\nu$-PN can be seen as a subclass of Data Nets with fresh name creation as defined in [3], which are WSTS. Let us now see that this class does not actually extend the expressive power of ν-PN.

Proposition 8. $\mathcal{L}^G(w\nu\text{-}PN) = \mathcal{L}^G(\nu\text{-}PN)$

Proof. Since $w\nu$-PN extend ν-PN (as seen after Def. 9), clearly $\mathcal{L}^G(\nu\text{-PN}) \subseteq \mathcal{L}^G(w\nu\text{-PN})$. Let us see the converse, working with $\nu_{\neq}$-PN. In this proof we will specify $\nu_{\neq}$-PNs in a notation *à la CMRS.* For instance, the transition in Fig. 1 is denoted by $[p_1(x), p_1(y), p_2(y)] \to [p_3(x), p_3(\nu_1), p_4(\nu_1), p_4(\nu_2)]$.

Rationale The rationale behind the encoding of a $w\nu$-PN using ν-PN transitions is as follows. We first encode a $w\nu$-PN marking using an additional place called *Legal* that keeps track of the current set of names used in the ν-PN encoding of a configuration. If a name b is not contained in the place *Legal*, then all tokens with that name become dead (i.e. they cannot be moved by any transition). Using a CMRS-like notation, the $w\nu$-PN initial marking

$$[p(a), p(a), q(a), p(b), q(c)]$$

is encoded as

$$[run, p(a), p(a), q(a), p(b), q(c), legal(a), legal(b), legal(c)]$$

the *run* place is used to mark the beginning of a simulated firing step.

Each $w\nu$-PN transition t is encoded then using a set of ν-PN transitions that implement four distinct phases. As guide example, let us consider a transition t operating over two generic names x and y and generating a fresh one v. Transition t removes one token with name x from place p via F_t. Using G_t, for each token with name x in p, it adds two tokens with the same name x to p and one token with name v to q. Finally, it adds one token with name y to q.

In a first phase we non-deterministically select the names over which the transition operates (i.e. we must associate names to the generic variables x and y). In our example we can do this by using a transition like

$$[run, p(x), Legal(x), Legal(y)]) \to [sim_1, \iota_1(x), \iota_2(y), \iota_3(v)]$$

that applies F_t (it removes one token x from p), stores in ι_1 and ι_2 references to x and y, and generates a fresh name v whose reference is kept in ι_3. We remark that, by using distinct predicates ι_1 and ι_2, we can always refer to the correct name in the rest of the encoding (ι can be viewed as the representation of a valuation for the variables x, y).

We now have to simulate a multiplication step, i.e., for each token $p(x)$ we must generate the submarking $[p(x), p(x), q(v)]$.

This is the more subtle part of the encoding. We immediately notice that we cannot simply rewrite $[p(x)]$ into $[p(x), p(x)]$ without introducing potential cyclic rewriting steps. To avoid to fall into infinite rewritings, we proceed as follows. We first try to rename all the tokens with name x contained in p using a fresh name u. We do this while moving a token with name x from place p to a new place $\bar{p}$. Specifically, we first associate a fresh name u to x by using the transition

$$[sim_1, \iota_1(x)]) \to [sim_2, \iota_1(x), \zeta_1(u)]$$

and then start moving tokens from p to $\bar{p}$ while updating their name

$$[sim_2, p(x), \iota_1(x), \zeta_1(u)]) \to [sim_2, \bar{p}(u), \iota_1(x), \zeta_1(u)]$$

The copy phase is non-deterministically stopped by using the rule

$$[sim_2, \iota_1(x), \iota_2(y), \iota_3(v), \zeta_1(u)]) \to [sim_3, \zeta_1(u), \zeta_2(y), \zeta_3(v)]$$

that updates the current the set of selected names from x, y, v to u, y, v (stored in ζ_i for $i : 1, \ldots, 3$). Similar rules must be applied to copy tokens with name x occurring in other places (e.g. $q(a)$ to their corresponding copy-version (e.g. $\bar{q}(a'))$.

From now on all tokens with name x become unusable (i.e. they are dead). This implies that our encoding of the multiplication step is lossy. This however does not change the correspoding G-language if require that in the target configuration of the ν-PN encoding all names are declared as *legal*.

The effect of the previous $\nu - PN$ transitions on the initial configuration

$$[p(a), p(a), q(a), p(b), q(c), legal(a), legal(b), legal(c)]$$

is that of producing new configurations like

$$M_1 = [sim_3, p(a), \bar{p}(a'), \bar{q}(a'), p(b), q(c), \zeta_1(a'), \zeta_2(b), \zeta_3(d), legal(c)]$$

where $p(a)$ is a dead token. We can now apply G_t to $\bar{p}(a')$ as follows

$$[sim_3, \bar{p}(x), \zeta_1(x), \zeta_3(y)] \to [sim_3, p(x), p(x), q(y), \zeta_1(x), \zeta_3(y)]$$

With this rule for each token in place $\bar{p}$ we produce the specified number of tokens (with corresponding names) in all the other places (the rule can be applied only once to each $\bar{p}$-token). We use a similar rule to copy tokens from $\bar{q}$ to q:

$$[sim_3, \bar{q}(x), \zeta_1(x)] \to [sim_3, q(x), \zeta_1(x)]$$

When applied to configuration M_1 we obtain

$$M_2 = [sim_3, p(a), p(a'), p(a'), q(d), q(a'), p(b), q(c), \zeta_1(a'), \zeta_2(b), \zeta_3(d), legal(c)]$$

We can now simulate H_t by using the rule

$$[sim_3, \zeta_2(x)] \to [sim_4, \zeta_2(x), q(x)]$$

The encoding of transition t is terminated by updating the set of legal names and by returning to the *run* state:

$$[sim_3, \zeta_1(x), \zeta_2(y), \zeta_3(u)] \to [run, Legal(x), Legal(y), Legal(u)]$$

Its firing on marking M_2 produces

$$M_3 = [run, p(a), p(a'), p(a'), q(d), q(a'), p(b), q(c), Legal(a'), Legal(b), Legal(c), Legal(d)]$$

From now on, a new transition can be fired by using the subset of names a', b, c, d (i.e. $p(a)$ got lost during the simuation).

If we now consider a target marking M_f for G-acceptance in the original $\omega\nu$-PN like $[p(a), q(b)]$ we can just encoding as the the ν-PN marking $M'_f = [run, p(a), q(b), Legal(a), Legal(b)]$. *General Definition* We now define a general version of the lossy encoding in which several of the above mentioned steps are often merged in order to obtain more compact ν-PN transitions. We write F_t^x to denote the multiset given by $F_t^x(p(x)) = F_t(x)(p)$, and analogously for H_t^x. Moreover, we denote by $G_t^{x,p}(q(y)) = G_t(x,y)(p,q)$.
Initial/Target Markings: We encode the initial marking M_0 with names $\{a_1, \ldots, a_n\}$ as the new marking

$$M'_0 = M_0 + [run, Legal(a_1), \ldots, Legal(a_n)]$$

We encode the target marking M_f with names $\{a_1, \ldots, a_n\}$ a as the new marking

$$M'_f = M_f + [run, Legal(a_1), \ldots, Legal(a_n)]$$

G-reachability from M_0 to M_f in N is encoded as G-reachability in the ν-PN N' defined as follows.

We simulate the firing of a transition t with $Var(t) = \{x_1, ..., x_k\}$ in four steps: subtraction, copy, multiplication and addition. In order to identify the current step, we will use places run, $copy_t$, $mult_t$ and add_t, for each $t \in T$.
Subtraction: In the subtraction phase we have to choose, in a non-deterministic way, which names are chosen for the firing of the transition. We have places $\iota_1, \ldots, \iota_k$, so that a token a in ι_i represents the fact that a has been the i-th name chosen for the firing. The places $\zeta_1, \ldots, \zeta_k$ are used in the copy phase. In them, we create k fresh names, that will replace the ones to which $x_1, ..., x_k$ are instantiated.

$$[run] + \sum_{i=1}^{k}(F_t^{x_i} + [Legal(x_i)]) \to \sum_{i=1}^{k}[\iota_i(x_i), \zeta_i(\nu_i)] + [copy_t]$$

Copy: In this phase we create a distinct copy of the tokens that carry one of the names chosen. For that purpose, we use the places $\zeta_1, \ldots, \zeta_k$. For each $p \in P$ and $i \in [k]$ we consider the following transition:

$$[copy_t, p(x), \iota_i(x), \zeta_i(y)] \to [copy_t, \bar{p}(y), \iota_i(x), \zeta_i(y)]$$

Intuitively, the previous transition replaces the name in p (stored in ι_i) by the fresh name in ζ_i. Hence, the repeated firing of the previous transition can create a fresh copy (in the new places of the form $\bar{p}$) of the part of the marking selected for the firing of t. The following transition ends the copying:

$$[copy_t] + \sum_{i=1}^{k}[\iota_i(x_i), \zeta_i(y_i)] \to [mult_t] + \sum_{i=1}^{k}[\iota_i(y_i), Legal(y_i)]$$

In particular, for every $i \in \{1, .., k\}$ it moves the token in ζ_i to $Legal$. Notice that this last transition can be fired before all the tokens have been copied,

so that we simulate a lossy version of the $w\nu$-PN which is irrelevant for G-languages.
Multiplication: In the multiplication phase we simulate the effect of G_t. For that purpose, for each $p \in P$, and each $l \in [k]$, we consider the following transition:

$$[mult_t, \bar{p}(x_l)] + \sum_{i=1}^{k}[\iota_i(x_i)] \to [mult_t] + \sum_{i=1}^{k}[\iota_i(x_i)] + G_t^{x_l,p}$$

By firing the previous transition, for every x_l-token in $\bar{p}$, $G_t(p,q)(x_l,x_i)$ x_i-tokens are put in q. The next transition ends the multiplication phase, possibly before all the tokens have been processed:

$$[mult_t] \to [add_t]$$

Again, the last rule could be applied when there are still remaining $\bar{p}(x)$ to consider, so this part of the simulation is again lossy, by the same reasons of the previous step.
Addition: Finally, we just simulate the effect of H_t, in particular creating fresh names as demanded by H, which are legal and therefore put in $Legal$.

$$[add_t] + \sum_{i=1}^{k}[\iota_i(x_i)] \to [run] + \sum_{i=1}^{k} H_t^{x_i} + \sum_{\substack{\nu \in \Upsilon \\ H_t(\nu) \neq \mathbf{0}}} (H_t(\nu) + [Legal(\nu)])$$

Moreover, the previous transition removes all the tokens in the ι_i places, and the token in add_t is moved to run, hence finishing the simulation of t. Of all the transitions we have used to simulate t, only one (the last one, for instance) has a label different from ϵ, and equal to the label of t. The initial marking of the ν-PN extends the one of the $w\nu$-PN with a token in run, a single copy of every token in $Legal$, and empty elsewhere. The final marking extends the final marking of the $w\nu$-PN with a token in run, a single copy of every token in $Legal$, and empty elsewhere. □

6. Pure Names vs Black Tokens

In this section we compare ν-PNs with AWNs [10], a well structured extension of Petri nets that allows whole-place operations. An *affine well-structured net* (AWN) N is given by a set of n places and a set of transitions. Each transition comes equipped with two n-vectors, F_t and H_t, and an $n \times n$-matrix G_t. A marking M of an AWN must specify how many (black) tokens are there in each place, so that it is also an n-vector. We compare n-vectors (markings in particular) with the component-wise order $\leqslant$. A transition t can be fired whenever $F_t \leqslant M$, and the reached marking after the firing is $M' = (M - F_t) * G_t + H_t$. The matrices G_t are responsible for the whole place operations. For instance, if the i-th column of G_t is null, then G_t resets the i-th place, that is, it empties its content. If G_t is the identity matrix for all t, then N is a PN.

Let us now see that ν-PN are more expressive that AWN.

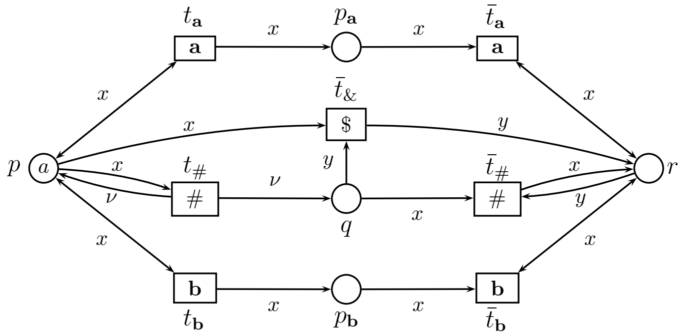

Figure 8: N_Σ with $\Sigma = \{\mathbf{a}, \mathbf{b}\}$

Proposition 9. $\mathcal{L}^G(AWN) \subseteq \mathcal{L}^G(\nu\text{-}PN)$

Proof. It is enough to consider that ν-PN are equivalent to $w\nu$-PN, and that AWN are subsumed by $w\nu$-PN. Indeed, an AWN is a simple $w\nu$-PN for which $Var(t)$ is a singleton for each $t \in T$, and so that $H_t(\nu) = \mathbf{0}$ for every $\nu \in \Upsilon$. □

Let us now see that the previous inclusion is a strict one. We have to find a language recognized by some ν-PN, but not recognized by any AWN. We apply a proof scheme introduced in [1] for comparing other models, that in particular uses an order that compares words using their Parikh images.

Definition 10. Let Σ be any alphabet not containing the symbols # and \$. We define the order $\leqslant$ over Σ^* as $\mathbf{a}_1...\mathbf{a}_n \leqslant \mathbf{b}_1...\mathbf{b}_m$ iff $\{\mathbf{a}_1, ..., \mathbf{a}_n\} \leqslant^{\oplus} \{\mathbf{b}_1, ..., \mathbf{b}_m\}$. Let S_Σ be the set of words of the form $w_1\# \ldots \# w_n$, with $w_i \in \Sigma^*$. We define $w_1\# \ldots \# w_n \leqslant v_1\# \ldots \# v_m$ if there is an injection $h : [n] \to [m]$ such that $w_i \leqslant v_{h(i)}$ for every $i \in [n]$. Finally, we define $L_\Sigma = \{s\$s' \mid s, s' \in S_\Sigma,\ s' \leqslant s\}$.

Let us first see that L_Σ is in $\mathcal{L}^G(\nu\text{-PN})$.

Proposition 10. *For any Σ there is a ν-PN N_Σ such that $\mathcal{L}^G(N_\Sigma) = L_\Sigma$.*

Proof. We define $N_\Sigma = (P, T, F, \lambda)$ as follows (Fig. 8 shows N_Σ for $\Sigma = \{\mathbf{a}, \mathbf{b}\}$):

- $P = \{p, q, r\} \cup \{p_{\mathbf{a}} \mid \mathbf{a} \in \Sigma\}$,
- $T = \{t_{\mathbf{a}}, \bar{t}_{\mathbf{a}} \mid \mathbf{a} \in \Sigma \cup \{\#\}\} \cup \{t_\$\}$,
- $F(p, t_{\mathbf{a}}) = F(t_{\mathbf{a}}, p) = F(t_{\mathbf{a}}, p_{\mathbf{a}}) = F(p_{\mathbf{a}}, \bar{t}_{\mathbf{a}}) = F(\bar{t}_{\mathbf{a}}, r) = F(r, \bar{t}_{\mathbf{a}}) = \{x\}$ for all $\mathbf{a} \in \Sigma$,
- $F(p, t_\#) = F(t_\#, q) = F(q, \bar{t}_\#) = F(\bar{t}_\#, r) = \{x\}$ and $F(t_\#, p) = \{\nu\}$,
- $F(q, t_\$) = F(t_\$, r) = F(r, \bar{t}_\#) = \{y\}$.

Moreover, $t_{\mathbf{a}}$ and $\bar{t}_{\mathbf{a}}$ are labelled by $\mathbf{a}$ for every $\mathbf{a} \in \Sigma \cup \{\#\}$, and $t_\$$ is labelled by \$. The initial marking is M_0 given by $M_0(p) = M_0(q) = \{a\}$ and empty elsewhere, and the final marking is M_f with $M_f(r) = \{a\}$ and empty elsewhere.

Let us see that N_Σ accepts L_Σ. Intuitively, a different name is used to represent each w_i in $w_1\#...\#w_n \in S_\Sigma$. While generating w_i, that name is stored in p. If w_i contains k $\mathbf{a}$'s, and the identifier a is being used to represent w_i, then N_Σ stores in $p_{\mathbf{a}}$ k a-tokens. Moreover, every time $t_\#$ is fired, the name in p is replaced by a fresh one, which is also put in q. Hence, a different name is used to represent w_{i+1}, and in q we store one copy of all the names used (notice that the first one used is already initially both in p and q).

At any point, after producing some $s \in S_\Sigma$ as output, N_Σ can fire $t_\&$, moving one token from q to r. During this new phase, it can produce any $s' \in S_\Sigma$ with $s' \leqslant s$. If a is the first name put in r, and w is the word represented by a, then N_Σ can output any $w' \in \Sigma^*$ with $w' \leqslant w$ (with the order in Def. 10), just by firing the transitions $\bar{t}_{\mathbf{a}}$. Notice that if w contains k $\mathbf{a}$'s, then k a-tokens were put in $p_{\mathbf{a}}$. Therefore, $\bar{t}_{\mathbf{a}}$ can be fired at most k times. Moreover, every time $\bar{t}_\#$ is fired the name in r is replaced by another name taken from q. Since the final marking is that with one token in r, all the words generated during this second phase are in the G-language of N_Σ, so that it accepts L_Σ. □

Let us see that L_Σ is not the language of any AWN, for some alphabets Σ.

Proposition 11. *There is Σ_0 such that $L_{\Sigma_0} \notin \mathcal{L}^G(AWN)$.*

Proof. Let $\Sigma_0 = \{a, b, 0, 1\}$, $S = S_{\Sigma_0}$ and $L_0 = L_{\Sigma_0}$. We prove that L_0 is not the G-language of any AWN. The proof is per absurdum. Suppose there exists an AWN N that recognizes L_0 with initial marking M_{init} and accepting marking M_f. Assume that N has places $p_1, \ldots, p_n$.

We first notice that, for any $s \in S$, $s\$s \in L_0$. Under our hypothesis, we have then that, for each $s \in S$, there is a marking M_s such that $M_{init} \xrightarrow{s\$} M_s \xrightarrow{s} M$ and $M_f \leqslant M$. Consider the sequences $s_0, s_1, s_2, \ldots$ and $M_{s_0}, M_{s_1}, M_{s_2}, \ldots$ of words in S and markings of N, respectively, defined as follows:

- $s_0 := b\#b \ldots \#b$ with n occurrences of b;
- If $M_{s_i} = (m_1, \ldots, m_n)$ then $s_{i+1} := a^{m_1}q_1\# a^{m_2}q_2\# \cdots \# a^{m_n}q_n$, for $i = 0, 1, \ldots$, where $q_1, \ldots, q_n$ are unary encodings of the positions $1, \ldots, n$ over n bits, i.e., $q_1 = 10\cdots0$, $q_2 = 110\cdots0$, $\ldots$, $q_n = 111\cdots1$.

Since b occurs only in s_0, $s_0 \nleqslant s_i$ for all $i > 0$. Furthermore, for any $i < j$, $M_{s_i} \leqslant M_{s_j}$ iff $s_{i+1} \leqslant s_{j+1}$. This holds because s_{i+1} and s_{j+1} have both $n-1$ occurrences of the separator $\#$ and because any injection needed in the definition of $\leqslant$ is forced to preserve positions (their unary representation) in our encoding of markings. Since the marking order is a wqo, there exist i, j such that $i < j$ and $M_{s_i} \leqslant M_{s_j}$. Now let j be the smallest natural number satisfying this property. Then, we have that $M_{s_{i-1}} \nleqslant M_{s_{j-1}}$ and $s_i \nleqslant s_j$ for $i > 0$. Furthermore, since by definition $s_0 \nleqslant s_j$, we have that $s_i \nleqslant s_j$ for any $i \geqslant 0$. Since $M_{s_i} \leqslant M_{s_j}$, by monotonicity of AWNs, we have that $M_{s_i} \xrightarrow{s_i} M$ with $M_f \leqslant M$ implies that $M_{s_j} \xrightarrow{s_i} M'$ with $M_f \leqslant M \leqslant M'$. Hence, we obtain $M_{init} \xrightarrow{s_j\$s_i} M'$ and $s_j\$s_i \in \mathcal{L}^G(N)$ which is in contradiction with the hypothesis that $\mathcal{L}^G(N) = L_0$, since $s_i \nleqslant s_j$. $\square$

Summing up, we obtain the following relation between the G-languages of all the models considered in the paper.

Corollary 3.

$$\mathcal{L}^G(AWN) \subset \mathcal{L}^G(\nu\text{-}PN) = \mathcal{L}^G(\nu_{\neq}\text{-}PN) = \mathcal{L}^G(w\nu\text{-}PN) \subset \mathcal{L}^G(\text{CMRS})$$

7. Closure Properties of $\mathcal{L}^G(\nu\text{-PN})$

In this section, we briefly study several closure properties of the class of G-languages generated by ν-PNs. The family of languages $\mathcal{L}^G(\nu\text{-PN})$ satisfies a good number of closure properties, which are summarized in the following result.

Proposition 12. *$\mathcal{L}^G(\nu\text{-}PN)$ is a semi-full AFL closed under concatenation and intersection.*

Proof. We have to see that it is closed under intersection, concatenation, union, homomorphism, and inverse homomorphism. Let $L_1 = \mathcal{L}^G(N_1)$ and $L_2 = \mathcal{L}^G(N_2)$ with N_1 and N_2 ν-PN. We assume that the set of variables used in N_1 and N_2 are disjoint. Let $N_1 = (P_1, T_1, F_1, \lambda_1)$ with initial and final marking M_1 and M_1^f, respectively. Analogously, let $N_2 = (P_2, T_2, F_2, \lambda_2)$ with initial and final marking M_2 and M_2^f, respectively. We can safely assume that the set of names in N_1 and N_2 are disjoint. We need the following notations: If f_1 and f_2 are two mappings defined over two disjoint sets A_1 and A_2, we define $f_1 + f_2$ as the mapping given by $(f_1 + f_2)(a) = f_i(a)$ if $a \in A_i$, for $i \in \{1, 2\}$.

- **Intersection**: We build $N = (P_1 \cup P_2, T, F, \lambda)$, with initial marking $M_1 + M_2$, and final marking $M_1^f + M_2^f$, that accepts $L_1 \cap L_2$. We take $T = \{t \in T_1 \mid \lambda_1(t) = \epsilon\} \cup \{t \in T_2 \mid \lambda_2(t) = \epsilon\} \cup \{(t_1, t_2) \mid t_i \in T_i,\ \lambda(t_1) = \lambda(t_2) \neq \epsilon\}$ with their obvious labellings. Finally, $F(p, t) = F_1(p, t)$ if $t \in T \cap T_1$ (analogously if $t \in T \cap T_2$), and $F(p, (t_1, t_2)) = F_1(p, t_1)$ if $p \in P_1$, or $F(p, (t_1, t_2)) = F_2(p, t_2)$ if $p \in P_2$. Analogously, we define $F(t, p)$ and $F((t_1, t_2), p)$.
- **Concatenation**: For L_1L_2 we consider two fresh places p_1 and p_2, and build $N = (P_1 \cup P_2 \cup \{p_1, p_2\}, T_1 \cup T_2 \cup \{t\}, F, \lambda)$, with initial marking $M_1 + M_2$, plus a token in p_1, and final marking M_2^f, plus a token in p_2. F is such that the transitions in T_1 can fire when p_1 is marked, while the ones in T_2 can fire when p_2 is marked. Moreover, F is such that the new

transition t (labelled by ϵ) removes M_1^f and moves the token in p_1 to p_2. Hence, N behaves as N_1 until its final marking is covered, and then it can behave like N_2.

- **Union**: The construction follows the same ideas as the previous case, so we do not go into the details. We add control places to both N_1 and N_2, preconditions and postconditions of every transition of the corresponding ν-PN. Then we add a non-deterministic choice between two new transitions, marking the control place of N_1 or that of N_2.

- **Homomorphism**: If for a symbol $\mathbf{a}$, $h(\mathbf{a})$ is some symbol or ϵ, then it is enough to rename labels accordingly. Otherwise, if $h(\mathbf{a}) = \mathbf{a}_1 \dots \mathbf{a}_n$ with $n > 1$ we just have to expand every transition t labelled by $\mathbf{a}$ into n transitions $t_1, \dots, t_n$, labelled by $\mathbf{a}_1, \dots, \mathbf{a}_n$, respectively, fired sequentially with the help of new control places.

- **Inverse homomorphism**: Similarly as in the previous case, if $h(\mathbf{a}) = \mathbf{a}_1 \dots \mathbf{a}_n$ then we have to build N' that outputs $\mathbf{a}$ every time N outputs $\mathbf{a}_1 \dots \mathbf{a}_n$. For that purpose, we add again control places to "detect" when that sequence is fired in a row (with labels renamed to ϵ), in which case it fires an extra transition labelled by $\mathbf{a}$ (or equivalently, synchronize the ν-PN with a finite automaton for $h(\Sigma)$). □

Therefore, the families of G-languages recognized by ν-PNs, with coverability as accepting condition, are semi-full AFLs, but we do not know if they are also full AFLs, since we have not proved whether they are closed under iteration.

On the other hand, it is easy to see that they are not closed under complement. Indeed, there is a language accepted by some PN, but the complement of this language is not even a WSL [11]. For instance, $\{a^n b^m \mid m \leqslant n\}$ is easily accepted by some PN, but its complement is not a WSL, which can be seen by using the pumping lemma for WSL [11]. However, we can prove the following.

Proposition 13. *The following holds:*

1. *If $L \in \mathcal{L}^G(AWN)$ then $L^+ \in \mathcal{L}^G(AWN)$.*
2. *If $L \in \mathcal{L}^G(PN)$ then $L^+ \in \mathcal{L}^G(AWN)$.*
3. *If $L \in \mathcal{L}^G(PN)$ then $L^+ \in \mathcal{L}^G(\nu\text{-}PN)$.*

Proof. (2) and (3) follow immediately from (1), considering that $\mathcal{L}^G(\text{PN}) \subset \mathcal{L}^G(\text{AWN}) \subset \mathcal{L}^G(\nu\text{-}PN)$. For (1), given an AWN N with initial and final markings m_0 and m_f, respectively, it is enough to add a transition t^* with $F_{t*} = m_f$, $G_{t*} = \mathbf{0}$ and $H_{t*} = m_0$, labelled by ϵ, that can be fired from any marking that covers m_f, producing m_0. □

It would be interesting to see what happens with iteration for $\mathcal{L}^G(\nu\text{-PN})$. We know that the class of G-languages of Petri nets is not closed under iteration [11]. We conjecture that this is also the case for ν-PN. The intuitive reasoning is the same as for $\mathcal{L}^G(\text{PN})$, namely the fact that by means of coverability we cannot distinguish between different "executions" within the same net (we cannot throw away arbitrary garbage). Nevertheless, we can prove that the iteration of the G-language of a ν-PN is the G-language of a CMRS. We obtain it as a corollary of the following result.

Proposition 14. *The class of G-languages of CMRS is closed under iteration.*

Proof. Let L be the language accepted by a CMRS S with predicate symbols in $\mathbb{P}$. We build a CMRS S' that accepts L^+ as follows. We introduce two new predicates *Left* and *Right* used to maintain the interval of values used in a computation. Without loss of generality, we assume that S has no constraints of the form $> c, < c, = c$ for any constant c (but we allow constraints of the form $x + c < y$) and the initial configuration is the CMRS configuration $[init]$ and the accepting configuration $[accept]$. S' has the rule

$$[init'] \rightsquigarrow [Left(x), init, Right(y)] \ : \ x < y$$

With this rule, we guess the interval of values needed to accept words in L. Furthermore, for each rule $M \rightsquigarrow M' : \psi$ in S with variables $x_1, \ldots, x_n$ we add to S' the rule

$$M + [Left(x), Right(y)] \rightsquigarrow M' + [Left(x), Right(y)] \ : \ \psi \wedge \bigwedge_{i:1,\ldots,n} x < x_i, x_i < y$$

With these rules, we require all values occurring in a computation to be within the guessed interval.

Finally, we add to S' the following rule

$$[accept, Left(x), Right(y)] \rightsquigarrow [Left(x'), init, Right(y')] \ : \ y < x' < y'$$

With this rule we guess a new, fresh interval and restart the computation. Using this idea, every time we accept a word $w \in L$, we can restart S without running the risk of confusing values with those used in the previous computations. Thus, S' can repeat the accepting run for any word in the language of S an arbitrary number of times. $\square$

Corollary 4. *If $L \in \mathcal{L}^G(\nu\text{-}PN)$, then $L^+ \in \mathcal{L}^G(CMRS)$.*

Proof. Since $\mathcal{L}^G(\nu\text{-PN}) \subset \mathcal{L}^G(CMRS)$, it is enough to consider the previous result.

8. Decision problems of $\mathcal{L}^G(\nu$-PN)

Before studying the decision problems of $\mathcal{L}^G(\nu\text{-PN})$, we introduce some concepts and notations we will need in this section. Given a quasi order $(A, \leqslant)$ and $B \subseteq A$, the *upward closure* of B is $B\!\uparrow = \{a \in A \mid \text{there is } b \in B \text{ s.t. } b \leqslant a\}$. Analogously, we define $B\!\downarrow$, the *downward closure* of B. We say B is *upward closed (downward closed)* if $B = B\!\uparrow$ $(B = B\!\downarrow)$. A *basis* of an upward closed set

B is any $C \subseteq$ such that $C\!\uparrow = B$. It is well known that any upward closed set has a finite basis when $\leqslant$ is a wqo.

Clearly, the emptiness problem, that of deciding given a ν-PN N whether $\mathcal{L}^G(N) = \emptyset$, is decidable, since this is the case for all effective WSTS. A WSTS is effective if it is finitely-branching, the underlying wqo is decidable, and a finite basis of the set of predecessors of an upward-closed set of states is always computable. All the WSTS considered in this paper are effective. Checking emptiness amounts to deciding a coverability problem, which is decidable for all effective WSTS. Moreover, we can prove the following:

Proposition 15. *The following problems are decidable, with a non-primitive recursive complexity: (1) emptiness, (2) empty intersection, that is, the problem of deciding, given two ν-PN N_1 and N_2, whether $\mathcal{L}^G(N_1) \cap \mathcal{L}^G(N_2) = \emptyset$, and (3) membership.*

Proof. They all follow from the fact that coverability for ν-PN is decidable with a non-primitive recursive complexity [24]. (1) The emptiness problem for $\mathcal{L}^G(\nu\text{-PN})$ is just coverability for ν-PN. (2) For decidability it is enough to consider that $\mathcal{L}^G(\nu\text{-PN})$ is closed under intersection, and that emptiness is decidable for it. For the hardness, notice that the emptiness problem can be reduced to the problem of empty intersection just by taking any N_2 that accepts Σ^*, so that the G-language of N is empty iff $\mathcal{L}^G(N) \cap \mathcal{L}^G(N_2) = \emptyset$. (3) Given $w \in \Sigma^*$ and a ν-PN N, $w \in \mathcal{L}^G(N) \Leftrightarrow \{w\} \cap \mathcal{L}^G(N) \neq \emptyset$. Since we can clearly build a ν-PN accepting $\{w\}$, by the previous item we are done for decidability. For hardness, notice that the emptiness problem can be reduced to the membership problem, just by setting $\lambda(t) = \epsilon$ for all $t \in T$, so that $\mathcal{L}^G(N) = \emptyset$ iff $\epsilon \notin \mathcal{L}^G(N')$ (where N' is the modified ν-PN). $\square$

Unfortunately, the rest of the problems we will study are undecidable. Universality, the problem of deciding whether $\mathcal{L}^G(N) = \Sigma^*$ for some given ν-PN N, is undecidable, since this is already the case for Petri nets extended with non-blocking arcs, whose expressiveness lies in between PN and AWN [11]. Then, as a direct consequence of [11], the problem of deciding whether a given ν-PN and a given finite automaton, is undecidable.

Proposition 16. *Given a ν-PN N and a regular language L, it is undecidable whether $\mathcal{L}^G(N) = L$.*

We remark that the previous result, in the case of PN, is decidable. More precisely, the problem of deciding whether a given PN is equivalent to a given finite automaton is decidable [17]. In [17], the authors also prove that the regularity problem, that of deciding whether $\mathcal{L}^G(N)$ is regular for a given N, is already undecidable for PN (even without ϵ-labelling). Therefore, the same holds for every model above PN, and in particular for ν-PN.

Proposition 17. *The regularity problem, that of deciding whether $\mathcal{L}^G(N)$ is regular for a given ν-PN N, is undecidable.*

Here we complete our view in two ways. First we prove that it is undecidable whether the G-language of a ν-PN is the G-language of some PN. Actually, we obtain it as a corollary (Cor. 5) of the analogous result for AWN (Prop. 18). Then, in Prop. 19 we will prove that it is undecidable whether the G-language of a given ν-PN is the G-language of some AWN.

In the following results we will use the fact that place boundedness is undecidable for AWN [8] and for ν-PN [24]. The place boundedness problem for AWN consists in deciding, given an AWN N and a place p of N, whether there is k such that any reachable marking m satisfies $m(p) \leqslant k$. Place boundedness for ν-PN consists in deciding, given a ν-PN N and a place p of N, whether there is k such that every reachable marking M satisfies $|M(p)| \leqslant k$.

Proposition 18. *Given an AWN N, it is undecidable whether $\mathcal{L}^G(N)$ is the G-language of some PN.*

Proof. We reduce the place boundedness problem for AWN. Given an AWN N and a place p, we build N' such that p is unbounded in N if and only if $\mathcal{L}^G(N')$ is the G-language of some PN. More precisely:

- If p is bounded in N then $\mathcal{L}^P(N')$ is regular, hence the language of some PN.
- If p is unbounded in N then $\mathcal{L}^P(N') = \{\mathbf{a}^n\mathbf{b}^m \mid m \leqslant n\}^+$, which is not the G-language of any PN [11].

We build N' starting from N as follows:

- We relabel every transition of N by ϵ.
- We add a place *run*, initially marked, which is a precondition/postcondition of every transition in N.
- We add a transition that at any point can move the token in *run* to a new place *stop*1.
- When *stop*1 is marked, we can move one by one the tokens in p to a new place p' by means of a new transition t_1 (labelled by $\mathbf{a}$); moreover, at any time the token in *stop*1 can be transferred to a new place *stop*2.
- When *stop*2 is marked, we can remove one by one the tokens in p' by means of a new transition t_2 (labelled by $\mathbf{b}$); moreover, at any time a transition that resets all the places in the net can be fired, also setting the initial marking of N, and putting a token in *run*.

Transitions other than t_1 and t_2 are labelled by ϵ. Notice that we are considering the P-language of N', so that we do not have to specify the final marking. N' as defined above satisfies the previous conditions. Indeed, if p is bounded in N, there is $k \geqslant 0$ such that any reachable marking m satisfies $m(p) \leqslant k$. Then, t_1 can be fired at most k times consecutively, and the language accepted by

N' is $\{\mathbf{a}^i\mathbf{b}^j \mid j \leqslant i \leqslant k\}^+$, which is regular because $\{\mathbf{a}^i\mathbf{b}^j \mid j \leqslant i \leqslant k\}$ is finite. Conversely, if p is unbounded, for any $n \in \mathbb{N}$ there is some reachable marking m in N such that $m(p) \geqslant n$. Then, for any $n \in \mathbb{N}$ the transition t_1 can be fired n times consecutively. Therefore, the P-language accepted by N' is $\{\mathbf{a}^i\mathbf{b}^j \mid j \leqslant i\}^+$, and we are done. □

Corollary 5. *Given a ν-PN N, it is undecidable whether $\mathcal{L}^G(N)$ is the G-language of some PN.*

Let us now see the analogous result for ν-PN and AWN.

Proposition 19. *Given a ν-PN N, it is undecidable whether $\mathcal{L}^G(N)$ is the G-language of some AWN.*

Proof. We reduce the place boundedness problem for ν-PN. Given a ν-PN N and a place p, we build N' such that p is unbounded in N if and only if $\mathcal{L}^G(N')$ is the G-language of some AWN. By Prop. 10 and Prop. 11 there is L_Σ which is the G-language of some ν-PN N_Σ, that is not the G-language of any AWN. We build N' such that:

- If p is bounded in N then $\mathcal{L}^G(N')$ is finite, hence the G-language of some AWN.
- If p is unbounded in N then $\mathcal{L}^G(N') = L_\Sigma$, which is not the G-language of any AWN.

We build N' starting from N as follows. We relabel every transition of N by ϵ. We add a place *run* (initially marked) and a place *stop* (initially empty). The place *run* is a precondition/postcondition of all the transitions in N. We add to N' a disjoint copy of the net N_Σ. Moreover, *stop* is a precondition/postcondition of all the transitions in N_Σ, and p is a precondition of all the transitions in N_Σ. Then, every transition of N_Σ removes a token from p. Finally, the final marking of N' is that with a token in the place r of N_Σ. Let us see that N' satisfies the previous conditions. If p is bounded in N, then there is $k \geqslant 0$ such that any marking m reachable in N satisfies $|m(p)| \leqslant k$. Then, the length of the words accepted by N' is at most k, so that $\mathcal{L}^G(N')$ is finite. Conversely, if p is unbounded in N, then N' can accept words in L_Σ of an arbitrary length, that is, it can accept the whole language. □

We conclude the paper with a result regarding the downward-closure of R-languages, for $R \in \{L, G, T, P\}$. Given an arbitrary language $L \subseteq \Sigma^*$, and a quasi-order $\leqslant$, we can define its downward closure $L{\downarrow} = \{u \in \Sigma^* \mid u \leqslant v \in L\}$. If $\leqslant$ is the embedding order, or Higman's order, then it is well known that for an arbitrary language L, $L{\downarrow}$ is regular. Indeed, $\leqslant$ is a wqo [16], so that the complement of $L{\downarrow}$ (which is upward closed) has a finite basis, and it is easy to devise from it a regular expression that generates it. Hence, it is regular and so is $L{\downarrow}$.

We now address the problem of computing, given a ν-PN N, a regular expression E such that $\mathcal{L}^R(N)\!\downarrow = L(E)$. Unfortunately, we will see that this regular expression, even if it always exists, cannot be computed for ν-PN (neither for AWN), even in the case of ϵ-free and injective labellings. This fact contrasts with PN, for which that regular expression can always be computed [14].

We introduce some notations. Given a language $L \subseteq \Sigma^*$ and $\Sigma' \subseteq \Sigma$, we define $L|_{\Sigma'}$ as the image of L through the homomorphism h given by $h(a) = a$ if $a \in \Sigma'$, and $h(a) = \epsilon$ if $a \notin \Sigma'$. In words, $L|_{\Sigma'}$ is the language that results from removing from every word of L every occurrence of symbols not in Σ'. Since $L|_{\Sigma'}$ is the image of L through a homomorphism, if L is regular so is $L|_{\Sigma'}$. Moreover, by using standard techniques, given a regular expression that generates L, one can compute a regular expression that generates $L|_{\Sigma'}$.

Again, we will see our result for ν-PN as a corollary of the corresponding result for AWN.

Proposition 20. *Given an AWN N with an ϵ-free and injective labelling, a regular expression E such that $\mathcal{L}^R(N)\!\downarrow = L(E)$ is not computable, for $R \in \{P, G, L, T\}$.*

Proof. Again, we reduce place boundedness to the computation of E. Assume by contradiction that we can always compute such regular expression. Given an AWN N and a place p of N, we build a labelled AWN N' (with ϵ-free and injective labelling) as follows:

- We add a place *run*, initially marked, which is a precondition/postcondition of every transition in N.
- We add a new transition $\bar{t}$, that can move a token from *run* to *stop*.
- We add a transition t_p that can remove a token from p when *stop* is marked.

Any word in $\mathcal{L}^P(N')\!\downarrow$ starts with a subword of $\mathcal{L}^P(N)$, possibly followed by $\bar{t}$, followed by a word in t_p^*. Let E be the regular expression such that $\mathcal{L}^P(N')\!\downarrow = L(E)$. As mentioned above, we can compute E' such that $L(E') = L(E)|_{t_p}$. Then, p is unbounded if and only if $L(E')$ is infinite. Since the latter can be decided, we can decide boundedness of p, thus reaching a contradiction. This concludes the proof for P-languages and therefore also for G-languages.

Let us now see it for L-languages and T-languages. We slightly modify the previous construction: instead of adding one transition t_p that removes a token from p when *stop* is marked, we add a transition t_q for each place $q \in P$ that can remove a token from q when *stop* is marked. Let us remark that the only dead marking in N' is that with a token in *stop* and empty elsewhere. Hence, if we consider that marking as final marking, we have $\mathcal{L}^L(N') = \mathcal{L}^T(N')$. The proof proceeds exactly as above. If E accepts the downward-closure of $\mathcal{L}^L(N')$ then we can compute E' that generates $L(E)|_{t_p}$, so that p is unbounded if and only if $L(E')$ is infinite. □

Corollary 6. *Given a ν-PN N with an ϵ-free and injective labelling, a regular expression E such that $\mathcal{L}^R(N)\!\downarrow = L(E)$ is not computable, for $R \in \{P, G, L, T\}$.*

9. Conclusions and Open Problems

The study of the expressive power of computation models in between Petri nets and Turing machines, and in particular of the class of well-structured transition systems, is a challenging research problem with several open questions. In this paper we have extended the classification of well-structured transition systems studied in [18, 1, 2] by comparing infinite-state models like *Affine Well-structured Nets* (AWN) [10], Data nets [18], and CMRS [4] with ν-PN, an extension of Petri nets in which tokens are pure names [24]. In [1, 2, 3] coverability acceptance is chosen in order to give a strict hierarchy for the expressive power of Petri nets, affine well structured nets, and CMRS. In [2, 3] it is proved that CMRS and Data nets define the same class of languages and that Data nets extended with name creation (i.e. selection of data that must be fresh) are equivalent to CMRS/Data nets. In the present paper we have extended to ν-PN the hierarchy of well-structured systems.

We can conclude that pure names can simulate whole-place operations with black tokens, even whole-place operations performed on the set of names chosen for the firing of a transition. Moreover, having pure names gives us strictly more expressive power than having whole-place operations on black tokens. We have also seen that the class of G-languages of ν-PN satisfy a good number of closure properties. However, when we disallow name matching then their expressive power boils down to that of Petri nets.

Concerning open problems, we conjecture that ν-PN and lossy FIFO channel systems [6] define incomparable classes of G-languages. In [5] a framework to prove non-inclusions of classes of G-languages is defined. It relies on the *order type* of the underlying state space. However, even if the state space of LCS and ν-PN are quite different, their order types can be the same. Another open problem is whether the class of G-languages of ν-PN is closed under iteration. Even if we have proved that adding whole-place operations to ν-PN does not add any expressive power, this may no be longer true if we also consider broadcasts, that is, operations in which an unbounded number of names are involved. We claim that such extension with broadcasts is closed under iteration, so a possible way to address the problem of iteration is to compare both classes. Finally, the distinction between unordered Petri Data nets (the unordered version of Data nets without whole-place operations and broadcasts), in which freshness of created names is not guaranteed and ν-PN, remains as an interesting open problem.

[1] P. A. Abdulla, G. Delzanno, and L. Van Begin. *Comparing the Expressive Power of Well-Structured Transition Systems.* 21st Int. Workshop on Computer Science Logic, CSL'07, LNCS vol. 4646, pp. 99-114. Springer, 2007.

[2] P. A. Abdulla, G. Delzanno, and L. Van Begin. *A Language-Based Comparison of Extensions of Petri Nets with and without Whole-Place Operations.* 3rd Int. Conference on Language and Automata Theory and Applications, LATA'09, LNCS vol. 5457, pp. 71-82. Springer, 2009.

[3] P. A. Abdulla, G. Delzanno, and L. Van Begin. *A Classification of the Expressive Power of Well-structured Transition Systems.* Inf. Comput. 209(3): 248-279 (2011)

[4] P. A. Abdulla, and G. Delzanno. *On the coverability problem for constrained multiset rewriting.* In: AVIS 2006, an ETAPS Workshop, 2006.

[5] R. Bonnet, A. Finkel, S. Haddad, F. Rosa-Velardo. *Ordinal Theory for the expressiveness of Well Structured Transition Systems.* In 14th Int. Conference on Foundations of Software Sciences and Computation Structures, FOSSACS'11. LNCS 6604, pp. 153-167. Springer-Verlag, 2011.

[6] G. Cécé, A. Finkel, and S.P. Iyer. *Unreliable channels are easier to verify than perfect channels.* Information and Computation 124(1):20-31, 1996.

[7] G. Decker, M. Weske. *Instance Isolation Analysis for Service-Oriented Architectures.* IEEE Int. Conf. on Services Computing, SCC 2008, pp. 249-256, 2008.

[8] C. Dufourd, A. Finkel and Ph. Schnoebelen. *Reset nets between decidability and undecidability.* 25th Int. Col. on Automata, Languages and Programming, ICALP'98. LNCS 1443, pp. 103-115. Springer, 1998.

[9] A.Finkel, and P.Schnoebelen. *Well-Structured Transition Systems Everywhere!* Theoretical Computer Science 256(1-2):63-92 (2001).

[10] A. Finkel, P. McKenzie, and C. Picaronny. *A well-structured framework for analysing petri net extensions.* Information and Computation, vol. 195(1-2):1-29 (2004).

[11] G. Geeraerts, J. Raskin, and L. Van Begin. *Well-structured languages.* Acta Informatica, 44:249-288. Springer, 2007.

[12] S. Ginsburg. Algebraic and Automata-Theoretic Properties of Formal Languages Elsevier Science Inc. New York, NY, USA, 1975.

[13] A. Gordon. *Notes on Nominal Calculi for Security and Mobility.* Foundations of Security Analysis and Design, FOSAD'00. Lecture Notes in Computer Science vol. 2171, pp. 262-330. Springer, 2001.

[14] P. Habermehl, R. Meyer, and H. Wimmel. *The Downward-Closure of Petri Net Languages.* In ICALP'10, LNCS vol. 6199, pp. 466-477 (2010)

[15] K. van Hee, A. Serebrenik, N. Sidorova and M. Voorhoeve. *Soundness of Resource-Constrained Workflow Nets.* In ICATPN'05, LNCS vol. 3536, pp. 250-267 (2005)

[16] G. Higman. *Ordering by divisibility in abstract algebras.* In Proc. of the London Mathematical Society, (3) 2 (7): 326-336 (1952)

[17] P. Jancar, J. Esparza, and F. Moller. *Petri Nets and Regular Processes.* J. Comput. Syst. Sci. 59(3): 476-503 (1999)

[18] R. Lazic, T.C. Newcomb, J. Ouaknine, A.W. Roscoe, and J. Worrell. *Nets with Tokens Which Carry Data*. Fundamenta Informaticae 88(3):251-274. IOS Press, 2008.

[19] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, 1971.

[20] J.L. Peterson. Petri Net Theory and the Modeling of Systems. Prentice Hall PTR Upper Saddle River, NJ, USA, 1981.

[21] W. Reisig. *"Petri Nets, An Introduction"*. EATCS, Monographs on Theoretical Computer Science, Springer Verlag, Berlin, (1985)

[22] F. Rosa-Velardo and D. de Frutos-Escrig. Name creation vs. replication in Petri Net systems. *Fundamenta Informaticae* 88(3). IOS Press (2008) 329-356.

[23] F. Rosa-Velardo and G. Delzanno. Language-Based Comparison of Petri Nets with Black Tokens, Pure Names and Ordered Data. In 4th International Conference on Language and Automata Theory and Applications, LATA 2010. LNCS vol. 6031, pp. 524-535 (2010)

[24] F. Rosa-Velardo, and D. de Frutos-Escrig. Decidability and Complexity of Petri Nets with Unordered Data. Theor. Comput. Sci. 412(34): 4439-4451 (2011)