Formal Verification of Petri Nets with Names

Marco Montali, Andrey Rivkin

Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
montali,rivkin@inf.unibz.it

Abstract. Petri nets with name creation and management have been recently introduced so as to make Petri nets able to model the dynamics of (distributed) systems equipped with channels, cyphering keys, or computing boundaries. While traditional formal properties such as boundedness, coverability, and reachability, have been thoroughly studied for this class of Petri nets, formal verification against rich temporal properties has not been investigated so far. In this paper, we attack this verification problem. We introduce sophisticated variants of first-order μ-calculus to specify rich properties that simultaneously account for the system dynamics and the names present in its states. We then analyse the (un)decidability boundaries for the verification of such logics, by considering different notions of boundedness. Notably, our decidability results are obtained via a translation to data-centric dynamic systems, a recently devised framework for the formal specification and verification of business processes working over relational database with constraints. In this light, our results contribute to the cross-fertilization between areas that have not been extensively related so far.

1 Introduction

Verifying the correctness of distributed systems, such as interacting web services, requires not only to check whether distributed components interoperate with each other in terms of their control- and message-flow, but also to consider the information they exchange. In the Petri net literature, several variants of colored nets have been thoroughly studied in the last decade so as to enrich classical P/T nets with information that go beyond the control-flow dimension. A notable class of colored Petri nets is constituted by ν-PNs [9,10], which are Petri nets with name creation and management. ν-PNs have been recently introduced so as to make Petri nets able to model the dynamics of (distributed) systems equipped with channels, cyphering keys, or computing boundaries [10]. Interestingly, variants of ν-PNs have been also investigated to express and verify message correlation for interacting web services [5].

At the same time, the field of database theory has produced a flourishing literature on the verification of database-driven dynamic systems and data-aware business processes [4]. In particular, the framework of data-centric dynamic systems (DCDSs for short) has been recently devised as a rich framework for the formal specification and verification of business processes working over full-fledged relational database with constraints [2].

While traditional formal properties such as boundedness, coverability, and reachability, have been extensively studied for ν-PNs, formal verification against rich temporal properties has instead not been investigated so far. The aim of this paper is to attack this verification problem, taking advantage from the interesting decidability results obtained for DCDSs [2]. Specifically we introduce two sophisticated variants of first-order

μ-calculus to specify rich temporal properties that simultaneously account for the dynamics of ν-PNs and relate the names present in their states. We then characterise the (un)decidability boundaries for the verification of such logics over ν-PNs, by considering different notions of boundedness on the used names, on the amount of repetitions of the same name, and combinations of these two dimensions. Notably, the key decidability results obtained in this paper are obtained via a translation from ν-PNs to DCDSs that can provide the basis for the systematic transfer of (un)decidability and complexity results between variants of colored Petri nets and DCDSs.

2 ν-Petri Nets

ν-Petri nets (ν-PNs for short) are an extension of P/T nets [8] with pure name creation and management [9,10]. In a ν-PN, each token carries a name. Fresh names can be dynamically created, and transitions may impose matching restrictions on token names to fire. We briefly introduce ν-PNs, following the formulation in [10].

In the remainder of the paper, we consider the standard notion of *multiset*. Given a set A, the *set of finite multisets* over A, written $A^{\oplus}$, is the set of mappings of the form $m : A \to \mathbb{N}$. Given a multiset $S \in A^{\oplus}$ and an element $a \in A$, $S(a) \in \mathbb{N}$ denotes the number of times a appears in S. Given $a \in A$ and $n \in \mathbb{N}$, we write $a^n \in S$ if $S(a) = n$. The *support* of S is the set of elements that appear in S at least once: $supp(S) = \{a \in A \mid S(a) > 0\}$. We also consider the usual operations on multisets: given $S_1, S_2 \in A^{\oplus}$, we have: *(i)* $S_1 \subseteq S_2$ (resp., $S_1 \subset S_2$) if, for each $a \in A$, $S_1(a) \leq S_2(a)$ (resp., $S_1(a) < S_2(a)$); *(ii)* $S_1 + S_2 = \{a^n \mid a \in A \text{ and } n = S_1(a) + S_2(a)\}$; *(iii)* if $S_1 \subseteq S_2$, $S_2 - S_1 = \{a^n \mid a \in A \text{ and } n = S_2(a) - S_2(a)\}$.

Name management in ν-PNs is formalized by adding to ordinary tokens also a special form of colored tokens, each one carrying a name taken from a countably, unordered infinite set Id of names. In order to define behaviors that are affected not only by the presence of tokens in certain places, but also by the names they carry, all arcs in the net are labelled with matching variables, taken from a countably infinite set Var. Furthermore, to model the ability of an arc to create fresh names upon firing, a special subset $\Upsilon \subset Var$ of variables is introduced, with the constraint that a variable $\nu \in \Upsilon$ can only match with a fresh name, that is, a name not currently present in the net. To preserve usual P/T net notation in $\nu - PN$s, black, uncolored tokens are considered, by simply assuming that a special name $\bullet \in Id$ is used for them. We correspondingly introduce a special variable $\epsilon \in Var$, which only matches with black tokens. We have now all the ingredients to formally introduce ν-PNs and their execution semantics.

Definition 1. A *ν-PN* is a tuple $N = \langle P, T, F \rangle$, where: *(i)* P is a finite set of *places*; *(ii)* T is a finite set of *transitions*, disjoint from P; *(iii)* $F : (P \times T) \cup (T \times P) \to Var^{\oplus}$ is a *flow relation*, s.t., for every $t \in T$, we have $\Upsilon \cap pre(t) = \emptyset$ and $post(t) \setminus \Upsilon \subseteq pre(t)$, where $pre(t) = \bigcup_{p \in P} supp(F(p,t))$ and $post(t) = \bigcup_{p \in P} supp(F(t,p))$. ■

The first condition for the flow relation indicates that new name variables cannot be associated to arcs that go from a place to a transition; in fact, by definition, a variable in Υ cannot match with any name present in the net. The second condition expresses instead that arcs that go from a transition t to a place can be decorated with fresh name variables and/or variables that appear in one of the incoming arcs pointing to t; the

former case denotes the ability of t of generating new names upon firing, whereas the latter case models the matching between names of tokens consumed by t with names of tokens produced by t upon firing.

The usual notion of marking in Petri nets is suitably extended for ν-PNs so as to assign a name to each of the tokens present in the net.

Definition 2. Given a ν-PN $N = \langle P, T, F \rangle$, a *marking* m over N is a function $m : P \to Id^{\oplus}$. A *marked ν-PN* $\overline{N}$ is a tuple $\langle P, T, F, m \rangle$, where $N = \langle P, T, F \rangle$ is a ν-PN, and m is a marking over N. ■

Given a place $p \in P$, $m(p)$ denotes the multiset of names assigned by m to p, and $Id(m)$ denotes the overall set of names present in m: $Id(m) = \bigcup_{p \in P} supp(m(p))$. Furthermore, given a place $p \in P$ and a name $a \in Id$, $m(p)(a)$ denotes the number if times a is assigned by m to p. Let us now discuss how the standard notion of firing in P/T nets is suitably extended to deal with names in ν-PN. Similarly to any class of colored Petri nets, the firing of a transition $t \in T$ is defined w.r.t. a *mode* $\sigma : Var(t) \to Id$, where $Var(t) = pre(t) \cup post(t)$. Intuitively, σ assigns a specific name to each of the variables that annotate the input or output arcs of t. However, to properly fire t, the mode σ must satisfy the different matching conditions expressed by new name variables and by variables that are repeated in the input and output transitions of t.

Definition 3. Consider a ν-PN $N = \langle P, T, F \rangle$, a transition $t \in T$, a marking m over N, and a mode σ for t. We say that t is *enabled in m with mode σ* , written $m[t, \sigma\rangle$, if: *(i)* the mode agrees with the distribution of named tokens in m, i.e., $\sigma(F(p, t)) \subseteq m(p)$ for every $p \in P$; *(ii)* the mode assigns fresh names to the new name variables attached to the output arcs of t, i.e., $\sigma(\nu) \notin Id(M)$ for every $\nu \in \Upsilon \cap Var(t)$.

An enabled transition can fire. In particular, given two markings m and m' over N, a transition $t \in T$, and a mode σ for t, we say that *t fires with mode σ in m producing m'*, written $m[t, \sigma\rangle m'$ if: *(i)* t is enabled in m with mode σ; *(ii)* m' is such that for every $p \in P$, we have $m'(p) = (m(p) - \sigma(F(p, t))) + \sigma(F(t, p))$. ■

Execution Semantics. Starting from an initial marking, we define the execution semantics of a ν-PN N in terms of a possibly infinite-state transition system, whose states are labeled by reachable markings, and where each transition corresponds to the firing of a transition in N. Notice that the definition is different from the definition of reachability graph given in [10], in which a sort of *name abstraction* procedure (called α-equivalence in [10]) is applied while computing the reachable markings. As we will see in Section 3, applying α-equivalence when constructing the execution semantics of a ν-PN leads to a transition system that, in general, does not faithfully reproduce the behaviors of the net when it comes to verification of temporal properties that relate names over time.

Formally, the execution semantics of a marked ν-PN $\overline{N} = \langle P, T, F, m_0 \rangle$ is defined in terms of a transition system $\Gamma_{\overline{N}} = \langle M, m_0, \to \rangle$, where:

- M is a (possibly infinite) set of markings over N.
- $\to \subseteq M \times T \times M$ is a T-labelled transition relation between pairs of markings.
- M and $\to$ are defined by simultaneous induction as the smallest sets satisfying the following conditions: *(i)* $m_0 \in M$; *(ii)* if $m \in M$, then for every transition $t \in T$, mode σ and marking m' over N s.t. $m[t, \sigma\rangle m'$, we have $m' \in M$ and $m \xrightarrow{t} m'$.

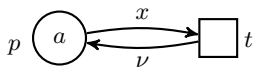

Fig. 1: A ν-PN that is width- and depth-bounded but not run-bounded.

A *run* τ over $\Gamma_{\overline{N}}$ is a possibly infinite sequence of markings $m_0, m_1, \cdots$ where, for every m_i, m_{i+1} in τ, there exists $t \in T$ s.t. $m_i \xrightarrow{t} m_{i+1}$.

Forms of Boundedness. Given a marked ν-PN $\overline{N}$, $\Gamma_{\overline{N}}$ could be infinite-state for different reasons: *(i) width-unboundedness* [10], i.e., accumulation of an unbounded number of different names in a state; *(ii) depth-unboundedness* [10], i.e., accumulation of an unbounded amount of tokens with the same name; *(iii) run-unboundedness*, i.e., presence of unboundedly many names along a run of $\Gamma_{\overline{N}}$; *(iv)* a combination of these conditions. We formally introduce the three corresponding notions of boundedness, and then relate them with the previous literature on ν-PNs.

Definition 4. A marked ν-PN $\overline{N} = \langle P, T, F, m_0 \rangle$ with transition system $\Gamma_{\overline{N}} = \langle M, m_0, \rightarrow \rangle$ is:

- *width-bounded* if there is $n \in \mathbb{N}$ s.t., for each $m \in M$, we have $|Id(m)| \leq n$;
- *depth-bounded* if there is $n \in \mathbb{N}$ s.t., for each $m \in M$, place $p \in P$, and name $a \in Id$, we have $m(p)(a) \leq n$;
- *run-bounded* if there is $n \in \mathbb{N}$ s.t., for every (possibly infinite) run τ over $\Gamma_{\overline{N}}$, we have $|\bigcup_{m \text{ in } \tau} Id(m)| \leq n$. ■

Lemma 1. *If marked ν-PN is run-bounded, then it is width-bounded and depth-bounded.*

Proof. Immediate from Definition 4. □

The converse implication does not hold: as witnessed by the following example, there are ν-PN that are width-bounded and depth-bounded, but not run-bounded. In this light, It is important to observe that in the original formulation of ν-PN reachability graph [10], run-unboundedness does not appear as a source of unboundedness, due to α-equivalence. However, what the authors call boundedness in [10] does not imply that $\Gamma_{\overline{N}}$ is finite-state, because it could still be run-unbounded.

Example 1. Consider the marked ν-PN in Fig. 1. The net is width- and depth-bounded: each marking in its transition system contains exactly one token. However, it is not run-bounded: there is an infinite run in which no name is repeated twice. ■

With the refined execution semantics we consider here, what is called boundedness in [10] corresponds in fact to the following notion of boundedness, which intuitively states that the amount of tokens present in each marking (and thus the corresponding set of names) is bounded.

Definition 5. A marked ν-PN $\overline{N} = \langle P, T, F, m_0 \rangle$ with transition system $\Gamma_{\overline{N}} = \langle M, m_0, \rightarrow \rangle$ is *state-bounded* if there is $n \in \mathbb{N}$ s.t., for each $m \in M$, we have $\sum_{p \in P, a \in Id} m(p)(a) \leq n$; ■

The following results reconstruct those in [10] by considering the notion of state boundedness as defined here.

Lemma 2. *A marked ν-PN is state-bounded if and only if it is width-bounded and depth-bounded.*

Proof. Immediate from Definitions 5 and 4. □

Lemma 3. *A marked ν-PN is state-bounded if and only if its reachability graph, as defined in [10], is bounded.*

Proof. Immediate from Lemma 2 and Proposition 6 in [10]. □

Theorem 6. *Checking whether a marked ν-PN is state-bounded is decidable.*

Proof. Immediate from Lemma 3 and Proposition 3 in [10]. □

3 Verification of ν-PNs

We now consider formal verification of ν-PNs. To specify temporal properties of interest, the logic of choice must provide support for: *(i)* temporal modalities to predicate over the dynamics of the net; *(ii)* first-order (FO) local queries to predicate over the local states of the system, i.e., over markings; *(iii)* FO quantification across states, so as to allow one to relate names and places in different moments of the system evolution.

To support such fundamental requirements, we resort to a Petri net-variant of the $\mu\mathcal{L}_A$ logic defined in [2]. We call this logic $\mu\mathcal{L}_A^N$. Intuitively, $\mu\mathcal{L}_A^N$ allows for complex temporal formulae based on the μ-calculus, virtually the most expressive temporal logic used in verification, and for local queries inspecting the number of names present in the different places.

Definition 7. Given a marked ν-PN $\overline{N} = \langle P, T, F, m_0 \rangle$, a $\mu\mathcal{L}_A^N$ formula Φ over $\overline{N}$ is defined as:

$$\Phi ::= true \mid Z \mid [\#p \leq c] \mid [\#p(x) \leq c] \mid x = y \mid \Phi_1 \wedge \Phi_2 \mid \neg\Psi \mid \exists x.\textsc{live}(x) \wedge \Psi \mid \langle - \rangle \Psi \mid \mu Z.\Psi$$

where $p \in P$, $c \in \mathbb{N}$, x is either a variable or a constant name from $Id(m_0) \cup \{\bullet\}$, and Z is a second order predicate variable of arity 0. We make use of the following abbreviations: $[\#p \circledast c]$ and $[\#p(x) \circledast c]$, where $\circledast \in \{>, \geq, =, <\}$; $\forall x.\textsc{live}(x) \rightarrow \Psi = \neg(\exists x.\textsc{live}(x) \wedge \neg\Psi)$; $\Phi_2 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$; $[-]\Psi = \neg\langle - \rangle\neg\Psi$; $\nu Z.\Psi = \neg\mu.\neg\Psi\,[Z/\neg Z]$. ■

The intuitive meaning of such formulae is: *(i)* $[\#p \leq c]$ is true if the overall amount of tokens in p does not exceed c; *(ii)* $[\#p(x) \leq c]$ is true if the overall amount of tokens in p that match name x does not exceed c; *(iii)* $\textsc{live}(x)$ is true if name x is present in the current marking; *(iv)* $\langle - \rangle\Psi$ is true if there exists a successor marking in which Ψ holds; *(v)* $[-]\Psi$ is true if in all successor markings, Ψ holds; *(vi)* $\mu Z.\Psi$ and $\nu Z.\Psi$ respectively represent the least and greatest fixpoint operator. As usual in the μ-calculus, for fixpoints we require the *syntactic monotonicity* of Ψ w.r.t. Z, that is, every occurrence of the

$$
\begin{aligned}
||\top||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= \Sigma \\
||Z||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= \mathcal{V}(Z) \\
|||[\#p \leq c]||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= \{m \in \Sigma \mid m(p) \leq c\} \\
|||[\#p(x) \leq c]||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= \{m \in \Sigma \mid m(p)(xv) \leq c\} \\
||\textsc{live}(x)||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= \{m \in \Sigma \mid x/d \in v \text{ implies } d \in Id(m)\} \\
||x = y||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= \{m \in \Sigma \mid x/d \in v \text{ if and only if } y/d \in v\} \\
||\Phi_1 \wedge \Phi_2||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= ||\Phi_1||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} \cap ||\Phi_2||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} \\
||\neg F||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= X \setminus ||F||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} \\
||\exists x.\Phi||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= \{m \in \Sigma \mid \exists a \in Id \text{ s.t. } m \in ||\Phi||^{\Gamma_{\overline{N}}}_{\mathcal{V},v[x/a]}\} \\
||\langle\!\rightarrow\!\rangle\Phi||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= \{m \in \Sigma \mid \exists m' \in \Sigma \text{ s.t. } m \rightarrow m' \text{ and } m' \in ||\Phi||^{\Gamma_{\overline{N}}}_{\mathcal{V},v}\} \\
||\mu Z.\Phi||^{\Gamma_{\overline{N}}}_{\mathcal{V},v} &= \bigcap\{Y \subseteq \Sigma \mid ||\Phi||^{\Gamma_{\overline{N}}}_{\mathcal{V}[Z/Y],v} \subseteq Y\}
\end{aligned}
$$

Fig. 2: Semantics of $\mu\mathcal{L}^N_A$

variable Z in Ψ must be within the scope of an even number of negation signs. This guarantees that the least and greatest fixpoints always exist.

Formally, a $\mu\mathcal{L}^N_A$ formula Φ over $\overline{N} = \langle P, T, F, m_0\rangle$ is interpreted over the transition system $\Gamma_{\overline{N}} = \langle \Sigma, m_0, \rightarrow\rangle$. Since Φ can contain both individual and predicate free variables, we introduce an individual variable valuation v mapping individual variables x to names in Id, and a predicate variable valuation V mapping predicate variables Z to subsets of Σ. The semantics of $\mu\mathcal{L}^N_A$ formulae is then defined through an *extension function* $||\cdot||^{\Gamma_{\overline{N}}}_{v,V}$ that maps formulae to subsets of Σ. The extension function is inductively defined as shown in Fig. 2.

When Φ is a closed formula, its valuation does not depend neither on V nor v. Hence, we denote the extension of Φ by simply using $||\Phi||^{\Gamma_{\overline{N}}}$. Given a closed $\mu\mathcal{L}^N_A$ property Φ and a marked ν-PN $\overline{N} = \langle P, T, F, m_0\rangle$, we say that $\overline{N}$ *verifies* Φ, written $\overline{N} \models \Phi$, if $m_0 \in ||\Phi||^{\Gamma_{\overline{N}}}$.

Example 2. Formula $\Phi_d = \nu Z.(\forall x.[\#p(x) = 1] \rightarrow \nu Y.\neg\textsc{live}(x) \wedge \langle\!\rightarrow\!\rangle Y) \wedge [\!\rightarrow\!]Z$ holds in the ν-PN of Fig. 1. Φ_d expresses that it is always the case that, whenever place p contains exactly one token named x, then there exists an ongoing run in which x never reappears. Notice that, although LIVE is not explicitly used in Φ_d, it is a $\mu\mathcal{L}^N_A$ formula, because $[\#p(x) = 1]$ implies $\textsc{live}(x)$. ■

We observe that there cannot be any finite-state representation of the transition system for the ν-PN in Fig. 1 in which the $\mu\mathcal{L}^N_A$ property in Ex. 2 holds. In fact, the net always has a successor, and if the transition system is finite-state, infinite runs should sooner or later go back visiting a marking that was visited before, violating the fact that there must exist a run in which names never reappear. Since the ν-PN of Fig. 1 is bounded in the sense of [10], its reachability graph is finite-state, hence it does not properly represent the execution semantics of the ν-PN when it comes to verification of $\mu\mathcal{L}^N_A$ properties. The intuitive reason is that $\mu\mathcal{L}^N_A$ properties can "store" visited names inside a FO quantifier, and check the presence or absence of such names in markings that are arbitrarily far away from the state in which the quantifier was bound. Consequently, the name renaming allowed by α-equivalence in [10] does not preserve $\mu\mathcal{L}^N_A$ properties.

4 Undecidability Results

We prove here two key undecidability results related to the verification of $\mu\mathcal{L}_A^N$ properties over ν-PNs. This motivates the fine-grained study presented in Sec. 5.

Theorem 8. *Verification of $\mu\mathcal{L}_A^N$ properties over ν-PNs is undecidable even when formulae do not make use of FO quantification, and only employ a single, constant name.*

Proof. Consider marked ν-PNs that only use the special name $\bullet$ and the special variable ϵ. This class of ν-PNs coincides with classical P/T nets. Now consider $\mu\mathcal{L}_A^N$ properties that do not make use of FO quantification, and whose local predicates only employ the special name $\bullet$. This class of formulae resembles the propositional μ-calculus whose verification over P/T nets is shown to be undecidable in [7]. □

One could wonder what happens when the considered ν-PN obeys to some boundedness criterion among those discussed in Sec. 2. The following negative result, however, holds for the important class of state-bounded ν-PNs.

Theorem 9. *Verification of $\mu\mathcal{L}_A^N$ properties over state-bounded ν-PNs is undecidable.*

Proof. The proof resembles the proof of Theorem 5.1 in [1], and is based on a reduction from satisfiability of LTL with freeze quantifier over infinite data words, shown to be undecidable in [6], to verification of (linear-time) $\mu\mathcal{L}_A^N$ properties over state-bounded ν-PNs. An infinite data word is an infinite sequence of pairs (σ_i, d_i), where σ_i is a label taken from a finite set $\mathcal{L}$, and d_i is a datum taken from a countably infinite set $\mathcal{D}$ (which, without loss of generality, we consider here to coincide with Id).[1]

The idea is to construct a "universal" ν-PN whose transition system runs correspond to all possible data words, and then reduce satisfiability to verification over such transition system. Given a finite set $\mathcal{L} = \{\sigma_1, \ldots, \sigma_n\}$, we construct the marked ν-PN $\overline{U}^{\mathcal{L}} = \langle P, T, F, m_0 \rangle$ as follows: *(i)* $P = \{p_0, p_{\sigma_1}, \ldots, p_{\sigma_n}\}$, i.e., P contains a starting place p_0 and one dedicated place for each $\sigma_i \in \mathcal{L}$; *(ii)* $T = \{t_{0i} \mid i \in \{1, \ldots, n\}\} \cup \{t_{ij}^{=}, t_{ij}^{\neq} \mid i, j \in \{1, \ldots, n\}\}$; *(iii)* m_0 assigns a single, black token $\bullet$ to p_0; *(iv)* F is such that:

- for each transition t_{0i}, $F(p_0, t_{0i}) = \emptyset$ and $F(t_{0i}, p_{\sigma_i}) = \{\nu\}$; this models an initial pure nondeterministic choice from p_0, in which a fresh name is put in one of p_{σ_i}.
- for each transition $t_{ij}^{=}$, $F(p_{\sigma_i}, t_{ij}^{=}) = F(t_{ij}^{=}, (p_{\sigma_j}) = \{x\}$; this models the situation where a token currently present in p_{σ_i} is moved into p_{σ_j}, maintaining its name.
- for each transition $t_{ij}^{\neq}$, $F(p_{\sigma_i}, t_{ij}^{\neq}) = \emptyset$ and $F(t_{ij}^{\neq}, (p_{\sigma_j}) = \{\nu\}$; this models the situation where a token present in p_{σ_i} is moved into p_{σ_j}, by changing its name.

Fig. 4 in the appendix shows the marked ν-PN obtained by following this procedure when $\mathcal{L} = \{\sigma_1, \sigma_2\}$. We observe that $\overline{U}^{\mathcal{L}}$ is state-bounded: every marking in $\Gamma_{\overline{U}^{\mathcal{L}}}$ assigns a single token to one of the places, leaving all other places empty.

Now consider an LTL formula φ with freeze quantifier. We first apply the following transformation rules to obtain a $\mu\mathcal{L}_A^N$ formula φ_N:

- Temporal modalities in φ are expressed in their corresponding μ-calculus form (recall that μ-calculus subsumes both CTL and LTL).
- Boolean connectives are maintained unaltered.

[1] For a detailed description of LTL with freeze quantifier, the interested reader can refer to [6].

- Each freeze-quantifier $\downarrow_n$ in φ corresponds to $\exists x_n.\textsc{live}(x_n)$ in φ_N.
- Each occurrence of $\uparrow_n$ in φ corresponds to $\textsc{live}(x_n)$ in φ_N.
- Each proposition $\sigma_i \in \mathcal{L}$ appearing in φ becomes $[\#p_{\sigma_i} = 1]$ in φ_N.

For example, property $\downarrow_1 \mathbf{X}(\mathbf{F}(\sigma_1 \rightarrow \uparrow_1))$ becomes

$$\exists x_1.\textsc{live}(x_1) \wedge \mu Z.([\#p_{\sigma_1} = 1] \rightarrow \textsc{live}(x_1)) \vee [\dashv] Z$$

We now take φ_N and we set $\Phi_N = \langle\dashv\rangle\neg\varphi_N$. Intuitively, $\langle\dashv\rangle$ is applied to move one step away from the initial marking, which does not correspond to a proper data word element. It is now easy to see that an LTL formula φ with freeze quantifier is *unsatisfiable* over infinite data words and labels $\mathcal{L}$ if and only if $\Gamma_{\overline{U}^{\mathcal{L}}} \models \Phi_N$. □

5 Decidability of Verification

In this section, we provide the two key decidability results of this paper. We start overviewing the salient features of DCDSs, and proposing a translation mechanism from ν-PNs to DCDSs, that is instrumental towards such decidability results.

5.1 Data-Centric Dynamic Systems

We recall the main aspects of Data-Centric Dynamic Systems (DCDSs) [2]. A DCDS is a pair $\mathcal{S} = \langle \mathcal{D}, \mathcal{P} \rangle$ where $\mathcal{D}$ is a *data layer* and $\mathcal{P}$ is a *process layer*. Both layers are interacting as follows: the data layer stores all the data of interest, while the process layer modifies and evolves such data.

Definition 10. A *data layer* is a tuple $\mathcal{D} = \langle \mathcal{C}, \mathcal{R}, \mathcal{E}, \mathcal{I}_0 \rangle$ where:

- $\mathcal{C}$ is a countably infinite set of constants/values;
- $\mathcal{R} = \{R_1, ..., R_n\}$ is a database schema, constituted by a finite set of relation schemas;
- $\mathcal{E}$ is a finite set $\{\mathcal{E}_1, ..., \mathcal{E}_m\}$ of equality constraints. Each $\mathcal{E}_i$ has the form $Q_i \rightarrow \bigwedge_{j=1,...,k} z_{ij} = y_{ij}$[2], where Q_i is a domain independent FOL query over $\mathcal{R}$ using constants from $\mathtt{ADOM}(\mathcal{I}_0)$[3] whose free variables are $\vec{x}$, and z_{ij} and y_{ij} are either variables in $\vec{x}$ or constants in $\mathtt{ADOM}(\mathcal{I}_0)$[4];
- $\mathcal{I}_0$ is a database instance that represents the initial state of the data layer, which conforms to the schema $\mathcal{R}$ and satisfies the constraints $\mathcal{E}$: namely, for each constraint $Q_i \rightarrow \bigwedge_{j=1,...,k} z_{ij} = y_{ij}$ and for each tuple (i.e., substitution for the free variables) $\theta \in ans(Q_i, \mathcal{I}_0)$, it holds that $z_{ij}\theta = y_{ij}\theta$. ■

The process layer constitutes the progression mechanism for the DCDS. It is assumed that at every time the current instance of the data layer can be both arbitrarily queried and updated (through action executions), possibly involving external service calls to get new values from the environment.

[2] Instead of $z_{ij} = y_{ij}$ we can also use $z_{ij} \neq y_{ij}$ or $\bot$ (forming a full denial constraint).

[3] Given a database instance $\mathcal{I}$, its active domain $\mathtt{ADOM}(\mathcal{I})$ is the subset of $\mathcal{C}$ such that $u \in \mathtt{ADOM}(\mathcal{I})$ if and only if u occurs in $(\mathcal{I})$.

[4] For convenience, and without loss of generality, we assume that all constants used inside formulas appear in $\mathcal{I}_0$

Definition 11. A *process layer* $\mathcal{P}$ over a data layer $\mathcal{D} = \langle \mathcal{C}, \mathcal{R}, \mathcal{E}, \mathcal{I}_0 \rangle$ is a tuple $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \rho \rangle$. $\mathcal{F}$ is a finite set of *functions*, each representing the interface to an external service. Such services can be called, and as a result the function is activated and the answer is produced. How the result is actually computed is unknown to the DCDS since the services are indeed external.

$\mathcal{A}$ is a finite set of *actions*, whose execution updates the data layer, and may involve external service calls. Formally, an *action* $\alpha \in \mathcal{A}$ is an expression $\alpha(p_1, ..., p_n) : \{e_1, ..., e_m\}$ where:

- $\alpha(p_1, ..., p_n)$ is the action *signature*, constituted by a name α and a sequence $p_1, ..., p_n$ of *parameters*, to be substituted with values when the action is invoked;
- $\{e_1, ..., e_m\}$, also denoted as $\texttt{EFFECT}(\alpha)$, is a set of *specifications of effects*, which are assumed to take place simultaneously. Each e_i has the form $q_i^+ \wedge Q_i^- \rightsquigarrow E_i$. The formula $q_i^+ \wedge Q_i^-$ is a query over $\mathcal{R}$ whose terms are variables, action parameters, and constants from $\texttt{ADOM}(\mathcal{I}_0)$, where q_i^+ is a union of conjunctive queries, and Q_i^- is an arbitrary FOL formula whose free variables are among those of q_i^+. Intuitively, q_i^+ selects the tuples to instantiate the effect, and Q_i^- filters away some of them. E_i is the *effect*, i.e., a set of facts for $\mathcal{R}$, which includes as terms the following: terms in $\texttt{ADOM}(\mathcal{I}_0)$, free variables of q_i^+ and Q_i^- (including action parameters), and Skolem terms formed by applying a function $f \in \mathcal{F}$ to one of the previous kinds of terms. Such Skolem terms involving functions represent external (nondeterministic) service calls and are interpreted as the returned value chosen by an external user/environment when executing the action.

Finally, ρ is a finite set of *condition-action rules* (of the form $Q \rightarrow \alpha$, where $\alpha \in \mathcal{A}$ and Q is a FOL query[5] over $\mathcal{R}$) that form the specification of the overall *process*, which tells at any moment which actions can be executed. ■

Execution semantics. The execution semantics of a DCDS $\mathcal{S}$ is defined in terms of a possibly infinite transition system $\Lambda_{\mathcal{S}}$ whose states are labeled by databases. Such a transition system represents all possible computations that the process layer can do on the data layer. Formally, $\Lambda_{\mathcal{S}} = \langle \Delta, \mathcal{R}, \Sigma, s_0, db, \Rightarrow \rangle$, where: *(i)* Δ is a countably infinite set of values; *(ii)* Σ is a set of states; *(iii)* $s_0 \in \Sigma$ is the initial state; *(iv)* db is a function such that for each state $s \in \Sigma$ returns a database $D \subseteq \Delta$ conforming to $\mathcal{R}$; *(v)* $\Rightarrow \subseteq \Sigma \times \Sigma$ is a transition relation over states.

Given a DCDS $\mathcal{S} = \langle \mathcal{D}, \mathcal{P} \rangle$ with $\mathcal{D} = \langle \mathcal{C}, \mathcal{R}, \mathcal{E}, \mathcal{I}_0 \rangle$ and $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \rho \rangle$ one can construct $\Lambda_{\mathcal{S}}$ following the next steps. Starting from $\mathcal{I}_0$, condition-action rules in ρ are evaluated and the set of all possible executable actions with corresponding ground parameter assignments is defined. Then, nondeterministically, one such action with parameter assignments $\alpha\vec{p}$ (α is partially grounded with the parameter assignment $\vec{p}$) is selected and executed over $\mathcal{I}_0$ by calculating all the answers of action's left-hand side and grounding the right-hand side accordingly. In case that the right-hand side is containing service calls, the latter are issued nondeterministically returning values from $\mathcal{C}$. Then, every service call is substituted with its actual result (i.e., the values yielded on the previous step). The overall set of ground facts obtained is constituting the next database instance. Hence, the transition system construction is determined as process

[5] Its free variables are exactly the parameters of α, and other terms can be quantified variables or constants in $\texttt{ADOM}(\mathcal{I}_0)$.

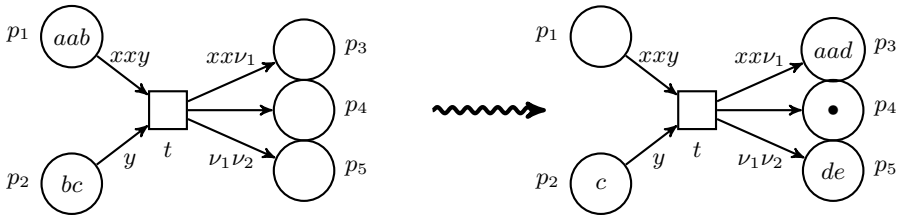

Fig. 3: A marked ν-PN, where t is fired with mode σ: $\sigma(x) = a$, $\sigma(y) = b$, $\sigma(\nu_1) = d$, $\sigma(\nu_2) = e$, where d and e are fresh names

of all possible successors extraction, each of which is obtained by selecting one of the executable actions with corresponding parameters, and one result perch each service call involved. The construction proceeds recursively over the newly generated states. For a formal description of the execution semantics, one can check [2].

5.2 From ν-PNs to DCDSs

In this section we discuss how a ν-PN $\overline{N}$ can be encoded into a DCDS $\tau(\overline{N}) = \langle \mathcal{D}_N, \mathcal{P}_N \rangle$ that faithfully reproduces the execution semantics of $\overline{N}$. We discuss the translation using the marked ν-PN $\overline{N}$ shown in Fig. 3. The example is simple but illustrates all the main difficulties of the translation. The general translation is provided in the appendix. Specifically, there are two fundamental critical issue in the translation. First, ν-PNs have a bag semantics, whereas DCDSs rely on set semantics. Hence, the only way for faithfully reconstructing the execution semantics of $\overline{N}$ in terms of a DCDS, is to introduce implicit token identifiers that are in the DCDS to distinguish two distinct tokens that belong to the same place and carry the same name. The same strategy must be consistently maintained when, upon firing, two distinct tokens with the same name must be inserted into the same place.

Second, to inject new data (corresponding, in our case, to token names and token identifiers) in the system, DCDSs employ the notion of service call. However, an issued service call may not necessarily return a fresh value, but could (nondeterministically) return a value that is currently used in the system. To properly reconstruct the fresh name generation of ν-PNs, temporary relations and specific database constraints must be employed.

In the remainder of the section, we use the following typographical conventions: v for variables, v for constants.

Data Layer. The data layer is $\mathcal{D}_N = \langle \mathcal{C}_N, \mathcal{R}_N, \mathcal{E}_N, \mathcal{I}_0^N \rangle$. The data domain $\mathcal{C}_N$ contains the overall set Id of names, together with additional constants used to denote the different elements of $\overline{N}$: $Id \cup T \cup P \cup Var \subseteq \mathcal{C}_N$.

The relation schema is $\mathcal{R}_N = \{p_i/2 \mid i \in \{1, \ldots, 5\}\} \cup NewID/5 \cup FL/0$. Relation $p_i(\mathsf{id}, \mathsf{n})$ represents that p_i currently contains a token with identifier id and name n;

$NewID(\mathsf{t}, \mathsf{p}, \mathsf{v}, \mathsf{d}, \mathsf{id})$ is a temporary relation used to store that transition t is has produced a token with identifier id and name d, that this token is matched with variable v, and that the destination of this token in the next step is place p. Relation FL is a flag distinguishing between the two modes in which the DCDS $\tau(\overline{N})$ operates to simulate

the firing of a transition t: *(i)* the first consisting in the consumption of tokens from the input places of t, and the contemporary generation of token identifiers and names, to be stored in the temporary relations $NewID$ of t; *(ii)* the second consisting in the forwarding of tokens from such temporary relations to the output places of t, provided that the previous step has completed correctly, i.e., without violating any constraint in $\mathcal{E}_N$.

The relation schema $\mathcal{C}_N$ is subject to constraints $\mathcal{E}_N = \{\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4\}$. Constraint $\mathcal{E}_1$ and $\mathcal{E}_2$ ensures that identifiers present in the temporary relation $NewID$ are unique, i.e., distinct from all identifiers stored in the place relations (cf. $\mathcal{E}_1$), and from those stored in other tuples of $NewID$ (cf. $\mathcal{E}_2$):

$$\mathcal{E}_1 = \forall id.NewID(\_,\_,\_,id,\_) \wedge \bigvee_{i\in\{1,\ldots,5\}} p_i(id,\_) \rightarrow \bot$$
$$\mathcal{E}_2 = \forall id,t,p,v,n,t',p',v',n'.\ NewID(t,p,v,id,n) \wedge NewID(t',p',v',id,n') \rightarrow t = t' \wedge p = p' \wedge v = v' \wedge n = n'$$

$\mathcal{E}_3$ and $\mathcal{E}_4$ mirror $\mathcal{E}_1$ and $\mathcal{E}_2$ by considering names stored in $NewID$ tuples that refer to new name variables in Υ, i.e., in our case ν_1 and ν_2 (which are constants in the DCDS):

$$\mathcal{E}_3 = \forall n. \bigvee_{\mathsf{c}\in\{\nu_1,\nu_2\}} NewID(\_,\_,\mathsf{c},\_,n) \wedge \bigvee_{i\in\{1,\ldots,5\}} p_i(\_,n) \rightarrow \bot$$
$$\mathcal{E}_4 = \forall n_1,n_2.NewID(\_,\_,\nu_1,\_,n_1) \wedge NewID(\_,\_,\nu_2,\_,n_2) \rightarrow n_1 \neq n_2$$

Finally, $\mathcal{I}_0^N$ encodes m_0 by maintaining the name distribution of tokens, as well as their cardinality. The latter requirement is enforced by picking pairwise distinct identifiers from $\mathcal{C}_N$. It is worth noting that identifiers and names are always treated separately by the DCDS, and therefore also identifiers can be picked from the set Id of names. Consequently, a suitable choice for $\mathcal{I}_0^N$ is $\{p_1(\mathsf{a},\mathsf{a}), p_1(\mathsf{b},\mathsf{a}), p_1(\mathsf{c},\mathsf{a}), p_2(\mathsf{d},\mathsf{b}), p_2(\mathsf{e},\mathsf{b})\}$.

Process layer. The process layer of $\tau(\overline{N})$ is defined as $\mathcal{P}_N = \langle \mathcal{F}_N, \mathcal{A}_N, \rho_N \rangle$. $\mathcal{F}_N$ contains service calls that are used to inject token identifiers and names into the system. In particular, each time a transition t fires, tokens are generated according to the arcs that have t as input. The generation of a new identifier depends on: *(i)* the considered arc, which is in turn determined by the input transition and output place; *(ii)* the variable name; *(iii)* the occurrence of the variable name. For this reason, we introduce a service call $genID/4$, where $genID(\mathsf{t},\mathsf{p},\mathsf{v},\mathsf{i})$ represents the new id generation for the i-th occurrence of variable v on the arc that connects t to p. Name generation depends instead only on the considered new name variable in Υ. Therefore, we introduce a service call $genName/1$, where $genFresh(\nu)$ represents the generation of a name for ν.

The process and action components ρ_N and $\mathcal{A}_N$ contain an action gen_t for each transition t of $\overline{N}$, and an additional action $transf$. Intuitively, the firing of t is simulated in the corresponding DCDS by the serial execution of gen_t and $transf$. The first action is executable only when t is enabled, and its execution leads to consume the named tokens in the input places of t (in accordance with the variables attached to the arcs), and to generate (and store in the temporary relation $NewID$) the named tokens produced by t (again in accordance with the variables attached to the arcs). Action $transf$ then takes care of transferring such newly generated tokens into the corresponding output places.

Formally, in our example we have a single transition, hence $\mathcal{A}_N = \{gen_t\}$. The process ρ_N expresses the executability of gen_t as follows:

$$p_1(id_1,x) \wedge p_1(id_2,x) \wedge p_1(id_3,y) \wedge p_2(id_4,y) \wedge \bigwedge_{i,j\in\{1,4\},i\neq j} id_i \neq id_j \wedge \neg FL \mapsto gen_t(id_1id_2,id_3,id_4,x,y)$$

Observe also the consistent usage of variables x and y w.r.t. the ν-PN in Fig. 3, and the fact that gen_t takes in input the pairs identifiers of selected tokens, as well as the matched names. This allows us to make gen_t able to consume the selected tokens (without touching the unselected ones), and at the same time able to transfer the matched names to the output. Specifically, gen_t works as follows (notice the difference between constant x and parameter x): $gen_t(id_1 id_2, id_3, id_4, x, y) =$

$$
\begin{aligned}
\{\, & p_1(id,n) \wedge \bigwedge_{i\in\{1,2,3\}} id \neq id_i \rightsquigarrow \{p_1(id,n)\} \\
& p_2(id,n) \wedge id \neq id_4 \rightsquigarrow \{p_2(id,n)\} \\
& p_i(id,n) \rightsquigarrow \{p_i(id,n)\} \qquad \text{for } i \in \{3,4,5\} \\
& true \rightsquigarrow \{NewID(\mathsf{t},\mathsf{p_3},\mathsf{x},genID(\mathsf{t},\mathsf{p_3},\mathsf{x},1),x), \\
& \qquad NewID(\mathsf{t},\mathsf{p_3},\mathsf{x},genID(\mathsf{t},\mathsf{p_3},\mathsf{x},2),x)\} \\
& \qquad NewID(\mathsf{t},\mathsf{p_3},\nu_1,genID(\mathsf{t},\mathsf{p_3},\nu_1,1),genName(\nu_1))\} \\
& true \rightsquigarrow \{NewID(\mathsf{t},\mathsf{p_4},\epsilon,genID(\mathsf{t},\mathsf{p_3},\epsilon,1),\bullet)\} \\
& true \rightsquigarrow \{NewID(\mathsf{t},\mathsf{p_5},\nu_1,genID(\mathsf{t},\mathsf{p_5},\nu_1,1),genName(\nu_1)), \\
& \qquad NewID(\mathsf{t},\mathsf{p_5},\nu_1,genID(\mathsf{t},\mathsf{p_5},\nu_2,1),genName(\nu_2))\} \\
& true \rightsquigarrow FL \,\}
\end{aligned}
$$

The first block of effects is dedicated to maintain those tokens that are not consumed by the transition t. For places that are input of t, this requires to properly filter out those tokens that were selected in the precondition of gen_t (which has non-empty answers only if t is enabled). The second block of effects is instead focused on the generation of one distinct token for each of the variables that decorate the output arcs of t, with the corresponding restrictions over matching names. The last effect switches the flag from false to true, inhibiting the possibility of reapplying actions of the type gen_t; this is imposed in the precondition of gen_t, which contains $\neg FL$ among its conjuncts.

Observe that, once gen_t is applied, the newly obtained database must conform to all constraints in $\mathcal{E}$. Consequently, only states that correctly assign new identifiers and fresh names in the case of Υ variables are kept as valid successors. This explains why this two-step approach is needed when formalizing the firing of t.

Let us now turn to the action $transf$. It is independent of the specific transition, because what it only needs to do is transfer the tokens generated in a generation step from the temporary relations $NewID$ to the proper output places. The only precondition of $transf$ is then just to check whether FL is true (which attests that a generation action has been applied in the previous computation step). I.e., ρ_N contains the following condition-action rule: $FL \mapsto transf()$. In the case illustrated by Fig. 3, we then have:

$$
\begin{aligned}
transf() = \{ \quad & p_i(id,n) \rightsquigarrow \{p_i(id,n)\} \qquad \text{for } i \in \{1,\ldots,5\} \\
& NewId(\_,\mathsf{p_j},\_,id,n) \rightsquigarrow \{p_j(id,n)\} \qquad \text{for } j \in \{1,\ldots,5\} \,\}
\end{aligned}
$$

Notice that since FL is not explicitly copied by $transf$, it is implicitly toggled, making $transf$ not applicable anymore (it will be again applicable after a generation step).

From $\mu\mathcal{L}_A^N$ to $\mu\mathcal{L}_A$. We now continue the correspondence between verification problems in ν-PNs and verification problems in DCDSs, by considering the temporal logics used to specify properties. As already mentioned in Sec. 3, $\mu\mathcal{L}_A^N$ has been inspired by the $\mu\mathcal{L}_A$ logic for DCDSs. We now establish a precise translation ξ that takes a $\mu\mathcal{L}_A^N$ property Φ

and produces a corresponding $\mu\mathcal{L}_A$ property $\xi(\Phi)$:

$$\xi(\Phi) = \begin{cases} \Phi & \text{if } \Phi \in \{true, x = y, Z\} \\ \forall i_1, ..., i_c, i_{c+1}. \bigwedge_{j \in \{1,..,c+1\}} p(i_j, \_) \rightarrow \bigvee_{k,l \in \{1,..,c+1\}, k<l} i_k = i_l & \text{if } \Phi = [\#p \leq c] \\ \forall i_1, ..., i_c, i_{c+1}. \bigwedge_{j \in \{1,..,c+1\}} p(i_j, x) \rightarrow \bigvee_{k,l \in \{1,..,c+1\}, k<l} i_k = i_l & \text{if } \Phi = [\#p(x) \leq c] \\ \neg\xi(Psi) & \text{if } \Phi = \neg\Psi \\ \xi(\Phi_1) \vee \xi(\Phi_2) & \text{if } \Phi = \Phi_1 \vee \Phi_2 \\ \exists x.\xi(\Psi) & \text{if } \Phi = \exists x.\Psi \\ \langle\rightarrow\rangle\langle\rightarrow\rangle\xi(\Psi) & \text{if } \Phi = \langle\rightarrow\rangle\Psi \\ \mu Z.\xi(\Psi) & \text{if } \Phi = \mu Z.\Psi \end{cases}$$

The only non-trivial cases are local queries, which are expressed as counting queries over the current database, and the case of $\langle\rightarrow\rangle$, which is translated by doubling the $\langle\rightarrow\rangle$ operator since every step in the transition system of a marked ν-PN corresponds to a sequence of two steps (generation and transfer) in the transition system of the corresponding DCDS.

5.3 Decidability Results

We are now in the position of formally assessing the connection between ν-PNs and DCDSs, leveraging on the translation functions τ and ξ.

Theorem 12. *For every marked ν-PN $\overline{N}$ and every closed $\mu\mathcal{L}_A^N$ formula Φ, $\overline{N} \models \Phi$ if and only if $\tau(\overline{N}) \models \xi(\Phi)$.*

Proof. Let $\overline{N} = \langle P, T, F, m_0 \rangle$, and $\tau(\overline{N}) = \langle \mathcal{D}_N, \mathcal{P}_N \rangle$. Recall that for each transition $t \in T$ there are an action gen_t and a condition-action rule $Q(\vec{x}) \mapsto gen_t$ in $\mathcal{P}_N$. In addition, also $transf$ is an action in $\mathcal{P}_N$.

The proof is a variation of the simpler proof given in [3] for the comparison between P/T nets and (lossy) DCDSs. To compare the states of $\Gamma_{\overline{N}}$ with those of $\Lambda_{\tau(\overline{N})}$, we define the following *name cardinality-equivalence* relation: given a marking m in $\Gamma_{\overline{N}}$ and a state s in $\Lambda_{\tau(\overline{N})}$, we say that m is name cardinality-equivalent to s, written $m \approx s$, if, for each place $p_i \in P$ and name $a \in Id$, $m(p)(a) = n$ if and only if $db(s)$ contains n tuples $\langle \_, a \rangle$ in relation p. By definition, $m_0 \approx s_0$. It can then be shown, inductively, that, given m in $\Gamma_{\overline{N}}$ and s in $\Lambda_{\tau(\overline{N})}$ such that $m \approx s$, for every transition $t \in T$ (and corresponding action gen_t in $\tau(\overline{N})$): *(i)* for every t, σ and m' s.t. $m[t, \sigma\rangle m'$, there exists a sequence $s \rightarrow s' \rightarrow s''$ in $\Lambda_{\tau(\overline{N})}$, where s' is produced by the application of gen_t with parameter substitution σ', and where s'' is produced from s' by the application of $transf$, s.t. $m' \approx s''$; *(ii)* for every sequence $s \rightarrow s' \rightarrow s''$ produced by the application of action gen_t with parameter substitution σ, followed by the application of $transf$, there exists σ' and m' s.t. $m[t, \sigma'\rangle m'$ and $m' \approx s''$. The proof concludes by making the following two observations. First, the two transition systems have the same structure, apart from the fact that each transition in $\Gamma_{\overline{N}}$ corresponds to a sequence of two transitions in $\Lambda_{\tau(\overline{N})}$, a feature that is correctly mirrored by ξ. Second, name cardinality-equivalence preserves the answers of local queries, i.e., given m and s s.t. $m \approx s$:

- for every number $n \in \mathbb{N}$ and every boolean $\mu\mathcal{L}_A^N$ query of the form $[\#p \leq c]$, $[\#p \leq c]$ is true in m if and only if $\xi([\#p \leq c])$ is true in $db(s)$;

– for every number $n \in \mathbb{N}$, every open $\mu\mathcal{L}_A^N$ query of the form $[\#p(x) \leq c]$, and every substitution $\theta = [x/\mathsf{n}]$, $[\#p(x) \leq c]\theta$ is true in m if and only if $\xi([\#p(x) \leq c]\theta)$ is true in $db(s)$. □

With Theorem 12 at hand, we can now prove the following key decidability result:

Theorem 13. *Verification of $\mu\mathcal{L}_A^N$ properties over run-bounded marked ν-PNs is decidable, and reducible to finite-state model checking.*

Proof. From the proof of Theorem 12, we know that given a marked ν-PN $\overline{N}$, $\tau(\overline{N})$ faithfully reconstruct the execution semantics of $\overline{N}$. In particular, the states of $\Gamma_{\overline{N}}$ and those of $\Lambda_{\tau(\overline{N})}$ are connected by the name cardinality-equivalence relation. This immediately leads to the fact that if $\overline{N}$ is run-bounded, then $\tau(\overline{N})$ is a run-bounded DCDS (in the sense defined in [2]). The claim is then obtained by combining Theorem 12 with the fact that verification of $\mu\mathcal{L}_A$ properties over run-bounded DCDSs is decidable, and reducible to finite-state model checking [2]. □

The notion of run-boundedness is quite restrictive, because it does not allow for infinite runs triggering unboundedly many times an arc decorated with a new name variable from Υ. In [2] also a decidability result for the verification of state-bounded DCDSs is given, by limiting the power of $\mu\mathcal{L}_A$ when using FO quantification that spans across system states. In particular, decidability is proven for the logic $\mu\mathcal{L}_P$, in which FO quantification only applies to those objects that *persist* in the current active domain. When quantification is applied to an object that disappears from the active domain, then it is possible to control whether the property, applied to that object, trivializes to $true$ or $false$. By incorporating this idea in the setting considered here, we obtain the logic $\mu\mathcal{L}_P^N$.

Definition 14. Given a marked ν-PN $\overline{N} = \langle P, T, F, m_0 \rangle$, a $\mu\mathcal{L}_P^N$ formula Φ over $\overline{N}$ is defined as: $\Phi ::= true \mid Z \mid [\#p \leq c] \mid [\#p(x) \leq c] \mid x = y \mid \Phi_1 \wedge \Phi_2 \mid \neg\Psi \mid$
$\exists x.\textsc{live}(x) \wedge \Psi \mid \textsc{live}(\vec{x}) \wedge \langle\!-\!\rangle\Psi \mid \textsc{live}(\vec{x}) \wedge [\!-\!]\Psi \mid \mu Z.\Psi$
with the usual assumptions done for $\mu\mathcal{L}_A^N$, and the additional assumption that in $\textsc{live}(\vec{x}) \wedge \langle\!-\!\rangle\Psi$ and $\textsc{live}(\vec{x}) \wedge [\!-\!]\Psi$, variables $\vec{x}$ are exactly the free variables of Φ, once we replace each bound predicate variable Z in Φ with its bounding formula $\mu Z.\Phi'$. Beside the usual abbreviations, we also make use of $\textsc{live}(\vec{x}) \rightarrow \langle\!-\!\rangle\Psi = \neg(\textsc{live}(\vec{x}) \wedge \langle\!-\!\rangle\neg\Psi)$ and $\textsc{live}(\vec{x}) \rightarrow [\!-\!]\Psi = \neg(\textsc{live}(\vec{x}) \wedge [\!-\!]\neg\Psi)$. ■

As shown by the following example, $\mu\mathcal{L}_P^N$ properties are particularly useful in all those cases where names maintain an identity only if they persist in the system.

Example 3. The $\mu\mathcal{L}_P^N$ formula

$$\nu Z.(\exists n.[\#p_1(n) = 1] \wedge \mu Y.([\#p_2(n) = 1]) \vee (\textsc{live}(n) \wedge \langle\!-\!\rangle Y)) \wedge [\!-\!]Z$$

expresses that in every state of the system, place p_1 must contain a single name n that persists in the system until it is the unique name present in p_2. Instead, formula

$$\nu Z.(\exists n.[\#p_1(n) = 1] \wedge \mu Y.([\#p_2(n) = 1]) \vee (\textsc{live}(n) \rightarrow \langle\!-\!\rangle Y)) \wedge [\!-\!]Z$$

expresses that in every state of the system, place p_1 must contain a single name n that either disappears from the system or becomes eventually the unique name present in p_2. ■

By exploiting again Theorem 12, we finally obtain:

Theorem 15. *Verification of $\mu\mathcal{L}_P^N$ properties over state-bounded marked ν-PNs is decidable, and reducible to finite-state model checking.*

Proof. Let $\overline{N}$ be a ν-PN. By adopting the same line of reasoning of the proof for Theorem 13, we get that if $\overline{N}$ is state-bounded, then the DCDS $\tau(\overline{N})$ is state-bounded (in the sense defined in [2]). It is easy to see that if the translation function ξ is applied to a $\mu\mathcal{L}_P^N$ formula Φ, the resulting formula $\xi(\Phi)$ is in $\mu\mathcal{L}_P$. The claim is then obtained by combining Theorem 12 with the fact that verification of $\mu\mathcal{L}_P$ properties over state-bounded DCDSs is decidable, and reducible to finite-state model checking [2]. □

6 Conclusion

We have studied the decidability boundaries related to the verification of ν-PNs against rich temporal properties specified using first-order variants of the μ-calculus. The decidability results are obtained via a translation to DCDSs, showing that interesting, new results can be obtained by cross-fertilizing research areas that have not been extensively related so far. It is interesting to observe that checking whether a DCDS is state-bounded is undecidable even when it is very restricted [3]. Thanks to the decidability of state-boundedness for ν-PNs, the DCDSs obtained from our translation mechanism represent an interesting DCDS fragment for which state-boundedness is indeed decidable to check. We plan to continue the combined investigation of these research areas, and foresee a systematic transfer of results between classes of colored Petri nets and DCDSs, leveraging on the connections drawn in this paper.

References

1. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: Verification of relational data-centric dynamic systems with external services. Corr technical report, arXiv.org e-Print archive (2012), available at `http://arxiv.org/abs/1203.0024`
2. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: Verification of relational data-centric dynamic systems with external services. In: Proc. of PODS. pp. 163–174. ACM (2013)
3. Bagheri Hariri, B., Calvanese, D., Deutsch, A., Montali, M.: State boundedness in data-aware dynamic systems. In: Proc. of KR (2014)
4. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data aware process analysis: A database theory perspective. In: Proc. of PODS (2013)
5. Decker, G., Weske, M.: Instance isolation analysis for service-oriented architectures. In: Proc. of SCC. pp. 249–256. IEEE Computer Society (2008)
6. Demri, S., Lazic, R.: Ltl with the freeze quantifier and register automata. ACM Trans. Comput. Log. 10(3) (2009)
7. Esparza, J.: On the decidability of model checking for several μ-calculi and petri nets. In: Proc. of CAAP. LNCS, vol. 787, pp. 115–129. Springer (1994)
8. Reisig, W., Rozenberg, G. (eds.): Lectures on Petri Nets I: Basic Models, LNCS, vol. 1491. Springer (1998)
9. Rosa-Velardo, F., de Frutos-Escrig, D.: Name creation vs. replication in petri net systems. Fundam. Inform. 88(3), 329–356 (2008)
10. Rosa-Velardo, F., de Frutos-Escrig, D.: Decidability and complexity of petri nets with unordered data. Theor. Comput. Sci. 412(34), 4439–4451 (2011)

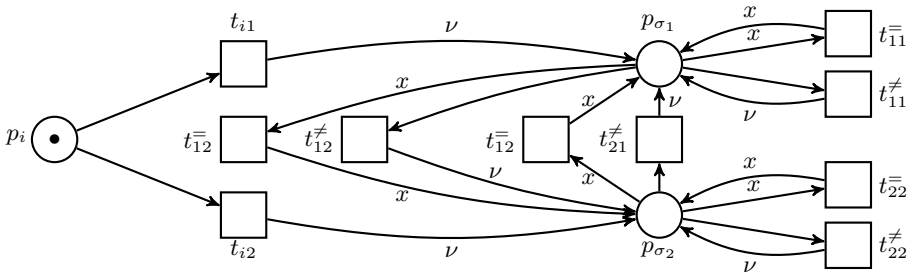

Fig. 4: Marked ν-PN representing all possible infinite data words with labels $\{\sigma_1, \sigma_2\}$.

A From ν-PNs to DCDSs

We define a translation function τ that, given a marked ν-PN $\overline{N} = (P, T, F, m_0)$, produces a DCDS $\tau(\overline{N}) = \langle \mathcal{D}_N, \mathcal{P}_N \rangle$, whose execution semantics faithfully reproduces that of $\Gamma_{\overline{N}}$. The data layer of $\tau(N)$ is $\mathcal{D}_N = \langle \mathcal{C}_N, \mathcal{R}_N, \mathcal{E}_N, \mathcal{I}_0^N \rangle$, where:

1. $\mathcal{C}_N = Id \cup T \cup P \cup Var$
2. $\mathcal{R}_N$ contains:
   - $p_i/2$ for each $p_i \in P$
   - $NewID/5 = NewID(\mathsf{t}, \mathsf{p}, \mathsf{v}, \mathsf{d}, \mathsf{id})$
   - $FL/0$
3. $\mathcal{E}_N$ is constituted by the following constraints:
   - $\mathcal{E}_1$ and $\mathcal{E}_2$ ensure that the new identifiers from $NewID$ are unique:

$$\mathcal{E}_1 = \forall id. NewID(\_, \_, \_, id, \_) \wedge \bigvee_{p_i \in \mathcal{R}_N} p_i(id, \_) \rightarrow \bot$$

$$\mathcal{E}_2 = \forall id, t, p, v, n, t', p', v', n'.\ NewID(t, p, v, id, n) \wedge NewID(t', p', v', id, n') \rightarrow t = t' \wedge p = p' \wedge v = v' \wedge n = n'$$

   - $\mathcal{E}_3$ and $\mathcal{E}_4$ ensure the uniqueness of the fresh names stored in $NewID$ (here Υ_N contains all the fresh variables present in N):

$$\mathcal{E}_3 = \forall n. \bigvee_{\mathsf{c} \in \Upsilon_N} NewID(\_, \_, \mathsf{c}, \_, n) \wedge \bigvee_{i \in \{1,\ldots,5\}} p_i(\_, n) \rightarrow \bot$$

$$\mathcal{E}_4 = \forall n_1, n_2. \bigvee_{\nu_1, \nu_2 \in \Upsilon_N, \nu_1 \neq \nu_2} \Big(NewID(\_, \_, \nu_1, \_, n_1) \wedge NewID(\_, \_, \nu_2, \_, n_2) \rightarrow n_1 \neq n_2 \Big)$$

4. for each $p \in P$ such that $M_0(p) \neq \emptyset$, $\mathcal{I}_0^N$ is containing the following set of facts:

$$\Big\{ p(\mathsf{id}_k^p, \mathsf{d}) :\ \mathsf{d} \in M_0(p),\ \mathsf{id}_k^p \in X_p \subset \mathbb{N},\ \forall i, j, .i \neq j \text{ we have } \mathsf{id}_i^p \neq \mathsf{id}_j^p, \text{ and } \forall p, p'. X_p \cap X_{p'} = \emptyset \Big\}$$

The process layer of $\tau(N)$ is $\mathcal{P}_N = \langle \mathcal{F}_N, \mathcal{P}_N, \rho_N \rangle$, where:

1. $\mathcal{F}_N$ includes the following non-deterministic services:
   - a new id generator $genID(\mathsf{t}, \mathsf{p}, \mathsf{v}, \mathsf{i})$,
   - a fresh data generator $genFresh(\nu)$.
2. Sets of process and action components ρ_N and $\mathcal{A}_N$ are constructed as follows:
   - for each transition $t \in T$ we define *(i)* a process

$$\bigwedge_{CA^1_G} (p_j(id^j_k, v) \wedge id^j_k \neq id^j_m) \wedge \neg FL \mapsto gen_t(\vec{v}, \vec{id}),$$

where $CA^1_G = \Big\{ p_j \in^\bullet t,\ m, k \in \{1, ..., |pre(t, p_j)|\},\ m \neq k,\ v \in pre(t, p_j) \Big\}$ is the guard condition, and *(ii)* a corresponding action $gen_t(\vec{v}, \vec{id}) =$

$$\begin{array}{rll}
\{\, p_j(id, n) \wedge \bigwedge_{i \in I_{p_j}} id \neq id^j_i \rightsquigarrow \{p_j(id, n)\} & \text{for } p_j \in {}^\bullet t & \\
 & \text{and } I_{p_j} = \{1, ..., |pre(t, p_j)|\}\} & \\
p_i(id, n) \rightsquigarrow \{p_i(id, n)\} & \text{for } p_i \in P \setminus {}^\bullet t & \\
true \rightsquigarrow \{NewID(\mathsf{t}, \mathsf{p_k}, \mathsf{x}, genID(\mathsf{t}, \mathsf{p_k}, \mathsf{x}, \mathsf{num_x}), v) : A_1\}, & & \\
 & & \\
true \rightsquigarrow \{NewID(\mathsf{t}, \mathsf{p_k}, \nu, genID(\mathsf{t}, \mathsf{p_k}, \nu, 1), & & \\
\quad genName(\nu)) : A_2\}, & & \\
true \rightsquigarrow FL\,\}, & &
\end{array}$$

where A_1 and A_2 are two conditions which are defined as follows:
(a) $A_1 = \Big\{ p_k \in t^\bullet, \mathsf{x} \in post(t, p_j) \setminus \Upsilon, \mathsf{num_x} = \{1, ..., post(t, p_j)(var)\} \Big\}$,
(b) $A_2 = \Big\{ p_k \in t^\bullet, \nu \in post(t, p_j) \cap \Upsilon \} \Big\}$;

   - define an additional action

$$\begin{array}{rll}
transf() = \{ \quad p_i(id, n) \rightsquigarrow \{p_i(id, n)\} & \text{for } p_i \in P & \\
NewId(_, \mathsf{p_j}, _, id, n) \rightsquigarrow \{p_j(id, n)\} & \text{for } p_j \in P\ \} &
\end{array}$$

and a condition-action rule corresponding to it:

$$FL \mapsto transf()$$