Identifying Duplicates in Large Collections of Petri Nets and Nested-Unit Petri Nets

Pierre Bouvier Hubert Garavel

Univ. Grenoble Alpes, Inria, Cnrs, Grenoble Inp, Lig, France
E-mail: {pierre.bouvier,hubert.garavel}@inria.fr

Abstract

We propose efficient techniques for detecting isomorphism between nets, i.e., for identifying, in large collections of (safe) Petri nets or Nested-Unit Petri Nets, all the nets that are identical modulo a permutation of places, a permutation of transitions, and/or a permutation of units. Our approach relies upon the successive application of diverse algorithms of increasing complexities: net signatures, net canonizations, and characterization of isomorphic nets in terms of isomorphic graphs. We implemented this approach in a complete tool chain that we successfully assessed on four collections, the largest of which comprises 241,000+ nets with many duplicates.

1 Introduction

The present work deals with large collections of Petri nets developed for non-regression testing or software competitions. Building and properly maintaining on the long run such collections, which gather hundreds or thousands of nets, requires a substantial amount of work. A common problem is the presence of *duplicates*, i.e., multiple occurrences of nets that are identical or very similar. Duplicates may be present for three reasons: (i) the collection consists of nets sent by different contributors; (ii) the collection is managed by several persons, who may insert the same net independently; (iii) duplicates may arise from transformations applied to existing models, e.g., conversion of colored Petri nets to P/T nets, removal of dead places or dead transitions [3], etc.

In practice, duplicates are undesirable for at least four reasons: (i) they waste disk space and backup storage, especially when nets are encoded in XML-based formats (such as the standard PNML format [11]), which are particularly verbose; (ii) they waste processor time in redundant calculations, which may be expensive due to state-space explosion issues; (iii) they may introduce biases in benchmarking experiments and software competitions by increasing the weight

of certain nets unduly; (iv) their presence often raises time-consuming questions and debates between users and administrators of net collections.

The present article addresses this problem by proposing methods and tools to detect duplicates that may be present in existing net collections, and to prevent duplicates from being created when new nets are inserted in collections under construction.

The classes of nets considered are one-safe P/T nets and NUPNs (Nested-Units Petri Nets) [7], an extension of Petri nets with the concept of *units*, which provide for modularity and hierarchy. Non-safe P/T nets are also partially supported in the sense that, for such nets, our approach produces over-approximated results, i.e., reports a superset of duplicates, possibly including false positives.

To formalize the problem, we consider that two nets are duplicates if there exist a bijective mapping between their places, their transitions and, in the case of NUPNs, their units. This mapping should preserve the arcs, the initial markings, and, in the case of NUPNs, the root units, the nesting of units, and the location of places in units. Such a definition extends the classical notion of *graph isomorphism* to Petri nets and NUPNs, keeping in mind that Petri nets are directed bipartite graphs, which NUPNs extend with a tree of units.

The detection of duplicates therefore amounts to the efficient partition of a set of nets according to such a *net isomorphism* relation. To the best of our knowledge, there is little prior work on this problem, probably because the construction of large net collections is a recent phenomenon.

The present article is organized as follows. Section 2 gives preliminary definitions. The next sections introduce the various approaches we developed for detecting duplicates: Section 3 exposes how the problem can be reduced to graph isomorphism; Section 4 presents the concept of net signatures; and Section 5 discusses the idea of net canonization. Section 6 presents the integration of all these approaches in a coherent tool chain. Section 7 gives experimental results obtained on four collections of Petri nets and NUPNs. Finally, Section 8 gives a few concluding remarks.

2 Definitions

2.1 Petri Nets and Nested-Unit Petri Nets

We briefly recall the usual definitions of Petri nets and refer the reader to classical surveys, e.g., [13], for a more detailed presentation of Petri nets.

Definition 1 *A (marked)* Petri Net *is a 4-tuple* (P, T, F, M_0) *where:*

1. P *is a finite, non-empty set; the elements of* P *are called* places.
2. T *is a finite set such that* $P \cap T = \varnothing$*; the elements of* T *are called* transitions.
3. F *is a subset of* $(P \times T) \cup (T \times P)$*; the elements of* F *are called* arcs.
4. M_0 *is a non-empty subset of* P*;* M_0 *is called the* initial marking.

Notice that the above definition only covers *ordinary* nets (i.e., it assumes all arc weights are equal to one). Also, it only considers *safe* nets (i.e., each place contains at most one token), which enables the initial marking to be defined as a subset of P, rather than a function $P \to \mathbb{N}$ as in the usual definition of P/T nets. We now recall the basic definition of a NUPN, referring the interested reader to [7] for a complete presentation of this model of computation.

Definition 2 *A (marked)* Nested-Unit Petri Net *(acronym: NUPN) is a 8-tuple* $(P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *where* (P, T, F, M_0) *is a Petri net, and where:*

5. U *is a finite, non-empty set such that* $U \cap T = U \cap P = \varnothing$*; the elements of* U *are called* units*.*
6. u_0 *is an element of* U*;* u_0 *is called the* root unit*.*
7. $\sqsubseteq$ *is a binary relation over* U *such that* $(U, \sqsupseteq)$ *is a tree with a single root* u_0*, where* $(\forall u_1, u_2 \in U)\ u_1 \sqsupseteq u_2 \stackrel{\text{def}}{=} u_2 \sqsubseteq u_1$*; intuitively*[1]*,* $u_1 \sqsubseteq u_2$ *expresses that unit* u_1 *is transitively nested in or equal to unit* u_2*.*
8. unit *is a function* $P \to U$ *such that* $(\forall u \in U \setminus \{u_0\})\ (\exists p \in P)\ \mathsf{unit}(p) = u$*; intuitively,* $\mathsf{unit}(p) = u$ *expresses that unit* u *directly contains place* p*.*

We now recall a few usual definitions for ordinary safe nets.

Definition 3 *Let* (P, T, F, M_0) *be a Petri Net.*

- *A* marking M *is defined as a set of places* ($M \subseteq P$)*. Each place belonging to a marking* M *is said to be* marked *or, also, to possess a* token*.*
- *The* pre-set *of a transition* t *is the set of places* ${}^\bullet t \stackrel{\text{def}}{=} \{p \in P \mid (p, t) \in F\}$*.*
- *The* post-set *of a transition* t *is the set of places* $t^\bullet \stackrel{\text{def}}{=} \{p \in P \mid (t, p) \in F\}$*.*
- *The* pre-set *of a place* p *is the set of transitions* ${}^\bullet p \stackrel{\text{def}}{=} \{t \in T \mid (t, p) \in F\}$*.*
- *The* post-set *of a place* p *is the set of transitions* $p^\bullet \stackrel{\text{def}}{=} \{t \in T \mid (p, t) \in F\}$*.*

Because NUPNs merely extend Petri nets by grouping places into units, Petri-net properties (including the standard firing rules for transitions) are preserved when NUPN information is added. Thus, all the concepts of Def. 3 for Petri nets also apply to NUPNs. The next definition provides useful notations used throughout this article.

Definition 4 *Let* $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN.*

- $u_1 \sqsubset u_2 \stackrel{\text{def}}{=} (u_1 \sqsubseteq u_2)\ \ \&\ \ (u_1 \neq u_2)$ *is the strict nesting partial order.*
- $\mathsf{disjoint}(u_1, u_2) \stackrel{\text{def}}{=} (u_1 \not\sqsubseteq u_2)\ \ \&\ \ (u_2 \not\sqsubseteq u_1)$ *characterizes pairs of units neither equal nor nested one in the other.*
- $\mathsf{places}(u) \stackrel{\text{def}}{=} \{p \in P \mid \mathsf{unit}(p) = u\}$ *gives all places directly contained in* u*; these are called the* local places *(or* proper places*) of* u*.*

[1] $\sqsubseteq$ is reflexive, antisymmetric, transitive, and u_0 is the greatest element of U for $\sqsubseteq$.

- $\mathsf{places}^*(u) \stackrel{\text{def}}{=} \{p \in P \mid (\exists u' \in U)\ (u' \sqsubseteq u)\ \ \&\ \ (\mathsf{unit}\,(p) = u')\}$ *gives all places transitively contained in* u *or its sub-units.*
- $\mathsf{subunits}\,(u) \stackrel{\text{def}}{=} \{u' \in U \mid (u' \sqsubset u)\ \ \&\ \ (\nexists u'' \in U)\ (u' \sqsubset u'')\ \ \&\ \ (u'' \sqsubset u)\}$ *gives all units directly nested in* u.
- $\mathsf{subunits}^*(u) \stackrel{\text{def}}{=} \{u' \in U \mid (u' \sqsubset u)\}$ *gives all units transitively nested in* u.
- $\mathsf{leaf}\,(u) \stackrel{\text{def}}{=} (\mathsf{subunits}\,(u) = \varnothing)$ *characterizes the units having no nested sub-unit, i.e., the minimal elements of* $(U, \sqsubseteq)$.
- $\mathsf{depth}\,(u)$ *is the length of the longest chain* $u \sqsubset ... \sqsubset u_0$ *of nested units from* u *to the root unit* u_0. *In particular,* $\mathsf{depth}\,(u_0) = 0$.
- $\mathsf{height}\,(u)$ *is the length, plus one, of the longest chain* $u_n \sqsubset ... \sqsubset u$ *of nested units from any leaf unit* u_n *to* u. *In particular,* $\mathsf{leaf}\,(u) \Leftrightarrow \mathsf{height}\,(u) = 1$.
- $\mathsf{width}\,(u)$ *is the number of leaf units contained in* $\{u\} \cup \mathsf{subunits}^*(u)$.
- *A* trivial *NUPN is such that* $\mathsf{width}\,(u_0)$ *equals the number of places* $\mathrm{card}\,(P)$, *meaning that the net carries no more NUPN information than a Petri net; in a trivial NUPN, each unit has a single local place, except the root unit* u_0, *which has either zero or one.*

Finally, a NUPN N is said to be *unit safe* [7] iff its underlying Petri net (P, T, F, M_0) is one-safe and, in any reachable marking M, all the places of M are contained in disjoint units.

2.2 Graph and Net Isomorphisms

We first recall the classical definitions of *graph* and *graph isomorphism*.

Definition 5 *A* vertex-colored directed graph *(or* colored graph *for short) is a 3-tuple* (V, E, c) *such that:* V *is a set of vertices,* $E \subseteq V \times V$ *is a set of (directed) edges, and* $c : V \to \mathbb{N}$ *is a function associating for each vertex a natural number representing a color. If the relation* E *is symmetric, then the graph is said to be* undirected.

Definition 6 *Two colored graphs* $G = (V, E, c)$ *and* $G' = (V', E', c')$ *are* isomorphic *iff there exists a bijection* $\pi_v : V \to V'$ *such that:*

- $(\forall v_1, v_2 \in V)\ (v_1, v_2) \in E \Leftrightarrow (\pi_v(v_1), \pi_v(v_2)) \in E'$.
- $(\forall v \in V)\ c(v) = c'(\pi_v(v))$.

We then define the concept of *net isomorphism* used throughout this article.

Definition 7 *Let* $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *and* $N' = (P', T', F', M'_0, U', u'_0, \sqsubseteq', \mathsf{unit}')$ *be two NUPNs.* N *and* N' *are said to be* isomorphic *iff there exist three bijections* $\pi_p : P \to P'$, $\pi_t : T \to T'$, *and* $\pi_u : U \to U'$ *such that:*

- $(\forall (p, t) \in P \times T)\ (p, t) \in F \Leftrightarrow (\pi_p(p), \pi_t(t)) \in F'$.
- $(\forall (t, p) \in T \times P)\ (t, p) \in F \Leftrightarrow (\pi_t(t), \pi_p(p)) \in F'$.

- $(\forall p \in P)\ p \in M_0 \Leftrightarrow \pi_p(p) \in M'_0$.
- $u'_0 = \pi_u(u_0)$.
- $(\forall u_1, u_2 \in U)\ u_1 \sqsubseteq u_2 \Leftrightarrow \pi_u(u_1) \sqsubseteq' \pi_u(u_2)$ — *or, expressed in an equivalent way:* $(\forall u_1, u_2 \in U)\ u_1 \in \mathsf{subunits}\,(u_2) \Leftrightarrow \pi_u(u_1) \in \mathsf{subunits}'(\pi_u(u_2))$.
- $(\forall p \in P)\ \mathsf{unit}'(\pi_p(p)) = \pi_u(\mathsf{unit}\,(p))$.

Alternative definitions of net isomorphism can be found in the literature. The definition given in [10] takes into account places, place labels, transitions, transition labels, and arc weights, but not the initial marking. In [6, 5] and [8], a less general definition of net isomorphism is given, in which only places are considered: two isomorphic Petri nets may have their places permuted but must have identical transitions. In [4], [1], and [2], two nets are said to be isomorphic iff their marking graphs are isomorphic; notice that the left-to-right implication of this behavioural definition is also ensured by our purely structural Definition 7.

Finally, we recall the notion of disjoint union on sets and functions.

Definition 8 *Let S and S' be two disjoint sets. Let f and f' be two functions respectively defined on S and S'.*

- *$S \uplus S'$ denotes the* set union *$S \cup S'$, knowing that $S \cap S' = \varnothing$.*
- *$f \uplus f'$ denotes the* function union *of f and f', i.e., the function defined on $S \uplus S'$ such that $(f \uplus f')(x) = f(x)$ if $x \in S$ and $(f \uplus f')(x) = f'(x)$ if $x \in S'$.*

3 Net Isomorphism in Terms of Graph Isomorphism

Our first approach expresses net isomorphism in terms of graph isomorphism, a problem for which software tools are available [9].

3.1 Theoretical Aspects

Definition 9 *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. We associate to N a colored directed graph $\mathcal{G}_N = (V, E, c)$ such that:*

- $V \stackrel{\text{def}}{=} P \uplus T \uplus U$.
- $E \stackrel{\text{def}}{=} F \uplus \{(p, u) \in P \times U \mid u = \mathsf{unit}\,(p)\} \uplus \{(u, u') \in U \times U \mid u \in \mathsf{subunits}\,(u')\}$.
- $(\forall v \in V)\ c(v) \stackrel{\text{def}}{=} 0$ *if* $v \in P \setminus M_0, 1$ *if* $v \in M_0, 2$ *if* $v \in T$, *or* 3 *if* $v \in U$.

The function $N \to \mathcal{G}_N$ is an injection, but not a surjection (as not every graph corresponds to a NUPN). The following definition computes the inverse function.

Definition 10 *Let $G = (V, E, c)$ be the colored directed graph associated to some NUPN via Def. 9. Let $\mathcal{N}_G \stackrel{\text{def}}{=} (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be defined as follows:*

- $P \stackrel{\text{def}}{=} \{v \in V \mid c(v) \leq 1\}$.

- $T \stackrel{\text{def}}{=} \{v \in V \mid c(v) = 2\}$.
- $F \stackrel{\text{def}}{=} E \cap ((P \times T) \cup (T \times P))$.
- $M_0 \stackrel{\text{def}}{=} \{v \in V \mid c(v) = 1\}$.
- $U \stackrel{\text{def}}{=} \{v \in V \mid c(v) = 3\}$.
- u_0 *is the unique element such that* $(u_0 \in U)$ & $(E \cap (\{u_0\} \times U) = \varnothing)$.
- $\sqsubseteq$ *is the reflexive transitive closure of the relation* $E \cap (U \times U)$.
- unit *is the function* $p \mapsto u$ *such that* $(p, u) \in E \cap (P \times U)$.

It is easy to see that $\mathcal{N}_G$ is a NUPN, i.e., that: P, T, and U are pairwise disjoint; M_0 is not empty and contained in P; and $\sqsubseteq$ is a tree with a single root u_0.

Proposition 1 *Two NUPNs N and N' are isomorphic iff their corresponding graphs $\mathcal{G}_N$ and $\mathcal{G}_{N'}$ are isomorphic.*

Proof. Let $G \stackrel{\text{def}}{=} \mathcal{G}_N = (V, E, c)$ and let $G' \stackrel{\text{def}}{=} \mathcal{G}_{N'} = (V', E', c')$. By double implication. Direct: Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ and let $N' = (P', T', F', M_0', U', u_0', \sqsubseteq', \mathsf{unit}')$. If N and N' are isomorphic, there exist three bijections π_p, π_t, and π_u satisfying the conditions of Def. 7. The function $\pi_v \stackrel{\text{def}}{=} \pi_p \uplus \pi_t \uplus \pi_u$ is a bijection from V to V' and satisfies both conditions of Def. 6: it preserves the edges (proven by disjunction of cases, depending whether each edge of E belongs to $P \times T$, $T \times P$, $U \times P$, or $U \times U$) and preserves the colors (also proven by disjunction of cases, depending whether each vertex of V belongs to $P \setminus M_0$, M_0, T, or U). Converse: Let $\mathcal{N}_G = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ and let $\mathcal{N}_{G'} = (P', T', F', M_0', U', u_0', \sqsubseteq', \mathsf{unit}')$. If G and G' are isomorphic, there exists a bijection $\pi_v : V \to V'$ satisfying the two conditions of Def. 6. The second condition, the bijective nature of π_v, and the definitions of c and c' in Def. 9 imply that $\operatorname{card}(P) = \operatorname{card}(P')$, $\operatorname{card}(T) = \operatorname{card}(T')$, $\operatorname{card}(M_0) = \operatorname{card}(M_0)$, and $\operatorname{card}(U) = \operatorname{card}(U')$. Let $\pi_p \stackrel{\text{def}}{=} \pi_v|_P$, $\pi_t \stackrel{\text{def}}{=} \pi_v|_T$, and $\pi_u \stackrel{\text{def}}{=} \pi_v|_U$ be the restrictions of π_v to P, T, and U, respectively. Because π_v is a bijection, π_p is an injection from P to P', and even a bijection since $\operatorname{card}(P) = \operatorname{card}(P')$; similarly, π_t is a bijection from T to T' and π_u a bijection from U to U'. The six conditions of Def. 7 then follow from the combined assumptions of Def. 6 and Def. 9.

3.2 Practical Aspects

To assess on concrete examples the efficiency of the approach presented in Sect. 3.1, we selected the two reference tools dedicated to graph isomorphism, NAUTY and TRACES[2] [12] because of their high reputation of efficiency. These tools, which provide both an API and a command-line interface, can put a graph under canonical form or decide whether two graphs are isomorphic (i.e., iff their respective canonical graphs are identical).

[2] `https://pallini.di.uniroma1.it`

As for benchmark, we selected the 1387 (non-colored) Petri Nets and NUPNs used for the 2022 edition of the Model Checking Contest, knowing that duplicates are present in this collection. Using a Python script implementing Def. 9, each net was translated to a graph in NAUTY/TRACES input format. We ran our experiments in parallel on the French Grid'5000 testbed[3], allocating to each model a dedicated server with 96 GB RAM and one hour of wallclock time.

The results were disappointing: NAUTY managed to put 310 graphs under canonical form (success rate: 22.4%) but failed on all other models, either due to lack of memory (on 8 graphs) or by hitting the one-hour timeout (on 1069 graphs). Furthermore, no duplicate was detected.

To improve these results, we did additional attempts in two directions: (i) devising alternative translations to the one of Def. 9, taking advantage of the specificities of NAUTY to get as much performance as possible, and (ii) experimenting also with TRACES, which is more recent and slightly faster (by a few percents, as we observed) than NAUTY.

Rather than assigning to vertices four colors only (i.e., $\{0...3\}$ in Def. 9), one may increase the number of colors to better distinguish between the various vertices v_p associated to all places $p \in P$. For instance, one may choose $c(v_p) \stackrel{\text{def}}{=} \mathsf{depth}(\mathsf{unit}(p))$, together with $c(v) \stackrel{\text{def}}{=} \mathsf{height}(u_0)$ if $v \in T$ and $c(v) \stackrel{\text{def}}{=} \mathsf{height}(u_0) + 1$ if $v \in U$ — keeping in mind that, for each $u \in U$, $\mathsf{depth}(u) < \mathsf{height}(u_0)$. With such colors, the information that a place p belongs to the initial marking M_0 can be expressed differently, e.g., by adding a looping arc (v_p, v_p) to E (noticing that E contains no arcs from P to P) or by adding a special vertex v_0 with a unique color and an arc (v_p, v_0).

Another idea is to reduce the number of vertices by no longer associating a vertex to each unit (i.e., $V \stackrel{\text{def}}{=} \{v_0\} \uplus P \uplus T$). In this approach, the root unit u_0, the function unit, and the relation $\sqsubseteq$ can be encoded by extra arcs, e.g., by adding an arc (v_0, v_p) for each place $p \in \mathsf{places}(u_0)$, and by adding an arc $(v_p, v_{p'})$ for each pair of places p and p' such that $\mathsf{unit}(p) \in \mathsf{subunits}(\mathsf{unit}(p'))$.

Because TRACES does not support directed graphs, we adapt the translation of Def. 9 by associating two unique vertices v_p and v'_p to each place $p \in P$, assigning distinctive colors to v_p and v'_p (e.g., $c(v_p) \stackrel{\text{def}}{=} 2 \times \mathsf{depth}(\mathsf{unit}(p))$ and $c(v'_p) \stackrel{\text{def}}{=} c(v_p) + 1$), and adding an edge $\{v_p, v'_p\}$ to express that both vertices are related to the same place. As before, a unique vertex v_t is associated to each transition $t \in T$. Then, each arc $(t, p) \in F$ is represented by an edge $\{v_t, v_p\}$ and each arc $(p, t) \in F$ is represented by an edge $\{v'_p, v_t\}$.

We implemented these ideas in five different translations, which we assessed on the aforementioned benchmark (2022 edition of the Model Checking Contest). In the most effective approach, NAUTY managed to put 498 graphs under canonical form (success rate: 35.9%) but failed due to lack of memory (on 15 graphs) or by

[3] https://www.grid5000.fr

hitting the one-hour timeout (on 874 graphs). Again, no duplicate was detected.

Thus, even if net isomorphism can theoretically be expressed in terms of graph isomorphism, this does not seem to be a practical solution. We now present alternative approaches specifically tailored for Petri nets and NUPNs.

4 Net Signatures

Our second approach is based on the idea of *net signature*, which borrows from the concepts of hash and checksum functions.

Definition 11 *A* net signature *(or* signature *for short) is a function* sig *defined on Petri nets or NUPNs, such that, for any two nets* N *and* N'*, if* N *and* N' *are isomorphic, then* $\mathsf{sig}(N) = \mathsf{sig}(N')$.

In practice, one uses the contraposition of this implication: two nets having different signatures are not isomorphic. The reverse implication is not required: two nets having the same signature are not necessarily isomorphic (there is a risk of collision between their signatures).

Proposition 2 *If* sig *is a signature, then for any net* N *and any permutation* π *of places, transitions, and/or units,* $\mathsf{sig}(\pi(N)) = \mathsf{sig}(N)$.

To discriminate as many nets as possible, a signature should be extensive enough to contain all information that is invariant by permutations, but it should also be fast to compute. We now introduce a few definitions upon which an effective signature can be built.

4.1 Multiset Hashing

Multisets are an extension of sets and may contain several instances of each element. We note multisets $\{\!|\ ...\ |\!\}$ to distinguish them from (normal) sets, noted $\{...\}$. Given a NUPN $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$, two examples of multisets are $\{\!|\ \mathsf{card}\,({}^\bullet t)\ \mid\ t \in T\ |\!\}$ and $\{\!|\ \mathsf{height}\,(u)\ \mid\ u \in U\ |\!\}$. To check the equality of such multisets, the lengths of which can be fixed or variable, we adopt a hash-based approach that converts each multiset into a (fixed-size) *digest*, such that the equality of two multisets implies the equality of the two corresponding digests. Thus, the chosen hash function is not assumed to be perfect (hash collisions may exist among digests). Yet, it is desirable that this function returns a result independent from the order of elements (multisets are not lists). A simple solution would be to sort the elements of a multiset and concatenate them to form a bit string on which some standard (cryptographic or not) hash function would be applied. However, this approach is slow (due to sorting, at least) and produces hash results that are not meaningful to humans. We thus adopt

an alternative approach based on the following hash function, which does not require sorting and returns a tuple, many fields of which can be easily checked by inspection.

Definition 12 *Let a digest be a 5-tuple of natural numbers.*

- *Let* $\mathbb{D} \stackrel{\text{def}}{=} \mathbb{N}^5$ *denote the set of digests.*
- *For* $d \in \mathbb{D}$*, let* $d.\mathsf{card}$*,* $d.\mathsf{min}$*,* $d.\mathsf{max}$*,* $d.\mathsf{sum}$*, and* $d.\mathsf{prod}$ *denote, respectively, each of the five components of* d*.*
- *Let* $\mathcal{H}$: (**multiset of** $\mathbb{N}$) $\rightarrow \mathbb{D}$ *be the hash function defined as follows:* $\mathcal{H}(\varnothing) \stackrel{\text{def}}{=} (0,0,0,0,1)$ *and, for any natural* $n \geq 1$*,* $\mathcal{H}(\{\!|\, x_1, ..., x_n \,|\!\}) \stackrel{\text{def}}{=} (n, min(x_1, ..., x_n), max(x_1, ..., x_n), x_1 + ... + x_n, (2x_1 + r) \times ... \times (2x_n + r)/2)$*, where* r *is the constant* $2,654,435,769$*.*
- *Let* $\mathcal{M}$: (**multiset of** $\mathbb{D}$) $\rightarrow \mathbb{D}$ *be the "hash-merge" function defined as follows:* $\mathcal{M}(\varnothing) \stackrel{\text{def}}{=} (0,0,0,0,1)$ *and, for any* $n \geq 1$*,* $\mathcal{M}(\{\!|\, d_1, ..., d_n \,|\!\}) \stackrel{\text{def}}{=} (d_1.\mathsf{card} + ... + d_n.\mathsf{card}, min(d_1.\mathsf{min}, ..., d_n.\mathsf{min}), max(d_1.\mathsf{max}, ..., d_n.\mathsf{max}), d_1.\mathsf{sum} + ... + d_n.\mathsf{sum}, (2 \times d_1.\mathsf{prod} + 1) \times ... \times (2 \times d_n.\mathsf{prod} + 1)/2)$*.*

Function $\mathcal{H}$ handles multisets of natural numbers, whereas function $\mathcal{M}$, at a higher level ("hash of hashes"), handles multisets of digests. Both functions can be computed by induction on the size n of their input multisets: there is no need for preliminary sorting, as all the operations involved in $\mathcal{H}$ and $\mathcal{M}$ are commutative and associative. In practice, the fields of $\mathbb{D}$ are implemented using machine integers, so that all arithmetical calculations are done modulo, e.g., 2^{32} or 2^{64}. All components of $\mathbb{D}$, but prod, are readable by humans and express meaningful properties of the corresponding multiset. Instead, prod uses a form of multiplicative hashing[4] that seeks to enhance dispersion for large multisets. Notice that, all factors of prod being odd, their product never becomes zero, even under modular arithmetic; the final division of this product by two eliminates the least significant bit, which is always equal to one.

4.2 Signature Function

We can propose a particular sig function defined on NUPNs; this function supports ordinary, safe Petri nets as a particular case (i.e., trivial NUPNs).

Definition 13 *Let* $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. We define* $\mathsf{sig}(N)$ *to be a fixed-size tuple, each component of which is a natural number or a digest computed from* N*. The components are divided into three parts, respectively based on the places, transitions, and units of* N*. These parts, noted* $(h_1, ...)$*,* $(k_1, ...)$*, and* $(l_1, ...)$*, are defined below (see Def. 18, 19, and 20).*

[4] https://stackoverflow.com/questions/1536393/good-hash-function-for-permutations

4.3 Attributes for Places and Transitions

The definition of our signature function relies on various attributes computed for each place and transition of the net. These attributes contain information that helps differentiating the various places (resp. transitions). At first sight, all places are seemingly alike (except those of the initial marking), but they can be distinguished using local information (e.g., the number of arcs and transitions connected to them) as well as global information (e.g., their distance to other remarkable places of the net: initial places, sink places, etc.).

Definition 14 *Let $N = (P, T, F, M_0)$ be a Petri net. A set of places $p_0, ..., p_n$ and a set of transitions $t_1, ..., t_n$ are said to be a* chain of length n from p_0 to p_n *iff* $(\forall i \in \{1, ..., n\})\ (p_{i-1}, t_i) \in F\ \ \&\ \ (t_i, p_i) \in F$. *Given two places p and p', the* distance from p to p' *is defined as the length of the shortest chain from p to p'; if no such chain exists, this distance is equal to* $\mathrm{card}\,(P) + 1$.

Definition 15 *To each place p, one associates three attributes:*

- $\mathsf{distance1}(p)$ *is defined as the minimal distance from p to any place of the initial marking M_0.*
- $\mathsf{distance2}(p)$ *is defined as the minimal distance from any place of the initial marking to p.*
- $\mathsf{distance3}(p)$ *is defined as the minimal distance from p to any sink place (i.e., any place p' such that $p'^{\bullet} = \varnothing$).*

Definition 16 *To each transition t, one associates three Boolean attributes and six attributes of type $\mathbb{D}$:*

- $\mathsf{decreasing}(t) \stackrel{\text{def}}{=} (\mathrm{card}\,({}^{\bullet}t) > \mathrm{card}\,(t^{\bullet}))$
- $\mathsf{conservative}(t) \stackrel{\text{def}}{=} (\mathrm{card}\,({}^{\bullet}t) = \mathrm{card}\,(t^{\bullet}))$
- $\mathsf{increasing}(t) \stackrel{\text{def}}{=} (\mathrm{card}\,({}^{\bullet}t) < \mathrm{card}\,(t^{\bullet}))$
- *for* $i \in \{1, 2, 3\}$, $\mathsf{input_distance}\,i(t) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{distance}\,i(p) \mid p \in {}^{\bullet}t \,|\!\})$
- *for* $i \in \{1, 2, 3\}$, $\mathsf{output_distance}\,i(t) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{distance}\,i(p) \mid p \in t^{\bullet} \,|\!\})$

Definition 17 *To each place p, one associates seven natural-number attributes and sixteen attributes of type $\mathbb{D}$:*

- $\mathsf{nb_loops}(p) \stackrel{\text{def}}{=} \mathrm{card}\,({}^{\bullet}p \cap p^{\bullet})$
- $\mathsf{nb_decreasing_input_transitions}(p) \stackrel{\text{def}}{=} \mathrm{card}\,(\{t \in {}^{\bullet}p \mid \mathsf{decreasing}(t)\})$
- $\mathsf{nb_conservative_input_transitions}(p) \stackrel{\text{def}}{=} \mathrm{card}\,(\{t \in {}^{\bullet}p \mid \mathsf{conservative}(t)\})$
- $\mathsf{nb_increasing_input_transitions}(p) \stackrel{\text{def}}{=} \mathrm{card}\,(\{t \in {}^{\bullet}p \mid \mathsf{increasing}(t)\})$
- $\mathsf{nb_decreasing_output_transitions}(p) \stackrel{\text{def}}{=} \mathrm{card}\,(\{t \in p^{\bullet} \mid \mathsf{decreasing}(t)\})$
- $\mathsf{nb_conservative_output_transitions}(p) \stackrel{\text{def}}{=} \mathrm{card}\,(\{t \in p^{\bullet} \mid \mathsf{conservative}(t)\})$
- $\mathsf{nb_increasing_output_transitions}(p) \stackrel{\text{def}}{=} \mathrm{card}\,(\{t \in p^{\bullet} \mid \mathsf{increasing}(t)\})$

- $\mathsf{pred_nb_input_places}(p) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \text{card}\,({}^\bullet t) \mid t \in {}^\bullet p \,|\!\})$
- $\mathsf{pred_nb_output_places}(p) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \text{card}\,(t^\bullet) \mid t \in {}^\bullet p \,|\!\})$
- $\mathsf{succ_nb_input_places}(p) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \text{card}\,({}^\bullet t) \mid t \in p^\bullet \,|\!\})$
- $\mathsf{succ_nb_output_places}(p) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \text{card}\,(t^\bullet) \mid t \in p^\bullet \,|\!\})$
- *for* $i \in \{1,2,3\}$, $\mathsf{pred_input_distance}\, i(p) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{input_distance}\, i(t) \mid t \in {}^\bullet p \,|\!\})$
  and $\mathsf{pred_output_distance}\, i(p) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{output_distance}\, i(t) \mid t \in {}^\bullet p \,|\!\})$
- *for* $i \in \{1,2,3\}$, $\mathsf{succ_input_distance}\, i(p) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{input_distance}\, i(t) \mid t \in p^\bullet \,|\!\})$
  and $\mathsf{succ_output_distance}\, i(p) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{output_distance}\, i(t) \mid t \in p^\bullet \,|\!\})$

4.4 Signature Part Based on Places

The first part of our signature function is defined as follows.

Definition 18 *Let* $N = (P, T, F, M_0)$ *be a Petri net. The* place-based part *of the* $\mathsf{sig}(N)$ *function of Def. 13 is a tuple* $(h_1, ..., h_{16})$ *of natural numbers or values of type* $\mathbb{D}$*. The components of this tuple are the following:*

- $h_1 \stackrel{\text{def}}{=} \text{card}\,(P)$*, i.e., the number of places.*
- $h_2 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{distance1}(p) \mid p \in P \,|\!\})$.
- $h_3 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{distance2}(p) \mid p \in P \,|\!\})$.
- $h_4 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{distance3}(p) \mid p \in P \,|\!\})$.
- $h_5 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_loops}(p) \mid p \in P \,|\!\})$.
- $h_6 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_decreasing_input_transitions}(p) \mid p \in P \,|\!\})$.
- $h_7 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_conservative_input_transitions}(p) \mid p \in P \,|\!\})$.
- $h_8 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_increasing_input_transitions}(p) \mid p \in P \,|\!\})$.
- $h_9 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_decreasing_output_transitions}(p) \mid p \in P \,|\!\})$.
- $h_{10} \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_conservative_output_transitions}(p) \mid p \in P \,|\!\})$.
- $h_{11} \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_increasing_output_transitions}(p) \mid p \in P \,|\!\})$.
- $h_{12} \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{pair}(\text{card}\,({}^\bullet p), \text{card}\,(p^\bullet)) \mid p \in P \,|\!\})$*, where* $\mathsf{pair} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ *is a pairing function that maps two natural numbers to a single one.*
- $h_{13} \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_nb_input_places}(p) \mid p \in P \,|\!\})$.
- $h_{14} \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_nb_output_places}(p) \mid p \in P \,|\!\})$.
- $h_{15} \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_nb_input_places}(p) \mid p \in P \,|\!\})$.
- $h_{16} \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_nb_output_places}(p) \mid p \in P \,|\!\})$.

The following components are excluded from the place-based part of the signature: $h_2.\mathsf{card}$*,* $h_3.\mathsf{card}$*, ...,* $h_{12}.\mathsf{card}$ *(because they are all equal to* h_1*);* $h_{14}.\mathsf{card}$ *(which is equal to* $h_{13}.\mathsf{card}$*);* $h_{16}.\mathsf{card}$ *(which is equal to* $h_{15}.\mathsf{card}$*); and* $h_{12}.\mathsf{sum}$ *(which a linear combination of* $h_6.\mathsf{sum}$*, ...,* $h_{11}.\mathsf{sum}$*).*

4.5 Signature Part Based on Transitions

The second part of our signature function is defined as follows.

Definition 19 *Let $N = (P, T, F, M_0)$ be a Petri net. The* transition-based part *of the* $\mathsf{sig}(N)$ *function of Def. 13 is a tuple $(k_1, ..., k_3)$ of natural numbers or values of type $\mathbb{D}$. The components of this tuple are the following:*

- $k_1 \stackrel{\text{def}}{=} \mathrm{card}\,(T)$, *i.e., the number of transitions.*
- $k_2 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathrm{card}\,({}^\bullet t) \mid t \in T \,|\!\})$.
- $k_3 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathrm{card}\,(t^\bullet) \mid t \in T \,|\!\})$.

The following components are excluded from the transition-based part of the signature: $k_2.\mathsf{card}$ *and* $k_3.\mathsf{card}$ *(because they are equal to* k_1*);* $k_2.\mathsf{sum}$ *(which is equal to* $h_9.\mathsf{sum} + h_{10}.\mathsf{sum} + h_{11}.\mathsf{sum}$*); and* $k_3.\mathsf{sum}$ *(which is equal to* $h_6.\mathsf{sum} + h_7.\mathsf{sum} + h_8.\mathsf{sum}$*).*

4.6 Signature Part Based on Units

The third part of our signature function is defined as follows.

Definition 20 *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. The* unit-based part *of the* $\mathsf{sig}(N)$ *function of Def. 13 is a tuple $(l_1, ..., l_{13})$ of natural numbers or values of type $\mathbb{D}$. The components of this tuple are the following:*

- $l_1 \stackrel{\text{def}}{=} \mathrm{card}\,(U)$, *i.e., the number of units.*
- $l_2 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathrm{card}\,(\mathsf{subunits}\,(u)) \mid u \in U \,|\!\})$.
- $l_3 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathrm{card}\,(\mathsf{subunits}^*(u)) \mid u \in U \,|\!\})$.
- $l_4 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathrm{card}\,(\mathsf{places}\,(u)) \mid u \in U \,|\!\})$.
- $l_5 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathrm{card}\,(\mathsf{places}^*(u)) \mid u \in U \,|\!\})$.
- $l_6 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathrm{card}\,(\mathsf{places}\,(u) \cap M_0) \mid u \in U \,|\!\})$.
- $l_7 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathrm{card}\,(\mathsf{places}^*(u) \cap M_0) \mid u \in U \,|\!\})$.
- $l_8 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{depth}\,(u) \mid u \in U \,|\!\})$.
- $l_9 \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{height}\,(u) \mid u \in U \,|\!\})$.
- $l_{10} \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{width}\,(u) \mid u \in U \,|\!\})$,
- $l_{11} \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{in}(u) \mid u \in U \,|\!\})$, *where* $\mathsf{in}(u) \stackrel{\text{def}}{=} \sum_{t \in T} \mathrm{card}\,({}^\bullet t \cap \mathsf{places}\,(u))$.
- $l_{12} \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{out}(u) \mid u \in U \,|\!\})$, *where* $\mathsf{out}(u) \stackrel{\text{def}}{=} \sum_{t \in T} \mathrm{card}\,(t^\bullet \cap \mathsf{places}\,(u))$.
- $l_{13} \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{mix}(\mathrm{card}\,(\mathsf{subunits}\,(u)), \mathrm{card}\,(\mathsf{subunits}^*(u)), \mathrm{card}\,(\mathsf{places}\,(u)), \mathrm{card}\,(\mathsf{places}^*(u)), \mathrm{card}\,(\mathsf{places}\,(u) \cap M_0), \mathrm{card}\,(\mathsf{places}^*(u) \cap M_0), \mathsf{depth}\,(u), \mathsf{height}\,(u), \mathsf{width}\,(u), \mathsf{in}(u), \mathsf{out}(u) \,|\!\})$, *where* $\mathsf{mix} : \mathbb{N}^{11} \to \mathbb{N}$ *is a generalized pairing function.*

The following components are excluded from the unit-based part of the signature because their presence would be redundant: $l_2.\mathsf{card} = l_1$, $l_2.\mathsf{min} = 0$, $l_2.\mathsf{sum} =$

$l_1 - 1$, $l_3.\mathsf{card} = l_1$, $l_3.\mathsf{min} = 0$, $l_4.\mathsf{card} = l_1$, $l_4.\mathsf{sum} = \mathrm{card}\,(P)$, $l_5.\mathsf{card} = l_1$, $l_6.\mathsf{card} = l_1$, $l_6.\mathsf{min} \leq 1$ *and* $l_6.\mathsf{max} \leq 1$ *if* N *is unit safe,* $l_7.\mathsf{card} = l_1$, $l_7.\mathsf{min} \leq 1$ *if* N *is unit safe,* $l_7.\mathsf{max} = \mathrm{card}\,(M_0)$, $l_8.\mathsf{card} = l_1$, $l_8.\mathsf{min} = 0$, $l_9.\mathsf{card} = l_1$, $l_9.\mathsf{min} = 1$, $l_9.\mathsf{max} = l_8.\mathsf{max} + 1$, $l_{10}.\mathsf{card} = l_1$, $l_{10}.\mathsf{min} = 1$, $l_{11}.\mathsf{card} = l_1$, $l_{12}.\mathsf{card} = l_1$, *and* $l_{13}.\mathsf{card} = l_1$.

5 Net Canonization

Our third approach relies on an idea derived from the concept of normal form.

Definition 21 *A* net canonization *(or* canonization *for short) is a function* can *defined on Petri nets or NUPNs, such that, for any two nets* N *and* N'*, if* $\mathsf{can}(N) = \mathsf{can}(N')$*, then* N *and* N' *are isomorphic.*

The reverse implication is not required: two isomorphic nets do not have necessarily the same image by canonization. In the sequel, we propose a particular can function defined as the composition of three successive permutations of units, places, and transitions. Units are permuted first, because in a non-trivial NUPN, there are less units than places (in a trivial NUPN, $\mathrm{card}\,(U) \leq \mathrm{card}\,(P) + 1$); transitions are permuted last, because there are usually more transitions than places in a net.

Definition 22 *Let* $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. We assume in this section that* P *(resp.* T*,* U*) is the natural range* $\{1, ..., \mathrm{card}\,(P)\}$ *(resp.* $\{1, ..., \mathrm{card}\,(T)\}$*,* $\{1, ..., \mathrm{card}\,(U)\}$*) and that each place* p *(resp. each transition* t*, each unit* u*) is represented by a unique number noted* $\#p$ *(resp.* $\#t$*,* $\#u$*).*

- *Let* $\pi_{u[N]} : U \to U$ *denote a permutation of the units of* N*; the definition we chose for* $\pi_{u[N]}$ *is given below in Sect. 5.1.*
- *Let* $\pi_{p[N]} : P \to P$ *denote a permutation of the places of* N*; the definition we chose for* $\pi_{p[N]}$ *is given below in Sect. 5.2.*
- *Let* $\pi_{t[N]} : T \to T$ *denote a permutation of the transitions of* N*; the definition we chose for* $\pi_{t[N]}$ *is given below in Sect. 5.3.*
- *Let* N_1 *be the NUPN obtained by permuting the units of* N *with* $\pi_{u[N]}$*.*
- *Let* N_2 *be the NUPN obtained by permuting the places of* N_1 *with* $\pi_{p[N_1]}$*.*
- *Let* N_3 *be the NUPN obtained by permuting the transitions of* N_2 *with* $\pi_{t[N_2]}$*.*

Finally, we define can *to be the function that maps* N *to* N_3*.*

5.1 Unit Sorting

The unit-permutation function $\pi_{u[N]}$ mentioned in Def.22 is defined as follows.

Definition 23 *Let* $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ *be a NUPN. For each unit* u*, one builds a tuple* $m(u) \stackrel{\text{def}}{=} (m_1(u), ..., m_{35}(u))$ *of natural numbers or values of type* $\mathbb{D}$*. The components of this tuple are the following:*

- $m_1(u) \stackrel{\text{def}}{=} \mathsf{depth}\,(u)$.
- $m_2(u) \stackrel{\text{def}}{=} \mathrm{card}\,(\mathsf{subunits}\,(u))$.
- $m_3(u) \stackrel{\text{def}}{=} \mathrm{card}\,(\mathsf{places}\,(u))$.
- $m_4(u) \stackrel{\text{def}}{=} \mathrm{card}\,(\mathsf{places}\,(u) \cap M_0)$.
- $m_5(u) \stackrel{\text{def}}{=} \mathrm{card}\,(\mathsf{subunits}^*(u))$.
- $m_6(u) \stackrel{\text{def}}{=} \mathrm{card}\,(\mathsf{places}^*(u))$.
- $m_7(u) \stackrel{\text{def}}{=} \mathrm{card}\,(\mathsf{places}^*(u) \cap M_0)$.
- $m_8(u) \stackrel{\text{def}}{=} \mathsf{height}\,(u)$.
- $m_9(u) \stackrel{\text{def}}{=} \mathsf{width}\,(u)$.
- $m_{10}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{distance1}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{11}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{distance2}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{12}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{distance3}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{13}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_loops}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{14}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_decreasing_input_transitions}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{15}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_conservative_input_transitions}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{16}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_increasing_input_transitions}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{17}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_decreasing_output_transitions}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{18}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_conservative_output_transitions}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{19}(u) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\, \mathsf{nb_increasing_output_transitions}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{20}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_nb_input_places}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{21}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_nb_output_places}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{22}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_nb_input_places}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{23}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_nb_output_places}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{24}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_input_distance1}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{25}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_output_distance1}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{26}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_input_distance1}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{27}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_output_distance1}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{28}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_input_distance2}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{29}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_output_distance2}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{30}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_input_distance2}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{31}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_output_distance2}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{32}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_input_distance3}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{33}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{pred_output_distance3}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{34}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_input_distance3}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.
- $m_{35}(u) \stackrel{\text{def}}{=} \mathcal{M}(\{\!|\, \mathsf{succ_output_distance3}(p) \mid p \in \mathsf{places}\,(u) \,|\!\})$.

The following components are excluded from $m(u)$*:* $m_{10}(u).\mathsf{card}$, $m_{11}(u).\mathsf{card}$,

..., $m_{18}(u)$.card (because they are all equal to $m_3(u)$); $m_{21}(u)$.card (which is equal to $m_{20}(u)$.card); $m_{23}(u)$.card (which is equal to $m_{22}(u)$.card); $m_{24}(u)$.card, $m_{28}(u)$.card, and $m_{32}(u)$.card (which are all equal to $m_{20}(u)$.sum); $m_{25}(u)$.card, $m_{29}(u)$.card, and $m_{33}(u)$.card (which are all equal to $m_{21}(u)$.sum); $m_{26}(u)$.card, $m_{30}(u)$.card, and $m_{34}(u)$.card (which are all equal to $m_{22}(u)$.sum); and $m_{27}(u)$.card, $m_{31}(u)$.card, and $m_{35}(u)$.card (which are all equal to $m_{23}(u)$.sum).

Definition 24 *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. The function $\pi_{u[N]}$ mentioned in Def.22 is defined to be any permutation $\pi : U \to U$ such that $(\forall u, u' \in U)\ \#u \leq \#u' \Rightarrow m(\pi(u)) \preceq m(\pi(u'))$, where $\preceq$ is the lexicographic order over tuples; thus, π sorts all units u by increasing values of $m(u)$.*

In practice, one can obtain a unique permutation π by extending the tuple $m(u)$ with an extra component $m_{36}(u) \stackrel{\text{def}}{=} \#u$. Doing so guarantees that π is a stable sort, i.e., does not permute indistinguishable units needlessly.

5.2 Place Sorting

The place-permutation function $\pi_{p[N]}$ mentioned in Def.22 is defined as follows.

Definition 25 *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. For each place u, one builds a tuple $n(p) \stackrel{\text{def}}{=} (n_1(p), ..., n_{27}(p))$ of natural numbers or values of type $\mathbb{D}$. The components of this tuple are the following:*

- $n_1(p) \stackrel{\text{def}}{=} \#(\mathsf{unit}\,(p))$.
- $n_2(p) \stackrel{\text{def}}{=} \mathsf{distance1}(p)$.
- $n_3(p) \stackrel{\text{def}}{=} \mathsf{distance2}(p)$.
- $n_4(p) \stackrel{\text{def}}{=} \mathsf{distance3}(p)$.
- $n_5(p) \stackrel{\text{def}}{=} \mathsf{nb_loops}(p)$.
- $n_6(p) \stackrel{\text{def}}{=} \mathsf{nb_decreasing_input_transitions}(p)$.
- $n_7(p) \stackrel{\text{def}}{=} \mathsf{nb_conservative_input_transitions}(p)$.
- $n_8(p) \stackrel{\text{def}}{=} \mathsf{nb_increasing_input_transitions}(p)$.
- $n_9(p) \stackrel{\text{def}}{=} \mathsf{nb_decreasing_output_transitions}(p)$.
- $n_{10}(p) \stackrel{\text{def}}{=} \mathsf{nb_conservative_output_transitions}(p)$.
- $n_{11}(p) \stackrel{\text{def}}{=} \mathsf{nb_increasing_output_transitions}(p)$.
- $n_{12}(p) \stackrel{\text{def}}{=} \mathsf{pred_nb_input_places}(p)$.
- $n_{13}(p) \stackrel{\text{def}}{=} \mathsf{pred_nb_input_places}(p)$.
- $n_{14}(p) \stackrel{\text{def}}{=} \mathsf{succ_nb_input_places}(p)$.
- $n_{15}(p) \stackrel{\text{def}}{=} \mathsf{succ_nb_input_places}(p)$.
- $n_{16}(p) \stackrel{\text{def}}{=} \mathsf{pred_input_distance1}(p)$.
- $n_{17}(p) \stackrel{\text{def}}{=} \mathsf{pred_output_distance1}(p)$.

- $n_{18}(p) \stackrel{\text{def}}{=} \mathsf{succ_input_distance1}(p)$.
- $n_{19}(p) \stackrel{\text{def}}{=} \mathsf{succ_output_distance1}(p)$.
- $n_{20}(p) \stackrel{\text{def}}{=} \mathsf{pred_input_distance2}(p)$.
- $n_{21}(p) \stackrel{\text{def}}{=} \mathsf{pred_output_distance2}(p)$.
- $n_{22}(p) \stackrel{\text{def}}{=} \mathsf{succ_input_distance2}(p)$.
- $n_{23}(p) \stackrel{\text{def}}{=} \mathsf{succ_output_distance2}(p)$.
- $n_{24}(p) \stackrel{\text{def}}{=} \mathsf{pred_input_distance3}(p)$.
- $n_{25}(p) \stackrel{\text{def}}{=} \mathsf{pred_output_distance3}(p)$.
- $n_{26}(p) \stackrel{\text{def}}{=} \mathsf{succ_input_distance3}(p)$.
- $n_{27}(p) \stackrel{\text{def}}{=} \mathsf{succ_output_distance3}(p)$.

The following components are excluded from $n(p)$: $n_{12}(p)$.`card` *and $n_{13}(p)$.*`card` *(because they are equal to $n_6(p) + n_7(p) + n_8(p)$); $n_{14}(p)$.*`card` *and $n_{15}(p)$.*`card` *(because they are equal to $n_9(p)+n_{10}(p)+n_{11}(p)$); $n_{16}(p)$.*`card`*, $n_{20}(p)$.*`card`*, and $n_{24}(p)$.*`card` *(which are all equal to $n_{12}(p)$.*`sum`*); $n_{17}(p)$.*`card`*, $n_{21}(p)$.*`card`*, and $n_{25}(p)$.*`card` *(which are all equal to $n_{13}(p)$.*`sum`*); $n_{18}(p)$.*`card`*, $n_{22}(p)$.*`card`*, and $n_{26}(p)$.*`card` *(which are all equal to $n_{14}(p)$.*`sum`*); and $n_{19}(p)$.*`card`*, $n_{23}(p)$.*`card`*, and $n_{27}(p)$.*`card` *(which are all equal to $n_{15}(p)$.*`sum`*).*

Definition 26 *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. The function $\pi_{p[N]}$ mentioned in Def.22 is defined to be any permutation $\pi : P \to P$ such that $(\forall p, p' \in P)\ \#p \leq \#p' \Rightarrow n(\pi(p)) \preceq n(\pi(p'))$, where $\preceq$ is the lexicographic order over tuples; thus, π sorts all places p by increasing values of $n(p)$.*

In practice, an extra component $n_{28}(p) \stackrel{\text{def}}{=} \#p$ can be added to tuple $n(p)$ to obtain a unique permutation π that is also a stable sort.

5.3 Transition Sorting

The transition-permutation function $\pi_{t[N]}$ of Def.22 is defined as follows.

Definition 27 *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. For each transition t, one builds an tuple $o(t) \stackrel{\text{def}}{=} (o_1(t), o_2(t))$ of natural numbers or values of type $\mathbb{D}$. The components of this tuple are the following:*

- $o_1(t) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\ \#p \mid p \in {}^\bullet t\ |\!\})$, *noticing that* $o_1(t).\texttt{card} = \mathrm{card}\,({}^\bullet t)$.
- $o_2(t) \stackrel{\text{def}}{=} \mathcal{H}(\{\!|\ \#p \mid p \in t^\bullet\ |\!\})$, *noticing that* $o_2(t).\texttt{card} = \mathrm{card}\,(t^\bullet)$.

Definition 28 *Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ be a NUPN. The function $\pi_{t[N]}$ mentioned in Def.22 is defined to be any permutation $\pi : T \to T$ such that $(\forall t, t' \in T)\ \#t \leq \#t' \Rightarrow o(\pi(t)) \preceq o(\pi(t'))$, where $\preceq$ is the lexicographic order over tuples; thus, π sorts all transitions t by increasing values of $o(t)$.*

In practice, an extra component $o_3(t) \stackrel{\text{def}}{=} \#t$ can be added to tuple $o(t)$ to obtain a unique permutation π that is also a stable sort.

5.4 Unique Sorting

As mentioned above, the reverse implication of Def. 21 is not guaranteed: the canonization function can, applied to two isomorphic nets, may return different results. This is especially the case when the nets contain fragments that are locally symmetric (e.g., circular rings, complete subgraphs, etc.), for which several permutations exist. In other cases, however, the reverse implication may hold.

Proposition 3 *Let N and N' be two NUPNs. If each of the three permutations $\pi_{u[N]}$, $\pi_{p[N_1]}$, and $\pi_{t[N_2]}$ mentioned in Def. 22 to compute $\mathsf{can}(N)$ is unique i.e., if lexicographic order $\preceq$ over the tuples m, n, and o (see Def. 24, 26, and 28) defines three total order relations, then N and N' are isomorphic iff $\mathsf{can}(N) = \mathsf{can}(N')$.*

Proof. Let $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$ and $N' = (P', T', F', M'_0, U', u'_0, \sqsubseteq', \mathsf{unit}')$ be two isomorphic NUPNs. Let (π_p, π_t, π_u) be the three bijections of Def. 7 relating N and N'. First: for each $u \in U$ and $i \in \{1, ..., 35\}$, one can prove that $m_i(u) = m'_i(\pi_u(u))$ by combining Def 7 and Def. 23; if $\pi_u[N]$ is unique, then $\pi_u[N']$ is unique too, and the two NUPNs N_1 and N'_1 obtained from N and N' by applying $\pi_u[N]$ and $\pi_u[N']$, respectively, are isomorphic and related by the three bijections (π_p, π_t, id), where id is the identity function on $\mathbb{N}$. Second: for each $p \in P$ and $i \in \{1, ..., 27\}$, one can prove that $n_i(p) = n'_i(\pi_p(p))$ by combining Def 7 and Def. 25; similarly, the two NUPNs N_2 and N'_2 obtained from N_1 and N'_1 by applying $\pi_p[N_1]$ and $\pi_p[N'_1]$, respectively, are isomorphic and related by the three bijections (id, π_t, id). Third: for each $t \in T$ and $i \in \{1, 2\}$, one can prove that $o_i(t) = o'_i(\pi_t(t))$ by combining Def 7 and Def. 27; similarly, the two NUPNs N_3 and N'_3 obtained from N_2 and N'_2 by applying $\pi_t[N_2]$ and $\pi_t[N'_2]$, respectively, are related by the three bijections (id, id, id). Therefore, $\mathsf{can}(N) = \mathsf{can}(N')$.

6 Implementation

We implemented these ideas in a software tool chain that combines: (i) tools specifically developed for the purpose of the present article; (ii) tools already developed at Inria Grenoble, which we extended for the same purpose; and (iii) third-party tools that we reused without modification.

Our tool chain takes as input a collection of nets and determines which ones are isomorphic. It accepts Petri nets and NUPNs given in the standard PNML format [11] or in the "`.nupn`" format[5] (conversion from PNML to "`.nupn`" format can be done using the Pnml2nupn translator[6]). Our tool chain implements all

[5] `https://cadp.inria.fr/man/nupn.html`
[6] `http://pnml.lip6.fr/pnml2nupn`

the approaches presented in Sections 3–5, ordered by increasing complexities; the tool chain stops as soon has all the duplicates in a collection have been found. All steps of the tool chain have been carefully validated using miscellaneous techniques that cannot be presented here by lack of space.

6.1 File Deduplication

The first and simplest way to search for duplicates in a collection of nets is to search for identical files. Among the many tools for this purpose, we selected FDUPES[7] which is a fast, reliable Unix command-line tool; the alternative (seemingly faster) tool JDUPES[8] is also a good option.

Obviously, this approach is very limited. For instance, inserting an extra space in a PNML file may prevent two isomorphic nets from being detected this way. There is thus a clear trade-off between the flexibility of a net format and the ability to detect duplicates using mere file comparison. In this respect, the "`.nupn`" format is preferable to PNML because it is more stringent: places, transitions, and units are named using natural numbers instead of alphanumeric identifiers; lexical tokens must be separated using exactly one space; blank lines are forbidden, as well as trailing spaces before end of lines, etc. For this reason, our toolchain employs the "`.nupn`" format rather than PNML. In practice, translation from PNML to "`.nupn`", followed by an invocation of FDUPES, is often sufficient to detect duplicates that do not involve permutations.

6.2 Pre-canonization

Even if the "`.nupn`" format is more stringent than PNML, it still offers a degree of flexibility that allows a given net to be expressed under different forms, even in absence of any permutation of places, transitions, or units. To address this problem, the NUPN_INFO tool[9] has been extended with a "`-precanonical-nupn`" option that takes as input a net in "`.nupn`" format and produces as output the same net in which: (i) all places (resp. transitions and units) are renumbered starting from zero; (ii) all lists of places (resp. transitions and units) are sorted by increasing numbers; (iii) all labels of places (resp. transitions and units) are deleted; and (iv) all pragmas are removed. After putting all nets under such pre-canonical form, FDUPES is invoked to detect duplicate files.

6.3 Signatures

We extended the CÆSAR.BDD tool[10] with a "`-signature`" option that computes the signature (as defined in Sect. 4) of a net given in "`.nupn`" format. Written

[7] https://github.com/adrianlopezroche/fdupes
[8] https://github.com/jbruchon/jdupes
[9] https://cadp.inria.fr/man/nupn_info.html
[10] https://cadp.inria.fr/man/caesar.bdd.html

in C, the computation is fast (0.12 second per net on average) and always succeeds. A shell script identifies classes of nets with the same signatures.

To check the correctness of signatures, we developed a Python script that, given a NUPN $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$, generates random permutations $\pi_p : P \to P$, $\pi_t : T \to T$, and $\pi_u : U \to U$. Since the "`.nupn`" format requires that places of the same unit have contiguous numbers in a range, each generated function π_p only permutes places within their units, i.e., for each $p \in P$, $\mathsf{unit}\,(\pi_p(p)) = \mathsf{unit}\,(p)$, without loss of generality. The NUPN_INFO tool (with options "`-place-permute`", "`-transition-permute`", and "`-unit-permute`") is then invoked on N to produce as output a permuted NUPN; when applying π_p, π_t and π_u to permute P, T, and U, NUPN_INFO updates F, M_0, u_0, $\sqsubseteq$, and unit to enforce the constraints of Def. 7. Finally, we validated Prop. 2 by checking, on tenths of thousands of NUPNs and tenths of millions of random permutations, that the signatures of the original and permuted nets are identical.

6.4 Canonization

We further extended the CÆSAR.BDD tool with three new options ("`-unit-order`", "`-place-order`", and "`-transition-order`") that compute, for a net given in "`.nupn`" format, all the tuples $m(u)$, $n(p)$, and $o(t)$ defined in Sect. 5. CÆSAR.BDD then invokes the Unix "`sort`" command to sort these tuples lexicographically and performs, for the "`-unit-order`" option only, further calculations that may help distinguishing units having the same $m(u)$ value. An Awk script is then invoked to transform these results into permutations of units, places, or transitions, and report whether such permutations are unique or not. The input net and the three permutations are then given to NUPN_INFO, which produces as output a canonized net. A new option "`-canonical-nupn`" that automates all these steps, including the three invocations to CÆSAR.BDD, was added to NUPN_INFO. Written in C and Awk, canonization is generally fast (8 seconds per net on average) but took, in the two worst cases, 8 minutes and 90 minutes on two nets of the Model Checking Contest having the largest number of places (143,908 and 537,708 places respectively). Finally, FDUPES is invoked to detect file duplicates in the set of canonized nets.

To increase confidence in our implementation of canonization, we checked on each net that canonization is idempotent, meaning that two successive invocations of NUPN_INFO with its "`-canonical-nupn`" option produce the same output as one single invocation. Also, for tenths of thousands of NUPNs $N = (P, T, F, M_0, U, u_0, \sqsubseteq, \mathsf{unit})$, we generated tenths of millions of random permutations π_p, π_t, and π_u, and verified that: (i) for each $i \in \{1, ..., 35\}$, for each $u \in U$, $m_i(u) = m'_i(\pi_u(u))$, where m' is the tuple of Def. 23 computed on the net obtained from N by applying π_u, π_p, and π_t; (ii) for each $i \in \{1, ..., 27\}$, for each $p \in P$, $n_i(p) = n'_i(\pi_p(p))$, where n' is the tuple of Def. 25 computed on the net obtained from N by applying π_p and π_t; and (iii) for each $i \in \{1, 2\}$, for each $t \in T$, $o_i(t) = o'_i(\pi_t(t))$, where n' is the tuple of Def. 27 computed on the

net obtained from N by applying π_t; the nets are permuted using NUPN_INFO and the tuples m, m', n, n', o, and o' are computed using CÆSAR.BDD.

6.5 Graph Isomorphism

As mentioned in Sect. 3.2, we selected the TRACES software, which is deemed to be a reference tool for graph isomorphism. We developed a Python script that converts a net in "`.nupn`" format to a colored graph (see Sect. 3), and then invokes TRACES to put this graph under canonical form. To process a collection, our script is first invoked on each net of the collection; FDUPES is then used to detect duplicate files among the canonized graphs. This detection may be incomplete since TRACES sometimes aborts or times out on large graphs.

We validated our implementation as follows: when two nets N and N' have been found isomorphic via their associated graphs $\mathcal{G}_N$ and $\mathcal{G}_{N'}$, TRACES produces two bijections π and π' that map the vertices of $\mathcal{G}_N$ and $\mathcal{G}_{N'}$ to the vertices of their respective canonical graphs (which are identical). Let $\pi_v \stackrel{\text{def}}{=} \pi'^{-1} \circ \pi$; from π_v, we compute three bijections π_p, π_t, and π_u as explained in the proof of Prop. 1, easily adapted to our optimized translation mentioned in Sect. 3.2; we finally check that $(\pi_t \circ \pi_p \circ \pi_u)(N) = N'$. We also cross-checked the results of the net-canonization approach with those of the graph-isomorphism approach by validating Def. 21, i.e., if two NUPNs N and N' satisfy $\mathsf{can}(N) = \mathsf{can}(N')$, then their associated graphs $\mathcal{G}_N$ and $\mathcal{G}_{N'}$ should be found isomorphic by TRACES (when this tool can handle them).

6.6 Tool Chain

The five approaches of Sects. 6.1 to 6.5 are successively applied, in this order. Each approach only considers the problems not solved by prior approaches, and one stops as soon as all problems have been solved. Figure 1 depicts the application of our tool chain to a collection of 10 nets named from 'a' to 'j'. Some approaches (identical files, pre-canonization, canonization, and graph isomorphism) detect certain nets that are isomorphic: we represent this information using solid boxes that gather isomorphic nets. Other approaches (signatures, canonization when the assumptions of Prop. 3 hold, and graph isomorphism) detect certain nets that are not isomorphic: we represent this information by partitioning the collection into dashed boxes (using the partition-refinement idea), such that all nets belonging to distinct dashed boxes are pairwise non-isomorphic.

7 Experiments

We assessed our tool chain on four collections of nets listed in Table 1 below:

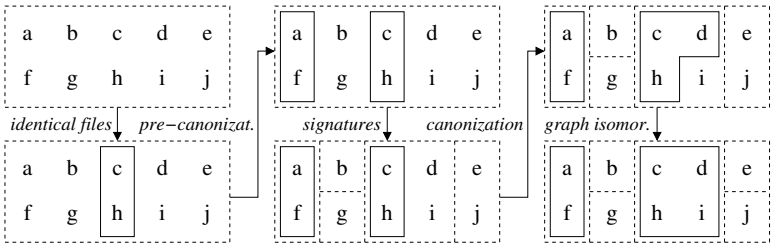

Figure 1: Pipelined application of our 5 approaches to a collection of 10 nets

	collection 1		collection 2		collection 3		collection 4	
	avg.	max.	avg.	max.	avg.	max.	avg.	max.
#places	15.4	200	2,801.5	537,708	345.8	131,216	740.8	131,216
#trans.	11.8	51	10,798	1,070,836	7,998.1	16,967,720	15,645	16,967,720
#arcs	34.2	400	83,384	25,615,632	71,217.9	146,528,584	113,102.9	146,528,584
#units	—	—	1,970	537,709	123.4	78,644	270.4	78,644
height	—	—	15.4	2,891	4.3	2,891	6.3	2,891
width	—	—	1,959.1	537,708	117.6	78,643	259.9	78,643

Table 1: Numerical statistics about the four net collections used in experiments

- *Collection 1*: 244 Petri nets (without NUPN structure) made available by the University of Zielona Góra[11];
- *Collection 2*: 1387 Petri nets obtained by taking all the (non-colored) models used for the 2022 edition of the Model Checking Contest[12]; 44% of these nets have initial places with more than one tokens and/or arcs with multiplicity greater than one, whereas 50% of these nets are non-trivial unit-safe NUPNs;
- *Collection 3*: 16,200 unit-safe NUPNs from diverse origins, containing few duplicates, gathered at Inria Grenoble to be used in scientific experiments;
- *Collection 4*: 241,657 unit-safe NUPNs (135 GB of disk space) produced at Inria Grenoble by removing, using Fdupes, all identical files from a larger set of 840,838 NUPNs that was obtained after extending collection 3 with additional NUPNs and applying numerous permutations to all these nets; therefore, collection 4 contains many duplicates (i.e., isomorphic NUPNs).

As mentioned above, our experiments were performed on the Grid'5000 testbed, each server having an Intel Xeon Gold 5220 (2.2 GHz) processor, 96 GB RAM, and running Linux Debian 11 with a shared NFS filesystem. To reduce the variability in results, each server was executing only one experiment at a time.

[11] `http://www.hippo.iie.uz.zgora.pl` (retrieved on January 23, 2023)
[12] `http://mcc.lip6.fr`

	collection 1			collection 2			collection 3			collection 4		
	dupl. (%)	uniq. (%)	unkn. (%)	dupl. (%)	uniq. (%)	unkn. (%)	dupl. (%)	uniq. (%)	unkn. (%)	dupl. (%)	uniq. (%)	unkn. (%)
identical files	4.10	0.00	95.90	0.00	0.00	100.0	0.00	0.00	100.0	0.00	0.00	100.0
pre-canonizat.	4.10	0.00	95.90	—	—	—	0.17	0.00	99.83	22.35	0.00	77.65
signatures	4.10	86.88	9.02	0.00	98.56	1.44	0.17	92.87	6.96	22.35	0.12	77.53
canonization	5.74	91.39	2.87	0.58	98.84	0.58	2.26	94.87	2.87	79.44	4.74	15.82
graph isomor.	6.97	93.03	0.00	0.58	99.42	0.00	2.79	97.20	0.01	90.05	9.01	0.94

Table 2: Results obtained by our tool chain on the four net collections

The results of applying our tool chain to these four collections are displayed in Table 2 below. After each step of the tool chain, we give three figures: *dupl.* is the percentage of nets that can be removed, since they were found isomorphic to other nets that will be kept in the collection; *uniq.* is the percentage of nets found to be unique in the collection after removing all duplicates; *unkn.* is the remaining percentage of nets whose status is not yet determined[13]. Notice that the values of *dupl.* and *uniq.* in Table 2 increase from top to bottom, as each line builds upon the cumulated successes reported in upper lines; thus, the contribution of each approach can be obtained as the difference between the percentage given on the corresponding line and the percentage given on the previous line.

The main finding is that our tool chain was conclusive for 99%–100% of each collection. More detailed remarks can be made:

- The simple application of FDUPES detected 10 duplicate files in collection 1.
- Pre-canonization detected 54,018 duplicates in collection 4; pre-canonization was not applied to collection 2 in order to preserve those "`.nupn`"-format pragmas giving information about multiple arcs and multiple initial tokens.
- Signatures massively identified unique nets in collections 1–3, but had no impact on collection 4, in which each net has at least one duplicate.
- Canonization was effective, both in identifying duplicate and unique nets.
- Use of graph isomorphism (with a one-hour timeout) clarified the case of most nets whose status remained unknown after canonization.
- Interestingly, our tool chain detected 17 duplicates in collection 1 and eight duplicates in collection 2; for the latter, one duplicate is certain (the corresponding nets are one-safe) and seven are uncertain, but very likely.

8 Conclusion

Starting from the concrete problem of finding duplicate models in large collections of Petri nets or NUPNs, we devised three approaches for the detection of isomorphic nets: reduction to graph isomorphism, net signatures (an over-

[13] Our success statistics could be slightly improved by considering that, among each set of $n > 0$ nets with undetermined status, there is at least one unique net.

approximation), and net canonization (an under-approximation).

These approaches, which draw on the careful examination of thousands of concrete benchmarks, have been fully implemented in an efficient tool chain, the successive steps of which are ordered by increasing complexity. To process large collections of nets, the calculations can easily be distributed on computer clusters or grids, as most steps deal with individual nets. Only the detection of identical files is not easy to parallelize, but did not cause bottlenecks, since the tool we selected is fast enough.

We assessed our tool chain on four collections ranging from 244 to 241,657 nets and containing either few or many duplicates. We observed a success rate of 99%–100% in the detection of isomorphic nets.

The present work could be pursued in, at least, two directions: (i) one could try shortening the component lists used in signatures and canonization to retain only those components that are most effective in practice; (ii) one could extend the proposed approaches to support wider classes of nets, such as non-safe Petri nets (these are currently handled using over-approximations) and colored nets.

Acknowledgements

Our experiments have been performed using the French Grid'5000 testbed.

References

[1] Eric Badouel and Philippe Darondeau. Theory of Regions. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer, September 1996.

[2] Eike Best and Raymond R. Devillers. Synthesis and reengineering of persistent systems. *Acta Informatica*, 52:35–60, February 2015.

[3] Pierre Bouvier and Hubert Garavel. Efficient Algorithms for Three Reachability Problems in Safe Petri Nets. In Didier Buchs and Josep Carmona, editors, *Proceedings of the 42nd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS'21), Paris, France*, volume 12734 of *Lecture Notes in Computer Science*, pages 339–359. Springer, June 2021.

[4] Jörg Desel and Wolfgang Reisig. The Synthesis Problem of Petri Nets. *Acta Informatica*, 33:297–315, June 1996.

[5] Raymond R. Devillers. Articulations and Products of Transition Systems and their Applications to Petri Net Synthesis. *CoRR*, abs/2111.00202, 2021.

[6] Raymond R. Devillers and Uli Schlachter. Factorisation of Petri Net Solvable Transition Systems. In Victor Khomenko and Olivier H. Roux, editors, *Proceedings of the 39st International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS'18), Bratislava, Slovakia*, volume 10877 of *Lecture Notes in Computer Science*, pages 82–98. Springer, June 2018.

[7] Hubert Garavel. Nested-Unit Petri Nets. *Journal of Logical and Algebraic Methods in Programming*, 104:60–85, April 2019.

[8] Roberto Gorrieri. Team equivalences for finite-state machines with silent moves. *Information and Computation*, 275:104603, 2020.

[9] Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Communications of the ACM*, 63(11):128–134, 2020.

[10] Martin Hesketh and Maciej Koutny. An Axiomatisation of Duplication Equivalence in the Petri Box Calculus. In Jörg Desel and Manuel Silva Suárez, editors, *Proceedings of the 19th International Conference on Application and Theory of Petri Nets (ICATPN'98), Lisbon, Portugal*, volume 1420 of *Lecture Notes in Computer Science*, pages 165–184. Springer, June 1998.

[11] ISO/IEC. High-level Petri Nets – Part 2: Transfer Format. International Standard 15909-2:2011, International Organization for Standardization – Information Technology – Systems and Software Engineering, Geneva, 2011.

[12] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, September 2014.

[13] Tadao Murata. Petri Nets: Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.