Equivalence of Data Petri Nets with Arithmetic

Marco Montali and Sarah Winkler(✉)

Free University of Bozen-Bolzano, Bolzano, Italy
{montali,winkler}@inf.unibz.it

Abstract. Data Petri nets (DPNs) with arithmetic have gained popularity as a model for data-aware processes, thanks to their ability to balance simplicity with expressiveness and because they can be automatically discovered from event logs. While model checking techniques for DPNs have been studied, there are analysis tasks highly relevant for BPM that are beyond these methods. We focus here on process equivalence and process refinement with respect to language and configuration spaces; such comparisons are important in the context of process repair and discovery. To solve these tasks, we propose an approach for bounded DPNs based on *constraint graphs*, which are faithful abstractions of the reachable state space. Though the considered verification tasks are undecidable in general, we show that our method is a decision procedure for large classes of DPNs relevant in practice.

Keywords: data-aware processes · data Petri nets · process equivalence · process refinement

1 Introduction

Within the growing area of data-aware processes, Data Petri nets (DPNs) with arithmetic data have recently gained increasing popularity thanks to their ability to balance simplicity with expressiveness. DPNs can also be mined automatically [5,17], but automatic mining techniques typically come without any correctness guarantees. However, the complex interplay between the control structure and data makes it hard to check whether DPNs satisfy properties of interest; indeed, all non-trivial verification tasks are undecidable. While linear- and branching-time model checking procedures for DPNs were developed [11–13,18], many analysis tasks relevant in BPM go beyond these techniques. Here we focus on checking equivalence and refinement of processes, which is an important task in many contexts [1,6]: to match an organization-specific model to a reference model, to relate an automatically mined model to a normative one by domain experts, or to compare a refined version of a process model with the original one.

This work was partially funded by the UNIBZ project ADAPTERS, and the PRIN MIUR project PINPOINT Prot. 2020FNEB27.

J. De Weerdt and L. Pufahl (Eds.): BPM 2023 Workshops, LNBIP 492, pp. 409–421, 2024.
https://doi.org/10.1007/978-3-031-50974-2_31

Example 1. The DPN in Fig. 1 models a management process for road fines by the Italian police [20], where assignments with right-hand side *?* indicate a non-deterministic write operation.

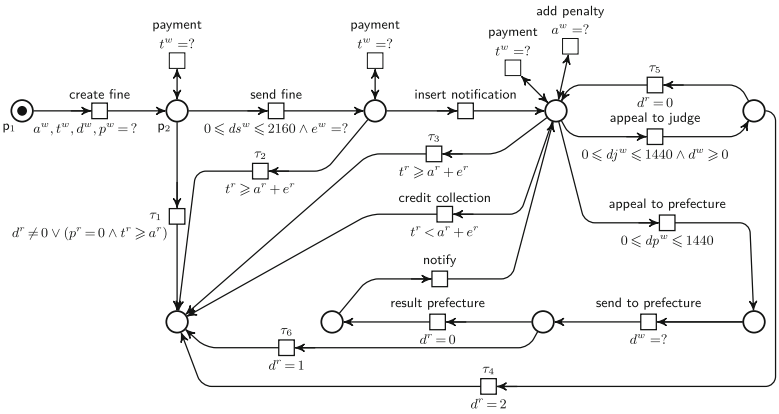


Fig. 1. Data Petri net for road fine management process

This normative process model was designed by domain experts, but other versions of this process were discovered by automatic techniques [19]. This raises a number of questions that are important in the context of process discovery and repair: Do the DPNs admit the same set of configurations and data values? Are all process runs of one model also possible in the other? Do the possible data values in a final state coincide?

To answer such questions, we consider in this paper different notions of behavioural refinement and equivalence for DPNs, comparing markings, configurations, or language, and we propose techniques to check them. Since even state reachability is undecidable for DPNs [12, Rem. 2.6], it does not come as a surpise that these verification tasks are in general undecidable as well.

In this paper, we thus impose two restrictions on DPNs to make verification decidable: (1) To tame the control flow perspective, we assume that the Petri nets underlying DPNs are *bounded*. (2) To tame the data perspective, we assume that the DPNs have a *finite constraint graph*. Constraint graphs (CGs) are symbolic abstractions of the reachable state space that are used for data-aware soundness and model checking of DPNs [10–13]. In general, CGs are infinite. However, it was shown that *finite* constraint graphs can in fact be computed for a wide range of DPNs from the literature. These include DPNs where all constraints are variable-to-variable/constant comparisons, as produced by automatic guard discovery techniques [5,17], or *bounded lookback* DPNs whose behavior depends only on a bounded amount of information from the process run [4,12]. It was shown that these classes comprise almost all DPNs in the literature [13], also e.g. the process of Example 1. Notably, we do *not* assume that DPNs are acyclic.

The contributions of this paper can be summarized as follows: (1) We show how natural notions of equivalence and refinement of DPNs with respect to markings, configurations, and language can be checked based on constraint graphs. (2) Our technique is a decision procedure for bounded DPNs where finite constraint graphs exist, which proves decidability of process refinement/equivalence for such DPNs. These include DPNs where all guards are variable-to-variable/-constant comparisons, and DPNs with bounded lookback. (3) If equivalence or refinement does not hold, counterexamples that distinguish the two processes can be computed by our approach.

Related work. For process models without data, a variety of comparison techniques were developed. An early first approach for process equivalence was presented in [1]. A basic taxonomy of similarity measures was proposed in [6], distinguishing similarity based on either *element labels*, *structure*, or *behaviour*. For the first kind, schema and ontology matching techniques are used [8]. For structural similarity measures, graph matching algorithms were studied [7]. Our approach falls within the class of behavioural similarity, and to the best of our knowledge, no respective approaches exist to compare data-aware processes. However, while most works on process comparison are quantitative (i.e., they quantify process similarity with respect to some measure [16,21], this paper is purely qualitative, in the sense that our techniques check process equivalence or refinement, but the difference between models is not quantified.

2 Background

In this section we summarize some background on constraints, DPNs and data-aware dynamic systems as process models, as well as constraint graphs.

We assume a set of *process variables* V, each of which is associated with a sort from the set $\Sigma = \{\texttt{int}, \texttt{rat}\}$ with associated domains integers $\mathcal{D}(\texttt{int}) = \mathbb{Z}$ and rationals $\mathcal{D}(\texttt{rat}) = \mathbb{Q}$. For instance, in Example 1 the set of process variables is $V = \{a, d, dj, dp, ds, p, t\}$, where a and t are of sort `rat` and the others of sort `int`. For $\sigma \in \Sigma$, V_σ denotes the subset of variables in V of type σ. To manipulate variables, we consider linear arithmetic expressions c, called *constraints*:

$$c := n \geqslant n \mid n \neq n \mid n = n \mid r \geqslant r \mid r > r \mid r \neq r \mid r = r \mid c \wedge c$$
$$n := v_i \mid k \mid n + n \mid -n \qquad\qquad r := v_r \mid q \mid r + r \mid -r$$

where $v_i \in V_{\texttt{int}}$, $v_r \in V_{\texttt{rat}}$, $k \in \mathbb{Z}$, and $q \in \mathbb{Q}$. These expressions will be used to capture conditions on the values of variables that are read and written during the execution of process activities. The set of constraints over a set of variables V is denoted $\mathcal{C}(V)$. We will also consider first-order formulas that have constraints as atoms. Given the definition of constraints, such formulas are in the theory of linear arithmetic, which is decidable [2]. Moreover, *quantifier elimination* can produce a quantifier-free, equivalent for any formula of the form $\exists x.\varphi$, cf. [2]. We denote logical equivalence by $\equiv$, and logical entailment by $\models$.

Data Petri nets. We adopt the standard definition of Data Petri Nets (DPNs) [19,20]. We consider two disjoint, marked copies of the set of process variables V, denoted $V^r = \{v^r \mid v \in V\}$ and $V^w = \{v^w \mid v \in V\}$, called the *read* and *write* variables. They will refer to variable values before and after a transition, respectively. We also write $\overline{V}$ for a vector that orders V in an arbitrary, fixed way, and $\overline{V}^r$ and $\overline{V}^w$ for vectors ordering V^r and V^w in the same way.

Definition 1. *A* data Petri net *(DPN) is a tuple* $\mathcal{N} = \langle P, T, F, \ell, \mathcal{A}, V, guard \rangle$*, where (1)* $\langle P, T, F, \ell \rangle$ *is a labelled Petri net with non-empty, disjoint sets of places* P *and transitions* T*, a flow relation* $F \colon (P \times T) \cup (T \times P) \mapsto \mathbb{N}$ *and a labelling function* $\ell \colon T \mapsto \mathcal{A} \cup \{\tau\}$*, where* $\mathcal{A}$ *is a finite set of activity labels and* τ *is a special symbol for silent transitions; (2)* V *is a set of process variables with a sort in* Σ*; and (3)* $guard \colon T \mapsto \mathcal{C}(V^r \cup V^w)$ *is a guard mapping.*

Example 2. The process in Fig. 2(a) is a DPN modelling a simple auction process. It maintains the set of variables $V = \{o, t\}$ of sort `rat`, where o holds the last offer issued by a bidder, and t is a timer. The action init sets the timer t to a positive value and the offer o to 0; while the timer did not expire, it can be decreased (action timer), or bids can be issued, increasing the current offer (bid); the item can be sold if the timer expired and the offer is positive (hammer). We denote this DPN, consisting of all actions drawn in black in Fig. 2, by $\mathcal{N}$. Moreover, we consider a variant of this DPN with an additional reset action that restarts the process if the offer is 0 (drawn in blue), and call this DPN $\mathcal{N}_{\mathsf{reset}}$.

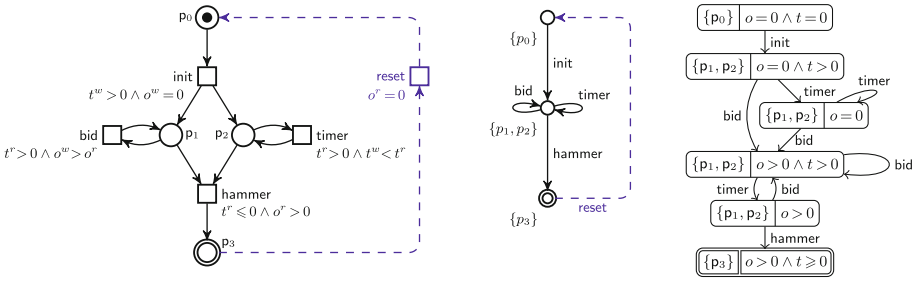


Fig. 2. A DPN with its DDSA and constraint graph.

Also the process shown in Example 1 is a DPN.

The variables read and written by a transition t are denoted by $read(t) = \{v \mid v^r$ occurs in $guard(t)\}$ and $write(t) = \{v \mid v^w$ occurs in $guard(t)\}$. For instance, for t the activity bid in Fig. 2, $write(t) = \{o\}$ and $read(t) = \{o, t\}$. An assignment with domain V is called a *state variable assignment*, to distinguish it from a *transition variable assignment* β that assigns values to the set of variables $V^r \cup V^w$. All assignments are supposed to map variables to elements of their domain.

A *configuration* in a DPN $\mathcal{N}$ is a pair (M, α) given by a marking $M\colon P \mapsto \mathbb{N}$ for the underlying Petri net, together with a state variable assignment α. A configuration thus simultaneously accounts for the control flow progress and for the current values of all variables in V, as specified by α. For instance, $(\{\mathsf{p}_0\}, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right])$ is a configuration of the DPNs of Example 2.

Definition 2 (Transition firing). *A transition $t \in T$ is* enabled *in (M, α) if a transition variable assignment β exists such that:*

(i) $\beta(v^r) = \alpha(v)$ for every $v \in read(t)$, i.e., β is as α for read variables;
(ii) $\beta \models guard(t)$, i.e., β satisfies the guard; and
(iii) $M(p) \geqslant F(p,t)$ for every p so that $F(p,t) \geqslant 0$.

An enabled transition may fire*, producing a new configuration (M', α'), s.t. $M'(p) = M(p) - F(p,t) + F(t,p)$ for every $p \in P$, and $\alpha'(v) = \beta(v^w)$ for every $v \in write(t)$, and $\alpha'(v) = \alpha(v)$ for every $v \notin write(t)$. A pair (t, β) as above is called (valid)* transition firing*, and we denote its firing by $(M, \alpha) \xrightarrow{(t,\beta)} (M', \alpha')$.*

Thus, a guard simultaneously expresses a condition on read variables, and an update on written ones: e.g., bid in Fig. 2 requires the current value of t to be positive and non-deterministically sets o to a new value that exceeds the current.

Given $\mathcal{N}$, we fix one configuration (M_I, α_0) as *initial*, where M_I is the initial marking of the underlying Petri net and α_0 is a state variable assignment that specifies the initial values of all variables in V. The final marking is denoted M_F. For instance, $\mathcal{N}$ in Example 2 admits a transition firing $(\{\mathsf{p}_0\}, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right]) \xrightarrow{\mathsf{init}} (\{\mathsf{p}_1, \mathsf{p}_2\}, \left[\begin{smallmatrix} t=1 \\ o=0 \end{smallmatrix}\right])$ from its initial state; and $\{\mathsf{p}_3\}$ is the final marking.

A state (M', α') is *reachable* in a DPN if it is reached by a transition sequence from the initial state $(M_I, \alpha_0) \xrightarrow{(t_1,\beta_1)} \dots \xrightarrow{(t_n,\beta_n)} (M', \alpha')$. Such a sequence is also written as $(M_I, \alpha_0) \to^* (M', \alpha')$. We denote by $\mathcal{M}ark(\mathcal{N})$ the set of all such M', and by $\mathcal{C}onf(\mathcal{N})$ the set of all such (M', α), i.e., the sets of reachable markings and configurations. A transition sequence as above is a *process run* if $M' = M_F$. In this paper, we will assume that DPNs are *bounded*, i.e., that the number of tokens in reachable markings is upper-bounded by some $k \in \mathbb{N}$.

Data-aware Dynamic Systems with Arithmetic (DDSAs) are a simpler, equivalent model [9,10] that we will use for analysis tasks.

Definition 3. *A* DDSA $\mathcal{B} = \langle B, b_I, \mathcal{A}, T, B_F, V, \alpha_I, guard \rangle$ *is a labeled transition system where (1) B is a finite set of* control states*, with $b_I \in B$ the initial one; (2) $\mathcal{A}$ is a set of* actions*; (3) $T \subseteq B \times \mathcal{A} \times B$ is a* transition relation*; (4) $B_F \subseteq B$ are* final states*; (5) V is the set of* process variables*; (6) α_I the* initial variable assignment*; and (7) guard$\colon \mathcal{A} \mapsto \mathcal{C}(V^r \cup V^w)$ specifies* executability constraints *for actions over variables $V^r \cup V^w$.*

Every bounded DPN $\mathcal{N}$ can be equivalently expressed as a DDSA $\mathcal{B}$ over the same set of process variables V, by unfolding all possible markings (see [13] for

details). The set of control states of $\mathcal{B}$ coincides thus with the set of markings of $\mathcal{N}$. Figure 2(b) shows a DDSA which corresponds to the DPN in Fig. 2(a). The action guards are the same as in the DPN, but have been omitted for readability.

If a control state $b \in B$ admits a transition to b' via action a, i.e., $(b, a, b') \in T$, this is denoted by $b \xrightarrow{a} b'$. A *configuration* of $\mathcal{B}$ is a pair (b, α) where $b \in B$ and α is a state variable assignment, and (b_I, α_I) is the initial one. As defined next, an action a transforms a configuration (b, α) into a new configuration (b', α') by updating the assignment α according to the action guard, exactly as in DPNs:

Definition 4. *A DDSA* $\mathcal{B} = \langle B, b_I, \mathcal{A}, T, B_F, V, \alpha_I, guard \rangle$ admits a step *from configuration* (b, α) *to* (b', α') *via action* a*, denoted* $(b, \alpha) \xrightarrow{a} (b', \alpha')$*, if* $b \xrightarrow{a} b'$*,* $\alpha'(v) = \alpha(v)$ *for all* $v \in V \setminus write(a)$*, and the transition assignment* β *given by* $\beta(v^r) = \alpha(v)$ *and* $\beta(v^w) = \alpha'(v)$ *for all* $v \in V$*, satisfies* $\beta \models guard(a)$*.*

A *run* ρ of a DDSA $\mathcal{B}$ is a sequence of steps $\rho\colon (b_I, \alpha_I) = (b_0, \alpha_0) \xrightarrow{a_1} (b_1, \alpha_1) \xrightarrow{a_2} \dots \xrightarrow{a_n} (b_n, \alpha_n)$, and it is *final* if $b_n \in B_F$. We call the *abstraction* of a run the respective transition sequence $b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} b_n$. For instance, for the DDSA in Fig. 2(b), the following is a run ending in a final state (note that each state corresponds to a marking of the DPN):

$$(\{\mathsf{p}_0\}, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right]) \xrightarrow{\mathsf{init}} (\{\mathsf{p}_1, \mathsf{p}_2\}, \left[\begin{smallmatrix} t=1 \\ o=0 \end{smallmatrix}\right]) \xrightarrow{\mathsf{bid}} (\{\mathsf{p}_1, \mathsf{p}_2\}, \left[\begin{smallmatrix} t=1 \\ o=5 \end{smallmatrix}\right]) \xrightarrow{\mathsf{timer}} (\{\mathsf{p}_1, \mathsf{p}_2\}, \left[\begin{smallmatrix} t=0 \\ o=5 \end{smallmatrix}\right]) \xrightarrow{\mathsf{hammer}} (\{\mathsf{p}_3\}, \left[\begin{smallmatrix} t=0 \\ o=5 \end{smallmatrix}\right])$$

The number of runs and configurations of a DPN or DDSA are typically infinite, due to the infinite number of possible valuations. For analysis tasks, we thus resort to the following abstraction:

Constraint graphs (CGs) are an abstraction of the reachable state space that was introduced for soundness checking [10,13]. The key idea is that formulas are used to represent sets of configurations.

Let $\mathcal{B} = \langle B, b_I, \mathcal{A}, \Delta, B_F, V, \alpha_I, guard \rangle$ be a DDSA. The *transition formula* Δ_a of action a is given by $\Delta_a(\overline{V}^r, \overline{V}^w) = guard(a) \wedge \bigwedge_{v \notin write(a)} v^w = v^r$. This formula simply expresses conditions on variables *before and after* executing the action: $guard(a)$ must hold, and the values of all variables that are not written are copied. E.g., for action bid in Fig. 2(b), $write(\mathsf{bid}) = \{o\}$, so $\Delta_{\mathsf{bid}} = (t^r > 0) \wedge (o^w > o^r) \wedge (t^w = t^r)$. Next, we define an *update* operation, to express how a set of configurations, captured by formula φ, changes when executing an action.

Definition 5. *For a formula* φ *with free variables* V *and action* a*,* $update(\varphi, a) = \exists \overline{U}.\varphi[\overline{U}/\overline{V}] \wedge \Delta_a[\overline{U}/\overline{V}^r, \overline{V}/\overline{V}^w]$*, where* U *is a set of variables that has the same cardinality as* V *and is disjoint from all variables in* φ*.*

Here, $\varphi[\overline{U}/\overline{V}]$ is the result of replacing variables $\overline{V}$ in φ by $\overline{U}$, and similar for Δ_a. For instance, if $\overline{V} = (o, t)$ we can take the renamed variables $\overline{U} = (o', t')$; for $\varphi = (t > 0) \wedge (o = 0)$ we then get $update(\varphi, \mathsf{bid}) = \exists o'\, t'.(t' > 0) \wedge (o' = 0) \wedge (o > o') \wedge (t = t')$, which is equivalent to $(t > 0) \wedge (o > 0)$. This reflects that, if in the process of Fig. 2, $t > 0$ and $o = 0$ hold, and bid is executed, then afterwards we still have $t > 0$ but also o is positive. Below, let $c_{\alpha_I} := \bigwedge_{v \in V} v = \alpha_I(v)$.

Definition 6. *The* constraint graph $\mathrm{CG}_{\mathcal{B}}$ *of* $\mathcal{B}$ *is a quadruple* $\langle S, s_0, \gamma, S_F \rangle$ *where the set of nodes* S *consists of tuples* (b, φ) *for* $b \in B$ *and a formula* φ *with free variables* V*, and* $\gamma \subseteq S \times \mathcal{A} \times S$*, inductively defined as follows:*

(i) $s_0 := (b_0, c_{\alpha_I}) \in S$ *is the initial node; and*
(ii) if $(b, \varphi) \in S$ *and* $b \xrightarrow{a} b'$ *such that* $update(\varphi, a)$ *is satisfiable, there is some* $(b', \varphi') \in S$ *with* $\varphi' \equiv update(\varphi, a)$*, and* $(b, \varphi) \xrightarrow{a} (b', \varphi')$ *is in* γ*, and*
(iii) the set of final nodes S_F *consists of all* (b, φ) *such that* $b \in B_F$*.*

Intuitively, the constraint graph describes all configurations reachable in $\mathcal{B}$: Every node combines a control state b with a formula φ: it represents all configurations (b, α) such that α satisfies φ. Figure 2(c) shows the CG for the DDSA obtained from $\mathcal{N}$ in Example 2, (final nodes are drawn with a double border). In fact, Fig. 2(c) is also the CG for the DDSA of $\mathcal{N}_{\mathsf{reset}}$ (basically, because the transition reset is not reachable). The crucial property of CGs is that they faithfully and completely represent the configuration space, in the following sense:

Lemma 1 ([13, Lem. 2])**.** $\mathrm{CG}_{\mathcal{B}}$ *has a path* $\pi\colon (b_I, c_{\alpha_I}) \to^* (b, \varphi)$ *s.t.* φ *is satisfied by* α *iff* $\mathcal{B}$ *has a run* $(b_I, \alpha_I) \to^* (b, \alpha)$ *whose abstraction is* $\sigma(\pi)$*.*

Here, for a path π in the CG, $\sigma(\pi)$ is the DDSA transition sequence along this path. Thus a path π in the CG captures all runs ρ with the same sequence of control states and actions such that the last assignment in ρ satisfies the formula in the last node of π. CGs are infinite in general, but for many classes of DDSAs occurring in practice, finite CGs can be computed [10,13]. These include DDSAs where all constraints are variable-to-variable/constant comparisons over $\mathbb{Q}$ like Example 2, and *bounded lookback* DDSAs whose behaviour, intuitively, depends only on a bounded number of past steps (this holds e.g. for Example 1).

3 Marking and Configuration Equivalence

Two Petri nets are *marking equivalent* if their sets of reachable markings coincide. While marking equivalence is in general undecidable for Petri nets [14], it is easy to decide for bounded Petri nets, by enumerating all markings. Here we consider marking equivalence, as well as the related problem of marking inclusion, for bounded DPNs. First, we note that if a DPN $\mathcal{N}$ was transformed into a DDSA $\mathcal{B}$ as described in [13], then the set of possible markings $\mathcal{M}ark(\mathcal{N})$ coincides with the set of reachable states in $\mathcal{B}$. Two DPNs $\mathcal{N}_1$ and $\mathcal{N}_2$ with respective DDSAs $\mathcal{B}_1$ and $\mathcal{B}_2$ are thus marking equivalent iff $\mathcal{B}_1$ and $\mathcal{B}_2$ have the same sets of reachable states. Since reachability of a single state in a DDSA is already undecidable (cf. [12, Rem. 2.6]), also marking equivalence of DPNs is undecidable.

However, we show that for bounded DPNs with finite CGs, marking equivalence can be read off the CGs: Suppose two DPNs were transformed into DDSAs $\mathcal{B}_1 = \langle B, b_I, \mathcal{A}, T, B_F, V, \alpha_I, guard \rangle$ and $\mathcal{B}_2 = \langle B, b'_I, \mathcal{A}, T', B_F, V, \alpha'_I, guard \rangle$. We assume that all components of the DDSAs coincide, except for initial states and

transitions, but this does not restrict generality as control states can be unreachable. For a DDSA $\mathcal{B}$, let $MReach(\mathcal{B}) = \{b \mid (b, \varphi) \in S\}$ for S the set of nodes in $CG_{\mathcal{B}}$, i.e., $MReach(\mathcal{B})$ is the set of control states of $\mathcal{B}$ that occur in the CG of $\mathcal{B}$.

Proposition 1. *Two DPNs $\mathcal{N}_1$ and $\mathcal{N}_2$ that correspond to DDSAs $\mathcal{B}_1$ and $\mathcal{B}_2$ with finite CGs satisfy $\mathcal{M}ark(\mathcal{N}_1) \subseteq \mathcal{M}ark(\mathcal{N}_2)$ iff $MReach(\mathcal{B}_1) \subseteq MReach(\mathcal{B}_2)$, and are marking equivalent iff $MReach(\mathcal{B}_1) = MReach(\mathcal{B}_2)$.*

Proof. First, suppose $\mathcal{M}ark(\mathcal{N}_1) \subseteq \mathcal{M}ark(\mathcal{N}_2)$, and let $M \in MReach(\mathcal{B}_1)$. By Lemma 1, there is a run of $\mathcal{B}_1$ ending in a configuration (M, α). Thus M is a reachable state of $\mathcal{B}_1$, i.e., a reachable marking of $\mathcal{N}_1$, and hence also of $\mathcal{N}_2$, so there is a process run of $\mathcal{N}_2$ (and thus a run of $\mathcal{B}_2$) ending in a configuration (M, α'). By Lemma 1, the CG for $\mathcal{B}_2$ has a node (M, φ), i.e., $M \in MReach(\mathcal{B}_2)$.

Second, if $MReach(\mathcal{B}_1) \subseteq MReach(\mathcal{B}_2)$ and $M \in \mathcal{M}ark(\mathcal{N}_1)$, some process run of $\mathcal{N}_1$ and run of $\mathcal{B}_1$ end in a configuration (M, α). By Lemma 1, the CG for $\mathcal{B}_1$ has a node (M, φ). Since $MReach(\mathcal{B}_1) \subseteq MReach(\mathcal{B}_2)$, the CG for $\mathcal{B}_2$ has a node (M, φ'). By Lemma 1, there is a run of $\mathcal{B}_2$ ending in a configuration (M, α'). This shows the inclusion statement, so the one for equivalence follows. □

Configuration Equivalence. For DPNs, the perhaps more relevant notion than marking equivalence is equivalence of sets of configurations. Let two DPNs be *configuration equivalent* if their sets of reachable configurations coincide. First, note that for a DPN $\mathcal{N}$ with associated DDSA $\mathcal{B}$, the sets of configurations of $\mathcal{N}$ and $\mathcal{B}$ coincide. Thus, we can again check the problem on the level of DDSAs. For a DDSA $\mathcal{B}$ with constraint graph with node set S, and M a state of $\mathcal{B}$, consider $\varphi_{reach}(\mathcal{B}, M) = \bigvee\{\varphi \mid (M, \varphi) \in S\}$ as a formula representation of the configurations that can occur together with M.

Proposition 2. *Let two DPNs $\mathcal{N}_1$ and $\mathcal{N}_2$ correspond to DDSAs $\mathcal{B}_1$ and $\mathcal{B}_2$.*

(1) $\mathcal{C}onf(\mathcal{N}_1) \subseteq \mathcal{C}onf(\mathcal{N}_2)$ iff $\mathcal{M}ark(\mathcal{N}_1) \subseteq \mathcal{M}ark(\mathcal{N}_2)$ and $\varphi_{reach}(\mathcal{B}_1, M) \models \varphi_{reach}(\mathcal{B}_2, M)$ for all $M \in \mathcal{M}ark(\mathcal{N}_1)$.
(2) $\mathcal{N}_1$ and $\mathcal{N}_2$ are configuration equivalent iff they are marking equivalent and $\varphi_{reach}(\mathcal{B}_1, M) \equiv \varphi_{reach}(\mathcal{B}_2, M)$ for all $M \in \mathcal{M}ark(\mathcal{N}_1)$.
(3) If $M \in \mathcal{M}ark(\mathcal{N}_1) \cap \mathcal{M}ark(\mathcal{N}_2)$ and there is some assignment α that satisfies $\varphi_{reach}(\mathcal{B}_1, M) \wedge \neg\varphi_{reach}(\mathcal{B}_2, M)$ then $(M, \alpha) \in \mathcal{C}onf(\mathcal{N}_1) \setminus \mathcal{C}onf(\mathcal{N}_2)$.

Proof. (1) First, assume $\mathcal{C}onf(\mathcal{N}_1) \subseteq \mathcal{C}onf(\mathcal{N}_2)$, so $\mathcal{M}ark(\mathcal{N}_1) \subseteq \mathcal{M}ark(\mathcal{N}_2)$. Let $\alpha \models \varphi_{reach}(\mathcal{B}_1, M)$, so $\alpha \models \varphi$ for some (M, φ) in the CG of $\mathcal{B}_1$. By Lemma 1, there is a run of $\mathcal{B}_1$ ending in a configuration (M, α). So (M, α) is a configuration of $\mathcal{N}_1$, and hence of $\mathcal{N}_2$, so (M, α) is also reachable in $\mathcal{B}_2$. Again by Lemma 1, the CG for $\mathcal{B}_2$ has a node (M, φ') such that $\alpha \models \varphi'$, so $\alpha \models \varphi_{reach}(\mathcal{B}_2, M)$.

Second, suppose $\mathcal{M}ark(\mathcal{N}_1) \subseteq \mathcal{M}ark(\mathcal{N}_2)$ and $\varphi_{reach}(\mathcal{B}_1, M) \models \varphi_{reach}(\mathcal{B}_2, M)$ for all $M \in \mathcal{M}ark(\mathcal{N}_1)$. Let $(M, \alpha) \in \mathcal{C}onf(\mathcal{N}_1)$, so reachable in $\mathcal{B}_1$. By Lemma 1, the CG for $\mathcal{B}_1$ has a node (M, φ) such that $\alpha \models \varphi$, so $\alpha \models \varphi_{reach}(\mathcal{B}_1, M)$, hence $\alpha \models \varphi_{reach}(\mathcal{B}_2, M)$. By Lemma 1, some run of $\mathcal{B}_2$ (hence of $\mathcal{N}_2$) ends in (M, α). This shows (1), which implies (2); for (3) the reasoning is similar. □

Note that marking equivalence can also be decided by finitely many reachability queries, but not configuration equivalence if the state space is infinite.

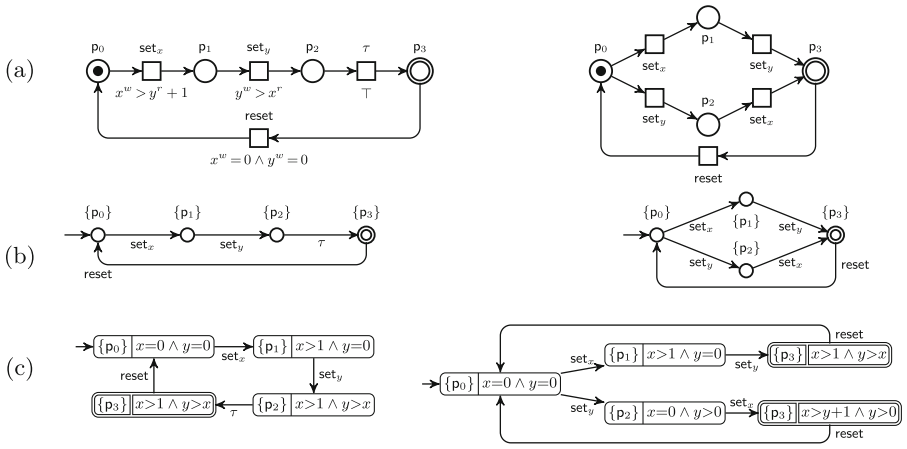


Fig. 3. Two DPNs with their DDSAs and constraint graphs.

Example 3. Consider the DPNs $\mathcal{N}_1$ and $\mathcal{N}_2$ in Fig. 3(a) over variables $V = \{x, y\}$ with sort `rat`, with $\alpha_I(x) = \alpha_I(y) = 0$ (the guards of actions in $\mathcal{N}_2$ coincide with those in $\mathcal{N}_1$). The respective DDSAs $\mathcal{B}_1$ and $\mathcal{B}_2$ and their constraint graphs are shown in Fig. 3(b) and (c). The markings occurring in the CGs coincide, so the two DPNs are marking equivalent. However, they are not configuration equivalent: For instance, the formulas $\varphi_{reach}(\mathcal{B}_1, \{\mathsf{p}_3\}) = (x{>}1 \wedge y{>}x)$ and $\varphi_{reach}(\mathcal{B}_2, \{\mathsf{p}_3\}) = (x{>}1 \wedge y{>}x) \vee (x{>}y{+}1 \wedge y{>}0)$ are not equivalent. This is witnessed by any assignment that satisfies $\varphi_{reach}(\mathcal{B}_2, M) \wedge \neg\varphi_{reach}(\mathcal{B}_1, M) \equiv (x{>}y{+}1 \wedge y{>}0)$, e.g., $\alpha(x) = 2$ and $\alpha(y) = 1$, so $(\{\mathsf{p}_3\}, \alpha)$ is a configuration of $\mathcal{N}_2$ but not of $\mathcal{N}_1$. However, $\varphi_{reach}(\mathcal{B}_1, M) \models \varphi_{reach}(\mathcal{B}_2, M)$ for all M, so $\mathcal{C}onf(\mathcal{N}_1) \subseteq \mathcal{C}onf(\mathcal{N}_2)$.

For another example, as the CGs of $\mathcal{N}$ and $\mathcal{N}_{\mathsf{reset}}$ from Example 2 coincide, the DPNs are marking and configuration equivalent. One can also use Propositions 1 and 2 to check that the DPN in Example 1, and the version in [19, Fig. 12.7] in which guards were discovered automatically, are marking but not configuration equivalent.

4 Language Equivalence

Language equivalence is undecidable for unbounded Petri nets [14], but decidable for bounded nets, in which case it basically amounts to checking equivalence of two regular languages [15]. Here we consider the respective problem for bounded

Algorithm 1. Checking language equivalence of node sets and DDSAs

```
1: procedure SetEquiv(X, Y, G, G')
2:     R := ∅, todo := {(X, Y)}
3:     while todo ≠ ∅ do
4:         extract (X, Y) from todo
5:         if (X, Y) ∈ R then
6:             continue
7:         if fin_B(X) ≠ fin_B'(Y) then
8:             return false
9:         for each a ∈ A do
10:            add (next_G(X, a), next_G'(Y, a)) to todo
11:        add (X, Y) to R
12:    return true
1: procedure Equiv(B, B')
2:     compute CG_B and CG_B', let s_0 and s'_0 be the initial states
3:     return SetEquiv({s_0}, {s'_0}, CG_B, CG_B')
```

DPNs. Given a DPN $\mathcal{N}$, let its *language* $\mathcal{L}(\mathcal{N})$ be the set of all $\ell(t'_1), \ldots \ell(t'_m)$ such that there is a process run $(M_I, \alpha_0) \xrightarrow{(t_1,\beta_1)} \ldots \xrightarrow{(t_n,\beta_n)} (M', \alpha')$ and $t'_1, \ldots, t'_m$ is the maximal subsequence of $t_1, \ldots, t_n$ such that $\ell(t'_i) \neq \tau$. We assume that silent transitions in DPNs do not write variables, and moreover, that there are no cycles that consist of silent transitions only. Thus, after transforming a DPN into a DDSA, we can eliminate silent transitions by replacing them with non-silent transitions, similar as done in NFAs: if $b \xrightarrow{t} b'$ with t silent with guard c, and $b' \xrightarrow{t_1} b_1, \ldots, b' \xrightarrow{t_m} b_m$ are all transitions starting from b', we remove $b \xrightarrow{t} b'$ and add transitions $b \xrightarrow{t'_1} b_1, \ldots, b \xrightarrow{t'_m} b_m$ with $guard(t'_i) = guard(t_i) \wedge c$. This replacement can be repeated until there are no silent transitions left. Thus, we can consider the language equivalence problem for DDSAs without silent transitions. Let the language $\mathcal{L}(\mathcal{B})$ of a DDSA $\mathcal{B}$ be the set of all words $a_1, \ldots, a_n$ such that $\mathcal{B}$ has a run $(b_I, \alpha_I) \xrightarrow{a_1} (b_1, \alpha_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (b_n, \alpha_n)$ and $b_n \in B_F$. Then, for every bounded DPN $\mathcal{N}$ there is a DDSA $\mathcal{B}$ such that $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{B})$.

We now consider two DPNs $\mathcal{N}_1$ and $\mathcal{N}_2$ that correspond to DDSAs $\mathcal{B}_1 = \langle B_1, b_I, \mathcal{A}, T, B_F, V, \alpha_I, guard \rangle$ and $\mathcal{B}_2 = \langle B_2, b'_I, \mathcal{A}, T_2, B'_F, V, \alpha'_I, guard \rangle$. Note that the data variables, actions, and guards are supposed to coincide, but control states, transitions and the initial assignment may be different.

Language equivalence can be checked by an adaptation of an algorithm to check language equivalence of NFAs [3, Fig. 4], see the procedure Equiv in Algorithm 1. We use the following shorthands: for a set of nodes X in a constraint graph $\mathcal{G}$, $fin_{\mathcal{B}}(X)$ is true iff X contains a node that is final in $\mathcal{G}$. Moreover, $next_{\mathcal{G}}(X, a)$ is the set of all CG nodes s' such that $\mathcal{G}$ has an edge $s \xrightarrow{a} s'$ for some $s \in X$.

For a set of nodes X in a CG $\mathcal{G}$, we now write $\mathcal{L}_{\mathcal{G}}(X)$ for the set of words accepted starting from a state in X, i.e., the set of words $a_1, \ldots, a_n$ such that $\mathcal{G}$ has a path $(b, \varphi_0) \xrightarrow{a_1} (b_1, \varphi_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (b_n, \varphi_n)$ from some $b \in X$ such that (b_n, φ_n) is final in $\mathcal{G}$. Then, procedure SetEquiv can be used to check whether

two sets of states accept the same words. This is formally stated in the next result, which follows directly from [3, Prop. 2], considering $\mathcal{G}$ and $\mathcal{G}'$ as NFAs and X and Y as sets of states therein.

Proposition 3. *For sets of CG nodes X in $\mathcal{G}$ and Y in $\mathcal{G}'$,* SetEquiv$(X, Y, \mathcal{G}, \mathcal{G}')$ *is true iff* $\mathcal{L}_{\mathcal{G}}(X) = \mathcal{L}_{\mathcal{G}'}(Y)$.

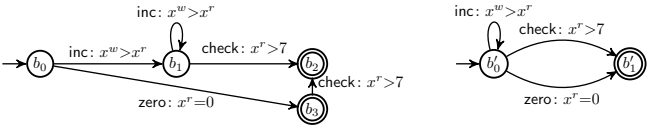
Proposition 4. *For two DPNs $\mathcal{N}_1$, $\mathcal{N}_2$ and DDSAs $\mathcal{B}_1$, $\mathcal{B}_2$ without silent transitions such that $\mathcal{L}(\mathcal{N}_1) = \mathcal{L}(\mathcal{B}_1)$ and $\mathcal{L}(\mathcal{N}_1) = \mathcal{L}(\mathcal{B}_1)$,* Equiv$(\mathcal{B}_1, \mathcal{B}_2) =$ true *iff $\mathcal{N}_1$ and $\mathcal{N}_2$ are language equivalent.*

Proof (sketch). Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be the CGs of $\mathcal{B}_1$ and $\mathcal{B}_2$. We have Equiv$(\mathcal{B}_1, \mathcal{B}_2)$ = SetEquiv$(\{s_0\}, \{s'_0\}, \mathcal{G}_1, \mathcal{G}_2)$, which is true iff $\mathcal{L}_{\mathcal{G}_1}(\{s_0\}) = \mathcal{L}_{\mathcal{G}_2}(\{s'_0\})$ by Proposition 3. From Lemma 1 it follows that $\mathcal{L}_{\mathcal{G}_1}(\{s_0\})$ coincides with $\mathcal{L}(\mathcal{B}_1) = \mathcal{L}(\mathcal{N}_1)$, and similar for $\mathcal{B}_2$, so the claim follows. □

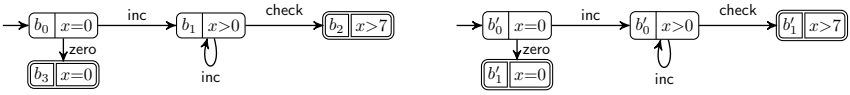
Language inclusion can be reduced to language equivalence since $\mathcal{L}(X) \subseteq \mathcal{L}(Y)$ iff $\mathcal{L}(X) \cup \mathcal{L}(Y) \subseteq \mathcal{L}(Y)$ iff $\mathcal{L}(X \cup Y) \subseteq \mathcal{L}(Y)$, cf. [3], so Algorithm 1 also serves to decide language inclusion.

For instance, since the constraint graphs of $\mathcal{N}$ and $\mathcal{N}_{\mathsf{reset}}$ from Example 2 coincide, the DPNs are language equivalent. On the other hand, Equiv can be used to detect that the languages of the DPNs in Fig. 3 are not equal (e.g., $\mathsf{set}_y, \mathsf{set}_x$ is not accepted by the first CG). The next example shows that it does not suffice to execute Equiv on the DDSAs instead of the constraint graphs.

Example 4. Consider the following two DDSAs:



If both $\alpha_I(x) = \alpha'_I(x) = 0$, their CGs coincide, so Equiv concludes language equivalence (the language being $\mathsf{inc}^+\mathsf{check}{+}\mathsf{zero}$, in regular expression notation).



However, note that the procedure Equiv could not be run on the DDSAs directly, since detection of dead transition requires a reasoning based on reachable configurations, as done in CGs. Moreover, note that with e.g. $\alpha_I(x) = \alpha'_I(x) = 10$, the DDSAs would not be language equivalent because the single-letter word check would be in the language of the second, but not of the first DDSA.

Algorithm 1 can be modified to return a witness if equivalence does not hold. For reasons of space, we cannot formalize this here, but the main idea is

straightforward: every pair in *todo* can be associated with a word that led to this pair of node sets, starting with the empty word ϵ for the initial nodes. When returning false in line 9, SetEquiv can then also return the word accumulated up to this point, which witnesses the difference.

It can also be noted that if $\mathcal{N}_1$ and $\mathcal{N}_2$ are language equivalent and $\alpha_I = \alpha'_I$, then the DPNs are also configuration equivalent, because actions have the same guards in both DPNs. However, the converse does not hold: if $\mathcal{N}_1$ is a copy of $\mathcal{N}_2$ where actions are renamed but have the same guards, then the two DPNs are configuration equivalent but not language equivalent.

Conclusion. We proposed techniques to check marking, configuration, and language equivalence for bounded DPNs with finite constraint graphs. Our correctness results thus imply that these notions are decidable for bounded DPNs with finite CGs, which captures many DPNs from practice [13]. To the best of our knowledge, these are the first results to compare DPNs based on behaviour. In future work, it would be interesting to study other notions of equivalence, e.g. language equivalence taking data into account. Also the study of quantitative similarity measures would be of interest, to e.g. express *how large* the intersection/difference of configuration spaces/language is. Our approach could also be implemented on top of the tool `ada` [11,13], which already computes CGs.

References

1. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Process equivalence: comparing two process models based on observed behavior. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 129–144. Springer, Heidelberg (2006). https://doi.org/10.1007/11841760_10
2. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, 2nd edn., vol. 336, pp. 1267–1329. IOS Press (2021)
3. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: Proceedings of 13th POPL, pp. 457–468. ACM (2013)
4. Damaggio, E., Deutsch, A., Vianu, V.: Artifact systems with data dependencies and arithmetic. ACM Trans. Database Syst. **37**(3), 22:1–22:36 (2012)
5. de Leoni, M., Mannhardt, F.: Decision discovery in business processes. In: Sakr, S., Zomaya, A. (eds.) Encyclopedia of Big Data Technologies, pp. 1–12. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-63962-8_96-1
6. Dijkman, R.M., Dumas, M., van Dongen, B.F., Käärik, R., Mendling, J.: Similarity of business process models: metrics and evaluation. Inf. Syst. **36**(2), 498–516 (2011)
7. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_5
8. Euzenat, J., Shvaiko, P.: Ontology Matching, 2nd edn. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38721-0
9. Felli, P., de Leoni, M., Montali, M.: Soundness verification of decision-aware process models with variable-to-variable conditions. In: Proceedings of 19th ACSD, pp. 82–91. IEEE (2019)

10. Felli, P., de Leoni, M., Montali, M.: Soundness verification of data-aware process models with variable-to-variable conditions. Fund. Inf. **182**(1), 1–29 (2021)
11. Felli, P., Montali, M., Winkler, S.: CTL* model checking for data-aware dynamic systems with arithmetic. In: Proceedings of 11th IJCAR, vol. 13385, pp. 36–56 (2022)
12. Felli, P., Montali, M., Winkler, S.: Linear-time verification of data-aware dynamic systems with arithmetic. In: Proceedings of 36th AAAI, pp. 5642–5650 (2022)
13. Felli, P., Montali, M., Winkler, S.: Soundness of data-aware processes with arithmetic conditions. In: Franch, X., Poels, G., Gailly, F., Snoeck, M. (eds.) Proceedings of 34th CAiSE, LNCS, vol. 13295, pp. 389–406. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-07472-1_23
14. Hack, M.: Decidability questions for Petri Nets. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge (1976)
15. Jančar, P., Moller, F.: Checking regular properties of petri nets. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 348–362. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60218-6_26
16. Kunze, M., Weidlich, M., Weske, M.: Behavioral similarity – a proper metric. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 166–181. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2_15
17. de Leoni, M., Dumas, M., García-Bañuelos, L.: Discovering branching conditions from business process execution logs. In: Cortellessa, V., Varró, D. (eds.) FASE 2013. LNCS, vol. 7793, pp. 114–129. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37057-1_9
18. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. In: Trujillo, J.C., et al. (eds.) ER 2018. LNCS, vol. 11157, pp. 219–235. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00847-5_17
19. Mannhardt, F.: Multi-perspective Process Mining. Ph.D. thesis, Technical University of Eindhoven (2018)
20. Mannhardt, F., de Leoni, M., Reijers, H., van der Aalst, W.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
21. Schoknecht, A., Thaler, T., Fettke, P., Oberweis, A., Laue, R.: Similarity of business process models - a state-of-the-art analysis. ACM Comput. Surv. **50**(4), 52:1–52:33 (2017)