Soundness of Data-Aware Processes with Arithmetic Conditions

Paolo Felli[0000−0002−8021−3430], Marco Montali[0000−0002−8021−3430], and Sarah Winkler[0000−0001−8114−3107]⋆

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

Abstract. Data-aware processes represent and integrate structural and behavioural constraints in a single model, and are thus increasingly investigated in business process management and information systems engineering. In this spectrum, Data Petri nets (DPNs) have gained increasing popularity thanks to their ability to balance simplicity with expressiveness. The interplay of data and control-flow makes checking the correctness of such models, specifically the well-known property of soundness, crucial and challenging. A major shortcoming of previous approaches for checking soundness of DPNs is that they consider data conditions without arithmetic, an essential feature when dealing with real-world, concrete applications. In this paper, we attack this open problem by providing a foundational and operational framework for assessing soundness of DPNs enriched with arithmetic data conditions. The framework comes with a proof-of-concept implementation that, instead of relying on ad-hoc techniques, employs off-the-shelf established SMT technologies. The implementation is validated on a collection of examples from the literature, and on synthetic variants constructed from such examples.

Keywords: Soundness · Data Petri nets · arithmetic conditions · SMT.

1 Introduction

Integrating structural and behavioral aspects to holistically capture how information systems dynamically operate over data through actions and processes is a central problem in business process management (BPM) [20] and information systems engineering [23]. This is witnessed by the mutual cross-fertilization of the two areas on this topic, with models and approaches originating from BPM and its underlying formal foundations being then applied to information and enterprise systems [18,11,21], and vice-versa [3,24].

The interplay of data and control-flow makes checking the correctness of such models crucial and challenging. From the formal point of view, the problem is undecidable even for severely restricted models and correctness properties, both in the case of simple data variables [13] and richer relational structures [5,7].

⋆ This work is partially supported by the UNIBZ projects DaCoMan, QUEST, SMART-APP, VERBA, and WineId.

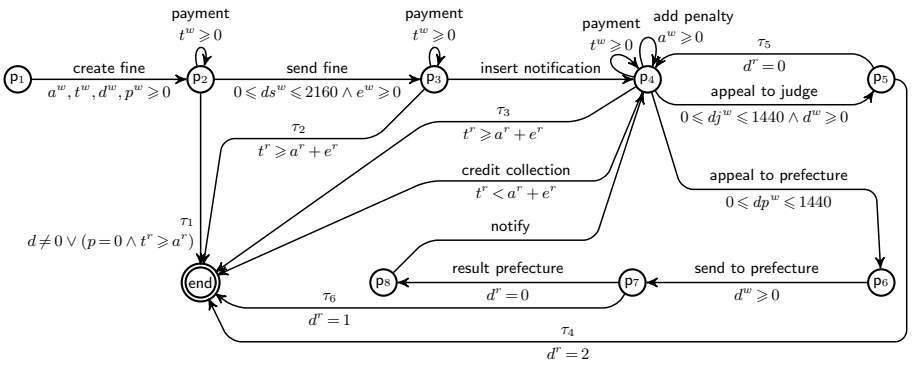


Fig. 1. Data-aware process for road fines [17].

From the modeling perspective, the difficulty in combining these two dimensions is exacerbated by the fact that, more and more, models are obtained through a two-step approach: a first, automated discovery step produces a baseline model from event data, followed by a refinement and modification step driven by human ingenuity. The following example illustrates the challenge.

Example 1. A management process for road fines from an information system of the Italian police was presented as in [17] using a Data Petri nets (DPN). DPNs have gained increasing popularity thanks to their ability to balance simplicity with expressiveness. They focus on the evolution of a single (case) object evolved by the process (or a fixed number of inter-related objects), combining a Petri net-based control-flow with case variables and data conditions, capturing decisions and constrained updates. The process maintains seven case data variables: a (amount), t (total amount), d (a dismissal code), p (points deducted), e (expenses), and three time intervals ds, dp, dj. The process starts by creating a fine for a traffic offense in the system (create fine). A notification is sent to the offender within 90 days, i.e., 2160h, by action send fine) and this is entered in the system (insert notification). If the offender pays an amount t that exceeds the fine a plus expenses e, the process terminates via τ_1, τ_2, or τ_3. For the less happy paths, there is a credit collection action if the paid sum was not enough; and the offender may file a protest, via appeal to judge, appeal to prefecture, and subsequent actions. The appeals again need to respect a certain time frame.

For simplicity, in Figure 1 we present the model as a transition system instead of a Petri net. It was generated from real-life logs through multi-perspective process mining techniques, then enriched manually with more sophisticated arithmetic constraints extracted from domain knowledge [17]. *What is not obvious is that the process gets stuck in state* p_7 *if send to prefecture writes value* $d > 1$.

Examples like this call for a virtuous circle where process mining, human modelling, and *automated* verification techniques for correctness checking empower each other. A well-established formal notion of correctness for dynamic

systems is that of *soundness* [1], defined over the well-known Petri net class of workflow nets. Intuitively, this property requires (i) that there are no activities in the process that cannot be executed in any of the possible executions; (ii) that from every reachable configuration the process can always be concluded by reaching a *final* configuration and (iii) that final configurations are always reached in a 'clean way', without leaving any thread of the process still hanging. After the seminal work in [1], which solely focuses on the evolution of single process instances in pure control-flow terms, several follow-up approaches were brought forward to define and study soundness for richer control-flow structures [2], several isolated cases [12], and presence of resources [22], showing decidability of the problem without entering into the engineering of verification tools.

When considering data-aware processes, the standard formulation of soundness is insufficient, as it does not consider how data affects the execution. This makes prior works not readily applicable to solve the problem. Refined notions of soundness have in fact been put forward to take data into account. Specifically, in [13] the property of *data-aware* soundness was obtained by lifting the standard soundness property of workflow nets to DPNs [16,13] (see the example above), by resorting to a translation to colored Petri nets. However, data conditions attached to activities were restricted to variable-to-constant comparisons. The approach was later extended to DPNs with a guard language that supports direct comparison of case data [8]. In parallel, [4] introduced notions of *decision-aware* soundness, where the focus is on data consumed and produced by (DMN) decision tables attached to the process. It was later shown in [15] how DPNs could be used to capture BPMN processes enriched with DMN S-FEEL decision tables, and how the different decision-aware soundness notions [4] could be recast as data-aware soundness [13].

While data-aware soundness is a crucial notion that captures also the problem in Ex. 1, a common shortcoming present in the literature is the limited expressivity of data conditions attached to activities and decision rules: *they cannot handle expressions with arithmetic computations.* For instance, one can check that the current credit card balance b is equal or larger than the price p of the purchased item (i.e., that $b \geqslant p$), but not that it is greater than the price plus some threshold amount t that could be obtained through a human task (i.e., that $b \geqslant p+t$). Clearly, this makes the existing technique not applicable to a very large number of real world applications (for instance, Ex. 1), revealing a research gap in the field that motivates the need of novel results in this spectrum.

Contributions and methodology. Having identified this open research problem, we aim at contributing to the advancement of the body of knowledge in information systems engineering by answering three research questions:

1. Is soundness checking decidable for DPNs equipped with arithmetic?
2. Is there an operational way to conduct the check?
3. Is this operational way effective from the computational point of view?

We answer these through theoretical and algorithmic research, and through the creation of a concrete IT proof-of-concept artifact for soundness checking.

Specifically, we focus on DPNs supporting unlimited addition of variables but only constant multiplication, that is, *linear arithmetic*, which captures many real-world use cases. We address the first two research questions at once by lifting the approach in [8] to our richer setting, introducing a soundness checking procedure consisting of three algorithmic steps: (1) we transform the DPN into a labelled transition system called *data-aware dynamic system* (DDS) [14]; (2) we construct a *constraint graph*, which acts as a symbolic representation of the reachable state space via a finite set of formulas; (3) a set of satisfiability checks is performed using the formulas in the graph, and we prove that the DPN is unsound if and only if one of these checks succeeds. The constraint graph built for a DDS with arithmetic may in general be infinite. However, it is finite and computable, so that our check becomes a decision procedure, when the given process guarantees that reachable configurations are suitably limited (e.g. in that only a bounded part of the computation history is relevant, or the constraint language is sufficiently restricted). This requirement holds for well-identified classes of processes, formally captured by a *finite history set* [9]. For instance, it applies to all DPNs used in our evaluation, including Ex. 1.

Towards answering the third research question, we provide a proof-of-concept implementation of our framework in the tool `ada`. Being research in this setting at an early stage, we cannot rely on well-established empirical or experimental methods to validate this IT artifact. To mitigate this problem, we proceed as follows. First and foremost, instead of relying on ad-hoc techniques, our tool employs off-the-shelf SMT solvers as a backend. This guarantees that the main computation burden, namely the satisfiability checks in the third algorithmic step, is handled by third-party, industrially-validated software. Secondly, since there is no benchmark for DPNs, we set up a preliminary, performance evaluation in two steps: *(i)* we collect, and check soundness of, all DPN examples/case studies present in the literature to model real-world data-aware processes in information systems of various types; *(ii)* we construct synthetic variants of some of these examples, in order to test how the performance of `ada` changes by increasing actions, variables and conditions present in the model.

The paper is structured as follows. In Sec. 2, we fix our DPN model and define data-aware soundness, illustrating its high-level verification procedure in Sec. 3. The following sections detail the required steps: in Sec. 4 we relate data-aware soundness of a DPN to that of a corresponding transition system. We explain the constraint graph in Sec. 5, and show in Sec. 6 how it can be used to check data-aware soundness. Our implementation and experiments are the topic of Sec. 7. In Sec. 8 we conclude and comment on future work.

Proofs of the technical results are available in an extended report [10].

2 Background

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

Constraints. We start by fixing a set of data types for the variables manipulated by a process: let $\Sigma = \{\texttt{bool}, \texttt{int}, \texttt{rat}\}$ with associated domains of booleans $\mathcal{D}(\texttt{bool}) = \mathbb{B}$, integers $\mathcal{D}(\texttt{int}) = \mathbb{Z}$, and rationals $\mathcal{D}(\texttt{rat}) = \mathbb{Q}$. We assume a fixed set of *process variables* V, so there is a function $type\colon V \mapsto \Sigma$ assigning a type to each variable. For instance, in Ex. 1 the set of process variables is $V = \{a, d, dj, dp, ds, p, t\}$ all of type `int` (i.e., $type(a) = \texttt{int}$, etc). For a type $\sigma \in \Sigma$, V_σ denotes the subset of variables of type σ. To manipulate variables, we consider expressions c with the following grammar:

$$c := x_{\texttt{bool}} \mid b \mid n_1 \ op \ n_2 \mid r_1 \ op \ r_2 \mid c_1 \wedge c_2$$
$$op := \neq \mid = \mid \geqslant \mid > \quad n := x_{\texttt{int}} \mid k \mid k_1 \cdot n_1 + k_2 \cdot n_2 \quad r = x_{\texttt{rat}} \mid q \mid q_1 \cdot r_1 + q_2 \cdot r_2$$

where: $x_{\texttt{bool}} \in V_{\texttt{bool}}, x_{\texttt{int}} \in V_{\texttt{int}}$, and $x_{\texttt{rat}} \in V_{\texttt{rat}}$ respectively denote a boolean, integer, and rational variable, while $b \in \mathbb{B}$, $k \in \mathbb{Z}$, and $q \in \mathbb{Q}$ respectively denote a boolean, integer, and rational constant. We consider booleans, integers, and rationals as three prototypical examples of three datatypes, respectively relying on a finite, infinite discrete, and infinite dense domain. Similar datatypes, such as strings equipped with equality and real numbers, can be seamlessly handled. These expressions will be used to capture conditions on the values of variables that are read and written during the execution of process activities. For this reason, we call them *constraints*. The set of constraints over V is denoted $\mathcal{C}(V)$.

For our process variables V, we consider two disjoint sets of *annotated* variables $V^r = \{v^r \mid v \in V\}$ and $V^w = \{v^w \mid v \in V\}$ which are read and written by process activities, respectively, as explained below, and we assume $type(v^r) = type(v^w) = type(v)$ for every $v \in V$. For instance, the constraint $t^r \geqslant a^r + e^r$ in Ex. 1 dictates that the current value of variable t is greater or equal than the sum of the values of a and r; whereas $0 \leqslant dj^w \wedge dj^w \leqslant 1440$ requires that the new value given to dj (i.e., assigned to dj as a result of the execution of the activity to which this constraint is attached) is between 0 and 1440. On the other hand, $a^w > a^r$ would mean that the new value of a is larger than its current value. More generally, given a constraint c as above, we refer to the annotated variables in V^r and V^w that appear in c as the *read* and *written variables*, respectively.

An *assignment* α is a total function $\alpha\colon V \mapsto D$ mapping each variable in V to a value in its domain. We say that α *satisfies* a constraint c over V, written $\alpha \models c$, if the evaluation of c under α is true. For instance, the assignment α such that $\alpha(t) = 10$, $\alpha(a) = 7$, and $\alpha(v) = 0$ for $v \in V$ otherwise, satisfies $t^r \geqslant a^r + e^r$.

Our constraint language is that of linear arithmetic over integers and rationals, which is decidable, and for which a range of mature SMT (satisfiability modulo theories) solvers is available. Moreover, linear arithmetic is known to enjoy *quantifier elimination* [19]: if φ is a formula with atoms in $\mathcal{C}(V \cup \{x\})$, there is some φ' with free variables V that is logically equivalent to $\exists x.\varphi$, i.e., $\varphi' \equiv \exists x.\varphi$. We assume that qe is a quantifier elimination procedure that returns such a formula, as implemented in off-the-shelf SMT solvers.

We adopt the following standard definition of Data Petri Nets (DPNs) [16,17].

Definition 1 (DPN). *A 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 Petri net with non-empty, disjoint sets of places P and tran-*

sitions T, a flow relation $F : (P \times T) \cup (T \times P) \mapsto \mathbb{N}$ and a labelling function $\ell\colon T \mapsto \mathcal{A}$, where $\mathcal{A}$ is a finite set of activity labels; (2) V is a set of process variables (all with a type); and (3) guard$\colon T \mapsto \mathcal{C}(V^r \cup V^w)$ is a guard mapping.

Example 2. Consider a simple auction process modeled by the DPN in Fig. 2. The initial and final markings are $M_I = \{\mathsf{p}_0\}$ and $M_F = \{\mathsf{p}_3\}$. It maintains the set of variables $V = \{o, t\}$, where o (domain $\mathbb{Q}$) holds the last offer issued by a bidder, and t (domain $\mathbb{Z}$) is a timer. The initial assignment is $\alpha_I(o) = \alpha_I(t) = 0$. We briefly explain the working of the process: the action init initializes the timer t to a positive value (e.g., of days) and the offer o to 0; as long as the timer has not expired, 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 $\mathcal{N}$. For illustration purposes, we will also consider two variants of this DPN: $\mathcal{N}_{\mathsf{reset}}$ extends $\mathcal{N}$ by a reset action that restarts the process if the offer in the final state is 0 (drawn in red), and $\mathcal{N}_{\mathsf{thresh}}$ adds to $\mathcal{N}$ the transition thresh which leads to the final state if the offer exceeds a threshold (drawn in blue).

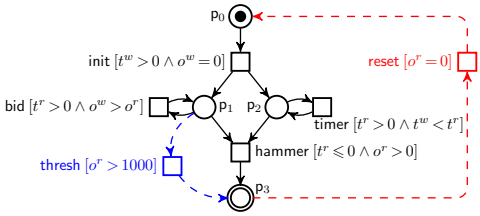


Fig. 2. DPN for simple auction model.

The variables that are read and written by a transition t are denoted $read(t)$=$\{v \mid v^r$ occurs in $guard(t)\}$, $write(t)$=$\{v \mid v^w$ occurs in $guard(t)\}$, respectively. For instance, for t the activity labelled bid in Fig. 2, $write(t) = \{o\}$ and $read(t) = \{o, w\}$.

We call a *state variable assignment*, denoted α, an assignment with domain V. In contrast, a *transition variable assignment*, denoted β, is a (partial) function that assigns values of correct type to the annotated variables $V^r \cup V^w$, used to specify how variables change during activity executions (cf. Def. 2).

For a DPN $\mathcal{N}$ with underlying Petri net (P, T, F, ℓ), a *marking* $M : P \mapsto \mathbb{N}$ assigns every place a number of tokens. A *state* of $\mathcal{N}$ is a pair (M, α) of a marking and a state variable assignment, which thus accounts for both the control flow progress and the current values of variables in V. For instance, $(\{\mathsf{p}_0\}, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right])$ is a state for the net of Ex. 2. We next define when transitions may fire in a DPN.

Definition 2 (Transition firing). *A transition $t \in T$ is* enabled *in a state (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 assigns read variables as by α,
(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 state (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 $(M, \alpha) \xrightarrow{(t,\beta)} (M', \alpha')$.*

Given $\mathcal{N}$, we fix one state (M_I, α_0) as *initial*, where M_I is the initial marking of the underlying Petri net (P, T, F, ℓ) and α_0 is a state variable assignment

that specifies the initial value of all variables in V. Similarly, we denote the final marking as M_F, and call *final* any state of the form (M_F, α_F) for some α_F. For instance, the net in Ex. 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, while $(\{\mathsf{p}_3\}, \left[\begin{smallmatrix}t=0\\o=5\end{smallmatrix}\right])$ is one final state.

We say that (M', α') is *reachable* in a DPN iff there exists a sequence of transition firings $(M_I, \alpha_0) \xrightarrow{(t_1,\beta_1)} \dots \xrightarrow{(t_n,\beta_n)} (M', \alpha')$, denoted also as $(M_I, \alpha_0) \to^* (M', \alpha')$. Such a sequence is a (valid) *process run* if the resulting state (M', α') is final. For instance, a possible sequence of transition firings in Ex. 2 (in which the timer t is initialized to 1 day, then decremented) is:

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

For simplicity of presentation, in the remainder of this paper, we restrict to *bounded* DPNs, that is, DPNs where the number of tokens in reachable markings is bounded by some $m \in \mathbb{N}$. Indeed, detecting unboundedness (which in turn witnesses unsoundness) can be done as in [8], where it is shown that the standard unboundedness detection techniques based on coverability graphs also apply to the data-aware setting. For instance, the DPNs $\mathcal{N}$, $\mathcal{N}_{\mathsf{reset}}$, and $\mathcal{N}_{\mathsf{thresh}}$ in Ex. 2 are 1-bounded. Next, we define the crucial property of data-aware soundness.

Definition 3 (Data-aware soundness). *A DPN is* data-aware sound *iff:*

(P1) *if $(M_I, \alpha_0) \to^* (M, \alpha)$ there is some α' such that $(M, \alpha) \to^* (M_F, \alpha')$ for all M, α, i.e., any sequence can be continued to a process run;*

(P2) *if $(M_I, \alpha_0) \to^* (M, \alpha)$ and $M \geqslant M_F$ then $M = M_F$ for all M, α, i.e., termination is clean; and*

(P3) *for all $t \in T$ there is a sequence $(M_I, \alpha_0) \to^* (M, \alpha) \xrightarrow{(t,\beta)} (M', \alpha')$ for some M, M', α, α', and β, i.e., there are no dead transitions.*

For instance, the DPN $\mathcal{N}$ from Ex. 2 violates (P1) because after the sequence (1) above no further transition is applicable, but the reached state is not final. $\mathcal{N}_{\mathsf{reset}}$ also violates (P3) because the transition reset is dead: if a token reaches the place p_3, o will never have value 0. On the other hand, $\mathcal{N}_{\mathsf{thresh}}$ violates also (P2) as the following steps lead to marking $\{\mathsf{p}_2, \mathsf{p}_3\} > \{\mathsf{p}_3\} = M_F$:

$$(\{\mathsf{p}_0\}, \left[\begin{smallmatrix}t=0\\o=0\end{smallmatrix}\right]) \xrightarrow{\mathsf{init}} (\{\mathsf{p}_1, \mathsf{p}_2\}, \left[\begin{smallmatrix}t=1\\o=0\end{smallmatrix}\right]) \xrightarrow{\mathsf{bid}} (\{\mathsf{p}_1, \mathsf{p}_2\}, \left[\begin{smallmatrix}t=1\\o=1000\end{smallmatrix}\right]) \xrightarrow{\mathsf{thresh}} (\{\mathsf{p}_2, \mathsf{p}_3\}, \left[\begin{smallmatrix}t=1\\o=1000\end{smallmatrix}\right])$$

3 Soundness Checking: The High-Level Perspective

Alg. 1 gives a bird's-eye view of our soundness checking procedure. The initial step is to transform the given DPN $\mathcal{N}$ into a special kind of transition system (called DDS) $\mathcal{B}$, by unfolding the interleaving semantics. The respective procedure DPNtoDDS is detailed in Sec. 4. Next, in line 3, the procedure computeCG constructs the constraint graph of $\mathcal{B}$ as a symbolic representation of all reachable states, as explained in Sec. 5. In lines 4, 6, and 8 the routines badTermination, deadTransition, and blockedState then use the constraint graph $\mathrm{CG}_{\mathcal{B}}$ to check whether $\mathcal{N}$ violates the properties (P2), (P3), and

(P1) of Def. 3, respectively (see Sec. 6). If one of these properties does not hold, the procedure returns *false* immediately, otherwise data-aware soundness is confirmed by returning *true* in line 10. The reason why we check (P1) last is that the other two checks are significantly cheaper.

Algorithm 1 Procedure to check data-aware soundness of a DPN

```
1: procedure checkSound(𝒩)
2:     ℬ ← DPNtoDDS(𝒩)
3:     CG_ℬ ← computeCG(ℬ)
4:     if badTermination(CG_ℬ, 𝒩) then return false        ▷ see Alg. 2
5:     if deadTransition(CG_ℬ, 𝒩) then return false        ▷ see Alg. 2
6:     if blockedState(CG_ℬ, 𝒩) then return false          ▷ see Alg. 2
7:     return true
```

4 From DPNs to Transition Systems

This section details the first step in our soundness checking procedure: to unfold the interleaving semantics of the given DPN into a labelled transition system called *data-aware dynamic system* (DDS) [14]. We start by defining DDSs.

Definition 4. *A* DDS $\mathcal{B} = \langle B, b_I, \mathcal{A}, \Delta, B_F, V, \alpha_I, guard \rangle$ *is a labelled transition system such that (i)* B *is a finite set of* states*, with* $b_I \in B$ *the initial one; (ii)* $\mathcal{A}$ *is a set of* actions*; (iii)* $\Delta \subseteq B \times \mathcal{A} \times B$ *is a* transition relation*; (iv)* $B_F \subseteq B$ *are* final states*; (v)* V *is the set of* process variables*; (vi)* α_I *is the* initial assignment*; (vii)* $guard \colon \mathcal{A} \mapsto \mathcal{C}(V^r \cup V^w)$ *fixes* executability constraints *on actions.*

Fig. 3 shows three example DDSs (that are in fact obtained from transforming the DPNs in Ex. 2, as defined below). The action guards are the same as in Fig. 2, but have been omitted for readability. We denote a transition from state b to b' by executing an action $a \in \mathcal{A}$ as $b \xrightarrow{a} b'$. For instance, the DDS $\mathcal{B}$ in Fig. 3 admits a transition $\mathsf{p}_0 \xrightarrow{\mathsf{init}} \mathsf{p}_{12}$. A *configuration* of $\mathcal{B}$ is a pair (b, α) where $b \in B$ and α is an assignment. For instance, $(\mathsf{p}_0, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right])$ is the initial configuration of $\mathcal{B}$ in Fig. 3. An *action firing* is a pair (a, β) of an action $a \in \mathcal{A}$ and a transition variable assignment β, i.e., a function $\beta \colon V^r \cup V^w \mapsto D$. As defined next, an action firing (a, β) transforms a configuration (b, α) into a new configuration (b', α') by changing state as defined by action a, and updating the assignment α to α', in agreement with the action guard. In the new assignment α', variables that are not written keep their previous value as per α, whereas written variables are updated according to β. Let $write(a) = \{x \mid x^w \in V^w \text{ occurs in } guard(a)\}$.

Definition 5. *A DDS* $\mathcal{B} = \langle B, b_I, \mathcal{A}, \Delta, B_F, V, \alpha_I, guard \rangle$ admits a step *from configuration* (b, α) *to* (b', α') *via action firing* (a, β)*, denoted* $(b, \alpha) \xrightarrow{a,\beta} (b', \alpha')$*, if* $b \xrightarrow{a} b'$ *and (i)* $\beta(v^r) = \alpha(v)$ *for all* $v \in V$*; (ii) the new state variable assignment* α' *satisfies* $\alpha'(v) = \alpha(v)$ *if* $v \in V \setminus write(a)$*, and* $\alpha'(v) = \beta(v^w)$ *otherwise; (iii)* $\beta \models guard(a)$*, i.e., the guard is satisfied by* β*.*

Thus, the variable update works exactly as for the case of DPNs. For instance, $\mathcal{B}$ in Fig. 3 admits a step $(\mathsf{p}_0, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right]) \xrightarrow{\mathsf{init},\beta} (\mathsf{p}_{12}, \left[\begin{smallmatrix} t=1 \\ o=0 \end{smallmatrix}\right])$ where $\beta(t^r) = \beta(o^r) = \beta(o^w) = 0$ and $\beta(o^w) = 1$. Given a DDS $\mathcal{B}$, a *derivation* ρ of length n from a configuration (b, α) is a sequence of steps:

$$\rho\colon (b,\alpha) = (b_0,\alpha_0) \xrightarrow{a_1,\beta_1} (b_1,\alpha_1) \xrightarrow{a_2,\beta_2} \cdots \xrightarrow{a_n,\beta_n} (b_n,\alpha_n)$$

We also associate with ρ the *symbolic derivation* σ that *abstracts* ρ, i.e., the sequence $\sigma\colon b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} b_n$ where only the state and action sequences are recorded, but no concrete assignments are given. For some $m < n$, $\sigma|_m$ is the prefix of σ that has m steps. We call a *run* of $\mathcal{B}$ a derivation starting from (b_I, α_I), and a *symbolic run* a symbolic derivation starting from b_I. For instance,

$$\rho\colon (\mathsf{p}_0, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right]) \xrightarrow{\mathsf{init}} (\mathsf{p}_{12}, \left[\begin{smallmatrix} t=1 \\ o=0 \end{smallmatrix}\right]) \xrightarrow{\mathsf{timer}} (\mathsf{p}_{12}, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right]) \tag{2}$$

is a derivation of the DDS $\mathcal{B}$ from Fig. 3, and also a run because it starts in the initial state p_0; ρ is abstracted by the symbolic run $\mathsf{p}_0 \xrightarrow{\mathsf{init}} \mathsf{p}_{12} \xrightarrow{\mathsf{timer}} \mathsf{p}_{12}$. One may notice the similarity with the sequence of transition firings (1) in Sec. 2.

Transformation. It is straightforward to define the procedure DPNtoDDS($\mathcal{N}$) used in Alg. 1 to transform a given, bounded DPN $\mathcal{N}$ into a DDS. To this end, we consider in the rest of this section a k-bounded DPN $\mathcal{N} = \langle P, T, F, \ell, \mathcal{A}, V, \mathit{guard}\rangle$ with initial variable assignment α_I, initial marking M_I, and final marking M_F. We define DPNtoDDS($\mathcal{N}$) as the DDS $\mathcal{B} = \langle B, M_I, \mathcal{A}, \Delta, \{M_F\}, V, \alpha_I, \mathit{guard}\rangle$ where B is the set of all k-bounded markings of $\mathcal{N}$; and $(M, a, M') \in \Delta$ iff there is some $t \in T$ such that $\ell(t) = a$, $M(p) \geqslant F(p,t)$ for every p so that $F(p,t) \geqslant 0$ and $M'(p) = M(p) - F(p,t) + F(t,p)$ for every $p \in P$. Indeed, Fig. 3 shows the DDSs obtained for the DPNs $\mathcal{N}$, $\mathcal{N}_{\mathsf{reset}}$, and $\mathcal{N}_{\mathsf{thresh}}$ from Ex. 2.

After having defined the transformation from DPNs to DDSs, it remains to relate data-aware soundness of a DPN with properties of its DDS representation. To that end, we define some notions that turn out to be useful: A DDS $\mathcal{B} = \langle B, b_I, \mathcal{A}, \Delta, B_F, V, \alpha_I, \mathit{guard}\rangle$ has a *blocked state* if there is a run $\rho\colon (b_I, \alpha_I) \to^* (b, \alpha)$ to some configuration (b, α) such that there is no derivation $(b, \alpha) \to^* (b_f, \alpha')$ with $b_f \in B_F$. Moreover, let a state $b \in B$ be *reachable* if there is a run $(b_I, \alpha_I) \to^* (b, \alpha)$ for some α; and a transition $(b, a, b') \in \Delta$ be *reachable* if there is a run $(b_I, \alpha_I) \to^* (b, \alpha) \xrightarrow{a,\beta} (b', \alpha')$ for some α, α', and β. It is then not hard to observe the following relationship between the properties (P1), (P2), and (P3) in Def. 3 and properties of the DDS representation:

Lemma 1. *If $\mathcal{N}$ is a DPN and $\mathcal{B} =$ DPNtoDDS($\mathcal{N}$) has control states B,*
- *$\mathcal{N}$ satisfies (P1) iff $\mathcal{B}$ has no blocked states,*
- *$\mathcal{N}$ satisfies (P2) iff all $M \in B$ with $M \geqslant M_F$ are unreachable, and*
- *$\mathcal{N}$ satisfies (P3) iff for all transitions $t \in T$ of $\mathcal{N}$ there are some $M, M' \in B$ such that $(M, \ell(t), M') \in \Delta$ is reachable.*

This relationship allows us to check data-aware soundness on the level of DDSs. For instance, reset is not reachable in $\mathcal{B}_{\mathsf{reset}}$ as p_3 is only reached via hammer, i.e., if $o > 0$, so $\mathcal{N}_{\mathsf{reset}}$ does not satisfy (P3). Also, $\mathcal{B}_{\mathsf{thresh}}$ admits the run (3) below to

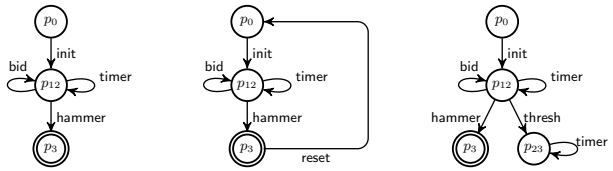


Fig. 3. DDSs $\mathcal{B}$, $\mathcal{B}_{\mathsf{reset}}$, and $\mathcal{B}_{\mathsf{thresh}}$ for DPNs $\mathcal{N}$, $\mathcal{N}_{\mathsf{reset}}$, and $\mathcal{N}_{\mathsf{thresh}}$.

state p_{23}, corresponding to marking $\{p_2, p_3\}$ in $\mathcal{N}_{\mathsf{thresh}}$, violating (P2). Finally, $\mathcal{B}$, $\mathcal{B}_{\mathsf{thresh}}$ and $\mathcal{B}_{\mathsf{reset}}$ have the blocked state $(\{\mathsf{p}_1, \mathsf{p}_2\}, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right])$, reachable via run (2).

$$\rho\colon (\mathsf{p}_0, \left[\begin{smallmatrix} t=0 \\ o=0 \end{smallmatrix}\right]) \xrightarrow{\mathsf{init}} (\mathsf{p}_{12}, \left[\begin{smallmatrix} t=1 \\ o=0 \end{smallmatrix}\right]) \xrightarrow{\mathsf{bid}} (\mathsf{p}_{12}, \left[\begin{smallmatrix} t=1 \\ o=1001 \end{smallmatrix}\right]) \xrightarrow{\mathsf{thresh}} (\mathsf{p}_{23}, \left[\begin{smallmatrix} t=1 \\ o=1001 \end{smallmatrix}\right]) \tag{3}$$

5 Constraint Graph

While numerical data and arithmetic are required to faithfully model processes in many real-life information systems, they render the state space infinite. For instance, the DDS $\mathcal{B}$ in Fig. 3 has infinitely many configurations such as $(\mathsf{p}_{12}, \left[\begin{smallmatrix} t=1 \\ o=5 \end{smallmatrix}\right])$, $(\mathsf{p}_{12}, \left[\begin{smallmatrix} t=2 \\ o=3 \end{smallmatrix}\right])$, and $(\mathsf{p}_{12}, \left[\begin{smallmatrix} t=0 \\ o=3 \end{smallmatrix}\right])$. However, not all state variable assignments differ with respect to possible next actions: action hammer requires $o > 0$ and $t \leqslant 0$, while bid and timer need $t > 0$; but it is irrelevant whether, say, $o > 4$. Therefore, $(\mathsf{p}_{12}, \left[\begin{smallmatrix} t=1 \\ o=5 \end{smallmatrix}\right])$ and $(\mathsf{p}_{12}, \left[\begin{smallmatrix} t=2 \\ o=3 \end{smallmatrix}\right])$ are indeed *equivalent* with respect to possible next steps, but the configurations $(\mathsf{p}_{12}, \left[\begin{smallmatrix} t=2 \\ o=3 \end{smallmatrix}\right])$ and $(\mathsf{p}_{12}, \left[\begin{smallmatrix} t=0 \\ o=3 \end{smallmatrix}\right])$ are not. Now, the key idea of the constraint graph is to symbolically represent equivalent configurations using a tuple (b, φ) of a control state b and a formula φ over variables V. For instance, for $\mathcal{B}$ we will distinguish $(\mathsf{p}_{12}, (o = 0) \wedge (t > 0))$ (both bid and timer apply) from $(\mathsf{p}_{12}, (o = 0))$ (we have no information about t, so only bid applies).

To formalize this idea, let $\mathcal{B} = \langle B, b_I, \mathcal{A}, \Delta, B_F, V, \alpha_I, \mathit{guard} \rangle$ be a given DDS. We start with some auxiliary notions: The *transition formula* Δ_a of action a is given by $\Delta_a(\overline{V}^r, \overline{V}^w) = \mathit{guard}(a) \wedge \bigwedge_{v \notin \mathit{write}(a)} v^w = v^r$. It simply expresses conditions on variables *before and after* executing the action: $\mathit{guard}(a)$ must hold, and the values of all variables that are not written are copied. E.g., for action bid in Fig. 3, we have $\mathit{write}(\mathsf{bid}) = \{o\}$, and $\Delta_{\mathsf{bid}} = (t^r > 0) \wedge (o^w > o^r) \wedge (t^w = t^r)$. Next, we use the transition formula to define an *update* operation, representing how a current state, captured by a formula φ, changes when executing action a.

Definition 6. *For a formula φ and action a, let $\mathit{update}(\varphi, a) = qe(\exists \overline{U}.\varphi[\overline{U}/\overline{V}] \wedge \Delta_a[\overline{U}/\overline{V}^r, \overline{V}/\overline{V}^w])$, 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 $\mathit{update}(\varphi, \mathsf{bid}) = qe(\exists o'\, t'.\ (t' > 0) \wedge (o' = 0) \wedge (o > o') \wedge (t = t'))$, which is simplified by quantifier elimination to $(t > 0) \wedge (o > 0)$. The use of a quantifier in Def. 6 might look like a complication, but it allows

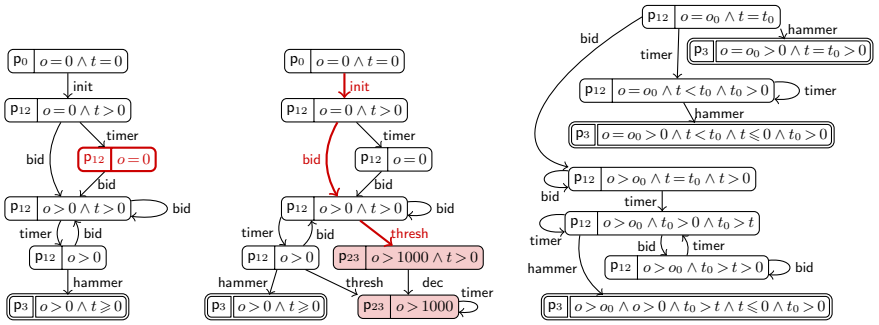


Fig. 4. Constraint graphs $\mathrm{CG}_{\mathcal{B}}$, $\mathrm{CG}_{\mathcal{B}_{\mathsf{thresh}}}$, and $\mathrm{CG}_{\mathcal{B}}(\mathsf{p}_{12})$.

us to remember the previous state φ, even if variables are overwritten by action a; afterwards, quantifier elimination can produce a logically equivalent formula without $\exists$. Next, given assignment α, let C_α be the formula $C_\alpha \doteq \bigwedge_{v \in V} v = \alpha(v)$.

Definition 7. *A* constraint graph $\mathrm{CG}_{\mathcal{B}}(b_0, \alpha)$ *for* $\mathcal{B}$*, a state* $b_0 \in B$*, and assignment* α *is a triple* $\langle S, s_0, \gamma \rangle$ *where the set of nodes* S *consists of tuples* (b, φ) *for* $b \in B$ *and a formula* φ*, and* $\gamma \subseteq S \times \mathcal{A} \times S$*, inductively defined as follows:*
(i) $s_0 = (b_0, C_\alpha) \in S$ *is the initial node; and*
(ii) if $(b, \varphi) \in S$ *and* $b \xrightarrow{a} b'$ *such that* $\mathit{update}(\varphi, a)$ *is satisfiable, there is some* $(b', \varphi') \in S$ *with* $\varphi' \equiv \mathit{update}(\varphi, a)$*, and* $(b, \varphi) \xrightarrow{a} (b', \varphi')$ *is in* γ*.*

Intuitively, the constraint graph describes symbolically the states reachable in $\mathcal{B}$. Specifically, we write $\mathrm{CG}_{\mathcal{B}}$ for the graph $\mathrm{CG}_{\mathcal{B}}(b_I, \alpha_I)$ starting at the initial state and the initial assignment. This is also the graph returned by the procedure COMPUTECG($\mathcal{B}$) used in Alg. 1. For instance, the first two graphs in Fig. 4 show $\mathrm{CG}_{\mathcal{B}}$ and $\mathrm{CG}_{\mathcal{B}_{\mathsf{thresh}}}$, respectively. Nodes that have the control state p_3 that is final in $\mathcal{B}$ are drawn with double border; the coloring will be explained later.

For technical reasons, our procedure often requires to consider constraint graphs that are built from an arbitrary state b and that, instead of assigning variables V to specific values, only impose that they have the same value of fresh placeholder variables V_0. We denote this by $\mathrm{CG}_{\mathcal{B}}(b)$. E.g., the rightmost graph in Fig. 4 shows $\mathrm{CG}_{\mathcal{B}}(\mathsf{p}_{12})$, representing the states reachable in $\mathcal{B}$ from p_{12} where $V = \langle o, t \rangle$ is initially assigned the placeholder variables $V_0 = \langle o_0, t_0 \rangle$.

We next establish properties that connect constraint graphs to derivations of the DDS $\mathcal{B}$. For a *path* $\pi\colon (b_0, \varphi_0) \xrightarrow{a_1} (b_1, \varphi_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} (b_n, \varphi_n)$ in a constraint graph, we denote by $\sigma(\pi)$ the symbolic derivation $b_0 \xrightarrow{a_1} b_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} b_n$ that has the same control state and action sequences. We now show that every combination of a path in a constraint graph and a satisfying assignment for the formula in its final node corresponds to a run in the DDS, and vice versa. To that end, we need a fixed variable renaming $\hat{\alpha}\colon V \mapsto V_0$.

Lemma 2. *(1)* $\mathrm{CG}_{\mathcal{B}}$ *has a path* $\pi\colon (b_I, C_{\alpha_I}) \to^* (b, \varphi)$ *where* φ *is satisfiable by* α*, iff* $\mathcal{B}$ *has a run* $(b_I, \alpha_I) \to^* (b, \alpha)$ *whose abstraction is* $\sigma(\pi)$*.*

Algorithm 2 Checking soundness properties for $\mathrm{CG}_{\mathcal{B}} = \langle S, s_0, \gamma \rangle$ and DPN $\mathcal{N}$

procedure BADTERMINATION($\mathrm{CG}_{\mathcal{B}}, \mathcal{N}$)
return $\exists (b, \varphi) \in S$ such that b corresponds to marking M in $\mathcal{N}$ and $M \geqslant M_F$

procedure DEADTRANSITION($\mathrm{CG}_{\mathcal{B}}, \mathcal{N}$)
return $\exists$ transition t of $\mathcal{N}$ such that $\nexists (s, \ell(t), s') \in \gamma$ for some s, s'

procedure BLOCKEDSTATE($\mathrm{CG}_{\mathcal{B}}, \mathcal{N}$)
return $\exists (b, \varphi) \in S$ such that $b \neq b_F$ and *blocked*(b, φ) satisfiable

(2) $\mathrm{CG}_{\mathcal{B}}(b)$ *has a path* $\pi \colon (b, C_{\widehat{\alpha}}) \to^* (b', \varphi)$ *s.t.* φ *is satisfiable by* α*, iff* $\mathcal{B}$ *has derivation* $(b, \alpha_0) \to^* (b', \alpha_n)$ *abstracted by* $\sigma(\pi)$ *with* $\alpha_0 = \alpha|_{V_0}$ *and* $\alpha_n = \alpha|_V$.

To illustrate this result, e.g. the run (3) corresponds to the path in $\mathrm{CG}_{\mathcal{B}_{\mathsf{thresh}}}$ shown in red (see Fig. 3). On the other hand, the lemma reveals that $\mathcal{B}$ has runs with the same action sequence for *all* assignments that satisfy $(o > 1000) \wedge (t > 0)$.

As stated above, the construction of the constraint graph according to Def. 7 need not terminate. However, it does in many practical examples, which is related to the following property identified in [9]: A DDS $\mathcal{B}$ has a *finite history set* if the set of formulas φ obtained during the construction of the constraint graph (called *history constraints* in [9]) is finite up to equivalence. Thus, if $\mathcal{B}$ has a finite history set, and the procedure COMPUTECG($\mathcal{B}$) checks eagerly for equivalent nodes while executing Def. 7, the construction must produce a finite graph. Crucially, this holds for a clearly identifiable class of systems used in the literature: it was shown that if either the constraint language in $\mathcal{B}$ is restricted to variable-to-variable and variable-to-constant comparisons, or if the control flow is such that the current state depends only on finitely many actions in the past, the DDS $\mathcal{B}$ has indeed a finite history set [9, Thms 5.2 and 5.9]. All examples of DPNs collected from the literature (see Sec. 7) fall in one of these categories.

6 Data-aware Soundness

In this section we harness the constraint graph to check data-aware soundness. To that end, we assume a DDS $\mathcal{B} = \langle B, b_I, \mathcal{A}, \Delta, \{b_F\}, V, \alpha_I, \mathit{guard} \rangle$ obtained by translating a DPN $\mathcal{N}$, such that $b_I = M_I$ and $b_F = M_F$ correspond to the initial and final markings of $\mathcal{N}$; and we assume that $\mathrm{CG}_{\mathcal{B}}$ is the constraint graph of $\mathcal{B}$. The three requirements of Def. 3 are then checked by the procedures in Alg. 2:

• BADTERMINATION returns *true* if in the node set S of the constraint graph $\mathrm{CG}_{\mathcal{B}}$ there is a node (b, φ) such that b corresponds to a marking M of the DPN $\mathcal{N}$ with $M \geqslant M_F$. For instance, it returns *true* for $\mathrm{CG}_{\mathcal{B}_{\mathsf{thresh}}}$ in Fig. 4 since the red nodes correspond to marking $\{\mathsf{p}_2, \mathsf{p}_3\}$; while it would return *false* for $\mathrm{CG}_{\mathcal{B}}$.

• DEADTRANSITION returns *true* if there is a transition in the DPN $\mathcal{N}$ whose label does not occur in $\mathrm{CG}_{\mathcal{B}}$. For instance, the constraint graph for the DDS $\mathcal{B}_{\mathsf{reset}}$ in Fig. 3 coincides with the graph $\mathrm{CG}_{\mathcal{B}}$ in Fig. 4, which does not contain reset. Thus, DEADTRANSITION($\mathrm{CG}_{\mathcal{B}}, \mathcal{N}_{\mathsf{reset}}$) returns *true*.

• For BLOCKEDSTATE, we use the formulas $\mathit{blocked}(b,\varphi)$ defined next. For $b \in B$ and constraint graph $\mathrm{CG}_{\mathcal{B}}(b) = \langle S', \gamma', s'_0 \rangle$, let $\mathit{final}(b) = \{\varphi \mid (b_F, \varphi) \in S'\}$ be all formulas in $\mathrm{CG}_{\mathcal{B}}(b)$ that occur together with final states. Then,

Definition 8. *For* $\mathrm{CG}_{\mathcal{B}} = (S, \gamma, s_0)$ *and* $(b,\varphi) \in S$*, let*

$$\mathit{blocked}(b,\varphi) = \varphi[\overline{V}/\overline{V}_0] \wedge \neg \left(\exists \overline{V}.\, \bigvee\nolimits_{\psi \in \mathit{final}(b)} \psi\right).$$

Now, BLOCKEDSTATE returns *true* if there is some node $(b,\varphi) \in S$ in $\mathrm{CG}_{\mathcal{B}}$ such that $\mathit{blocked}(b,\varphi)$ is satisfiable. This formula expresses that the process reaches control state b that prohibits to reach a final state: Indeed, $\Psi := \exists \overline{V}.\, \bigvee_{\psi \in \mathit{final}(b)} \psi$ states conditions to reach a final state from b and variables assigned to $\overline{V}_0$ (where $\exists \overline{V}$ reflects that we do not care about the final values of the data variables). Thus, $\neg\Psi$ states that *no* final state can be reached, and we take the conjunction with φ (with variables renamed appropriately) to combine this with the assumptions of the current constraint graph node (b,φ). For instance, we can check whether $\mathcal{B}$ in Fig. 3 admits a deadlock at a run the is captured by the node $(\mathtt{p}_{12}, o = 0)$ (drawn in red) of $\mathrm{CG}_{\mathcal{B}}$ in Fig. 4, as follows: There are three final nodes in $\mathrm{CG}_{\mathcal{B}}(\mathtt{p}_{12})$ in Fig. 4 labelled $\varphi_1 \doteq (o = o_0 \wedge o_0 > 0 \wedge t = t_0 \wedge t_0 > 0)$, $\varphi_2 \doteq (o = o_0 \wedge o_0 > 0 \wedge t_0 > t \wedge t \leqslant 0 \wedge t_0 > 0)$, and $\varphi_3 \doteq (o > o_0 \wedge o > 0 \wedge t_0 > t \wedge t \leqslant 0 \wedge t_0 > 0)$, so $\mathit{final}(\mathtt{p}_{12}) = \{\varphi_1, \varphi_2, \varphi_3\}$. We hence get

$$\mathit{blocked}(\mathtt{p}_{12}, o = 0) = (o_0 = 0) \wedge \neg\,(\exists o\, t.\, (\varphi_1 \vee \varphi_2 \vee \varphi_3))$$

which is simplified using quantifier elimination to $(o_0 = 0) \wedge (t_0 \leqslant 0)$, and e.g. satisfiable by $\alpha(o_0) = \alpha(t_0) = 0$. Thus BLOCKEDSTATE$(\mathrm{CG}_{\mathcal{B}}, \mathcal{N})$ returns *true*, reflecting the blocked sequence (1) shown at the end of Sec. 2.

Note that all checks in Alg. 2 are effective if $\mathrm{CG}_{\mathcal{B}}$ and all $\mathrm{CG}_{\mathcal{B}}(b)$ are finite. Finally, we relate the procedures in Alg. 2 to properties of $\mathcal{B}$, which together with Lem. 1 shows that data-aware soundness of DPNs is effectively checked.

Theorem 1. *Let* $\mathrm{CG}_{\mathcal{B}}$ *be a constraint graph for a DDS* $\mathcal{B}$*.*
(1) BLOCKEDSTATE$(\mathrm{CG}_{\mathcal{B}}, \mathcal{B})$ *returns true iff* $\mathcal{B}$ *has a blocked state.*
(2) DEADTRANSITION$(\mathrm{CG}_{\mathcal{B}}, \mathcal{B})$ *returns true iff* $\mathcal{N}$ *has a transition* t *of* $\mathcal{N}$ *such that* $(b, \ell(t), b') \in \Delta$ *is unreachable for all* $b, b' \in B$*, and*
(3) BADTERMINATION$(\mathrm{CG}_{\mathcal{B}}, \mathcal{B})$ *returns true iff some* $b \in B$ *corresponding to* M *with* $M \geqslant M_F$ *is reachable.*

7 Implementation and Experiments

We implemented our approach in the tool `ada` (arithmetic DDS analyzer) in Python; source code, benchmarks, and a web interface are available.[1] The tool takes a (bounded) DPN in `.pnml` format as input, and checks data-aware soundness following Alg. 1 and Alg. 2. As output, it produces graphical representations of the DDS $\mathcal{B}$ and the constraint graph $\mathrm{CG}_{\mathcal{B}}$, and if data-aware soundness is violated, a witness is constructed. CVC5 (cvc5.github.io) and Z3 (z3prover.github.io), which support all datatypes mentioned in Sec. 2.

[1] https://soundness.adatool.dev

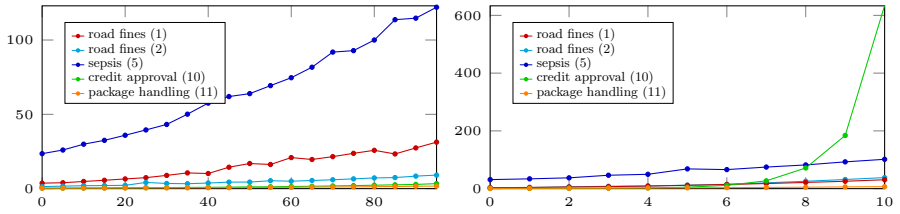
process		sound	time	checks	$\|V\|$	$\|\mathcal{B}\|$	$\|\mathbf{CG}_{\mathcal{B}}\|$
(1) road fines (normative)	[17, Fig. 7]	no P1	3.1s	3909	8	9/19	29/44
(2) road fines (mined)	[16, Fig. 12.7]	no P1	3.1s	3811	8	9/19	59/104
(3) road fines (mined)	[13, Fig. 13]	yes	2m16s	114,005	5	9/19	234/376
(4) hospital billing	[16, Fig. 15.3]	yes	3m1s	229,467	4	17/40	360/703
(5) sepsis (normative)	[16, Fig. 13.3]	yes	19s	831	3	301/1630	793/4099
(6) sepsis (mined)	[16, Fig. 13.6]	yes	1m43s	8085	4	301/1630	1117/5339
(7) digital whiteboard: register	[16, Fig. 14.3]	yes	0.1s	16	2	7/6	7/6
(8) digital whiteboard: transfer	[16, Fig. 14.3]	no P1	0.1s	19	3	7/6	7/6
(9) digital whiteboard: discharge	[16, Fig. 14.3]	yes	0.1s	30	4	6/6	7/6
(10) credit approval	[6, Fig. 3]	yes	1.2s	434	5	6/10	26/27
(11) package handling	[8, Fig. 5]	no P3	1.3s	242	5	16/28	68/67
(12) auction	[9, Ex. 1.1]	no P1	5.8s	1007	5	5/7	13/15

Table 1. Experiments with `ada` on DPNs from the literature.

As DPNs are a relatively recent framework, an extensive set of benchmarks is still missing. To mitigate this, we have collected all available DPN examples/use cases from the literature, and used `ada` to check soundness. The results are shown in Tab. 1, which indicates data-aware soundness (and the violated property of Def. 3), the verification time, number of SMT checks, number of variables in the DDS $\mathcal{B}$, and the sizes of $\mathcal{B}$ and $\mathrm{CG}_{\mathcal{B}}$ as number of nodes/transitions. All tests were run on an Intel Core i7 (4×2.60GHz, 19GB RAM), using CVC5 as backend. Benchmarks (1)–(3) model the handling of traffic offenses in an information system of the Italian police; in a normative model and two versions where decision rules were mined automatically. The former two have the same unsoundness issue (see Ex. 1), related to missing guards on written variables. (4) models the billing process in a hospital, it was mined from a real-life log with 100k traces, discovering guards by overlapping decision mining. (5) and (6) reflect the triage process for sepsis patients, based on a log obtained from a hospital's ERP system for 1,050 patients. (5) is a normative model; for (6), guards were discovered by decision mining. (7)–(9) are activity patterns for patient logistics designed based on domain knowledge and logs of a hospital information system. (10) is a faithful though hand-made process of granting loans to clients of a bank. (11) is a manually designed order-to-delivery process, obtained as a DPN translation of a DBPMN model (a data- and decision-aware model that builds on BPMN and DMN S-FEEL). (12) is a manually designed model for an English auction.

We stress that the benchmarks (1), (5), (7), (10), and (12) are out of reach of the earlier approaches [13,8], as their constraint language cannot express addition and multiplication. Moreover, while example (3) took 1.9h with the technique of [13], soundness can be detected by `ada` in less than 3 minutes.

An extensive DPN benchmark set with a wide range of problem sizes is not yet available. To provide some indications on the scalability of our method, we therefore modified some of the above benchmarks, adding (a) up to 100 sequential control states, and (b) up to 10 data variables $z_1, \ldots, z_k$ for every type, in the latter case obfuscating constraints of the form $e \odot e'$ to $e = z_1 \wedge z_1 = z_2 \wedge \cdots \wedge z_k \odot e'$. The results are depicted in Fig. 5, where the x-axis reports the number of added states/variables, and the y-axis the computation time.



(a) scalability: sequential control-flow (b) scalability: data variables/constraints

Fig. 5. Scalability of `ada` considering control-flow (a) and data variables (b).

The chart in (a) suggests that the addition of sequential tasks in the control-flow increases the computation time only linearly. For (b), we also observe a linear behaviour for many systems; but for benchmarks with a more complex constraint structure such as the credit approval example, performance can be considerably harmed. However, note that the benchmarks generated in (b) exhibit far larger constraints than the real-world systems, and can hence be considered extreme cases. Finally, it it interesting to observe that similar trends are obtained for (b) when using operators other than equality in building the expanded constraints.

8 Conclusion

The presence of numerical data in data-aware process models, either designed by hand or discovered from logs, render it highly intricate (undecidable in general) to manually check correctness properties such as soundness. We presented the first automatic technique that can verify data-aware soundness for DPNs with linear arithmetic, along with a prototype implementation. Our experiments show that the approach is effective and efficient, and can detect soundness bugs.

In future work, we aim at realizing a tighter integration between manual and automated approaches for data-aware process discovery and correctness analysis. Specifically, we plan to study the integration of this technique with automated approaches for process discovery to either guarantee by design the soundness of the discovered processes, or to provide specific indications on how to repair them. We also intend to deepen our understanding of the scalability of the approach starting from the preliminary evaluation presented here, with the goal of isolating the main sources of computational complexity, and of incorporating specific methods to handle them. Finally, we hope that having a solid foundational framework paired with a proof-of-concept IT artefact will trigger empirical research focussed on on-field validation of soundness for data-aware processes.

References

1. van der Aalst, W.: The application of Petri Nets to workflow management. Journal of Circuits, Systems, and Computers **8**(1), 21–66 (1998)

2. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects Comput. **23**(3), 333–363 (2011)
3. Artale, A., Calvanese, D., Montali, M., van der Aalst, W.M.P.: Enriching data models with behavioral constraints. In: Ontology Makes Sense. FAIA, vol. 316, pp. 257–277. IOS Press (2019)
4. Batoulis, K., Haarmann, S., Weske, M.: Various notions of soundness for decision-aware business processes. In: Proc. 36th ER. LNCS, vol. 10650, pp. 403–418 (2017)
5. Calvanese, D., de Giacomo, G., Montali, M.: Foundations of data-aware process analysis: a database theory perspective. In: Proc. 32nd PODS. pp. 1–12 (2013)
6. de Leoni, M., Mannhardt, F.: Decision discovery in business processes. In: Encyclopedia of Big Data Technologies, pp. 1–12. Springer (2018)
7. Deutsch, A., Hull, R., Li, Y., Vianu, V.: Automatic verification of database-centric systems. ACM SIGLOG News **5**(2), 37–56 (2018)
8. 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)
9. Felli, P., Montali, M., Winkler, S.: Linear-time verification of data-aware dynamic systems with arithmetic. In: Proc. 36th AAAI (2022)
10. Felli, P., Montali, M., Winkler, S.: Soundness of data-aware processes with arithmetic conditions (ext. version). CoRR (2022), https://arxiv.org/abs/2203.14809
11. Fettke, P., Reisig, W.: Modelling service-oriented systems and cloud services with heraklit. In: Proc. of Int. WSof ESOCC. CCIS, vol. 1360, pp. 77–89 (2020)
12. van Hee, K.M., Sidorova, N., Voorhoeve, M.: Generalised soundness of workflow nets is decidable. In: Proc. 25th ICATPN. LNCS, vol. 3099, pp. 197–215 (2004)
13. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. In: Proc. 37th ER. LNCS, vol. 11157, pp. 219–235 (2018)
14. de Leoni, M., Felli, P., Montali, M.: Strategy synthesis for data-aware dynamic systems with multiple actors. In: Proc. 17th KR. pp. 315–325 (2020)
15. de Leoni, M., Felli, P., Montali, M.: Integrating BPMN and DMN: modeling and analysis. J. Data Semant. **10**(1), 165–188 (2021)
16. Mannhardt, F.: Multi-perspective Process Mining. Ph.D. thesis, Technical University of Eindhoven (2018)
17. Mannhardt, F., de Leoni, M., Reijers, H., van der Aalst, W.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
18. Polyvyanyy, A., van der Werf, J.M.E.M., Overbeek, S., Brouwers, R.: Information systems modeling: Language, verification, and tool support. In: Proc. 31st CAiSE. LNCS, vol. 11483, pp. 194–212 (2019)
19. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du I congres de Mathem. des Pays Slaves. pp. 92–101 (1929)
20. Reichert, M.: Process and data: Two sides of the same coin? In: OTM 2012. LNCS, vol. 7565, pp. 2–19 (2012)
21. Ritter, D., Rinderle-Ma, S., Montali, M., Rivkin, A.: Formal foundations for responsible application integration. Inf. Syst. **101**, 101439 (2021)
22. Sidorova, N., Stahl, C.: Soundness for resource-constrained workflow nets is decidable. IEEE Trans. Syst. Man Cybern. Syst. **43**(3), 724–729 (2013)
23. Snoeck, M.: Enterprise Information Systems Engineering - The MERODE Approach. The Enterprise Engineering Series, Springer (2014)
24. Snoeck, M., De Smedt, J., De Weerdt, J.: Supporting data-aware processes with MERODE. In: Proc. 22nd BPMDS. LNBIP, vol. 421, pp. 131–146 (2021)