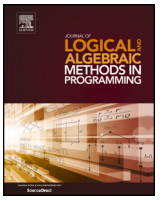
J. Log. Algebraic Methods Program. 138 (2024) 100953

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

journal homepage: www.elsevier.com/locate/jlamp



Verification of data-aware process models: Checking soundness of data Petri nets

Nikolai M. Suvorov *, Irina A. Lomazova

HSE University, Myasnitskaya ul. 20, Moscow, 101000, Russia

ARTICLE INFO

Keywords:
Verification of distributed processes with data
Formal models of distributed systems
Data Petri net
Data-aware soundness
Soundness verification

ABSTRACT

During recent years, significant research has been done in the direction of enriching the traditional control-flow perspective of processes with additional dimensions, such as data and decisions. To represent data-aware process models, various formalisms have been proposed. In this work, we focus on Data Petri nets (DPNs), an extension to a Petri net with data. Data in a DPN is set as variable values. Process activities, represented as transitions, can inspect and update variable values. This work is dedicated to soundness verification of data-aware process models represented as DPNs. We show the flaw in one of the algorithms for checking soundness of DPNs with variable-operator-variable conditions. The algorithm fails to detect some types of livelocks and, thus, is incorrect in the general case. In this report, we propose an advanced version of this algorithm, which correctly verifies soundness of DPNs and which can also be used for DPNs has composite conditions on transitions. To verify soundness, the algorithm refines a DPN by splitting some of its transitions, constructs an abstract state space of a refined DPN, and inspects it for soundness properties. The report justifies correctness of the proposed algorithm for DPNs with variables of real data type or any finite data types. The algorithm is implemented, and the results of its performance evaluation demonstrate practical applicability of the algorithm for process models of small and medium sizes.

1. Introduction

Distributed systems are becoming increasingly common in various domains, such as cloud computing, the Internet of Things, blockchain, and edge computing. These systems involve multiple nodes that interact and communicate with each other to perform complex tasks. Performance and reliability of these systems depend on the efficiency of their communication and coordination mechanisms. The analysis of distributed processes is crucial for designing and optimizing these systems to meet the increasing demands for scalability, reliability, and efficiency. The analysis helps to identify potential bottlenecks, performance issues, and failure points in the system.

A specific class of distributed processes is business processes. A business process consists of a set of activities that are performed in coordination in an organizational and technical environment and allow achieving a business goal [1]. Analysis of business processes allows finding inconsistencies and vulnerabilities in processes and provides information that may form a basis for making decisions to improve and optimize processes.

* Corresponding author.
E-mail address: nmsuvorov@hse.ru (N.M. Suvorov).

https://doi.org/10.1016/j.jlamp.2024.100953
Received 24 May 2023; Received in revised form 31 January 2024; Accepted 2 February 2024

Available online 7 February 2024

Soundness is one of correctness criteria for process models. A process model is sound if the process always properly terminates and each activity can occur in a process instance [2]. If a process model is unsound, the discussions that are based on this model can be misleading, the Process-Aware Information Systems may not function properly and the analysis of the model may provide incorrect results. Soundness verification is an important step at the design stage that allows detecting possible deadlocks and livelocks in the designed process. For soundness checking, multiple algorithms have already been introduced and implemented. The ready-to-use solutions can be found in PRoM framework [3] and PM4Py library [4]. Most of these algorithms require a process model to be represented as a Petri net.

A Petri net is a simple and unambiguous formalism that is commonly used for internal process model representation. Transitions in Petri nets represent process activities and markings (tokens in places) represent states. The initial marking depicts the initial state of the process, transition firing denotes activity execution and the final marking depicts the final state of the process. A workflow net (WF-net) is a Petri net that has the unique input and the unique output place [5]. WF-nets are used to describe the control flow of the process. Since properties of WF-nets can be faithfully and unambiguously analyzed, most process mining algorithms, including discovery, conformance checking, and enhancement algorithms, are designed specifically for WF-nets.

WF-nets are used to represent the control flow of the process, but the execution of real processes often depends on data. In particular, a process may contain decision points, where the choice between consequent activities is made based on the current state of process variables. To capture both the data flow and the control flow of the process, several extensions to Petri nets have been proposed. One such extension is a Data Petri Net (DPN) [6,7].

A DPN is a Petri net associated with a set of variables. Each state in a DPN is defined by a pair that includes a marking and a state of variables. DPN transitions are associated with constraints that define input and output conditions for variables. A transition may fire only if the input conditions hold in the current state of variables. If a transition has output conditions for some variables, then when the transition fires, the values of these variables are updated so that their new values satisfy the prescribed conditions. Several extensions to ProM Framework have been developed specifically for DPNs, including process discovery [7] and conformance checking [8] algorithms. The results of tests on synthetic and real-life event logs prove the practical applicability of this formalism for solving real tasks.

This work studies soundness verification of data-aware process models that can be represented as DPNs. Classical soundness, defined in [2], considers a state as a marking of a Petri net and determines correctness only of the control flow of the model. Since this approach ignores process data and conditions over it, some deadlocks and livelocks may remain undetected. In this work, as a correctness criterion for process models, we use *data-aware soundness* [9]. Data-aware soundness considers a state as a pair that includes a marking and a state of variables and, thus, considers both data flow and control flow perspectives of a process. Data-aware soundness is introduced specifically for integrated models, which can be represented as DPNs.

There exist several algorithms that are used to verify data-aware soundness of DPNs. Algorithm [9] is based on translating a DPN into a colored Petri net (CPN) and verifying soundness of the CPN. The algorithm can be used for DPNs, where each transition constraint is either an atomic condition or a condition composed of atomic ones using conjunction and/or disjunction. Atomic conditions can only be of type $(v\ p\ c)$, where v is a variable, p is a comparison operator and c is a constant. $a^r > 5$, $(b^r > 3) \wedge (c^w < 4)$, and $(c^w \leq 5) \vee (b^r > 5) \wedge (c^w > 5)$ are examples of transition constraints allowed by this algorithm. This algorithm has been extended in [10] for the case when atomic conditions have the form $(v\ p\ x)$, where v is a variable, p is a comparison operator and x is either a variable or a constant. However, this algorithm does not allow composite conditions as transition constraints: $a^w > b^r$, $b^r > 5$, $c^r > a^r$ are examples of transition constraints allowed by this algorithm. To verify soundness of a DPN, the algorithm constructs a state-transition structure called a *constraint graph* for a DPN and verifies specific data-aware soundness properties on this structure. Further, the latter algorithm has been used as a basis for the algorithm of verifying soundness of DPNs with arithmetic conditions [11].

In this work, we show that in some cases the algorithm proposed in [10] gives an incorrect answer. In Section 2, we present a DPN for which this algorithm fails to faithfully verify soundness. This paper uses the ideas of this algorithm as a basis and proposes a new correct solution for this task for the more general case. Specifically, we consider DPNs, where each transition constraint may be either an atomic condition or a condition composed of atomic ones using conjunction and/or disjunction. Each atomic condition is a condition of type $(v\ p\ x)$, where v is a variable, p is a comparison operator and x is either a variable or a constant. Conditions $a^r > 5$, $b^w > a^r$, $(c^w > 0) \wedge (b^w > c^r) \vee (c^w > b^w) \wedge (b^r < c^r) \wedge (c^r < 5)$ are examples of transition constraints that are allowed by our algorithm. We narrow our scope to DPNs with real and finite types of variables.

In this work, we introduce a labeled transition system (LTS) of a DPN, which is a generalization of a reachability graph such that each node represents not a single state of a DPN but a set of states with the same marking but different variable states. We show that the modification of the source DPN can make soundness verification using labeled transition systems and constraint graphs faithful. Specifically, we split transitions occurring in cycles and add τ-transitions, whose guards are negations of guards of the existing DPN transitions. In this regard, the existence of a livelock in a DPN always results in the presence of a cycle without an exit in the LTS, whereas the existence of a deadlock results in the presence of a dead node in the corresponding LTS. Then, the soundness verification procedure consists of five main steps: (1) checking boundedness of a DPN, (2) splitting DPN transitions occurring in cycles, (3) adding τ-transitions, (4) constructing an LTS for the resultant DPN, (5) analyzing the LTS for data-aware soundness properties.

We prove that the proposed algorithm terminates and returns the correct answer regarding data-aware soundness of DPNs with real and finite types of variables. The algorithm is implemented as a research prototype and the results of performance evaluation show that the algorithm is applicable for process models of small and medium sizes: for DPNs with less than 60 transitions, the algorithm generally requires less than 10 minutes to verify soundness, whereas for DPNs with less than 30 transitions, the algorithm usually requires less than 20 seconds to verify soundness.

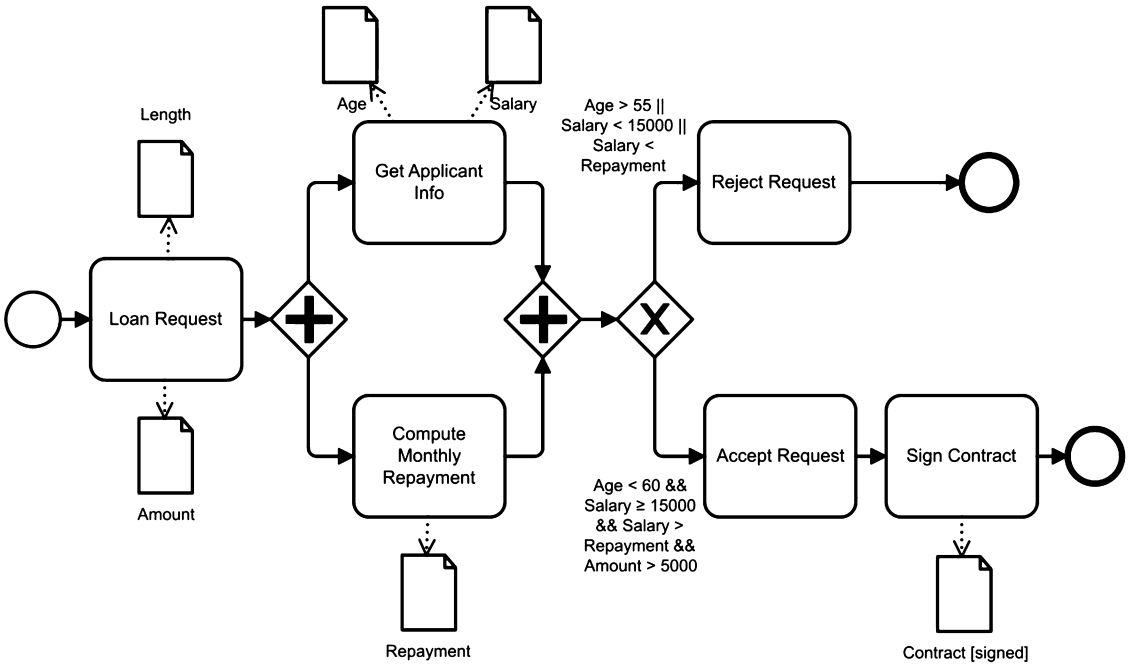


Fig. 1. BPMN model of a loan application process.

Our approach exploits a different background theory, comparing with existing soundness verification algorithms for DPNs, and, thus, represents a novel technique for soundness verification task. Our algorithm overcomes the problems of algorithm [10] and extends it allowing composite conditions as transition constraints. Comparing with algorithm [11] that correctly verifies soundness of DPNs with variable-operator-variable conditions, our algorithm does not require LTS construction for each DPN marking and, thus, usually allows to verify soundness quicker.

The main research contributions of this work are as follows:

1. A found mistake in the soundness verification algorithm presented in [10].
2. A redesigned definition of DPNs that allows composite conditions on transitions, where each atomic condition compares a variable either with another variable or with a constant.
3. A new data-aware soundness verification algorithm for DPNs with composite conditions.
4. Justification of the algorithm correctness for DPNs with variables of real and finite types.
5. A fully-fledged research prototype implementing the proposed algorithm.
6. Performance evaluation of the algorithm that justifies its practical applicability in realistic application domains.

This paper is organized as follows. Section 2 provides motivation and running examples. Section 3 defines a DPN and data-aware soundness. Section 4 describes the proposed algorithm for soundness verification of DPNs. Section 5 justifies algorithm termination for DPNs with variables of real and finite types. Section 6 provides a proof showing that the algorithm returns the correct answer regarding data-aware soundness for DPNs with variables of real and finite types. Section 7 describes cases when our algorithm may not be applicable. In particular, it demonstrates that for DPNs with variables of integer types, the algorithm may not terminate. Section 8 provides information on the algorithm's implementation, the conducted experiments, and their outcomes. Section 9 describes existing methods for data-aware soundness verification and compares them with the results achieved in this work. Section 10 concludes the paper.

2. Motivation and running examples

As we have mentioned in the introduction, the execution of real processes often depends on data. Fig. 1 shows an example of a loan application process, where data influences the process execution. In this process, the result of whether or not the customer's request is accepted is based on the values of such variables as salary, repayment, age, amount, and length. For instance, the loan request is rejected if the salary is less than the monthly repayment.

The process from Fig. 1 cannot be represented as a classical Petri net. This formalism can only capture conditions on activities if variables' domains are finite, but even in this case, the resulting Petri net significantly grows in size [12], which makes cumbersome the analysis of this net. In the case of infinite domains, it may be not possible to construct a finite Petri net, which makes this formalism inapplicable for modeling processes, where variables are defined over infinite domains. However, such processes can be represented as DPNs. Fig. 2 shows a DPN representation of the process from Fig. 1.

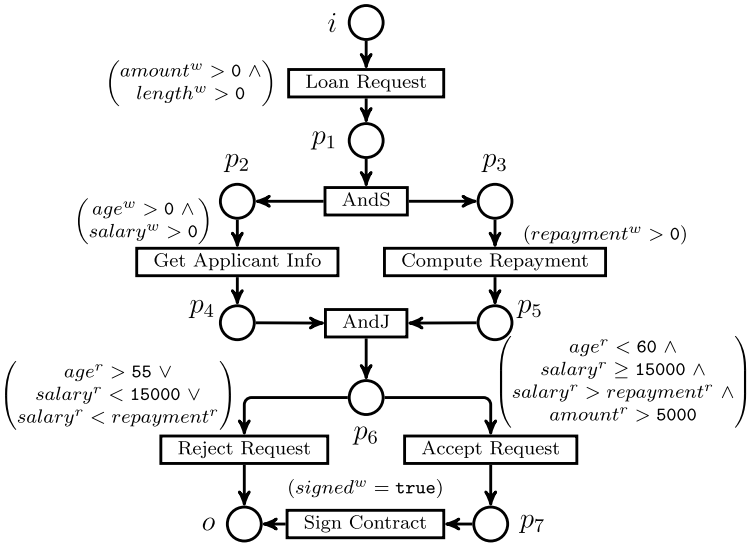


Fig. 2. Data Petri Net modeling a process from Fig. 1. $\alpha_I(amount) = 0, \alpha_I(length) = 0, \alpha_I(age) = 0, \alpha_I(salary) = 0, \alpha_I(repayment) = 0$ and $\alpha_I(ok) = \texttt{false}$. $M_I = [i]$, $M_F = [o]$.

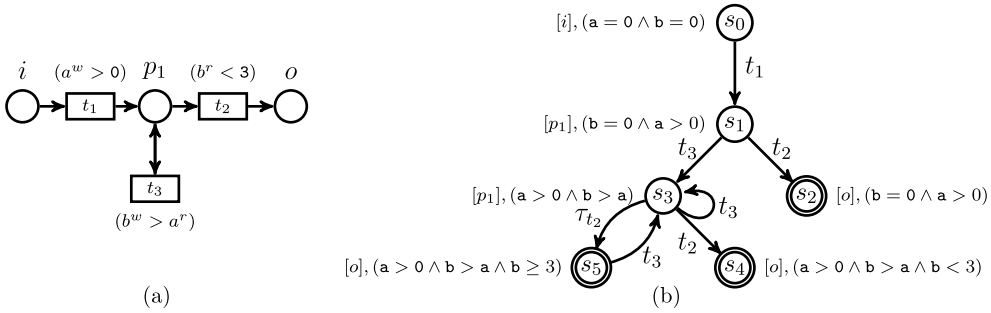


Fig. 3. Counterexample to the algorithm proposed in [10]. (a) DPN $\mathcal{N}$ for which the theorem from [10] does not hold. $M_I = [i]$ and $M_F = [o]$. Variables a and b are reals and $\alpha_I(a) = 0$ and $\alpha_I(b) = 0$. (b) Constraint graph for DPN $\mathcal{N}$. Final nodes are highlighted with double circles. According to [10], the constraint graph is data-aware sound.

Note that the illustrated process has a deadlock: if $amount < 5000$, $age \leq 55$, $salary \geq 15000$, and $salary \geq repayment$, then no activity can be chosen at the XOR-split. Thus, the process model is unsound. Detecting model unsoundness before the process implementation can help to avoid such unexpected behavior. However, the corresponding WF-net for this process does not have any deadlocks since the control flow is sound. To check soundness of this model, we can construct its DPN representation and verify data-aware soundness of this net.

This net corresponds to the setting of DPNs considered in algorithm [10] but also includes boolean connectives on transition constraints. In real processes, it is often the case when the choice at the decision point is made based on multiple conditions; thus, allowing composite conditions on DPN transitions seems relevant. Although we can represent the BPMN from Fig. 1 as a DPN with atomic conditions, this approach, in general case, can lead to a significant increase in the process model size making soundness verification unfeasible. Additionally, transforming a process model with composite conditions to a process model with atomic conditions, while maintaining soundness of the model, is often quite tricky. Thus, it would be very useful to have a soundness verification algorithm that can be straightforwardly applied for DPNs with composite conditions, without requiring prior net transformation.

At a glance, it can be a good idea to extend algorithm [10] to the case of logical expressions as transition constraints. However, during our research, we have found out that this algorithm is incorrect in the general case. Specifically, this algorithm fails to verify soundness of the DPN from Fig. 3. Although there exists a livelock, when t_1 sets $a \geq 3$ and t_3 fires at least once, each data-aware soundness property holds for the constraint graph of this DPN and, thus, the constraint graph is claimed to be data-aware sound. In [10], the authors introduce a theorem that states that the constraint graph for a DPN is data-aware sound if and only if the DPN is data-aware sound. Thus, according to this theorem, the DPN should be sound, but the DPN from Fig. 3 is not.

The problem in algorithm [10] is that livelocks in a DPN do not always correspond to cycles without an exit in a constraint graph as is seen in Fig. 3. In some cases, the corresponding cycles may have an exit although livelocks, which they represent, cannot be left. Specifically, this happens if some transition in a cycle updates a variable that is tested on a transition leading out of the cycle. In this example, the transition that updates a variable is t_3 and the transition that leads out of the cycle is t_2. Transition t_3 updates

the value of b and t_2 checks that the value of b is less than 3. This shows that the proposed state-space abstraction (a constraint graph) is, by itself, too rough to be used for soundness verification of the source DPN. However, our research shows that it is possible to make adjustments to this soundness verification algorithm to make it correct. This can be done by splitting DPN transitions that update variables in cycles before constructing a constraint graph. In this case, a constraint graph will have a cycle without exit for each livelock in a DPN. The repaired algorithm can then be extended to the case when a DPN has logical expressions as transition constraints. In this work, we introduce this extended algorithm and justify its correctness for DPNs with variables of real type or any finite type.

3. Data Petri nets

In this section, we illustrate how data-aware processes are represented by Data Petri nets (DPNs). We redesign the definition proposed in [10] to allow atomic and composite conditions as transition constraints.

DPNs are an extension of Petri nets with data. Data is set as variable values. Process activities, represented as transitions, can inspect and update variable values. We define a language of *constraints* capable of representing such input/output conditions imposed by process activities. The language described here is also used further to define a language of state constraints. As a set of variables in this definition, we consider an arbitrary set X that we further specify when defining concrete languages of constraints.

Definition 3.1 *(Language of constraints).* Let $Const$ be a set of constant symbols and $\mathcal{P}$ be a set of binary relation symbols. Let X be a set of variables. Language of constraints $\Phi(X)$ over variables from X is defined as follows:

- For $Q \in \mathcal{P}$ and $y, z \in (X \cup Const)$, $Q(y, z) \in \Phi(X)$.
- For $\phi \in \Phi(X)$, $\neg\phi \in \Phi(X)$.
- For $\phi_1, \phi_2 \in \Phi(X)$, $\phi_1 \vee \phi_2 \in \Phi(X)$.
- For $\phi_1, \phi_2 \in \Phi(X)$, $\phi_1 \wedge \phi_2 \in \Phi(X)$.

As an example, let $Const = \{1, 2, 3\}$, $\mathcal{P} = \{>, \geq, <, \leq, =, \neq\}$ and $X = \{y, z\}$. Then, $y < z$, $z \neq 3$, $(z > 1) \vee (z \leq 2) \wedge (y = 1)$, and $(y \geq 3) \wedge (z > y)$ are formulas of language $\Phi(X)$.

Let D be a domain of values. We suppose a fixed interpretation $I : Const \mapsto D$, $I : \mathcal{P} \mapsto f$, where $f : (D \times D) \mapsto \{\texttt{true}, \texttt{false}\}$. By abuse of notation, we write $c \in Const$ both for the constant symbol and its value. Similar for predicates from $\mathcal{P}$. Let $\beta : X \mapsto D$ be a state assigning a value from D to each variable from X. By $\mathcal{B} = \{\beta_1, \beta_2, ...\}$ we denote the set of all states. To extend a state to constants, for $\sigma : X \mapsto D$, we define $\tilde{\beta} : (X \cup Const) \mapsto D$ as follows:

- For $c \in Const$, $\tilde{\beta}(c) = I(c)$.
- For $x \in X$, $\tilde{\beta}(x) = \beta(x)$.

For a formula $\phi \in \Phi(X)$, we define its logical value in state β (denoted $\phi[\beta]$) as follows:

- $Q(y, z)[\beta]$ is true iff $Q(\tilde{\beta}(y), \tilde{\beta}(z))$ is true.
- $\neg\phi[\beta]$ is true iff $\phi[\beta]$ is false.
- $(\phi_1 \vee \phi_2)[\beta]$ is true iff $\phi_1[\beta]$ is true or $\phi_2[\beta]$ is true.
- $(\phi_1 \wedge \phi_2)[\beta]$ is true iff $\phi_1[\beta]$ is true and $\phi_2[\beta]$ is true.

We denote a set of states, in which the logical value of ϕ is true, as $[[\phi]] = \{\beta \in B | \phi[\beta]\}$. Then, two formulas $\phi_i, \phi_j \in \Phi(X)$ are logically equivalent (denoted $\phi_i \sim \phi_j$) iff $[[\phi_i]] = [[\phi_j]]$.

Now we come to define a language of constraints of DPNs. A state of a DPN is a pair that includes a marking and a variable state. Transitions in a DPN can transform variable states. A state transformer is defined declaratively by a transition constraint defining a non-deterministic transformation of the input state into the output state. A transition constraint is a formula of a language of constraints, which links the input and output values of variables. Let V be a set of variables of a DPN. Then, to distinguish between the input and output values of variables, for each variable $v \in V$, we additionally introduce symbol v^r for its input value, and symbol v^w for its output value. We consider $V^r \doteq \{v^r \mid v \in V\}$ and $V^w \doteq \{v^w \mid v \in V\}$. Then, the language of transition constraints is defined as $\Phi(V^r \cup V^w)$, while a value for each formula $\phi \in \Phi(V^r \cup V^w)$ is defined for $\beta : (V^r \cup V^w) \mapsto D$, which assigns a value to each read and written variable.

We define a DPN as follows:

Definition 3.2 *(Data Petri Net).* A DPN $\mathcal{N}$ is a tuple $(P, T, F, V, \Phi, guard)$, where:

- P is a set of places;
- T is a set of transitions;
- $F : (P \times T) \cup (T \times P) \mapsto \mathbb{N}$ is a flow relation;
- V is a finite set of variables;

- Φ is a language of constraints as defined above;
- $guard : T \to \Phi(V^r \cup V^w)$ is a function labeling transitions with constraints.

Given $t \in T$, we also define $read(t)$ and $write(t)$ to denote, respectively, the set of variables V^r and V^w that occur in $guard(t)$.

3.1. Execution semantics

To describe the execution semantics of DPNs, we first define a state in a DPN. Let $\mathcal{N}$ be a DPN. A state in $\mathcal{N}$ is a pair (M, α), where $M : P \mapsto \mathbb{N}$ is a marking of the net, and $\alpha : V \mapsto D$ is a variable state assigning a value from D to each variable $v \in V$. By $\mathcal{A} = \{\alpha_1, \alpha_2, ...\}$ we denote a set of all variable states.

For state $\beta : (V^r \cup V^w) \mapsto D$, which assigns a value to each read and written variable, we define input variable state $\beta|_{V^r} : V^r \mapsto D$ and output variable state $\beta|_{V^w} : V^w \mapsto D$. We introduce notation $\langle \beta|_{V^r}, \beta|_{V^w} \rangle$ that can be used interchangeably with β. Given two variable states α_1 and α_2, we use notation $\langle \alpha_1, \alpha_2 \rangle : (V^r \cup V^w) \mapsto D$ to denote a state, which assigns a value to each read and written variable, with distinguished input variable state α_1 and output variable state α_2.

Now we can define a firing rule:

Definition 3.3 *(Transition firing).* Given DPN $\mathcal{N}$ and some state (M, α), we say transition $t \in T$ may *fire* at (M, α) yielding a new state (M', α') iff:

- $\forall p \in P : M(p) \geq F(p, t)$,
- $\forall p \in P : M'(p) = M(p) - F(p, t) + F(t, p)$,
- $guard(t)[\langle \alpha, \alpha' \rangle]$ is true,
- $\forall v^w \in (V^w \setminus write(t)) : \alpha'(v) = \alpha(v)$.

We denote a transition firing by writing $(M, \alpha) \xrightarrow{t} (M', \alpha')$. We extend this definition to sequences $\sigma = \langle t^1, \dots, t^n \rangle$ of n transition firings, called *traces*, and denote the corresponding *run* by $(M^0, \alpha^0) \xrightarrow{t^1} (M^1, \alpha^1) \xrightarrow{t^2} \dots \xrightarrow{t^n} (M^n, \alpha^n)$ or equivalently by $(M^0, \alpha^0) \xrightarrow{\sigma} (M^n, \alpha^n)$. For a DPN $\mathcal{N}$ with initial state (M_I, α_I), we define the *runs of* $\mathcal{N}$ and *traces of* $\mathcal{N}$ as the set of runs and traces as above, of any length, such that $(M_I, \alpha_I) \xrightarrow{\sigma} (M, \alpha)$ for some marking M and variable state α.

Let $\mathcal{N}$ be a DPN with the initial state (M_I, α_I). The set of states reachable from (M_I, α_I) (the reachability set of $\mathcal{N}$, denoted $Reach_{\mathcal{N}}$) is the smallest set of states such that:

- $(M_I, \alpha_I) \in Reach_{\mathcal{N}}$; and
- if $(M, \alpha) \xrightarrow{t} (M', \alpha')$ with $t \in T$ and $(M, \alpha) \in Reach_{\mathcal{N}}$, then $(M', \alpha') \in Reach_{\mathcal{N}}$.

We now can define a *reachability graph* for a DPN:

Definition 3.4 *(Reachability graph of a DPN).* Let $\mathcal{N}$ be a DPN with initial state (M_I, α_I). The *reachability graph* of $\mathcal{N}$ (denoted $RG_{\mathcal{N}}$) is a graph $\langle V, E \rangle$, where:

- $V = Reach_{\mathcal{N}}$ is the set of reachable states of $\mathcal{N}$, each of the form (M, α); and
- $E \subseteq V \times T \times V$ is the set of arcs s.t. there exists arc $v \xrightarrow{t} v'$ in $RG_{\mathcal{N}}$ iff $v \in Reach_{\mathcal{N}}$, t is a transition that may fire at v, and v' is a state yielded by firing transition t at v.

A transition constraint in DPNs defines a non-deterministic transformation of an input state into an output state. Thus, a node in the reachability graph of $\mathcal{N} = (P, T, F, V, \Phi, guard)$ may have an infinite set of successors if the interpretation domain of Φ is infinite.

Recall that a classical Petri net is bounded if for each place $p \in P$, there exists a finite bound $k \in \mathbb{N}$ s.t. $M(p) \leq k$ for each reachable marking M. We extend this notion to DPNs. We say that DPN $\mathcal{N}$ is *bounded* if for each place $p \in P$, there exists a finite bound $k \in \mathbb{N}$ s.t. $\forall (M, \alpha) \in Reach_{\mathcal{N}} : M(p) \leq k$.

The example of DPN $\mathcal{N}$, representing a process of requesting a loan from a bank, is presented in Fig. 2. Initially, $amount, length, age, salary, repayment$ are equal to 0, and ok is equal to `false`. At (M_I, α_I), only *Loan Request* may fire. This transition firing updates the values of *amount* and *length*, so that the new values of these variables are greater than or equal to 0. After that, *And S* fires. Then, *Get Applicant Info* and *Compute Repayment* fire in any order. These transition firings update the values of *age*, *salary*, and *repayment*, making them greater than or equal to 0. After that, *And J* fires and the process meets a decision point. The choice between *Reject Request* and *Accept Request* is made based on the current values of variables $age, salary, repayment$, and *amount*. If *Reject Request* fires, the final marking is reached and the process terminates. If *Accept Request* fires, then the only transition that may fire is *Sign Contract* that assigns *signed* to `true`. Firing *Sign Contract* leads to the final marking. A fragment of the reachability graph for the DPN from Fig. 2 is depicted in Fig. 4. Although DPN $\mathcal{N}$ is bounded, the interpretation domain D of Φ is infinite, which is why the reachability graph of $\mathcal{N}$ is infinite.

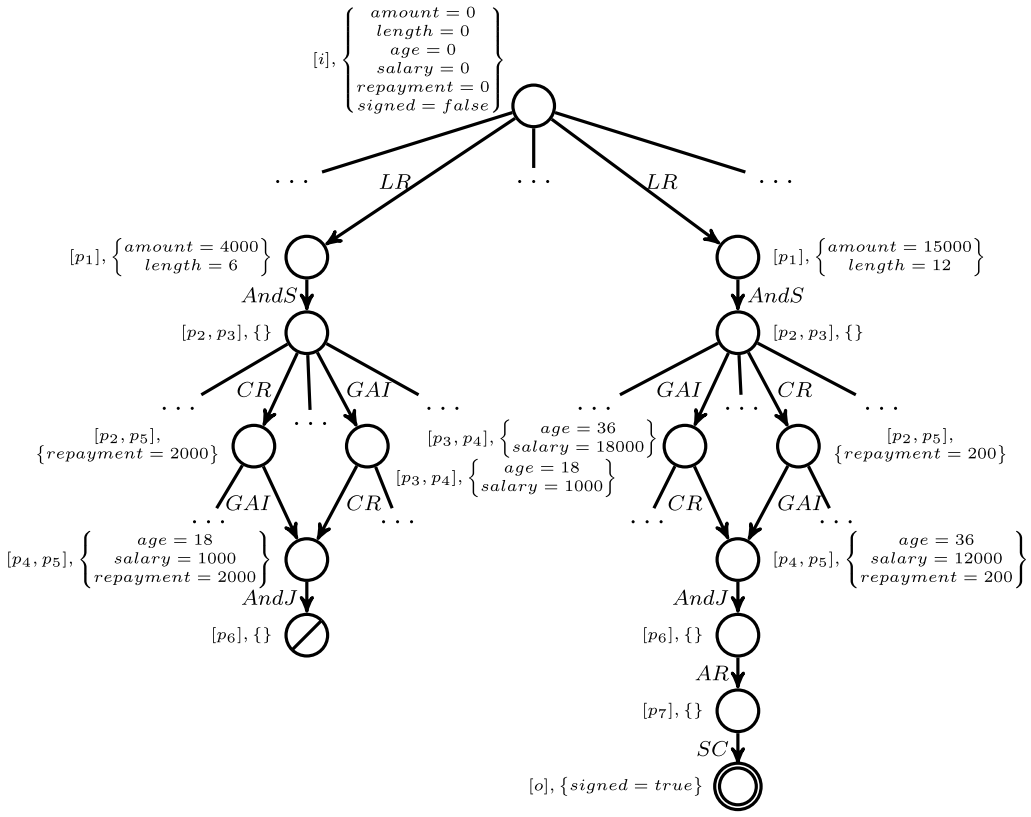


Fig. 4. A fragment of the reachability graph for $\mathcal{N}$ from Fig. 2. Arcs are labeled with the initials of the transition names. Variable states are not repeated for brevity: at each state, we only denote variable assignments performed by the preceding transition firings. For instance, after firing *Loan Request*, only *amount* and *length* are updated and, thus, only these variable assignments are denoted by the states resulted from executing this transition. Final nodes are highlighted with double circles. Deadlocks are highlighted with forbidden signs.

3.2. Data-aware soundness

DPNs can be constructed either manually or through various discovery techniques [13], but neither guarantees that the resulting model is sound. Soundness verification allows checking whether the process represented as a DPN always properly terminates and each process activity can occur in a process trace. As a correctness criterion that quantifies not only over the reachable markings of the net but also over the states of the variables, a notion of data-aware soundness has been proposed [9].

Given a DPN $\mathcal{N}$, in what follows we write $(M,\alpha)\xrightarrow{*}(M',\alpha')$ to mean that there exists a trace σ s.t. $(M,\alpha)\xrightarrow{\sigma}(M',\alpha')$ or that $(M,\alpha)=(M',\alpha')$. Also, given two markings M' and M'' of a DPN $\mathcal{N}$, we write $M''\geq M'$ iff for all $p\in P$ of $\mathcal{N}$ we have $M''(p)\geq M'(p)$, and we write $M''>M'$ iff $M''\geq M'$ and there exists $p\in P$ s.t. $M''(p)>M'(p)$. Now we can define data-aware soundness:

Definition 3.5 *(Data-aware soundness).* Let $\mathcal{N}$ be a DPN with initial state (M_I,α_I) and final marking M_F. Let $Reach_{\mathcal{N}}$ be a set of reachable states of $\mathcal{N}$. Then, $\mathcal{N}$ with initial state (M_I,α_I) and final marking M_F is *data-aware sound* iff all the following properties hold:

P1. $\forall(M,\alpha)\in Reach_{\mathcal{N}}\colon \exists\alpha'.\ (M,\alpha)\xrightarrow{*}(M_F,\alpha')$
P2. $\forall(M,\alpha)\in Reach_{\mathcal{N}}\colon M\geq M_F \Rightarrow (M=M_F)$
P3. $\forall t\in T.\ \exists M_1,M_2,\alpha_1,\alpha_2\colon (M_1,\alpha_1)\in Reach_{\mathcal{N}}$ and $(M_1,\alpha_1)\xrightarrow{t}(M_2,\alpha_2)$.

The first condition states the reachability of the output state. The second condition captures that the output state is always reached without having *in addition* other tokens in the net. The third condition requires the absence of dead transitions.

Referring to the example given in Fig. 2, DPN $\mathcal{N}$ is not data-aware sound since property $P1$ does not hold: if *Loan Request* assigns a value not greater than 5000 to *amount*, *Get Applicant Info* assigns a value greater than 15000 to *salary* and a value less than or equal to 55 to *age*, while *Compute Repayment* assigns a value that is less than the value of *salary* to *repayment*, then there appears a deadlock at $M=[p_6]$ as no transition may fire w.r.t. the current variable values and, thus, the output state cannot be reached. One of the traces leading to a deadlock is illustrated in the fragment of the reachability graph presented in Fig. 4.

4. Soundness verification of DPNs

In this section, we propose an algorithm that verifies data-aware soundness of DPNs with both atomic and composite conditions on transitions, where each atomic condition has a variable-operator-constant or variable-operator-variable form. The algorithm first checks a DPN for boundedness. If the DPN is unbounded, it returns $false$. If the DPN is bounded, the algorithm refines the DPN by splitting DPN transitions that update variable values and occur in cycles. Then, it adds τ-transitions to the refined DPN to further examine cases when the existing DPN transitions may not fire. Lastly, for the resultant DPN, a labeled transition system (LTS) is constructed and analyzed for soundness properties.

We start this section by introducing a labeled transition system of a DPN. A labeled transition system is a generalization of a *reachability graph* (see Definition 3.4), such that each node represents not a single state in a DPN but a set of states with the same marking but different variable states. Unlike the reachability graph, each node in the LTS has a finite set of immediate successors. Using state-space abstractions allows to tackle the infiniteness of states in a reachability graph. Similar approaches are used in [14] to verify soundness of Time Petri nets (TPNs) and in [10,11] to verify soundness of DPNs with atomic conditions and conjunctions of arithmetic conditions. Classically, a set of variable states (or states of clocks, if we talk about TPNs) is represented as a single constraint or a union of constraints. In our work, we use logical expressions to represent sets of variable states, which allows both conjunction and disjunction of constraints to be used as DPN transition conditions.

However, if we analyze an LTS for $\mathcal{N}$, some deadlocks and livelocks may not be detected. Thus, in our soundness verification algorithm, we first refine the DPN and add τ-transitions and only then construct and analyze the LTS. The labeled transition system of the resultant DPN reveals all the deadlocks and livelocks that occur in $\mathcal{N}$ allowing us to correctly verify soundness of $\mathcal{N}$.

Each DPN transition defines a non-deterministic transformation of the input variable state into the output variable state. To define an LTS of a DPN, we introduce an image function Im_t that maps input variable state α to a set of output variable states $Im_t(\alpha)$, where $Im_t(\alpha) = \{\alpha' \in \mathcal{A} | guard(t)[\langle \alpha, \alpha' \rangle]\}$. We also extend an image function to sets of variable states. Let A be some set of variable states and t be some transition. Then, an image of A under t is defined as $Im_t(A) = \{\alpha' | \alpha \in A \wedge \alpha' \in Im_t(\alpha)\}$.

Now we can define a labeled transition system (LTS) of DPN $\mathcal{N}$:

Definition 4.1 *(Labeled Transition System of a DPN).* Let $\mathcal{N} = (P, T, F, V, \Phi, guard)$ be a DPN. Let $\mathcal{M}_{\mathcal{N}}$ be a set of markings in $\mathcal{N}$ with initial state (M_I, α_I). Let $\mathcal{A}_{\mathcal{N}}$ be a set of variable states in $\mathcal{N}$. Labeled Transition System $LTS_{\mathcal{N}}$ of $\mathcal{N}$ is a tuple $\langle S, E, s_0 \rangle$, where:

- $S \subseteq \mathcal{M}_{\mathcal{N}} \times 2^{\mathcal{A}_{\mathcal{N}}}$ is a set of nodes;
- $E \subseteq S \times T \times S$ is a set of arcs labeled with transitions s.t. a triple $((M, A), t, (M', A')) \in E$ iff:
  - for each $p \in P$, $M(p) \geq F(p, t)$ and $M'(p) = M(p) - F(p, t) + F(t, p)$;
  - $A' = Im_t(A)$ and $A' \neq \emptyset$.
- $s_0 = (M_I, A_I) \in S$ is the initial node with $A_I = \{\alpha_I\}$.

If $((M, A), t, (M', A')) \in E$, we say that $((M, A), t, (M', A'))$ is a transition firing in $LTS_{\mathcal{N}}$. We denote a transition firing by writing $(M, A) \xrightarrow{t} (M', A')$ or $(M, A) \rightarrow (M', A')$. We extend this definition to sequences $\sigma = \langle t^1, \ldots, t^n \rangle$ of n transition firings, called *traces*, and denote the corresponding *run* by $(M^0, A^0) \xrightarrow{t^1} (M^1, A^1) \xrightarrow{t^2} \ldots \xrightarrow{t^n} (M^n, A^n)$ or equivalently by $(M^0, A^0) \xrightarrow{\sigma} (M^n, A^n)$. By $Reach(LTS_{\mathcal{N}})$ we denote a set of states reachable from s_0.

Given $LTS_{\mathcal{N}} = \langle S, E, s_0 \rangle$, we define $\mathfrak{A}(LTS_{\mathcal{N}}) = \{A | \exists M : (M, A) \in S\}$ as a set of sets of variable states in $LTS_{\mathcal{N}}$. Since sets of variable states in $\mathfrak{A}(LTS_{\mathcal{N}})$ may be infinite (i.e. in case of infinite domains), we need some finite representation of them. To work with sets of variable states, we introduce a language of *state constraints*. Let V be a set of variables of DPN $\mathcal{N}$. We define a language of state constraints for a labeled transition system of $\mathcal{N}$ as $\Phi(V)$. Each formula $\phi \in \Phi(V)$ represents conditions imposed on values of variables from V. We say that $LTS_{\mathcal{N}}$ is defined over Φ if each node in $LTS_{\mathcal{N}}$ is represented by a pair (M, ϕ), where M is a marking and $\phi \in \Phi(V)$ is a formula representing a set of variable states. In our algorithms, to tackle possible infiniteness of sets of variable states, we only consider $LTS_{\mathcal{N}}$ defined over Φ. Nonetheless, Definition 4.1 is put in general form to highlight the problem of language expressibility for representing sets of variable states, which we describe further. This also allows us to give termination and correctness proofs in general forms regardless of the language used.

In Figs. 5 and 6, we show examples of labeled transition systems defined over the DPN constraint languages. Fig. 5 illustrates $LTS_{\mathcal{N}}$ for DPN $\mathcal{N}$ from Fig. 2. Fig. 6 demonstrates $LTS_{\mathcal{N}}$ for DPN $\mathcal{N}$ from Fig. 3.

However, to describe sets of variable states in labeled transition systems and allow effective construction of $LTS_{\mathcal{N}}$, language Φ should be expressive enough. Let Φ be a language of constraints from Definition 3.1 interpreted over domain D. We extend language Φ by adding existential quantification, denoted by $\exists$, with standard semantics as in classical first-order logic and denote this language by $\tilde{\Phi}$. Let X be any finite set of variables. We call language Φ as *image-closed* if each of these properties hold:

- for each formula $\tilde{\phi} \in \tilde{\Phi}(X)$, a formula $\phi \in \Phi(X)$ that is logically equivalent to $\tilde{\phi}$ (i.e. $\phi \sim \tilde{\phi}$) is computable;
- $\phi \sim \phi'$ is decidable for any $\phi, \phi' \in \Phi(X)$;
- Φ includes equality relation.

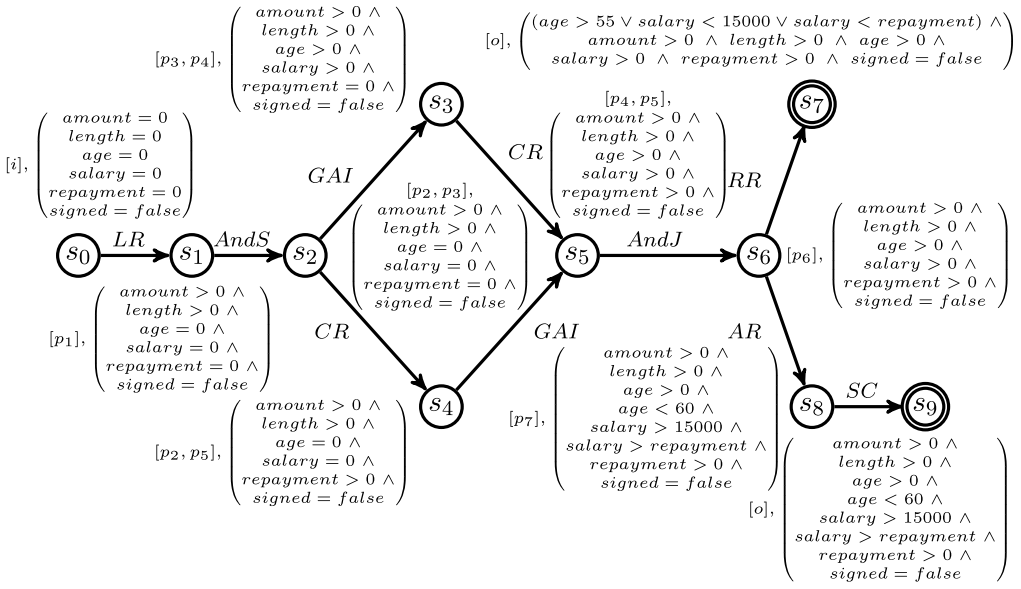


Fig. 5. A labeled transition system $LTS_{\mathcal{N}}$ constructed for DPN $\mathcal{N}$ from Fig. 2. Final nodes are highlighted by a double circle. Deadlocks are denoted by forbidden signs.

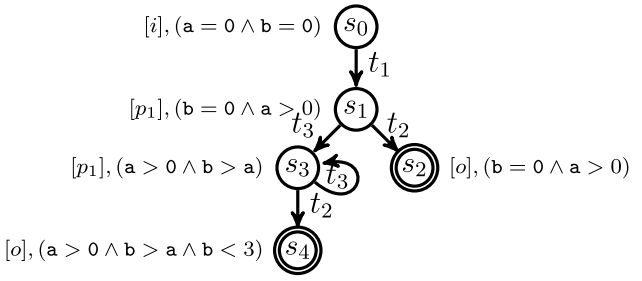


Fig. 6. A labeled transition system $LTS_{\mathcal{N}}$ constructed for DPN $\mathcal{N}$ from Fig. 3. Final nodes are highlighted by a double circle.

In practice, image-closeness of Φ means the decidability of quantifier elimination for formulas of $\tilde{\Phi}$ (in such a way that the formula resulted from the quantifier elimination is a formula of Φ) and the possibility of representing each DPN variable state α by some formula of Φ. If language Φ of DPN $\mathcal{N}$ is image-closed, then Φ is expressive enough to describe sets of variables states in $LTS_{\mathcal{N}}$. Furthermore, image-closeness of Φ allows staying within boundaries of Φ throughout the whole soundness verification procedure. In our work, we assume languages of constraints to be image-closed. An example of an image-closed language is language $\Phi^{\mathbb{R}}$ interpreted over $\mathbb{R}$ with $\mathcal{P} = \{>, \geq, <, \leq, =, \neq\}$. Language $\Phi^{\mathbb{R}}$ is also interesting due to the fact that for each $\tilde{\phi} \in \tilde{\Phi}^{\mathbb{R}}$, there always exists a quantifier-free formula $\phi \in \Phi^{\mathbb{R}}$, s.t. ϕ only has constants and variables that were present in $\tilde{\phi}$. The latter ensures that the set of formulas in $LTS_{\mathcal{N}}$ over $\Phi^{\mathbb{R}}$ is finite, which guarantees finiteness of the whole $LTS_{\mathcal{N}}$ when $\mathcal{N}$ is bounded.

When constructing $LTS_{\mathcal{N}}$, a set of successors is computed for each node. If Φ is image-closed and $LTS_{\mathcal{N}}$ is defined over Φ, there exists an effective procedure for calculating constraints for the successors. Let $s = (M, \phi)$ be an LTS node. Let $t \in T$ be some transition. Algorithm 1 defines a procedure, denoted $\phi \oplus guard(t)$, that computes a formula ϕ' representing the result of executing t at $[[\phi]]$. Specifically, $[[\phi']] = Im_t([[\phi]])$. If t may fire at M and $[[\phi']] \neq \emptyset$, then $s' = (M', \phi')$, where $\forall p \in P : M'(p) = M(p) - F(p,t) + F(t,p)$, is a successor of s, i.e. $s \xrightarrow{t} s'$.

Algorithm 1 initially converts ϕ_{state} to a formula of $\Phi(V^r \cup V^w)$ by substituting v for v^r, which is denoted by $[v/v^r]$ (lines 1-2). Then, ϕ_{united} is constructed as a conjunction of ϕ_{state} and ϕ_{trans} (line 3) and variables that must be overwritten are defined (line 4). After that, conditions on the overwritten variables are excluded by adding and, then, eliminating an existential quantifier on overwritten variables (lines 5). If Φ is image-closed, there exists an algorithm that effectively computes formula $\phi_{res} \in \Phi(V^r \cup V^w)$, s.t. $\phi_{res} \sim \exists V_{overwritten} : \phi_{united}$. We denote this algorithm by $EliminateQuantifiers$ (line 5). Lastly, the algorithm converts ϕ_{res} to a formula of $\Phi(V)$ by substituting v^r and v^w for v (lines 6-7).

Let $\mathcal{N} = (P, T, F, V, \Phi, guard)$ be a DPN with image-closed language Φ. A procedure that constructs $LTS_{\mathcal{N}}$ of $\mathcal{N}$ over Φ is shown in Algorithm 2. A node in $LTS_{\mathcal{N}}$ is represented by a pair (M, ϕ), where M is a marking and $\phi \in \Phi(V)$ is a formula representing a set of variable states. Set S represents a set of nodes of $LTS_{\mathcal{N}}$. Set E represents a set of arcs of $LTS_{\mathcal{N}}$. Set L contains nodes of $LTS_{\mathcal{N}}$ that still need to be processed. The algorithm explores the state space of the DPN (lines 6-15) and fills sets S, L, E. When

Algorithm 1: Procedure for computing $\phi_{res} \doteq \phi_{state} \oplus \phi_{trans}$.

Input: $\phi_{state} \in \Phi(V)$, $\phi_{trans} \in \Phi(V^r \cup V^w)$
Result: $\phi_{res} \in \Phi(V)$
1 **foreach** $v \in V$ **do**
2 $\quad \phi_{state} \leftarrow \phi_{state}[v/v^r]$
3 $\phi_{united} \leftarrow \phi_{state} \wedge \phi_{trans}$
4 $V_{overwritten} = \{v^r \mid v \in V \wedge Q(v^r, z) \in \phi_{united} \wedge Q(v^w, y) \in \phi_{united}\}$
5 $\phi_{res} \leftarrow EliminateQuantifiers(\exists V_{overwritten} : \phi_{united})$
6 **foreach** $v \in V$ **do**
7 $\quad \phi_{res} \leftarrow \phi_{res}[v^r/v][v^w/v]$
8 **return** ϕ_{res}

the whole state space is traversed, $LTS_{\mathcal{N}}$ is returned (line 16). In line 10, to compute a formula of a new node, Algorithm 2 calls $\oplus$-procedure, defined in Algorithm 1.

Algorithm 2: $ConstructLTS(\mathcal{N}, (M_I, \alpha_I))$.

Input: A DPN $\mathcal{N} = (P, T, F, V, \Phi, guard)$ with initial state (M_I, α_I) and final marking M_F.
Result: Labeled Transition System of $\mathcal{N}$.
1 $\phi_0 \leftarrow \bigwedge_{v \in V} \{v = \alpha_I(v)\}$,
2 $s_0 \leftarrow (M_I, \phi_0)$,
3 $S \leftarrow \{s_0\}$,
4 $E \leftarrow \emptyset$,
5 $L \leftarrow \{s_0\}$
6 **while** $L \neq \emptyset$ **do**
7 $\quad (M, \phi) \leftarrow$ `Pick(L) // Take a node from L`
8 $\quad L \leftarrow L \setminus \{(M, \phi)\}$
9 $\quad$ **foreach** $t \in T$ *s.t.* $M \xrightarrow{t} M'$ **do**
10 $\quad\quad \phi' \leftarrow \phi \oplus guard(t)$
11 $\quad\quad$ **if** $\neg(\phi' \sim false)$ **then**
12 $\quad\quad\quad E \leftarrow E \cup \{\langle (M, \phi), t, (M', \phi') \rangle\}$
13 $\quad\quad\quad$ **if** $\forall (\bar{M}, \bar{\phi}) \in S : \bar{M} \neq M' \vee \neg(\phi' \sim \bar{\phi})$ **then**
14 $\quad\quad\quad\quad S \leftarrow S \cup \{(M', \phi')\}$
15 $\quad\quad\quad\quad L \leftarrow L \cup \{(M', \phi')\}$
16 **return** $\langle S, s_0, A \rangle$

A labeled transition system for DPN $\mathcal{N}$ may be infinite. Specifically, it is a case when a set of reachable markings in $\mathcal{N}$ is infinite or when set $\mathfrak{A}(LTS_{\mathcal{N}})$ is infinite. To cope with the infiniteness of reachable markings, we consider a coverability graph. If $\mathfrak{A}(LTS_{\mathcal{N}})$ is finite, we can verify the finiteness of reachable markings on the coverability graph. If a coverability graph contains some strictly covering node, then $\mathcal{N}$ is unbounded and, therefore, the set of reachable markings in $\mathcal{N}$ is infinite.

To define a coverability graph of $LTS_{\mathcal{N}}$, we first define the quasi-ordering relation on states of $LTS_{\mathcal{N}}$:

Definition 4.2. Let S be a set of states in $LTS_{\mathcal{N}}$ for some DPN $\mathcal{N}$. We define quasi-ordering (qo) relation $\sqsubseteq$ on states S, s.t. for each $\langle M, A \rangle, \langle M', A' \rangle \in S$, $\langle M, A \rangle \sqsubseteq \langle M', A' \rangle$ iff $A = A'$ and $M \preceq M'$.

We say that node $\langle M', A' \rangle$ *covers* $\langle M, A \rangle$ if $\langle M, A \rangle \sqsubseteq \langle M', A' \rangle$. We say that $\langle M', A' \rangle$ *strictly covers* $\langle M, A \rangle$, denoted by $\langle M, A \rangle \sqsubset \langle M', A' \rangle$, if $A = A'$ and $M \prec M'$.

Now we can define the coverability graph:

Definition 4.3. (Coverability Graph of an LTS) Let $\mathcal{N}$ be a DPN. Let $LTS_{\mathcal{N}} = \langle S, E, s_0 \rangle$ be an LTS of $\mathcal{N}$. Let $\sqsubseteq$ be a quasi-ordering relation on S. Coverability graph $CG_{LTS_{\mathcal{N}}} = \langle S_{CG}, E_{CG}, s_0 \rangle$ is defined as follows:

- $S_{CG} \subseteq S$ is a set of nodes, where each node is either dead or live:
  - $s' \in S_{CG}$ is a dead node if s' does not have successors in $LTS_{\mathcal{N}}$ or there exists node $s \in S_{CG}$ along the path from s_0 to s', s.t. $s \sqsubset s'$ (i.e. s' strictly covers s);
  - $s' \in S_{CG}$ is a live node, otherwise.
- $E_{CG} \subseteq E$ is a set of arcs labeled with transitions, s.t. $(s, t, s') \in E_{CG}$ iff
  - $(s, t, s') \in E$;
  - s is a live node or the initial node.
- s_0 is the initial node.

When relation $\sqsubseteq$ is a well-quasi-ordering (wqo) [15], the set of reachable markings in $\mathcal{N}$ is finite if and only if $CG_{LTS_{\mathcal{N}}}$ does not have any strictly covering nodes. In Section 5, we show it by incorporating the theory of well-structured transition systems [16]. We also prove that $\sqsubseteq$ is a wqo if the DPN constraint language is $\Phi^{\mathbb{R}}$.

Algorithm 3: $CheckBoundedness(\mathcal{N}, (M_I, \alpha_I))$.

Input: A DPN $\mathcal{N} = (P, T, F, V, \Phi, guard)$ with initial state (M_I, α_I).
Result: Whether or not $\mathcal{N}$ is bounded.

```
1  φ_0 ← ⋀_{v∈V} {v = α_I(v)},
2  s_0 ← (M_I, φ_0),
3  S_CG ← ∅
4  E_CG ← ∅,
5  L ← {s_0}
6  while L ≠ ∅ do
7      (M, φ) ← pick(L) // Take a node from L
8      L ← L \ {(M, φ)}
9      foreach t ∈ T s.t. M -t-> M' do
10         φ' ← φ ⊕ guard(t)
11         if ¬(φ' ~ false) then
12             foreach state (M_p, φ_p), from which (M, φ) is reachable do
13                 if (M' > M_p) ∧ (φ' ~ φ_p) then
14                     return false
15             E_CG ← E_CG ∪ {⟨(M, φ), t, (M', φ')⟩}
16             if ∀(M̄, φ̄) ∈ S_CG : M̄ ≠ M' ∨ ¬(φ' ~ φ̄) then
17                 S_CG ← S_CG ∪ {(M', φ')}
18                 L ← L ∪ {(M', φ')}
19 return true
```

Algorithm 3 verifies boundedness of DPN $\mathcal{N}$ by constructing a coverability graph of $LTS_{\mathcal{N}}$ over image-closed language Φ. Each node in the coverability graph is represented by a pair (M, ϕ), where M is a marking and $\phi \in \Phi(V)$ is a formula describing a set of variable states. We use the set of nodes S_{CG} and the set of edges E_{CG} to store the graph (lines 3-4). Set L contains vertices that must be expanded (line 5). The algorithm tries to expand each vertex of the graph by firing transitions from T (lines 9-18). If some strictly covering node is found, the algorithm terminates and returns `false` (lines 13-14). Otherwise, the algorithm continues the exploration of an abstract state space of a DPN until the whole state space is traversed. If no strictly covering node is found, the algorithm returns `true` meaning that the DPN is bounded (line 19). In line 10, to compute a formula of a new node, Algorithm 3 calls $\oplus$-procedure, defined by Algorithm 1.

Now we define a *refined DPN*, denoted by $\mathcal{N}_R$, as a result of executing Algorithm 4. Algorithm 4 refines a DPN by splitting transitions occurring in cycles. By construction, $\mathcal{N}_R$ is behaviorally equivalent to $\mathcal{N}$ (their reachability graphs are equivalent, see Proposition 6.1), but refining a DPN allows a more subtle investigation of execution scenarios using labeled transition systems. This is due to the fact that each trace of $LTS_{\mathcal{N}}$ is represented by a set of traces in $LTS_{\mathcal{N}_R}$, which is a result of splitting DPN transitions in $\mathcal{N}_R$. It is the refinement step that allows us to further capture DPN livelocks as cycles without an exit on the LTS.

To describe the algorithm, we first introduce partial function R that maps transition t to a set $R(t)$ of refined transitions obtained by splitting transition t. Initially, R is not defined for all $t \in T$. The algorithm, in a loop, refines DPN transitions by constructing $R(t)$ and substituting t for $R(t)$ for each DPN transition t until the refinement stops producing any new transitions.

Algorithm 4: $RefineDPN(\mathcal{N}, (M_I, \alpha_I))$.

Input: A DPN $\mathcal{N} = (P, T, F, V, \Phi, guard)$ with initial state (M_I, α_I).
Result: $\mathcal{N}_R = (P, T_R, F_R, V, \Phi, guard)$ for $\mathcal{N}$.

```
1  toProceed = true
2  while toProceed do
3      ⟨S, s_0, E⟩ = ConstructLTS(N, (M_I, α_I))
4      Cycles = GetMaxDisjointCycles(⟨S, s_0, A⟩)
5      foreach t ∈ T do
6          Cycles_t ← FilterCyclesByTransition(Cycles, t)
7          T_out ← GetTransitionsLeadingOutOfCycles(Cycles_t)
8          R(t) ← RefineTransition(t, T_out)
9          foreach t_r ∈ R(t) do
10             foreach p ∈ P do
11                 F_R(p, t_r) ← F(p, t)
12                 F_R(t_r, p) ← F(t, p)
13     T_R ← ⋃_{t∈T} R(t)
14     toProceed ← IsModifiedAtCurrentIteration(N)
15     N ← (P, T_R, F_R, V, Φ, guard)
16 foreach t ∈ T do
17     R(t) ← Flatten(R(t))
18 N_R ← N
19 return N_R
```

Inside the loop, the algorithm constructs $LTS_{\mathcal{N}}$ of $\mathcal{N}$ (line 3), determines maximum disjoint cycles in $LTS_{\mathcal{N}}$ (line 4), and tries to refine each transition $t \in T$ (lines 5-12). In line 4, procedure $GetMaxDisjointCycles$ finds maximal mutual disjoint cycles in $LTS_{\mathcal{N}}$.

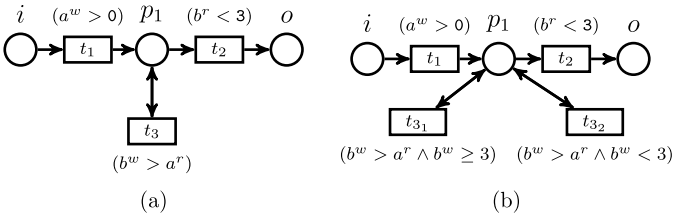


Fig. 7. Demonstration of DPN refinement. (a) $\mathcal{N}$ from Fig. 3. (b) Refined DPN $\mathcal{N}_R$ for $\mathcal{N}$. t_{3_1} and t_{3_2} are transitions resulted from splitting t_3 based on input condition of t_2.

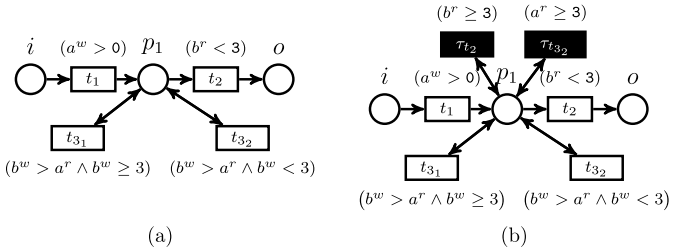


Fig. 8. Demonstration of constructing τ-DPN. (a) $\mathcal{N}_R$ from Fig. 7. (b) $\mathcal{N}_{R_\tau}$ for $\mathcal{N}_R$. τ_{t_2} and $\tau_{t_{3_2}}$ are τ-transitions for t_2 and t_{3_2}, respectively.

We call two cycles disjoint if they do not have a common vertex. A cycle is called maximal if it is disjoint with all other cycles. Each elementary cycle (a cycle that does not pass through the same vertex twice) in $LTS_{\mathcal{N}}$ is included in a single maximal disjoint cycle constructed by this procedure. To refine transition t, we search for transitions that allow leaving cycles containing t (line 6). For these purposes, we first find all the maximal disjoint cycles that include transition t by executing $FilterCyclesByTransition$ and second detect transitions that allow leaving these cycles by executing $GetTransitionsLeadingOutOfCycles$. The detected transitions are saved into T_{out} (lines 6-7). If $T_{out} = \emptyset$, $R(t) = t$. Otherwise, based on input conditions of transitions from T_{out}, set $R(t)$ of the refined transitions that replace transition t is constructed (line 8). The transition refinement produces a set of transitions $R(t)$, s.t. firing each $t_r \in R(t)$ yields a variable state that satisfies input conditions of the *unique* set of transitions that lead out of the cycles containing t. A guard of each $t_r \in R(t)$ is a conjunction of $guard(t)$ and input conditions or negations of input conditions of transitions from T_{out}. When set $R(t)$ is constructed, arcs connecting each transition from $R(t)$ to places adjacent to t are constructed (lines 9-12). At the end of the iteration, a set of transitions T_R is constructed as the union of $R(t)$ for each $t \in T$ (line 13). If the current iteration does not change a DPN, which is checked by $IsModifiedAtCurrentIteration$, the algorithm returns the resultant net as a refined DPN (lines 18-19); otherwise, a new loop iteration starts (line 2).

The example of constructing $\mathcal{N}_R$ for $\mathcal{N}$ from Fig. 3 is presented in Fig. 7. Transition t_3 is split into t_{3_1} and t_{3_2} based on the input condition on variable b of transition t_2. Note that t_3 updates variable b. Adding any input conditions on variable b to t_3 is meaningless since these constraints are reset after firing t_3. Thus, instead of adding $(b^r \geq 3)$ or $(b^r < 3)$ to the constraints of the split transitions, we add $(b^w \geq 3)$ to the constraint of t_{3_1} and $(b^w < 3)$ to the constraint of t_{3_2}.

Note that DPN $\mathcal{N}$ from Fig. 2 does not have cycles. This entails that neither of the transitions in $\mathcal{N}$ is split during the refinement procedure and, thus, $\mathcal{N}_R$ is equal to $\mathcal{N}$.

Now, we define *tau-DPN* $\mathcal{N}_\tau$, which is constructed by supplementing DPN $\mathcal{N}$ with τ-transitions. Each τ-transition τ_t with $t \in T$ corresponds to the case when $guard(t)$ does not hold in the current variable state. Thus, $guard(\tau_t)$ is set to $\neg(\exists write(t) : guard(t))$. If $Im_t(\alpha) = \emptyset$ for some α, then $Im_{t_\tau}(\alpha) = \alpha$. Given some set of variable states A, $Im_{t_\tau}(A) = \{\alpha \in A | Im_t(\alpha) = \emptyset\}$. If language Φ of DPN $\mathcal{N}$ is image-closed, the constraint of each transition of $\mathcal{N}_\tau$ can be expressed as a formula of Φ. Constructing LTS for $\mathcal{N}_{R_\tau}$ allows investigation of cases when particular transitions may not fire and, by that, detect deadlocks and livelocks in the net. We define a tau-DPN as follows:

Definition 4.4 *(Tau-DPN).* Let $\mathcal{N} = (P, T, F, V, \Phi, guard)$ be a DPN. Tau-DPN $\mathcal{N}_\tau$ is a tuple $(P, T \cup T_\tau, F \cup F_\tau, V, \Phi, guard)$, where:

- $T_\tau = \{\tau_t | t \in T\}$;
- $F_\tau(p, \tau_t) = F_\tau(\tau_t, p) = F(p, t)$ for each $\tau_t \in T_\tau, p \in P$;
- $guard(\tau_t) = \neg(\exists write(t) : guard(t))$ for each $\tau_t \in T_\tau$.

Fig. 8 illustrates $\mathcal{N}_{R_\tau}$ constructed for $\mathcal{N}_R$ from Fig. 7. τ-transitions are only added for t_2 and t_{3_2} since other transitions do not have input conditions. Transition t_2 has an input condition $(b^r < 3)$. The negation of this condition, namely $(b^r \geq 3)$, becomes a guard of τ_{t_2}. Transition t_{3_2} has an implicit input condition $(a^r < 3)$. Note that if this condition does not hold, t_{3_2} can never fire. The negation of this condition, namely $(a^r \geq 3)$, becomes a guard of $\tau_{t_{3_2}}$.

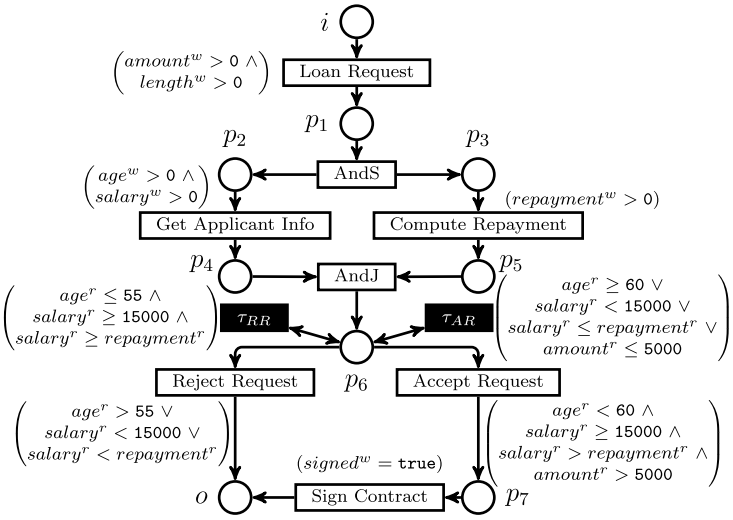


Fig. 9. $\mathcal{N}_\tau$ for DPN $\mathcal{N}$ from Fig. 2. τ_{RR} represents a τ-transition for *Reject Request*. τ_{AR} represents a τ-transition for *Accept Request*.

Fig. 9 shows $\mathcal{N}_\tau$ constructed for $\mathcal{N}$ from Fig. 2. Recall that for this DPN, $\mathcal{N}_R$ is equal to $\mathcal{N}$; thus $\mathcal{N}_\tau$ equals to $\mathcal{N}_{R_\tau}$. τ-transitions are only added for *Reject Request* and *Accept Request* since only these transitions have input conditions on variable values. The constraints of these transitions contain only input conditions on variables. Thus, to obtain the guard of the corresponding τ-transition, we only need to negate the constraint of the source transition.

Algorithm 5 presents a procedure to verify data-aware soundness of DPN $\mathcal{N}$. The algorithm first checks boundedness of $\mathcal{N}$ (line 1). If the net is unbounded, `false` is returned (line 2). If the net is bounded, the algorithm refines DPN $\mathcal{N}$ (line 3). For $\mathcal{N}_R$, $\mathcal{N}_{R_\tau}$ is constructed according to the constructive Definition 4.4 (line 4). Lastly, the algorithm constructs the labeled transition system for $\mathcal{N}_{R_\tau}$ (line 5) and analyzes it for soundness properties (line 6). Verification of properties P1-P3 as in Definition 3.5 is done by function $AnalyzeLTS$ that executes search procedures on this finite-state graph: from each node, a final node must be reachable (P1), there must exist no node with $M \succ M_F$ (P2), and for each $t \in T$, there must exist a reachable (from the initial node) transition firing of at least one $t_r \in R(t)$ (P3).

Fig. 10 presents an LTS constructed for $\mathcal{N}_{R_\tau}$. Note that this LTS has node s_6 from which no final node is reachable and, thus, P1 from Definition 3.5 does not hold for this LTS. Specifically, this node represents a livelock that $\mathcal{N}$ meets at p_0 when $b > a \wedge a \geq 3 \wedge b \geq 3$.

Algorithm 5: $CheckSoundnessDirect(\mathcal{N}, (M_I, \alpha_I), M_F)$.

Input: A DPN $\mathcal{N} = (P, T, F, V, \Phi, guard)$ with initial state (M_I, α_I) and final marking M_F.
Result: Whether or not $\mathcal{N}$ is data-aware sound.

```
1 if CheckBoundedness(N, (M_I, α_I)) = false then
2 |   return false
3 N_R = RefineDPN(N, (M_I, α_I))
4 N_Rτ = GetTauDPN(N_R)
5 LTS_{N_Rτ} = ConstructLTS(N_Rτ, (M_I, α_I))
6 return AnalyzeLTS(LTS_{N_Rτ}, M_F)
```

In the universe of classical Petri nets, most of the nets are unsound. The same applies to Data Petri nets. However, some data-aware soundness properties can be checked without refining a DPN. Such properties include the absence of dead transitions, markings that strictly cover M_F, and deadlocks. DPN refinement is a time-consuming procedure and, thus, it may be purposeful to postpone the refinement of a DPN to a case when the DPN is verified to have no deadlocks, markings strictly covering M_F, and dead transitions.

Algorithm 6 presents an improved data-aware soundness verification algorithm, where the refinement procedure is performed only when a DPN is verified to have no deadlocks, dead transitions, and markings strictly covering M_F. The first step of the algorithm is to check boundedness of the net (lines 1-2). If the net is bounded, LTS for DPN $\mathcal{N}$ is constructed (line 3). The algorithm checks the absence of dead transitions and markings strictly covering M_F on this LTS (line 4). If there are none, $\mathcal{N}_\tau$ is constructed (line 5). For $\mathcal{N}_\tau$, the LTS is constructed (line 6). The algorithm verifies the absence of deadlocks on this LTS (line 7). If DPN $\mathcal{N}$ contains a deadlock, the algorithm returns `false` on this step. If these preliminary checks do not show that DPN $\mathcal{N}$ is unsound, the refinement procedure is executed (line 8). Since $LTS_\mathcal{N}$ has already been constructed, the step of constructing the LTS of $\mathcal{N}$ during the refinement can be omitted. After the refinement, $\mathcal{N}_{R_\tau}$ for $\mathcal{N}_R$ is constructed (line 9), and the absence of livelocks is checked on $LTS_{\mathcal{N}_{R_\tau}}$ (line 10-11). Procedures $NoNodesStrictlyCoveringFinal$, $NoDeadTransitions$, $NoDeadlocks$, and $NoLivelocks$ use graph traversing techniques similar to those described for Algorithm 5.

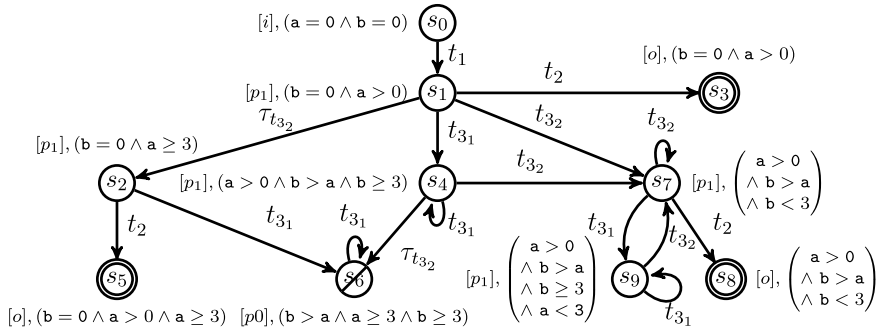


Fig. 10. $LTS_{\mathcal{N}_{R_\tau}}$ constructed for $\mathcal{N}$ from Fig. 7. Loops with τ-transitions are omitted for brevity. Double circles denote final states. Forbidden signs denote states from which no final state is reachable.

Algorithm 6: $CheckSoundnessImproved(\mathcal{N}, (M_I, \alpha_I), M_F)$.

Input: A DPN $\mathcal{N} = (P, T, F, V, \Phi, guard)$ with initial state (M_I, α_I) and final marking M_F.
Result: Whether or not $\mathcal{N}$ is data-aware sound.

```
if CheckBoundedness(N, (M_I, α_I)) = false then
    return false
LTS_N = ConstructLTS(N, (M_I, α_I))
if NoNodesStrictlyCoveringFinal(LTS_N, M_F) ∧ NoDeadTransitions(LTS_N) then
    N_τ = GetTauDPN(N)
    LTS_{N_τ} = ConstructLTS(N_τ, (M_I, α_I))
    if NoDeadlocks(LTS_{N_τ}, M_F) then
        N_R = RefineDPN(N, LTS_N)
        N_{R_τ} = GetTauDPN(N_R)
        LTS_{N_{R_τ}} = ConstructLTS(N_{R_τ}, (M_I, α_I))
        return NoLivelocks(LTS_{N_{R_τ}}, M_F)
return false
```

Fig. 11 shows an LTS for DPN $\mathcal{N}_\tau$ from Fig. 9. Constructing and analyzing an LTS for $\mathcal{N}_\tau$ (instead of $\mathcal{N}_{R_\tau}$) allows detecting deadlocks in $\mathcal{N}$. In this example, there exists a dead node s_{13} from which no final node is reachable and, thus, P1 from Definition 3.5 does not hold for this LTS. Specifically, this node denotes a deadlock occurring in $\mathcal{N}$ at $M = [p_6]$. If a DPN has deadlocks, markings strictly covering M_F, or dead transitions, they can be detected without the refinement procedure by the proposed Algorithm 6, which may save a sufficient amount of time as it is illustrated in Section 8.

Upper bound time complexities for each algorithm demonstrated in this section are presented in Appendix A.

5. Algorithm termination

To prove termination of the algorithm, we use a theory of well-structured transition systems [16]. First, we define a well-structured transition system:

Definition 5.1 *([16]).* A well-structured transition system (WSTS) is an $LTS = \langle S, E, s_0 \rangle$ equipped with qo $\leq \subseteq S \times S$, s.t.:

- $\leq$ is a wqo, i.e. for any infinite sequence $s_0, s_1, s_2, \ldots$ in S, there exist indexes $i < j$ with $s_i \leq s_j$;
- $\leq$ is compatible with $\rightarrow$, i.e. for all $s_1 \leq q_1$ and transition $s_1 \rightarrow s_2$, there exists a transition $q_1 \rightarrow q_2$, s.t. $s_2 \leq q_2$.

If $\mathfrak{A}(LTS_\mathcal{N})$ is infinite, the theory of well-structured transition systems is not applicable. Thus, we need some property that, if satisfied, guarantees the finiteness of a $\mathfrak{A}(LTS_\mathcal{N})$. As such a property, we use finiteness of the closure of $A_I = \{\alpha_I\}$ under Im_t w.r.t. each $t \in T$.

Definition 5.2. Let $\mathcal{N}$ be a DPN with initial state (M_I, α_I). Let $\mathcal{A}_\mathcal{N}$ be a set of variable states in $\mathcal{N}$. The closure of $A_I = \{\alpha_I\}$ under Im_t w.r.t. each $t \in T$, denoted $\mathcal{A}^{cl}_\mathcal{N}$, is the smallest set, s.t.:

- $\mathcal{A}^{cl}_\mathcal{N} \subset 2^{\mathcal{A}_\mathcal{N}}$;
- $A_I \in \mathcal{A}^{cl}_\mathcal{N}$;
- $\forall t \in T\ \forall A \in \mathcal{A}^{cl}_\mathcal{N} : Im_t(A) \in \mathcal{A}^{cl}_\mathcal{N}$.

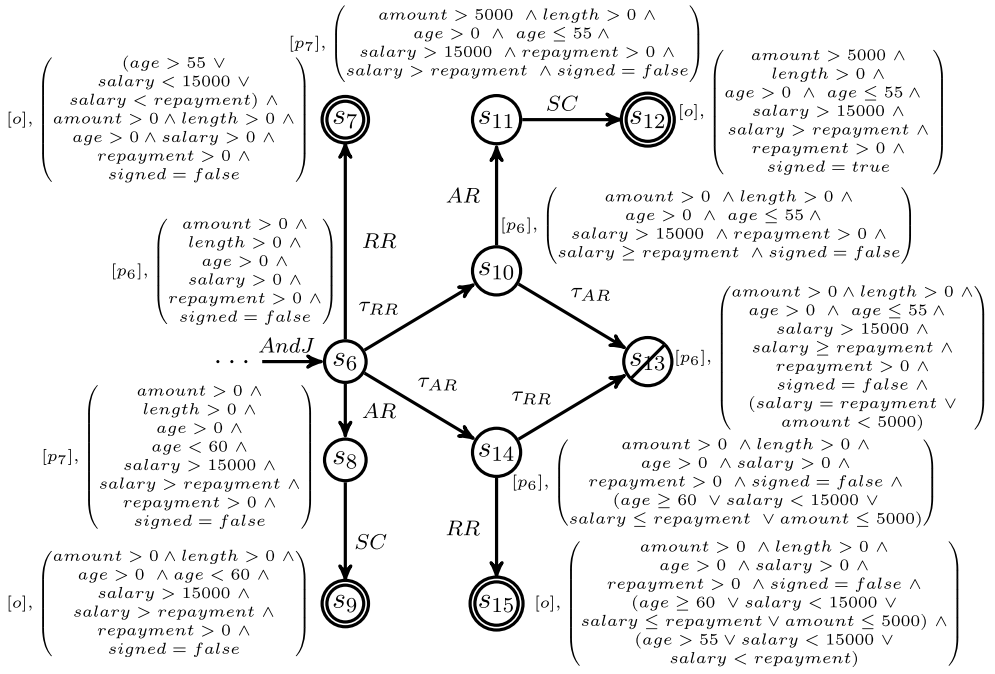


Fig. 11. A labeled transition system $LTS_{\mathcal{N}_2}$ constructed for DPN $\mathcal{N}$ from Fig. 2. Final nodes are highlighted by a double circle. Deadlocks are denoted by forbidden signs.

Note that $\mathfrak{A}(LTS_{\mathcal{N}}) \subseteq \mathcal{A}_{\mathcal{N}}$. The following statements prove that $LTS_{\mathcal{N}} = \langle S, E, s_0 \rangle$ equipped with qo $\sqsubseteq$ is a WSTS if $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite. First we show that $\sqsubseteq$ on set S is a wqo if $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite:

Proposition 5.1. *Let $\mathcal{N}$ be a DPN with initial state (M_I, α_I). Let $LTS_{\mathcal{N}} = \langle S, E, s_0 \rangle$ be a labeled transition system of $\mathcal{N}$. If $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite, then $\sqsubseteq$ on S is a wqo.*

Proof. Since $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite, set $\mathcal{A}_{LTS}$ of sets of variable states occurring in $LTS_{\mathcal{N}}$ is finite. Then, there exists some $A_i \in \mathcal{A}_{LTS}$ such that $(M_{i_1}, A_i), (M_{i_2}, A_i), (M_{i_3}, A_i), \dots$ is an infinite sequence. By Dickson's Lemma [17], for any infinite sequence $M_{i_1}, M_{i_2}, M_{i_3}, \dots$, there exist two indices $i_k < i_j$ such that $M_{i_k} \preceq M_{i_j}$. This entails that there exist two indices $i_k < i_j$ such that $(M_{i_k}, A_i) \sqsubseteq (M_{i_j}, A_i)$. Then, $\sqsubseteq$ defined on S is a wqo. □

Now we show that $\sqsubseteq$ is compatible with transition relations of $LTS_{\mathcal{N}}$:

Proposition 5.2. *Let $\mathcal{N}$ be a DPN with initial state (M_I, α_I). Let $LTS_{\mathcal{N}} = \langle S, E, s_0 \rangle$ be a labeled transition system of $\mathcal{N}$ equipped with qo $\sqsubseteq$. Then, $\sqsubseteq$ is compatible with transition relations of $LTS_{\mathcal{N}}$.*

Proof. Let $\langle M, A \rangle$, $\langle M', A' \rangle$ be states of $LTS_{\mathcal{N}}$ s.t. $\langle M, A \rangle \sqsubseteq \langle M', A' \rangle$. Then each transition that may fire at $\langle M, A \rangle$ may also fire at $\langle M', A' \rangle$. Let $\langle M_1, A_1 \rangle$ and $\langle M'_1, A'_1 \rangle$ be states yielded by firing t at $\langle M, A \rangle$ and $\langle M', A' \rangle$, respectively. Since $M \preceq M'$, $M_1 \preceq M'_1$. Since $A = A'$, $Im_t(A) = Im_t(A')$. Thus, $\langle M_1, A_1 \rangle \sqsubseteq \langle M'_1, A'_1 \rangle$ holds. Consequently, for all states $\langle M, A \rangle \sqsubseteq \langle M', A' \rangle$ and a transition $\langle M, A \rangle \xrightarrow{t} \langle M_1, A_1 \rangle$, there always exists a transition $\langle M', A' \rangle \xrightarrow{t} \langle M'_1, A'_1 \rangle$, s.t. $\langle M_1, A_1 \rangle \sqsubseteq \langle M'_1, A'_1 \rangle$. Thus, $\sqsubseteq$ is compatible with transition relations of $LTS_{\mathcal{N}}$. □

Hence we have:

Lemma 5.1. *Let $\mathcal{N}$ be a DPN with initial state (M_I, α_I). Let $LTS_{\mathcal{N}} = \langle S, E, s_0 \rangle$ be a labeled transition system of $\mathcal{N}$. If $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite, then $LTS_{\mathcal{N}}$ is a WSTS w.r.t. qo $\sqsubseteq$.*

Proof. Follows from Propositions 5.1 and 5.2. □

In the previous section, we have defined a coverability graph for an LTS equipped with relation $\sqsubseteq$ (see Definition 4.3). In [16], it is proved that the coverability graph for a WSTS is finite. Consequently, coverability graph $CG_{LTS_{\mathcal{N}}}$ for $LTS_{\mathcal{N}}$ is guaranteed to be

finite if $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite. However, it is important that coverability graph $CG_{LTS_{\mathcal{N}}}$ can be effectively constructed. According to [16], coverability graph $CG_{LTS_{\mathcal{N}}}$ of WSTS $LTS_{\mathcal{N}}$ can be effectively constructed if there exists an effective procedure for calculating the set of immediate successors for each node s and order relation $\sqsubseteq$ is decidable (i.e. there exists an effective procedure for verifying the truth of $s \sqsubseteq s'$ for any s and s'). However, both of these properties hold if $CG_{LTS_{\mathcal{N}}}$ is defined over image-closed language Φ. Thus, we make the following corollary:

Corollary 5.1. *Let $\mathcal{N} = (P, T, F, V, \Phi, guard)$ be a DPN with initial state (M_I, α_I). If $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite and Φ is image-closed, coverability graph $CG_{LTS_{\mathcal{N}}}$ of labeled transition system $LTS_{\mathcal{N}}$ over Φ can be effectively constructed.*

Now we can justify correctness of Algorithm 3:

Proposition 5.3. *Let $\mathcal{N} = (P, T, F, V, \Phi, guard)$ be a DPN, where Φ is image-closed. Let (M_I, α_I) be an initial state of $\mathcal{N}$. If $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite, then*

1) *$CheckBoundedness(\mathcal{N}, (M_I, \alpha_I))$ terminates.*
2) *$CheckBoundedness(\mathcal{N}, (M_I, \alpha_I))$ returns `true` iff $\mathcal{N}$ is bounded.*

Proof. 1) Follows from Corollary 5.1.

2) ($\Rightarrow$) Let the result of executing $CheckBoundedness(\mathcal{N}, (M_I, \alpha_I))$ be `true`. Then $CG_{LTS_{\mathcal{N}}}$ is finite and does not have any strictly covering nodes. This implies that $LTS_{\mathcal{N}}$ is finite. By construction, the set of markings in $LTS_{\mathcal{N}}$ is equal to the set of markings in $RG_{\mathcal{N}}$. Consequently, the set of markings in $RG_{\mathcal{N}}$ is finite and, thus, $\mathcal{N}$ is bounded.

($\Leftarrow$) Let $\mathcal{N}$ be bounded. Since the set of markings in $LTS_{\mathcal{N}}$ is equal to the set of markings in $RG_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite, $LTS_{\mathcal{N}}$ is finite. This entails that $CG_{LTS_{\mathcal{N}}}$ is finite and does not have any strictly covering nodes. If $CG_{LTS_{\mathcal{N}}}$ is finite and does not have any strictly covering nodes, $CheckBoundedness(\mathcal{N}, (M_I, \alpha_I))$ terminates and returns `true`. □

The result of verifying boundedness influences the execution of Algorithm 5. Having proved that Algorithm 3 returns `true` iff $\mathcal{N}$ is bounded, we can now justify termination of the entire soundness verification algorithm.

Theorem 5.2. *Let $\mathcal{N} = (P, T, F, V, \Phi, guard)$ be a DPN, where Φ is image-closed. Let (M_I, α_I) be an initial state of $\mathcal{N}$ and M_F be a final marking of $\mathcal{N}$. Let $\mathcal{N}_R$ be refined $\mathcal{N}$ and $\mathcal{N}_{R_\tau}$ be a tau-DPN for $\mathcal{N}_R$. If $\mathcal{A}^{cl}_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ are finite, then procedure $CheckSoundness(\mathcal{N}, (M_I, \alpha_I), M_F)$ terminates.*

Proof. The first step of Algorithm 5 is to verify boundedness of $\mathcal{N}$. Since $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite, procedure $CheckBoundedness(\mathcal{N}, (M_I, \alpha_I))$ always terminates (follows from Proposition 5.3). If $\mathcal{N}$ is unbounded, Algorithm 5 returns `false`; otherwise, it continues execution. If $\mathcal{N}$ is verified to be bounded, $\mathcal{N}_{R_\tau}$ is constructed and $LTS_{\mathcal{N}_{R_\tau}}$ is analyzed for data-aware soundness properties. Consider $RefineDPN(\mathcal{N}, (M_I, \alpha_I))$, which constructs $\mathcal{N}_R$ for bounded $\mathcal{N}$. Inside the loop, procedure $RefineDPN(\mathcal{N}, (M_I, \alpha_I))$ constructs $LTS_{\mathcal{N}}$, finds maximal disjoint cycles in this structure and substitutes each t for $R(t)$. Let $\mathcal{A}^{cl}_{\mathcal{N}_i}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{i+1}}$ be the closures of A_I under Im_t, w.r.t. each $t \in T$, of $\mathcal{N}$ on i-th and $i+1$-th iteration of the DPN refinement, respectively. Since, inside the loop, the algorithm splits some transitions, the number of elements in the closure of A_I under Im_t increases or remains the same, which means that $|\mathcal{A}^{cl}_{\mathcal{N}_i}| \leq |\mathcal{A}^{cl}_{\mathcal{N}_{i+1}}|$. Since $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ is also finite, $\mathcal{A}^{cl}_{\mathcal{N}_R}$ is finite and, thus, at each refinement iteration j, $\mathcal{A}^{cl}_{\mathcal{N}_j}$ is finite. At each iteration, the LTS can be effectively constructed since $\mathcal{N}$ is bounded and the closure of A_I under Im_t, w.r.t. each $t \in T$, is finite. The set of maximal disjoint cycles can be found in finite time if the LTS is finite. On each iteration, each transition is split into a finite set of transitions and, thus, substituting each $t \in T$ for $R(t)$ can be done in finite time. Consequently, each refinement iteration terminates and the number of these iterations is finite. Thus, if $\mathcal{N}$ is bounded and $\mathcal{A}^{cl}_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ are finite, then procedure $RefineDPN(\mathcal{N}, (M_I, \alpha_I))$ always terminates. To construct $\mathcal{N}_{R_\tau}$ for $\mathcal{N}_R$, procedure $GetTauDPN(\mathcal{N}_R)$ is executed. Since the set of transitions in $\mathcal{N}_R$ is finite, $GetTauDPN(\mathcal{N}_R)$ terminates. For $\mathcal{N}_{R_\tau}$, Algorithm 5 constructs $LTS_{\mathcal{N}_{R_\tau}}$ and verifies data-aware soundness properties against it. $LTS_{\mathcal{N}_{R_\tau}}$ is guaranteed to be finite if $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ is finite and $\mathcal{N}$ is bounded. If $LTS_{\mathcal{N}_{R_\tau}}$ is finite, verification of data-aware soundness properties, which is performed by $AnalyzeLTS(LTS_{\mathcal{N}_{R_\tau}}, M_F)$, can be done effectively by executing search procedures on this finite-state graph: from each node, a final node must be reachable (P1), there must exist no node with $M \succ M_F$ (P2), and for each $t \in T$, there must exist a reachable (from the initial node) transition firing of at least one $t_r \in R(t)$ (P3). Thus, $AnalyzeLTS$ terminates. Since it is the last step of Algorithm 5, procedure $CheckSoundness(\mathcal{N}, (M_I, \alpha_I), M_F)$ terminates. □

Now we show that Algorithm 5 terminates for any DPN $\mathcal{N} = (P, T, F, V, \Phi, guard)$, where the interpretation domain of Φ is $D = \mathbb{R}$.

Proposition 5.4. *Let $\mathcal{N} = (P, T, F, V, \Phi, guard)$ be a DPN, where Φ is a language of constraints, s.t. interpretation domain D is $\mathbb{R}$, set of predicates $\mathcal{P} = \{<, \leq, >, \geq, =, \neq\}$ and each predicate $P \in \mathcal{P}$ is interpreted over $\mathbb{R}$ in the standard way. Let (M_I, α_I) be an initial state of $\mathcal{N}$ and M_F be a final marking of $\mathcal{N}$. Procedure $CheckSoundness(\mathcal{N}, (M_I, \alpha_I), M_F)$ terminates.*

Proof. By Theorem 5.2, to prove termination of $CheckSoundness$, it is sufficient to show that language Φ is image-closed (1) and that $\mathcal{A}^{cl}_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ are finite (2).

1) Let X be any set of variables. Φ is image-closed if equality relation is included in $\mathcal{P}$ (a), an existential quantifier can be effectively eliminated from formulas of type $\exists v \in X : \phi$, where $\phi \in \Phi(X)$ (b), and $\sim$ is decidable for any $\phi, \phi' \in \Phi(X)$ (c). Consider each property:

(a) Follows from the definition of $\mathcal{P}$ of Φ.

(b) Each $\phi \in \Phi$ can be represented in a DNF, s.t. each disjunct of a formula is a conjunction of atomic formulas. Let ϕ_{DNF} be ϕ in DNF. Then, each disjunct ϕ_i of ϕ_{DNF} defines a semi-algebraic set over $\mathbb{R}$. In [18], it is proved that quantifier elimination is decidable for semi-algebraic sets defined over $\mathbb{R}$. Consequently, quantifier elimination in $\exists v \in X : \phi_i$ and, therefore, in $\exists v \in X : \phi$ is decidable. However, this does not entail that a sequence of equations and inequalities that define $\exists v \in X : \phi$ can be presented as a formula of $\Phi(X)$. Let us prove that there exists a procedure that computes $\phi_{res} \in \Phi(X)$, s.t. $\phi_{res} \sim \exists v \in X : \phi$. Let ϕ' be a formula constructed based on ϕ by replacing atoms $(y \neq z)$ with $(y > z \vee y < z)$ and atoms $(y = z)$ with $(y \geq z \vee y \leq z)$, where $y, z \in (Const \cup V)$. Transform ϕ' to a DNF. $\phi' \sim \phi$. Let $\phi_{res} = EliminateQuantifiers(\exists v : \phi')$, where $v \in X$. $\phi_{res} \sim \bigvee_{i=1}^{m} EliminateQuantifiers(\exists v : \phi'_i)$, where ϕ'_i is a disjunct in ϕ', since $\exists v : \phi' \sim \bigvee_{i=1}^{m} \exists v : \phi'_i$. Formula ϕ'_i is a conjunction of inequalities. Thus, Fourier–Motzkin elimination algorithm [19] can be applied to $\exists v : \phi'$ to eliminate the existential quantifier. The algorithm returns a sequence of inequalities that can be effectively represented as a conjunction of inequalities, which we denote by ϕ''_i. According to the definition of the algorithm, the set of predicates in ϕ''_i is $\{\leq, \geq, <, >\}$. The set of variables in ϕ''_i is a subset of variables in ϕ'_i. Since inequalities in ϕ'_i are atomic formulas interpreted over $\mathbb{R}$, the set of constants in ϕ''_i is a subset of constants in ϕ'_i. This entails that $\phi''_i \in \Phi(X)$. Let $\phi'' = \bigvee_{i=1}^{m} \phi''_i$. $\phi'' \in \Phi(X)$. $\phi_{res} \sim \phi''$. A set of disjuncts in ϕ' is finite. For each disjunct ϕ'_i, Fourier–Motzkin elimination algorithm effectively computes $\phi''_i \sim \exists v : \phi'_i$. This entails that ϕ'' can be effectively computed.

(c) A task of verifying truth of $\phi \sim \phi'$ can be reduced to a task of verifying satisfiability of $\phi_{eq} = (\neg\phi_i \wedge \phi_j) \vee (\phi_i \wedge \neg\phi_j)$. Decidability of verifying satisfiability of ϕ_{eq} follows from decidability of quantifier elimination for semi-algebraic sets defined over $\mathbb{R}$.

2) According to 1), Φ is image-closed. Each $A \in \mathcal{A}^{cl}_{\mathcal{N}}$ can be described by some formula $\phi \in \Phi(V)$. Let $A \in \mathcal{A}^{cl}_{\mathcal{N}}$ and $\phi \in \Phi(V)$, s.t. $[[\phi]] = A$. In the base case, $A = A_I$ and $\phi = \phi_0$. Let t be some transition. Let $\phi' = \phi \oplus guard(t)$. Then $Im_t(A) = [[\phi']]$. According to the definition of Algorithm 1, the set of constants in ϕ' is a subset of constants occurring in ϕ or $guard(t)$ if for any $\phi_b \in \Phi(X)$ and $v \in X$, the set of constants in $\phi_{qe} = EliminateQuantifiers(\exists v : \phi_b)$ is a subset of constants in ϕ_b, i.e. quantifier elimination does not generate any new constants. As is shown in the proof of 1.b), the usage of Fourier–Motzkin elimination algorithm [19] to eliminate quantifiers in formulas interpreted over $\mathbb{R}$ allows constructing $\phi_{qe} \in \Phi(X)$, s.t. $\phi_{qe} \sim \exists v : \phi_b$ and a set of constants in ϕ_{qe} is a subset of constants in ϕ_b. Thus, for any ϕ and t, there exists a formula $\phi'' \sim \phi \oplus guard(t)$, s.t. $\phi'' \in \Phi(V)$ and a set of constants in ϕ'' is a subset of constants occurring in ϕ or $guard(t)$. Since V is finite, $\mathcal{P}$ is finite and the set of constants in $\mathcal{N}$ is finite, a finite language $L(V) \in \Phi(V)$ can be used to describe any $A \in \mathcal{A}^{cl}_{\mathcal{N}}$. Thus, $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite. Since Φ is image-closed, each transition constraint in $\mathcal{N}_{R_\tau}$ can be represented as a formula of $\Phi(V)$. Thus, the same approach can be used to prove the finiteness of $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$. □

Note. Although $\mathcal{A}^{cl}_{\mathcal{N}}$ is finite for $\mathcal{N}$, where the interpretation domain of Φ is $\mathbb{R}$, the same does not hold for $\mathcal{N}$, where the interpretation domain of Φ is $\mathbb{Z}$. Fig. 12 shows DPN $\mathcal{N}$ for which $\mathcal{A}^{cl}_{\mathcal{N}}$ is infinite.

The following shows that Algorithm 5 terminates for DPN $\mathcal{N}$ if the interpretation domain of Φ is finite and Φ includes an equality relation.

Proposition 5.5. *Let $\mathcal{N} = (P, T, F, V, \Phi, guard)$ be a DPN, where Φ is a language of constraints, s.t. interpretation domain D is finite and equality relation is included in $\mathcal{P}$. Let (M_I, α_I) be an initial state of $\mathcal{N}$ and M_F be a final marking of $\mathcal{N}$. Procedure $CheckSoundness(\mathcal{N}, (M_I, \alpha_I), M_F)$ terminates.*

Proof. By Theorem 5.2, to prove termination of $CheckSoundness$, it is sufficient to show that language Φ is image-closed (1) and that $\mathcal{A}^{cl}_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ are finite (2).

1) Let X be any set of variables. Φ is image-closed, if equality relation is included in $\mathcal{P}$ (a), an existential quantifier can be effectively eliminated from formulas of type $\exists v \in X : \phi$, where $\phi \in \Phi(X)$ (b), and $\sim$ is decidable for any $\phi, \phi' \in \Phi(X)$ (c). Consider each property:

(a) Follows from the definition of $\mathcal{P}$ of Φ.

(b) Since set of variables X and interpretation domain D are finite, set of all variable states $\mathcal{A}_{\mathcal{N}}$ is finite. Let $\phi' = \exists v \in X : \phi$, where $\phi \in \Phi(X)$. Let $A' = \{\alpha \in \mathcal{A}_{\mathcal{N}} : \phi'[\alpha]\}$. Set A' can be computed effectively since $\mathcal{A}_{\mathcal{N}}$ is finite. Let $\phi'' = \bigvee_{\alpha' \in A'} \bigwedge_{v \in X} (v = \alpha'(v))$. Formula ϕ'' can be effectively constructed due to the finiteness of X and A'. Since equality relation is included in $\mathcal{P}$, $\phi'' \in \Phi(X)$. As $\phi'' \sim \phi'$, an existential quantifier can be effectively eliminated from formulas of type $\exists v \in X : \phi$.

(c) Let $\phi, \phi' \in \Phi(X)$. Since $\mathcal{A}_{\mathcal{N}}$ is finite, $\phi \sim \phi'$ can be effectively computed by verifying $\phi[\alpha] = \phi'[\alpha]$ for each $\alpha \in \mathcal{A}_{\mathcal{N}}$.

2) Since V is finite and D is finite, both $\mathcal{A}^{cl}_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ are finite. □

In the next section, we prove that Algorithm 5 returns `true` iff DPN $\mathcal{N}$ is data-aware sound.

6. Algorithm correctness

To prove that Algorithm 5 gives the correct answer regarding data-aware soundness of a DPN, we first show that $\mathcal{N}$ and $\mathcal{N}_R$ are behaviorally equivalent. By behavioral equivalence of $\mathcal{N}$ and $\mathcal{N}_R$, we mean that transition t may fire at $(M,\alpha) \in Reach_{\mathcal{N}}$ yielding state (M',α') iff $(M,\alpha) \in Reach_{\mathcal{N}_R}$ and there exists $t_r \in R(t)$ that may fire at (M,α) yielding state (M',α').

Proposition 6.1. *Let $\mathcal{N} = (P,T,F,V,\Phi,guard)$ be a DPN and $\mathcal{N}_R = (P,T_R,F_R,V,\Phi,guard)$ be a refined DPN with $T_R = \bigcup_{t\in T} R(t)$. Let (M_I,α_I) be an initial state of $\mathcal{N}$ and $\mathcal{N}_R$. Then $(M,\alpha)\xrightarrow{t}(M',\alpha')$ is in $RG_{\mathcal{N}}$ iff $\exists t_r \in R(t)$, s.t. $(M,\alpha)\xrightarrow{t_r}(M',\alpha')$ is in $RG_{\mathcal{N}_R}$.*

Proof. Follows from the construction of $\mathcal{N}_R$ (see Algorithm 4).

($\Rightarrow$) Consider a base case. Let (M_I,α_I) be an initial state for both $\mathcal{N}$ and $\mathcal{N}_R$. Let $(M_I,\alpha_I)\xrightarrow{t_0}(M_1,\alpha_1)$ be a transition firing in $\mathcal{N}$. By construction, $guard(t_0) \sim \bigvee_{t_r\in R(t_0)} guard(t_r)$. Thus, $Im_{t_0}(\alpha_I) = \bigcup_{t_r\in R(t_0)} Im_{t_r}(\alpha_I)$. Consequently, there exists $t_r \in R(t_0)$, s.t. $guard(t_r)[\langle\alpha_I,\alpha_1\rangle]$ is true. Thus, if $(M_I,\alpha_I)\xrightarrow{t_0}(M_1,\alpha_1)$ is a transition firing in $\mathcal{N}$, then there exists $t_r \in R(t_0)$, s.t. $(M_I,\alpha_I)\xrightarrow{t_r}(M_1,\alpha_1)$ is a transition firing in $\mathcal{N}_R$. Induction step. Let $n = k$ and for k it is true. Let (M_n,α_n) be a state in $\mathcal{N}$ and in $\mathcal{N}_R$. Let $(M_n,\alpha_n)\xrightarrow{t_n}(M_{n+1},\alpha_{n+1})$ be a transition firing in $\mathcal{N}$. Since $guard(t_n) \sim \bigvee_{t_r\in R(t_n)} guard(t_r)$, there exists $t_r \in R(t_n)$, s.t. $(M_n,\alpha_n)\xrightarrow{t_r}(M_{n+1},\alpha_{n+1})$ is a transition firing in $\mathcal{N}_R$.

($\Leftarrow$) Consider a base case. Let (M_I,α_I) be an initial state for both $\mathcal{N}$ and $\mathcal{N}_R$. Let $(M_I,\alpha_I)\xrightarrow{t_{r_0}}(M_1,\alpha_1)$ be a transition firing in $\mathcal{N}_R$. By construction, there exists transition t_0 in $\mathcal{N}$, s.t. $t_{r_0} \in R(t_0)$. $Im_{t_{r_0}}(\alpha_I) \subseteq Im_{t_0}(\alpha_I)$. Consequently, if $guard(t_{r_0})[\langle\alpha_I,\alpha_1\rangle]$ is true, then $guard(t_0)[\langle\alpha_I,\alpha_1\rangle]$ is true. Thus, if there exists $t_{r_0} \in R(t_0)$, s.t. $(M_I,\alpha_I)\xrightarrow{t_{r_0}}(M_1,\alpha_1)$ is a transition firing in $\mathcal{N}_R$, then $(M_I,\alpha_I)\xrightarrow{t_0}(M_1,\alpha_1)$ is a transition firing in $\mathcal{N}$. Induction step. Let $n = k$ and for k it is true. Let (M_n,α_n) be a state in $\mathcal{N}$ and $\mathcal{N}_R$. Let $(M_n,\alpha_n)\xrightarrow{t_{r_n}}(M_{n+1},\alpha_{n+1})$ be a transition firing in $\mathcal{N}_R$. Then there exists transition t_n in $\mathcal{N}$, s.t. $t_{r_n} \in R(t_n)$. $Im_{t_{r_n}}(\alpha_n) \subseteq Im_{t_n}(\alpha_n)$. Thus, if there exists $t_{r_n} \in R(t_n)$, s.t. $(M_n,\alpha_n)\xrightarrow{t_{r_n}}(M_{n+1},\alpha_{n+1})$ is a transition firing in $\mathcal{N}_R$, then $(M_n,\alpha_n)\xrightarrow{t_n}(M_{n+1},\alpha_{n+1})$ is a transition firing in $\mathcal{N}$. □

Proposition 6.1 entails that run $(M_0,\alpha_0)\xrightarrow{t_1}(M_1,\alpha_1)\xrightarrow{t_2}\dots\xrightarrow{t_n}(M_n,\alpha_n)$ is in $\mathcal{N}$ iff there exist $t_{r_1} \in R(t_1), t_{r_2} \in R(t_2), \dots, t_{r_n} \in R(t_n)$, s.t. run $(M_0,\alpha_0)\xrightarrow{t_{r_1}}(M_1,\alpha_1)\xrightarrow{t_{r_2}}\dots\xrightarrow{t_{r_n}}(M_n,\alpha_n)$ is in $\mathcal{N}_R$.

Now we show the correspondence between runs in $LTS_{\mathcal{N}_\tau}$ and $RG_{\mathcal{N}}$. Considering a special empty symbol ϵ for transitions, we define *observation function* $O : T \cup T_\tau \to T \cup \{\epsilon\}$ s.t. $O(t) \doteq t$ if $t \in T$ and $O(t) \doteq \epsilon$ if $t \in T_\tau$. Given a trace $\sigma = t^1 \cdots t^n$ of $LTS_{\mathcal{N}_\tau}$ as above, the corresponding *observed trace* is $O(\sigma) \doteq O(t^1) \cdots O(t^n)$. Now we show the correspondence between traces in $RG_{\mathcal{N}}$ and $LTS_{\mathcal{N}_\tau}$:

Lemma 6.1. *Let $\mathcal{N}$ be a DPN and $RG_{\mathcal{N}} = \langle V,E\rangle$ be a reachability graph of $\mathcal{N}$. Let $LTS_{\mathcal{N}_\tau} = \langle S,E,s_0\rangle$ be a labeled transition system of tau-DPN $\mathcal{N}_\tau$. Then $(M_I,A_I)\xrightarrow{\sigma}(M,A)$ is in $LTS_{\mathcal{N}_\tau}$ iff for each $\alpha \in A$, $(M_I,\alpha_I)\xrightarrow{O(\sigma)}(M,\alpha)$ is in $RG_{\mathcal{N}}$.*

Proof. ($\Rightarrow$) Let $(M_I,A_I)\xrightarrow{\sigma}(M',A')$ be some trace in $LTS_{\mathcal{N}_\tau}$. If $O(\sigma)$ is empty, no path is required in $RG_{\mathcal{N}}$ and, thus, this lemma holds. Consider the case when $O(\sigma)$ is not empty. Node (M_I,α_I) exists in $RG_{\mathcal{N}}$ by construction. By Definition 4.1, $A_I = \{\alpha_I\}$. Let $O(\sigma) = \langle t_1 \dots t_k\rangle$. Consider the base case. Let $\tau_1,\tau_2,\dots\tau_m$ be a sequence of τ-transitions executed at (M_I,A_I) before firing t_1, yielding (M_I,A'_I). Since there exists transition t_1 that may fire at (M_I,A'_I), $A'_I = A_I$. Let (M_1,A_1) be a state yielded by executing t_1 at (M_I,A'_I). $A_1 = Im_{t_1}(A_I) = Im_{t_1}(\alpha_I)$. Let $\tau_1,\tau_2,\dots\tau_m$ be a sequence of τ-transitions executed at (M_1,A_1), yielding (M_1,A'_1). $A'_1 \neq \emptyset$; otherwise, there exists no node (M_1,A'_1) in $LTS_{\mathcal{N}_\tau}$. Consider $RG_{\mathcal{N}}$. By construction, initial node (M_I,α_I) has the set of successors $\{(M_1,\alpha_1)|\alpha_1 \in Im_t(\alpha_I)\}$ resulted from firing t_1 at (M_I,α_I). Since $A'_1 \subseteq Im_t(\alpha)$, for each $\alpha_1 \in A'_1$, $(M_I,\alpha_I)\xrightarrow{t_1}(M_1,\alpha_1)$ is in $RG_{\mathcal{N}}$ if $(M_I,A_I)\xrightarrow{\sigma_1}(M_1,A'_1)$, where $O(\sigma_1) = t_1$, is in $LTS_{\mathcal{N}_\tau}$. Induction step. Let $n = k$ and for k it is true. Let $(M_I,A_I)\xrightarrow{\sigma_n}(M_n,A'_n)$ be in $LTS_{\mathcal{N}_\tau}$, where $O(\sigma_n) = \langle t_1,\dots,t_n\rangle$. Consider (M_n,A'_n). τ-transitions have been executed at step n, the next transition to execute is t_{n+1}. The result of executing t_{n+1} at (M_n,A'_n) is (M_{n+1},A_{n+1}) with $A_{n+1} = Im(A'_n)$. Let $\tau_1,\tau_2,\dots\tau_m$ be a sequence of τ-transitions executed at (M_{n+1},A_{n+1}), yielding (M_{n+1},A'_{n+1}). $A'_{n+1} \subseteq A_{n+1}$ and $A'_{n+1} \neq \emptyset$. Consider $RG_{\mathcal{N}}$. Set of states $\{(M_n,\alpha'_n)|\alpha'_n \in A'_n\}$ is obtained by executing t_n on the previous step. For each $\alpha'_{n+1} \in A'_{n+1}$, there exist $\alpha'_n \in A'_n$, s.t. $(M_n,\alpha'_n)\xrightarrow{t_{n+1}}(M_{n+1},\alpha'_{n+1})$ is a transition firing in $\mathcal{N}$. Consequently, for each $\alpha'_{n+1} \in A'_{n+1}$, $(M_I,\alpha_I)\xrightarrow{O(\sigma_{n+1})}(M_{n+1},\alpha'_{n+1})$ is in $RG_{\mathcal{N}}$ if $(M_I,A_I)\xrightarrow{\sigma_n}(M_{n+1},A'_{n+1})$ is in $LTS_{\mathcal{N}_\tau}$.

($\Leftarrow$) Let $(M_I,\alpha_I)\xrightarrow{\sigma}(M,\alpha)$ be in $RG_{\mathcal{N}}$. If σ is empty, no path is required in $LTS_{\mathcal{N}_\tau}$ and, thus, this lemma holds. Consider the case when σ is not empty. Let $\sigma = \langle t_1 \dots t_n\rangle$. $O(\sigma) = \sigma$. By construction, $(M_I,A_I)\xrightarrow{\sigma}(M,A)$ is in $LTS_{\mathcal{N}_\tau}$. Thus, if $(M_I,\alpha_I)\xrightarrow{\sigma}(M,\alpha)$ is in $\mathcal{N}$, then there exists σ', s.t. $O(\sigma') = \sigma$ and $(M_I,A_I)\xrightarrow{\sigma'}(M,A)$, where $\alpha \in A$, is in $LTS_{\mathcal{N}_\tau}$. □

The above lemma is given in general form. Note that refined DPN $\mathcal{N}_R$ is also a DPN and, thus, Lemma 6.1 can be applied to $\mathcal{N}_R$, as well.

Now we can prove that $CheckSoundness(\mathcal{N}, (M_I, \alpha_I), M_F)$ gives the correct answer regarding data-aware soundness of $\mathcal{N}$:

Theorem 6.2. *Let $\mathcal{N} = (P, T, F, V, \Phi, guard)$ be a DPN with initial state (M_I, α_I) and final marking M_F. Let $\mathcal{N}_R$ be refined $\mathcal{N}$. Let $\mathcal{N}_{R_\tau}$ be a tau-DPN for $\mathcal{N}_R$. When $\mathcal{A}^{cl}_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ are finite, $\mathcal{N}$ is data-aware sound iff $CheckSoundness(\mathcal{N}, (M_I, \alpha_I), M_F)$ returns* `true`.

Proof. Let $\mathcal{A}^{cl}_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ be finite. Then $CheckSoundness(\mathcal{N}, (M_I, \alpha_I), M_F)$ returns `true` iff $\mathcal{N}$ is bounded (follows from Proposition 5.3) and procedure $AnalyzeLTS(LTS_{\mathcal{N}_{R_\tau}}, M_F)$ returns `true`. $AnalyzeLTS(LTS_{\mathcal{N}_{R_\tau}}, M_F)$ verifies the following properties and returns `true` if each of them holds:

P1. $\forall (M, A) \in Reach(LTS_{\mathcal{N}_{R_\tau}})\ \exists A' : (M, A) \xrightarrow{*} (M_F, A')$;
P2. $\forall (M, A) \in Reach(LTS_{\mathcal{N}_{R_\tau}}) :\ M \geq M_F \Rightarrow (M = M_F)$;
P3. For each $t \in T$, there exists $t_r \in R(t)$, s.t. $\exists M, M', A, A' :\ (M, A) \in Reach(LTS_{\mathcal{N}_{R_\tau}})$ and $(M, A) \xrightarrow{t_r} (M', A')$.

Note that if each of these properties holds for $LTS_{\mathcal{N}_{R_\tau}}$, $\mathcal{N}$ is guaranteed to be bounded. Thus, to prove this theorem, it is sufficient to show that $\mathcal{N}$ is data-aware sound iff each of the mentioned above properties holds for $LTS_{\mathcal{N}_{R_\tau}}$.

($\Rightarrow$) Let $\mathcal{N}$ be data-aware sound.

Let $P1$ does not hold for $LTS_{\mathcal{N}_{R_\tau}}$, i.e. there exists state $(M, A) \in Reach(LTS_{\mathcal{N}_{R_\tau}})$, s.t. M_F is not reachable from (M, A). By Lemma 6.1, there exists $\alpha \in A$, s.t. $(M, \alpha) \in Reach_{\mathcal{N}_R}$. Since $\mathcal{N}$ is data-aware sound, there exists α', s.t. $(M, \alpha) \xrightarrow{*} (M_F, \alpha')$. By Proposition 6.1, $(M, \alpha), (M_F, \alpha') \in Reach_{\mathcal{N}_R}$ and (M_F, α') is reachable from (M, α) in $\mathcal{N}_R$. Let σ_F be a trace in $\mathcal{N}_R$, s.t. $(M, \alpha) \xrightarrow{\sigma_F} (M_F, \alpha')$. Then, by Lemma 6.1, $(M, A) \xrightarrow{\sigma'_F} (M_F, A')$, where $O(\sigma'_F) = \sigma_F$, is in $LTS_{\mathcal{N}_{R_\tau}}$ and, thus, M_F is reachable from (M, A). This leads to a contradiction.

Let $P2$ does not hold for $LTS_{\mathcal{N}_{R_\tau}}$, i.e. there exists some state $(M', A) \in Reach(LTS_{\mathcal{N}_{R_\tau}})$, s.t. $M' \succ M_F$. Then, for each $\alpha' \in A'$, $(M', \alpha) \in Reach_{\mathcal{N}_R}$. Since $\mathcal{N}$ is data-aware sound, there exists no state $(M, \alpha) \in Reach_{\mathcal{N}}$, s.t. $M \succ M_F$. Proposition 6.1, implies that there is no state $(M, \alpha) \in Reach_{\mathcal{N}_R}$, s.t. $M \succ M_F$, which leads to a contradiction.

Let $P3$ does not hold for $LTS_{\mathcal{N}_{R_\tau}}$, i.e. for some $t \in T$, $LTS_{\mathcal{N}_{R_\tau}}$ does not contain an arc labeled by any $t_r \in R(t)$. Since $\mathcal{N}$ is data-aware sound, for each $t \in T$, there exists state $(M, \alpha) \in Reach_{\mathcal{N}}$ at which t may fire. Proposition 6.1 entails that for each $t \in T$, there exists $t_r \in R(t)$, s.t. t_r may fire at some $(M, \alpha) \in Reach_{\mathcal{N}_R}$. Then, according to Lemma 6.1, for each $t \in T$, $LTS_{\mathcal{N}_{R_\tau}}$ contains an arc labeled by some $t_r \in R(t)$. This leads to a contradiction.

($\Leftarrow$) Let properties $P1$,$P2$ and $P3$ hold for $LTS_{\mathcal{N}_{R_\tau}}$.

Let $P1$ does not hold for $RG_{\mathcal{N}}$. There are two possible cases. First, there exists $(M, \alpha) \in Reach(RG_{\mathcal{N}})$, s.t. $M_F \neq M$ and no transition may fire at (M, α). Second, there exists infinite run $\rho = (M_I, \alpha_I) \xrightarrow{t_1} (M_1, \alpha_1) \xrightarrow{t_2} (M_2, \alpha_2) \dots$, s.t. starting from some $k \in \mathbb{N}$, for each (M_n, α_n) with $n > k$, final marking M_F is not reachable.

- Consider the first case. By Lemma 6.1, state (M, A), where $\alpha \in A$, is in $LTS_{\mathcal{N}_{R_\tau}}$. Let T^M be a set of transitions that may fire at M. T^M is not empty; otherwise, M_F is not reachable from (M, A). For each $t \in T^M$, $Im_t(\alpha) = \emptyset$. Let $A_{inter} = \cap_{t \in T^M} Im_{t_\tau}(A)$. $\alpha \in A_{inter}$. State (M, A_{inter}) exists in $LTS_{\mathcal{N}_{R_\tau}}$ by construction (can be reached from (M, A) by firing τ_t for each $t \in T^M$). Since for each $t \in T^M$, $Im_t(A_{inter}) = \emptyset$ and $Im_{t_\tau}(A_{inter}) = A_{inter}$, there exists no arc from (M, A_{inter}) that is not a loop. This entails that a node with M_F is not reachable from (M, A_{inter}), which leads to a contradiction.
- Consider the second case. Let $\mathcal{R}$ be a set of runs in $\mathcal{N}$, which start from (M_n, α_n) with $n > k$. If there exists some finite run $\rho \in \mathcal{R}$, then ρ leads to state (M, α) at which no transition $t \in T$ may fire. This case is considered above. Consider a case when each $\rho \in \mathcal{R}$ is infinite. Let $N = (P, T, F)$ be a Petri net forming a *backbone* of $\mathcal{N}$, and RG_N be a reachability graph of N. Consider cycles in RG_N that do not include M_F. For each cycle c in RG_N, we define set T_{in_c} of transitions occurring in cycle c and set T_{out_c} of transitions that lead out of cycle c. If each $\rho \in \mathcal{R}$ is infinite, there exists some run $\rho^{inf}_{RG} \in \mathcal{R}$ with suffix $\rho^{cycle}_{RG} = (M_l, \alpha_l) \xrightarrow{t_{l+1}} (M_{l+1}, \alpha_{l+1}) \xrightarrow{t_{l+2}} (M_{l+2}, \alpha_{l+2}) \dots$, s.t. run $M_l \xrightarrow{t_{l+1}} M_{l+1} \xrightarrow{t_{l+2}} M_{l+2} \dots$ forms some cycle c in RG_N and from any state occurring in ρ^{cycle}_{RG}, there is no reachable state at which any transition from T_{out_c} may fire (i.e. it is not possible to leave cycle c). By Proposition 6.1, there exist run $\rho^{inf}_{RG_R}$ of $\mathcal{N}_R$ that corresponds to ρ^{inf}_{RG} of $\mathcal{N}$. Let $\rho^{cycle}_{RG_R}$ be a suffix of $\rho^{inf}_{RG_R}$ that corresponds to ρ^{cycle}_{RG}. Let $T_{R_{in_c}} = \bigcup_{t \in T_{in_c}} R(t)$ and $T_{R_{out_c}} = \bigcup_{t \in T_{out_c}} R(t)$. $\rho^{cycle}_{RG_R}$ executes transitions from $T_{R_{in_c}}$. Neither of transitions from $T_{R_{out_c}}$ may fire at any state reachable from states occurring in $\rho^{cycle}_{RG_R}$. This entails that the choice of transitions from $T_{R_{in_c}}$ in this run forbids quitting cycle c. Let σ be a trace executed by $\rho^{inf}_{RG_R}$. Recall that $\rho^{inf}_{RG_R}$ starts from (M_n, α_n). By Lemma 6.1, there exists (M_n, A_n) in $LTS_{\mathcal{R}_\tau}$, s.t. $\alpha_n \in A_n$. Let $\mathcal{R}_{LTS_{R_\tau}}$ be a set of runs in $LTS_{\mathcal{N}_{R_\tau}}$ starting from (M_n, A_n), where each $\rho \in \mathcal{R}_{LTS_{R_\tau}}$ executes σ', s.t. $O(\sigma') = \sigma$. $LTS_{\mathcal{N}_{R_\tau}}$ is finite since properties P1 and P2 hold for $LTS_{\mathcal{N}_{R_\tau}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ is finite.

This implies that each $\rho \in \mathcal{R}_{LTS_{R_\tau}}$ ends with some cyclic run ρ^{cycle}. By construction, there exists run $\rho_{LTS_{R_\tau}} \in \mathcal{R}_{LTS_{R_\tau}}$, s.t. each transition firing $(M,\alpha)\xrightarrow{t}(M',\alpha')$ in $\rho^{inf}_{RG_R}$ is represented as a sequence of transition firings $(M,A)\xrightarrow{\sigma_t}(M',A')$ in $\rho_{LTS_{R_\tau}}$, s.t. $\sigma_t = \langle \tau_{t_1}, \tau_{t_2}, \dots, t\rangle$, where t' is in $t_1, t_2, \dots$ iff $\forall p \in P : M(p) \geq F(p,t')$ and $Im_{t'}(\alpha) = \emptyset$. Let $\rho^{cycle}_{LTS_{R_\tau}}$ be a suffix of $\rho_{LTS_{R_\tau}}$ formed as a cycle in $LTS_{\mathcal{N}_{R_\tau}}$. Let (M,α) be a state in $\rho^{cycle}_{RG_R}$. Let (M,A) be a state in $\rho^{cycle}_{LTS_{R_\tau}}$ that corresponds to (M,α). Let t be some transition occurring in cycle c. Let $t^+ \in R(t)$ be a transition, s.t. firing t^+ always yields a state that satisfies an input condition of some $t_o \in T_{out_c}$, and $t^- \in R(t)$ be a transition, s.t. firing t^- never yields a state that satisfies input condition of t_o. t_o may fire iff there exists $t_{R_o} \in R(t_o)$ that may fire. Note that run $\rho^{cycle}_{RG_R}$ must include firings of t^-; otherwise, cycle c can be left. If $(M,\alpha)\xrightarrow{t^-}(M',\alpha')$ is a transition firing in $\rho^{cycle}_{RG_R}$ and $Im_{t^+}(\alpha) = \emptyset$, then $\sigma = \langle t^+_\tau, \dots, t^-\rangle$, where $O(\sigma) = t^-$, is executed at (M,A). Consider $(M,A)\xrightarrow{\sigma}(M',A')$. Let $A_\Delta \subseteq A$ be a set of variable states, s.t. $\langle t, t_o\rangle$ cannot be executed at any $\alpha \in A_\Delta$, i.e. $Im_{t_o}(Im_t(\alpha)) = \emptyset$. Then $A' = Im_t(A_\Delta)$. Execution of t^+_τ and t^- together allows narrowing the set of variable states to those which do not allow firing t_o and which are yielded from states firing t at which never yields a state allowing firing t_o. By executing refined transitions forbidding firing some $t \in T_{out_c}$ and τ-transitions of transitions allowing firing some $t \in T_{out_c}$, we construct cycle $\rho^{cycle}_{LTS_{R_\tau}}$ containing sets of DPN states narrowed to those at which cycle c cannot be left. From each state occurring in $\rho^{cycle}_{LTS_{R_\tau}}$, there exists no reachable state, at which any $t_o \in T_{R_{out_c}}$ may fire. This implies that M_F is not reachable from any state in $\rho^{cycle}_{LTS_{R_\tau}}$. This leads to a contradiction.

Let $P2$ does not hold for $RG_{\mathcal{N}}$. Then, there exists state $(M,\alpha) \in Reach_{\mathcal{N}}$, s.t. $M \succ M_F$. Proposition 6.1 implies that $(M,\alpha) \in Reach_{\mathcal{N}_R}$. By Lemma 6.1, there exists state (M,A) in $LTS_{\mathcal{N}_{R_\tau}}$, s.t. $\alpha \in A$. This leads to a contradiction since $P2$ holds for $LTS_{\mathcal{N}_{R_\tau}}$.

Let $P3$ does not hold for $RG_{\mathcal{N}}$. As $P3$ holds for $LTS_{\mathcal{N}_{R_\tau}}$, for each $t \in T$, there exists $t_r \in R(t)$, s.t. some arc of $LTS_{\mathcal{N}_{R_\tau}}$ is labeled by t_r. By Lemma 6.1, for each $t \in T$, there exists state $(M,\alpha) \in Reach_{\mathcal{N}_R}$, at which some $t_r \in R(t)$ may fire. Then Proposition 6.1 implies that for each $t \in T$, there exists state $(M,\alpha) \in Reach_{\mathcal{N}}$, at which t may fire. This leads to a contradiction. □

Note. If for each transition $t \in T$, there exists an arc in $LTS_{\mathcal{N}}$ labeled by t, then P3 always holds for $LTS_{\mathcal{N}_{R_\tau}}$. If there is no marking $M \succ M_F$ in $LTS_{\mathcal{N}}$, then P2 always holds for $LTS_{\mathcal{N}_{R_\tau}}$. If M_F is not reachable from some state in $LTS_{\mathcal{N}}$ (or $LTS_{\mathcal{N}_\tau}$), then P1 does not hold for $LTS_{\mathcal{N}_{R_\tau}}$. However, if M_F is reachable from each state in $LTS_{\mathcal{N}}$ (or $LTS_{\mathcal{N}_\tau}$), it does not guarantee that P1 holds for $LTS_{\mathcal{N}_{R_\tau}}$. This rationale justifies the necessity of refining a DPN to faithfully verify the DPN for soundness and proves that Algorithm 6, which performs preliminary checks before refining a DPN and verifying soundness on $LTS_{\mathcal{N}_{R_\tau}}$, gives the same result as Algorithm 5.

Now we show that Algorithm 5 gives a correct answer regarding data-aware soundness of DPN $\mathcal{N}$ if the interpretation domain of Φ is $D = \mathbb{R}$.

Proposition 6.2. *Let $\mathcal{N} = (P,T,F,V,\Phi,guard)$ be a DPN, where Φ is a language of constraints, s.t. interpretation domain D is $\mathbb{R}$, set of predicates $\mathcal{P} = \{<,\leq,>,\geq,=,\neq\}$ and each predicate $P \in \mathcal{P}$ is interpreted over $\mathbb{R}$ in the standard way. Let (M_I,α_I) be an initial state of $\mathcal{N}$ and M_F be a final marking of $\mathcal{N}$. Then $\mathcal{N}$ is data-aware sound iff $CheckSoundness(\mathcal{N},(M_I,\alpha_I),M_F)$ returns* `true`.

Proof. It is proved in Proposition 5.4 that $\mathcal{A}^{cl}_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ are finite for $\mathcal{N}$. Thus, by Theorem 6.2, $\mathcal{N}$ is data-aware sound iff $CheckSoundness(\mathcal{N},(M_I,\alpha_I),M_F)$ returns `true`. □

The following demonstrates that Algorithm 5 gives a correct answer regarding data-aware soundness of DPN $\mathcal{N}$ if the interpretation domain of Φ is finite.

Proposition 6.3. *Let $\mathcal{N} = (P,T,F,V,\Phi,guard)$ be a DPN, where Φ is a language of constraints, s.t. interpretation domain D is finite and equality relation is included in $\mathcal{P}$. Let (M_I,α_I) be an initial state of $\mathcal{N}$ and M_F be a final marking of $\mathcal{N}$. Then $\mathcal{N}$ is data-aware sound iff $CheckSoundness(\mathcal{N},(M_I,\alpha_I),M_F)$ returns* `true`.

Proof. It is proved in Proposition 5.5 that $\mathcal{A}^{cl}_{\mathcal{N}}$ and $\mathcal{A}^{cl}_{\mathcal{N}_{R_\tau}}$ are finite for $\mathcal{N}$. Thus, by Theorem 6.2, $\mathcal{N}$ is data-aware sound iff $CheckSoundness(\mathcal{N},(M_I,\alpha_I),M_F)$ returns `true`. □

7. Algorithm applicability

We have proven termination of the soundness verification algorithm for specific interpretation domains of Φ, namely, for finite domains and $\mathbb{R}$. However, the algorithm may not terminate for other domains of interpretation. This section describes cases for which Algorithms 5 and 6 are not applicable.

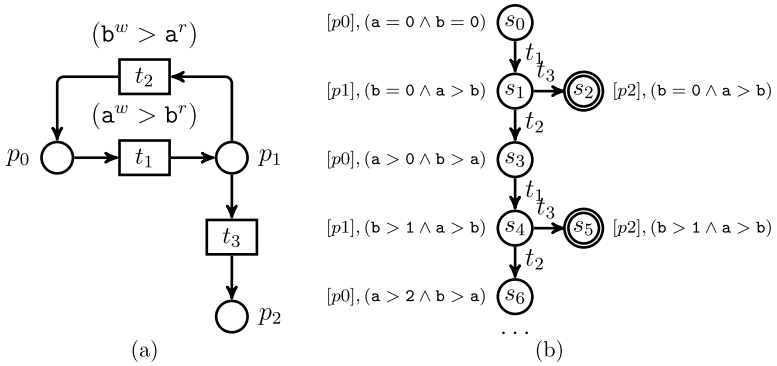


Fig. 12. (a) A DPN $\mathcal{N}$, where $M_I = [p_0]$ and $M_F = [p_2]$, a and b are integers and $\alpha_I(\mathtt{a}) = 0$ and $\alpha_I(\mathtt{b}) = 0$. (b) A part of a labeled transition system $LTS_{\mathcal{N}}$ constructed for the DPN $\mathcal{N}$. Double circles denote final states.

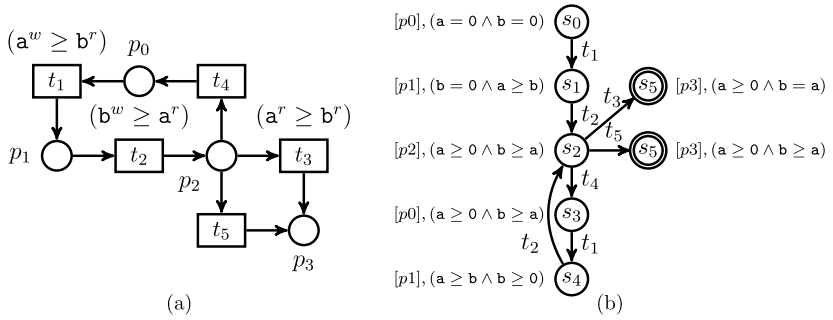


Fig. 13. (a) A DPN $\mathcal{N}$, where $M_I = [p_0]$ and $M_F = [p_3]$, a and b are integers and $\alpha_I(\mathtt{a}) = 0$ and $\alpha_I(\mathtt{b}) = 0$. (b) A labeled transition system $LTS_{\mathcal{N}}$ constructed for the DPN $\mathcal{N}$. Double circles denote final states.

There are two main cases when Algorithms 5 and 6 may not terminate. Termination is not guaranteed either if the coverability graph of $LTS_{\mathcal{N}}$ is infinite or if $LTS_{\mathcal{N}_{R_\tau}}$ is infinite. Note that if the LTS constructed on any iteration of the DPN refinement is infinite, then $LTS_{\mathcal{N}_{R_\tau}}$ is infinite, as well; thus, we can group all these cases as the case when $LTS_{\mathcal{N}_{R_\tau}}$ is infinite. If $CG_{LTS_{\mathcal{N}}}$ is infinite, then Algorithm 3 either returns `false` if it succeeds to find some covering node before entering an endless loop or does not terminate at all. $CG_{LTS_{\mathcal{N}}}$ is infinite if a $\mathfrak{A}(LTS_{\mathcal{N}})$ is infinite. However, if $CG_{LTS_{\mathcal{N}}}$ is finite, $LTS_{\mathcal{N}_{R_\tau}}$ may be infinite since $\mathfrak{A}(LTS_{\mathcal{N}})$ may differ from $\mathfrak{A}(LTS_{\mathcal{N}_{R_\tau}})$.

Fig. 12 shows DPN $\mathcal{N}$ defined on domain $D = \mathbb{Z}$ with $LTS_{\mathcal{N}}$ containing an infinite set of sets of variable states. Transitions t_1, t_2 and places p_0, p_1 form a cycle that infinitely increments minimum possible values for variables a and b that are associated with domain D. Since this cycle generates an infinite set of sets of variable states in $LTS_{\mathcal{N}}$, $CG_{LTS_{\mathcal{N}}}$ becomes infinite, Algorithm 3 does not terminate and, thus, Algorithms 5 and 6 are not applicable.

Fig. 13 shows DPN $\mathcal{N}$ containing only reflexive operators in transition constraints, and $LTS_{\mathcal{N}}$ for this DPN. There is a cycle formed by places p_0, p_1, p_2 and transitions t_1, t_2, t_4, but the minimum possible values for variables a and b are not incremented due to reflexiveness of comparison operators in transition guards. Therefore, $\mathfrak{A}(LTS_{\mathcal{N}})$ is finite and, thus, $CG_{LTS_{\mathcal{N}}}$ is finite.

Consider DPN $\mathcal{N}$ that has a finite labeled transition system $LTS_{\mathcal{N}}$. Adding tau-transitions or splitting DPN transitions influences the number of nodes in a labeled transition system. It may be a case that $LTS_{\mathcal{N}}$ is finite but $LTS_{\mathcal{N}_R}$, $LTS_{\mathcal{N}_\tau}$ and, respectively, $LTS_{\mathcal{N}_{R_\tau}}$ have an infinite number of nodes. Note that $\mathcal{N}_\tau$ has additional τ-transitions with $guard(\tau_t) = \neg\exists(write(t)) : guard(t)$. Refinement procedure, in turn, splits some transition t by joining $guard(t)$ with either $\neg\exists(write(t')) : guard(t')$ or $\exists(write(t')) : guard(t')$, where t' is a transition that allows leaving a cycle which contains t, and replacing each read variable v^r in the input condition of t' with corresponding write variable v^w if v^w occurs in $guard(t)$. Due to the added negations, the reflexiveness of comparison operators, which was preserved in $\mathcal{N}$, can be lost in $\mathcal{N}_R$, $\mathcal{N}_\tau$ and, respectively, in $\mathcal{N}_{R_\tau}$. The appearance of irreflexive comparison operators in such DPNs can lead to the labeled transition systems of such DPNs being infinite.

Fig. 14 shows $\mathcal{N}_\tau$ for DPN $\mathcal{N}$ from Fig. 13. According to Definition 4.4, it is only possible to add one silent transition τ_{t_3}. Since $guard(\tau_{t_3})$ negates $guard(t_3)$, reflexive operator $\geq$ is replaced by irreflexive $<$ in the transition constraint of τ_{t_3}. The appearance of a strict operator provides a possibility to form a cycle that infinitely increments the minimum possible values for variables a and b. In Fig. 14, this cycle is formed by places p_0, p_1, p_2 and transitions $t_1, t_2, \tau_{t_3}, t_4$. This cycle generates an infinite set of sets of variable states in $LTS_{\mathcal{N}_\tau}$. Because of that, Algorithms 5 and 6 do not terminate although $CG_{LTS_{\mathcal{N}}}$ for $LTS_{\mathcal{N}}$ is finite.

Fig. 15 shows $\mathcal{N}_R$ constructed for $\mathcal{N}$ from Fig. 13. DPN $\mathcal{N}$ contains a cycle formed by transitions t_1, t_2, t_4 with transitions t_3, t_5 allowing to leave the cycle. As a result of the refinement procedure, transition t_2 is split into transitions t_2^+ and t_2^-. Transition t_2^- is

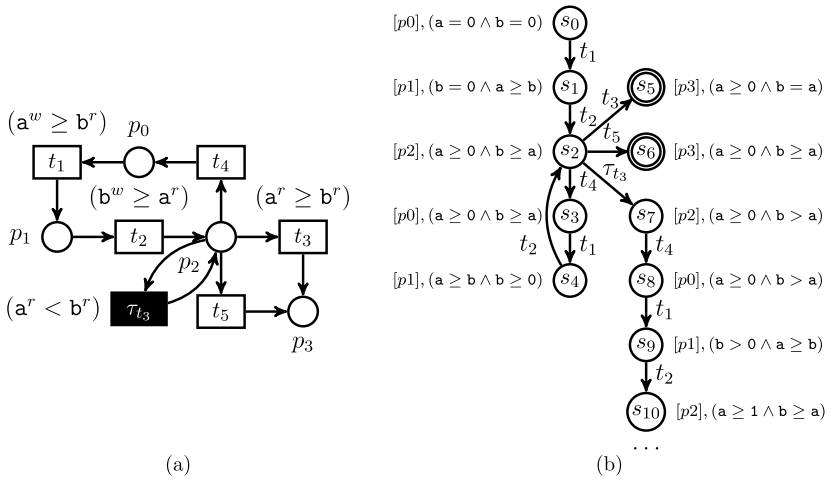


Fig. 14. (a) DPN $\mathcal{N}_\tau$ for the net $\mathcal{N}$ from Fig. 13. (b) A fragment of labeled transition system $LTS_{\mathcal{N}_\tau}$ constructed for $\mathcal{N}_\tau$. Double circles denote final states.

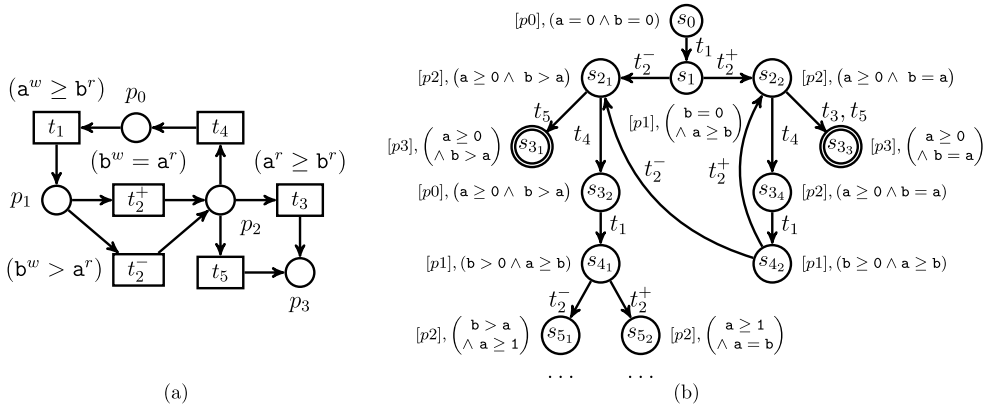


Fig. 15. (a) DPN $\mathcal{N}_R$ for the net $\mathcal{N}$ from Fig. 13. (b) Labeled transition system $LTS_{\mathcal{N}_R}$ constructed for $\mathcal{N}_R$. Double circles denote final states.

formed by adding negated input condition of t_3 to $guard(t_2)$; and, thus, $b^w \geq a^r$ is transformed to $b^w > a^r$. The appearance of a strict comparison operator makes it possible to form a cycle t_1, t_2^-, t_4 that infinitely increments the minimum possible values for variables $\mathtt{a}$ and $\mathtt{b}$. This cycle generates an infinite set of sets of variable states in $LTS_{\mathcal{N}_R}$. Because of that, Algorithms 5 and 6 do not terminate.

Unfortunately, we failed to imitate inhibitor arcs in a DPN with variable-operator-constant and variable-operator-variable conditions as it was previously done for DPNs with arithmetic conditions in [13]. Note that for DPNs with variables of integer type, the quasi-ordering relation $\sqsubseteq$ (Definition 4.2) is generally speaking not a wqo. This is because some $LTS_{\mathcal{N}}$ contain infinite subsets of pairwise incomparable elements. Therefore, DPNs with integer domains are not WSTS with respect to $\sqsubseteq$. Currently, we do not know whether it is possible to define another qo on LTS states to get a WSTS. This question is open and requires future research. If it is possible, then our algorithms could potentially be generalized to this class of DPNs. Regarding Turing-completeness of this class of DPNs, S. Lasota in [20] proposed the conjecture that a class of Petri nets is Turing-complete, if this class is not a WSTS, in particular, when it is not possible to define a wqo on its models' sets of states. Thus, the question of decidability of soundness verification such DPNs with integer domains also remains open.

8. Implementation and experiments

The proposed algorithm for data-aware soundness verification has been implemented as a research prototype, presented as a .NET desktop application available for download on Github.[1] The application allows a user to import a DPN in a PNML-based file format, visualize it, and verify its data-aware soundness. To perform operations over expressions, the satisfiability modulo solver (SMT) Z3 [21] is used. The developed tool allows using both versions of data-aware soundness verification algorithm proposed in

[1] https://github.com/SuvorovNM/DataPetriNet.

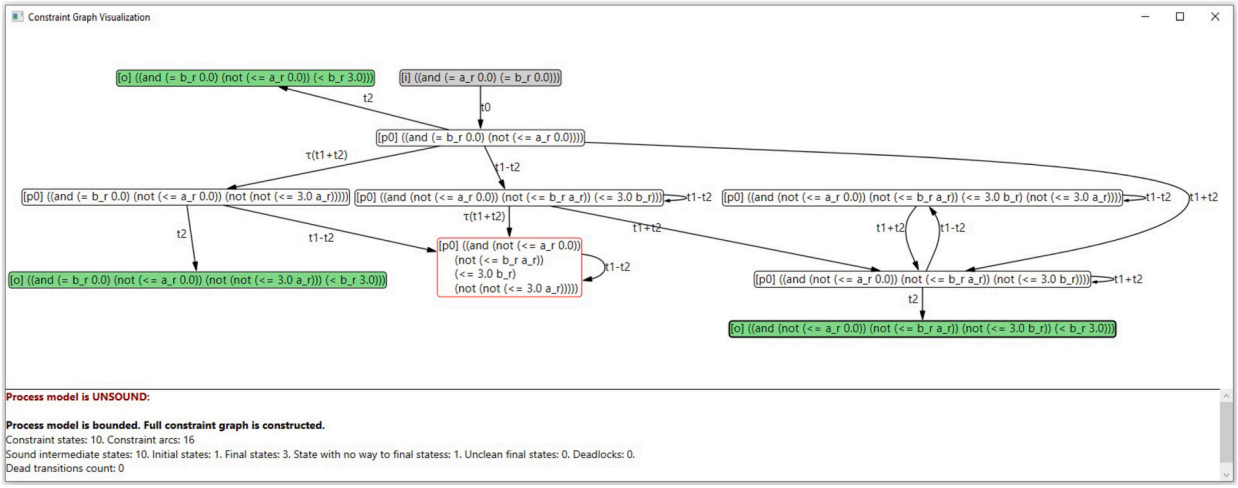


Fig. 16. Screenshot of the tool that implements the data-aware soundness verification algorithm. The tool demonstrates $LTS_{\mathcal{N}_{R_\tau}}$ built for DPN $\mathcal{N}$ from Fig. 3.

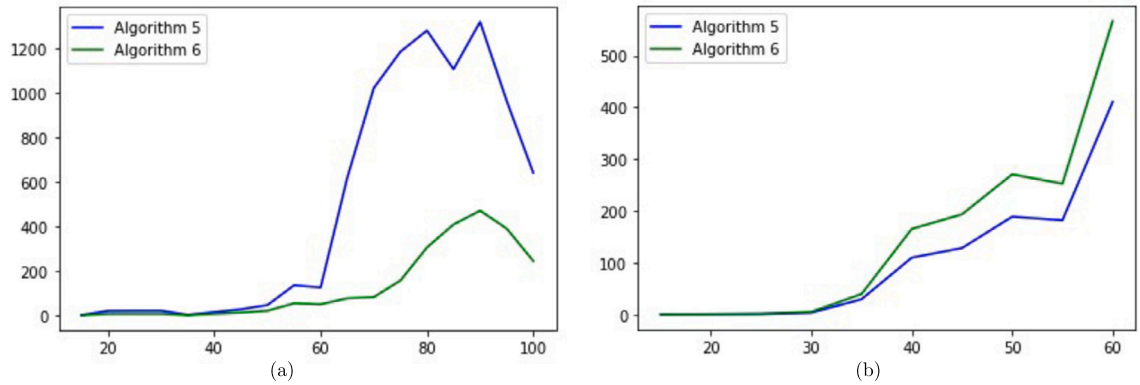


Fig. 17. Time spent by the soundness verification algorithm versions for different n. Abscissa represents n. Ordinate represents time (in seconds). Lines represent time spent by Algorithms 5, 6. (a) Bounded unsound DPNs with less than 50% of dead transitions. (b) Sound DPNs.

this paper (namely, Algorithm 5 and Algorithm 6). Fig. 16 demonstrates the outputs provided by the application for DPN $\mathcal{N}$ from Fig. 3.

We have evaluated performance of the developed algorithms on synthetic and real-life data. All the experiments have been conducted on Intel Core i7-7700HQ (4× 2.80 GHz) with 16 GB RAM.

To test the algorithm on the synthetic data, a tool that generates sample DPNs of different sizes has been developed. The tool allows generating a DPN according to the predefined parameters. DPNs produced by this tool are checked for soundness, and the time required by the soundness verification algorithm is measured. A detailed description of this tool is given in the Supplementary Materials.

Given $n \in \mathbb{N}$, we first consider DPNs of the following setup:

- $1.2n$ places,
- n transitions,
- $2.15n$ arcs for unsound DPNs and $2n$ arcs for sound DPNs,
- $0.25n$ variables, and
- $0.5n$ conditions.

We generate 3 DPNs with less than 50% of dead transitions for each n. We start with $n = 5$. After each iteration, we increase n by 5. Generated DPNs are verified for soundness using both versions of the soundness verification algorithm. Fig. 17 compares the time needed for the algorithm versions to verify soundness of DPNs. Generation of sound DPNs is a very time-consuming task since the portion of sound DPNs among all DPNs is very low and decreases with the growth of a DPN size; thus, the maximum n for sound DPNs is only 60.

The results show that although the time needed for the algorithm grows exponentially with an increase in a DPN size, the algorithm is still applicable for process models of small and medium sizes. For unsound DPNs, Algorithm 6, in general, verifies

Table 1
Algorithm evaluation on specific data-aware process models. $|P|$ is the number of DPN places, $|T|$ is the number of DPN transitions, $|F|$ is the number of DPN arcs, $|\Phi_{atom}|$ is the number of atomic formulas in the DPN, $|V|$ is the number of DPN variables.

Model	Sound	$\|P\|$	$\|T\|$	$\|F\|$	$\|\Phi_{atom}\|$	$\|V\|$	Algorithm 5 (sec.)	Algorithm 6 (sec.)
Hospital Billing [13, Fig. 15.3]	yes	17	36	74	36	4	108.13	188.59
Sepsis [13, Fig. 13.6]	yes	24	36	80	37	4	101.72	159.83
Digital Whiteboard Discharge [13, Fig 14.3]	yes	6	6	12	5	4	0.04	0.06
Digital Whiteboard Transfer [13, Fig 14.3]	no	7	6	12	3	4	0.08	0.04
Road Fines [13, Fig. 12.7]	no	9	19	38	30	8	0.43	0.24
BPMN Example (Fig. 2)	no	9	8	18	13	6	0.07	0.08
Livelock Example (Fig. 3)	no	3	3	6	3	2	0.16	0.16

soundness of a DPN quicker than Algorithm 5. This can be reasoned by the fact that for most unsound DPNs, Algorithm 6 allows verifying soundness of a DPN without refining this DPN, i.e. not all verification steps are executed. The relative difference in verification time spent by the algorithms tends to increase with an increase in the DPN size. For sound DPNs, Algorithm 5 is always quicker than Algorithm 6 since for each of these DPNs, the refinement procedure must be executed. According to the results obtained, for each sound DPN, despite its size, the time spent by Algorithm 6 is nearly 1.4 higher than the time spent by Algorithm 5.

Table 1 shows results of the algorithm evaluation on specific data-aware process models. We have tested our algorithms on DPNs presented in Section 2 and on DPNs from the literature, most of which were mined from real-life event logs. We would like to express our sincere gratitude to the authors of [11], who have shared some DPNs obtained from mining these event logs. Shared DPNs have been used by authors for evaluating the performance of the soundness verification algorithm for DPNs with arithmetic conditions. This allows us to compare the performance of that algorithm with the algorithm proposed in our paper. However, we should note that since our algorithm does not allow transition constraints to have arithmetic operators, we were not able to check our algorithm performance on some of the shared DPNs. For each of the considered models (models without arithmetic operators in the conditions), our algorithm spent less than 2 minutes to verify soundness. On the Hospital Billing model, our algorithm takes 108 seconds, while algorithm [11] takes 181 seconds. On the Road Fines model, our algorithm spends 430 milliseconds, whereas algorithm [11] spends 3.1 seconds. To verify the Sepsis model and Digital Whiteboard models, both algorithms take nearly the same time: 102 seconds for the Sepsis model and 0.1 seconds for the Digital Whiteboard models. These results show that the algorithms proposed in our paper are indeed applicable for verifying soundness of real-life process models and that there are cases when our verification technique takes substantially less amount of time than the soundness verification technique proposed in [11].

9. Related work

A lot of researchers are currently focused on the problem of process model soundness verification. However, most of the works are devoted specifically to the control-flow perspective [22] and ignore the data flow of process models. Not considering the data-flow perspective is acknowledged to be a significant limitation of these works [9,23]. In this section, we highlight some attempts to verify soundness of integrated models, which incorporate data and decisions.

In the last decade, one of the standardized approaches to model a data-aware process was to use BPMN together with DMN (Decision Model and Notation) [24]. The DMN was proposed to achieve separation of concerns and to possibly complement the BPMN for designing decisions related to process models. Consequently, there appeared some algorithms, e.g. [25,26], that allowed to verify soundness of such models. However, the algorithms proposed for this task are mainly based on verifying single DMN tables for correctness, which means that verification is only performed locally to single decision points. This entails that some deadlocks and livelocks may remain undetected. There has also been an attempt to verify standalone BPMN models with conditions on arcs by examining properties expressed in LTL formulas [27], which indeed allows to faithfully check whether or not some data-aware properties hold for the process. However, this approach cannot be applied for verifying process model soundness since termination properties cannot be expressed in LTL and, therefore, cannot be checked using this algorithm.

Verifying soundness of BPMN models is quite a difficult task due to the complexity of this notation. Thus, most works dedicated to data-aware soundness verification use workflow nets and their extensions as a process model representation instead of high-level BPMN models. Such works mainly differ in the way the interplay of data-flow and control-flow perspectives is captured. One of the first workflow extensions that captures the data perspective of the process has been proposed in [6]. The authors introduce a conceptual workflow model extended with data operations (WFD-net) and a technique to verify soundness of this model. Each state of the model is represented by a marking and a subset of satisfied guards. In [10], it is argued that this formalism is not expressive enough to capture the data perspective of real processes since activities are assumed to read and write entire guards instead of single data variables. [28] supports this opinion and proposes a workflow model extended with SQL operations on tables (WFT-net) to overcome this limitation and an algorithm to verify its soundness. In [29], the authors extend this formalism and add domain constraints to the model that must hold at each model state. Each state of this formalism is represented by a marking, a state of data elements, a state of tables, and a subset of satisfied guards. However, this formalism is quite difficult to perceive both due to introducing separate notions of data elements and tables and due to attaching SQL-expressions on transitions. The data perspective is taken quite abstractly, which is why it is not possible to examine what specific values the variables can have during the process execution.

As a formalism that allows to elegantly capture both data and control flows of a process model, a Data Petri net has been proposed in [7]. One of the successful approaches to verify data-aware soundness of integrated models represented as DPNs has been proposed in [9]. The approach is based on encoding a DPN into a colored Petri net (CPN) and verifying a corresponding, effectively checkable variant of soundness on the obtained CPN. The input model for this algorithm cannot contain conditions that compare the values of two variables. The algorithm only allows atomic conditions of type $(v\ p\ c)$, where v is a variable, p is a comparison operator and c is a constant. However, each transition constraint can be composed of a set of atomic conditions using conjunction and/or disjunction.

In [10], the authors provide a data-aware soundness verification algorithm that is based on the construction of a state-transition system called a *constraint graph*. The constraint graph for DPN $\mathcal{N}$ is similar to the labeled transition system, introduced in Definition 4.1, for $\mathcal{N}_\tau$. The algorithm explores the state space of a DPN, constructs a constraint graph, and verifies it for specific soundness properties described in [10]. The algorithm only allows atomic conditions as transition constraints. Each atomic condition is a condition of type $(v\ p\ x)$, where v is a variable, p is a comparison operator and x is either a constant or a variable. However, this algorithm does not always correctly verify soundness. Consider the DPN in Fig. 3, which enters a livelock when $t1$ sets any value larger than 3 for a. The DPN is not data-aware sound, but if we built the constraint graph as done in this work, the constraint graph would be data-aware sound. Then, according to the theorem proposed in [10], the DPN must be data-aware sound but it is not.

In [30], there has been proposed an algorithm to verify *controllable* data-aware soundness, i.e. verify whether a given actor can guarantee soundness of a whole DPN. We have found out that this algorithm can also be used for classical data-aware soundness verification of DPNs with conditions of type $(v\ p\ x)$, where v is a variable, p is a comparison operator and x is a constant or a variable. The algorithm uses an eager constraint graph, where each node is a tuple of a marking and a set of constraints, but for each marking, *all possible combinations* of constraints that may hold on this marking are considered. The eager constraint graph is significantly bigger than the classical constraint graph but allows a more subtle analysis of execution scenarios. This algorithm verifies soundness correctly but requires to consider all realizable combinations of markings and sets of constraints which makes this algorithm not viable for models of large and medium sizes. Applying the same technique for DPNs with logical expressions as constraints could make the algorithm not applicable even for rather small process models.

Paper [11] extends the work [10] and proposes a soundness verification algorithm for DPNs with arithmetic conditions. Each transition constraint of such DPNs is a conjunction of atomic arithmetic conditions. For each reachable marking, the algorithm constructs a constraint graph starting from this marking and verifies the reachability of the final states on these graphs. The algorithm does not have a flaw as the algorithm from [10] and, thus, allows to faithfully verify soundness of the DPN from Fig. 3. However, in general, soundness verification for DPNs with arithmetic conditions is undecidable. It is stated that the algorithm terminates if either the constraint language only has formulas of variable-operator-constant or variable-operator-variable types, or if it holds for the control flow that each state depends only on finitely many actions in the past [31]. This work, in comparison to ours, does not allow disjunctions of conditions as transition constraints. In contrast, our work does not allow arithmetic conditions as transition constraints. One of the performance issues connected with this algorithm is constructing constraint graphs for each DPN marking. In our algorithm, we also construct several transition systems for a DPN, but, in general, that number is lower: we construct an LTS for each DPN refinement iteration (in general, there are a few of them) and we construct a single LTS to conduct the last step of soundness verification. This number is usually significantly lower than the number of DPN markings.

In our work, to verify soundness of a DPN, we construct and analyze a labeled transition system for the tau-refined DPN. Refining a DPN and adding tau-transitions allows a more subtle investigation of the DPN behavior on the LTS. This makes the proposed algorithm capable of capturing livelocks that remained undetected by algorithm [10]. Our algorithm can also be used for checking soundness of DPNs with composite conditions as transition constraints. Previously, the only way to verify soundness of such models was to transform them into models with simpler conditions and verify the resultant models for soundness. This approach was not always viable due to the complexity of transforming all transition constraints to normal forms and the possibility of a significant increase in the model size after the transformation. In contrast, our approach does not transform formulas to any of the normal forms. It only splits transitions that simultaneously occur in cycles and update variables tested by transitions leading out of these cycles. Compared with the latest developed soundness verification algorithm for DPNs [11], our algorithm also allows minimizing the number of the state-space structure constructions needed for soundness verification, which helps to decrease the overall verification time.

10. Conclusions

In this paper, we have proposed a soundness verification algorithm for data-aware process models with composite conditions on activities. To represent data-aware process models, we have used Data Petri nets. The proposed algorithm allows verifying soundness DPNs with variable-operator-constant and variable-operator-variable conditions combined using conjunction and/or disjunction. Compared with the latest algorithms for soundness verification of DPNs, our algorithm allows for substantial minimization of the number of state-space constructions needed for the verification, which helps to perform soundness verification in a shorter time. Additionally, supporting disjunctions out-of-the-box allows to skip DPN transformation stage that was previously needed to conduct the verification using state-of-the-art algorithms [11,30]. The correctness of the proposed algorithm is justified for DPNs with conditions defined over real numbers. The algorithm is implemented as a research prototype that is available for download.

The conducted experiments have shown that the proposed algorithm is applicable for verifying soundness of DPNs of small and medium sizes. Although the verification time grows exponentially with the growth of the DPN size, the results of experiments have shown that for DPNs with less than 60 transitions, the algorithm generally requires less than 10 minutes to verify soundness. The proposed algorithm can be used right after discovering or manually constructing a data-aware process model to verify correctness of

both control and data flows of the obtained model. Since existing discovery techniques of data-aware process models do not guarantee soundness of the resultant models, soundness verification of these models is an important step that helps to avoid incorrectness of analyses performed on the discovered models and detect possible deadlocks and livelocks in process-aware information systems that base on the discovered models.

In the future, we plan to use this research as a basis for developing repair algorithms for unsound data-aware process models. Currently, the repair of any unsound data-aware process models is done fully manually which forces modelers to waste a significant amount of time on this process. The developed soundness repair algorithms could automate this error-prone and time-consuming process and provide modelers with repaired models that are proved by the system to be data-aware sound.

CRediT authorship contribution statement

Nikolai M. Suvorov: Writing – review & editing, Writing – original draft, Visualization, Software, Resources, Investigation, Formal analysis, Data curation, Conceptualization. **Irina A. Lomazova:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The manuscript contains a link to the source code repository.

Acknowledgements

We would like to express our deepest gratitude to Massimiliano de Leoni for inspiring discussions and valuable comments.
This study has been supported by the Basic Research Program at HSE University, Russia.

Appendix A. Time complexity estimation for presented algorithms

In this section, we highlight the time complexity of the algorithms presented in the paper. For simplicity, we consider the case when the constraint language of a DPN is a language with the interpretation domain $D = \mathbb{R}$ and $\mathcal{P} = \{<, \leq, >, \geq, =, \neq\}$. In other words, each DPN variable is of real type, and only basic comparison operators are allowed.

First, consider Algorithm 1 that computes the formula of a new LTS state. The algorithm's time complexity mainly depends on the time complexity of the quantifier elimination. According to [32], the time complexity of the quantifier elimination for the language that we consider is $O(2^{2^L})$, where L is the length of a formula. Then, the time complexity of Algorithm 1 is $O(V \times (L + M) + 2^{2^{(L+M)}})$, where V is the number of variables in a DPN, L is the length of a formula of the current LTS state, and M is the length of a transition constraint. Algorithm 1 execution time highly depends on the length of the input formulas; thus, this algorithm may take an unreasonable amount of time when large formulas are given as an input. During the LTS construction, formulas in LTS states grow in size, but, fortunately, in our experiments, the formulas did not reach the size at which Algorithm 1 becomes inapplicable.

Second, consider Algorithm 2 that constructs an LTS for a DPN. At each state, the algorithm tries to fire each enabled DPN transition, checks whether the formula of a state obtained by firing a transition is satisfiable, and if the formula is satisfiable, compares the state with all existing states to verify whether this node has already been added before. To simplify the following formulas, let us define L as the sum of the largest formula length in the LTS and the largest transition constraint length in the DPN. The time complexity of getting all enabled transitions at a given LTS state is $O(T \times F)$, where T is the number of DPN transitions, and F is the number of DPN arcs. The time complexity of computing a formula for a new state can be estimated as $O(V \times L + 2^{2^L})$, where V is the number of variables in a DPN. Since in the worst case, $V \times L \ll 2^{2^L}$, we can simplify the complexity to $O(2^{2^L})$. Checking satisfiability can be done through the quantifier elimination; thus, the time complexity can be estimated as $O(2^{2^L})$. The time complexity of trying to find the same node in the LTS is $O(S \times (P + 2^{2^{2L}}))$, where S is the number of states in the LTS and P is the number of places. Thus, the time complexity of Algorithm 2 can be estimated as $O(S \times T \times (F + 2^{2^L} + S \times (P + 2^{2^{2L}})))$. This can be simplified to $O(S^2 \times T \times (P + 2^{2^{2L}}))$. We can omit F here since $F \leq 2 \times T \times P$, whereas $2 \times T \times P$, in the worst case, is much lower than $S \times P$. Consider Algorithm 3 that checks boundedness of a DPN. In the worst case, the DPN is bounded and the algorithm must construct the whole LTS (in the case of boundedness, $CG_{LTS_{\mathcal{N}}}$ is equivalent to $LTS_{\mathcal{N}}$). In this case, the algorithm time complexity becomes equivalent to the time complexity of constructing $LTS_{\mathcal{N}}$. Although, for each coverability graph node, we have to maintain the set of nodes, from which this node is reachable, this does not significantly contribute to the algorithm complexity. Estimating the time and space complexity of Algorithm 3 for cases when the DPN is unbounded is a challenging task and is currently left for future research.

Consider Algorithm 4 that refines a DPN. The refinement is conducted in a loop. In the loop, the algorithm constructs an LTS, finds maximal disjoint cycles in the LTS, and refines transitions. We skip the first step, as it is described above, and consider time complexity of the two next steps in the following paragraphs.

Consider the complexity of finding maximal disjoint cycles. This can be done by finding elementary cycles and, then, joining those which have a common node. Let S be the number of nodes in the LTS, E be the number of LTS edges. All the elementary cycles can be found with $O(S+E)$ time complexity using the Johnson's algorithm [33]. Joining cycles can be done in $O(S)$. By Definition 4.1, $E \leq S \times T$, where T is the number of DPN transitions; thus, the time complexity of finding maximal disjoint cycles can be estimated as $O(S \times T)$.

When refining transitions, in the worst case, we need to perform two quantifier eliminations against each DPN transition. Each transition can be split into not more than 2^{T-1} transitions. Then, we need to add arcs to each refined transition, the number of which has the upper bound of $T \times 2^{T-1}$, where T is the number of original DPN transitions. Consequently, the time complexity of refining transitions is $O(T \times 2^{2^L} + T \times 2^{T-1} + T \times 2^{T-1} \times F)$, where T and F are the numbers of DPN transitions and arcs, respectively. We can simplify it to $O(T \times 2^{2^L} + T \times 2^T \times F)$, where F can be replaced with $P \times T$, according to Definition 3.2.

The time complexity of a single refinement iteration is the sum of the time complexities of constructing an LTS, finding maximal disjoint cycles, and refining transitions. Thus, single refinement iteration time complexity is $O(S^2 \times T \times (P + 2^{2^{2L}}) + T^2 \times 2^T \times P)$. Then, the overall time complexity of DPN refinement is $O(C(S^2 \times T \times (P + 2^{2^{2L}}) + T^2 \times 2^T \times P)$, where C is the number of refinement iterations. In our experiments, the number C was always not greater than 6, so it is usually quite a low number, but there potentially can be cases, when this number is high. According to the constructive definition of Algorithm 4, C is always guaranteed to be not greater than $A \times T$, where A is the number of mutually non-equivalent formulas in the DPN constraint language Φ and T is the number of DPN transitions. However, we are sure that even in the worst case, C is much lower than $\Phi \times T$; thus, using this notation can give a misleading impression of the complexity of the algorithm. In the future, we plan to give a more precise estimation of the upper bound for C, while in the current state, we will continue denoting the number of refinement iterations by C.

Consider Algorithm 5 that verifies soundness of a DPN. The algorithm checks boundedness of a DPN, refines a DPN, constructs an LTS for the tau-refined DPN, and analyzes it for soundness properties. Time complexities of checking boundedness, refining a DPN, and constructing an LTS are given above. Time complexity of $GetTauDPN$, which constructs a tau-DPN according to Definition 4.4, is $O(T \times 2^{2^L})$. Procedure $AnalyzeLTS$, which analyzes an LTS for soundness properties from Definition 3.5, verifies that from each LTS node, the final node is reachable (P1), that there are no nodes with markings that strictly cover M_F (P2), and that each DPN transition is represented by some arc in the LTS (P3). Let S be the number of states in the LTS, and E be the number of arcs in the LTS. Then, the time complexity of checking P1 is $O(S \times E)$, the time complexity of checking P2 is $O(S)$, and the time complexity of checking P3 is $O(E)$. Consequently, the time complexity of $AnalyzeLTS$ is $O(S \times E)$. The time complexity of Algorithm 5 is the sum of time complexities for checking boundedness, refining a DPN, constructing a tau-DPN, constructing its LTS, and verifying it for soundness. We simplify the resultant formula and obtain the following estimation of Algorithm 5 time complexity in the case of DPN boundedness: $O(C(S^2 \times T \times (P + 2^{2^{2L}}) + T^2 \times 2^T \times P))$, where C is the number of refinement iterations, S is the number of states in LTS_{R_τ}, T and P are the numbers of places and transitions in $\mathcal{N}_{R_\tau}$, respectively, and L is the sum of the largest formula length in LTS_{R_τ} and the largest transition constraint length of $\mathcal{N}_{R_\tau}$.

It is a challenging task to define the time complexity of the algorithm using the parameters of the source DPN since the formulas in the LTS nodes and transition constraints substantially grow in size during the DPN refinement. One option is to presume that all the formulas in the LTS and on transitions are always in some normal form, i.e. the perfect disjunctive normal form (PDNF). To use this assumption, we need to convert formulas, which we obtain during the algorithm execution, to the PDNF. Fortunately, the number of formulas in the PDNF that can be constructed using the language that we consider in this section is finite. Thus, we can create a set of all possible PDNF formulas and substitute newly obtained state or transition constraints for the corresponding PDNF formulas, which can be found in finite time. Both for the state constraint language and transition constraint language, the number of formulas in the PDNF that can be constructed is not greater than $2^{4V(2Const+3V)}$, where V is the number of DPN variables, $Const$ is the number of DPN constants. The maximum length of such PDNF formulas is also not greater than $2^{4V(2Const+3V)}$.

Using the assumption that we strive to maintain all the formulas in the PDNF, we can try to estimate the time complexity of the whole soundness verification algorithm based on the parameters of the source DPN. We consider the worst case, when the DPN is bounded. Although finding a PDNF representation for any formula is a complex task, its effect on the overall complexity is negligible; thus, it is not included in the complexity formula. Let $\mathcal{N}$ be a DPN that we want to verify for soundness. Let P and T be the numbers of places and transitions in $\mathcal{N}$, respectively. Let M be the number of reachable markings in $\mathcal{N}$. Let V and $Const$ be the numbers of variables and constants in $\mathcal{N}$, respectively. Let $A = 2^{4V(2Const+3V)}$. We estimate the number of states in $LTS_{\mathcal{N}_{R_\tau}}$ as $M \times A$. Then, the time complexity can be estimated as $O(C(M^2 \times A^3 \times (P + 2^{2^{2A}})))$. If we estimate C as $A \times T$, as it was discussed above, then the time complexity becomes $O(M^2 \times A^4 \times T \times (P + 2^{2^{2A}})))$.

Based on the complexity formula, we can state that the number of reachable markings and the number of variables and constants in DPN formulas contribute most to the overall algorithm complexity. However, we should note that there was probably a high overestimation for the number of refinement iterations (that is very small in real cases; much less than $A \times T$) and for the number of formulas and the length of formulas (in practice, many variables are isolated and do not affect other ones; thus, a big portion of formulas would never occur in labeled transition systems and DPNs). The proposed algorithm has been tested both on synthetic and real-life process models (see Section 8), and the results show that the algorithm is applicable for process models of small and medium sizes despite its high time complexity in the worst case. Nevertheless, checking soundness is known to be a complex task even for classical Petri nets. Recent estimates of soundness verification time complexity for classical Petri nets can be found in [34,35].

Appendix B. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jlamp.2024.100953.

References

[1] M. Weske, Introduction, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 3–23.
[2] W.M.P. van der Aalst, Verification of workflow nets, in: P. Azéma, G. Balbo (Eds.), Application and Theory of Petri Nets 1997, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 407–426.
[3] W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K.A. de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, A.J.M.M. Weijters, Prom 4.0: comprehensive support for real process analysis, in: J. Kleijn, A. Yakovlev (Eds.), Petri Nets and Other Models of Concurrency – ICATPN 2007, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 484–494.
[4] A. Berti, S.J. van Zelst, W. van der Aalst, Process mining for python (pm4py): bridging the gap between process- and data science, arXiv:1905.06169, 2019.
[5] W.M. van der Aalst, The application of petri nets to workflow management, J. Circuits Syst. Comput. 8 (1998) 21–66.
[6] N. Sidorova, C. Stahl, N. Trčka, Soundness verification for conceptual workflow nets with data: early detection of errors with the most precision possible, Inf. Syst. 36 (7) (2011) 1026–1043, https://doi.org/10.1016/j.is.2011.04.004.
[7] M. de Leoni, W.M.P. van der Aalst, Data-aware process mining: discovering decisions in processes using alignments, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Association for Computing Machinery, New York, NY, USA, 2013, pp. 1454–1461.
[8] F. Mannhardt, M. de Leoni, H.A. Reijers, W.M.P. van der Aalst, Balanced multi-perspective checking of process conformance, Computing 98 (4) (2016) 407–437, https://doi.org/10.1007/s00607-015-0441-1.
[9] M. de Leoni, P. Felli, M. Montali, A holistic approach for soundness verification of decision-aware process models, in: J.C. Trujillo, K.C. Davis, X. Du, Z. Li, T.W. Ling, G. Li, M.L. Lee (Eds.), Conceptual Modeling, Springer International Publishing, Cham, 2018, pp. 219–235.
[10] P. Felli, M. de Leoni, M. Montali, Soundness verification of decision-aware process models with variable-to-variable conditions, in: ACSD 2019, 2019, pp. 82–91.
[11] P. Felli, M. Montali, S. Winkler, Soundness of data-aware processes with arithmetic conditions, in: X. Franch, G. Poels, F. Gailly, M. Snoeck (Eds.), Advanced Information Systems Engineering, Springer International Publishing, Cham, 2022, pp. 389–406.
[12] A. Awad, G. Decker, N. Lohmann, Diagnosing and repairing data anomalies in process models, in: S. Rinderle-Ma, S. Sadiq, F. Leymann (Eds.), Business Process Management Workshops, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 5–16.
[13] F. Mannhardt, Multi-perspective process mining, Ph.D. thesis, Eindhoven University of Technology, 2018.
[14] G. Liu, Time Petri Nets and Time-Soundness, Springer Nature Singapore, Singapore, 2022, pp. 203–236.
[15] J.B. Kruskal, The theory of well-quasi-ordering: a frequently discovered concept, J. Comb. Theory, Ser. A 13 (3) (1972) 297–305, https://doi.org/10.1016/0097-3165(72)90063-5, https://www.sciencedirect.com/science/article/pii/0097316572900635.
[16] A. Finkel, P. Schnoebelen, Well-structured transition systems everywhere!, Theor. Comput. Sci. 256 (1) (2001) 63–92, https://doi.org/10.1016/S0304-3975(00)00102-X.
[17] L.E. Dickson, Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors, Am. J. Math. 35 (4) (1913) 413–422.
[18] A. Tarski, A decision method for elementary algebra and geometry, J. Symb. Log. 14 (3) (1949) 188, https://doi.org/10.2307/2267068.
[19] L. Khachiyan, Fourier–Motzkin Elimination MethodFourier–Motzkin Elimination MethodFourier–Motzkin Elimination Method, Springer US, Boston, MA, 2009, pp. 1074–1077.
[20] S. Lasota, Decidability border for petri nets with data: Wqo dichotomy conjecture, in: F. Kordon, D. Moldt (Eds.), Application and Theory of Petri Nets and Concurrency, Springer International Publishing, Cham, 2016, pp. 20–36.
[21] L. de Moura, N. Bjørner, Z3: an efficient smt solver, in: C.R. Ramakrishnan, J. Rehof (Eds.), TACAS, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 337–340.
[22] S. Morimoto, A survey of formal verification for business process modeling, in: M. Bubak (Ed.), Computational Science – ICCS 2008, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 514–522.
[23] S. Sadiq, M. Orlowska, W. Sadiq, C. Foulger, Data flow and validation in workflow modelling, in: ADC 2004 Proceedings, 02 2004.
[24] E. Bazhenova, F. Zerbato, B. Oliboni, M. Weske, From bpmn process models to dmn decision models, Inf. Syst. 83 (2019) 69–88, https://doi.org/10.1016/j.is.2019.02.001, https://www.sciencedirect.com/science/article/pii/S0306437918300231.
[25] D. Calvanese, M. Dumas, Ü. Laurson, F.M. Maggi, M. Montali, I. Teinemaa, Semantics and analysis of dmn decision tables, in: BPM, 2016.
[26] K. Batoulis, M. Weske, A tool for checking soundness of decision-aware business processes, in: BPM, 2017.
[27] D. Knuplesch, L.T. Ly, S. Rinderle-Ma, H. Pfeifer, P. Dadam, On enabling data-aware compliance checking of business process models, in: J. Parsons (Ed.), Conceptual Modeling – ER 2010, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 332–346.
[28] X. Tao, G. Liu, B. Yang, C. Yan, C. Jiang, Workflow nets with tables and their soundness, IEEE Trans. Ind. Inform. 16 (3) (2020) 1503–1515, https://doi.org/10.1109/TII.2019.2949591.
[29] J. Song, G. Liu, Model checking ofworkflow nets with tables and constraints, arXiv:2307.03685, 2023.
[30] P. Felli, M. de Leoni, M. Montali, Soundness verification of data-aware process models with variable-to-variable conditions, Fundam. Inform. 182 (1) (2021) 1–29, https://doi.org/10.3233/FI-2021-2064.
[31] P. Felli, M. Montali, S. Winkler, Linear-time verification of data-aware dynamic systems with arithmetic, Proc. AAAI Conf. Artif. Intell. 36 (5) (2022) 5642–5650, https://doi.org/10.1609/aaai.v36i5.20505, https://ojs.aaai.org/index.php/AAAI/article/view/20505.
[32] V. Weispfenning, The complexity of linear problems in fields, J. Symb. Comput. 5 (1) (1988) 3–27, https://doi.org/10.1016/S0747-7171(88)80003-8, https://www.sciencedirect.com/science/article/pii/S0747717188800038.
[33] D.B. Johnson, Finding all the elementary circuits of a directed graph, SIAM J. Comput. 4 (1975) 77–84, https://api.semanticscholar.org/CorpusID:15381078.
[34] G. Liu, Pspace-completeness of the soundness problem of safe asymmetric-choice workflow nets, in: R. Janicki, N. Sidorova, T. Chatain (Eds.), Application and Theory of Petri Nets and Concurrency, Springer International Publishing, Cham, 2020, pp. 196–216.
[35] M. Blondin, F. Mazowiecki, P. Offtermatt, The complexity of soundness in workflow nets, in: Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '22, Association for Computing Machinery, New York, NY, USA, 2022.