A Holistic Approach for Soundness Verification of Decision-Aware Process Models

Massimiliano de Leoni[1], Paolo Felli[2], and Marco Montali[2]

[1] Eindhoven University of Technology, the Netherlands
m.d.leoni@tue.nl
[2] Free University of Bozen-Bolzano,
{pfelli,montali}@inf.unibz.it

Abstract. The last decade has witnessed an increasing transformation in the design, engineering, and mining of processes, moving from a pure control-flow perspective to more integrated models where also data and decisions are explicitly considered. This calls for methods and techniques able to ascertain the correctness of such integrated models. Differently from previous approaches, which mainly focused on the local interplay between decisions and their corresponding outgoing branches, we introduce a holistic approach to verify the end-to-end soundness of a Petri net-based process model, enriched with case data and decisions. In addition, we present an effective, implemented technique that verifies soundness by translating the input net into a colored Petri net with bounded color sets, on which standard state space analysis techniques are subsequently applied. Experiments on real life illustrate the relevance and applicability in real settings.

1 Introduction

The fundamental problem of verifying the correctness of business process models has been traditionally tackled by exclusively considering the control flow perspective, namely by only considering the ordering relations among activities present in the model. In this setting, one of the most investigated formal notions of correctness is that of *soundness*, originally introduced by van der Aalst in the context of workflow nets (a class of Petri nets that is suitable to capture the control flow of business processes) [2]. Intuitively, soundness guarantees the two good properties of "possibility of clean termination" and of "absence of deadlocks". This ensures that a process instance (i) always has the possibility of reaching its completion; (ii) when it does so, no running concurrent thread is still active; and (iii) all parts of the process can be executed, i.e., the process does not contain dead activities that are impossible to enact in some scenario.

The control-flow perspective is certainly of high importance as it can be considered the main process backbone; however, many other perspectives should also be taken into account. In fact, the last decade has witnessed an increasing transformation in the design, engineering, and mining of processes, moving from a pure control-flow perspective to more integrated models where also data and decisions are explicitly considered. This trend is also testified by the recent introduction and development of the Decision Model and Notation (DMN), an OMG standard [1]. This calls for methods and techniques able to ascertain the correctness of such integrated models, which is important not only during the design phase of the business process lifecycle, but also when it comes to decision and guard mining [13, 14], as well as compliance checking [12].

This work introduces a holistic, formal and operational approach to verify the end-to-end soundness of Data Petri nets (DPNs) [13], which models the order with which activities can occur (a.k.a. control flow) as well as the decision and the data of the process. Their solid formal foundation allows DPNs to unambiguously extend soundness to incorporate the decision perspective.

In the general case, verifying soundness of DPNs is undecidable due to the presence of case data and the possibility of manipulating them so as to reconstruct Turing-powerful computational devices. This applies, in particular, when case data can be updated using arithmetical operators. We isolate here a decidable class of DPNs that employs both non-numerical and numerical domains, and is expressive enough to capture data-aware process models equipped with S-FEEL DMN decisions [1], such as those recently proposed in [4, 3]. Such DPNs cannot be directly analyzed algorithmically, since they induce an infinite state space even when employing bounded workflow nets. Hence, inspired to predicate abstraction [7], and in particular the approach of [12], we present an effective (i.e., sound, complete) approach for verifying soundness by translating the input net into a colored Petri net (CPN) with bounded color domains, which can then be analyzed with conventional state space exploration techniques. This has been implemented as a plug-in for the well-established ProM process mining framework.

The paper is organized as follows. In Section 2 we discuss related work. In Section 3 we recall DPNs, define their execution semantics and soundness. Section 4 illustrates our approach for translating a DPN into a corresponding CPN. Section 5 discusses the ProM implementation and reports on experiments on real-life processes, either designed by hand or obtained though manual-design and process discovery combined, showing that the technique is applicable to real-life case studies. Section 6 concludes the paper.

For space reasons, full proofs and additional technical details of this work are given in an extended technical report [8].

2 Related Work

Within the field of database theory, many approaches have been proposed to formalize and verify very sophisticated variants of data-aware processes [5], also considering data-aware extensions of soundness [15]. However, such works are mainly foundational, and do not currently come with effective verification algorithms and implementations. Within the field of business process management and information systems engineering, a plethora of techniques and tools exists for verifying soundness of process models that only capture the control-flow perspective, but not much research has been carried out to incorporate the data and decision perspective in the analysis. Sidorova et al. proposed a conceptual extension of workflow nets, equipped with an abstract, high-level data model [18]. Here, activities read and write entire guards instead of reading and writing data variables that affect the satisfaction of guards. Although simple (reading and writing guards is equivalent to reading and writing boolean values), this is not realistic: as testified by modern process modeling notations such as BPMN and DMN, the data perspective requires (at least) data variables and full-fledged guards and updates. [6] focuses on single DMN tables to verify whether they are correct or contain inconsistent, missing or overlapping rules. This certainly fits in the context of data-aware soundness, although the analysis is only conducted locally to decision points, and local forms of analysis do not suffice to guarantee good behavioral properties of the entire process

[12]. A similar drawback is also present in [4], where decisions are considered either individually or in relation to their immediate outgoing sequence flows. Further, as mentioned in the introduction, soundness verification plays a key role in decision and guard mining [13, 14]. Here, an initial process model is discovered by first considering only the control-flow perspective and then by enriching its decision points with decisions and conditions which are again inferred from the event data in the log. However, this "local enrichment" clearly cannot guarantee the soundness of the process, so soundness verification techniques must be employed to discard incorrect results.

The two closest works to our contribution are [3, 12]. In [3], the authors consider the interplay between BPMN and DMN, providing different notions of data-aware soundness on top of such process models (once the BPMN component is encoded into a Petri net, which can be seamlessly tackled by known techniques [9, 11]). Our representation of processes is expressive enough to capture the soundness properties in [3], therefore our verification technique based on an encoding into CPNs guarantees that the obtained CPN is behaviorally equivalent to the input DPN and, in turn, that the notion of soundness we introduce as well as all variants of soundness defined in [3] can be actually verified using this approach. Additional details can be found in [8]. In [12], the authors introduce an abstraction approach which shares the same spirit of our technique: it is faithful (i.e., it preserves properties), and it is based on the idea of considering only boundedly many representative values in place of the entire data domains. There are however four fundamental differences. First, in [12] abstractions are used to shrink the state space of the analysis, not to tame the infinity brought by the presence of data and the possibility of updating them. Second, [12] defines abstract process graphs that do not come with a formal execution semantics, hence not allowing to formally prove the correctness of the abstraction. Since our approach is expressive enough to capture the model of [12] (see [8]), our correctness result in Theorem 1 can be also lifted to [12]. Third, [12] focuses on compliance checking against LTL-based rules, which cannot capture soundness (in particular, the "possibility of termination", as this has an intrinsic branching nature); on the other hand, our encoding produces a CPN that is behaviorally equivalent to the original DPN, so it preserves all the runs and thus all LTL properties. Finally, while [12] translates the problem of compliance checking into a temporal model checking problem, we resort to Petri net-based techniques.

3 Syntax and Semantics of DPNs

We provide the necessary background on the DPN model [13], then providing for the first time a full account of their execution semantics, and lifting the standard notion of soundness to their richer, data-aware setting. We first define the notion of domain for case variables, assuming an infinite universe of possible values $\mathcal{U}$.

Definition 1 (Domain). *A* domain *is a couple $\mathcal{D} = \langle \Delta_{\mathcal{D}}, \Sigma_{\mathcal{D}} \rangle$ where $\Delta_{\mathcal{D}} \subseteq \mathcal{U}$ is a set of possible values and $\Sigma_{\mathcal{D}}$ is the set of binary predicates defined on $\Delta_{\mathcal{D}}$.*

Consider a set of domains $\mathfrak{D}$, and in particular the notable domains $\mathcal{D}_{\mathbb{R}} = \langle \mathbb{R}, \{<, >, =, \neq\} \rangle$, $\mathcal{D}_{\mathbb{Z}} = \langle \mathbb{Z}, \{<, >, =, \neq\} \rangle$, $\mathcal{D}_{bool} = \langle \{True, False\}, \{=, \neq\} \rangle$, $\mathcal{D}_{\mathbb{S}} = \langle \mathbb{S}, \{=\} \rangle$ for real numbers, integers, booleans, and strings ($\mathbb{S}$ denotes the set of all strings).

Given a set of variables V, for each $v \in V$ we write v^r or v^w to denote that the variable v is read or written, hence we consider two distinct sets $V^r = \{v^r \mid v \in V\}$

and $V^w = \{v^w \mid v \in V\}$. When we do not which to distinguish, we still use the symbol v to denote any member of $(V^r \cup V^w)$. Moreover, in order to talk about their possible values, we need to associate domains to variables. If a variable v is assigned a domain $\mathcal{D} = \langle \Delta_\mathcal{D}, \Sigma_\mathcal{D} \rangle$, for brevity we denote by $v_\mathcal{D}$ the corresponding *typed variable*, that is a shorthand to specify that v can only assume values in $\Delta_\mathcal{D}$.

Definition 2 (Guards). *Given a set of variables V with associated domains, the set of possible* guards $\Phi(V)$ *is the largest set containing the following:*
- v *iff* $v \in (V^r \cup V^w)$*;*
- $v_\mathcal{D} \odot \Delta_\mathcal{D}$ *iff* $v \in (V^r \cup V^w)$ *and* $\odot \in \Sigma_\mathcal{D}$*;*
- $\phi_1 \wedge \phi_2$ *and* $\phi_1 \vee \phi_2$ *iff* ϕ_1 *and* ϕ_2 *are guards in* $\Phi(V)$.

Hence, a guard can be any conjunction or disjunction of atoms of the form variable-operator-constant, whereas it is not possible to have guards of the form variable-operator-variable, which is left as future work. A *variable assignment* is a function $\beta : (V^r \cup V^w) \to \mathcal{U} \cup \{\perp\}$ assigning a value to read and written variables, with the restriction that $\beta(v)$ is a possible value for v: if $v_\mathcal{D}$ is the corresponding typed variable then $\beta(v) \in \Delta_\mathcal{D}$. The symbol $\perp$ is used to denote a null value, i.e., that the variable is not set. Given a variable assignment β and a guard ϕ, we say that ϕ is satisfied by the variable assignment β, written $\phi_{[\beta]} = \texttt{true}$, iff:
- if $\phi = v$ then $\perp \neq \beta(v)$;
- if $\phi = v \odot k$, then $\odot(x, k)$ for $x = \beta(v)$;
- if $\phi = \phi_1 \wedge \phi_2$ then $\phi_{1[\beta]} = \texttt{true}$ and $\phi_{2[\beta]} = \texttt{true}$;
- if $\phi = \phi_1 \vee \phi_2$ then $\phi_{1[\beta]} = \texttt{true}$ or $\phi_{2[\beta]} = \texttt{true}$.

In words, a guard is satisfied by evaluating it after assigning values to read and written variables, as specified by β. A *state variable assignment*, abbreviated hereafter as SV assignment, is instead a function $\alpha : V \to \mathcal{U} \cup \{\perp\}$ which assigns values to each variable $v \in V$, with the restriction that $\alpha(v_\mathcal{D}) \in \Delta_\mathcal{D}$. Note that this is different from variable assignments, which are defined over $(V^r \cup V^w)$. We can now define our DPNs.

Definition 3 (Data Petri Net). *A DPN* $\mathcal{N} = (P, T, A, V, dom, \alpha_I, read, write, guard)$ *is a Petri net* (P, T, A) *with places* P*, transitions* T *and arcs* A*, where:*
- V *is a finite set of process variables;*
- dom *is a function assigning a domain* $\mathcal{D}$ *to each* $v \in V$*;*
- α_I *is the initial SV assignment;*
- $read : T \to pwr(V)$ *returns the set of variable* read *by a transition;*
- $write : T \to pwr(V)$ *returns the set of variable* written *by a transition;*
- $guard : T \to \Phi(V)$ *associates each transition* t *with a guard, so that* v^r *appears in* $guard(t)$ *only if* $v \in read(t)$*, and* v^w *appears in* $guard(t)$ *only if* $v \in write(t)$.

3.1 Execution Semantics

By considering the usual semantics for the underlying Petri net together with the guards associated to its transitions, we define the resulting data-aware execution semantics for DPNs. The set of possible states of any such DPN is formed by all pairs (M, α) where:
- $M \in \mathbb{B}(P)$[3], i.e., is the marking of the Petri net (P, T, A), and
- α is a SV assignment, defined as in the previous section.

[3] $\mathbb{B}(X)$ indicates the set of all multisets of elements of X

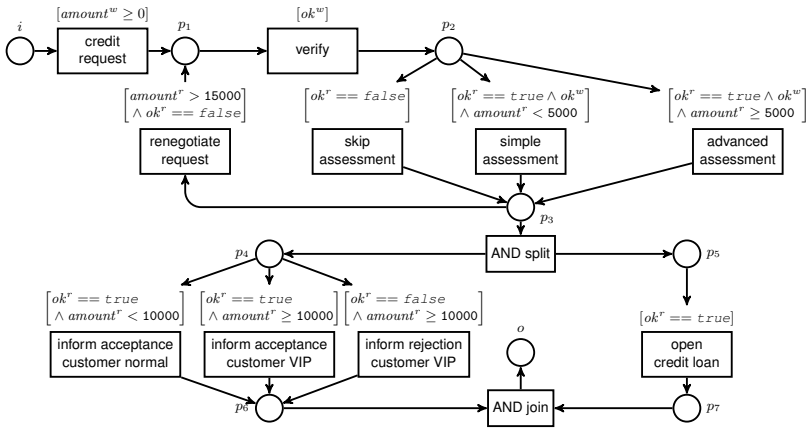

Fig. 1. Our working example of a DPN. Writing operations exist every time guards mention ok^w or $amount^w$. Terms ok^w in the guards of verify, simple assessment and advanced assessment explicitly indicate that the variable is written and can take on either `true` or `false`.

In any state, zero or more transitions of a DPN may be able to fire. Firing a transition updates the marking, reads the variables specified in $read(t)$ and selects new values for those in $write(t)$. We model this through a variable assignment β which assigns a value to *all and only* those variables that are read or written. A couple (t, β) is called *transition firing*, said to be legal when consistent with the current state.

Definition 4 (Legal transition firing). *A DPN $\mathcal{N}$ as above evolves from state (M, α) to state (M', α') via the transition firing (t, β) with $guard(t) = \phi$ if and only if:*

- $\beta(v^r) = \alpha(v)$ *if* $v \in read(t)$*: β assigns values as α for read variables;*
- *the new SV assignment is s.t.* $\alpha'(v) = \begin{cases} \alpha(v) & \text{if } v \notin write(t), \\ \beta(v^w) & \text{otherwise}; \end{cases}$
- β *is* valid*, namely* $\phi_{[\beta]} = \texttt{true}$*: the guard is satisfied under β;*
- $(M(p) > 0)$ *for any* $p \in P$ *such that* $(p, t) \in A$*: each input place of t has tokens;*
- *the new marking is calculated as usual, namely* $M \xrightarrow{t} M'$.

We denote this by writing $(M, \alpha) \xrightarrow{t,\beta} (M', \alpha')$, and extend the notation to sequences $\sigma = \langle (t^1, \beta^1), \ldots, (t^n, \beta^n) \rangle$ of n legal transition firings, called *traces*, and denote the corresponding *run* by $(M^0, \alpha^0) \xrightarrow{t^1,\beta^1} (M^1, \alpha^1) \xrightarrow{t^2,\beta^2} \ldots \xrightarrow{t^n,\beta^n} (M^n, \alpha^n)$ or equivalently by $(M^0, \alpha^0) \xrightarrow{\sigma} (M^n, \alpha^n)$. By restricting to the initial marking M_I and the initial variable assignment α_I, we define the traces of $\mathcal{N}$ as the set of sequences σ as above, of any length, such that $(M_I, \alpha_I) \xrightarrow{\sigma} (M, \alpha)$ for some $M \in \mathbb{B}(P)$ and α.

Example 1. Figure 1 shows a DPN modelling a process for managing credit requests and corresponding loans. There are two case variables, $amount$ and ok, respectively representing the requested amount and whether the credit request is accepted or not. The process starts by acquiring the amount of the credit request (writing $amount$), which must be positive. Then a verification step is performed, determining whether to accept or reject the request (writing ok). If rejected, a new verification may be performed provided that $amount$ exceeds 15000 euros. If accepted, depending on the amount, a simple or advanced assessment is performed (updating ok). The second phase of the process deals, concurrently, with the opening of a loan (for positive assessments) and with a communication sent to the customer, which depends again on the case variables. □

3.2 Data-aware Soundness

We now lift the standard notion of soundness [2] to the case of DPNs. This notion requires not only to quantify over the markings of the net, but also on the assignments of its case variables, thus making soundness *data-aware* (we use 'data-aware' to distinguish our notion from the one of decision-aware soundness in the literature). In what follows, we write $(M, \alpha) \xrightarrow{*} (M', \alpha')$ to implicitly quantify *existentially* on traces σ, and denote by M_F the *final marking* of a DPN: it is the marking that, when reached, indicates the conclusion of the execution of the process instance.

Definition 5 (Data-aware soundness). *A DPN is* data-aware sound *iff:*
P1: $\forall (M, \alpha).\ ((M_I, \alpha_I) \xrightarrow{*} (M, \alpha) \Rightarrow \exists \alpha'.\ (M, \alpha) \xrightarrow{*} (M_F, \alpha'))$
P2: $\forall (M, \alpha).\ (M_I, \alpha_I) \xrightarrow{*} (M, \alpha) \wedge M \geq M_F \Rightarrow (M = M_F)$
P3: $\forall t \in T.\ \exists M_1, M_2, \alpha_1, \alpha_2, \beta.\ (M_I, \alpha_I) \xrightarrow{*} (M_1, \alpha_1) \xrightarrow{t,\beta} (M_2, \alpha_2)$

The first condition checks the reachability of the output state, i.e., that it is *always* possible to reach the final marking of $\mathcal{N}$ by suitably choosing a continuation of the current run (i.e., legal transitions firings). The second condition captures that the output state is always reached in a "clean" way, i.e., without having *in addition* other tokens in the net. The third condition verifies the absence of dead transitions, where a transition is considered dead if there is no way of assigning the case variables so as to enable it.

Example 2. The DPN in Figure 1 is unsound. Suppose that the verification and assessment steps assign ok to `false`. Once the execution assigns a token to p_3, and the following AND-split transition is fired, two tokens are produced, in p_4 and p_5. Since the guard of open credit loan is false, token p_5 cannot be consumed and it is not possible to properly complete the execution. Also, if $amount$ is less than 10000 an analogous situation occurs for the token in p_4. □

4 Soundness Verification

In this section we propose an effective technique to check soundness of any DPN, by encoding it into a corresponding Colored Petri net (CPN) that enjoys two key properties: it employs finite color domains that suitably abstract away the source of infinity coming from the manipulation of data, but still preserves soundness, i.e., the original DPN is data-aware sound if and only if its corresponding CPN satisfies a corresponding, effectively checkable variant of soundness. This allows us to employ conventional Petri-net state space analysis techniques to ascertain data-aware soundness of DPNs. Indeed, the reachability graph may still be infinite, although the source of such infiniteness is handled by native Petri nets state-space analysis (cf. coverability graph or analogous well known techniques [2]). CPNs are an extension to DPNs that have a better support for time and resource [10], and can also be simulated through CPN Tools [16], making it possible to build on existing techniques to verify soundness. Differently from DPNs, where variables are global, CPNs encode the data aspects in the tokens, attaching them a data value: the *color*. Each place in a CPN can contain tokens of one type, called *color set* of the place.

Definition 6 provides a simplified definition of a CPN, which is enough to cover all the cases necessary in this paper. Note that tokens can also be associated with no values: in this case we introduce the color set $\bullet = \{\circ\}$, so that places with color set $\bullet$ can only contain tokens with value $\circ$, corresponding to black tokens in normal Petri nets. Similarly, variable $v_\bullet$ is a special variable that can only take on one value, i.e., $\circ$.

Definition 6 (CPN). *A CPN is a tuple* $(P, T, A, \Sigma, V, C, N, E, G, I)$ *where:*

- (P, T, A) *is a Petri net with places* P*, transitions* T *and direct arcs* A*;*
- Σ *is a set of color sets defined within the CPN model and* V *a set of variables;*
- $C : P \to \Sigma \cup \{\bullet\}$ *is a* color function *mapping each place to a color set in* $\Sigma \cup \{\bullet\}$*;*
- $N : A \to (P \times T) \cup (T \times P)$ *is a* node function *that maps each arc to either a pair* (p, t) *indicating that the arc is between a place* $p \in P$ *and transition* $t \in T$*, or* (t, p) *indicating that the arc connects* $t \in T$ *to* $p \in P$*;*
- $E : A \to V \cup \{v_\bullet\}$ *is an arc expression function, assigning variables to arcs;*
- $G : T \to \Phi(V)$ *is a guard function that maps each transition* $t \in T$ *to an expression* $G(t)$ *with the additional constraint that* $G(t)$ *can only employ variables used to annotate the arcs entering in* t*:* $G(t) \in \bigcup_{a \in A.N(a)=(p,t)} E(a)$*;*
- $I : P \to \mathbb{B}(\Sigma \cup \{\bullet\})$ *is an initialization function assigning color values to places. For* $p \in P$*,* $I(p)$ *returns the token colors initially present in* p*, with* $I(p) \in C(p)$*.*

In general, for any arc $a \in A$, the expression $E(a)$ can be more complex than just being a single variable. However, this simplification covers all the expressions we consider here. The concept of a marking M can be easily extended to CPNs as a function $M : P \to \mathbb{B}(\Sigma \cup \{\bullet\})$ such that $M(p)$ is a multiset of elements, each of which is the data (a.k.a. color in CPN) associated to a different token in p.

A CPN run is of the form $M^0 \xrightarrow{t^1,\gamma^1} M^1 \xrightarrow{t^2,\gamma^2} \dots \xrightarrow{t^n,\gamma^n} M^n$ where $M^0 = I$ and $\gamma^i : V \to (\Sigma \cup \{\circ\})$ is the so-called *binding* function, defined over the set of variables of the arcs entering transition t^i, for all $i \in [1, n]$. When firing each transition t^i from marking M^i, only legal bindings are possible. Binding γ^i is legal for t^i in M^i if:[4]

1. The binding considers exactly those variables that annotate the input arcs of t^i: $dom(\gamma^i) = \bigcup_{a \in A, p \in {}^\bullet t^i . N(a)=(p,t^i)} E(a)$.
2. If the same variable v annotates multiple input arcs of t^i, then $\gamma^i(v)$ consistently assigns v to a value that is carried by at least one token present in each of the source places of such arcs, according to M^i: for each $v \in dom(\gamma^i)$ and for each $p \in P$ s.t. there exists $a \in A$ with $N(a) = (p, t^i)$ and $E(a) = v$, we have $\gamma^i(v) \in M^i(p)$.
3. The guard of t is true when variables are substituted as per γ^i: $\phi_{[\gamma^i]} = \texttt{true}$.

Firing t^i with γ^i in marking M^i leads to a marking M^{i+1}, constructed as follows:[5]

$$M^{i+1}(p) = \begin{cases} M^i(p) & \text{if } p \notin ({}^\bullet t^i \cup t^{i\bullet}) \\ M^i(p) \setminus [\gamma^i(E(a_{(p,t^i)}))] & \text{if } p \in {}^\bullet t^i \\ M^i(p) \uplus [\gamma^i(E(a_{(t^i,p)}))] & \text{if } p \in t^{i\bullet} \end{cases}$$

We denote this by writing $M^i \xrightarrow{t^i,\gamma^i} M^{i+1}$, which is legal if γ^i is a valid binding of t^i. As for DPN runs, a CPN run is legal if it is a sequence of legal firings.

4.1 Translating DPNs into Colored Petri Nets

We now illustrate how a DPN $\mathcal{N} = (P, T, A, V, dom, \alpha_I, read, write, guard)$ can be converted into a CPN $\mathcal{N}^c = (P^c, T^c, A^c, \Sigma^c, V^c, C^c, N^c, E^c, G^c, I^c)$ that suitably

[4] In the remainder, given a transition $t \in T$, we denote by ${}^\bullet t = \{p \in P. \exists a \in A. N(a) = (p, t)\}$ and $t^\bullet = \{p \in P. \exists a \in A. N(a) = (t, p)\}$ the usual notions of preset and postset of t.

[5] Notation $a_{(p,t^i)}$ denotes the arc $a \in A$ s.t. $N(a) = (p, t^i)$ and cannot be employed if such an arc does not exist. The set-difference operator $\setminus$ is overridden for multisets: given two multisets A and B, for each element $x \in B$ with cardinality $b_x > 0$ in B and cardinality $a_x \geq 0$ in A, the cardinality of x in $B \setminus A$ is $\max(0, b_x - a_x)$; moreover, if $x \notin B$ then $x \notin B \setminus A$.

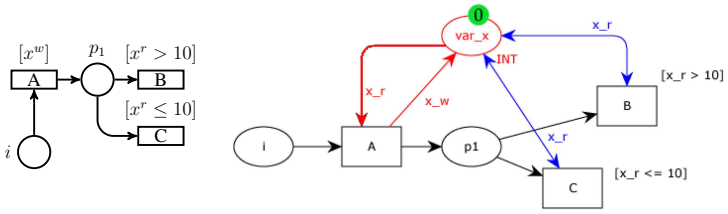

Fig. 2. Conversion of a simple DPN to CPN (left to right). The green token represents a token with value 0. Arcs without annotations are considered annotated by $v_\bullet$ and places with no color are associated with $\bullet$. Double-headed arcs stand for two arcs with same inscription in both directions.

mimics its execution semantics. Intuitively, as exemplified in Figure 2, the transitions and places of the DPN become transitions and places of the CPN. Each variable v of the DPN becomes one *variable place* associated with the color set as the variable type of v, e.g., place var_x in Figure 2 (right). These places always contain exactly one token, holding the variable's current value. Guards are as in the CPN, and if a transition writes a variable v, the token in the variable place for v is consumed and a new token is generated to model the update of v. For instance, the fact that transition A of the DPN Figure 2 (left) writes a new value for variable x (denoted x^w) is modelled in Figure 2 (right) through the two red arcs annotated with x_r and x_w, respectively entering and exiting transition A. This allows the token holding the value of v to change value when returned back to the place. A read operation is modelled as shown by the the blue arcs in Figure 2 (right), which have the same annotation, so that the token from the variable place is consumed and then put back with the same value. The initial marking of the DPN is part of the initial marking of the CPN, and each variable places is initialized with a token holding the initial value of the variable. In Figure 2 (right), the place var_x contains a token with value 0, assuming $\alpha_I(x) = 0$. We now formalize this intuition.

Places. The places of the CPN consist of all places of the DPN, plus one dedicated extra place $\xi(v)$, hereafter called *variable place*, for each DPN variable $v \in V$. Hence, $P^c = P \cup_{v \in V} \xi(v)$, where each variable place $\xi(v)$ always has one token, and precisely the one holding the current value of variable v at each step of the simulation of the CPN.
Transitions. The transitions of the CPN and DPN are the same: $T^c = T$.
Arcs. Each arc in A is preserved, and for each transition $t \in T$ and variable read and/or written in t, we add two extra arcs: $A^c = A \cup \{(t, \xi(v)), (\xi(v), t) \mid t \in T, v \in read(t) \cup write(t)\}$. The node function is defined as $N^c(a) = a$ for every $a \in A^c$.
Color sets. The CPN supports the same variable types as the DPN, and we consider the color sets $\Sigma^c = \{\mathbb{Z}, \mathbb{R}, \mathsf{bool}, \mathsf{Strings}, \bullet\}$ corresponding to the domains defined at the beginning of Section 3 for integers, reals, booleans and strings, respectively (plus $\bullet$).
Variables. For each variable $v \in V$ the CPN considers the variables v^r and v^w, i.e., $V^c = \{v^r, v^w | v \in V\} \cup \{v_\bullet\}$, where $v_\bullet$ is the special variable defined earlier.
Color functions. Recalling the shorthand notation $v_{\mathcal{D}}$ for typed variables in V, each place $p \in P_C$ is associated with a color set: if $p \in P$, i.e. it is also a place of the DPN, then $C^c(p) = \bullet$, otherwise the additional variable places are assigned the color set corresponding to the domain $\mathcal{D}$ of v. That is, $C^c(p) = \mathbb{Z}$ if there is $v_\mathbb{Z} \in V$ so that $p = \xi(v_\mathbb{Z})$, and the same for booleans, reals and strings (i.e. $\mathsf{bool}, \mathbb{R}, \mathbb{S}$, respectively).

Guards. Guards are not changed: $G^c(t) = guard(t)$ for each $t \in T^c$.
Arc expressions. The expression associated with any arc between a source node $s \in$

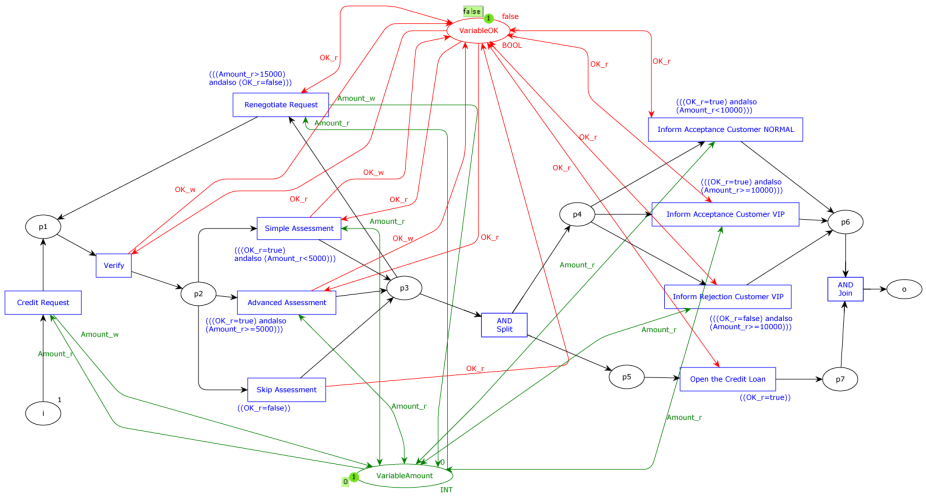

Fig. 3. Translation of DPN in Figure 1. Double-headed arcs are a shortcut to indicate that there are two arcs with the same inscription in either of directions.

$P^c \cup T^c$ and a target $d \in P^c \cup T^c$ with $(s,d) \in A^c$ is as follows. If $(s,d) \in A$ then $E^c(s,d) = v_\bullet$, otherwise:

$$E^c(s,d) = \begin{cases} v^r & \text{if } d \in T \text{ and } s = \xi(v); \\ v^r & \text{if } s \in T \text{ and } d = \xi(v) \text{ and } v \notin \mathit{write}(s); \\ v^w & \text{if } s \in T \text{ and } d = \xi(v) \text{ and } v \in \mathit{write}(s); \end{cases}$$

The first case refers to arcs of the CPN that are also present in the original DPN (e.g. belonging to the set A of arcs in DPN); the places involved in these arcs contain tokens with no value associated, which we represent by $\circ$, and thus the arcs are annotated with the $v_\bullet$ variable. The remaining cases refer to arcs connecting the variable places for each $v \in V$ to a transition $t \in T^c$. If v is written by t then the incoming arc $(\xi(v), t)$ and the outgoing arc $(t, \xi(v))$ are annotated with v^r and v^w, respectively. This allows the token holding the value of v to change value when returned back to $\xi(v)$. If instead v is not written by t then both arcs are annotated with the same inscription v^r, guaranteeing that the value of the token does not change.

Initialization. Let M_I be the initial marking of the DPN. Places that are also in the DPN take on the same number of tokens as in the DPN, whereas each variable place $\xi(v)$ is initialized with a token holding the value specified by the initial SV assignment of the DPN. Namely, $I^c(p) = [\circ^{M_I(p)}]$ if $p \in P$, i.e., p is a place in the original net; otherwise $I^c(p) = [\alpha_I(v)]$ when $p = \xi(v)$.

This translation is correct (see Section 4.3): a DPN is sound if and only if its CPN translation is sound. As already discussed, this in turn allows one to exploit standard CPN analysis techniques [2] to ascertain the properties in Definition 5.

Example 3. Figure 3 shows how the working example is translated into a CPN. The red and the green elements implement the reading and the updating of the variables ok and $amount$. □

4.2 Taming Infinity via Representatives

Although the translation is correct, it is easy to see that the reachability graph can have infinitely many distinct states. The source of infiniteness is twofold: on the one hand, the original DPN and, hence, the resulting CPN is unbounded, namely the number of tokens in some places can grow indefinitely; on the other hand, the process variables of the original DPN determine color sets in the CPN over infinite domains, in turn requiring to consider potentially infinitely many different assignments for the tokens in the variable places. Standard techniques can be employed to check whether a DPN is unbounded and consequently unsound, as proven in [2]. If the DPN is bounded, the reachability graph of the CPN resulting from the translation needs to be built, which is however still infinite if the color sets of the CPN are defined over infinite domains. The remainder illustrates that soundness checking is still possible, by replacing the infinite domains of CPN color sets with a finite set of representative elements, chosen based on the set of constants appearing in the guards.

Definition 7 (Constants of the process). *The set of* constants $\mathcal{C}_v \subset \Delta_{\mathcal{D}}$ *related to a typed variable* $v_{\mathcal{D}} \in V$ *of a DPN is defined as the set of all the values* k *such that either* $v^r \odot k$ *or* $v^w \odot k$ *appears in any guard of any* $t \in T$*, with* $\odot \in \Sigma_{\mathcal{D}}$*.*

For each variable v, observing that the set $\mathcal{C}_v$ is finite and ordered, we partition the universe $\mathcal{U}$ into $|\mathcal{C}_v| + 1$ intervals of values in which $\mathcal{U}$ can be partitioned w.r.t. v. For each interval we elect a *representative*, which can be chosen arbitrarily among the values in the interval. To correctly handle the case in which the domain $\Delta_{\mathcal{D}}$ of a variable v has no minimal or maximal elements, we define the set $\mathcal{C}_v^+$ as $\mathcal{C}_v$ with either or both of these two elements added, when needed. Hence the set of representatives for $v \in V$ is computed as $\bar{\Delta}_v := \{x \in \Delta_{\mathcal{D}} \mid x \in \mathcal{C}_v \text{ or } x = pick(x_1, x_2) \text{ for consecutive } x_1, x_2 \in \mathcal{C}_v^+\}$ where $pick$ is a deterministic function returning a representative value in the specified interval, excluding the endpoints.[6] For a given value x in the original domain $\Delta_{\mathcal{D}}$, we denote its representative as $rep(x)$, namely $rep(x) := x$ iff $x \in \mathcal{C}_v$, otherwise $rep(x) = y$ implies both $y \in (x_1, x_2)$ and $x \in (x_1, x_2)$. For $\perp$, we define $rep(\perp) := \perp$.

Let $\bar{\Delta} := \{\bar{\Delta}_{v_1}, \ldots, \bar{\Delta}_{v_q}\}$. Given a SV assignment α on the original variable domains, we define a SV assignment *restricted to* $\bar{\Delta}$ as a function $\alpha_{\bar{\Delta}} : V \to \cup_v \bar{\Delta}_v$ computed as $\alpha_{\bar{\Delta}}(v) := rep(\alpha(v))$, with the requirement that $\alpha_{\bar{\Delta}}(v) \in \bar{\Delta}_v$ for any v.

By considering a finite number of representative values, we can verify the soundness of a DPN by checking the soundness of the corresponding CPN if one restricts the values which can be assigned to each v to the finite set $\bar{\Delta}_v$, i.e., the set of representative values for v. As we are going to show, this suitably eliminates the infiniteness originating from the process data, leaving the unboundedness of the underlying net as the sole possible source of infiniteness in the reachability graph of the CPN. Such unboundedness can be detected and handled through standard Petri net analysis techniques.

To this end, we need to augment further the CPN $\mathcal{N}^c$, constructing a new CPN $\mathcal{N}^c_{\bar{\Delta}}$ which only makes use of representative values, as follows. For each variable $v \in V$ in the DPN, we add an additional place $\rho(v)$ to the set places P^c of the CPN, which is meant to represent the restricted set of possible values of v, namely $\bar{\Delta}_v$. Therefore $\rho(v)$ is assigned the same colorset as that of the variable place $\xi(v)$, and it holds one

[6] For dense domains such as real numbers such intervals are always nonempty, whereas for non-dense domains they might be empty. In this case, we consider $pick$ undefined.

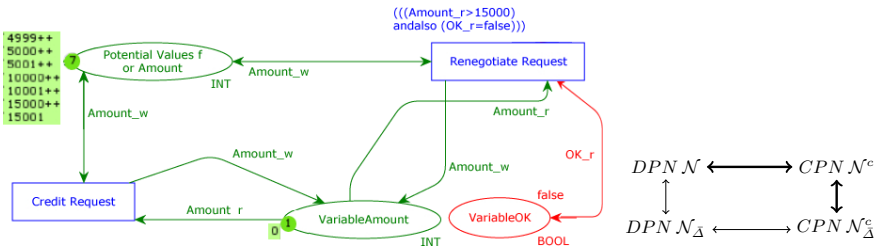

Fig. 4. (a) A fragment of $\mathcal{N}^c_{\bar{\Delta}}$, showing both additional places $\xi(amount)$ and $\rho(amount)$ for case variable $amount$ (see [8] for the complete figure). **(b)** An intuitive diagram of our approach.

token for each possible representative value in $\bar{\Delta}_v$. This is achieved by extending the initialization function of $\mathcal{N}^c$, imposing $I(\rho(v)) = \uplus_{x \in \Delta_v}[x]$. Then, for any transition $t \in T^c$ and for each variable $v \in write^c(t)$, the representative value held by one token in $\rho(v)$ is used to update the value of the token in the variable place $\xi(v)$.

More formally, we add two arcs to A^c: an arc $(t, \rho(v))$ from t to the newly-introduced place $\rho(v)$ and a second arc $(\rho(v), t)$, and define the expression function E^c so that, for transition $t \in T$ and $v \in write(t)$, $E^c((\rho(v), t)) = E^c((\rho(v), t)) = v^w$.

Example 4. Consider, e.g., the transition credit request in the model in Figure 3. This transition writes the integer variable $amount$. By inspecting all the guards, it is easy to see that the set of constants related to $amount$ is $\mathcal{C}_{amount} = \{5000, 10000, 15000\}$, from which we select the set of representatives $\bar{\Delta}_{amount} = \{4999, 5000, 5001, 10000, 10001, 15000, 15001\}$ by including an *arbitrary* value for each interval (e.g., in this case, 5001 was arbitrarily chosen to represent all the values in the interval $(5000, 10000)$). A token for each element of $\bar{\Delta}_{amount}$ is created in $\rho(amount)$. As it can be seen in Figure 4 (a), which depicts a small fragment of the resulting CPN $\mathcal{N}^c_{\bar{\Delta}}$, $\rho(amount)$ is called *Potential values for Amount* and its tokens can be used as possible values for the variable $Amount\_w$, which in this figure stands for $amount^w$. □

4.3 Correctness of the Translation

We now discuss the correctness of the approach, summarized by Figure 4(b). We need to show that the CPN $\mathcal{N}^c_{\bar{\Delta}}$ built as defined in the previous section (i.e. obtained by translating the original DPN $\mathcal{N}$ first into a CPN $\mathcal{N}^c$ and thus into $\mathcal{N}^c_{\bar{\Delta}}$) preserves soundness. This is based on the fact that the set of all legal runs of $\mathcal{N}$ and the set of all legal runs of $\mathcal{N}^c_{\bar{\Delta}}$ are in direct correspondence: for every run of $\mathcal{N}$ there exists a run of $\mathcal{N}^c_{\bar{\Delta}}$ that traverse the very same transitions and vice-versa. We call this property *trace-equivalence*.

Formally, we say that a DPN run $\tau = (M_I, \alpha_I) \xrightarrow{t^1,\beta^1} \cdots \xrightarrow{t^n,\beta^n} (M^n, \alpha^n)$ and a CPN run $\tau_c = M_{Ic} \xrightarrow{t^1_c,\gamma^1} \cdots \xrightarrow{t^n_c,\gamma^n} M^n_c$ are trace-equivalent iff $t^i = t^i_c$ for each $i \in [1, n]$, i.e. they have the same transitions (M_{Ic} is the initial marking of the CPN).

Theorem 1. *A DPN $\mathcal{N}$ and the CPN $\mathcal{N}^c_{\bar{\Delta}}$ defined as in the previous section are trace-equivalent. Moreover, the soundness properties in Definition 5 hold in $\mathcal{N}$ iff their translations hold in $\mathcal{N}^c_{\bar{\Delta}}$, hence $\mathcal{N}$ is sound iff $\mathcal{N}^c_{\bar{\Delta}}$ is sound.*

The translation mentioned in the theorem is a simple rewriting of the properties in Definition 5 in terms of states and runs of CPNs instead of those of DPNs. Proving the

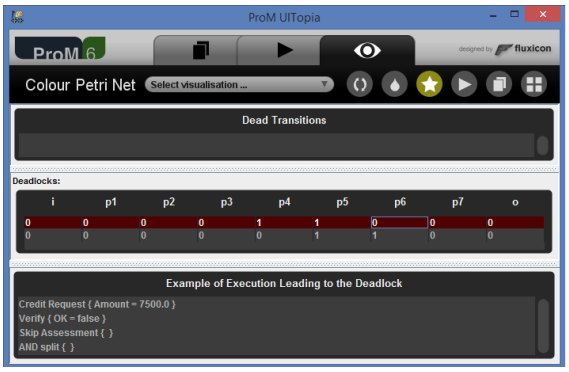

Fig. 5. Screenshot of the tool that implements the soundness-checking technique here described.

result requires articulated intermediate steps, as translating the original DPN $\mathcal{N}$ first into CPN $\mathcal{N}^c$ and consequently into $\mathcal{N}^c_{\bar{A}}$, as done in the previous section, implies not only comparing a DPN with a CPN that is syntactically very different but also handling infinite domains for variables. Due to lack of space, the proof is here omitted, but it is available in full in [8]. We give here the intuition: as illustrated in Figure 4(a), we (i) first define a DPN $\mathcal{N}_{\bar{A}}$ that employs only representative values in place of the possibly infinite domain values of variables, and prove that such $\mathcal{N}_{\bar{A}}$ is a correct abstraction of $\mathcal{N}$; then (ii) show that for any run of $\mathcal{N}_{\bar{A}}$ there exists a trace-equivalent run of $\mathcal{N}^c_{\bar{A}}$. Since the properties in which we are interested only rely on the existence of legal traces, i.e. sequences of legal transition firings, and not on the values assigned to the case variables, it follows that we can analyse $\mathcal{N}^c_{\bar{A}}$ to assess the soundness of $\mathcal{N}$.

Example 5. The CPN $\mathcal{N}^c_{\bar{A}}$ obtained for Example 1 exhibits the same deadlocks of the original net, and it is indeed unsound. While in $\mathcal{N}$ there are infinite number of values of $amount$ for which a deadlock is reached, all these are represented by a finite number of runs in $\mathcal{N}^c_{\bar{A}}$, one for each combination of representative values for $amount$ and ok. □

Note that, since our encoding produces a CPN $\mathcal{N}^c_{\bar{A}}$ that is behaviorally equivalent (i.e trace-equivalent) to the original DPN $\mathcal{N}$, the result of the previous theorem goes beyond soundness, and in particular to the different interesting properties discussed in [3] for decision-aware processes. In fact, our data-aware soundness coincides with the decision-aware soundness introduced in [3].

5 Implementation And Experiments

Our soundness-checking technique has been implemented as a Java plug-in for ProM, an established open-source framework for implementing process mining algorithms and tools (see `http://www.promtools.org/`). It supports both the PNML and the BPMN file formats for process models, and implements numerous algorithms for discovering process models that integrate the decision perspective (e.g. [13, 14]). Thanks to this, we can employ our technique to validate the soundness of models where the decision perspective is mined from event data and models can be expressed in the two mentioned notations. In particular, the soundness-checking technique is available in the ProM *nightly* build after ensuring that the ProM package *DataPetriNets* is installed. The

plug-in is named *Compute Soundness of a Data Petri Net* and takes a DPN as input. We performed a number of experiments with data-aware models of real-life processes in the literature. First, as shown in Figure 5, our implementation correctly classify the process of Example 1 as unsound, showing a possible run that leads to a deadlock.

1. Road-traffic fines example. We used the model of the real-life process for the management of road-traffic fines, which is illustrated in Figures 7 and 8 from [13]. By using our plug-in, we generated an execution that leads to a deadlock (i.e. with a token in place $pl10$), that is caused by the fact that transition *Appeal to Judge* can assign any value to variable *dismissal*. This transition is followed by a XOR split where two alternatives are possible, depending on the value of variable *dismissal*: `NIL` or `#`. However, the model does not impede *Appeal to Judge* to assign other values, e.g. `G`, causing a deadlock.

2. Road-traffic fines with guard discovery. We used the same model as at point 1 but, instead of keeping the pre-existing guards, we employed the guard discovery technique discussed in [14], which clearly does not guarantee, in general, the properties of Definition 5. The resulting model, not shown here for lack of space (see [8]), is data-aware sound. The analysis has not reported dead transitions nor deadlocks.

3. Hospital example. We checked the soundness of the data-aware models reported in Figures 13.6 and 15.6 of the Ph.D. thesis by Mannhardt [13]. Both models refer to processes that are executed within hospitals: the former is about curing patients with sepsis and the latter manages the hospital billing to patients. These models were partly hand designed and partly mined through process-discovery techniques and the analysis shows that they are data-aware sound.

The models at points 1 and 3 were analysed for deadlocks and dead transitions in a matter of seconds. The model at point 2 required 1.9 hours to produce the analysis results. This difference is due to the fact that the model at point 2 is over-fitting for what concerns the decisions. In fact, the decisions are modelled through complex guards with several atoms; as a consequence, the search space to visit grows significantly.

6 Conclusions

In this paper we have introduced a holistic, formal and operational approach to verify the end-to-end soundness of Data Petri nets. Thanks to the solid formal foundation of DPNs, we defined a notion of soundness for these nets to incorporate the decision perspective, and developed an effective technique for assessing such property that can be directly implemented on existing tools. Since our DPNs are expressive enough to capture known data-aware process such as those equipped with S-FEEL DMN decisions (employing a translation similar to that in [6] – see [8] for details), this also allows us to consider various notions of soundness from the literature and in particular those in [3]. In future work, we plan to address more sophisticated guard languages, for instance by allowing to compare variables through guards such as $(v_1^w \geq v_2^r \wedge v_1^w \neq v_3^w)$. Note however that this goes beyond DMN S-FEEL and requires more sophisticated encoding techniques, although we believe this to be a decidable setting. Further, we aim at extending our results to other data domains. This is a quite delicate task, since even minimal extensions may lead to undecidability. For instance, by enriching integer domains by a successor predicate, we immediately get an undecidability result for soundness, even in the simple case of DPNs with two case variables. Finally, we plan to

optimize the technique presented in this paper. In its current form, nondeterminism is managed *eagerly*, that is, by generated branches for possible values as soon as a variable is written. It appears instead promising to manage nondeterminism *lazily*, i.e., by postponing such choice to the moment where the variable actually appears in a guard, hence considering sets of possible representatives at the same time. This would not preserve trace-equivalence, but could still preserve data-aware soundness.

References

1. Decision model and notation (DMN) v1.1 (2016), http://www.omg.org/spec/DMN/1.1/
2. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers **8**(1), 21–66 (1998)
3. Batoulis, K., Haarmann, S., Weske, M.: Various notions of soundness for decision-aware business processes. In: Proc. of ER 2017. LNCS, vol. 10650, pp. 403–418. Springer (2017)
4. Batoulis, K., Weske, M.: Soundness of decision-aware business processes. In: Proc. of BPM Forum 2017. pp. 106–124. Springer (2017)
5. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data aware process analysis: A database theory perspective. In: Proc. of PODS 2013. ACM (2013)
6. Calvanese, D., Dumas, M., Laurson, Ü., Maggi, F.M., Montali, M., Teinemaa, I.: Semantics and analysis of DMN decision tables. LNCS, vol. 9850, pp. 217–233. Springer (2016)
7. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. **16**(5), 1512–1542 (1994)
8. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. CoRR Technical Report arXiv:1804.02316, arXiv.org e-Print archive (2018), available at https://arxiv.org/abs/1804.02316
9. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information & Software Technology **50**(12), 1281–1294 (2008)
10. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer Publishing Company, Incorporated, 1st edn. (2009)
11. Kalenkova, A.A., van der Aalst, W.M.P., Lomazova, I.A., Rubin, V.A.: Process Mining Using BPMN: Relating Event Logs and Process Models. Software & Systems Modeling **16**(4), 1019–1048 (2017)
12. Knuplesch, D., Ly, L.T., Rinderle-Ma, S., Pfeifer, H., Dadam, P.: On enabling data-aware compliance checking of business process models. LNCS, vol. 6412, pp. 332–346. Springer (2010)
13. Mannhardt, F.: Multi-perspective process mining. Ph.D. thesis, Department of Mathematics and Computer Science (2 2018), https://pure.tue.nl/ws/portalfiles/portal/90463927
14. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Decision mining revisited - discovering overlapping rules. In: CAiSE 2016. vol. 9694, pp. 377–392. Springer (2016)
15. Montali, M., Calvanese, D.: Soundness of data-aware, case-centric processes. Int. Journal on Software Tools for Technology Transfer (2016)
16. Ratzer, A.V., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K.: Cpn tools for editing, simulating, and analysing coloured petri nets. In: Proc. of ICATPN. pp. 450–462. Springer (2003)
17. Sidorova, N., Stahl, C., Trčka, N.: Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. Information Systems **36**(7), 1026–1043 (2011)