Soundness Verification of Decision-Aware Process Models with Variable-to-Variable Conditions

Paolo Felli
Faculty of Computer Science
Free University of Bozen-Bolzano
Bolzano, Italy
pfelli@inf.unibz.it

Massimiliano de Leoni
Department of Mathematics
University of Padua
Padua, Italy
deleoni@math.unipd.it

Marco Montali
Faculty of Computer Science
Free University of Bozen-Bolzano
Bolzano, Italy
montali@inf.unibz.it

***Abstract*—In recent years, there has been an increasing interest in enriching the traditional control-flow perspective of processes with additional dimensions, the data perspective being the most prominent one. At the same time, variants of Petri nets with data have been extensively studied, giving raise to a plethora of formal models with different expressive power and computational guarantees. In this work, we focus on DPNs, a data-aware extension of P/T nets where the net is enriched with data variables of different types, and transitions are guarded by formulae that inspect and update such variables. Even though DPNs are less expressive than Petri nets where data are carried by tokens, they elegantly capture business processes operating over simple case data and taking complex decisions based on these data. Notably, various techniques have been implemented to discover DPNs from event data. However, such techniques do not guarantee that the discovered DPN is actually sound. In previous work, we have then studied how to check soundness of DPNs with simple data-based guards that can only compare variables with constants. In this paper, we generalize the study of soundness to DPNs to the fundamental case where the evolution of the process depends on the comparison between the values carried by different variables through linear inequations. Our main contribution is to show decidability of soundness for this sophisticated class of DPNs. This is done by constructing an abstract state space of the net relying on the manipulation of constraints, and by showing that such an abstract state space can be faithfully and effectively inspected for soundness. The construction lends itself to be directly implemented by combining standard state-space construction methods with constraint programming techniques.**

***Keywords*-data-aware processes, soundness**

I. Introduction

In recent years, there has been an increasing interest in enriching the traditional control-flow perspective of processes with additional dimensions. In particular, the Business Process Management (BPM) field has witnessed an increasing shift from conceptual models solely focusing on the flow of activities, to so-called *multi-perspective* models linking the flow of activities to other central aspects such as data, decisions, resources, and time. In particular, the integration between (business) processes and data has been extensively studied, towards a more holistic understanding of how processes induce the evolution of data, and how data and decisions based on such data influence the process execution [4], [13].

This, in turn, has led to a flourishing line of research focused on the formalisation of integrated models for processes and data, and on the corresponding study of the boundaries of decidability for their formal analysis. Within this line, two main different paradigms can be identified. The first is about data-centric models where relational databases are enriched with processes operating over them [4], [19]. The second concentrates activity-centric models based variants of Petri nets enriched with data, data-driven decisions, and data-manipulation operations. Within this latter tradition, we consider in particular the model of *data Petri Nets* (DPNs), introduced in [10] and further studied in [8]. DPNs constitute a data-aware extension of P/T nets where the net is enriched with data variables of different types (including numerical ones), and transitions are guarded by formulae that inspect and update such variables. Even though DPNs are less expressive than Petri net-based models for data-aware processes where data are carried by tokens, such as those in [11], [16], they are particularly interesting to study for a twofold reason:

1) As argued in [8], DPNs elegantly capture the interesting class of activity-centric business processes that operate over scalar case data, and that use decision models based on the DMN S-FEEL standard [1] to route the process depending on case data [2];
2) As witnessed by [10], there are state-of-the-art process mining techniques that automatically discover DPNs from event logs, and that employ DPNs for conformance checking.

In [8], a suitable formal notion of so-called *data-aware soundness* (simply referred to as *soundness* from now on) has been introduced to provide a general criterion of correctness that builds on the well-known notion of soundness in workflow nets [18]. In particular, [8] has brought forward an abstraction technique that faithfully ascertains (data-aware) soundness for DPNs with simple data-based guards that can only compare variables with constants. Notably, the resulting framework can be used to study all the different "decision-aware" variants of soundness discussed in [2] to carry out a fine-grained study of the impact that data and decisions have on the process control-flow. This is not only of foundational interest, but has also direct practical relevance considering that the DPNs discovery algorithms do not guarantee that the inferred models are indeed correct, and thus must be complemented with a further soundness verification step.

In this work, *we generalize the study of soundness to DPNs with a large class of conditions*, by also considering the fundamental case where the evolution of the process depends on the comparison between the values carried by different variables through linear inequations. Our main contribution is to show decidability of soundness for this sophisticated class of DPNs. This is done by constructing an abstract state space of the net relying on the manipulation of (numerical) constraints, and by showing that such an abstract state space can be faithfully and effectively inspected for soundness. The construction lends itself to be directly implemented by combining standard state-space construction methods for P/T nets with constraint programming techniques. In addition, it also paves the way towards the verification of temporal properties that go beyond soundness.

The remainder of the paper is organised as follows: in Section II we discuss related work; in Section III we introduce the type of Petri nets that we will use for modelling data-aware processes and we provide their execution semantics, used to describe the (possibly infinite) executions of these nets; in Section IV we recall the notion of data-aware soundness and provide our main result, showing how this property can be effectively checked by analyzing a specific finite-state representation of the infinite executions of the process, that compactly represents them all. Finally, in Section V we discuss future work.

II. Related work

A large body of research exists to verify the soundness of process models. Most of these works only focus on the control flow [12], starting from the seminal work of van der Aalst et al. [17]. These works ignore the data decision perspective, a significant limitation as also acknowledged by Sadiq et al. [14]. Corradini et al. focus on verification of BPMN choreographies and the message exchanges among collaborating processes [6], [7], but the focus remains on the control flow.

In fact, some attempts exist to also incorporate data and decisions. Sidorova et al. proposed a conceptual extension of workflow nets, equipped with an abstract, high-level data model [15]. In fact, data are captured abstractly, and activities are assumed to read and write entire guards instead of the single data variables that affect the satisfaction of guards. This is not realistic: as testified by modern process modeling notations such as BPMN and DMN, the data perspective requires data variables and full-fledged guards and updates.

Calvanese et al. [5] focus on single DMN tables to verify whether they are correct or contain inconsistent, missing or overlapping rules. However, the analysis is only conducted locally to single decision points, whereas our notion of data-aware soundness is a global property that considers entire process' runs. A similar drawback is also present in [3]. Knuplesch et al. [9] propose a technique to verify properties expressed in LTL against models that incorporate the data perspective. Unfortunately, data-aware soundness cannot be expressed as a (set of) LTL formulas; this is specifically related to the termination aspects: the property of an option to complete cannot be represented as an LTL formula.

III. Data Petri nets

In this section we illustrate how data-aware processes are represented by data Petri nets (DPNs) [10]. Specifically, we adopt the formalisation introduced in previous work [8], which provided for the first time a full account of the syntax and semantics of these nets, allowing to lift the standard notion of soundness to their richer, data-aware setting. However, the DPNs considered in [8] are restricted to allow only transitions associated to conditions that are conjunctions or disjunctions of atoms of the form variable-operator-constant. In this paper we relax such a restriction, and generalize it to the case of variable-operator-variable.

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 on $\Delta_{\mathcal{D}}$.*

We consider a fixed set of domains, and in particular the notable domains $\mathcal{D}_{\mathbb{R}} = \langle \mathbb{R}, \{<, >, =\} \rangle$, $\mathcal{D}_{\mathbb{Z}} = \langle \mathbb{Z}, \{<, >, =\} \rangle$, $\mathcal{D}_{bool} = \langle \{\mathtt{true}, \mathtt{false}\}, \{=\} \rangle$, $\mathcal{D}_{string} = \langle \mathbb{S}, \{=\} \rangle$ which, respectively, account for real numbers, integers, booleans, and strings ($\mathbb{S}$ denotes here the infinite set of all strings). Given $\mathcal{D}$, the symbol $\perp \in \Delta_{\mathcal{D}}$ is used as a special domain value for variables, denoting an undefined value. Although our approach can be applied to arbitrary domains, from now on we restrict to the ones above.

We assume that $\Sigma_{\mathcal{D}}$ is closed under negation, namely that for every predicate $\odot$ also its complement is included. For instance, if $=$ is in $\Sigma_{\mathcal{D}}$ then also $\neq$ is in the set (the domains above are extended accordingly).

Consider a set V of *variables*. Given a variable $v \in V$ we write v^r or v^w to denote that the variable v is, respectively, read or written by an activity in the process, hence we consider two distinct sets V^r and V^w defined as $V^r = \{v^r \mid v \in V\}$ and $V^w = \{v^w \mid v \in V\}$. When we do need to distinguish, we still use the symbol v to denote any member of $(V^r \cup V^w)$. To talk about the possible values that variables may take, 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 take values in $\Delta_{\mathcal{D}}$.

Variables provide the basic building block to define logical conditions on data, used to specify conditions on the possible evolutions of the process, depending on the value of read and written variables. We call such conditions *constraints*.

Definition 2 (Constraints)**.** *Given a set of typed variables V, the set of possible* constraints $\mathcal{C}_V$ *is the largest set containing the following:*

- $v_\mathcal{D} \odot \Delta_\mathcal{D}$ *iff* $v \in (V^r \cup V^w)$ *and* $\odot \in \Sigma_\mathcal{D}$*;*
- $v_{1\mathcal{D}} \odot v_{2\mathcal{D}}$ *iff* $v_1 \in (V^r \cup V^w)$*,* $v_2 \in V^r$ *and* $\odot \in \Sigma_\mathcal{D}$*;*

In words, a constraint allows to compare a variable with a constant (among those in their associated domain $\Delta_\mathcal{D}$) or with another variable with the same domain $\mathcal{D}$. Parentheses around constraint are used throughout the paper to enhance readability. As shorthand notation, we denote by $(v \odot k)$ a constraint in which a variable, either read or written, is compared with a constant, i.e., where $v \in (V^r \cup V^w)$, $\mathcal{D}$ is the domain of v and $k \in \Delta_\mathcal{D}$. When we need to specify not only the shape of a constraint ϕ but also whether a variable is read or written, we then use the notations $(v^r \odot k)$ and $(v^w \odot k)$. Analogously, for comparing variables between them, we use the notations $(v_1 \odot v_2)$, $(v_1^r \odot v_2^r)$ and $(v_1^w \odot v_2^r)$. When the right-hand side can be either a constant or a variable, we use the symbol x. Finally, given a constraint $(v \odot x)$, we denote by $\neg(v \odot x)$ the constraint in which $\odot$ is replaced by its negation.

We use constraints to formalise the conditions that we can associated to activities in the process: each activity in the process, modelled as a transition in our net representation, is associated to a constraint called *guard*. With respect to [8], we are allowed to specify guards that compare the (written or read) value of variables not only against constants, but also against other variables. Guards allow to model conditions between the value of a variable and a constant (e.g., $\mathtt{a}^r > 0$), between the written value of a variable and the current value of another variable (e.g., $\mathtt{a}^w > \mathtt{b}^r$) or between the current value of two variables (e.g. $\mathtt{a}^r > \mathtt{b}^r$) – in these examples, $\mathtt{a}, \mathtt{b} \in V$.

Note that conjunctions and disjunction are not allowed for simplicity but without loss of generality: disjunctive guards can be mimicked by having multiple transitions from and to the same places, each having a disjunct as guard, whereas conjunctive guards can be modelled as 'non-interruptible' sequences. This also allows us to express, e.g., guards such as $\mathtt{a} \geq \mathtt{0}$ for a variable $\mathtt{a}_\mathcal{D}$ with $\geq \notin \Sigma_\mathcal{D}$ (an example will be given later). The simplification is only aimed at simplifying the technical details that follow.

A *variable assignment* is a function $\beta : (V^r \cup V^w) \to \mathcal{U} \cup \{\perp\}$, which assigns a value to read and written variables, with the restriction that $\beta(v)$ is a possible value for v, that is if $v_\mathcal{D}$ is the corresponding typed variable then $\beta(v) \in \Delta_\mathcal{D}$. Given a variable assignment β and a guard $\phi = (v \odot x)$, we say that ϕ evaluates to true when variables are substituted as per β, written $\phi_{[\beta]} = \mathtt{true}$, iff x is a constant and $\odot(k, x)$ for $k = \beta(v)$ or x is a read variable and $\odot(k_1, k_2)$ for $k_1 = \beta(v)$ and $k_2 = \beta(x)$. In other 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 β, which are defined over $(V^r \cup V^w)$.

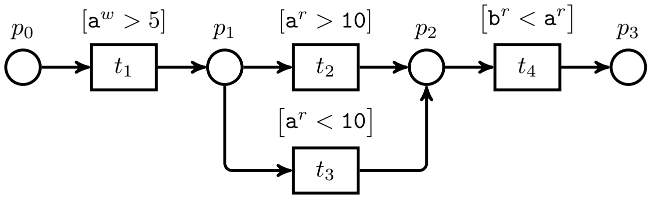


Fig. 1. A simple DPN $\mathcal{N}$, with $M_I = \{p_0\}$ and $M_F = \{p_3\}$. Variables $\mathtt{a}$ and $\mathtt{b}$ are integers and we assume $\alpha_I(\mathtt{a}) = \mathtt{0}$ and $\alpha_I(\mathtt{b}) = \mathtt{10}$. Guards of transitions are shown between squared brackets.

We can now formalise DPNs as in [8].

Definition 3 (Data Petri Net)**.** *Let V be the set of process variables. A* Data Petri Net (DPN) $\mathcal{N} = \langle P, T, F, V, \mathit{dom}, \alpha_I, \mathit{read}, \mathit{write}, \mathit{guard} \rangle$ *is a Petri net (P, T, F) with additional components, used to describe the additional perspectives of the process model:*

- *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;*
- *$\mathit{read} : T \to 2^V$ returns the set of variable* read *by a transition;*
- *$\mathit{write} : T \to 2^V$ returns the set of variable* written *by a transition;*
- *$\mathit{guard} : T \to \Phi(V)$ returns a guard associated with the transition.*

Hence, given a generic transition t, $\mathit{read}(t)$ and $\mathit{write}(t)$ denote, respectively, the set of variables in V^r and V^w that are mentioned in $\mathit{guard}(t)$. The latter is always a singleton.

Moreover, we assume that a DPN is always associated with an arbitrary *initial marking M_I* and an arbitrary *final marking M_F*. The latter, when reached, indicates the conclusion of the execution of the process instance.

Consider as en example the very simple DPN in Figure 1, where we assume $\mathtt{a}$ and $\mathtt{b}$ to have domain $\mathcal{D}_\mathbb{Z}$, and an initial SV assignment $\alpha_I(\mathtt{a}) = \mathtt{0}$ and $\alpha_I(\mathtt{b}) = \mathtt{10}$. From the initial marking $M_I = \{p_0\}$ a transition t_1 updates the value of $\mathtt{a}$ to any integer greater of $\mathtt{5}$. Then, either t_2 or t_3 are executable, depending on the current value assigned in t_1 being greater or smaller than $\mathtt{10}$. Similarly, t_4 can be executed only if the initial value of $\mathtt{b}$ is smaller than the current value of $\mathtt{a}$. This process presents several evident issues related to the possibility of marking the output place, as not any possible assignment of $\mathtt{a}$ is taken into account (for instance, for the case in which t_1 assigns value $\mathtt{10}$ to $\mathtt{a}$). Clearly, a very simplistic analysis that disregards the possible SV assignments of case variables that the process can generate at each step would still conclude that the process is (classically) sound. Hence, before being able to address this issue, we first need to characterise the possible evolutions of the process, in which data are also considered. This will be done in the next section.

A. *Execution Semantics*

By considering the usual semantics for the underlying Petri net together with the guards associated to each of its transitions, we define the resulting execution semantics for DPNs in terms of possible states and possible (legal) evolutions from a state to the next. Let $\mathcal{N}$ as above be a DPN. Then the set of possible states of $\mathcal{N}$ is formed by all pairs (M, α) where:

- $M \in \mathbb{B}(P)^1$, that is, M is the marking of the Petri net (P, T, F), and
- α is a SV assignment, defined as in the previous section.

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 a new, suitable value for those in $write(t)$. We model this through a variable assignment β for the transition (cf. the previous section), which assigns a value to all and only those variables that are read or written. A pair (t, β) is called *transition firing*.

Definition 4 (Legal transition firing). *A DPN $\mathcal{N} = \langle P, T, F, V, dom, \alpha_I, read, write, guard\rangle$ evolves from state (M, α) to state (M', α') via the transition firing (t, β) with $guard(t) = \phi$ iff:*

- *$\beta(v^r) = \alpha(v)$ if $v \in read(t)$: the variable assignment β assigns values as α for read variables;*
- *the new SV assignment α' is as α but updated as per β. It is computed as:*
$$\alpha'(v) = \begin{cases} \alpha(v) & \text{if } v \notin write(t), \\ \beta(v^w) & \text{otherwise}; \end{cases}$$
- *β is* valid*, namely $\phi_{[\beta]} = \mathtt{true}$: the guard is satisfied when we assigns value to variables according to β;*
- *each input place of t contains at least one token: $(M(p) > 0)$ for any $p \in P$ then $(p, t) \in F$;*
- *the new marking is computed according to the Petri net execution semantics as usual, denoted $M[t\rangle M'$.*

We denote a legal transition firing by writing $(M, \alpha) \xrightarrow{t,\beta} (M', \alpha')$. We also extend this definition to sequences $\sigma = \langle(t^1, \beta^1), \ldots, (t^n, \beta^n)\rangle$ of n legal transition firings, called *traces*, an 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 of a DPN $\mathcal{N}$ together with the initial variable assignment α_I, we define the legal process 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 marking M and SV assignment α, and the *trace set* of $\mathcal{N}$ as the set of process traces σ such that $(M_I, \alpha_I) \xrightarrow{\sigma} (M_F, \alpha)$ is a run for some α, where M_F is the final marking of $\mathcal{N}$.

For instance, referring to the simple DPN $\mathcal{N}$ in Figure 1, a possible one-step run fragment from of the initial state is $(\{i\}, \{\alpha_I(\mathtt{a}) = 0, \alpha_I(\mathtt{b}) = 10\}) \xrightarrow{t_1, \{\beta(\mathtt{a}^w)=7\}} (\{p_1\}, \{\alpha(\mathtt{a}) = 7, \alpha(\mathtt{b}) = 10\})$, and it is clearly the case that one such run fragment can be legally executed for any possible variable assignment for $\mathtt{a}$ that satisfies the guard of transition t_1.

[1]The notation $\mathbb{B}(X)$ indicates the set of all multisets of elements of X. However, for readability, in the case of 1-bounded DPNs we will simply denote a marking as a set of places.

B. *Data-aware soundness*

We recall here the lifting of the standard notion of soundness [17] to the the data-aware setting of DPNs, as illustrated in [8]. The resulting notion is *data-aware*, as it requires not only to quantify over the reachable markings of the net, but also on the SV assignments for its case variables. It is thus distinguished from the simpler case of decision-aware soundness in the literature.

Given a DPN $\mathcal{N}$, in what follows we write $(M, \alpha) \xrightarrow{*} (M', \alpha')$ to implicitly quantify *existentially* on traces σ in the trace set of $\mathcal{N}$. Let M' and M'' be two markings of a DPN $\mathcal{N} = \langle P, T, F, V, dom, \alpha_I, read, write, guard\rangle$. We say that M'' is greater than marking M', denoted as $M'' > M'$, iff, for any place $p \in P$ of the DPN, $M''(p) \geq M'(p)$ and there exists $p \in P$ s.t. $M''(p) > M'(p)$.

Definition 5 (Data-aware soundness). *A DPN with initial marking M_I and final marking M_F is* data-aware sound *iff all the following properties hold. By denoting as $Reach_{\mathcal{N}}$ the set of reachable states of $\mathcal{N}$, namely the set $\{(M, \alpha) \mid (M_I, \alpha_I) \xrightarrow{*} (M, \alpha)\}$, these are:*

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

The first condition imposes the reachability of an *output state*, namely a state in which the first component is the final marking on the DPN. This corresponds to requiring 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 an output state is always reached in a "clean" way, i.e., without having tokens in the net that are not in the output place. 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, through the execution of the process, so as to eventually enable it.

As an example, and as already commented, the DPN in Figure 1 is not data-aware sound because property $P1$ is false: when transition t_1 assigns a value not greater than $\mathtt{10}$ to $\mathtt{a}$ there exists no run from there which reaches an output state.

IV. Data-aware soundness of DPNs

In this section we show how to extend the general technique employed in [8] to our setting, namely to the case of more complex variable-to-variable conditions. Specifically, that work introduced an implemented technique for verifying data-aware soundness by first translating the input DPN into a DPN whit only finitely many possible variable assignments, and then into a colored Petri net (CPN) with bounded color domains. This allowed to analyse the resulting CPN with

Algorithm 1: Procedure for computing $C' \doteq C \oplus c$

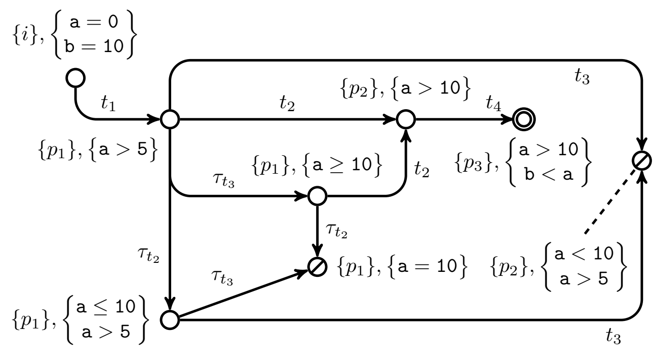
```
1  if $c = (v^r \odot x)$ then
2      $C' \leftarrow C' \cup \{(v \odot x)\}$
3  if $c = (v^w \odot x)$ then
4      $C' \leftarrow C' \cup \{(v^w \odot x)\}$
5      $C' \leftarrow$ saturate($C'$)
6      foreach $c = (v^r \odot y)$ or $c = (y \odot v^r)$ in $C'$ do
7          $C' \leftarrow C' \setminus \{c\}$
8      foreach $(v^w \odot z)$ in $C'$ do
9          $C' \leftarrow C' \setminus \{(v^w \odot z)\}$
10         if $z \neq v^r$ then
11             $C' \leftarrow C' \cup \{(v^r \odot z)\}$
12 return saturate($C'$)
```

conventional tools. Here we follow a similar approach, however limited to the first step: we show how the (possibly infinite) process traces of the DPN, as defined at the end of Section III-A, can be described finitely through a special kind of state-transition structure which we call *constraint graph*. A constraint graph can thus be regarded as a *faithful abstraction* of the original process, at least with respect to soundness: being finite-state, these can effectively be analysed for assessing data-aware soundness of the original process.

First, given a constraint set C, we define the procedure of computing the new constraint set C' resulting from the addition of a constraint c to C so that C' is univocally determined. This is shown in Algorithm 1, where we maintain the same notation as before, so that x, y, z can be either constants or read variables in V^r. It requires a `saturate` procedure that, given a set C' of constraints as input, returns the constraints in C' in addition to all the constraints implied by C' (using only variables and constants appearing in C'). Since the constraint language is based on comparisons (cf. Definition 2), it trivially follows that only finitely many constraints can be added with this procedure. When given an unsatisfiable constraint set, we assume `saturate` to return the same set as output. We denote such operation as $C' = C \oplus c$. Hereafter, we assume a canonical ordering of variables and, hence, constraints in a set. This allows us to efficiently compute the equality of two sets of constraints.

Referring to the algorithm, the former case (line 1) applies to constraints c that are predicated over the current value of variables, and therefore restrict the set of possible solutions for a constraint set C. The latter case (line 3) applies to constraints that overwrite the current constraints on the variable v, hence c is added (line 4) whereas all previous constraints mentioning v are removed (line 7). After this, at lines 9-11 we replace each constraint of the form $(v^w \odot z)$ with one of the form $(v^r \odot z)$. The test at line 10 applies to constraints c such as $v^w > v^r$ for some variable v, for which line 11 must not be executed (as in the previous example it would result in the constraint $v^r > v^r$). Note that the resulting C' may be not satisfiable.

Next, we consider a set of extra transition symbols τ_t, with $t \in T$. Each τ_t denotes the *silent* transition that corresponds to the explicit case-based reasoning hypothesis of assuming that $guard(t)$ does not hold in the current state: intuitively, by performing this silent action, we assume true the negation of the guard of t. Given a set $E \subseteq T$, we define $\tau_E \doteq \{\tau_t \mid t \in E\}$. We can finally give the formal definition of constraint graph of a given DPN.



$\{i\}, \left\{\begin{matrix}\mathtt{a} = 0 \\ \mathtt{b} = 10\end{matrix}\right\}$
t_1
t_2
$\{p_2\}, \{\mathtt{a} > 10\}$
t_4
t_3
$\{p_1\}, \{\mathtt{a} > 5\}$
τ_{t_3}
$\{p_1\}, \{\mathtt{a} \geq 10\}$
t_2
$\{p_3\}, \left\{\begin{matrix}\mathtt{a} > 10 \\ \mathtt{b} < \mathtt{a}\end{matrix}\right\}$
τ_{t_2}
τ_{t_2}
τ_{t_3}
$\{p_1\}, \{\mathtt{a} = 10\}$
$\{p_2\}, \left\{\begin{matrix}\mathtt{a} < 10 \\ \mathtt{a} > 5\end{matrix}\right\}$
$\{p_1\}, \left\{\begin{matrix}\mathtt{a} \leq 10 \\ \mathtt{a} > 5\end{matrix}\right\}$
t_3

Fig. 2. The constraint graph $CG_{\mathcal{N}}$ for the DPN $\mathcal{N}$ in Figure 1. We list next to each node the couple (M, C), where a marking M is denoted as a set (since the DPN is 1-bounded). The constraint $\mathtt{b} = \mathtt{10}$ is not repeated in each node after the initial one, for brevity. The DPN is clearly unsound, as a can be assigned a value smaller or equal to 10, which would lead to one of the dead-end nodes (which do not correspond to the final marking). We highlight nodes with the final markings by a double circle, and nodes that are dead-ends as forbidden signs.

Definition 6 (Constraint Graph of a DPN). *Let $\mathcal{N} = \langle P, T, F, V, dom, \alpha_I, read, write, guard\rangle$ be a DPN. Let $\mathcal{M}$ be the set of markings of $\mathcal{N}$, and M_I the initial marking. Recalling that $\mathcal{C}_V$ denotes the set of possible constraints on V, the constraint graph $CG_{\mathcal{N}}$ of $\mathcal{N}$ is a tuple $\langle S, s_0, A\rangle$ where:*

- *$S \subseteq \mathcal{M} \times 2^{\mathcal{C}_V}$ is a set of states of the graph, which we call* nodes *to distinguish them from the notion of states of the DPN;*
- *$s_0 = (M_I, C_0) \in S$ is the initial node, where the initial constraints set is computed as $C_0 = \bigcup_{v \in V} \{v = \alpha_I(v)\}$;*
- *$A \subset S \times (T \cup \tau_T) \times S$ is the set of arcs, which is defined with S by mutual induction:*
  - *a transition $((M, C), t, (M', C'))$ is in A iff:*
    - *(i) $M[t\rangle M'$;*[2]
    - *(ii) $C' = C \oplus guard(t)$ is satisfiable.*
  - *a transition $((M, C), \tau_t, (M, C''))$ is in A iff:*
    - *(i) $write(t) = \emptyset$;*
    - *(ii) $\exists M'$ s.t. $M[t\rangle M'$;*
    - *(iii) $C'' = C \oplus \neg guard(t)$ is satisfiable.*

The first two items are very simple: first, the set of nodes is the set of all possible couples in which the first component is a marking on the DPN $\mathcal{N}$ and the second is a constraint set; second, the initial node is identified by the initial marking and the constraint set which simply encodes the initial SV assignment of the case variables.

[2]In the remainder, given a $\mathcal{N} = \langle P, T, F, V, dom, \alpha_I, read, write, guard\rangle$ and two of its markings M and M', we use the notation $M[t\rangle M'$ to denote that a transition t is enabled at marking M and leads to marking M' according to the classical Petri net $\langle P, T, F\rangle$.

The third item defines the transition relation between nodes (i.e., the edges): there are two kinds of transitions. First, given a node (M, C), a new node (M', C') can be reached through a transition $t \in T$ of the DPN iff $A((M, C), t, (M', C'))$, which analogously to DPNs we denote as $(M, C) \xrightarrow{t} (M', C')$. The conditions are as follows: first (i) M' is the marking resulting from firing transition t from M according to the standard underlying Petri net semantics; second (ii) the constraint set C' obtained by adding the guard of t to the current set C, as defined by Algorithm 1, is satisfiable. This case covers the expected semantics of DPNs, as described in the previous section: after firing a transition, the guard of the transition must be true and compatible with the new SV assignment.

Second, given a node (M, C), a new node (M', C') can be also reached through a silent transition τ_t iff $A((M, C), \tau_t, (M', C'))$, denoted $(M, C) \xrightarrow{\tau_t} (M', C')$. For this case, the conditions require that: first (i) the transition t is not writing a variable; second (ii) t can fire given the marking M of the original DPN; third (iii) the constraint set C'' obtained by adding the negation of the guard of t is satisfiable. This case simulates the reasoning by case that is required to take into consideration every possible SV assignment that is produced, in the original DPN, after a transition is fired. Intuitively, an edge labelled with τ_t is intended to model all the SV assignments, consistent with the current constraint set, for which the guard of t is not true.

An example of constraint graph $CG_{\mathcal{N}}$ for the DPN $\mathcal{N}$ in Figure 1 is shown in Figure 2. The DPN is clearly unsound, as `a` can be assigned a value smaller or equal to `10`. This corresponds, in $CG_{\mathcal{N}}$, to the nodes with no outgoing edges.

Definition 6 is constructive, as it formally defines all the conditions that are required in order to build the transitions between the nodes of a constraint graph, starting from the initial node. As such, it can be used to devise a *procedure* for building $CG_{\mathcal{N}}$, which is listed here as Algorithm 2.

The algorithm uses a set L to hold the nodes of the constraint graph being built that still need to be expanded. As new nodes of the form (M, C) are generated, these are added to L and are removed only when all transitions t such that $M[t\rangle M'$ is in $\mathcal{N}$, for some M', have been considered in the for-each loop at line 9 (see also lines 6-8). S and A are, respectively, the nodes and the arcs of the constraint graph returned by the algorithm. A transition labelled with t is built for every $t \in T$ in the DPN $\mathcal{N}$, and at the beginning of the foreach cycle at line 9 the constraint set C' is obtained by adding to C the guard of t (and saturating). Then, consistently with the definition, if the guard of t is a test on the current value of variables, then C'' is computed by updating C with the negated guard, to explicitly represent those SV assignments in which the guard is not satisfied. However, if the constraint graph contains a node (M, C') with the same constraint set and $M' > M$, it means that at least one of the places of the DPN is unbounded, therefore that the DPN is certainly unsound [18], so *`false`* is returned (lines 15-16). If this is not the case, an edge is created for the silent transition τ_t. In particular, the node reached by this edge is such that the

Algorithm 2: Data-aware soundness-checking procedure

```
   Input: A DPN $\mathcal{N} = \langle P, T, F, V, dom, \alpha_I, read, write, guard\rangle$ and an initial marking $M_I$ for $\mathcal{N}$.
   Result: Whether or not $\mathcal{N}$ is data-aware sound
1  $C_0 \leftarrow \bigcup_{v \in V} \{v = \alpha_I(v)\}$,
2  $s_0 \leftarrow \langle M_I, C_0\rangle$,
3  $S \leftarrow \{s_0\}$,
4  $A \leftarrow \emptyset$,
5  $L \leftarrow \{s_0\}$
6  while $L \neq \emptyset$ do
7      $(M, C) \leftarrow \texttt{pick}(L)$
8      $L \leftarrow L \setminus \{(M, C)\}$
9      foreach $t \in T$ s.t. $M \xrightarrow{t} M'$ do
10         $C' \leftarrow C \oplus guard(t)$
11         $C'' \leftarrow C$
12         if $write(t) = \emptyset$ then
13             $C'' \leftarrow C'' \oplus \neg guard(t)$
14         if $\texttt{satisfiable}(C')$ then
15             if $\exists(\bar{M}, \bar{C}) \in S$ s.t. $M' > \bar{M} \wedge C' = \bar{C}$ then
               //The net is unbounded
16                 return false
17             $S \leftarrow S \cup \{(M', C')\}$
18             $A \leftarrow A \cup \{\langle(M, C), t, (M', C')\rangle\}$
19             $L \leftarrow L \cup \{(M', C')\}$
20         if $\texttt{satisfiable}(C'') \wedge C \neq C''$ then
21             $S \leftarrow S \cup \{(M, C'')\}$
22             $A \leftarrow A \cup \{\langle(M, C), \tau_t, (M, C'')\rangle\}$
23             $L \leftarrow L \cup \{(M, C'')\}$
24 return analyzeConstraintGraph $(\langle S, s_0, A\rangle)$
```

marking is not updated (as no transition in the DPN was fired), whereas the new constraint set is the C'' computed as above. The algorithms then continues considering a new transition.

Note that, apart from computing the constraint graph $CG_{\mathcal{N}}$ of a given DPN $\mathcal{N}$, the algorithm also checks *on the fly* whether $\mathcal{N}$ is data-aware sound. The following theorem states the soundness of the approach.

Theorem 1. *Algorithm 2 terminates and returns* `true` *iff $\mathcal{N}$ is data-aware sound.*

Arguments for supporting the soundness of the procedure are deferred to the next section. Here we only note that the algorithm always terminates, when the net is bounded, because the sets of possible markings $\mathcal{M}$, the set of transitions T (and thus the set τ_T) are finite. If instead the net is unbounded then $\mathcal{M}$ is infinite, but this will be eventually detected by the algorithm, which will then terminate and simply return *`false`*. This implies that the number of possible nodes is also bounded, and thus $CG_{\mathcal{N}}$ is finite-state for any $\mathcal{N}$: the set of possible markings and possible constraint sets are finite.

As a consequence, if and only if the procedure `analyzeConstraintGraph` returns *`true`* for a given DPN $\mathcal{N}$ then we conclude that $\mathcal{N}$ is data-aware sound. We here do not present an algorithm corresponding to such procedure,

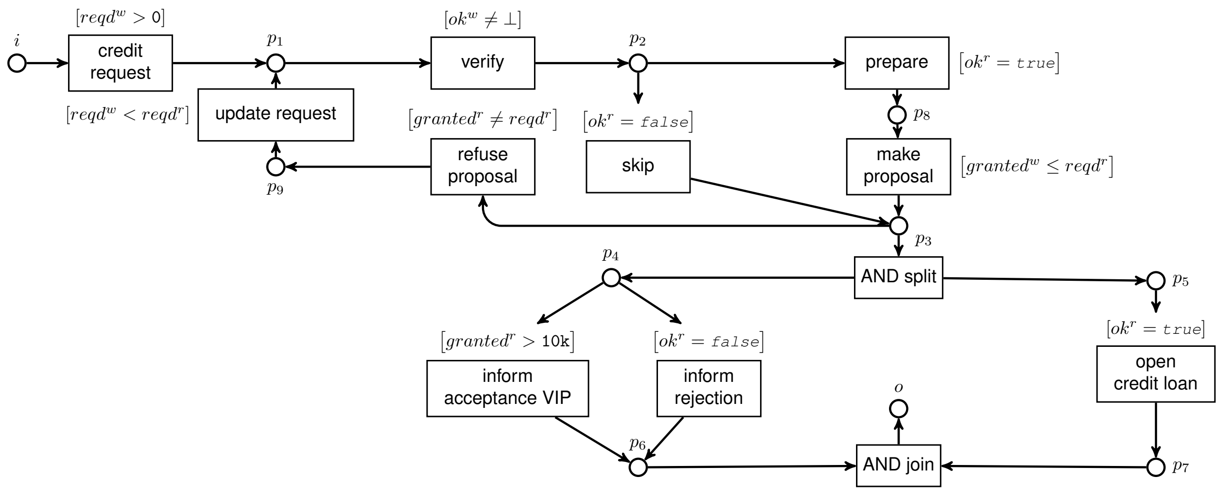


Fig. 3. An example DPN inspired to the example in [8], modeling the process of requesting and approving a loan. $M_I = \{i\}$ and $M_F = \{o\}$. Guards of transitions are shown between squared brackets. We assume that, at the beginning of the process, no variable is set. The and-split and and-join transitions are considered to have some tautologically true guard.

as it only needs to apply the definition of data-aware soundness on the constraint graph $CG_\mathcal{N}$ for $\mathcal{N}$, which is finite-state. This can be implemented by exploiting either verification or search techniques, similarly to what was done in [8] for the case of DPNs with restricted guards.

A. Example

We now introduce a more complex process than the one modeled in Figure 1, which we use as running example with the purpose of illustrating the technical details which follow (for this reason, the process does not sometimes match a reasonable modeling of a real world scenario). The example, depicted in Figure 3, models a process in which a customer applies for a loan, and it is a modification of the one in [8].

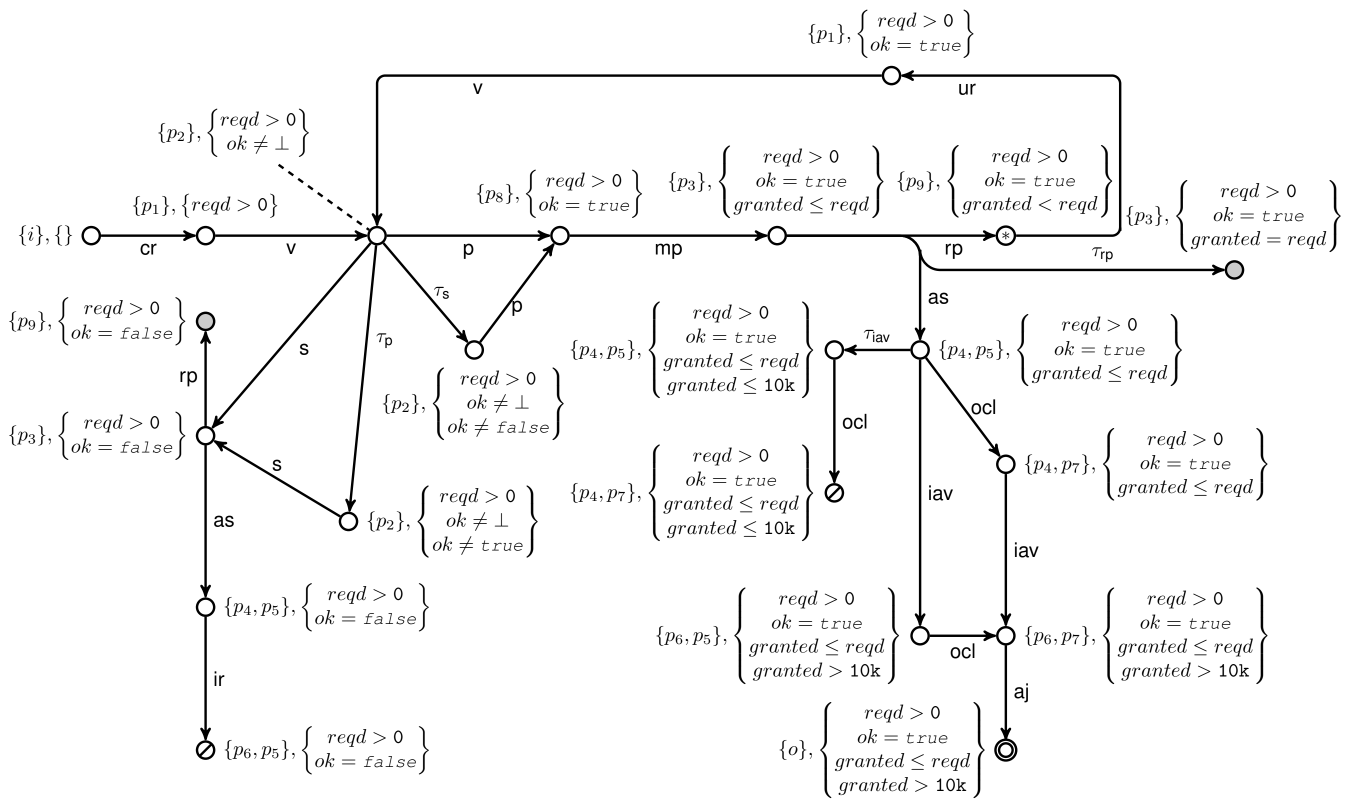
The process is as follows. The initial marking is $M_I = \{i\}$, that is only one token is in the input place. A credit request transition models the activity of requesting a certain positive amount, hence writing the variable $reqd$ (we assume for such a variable the domain of integers, i.e. $\mathcal{D}_\mathbb{Z}$). After this, a verify transition models the activity of performing some background checks, which results in determining the value of a boolean variable ok. Depending on the outcome of the check, two transitions can be fired: either prepare or skip. Their only purpose is to make sure that make proposal can be only executed when the check was successful, and they could be equivalently modeled as silent transitions (which however we do not allow in the DPN, to simplify the technical details). The transition make proposal determines the amount that the bank is actually willing to lend, and therefore a further variable $granted$ is written, assigning a value that is smaller or equal to the requested amount. Also $granted$ has domain $\mathcal{D}_\mathbb{Z}$, as it is required by Definition 2: constraints (thus guards) comparing variables are only possible when these variables share the same domain. If the proposed amount is smaller than the requested amount, the transition refuse proposal can be fired, to restart the procedure with a smaller request, by executing a further transition update request. Alternatively, two parallel branches can be initiated: if the check represented by verify was successful, the transition open credit loan can be fired; at the same time, depending on the value of ok and $granted$, one of two transitions can be fired to notify the customer. These are inform acceptance VIP and inform rejection.

The DPN is clearly problematic, and it is indeed not data-aware sound: if verify assigns to ok the value $\texttt{false}$ then the and-join will never be executable, and thus the final marking $M_F = \{o\}$ never reached. A similar case happens when $granted$ is assigned a value smaller or equal to $\texttt{10k}$.

We now comment the construction of the constraint graph $CG_\mathcal{N}$ for $\mathcal{N}$, which is partially depicted in Figure 4, showing how the same conclusion can be reached by analysing its possible runs. The initial node is $(\{i\}, \{reqd = \bot, ok = \bot, granted = \bot\})$, that is, all case variables are not set. As depicted, runs exist which reach nodes that have no outgoing transitions and thus cannot be extended to reach a node with the final marking. For instance, the run in which the transition make proposal writes a value for $granted$ that is smaller or equal to $\texttt{10k}$ while ok is $\texttt{true}$: in this case, the place p_6 can never be marked. Analogously, when ok is assigned value $\texttt{false}$, then the current run cannot be extended so that place p_7 is marked, and therefore an output marking cannot be reached. Therefore, one should be able to conclude that $\mathcal{N}$ is not data-aware sound. In the next section we formalise this intuition, and further comment on the structure of $CG_\mathcal{N}$ for this example. For now, note how the firing of transition refuse proposal induces a constraint set in which $granted \leq reqd$ is replaced by $granted < reqd$ in the resulting node, and then how the satisfiability requirements at lines 14 and 20 of Algorithm 2 prevents several nodes and edges from appearing in the constraint graph.

B. Data-aware soundness checking on the constraint graph

In this section we prove that Algorithm 2 is correct, in that it correctly establishes the soundness of any given DPN. First, observe that the algorithm simply constructs



$\{p_1\}, \{reqd > 0, ok = true\}$
v
ur
$\{p_2\}, \{reqd > 0, ok \neq \bot\}$
$\{p_8\}, \{reqd > 0, ok = true\}$
$\{p_3\}, \{reqd > 0, ok = true, granted \leq reqd\}$
$\{p_9\}, \{reqd > 0, ok = true, granted < reqd\}$
$\{p_3\}, \{reqd > 0, ok = true, granted = reqd\}$
$\{p_1\}, \{reqd > 0\}$
$\{i\}, \{\}$
cr
v
p
mp
rp
τ_{rp}
τ_s
p
as
$\{p_9\}, \{reqd > 0, ok = false\}$
s
τ_p
$\{p_4, p_5\}, \{reqd > 0, ok = true, granted \leq reqd, granted \leq 10k\}$
τ_{iav}
$\{p_4, p_5\}, \{reqd > 0, ok = true, granted \leq reqd\}$
rp
$\{p_2\}, \{reqd > 0, ok \neq \bot, ok \neq false\}$
$\{p_3\}, \{reqd > 0, ok = false\}$
s
ocl
ocl
$\{p_4, p_7\}, \{reqd > 0, ok = true, granted \leq reqd, granted \leq 10k\}$
iav
$\{p_4, p_7\}, \{reqd > 0, ok = true, granted \leq reqd\}$
as
$\{p_2\}, \{reqd > 0, ok \neq \bot, ok \neq true\}$
iav
$\{p_4, p_5\}, \{reqd > 0, ok = false\}$
$\{p_6, p_5\}, \{reqd > 0, ok = true, granted \leq reqd, granted > 10k\}$
ocl
$\{p_6, p_7\}, \{reqd > 0, ok = true, granted \leq reqd, granted > 10k\}$
ir
aj
$\{p_6, p_5\}, \{reqd > 0, ok = false\}$
$\{o\}, \{reqd > 0, ok = true, granted \leq reqd, granted > 10k\}$

Fig. 4. A fragment of the constraint graph $CG_{\mathcal{N}}$ for the DPN $\mathcal{N}$ in Figure 3, built by following Algorithm 2 (grey nodes are not expanded further for lack of space). Arcs are labelled with the initials of the transition names of the DPN, and constraints that assign variables to $\bot$ (which is the case for all variables in the initial node) are not shown. Not every node is expanded through τ edges, because the resulting nodes would have inconsistent constraint sets.

the constraint graph $CG_{\mathcal{N}}$ when $\mathcal{N}$ is given as input and then, after checking that $\mathcal{N}$ is not unbounded (as this implies unsoundness), it launches an auxiliary procedure `analyzeConstraintGraph` on $CG_{\mathcal{N}}$, which analyses the constraint graph for undesired deadlocks.

First, for comparing the executions of the constraint graph to those of the original DPN $\mathcal{N}$, we need to define a structure which incorporates the trace set of $\mathcal{N}$. We call this the reachability graph of a DPN.

Definition 7 (Reachability graph of a DPN)**.** *Given a DPN $\mathcal{N} = \langle P, T, F, V, dom, \alpha_I, read, write, guard\rangle$, the reachability graph of $\mathcal{N}$ is a graph $\langle V, E\rangle$ where*

- *$V = Reach_{\mathcal{N}}$ is the set of reachable states of $\mathcal{N}$; and*
- *$E \subseteq V \times T \times V$ is the set of arcs such that there exists an arc $v \xrightarrow{t,\beta} v'$ in $RG_{\mathcal{N}}$ iff $v \xrightarrow{t,\beta} v'$ in $\mathcal{N}$.*

Fixing a given DPN $\mathcal{N}$, we denote the set of transitions enabled in a marking M (i.e., by only looking at the underlying Petri net semantics without considering guards and SV assignments) as *enabled*$(M) \doteq \{t \mid M[t\rangle\}$. Likewise, with some abuse of notation, given a state (M, α) of its $RG_{\mathcal{N}}$, we denote the set of transitions enabled in such state as *enabled*$((M, \alpha)) \doteq \{t \in T \mid (M, \alpha) \xrightarrow{t,\beta} (M', \alpha')$ for some $M', \alpha', \beta\}$. The same notation applies to constraint graphs, by defining *enabled*$((M, C)) \doteq \{t \in T \mid (M, C) \xrightarrow{t} (M', C')\}$. We now define a custom simulation notion between reachability graphs and constraint graphs. This will allow us to express a formal relationship between their possible executions. First, we need to introduce some formal definitions.

Considering a special empty symbol ϵ for transitions, we define the *observation function* $obs : T \cup \tau_T \to T \cup \{\epsilon\}$ so that $obs(t) \doteq t$ if $t \in T$ and $obs(t) \doteq \epsilon$ otherwise (i.e., $t \in \tau_T$). Given a trace $\sigma = t^1 \cdots t^n$ of a constraint graph as above, the corresponding *observed trace* is the sequence $obs(\sigma) \doteq obs(t^1) \cdots obs(t^n)$. For instance, for $\sigma = t \cdot \tau_{t'} \cdot t''$ ($\cdot$ is used to denote concatenation) we have $obs(\sigma) = t \cdot t''$. A trace σ is said to be *one-step* iff $obs(\sigma)$ is not equal to ϵ and it has length one: it is composed by an arbitrary number of silent transitions and exactly one transition in T.

Definition 8 (*obs*-Simulation Relation)**.** *Given a reachability graph $RG_{\mathcal{N}} = \langle V, E\rangle$ of a DPN $\mathcal{N}$ and its constraint graph $CG_{\mathcal{N}} = \langle S, s_0, A\rangle$, we say that a relation $R \subseteq S \times V$*

- *is a obs-simulation relation of $RG_{\mathcal{N}}$ by $CG_{\mathcal{N}}$ iff $(s, (M, \alpha)) \in R$ implies that for any $(M, \alpha) \xrightarrow{t,\beta} (M', \alpha')$ there exists a one-step trace fragment σ with $obs(\sigma) = t$ and $s \xrightarrow{\sigma} s'$, so that $(s', (M', \alpha')) \in R$.*
- *is a obs-simulation relation of $CG_{\mathcal{N}}$ by $RG_{\mathcal{N}}$ iff $(s, (M, \alpha)) \in R$ implies that for any one-step trace fragment σ with $s \xrightarrow{\sigma} s'$ in $CG_{\mathcal{N}}$ there exists $(M, \alpha) \xrightarrow{t,\beta} (M', \alpha')$ in $RG_{\mathcal{N}}$, with $obs(\sigma) = t$, such*

that $(s', (M', \alpha')) \in R$.

A node of $CG_{\mathcal{N}}$ obs-simulates a state in $RG_{\mathcal{N}}$ iff there exists a obs-simulation relation R of $RG_{\mathcal{N}}$ by $CG_{\mathcal{N}}$ such that these are included in the relation, and we say that $CG_{\mathcal{N}}$ obs-simulates $RG_{\mathcal{N}}$ if s_0 obs-simulates (M_I, α_I). Similarly for the other direction.

Lemma 1. *$CG_{\mathcal{N}}$ obs-simulates $RG_{\mathcal{N}}$.*

Proof: We show this by induction. First, note that the marking reached by executing a sequence of observable transitions is the same, in both the reachability graph (hence the DPN) and the constraint graph (irrespective of the unobservable transitions added). This is because the variable assignment β^i considered at each step plays no role in computing the next marking of the DPN. Hence in what follows we use the same symbol M for the new marking of both $RG_{\mathcal{N}}$ and $CG_{\mathcal{N}}$. To prove the claim, we first need to make sure that the same set of transitions $t \in T$ are enabled in both (M_I, α_I) and (M_I, C_0), i.e., the initial state of $RG_{\mathcal{N}}$ and initial node of $CG_{\mathcal{N}}$. This is true by construction because $C_0 \doteq \bigcup_{v \in V}\{(v = \alpha_I(v))\}$ and thus *enabled*$((M_I, \alpha_I))$ = *enabled*$((M_I, C_0))$. Then, if $(M_I, \alpha_I) \xrightarrow{t,\beta} (M, \alpha)$ there must exist a trace σ with $obs(\sigma) = t$ such that $(M_I, C_0) \xrightarrow{\sigma} (M, C)$ is in $CG_{\mathcal{N}}$, and so that (M, C) obs-simulates (M, α). In particular, this is true for $\sigma = t \cdot \tau_E$ with $E = \{t' \in T \mid t' \neq t \wedge t' \in$ *enabled*$(M) \wedge t' \notin$ *enabled*$((M, \alpha)) \wedge write(t') = \emptyset\}$. The idea is that $\sigma = t \cdot \tau_E$ corresponds to the execution of t in the constraint graph, followed by an unobservable transition for each transition $t' \in$ *enabled*(M) which cannot be fired from the new state (M, α) of the reachability graph due to the SV assignment α. Such σ makes sure that the new generated constraint set C correctly encodes the initial set C_0 updated with the negated guard of each transition that cannot be fired next in $RG_{\mathcal{N}}$. Notice that, by construction, a run fragment with trace σ always exists in $CG_{\mathcal{N}}$: unobservable transitions τ_t are always added for any t with $write(t) = \emptyset$ as long as the resulting constraint set is satisfiable (cf. Definition 6). Since $obs(\sigma) = t$, it remains to show that (M, C) obs-simulates (M, α). This can be done by repeating in the inductive step the same reasoning as above, since the marking is the same (hence the set of transitions in *enabled*(M)) and also *enabled*$((M, \alpha)) \subseteq$ *enabled*$((M, C))$. ■

The above result implies that the constraint graph can 'mimic' any possible execution of the reachability graph, hence of the DPN. This gives us a very formal and precise characterisation of the ability of the constraint graph, which is finite-state, to account for the possibly infinite executions of the DPN. This, however, does not imply that *any* property true in $RG_{\mathcal{N}}$ is also true in $CG_{\mathcal{N}}$, or vice-versa.

Interestingly, the converse does not hold: $RG_{\mathcal{N}}$ does not obs-simulate $CG_{\mathcal{N}}$ in general. As an evidence of this, let us consider again the DPN $\mathcal{N}$ and the constraint graph $CG_{\mathcal{N}}$ in Figure 4. From the node marked with an asterisk we can execute the sequence of transitions update request, verify, prepare, make proposal, refuse proposal, which corresponds to a cycle in the constraint graph (it reaches again the same node). The constraint graph is indeed allowed to be cyclic. Note, however, that a corresponding cycle in the reachability graph $RG_{\mathcal{N}}$ does not exist: variable $reqd$ has non-dense domain of integers, which implies that the sequence of transitions composing the cycle in $CG_{\mathcal{N}}$ can in fact be executed only a bounded number of times (since $reqd$ is decreased at each iteration). This can be taken as one of the intuitions about the differences between $RG_{\mathcal{N}}$ and its abstraction $CG_{\mathcal{N}}$, for which it is not true that $RG_{\mathcal{N}}$ always obs-simulates $CG_{\mathcal{N}}$ for any $\mathcal{N}$. As commented in the future work section, however, we plan to formulate conditions to be checked directly on the original DPN $\mathcal{N}$ which allow to characterise this sort of mismatch in terms of the properties that can be checked on $CG_{\mathcal{N}}$.

We now adapt the definition of data-aware soundness to constraint graphs, as our objective is to assess these properties on these objects. The three properties $P1 - P3$ as in Definition 5 become as follows. By denoting as $Reach_{CG_{\mathcal{N}}}$ the set of reachable nodes of $CG_{\mathcal{N}}$, namely the set $\{(M, C) \mid (M_I, C_0) \xrightarrow{*} (M, C)\}$ we then have:

$P1$: $\forall (M, C) \in Reach_{CG_{\mathcal{N}}}.\ \exists C'.(M, C) \xrightarrow{*} (M_F, C')$;
$P2$: $\forall (M, C) \in Reach_{CG_{\mathcal{N}}}.\ M \geq M_F \Rightarrow (M = M_F)$;
$P3$: $\forall t \in T.\ \exists M_1, M_2, C_1, C_2.\ (M_1, C_1) \in Reach_{CG_{\mathcal{N}}}$ and $(M_1, C_1) \xrightarrow{t} (M_2, C_2)$.

A constraint graph $CG_{\mathcal{N}}$ of a DPN $\mathcal{N}$ is said to be data-aware sound iff all these properties are true. With this definition at hand, we can now state our main result. Intuitively, it states that the constraint graph constitutes a faithful abstraction of the original DPN with respect to data-aware soundness.

Theorem 2. *$RG_{\mathcal{N}}$ is data-aware sound iff $CG_{\mathcal{N}}$ is data-aware sound.*

Proof: Consider the first property $P1$ of data-aware soundness, which requires the co-reachability of an 'output node' (i.e. reachability from any reachable node).

($\Rightarrow$). This direction follows by simulation. Assume to fix a state (M, α) reached by executing a trace σ', and for which the property must hold if $RG_{\mathcal{N}}$ is data-aware sound: there exists a trace σ such that $(M, \alpha) \xrightarrow{\sigma} (M_F, \alpha')$ for some α'. By obs-similarity, at least one node (M, C) exists that is reached in $RG_{\mathcal{N}}$ through a run with trace σ', for which (M, C) obs-simulates (M, α) – in the base case, these are the initial state and node, respectively. Then by Lemma 1 there must also exist a run $(M, C) \xrightarrow{\sigma''} (M_F, C')$, for some C', with $\sigma'' = obs(\sigma)$ and which only traverses nodes that (stepwise, by considering one-step traces) obs-simulate those in σ. If instead the property is not true for $RG_{\mathcal{N}}$, if $CG_{\mathcal{N}}$ is data-aware sound it means that either $RG_{\mathcal{N}}$ has additional runs which do not correspond to runs of $CG_{\mathcal{N}}$ (and which cannot be extended to reach a final node) or that there exists in $CG_{\mathcal{N}}$ at least one run, with trace σ, such that $obs(\sigma)$ is not a legal trace of $RG_{\mathcal{N}}$. Both cases are excluded by construction, as it can be deduced by inspecting Definition 6 or Algorithm 2.

($\Leftarrow$) First, consider a run fragment $s \xrightarrow{\sigma'} s'$ in $CG_{\mathcal{N}}$ where σ' is not required to be a one-step trace. Observe that if one such run exists which is cyclic, then one which ends in s' must exist that is acyclic as cycles do not affect the reachability of nodes (a run is cyclic iff the same node appears twice). Therefore in what follows we are allowed to restrict to these acyclic runs of $CG_{\mathcal{N}}$. To satisfy the requirement $P1$ it is indeed enough to find an acyclic run in $CG_{\mathcal{N}}$ which reaches a node with final marking that also includes a given node s, for every reachable s. Hence we show that by construction every acyclic run ρ on $CG_{\mathcal{N}}$ with trace σ has a corresponding run ρ' in $RG_{\mathcal{N}}$ with trace σ', and in particular one such that $\sigma' = obs(\sigma)$, so that for every $\rho \cdot \rho''$ there exists $\rho' \cdot \rho'''$ in $RG_{\mathcal{N}}$ with the same observed trace. At the beginning we have (M_I, C_0) and (M_I, α_I), with $C_0 \doteq \bigcup_{v \in V} \{v = \alpha_I(v)\}$, and $\mathit{enabled}((M_I, \alpha_I)) = \mathit{enabled}((M_I, C_0))$. Then, consider $(M, C) \xrightarrow{\sigma} (M', C')$ to be one-step, with (M, C) *obs*-simulating (M, α). Then a transition $(M, \alpha) \xrightarrow{t,\beta} (M', \alpha')$ with $obs(\sigma) = t$ so that $(M'C')$ *obs*-simulates (M', α') must exist as well by construction, as a transition exists in $CG_{\mathcal{N}}$ iff it is enabled in the current marking in $\mathcal{N}$, $\mathit{guard}(t)$ is satisfied in C, $M[t\rangle M'$ and C' is consistent. Since C' is consistent, a solution α' exists. If the property is not true for $CG_{\mathcal{N}}$, then there exists a run ρ reaching a node (M, C) from where a node with final marking cannot be reached, i.e., there is no run ρ' from (M, C) reaching such nodes. This implies, by construction, that if σ is the trace of the run $\rho \cdot \rho'$, $obs(\sigma)$ is not legal in $\mathcal{N}$.

For $P2$ and $P3$ we follow the same reasoning as for $P1$: they all require the existence of runs from states in $RG_{\mathcal{N}}$ (and nodes in $CG_{\mathcal{N}}$), that are reached through corresponding traces σ' and σ, respectively, so that $\sigma' = obs(\sigma)$. ■

By exploiting this theorem, we establish the decidability of the problem of checking data-aware soundness of processes represented as DPNs, and provide a practical and implementable procedure for doing so.

V. Conclusions and Future work

In this paper we generalized the study of soundness of DPNs to the case in which the evolution of the process also depends on the comparison between the values carried by the variables of the process. We have shown the decidability of the problem of assessing the soundness for this class of DPNs, by reducing it to the analysis of a finite-state abstraction of the possible executions of the original DPN. Our result extends to any constraint language that has two properties: (i) it generates only boundedly many constraints over a fixed set of variables and constants, and (ii) has decidable satisfiability. We have also defined and investigated the formal relationship between the reachability graph of DPNs and their finite-state abstraction.

In future work, we plan to investigate further such relationship, and identify the subclasses of processes for which we can identify more general properties, beyond data-aware soundness, which can be used to express complex interplays of temporal and data-aware requirements of processes, and which can be assessed with the same abstraction technique illustrated here. Further, we plan to implement the approach described in this paper by combining standard state-space construction methods with constraint programming techniques, needed to perform the satisfiability checks required by the abstraction.

Acknowledgements

This work was supported in part by the Unibz CRC project DACoMan and the Unibz RTD project SYNCED.

References

[1] Decision model and notation (DMN) v1.1, 2016.

[2] K. Batoulis, S. Haarmann, and M. Weske. Various notions of soundness for decision-aware business processes. In *Proc. of ER 2017*, volume 10650 of *LNCS*, pages 403–418. Springer, 2017.

[3] K. Batoulis and M. Weske. Soundness of decision-aware business processes. In *Proc. of BPM Forum 2017*, pages 106–124. Springer, 2017.

[4] D. Calvanese, G. De Giacomo, and M. Montali. Foundations of data aware process analysis: A database theory perspective. In *Proc. of PODS 2013*. ACM, 2013.

[5] D. Calvanese, M. Dumas, Ü. Laurson, F. M. Maggi, M. Montali, and I. Teinemaa. Semantics and analysis of DMN decision tables. volume 9850 of *LNCS*, pages 217–233. Springer, 2016.

[6] F. Corradini, F. Fornari, A. Polini, B. Re, and F. Tiezzi. A formal approach to modeling and verification of business process collaborations. *Science of Computer Programming*, 166:35 – 70, 2018.

[7] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, and A. Vandin. Bprove: A formal verification framework for business process models. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 217–228, Piscataway, NJ, USA, 2017. IEEE Press.

[8] M. de Leoni, P. Felli, and M. Montali. A holistic approach for soundness verification of decision-aware process models. In *Conceptual Modeling - 37th International Conference, ER 2018, Xi'an, China, October 22-25, 2018, Proceedings*, pages 219–235, 2018.

[9] D. Knuplesch, L. T. Ly, S. Rinderle-Ma, H. Pfeifer, and P. Dadam. On enabling data-aware compliance checking of business process models. volume 6412 of *LNCS*, pages 332–346. Springer, 2010.

[10] F. Mannhardt. *Multi-perspective process mining*. Ph.d. thesis, Eindhoven University of Technology, 2018. http://repository.tue.nl/b40869c0-2d11-4016-a92f-8e4ee9cd9d66.

[11] M. Montali and A. Rivkin. Db-nets: On the marriage of colored petri nets and relational databases. *T. Petri Nets and Other Models of Concurrency*, 12:91–118, 2017.

[12] S. Morimoto. A survey of formal verification for business process modeling. In *Proceedings of the 8th International Conference on Computational Science, Part II*, ICCS '08, pages 514–522, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] M. Reichert. Process and data: Two sides of the same coin? In *Proc. of OTM 2012*, volume 7565 of *LNCS*, pages 2–19. Springer, 2012.

[14] S. Sadiq, M. Orlowska, W. Sadiq, and C. Foulger. Data flow and validation in workflow modelling. In *Proceedings of the 15th Australasian Database Conference - Volume 27*, Proc. of ADC '04, pages 207–214, Darlinghurst, Australia, 2004. Australian Computer Society, Inc.

[15] N. Sidorova, C. Stahl, and N. Trčka. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Information Systems*, 36(7):1026–1043, 2011.

[16] M. Triebel and J. Sürmeli. Homogeneous equations of algebraic petri nets. In *Proc. of CONCUR 2016*, LNCS, pages 1–14. Springer, 2016.

[17] W. M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[18] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.

[19] V. Vianu. Automatic verification of database-driven systems: a new frontier. In *Proc. of ICDT 2009*, pages 1–13, 2009.