Business Process Regulatory Compliance is Hard

Author

Tosatto, Silvano Colombo, Governatori, Guido, Kelsen, Pierre

Published

2015

Journal Title

IEEE Transactions on Services Computing

Version

Accepted Manuscript (AM)

DOI

10.1109/TSC.2014.2341236

Business Process Regulatory Compliance is Hard

Silvano Colombo Tosatto, Guido Governatori and Pierre Kelsen

Abstract—Verifying whether a business process is compliant with a regulatory framework is a difficult task. In the present paper we prove the hardness of the business process regulatory compliance problem by taking into account a sub-problem of the general problem. This limited problem allows to verify only the compliance of structured processes with respect to a regulatory framework composed of a set of conditional obligations including a deadline. Experimental evidence from existing studies shows that compliance is a difficult task. In this paper, despite considering a sub-problem of the general problem, we provide some theoretical evidence of the difficulty of the task. In particular we show that the source of the complexity lies in the core language of verifying conditional obligations with a deadline. We prove that for this simplified case verifying partial compliance belongs to the class of **NP**-complete problems, and verifying full compliance belongs to the class of co**NP**-complete problems. Thus by proving the difficulty of a simplified compliance problem we prove that the general problem of verifying business process regulatory compliance is hard.

Index Terms—Compliance, Computational Complexity, **NP**-Completeness, Hamiltonian Path, Tautology

1 Introduction

Compliance initiatives are becoming more and more important in enterprises with the increase of the number of regulatory frameworks explicitly requiring businesses to show compliance with them. As a consequence IT support for compliance activities within enterprises is growing.

One of the possibilities to show compliance with a regulatory framework is through business process models. Using these models an enterprise can represent its ways to achieve a business objective. The business process regulatory compliance problem studies whether a business process is compliant with the obligations specified by the regulatory framework.

A business process can be compliant with the obligations specified by the regulatory framework in two ways. If all the ways of achieving the business objective contained in the business process fulfil the obligations, then the business process is fully compliant. Differently if at least one of the ways to achieve the business objective fulfils the obligations, then the business process is considered to be partially compliant with the regulatory framework. However, the business process is not compliant with the regulatory framework in the case where none of the ways of achieving a business objective contained in a business process fulfills the obligations.

- *S. Colombo Tosatto and P. Kelsen are with University of Luxembourg, Luxembourg.*
  E-mail: silvano.colombotosatto@uni.lu
  pierre.kelsen@uni.lu
- *G. Governatori is with NICTA, Brisbane, Australia, and with Queensland University of Technology, Brisbane, Australia.*
  E-mail: guido.governatori@nicta.com.au

Solutions for the business process regulatory compliance problem have been already proposed in the literature (e.g., Governatori et al. [1], [2], Goedertier and Vanthienen [3], Hoffmann et al. [4], Awad et al. [5], Ly et at. [6] and Ramezani et al. [7]). However the solutions proposed do not consider the complexity of the problem or, in case they do, they only provide approximate solutions.

The paper is structured as follows: Section 2 describes the compliance problem by defining its elements: the process models and the obligations. Section 3 contains the complexity proof showing that the problem of verifying partial compliance is **NP**-complete. Section 4 proves that the problem of verifying full compliance is co**NP**-complete. Section 5 gives the proof showing that the problem of verifying non compliance is co**NP**-complete. Section 6 concludes the paper.

2 The Business Process Regulatory Compliance Problem

This section describes the business process regulatory compliance problem. To describe the problem we first introduce the business process models and secondly the abstract framework defining the regulatory framework and compliance.

We adopt an abstract approach because of the goal of studying the complexity of the problem itself. By opting for an abstract approach we are capable of focusing on the most general and common features of the problem and leaving out the minor ones, used only by some specific instances of the problem.

2.1 Business Process

In the present paper we study the complexity of verifying the compliance of a particular class of business processes: the structured processes. Such class of business processes, similar to the structured workflows defined by Kiepuszewski et al. [8], is limited in its expressivity because it does not allow cycles and its components have to be properly nested. An advantage of using structured processes is that their correctness can be verified in polynomial time. While not all business processes are structured, the structured processes are a substantial class of real-life processes. According to Polyvanyy et al. [9], 406 of the 604 processes in the SAP reference models [10] are structured. In addition Polyvanyy et al. [9] identify conditions under which unstructured processes can be transformed into structured ones, and proposes an algorithm for the transformation. They also report that seventy-eight of the unstructured processes in the SAP reference models can be converted into behaviourally equivalent structured process models.

We define the structured processes used in this paper compositionally and following the semantics used by *Business Process Model and Notation 2.0*[1]. We first define the basic blocks constituting a structured process. The most basic element is the task; it abstractly represents an atomic action that can be executed to help towards the achievement of the business objective purposed by the business process. The tasks can be then combined in more complex structures, like sequences, *and* blocks and *xor* blocks. In turn such structures can be used to build more complex structures.

Definition 1 (Process Block): A process block B is a directed graph: the nodes are called *elements* and the edges are called *transitions*. We also use the terms nodes and vertices for elements and edges for transitions. The set of elements of a process block is identified by the function $V(B)$ and the set of transitions by the function $E(B)$. The set of elements is composed of tasks and coordinators. The coordinators are of 4 types: and_split, and_join, xor_split and xor_join. Each process block B has two distinguished nodes called the initial and final element. The initial element has no incoming transition from other elements in B and is denoted by $b(B)$. Similarly the final element has no outgoing transitions to other elements in B and is denoted by $f(B)$.

A directed graph composing a process block is defined inductively as follows:

- A single task constitutes a process block. The task is both initial and final element of the block.
- Let $B_1, \ldots, B_n$ be process blocks with $n > 1$:
  - SEQ$(B_1, \ldots, B_n)$ denotes the process block with node set $\cup V(B_i)$ and edge set $\cup E(B_i) \cup \{(f(B_i), b(B_{i+1})) : 1 \leq i < n\}$.
  - XOR$(B_1, \ldots, B_n)$ denotes the block with vertex set $\cup V(B_i) \cup \{xsplit, xjoin\}$ and edge set $\cup E(B_i) \cup \{(xsplit, b(B_i)), (f(B_i), xjoin) : 1 \leq i \leq n\}$ where $xsplit$ and $xjoin$ denote an xor_split coordinator and an xor_join coordinator, respectively.
  - AND$(B_1, \ldots, B_n)$ denotes the block with vertex set $\cup V(B_i) \cup \{asplit, ajoin\}$ and edge set $\cup E(B_i) \cup \{(asplit, b(B_i)), (f(B_i), ajoin) : 1 \leq i \leq n\}$ where $asplit$ and $ajoin$ denote an and_split and an and_join coordinator, respectively.

Using the process blocks introduced in Definition 1, it is then possible to define the structured business processes. These type of processes are defined by enclosing a process block within two specific pseudo-tasks: the *start* and *end*, which are respectively placed before and after a process block to construct a structured business process.

Definition 2 (Structured Process Model): The pseudo-tasks start and end are used respectively to identify the beginning of a structured process model and when it terminates. A structured process model P is a directed graph composed of a process block B called the main process block. The vertex set of P is $V(P) = V(B) \cup \{\mathsf{start}, \mathsf{end}\}$ and its edge set is $E(P) = E(B) \cup \{(\mathsf{start}, b(B)), (f(B), \mathsf{end})\}$.

The processes used in this paper can be graphically represented using *Business Process Model and Notation 2.0*. We use ◯ to represent the start coordinator and ◉ to represent the end coordinator. The and_split and and_join coordinators are represented both by ✚. The and_split is identified by a single incoming transition and multiple outgoing transitions. The opposite is true for the and_join, which is identified by multiple incoming transitions and a single outgoing transition. In the same way, the operator ✖ identifies both xor_split and xor_join coordinators.

In a structured process model XOR blocks and AND blocks have to be properly nested, meaning that if the block A starts inside the block B, A has to end within B. An example of a structured process model is shown in Fig. 1.

Example 1: Fig. 1 shows a structured business process containing four tasks labelled $t_1, \ldots, t_4$. The structured process contains an XOR block delimited by the xor_split and the xor_join. The XOR block contains the tasks t_1 and t_2. The XOR block is itself nested inside an AND block with the task t_3. The AND block is nested in a SEQ block where it is followed by task t_4.

Structured processes exclude business processes containing badly nested blocks (Fig. 2.(a)) and business processes with loops (Fig. 2.(b)).

An execution of a structured process is a sequence of a subset of the tasks belonging to the model and it represents a possible way to achieve the business objective, which has been modelled in the process. A valid execution identifies a path from the start to the

1. http://www.omg.org/spec/BPMN/2.0

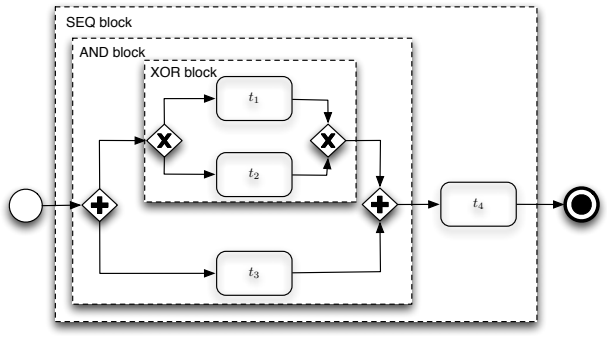

Fig. 1. A structured business process

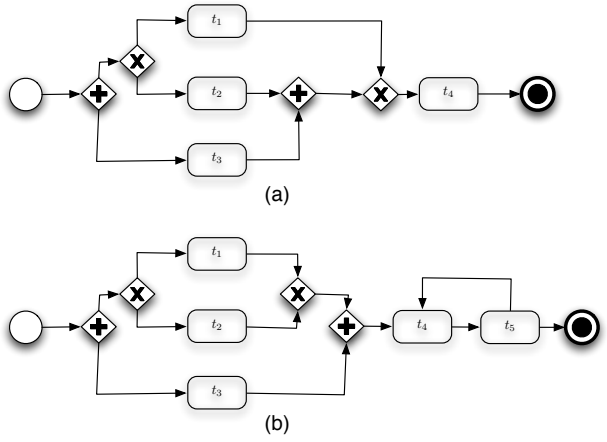

Fig. 2. Examples of non-structured processes

end of the process and follows the semantics of the coordinators and transitions that are traversed.

Before proceeding to define an execution of a business process, we recall the definition of partial ordered set which is used as an auxiliary concept in defining the serialisation of a process block. A process block serialisation is then used to define a business process execution. In addition to recalling the definition of partial ordered set, we introduce some operations on this type of set.

Definition 3 (Partial Ordered Set): A partial order set $\mathbb{P} = (\mathcal{S}, \prec_s)$ is a tuple where $\mathcal{S}$ is a set of elements and $\prec_s$ is a set of ordering relations between two elements of $\mathcal{S}$ such that $\prec_s \subseteq \mathcal{S} \times \mathcal{S}$ and for which *transitivity* and *antisymmetry*[2] hold.

Two special cases of a partial ordered set are the *set* and the *sequence*:

- *Set*: a set is a partial ordered set where no ordering relations have been defined between its elements, formally: $(\mathcal{S}, \emptyset)$.
- *Sequence*: a sequence is a particular partial ordered set, called total order, where an ordering relation is defined between each pair of elements belonging to the set, formally $(\mathcal{S}, \prec_s)$ where $\forall x, y \in \mathcal{S}$ such that $x \neq y, x \prec y \in \prec_s$ or $y \prec x \in \prec_s$.

2. *Antisymmetry*: if $a \prec_s b$ and $b \prec_s a$, then $a = b$.

Let $\mathbb{P}_1 = (\mathcal{S}_1, \prec_{s_1})$ and $\mathbb{P}_2 = (\mathcal{S}_2, \prec_{s_2})$ be partial ordered sets, we define the following four operations:

- Union: $\mathbb{P}_1 \cup_{\mathbb{P}} \mathbb{P}_2 = (\mathcal{S}_1 \cup \mathcal{S}_2, \prec_{s_1} \cup \prec_{s_2})$, where $\cup$ is the disjoint union.
- Intersection: $\mathbb{P}_1 \cap_{\mathbb{P}} \mathbb{P}_2 = (\mathcal{S}_1 \cap \mathcal{S}_2, \prec_{s_1} \cap \prec_{s_2})$
- Concatenation: $\mathbb{P}_1 +_{\mathbb{P}} \mathbb{P}_2 = (\mathcal{S}_1 \cup \mathcal{S}_2, \prec_{s_1} \cup \prec_{s_2} \cup \{s_1 \prec s_2 | s_1 \in \mathcal{S}_1 \text{ and } s_2 \in \mathcal{S}_2\})$.
- Linear Extensions: $\mathcal{I}(\mathbb{P}_1) = \{(\mathcal{S}, \prec_s) | \mathcal{S} = \mathcal{S}_1, (\mathcal{S}, \prec_s) \text{ is a sequence and } \prec_{s_1} \subseteq \prec_s\}$.

In other words, the linear extensions of a partial ordered set is the set containing all the possible total orders over the elements of the partial ordered set which keeps true the orderings specified in the partial ordered set.

The *associative* property holds for Union, Intersection and Concatenation.

A serialisation of a process block is a linear extension of the partial order set representing the semantics of such process block.

Definition 4 (Process Block Serialisations): Given a process block B, the set of serialisations of B, written $\Sigma(B) = \{\epsilon | \epsilon \text{ is a sequence and is a serialisation of } B\}$. The function $\Sigma(B)$ is defined as follows:

1) If B is a task t, then $\Sigma(B) = \{(\{t\}, \emptyset)\}$
2) if B is a composite block with sub-blocks $B_1, \ldots, B_n$ let ϵ_i be the projection of ϵ on block B_i (obtained by ignoring all tasks which do not belong to B_i)
   a) If $B = \mathsf{SEQ}(B_1, \ldots, B_n)$, then $\Sigma(B) = \{\epsilon_1 +_{\mathbb{P}} \cdots +_{\mathbb{P}} \epsilon_n | \epsilon_i \in \Sigma(B_i)\}$
   b) If $B = \mathsf{XOR}(B_1, \ldots, B_n)$, then $\Sigma(B) = \Sigma(B_1) \cup \cdots \cup \Sigma(B_n)$
   c) If $B = \mathsf{AND}(B_1, \ldots, B_n)$, then $\bigcup_{\epsilon_1, \ldots, \epsilon_n} \mathcal{I}(\epsilon_1 \cup_{\mathbb{P}} \cdots \cup_{\mathbb{P}} \epsilon_n | \forall \epsilon_i \in \Sigma(B_i))$

Given a structured process model, we can now define its possible executions in terms of the serialisations of its main process block. Each execution of a process model corresponds to one of the serialisations of its main process block to which are attached the pseudo-tasks start and end.

Definition 5 (Execution): Given a structured process P whose main process block is B, the set of possible executions of P is $\Sigma(P) = \{\mathbb{P}_{\mathsf{start}} +_{\mathbb{P}} \epsilon +_{\mathbb{P}} \mathbb{P}_{\mathsf{end}} | \epsilon \in \Sigma(B)\}$ where $\mathbb{P}_{\mathsf{start}} = (\{\mathsf{start}\}, \emptyset)$ and $\mathbb{P}_{\mathsf{end}} = (\{\mathsf{end}\}, \emptyset)$.

Example 2 (Execution): The executions of the structured process illustrated in Fig. 3 are shown in the first column of Table 1.

We use a set of literals to represent the *world* at a given point of the execution of a business process and we call it the *state* of the process.

The state of a process can evolve during the execution of the process. An annotated process is a process whose tasks are associated with consistent sets of literals. These set of literals are called *annotations* [11]

and determine how the state of the process changes when a task is executed.

Definition 6 (Consistent literal set): A set of literals L is *consistent* if and only if it does not contain both l and its complement $\tilde{l}$ for each literal $l \in L$, where $\tilde{l} = a$ if $l = \neg a$, or $\tilde{l} = \neg a$ if $l = a$.

Definition 7 (Annotated process): Let P be a structured process and let T be the set of tasks contained in P. An annotated process is a pair (P, ann), where ann is a partial function associating to each task in T a consistent set of literals: $\mathsf{ann} : T \mapsto 2^{\mathcal{L}}$.

We define the function ann as partial to allow tasks to be annotated by an empty set of literals.

Example 3: Fig. 3 shows a structured process containing four tasks labeled t_1, t_2, t_3 and t_4 and their annotations. The process contains an AND block followed by a task and an XOR block nested within the AND block. The annotations indicate what has to hold after a task is executed. If t_1 is executed, then the literal a has to hold in the state of the process.

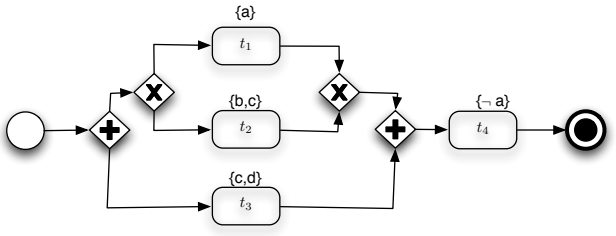

Fig. 3. An annotated process

We represent the state of a process as a pair containing a set of literals and a task. The set of literals describes the *world* holding after the execution of the task contained in the state.

Definition 8 (Process State): The state of a process is represented by a pair $\sigma = (t, L)$ where L is the set of literals holding after the execution of the task t.

We define an update operator (inspired by AGM belief revision [12]).

Definition 9 (Literal set update): Given two consistent sets of literals L_1 and L_2, the update of L_1 with L_2, denoted by $L_1 \oplus L_2$ is a set of literals defined as follows:

$$L_1 \oplus L_2 = L_1 \setminus \{\tilde{l} \mid l \in L_2\} \cup L_2$$

A *trace* represents the evolution of the state of a process during one of its executions. It is represented by a sequence of states, holding at the different stages of an execution.

Definition 10 (Trace): Given an annotated process (P, ann) and an execution sequence $\epsilon = (t_1, \ldots, t_n)$ such that $\epsilon \in \Sigma(P)$, a trace θ is a finite sequence of states: $(\sigma_1, \ldots, \sigma_n)$. Each state of $\sigma_i \in \theta$ contains a set of literals L_i capturing what holds after the execution of a task t_i. Each L_i is a set of literals such that:

1) $L_1 = \mathsf{ann}(t_1)$;
2) $L_{i+1} = L_i \oplus \mathsf{ann}(t_{i+1})$, for $1 \leq i < n$.

We use $\Theta(B, \mathsf{ann})$ to denote the set of traces of a process block B given an annotation function ann. In a similar way we use $\Theta(P, \mathsf{ann})$ to denote the set of traces of a process P.

Example 4: Table 1 shows the traces of the annotated process (P, ann) illustrated in Fig. 3. The first column contains the possible executions of P. The second column the corresponding traces.

2.2 Regulatory Framework

The regulatory framework defines, using a set of conditional obligations, which are the correct ways to achieve a business objective. Each trace contained in a business process defines a possible way of achieving a business objective. We define different extents of compliance depending on the amount of traces, contained in a business process, fulfilling the obligations.

A business process is *fully compliant* if all its traces fulfil the obligations. Similarly, a business process is *partially compliant* if at least one of its traces fulfils the obligations. In the remaining case, when none of the traces of a business process fulfil the obligations, then this business process is *not compliant*.

We represent the regulatory framework as a set of obligations $\circledcirc = \{\mathfrak{O}_1, \ldots \mathfrak{O}_k\}$, where $\mathfrak{O}_i$ represents an obligation. We use a subset of Process Compliance Logic (PCL) [13] to specify the obligations.

Each obligation has a lifeline and a deadline. These elements define the validity period of the obligation. Once triggered by its lifeline, an obligation becomes active. If an obligation is already active, further triggers of its lifeline have no effect. Similarly when its deadline is triggered, an obligation is deactivated. The last state of a trace deactivates every obligation.

An obligation can be of two types: achievement and maintenance. We also consider the punctual obligation which is a special case of both achievement and maintenance. The condition of an obligation, along with its type, determines how an activated obligation should be fulfilled.

The lifeline, deadline and condition of an obligation are represented using propositional formulae over a set of literals.

Definition 11 (Obligations): Let φ_c, φ_b and φ_d be propositional formulae. An obligation $\mathfrak{O}$ is a triple $\mathfrak{O} = \langle \mathcal{O}, \varphi_b, \varphi_d \rangle$ where φ_b is the lifeline condition, φ_d is the deadline condition and $\mathcal{O}$ is one of the following types, where φ_c is the fulfilment condition:

$$\begin{array}{rcll} \mathcal{O} & ::= & O^a(\varphi_c) & \text{achievement} \\ & | & O^m(\varphi_c) & \text{maintenance} \end{array}$$

Example 5 (Achievement Obligation): In a scenario where a customer dines in a restaurant, there exists the obligation that the bill has to be payed before leaving. In this case we can picture the dining at a restaurant as a process and paying the bill as an

$\Sigma(P)$	$\Theta(P, \mathsf{ann})$
$(\mathsf{start}, t_1, t_3, t_4, \mathsf{end})$	$((\mathsf{start}, \emptyset), (t_1, \{a\}), (t_3, \{a, c, d\}), (t_4, \{\neg a, c, d\}), (\mathsf{end}, \{\neg a, c, d\}))$
$(\mathsf{start}, t_2, t_3, t_4, \mathsf{end})$	$((\mathsf{start}, \emptyset), (t_2, \{b, c\}), (t_3, \{b, c, d\}), (t_4, \{\neg a, b, c, d\}), (\mathsf{end}, \{\neg a, b, c, d\}))$
$(\mathsf{start}, t_3, t_1, t_4, \mathsf{end})$	$((\mathsf{start}, \emptyset), (t_3, \{c, d\}), (t_1, \{a, c, d\}), (t_4, \{\neg a, c, d\}), (\mathsf{end}, \{\neg a, c, d\}))$
$(\mathsf{start}, t_3, t_2, t_4, \mathsf{end})$	$((\mathsf{start}, \emptyset), (t_3, \{c, d\}), (t_2, \{b, c, d\}), (t_4.\{\neg a, b, c, d\}), (\mathsf{end}, \{\neg a, b, c, d\}))$

TABLE 1
Executions and Traces of the annotated process in Fig. 3.

achievement obligation triggered when the customer orders. This achievement obligation has to be fulfilled before leaving the restaurant, which corresponds with the deadline.

Example 6 (Maintenance Obligation): While accessing secure data there exists the obligation to have the proper credentials for the whole period. In this case we can see "having the proper credentials" as a maintenance obligation which is triggered when the secure data is being accessed. The deadline is represented by terminating the access to the secure data.

An obligation is activated in a state satisfying the lifeline and deactivated in a state satisfying the deadline. If an obligation is already active, then a state satisfying the lifeline would have no effect. The same applies if an obligation is not active and a state satisfies the deadline.

The propositional formulae, used to represent the lifeline, deadline and condition of an obligation, are satisfied in a state if and only if the interpretation of the propositional variables given by such state makes the propositional formulae true.

Definition 12 (Formula Entailment): Given a state $\sigma = (t, L)$ and a formula φ, $\sigma \models \varphi$ if and only if $\bigwedge x \wedge \bigwedge \neg y \models \varphi$, where each $x \in L$ and each $y \notin L$.

The following exceptions apply:

- Given the state $\sigma = (\mathsf{end}, L)$ and a formula φ, $\sigma \models \varphi$ is always the case if φ is a deadline condition.
- Given the state $\sigma = (\mathsf{end}, L)$ and a formula φ, $\sigma \not\models \varphi$ is always the case if φ is a lifeline condition.

To be fulfilled, achievement obligations' fulfilment condition needs to be satisfied in at least one state within their activation period. Once fulfilled an achievement obligation is deactivated. Differently, maintenance obligations need that their fulfilment condition is satisfied in each state of their activation period and are deactivated only when the deadline is triggered. In case an obligation has multiple activation periods, it must be fulfilled in each of them.

In legal theory it is often the case that a compensation is provided when an obligation is not fulfilled. Compensations represent additional obligations to be fulfilled in case others were not. However, in the present paper we do not include compensations in the sub-problem being used to analyse the complexity of the business process regulatory compliance. Thus when an obligation is not fulfilled, we consider a trace to be violating such obligation and we avoid further analysis. Because of this limitation we also avoid to deal with *perdurant* obligations, which are still considered active even when not fulfilled. Thus when a trace does not fulfil an obligation, we revert the status of the obligation to inactive, and we consider the trace to be not compliant with the regulatory framework.

Definition 13 (Obligation Fulfilment): Given an obligation $\mathfrak{O} = \langle \mathcal{O}, \varphi_b, \varphi_d \rangle$ and a trace θ, θ fulfils $\mathfrak{O}$, written $\theta \vdash \mathfrak{O}$, iff:

- $\mathcal{O} = O^a(\varphi_c)$: $\theta \vdash \langle O^a(\varphi_c), \varphi_b, \varphi_d \rangle$ iff:
  $\forall \sigma_i \in \theta$ where $\sigma_i \models \varphi_b$ implies $\exists \sigma_j \in \theta$ such that $\sigma_j \models \varphi_c$ and $\sigma_j \succ \sigma_i$, and $\neg \exists \sigma_h \in \theta$ such that $\sigma_h \models \varphi_d$ and $\sigma_i \prec \sigma_h \prec \sigma_j$.
- $\mathcal{O} = O^m(\varphi_c)$: $\theta \vdash \langle O^m(\varphi_c), \varphi_b, \varphi_d \rangle$ iff:
  $\forall \sigma_i \in \theta$ where $\sigma_i \models \varphi_b$ implies $\exists \sigma_h \in \theta$ such that $\sigma_h \models \varphi_d$ and $\forall \sigma_j \in \theta$ such that $\sigma_j \models \varphi_c$ and $\sigma_i \prec \sigma_j \preceq \sigma_h$.

Otherwise θ does not fulfil $\mathfrak{O}$, written $\theta \nvdash \mathfrak{O}$.

An alternative way of representing the activation period of an obligation is by using a finite state automaton. Fig. 4.(a) shows the automaton modelling the activation period of an achievement obligation. Fig. 4.(b) represents the automaton modelling the activation period of a maintenance obligation. We can notice that in both cases, an obligation becomes active only if is inactive and a state triggering the lifeline is found. Finding such state while the obligation is already active has no impact on the activation period of the obligation. Similarly, when an obligation is inactive, states triggering the deadline or the condition of an obligation do not influence its state.

The activation period always terminates when a state fulfilling the deadline is found. Moreover for achievement obligations, the activation period can terminate if a state fulfilling the condition is found. Differently for a maintenance obligation, an active obligation becomes not fulfilled and inactive when a state not fulfilling the condition is found.

The two automata are consistent with the semantics of Definition 13. Notice that for the sake of clarity we avoid representing explicitly in the figure the transitions which would not have changed the state of the automaton. For instance a state σ of a trace, where $\sigma \models \varphi_b$, reading σ does not change the state of the automaton if the automaton is in the *Active* state. Also notice that the ϵ in the automata mean that from the states *Fulfilled* and *Not Fulfilled*, the state of the automata becomes *Inactive* without reading and consuming a state of a trace.Once the obligation is

either fulfilled or not, the state of the automaton is brought back to *Inactive* thanks to the ϵ transitions. This represents that an obligation can be activated multiple times by a trace.

Given a trace and an obligation, the automaton in Fig. 4 can be used to determine whether an obligation is active or inactive with respect to the states of the given trace. For this reason we avoid to represent any final state in the automaton.

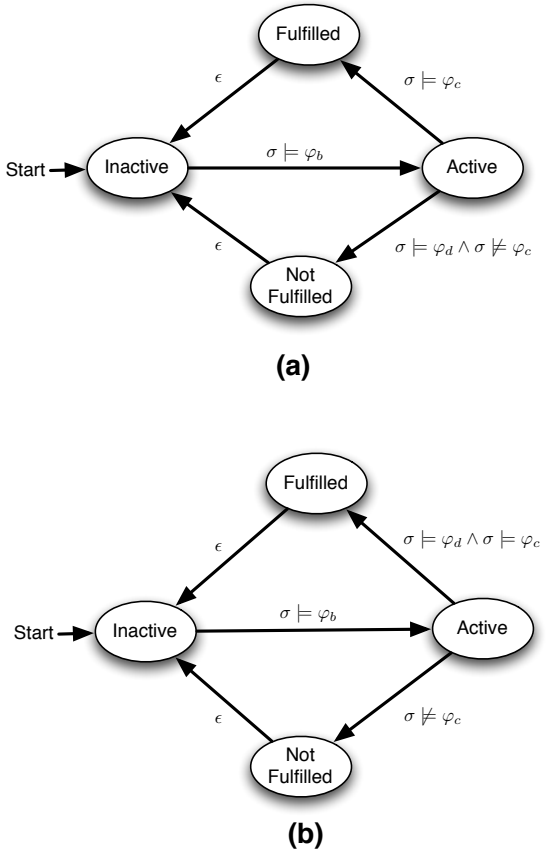

Fig. 4. Activation Periods using Finite State Automaton

2.2.1 Punctual Obligations

Punctual obligations are a special type of obligations. The peculiarity of this type of obligation is, that it has to be fulfilled in exactly one state. This means that such obligations become active for exactly one state. To allow such behaviour, the deadline of punctual obligations can be satisfied by any state. Punctual obligations are a particular kind of both achievement and maintenance. If these obligations can be fulfilled in only one state, independently on the type the condition of the obligation has to be achieved in such state.

For this reason, in the following definition we represent the deadline of a punctual obligation using the tautology formula $\top$. Knowing that the activation period of this type of obligation is limited to a single state, we define define the semantics of punctual obligations as follows:

Definition 14 (Punctual Obligation Fulfilment): Given a punctual obligation $\mathfrak{O} = \langle O^p(\varphi_c), \varphi_b, \top \rangle$ and a trace $\theta = (\sigma_0, ..., \sigma_n)$, $\theta \vdash \mathfrak{O}$ iff:

$$\forall \sigma_i \in \theta, 0 \leq i \leq n\colon \text{ if } \sigma_i \models \varphi_b \text{ then } \sigma_{i+1} \models \varphi_c.$$

Otherwise $\theta \nvdash \mathfrak{O}$.

Notice that following from Definition 7 and Definition 10, a trace of a process model always ends with a state containing the pseudo-task end. Thus the last state of a proper trace would never trigger the lifeline of a punctual obligation according to Definition 12, hence verifying $\sigma_{i+1} \models \varphi_c$ is still possible.

2.2.2 Set Compliance

A trace is compliant with a set of obligations if it fulfils all the obligations belonging to the set. Note that according to Definition 13 (and Definition 14 for punctual obligations), an obligation never activated by a trace is considered to be fulfilled by such trace.

Definition 15 (Set Fulfilment): Given a trace θ and a set of obligations $\circledcirc = \{\mathfrak{O}_1, \ldots, \mathfrak{O}_n\}$,

$$\theta \vdash \circledcirc \text{ iff } \forall \mathfrak{O}_i \in \circledcirc, (\theta \vdash \mathfrak{O}_i)$$

Otherwise $\theta \nvdash \circledcirc$.

2.2.3 Types of Compliance

Given a set of obligations, an annotated process can be fully compliant, partially compliant or not compliant with such set. An annotated process is fully compliant if each trace is compliant with the set of obligations. It is partially compliant, if at least one trace of the annotated process is compliant with the set of obligations. If none of the traces are compliant with the set of obligations, then a process is not compliant.

Definition 16 (Process Set Compliance): Given an annotated process (P, ann) and a set of obligations $\circledcirc$.

- **Full Compliance**:

$$(P, \mathsf{ann}) \vdash^{\mathrm{F}} \circledcirc \text{ iff } \forall \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc.$$

- **Partial Compliance**:

$$(P, \mathsf{ann}) \vdash^{\mathrm{P}} \circledcirc \text{ iff } \exists \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc.$$

- **Non Compliance**:

$$(P, \mathsf{ann}) \nvdash \circledcirc \text{ iff } \neg\exists \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc.$$

In first order logic $\forall x$ does not necessarily implies $\exists x$, because the former is true while considering the empty set but the latter is not. However, in the present context where we want to classify whether a business process is fully, partially or not compliant with a set of obligations, we can safely say that full compliance implies partial compliance. Following from Definition 2 a business process contains at least a trace, hence the case where a business process would be fully

compliant but not partially due to its set of traces being empty does not apply in this setting.

Observation 1: In business process regulatory compliance, whenever a process is fully compliant with a set of obligations, then such a process is also partially compliant with the same set.

3 Verifying Partial Compliance is NP-complete

In this section we prove that verifying whether a structured annotated process is partially compliant with a set of obligations is an **NP**-complete problem.

Definition 17 (NP-complete): A decision problem is **NP**-complete if it is in the set of **NP** problems and if every problem in **NP** is polynomial-time many-one reducible to it.

To prove the **NP**-completeness of the problem of verifying whether a business process is partially compliant with a regulatory framework, first we show that the problem is in **NP** and second that another **NP**-complete problem is polynomial-time many-one reducible to it.

3.1 NP Membership

To prove membership in **NP**, we need to show that an annotated business process is partially compliant with a set of obligations if and only if there is a certificate whose size is at most polynomial in terms of the length of the input (comprising the annotated business process and the set of obligations) with which we can check whether it fulfils the regulatory framework in polynomial time. As a certificate we will choose a particular trace satisfying the obligations composing the regulatory framework. The size of any proper traces is always polynomial with respect to the business process considered and the set of literals.

Since the type of business processes considered in the present paper is structured, cycles are not allowed and a task belonging to this type of processes can be executed at most once. Thus given a structured process, the maximum length of a proper trace is not greater than the number of the tasks contained in such process. Additionally the size of the states contained in a trace is at most as big as the set of literals used in the process. Before verifying the compliance of such a trace, we first need to check that the trace is indeed a valid trace for the annotated business process (done by Algorithm 1 below) and that the trace does indeed satisfy the obligations (done by Algorithm 2 described below). We will further show that the time complexity of both algorithms is at most polynomial in the size of the input, thus concluding that verifying partial compliance is in **NP**.

Claim 1: Verifying whether a structured business process is partially compliant with a regulatory framework is in **NP**.

3.1.1 Verifying the Validity

We hereby describe Algorithm 1 which checks whether a certificate is a valid trace of an annotated structured business process.

Algorithm 1: Given a trace $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ where $\sigma_{start} = (\mathsf{start}, L_0)$ and $\sigma_{end} = (\mathsf{end}, L_{n+1})$, an execution $\epsilon = (t_1, \ldots, t_n)$ representing the corresponding serialisation of θ, and an annotated process (P, ann) where B is the main process block of P, the following algorithm $A_1(\theta, \epsilon, (P, \mathsf{ann}), B)$ decides if θ is a valid trace of (P, ann).

Algorithm A_1

1: **if** $\mathbf{P_1}(\epsilon, \mathbf{B})$ and $\mathbf{P_2}(\theta, (\mathbf{P}, \mathsf{ann}))$ **then**
2: **return** $\theta \in \Theta(P, \mathsf{ann})$
3: **else**
4: **return** $\theta \notin \Theta(P, \mathsf{ann})$
5: **end if**

$\mathbf{P_1}(\epsilon, \mathbf{B})$ verifies wether ϵ is a correct serialisation of B. $\mathbf{P_1}$ returns $true$ or $false$ accordingly to the result and uses the following recursive procedure:

Procedure 1: $\mathbf{P_1}(\epsilon, \mathbf{B})$

1) if $B = t$, then ϵ is valid if $\epsilon = (t)$
2) if B is a composite block with sub-blocks $B_1, \ldots, B_n$ let ϵ_i be the projection of ϵ on block B_i (obtained by ignoring all tasks which do not belong to B_i)
   a) if $B = \mathsf{SEQ}(B_1, \ldots, B_n)$ then ϵ is valid if it is the concatenation of $\epsilon_1, \ldots, \epsilon_k$ and each ϵ_i is a valid serialisation for B_i
   b) if $B = \mathsf{XOR}(B_1, \ldots, B_n)$, then ϵ is valid if exactly one ϵ_i is non-empty and that ϵ_i is valid for B_i
   c) if $B = \mathsf{AND}(B_1, \ldots, B_n)$, then ϵ is valid if the set of tasks in ϵ is the disjoint union of the sets of tasks in ϵ_i (for each i) and each ϵ_i is a valid serialisation for B_i

$\mathbf{P_2}(\theta, (\mathbf{P}, \mathsf{ann}))$ verifies wether the sequence of states in θ is valid for (P, ann):

Procedure 2: $\mathbf{P_2}(\theta, (\mathbf{P}, \mathsf{ann}))$

- $L_0 = \emptyset$
- For each $L_i \in \theta$ and $i > 0$: $L_i = L_{i-1} \oplus \mathsf{ann}(t_i)$
- $L_n = L_{n+1}$

$\mathbf{P_2}$ returns $true$ if all of these properties hold and $false$ otherwise.

Correctness:

Proof:

The correctness of procedure $\mathbf{P_1}$ follows from Definition 4. The first part of the procedure verifies the first property of the definition. The uniqueness of the task is instead given by construction of a process model as in Definition 1. The second part of the procedure verifies the three properties of the Definition 4.

The correctness of procedure $\mathbf{P_2}$ follows directly from Definition 10.

The correctness of the algorithm A_1 follows directly from Definitions 4 and 10. □

Complexity:

To analyse the complexity of checking whether a trace is a valid serialisation of a business process whose main process block is B (procedure $\mathbf{P_1}$), consider the tree reflecting the hierarchical structure of a process block. If B is a single task, the tree consists of a single node representing a task. Otherwise the tree has a root corresponding to B and subtrees representing the different sub-blocks B_i of B. The recursive procedure spends polynomial time (as a function of n, where n is the number of tasks in B) for each node of the tree for pre-processing, launching the recursive calls and recombining the results. Since the size of the tree itself is $\mathbf{O}(n)$ the overall time for the procedure is polynomial in n.

Procedure $\mathbf{P_2}$ can clearly be executed in time polynomial in $n \times k$ where k is the size of the set of literals. Therefore the time complexity of Algorithm 1 is $\mathbf{O}(n \times k)$, which is polynomial in the size of the input.

3.1.2 Verifying the Fulfilment

In the present sub-section we describe Algorithm 2, which verifies whether a certificate fulfils the obligations contained in a regulatory framework.

Algorithm 2: Given a set of obligations $\circledcirc$ and a trace $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ such that $\sigma_{start} = (\mathsf{start}, L_0)$ and $\theta \in \Theta(P, \mathsf{ann})$, the algorithm $A_2(\theta, \circledcirc)$ is defined as follows (in the following, Ob denotes the set of active obligations and we treat θ as a vector):

Algorithm A_2

1: $\mathsf{Ob} = \emptyset$
2: **for** $j = 0; j \leq n+1; j++$ **do**
3: $\quad \sigma_i = \theta[j]$
4: $\quad$ **for each** $\langle \mathcal{O}, \varphi_b, \varphi_d \rangle$ in Ob **do**
5: $\quad\quad$ **if** $\mathcal{O} = O^a(\varphi_c)$ **then**
6: $\quad\quad\quad$ **if** $\sigma_i \models \varphi_c$ **then**
7: $\quad\quad\quad\quad \mathsf{Ob} = \mathsf{Ob} \setminus \langle O^a(\varphi_c), \varphi_b, \varphi_d \rangle$
8: $\quad\quad\quad$ **else**
9: $\quad\quad\quad\quad$ **if** $\sigma_i \models \varphi_d$ **then**
10: $\quad\quad\quad\quad\quad$ **return** $\theta \not\vdash \circledcirc$
11: $\quad\quad\quad\quad$ **end if**
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **else**
14: $\quad\quad\quad$ **if** $\mathcal{O} = O^m(\varphi_c)$ **then**
15: $\quad\quad\quad\quad$ **if** $\sigma_i \not\models \varphi_c$ **then**
16: $\quad\quad\quad\quad\quad$ **return** $\theta \not\vdash \circledcirc$
17: $\quad\quad\quad\quad$ **end if**
18: $\quad\quad\quad\quad$ **if** $\sigma_i \models \varphi_d$ **then**
19: $\quad\quad\quad\quad\quad \mathsf{Ob} = \mathsf{Ob} \setminus \langle O^m(\varphi_c), \varphi_b, \varphi_d \rangle$
20: $\quad\quad\quad\quad$ **end if**
21: $\quad\quad\quad$ **end if**
22: $\quad\quad$ **end if**
23: $\quad$ **end for each**
24: $\quad$ **for each** $\langle \mathcal{O}, \varphi_b, \varphi_d \rangle$ in $\circledcirc$ **do**
25: $\quad\quad$ **if** $\sigma_i \models \varphi_b$ **then**
26: $\quad\quad\quad \mathsf{Ob} = \mathsf{Ob} \cup \langle \mathcal{O}, \varphi_b, \varphi_d \rangle$
27: $\quad\quad$ **end if**
28: $\quad$ **end for each**
29: **end for**
30: **return** $\theta \vdash \circledcirc$;

Algorithm 2 identifies wether a certificate fulfils a set of obligations. If the certificate is a valid trace of a structured process, then following from Definition 16, the fact that the certificate fulfils the set of obligations is a sufficient condition to say that the structured process is partially compliant with the regulatory framework containing such set of obligations.

Correctness:

Proof: **Soundness:** $\theta \vdash \circledcirc \Rightarrow A_2(\theta, \circledcirc) = \theta \vdash \circledcirc$. Direct proof:

From the hypothesis we know that $\theta \vdash \circledcirc$, hence we know that $\forall \mathbf{O} \in \circledcirc, \theta \vdash \mathbf{O}$ from Definition 15. Independently of the type of an obligation, if its lifeline is never triggered, then it is fulfilled (Definition 13). This is captured in Algorithm 2 because the lines returning *not compliant* are inside the for-each cycle (from line 4 to 28), which requires the obligation's lifeline to be triggered.

In the case the lifeline of an obligation is triggered, we distinguish whether the obligation to fulfill it is an achievement or a maintenance obligation.

- **Achievement**: In this case the lines from 5 to 12 are concerned. Among those, the only line returning *not compliant* is 10, which is executed when the state analyzed by the algorithm satisfies the deadline condition. However we know that each obligation is fulfilled by the trace and from Definition 13 it follows that: $\exists \sigma_j \in \theta$ such that $\sigma_j \models \varphi_c$ and $\neg \exists \sigma_h \in \theta$ such that $\sigma_h \models \varphi_d$ and $\sigma_h \prec \sigma_j$. Thus because the algorithm analyzes the states of the trace in order, the condition of line 9 cannot be fulfilled before the condition at line 6. When the condition at line 6 is fulfilled, it removes the obligation from the cycle, hence preventing the result *not compliant*.
- **Maintenance**: In this case the lines from 14 to 21 are concerned. Among those, the only line returning *not compliant* is 16, which is executed when the state analyzed by the algorithm does not satisfy the condition of the obligation. However we know that each obligation is fulfilled by the trace and from Definition 13 it follows that: $\exists \sigma_h \in \theta$ such that $\sigma_h \models \varphi_d$ and $\forall \sigma_j \in \theta$ such that $\sigma_j \models \varphi_c$ and $\sigma_j \preceq \sigma_h$. Thus the condition at line 15 is never satisfied in the same state or in one preceding a state satisfying the condition at line 18, which removes the obligation and prevents the result *not compliant*.

Thus we have shown that if the trace being analyzed fulfills the set of obligations, then Algorithm 2 returns *compliant* as result. □

Proof: **Completeness:** $A_2(\theta, \circledcirc) = \theta \vdash \circledcirc \Rightarrow \theta \vdash \circledcirc$. Proof by contradiction:

We assume that $\theta \not\vdash \circledcirc$. From this assumption, it follows that exists an obligation $\mathbf{O}$ in $\circledcirc$ such that $\theta \not\vdash \mathbf{O}$ (Definition 15). Independently of the type of the obligation, in order not to be fulfilled by a trace, the obligation's lifeline has to be triggered by such trace at least once (Definition 13). We analyse independently

two cases, depending on the type of the obligation to be fulfilled:

- **Achievement**: If an achievement obligation is not fulfilled by a trace, it means that it is triggered in a state σ_h and the following holds: $\exists \sigma_i \in \theta$ such that $\sigma_i \succ \sigma_h$ and $\sigma_i \models \varphi_d$ and $\neg\exists \sigma_j \in \theta$ such that $\sigma_j \models \varphi_c$ and $\sigma_h \prec \sigma_j \prec \sigma_i$ (Definition 13).
  Because the obligation's lifeline is triggered, we know that the for-each cycle (from line 4 to line 28) is entered. In particular, because the obligation to be fulfilled is an achievement, we consider the lines between 5 and 12 of Algorithm 2.
  Because the algorithm analyses the states of the trace in order and this achievement obligation is not fulfilled by the trace, we have that line 6 is never fulfilled before the condition in line 9. Thus the obligation is never removed from the loop which will end in fulfilling line 9 and executing line 10 which returns $\theta \not\vdash \circledcirc$. Line 9 is guaranteed to be fulfilled because the last state of the trace always fulfils the deadline condition (Definition 13). In this case we have that $A_2(\theta, \circledcirc) = \theta \not\vdash \circledcirc$.
- **Maintenance**: If a maintenance obligation is not fulfilled by a trace, it means that it has been triggered in a state σ_h and the following holds: $\exists \sigma_i \in \theta$ such that $\sigma_i \succ \sigma_h$ and $\sigma_i \models \varphi_d$ and $\exists \sigma_j \in \theta$ such that $\sigma_j \not\models \varphi_c$ and $\sigma_h \prec \sigma_j \preceq \sigma_i$ (Definition 13).
  Because the obligation's lifeline is triggered, we know that the for-each cycle (from line 4 to line 28) is entered. In particular, because the obligation to fulfil is a maintenance, we consider the lines between 14 and 21 of Algorithm 2.
  Because the algorithm analyses the states of the trace in order and this maintenance obligation is not fulfilled by the trace, we have that line 18 is never fulfilled before the condition in line 15. Thus the obligation is never removed from the loop which will end in fulfilling line 15 is and executing 16 which returns $\theta \not\vdash \circledcirc$. Line 15 is guaranteed to be fulfilled because the last state of the trace always fulfils the deadline condition (Definition 13). In this case we have that $A_2(\theta, \circledcirc) = \theta \not\vdash \circledcirc$.

We have shown that independently of the type of the obligation to be fulfilled, if we assume that $\theta \not\vdash \circledcirc$, then $A_2(\theta, \circledcirc) = \theta \not\vdash \circledcirc$. The result contradicts the premise that $A_2(\theta, \circledcirc) = \theta \vdash \circledcirc$, hence $A_2(\theta, \circledcirc) = \theta \vdash \circledcirc$, then $\theta \vdash \circledcirc$ is true.

□

Complexity:
The time complexity of checking whether a trace is compliant with the set of obligations using Algorithm 2 is at most $\mathbf{O}(n \times o \times T)$ where n is the number of tasks in the process, o is the number of obligations and T is the maximum time to check whether a state satisfies a formula. Since checking whether a state satisfies a propositional formula can be done in time that is at most polynomial (in fact linear) in the length of the formula, the above asymptotic time bound is at most polynomial in the length of the input (which includes the annotated business process and the set of obligations, including the associated formulas).

We conclude that given a yes-instance of the partial compliance problem, there is a certificate of size polynomial in the length of the input (namely the trace that satisfies the obligations) for which we can check compliance in time polynomial in the length of the input. We conclude that verifying Partial Compliance is indeed in **NP**.

3.2 NP-Hardness

After having proven the **NP** membership of the problem in the previous sub-section, to prove that the problem is **NP**-complete we have to show that the problem is **NP**-Hard.

To prove the **NP**-hardness of *Verifying Partial Compliance*, we show that the problem of deciding whether a directed graph contains an hamiltonian path (another **NP**-Complete problem) is polynomial-time many-one reducible to it.

In graph theory, the hamiltonian path problem is the decision problem of determining wether an hamiltonian path exists in a given directed graph. This problem is part of the commonly known **NP**-complete problems.

In a directed graph $G = (N, D)$ where N is a set of nodes and D is a set of directed edges represented as a binary relation $N \times N$, a hamiltonian path is a path in G that visits each node exactly once. A path can travel from one node to another if there exists a directed edge starting from a node and pointing to the one following it in the path.

Definition 18 (Hamiltonian Path): Let $G = (N, D)$ be a directed graph where the size of N is n. A hamiltonian path $ham = (v_1; \ldots; v_n)$ satisfies the following properties:

1) $N = \{v_1, \ldots, v_n\}$
2) $\forall i, j((v_i, v_j \in ham \wedge j = i + 1), ((v_i, v_j) \in D)$

Claim 2: Hamiltonian Path Problem $\leq_p$ Verifying Partial Compliance

Given a directed graph $G = (N, D)$, we reduce the problem of deciding whether G contains an hamiltonian path to the decision problem of deciding whether an annotated structured process (P, ann) is partially compliant with a regulatory framework.

3.2.1 Reduction

Given a directed graph $G = (N, D)$, it can be translated to an annotated structured process (P, ann) as follows:

1 Assuming that B is the main process block of P, B contains a task labeled $Node_i$ for each vertex

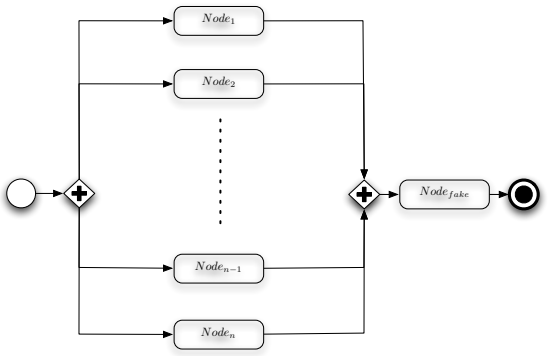

Fig. 5. Hamiltonian path problem as verifying partial compliance.

v_i contained in N. In addition, B also contains a task labeled $Node_{fake}$.
The main process block B is structured as an AND block followed by a task. The AND block contains in each branch a single task $Node_i$. The task $Node_{fake}$, follows the AND block: $\mathsf{SEQ}(\mathsf{AND}(Node_1, \ldots, Node_n), Node_{fake})$.
Intuitively a serialisation of the AND block represents a tentative hamiltonian path. The task $Node_{fake}$ has no correspondence in the original graph, its purpose is to terminate the serialisation of vertices. Annotations and obligations are used to verify that two adjacent nodes in the serialisation can be indeed also adjacent in an hamiltonian path (explained in detail in **2**). Thus, since the last vertex in an hamiltonian path does not need a successor, the task $Node_{fake}$ allows to ignore the obligations triggered by such vertex.

2 In this reduction we use the annotations to identify which node is being selected in the sequence constituting the tentative hamiltonian path. Thus we use for the annotations a language containing a literal for each node in G. The annotation of each task in (P, ann) is the following:

- $\forall i | 1 \leq i \leq k, \mathsf{ann}(Node_i) = \{\neg l_1, \ldots, \neg l_n\} \oplus \{l_i\}$
- $\mathsf{ann}(Node_{fake}) = \{\neg l_1, \ldots, \neg l_n\}$

The obligations are used to represent the directed edges departing from a vertex, in other words which vertices are the suitable successors in the hamiltonian path. The set $\circledcirc$ contains the following obligations:

- $\forall v_i, v_j | (v_i, v_j) \notin D, \langle O^p(\neg l_j), l_i, \top \rangle$

Notice that the annotation of $Node_{fake}$ is composed in such a way that it always fulfils all the obligations triggered by the task serialised last in the AND block.

We claim that there exists a trace $\theta \in \Theta(P, \mathsf{ann})$ such that $\theta \vdash \circledcirc$ if and only if G has an hamiltonian path.

Correctness:

Here we prove the soundness ($(P, \mathsf{ann}) \overset{\mathrm{P}}{\vdash} \circledcirc \Rightarrow \exists ham$) and the completeness ($\exists ham \Rightarrow (P, \mathsf{ann}) \overset{\mathrm{P}}{\vdash} \circledcirc$) of our reduction. We refer to the two conditions stated in Definition 18 as (1) and (2) respectively.

Proof:

Soundness: $(P, \mathsf{ann}) \overset{\mathrm{P}}{\vdash} \circledcirc \Rightarrow \exists ham$
Direct Proof:

The condition (1) is fulfilled by construction of the reduction because all the tasks representing the nodes of the graph are included in an AND block which has always to be serialised. Thus each possible serialisation of the process and consequently each of its traces contains each of the tasks in the AND block exactly once (Definition 4).

From the hypothesis it follows that $\exists \theta \in \Theta(P, \mathsf{ann})$ such that $\theta \vdash \circledcirc$ (Definition 13). Thus it also follows that $\forall \mathcal{O} \in \circledcirc, \theta \vdash \mathcal{O}$ (Definition 15).

By construction of the reduction we know that each obligation in $\circledcirc$ is of type punctual. We know that there exists a trace $\theta = ((\mathsf{start}, L_0), (Node_{i_1}, L_1), \ldots, (Node_{i_n}, L_n), (Node_{fake}, L_{n+1}), (\mathsf{end}, L_{n+2}))$ fulfilling each of the punctual obligations in $\circledcirc$, hence for each $(Node_{i_k}, L_k)$ and $(Node_{i_{k+1}}, L_{k+1})$ in θ, it follows that there is no punctual obligation $\langle O^p(\neg l_{i_{k+1}}), l_{i_k}, \top \rangle \in \circledcirc$. Thus by construction of the reduction it follows that for each $(Node_{i_k}, L_k)$ and $(Node_{i_{k+1}}, L_{k+1})$ in θ, there exists $(v_{i_k}, v_{i_{k+1}}) \in D$, which fulfils the condition (2) because there exists and edge between v_{i_k} and $v_{i_{k+1}}$ in G.

Because there exists at least a trace in the process fulfilling the conditions (1) and (2), it follows that $\exists ham$ in G. □

Proof: **Completeness:** $\exists ham \Rightarrow (P, \mathsf{ann}) \overset{\mathrm{P}}{\vdash} \circledcirc$
Direct Proof:

From the hypothesis we know that $\exists ham = (v_1; \ldots; v_n)$ satisfying conditions (1) and (2) in Definition 18. If we substitute each v_i in ham with $Node_i$ we obtain a valid process block serialisation $\epsilon = (Node_1, \ldots, Node_n)$ for the AND block of the process obtained with the reduction (Definition 4). If we append $Node_{fake}$ at the end of ϵ obtaining $\epsilon' = (Node_1, \ldots, Node_n, Node_{fake})$, we have that $\epsilon' \in \Sigma(P)$ is a valid execution of the main block of P (Definition 5). Using the annotations in P we can use ϵ' to construct a trace $\theta = ((\mathsf{start}, L_0), (Node_1, L_1), \ldots, (Node_k, L_k), (Node_{fake}, L_{n+1}), (\mathsf{end}, L_{n+2}))$ which is a valid trace of P (Definition 10).

Because (2) is fulfilled we know that for each $v_i, v_{i+1} \in ham$ there exists $(v_i, v_{i+1}) \in D$. From this and the construction of the reduction it follows that there is no obligation $\mathbf{O} = \langle O^p(\neg l_{i+1}), l_i, \top \rangle$. By construction of the reduction we know that only punctual obligations are allowed, hence it is sufficient to consider two neighbouring states in the trace to verify their fulfilment. By construction of the annotations we know that for each $(Node_i, L_i)$ and $(Node_{i+1}, L_{i+1})$ in

θ, only $\mathfrak{O}$ would not be fulfilled in θ (Definition 13). Thus, because there is no such obligation like $\mathfrak{O}$ for any $(Node_i, L_i)$ and $(Node_{i+1}, L_{i+1})$ in θ, then $\theta \vdash \circledcirc$ (Definition 14).

Because there exists at least a trace of P fulfilling each obligation in $\circledcirc$, then $(P, \mathsf{ann}) \overset{\mathrm{P}}{\vdash} \circledcirc$.

□

Complexity:

The complexity of reducing the input of an hamiltonian path problem to a problem deciding whether an annotated structured process is partially compliant with a regulatory framework is polynomial in terms of the size of the input. The time complexity of constructing the annotated structured business process is $\mathbf{O}(n^2)$, where n is the number of vertices in G. The time complexity of constructing the regulatory framework is $\mathbf{O}(e)$, where e is the number of edges in G. Since e is at most $n \times n$, we can conclude that the time complexity of the reduction is $\mathbf{O}(n^2)$.

4 Verifying Full Compliance is coNP-complete

In this section we prove that the problem of verifying whether a structured annotated process is fully compliant with a set of obligations is a co**NP**-complete problem.

Definition 19 (coNP-complete): A decision problem is co**NP**-complete if it is in co**NP** and if every problem in co**NP** is polynomial-time many-one reducible to it. A decision problem is in co**NP** if and only if its complement is in the complexity class **NP**.

To prove that a decision problem is co**NP**-complete we first show that the complementary problem belongs to **NP** and second we show that the the *tautology problem*, a known co**NP**-complete problem, is reducible to the problem of verifying whether a business process is fully compliant with a set of obligations.

4.1 *Not* full compliance is in NP

We define the complement of the problem of verifying full compliance as its negation. This means verifying whether a structured business process is not fully compliant with a given regulatory framework, which is composed by a set of obligations.

According to Definition 16, full compliance with respect to a set of obligations $\circledcirc$ is defined as follows:

Full Compliance:

$$(P, \mathsf{ann}) \overset{\mathrm{F}}{\vdash} \circledcirc \text{ iff } \forall \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc.$$

Therefore, we define the complement of full compliance, *not full compliance*, as follows:

*Definition 20 (*Not *Full Compliance):* Given an annotated process (P, ann) and a set of obligations $\circledcirc$.

***Not* Full Compliance**:

$$\neg(P, \mathsf{ann}) \overset{\mathrm{F}}{\vdash} \circledcirc \text{ iff } \exists \theta \in \Theta(P, \mathsf{ann}), \theta \nvdash \circledcirc.$$

From Definition 20 it follows that to verify not full compliance it is sufficient to show that there exists a trace belonging to the structured business process which does not fulfil the regulatory framework.

4.1.1 *NP Membership*

To prove membership in **NP**, we show that an annotated business process is not fully compliant with a set of obligations if and only if there is a certificate whose size is at most polynomial in terms of the length of the input and which can be verified in polynomial time. As a certificate we choose a particular trace. Moreover we need to show that verifying whether it is a valid trace of the annotated business process and is not fully compliant can be done in polynomial time.

Proof:

To verify whether a certificate is indeed a valid trace of the annotated business process we can reuse Algorithm 1 introduced in Section 3.1.

In the same way, we can reuse Algorithm 2 to verify the not full compliance of the annotated business process. This can be done because Algorithm 2 returns either $\theta \vdash \circledcirc$ or $\theta \nvdash \circledcirc$. Thus in case the algorithm returns $\theta \nvdash \circledcirc$, according to Lemma 20 the certificate proves that the structured annotated process is not fully compliant with the set of obligations.

In Section 3.1 it is proven that the complexity of both Algorithms 1 and 2 is polynomial in the length of the input, hence the problem of verifying whether a structured annotated process is not fully compliant is indeed in **NP**. □

4.2 Completeness of Full Compliance

To show that the problem of verifying whether a business process is fully compliant with a set of obligations is co**NP**-complete, we reduce the tautology problem to it.

Definition 21 (Tautology): A formula of propositional logic is a tautology if the formula itself is always true regardless of which valuation is used for the propositional variables.

4.2.1 *Reduction*

Let φ be a propositional formula for which we want to verify whether it is a tautology or not, and let L be the set of literals contained in φ. We include in L only the positive version of a literal, for instance if l or $\neg l$ are contained in φ, then only l is included in L.

For each literal l belonging to L we construct an XOR block containing two tasks, one labeled and containing in its annotation the positive literal (i.e. l) and the other the negative literal (i.e. $\neg l$). All the XOR blocks constructed from L are then included within a single AND block. This AND block is in turn followed by a task labeled "test" and containing a single literal in its annotation: l_{test}. The sequence containing the AND block and the task *test* is then enclosed within a

start and an end, composing the annotated business process model (P, ann), graphically represented in Figure 6.

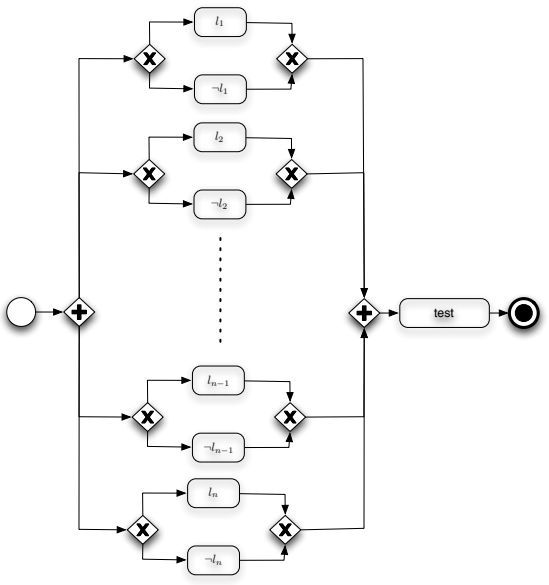

Fig. 6. Tautology problem as verifying full compliance.

The set of obligations, to which the constructed business process has to be verified to be fully compliant with, is composed of a single obligation constructed as follows from the propositional formula φ:

$$\langle O^a(\varphi), l_{test}, \bot \rangle$$

We claim that for all traces $\theta \in \Theta(P, \mathsf{ann})$ we have $\theta \vdash \circledcirc$ if and only if φ is a tautology.

Correctness:

Here we prove the soundness $((P, \mathsf{ann}) \overset{\mathrm{F}}{\vdash} \circledcirc \Rightarrow \varphi \equiv \top)$ and the completeness $(\varphi \equiv \top \Rightarrow (P, \mathsf{ann}) \overset{\mathrm{F}}{\vdash} \circledcirc)$ of our reduction.

Proof:

Soundness: $(P, \mathsf{ann}) \overset{\mathrm{F}}{\vdash} \circledcirc \Rightarrow \varphi \equiv \top$

From the hypothesis and Definition 16, we know that each trace of the business process (P, ann) fulfils the obligations in $\circledcirc$. Following from the construction of the reduction we know that the only obligation belonging to $\circledcirc$ is $\langle O^a(\varphi), l_{test}, \bot \rangle$.

From Definition 10 and the construction of the reduction we know that each trace of P contains the task l_{test}. Therefore, according to Definition 13, in order for the obligation $\langle O^a(\varphi), l_{test}, \bot \rangle$ to be fulfilled each trace contains a state following the one where l_{test} appears.

From the construction of the reduction, in particular how (P, ann) is constructed, and Definition 5 we have that in the only state following the one where l_{test} appears the first time, the set of literals associated to that state corresponds to an interpretation of the propositions contained in φ. Moreover, again from the construction of the reduction, we know that in all the traces of (P, ann), all the possible combinations of interpreting the propositions belonging to φ are considered.

Therefore, since the obligation $\langle O^a(\varphi), l_{test}, \bot \rangle$ is fulfilled by each trace and each trace corresponds to an interpretation, it follows from Definition 21 that φ is indeed a tautology. □

Proof:

Completeness: $\varphi \equiv \top \Rightarrow (P, \mathsf{ann}) \overset{\mathrm{F}}{\vdash} \circledcirc$

From the construction of the reduction we know that the condition of the only obligation contained in $\circledcirc$ is constituted by φ. However from the hypothesis we know that φ is a tautology, hence according to Definition 13, such obligation is always fulfilled independently on the trace taken into consideration. Therefore following from Definition 16, it follows that if φ is a tautology, then the compliance problem constructed using the reduction results in full compliance.

□

Complexity:

The process P and the obligation $\langle O^a(\varphi), l_{test}, \bot \rangle$ can be constructed in time proportional to $|L| + |\varphi|$ where $|\varphi|$ denotes the length of formula φ. Since $|L| \leq |\varphi|$ by construction, the time is at most polynomial in the length of the formula φ.

5 Verifying Non Compliance is coNP-complete

In this section we prove that the problem of verifying whether a structured annotated process is not compliant with a regulatory framework is a co**NP**-complete problem. In the same way as we proved that verifying full compliance is co**NP**-complete, also in this case we have to prove that the complementary problem of verifying non compliance is in **NP**-complete.

5.1 *Not* non compliance is NP-complete

Once again we define the complement of the problem as its negation.

According to Definition 16, non compliance with respect to a set of obligations $\circledcirc$ is defined as follows:

Non Compliance:

$$(P, \mathsf{ann}) \nvdash \circledcirc \text{ iff } \neg\exists \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc.$$

Therefore, we define the complement of non compliance, *not non compliance*, as follows:

*Definition 22 (*Not *Non Compliance):* Given an annotated process (P, ann) and a set of obligations $\circledcirc$.

***Not* Non Compliance**:

$$\neg(P, \mathsf{ann}) \nvdash \circledcirc \text{ iff } \exists \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc.$$

We can see that the condition of non compliance is the same as that of partial compliance. Thus the **NP**-completeness of partial compliance implies the **NP**-completeness of *not* non compliance and hence also the co**NP**-completeness of non compliance.

6 Conclusion

In this paper we show that in general the problem of verifying business process regulatory compliance is hard. More specifically we prove that the problem of verifying whether a structured process is partially compliant with a set of obligations is **NP**-complete, and that the problem of verifying whether a structured process is fully compliant with a set of obligation is co**NP**-complete.

In the third section of the paper we prove that verifying partial compliance is **NP**-complete by showing the membership in **NP** and the **NP**-Hardness of the problem. The **NP**-Hardness is shown by reducing the hamiltonian path problem to a problem of verifying partial compliance.

In the fourth section of the paper we prove that verifying full compliance is co**NP**-complete by showing that verifying the complement is an **NP** problem and we prove the completeness by reducing the tautology problem to it. We prove the **NP** membership of the complementary problem by reusing some of the results obtained in Section 3.

In a similar way, in the fifth section we prove that verifying non compliance is also co**NP**-complete.

The results obtained are in line with the statement of Observation 1, where it is stated that full compliance implies partial.

Our results explain why existing solutions like Hoffmann et al. [4], Ghose and Koliadis [14], Governatori et al. [1], Awad et al. [5] and Ramezani et al. [7] do not provide efficient solutions, or in case they do, the solutions provided are an approximation of the real ones.

The approaches of Awad et al. [5], Ramezani et al. [7] and Elgammal et al. [15] use Linear Temporal Logic (LTL) to model processes and compliance requirements. In particular they define business process patterns corresponding to compliance requirements. The problem of determining whether a business process is compliant reduces to a model checking problem in the underlaying logic, LTL, which is known to be PSPACE-complete [16]. Accordingly, such systems adopt a more complex (and more expressive) formalism. Despite the more expressive language these frameworks are restricted to "structural" compliance, namely the compliance requirements are just about the presence of tasks in a process and relationships among tasks and not about the effects of the tasks. In addition there are some concerns that temporal logics, and in particular LTL, might not be able to model compliance requirements as shown by Governatori [17].

The approaches of Governatori and Sadiq [1], Ghose and Koliadis [14], and Hoffmann et al. [4] attach effects to the tasks in a process and propagate the effects while traversing a process model.

The approach proposed by Governatori and Sadiq [1] and further developed in [2] defines a sound and complete polynomial time algorithm (implemented in [18]) to check whether a trace of a process is compliant. For a process the algorithm has to be applied to all traces in a process, but the number of traces in a given process, in general, is exponential in terms of the number of tasks and gateways in the process.

Ghose and Koliadis [14] adopt the same update semantics presented in this paper. In case of XOR and AND splits a copy of the current state of the process before the split is created for each branch of the split and then propagated independently from the other branches. The states are merged using union in cases of AND joins. The number of copies generated is exponential in the number of the XOR gateways occurring in the process, and traces corresponding to interleaving tasks from different branches of AND blocks are not considered.

The approach by Hoffman et al. [4] offers a polynomial time approximate solution to the business process regulatory compliance problem. This approach is based on the technique of I-propagation. Similarly to Ghose and Koliadis [14] the annotations of the branches in AND and OR blocks are computed independently. The difference is that for XOR join they consider the intersection of the incoming set of literals (one set for each incoming branch). This permits an efficient computation, but it loses information related to the XOR splits. Also, as in Ghose and Koliadis [14] it does not consider the traces resulting from interleaving the tasks in parallel branches of AND blocks.

Finally PENELOPE [3] and SeaFlow [6] use, respectively, Event Calculus and first order logic to model compliance requirements. Given that Event Calculus is a specialised first order logic theory, in both cases compliance is reduced to an entailment problem in first order logic, which is known to be computationally intractable.

The goal of the present paper is to show the intractability of the business process regulatory compliance problem. We represent the possible ways of achieving a business objective using a business process model and we consider the regulatory framework to be composed by a set of obligations. Given these elements, we formally prove that verifying whether a business process is partially compliant with the regulatory framework is an **NP**-complete problem and the complexity of verifying whether a business process is either fully compliant, or not compliant with the regulatory framework is co**NP**-complete.

As future work we plan to extend the present work in two ways: first, we plan to study larger and more complete problems, like the ones including compensations for the obligations violated in the form of contrary to duties described by Prakken and Sergot[19] and by Jones and Carmo [20]. With this further analysis we want to see whether the complexity of the problem increases and to which extent. Second we plan to study the complexity of the sub-

problems of the problem tackled in the present paper. The results of this analysis will help the community by pointing out which sub-problems may be solved efficiently.

ACKNOWLEDGMENTS

- We thank Prof. Leendert van der Torre from University of Luxembourg for his helpful suggestions and comments.
- NICTA is funded by the Australian Government as represented by the Department of Communications and the Australian Research Council through the ICT Centre of Excellence program.
- Silvano Colombo Tosatto is supported by the National Research Fund, Luxembourg.

REFERENCES

[1] G. Governatori and S. Sadiq, "The journey to business process compliance," in *Handbook of Research on BPM*, J. Cardoso and W. van der Aalst, Eds. IGI Global, 2009, ch. 20, pp. 426–454.

[2] G. Governatori and A. Rotolo, "A conceptually rich model of business process compliance," in *APCCM 2010*, ser. CRPIT, S. Link and A. Ghose, Eds., vol. 110. Australian Computer Society, 2010, pp. 3–12.

[3] S. Goedertier and J. Vanthienen, "Designing compliant business processes with obligations and permissions," in *Business Process Management (BPM) Workshops*, 2006, pp. 5–14.

[4] J. Hoffmann, I. Weber, and G. Governatori, "On compliance checking for clausal constraints in annotated process models," *Information Systems Frontiers*, vol. 14, no. 2, pp. 155–177, 2012.

[5] A. Awad, M. Weidlich, and M. Weske, "Visually Specifying Compliance Rules and Explaining their Violations for Business Processes," *Journal of Visual Languages & Computing*, vol. 22, no. 1, pp. 30–55, 2011.

[6] L. T. Ly, S. Rinderle-Ma, K. Göser, and P. Dadam, "On enabling integrated process compliance with semantic constraints in process management systems," *Information Systems Frontiers*, vol. 14, no. 2, pp. 195–219, 2012.

[7] E. Ramezani, D. Fahland, and W. M. P. van der Aalst, "Where did I misbehave? Diagnostic information in compliance checking," in *BPM 2012*, ser. LNCS, A. P. Barros, A. Gal, and E. Kindler, Eds., vol. 7481. Springer, 2012, pp. 262–278.

[8] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler, "On structured workflow modelling," in *CAiSE 2000*, ser. LNCS, B. Wangler and L. Bergman, Eds., vol. 1789. London, UK: Springer-Verlag, 2000, pp. 431–445.

[9] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas, "Structuring acyclic process models," *Information Systems*, vol. 37, no. 6, pp. 518–538, 2012.

[10] G. Keller and T. Teufel, *SAP R/3 Process Oriented Implementation: Iterative Process Prototyping*. ddison-Wesley, 1998.

[11] G. Governatori, J. Hoffmann, S. W. Sadiq, and I. Weber, "Detecting regulatory compliance for business process models through semantic annotations," in *Business Process Management Workshops*, ser. LNBIP, D. Ardagna, M. Mecella, and J. Yang, Eds., vol. 17. Springer, 2008, pp. 5–17.

[12] C. E. Alchourrón, P. Gärdenfors, and D. Makinson, "On the logic of theory change: Partial meet contraction and revision functions," *Journal of Symbolic Logic*, vol. 50, no. 2, pp. 510–530, 1985.

[13] G. Governatori and A. Rotolo, "Norm compliance in business process modeling," in *Proceedings of the 4th International Web Rule Symposium: Research Based and Industry Focused (RuleML 2010)*, ser. LNCS, vol. 6403. Springer, 2010, pp. 194–209.

[14] A. Ghose and G. Koliadis, "Auditing business process compliance," in *ICSOC 2007*, ser. LNCS, B. J. Krämer, K.-J. Lin, and P. Narasimhan, Eds., vol. 4749. Springer, 2007, pp. 169–180.

[15] A. Elgammal, O. Turetken, W.J. van den Heuvel, and M. Papazoglou, "Formalising and applying compliance patterns for business process compliance," *Software and Systems Modeling*, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10270-014-0395-3

[16] A. P. Sistla and E. M. Clarke, "The complexity of propositional linear temporal logics," *Journal of ACM*, vol. 32, no. 3, pp. 733–749, Jul. 1985. [Online]. Available: http://doi.acm.org/10.1145/3828.3837

[17] G. Governatori, "Thou shalt is not you will," NICTA, Tech. Rep. 8026, 2014.

[18] G. Governatori and S. Shek, "Regorous: a business process compliance checker," in *ICAIL*, E. Francesconi and B. Verheij, Eds. ACM, 2013, pp. 245–246.

[19] H. Prakken and M. Sergot, "Dyadic deontic logic and contrary-to-duty obligations," in *Defeasible Deontic Logic*, D. Nute, Ed. Kluwer Academic Publishers, 1997, pp. 223–262.

[20] A. Jones and J. Carmo, "Deontic logic and contrary-to-duties," in *Handbook of Philosophical Logic*, D. Gabbay and F. Guenthner, Eds. Kluwer Academic Publishers, 2002, pp. 265–343.

Silvano Colombo Tosatto studied computer science at the Universitá di Torino where he got his master degree in 2010 defending his thesis entitled "Neural Symbolic Learning Systems: Neural Networks for Normative Reasoning". Silvano is currently a Ph.D. Student at the university of Luxembourg where he is mainly working on the complexity of proving regulatory compliance. Silvano's main research interest is artificial intelligence, in particular agent architectures, multi agent systems and normative reasoning.

Guido Governatori is a senior principal researcher in the Software Systems Research Group at NICTA where he leads the research activities on Business Process Compliance. He is also an adjoint professor in the BPM Group at Queensland University of Technology. He received his PhD in Legal Informatics from the University of Bologna. His research interests include defeasible reasoning, modal deontic and non-classical logics and their applications to normative reasoning, agent systems and business process modelling.

Pierre Kelsen is professor of computer science at the University of Luxembourg where he leads the Laboratory for Advanced Software Systems. He received a PhD in computer science at the University of Illinois at Urbana-Champaign in the area of parallel graph algorithms. He held postdoctoral positions at the University of British Columbia and the Max-Planck Institute for Informatics. His research interests include model-driven engineering, formal methods, and algorithms.