Proving Regulatory Compliance: Full Compliance Against an Expressive Unconditional Obligation is coNP-Complete

Silvano Colombo Tosatto
silvano.colombotosatto@data61.csiro.au

Guido Governatori
guido@governatori.net

Nick van Beest
nick.vanbeest@data61.csiro.au

1CSIRO Data61
41 Boggo Rd, Dutton Park QLD 4102, Brisbane, Australia

December 13, 2022

Abstract
Organisations are required to show that their procedures and processes satisfy the relevant regulatory requirements. The computational complexity of proving regulatory compliance is known to be generally hard. However, for some of its simpler variants the computational complexity is still unknown. We focus on the eight variants of the problem that can be identified by the following binary properties: whether the requirements consists of one or multiple obligations, whether the obligations are conditional or always in force, and whether only propositional literals or formulae can be used to describe the obligations. This paper in particular shows that proving full compliance of a model against a single unconditional obligation whose elements can be described using formulae is coNP-complete. Finally we show how this result allows to fully map the computational complexity of these variants for proving full and non compliance, while for partial compliance the complexity result of one of the variants is still missing.

1 Introduction

The problem of proving compliance of process models consists of verifying whether the possible executions of these models satisfy some given requirements. Organisations are generally required to show that their procedures, which can be represented as process models, satisfy the regulatory requirements. The number of regulations and rules that the organisation needs to comply with is generally large [8], and non-compliance may lead to significant fines and law suits [12].

Although the computational complexity of the general problem of proving regulatory compliance is known to be hard, the computational complexity of some of its variants is still unknown, and in this paper we provide the computational complexity for the following variant:
1. Proving whether all executions of a process model satisfy a single requirement holding for the whole duration of the execution, and allowing the requirement to be described using propositional formulae.

Considering the other variants of the problem that can be identified by the three following binary properties: 

a) the number of obligations describing the regulatory requirements,

b) whether these requirements are conditional, and

c) whether these requirements conditions are restricted to be represented as single propositions, or can be represented using formulae. We position the new computational complexity result among the existing ones and show that we can now determine the computational complexity of all the variants when it comes to evaluating full and non compliance.

The remainder of the paper is structured as follows: Section 2 introduces the formal concepts around business processes and business process models, together with the regulatory framework they are subject to. Section 3 provides the new computational complexity result, which is placed in context with existing results in Section 4. Finally, Section 5 concludes the paper.

2 Preliminaries

2.1 Acyclic Structured Process Models

We first introduce process models, which represent a collection of possible ways in which an organisation can achieve a given objective. Verifying these models against some regulations corresponds to show that the ways the organisation achieves these objectives follow the regulations.

We limit our study to problems dealing with acyclic structured process models, similar to the structured workflows defined by Kiepuszewski et al. [7], which can be preliminarily defined as the models whose components are properly nested. Additionally, we consider only acyclic process models, meaning that these models do not contain loops and every task is limited to be executed at most once in each of the model’s executions.

Business processes can be modelled using different notations, which mostly use an intuitive graphical representation. Petri nets are a well-known tool for modelling concurrent processes, for which a rich body of theory and tools to verify their properties have been defined. In the context of this work, we use a specific sub-class called workflow nets (WF nets), for which we define the formal framework below.

2.1.1 Petri Nets

A Petri net is a directed bipartite graph consisting of places, transitions, and arcs between transition-place pairs. The activities in a business process are represented as transitions in a Petri net, while the execution states are represented as combinations of places, which are token containers. A Petri net is formally defined as follows [9, 11]:

**Definition 1 (Petri net)** A Petri net $\mathcal{P}$ is a tuple $(\mathcal{P}, \mathcal{T}, \mathcal{A}, \lambda)$, where:

1Whether all executions of a model, or none respectively, satisfy the given requirements.
- $P$ is a finite set of places,
- $T$ is a finite set of transitions, such that $P \cap T = \emptyset$,
- $A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, and
- $\lambda : P \cup T \rightarrow \mathcal{L}$ is a labelling function.

Each place of a Petri net can contain a token, and the net marking (Definition 3) of a Petri net is determined by which of its places contain tokens and which does not. A transition in a Petri net represents an operation towards the goal achieved by executing the Petri net. For instance, in a Petri net representing a cooking recipe, its transitions can correspond to the preparation steps required to prepare the recipe.

Depending on the net marking of a Petri net, some of its transitions can be executed, and their execution manipulate the tokens in the Petri net’s places, altering the net marking. These correspond to the pre-set and post-set of a transition as described in Definition 2.

Definition 2 (Pre/Post-sets) Given a Petri net $P = (P, T, A, \lambda)$, for each node $y \in P \cup T$:

- The pre-set of $y$, denoted by $\bullet y$, is the set of all input nodes of $y$: $\bullet y = \{x \in P \cup T \mid (x, y) \in A\}$.
- The post-set of $y$, denoted by $y \bullet$, is the set of all output nodes of $y$: $y \bullet = \{x \in P \cup T \mid (y, x) \in A\}$.

Definition 3 (Petri net state, net marking) A net marking $\mu$ of a Petri net $P = (P, T, A, \lambda)$ is a function that associates places with numbers, i.e., $\mu : P \rightarrow \mathbb{N}_0$.

Definition 4 (Marked net) A marked net is a tuple $P = (P, T, A, \lambda, \mu_0)$, where $(P, T, A, \lambda)$ is a Petri net and $\mu_0$ is the initial marking.

Definition 5 (Enabled transition) Given a net marking $\mu$, transition $t \in T$ is said to be enabled if $\forall p \in \bullet t : \mu(p) > 0$.

Definition 6 (Firing rule) Given a marked net $P = (P, T, A, \lambda, \mu_0)$ and an enabled transition $t$, the firing of $t$ is denoted by $\mu \xrightarrow{t} \mu'$ and leads to a new marking $\mu'$ such that:

- $\forall p \in \bullet t, \mu'(p) = \mu(p) + 1$,
- $\forall p \in t \bullet, \mu'(p) = \mu(p) - 1$.

Petri nets can be restricted with some extra properties, so that it resembles a business process having a single source place and a single sink place (i.e. a fixed start and end of the process). These special Petri nets are referred to as workflow nets (WF-nets for short) and can be formally defined as follows:

Definition 7 (WF-net) A marked net $P = (P, T, A, \lambda, \mu_0)$ is a WF-net iff:
• \( P \) has a special source place \( i \), such that \( \bullet i = \emptyset \) and \( \forall p \in P \setminus i, \bullet p \neq \emptyset \).

• \( P \) has a special sink place \( o \), such that \( oo = \emptyset \) and \( \forall p \in P \setminus o, p\bullet \neq \emptyset \).

• if we add a transition \( t_w \) to \( P \) such that \( \bullet t_w = o \) and \( t_w\bullet = i \), then \( P \) is strongly connected.

An execution of a WF-net is any sequence of transitions where the net marking resulting from a transition in the sequence is in the pre-set of the next transition in the sequence.

**Definition 8 (Execution)** Given a WF-net \( P = (P, T, A, \lambda, \mu_0) \), an execution \( \epsilon \) of \( P \) is a sequence of transitions \( t_0, \ldots, t_n \) such that:

• \( \forall t_i \in \epsilon, t_i \in T \),

• \( \bullet t_0 = \{i\} \),

• \( t_n\bullet = \{o\} \),

• \( \mu_0 \stackrel{t_0}{\rightarrow} \mu_1 \stackrel{t_1}{\rightarrow} \cdots \stackrel{t_n}{\rightarrow} \mu_n \), with no enabled transitions in \( \mu_n \).

### 2.1.2 Structuredness

Structured WF-nets can be characterised by their ability to be decomposed into so-called single-entry-single-exit (SESE) regions \([10]\), which we will refer to as process components throughout the remainder of this paper. A process component is a subset of nodes of a WF-net such that the subgraph induced by these nodes has a single entry node and a single exit node. These two nodes are called boundary nodes as they connect the component with the rest of the model.

**Definition 9 (Boundary nodes)** Given a process component \( r \), the single entry node and single exit node of \( r \) are referred to as boundary nodes. We use the shorthand \( \triangleright r \) to denote the entry node of \( r \) and \( r\triangleright \) to denote the exit node of \( r \).

**Definition 10 (Canonical process component)** A process component is canonical, if it does not share any of its nodes with another process component. That is, any two canonical process components are either disjoint or one is contained in the other.

We adapt the definitions provided in \([10]\), to further define the types of process components specific to WF-nets:

**Definition 11 (Trivial, Polygon, Bond, Rigid)** Let \( C \) be a process component of a WF-net \( P \), with boundary nodes \( \triangleright C, C\triangleright \in P \cup T \).

• \( C \) is a trivial component, iff \( C \) is singleton, i.e., \( C \) contains a single node such that \( \triangleright C = C\triangleright \).
• C is a polygon component, iff there exists a sequence \((r_0, \ldots, r_n), n \in \mathbb{N}\), of canonical components of \(P\), such that:
  - \(C = \triangleright C \cup C\triangleright \bigcup_{i=0}^{n} r_i\)
  - \((\triangleright C, \triangleright r_0), (r_n, C\triangleright) \in A\)
  - \((r_j, r_{j+1}) \in A, 0 \leq j < n\)

• C is a bond component, iff there exists a set \(R\) of canonical components of \(P\), such that:
  - \(C = \triangleright C \cup C\triangleright \bigcup_{r \in R} r\)
  - \(\forall r \in R : (\triangleright C, \triangleright r), (r, C\triangleright) \in A\)
  - \(x, y \in X \lor x, y \in T\)

• C is a rigid component, iff C is neither a trivial, nor a polygon, nor a bond component.

**Definition 12 (Well-structured WF-net)** A WF-net \(P\) is (well-)structured, iff its set of all its canonical process components contains no rigid process component; otherwise the process model is unstructured.

### 2.1.3 States and Traces

Transitions in a WF-net can be annotated with the set of propositions that hold or not after its execution. As such, each transition can be annotated, resulting in a so-called annotated WF-net.

**Definition 13 (Annotation)** Given a WF-net \(P = (P, T, A, \lambda, \mu_0)\), the annotation function \(\alpha : T \rightarrow \mathcal{L}^2\) where \(\mathcal{L}\) is the propositional literals language, representing the set of propositions holding true after the execution of a transition.

**Definition 14 (Annotated WF-net)** Given a WF-net \(P = (P, T, A, \lambda, \mu_0)\) and an annotation function \(\alpha\). We refer to an annotated WF-Net as \(P = (P, T, A, \lambda, \mu_0, \alpha)\).

An annotation function describes what holds true after a transition is executed. After the execution of a sequence of transitions, we can compose the collection of annotations that hold true up to that point by performing a state update after execution of each transition, as defined formally in Definition 15 below.

**Definition 15 (State update)** Given two consistent sets of literals \(L_1\) and \(L_2\), representing the process state and the annotation of a transition being executed, the update of \(L_1\) with \(L_2\), denoted by \(L_1 \oplus L_2\) is a set of literals defined as follows:

\[
L_1 \oplus L_2 = L_1 \setminus \{\neg l \mid l \in L_2\} \cup L_2
\]

We use the term *trace* to refer to an execution of a WF-Net along with the associated set of literals that hold after the execution of each transition.
Definition 16 (Trace) Given an annotated WF-Net $P = (P, T, A, \lambda, \mu_0)$ and an execution $\epsilon = (t_1, \ldots, t_n)$ of $P$, the corresponding trace is $\theta = (L_1, \ldots, L_n)$, where each $L_i$ is a set of propositions and is computed as follows:

1. $L_0 = \emptyset$
2. $L_{i+1} = L_i \oplus \text{ann}(t_{i+1})$, for $1 \leq i < n$.

Each $L_i$ in a trace represents the execution’s state of the annotated WF-Net holding after executing the transition $t_i$ in the corresponding execution.

In Example 1, we illustrate an annotated WF-Net along with its possible executions and traces. In the remainder of this paper, we refer to annotated WF-nets simply as process models.

Example 1 (Annotated Process Model) Fig. 1 shows a structured process containing four tasks labelled $t_1, t_2, t_3$ and $t_4$ and their annotations. The dashed lines mark the edges of the different process components, indicating polygon and bond components by $P$ and $B$, respectively. There are two bond components, of which the outer is an AND block with two concurrent paths. The top path is another bond component representing an XOR block, whereas the bottom path is a polygon representing a single task. 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 that state of the process. Table 1 shows the possible executions ($\Sigma(B)$) and traces ($\Theta(B, \text{ann})$) of the model.

2.2 Regulatory Framework

The regulatory framework describes the requirements that a process model must follow to be considered compliant. Depending on whether all, some, or none of the the
executions satisfy these requirements, a different compliance level is determined for the process model. The regulatory requirements in a regulatory framework are described using obligations. These obligations are specified using a fragment of Process Compliance Logic (PCL), introduced by Governatori and Rotolo [6].

2.2.1 Semantics

Obligations are evaluated over traces of a process model. We refer to the sub-segments of a trace where an obligation is evaluated as in force intervals, and depending on the obligation’s type these in force intervals are evaluated as follows: achievement obligations require that a state within each in force interval to satisfy the obligation’s requirement, and maintenance obligations require that each state in each interval satisfies the requirement.

Definition 17 (Obligations) An obligation is a tuple $O_o \langle \pi, \tau, \delta \rangle$, where $o$ is its type, and $\pi, \tau, \delta$ are propositional formulae representing respectively the obligation’s requirement, trigger and deadline.

Given a trace $\theta = (L_1, \ldots, L_n)$, $O_o(\pi, \tau, \delta)$ is either satisfied or violated as follows, depending on its type:

achievement :

satisfied $\forall L_i \in \theta | L_i \models \tau :$

- $\exists L_j \in \theta | L_j \models \pi$ and $i \leq j$, and
- $\forall L_k \in \theta | L_k \models \delta$ and $k < j$.

violated otherwise.

maintenance :

satisfied $\forall L_i \in \theta | L_i \models \tau :$

- $\exists L_j \in \theta | L_j \models \delta$ and $i \leq j$, and
- $\forall k | i \leq k \leq j : L_k \models \pi$.

violated otherwise.

Figure 2 illustrates an in force interval of a local obligation within a trace. This simplistic representation shows that an in force interval is identified in a trace between two points: the trigger and the deadline.

From Definition 17 it follows that states satisfying the deadline of an obligation are critical to determine whether the obligation is satisfied or violated. To avoid a scenario
where an obligation being evaluated reaches the end of a trace without encountering a state satisfying the deadline (thus potentially leaving the obligation in an unknown state), we adopt Assumption 1.

**Assumption 1 (Final Deadline)** Given a trace $\theta = (L_1, \ldots, L_n)$ and an obligation $O^\omega(\pi, \tau, \delta)$, we assume that $L_n \models \delta$ independently from the contents of $L_n$.

### 2.2.2 Discussion on Multiple in Force Intervals

A trace may contain multiple states satisfying the trigger and the deadline of an obligation, such as for instance shown in Figure 3. In this example, two states satisfy the trigger, labelled `trigger 1` and `trigger 2`, and two states satisfying the deadline, labelled `deadline 1` and `deadline 2`. From Definition 17 it follows that each of such intervals needs to be satisfied for the trace to satisfy the obligation.

When two in force instances intersect, the only possible intersection is shown in Figure 4. When this is the case, Lemma 1 states when two intervals intersect then both can be evaluated by evaluating a single interval.

**Figure 2: Conditional Obligation**

**Figure 3: Multiple In Force Instances**

**Figure 4: Successful Termination of 2 Overlapping In Force Instances**
Lemma 1 (Overlapping In Force Intervals) Consider a trace $\theta$ and a local obligation $O^o(\pi, \tau, \delta)$ having two in force intervals $i_1, i_2$ in $\theta$. If $i_1$ and $i_2$ overlap\(^2\) where $i_2 \subset i_1$ (as shown in Figure 4), then:

- **o = achievement** evaluating $i_2$ provides the evaluation for both $i_1$ and $i_2$, and
- **o = maintenance** evaluating $i_1$ provides the evaluation for both $i_1$ and $i_2$.

Proof 1 We prove the lemma by checking the two cases depending on the type of the obligation:

**Achievement** From Definition 17 we know that if $i_2$ is satisfied, then the state satisfying $i_2$ would also satisfy $i_1$, and if $i_2$ is not satisfied, then the evaluation of $i_1$ becomes irrelevant as the trace would not satisfy the obligation producing these two in force intervals.

**Maintenance** From Definition 17 we know that a required property needs to be maintained within an in force interval. When such property needs to be maintained for the duration of two in force intervals ($i_1$ and $i_2$), and these intervals are overlapping, this is equivalent to maintain the property for the first interval, as the deadline of the first interval would also terminate the second interval.

2.3 Compliance Classifications

A process checked for compliance can be classified as follows: **fully compliant** if every trace of the process is compliant with the obligations composing the regulatory framework, **partially compliant** if there exists at least one compliant trace, and **not compliant** if none of the traces comply with the obligations.

**Definition 18 (Process Compliance)** Given a process $(P, \text{ann})$ and a regulatory framework composed by a set of obligations $O$, the compliance of $(P, \text{ann})$ with respect to $O$ is determined as follows:

- **Full compliance** $(P, \text{ann}) \vdash^F O$ if and only if $\forall \theta \in \Theta(P, \text{ann}), \theta \vdash O$.
- **Partial compliance** $(P, \text{ann}) \vdash^P O$ if and only if $\exists \theta \in \Theta(P, \text{ann}) | \theta \vdash O$.
- **Non compliance** $(P, \text{ann}) \not\vdash O$ if and only if $\neg \exists \theta \in \Theta(P, \text{ann}) | \theta \vdash O$.

2.4 Problem Variants

Given the components constituting a compliance problem introduced in the previous sections, we introduce three binary properties of the regulatory framework that allow to distinguish eight different variants of the compliance problem.

\(^2\)Two in force intervals are considered to be overlapping if one of the intervals has its trigger point or deadline point in the other interval.
1/n A regulatory framework containing a single obligation (1) or a set of obligations (n).

G/L A regulatory framework limited to contain global obligations. Global obligations are a special case where their in-force instance is set by default to the full trace duration, and we represent them by omitting the trigger and deadline as follows: $O^o(\pi, \bot)$, or conditional obligations are allowed (L).

-/+ A regulatory framework containing obligations limited to be described using propositional literals (-) or allowing such obligations to be described using propositional formulae (+).

Each of the binary property allows an easy and a hard option:

- It is easier to check compliance against a single obligation (1) than multiple ones (n).
- A global obligation (G) allows a single in force instance spanning for the entirety of the traces, making it the easier counterpart, as conditional obligations (L) may allow multiple in force instances, which also need to be computed in each possible trace of the process model.
- Obligations limiting their elements to propositions (-) instead of formulae (+) are easier to verify, as the satisfaction of their elements can be reduced to checking the execution of tasks having such elements in their annotations. Checking the satisfaction of the elements of an obligation allowing formulae always requires to consider the trace’s history, making it harder.

Considering two variants of the problem sharing two of the three binary properties, it is fair to assume that the one being defined by the harder differentiating property, is at least as computationally complex as the one defined by the easier differentiating property. For instance, the variant $nG+$ is at least as difficult as both $nG-$ or $1G+$. This is illustrated graphically in Figure 5 below, where the potential increase in complexity is indicated by the direction of the arrows.

![Figure 5: Compliance Complexity Lattice.](image-url)
3 Full Compliance: 1G+ is coNP-Complete

The tautology problem consists of proving whether a given propositional logic formula \( \varphi \) is true in all of its possible interpretations, and its computational complexity is known to be coNP-complete. In this section, we show that proving full compliance of the variant 1G+ is coNP-complete, by reducing the tautology problem to 1G+. As such, we require a WF-net representing a set of traces covering each interpretation of \( \varphi \):

Definition 19 (Interpretation WF-net construction) Given a propositional logic formula \( \varphi \), a WF-net \( \mathcal{P} \), containing a set of traces whose aggregated collection of process states represent each possible interpretation of \( \varphi \), is constructed as follows:

1. For each proposition appearing in \( \varphi \) a bond component representing an XOR block is created containing the following:
   - A trivial component with a transition having the proposition annotated.
   - A trivial component with a transition having the negation of the proposition annotated.

2. Construct a polygon component representing a SEQ block starting with the start and concluded by the end, having between these places the following components included:
   - A trivial component composed by a transition labeled \( \text{init} \) and annotated with the set of proposition containing all the positive variants of the propositions appearing in \( \varphi \), followed by:
   - All XOR blocks constructed in 1. The order in which these XOR blocks are included is not relevant.

Given a propositional formula \( \varphi \), proving whether \( \varphi \) is a tautology can be reduced to a 1G+ problem as follows:

Reduction 1

WF-Net The model is constructed as an Interpretation Process as described in Definition 19. This construction allows to represent each possible interpretation of \( \varphi \) in the traces of the model. A generic result of the construction process detailed above is shown in Figure A. When one of the XOR blocks constructed in step 1 is executed, one of the transitions composing it is executed and either the positive or negative variant of a proposition is introduced in the process state. When all of the XOR blocks are executed, the resulting process state contains a positive or negative variant of each proposition appearing in \( \varphi \), corresponding to one of the possible interpretations of the formula, hence the order in which the XOR are included in the SEQ is not relevant. The inclusion of the init transition allows to always have in the execution’s state of the WF-Net a complete interpretation for the variables of \( \varphi \).
The regulatory framework is composed by the following global maintenance obligation: \( O^\omega (\varphi, \omega) \). Note that at any step of the execution, the values of the variables composing the process state always represents a possible interpretation for the formula \( \varphi \), hence by representing the obligation as a maintenance we ask that every possible interpretation makes the formula true.

![Figure 6: Interpretation WF-Net Construction.](image)

**Proof 2**

1. From the construction of the WF-net in Definition 19, it follows that each possible trace contains in its last state a possible interpretation for the proposition composing \( \varphi \).

2. From the construction of the WF-net in Definition 19, it follows that every possible interpretation of the literals included in \( \varphi \) is considered in at least a state of a trace of the business process model.

3. Given the initial annotation of the task `start`, from Definition 15 and the construction of the WF-net in Definition 19 it follows that the states evolving through the trace of the model represent the interpretations of the propositions composing \( \varphi \) that are still possible.

4. From 1., 2., 3., and the construction of the obligation, it follows that if none of the states in every trace of the business process model falsifies the condition of the maintenance obligation constituting the regulatory framework, this means that \( \varphi \) satisfies each of its possible interpretations. Therefore, \( \varphi \) is a tautology.

## 4 Computational Complexity Overview

In this section, we show how the new complexity result integrates with existing results and how they allow to map the computational complexity of each variant of the problem for proving full, and non compliance.

### 4.1 Existing Complexity Results for Full Compliance

For the problem of proving full compliance, the computational complexity results already exist for five out of eight variants of the problem, as shown in Table 2.
### Table 2: Full Compliance Complexity

| Variant | Source | Complexity Class |
|---------|--------|-----------------|
| 1G-     | Colombo Tosatto et al. [5] | P               |
| 1L-     | Colombo Tosatto et al. [3]  | P               |
| nL-     | Colombo Tosatto et al. [3]  | P               |
| 1L+     | Colombo Tosatto et al. [4]  | coNP-complete   |
| nL+     | Colombo Tosatto et al. [1]  | coNP-complete   |

#### 4.1.1 Complexity Overview

The computational complexity results of each variant for proving full compliance are summarised in Figure 7.

Note that Figure 7 also contains results for the variants nG- and nG+, which were not explicitly provided. These results are derived using the relations between the variants of the problem shown earlier in Figure 5. The following relationship holds regarding the computational complexity: 1G- ≤ nG- ≤ nL-. Therefore, by knowing that both 1G- and nL- are in P, it follows that nG- is in P as well. Similarly, we know that with respect to computational complexity it holds that 1G+ ≤ nG+ ≤ nL+. As the extremes belong to the same computational complexity class, then also nG+ is in coNP-complete (shortened to coNP-c in Figure 7).

When proving full compliance, the computational complexity results show that variants limiting the expressivity of the elements of the obligations to atomic propositions can be solved in polynomial time, while when this limitation is absent, the complexity of solving the problem is in NP.

#### 4.2 Relations with Proving Non Compliance

Having provided the computational complexity results for the variants of the problem of proving both full and partial compliance of process models, we can extrapolate the computational complexities of proving non compliance of the same variants. Proving
non compliance of a process model is the complement of proving its partial compliance, as it requires to verify that none of the traces comply with the given regulations. Figure 8 illustrates this graphically by showing that the set of partially compliant processes are complementary to the non compliant ones. Moreover the illustration also shows that fully compliant process models are also partially compliant, as they always contain at least a compliant trace

![Figure 8: Compliance relations.](image)

### 5 Summary

This paper introduces a new complexity for one of the variants of the problem of proving regulatory compliance of models. When considered together with the existing computational complexity results, we can completely map the computational complexity for all these variants when the problem is to verify either full or non compliance of a model.

When the problem is to evaluate whether is partially compliant with the regulatory requirements, computational complexity results exist for 7 of the 8 variants, as reported in Table 3 and the computational complexity for the variant (1L-) is still not known, as summarised in Figure 9.

| Variant | Source                      | Complexity Class       |
|---------|-----------------------------|------------------------|
| 1G-     | Colombo Tosatto et al. [5]  | P                      |
| nG-     | Colombo Tosatto et al. [2]  | NP-complete            |
| 1G+     | Colombo Tosatto et al. [2]  | NP-complete            |
| nG+     | Colombo Tosatto et al. [2]  | NP-complete            |
| nL-     | Colombo Tosatto et al. [4]  | NP-complete            |
| 1L+     | Colombo Tosatto et al. [2]  | NP-complete            |
| nL+     | Colombo Tosatto et al. [4]  | NP-complete            |

Table 3: Partial Compliance Complexity

---

*Considering how we define process models, it cannot be the case that a model contains no traces, hence for this particular problem we can say that a for all implies an exists, which is generally not always the case in classical logic.*
Figure 9: Partial Compliance Complexity Lattice.

References

[1] Colombo Tosatto, S.: Proving Regulatory Compliance: Business Processes, Logic, Complexity. Ph.D. thesis, University of Luxembourg and Università di Torino (1 2015)

[2] Colombo Tosatto, S., Governatori, G., van Beest, N.: Checking regulatory compliance: Will we live to see it? In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) Business Process Management. pp. 119–138. Springer International Publishing, Cham (2019)

[3] Colombo Tosatto, S., Governatori, G., van Beest, N.R.: Business process full compliance with respect to a set of conditional obligation in polynomial time. https://arxiv.org/abs/2001.10148 (1 2020)

[4] Colombo Tosatto, S., Governatori, G., Kelsen, P.: Business process regulatory compliance is hard. IEEE Transactions on Services Computing 8(6), 958–970 (2015). https://doi.org/10.1109/TSC.2014.2341236, https://doi.org/10.1109/TSC.2014.2341236

[5] Colombo Tosatto, S., Kelsen, P., Ma, Q., Kharbili, M.E., Governatori, G., van der Torre, L.W.N.: Algorithms for tractable compliance problems. Frontiers of Computer Science 9(1), 55–74 (2015). https://doi.org/10.1007/s11704-014-3239-y, https://doi.org/10.1007/s11704-014-3239-y

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

[7] Kiepuszewski, B., Hofstede, A.H.M.t., Bussler, C.: On structured workflow modelling. In: Proceedings of the 12th International Conference on Advanced Information Systems Engineering. pp. 431–445. CAiSE ’00, Springer-Verlag, London, UK, UK (2000), http://dl.acm.org/citation.cfm?id=646088.679917

15
[8] Morgan, T.: Business rules and information systems: aligning IT with business goals. Addison-Wesley Professional (2002)

[9] Petri, C.: Communication with automata. Ph.D. thesis, Universität Hamburg (1966)

[10] Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Information Systems 37(6), 518–538 (2012)

[11] Reisig, W., Rozenberg, G.: Lectures on petri nets i: basic models: advances in petri nets. Springer Science & Business Media (1998)

[12] Zhang, I.X.: Economic consequences of the sarbanes–oxley act of 2002. Journal of accounting and economics 44(1-2), 74–115 (2007)