# Computational Complexity of Compliance and Conformance: Drawing a Line Between Theory and Practice

Silvano Colombo Tosatto
*Data61, CSIRO*
silvano.colombotosatto@data61.csiro.au

Guido Governatori
*Data61, CSIRO*
guido.governatori@data61.csiro.au

**Abstract**

In the present chapter we focus our attention on the computational complexity of proving regulatory compliance of business process models. While the topic has never received the deserved attention, we argue that the theoretical results, both existing and yet to find, are far reaching for many areas related to the problem of proving compliance of process models. Therefore, we provide here and discuss the existing results concerning the theoretical computational complexity of the problem, as well as discussing some further areas that can potentially advance the knowledge about the issue, and other closely related disciplines that can either bring or take insights to this area.

## 1 Introduction

In this chapter we investigate the computational complexity of the problem of proving regulatory compliance of process models. This problem consists of verifying whether a process model, representing a set of executions of an organisation's procedures, complies with some given regulations. We consider an execution to be compliant with some regulations when no violations occurs in such execution with respect of the regulations. Different degrees of compliance are determined depending on whether every execution in a model comply with the regulations, when some comply, and when none comply.

Proving regulatory compliance has been receiving more attention during the past years, as shown by approaches of varying complexity being proposed in the literature (Some of the recently proposed approaches: [41, 34, 35]). Analysis of the current expenses [57] from real businesses and companies towards showing their compliance with the relevant regulations, have brought an interests in finding automated solutions in order to bring down such costs. For a recent survey of the approaches to business process compliance and open research question in the field see Hashmi et al. [37].

Despite the various approaches proposed to address the problem of proving the regulatory compliance of business process models, meaning "ensuring that, executing such a business process model to achieve a business objective, is compliant with the regulations", or dealing with conformance: "verifying whether existing executions, usually logs, have been performed in accordance to the regulatory requirements", in general, the computational complexity of the problem it has been for the most part ignored. Despite knowing that in general the problem is NP-complete [10], many approaches have shown to be able to solve current real problem without being hindered by the theoretical complexity of it. While this allows currently to use this kind of solutions without any sort of big issue, due to the current race towards automation, it is only bound that the business process models and the regulatory frameworks required to be verified in the future are increasing in size and complexity. This, in turn could potentially put a hard stop to the approaches currently used, due to them ignoring such theoretical complexity concerns.

In this chapter we first provide a computational complexity analysis of the general problem of proving regulatory compliance of business process models, and its variants obtainable by manipulating the properties of the regulatory framework being used. We consider three different properties that a regulatory framework can have: the number of regulations contained in the framework, whether the regulations affect the entirety of the executions of a business process model, or whether some parts of them given some additional conditions. The third property concerns whether the regulations are expressed using atomic boolean propositions, or full formulae. Given these binary properties we identify 8 variants of the problems, for which we study and provide their computational complexity classes.

Additionally, the computational complexity of the problem can change depending on the features of the business process models being verified. Taking as the basic variant in this scenario structured business process models, namely process models whose structure can be defined as a properly nested structure, which has technical advantages over processes not following such constraints, which in turn ends up being more expressive. We consider some additional features that can be desirable to represent real life processes, such as unstructured process models and the inclusion

of loops, and we discuss how these additions influence the computational complexity of the problem's variants.

After having discussed the theoretical computational complexity of the problem, we consider some of the existing approaches aimed at solving the problem of proving regulatory compliance of business process models, and we assign them to the problem variants identified in this chapter, hence associating them to a computational complexity class. Starting form this classification, we provide a preliminary study of the behaviour of these approaches in a future where the components of the problem increase in size and complexity, namely the business process models and / or the regulatory frameworks. We aim with this preliminary analysis to provide some insights concerning which approaches may be hindered by the theoretical complexity limitations of the problem as bigger and more complex problems will be required to be solved, and which may be potentially be still used to tackle these larger problems.

Furthermore, we discuss a problem related to the problem of proving regulatory compliance: conformance, discussing a few of the techniques used to solve this tangential problem.

Finally, we conclude this chapter by discussing some of the open problem concerning the computational complexity analysis of the problem of proving regulatory compliance of business process models.

## 2 The Problem: Proving Regulatory Compliance

In this section, we introduce the problem of proving regulatory compliance of business process models analysed in this chapter. The problem consists of two components:

**i)** the business process model compactly describing a set of possible executions, and

**ii)** a regulatory framework, describing the compliance requirements.

### 2.1 Structured Business Processes

Generally, process models can be seen as a compact way to represent the set of possible ways that a company have to achieve some given business objectives. These models contain the tasks, which correspond to the atomic activities that can be executed to bring forward the achievement of the business objective pursued by the executions included in the model. These tasks are organised within the process model and describe a set of possible orders in which they can be executed. Example 1 illustrates an instance of a process that can be possibly used to describe the sale procedures in a shop.

**Example 1** (Shop Sale Process)**.** *Considering the scenario of a shop selling goods to costumers, the sale procedure can be summarised as a process by considering the sequence of steps listed below:*

1. *The customer chooses the goods he/she wants to purchase.*
2. *The total cost of the goods is tallied up.*
3. *The customer pays the calculated amount.*
4. *The sale is concluded.*

Using such formal models to represent their business procedures, companies allow to ensure that such procedures follow the required regulations by checking these models.

In this paper we focus our analysis on structured process models, such type of processes is similar to structured workflows defined by [44]. The advantage of focusing our initial analysis on these kind of processes is that their soundness[1] can be verified in polynomial time with respect to their size, and that the amount of possible executions belonging to the process model is finite, as it does not contain *loops* that can be potentially iterated any number of times, leading to business process models containing an infinite amount of possible executions.

Despite their simplicity, such kind of business process models can be used to represent 406 out of 604 processes in the SAP reference model [42], as shown by [55], illustrating also that unstructured processes, under certain conditions, can be translated into structured process models.

**Definition 1** (Process Block)**.** *A process block $B$ is a directed graph: the nodes are called* elements *and the directed edges are called* arcs*. The set of elements of a process block are identified by the function $V(B)$ and the set of arcs by the function $E(B)$. The set of elements is composed of tasks and coordinators. There are 4 types of coordinators:* 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 arc from other elements in $B$ and is denoted by $b(B)$. Similarly the final element has no outgoing arcs 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.*

[1] A process is sound, as defined by van der Aalst [67, 68], if it avoids livelocks and deadlocks.

- *Let* $B_1, \ldots, B_n$ *be distinct process blocks with* $n > 1$*:*
  - $\mathsf{SEQ}(B_1, \ldots, B_n)$ *denotes the process block with node set* $\bigcup_{i=0}^{n} V(B_i)$ *and edge set* $\bigcup_{i=0}^{n}(E(B_i) \cup \{(f(B_i), b(B_{i+1})) : 1 \leq i < n\})$.
    *The initial element of* $\mathsf{SEQ}(B_1, \ldots, B_n)$ *is* $b(B_1)$ *and its final element is* $f(B_n)$.
  - $\mathsf{XOR}(B_1, \ldots, B_n)$ *denotes the block with vertex set* $\bigcup_{i=0}^{n} V(B_i) \cup \{\mathsf{xsplit}, \mathsf{xjoin}\}$ *and edge set* $\bigcup_{i=0}^{n}(E(B_i) \cup \{(\mathsf{xsplit}, b(B_i)), (f(B_i), \mathsf{xjoin}) : 1 \leq i \leq n\})$ *where* xsplit *and* xjoin *respectively denote an* xor_split *coordinator and an* xor_join *coordinator, respectively. The initial element of* $\mathsf{XOR}(B_1, \ldots, B_n)$ *is* xsplit *and its final element is* xjoin.
  - $\mathsf{AND}(B_1, \ldots, B_n)$ *denotes the block with vertex set* $\bigcup_{i=0}^{n} V(B_i) \cup \{\mathsf{asplit}, \mathsf{ajoin}\}$ *and edge set* $\bigcup_{i=0}^{n}(E(B_i) \cup \{(\mathsf{asplit}, b(B_i)), (f(B_i), \mathsf{ajoin}) : 1 \leq i \leq n\})$ *where* asplit *and* ajoin *denote an* and_split *and an* and_join *coordinator, respectively. The initial element of* $\mathsf{AND}(B_1, \ldots, B_n)$ *is* asplit *and its final element is* ajoin.

By enclosing a process block as defined in Definition 1 along with a start and end task in a sequence block, we obtain a *structured process model*. Therefore, a structured process model can be understood as a structure recursively composed by process blocks, where at the lowest recursion level are the process blocks representing the tasks of the process model.

The effects of executing the tasks contained in a business process model are described using annotations as shown in Definition 2.

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

The status of the process execution is represented by a process' state. Such state contains a set of literals representing what is considered to be the case at that step of the execution. The literals contained in the process' state is determined by the sequence of the task being executed, and it is updated after each task execution.

The update between the states of a process during its execution is inspired by the AGM[2] belief revision operator [2] and is used in the context of business processes to define the transitions between states [23, 39], which in turn are used to define the *traces*.

[2]The approach is named after the initials of the authors introducing it: Alchourrón, Gärdenfors, and Makinson.

**Definition 3** (State update)**.** *Given two consistent sets of literals $L_1$ and $L_2$, representing the process state and the annotation of a task 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:*

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

**Definition 4** (Executions and Traces)**.** *Given a structured process model identified by a process block $B$, the set of its executions, written $\Sigma(B) = \{\epsilon | \epsilon$ is a sequence and is an execution of $B\}$. The executions contained in $\Sigma(B)$ are recursively constructed as follows:*

1. *If $B$ is a task $t$, then $\Sigma(B) = \{(t)\}$*
2. *if $B$ is a composite block with sub-blocks $B_1, \ldots, B_n$:*
   (a) *If $B = \mathsf{SEQ}(B_1, \ldots, B_n)$, then $\Sigma(B) = \{\epsilon_1 +_{\mathcal{E}} \cdots +_{\mathcal{E}} \epsilon_n | \epsilon_i \in \Sigma(B_i)\}$ and $+_{\mathcal{E}}$ the operator concatenating two executions.*
   (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 $\Sigma(B) = \{$the union of the interleavings of: $\epsilon_1, \ldots, \epsilon_n | \epsilon_i \in \Sigma(B_i)\}$*

*Given an annotated process $(B, \mathsf{ann})$ and an execution $\epsilon = (t_1, \ldots, t_n)$ such that $\epsilon \in \Sigma(B)$, a trace $\theta$ is a finite sequence of states: $(\sigma_1, \ldots, \sigma_n)$. Each state $\sigma_i \in \theta$ is a pair: $(t_i, L_i)$ capturing what holds after the execution of a task $t_i$, expressed by a set of literals $L_i$. A set $L_i$ is constructed as follows:*

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

*To denote the set of possible traces resulting from a process model $(B, \mathsf{ann})$, we use $\Theta(B, \mathsf{ann})$.*

**Example 2.** *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 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 that state of the process.*

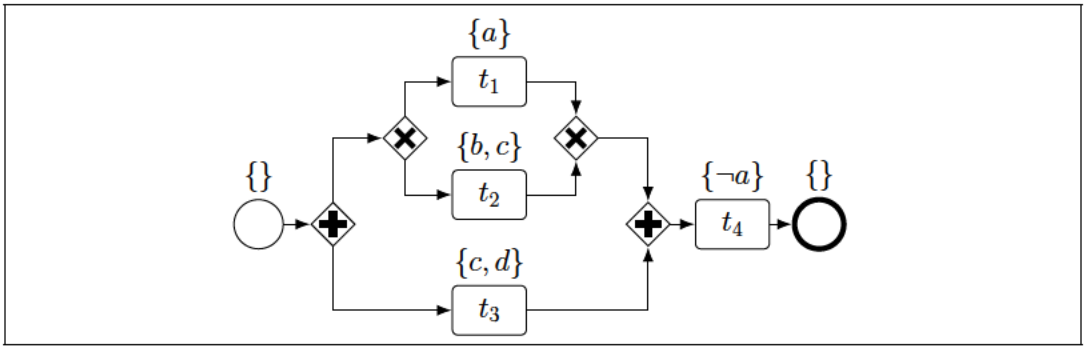


Figure 1: An annotated process

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

Table 1: Executions and Traces of the annotated process in Fig. 1.

## 2.2 Regulatory Framework

When considering a compliance problem, one of its components is the set of regulations that the model is required to follow. We refer to such component as the regulatory framework, and we consider it as a set obligations, representing the set of regulations governing the process model within the scope of the problem.

As such, we use a subset of Process Compliance Logic (PCL), introduced by Governatori and Rotolo [30, 31], to specify the semantics for different types of obligations proposed by [38].

The first distinction in the semantics of obligations, is that obligations can be either *global* or *local*. A global obligation is in force for the entire duration of an execution, while the in force interval of a local obligation is determined by its trigger and deadline conditions. Additionally, an obligation can be either an achievement or a maintenance obligation and it determines how such an obligation is fulfilled by an execution when in force.

**Definition 5** (Global and Local Obligations). *The in force interval of an obligation depends on whether it is a global or a local obligation, described as follows:*

**Global** *A global obligation $\mathcal{O}^o\langle c \rangle$, where $o \in \{a, m\}$ represents whether the obligation is achievement or maintenance. The element $c$ represents the fulfilment condition of the obligation.*

**Local** *A local obligation $\mathcal{O}^o\langle c, t, d\rangle$, where $o \in \{a, m\}$ represents whether the obligation is achievement or maintenance. The element $c$ represents the fulfilment condition of the obligation, the element $t$ the trigger, and the element $d$ the deadline.*

While the in force interval of a global obligation spans the entire duration of a trace, the in force interval of a local obligation is determined as a sub-trace where the first state of such a sub-trace satisfies the trigger, and the last state satisfies the deadline.

Generally the trigger, deadline and condition of an obligation are defined as propositional formulae. Assuming the literals from $\mathcal{L}$ contained in a state to be true, then a propositional formula is true when that state implies it.

Finally, in the semantic we study for each obligation we allow a single in force interval at any given time. Meaning that when an in force interval is already active for an obligation, further triggers would not produce additional in force intervals. This has the consequence that it simplifies the analysis as it is not required to keep track of multiple in force instances, and which in force instance is satisfied by which event when executing a task.

**Evaluating the Obligations.**

Whether an in force obligation is fulfilled or violated is determined by the states of the trace included in the in force interval of the obligation. Moreover, whether an in force obligation is fulfilled depends on the type of an obligation, as described in Definition 6.

**Definition 6** (Achievement and Maintenance Obligations)**.** *How an in force obligation is fulfilled depends on its type as follows:*

**Achievement** *If this type of obligation is in force in an interval, then the fulfilment condition specified by the obligation must be satisfied by the execution in at least one point in the interval before the deadline is satisfied. If this is the case, then the obligation in force is considered to be satisfied. Otherwise it is violated.*

**Maintenance** *If this type of obligation is in force in an interval, then the fulfilment condition must be satisfied* continuously *in all points of the interval until the deadline is satisfied. Again, if this is the case, then the obligation in force is then satisfied, otherwise it is violated.*

**Process Compliance.**

The procedure of proving whether a process is compliant with a regulatory framework can return different answers. A process is said to be *fully compliant* if every trace of the process is compliant with the regulatory framework[3]. A process is *partially compliant* if there exists at least one trace that is compliant with the regulatory framework, and *n*ot compliant if there is no trace complying with the framework.

**Definition 7** (Process Compliance)**.** *Given a process* $(P, \mathsf{ann})$ *and a regulatory framework composed by a set of obligations* $\mathbb{O}$*, the compliance of* $(P, \mathsf{ann})$ *with respect to* $\mathbb{O}$ *is determined as follows:*

- ***Full compliance*** $(P, \mathsf{ann}) \vdash^{F} \mathbb{O}$ *if and only if* $\forall \theta \in \Theta(P, \mathsf{ann})$*,* $\theta$ *satisfies each obligation in* $\mathbb{O}$*.*
- ***Partial compliance*** $(P, \mathsf{ann}) \vdash^{P} \mathbb{O}$ *if and only if* $\exists \theta \in \Theta(P, \mathsf{ann})$*,* $\theta$ *satisfies each obligation in* $\mathbb{O}$*.*
- ***Not compliant*** $(P, \mathsf{ann}) \nvdash \mathbb{O}$ *if and only if* $\neg\exists \theta \in \Theta(P, \mathsf{ann})$*,* $\theta$ *satisfies each obligation in* $\mathbb{O}$*.*

Note that we consider a trace to be compliant with a regulatory framework if it satisfies every obligation belonging to the set composing the framework.

# 3 Theoretical Computational Complexity in Structured Process Models

In this section we discuss the existing results concerning verifying regulatory compliance of structured business process models. We first introduce the acronyms used through the section to identify the different variants of the problem, and then we separately analyse and discuss the computational complexity results related to full, and partial compliance separately.

## 3.1 Problem Acronyms

Before discussing the existing computational complexity results, we first introduce a compact system to refer to different variants of the problem dealing with verifying compliance of structured process models. Notice that the acronyms refer to

[3]Notice that by "compliant with the regulatory framework", we refer to a trace fulfilling each in force interval along the trace itself for each obligation belonging to the regulatory framework.

the properties of the regulatory framework being evaluated against the structured process model.

**Definition 8** (Compact Acronyms)**.** *The variants of the problem we refer to in this paper constantly aim to check regulatory compliance of a structured process model. The acronym system refers to the properties of the obligations being checked against the process model.*

**1/n** *Whether the structured process is checked against a single (**1**) or a set of (**n**) obligations.*

**G/L** *Whether the* in force *interval of the obligations is **G**lobal, meaning that it spans the entirety of an execution of the model, or it is **L**ocal, meaning that the in force interval is determined by the trigger and deadline elements of an obligation.*

**-/+** *Whether the elements of the obligation being checked on the structured process model are composed by literals (-), or by propositional formulae (+).*

For instance, the variant **1G-** consists of verifying whether a structured process model is compliant with a single obligation, whose condition is expressed as a propositional literal and its in force interval spans the entire execution of the process model.

Note that in the binary properties of the problems considered in this paper, the leftmost, i.e., **1** in **1/n** represents a subset of the right side. Intuitively, the case on the right side is at least as complex as the left case. For instance, a solution for a problem including a set of regulations requires also to solve the case where the set of regulations is composed of exactly one regulation.

## 3.2 Partial Compliance

We focus now on discussing the computational complexity of proving partial compliance of structured business process models. As we see in the remainder of this section, many of the variants belong to the **NP**-complete computational complexity class. Thus we provide quick reminder before proceeding by discussing the existing results.

**Definition 9** (NP-complete)**.** *A decision problem is **NP**-complete if it is in the set of **NP** problems and if every problem in **NP** is reducible to it in polynomial-time.*

To prove membership in **NP** of a variant of the problem of proving partial compliance, we show that a 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 business process model and the set of obligations) with which we can check whether it fulfils the regulatory framework in polynomial time. As a certificate we use a trace of the model and we check whether it satisfies the regulatory framework.

We illustrate in the following Algorithm 1 how **1G-** is solvable in time polynomial. Notice that, while the algorithm reported applies only to achievement obligations, in the original paper by Colombo Tosatto et al. [14], from which we took this approach, an algorithm dealing with a regulatory framework composed of a maintenance obligation is also provided. Moreover, notice that the algorithm reported is capable to prove either partial, full, and non-compliance in polynomial time.

**Algorithm 1** (**1G-** is in **P**)**.** *Given an annotated process* $(P, \mathsf{ann})$ *and a regulatory framework* $\mathbb{O}$ *containing a single global achievement obligation* $\mathcal{O}^a\langle c\rangle$*, this algorithm returns whether* $(P, \mathsf{ann})$ *is compliant with* $\mathbb{O}$*.*

1: ***if*** $\forall$ *task* $t$ *in* $P, c \notin \mathsf{ann}(t)$ ***then***
2: ***return*** $(P, \mathsf{ann}) \not\vdash \mathbb{O}$
3: ***else***
4: ***if*** $\mathsf{Remove}(P, \{t \mid t \text{ is a task in } P \text{ and } c \in \mathsf{ann}(t)\}) = \bot$ ***then***
5: ***return*** $(P, \mathsf{ann}) \vdash^F \mathbb{O}$
6: ***else***
7: ***return*** $(P, \mathsf{ann}) \vdash^P \mathbb{O}$
8: ***end if***
9: ***end if***

*Where the* Remove *functions removes the tasks from* $P$ *having* $c$ *annotated, and later checks whether there is a path, in other words an execution, from the start to the end of the process. If no such path exists then the function returns* $\bot$*, which means that there is no execution that does not execute a task having* $c$ *annotated. Meaning that the process is fully compliant.*

In Reduction 1 we show the reduction provided by Colombo Tosatto et al. [10], and showing that the problem of finding whether a graph contains an Hamiltonian path can be reduced to the problem of proving partial compliance in **nL-**. Meaning that the computational complexity of **nL-** is at least the same as proving whether a graph contains an Hamiltonian path, which is in **NP**-complete.

**Definition 10** (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. $\forall i, j((v_i, v_j \in ham \wedge j = i + 1), ((v_i, v_j) \in D)$

**Reduction 1** (Hamiltonian Path to Proving Partial Compliance in **nL-**). *Considering the problem of finding an Hamiltonian Path in a graph as described in Definition 10.*

*Given a hamiltonian path problem containing a directed graph $G = (N, D)$, it can be translated into a regulatory compliance problem involving a process $(P, \mathsf{ann})$ and a set of obligations $\mathbb{O}$ as follows:*

**1** *Consider a process model $P$ that contains a task labeled $Node_i$ for each vertex $v_i$ contained in $N$ (Figure 2).*

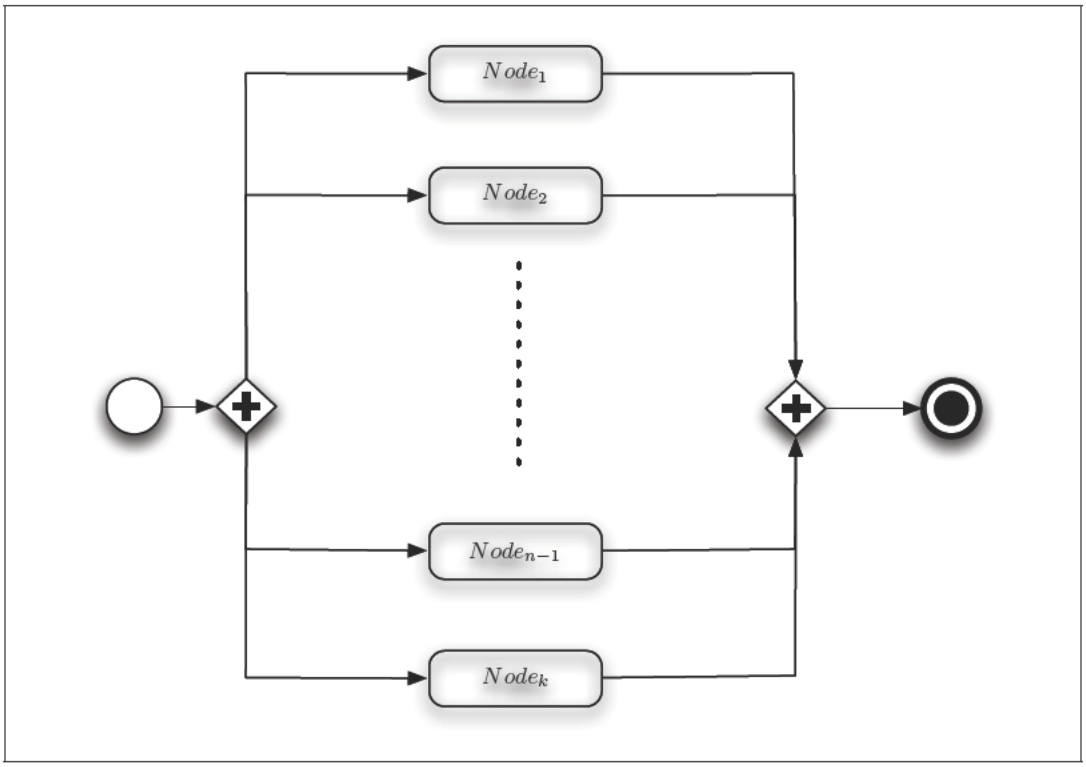


Figure 2: Hamiltonian path problem as verifying partial compliance.

*The process block of $P$ is structured as an* AND *block. The* AND *block contains in each branch a single task $Node_i$ for each node in the given directed graph:* $\mathsf{AND}(Node_1, \ldots, Node_n)$.

*Intuitively a serialisation of the* AND *block represents a tentative hamiltonian path. 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****).*

**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 $(P, \mathsf{ann})$ is the following:*

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

*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 $\mathbb{O}$ contains the following local maintenance obligations:*

- $\forall v_i, v_j | (v_i, v_j) \notin D, \mathcal{O} = \mathcal{O}^m \langle \neg l_j, l_i, \neg l_i \rangle$

*Using the proposed reduction, verifying whether the constructed process model is partially compliant corresponds to identifying whether the original graph contains a hamiltonian path. Concluding that the problem of verifying partial compliance is at least as hard as finding whether a graph contains a hamiltonian path.*

We collect in Table 2 the existing computational complexity results concerning solving the variants of the problem of proving partial compliance of structured process models.

Notice that given the three binary properties associated to the regulatory framework being checked against the structured process model, of the 8 possible problem variants, only 7 computational complexity results are provided in Table 2. This is more apparent by illustrating the results in Figure 3, where the problem's variants have their computational complexities associated and the relations between the variants are highlighted by the connections in the picture. Notice that the directed arrows connecting one variant of the problem to another refer, according to their direction, that the computational complexity of a variant of the problem at the origin of an arrow, is at most as difficult as the variant of the problem which is pointed at by the same arrow.

| Problem Variant | Source | Complexity Class |
|---|---|---|
| **1G-** | Colombo Tosatto et al. [14] | **P** |
| **nG-** | Colombo Tosatto et al. [11] | **NP**-complete |
| **1G+** | Colombo Tosatto et al. [11] | **NP**-complete |
| **nG+** | Colombo Tosatto et al. [11] | **NP**-complete |
| **nL-** | Colombo Tosatto et al. [10] | **NP**-complete |
| **1L+** | Colombo Tosatto et al. [11] | **NP**-complete |
| **nL+** | Colombo Tosatto et al. [10] | **NP**-complete |

Table 2: Partial Compliance Complexity

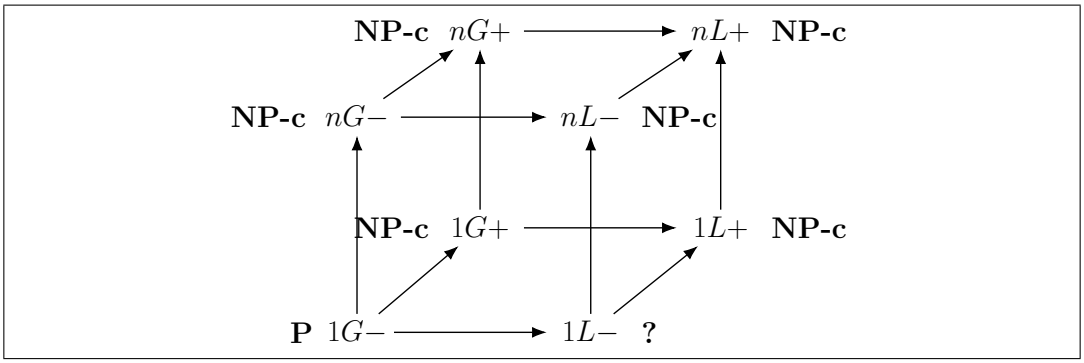


Figure 3: Partial Compliance Complexity Lattice.

It can be noticed in Figure 3, that the variant **1L-** does not have a computational complexity classification yet. While the computational complexity analysis for the considered problem is currently incomplete, Colombo Tosatto et al. [11] conjectured that this variant of the problem to in **P**. However, while we have not yet managed to provide a conclusive computational complexity classification, we conjecture that **1L-** is instead in **NP-c** as explained in Conjecture 1

**Conjecture 1** (**1L-** is in **NP**)**.** *We currently have no information about the computational complexity of **1L-**. That is, we cannot infer its belonging to a computational complexity class in a similar way as for **nG+**, as in this case the simpler variant (**1G-**) is in **P**.*

*While it seems like that moving from **G** to **L** seems to not increase the complexity of the problem as much as when moving from **-** to **+**, or from **1** to **n**, we believe that such movement should be still be capable of bringing the computational complexity of the problem's variant into **NP-c**.*

*We back our conjecture using the intuition that by moving towards conditional*

*obligations, allows multiple instances of the same obligation to be in force over a single trace. Which means that even for the variant **1L-**, multiple instances would be required to be verified for every trace. Which resembles the variant **nG-**, where multiple obligations are required to be verified over a trace, and it is in **NP-c**.*

Mind that the conjecture does not represent a computational complexity result in itself, hence identifying the computational complexity of the variant **1L-** remains an open problem.

### 3.3 Full Compliance

We focus now on discussing the computational complexity of the variants of the problem of proving full compliance of a structured process model. As many of the variants of the problem belong to the co**NP**-complete computational complexity class, we provide its definition before proceeding with the discussion.

**Definition 11** (co**NP**-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**.*

We show in Reduction 2 how Colombo Tosatto et al. [10] have shown that checking for full compliance in a variant of the problem **1L+** is in co**NP**-complete.

**Definition 12** (Tautology)**.** *A formula of propositional logic is a tautology if the formula itself is always true regardless of which evaluation is used for the propositional variables.*

**Reduction 2** (Tautology to Proving Full Compliance in **1L+**)**.** *Considering the problem to decide whether a given formula is a Tautology as described in Definition 12.*

*Let $\varphi$ 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 $\varphi$. We include in $L$ only the positive version of a literal, for instance if $l$ or $\neg l$ are contained in $\varphi$, 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 process $(P, \mathsf{ann})$, graphically represented in Figure 4.*

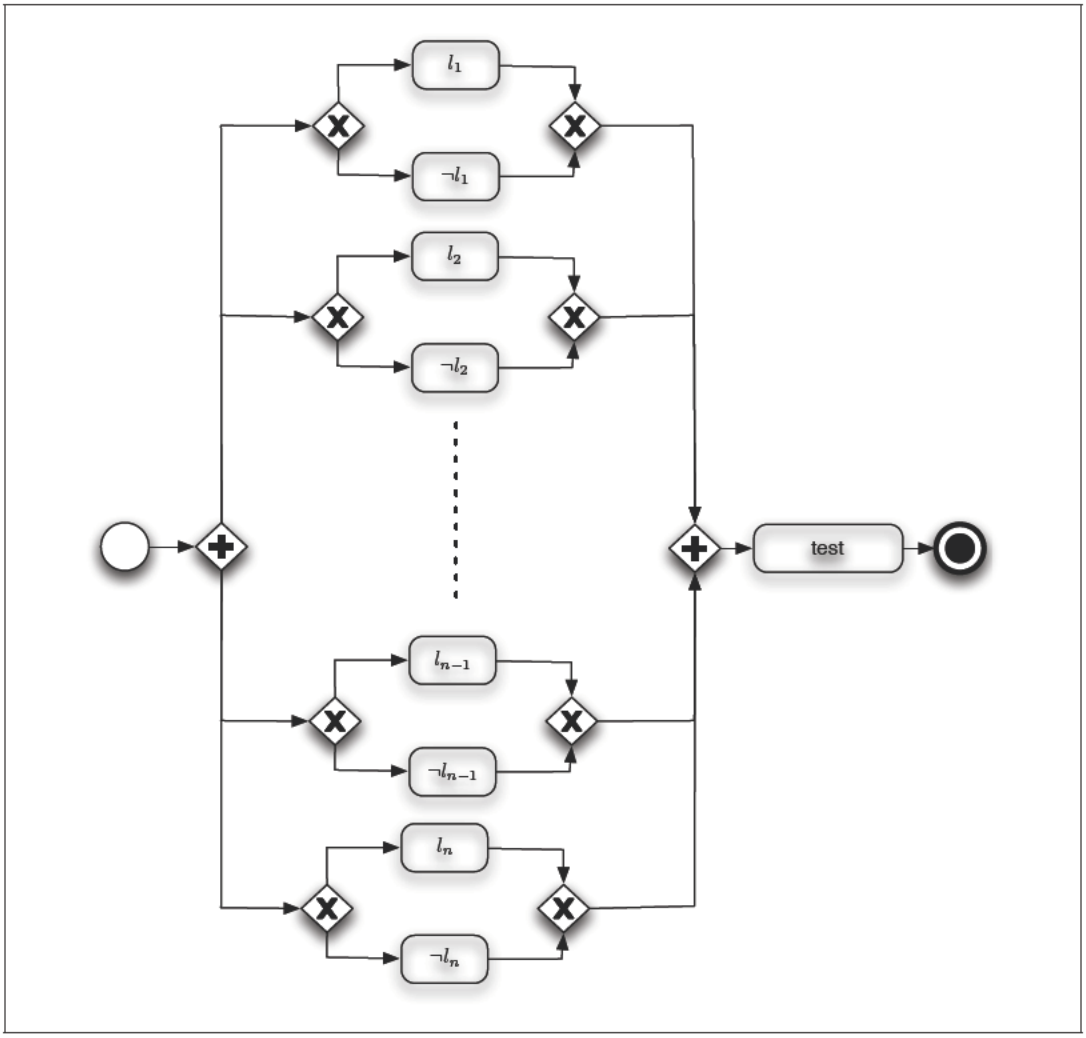


Figure 4: Tautology problem as verifying full compliance.

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

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

Notice that Algorithm 1 can also be used to prove full compliance of the variant **1G-**. For full compliance, we informally describe in Algorithm 2 the procedure

introduced by Colombo Tosatto et al. [12], capable of proving full compliance of the variant **nL-** of the problem in time polynomial with respect to the problem size.

**Definition 13** (Process Tree Model)**.** *Let $P$ be a structured process model. A* Process Tree *$PT$ is an abstract hierarchical representation of $P$, where:*

- *Each process block $B$ in $P$ corresponds to a node $N$ in $PT$.*
- *Given a process block $B(B_1, \ldots, B_n)$, where $B_1, \ldots, B_n$ are the process blocks directly nested in $B$, the nodes $N_1, \ldots, N_n$ in $PT$, corresponding to $B_1, \ldots, B_n$ in $P$, are children of $N$, corresponding to $B$ in $P$. Mind that the order between the sub-blocks of a process block is preserved between the children of the same node.*

**Algorithm 2.** *The approach is based on identifying whether a structured business process model, in its tree representation form as described in Definition 13, contain a trace violating one of the obligations belonging to the regulatory framework.*

*The advantage of looking for a violating condition, is that finding a single instance within a process model where such condition is positively evaluated, it is a sufficient condition to return the answer that the structured process being evaluated is not* fully compliant *with the regulatory framework.*

*The tree representation of a process model has as its leaves the tasks composing the process. Considering a generic obligation $\mathcal{O}^o\langle c, t, d\rangle$, each of the leaves in a process tree associated to a task having $t$ annotated are considered* trigger leaves*. A bottom up aggregation of the properties of the leaves of the tree, in accordance to their associated annotated tasks, and to the violation condition being looked for, leads to allow whether a process tree contains a violation for a given* trigger leaf *in a number of steps equal to the number of nodes in the process tree.*

*Repeating this procedure for each trigger leaf, for each violation condition of each obligation in a regulatory framework, allows to decide, when no violation condition is satisfied, that the business process model being evaluated is fully compliant, and this is decidable in time polynomial with respect to the size of the problem.*

Table 3 outlines the existing complexity results concerning some of the variants of the problem of proving full compliance of structured process models.

Similarly as for partial compliance, we illustrate the result concerning the computational complexity of proving full compliance of a process model graphically in Figure 5.

Notice that Figure 5 contains a result for the problem variant **nG-**, which is not included in Table 3. This result is derived from other existing ones. As the relations

| Problem Variant | Source | Complexity Class |
|---|---|---|
| **1G-** | Colombo Tosatto et al. [14] | **P** |
| **1L-** | Colombo Tosatto et al. [12] | **P** |
| **nL-** | Colombo Tosatto et al. [12] | **P** |
| **1G+** | Colombo Tosatto et al. [13] | co**NP**-complete |
| **nG+** | Colombo Tosatto et al. [13] | co**NP**-complete |
| **1L+** | Colombo Tosatto et al. [10] | co**NP**-complete |
| **nL+** | Colombo Tosatto [9] | co**NP**-complete |

Table 3: Full Compliance Complexity

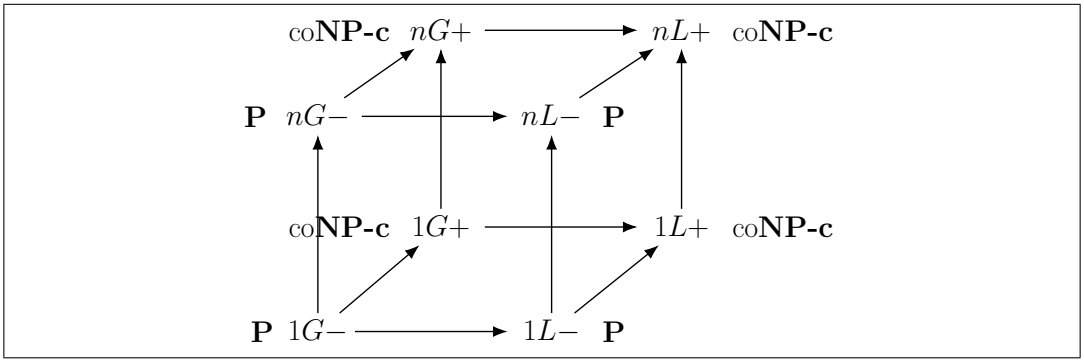


Figure 5: Full Compliance Complexity Lattice.

in the lattice in Figure 5 represent the relation $\leq$ between the computational complexities of the variants of the problem, if we consider the three variants **1G-**, **nL-**, and **nG-**, the follow relationship holds regarding their computational complexity: **1G-** $\leq$ **nG-** $\leq$ **nL-**. Therefore, by knowing that both **1G-** and **nL-** are in **P**, it follows that also **nG-** is in **P**.

### 3.4 Climbing the Polynomial Hierarchy

The computational complexity results reported so far concerning the problems of proving both partial and full compliance of structured process model, rely on an assumption regarding how formulae composing the obligations are evaluated over the states composing the traces.

**Assumption 1** (States Satisfying Formula)**.** *Given a state $\sigma$, composed by a set of positive and/or negative literals, and a propositional formula $\varphi$, $\varphi$ is satisfied by $\sigma$ if and only if considering every literal in $\sigma$ true, is a sufficient interpretation to*

*make $\varphi$ true. This is equivalent to evaluate a formula over a partial set of possible interpretations, the ones that explicitly appear in the state.*

While the assumption does not appear to be too surprising, it can lead to some interesting behaviours, such as the following: consider the formula $\alpha \vee \neg\alpha$, which is a tautology. Now, if we consider whether an empty state of a trace would satisfy, the formula, then the answer is counter-intuitively *no* in accordance to Assumption 1.

The effect of Assumption 1 on how formulae composing the obligations are verified over the states of the traces, is to simplify the verification, as the only interpretation required to be verified is the one where every proposition in the state is considered as true. If such interpretation is sufficient to evaluate the formula as true, then its associated effects, according to its place in the obligation, are applied. Differently, if the interpretation provided by the state is not sufficient to fully evaluate the formula, then it is assumed to be false in that state. Normally, without Assumption 1, when a state does not contain a sufficient interpretation, then the various cases are considered for the propositions which have not an assigned truth value. This can potentially increase the computational complexity of solving the problem, as evaluating a formula over a state can be reduced to a *Satisfiability* problem, which is known to be **NP**-complete. In order to properly classify the variants of the problem when Assumption 1 is dropped, we need first to introduce the *Polynomial Hierarchy.*

The Polynomial Hierarchy is a hierarchy of computational complexity classes describing both the classes already discussed in the present chapter (**P**, **NP** and co**NP**), as well as more complex classes. In the Polynomial Hierarchy **P** is also represented as either $\Sigma_0^{\mathsf{P}}$ or $\Pi_0^{\mathsf{P}}$, while **NP** and co**NP** are respectively represented as $\Sigma_1^{\mathsf{P}}$ and $\Pi_1^{\mathsf{P}}$.

**Definition 14** ($\Sigma_1^{\mathsf{P}}$)**.** *A problem $P$ is in $\Sigma_1^{\mathsf{P}}$ if there exists a polynomial time Turing machine $T$ and a polynomial $p$ such that:*

*for each instance $x$ of $P$: there exists a solution $s$, $|s| \leq p(|x|), T(x, s) = true$*

**Definition 15** ($\Pi_1^{\mathsf{P}}$)**.** *A problem $P$ is in $\Pi_1^{\mathsf{P}}$ if there exists a polynomial time Turing machine $T$ and a polynomial $p$ such that:*

*for each instance $x$ of $P$: for each solution $s$, $|s| \leq p(|x|), T(x, s) = true$*

Considering now the problem of proving partial compliance of a structured process model, when Assumption 1 is dropped. We have that for the variants of the problem allowing formulae to describe the elements of the obligations, the problem becomes the following: there exists a trace of the model such that, for each state

state in the trace, and for each possible interpretations of the state the formulae composing the obligations are satisfied in such a way that no obligation is violated. It can be noticed, that this problem involves an existential quantifier followed by two universal quantifiers: $\exists\forall\forall$. While not delving too much into the technical details, when multiple quantifier of the same type directly follow each other, they can be collapsed as a single quantifier of the same type. Given that, and Definition 16, we can see that the variants of the problem of proving partial compliance, and allow formulae in their obligations, can be classified as $\Sigma_2^{\mathsf{P}}$. Furthermore, notice that the variants restricting the expressivity of their obligations to simple propositions are not affected and remain in $\Sigma_1^{\mathsf{P}}$.

**Definition 16** ($\Sigma_2^{\mathsf{P}}$)**.** *A problem $P$ is in $\Sigma_2^{\mathsf{P}}$ if there exists a polynomial time Turing machine $T$ and a polynomial $p$ such that:*

*for each instance $x$ of $P$: there exists a solution $s$, $|s| \leq p(|x|)$ such that each $s'$, $|s'| \leq p(|x|)$, $T(x, s, s') = true$*

The lattice with the computational complexity classifications according to the Polynomial Hierarchy, after dropping Assumption 1, is shown in Figure 6.

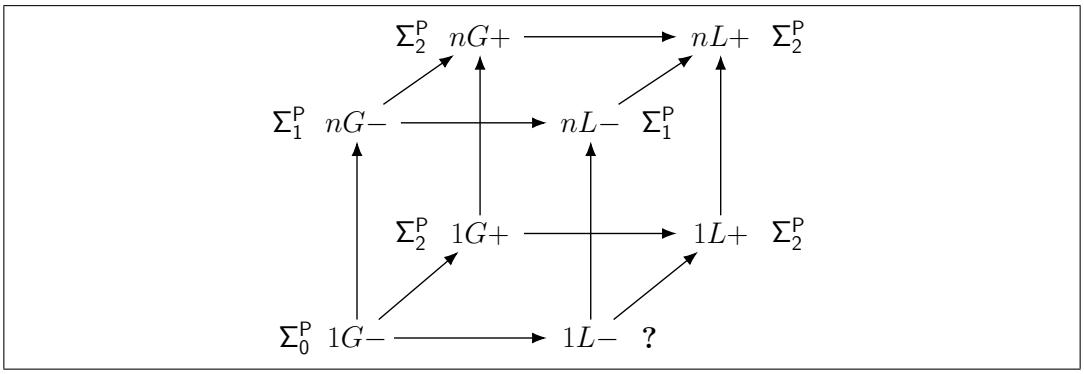


Figure 6: Partial Compliance Complexity Lattice with Polynomial Hierarchies.

Similarly, we can consider the problem of proving full compliance of structured process models, when Assumption 1 is dropped. Again, we have that for the variants allowing formulae in their obligations the problem becomes: for each trace of the model such that, for each state state in the trace, and for each possible interpretations of the state the formulae composing the obligations are satisfied in such a way that no obligation is violated. It can be noticed, that this problem involves three universal quantifiers: $\forall\forall\forall$. Again, we can collapse the quantifiers of the same type with the neighbouring ones, which leads to a single universal quantifier in this case. It can be noticed that these problems do not have the sufficient properties to be

classified as $\Pi_2^P$, as described in Definition 17, but they can be classified as $\Pi_1^P$, as described in Definition 15.

**Definition 17** ($\Pi_2^P$)**.** *A problem $P$ is in $\Pi_2^P$ if there exists a polynomial time Turing machine $T$ and a polynomial $p$ such that:*

*for each instance $x$ of $P$: for each solution $s$, $|s| \leq p(|x|)$ such that there exists a $s'$, $|s'| \leq p(|x|)$, $T(x, s, s') = true$*

Therefore we can conclude that for the problem of proving full compliance, releasing Assumption 1 does not increase the complexity, as the variants allowing propositional formulae are still in co**NP**. For completeness we show the lattice with the Polynomial Hierarchy classifications for the variants of the problem of proving full compliance if Figure 7.

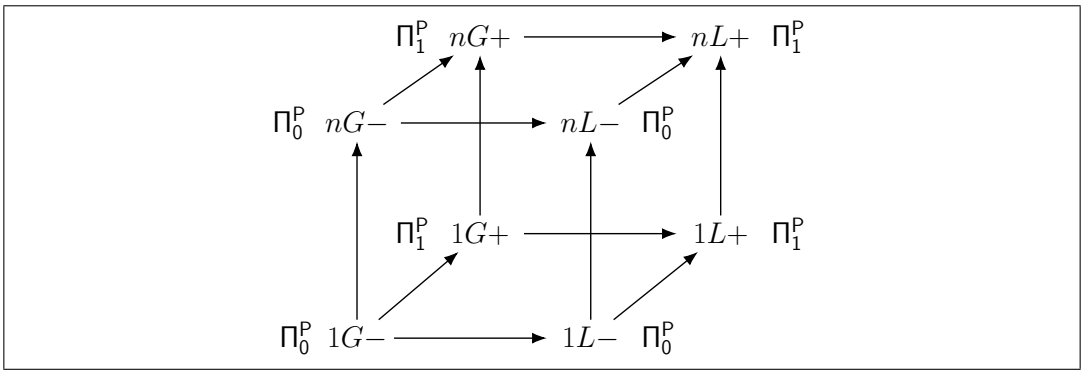


Figure 7: Full Compliance Complexity Lattice with Polynomial Hierarchies.

## 3.5 Summary

The computational complexity results illustrated in this section show that, when considering variants of the problem only allowing literals to represent the components of the obligations, proving full compliance of structured process model is generally easier than proving partial compliance. Intuitively, the former is easier since it is sufficient to find a violation for an obligation in one of the traces of the model, while for the former, it is required to identify a trace, and ensure that no obligation is violated in such a trace.

Moreover, it can be noticed from the computational complexity lattice in Figure 7, that for full compliance the computational complexity is be governed by the complexity of the language, namely by how expressively we can represent the elements representing the obligations.

Differently, partial compliance seems to be more complex to verify, as it does not seem to allow an easy way to identify its complement, and identifying a compliant trace in a process model is shown to be intractable apart from the the easiest, or maybe two easiest, variant(s) of the problem as illustrated in Figure 6.

Finally, we have also shown that by dropping Assumption 1, the computational complexity of the problem of proving partial compliance starts to climb the Polynomial Hierarchy.

# 4 Computational Complexity of Additional Business Process Features

The variants of the problem discussed earlier in this chapter have their computational complexity depending solely on the varying properties of the regulatory framework while keeping the properties of the process model static. Additional computational complexity analysis can be done when considering more complex variants of the process models used.

In this section we discuss about the computational complexity of proving compliance of process models by including additional features in the process models. In particular we discuss about the computational complexity of verifying compliance of unstructured process model, and the computational complexity impact of including loops in business process models.

## 4.1 Unstructured Process Models

The computational complexity analysis included in Section 3 focused on problems where the the process models were structured. As mentioned earlier, one of the advantages of such models is that their soundness can be verified in time polynomial with respect to the size of the model. Verifying soundness means to check whether every execution in the model is a proper execution, and capable of reaching the *end* of the model. In other words checking that the process model do not contain livelocks and or deadlocks preventing any of the contained execution to successfully complete.

Unstructured business process models, which are not composed by properly nested process blocks, as the instance shown in Figure 8, do not guarantee that their soundness can be verified in polynomial time. As it has been shown by van der Aalst [66], and Lohmann et al. [47], the semantics of business process models can be captured by Petri Nets [52]. While this does not provide any direct result concerning the computational complexity about verifying compliance of unstructured process

model, it still provides an upper complexity bound, as coloured petri nets, one of the more complex variants of this formalism, is known to be undecidable [19, 51].

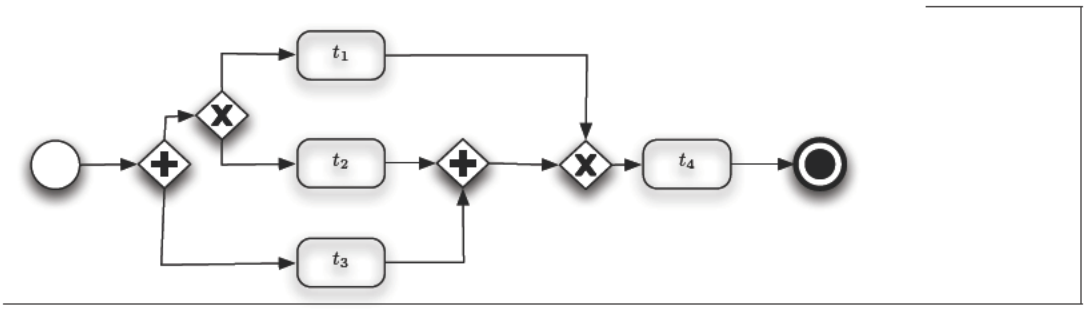


Figure 8: Unstructured Process Model

**Open Problem**

Studying the computational complexity of the problem of proving regulatory compliance of unstructured process models is still an open problem. Knowing already that if a unstructured process model is complex enough to require a coloured petri net to represent its semantics, then also the corresponding problem of verifying compliance becomes undecidable. Therefore, one option is to study which classes of unstructured processes do not require coloured petri nets to represent their semantics, and how they affect the computational complexity of solving the problem of proving regulatory compliance.

When studying how unstructured process models affect the computational complexity of the compliance problem, it is also crucial to consider that such models can be potentially translated into structured models, as discussed by Polyvyanyy et al. [54]. However the computational complexity cost of such translations should be still factored in the computational complexity of the problem, which still requires to be studied.

## 4.2 Loops

Loops are structures that allow to repeat the execution of parts of a process model. Figure 9 shows a process model containing a loop structure, in this model, the execution of the pair of tasks $t_4$ and $t_5$ can be executed any number of times between one and infinite. By allowing repeated executions of some of its elements, some issues can arise affecting the computational complexity of the problem of verifying regulatory compliance. In this section we discuss some of out intuitions behind these possible issues affecting the computational complexity of the problem.

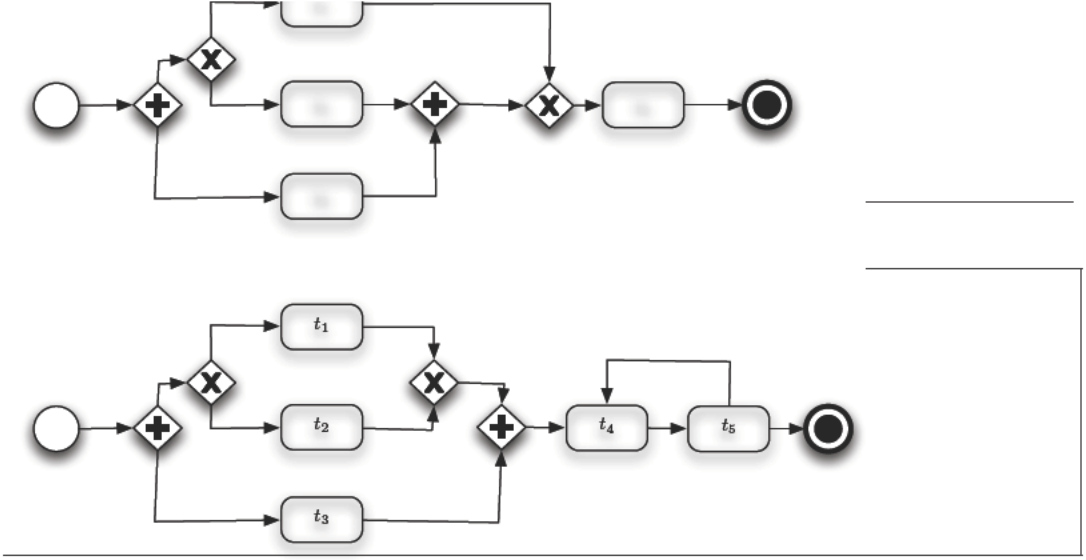


Figure 9: Process Model containing Loop

### Infinite Possible Executions

By allowing repeated executions of some components of a process model, the maximal length of their executions is not constrained anymore by the number of tasks contained in the process model, but can become potentially infinite as a task within a loop can be executed an infinite number of times. Therefore, approaches based on verifying directly whether the traces[4] of a model comply with the regulatory framework, are not applicable to this variant of the problem as some of these traces can in theory be infinite, and such approaches could never terminate and return a result.

Therefore, introducing loops into the model would make pure brute force approaches, the ones analysing the generated traces of a model, completely obsolete. We argue that similar issues would apply to every approach analysing the traces instead of the structure of the model, however techniques to recognise repeating patterns in the process' states could be potentially used to recognise loops and study their compliance effects. However whether this techniques would work, and the actual computational complexity of the problem including loops are both unanswered research questions.

### Structural Analysis

Differently, approaches that analyse the process's structure without the need to analyse each trace explicitly, would still be able to terminate in a finite amount of steps. Thus, we argue that such kind of approaches, which do not need to explicitly analyse the traces, are the only viable approaches when dealing with process models containing loops.

[4]As a reminder, a trace is the sequence of the tasks in an execution with associated process' states at every step of the execution.

However, the theoretical computational complexity of the variants of the problem identified in Section 3, with the inclusion of loops, is at least as difficult as the ones not including loops. This is due to the additional complexity brought by including the loops in the process model. While the added complexity may not necessarily be enough to increase the computational complexity class of the variants to harder ones, a through analysis of the computational complexity of the new variants including loops is still required in order to properly classify them.

### Relating to Complete Turing Machines

We discuss now the intuition concerning why introducing loops in business process model, in addition to including conditions in the decision points[5], would allow potentially to simulate universal Turing machines using these extended business process models.

**Definition 18** (Turing Completeness)**.** *In computability theory, a system of data-manipulation rules (such as a computer's instruction set, a programming language, or a cellular automaton) is said to be Turing complete or computationally universal if it can be used to simulate any Turing machine.*

Considering now adding into the business process models conditions for its loops, like starting and exit conditions, and decision condition for the mutual exclusive paths in the process model, it becomes more and more evident how the elements of a business process model can simulate different structures common to programming languages, such as various type of cycles and decision blocks. As these programming languages are generally known to be Turing complete languages, such as for instance *Java* and *C++*, considering the process state as the computational state of Turing machine, and the possible executions of a model as the possible computations of the Turing machine, then we can conclude that with these additions, such models become Turing complete.

As a consequence, considering the *halting problem*[6] [65] affecting Turing machines, it would then also affect the problem of proving regulatory compliance of business process models including loops. Which, in turn, would make the problem of proving compliance in general *undecidable*.

[5]We consider as decision points XOR blocks and loops, where a decision is required to be made concerning which branch to execute, or whether to exit the loop.

[6]The halting problem is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running or continue to run forever.

### Relation to Fairness

Given the relation between the problem of proving regulatory compliance of business process models and computer programs, which becomes particularly clear when loops and conditions are introduced within the process model. It is only fair to discuss some of the related work dealing with the termination of computer programs, in particular as the computational complexity of dealing with such more advanced variants of the problem is related to the undecidability of halting problem.

The relation we are going to discuss is the one with the concept of *fairness for model checking verification* as discussed earlier by Francez [21], which can be intuitively understood as the following: given two properties of comparable importance for a problem, if a certain effort is put into the verification of one of the property, then it is only *fair* that the same amount of effort is put towards the verification of the other property. Such concept led to Cook et al. [15] to investigate the verification of liveness properties of programs in addition to their safety properties. Others, such as Dobrikov et al. [17] proposed implementations under *fairness* assumptions, capable of verifying how fairly model checking approaches perform, with limited overhead. Moreover, Kesten et al. [43] propose a fairness based approach based on LTL for model checking verification, showing that the introduction of fairness allows to close the gap with other approaches using CTL, as by using strong fairness, LTL properties can be verified on the model being checked without having to completely unfold the model to generate the possible states.

Adopting fairness to verify properties of models allows to do so without having to unfold these models and explicitly verify the possible states, in particular by adopting *strong fairness*, which is also referred as *compassion*. This requirement, compassion, stipulates that, given two types of states, then in every computation which verifies infinitely many of one of the types, is also required to verify infinitely many of the other type.

Considering now the problem of verifying regulatory compliance of business process models including loops, and assuming the existence of conditions governing choices such as mutually exclusive paths in the model, and whether a loop should be repeated or exited. It can be noticed that fairness based approached for model checking can be adapted to deal with such kind of problems. In particular, if we consider loops in a process model and its entering and exiting conditions as conditions leading to two different types of states for which we require *strong fairness*, then approaches verifying compliance, even by analysing the traces of the model explicitly, would be required to analyse an equivalent amount of states within and outside loops, guaranteeing to consider in such a way traces that represent full executions of the model. While this technique can prove useful to verify partial compliance

of process models, as it requires to find a compliant trace, further analysis may be necessary when dealing with full compliance, as in order to be classified as such, every possible trace must be considered and verified.

### 4.3 Summary

Despite the added expressivity introduced by using unstructured business process models and loops can be useful to represent real world problems more faithfully, how such additions would affect the computational complexity of the variants of the problem discussed in Section 3 remain for the major part an unanswered research question.

## 5 Classification of Existing Approaches

In this section we consider and classify according to the variants identified in the present chapter some of the existing approaches proposed in the literature, and aiming to prove regulatory compliance of business process models. We organise the approaches according to the main technique used to solve the problem of proving regulatory compliance.

### 5.1 Control-Flow Based

These approaches focus on checking the execution order of the tasks in the business process models. To do so, temporal logic is generally used to verify properties over the execution orders of the tasks, some instances of this type of solution have been provided by Awad et al. [4], and by Lu et al. [48]. As the properties expressed through temporal logic formulae apply through the whole extents of the executions of the model, and refer directly to the tasks[7], then we can assign these approaches to the variant **nG-** of the problem.

Other approaches based on the control-flow can use different formalisms to represent the constraints between the execution order of the tasks. One of such approaches proposed by Groefsema et al. [34], uses $CTL^*$[8] to describe the constraints between the execution order of the tasks. As this approach allows to express conditional constraint, it can then be classified as belonging to the variant **nL-**. Mind

[7]Having constraints referring directly to the tasks, in this case where temporal logic is used, it means that some constraint over the execution order is expressed between two particular tasks. When this is the case, the constraints refer to the labels of the tasks, which can be considered as propositional literal.

[8]CTL* is a combination of computational tree logic and linear temporal logic, which allows to combine path quantifiers and temporal operators.

that the approach proposed by [34] adopts also some practical optimisation, as it does reduce the space-state while preserving the information contained in the concurrent components of the model. This provides only a practical optimisation for the problem, while the theoretical computational complexity still holds for its worst case scenarios.

### 5.2 Temporal Logic Related

The approach proposed by Elgammal et al. [18] introduces a new language to represent the execution constraints between the tasks of a business process model. This language, named *Compliance Request Language*, is used by [18] to define patterns that must be followed by the executions of the model. While the language proposed allows to compactly express these patterns, the authors shows that *Linear Temporal Logic* can be used to represent the patterns. As the constraints over the executions of the business process model are conditional, we can assign this approach to the variant **nL-**.

### 5.3 Classical Logic Related

Considering now approaches [45, 23, 41, 35] adopting classical logic formalisms to represent the constraints over the execution of the process model, the introduction of logical formulae allows to represent more complex conditions and constraints. For instance the constraint over the order execution of a task can be conditional with respect to the execution of a set of tasks, or the combined effects of a set of tasks must be achieved before a given deadline. Given the expressivity that can be achieved by such approaches, then we classify as belonging to the variant **nL+**.

### 5.4 Modal Logic Related

Other approaches, such as the ones proposed by Sadiq et al. [59], and Governatori [24], adopt modal logic to represent the constraints over the allowed execution orders of the tasks of a business process model. While the inclusion of modalities over classical logic based approaches allows to improve the expressivity of the constraints, the more expressive constraints are still compatible with the variant **nL+** of the problem.

### 5.5 Practical Optimisations

Given the inherent computational complexity of the problem, several approaches have adopted techniques allowing to reduce the search state-space of the problem to

| Problem Variant | Approach |
|---|---|
| **nG-** | Control-Flow Based [4] |
| | Control-Flow Based [48] |
| **nL-** | Control-Flow Based using CTL* [34] |
| | Temporal Logic [18] |
| | Practical Optimisation [53] |
| | Practical Optimisation [8] |
| | Practical Optimisation [60] |
| | Practical Optimisation [20] |
| | Practical Optimisation [39] |
| **nL+** | Classical Logic [45] |
| | Classical Logic [23] |
| | Classical Logic [41] |
| | Classical Logic [35] |
| | Modal Logic [59] |
| | Modal Logic [24] |

Table 4: Classifying Existing Approaches

limit the state explosion in concurrent processes. However, these approaches either generate large amounts of overhead, such as for instance the one introduced by Nakajima [53], or lose information on concurrency and the orders of local tasks due to the linearisation of the concurrent components of the process model, as shown in the following approaches [8, 60, 20, 39]. We classify the approaches based on practical optimisations as belonging to the variant **nL-**.

## 5.6 Summary

We conclude this section by summarising the classification of some of the existing approaches in Table 4.

The first thing that can be noticed is that every single approach falls into the **NP**-complete computational complexity class when the goal is to prove partial compliance of a business process model. Differently, if the aim is to verify whether a model is fully compliant with the given regulations, then the logic based approaches are in co**NP**-complete, while the others can be theoretically solved in polynomial time.

The second and final observation over Table 4, concerns the distribution of the approaches over the various variants of the problem. We would like to point out that when global ordering constraints, as given in *control-flow* based approaches, then

the problem lies in the **nG-** variant. Introducing conditional constraints, usually through the adoption of *temporal logics* or derivates, moves the problems into the **nL-** variant. Finally, when full fledged *logical formalisms* are used to represent the constraints, then the problem reaches the most difficult variant discussed in this chapter: **nL+**.

# 6 Conformance and Normative Reasoning

In this section we discuss some disciplines related to the problem of proving regulatory compliance of business process models. In particular we discuss *conformance*, verifying whether a trace from a log is a proper execution of a given process model, and normative reasoning, the discipline tasked with reasoning and deal with normative concepts, such as obligations and violations.

In addition to discuss the relations of these disciplines with the problem discussed in this chapter, we also discuss their computational complexity and how it relates to the results presented in this chapter for the problem of proving regulatory compliance.

## 6.1 Conformance Checking

Conformance checking, as defined by van der Aalst [69], refers to the discipline of verifying whether the executions contained in a given event log are the proper executions of a given process model. To put it differently, using van der Aalst words: "The goal is to find commonalities and discrepancies between the modeled behaviour and the observed behaviour."

While conformance checking and compliance checking are orthogonal disciplines, mainly related as both deal with business process models, both prove useful in verifying the properties of models and their actual behaviour, and used together allows to ensure the compliance of the actual behaviours of business process models in real life scenarios. Considering a business process model used by an organisation, its compliance can be verified by using one of the many available techniques. In particular, if we focus on the case where *full compliance* is being proven for such a model, what is verified is that every proper execution of the model is compliant with the regulatory framework in place. However, while this is indeed a desirable property of a business process model, as van der Aalst mentions, its not always the case that the modelled behaviour of a business process model in an organisation, perfectly reflects the actual observed behaviour of how such organisation performs its business. Therefore, *conformance* becomes extremely important to verify whether the actual behaviour follows the modelled one, ensuring in this way that the organisation is

compliant with the regulations. Moreover, when discrepancies are detected, it is desirable to realign the modelled behaviour of the organisation with the observed one, using available techniques such as for instance *process mining* [70][9], in such a way the compliance of the realigned model can be rechecked and if the actual behaviour of the organisation keeps following the realigned model, then regulatory compliance is assured.

**Token Replay**

A technique to verify conformance of traces in a given event log with respect to a business process model is by using token replay [58]. This technique, as the name suggests, consists of trying to replay the traces over the model. One thing to notice, is that this technique is originally designed to verify the conformance of event logs with workflow models based on petri nets, such as the approach proposed by [1], as described in Chapter 4, Part 2, Section 2.2. Despite this difference, the technique can be adapted to deal with business process models as well, especially given their similarities as pointed out by [44].

The original technique, replaying traces over workflow models constructed using petri nets, is based on going through the list of tasks representing the trace being checked whether it conforms with the workflow model. The workflow is setup having a token in its starting *place*, and each task in the trace is then checked to verify whether the current state of the model allows its execution, in accordance to the state of the tokens[10]. After, tokens in the precondition set are consumed and recreated in the postcondition set of the task in the model. This is repeated for every task composing the trace, and at the end of the analysis, every discrepancy detected, like missing required tokens to execute a task in the order defined by the trace, as well as remaining tokens in the model's places, with the exception of the final place[11], is considered to determine how much deviation there is between the observed behaviour and the expected behaviour of a model. Naturally, when the trace is a proper trace of the model, then no discrepancies are detected.

[9] With the term process mining, we refer to techniques capable of distilling a business process model fitting a given event log of traces. Such techniques can be used to construct a process model from scratch, or to adapt existing process models to properly fit the actual observed behaviour. While this is another relevant discipline related to business process models, we do not delve in its details in this chapter as it is only marginally related to the problem of proving regulatory compliance.

[10] As a reminder, in a Petri Net a task, also referred to as a transition, can be executed if every place in its precondition set contains a token.

[11] A workflow model based on Petri Nets, is considered to be sound if it is executable without leaving tokens within its internal places after the execution is concluded.

### Data Driven Conformance

Understanding whether an execution, composed by simple sequence of tasks, belongs to a process model is important. Considering only the execution sequence may be in fact not enough to properly measure conformance. The execution of business process models involves additional factors, such as the state of the execution, in other words the data corresponding to the execution. This is represented in the present chapter as the process' states and associated to the execution of the tasks of the process in their corresponding traces.

Efforts towards this taking into account data while measuring conformance has already been made, as for instance by De Leoni et al. [16], which adopt an approach using $A^*$ to calculate the alignment[12] between the trace being evaluated and the process model, which allows to evaluate data and resources in addition to the execution order of the tasks in a trace.

While De Leoni et al. [16] claim their approach to be sub-linear in time with respect to the size of the log and model being evaluated, it must be considered that being $A^*$ heuristic based, hence trying to optimise the search space being investigated by smartly reducing it, there is always the possibility that part of the search space containing the optimal solution (or a solution) for the problem to be discarded. In general, conformance verification procedures are solvable in time polynomial with respect to the size of the problem, as for instance the approach proposed by Sun and Su [64], based on solving syntactic characterisations of some subclasses of *DecSerFlow* constraints[13].

### Conformance and Legal Requirements

In addition to data, sometimes it is necessary to verify whether the actual behaviour of an organisation (its logs) conforms with the legal requirements in place. While business process regulatory compliance represents a way to indirectly verify this through its pairwise use with conformance checking, sometimes a more direct approach is desirable in particular to determine if the actual execution of instances of the process do not violate legal requirements. In this case, we can speak of run-time compliance (if checked while a process instance is executed) or auditing (if it is a post-mortem analysis of the instances). Generally, run-time compliance with the legal requirements can be handled with the same techniques adopted for design-time compliance. However, there are a few differences: the first is about the data to be

[12] Alignment is a measure related to conformance, and it measure how close, aligned, is a given execution with the possible executions of a given process model.

[13] DecSerFlow is an extensible language, which stands for: *declarative service flow* language. It can be used to specify and monitor service flows, in addition to verify their conformance.

used for the annotations. At design time, we do not know the actual value for the data (and most approaches assume annotations expressed as propositional/boolean variables) and those values must be instantiated by the actual value occurring in the instances of the processes. After the data has been instantiated, the theoretical complexity of auditing is reduced to the complexity of the underlying logic/framework given that the number of states is linear and it is determined by the number of entries in the process log (and every instance corresponds to a single trace of the process model). For run-time compliance, the issue is whether one is interested to check if the current instance at the then current task is compliant, in which case the complexity is the same as the complexity of auditing (given that the problem is reduced to the case of a single trace); alternatively, one can check if it is possible to terminate the current instance with no violations or all possible terminations are compliant. Clearly, both cases reduce to the situation where we have to determine if a sub-process model is compliant; in particular the (sub-)process model obtained by the original process model where we delete all paths not passing from the current task, and identifying the start of the (sub-)process model with the current task. Hence, the problem of determining if there is a compliant termination is reduced to the case of partial compliance, and all possible termination are compliant to full compliance.

## 6.2 Normative Reasoning

Finally, after having discussed the relation between business process compliance and conformance, we discuss the further relations with the area of *Normative Reasoning*, tasked about reasoning about norms and normative concepts, and how they affect various type of environment. While many different formalisms/logics/frameworks have been proposed for normative reasoning, ranging from various deontic logics [22], different systems of non-monotonic reasoning [56, 40, 25], event calculus [61], Input/Output logic [50] and various forms of expert systems and AI and Law systems [5], the study of the complexity of legal and normative reasoning has been largely neglected. Despite this, the complexity classes for the different approach is well understood: modal logic [46, 36] for deontic logics, though, with almost no results for conditional and dyadic deontic logic[14]; complexity of default logic, argumentation [6] for non-monotonic reasoning, and ad hoc results for event calculus [7]. Practically, all approaches are **NP**-complete or with higher computational complexity. In what follows we briefly discuss some notable exceptions.

Some of the work dedicated to the complexity of normative reasoning concerns

[14]In general conditional logics received much less attention than their modal counterparts, for some complexity results see [3].

the investigation of the complexity of Input/Output logic [62, 63] where the complexity of some I/O variants is investigated also in connection to the representation of norms (including I/O with permissive norms; however, the work is dedicated to the study the complexity of logics, and not to normative reasoning problems. Most the problems (e.g., consistency, fulfilment) discussed by Sun and co-workers are not tractable. For example, consistency is some of the basic I/O logic (simple-minded I/O logic) is **NP**-complete, and fulfillment is **coNP**-complete, with higher complexity for constrained I/O logics.

The second area of research related to computational complexity and normative reasoning is the work on Defeasible Deontic Logic. Contrary to the work reported above the Defeasible Deontic Logic (also known as PCL[15] [31]), is computationally feasible. [29] and [26] extended the result by [49] proving that the extensions of Defeasible Logic with deontic operators and violation operator of [27] is still computationally feasible, and the extension of a defeasible theory can be computed in **P**, more specifically, the problem is linear in the size of the theory, where the size of a theory is given by the number of rules and instances of literals in the theory. The result was further extended to included permission and weak permissions [25]. Similarly, [33] prove that temporalising PCL, allowing for explicit deadlines, and compensation does not increase the complexity of the logic, and the temporal extension can still be computed in time linear to the size of the theory, where, in this case, the size depends also on the distinct instants of time explicitly appearing in the given theory; this extends the result in [32].

[26] applied the work to the execution of business contracts, thus the performance of a contract can be executed in linear time. Furthermore, they discussed the issue of comparing contracts and proposed a normalisation procedure to this end. However, they did not investigate the complexity of the normalisation problem. [33] address this issue in the context of a temporal extension of the logic, and while they do not give a complexity result they provide an (exponential) upper-bound. Accordingly, they conjecture the problem to be computationally hard but argue that it might not be a problem for real life applications since the problematic cases are typically not very frequent and limited in the number of parameters.

In [28] the logic was used for modelling agents, in particular to the modelling of the so called social agents, i.e., agents where there is a conflict between one of their intention and a norm, they give an higher preference to the norm, dropping thus the conflicting intention. However, Governatori and Rotolo shown that there are situations where, even for social agents, adhering to the agent plan ends up in a non-compliant situation. Accordingly, the restoring sociality problem is to identify

[15] Process Compliance Logic.

a set of agent's intentions to drop to prevent the agent's plan to be non-compliant. Governatori and Rotolo proved that when norms and agents are represented using Defeasible Deontic Logic the restoring sociality problem is **NP**-complete. The logic employed in [28] can be used to model business processes, after all, one can consider a business process as the set of traces, where each trace is a sequence of task, where the annotations corresponds to the effects of the tasks, and the states include the preconditions of the tasks. Hence, the plan library of an agent can be understood as business process (or a set of business processes), where the intentions and the facts of a theory determine what are the traces/processes/sub-processes to be executed. Accordingly, the restoring sociality problem can be seen as a special case to recovery from non-compliance for business processes (in the **nL-** space).

# 7 Summary and Open Problems

In this chapter we focused our attention on the computational complexity of proving regulatory compliance of business process models. We first describe the variants of the problem by reusing the same classification used by Colombo Tosatto et al. [11]. After discussing the existing computational complexity results, we moved to discussing neighbouring areas which still require much investigation, in order to understand the computational complexity of a broader spectrum of the variants of the problem.

Finally, we conclude this chapter by listing the open problems identified.

## 7.1 Proving Partial Compliance for the Variant 1L-

This particular variant of the problem of proving regulatory compliance, identified by Colombo Tosatto et al. [11], involves verifying whether a structured business process model is compliant with a single conditional obligation whose parameters are represented by using propositional literals.

While Colombo Tosatto et al. proposed a conjecture, reported in Conjecture 2, stating that the variant **1L-** should be able to be solved in time polynomial with respect to the size of the problem, we proposed in the present paper the opposite conjecture in Conjecture 1. However, no formal proof have been provided to show that the conjecture is correct. Therefore, proving that either **1L-** belongs to the computational complexity class **NP-c**, or to the class **P**, remains a problem to be solved.

**Conjecture 2** (**1L-** is in **P**)**.** *We currently have no information about the computational complexity of **1L-**. That is, we cannot infer its belonging to a computational*

*complexity class in a similar way as for **nG+**, as in this case the simpler variant (**1G-**) is in **P**.*

*Our conjecture is that the computational complexity of **1L-** is in **P**. We have proven that moving from **-** to **+**, or from **1** to **n**, definitely brings the complexity into **NP-c**. In general, solutions tackling such variants have to explore the entire set of possible executions in the worst case scenarios, which precludes efficient solutions. Despite moving from **G** to **L** seems to definitely increase the complexity, we strongly believe that it does not influence the computational complexity of the problem enough to move it into **NP-c**, and polynomial solutions are still possible.*

## 7.2 Proving Regulatory Compliance of Unstructured Process Models

While the computational complexity of proving regulatory compliance of structured business process models has been extensively studied, the same cannot be said for unstructured process models. Therefore, a thorough analysis of the computational complexity for these unstructured variants of the problem is still an open problem. Considering that unstructured process models become structurally very similar to petri nets, investigating this similarity can be the initial step towards this analysis.

Moreover, as currently for structured process models, the variants of the problem are identified solely on adopting different properties of the regulatory framework being used to check regulatory compliance, identifying a set of structural properties of the models would allow to identify additional variants of the problem on the top of the ones already identified. The advantage in this case could be to allow a *divide and conquer* approach for studying the computational complexity of the problem, and possibly identifying simpler and harder versions of the problem.

Finally, given the relations with petri nets, correlating the structural properties of the variants of the compliance problems involving unstructured processes with known issues of petri nets (i.e., the undecidability of coloured Petri Nets) may be able to provide interesting results, which can potentially benefits both problems.

## 7.3 The Impact of Loops

Loops can be included in the business process models to improve the expressivity of the problem, allowing the repeated execution of tasks. Despite the obvious usefulness of including these type of constructs in process models, how much their inclusion increases the computational complexity of the problem in either structured and unstructured variants has not yet been studied.

As discussed in this chapter, introducing loops in business process models brings

them closer to *complete Turing machines*, which also leads to the inheritance of the problems affecting them (i.e., undecidability due to the *halting problem*). Therefore, as mentioned while discussing the open problems related to proving compliance of unstructured processes, identifying a set of properties allowing to identify a relevant number of variants can help in the computational complexity analysis, as well as allowing to identify, if it exists, the line between these problem's variants separating the ones which can be considered complete Turing machines from the ones which cannot.

### 7.4 Conformance

While only tangentially related to the problem of proving regulatory compliance of business process models, the solutions for these problem can be used in combination to ensure stronger properties. While the computational complexity of verifying conformance has been shown to not be a big obstacle for the problems considered, their scope can be definitely broadened to cover more interesting variants, such as for instance considering the regulatory requirements provided by a regulatory framework while conformance is being verified, where the additional challenges are related to the data (how to ensure that the "concrete" data at run-time/auditing correspond to the "abstract" data specified in the annotations of the tasks. In addition, normative requirements can span across multiple instances of the process (and multiple processes) and, in general, the instances are not synchronised.

## References

[1] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M.P. van der Aalst. Towards robust conformance checking. In *International Conference on Business Process Management*, pages 122–133. Springer, 2010.

[2] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[3] Régis Alenda, Nicola Olivetti, and Gian Luca Pozzato. Nested sequent calculi for normal conditional logics. *Journal of Logic and Computation*, 26(1):7–50, 2016.

[4] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using BPMN-Q and temporal logic. In *International Conference on Business Process Management*, pages 326–341. Springer, 2008.

[5] Trevor Bench-Capon, Michał Araszkiewicz, Kevin Ashley, Katie Atkinson, Floris Bex, Filipe Borges, Daniele Bourcier, Paul Bourgine, Jack G. Conrad, Enrico Francesconi, et al. A history of AI and Law in 50 papers: 25 years of the international conference on AI and Law. *Artificial Intelligence and Law*, 20(3):215–319, 2012.

[6] Marco Cadoli and Marco Schaerf. A survey of complexity results for non-monotonic logics. *The Journal of Logic Programming*, 17(2-4):127–160, 1993.

[7] Luca Chittaro and Alberto Montanari. Efficient temporal reasoning in the cached event calculus. *Computational Intelligence*, 12(3):359–382, 1996.

[8] Yongsun Choi and J. Leon Zhao. Decomposition-based verification of cyclic workflows. In *Automated Technology for Verification and Analysis*, pages 84–98. Springer, 2005.

[9] Silvano Colombo Tosatto. *Proving Regulatory Compliance: Business Processes, Logic, Complexity*. PhD thesis, University of Luxembourg and Università di Torino, 2015.

[10] Silvano Colombo Tosatto, Guido Governatori, and Pierre Kelsen. Business process regulatory compliance is hard. *IEEE Transactions on Services Computing*, 8(6):958–970, 2015.

[11] Silvano Colombo Tosatto, Guido Governatori, and Nick R.T.P. van Beest. Checking regulatory compliance: Will we live to see it? 09 2019.

[12] Silvano Colombo Tosatto, Guido Governatori, and Nick R.T.P. van Beest. Business process full compliance with respect to a set of conditional obligation in polynomial time. https://arxiv.org/abs/2001.10148, 1 2020.

[13] Silvano Colombo Tosatto, Guido Governatori, and Nick R.T.P. van Beest. Proving regulatory compliance: A comprehensive computational complexity analysis. *TBD*, 2021 forthcoming.

[14] Silvano Colombo Tosatto, Pierre Kelsen, Qin Ma, Marwane El Kharbili, Guido Governatori, and Leendert W.N. van der Torre. Algorithms for tractable compliance problems. *Frontiers of Computer Science*, 9(1):55–74, 2015.

[15] Byron Cook, Alexey Gotsman, Andreas Podelski, Andrey Rybalchenko, and Moshe Y. Vardi. Proving that programs eventually do something good. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, pages 265–276, New York, NY, USA, 2007. ACM.

[16] Massimiliano de Leoni, Wil M.P. van der Aalst, and Boudewijn F. van Dongen. Data- and resource-aware conformance checking of business processes. In Witold Abramowicz, Dalia Kriksciuniene, and Virgilijus Sakalauskas, editors, *Business Information Systems*, pages 48–59, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[17] Ivaylo Dobrikov, Michael Leuschel, and Daniel Plagge. Ltl. pages 204–211, 07.

[18] Amal Elgammal, Oktay Turetken, Willem-Jan van den Heuvel, and Mike Papazoglou. Formalizing and appling compliance patterns for business process compliance. *Software & Systems Modeling*, 15(1):119–146, 2016.

[19] Javier Esparza. On the decidability of model checking for several $\mu$-calculi and petri nets. In Sophie Tison, editor, *Trees in Algebra and Programming — CAAP'94*, pages 115–129, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[20] Sven Feja, Andreas Speck, and Elke PulvermÃijller. Business process verification. In *GI Jahrestagung*, pages 4037–4051, 2009.

[21] Nissim Francez. *Fairness*. Springer-Verlag, Berlin, Heidelberg, 1986.

[22] Dov Gabbay, Jeff Horty, Xavier Parent, Ron van der Meyden, and Leendert W.N.

van der Torre, editors. *Handbook of deontic logic and normative systems*. College Publication, 2013.

[23] Aditya Ghose and George Koliadis. Auditing business process compliance. In *ICSOC 2007*, pages 169–180, 2007.

[24] Guido Governatori. The Regorous approach to process compliance. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, pages 33–40. IEEE, 2015.

[25] Guido Governatori, Francesco Olivieri, Antonino Rotolo, and Simone Scannapieco. Computing strong and weak permissions in defeasible logic. *Journal of Philosophical Logic*, 42(6):799–829, 2013.

[26] Guido Governatori and Duy Hoang Pham. DR-CONTRACT: an architecture for e-contracts in defeasible logic. *International Journal of Business Process Integration and Management*, 4(3):187–199, 2009.

[27] Guido Governatori and Antonino Rotolo. Logic of violations: A Gentzen system for reasoning with contrary-to-duty obligations. *Australasian Journal of Logic*, 4:193–215, 2006.

[28] Guido Governatori and Antonino Rotolo. BIO logical agents: Norms, beliefs, intentions in defeasible logic. *Journal of Autonomous Agents and Multi Agent Systems*, 17(1):36–69, 2008.

[29] Guido Governatori and Antonino Rotolo. A computational framework for institutional agency. *Artificial Intelligence and Law*, 16(1):25–52, 2008.

[30] Guido Governatori and Antonino Rotolo. A conceptually rich model of business process compliance. In Sebastian Link and Aditya Ghose, editors, *7th Asia-Pacific Conference on Conceptual Modelling*, volume 110 of *CRPIT*, pages 3–12. ACS, 2010.

[31] Guido Governatori and Antonino Rotolo. Norm compliance in business process modeling. In *Proceedings of the 4th International Web Rule Symposium: Research Based and Industry Focused (RuleML 2010)*, volume 6403 of *LNCS*, pages 194–209. Springer, 2010.

[32] Guido Governatori and Antonino Rotolo. Computing temporal defeasible logic. In *RuleML 2013*, pages 114–128, 2013.

[33] Guido Governatori and Antonino Rotolo. Time and compensation mechanisms in checking legal compliance. *Journal of Applied Logics – IFCoLog Journal of Logics and their Applications*, 6(5):817–847, 2019.

[34] Heerko Groefsema, Nick R.T.P van Beest, and Marco Aiello. A formal model for compliance verification of service compositions. *IEEE Transactions on Services Computing*, 11(3):466–479, 2018.

[35] Stephan Haarmann, Kimon Batoulis, and Mathias Weske. Compliance checking for decision-aware process models. In *International Conference on Business Process Management*, pages 494–506. Springer, 2018.

[36] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial intelligence*, 54(3):319–379, 1992.

[37] Mustafa Hashmi, Guido Governatori, Ho-Pun Lam, and Moe Wynn. Are we done with business process compliance: State-of-the-art and challenges ahead. *Knowledge and Information Systems*, 01 2018.

[38] Mustafa Hashmi, Guido Governatori, and Moe Thandar Wynn. Normative requirements for regulatory compliance: An abstract formal framework. *Information Systems Frontiers*, 18(3):429–455, 2016.

[39] Jörg Hoffmann, Ingo Weber, and Guido Governatori. On compliance checking for clausal constraints in annotated process models. *Information Systems Frontiers*, 14(2):155–177, 2012.

[40] John F. Horty. Deontic logic as founded on nonmonotonic logic. *Annals of Mathematics and Artificial Intelligence*, 9(1-2):69–91, 1993.

[41] Conrad Indiono, Walid Fdhila, and Stefanie Rinderle-Ma. Evolution of instance-spanning constraints in process aware information systems. In *OTM Confederated International Conference "On the Move to Meaningful Internet Systems"*, pages 298–317. Springer, 2018.

[42] Gerhard Keller and Thomas Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1998.

[43] Yonit Kesten, Amir Pnueli, Li-On Raviv, and Elad Shahar. Model checking with strong fairness. *Formal Methods in System Design*, 28(1):57–84, Jan 2006.

[44] Bartek Kiepuszewski, Arthur H.M. ter Hofstede, and Christoph Bussler. On structured workflow modelling. In *Proceedings of the 12th International Conference on Advanced Information Systems Engineering*, CAiSE '00, pages 431–445, London, UK, UK, 2000. Springer-Verlag.

[45] David Knuplesch, Linh Thao Ly, Stefanie Rinderle-Ma, Holger Pfeifer, and Peter Dadam. On enabling data-aware compliance checking of business process models. In *International Conference on Conceptual Modeling*, pages 332–346. Springer, 2010.

[46] Richard E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM journal on computing*, 6(3):467–480, 1977.

[47] Niels Lohmann, Eric Verbeek, and Remco Dijkman. *Petri Net Transformations for Business Processes – A Survey*, pages 46–63. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[48] Ruopeng Lu, Shazia Sadiq, and Guido Governatori. Compliance aware business process design. In *International Conference on Business Process Management*, pages 120–131. Springer, 2007.

[49] Michael J. Maher. Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming*, 1(6):691–711, 2001.

[50] David Makinson and Leendert W.N. van der Torre. Permission from an input/output perspective. *Journal of philosophical logic*, 32(4):391–416, 2003.

[51] Mediatrix Makungu, Michel Barbeau, and Richard St-Denis. Synthesis of controllers of processes modeled as colored petri nets. *Discrete Event Dynamic Systems*, 9:147–169, 05 1999.

[52] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[53] Shin Nakajima. Verification of Web service flows with model-checking techniques. In *Proceedings of First International Symposium on Cyber Worlds*, pages 378–385, 2002.

[54] Artem Polyvyanyy, Luciano García-Bañuelos, Dirk Fahland, and Mathias Weske. Maximal structuring of acyclic process models. *The Computer Journal*, 57, 01 2014.

[55] Artem Polyvyanyy, Luciano García-Ba nuelos, and Marlon Dumas. Structuring acyclic process models. *Information Systems*, 37(6):518 – 538, 2012.

[56] Henry Prakken and Giovanni Sartor. Law and logic: A review from an argumentation perspective. *Artificial Intelligence*, 227:214–245, 2015.

[57] PWC. *2017 Risk and Compliance Benchmarking Survey*, 2017.

[58] Anne Rozinat and Wil M.P. Van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

[59] Shazia Sadiq, Guido Governatori, and Kioumars Namiri. Modeling control objectives for business process compliance. In *International conference on business process management*, pages 149–164. Springer, 2007.

[60] Shazia Sadiq, Maria E. Orlowska, and Wasim Sadiq. Specification and validation of process constraints for flexible workflows. *Information System*, 30(5):349–378, 2005.

[61] Marek J. Sergot, Fariba Sadri, Robert A. Kowalski, Frank Kriwaczek, Peter Hammond, and H. Terese Cory. The british nationality act as a logic program. *Communications of the ACM*, 29(5):370–386, 1986.

[62] Xin Sun and Diego Agustín Ambrossio. Computational complexity of input/output logic. In Antonis Bikakis and Xianghan Zheng, editors, *Multi-disciplinary Trends in Artificial Intelligence - 9th International Workshop, MIWAI 2015, Fuzhou, China, November 13-15, 2015, Proceedings*, volume 9426 of *Lecture Notes in Computer Science*, pages 72–79. Springer, 2015.

[63] Xin Sun and Livio Robaldo. On the complexity of input/output logic. *Journal of Applied Logic*, 25:69–88, 2017.

[64] Yutian Sun and Jianwen Su. Conformance for decserflow constraints. In Xavier Franch, Aditya K. Ghose, Grace A. Lewis, and Sami Bhiri, editors, *Service-Oriented Computing*, pages 139–153, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[65] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 01 1937.

[66] Wil M.P. van der Aalst. *A class of Petri nets for modeling and analyzing business processes*. Computing science reports. Technische Universiteit Eindhoven, 1995.

[67] Wil M.P. van der Aalst. Verification of workflow nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, ICATPN '97, pages 407–426, London, UK, 1997. Springer-Verlag.

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

[69] Wil M.P. van der Aalst. Distributed process discovery and conformance checking. In Juan de Lara and Andrea Zisman, editors, *Fundamental Approaches to Software Engineering*, pages 1–25, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[70] Wil M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer, Heidelberg, 2 edition, 2016.

 Received 1 February 2021