UNIVERSITÉ DU LUXEMBOURG
PhD-FSTC-2015-2
The Faculty of Science,
Technology and Communication

Universitá di Torino
Dipartimento di Informatica

DISSERTATION

Defense held on 15/01/2015 in Luxembourg
to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG
EN INFORMATIQUE
AND
DOTTORE DELL'UNIVERSITÁ DI TORINO
IN INFORMATICA

Proving Regulatory Compliance:
Business Processes, Logic, Complexity

by

Silvano Colombo Tosatto
Born on 27/04/1984 in Torino (Italy)

Dissertation defense committee

Dr Leendert van der Torre, Dissertation Supervisor
Professor, Université du Luxembourg

Dr Guido Boella, Dissertation Supervisor
Professor, Universitá di Torino

Dr Stefanie Rinderle-Ma, Chairman
Universität Wien

Dr Marco Montali, Vice-Chairman
Senior Researcher, Free University of Bolzen-Bolzano

Dr Guido Governatori, Member
Senior Principal Researcher, Australias Information and Communication Technology Research Center of Excellence (NICTA) and Queensland University of Technology

Acknowledgements

This thesis is the result of research conducted during my period as a PhD candidate, and would not have been possible without the help of many others whom I would like to thank here. First of all, my two supervisors Prof. Guido Boella and Prof. Leendert van der Torre, have my sincerest gratitude for their support, especially during my initial years as a PhD student.

Two additional important people that have greatly helped me during my PhD studies are Prof. Pierre Kelsen and Dr. Guido Governatori, with whom I have had the pleasure of working several times. On those occasions, their technical expertise was invaluable and gave me the opportunity to learn a great deal from them.

Finally, I wish to thank all the people of Torino and Luxembourg I have had the chance to interact and work with during all these years, both within and outside the academic environment.

Abstract. *The problem of proving regulatory compliance of a business process model is composed of two main elements. One of these elements is the business process model, which provides a formal compact description of the available executions capable of achieving a given business objective. The other element is the regulatory framework that the business process must follow, describing the compliance requirements given by the law or by a company's own internal regulations. The problem consists of verifying whether a given business process model is compliant with the regulatory framework, which is carried out by verifying whether the executions of the model comply with the requirements of the regulatory framework.*

Solutions to prove the regulatory compliance of business processes have been already proposed in the past. Some solutions disregard the computational complexity aspect of the problem, while other solutions either solve a simplified version of the problem efficiently or provide approximate solutions for the general one. However, none of these solutions have formally studied the computational complexity of the problem of proving regulatory compliance. This thesis addresses that issue, showing in addition why efficient solutions of the general problem are not possible. In particular I study the computational complexity of a problem of proving regulatory compliance whose regulatory framework is defined using conditional obligations. The approach I adopt to represent the compliance requirement is semantically similar to some of the existing solutions proposed by other researchers, such as van der Aalst and many others, who adopts linear temporal logic over finite traces, or different variants of such temporal logic, to define the compliance requirements. More precisely, the approach used in the present thesis adopts propositional logic as base logic, and defines the semantics of the regulatory framework in a similar way as Process Compliance Logic introduced by Governatori and Rotolo.

The study of the computational complexity of the problem is approached by dividing it in sub-classes and then combining their analysis to obtain the result for the target problem. The division is done according to three features of the regulatory framework. These features define whether the framework is composed of a single or a set of obligations, whether the obligations are conditional and whether violations can be compensated. These features can be omitted to identify simpler sub-classes of the problem. After having identified the different sub-classes of the problem, I study the computational complexity of some of these sub-classes and combine the results obtained to identify the computational complexity of the general problem tackled in the present thesis, where each of the three features identified are used to describe the compliance requirements.

The results of the computational complexity analysis show that proving the existence of an execution of a business process compliant with the regulatory framework is an ***NP****-complete problem. Differently proving that for all of the executions of a business process model, they are either compliant or not with the regulatory framework, is a co****NP****-complete problem. The results show that combining the two elements composing the problem of proving regulatory compliance, the process model and the regulatory framework, which are tractable when considered individually, leads to an intractable problem. In addition to the computational complexity results, the analysis provided in the thesis has also shown that tractable sub-*

classes of the problem, where the computational complexity is at most polynomial with respect to the size of the input, can be obtained by trivialising the expressivity of either one of the elements composing the problem. However the expressivity of these sub-classes is limited. Thus I identify a different tractable sub-class of the problem by weakening the expressivity of both elements composing the problem, but where each element is not trivialised as for the other sub-classes. Whether the sub-class identified can be considered expressive is arguable, however it represents a first step towards identifying a sub-class of the problem being both tractable and expressive, taking into account the limitation of employing propositional logic as base logic.

Contents

0.1 Tables of Notation

The following tables provide a summarised description of the notation appearing in the present thesis and referencing to the corresponding definition when possible or where they first appear.

Name	Symbol	Description	Reference
Process Block	B	A process block is the building block constituting the processes.	Definition 1
Process	P	A structured process model composed of a main process block B.	Definition 2
Annotated Process	(P, ann)	A structured process model with associated an annotation function ann describing the effects of executing its tasks	Definition 8
Execution	ϵ	An execution of a business process model corresponding to one of its possible serialisations. The set of finite serialisations of a process model is denoted as $\Sigma(P)$.	Definition 5
Process State	σ	The process state describes the state of affairs at a given point in time of the execution of a business process and is represented as a consistent set of literals.	Definition 9
Trace	θ	A trace of a business process model, describing the evolution of the process' state in correspondence of the tasks being executed. The set of traces of an annotated process is denoted as $\Theta(P, \mathsf{ann})$. For each execution there exists a corresponding trace.	Definition 11

Name	Symbol	Description	Reference
Obligation	$\mathbf{\Theta}$	An obligation, either local or global, describes a compliance requirement of a regulatory framework.	Definition 12 Definition 17
Global Obligation	$\mathcal{O}^t\langle\varphi\rangle$	A global obligation, which activation period spans for the entirety of a process execution, where t defines its type and φ describes the requirement through a propositional formula	Definition 12
Local Obligation	$\mathcal{O}^t\langle\varphi_l, \varphi_d, \varphi_c\rangle$	A local obligation, which activation period is determined by both lifeline condition (φ_l) and deadline condition (φ_d), where t defines its type and φ_c describes the requirement. φ_l, φ_d and φ_c are described using propositional formulae.	Definition 17
Set of Obligations	$\circledcirc$	A set of obligations, either local or global.	Definition 26
State Fulfilment	$\models$	This boolean binary operator, determines whether either a literal or a propositional formula is fulfilled by a process state.	Definition 13 Definition 44
Compensation Operator	$\otimes$	The compensation operator is a binary operator defining when an obligation can be used to compensate the violation of another one as follows: $\mathbf{\Theta}_1 \otimes \mathbf{\Theta}_2$. In this case $\mathbf{\Theta}_2$ is used to provide a compensatory action for the violations of $\mathbf{\Theta}_1$.	Definition 28
Compensation Chain	$\lightning$	A compensation chain refers to a sequence of obligations joined by the compensation operator.	Definition 28

Chapter 1

Introduction

1.1 The Problem of Proving Regulatory Compliance

The term *regulatory compliance* describes the effort of organisations and business companies to comply with the regulations and laws governing their business sector. The effort of proving that a company complies with the regulations and laws presents different challenges. Among these challenges, as pointed out by Boella et al. [15], one consists of the fact that regulations are designed to be as general as possible in order to be able to cover a wide range of possible scenarios. By contrast the processes of companies, describing how they handle their business, are tailored to deal with specific situations. Therefore it becomes necessary to interpret the laws and regulations before verifying whether the processes of a company comply with them. Another challenge derives from the fact that regulatory compliance involves experts from different disciplines, such as compliance officers, legal experts, process designers, etc.

In the present thesis I narrow down the scope and tackle a subset of the whole problem of proving regulatory compliance. More precisely, I tackle the problem of proving regulatory compliance adopting a computer science approach. The business processes of a company are represented using the *Business Process Modelling and Notation 2.0 (BPMN2.0)*[1] and the laws and regulations are formally represented as a regulatory framework composed of obligations based on conditional rules.

By narrowing the scope, many of the challenges pointed out by Boella et al. [15] are not addressed or tackled in the present thesis. The interpretation of the laws and regulations is given, and it is assumed that it corresponds to the rule-based obligations composing the regulatory framework. The problem is tackled from a computer science perspective, disregarding in this thesis the other points of view, such as for instance the one of compliance officers. One of the challenges pointed out by Boella et al. consists of the fact that compliance management takes place in a dynamic environment, where both

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

elements composing the problem, the law and the processes of a company, can evolve over time. Thus dealing with this dynamic aspect is also an important part of the problem. In the present work the problem is studied in a static environment where the regulations and process models do not evolve during the analysis.

This thesis focuses on the problem of formally proving whether a business process model is compliant with a regulatory framework specifying the compliance requirements. This problem, known as the problem of proving regulatory compliance, is composed of two components, the business process model describing the procedures available to a business company for achieving a business objective, where each of these procedure is represented as a possible execution of the model, and the regulatory framework representing the compliance requirements that the company must follow while pursuing its business objective, which are reflected as constraints that must be respected by the possible executions of the business process model, corresponding to the company's processes.

Example 1 (Car Company)**.** *Consider an automotive company whose business objective is to build cars. The processes of this company consist of its possible ways of building a car. However the company may not be free to build a car in every possible way but it may be constrained by regulations, like for instance safety regulations issued by the state and aiming at ensuring some safety protocols for the cars, or even regulations internal to the company itself aiming at minimising production costs.*

Example 1 illustrates an instance of the regulatory compliance problem. In this case an automotive company has to prove to be following the governing regulations while building cars. The two components of the problem appear in the example as the processes of the company, which can be modelled in a business process model and the regulations, both internal of the company and external from the state, composing the regulatory framework.

1.1.1 Business Process Models

To represent the business process models I use in the present thesis a notation similar to *Business Process Modelling and Notation 2.0 (BPMN2.0)*. This notation is capable of representing business processes as well as workflows, and has some similarities with *Petri nets*, which are mathematical modelling languages for the description of distributed systems. Petri nets are directed bipartite graphs where the nodes belonging to the two partitions have different functions. A partition of these nodes represents the *places*, which are the states in which the net can be, and the other partition represents the transitions, the events that can occur between the states. Petri nets use *tokens* that travel between the states passing through the transitions. The notation I use to model business processes in this thesis borrows the graphical notation from *BPMN2.0*, which I use to represent the activities composing the processes. In addition to the graphical notation I adopt a subset of the flow operators used by *BPMN2.0* to identify the possible executions of a model, to which I refer as coordinators in the present thesis. More precisely I use three types of coordinators: one

to define a strict order between the activities, one to define when activities are mutually exclusive and one to represent the absence of ordering constraints between activities. An additional restriction is that the processes need to be block-structured. Thus process blocks, identified by activities contained between two coordinators of the same type, require to be nested properly, meaning that a block that starts inside another must also end inside it. Another restriction of the formalism used in the present thesis with respect to *BPMN2.0* is that additional features, such as resources and events, are not represented.

The business process models I am using can be seen as simplified petri nets where the places correspond to the activities, the transitions correspond to the coordinators and a single token is used to navigate the model. Such restrictions do not allow a concurrent execution of the activities included in the model, for instance even when no ordering constraints are given for a set of activities, a valid execution corresponds to a linear order of such activities. A similar approach has been also adopted by van der Aalst [79], studying the problem of interorganisational workflows, where petri nets are used to model the workflows of the individual organisations. Another similar approach has been adopted by Governatori [32], proposing the *Regorous* architecture, a tool to provide support to prove regulatory compliance of business processes, which are expressed using a similar notation to the one used here.

Example 2 (Car Company, continued)**.** *Considering Example 1, we can represent the various processes of how an automotive company could build a car using a business process model. This model would include the different activities, like for instance building the chassis, building the engine and assembling the whole car. Some of these activities are independent like building the chassis and the engine, in the sense that it does not matter which one is executed first. However, some activities require that others have been already executed, like assembling the car requires that activities like building the chassis and the engine have been done already. Finally some of these activities are mutually exclusive, like building a fuel or a diesel engine, since a car requires only one engine.*

This example abstractly describes the elements composing a business process model that can be later proven to be compliant with a set of legal requirements. This model requires to include the different activities that can be executed to achieve the business objectives and the coordinators constraining the executability of such activities. The models presented in this thesis deal with three types of constraints: independence when the order of execution of some activities does not matter, temporal dependence when it does and mutual exclusive activities when executing a set of activities excludes the execution of another set of them.

1.1.2 Regulatory Framework

The regulatory framework is the second component of the problem of proving regulatory compliance, it describes the regulation that must be followed by the company pursuing its business objectives. These regulations define the acceptable ways of achieving these

objectives and are expressed in the regulatory framework through obligations. The obligations determine which are the acceptable executions of a process model by defining which activities must be executed and which instead must be avoided. These conditions are determined dependently on the state of the process, which is in turn determined by the activities already executed.

To represent the obligations in the regulatory framework I am using propositional logic as object language to represent the elements of an obligation, which are in turn defined using a language and semantics similar to the one defined by Governatori and Rotolo [36]. The formalism proposed by Governatori and Rotolo allows to express the desired features for the regulatory framework, such as identifying when an obligation is active and define possible compensations. The formalism's semantics uses a set of conditions to define the obligations that one needs to comply with. These conditions include lifelines and deadlines, described as propositional formulae, to specify when these obligations need to be complied with. The obligations can be of different types and each type has its specific semantics. Another feature used in the regulatory framework in the present thesis are compensations, which provide a way of coping with compliance breaches. In other words if it is not possible to comply with an obligation, the compensation provides an additional obligation to which the processes must comply in order to be considered compliant.

Example 3 (Car Company, continued)**.** *Considering again Example 1, the set of obligations can be composed of both the internal regulations of the company and the state regulations. For instance, assuming that the activities that can be done have a cost, an internal regulation could try to minimise the costs of building a car by setting a threshold. Differently a state regulation could for instance try to regulate CO_2 emissions by prohibiting certain types of engines.*

When proving the regulatory compliance of a business process model, regulations similar to the ones illustrated in Example 3 must be complied with. Therefore it may be the case that even if certain processes are available to a company to achieve their business objectives, they should not be adopted since they may violate some of the obligations describing the regulations. Considering again the example, we can assume that an automotive company may have the possibility of using old engines to produce its cars in order to save on the costs. However, assuming also that these old engines are not up to par with the environmental standards imposed by the regulations, these processes involving these old engines are not compliant with the regulations and should not be used to achieve the business objective of building a car.

1.1.3 Proving Compliance

The problem of proving regulatory compliance consists of verifying whether a business process model of a company, which contains the processes able to achieve a given business objective, is compliant with the obligations describing the governing regulations. In order

to formally prove the compliance of a business process model, both the model and the obligations need to be formally defined.

The business process models are formally defined using a subset of the *BPMN2.0* language, where the semantics of the tasks, representing the activities, and the semantics of the coordinators are able to identify which company's processes are embedded in the model. I opted for using a subset of *BPMN2.0*, where I can represent the presence and absence of ordering constraints between the activities and their mutual exclusion. These features allow to models some of the aspects of real problems, such as the one illustrated in Example 2. However, I am not claiming that the representation used is capable to represent every relevant feature of a problem, in fact the representation used avoids more complex features like for instance resource constraints and parallel executions of the activities. The representation used allows to focus on a sub-class of the problem of proving compliance where some of the possible constraints are disregarded. Even though the problem studied is a simplified version, I show in the present thesis that it is already computationally complex.

Therefore the starting point of the analysis of proving compliance is always a business process model, from which the single processes can be identified as possible executions of it. An execution of a process model consists of a sequence of some of the available activities. The effects of executing these activities are represented by propositions, which are collected in a set when an activity is executed, constituting the state of the process.

Example 4 (Car Company, continued)**.** *Considering again Example 1 and more precisely: Example 2 for the business process model and Example 3 for the regulatory framework. Recalling that among the obligations describing the regulatory framework there is one prohibiting to use* old engines *due to regulations on CO_2 emissions. Assuming now that the business process model of the company includes in its choice among which engine to use also* old engines*, then each of the processes using these* old engines *does not belong to the set of compliant processes.*

However if we assume that the company does not include in their processes the possibility to use old engines *for their cars, then the whole process model containing the processes of the company would be compliant with the regulatory framework.*

The example illustrates that a business process model can be compliant in different ways with a regulatory framework, partially in the case where only part of the processes modelled are compliant with the obligations, fully when each of the processes contained in the model are compliant. The example does not illustrate the third possibility, consisting of a process model containing only processes not compliant with the regulatory framework; in this case the whole process model is considered not compliant. In this thesis I focus on proving which of the three classes of compliance a business process model belongs to.

The obligations composing the regulatory framework impose conditions on the possible process' state obtainable by executing activities of the process model. Meaning that the execution of certain activities may be required or prohibited depending on the conditions imposed on the process' state. The object language used in this thesis to represent these

conditions and the process' state is *propositional logic*. This logic allows to represent the conditions of the obligations forming the regulatory framework and verifying them over the process's states identified by the possible executions of the process model. By choosing such a simple logic, I avoid to deal with the complexities that would be brought along by using more complex and expressive logics, such as for instance *temporal logic* and *dynamic deontic logic*. Despite of the simplicity of the logic adopted to study the problem, it is expressive enough to define the obligations composing the regulatory framework and to use them to identify whether a business process model is compliant with it.

More precisely, propositional logic is used to define the conditions identifying which states of a process needs to comply with the conditions specified by the obligations. In general not each state describing a process needs to comply with the specifications of every obligation. The states which need to comply with the specification of the obligation are identified using two additional conditions, expressed through propositional logic formulae, which identify a subsequence of the process where the specification has to be complied with. I refer to the condition identifying the beginning of these subsequences as *lifelines*, and to the conditions identifying the end of these subsequences as *deadlines*. Therefore an obligation is generally composed of these three elements: a lifeline, a deadline and a fulfilment condition.

Example 5 (Car Company, continued)**.** *Considering the previous examples used to illustrate the problem of proving regulatory compliance, I illustrate in this example how propositional logic can be used to describe the problem through its relevant elements. Assume that the propositional formula α represents the property* ecologically friendly *of a car. Depending on the type of engine used, therefore on the process used to build it, a car can either be α or not. Therefore constructing a car not ecologically friendly would be represented in the process state as the negation of the proposition, written $\neg\alpha$.*

Considering now the regulatory framework, the regulations on CO_2 emissions can be represented by obligations requiring that a process whose objective is to build a car, builds it in such a way that α is true. More complex regulations stating disjunct conditions, such that a car must be ecologically friendly or being a suburban vehicle (SUV), can be expressed as $\alpha \vee \beta$, where β is a proposition representing the property being a suburban vehicle.

Example 5 illustrates how propositional formulae can be used to represent the conditions of the obligations that verify the states of the processes being evaluated. Additionally these formulae are also used to represent in exactly which states of a process the condition of an obligation must be evaluated. For instance considering a business process model containing processes describing how to build waterproof watches, the property *waterproof* needs to be evaluated only at the end of the processes.

Despite the fact that using propositional logic allows a simpler representation of the states of the processes, hence simplifying the complexity of verifying properties on the states, a drawback of this approach is the limited expressivity of the problem resulting from the simpler language adopted. For instance using a temporal logic it would be possible to

express temporal relations among the various properties being introduced in the process state by the execution of a single activity. However using propositional logic does not allow this and these temporal relations can only be inferred from the ordering execution of the activities themselves. Therefore using propositional logic leads to a more coarse grained representation of the problem, but avoids the complexities derivable from more expressive logics, such as temporal logics.

1.1.4 Complexity of the Problem

Proving the compliance of a single process is in general not a complex task, especially considering the scope of the thesis where propositional logic is used to represent the process' states and the obligations. As I will show in more detail in the later chapters, this is not a time consuming task since it requires to analyse the process states, expressed through a set of propositional literals, in order and check them against the different obligations composing the regulatory framework. The computational complexity in this case is low since a process' state identifies a truth assignment for the propositional literals. This truth assignment is then used to evaluate the truth value of a propositional formula.

Differently, proving the compliance of a business process model can be instead a time consuming problem. The source of the complexity is that a single business process model can be executed in an exponential number of different ways, with respect to the size of the model, due to the possible interleaving between the activities contained and the different mutually exclusive activities. Because of this a brute force approach to the problem would not be able to solve it efficiently. Therefore, as other researchers in this area have tried, I approach the problem directly by analysing the process model.

In general, the current literature about solving the problem of proving regulatory compliance of business process models has neglected the complexity of the problem. This has led to a number of solutions whose worst case computational complexity is prohibitive. Nevertheless some of the current works on the subject have taken into account the computational complexity and proposed some feasible solutions. For instance Ghose and Koliadis [30] propose a computationally acceptable solution by not considering each possible interleaving of the activities. In this way Ghose and Koliadis can only prove the compliance of business process models where the interleaving between the activities is not allowed. Another computationally acceptable solution has been proposed by Hoffman et al. [44], who propose a polynomial time approximate solution to the problem of proving regulatory compliance of business processes. This approach is based on the technique of I-propagation. Similarly to Ghose and Koliadis [30] they avoid considering the interleaving of the activities. Moreover Hoffman et al. to improve their efficiency of their computation they approximate the results of executing mutually exclusive sets of activities.

1.2 Research Question

Proving the regulatory compliance of business process models is a key issue of business companies adopting them to guide the work of their employees or to describe their processes. Example 6 illustrates some of the different aspects that must be considered while proving regulatory compliance.

Example 6 (A Difficult Problem)**.** *A banking company must follow a variety of regulations while performing the financial operations requested by its clients. Some of these operations may involve opening an account for a new client, performing a transaction from an account to another, allowing a client to take a loan from the bank and many more. Some instances of the regulations that a bank may have to follow while performing these operations are the following: the bank must verify the identity of a new customer willing to open an account and must handle the information collected according to privacy regulations, the bank must verify that financial transactions between accounts are not made for illegal purposes, etc.*

Additionally some of these operations may be requested to be executed sequentially, like for instance a new client asking for an account and a loan. These sequences of operations are recorded by the bank through logs *and each of these logs is required to comply with the regulations.*

The complexity of ensuring that the series of operations performed comply with the regulations is increased since a bank may be required to identify and handle exceptional situations which should not follow the standard procedures. These exceptional cases also include situations where one or more violations of the regulations have been detected and compensatory measures are required.

Finally a banking company can adopt models to streamline the way these operations have to be handled, providing in such a way guidelines for its employees. However, while producing these guiding models, a company must also ensure that the outcomes of using such models are compliant with the governing regulations.

From the instance of the problem illustrated in Example 6, it emerges that the problem is complex. Nevertheless the computational complexity of the problem of proving regulatory compliance has not yet received much attention. Most of the research effort has been focused on finding automated solutions and effectively representing the problem. Moreover, the lack of efficient solutions, apart from the ones proposing approximated results or the ones working on very restricted sub-classes of the problem, suggests that the complexity of the problem itself is most likely to be hard.

Therefore the main research question I am answering in the present thesis concerns identifying the computational complexity of the general problem of proving regulatory compliance described by the framework adopted in this thesis. More precisely:

RQ What is the computational complexity of the general problem of proving the regulatory compliance of a business process model?

The answer to the research question can be both seen as computational complexity upper bound of the compliance framework adopted in the thesis, and as a computational complexity lower bound for more expressive compliance frameworks.

When studying complex problems such as the one of proving regulatory compliance, it is usually good practice to analyse simpler versions of the problem and reuse the results to compute the one for the general problem. This consists of dividing the general problem into smaller sub-classes and tackle them individually. Adopting this strategy allows to define additional research subquestions supporting the original one of studying the complexity of the general problem.

RSQ1 What is the computational complexity of the sub-classes of the problem of proving regulatory compliance?

RSQ2 Which are the sub-classes of the problem of proving regulatory compliance that are non-trivial and tractable?

Once the problem has been divided into sub-classes, then each of its sub-classes can be studied independently. Thus the first research subquestion arises: which are the computational complexities of these sub-classes of the problem, and how does studying them help us understanding the computational complexity of the general problem.

The second research subquestion derives from the first one, aiming at identifying the computational complexity of the sub-classes of the problem. Answering the first research subquestion may identify a tractable subset of the sub-classes of the problem. However the second research subquestion aims also at finding non-trivial sub-classes of the problem. The term non-trivial has to be understood in this context as a sub-class of the problem which retain enough expressivity to be able to represent and reason about relatively complex real world problems.

1.2.1 Scope

I restrict the study of the complexity of the problem of proving regulatory compliance of structured business process models. This restricted family of business process models includes the ones where their components are be properly nested. Moreover, proving the correctness of structured process models is tractable, as it has been shown by Kiepuszewski et al. [49] for structured workflows. Structured workflows are models comparable to the ones used in this thesis. A property of these structured business process models is that their correctness, meaning that each of their execution terminates with the absence of lifelock and deadlock, is verifiable in time polynomial with respect to the size of the model. A lifelock represents a state of a process in which even though it can execute activities, it cannot terminate. Differently, a deadlock represent a state of a process in which it cannot execute any activity and therefore it cannot terminate. Structured process models are often used to model real life processes as Keller and Teufel [47] point out; indeed a

relevant part of the 406 of the 604 processes in the SAP reference models are structured, corresponding of about two thirds of the reference models. An additional restriction I am adopting over the ones adopted in the structured workflows by Kiepuszewski et al., is that the models in this thesis are further constrained by the fact that they do not allow cycles. This further restriction present the advantage that the activities contained in the business process model can be each executed at most once. Which in turn it means that each execution of such a restricted structured process model is bound to be finite, since the longest possible execution would execute each of the activities exactly once. The drawback of such restriction is that repeatable activities cannot be properly represented, except for the fact that such activities may be repeated within the model, representing in such a way a bounded cycle. However cycles containing activities that can in theory be executed countless times cannot be expressed due to the restriction.

The second restriction concerns the object language used to represent the states of the processes and the elements of the obligations. I limit the scope in this case to the use of propositional logic since it allows enough expressivity to model the different conditions necessary to describe the obligations and verify them against the states of the processes. However by restricting the scope to propositional logic I do not allow the use of more expressive logics like for instance temporal logics and epistemic logics which would have increased the expressivity of the framework describing the problem and its complexity. As a consequence, even though the expressivity of the language used is restricted when compared to more complex ones, the language adopted to represent one of the elements of the problem of proving regulatory compliance is tractable.

The third and last restriction concerns the obligations. Obligations are often understood as conditionals in *normative reasoning*, as discussed by Hansson [42]. The field of research concerned about representing and reasoning about normative concepts such as obligations. For an obligation to be conditional it means that it needs to be fulfilled in case a condition is met. These conditions that activate an obligation can refer to the context or even to other obligations that need to be fulfilled already. Some of the sub-classes of the problem analysed in the present thesis assume that the active obligations are given, by doing so these sub-classes avoid to deal with the complexity layer of calculating whether an obligation's condition has been triggered or not. Although conditional obligations are used in some of the sub-classes analysed in this thesis, the conditional obligations used are an abstraction since their activation can only depend on the context, but not on other conditions such as other active obligations. Avoiding conditional obligations basing their activation on other active obligations reduces the complexity of calculating which obligations are active, however the tradeoff reduces the expressivity of the problem studied.

The scope of the present thesis restricts the expressivity of the individual elements, the business process model and the regulatory framework, composing the problem of proving regulatory compliance. The restriction ensures that considered individually both elements are tractable. However, as I show in this thesis, the problem of proving regulatory compliance resulting by combining these tractable elements is intractable instead.

1.3 Methodology

I describe in this section how I approach the main research question about how to study the computational complexity of the problem of proving regulatory compliance. I first describe how the problem of proving regulatory compliance is divided in sub-classes, which are then studied to answer the two research subquestions concerning the sub-classes.

The division into sub-classes I use to study the problem focuses on simplifying the regulatory framework describing the obligations that a business process model needs to comply with. I identify three orthogonal binary features, meaning that each of these features has an easier and a more difficult option from which to chose. Using these features I can define the different sub-classes of the problem depending on which option is used for each feature, where the most general problem is defined by using the most difficult option in each of the features. Each other combination of the binary features identified defines one of the sub-classes of the problem of proving regulatory compliance.

1.3.1 Sub-Classes

The three orthogonal features I use to define the sub-classes of the problem determine how many obligations compose the regulatory framework, whether these obligations are always active or the activation period is subject to some conditions, and whether violations of the obligations can be compensated or not.

The following list summarises the three binary features.

1.
   - Regulatory control constituted by a single obligation
   - Regulatory control constituted by a set of obligations
2.
   - Only global obligations
   - Allowing local obligations
3.
   - Obligations without compensations
   - Obligations that can have compensations

The first feature determines how many obligations can be included in the regulatory framework. Verifying whether a business process complies with a single obligation is simpler than verifying whether it complies with a set of them. To be compliant with a set of obligations, the processes contained in a model need to comply with each one of the obligations belonging to the set. Meaning that these processes must be checked against each obligation.

The second feature determines whether the obligations allow the use of conditionals to determine when they are active. These conditionals are referred to as lifelines and deadlines, the first one defining the activation condition and the second one the deactivation condition. I refer to an obligation not allowing these features as a global obligation, otherwise it

is a local obligation. In the simplest case, where these conditions are not allowed, the obligations are active for the whole duration of the processes in the model. Differently, in the difficult case, the obligations are initially considered inactive and the periods in which they are active need to be calculated, introducing a computational overhead for each of the obligation composing in the regulatory framework.

Example 7 (Global and Local Obligations)**.** *"The production cost of a car must not exceed one third of its selling price" is an instance of a global obligation. This can be classified as a global obligation since it does not require any condition to be applied but is enforced for the whole duration of a process of building a car.*

Differently if we consider the following obligation: "If a car is a suburban vehicle, then it must have four-wheel drive", then such obligation is a local one since it applies only in the case when the car being built is of a specific type.

The last feature determines whether the obligations in the regulatory framework allow compensations. The simplest case, where compensations are not allowed, does not require additional analysis when an obligation is not complied with. This is not the case when compensations are used since additional analysis is then required when a obligation is not complied with to verify whether it is at least compliant with the associated compensation. Notice that a compensation is considered inactive until the associated obligation is violated, hence a compensation become satisfiable only in the presence of a violation of the associated obligation.

Example 8 (Compensable Obligations)**.** *"Students must enrol to the semester before the deadline, otherwise they must pay an additional fee". This obligation is compensable since it provides an additional obligation, paying an additional fee, that has to be fulfilled in case the first one, enrolling before the deadline, is not. A regulatory framework not allowing compensable obligation would be able to capture only the first part of the obligation: "Students must enrol to the semester before the deadline".*

By considering each of these feature as a vector, it is possible to describe the general problem and its sub-classes using a visual representation. In the remainder of this thesis I refer to these features as difficulty vectors, since by moving on them it is possible to alter the complexity of the problem.

In Figure 1.1 the 8 sub-classes of the problem are represented graphically. Each feature of the regulatory controls belongs to one of the three dimensions as shown on the left side of the picture. The single-multiple vector refers to whether a business process has to be checked against a single obligation or a set of them. The global-local vector refers to whether the obligations used contain lifelines and deadlines identifying their activation periods or they are assumed to hold for the entirety of each execution. Finally the atomic-compensation vector refers to whether the obligations used allow compensations in case they are violated or not (written as atomic).

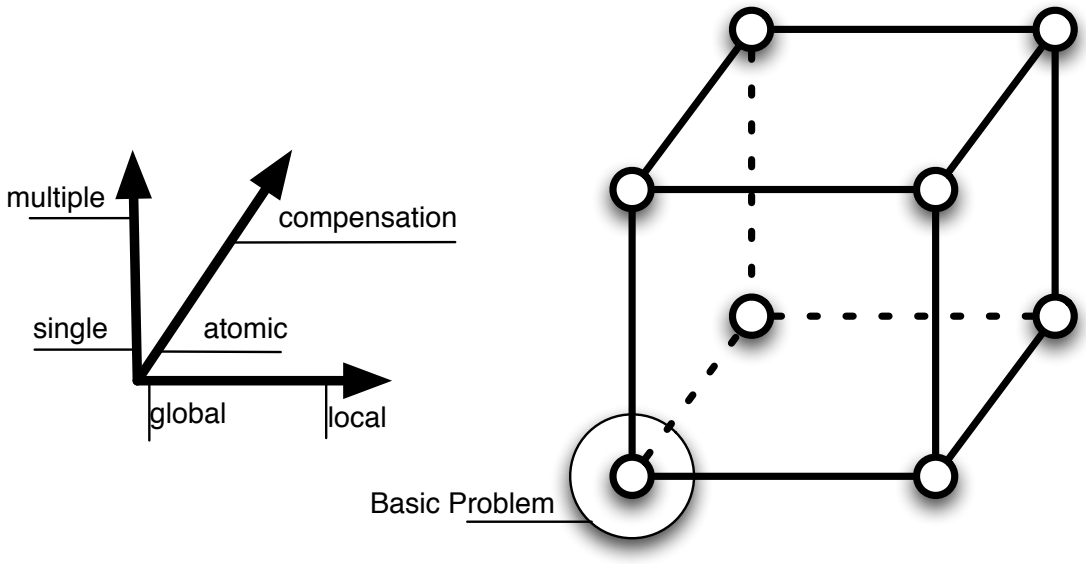

Figure 1.1: Sub-Classes of the Problem

On the righthand side of the picture, the sub-classes are represented as a cube. The foremost bottom left vertex represents the most basic sub-class of the problem, where the compliance of a business process model has to be verified with a single obligation with no compensation associated and with a global activation period.

Figure 1.2 represents the different sub-classes in a lattice where the problems are ordered from the simplest (bottom) to the most difficult (top) according to the amount of features considered. The figure also introduce the naming structure of the sub-classes, the problems are divided in four families named Cx, where x identifies the amount of difficult features from the difficulty vectors included in the problem. For instance the sub-class *C0* does not include any of the difficult features, meaning that the regulatory framework considered in this case is composed of a single global atomic obligation. Differently a sub-class labeled *C1* has its regulatory framework containing exactly one of the difficult features generated by the difficulty vectors.

To distinguish the different sub-classes including the same number of difficulty features I include as a subscript a string composed of three characters identifying exactly the features included in the sub-class. The first character can be either 1 or n and distinguishes where the problem lies on the difficulty vector determining whether the regulatory framework contains a single obligation (1) or multiple obligations (n). The second character considers the difficulty vector determining whether the obligations composing the framework are global (g) or local (l). The third and last character considers the difficulty vector determining

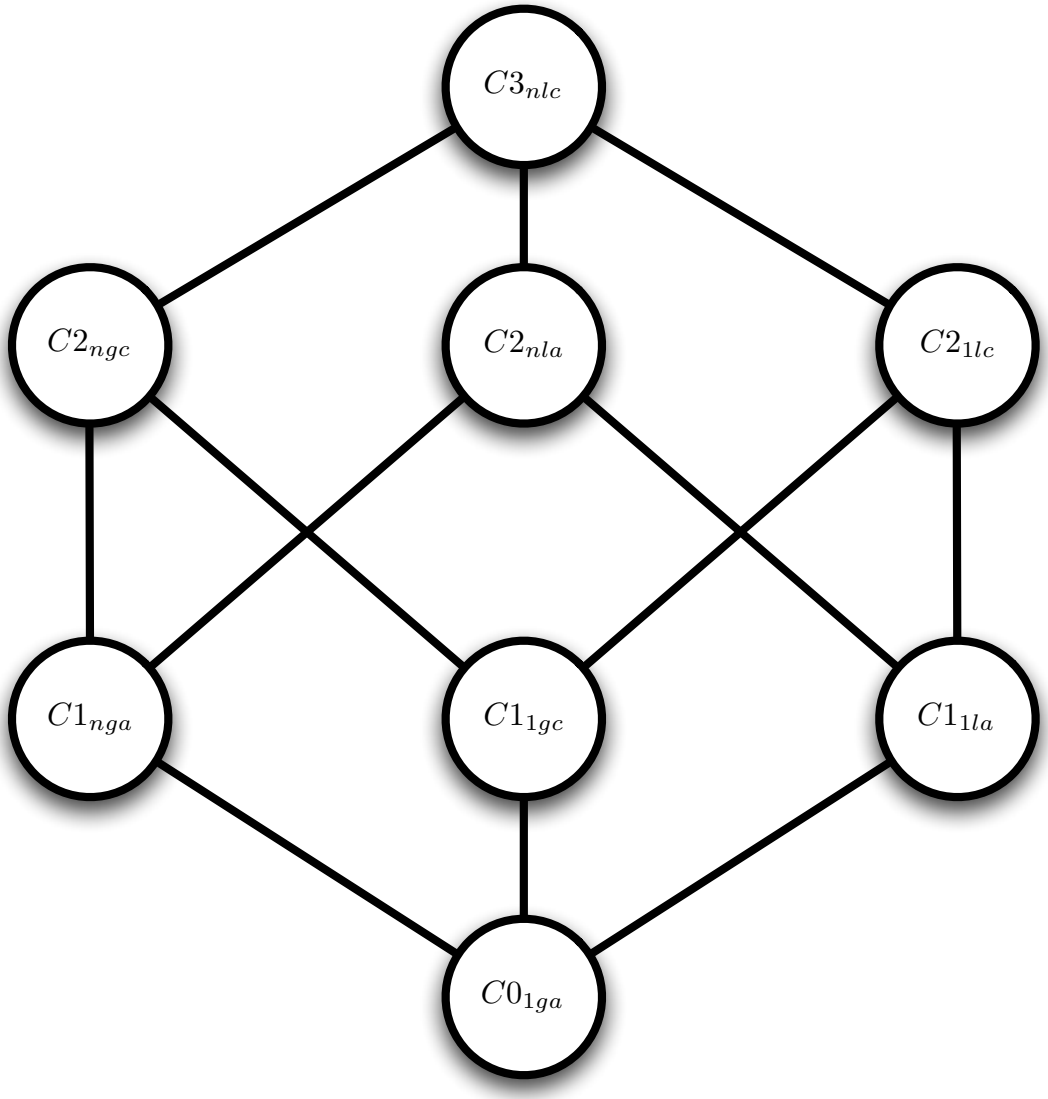

Figure 1.2: Lattice of the Sub-Classes of the Problem

whether the obligations composing the framework are atomic (a) or compensable (c).

For instance both sub-classes $C1_{1la}$ and $C1_{nga}$ contain a single difficult feature each, belonging both to the family $C1$. However they are different since the first one increases the difficulty by moving on the vector *global-local* and the second moves on the vector

single-multiple.

Using these features it is possible to identify eight sub-classes, however I am considering one additional binary feature that brings the amount of sub-classes to sixteen. This last feature concerns how the elements, namely the lifelines, the deadlines and the fulfilment conditions of the obligations are represented. The two options composing this feature are whether to represent these elements using propositional formulae or propositional literals. The first set of sub-classes of the problem, where propositional formulae are used to represent the elements of an obligation, is indeed more expressive and potentially more difficult. In the second set of sub-classes, where the elements of an obligation are restricted to propositional literals, it is sufficient to verify whether the literal used belongs or not to a process' state. However, when propositional formulae are involved, the process state is used to provide an evaluation of a formula, which can lead to a more complex problem of proving regulatory compliance, especially when considering the set of possible executions of a business process model.

Introducing this additional feature allows to create a second lattice of sub-classes of problems, where the elements are restricted to propositional literals. In Figure 1.3 are shown these sub-classes of the problems, named Cx^{-} to distinguish them from the problems illustrated in Figure 1.2, where the elements composing an obligation can be represented using propositional formulae. Given that each of the problems contained in this second lattice (Figure 1.3) have the elements describing their obligations composed of only literals. Therefore each problem Cx^{-} is at most as difficult to the corresponding problem Cx (Figure 1.2), where by corresponding it means that it uses the three features identified by the difficulty vectors.

By dividing the problem of proving regulatory compliance in this way the computational complexity can be studied in the different sub-classes of the problem individually and the results can be used to identify the complexity of other sub-classes by taking into consideration the complexity relations as expressed in the lattices.

1.4 Relevance

On the 30 July 2002, when the "Public Company Accounting Reform and Investor Protection Act" and "Corporate and Auditing Accountability and Responsibility Act", usually known as *SarbanesOxley Act of 2002* (SOX) [1], has been approved to react to a number of major corporate and accounting scandals including those affecting Enron, Tyco International, Adelphia, Peregrine Systems and WorldCom.

The *Sarbanes-Oxley* act is just an example of these regulatory acts. In response to it and to the perception that stricter enterprise regulations are needed, similar laws have been subsequently enacted in different countries. These laws impose on enterprises that they need to show their compliance with the requirements defined by the state's laws.

As a consequence of the proliferation of these laws, the study of information technology

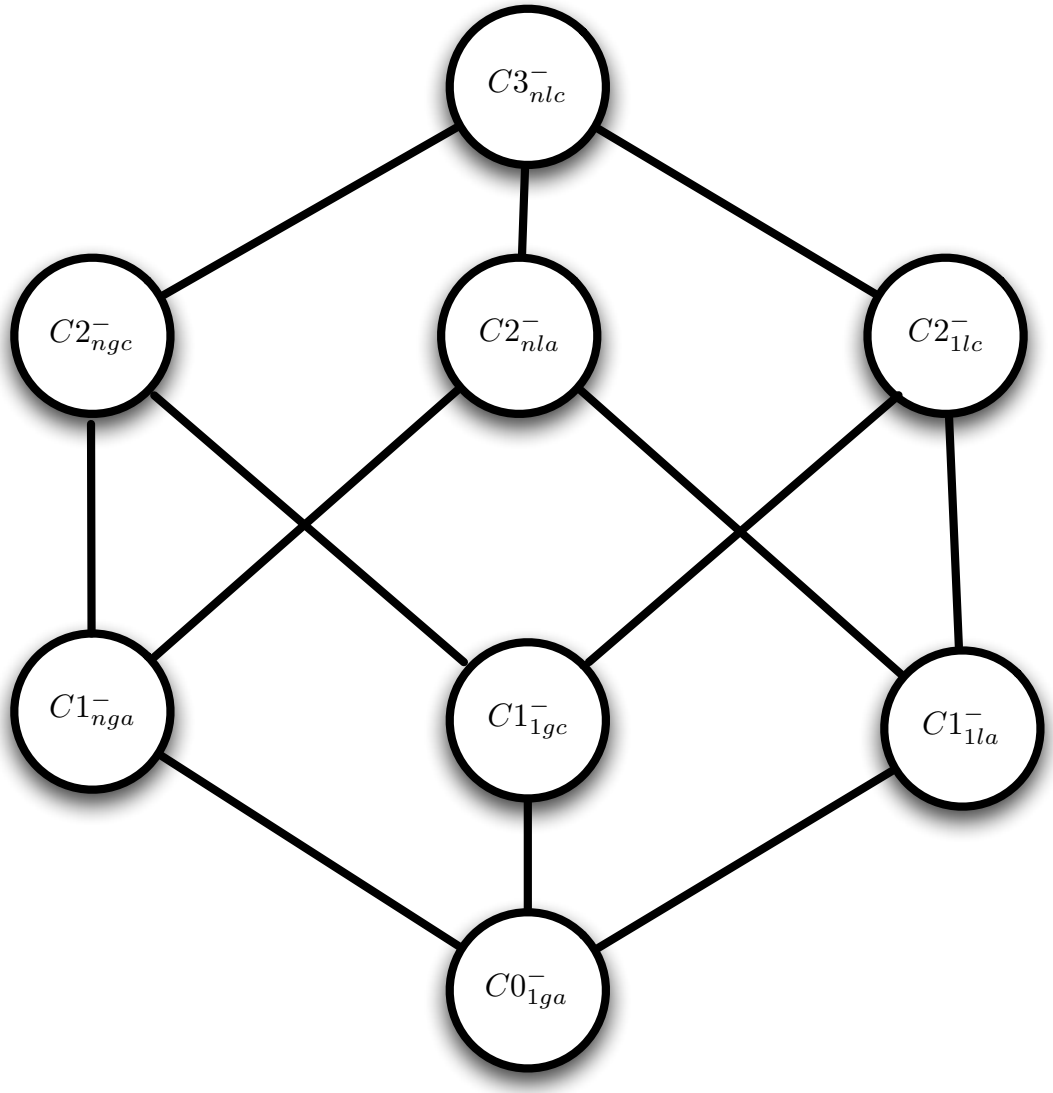

Figure 1.3: Alternative Lattice of the Sub-Classes of the Problem

techniques supporting compliance initiatives is growing. It has been estimated that the IT financial compliance management will rise between 10 and 15 percent per year [11]. The problem of automating the procedures of proving that the processes used by a company are compliant with the governing regulations has become quite popular in computer science as can be seen from various approaches proposed in the current literature concerning *regulatory*

compliance.

More recently, in response to the 2008 financial crisis, the BCBS (Basel Committee on Banking Supervision) proposed the *BASEL III* guidelines [12]. These guidelines are aimed at strengthening Individual Financial Institutions as well as the overall Financial System by eliminating the weaknesses which were present in *BASEL II* and were revealed during the crisis.

In general implementing the guidelines proposed by BASEL III consists of analysing the existing models describing the processes of a financial institution and verifying whether they are compliant with the regulatory guidelines provided. The problem which the guidelines proposed by BASEL III aim at solving is the problem of proving regulatory compliance, which is the one being studied in this thesis.

Another approach aimed at ensuring compliance is *auditing*. This approach aims at analysing executed processes, also referred to as logs, of companies to identify whether the regulatory requirements have been fulfilled. A classification of this approaches to ensure compliance is provided by Sadiq and Governatori [64], in which they divide them in preventive, approaches analysing the models in order to avoid executing faulty processes; and detective, approaches like auditing which focus on identifying compliance breaches when they arise.

A question that arises at this point is: *why do companies not tailor their processes to be compliant with the regulations?* Unfortunately, as pointed out by Sadiq et al. [65], the life cycles of the companies' procedures and the law are not aligned in terms of time, governance and stakeholders. Therefore tailoring the procedures according to the regulations is not always possible, especially in scenarios where a multi-national company uses some standardised processes which have to be used in different countries with different laws.

1.4.1 Some Existing Approaches

Many approaches have been provided in the past years capable of automatically verifying the compliance of companies' business process models. El Kharbili [48] carried out a survey and a comparative analysis of compliance solutions proposed in research on regulatory compliance management from the perspective of enterprise modelling.

Initiatives tailored to check compliance can be classified in two categories: detective and preventive. The first category includes the approaches that analyse existing executions of processes, such as logs, and verifies whether violations of the compliance requirements have occurred. The second approach includes the approaches that analyse the process model and verifies whether the possible executions of the model are going to violate the compliance requirements. In the present thesis I focus on preventive approaches.

One of these preventive approaches is proposed by Governatori and Sadiq [38] and involves a methodology based on the use of business process models to describe the activities of an enterprise and to couple them with formal specifications of the regulatory frameworks regulating the business. Wynn et al. [83] investigate the requirements for evaluating

running business process models and propose an architecture addressing these requirements. Governatori and Sadiq represent the business process models using *BPMN 2.0*. However this is not the only business modelling tool available as we can see from other existing approaches. Some of these approaches are based on workflow modelling, such as the one proposed by Rito Silva [62] and the one proposed by van der Aalst et al. [78]. An additional approach, proposed by Grigori et al. [39], presents a set of integrated tools that supports the management of business process execution quality.

The approaches proposed are based on different logical formalisms and propose solutions either for the general problem or for fragments of it. Some of these solutions have been provided by van der Aalst and Pesic [76], Goedertier and Vanthienen [31], Awad et al. [9], Hoffmann et al. [44], Roman and Kifer [63], and many others. Some of the logical formalisms that have been used to study the problem are *deontic logic*, used for instance by Governatori et al. [35], linear temporal logic over finite traces, used for instance by van der Aalst and Pesic [78], clause based logic/logic programming, used for instance by Ghose and Koliadis [30] and by Governatori et al.[33]. Some of the approaches have also focused on extending the BPMN language to increase its expressivity, as has been done by Awad et al. [9].

1.5 Interdisciplinary Aspects

The problem of proving regulatory compliance involves research areas from different disciplines. Designing the process models is mostly a problem of knowledge engineering, while designing the regulatory framework is mostly a problem of normative reasoning.

1.5.1 Business Process Models

Constructing the models representing the processes of a company is a problem of knowledge representation, acquisition and management. Many researchers have already studied this particular problem, proposing various kinds of solutions that companies can use to represent their procedures. The solution proposed by van der Aalst [79] shows how Petri nets can be used to represent procedures, also known as workflows, of individual companies. In this particular work van der Aalst also describes how messages can be used to describe the interaction between different models, being from different companies or different models from the same one. Another solution has been proposed by Ankolekar et al. [6], where they use an ontology to describe the properties and capabilities of Web Services. An additional solution proposed by Lutz and Sattler [52] uses description logic to describe actions, which in turn can be used to describe Web Services. The two latter instances of proposals deal with Web Services instead of business companies. While there is a difference between the two, it can be seen that describing how a Web Service provides its services is a similar problem to describe how a company provides its.

1.5.2 Regulatory Framework

The regulatory framework is based on guidelines and laws, which are usually written in natural language. Therefore a necessary first step to automatise the process of verifying whether a model describing the procedures of a company is compliant with the laws, is to model these laws in a format easier for machines to understand and handle. This side of the problem is also a problem of knowledge representation which has been mostly been studied in the field of normative reasoning. The normative reasoning field is concerned about representing and reasoning about obligations using formal methods. These systems dealing with laws, usually referred to as norms in computer science, are known as *normative systems*. According to Jones and Sergot [46] a normative systems can be used to describe various kinds of organisational structures as can be seen in the following quote:

> ... law, computer systems, and many other kinds of organisational structure may be viewed as instances of normative systems...

Another prolific area in normative reasoning is the one of *normative multi-agent systems*. These systems are populated by autonomous entities, the agents, whose behaviour is restricted by the normative system in place, as described by Lopez et al.[84]. A similarity can be seen between agents and companies, being both autonomous entities whose behaviours (ways of achieving their objectives in case of the companies) are constrained by the governing normative system. Much work has been already done in this specific topic, some of which has been surveyed by Hollander and Wu [45], and Criado et al. [25].

Within the field of normative multi agent systems, researchers have focused their efforts on different aspects relevant in the field. For instance Aucher et al. [7] define a language, based on *dynamic epistemic logic*, capable of verifying whether a given situation is compliant with the enforced privacy policies. While other researchers, like Telang and Singh [72] develop cross-organisational models based on commitments, which are constructs extending the semantics of obligations to perform certain actions by including who (which agent) is committed towards who in doing so. Another approach by Boella and van der Torre [17] proposes a game theoretic approach to normative multi-agent systems.

The regulatory framework adopted in the compliance framework in the present thesis to study the problem of proving regulatory compliance focuses on regulating the control flow of business process models, in other words the order in which the activities composing them can be executed. Due to the chosen focus, other aspects, such as data-centric approaches, aimed at analysing the data produced by executing the activities, are left out of the scope of the present thesis.

1.6 Related Publications

Some of the material discussed in the present thesis has already appeared in the proceedings of some conferences and workshops, as well as some journals. The basic problem, discussed

in Chapter 3, has been introduced and discussed [24] and coauthored with Marwane el Kharbili, Guido Governatori, Qin Ma, Pierre Kelsen and Leendert van der Torre. The characterisation of conflicting obligations in dynamic settings discussed in Chapter 4, such as executions of business process models, has been originally introduced in [23] and has been coauthored with Guido Governatori and Pierre Kelsen. The complexity proof of one of the sub-classes of the problem of proving regulatory compliance, discussed in Chapter 5, has already appeared in [74] and has been coauthored with Guido Governatori and Pierre Kelsen.

1.7 Outline of the Thesis

The remainder of the present thesis is structured as follows: Chapter 2 discusses some related works in deeper details and points out their differences and similarities. Chapter 3 introduces the abstract framework used to define the problem of proving regulatory compliance, in particular this chapter discusses the most basic sub-class, for which some low order complexity solutions are proposed. Chapter 4 formally introduces the difficulty vectors defining the features of the regulatory framework that identify the different sub-classes. The fourth chapter pays also particular attention to the case where multiple obligations are used to define a regulatory framework, characterising in this case conflicting obligations and proposing conditions to identify them. Chapter 5 analyses the computational complexity of two sub-classes, where in both cases the regulatory frameworks are composed of a set of local obligations, but in one case the obligations allow propositional formulae and in the other are restricted to propositional literals. Chapter 6 reuses the results obtained by Chapter 5 to obtain the computational complexities of other sub-classes, among which also for the sub-class equivalent to the general problem of proving compliance. Chapter 7 investigates an alternative route with the goal to identify tractable sub-classes of the problem. Differently from the previous chapters, the seventh one analyses a sub-class obtained by wakening the expressivity of both the regulatory framework and the business process model, the results show that such sub-class of the problem is tractable. Finally Chapter 8 concludes the thesis by summarising the results and highlighting the research questions still open as a direction for future research.

Chapter 2

Related Work

In the present chapter I discuss some of the existing work related to the problem of proving regulatory compliance of business process models. The work I am discussing in this chapter can be classified in three categories: comparative frameworks, aiming at providing a framework to evaluate and compare existing approaches other than identifying the desirable functionalities. The second class includes the approaches adopting structural patterns to represent and verify the compliance requirements that the business process models must follow. Finally, the third class includes the approaches adopting logic oriented solutions, such as temporal logics, to represent and verify the compliance requirements.

A common element among the related work I discuss in the present chapter concerns the computational complexity of the problem of proving regulatory compliance. In general these approaches do not consider the computational complexity of the problem when tackling it. However, the approaches that do consider it, often tackle a subset of the general problem or provide approximate solutions in order to keep the complexity of their approaches to a tractable level. Nevertheless, in general researchers studying the problem of proving regulatory compliance acknowledge the difficulty of the problem, but a formal study of its computational complexity has not been done yet.

This chapter is structured in three sections. The first section discusses the comparative framework, the second section discusses the approaches adopting structural patterns and the third section the approaches adopting logic oriented solutions.

2.1 Comparison Framework

The first part of the present chapter discusses the work of Thao Ly et al. [54], presenting a framework for comparing existing compliance monitoring approaches. Thao Ly et al. distinguish and classify existing approaches through the use of ten desirable functionalities that they identify.

After discussing the comparative framework introduced by Thao Ly et al. I analyse how

many of the ten typical functionalities are covered by the approach adopted in the present thesis.

2.1.1 A Framework for the Systematic Comparison and Evaluation of Compliance Monitoring Approaches

The first related work I discuss in the present chapter is the comparative framework developed by Thao Ly et al. [54]. The comparative frameworks aims at comparing existing approaches towards tackling the problem of proving regulatory compliance through the use of ten desirable functionalities identified by the authors. Additionally the functionalities can also be used as guidelines while designing new approaches to tackle the problem.

The discussion of the present related work follows closely the ten typical functionalities identified by the authors and is structured according to the one being described.

1st Functionality

The first functionality pointed out by Thao Ly et al. concerns constraints referring to time. These time constraints determine temporal conditions that must be respected when executing the activities of a process. They distinguish two types of temporal constraints: *qualitative*, which determines typical temporal patterns obtainable by imposing orderings using "before" and "after", and *quantitative*, which determines the time distance required between two activities. The authors mention some approaches that can be used to verify this functionality while proving compliance, like temporal logics such as *Linear Temporal Logic* (LTL) and *Computational Tree Logic* (CTL).

2nd Functionality

The second functionality concerns constraints referring to data. These constraints describe the expectations concerning the data and their values. Again the authors differentiate this functionality in two types: *unary data conditions*, governing a single data object, and *extended conditions*, defining constraints involving and relating multiple data objects. To implement this functionality the authors underline that the approach must be able to evaluate the truth of the data conditions, which requires the ability to access the data sources in the process environment.

3rd Functionality

The third functionality concerns constraints referring to resources. These constraints describe the requirements and expectations concerning the usage of resources associated to activities and events in a process. In a very similar way to data constraints, the authors distinguish two types of resource constraints analogous to the two types identified for data constraints: unary conditions involving a single resource and extended conditions when

relating multiple resources. The implementation of these constraints is described in a similar way as the constraints concerning data, which requires access to the resource information during the process execution.

4th and 5th Functionalities

The fourth and fifth functionality concern the use of non atomic activities and their lifecycle. Differently from atomic activities, whose duration is associated to a single event, a non atomic activity is associated to multiple events and can therefore be suspended, resumed and aborted when necessary. Compliance approaches can either be *explicit*, which means by regulating the atomic events composing these activities, or *implicit*, which considers this activities as a whole without taking into account its components. In case of explicit compliance checking, it becomes relevant to monitor and verify that the executions of the atomic events composing a non atomic activity follow some given constraints. The authors point out that a way to implement non atomic activities consists of using *point-based algebra*. This algebra allows to represent the execution time of the various activities and relate them. On the other side, the lifecycle of these non atomic activities can be implemented by using the notion of states to monitor their execution and eventually identify deviations from the expected lifecycle.

6th Functionality

The sixth functionality concerns constraints that can be instantiated multiple times. With this functionality the authors aim at capturing these constraints which can be instantiated multiple times and additionally they depend on the context in which they are applied, in other words where they are instanced. The authors point out that implementing this functionality requires to precisely characterise the information defining the context in which a constraint is applied and when separate instances are created.

7th Functionality

The seventh functionality concerns reacting to compliance violations. Additionally to identifying violations of the constraints, a monitoring systems should propose also procedures to compensate the violations identified as well as being capable to continue the monitoring process once a violation has been detected. The authors suggest that to implement such feature, a system may require an event to identify compliance violations and the introduction of an operator expressing which compensatory actions are required when a violation is identified.

8th Functionality

The eight functionality concerns proactively detecting and managing violations. Since a violation cannot be undone, other than compensated, the authors propose as a relevant functionality the one to be able to anticipate future violation and adopt preventive procedures to avoid them. The authors propose to proactively identify violations to analyse whether the constraints in place conflict with each other, forcing in such a way a violation.

9th Functionality

The ninth functionality concerns explaining the causes of a violation. This functionality focuses on identifying the events causing a violation of the constraints. This feature can become very useful when a constraint may be violated in multiple ways. The authors point out that finding the root cause of a violation is far from being a trivial task and is in general impossible. Implementing the feature should be realised using diagnostic approaches.

10th Functionality

The tenth and last functionality concerns the ability of quantifying the degree of compliance. The aim of this functionality is the ability to provide a feedback to the users concerning the compliance constraints. The authors propose to identify additional degrees of compliance than the simpler *compliant* and *not compliant*. Thao Ly et al. propose as a possibility to implement such *fuzzy compliance* to characterise the degree of compliance according to the number of violations occurring.

2.1.2 Thesis' Approach

The approach adopted in the present thesis to tackle the problem of proving regulatory compliance of a business process model partially covers the functionalities identified by Thao Ly et al. More precisely the approach adopted fully supports two of the ten functionalities identified by Thao Ly et al. It supports the verification of temporal constraints and ordering executions of the tasks through the use of conditional obligations. Moreover violations can be detected and compensatory actions can be allowed through the use of additional obligations.

Three of the other functionalities identified by Thao Ly et al. are partially supported by the approach adopted in this thesis. The first one concerns data constraints verification which is partially supported through the use of semantic annotations associated to the activities being executed. This simplified representation allows to verify some properties of the data, but not the most complex ones such as for instance relations between different data. The second one concerns the proactive detection of compliance breaches, which is done by preemptively studying the business process models. The third one concerns identifying different levels of compliance, which is is supported by the adopted approach

by distinguishing three levels of compliance of a business process model: one where each of the possible executions is compliant with the requirements, one where there exists an execution compliant with the requirements and one where none of the executions of the model is compliant with the requirements.

Despite the restricted number of functionalities supported by the approach adopted in the thesis, one should not forget the goal of the current thesis: studying the computational complexity of the problem. As I show, the computational complexity of an approach supporting less than half of the functionalities identified by Thao Ly et al. is already hard.

2.2 Structural Patterns

The second part of the present chapter discusses some of the existing approaches adopting structural patterns to represent the compliance requirements and verify them. These patterns can be represented using structures like state automata and petri nets. The state of the structures adopted to represent the compliance requirements changes in accordance to the execution of the processes and allows to identify whether a violation occurs depending on their state. The related work discussed in this part of the chapter are the following: [53, 28, 61, 3].

2.2.1 SeaFlows Toolset - Compliance Verification Made Easy

Thao Ly et al. [53] propose the SeaFlows Toolset as a user-friendly formalism for modelling compliance rules. The toolset proposed allows for both structural compliance checking as well as for data-aware compliance. The compliance rules are modelled using a graph-based specification language. The models used by the authors allow in addition to express temporal constraints, such as delays between the activities being executed, by using temporal annotations. Additionally these annotations can be also used to express conditions on the data handled by the activities being executed. The whole rule modelling language is additionally supported by a graphical interface to provide a user friendly environment.

The structural compliance of a business process model is checked first by deriving *process' structure criteria* from the compliance rules and second by verifying whether the model satisfies them. In the case a derived process' structure criteria is not satisfied by the model, additionally than identifying the compliance breach, the criteria can also be used to provide diagnostic feedback and identify where the process model failed to comply with the compliance rules.

While dealing with data-aware compliance, the authors acknowledge the complexity of dealing with data constraints since it can lead to a state explosion when exploring the data dimension, hence leading to the intractability of verifying these data constraints. The authors address the computational complexity of proving data-aware compliance of a business process model by applying a context-sensitive abstraction to the model. This

allows to derive a more compact model to be checked, whose state space representation does not explode.

The compliance verification toolset proposed by Thao Ly et al. allows to check both structural and data-aware compliance of a business process model. The authors finally point out some ways of extending the present approach, such as by including online compliance checking as well as by extending the visual interface to provide advanced feedback for the users.

2.2.2 Root-Cause Analysis of Design-time Compliance Violations on the basis of Property Patterns

Elgammal et al. [28] propose in their work an approach capable of verifying regulatory compliance and identifying root-causes of compliance violations identified in the business process models being checked. With the term root-cause it is understood the initial cause that triggered the chain of events that finally led to the compliance violation. Compliance requirements are modelled using property patterns and temporal logic. These patterns are then checked against the business process model, allowing to identify compliance breaches and their root-causes. As the authors point out, identifying the root-causes of these compliance is a key step to first identify which part of the model is responsible and eventually resolve these design-time compliance violations.

The focus of the work of Elgammal et al. is on preemptive compliance, which considers the compliance requirements at the process design time and thus allow more sustainable business process models since they are already tailored to be compliant with the requirements. The compliance requirements are expressed using Dwyers property specification patterns [27] and Linear Temporal Logic (LTL) [58]. The authors, using these two formalisms, present additional pattern extensions and introduce new ones that are frequently used to specify compliance constraints. The pattern-based expressions are automatically transformed into LTL formulae, based on the mapping rules between the patterns and LTL. The advantage of such translation is that the automatic verification can be performed by model-checkers. Some of the patterns used in the work of Elgammal et al. include when the execution of an activity later requires the execution of another later, when activities are mutually exclusive and many other.

Given that a compliance violation may occur due to a variety of reasons, identifying the root-cause of these violations helps in resolving these violations and preventing future ones. Elgammal et al. adapted the *Current Reality Tree* technique from Goldratts Theory of Constraints [26], where a current reality tree represents a problem and the possible symptoms arising from it. It maps a sequence of causes and effects from the problem. The root of the tree is associated to a compliance violation and the branches of the tree represent the possible causes. The authors define reality trees to be used to identify root-causes for each individual compliance pattern as well as for composite patterns.

2.2.3 Where Did I Misbehave? Diagnostic Information in Compliance Checking

Ramezani et al. [61] list in their work the five types of compliance related activities: *compliance elicitation*, consisting in determining the compliance requirements to be satisfied; *compliance formalisation*, consisting in precisely formulating the compliance requirements; *compliance implementation*, consisting of configuring the processes to comply with the requirements; *compliance checking*, investigating whether the requirements will be met by the processes or have been met; and *compliance improvement*, consisting of refactoring the processes to avoid further violations identified by the compliance checking activity.

In this paper, the Ramezani et al. focus on the compliance checking activity, in particular whether the processes have been compliant with the requirements, which is also referred to as *backward compliance* and *auditing*. Ramezani et al. propose a collection of control flow compliance rules formalised in terms of petri-net patterns to check backward compliance using process logs. These logs are aligned with the processes, meaning that a violation identified in the log allows to identify the root cause in the process model.

Using petri net patterns the authors are capable of representing compliance requirements constraining the bounded existence of a task, in other words the number of time it is executed, ordering constraints like the direct precedence of a task with respect to another and simultaneous execution of tasks. These patterns can be also combined in order to check requirements like bounded existence sequences of tasks.

Two other problems tackled by the approach proposed by Ramezani et al. concern complying with data flow constraints and organisational constraints. The first problem is tackled by parametrising the events occurring in the logs to be able to identify and verify the constraints between the data contained and handled by these events. The second problem is tackled by extending the representation of the event to include information about the actor executing it; in this way it is then possible to verify whether the correct actor executed the activity.

Ramezani et al. point out that their current approach can be improved in different ways, such as by introducing additional compliance requirements involving resources and concerning the processing time. In this way the authors plan as future research to broaden the compliance coverage of their current approach.

2.2.4 Automated Certification for Compliant Cloud-based Business Processes

Accorsi et al. [3] propose a compliance monitoring approach named *Comcert*, which is tailored for the automated analysis and certification of business processes. Comcert uses Petri nets to specify the compliance requirements through patterns, which are then used to automatically verify whether a business process is compliant with the compliance requirements. The authors focus their approach on validating business processes aimed to be used in cloud

computing, where the automatic verification proposed by Accorsi et al. allows to foster a wider deployment of said business processes.

The regulatory requirements considered by Accorsi et al. are obtained by surveying various guidelines and directives involving regulations about how to handle personal and medical data. These requirements concern in most of the cases the execution ordering of the activities composing the business processes being evaluated. Comcert evaluates whether the business processes are compliant with the requirements by considering each of their possible executions. While evaluating an execution of a business process, the transitions in the Petri nets corresponding to the requirements are triggered accordingly to activities being executed. The final state of the Petri nets would then identify whether the execution complied with the requirements. Finally the compliance of a business process is assessed according to the compliance results of the executions composing it.

In addition to the execution orderings between the activities, Comcert is capable of assessing the compliance of a business process by analysing additional requirements that can involve constraints concerning the data, location, time limits and resources. Considering the setting of cloud-based business processes, multiple locations can be active at a given time, especially when services are outsourced. These types of constraints are dealt with by using predicates to describe the relevant details to assess the compliance of the business process.

The contribution of Accorsi et al. in the paper being analysed is twofold: they first propose a classification of compliance rules and second propose Comcert, as an automated approach for the certification of business process compliance. The experiments carried out in the paper analysed show that the approach proposed effectively detects compliance breaches in the business processes.

2.3 Logic Based

The third part of this chapter discusses some of the existing approaches for monitoring and verifying regulatory compliance of business processes using logics to represent the compliance requirements and verify them. Different types of logics can be combined in these approaches to capture the necessary features to represent the compliance requirements. Some of the logics commonly used are temporal logics, allowing to express and verify temporal relations between the activities being executed, and deontic logics, allowing to characterise violations and eventually react to them. The works discussed in this last part of the chapter are the following: [57, 10, 36].

2.3.1 Verification of Data-Aware Commitment-Based Multiagent System

Montali et al. [57] tackle in their paper discussed here the problem of verifying data-aware commitments in a multiagent setting. A multiagent system is a system where multiple

agents, which are entities capable of proactive behaviours according to their goals, interact with one another. The social commitments used by Montali et al. in this setting represent the interaction protocols that the agents must follow while interacting among each other. Commitments are used to express the responsibility of an agent towards another to bring about something in case a given situation is met. One can see these commitments as conditional obligations with the additional information concerning who is responsible for fulfilling the obligation.

In the framework the authors introduce, the agents interact with each other exchanging messages containing events and some data. The data contained in these messages exchanged by the agents is expressed using a first-order formalism to allow an increased expressivity. This added expressivity also allows to create instances of commitments, which as also pointed out by Thao Ly et al. [54] is one of the desired functionalities of compliance monitoring approaches.

The regulations, or contractual specifications as Montali et al. call them, are defined using a set of commitment rules. Each of the commitments used contains a precondition, which activates it when verified and a discharge condition which instead is used to deactivate it. In the present thesis these two elements are mirrored by the lifelines, activating an obligation, and the deadlines, deactivating it. The lifecycle of the commitments is handled by a *commitment machine*, inspired by the work of Singh [68], which is also capable of determining whether a commitment becomes violated instead of being discharged.

The execution semantics of the framework described by Montali et al. is defined in terms of a *transition systems*, which is comparable to the business process models which are being studied in the present thesis. The properties to be verified in the framework, defined by the commitments, are expressed using a first order version of the μ-calculus [69], which is capable to express both linear temporal logics and branching time logics.

Montali et al. point out the intractability of the general problem approached in such way given the number of possible states, which does not allow to decide even for simple propositional temporal formulae. Additionally they show that the problem becomes decidable if the size of the states of the agents that requires checking against the specification is bounded.

2.3.2 Visually Specifying Compliance Rules and Explaining Their Violations for Business Processes

Awad et al. [10] propose in their work a language to prove the compliance of business process models. The compliance requirements are expressed using a visual language, *BPM-Q* [8], whose semantics is formalised using computational temporal logic (CTL). Using this combination the authors are capable of expressing the requirements through patterns.

The language BPM-Q allows to express constraints between the activities that can be executed in a process model. These constraints allow to define conditional constraints, such as that certain activities must be executed when others are executed and in which cases the

execution of an activity should prevent the execution of others. The language also allows to express data dependencies between the activities. Awad et al. show that the requirements expressed using BPM-Q can be formally represented using CTL. The authors point out that when formally representing the semantics of data-aware compliance requirements using CTL is not as straightforward as for the other constraints, since additional knowledge is required to properly express them.

The temporal logic formulae corresponding to the compliance rules can be verified against the business process model using some *model checking machinery*. At this stage of the compliance checking procedure, it is also relevant to be able to provide feedback to the process analysts when a process model is not compliant in order to allow to identify and repair the compliance breach. To do so the authors use *anti-patterns* generated from the requirements expressed using BMP-Q which are capable of describing potential violations directly on the structure of the model being analysed. In general, these anti-patterns are are derived by negating the CTL formula describing the semantics of the original compliance pattern. Concerning these anti-patterns, the authors point out that using them directly to verify compliance can produce some false-negative results, meaning that it can identify compliance breaches where there are none. Therefore they use these anti-pattern as a complementary technique to model checking to better explain where compliance breaches occur in a process model.

The approach proposed by Awad et al. is validated using a case study in a financial domain. In such domain the particular business process models being verified concern opening a bank account. The compliance requirements used in the case study refers to anti-money laundering regulations, which are translated in the BPM-Q formalism and checked on the business process model. The authors conclude their paper by promising a more thorough analysis of their approach to study whether the expressivity of the proposed approach suffices to study real problems and eventually extending the approach when this is not the case.

2.3.3 Norm Compliance in Business Process Modelling

Governatori and Rotolo [36] propose a logic for proving regulatory compliance of business processes called *Process Compliance Logic*. Such logic is an extension of formal contract logic derived as a combination of defeasible logic and a deontic logic of violations. The logic used by Governatori and Rotolo is capable of capturing both semantic and structural compliance requirements.

In their work the authors point out three complexity sources of the problem of proving compliance. The first resides in the compensatory actions required when violating the regulations in certain cases, which in turn can require additional compensatory actions. These compensatory action may give rise to very complex rule dependencies. The second complexity source concerns the fact that a process can be governed by many kinds of regulations, some required to be fulfilled for the whole duration, while other only for some

fragment of the process. This means that in the latter case the fulfilment period of a requirement needs also to be computed. The third complexity source is related to the different types of conditions that can be imposed on business processes: the regulations can impose conditions over the activities that can be performed, their order of execution and so on.

The logic proposed by Governatori and Rotolo allows to distinguish the different types of conditions, which the authors refer to as obligations, imposed by the regulations. They distinguish maintenance obligations, which require to maintain a condition for given period, achievement obligations which require to achieve a condition once during a given period and punctual which requires to achieve a condition at an exact point in time. Additionally the logic allows to handle other features of these obligations like persistence, which allows an obligation to be still active if not achieved while the deadline is reached, and finally preemptiveness, which allows obligations to be fulfilled by events occurring before they are activated. A similar semantics, capable of requiring the achievement or the maintenance of a condition, has been also proposed by Chesani et al. [18] in the context of commitments and event calculus.

Process Compliance Logic represents the regulations through the use of conditional rules consisting of an antecedent, representing the activation condition of a regulation, and a conclusion, representing the obligation active when the antecedent is satisfied by a process' state. The logic based approach adopted by Governatori and Rotolo allows to remove redundancies between the regulations by merging the corresponding rules and solving eventual conflicts by using superiority relations.

The business process models introduced in the work of Governatori and Rotolo, in addition to the standard elements, make use of semantic annotations for the activities to represent which effects they have on the process' state. The compliance of these processes is computed by considering each of their possible executions, where in turn for each activity composing it is checked whether an obligation is activated or an active one is discharged, and finally for each of the activities being verified, each active obligation is checked to determine whether the current tasks fulfils it, violates it or the same obligation needs to be checked on the following activities.

2.4 Summary

In the present chapter I discussed some of the existing works related to the problem of proving regulatory compliance of business process models. The differences between the approaches discussed allowed me to distinguish them in three categories. The first one, containing only the work of Thao Ly et al. [54], where no approaches to prove regulatory compliance are proposed, but a framework listing ten desirable functionalities that a compliance monitoring system should include and that can be used to compare existing approaches by measuring the number of features covered. The other two categories distinguished both involve approaches

to verify the compliance of a process model. The two categories differ according to the method used to do so: one of them adopts structural patterns [53, 28, 61, 3], while the second prefers a more logic oriented approach [57, 10, 36]. The approach used in this thesis is closer to the logic oriented category.

Despite the differences between each of the approaches discussed in this chapter, there is a common element shared among these works. This common element is that none of them thoroughly analysed the complexity of the problem of proving regulatory compliance. In some cases the authors hinted that proving regulatory compliance is hard and adopted some approximate solutions to reduce the computational complexity of the problem being tackled. In other cases, such as *BPM-Q* [8], it has been show that the upper bound computational complexity is equivalent to the one of model checking using linear temporal logic [19], which is *PSPACE*. Therefore formally analysing the computational complexity of proving regulatory compliance is indeed an open problem which I am tackling in the present thesis.

Chapter 3

The Abstract Framework

In this chapter I introduce the abstract framework to study the problem of proving regulatory compliance of a business process. The framework is composed of two distinct elements, the first one is the model describing the processes we want to prove to be compliant and the second is the regulatory framework describing the obligations that must be fulfilled by the process model.

The proposed framework is abstract. The reason to do so is to keep the framework as simple as possible to be able to focus on the main problem tackled in the thesis: the study of the computational complexity of proving regulatory compliance.

This chapter is divided in three main parts, the first one introduces the syntax used to represent the business process models as well as describing their semantics and some of their properties. The second part of the chapter introduces the regulatory framework which defines, through the use of obligations, which are the constraints that the business process model must follow in order to be considered compliant. The third and last part of the chapter analyses the complexity of proving the compliance of the problem represented in such abstract framework.

The framework introduced in this chapter does not describe the general problem of proving regulatory compliance, but it presents its most basic sub-class. This sub-class of the problem is obtained by using a regulatory framework which does not contain any of the difficult features. This means that the regulatory framework used by the basic problem contains a single global atomic obligation, which is an obligation whose activation is fixed to hold for the whole duration of the processes described in the model being analysed and no compensations are allowed in case the obligation is violated by the processes. Additionally the elements composing the obligation are expressed using only propositional literals and not formulae. This problem corresponds to the sub-class of the problem $C0^{-}_{1ga}$ in Figure 1.3 described in Section 1.3. In Chapter 4 I describe how the regulatory framework can be extended using the difficult features and I will gradually study more difficult problems by increasing the number of difficult features included in the regulatory framework. The

work being discussed in the present chapter has already been published [24] and has been coauthored with Marwane el Kharbili, Guido Governatori, Qin Ma, Pierre Kelsen and Leendert van der Torre.

3.1 The Business Process Models

In this first part of the chapter I introduce the semantics and the graphical representation of the *Business Process Models*. Business process models represent a collection of procedures capable of achieving a given goal. Therefore a company can use business process models to represent its possible ways of achieving one of its business objectives. For instance an automotive company can use a business process model to represent its ways of building a car. In this case, each plan contained in the business process model represents the sequence of activities necessary to build a car. These activities represent the basic operations available to an automotive company to achieve their business objective to build a car.

3.1.1 Structured Business Processes

In this thesis I limit the scope to a particular class of business processes: structured processes. Such class of business processes is similar to the structured workflows defined by Kiepuszewski et al. [49], which is limited in its expressivity with respect to other types of workflows because it only allows properly nested components. The structured business process I am introducing in this section have an additional constraints over the structured workflows of Kiepuszewski et al., which is being acyclic. From this follows that each component belonging to a structured business process can be executed at most once.

An advantage of using structured processes is that their soundness can be verified in polynomial time. A structured process model is sound, as defined by van der Aalst [2, 77], if it avoids livelocks and deadlocks, in other words a model whose computation cannot get stuck and terminates in a finite amount of steps. It is shown by van der Aalst et al. [75] that in general the soundness of structured workflows is not decidable if these workflows include inhibitor or reset transitions.

The models used in the present thesis are both structured and acyclic. This means that an execution of a process model cannot get stuck indefinitely in one of these cycles. Finally, concerning the limitation of not allowing cycles in the models, assuming that a process model involving cycles does not end up in livelocks, thanks for instance to specifying the maximum number of times that a cycle can be executed, it is then possible to approximate theses processes using models without cycles by explicitly representing in the model the various possible executions depending on the number of iterations of the cycle.

While not all business processes are structured, the structured processes are a substantial class of real-life processes. According to Polyvanyy et al. [59], 406 of the 604 processes in the SAP reference models [47] are structured. In addition Polyvanyy et al. [59] identify conditions under which unstructured processes can be transformed into structured ones,

and proposes an algorithm for the transformation. They also report that seventy-eight of the unstructured processes in the SAP reference models can be converted into behaviourally equivalent structured process models.

The structured processes defined here follow the semantics used by *Business Process Model and Notation 2.0*.[1] I define them by first defining the basic blocks constituting a structured process. The most basic element is the task; it abstractly represents an atomic operation that can be executed to help towards the achievement of the business objectives purposed by the business process model. The tasks can be then combined in more complex structures, like sequences, *and* blocks and *xor* blocks. In turn such structures can be used by nesting them to build more complex structures.

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

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

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

Using the process blocks introduced in Definition 1, it is then possible to define the structured business processes. These type of processes are defined by enclosing a process

[1] `http://www.omg.org/spec/BPMN/2.0`

block within two specific pseudo-tasks: the *start* and *end*, which are respectively placed before and after a process block to construct a structured business process.

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

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

In a structured process model XOR blocks and AND blocks have to be properly nested, meaning that if the block A starts inside the block B, A has to end within B. An example of a structured process model is shown in Fig. 3.1.

Example 9 (Structured Process Model)**.** *Fig. 3.1 shows a structured business process containing four tasks labelled $t_1, \ldots, t_4$. The structured process contains an* XOR *block delimited by the* xor_split *and the* xor_join*. The* XOR *block contains the tasks t_1 and t_2. The* XOR *block is itself nested inside an* AND *block with the task t_3. The* AND *block is preceded by the* start *and followed by task t_4 which in turn is followed by the* end*.*

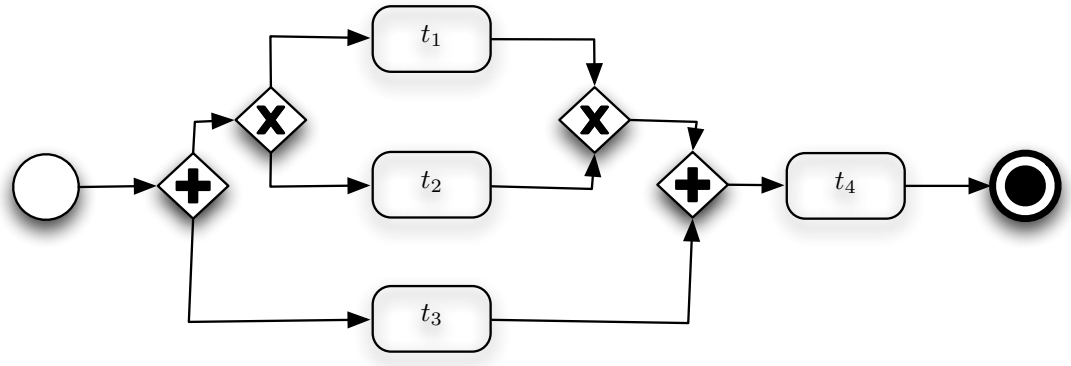

Figure 3.1: A structured business process

Considering the structured process in Fig. 3.1 as a sequence block, we can represent it as follows:

$$P = \mathsf{SEQ}(\mathsf{start}, \mathsf{SEQ}(\mathsf{AND}(\mathsf{XOR}(t_1, t_2), t_3), t_4), \mathsf{end})$$

The class of structured processes considered in the present thesis exclude business processes containing badly nested blocks (Fig. 3.2.(a)) and business processes with loops (Fig. 3.2.(b)).

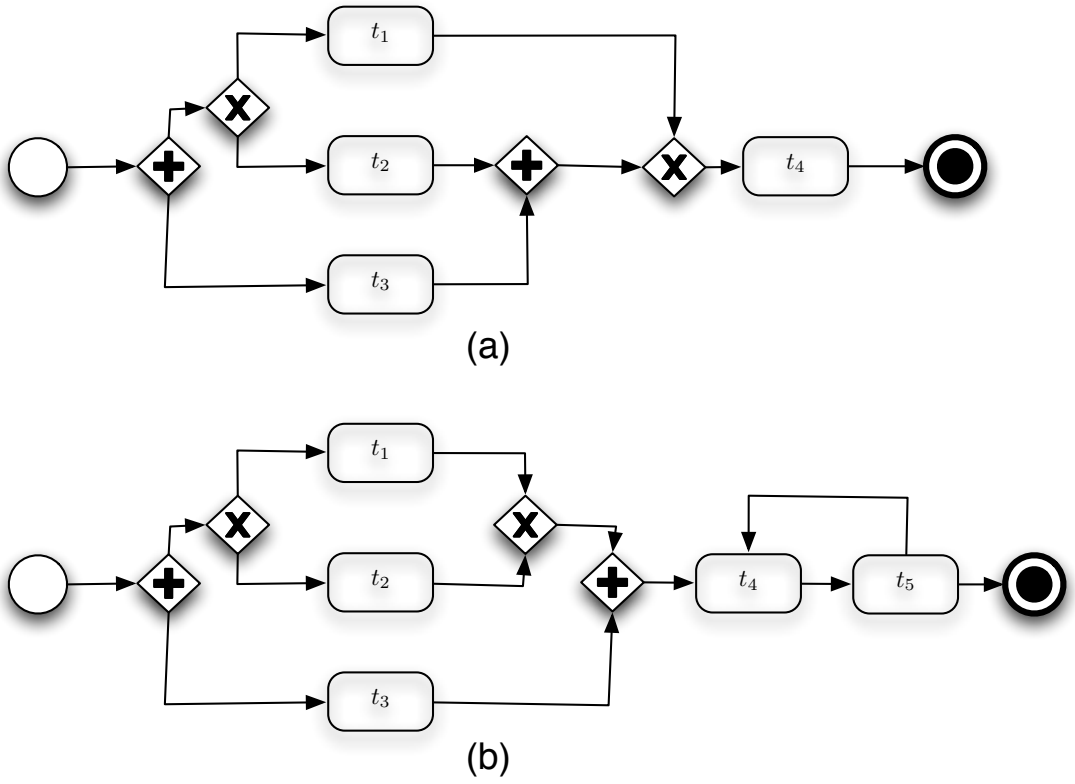

Figure 3.2: Examples of non-structured processes

An execution of a structured process model is a sequence of a subset of the tasks belonging to the model and it represents one of the processes that a company can use to achieve one of their business objectives. In a structured process model a valid execution identifies a path from the pseudo-task start to the pseudo-task end. A path is constructed by following the transitions between the elements in the model and following the semantics of the coordinators being traversed. The tasks traversed by a path are the ones being executed in the execution identified by that path.

I define the executions of the business process models using partial ordered sets. Before proceeding to define how an execution of a structured business process is created from the model, I recall the definition of partial ordered set which is used as an auxiliary concept in defining a serialisation of a process block, which corresponds to a possible execution, in other words to a process belonging to the model. In addition to recalling the definition of partial ordered set, I introduce some operations for this type of sets that are used to define the serialisations.

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

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

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

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

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

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

A serialisation of a process block is a sequence of a subset of the tasks, also known as a totally ordered set, contained in such process block. The sequence of tasks representing the serialisation has to follow the semantics of the coordinators contained in the process block. A serialisation of a main process block of a business process is also an execution of the business process itself.

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

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

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

Notice that the semantics of the serialisation of an AND block corresponds to the standard interleaving semantics for parallelism.

Corollary 1 (Process Block Serialisation)**.** *Given a process block B, t is a task in B, if and only if $\exists \epsilon \in \Sigma(B)$ such that $t \in \epsilon$.*

Proof (Process Block Serialisation). Proven by structural induction on the process block (definition 4).

□

Corollary 2. *Given a process block B, each serialisation ϵ such that $\epsilon \in \Sigma(B)$ is a finite sequence of tasks.*

Proof. Let B be a process block and $\epsilon \in \Sigma(B)$ be a serialisation of B. By construction of B we know that it contains a finite number of tasks. Thus it follows from Definition 4 that each task in B can appear at most once in ϵ. Therefore the amount of tasks in ϵ is bounded by the number of tasks in B which by construction is always finite. □

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

Definition 5 (Execution)**.** *Given a structured process P whose main process block is B, an execution of P corresponds to a serialisation of B. To help distinguishing executions from serialisations, I represent the former including the pseudo-tasks* start *and* end *as follows* $\Sigma(P) = \{\mathbb{P}_{\mathsf{start}} +_{\mathbb{P}} \epsilon +_{\mathbb{P}} \mathbb{P}_{\mathsf{end}} | \epsilon \in \Sigma(B)\}$. *However for compliance verification purposes, the executions are always considered without the pseudo-tasks* start *and* end*.*

According to Definition 5, each execution corresponds to a serialisation but the opposite is not true. Because only the serialisations of the main process block are considered as the executions of the process. The pseudo blocks Start and end are not considered while verifying the compliance of the process, since their main purpose is to identify the main process block and they do not contribute on describing how the *state of the process* evolves, representing the situation of executing the process at a given point in time.

Example 10 (Execution)**.** *The executions of the structured process illustrated in Fig. 3.3 are shown in the first column of Table 3.1.*

I refer with the term *universe* to the set of literals that can be used during the execution of a process to represent its states, and that can represent the effects of executing an activity.

Definition 6 (Universe $\mathcal{L}$)**.** *Given a* finite *set of atomic elements E, the universe $\mathcal{L}$ is $E \cup \{\neg e \mid e \in E\}$. For $e \in E$, let $\overline{a} = \neg e$ iff $a = e$ and $\overline{a} = e$ iff $a = \neg e$.*

The *state* of a process is represented using a set of literals which describe the situation depending on the meaning associated to the literals belonging to the set. Given that usually a process to complete takes time, it is often the case that its state changes multiple times during its execution. During the execution of a process the states are measured each time a task is executed to measure whether and how the state of the process has changed.

I assume in this thesis that the changes in the process state are not influenced by factors external to the process but depend on the tasks being executed during the process. How tasks influence the state is described using *annotations* [33] which are represented as states of literals associated to the tasks. An annotated process is a process whose tasks are associated with consistent sets of literals.

Both the state of a process and the annotations of the tasks are represented by sets of literals. These sets are required to be consistent, meaning that a particular literal cannot be both true and false in the same set. These literals describe the features of the state, a feature being present using a true literal and a feature missing using a false literal. Since it would not make sense for a feature to be both present and not, I restrict these sets to be consistent sets of literals.

Definition 7 (Consistent literal set)**.** *A set of literals L is* consistent *if and only if it does not contain both l and its complement $\overline{l}$ for each literal $l \in L$, where $\overline{l} = a$ if $l = \neg a$, or $\overline{l} = \neg a$ if $l = a$.*

Definition 8 (Annotated process)**.** *Let P be a structured process and let T be the set of tasks contained in P. An annotated process is a pair: (P, ann), where* ann *is a function associating to each task in T a consistent set of literals: $\mathsf{ann} : T \mapsto 2^{\mathcal{L}}$. The function* ann *is constrained to the consistent literals sets in $2^{\mathcal{L}}$.*

The case where a task is annotated with an empty set of literals is covered by the fact that the power set of literals includes the empty set.

Example 11 (Annotated Process Model)**.** *Fig. 3.3 shows a structured process containing four tasks labeled t_1, t_2, t_3 and t_4 and their annotations. The process contains an* AND *block followed by a task and an* XOR *block nested within the* AND *block. The annotations indicate what has to hold after a task is executed. If t_1 is executed, then the literal a has to hold in the state of the process.*

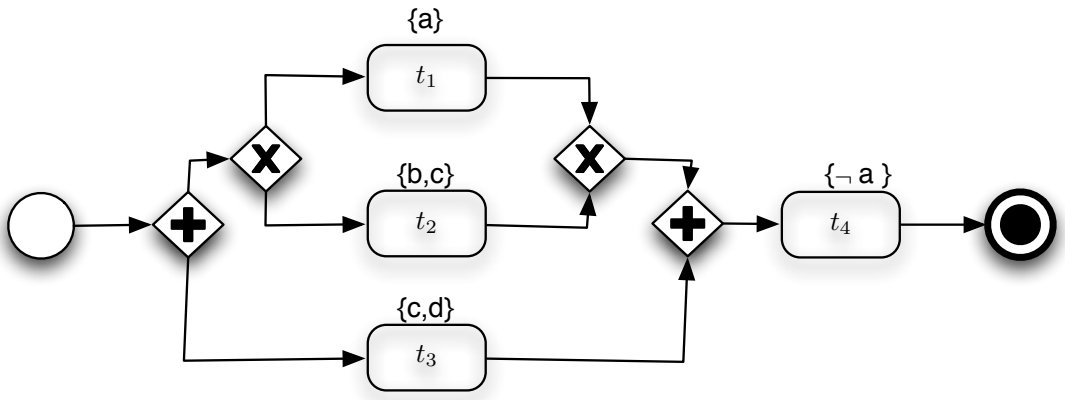

Figure 3.3: An annotated process

The state of the process is measured at some points in time corresponding to the executions of the tasks. Thus these states are represented by pairs, where one of the elements is a set of literals describing the state and the second the task after whose execution the state holds.

Definition 9 (State)**.** *Let $\mathcal{T} = \{t_1, t_2, \ldots\}$ be a set of tasks and L is a consistent literal set such that $L \subseteq \mathcal{L}$.*

A state is a tuple $\sigma = (t_i, L)$.

How the state of the process is updated by an execution of a task is described through an update operator. This operator has been inspired by the AGM belief revision operator [4] and is used in the context of business processes to define the transitions between states which in turn are used to define the *traces*.

Definition 10 (Literal set update)**.** *Given two consistent sets of literals L_1 and L_2, the update of L_1 with L_2, denoted by $L_1 \oplus L_2$ is a set of literals defined as follows:*

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

Finally, having defined how the state of a process is modelled and how the execution of a task can alter it, I can define the *trace*, which represents the evolution of the state of a process during one of its executions. It is represented by a sequence of states, holding at the different stages of an execution.

Definition 11 (Trace)**.** *Given an annotated process (P, ann) and an execution sequence $\epsilon = (t_1, \ldots, t_n)$ such that $\epsilon \in \Sigma(P)$. A trace θ is a finite sequence of states: $(\sigma_1, \ldots, \sigma_n)$. Each state of $\sigma_i \in \theta$ contains a set of literals L_i capturing what holds after the execution of a task t_i. Each L_i is a set of literals such that:*

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

Table 3.1: Executions and Traces of the annotated process in Fig. 3.3.

1. $L_1 = \mathsf{ann}(t_1)$;
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 an annotated process block (B, ann), *where* B *is a process block and* ann *is an annotation function, I use* $\Theta(B, \mathsf{ann})$.

Example 12. *Table 3.1 shows the traces of the annotated process* (P, ann) *illustrated in Fig. 3.3. The traces are shown in the second column and each of them is associated to an execution of the process* P. *It is possible to notice that the the order in which the tasks are executed in a trace is the same as the one appearing in the corresponding execution.*

From Table 3.1, in particular the column showing the traces we can see that the states of the process corresponding to the pseudo tasks start and end do not contribute to the information contained in the traces since the state associated to the pseudo task start is always the empty set and the state associated to end is always the same as the one holding when the previous task is executed. Therefore when verifying compliance considering the traces of a process, the states corresponding to the pseudo tasks start and end are not considered.

3.2 Regulatory Framework

In this second part of the chapter I introduce the regulatory framework. A regulatory framework describes through the use of obligations, which are the conditions that a process model needs to fulfil in order to be compliant with it. The regulatory framework introduced in this chapter captures the most basic compliance problem by adopting only the simple features available to define the obligations. The different sub-classes of the problem of proving regulatory compliance can be represented as the vertices of a cube where the different dimensions represent the available features for the obligations as shown in Figure 3.4. Moreover the figure highlights in which vertex the basic problem is situated according the the difficulty vectors describing the features and associated to the three spacial dimensions.

The most basic compliance problem focuses on proving whether a process is compliant with a single global atomic obligation and is named $C0_{1ga}$. An obligation is said to be

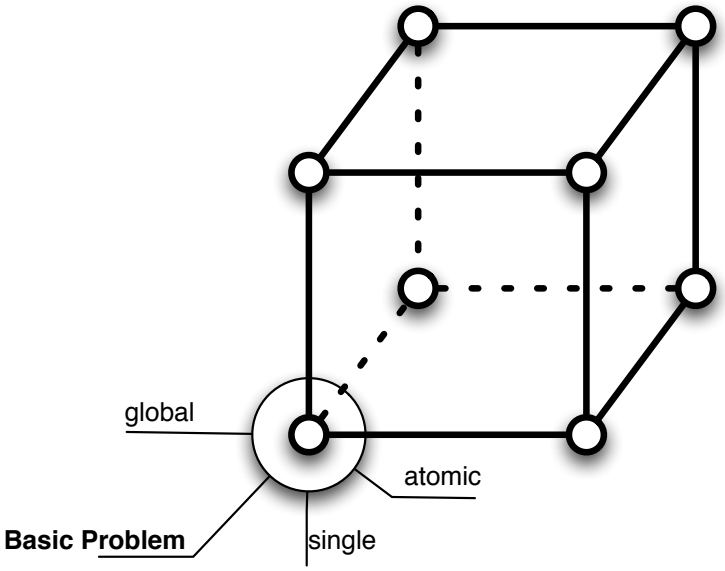

Figure 3.4: Basic Compliance Problem

global when its validity spans for the whole length of the traces belonging to the process being analysed. From now on I refer to the validity of an obligation as its activation interval, since it measures the intervals of a trace where the obligation is active.

The sub-class $C0_{1ga}$ is the simplest of the problems since in this case we do not have to deal with identifying the activation interval of the obligation using lifelines (a description of the states which activate an obligation) and deadlines (a description of the states which deactivate an active obligation), since the activation interval of global obligations is fixed to hold from the beginning to the end. Additionally it is easier to prove compliance with a single obligation rather than having to prove that the possible traces of a process model are compliant with multiple obligations. Finally by not considering compensations for violated obligations it is easier to prove or disprove compliance since there is no need of additional analysis in case a violation is detected.

In general the regulatory framework is composed by a set of obligations and is defined according to their semantics. In the case of the basic problem of proving regulatory compliance, the regulatory framework is defined by the semantics of the only global obligation contained in the set. Even though the framework is composed of a single obligation in this case, this obligation can be of different types with different semantics. Therefore I will introduce the syntax and the semantics of the different types of obligations used to define the framework.

After having defined the regulatory framework I define the problem of proving regulatory

compliance of a business process model (introduced in the first part of this chapter), which consists in verifying whether the possible traces of a process fulfil the obligations contained in the regulatory framework. This chapter concludes by providing algorithms of polynomial complexity with respect to the size of the business process model and the regulatory framework, capable of proving the compliance of a business process model with respect to a regulatory framework containing a single global obligation.

Before proceeding in introducing the framework I briefly discuss why *Standard Deontic Logic* would not be suited to deal with the problem at hand. Moreover, although *Dynamic Deontic Logic* would have been able to study the problem of proving the compliance of traces, I decided to adopt a simpler semantics to avoid to deal with the other features of dynamic deontic logic not necessary to handle the problem of proving regulatory compliance, such as for instance permissions and prohibitions.

3.2.1 Dealing with Traces

Normative reasoning is a field which focuses on representing and reasoning about obligations and regulations in general. One of the most popular logics used to represent the obligations and their semantics is standard deontic logic. Unfortunately this logic in its most basic form is used to evaluate obligations considering a single state, which does not fit well in the setting of proving regulatory compliance since the obligations used need to be evaluated over sequences of states, the traces (Definition 11).

Standard Deontic Logic

It is fair to say that deontic logic has been firstly introduced in 1951 by von Wright [81] as a system for reasoning about what is necessary or allowed. *Standard Deontic Logic* is one of the successors of the system introduced by von Wright. This branch of symbolic logic has been mostly concerned with reasoning about what is permitted and obligatory in a given context.

The syntax of this logic is composed by an infinite set of propositional variables, the classical logical operators: negation ($\neg$), logical and ($\wedge$), logical or ($\vee$), and material implication ($\rightarrow$). In addition to these operators, two modal operators $\mathcal{O}$ and $\mathcal{P}$ are used respectively to identify what is obligatory and what is permitted.

The semantics of Standard Deontic Logic is usually defined using *Kripke*'s semantics, also known as possible worlds. Informally, such semantics can be described using a tree where the root represents the current world and the children of the root are the possible worlds accessible from the current one. A world describes the state holding in that moment. If something is obligatory in the current world w_0, written $\mathcal{O}(\alpha)$, then $\forall w_i : w_i$ is a world accessible from $w_0, w_i \models \alpha$. In a similar way, when something is permitted in the current world, written $\mathcal{P}(\alpha)$, then $\exists w_i : w_i$ is a world accessible from $w_0, w_i \models \alpha$.

One can notice from the semantics of standard deontic logic that obligations and

permissions are evaluated in a single world or state describing a possible instant in the future. However as it is true in many real world scenarios, considering a single time instant is often not enough to decide whether an obligation has been fulfilled or violated. The following example illustrates one of these scenarios.

Example 13. *The authors of a paper want to submit it to a conference, hence they have to submit it before the submission deadline. This also means that the paper has to be finished before being submitted.*

The scenario contained in Example 13 illustrates a situation comprising an obligation for which considering a single state to evaluate it is often not sufficient. A state represent the situation of the world in a given time instant, like describing whether the paper has been submitted or not and whether the deadline has already passed or not. Analysing a single state is not always sufficient to verify whether the obligation of submitting the paper in time is fulfilled. For instance the state can contain both that the paper has been submitted and the deadline has passed, however in this case the information contained in the state is not sufficient since it does not tell whether the submission has been done before or after the deadline.

The traces used in the process of proving the compliance of processes in this thesis are very similar to the proposed example. The traces use states to define situations at a given point in time and the temporal relations between these states is given by the order in which they appear in the trace itself.

Dynamic Deontic Logic

To be able to handle cases like the one illustrated in Example 13 it is necessary to extend standard deontic logic. A variant of it has been introduced by Segerberg [67], capable of dealing with these scenarios and is known by the name of *dynamic deontic logic*.

Segerberg's dynamic deontic logic evaluates the obligation on structures very similar to the traces used in this thesis. These structures are called histories by Segerberg. In these histories ramifications are allowed in order to represent different possible future scenarios. Due to the ramifications representing these possible futures, these models resemble trees rather than the sequences used in this thesis. Segerberg's histories resemble Kripke models used in deontic logic.

The transitions between the states in the histories are described as a set of contributions changing a state to the following one in the model. These contributions are made by some agents and Segerberg uses different operators to distinguish whether one of these contributions is about to be done or has been done. In addition to the operators on the agent's actions, Segerberg uses deontic operators to define obligations, permissions and prohibitions, which in addition to temporal operators like until and always in the future, are expressive enough to be able to represent and verify properly situations like the one described in Example 13.

Dynamic deontic logic is subject to some limitations such as the agent's actions bringing a state to the following one in the histories cannot be simultaneous, meaning that the agents must act one at a time. These actions are considered to be atomic and cannot be interrupted. Epistemic concepts such as knowledge and beliefs are not used by the agents. Finally, no analysis is made to identify which agent contributed to the changes from a state to another, so it is not possible to identify which agent is responsible for a particular change in a history.

In this thesis I adopt a simpler semantics to describe and reason about the obligations. First of all the models used to represent the context, the trace of a business process model, are linear and not branching, meaning that we do not have to deal with multiple possible future states. Since the goal of the present thesis is to study the compliance problem, the semantics used focuses on obligations and leaves permissions out because they do not add information to the compliance checking process. Finally the approach used in this thesis does not take into account multiple agents contributing to the evolution of the context and, considering the model of the process as an agent, its actions (represented by the tasks in the model) are considered to be deterministic.

3.2.2 Obligations' Semantics

Since I am studying the problem of proving regulatory compliance, I will leave permissions out of the picture and focus on defining the semantics of the obligations. To specify them I use a subset of Process Compliance Logic (PCL) [36], introduced by Governatori and Rotolo.

Given that this chapter focuses on defining the abstract framework capturing the problem of proving regulatory compliance in the most basic case, I introduce in this section only the semantics of global obligations, which are obligations whose activation period is predetermined and holds for the entire duration of a process. Mind that the following definitions use literals to define the obligations, while in the more general setting propositional formulae can be used.

Definition 12 (Global Obligation)**.** *A global obligation* $\mathbf{O}$ *is a structure* $\langle t, l \rangle$*, where* $t \in \{a, m\}$ *and represents the type of the obligation. The element* l *is a literal belonging to* $\mathcal{L}$ *and represents the condition of the obligation.*

I use $\mathbf{O} = \mathcal{O}^t\langle l \rangle$ *to represent a global obligation.*

As mentioned previously, I distinguish two types of obligations: achievement ($\mathcal{O}^a$), which captures the semantics of the obligations like to the one in Example 13; and maintenance ($\mathcal{O}^m$), which captures the semantics of obligations similar to the one in Example 15.

Evaluating the Obligations

Whether an obligation is fulfilled or violated is determined by the trace being analysed. In particular the states composing the trace are analysed. As they have been defined in

Definition 9, the states composing a trace are not necessarily complete, meaning that given a literal l constituting the fulfilment condition of a global obligation, a state can either contain such literal, its negation ($\neg l$) or neither of them. Whether a state satisfies the condition of an obligation is represented through membership. In other words a state satisfies the condition of an obligation composed by a literal if and only if the state contains that particular literal.

Definition 13 (Literal Entailment)**.** *Given a state $\sigma = (t, L)$ and a literal l, $\sigma \models l$ if and only if $l \in L$.*

The following examples illustrate the two different types of obligations, achievement obligations which require to achieve a condition and maintenance obligations which require to maintain a condition.

Example 14 (Achievement Obligation)**.** *In a scenario where a customer dines in a restaurant, there is the obligation that the bill has to be payed before leaving. In this case we can picture the dining at a restaurant as a process and paying the bill as a global achievement obligation that has to be fulfilled once.*

To comply with a global achievement obligation, the condition has to be verified in at least one state of the trace. Consequently an achievement obligation is violated if no state of the trace satisfies the condition.

Example 15 (Maintenance Obligation)**.** *While accessing secure data there is the obligation to have the proper credentials for the whole period. In this case we can see the process as the operations being done on the secured data, terminating with the end of the access. Keeping the proper credential for the whole access can be seen as a global maintenance obligation.*

To comply with a global maintenance obligation, the condition of the obligation has to be verified in every state belonging to the trace. Consequently a global maintenance obligation is violated if a trace contains a state where the condition is not satisfied.

How to comply with the different types of obligation is formally described in the following definitions.

Definition 14 (Comply with Global Achievement)**.** *Given a global achievement obligation $\mathbf{O} = \mathcal{O}^a\langle l \rangle$ and a trace θ, θ is compliant with $\mathbf{O}$, written $\theta \vdash \mathbf{O}$, if and only if:*
$\exists \sigma \in \theta$ *such that* $\sigma \models l$.

Recalling the obligation expressed in Example 14, it can be expressed using a global achievement obligation whose condition is represented by "paying the bill".

Definition 15 (Comply with Global Maintenance)**.** *Given a global maintenance obligation $\mathbf{O} = \mathcal{O}^m\langle l \rangle$ and a trace θ, θ is compliant with $\mathbf{O}$, written $\theta \vdash \mathbf{O}$, if and only if:*
$\forall \sigma \in \theta$, *it is true that* $\sigma \models l$

The obligation presented in Example 15 can be expressed as a global maintenance obligation whose condition is represented by "having the proper credentials".

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. The regulatory framework is at the moment composed of a single obligation, hence complying with the obligation is equivalent to comply with the framework[3]. A process is partially compliant if there exists a trace compliant with the obligation. If none of the traces of a process are compliant with the obligation, then the process is not compliant.

Definition 16 (Process Compliance)**.** *Given a process* (P, ann) *and a global obligation* $\mathbf{O} = \mathcal{O}^t\langle l \rangle$*, the compliance of* (P, ann) *with respect to* $\mathcal{O}$ *is determined as follows:*

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

According to Definition 1, the minimal process block is composed of a single task. Thus, because a process contains at least a process block (Definition 2) and following from Definition 5, each process model contains at least one execution, hence at least one trace. Since the set of traces of a process model is never empty, it follows from the definition just provided that full compliance implies partial compliance. This means that if a process is fully compliant with an obligation, then such process is also partially compliant with the same obligation. These relations among the different compliance results are represented in Fig. 3.5 using *Venn Diagrams*.

3.3 Algorithms and Complexity

In this last part of the chapter I show that the problem $C0^{-}_{1ga}$, namely the problem of proving the compliance of a process with respect to a regulatory framework composed of a single global obligation which does not allow compensations and whose obligation is composed of a single literal, can be solved in polynomial time with respect to the size of the problem.

To do so I present here the algorithms to prove the compliance of a process with respect to a global obligation. Two different algorithms are designed to deal with the two different

[3]Later on I will introduce regulative frameworks composed of a set of obligations. In those cases being compliant with the framework is equivalent to being compliant with the set.

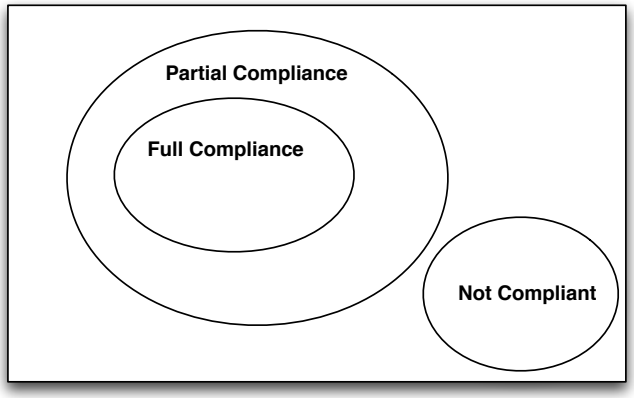

Figure 3.5: Relations between the compliance results (Definition 16)

types of obligations. For each of the two algorithms I show their correctness by proving that they are both sound and complete, and I furthermore show that their complexity is in time polynomial with respect to the size of the input, which is composed of the process model and the global obligation.

3.3.1 Algorithm for Global Achievement Obligations

The algorithms I am going to present in this part of the chapter use procedures to both keep their length under control and to decouple the different functions. This decoupling allows a better understanding of the purpose of these procedures and allows them to be reused in other algorithms when necessary.

The algorithm for achievement obligations uses the procedure *Task Removal*. This procedure is used to remove a set of tasks from a process block. By doing so the executions that contain these tasks are no longer allowed. In some cases by removing one or more tasks from a block it is possible that no executions remain possible. If this is the case the function does not return a process block but $\perp$. Notice that $\perp$ is not a process block but the process uses it as a *pseudo-block*, similarly as start and end are used as pseudo-tasks.

Procedure 1 (Task Removal)**.** *Given a process block B and a set of tasks $\mathcal{T} = \{t | t \in V(B)$ and t is a task$\}$,* task removal *$R(B, \mathcal{T})$ returns either a new process block B' or $\perp$ as follows:*

1: ***if*** *B is a task t* ***then***
2: ***if*** *$t \in \mathcal{T}$* ***then***
3: ***return*** *$\perp$*
4: ***else***
5: ***return*** *B*

6: ***end if***
7: ***end if***
8: ***if*** $B = \mathsf{SEQ}(B_1, \ldots, B_k)$ ***then***
9: ***if*** $\exists B_i, 1 \leq i \leq k$ *such that* $R(B_i, \mathcal{T}) = \perp$ ***then***
10: ***return*** $\perp$
11: ***else***
12: ***return*** $\mathsf{SEQ}(R(B_1, \mathcal{T}), \ldots, R(B_k, \mathcal{T}))$
13: ***end if***
14: ***end if***
15: ***if*** $B = \mathsf{XOR}(B_1, \ldots, B_k)$ ***then***
16: ***if*** $\forall B_i, 1 \leq i \leq k$, $R(B_i, \mathcal{T}) = \perp$ ***then***
17: ***return*** $\perp$
18: ***else***
19: ***if*** $\exists! B_i, 1 \leq i \leq k$ *such that* $R(B_i, \mathcal{T}) \neq \perp$ ***then***
20: ***return*** $R(B_i, \mathcal{T})$
21: ***else***
22: ***return*** $\mathsf{XOR}(R(B_{m_1}, \mathcal{T}), \ldots, R(B_{m_n}, \mathcal{T}))$
where $B_{m_j} \in \{B_1, \ldots, B_k\}$ *and* $R(B_{m_j}, \mathcal{T}) \neq \perp$
23: ***end if***
24: ***end if***
25: ***end if***
26: ***if*** $B = \mathsf{AND}(B_1, \ldots, B_k)$ ***then***
27: ***if*** $\exists B_i, 1 \leq i \leq k$ *such that* $R(B_i, \mathcal{T}) = \perp$ ***then***
28: ***return*** $\perp$
29: ***else***
30: ***return*** $\mathsf{AND}(R(B_1, \mathcal{T}), \ldots, R(B_k, \mathcal{T}))$
31: ***end if***
32: ***end if***

Depending on the type of process block being processed by the procedure, the procedure analyses it differently and returns either the block itself, part of it or $\perp$. The result depends of the result of some recursive calls of the procedure itself on some of the subblocks composing the block being analysed. The decision is driven by the presence of subblocks returning $\perp$. In the basic case, when the procedure is applied to a block composed of a single task, task removal returns $\perp$ if and only if the task belongs to the set $\mathcal{T}$.

Lemma 1 (Task Removal)**.** *Given a process block* B *and a set* $\mathcal{T}$ *of tasks* $\{t | t \in V(B)$ *and* t *is a task*$\}$*:*

1. *If* $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ *such that* $t \in \mathcal{T}$*, then* $R(B, \mathcal{T}) = \perp$
2. *If* $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ *such that* $t \in \mathcal{T}$*, then* $R(B, T) = B'$ *and* $\forall \epsilon' \in \Sigma(B'), \neg\exists t' \in \epsilon'$ *such that* $t' \in \mathcal{T}$

3. *If* $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ *such that* $t \in \mathcal{T}$*, then* $\forall \epsilon \in \Sigma(B), \neg\exists t \in \epsilon, \exists \epsilon' \in \Sigma(R(B, \mathcal{T}))$ *such that* $\epsilon \equiv \epsilon'$

The lemma just introduced highlights the properties of the process block returned by the procedure task removal. These properties describe that the resulting process block allows exactly the serialisations of the original block that do not contain tasks belonging to the set $\mathcal{T}$. In case all the serialisations of the initial block contain at least a task belonging to $\mathcal{T}$, then the procedure returns $\perp$.

The following example illustrates how the procedure is used to remove a set of tasks from a process model. The example illustrates two cases resulting from processing the main process block of the process. The first shows a case where the procedure returns a process block containing a subset of the serialisations that were possible in the main process block and the second a case where the result is $\perp$.

Example 16 (Applying Procedure 1)**.** *Let P be the process represented in Fig. 3.1 whose main process block is $B = \mathsf{SEQ}(\mathsf{AND}(\mathsf{XOR}(t_1, t_2), t_3), t_4)$ and let $\mathcal{T} = \{t_2\}$. The result of applying the procedure task removal $R(B, \mathcal{T})$ is the following process block: $B' = \mathsf{SEQ}(\mathsf{AND}(\mathsf{XOR}(t_1), t_3), t_4)$. In the result we can drop the* XOR *block identifier since it contains only the task t_1, hence its behaviour is the same as the one of a single task. Therefore we can rewrite the block as $B' = \mathsf{SEQ}(\mathsf{AND}(t_1, t_3), t_4)$.*

The rewritten version of B' is shown in Fig. 3.6. The available executions of the resulting process block are only (t_1, t_3, t_4) and (t_3, t_1, t_4), which are exactly the executions of B that are not containing t_2 as we can see from Table 3.1.

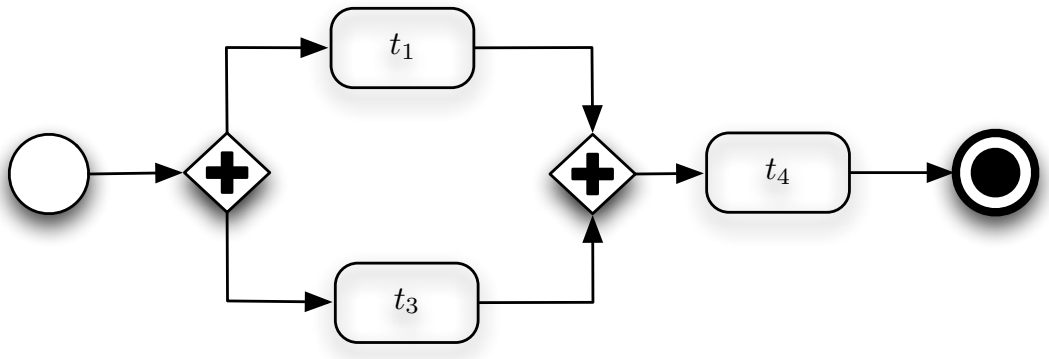

Figure 3.6: Resulting Process of applying Task Removal (Example 16)

If we consider the set of tasks $\mathcal{T} = \{t_4\}$ to be removed from the process block B using task removal. The result returned by the procedure is $\perp$. As we can see from Table 3.1,

t_4 belongs to each of the possible executions, hence if we remove it from the process block, no executions are possible anymore. The procedure recognises this by substituting $\perp$ to t_4, obtaining the process block $\mathsf{SEQ}(\mathsf{AND}(\mathsf{XOR}(t_1, t_2), t_3), \perp)$*, which in turn results in $\perp$ according to line 9 of the algorithm describing the procedure.*

Algorithm 1. *Given an annotated process (P, ann) and a global achievement obligation $O^a(l)$, this algorithm returns whether (P, ann) is compliant with $O^a(l)$.*

1: *Suppose* $P = \mathsf{start}\ B\ \mathsf{end}$
2: ***if*** $\forall t$ *in* $B, l \notin \mathsf{ann}(t)$ ***then***
3: ***return*** $(P, \mathsf{ann}) \not\vdash O^a(l)$
4: ***else***
5: ***if*** $R(B, \{t \mid t \text{ is a task in } B \text{ and } l \in \mathsf{ann}(t)\}) = \perp$ ***then***
6: ***return*** $(P, \mathsf{ann}) \vdash^F O^a(l)$
7: ***else***
8: ***return*** $(P, \mathsf{ann}) \vdash^P O^a(l)$
9: ***end if***
10: ***end if***

Following from the semantics in Definition 14, an achievement obligation $O^a(l)$ is satisfied when a task containing l in its annotation is executed. Therefore if an annotated process does not contain tasks having l in its annotation, then none of its possible traces is able to comply with obligation, hence the process is not compliant with respect to that obligation. However if the process contains at least a task having l in its annotation, then Algorithm 1 can decide whether the process is fully or partially compliant through the use of the procedure *task removal*. The procedure returns either a process block containing a subset of the original serialisations of the main process block of P or $\perp$ depending on the set of tasks used by the procedure. The algorithm feeds the procedure with a set of tasks containing every task that has l in its annotation. Therefore if the procedure returns $\perp$ it means that every execution of P contains at least one of those tasks, hence the process is fully compliant. Otherwise it means that some of the possible executions of the process do not contain these tasks and the algorithm classifies the process as partially compliant since some of its traces are compliant and some are not.

Notice that Algorithm 1 classifies a process to be partially compliant with a global achievement obligation in the case where the process is not also fully compliant with the same obligation. I opted for this choice in order to distinguish when a process is exactly partially compliant from when a process is partially compliant because of being fully compliant as it follows from Definition 16 and can be clearly seen in Figure 3.5.

Example 17 (Applying Algorithm 1)**.** *Let (P, ann) be the process shown in Fig. 3.3. Algorithm 1 is capable of proving whether the process is compliant with a single atomic global achievement obligation. Lets assume now that we want to prove whether (P, ann) is compliant with $O^a(e)$. We can see that the condition at line 2 of the algorithm is satisfied*

since none of the tasks of P contain the literal e in their annotations. Thus line 3 of the algorithm is executed and the result $(P, \mathsf{ann}) \not\vdash O^a(e)$ is returned. We can immediately see that the returned solution is correct since the process has no possible execution capable to achieve the condition of the obligation.

Lets assume now that we want to prove whether (P, ann) is compliant with $O^a(\overline{a})$. In this case Algorithm 1 returns full compliance, $(P, \mathsf{ann}) \vdash^F O^a(\overline{a})$. Because t_4 is the only task containing $\overline{a}$ in its annotation, the procedure Task Removal *applied at line 5 is the following: $R(E, \{t_4\})$. The result of the procedure is $\perp$ because there are no executions of B which do not contain t_4 as can be seen in Table 3.1. Thus the algorithm returns $(P, \mathsf{ann}) \vdash^F O^a(\overline{a})$ according to line 6.*

Finally, assuming that we want to prove the compliance of (P, ann) with respect to the achievement obligation $O^a(a)$. In the case the conditions at lines 2 and 5 of the algorithm are not satisfied: the condition at line 2 is not satisfied because t_1 contains a in its annotation; the condition at line 5 is also not satisfied because P contains some executions without t_1 as can be seen in Table 3.1, hence applying the procedure task removal *does not return $\perp$. Thus line 8 of the algorithm is executed and the returned result is $(P, \mathsf{ann}) \vdash^P O^a(a)$.*

To show that solving this particular sub-class of proving regulatory compliance is in **P**, I have to show that the complexity of Algorithm 1 is in time polynomial with respect to the size of the input, which consists of the process model and the atomic global achievement obligation.

Complexity of Algorithm 1

The block structure of a process can be represented by a tree, which I will refer to as *process tree*. The root of a process tree consists of the process' main block. The children nodes of a node in the process tree correspond to the subblocks of the block represented by the parent node. Finally the leaves of a process tree represent the tasks of the process.

Notice that the complexity of the Algorithm 1 is dominated by the complexity of the procedure task removal. Each call of the procedure task removal can be associated to a node in the process tree. By the recursive definition of the procedure task removal, the time required for combining the results of the calls for the subblocks is at most proportional to the number of children subblocks. Since the complexity of the calls at the level of the leaf nodes of the process tree is $\mathbf{O}(1)$, the overall complexity of task removal is at most proportional to the size of the process tree, which is linear in the size of the process. Overall I conclude that the complexity of Algorithm 1 is at most linear in the size of the process (including the annotations).

3.3.2 Algorithm for Global Maintenance Obligations

The algorithm for proving whether a process is compliant with an atomic global maintenance obligation uses some additional procedure besides task removal already introduced and

used by the algorithm to prove compliance with an achievement obligation.

Therefore before introducing the algorithm itself I am introducing the various procedures used by it. These procedures are *First*, used to identify which are the tasks that can be executed as first ones in at least one of the possible executions, and *Task Rooting*, that allows to consider subsets of the possible executions depending on the first task being executed

I first introduce the procedure *First*. The objective of this procedure is to identify which are the tasks that have the possibility of being executed as first ones in one of the possible executions of the process.

Procedure 2 (First)**.** *Given a process block B,* $\mathsf{First}(B)$ *returns a set of tasks as follows:*

1: ***if*** $E = t$ ***then***
2: ***return*** $\{t\}$
3: ***end if***
4: ***if*** $E = \mathsf{SEQ}(B_1, \ldots, B_k)$ ***then***
5: ***return*** $\mathsf{First}(B_1)$
6: ***end if***
7: ***if*** $E = \mathsf{AND}(B_1, \ldots, B_k)$ ***then***
8: ***return*** $\bigcup_{i=1}^{k} \mathsf{First}(B_i)$
9: ***end if***
10: ***if*** $E = \mathsf{XOR}(B_1, \ldots, B_k)$ ***then***
11: ***return*** $\bigcup_{i=1}^{k} \mathsf{First}(B_i)$
12: ***end if***

Since a global maintenance obligation's activation spans for the whole length of a trace, identifying the first tasks being executed becomes relevant because it must already update the state in such a way to comply with the condition of the obligation. Otherwise the trace is surely not compliant with the obligation.

Lemma 2 (First)**.** *Let the function $f(\epsilon)$ return the first task of the sequence corresponding to the serialisation ϵ. The set returned by the procedure* First Tasks *fulfils the following condition: $t \in First(B)$ if and only if $\exists \epsilon \in \Sigma(B) | f(\epsilon) = t$.*

Example 18 (Applying Procedure 2)**.** *Let P be the process in Fig. 3.1 and $B =$ $\mathsf{SEQ}(\mathsf{AND}(\mathsf{XOR}(t_1, t_2), t_3), t_4)$ be the main process block contained in P.*

The procedure in its first step, since B is a sequence block, recursively recalls itself with the following input: $\mathsf{AND}(\mathsf{XOR}(t_1, t_2), t_3)$*. In the second step the process block is an and block, hence the block is split in* $\mathsf{XOR}(t_1, t_2)$ *and t_3. The procedure is then called again on both subblocks and since t_3 is a single task we can consider it as part of the final result. In the third step we consider the block* $\mathsf{XOR}(t_1, t_2)$*, which in the same way as an and block is split. The resulting subblocks, in this case two tasks are processed by the procedure again and become part of the solution which for B is $\{t_1, t_2, t_3\}$.*

Therefore applying the procedure First *to B returns the set $\{t_1, t_2, t_3\}$ which, as can be seen from Table 3.1, contains all the tasks that appear as the first one in the executions of B.*

The second procedure used by the algorithm is *Task Rooting*. This procedure, given a block B and a task $t \in \mathsf{First}(B)$, identifies the subset of serialisations in $\Sigma(B)$ which have t as the first task being executed by returning a modified version of the block B whose serialisations are those starting with t. The process block returned by procedure task rooting does not always contain every execution from the original block starting with t. However I show with the help of Lemma 3 that the approximation does not affect the result of checking compliance for maintenance obligations.

Procedure 3 (Task Rooting)**.** *Given a process block B and a task $t \in \mathsf{First}(B)$,* task rooting *$F(B,t)$ returns a new process block as follows:*

1: ***if*** $B = t$ ***then***
2: ***return*** B
3: ***end if***
4: ***if*** $B = \mathsf{SEQ}(B_1, \ldots, B_k)$ ***then***
5: ***return*** $\mathsf{SEQ}(F(B_1, t), B_2, \ldots, B_k)$
6: ***end if***
7: ***if*** $B = \mathsf{XOR}(B_1, \ldots, B_k)$ ***then***
8: ***return*** $F(B_p, t)$ *where* $B_p \in \{B_1, \ldots, B_k\}$ *and* $t \in B_p$
9: ***end if***
10: ***if*** $B = \mathsf{AND}(B_1, \ldots, B_k)$ ***then***
11: ***if*** $k = 2$ ***then***
12: ***return*** $\mathsf{SEQ}(F(B_p, t), B_q)$, *where* $\{p, q\} = \{1, 2\}$ *and* $t \in B_p$
13: ***end if***
14: ***return*** $\mathsf{SEQ}(F(B_p, t), \mathsf{AND}(B_{i_1}, \ldots, B_{i_{k-1}}))$, *where* $\{i_1, \ldots, i_{k-1}, p\} = \{1, \ldots, k\}$ *and* $t \in B_p$
15: ***end if***

Lemma 3 (Task Rooting)**.** *Let B be a process block, t be a task such that $t \in \mathsf{First}(B)$, $\Sigma_t(B)$ be the set of executions of B that start with t and $\Theta_t(B, \mathsf{ann})$ be the set of traces associated to $\Sigma_t(B)$ given an annotation function* ann*. The procedure task rooting, $F(B,t)$, is subject to the following properties:*

1. $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$
2. $\forall \epsilon \in \Sigma_t(B), \exists \epsilon' \in \Sigma(F(B,t))$ *such that:*
   (a) $f(\epsilon) = f(\epsilon') = t$
   (b) *The task set of* ϵ = *the task set of* ϵ'

The term "task set" used in Lemma 3 refers to set of tasks appearing in an execution. Thus the condition 2.(b) of the lemma states that the two executions execute the same tasks but the execution order may be different.

Lemma 4 (Task Rooting Approximation)**.** *Let ϵ_1 and ϵ_2 be two executions containing the same task set and $f(\epsilon_1) = f(\epsilon_2)$. Let θ_1 and θ_2 be the traces corresponding to the two executions. Given a maintenance obligation $O^m(l)$, $\theta_1 \vdash O^m(l)$ iff $\theta_2 \vdash O^m(l)$.*

The intuition behind this lemma follows from the properties expressed in Lemma 3. These properties state that even if the approximation does not consider some of the possible executions, for each possible execution, there exists one execution among the approximated set that executes exactly the same set of tasks. Therefore, assuming that we fix with task rooting a starting task that makes the condition of a global maintenance obligation true, then the execution order of the other tasks is not relevant since if one would invalid the condition it does not matter when it is executed.

Example 19 (Applying Procedure 3)**.** *Let P be the process in Fig. 3.1 and $B = \mathsf{SEQ}(\mathsf{AND}(\mathsf{XOR}(t_1, t_2), t_3), t_4)$ be its main process block. Considering the set of tasks $\{t_1, t_2, t_3\}$ resulting from Example 11 I show how the procedure* Task Rooting *defines the resulting process block for the two following cases: $F(B, t_1)$ and $F(B, t_3)$.*

For $F(B, t_1)$, since B is a sequence, then line 5 is executed returning the following: $\mathsf{SEQ}(F(\mathsf{AND}(\mathsf{XOR}(t_1, t_2), t_3), \underline{t_1}), t_4)$. I underline t_1 to mark it as part of the input of F and distinguish it from the tasks of B. The block given in input to F is now an and block, hence the procedure returns $\mathsf{SEQ}(F(\mathsf{XOR}(t_1, t_2), \underline{t_1}), t_3)$ because $t_1 \in \mathsf{XOR}(t_1, t_2)$. At this step the block given in input to F is an xor block, hence the procedure returns $F(t_1, \underline{t_1})$, which returns t_1 since the input is a block composed of a single task. The result of the recursive calls is the following process block: $\mathsf{SEQ}(t_1, t_3, t_4)$, which is shown as a process in Fig. 3.7.

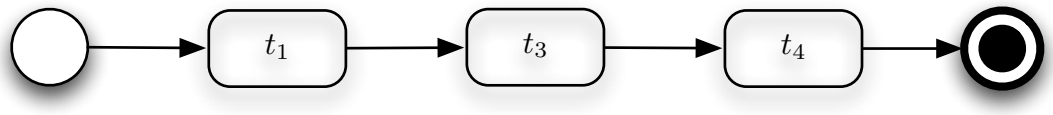

Figure 3.7: Resulting Process of applying Task Rooting with t_1 (Example 19)

For $F(B, t_3)$ the first step is the same as for $F(B, t_1)$, the main difference appears in the second step where the procedure returns $\mathsf{SEQ}(F(t_3, \underline{t_3}), \mathsf{XOR}(t_1, t_2))$. Since F takes now as input a process block containing a single task, then t_3 is returned. The result for $F(B, t_3)$ is the process block: $\mathsf{SEQ}(t_3, \mathsf{XOR}(t_1, t_2), t_4)$, shown as a process in Fig. 3.8.

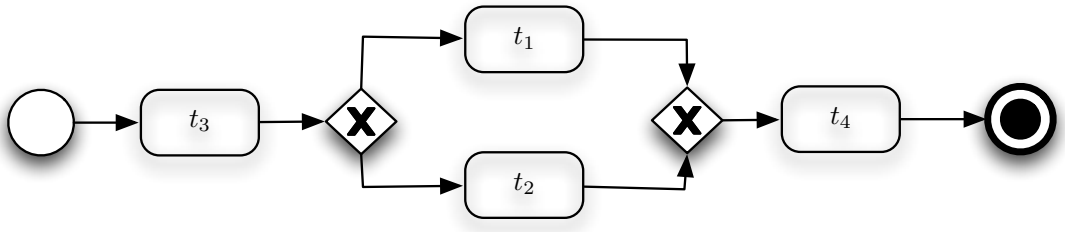

Figure 3.8: Resulting Process of applying Task Rooting with t_3 (Example 19)

Having introduced the auxiliary procedures *first* and *task rooting*, I now introduce Algorithm 2, which is capable of proving whether a process is fully, partially or not compliant with an atomic global maintenance obligation.

Algorithm 2. *Given a process* (P, ann) *and a global maintenance obligation* $O^m(l)$*, this algorithm returns whether* (P, ann) *is compliant with* $O^m(l)$*.*

Suppose $P = \mathsf{start}\ B\ \mathsf{end}$
$\mathcal{T}_F = \mathsf{First}(B)$
$\mathcal{T}_{\bar{l}} = \{t | t$ *is a task in* B *and* $\bar{l} \in \mathsf{ann}(t)\}$
if $\forall t$ *in* $\mathcal{T}_F, l \in \mathsf{ann}(t)$ *and* $\mathcal{T}_{\bar{l}} = \emptyset$ ***then***
  return $(P, \mathsf{ann}) \vdash^F O^m(l)$
else
  for each $t \in \mathcal{T}_F$ *such that* $l \in \mathsf{ann}(t)$ ***do***
    if $R(F(B,t), \mathcal{T}_{\bar{l}}) \neq \bot$ ***then***
      return $(P, \mathsf{ann}) \vdash^P O^m(l)$
    end if
  end for each
  return $(P, \mathsf{ann}) \nvdash O^m(l)$
end if

Algorithm 2 first identifies the set of tasks that can appear at first in the possible executions of the process by using the procedure *first*. The algorithm later identifies, using the procedure *task rooting*, which executions have the possibility to be compliant by starting with a task having l annotated. Finally the algorithm analyses these executions using the procedure *task removal* to verify whether existing tasks with $\bar{l}$ annotated would affect the compliance of these executions.

Notice that also for this algorithm, as for Algorithm 1, it returns $(P, \mathsf{ann}) \vdash^{\mathrm{P}} O^m(l)$ only in the case the process is partially compliant with the obligation but not fully compliant with it.

Example 20 (Applying Algorithm 2)**.** *Let P be the process shown in Fig. 3.2. Lets prove the compliance of P with respect to the maintenance obligation $O^m(c)$ using Algorithm 2. At line 2 $\mathcal{T}_F = \{t_1, t_2, t_3\}$ (as explained in Example 18). At line 3 $\mathcal{T}_{\bar{l}} = \emptyset$ because none of the tasks of P have annotated $\overline{c}$. The condition at line 4 is not satisfied because $t_1 \in \mathcal{T}_F$ does not contain c in its annotation. The for each cycle at line 7 is executed for t_2 and t_3. However because $\mathcal{T}_{\bar{l}}$ is empty, then line 8 is always satisfied and the algorithm returns $(P, \mathsf{ann}) \vdash^P O^m(c)$. The result can be verified from Table 3.1, where it is possible to see that every trace contains c in its states (excluding the states associated to the pseudo tasks), except for the first one.*

Differently, if we try to prove the compliance of P with the maintenance obligation $O^m(a)$, then the set at line 2 is still $\mathcal{T}_F = \{t_1, t_2, t_3\}$ because the structure of the process analysed is the same. However in this case we have that $\mathcal{T}_{\bar{l}} = \{t_4\}$ because $\mathsf{ann}(t_4) = \{\neg a\}$. Line 4 is again not satisfied in this case, because both $\mathcal{T}_{\bar{l}}$ is not empty and $t_2, t_3 \in \mathcal{T}_F$ do not contain a in their annotation. The for each cycle at line 7 is executed only for t_1. However, because t_4, having $\overline{a}$ in its annotation, belongs to each execution of P and belongs to $\mathcal{T}_{\bar{l}}$, it follows that $R(F(B, t_1), \{t_4\}) = \bot$, hence line 9 is never executed and the algorithm reaches line 12 returning $(P, \mathsf{ann}) \nvdash O^m(a)$. Again the result can be verified from Table 3.1, where it is possible to see that there is no trace where a holds in each of its states.

Complexity of Algorithm 2

The procedures First, R and F can be computed in time $\mathbf{O}(n)$ where n is the number of tasks in B. This has already been shown for task removal R in the complexity proof for Algorithm 1; the same can be shown for First and F using a similar argumentation based on the process tree (see complexity proof of Algorithm 1). Thus the call to R on line 8 can also be computed in time $\mathbf{O}(n)$. Assuming each annotation has size $\mathbf{O}(1)$ we then see that the overall complexity is $\mathbf{O}(n^2)$.

3.3.3 Applying the Algorithms

In this chapter I propose two algorithms capable of verifying in polynomial time whether an annotated structured process is compliant with a global obligation. Despite the restricting characteristics of this kind of obligation, we show in this section how some of the regulations used in real world scenarios can be modelled as global obligations.

In particular I show how two regulations extracted from the *Telecommunications Consumer Protections Code*[4] of 2012 can be translated into global obligations. The aim of the regulations contained in this code is to ensure good service and fair outcomes for all consumers of telecommunications products in Australia.

[4]Telecommunications Consumer Protections Code: `http://www.acma.gov.au/Industry/Telco/Reconnecting-the-customer/TCP-code/the-tcp-code-telecommunications-consumer-protections-code-acma`

3.3.4 Global Achievement Obligation

We consider now a regulation extracted from the Telecommunications Consumer Protections Code describing how complaints should be handled. We do not quote the entire regulation but only the fragment we focus on.

> **8.1 Provision of a Complaint handling process that is accessible, transparent and free of charge** Suppliers must provide Consumers with a Complaint handling process that:
>
> **(a)** is accessible, transparent and easily understood by Consumers and former Customers;
>
> **(b)** is free of charge, other than as expressly provided for in this chapter; and
>
> **(c)** provides for the courteous, timely and fair Resolution of Complaints.
>
> 8.1.1 A Supplier must take the following actions to enable this outcome:
>
> [. . .]
>
> **(d) Monitor and report**: formally monitor and report
>
> **(i)** at least annually to the Chief Executive Officer (or equivalent) regarding their compliance with their Complaint handling process and opportunities for improvement; and
>
> **(ii)** . . .

In this case let me focus on clause $(d).(i)$ of 8.1.1. This clause prescribes the monitoring of the complaint handling procedures and reporting to the executive officer at least once a year. This clause can be seen as a global achievement obligation where the condition, reporting to the executive officer, has to be achieved at least once. We can formally write the global obligation as $O^a(rep)$ where rep represents a report being sent to the executive officer, again assuming that the execution of the process spans the time of a year..

Let (P, ann) be an annotated structured process model describing the complaint handling monitoring process over a year. We can use the global achievement obligation $O^a(rep)$ and verify whether (P, ann) is compliant with it. To do so, according to the semantics of achievement obligations, Definition 14, we require that an execution $\epsilon \in \Sigma(P)$ executes a task of P whose annotation contains rep in order to fulfil the obligation. Moreover, given the assumption that the process model represents the complaint handling procedure spanning a year, we are verifying the compliance with respect to the regulation in clause $(d).(i)$ of 8.1.1.

Algorithm 1 makes use of Procedure 1 to verify whether the process is fully, partially or not compliant with the global achievement obligation. The procedure removes from the process model any task t where $rep \in \mathsf{ann}(t)$. In other terms it removes the tasks that would fulfil the obligation $O^a(rep)$. The algorithm then verifies whether the resulting process

model still contains some valid executions, that it executions not including any of the tasks removed, which in turn would not fulfil the obligation. Therefore the algorithm is able to classify the process as fully compliant if no valid executions are available, not compliant if no tasks were removed by Procedure 1, and partially compliant otherwise, namely in the case where some task were removed but there were still some possible executions of P not containing them.

3.3.5 Global Maintenance Obligation

We consider now a fragment of another regulation tailored to deal with the complaint handling process. The fragment considered here aims at ensuring that the customers complaints are handled properly and efficiently.

> **8.4 Resourcing of Complaint handling processes** Suppliers must ensure that their Complaint handling processes are resourced to ensure that they will meet their obligations under this Code.
>
> 8.4.1 A Supplier must take the following actions to enable this outcome:
>
> [. . .]
>
> **(c)** Skilled staff: ensure staff with Complaints management responsibilities are trained and supervised, have the authority to Resolve Complaints and have the necessary interpersonal and communication skills, to ensure that the Supplier's obligations under this chapter are met;
>
> **(d)** . . .

In this case, let me focus on clause (c) of 8.4.1 which requires that the complaints are handled by competent staff. This requirement can be seen as a maintenance obligation stating that complaints must not be handled by unskilled personnel. This global obligation can be written as $O^m(skill)$ where *skill* represents that a skilled employee handles the complaint being processed.

Let (P, ann) be an annotated structured process model describing the procedures adopted by a company to handle complaints. Our framework can verify whether the process is compliant with the global maintenance obligation $O^m(skill)$. According to the semantics of a maintenance obligation described in Definition 15, a trace of the annotated process model (P, ann) must contain the proposition *skill* in each of the states composing it in order to fulfil the obligation. This requirement captures the fact that a complaint must be handled by a skilled worker as requested by the regulation.

Algorithm 2 can be used to verify whether the business process model (P, ann) is compliant with the global maintenance obligation $O^m(skill)$ and to which extent. The algorithm uses some auxiliary procedures. The first being used is Procedure 2 which identifies what are the tasks that can appear as first task in the possible serialisations of P.

Since the obligation requires that each state of a trace resulting from the model contains the condition, in this case *skill*, and because we assume that the state describing the state of a process before it starts is always the empty state, it is then necessary that the first task being executed contains *skill* for a trace to have a possibility to fulfil the obligation. The algorithm verifies at this stage which of the tasks returned by Procedure 2 are annotated with *skill*, if each of these tasks is annotated with *skill* and tasks in (P, ann) annotated with $\neg skill$ do not exist, then the algorithm returns full compliance since it is guaranteed that every state of every trace of the model contains the required proposition to fulfil the obligation.

However, if this is not the case, then the algorithm needs to verify whether for each trace where *skill* holds in the first state, *skill* is not removed in the following states. This analysis is done by firstly applying Procedure 3 which considers only the executions starting with one of the tasks identified in the previous step, in this case one of the initial tasks annotated by the proposition *skill*. At this stage Procedure 1 is applied and every task containing $\neg skill$ is removed from the model. Then the algorithm verifies whether there possible executions after the removal of the task still exist. If this is the case, then the algorithm returns partial compliance since there exists at least a trace where *skill* holds in each of its states. Otherwise if no execution is available then the algorithm analyses a new configuration of the model, where another task whose annotation contains *skill* is forced to appear as first in every execution.

Finally the algorithm returns non compliance in case everything else failed, meaning that none of the possible traces of (P, ann) contain *skill* in each of their states.

3.4 Summary

In this chapter I firstly introduced the syntax and the semantics of the annotated business process models used in the thesis to represent the processes. These models are represented using a simplified version of *Business Process Model and Notation 2.0* and follow the construction rules of *structured workflow models*, which means that the blocks composing these models must be properly nested.

In the second part of this chapter I introduce the regulatory framework, which along with the process model describes the *basic problem* of proving regulatory compliance. This problem consists in proving the compliance of process models with respect to a single atomic global obligation whose condition is expressed using a literal.

In the last part of the chapter I introduce two algorithms, one for each type of global obligation: achievement and maintenance, which are capable of proving the compliance of a process in the context of the basic problem in time polynomial with respect to the size of the problem. Therefore I can now claim that the problem $C0^{-}_{1ga}$ is tractable, meaning that the computational complexity of the problem does not risk to explode by increasing the size of it. Finally in this last part I also show some real world scenarios where global

obligations are actually present and I show how the algorithms introduced can be used to verify the compliance of processes with respect to these obligations.

Chapter 4

Difficulty Vectors and Conflicting Obligations

In this chapter I discuss in more details the features of the regulatory framework that allows to define the different sub-classes of the problem of proving regulatory compliance. Recalling these features, they regulate the amount of obligations that can be contained in the regulatory framework, whether the obligations are conditionally activated and whether compensations are allowed in case of violations. I formally describe the difficult options of the difficulty vectors in the first part of this chapter.

By introducing the option of using multiple obligations in the regulatory framework, it becomes important to avoid having conflicting obligations belonging to the same framework. With conflicting obligations I refer to pairs of obligations where complying with one of them necessarily implies violating the other one. Thus a regulatory framework containing conflicting obligations cannot be complied with. Therefore I am discussing in the second part of this chapter about how to identify whether two obligations are in conflict. This is done by specifying the sufficient and necessary conditions, which can be then used while designing a regulatory framework to avoid conflicting obligations. The second part of the present chapter is based on an existing work coauthored with Guido Governatori and Pierre Kelsen [23].

4.1 Difficulty Vectors

The first part of this chapter discusses the difficulty vectors by introducing their difficult options. These features can be used to identify non basic problems by extending the basic problem introduced in Chapter 3.

In Figure 4.1 are shown the problems reachable by employing exactly one of the difficult features. These problems are referred to as $C1$, where the digit refers to the amount of difficult features used. Recalling the difficulty vectors, these are: global-local, single-multiple

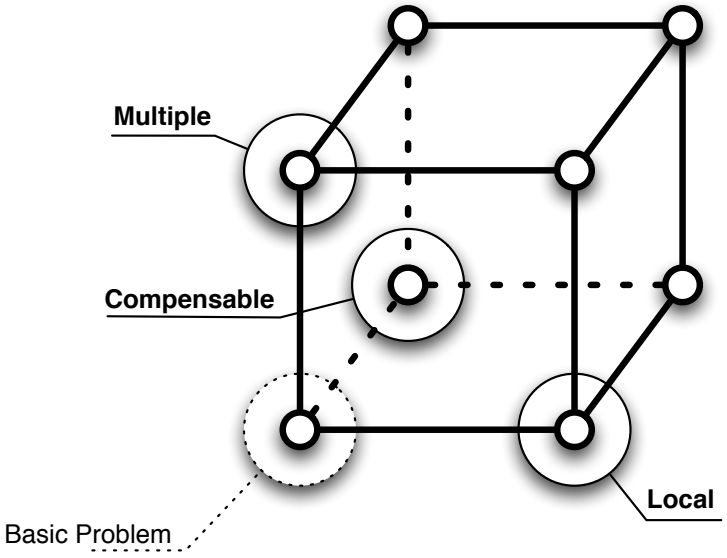

Figure 4.1: The three Non-Basic Compliance Problems

and atomic-compensable which, starting from the basic problem $C0^{-}_{1ga}$, respectively lead to the sub problems $C1^{-}_{1la}$, $C1^{-}_{nga}$ and $C1^{-}_{1gc}$.

I describe in this section the semantics of the different regulatory frameworks obtainable by increasing the complexity of the obligations navigating exactly one of the difficulty vectors at a time. More complex frameworks can be obtained by combining multiple difficult features and will be discussed and analysed in the following chapters of this thesis.

4.1.1 Fulfilling Local Obligations

The first difficulty vector I consider is the global-local vector. By starting from the sub-class $C0^{-}_{1ga}$ of the problem and moving from global obligations to local obligations we obtain the sub-class $C1^{-}_{1la}$ of the problem by introducing two elements into the obligations, defining the activation periods of each obligation. This differs from global obligations, where the activation period is given and equal for each global obligation, in other words the activation period of a global obligation always spans for the entirety of each execution. In local obligations the activation periods over an execution are a subset of disjoint subsequences of that execution. These subsequences are identified by the two new elements introduced: the lifeline defining when an activation period of a local obligation begins and the deadline defining when an activation period terminates.

Definition 17 (Local Obligation)**.** *A local obligation* $\mathbf{O}$ *is a structure* $\langle t, c, l, d \rangle$*, where* $t \in$

$\{s, a, m\}$ and represents the type of the obligation. The elements c, l and d are propositional literals in $\mathcal{L}$. c is the content of the obligation, l is the trigger (lifeline) of the obligation and d is the deadline of the obligation.

I use the notation $\mathbf{O} = \mathcal{O}^t\langle c, l, d\rangle$ to represent a local obligation.

A local obligations can be of three types, two of these types, achievement and maintenance, are the same types used by global obligations. The additional type introduced is the *standard*, which aims at replicating the semantics of obligations expressed in standard deontic logic. This last type requires that the condition of a local obligation is satisfied in exactly one state, which corresponds to the duration of its activation periods.

Definition 18 (Deadline)**.** *Let $\mathcal{O}^t\langle c, l, d\rangle$ be a local obligation and θ a trace where σ_{end} is the last state of θ. Regardless of the literals holding in σ_{end}, $\sigma_{end} \models d$ always.*

This definition presents an exception concerning the fulfilling of deadlines. The exception consists in that the last state of a trace always fulfils a deadline independently from which literals hold in the state. This exception is necessary since in some cases to evaluate an obligation its activation period needs to terminate. These cases are the following: evaluating when an achievement obligation is violated and when a maintenance obligation is fulfilled.

Standard Obligations

Obligations in standard deontic logic are evaluated in a single state. This semantics can be mirrored by forcing the activation periods to hold for exactly one state. I refer to this type of local obligations as *standard*, since their semantics has been mirrored from the semantics of standard deontic logic obligations.

A standard obligation is represented as follows: $\mathcal{O}^s\langle c, l, \top\rangle$. I use the tautology $\top$ to represent the deadline, hence the deadline is satisfied independently on the state, making this type of obligation always active for exactly one state.

Definition 19 (Comply with Standard)**.** *Given a standard obligation $\mathbf{O} = \mathcal{O}^s\langle c, l, \top\rangle$ and a trace θ, θ is compliant with $\mathcal{O}$, written $\theta \vdash \mathbf{O}$, if and only if: $\forall \sigma_i \in \theta$, if $\sigma_i \models l$, then $\sigma_i \models c$*

Standard obligations can be used to represent obligations in the real world that must be fulfilled as soon as they are triggered. The following example illustrates a case where such type of obligation can be used to describe an existing regulation.

Example 21 (Standard Obligation)**.** *Privacy regulations aim at protecting personal information by forbidding its collection without consent. Any collection of personal information without consent is considered illegal and whenever personal information is collected illegally, then such information must be destroyed immediately.*

The last part of the example: "whenever personal information is collected illegally, then such information must be destroyed immediately" can be represented as an obligation of type standard where the triggering condition consists of the illegal collection, the fulfilling

condition consists of the destruction of such information, which should be done immediately, in other words the deadline is set as now*.*

Non-Standard Obligations

The semantics of the two remaining types of local obligations is defined similarly as the respective global obligations introduced in Section 3.2. A similar semantics has already been introduced by Governatori et al. [34].

Example 22 (Achievement Obligation)**.** *A customer going to a bar to get a coffee has the obligation to pay for it. This obligation can be considered as a local obligation of type achievement with the following elements: the lifeline is constituted by ordering the coffee, the deadline by leaving the bar and the fulfilling condition by paying for the coffee.*

As it has been pointed out by Example 22, a local achievement obligation requires that at least a state included in each of its activation periods satisfies the fulfilment condition.

Definition 20 (Comply with Local Achievement)**.** *Given a local achievement obligation* $\mathbf{O} = \mathcal{O}^a\langle c, l, d\rangle$ *and a trace* θ*,* θ *is compliant with* $\mathbf{O}$*, written* $\theta \vdash \mathbf{O}$*, if and only if:* $\forall \sigma_i \in \theta$ *such that* $\sigma_i \models l$ *then* $\exists \sigma_j \in \theta$ *such that* $\sigma_j \models c$ *and* $\sigma_j \succeq \sigma_i$*, and* $\neg\exists \sigma_h \in \theta$ *such that* $\sigma_h \models d$ *and* $\sigma_i \preceq \sigma_h \prec \sigma_j$

Example 23 (Maintenance Obligation)**.** *While using public transportation, like busses or trains, it is obligatory to always have with yourself a valid ticket. This kind of obligation can be modelled as a local maintenance obligation with the following elements: the lifeline is constituted by getting on a public transport, the deadline by getting out and the fulfilment condition by having a valid ticket.*

Similarly to an achievement obligation, a maintenance obligation also requires to verify the condition between the lifeline and the deadline. However as we can see from Example 23, each state belonging to the activation periods, defined by the lifeline and the deadline, needs to satisfy the obligations' fulfilment condition.

Definition 21 (Comply with Local Maintenance)**.** *Given a local maintenance obligation* $\mathbf{O} = \mathcal{O}^m\langle c, l, d\rangle$ *and a trace* θ*,* θ *is compliant with* $\mathbf{O}$*, written* $\theta \vdash \mathbf{O}$*, if and only if:*

$\forall \sigma_i \in \theta$ *such that* $\sigma_i \models l$ *then* $\exists \sigma_h \in \theta$ *such that* $\sigma_h \models d$ *and* $\forall \sigma_j \in \theta$ *such that* $\sigma_i \preceq \sigma_j \preceq \sigma_h : \sigma_j \models c$

Activation Periods

The activation period of a local obligation identifies the subsequences of a trace where the obligation needs to be fulfilled. In other words, given an activation period for an obligation of type achievement, this obligation is satisfied by an activation period if there exists a state in such period that entails the fulfilment condition. Similarly, given an activation

period for an obligation of type maintenance, this obligation is satisfied if all the states belonging to the period entail the fulfilment condition.

An activation period is generally identified by a subtrace where the first state entails the lifeline and the last one entails the deadline. However, an activation period can be terminated prematurely. For obligations of type achievement, the activation period terminates when the obligation is fulfilled in that period, this means that an activation period for this type of obligation can terminate either when the deadline is reached or when the obligation is fulfilled by finding a state entailing the fulfilment condition. For obligations of type maintenance, their activation periods can prematurely terminate in case the obligation is violated, namely when a state belonging to the activation period does not entail the fulfilment condition.

The activation periods of a local obligation can be represented by using a finite state automaton. Fig. 4.2.(a) shows the automaton modelling the activation period of an achievement obligation. Fig. 4.2.(b) represents the automaton modelling the activation period of a maintenance obligation. It can be noticed that in both cases, an obligation becomes active only if is inactive and a state triggering the lifeline is found. Finding such state while the obligation is already active has no impact on the activation period of the obligation. Similarly, when an obligation is inactive, states triggering the deadline or the condition of an obligation do not influence its state.

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

The two automata are consistent with the semantics of Definitions 20 and 21. Notice that for the sake of clarity I avoid representing explicitly in the figure the transitions which would not have changed the state of the automaton, transitions starting and ending in the same state. For instance a state σ of a trace, where $\sigma \models \varphi_b$, reading σ does not change the state of the automaton if the automaton is in the *Active* state. Also notice that the ϵ in the automata mean that from the states *Fulfilled* and *Not Fulfilled*, the state of the automata becomes *Inactive* without reading and consuming a state of a trace.

Given a trace and an obligation, the automaton in Fig. 4.2 can be used to determine whether an obligation is active or inactive with respect to the states of the given trace. For this reason I avoid to represent any final state in the automaton.

Notice that, within a single activation period, local and global obligation behave the same. Obligations of the type achievement need to be fulfilled in at least one state belonging to the activation period, while obligations of type maintenance need to be fulfilled in all states belonging to the activation period.

When a trace does not activate a local obligation in any of its states, then such trace is considered to fulfil that obligation.

Lemma 5. *Given a θ and a local obligation $\mathbf{O} = \mathcal{O}^t\langle c, l, d\rangle$. If $\neg\exists\sigma \in \theta$ such that $\sigma \models l$,*

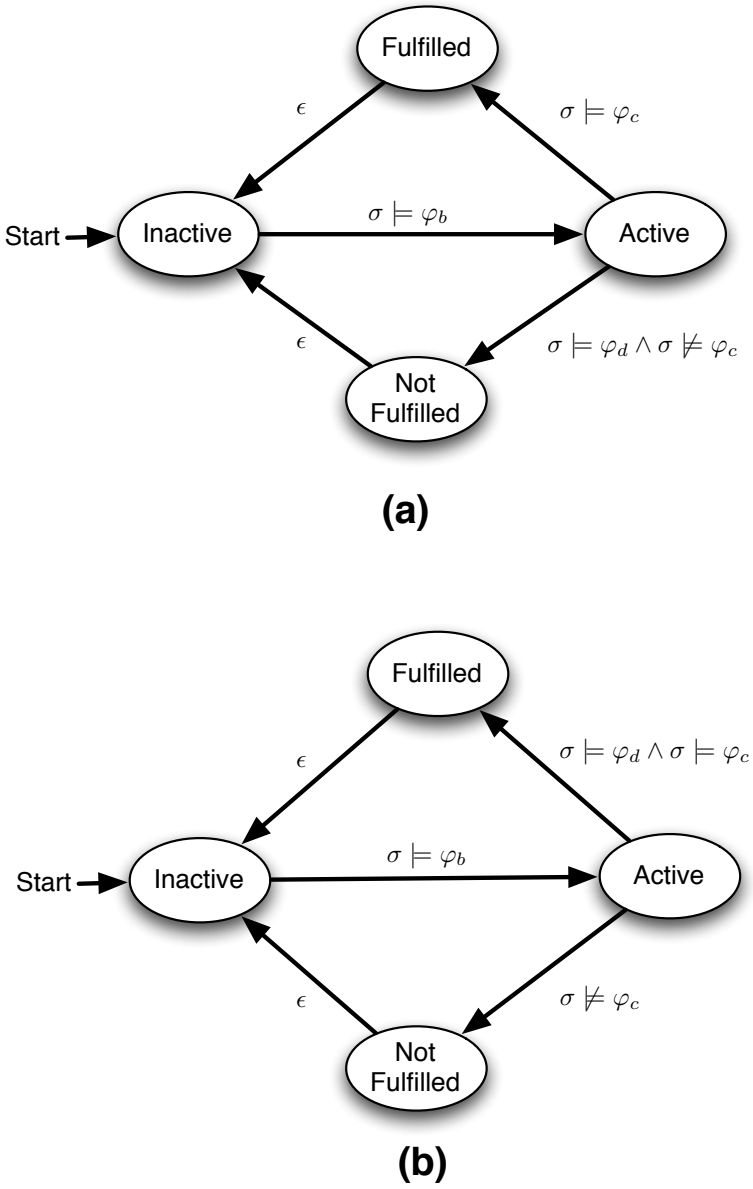

Figure 4.2: Activation Periods using Finite State Automaton

then $\theta \vdash \mathbf{\Theta}$.

Due to the possibility of having either global and local obligations belonging to a set of obligations, a way of abstracting from having to determine the activation periods of a

local obligation using the lifeline and the deadline is by using the function *Force* which determines, given a state of a trace, the set of obligations holding in that state.

Definition 22 (Obligation in force)**.** *Given a state σ composed of a task $t \in \mathcal{T}$ and a set of literals $L \in \mathcal{L}$. I define a function*

$$Force : \mathcal{T} \times 2^{\mathcal{L}} \mapsto 2^{\circledcirc}$$

where $\circledcirc$ is a set of obligations.

Given a trace, this function returns for each state belonging to the trace the set of obligations which include that state in one of their activation periods. Due to the possibility that a state does belong to none of the activation periods of the obligations.

Using the function *Force* I redefine the semantics of both global (Definitions 14 and 15) and local (Definitions 20 and 21) non-standard obligations. The new semantics is obtained by removing the elements defining the lifeline and the deadline from the structures representing the obligations, since the activation periods of an obligation can be now given by the function *Force*.

Definition 23 (Comply with Achievement)**.** *Given an achievement obligation $\mathcal{O}^a\langle c\rangle$ and a trace θ, θ is compliant with $\mathcal{O}^a\langle c\rangle$ if and only if:*

$\forall(\sigma_i, \ldots, \sigma_n) \in \theta$ such that $\forall\sigma_j \in (\sigma_i, \ldots, \sigma_n), \mathcal{O}^a\langle c\rangle \in Force(\sigma_j)$ and $(\sigma_i, \ldots, \sigma_n)$ is maximal , $\exists\sigma_h \in (\sigma_i, \ldots, \sigma_n)$ such that $\sigma_h \models c$.

Definition 24 (Comply with Maintenance)**.** *Given a maintenance obligation $\mathcal{O}^m\langle c\rangle$ and a trace θ, θ is compliant with $\mathcal{O}^m\langle c\rangle$ if and only if:*

$\forall\sigma_i \in \theta$ such that $\mathcal{O}^m\langle c\rangle \in Force(\sigma_i), \sigma_i \models c$.

Assuming that the activation periods of an obligation in a trace identified by the function *Force* are the same as the ones that would be identified by checking the lifelines and deadlines, then the newly defined semantics are equivalent to the original ones. The advantage of adopting the new semantics relies in the fact that the representation is less cluttered by avoiding to explicitly represent the lifelines and deadlines within the obligations. I define now another function, *Interval*, which by relying on *Force* is able to identify the activation periods of an obligation in a given trace.

Definition 25 (Interval)**.** *Given a trace θ, let θ_p be the complete set of the subintervals of θ. Given an obligation $\mathbf{O}$, the partial function* Interval *is defined as follows:*

$$Interval : 2^{\mathbf{O}} \times \theta \mapsto 2^{\theta_p} \text{ such that } \forall\varphi \in Interval(\mathbf{O}, \theta), \forall\sigma \in \varphi, \mathbf{O} \in Force(\sigma)$$

The function $Force$ returns all the intervals of a trace in which an obligation is active. $Force$ is defined as a partial function since it can be the case that an obligation is never activated in a trace, hence the set of intervals determining the activation period would be represented by the empty set.

Relations with Standard Obligations

I show now that standard obligations are a particular case of both achievement and maintenance obligations. If we constrain the activation period of an achievement obligation to a single state, then such state must satisfy the condition. The same applies to maintenance obligations, if the activation period is limited to only one state, then such state has to fulfil the condition. Therefore if the activation is limited to a single state, then the semantics of both achievement and maintenance obligations collapse in the semantics of standard obligations.

4.1.2 Fulfilling Multiple Obligations

The second difficulty vector being considered is the single-multiple vector. By moving forward on this difficulty vector, the computational complexity of verifying compliance with the regulatory framework increases because instead of having to complying with a single obligation, the process needs now to comply with a set of obligations. The sub-class of the problem obtainable by moving from $C0^{-}_{1ga}$ using this difficulty vector is $C1^{-}_{nga}$.

A trace is compliant with a set of local obligations if it fulfils all the local obligations belonging to the set. Note that according to Definitions 19, 20 and 21 a local obligation never activated by a trace is considered to be fulfilled by such trace.

Definition 26 (Set Fulfilment)**.** *Given a trace θ and a set of obligations $\circledcirc = \{\mathfrak{O}_1, \ldots, \mathfrak{O}_n\}$, $\theta \vdash \circledcirc$, iff:*

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

Otherwise $\theta \nvdash \circledcirc$.

4.1.3 Fulfilling Compensable Obligations

The third difficulty vector considered in this section is the atomic-compensable vector. By moving forward on this difficulty vector, obligations become compensable. This means that a compensation is associated to the primary obligation and allows to cope with violations of it if the compensation, composed of another obligation which I call secondary to distinguish it from the primary one, is fulfilled. The sub-class of the problem obtained in this case is the last belonging to the $C1^{-}$ class: $C1^{-}_{1gc}$

The case in which an obligation is not fulfilled is also referred as an occurring violation. For an achievement obligation, a violation can only be detected at the end of its in force period because before reaching that point, there is always chance for the content literal to hold once. Differently, a violation of a maintenance obligation is detected at the first moment when the content literal stops holding.

Introducing compensable obligations allows some flexibility while dealing with complex systems, where the possibility that regulations may not be followed has to be taken into account. Lomuscio and Sergot [51] studied this in the context of multi-agent systems.

Compensable obligations define what needs to be done when they are violated through secondary obligations as defined by Governatori and Rotolo [37]. Secondary obligations are a particular type of obligation whose lifeline is the violation of the obligation they try to compensate.

Definition 27 (Violations)**.** *Given a trace θ and an obligation $\mathbf{O}$, if θ is not compliant with $\mathbf{O}$, then a violation is raised for each activation of $\mathbf{O}$ which cannot be fulfilled by θ.*

Let V be a function returning the violations raised by a θ with respect to $\mathbf{O}$. Each element of $V(\theta, \mathbf{O})$ is identified in the earliest state of θ which makes an activation of $\mathbf{O}$ not fulfillable and the activation period terminates.

Notice that according to the semantics of the different types of obligations and how a violation is defined, namely in the first state where it becomes clear that an obligation cannot be fulfilled, a violation of a local achievement obligation is always identified in correspondence of a deadline and a violation of both global and local maintenance obligation is identified by the earliest state within the activation period which does not fulfil the condition. Also notice that for standard obligations, a violation is identified the state corresponding to the deadline, which is also the first state non fulfilling the condition. This condition is also in line with the fact that standard obligations are special cases of both achievement and maintenance, hence identifying a violation in an obligation of type standard can be done by using either one of the methods that can be used for achievement or maintenance obligations. Violations for achievement and maintenance obligations are illustrated in Example 24.

Example 24 (Violations)**.** *Considering an achievement obligation stating the following "The bill must be payed before leaving the table", aiming at governing customers' behaviour in a restaurant. A violation of such obligation is identified in a state where the customer leaves the table without having paid the bill, which corresponds to the state terminating the activation period of the obligation according to its deadline.*

Considering a maintenance obligation stating the following: "Personal information must not be collected without consent", tailored for privacy protection. A violation of such obligation is detected in the first occurring state within the activation period of the obligation where personal information is collected without consent, corresponding to the first occurring state not fulfilling the condition of the obligation.

Definition 28 (Compensable Obligation)**.** *A compensable obligation, written $\lightning = \mathbf{O} \otimes \mathbf{O}_c$, is composed of a primary obligation $\mathbf{O}$ and a compensation $\mathbf{O}_c$.*

The relations between the activation periods of $\mathbf{O}$ and $\mathbf{O}_c$ are the following: $\forall I \in Interval(\mathbf{O}_c, \theta), \exists v \in V(\mathbf{O}, \theta) : min(I) = v$. Moreover $\forall v \in V(\mathbf{O}, \theta), \exists I \in Interval(\mathbf{O}, \theta) : max(I) = v$.

The compensation $\mathbf{O}_c$ can be as well a compensable obligation.

Compensable obligations are written as sequences of obligations connected using the operator $\otimes$. The semantics of the operator $\otimes$ states that the obligation on its right must

be fulfilled if the obligations on the left is violated. More precisely for each violation of the obligation on the left of $\otimes$, an activation period of the obligation on the right of the operator is triggered.

Definition 29 (Comply with Compensable Obligations)**.** *Given a trace θ and a compensable obligation $\lightning = \mathbf{O} \otimes \mathbf{O}_c$. θ is compliant with $\lightning$ if and only if θ is compliant with $\mathbf{O}_c$.*

A trace is compliant with a compensable obligation if it is compliant with its secondary obligation. This follows from Definitions 14, 15, 19, 20 and 21, where a trace θ is always considered to be compliant with an obligation $\mathbf{O}$ if the set of activation periods of $\mathbf{O}$ is empty. This means in this case that either the primary obligation is not violated or if it is violated, then each violation has been compensated.

Similarly for atomic obligations, also for compensable obligations a single instance of the obligation can be active at a given time. Moreover when a compensable obligation $\lightning = \mathbf{O}_1 \otimes \ldots \otimes \mathbf{O}_n$ is active, exactly one of the obligations belonging to the chain is considered to be active at a given time. This means that as soon as a compensable obligation is activated, the first obligation of the chain becomes active. When the active obligation belonging to the chain is violated, it is then deactivated and the following one is activated. The whole compensable obligation is then deactivated in two cases: first if one of the obligations composing its chain is fulfilled, meaning that the whole compensable obligation is fulfilled in this activation; second if the last obligation in the chain is violated, meaning that the compensable obligation is violated in this activation. This semantics can be represented using an automaton as shown in Figure 4.3. The automaton analyses in order the states σ of a trace θ. We can see that an arbitrary compensable obligation $\lightning = \mathbf{O}_1 \otimes \mathbf{O}_2 \otimes \ldots \otimes \mathbf{O}_n$ is activated when the lifeline of the first obligation, written $l_b^{\mathbf{O}_1}$, of the chain is satisfied by the state being analysed. The compensable obligation is deactivated if one of the obligations belonging to the chain are fulfilled or when all the obligations belonging to the chain are not fulfilled. The different obligations belonging to the chain are fulfilled or not fulfilled according to the semantics illustrated by the automata in Figure 4.2.

The deliberately simple syntax and semantics chosen to represent and reason about the obligations does not allow to activate more than a single instance for each obligation. However this approach allows to study the problem while avoiding being overwhelmed by the excessive complexities of more expressive syntaxes and semantics. Also in the case of the limitation concerning having a single instance of an obligation active at a single time, this particular limitation can be overcome by using an approach similar to first order logic, as it has been used by Montali et al. [57] to reason about commitments in an instance based way. Using this approach, each trigger of an obligation activates an instance of it which is handled separately from other instances of the same obligation.

Example 25. *An example of compensable obligations is the following: When you dine at a restaurant you have to pay for your meal. If you don't, then you have to wash the dishes.*

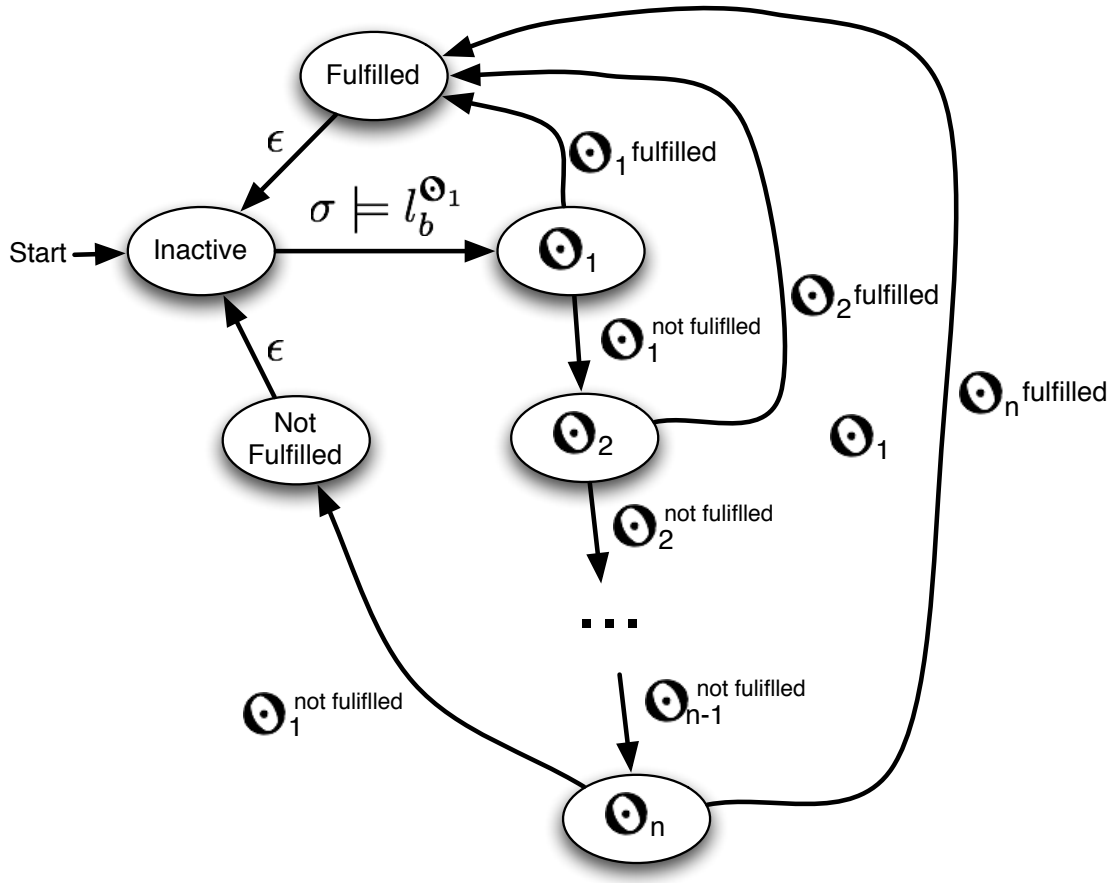

Figure 4.3: Activation Periods of Compensable Obligations

This compensable obligation shown in Example 25 can be formalised using a global compensable obligation as follows: $\mathcal{O}^a\langle\alpha\rangle \otimes \mathcal{O}^a\langle\beta\rangle$, where α represents "paying the bill" and β represents "washing the dishes". Following from the semantics it occurs that having to wash the dishes becomes obligatory only in the case a customer fails to pay its bill, which is what the proposed formalisation aims at representing.

4.2 Conflicting Obligations

In the previous part of this chapter I have introduced the semantics of more complex regulatory frameworks. In this second part of the chapter I am analysing the problem of conflicting obligations that can arise when dealing with multiple obligations. This problem

arises when a set of obligations contains an obligation stating that something is obligatory and another obligation stating that the opposite is obligatory. It is clear in this case that a business process model could never be compliant with such set since it is required to do something and its opposite which is not allowed. We will see later that in order for a conflict to arise, these conflicting obligations need to be activated together.

The same problem addressed in the setting of desirable behaviours and moral dilemmas has been already studied by Lemmon [50], where he shows that if a moral dilemma arise, then an individual cannot identify which behaviour is the most desirable, leading to the impossibility to decide which is best to adopt. Therefore I claim that a clear understanding about how obligations interact is imperative to avoid situations where it is necessary to fulfil obligation that contradict each other.

Example 26. *The "working week" defines that workers have to work from Monday to Friday. However, Islam defines that Friday is an holy day and it is forbidden to work.*

The example describes a conflicting situation resulting from merging different regulations, religious and business. The issue of conflicting regulations has been already studied in *normative reasoning*, like by Elhag et al. [29], Beirlaen and Straßer [13], and Sartor [66]. In particular, since regulations define what is obligatory, prohibited and permitted, *deontic logic* [43] and its variants have been extensively used to reason about them. For instance Hansen [40] studies the conflicts between obligations using *dyadic deontic logic*.

The deficiency of standard deontic logic to deal with conflicts has been already studied by Beirlaen et al. [14]. Whereas Beirlaen et al. focus on identifying conflicts between both permissions and obligations in single time instants, in this section I study the conflicts between obligations in scenarios involving traces. I first show that the consistency measure used in standard deontic logic appears to be too restrictive while reasoning about normative conflicts in this setting. Therefore I propose an alternative definition of consistency more suited to detect conflicts in settings involving traces.

Second I show how the new definition of conflicting obligations can be used to identify the sufficient and necessary conditions that the obligations belonging to a set must follow in order to avoid conflicting obligations. This analysis allows to avoid conflicting obligations in the sub-classes containing multiple obligations. The simplest of these sub-classes of the problem are $C1_{nga}$ and the corresponding $C1^{-}_{nga}$, which is restricted to literals in its possibility of expressing the elements of the obligations used.

4.2.1 Consistency

In standard deontic logic the axioms: $\mathcal{P}\top$ and $\mathcal{O}\alpha \rightarrow \neg\mathcal{O}\neg\alpha$, and the equivalence $\mathcal{O}\alpha \equiv \neg\mathcal{P}\neg\alpha$ hold. The equivalence expresses a relation between obligations and permissions, in other words it states that if something is obligatory, then the opposite is not permitted. The first axiom: $\mathcal{P}\top$, states that tautologies are always permitted and the second axiom:

$\mathcal{O}\alpha \rightarrow \neg\mathcal{O}\neg\alpha$, states that if something is obligatory then its complement must not be obligatory.

The two consistency measures: internal consistency and external consistency can be defined using the two axioms and the equivalence. Internal consistency expresses the fact that something contradictory, like a proposition and its negation, cannot be obligatory.

Definition 30 (Internal Consistency)**.** *A set of norms is internally consistent iff there is no formula such that* $\mathcal{O}(\alpha \wedge \neg\alpha)$ *is entailed by the set of norms.*

Accordingly internal consistency corresponds to axiom $\mathcal{P}\top$:

$$\neg\mathcal{O}(\alpha \wedge \neg\alpha) \equiv \neg\mathcal{O}\bot \equiv \mathcal{P}\top$$

External consistency expresses that two contradictory obligations cannot coexist, like for instance the obligation of performing an action along with the prohibition of performing it.

Definition 31 (External Consistency)**.** *A set of norms is externally consistent iff there are no formulae such that* $\mathcal{O}\alpha \wedge \mathcal{O}\neg\alpha$ *is entailed by the set of norms.*

Accordingly internal consistency corresponds to axiom $\mathcal{O}\alpha \rightarrow \neg\mathcal{O}\neg\alpha$:

$$\neg(\mathcal{O}\alpha \wedge \mathcal{O}\neg\alpha) \equiv \mathcal{O}\alpha \rightarrow \neg\mathcal{O}\neg\alpha \equiv \mathcal{O}\alpha \rightarrow \mathcal{P}\alpha$$

In standard deontic logic the two axioms $\mathcal{P}\top$ and $\mathcal{O}\alpha \rightarrow \neg\mathcal{O}\neg\alpha$ are equivalent. The two consistency measures defined in the present section are used in standard deontic logic to identify inconsistencies. Although inconsistencies involving permissions are also possible, I focus in this thesis on inconsistencies among obligations, which are the relevant ones involved in the problem of proving regulatory compliance as it has been described in this thesis.

Notice that even if the consistency measures are expressed over formulae, they can apply to the obligations defined in Section 3.2 and 4.1 by considering as the fulfilment conditions of these obligations as formulae composed of a single literal. This means that I analyse conflicting obligations in the set of problems belonging to the C class shown in Figure 1.2. Being the class of problem C^- a particular case of the class C, it means that the analysis of conflicting obligations for the class C is also applicable to the problems belonging to the class C^-.

4.2.2 Consistency of Standard Obligations

It is natural to expect that both internal and external consistency measures (Definitions 30 and 31) still apply to standard obligations, since their semantics mirrors the semantics of the obligations expressed using standard deontic logic. I introduce now two propositions reflecting that the consistency measures apply to standard obligations and prove them.

Proposition 1. *$\neg\exists\theta|\theta$ complies with $\mathcal{O}^s\langle\alpha\wedge\neg\alpha\rangle$.*

Sketch. If we assume an obligation $\mathcal{O}(\alpha\wedge\neg\alpha)$ to be possible, then the translated standard obligation would be the following: $\mathcal{O}^s\langle\alpha\wedge\neg\alpha\rangle$.

From Definition 19 it follows that a trace must contain a state σ_i such that $\sigma_i\models\alpha\wedge\neg\alpha$ in order to comply with the standard obligation. However such state could not exist according to Definition 9 since each state must be consistent. Thus a standard obligation whose condition is a contradiction would never be complied by any trace.

Therefore internal consistency applies to standard obligations.

□

Proposition 2. *$\neg\exists\theta|\theta$ complies with $\mathcal{O}^s\langle\alpha\rangle$ and θ complies with $\mathcal{O}^s\langle\neg\alpha\rangle$.*

Sketch. Assume a trace containing the state σ_i, where $\{\mathcal{O}^s\langle\alpha\rangle,\mathcal{O}^s\langle\neg\alpha\rangle\}\in \mathit{Force}(\sigma_i)$.

From Definition 31, $\mathcal{O}\alpha\wedge\mathcal{O}\neg\alpha$ is translated in standard obligations as follows: $\mathcal{O}^s\langle\alpha\rangle$ and $\mathcal{O}^s\langle\neg\alpha\rangle$. According to Definition 19, since both standard obligations are in force in σ_i, then both conditions have to be verified in the same state.

A state σ_i, in order to fulfil both obligations, needs to contain in its state both α and $\neg\alpha$, however this is in contradiction with Definition 9, stating that a state has to be consistent. Thus it follows that a state σ_i satisfying both α and $\neg\alpha$ cannot exist.

Therefore a trace compliant with both standard obligations $\mathcal{O}^s\langle\alpha\rangle$ and $\mathcal{O}^s\langle\neg\alpha\rangle$ cannot exist. Thus no solution can exist when such pair of obligations is considered.

□

4.2.3 SDL Consistency is too restrictive

I show here that the external consistency measure expressed in standard deontic logic is too restrictive when used in a dynamic setting involving traces. As one can expect, the unsuitability follows from the fact that SDL's consistency measure do not take into account the temporal relations between the states composing a trace. This is shown in the following example extending Example 13.

Example 27. *The authors of a paper must submit it to the conference Δeon before the deadline, which is set on Sunday. This also means that the paper has to be finished before the submission deadline. However, as usual on the weekends, the authors must go to the pub to meet their friends on Saturday or Sunday.*

Example 27 contains two obligations: submitting the paper and going to the pub. Lets assume that the authors cannot finish and submit the paper while at the pub, hence we consider these obligations to be complementary. Thus if the proposition α represents "finishing and submitting the paper", then we can use $\neg\alpha$ to represent "going to the pub".

To formalise the example I discretise time in days, which in turn represent the states. I use the propositions *sat* and *sun* to represent Saturday and Sunday respectively. Lastly I

use the proposition *aut* to represent being an author of a paper and formalise the obligation of going to the pub: $\mathcal{O}^a\langle\neg\alpha\rangle$ and the obligation of submitting the paper as: $\mathcal{O}^a\langle\alpha\rangle$.

Both obligations are of type achievement. Despite the conditions of the obligations being complementary, it is still possible to provide a trace complying both obligations.

$$\theta = (\ldots, (t_i, \{aut\}), \ldots, (t_j, \{sat, \neg\alpha\}), (t_k, \{sun, \alpha\}))$$

Assuming that $\mathcal{O}^a\langle\neg\alpha\rangle$ is in force in both $(t_j, \{sat, \neg\alpha\})$ and $(t_k, \{sun, \alpha\})$ and $\mathcal{O}^a\langle\alpha\rangle$ is in force from $(t_i, \{aut\})$ till the end.

Example 27 describes a situation where two complementary obligations coexist during the weekend, but can be both fulfilled. According to the consistency measures provided by standard deontic logic, this situation would result in a conflict since it violates the external consistency measure. From the present analysis it follows that standard deontic logic is ill suited to reason about traces, more precisely the external consistency measure is too restrictive, which is not surprising since it is not a dynamic logic.

Redefining Conflicts

I now propose a new definition of inconsistent obligations, better suited to be used in dynamic settings involving traces, I refer to this as dynamic conflict.

Definition 32 (Dynamic Conflict)**.** *A set of obligations, written $\circledcirc$, is conflicting if and only if it is not possible to construct a trace in such a way that it is compliant with each obligation belonging to the set.* $\neg\exists\theta|\theta$ *compliant with* $\mathbf{O}, \forall\mathbf{O} \in \circledcirc$

The necessary conditions for two obligations to be conflicting is that their fulfilment conditions are complementary and their activation periods need to overlap. Depending on the type of obligations considered, the necessary condition may not be sufficient to determine whether they are conflicting.

Pair-wise Conflicts

In Definition 32, conflicts are defined between sets of obligations. The following example illustrates a case where a conflict arises from a set of obligations and, when any proper subset of the obligations is considered a conflict does not arise.

Example 28 (Conflicting Set)**.** *Assume a trace θ and a set of obligations composed of a single achievement obligation $\mathcal{O}^a\langle\alpha\rangle$ and k standard obligations $\mathcal{O}^s\langle\neg\alpha\rangle$ such that $Interval(\mathcal{O}^a\langle\alpha\rangle, \theta) \equiv \bigcup I \in Interval(\mathcal{O}^s\langle\neg\alpha\rangle, \theta)$ and $\bigcap I \in Interval(\mathcal{O}^s\langle\neg\alpha\rangle, \theta) = \emptyset$. In other words the activation periods of the standard obligations are all distinct and entirely cover the activation period of the achievement obligation.*

From Example 28, we can see that a trace compliant with all the obligations belonging to the set proposed cannot exist because it would require a state containing both α and $\neg\alpha$.

The behaviour of the standard obligations in Example 28 can be simulated using a single maintenance obligation. The behaviour required from a trace to be compliant with the set of standard obligations is that in such trace $\neg\alpha$ holds for the interval determined by the obligations. The same result can be obtained by using a single maintenance obligation requiring $\neg\alpha$ to hold for the same interval. Thus the set of standard obligations can be substituted with a single maintenance obligation satisfying the following condition on the activation period:

$$\bigcup \varphi \in \mathit{Interval}(\mathcal{O}^s\langle\neg\alpha\rangle, \theta) \equiv \mathit{Interval}(\mathcal{O}^m\langle\neg\alpha\rangle, \theta)$$

Therefore I focus from now on into analysing pair-wise conflicts between obligations.

4.2.4 Conflict Detection

The two necessary conditions to detect conflicting obligations are the following:

1. Their fulfilment conditions have to be complementary: $\mathbf{O}_1\langle\alpha\rangle$ and $\mathbf{O}_2\langle\beta\rangle$, such that $\alpha \wedge \beta \rightarrow \bot$.
2. The intersection of their activation periods must be not empty: $\exists x, y | x \in \mathit{Interval}(\mathbf{O}_1\langle\alpha\rangle, \theta), y \in \mathit{Interval}(\mathbf{O}_2\langle\beta\langle, \theta)$ and $x \cap y \neq \emptyset$.

I identify here the sufficient conditions to decide whether two obligations are conflicting. Being standard obligations a special case of both achievement and maintenance, it is sufficient to analyse the three combinations involving these types ($\mathcal{O}^m - \mathcal{O}^m$, $\mathcal{O}^m - \mathcal{O}^a$ and $\mathcal{O}^a - \mathcal{O}^a$). To do so I introduce two auxiliary functions, which applied to an interval or a trace, returns the first state belonging to them: *min*, or the last state: *max*.

Maintenance - Maintenance

I define now the sufficient condition to detect whether two maintenance obligations are conflicting.

Definition 33 ($\mathcal{O}^m - \mathcal{O}^m$ Conflict)**.** *Let $\mathcal{O}^m\langle\alpha\rangle$ and $\mathcal{O}^m\langle\beta\rangle$ be two complementary maintenance obligations. $\mathcal{O}^m\langle\alpha\rangle$ and $\mathcal{O}^m\langle\beta\rangle$ are conflicting if and only if:*

$$\exists I \in \mathit{Interval}(\mathcal{O}^m\langle\alpha\rangle, \theta) \text{ and } \exists I' \in \mathit{Interval}(\mathcal{O}^m\langle\beta\rangle, \theta) : I \cap I' \neq \emptyset$$

Proposition 3 ($\mathcal{O}^m - \mathcal{O}^m$ Conflict)**.** *Let $\mathcal{O}^m = \langle\alpha\rangle$ and $\mathcal{O}'^m = \langle\beta\rangle$ be conflicting maintenance obligations, then does not exist a trace complying with both obligations.*

Two maintenance obligations are conflicting as soon as they are complementary and their activation periods overlap. In this case the sufficient condition is also the necessary condition previously introduced.

4.2.5 Maintenance - Achievement

I define now the sufficient condition to detect whether a maintenance and an achievement obligation are conflicting.

Definition 34 ($\mathcal{O}^m - \mathcal{O}^a$ Conflict)**.** *Let $\mathcal{O}^m\langle\alpha\rangle$ be a maintenance obligation and $\mathcal{O}^a\langle\beta\rangle$ be a complementary achievement obligation. $\mathcal{O}^m\langle\alpha\rangle$ and $\mathcal{O}^a\langle\beta\rangle$ are conflicting if and only if:*

$$\exists I \in Interval(\mathcal{O}^a\langle\beta\rangle, \theta) \text{ and } \exists I' \in Interval(\mathcal{O}^m\langle\alpha\rangle, \theta) : I \subseteq I'$$

The sufficient condition captures the fact that an achievement obligation requires be fulfilled in a single state, hence a conflict arise only if the activation period of the maintenance obligation is a superset of the activation period of the achievement obligation.

4.2.6 Achievement - Achievement

I define now the sufficient condition to detect whether two achievement obligations are conflicting.

Definition 35 ($\mathcal{O}^a - \mathcal{O}^a$ Conflict)**.** *Let $\mathcal{O}^a\langle\alpha\rangle$ and $\mathcal{O}^a\langle\beta\rangle$ be two conflicting achievement obligations. $\mathcal{O}^a\langle\alpha\rangle$ and $\mathcal{O}^a\langle\beta\rangle$ are conflicting if and only if:*

$$\exists I \in Interval(\mathcal{O}^a\langle\alpha\rangle, \theta) : I \in Interval(\mathcal{O}^a\langle\beta\rangle, \theta) \text{ and } ||I|| = 1$$

The sufficient condition requires that there exists an activation period common to the two complementary achievement obligations and that such activation period is of length one. These restrictive conditions are necessary due to the flexibility allowed to comply with achievement obligations. Two achievement obligations are actually conflicting if and only if both behave as standard obligations in at least a shared activation period.

In the present section I have not reported the abstract semantics of standard obligations, however to identify conflicting standard obligations (Definition 19), the following sufficient condition is enough:

$$\exists I \in Interval(\mathcal{O}^s\langle\alpha\rangle, \theta) : I \in Interval(\mathcal{O}^s\langle\beta\rangle, \theta)$$

As it is expected, this sufficient condition is a particular case of all the other conditions identified in the present section.

4.2.7 Conflicts involving Compensable Obligations

One of the difficulty vectors introduces compensable obligations. In case the complexity of the regulatory compliance checking problem is increased through multiple difficulty vectors, meaning that the problem may involve checking the compliance of a process against a set of compensable obligations, it also become important to identify when compensable obligations are conflicting.

4.2.8 Conflicts for Compensable Obligations

I define now the sufficient conditions to identify pair-wise conflicts involving compensable obligations. A compensable obligation is not a new type of obligation, but rather a way of structuring the existing types of obligations. We can consider an atomic obligation to be a special case of compensable which compensation obligation cannot be fulfilled if triggered (i.e: $\lightning = \mathbf{O} \otimes \bot$). Therefore I analyse the more general case of deciding which are the sufficient conditions to determine whether two compensable obligations are conflicting.

Definition 36 ($\mathbf{O}$ - $\mathbf{O}$ Conflict)**.** *Let $\lightning = \mathbf{O} \otimes \mathbf{O}_c$ and $\lightning' = \mathbf{O}' \otimes \mathbf{O}'_c$ be two compensable obligations. $\lightning$ and $\lightning'$ are conflicting if and only if:*

$$\mathbf{O}_c \textit{ is conflicting with } \mathbf{O}'_c$$

To determine whether two obligation "conflict" I reuse the sufficient conditions from Definitions 33, 34 and 35. The sufficient condition expressed in Definition 36 requires that the compensations of the two compensable obligations are conflicting. A compensation $\mathbf{O}_c$ is triggered by a violation of the primary obligation $\mathbf{O}$, hence $||Interval(\mathbf{O}_c, \theta)|| = ||V(\mathbf{O}, \theta)||$. If the two secondary obligation are conflicting, it means that both $V(\mathbf{O}, \theta)$ and $V(\mathbf{O}', \theta)$ are not empty due to existing conflicts between $\mathbf{O}$ and an obligation in $\lightning'$ and vice versa.

4.3 Summary

In this chapter I first introduce a three additional regulatory frameworks obtainable by increasing the complexity of the obligations involved by moving on exactly one of the difficulty vectors defining one of the families of the sub-classes of the problem. The frameworks introduced in this chapter are a step more difficult than the basic problem. The semantics of the frameworks obtained can be combined to identify more complex sub-classes of the problem of proving regulatory compliance.

The second part of this chapter focuses on analysing conflicting obligations. While trying to prove the compliance of a business process, it is very important that the regulatory framework does not contain conflicting obligations, otherwise a model has no possibility of complying with it. Therefore I first provided an alternative semantics more suited to reason about obligations in such a setting and second I define the sufficient and necessary conditions for identifying conflicting obligations. The conditions provided can be also used to detect conflicts in existing systems and then resolving them using methodologies like the the one developed by Prakken and Sartor [60] or by Vasconcelos et al. [80].

It is worth mentioning that another important element in normative reasoning is constituted by permissions, which, as described by Boella and van der Torre [16], and Makinson and van der Torre [55], can be used as a mean to limit the applicability of obligations and prohibitions, as already has been studied by Stolpe [70] where the semantics is defined using AGM belief revision [5] and Input/Output logic [56]. Conflict detection

involving permission has already been studied by Hansen [41], however I did not discussed them in the present setting since it falls outside the scope of this thesis.

Chapter 5

Some Complexity Results

This chapter answers part of the first research subquestion that the present thesis asks: *What is the computational complexity of the sub-classes of the problem of proving regulatory compliance*? In particular in this chapter I answer the following questions: what is the computational complexity of the sub-classes $C2^{-}_{nla}$ and $C2_{nla}$ of the problem of proving regulatory compliance.

I answer to the first part of the research question in the first part of the present chapter, concerning the complexity of the sub-class $C2^{-}_{nla}$, consisting of verifying the compliance of a business process against a set of local obligations. The second part of the chapter provides an answer about the computational complexity of the problem $C2_{nla}$, a sub-class of the problem similar to the previous one, but where the elements of the obligations can be expressed using propositional formulae and are not limited to propositional literals. Part of the results discussed in the present chapter have already been published and have been coauthored with Guido Governatori and Pierre Kelsen [74].

5.1 Computational Complexity of $C2^{-}_{nla}$

In this first section of the chapter I analyse one of the more complex sub-classes of the problem obtainable by combining two of the three features discussed in Section 4.1. More precisely I analyse here the sub-class $C2^{-}_{nla}$, highlighted in Figure 5.1. I opted to analyse directly the sub-class $C2^{-}_{nla}$ and not the easier sub-classes $C1^{-}_{1la}$ and $C1^{-}_{nga}$, obtainable by adding a difficult feature to the basic problem, because such sub-class of the problem is expressive enough to allow the reductions used in the present chapter to prove its computational complexity.

5.1.1 Regulatory Framework

The regulatory framework of a problem $C2^{-}_{nla}$ is composed of multiple local obligations. The semantics of the different types of local obligations used in this section is the same as

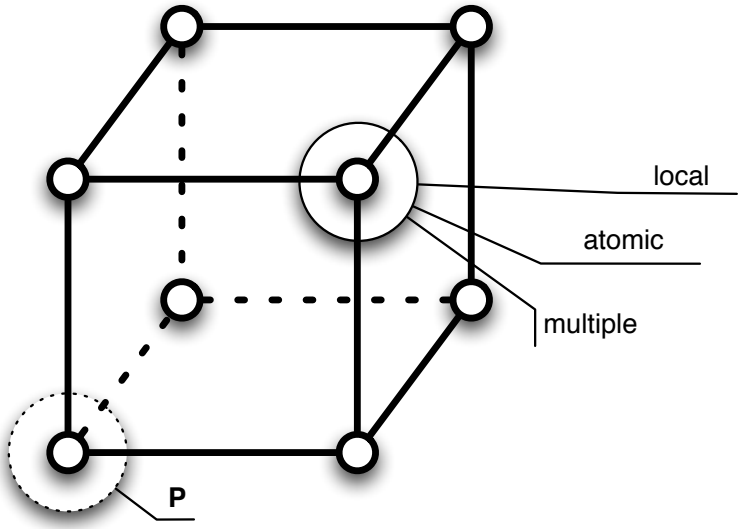

Figure 5.1: Multiple Local Obligations Compliance Problem

defined in Definitions 19, 20 and 21, where are respectively defined standard, achievement and maintenance obligations. These semantics are used along with Definition 26, describing how a trace fulfils a set of obligations, to obtain the following definition which describes when and how a process is compliant with a regulatory framework composed of a set of local obligations.

Definition 37 (Process Set Compliance)**.** *Given a process* (P, ann) *and a set of obligations* $\circledcirc$*.*

- ***Full Compliance****:* $(P, \mathsf{ann}) \vdash^{F} \circledcirc$
  iff $\forall \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc$*.*
- ***Partial Compliance****:* $(P, \mathsf{ann}) \vdash^{P} \circledcirc$
  iff $\exists \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc$*.*
- ***Non Compliance****:* $(P, \mathsf{ann}) \nvdash \circledcirc$
  iff $\neg\exists \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc$*.*

The remainder of this first part of the chapter, differently than Section 3.2, does not introduce new algorithms to solve the sub-class $C02^{-}_{nla}$, instead it provides an analysis of the complexity of the sub-class. This section is divided into three subsections: the first analyses the complexity of proving that a process is partially compliant, the second analyses the complexity of proving that a process is not compliant and the last one analyses the complexity of proving that a process is fully compliant.

5.1.2 Proving Partial Compliance is NP-complete

I first show that proving partial compliance of a problem $C2^{-}_{nla}$ is an **NP**-complete problem.

Definition 38 (NP-complete)**.** *A decision problem is **NP**-complete if it is in the set of **NP** problems and if every problem in **NP** is polynomial-time many-one reducible to it.*

To prove the **NP**-completeness of $C2_{nla}$ I first show that the problem is in **NP** and second that another **NP**-hard problem, in this case the problem to find whether a graph possesses an hamiltonian path, is polynomial-time many-one reducible to it.

NP Membership

To prove membership in **NP**, I need to 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 process and the set of obligations) with which we can check whether it fulfils the regulatory framework in polynomial time. As a certificate I choose a particular trace satisfying the obligations composing the regulatory framework.

The size of any proper traces is always polynomial with respect to the process considered and the set of literals, because the type of processes considered in this thesis are structured, meaning that cycles are not allowed and a task belonging to this type of processes can be executed at most once. Thus given a structured process, the maximum length of a proper trace is finite and is not greater than the number of the tasks contained in such process. Additionally the size of the states contained in a trace is at most as big as the set of literals used in the process. Before verifying the compliance of such a trace, we first need to check that the trace is indeed a valid trace for the process (done by Algorithm 3) and that the trace does indeed satisfy the obligations (done by Algorithm 4). I will further show that the time complexity of both algorithms is at most polynomial in the size of the input, thus concluding that verifying partial compliance is in **NP**.

Claim 1. Proving partial compliance for the sub-class $C2_{nla}$ is in **NP**.

Verifying the Validity

I hereby describe Algorithm 3 which checks whether a certificate is a valid trace of a process. In other words, given a process model and a trace, the algorithm verifies whether the sequences of states and tasks composing the trace is a possible trace of the given model.

Algorithm 3. *Given a trace $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ where $\sigma_{start} = (\mathsf{start}, L_0)$ and $\sigma_{end} = (\mathsf{end}, L_{n+1})$, the corresponding execution $\epsilon = (t_1, \ldots, t_n)$, and process (P, ann) where B is the main process block of P, the following algorithm $A_3(\theta, \epsilon, (P, \mathsf{ann}), B)$ decides if θ is a valid trace of (P, ann).*

Algorithm A_3

1: ***if*** $P_1(\epsilon, B)$ *and* $P_2(\theta, (P, \mathsf{ann}))$ ***then***
2: ***return*** $\theta \in \Theta(P, \mathsf{ann})$

3: ***else***
4: ***return*** $\theta \notin \Theta(P, \mathsf{ann})$
5: ***end if***

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

Procedure 4. $P_1(\epsilon, B)$

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

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

Procedure 5. $P_2(\theta, (P, \mathsf{ann}))$

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

P_2 *returns true if all of these properties hold and* $false$ *otherwise.*

Complexity:
To analyse the complexity of checking whether a trace is a valid serialisation of a process whose main process block is B (procedure $\mathbf{P_1}$), consider the tree reflecting the hierarchical structure of a process block. If B is a single task, the tree consists of a single node representing a task. Otherwise the tree has a root corresponding to B and subtrees representing the different subblocks B_i of B. The recursive procedure spends polynomial time (as a function of n, where n is the number of tasks in B) for each node of the tree for pre-processing, launching the recursive calls and recombining the results. Since the size of the tree itself is $\mathbf{O}(n)$ the overall time for the procedure is polynomial in n.
Procedure $\mathbf{P_2}$ can clearly be executed in time polynomial in $n \times k$ where k is the size of the set of literals. Therefore the time complexity of Algorithm 3 is $\mathbf{O}(n \times k)$.

Verifying the Fulfilment

I describe now Algorithm 4, which verifies whether a certificate fulfils the obligations contained in a regulatory framework. Given a trace and a set of obligations, this algorithm verifies whether the trace is compliant with a regulatory framework composed of the given set of obligations. Since the set of obligations contains local obligations, the algorithm verifies whether the trace is compliant with the regulatory framework in the setting of a $C2^{-}_{nla}$ problem.

Algorithm 4. *Given a set of obligations $\circledcirc$ and a trace $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ such that $\sigma_{start} = (\mathsf{start}, L_0)$ and $\theta \in \Theta(P, \mathsf{ann})$, the algorithm $A_4(\theta, \circledcirc)$ is defined as follows (In the following, Ob denotes the set of active obligations and we treat θ as a vector):*

Algorithm A_4

1: $\mathsf{Ob} = \emptyset$
2: ***for each*** σ *in* θ ***do***
3: ***for each*** $\mathfrak{O} = \mathcal{O}^t\langle l_c, l_b, l_d\rangle$ *in* $\circledcirc$ ***do***
4: ***if*** $\sigma \models l_b$ ***then***
5: $\mathsf{Ob} = \mathsf{Ob} \cup \mathfrak{O}$
6: ***end if***
7: ***end for each***
8: ***for each*** $\mathfrak{O} = \mathcal{O}^t\langle l_c, l_b, l_d\rangle$ *in* Ob ***do***
9: ***if*** $t = a$ ***then***
10: ***if*** $\sigma \models l_c$ ***then***
11: $\mathsf{Ob} = \mathsf{Ob} \setminus \mathfrak{O}$
12: ***else***
13: ***if*** $\sigma \models l_d$ ***then***
14: ***return*** $\theta \nvdash \circledcirc$
15: ***end if***
16: ***end if***
17: ***else***
18: ***if*** $t = m$ ***then***
19: ***if*** $\sigma \not\models l_c$ ***then***
20: ***return*** $\theta \nvdash \circledcirc$
21: ***end if***
22: ***if*** $\sigma \models l_d$ ***then***
23: $\mathsf{Ob} = \mathsf{Ob} \setminus \mathfrak{O}$
24: ***end if***
25: ***end if***
26: ***end if***
27: ***end for each***
28: ***end for each***
29: ***return*** $\theta \vdash \circledcirc$;

Algorithm 4 identifies wether a certificate fulfils a set of obligations. If the certificate is a valid trace of a process, then following from Definition 37, the fact that the certificate fulfils the set of obligations is a sufficient condition to say that the process is partially compliant

with the regulatory framework containing such set of obligations.

Complexity:

The complexity of checking whether a trace is compliant with the set of obligations using Algorithm 4 is at most polynomial in time $\mathbf{O}(n \times o)$ where n is the number of tasks in the process and o is the number of obligations. Since checking whether a state contains a literal can be done in constant time, it does not affects the complexity of the algorithm. The above asymptotic time bound is at most polynomial in the length of the input (which includes the process and the set of obligations).

I conclude that given a yes-instance of the partial compliance problem, there is a certificate of size polynomial in the length of the input (namely the trace that satisfies the obligations) for which we can check compliance in time polynomial in the length of the input. Following from Definition 37, verifying that a single trace is compliant with a regulatory framework is sufficient to infer that the process model containing that trace is partially compliant with the same regulatory framework. Therefore I can conclude that verifying Partial Compliance is indeed in **NP**.

NP-Hardness

After having proven the **NP** membership of the problem in Section 5.1.2, to prove that the problem is **NP**-complete we have to show that the problem is **NP**-Hard.

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

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

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

Definition 39 (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. $\forall i, j((v_i, v_j \in ham \wedge j = i + 1), ((v_i, v_j) \in D)$

Let $\leq_p$ denote polynomial time reduction. Using this I can express my current claim, stating that the problem of proving partial compliance of a given process is at least as difficult as the problem of finding an hamiltonian path in a given graph.

Claim 2. Hamiltonian Path Problem $\leq_p$ Proving Partial Compliance in $C2_{nla}$

Given a directed graph $G = (N, D)$, we reduce the problem of deciding whether G contains a hamiltonian path to the decision problem of deciding whether a process (P, ann) is partially compliant with a regulatory framework of a $C2_{nla}$ problem, where such process is required to contain at least one trace fulfilling a set of local obligations.

Reduction

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, ann) and a set of obligations $\circledcirc$ as follows:

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

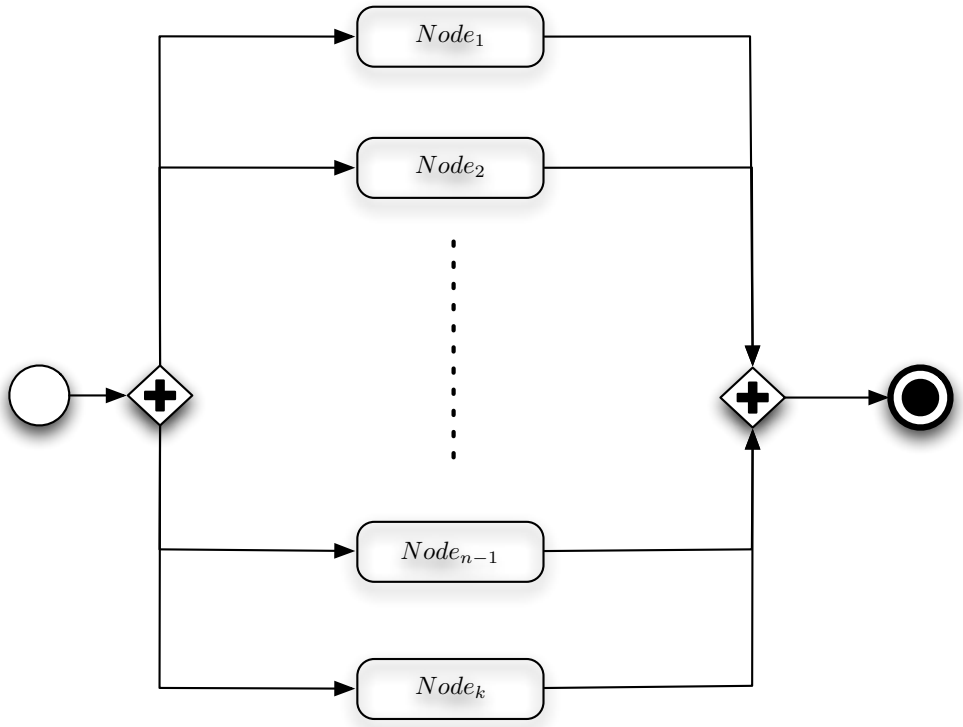

Figure 5.2: Hamiltonian path problem as verifying partial compliance.

The main process block B 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, 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 $\circledcirc$ contains the following local maintenance obligations:

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

Reduction 1. *Given a directed graph G and the problem of proving regulatory compliance reduced from it, where the problem is composed by a process model (P, ann) and a set of local obligations $\circledcirc$. There exists a trace $\theta \in \Theta(P, \mathsf{ann})$ such that $\theta \vdash \circledcirc$ if and only if G has an hamiltonian path.*

Complexity:
The complexity of reducing the input of an hamiltonian path problem to proving partial compliance of a problem $C2_{nla}$ is polynomial in terms of the size of the input. The time complexity of constructing the process is $\mathbf{O}(n^2)$, where n is the number of vertices in G. The time complexity of constructing the process model is $\mathbf{O}(n)$, while the time complexity of constructing the regulatory framework is $\mathbf{O}(e)$, where e is the number of edges in G. Since e is at most $n \times n$, we can conclude that the time complexity of the reduction is dominated by the complexity of constructing the regulatory framework, which is $\mathbf{O}(n^2)$.

5.1.3 Proving Non Compliance is coNP-complete

In this second part of the section I show that proving non compliance of a problem $C2^{-}_{nla}$ is a co**NP**-complete problem.

Definition 40 (coNP-complete)**.** *A decision problem is co**NP**-complete if it is in co**NP** and if every problem in co**NP** is polynomial-time many-one reducible to it. A decision problem is in co**NP** if and only if its complement is in the complexity class **NP**.*

I prove that this problem belongs to this complexity class by proving that the complementary problem is **NP**-complete.

***Not* non compliance is in NP-complete**

I define now the complementary problem proving non compliance of $C2^{-}_{nla}$. Following from Definition 37, non compliance of a process (P, ann) with respect to a set of obligations $\circledcirc$ is defined as follows:

Non Compliance: $(P, \mathsf{ann}) \not\Vdash \circledcirc$
iff $\neg\exists\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc$.

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

Definition 41 (*Not* Non Compliance)**.** *Given a process* (P, ann) *and a set of obligations* $\circledcirc$*.*

Not *Non Compliance*:
$\neg(P, \mathsf{ann}) \not\Vdash \circledcirc$ *iff* $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc$*.*

We can notice now that the condition to verify whether a process is not non compliant is exactly the same required to verify whether a process is partially compliant with a set of obligations. Thus in this case proving partial compliance of $C2^{-}_{nla}$ is the complementary of proving not compliance of $C2^{-}_{nla}$. Therefore having already shown in Section 5.1.2 that proving partial compliance in a sub problem $C2^{-}_{nla}$ is **NP**-complete, it follows that proving non compliance in a sub problem $C2^{-}_{nla}$ is co**NP**-complete.

5.1.4 Proving Full Compliance is in coNP

In the last part of this section I show that proving full compliance of a sub-class $C2^{-}_{nla}$ is co**NP**. Differently than for the problem of verifying not compliance, in this case I am not able to verify that the problem is also complete due to the lack of expressiveness given by the restriction stating that the elements composing an obligation are literals. However, I show in the next section that if we drop this restriction, the problem of verifying full compliance can be proven to be co**NP**-complete.

***Not* full compliance is in NP**

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

According to Definition 37, full compliance with respect to a set of obligations $\circledcirc$ is defined as follows:

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

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

Definition 42 (*Not* Full Compliance)**.** *Given a process* (P, ann) *and a set of obligations* $\circledcirc$*.*

Not *Full Compliance*: $\neg(P, \mathsf{ann}) \vdash^{F} \circledcirc$
iff $\exists \theta \in \Theta(P, \mathsf{ann}), \theta \nvdash \circledcirc$*.*

From Definition 42 it follows that to verify not full compliance it is sufficient to show that there exists a trace belonging to the process which does not fulfil the regulatory framework.

***NP** Membership*

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

To verify whether a certificate is indeed a valid trace of the process we can reuse Algorithm 3 introduced in Section 5.1.2.

In the same way, we can reuse Algorithm 4 to verify the not full compliance of the process. This can be done because Algorithm 4 returns either $\theta \vdash \circledcirc$ or $\theta \nvdash \circledcirc$. Thus in case the algorithm returns $\theta \nvdash \circledcirc$, according to Definition 42 the certificate proves the not full compliance of a sub-class $C2^{-}_{nla}$ of the problem.

In Section 5.1.2 it is proven that the complexity of both Algorithms 3 and 4 is polynomial in the length of the input, hence the problem of proving not full compliance of a sub-class $C2^{-}_{nla}$ of the problem is indeed in **NP**.

Having shown that the complementary problem of proving full compliance in a sub-class $C2^{-}_{nla}$ is in **NP**, I can therefore conclude that proving full compliance of of a sub-class $C2^{-}_{nla}$ of the problem is in co**NP**.

Notice that in this last case, I have shown only that proving full compliance of a sub-class $C2^{-}_{nla}$ is co**NP** without showing that it is complete like for proving partial and non compliance. My intuition is that limiting the elements of the obligations to be composed only of propositional literals limits the expressivity of the problem in such a way that is computational complexity is only co**NP**. However this is only an intuition and whether proving full compliance of a sub-class $C2^{-}_{nla}$ is co**NP** or co**NP**-complete is still an open question.

5.2 Computational Complexity of $C2_{nla}$

In this second part of the chapter I analyse the complexity of proving regulatory compliance of a sub-class $C2_{nla}$ of the problem, highlighted in the lattice of the sub-classes of the problem illustrated in Figure 5.3. More precisely I show that by releasing the restriction of limiting the elements composing the obligations to be composed of propositional literals does not increase the complexity of the problem. I show in fact that the complexity of proving both partial and non compliance of a sub-class $C2_{nla}$ is the same as proving them in a

sub-class $C2^{-}_{nla}$. Moreover given the enhanced expressivity due to releasing the restriction, I can show that the problem of proving full compliance of a sub-class $C2_{nla}$ is co**NP**-complete.

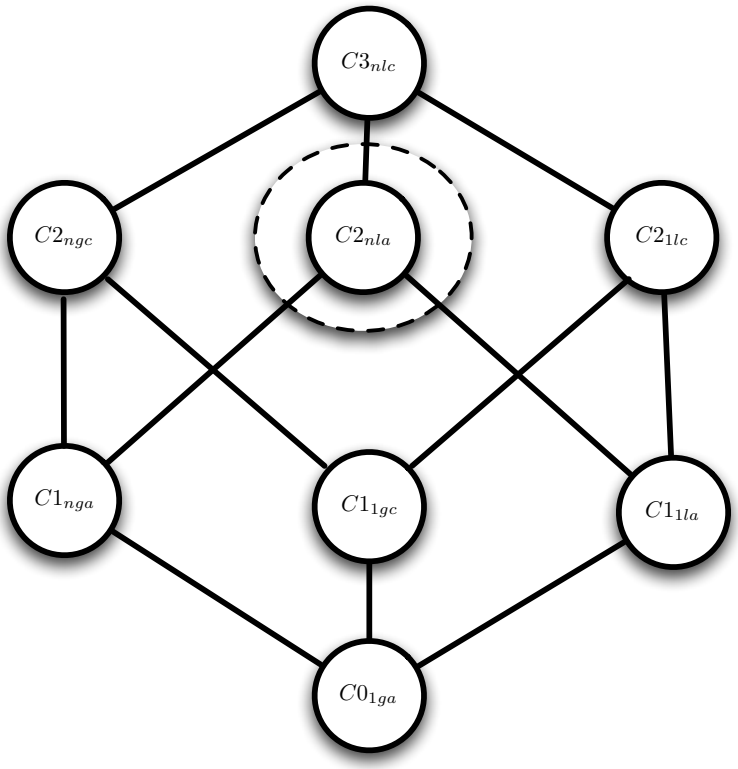

Figure 5.3: Multiple Local Obligations Compliance Problem

Allowing propositional formulae as the elements composing the obligations does not influence the processes, hence their semantics remains the same as the one defined in Section 3.1.

The current sub-class uses obligations composed of propositional formulae, which I redefine as follows.

Definition 43 (Local Obligation with formulae)**.** *A local obligation* $\mathbf{O}$ *is a structure* $\langle t, \varphi_c, \varphi_l, \varphi_d \rangle$*, where* $t \in \{s, a, m\}$ *and represents the type of the obligation. The elements* φ_c, φ_l *and* φ_d *are propositional formulae composed by elements in* $\mathcal{L}$*.* φ_c *is the content of the obligation,* φ_l *is the trigger (lifeline) of the obligation and* φ_d *is the deadline of the obligation.*

I use $\mathbf{O} = \mathcal{O}^t\langle \varphi_c, \varphi_l, \varphi_d \rangle$ *to represent a local obligation.*

Differently from the sub-class $C2^{-}_{nla}$, where propositional literals are verified through inclusion in a state, in the sub-class $C2_{nla}$ the propositional formulae, used to represent the lifeline, deadline and condition of an obligation, are satisfied in a state if and only if the interpretation of the propositional variables given by such state makes the propositional formulae true.

Definition 44 (Formula Entailment)**.** *Given a state $\sigma = (t, L)$ and a formula φ, $\sigma \models \varphi$ if and only if $\bigwedge x \models \alpha$, where each $x \in L$.*

Similarly as the previous part of the chapter, the present part is divided in three parts: the first part analyses the complexity of verifying partial compliance, the second part the complexity of verifying not compliance and the third part the complexity of verifying full compliance.

5.2.1 Proving Partial Compliance is NP-complete

By lifting the restriction limiting the elements composing the obligations to be composed of simple literals, Algorithm 4 is not capable of handling the problems in this new setting. Therefore I propose here a revised version that takes into account the fact that the elements composing an obligation consists of a formula.

Algorithm 5. *Given a set of obligations $\circledcirc$ and a trace $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ such that $\sigma_{start} = (\mathsf{start}, L_0)$ and $\theta \in \Theta(P, \mathsf{ann})$, the algorithm $A_5(\theta, \circledcirc)$ is defined as follows (In the following, Ob denotes the set of active obligations and we treat θ as a vector):*

Algorithm A_5

1: $\mathsf{Ob} = \emptyset$
2: ***for each*** σ_i *in* θ ***do***
3: ***for each*** $\mathcal{O}^t\langle\varphi_c, \varphi_b, \varphi_d\rangle$ *in* $\circledcirc$ ***do***
4: ***if*** $\sigma_i \models \varphi_b$ ***then***
5: $\mathsf{Ob} = \mathsf{Ob} \cup \mathcal{O}^t\langle\varphi_c, \varphi_b, \varphi_d\rangle$
6: ***end if***
7: ***end for each***
8: ***for each*** $\mathcal{O}^t\langle\varphi_c, \varphi_b, \varphi_d\rangle$ *in* Ob ***do***
9: ***if*** $t = a$ ***then***
10: ***if*** $\sigma_i \models \varphi_c$ ***then***
11: $\mathsf{Ob} = \mathsf{Ob} \setminus \mathcal{O}^a\langle\varphi_c, \varphi_b, \varphi_d\rangle$
12: ***else***
13: ***if*** $\sigma_i \models \varphi_d$ ***then***
14: ***return*** $\theta \nvdash \circledcirc$
15: ***end if***
16: ***end if***
17: ***else***
18: ***if*** $t = m$ ***then***
19: ***if*** $\sigma_i \not\models \varphi_c$ ***then***
20: ***return*** $\theta \nvdash \circledcirc$

21: ***end if***
22: ***if*** $\sigma_i \models \varphi_d$ ***then***
23: $\mathsf{Ob} = \mathsf{Ob} \setminus \mathcal{O}^m\langle\varphi_c, \varphi_b, \varphi_d\rangle$
24: ***end if***
25: ***end if***
26: ***end if***
27: ***end for each***
28: ***end for each***
29: ***return*** $\theta \vdash \circledcirc$;

Algorithm 5 identifies wether a trace fulfils a set of local obligations without compensations.

We can immediately see that Algorithm 5 differs from Algorithm 4 only because of the way the lifelines, deadlines and fulfilment conditions of the obligations are verified in the states due to being composed of propositional formulae, which follows Definition 44.

Verifying the entailment of a formula with respect to a set of literals is indeed more difficult than simply checking whether a literal belongs to a set. However, as shown in the following complexity analysis, Algorithm 5 is still capable of returning a result in polynomial time.

Complexity:

The complexity of checking whether a trace is compliant with the set of obligations using Algorithm 5 is at most polynomial in time $\mathbf{O}(n \times o \times T)$ where n is the number of tasks in the process, o is the number of obligations and T is the maximum time to check whether a state satisfies a formula. Since checking whether a state satisfies a propositional formula can be done in time that is at most polynomial (in fact linear) in the length of the formula, the above asymptotic time bound is at most polynomial in the length of the input (which includes the process and the set of obligations, including the associated formulae).

Proving Partial Compliance is still in NP

As done in the previous part of the chapter, I prove now the complexity of proving partial compliance of a process by considering as a certificate a trace of the process being evaluated. If the certificate is a valid trace of a process and is compliant with the regulatory framework, then from Definition 37 it follows that the process is partially compliant with the regulatory framework.

Since introducing the formulae did not changed the way traces are constructed from a process, to verify the validity of a certificate passed to Algorithm 5 we can reuse Algorithm 3 defined in Section 5.1.2.

Therefore I can conclude that proving partial compliance of a $C2_{nla}$ problem is indeed in **NP**, even after lifting the constraint of having the elements composing the obligations to be single literals.

Proving Partial Compliance is still NP-complete

The reduction proposed in Section 5.1.2 is still valid for the problem involving obligations which elements are composed of formulae. This is true since a formula can consists of a single literal.

Therefore it follows that the problem of proving partial compliance of problem $C2_{nla}$ where the elements composing an obligation are composed of formulae is still **NP**-complete.

5.2.2 Proving Non Compliance is coNP-complete

The complexity of proving non compliance does not change after having lifted the restriction of limiting the conditions to be composed only of literals.

Having already proven that the problem of proving partial compliance is **NP**-complete, I can reuse the proof in Section 5.1.3 showing that the complementary problem of verifying non compliance is verifying partial compliance. Therefore it follows that the complexity of the problem at hand is still co**NP**-complete even after lifting the restrictions.

5.2.3 Proving Full Compliance is coNP-complete

I show that the problem of proving full compliance is still in co**NP**. Moreover thanks to the added expressivity given by the releasing of the constraints over the obligations' elements, I can now prove that this problem is co**NP**-complete.

To prove that a decision problem is co**NP**-complete I have to show that the complementary problem is in **NP** and that the another co**NP**-complete problem, in this case *tautology*, can be reduced to the problem at hand.

Since the different types of compliance have not changed with the lifting of the constraints, I can reuse the same proof as the one in Section 5.1.4 to prove that the complementary problem is in **NP**.

Therefore what is left to prove is that another co**NP**-complete problem is reducible to the problem of verifying full compliance.

CoNP Hardness

To show that the problem of proving full compliance of a problem $C2_{nla}$ is co**NP**-complete, I reduce the tautology problem to it.

Definition 45 (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

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

For each literal l belonging to L we construct an XOR block containing two tasks, one labeled and containing in its annotation the positive literal (i.e. l) and the other the negative literal (i.e. $\neg l$). All the XOR blocks constructed from L are then included within a single AND block. This AND block is in turn followed by a task labeled "test" and containing a single literal in its annotation: l_{test}. The sequence containing the AND block and the task *test* is then enclosed within an start and a end, composing the process (P, ann), graphically represented in Figure 5.4.

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 φ:

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

Reduction 2. *Given a propositional formula φ and the problem of proving regulatory compliance reduced from it, where the problem is composed by a process model (P, ann) and a set of local obligations $\circledcirc$. For all traces $\theta \in \Theta(P, \mathsf{ann})$ we have $\theta \vdash \circledcirc$ if and only if φ is a tautology.*

Complexity:

The process P can be constructed in time proportional to $|L| + |\varphi|$ where $|\varphi|$ denotes the length of formula φ. Since $|L| \leq |\varphi|$ by construction, the time is at most polynomial in the length of the formula φ.

5.3 Summary

I show in the first part of this chapter that the sub-class $C2^{-}_{nla}$, namely verifying whether a process is compliant with a set of local obligations, is already hard. More precisely I shown that proving partial compliance of a sub-class $C2^{-}_{nla}$ is **NP**-complete, proving not compliance in a sub-class $C2^{-}_{nla}$ is co**NP**-complete and proving full compliance of a sub-class $C2^{-}_{nla}$ is in co**NP**. Whether the sub-class $C2^{-}_{nla}$ is in co**NP**-complete remains an open question.

The preliminary results contained in this section help already to explain why existing solutions for the general problem, like Governatori and Hoffmann [33], Ghose and Koliadis [30], Governatori et al. [35] and van der Aalst et al. [78] are not efficient, or in case they are, the solutions provided are either an approximation of the real ones or are provided for a simplified version of the general problem of proving regulatory compliance.

In the second part of this chapter I show that even by releasing the constraint of having only literals composing the elements of local obligations, the complexity of proving the different types of compliance of a $C2_{nla}$ does not increase. Moreover the added expressivity allows to prove that proving full compliance of a $C2_{nla}$ problem is co**NP**-complete.

Given that the goal of the present thesis is to identify the computational complexity of the general problem of proving regulatory compliance, In the following chapters I focus on the

sub-classes allowing to represent the elements composing the obligations using propositional formulae. Moreover given that the computational complexity does not increase while getting rid of the restriction over the elements composing the obligations. Even if it has been proven only for proving partial and non compliance in the subclass $C2_{nla}$, is most likely that the computational results obtained for the non restricted sub-classes of problems also hold for the ones which only allow propositional literals to describe the elements of the obligations.

The present chapter partially answers the research subquestion **RSQ1** by studying the computational complexity of two sub-classes of the problem: the sub-class $C2_{nla}$ and the sub-class $C2^{-}_{nla}$.

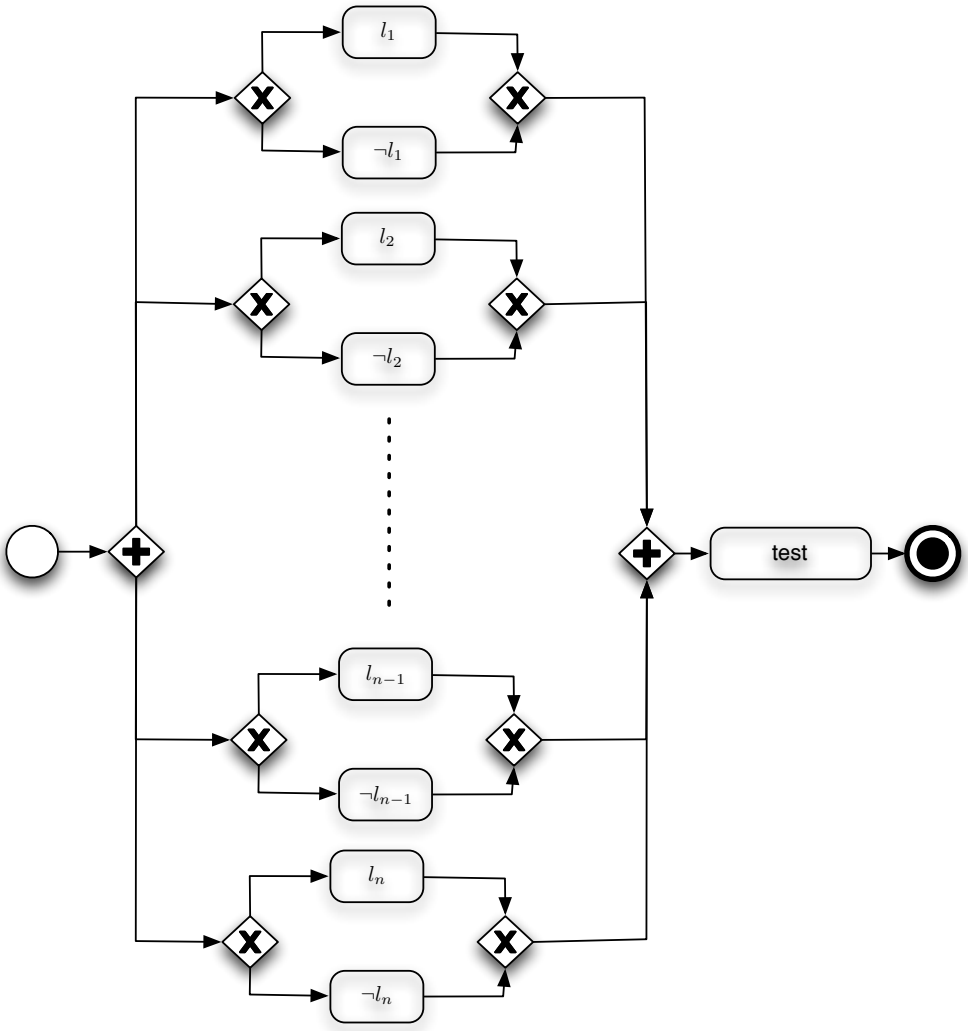

Figure 5.4: Tautology problem as verifying full compliance.

Chapter 6

Propagating the Results

In the previous chapter I have shown that for the sub-class $C2_{nla}$ of the problem, the computational complexity of proving either full or non compliance is co**NP**-complete, while proving partial compliance is **NP**-complete. I have also shown that the same computational complexity results apply to the sub-class $C2^{-}_{nla}$, with the exception of proving full compliance, which is proven to be co**NP** because of the limited expressivity following from limiting the elements of the obligation to propositional literals.

The present chapter is divided into two main parts. The first part analyses the complexity of the more general sub-class $C3_{nlc}$ of the problem, corresponding to the general problem of proving regulatory compliance. In this part I reuse most of the complexity proof used to prove the complexities for the sub-class $C2_{nla}$ with some differences in order to take into account the introduction of the compensable obligations.

The second part of this chapter focuses on identifying the computational complexity of other sub-classes of the problem by reusing the obtained results. In particular I analyse the computational complexity of the sub-classes $C1_{1la}$ and $C2_{1lc}$.

6.1 Computational Complexity of $C3_{nlc}$

I analyse in his section the computational complexity of the general problem or proving regulatory compliance. The problem corresponds to the sub-class $C3_{nlc}$, highlighted in the cube obtained by the difficulty vectors in Figure 6.1. The general problem of proving regulatory compliance is composed of a regulatory framework containing a set of local compensable obligations. The obligations contained in the framework have dynamic activation periods which depend on the trace where they are being evaluated and are compensable, meaning that a trace violating an obligation can still be considered to be compliant in case it complies with the obligation assigned to compensated the violated obligation.

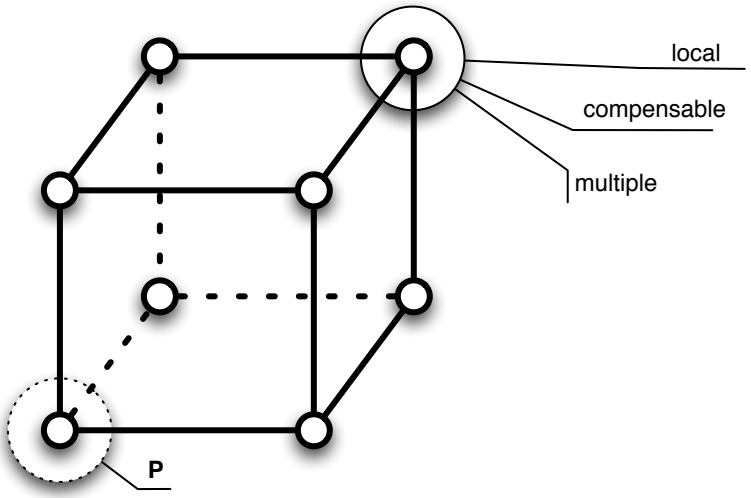

Figure 6.1: Multiple Local Compensable Obligations Compliance Problem

6.1.1 Complexity of the Problem

To show the complexity of the general problem I reuse part of the proof used to prove the complexity of the sub-class $C2_{nla}$ discussed in Section 5.2. The difference between the two problems consists in the compensable obligations available in the general problem. Therefore I first introduce an updated version of Algorithm 5 capable of dealing with compensable obligations.

I show that the computational complexity of solving the general problem of proving regulatory compliance is the same as the computational complexity of solving the sub-class $C2_{nla}$. To show this I can reuse Algorithm 3 to verify whether a given certificate, a trace, is a proper trace of a given process. I can reuse the same algorithm since the structure of the process has not been altered. To complete the proof that the general problem belongs in **NP** I also have to show that the computational complexity of the updated version of Algorithm 5 is still polynomial.

As I discuss more in details later, to prove the **NP**-Hardness of the problem and therefore its **NP**-Completeness for proving partial compliance and co**NP**-Completeness for proving full and non compliance, I can reuse the reductions used for the sub-class $C2_{nla}$ given that it is a special case of the general problem.

Proving Partial Compliance is NP-Complete

Before proceeding to introduce the updated version of Algorithm 5, capable of dealing with compensable obligations, I provide a slightly different definition for Compensable

obligations, which provides a representation better suited to be handled by the updated version of the verification algorithm.

Definition 46 (Compensable Obligation List)**.** *A compensable obligation* $\lightning = \mathbf{O}_1 \otimes \ldots \mathbf{O}_n \otimes \bot$ *is a sequence of obligations, where the first is activated when a state satisfies its lifeline and the following ones are activated when the preceding obligation is active and violated. The last element of the sequence is* $\bot$*, which is not a proper obligation, but a symbol used by the algorithm to identify that the compensation chain has ended.*

From now on I am treating a compensable obligation as an ordered list, where each element of the list corresponds to the relative obligation in the sequence. Notice that the shortest sequence possible is composed by a compensable obligation where no compensations are provided, represented as follows: $\lightning = \mathbf{O}_1 \otimes \bot$*.*

Given a compensable obligation $\lightning$*, at most one of the obligations composing it can be active at a time, and I use two methods to determine which one is active. The method* first*, used as* $\lightning.\mathsf{first}$*, activates the first obligations of the list and the method* next*, used as* $\lightning.\mathsf{next}$ *deactivates the currently active obligation in the list and activates the following one.*

The semantics of compensable obligations is the same as the one defined by Definition 28. This means that only a single instance of each compensable obligation can be active in a single moment. More precisely, in addition that only a single instance of a compensable obligation can be active, only one obligation belonging to the chain can be active at a given point in time.

Verifying whether a compensable obligation is fulfilled by a trace can be done by checking whether the last obligation composing it, the one preceding the symbol $\bot$ according to Definition 46, is fulfilled by the trace. This is formalised by the following lemma.

Lemma 6. *Given a compensable obligation* $\lightning = \mathbf{O}_1 \otimes \ldots \otimes \mathbf{O}_{n-1} \otimes \bot$ *composed of a chain containing* $n-1$ *obligations,* $\theta \vdash \lightning$ *if and only if* $\theta \vdash \mathbf{O}_{n-1}$*.*

The correctness of Lemma 6 can be intuitively explained by considering two main points. The first one is that independently of the type of the obligation, an obligation which is not activated is considered to be fulfilled. The second point is that the activation period of the last obligation composing a compensable obligation depends on the activation periods of the previous ones, more precisely it depends on the violations of the previous obligations. Therefore we can see now that either the obligation is activated and fulfilled or one of the previous obligation is fulfilled, leading to the last obligation to not be activated and therefore fulfilled, following from Definitions 23 and 24 about the semantics of local obligations.

Algorithm 6. *Given a set of obligations* $\circledcirc$ *and a trace* $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ *such that* $\sigma_{start} = (\mathsf{start}, L_0)$ *and* $\theta \in \Theta(P, \mathsf{ann})$*, the algorithm* $A_6(\theta, \circledcirc)$ *is defined as follows (In the following,* Ob *denotes the set of active obligations and I treat* θ *as a vector of states):*

Algorithm A_6

1: $\mathsf{Ob} = \emptyset$

```
2: for each σ in θ do
3:   for each ϟ in ⊙ do
4:     if ϟ not in ob then
5:       Let ϟ = 𝛰_1 ⊗ ... ⊗ ⊥ and let 𝛰_1 = O^t⟨φ_c, φ_b, φ_d⟩
6:       if σ_i ⊨ φ_b then
7:         Ob = Ob ∪ ϟ.first
8:       end if
9:     end if
10:  end for each
11:  for each ϟ in Ob do
12:    w = true
13:    while w do
14:      w = false
15:      Let the active obligation in ϟ be O^t⟨φ_c, φ_b, φ_d⟩
16:      if t = a then
17:        if σ_i ⊨ φ_c then
18:          Ob = Ob \ ϟ
19:        else
20:          if σ_i ⊨ φ_d then
21:            ϟ.next
22:            w = true
23:            if The active obligation in ϟ is ⊥ then
24:              return θ ⊬ ⊙
25:            end if
26:          end if
27:        end if
28:      else
29:        if t = m then
30:          if σ_i ⊭ φ_c then
31:            ϟ.next
32:            w = true
33:            if The active obligation in ϟ is ⊥ then
34:              return θ ⊬ ⊙
35:            end if
36:          end if
37:          if σ_i ⊨ φ_d then
38:            Ob = Ob \ ϟ
39:          end if
40:        end if
41:      end if
42:    end while
43:  end for each
44: end for each
45: return θ ⊢ ⊙;
```

Algorithm 6 identifies wether a certificate, consisting of a trace θ, fulfils a set of local compensable obligations.

The first line of the algorithm A_6 initialises the set of active compensable obligations, which is empty at the beginning. The algorithm proceeds then to analyse the states of the trace given in input, following the order in which they appear. For each state, the algorithm verifies for each non active compensable obligation if it is activated by the state being processed.

After, the algorithm proceeds for each active obligation to verify whether it is violated, fulfilled or if this cannot be determined yet, meaning that in this last case the current active obligation of a chain remains active for the following state of the trace.

Depending on the type of the obligation, the condition is checked against the state. If the obligation is fulfilled, then the whole chain is removed from the set of active ones. Otherwise, if the obligation is violated, then the obligation is deactivated and the following one in the chain is activated and checked in the same state. In this case if a $\perp$ obligation becomes active, then the algorithm returns not compliance $\theta \nvdash \circledcirc$, because it means that the trace violated every obligation belonging to a chain.

If none of the $\perp$ obligations become active during the verification process, the algorithm returns compliance $\theta \vdash \circledcirc$, since each compensable obligation is either fulfilled or each of it violations has been compensated.

Complexity:

The computational complexity of verifying whether a trace is compliant with a set of compensable local obligation using Algorithm 6 is at most polynomial in time $\mathbf{O}(n \times o \times T \times z)$. It can be noticed that the computational complexity is the same as the one for Algorithm 5, which is $\mathbf{O}(n \times o \times T)$, multiplied by z which is the length of the longest compensation chain in $\circledcirc$. As already stated, Algorithm 6 updates Algorithm 5 by analysing the chains of obligations composing the compensable obligations. Therefore the computational complexity is multiplied by the number of obligations composing the chains, for which I take the longest as the worst case.

Given that the sub-class $C2_{nla}$ is a particular case of the general problem $C3_{nlc}$ addressed in this section. Because a sub-class $C2_{nla}$ can be described as a sub-class $C3_{nlc}$ where each obligation $\mathcal{O} \in \circledcirc$ is translated as a compensation chain composed by the obligation itself plus the symbol $\perp$ as follows: $\lightning = \mathcal{O} \otimes \perp$. Therefore I can reuse the exact reduction proposed for the sub-class to prove the **NP**-hardness of the general problem. Recalling that the reduction reduces the problem of finding whether an hamiltonian path exists in a directed graph to a problem of proving partial compliance as stated by Reduction 1. I have therefore shown that the computational complexity of verifying partial compliance of a business process model in the general case, where the process has to be verified against a set of compensable local obligations, is **NP**-complete.

Proving Non Compliance is coNP-Complete

Since it has been already shown in in Section 5.1.3 that the problem of proving partial compliance is the complement of proving non compliance, the computational complexity

of proving non compliance in the general problem does not increase with respect to the sub-class $C2_{nla}$ of the problem and remains co**NP**-complete.

Proving Full Compliance is coNP-Complete

Similarly as it has been shown in Section 5.1.4 to prove that proving full compliance in a sub-class $C2^{-}_{nla}$ of the problem is co**NP**, I show now that the computational complexity of proving full compliance in the general problem is co**NP**-complete.

To show this I reuse Definition 42, stating that the complementary problem of showing *not* full compliance consists of identifying whether a trace is non compliant with a regulatory framework. By using the definition and Algorithm 6 it can be immediately seen that the algorithm can be used to identify whether a trace is non compliant with a set of compensable local obligations, from which it follows that of showing *not* full compliance in the general problem is in **NP**. Therefore showing full compliance in the general problem is indeed in co**NP**.

To show the co**NP**-hardness of this problem I reuse a previous reduction as it has been done for the problem of proving partial compliance. In this case I can reuse Reduction 2, where the problem of verifying whether a propositional formula is a tautology is reduced to verifying full compliance in a sub-class $C2_{nla}$ of the problem. Since I have already shown that a sub-class $C2_{nla}$ is a particular case of the general problem and the tautology problem is a known co**NP**-complete problem, I can conclude that indeed proving full compliance in the general problem is in co**NP**-complete, which is the same computational complexity class as for the sub-class $C2_{nla}$ of the problem.

6.2 Further Propagating the Results

In this second part of the present chapter I propagate the complexity results further through the lattice illustrating the set of sub-classes where the elements of the obligations composing the regulatory framework are defined using propositional formulae. In Figure 6.2 shows the lattice and highlights the two sub-classes for which I already provided the complexity results, namely $C2_{nla}$ and $C3_{nlc}$.

From the picture we can see that the sub-classes $C2_{nla}$ and $C3_{nlc}$ have the same computational complexity. I show in this part of the chapter that also the sub-classes $C1_{1la}$ and $C2_{1lc}$ have the same computational complexity.

6.2.1 Computational Complexity of $C1_{1la}$

To prove the computational complexity of this sub-class of the problem I reuse some of the results contained in Section 5.2.3. Differently from the previous section where the complexity analysis started by showing the complexity of proving partial compliance, for this sub-class of the problem I start by showing the complexity of proving full compliance.

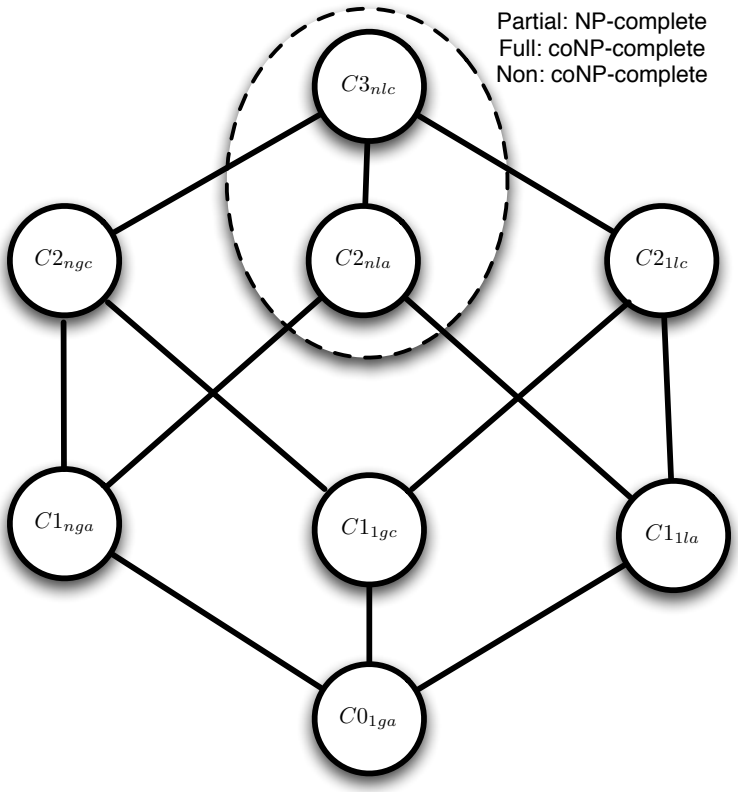

Figure 6.2: Current Complexity Results

Proving Full Compliance is coNP-complete

To show that the computational complexity of proving full compliance in a sub-class $C1_{1la}$ of the problem is in co**NP**-complete I reuse Reduction 2, which reduces the tautology problem to a problem of proving regulatory compliance.

We have already seen that the computational complexity of proving full compliance in the sub-class $C2_{nla}$, where Reduction 2 is used, is co**NP**-complete. Recalling now the procedure reducing a tautology problem to a problem of proving regulatory compliance, the procedure constructs a process model containing the propositions appearing in the propositional formula being evaluated and a single local obligation with the propositional formula set as condition is constructed and needs to be fulfilled by every trace of the model in order to prove full compliance and indirectly that the formula is a tautology.

We can see now that the reduction requires a single local obligation, meaning that the reduction actually reduces the problem of proving that a propositional formula is a

tautology to proving full compliance of a sub-class $C1_{1la}$ of the problem. Moreover we can also see that the sub-class analysed now is a particular case of the sub-class $C2_{nla}$ of the problem studied in Section 5.2.3, from which we can conclude that the sub-class $C1_{1la}$ is in co**NP**. Therefore I can conclude that proving full compliance of a sub-class $C1_{1la}$ of the problem is indeed in co**NP**-complete.

Proving Non Compliance is coNP-complete

Because the sub-class $C1_{1la}$ is a particular case of the sub-class $C2_{nla}$ and proving non compliance of the latter problem is in co**NP**, I can conclude that the proving non compliance of $C1_{1la}$ is at most as complex as the sub-class $C2_{nla}$. Therefore to prove that proving non compliance in a sub-class $C1_{1la}$ is in co**NP**-complete I have to show that a co**NP**-complete problem is reducible to it.

I consider now a know co**NP**-complete problem similar to the tautology problem, the contradiction problem, which consists of proving that a given propositional formula is not satisfied by any of its propositional assignments. This means that no matter what truth values are assumed by the propositions composing the formula, the truth value of the formula is always false.

Definition 47 (Contradiction)**.** *A propositional logic formula is a contradiction if the formula itself is always unsatisfiable regardless of which evaluation is used for the propositional variables.*

To prove the computational complexity of the sub-class I can reuse once more Reduction 2. Even if the reduction involves reducing a tautology problem, from the axioms of propositional logic it follows the following lemma.

Lemma 7. *Let α be a tautology and β be a contradiction. $\alpha \equiv \neg\beta$.*

The lemma states that the negation of a tautology is a contradiction and vice versa. The statement follows from the semantics of the negation which flips the truth value, meaning that a tautology negated is always false, which corresponds to a contradiction.

From Lemma 7 it follows that a tautology problem can be transformed in a contradiction problem by negating the formula. Therefore by reusing Reduction 2 and negating the starting formula, what we are actually reducing is a contradiction problem. Therefore if the formula being evaluated is indeed a contradiction, then none of the traces (corresponding to every possible evaluation) of the process model would fulfil the obligation resulting from the reduction, which is the following: $\mathcal{O}^s\langle\neg\varphi, l_{test}, \top\rangle$ where φ is a tautology.

Therefore according to Definition 16 I can conclude that the reduced problem returns non compliance in case the negated starting obligation is a contradiction. Thus I can conclude that proving non compliance of a sub-class $C1_{1la}$ is indeed in co**NP**-complete.

Proving Partial Compliance is NP-complete

Having shown that proving non compliance of a sub-class $C1_{1la}$ of the problem is co**NP**-complete and knowing that proving partial compliance is the complement of proving non compliance as shown in Definition 41. I can conclude, since the complement of a co**NP**-complete problem is **NP**-complete, that proving partial compliance of a sub-class $C1_{1la}$ of the problem is indeed in **NP**-complete, which is the same computational complexity identified for the sub-classes $C2_{nla}$ and $C3_{nlc}$.

6.3 Computational Complexity of $C2_{1lc}$

I show now the last complexity analysis of the chapter in this section, where I show that the computational complexity of the sub-class $C2_{1lc}$ is equivalent to the other sub-classes already studied in the lattice shown in Figure 6.2. Differently from the other analysis performed for the other sub-classes of the problem, I show the computational complexity of the sub-class at hand using a comparative analysis with the results already obtained.

Before proceeding I formally discuss the relations between the sub-classes of the problem contained in the lattice in Figure 6.2. The lattice shows the sub-classes by ordering them from the simplest, at the bottom, to the most complex, at the top. The connections between the sub-classes represent that a simpler problem is a particular case of the more complex one to which it is connected. Meaning that the complexity of a simpler sub-class connected to a more complex one is at most the complexity as the more complex sub-class.

Proposition 4 (Complexity Relations)**.** *Let $Cx_{\mathcal{F}}$ and $Cy_{\mathcal{F}'}$ be sub-classes of the problem of proving compliance, where x and y enumerate the amount of difficult features in each of the sub-classes and, $\mathcal{F}$ and $\mathcal{F}'$ are sets representing which difficult features are included in each sub-class. The sub-class $Cx_{\mathcal{F}}$ is at most as computationally complex as the sub-class $Cy_{\mathcal{F}'}$ if and only if $\mathcal{F} \subseteq \mathcal{F}'$.*

The proposition introduced can be shown graphically in the figure illustrating the lattice by considering the directly connected sub-classes of the problem. A sub-class Cx is at most as computationally complex as a sub-class Cy, if Cx is directly connected to Cy and $x < y$. Notice that the complexity relations between the sub-classes are symmetric and transitive. The correctness of the proposition shown immediately follows from the fact that solving a restricted problem is at most as difficult as solving a more general related problem.

Using the proposition introduced and considering the complexity results already obtained, we can see that the sub-class $C2_{1lc}$ of the problem is at most as complex as the sub-class $C3_{nlc}$. Moreover we also know that the sub-class $C1_{1la}$ is at most as complex as the sub-class $C2_{1lc}$. Since I have already shown that the complexity of the sub-class $C1_{1la}$ is equivalent to the complexity of the sub-class $C3_{nlc}$, I can then conclude that also the sub-class $C2_{1lc}$ share the same computational complexity.

Therefore I can conclude that also in a sub-class $C2_{1lc}$ of the problem of proving regulatory compliance, proving partial compliance is **NP**-complete, proving full compliance is co**NP**-complete and proving non compliance is co**NP**-complete.

6.4 Summary

In the present chapter I propagated the existing complexity results to some of the other sub-classes of the problem of proving regulatory compliance, in particular I propagated the complexity results obtained for the problem $C2_{nla}$. We can see in Figure 6.3 the updated complexity results, which now includes in the set of sub-classes where proving partial compliance is **NP**-complete and proving both full and non compliance is co**NP**-complete also the sub-classes $C1_{1la}$, $C2_{1lc}$ and $C3_{nlc}$.

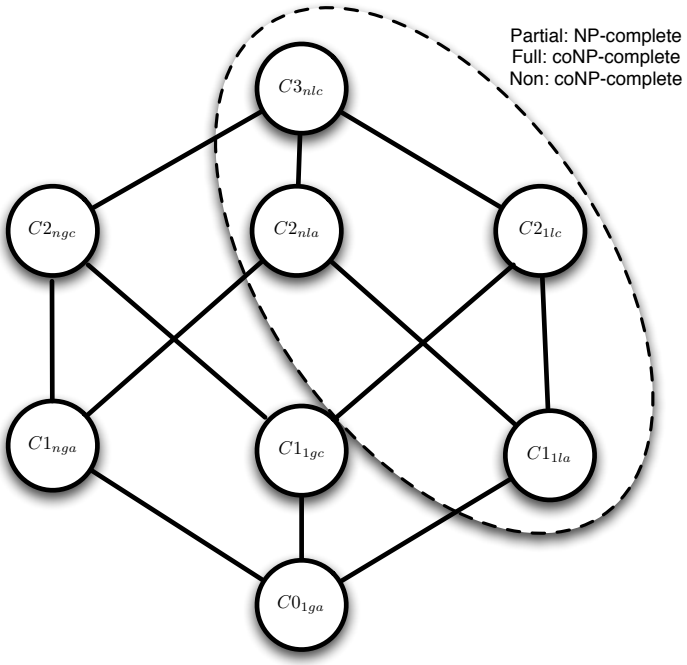

Figure 6.3: Complexity Results Updated

From Figure 6.3 we can see that each of the sub-classes of the problem for which I prove their computational complexity, the sub-class contains among the difficult features the local obligations, which I recall to be obligations whose activation periods are variable

and depend on the traces being analysed.

From the present state of the analysis it is possible to see that local obligations appear to be a source of the complexity. However I cannot claim them to be the source of the complexity since I have not provided any complexity result for the sub-class $C0_{1ga}$, therefore I cannot exclude the case where such problem is as difficult as the ones analysed till now. Especially since there is currently no proof that the complexity of this sub-class is polynomial as is instead shown for the sub-class $C0^{-}_{1ga}$.

Nevertheless the present chapter manages to answer partially to the research subquestion **RSQ1**: "What is the computational complexity of the sub-classes of the problem of proving regulatory compliance?" which, despite being a partial answer since manages to determine the computational complexities only of the sub-classes $C1_{1la}$, $C2_{nla}$, $C2_{1lc}$ and $C3_{nlc}$, it determines the complexity of the sub-class corresponding to the general problem analysed in this thesis, which answer the main research question of the thesis: **RQ** "What is the computational complexity of the general problem of proving the regulatory compliance of a business process model?". The results identified in the present chapter, also identify the upper bound of the computational complexity of the problem studied in this thesis. Additionally, the results obtained explain why no efficient solutions have been proposed in the literal for complex enough problems of proving regulatory compliance, since such efficient results do not exists[1].

Despite the present chapter answers to the main research question, the second sub-question remains still unanswered **RSQ2** "Which are the sub-classes of the problem of proving regulatory compliance that are non-trivial and tractable?". Despite having proven that the sub-class $C0^{-}_{1ga}$ of the problem is tractable and having shown that it is expressive enough to handle some real case scenarios, such sub-class of the problem is far from being *non-trivial*. Moreover other sub-classes of the problem may be still identified as tractable. Thus I further analyse the problem in the following chapter looking to answer the second research subquestion. The results obtained in this chapter suggest that the source of the complexity is most likely not isolated in the regulatory framework, hence in the following chapter, dedicated to finding a non-trivial tractable sub-class, I analyse a sub-class not yet discussed but resulting from restricting the expressivity of the process model itself along with the expressivity of the regulatory framework.

[1] Unless $P \equiv NP$.

Chapter 7

Towards a Tractable Sub-Class of the Problem

The present chapter focus on the second research subquestion **RSQ2**: "Which are the sub-classes of the problem of proving regulatory compliance that are non-trivial and tractable?". In the previous chapter I have shown that the computational complexity of proving regulatory compliance of a business process is in the general case hard. This means that polynomial time solutions with respect to the size of the problem are not possible. Thus the time necessary to prove the compliance of a business process can be much higher when compared with the size of the problem being evaluated.

By dividing the problem of proving regulatory compliance into sub-classes to analyse its computational complexity, I have also shown that the expressivity of the regulatory framework, one of the two elements composing the problem, is not the only source of computational complexity. In fact I have shown in the previous two chapters that the problem is still intractable even when considering regulatory frameworks restricted in their expressivity.

The analysis of the computational complexity of the sub-classes of the problem performed in the previous chapters suggests that tractable solutions for such sub-classes are not possible when the elements composing the obligations are not restricted to propositional literals. Mind that I cannot claim that such solutions do not exists, since the computational complexity of each of the sub-classes of the problem involving obligations whose elements can be expressed by propositional formulae is not exhaustive in the present thesis. Nevertheless propositional formulae do not seem to be the only source of computational complexity since, as it has been shown in Chapter 5, even sub-classes of the problem limiting the expressivity of the elements composing the obligations to propositional literals, such as $C2^{-}_{nla}$, can also be hard.

When limiting the obligations to be expressed through propositional literals, the sub-classes of the problem obtainable by limiting the expressivity of the regulatory framework

does not seem to produce non trivial sub-classes of the problem[1], where solutions in time polynomial with respect to the size of the input are possible. Therefore in the present chapter, starting from the sub-class $C1^{-}_{1la}$ of the problem, I propose a different sub-class, which I refer to as $C1^{*}_{1la}$, obtainable by restricting the semantics of the business process models and for which the computational complexity of proving its regulatory compliance is linear in time with respect to the size of the input.

I conclude the present chapter by analysing why the restriction applied on the business process model to obtain the sub-class $C1^{*}_{1la}$ of the problem, do not allow to obtain tractable sub-classes of the problem when applied to more complex sub-classes of the family of problems C^{*}.

7.1 An Existing Polynomial Solution

Before proceeding to introduce and discuss the sub-class $C1^{*}_{1la}$ of the problem and the solution to prove its compliance in time linear with respect to the size of the problem, I discuss Hoffmann et al. [44] polynomial time solution, which allows to identify approximate solution when considering the generic problem of proving regulatory compliance and allows exact solutions when considering one of its sub-classes. In this section I focus on discussing the limitations adopted by Hoffmann et al. to provide an exact solution to the problem efficiently. Part of the restrictions that they adopted to identify a tractable sub-class can be reused to identify the sub-class of the problem that I efficiently tackle in the present. To get rid of the limitations appearing in the approach adopted by Hoffmann et al. I adopt additional restrictions to simplify the sub-class of the problem being tackled.

The solution proposed by Hoffmann et al. is based on the *I-propagation* algorithm originally introduced by Weber et al. [82]. The I-propagation algorithm identifies for each transition between the tasks of a process model which values can be expected to be true when they are traversed. The algorithm calculates this information by considering the annotations of the tasks and propagating them through the transitions according to their direction and the semantics of the coordinators being traversed.

The approach adopted in the present chapter to solve the sub-class of the problem in polynomial time uses a similar approach to the I-propagate algorithm of Weber et al. This means that also the solution proposed here propagates the values annotated in the tasks through the transitions and coordinators of the process model. However as we will see later on, the two approaches differ since the sub-classes they aim to solve are different. For instance one of these differences, and probably the most visible, is that the I-propagate algorithm uses first-order logic while the solution I propose use propositional logic.

In order to provide a low-order polynomial time solution for the problem of proving

[1] I have shown in Chapter 3 that for the most basic sub-class of the problem, when the elements composing the obligations are limited to propositional literals, then polynomial solutions are possible. However due to the limited expressivity of the sub-class only a small fragment of real problems can be captured.

regulatory compliance, Hoffman et al. adopt the following list of restrictions to reduce the generic problem, as defined in the present thesis as the sub-class $C3_{nlc}$, to a more tractable one.

1. They restrict the regulatory framework to only be composed of obligations of type maintenance. In other words the do not allow obligations of type achievement, while obligations of type standard would be allowed since they are a particular case of maintenance obligations. However as we see from the following restriction adopted by Hoffmann et al., they are not allowed either.

2. The semantics of the obligations allowed is restricted since deadlines are not allowed. The obligations allowed can still use lifelines to identify from when they must be verified, however by not allowing a deadline, the obligation must be verified for the whole remainder of the process once activated. Moreover since according to the previous restriction only obligations of type maintenance are allowed, it means that if an obligation become active, then its condition has to be kept true until the end of the process.

3. The semantics of the regulatory framework is further reduced by not allowing compensable obligations, meaning that only atomic obligations are allowed and a violation of the obligation cannot be compensated by complying with an additional obligation.

4. The results provided for the sub-class of the problem tackled are partly exact and partly approximated since they can guarantee only either the soundness or the completeness of the solution provided. This derives from the fact that they approximate the results of AND and XOR blocks, where by result it is meant which propositions can be considered true after the block is executed.

Thanks to the restrictions adopted, Hoffmann et al. are capable of proving efficiently whether a business process model is compliant with a regulatory framework. However it is necessary to remember that the solutions provided are not free of approximations. Therefore I will later propose a different sub-class of the problem, perhaps more restricting than the one used by Hoffmann et al., but where the solutions provided are exact and the soundness and completeness of the algorithms used to achieve them can be proven.

7.1.1 Structural Restrictions

One of the problematic components of the business process models that also forced Hoffmann et al. to adopt an approximated approach are the AND blocks. In fact also the solution proposed by Ghose and Koliadis [30] is tailored to prove the compliance of process models, which do not contain such type of blocks.

Despite of the great deal of flexibility allowed by AND blocks, which allow to a business process model designer to avoid to impose arbitrary orderings between tasks where such

constraints do not naturally exist, these kind of structures may represent a problem due to amount of the possibilities allowed. Remind that in an AND block each possible interleaving of the tasks belonging to different branches of the block is possible. The amount of executions allowed by such structure is exponential with respect to the number of task contained in the process block. Thus one can see that the amount of possible executions obtainable by such structure can lead to some computational difficulties. Especially in the worst case scenario, where in order to prove the regulatory compliance of the model it may be necessary to consider each of its possible executions, since it may be necessary to ensure that every possible ordering resulting from an AND block does not produce executions not compliant with the obligations composing the regulatory framework.

As a consequence of the complexity brought by AND blocks to the problem of verifying regulatory compliance, I restrict the sub-class of the problem being tackled in the present chapter to *and-less process models*. This particular class of business process models is the one where $\mathsf{and}-\mathsf{join}$ and $\mathsf{and}-\mathsf{split}$ coordinators are not allowed.

A problem that may arise by having to chose a fixed ordering over tasks that in general do not require a particular one, is that the possibilities are artificially restricted to the order adopted. This can lead to cases where a model where an arbitrary ordering has been chosen to approximate an AND block, can be for instance classified as fully compliant with the regulatory framework. However due to the approximation, information about a particular execution ordering of the tasks which would have been not compliant may have been lost, which would have made the *real* process model to be classified as partially compliant instead. The same case can apply to an approximated process classified as not compliant.

Therefore by limiting the processes to be and-less, there is the risk of constructing process models which are not capable of expressing the actual possible executions of the tasks but only a subset of them. To minimise the negative effects of removing AND blocks from the process model, one can decide to represent multiple arbitrary possible orderings of an AND block within a XOR block, representing in this way a less approximated model which has an higher probability of capturing the features of the real problem.

Proposition 5. *Given an* AND *block* $B_A = \mathsf{AND}(B_1, ..., B_n)$*, let* $B_X = \mathsf{XOR}(B'_1, ..., B'_n)$ *be a* XOR *block where* $\Sigma(B_A) \equiv \bigcup_{i=1}^{n} \Sigma(B'_i)$*. Therefore* $\Sigma(B_A) \equiv \Sigma(B_X)$*.*

Proof. Follows directly from Definition 4. □

The proposition introduced above formalises the concept that an AND block can be represented as a XOR block. In particular such translation requires that each branch of the XOR block contains a subset of the possible executions of the AND block and their union is equal to the whole set of possible executions. In case the second condition, that the union is equal to the set of the executions of the AND block is ignored, then the XOR block represents an approximation of the original AND block. Notice that the size of the process block resulting from the transformation is exponential with respect to the size of the original one.

7.2 The Sub-Class $C1^*_{1la}$

I define in this section the limitations that starting from the sub-class $C1^-_{1la}$of the problem lead to the sub-class $C1^*_{1la}$. We already saw in the Chapter 5 and Chapter 6 that simplifying the obligations composing the regulatory framework does not reduce the complexity of the problem except in the most restricted case, consisting of the sub-class C^-_{1ga} of the problem, shown in Chapter 3. Therefore, starting from one of the easiest sub-classes of the problem not proven to be solvable in time polynomial, I further restrict the semantics of the business process models used by taking inspiration from the existing approach mentioned in the present chapter: Hoffmann et al. [44].

The limitation applied to the general problem to obtain the tractable sub-class are the following:

1. The elements defining the obligations composing the regulatory framework are restricted to propositional literals.

2. The business process model does not allow and_split and and_join coordinators, hence AND blocks are not allowed.

3. The evolution of the states composing a trace is forced to be monotonic, in other words when information, described as a propositional literal, becomes true and is introduced in the process' state, then it cannot become false and removed from the state.

The following subsections describe each of the limitation adopted in detail and provide the intuitive reasons why these limitations are necessary to achieve a tractable sub-class of the problem.

7.2.1 Propositional Obligations

In Chapter 3 it has been proven that verifying regulatory compliance of a sub-class $C0^-_{1ga}$ of the problem can be solved in time polynomial with respect to the size of the input. The elements composing the obligations of such sub-class of the problem are limited to be expressed by propositional literals. Given that the sub-class $C1^*_{1la}$ allows only positive literals in its annotations, I further restrict the obligations of such the sub-class of the problem to be composed only of positive literals.

7.2.2 AND-less Business Process Models

The second limitation, also adopted in other approaches, like for instance the approach of Ghose and Koliadis [30], consists of restricting the expressivity of the business process models by not allowing them to use AND blocks.

Using AND blocks in the process models allows to represent tasks whose execution order is irrelevant. The advantage of this construct is that all the possible executions of these tasks, generated by the interleaving of these independent tasks, are represented in a compact way using AND blocks. The disadvantage of these constructs arises when considering the regulatory framework, in particular when the tasks supposed to be independent are not with respect to the obligations composing the framework. If this is the case, then each possible ordering of the problematic tasks needs to be verified, which given the compact representation of the construct can lead to an exponential number of possible executions with respect to the size of an AND block.

Assuming that the tasks belonging to an AND block are truly independent, then the order in which they are executed should not influence whether the process model containing them is compliant or not. Therefore given this assumption, to prove the compliance of a process it is sufficient to consider the results of executing an AND block, independently on the order in which the tasks composing it are executed, hence it can be approximated as a single task, or a set of disjoint tasks in case different results are possible.

7.2.3 Monotonic Business Process Models

The third and last limitation concerns the semantics of the business process model. I restrict the traces resulting from such limited models, which I refer to as *monotonic business process models*, to have a monotonic evolution of the states composing them. In other words when some information becomes true and is included in the state of the process, then it cannot become false and retracted later on by the execution of other tasks.

Definition 48 (Monotonic Processes)**.** *A process* (P, ann) *is said to be monotonic if and only if:* $\forall \theta \in \Theta(P, \mathsf{ann}), \forall \sigma_i, \sigma_j \in \theta | \sigma_i \preceq \sigma_j, L_i \subseteq L_j$ *where* $L_i \in \sigma_i$ *and* $L_j \in \sigma_j$*.*

The definition describes these new class of business processes again in terms on their possible traces. The monotonic evolution of the states composing the traces of these monotonic processes is expressed by constraining a state to be a subset of each of its following ones, which means that a state contains at least the same information included in each of its predecessors.

A straightforward way to move from processes to monotonic processes without having to redefine the semantics of the models, hence being able to reuse the definitions and properties of the unrestricted processes, is to restrict the universe of the literals $\mathcal{L}$ to a restricted one where only positive literals can be expressed.

Definition 49 (Positive Universe $\mathcal{L}_P$)**.** *Given a* finite *set of atomic elements* E*, the positive universe* $\mathcal{L}_P$ *is* E*.*

A consequence of using a positive universe to describe the annotations, obligations' elements and process' states, is that the expressivity is reduced from three values to two. The values originally available for each element of the universe was either one of the following three:

- The element is included in the process state.
- The negation of the element is included in the process state.
- Neither the element nor its negation are included in the process state.

The consequence of restricting the expressivity of the obligations and the business process models to the positive universe, is that an element, described by a proposition, can either belong to a process' state or not belong to it. Assuming that in the restricted case when an element does not belong to a process' state, then it counts as *false*, like as defined using *negation as failure*, then the value lost by the restriction is the one representing the ignorance about the truth value of an element. Therefore a consequence of introducing such restriction is that the models move from a setting where the information about the world is partial to a setting where the information is complete, in other words given a proposition in the restricted setting, the system always knows whether that proposition is true or false.

The following corollary formally shows the connection between adopting a positive universe and obtaining a business process model behaving monotonically.

Corollary 3. *If the annotation of the tasks of a process model* (P, ann) *is restricted to a positive universe* $\mathcal{L}_P$*, then* (P, ann) *is a monotonic process model.*

Proof. The monotonic behaviour of the resulting process model, described in Definition 48, follows directly from Definitions 8, 10, 11 and 49. □

7.2.4 Resulting Differences

The restrictions adopted to obtain a tractable sub-class of the problem, in particular restricting the the problem to monotonic process models and limiting the elements of the obligations to be described by positive propositions, requires to redefine some of the obligations used in the regulatory framework.

The limitation adopted over the obligations' elements may not affect a system designer since the meaning associated to a proposition can also be negative, for instance one can associate to a positive proposition l the meaning "the light is off". This limitation along with limiting the process models to be monotonic reduces the semantics of maintenance obligations to standard obligations. Since maintenance obligations require that a property holds for a set of successive states, give the monotonicity of the evolution of the states, it become sufficient to verify whether the property holds in the first of the states. This verification procedure can be handled by an obligation of type standard, which is also a special case of achievement obligations. Thus by restricting the scope on monotonic processes, the semantics of maintenance obligations collapses in the semantics of achievement obligations.

While maintenance obligations can be removed from the framework since they become redundant, given that achievement obligations can now verify them, an additional type of

obligation can be introduced to capture the requirement of avoiding that a given proposition becomes true in the process' state. Therefore I introduce a new type of obligation whose goal is to verify whether a set of successive states do not contain a given proposition. The semantics of this new type of obligation, which I refer to as *prohibition*, is formalised in the following definition.

Definition 50 (Comply with Local Prohibition)**.** *Given a local prohibition* $\mathbf{O} = \mathcal{O}^p\langle c, l, d\rangle$ *and a trace* θ*,* θ *is compliant with* $\mathbf{O}$*, written* $\theta \vdash \mathbf{O}$*, if and only if:*

$\forall \sigma_i \in \theta$ *such that* $\sigma_i \models l$ *then* $\exists \sigma_h \in \theta$ *such that* $\sigma_h \models d$ *and* $\forall \sigma_j \in \theta$ *such that* $\sigma_i \preceq \sigma_j \preceq \sigma_h : \sigma_j \not\models c$

Notice that the requirement being verified by an obligation of type prohibition, would be equivalent to verify the maintenance of the absence of a given proposition. However, given the positive universe adopted to represent the elements composing the obligations, it is not possible to express such requirement using an obligation of type maintenance.

Another consequence of limiting the elements composing the obligations to positive propositions and the business process models to monotonic business processes, is that verifying whether a trace fulfils an obligation of type achievement and of type prohibition can be checked by considering exactly one of their activation periods. More precisely in the current sub-class of the problem being studied, when an activation period of an achievement obligation satisfies the fulfilling condition, then each following activation period also satisfies the condition. Differently, considering obligations of type prohibition, when an activation period does not satisfy the condition, then each following activation period also do not satisfy the condition.

Lemma 8 (Achievement Monotonicity)**.** *Given a trace* θ *and* $\mathbf{O}$ *an obligation of type achievement. Let* $I_1 \prec I_2$ *be true if the activation period* I_1 *of* $\mathbf{O}$ *precedes the activation period* I_2 *in* θ*. If* $\exists I_1 \in Interval(\mathbf{O}, \theta)$ *such that* I_1 *satisfies* $\mathbf{O}$*, then* $\forall I_2 \in Interval(\mathbf{O}, \theta)$ *such that* $I_1 \prec I_2$*,* I_2 *satisfies* $\mathbf{O}$*.*

Proof. Follows directly from Definitions 20 and 48. □

Lemma 9 (Prohibition Monotonicity)**.** *Given a trace* θ *and* $\mathbf{O}$ *an obligation of type prohibition. Let* $I_1 \prec I_2$ *be true if the activation period* I_1 *of* $\mathbf{O}$ *precedes the activation period* I_2 *in* θ*. If* $\exists I_1 \in Interval(\mathbf{O}, \theta)$ *such that* I_1 *does not satisfy* $\mathbf{O}$*, then* $\forall I_2 \in Interval(\mathbf{O}, \theta)$ *such that* $I_1 \prec I_2$*,* I_2 *does not satisfy* $\mathbf{O}$*.*

Proof. Follows directly from Definitions 50 and 48. □

7.3 Proving Regulatory Compliance in $C1^*_{1la}$

Before introducing the algorithm capable of proving regulatory compliance in a sub-class $C1^*_{1la}$ of the problem in linear time I describe the auxiliary functions and procedures used by it.

During the analysis of a business process model, to determine whether it is compliant with the regulatory framework, the algorithm uses labelings to identify relevant tasks in the model. More precisely the tasks are labeled dependending on their annotation and according to whether they can trigger the lifeline, the deadline or the fulfilment condition of the obligation being analysed. Which means that a label is associated to a proposition describing one of these elements. Two types of labelings are used by the algorithm: total and conditional. Given a label α, a task totally labeled in a business process model represents that when executed, the proposition associated to such label is true in the resulting process' state. A task conditionally labelled in a business process model represents that when executed, in some but not every case the proposition associated to such label is true in the resulting process' state. I use the notation α to represent the total labelling and α_c the conditional labelling.

7.3.1 Auxiliary Functions and Procedures

Before introducing the algorithm I introduce the two auxiliary functions used by it to prove regulatory compliance in a sub-class $C1^*_{1la}$ of the problem.

The functions I describe represent computationally low cost steps for which I provide the preconditions and the postconditions of applying them. An example of a function is labelling a subset of the tasks belonging to a process model according to a given condition. Notice that for this example verifying the condition must not be computationally costly.

Function: Conditional Labelling

The function *conditional labelling* converts some of the existing total labelings in a labeled process block into conditional labels. More precisely the function converts to a conditional label each of the total labels within a XOR block where at least a branch belonging to the same block does not contain another task totally labeled with the same label.

Notice that a labeled task may be contained in more than a XOR block if they are nested. In this case, whether the task should be conditionally labeled is checked for each of the XOR blocks containing it. In this case to avoid to recompute whether a task should be relabelled conditional it is sufficient to start from the most nested XOR block.

Function 1 (Conditional Labelling)**.** *Given a process block B labeled with α, β and γ, the function* conditional labelling*, written $CL(B)$, returns a process block $\mathcal{B}$ where the following holds:*

An element e is conditionally labeled in $\mathcal{B}$ if and only if:

the element e is totally labeled in B

and *the element is in an* xor *block*

and *exists a branch of the* xor *block not containing an element with the same label as e*

An element e is totally labeled in $\mathcal{B}$ if and only if:

the element e is totally labeled in B

and *the element is not conditionally labeled in $\mathcal{B}$*

A conditionally labeled task t in a XOR block X, where X is not nested in any additional XOR blocks, represents that there exists a serialisation of X that does contain at least a task with the same conditional label as t, note that this task is not necessarily t since there may exist other tasks with the same conditional label. However another task totally labelled within the same XOR block cannot exists. Moreover if t is conditionally labeled it also means that there exists a serialisation of X which does not contain a task with the same conditional label as t.

Example 29 (Conditional Labelling)**.** *I illustrate in the present example when a task belonging to a* XOR *block and totally labeled is translated to a conditionally labeled block. Considering first illustration* (*a*) *of Figure 7.1, showing a* XOR *block composed of four tasks, of which two of them are totally labeled, more precisely one for each of the two branches of the process block. Therefore in this first case, after applying the function* conditional labelling*, the two labelings remain unchanged, since each of the branches of the* XOR *block contains at least a totally labeled task.*

Considering now illustration (*b*) *of of Figure 7.1, showing a* XOR *block similar to the one previously considered but with an additional branch composed of a single unlabelled task. Differently, applying the function* conditional labelling *in this case would change the labels of the tasks t_1 and t_4 from their total label α to a conditional one α_c, because of the third branch of the* XOR *block, composed by the task t_5, which is not totally labelled.*

Algorithmically the function conditional labelling can be computed by analysing the XOR blocks individually and modifying the label to conditional when necessary. In the case of XOR blocks nested within other XOR blocks, the analysis is required to start from the inner most ones. By analysing nested XOR blocks in such order is ensured that the resulting labelled process block follows the postconditions specified by Function 1. Moreover the computational complexity of carrying on such analysis is linear in terms of the size of the model since each task of the model needs to be analysed at most once.

Function: Propagate Labelling

The function *propagate labelling* propagates the labels through the tasks in a process block B according to the transitions in the block. The propagation simulates the evolution of the state of the process. The propagated labels represent that the state of the process after executing the task *contains* the propositions annotated in the task from which the label originated in case the label is total, otherwise it *may* contain the annotated propositions in case the label is conditional. For instance if an element e of B is labeled α, then all

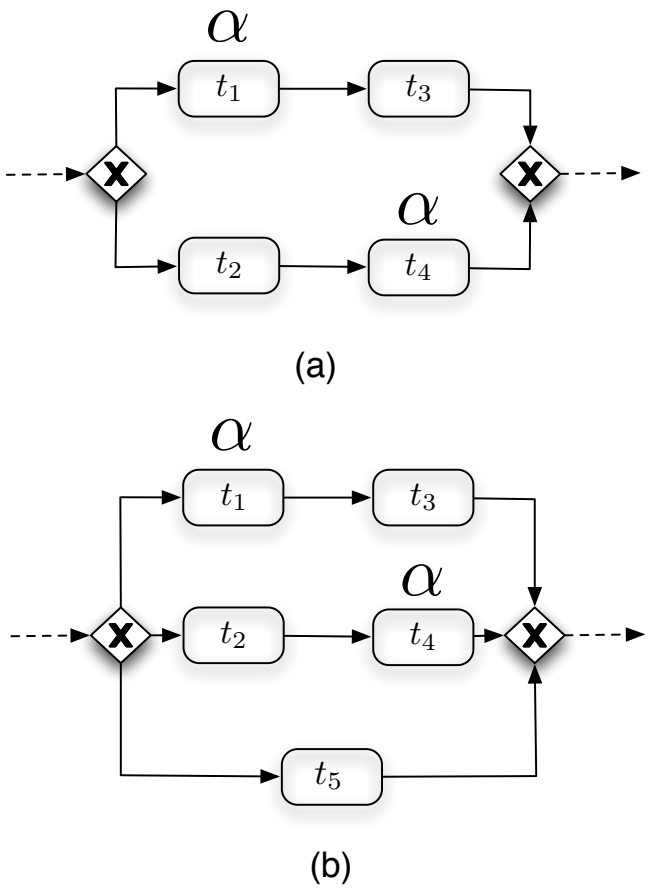

Figure 7.1: Conditional Labelling

elements reachable through the transitions from e become also labeled as α. This applies for both conditional and total labels, however in the case where an element would end up being both totally and conditionally labeled for the same label, then that element ends up being totally labeled for that particular label and not conditionally.

Function 2 (Propagate Labelling)**.** *Given a process block B labeled with α, β and γ the function* conditional labelling *, written $PL(B)$, returns a process block $\mathcal{B}$ where the following holds:*

An element e is totally labeled in $\mathcal{B}$ if and only if:

exists a path from an element totally labeled in B to e

An element e is conditionally labeled in $\mathcal{B}$ if and only if:

exists a path from an element conditionally labeled in B to e

and *does not exist a path from an element totally labeled in B to e*

In a process block B where its labels have been propagated, the labels of a task are an abstract representation of the state holding in a trace executing that task (as it is shown in Algorithm 7). Given that to each label it is associated a proposition, in the case a task t is totally labeled as α, then in every possible trace of B, the state holding after executing the task t always contains the literal associated to the label α. Differently, if a task t is conditionally labeled as α, then some but not all the traces of B contain the literal associated to α in the state after t is executed. More precisely, all the traces of B executing the task from which the conditional label α has been propagated contains in the state holding after the execution of t the literal associated to α.

Example 30 (Propagate Labelling)**.** *Figure 7.2 illustrates a business process model composed of six tasks of which one, task t_3, is conditionally labeled α_c and another, task t_5, is totally labeled α. Notice that according to the function conditional labelling, the task being conditionally labeled belongs to a* XOR *block which contains an edge without such label.*

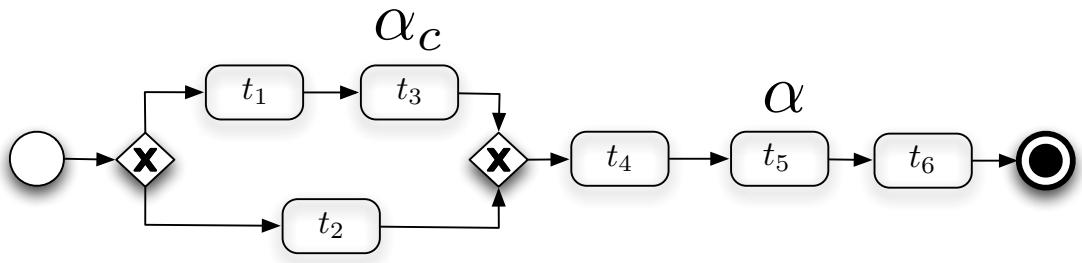

Figure 7.2: Starting Labelling

The result of applying the function propagate labelling *to the labelled process illustrated in Figure 7.2 is shown in Figure 7.3. From the result we can see that the conditional label propagated from the task t_3 to the task t_4. The conditional label would have also propagated to the tasks t_5 and t_6, however since these two tasks are also totally labelled, one originally and one from the propagation, then the conditional label is removed.*

A different result of applying the function propagate labelling *would have been obtained in case the task t_3 would have been conditionally labelled with a different label than task t_5, for instance β_c. In this case the propagation of the total label α would remain the same, while the propagation of the conditional label β_c would cover the following tasks in the results: t_3, t_4, t_5 and t_6. Which would have lead to having tasks t_5 and t_6 labelled by both α and β_c.*

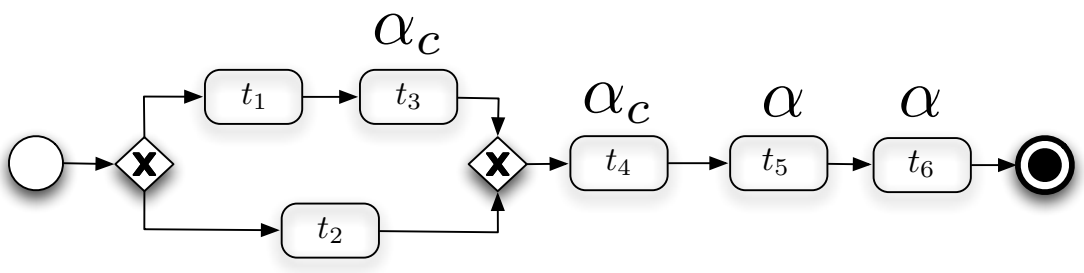

Figure 7.3: Propagated Labelling

To algorithmically compute the function propagate labelling, the tasks belonging to the business process model can be analysed starting from the one directly connected to the pseudo-task start. The other tasks belonging to the model are then analysed when each of the preceding tasks have been. When a task is being analysed, its label is then propagated to the tasks directly following it, in this case tasks separated by an xor coordinator are considered as directly connected. During the propagation total labels can rewrite conditional ones, while the opposite is not allowed. Computing the the function in such a way ensures that the postconditions listed in Function 2 are satisfied. The computational complexity of computing the function in such a way is linear in term of the size of the business process model.

7.3.2 Main Algorithm

I introduce now the algorithm capable of proving regulatory compliance of a sub-class $C1^*_{1la}$ of the problem in time linear with respect to the size of the input. The algorithm uses labels to identify the tasks that when executed produce a process state satisfying at least one of the elements of the obligation composing the regulatory framework. These tasks are identified according to their annotation, if an annotation contains the proposition which triggers one of the elements of the obligation, then the task is labeled with the label associated to that element.

The algorithm uses the function *conditional labelling* to distinguish between the labeled tasks which are the ones that belong to an XOR block which does not guarantee the proposition associated to the label to hold in the process state independently on the branch of the block being executed. After the algorithm uses the function *propagate labelling* to propagate between the tasks the labels according to the transitions in the process model. After the propagation, the algorithm proceeds to evaluate the labeled process model depending on the type of the given obligation, and returns whether the process model and to which extent it is compliant with the regulatory framework.

The idea behind the algorithm is that due to the forced monotonic behaviour of

the process state, when a proposition is included then it stays true for the the whole process. Because of this it is not required to check whether there are tasks that remove the propositions from the state. Therefore, since the order in which the tasks can be executed is governed by the transitions in the model, by propagating the labels according to the transitions allows to identify which proposition considered relevant is true when when a task is executed. Moreover in case the label propagated is conditional, then it means that not every execution of the task contains the relevant proposition, but only when one of the tasks from which the label has been propagated has been executed.

Therefore, depending on the type of obligation being analysed, the algorithm can check the labeled process model to determine whether there exists a labelling ensuring that none of the traces of the model fulfil the obligation or some of them. Given that only three compliance results are available, in case that neither of the two conditions are verified, then it means that the process is full compliant since each of its traces fulfil the obligation.

The conditions used by the algorithm to check the labelling of a task are expressed using propositions, where the truth value associated to a single label is true if it is associated to a task being checked and otherwise false.

Algorithm 7. *Given a* $C1^{*}_{1la}$ *problem composed of a monotonic process* (P, ann) *and a local atomic obligation* $\mathbf{O}$*,* $A_7((P, \mathsf{ann}), \mathbf{O})$ *returns whether* (P, ann) *is fully, partially or not compliant with* $\mathbf{O}$*.*

Algorithm A_7

Let $\mathbf{O} = \mathcal{O}^x\langle c, l, d\rangle$
Label α *the tasks in* P *where* $l \in \mathsf{ann}(t)$*;*
Label β *the tasks in* P *where* $d \in \mathsf{ann}(t)$ *and the pseudo-task* end*;*
Label γ *the tasks in* P *where* $c \in \mathsf{ann}(t)$*;*
$B = CL(B)$*;*
$B = PL(B)$*;*
if $x == a$ ***then***
  if $\exists$ *task* $\in B$ *labeled:* $\alpha \wedge \beta \wedge \neg(\gamma \vee \gamma_c)$ ***then***
    return $(P, \mathsf{ann}) \nvdash \mathbf{O}$
  else
    if $(\exists$ *task* $\in B$ *labeled:* $(\alpha \vee \alpha_c) \wedge (\beta \vee \beta_c) \wedge \neg(\gamma \vee \gamma_c)$ ***then***
      return $(P, \mathsf{ann}) \vdash^{P} \mathbf{O}$
    else
      return $(P, \mathsf{ann}) \vdash^{F} \mathbf{O}$
    end if
  end if
else
  if $\exists$ *task* $\in B$ *labeled:* $\alpha \wedge \neg(\beta \vee \beta_c) \wedge \gamma$ ***then***
    return $(P, \mathsf{ann}) \nvdash \mathbf{O}$
  else
    if $(\exists$ *task* $\in B$ *labeled:* $(\alpha \vee \alpha_c) \wedge \neg(\beta \vee \beta_c) \wedge (\gamma \vee \gamma_c)$ ***then***
      return $(P, \mathsf{ann}) \vdash^{P} \mathbf{O}$
    else

```
24:          return (P, ann) ⊢^F 𝔒
25:        end if
26:      end if
27:  end if
```

Complexity

The complexity of the functions used by the algorithm are both $\mathbf{O}(n)$ where n is the number of tasks contained in the model. The process of labelling the tasks requires to analyse the annotation of each of them, resulting again in a complexity of $\mathbf{O}(n)$. The same applies for the two evaluating steps which I can assume that requires to analyse each task again after the function *propagate labelling* propagated the labels in the model. The complexity of checking membership of a literal in an annotation is $\mathbf{O}(A)$, where A is the total size of the annotations. Therefore the general complexity of Algorithm 7 is $\mathbf{O}(n + A)$, which is linear with respect to the size of the input.

7.4 Tackling More Difficult Sub-Classes of the Problem

After having shown that a sub-class $C1^*_{1la}$ of the problem can be solved in time linear with respect to the size of the input, I provide in the last part of this chapter an analysis why the current restrictions are not sufficient to allow tractable solutions for more difficult sub-classes like $C2^*_{nla}$ or $C2^*_{1lc}$.

7.4.1 $C2^*_{nla}$ is Still Intractable

While considering a sub-class $C2^*_{nla}$ of the problem, where a set of local obligations compose the regulatory framework, the restrictions adopted are most likely to be insufficient to allow tractable solutions.

Intuitively, I show why a tractable solution does not seem to be possible reusing Algorithm 7. Since the difference between the sub-class $C1^*_{1la}$ and the more complex $C2^*_{nla}$ is that in the latter the regulatory framework is composed of multiple obligations, the straightforward approach would be to iteratively apply the algorithm used to verify a single obligation for each obligation composing the framework.

Blindly reiterating the algorithm would not be able to correctly determine whether a process is partially compliant since according to Definition 26, a trace needs to fulfil each of the obligation of set set in order to comply with the set of obligations. However in the current state the algorithm is not capable of determining whether the trace fulfilling different obligation is the same or is different for each obligation.

However trying to keep track of which set of traces were actually fulfilling the obligations composing the regulatory framework may lead in the worst case to have to consider each separate trace individually, which is intractable since the amount of traces is exponential with respect to the number of tasks. I show in Figure 7.4 an example where the structure

of a process model and its labelings requires to consider different cases while proving the compliance of a process.

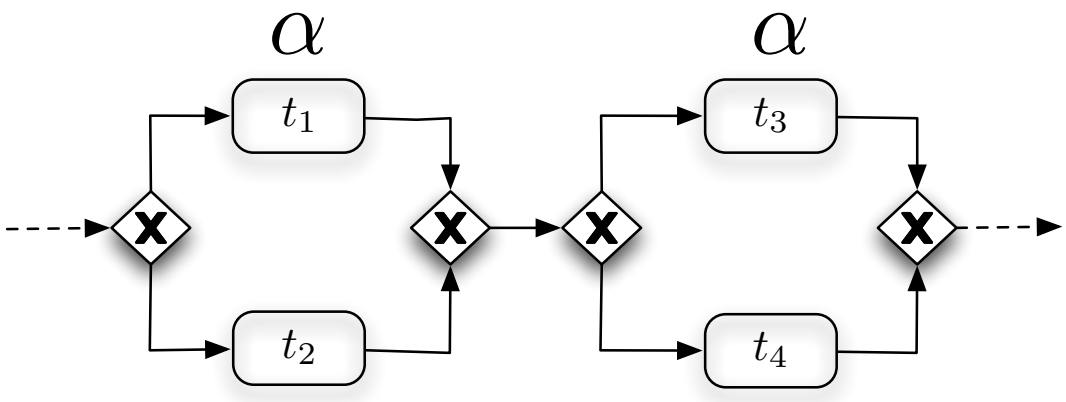

Figure 7.4: Branching SubStructure

The picture shows a substructure of a process model where an XOR block follows another and the tasks t_1 and t_3 have been labeled with the label α. Assuming now that we want to analyse the substructures ensuring at least a task labeled α to be executed, we have to considered two substructures: $\mathsf{SEQ}(t_1, \mathsf{XOR}(t_3, t_4))$ and $\mathsf{SEQ}(t_2, t_3)$, which exclude the execution composed by the tasks t_2 and t_4 which does not contain tasks labeled with α.

Assuming now that we want to continue our analysis to verify whether the two substructures identified contains traces fulfilling the other obligations composing the regulatory framework, we need to repeat the process for each of the obligation. However if we assume the case where instead of having atomic tasks in the branches we have more branching substructures and possibly more branches, it is straightforward to see that in the worst case the algorithm divides the model in sequences where each sequence corresponds to a single execution, which corresponds to a brute force analysis, which in turn is not tractable given the exponential number of possible executions derivable from such structure of the process.

7.4.2 $C2^*_{1lc}$ is Still Intractable

Finally, considering a sub-class $C2^*_{1lc}$ of the problem, where a single local obligation which allows compensations compose the regulatory framework, the restriction adopted in this chapter seems not to be sufficient to allow tractable solutions. The analysis why also this more complex sub-class does not allow tractable solution is analogous to the analysis made for the sub-class $C2^*_{nla}$ of the problem. The difference in this case is that the iterations are linked to the compensation chain associated and each iteration verifies one

of the obligation composing the chain. Nevertheless also in this case when considering the branching substructures, the analysis reduces itself in the worst case to a brute force analysis of every possible execution. A thorough analysis of this sub-class of the problem of proving regulatory compliance has been already provided by Colombo Tosatto et al. [22].

7.5 Summary

In this chapter I have shown that by simplifying a sub-class $C1^{-}_{1la}$ of the problem to another sub-class $C1^{*}_{1la}$, obtained by mostly restricting the expressivity of the business process model being checked, allows to prove regulatory compliance in time linear with respect to the size of the input.

The business process models are restricted to not contain AND blocks and to allow only traces where the process' state evolution is monotonic. Additionally the expressivity of the obligation is also slightly restricted by allowing the elements to be represented only by propositions, excluding in this way the possibility of using negative literals.

One may argue that the limitations imposed are too restrictive and a tractable sub-class of the problem, for instance solvable in time polynomial with respect to the size of the input, may be found using less harsh restrictions. However from the analysis of the two more complex sub-classes $C2^{*}_{nla}$ and $C2^{*}_{1lc}$, it results that such restrictions are not sufficient to allow tractable solutions as soon as another difficult characteristic is included in the regulatory framework.

Considering the second research subquestion concerned about finding a non-trivial tractable sub-class of the problem of proving regulatory compliance, the current chapter does not provide a definitive answer since the expressivity of the sub-class $C1^{*}_{1la}$ of the problem hardly makes it non-trivial. However the study performed in this chapter about the tractability of sub-class $C1^{*}_{1la}$ points out that tractable sub-classes of the problem may be found while not extremely trivialising one of the two elements composing the problem. However, whether tractable non-trivial sub-classes of the problem obtainable by simplifying both elements composing it exists and which are these sub-classes, remains still an open question.

Chapter 8

Conclusion

In this thesis I presented a formal analysis of the computational complexity of the problem of proving regulatory compliance of business process models. The problem of proving regulatory compliance consists of verifying whether a given business process model conforms with the given regulatory requirements. In this thesis I characterise the problem as composed by two elements: the business process model being studied and the regulatory framework describing the regulatory requirements through the use of obligations.

In the introduction I raised the main research question and two related research sub-questions. The present thesis answers the three questions as follows:

RQ What is the computational complexity of the general problem of proving the regulatory compliance of a business process model?

answer: The computational complexity of the general problem, corresponding to the sub-class $C3_{nlc}$, is **NP**-complete when proving partial compliance and co**NP**-complete when proving either full or non compliance.

The main research question has then been followed by two research subquestions, aimed at analysing in deeper details the problem of proving regulatory compliance. The two research subquestions are the following.

RSQ1 What is the computational complexity of the sub-classes of the problem of proving regulatory compliance?

answer: For the sub-classes: $C3_{nlc}$, $C2_{nla}$, $C2_{1lc}$ and $C1_{1la}$ verifying partial compliance is **NP**-complete and verifying either full or non compliance is co**NP**-complete. For the sub-class $C2^{-}_{nla}$ of the problem verifying partial compliance is **NP**-complete, verifying non compliance is co**NP**-complete and verifying full compliance is co**NP**. Whether verifying full compliance for the sub-class $C2^{-}_{nla}$ of the problem is co**NP**-complete is still an open question. For the sub-class $C0^{-}_{1ga}$, verifying either of full, partial and

Sub-Class	Partial Compliance	Full Compliance	Non Compliance
$C3_{nlc}$	**NP**-complete	co**NP**-complete	co**NP**-complete
$C2_{nla}$	**NP**-complete	co**NP**-complete	co**NP**-complete
$C2_{1lc}$	**NP**-complete	co**NP**-complete	co**NP**-complete
$C1_{1la}$	**NP**-complete	co**NP**-complete	co**NP**-complete
$C2^{-}_{nla}$	**NP**-complete	co**NP**	co**NP**-complete
$C0^{-}_{1ga}$	$\mathbf{O}(n^2)$	$\mathbf{O}(n^2)$	$\mathbf{O}(n^2)$
$C2^{*}_{nla}$	$\mathbf{O}(n + A)$	$\mathbf{O}(n + A)$	$\mathbf{O}(n + A)$
Verifying a Trace	$\mathbf{O}(n \times o \times T \times z)$	$\mathbf{O}(n \times o \times T \times z)$	$\mathbf{O}(n \times o \times T \times z)$

Table 8.1: Complexity Results

not compliance can be done in polynomial time. For the sub-class $C2^{*}_{nla}$, verifying either of full, partial and not compliance can be done in time linear with respect to the size of the problem.

RSQ2 Which are the sub-classes of the problem of proving regulatory compliance that are non-trivial and tractable?

answer: This research subquestion is still open. Even though some tractable sub-classes of the problem, such as $C0^{-}_{1ga}$ and $C2^{*}_{nla}$, have been identified, these sub-classes of the problem can hardly be considered non-trivial.

Table 8.1 summarises the complexity results for the sub-classes analysed in the present thesis. Mind that in the table n is the number of tasks contained in the model, A is the total size of the annotations, o is the number of obligations contained in the regulatory framework and z is the length of the longest compensation chain. T is the maximum time to check whether a state satisfies a formula.

In this final chapter of the thesis I recap the results obtained in the thesis. Additionally to the results found I also discuss the limitations of the approach adopted and about which part of the research questions has not yet been answered in the current thesis and which other question arose while tackling the current ones, which then can be used as a guidance for further research in the area of proving compliance.

8.1 Results

From the computational complexity analysis, it resulted that a set of the sub-classes of the problem of proving regulatory compliance, including the general problem, belong to the same complexity class, as it is illustrated in Figure 6.3 in Chapter 6. More precisely the results show that proving whether a business process model is partially compliant, meaning that exists an execution of the business process model which is compliant with the

regulations, is a **NP**-complete problem. Differently, proving either that a business process model is fully compliant or it is not compliant, meaning that all executions of the business process model or none of them are compliant with the regulations, are both co**NP**-complete problems.

The computational complexity results obtained for the subset of sub-classes of the problem identified in this thesis allow to answer the main research question **RQ**. The main research question aims at identifying the computational complexity of the general problem of proving regulatory compliance. The general problem, corresponding to the sub-class $C3_{nlc}$ where a business process model is checked against a set of conditional obligations where compensations for eventual violations are allowed, is among the sub-classes for which proving partial compliance is **NP**-complete and proving either full or not compliance is co**NP**-complete.

However, if we move the focus on the first research subquestion **RSQ1**, it has been answered only partially by the present thesis. Some of the sub-classes of the problem have been shown to have the same computational complexity of the general problem. While other more restricted sub-classes, like the one discussed in Chapter 5, where the regulatory framework is restricted to a set of local obligations and the obligations themselves are restricted to be described using propositional literals and not formulae, has been shown to have a similar complexity to the general problem, however in this particular case, the restriction on the elements composing the obligations allowed only to show that proving full compliance of such sub-class of the problem is co**NP**. In this case, whether this proving full compliance of this particular sub-class of the problem is co**NP**-complete, is still an open question.

An additional result provided in this thesis and concerning **RSQ1** has been shown in Chapter 3. In that chapter it has been shown that the sub-class of the problem, where the expressivity of the regulatory framework is the most limited given the features identified, can be solved in time polynomial with respect to the size of the input. This particular sub-class of the problem is among one of the tractable ones, which the second research subquestion **RSQ2** aims at finding. However, given the extreme limitations applied on the regulatory framework in this particular case, this sub-class of the problem does not classify as non-trivial.

The second research subquestion is yet to be properly answered. The present thesis pointed out three tractable sub-classes of the problem. The first one, introduced in Chapter 3, and obtained by verifying a business process model against a very limited regulatory framework. The second one, implicitly introduced in Chapter 6, and obtained by verifying a very simple business process model against a regulatory framework. This sub-class has not been introduced explicitly, however given that a trace can be considered as a very simple business process model, whose executions correspond to exactly that trace, then the computational complexity analysis of such sub-class of the problem is provided by Algorithm 6, which complexity is in time polynomial with respect to the size of the input. The third and last tractable sub-class of the problem investigated in the present thesis, discussed

in Chapter 7, is obtained by limiting the expressivity of both regulatory framework and business process model, however applying lighter limitations than the ones applied on the individual elements by the two other tractable sub-classes.

Even though the third tractable sub-class of the problem studied in the thesis can be solved efficiently, in fact in time linear with respect to the size of the input, the restriction applied on both elements determining the sub-class would still not allow to classify it as non-trivial. However I argue that the result, shown by studying the computational complexity of such sub-class of the problem, represents a first step towards exploring the search space where we are trying to maximise three parameters: *expressivity of the regulatory framework*, *expressivity of the business process model* and *tractability*, as graphically illustrated in Figure 8.1

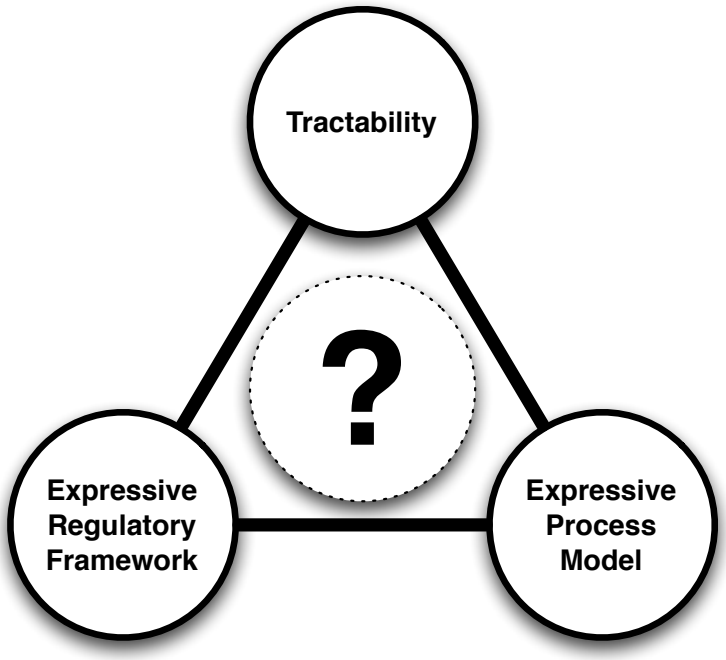

Figure 8.1: Non-Trivial Tractable Sub-Classes

8.2 Limitations of the Approach

The approach adopted in the present thesis to tackle the problem of proving regulatory compliance is not free of limitations. Thao Ly et al. [54] introduce in their work ten desirable functionalities that a compliance monitoring system should provide. As already

discussed in Chapter 2, the approach adopted in the present thesis covers only a fraction of the functionalities discussed by Thao Ly et al. More precisely the approach adopted fully covers two of the functionalities and partially cover three of them. This is indeed a limitation of the approach adopted to tackle the problem of proving compliance, since an approach covering a higher number of the functionalities identified by Thao Ly et al. would be a more effective approach, in other words it would be able to tackle a wider range of problems.

An additional limitation of the approach adopted in this thesis concerns the computational complexity results more than the actual approach. Disregarding for the moment the limitations of the approach adopted, I focus now on applying such methodology for proving regulatory compliance in practical scenarios. Even though the results of the present thesis confirm that the problem is in general hard and intractable, the study of the computational complexity of the problem performed in the present thesis is purely theoretical and solving practical cases may not be as complex as suggested by the results.

The computational complexity of solving practical cases is most likely to be difficult, however it may not be as intractable as the current analysis points out. The computational complexity hardness of the problem of proving regulatory compliance derives from the necessity in the worst case scenarios to have to analyse independently different cases. The branching generating these different cases can iteratively lead to an exponential number of cases to be analysed, which in turn leads to the intractability of the problem. However, the amount of branchings required to analyse a practical case is generally bounded to a low number, leading to a low order exponential number of cases to analyse, which is not intractable. One of these parameters, that may require to consider different cases during the analysis, but is in general bounded to a low number in practical cases, consists of compensations. Compensations define what becomes necessary to fulfil in case an obligation is being violated. Compensations, being themselves obligations, can also be violated and eventually compensated, leading to obligations being composed of sequences of compensations. The different cases needed to be analysed can derive from the different alternatives that would have violated an obligation. This branching can then occur for each compensation in the sequence. In practical scenarios the amount of compensations composing such sequence is generally limited to a low number, around three or four usually. The designers of the regulations often think about what becomes necessary when a primary obligation is violated, sometimes they provide compensatory measures for the compensation itself, but seldom this process goes beyond the fourth step. Therefore we can see that the computational complexity deriving from the sequences of obligations is in practice generally bounded. Other sources of complexity involve the structure of the business process model. One of those, illustrated in Figure 7.4 in Chapter 7 can again lead to an exponential amount of cases that need to be studied independently. Again also in this case, practical cases may not necessarily contain such structures that can lead to the intractability of the problem.

8.3 Further research

Following from the previous section, where I discussed the limitations of the approach adopted in this thesis to prove regulatory compliance of business process models, it results that the approach can be improved to tackle a wider range of problems of proving regulatory compliance.

8.3.1 Covering Additional Functionalities

The first way discussed in this section to improve the approach introduced by this thesis is to extend it in order to cover each of the ten desirable functionalities pointed out by Thao Ly et al. [54]. The approach proposed currently fully covers two of the functionalities and partially cover three of them. The two functionalities fully covered are the temporal constraints concerning the ordering executions of the activities and the possibility of handling violations of the regulations. The three functionalities partially covered by this thesis' approach are the data constraints concerning the process' state, the proactive detection of compliance violations and the detection of different levels of compliance. Some preliminary investigations about how to extend the current approach to cover additional functionalities have been already done. For instance, concerning the inclusion of resource constraints when proving regulatory compliance of a business process model, Colombo Tosatto et al. [73] propose a richer variant of business process models where resource consumption and availability can be monitored. Extending the current approach with the one proposed by Colombo Tosatto et al. would allow to verify constraints concerning the resource consumption while executing a business process.

In addition to extending the expressivity of the approach by including additional functionalities, a further computational complexity analysis would be in order to study whether by including additional functionalities, the computational complexity of the resulting problem of proving regulatory compliance increases or remains the same. Moreover analysing the computational complexity of introducing each of the functionalities discussed by Thao Ly et al., allows to identify which of them can be freely included in the framework without having to worry about an increase of the computational complexity.

8.3.2 Study Real Practical Cases

The second way discussed concerns analysing real world scenarios to identify whether there exists a computational complexity gap between theoretical and practical problems. As pointed out in the previous section, about the limitations of the approach adopted in the present thesis, the worst case scenarios where the complexity of proving compliance explodes may not happen while analysing real cases of proving regulatory compliance. Even though the theoretical computational complexity analysis may not provide the exact computational complexity of solving real problems of proving regulatory compliance, the theoretical analysis of the complexity provided an upper bound for the computational complexity

of the problem of proving regulatory compliance. However studying the computational complexity of practical problems would help also to identify which of the features of the problem would not contribute massively to the complexity of solving it. Additionally, by identifying this distance between the practical and theoretical instances of the problems of proving regulatory compliance, it may help to identify a sub-class of the problem simple enough to be tractable, but expressive enough to be able to reason and solve a relevant part of the practical problems analysed.

8.3.3 Extending the Regulatory Framework

The approach adopted in this thesis use an abstraction when dealing with the obligations describing the compliance requirements that a business process model needs to fulfil. In *normative reasoning*, the field of computer science specialised in reasoning about what is necessary given the laws and regulations in a given context, the obligations are just a fragment of the whole problem. Obligations are determined by regulatory norms in normative reasoning, however there are other type of norms that interact in the system and define what is allowed and necessary. As discussed in Colombo Tosatto et al. [20], different types of norms, such as constitutive norms and permissive norms, can be used to determine what is obligatory and permitted in a given context. Such approach can be integrated in the current approach used in this thesis in order to capture the details behind reasoning about the requirements that a business process model needs to fulfil, as well as what it is allowed to do.

8.3.4 Adopting a Visual Approach

As already adopted in some of the existing approaches, such as Awad et al. [10], graphically visualising the compliance requirements that needs to be fulfilled by a business process model allows a more user-friendly interface that users not too involved in logic formalisms can easily use. In the approach adopted in the present thesis the representation of the obligations governing the processes does not include a graphical part. Such limitation can be addressed by adopting a *graph-like* representation such the one introduced by Colombo Tosatto et al. [21], to represent and reason about the normative concepts such as the different type of norms that can compose the regulatory framework. Extending the current approach in this direction would lead to a system where both elements can be represented graphically, the business process models using BPMN 2.0 and the regulatory framework using a graph-like representation similar to the one introduced by Colombo Tosatto et al. A result of allowing to abstract from the technical details through a graphical representation of the whole problem would create a more *user-friendly* framework that can be used without having to deal with the logical machinery behind.

8.3.5 Extending to Multiple Business Process Models

The approach proposed in the present thesis to study the problem of proving regulatory compliance focused on a single business process model. However in practice it is not unusual that processes interact with other ones while pursuing their goal. An instance of this scenario can be found when some functionalities are outsourced to other providers, such as in the case of *cloud computing*, as already pointed out by Accorsi et al. [3].

When dealing with multiple business process models interacting with each other it becomes necessary to consider additional aspects while proving the regulatory compliance of these process models. In some cases regulations may constrain the interaction between different process models, such as for instance constraining the use of information originating from another process. In this case it is not anymore sufficient to analyse the different models individually but they must be considered as a whole.

Again, studying practical cases of proving regulatory compliance would be useful in order to identify which interactions among the business process models are required to be monitored and verified whether they are compliant with the regulatory framework or not.

8.4 Concluding Remarks

The analysis of the computational complexity of the problem of proving regulatory compliance of business process models conducted in the present thesis, outlines that in general the problem is intractable, since it belongs to either the class of **NP**-complete problems or to the class of co**NP**-complete problems depending on the type of compliance being evaluated. The general problem has been shown to be indeed theoretically difficult. In particular the results provided in this thesis identify the upper bound computational complexity of every sub-class of the problem of proving regulatory compliance adopted in the present thesis. However different approaches to tackle the problem of proving regulatory compliance may belong to different complexity classes. Therefore, to analyse the suitability of the approach adopted to solve compliance problems, as future work has been pointed out that a study of real practical scenarios should be done. This analysis of practical cases would also allow to identify whether the worst case scenarios happening in these cases are the same as the theoretical ones that determined the computational complexity resulting from the analysis in the present thesis.

Finding a theoretically tractable subset of the sub-classes of the problem is also a challenging task. However at the moment this remains an open question. Identifying such subset, especially a subset containing sub-classes capable of handling non-trivial instances of the problem, would allow to efficiently deal with some of the instances of the problem of proving regulatory compliance. In this thesis I identified three theoretically tractable sub-classes of the problem, two of them consisting of the borderline cases where the expressivity of one of the elements composing the problem have been severely limited. However the third sub-class, even if not non-trivial has been obtained by weakening the expressivity of

both regulatory framework and the business process model, which can represent a first step towards this theoretically tractable non-trivial sub-class.

However whether among the tractable sub-classes of the problem there exists one or more capable of dealing with practical instances of the problem of proving regulatory compliance is still an open question. Nevertheless, I argue that an answer for such question may be found by thoroughly analysing the computational complexity of the sub-classes of the problem, and verifying whether the tractable ones are capable of dealing with existing practical problems. Moreover, the answer to the previous question may then be key in driving further research in the area of proving regulatory compliance, more precisely in deciding whether the focus of finding solutions to the problem should be on exact solutions or on approximate ones, such as solutions using *heuristics*.

Appendix A

The Abstract Framework

A.1 Procedure Task Removal

Procedure 1 (Task Removal)**.** *Given a process block B and a set of tasks $\mathcal{T} = \{t | t \in V(B)$ and t is a task$\}$,* task removal $R(B, \mathcal{T})$ *returns either a new process block B' or $\perp$ as follows:*

1: ***if*** B *is a task* t ***then***
2: ***if*** $t \in \mathcal{T}$ ***then***
3: ***return*** $\perp$
4: ***else***
5: ***return*** B
6: ***end if***
7: ***end if***
8: ***if*** $B = \mathsf{SEQ}(B_1, \ldots, B_k)$ ***then***
9: ***if*** $\exists B_i, 1 \leq i \leq k$ *such that* $R(B_i, \mathcal{T}) = \perp$ ***then***
10: ***return*** $\perp$
11: ***else***
12: ***return*** $\mathsf{SEQ}(R(B_1, \mathcal{T}), \ldots, R(B_k, \mathcal{T}))$
13: ***end if***
14: ***end if***
15: ***if*** $B = \mathsf{XOR}(B_1, \ldots, B_k)$ ***then***
16: ***if*** $\forall B_i, 1 \leq i \leq k$, $R(B_i, \mathcal{T}) = \perp$ ***then***
17: ***return*** $\perp$
18: ***else***
19: ***if*** $\exists! B_i, 1 \leq i \leq k$ *such that* $R(B_i, \mathcal{T}) \neq \perp$ ***then***
20: ***return*** $R(B_i, \mathcal{T})$
21: ***else***
22: ***return*** $\mathsf{XOR}(R(B_{m_1}, \mathcal{T}), \ldots, R(B_{m_n}, \mathcal{T}))$

where $B_{m_j} \in \{B_1, \ldots, B_k\}$ *and* $R(B_{m_j}, \mathcal{T}) \neq \perp$
23: ***end if***
24: ***end if***
25: ***end if***
26: ***if*** $B = \mathsf{AND}(B_1, \ldots, B_k)$ ***then***
27: ***if*** $\exists B_i, 1 \leq i \leq k$ *such that* $R(B_i, \mathcal{T}) = \perp$ ***then***
28: ***return*** $\perp$
29: ***else***
30: ***return*** $\mathsf{AND}(R(B_1, \mathcal{T}), \ldots, R(B_k, \mathcal{T}))$
31: ***end if***
32: ***end if***

Lemma 1 (Task Removal)**.** *Given a process block* B *and a set* $\mathcal{T}$ *of tasks* $\{t | t \in V(B)$ *and* t *is a task*$\}$*:*

1. *If* $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ *such that* $t \in \mathcal{T}$*, then* $R(B, \mathcal{T}) = \perp$
2. *If* $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ *such that* $t \in \mathcal{T}$*, then* $R(B, T) = B'$ *and* $\forall \epsilon' \in \Sigma(B'), \neg\exists t' \in \epsilon'$ *such that* $t' \in \mathcal{T}$
3. *If* $\exists \epsilon \in \Sigma(E), \neg\exists t \in \epsilon$ *such that* $t \in T$*, then* $\forall \epsilon \in \Sigma(E), \neg\exists t \in \epsilon, \exists \epsilon' \in \Sigma(R(E, T))$ *such that* $\epsilon \equiv \epsilon'$

The three statements of Lemma 1 are proven separately.

Proof. I first prove the following: If $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ such that $t \in \mathcal{T}$, then $R(B, \mathcal{T}) = \perp$
I use a proof by induction.
Base Case: B is a single task t.
I prove this case directly:

1. From the assumption that $B = x$ and Definition 4) it follows that $\Sigma(B) = \{(x)\}$, where (x) is the only possible serialisation.
2. From the premise: $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ such that $t \in \mathcal{T}$, and from 1 it follows that $x \in \mathcal{T}$.
3. From 1 and 2 it follows that lines 1 and 2 of Procedure 1 are satisfied, hence $R(B, \mathcal{T})$ returns $\perp$ in line 3.

Inductive Cases:

1. $B = \mathsf{SEQ}(B_1, \ldots, B_k)$

   I prove this case by contradiction:

   (a) Suppose that $R(B, \mathcal{T}) \neq \perp$

(b) From the assumption $B = \mathsf{SEQ}(B_1, \ldots, B_k)$.

(c) From (a) and (b) it follows that line 10 of Procedure 1, returning $\perp$, is not executed.

(d) From (a) and (c) it follows that in Procedure 1, since the condition at line 8 is true, then the condition at line 9 must be false.

(e) From (d) it follows that $\neg\exists B_i, 1 \leq i \leq k$ such that $R(B_i, \mathcal{T}) = \perp$.

(f) From (e), Definition 4 and the induction hypothesis it follows that $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(g) (f) contradicts the premise $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(h) Therefore, if $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ such that $t \in \mathcal{T}$, then $R(B, \mathcal{T}) = \perp$.

2. $B = \mathsf{XOR}(B_1, \ldots, B_k)$

I prove this case by contradiction:

(a) Suppose that $R(B, \mathcal{T}) \neq \perp$

(b) From the assumption $B = \mathsf{XOR}(B_1, \ldots, B_k)$.

(c) From (a) and (b) it follows that line 17 of Procedure 1, returning $\perp$, is not executed.

(d) From (a) and (c) it follows that in Procedure 1, since the condition at line 15 is true, then the condition at line 16 must be false.

(e) From (d) it follows that $\exists B_i, 1 \leq i \leq k$ such that $R(B_i, \mathcal{T}) \neq \perp$.

(f) From (e), Definition 4 and the induction hypothesis it follows that $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(g) (f) contradicts the premise $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(h) Therefore, if $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ such that $t \in \mathcal{T}$, then $R(B, \mathcal{T}) = \perp$.

3. $B = \mathsf{AND}(B_1, \ldots, B_k)$

I prove this case by contradiction:

(a) Suppose that $R(B, \mathcal{T}) \neq \perp$

(b) From the assumption $B = \mathsf{AND}(B_1, \ldots, B_k)$.

(c) From (a) and (b) it follows that line 28 of Procedure 1, returning $\perp$, is not executed.

(d) From (a) and (c) it follows that in Procedure 1, since the condition at line 26 is true, then the condition at line 27 must be false.

(e) From (d) it follows that $\neg\exists B_i, 1 \leq i \leq k$ such that $R(B_i, \mathcal{T}) = \perp$.

(f) From (e), Definition 4 and the induction hypothesis it follows that $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(g) (f) contradicts the premise $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(h) Therefore, if $\forall \epsilon \in \Sigma(B), \exists t \in \epsilon$ such that $t \in \mathcal{T}$, then $R(B, \mathcal{T}) = \bot$.

In each of the possible inductive cases I have shown that: If $\forall \epsilon \in \Sigma(B) \exists t \in \epsilon$ such that $t \in \mathcal{T}$, then $R(B, \mathcal{T}) = \bot$. Thus the statement $\forall \epsilon \in \Sigma(B) \exists t \in \epsilon$ such that $t \in \mathcal{T}$, then $R(B, \mathcal{T}) = \bot$ holds for each B where B is a process block.

□

Proof. I now prove the second statement of Lemma 1: If $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$, then $R(B, T) = B'$ and $\forall \epsilon' \in \Sigma(B'), \neg\exists t' \in \epsilon'$ such that $t' \in \mathcal{T}$

I use a proof by induction.

Base Case: B is a single task x.

1. From the assumption that $B = x$ and Definition 4) it follows that $\Sigma(B) = \{(x)\}$, where (x) is the only possible serialisation.

2. From the premise: $\exists \epsilon \in \Sigma(B), \exists x \in \epsilon$ such that $x \notin \mathcal{T}$, and from 1 it follows that $x \notin \mathcal{T}$.

3. From 1 and 2 it follows that lines 1 of Procedure 1 is satisfied but line 2 is not, hence $R(B, \mathcal{T})$ returns B in line 5.

A process block containing a single task t contains a single serialisation ϵ which is composed by t itself (Definition 4). From the hypothesis we know that $t \notin \mathcal{T}$. Thus it follows that the condition at line 2 of Procedure 1 is not satisfied and the procedure returns B at line 5, containing all the serialisations of B not containing a task appearing in $\mathcal{T}$, which in this case are the same as the original process block.

Inductive Cases:

1. $B = \mathsf{SEQ}(B_1, \ldots, B_k)$

   I prove this case directly:

   (a) From the assumption it follows that $B = \mathsf{SEQ}(B_1, \ldots, B_k)$.

   (b) From the premise it follows that $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$.

   (c) From (a), (b) and Definition 4 it follows that $\epsilon \in \Sigma(B_1) +_{\mathbb{P}} \ldots +_{\mathbb{P}} \Sigma(B_k)$.

   (d) From (b) and (c) it follows that for $1 \leq i \leq k, \exists \epsilon \in \Sigma(B_i), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$.

   (e) From (d) and Induction Hypothesis it follows that for $1 \leq i \leq k, R(B_i, \mathcal{T}) \neq \bot$.

(f) From (a) it follows that line 8 of the procedure is satisfied and from (e) it follows that line 9 of the procedure is not satisfied, hence the procedure returns $\mathsf{SEQ}(R(B_1, \mathcal{T}), \ldots, R(B_k, \mathcal{T}))$.

(g) From (f) and Induction Hypothesis if follows that each serialisation of $\mathsf{SEQ}(R(B_1, \mathcal{T}), \ldots, R(B_k, \mathcal{T}))$ does not contain tasks belonging to $\mathcal{T}$.

2. $B = \mathsf{XOR}(B_1, \ldots, B_k)$

I prove this case directly:

(a) From the assumption it follows that $B = \mathsf{XOR}(B_1, \ldots, B_k)$.

(b) From the premise it follows that $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(c) From (a), (b) and Definition 4 it follows that $\epsilon \in \Sigma(B_1) \cup \ldots \cup \Sigma(B_k)$.

(d) From (b) and (c) it follows that for $\exists B_i \in B, \exists \epsilon \in \Sigma(B_i), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(e) From (d) and Induction Hypothesis it follows that for $\exists B_i \in B$ such that $R(B_i, \mathcal{T}) \neq \bot$.

(f) From (a) it follows that line 15 of the procedure is satisfied and from (e) it follows that line 16 of the procedure is not satisfied.

(g) From (f) it follows that the procedure returns either $R(B_i, \mathcal{T})$ or $\mathsf{XOR}(R(B_{m_1}, \mathcal{T}), \ldots, R(B_{m_n}, \mathcal{T}))$.

(h) From (g) and Induction Hypothesis it follows that each serialisation of $R(B_i, \mathcal{T})$ or $\mathsf{XOR}(R(B_{m_1}, \mathcal{T}), \ldots, R(B_{m_n}, \mathcal{T}))$ does not contain tasks belonging to $\mathcal{T}$.

3. $B = \mathsf{AND}(B_1, \ldots, B_k)$

I prove this case directly:

(a) From the assumption it follows that $B = \mathsf{AND}(B_1, \ldots, B_k)$.

(b) From the premise it follows that $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(c) From (a), (b) and Definition 4 it follows that $\epsilon \in \Sigma(B_1) \cup_{\mathbb{P}} \ldots \cup_{\mathbb{P}} \Sigma(B_k)$.

(d) From (b) and (c) it follows that for $1 \leq i \leq k, \exists \epsilon \in \Sigma(B_i), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$.

(e) From (d) and Induction Hypothesis it follows that for $1 \leq i \leq k, R(B_i, \mathcal{T}) \neq \bot$.

(f) From (a) it follows that line 26 of the procedure is satisfied and from (e) it follows that line 27 of the procedure is not satisfied, hence the procedure returns $\mathsf{AND}(R(B_1, \mathcal{T}), \ldots, R(B_k, \mathcal{T}))$.

(g) From (f) and Induction Hypothesis if follows that each serialisation of $\mathsf{AND}(R(B_1, \mathcal{T}), \ldots, R(B_k, \mathcal{T}))$ does not contain tasks belonging to $\mathcal{T}$.

For each of the inductive cases I have shown that: If $\exists \epsilon \in \Sigma(B), \neg\exists t \in \epsilon$ such that $t \in \mathcal{T}$, then $R(B,T) = B'$ and $\forall \epsilon' \in \Sigma(B'), \neg\exists t' \in \epsilon'$ such that $t' \in \mathcal{T}$.

□

Proof. I now prove the third an last statement of Lemma 1: If $\exists \epsilon \in \Sigma(E), \neg\exists t \in \epsilon$ such that $t \in T$, then $\forall \epsilon \in \Sigma(E), \neg\exists t \in \epsilon, \exists \epsilon' \in \Sigma(R(E,T))$ such that $\epsilon \equiv \epsilon'$

I use a proof by contradiction.

1. Assume that $\exists \epsilon \in \Sigma(E), \neg\exists t \in \epsilon$ such that $t \in T$ and $\epsilon \notin \Sigma(R(E,T))$.
2. From 1), it follows that ϵ is removed by the procedure.
3. From 1), it follows that $R(E,T) \neq \bot$.
4. From 2) and 3), it follows that either line 20 or line 22 are executed.
5. From 4), it follows that the sub blocks of E removed are the ones where applying the procedure returns $\bot$
6. From 5) and the first statement of Lemma 1, it follows that the procedure returns $\bot$ if every ϵ in the block contains a task in T.
7. From 5), 6) and Definition 4, it follows that each execution eliminated by the procedure contains a task in T.
8. 7) and 2) are in contradiction.

Therefore it holds that: If $\exists \epsilon \in \Sigma(E), \neg\exists t \in \epsilon$ such that $t \in T$, then $\forall \epsilon \in \Sigma(E), \neg\exists t \in \epsilon, \exists \epsilon' \in \Sigma(R(E,T))$ such that $\epsilon \equiv \epsilon'$. □

A.2 Global Achievement Algorithm

Algorithm 1. *Given an annotated process* (P, ann) *and a global achievement obligation* $O^a(l)$*, this algorithm returns whether* (P, ann) *is compliant with* $O^a(l)$*.*

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

10: ***end if***

Correctness Proof of Algorithm 1. Given an annotated process (P, ann), where $P = \mathsf{start}\ B\ \mathsf{end}$, and a global achievement obligation $O^a(l)$, I prove the correctness of Algorithm 1 by showing both soundness and completeness.

Soundness Following from Definition 16 a process can be either fully compliant, or partially compliant, or not compliant with an atomic global achievement obligation. I prove the soundness of Algorithm 1 by cases considering the three possible cases listed above, and showing that the process given in input belongs to such compliance class.

1. Algorithm 1 returns $(P, \mathsf{ann}) \nvdash O^a(l)$
   (a) From the assumption, it follows that the condition at line 2 of the algorithm is true.
   (b) From (a),Definition 11 and Definition 10, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \neg\exists\sigma \in \theta$ such that $\sigma \models l$.
   (c) From (b) and Definition 14, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \nvdash O^a(l)$.
   (d) From (c) and Definition 16, it follows that (P, ann) is not compliant with $O^a(l)$.
2. Algorithm 1 returns $(P, \mathsf{ann}) \vdash^{\mathrm{F}} O^a(l)$
   (a) From the assumption, it follows that the condition at line 2 of the algorithm is false and the condition at line 5 is true.
   (b) From (a) and Lemma 1, it follows that $\forall\epsilon \in \Sigma(P), \exists t \in \epsilon$ such that $l \in \mathsf{ann}(t)$.
   (c) From (b), Definition 11 and Definition 10, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \exists\sigma \in \theta$ such that $\sigma \models l$.
   (d) From (c) and Definition 14, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \vdash O^a(l)$.
   (e) From (d) and Definition 16, it follows that (P, ann) is fully compliant with $O^a(l)$.
3. Algorithm 1 returns $(P, \mathsf{ann}) \vdash^{\mathrm{P}} O^a(l)$
   (a) From the assumption, it follows that the condition at line 2 of the algorithm is false and the condition at line 5 is false.
   (b) From (a) and Lemma 1, it follows that $\neg\forall\epsilon \in \Sigma(P), \exists t \in \epsilon$ such that $l \in \mathsf{ann}(t)$.
   (c) From (a),Definition 11 and Definition 10, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \exists\sigma \in \theta$ such that $\sigma \models l$.
   (d) From (b) and Definition 14, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \vdash O^a(l)$.
   (e) From (c) and Definition 14, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \nvdash O^a(l)$.

(f) From (d), (e) and Definition 16, it follows that (P, ann) is partially compliant with $O^a(l)$ but is not fully compliant with $O^a(l)$.

I have shown the soundness of Algorithm 1 by showing that in each of the three possible cases, whether the algorithm returns fully, partially or not compliant, then the process model given as input is correctly classified.

Completeness I prove by cases the completeness of Algorithm 1, by showing that for each of the three compliance classes to which a process can belong, the result of the algorithm is indeed corresponding to such class.

1. (P, ann) is fully compliant with $O^a(l)$
   (a) From the assumption and from Definition 16, it follows that $\forall \theta \in \Theta(P, \mathsf{ann})$, $\theta \vdash O^a(l)$.
   (b) From (a) and Definition 14, it follows that $\forall \theta, \exists \sigma \in \theta$ such that $\sigma \models l$.
   (c) From (b), Definition 11 and Definition 10, it follows that $\exists t \in P$ such that $l \in \mathsf{ann}(t)$.
   (d) From (c), it follows that the condition at line 2 of the algorithm is false and the else case is executed.
   (e) From (b) and Lemma 1, it follows that $R(B, \{t \mid \text{t is a task in } B \text{ and } l \in \mathsf{ann}(t)\}) = \bot$.
   (f) From (e) and (d), it follows that the condition at line 5 of the algorithm is reached and is true.
   (g) From (f), it follows that the algorithm returns $(P, \mathsf{ann}) \vdash^{\mathrm{F}} O^a(l)$ at line 6.
2. (P, ann) is partially compliant with $O^a(l)$ and is not fully compliant with $O^a(l)$
   (a) From the assumption and from Definition 16, it follows that $\exists \theta \in \Theta(P, \mathsf{ann})$, $\theta \vdash O^a(l)$.
   (b) From the assumption and from Definition 16, it follows that $\neg\forall \theta \in \Theta(P, \mathsf{ann})$, $\theta \vdash O^a(l)$.
   (c) From (a) and Definition 14, it follows that $\exists \theta, \exists \sigma \in \theta$ such that $\sigma \models l$.
   (d) From (b) and Definition 14, it follows that $\exists \theta, \neg\exists \sigma \in \theta$ such that $\sigma \models l$.
   (e) From (c), Definition 11 and Definition 10, it follows that $\exists t \in P$ such that $l \in \mathsf{ann}(t)$.
   (f) From (e), it follows that the condition at line 2 of the algorithm is false and the else case is executed.

(g) From (d) and Lemma 1, it follows that $R(B, \{t \mid \text{t is a task in } B \text{ and } l \in \mathsf{ann}(t)\}) \neq \bot$.

(h) From (f) and (g), it follows that the condition at line 5 of the algorithm is reached and is false.

(i) From (h), it follows that the algorithm returns $(P, \mathsf{ann}) \vdash^{\mathrm{P}} O^a(l)$ at line 8.

3. (P, ann) is not compliant with $O^a(l)$

(a) From the assumption and from Definition 16, it follows that $\neg\exists\theta \in \Theta(P, \mathsf{ann})$, $\theta \vdash O^a(l)$.

(b) From (a) and Definition 14, it follows that $\forall\theta, \neg\exists\sigma \in \theta$ such that $\sigma \models l$.

(c) From (b), Definition 11 and Definition 10, it follows that $\neg\exists t \in P$ such that $l \in \mathsf{ann}(t)$.

(d) From (c), it follows that the condition at line 2 of the algorithm is true.

(e) From (d), it follows that the algorithm returns $(P, \mathsf{ann}) \nvdash O^a(l)$ at line 3.

I thus have shown that the algorithm returns its outcome in accordance to the compliance class to which the process given in input belongs to.

Having proven both soundness and completeness of Algorithm 1, I can conclude that the algorithm is correct. □

A.3 Procedure First

Procedure 2 (First)**.** *Given a process block B,* $\mathsf{First}(B)$ *returns a set of tasks as follows:*

1: ***if*** $E = t$ ***then***
2: ***return*** $\{t\}$
3: ***end if***
4: ***if*** $E = \mathsf{SEQ}(B_1, \ldots, B_k)$ ***then***
5: ***return*** $\mathsf{First}(B_1)$
6: ***end if***
7: ***if*** $E = \mathsf{AND}(B_1, \ldots, B_k)$ ***then***
8: ***return*** $\bigcup_{i=1}^{k} \mathsf{First}(B_i)$
9: ***end if***
10: ***if*** $E = \mathsf{XOR}(B_1, \ldots, B_k)$ ***then***
11: ***return*** $\bigcup_{i=1}^{k} \mathsf{First}(B_i)$
12: ***end if***

Lemma 2 (First)**.** *Let the function $f(\epsilon)$ returns the first task of the sequence corresponding to the serialisation ϵ. The set returned by the procedure* First Tasks *fulfils the following condition: $t \in First(B)$ if and only if $\exists\epsilon \in \Sigma(B) | f(\epsilon) = t$.*

Proof. The proof directly follows from Definition 4. The procedure analyses the possible types of blocks and recursively selects the sub-blocks which can contain a task which can appear first in one of the executions of B. □

A.4 Procedure Task Rooting

Procedure 3 (Task Rooting)**.** *Given a process block B and a task $t \in \mathsf{First}(B)$,* task rooting *$F(B,t)$ returns a new process block as follows:*

1: ***if*** $B = t$ ***then***
2: ***return*** B
3: ***end if***
4: ***if*** $B = \mathsf{SEQ}(B_1, \ldots, B_k)$ ***then***
5: ***return*** $\mathsf{SEQ}(F(B_1,t), B_2, \ldots, B_k)$
6: ***end if***
7: ***if*** $B = \mathsf{XOR}(B_1, \ldots, B_k)$ ***then***
8: ***return*** $F(B_p,t)$ *where* $B_p \in \{B_1, \ldots, B_k\}$ *and* $t \in B_p$
9: ***end if***
10: ***if*** $B = \mathsf{AND}(B_1, \ldots, B_k)$ ***then***
11: ***if*** $k = 2$ ***then***
12: ***return*** $\mathsf{SEQ}(F(B_p,t), B_q)$, *where* $\{p,q\} = \{1,2\}$ *and* $t \in B_p$
13: ***end if***
14: ***return*** $\mathsf{SEQ}(F(B_p,t), \mathsf{AND}(B_{i_1}, \ldots, B_{i_{k-1}}))$, *where* $\{i_1, \ldots, i_{k-1}, p\} = \{1, \ldots, k\}$ *and* $t \in B_p$
15: ***end if***

Lemma 3 (Task Rooting)**.** *Let B be a process block, t be a task such that $t \in \mathsf{First}(B)$, $\Sigma_t(B)$ be the set of executions of B that start with t and $\Theta_t(B, \mathsf{ann})$ be the set of traces associated to $\Sigma_t(B)$ given an annotation function ann. The procedure task rooting, $R(B,t)$, is subject to the following properties:*

1. $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$
2. $\forall \epsilon \in \Sigma_t(B), \exists \epsilon' \in \Sigma(F(B,t))$ *such that:*
   (a) $f(\epsilon) = f(\epsilon') = t$
   (b) *The task set of ϵ = the task set of ϵ'*

I prove the two statements of Lemma 3 separately.

Proof. I prove here the first statement of Lemma 3 by induction: $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$
Base Case: B is a single task t.

1. From the assumption that $B = t$ and Definition 4, it follows that $\Sigma(B) = \{(t)\}$.
2. From the assumption that $t \in \mathsf{First}(B)$ and 1, it follows that $\Sigma_t(B) = \{(t)\}$.
3. From the assumption that $B = t$, it follows that the procedure returns B at line 2.
4. From 3, 2 and 1, it follows that $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$.

Inductive Cases:

1. $B = \mathsf{SEQ}(B_1, \ldots, B_k)$
   (a) From the assumption, it follows that the procedure returns $\mathsf{SEQ}(F(B_1,t), B_2, \ldots, B_k)$ at line 5.
   (b) From (a) and Definition 4, it follows that $\Sigma(F(B,t)) = \Sigma(F(B_1,t)) +_{\mathbb{P}} \Sigma(B_2) +_{\mathbb{P}} \ldots +_{\mathbb{P}} \Sigma(B_k)$.
   (c) From (b) and induction hypothesis, it follows that $\Sigma(F(B_1,t)) \subseteq \Sigma_t(B_1)$.
   (d) From (c) and (b), it follows $\Sigma(F(B_1,t)) +_{\mathbb{P}} \Sigma(B_2) +_{\mathbb{P}} \ldots +_{\mathbb{P}} \Sigma(B_k) \subseteq \Sigma_t(B_1) +_{\mathbb{P}} \Sigma(B_2) +_{\mathbb{P}} \ldots +_{\mathbb{P}} \Sigma(B_k)$.
   (e) From (d) and Definition 4, it follows that $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$.
2. $B = \mathsf{XOR}(B_1, \ldots, B_k)$
   (a) From the assumption, it follows that the procedure returns $F(B_p,t)$ where $B_p \in \{B_1, \ldots, B_k\}$ and $t \in B_p$ at line 8.
   (b) From (a) and Definition 4, it follows that $\Sigma(F(B_p,t)) \subset \Sigma(B)$.
   (c) From (b) and induction hypothesis, it follows that $\Sigma(F(B_p,t)) \subseteq \Sigma_t(B_p)$.
   (d) From (c) and Definition 4, it follows that $\Sigma_t(B_p) \subseteq \Sigma_t(B)$.
   (e) From (d), it follows that $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$.
3. $B = \mathsf{AND}(B_1, \ldots, B_k)$
   (a) From the assumption, it follows that the procedure either returns $\mathsf{SEQ}(F(B_p,t), B_q)$, where $B_p, B_q \in \{B_1, \ldots, B_k\}, t \in B_p$ and $p \neq q$ at line 12 or it returns $\mathsf{SEQ}(F(B_p,t), \mathsf{AND}(B_{i_1}, \ldots, B_{i_{k-1}}))$, where $\{i_1, \ldots, i_{k-1}, p\} = \{1, \ldots, k\}$ and $t \in B_p$ at line 14.
      i. Assume that the procedure returns $\mathsf{SEQ}(F(B_p,t), B_q)$, where $B_p, B_q \in \{B_1, \ldots, B_k\}, t \in B_p$ and $p \neq q$ at line 12.
      ii. From i and Definition 4, it follows that $\Sigma(\mathsf{SEQ}(F(B_p,t), B_q))$.
      iii. From ii and induction hypothesis, it follows that $\Sigma(F(B_p,t)) \subseteq \Sigma_t(B_p)$.

iv. From *iii* and Definition 4, it follows that $\Sigma_t(B_p) \subseteq \Sigma_t(B)$.

v. From *iv*, it follows that $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$.

i. Assume that the procedure returns $\mathsf{SEQ}(F(B_p,t), \mathsf{AND}(B_{i_1}, \ldots, B_{i_{k-1}}))$, where $\{i_1, \ldots, i_{k-1}, p\} = \{1, \ldots, k\}$ and $t \in B_p$ at line 14.

ii. From *i* and Definition 4 it follows that $\Sigma(\mathsf{SEQ}(F(B_p,t), \mathsf{AND}(B_{i_1}, \ldots, B_{i_{k-1}})))$.

iii. From *ii* and induction hypothesis, it follows that $\Sigma(F(B_p,t)) \subseteq \Sigma_t(B_p)$.

iv. From *iii* and Definition 4, it follows that $\Sigma_t(B_p) \subseteq \Sigma_t(B)$.

v. From *iv*, it follows that $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$.

(b) Therefore in both cases, it follows that $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$.

Therefore it follows that for every process block B, the result of procedure is subject to $\Sigma(F(B,t)) \subseteq \Sigma_t(B)$. □

Proof. I prove here the second statement of Lemma 3 by induction: $\forall \epsilon \in \Sigma_t(B), \exists \epsilon' \in \Sigma(F(B,t))$ such that:

1. $f(\epsilon) = f(\epsilon') = t$
2. The task set of ϵ = the task set of ϵ'

Base Case:
B is a single task.

1. From the assumption that $B = t$ and Definition 4, it follows that $\Sigma(B) = \{(t)\}$.
2. From the assumption that $B = t$, it follows that the procedure returns B at line 2.
3. From the assumption that $t \in \mathsf{First}(B)$ and 1, it follows that $\Sigma_t(B) = \{(t)\}$.
4. From 2 and 3, it follows that $\forall \epsilon \in \Sigma_t(B), \exists \epsilon' \in \Sigma(F(B,t))$ such that $f(\epsilon) = f(\epsilon') = t$.
5. From 2 and 3, it follows that $\forall \epsilon \in \Sigma_t(B), \exists \epsilon' \in \Sigma(F(B,t))$ such that the task set of ϵ = the task set of ϵ'.

Inductive Cases:

1. $B = \mathsf{SEQ}(B_1, \ldots, B_k)$

   (a) From the assumption, it follows that the procedure returns $\mathsf{SEQ}(F(B_1,t), \ldots, B_k)$ at line 5.

   (b) From (a) and induction hypothesis, it follows that $\forall \epsilon \in \Sigma_t(B), \exists \epsilon' \in \Sigma(F(B,t))$ such that $f(\epsilon) = f(\epsilon') = t$.

(c) Let B' be the sequence block B without B_1 and let B'' be the sequence block $\mathsf{SEQ}(F(B_1,t),\ldots,B_k)$ without $F(B_1,t)$.

(d) From (c) and Definition 4, it follows that $\Sigma(B') = \Sigma(B'')$.

(e) From (c) and induction hypothesis, it follows that $\forall\epsilon \in \Sigma_t(B_1), \exists\epsilon' \in \Sigma(F(B_1,t))$ such that the task set of ϵ = the task set of ϵ'.

(f) From (d) and (e), it follows that $\forall\epsilon \in \Sigma_t(B), \exists\epsilon' \in \Sigma(F(B,t))$ such that the task set of ϵ = the task set of ϵ'.

2. $B = \mathsf{XOR}(B_1,\ldots,B_k)$

(a) From the assumption, it follows that the procedure returns $F(B_i,t)$ at line 8, where $t \in B_i$.

(b) From (a) and induction hypothesis, it follows that $\forall\epsilon \in \Sigma_t(B_i), \exists\epsilon' \in \Sigma(F(B_i,t))$ such that $f(\epsilon) = f(\epsilon') = t$.

(c) From (b) and Definition 4, it follows that$\forall\epsilon \in \Sigma_t(B), \exists\epsilon' \in \Sigma(F(B,t))$ such that $f(\epsilon) = f(\epsilon') = t$[1].

(d) From (a) and induction hypothesis, it follows that $\forall\epsilon \in \Sigma_t(B_i), \exists\epsilon' \in \Sigma(F(B_i,t))$ such that the task set of ϵ = the task set of ϵ'.

(e) From the assumption and (d), it follows that $\forall\epsilon \in \Sigma_t(B), \exists\epsilon' \in \Sigma(F(B,t))$ such that the task set of ϵ = the task set of ϵ'.

3. $B = \mathsf{AND}(B_1,\ldots,B_k)$

(a) From the assumption, it follows that the procedure either returns $\mathsf{SEQ}(F(B_p,t),B_q)$, where $B_p, B_q \in \{B_1,\ldots,B_k\}, t \in B_p$ and $p \neq q$ at line 12 or it returns $\mathsf{SEQ}(F(B_p,t),\mathsf{AND}(B_{i_1},\ldots,B_{i_{k-1}}))$, where $\{i_1,\ldots,i_{k-1},p\} = \{1,\ldots,k\}$ and $t \in B_p$ at line 14.

i. Assume that the procedure returns $\mathsf{SEQ}(F(B_p,t),B_q)$, where $B_p, B_q \in \{B_1,\ldots,B_k\}, t \in B_p$ and $p \neq q$ at line 12.

ii. From i and induction hypothesis, it follows that $\forall\epsilon \in \Sigma_t(B_p), \exists\epsilon' \in \Sigma(F(B_p,t))$ such that $f(\epsilon) = f(\epsilon') = t$.

iii. From ii and Definition 4, it follows that$\forall\epsilon \in \Sigma_t(B), \exists\epsilon' \in \Sigma(F(B,t))$ such that $f(\epsilon) = f(\epsilon') = t$[2].

iv. From the assumption, i, Definition 4 and the induction hypothesis, it follows that $\forall\epsilon \in \Sigma_t(B), \exists\epsilon' \in \Sigma(F(B,t))$ such that the task set of ϵ = the task set of ϵ'.

[1]This follows because each task is unique.
[2]This follows because each task is unique.

i. Assume that the procedure returns $\mathsf{SEQ}(F(B_p,t),\mathsf{AND}(B_{i_1},\ldots,B_{i_{k-1}}))$, where $\{i_1,\ldots,i_{k-1},p\}=\{1,\ldots,k\}$ and $t\in B_p$ at line 14.

ii. From i and induction hypothesis, it follows that $\forall\epsilon\in\Sigma_t(B_p),\exists\epsilon'\in\Sigma(F(B_p,t))$ such that $f(\epsilon)=f(\epsilon')=t$.

iii. From ii and Definition 4, it follows that$\forall\epsilon\in\Sigma_t(B),\exists\epsilon'\in\Sigma(F(B,t))$ such that $f(\epsilon)=f(\epsilon')=t$[3].

iv. From the assumption, i, Definition 4 and the induction hypothesis, it follows that $\forall\epsilon\in\Sigma_t(B),\exists\epsilon'\in\Sigma(F(B,t))$ such that the task set of ϵ = the task set of ϵ'.

(b) Therefore in both cases, it follows that $\forall\epsilon\in\Sigma_t(B_p),\exists\epsilon'\in\Sigma(F(B_p,t))$ such that $f(\epsilon)=f(\epsilon')=t$ and the task set of ϵ = the task set of ϵ'.

Therefore it follows that for every process block B, the result of procedure is subject to $\forall\epsilon\in\Sigma_t(B_p),\exists\epsilon'\in\Sigma(F(B_p,t))$ such that $f(\epsilon)=f(\epsilon')=t$ and the task set of ϵ = the task set of ϵ'.

□

Lemma 4 (Task Rooting Approximation)**.** *Let ϵ_1 and ϵ_2 be two executions containing the same task set and $f(\epsilon_1)=f(\epsilon_2)$. Let θ_1 and θ_2 be the traces corresponding to the two executions. Given a maintenance obligation $O^m(l)$, $\theta_1\vdash O^m(l)$ iff $\theta_2\vdash O^m(l)$.*

Proof. Follows directly from Definitions 11, 15 and Lemma 3. □

A.5 Global Maintenance Algorithm

Algorithm 2. *Given a process (P,ann) and a global maintenance obligation $O^m(l)$, this algorithm returns whether (P,ann) is compliant with $O^m(l)$.*

1: *Suppose* $P=\mathsf{start}\ B\ \mathsf{end}$
2: $\mathcal{T}_F=\mathsf{First}(B)$
3: $\mathcal{T}_{\tilde{l}}=\{t|t$ *is a task in* B *and* $\tilde{l}\in\mathsf{ann}(t)\}$
4: ***if*** $\forall t$ *in* $\mathcal{T}_F, l\in\mathsf{ann}(t)$ *and* $\mathcal{T}_{\tilde{l}}=\emptyset$ ***then***
5: ***return*** $(P,\mathsf{ann})\vdash^F O^m(l)$
6: ***else***
7: ***for each*** $t\in\mathcal{T}_F$ *such that* $l\in\mathsf{ann}(t)$ ***do***
8: ***if*** $R(F(B,t),\mathcal{T}_{\tilde{l}})\neq\perp$ ***then***
9: ***return*** $(P,\mathsf{ann})\vdash^P O^m(l)$
10: ***end if***
11: ***end for each***

[3]This follows because each task is unique.

12: ***return*** $(P, \mathsf{ann}) \nvdash O^m(l)$
13: ***end if***

Correctness Proof of Algorithm 2. Given a process (P, ann), where $P = \mathsf{start}\ B\ \mathsf{end}$, and a global maintenance obligation $O^m(l)$, I prove the correctness of Algorithm 2 by showing both soundness and completeness.

Soundness I prove by cases the soundness of Algorithm 2, by showing that for each possible result of the algorithm, the process being checked against an atomic global maintenance obligation is indeed either fully, partially or not compliant according to the result.

1. Algorithm 2 returns $(P, \mathsf{ann}) \vdash^{\mathrm{F}} O^m(l)$:
   (a) From the assumption, it follows that $\forall t$ in $\mathcal{T}_F, l \in \mathsf{ann}(t)$ and $\neg\exists t \in P$ such that $\tilde{l} \in \mathsf{ann}(t)$.
   (b) From (a), Definition 4 and Definition 10, it follows that $\forall\theta \in \Theta(P), \forall\sigma \in \theta, \sigma \models l$.
   (c) From (b) and Definition 15, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \vdash O^m(l)$.
   (d) From (c) and Definition 16, it follows that (P, ann) is fully compliant with $O^m(l)$.
2. Algorithm 2 returns $(P, \mathsf{ann}) \vdash^{\mathrm{P}} O^m(l)$:
   (a) From the assumption, it follows that $\exists t \in \mathsf{First}(B)$ such that $l \in \mathsf{ann}(t)$ and $R(F(B,t), \mathcal{T}_{\tilde{l}}) \neq \bot$ where $\mathcal{T}_{\tilde{l}}$ is the set of tasks in B such that $\tilde{l}$ is in their annotation.
   (b) From (a) and Lemma 1, it follows that $\exists\epsilon \in \Sigma(B)$ such that the first task executed has l annotated and does not execute any tasks having $\tilde{l}$ annotated.
   (c) From (b) and Definition 10, it follows that $\exists\theta \in \Theta(P, \mathsf{ann})$ such that $\forall\sigma \in \theta, \sigma \models l$.
   (d) From (c) and Definition 15, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \vdash O^m(l)$.
   (e) From the assumption, it follows that condition at line 4 was false, hence $\exists t$ in $\mathcal{T}_F, l \notin \mathsf{ann}(t)$ or $\exists t \in B$ such that $\tilde{l} \in \mathsf{ann}(t)$.
   (f) From (e), Definition 4 and Definition 10, it follows that $\exists\theta \in \Theta(P, \mathsf{Ann})$ such that $\exists\sigma \in \theta, \sigma \not\models l$.
   (g) From (f) and Definition 15, it follows that $\exists\theta \in \Theta(P\mathsf{ann}), \theta \nvdash O^m(l)$.
   (h) From (d), (f) and Definition 16, it follows that (P, ann) is partially compliant with $O^m(l)$ but not fully compliant with $O^m(l)$.
3. Algorithm 2 returns $(P, \mathsf{ann}) \nvdash O^m(l)$:
   (a) From the assumption, it follows that $\forall t \in \mathsf{First}(B)$ such that $l \in \mathsf{ann}(t), R(F(B,t), \mathcal{T}_{\tilde{l}}) = \bot$, where $\mathcal{T}_{\tilde{l}}$ contains each task in B having $\tilde{l}$ in their annotation.

(b) From (a) and Lemma 1, it follows that $\forall\epsilon$ where the first task executed contains l in their annotation, $\exists t \in \epsilon$ such that $\tilde{l} \in \mathsf{ann}(t)$.

(c) From (b) and Definition 10, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \exists\sigma \in \theta$ such that $\sigma \not\models l$.

(d) From (c) and Definition 15, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \not\vdash O^m(l)$.

(e) From (d) and Definition 16, it follows that (P, ann) is not compliant with $O^m(l)$.

I thus have shown that for each of the three possible outcomes, the algorithm correctly classifies a process according whether it is fully, partially or not compliant with an atomic global achievement obligation.

Completeness Following from Definition 16 a process can be either fully compliant, or partially compliant, or not compliant with an atomic global maintenance obligation. I prove the completeness of Algorithm 2 by cases considering the three possible cases listed above, and showing that for each case the algorithm returns the corresponding result.

1. (P, ann) is fully compliant with $O^m(l)$:

   (a) From the assumption and Definition 16, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \vdash O^m(l)$.

   (b) From (a) and Definition 15, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \forall\sigma \in \theta, \sigma \models l$.

   (c) From (b) and Lemma 2, it follows that $\forall t \in \mathsf{First}(B), l \in \mathsf{ann}(t)$.

   (d) From (b), Lemma 1 and Definition 10, it follows that $\neg\exists t \in P$ such that $\tilde{l} \in \mathsf{ann}(t)$.

   (e) From (c) and (d), it follows that the condition at line 4 of the algorithm is true, hence the algorithm returns $(P, \mathsf{ann}) \vdash^{\mathrm{F}} O^m(l)$ at line 5.

2. (P, ann) is partially compliant with $O^m(l)$ and not fully compliant with the same obligation:

   (a) From the assumption and Definition 16, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \vdash O^m(l)$ and $\neg\forall\theta \in \Theta(P, \mathsf{ann}), \theta \vdash O^m(l)$.

   (b) From (a) and Definition 15, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \forall\sigma \in \theta, \sigma \models l$.

   (c) From (a) and Definition 15, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \exists\sigma \in \theta, \sigma \not\models l$.

   (d) From (b) and Lemma 2, it follows that $\exists t \in \mathsf{First}(B), l \in \mathsf{ann}(t)$.

   (e) From (c), it follows that either $\exists t \in \mathsf{First}(B), l \notin \mathsf{ann}(t)$ or $\exists t \in B$ such that $\tilde{l} \in \mathsf{ann}(t)$.

   (f) From (e), it follows that condition at line 4 is false hence the cycle at line 7 is entered.

(g) From (f), (b) and Lemma 1, it follows that the condition at line 8 is true, hence the algorithm returns $(P, \mathsf{ann}) \stackrel{\mathrm{P}}{\vdash} O^m(l)$ at line 9.

3. (P, ann) is not compliant with $O^m(l)$:

   (a) From the assumption and Definition 16, it follows that $\forall \theta \in \Theta(P, \mathsf{ann}), \theta \not\vdash O^m(l)$.

   (b) From (a) and Definition 15, it follows that $\forall \theta \in \Theta(P, \mathsf{ann}), \exists \sigma \in \theta, \sigma \not\models l$.

   (c) From (b), Definition 4 and Definition 10, it follows that either $\forall t \in \mathsf{First}(B), l \in \mathsf{ann}(t), \exists t \in B$ such that $\tilde{l} \in \mathsf{ann}(t)$.

   (d) From (d), it follows that the condition at line 4 is false and the cycle at line 7 is entered.

   (e) From (e), (d) and Lemmalemma.taskremoval, it follows that the condition at line 8 is always false.

   (f) From (f), it follows that the algorithm returns $(P, \mathsf{ann}) \not\vdash O^m(l)$ at line 12.

   From the premise and Definition 16 it follows that none of the traces of (P, ann) fulfills $O^m(l)$. Thus from Definition 15 it follows that for each trace of (P, ann), there exists a state not verifying l. Assuming that $P = \mathsf{start}\ B\ \mathsf{end}$, this state can be either a) the first of a trace, meaning that some of the tasks of $First(B)$ do not contain l in their annotation, or b) one of the other states, meaning that a task is executed which contains $\tilde{l}$ in its annotations.

I thus have shown the soundness of Algorithm 2 by showing that in each of the three possible cases, whether a process is fully, partially or not compliant, the algorithm returns the correct answer.

Having proven both soundness and completeness of Algorithm 2, I can conclude that the algorithm is correct.

□

Appendix B

Difficulty Vectors and Conflicting Obligations

B.1 Local Obligations

Lemma 5. *Given a θ and a local obligation $\mathfrak{O} = \mathcal{O}^t\langle c, l, d\rangle$. If $\neg\exists\sigma \in \theta$ such that $\sigma \models l$, then $\theta \vdash \mathfrak{O}$.*

Proof. Follows directly from Definitions 19, 20 and 21. □

B.2 Conflicting Maintenance Obligations

Definition 33 ($\mathcal{O}^m - \mathcal{O}^m$ Conflict)**.** *Let $\mathcal{O}^m\langle\alpha\rangle$ and $\mathcal{O}^m\langle\beta\rangle$ be two complementary maintenance obligations. $\mathcal{O}^m\langle\alpha\rangle$ and $\mathcal{O}^m\langle\beta\rangle$ are conflicting if and only if:*

$$\exists I \in \mathit{Interval}(\mathcal{O}^m\langle\alpha\rangle, \theta) \text{ and } \exists I' \in \mathit{Interval}(\mathcal{O}^m\langle\beta\rangle, \theta) : I \cap I' \neq \emptyset$$

Proposition 3 ($\mathcal{O}^m - \mathcal{O}^m$ Conflict)**.** *Let $\mathcal{O}^m = \langle\alpha\rangle$ and $\mathcal{O}'^m = \langle\beta\rangle$ be conflicting maintenance obligations, then there not exists a trace complying with both obligations.*

$\mathcal{O}^m - \mathcal{O}^m$ Conflict. I prove that the condition provided in Definition 33 is sufficient to identify whether two maintenance obligations are conflicting.

1. Let $\mathcal{O}^m = \langle\alpha\rangle$ and $\mathcal{O}'^m = \langle\beta\rangle$ be two complementary maintenance obligations, meaning that $\alpha \wedge \beta \rightarrow \bot$.
2. From the hypothesis we know that $\exists I, I'$ such that $I \in \mathit{Interval}(\mathcal{O}^m, \theta), I' \in \mathit{Interval}(\mathcal{O}'^m, \theta)$ and $\exists\sigma$ such tha
   I and $\sigma \in I'$.
3. From Definition 24 and 2. it follows that $\forall\sigma \in I, \sigma \models \alpha$ and $\forall\sigma' \in I'\sigma' \models \beta$.

4. Assume that there exists a trace θ such that θ is compliant with $\mathcal{O}^m$ and $\mathcal{O}'^m$.

5. From 4. it follows that $\forall I, I'$ such that $I \in \mathit{Interval}(\mathcal{O}^m, \theta)$ and $I' \in \mathit{Interval}(\mathcal{O}'^m, \theta), I \subseteq \theta$ and $I' \subseteq \theta$ and $\forall \sigma \in I, \sigma \models \alpha$ and $\forall \sigma' \in I', \sigma' \models \beta$.

6. From 2. and 5. it follows that $\exists \sigma\theta$ such that $\sigma \models \alpha$ and $\sigma \models \beta$.

7. From Definition 13 and 6. it follows that $\exists \sigma \in \theta$ such that $\{\alpha, \beta\} \in \sigma$.

8. From 1. we know that $\alpha \wedge \beta \rightarrow \bot$, hence from 7. and Definition 9 it follows that a state σ is inconsistent and a trace containing such state cannot exists.

Therefore I have proven that the condition provided in Proposition 3 is sufficient to identify to conflicting complementary maintenance obligations.

□ □

I do not explicitly provide propositions and formal proves for the other definitions concerning conflicting obligations since they are analogous of the one provided for Definition 33. Which for each definition of conflicting obligations shows that a trace fulfilling both the obligations involved given the condition identified would require to include an inconsistent state which is not allowed according to Definition 9.

Appendix C

Some Complexity Results

C.1 $C2^{-}_{nla}$

C.1.1 Partial Compliance

Algorithm 3. *Given a trace* $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ *where* $\sigma_{start} = (\mathsf{start}, L_0)$ *and* $\sigma_{end} = (\mathsf{end}, L_{n+1})$*, the corresponding execution* $\epsilon = (t_1, \ldots, t_n)$*, and process* (P, ann) *where* B *is the main process block of* P*, the following algorithm* $A_1(\theta, \epsilon, (P, \mathsf{ann}), B)$ *decides if* θ *is a valid trace of* (P, ann)*.*

Algorithm A_1

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

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

Procedure 4. $P_1(\epsilon, B)$

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

(c) *if* $B = \mathsf{AND}(B_1, \ldots, B_n)$*, then* ϵ *is valid if the set of tasks in* ϵ *is the disjoint union of the sets of tasks in* ϵ_i *(for each i) and each* ϵ_i *is a valid serialisation for* B_i

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

Procedure 5. $P_2(\theta, (P, \mathsf{ann}))$

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

P_2 *returns true if all of these properties hold and* $false$ *otherwise.*

Correctness:

Proof. I prove the correctness directly.

1. The first part of procedure $\mathbf{P_1}$ verifies the first property of Definition 4.
2. The second part of the procedure $\mathbf{P_1}$ verifies the three parts of the second property of Definition 4.
3. The uniqueness of a task, required by Definition 1, is given by construction of the trace.
4. From 1, 2 and 3, it follows that $\mathbf{P_1}$ is correct.
5. The correctness of procedure $\mathbf{P_2}$ follows directly from Definition 11.
6. From 4 and 5, it follows that algorithm A_1 is correct.

□

Algorithm 4. *Given a set of obligations* $\circledcirc$ *and a trace* $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ *such that* $\sigma_{start} = (\mathsf{start}, L_0)$ *and* $\theta \in \Theta(P, \mathsf{ann})$*, the algorithm* $A_2(\theta, \circledcirc)$ *is defined as follows (In the following,* Ob *denotes the set of active obligations and we treat* θ *as a vector):*

Algorithm A_2

1: $\mathsf{Ob} = \emptyset$
2: ***for each*** σ *in* θ ***do***
3: ***for each*** $\mathbf{O} = \mathcal{O}^t\langle l_c, l_b, l_d\rangle$ *in* $\circledcirc$ ***do***
4: ***if*** $\sigma \models l_b$ ***then***
5: $\mathsf{Ob} = \mathsf{Ob} \cup \mathbf{O}$
6: ***end if***

```
    end for each
    for each ❍ = O^t⟨l_c, l_b, l_d⟩ in Ob do
      if t = a then
        if σ ⊨ l_c then
          Ob = Ob \ ❍
        else
          if σ ⊨ l_d then
            return θ ⊬ ◎
          end if
        end if
      else
        if t = m then
          if σ ⊭ l_c then
            return θ ⊬ ◎
          end if
          if σ ⊨ l_d then
            Ob = Ob \ ❍
          end if
        end if
    end for each
  end for each
  return θ ⊢ ◎;
```

Algorithm 4 identifies wether a certificate fulfils a set of obligations. If the certificate is a valid trace of a process, then following from Definition 37, the fact that the certificate fulfils the set of obligations is a sufficient condition to say that the process is partially compliant with the regulatory framework containing such set of obligations.

Correctness:

Proof. **Soundness:** $A_2(\theta, \circledcirc) = \theta \vdash \circledcirc \Rightarrow \theta \vdash \circledcirc$.
I prove the soundness by contradiction:

1. Assume that $\theta \nvdash \circledcirc$
2. From 1, Definition 16 and Definition 26, it follows that exists an obligation ❍ in $\circledcirc$ such that $\theta \nvdash$ ❍.
3. From 2 and Lemma 5, it follows that ❍ must be activated to be not fulfilled.
4. In the case the lifeline of an obligation is triggered, the obligation activated can be either an achievement or a maintenance obligation. I analyse the two cases separately:
5. Let an ❍ be an achievement obligation.

(a) From 2, 3, 5 and Definition 20, it follows that $\mathbf{O}$ is triggered in a state σ_h and the following holds: $\exists\sigma_i \in \theta$ such that $\sigma_i \succeq \sigma_h$ and $\sigma_i \models l_d$ and $\neg\exists\sigma_j \in \theta$ such that $\sigma_j \models l_c$ and $\sigma_h \prec \sigma_j \prec \sigma_i$.

(b) From 3 and 5, it follows that the lines between 9 and 16 of Algorithm 4 are executed.

(c) From the fact that the algorithm analyses the states of a trace chronologically, (a) and (b), it follows that the condition at line 9 is never fulfilled before the condition in line 13.

(d) From (c), it follows that line 14 is executed and returns $\theta \nvdash \circledcirc$.

6. Let an $\mathbf{O}$ be a maintenance obligation.

(a) From 2, 3, 6 and Definition 21, it follows that $\mathbf{O}$ has been triggered in a state σ_h and the following holds: $\exists\sigma_i \in \theta$ such that $\sigma_i \succeq \sigma_h$ and $\sigma_i \models \varphi_d$ and $\exists\sigma_j \in \theta$ such that $\sigma_j \not\models \varphi_c$ and $\sigma_h \prec \sigma_j \preceq \sigma_i$.

(b) From 3 and 6, it follows that the lines between 18 and 25 of Algorithm 4 are executed.

(c) From the fact that the algorithm analyses the states of a trace chronologically, (a) and (b), it follows that the condition at line 22 is never fulfilled before the condition in line 19.

(d) From (c), it follows that the obligation is never removed from the loop which will end in fulfilling line 19 is and executing 20.

(e) From (d), it follows that the algorithm returns $\theta \nvdash \circledcirc$.

7. In both cases (5.(d) and 6.(e)) the result contradicts the premise that $A_2(\theta, \circledcirc) = \theta \vdash \circledcirc$.

8. Therefore from 7, it follows that if $A_2(\theta, \circledcirc) = \theta \vdash \circledcirc$, then $\theta \vdash \circledcirc$ is true.

□

Proof. **Completeness:** $\theta \vdash \circledcirc \Rightarrow A_2(\theta, \circledcirc) = \theta \vdash \circledcirc$.
I prove the completeness directly:

1. From the hypothesis, Definition 16 and Definition 26, it follows that $\forall\mathbf{O} \in \circledcirc, \theta \vdash \mathbf{O}$.

2. From line 3 of the algorithm, Definition 20 and Definition 21, it follows that if an obligation is activated, it will be in Ob.

3. In the case the lifeline of an obligation is triggered, the obligation activated can be either an achievement or a maintenance obligation. I analyse the two cases separately:

4. Let an $\mathbf{O} \in \mathsf{Ob}$ be an achievement obligation.
   (a) From 4 it follows that in this case the lines of the algorithm from 9 to 16 are concerned.
   (b) From (a) it follows that the only line of the algorithm returning *not compliant* is 14.
   (c) From 1 and Definition 20, it follows that $\exists \sigma_j \in \theta$ such that $\sigma_j \models c$ and $\sigma_j \succeq \sigma_i$, and $\neg\exists \sigma_h \in \theta$ such that $\sigma_h \models d$ and $\sigma_i \preceq \sigma_h \prec \sigma_j$.
   (d) From (c) and because the algorithm analyses the states of a trace in chronological order, it follows that the condition at line 13 of the algorithm is never satisfied, because the condition of line 13 cannot be fulfilled before the condition at line 10.
   (e) From (d), it follows that when the condition at line 10 is fulfilled, it removes the obligation from the cycle, hence preventing the execution of line 14.
   (f) From (b) and (e), it follows that the result *not compliant* is prevented while analysing achievement obligations.
5. Let an $\mathbf{O} \in \mathsf{Ob}$ be a maintenance obligation.
   (a) From 5 it follows that in this case the lines of the algorithm from 18 to 25 are concerned.
   (b) From (a), it follows that the only line of the algorithm returning *not compliant* is 20.
   (c) From 1 and Definition 21, it follows that $\exists \sigma_h \in \theta$ such that $\sigma_h \models d$ and $\forall \sigma_j \in \theta$ such that $\sigma_i \preceq \sigma_j \preceq \sigma_h : \sigma_j \models c$.
   (d) From (c) and because the algorithm analyses the states of a trace in chronological order, it follows that the condition at line 19 is never satisfied in the same state or in one preceding a state satisfying the condition at line 22.
   (e) From (d), it follows that when the condition at line 22 is fulfilled, it removes the obligation from the cycle, hence preventing the execution of line 20.
   (f) From (b) and (e), it follows that the result *not compliant* is prevented while analysing maintenance obligations.
6. I have thus shown that in both cases the algorithm cannot return *not compliance*. Therefore Algorithm 4 returns *compliant*, $\theta \vdash \mathbf{O}$, as result.

□

Reduction 1. *Given a directed graph G and the problem of proving regulatory compliance reduced from it, where the problem is composed by a process model (P, ann) and a set of local obligations $\circledcirc$. There exists a trace $\theta \in \Theta(P, \mathsf{ann})$ such that $\theta \vdash \circledcirc$ if and only if G has an hamiltonian path (ham).*

Correctness:

Here I prove the soundness ($(P, \mathsf{ann}) \vdash^{\mathrm{P}} \circledcirc \Rightarrow \exists ham$) and the completeness ($\exists ham \Rightarrow (P, \mathsf{ann}) \vdash^{\mathrm{P}} \circledcirc$) of our reduction. We refer to the two conditions stated in Definition 39 as (1) and (2) respectively.

Proof. **Soundness:** $(P, \mathsf{ann}) \vdash^{\mathrm{P}} \circledcirc \Rightarrow \exists ham$
I prove the soundness directly:

1. By construction each node is included in an AND block.
2. From 1 and Definition 4, it follows that each block in the AND block must be serialised.
3. From 2, it follows that the condition (1) of Definition 39 is satisfied.
4. From the hypothesis and Definition 16, it follows that $\exists \theta \in \Theta(P, \mathsf{ann})$ such that $\theta \vdash \circledcirc$.
5. From 4 and Definition 26, it follows that $\forall \mathbf{O} \in \circledcirc, \theta \vdash \mathbf{O}$.
6. By construction each obligation $\mathbf{O}$ in $\circledcirc$ is of type maintenance.
7. By construction and Definition 4, it follows that there exists a trace $\theta = ((\mathsf{start}, L_0), (Node_{i_1}, L_1), \ldots, (Node_{i_n}, L_n), (\mathsf{end}, L_{n+1}))$.
8. From 7 and 4, it follows that $\theta \vdash \circledcirc$.
9. From 8, it follows that no maintenance obligation $\langle O^m(\neg l_{i_{k+1}}), l_{i_k}, \neg l_{i_k} \rangle \in \circledcirc$ is violated.
10. From 9 and by construction of the maintenance obligations, it follows that for each $(Node_{i_k}, L_k)$ and $(Node_{i_{k+1}}, L_{k+1})$ in θ, there exists $(v_{i_k}, v_{i_{k+1}}) \in D$.
11. From 10, it follows there exists and edge between v_{i_k} and $v_{i_{k+1}}$ in G.
12. From 11, it follows that condition (2) of Definition 39 is fulfilled.
13. From 3 and 12, it follows that $\exists ham$ in G.

□

Proof. **Completeness:** $\exists ham \Rightarrow (P, \mathsf{ann}) \vdash^{\mathrm{P}} \circledcirc$
I prove the completeness directly:

1. From the hypothesis we know that $\exists ham = (v_1; \ldots; v_n)$ satisfying conditions (1) and (2) of Definition 39.

2. By construction of the process model, it follows that each v_i in *ham* corresponds to a $Node_i$ in P.

3. From 2 and Definition 4, it follows that $\epsilon = (Node_1, \ldots, Node_n)$ is a valid execution of P.

4. From 3 and Definition 11, it follows that $\theta = ((\mathsf{start}, L_0), (Node_1, L_1), \ldots, (Node_k, L_k), (\mathsf{end}, L_{n+1}))$ is a valid trace of (P, ann).

5. From 1, it follows that for each $v_i, v_{i+1} \in ham$ there exists $(v_i, v_{i+1}) \in D$.

6. From 5 and the construction of the maintenance obligations, it follows that there is no obligation $\mathbf{O} = \langle O^m(\neg l_{i+1}), l_i, \neg l_i \rangle$.

7. By construction of the annotations Definition 11 and Definition 21, it follows that for each $(Node_i, L_i)$ and $(Node_{i+1}, L_{i+1})$ in θ, only $\mathbf{O}$ would not be fulfilled in θ.

8. From 6, 7 and Definition 26, it follows that $\theta \vdash \circledcirc$.

9. From 8 and Definition 16, it follows that $(P, \mathsf{ann}) \stackrel{\mathrm{P}}{\vdash} \circledcirc$.

□

C.2 $C2_{nla}$

C.2.1 Proving Partial Compliance is NP-complete

Algorithm 5. *Given a set of obligations* $\circledcirc$ *and a trace* $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ *such that* $\sigma_{start} = (\mathsf{start}, L_0)$ *and* $\theta \in \Theta(P, \mathsf{ann})$*, the algorithm* $A_3(\theta, \circledcirc)$ *is defined as follows (In the following,* Ob *denotes the set of active obligations and we treat* θ *as a vector):*

Algorithm A_3

1: $\mathsf{Ob} = \emptyset$
2: ***for each*** σ_i *in* θ ***do***
3: ***for each*** $\mathcal{O}^t\langle\varphi_c, \varphi_b, \varphi_d\rangle$ *in* $\circledcirc$ ***do***
4: ***if*** $\sigma_i \models \varphi_b$ ***then***
5: $\mathsf{Ob} = \mathsf{Ob} \cup \mathcal{O}^t\langle\varphi_c, \varphi_b, \varphi_d\rangle$
6: ***end if***
7: ***end for each***
8: ***for each*** $\mathcal{O}^t\langle\varphi_c, \varphi_b, \varphi_d\rangle$ *in* Ob ***do***
9: ***if*** $t = a$ ***then***
10: ***if*** $\sigma_i \models \varphi_c$ ***then***
11: $\mathsf{Ob} = \mathsf{Ob} \setminus \mathcal{O}^a\langle\varphi_c, \varphi_b, \varphi_d\rangle$
12: ***else***
13: ***if*** $\sigma_i \models \varphi_d$ ***then***
14: ***return*** $\theta \not\vdash \circledcirc$

```
15:            end if
16:          end if
17:        else
18:          if t = m then
19:            if σ_i ⊭ φ_c then
20:              return θ ⊬ ⊙
21:            end if
22:            if σ_i ⊨ φ_d then
23:              Ob = Ob \ O^m⟨φ_c, φ_b, φ_d⟩
24:            end if
25:          end if
26:        end if
27:      end for each
28:    end for each
29:    return θ ⊢ ⊙;
```

Algorithm 5 identifies wether a trace fulfils a set of local obligations without compensations.

Correctness:

Proof. Because the semantics of verifying partial compliance (Definition 37) and the semantics of local obligations has not changed (Definitions 19, 20 and 21), the correctness proof provided in Section 5.1.2 for Algorithm 4 holds also for Algorithm 5. □

C.2.2 Proving Non Compliance is coNP-complete

Reduction 2. *Given a propositional formula φ and the problem of proving regulatory compliance reduced from it, where the problem is composed by a process model (P, ann) and a set of local obligations $\circledcirc$. For all traces $\theta \in \Theta(P, \mathsf{ann})$ we have $\theta \vdash \circledcirc$ if and only if φ is a tautology.*

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

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

1. From the hypothesis and Definition 37, it follows that each trace of the process (P, ann) fulfils the obligations in $\circledcirc$.
2. From the construction of the set of obligations, it follows that the only obligation belonging to $\circledcirc$ is $\mathbf{O} = \mathcal{O}^a\langle \varphi, l_{test}, \bot \rangle$.
3. From Definition 11 and the construction of the process, it follows that each trace of P contains the task l_{test}.

4. From 2, 3 and Definition 20, it follows that the only state where $\mathbf{O}$ is active is always the last of every trace of (P, ann), which are also the only one containing l_{test}.
5. From 4 and Definition 11, it follows that the set of literals associated to the last state corresponds to an interpretation of the propositions contained in φ.
6. From 5 and the construction of the process, it follows that all the possible combinations of interpreting the propositions belonging to φ are considered.
7. From 1 and 6, it follows that φ is satisfied by every interpretation.
8. Therefore from Definition 47, it follows that φ is indeed a tautology.

□

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

1. From the construction of the set of obligations, it follows that the only obligation contained in $\circledcirc$ is $\mathbf{O}$.
2. From the construction of the obligation, it follows that the condition of $\mathbf{O}$ is constituted by φ.
3. From the hypothesis and Definition 44, it follows that φ is verified in every state.
4. From 2, 3 and Definition 20, it follows that $\forall \theta, \theta \ \vdash \mathbf{O}$.
5. From 4, 1 and Definition 26, it follows that $\forall \theta, \theta \ \vdash \circledcirc$.
6. From 5, it follows that $\forall \theta \in (P, \mathsf{ann}), \theta \ \vdash \circledcirc$.
7. From 6 and Definition 37, it follows that $(P, \mathsf{ann}) \ \vdash^{\mathrm{F}} \ \circledcirc$.

□

Appendix D

Propagating the Results

D.1 $C3_{nlc}$

D.1.1 Proving Partial Compliance is NP-Complete

Lemma 6. *Given a compensable obligation* $\lightning = \mathfrak{O}_1 \otimes \ldots \otimes \mathfrak{O}_{n-1} \otimes \bot$ *composed of a chain containing* $n-1$ *obligations,* $\theta \vdash \lightning$ *if and only if* $\theta \vdash \mathfrak{O}_{n-1}$.

Proof. The proof follows directly by induction on the sequential structure of the chain composing a compensable obligation and Definition 29. □

Algorithm 6. *Given a set of obligations* $\circledcirc$ *and a trace* $\theta = (\sigma_{start}, \sigma_1, \ldots, \sigma_n, \sigma_{end})$ *such that* $\sigma_{start} = (\mathsf{start}, L_0)$ *and* $\theta \in \Theta(P, \mathsf{ann})$*, the algorithm* $A_3(\theta, \circledcirc)$ *is defined as follows (In the following,* Ob *denotes the set of active obligations and we treat* θ *as a vector):*

Algorithm A_4

1: $\mathsf{Ob} = \emptyset$
2: ***for each*** σ *in* θ ***do***
3: ***for each*** $\lightning$ *in* $\circledcirc$ ***do***
4: ***if*** $\lightning$ *not in* ob ***then***
5: *Let* $\lightning = \mathfrak{O}_1 \otimes \ldots \otimes \bot$ *and let* $\mathfrak{O}_1 = \mathcal{O}^t\langle\varphi_c, \varphi_b, \varphi_d\rangle$
6: ***if*** $\sigma_i \models \varphi_b$ ***then***
7: $\mathsf{Ob} = \mathsf{Ob} \cup \lightning.\mathsf{first}$
8: ***end if***
9: ***end if***
10: ***end for each***
11: ***for each*** $\lightning$ *in* Ob ***do***
12: $w = \mathsf{true}$
13: ***while*** w ***do***
14: $w = \mathsf{false}$
15: *Let the active obligation in* $\lightning$ *be* $\mathcal{O}^t\langle\varphi_c, \varphi_b, \varphi_d\rangle$
16: ***if*** $t = a$ ***then***
17: ***if*** $\sigma_i \models \varphi_c$ ***then***

18: $\mathsf{Ob} = \mathsf{Ob} \setminus \lightning$
19: ***else***
20: ***if*** $\sigma_i \models \varphi_d$ ***then***
21: $\lightning$.next
22: $w = \mathsf{true}$
23: ***if*** *The active obligation in* $\lightning$ *is* $\perp$ ***then***
24: ***return*** $\theta \nvdash \circledcirc$
25: ***end if***
26: ***end if***
27: ***end if***
28: ***else***
29: ***if*** $t = m$ ***then***
30: ***if*** $\sigma_i \not\models \varphi_c$ ***then***
31: $\lightning$.next
32: $w = \mathsf{true}$
33: ***if*** *The active obligation in* $\lightning$ *is* $\perp$ ***then***
34: ***return*** $\theta \nvdash \circledcirc$
35: ***end if***
36: ***end if***
37: ***if*** $\sigma_i \models \varphi_d$ ***then***
38: $\mathsf{Ob} = \mathsf{Ob} \setminus \lightning$
39: ***end if***
40: ***end if***
41: ***end if***
42: ***end while***
43: ***end for each***
44: ***end for each***
45: ***return*** $\theta \vdash \circledcirc$;

Algorithm 6 identifies wether a certificate, consisting of a trace θ, fulfils a set of local compensable obligations.

Correctness:

Proof. **Soundness:** $A_2(\theta, \circledcirc) = \theta \vdash \circledcirc \Rightarrow \theta \vdash \circledcirc$.
I prove the soundness of Algorithm 6 directly:

1. From the hypothesis, it follows that line 45 of Algorithm 6 is executed.
2. From 1., it follows that the lines 24 and 34 of the algorithm are not executed.
3. Given an obligation $\mathbf{O}$ being processed of an activated compensable obligation Co, assuming that $\mathbf{O}$ is violated, from 2 it follows that $\lightning.\mathsf{next} \neq \perp$.
4. Given that each $\lightning \in \circledcirc$ is composed by a finite sequence of obligations.
5. From 3. and 4. and Definition 46, it follows that $\forall \lightning \in \circledcirc, \exists \mathbf{O} \in \lightning$ such that $\theta \vdash \mathbf{O}$.

6. From 5. and Lemma 5, it follows that $\forall \lightning \in \circledcirc, \theta \vdash \mathbf{O}_{n-1}$ where $\mathbf{O}_{n-1}$ is the obligation preceding $\bot$ in each $\lightning$.
7. From 6. and Lemma 6, it follows that $\forall \lightning \in \circledcirc, \theta \vdash \lightning$.
8. From 7. and efinition 26, it follows that $\theta \vdash \circledcirc$.

□

Proof. **Completeness:** $\theta \vdash \circledcirc \Rightarrow A_4(\theta, \circledcirc) = \theta \vdash \circledcirc$.
I prove the completeness of Algorithm 6 directly:

1. From the hypothesis and Definition 26, it follows that $\forall \lightning \in \circledcirc, \theta \vdash \lightning$.
2. From 1. and Lemma 6, it follows that if $\theta \vdash \lightning$ then $\theta \vdash \mathbf{O}_{n-1}$.
3. From line 11 and line 15 of the algorithm, it follows that each activated compensable obligation is processed individually and the obligations composing it are processed in the given order.
4. Assume that the obligation being processed of an activated compensable obligation is of type achievement, then the if block starting in line 16 is used.
   (a) Assume the case where each previous obligation of a compensable obligation have been violated and the processed obligation is $\mathbf{O}_{n-1}$.
   (b) From 2. and Definition 20, it follows that either line 17 is fulfilled or line 20 is not.
   (c) From (*b*), it follows that the following obligation on the chain, $\bot$, is not processed.
   (d) From (*c*), it follows that the condition at line 23 is never fulfilled.
   (e) From (*d*), it follows that the only line reachable returning $\theta \nvdash \circledcirc$ when processing an obligation of type achievement is not reachable.
5. Assume that the obligation being processed of an activated compensable obligation is of type maintenance, then the if block starting in line 29 is used.
   (a) Assume the case where each previous obligation of a compensable obligation have been violated and the processed obligation is $\mathbf{O}_{n-1}$.
   (b) From 2. and Definition 20, it follows that line 30 is never fulfilled.
   (c) From (*b*), it follows that the following obligation on the chain, $\bot$, is not processed.
   (d) From (*c*), it follows that the condition at line 33 is never fulfilled.
   (e) From (*d*), it follows that the only line reachable returning $\theta \nvdash \circledcirc$ when processing an obligation of type achievement is not reachable.

6. Thus it follows that for every type of obligation the algorithm does not return $\theta \nvdash \circledcirc$, hence the algorithm returns $\theta \vdash \circledcirc$ at line 45.

□

Appendix E

Towards a Tractable Sub-Class of the Problem

E.1 $C1^*_{1la}$

Algorithm 7. *Given a* $C1^*_{1la}$ *problem composed of a monotonic process* (P, ann) *and a local atomic obligation* $\mathfrak{O}$*,* $A_5((P, \mathsf{ann}), \mathfrak{O})$ *returns whether* (P, ann) *is fully, partially or not compliant with* $\mathfrak{O}$*.*

Algorithm A_5

Let $\mathfrak{O} = \mathcal{O}^x\langle c, l, d\rangle$
Label α *the tasks in* P *where* $l \in \mathsf{ann}(t)$*;*
Label β *the tasks in* P *where* $d \in \mathsf{ann}(t)$ *and the pseudo-task* end*;*
Label γ *the tasks in* P *where* $c \in \mathsf{ann}(t)$*;*
$B = CL(B)$*;*
$B = PL(B)$*;*
if $x == a$ ***then***
  if $\exists$ *task* $\in B$ *labeled:* $\alpha \wedge \beta \wedge \neg(\gamma \vee \gamma_c)$ ***then***
    return $(P, \mathsf{ann}) \nvdash \mathfrak{O}$
  else
    if $(\exists$ *task* $\in B$ *labeled:* $(\alpha \vee \alpha_c) \wedge (\beta \vee \beta_c) \wedge \neg(\gamma \vee \gamma_c)$ ***then***
      return $(P, \mathsf{ann}) \vdash^P \mathfrak{O}$
    else
      return $(P, \mathsf{ann}) \vdash^F \mathfrak{O}$
    end if
  end if
else
  if $\exists$ *task* $\in B$ *labeled:* $\alpha \wedge \neg(\beta \vee \beta_c) \wedge \gamma$ ***then***
    return $(P, \mathsf{ann}) \nvdash \mathfrak{O}$
  else
    if $(\exists$ *task* $\in B$ *labeled:* $(\alpha \vee \alpha_c) \wedge \neg(\beta \vee \beta_c) \wedge (\gamma \vee \gamma_c)$ ***then***
      return $(P, \mathsf{ann}) \vdash^P \mathfrak{O}$

23: ***else***
24: ***return*** $(P, \mathsf{ann}) \vdash^{F} \mathbf{O}$
25: ***end if***
26: ***end if***
27: ***end if***

Correctness

Correctness Proof of Algorithm 7. Given an annotated process (P, ann), where $P = \mathsf{start}\ B\ \mathsf{end}$, and a global achievement obligation $O^a(l)$, I prove the correctness of Algorithm 1 by showing both soundness and completeness.

Soundness I prove the soundness of Algorithm 7 by cases considering the three possible compliance results that the algorithm can return, and showing that for each result, the process given in input belongs to such compliance class.

1. Algorithm 7 returns $(P, \mathsf{ann}) \nvdash \mathbf{O}$
   (a) Assume that $\mathbf{O}$ is an achievement obligation.
      i. From the hypothesis and (a), it follows that the condition at line 8 of the algorithm is true.
      ii. From i, Function 1 and Function 2, it follows that each execution of a task with such labelling leads to a state containing $\{l, d\}$ and not containing $\{c\}$.
      iii. From ii and Function 2, it follows that there are no traces avoiding a state containing $\{l, d\}$ and not containing $\{c\}$.
      iv. From iii and Definition 20, it follows that $\forall \theta \in \Theta(P, \mathsf{ann}), \theta \nvdash \mathbf{O}$.
      v. From iv and Definition 16, it follows that (P, ann) is not compliant with $\mathbf{O}$.
   (b) Assume that $\mathbf{O}$ is a prohibition.
      i. From the hypothesis and (b), it follows that the condition at line 18 of the algorithm is true.
      ii. From i, Function 1 and Function 2, it follows that each execution of a task with such labelling leads to a state containing $\{l, c\}$ and not containing $\{d\}$.
      iii. From ii and Function 2, it follows that there are no traces avoiding a state containing $\{l, c\}$ and not containing $\{d\}$.
      iv. From iii and Definition 50, it follows that $\forall \theta \in \Theta(P, \mathsf{ann}), \theta \nvdash \mathbf{O}$.
      v. From iv and Definition 16, it follows that (P, ann) is not compliant with $\mathbf{O}$.
2. Algorithm 7 returns $(P, \mathsf{ann}) \vdash^{\mathrm{P}} \mathbf{O}$
   (a) Assume that $\mathbf{O}$ is an achievement obligation.

i. From the hypothesis and (a), it follows that the condition at line 8 of the algorithm is false and the one at line 11 is true.

ii. From i, Function 1 and Function 2, it follows that some executions of a task with such labelling leads to a state containing $\{l, d\}$ and not containing $\{c\}$.

iii. From ii and Function 2, it follows that there are some traces with state containing $\{l, d\}$ and not containing $\{c\}$, and some which does not.

iv. From iii and Definition 20, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \nvdash \mathbf{O}$ and $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \mathbf{O}$.

v. From iv and Definition 16, it follows that (P, ann) is partially compliant with $\mathbf{O}$.

(b) Assume that $\mathbf{O}$ is a prohibition.

i. From the hypothesis and (b), it follows that the condition at line 18 of the algorithm is false and the one at line 21 is true.

ii. From i, Function 1 and Function 2, it follows that some executions of a task with such labelling leads to a state containing $\{l, c\}$ and not containing $\{d\}$.

iii. From ii and Function 2, it follows that there are some traces with state containing $\{l, c\}$ and not containing $\{d\}$, and some which does not.

iv. From iii and Definition 50, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \nvdash \mathbf{O}$ and $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \mathbf{O}$.

v. From iv and Definition 16, it follows that (P, ann) is partially compliant with $\mathbf{O}$.

3. Algorithm 7 returns $(P, \mathsf{ann}) \vdash^{\mathrm{F}} \mathbf{O}$

(a) Assume that $\mathbf{O}$ is an achievement obligation.

i. From the hypothesis and (a), it follows that the conditions at line 8 and at line 11 are both false.

ii. From i, Function 1 and Function 2, it follows that no execution of a task with such labelling leads to a state containing $\{l, d\}$ and not containing $\{c\}$.

iii. From ii and Function 2, it follows that there are no traces with a state containing $\{l, d\}$ and not containing $\{c\}$.

iv. From iii and Definition 20, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \mathbf{O}$.

v. From iv and Definition 16, it follows that (P, ann) is fully compliant with $\mathbf{O}$.

(b) Assume that $\mathbf{O}$ is a prohibition.

i. From the hypothesis and (b), it follows that the conditions at line 18 and at line 21 are both false.

ii. From i, Function 1 and Function 2, it follows that no execution of a task with such labelling leads to a state containing $\{l, c\}$ and not containing $\{d\}$.

iii. From *ii* and Function 2, it follows that there are no traces with a state containing $\{l, c\}$ and not containing $\{d\}$.
iv. From *iii* and Definition 50, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \mathbf{O}$.
v. From *iv* and Definition 16, it follows that (P, ann) is fully compliant with $\mathbf{O}$.

I have shown the soundness of Algorithm 7 by showing that in each of the three possible cases, whether the algorithm returns fully, partially or not compliant, then the process model given as input is correctly classified.

Completeness I prove by cases the completeness of Algorithm 1, by showing that for each of the three compliance classes to which a process can belong, the result of the algorithm is indeed corresponding to such class.

1. (P, ann) is not compliant with $\mathbf{O}$
   (a) Assume that $\mathbf{O}$ is an achievement obligation.
      i. From the hypothesis, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \nvdash \mathbf{O}$.
      ii. From (a), *i*, Definition 20 and Definition 48, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \exists\sigma \in \theta$ such that $\sigma \models l, \sigma \models d$ and $\sigma \not\models c$.
      iii. From *ii*, Function 1 and Function 2, it follows that exists a task labeled $\alpha \wedge \beta \wedge \neg(\gamma \vee \gamma_c)$.
      iv. From *iii* and (a), it follows that the condition at line 8 is satisfied.
      v. From *iv*, it follows that the algorithm returns $(P, \mathsf{ann}) \nvdash \mathbf{O}$.
   (b) Assume that $\mathbf{O}$ is a prohibition.
      i. From the hypothesis, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \nvdash \mathbf{O}$.
      ii. From (b), *i*, Definition 50 and Definition 48, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \exists\sigma \in \theta$ such that $\sigma \models l, \sigma \models c$ and $\sigma \not\models d$.
      iii. From *ii*, Function 1 and Function 2, it follows that exists a task labeled $\alpha \wedge \neg(\beta \vee \beta_c) \wedge \gamma$.
      iv. From *iii* and (b), it follows that the condition at line 18 is satisfied.
      v. From *iv*, it follows that the algorithm returns $(P, \mathsf{ann}) \nvdash \mathbf{O}$.

2. (P, ann) is partially compliant with $\mathbf{O}$
   (a) Assume that $\mathbf{O}$ is an achievement obligation.
      i. From the hypothesis, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \mathbf{O}$.
      ii. From (a), *i*, Definition 20 and Definition 48, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \neg\exists\sigma \in \theta$ such that $\sigma \models l, \sigma \models d$ and $\sigma \not\models c$.
      iii. From the hypothesis, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \nvdash \mathbf{O}$.

iv. From (a), iii, Definition 20 and Definition 48, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \exists\sigma \in \theta$ such that $\sigma \models l, \sigma \models d$ and $\sigma \not\models c$.

v. From ii, Function 1 and Function 2, it follows that does not exist a task labeled $\alpha \wedge \beta \wedge \neg(\gamma \vee \gamma_c)$.

vi. From v and (a), it follows that the condition at line 8 is not satisfied.

vii. From ii, iv, Function 1 and Function 2, it follows that exists a task labeled $(\alpha \vee \alpha_c) \wedge (\beta \vee \beta_c) \wedge \neg(\gamma \vee \gamma_c)$.

viii. From vii, it follows that the condition at line 11 is satisfied.

ix. From vi and $viii$, it follows that the algorithm returns $(P, \mathsf{ann}) \vdash^{\mathrm{P}} \mathbf{O}$.

(b) Assume that $\mathbf{O}$ is a prohibition.

i. From the hypothesis, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \mathbf{O}$.

ii. From (b), i, Definition 50 and Definition 48, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \exists\sigma \in \theta$ such that $\sigma \models l, \sigma \models d$ and $\sigma \not\models c$.

iii. From the hypothesis, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \theta \not\vdash \mathbf{O}$.

iv. From (b), iii, Definition 50 and Definition 48, it follows that $\exists\theta \in \Theta(P, \mathsf{ann}), \neg\exists\sigma \in \theta$ such that $\sigma \models l, \sigma \models d$ and $\sigma \not\models c$.

v. From ii, Function 1 and Function 2, it follows that does not exist a task labeled $\alpha \wedge \neg(\beta \vee \beta_c) \wedge \gamma$.

vi. From v and (b), it follows that the condition at line 18 is not satisfied.

vii. From ii, iv, Function 1 and Function 2, it follows that exists a task labeled $(\alpha \vee \alpha_c) \wedge \neg(\beta \vee \beta_c) \wedge (\gamma \vee \gamma_c)$.

viii. From vii, it follows that the condition at line 21 is satisfied.

ix. From vi and $viii$, it follows that the algorithm returns $(P, \mathsf{ann}) \vdash^{\mathrm{P}} \mathbf{O}$.

3. (P, ann) is fully compliant with $\mathbf{O}$

(a) Assume that $\mathbf{O}$ is an achievement obligation.

i. From the hypothesis, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \mathbf{O}$.

ii. From (a), i, Definition 20 and Definition 48, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \neg\exists\sigma \in \theta$ such that $\sigma \models l, \sigma \models d$ and $\sigma \not\models c$.

iii. From ii, Function 1 and Function 2, it follows that a task labeled $(\alpha \vee \alpha_c) \wedge (\beta \vee \beta_c) \wedge \neg(\gamma \vee \gamma_c)$ does not exist.

iv. From iii and (a), it follows that the conditions at line 8 and line 11 are not satisfied.

v. From iv, it follows that the algorithm returns $(P, \mathsf{ann}) \vdash^{\mathrm{F}} \mathbf{O}$.

(b) Assume that $\mathbf{O}$ is a prohibition.

i. From the hypothesis, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \theta \vdash \mathbf{O}$.

ii. From (b), i, Definition 50 and Definition 48, it follows that $\forall\theta \in \Theta(P, \mathsf{ann}), \neg\exists\sigma \in \theta$ such that $\sigma \models l, \sigma \models c$ and $\sigma \not\models d$.

iii. From ii, Function 1 and Function 2, it follows that a task labeled $(\alpha \vee \alpha_c) \wedge \neg(\beta \vee \beta_c) \wedge (\gamma \vee \gamma_c)$ does not exist.

iv. From iii and (a), it follows that the conditions at line 18 and line 21 are not satisfied.

v. From iv, it follows that the algorithm returns $(P, \mathsf{ann}) \vdash^{\mathrm{F}} \mathbf{\Theta}$.

I thus have shown that the algorithm returns its outcome in accordance to the compliance class to which the process given in input belongs to.

Having proven both soundness and completeness of Algorithm 7, I can conclude that the algorithm is correct. □

Bibliography

[1] 107th Congress Public Law 204. Sarbanes-oxley act of 2002. corporate responsibility. Legislative Act, 2002. `http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/html/PLAW-107publ204.htm`.

[2] 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, UK, 1997. Springer-Verlag.

[3] Rafael Accorsi, Lutz Lowis, and Yoshinori Sato. Automated certification for compliant cloud-based business processes. *Business & Information Systems Engineering*, 3(3):145–154, 2011.

[4] 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.

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

[6] Anupriya Ankolekar, Mark Burstein, JerryR. Hobbs, Ora Lassila, David Martin, Drew McDermott, SheilaA. McIlraith, Srini Narayanan, Massimo Paolucci, Terry Payne, and Katia Sycara. Daml-s: Web service description for the semantic web. In Ian Horrocks and James Hendler, editors, *The Semantic Web ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, pages 348–363. Springer Berlin Heidelberg, 2002.

[7] Guillaume Aucher, Guido Boella, and Leendert van der Torre. A dynamic logic for privacy compliance. *Artif. Intell. Law*, 19(2-3):187–231, 2011.

[8] Ahmed Awad. BPMN-Q: A language to query business processes. In Manfred Reichert, Stefan Strecker, and Klaus Turowski, editors, *EMISA*, volume P-119 of *LNI*, pages 115–128, St. Goar, Germany, 2007. GI.

[9] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using bpmn-q and temporal logic. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2008.

[10] Ahmed Awad, Matthias Weidlich, and Mathias Weske. Visually specifying compliance rules and explaining their violations for business processes. *J. Vis. Lang. Comput.*, 22(1):30–55, February 2011.

[11] John Bace, Carol Rozwell, Joseph Feiman, and Bill Kirwin. Understanding the costs of compliance. *Gartner Research*, 7, 2006.

[12] Basel Committee on Banking Supervision. Group of governors and heads of supervision announces higher global minimum capital standards. Press Release, September 2010. `http://www.bis.org/press/p100912.pdf`.

[13] Mathieu Beirlaen and Christian Straßer. A paraconsistent multi-agent framework for dealing with normative conflicts. In João Leite, Paolo Torroni, Thomas Ågotnes, Guido Boella, and Leon van der Torre, editors, *CLIMA*, volume 6814 of *Lecture Notes in Computer Science*, pages 312–329. Springer, 2011.

[14] Mathieu Beirlaen, Christian Straßer, and Joke Meheus. An inconsistency-adaptive deontic logic for normative conflicts. *J. Philosophical Logic*, 42(2):285–315, 2013.

[15] Guido Boella, Marijn Janssen, Joris Hulstijn, Llio Humphreys, and Leendert van der Torre. Managing legal interpretation in regulatory compliance. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law*, ICAIL '13, pages 23–32, New York, NY, USA, 2013. ACM.

[16] Guido Boella and Leendert W. N. van der Torre. Permissions and obligations in hierarchical normative systems. In *ICAIL*, pages 109–118, 2003.

[17] Guido Boella and Leendert W.N. van der Torre. A game-theoretic approach to normative multi-agent systems. In *Normative Multi-agent Systems*, 2007.

[18] Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni. Representing and monitoring social commitments using the event calculus. *Autonomous Agents and Multi-Agent Systems*, 27(1):85–130, 2013.

[19] Manuel Clavel, Francisco Durn, Steven Eker, Patrick Lincoln, Narciso Mart-Oliet, Jos Meseguer, and Carolyn Talcott. Ltl model checking. In *All About Maude - A High-Performance Logical Framework*, volume 4350 of *Lecture Notes in Computer Science*, pages 385–418. Springer Berlin Heidelberg, 2007.

[20] Silvano Colombo Tosatto, Guido Boella, Leendert van der Torre, and Serena Villata. Abstract normative systems: Semantics and proof theory. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 358–368, 2012.

[21] Silvano Colombo Tosatto, Guido Boella, Leon van der Torre, and Serena Villata. Visualizing normative systems: an abstract approach. In *Deontic Logic in Computer Science - 11th International Conference, DEON 2012*, pages 16–30, 2012.

[22] Silvano Colombo Tosatto, Guido Governatori, and Pierre Kelsen. Towards an abstract framework for compliance. In *EDOC 2013*, pages 79–88, Vancouver, sep 2013. IEEE.

[23] Silvano Colombo Tosatto, Guido Governatori, and Pierre Kelsen. Detecting deontic conflicts in dynamic settings. In Fabrizio Cariani, Davide Grossi, Joke Meheus, and Xavier Parent, editors, *Deontic Logic and Normative Systems*, volume 8554 of *Lecture Notes in Computer Science*, pages 65–80. Springer International Publishing, 2014.

[24] Silvano Colombo Tosatto, Pierre Kelsen, Qin Ma, Marwane el Kharbili, Guido Governatori, and Leendert van der Torre. Algorithms for tractable compliance problems. *Frontiers of Computer Science*, pages 1–20, 2014.

[25] Natalia Criado, Estefania Argente, and Vicente J. Botti. Open issues for normative multi-agent systems. *AI Commun.*, 24(3):233–264, 2011.

[26] H. William Dettmer. *Goldratts Theory of Constraints: a systems approach to continuous improvement*. ASQC Quality Press, 1997.

[27] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 411–420, New York, NY, USA, 1999. ACM.

[28] Amal Elgammal, Oktay Türetken, Willem-Jan van den Heuvel, and Mike P. Papazoglou. Root-cause analysis of design-time compliance violations on the basis of property patterns. In Paul P. Maglio, Mathias Weske, Jian Yang, and Marcelo Fantinato, editors, *Service-Oriented Computing - 8th International Conference, ICSOC 2010, San Francisco, CA, USA, December 7-10, 2010. Proceedings*, volume 6470 of *Lecture Notes in Computer Science*, pages 17–31, 2010.

[29] Abdullatif A.O. Elhag, Joost A. Breuker, and Bob W. Brouwer. On the formal analysis of normative conflicts. In H.Jaap van den Herik et al., editor, *JURIX 1999: The Twelfth Annual Conference*, Frontiers in Artificial Intelligence and Applications, pages 35–46, Nijmegen, 1999. GNI.

[30] Aditya Ghose and George Koliadis. Auditing business process compliance. In Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors, *ICSOC*, volume 4749 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2007.

[31] Stijn Goedertier and Jan Vanthienen. Designing compliant business processes with obligations and permissions. In *Business Process Management (BPM) Workshops*, pages 5–14, 2006.

[32] Guido Governatori. ICT support for regulatory compliance of business processes. In *29th World Continuous Auditing and Reporting Symposium*, pages 1–10, Brisbane, Australia, nov 2013.

[33] Guido Governatori, Jörg Hoffmann, Shazia Wasim Sadiq, and Ingo Weber. Detecting regulatory compliance for business process models through semantic annotations. In Danilo Ardagna, Massimo Mecella, and Jian Yang, editors, *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 5–17. Springer, 2008.

[34] Guido Governatori, Joris Hulstijn, Régis Riveret, and Antonino Rotolo. Characterising deadlines in temporal modal defeasible logic. In Mehmet A. Orgun and John Thornton, editors, *Australian Conference on Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 486–496. Springer, 2007.

[35] Guido Governatori, Zoran Milosevic, and Shazia Sadiq. Compliance checking between business processes and business contracts. In Patrick C. K. Hung, editor, *10th International Enterprise Distributed Object Computing Conference (EDOC 2006)*, pages 221–232. IEEE, 2006.

[36] 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.

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

[38] Guido Governatori and Shazia Sadiq. The journey to business process compliance. In Jorge Cardoso and Wil van der Aalst, editors, *Handbook of Research on BPM*, chapter 20, pages 426–454. IGI Global, 2009.

[39] Daniela Grigori, Fabio Casati, Malu Castellanos, Umeshwar Dayal, Mehmet Sayal, and Ming-Chien Shan. Business process intelligence. *Computers in Industry*, 53(3):321 – 343, 2004. Process / Workflow Mining.

[40] Jörg Hansen. Conflicting imperatives and dyadic deontic logic. In A. Lomuscio and D. Nute, editors, *Proceedings of the 7th International Workshop on Deontic Logic in Computer Science (DEON 2004), Madeira, Portugal, May 26-28, 2004*, number 3065/2004 in LNCS, pages 146–164. Springer, 2004.

[41] Jorg Hansen. *Outstanding Contributions to Logic*, chapter Reasoning About Permission and Obligation. In Sven Ove Hansson and Wansing [71], 2014.

[42] Bengt Hansson. An analysis of some deontic logics. *Nôus*, 3:373–398, 1969.

[43] Risto Hilpinen and Paul McNamara. Deontic logic: a historical survey and introduction. In Dov Gabbay, John Horty, Ron van der Meyden, Xavier Parent, and Leendert van der Torre, editors, *Handbook of Deontic Logic and Normative Systems*. College Publications, 2013.

[44] 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.

[45] Christopher D. Hollander and Annie S. Wu. The current state of normative agent-based systems. *Journal of Artificial Societies and Social Simulation*, 14(2):6, 2011.

[46] Andrew J. I. Jones and Marek Sergot. On the characterisation of law and computer systems: The normative systems perspective. In *Deontic Logic in Computer Science: Normative System Specification*, pages 275–307. John Wiley & Sons, 1993.

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

[48] M.E. Kharbili. Business process regulatory compliance management solution frameworks: A comparative evaluation. In *Asia-Pacific Conference on Conceptual Modelling (APCCM 2012)*, volume 130 of *CRPIT*, pages 23–32. ACS, 2012.

[49] 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.

[50] E. J. Lemmon. Moral dilemmas. *The Philosophical Review*, 71(2):139–158, April 1962.

[51] Alessio Lomuscio and Marek Sergot. Violation, error recovery, and enforcement in the bit transmission problem. In John Horty and Andrew J. I. Jones, editors, *Proceedings of the 6th International Workshop on Deontic Logic in Computer Science (DEON 2002), London, England, May 22-24, 2002*, pages 181–202. Imperial College London, Informal Proceedings, 2002.

[52] Carsten Lutz and Ulrike Sattler. A proposal for describing services with dls. In *In Proc. of DL 2002*, pages 128–139, 2002.

[53] Linh Thao Ly, David Knuplesch, Stefanie Rinderle-Ma, Kevin Göser, Manfred Reichert, and Peter Dadam. Seaflows toolset - compliance verification made easy. In *CAiSE'10 Forum*, June 2010.

[54] Linh Thao Ly, Fabrizio Maria Maggi, Marco Montali, Stefanie Rinderle-Ma, and Wil M. P. van der Aalst. A framework for the systematic comparison and evaluation of compliance monitoring approaches. In *EDOC*, pages 7–16, 2013.

[55] David Makinson and Leendert van der Torre. Permission from an input/output perspective. *J. Philosophical Logic*, 32(4):391–416, 2003.

[56] David Makinson and Leendert W. N. van der Torre. What is input/output logic? input/output logic, constraints, permissions. In Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen, editors, *Normative Multi-agent Systems*, volume 07122 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[57] Marco Montali, Diego Calvanese, and Giuseppe De Giacomo. Verification of data-aware commitment-based multiagent system. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 157–164, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[58] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

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

[60] Henry Prakken and Giovanni Sartor. A dialectical model of assessing conflicting arguments in legal reasoning. *Artif. Intell. Law*, 4(3-4):331–368, 1996.

[61] Elham Ramezani, Dirk Fahland, and WilM.P. van der Aalst. Where did I misbehave? diagnostic information in compliance checking. In Alistair Barros, Avigdor Gal, and Ekkart Kindler, editors, *Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 262–278. Springer Berlin Heidelberg, 2012.

[62] António Rito Silva. A blended workflow approach. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 25–36. Springer Berlin Heidelberg, 2012.

[63] Dumitru Roman and Michael Kifer. Reasoning about the behaviour of semantic web services with concurrent transaction logic. In *VLDB*, pages 627–638, 2007.

[64] Shazia Sadiq and Guido Governatori. Managing regulatory compliance in business processes. In Jan van Brocke and Michael Rosemann, editors, *Handbook of Business Process Management*, volume 2, chapter 8, pages 157–173. Springer, Berlin, 2010.

[65] Shazia Sadiq, Guido Governatori, and Kioumars Namiri. Modeling control objectives for business process compliance. In *Proceedings of the 5th International Conference on Business Process Management*, BPM'07, pages 149–164, Berlin, Heidelberg, 2007. Springer-Verlag.

[66] Giovanni Sartor. Normative conflicts in legal reasoning. *Artificial Intelligence and Law*, 1(2-3):209–235, 1992.

[67] Krister Segerberg. D delta l : a dynamic deontic logic. *Synthese*, 185:1–17, 2012.

[68] Munindar P. Singh. Formalizing communication protocols for multiagent systems. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 1519–1524, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[69] Colin Stirling. *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer, 2001.

[70] Audun Stolpe. *Outstanding Contributions to Logic*, chapter Abstract Interfaces of Input/Output Logic. In Sven Ove Hansson and Wansing [71], 2014.

[71] Lawrence Moss Sonja Smets Sven Ove Hansson, Marcus Kracht and Heinrich Wansing, editors. *Outstanding Contributions to Logic*. Springer Netherlands, Dordrecht, 2014.

[72] Pankaj R. Telang and Munindar P. Singh. Comma: a commitment-based business modeling methodology and its empirical evaluation. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 1073–1080, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[73] Silvano Colombo Tosatto, Yehia Elrakaiby, and Pouyan Ziafati. Compliance in resource-based process models. *The 25th Benelux Conference on Artificial Intelligence (BNAIC 2013)*, 2013.

[74] Silvano Colombo Tosatto, Guido Governatori, and Pierre Kelsen. Business process regulatory compliance is hard. *IEEE Transactions on Services Computing*, 99(PrePrints):1, 2014.

[75] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of workflow nets: Classification, decidability, and analysis. *Form. Asp. Comput.*, 23(3):333–363, May 2011.

[76] Wil van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures*, volume 06291 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[77] 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.

[78] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.

[79] W.M.P. van der Aalst. Interorganizational workflows: An approach based on message sequence charts and petri nets. In *Systems Analysis Modelling Simulation*, volume 34, pages 335–367, 1999.

[80] WambertoW. Vasconcelos, MartinJ. Kollingbaum, and TimothyJ. Norman. Normative conflict resolution in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 19(2):124–152, 2009.

[81] Georg Henrik Von Wright. Deontic logic. *Mind*, 60(237):1–15, 1951.

[82] Ingo Weber, Jrg Hoffmann, and Jan Mendling. Semantic business process validation. In *Proceedings of the 3rd International Workshop on Semantic Business Process Management (SBPM'08)*, 2008.

[83] Moe Thandar Wynn, Marlon Dumas, ColinJ. Fidge, ArthurH.M. ter Hofstede, and WilM.P. van der Aalst. Business process simulation for operational decision support. In Arthur ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors, *Business Process Management Workshops*, volume 4928 of *Lecture Notes in Computer Science*, pages 66–77. Springer Berlin Heidelberg, 2008.

[84] Fabiola Lopez y Lopez, Michael Luck, Mark d'Inverno, and Southampton Uk. Constraining autonomy through norms, 2002.