On the equivalence of Defeasible Deontic Logic and Temporal Defeasible Logic

Marc Allaire and Guido Governatori

NICTA Queensland, Brisbane, Australia*

Abstract. In this paper we formally prove that compliance results derived from temporal defeasible logic are equivalent to the ones obtained in the standard defeasible deontic logic. In order to do so we first introduce an operator allowing us to translate rules from the standard to the temporal framework. Then we consider the sets of obligations used in the compliance checking algorithm from [19] and prove that they are isomorphic to the previously defined operator. Being able to add time to standard deontic logic will allow for a better and more elegant representation of obligations and improvement in computational efficiency.

1 Introduction

An important aspect of Normative Multi-Agent Systems (NorMAS) is to study whether the behaviour of an agent complies with the regulations governing the environment in which the agents is situated.

According to standard agent architectures (for example, the seminal BDI architecture) an agent is equipped with a plan library, and after the deliberation phase (i.e., the phase in the life-cycle of an agent when the agent identifies what goals the agent commits to), the agent select which plan has to be executed to reach the goals. An agent plan has to be understood as in classical AI, where a plan is just a sequence of actions or tasks, where every task can be associate to its pre-conditions and post-conditions of effects [14].

The notion of compliance has been investigated in the field of business processes [22, 39]. A business process model is a self-contained, temporal and logical order in which a set of activities are executed to achieve a business goal. Typically a process model describes what needs to be done and when (control flow), who is going to do what (resources), and on what it is working on (data). The combination of tasks and connectors defines the possible ways in which a process can be executed. Where a possible execution, called *process trace* or simply *trace*, is a sequence of tasks respecting the order given by the connectors. In this perspective a trace is isomorphic to a plan, thus a business process can be understood as a set of traces/plan, or in other words a business process is the plan library an organisation has to achieve a particular business objective.

Governatori and Rotolo [19] propose to use PCL (Process Compliance Logic), an extension of defeasible logic with deontic operators including an operator to handle the combination of violations and compensations [18] for the modelling of norms in a business process compliance point of view. The deontic operators offer a rich conceptual

* NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

model of obligations able to capture all type of obligations when one consider their validity and compliance aspects over a sequence of tasks or a timelime. Furthermore, they propose a linear time algorithm to check whether a process trace complies with a regulation (modelled in PCL). The proposed model has been implemented and successfully validated with industry scale case studies [23]. The obligation types captured by the deontic operators have an essential temporal nature, thus [21] presented a temporalised version of the logic. An important features of the resulting logic is that it can check whether a given theory is compliant using only the proof theory of the logic itself without any external algorithm.

The aim of this paper is to study the relationships between the notions of compliance proposed in [19] and [21]. We are going to show that the temporal model of [21] is able to simulate that of [19].

The structure of the paper is as follows: in the next section we outline the basics of defeasible logic and deontic defeasible logic (PCL). In Section 3 we outline the idea of business process compliance, and in Section 4 we recall the linear time algorithm presented in [19] to check if a trace is compliant with a set of norms. In the next section we outline the temporal defeasible deontic logic of [21]. Section 6is dedicated to establish the relationships between the two logic. In Section 7 we summarise the results and shortly discuss related work.

2 Defeasible Deontic Logic

Defeasible reasoning as presented in [31] is a non-monotonic type of reasoning where one cannot reach a full, undoubted conclusion because any conclusion can always be defeated if further evidence of the contrary is demonstrated. Both computer scientists and philosophers have shown an interest in this field. The philosophical interest can be traced back to ancient Greece and Aristotle. Although the scientific reasoning is built on deductive logic, for everyday life we rely mainly on defeasible reasoning. We try to make general statements out of personal experience, for example we could say that all birds fly. This proposition would be true until we experience a bird that cannot fly such as a penguin. This would defeat the first rule.

Computer science interest in defeasible logic has grown during the last 40 years especially in the field of artificial intelligence. An intelligent program needs a formal representation of the world, a formal language to represent knowledge, causality and ability in order to achieve its given goal. This requirement was first described in [35].

Defeasible Logic was first introduced by Donal Nute in [37] as a formalism to represent defeasible reasoning in a logical way. As stated in [4] defeasible logic is a flexible non-monotonic formalism able to represent a large set of non-monotonic reasoning intuitions. Several powerful implementations have been proposed with good complexity properties allowing a feasible computational time. This has been made possible by the design of defeasible logic that makes implementation easy yet efficient.

Let us introduce the basics of Defeasible Logic as given in [3]. A defeasible theory gives us five different ways of representing knowledge *facts*, *strict rules*, *defeasible rules*, *defeaters* and a *superiority relation*.

Facts are indisputable statements for example Tux is a penguin which could be written formally as $penguin(Tux)$

Strict rules are rules as in classical logic. It is the kind of rule we find in scientific reasoning. The conclusion is irrefutable if the premises also are. These are formally represented as:

$$penguin(X) \rightarrow bird(X)$$

Defeasible rules are rules that can be defeated by evidence of the contrary. To draw a parallel with everyday reasoning one could generalize from experience that "Birds fly" a statement that would be true until the opposite is derived. These rules are formally represented as:

$$bird(X) \Rightarrow flies(X)$$

Defeaters are weaker rules that cannot be used to draw any conclusion but can prevent one. They are used to defeat other rules (hence the name) because they produce evidence of the contrary.

$$heavy(X) \rightsquigarrow \neg flies(X)$$

From this defeater we cannot conclude that because someone or something is heavy it cannot fly, it is only here to prevent the conclusion of $flies(X)$.

Superiority relation is used to create an order in a rule set. It is important to note that this relation does not have the properties of a proper order relation, it is not transitive. When we have two different rules which derive something and its negation we cannot draw a conclusion since defeasible logic is sceptical. The superiority relation allows us to come to a conclusion. For example:

$$\begin{aligned} r: &\quad bird(X) \Rightarrow flies(X) \\ r': &\quad brokenWing(X) \Rightarrow \neg flies(X) \\ &\quad r' > r \end{aligned}$$

In this case we cannot reach a conclusion since r and r' reach opposite conclusions. By introducing the superiority relation we say that r' is strictly stronger than r and therefore we can conclude that the bird cannot fly.

Now that we are more familiar with the concepts of defeasible logic we can show how we can reach a defeasible conclusion using its proof theory. Four proof types for a conclusion have been defined. Given a Defeasible Theory D

$+\Delta q$ means that the literal q is definitely provable in D;
$-\Delta q$ means that the literal q is definitely refuted in D;
$+\partial q$ means that the literal q is defeasibly provable in D
$-\partial q$ means that the literal $q$is defeasibly refuted in D.

In [3] provability is defined using the concept of *derivation* of a conclusion from a Defeasible Theory D. Where, formally a Defeasible Theory D is a triple $(F,R,>)$ of set of facts F, set of rules R and superiority relation $>$. A derivation P can be seen as several steps in a demonstration. At step $P(i)$ of the proof we have a given set $(F,R,>)$ from this we can prove $P(i+1)$ either definitely or defeasibly. In what follows we restrict our

attention to the propositional variant of the logic built on a set of atomic propositions (the examples given so far can be consider as schema corresponding to the set of all their ground instances). A literal is either an atomic proposition or its negation. Given a literal l, we use $\sim l$ to denote its complement, that is $\neg p$ if $l = p$, where p is an atomic proposition, and p if $l = \neg p$. A rule is an expression

$$r\colon A(r) \hookrightarrow C(r)$$

where r is the label of the rule; $A(r)$, the antecedent of the rule, is a (possibly empty) t set of literals; $C(r)$, the conclusion of the rule, is a literal; and $\hookrightarrow \in \{\rightarrow, \Rightarrow, \rightsquigarrow\}$. Given a set of rules R we use $R[q]$ to denote the set of rules whose head is q, i.e., $C(r) = q$, R_s is the set of the strict rules in R, and R_{sd} is the subset of strict and defeasible rules in R.

To reach a definitive conclusion $+\Delta q$ we need to have a strict rule that deduces q or have q as a fact. We will not get into details about how to definitely prove a literal since the focus of this article is more on defeasible proofs. We refer the reader to [3] for the full details.

The following definition exposes how to defeasibly conclude a literal q at $P(i+1)$.

Definition 1 *If $P(i+1) = +\partial q$ then either*

1. $+\Delta q \in P(1..i)$ *or*
2. *(a)* $\exists r \in R_{sd}[q], \forall a \in A(r) : +\partial a \in P(1..i)$ *and,*
   (b) $-\Delta{\sim}q \in P(1..i)$ *and,*
   (c) $\forall s \in R[{\sim}q]$ *either*
      i. $\exists a \in A(s) : -\partial a \in P(1..i)$ *or*
      ii. $\exists t \in R[q]$ *such that* $\forall a \in A(t) : +\partial a \in P(1..i)$ *and* $t > s$

In less mathematical terms this definition means that in order to defeasibly prove q we can follow two paths. Either prove that q is definitely provable or work the defeasible part. For the defeasible path, three conditions apply:

1. there is a rule r that concludes q for a set of literals $A(r)$ such as every $a \in A(r)$ has been defeasibly proven in a previous step $P(1..i)$ and,
2. ${\sim}q$ has not been definitively proven in a previous step $P(1..i)$ and,
3. for every rule s that conclude $\neg q$ for a literal a either
   (a) a has been defeasibly refuted in a previous step $P(1..i)$ or
   (b) there is an applicable rule t that concludes q such as $t > s$

2.1 Using Defeasible Deontic Logic to Represent Legal Norms and Regulations

Regulations and legal norms are an important concern for government and businesses. They are complex and hard to study especially when multiple regulations written separately are applied to a given situation. In a world where compliance to regulation is becoming both harder and more important because of their growing number and the sanctions applied for non compliance, a normative logical framework is needed to be able to reason about regulations.

Deontic logic is the branch of symbolic logic concerned with the logic of obligations and permissions. Therefore it is exactly the kind of logical framework we want to be

able to express regulations. Unfortunatel, y standard deontic logic is unable to represent simple notions of normative reasoning such as prima-facie obligations or contrary-to-duty obligations [36]. This lack of expressibility has driven away the very people that would have used deontic logic the most [38].

Let us take a closer look at prima-facie obligation for example and see how we can express these in the light of defeasible logic. Prima-facie means "at first sight" hence a prima-facie obligation is an obligation that stands at first sight, one that can be defeated if new facts can prove otherwise. We can see that this type of obligation can easily be expressed using defeasible logic, it is defeasible by definition. Furthermore regulations contain exceptions that are easily represented using defeasible logic.

There are many benefits to use a logical framework to represent regulations, some of those are presented in [2]. They are subdivided into two main area of application:

- **The understanding and application of regulations** for agents not familiar with legal writing and do not want to study a regulation yet being under the obligation to comply.
  Decision support: If an agent take a decision, is the agent compliant? The agent can run its process against a set of regulations and see if it complies. A formal framework for expressing processes and norms is needed too in this case.
  Explanation: The agent can examine the complete reasoning chain that lead to the given answer. It is therefore easier for the agent to understand what cause this answer for their request or what caused non-compliance.
- **The creation of regulation** for assisting legal professional in their work.
  Anomaly detection Having a formal logical framework backing the drafting of regulation allows for an easy detection of anomalies such as inconsistencies or loops.
  Hypothetical reasoning It is possible, like for decision support, to inspect the effects of a regulation on the entire system.
  Debugging When a regulation is not yielding the expected answer to a given query it is possible to debug it.

Now that we explained the need for a logical framework for legal reasoning and how good deontic defeasible logic is we can introduce the different types of obligations. Indeed to accurately represent the complexity of norms and regulations it is necessary to have a range of different types of obligations to be able to translate legal text into an equivalent logical representation as explained in [19]. There are three main types of obligations [16]:

- **Achievement Obligation**: There is an obligation to meet once before the deadline. For example *You must change your tires before they are worn out*
- **Maintenance Obligation**: There is an obligation to meet at all instant before the deadline. For example *You must provide for your children until they are 18*
- **Punctual Obligation**: There is an obligation to meet at one instant. They must be fulfilled at the same moment they were triggered.

We will use the modal operators O^a, O^m and O^p for, respectively, achievement, maintenance and punctual obligations.

Now that we described the broad range of obligations giving us the necessary vocabulary to translate regulations into our logical framework, we are still missing one critical point of regulations: reparation chains. If an obligation is violated, you are not complying with the regulation unless there is a reparation chain that kicks in and leaves you in an unoptimal but still compliant situation.

For example let us consider the following rules:

$$r: \quad invoice \Rightarrow O^p pay$$
$$r': \quad \neg pay \Rightarrow O^a payFine$$

They can be reduced and be expressed as a $\otimes$-expression such as these two obligation cannot be seen any more as independent.

$$r: invoice \Rightarrow O^p pay \otimes O^a payFine$$

We can now create chains of obligations started by a given set of literal and giving the actor a chance to stay compliant even if an obligation was violated [20].

3 Business Process Compliance

Business Process Compliance [22, 39] is the research area studying techniques to ensure that the business processes of an organisation do not violate the regulations and the law governing the business. To formalise business process compliance two components are needed. The first is a formalism to represent the norms. In Section 2 we proposed Deontic Defeasible Logic for this task. The second component is the representation of the business processes. This has been extensively studied in the field of business process modelling, see [9]. A business process model is defined as a self-contained, temporal and logical order in which a set of activities are executed to achieve a business goal. Typically a process model describes what needs to be done and when (control flow), who is going to do what (resources), and on what it is working on (data). Many different formalisms (Petri-Nets, Process algebras, ...) and notations (BPMN, YAWL, EPC, ...) have been proposed to represent business process models. Besides the difference in notation, purposes, and expressive power, business process languages typically contain *tasks* and *connectors*, where a task corresponds to a (complex) business activity, and connectors (e.g., sequence, and-join, and-split, (x)or-join, (x)or-split) define the relationships among tasks to be executed. The combination of tasks and connectors defines the possible ways in which a process can be executed. Where a possible execution, called *process trace* or simply *trace*, is a sequence of tasks respecting the order given by the connectors.

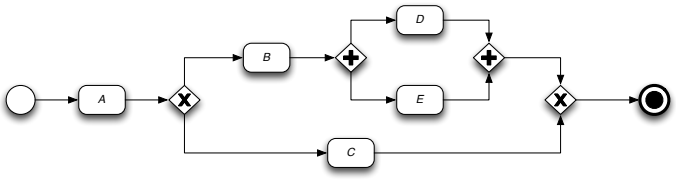

Fig. 1: Example of a business process model in standard BPMN notation

Consider the process in Figure 1, in standard BPMN notation, where we have a task A followed by an xor split. In the xor split in one of the branches we have task B followed by the and-split of a branch with task D, and a brach consisting of only task E. The second branch of the xor-split has only one task: C. The traces corresponding to the process are $\langle A,C\rangle$, $\langle A,B,D,E\rangle$ and $\langle A,B,E,D\rangle$.

Compliance is not only about the tasks that an organisation has to perform to achieve its business goals, but it is concerned also on their effects (i.e., how the activities in the tasks change the environment in which they operate), and the artefacts produced by the tasks (for example, the data resulting from executing a task or modified by the task) [27]. To capture this aspect [40] proposed to enrich process models with semantic annotations. Each task in a process model can have attached to it a set of semantic annotations. An annotation is just a set of formulas giving a (partial) description of the environment in which a process operates. Then, it is possible to associate to each task in a trace a set of formulas corresponding to the state of the environment after the task has been executed in the particular trace.

4 An Algorithm for Business Process Compliance

In the previous sections we described the framework used to express laws and regulations in a proper logical way, and we paired it with business process modelling. In the following we will present the business process compliance algorithm proposed in [19].

For a given business process the algorithm starts by computing all the possible traces, all possible executions of the business process. The reachability graph is computed first using the method described in [34]. From this all possible executions of the process are drawn. Now that we have all possible traces we will focus on one. For each task in the trace several actions are performed. The first step is to compute the state corresponding to the task. We cumulate the effects or semantic annotations attached to task using an update semantics, that is, in case of a conflict, the literal in the current task prevails over the the literal form the previous task. Then, a call is made to the rule engine with the informations about the task. It will return the new obligations generated by these antecedents. These new rules are added to the Current set which contains all rules in-force at a given task. The elements in Current are triples $[T,R,C]$, where T is a task identifier, R is a rule label and C is an $\otimes$-expression. T is the task where rule R triggers the chain of obligations C. The sets Violated and Compensated, as their names suggest, are used to keep track of which obligations in force have been violated and which violations have been compensated for. In both cases the structure of their elements is $[T,R,C,V]$, where T, R and C are as before and V is a literal indicating which obligation in the chain C has been violated. The Unfulfilled set contains all achievement and maintenance rules that were triggered but not fulfilled yet. The elements of Unfulfilled have the same structure as those of Current.

The operations in the algorithm depend on the set of $\otimes$-chains of obligations and on which element of an $\otimes$-chain C we operate on. Accordingly, we will explode C as either $A_1 \otimes A_2$, where A_1 is the element/literal under analysis and A_2 is the remainder of the $\otimes$-chain, or as $B_1 \otimes B_2 \otimes A_1 \otimes A_2$.[1] The cases when we use the second format is when

[1] Any $\otimes$-chain C can be reduced to one of the two expression given the equivalence $A \equiv A \otimes \bot$.

the "current" $\otimes$-chain C is the compensation of a previous violation. The use of B_2 is used to signify that that compensation can be used to compensate the violation of an obligation that itself is a compensation; similarly for A_2. Finally, the "add" and "remove" functions add or remove an element from a given set.

The Terminated set contains rules that were terminated according to the following definition from [19]. A chain C is terminated by a task n if C was active at task n and that another rule r triggered at task n derives the opposite of C. The rule r must not be weaker than the rule that originally yielded C.

for all $C \in$ Current **do**
  if $A_1 = O^p B$ **then**
    if $B \in S$ **then**
      remove($[T,R,A_1 \otimes A_2]$, Current)
      if $[T,R,B_1 \otimes B_2 \otimes A_1 \otimes A_2, B_2] \in$ Violated **then**
        add($[T,R,B_1 \otimes B_2 \otimes A_1 \otimes A_2, B_2]$, Compensated)
      end if
    else
      remove($[T,R,A_1 \otimes A_2]$, Current)
      add($[T,R,A_1 \otimes A_2, B]$, Violated)
      add($[T,R,A_2]$, Current)
    end if
  end if
  if $A_1 = O^a B$ **then**
    if $B \in S$ **then**
      remove($[T,R,A_1 \otimes A_2]$, Current)
      remove($[T,R,A_1 \otimes A_2]$, Unfulfilled)
      if $[T,R,B_1 \otimes B_2 \otimes A_1 \otimes A_2, B_2] \in$ Violated **then**
        add($[T,R,B_1 \otimes B_2 \otimes A_1 \otimes A_2, B_2]$, Compensated)
      end if
    else
      add($[T,R,A_1 \otimes A_2]$, Unfulfilled)
    end if
  end if
  if $A_1 = O^m B$ **then**
    if $b \notin S$ or $\neg B \in S$ **then**
      add($[T,R,A_1 \otimes A_2, B]$, Violated)
      add($[T,R,A_2]$, Current)
    end if
  end if
end for

for all $C \in$ Terminated **do**
  if $C \in$ Unfulfilled **then**
    add($[T,R,A_1 \otimes A_2, A_1]$, Violated)
    add($[T,R,A_2]$, Current)
  end if
  if $A_1 = O^a$ **then**
    remove($[T,R,A_1 \otimes A_2]$, Current)
  end if
end for

Given a trace/plan we extract the effects of each task in it, and for each task, we first compute the defeasible extension of the logic to determine what are the obligation in force, i.e., the obligations to be included in Current. Then we apply the algorithm above for each task.

At the end of the trace/plan we determine if the trace/plan is compliant. A trace/plan is compliant if

1. Current is empty (meaning that all pending obligation have been fulfilled); and
2. All obligations in Violated are also in Compensated.

For a stronger notion of compliance, the second condition can be replaced by that the set Violation is empty. In the former case, the meaning is that there were some violations, but they were compensated for. The stronger notion, on the other hand, requires that there are no violations at all.

5 Temporal Defeasible Deontic Logic

Adding time in defeasible deontic logic is a much anticipated feature because it implements at the source the essence of deadline allowing us, as we will see later on, to represent obligation types elegantly. In [24] extensions to include time in the logic are proposed and we will use these semantics and notations.

The presentation in this section is based on [21] which introduces new notations, semantics and concepts to deal with time in defeasible deontic logic. They represent time as a discrete linear order of instants $\mathscr{T} = (t_1, t_2, \ldots, t_n)$.

The following list sums up the notations introduced in [21] and their semantics:

- If l is a literal then l^t is a temporal literal. We will refer to the set of temporal literal as *TLit*. We also introduce $\top$ and $\bot$ which are also temporal literals, they are propositions that are respectively always complied with and impossible to comply with.
- If l^t is a temporal literal then Ol^t and its negation are deontic literals meaning that the obligation to do l holds at time t. We will refer to the set of deontic literals as *DLit*.
- If a^{t_a} and b^{t_b} are temporal literals, $t \in \mathscr{T}$ and $x \in \{a, m, p\}$ (for achievement, maintenance, and punctual) then $a^{t_a} \otimes_t^x b^{t_b}$ is an $\otimes$-chain used to express chain of reparation in laws. If $x = p$, then we impose that $t = t_a$; otherwise that $t > t_a$.
- If α is an $\otimes$-chain, t and $t_a \in \mathscr{T}$ and a^{t_a} is a temporal literal then $\alpha \otimes_t^x a^{t_a}$ is an $\otimes$-chain. A deontic expression is an $\otimes$-chain composed of temporal literals or sub-$\otimes$-chain and finishing with $\bot$.

An $\otimes$-chain like $\alpha \otimes_t^a a^{t_a} \otimes_{t'}^y b^{t_b} \otimes \bot$ means that the violation of α who holds until time t triggers an achievement obligation a from t_a to t'.

Temporal defeasible logic also defines new defeasible proof conditions. Definition 2 shows when a rule is applicable at index i, meaning that the obligation at index i in the $\otimes$-chain is in force.

Definition 2 *A rule r is applicable at index i in a proof P at line $P(n+1)$ iff*

1. $\forall a \in A(r)$
   (a) *if* $a \in TLit$, *then* $a \in F$ *and*
   (b) i. *if* $a = Ol^t$ *then* $+\partial l^t \in P(1..n)$
   ii. *if* $a = \neg Ol^t$ *then* $-\partial l^t \in P(1..n)$ *and*
2. $\forall c_j \in C(r)$, $1 \leq j \leq i$
   (a) *if mode*(c_j) = *punctual, then* $c_j \notin F$ *or* $\sim c_j \in F$
   (b) *if mode*(c_j) = *achievement, then* $\forall t$, *start*$(c_j) \leq t \leq$ *end*(c_j), $c_j^t \notin F$ *or* $\sim c_j^t \in F$
   (c) *if mode*(c_j) = *maintenance, then* $\exists t$, *start*$(c_j) \leq t \leq$ *end*(c_j), $c_j^t \notin F$ *or* $\sim c_j^t \in F$

In [21] different proof conditions are defined for each obligation type. In Definition 3 we present them in a condensed form. x is used to represent the mode of the obligation, it can be replace by one of a, m, p.

Definition 3 *If* $P(n+1) = +\partial p^t$ *then*

1. $\exists r \in R^x_{\Rightarrow}[p^t, i]$, *r is applicable at index i and,*
2. $\forall s \in R[\sim p^t, j]$ *either*
   (a) *s is discarded at index j*
   (b) $\exists w \in R^x_{\Rightarrow}[p^t, k]$, *w is applicable at index k and* $w \succ s$
3. $\exists x \in R^{a,m}_{\Rightarrow}\left[p^{t'}, i'\right]$, $t' < t$, *end*$(t') \geq t$
   (a) *x is applicable at index i', and*
   (b) $\forall y \in R\left[\sim p^{t''}, j'\right]$, $t' \leq t'' < t$ *either*
      i. *y is discarded at index j' or*
      ii. $\exists z \in R\left[\sim p^{t''}, k'\right]$, *z is applicable at k' and* $z \succ y$; *and for* $+\partial^a$
   (c) $\forall t'''$, $t'' < t''' \leq t$, $p^{t'''} \notin F$.

Conditions 1. and 2. are enough to defeasibly prove a punctual obligation. Condition 3. only applies for maintenance and achievement obligations. The final line 3.c only applies to achievement obligations for which fulfilment terminates the obligation.

The last step is to give the conditions under which a theory is compliant. Checking compliance simply amount to show that $-\partial\perp$, and, conversely non-compliance is signalled by $+\partial\perp$.

6 From Defeasible Deontic Logic to Temporal Defeasible Deontic Logic

In this section we present constructions that allow us to encode the norms regulating a business process and the semantic annotations of the process in Temporal Defeasible Logic and to check compliance directly in that logic. In addition the computation is equivalent to the same compliance results as the combination of Defeasible Deontic Logic and the compliance algorithm presented in Section 4.

A defeasible theory is defined by the tuple $(F, R, >)$ where F is a finite set of literals, R a finite set of rules and $>$ a superiority relation on R. In the context of business process compliance we consider $S_1, \ldots, S_n$ sets of literals representing the literals attached to every task. Therefore for each task we have a different theory. At task n we have

$F = \bigcup_{i=1}^{n} S_i \setminus \{\sim l : l \in S_i\}$, we also refer to F at task n as State(n). The set of rules stays the same although the obligations in force can change from one task to another.

A temporal defeasible theory is not so different from its classical counterpart. It is defined by the tuple $(F^t, R^t, \succ)$ where F^t is a finite set of temporal literals, R^t is a finite set of temporal rules and $\succ$ is a superiority relation on R^t. The Facts and Rule sets are dependent of the current task in a given trace. We highlight this dependence in the following.

We are defining the $\mathscr{T}$ operator which takes a defeasible theory and a point in time and returns the temporal equivalent, formally defined as:

$$\mathscr{T} : \mathbb{N} \times (F, R, >) \to (F^t, R^t, \succ) \tag{1}$$

Every literal in F is temporally annotated with the task number it is attached to. We know that at a given task F is the union of the previous S_i therefore at a given task t we have.

$$F^t = \{q^t \mid \forall q \in S_t\} \tag{2}$$

Every rule in R is basically translated by annotating temporally with the current task number all its antecedent and consequent. Hence the temporal rule arising from a classical rule depends on the task number in a given trace. For each task in every possible trace we define a set of temporal rules corresponding to the body of "classic" rules. All of the antecedents and effects of the rules are annotated with the task number which will play the role of time as the set of task numbers is isomorphic to $\mathbb{N}$ which is a perfect candidate for time. Since defeasible deontic logic does not include deadlines we transform achievement and maintenance obligations to permanent ones defining a parametric deadline using the *viol* operator.

Let us introduce the mode function that returns the mode of a given obligation:

$$mode(O) = \begin{cases} a & \text{if } O \text{ is an achievement obligation} \\ p & \text{if } O \text{ is a punctual obligation} \\ m & \text{if } O \text{ is a maintenance obligation} \end{cases} \tag{3}$$

Here we will show how the set of temporal rules R^t is derived from the set of classical rules R. First in (4) we define a general form for classical rules we will use to define how we translate to temporal rules.

$$\forall r \in R, \quad r : a_1, \ldots, a_n \Rightarrow p_1 \otimes \cdots \otimes p_m \tag{4}$$

In (5) we introduce one of the tools we will need for this demonstration: the *naf* operator [5]. It stands for negation as failure which means that we failed to prove an element. It is defined as:

$$\begin{aligned} r_1 : &\quad \Rightarrow naf\ p \\ r_2 : &\quad \neg p \Rightarrow naf\ p \\ r_3 : &\quad p \Rightarrow \neg naf\ p \\ &\quad r_3 > r_2 > r_1 \end{aligned} \tag{5}$$

In other words we have *naf p* either when p has not been concluded or $\neg p$ has been concluded. If p is concluded then this rule is stronger than the other two and we conclude $\neg$*naf p*

For demonstration purposes we introduce in 6 the *viol* function which takes an obligation and returns the index of the task where it was violated. If the obligation is never violated it returns the index of the last task. This will allow us to express chain of reparation from the classical framework where no deadlines are defined into the temporal one where we need deadlines. This *viol* operator will create artificial parametrized deadlines for chain of obligations as it will be presented next.

$$viol(X)\colon Obligations \to \mathbb{N} \tag{6}$$

– For a maintenance obligation $O^m p$, *viol* will return the index of the first task where the obligation is applicable and we can conclude *naf p*.

$$\begin{aligned} &\text{At task } i \text{ if we have: } O^m p, \textit{naf } p \in F \\ &\text{then: } viol(O^m p) = i \end{aligned} \tag{7}$$

– For an achievement obligation $O^a p$, *viol* will return the index of the first task where the obligation is applicable, we can conclude *naf p*, and the obligation is lifted at the next task (this can be done with defeaters).

$$\begin{aligned} &\text{At task } i \text{ if we have: } O^a p, \textit{naf } p, \neg O^a p \in F \\ &\text{then: } viol(O^a p) = i \end{aligned} \tag{8}$$

– It is not necessary to define *viol* for punctual obligations since they are always violated or fulfilled at the task after they were triggered. For example $O^p p$ is triggered at task i then it is either complied with or *viol* will return $i+1$.

Now we will show how we translate temporally each of these rules, given a classical rule r in the form defined in 4 at a given task t the set of temporal rules is composed of rules.

$$r_i\colon a_1, \ldots, a_n \Rightarrow p_1 \otimes \cdots \otimes p_m \tag{9}$$

$$\begin{aligned} r_i(task)\colon a_1^{task}, \ldots, a_n^{task} &\Rightarrow^{mode(p_1)} \\ & p_1^{task} \otimes^{mode(p_2)}_{viol(p_1)} p_2^{viol(p_1)} \cdots \otimes^{mode(p_m)}_{viol(p_{m-1})} p_m^{viol(p_{m-1})} \otimes \bot \end{aligned} \tag{10}$$

Now we are considering the sets of obligations we find in the algorithm for business process compliance checking. In this we find at each task three main sets of obligations and literals: Current, Violated and State. They contain respectively obligations in force at the given task, obligations that were violated in previous tasks and deontic literals attached to the current or previous tasks. We aim to prove that the transformation of these sets from classical formalism to temporal is isomorphic and our transformation bijective. All these set are depending on the current task we consider, they will be referred as: Set(n)

First let us prove that

$$\text{if } p \in \text{State}(n) \text{ then } +\partial p^n$$

this is trivially proven by definition of our translation where every literal associated with a task is annotated temporally with the task number in the trace. Therefore if p is in the State(n) it has been proven at a step $(1..n)$.

Now what about

$$\text{if } q \in \text{Current}(n) \text{ then } +\partial Oq^n$$

If an obligation is in force at a task n in the classical formalism is it also in the temporal one, in other word is the rule applicable at the index where it triggers q. If q is in the set of current obligation it means that there is a rule r that yields q at task k and that this rule was fired meaning all of the antecedents have been proven. In other words:

$$\begin{aligned}
&\forall a \in A(r),\ a \in \text{ State}(k)\\
&\text{so if } a \text{ is a literal then } a \in \text{Facts}\\
&\text{or if } a \text{ is an obligation } Ol \text{ then } +\partial l^{1..k}
\end{aligned}$$

Or, the conditions on the Antecedents for a rule to be applicable are:

- if the antecedent is a litteral then it must be in the Facts (attached to a previous task)
- if the antecendent is an obligation then it must have been defeasibly proven beforehand.

Both conditions are met so the rule would also trigger in temporal defeasible logic. Now we have to see if it would trigger at the right index in the $\otimes$-chain.

If $q = Ol$ is part of an $\otimes$-chain like $A_1 \otimes \cdots \otimes A_n \otimes q \otimes B_1 \otimes \cdots \otimes B_n$ and if q is in current that means that for all obligation A_i, $-\partial A_i$ has been proven for a previous task $(1..k-1)$. Let m be the task where the rule was triggered first. Which translates into:

- for punctual obligations this means we either have $\neg l$ at task k when the obligation was triggered or that l was not in the State set at task k.
- for achievement obligations this means that we have $\neg l$ at a task between m and $k-1$
- for maintenance obligations this means that we either have $\neg l$ at a task between m and $k-1$ or that l was not in the State set at a given task between m and k

Whatever the obligation this means that at some point we were able to conclude $-\partial A_i$ for all the obligations before q which means that the rule r is also applicable at the index where it triggers q implying that q is also in the set of Current obligations in the temporal formalism.

We can easily translate this reflection to the Violated set. If a obligation Ol is in the violated set this means that at some task between $1..n$ we have one of the three conditions aforementioned for each type of obligation. Which trivially translates into being able to prove $-\partial l$ at some task $(1..n)$ implying that the obligation is also in the Violated set in the temporal formalism.

The use of Temporal Defeasible Deontic Logic allows us to use the same logic to model norms, semantic annotations and to check whether a process is compliant. This can be done by simply computing the extension of the theory encoding all such information. This means that this logic offers an holistic and more conceptually sound approach to the

problem of business process compliance. In addition the temporal framework is expected to improve the computational efficiency of the system. For now we check compliance at each task of the business process collecting the new rules and forwarding them to the next task. With time we could do all of this at once. With temporal rules we would only need to create traces which would yield a set of temporal literals and check this set against the set of temporal rules. Let us consider again the example in Figure 1. In this case we have first to compute all the possible traces; as we have seen we have three traces, namely $\langle A, C\rangle$, $\langle A, B, D, E\rangle$ and $\langle A, B, E, D\rangle$. Each trace corresponds to a serialisation of the process. After the serialisation we go though each task, accumulate effects, derive rules in force, check for compliance and then forward effects and obligations to the next task. For a trace of size n we have to do each operation n times. The result of [8] shows that the problem of checking whether a business process is compliant with a set of norms is an NP-complete problem, however, the temporal approach could eliminate some of the overheads of the other method. In particular if we add time though, we only need to do the computation of the extension only once for each trace instead of repeating it for each task in the trace. In the worst case the total number of temporal literals given as facts is equal to the sum of literals used as semantic annotations in the single traces. Similarly, the number of rules in the temporal version does not exceed the number of rules in the non-temporal version times the number of the tasks in a trace. On the other hand the computation of what obligations have been fulfilled, violated and compensated is part of the computation of the extension, and not of what obligation chains are in force, and then calling the compliance checking algorithm. However, a proper experimental evaluation is required to determine whether the temporal approach speeds up the computation.

7 Summary and Related Work

This paper first presents defeasible deontic logic as presented in [18] and later extensions by [21] with time. This new framework is better suited to represent obligations as these often feature temporal deadlines. However, to the best of our knowledge no work in the field has so far attempted to formally prove that the results from the compliance checking algorithm introduced in [19] yields the same compliance results when porting a theory from the classical to the temporal framework. This proof introduces a new operator to translate a set of rules and then shows that sets of obligations from the compliance algorithm are isomorphic to it. Or in other words that this operator defines a bijection from the classical to the temporal formalism. This result paves the way for future work in adapting the compliance checking algorithm to the temporal framework.

The literature on norm compliance is NorMAS is large (see, e.g., [11, 1, 13, 25, 7, 29, 12, 30]). However, to the best of our knowledge no work in the field has so far attempted to model a *legal* compliance pertaining to realistic systems where complex norm-enforcement mechanisms such as $\otimes$-chains are combined with a rich ontology of obligations as the one described here.

The literature on business process compliance is equally vast (see, e.g., [15, 33, 10, 32, 6]). But it suffers form the same problems as that in NorMAS. Most of these approaches fails to address the proper modelling of norms and normative reasonings. See [26] for a detailed evaluation and comparison of various business process compliance

frameworks. Most of such approaches are based on first oder logic or temporal logic and limited to check the structural compliance of a processes (e.g., correct order of the tasks and presence or absence of tasks). In addition [28] shows that first order logic is not appropriate for the modelling of legal reasoning. Similarly there are some concerns that temporal logics, and in particular LTL, might not be able to model compliance requirements [17].

References

1. Alberti, M., Gavanelli, M., Lamma, E., Chesani, F., Mello, P., Torroni, P.: Compliance verification of agent interaction: a logic-based software tool. Applied Artificial Intelligence 20(2-4), 133–157 (2006)
2. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: On the modeling and analysis of regulations. In: ACIS 1999. pp. 20–29 (1999)
3. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. ACM Transactions on Computational Logic 2(2), 255–287 (2001)
4. Antoniou, G., Billington, D., Governatori, G., Maher, M.J., Rock, A.: A family of defeasible reasoning logics and its implementation. In: ECAI 2000. pp. 459–463. IOS Press (2000)
5. Antoniou, G., Maher, M.J., Billington, D.: Defeasible logic versus logic programming without negation as failure. J. Log. Program. 42(1), 47–57 (2000)
6. Awad, A., Weidlich, M., Weske, M.: Visually Specifying Compliance Rules and Explaining their Violations for Business Processes. Journal of Visual Languages & Computing 22(1), 30–55 (2011)
7. Boella, G., Broersen, J., van der Torre, L.: Reasoning about constitutive norms, counts-as conditionals, institutions, deadlines and violations. In: PRIMA 2008. LNCS, vol. 5357, pp. 86–97. Springer (2008)
8. Colombo Tosatto, S., Governatori, G., Kelsen, P.: Business process regulatory compliance is hard. IEEE Transactions on Services Computing (2014)
9. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2013)
10. Elgammal, A., Türetken, O., van den Heuvel, W.J.: Using patterns for the analysis and resolution of compliance violations. Int. J. Cooperative Inf. Syst. 21(1), 31–54 (2012)
11. Flores, R.A., Chaib-draa, B.: Modelling flexible social commitments and their enforcement. In: ESAW V. LNCS, vol. 3451, pp. 139–151. Springer (2004)
12. Gabaldon, A.: Making Golog norm compliant. In: CLIMA XII. LNCS, vol. 6814, pp. 275–292. Springer (2011)
13. Gaertner, D., García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A., Vasconcelos, W.: Distributed norm management in regulated multiagent systems. In: COIN 2007. LNCS, vol. 4870, pp. 275–286 (2007)
14. Ghallab, M., Nau, D., Traverso, P.: Automated planning – theory and practice. Elsevier (2004)
15. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes with Obligations and Permissions. In: BPM Workshops, LNCS, vol. 4103, pp. 5–14. Springer (2006)
16. Governatori, G.: Business Process Compliance: An Abstract Normative Framework. it-Information Technology 55, 231–238 (2013)
17. Governatori, G.: Thou shalt is not you will. Tech. Rep. 8026, NICTA (2014)
18. Governatori, G., Rotolo, A.: Logic of violations: A Gentzen system for reasoning with contrary-to-duty obligations. Australasian Journal of Logic 4, 193–215 (2006)
19. Governatori, G., Rotolo, A.: A conceptually rich model of business process compliance. In: APCCM 2010. pp. 3–12. ACS (2010)

20. Governatori, G., Rotolo, A.: Norm compliance in business process modeling. In: RuleML 2010. LNCS, vol. 6403, pp. 194–209. Springer (2010)
21. Governatori, G., Rotolo, A.: Justice delayed is justice denied: Logics for a temporal account of reparations and legal compliance. In: CLIMA XII. LNCS, vol. 6814, pp. 364–382. Springer (2011)
22. Governatori, G., Sadiq, S.: The journey to business process compliance. In: Cardoso, J., van der Aalst, W. (eds.) Handbook of Research on BPM, pp. 426–454. IGI Global (2009)
23. Governatori, G., Shek, S.: Regorous: a business process compliance checker. In: ICAIL 2013. pp. 245–246. ACM (2013)
24. Governatori, G., Terenziani, P.: Temporal extensions to defeasible logic. In: AI 2007. LNCS, vol. 4830, pp. 476–485. Springer (2007)
25. Grossi, D., Aldewereld, H., Dignum, F.: Ubi lex, ibi poena: Designing norm enforcement in e-institutions. In: COIN 2006. LNCS, vol. 4386, pp. 101–114. Springer (2006)
26. Hashmi, M., Governatori, G.: A methodological evaluation of business process compliance management frameworks. In: AP-BPM. LNBIP, vol. 159, pp. 106–115. Springer (2013)
27. Hashmi, M., Governatori, G., Wynn, M.T.: Business process data compliance. In: RuleML 2012. LNCS, vol. 7438, pp. 32–46. Springer, Heidelberg (2012)
28. Herrestad, H.: Norms and formalization. In: ICAIL 1991. pp. 175–184. ACM (1991)
29. Hübner, J.F., Boissier, O., Bordini, R.: From organisation specification to normative programming in multi-agent organisations. In: CLIMA XI. LNCS, vol. 6245, pp. 117–134. Springer (2010)
30. Knorr, M., Gabaldon, A., Gonçalves, R., Leite, J., Slota, M.: Time is up! – norms with deadlines in action languages. In: CLIMA XIV. LNCS, vol. 8143, pp. 223–238. Springer (2013)
31. Koons, R.: Defeasible reasoning. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Spring 2014 edn., `http://plato.stanford.edu/archives/spr2014/entries/reasoning-defeasible/`
32. Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems - requirements, challenges, solutions. Information Systems Frontiers 14(2), 195–219 (2012)
33. Maggi, F., Montali, M., Westergaard, M., van der Aalst, W.: Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. In: BPM 2011, LNCS, vol. 6896, pp. 132–147. Springer (2011)
34. Mailund, T., Westergaard, M.: Obtaining memory-efficient reachability graph representations using the sweep-line method. In: TACAS 2004, LNCS, vol. 2988, pp. 177–191. Springer (2004)
35. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. Stanford University USA (1968)
36. McNamara, P.: Deontic logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Spring 2014 edn. (2014), `http://plato.stanford.edu/archives/spr2014/entries/logic-deontic/`
37. Nute, D.: Defeasible logic. In: Handbook of logic in artificial intelligence and logic programming, vol. 3, pp. 353–395. Oxford University Press (1994)
38. Nute, D. (ed.): Defeasible deontic logic. Springer (1997)
39. Sadiq, S., Governatori, G.: Managing regulatory compliance in business processes. In: vom Brocke, J., Rosemann, M. (eds.) Handbook on Business Process Management 2, pp. 265–288. Springer Berlin Heidelberg (2015)
40. Sadiq, S., Governatori, G., Naimiri, K.: Modelling of control objectives for business process compliance. In: BPM 2007. LNCS, vol. 4714, pp. 149–164. Springer (2007)