Detecting Regulatory Compliance for Business Process Models through Semantic Annotations

Guido Governatori[1], Jörg Hoffmann[2], Shazia Sadiq[3], and Ingo Weber[2,⋆]

[1] National ICT Australia, Queensland Research Lab, Brisbane, Australia
guido.governatori@nicta.com.au
[2] SAP Research Karlsruhe, Germany
{joe.hoffmann,ingo.weber}@sap.com
[3] School of School of Information Technology and Electrical Engineering
The University of Queensland, Brisbane, Australia
shazia@itee.uq.edu.au

Abstract. A given business process may face a large number of regulatory obligations the process may or comply with. Providing tools and techniques to evaluate the compliance degree of a given process is a key objective in emerging business process platforms. We propose a diagnostic framework to assess the compliance gaps present in a given process. Checking whether a process is compliant with the rules involves enumerating all reachable states and is hence, in general, a hard search problem. The approach taken here allows to provide useful diagnostic information in polynomial time based on two underlying techniques. A conceptually faithful representation for regulatory obligations is firstly provided by a formal rule language based on a non-monotonic deontic logic of violations. Secondly, processes are formalized through semantic annotations that allow a logical state space to be created. The intersection of the two allows us to devise an efficient method to detect compliance gaps.

Keywords: Regulatory Compliance, Business Process Models, Semantic Annotations.

1 Background and Motivation

Compliance management is an area of increasing importance in several industry sectors where there is a high incidence of regulatory control e.g. financial services, gaming, and healthcare. Ensuring that business practices reflected in business process models are compliant to required regulations (existing and new) is a highly challenging task due to the following reasons. Firstly, the lifecycles of the two (regulatory obligations vs. business strategy) are not aligned in terms of time, governance, or stakeholders [1]. Often, the source of objectives for the two will be distinct both from an ownership and governance perspective, as well as from a timeline perspective. Whereas businesses will base their process design on business objectives, (regulatory) control objectives will be dictated by mostly external sources and at different times. Hence compliance

[⋆] This work has in part been funded by the European Union (IST FP6-026850, http://www.ip-super.org).

D. Ardagna et al. (Eds.): BPM 2008 Workshops, LNBIP 17, pp. 5–17, 2009.

requirements cannot simply be incorporated into the initial design of process models. Secondly, conceptually faithful specifications for compliance rules and process models respectively are fundamentally different from a representational point of view thus making it difficult to provide comparison methods. Furthermore, there is likelihood of conflicts, inconsistencies and redundancies within the two and hence the intersection of the two needs to be carefully studied.

We aim at providing computationally effective methods for studying the intersection of the two specifications: regulatory controls and business processes. The proposed framework assesses whether a given business process complies with a set of regulatory control objectives. In general, deciding whether a non-compliant state exists involves enumerating all the, exponentially many, reachable states. However, as we show herein, useful diagnostic information can be obtained in polynomial time. We draw on recent methods for semantically annotated business processes, which generate a kind of summary of the states that may be reached by a process. We devise new algorithms that exploit such summaries to detect compliance gaps. The algorithms guarantee to detect all obligations that will necessarily arise during execution, but that will not necessarily be fulfilled. Upon completion, our procedure provides a status report on the activities in a business process. The report labels problematic cases with the control objective that may be violated, and provides information as to whether a subset of the possible executions – or even all executions – will violate that objective at this point in the process.

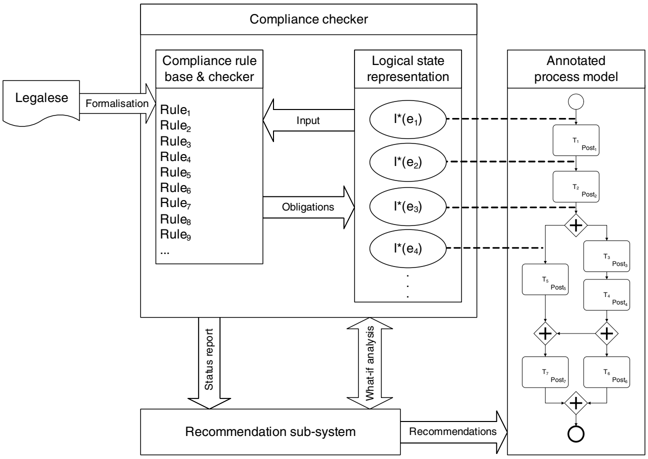

Fig. 1. An Overview of Compliance Checking Framework

Fig 1 provides an overview of our diagnostic framework. On the right hand side of the figure, we see the business process model, whose individual activities have been annotated in terms of the effects they produce thereby providing a logical state representation. On the left hand side of the figure, we see the compliance checker, which consists of an interaction between the compliance rule base and the logical state representation. The output of the compliance checker is a status report, as explained above.

2 Preliminaries

To facilitate discussion and illustration, we introduce an account opening process as depicted in Fig 2.

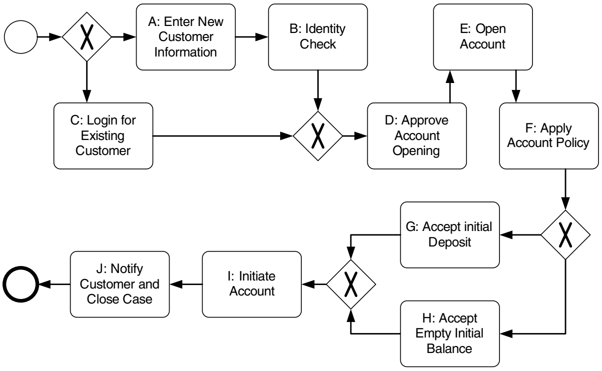
A: Enter New Customer Information
B: Identity Check
E: Open Account
C: Login for Existing Customer
D: Approve Account Opening
F: Apply Account Policy
G: Accept initial Deposit
J: Notify Customer and Close Case
I: Initiate Account
H: Accept Empty Initial Balance

Fig. 2. Example account opening process in private banking

The control objectives for this example are motivated by the following scenario: A new legislative framework has recently been put in place in Australia for anti-money laundering. The first phase of reforms for the *Anti-Money Laundering and Counter-Terrorism Financing Act 2006* (AML/CTF), covers the financial sector and imposes a number of obligations, which include: customer due diligence (identification, verification of identity and ongoing monitoring of transactions); reporting (suspicious matters, threshold transactions and international funds transfer instructions); and record keeping. AML/CTF is a principles or risk based regulation and hence businesses need to determine the exact manner in which they will fulfil the obligations, which comprises the design of internal controls specific to the organization. Table 1a contains a natural language description of the control objectives and corresponding internal controls for this process; Table 1b shows the semantic effect annotations of the process activities.

Table 1. Control objectives (left) and annotations (right) for the process in Fig 2

Control Objective	**Internal Control**
Customer due diligence	All new customers must be scanned against provided databases for identity checks.
	Accounts must maintain a positive balance, unless approved by bank manager, or for VIP customers.
Record keeping	Retain history of identity checks performed.

Task	Semantic Annotation
A	$newCustomer(x)$
B	$checkIdentity(x)$
C	$checkIdentity(x)$, $recordIdentity(x)$
E	$owner(x,y)$, $account(y)$
F	$accountType(y,type)$
G	$positiveBalance(y)$
H	$\neg positiveBalance(y)$
I	$accountActive(y)$
J	$notify(x,y)$

2.1 Modeling Control Objectives

Compliance can be understood in terms of the normative positions (i.e., obligations, prohibitions, etc.) a business process has to comply with. This means that to tackle this issue one has to adopt a formalism capable to model and reason with such notions.

Many formalisms have been proposed to represent normative notions such as obligations, prohibitions and permissions. In this paper we adopt FCL (Formal Contract Language) [2] as formalism to model the control objective (aka 'normative' specifications). FCL is a combination of an efficient non-monotonic formalism (defeasible logic [3]) and a deontic logic of violations [4]. This particular combination allows us to represent exceptions as well as the the ability to capture violations and the obligations resulting from the violations, and the reparations; in addition FCL has good computational properties: the extension of a theory (i.e., the set of conclusions/normative positions following from a set of facts can be computed in time linear to the size of the theory).

We illustrate how to use FCL to represent and reason about "normative" specifications relative to a business process. See [2,5] for detailed presentations.

A rule in FCL is an expression of the form $r : A_1, \dots, A_n \Rightarrow B$, where r is the (unique) name of the rule, $A_1, \dots, A_n$ are the *premises* (propositions in the logic), and B is the *conclusion* (also a proposition of the logic). The propositions of the logic are built from a finite set of atomic propositions, and the following operators: $\neg$(negation), O (obligation), P (permission), and $\otimes$ (violation/reparation).

If p is an atomic proposition, then $\neg p$ is a proposition. If p is a proposition, then Op is an *obligation proposition* and Pp is a *permission proposition*; both are called *deontic propositions.*[1] If $p_1, \dots, p_n$ are obligation propositions and q is a deontic proposition, then $p_1 \otimes \cdots \otimes p_n \otimes q$ is a *reparation chain*. A simple proposition corresponds to a factual statement. A reparation chain captures obligations and normative positions arising in response to violations of obligations. For example, $B_1 \otimes B_2$ means that the process is obliged to perform B_1; and in case B_1 is not fulfilled (i.e., the obligation is violated), the "secondary" obligation B_2 must be fulfilled. While single obligations and permissions (and their negations) can appear in the premises of a rule, reparation chains can be used only in rule conclusions.

FCL is equipped with a superiority relation (a binary relation) over the rule set. The superiority relation ($\prec$) determines the relative strength of two rules, and it is used when rules have potentially conflicting conclusions. For example given the rule $r_1 : A \Rightarrow B \otimes C$ and $r_2 : D \Rightarrow \neg C$. $r_1 \prec r_2$ means that rule r_1 prevails over rule r_2 in situation where both fire and they are in conflict (i.e., rule r_2 fires for the secondary obligation C).

In the context of business process it is important to distinguish different types of obligations and when they are fulfilled or violated. Here we follow the classification proposed by [6], where obligations are classified in the following classes:

Persistent maintenance obligations indicated as $O^{p,m}$. Whenever such an obligation arises in an execution state s, it persists for all states that come after s. Any state, starting from s, that does not satisfy the obligation, is non-compliant.

[1] We assume the standard relationships between the deontic operators: $Op \equiv \neg P \neg p$ and $Pp \equiv \neg O \neg P$. A prohibition is represented by $O\neg$.

Persistent achievement obligations denoted by $O^{p,a}$. Such obligations also persist. However, it is sufficient if they are achieved at some point later than s; they may thereafter be falsified again without violating compliance. A violation of an achievement obligation can be detected when the obligation expires (i.e., when we reach the deadline to fulfil it, see [6]). Here we do not consider temporal constraints, and we set the deadlines of all achievement obligations at the end of the process where they occur.

Non-persistent obligations indicated as O^n. Such obligations are evaluated on a state-by-state basis: whenever they arise, they must immediately be satisfied; but they have no consequence on any of the states yet to come.

Our aim is to identify whether a given process complies with a set of rules. Thus we must be able to determine all and only obligations generated by a set of facts. We use the *normalisation* procedure of FCL (see [5]) to generate the set of rules with all unique maximal reparation chains. The compliance checkers use the maximal chains to determine whether a task in a process and then a process itself complies with the rules.

The control objectives in Table 1a can be expressed by the following FCL rules:

1. All new customers must be scanned against provided databases for identity checks.

$$r_1 : \mathit{newCustomer}(x) \Rightarrow O^n \mathit{checkIdentity}(x)$$

The predicate $\mathit{newCustomer}(x)$ means that the input data with $Id = x$ is a new customer, for which we have the (non-persistent) obligation to check the provided data against provided databases $\mathit{checkIdentity}(x)$. This means that as soon as a check has been performed, the obligation is no longer in force.

2. Retain history of identity checks performed.

$$r_2 : \mathit{checkIdentity}(x) \Rightarrow O^{p,m} \mathit{recordIdentity}(x)$$

This rule establishes that there is a permanent obligation to keep record of the identity for to the (new) customer identified by x. In addition this obligation is not fulfilled by the achievement of the activity (for example, by storing it in a database). We have a violation of the condition, if for example, the record x is deleted from the database.

3. Accounts must maintain a positive balance, unless approved by a bank manager, or for VIP customers.

$$r_3 : \mathit{account}(x) \Rightarrow O^{p,m} \mathit{positiveBalance}(x) \otimes O^{p,a} \mathit{approveManager}(x)$$

The primary obligation (a persistent maintenance obligation) is that each account has to maintain a positive balance *positiveBalance*; if this condition is violated (for any reason the account is not positive), then we still are in an acceptable situation if a bank manager approve the account not to be positive. In this case the obligation (a persistent achievement obligation) of approving it persists until a manager approves the situation; after the approval the obligation is no longer in force.

$$r_4 : \mathit{account}(x), \mathit{owner}(x,y), \mathit{accountType}(x, VIP) \Rightarrow P^n \neg \mathit{positiveBalance}(x)$$

This rule creates an exception to rule r_3. Accounts of type VIP are allowed to have a non positive a balance and no approval is required for this type of accounts (this is achieved by imposing that rule r_4 is stronger than rule r_3, $r_4 \prec r_3$). Notice that the normative position associated to r_4 is a permission.

2.2 Annotated Process Model

The basic execution semantics of the control flow aspect of a business process model is defined using token-passing mechanisms, as in Petri Nets. The definitions used here extend [7] with semantic annotations in the form of effects and their meaning [8].

A process model is seen as a graph with nodes of various types – a single start and end node, task nodes, XOR split/join nodes, and parallel split/join nodes – and directed edges (expressing sequentiality in execution). The number of incoming (outgoing) edges are restricted as follows: start node 0 (1), end node 1 (0), task node 1 (1), split node 1 ($>$1), and join node $>$1 (1). The location of all tokens, referred to as a *marking*, manifests the state of a process execution. An execution of the process starts with a token on the outgoing edge of the start node and no other tokens in the process, and ends with one token on the incoming edge of the end node and no tokens elsewhere (cf. *soundness*, e.g., [9]). Task nodes are executed when a token on the incoming link is consumed and a token on the outgoing link is produced. The execution of a XOR (Parallel) split node consumes the token on its incoming edge and produces a token on one (all) of its outgoing edges, whereas a XOR (Parallel) join node consumes a token on one (all) of its incoming edges and produces a token on its outgoing edge.

As for the semantic annotations, the vocabulary is presented as a set of predicates P. There is a set of process variables (x and y in Table 1b), over which logical statements can be made, in the form of literals involving these variables. The task nodes can be annotated using *effects* (eff, also referred to as *postconditions*), which are conjunctions of literals using the process variables. The meaning is that, if executed, a task changes the state of the world according to its effect: every literal mentioned by the effect is true in the resulting world; if a literal l was true before, and is not contradicted by the effect, then it is still true (i.e., the world does not change of its own accord).

2.3 Logical State Representation

To check compliance we have to figure out (a) the obligations that will definitely appear when executing the process, and (b) which of those obligations may not be fulfilled. To this end we use as input the information provided by an algorithm called *I-propagation*, a restricted special case of a semantic validation framework for business processes [8]. This algorithm determines, at every edge e in the process graph, the set $\bigcap e$ of literals that is true in *every* reachable state where e is activated. Here we focus on how to make use of the sets $\bigcap e$, through additional algorithms, to provide answers to (a) and (b).

I-propagation assumes that there are no loops in the process and that there are no “effect conflicts”, i.e., parallel nodes with conflicting effects (which is clearly undesirable anyway since such effects may occur at the same time). Parallel nodes are detected by a pre-process to the I-propagation, and any effect conflicts can be pointed out to the process modeller. Once this is completed, the I-propagation itself starts. It maintains a set $I(e)$ for every edge e. Initially, all these sets are undefined except for the start node n_0 where $I(n_0)$ is set to be equal to eff(n_0). The algorithm then performs propagation steps until $I(e)$ is defined for all edges in the graph. Each propagation step “fires” a graph node. A node can only be fired if $I(e)$ is defined for all its incoming edges, and $I(e)$ is undefined for all its outgoing edges. What a propagation step does, depends on the kind of node fired. Parallel and XOR splits simply copy the $I(e)$ from their incoming edge to all their outgoing edges. XOR joins with incoming edges E and outgoing edge e' assign $I(e') := \bigcap_{e \in E} I(e)$ since only what's made true on all paths is guaranteed to be true beyond the join. Parallel joins assign $I(e') := \bigcup_{e \in E} I(e)$. The rationale for this is that all the paths will be executed; and without effect conflicts no contradictions will appear. Task nodes n, finally, assign $I(e') := I(e)$ where e is the incoming edge and e' is the outgoing edge of n; they then subtract the negation of their effect from $I(e')$, and insert the effect itself. The intuition behind this should be clear. A more subtle required operation is to subtract the negation of the effect from the edges of any node n' that is parallel to n. This is required since n' might have inherited (though not established itself, due to the postulated absence of effect conflicts) such literals; they may be negated by n and are hence not always true at the outgoing edges of n'. The final sets are denoted with $I^*(e)$; we have $I^*(e) = \bigcap e$.

Consider tasks A, B, and C in Fig 2. At the incoming edges of A and C, I^* is empty. At the outgoing edge of A we have $I^* = \{newCustomer(x)\}$; at the outgoing edge of B we have $I^* = \{newCustomer(x), checkIdentity(x)\}$; at the outgoing edge of C we have $I^* = \{checkIdentity(x),\ recordIdentity(x)\}$. Taking the intersection at the subsequent XOR join, we get $I^* = \{checkIdentity(x)\}$ at the join's outgoing edge.

3 Compliance Checking

In this section, we will demonstrate how the rule base created through FCL and the sets $I^*(e)$ are used to perform compliance checking on business process models. The proposed method produces a *status marker* for each obligation arising at each edge e. A status marker can represent different values such as fulfilled, violated etc. The compliance checking method basically determines: (1) which obligations necessarily arise at any time a particular edge e is activated. This is done by calling FCL rule evaluation on the set of literals $I^*(e)$. The outcome of FCL evaluation is a set of reparation chains rc, each taking the form $rc = \langle O_1^{rc}, \ldots, O_k^{rc} \rangle$. Each reparation chain rc has an identifier corresponding to the name of the rule where the chain appears. We take rc to be that identifier. (2) which of the obligations may remain unfulfilled. This depends on the kind of obligation as given in Section 2.1. Since persistent and non-persistent obligations can *not* be mixed within a reparation chain, we need to devise two methods: one for reparation chains containing only non-persistent obligations, and one for reparation chains containing persistent maintenance or achievement obligations.

Non-Persistent Obligations must be satisfied immediately whenever they arise; if the state changes, the obligation becomes inactive again. We can hence check any possible non-compliance, at the various points in the process, by a simple loop over all edges e. We first set $Chains^{np}(e)$ as the subset of reparation chains consisting of non-persistent obligations. Based on $I^*(e)$, we can assign each O_i^{rc} a particular *status marker*:

Fulfilled: $O_i^{rc} \in I^*(e)$. Then we know that this obligation is definitely fulfilled, i.e., it is satisfied in all possible execution paths whenever e carries a token. We replace O_i^{rc} with $\top_i^{rc}$ to indicate this status; we will sometimes simply say O_i^{rc} *is* $\top$.

Violated: $\neg O^{rc}_i \in I^*(e)$. Then we know that this obligation is definitely not fulfilled, whenever e carries a token. We replace O^{rc}_i with $\bot^{rc}_i$ to indicate this status; we will sometimes say O^{rc}_i *is* $\bot$.

Possibly violated: Neither of the above. Then there exists at least one execution path where this obligation is not fulfilled, in a state where e is activated; there also exists such a path where the obligation is fulfilled. In this case we leave O^{rc}_i as it is (do not replace it with any other symbol); we sometimes say that O^{rc}_i *is unknown*.

Based on these status markers for every element of the reparation chain rc, we can easily provide a status report for the overall chain rc. We consider the entries O^{rc}_i from front to back. If O^{rc}_i is $\top$, then we can stop since we know that the obligations O^{rc}_j, $j > i$, need not be considered; in particular if O^{rc}_1 is $\top$ then no status report is needed at all. If O^{rc}_i is $\bot$, then we report that the obligations O^{rc}_j, $j > i$, *must* be considered. If O^{rc}_i is unknown, then we report that it may be violated.

Consider this rule on the process from Fig 2: $newCustomer(x) \Rightarrow O^n checkIdentity(x)$. According to the annotations of the process, c.f. Table 1b, $newCustomer(x)$ is set after activity A, and $checkIdentity(x)$ is set after activities B and C. The literal $newCustomer(x)$ is contained in $I^*(e)$ of the outgoing edges of B and C – these are the points in the process where this rule will definitely fire. After B, $checkIdentity(x)$ is not set and hence the obligation is violated. After C, the obligation is fulfilled.

Persistent Obligations require additional propagation mechanisms, because we need to remember which obligations we will inherit from earlier on. For achievement obligations, we additionally need to remember whether or not the obligation is certain to have been fulfilled yet (and, conversely, whether or not the obligation is certain to always be violated so far). We first consider the propagation of obligations.

The algorithm maintains a set $Chains^p(e)$ of reparation chains, for every edge e. Initially, $Chains^p(e)$ is undefined for all edges e. As with the I-propagation, the algorithm then performs propagation steps until $Chains^p(e)$ is defined for all edges in the graph; each propagation step "fires" a graph node whose incoming edges are defined and whose outgoing edges are undefined.

Any node n first calls FCL rule evaluation on all its outgoing edges e', based on the sets $I^*(e')$; i.e., we set $Chains^p(e')$ is the set of reparation chains rc consisting of persistent obligations returned by FCL rule evaluation. Thereafter, any n except join nodes takes $Chains^{p,m}(e)$ from its incoming edge e, and sets $Chains^{p,m}(e') := Chains^{p,m}(e') \cup Chains^{p,m}(e)$. For XOR joins with incoming edges E and outgoing edge e', instead $Chains^{p,m}(e') := Chains^{p,m}(e') \cup \bigcap_{e \in E} Chains^{p,m}(e)$ is taken. For parallel joins, instead $Chains^{p,m}(e') := Chains^{p,m}(e') \cup \bigcup_{e \in E} Chains^{p,m}(e)$ is taken. Once no more propagations are possible, $Chains^p(e)$ contains exactly the persistent reparation chains that are certain to be active whenever e is active. The operations here are over chain *identifiers*, i.e., over the names of the respective responsible FCL rules. Thus in our status report we can refer every unfulfilled obligation back to the rule that caused it.

After the propagation of rules is finished, a second pass over the process is necessary in order to assign the correct status markers to all the obligations. This is done in three phases. Phase (I) assigns *local* status markers. This is done exactly as explained for non-persistent obligations above. Phase (II) performs an additional propagation algorithm necessary to keep track of how the status of achievement obligations develops across node executions. Finally, phase (III) performs a propagation keeping track of the overall status of each reparation chain.

For illustration of the $\mathit{Chains}^p(e)$ propagation and phase (I), consider this rule on the process from Fig 2: $\mathit{checkIdentity}(x) \Rightarrow O^{p,m}\mathit{recordIdentity}(x)$. According to the annotations of the process, $\mathit{checkIdentity}(x)$ is set after B and C; the obligation $\mathit{recordIdentity}(x)$ arises at both; so it is contained in the intersection of the $\mathit{Chains}^{p,m}(e)$ sets, taken at the following XOR join. Thereafter, the obligation arises at every edge in the graph. Regarding local status markers, the obligation is accomplished only by the effect annotation of C. Hence it is fulfilled (is marked $\top$) at the outgoing edge of C, but it is violated (marked $\bot$) at the outgoing edge of B, and it is possibly violated (marked unknown) at every edge later on in the graph. This adequately reflects the status of this obligation, which arises in every execution path but is violated if x is a new customer.

We now explain phase (II), propagating status markers for achievement obligations. The propagation works on the status markers in the chains $rc \in \mathit{Chains}^p(e)$. For the sake of readability, we act in the following as if rc contained only achievement obligations; the algorithm for mixed chains simply ignores the maintenance obligations in the chain. The status markers are propagated in a way so that O_i^{rc} is $\top$ iff, in every execution of the process, O_i^{rc} has been true sometime between activation of rc and activation of e. O_i^{rc} is $\bot$ iff, in every execution of the process, O_i^{rc} has always been false between activation of rc and activation of e. The propagation steps are as follows:

Splits: If n is a parallel split or an XOR split, then the $\top$ and $\bot$ markers are simply copied from n's (single) incoming edge to all of n's outgoing edges.

Task nodes: Say n is a task node, with incoming edge e and outgoing edge e'. For all $rc \in \mathit{Chains}^p(e') \cap \mathit{Chains}^p(e)$ and all O_i^{rc}, we now compare the markers in e (which are set by our previous propagation) and e' (which were set in phase (I)). If O_i^{rc} is $\top$ at e, we set it to $\top$ at e', regardless of its previous status – we have already achieved O_i^{rc} and hence the obligation is fulfilled. Afterwards, if O_i^{rc} is $\bot$ at e' but is not $\bot$ at e, then we set it to be unknown at e' – reflecting the fact this obligation may have already been achieved.[2] The chains in $\mathit{Chains}^p(e') \setminus \mathit{Chains}^p(e')$ are left unaffected, i.e., these chains are new and their markers are as per phase (I).

Parallel joins: Say n is a parallel join with incoming edges E and outgoing edge e'. Then the old markers (generated by phase (I)) are kept entirely intact for every chain rc that was not present before, i.e., that is not contained in any $\mathit{Chains}^p(e)$ set, for $e \in E$. Note that such rc may be present since $I^*(e')$ may contain more literals than any of $I^*(e)$, $e \in E$. For the chains rc that *were* present before, the old markers are over-written as follows.

The $\top$ markers are combined from $e \in E$ by point-wise OR. That is, if a chain with the rule identifier rc appears in more than one of the $\mathit{Chains}^p(e)$ sets, for $e \in E$, and O_i^{rc} is $\top$ in at least one of those, then O_i^{rc} is set to $\top$ at e'. Since all the incoming paths will be executed, O_i^{rc} is certain to have been achieved already if that is the case on at least one of the paths.

Conversely, $\bot$ symbols are combined by point-wise AND, meaning they are $\bot$ in e' iff that is the case for all incoming edges e.

Any O_i^{rc} not affected by the above is set to be unknown at e'.

[2] Note here that, due to the previous update of $\top$ markers, O_i^{rc} is unknown at e in this case.

XOR joins: Say n is an XOR join with incoming edges E and outgoing edge e'. Then all old (phase (I)) markers are over-written – note that no new chains can arise at the outgoing edges of XOR joins, so all chains were present beforehand.

The $\top$ markers are combined by point-wise AND; that is, O_i^{rc} is set to $\top$ at e' iff that has been done in every $Chains^p(e)$, $e \in E$, which contains rc. This reflects the fact that, since only one of the incoming paths will be executed, O_i^{rc} is certain to have been achieved only if that holds for all possible paths. Somewhat counter-intuitively at first sight, the same is the case of $\perp$ markers: O_i^{rc} is certain to *not* have been achieved only if that holds for all possible paths. Hence $\perp$ markers are also combined by point-wise AND. Any unaffected obligation is set unknown.

To illustrate the above, consider this rule on the process from Fig 2:

$$account(y) \Rightarrow O^{p,m} positiveBalance(y) \otimes O^{p,a} approveManager(y).$$

According to the annotations of the process, $account(y)$ is set by activity E. From there on, $O^{p,m} positiveBalance(y) \otimes O^{p,a} approveManager(y)$ is active – is contained in $Chains^p(e)$ – at every edge e of the process. $positiveBalance(y)$ is provided only by activity G, and is hence not contained in any $I^*(e)$ set other than at the outgoing edge of G. $approveManager(y)$ does not appear at all in the process, i.e., this step is not modelled. Our formalism assumes that, hence, we do not know anything about this fact and can treat it neither as being true nor as being false. Concretely, due to the algorithms above neither $approveManager(y)$ nor $\neg approveManager(y)$ appear in any $I^*(e)$ set, and hence this obligation is neither marked with $\top$ not with $\perp$ anywhere. We rightly conclude that there exists at least one execution of the process that violates both obligations in the chain.

Now, say we insert an activity H2 after H, providing $approveManager(y)$. Then both sides of the XOR split are handled correctly – the top path with G achieves the first obligation of the rule, while the bottom path with $H2$ achieves the second obligation – so one would expect things to be fine as well directly after the XOR join. However, the markings of the rule are both unknown at the outgoing edge of the XOR join. Each obligation is $\top$ on one of the incoming edges, but $\perp$ ($positiveBalance(y)$ on the bottom path) respectively unknown ($approveManager(y)$ on the top path) on the other incoming edge. Hence the status report after the XOR join would wrongly report that the chain of obligations may be violated. Put in formal terms, the problem is that we cannot accurately deal with *disjunctions* of propositions: we know that neither $positiveBalance(y)$ nor $approveManager(y)$ are always true, but we don't know that at least one of them is always true. We are currently investigating whether disjunctions can be dealt with accurately, in our context, in polynomial time.

While we cannot accurately determine whether at least one part of a chain is always made true at a particular point in the process, we *can* provide a useful approximation. This is the role of phase (III) of our algorithm. That phase performs propagation steps which keep track of status markers $\top$ associated with *entire chains rc*. If rc is set to $\top$ at e then this means that we can prove that at least one of rc's obligations will always be true whenever e carries a token. If rc is not set to $\top$ (i.e., rc is set to be unknown), then we were not able to prove anything. In other words, we are conservative and, if we do not report a possible error, then we have proved that such an error does not exist; we may issue warnings that do not actually correspond to real errors.

For lack of space, we ommit the details of the propagation algorithm. One key trick is that, at an XOR join, if *rc* is set to $\top$ on all incoming edges, then *rc* is set to $\top$ on the outgoing edge. This reflects the fact that, if every possible path achieves at least one of *rc*'s items, then the same is certain to be true after the join. To illustrate this, reconsider the above example, where we inserted H2, after H, into the process. The relevant FCL rule is: $account(y) \Rightarrow O^{p,m}positiveBalance(y) \otimes O^{p,a}approveManager(y)$. H2 provides $approveManager(y)$ and hence both paths (G respectively H,H2) of the XOR construct fulfill one of the rule's obligations. Hence the rule is marked $\top$ after the join.

Let us finally explain how the status reports are created, for every edge e and for every $rc \in Chains^p(e)$. Four cases are possible:

Fulfilled: One of *rc*'s obligations is $\top$ at e, or *rc* itself is $\top$ at e. Then we proved that every reachable state activating e fulfills at least one of the obligations in the chain.

Violated: All of *rc*'s obligations are $\perp$ at e. Then we proved that no reachable state activating e fulfills any of the obligations in the chain.

Possibly violated: None of the above, and only one of *rc*'s obligations is unknown. Then we proved that at least one reachable state activating e does not fulfill any of the obligations in the chain.

Warning: None of the above, i.e., *rc* is unknown, every O_i^{rc} is either $\perp$ or unknown, and at least two O_i^{rc} are unknown. Then it may or may not be the case that at least one reachable state activating e does not fulfill any of the obligations in the chain. We can hence issue a warning to the user.

4 Related Work and Conclusions

The main contribution of this paper is a framework to check the compliance of a business process against a set of normative specifications representing obligations of the business towards regulatory compliance, based on a rich and comprehensive classifications of obligations identified in [6], as far as we know this is the first work addressing compliance based on a conceptually faithful representation of the obligations a process is subject to. The proposed framework provides the ability to (a) identify the obligations that will definitely arise in a given business process and (b) which of those obligations are definitely fulfilled, violated or remain unfulfilled. This is achieved by combination of two formal methods: a logic to support the reasoning with deontic concepts and a formalism to model the execution semantics of a business process. For second part we extend the technique of [8] to identify which obligations must be propagated from one task to successive ones based on the classification of the obligations.

Current approaches to compliance management are heavily inclined towards retrospective checking. The approach we present follows a more proactive and preventative thinking, which although recognized from organizational point of view [10], is lacking in IT solutions. Notable exceptions include [11], that provides the ability to check business processes against rules emerging from business contracts, although the method is limited to checking task sequences rather than detailed process states as provided in this paper. The technique of [11] is then used to determine the degree of compliance of a business process [12]. [13] takes a similar approach of checking process models against compliance rules, although the visual rule language, namely BPSL is general purpose and does not directly address the deontic notions prevalent in compliance requirements. [14] presents a logical language PENELOPE with the ability to verify temporal constraints arising from compliance requirements on business processes.

[15] considers a similar approach where the tasks of a business process model, written in BPMN, are annotated with the effects of the tasks, and a technique to propagate and cumulate the effects from a task to a successive contiguous one is proposed. Contrary to what we do this approach does not determine at design time whether a business process is both executable and compliant. [16] investigates compliance in the context of agents and multi-agent systems based on a classification of paths of tasks.

References

1. Sadiq, S., Governatori, G., Namiri, K.: Modelling control objectives for business process compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 149–164. Springer, Heidelberg (2007)
2. Governatori, G.: Representing business contracts in RuleML. International Journal of Cooperative Information Systems 14, 181–216 (2005)
3. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. ACM Transactions on Computational Logic 2, 255–287 (2001)
4. 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)
5. Dongen, B., Mendling, J., Aalst, W.: Structural Patterns for Soundness of Business Process Models. In: Proceedings EDOC 2006, pp. 116–128. IEEE, Los Alamitos (2006)
6. Governatori, G., Hulstijn, J., Riveret, R., Rotolo, A.: Characterising deadlines in temporal modal defeasible logic. In: Orgun, M.A., Thornton, J. (eds.) AI 2007. LNCS, vol. 4830, pp. 486–496. Springer, Heidelberg (2007)
7. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models though SESE Decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
8. Weber, I., Hoffmann, J., Mendling, J.: Semantic business process validation. In: SBPM (2008)
9. Wynn, M.T., Verbeek, H., van der Aalst, W.M., ter Hofstede, A.H., Edmond, D.: Business process verification - finally a reality! Business Process Management Journal (2007); preprint: QUT ePrint 9107
10. KPMG Advisory: The Compliance Journey: Balancing Risk and Controls with Business Improvement (2005)
11. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: Hung, P.C.K. (ed.) Proc. EDOC 2006, pp. 221–232. IEEE, Los Alamitos (2006)
12. Lu, R., Sadiq, S.W., Governatori, G.: Compliance aware business process design. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 120–131. Springer, Heidelberg (2008)

13. Liu, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Syst. J. 46, 335–361 (2007)
14. Goedertier, S., Vanthienen, J.: Designing compliant business processes with obligations and permissions. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 5–14. Springer, Heidelberg (2006)
15. Ghose, A., Koliadis, G.: Auditing business process compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)
16. Chopra, A.K., Sing, M.P.: Producing compliant interactions: Conformance, coverage and interoperability. In: Baldoni, M., Endriss, U. (eds.) DALT 2006. LNCS, vol. 4327, pp. 1–15. Springer, Heidelberg (2006)