Business Process Compliance is Hard

Silvano Colombo Tosatto[1], Guido Governatori[2],
Pierre Kelsen[1], and Leendert van der Torre[1]

[1] University of Luxembourg
silvano.colombotosatto@uni.lu
pierre.kelsen@uni.lu
leon.vandertorre@uni.lu
[2] NICTA
guido.governatori@nicta.com.au

Abstract.
In the present paper we analyze the complexity of a fragment of the compliance checking problem. Although the fragment studied leaves out many feature of the original problem, like compensations and non-structured processes, we prove that the complexity of such fragment is already **NP**-complete.

1 Introduction

Companies have to cope with the problem to prove that their business processes are compliant with the regulations given by the state or other relevant authorities.

Existing solutions tackling the problem or fragments of it have been provided by van der Aalst and Pesic [7], Goedertier and Vanthienen [2], Awad et al. [1], Hoffmann et al. [4] and many others.

The aim of the present paper is not to provide another solution for the problem, but to show that even by taking a small fragment of the compliance problem, the complexity of the problem is **NP**-complete. In particular we consider a fragment of the problem that checking partial compliance of structured processes with a set of regulations which do not allow compensations.

In the present paper we prove that this fragment of the compliance checking problem is **NP**-complete by using a proof by reduction. In the proof we reduce the *hamiltonian path* problem, a well known **NP**-complete problem.

The paper is structured as follows: Section 2 introduces the problems to which follows the complexity proof. Section 3 concludes the paper.

2 The problems

The PARTIAL COMPLIANCE CHECKING problem is:

Input: $\langle P, \circledcirc \rangle$ where P is a process and $\circledcirc$ is a set of punctual obligations
Question: Does P has at least one trace compliant with $\circledcirc$?

A process models a collection of methods to perform an activity. For instance an activity can be preparing coffee, a process modeling such activity would comprehend many ways to prepare coffee, such as using lyophilized coffee, brewing it, etc.

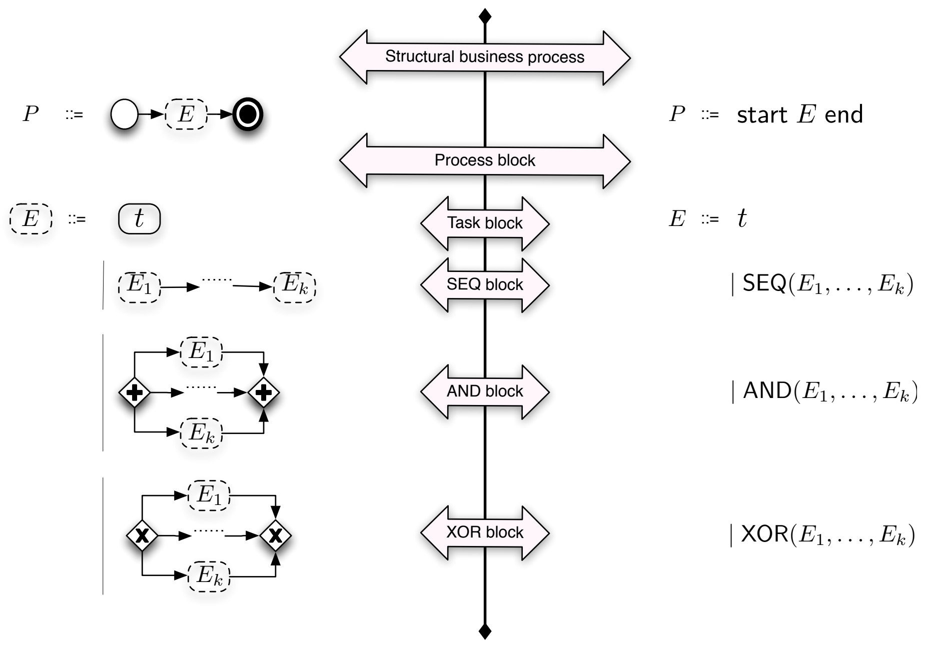
A process is composed by tasks and coordinators. Tasks represents the actions that can be done during the execution of a process. For instance considering the process that models how to prepare coffee, an activity can be to heat the water. Coordinators are used to define the valid executions of a process. For instance coordinators can define that a certain task has to be done before another one or which tasks are mutually exclusive. Arrows connecting the elements of a process identify a general order in which the elements can be executed.

To represent a process we use a fragment of BPMN[3]. The fragment considered uses only AND and XOR coordinators in addition to start and end. The AND coordinator is used to coordinate tasks which can be concurrently executed. The XOR is used to define which tasks are mutually exclusive. AND and XOR coordinators consist in blocks of tasks within the process which are enclosed between two coordinators of the same type.

We consider only processes which do not contain cycles and are structured. A process is structured if the blocks of the process are properly nested, meaning that if block b starts inside block a, b must be closed before a is. For more information on structured processes and their properties see, for example [5, 6].

Definition 1 (Process). *A structured business process P is a business process generated by the following grammar given in the format of an graphical extension of BNF:*

[3] Business Process Model Notation, Version 2.0, http://www.omg.org/spec/BPMN/2.0



P ::= start E end

E ::= t

| SEQ$(E_1, \ldots, E_k)$

| AND$(E_1, \ldots, E_k)$

| XOR$(E_1, \ldots, E_k)$

where $C = \{$ ○ , ◉ , ✚ , ✖ $\}$, T consists of all the tasks, i.e., $\boxed{t}$, and F is denoted by arrows.

The coordinator ○ is called start and the coordinator ◉ is called end. The coordinator ✚ is called ANDsplit in case of multiple outgoing arrows and ANDjoin in case of multiple incoming arrows. A pair of ANDsplit and ANDjoin coordinators groups a set of sub-blocks indicating a logical relationship to activate all the sub-blocks concurrently. Finally, the coordinator ✖ is called XORsplit in case of multiple outgoing arrows and XORjoin in case of multiple incoming arrows. A pair of XORsplit and XORjoin coordinators groups a set of sub-blocks indicating a logical relationship to activate exactly one of the sub-blocks, which is chosen arbitrarily.

We assume that all the tasks in a structured business process carry a distinct identity that constitutes a key part of the label of a task. Therefore, a task $\boxed{t}$ can directly be referenced by its label t. Similarly, (process) block identities are also distinct hence a block $\boxed{E}$ can directly be referenced by its label E. As a consequence, for simplicity, we also allow a textual way to reference the graphical representation of structured business processes as summarized to the right hand side of the corresponding syntactical constructs in Definition 1.

Example 1. In Fig. 1 we provide an example of a process containing four tasks labeled $t_1, \ldots, t_4$. Within the process it is shown an XOR block containing in different branches the tasks t_1 and t_2. The XOR block is nested within an AND block, forming one of its branches while task t_3 forms the other one. The AND

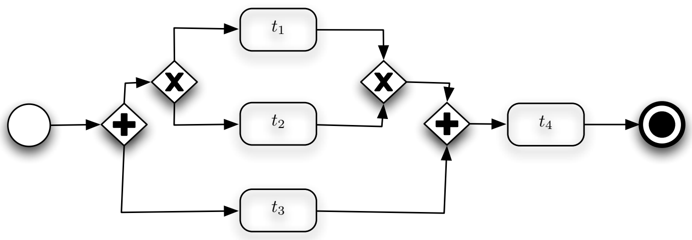


Fig. 1. Example of a process

block is preceded by the start coordinator and followed by task t_4 which in turn is followed by the end coordinator.

Fig. 2 contains two models which are not processes. Illustration **(a)** is not a process because of the two badly nested blocks. We can notice that the XOR block starts after the AND block but ends after the first one. Illustration **(b)** is not a process either because contains a loop between the tasks t_5 and t_4.

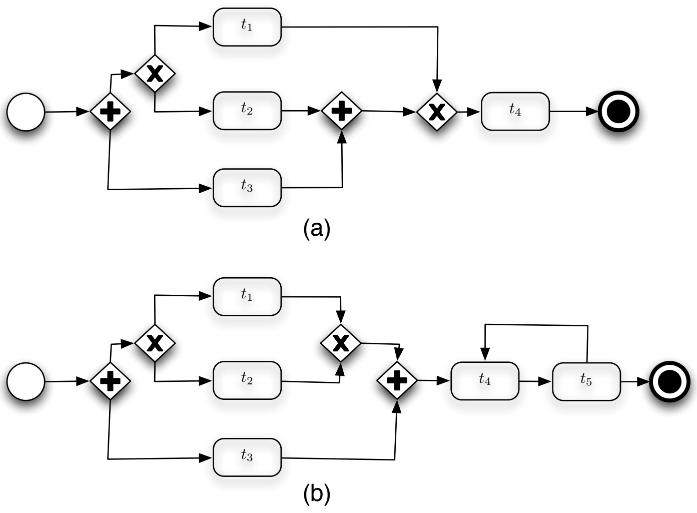


Fig. 2. Examples of non-processes

Given a process modeling an activity, an execution of such process represents one way to complete it. An execution is a valid serialization of a subset of tasks composing the process. A serialization is considered valid if it starts from the start coordinator and terminates at the end. In addition a valid serialization has to comply with the semantics of the coordinators and the connections between the tasks.

A process is defined as $P = \mathsf{start}\ E\ \mathsf{end}$. An execution of P is equivalent to executing the block E within start and end. Thus we will provide the formal semantics for executing blocks which can be used for executions as well.

Definition 2 (Block Execution). *A process block E can be serialized into a set of finite sequences of tasks , written $\Sigma(E)$, defined by the following structural recursion. We call each sequence in $\Sigma(E)$ an execution of E, ranged over by ϵ.*

1. *$E = t$: $\Sigma(E) = \{(t)\}$;*
2. *$E = \mathsf{SEQ}(E_1, \ldots, E_k)$: $\Sigma(E) = \{\epsilon_1; \ldots; \epsilon_k \mid \epsilon_1 \in \Sigma(E_1), \ldots, \epsilon_k \in \Sigma(E_k)\}$, where ; stands for sequence concatenation.*
3. *$E = \mathsf{XOR}(E_1, \ldots, E_k)$: $\Sigma(E) = \Sigma(E_1) \cup \ldots \cup \Sigma(E_k)$;*
4. *$E = \mathsf{AND}(E_1, \ldots, E_k)$: $\Sigma(E) = \{(t_1, \ldots, t_n)\}$ such that*
   (a) $\forall i, 1 \leq i \leq k, \exists \epsilon_i \in \Sigma(E_i)$ such that $\{t_1, \ldots, t_n\} = \bigcup_{1 \leq i \leq k} \mathsf{Tasks}(\epsilon_i)$
   (b) $\forall E_i \in E = \mathsf{AND}(E_1, \ldots, E_k), t_h, t_j \in E_i | t_h < t_j \to \forall \epsilon \in \Sigma(E), t_h > t_j$.
   Namely $\Sigma(E)$ is the set of sequences each of which merges a sequence of $\Sigma(E_1)$, ..., and of $\Sigma(E_k)$. Merging a set of sequences gives rise to a sequence that includes all the elements of the operand sequences. Moreover, the ordering in the result sequence should be compatible with the ordering in the operand sequences.

In an arbitrary process and its possible executions. If the process is conform with Definition 1, then a task belonging to the process appears in at least one of its executions. This means that each task contained in a process has the possibility to be executed as stated in the following lemma.

Lemma 1 (Block Execution). *Given a process block E and a task t in E, $\exists \epsilon \in \Sigma(E)$ such that $t \in \epsilon$.*

Proof. Prove by structural induction on E.

Example 2. Taking into account the process in Fig. 1 as $P = \mathsf{start}\ E\ \mathsf{end}$. We have that $\Sigma(E) = \{\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4\}$ where $\epsilon_1 = (t_1, t_3, t_4), \epsilon_2 = (t_2, t_3, t_4), \epsilon_3 = (t_3, t_1, t_4), \epsilon_4 = (t_3, t_2, t_4)$. $\Sigma(E)$ contains the four possible executions of the process P. An execution not contained in $\Sigma(E)$, like $\epsilon_5 = (t_3, t_4, t_1)$, is not a valid execution of P. In this particular case one of the reasons why ϵ_5 is not a valid execution is because after t_4 the task t_1 is executed, which is not possible because there is no arrow from t_4 to t_1 and the two tasks are not within an AND block.

The state of the process changes while executing the tasks. We represent the state of a process as an incomplete consistent set of literals. Given a language, a set is called incomplete if it is a subset of the literals belonging to the language. In a consistent set a literal and its negation cannot both be part of a state at the same time.

Definition 3 (Consistent literal set). *A set of literals L is* consistent *if and only if it does not contain l and $\tilde{l}$ for all literal $l \in L$.*

Example 3. In a language of literals containing $\{\alpha, \beta, \gamma\}$, the following states: $L_1 = \{\alpha, \neg\beta\}, L_2 = \{\neg\alpha, \neg\beta, \gamma\}, L_3 = \{\alpha, \neg\alpha, \beta\}$. L_1 is an incomplete state because does not contain either γ or its negation. L_1 is also consistent because it does not contain a literal and its negation. L_2 is a complete state because it contains all the literals belonging to the alphabet and L_3 in inconsistent because it contains both α and $\neg\alpha$.

Executing a task can change the current state of the process. Such changes depend on a consistent set of literals associated to the task being executed. We refer to a task with an associated set of literals associated as *annotated task*. The set of literals of an annotated task is seen as the postconditions of the execution of such task, which have to hold after the task is executed. A process containing annotated tasks is called an annotated process.

Definition 4 (Annotated process). *An annotated process is a pair: (P, ann), where P is a structured process and the set T contains the tasks in P. $\mathsf{ann}T \rightarrow 2^{\mathcal{L}}$, is a function from the tasks of P to consistent sets of literals.*

Example 4. Fig. 2 resumes the previous example shown in Fig. 1 including annotations for its tasks. Task's annotations represent which postconditions must hold in the state of the process after the task has been executed. In Fig. 3 we can see that after executing for instance t_1 the literal a has to hold in the state. Annotations are not limited to single literals: tasks t_2 and t_3 are both annotated by multiple literals.

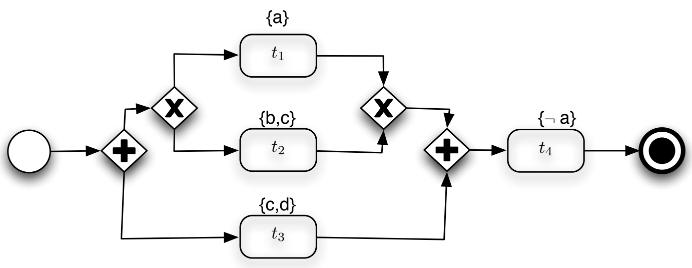


Fig. 3. Example of an annotated process

The execution state of a process has to be kept consistent. This means that after the execution of an annotated task, the literals in the set associated to such task must hold in the state but the state has to be kept consistent. To allow such behavior, before updating the current state we remove from it the literals which could cause inconsistencies with the ones introduced by the task execution. After this step the state can be updated by including the literals in the set of the annotated task. Being the set of literals introduced consistent by definition, the result is still a consistent set.

Definition 5 (Literal set update). *Given two consistent sets of literals L_1 and L_2, the update of L_1 with L_2 is a set of literals defined as follows:*

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

Example 5. This example shows how the state of a process is updated after executing a task. Given three sets of literals: $L_1 = \{a, b\}, L_2 = \{a, b, c\}$ and $L_3 = \{\neg c\}$. In case of $L_1 \oplus L_3$ the result is the set $\{a, b, \neg c\}$ which represent the union of L_1 and L_3. Differently if we consider $L_2 \oplus L_3$ the result is again $\{a, b, \neg c\}$ but this time the result in not equivalent to $L_2 \cup L_3$ because L_3 contains $\neg c$ that is the complement of one of the literals in L_2. The literal c is discarded from L_2 before joining it with L_3 so that the result is a consistent set. We can notice that $\oplus$ is not commutative because in the case $L_3 \oplus L_2$ the result would be $\{a, b, c\}$.

During one of its possible executions, a process typically goes through several states. Each of these states can be associated to the execution of one of the annotated tasks belonging to the execution. We call a *trace* such sequence of states and tasks.

Definition 6 (Trace). *The trace θ corresponding to an execution $\epsilon = (t_1, \ldots, t_k)$ of an annotated process (P, ann) is a finite sequence of pairs of the form $((t_1, L_1), \ldots, (t_k, L_k))$, where $L_1, \ldots, L_k$ are sets of literals such that:*

1. *$L_1 = \mathsf{ann}(t_1)$;*
2. *$L_{i+1} = L_i \oplus \mathsf{ann}(t_{i+1})$, for $1 \leq i < k$.*

We write $\Theta((P, \mathsf{ann}))$ to denote the set of traces of an annotated process.

$\Theta((P, \mathsf{ann}))$ and $\epsilon \in \Sigma(E)$ where $\forall \theta \in \Theta((P, \mathsf{ann}))\exists! \epsilon \in \Sigma(E) | \forall i, j | 1 \leq i, j \leq k$ and $t_i \in \theta$ and $t_j \in \epsilon, t_i = t_j$ iff $i = j$.

Example 6. This example shows the traces of the annotated process (P, ann) illustrated in Fig. 3. Taking into account the executions shown in Example 2 which corresponds to the possible executions of P where $P = \mathsf{start}\ E\ \mathsf{end}$. In the following table we show for each execution $\epsilon \in \Sigma(E)$ the corresponding trace $\theta \in \Theta((P, \mathsf{ann}))$. Each trace is represented as a sequence of pairs where every pair represents the task executed and the state holding after its execution.

ϵ	θ
(t_1, t_3, t_4)	$((t_1, \{a\}), (t_3, \{a, c, d\}), (t_4, \{\neg a, c, d\}))$
(t_2, t_3, t_4)	$((t_2, \{b, c\}), (t_3, \{b, c, d\}), (t_4, \{\neg a, b, c, d\}))$
(t_3, t_1, t_4)	$((t_3, \{c, d\}), (t_1, \{a, c, d\}), (t_4, \{\neg a, c, d\}))$
(t_3, t_2, t_4)	$((t_3, \{c, d\}), (t_2, \{b, c, d\}), (t_4 . \{\neg a, b, c, d\}))$

A trace is said to be compliant if it respects a given set of obligation $\circledcirc = \{\mathcal{O}_1, \dots \mathcal{O}_k\}$. We use a subset of Process Compliance Logic (PCL) [3] to specify the obligations belonging to $\circledcirc$. Each punctual obligation in $\circledcirc$ is a local obligation.

To each local obligation is associated a lifeline and a deadline. These two elements determine the duration of the obligation. The lifeline is used to define from which state of the trace a local obligation has to be verified. The deadline says when the validity of a local obligation ceases. States that satisfies the lifeline condition are not relevant if exists a previous state satisfying the same condition. This means that a lifeline does not raise an instance of the obligation but the obligation itself. However if a second state fulfilling the lifeline condition is encountered after such obligation has been terminated by a deadline or by other means, then the obligation is raised again. In any case if there is no state in a trace that fulfills the deadline condition of a local obligation, the last state of the trace is considered to fulfill every deadline condition.

A punctual obligation is a special case of two more general local obligations: achievement and maintenance.

Definition 7 (Local Obligations). *Let f_c, f_b and f_d be propositional formulas. A local obligation is a triple $\langle \mathcal{O}, f_b, f_d \rangle$ where f_b represents the lifeline condition, f_d represents the deadline condition and $\mathcal{O}$ is one of the following types of obligation the different types of obligations where f_c is the condition of the obligation:*

$$\begin{aligned} \mathcal{O} ::= &\ O^a(f_c) \ \textit{Atomic non-preemptive achievement} \\ \mid &\ O^m(f_c) \ \textit{Atomic maintenance} \end{aligned}$$

The formulas representing the lifeline, deadline and condition of an atomic obligation are true in a given state of a trace if considering the literals belonging to such state as true, the truth value of the formula is true.

The condition of an achievement obligation, has to be verified in at least one state of the trace between the lifeline and the deadline of such obligation. An achievement obligation is violated if no state before the deadline satisfies the condition. Achievement obligations terminate not only when the deadline is encountered but also when the condition of the obligation is satisfied. To represent this behavior we consider that the formula representing the deadline for achievement obligation is always verified when the condition of the obligation is.

For maintenance obligations, every state between the lifeline and the deadline has to fulfill the condition. A maintenance obligation is violated as soon as a state does not fulfill the condition.

Definition 8 (Local Obligation Fulfillment). *Let $L \models f$ be true iff the truth value of f is true given as true each $l \in L$. Given a local obligation $\langle \mathcal{O}, f_b, f_d \rangle$ and a trace $\theta = ((t_1, L_1), \dots, (t_k, L_k))$, θ fulfills $\langle \mathcal{O}, f_b, f_d \rangle$ ($\theta \vdash \langle \mathcal{O}, f_b, f_d \rangle$) iff:*

$\mathcal{O} = O^a(L_c)$ $\theta \vdash \langle O^a(f)_c, f_b, f_d \rangle$ *iff* $\forall L_i \in \theta | L_i \models f_b \exists L_j \in \theta | L_j \models f_c$ *and* $L_j > L_i$ *and* $\neg \exists L_h \in \theta | L_h \models f_d$ *and* $L_i < L_h < L_j$.

$\mathcal{O} = O^m(L_c)$ $\theta \vdash \langle O^m(f)_c, f_b, f_d\rangle$ *iff* $\forall L_i \in \theta | L_i \models f_b, \exists L_h \in \theta | L_h \models f_d$ *or* $L_h = L_k \in \theta, \forall L_j \in \theta | L_i < L_j \leq L_h, L_j \models f_c$.

Example 7 (Achievement). In a scenario where a costumer dines in a restaurant, there is the obligation that after he or she orders, he or she has to pay the bill before leaving.

Example 8 (Maintenance). While accessing secure data there is the obligation to have the proper credentials for the whole period while accessing the data.

Punctual obligations are characterized by the fact that they remain valid for exactly one state. This means that the deadline of such type of local obligations is satisfied by the state which occurs after the lifeline. Punctual obligations are a particular kind of both local achievement and maintenance obligations. Achievement and maintenance obligations behave in the same way if the deadline occurs in the state following the lifeline. Namely both obligations has to achieve their condition in the only state in which the obligation is valid.

Definition 9 (Punctual Obligation Fulfillment). *Given a punctual obligation* $\langle O^p(f_c), f_b, \top\rangle$ *and a trace* $\theta = ((t_1, L_1), \ldots, (t_k, L_k))$, θ *fulfills* $\langle O^p(f_c), f_b, \top\rangle$ $(\theta \vdash \langle O^p(f_c), f_b, \top\rangle)$ *iff:*

$$\forall L_i \in \theta | L_i \models f_b, L_{i+1} \models f_c$$

A trace is considered to be compliant with a set of obligations if it fulfills all the obligations belonging to the set.

Definition 10 (Set Compliance). *Given a trace* θ *and a set of obligations* $\circledcirc$, θ *is compliant with* $\circledcirc$ $(\theta \vdash \circledcirc)$ *iff:*

$$\forall \langle \mathcal{O}_i, f_b, f_d\rangle \in \circledcirc, (\theta \vdash \langle \mathcal{O}_i, f_b, f_d\rangle)$$

Given a set of obligations, an annotated process can be fully compliant, partially compliant or not compliant with such set. An annotated process is fully compliant if each trace that can be generated from such process is compliant with the set of obligations. In case of partial compliance at least one trace has to be compliant with the set of obligations. If none of the traces that can be generated from an annotated process are compliant with the set of obligations, then such annotated process is not compliant.

Definition 11 (Process Compliance). *Given an annotated process* (P, ann) *and a set of obligations* $\circledcirc$.

Full Compliance $(P, \mathsf{ann}) \vdash^{F} \circledcirc$ *if* $\forall \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc$.
Partial Compliance $(P, \mathsf{ann}) \vdash^{P} \circledcirc$ *if* $\exists \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc$.
Not Compliant $(P, \mathsf{ann}) \not\vdash \circledcirc$ *if* $\neg\exists \theta \in \Theta(P, \mathsf{ann}), \theta \vdash \circledcirc$.

According to the definition of process compliance, we can notice that full compliance implies partial compliance. This means that if a process is full compliant with a set of obligations, then such process is also partial compliant with the same set.
The HAMILTONIAN PATH problem is:

Input: $\langle G \rangle$ where G is a graph
Question: Does G has an hamiltonian path?

In a graph $G = (N, D)$ where N is a set of nodes n_i and D is a set of directed edges represented as a binary relation $N \times N$.

An hamiltonian path is a path in a directed graph that visits each node of the graph exactly once. A path can travel from one node to another only if exists a directed edge between starting from a node and pointing to the one following it in the path.

Definition 12 (Hamiltonian Path). *An hamiltonian path* $ham = (n_1; \ldots; n_m)$ *satisfies the three following properties:*

1. $\forall n_i \in ham, n_i \in N$
2. $\forall i, j((n_i, n_j \in ham), (n_i \neq n_j))$
3. $\forall i, j((n_i, n_j \in ham \wedge j = i + 1), ((n_i, n_j) \in D)$

Claim. PARTIAL COMPLIANCE CHECKING is in **NP**.

Proof. The following algorithm verifies if a trace is compliant with a set of punctual obligations and runs in time polynomial in the...

Algorithm 1 *Let* Ob *be the set of obligations which lifeline has been triggered and have to be verified, given a set of punctual obligations* $\circledcirc$ *and a trace* $\theta = ((t_1, L_1), \ldots, (t_k, L_k))$ *such that* $\theta \in \Theta(P, \mathsf{ann})$*, the algorithm* $A(\theta, \circledcirc)$ *is defined as follows:*

```
1:  $\mathsf{Ob} = \emptyset$;
2:  for each $(t_i, L_i)$ in $\theta$ do
3:      for each $\langle O^p(f_c), f_b, \top \rangle$ in $\mathsf{Ob}$ do
4:          if $L_i \not\models f_c$ then
5:              return $\theta \not\vdash \circledcirc$;
6:          end if
7:      end for each
8:      $\mathsf{Ob} = \emptyset$;
9:      for each $\langle O^p(f_c), f_b, \top \rangle$ in $\theta$ do
10:         if $L_i \models f_b$ then
11:             $\mathsf{Ob} = \mathsf{Ob} \cup \langle O^p(f_c), f_b, \top \rangle$
12:         end if
13:     end for each
14: end for each
15: return $\theta \vdash \circledcirc$;
```

According to Definition 11, if exists a trace of $\Theta(P, \mathsf{ann})$ which is compliant, then the process (P, ann) is partial compliant.

COMPLEXITY OF algorithm 1

The **NP**-hardness of PARTIAL COMPLIANCE CHECKING is implied by the following.

Claim. HAMILTONIAN PATH $\leq_p$ PARTIAL COMPLIANCE CHECKING

Proof. **Reduction to Partial Compliance Problem**

We reduce the hamiltonian path problem composed by a graph $G = (N, D)$ to a partial compliance problem composed by an annotated process (P, ann) as shown in Figure 4 and a set of punctual obligations $\circledcirc$.

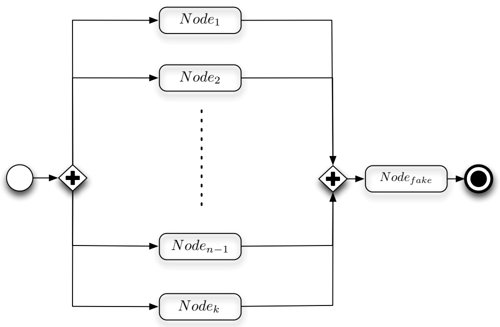


Fig. 4. Hamilton path problem as a structured process.

The process P is structured as a sequence block containing two elements: an AND block followed by a task. The AND block contains for each branching a single task $Node_i$ representing one of the nodes n_i in N. The task following the AND block, named $Node_{fake}$, has no correspondence with the original graph G. The task $Node_{fake}$ is annotated in such a way that is capable to fulfill every obligation belonging to $\circledcirc$ without triggering any lifeline. Without such auxiliary task, every obligation which lifeline is triggered by the last task executed in the AND block, would be always violated because the end coordinator would follow such task.

Let $\theta = ((t_1, L_1), \ldots (t_k, L_k))$ be a trace, where L_i is the state holding after the execution of t_i. We represent with $L_0 = \emptyset$ the state holding before the execution of the first task. The annotation of each task in (P, ann) is the following:

- $\forall i | 1 \leq i \leq k, \mathsf{ann}(Node_i) = \{\bigwedge_{j=1}^{i-1}(\neg l_j) \wedge l_i \wedge \bigwedge_{z=i+1}^{k}(\neg l_z)\}$
- $\mathsf{ann}(Node_{fake}) = \{\bigwedge_{j=1}^{n}(\neg l_j)\}$

The set of obligations $\circledcirc$ contains the following punctual obligations:

– $\forall n_i \in N, \forall n_j | (n_i, n_j) \notin D, \langle O^p(\neg l_j), l_i, \top\rangle$

We claim that if exists a trace $\theta \in \Theta(P, \mathsf{ann})$ such that $\theta \vdash \circledcirc$, which means that $(P, \mathsf{ann}) \vDash^{\mathrm{P}} \circledcirc$ then G has an hamiltonian path.

Proof

In this section we prove the soundness ($(P, \mathsf{ann}) \vDash^{\mathrm{P}} \circledcirc \Rightarrow \exists ham$) and completeness ($\exists ham \Rightarrow (P, \mathsf{ann}) \vDash^{\mathrm{P}} \circledcirc$) of our reduction. We refer to the three conditions stated in Definition 12 as (1),(2) and (3) respectively.

$(P, \mathsf{ann}) \vDash^{\mathrm{P}} \circledcirc \Rightarrow \exists ham$

Direct proof:

1. By construction of the reduction, each task in P except $Node_{fake}$ is associated to a node of G, thus due to the structure of P each task has to be executed (Definition 2), fulfilling condition (1) because each task represent a node. Condition (2) is also fulfilled because each task can be executed only once (Definition 2).
2. From the hypothesis $(P, \mathsf{ann}) \vDash^{\mathrm{P}} \circledcirc$ and Definition 11 we have that $\exists \theta \in \Theta(P, \mathsf{ann})$ and $\theta \vdash \circledcirc$, where $\theta = ((t_1, L_1), \ldots, (t_k, L_k), (Node_{fake}, L_{k+1})))$ from Definition 6.
3. By construction of the reduction we have that each obligation in the set is a punctual obligation: $\forall \mathcal{O} \in \circledcirc$, $\mathcal{O} = \langle O^p(\neg l_c), l_b, \top\rangle$, where $\langle O^p(\neg l_c), l_b, \top\rangle \in \circledcirc$ iff $(n_b, n_c) \notin D$.
4. Taking into account a task t_j that follows t_i in θ, by construction of the reduction we have that each task have annotated a positive literal for the node that it represents and a negative literal for each other node in N. From 3. follows that a task t_j following t_i in θ violates an obligation in $\circledcirc$ iff $(n_i, n_j) \notin D$.
5. Following from 2. we have that $\forall \langle O^p(\neg l_c), l_b, \top\rangle \in \circledcirc$, $\theta \vdash \langle O^p(\neg l_c), l_b, \top\rangle$. Thus none of the obligations in $\circledcirc$ are violated. This means that $\forall i | i \leq 1 \leq k-1, t_i \in \theta$ and $t_{i+1} \in \theta$ then $\exists (n_x, n_y) \in D$ where n_x is represented by t_i and n_y is represented by t_{i+1}. From this follows that condition (3) is satisfied.
6. Following from 1. and 5. we have that the three conditions in Definition 12 are satisfied if $(P, \mathsf{ann}) \vDash^{\mathrm{P}} \circledcirc$, thus $\exists ham$.

$\exists ham \Rightarrow (P, \mathsf{ann}) \vDash^{\mathrm{P}} \circledcirc$

Direct proof:

1. From the hypothesis we know that $\exists ham = (n_1; \ldots; n_m)$ satisfying the three conditions of Definition 12. Thus $\forall i | 1 \leq i \leq m-1, \exists (n_i, n_{i+1}) \in D$ due to condition (3).

2. From construction of the reduction and following from 1. being t_x the task in P associated to n_i and t_y the task associated to n_j, we have that for any $(n_i, n_j) \in D$, $\neg\exists\langle O^p(\neg l_y), l_x, \top\rangle \in \circledcirc$ where l_y belongs to the annotation of t_y and l_x to the annotation of t_x.
3. Let $\theta = ((t_1, L_1), \ldots, (t_k, L_k), (Node_{fake}, L_{k+1})))$ belonging to $\Theta(P, \mathsf{ann})$, where t_1 is associated to n_1 in ham and so on till t_k associated to n_m. Due to the structure of P we can see that θ is a valid trace for P.
4. From 2. and 3. we can conclude that for every task t_j following t_i in θ there are no obligations in $\circledcirc$ violated because exists a directed edge from the node associated to t_i to the node associated to t_j in D. Thus because no obligations in $\circledcirc$ are violated $\theta \vdash \circledcirc$.
5. From 4. and Definition 11 we can conclude that $(P, \mathsf{ann}) \vdash^{\mathrm{P}} \circledcirc$.

3 Conclusion

Even by considering a small fragment of the compliance checking problem, in the present paper we prove that the complexity of the problem is **NP**-complete. Thus for instances of the compliance checking problem involving complex processes and or large sets of local obligation with a compensation associated, computing the solution can be infeasible due to the complexity of the problem.

Time feasible solutions could be still found for even smaller fragments of the problem. As further work we plan to investigate such fragments.

Acknowledgements

– NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.
– Silvano Colombo Tosatto is supported by the National Research Fund, Luxembourg.

References

1. 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.
2. Stijn Goedertier and Jan Vanthienen. Designing compliant business processes with obligations and permissions. In *Business Process Management (BPM) Workshops*, pages 5–14, 2006.
3. 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.

4. 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.
5. Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Christoph Bussler. On structured workflow modelling. In Benkt Wangler and Lars Bergman, editors, *Advanced Information Systems Engineering (CAiSE 2000)*, volume 1789 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2000.
6. Artem Polyvyanyy, Luciano García-Bañuelos, and Marlon Dumas. Structuring acyclic process models. In Richard Hull, Jan Mendling, and Stefan Tai, editors, *Business Process Management (BPM 2010)*, volume 6336 of *Lecture Notes in Computer Science*, pages 276–293. Springer, 2010.
7. 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.