Fundamenta Informaticae XXI (2001) 1001–1025
DOI 10.3233/FI-2016-0000
IOS Press

Compliance in Business Processes with Incomplete Information and Time Constraints: a General Framework based on Abductive Reasoning*

Federico Chesani, Paola Mello
University of Bologna
viale Risorgimento 2, 40136–Bologna, Italy
{federico.chesani | paola.mello}@unibo.it

Marco Montali*, Sergio Tessaris
Free University of Bozen–Bolzano
piazza Università, 1, 39100 Bozen-Bolzano, Italy
{montali | tessaris}@inf.unibz.it

Riccardo De Masellis†, Chiara Di Francescomarino*, Chiara Ghidini*
FBK-IRST
Via Sommarive 18, 38050 Trento, Italy
{r.demasellis | dfmchiara | ghidini}@fbk.eu

Abstract. The capability to store data about Business Process (BP) executions in so-called Event Logs has brought to the identification of a range of key reasoning services (consistency, compliance, runtime monitoring, prediction) for the analysis of process executions and process models. Tools for the provision of these services typically focus on one form of reasoning alone. Moreover, they are often very rigid in dealing with forms of incomplete information about the process execution. While this enables the development of ad hoc solutions, it also poses an obstacle for the adoption of reasoning-based solutions in the BP community.

In this paper, we introduce the notion of Structured Processes with Observability and Time (SPOT models), able to support incompleteness (of traces and logs), and temporal constraints on the activity duration and between activities. Then, we exploit the power of abduction to provide a flexible, yet computationally effective framework able to reinterpret key reasoning services in terms of incompleteness and observability in a uniform way.

Address for correspondence: Federico Chesani, University of Bologna, viale Risorgimento 2, 40136–Bologna, Italy. E-Mail: federico.chesani@unibo.it

*This work is an extended version of a preliminary, position paper presented at the ECAI2016 conference [1].

†This research has partially been carried out within the Euregio IPN12 KAOS, which is funded by the "European Region Tyrol-South Tyrol-Trentino" (EGTC) under the first call for basic research projects.

Keywords: Business Processes, Incomplete traces, Observability, Temporal workflows, Abductive Logic Programming

1. Introduction

The proliferation of IT systems able to store process executions traces in so-called event logs has originated, in the Business Process (BP) community, a quest towards tools that offer the possibility of discovering, checking the conformance and enhancing process models based on actual behaviors [2]. Focusing on conformance, that is, on a scenario where the aim is to assess how a *prescriptive* (or "de jure") process model relates to the execution traces, this general notion can be declined in specific "use cases", such as *model consistency*, *trace compliance*, *runtime monitoring* and *prediction/recommendation* [3]. These reasoning services are often investigated in isolation and tailored to specific workflow languages. Thus [4] assesses runtime monitoring over processes with business constraints, while [5] assesses model consistency with business contracts.

While the construction of tools that offer specific reasoning services over specific workflow languages enables the development of ad hoc effective solutions, it also poses an obstacle for the adoption of reasoning-based solutions in the BP community. In fact, these tools often cover only few of the "use cases" and do not easily adapt to different workflow languages. This poses a problem, given the current trend of enriching BP languages with new constructs complementing the control flow knowledge. A paradigmatic example is the recent work of [6] where runtime monitoring/prediction (only) is investigated "ex-novo" on temporal workflows, i.e., workflows enriched with constructs for temporal constraints.

A second rigidity of current reasoning services is in dealing with *incomplete information* about the process execution. In fact, they either target ongoing process executions as in the case of run-time monitoring and recommendation, or they require complete end-to-end execution traces as in the case of compliance. This is another obstacle towards reasoning-based solutions in the BP community. In fact, the presence of not monitorable activities or errors in the logging procedure makes the ability of handling *incomplete event data* one of the main challenges of the BP community, as mentioned in the *process mining manifesto* [2].

Artificial Intelligence has a long tradition providing frameworks able to integrate diverse reasoning tasks in the presence of incomplete information. A paradigmatic example is Abductive Logic Programming (ALP) [7], where the integration of constraint solving features (ACLP) [8] has enhanced its practical utility by allowing expressive and flexible representations of the problem domain, and by making the abductive computation more efficient and powerful. Abduction fits in a natural manner with the scenario of execution traces: facts are observed in the execution traces, and need to be explained/diagnosed with respect to what is envisaged by the process model. This is indeed strictly related with the definition of abductive reasoning.

In this work we exploit the paradigm of ACLP [8], and the SCIFF abductive framework [9] to provide a general purpose environment able to support *conformance in its different "use cases"* in the presence of *incomplete event data* and *temporal constraints* on activity durations and between different activities. The novel contribution of the paper is as follows. First, we provide a formal definition of the scenario at hand, by introducing the notion of Structured Processes with Observablity

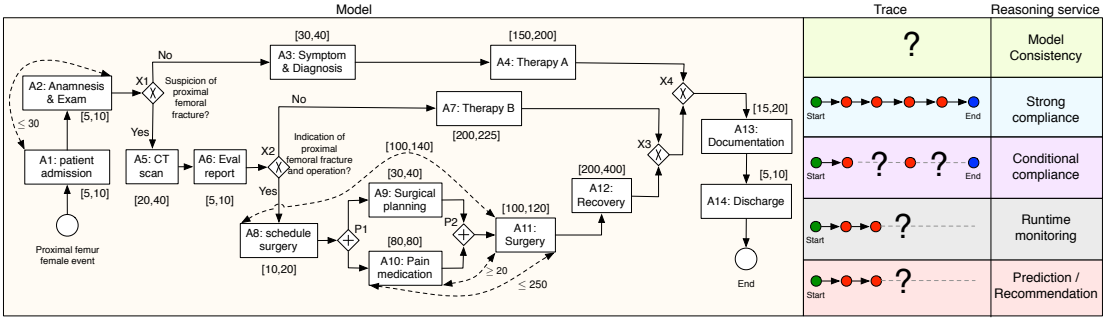

Figure 1. A process for femoral fracture treatment taken from [6], and reasoning services and incomplete execution traces.

and Time (SPOT models), and by reformulating business process reasoning services in terms of incompleteness (Sec. 2). This produces a refinement of the classical notion of compliance into **strong**, and **conditional** compliance to take into account different sources of incomplete knowledge in the analysed traces. Second, an encoding of SPOT models and event logs in SCIFF is provided. To show the flexibility of our approach to capture different workflow languages in a modular manner, we address structured process models enriched with temporal aspects (Sec. 3.1). The SCIFF proof procedure is then exploited and evaluated in Sec. 3.2, 3.3, and 4.

2. Process Models, Reasoning Services and Incompleteness

Given the *prescriptive* knowledge contained in a well-structured process model[1] enriched with temporal constraints, we aim at understanding how to re-interpret typical reasoning services as reasoning on incomplete traces.

2.1. Process Models and Incomplete Traces

We illustrate our investigation with an example of temporal workflow taken from [6], and reproduced in the left hand side of Figure 1. This workflow contains: 14 activities ($\mathsf{A1}, \ldots \mathsf{A14}$), 2 pairs of exclusive gateways ($\langle \mathsf{X1}, \mathsf{X4} \rangle$, $\langle \mathsf{X2}, \mathsf{X3} \rangle$), and 1 pair of parallel gateways ($\langle \mathsf{P1}, \mathsf{P2} \rangle$). In addition, the language contains constructs to denote temporal information: activities are labelled with expressions of the form $[d_{min}, d_{max}]$ indicating the *duration range* of the activity[2], and dashed arrows between activities expresses *inter-task constraints* involving the start or end of the activities; depending on the position of the arrow w.r.t. the box representing the activity. As an example, the dashed arrow between A1 and A2 indicates that a constraint of at most 30 time units exists between the start of A1 and the end of A2.

[1]We focus on structured process models in the spirit of [10]. Broadly speaking, this restricts to the class of models recursively composed of single-entry-single-exit blocks, where every split has a corresponding join, matching its type. This assumption rules out pathological patterns that are notoriously hard to characterise (e.g. involving nested OR joins), still providing coverage for a wide range of interesting use cases.

[2]In the remainder of this paper we will assume that the time domain relies on natural numbers. Alternative temporal domains can be seamlessly studied.

We assume that each execution of this process, hereafter called the Femur-Fracture (FF) process, is logged by an information system. We also assume that activities have a time span e.g., event $(\mathsf{A}, [T_s, T_e])$ indicates that activity A has been executed from T_s to T_e. A sample trace that logs the execution of a FF instance is:

$$\{(\mathsf{A1}, [2, 7]),(\mathsf{A2}, [10, 15]), (\mathsf{A3}, [16, 46]), (\mathsf{A4}, [50, 200]), (\mathsf{A13}, [300, 317]), (\mathsf{A14}, [320, 330])\} \quad (1)$$

A multiset of traces of the same process forms an event log.

In the aforementioned trace, all information have been explicitly logged. To incorporate incompleteness into the picture, the process models we consider are also equipped with *observability information*. This information provides a fine-grained characterization of different levels of incompleteness in the activities present in the process model, when it comes to "logging" the execution of such activities when running the process. In particular, three observability categories are considered:

- An *observable* activity is an activity that is explicitly logged. Here incompleteness may arise because some information (i.e., the activity name, the starting time, the ending time, or a combination thereof) may be missing.
- A *non-observable* activity is an activity that is never logged. Here incompleteness arises because such activity could have been actually executed, even though this is not traced in the log.
- A *partially observable* activity is an activity that may or may not be logged, depending on the context. In other words, some traces may contain an explicitly logged entry witnessing the execution of such an activity, while in others this could be missing.

Formally, we extend the approach in [6] with observability information, obtaining the following model.

Definition 2.1. (Structured Process with Observability and Time)
A *structured process with observability and time* (SPOT) is a tuple $\langle \mathcal{A}, obs, P, dur, tcon \rangle$, where:

- $\mathcal{A}$ is a finite set of *activity (names)*;
- $obs : \mathcal{A} \to \{\mathtt{o}, \mathtt{n}, \mathtt{p}\}$ is a total *observability function*, indicating for each activity in $\mathcal{A}$ whether it is observable (o), non-observable (n), or partially observable (p).
- P is the *top process block*, where a process block is inductively defined as follows:
  (base case) $\mathtt{a} \in \mathcal{A}$ is a process block, called *task block*;
  (inductive case) $\langle type, B_1, B_2 \rangle$ is a process block, where $type \in \{\mathtt{seq}, \mathtt{xor}, \mathtt{and}, \mathtt{or}\}$ indicates the block type (where seq stands for *sequence*, xor for *exclusive choice*, and for *parallel split*, and or for *inclusive split*), while B_1 and B_2 are process blocks.
- $dur : \mathcal{A} \to \mathbb{N} \times \mathbb{N} \cup \{\infty\}$ is a total *duration function*, assigning a duration interval to each activity in $\mathcal{A}$, where $dur(\mathtt{a}) = \langle m, n \rangle$ means that the duration of a is between m and n time units. We assume that for each activity $\mathtt{t} \in \mathcal{A}$ with $dur(\mathtt{t}) = \langle m, n \rangle$, n is either ∞ or a number $\geq m$. We use ∞ as a special symbol to indicate that the maximum duration is unconstrained.[3]
- $tcon : \mathcal{A} \times \{\mathfrak{s}, \mathfrak{e}\} \times \mathcal{A} \times \{\mathfrak{s}, \mathfrak{e}\} \to \mathbb{N} \times \mathbb{N} \cup \{\infty\}$ is a partial *inter-task constraint function*, indicating the expected duration interval that can intervene between the start/end of an activity and the start/end of another activity, where the start is marked with symbol $\mathfrak{s}$, and the end is marked with symbol $\mathfrak{e}$. E.g., $tcon(\mathtt{a}_1, \mathfrak{e}, \mathtt{a}_2, \mathfrak{s}) = \langle m, n \rangle$ if the time duration intervening between the end of $\mathtt{a}_1$ and the start of $\mathtt{a}_2$ belongs to the time interval $[m, n]$.

[3]Thus, if activity a has unconstrained duration, we have $dur(\mathtt{a}) = \langle 0, \infty \rangle$.

We assume that P is well-defined, in the sense that each activity $\mathtt{a} \in \mathcal{A}$ appears in *exactly one block*. This is a standard assumption guaranteeing that no process block can directly or indirectly refer to itself, and also that each task is unambiguously located within the process. It also implies that the sub-block relation induces a tree rooted in P and whose leaves are activities from $\mathcal{A}$. We say that block B_1 *is in* block B_2 if B_1 belongs to a sub-tree of such tree that is rooted in B_1.

Notice that the definition of blocks may be directly extended to the case of n sub-blocks instead of just 2. In particular, in the following we make use of a ternary sequence block $\langle \mathtt{seq}, B_1, B_2, B_3 \rangle$.Furthermore, it is well-known that an inclusive split block can be reformulated in terms of a choice considering the execution of the first inner-block only, the execution of the second inner-block only, or the parallel execution of both inner-blocks. Hence, from now on we will omit the case of inclusive blocks. The lack of loop blocks in Definition 2.1 is due to the the unclear semantics of the interplay between loops and inter-task constraints. In fact, loops are not present in temporal workflows [6, 11].

Example 2.2. Consider the fragment of temporal workflow constituted by the first two tasks in Figure 1, so that the first task is observable, whereas the second is not. It corresponds to SPOT $\langle \mathcal{A}, obs, P, dur, tcon \rangle$, where: *(i)* $\mathcal{A} = \{\mathsf{A1}, \mathsf{A2}\}$, *(ii)* obs is such that $obs(\mathsf{A1}) = \mathtt{o}$ and $obs(\mathsf{A2}) = \mathtt{n}$, *(iii)* $P = \langle \mathtt{seq}, \mathsf{A1}, \mathsf{A2} \rangle$, *(iv)* dur is such that $dur(\mathsf{A1}) = dur(\mathsf{A2}) = \langle 5, 10 \rangle$, *(v)* $tcon$ is such that $tcon(\mathsf{A1}, \mathfrak{s}, \mathsf{A2}, \mathfrak{e}) = \langle 0, 30 \rangle$.

From now on, we assume that the SPOT of interest is structured in the following *normal form*.

Definition 2.3. (Normal SPOT)
A SPOT $\langle \mathcal{A}, obs, P, dur, tcon \rangle$ with $\mathcal{A} = \mathcal{A}_a \uplus \mathcal{A}_b$ is in *normal form* (and, hence, a *normal SPOT*) if:
- Task blocks in P use only activities from $\mathcal{A}_a$.
- Set $\mathcal{A}_b$ is the smallest set satisfying the following condition: for each block B in P, $\mathcal{A}_b$ contains two special, atomic activities in_B and out_B.
- For every activity $\mathtt{b} \in \mathcal{A}_b$, we have that $obs(\mathtt{b}) = \mathtt{n}$, and $dur(\mathtt{b}) = \langle 0, 0 \rangle$.
- P has the form $\langle \mathtt{seq}, \mathsf{in}_P, P', \mathsf{out}_P \rangle$, and every inductive block of the form $\langle type, B_1, B_2 \rangle$ in P' is such that B_i has the form $\langle \mathtt{seq}, \mathsf{in}_{B_i}, B_i', \mathsf{out}_{B_i} \rangle$ for $i \in \{1, 2\}$.
- Function $tcon$ only mentions activities in $\mathcal{A}_a$.

Intuitively, in a normal SPOT each block is associated to two special activities that mark the entry and exit points into/outside the block. Such activities are atomic and nonobservable, and are used to "interleave" the nesting of blocks.

We next define the notion of trace, taking into account the presence of missing information units, denoted with the special symbol "_".

Definition 2.4. ((SPOT) trace)
A *(SPOT) trace* over a set $\mathcal{A}$ of activities is a finite set of event triples $\langle \mathtt{a}, s, e \rangle$, where $\mathtt{a} \in \mathcal{A} \cup \{\_\}$ is an activity, and $s, e \in \mathbb{N} \cup \{\_\}$ are the start and end timestamps of the event. A trace is *partially specified* if some of its event triples contain the special symbol "_", *fully specified* otherwise. We require fully specified traces to avoid repetitions of activities: given a fully specified trace $\mathcal{T}$ over $\mathcal{A}$, every activity $\mathtt{a} \in \mathcal{A}$ appears at most once in $\mathcal{T}$.

Absence of activity repetitions reflect the fact that SPOTs do not contain loops, and guarantee that each activity is employed in at most one task block.

As in our running example, we represent event triples using notation $(\mathsf{a}, [s, e])$. When a is atomic, i.e., $s = e$, we simply write (a, s). We say that activity a *occurs in* trace $\mathcal{T}$ if there exist $s, e \in \mathbb{N}$ such that $(\mathsf{a}, [s, e]) \in \mathcal{T}$. Notice that, independently of whether a trace is fully or partially specified, there is another dimension of incompleteness, which accounts for the distinction between *total* and *partial traces*. In a partial trace, additional event triples may be implicitly present even if not explicitly listed in the trace. This is the case when the process model of interest contains activities that are not always observable.

2.2. Compliance

Several fundamental reasoning services can be studied over SPOTs (cf. the right-hand side of Figure 1).

We start by defining a basic notion of trace compliance, which mirrors the intended execution semantics of SPOTs. Recall that we are employing the normal form for SPOTs.

Definition 2.5. (Block compliance)
Let $\mathcal{A}$ be a set of activities. A fully specified trace $\mathcal{T}$ over $\mathcal{A}$ *complies with* a block $B = \langle \mathtt{seq}, \langle \mathsf{in}_B, B', \mathsf{out}_B \rangle \rangle$ if either both activities in_B and out_B occur in $\mathcal{T}$ or none does, and one of the following conditions hold:

1. *(i)* $B' = \mathsf{a}$ with $\mathsf{a} \in \mathcal{A}$, *(ii)* in_B occurs in $\mathcal{T}$ if and only if a occurs in $\mathcal{T}$, *(iii)* if there exist $t_i, t_s, t_e, t_o \in \mathbb{N}$ such that $\{(\mathsf{in}_B, t_i), (\mathsf{a}, [t_s, t_e]), (\mathsf{out}_B, t_o)\} \subseteq \mathcal{T}$, then $t_i \leq t_s \leq t_e = t_o$.
2. *(i)* $B' = \langle \mathtt{seq}, B_1, B_2 \rangle$, *(ii)* B_1 and B_2 are compliant with $\mathcal{T}$, *(iii)* in_B occurs in $\mathcal{T}$ if and only if in_{B_1} occurs in $\mathcal{T}$ if and only if in_{B_2} occurs in $\mathcal{T}$, *(iv)* if there exist $t_i, t_{i1}, t_{o1}, t_{i2}, t_{o2}, t_o \in \mathbb{N}$ such that $\{(\mathsf{in}_B, t_i), (\mathsf{in}_{B_1}, t_i), (\mathsf{out}_{B_1}, t_{i1}), (\mathsf{in}_{B_2}, t_{o1}), (\mathsf{out}_{B_2}, t_o), (\mathsf{out}_B, t_{o2})\} \subseteq \mathcal{T}$, then $t_i = t_{i1} \leq t_{o1} = t_{i2} \leq t_{o2} = t_o$.
3. *(i)* $B' = \langle \mathtt{and}, B_1, B_2 \rangle$, *(ii)* B_1 and B_2 are compliant with $\mathcal{T}$, *(iii)* in_B occurs in $\mathcal{T}$ if and only if in_{B_1} occurs in $\mathcal{T}$ if and only if in_{B_2} occurs in $\mathcal{T}$, *(iv)* if there exist $t_i, t_{i1}, t_{o1}, t_{i2}, t_{o2}, t_o \in \mathbb{N}$ such that $\{(\mathsf{in}_B, t_i), (\mathsf{in}_{B_1}, t_i), (\mathsf{out}_{B_1}, t_{i1}), (\mathsf{in}_{B_2}, t_{o1}), (\mathsf{out}_{B_2}, t_o), (\mathsf{out}_B, t_{o2})\} \subseteq \mathcal{T}$, then $t_i = t_{i1} = t_{i2} \leq t_{o1}$, $t_i \leq t_{o2}$, and $t_o = \max(t_{o1}, t_{o2})$.
4. *(i)* $B' = \langle \mathtt{xor}, B_1, B_2 \rangle$, *(ii)* B_1 and B_2 are compliant with $\mathcal{T}$, *(iii)* in_B occurs in $\mathcal{T}$ if and only if either in_{B_1} or in_{B_2} occur in $\mathcal{T}$, *(iv)* it is not the case that in_{B_1} and in_{B_2} both occur in $\mathcal{T}$, *(v)* if there exist $t_i, t_{ij}, t_{oj}, t_o \in \mathbb{N}$ such that $\{(\mathsf{in}_B, t_i), (\mathsf{in}_{B_j}, t_i j), (\mathsf{out}_{B_j}, t_{ij}), (\mathsf{out}_B, t_{o2})\} \subseteq \mathcal{T}$ with $j \in \{1, 2\}$, then we have $t_i = t_{ij} \leq t_{oj} = t_o$.

Intuitively, Definition 2.5 states that a trace complies with a block if it does not go through the block at all, or if the trace enters the the block, then it also exits from the block, and suitably traverses it. Suitably traversing, in turn, depends on the block type. If the block is a task block, then the inner activity has to be executed after the trace enters into the block, and the block is completed as soon as the activity is finished. If the block is a sequence block, then the trace has to go through the inner blocks sequentially, and must comply with both. If the block is a parallel block, then the trace has to go through the inner blocks in whatever sequence, and must comply with both; in addition, the trace must exit from the block at the exact time when the latest of the two inner blocks is completed. Finally,

if the block is a choice block, the execution has to go through exactly one of the inner blocks, avoiding the other one; in addition, the trace has to comply with the inner blocks (this is vacuously true for the block that is not chosen, since such block is not even entered).

Definition 2.6. (Duration compliance)
Let $\mathcal{A}$ be a set of activities, and dur a duration function over $\mathcal{A}$. A complete, fully specified trace $\mathcal{T}$ over $\mathcal{A}$ *complies with* dur if, for each activity $\mathsf{a} \in \mathcal{A}$ with $dur(\mathsf{a}) = \langle m, n \rangle$, whenever there exist $s, e \in \mathbb{N}$ s.t. $(\mathsf{a}, [s, e]) \in \mathcal{T}$, we have $m \leq e - s \leq n$.

Definition 2.7. (Inter-task constraint compliance)
Let $\mathcal{A}$ be a set of activities, and $tcon$ an inter-task constraint function. A complete, fully specified trace $\mathcal{T}$ over $\mathcal{A}$ *complies with* $tcon$ if, for every pair of activities $\mathsf{a}, \mathsf{b} \in \mathcal{A}$ and $s_a, e_a, s_b, e_b \in \mathbb{N}$ such that $\{(\mathsf{a}, [s_a, e_a]), (\mathsf{b}, [s_b, e_b])\} \in \mathcal{T}$, the following conditions hold:

- if $tcon(\mathsf{a}, \mathfrak{s}, \mathsf{b}, \mathfrak{s}) = \langle m_1, n_1 \rangle$, then $m_1 \leq s_b - s_a \leq n_1$;
- if $tcon(\mathsf{a}, \mathfrak{s}, \mathsf{b}, \mathfrak{e}) = \langle m_2, n_2 \rangle$, then $m_2 \leq e_b - s_a \leq n_2$;
- if $tcon(\mathsf{a}, \mathfrak{e}, \mathsf{b}, \mathfrak{s}) = \langle m_3, n_3 \rangle$, then $m_3 \leq s_b - e_a \leq n_3$;
- if $tcon(\mathsf{a}, \mathfrak{e}, \mathsf{b}, \mathfrak{e}) = \langle m_4, n_4 \rangle$, then $m_4 \leq e_b - e_a \leq n_4$;

Intuitively, Definitions 2.6 and 2.7 impose that the trace respects the modeled temporal constraints, respectively ensuring that the duration of each activity execution agrees with the specified duration, and that whenever the trace contains two activities that are subject to an inter-task constraint, the time distance between their execution must agree with the specified constraint.

We put together the three definitions above so as to obtain the following general notion of strong compliance. Notice that, in addition to block compliance, we also require that the trace actually enters into the top block of the SPOT of interest.

Definition 2.8. (Strong compliance)
Let $\mathcal{S} = \langle \mathcal{A}, obs, P, dur, tcon \rangle$ be a SPOT. A total, completely specified trace $\mathcal{T}$ over $\mathcal{A}$ is *strongly compliant* with $\mathcal{S}$ if: *(i)* in_P occurs in $\mathcal{T}$, *(ii)* $\mathcal{T}$ is compliant with P, dur and $tcon$.

This notion of compliance is used to characterise the degree to which a given trace conforms/is aligned to the model. In fact, it reflects the usual notion of compliance for business processes, where compliance is typically defined under the assumption that the trace represents a complete end-to-end execution that can be fully replayed on the process model. This is why we call it "strong" compliance. An example of strong compliant trace is the one in (1). Replacing $(\mathsf{A2}, [10, 15])$ with $(\mathsf{A2}, [50, 60])$ in (1) makes it become not compliant. In fact, going from the start of A1 to the end of A2 would require 58 time units thus violating the inter-task constraint between these two activities.

2.3. The Additional Reasoning Services

Starting from the notion of (strong) compliance, we define the remaining reasoning tasks of Figure 1 (right hand side), covering the classical notion of consistency, as well as different notions of compliance that take into account the sources of incompleteness that may be present in the trace.

Model Consistency checks if a SPOT enables acceptable executions from start to end. This case bears no (that is, fully incomplete) information on the execution traces, and reasons on the model alone to determine whether it has acceptable executions.

Definition 2.9. (Consistency)
A SPOT is *consistent* if there exists at least one fully specified trace that strongly complies with it.

Figure 1 shows a consistent model. Modifying the inter-task constraint between the start of A1 to the end of A2 replacing ≤ 30 with ≤ 5 makes the model inconsistent. In fact executing A1 and A2 in sequence requires at least 10 units of time.

Conditional compliance handles the case where the trace under analaysis is indeed partial and/or partially specified. This source of incompleteness in a trace hinders the possibility of replaying it on the process model. However, strong compliance might be regained by assuming that the trace included additional information on the missing or partially specified events.

Definition 2.10. (Conditional compliance)
A trace $\mathcal{T}$ over $\mathcal{A}$ is *conditionally compliant* with a SPOT $\mathcal{S} = \langle \mathcal{A}, obs, P, dur, tcon \rangle$ if $\mathcal{T}$ is not strongly compliant with $\mathcal{S}$, but there exists a total, fully specified trace $\widehat{\mathcal{T}}$ that *(i)* is strongly compliant with $\mathcal{S}$, and *(ii)* satisfies the following conditions:

- for every event $(\mathsf{a}, [s, e]) \in \mathcal{T}$, there is a corresponding event $(\mathsf{a}', [s', e']) \in \widehat{\mathcal{T}}$, where $\mathsf{a}' = \mathsf{a}$ if $\mathsf{a} \in \mathcal{A}$, $s' = s$ if $s \in \mathbb{N}$, and $e' = e$ if $e \in \mathbb{N}$;[4]
- if $\widehat{\mathcal{T}}$ contains additional events, those refer only to unobservable or partially-observable activities from $\mathcal{A}$.

The first condition of Definition 2.10 captures the case of (partially specified) events present in the original trace under study, whereas the second condition deals with additional events, which may only refer to activities that are not always observable.

A sample partial trace over the FF model is:

$$\{(\mathsf{A1}, [2, 7]), (\mathsf{A2}, \_), (\mathsf{A4}, [50, \_]), (\mathsf{A13}, [300, 317]), (\mathsf{A14}, [320, 330])\} \tag{2}$$

Notice that "_" may refer to a missing event name $(\_, [1, 3])$, missing event time $(\mathsf{A2}, \_)$ or missing event time detail $(\mathsf{A4}, [50, \_])$. It is easy to see that (2) is compliant with FF, *if*

$$\begin{gathered} \mathsf{A2} \text{ was executed in an interval } [T2_s, T2_e] \text{ s.t. } 7 < T2_s; \\ T2_e < 50; 5 \leq T2_e - T2_s \leq 10 \text{ and } T2_e - 2 \leq 30 \end{gathered} \tag{3}$$

$$\begin{gathered} \text{an execution of } \mathsf{A3} \text{ was performed in an interval } [T3_s, T3_e] \\ \text{s.t. } T2_e < T3_s; T3_e < 50; \text{ and } 30 \leq T3_e - T3_s \leq 40 \end{gathered} \tag{4}$$

$$\begin{gathered} \mathsf{A4} \text{ was executed with end time } T4_e \text{ s.t. } T4_e < 300 \\ \text{and } 150 \leq T4_e - 50 \leq 200 \end{gathered} \tag{5}$$

[4]Recall that a and/or s and/or e may be filled with "_".

Summing up, a trace $\widehat{\mathcal{T}}$ as defined in Def. 2.10, that would make the trace $\mathcal{T}$ in 2 conditionally complaint would be:

$$\{(\mathsf{A1},[2,7]),(\mathsf{A2},[10,17]),(\mathsf{A3},[18,48]),(\mathsf{A4},[50,220]),(\mathsf{A13},[300,317]),(\mathsf{A14},[320,330])\} \quad (6)$$

Note that the set of assumptions needed to reconstruct full conformance is not necessarily unique. This because alternative strongly compliant real process executions might have led to the recorded partial trace. On the other hand, there are situations in which it is impossible to recover compliance formulating additional assumptions. In this case, the partial trace is considered *non-compliant*. E.g.,

$$\{(\mathsf{A1},[T1_s,T1_e]),(\mathsf{A3},\_)(\mathsf{A9},[\_,T9_e])\} \quad (7)$$

does not comply with FF since A3 and A9 belong to mutually exclusive branches in the model.

While compliance refers to terminated traces, **runtime monitoring** aims at dealing with ongoing executions in order to detect early violations of compliance / ensure the existence of a positive outcome. Here the incompleteness concerns future steps of an ongoing process execution as in the following trace whose last activity executed is A9:

$$\{(\mathsf{A1},[2,7]),(\mathsf{A2},[10,15]),(\mathsf{A5},[16,46]),(\mathsf{A6},[50,60]),(\mathsf{A8},[200,210]),(\mathsf{A9},[350,380])\} \quad (8)$$

This trace already violates the inter-task constraint between A8 and A11 (even if A11 has not been executed yet). In fact 170 units of time have already passed between the start of T8 and the end of T9 while an inter-task constraint requires that A11 starts at most 140 units of time after the start of A8. In the case of an evolving trace, conditional compliance can be directly defined by extending Def. 2.10 so that also events referring to observable activities in $\widehat{\mathcal{T}}$ can be added providing that these are hypothesized to occur at times that go beyond the current time (i.e., no observable event can be hypothesized in the past).

Similarly to runtime monitoring, **prediction/recommendation** deals with ongoing executions with the aim of providing a completion that satisfies certain conditions. For example, consider a process execution composed of the first 5 events of (8) (that is, surgery has been scheduled). One could ask a recommender system to provide the sequence of actions that minimize the time needed to discharge the patient and terminate the process, obtaining the following answer:

$$\begin{aligned}&\{(\mathsf{A1},[2,7]),(\mathsf{A2},[10,15]),(\mathsf{A5},[16,46]),(\mathsf{A6},[50,60]),\\&(\mathsf{A8},[200,210]),(\mathsf{A9},[211,241]),(\mathsf{A10},[211,291]),(\mathsf{A11},[311,411]),\\&(\mathsf{A12},[412,612]),(\mathsf{A13},[613,628]),(\mathsf{A14},[629,634])\}\end{aligned} \quad (9)$$

We close with an interesting observation on the normal form of SPOTs. An arbitrary SPOT $\mathcal{P}$ can be straightforwardly converted in normal form, by introducing dedicated special activities capturing entrance/exit into/from its blocks, and then recursively "normalizing" its top block by replacing each block with its normalized version. The following property shows that normalization preserves, to a certain extent, compliance. Notice that for a normal SPOT it only make sense to talk about conditional compliance, since it contains unobservable activities marking the boundaries of its process blocks.

Lemma 2.11. Let $\mathcal{T}$ be a trace, $\mathcal{P}$ be an arbitrary SPOT, and $\mathcal{P}_{norm}$ be the normalized version of $\mathcal{P}$. The following two properties hold:

- if $\mathcal{T}$ is conditionally/strongly compliant with $\mathcal{P}$, then it is conditionally compliant with $\mathcal{P}_{norm}$;
- it $\mathcal{T}$ is conditionally compliant with $\mathcal{P}_{norm}$, then it is either conditionally or strongly compliant with $\mathcal{P}$.

The proof is directly obtained from the construction of $\mathcal{P}_{norm}$, which maintains the structure of $\mathcal{P}$ only altering it by introducing unobservable activities to mark the boundaries of its blocks.

3. Abduction and Incomplete Processes

Abduction is a non-monotonic reasoning process where hypotheses are made to explain observed facts [12]. While deductive reasoning focuses on deciding if a formula ϕ logically follows from a set Γ of logical assertions known to hold, in abductive reasoning it is assumed that ϕ holds (as it corresponds to a set of observed facts) but it cannot be directly inferred by Γ. To make ϕ a consequence of Γ, abduction looks for a further set Δ of hypothesis, taken from a given set of abducible $\mathcal{A}$, which complements Γ in such a way that ϕ can be inferred (in symbols $\Gamma \cup \Delta \models \phi$). The set Δ is called *abductive explanation* (of ϕ). In addition, Δ must usually satisfy a set of (domain-dependent) integrity constraints $\mathcal{IC}$ (in symbols, $\Gamma \cup \Delta \models \mathcal{IC}$). A typical integrity constraint (IC) is a *denial*, which expresses that two explanations are mutually exclusive.

Abduction has been introduced in the framework of Logic Programming in [7]. There, an *Abductive Logic Program (ALP)* is defined as a triple $\langle \Gamma, \mathcal{A}, \mathcal{IC} \rangle$, where: *(i)* Γ is a logic program, *(ii)* $\mathcal{A}$ is a set of abducible predicates, and *(iii)* $\mathcal{IC}$ a set of ICs. Given a goal ϕ, abductive reasoning looks for a set of literals $\Delta \subseteq \mathcal{A}$ such that they entail $\phi \cup \mathcal{IC}$. The integration of constraint solving in abductive logic programming (ACLP) [7, 8] enhances the practical utility of ALP by enriching the representation of the problem domain and by improving the computation of abductive explanations.

In this paper we leverage on ACLP and on the SCIFF abductive logic programming framework [9], an extension of the IFF abductive proof procedure [13]. Beside the general notion of abducible, the SCIFF framework natively supports the key notions of *happened event*, *expectation*, and *compliance* of an observed execution with a set of expectations, which make SCIFF a suitable framework for dealing with event log incompleteness. Let a be an event corresponding to the execution of process activities, and T (possibly with subscripts) its execution time. Abducibles are used in this work to make hypothesis on events that are not recorded in the examined trace. They are denoted using $\mathbf{ABD}(\mathsf{a}, T)$. Happened events are non-abducible, and account for events that have been logged in the trace. They are denoted using $\mathbf{H}(\mathsf{a}, T)$. Expectations $\mathbf{E}(\mathsf{a}, T)$, instead, model events that should occur (and therefore should be present in a trace). Compliance is described in Section 3.2.

ICs in SCIFF are used to relate happened events / abduced predicates with expectations / predicates to be abduced. Specifically, an IC is a rule of the form $body \rightarrow head$, where $body$ contains a conjunction of happened events, general abducibles, and defined predicates, while $head$ contains a disjunction of conjunctions of expectations, general abducibles, and defined predicates.

Example 3.1. The fact that whenever an order is paid, then a receipt has to be emitted within 24 time units at the latest, can be encoded in SCIFF as the following IC:

$$\mathbf{H}(\mathsf{pay\text{-}order}, T_p) \rightarrow \mathbf{E}(\mathsf{emit\text{-}receipt}, T_e) \wedge T_e > T_p \wedge T_e < T_p + 24$$

3.1. Encoding SPOTs in SCIFF

We show how SPOTs and their traces can be encoded in SCIFF. We start by considering traces. Since SCIFF natively provides the notion of happened event, it also comes with a notion of trace.

Definition 3.2. (SCIFF trace)
A (SCIFF) *trace* $\mathcal{T}$ is a set of terms of type $\mathbf{H}(e, T_i)$, where e is a term describing the happened event, and $T_i \in \mathbb{N}$ is the time instant at which the event occurred.

Thanks to the fact that a SPOT trace $\mathcal{T}$ (in the sense of Definition 2.4) does not contain repeated activities, it can be directly encoded into a corresponding SCIFF trace $\overline{\mathcal{T}}$ (in the sense of Definition 3.2) by representing each entry $(\mathsf{a}, [t_s, t_e]) \in \mathcal{T}$ using two distinct happened events, one for the start, and one for the completion of a. Specifically:

- For each entry $(\mathsf{a}, [t_s, t_e]) \in \mathcal{T}$, we have that $\{\mathbf{H}(\mathsf{a\text{-}start}, t_s), \mathbf{H}(\mathsf{a\text{-}end}, t_e)\} \subseteq \overline{\mathcal{T}}$, where a-start and a-end are used to explicitly indicate the start and end events of a generic activity a;
- nothing else is in $\overline{\mathcal{T}}$.

Since this transformation is straightforward, we will interchangeably use the same symbol and the same term "trace" to refer to a SPOT trace or its corresponding SCIFF representation.

Example 3.3. Trace (1) is represented in SCIFF as:

$$\{\mathbf{H}(\mathsf{a1\text{-}start}, 2), \mathbf{H}(\mathsf{a1\text{-}end}, 7), \mathbf{H}(\mathsf{a2\text{-}start}, 10), \mathbf{H}(\mathsf{a2\text{-}end}, 15), \ldots\}$$

The encoding of a SPOT model $\mathcal{P} = \langle \mathcal{A}, obs, P, dur, tcon \rangle$ in SCIFF consists of a set $\mathcal{IC}_{\mathcal{P}}$ of integrity constraints that account for: *(i)* the semantics of activity execution, considering every activity in $\mathcal{A}$ together with its observability (as defined by obs) and duration (as defined by dur); *(ii)* the semantics of the process over $\mathcal{A}$, as defined by P and its inner blocks; *(iii)* the temporal constraints introduced by $tcon$. We review each such contribution next.

Semantics of activity execution. The semantics of activity execution relates the start event of each activity a in $\mathcal{A}$ with the corresponding end event. The nature of such relationship depends on the observability of a, as well as its duration. On the one hand, the observability determines whether the start/end events for a have to be found explicitly in the trace, or are instead hypothesized. In the first case, the abstractions of happened/expected event provided by SCIFF are employed; in the latter case, instead, a generic abducible **ABD** is used. On the other hand, the duration induces temporal constraints binding the timestamps of the start and end event.

Specifically, let $dur(\mathsf{a}) = \langle m, n \rangle$. If a is observable, that is, $obs(\mathsf{a}) = \mathtt{o}$, the set $\mathcal{IC}_\mathcal{P}$ contains the following integrity constraints:

$$\mathbf{H}(\text{a-start}, T) \wedge \mathbf{H}(\text{a-start}, T_2) \wedge T_2 \neq T \rightarrow \bot \quad (10)$$

$$\mathbf{H}(\text{a-end}, T) \wedge \mathbf{H}(\text{a-end}, T_2) \wedge T_2 \neq T \rightarrow \bot \quad (11)$$

$$\mathbf{H}(\text{a-start}, T_s) \rightarrow \mathbf{E}(\text{a-end}, T_e) \wedge T_e - T_s \geq m \wedge T_e - T_s \leq n \quad (12)$$

$$\mathbf{H}(\text{a-end}, T_e) \rightarrow \mathbf{E}(\text{a-start}, T_e) \wedge T_e - T_s \geq m \wedge T_e - T_s \leq n \quad (13)$$

where, in the case where $n = \infty$, constraint $T_e \leq T_s + n$ simply reduces to $true$.

Integrity constraints (10) and (11) express that every activity can be repeated at most once, in agreement with the notion of SPOT trace. Constraints (12) and (13), instead, relate the start and completion of each activity, stating that whenever the activity is started, it is expected to be completed at a time that is compatible with the duration of the activity, and vice-versa.

If a is unobservable, that is, $obs(\mathsf{a}) = \mathtt{n}$, the set $\mathcal{IC}_\mathcal{P}$ contains a variant of the three integrity constraints above, where the generic abducible **ABD** substitutes **H** and **E**:

$$\mathbf{ABD}(\text{a-start}, T) \wedge \mathbf{ABD}(\text{a-start}, T_2) \wedge T_2 \neq T \rightarrow \bot \quad (14)$$

$$\mathbf{ABD}(\text{a-end}, T) \wedge \mathbf{ABD}(\text{a-end}, T_2) \wedge T_2 \neq T \rightarrow \bot \quad (15)$$

$$\mathbf{ABD}(\text{a-start}, T_s) \rightarrow \mathbf{ABD}(\text{a-end}, T_e) \wedge T_e - T_s \geq m \wedge T_e - T_s \leq n \quad (16)$$

$$\mathbf{ABD}(\text{a-end}, T_e) \rightarrow \mathbf{ABD}(\text{a-start}, T_s) \wedge T_e - T_s \geq m \wedge T_e - T_s \leq n \quad (17)$$

where, in the case where $n = \infty$, constraint $T_e \leq T_s + n$ simply reduces to $true$.

Finally, if a is partially observable, that is, $obs(\mathsf{a}) = \mathtt{p}$, the formalization of the execution of a has to simultaneously account for the case where the execution of a is observed, as well that where the execution of a is not observed, and may consequently be hypothesized. This is done by imposing all integrity constraints (10)-(16), together with the following ones, expressing that the execution of a may be either observed or hypothesized, but not both:

$$\mathbf{H}(\text{a-start}, T_s) \wedge \mathbf{ABD}(\text{a-start}, T_s) \rightarrow \bot \quad (18)$$

$$\mathbf{H}(\text{a-end}, T_s) \wedge \mathbf{ABD}(\text{a-end}, T_s) \rightarrow \bot \quad (19)$$

Encoding of the process. The process is encoded in SCIFF starting from the top block P, formalizing its execution semantics according to the definition of compliance (Definition 2.5), then proceeding recursively over its inner blocks. As for the encoding, we assume that P and all its inner blocks are in normal form, and given a block B, we denote by in-B and out-B the events corresponding to the unobservable atomic tasks marking that the execution is respectively entering into and exiting from B. Technically, the set $\mathcal{IC}_\mathcal{P}$ of integrity constraints contains all constraints produced by the translation function τ applied to P, where τ is defined as follows. Notice that such constraints mirror in SCIFF the different aspects of Definition 2.8. In particular, each constraint is paired with one or (in the case of constraints with disjunctive heads) two corresponding constraints inverting the head and the body, thus realizing the if and only if semantics of Definition 2.5.

(Top block) Assuming $P = \langle \mathtt{seq}, \langle \mathsf{in}_P, P', \mathsf{out}_P \rangle \rangle$, we have that $\tau(P) = \{(20)\} \cup \tau(P')$, where:

$$true \rightarrow \mathbf{ABD}(\text{in-P}, 0) \quad (20)$$

models that the process has to start, mirroring point (i) of Definition 2.8.

(Task block - base case) Consider a block $B = \langle \mathtt{seq}, \langle \mathsf{in}_B, \mathsf{a}, \mathsf{out}_B \rangle \rangle$, where $\mathsf{a} \in \mathcal{A}$. Intuitively, the execution semantics of such a block states that whenever the block is entered, then a is expected to be started in the future, and that as soon as a is completed, the execution exits from the corresponding block. To formalize this intuition, we again have to proceed by cases, depending on the observability of a. If a is observable, $\tau(B)$ consists of:

$$\mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) \rightarrow \mathbf{E}(\mathsf{a\text{-}start}, T_s) \wedge T_s \geq T_{in} \quad (21)$$
$$\mathbf{ABD}(\mathsf{a\text{-}start}, T_s) \rightarrow \mathbf{E}(\mathsf{in\text{-}B}, T_{in}) \wedge T_s \geq T_{in} \quad (22)$$
$$\mathbf{H}(\mathsf{a\text{-}end}, T_e) \rightarrow \mathbf{ABD}(\mathsf{out\text{-}B}, T_e) \quad (23)$$
$$\mathbf{ABD}(\mathsf{out\text{-}B}, T_e) \rightarrow \mathbf{E}(\mathsf{a\text{-}end}, T_e) \quad (24)$$

Together with constraints (10)-(13), constraints (21)-(24) mirror point 1 in Definition 2.5, as well as Definition 2.6. If a is unobservable, then its start/end are hypothesized, hence $\tau(B)$ becomes:

$$\mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) \rightarrow \mathbf{ABD}(\mathsf{a\text{-}start}, T_s) \wedge T_s \geq T_{in} \quad (25)$$
$$\mathbf{ABD}(\mathsf{a\text{-}start}, T_s) \rightarrow \mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) \wedge T_s \geq T_{in} \quad (26)$$
$$\mathbf{ABD}(\mathsf{a\text{-}end}, T_e) \rightarrow \mathbf{ABD}(\mathsf{out\text{-}B}, T_e) \quad (27)$$
$$\mathbf{ABD}(\mathsf{out\text{-}B}, T_e) \rightarrow \mathbf{ABD}(\mathsf{a\text{-}end}, T_e) \quad (28)$$

Finally, if a is partially observable, the effect of entering into the block may be either that of expecting the start of a to occur in the trace, or to hypothesize it (cf. constraint (29) and its two "only if" constraints (30) and (31)). Specularly, the block is exited if and only if the completion of a is either observed or hypothesized, at the same time (cf. constraints (32)-(34)). Mutual exclusion between the case where the start (respectively, completion) of a is observed, and that where it is hypothesized, is guaranteed by the integrity constraint (18) (respectively, (19)). In formulae, $\tau(B)$ consists of:

$$\mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) \rightarrow \mathbf{E}(\mathsf{a\text{-}start}, T_s) \wedge T_s \geq T_{in} \vee \mathbf{ABD}(\mathsf{a\text{-}start}, T_s) \wedge T_s \geq T_{in} \quad (29)$$
$$\mathbf{H}(\mathsf{a\text{-}start}, T_s) \rightarrow \mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) \wedge T_s \geq T_{in} \quad (30)$$
$$\mathbf{ABD}(\mathsf{a\text{-}start}, T_s) \rightarrow \mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) \wedge T_s \geq T_{in} \quad (31)$$
$$\mathbf{H}(\mathsf{a\text{-}end}, T_e) \rightarrow \mathbf{ABD}(\mathsf{out\text{-}B}, T_e) \quad (32)$$
$$\mathbf{ABD}(\mathsf{a\text{-}end}, T_e) \rightarrow \mathbf{ABD}(\mathsf{out\text{-}B}, T_e) \quad (33)$$
$$\mathbf{ABD}(\mathsf{out\text{-}B}, T_e) \rightarrow \mathbf{E}(\mathsf{a\text{-}end}, T_e) \vee \mathbf{ABD}(\mathsf{a\text{-}end}, T_e) \quad (34)$$

(Inductive case - Sequence) Consider block B of the form $\langle \mathtt{seq}, \langle \mathsf{in}_B, \langle \mathtt{seq}, B_1, B_2 \rangle, \mathsf{out}_B \rangle \rangle$. The encoding $\tau(B)$ of B consists of a series of integrity constraints that chain the start/completion of the inner blocks. Specifically, $\tau(B) = \{(35) - (40)\} \cup \tau(B_1) \cup \tau(B_2)$, where:

$$\mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) \rightarrow \mathbf{ABD}(\mathsf{in\text{-}B_1}, T_{in}) \quad (35)$$
$$\mathbf{ABD}(\mathsf{in\text{-}B_1}, T_{in}) \rightarrow \mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) \quad (36)$$
$$\mathbf{ABD}(\mathsf{out\text{-}B_1}, T_1) \rightarrow \mathbf{ABD}(\mathsf{in\text{-}B_2}, T_1) \quad (37)$$
$$\mathbf{ABD}(\mathsf{out\text{-}B_2}, T_1) \rightarrow \mathbf{ABD}(\mathsf{in\text{-}B_1}, T_1) \quad (38)$$
$$\mathbf{ABD}(\mathsf{out\text{-}B_2}, T_{out}) \rightarrow \mathbf{ABD}(\mathsf{out\text{-}B}, T_{out}) \quad (39)$$
$$\mathbf{ABD}(\mathsf{out\text{-}B}, T_{out}) \rightarrow \mathbf{ABD}(\mathsf{out\text{-}B_2}, T_{out}) \quad (40)$$

Constraints (35) and (36) model that as soon as block B is entered, its first inner block in the sequence is entered, and vice-versa. Constraints (37) and (38) model that the execution exits from the first block of the sequence if and only if it enters into the consequent one. Finally, constraints (39) and (40) model that as soon as the execution exits from the second block of the sequence, it exits from the overall sequence block B, and vice-versa.

(Inductive case - Parallel split) Consider block B of the form $\langle \texttt{and}, \langle \mathsf{in}_B, \langle \texttt{seq}, B_1, B_2 \rangle, \mathsf{out}_B \rangle \rangle$. The encoding $\tau(B)$ of B captures the parallel execution of the two inner blocks, synchronizing upon their completion. Specifically, $\tau(B) = \{(41) - (46)\} \cup \tau(B_1) \cup \tau(B_2)$, where:

$$\begin{aligned}
\mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) &\rightarrow \mathbf{ABD}(\mathsf{in\text{-}B_1}, T_{in}) \wedge \mathbf{ABD}(\mathsf{in\text{-}B_2}, T_{in}) && (41)\\
\mathbf{ABD}(\mathsf{in\text{-}B_1}, T_{in}) &\rightarrow \mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) && (42)\\
\mathbf{ABD}(\mathsf{in\text{-}B_2}, T_{in}) &\rightarrow \mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) && (43)\\
\mathbf{ABD}(\mathsf{out\text{-}B_1}, T_{o1}) \wedge \mathbf{ABD}(\mathsf{out\text{-}B_2}, T_{o2}) \wedge T_{o1} \geq T_{o2} &\rightarrow \mathbf{ABD}(\mathsf{out\text{-}B}, T_{o1}) && (44)\\
\mathbf{ABD}(\mathsf{out\text{-}B_1}, T_{o1}) \wedge \mathbf{ABD}(\mathsf{out\text{-}B_2}, T_{o2}) \wedge T_{o1} < T_{o2} &\rightarrow \mathbf{ABD}(\mathsf{out\text{-}B}, T_{o2}) && (45)\\
\mathbf{ABD}(\mathsf{out\text{-}B}, T_{o}) &\rightarrow \mathbf{ABD}(\mathsf{out\text{-}B_1}, T_{o}) \wedge \mathbf{ABD}(\mathsf{out\text{-}B_2}, T_{o2}) \wedge T_{o} \geq T_{o2} && (46)\\
&\quad \vee \mathbf{ABD}(\mathsf{out\text{-}B_1}, T_{o1}) \wedge \mathbf{ABD}(\mathsf{out\text{-}B_2}, T_{o}) \wedge T_{o} \geq T_{o1}
\end{aligned}$$

Constraints (41)-(43) model that the execution enters B if and only if it enters B_1 if and only if it enters B_2. Constraints (44) and (45) model synchronization upon the completion of such sub-blocks, dictating that the execution exits from B as soon as both B_1 and B_2 are completed, respectively handling the case where B_1 completes simultaneously/after or before B_2. Their converse constraint is captured by (46), stating that whenever B completes, then its latest sub-block must complete at the same time.

(Inductive case - Exclusive choice) Consider block B of the form $\langle \texttt{xor}, \langle \mathsf{in}_B, \langle \texttt{seq}, B_1, B_2 \rangle, \mathsf{out}_B \rangle \rangle$. The encoding $\tau(B)$ of B captures the alternative execution of one of the two inner blocks. Specifically, $\tau(B) = \{(47) - (53)\} \cup \tau(B_1) \cup \tau(B_2)$, where:

$$\begin{aligned}
\mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) &\rightarrow \mathbf{ABD}(\mathsf{in\text{-}B_1}, T_{in}) \vee \mathbf{ABD}(\mathsf{in\text{-}B_2}, T_{in}) && (47)\\
\mathbf{ABD}(\mathsf{in\text{-}B_1}, T_{in}) &\rightarrow \mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) && (48)\\
\mathbf{ABD}(\mathsf{in\text{-}B_2}, T_{in}) &\rightarrow \mathbf{ABD}(\mathsf{in\text{-}B}, T_{in}) && (49)\\
\mathbf{ABD}(\mathsf{in\text{-}B_1}, T_{in}) \wedge \mathbf{ABD}(\mathsf{in\text{-}B_2}, T_{in}) &\rightarrow \bot && (50)\\
\mathbf{ABD}(\mathsf{out\text{-}B_1}, T_{out}) &\rightarrow \mathbf{ABD}(\mathsf{out\text{-}B}, T_{out}) && (51)\\
\mathbf{ABD}(\mathsf{out\text{-}B_2}, T_{out}) &\rightarrow \mathbf{ABD}(\mathsf{out\text{-}B}, T_{out}) && (52)\\
\mathbf{ABD}(\mathsf{out\text{-}B}, T_{out}) &\rightarrow \mathbf{ABD}(\mathsf{out\text{-}B_1}, T_{out}) \vee \mathbf{ABD}(\mathsf{out\text{-}B_2}, T_{out}) && (53)
\end{aligned}$$

Constraints (47)-(50) model that as soon as the execution enters into B, it enters into exactly one block between B_1 and B_2, and vice-versa. Specularly, constraints (51) and (52) model that as soon as the execution exits from one of blocks B_1 or B_2, it simultaneously exists from the overall block B as well. Conversely, constraint (53) captures if block B completes, then one of its inner sub-blocks have to complete at the same time.

Encoding of inter-task constraints. Each inter-task constraint is directly encoded in SCIFF using a dedicated integrity constraints imposing that whenever the two events mentioned in the inter-task

constraint occur, their corresponding execution time shall respect the imposed duration interval. Technically, let $ev : \mathcal{A} \times \{\mathfrak{s}, \mathfrak{e}\} \to \mathbb{S}$ be a total function that, given an activity and a start/end inscription, produces a string representing the corresponding event: for every $\mathsf{a} \in \mathcal{A}$, we have $ev(\mathsf{a}, \mathfrak{s}) = \mathsf{a}\text{-start}$, and $ev(\mathsf{a}, \mathfrak{e}) = \mathsf{a}\text{-end}$.

For every pair of activities $\mathsf{a}_1, \mathsf{a}_2 \in \mathcal{A}$, and every pair of inscriptions $i_1, i_2 \in \{\mathfrak{s}, \mathfrak{e}\}$ such that $tcon(\mathsf{a}_1, i_1, \mathsf{a}_2, i_2)$ is defined and gives $\langle m, n \rangle$, set $\mathcal{IC}_\mathcal{P}$ contains a dedicated integrity constraint, whose exact shape depends on the observability of a_1 and a_2. If a_1 and a_2 are both observable, we get:

$$\mathbf{H}(ev(\mathsf{a}_1, i_1), T_1) \wedge \mathbf{H}(ev(\mathsf{a}_2, i_2), T_2) \to T_2 - T_1 \geq m \wedge T_2 - T_1 \leq n \tag{54}$$

If a_1/a_2 is unobservable, the corresponding happened event is replaced by the generic abducible $\mathbf{ABD}(ev(\mathsf{a}_1, i_1), T_1)$/$\mathbf{H}(ev(\mathsf{a}_2, i_2), T_2)$. Finally, in the case of partial observability, both possibilities have to be considered, thus obtaining 2 to 4 distinct integrity constraints.

Example 3.4. The inter-task constraint between A8 and A11 in Figure 1 indicares that the elapsed time between the start of the scheduling of a surgery (A8), and the end of the surgery itself (A11), should be between 100 and 140 time units. Assuming that A8 is observable while A11 is partially observable, such inter-task constraint would be encoded in SCIFF using the following two integrity constraints:

$$\begin{aligned} \mathbf{H}(\mathsf{a8\text{-}start}, T_8) \wedge \mathbf{H}(\mathsf{a11\text{-}end}, T_{11}) &\to T_{11} - T_8 \geq 100 \wedge T_{11} - T_8 \leq 140. \\ \mathbf{H}(\mathsf{a8\text{-}start}, T_8) \wedge \mathbf{ABD}(\mathsf{a11\text{-}end}, T_{11}) &\to T_{11} - T_8 \geq 100 \wedge T_{11} - T_8 \leq 140. \end{aligned}$$

3.2. Compliance in SCIFF: Declarative Semantics

Besides the ability to capture different workflow constructs in a modular manner, the second important characteristic of our framework concerns the ability to represent the different forms of reasoning introduced in Section 2 in a uniform manner in terms of (strong or conditional) compliance.

Let t_c be the current time of an ongoing execution trace $\mathcal{T}$. Depending on the choice of t_c, compliance of the trace $\mathcal{T}$ w.r.t, a process model M can be used to simulate different types of reasoning as follows:

(1) if $t_c = 0$, i.e. the observed trace is empty, then conditional compliance is used to hypothesise at least one possible execution. Thus, we obtain *model consistency*;
(2) if $t_c = t_{final}$, i.e., the execution has reached a final state, then we are in the case of a-posteriori compliance checking (declined in *strong* and *conditional* depending on whether the trace is complete or not);
(3) if $0 < t_c < t_{final}$, we obtain *run-time monitoring*. In fact SCIFF can be used to check the (conditional) compliance of the executed part of the trace and make hypothesis on its future evolution. Adding further constraints (e.g., the minimization of overall execution time) enables SCIFF to provide also *prediction/recommendation*.

Hence, by providing a formal notion of strong and conditional compliance, we accomodate the reasoning tasks of Section 2 in the SCIFF setting. To make this approach operational for SPOTs, we simply take a normal SPOT $\mathcal{P}$ and translate it into the corresponding $\mathcal{S}_\mathcal{P} = \langle \emptyset, \mathcal{A}, \mathcal{IC}_\mathcal{P} \rangle$, where: *(i)* the logic program of $\mathcal{S}_\mathcal{P}$ is empty, *(ii)* the set of abducible predicates is defined as $\mathcal{A} = \{\mathbf{ABD}/2, \mathbf{E}/2\}$, where $\mathbf{E}/2$ predicates represent expectations, and $\mathbf{ABD}/2$ are predicates that can be abduced/hypothesized; *(iii)* $\mathcal{IC}_\mathcal{P}$ is the set of integrity constraints obtained by the translation function defined in Section 3.1.

To recall the formal notion of compliance in SCIFF, we first need to introduce what an abductive explanation is[5]:

Definition 3.5. (Abductive explanation Δ)
Given a SCIFF specification $\mathcal{S}$ and a trace $\mathcal{T}$, a set $\Delta \subseteq \mathcal{A}$ is an *abductive explanation* for $\langle \mathcal{S}, \mathcal{T} \rangle$ if and only if

$$Comp\left(\mathcal{KB} \cup \mathcal{T} \cup \Delta\right) \cup \mathrm{CET} \cup T_{\mathbb{N}} \models \mathcal{IC}$$

where *Comp* is the (two-valued) completion of a theory [14], *CET* stands for Clark Equational Theory [15] and $T_{\mathbb{N}}$ is the CLP constraint theory [16] for integers.

The following definition fixes the semantics for observable events, and provides the basis for understanding the alignment of a trace with a process model.

Definition 3.6. ($\mathcal{T}$-Fulfilment w.r.t. t_c)
Given a trace $\mathcal{T}$, an abducible set Δ is *trace-fulfilled w.r.t.* t_c if for every event $\mathtt{e} \in \{\Delta \cup \mathcal{T}\}$ and for each time $\mathtt{t} \leq t_c$, $\mathbf{E}(\mathtt{e}, \mathtt{t}) \in \Delta$ if and only if $\mathbf{H}(\mathtt{e}, \mathtt{t}) \in \mathcal{T}$.

The "only if" direction defines the semantics of expectation, indicating that an expectation is fulfilled when it finds the corresponding happening event in the trace. The "if" direction captures the prescriptive nature of process models, whose *closed* nature require that only expected events may happen. Notice that fulfilment is restricted to time instants prior to t_c, thus capturing the fact that the past (events before t_c) is somehow "closed", while the future is intrinsically open.

Given an abductive explanation Δ, fulfilment acts as a *compliance classifier*, which separates the legal/correct execution traces with respect to Δ from the wrong ones.

Definition 3.7. (Compliance)
A trace $\mathcal{T}$ is *compliant* with a SCIFF specification $\mathcal{S}$ *w.r.t. a time instant* t_c if there exists an abducible set Δ such that: *(i)* Δ is an abductive explanation for $\langle \mathcal{S}, \mathcal{T} \rangle$, and *(ii)* Δ is $\mathcal{T}$-fulfilled w.r.t. t_c. If Δ only contains **E** abducibles, then we say that it is *strongly-compliant*, otherwise it is *conditionally-compliant*.

We also say that $\mathcal{T}$ is *compliant* with $\mathcal{S}$ if $\mathcal{T}$ is compliant with $\mathcal{S}$ w.r.t. the maximum time instant mentioned in $\mathcal{T}$.

If no abductive explanation that is also $\mathcal{T}$-fulfilled can be found, then $\mathcal{T}$ is not compliant with the specification of interest. Contrariwise, the abductive explanation witnesses compliance. However, it may contain **ABD** predicates, abduced due to the incompleteness of $\mathcal{T}$, or additional constraints, hypothesized due to missing information units in the trace. In fact, the presence or absence of such abducibles determines whether $\mathcal{T}$ is conditionally or strongly compliant.

We close this section by commenting on the connection between the notion of compliance for SPOTs, and the corresponding notion of compliance in SCIFF, consequently establishing that the encoding of SPOTs in SCIFF is correct.

Theorem 3.8. A trace $\mathcal{T}$ is conditionally compliant with a normal SPOT $\mathcal{P}$ (in the sense of Definition 2.10) if and only if it is conditionally compliant with $\mathcal{S}_{\mathcal{P}}$ (in the sense of Definition 3.7).

[5]We do not consider the abductive goal, as it is not needed for our treatment.

Theorem 3.9. Let $\mathcal{P}$ be an arbitrary SPOT, and $\mathcal{P}_{norm}$ its normalized version. A trace $\mathcal{T}$ is strongly compliant with $\mathcal{P}$ (in the sense of Definition 2.8) if and only if it is conditionally compliant with $\mathcal{S}_{\mathcal{P}_{norm}}$ (in the sense of Definition 3.7), and every abductive explanation witnessing such conditional compliant is such that abduced **ABD** predicates only refer to the special activities introduced in $\mathcal{P}_{norm}$ to mark the boundaries of process blocks in $\mathcal{P}$.

Intuitively, Theorem 3.9 indicates that for an arbitrary SPOT, the SCIFF encoding of its normalized version does not only properly reconstruct conditional compliance, but also strong compliance, witnessed by the fact that the only hypothesized activities are those artificially introduced in the normalization.

The proof of Theorem 3.8 is a direct consequence of the encoding, which modularly preserves block compliance, duration compliance, and inter-task compliance. In particular, correctness can be shown by induction over the structure of process blocks, reasoning case by case and noticing that the encoding proposed in Section 3.1 closely reconstructs Definitions 2.5, 2.6, and 2.7. Theorem 3.9 follows from the fact that the activities artificially introduced in the normalization of SPOTs are unobservable, and from the fact that observable activities are encoded in SCIFF using actual happened events and expectations, which have to be found explicitly in the trace under study, whereas unobservable/partially observable activities are exclusively/also associated to the possibility of hypothesizing their execution.

Finally, recall that the SCIFF proof procedure, which can be actually employed as a reasoning machinery on top of SCIFF specifications, has been proven sound and complete w.r.t. the SCIFF declarative semantics [9] Our declarative semantics restricts the notions of fulfilment and compliance to a specific current time t_c, i.e., to open traces: hence soundness and completeness still hold [17], and can be directly applied to the case of monitoring as well. Furthermore, when reasoning over SCIFF specifications encoding SPOTs, it is guaranteed that the size of actual/hypothesized traces is bounded by the number of activities present in the SPOT under study. This, in turn, implies termination. In summary, we obtain that the SCIFF proof procedure can be effectively employed to reason on conditional/strong compliance of SPOTs either at run-time or a posteriori, obtaining correct answers in a finite amount of time. This is subject of the next section.

3.3. SCIFF at work

The SCIFF proof procedure can be queried by specifying a goal. While the query is goal driven and supports a backward reasoning style, the use of integrity constraints allows to support a forward reasoning style at the same time. In the following, we use $\mathsf{in\text{-}B_0}$ and $\mathsf{out\text{-}B_0}$ to refer to the start and completion of the top block of the (normal) SPOT of interest.

In our approach the goal is always to prove $\mathbf{ABD}(\mathsf{in\text{-}B_0}, 0)$. The goal itself can be satisfied immediately, since start can be abduced (and added to the answer Δ). However, abducing start will trigger an IC. For example, consider the normalized version of the FF process fragment in Example 2.2. In this case, $\mathsf{in\text{-}B_0}$ is related to the sequence between the two task blocks for A1 and A2, encoded through the instantiation of integrity constraints (35)-(40) to the actual block names of the sequence. Since $\mathbf{ABD}(\mathsf{in\text{-}B_0}, 0)$ appears in the body of (the instantiation of) constraint (35), the constraint triggers, generating the hypothesis that the first, inner task block is entered. This triggers the instantiation of constraint 21, generating an expectation about the future start of A1. From that, SCIFF either matches the expectation with the happening of A1, or hypothesises it (depending on the observability of A1).

In turn, this triggers other integrity constraints.

The proof procedure stops when no more integrity constraints can be triggered, and our encoding ensures that this happens only when the final activity out-B_0 is reached. The outcome is the set Δ of hypotheses under which the goal is true, and all the ICs are satisfied.

Notice that the abductive answer Δ might contain abducibles with variables representing time instants, and this needs to be taken care of. In fact, we exploit ACLP to support temporal constraints among activity executions: a network of CLP constraints is used to restrict variables representing happening times to specific domains. SCIFF authors report that their proof procedure is complete, provided the underlying CLP solver is complete as well. In our case the CLP solver is not complete, unless an explicit assignment of a value to all the variables is requested through a so-called *labelling* (grounding) step. As a consequence we may have an ALP solution Δ, that would be made infeasible by the temporal constraints. To rule out this case, we explicitly require labelling, when needed.

Given a trace, and a workflow encoded as described in Section 3.1, the SCIFF proof procedure can be used to provide the different reasoning services introduced in Section 2. A first general observation is that all reasoning tasks may require labelling of temporal-related variables. *Model consistency* is directly supported by the SCIFF when the execution trace is empty. However, all the activities in the workflow must be partially observable or non-observable, so that the proof procedure can make hypotheses. This proviso also holds for *runtime monitoring*. *Strong compliance* corresponds to the original reasoning task of the SCIFF, and it is still supported in our context. *Conditional compliance* is supported thanks to our proposed mapping, where hypotheses about non-observed events are directly supported by the ICs. Note that, depending on the completeness of the CLP solver used by SCIFF, non compliances due to temporal constraints might be detected as soon as a constraint becomes infeasible, i.e., *before* producing an entire abductive answer. Obtaining *prediction/recommendation* is slightly more complex. Consider, e.g., the request of an abductive answer that minimizes the overall execution time. A naive solution would compute all possible abductive answers, then choosing the one with minimum completion time. A *branch-and-bound* optimisation is obtained by adding the following rules, imposing that any observed/hypothesised event must always happen before the final event:

$$\mathbf{H}(\_, T_1) \wedge \mathbf{H}(\text{out-}\mathsf{B}_0, T_e) \rightarrow T_1 < T_e. \tag{55}$$

$$\mathbf{ABD}(\_, T_1) \wedge \mathbf{H}(\text{out-}\mathsf{B}_0, T_e) \rightarrow T_1 < T_e. \tag{56}$$

Intuitively, as long as no solution is found, out-B_0 is not hypothesised. As soon as a possible answer is found, it is. After that, all hypotheses are bound to happen before the best final time.

A prototype implementation of the framework is currently available for download at `http://ai.unibo.it/AlpBPM`.

4. Evaluation

In this section we investigate the performance of each reasoning task separately by changing the input parameters of interest. The experiments have been carried out on a Windows 7 pc with 8GB

Model	Size	Paral. %	Min/Max	# TCs	Consistency Time(s)	Inconsistency Time(s)
$M1$	20	0.14	6/11	4	0.16	3.71
$M2$	40	0.14	12/22	8	0.52	3.42
$M3$	42	1	12/22	8	2.01	92.17
$M4$	60	0.14	18/33	12	1.6	4.09
$M5$	62	0.14	6/22	12	0.78	9.23

Table 1. Model Consistency.

RAM and a 2.4 GhZ Intel-core i7. The encoding presented in Section 3.1 has been adopted, but few optimizations have been applied to exploit some peculiar features of the SCIFF proof procedure implementation.

We encoded 5 process models ($M1 \ldots M5$) with different characteristics. $M1$ is a realistic model - reported in Figure 1 and taken from [6], while $M2 \ldots M5$ are synthetic models built with the purpose to stress the scalability of the proposed approach. We encoded all tasks as *partially observable*, which is the worst case performance-wise, as we discuss later in the section.

Model consistency. Table 1 reports the characteristics of the models: size (column "Size"), i.e., the number of activities and gateways in BPMN, parallelism degree (column "Paral. %"), i.e., the percentage of activities that can be executed in parallel, min/max path length in the model (column "Min/Max") and number of temporal constraints (column "# TCs"). For each of these (consistent) models, an inconsistent variant is built by violating one of the constraints. Table 1 reports the time required by the abductive framework for assessing the *consistency* of $M1 \ldots M5$ (column "Consistency Time") as well as the inconsistency of their variants (column "Inconsistency Time").

As expected, the results reported in the table show that the size and the structure of the model influence the time required for checking the (in)consistency. For instance, detecting the consistency of $M3$, where a higher number of parallel gateways occurs, is more expensive than computing the consistency for models of larger size. Also, inconsistency detection requires overall more time than the consistency detection, as when no consistent path is found, all the paths of the model have to be explored.

Strong compliance. For each model in $M1 \ldots M5$ we consider 2 compliant and 2 non-compliant traces of different length. Table 2 reports, for each model (column "Model"), the 4 traces (column "Trace"), their length (column "Length"), as well as the outcome of the strong compliance procedure (column "Out."), i.e., c for compliant or nc for non-compliant and the time required by the SCIFF procedure to compute the result (column "Time"). We notice that the time required for the strong compliance ranges between less than one second to less than one minute (about 50 seconds) for short/medium traces (e.g., up to 12 events), and increases up to more than one hour as the trace becomes longer. By looking at the results related to models with similar characteristics and increasing size, such as $M1$, $M2$ and $M4$, we notice an exponential trend in computational time (see traces t1_1, t2_1 and t4_1). This is due to the fact that all tasks in the models are *partially observable*, which requires "reasoning by cases" for each task in the model. Indeed, as explained in Section 3, since each event expectation can be matched with an abducible *or* with an actual observation in the trace, if an event is partially observable, both options have to be considered. This cost can be avoided if information about fully observable/unobservable activity is given.

Model	Trace	Length	Out	Time(s)
$M1$	t1_1	6	c	0.48
	t1_2	6	nc	0.01
	t1_3	11	c	12.49
	t1_4	11	nc	0.01
$M2$	t2_1	12	c	357.08
	t2_2	12	nc	0.02
	t2_3	22	c	$>$ 1h
	t2_4	22	nc	0.032
$M3$	t3_1	12	c	35.64
	t3_2	12	nc	0.05
	t3_3	22	c	$>$ 1h
	t3_4	22	nc	0.09
$M4$	t4_1	18	c	2090.36
	t4_2	18	nc	0.02
	t4_3	33	c	$>$ 1h
	t4_4	33	nc	0.06
$M5$	t5_1	6	c	0.76
	t5_2	6	nc	0.38
	t5_3	11	c	50.8
	t5_4	11	nc	0.06

Table 2. Strong Compliance.

Conditional compliance and runtime monitoring/prediction. We tested out, both conditional

compliance and runtime monitoring with two levels of event incompleteness (50% and 80%) on the 4 traces and the 4 models used for evaluating the strong compliance, for a total of 80 tests. We remark that, while in the conditional compliance the missing events are distributed along the whole execution trace, in the runtime monitoring and prediction case the incompleteness characterizes the last part of the trace only (see right-hand part of Figure 1). Table 3 summarizes the results of the two reasoning services: for each model and for each trace, the percentage of incompleteness of the considered traces is reported (column "Inc. %"). We chose to let the incompleteness degree range between 50% (only half of the events of a complete path from the source to the target is observed in the trace) and 80% (about 20% of the events of a path from the source to the target are observed in the trace). Notice that the case where all the events are observed (incompleteness degree 0%) is covered by the strong compliance case. Column "Out." report the outcome of the service while columns "C. Compl" and "Monit. Time" the computation time for each of the two services.

The computation times required for the two services are quite similar though with some differences for large models. Overall, the framework seems to be more efficient when the source of the incompleteness is located along the trace rather than at the end, in case of large models with a high degree of parallelism (e.g., see traces t3_3a and t3_4a for $M3$). Conversely, with large models characterized by a low parallelism degree, the monitoring service seems to overcome the conditional compliance one (e.g., see traces t4_3a and t4_4b for $M4$). Although computing times are exponential in path length and model size as for the strong compliance, in this case we notice an improvement in performances compared to the strong compliance ones: the less events we observe in the trace the lower the computation time is. This happens because expectations of some events (the ones corresponding to missing events) immediately fail to be matched with happened events (as traces are incomplete). The procedure is hen ce forced to match these expectations with abducibles, ruling out options from reasoning by cases and improving the performances. Note that the time required for the *prediction/recommendation* service is the same of runtime monitoring.

Overall Observations. In this experimentation, we focused on the borderline case, in which no knowledge at all is available about the observability of the events (i.e., all the events are *partially*

Model	Trace	Length	Inc. %	Out.	C. Compl. Time(s)	Monit. Time(s)
$M1$	t1_1a	6	50	c	0.26	0.23
	t1_1b	6	80	c	0.22	0.1
	t1_2a	6	50	nc	0.01	0.21
	t1_2b	6	80	nc	0.01	0.06
	t1_3a	11	50	c	1.1	0.93
	t1_3b	11	80	c	0.12	0.19
	t1_4a	11	50	nc	0.01	0.61
	t1_4b	11	80	nc	0.07	0.17
$M2$	t2_1a	12	50	c	3.7	5.43
	t2_1b	12	80	c	0.52	5.21
	t2_2a	12	50	nc	0.02	0.01
	t2_2c	12	80	nc	0.51	0.01
	t2_3a	22	50	c	206.7	104.02
	t2_3b	22	80	c	1.12	2.5
	t2_4a	22	50	nc	84.65	63.03
	t2_4b	22	80	nc	1.12	1.87
$M3$	t3_1a	12	50	c	534.2	228.57
	t3_1b	12	80	c	161.63	62.41
	t3_2a	12	50	nc	367.52	0.06
	t3_2b	12	80	nc	160.06	0.08
	t3_3a	22	50	c	1117.79	$>$ 1h
	t3_3b	22	80	c	76.89	30.17
	t3_4a	22	50	nc	722.8	1223.13
	t3_4b	22	20	nc	26.29	28.8
M4	t4_1a	18	50	$c.$	157.45	162.43
	t4_1b	18	80	c	10.75	25.43
	t4_2a	18	50	nc	0.05	0.06
	t4_2b	18	80	nc	14.74	0.08
	t4_3a	33	50	c	$>$ 1h	1722.72
	t4_3b	33	80	c	50.03	168.64
	t4_4a	33	50	nc	0.06	0.06
	t4_4b	33	80	nc	66.49	0.06
$M5$	t5_1a	6	50	c	0.6	1.92
	t5_1b	6	80	c	1.47	1.78
	t4_2a	6	50	nc	0.47	0.11
	t4_2b	6	80	nc	1.47	1.4
	t4_3a	11	50	c	3.96	5.24
	t4_3b	11	80	c	1.84	1.95
	t4_4a	11	50	nc	0.11	4.06
	t4_4b	11	80	nc	1.59	1.25

Table 3. Conditional Compliance, Runtime Monitoring and Prediction.

observable). Purpose of the experimentation was understanding the applicability of the approach in practical scenarios. Overall, in a realistic scenario as the temporal model $M1$ presented in [6], all the reasoning services are carried out in less than 15 seconds. When no information is known about the observability of the events, the abductive framework is particularly performant in situations in which the execution trace is heavily incomplete either along or at the end of the trace. When a large part of the execution trace is unknown (e.g., about 80% of the trace), indeed, even for long paths the performances of the abductive framework are of the order of minutes.

5. Related Work

A consistent part of the BPM literature of the last decades has been devoted to provide reasoning services, such as process model consistency, trace compliance, runtime monitoring as well as scheduling/planning, on top of process models and their executions. *Consistency* of process models is investigated in a number of works, and particularly in the case of workflows enriched with different types of constraints, as for instance business contracts [5] or time [18, 19] constraints. For example, Bettini et al. [18] suggest an approach based on Simple Temporal Networks (STN), where each node represents a time point, edges are temporal relationships and each task is represented by a start and an end node. Planning and critical path methods are exploited instead by Eder and colleagues [19] where the compliance of workflow models is checked in presence of conditional activities.

The *(strong) compliance* of traces towards process models (also known as *conformance checking*) is another key reasoning service, as suggested by the several approaches proposed for dealing with the disalignment between models and traces looking at the control flow only [20, 21] or taking into account also data [22] or time constraints [23]. For example, in [23], compliance to time-constrained workflows is verified with a two-step procedure: the trace is first aligned to the model and then to the time constraint rules. A different perspective on the (strong) compliance is offered in [24], where potential temporal anomalies are detected with respect to a Bayesian model (automatically inferred from a Petri Net) enriched with temporal annotations extracted from historical execution traces.

Conditional compliance of traces towards process models has been tackled in the broader field of conformance checking, although in an indirect manner. Indeed, a number of works focus on the alignment of event logs and procedural/declarative process models: for example, [21, 25] explore the search space of the set of possible moves to find the best ones for aligning the log to the model. Conditional compliance is then viewed as a problem of alignment. Rather, the notion of conditional compliance adopted in this work focuses on prescriptive models and on incomplete logs. Only few approaches in the BPM literature more closely relate to such a notion, for example by leveraging on techniques as Satisfiability Modulo Theory [26], or planning techniques [27].

A number of approaches and tools have been proposed for the *runtime monitoring*, and applied for checking the compliance of running traces versus (enriched) process models, taking into consideration control-flow aspects [4], control-flow and data [28], control-flow and time [11]. For example, Combi and colleagues [11] propose a conceptual model for temporal processes, provide different types of design-time and runtime compliance, and introduce ad-hoc techniques for checking these different

levels of compliance. Concerning *prediction/recommendation* services, some effort has been devoted to recommend how to optimize dimensions such as resources [29] or time [30, 6]. In particular, the work in [6] (from which the example in Figure 1 is taken) presents an approach based on control-flow and temporal constraint satisfaction for the runtime monitoring as well as for the identification of the schedule for a case that better minimizes the time constraint violations.

The SCIFF framework, and in particular its support to abduction, have been exploited in the past to model both procedural [31] and declarative [32] processes, without considering the issue of incompleteness. In [33] instead we focused on the incompleteness issue only, discussing the different types of incomplete data that can be observed in real logs. However, in that work we did not discuss the temporal dimension, that was instead addressed for the first time in a poster presented at ECAI [1]. Hence, [33, 1] can be considered as initial steps towards the SPOT model introduced here. Interestingly, in [34] abduction is exploited to evaluate trace compliance, but checking user permission compliance only. Moreover, [34] deals with incomplete traces only (with complete events), while our solution takes a more sophisticated approach to incompleteness and reasoning services. Note also that the adopted abductive framework, CIFF [35], only supports ground abducibles and denial constraints.

The majority of works on temporal processes assume that the environment is fully observable and that the duration of each activity (as well as the other temporal constraints) are certain and under control of the system. Real world situations, however, show that activity durations are not always controllable, thus motivating the need of dealing with uncertainty in dynamic environments. A lot of research has been devoted to these issues: for example, in [36] the definition of Simple Temporal Networks with Uncertainty is given as an extension of Simple Temporal Networks [37], with distinction and support to controllable and uncontrollable events. Recent works deal with temporal planning aspects with uncertainty [38], and with the scheduling of multi-task applications [39] with uncertainty as well. Currently, our approach deals only with the partial observability of the environment, and the controllability issue is not taken into account.

Within the Artificial Intelligence field a number of alternative approaches to incompleteness exist, as well as other frameworks that support abductive reasoning (e.g., ASP [40]), which we will consider in future work. The aim of this contribution is to show that abduction, and in particular Abductive Logic Programming, is a natural choice for representing the problem of observed, incomplete traces. In addition, key features of SCIFF, such as native support of notions like expectations and fulfilment/violation of traces against model's prescriptions, make the formalism especially appealing to reason with the problem at hand.

6. Conclusions

We have presented an abductive framework to support business process compliance, by attacking the different forms of incompleteness that may be present in an execution trace, and by supporting also the temporal dimension in terms of constraints on the activity durations and between different activities. To this purpose, we introduced the notion of SPOT model, a coherent encoding into the SCIFF framework, and showed how different reasoning tasks can be addressed through the SCIFF proof procedure. A current limit is that SPOT models do not support loops. This is due to the lack of a clear semantics for temporal constraints between two activities: particularly ambiguous is the case when such a constraint

connects an activity involved in a loop. In previous work [33] we showed how to deal with loops using an encoding similar to the one presented here: however, [33] ignored the temporal dimension.

Concerning future development, the SCIFF framework is based on first-order logic, thus paving the way towards the incorporation of data [28] and the management of more sophisticated forms of incompleteness. A further reasoning service on temporal workflows is the one of controllability. We believe that an extension of our work to deal with dynamic controllability, for instance integrating constraint propagation and filtering as in [39], would be an interesting and feasible future work.

References

[1] Chesani F, De Masellis R, Francescomarino CD, Ghidini C, Mello P, Montali M, Tessaris S. Abducing Workflow Traces: A General Framework to Manage Incompleteness in Business Processes. In: Kaminka GA, Fox M, Bouquet P, Hüllermeier E, Dignum V, Dignum F, van Harmelen F (eds.), ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), volume 285 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. ISBN 978-1-61499-671-2, 2016 pp. 1734–1735. doi:10.3233/978-1-61499-672-9-1734. URL `http://dx.doi.org/10.3233/978-1-61499-672-9-1734`.

[2] van der Aalst WMP, et al. Process Mining Manifesto. In: BPM Workshops. Springer, 2012 .

[3] van der Aalst WMP. Business Process Management: A Comprehensive Survey. *ISRN Software Engineering*, 2013. doi:10.1155/2013/507984. Vol. 2013, Article ID 507984, 37 pages, 2013. doi:10.1155/2013/507984.

[4] Maggi FM, Montali M, van der Aalst WMP. An Operational Decision Support Framework for Monitoring Business Constraints. In: FASE, volume 7212 of *LNCS*. Springer, 2012 pp. 146–162.

[5] Governatori G, Milosevic Z, Sadiq S. Compliance Checking Between Business Processes and Business Contracts. In: Proc. of EDOC. IEEE Computing Society, 2006 pp. 221–232.

[6] Kumar A, Sabbella SR, Barton RR. Business Process Management: 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 – September 3, 2015, Proceedings, chapter Managing Controlled Violation of Temporal Process Constraints, pp. 280–296. Springer International Publishing, Cham. ISBN 978-3-319-23063-4, 2015. doi:10.1007/978-3-319-23063-4_20. URL `http://dx.doi.org/10.1007/978-3-319-23063-4_20`.

[7] Kakas AC, Kowalski RA, Toni F. Abductive Logic Programming. *J. Log. Comput.*, 1992. **2**(6).

[8] Denecker M, Kakas AC. Abduction in Logic Programming. In: Kakas AC, Sadri F (eds.), Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I, volume 2407 of *Lecture Notes in Computer Science*. Springer. ISBN 3-540-43959-5, 2002 pp. 402–436. doi:10.1007/3-540-45628-7_16.

[9] Alberti M, Chesani F, Gavanelli M, Lamma E, Mello P, Torroni P. Verifiable agent interaction in abductive logic programming: The SCIFF framework. *ACM Trans. Comput. Log.*, 2008. **9**(4).

[10] Kiepuszewski B, ter Hofstede AHM, Bussler CJ. On Structured Workflow Modelling. In: Seminal Contributions to Information Systems Engineering. Springer, 2013.

[11] Combi C, Gozzi M, Posenato R, Pozzi G. Conceptual modeling of flexible temporal workflows. *TAAS*, 2012. **7**(2):19.

[12] Kakas AC, Mancarella P. Abduction and Abductive Logic Programming. In: Proc. of ICLP. 1994 .

[13] Fung TH, Kowalski RA. The Iff Proof Procedure for Abductive Logic Programming. *J. Log. Program.*, 1997. **33**(2).

[14] Kunen K. Negation in Logic Programming. *J. Log. Program.*, 1987. **4**(4).

[15] Clark KL. Negation as Failure. In: Logic and Data Bases. Plenum Press, 1978.

[16] Jaffar J, Maher MJ, Marriott K, Stuckey PJ. The Semantics of Constraint Logic Programs. *J. Log. Program.*, 1998. **37**(1-3).

[17] Alberti M, Chesani F, Gavanelli M, Lamma E, Mello P, Torroni P. Verifiable Agent Interaction in Abductive Logic Programming: the SCIFF proof-procedure. Technical Report DEIS-LIA-06-001, University of Bologna (Italy), 2006. LIA Series no. 75.

[18] Bettini C, Wang XS, Jajodia S. Temporal Reasoning in Workflow Systems. *Distributed and Parallel Databases*, 2002. **11**(3):269–306.

[19] Eder J, Gruber W, Panagos E. Temporal modeling of workflows with conditional execution paths. In: Database and Expert Systems Applications. Springer, 2000 pp. 243–253.

[20] Rozinat A, van der Aalst WMP. Conformance Checking of Processes Based on Monitoring Real Behavior. *Inf. Syst.*, 2008. **33**(1):64–95.

[21] Adriansyah A, van Dongen BF, van der Aalst WMP. Conformance Checking Using Cost-Based Fitness Analysis. In: Proc. of EDOC. IEEE Computer Society, 2011 .

[22] de Leoni M, van der Aalst W, van Dongen BF. Data- and Resource-Aware Conformance Checking of Business Processes. In: Proc. of BIS. Springer, 2012. doi:10.1007/978-3-642-30359-3_5.

[23] Taghiabadi ER, Fahland D, van Dongen BF, van der Aalst WMP. Diagnostic Information for Compliance Checking of Temporal Compliance Requirements. In: CAiSE, volume 7908 of *LNCS*. Springer. ISBN 978-3-642-38708-1, 2013 pp. 304–320.

[24] Rogge-Solti A, Kasneci G. Temporal Anomaly Detection in Business Processes. In: Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings. 2014 pp. 234–249. doi:10.1007/978-3-319-10172-9_15.

[25] De Leoni M, Maggi FM, van der Aalst WMP. Aligning event logs and declarative process models for conformance checking. In: Proc. of BPM. Springer, 2012 .

[26] Bertoli P, Di Francescomarino C, Dragoni M, Ghidini C. Reasoning-Based Techniques for Dealing with Incomplete Business Process Execution Traces. In: Proc. of AI*IA. Springer, 2013 .

[27] Di Francescomarino C, Ghidini C, Tessaris S, Sandoval IV. Completing Workflow Traces Using Action Languages. In: Proc. of CAiSE. Springer, 2015 .

[28] De Masellis R, Maggi FM, Montali M. Monitoring data-aware business constraints with finite state automata. In: Proc. of ICSSP. ACM Press, 2014 doi:10.1145/2600821.2600835.

[29] Barba I, Weber B, Del Valle C. Supporting the Optimized Execution of Business Processes through Recommendations. In: Business Process Management Workshops (1), volume 99 of *LNBIP*. Springer, 2011 pp. 135–140.

[30] van der Aalst WMP, Schonenberg MH, Song M. Time Prediction Based on Process Mining. *Inf. Syst.*, 2011. **36**(2):450–475. doi:10.1016/j.is.2010.09.001.

[31] Chesani F, Mello P, Montali M, Storari S. Testing Careflow Process Execution Conformance by Translating a Graphical Language to Computational Logic. In: Proc. of AIME. 2007 .

[32] Montali M, Pesic M, van der Aalst WMP, Chesani F, Mello P, Storari S. Declarative specification and verification of service choreographiess. *TWEB*, 2010. **4**(1).

[33] Chesani F, De Masellis R, Francescomarino CD, Ghidini C, Mello P, Montali M, Tessaris S. Abducing Compliance of Incomplete Event Logs. In: Adorni G, Cagnoni S, Gori M, Maratea M (eds.), AI*IA 2016: Advances in Artificial Intelligence - XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 - December 1, 2016, Proceedings, volume 10037 of *Lecture Notes in Computer Science*. Springer. ISBN 978-3-319-49129-5, 2016 pp. 208–222. doi: 10.1007/978-3-319-49130-1_16. URL http://dx.doi.org/10.1007/978-3-319-49130-1_16.

[34] Mian US, den Hartog J, S Etalle NZ. Auditing with incomplete logs. In: Proc. of HotSpot. 2015 .

[35] Mancarella P, Terreni G, Sadri F, Toni F, Endriss U. The CIFF proof procedure for abductive logic programming with constraints: Theory, implementation and experiments. *TPLP*, 2009. **9**(6).

[36] Vidal T, Fargier H. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 1999. **11**:23–45.

[37] Dechter R, Meiri I, Pearl J. Temporal constraint networks. *Artificial Intelligence*, 1991. **49**(1):61–95.

[38] Cimatti A, Micheli A, Roveri M. Strong Temporal Planning with Uncontrollable Durations: A State-Space Approach. In: Proc.of AAAI. 2015 pp. 3254–3260.

[39] Lombardi M, Milano M, Benini L. Robust Scheduling of Task Graphs under Execution Time Uncertainty. *IEEE Trans. Computers*, 2013. **62**(1):98–111.

[40] Baral C. Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press, New York, NY, USA, 2003. ISBN 0521818028.