Data & Knowledge Engineering 109 (2017) 112–125

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Mining task post-conditions: Automating the acquisition of process semantics

Metta Santiputri[a,b], Aditya K. Ghose[a,*], Hoa Khanh Dam[a]

[a] Decision Systems Lab School of Computing and Information Technology University of Wollongong, Wollongong, NSW 2522, Australia
[b] Department of Informatics, Politeknik Negeri Batam, Batam 29461, Indonesia

ARTICLE INFO

Keywords:
Business process semantics
Mining post-conditions
Semantic annotation

ABSTRACT

Semantic annotation of business process model in the business process designs has been addressed in a large and growing body of work, but these annotations can be difficult and expensive to acquire. This paper presents a data-driven approach to mining and validating these annotations (and specifically context-independent semantic annotations). We leverage event objects in process execution histories which describe both activity execution events (typically represented as *process events*) and state update events (represented as *object state transition events*). We present an empirical evaluation, which suggests that the approach provides generally reliable results.

1. Introduction

A large and growing body of work explores the use of semantic annotation of business process designs [6,21,38,41,5,11] (we use the term *semantic annotation* to describe the annotation of process designs with semantic information, and specifically, post-conditions). A large body of work also addresses the problem of semantic annotation of web services in a similar fashion [29–31,37]. Common to all of these approaches is the idea that semantic annotation of process tasks or services provides value in ways that the process or service model alone cannot. Our focus in this paper is on *post-conditions* of tasks in the context of process models (pre-conditions are also of interest and we believe that an extension of the machinery presented here can address these, but are outside the scope of the present work). Ideally process designs annotated with post-conditions help answer the following question for any part of a process design: *what changes will have occurred in the process context if the process were to execute up to this point?* Arguably, a sufficiently detailed process model (for instance one that decomposes tasks down to the level of individual read or write operations) will require no additional information to answer this question. However, process models are most valuable when described at higher levels of abstraction, in terms of concepts and activities that stakeholders are familiar with. Processes annotated with post-conditions thus serve a crucial modeling function, providing an effective summary of a substantial body of knowledge regarding the "lower-level" workings of a process. Annotation with post-conditions can also help solve a range of problems such as process compliance management [11], goal satisfaction analysis [35], change management [25], enterprise process architectures [28] and the management of the business process life cycle [26].

The modeling and acquisition of these post-conditions poses a particularly difficult challenge. It is generally recognized that process modeling involves significant investment in time and effort, which would be multiplied manyfold if there were an additional obligation to specify semantic annotations. Analysts also tend to find semantic annotation difficult, particularly if the intent is to make these formal (as is required by all of the use cases referred to above). This paper seeks to address this challenge by offering a set

* Corresponding author.
E-mail addresses: ms804@uowmail.edu.au, metta@polibatam.ac.id (M. Santiputri), aditya@uow.edu.au (A.K. Ghose), hoa@uow.edu.au (H.K. Dam).

http://dx.doi.org/10.1016/j.datak.2017.03.007
Received 2 March 2017; Received in revised form 2 March 2017; Accepted 2 March 2017
Available online 24 March 2017

of techniques that mine readily available data associated with process execution to generate largely accurate "first-cut" post-conditions for process tasks or activities (we use the terms "task" and "activity" interchangeably in this paper).

Our approach leverages the generally understood notion of *event logging*. The events that occur in a process execution context can be viewed in general terms as being of two types: (1) events that describe the start or end of the execution of process activities and (2) events that describe state changes in the objects impacted by a process. In many settings, the existing event logging machinery is capable of logging both kinds of events. One such approach on event logging is the event processing framework for business process management by Herzberg et al. [16–20].

We leverage these two types of events in juxtaposition, and the time-stamped sequences of activity execution events and state-change events thus obtained, to generate the *sequence database* taken as input by a sequential rule miner (CMRules [7] in our instance, but others could be used instead). The key idea is to identify commonly occurring patterns of activity execution events, followed by sequences of state change events. As we show, the approach is generally quite effective. We also define techniques which leverage a *state update operator* (that defines how a specification of a state of affairs is updated as a consequence of the execution of an action) and the actual history of process execution provided by the juxtaposed activity executions and state changes to determine whether the mined post-conditions, if accumulated using the state update operator, would indeed generate the available execution histories. This forms a validation step for the mined results.

Our intent is to mine the *context-independent post-conditions* (or immediate outcome) of each activity. These are *contextualized* via iterated applications of the state update operator to obtain the *context-dependent post-conditions* of each activity (in the context of a process model)—a complete collection of these for each activity or event provides a semantically annotated process model. For instance, the outcome of turning a switch on is to complete a circuit. In the context of a light bulb circuit, the context-dependent post-conditions of this activity would be to turn the bulb on. In the context of a switching circuit for a chemical reactor, the context-dependent post-conditions of that same activity would be to bring the chemical reactor to an operational state. We envisage the machinery we present below being used in the following manner: given as input a set of events that describe the execution of activities, a set of state-change events, a process model (or a set of process models in the event that the logs describe the execution of instances of multiple process designs) and a state update operator, the machinery would generate the post-conditions of each activity referred to in the recorded events. These post-conditions could be used directly in annotating process models, or might be viewed as "first-cut" specifications, to be edited and refined by expert analysts.

The problem we solve can be summarized as follows. Given: (1) a log of process events, (2) a log of object state transition events, (3) a process model or models whose execution generated these logs and (4) a state update operator, compute: the context-independent post-conditions of every task/activity referred to in the process event log. Inputs (1) and (2) are used in the mining phase, while inputs (3) and (4) are used in the validation phase.

This paper extends the results presented in [36] in a number of important ways. First, this paper presents a more sophisticated approach to validation. Second, it offers a novel abductive framework for repairing mined post-conditions, based on soundness and completeness analysis contained in the validation approach. Third, the paper presents more extensive empirical analysis.

The rest of the paper organize as follows. We provide a running example in Section 2. In Section 3, we describe the event ontology that our approach uses. In Section 4, we describe the approach to semantic annotation of process models that sits at the core of our proposal. In Section 5, we describe the post-condition mining algorithm. In Section 6, we describe a sophisticated approach to validating the knowledge mined, while in Section 7, we provide an abductive approach to repairing the post-conditions that we mine. Section 8 presents an empirical evaluation of the proposal. Section 9 describes related work, while Section 10 presents conclusions.

2. Example

Process designs are intended to be abstract, enabling users to get a handle on a complex underlying reality. Thus the *effects* or *impact* of a process is often not directly reflected in the high-level abstractions contained in a process design. Our proposal offers a means of mining these effects and correlating these with elements of a process design. Compelling examples of such processes can be found in domains such as medicine, logistics, financial services and so on. We will use a clinical process as the running example in this paper.

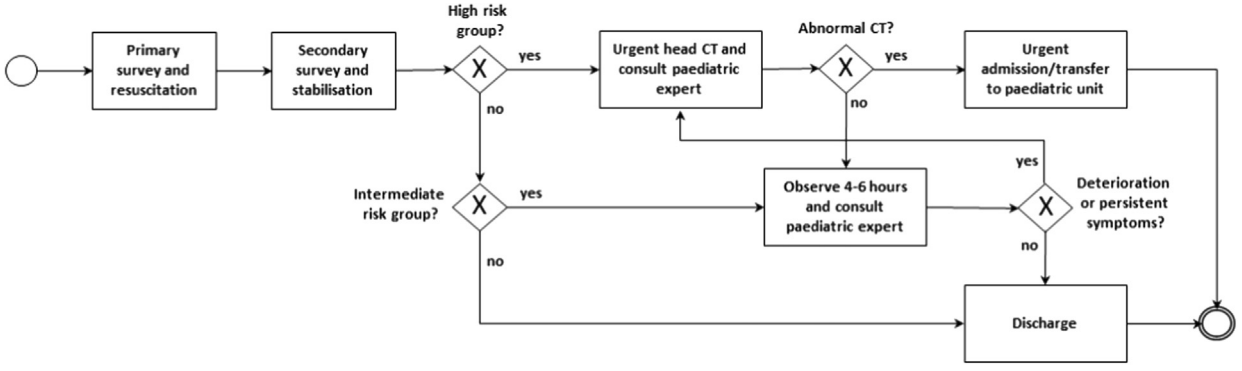

Fig. 1. Clinical process for treatment of juveniles with head injuries [33].

Specifically, we will focus on a clinical process for the treatment of juveniles with head injuries, drawn from [33]. Fig. 1 illustrates the complete process of head injury treatment. Initial evaluation aims to quickly determine the severity of injury and to initiate the appropriate treatment immediately. After the primary and secondary survey, the patient with head injury is treated according to the risk category. Patients with high risk of intracranial injury have to undergo a head CT scan and a consultation with a paediatric expert. Any abnormalities observable on a CT scan should be treated according to neurosurgical advice. In the absence of abnormalities, a period of prolonged observation is required due to the risk of cerebral oedema or delayed bleeding. This extended period of observation also applies for any patients displaying features of an intermediate risk group. If an acute deterioration or any persistent symptoms (vomiting, headache, irritability, abnormal behavior or unsteady gait) is detected at six hours after injury, a head CT is indicated. Otherwise, the patient may be discharged.

Consider four patients with different conditions. We describe the process instances for two of these patients below, while Table 1 describes the task sequence that applied to all four patients (we omit details for *patient3* and *patient4* which would be quite similar, and not necessary for the objectives of this example):

patient1 Patient presented as a member of the high risk group (abnormal cardio-respiratory function, loss of consciousness for more than 5 minutes, retrograde amnesia more than 5 minutes, abnormal behaviour, abnormal drowsiness, seizure although the patient is non-epileptic, non-accidental injuries, persistent headache, co-morbidity, fall from higher than 3 m height, laceration on the head, low GCS, oxygen saturation less than 95%, intubated). After the patient had undergone a head CT, the results indicated intra-cerebral bleeding, therefore the patient was transferred to the paediatric unit.

patient2 Patient presented as a member of the high risk group (normal cardio, abnormal respiratory, loss of consciousness for more than 5 minutes, retrograde amnesia more than 5 minutes, with abnormal behaviour, abnormal drowsiness, seizure although the patient is non-epileptic, non-accidental injuries, persistent headache, co-morbidity, victim of motor vehicle accident, swelling and laceration on the head, low GCS, oxygen saturation less than 95%, intubated). After the patient underwent a head CT, the results came back as normal, therefore the patient was put under observation for 4-6 hours. During the observation period, there was no further deterioration and the symptoms resolved, therefore the patient was discharged.

Table 1 shows an event log that records the sequence of clinical interventions for each of *patient1, patient2, patient3 and patient 4*.

Table 2 stores the condition of each patient (for ease of exposition, we only show the records for *patient1* and *patient2*). Every change in a patient's condition is recorded in this table together with a time-stamp. We use an underlying clinical vocabulary (or a state description language) to represent a patient's condition. For instance, in the first record, at time $t1$, *patient1*'s heart rate and blood pressure are measured and categorized as normal (represented as *heart_rate(patient1, normal)* $\wedge$*blood_pressure*(*patient1*, *normal*)). The condition of *patient2* is much the same when assessed at time $t5$ (represented as *heart_rate*(*patient2*, *normal*) $\wedge$*blood_pressure*(*patient2*, *normal*)). At time $t11$, *patient1* is intubated. The most obvious effect of this clinical intervention is recorded in the table as *intubated(patient1)*.

3. An event ontology

We derive our approach from the event processing framework for business process management by Herzberg et al. [16–20]. In this framework, a process model is correlated with a set of data objects and each data object has a defined life cycle. The notion of a

Table 1
Records of patient's treatment.

Time	Patientid	Treatment
t1	patient1	primary survey and resuscitation
t5	patient2	primary survey and resuscitation
t27	patient2	secondary survey and stabilisation
t30	patient1	secondary survey and stabilisation
t54	patient3	primary survey and resuscitation
t77	patient4	primary survey and resuscitation
t82	patient3	secondary survey and stabilisation
t84	patient4	secondary survey and stabilisation
t105	patient1	urgent head CT and consult paediatric expert
t124	patient4	discharge
t126	patient2	urgent head CT and consult paediatric expert
t135	patient3	observe 4-6 hours and consult paediatric expert
t141	patient2	observe 4-6 hours and consult paediatric expert
t148	patient1	urgent admission/transfer to paediatric unit
t154	patient2	discharge
t162	patient3	urgent head CT and consult paediatric expert
t173	patient3	urgent admission/transfer to paediatric unit

Table 2
Records of patient's conditions.

Time	Patientid	Conditions
t1	patient1	heart_rate(patient1, normal) ∧blood_pressure(patient1, normal)
t2	patient1	normothermia(patient1)
t3	patient1	¬ oxygen_saturation(patient1, normal) ∧¬ PaO2_level(patient1, normal) ∧¬ PaCO2_level(patient1, normal)
t4	patient1	GCS(patient1, low)
t5	patient2	heart_rate(patient2, normal) ∧blood_pressure(patient2, normal)
t6	patient2	normothermia(patient2)
t7	patient2	¬ oxygen_saturation(patient2, normal) ∧¬ PaO2_level(patient2, normal) ∧¬ PaCO2_level(patient2, normal)
t8	patient2	GCS(patient2, low)
t9	patient2	cervical_spine(patient2, immobilise)
t10	patient1	cervical_spine(patient1, immobilise)
t11	patient1	intubated(patient1)
t12	patient1	systemic_blood_pressure(patient1, adequate)
t13	patient1	maintenance_fluids_administered(patient1)
t14	patient1	opiates_administered(patient1)
t15	patient1	sedation_score(patient1, high)
t16	patient1	blood_glucose(patient1, normal)
t17	patient1	analgesia_administered(patient1)
t18	patient1	anti_emetics_administered(patient1)
t19	patient2	intubated(patient2)
t20	patient2	systemic_blood_pressure(patient2, adequate)
t21	patient2	maintenance_fluids_administered(patient2)
t22	patient2	opiates_administered(patient2)
t23	patient2	sedation_score(patient2, high)
t24	patient2	blood_glucose(patient2, normal)
t25	patient2	analgesia_administered(patient2)
t26	patient2	¬ anti_emetics_administered(patient2)
t27	patient2	loss_of_consciousness(patient2)
t28	patient2	¬ anterograde_amnesia(patient2) ∧retrograde_amnesia(patient2)
t29	patient2	mild_agitation(patient2) ∧altered_behaviour(patient2) ∧¬ abnormal_drowsiness(patient2)
t30	patient1	loss_of_consciousness(patient1)
t31	patient1	¬ anterograde_amnesia(patient1) ∧retrograde_amnesia(patient1)
t32	patient1	¬ mild_agitation(patient1) ∧¬ altered_behaviour(patient1) ∧abnormal_drowsiness(patient1)
t33	patient1	vomiting_without_other_cause(patient1)
t34	patient1	seizure(patient1) ∧non_epileptic(patient1)
t35	patient2	vomiting_without_other_cause(patient2)
t36	patient2	seizure(patient2) ∧non_epileptic(patient2)
t37	patient2	non_accidental_injury(patient2)
t38	patient2	headache(patient2)
t39	patient2	co-morbidities(patient2)
t40	patient2	¬ age_under_1yr(patient2)
t41	patient2	motor_vehicle_accident(patient2) ∧¬ fall(patient2)
t42	patient2	GCS(patient2, low)
t43	patient2	focal_neurological_abnormality(patient2)
t44	patient2	¬ penetrating_injury(patient2)
t45	patient2	¬ suspected_depressed_skull_fracture(patient2) ∧¬ suspected_depressed_base_of_skull_fracture(patient2)
t46	patient2	¬ scalp_bruise(patient2) ∧swelling(patient2) ∧laceration(patient2)
t47	patient2	¬ tense_fontanelle(patient2)
t48	patient1	non_accidental_injury(patient1)
t49	patient1	headache(patient1)
t50	patient1	co-morbidities(patient1)

data object permits us to abstract information (of various kinds including information that reflects states in the life-cycle of real-woprld objects) being processed or manipulated during process execution [19].

During process execution, a wide variety information about changes or exceptions in the business process environment can be represented through event objects, e.g. the start of a certain activity, the state change of certain data object, etc. In this paper, we focus on only two types of event objects: (1) *process events* which record the start of the execution of a task or activity, and (2) *object state transition events* that describe the impact of process execution via state changes in the impacted objects (which could be computational objects, such as data items, or real-world objects, such as a piece of machinery or a switch). We are only interested in recording the state of these objects that are the result of the state transitions, and do not record the prior states.

Since object state transition events represent the effects of executing a process, we will on occasion use the terms *object state transition* and *effect* interchangeably.

We can now relate these event types to our running example from the previous section. The process events in that example are recorded in Table 1. The object state transition events in that example are recorded in Table 2. It is useful to note that these latter events essentially describe the condition of a patient. For example, the first row at Table 1 indicates that activity *primary survey and*

resuscitation was started at time $t1$. The first row in Table 2 indicates at time $t1$ the condition of *patient1* as *heart_rate*(*patient1*, *normal*) *wedge blood_pressure*(*patient1*, *normal*). In this example, the objects are the patients and the object state transition events describe various aspects of the state of a patient after a particular activity/medical intervention has been performed.

These event objects can be obtained by instrumenting the process environment with object state monitors (both for physical objects as well as for computational/business objects). For our purpose, we describe the state changes or transitions using the state description language that might involve propositional state variables—the changes to describe would then be propositions becoming true or false, or more generally as disjunctions (in case state monitors have limited sensing capabilities). The underlying language might also admit non-Boolean state variables, in which case the states recorded would be the new value assignments to these objects. When annotating a process model with object state transitions caused by each task, it is convenient to use first-order sentence schemas. Thus, we would use a sentence schema such as *heart_rate*(*Patient*, *Status*), which would be instantiated with a ground sentence such as *heart_rate*(*patient1*, *normal*) in a log of object state transitions.

In our head injury treatment example, the "*primary survey and resuscitation*" activity would lead to a ground instance of the sentence schema *normothermia*(*patient*1) becoming available. In this setting, the precise grounding of the *Patient* objects are not of particular interest. Indeed, recording the actual values of these objects would lead to our procedure treating different groundings as distinct objects, when in fact we are only interested in recording the fact that a ground instance of that sentence schema has become available. For states of this sort, we only record a propositional effect of the form *normothermia-known*. In a similar fashion, it is sufficient to record *patient-heart-rate-known* rather than the fact that *patient1* has a *normal* heart-rate (as described in *heart_rate*(*patient1*, *normal*)). In other settings, we are interested in the precise instantiations of the objects in a sentence schema of the form $p(X, Y)$, in which case the full ground instance of $p(X, Y)$ is recorded in the object transition events table.

Our approach to mine the activity post-conditions involves (1) correlating process events and object state transition events as represented in the database (in this section), and then (2) filtering these by validating them (in the next section).

4. Semantic annotation

We assume that each task or event in a process is associated with post-conditions written as conjunctive normal form sentences in the underlying formal state description language, which might be propositional or first-order (we do not consider temporal logics in this work, but extensions are possible). We assume that each task or event has context-independent *post-conditions* that can be contextualized via iterated applications of a *state update operator* as in [11] and [21]. We permit the contextualized post-conditions to be non-deterministic—at any given point in a process, the actual states that might accrue would be one of a set of possible states. We need to support this non-determinism for two reasons. First, in any process with XOR-branches, one might arrive at a given task via multiple paths, and the contextualized states achieved must be contingent on the path taken. Since this analysis is done at design time, we need to admit the possibility of non-deterministic states since the specific path taken can only be determined at run-time. Second, many state update operators generate non-deterministic outcomes, since inconsistencies (that commonly appear in state update) can be resolved in multiple different ways. Of the two well-known state update operators in the literature—the Possible Models Approach (PMA) and the Possible Worlds Approach (PWA)—our work leverages the PWA [12]. Specifically, we use the operator $\oplus$defined below. In the following, we will leverage a background knowledge base *KB* (where present).

For two states s_i and s_j, and the knowledge base *KB*, if $s_i \not\models \bot$ and $s_j \not\models \bot$, then the *pair-wise effect accumulation* (or *state update*) $s_i \oplus s_j$ is defined as:

$$s_i \oplus s_j = \{s_j \cup s'_i | s'_i \subseteq s_i \wedge s'_i \cup s_j \cup KB \not\models \bot \wedge \text{there does not exist } s''_i \text{ such that } s'_i \subset s''_i \subseteq s_i \wedge s''_i \cup s_j \cup KB \not\models \bot\}$$

We will occasionally need to refer to a general version, $\oplus$, of the state update operator $\oplus$. If S is a set of states $\{s_1, \ldots, s_n\}$, then $S \oplus s = \{s_i \oplus s | s_i \in S\}$.

The outcome of the state update operation is not a unique state specification, but a set of non-deterministic possible state set. To see why this might be the case, consider a task T with a single associated post-conditions given by $\{p, q\}$ which is followed by task T' whose outcome is to make r true. Given a background knowledge base consisting of a single rule $r \rightarrow (\neg p \vee \neg q)$, the $\oplus$ operator would give us two distinct outcomes: $\{p, r\}$ and $\{q, r\}$.

To obtain a complete annotation of a process model, we repeatedly apply the $\oplus$ operator over pairs of contiguous tasks in a process model, with the first argument being the post-conditions associated with the prior task and the second argument being the post-conditions of the later task. Special techniques are provided for dealing with XOR and AND gateways in proposals such as [11,21] and [41], but these are not directly relevant for our current exposition and we omit details here.

5. Mining post-conditions

Our approach to post-conditions mining is predicated on the observation that the state transitions of objects impacted by executing an activity occur soon after the execution of the activity. State transitions that manifest a long period after the execution of an activity are typically not the effect of that activity alone, but of that activity plus some others (e.g., one may think of the arrival of a traditional "snailmail" letter 3 days after posting as an outcome of the action of letter-posting, when it actually involves several other activities executed by the postal service). The key pattern we leverage in mining post-conditions is the *sequence* that involves the execution of an activity and the manifestations of its *object state transitions*, using a sequential pattern miner. We are interested in identifying all the state transitions that occur always (or most of the time) after each activity is executed. Since the process executions are recorded as event objects and the state transitions occurrences are recorded as object transition events, we must first establish

the correlations between the two tables that records both events to obtain a joined table that serves as the *sequence database* for a sequential rule miner. We use the CMRules algorithm [7] although a number of other candidates exist [3,8,9,15], and the framework is flexible enough to allow the use of any of these.

While our focus is on the sequential patterns that relate event objects to object state transitions, we are not interested in the relative sequencing amongst state transitions. Indeed, it is undesirable for our purposes to have the sequential rule miner to view the sequences $\langle T, p, q\rangle$ and $\langle T, q, p\rangle$ as being distinct. We therefore enforce the rule that a contiguous sequence of state transitions in the sequence database must always be represented in lexicographic order (this would require the second sequence above to be re-written as the first sequence).

We consider the problem of post-conditions mining in two settings: (1) Settings characterized by the **unique activity assumption** which stipulates that only one activity may be performed at any point in time. This permits us to correlate all of the state transitions observed between the execution of a given activity and the start of the next activity with the first activity. (2) Settings characterized by the **concurrent actvity assumption** which admits the possibility of multiple activities executing concurrently (these could be activities associated with distinct instances of the same process or associated with different processes). The second setting is more general, but the first setting simplifies the post-conditions mining problem, and is worth considering if appropriate. We will apply the CMRules algorithm in both settings.

In general, a sequential rule $X \rightarrow Y$ consists of two parts: the antecedent X and the consequent Y, which are both assumed to be sequences of transactions. The rule states that if the elements of X occur in a given sequence in the sequence database being mined, then the elements of Y will follow in the same sequence and in a manner that preserves the sequential relations between the elements of X and between the elements of Y. All sequential rules must also satisfy certain criteria regarding their accuracy (minimum confidence) and the proportion of the data that they actually represent (minimum support). The CMRules algorithm takes as input a sequence database along with two user-specified thresholds: minimum support (*minSeqSup*) and minimum confidence (*minSeqConf*). It outputs the set of all sequential rules that satisfy the *minSeqSup* and *minSeqConf* thresholds. The algorithm consists of two steps. The first step involves obtaining a transaction database from a sequence database without considering the sequential information. The algorithm then finds all association rules from the transaction database using an association rule mining algorithm, such as Apriori [1]. All association rules discovered must satisfy the minimum support and minimum confidence thresholds which is set equal to *minSeqSup* and *minSeqConf*. In the next step, the algorithm then scans the original sequence database to calculate the support and confidence of each association rule, and eliminates the rules that do not satisfy *minSeqSup* or *minSeqConf*. The rules that satisfy both thresholds are considered as sequential rules. To apply this algorithm in our setting, we must first combine the *process event log* and the *object state transition log* into a *sequence database*.

Both in settings with the *unique activity assumption* and in settings with concurrent activities, we create a joined table from the event objects and the object transition events of the form: $\langle\langle T_l, \langle\langle e_{11}\rangle, \ldots, \langle e_{1n}\rangle\rangle\rangle, \ldots, \langle T_i, \langle\langle e_{i1}\rangle, \ldots, \langle e_{im}\rangle\rangle\rangle, \ldots, \langle T_p, \langle\langle e_{p1}\rangle, \ldots, \langle e_{pk}\rangle\rangle\rangle\rangle$ where each $\langle T_i, T_{i+1}\rangle$ pair represents contiguous activities and each e_{ij} represents the j-th state transition observed after the start of activity T_i and before the start of activity T_j. We shall henceforth refer to this table as the *Joined ProcessEvent-StateTransitionEvent table*. This table serves as the sequence database provided as input to the sequential rule miner. A special provision is needed for the last activity in case it does not have any subsequent activity. Instead of using the last record in the event objects table as the end timestamp, we assume that we have prior information about the maximal time of process execution, ϵ, and use it as the end time of the last activity in any case.

We then apply the CMRules algorithm, with the best results obtained when the values of *minSeqSup* and the *minSeqConf* are bounded from below by the number of distinct case-ID in which a specific activity occurs (as with any association rule mining technique, *minSeqSup* and *minSeqConf* represent the support and confidence respectively—higher values of these can give us more reliable results but rule out potentially interesting rules and vice versa). In *unique activity* settings with no noise, the sequence of state transitions following the exection of each activity and prior to the execution of the next activity in the process instance should be largely identical if the process design is fixed—we apply CMRules mainly to mitigate the effects of noise. In *concurrent activity* settings, these could vary significantly since the state transitions that follow an activity might not be the output of that particular activity but those of a distinct concurrent activity. In these settings, the sequential rule miner is essential to identify the commonly occurring patterns of state transitions following a given activity.

For example, consider *patient*1 in Table 2. The first activity, *primary survey and resuscitation*, has timestamp t_1 and the next activity for the same patient, *secondary survey and stabilisation*, has timestamp t_{30}; therefore, we associate activity *primary survey and resuscitation* with all state transitions observed between the timestamps t_1 and t_{30}. This gives us the sequence (*primary survey and resuscitation*)(*heart-rate-known*)(*blood-pressure-known*)(*normothermia-known*)(*oxygen-saturation-known*)(*PaO2-level-known*)(*PaCO2-level-known*)(*GCS-known*), etc. Similarly, activity *secondary survey and stabilisation* is associated with all state transitions with timestamps between t_{30} and t_{105}, and so on. Applying the same process to all the other cases, we obtain the sequences for all activities in the process instance for *patient*1. Next, these sequences are grouped into a sequence database based on their activity name. For example, the sequence for task *primary survey and resuscitation* for *patient*1 goes into the same sequence database with the sequence for the task *primary survey and resuscitation* for *patient*2 (along with task *primary survey and resuscitation* sequences for other patients).

Although the CMRules algorithm is able to generate all sequential rules from the sequence databases, further post-processing is required. Since we are interested only in relations between an activity and state transitions, only rules with a single activity name as antecedent are included in the results and all other rules are discarded.

6. Validation

We can use the state update operator and the available data to *validate* the mined post-conditions. The intuition is to leverage available data to determine if the mined post-conditions predict the object state transitions seen in the data. We offer tests for soundness and completeness, and an abductive framework to guide the repair of mined post-conditions. We consider two settings, the first mainly for tesing purposes and the second because it reflects real-life operations.

Unique activity assumption: The analysis described below can be performed in settings satisfying the **unique activity assumption** which precludes multiple concurrently executing process instances. Note that such settings are rare in practice. This analysis is nonetheless useful for two reasons. First, it is possible to create *test runs* of processes that satisfy this assumption. Second, this affords the opportunity to develop the overall validation approach, which is subsequently specialized for the more practical setting.

A joined ProcessEvent-StateTransitionEvent table associates with each task a set of effects that occured after the execution of that task and prior to the execution of the next task in the execution sequence. We use the following procedure to obtain, from a given joined ProcessEvent-StateTransitionEvent table, a *cumulative joined ProcessEvent-StateTransitionEvent table*. Each row in the latter associates with each task the set of *accumulated effects* of all tasks executed up to this point. Note that the remainder of our exposition ignores the initial state that accrued at the start of the process (mainly to reduce the complexity of the formalization), but this can be trivially added if required. Let each row in the joined ProcessEvent-StateTransitionEvent table be of the form $\langle T_i, E_i \rangle$ where E_i is a set of literals (i.e., indicators of object state transition events). We assume that there is also a background knowledge base *KB* defined in the same language as that in which the effects are described.

The procedure involves the following steps:

- We set the first entry of the cumulative joined ProcessEvent-StateTransitionEvent table to be $\langle T_1, \{E_1\} \rangle$.
- We obtain each subsequent entry in the cumulative joined ProcessEvent-StateTransitionEvent table (of the form $\langle T_i, \mathbb{E}_i \rangle$) from the prior entry using the following rule: $\mathbb{E}_{i+1} = \mathbb{E}_i \oplus E_{i+1}$.

The following example illustrates how this is done (we use this procedure to obtain Table 4 from Table 3).

An element of the joined ProcessEvent-StateTransitionEvent table can be viewed as a *semantic execution trace* (i.e., a sequence of tasks interleaved with observed effects after each task), or part of one, of the form: $\langle\langle T_1, \langle\langle e_{11}\rangle, \ldots, \langle e_{1n}\rangle\rangle\rangle, \ldots, \langle T_i, \langle\langle e_{i1}\rangle, \ldots, \langle e_{im}\rangle\rangle\rangle, \ldots \langle T_p, \langle\langle e_{p1}\rangle, \ldots, \langle e_{pk}\rangle\rangle\rangle$ for a process instance (case) with p activities, with each T_i representing an activity ID and each e_{ij} representing the result of the j-th state transition associated with task T_i. An element of the cumulative joined ProcessEvent-StateTransitionEvent table associates with each task both the effects observed after the execution of that task and the effects of prior tasks that persist. These are obtained, as shown above, by applying the state update operator. Since the outcome of the application of the state update operator can be non-deterministic in general, we associate with each task a set of sets of effects (as illustrated with task T_3 in Table 4 above). We shall refer to the sequence of activities $\langle T_1, \ldots, T_p \rangle$ as the *signature* of the semantic execution trace above, and note that multiple semantic execution traces might be obtained for the same signature (due to the fact that we might find the process in one of many possible non-deterministic states after the execution of a sequence of activities).

To validate the post-conditions mined using the procedure described in the previous section, it is useful to establish:

- *Soundness*: The soundness condition states that the mined post-conditions are correct, i.e., observed in the data. In other words, mined post-conditions, accumulated via the state update operator up to a given point in a process must be included in the observed set of accumulated post-conditions at that point in the process. Formally, for each semantic execution trace manifested in a cumulative joined ProcessEvent-StateTransitionEvent table and for each activity T_i there must exist an associated set of observed (accumulated) effects es_i (the entry in the cumulative joined ProcessEvent-StateTransitionEvent table corresponding to T_i), such that the following holds: $es_i \cup KB \models e$ for some $e \in e_{T_1} \oplus e_{T_2} \oplus \ldots \oplus e_{T_i}$ where each e_{T_i} denotes the mined post-conditions of activity T_i (recall that the application of the $\oplus$operator can lead to multiple non-deterministic outcomes, making $e_{T_1} \oplus e_{T_2} \oplus \ldots \oplus e_{T_i}$ a set). For a sufficiently extensive collection of process and object state transition events, we may also require that there must exist, for every $e \in e_{T_1} \oplus e_{T_2} \oplus \ldots \oplus e_{T_i}$, some entry in the cumulative joined ProcessEvent-StateTransitionEvent table with an es_i associated with T_i such that $es_i \cup KB \models e$.
- *Completeness*: The completeness condition requires that all observed post-conditions are mined. This is essentially the reverse of the previous entailment relation (i.e., $e \cup KB \models es_i$).

Table 3
An example of joined ProcessEvent-StateTransitionEvent table.

T_i	E_i
T_1	p, q
T_2	r, s
T_3	t

Table 4
An example of cumulative joined ProcessEvent-StateTransitionEvent table.

$\mathcal{KB}$: $t \rightarrow \neg (p \wedge r)$ T_i	$\mathbb{E}_i$
T_1	$\{\{p, q\}\}$
T_2	$\{\{p, q, r, s\}\}$
T_3	$\{\{p, q, s, t\}\{r, q, s, t\}\}$

Concurrent tasks: In settings which permit multiple active process instances and where multiple tasks might be concurrently executed, we cannot guarantee that the post-conditions observed between the start of an activity and the start of the next activity in the same process instance are necessarily the post-conditions of the former task (since concurrent activities from other process instances might have led to these states). In such settings, we validate by creating modified sequence databases, parameterized by an activity sequence length parameter n for use with CMRules. For instance, when the activity sequence length parameter is 2, for each contiguous pair of tasks $\langle T_i, T_j \rangle$, we take sequences of the form $\langle T_i, e_{i1}, \ldots, e_{in} \rangle$ and $\langle T_j, e_{j1}, \ldots, e_{jm} \rangle$ where the activities belong to the same process instance and where the timestamps associated with each e_{ik} is earlier than the start of T_j and create an entry in this modified sequence database of the form $\langle T_i, T_j, \tau(e_{i1} \wedge e_{i2} \wedge \ldots \wedge e_{in} \oplus e_{j1} \wedge e_{j2} \wedge \ldots \wedge e_{jm}) \rangle$[1]. The result of applying τ represents the result of performing state update on the effects of T_i with the effects of T_j. If the state update operation leads to multiple non-deterministic outcomes, we create separate entries for each (sharing the same prefix $\langle T_i, T_j \rangle$). We use CMRules to obtain rules of the form $\langle T_i, T_j \rangle \rightarrow \langle e_1, \ldots, e_p \rangle$ with the support and confidence being set as earlier to refer only to those process instances where T_i and T_j appear contiguously. We can now use the following *soundness* condition: There exists a modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ such that the corresponding suffix (viewed as the conjunction of its elements) $e_1 \wedge \ldots \wedge e_n \cup KB \models e$ for every $e \in e_{T_i} \oplus e_{T_j}$. We can similarly state a *completeness* condition: For every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ and a corresponding suffix $e_1 \wedge \ldots \wedge e_n$ (as before, we view the suffix as the conjunction of its elements), there must exist an $e \in e_{T_i} \oplus e_{T_j}$ such that $e \cup KB \models e_1 \wedge \ldots \wedge e_n$ (note that this will work only if we deal with contiguous sequences tasks starting with the first task). The approach generalizes to task sequences of arbitrary length, but we omit details for ease of exposition. A general validation strategy is to consider all task sequences of length $i = 1, \ldots, n$ where n is the length of the longest task sequence that conforms to the process design.

7. Abductive repair

We now consider the problem of what needs to be done when mined post-conditions are found to be unsound or incomplete according to the tests described above. An easy solution is to seek more data and mine again. More interestingly, we can offer guidance to analysts in manually modifying the first-cut post-conditions mined from available data by using a simple formulation as an abductive problem. Our discussion focuses on settings with concurrent tasks, but the approach easily extends to the simpler class of settings satisfying the unique task assumption.

We consider first the case where the mined set of task post-conditions are found to be incomplete. If we start our analysis with activity sequences of length 2, let $\langle T_i, T_j \rangle$ be the first pair of contiguous activities for which we violate the completeness condition. A finding of incompleteness entails that there are effects (object state transtions) observed in the data which are not predicted by the mined post-conditions. In other words, the mined post-conditions need to be *augmented* to redress this. We need to decide now what post-conditions to add and to which task. Formally, the abductive problem is to identify the minimal (with respect to set inclusion) $a \subseteq A$ where A is the set of *abducibles* (in this case the vocabulary of post-conditions being used), given mined post-condition e_{T_i} and e_{T_j} for tasks T_i and T_j such that at least one of the following hold:

- There exists an $e \in (e_{T_i} \wedge a) \oplus e_{T_j}$ for every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ and a corresponding suffix $e_1 \wedge \ldots \wedge e_n$ (from here on we will view effect sequences as the conjunction of their elements for simplicity) such that $e \cup KB \models e_1 \wedge \ldots \wedge e_n$
- There exists an $e \in e_{T_i} \oplus (e_{T_j} \wedge a)$ for every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ and a corresponding suffix $e_1 \wedge \ldots \wedge e_n$ such that $e \cup KB \models e_1 \wedge \ldots \wedge e_n$

The first condition above corresponds to augmenting the post-conditions of T_i with a while the second corresponds to augmenting the post-conditions of T_j with a. If both conditions can be satisfied, we make a non-deterministic choice of any one task (and augment its post-conditions).

We consider next the case where the mined set of task post-conditions are found to be unsound. If we start our analysis with activity sequences of length 2, let $\langle T_i, T_j \rangle$ be the first pair of contiguous activities for which we violate the soundness condition. A finding of unsoundness entails that there are mined post-conditions that are not observed in the data (object state transtions). In other words, we need to *restrict* or *contract* one or more sets of mined post-conditions to redress this. We need to identify $e_{T_i'} \subseteq e_{T_i}$

[1] τ is a function that takes a sentence in conjunctive normal form and outputs a sequence consisting of its conjuncts (recall that the relative sequencing between these is of no interest from our perspective). Thus $\tau(e_{i1} \wedge e_{i2} \wedge \ldots \wedge e_{in}) = \langle e_{i1}, e_{i2}, \ldots e_{in} \rangle$. This function is need mainly as a matter of notational convenience.

and $e_{T_j'} \subseteq e_{T_j}$ such that both of the following hold:

- There exists an $e \in e'_{T_i} \oplus e'_{T_j}$ for every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ (with a corresponding suffix $e_1 \wedge \ldots \wedge e_n$) such that $e_1 \wedge \ldots \wedge e_n \wedge KB \models e$
- There exists no e''_{T_i} where $e'_{T_i} \subset e''_{T_i} \subseteq e_{T_i}$ and no e''_{T_j} where $e'_{T_j} \subset e''_{T_j} \subseteq e_{T_j}$ which satisfies the condition that there exists an $e \in e''_{T_i} \oplus e''_{T_j}$ for every modified sequence database entry with the prefix $\langle T_i, T_j \rangle$ (with a corresponding suffix $e_1 \wedge \ldots \wedge e_n$) such that $e_1 \wedge \ldots \wedge e_n \wedge KB \models e$

The e'_{T_i} and/or e'_{T_j} identified via this analysis are set as the new post-conditions of T_i and T_j respectively.

We need to start this analysis with the first task T_1 (since the modified sequence database includes the accumulated effects of *all* tasks starting with the first), then incrementally expand the sequence of contiguous tasks. Thus the first sequence of tasks considered would be $\langle T_1, T_2 \rangle$, then $\langle T_1, T_2, T_3 \rangle$ and so on. Once we have ensured that a given sequence of mined post-conditions $\langle e_{T_1}, \ldots e_{T_i} \rangle$ is sound and complete, we expand the sequence by one task, obtaining $\langle e_{T_1}, \ldots e_{T_i}, e_{T_{i+1}} \rangle$. Ensuring the new mined post-condition (i.e., $e_{T_{i+1}}$) is sound and complete is simpler, since only one candidate set of post-conditions needs to be either augmented or contracted. As above, repairing $e_{T_{i+1}}$ for incompleteness involves identifying the minimal (with respect to set inclusion) $a \subseteq A$ where A is the set of *abducibles* (in this case the vocabulary of post-conditions being used), given mined post-condition $e_{T_1}, \ldots, e_{T_i}$ which are known to be sound and complete such that at least one of the following hold: There exists an $e \in e_{T_1} \oplus \ldots \oplus e_{T_i} \oplus (e_{T_{i+1}} \wedge a)$ such that for every modified sequence database entry with the prefix $\langle T_1, \ldots, T_i, T_{i+1} \rangle$ and a corresponding suffix $e_1 \wedge \ldots \wedge e_n$, $e \cup KB \models e_1 \wedge \ldots \wedge e_n$. Similarly, repairing $e_{T_{i+1}}$ for unsoundness we need to identify $e'_{T_{i+1}} \subset e_{T_{i+1}}$ such that both of the following hold:

- There exists an $e \in e_{T_1} \oplus \ldots e_{T_i} \oplus e'_{T_{i+1}}$ for every modified sequence database entry with the prefix $\langle T_1, \ldots, T_i, T_{i+1} \rangle$ (with a corresponding suffix $e_1 \wedge \ldots \wedge e_n$) such that $e_1 \wedge \ldots \wedge e_n \wedge KB \models e$
- There exists no $e''_{T_{i+1}}$ where $e'_{T_{i+1}} \subset e''_{T_{i+1}} \subseteq e_{T_{i+1}}$ which satisfies the condition that there exists an $e \in e_1 \oplus e_{T_i} \oplus e''_{T_i}$ for every modified sequence database entry with the prefix $\langle T_1, \ldots T_i, T_{i+1} \rangle$ (with a corresponding suffix $e_1 \wedge \ldots \wedge e_n$) such that $e_1 \wedge \ldots \wedge e_n \wedge KB \models e$

The $e'_{T_{i+1}}$ identified via this analysis is set as the new post-condition of T_{i+1}.

The abductive repair framework complements the post-condition mining technique in important ways. When the application of the validation techniques presented in the previous section reveals that the mined post-conditions are unsound or incomplete, there are three choices in terms of next steps. One option is to repeat the post-condition mining exercise by setting different values for *minimum confidence* or *minimum support*. Another option is to acquire and mine from a larger dataset. Abductive repair offers a useful third alternative, and sometimes preferable to the first two options. The first two options represent an unfocused search for new post-conditions and may or may not lead to results that fix the unsoundness or incompleteness previously detected. Abductive repair can be effective simply because it can help identify what needs to be added (for incompleteness) or deleted (for unsoundness) in a focused manner.

A number of abductive reasoning techniques can be used to support automation of this analysis, but we leave this outside the scope of this paper (see [34] for a good survey of available techniques).

8. Evaluation

Evaluation with synthetic process models: Our aim is to establish that our approach generates reasonably reliable results. We ran the first set of experiments with a synthetic semantically annotated process model (i.e., a hand-crafted one with T_1, T_2, … etc, for activity names and p, q, … for states/post-conditions). The model had 8 activities, with an AND-split nested inside an XOR-split and with each activity semantically annotated with 1 or 2 literals (in the 2 literal case, the states were conjunctions of the 2 literals), and one rule in the $\mathcal{KB}$. We simulated a large number of possible execution traces of this model, and obtained process and state transition events. These events involved the execution of multiple concurrent process instances. There were multiple possible states associated with some of the tasks in the process design, owing to the fact that XOR gateway contributed to alternative flows that could have led to the same point (none of the states were generated by alternative means of resolving inconsistency in the state update operator). We then investigated the effect of scaling up the complexity of the process model, by generating a second synthetic process model with 12 activities with an XOR-split leading to two alternative flows, one of which included a nested AND-split and the other a nested XOR-split. The semantic annotations were 2 or 3 literals long and involved a mix of conjunctions and disjunctions. The background $\mathcal{KB}$ had 4 rules. There were multiple possible states associated with most of the activities and these were generated both by alternative flows that could lead to an activity (on account of XOR gateways) and by alternative resolutions of inconsistency by the state update operator.

Table 5 below describes the results of 4 experiments with each of these two process models. We used progressively larger numbers of overlapping instances of each process (i.e., T_i in instance 2 would start after the start of T_i in instance 1, but before the start of T_{i+1} in instance 1, and so on). We note that our problem would be no harder if the multiple concurrent process instances were of multiple distinct process models. We obtained progressively larger sizes of the sequence database. We recorded the precision (number of correct post-conditions mined over the total number of post-conditions mined) and recall (the number of correct post-conditions mined over the total number of actual post-conditions). Although not entirely monotonically improving, the results for process 2 confirm the intuition that better results are obtained with larger datasets. The results for process 2 also showed that the

Table 5
The recall and precision measures from the evaluation.

	Process model 1				Process model 2			
Number of instances	5	10	100	500	5	10	100	500
Size of sequence DB	48	100	1082	5352	66	133	1297	6512
Recall	1.0	1.0	1.0	1.0	0.953	1.0	0.981	0.989
Precision	1.0	1.0	1.0	1.0	1.0	0.988	1.0	1.0

Noise percentage	Precision
5	1.00
10	0.83
15	0.78
20	0.50

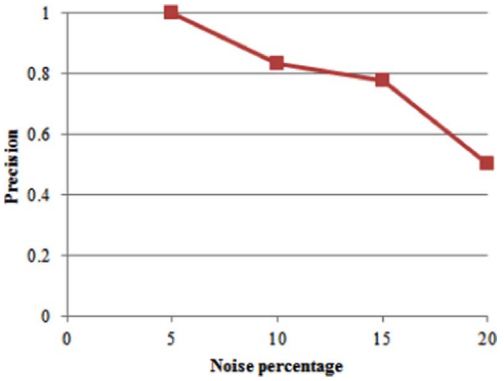

Fig. 2. Precision measures with noise in the effect log.

post-conditions mined tended to be incorrect for the last task in a process instance (in those settings where precision and recall values were less than 1). This was due to the sequence of states for the final activity not being bounded by the start of the next activity, but rather by the end of the table entries (artificially determined by length of the longest process).

The synthetic process and state transition events used in these examples considered all possible flows. Real-life data might involve more imperfections (such as certain XOR flows never being executed, certain activities never being executed and so on). We have also considered cases where noise is artificially added to the entries – as expected, precision and recall suffer as noise increases. We performed experiments with 500 instances of the second model. The proportion of noise in the complete effect log ranges from 5 to 20%. We plotted the performance of our technique (in terms of recall and precision) against this parameter. As expected, recall and precision decreases as the amount of noise increases. The results in Fig. 2 were consistently the same.

We took the mined post-conditions and used the validation technique from the previous section to repair the post-conditions (in the case of abductive repair we did not use any automated abductive framework, but used the abductive repair guidelines to perform manual repair). The result shown in Fig. 3 suggests that the approach is effective in identifying inaccurate post-conditions (and repairing them) leading to an increase in precision measures.

User-mediated evaluation: To evaluate our approach in a more real-life setting, we took a real-life semantically annotated process model that illustrates a *Holiday Booking* process followed by a travel agent in Fig. 4 (Table 6 shows some these post-condition annotations) and obtained a set of process and state transition events from an expert process modeler. We obtained a log of process events describing 10 execution instances (many of them with temporal overlaps) with a total of 110 entries, and a state transition events log with 154 entries. Excerpts of both tables are presented in Tables 7 and 8. In Table 8 we do not use real airline or hotel names.

We found that for 1 of the 11 activities in this process model, the post-conditions mined were incorrect, in the sense that the mined post-conditions did not correspond to the post-conditions provided by the expert process modeler. The specific activity with incorrectly mined post-conditions was *Check Flight Availability*. The duration of this activity overlapped in most instances with

Noise percentage	Precision (before)	Precision (after)
5	1.00	1.00
10	0.83	0.83
15	0.78	0.79
20	0.50	0.67

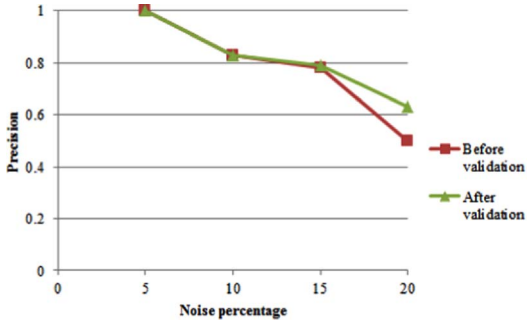

Fig. 3. Precision measures after validation with length parameter 2 and 3.

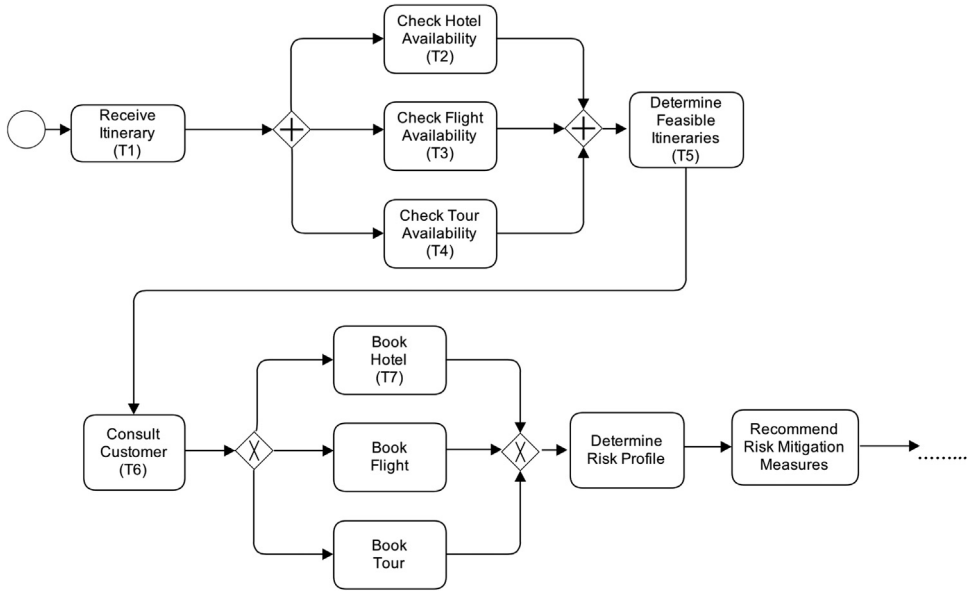

Fig. 4. A semantic annotated BPMN process model for *Holiday Booking* process.

longer duration activities from other instances of the same process (such as Receive Itinerary) in the event log. Consequently the mining procedure picked up the effects of some of these other overlapping activities and concluded that these were effects of *Check Flight Availability*. This problem would not occur with a large enough dataset, since it would be unlikely that the same set of overlaps would happen each time in a larger set of runs of the process.

It is important to note that our intent is not to necessarily mine post-conditions with perfect accuracy, but to assist analysts tasked with writing these post-conditions. Viewed in this light, the empirical results suggest that the approach can indeed be useful in practical settings (in that they generate near-accurate results that would require minimal editing on the part of analysts).

9. Related work

Artifact-centric business process modeling. An approach in the space of artifact-centric business process modeling is the GSM (Guard-Stage-Milestone) model by Hull et al. [4,24]. In the GSM model, the state of an artifact at any given point during the execution of the model is described using three elements: (a) milestone, which represents a business objective with achieving and/or invalidating conditions; (b) stage, which consists of a cluster of activities to achieve a milestone (in the atomic level, a stage consists of one activity); (c) guard, that controls whether a stage is active/open or not. Status change of a milestone and/or a stage is triggered by an incoming event in the form of a request or a task termination notification from the environment. Artifact-centric approaches such as GSM are of interest in our context mainly because of their focus on artifact lifecycles (in our vocabulary: object state transition events).

Semantic annotation of process models: A number of proposals in the literature consider semantic annotations of processes in a manner similar to ours, and would stand to benefit from implementations of our framework. A few examples of the

Table 6
Annotation of Holiday Booking Process with post-conditions.

Obtain Customer Requirements	*travel-dates (Cust, Dates)* ∧ *airline-preferences (Cust,Airline)* ∧ *airline-classoftravel (Cust, ClassOfTravel)* ∧ *departure-preferences (Cust,DepartTime)* ∧ *arrival-prefs (Cust, ArriveTime)* ∧ *meal-constraints (Cust, MealConstraints)* ∧ *freq-flyer (Cust, FreqFlyerDetails)* ∧ *hotel-pref (Cust, Hotel)* ∧ *room-prefs (Cust, RoomPrefs)* ∧ *tour-prefs (Cust, TourPrefs)*
Check Hotel Availability	*hotel-available (Hotel, Dates)*
Check Flight Availability	*flight-available (Flight, Airline, ClassOfTravel, DepartTimes, ArriveTimes)*
Check Tour Availability	*tour-available (Tour, DepartTimes, DepartLocation, Route, Stops)*
Determine Feasibility Itinerary	*feasible-itinerary (Flight, Hotel, Tour)*
Consult Customer	*customer-confirmation (Cust, Itinerary)*

Table 7
Excerpt from the process event log provided by the user.

Time	Customerid	Activity
46	cust4	Receive Itinerary
47	cust9	Receive Itinerary
53	cust3	Receive Itinerary
53	cust3	Check Flight Availability
72	cust1	Receive Itinerary
72	cust4	Check Flight Availability
77	cust1	Check Hotel Availability
78	cust3	Check Hotel Availability
81	cust4	Check Tour Availability
87	cust7	Receive Itinerary
93	cust5	Receive Itinerary
99	cust6	Receive Itinerary
106	cust9	Check Hotel Availability
116	cust1	Check Flight Availability
116	cust6	Check Tour Availability
125	cust3	Check Tour Availability
130	cust6	Check Hotel Availability

benefits that can be exploited from semantically annotated business process models including compliance checking, management level strategic alignment of business processes, and exception handling [10]. Di Francescomarino et al. [5] leverage the semantically labelled business processes to automatically verify if business processes fulfill a set of given constraints, and to formulate queries that involve both knowledge about the domain and the process structure. Ghose and Koliadis [11] provide a semantic characterization of a minimal revision strategy that is able to detect and partially automate compliance resolution using a notion of compliance patterns and obtain compliant process models from models that might be initially non-compliant. Happer and Stojanovic [14] propose a semantic business process management tool, Ontoprocess, to provide means for automatically checking the compliance of business processes with business rules by combining semantically described business processes with SWRL rules via a set of shared ontologies. Hoffmann et al. [22] propose a framework where processes are annotated to capture the semantics of task execution, and compliance is checked against a set of constraints posing restrictions on the desirable process states. Morrison et al. [32] propose a framework for strategic alignment to understand the relationship between a set of processes and the realization of a set of strategies and the optimal set of processes that can achieve these strategies using a semantically annotated process model.

A number of proposals add semantics to specify the dynamic behaviour of the business process, such as those by Weber et al. [41] and by Wong and Gibbons [43]. Semantic Business Process Validation (SBPV) [41] by Weber et al. is an approach that takes the annotations and the underlying ontology into account in order to determine whether the tasks are consistent with respect to each other, and with respect to the underlying workflow structure. Wong and Gibbons [43] propose a relative-timed semantic model for BPMN by introducing the notion of relative time in the form of delays to their model. The semantics is defined in the language of Communicating Sequential Processes (CSP). The annotated process model allows behavioural properties of BPMN diagrams to be mechanically verified. Koliadis et al. [27] propose an approach to analyse change against high-level models of the organization. Semantic EPC [39] by Thomas and Fellmann is a semantic extension of event-driven process chains.

A number of proposals seek to leverage semantics in assisting business analysts and process designers model business processes.

Table 8
Excerpt from the state transition event log provided by the user.

Time	Observed states
87	airline-preferences (cust4,Airline-23)
90	airline-classoftravel (cust4,ClassOfTravel-1)
114	departure-preferences(cust4,DepartTime-9)
117	arrival-prefs (cust4,ArriveTime-3)
120	meal-constraints(cust4,MealConstraints-47)
121	freq-flyer (cust4,FreqFlyer-53)
133	hotel-pref (cust4,Hotel-75)
137	room-prefs (cust4,RoomPref-95)
144	tour-prefs (cust4,TourPref-71)
155	hotel-available (Hotel-75,Dates-16)
173	flight-available (Flight-7,Airline-23,ClassOfTravel-1,DepartTime-9,ArriveTime-3)
191	tour-available (Tour-34,DepartTime-9,DepartLoc-35,Route-6 Stops-3)
203	feasible-itinerary (Flight-7,Hotel-75,Tour-71,CustPref-2)
220	customer-confirmation (cust4,Itinerary-30)
224	hotel-booking (cust4,Itinerary-30,Hotel-75,Dates-16)
233	flight-booking (cust4,Itinerary-30,Flight-7,Airline-23 ClassOfTravel-1,DepartTime-9,ArriveTime-3)
237	tour-booking (cust4,Itinerary-30,DepartTime-9,DepartLoc-35,Route-6,Stops-3)

Born and Dörr [2] extend the SAP Research modeling tool Maestro for BPMN. ProcessSEER [21] by Hinge et al. provides a user-friendly framework for analysts to explicitly annotate business process models and automatically computes the post-conditions associated with tasks selected by the user. Hornung et al. [23] propose a recommender system that suggests a list of correct and fitting process fragments for an edited business process model, which can be used to complete the process model being edited.

Mining process execution data:

A large body of work on process mining algorithms—such as the alpha algorithm [40], heuristic miner [42], and fuzzy miner [13]— offer the capability to extract the structure of the process model. Unlike this body of work, our focus is only on mining task post-conditions.

Our research integrates the two approaches of: (1) mining historical data to discover useful process information and (2) adding semantics to business process models, to obtain richer descriptions of business process designs which in turn can be used to support a variety of process analysis tasks such as compliance checking and resolution [11], goal satisfaction analysis [35] and so on.

10. Conclusions and future work

This paper offers an approach to mining business process task post-conditions from process and state changes events in process execution histories. Specifying post-conditions is notoriously difficult for process analysts, yet these post-conditions are critical to a variety of process analysis tasks such as process compliance management [11], goal satisfaction analysis [35], change management [25], enterprise process architectures [28] and the management of the business process life cycle [26]. The proposal involves the innovative use of sequential pattern mining on event logs. The proposal also leverages event data and the state update notion implicit in process execution to achieve a sophisticated validation technique, which in turn supports an abductive approach to the repair of the mined post-conditions. The empirical evaluation suggests that the results are generally reliable, pointing to prospects for further development of techniques that leverage these post-conditions in semantic analysis.

It is important to note that this approach relies on the availability of event logs, in line with existing work on process mining and analytics. This is not an unrealistic assumption since most enterprise systems generate event logs of some sort. An important limitation of this work is that his approach will not work well in settings where there is a significant delay between the execution of a task and the manifestation of its effects and there are no relevant tasks executed in-between that could be viewed as causing these effects. This could happen, for instance, in a clinical setting where the effects of administering a hormone-lowering drug are manifested some weeks later. We propose to address this in future work.

References

[1] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, in: Proc. of the ACM SIGMOD International Conference on Management of Data, Washington D.C., 1993, 207–216.

[2] M. Born, F. Dörr, I. Weber, User-friendly semantic annotation in business process modeling, in: M. Weske, M.S. Hacid, C. Godart, eds.: Web Information Systems Engineering, WISE 2007 Workshops, volume 4832 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, 260–271.

[3] K.C. Chan, W.H. Au,: An effective algorithm for mining interesting quantitative association rules, in: Proceedings of the 1997 ACM Symposium on Applied computing, ACM, 1997, 88–90.

[4] E. Damaggio, R. Hull, R. Vaculín, On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles, Inf. Syst. 38 (4) (2013) 561–584.

[5] C. Di Francescomarino, C. Ghidini, M. Rospocher, L. Serafini, P. Tonella, Reasoning on semantically annotated processes, in: A. Bouguettaya, I. Krueger, T. Margaria, eds.: Service-Oriented Computing ICSOC 2008. Volume 5364 of Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2008, pp. 132–146.

[6] D. Fensel, F. Facca, E. Simperl, Web service modeling ontology, in: Semantic Web Services, Springer, Berlin, Heidelberg, 2011, pp. 107–129.

[7] P. Fournier-Viger, U. Faghihi, R. Nkambou, E.M. Nguifo, CMRules: mining sequential rules common to several sequences, Knowl. Based Syst. 25 (1) (2012) 63–76.

[8] P. Fournier-Viger, R. Nkambou, V.S.M. Tseng, RuleGrowth: mining sequential rules common to several sequences by pattern-growth, in: Proceedings of the 2011 ACM Symposium on Applied Computing, ACM, 2011, 956–961.

[9] M.N. Garofalakis, R. Rastogi, K. Shim, SPIRIT: Sequential pattern mining with regular expression constraints, in: VLDB. Volume 99, 1999, 7–10.

[10] C. Ghidini, M. Rospocher, L. Serafini,: A formalisation of BPMN in description logics. FBK-irst, Tech. Rep. TR, 2008, 06–004.

[11] A. Ghose, G. Koliadis, Auditing business process compliance, Springer, 2007.

[12] M.L. Ginsberg, D.E. Smith, Reasoning about action I: a possible World approach, Artificial Intelligence 35 (2) (1988) 165–195.

[13] C. Gnther, W. van der Aalst,: Fuzzy mining adaptive process simplification based on multi-perspective metrics, in: G. Alonso, P. Dadam, M. Rosemann, eds.. Business Process Management. Volume 4714 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, 328–343.

[14] H.J. Happel, L. Stojanovic, Ontoprocess–a prototype for semantic business process verification using SWRL rules, in: Proceedings of the 3rd European Semantic Web Conference (ESWC), June 2006.

[15] S.K. Harms, J.S. Deogun, Sequential association rule mining with time lags, J. Intell. Inf. Syst. 22 (1) (2004) 7–22.

[16] N. Herzberg, M. Kunze, A. Rogge-Solti, Towards process evaluation in non-automated process execution environments, in: ZEUS, Citeseer, 2012, 97–103.

[17] N. Herzberg, A. Meyer, Improving process monitoring and progress prediction with data state transition events, Data Knowl. Eng. 98 (2015) 144–164.

[18] N. Herzberg, A. Meyer, M. Weske, An event processing platform for business process management, in: Proceedings of the 17th IEEE International Enterprise Distributed Object Computing Conference (EDOC), 2013, 107–116.

[19] N. Herzberg, A. Meyer, M. Weske, Improving business process intelligence by observing object state transitions, Proceedings of the 32th International Conference on Conceptual Modeling (ER 2013), 2013, 146–160.

[20] N. Herzberg, M. Weske, Enriching raw events to enable process intelligence: research challenges. Number 73. Universitätsverlag Potsdam, 2013.

[21] K. Hinge, A. Ghose, G. Koliadis, Process SEER: A tool for semantic effect annotation of business process models, in: Proceedings of the IEEE International Enterprise Distributed Object Computing Conference, 2009. EDOC'09, 2009, 54–63.

[22] J. Hoffmann, I. Weber, G. Governatori, On compliance checking for clausal constraints in annotated process models, Inf. Syst. Front. 14 (2) (2012) 155–177.

[23] T. Hornung, A. Koschmider, A. Oberweis, A recommender system for business process models, in: 17th Annual Workshop on Information Technologies & Systems (WITS), 2009.

[24] R. Hull, E. Damaggio, F. Fournier, M. Gupta, F.T. III Heath, S. Hobson, M. Linehan, S. Maradugu, A. Nigam, P. Sukaviriya, et al., Introducing the guard-stage-milestone approach for specifying business entity lifecycles, in: International Workshop on Web Services and Formal Methods, Springer Berlin Heidelberg, 2010, 1–24.

[25] G. Koliadis, A. Ghose,: Correlating business process and organizational models to manage change, in: Proceedings of the Australasian Conference on Information Systems, December 2006.
[26] G. Koliadis, A. Vranesevic, M. Bhuiyan, A. Krishna, A. Ghose, A combined approach for supporting the business process model lifecycle, in: Proceedings of the 10th Pacific Asia Conference on Information Systems (PACIS'06), 2006.
[27] G. Koliadis, A. Ghose, M. Bhuiyan,: Correlating business process and organizational models to manage change, in: Proceedings of the Australasian Conference on Information Systems. (2006) 1–10.
[28] G. Koliadis, A.K. Ghose, S. Padmanabhuni, Towards an enterprise business process architecture standard, in: 2008 IEEE Congress on Services-Part I, IEEE, 2008, 239–246.
[29] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, Naveen Srinivasan, K. Sycara, Bringing semantics to web services: The OWL-S approach, in: Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004, Volume 3387, 2005, 26–42.
[30] H. Meyer, On the semantics of service compositions, in: Web Reasoning and Rule Systems, Springer Berlin Heidelberg, 2007, 31–42.
[31] M. Montali, M. Pesic, W.M.P. van der Aalst, F. Chesani, P. Mello, S. Storari, Declarative specification and verification of service choreographiess, ACM Trans. Web 4 (2010) 1–62.
[32] E.D. Morrison, A.K. Ghose, H.K. Dam, K.G. Hinge, K. Hoesch-Klohe, Strategic alignment of business processes, in: Proceedings of the Service-Oriented Computing-ICSOC 2011 Workshops, Springer, 2012, 9–21.
[33] Office of Kids and Families, NSW Department of Health Australia: Infants and children: Acute management of Head Injury. 2 edn., 2011.
[34] G. Paul, Approaches to abductive reasoning: an overview, Artif.l intell. Rev. 7 (2) (1993) 109–152.
[35] K. Ponnalagu, A. Ghose, N.C. Narendra, H.K. Dam, Goal-aligned categorization of instance variants in knowledge-intensive processes, in: International Conference on Business Process Management, Springer, 2015 350–364.
[36] M. Santiputri, A.K. Ghose, H.K. Dam, X. Wen, Mining process task post-conditions, in: International Conference on Conceptual Modeling, Springer, 2015, 514–527.
[37] F. Smith, M. Missikoff, M. Proietti, Ontology-based querying of composite services, in: Business System Management and Engineering, Springer Berlin Heidelberg, 2012, 159–180.
[38] F. Smith, M. Proietti, Rule-based behavioral reasoning on semantic business processes, in: ICAART, SciTePress, 2013, 130–143.
[39] O. Thomas, M. Fellmann, Semantic EPC: enhancing process modeling using ontology languages, SBPM 251, 2007.
[40] W. van der Aalst, T. Weijters, L. Maruster, Workflow mining: discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (9) (2004) 1128–1142.
[41] I. Weber, J. Hoffmann, J. Mendling,: Semantic business process validation, in: Proceedings of the 3rd International Workshop on Semantic Business Process Management (SBPM08), CEUR-WS Proceedings, volume 472, 2008.
[42] A. Weijters, W.M. van Der Aalst, A.A. De Medeiros, Process mining with the heuristics miner-algorithm, Technische Universiteit Eindhoven, Tech. Rep. WP, 166, 2006, 1–34.
[43] P.Y. Wong, J. Gibbons, A relative timed semantics for BPMN, in: Proceedings of 7th International Workshop on the Foundations of Coordination Languages and Software Architectures, volume 229 of ENTCS, 2008.

Metta Santiputri received her bachelor degree in Informatics from Bandung Institute of Technology, Indonesia and the Master degree in Computer Science from University of Twente, the Netherlands. In 2001, she joined the Department of Informatics Engineering, State Polytechnic of Batam, as a Lecturer.

Currently, she is a Ph.D. candidate at Computer Science at the School of Computing and Information Technology, University of Wollongong, Australia. Her research interest, include business process modeling, data mining, semantic annotations, and goal-oriented requirements modeling.

Aditya Ghose is Professor of Computer Science at the School of Computing and IT at the University of Wollongong Australia, where he heads the Decision Systems Lab. He holds a Ph.D. and M.Sc. in Computing Science from the University of Alberta, Canada and a Bachelor of Computer Science and Engineering from Jadavpur University, India. His research interests are in knowledge representation and reasoning, business process management, service science, enterprise analytics and requirements engineering.

Hoa Khanh Dam is a Senior Lecturer in the School of Computing and Information Technology, University of Wollongong (UOW) in Australia. He is Associate Director for the Decision System Lab at UOW, heading its Software Engineering Analytics research program. His research interests lie primarily in the intersection of software engineering, business process management and service-oriented computing, focusing on such areas as software engineering analytics, process analytics and service analytics. He holds Ph.D. and Master degrees in Computer Science from RMIT University, and Bachelor of Computer Science degree from the University of Melbourne in Australia. His research has won multiple Best Paper Awards (at WICSA, APCCM, and ASWEC) and ACM SIGSOFT Distinguished Paper Award (at MSR).