Natural Language-based Detection of Semantic Execution Anomalies in Event Logs

Han van der Aa[a], Adrian Rebmann[a], Henrik Leopold[b,c]

[a] *Data and Web Science Group, University of Mannheim Mannheim, Germany*
[b] *Kühne Logistics University, Hamburg, Germany*
[c] *Hasso Plattner Institute, University of Potsdam, Potsdam, Germany*

Abstract

Anomaly detection in process mining aims to recognize outlying or unexpected behavior in event logs for purposes such as the removal of noise and identification of conformance violations. Existing techniques for this task are primarily frequency-based, arguing that behavior is anomalous because it is uncommon. However, such techniques ignore the semantics of recorded events and, therefore, do not take the meaning of potential anomalies into consideration. In this work, we overcome this caveat and focus on the detection of anomalies from a semantic perspective, arguing that anomalies can be recognized when process behavior *does not make sense*. To achieve this, we propose an approach that exploits the natural language associated with events. Our key idea is to detect anomalous process behavior by identifying semantically inconsistent execution patterns. To detect such patterns, we first automatically extract business objects and actions from the textual labels of events. We then compare these against a process-independent knowledge base. By populating this knowledge base with patterns from various kinds of resources, our approach can be used in a range of contexts and domains. We demonstrate the capability of our approach to successfully detect semantic execution anomalies through an evaluation based on a set of real-world and synthetic event logs and show the complementary nature of semantics-based anomaly detection to existing frequency-based techniques.

Keywords: Process mining, Natural language processing, Anomaly detection

1. Introduction

In many domains it is important that the execution of business processes adheres to certain rules and regulations. In a hospital, for instance, the treatment of patients with drugs such as opioids is subject to a number of strict conditions and checks. Similarly, a bank clerk must not provide a loan to a customer

Email addresses: han@informatik.uni-mannheim.de (Han van der Aa), rebmann@informatik.uni-mannheim.de (Adrian Rebmann), henrik.leopold@the-klu.org (Henrik Leopold)

without conducting a number of solvency checks. Violating such rules and regulations may have severe implications ranging from productivity loss [1] to financial penalties imposed by authorities [2].

To efficiently detect such undesirable behavior, *conformance checking* techniques have been introduced [3, 4]. They compare the actual behavior of employees, as recorded by information systems, to the desired behavior that is specified in a normative process model. In this way, non-conforming behavior can be automatically detected and respective measures for their prevention can be introduced. While these techniques have been found to be valuable in many settings, they are only applicable if a process model capturing the normative process behavior is available.

Recognizing that this is often not the case in practice, several authors have developed so-called *anomaly detection* techniques. Early work in this area applies traditional conformance checking algorithms against a process model that was automatically discovered from the considered event log [5]. More recent contributions leverage unsupervised machine learning techniques to detect anomalies in event logs without having any prior knowledge [6, 7]. A key assumption of all these techniques, however, is that anomalous behavior is significantly less frequent than desired behavior.

In this paper, we argue that this frequency-based perspective is limited, since it ignores the semantics of the recorded events and, therefore, does not take the meaning of potentially anomalous patterns into account. To bridge this gap, we put forward the idea of exploiting a, so far, disregarded dimension of event logs for anomaly detection: the natural language from event labels. More specifically, we build on the linguistic branch of *semantics*, which is concerned with the meaning of words. Our key idea is that anomalous process behavior can be identified based on the detection of semantically inconsistent execution patterns. As an example, consider a process instance in which an *order* is both *accepted* and *rejected*. From a semantic point of view, this is an undesirable constellation since *accepted* and *rejected* are opposites (so-called *antonyms*). Therefore, we can conclude that the execution of this particular process instance is semantically inconsistent and, thus, anomalous.

To detect semantic anomalies, we propose an approach for the *natural language-driven detection of semantic execution anomalies*. First, building on state-of-the-art natural language processing (NLP) techniques, our approach extracts semantic information, in terms of the business objects and actions, from the textual labels of events in a log. We then compare the extracted action and business object patterns against a process-independent knowledge base, which contains semantic assertions about the manner in which processes are expected to be executed. We populate this knowledge base with patterns stemming from linguistic, as well as from process-oriented resources. Since these patterns are intended to be generic, our semantic execution anomaly detection approach can be used in various contexts and domains. To demonstrate the capability of our approach to successfully detect semantic execution anomalies, we present an evaluation based on a set of real-world and synthetic event logs and compare our results against frequency-based techniques.

In the remainder, Section 2 illustrates the potential of detecting semantic anomalies based on natural

Trace 1	
A	Create order
C	Approve order
B	Check order
E	Create delivery
F	Complete delivery

Trace 2	
A	Create order
B	Check order
C	Approve order
D	Reject order
F	Complete delivery

Figure 1: Anomalous traces in an order handling process

language labels of events and Section 3 provides key definitions. Section 4 presents our approach for natural language-driven semantic anomaly detection. Section 5 discusses the evaluation of our approach. Section 6 reflects on related work before Section 7 concludes the paper.

2. Motivation

To illustrate the potential of natural language-driven anomaly detection, consider a simplified order handling process. Regular execution of this process starts by creating an order (event A). Orders are subsequently checked (B) before either being approved (C) or rejected (D). The latter case ends the process instance. For approved orders, a delivery is created (E) and afterwards completed (F).

Now consider the two recorded execution traces depicted in Figure 1. Even without being aware of the expected execution of this process, several execution anomalies can be detected by considering the labels of the recorded events:

- *Out-of-order execution*: In Trace 1, an order was approved (C) before it has actually been checked (B), clearly revealing an anomalous execution order.
- *Superfluous event*: In Trace 2, we observe that an order was both approved (C) and rejected (D). Naturally, the co-occurrence of these events indicates an anomaly since they are expected to exclude each other.
- *Missing event*: Finally, we observe that in Trace 2 a delivery was completed (F) that has never been created (E). Since Trace 1 shows that this creation event does exist in the log, one would expect that it occurs for every delivery, revealing a missed event.

These examples clearly illustrate the potential of leveraging event labels for the detection of semantic execution anomalies. While the detection of these particular cases is straightforward for humans, the complexity and size of real-world event logs call for automated support. However, this requires an approach that has an understanding about the order in which events should occur, when events should be exclusive, and when they should co-occur. In the next section, we tackle this challenge through an approach that combines the semantic parsing of event labels with the establishment of a knowledge base that captures assertions about

the semantics of appropriate process executions.

3. Preliminaries

Events and event logs. We define $\mathcal{E}$ to denote the universe of all events. The events recorded for a single execution (i.e., an instance) of a process is called a *trace*, which is modeled as a sequence of events, i.e., we denote a trace with n events as $\sigma = \langle e_1, \ldots, e_n \rangle$, where each event $e \in \sigma$ is part of $\mathcal{E}$. Given the focus of our work on the detection of semantic anomaly detection, we explicitly denote an *event label* as a natural language attribute describing the activity that an event corresponds to, defined as $e.l$.

Process models and model collections. We adopt a simple, generic definition of process models and collections. Using $\mathcal{M}$ to denote a process model collection, each process model $M \in \mathcal{M}$ is defined as a set of execution sequences, $M \subseteq T_M^*$ over the set of tasks in the model T_M. Note that we here use the term *task* to refer to any labeled step in a process, which may be an activity or event in a BPMN model, or a transition in a Petri net.

Given a sequence (whether consisting of events or tasks) $\sigma = \langle t_1, \ldots, t_n \rangle$, we will use $t_i < t_j$ as a short-hand to denote that t_i occurs before t_j in the sequence, i.e., that $i < j$.

4. Semantic Anomaly Detection

We propose an approach for the label-based detection of semantic execution anomalies that consists of three main components, as depicted in Figure 2.

1. The *event label parsing* component extracts semantic information from the labels associated with events in a log. In particular, label parsing aims to extract the *action* and the *business object* from event labels, such as illustrated for a "*reject order*" event in the figure.
2. The *knowledge base population* component collects assertions about the interrelations that should hold among actions applied to business objects, e.g., business objects cannot be *archived* without first being *created*. Note that a knowledge base is intended to be process-independent, meaning that users do not have to populate their own knowledge base but can employ an existing knowledge base, such as the one used to conduct our evaluation.
3. Finally, the *anomaly detection* component compares the semantic information resulting from label parsing to the assertions contained in the established knowledge base in order to detect semantically anomalous behavior. In particular, our approach currently detects anomalies in terms of (1) out-of-order executions, (2) superfluous events, and (3) missing events.

In the remainder, Sections 4.1 to 4.3 shall describe the three components in detail.

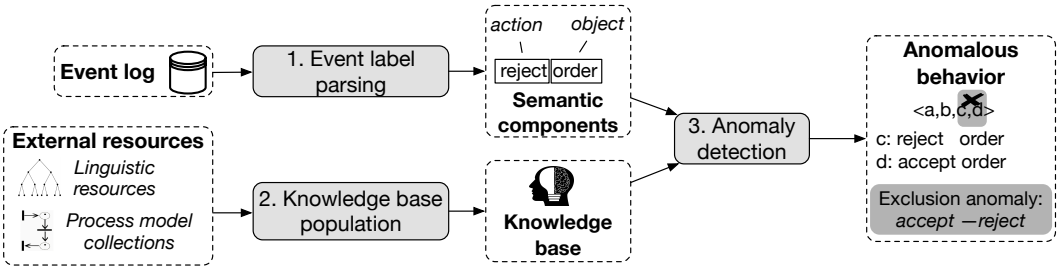

Figure 2: Overview of the proposed anomaly detection approach

4.1. Event Label Parsing

Our approach detects anomalous behavior in terms of undesired patterns of actions that occur for a particular business object. In line with established work on semantic process analysis [8], we use the term business object to broadly refer to the entity to which an event corresponds, such as a *purchase order*, an *applicant*, or a *request*. The action, then, captures the state change that is applied to this business object, e.g, *rejecting* (action) a *purchase order* (business object). Since information on these components is hardly ever explicitly represented in event logs, e.g., as dedicated event attributes, the first step of our approach sets out to extract these components from the natural language labels associated with events.

Challenges. This extraction step comes with considerable challenges due to the large heterogeneity observed in event labels, as illustrated through the real-world examples depicted in Table 1. Labels l_1 and l_2 represent rather standard cases in which a label consists of a single action and a business object. However, even for such standard cases, we observe a difference in order (e.g., action followed by business object versus the reverse). Furthermore, l_2 highlights that business objects can correspond to compound nouns like "*quality indicator*". Although l_3 also follows a common structure, it is important to recognize that, especially, business objects, are not always denoted using proper natural language, such as "*gfm17*" in this case. Furthermore, it is interesting to note here that the label's action, "*correction*", is a noun rather than a verb in this case. Finally, examples l_4 through l_6 depict cases that go beyond the typical combination of a business object and corresponding action. Labels l_4 and l_5 represent cases that, respectively, lack a business object and an action, whereas l_6 illustrates a case where multiple actions are described by a single event label.

ID	Source	Event label	Action(s)	Business object
l_1	[9]	close case	close	case
l_2	[10]	quality indicator fixed	fix	quality indicator
l_3	[11]	correction gfm17	correct	gfm17
l_4	[12]	a_approved	approve	–
l_5	[9]	VVGB positive	–	VVGB
l_6	[13]	draft and send request for advice	draft, send	request

Table 1: Exemplary event labels from real-world event logs

Tagging technique. To deal with these challenges, we adapt a tagging technique from our earlier work [14], which we specifically designed to extract semantic information from real-world event data. The tagging technique builds on BERT [15], a state-of-the-art language model developed to be applicable to a wide range of NLP tasks. The BERT model is pre-trained on huge text corpora in order to develop a general understanding of a language. This model can then be fine-tuned by training it on an additional, smaller training data collection to target a particular task. In this manner, the trained model combines its general language understanding with aspects that are specific to task at hand.

To fine-tune BERT for this tagging task, we manually annotated a collection of labels stemming from process models and event logs with semantic information. As opposed to our original technique [14], we have adapted the tagging and training mechanisms to only consider the tags that are relevant to our anomaly detection approach. Namely, we use the `ACT` and `BO` tags to, respectively, denote words that correspond to an action or business object (or part thereof). Any other word in a label receives an `X` tag.

Algorithm 1 Event label parsing

input event e
output a set of actions and associated business objects
$W \leftarrow$ `tokenize`$(e.l)$ ▷ Split the label into a sequence of words
$\pi \backslash T \leftarrow$ `tag(W)` ▷ Turn the tokens into a tagged sequence of chunks
result $\leftarrow \emptyset$
for chunk, tag $\in \pi \backslash T$ **do** ▷ Iterate over all chunks
    if tag = `ACT` **then**
        action $\leftarrow$ chunk
        object $\leftarrow$ `getAssociatedBO(chunk)` ▷ Get the associated `BO` chunk, if any
        res.`add`(action, object)
return result

Parsing algorithm. We employ the tagger as part of a function that extracts combinations of actions and business objects from an event's label, as denoted by Algorithm 1. Formally, we define this function as: $parse : \mathcal{E} \rightarrow \mathcal{A} \times \mathcal{O}$, where $\mathcal{E}$, $\mathcal{A}$, and $\mathcal{O}$ respectively denote the universes of all events, actions, and business objects.

The algorithm starts by tokenizing an event label (line 3), which splits the label into individual words (i.e., tokens). Since our approach needs to deal with information extracted from information systems, rather than with regular natural language text, this tokenization step needs to deal with peculiarities like camel-case notation or other notation styles, such as seen for label l_4, *a_ approved*. After tokenization, we employ the BERT-based tagger to turn the word sequence W into a sequence of tagged chunks $\pi \backslash T$, with $\pi = \langle \varphi_1, \ldots, \varphi_n \rangle$ and $T = \langle t_1, \ldots, t_n \rangle$. Each chunk φ_i for $i \in [1, n]$ consists of one or more words from W with t_i as its corresponding tag, e.g.,:

$\pi \backslash T = \langle$ `draft\ACT`, `and\X`, `send\ACT`, `request\BO`, `for advice\X` $\rangle$

Afterwards, the algorithm iterates over the chunks (line 6) and tries to associate each `ACT` chunk with a `BO` chunk (line 9). For this procedure of function `getAssociatedBO`, we employ the following heuristic. We first infer the direction in which actions appear in relation to the associated business objects in the label. If the first non-`X` tag of T is `ACT`, we assume that actions appear *before* their business objects, e.g., $\langle$`close\ACT, case\BO`$\rangle$, whereas if the first tag is `BO`, we assume the reverse order, e.g., $\langle$`case\BO, closed\ACT`$\rangle$. Then, the function determines the associated business object of an activity as the chunk with a `BO` tag that appears *closest* (given the identified direction) to the `ACT` chunk. This procedure can result in cases where multiple actions are associated with the same business object, e.g., for label l_6 both actions, *draft* and *send*, will be associated with the business object *request*. Furthermore, if there is no *closest* `BO` chunk, no business object will be associated with an action, e.g., for l_4, the action *approve* will have an empty business object $\perp$.

Output. The function $parse(e)$ returns a set of action-business object pairs obtained from the label of event e, where each pair is given as a tuple (act, bo) with $act \in \mathcal{A}$ and $bo \in \mathcal{O} \cup \{\perp\}$. Although this set may contain multiple pairs, e.g., as seen for label l_6, the remainder of this section shall consider the case in which the parsing of an event label results in a single action, denoted by $act(e) \in \mathcal{A}$ and a single (possibly empty) business object $obj(e) \in \mathcal{O} \cup \{\perp\}$. This simplification serves to greatly improve the readability and intuitiveness of the definitions that we provide in the following sections, while it does not lead to a loss of generality since our conceptual contributions can be straightforwardly lifted to the situation with multiple actions and business objects per event.

4.2. Knowledge Base Population

Our approach detects semantic anomalies based on a *knowledge base* that captures assertions about the interrelations that should hold among actions in a process. These assertions are stored in the form of

knowledge records, where each record specifies a relation that should hold between two actions. Formally, we define this as follows:

Definition 1 (Knowledge base, knowledge record). *A* knowledge base $\mathcal{K}$ *comprises a set of* knowledge records*, where each* $r \in \mathcal{K}$ *is a tuple,* $r = (a_1, rel, a_2, supp)$*, with* $a_1, a_2 \in \mathcal{A}$ *as its actions,* $rel \in \{\#, \prec, \Rightarrow\}$ *its* behavioral relation*, and* $supp \in \mathbb{N}$ *its* support value*.*

Behavioral relations. The *relation rel* of a knowledge record can be used to represent any kind of behavioral relation that should hold between two actions. In this work, we capture three types of behavioral relations, i.e., $\#$, $\prec$, and $\Rightarrow$:

- $(a_1, \prec, a_2)$: denotes an (indirect) *order relation*, i.e., a_2 should not occur before a_1;
- $(a_1, \#, a_2)$: denotes an *exclusion relation*, i.e., a_1 and a_2 should not co-occur;
- $(a_1, \Rightarrow, a_2)$: denotes a *co-occurrence relation*, i.e., if a_1 occurs a_2 should occur as well.

With these types, we cover the main kinds of behavioral assertions that one can make between pairs of actions, i.e., whether actions should occur in a particular order, should exclude each other, or should occur together. As such, these three types suffice to allow us to discover anomalies in terms of out-of-order execution, superfluous events, and missing events in a manner that makes sense from a semantic perspective. However, we stress that our contributions are independent of these particular types and that our work can be extended with additional relation types and associated anomalies if desired.

Support. The *support* value *supp* reflects the level of confidence that is placed in a particular knowledge record, where a higher *supp* indicates that there is stronger support for the captured assertion.

Resources for population. Since our goal is to detect *semantic* anomalies in *processes*, we consider two main types of resources for the population of a knowledge base $\mathcal{K}$: (1) linguistic resources that provide assertions based on the semantics of natural language and (2) process-oriented resources that provide assertions based on recurring behavioral patterns.

4.2.1. Linguistic-based Knowledge Records

We first propose to exploit semantic relations that are specified by linguistic resources to populate a knowledge base. These resources capture relations between words, which we propose can be lifted to a process context for anomaly detection.

A broad spectrum of resources exist that specify semantic relations between words and entities. Well-known linguistic resources include the lexical databases *WordNet* [16] and *VerbNet* [17]. Similarly, knowledge graphs (or ontologies) such as *DBpedia* [18], which encode structured information of entities and their inter-relations [19] may also be used for such purposes. For brevity and clarity, though, we here focus on a single linguistic resource that is particularly well-suited for the purposes of semantic anomaly detection in business processes: VerbOcean.

VerbOcean [20] is a broad-coverage resource that captures various semantic relations over a total of 29,165 pairs of verbs. A resource specifically targeting verbs is particularly useful for our purposes, since verbs are the primary vehicle to describe events and actions [17, 21]. In this regard, VerbOcean is especially promising because it defines several relations that are closely related to our purposes: `happens-before`, `enablement`, and `antonymy`.

Happens-before. The `happens-before` relation indicates that two verbs refer to two temporally disjoint intervals or instances, such as *marry* happens before *divorce*. Naturally, this relation closely corresponds to ordering relations that are expected to hold among actions in a business process, meaning that the linguistic relation can be employed to detect order anomalies in event logs. Therefore, we turn all 4,205 verb-pairs in a `happens-before` relation in VerbOcean into knowledge records capturing order relations ($\prec$) in our knowledge base $\mathcal{K}$. In this manner, we are able to identify process-related assertions such as:

- (consider, $\prec$, decide), i.e., *consider* occurs before *decide*;
- (schedule, $\prec$, reschedule), i.e., *schedule* occurs before *reschedule*;
- (bill, $\prec$, reimburse), i.e., *bill* occurs before *reimburse*.

Enablement. The `enablement` relation captures a type of causal relation according to the specification by Barker and Szpakowicz [22]. VerbOcean contains a total of 393 verb pairs in this relation, which can be highly useful to detect omissions of mandatory event executions in business processes. Therefore, we turn these 393 pairs into knowledge records capturing co-occurrence relations ($\Rightarrow$) in our knowledge base. These records include:

- (make, $\Rightarrow$, sell), i.e.,*make* can result in *sell*;
- (compare, $\Rightarrow$, describe), i.e.,*compare* can result in *describe*;
- (bill, $\Rightarrow$, reimburse), i.e.,*bill* can result in *reimburse*.

We recognize that there is an inherent overlap between the `happens-before` and `enablement` relations, since enabling something often also implies that there exists a certain order. For instance, as shown above, the verbs *bill* and *reimburse* are subject to both types of relations. Such overlaps are also transferred to our knowledge base, which allows the knowledge base to capture both an order and a co-occurrence relation between the two actions. However, not all `enablement` relations necessarily indicate an order. For instance, VerbOcean contains bidirectional enablement relations for the verbs *make* and *sell*, which reflects that making something can result in a sale (make-to-stock), whereas a sale can also lead to making (make-to-order).

Antonymy. The `antonymy` relation captures *semantic opposition* between verbs. Words can be opposites of each other in various manners, which means that the antonymy relation actually covers several subtypes [23]. Crucial for our purposes is the distinction between antonyms that indicate opposition in terms of alternatives, e.g., *accept* versus *reject*, and antonyms following from a so-called restitutive opposition [24], e.g., *damage* versus *repair*.

In a process context, the former subtype clearly corresponds to the exclusion relation we want to capture in a knowledge base, e.g., we want to include a record $r = (accept, \#, reject)$ to capture that a business object cannot be both *accepted* and *rejected*. However, antonyms of the latter subtype, i.e., which are in restitutive opposition, should not lead to exclusion relations in a knowledge base: actions with the opposite effect can naturally co-occur in a process instance, e.g., when something is *started* it can subsequently be *finished* and something that was *opened* can later be *closed*. As recognized [20], these cases can be distinguished by also considering the other relations that hold for a verb pair. In particular, verbs in restitutive opposition are both in `antonymy` as well as in a `happens-before` relation. Since our goal is to include the other kinds of antonyms, we establish (bidirectional) knowledge records in our knowledge base for each verb pair that is in an `antonymy` but not in a `happens-before` relation. This procedure results in the creation of 1,587 knowledge records (out of the 1,973 antonymy relations in VerbOcean), including:

- (accept, #, reject), i.e., *accept* excludes *reject*;
- (acquire, $\prec$, lose), i.e., *acquire* excludes *lose*;
- (penalize, $\prec$, reward), i.e., *penalize* excludes *reward*.

Overall, population using VerbOcean relation thus yields a total of 6,185 records, capturing 4,205 order, 393 co-occurrence, and 1,587 exclusion relations. Given the trustworthiness of these records, we propose to assign these records a comparably high support value, e.g., 100.

4.2.2. Process-oriented Resources

Second, we propose to employ process-oriented resources alongside linguistic ones for semantic anomaly detection since they allow for the identification of recurring action patterns in business processes. In particular, we propose to populate a knowledge base on the basis of a large repositories of process models, i.e., *process model collections*. These collections, such as the publicly available Academic Initiative [25] and the SAP Reference Model collection, capture information on how large numbers of processes, possibly covering various domains, are expected to be executed. In combination with the formal execution logic they capture, this makes these repositories prime candidates for the identification of recurring action patterns in processes [26].

Population procedure. To extract knowledge records from a process model collection $\mathcal{M}$, we aim to identify pairs of actions that commonly co-occur, exclude each other, or appear in a particular order. The procedure for this is described by Algorithm 2 and consists of two main parts:

The first part of the algorithm (lines 3–11) iterates over the models in a process model collection $\mathcal{M}$. For each model $M \in \mathcal{M}$, the algorithm aims to identify pairs of actions that are in one of the three behavioral relations we consider. To do this, the algorithm first determines if a pair of tasks correspond to the same business object (line 5), since we are only interested in actions that adhere to this requirement. Note that we here employ the same label parsing technique and notation as we use when dealing with event labels (see

Section 4.1). For actions that relate to the same business object, the algorithm then determines if the task pair adheres to a specific behavioral relation and, thus, can be used as support for a knowledge record. In particular, the algorithm checks if:

1. t_1 and t_2 never co-occur (line 6), supporting an exclusion relation $t_1 \# t_2$,
2. t_1 occurs before t_2, but t_2 never before t_1 (line 8), supporting an order relation $t_1 \prec t_2$, or
3. if t_1 never occurs without t_2 (line 10), supporting a co-occurrence relation $t_1 \Rightarrow t_2$.

Note that when determining these behavioral relations, we only consider execution traces that do not contain repeated activities, i.e., we exclude loops from consideration. Then, when support for a given relation type rel is found, the algorithm extracts the actions of t_1 and t_2 and adds the respective relation $(act(t_1), rel, act(t_2))$ to the *relations* multi-set.

Algorithm 2 Knowledge record extraction from a process model collection

1: **input** A process model collection $\mathcal{M}$, minimal support parameter k
2: $relations \leftarrow \emptyset$ ▷ Defines a bag (i.e., a multi-set) of action relations
3: **for** $M \in \mathcal{M}$ **do**
4: **for** $t_1, t_2 \in T_M$ **do**
5: **if** $t_1 \neq t_2 \wedge obj(t_1) = obj(t_2)$ **then** ▷ Check for equal business objects
6: **if** $\nexists\ \sigma \in M : t_1 \in \sigma \wedge t_2 \in \sigma$ **then** ▷ t_1 and t_2 do not co-occur
7: $relations$.`add`$(\ (act(t_1), \#, act(t_2))\ )$
8: **if** $\exists\ \sigma \in M : t_1 \in \sigma \wedge t_2 \in \sigma \wedge t_1 < t_2 \wedge$ ▷ t_1 can occur before t_2
$\nexists\ \sigma' \in M : t_1 \in \sigma' \wedge t_2 \in \sigma' \wedge t_2 < t_1$ **then** ▷ t_2 never occurs before t_1
9: $relations$.`add`$(\ (act(t_1), \prec, act(t_2))\ )$
10: **if** $\forall \sigma \in M : t_1 \in \sigma \wedge t_2 \in \sigma \vee t_1 \notin \sigma$ **then** ▷ t_1 only occurs with t_2
11: $relations$.`add`$(\ (act(t_1), \Rightarrow, act(t_2))\ )$
12: $records \leftarrow \emptyset$ ▷ Create result set
13: **for** $(a_1, rel, a_2) \in relations$ **do**
14: $c \leftarrow$ `count`$(a_1, rel, a_2, relations)$ ▷ Get relation's multiplicity
15: **if** $c \geq k$ **then** ▷ Check if relation is frequent enough
16: $records$.`add`$(\ (a_1, rel, a_2, c)\ )$
17: **return** $records$ ▷ Returns the records to be added to a knowledge base

In its second part (lines 12–17), the algorithm turns the combinations of actions in a particular behavioral relation that have been observed a sufficient number of times into knowledge records. Note that in line 14 we use `count`$(a_1, rel, a_2, relations)$ to denote the cardinality of the tuple (a_1, rel, a_2) in the *relations* multi-set. Here, we employ parameter k to capture the minimum number of times that a pattern must have been observed in order for the corresponding record to be included in the knowledge base, i.e., the minimal support value of knowledge records.

4.2.3. Conflict resolution

When knowledge records are extracted from different resources, this may lead to conflicting statements about expected process behavior contained in the knowledge base. For instance, certain process models may specify that a business object can be either *opened* or *closed*, indicating a relation *open#close*, whereas other models (as well as VerbOcean), reveal that these actions also often occur in a sequential manner (i.e., a case is first opened and afterwards closed). In this final step of knowledge base population, we filter out such conflicting records in order to improve the quality of the anomalies we can detect. For this, we employ two heuristics:

Filtering order relations. As described above, it may be the case that certain models indicate a particular order $act_1 \prec act_2$ between two actions, whereas other sources indicate that the opposite order should hold, resulting in a situation where some record $r_1 = (act_1, \prec, act_2, supp_1)$ indicates one order and another record indicates the reverse, $r_2 = (act_2, \prec, act_1, supp_1)$. Since including both records in a knowledge base could lead to conflicts, we filter out the knowledge record with the lowest support. E.g., if $supp_1 < supp_2$, we remove r_1 and retain r_2. Should the support for both be equal, we omit both records, given that this indicates that there is no particular order in which a_1 and a_2 should occur. Note that this notion of equal support could be replaced with a threshold, where both records would be removed if their support values are comparable, e.g., given a threshold $\tau \in [0, 1]$, we would omit both r_1 and r_2 if $(1 - \tau) * supp_1 \leq supp_2 \leq (1 + \tau) * supp_1$.

Filtering exclusion relations. We also omit *exclusion* records when there are more commonly occurring order or co-occurrence records included in the knowledge base that involve the same actions. Here, the intuition is that, when it is commonly observed that actions act_1 and act_2 should occur in a particular order (or co-occur), it is highly unlikely that these actions are supposed to exclude each other. As such, given $r_1 = (act_1, \#, act_2, supp_1)$ and $r_2 = (act_1, \prec, act_2, supp_2)$, we omit r_1 if $supp_1 < supp_2$.

After applying this filtering step to resolve conflicts, the knowledge base can be used for anomaly detection, as described next.

4.3. Semantic Anomaly Detection

Our approach uses the populated knowledge base to detect three kinds of anomalies: 1) order violations, i.e., out-of-order executions, 2) exclusion violations, i.e., superfluous events, and 3) co-occurrence violations, i.e., missing events.

Using the actions and business objects resulting from the extraction functions *act* and *obj* (cf. Section 4.1), we detect anomalies by first matching the parsed events in a trace to knowledge records and subsequently verifying their adherence to them.

4.3.1. Knowledge Record Matching

To determine whether a knowledge record is relevant to recorded events, we consider two events, e_i and e_j, which correspond to the same business object, i.e., $obj(e_i) = obj(e_j)$. We recognize that a knowledge record $r \in \mathcal{K}$ is relevant to these events when their actions, $act(e_i)$ and $act(e_j)$ correspond to the actions of r. To determine this correspondence, we consider cases where actions are fully equal, but also when they have a similar meaning. For this, we define a predicate $\texttt{equiv} : \mathcal{A} \times \mathcal{A} \rightarrow [true, false]$, which returns true when two actions are considered to be equivalent to each other. We instantiate this predicate in three manners:

1. **Equality:** This instantiation only considers actions to be equivalent when they have the same lemma, i.e., $\texttt{equiv}_{eq}(a_1, a_2) := \text{lemma}(a_1) = \text{lemma}(a_2)$. Here, the function *lemma* returns the *canonical form* of an action, e.g., $\text{lemma}(rejected) = reject$ and $\text{lemma}(ran) = run$. Lemmas can be straightforwardly computed using general-purpose NLP tools.
2. **Synonymous:** This instantiation considers two actions to be equivalent when their lemmas are synonyms of each other, i.e., $\texttt{equiv}_{syn}(a_1, a_2) := \text{lemma}(a_1) \in \text{syn}(a_2) \wedge \text{lemma}(a_2) \in \text{syn}(a_1)$. Here, we use $\text{syn}(a)$ to denote the set of synonyms of an action a, as, for instance, determined through the use of WordNet [16]. This predicate instantiation considerably increases the applicability of knowledge records to events since it adds a degree of generalization to them. For instance, given the record $r = (accept, \#, reject)$, synonymous equivalence recognizes that this also means that *approve order* and *refuse order* events should be exclusive as well.
3. **Semantic similarity:** Finally, we also consider equivalence based on semantic similarity measures. These measures quantify the similarity between words according to their meaning. While various measures exist, they can be largely grouped into two categories [27]: *WordNet-based* or *taxonomy-based* measures, cf. [28], consider similarity from the perspective of semantic relations, e.g., synonyms and hypernyms, which hold among terms. Measures based on *distributional similarity*, cf. [29, 30], instead, are based on the statistical analysis of co-occurrences of words in larger text corpora.

   Using $sim_{semantic}(a_1, a_2) \in [0, 1]$ to refer to a quantification of semantic similarity, we formalize equivalence based on it as $\texttt{equiv}_{sim}(a_1, a_2) := sim_{semantic}(a_1, a_2) \geq \tau_{sim}$. The difference between equivalence based on semantic similarity and the aforementioned synonymous equivalence is that the former is more flexible. Whereas two terms either are synonymous or not (according to a certain resource), semantic similarity reflects a scale in the range $[0, 1]$. As such, we can adjust threshold τ_{sim} in the definition of $\texttt{equiv}_{sim}$ to adjust the degree of similarity that is required for two actions to be considered equivalent.

Given any instantiation of `equiv` we then formally denote the applicability of a knowledge record to a pair of events as: $\texttt{match}(e_1, e_2, r) := \texttt{equiv}(act(e_1), r.a_1) \wedge \texttt{equiv}(act(e_2), r.a_2)$. Based on this `match` predicate, we then detect semantic violations as discussed next.

4.3.2. Detecting Violations

We detect semantic anomalies by determining if recorded process behavior violates the assertions captured in a knowledge base's records. As such, we recognize *order*, *exclusion*, and *co-occurrence* violations as follows:

Order violations. We observe an *order violation* if there exists a knowledge record $r \in \mathcal{K}$ that states two events should occur in the opposite order as they have been observed. Formally, we introduce the $order\_violation$ predicate for two events $e_i, e_j \in \sigma$:

$$\begin{aligned} &order\_violation(e_i, e_j, KB) := \\ &\quad i < j \ \wedge obj(e_i) = obj(e_j) \ \wedge \exists r \in KB : \texttt{match}(e_j, e_i, r) \ \wedge r.rel = \prec \end{aligned} \tag{1}$$

An example for an order violation could occur for "*Send offer*" and "*Create offer*" events. Obviously, no business object can be sent before it is created, thus leading to an order violation.

Exclusion violations. Given a trace σ, we observe an *exclusion violation* if there exists a knowledge record $r \in \mathcal{K}$ that defines an exclusion relation between two events in the trace, i.e., $e_i, e_j \in \sigma$. Formalized as follows:

$$\begin{aligned} &exclusion\_violation(e_i, e_j, KB) := \\ &\quad obj(e_i) = obj(e_j) \wedge \exists r \in KB : \texttt{match}(e_i, e_j, r) \ \wedge r.rel = \# \end{aligned} \tag{2}$$

An example of an exclusion violation follows from "*Accept offer*" and "*Reject offer*" events, since an offer is unlikely to be *accepted* and *rejected* in the same process instance.

Co-occurrence violations. The detection of *co-occurrence violations* is more complex than the others, given that these are based on the absence rather than the presence of certain event pairs. To illustrate their detection, consider a record $r = (create, \Rightarrow, archive, supp_r)$, which states that archived business objects must also be created. If a trace σ contains an event e with the label *archive case*, such that $act(e) = archive$, then according to the record r, trace σ must also contain an event e' with the label *create case*. However, it is well-imaginable that an event log does not contain any events with a *create event* label, for instance, because cases are *received* rather than *created* in this particular process. Therefore, since the assertion imposed by record r shall never be satisfied for such an event log, we do not detect an anomaly in these cases. In other words, even if a record specifies that an *order* should also be *created* when it is *archived*, we do not consider a trace that omits the *create* action an anomaly if this behavior simply never occurs in a log. To capture this consideration, we formally define the $co\text{-}occ\_violation$ predicate as follows:

$$\begin{aligned} &co\text{-}occ\_violation(L, \sigma, e_i, KB) := \exists r \in KB : \\ &\exists \sigma' \in L : \big(\exists e'_i, e'_j \in \sigma' : e'_i.l = e_i.l \wedge obj(e'_i) = obj(e'_j) \wedge \texttt{match}(e'_i, e'_j, r)\big) \wedge \\ &\nexists e_j \in \sigma : obj(e_i) = obj(e_j) \wedge \texttt{match}(e_i, e_j, r) \end{aligned} \tag{3}$$

In Equation 3, the first part captures that the specific co-occurrence relation r is satisfied by at least one trace $\sigma' \in L$, whereas there is no event $e_j \in \sigma$ that satisfies this knowledge record for the trace at hand (σ).

4.3.3. Loop Handling

When detecting anomalous behavior, the impact of loops, i.e., recurring behavior within a process instance, needs to be carefully considered. To illustrate this importance, consider the trace[1] depicted in Figure 3. In this trace, we observe that an order is both *rejected* and *accepted*, two actions that jointly lead to an *exclusion violation*. However, when we look at the other events in the trace, we also see that there is some repetitive behavior surrounding these events. In particular, after *rejecting* an order, it is *updated* and subsequently *checked* once more (a repeated event). Only after this recurring behavior, i.e., after the loop that seems to have corrected the initial reason to reject the order, is an order then accepted. As such, by considering the meaning of the entire trace, it is arguable that this behavior is not necessarily anomalous, despite the violation of the $(reject, \#, accept)$ assertion from a knowledge base.

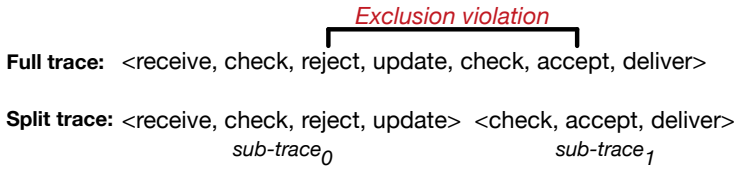

Figure 3: Impact and handling of loops in traces

We propose to prevent the detection of such false positives by only checking for anomalous behavior within the same cycle of an instance's execution, i.e., by avoiding the comparison of behavior occurring within a loop through the process. Although loops may be detected based on a (discovered) process model, this dependency on a model may be unfavorable for anomaly detection, given that this task inherently assumes that no accurate model is available. Therefore, we turn to the recognition of loops at a trace level. In particular, as depicted in the lower part of Figure 3, we split traces into sub-traces, where each sub-trace is established so that it does not have any recurring behavior. We achieve these splits by creating a new sub-trace at each occurrence of an event with a label that is also included in the current sub-trace, e.g., since an event with label *check* is already included in $sub\text{-}trace_0$, the sub-trace ends after 4 events and we create $sub\text{-}trace_1$ that starts at the previously seen event and continues until the last event in the trace.

Using $split(\sigma)$ to refer to the list of sub-traces obtained by splitting trace σ, we then propose to treat each sub-trace in $split(\sigma)$ as an individual trace for the purposes of anomaly detection.

[1]For brevity, the business object *order* is omitted from all event labels

4.3.4. Approach outcome

The result of our approach is a collection of detected violations V. We represent each violation $v \in V$ as a tuple $v = (E_i, E_j, r, n, L_v)$, where E_i and E_j are event classes that caused the specific violation (typically corresponding to activity labels $e.l$), r the relevant knowledge record, n is the number of occurrences, and $L_v \subseteq L$ represents the traces of the log L for which the violation was detected. As shown in the exemplary Table 2, a single record can be associated with multiple detected violations, involving different event-class pairs (rows 1 and 4). Furthermore, due to semantic similarity considerations, there are violations where the actions in the knowledge record are not necessarily equal to the ones in the event classes (row 2). Naturally, the results in V can be filtered and ranked in different manners. For instance, one may choose to omit all violations that only occur once or that are otherwise rare.

Table 2: Exemplary results of the approach (traces L_v omitted)

Type	**Event class 1**	**Event class 2**	**Knowledge record**	**Count**
co-occ	Send application	Application received	(send, $\Rightarrow$, receive, 100)	30
exclusion	Approve order	Refuse order	(accept, $\#$, decline, 100)	15
order	Evaluate application	Application received	(receive, $\prec$, evaluate, 20)	5
co-occ	Send invoice	Invoice received	(send, $\Rightarrow$, receive, 100)	2

5. Evaluation

This section presents our evaluation experiments. We assess the anomalies detected by our approach for both synthetic and real-world data and compare our approach against existing, frequency-based anomaly detection techniques. Our implementation, the employed data collections, and raw results are all available via a public repository.[2]

5.1. Data Collections

Our evaluation experiments are conducted using two data collections: The *BPMAI log collection* consists of event logs generated on the basis of real-world process models, whereas the *real-world log collection* consists of various widely-employed real-world event logs.

BPMAI log collection. To be able to assess the precision with which anomalies are detected, we generate a set of event logs from process models from the BPM Academic Initiative (BPMAI) repository [25]. This collection consists of thousands of process models created by users of modeling tools, covering various

[2] https://gitlab.uni-mannheim.de/processanalytics/semanticanomalydetection

modeling notations as well as different (natural) languages. As such, this part of the evaluation shall be conducted using *synthetic logs* generated from *real-world models* with real, user-defined activity labels.

From the BPMAI repository, we select all models that are (1) in the BPMN notation, (2) can be turned into a sound workflow net, (3) are in English, and (4) contain at least two activities that have the same, possibly empty, business object (as identified by the parser described in Section 4.1). The former two requirements ensure that we are able to generate proper event logs from the models, whereas the latter two ensure that each included model at least has a theoretical possibility of containing the kind of semantic anomalies our approach focuses on. After applying these filtering steps, we end up with a collection of 2,832 process models.

We transform each of these models into a workflow net, to be able to automatically generate event logs from them using PM4Py's [31] playout functionality. Subsequently, we generate a noisy counterpart for each of these event logs by first expanding the event log so that it contains 1,000 traces (if not already) and then inserting noise into the expanded log based on an established mechanism, which randomly adds, removes or swaps events in selected traces.

The characteristics of the BPMAI log collection obtained in this manner are described in Table 3. As shown, the logs greatly vary in terms of their complexity. For instance, the logs have on average of 8 event classes, but the maximal number of event classes observed is 64. Similarly, the average number of unique BOs (business objects) is 6.9, but the maximum is 52. The differences between the original and noisy logs manifest themselves clearly when considering the variants. While the original logs contain an average of 68.0 different variants, they increase to 336.7 for the noisy event logs. We also observe that the length of these variants increases due to the inserted noise, from 4.0 to 6.3 on average and an increase in the maximum length from 10 to 24 events.

Table 3: Characteristics of the BPMAI log collection (with macro-averages per log where applicable)

Collection	**Count**	**Event classes**		**Unique BOs**		**Variants**		**Variant length**	
		avg.	**max.**	**avg.**	**max.**	**avg.**	**max.**	**avg.**	**max.**
Original logs	2832	8.0	64	6.9	52	68.0	129 038	4.0	10
Noisy logs	2832	8.0	64	6.9	52	336.7	127 987	6.3	24

Real-world log collection. We complement the evaluation on the synthetic BPMAI logs with a qualitative evaluation of three real-world event logs from the Business Process Intelligence (BPI) challenges. In particular, we consider the following three logs:

- BPI12 [12]: This event log captures data from a process in which customers apply for a personal loan or overdraft with a financial institution. The log contains 262,200 events across 13,087 cases and has 24 unique event classes.

- BPI15 [9]: This event logs captures data from a permit process of a Dutch municipality. We specifically use the first log of the data set, which contains 52,217 events across 1,199 cases and has 398 unique event classes.
- BPI18 [11]: This event log captures data about the handling of applications for EU direct payments for German farmers from the European Agricultural Guarantee Fund. The log contains 2,514,266 events across 43,809 cases and has 41 unique event classes.

5.2. Setup

This section describes the implementation, parameters, baselines, setting, and metrics used to conduct our evaluation experiments.

Implementation. We implemented our anomaly detection approach and evaluation pipeline in Python. The publicly available implementation employs PM4Py [31] to handle the import and generation of event logs, and as a foundation for the employed baselines. The implementation, furthermore, uses BERT [15] for the tagging of natural language labels, as described in Section 4.1, and GloVe [32] for the efficient computation of semantic similarity between words.

Parameters. While conducting our experiments, we asses the impact of the following components and parameters on the obtained results:

- **Matching approach (`equiv` and τ_{sim}):** We vary the instantiation of the `equiv` predicate which determines when two actions are considered to be equivalent (Section 4.3.1), using: *equality* (EQ), *synonomous* (SYN), and *semantic similarity* (SEM). For the latter, we use employ different similarity thresholds, with $\tau_{sim} \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$.
- **Restricted similarity matching (`one_sim`):** When using the *SYN* or *SEM* matching approaches, if `one_sim` is employed, our approach only matches a record to a pair of events if at least one of the event's actions is equal to the corresponding action in the record, i.e., only the other action may be a synonym or semantically-related term.
- **Knowledge record conflict resolution (`fix_confl`):** We assess the impact of the conflict-resolution strategies described in Section 4.2.3, which filter out contradicting records from a knowledge base.
- **Loop handling (`split_loops`):** We assess the impact of the loop-handling method described in Section 4.3.3 by testing configurations with and without it.

Baselines. We compare our approach against three frequency-based anomaly detection approaches. In particular, we consider two established statistical baselines and one based on neural networks:

1. $\mathbf{BL_{freq}}$: This baseline, introduced by Bezerra et al. [5], simply considers any trace variant (i.e., an activity sequence) that comprises less than $x\%$ of the total traces in the event log as anomalous. Following the baseline's default settings, we use $x\% = 5\%$.

2. **BL$_{\text{sample}}$**: This baseline, identified as the best performing one by Bezerra et al. [5], consists of two stages. First, the approach takes a sample of $s\%$ traces from a log L and discovers a process model M_s based on this sample. Subsequently, any variant that comprises less than $x\%$ of the total traces in the event log and is not a proper execution sequence of M_s is considered as anomalous. Following the default settings, we use $s\% = 70\%$ and $x\% = 5\%$.
3. **BL$_{\text{binet}}$**: This baseline corresponds to the BINet approach proposed by Nolle et al. [33]. BINet is a deep learning approach that uses a neural network architecture for multi-perspective anomaly detection and classification. When employing the baseline, we use the settings that were reported as best performing in the baseline's original evaluation.

Cross-validation setting. Since the models from the BPMAI collection are used to both populate the knowledge base and for the generation of event logs in which we detect anomalies, we conduct our experiments using *10-fold cross-validation*. As such, we randomly split the collection, consisting of 2,832 models, into 10 chunks. Subsequently, we use the models in 9 chunks (i.e., the training set) to populate the knowledge base, whereas the anomaly detection approach is then applied on the noisy event logs corresponding to the models in the 10th chunk, i.e., the test chunk. We conduct these experiments so that each of the 10 chunks serves as the test chunk once.

Evaluation metrics. For the experiments involving the BPMAI collection, we argue that the process models established by users indicate behavior that is explicitly allowed in a process. Any behavior that is not allowed in the model can then be considered to be anomalous. As such, given an original log L and its noisy counterpart L', we apply our anomaly detection approach on L', which yields a collection of detected anomalies (or *violations*) $V_{L'}$ (see Section 4.3.4). For each detected anomaly $v \in V_{L'}$, we subsequently check if the detected behavior indeed violates the behavior from the original model, resulting in a true positive (tp) or false positive assessment (fp). Based on this, we quantify the number of true anomalies that our approach detects, as well as its precision, given as $\frac{tp}{tp+fp}$. Since we acknowledge that not all anomalous behavior is detectable by considering its meaning captured in event labels, we do not strive for a complete coverage in terms of recall.

To allow for an appropriate comparison between our approach and the baselines, we employ the same procedure but instead focus on the variant level, since the baseline approaches detect anomalies per trace (BL_{freq} and BL_{sample}) or at a different level of granularity (BL_{binet}) than our approach. Therefore, we assess if variants in which an anomaly has been detected according to our approach (or recognized as anomalous by a baseline) are permissible according to the original process model.

5.3. Results BPMAI Collection

Table 4 depicts the evaluation results obtained for the noisy logs generated from the models in the BPMAI collection, using 19 configurations of our approach. Overall, we observe that the performance varies

considerably across the configurations. The true positives range from 2,617 (configuration EQ_4) up to 6,494 (SEM_5), whereas the precision ranges between 0.57 (also SEM_5) and 0.76 (various configurations). Generally, as expected, configurations that identify more anomalies also have a lower precision. However, certain configuration options lead to a smaller decrease in precision in comparison to the additional anomalies they detect, making them favorable.

Table 4: Evaluation results for various configurations

Configuration		**Overall**		**Exclusion**		**Order**		**Co-occ.**	
ID	Extra options	TP	Prec.	TP	Prec.	TP	Prec.	TP	Prec.
EQ_1	—	4348	0.62	2566	0.63	1293	0.86	489	0.34
EQ_2	`split_loops`	4305	0.62	2539	0.62	1280	0.86	486	0.34
EQ_3	`fix_confl`	2609	0.76	1593	0.74	893	0.87	123	0.53
EQ_4	`fix_confl`, `split_loops`	2617	0.76	1625	0.74	878	0.87	114	0.51
SYN_1	—	5493	0.60	2793	0.58	2033	0.82	667	0.34
SYN_2	`split_loops`	5410	0.60	2765	0.58	2015	0.83	630	0.33
SYN_3	`fix_confl`	3589	0.71	1846	0.66	1549	0.84	194	0.47
SYN_4	`one_sim`	5341	0.60	2771	0.59	1942	0.84	628	0.33
SYN_5	`fix_confl`, `one_sim`	3463	0.73	1828	0.68	1445	0.87	190	0.48
SEM_1	$\tau_{sim} = 0.7$	4749	0.61	2652	0.62	1580	0.85	517	0.32
SEM_2	$\tau_{sim} = 0.7$, `split_loops`	4688	0.61	2624	0.62	1539	0.84	525	0.33
SEM_3	$\tau_{sim} = 0.7$, `fix_confl`	2950	0.76	1676	0.72	1151	0.88	123	0.45
SEM_4	$\tau_{sim} = 0.7$, `fix_confl`, `one_sim`	2970	0.76	1711	0.73	1126	0.89	133	0.48
SEM_5	$\tau_{sim} = 0.5$	6494	0.57	2996	0.55	2643	0.78	855	0.33
SEM_6	$\tau_{sim} = 0.5$, `fix_confl`	4491	0.66	1998	0.60	2120	0.80	373	0.44
SEM_7	$\tau_{sim} = 0.5$, `fix_confl`, `one_sim`	3860	0.69	1917	0.63	1673	0.84	270	0.49
SEM_8	$\tau_{sim} = 0.6$,	5393	0.59	2790	0.58	1987	0.82	616	0.32
SEM_9	$\tau_{sim} = 0.8$,	4469	0.62	2605	0.63	1346	0.84	518	0.35
SEM_{10}	$\tau_{sim} = 0.9$,	4403	0.62	2596	0.63	1312	0.85	495	0.34

When considering the three kinds of anomalies, we observe consistent trends across the configurations. Particularly, our approach consistently detects the most exclusion violations (2,222 on average) with a respectable level of precision (0.65 avg.). While it detects fewer order violations (1,507 avg.), the precision for these is high (0.85 avg.). Finally, the detection of co-occurrence violations trails in both aspects, with an average of 382 detected anomalies and a precision of 0.41. For the latter anomalies, the variance is also the greatest across the configurations, with the number of true positives ranging from 114 to 855 and the precision from 0.32 to 0.53. In the following, we investigate the impact of the various configuration options

in detail.

Record matching approaches (EQ vs. SYN vs. SEM). The results obtained using *equivalence*, *synonym*, and *semantic similarity*-based matching reveal various interesting observations. In general, we find that the usage of the similarity-based approaches (SYN and SEM) is favorable over the strictest, equivalence-based approach. When comparing otherwise equal configurations, the similarity-based approaches recognize more anomalies, with a minimal impact on precision. For instance, EQ_1 detects 4,348 true anomalies with a precision of 0.62, whereas a corresponding configuration based on semantic similarity (SEM_1), detects 464 additional anomalies with a near-identical precision of 0.61, while the corresponding SYN_1 configuration detects 1,121 additional anomalies with a minimal reduction in precision (0.60), when compared to EQ_1.

Since these trends are consistent across the other configuration options, we can conclude that the synonym and semantic-similarity configurations are favorable over the ones using strict matching. When comparing these latter two matching approaches, there is no obvious winner. For certain options, the synonym-based configurations perform better, e.g., for SYN_1 vs SEM_1 and SYN_2 vs SEM_2 we observe that the synonym-based approach detects considerably more anomalies with a minimal reduction in precision. However, the semantic similarity-based option can be more favorable when incorporating other configuration options. For example, the highest precision achieved by SYN approaches is 0.73 (SYN_5), whereas the SEM ones achieve a maximal precision of 0.76, for several configurations.

Restricted similarity matching. For the similarity-based SYN and SEM approaches, we also tested the `one_sim` option, which requires that one of the actions in a record is equivalent to observed behavior, whereas the other action can be a synonym (SYN) or semantically-similar to the recorded action (SEM). In general, introducing `one_sim` results in a clear trade-off between increased precision and the number of detected anomalies, e.g., between SEM_6 and SEM_7, the true positives are reduced from 4,491 to 3,760, while precision increases from 0.66 to 0.69.

Impact of KB conflict resolution. The results demonstrate that incorporating the conflict resolution option (`fix_confl`) greatly improves the precision of the anomaly detection approach, yet also considerably reduces the number of detected anomalies. For instance, for the otherwise equal configurations SYN_1 and SYN_3, the `fix_confl` option reduces the true positives from 5,493 to 3,589 (-35%) but increases the precision from 0.60 to 0.71. When considering the impact per anomaly type, we observe that this general trend also applies to exclusion violations, whereas it is amplified for co-occurrence violations. For instance, between SYN_1 and SYN_3, the option reduces the true positives from 667 to 194 (-71%) while increasing the precision from 0.34 to 0.47. For order violations, the conflict resolution strategy generally has a limited impact on the precision, while decreasing the number of true positives. For example, between SYN_1 and SYN_3, the precision is increased minimally (0.82 to 0.84), while the detected anomalies are reduced from 2,033 to 1,549.

As such, though results vary, a general guideline would be to not apply `fix_confl` for order anomalies,

whereas for the other anomaly types its application depends on the desired degree of precision versus detected anomalies.

Loop handling. The explicit handling of loops, in which anomalies are only detected in sub-traces without repeating parts, generally has a minimal impact on the results for this data set. Nevertheless, on a case by case basis, e.g., when a process is known to have repetitive behavior, the inclusion of this option can be favorable, given that it also does not appear to have a negative impact in other situations.

Support threshold for semantic similarity. As expected, changing the semantic similarity threshold τ_{sim} provides a clear trade-off between precision and true positives when lowering the parameter from 0.7 to 0.6 and 0.5, where a lower value leads to a considerable increase in true positives, (e.g., 6,494 for SEM_5 versus 4,749 for SEM_1) yet also to a some decrease in precision (0.57 to 0.61). However, it is interesting to observe that this pattern does not necessarily apply for stricter similarity thresholds. Increasing τ_{sim} to 0.8 or 0.9 yields equal precision values, while reducing the number of true positives. As such, a $\tau_{sim} \leq 0.7$ appears to be favorable, whereas the exact value depends on the desired precision and true positives.

Optimal settings. Overall, we observe that the various configuration options allow one to trade-off precision versus the number of detected anomalies as desired. For high precision, SEM_4, which uses restricted, semantic similarity matching and conflict resolution, is favorable. From the various configurations that achieve a precision of 0.76, this one detects the most true anomalies. By contrast, when aiming to detect more true anomalies, SEM_5 can be recommended. It detects the highest number of anomalies by a distance (6,494 versus at most 5,493 for others), while the precision of 0.57 is not that much lower than the 0.60 precision of those other configurations. In the remainder of the evaluation, we therefore continue employing these two configurations, i.e., SEM_4 and SEM_5.

5.4. Comparison to Frequency-Based Approaches

Table 5 presents the main results when comparing the two best configurations of our approach against the three baselines described in Section 5.2. In order to allow for a comparison between them and our approach, the results are presented on the trace variant-level, i.e., we assess which trace variants (unique execution sequences) each approach identified as anomalous and how many of those were indeed not part of the original process model.

Results comparison. When comparing the baselines among each other, it is interesting to note that the simpler BL_{freq} and BL_{sample} baselines outperform the learning-based BL_{binet} approach for the data collection at hand. When comparing our approach to the baselines, the results show that the baselines identify more anomalous variants than our approach. This is, naturally, to be expected, given that the baselines identify anomalies in a frequency-based manner, which is directly in line with the noise insertion approach we used to introduce anomalies in the first place. Nevertheless, it is clear that our approaches

Table 5: Comparison of results to baseline at the trace variant-level

Approach	TP	FP	Prec. (macro)	Prec. (micro)
BL_{freq}	860 385	110 990	0.886	0.988
BL_{sample}	674 861	28 826	0.959	0.991
BL_{binet}	789 727	94 817	0.981	0.893
SEM_4	110 711	1503	0.987	0.998
SEM_5	257 376	11 260	0.958	0.993

are in the same order of magnitude as the baselines in terms of true positives. On top of that, even though our approaches do not build on frequency analysis, their precision is still higher than the precision of the baselines, e.g., even achieving a micro precision of 0.998 for SEM_4. As such, this shows that the consideration of semantic anomalies yields a considerably lower number of false positives than the frequency-based approaches, e.g., just 1,503 for SEM_4 versus up to 110,990 anomalies for the most naive baseline.

Hybrid configurations. Overall, the main takeaway from this comparison is that our approach should be regarded as inherently different to the existing works, given that we aim to identify a different kind of anomalies. As such, the consideration of frequency and meaning should be regarded as complementary when detecting anomalies. To demonstrate this, we also combined our approach with the same frequency-based consideration of BL_{freq}. The resulting hybrid configuration considers behavior anomalous when our approach recognizes a semantic violation and the behavior occurs in less than 5% of the traces in the log. As shown in Table 6, the results obtained in this manner greatly outperform any configuration without frequency-based considerations. For example, the SEM_5 + BL_{freq} configuration achieves a much higher precision than any approach from Table 4, while detecting more anomalies than any configuration (aside from SEM_5 itself). While the benefits are noticeable for all types of anomalies, it is particularly interesting to observe the impact on the detection of co-occurrence anomalies. Here, the detected anomalies are reduced from 855 to 775 (-9%), yet the precision increases from 0.33 to 0.93 (+181%).

Table 6: Results when combining semantic and frequency-based anomaly detection

	Overall		Exclusion		Order		Co-occ.	
Configuration	TP	Prec.	TP	Prec.	TP	Prec.	TP	Prec.
SEM_4	2970	0.76	1711	0.73	1126	0.89	133	0.48
SEM_4 + BL_{freq}	2406	0.91	1470	0.86	815	0.99	121	0.97
SEM_5	6494	0.57	2996	0.55	2643	0.78	855	0.33
SEM_5 + BL_{freq}	5504	0.85	2629	0.76	2100	0.98	775	0.93

5.5. Results Real-World Log Collection

We also applied our approach on several real-world event logs, described in Section 5.1. Although there is no gold standard available for these processes in terms of which behavior is considered to be anomalous or not, these logs still allow us to investigate anomalies detected by our approach for real-world situations in a qualitative manner. For this, we employed a configuration similar to SEM_5, yet with explicit handling of loops in a process (`split_loops`).

Table 7: Selection of anomalies detected in real-world logs

Anomaly type	**Event log**	**Anomaly**	**Frequency**
Order violation	BPI12	A case was *cancelled* and then *accepted*	706
	BPI12	A case was *cancelled* and then *approved*	708
	BPI15	An updated plan was *received* before it was *requested*	2
	BPI18	*Finish preparations* before *Begin preparations*	2430
	BPI18	*Remove payment block* before it was *set*	16
Exclusion violation	BPI18	An application was both *refused* and *withdrawn*	46
	BPI18	An application was both *withdrawn* and *approved*	5
	BPI18	An application was both *withdrawn* and *approved*	5
Co-occ. violation	BPI15	A procedure confirmation was *created* but not *sent*	517
	BPI15	A procedure confirmation was *sent* but not *created*	609
	BPI18	An application was *saved* but never *created*	12 814
	BPI18	A payment has *begun* but not *finished*	6

Table 7 presents a number of anomalies detected in the real-world logs, selected based on their semantic clarity. We observe interesting cases for all three anomaly types. In terms of *order violations*, our approach detected various occurrences that should definitely not happen during proper process execution. For instance, our approach detected various instances that were canceled, yet still accepted or approved afterwards in BPI12. For BPI15, we identified that occasionally a business object was received before it was even requested (or requested after it had already been received). In terms of *exclusion violations*, we observed interesting cases for the BPI18 log, in which applications were both withdrawn and approved or refused, clearly indicating the occurrence of redundant activities. Finally, in terms of *co-occurrence violations*, we observed various interesting anomalies, such as confirmations that were created but never sent in BPI15, or ones that were sent but had never been created in the first place. Similarly, for BPI18, we found that many applications were saved, even though they had never been created.

Aside from the interestingness of these specific anomalies, the evaluation on the real-world logs also leads to several other observations:

- First, although our loop handling method (`split_loops`) had little impact when applied to the synthetic log collection, its benefits are clear for these real-world logs, which have longer traces with quite some repetitions. As such, by explicitly handling loops, our approach was able to avoid the detection of certain clear false positives.
- Second, although our approach still did not detect many co-occurrence violations in the real-world logs, it is clear from the examples in Table 7 that our approach is nonetheless able to detect some highly interesting anomalies of this type, thus clearly demonstrating the value of these considerations.
- Finally, it is important to note that our approach detected very infrequent anomalies, occurring only a handful of times, as well as anomalies that occurred thousands of times. While we do not claim that all detected anomalies are problematic from a business perspective, the listed cases are nevertheless clearly remarkable. For instance, it does not seem desirable that applications can exist in the system without having been *created*, even if this happens often.

Overall, these findings thus clearly demonstrate the benefits of our approach in real-world settings and, due to their ability to detect both common and uncommon (potential) anomalies, we again demonstrated the complementary nature of our semantic-based approach to existing frequency-based techniques.

6. Related Work

Anomaly detection is a well-studied problem that has been researched in diverse areas and application domains. In the context of discrete sequences, Chandola et al. [34] differentiate between three main types of anomaly detection techniques: sequence-based, contiguous subsequence-based, and pattern-based. Sequence-based techniques use supervised as well as unsupervised methods to assign an anomaly score to an entire sequence. One important class of sequence-based technique are window-based techniques [35]. They extract fixed-length overlapping windows from a sequence and assign each window an anomaly score. The anomaly scores of all windows are then aggregated to an anomaly score for the entire sequence. The main rationale is that analyzing windows allows to detect anomalies that, otherwise, would not be distinguishable from the variation that exists across sequences. Contiguous subsequence-based techniques aim to detect short contiguous subsequences in a sequence that are anomalous with respect to rest of the sequence. One popular strategy to achieve this are window scoring techniques [36]. They count how many times a window of a particular size occurs. The lower the frequency of a window, the higher its anomaly score. Pattern-based techniques require the user to provide a query pattern [37]. If the frequency of this pattern is then found to deviate from what is estimated based on a set of training sequences, the pattern is considered to be anomalous.

Anomaly detection techniques in the area of process mining build on these considerations to address various use cases. It is an inherent part of most process discovery algorithms, which aim to filter out anomalies

(often corresponding to *noise*) in order to preserve the most common process behavior [38, 39]. Other works present stand-alone approaches for this task, which detect anomalies based on a comparison to discovered process models [5] and using unsupervised machine learning [7]. Whereas most existing approaches detect anomalies based on control-flow information, other works also consider the data perspective, e.g., in the context of process discovery [40] and the repair of event log imperfections [41]. One of the most recent contributions is BINet, a deep learning approach that uses a neural network architecture for multi-perspective anomaly detection and classification [33]. As the authors demonstrate, BINet outperforms other available methods and, therefore, must be considered state-of-the-art in the area of business process anomaly detection. The work presented in this paper is complimentary to such existing works, where a combination would enable anomaly detection based on control-flow, data, *and* semantic aspects. Our evaluation experiments showed that such an hybrid approach is indeed promising and can greatly improve precision.

Our work also relates to various works that analyze the contents of textual labels associated with process models and events, such as our earlier parsing [21] and extraction techniques [14], which have been applied for purposes such as process model matching [42] and the identification of service and automation candidates [43, 44]. Finally, in a broader context, the detection of semantic anomalies relates to works that consider the notions of common sense in other scenarios. For example, various resources capture common sense relations, such as the linguistic resources discussed in Section 4.2.1 or common sense knowledge graphs [45, 46]. Close to our application context, such common sense consideration are, among others, employed to improve the quality of actions and state changes extracted from natural language texts [47, 48].

Still, our approach stands out as the first work that lifts the consideration of meaning associated with events to the detection of anomalies in a process analysis or process mining context.

7. Conclusion

In this paper, we presented a completely novel way of detecting anomalies in process mining by taking the meaning of observed process behavior into account. To achieve this, our approach exploits the natural language labels associated with events. The semantic components extracted from these labels are compared against a process-independent knowledge base, populated based on both linguistic and process-oriented resources. We demonstrated the capability of our approach to successfully detect anomalies in both synthetic and real-world logs and showed its compatibility with existing, frequency-based detection techniques. As such, our evaluation clearly highlights the potential of using natural language analysis for anomaly detection.

As the first of its kind, there are various manners in which we aim to expand and improve our approach in the future. This will include the coverage of other kinds of anomalies, such as anomalies involving the relations among different business objects. The population of a knowledge base will be extended by employing additional kinds of resources, e.g., involving knowledge graphs and state-of-the-art natural language process-

ing techniques. Furthermore, we recognize that the detection of semantic execution anomalies can be lifted to various new application scenarios, e.g., to support the introduction of noise for event log privatization [49], to resolve event data uncertainty [50], or to consider the severity of conformance violations [51].

Reproducibility: A link pointing towards the publicly available implementation and employed data collections is given in Section 5.

References

[1] F. Bagayogo, A. Beaudry, L. Lapointe, Impacts of IT acceptance and resistance behaviors: a novel framework, in: International Conference on Information Systems, 2013.

[2] R. Lu, S. Sadiq, G. Governatori, Compliance aware business process design, in: International Conference on Business Process Management, Springer, 2007, pp. 120–131.

[3] F. Caron, J. Vanthienen, B. Baesens, Comprehensive rule-based compliance checking and risk management with process mining, Decision Support Systems 54 (3) (2013) 1357–1369.

[4] A. Rozinat, W. M. P. Van der Aalst, Conformance checking of processes based on monitoring real behavior, Information Systems 33 (1) (2008) 64–95.

[5] F. Bezerra, J. Wainer, Algorithms for anomaly detection of traces in logs of process aware information systems, Information Systems 38 (1) (2013) 33–44.

[6] T. Nolle, A. Seeliger, M. Mühlhäuser, Unsupervised anomaly detection in noisy business process event logs using denoising autoencoders, in: International conference on discovery science, Springer, 2016, pp. 442–456.

[7] T. Nolle, S. Luettgen, A. Seeliger, M. Mühlhäuser, Analyzing business process anomalies using autoencoders, Machine Learning 107 (11) (2018) 1875–1893.

[8] J. Mendling, H. A. Reijers, J. Recker, Activity labeling in process modeling: Empirical insights and recommendations, Information Systems 35 (4) (2010) 467–482.

[9] Van Dongen, B.F., BPI Challenge 2015 (2015). doi:10.4121/UUID:31A308EF-C844-48DA-948C-305D167A0EC1.

[10] Van Dongen, B.F., BPI Challenge 2014 (2014). doi:10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35.

[11] Van Dongen, B.F., BPI Challenge 2018 (2018). doi:10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972.

[12] Van Dongen, B.F., BPI Challenge 2012 (2012). doi:10.4121/UUID:3926DB30-F712-4394-AEBC-75976070E91F.

[13] J. Buijs, Environmental permit application process ('wabo'), coselog project (2014). doi:10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270.

[14] A. Rebmann, H. van der Aa, Extracting semantic process information from the natural language in event logs, in: International Conference on Advanced Information Systems Engineering, 2021.

[15] J. Devlin, M. W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, NAACL 2019 1 (Mlm) (2019) 4171–4186.

[16] G. A. Miller, WordNet: a lexical database for english, Communications of the ACM 38 (11) (1995) 39–41.

[17] M. Palmer, C. Bonial, J. D. Hwang, Verbnet: Capturing english verb behavior, meaning and usage, The Oxford Handbook of Cognitive Science (2017) 315–336.

[18] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, Dbpedia: A nucleus for a web of open data, in: The semantic web, Springer, 2007, pp. 722–735.

[19] Y. Lin, Z. Liu, M. Sun, Y. Liu, X. Zhu, Learning entity and relation embeddings for knowledge graph completion, in: Twenty-ninth AAAI conference on artificial intelligence, 2015, pp. 2181–2187.

[20] T. Chklovski, P. Pantel, Verbocean: Mining the web for fine-grained semantic verb relations, in: Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, 2004, pp. 33–40.

[21] H. Leopold, H. van der Aa, J. Offenberg, H. A. Reijers, Using hidden markov models for the accurate linguistic analysis of process model activity labels, Information Systems 83 (2019) 30–39.

[22] K. Barker, S. Szpakowicz, Interactive semantic analysis of clause-level relationships, in: Proceedings of the Second Conference of the Pacific Association for Computational Linguistics (PACLING), 1995.

[23] C. Fellbaum, Wordnet, The encyclopedia of applied linguistics (2012).

[24] D. A. Cruse, Antonymy revisited: Some thoughts on the relationship between words and concepts, Frames, Fields, and Contrasts, Hillsdale, NJ, Lawrence Erlbaum associates (1992) 289–306.

[25] M. Weske, G. Decker, M. Dumas, M. La Rosa, J. Mendling, H. A. Reijers, Model Collection of the Business Process Management Academic Initiative (2020). doi:10.5281/zenodo.3758705.

[26] S. Smirnov, M. Weidlich, J. Mendling, M. Weske, Action patterns in business process model repositories, Computers in Industry 63 (2) (2012) 98–111.

[27] E. Agirre, E. Alfonseca, K. Hall, J. Kravalova, M. Pasca, A. Soroa, A study on similarity and relatedness using distributional and wordnet-based approaches, in: HLT-NAACL, 2009, pp. 19–27.

[28] D. Lin, An information-theoretic definition of similarity., in: ICML, Vol. 98, 1998, pp. 296–304.

[29] P. Kolb, Disco: A multilingual database of distributionally similar words, Proceedings of KONVENS-2008, Berlin (2008).

[30] T. K. Landauer, P. W. Foltz, D. Laham, An introduction to latent semantic analysis, Discourse processes 25 (2-3) (1998) 259–284.

[31] A. Berti, S. J. van Zelst, W. van der Aalst, Process mining for python (PM4Py): Bridging the gap between process-and data science, in: ICPM Demo Track 2019, 2019, pp. 13–16.

[32] J. Pennington, R. Socher, C. D. Manning, GloVe: Global vectors for word representation, in: EMNLP, 2014, pp. 1532–1543.

[33] T. Nolle, S. Luettgen, A. Seeliger, M. Mühlhäuser, Binet: Multi-perspective business process anomaly classification, Information Systems (2019) 101458.

[34] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection for discrete sequences: A survey, IEEE transactions on knowledge and data engineering 24 (5) (2010) 823–839.

[35] C. Warrender, S. Forrest, B. Pearlmutter, Detecting intrusions using system calls: Alternative data models, in: Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344), IEEE, 1999, pp. 133–145.

[36] E. Keogh, J. Lin, S.-H. Lee, H. Van Herle, Finding the most unusual time series subsequence: algorithms and applications, Knowledge and Information Systems 11 (1) (2007) 1–27.

[37] E. Keogh, S. Lonardi, B.-c. Chiu, Finding surprising patterns in a time series database in linear time and space, in: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, 2002, pp. 550–556.

[38] S. J. Leemans, D. Fahland, W. M. van der Aalst, Discovering block-structured process models from event logs containing infrequent behaviour, in: International conference on business process management, Springer, 2013, pp. 66–78.

[39] 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.

[40] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. van der Aalst, Data-driven process discovery-revealing conditional infrequent behavior from event logs, in: International Conference on Advanced Information Systems Engineering, Springer, 2017, pp. 545–560.

[41] P. M. Dixit, S. Suriadi, R. Andrews, M. T. Wynn, A. H. ter Hofstede, J. C. Buijs, W. M. van der Aalst, Detection and interactive repair of event ordering imperfection in process logs, in: International Conference on Advanced Information Systems Engineering, Springer, 2018, pp. 274–290.

[42] H. Leopold, M. Niepert, M. Weidlich, J. Mendling, R. Dijkman, H. Stuckenschmidt, Probabilistic optimization of semantic process model matching, in: International Conference on Business Process Management, Springer, 2012, pp. 319–334.

[43] H. Leopold, F. Pittke, J. Mendling, Automatic service derivation from business process model repositories via semantic technology, Journal of Systems and Software 108 (2015) 134–147.

[44] H. Leopold, H. van der Aa, H. A. Reijers, Identifying candidate tasks for robotic process automation in textual process descriptions, in: Enterprise, business-process and information systems modeling, Springer, 2018, pp. 67–81.

[45] J. Omeliyanenko, A. Zehe, L. Hettinger, A. Hotho, Lm4kg: Improving common sense knowledge graphs with language models, in: International Semantic Web Conference, Springer, 2020, pp. 456–473.

[46] C. Havasi, R. Speer, J. Alonso, Conceptnet 3: a flexible, multilingual semantic network for common sense knowledge, in: Recent advances in natural language processing, John Benjamins Philadelphia, PA, 2007, pp. 27–29.

[47] V. Losing, L. Fischer, J. Deigmoeller, Extraction of common-sense relations from procedural task instructions using bert, in: Proceedings of the 11th Global Wordnet Conference, 2021, pp. 81–90.

[48] N. Tandon, B. Dalvi, J. Grus, W.-t. Yih, A. Bosselut, P. Clark, Reasoning about actions and state changes by injecting commonsense knowledge, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 57–66.

[49] S. A. Fahrenkrog-Petersen, H. van der Aa, M. Weidlich, Pripel: Privacy-preserving event log publishing including contextual information, in: International Conference on Business Process Management, Springer, 2020, pp. 111–128.

[50] H. van der Aa, H. Leopold, M. Weidlich, Partial order resolution of event logs for process conformance checking, Decision Support Systems 136 (2020) 113347.

[51] J. Carmona, B. van Dongen, A. Solti, M. Weidlich, Conformance Checking: Relating Processes and Models, Springer, 2018.