# Guided Process Discovery - A Pattern-based Approach

Felix Mannhardt[a,*], Massimiliano de Leoni[a], Hajo A. Reijers[b,a], Wil M.P. van der Aalst[a], Pieter J. Toussaint[c,d]

[a] *Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*
[b] *Vrije Universiteit Amsterdam, De Boelelaan 1081, 1081 HV, Amsterdam, The Netherlands*
[c] *Norwegian University of Science and Technology, 7491 Trondheim, Norway*
[d] *SINTEF, P.O. Box 4760 Sluppen, NO-7465 Trondheim, Norway*

**Abstract**

Process mining techniques analyze processes based on events stored in event logs. Yet, low-level events recorded by information systems may not directly match high-level activities that make sense to process stakeholders. This results in discovered process models that cannot be easily understood. To prevent such situations from happening, low-level events need to be translated into high-level activities that are recognizable by stakeholders. This paper proposes the Guided Process Discovery method (GPD). Low-level events are grouped based on behavioral activity patterns, which capture domain knowledge on the relation between high-level activities and low-level events. Events in the resulting abstracted event log correspond to instantiations of high-level activities. We validate process models discovered on the abstracted event log by checking conformance between the low-level event log and an expanded model in which the high-level activities are replaced by activity patterns. The method was tested using two real-life event logs. We show that the process models discovered with the GPD method are more comprehensible and can be used to answer process questions, whereas process models discovered using standard process discovery techniques do not provide the insights needed.

*Keywords:* Process Mining, Process Discovery, Domain Knowledge, Event Logs, Event Abstraction

## 1. Introduction

Organizations use information systems to support their work. Often, information about the usage of those systems by workers is recorded in event logs [1]. Process mining techniques use such event data to analyze processes of organizations. Most techniques assume that recorded events correspond to meaningful

*Corresponding author

*Email addresses:* `f.mannhardt@tue.nl` (Felix Mannhardt), `m.d.leoni@tue.nl` (Massimiliano de Leoni), `h.a.reijers@vu.nl` (Hajo A. Reijers), `w.m.p.v.d.aalst@tue.nl` ( Wil M.P. van der Aalst), `pieter@idi.ntnu.no` (Pieter J. Toussaint)

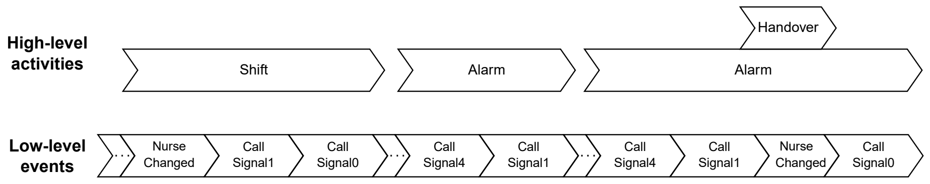


Figure 1: Multiple low-level events are recorded for one instance of a high-level activity.

activities in the instances of a process (i.e., cases). The information about recorded executions of activities can then be used, e.g., to *discover* models describing the observed behavior or to check *conformance* with existing process documentation. The ability to identify executions of activities based on events and discover recognizable models is crucial for any process mining technique. Events that do not directly correspond to activities recognizable for process workers are unsuitable for process analytics since their semantics are not clear to domain experts. Process models discovered based on such low-level events are often incomprehensible to stakeholders.

However, events recorded by information systems often do not match executions of high-level activities [2, 3]. Generally, there can be an n:m-relation between recorded events and high-level activities [2, 3]. As illustrated in Figure 1 one instantiation of a *high-level activity* may result in multiple *low-level events* being recorded (e.g., the execution *Shift* resulted in three low-level events being recorded). Moreover, one low-level event type may relate to multiple high-level activities (e.g., *CallSignal0* is both recorded by *Shift* and *Alarm*). When applying process discovery methods on low-level event logs, then, semantically related activities are not presented as such. *Event abstraction*, the grouping of low-level events to recognizable activities on a higher abstraction level, can help guiding process discovery methods towards discovering a process model that can be understood by stakeholders and is more useful for answering process questions. Grouping those related low-level events together in activities on a higher level facilitates model comprehension [4].

This paper proposes the *Guided Process Discovery method* (GPD). We use event abstraction based on domain knowledge on the process to guide process discovery techniques towards a process model that can be recognized by process stakeholders. Domain knowledge about the expected low-level behavior of high-level activities is captured by *activity patterns*. Activity patterns describe complex interactions of the control-flow, time, resource and, data perspective in terms of low-level events. An activity pattern also provides the name (i.e., activity label) of the high-level activity that it describes. It can be seen as description of the work practice on a fine granularity, i.e., low-level of abstraction. Multiple low-level events are captured within one activity pattern. We consider both *manual patterns*, which are created by a process analyst based on domain knowledge, and *discovered patterns*, which are obtained automatically from the event log and assigned suitable labels by domain experts. With a set of activity patterns at hand, we leverage *alignment techniques* to find an optimal mapping between the behavior defined by these activity patterns and the observed behavior in the event log. The use of alignments is justified by the fact that, in general, event logs are noisy. Hence, not all low-level events can be mapped onto high-level activities. Furthermore, the search for an optimal mapping requires considering entire traces. Alignment-based techniques do this by solving optimization problems. We use the information obtained from the alignment to create an abstracted *high-level event log*. Based on this abstracted event log, we *discover* a process model at the desired level of abstraction. The discovered process model needs to be *validated* since the abstraction is based on assumptions on the process. To validate the model, we expand the activities of the discovered model with their corresponding activity patterns. Then, we compute an alignment between this *expanded model* and the original event log. Quality measures (e.g., fitness) provided by the alignment technique indicate the *reliability* of the abstraction. Moreover, both the discovered high-level and the expanded lower-level process model can be used for further analysis.

We evaluated the GPD method using two real-life event logs. The first event log was retrieved from a digital whiteboard system used in a hospital in Norway. We successfully generated an abstracted event log and were able to analyze how nurses use the digital whiteboard system in their daily work. We could answer questions about the process that would have been difficult to answer using the low-level event log. The second event log was obtained from an ERP system in the emergency department of a hospital in The Netherlands. We used both discovered and manually created activity patterns based on domain knowledge of a doctor in charge of the emergency department. Then, we could show that the discovered process model obtained with the GPD method is *suitable for answering questions* about the process and to communicate with the stakeholders. In contrast, the process models discovered with state-of-the-art discovery methods *failed to provide the insights* needed.

Against this background, the remainder of this paper is structured as follows. First, we introduce necessary preliminaries (Section 2). In the main part of this paper, we present the seven steps of the GPD method (Section 3). We briefly present the implementation (Section 4). We report on the evaluation of the method using a new real-life data set (Section 5). Finally, we review related work on event abstraction (Section 6) and conclude the paper (Section 7).

## 2. Preliminaries

We introduce necessary preliminaries for the presentation of the GPD method. The input to the method is an *event log*. We use *process models* to capture activity patterns, and *alignment* techniques to obtain the abstraction mapping.

### *2.1. Event Logs*

An event log stores information about activities that were recorded by information systems while supporting the execution of a process. Each execution of a *process instance* results in a sequence of events. Events stored in the event log

Table 1: Excerpt of a trace $\sigma \in \mathcal{E}$ from an event log $L = (E, \Sigma, \#, \mathcal{E})$ recording low-level events. Each event is identified by a unique identifier **id** and records attributes **act**, **time**, **ai**, and **nurse**. We use symbol $\perp$ to denote an unrecorded attribute. Columns hl-act and hl-ai are added to illustrate which high-level activity caused the event.

| **id** | **act** | **time** | **ai** | **nurse** | **hl-act** | **hl-ai** |
|---|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... | ... |
| $e_{12}$ | NurseChanged | 122 | 12 | NurseA | Shift | 1 |
| $e_{13}$ | CallSignal1 | 122 | 13 | $\perp$ | Shift | 1 |
| $e_{14}$ | CallSignal0 | 124 | 14 | $\perp$ | Shift | 1 |
| ... | ... | ... | ... | ... | ... | ... |
| $e_{20}$ | CallSignal4 | 185 | 20 | $\perp$ | Alarm | 2 |
| $e_{21}$ | CallSignal1 | 194 | 21 | $\perp$ | Alarm | 2 |
| ... | ... | ... | ... | ... | ... | ... |
| $e_{30}$ | CallSignal4 | 310 | 30 | $\perp$ | Alarm | 3 |
| $e_{31}$ | CallSignal1 | 311 | 31 | $\perp$ | Alarm | 3 |
| $e_{32}$ | NurseChanged | 312 | 32 | NurseC | Handover | 4 |
| $e_{33}$ | CallSignal0 | 315 | 33 | $\perp$ | Alarm | 3 |

may relate to activities on different abstraction levels. Some events may record *low-level activities*, i.e., activities that do not correspond to recognizable pieces of work.

**Definition 1 (Event Log).** Given universes of attributes $A$ and values $U$, we define an event log $L$ as $L = (E, \Sigma, \#, \mathcal{E})$ with:

- $E$ is a non-empty set of unique event identifiers;
- $\Sigma \subseteq U$ is a non-empty set of activity names;
- $\# : E \rightarrow (A \not\rightarrow U)$ retrieves the attribute values assigned to an event[1];
- $\mathcal{E} \subseteq E^*$ is the non-empty set of traces over $E$. A trace $\sigma \in \mathcal{E}$ records the sequence of events for one process instance. Each event occurs only once, i.e., the same event may not appear twice in a trace or in two different traces.

Given an event $e \in E$ in the event log $L = (E, \Sigma, \#, \mathcal{E})$, we write $\#_a(e) \in U$ to obtain the value $u \in U$ recorded for attribute $a \in A$. Three mandatory attributes are recorded by each event: $\#_{act}(e) \in \Sigma$, the **name of the activity** that caused the event; $\#_{time}(e) \in U$, the **time** when the event occurred; $\#_{ai}(e) \in U$, the **activity instance**, i.e., an identifier linking multiple events, which are related to the same execution of a single activity.

[1]We use $\not\rightarrow$ to denote partial functions, i.e., the domain of the partial function $f : A \not\rightarrow U$ is a subset of $A$. Given a partial function $f$, $dom(f) \subseteq A$ denotes its domain.

**Example 1.** Table 1 shows an excerpt of a trace $\sigma \in \mathcal{E}$ obtained from a low-level event log $L = (E, \Sigma, \#, \mathcal{E})$ that is recorded by a digital whiteboard, which supports the work of nurses in a hospital. Each row represents a unique event $e \in E$ together with the produced data (i.e., attributes) created by a change in the system. The initial events are omitted. After 122 minutes a low-level $e_{12}$ with the activity name NurseChanged (NC) was recorded. The attribute *Nurse* is recorded as $\#_{Nurse}(e_{12}) = \text{NurseA}$. Next, low-level event $e_{13}$ with activity name CallSignal1 (CS1)) and low-level event $e_{14}$ with activity name CallSignal0 (CS0) are recorded by a call signal system, which is integrated with the whiteboard. Together, all three events relate to the same instance (hl-ai) of the high-level activity Shift (hl-act). Some further low-level events follow.

### *2.2. Process Model*

We use process models to capture behavior. Generally, our method is independent of the particular formalism (e.g., Petri nets, UML, Declare, BPMN) used to model processes. Therefore, we describe our method based on the set of process executions allowed by the model, independently of the formalism employed.

**Definition 2 (Process Model).** Given universes of variables $V$, values $U$, and transitions $T$, let $S = (T \times (V \not\rightarrow U))$ be the set of all possible process steps. A *process model* $p \subseteq S^*$ consists of sequences of process steps. We denote a sequence of process steps $\sigma_p \in p$ as a process trace. A process trace corresponds to one execution of the process (i.e., a process instance). Process steps $(t, w) \in \sigma_p$ in process traces correspond to executions of a transition $t \in T$ together with the variable assignment $w \in (V \not\rightarrow U)$. We denote with $P = \{p \subseteq S^*\}$ the *set of all process models.*

Executions of process step $(t, w) \in S$ can be observed as events. We use labeled process model to connect the transition $t$ to an observable activity name $l \in \Sigma$ and variables $v \in dom(w)$ to observable attribute names $a \in A$.

**Definition 3 (Labeled Process Model).** Given universes of variables $V$, values $U$, attributes $A$, transitions $T$ and activity names $\Sigma \subseteq U$, a *labeled process model* is a tuple $(p, \lambda, \nu)$ where:

- $p \subseteq (T \times (V \not\rightarrow U))^*$ is a process model,
- $\lambda : T \rightarrow \Sigma$ is an activity label function that returns the observable activity name of a transition,
- $\nu : V \rightarrow A$ is a variable label function that returns the observable attribute name of a variable.

Given a labeled process model $(p, \lambda, \nu)$, we can determine the activity name of a process step $(t, w) \in \sigma_p$. Two distinct transitions $t_1, t_2 \in T$ can be mapped to the same activity name, i.e., $\lambda(t_1) = \lambda(t_2)$ does not imply $t_1 = t_2$.

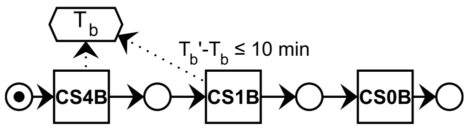


Figure 2: DPN implementation of a process model.

**Example 2.** Here, we use Data Petri nets (DPNs) [5] as language that we use to specify process models in the language-independent GPD method. This is just for the purpose of illustration: Other languages could be used. We use DPN as a rich notation with well-defined semantics, which can express the control-flow as well as the data-, resource- and time-perspective of a process. As we use DPNs only in the examples, we refer to [5] for a comprehensive introduction to DPNs. The set of valid process traces of a DPN corresponds directly to our definition of a process model based on its set of execution traces. Figure 2 depicts a DPN specification of a process model $\bar{p} = (p, \lambda, \nu)$. DPN transitions (i.e., rectangles in Figure 2) correspond to transitions in $T$ and DPN variables (i.e., hexagons in Figure 2) correspond to variables in $V$. The DPN specifies that transitions CS4B, CS1B and CS0B can only be executed in sequence and the time between executing CS4B and CS1B must be at most 10 minutes. Its activity label function is $\lambda(\text{CS4B}) = \text{CS4}$, $\lambda(\text{CS1B}) = \text{CS1}$, and $\lambda(\text{CS0B}) = \text{CS0}$. Its variable label function is $\nu(T_B) = \text{time}$, i.e., variable $T_B$ stores the execution time of the activity. Given this DPN specification, process trace $\sigma_p = \langle(\text{CS4B}, w_1), (\text{CS1B}, w_2), (\text{CS0B}, w_3)\rangle$ with $w_1(T_B) = 185\ min$, $w_2(T_B) = 194\ min$ and $dom(w_3) = \varnothing$ is valid process behavior. Process trace $\sigma'_p = \langle(\text{CS4B}, w'_1), (\text{CS1B}, w'_2)\rangle$ with $w'_1(T_B) = 185\ min$, $w'_2(T_B) = 196\ min$ is invalid, i.e., $\sigma'_p \notin p$. In trace $\sigma'_p$, an execution of transition $CS0$ is missing. Moreover, it took too long until activity CS1 was executed.

## *2.3. Alignment*

We use alignments to relate event logs to process models. An *alignment establishes a mapping between a log trace $\sigma \in \mathcal{E}$ and a process trace $\sigma_p \in p$.* Events in log traces are mapped to process steps in process traces. It may not be possible to align all events and process steps. Therefore, it is possible to map deviating events and process steps using a special step $\gg$. In case an event is mapped to $\gg$, the alignment could not find a corresponding process step for an event, i.e., the event is missing in the process model. In case a process step is mapped to $\gg$, the alignment could not find a corresponding log event, i.e., the process step was forcefully executed. We denote those pairs as *log move* and *model move*, respectively. Finally, it is possible that the activity name and the event label match, but the values of the process variables and event attributes do not match. We denote such pair as *incorrect synchronous move*.

**Definition 4 (Alignment).** Let $L = (E, \Sigma, \#, \mathcal{E})$ be an event log. Let $\bar{p} = (p, \lambda, \nu)$ be a labeled process model and let $S = (T \times (V \not\rightarrow U))$ be the set of all process steps. A *legal move* in an alignment is a pair $(e, s) \in (E \cup \{\gg\}) \times (S \cup \{\gg\})$ with:

- $(e, \gg)$ is a *log move* iff $e \in E$,
- $(\gg, s)$ is a *model move* iff $s \in S$,
- $(e, s)$ is an *incorrect synchronous move* iff $e \in E \wedge s \in S \wedge s = (t, w) \wedge \#_{act}(e) = \lambda(t) \wedge \exists v \in \mathit{dom}(w) : w(v) \neq \#_{\nu(v)}(e)$,
- $(e, s)$ is a *correct synchronous move* iff $e \in E \wedge s \in S \wedge s = (t, w) \wedge \#_{act}(e) = \lambda(t) \wedge \forall v \in \mathit{dom}(w) : w(v) = \#_{\nu(v)}(e)$.

All other moves are considered illegal, i.e., the move $(\gg, \gg)$ is illegal. We denote the *set of all legal moves* in an alignment as $\Gamma = ((E \cup \{\gg\}) \times (S \cup \{\gg\})) \setminus \{(\gg, \gg)\}$. The **alignment between a log trace $\sigma \in \mathcal{E}$ and a labeled process model** $\bar{p}$ is a sequence $\gamma_{\sigma,\bar{p}} \in \Gamma^*$ such that ignoring all occurrences of $\gg$, the projection of $\gamma_{\sigma,\bar{p}}$ on the first element of each move yields $\sigma$ and the projection on the second element yields a process trace $\sigma_p \in p$. Alignment $\gamma_{\sigma,\bar{p}}$ relates events $e \in \sigma$ to process steps $(t, w) \in \sigma_p$.

There are techniques [5, 6] that search for an *optimal alignment*, which minimizes the deviations between log trace and process trace. Deviations are scored according to a user-defined cost function, which specifies domain knowledge on the reliability of events. An optimal alignment according to the cost function can be seen as the most likely mapping between events and process steps.

**Example 3.** Assume we want to find a mapping between the labeled process model $\bar{p} = (p, \lambda, \nu)$ already introduced in Example 2 and the log trace $\sigma = \langle e_{20}, e_{21} \rangle$. Log trace $\sigma$ contains two of the events introduced in Table 1. Process model $\bar{p}$ prescribes that the transition CS0 has to be executed after the transition CS1. Therefore, $\sigma$ does not fit the behavior defined by the process model. Still, it is possible to align $\sigma$ and $\bar{p}$. One possible alignment is $\gamma_{\sigma,\bar{p}} = \langle (e_{20}, (\text{CSB4}, w_1)), (e_{21}, (\text{CSB1}, w_2)), (\gg, (\text{CSB0}, w_3)) \rangle$ with the recorded attributes $w_1(T_B) = 185\ \mathit{min}$, $w_2(T_B) = 194\ \mathit{min}$ and $\mathit{dom}(w_3) = \varnothing$. According to alignment $\gamma_{\sigma,\bar{p}}$ there is one deviation: transition CS0 was missing, i.e., $(\gg, (\text{CSB1}, w_3))$ is a model move. Alignment $\gamma_{\sigma,\bar{p}}$ is optimal, i.e., no other alignment with less deviations can be build.

## 3. Guided Process Discovery

We present GPD, a method for process discovery that is guided by domain knowledge on the relation between recorded low-level events and high-level activities in the process. We assume that multiple low-level events grouped together indicate the execution of a high-level activity. Moreover, we assume that domain knowledge on the supposed grouping between high-level activities and low-level events can be provided. For example, in Table 1 the execution of the high-level

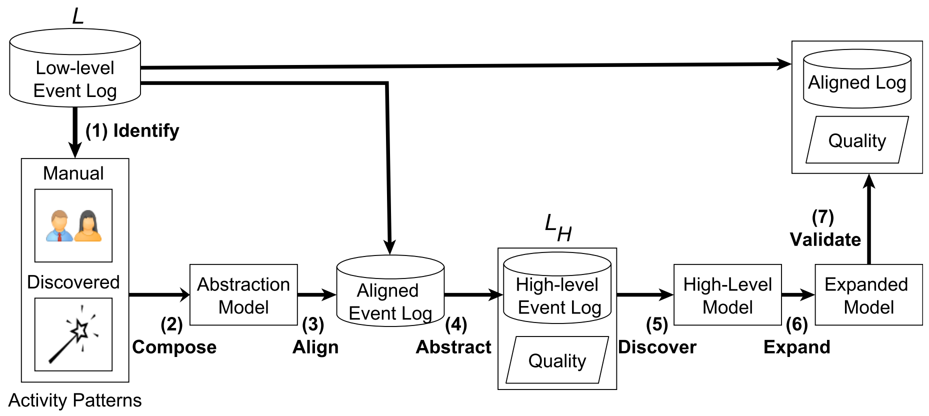


Figure 3: Overview of the proposed GPD method.

activity *Shift* is manifested as sequence of three low-level events *NurseChanged*, *CallSignal1*, and *CallSignal0*.

The input to the GPD method is a *low-level event log* $L = (E, \Sigma, \#, \mathcal{E})$, which contains low-level events recorded during the execution of a process, and *domain knowledge* on the grouping between the recorded low-level events and high-level activities of the process in the form of *activity patterns*. The method consists of the following 7 steps (Figure 3):

1. We *identify and encode multi-perspective activity patterns* that describe elements of high-level behavior as recorded in the low-level events of $L$.

2. We *compose activity patterns* in an integrated *abstraction model*.

3. We *align the abstraction model* with the low-level event log.

4. We *abstract the low-level event log* $L$ to a high-level event log $L_H = (E_H, \Sigma_H, \#^H, \mathcal{E}_H)$ at the desired level of abstraction using the alignment mapping.

5. We *discover a high-level process model* based on the abstracted high-level event log.

In order to validate the quality of the high-level process model, two additional steps can be employed.

6. We *expand activities* in the abstract model with their activity patterns.

7. We *validate the expanded model* against the original event log.

The proposed method can deal with noise, reoccurring and concurrent behavior, and shared functionality. In the following sections, we describe each step of the GPD method in detail.

## *3.1. Identify and Encode Activity Patterns*

We represent knowledge about the relation between low-level events in the event log $L$ and high-level activities $\Sigma_H$ with multi-perspective *activity patterns*. An activity pattern can be seen as description of the high-level activity in terms of the work practice on a fine granularity, i.e., the steps required to execute the high-level activity on a low-level of abstraction. An activity pattern also provides the name (i.e., activity label) of the high-level activity that it describes. We encode activity patterns as labeled process models to allow the specification of complex interactions of the control-flow, time, resource and, data perspective in terms of low-level events. The labeled process model specifies those events that are expected to be seen in the event log for one instance of the corresponding high-level activity.

**Definition 5 (Activity Pattern).** Given a set of transitions $T$, a set of high-level activities $\Sigma_H \subseteq U$, and a set of life-cycle transition $LT$, we define an activity pattern as $ap = (p, \lambda, \nu, hl, lt)$ with:

- $(p, \lambda, \nu)$, is a labeled process model that defines the set process traces that are expected when executing *one instance* of a high-level activity;
- $hl : T \rightarrow \Sigma_H$, a mapping between transitions and high-level activities.
- $lt : T \not\rightarrow LT$, a mapping between transitions and life-cycle transitions.

For each process trace $\sigma_p \in p$, steps $(t, w) \in \sigma_p$ correspond to low-level activities that are expected to be executed as part of the high-level activity $hl(t) \in \Sigma_H$. We denote the set of all activity patterns with $AP$.

Mapping $lt$ is motivated by the observation that activities rarely happen instantaneously. Activities have *life-cycles* [1]. The set of life-cycle transitions $LT$ and the mapping function $lt$ are specified by the user. In the remainder of this paper, we use the start and complete life-cycle transition, i.e., $LT = \{\text{start}, \text{complete}\}$. Moreover, we require that transitions are not shared between activity patterns. For all activity patterns $ap_1 = (p_1, \lambda_1, \nu_1, hl_1, lt_1) \in AP, ap_2 = (p_2, \lambda_2, \nu_2, hl_2, lt_2) \in AP$, if $\{t \in T \mid \exists_{w,\sigma_p}(\sigma_p \in p_1 \wedge (t, w) \in \sigma_p)\} \cap \{t \in T \mid \exists_{w,\sigma_p}(\sigma_p \in p_2 \wedge (t, w) \in \sigma_p)\} \neq \varnothing$ then $ap_1 = ap_2$. Similarly, we require that variables are not shared between activity patterns. Thus, we can uniquely identify to which pattern a process step belongs. This is not limiting: If this condition does not hold, transitions and variables of the activity pattern can be renamed to avoid overlaps in names. Renamed transitions and variables can be linked to the original names by using the activity label function $\lambda$ and the variable label function $\nu$ of the activity pattern. Transitions $t_1, t_2 \in T$ from different patterns may be associated with the same activity name, i.e., $\lambda_1(t_1) = \lambda_2(t_2)$ and variables $v_1, v_2 \in V$ may be associated with the same attribute name, i.e., $\nu(v_1) = \nu(v_2)$.

**Example 4.** Figure 4 shows three activity patterns $ap_a$, $ap_b$ and $ap_c$ that are implemented as DPNs. We use the abbreviated low-level activity name

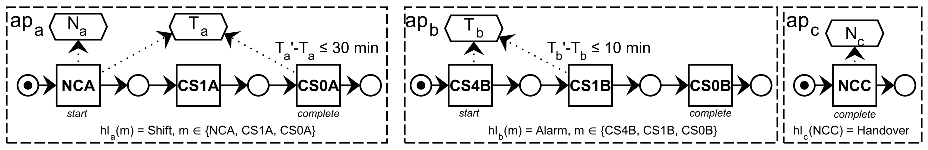


Figure 4: Three activity patterns $ap_a, ap_b, ap_c \in AP$ for the example with process models in DPN notation. The respective label functions $\lambda$ are implicitly encoded in the abbreviated transition names (e.g., $\lambda_a(\text{CS1A}) = \text{CallSignal1}$).

concatenated with the pattern name for transitions. For example, transition CS1A models activity CallSignal1, i.e., $\lambda_a(\text{CS1A}) = \text{CallSignal1}$. We depict the life-cycle transition mapped to a transition in italics below the transition, e.g., $lt_a(\text{NCA}) = \text{start}$. The *first pattern* $ap_a$ describes the high-level activity Shift, i.e., for all transitions $t$ in the pattern we assign $hl(t) = \text{Shift}$. First, the nurse responsible for the patient changes (NCA) and the name of the nurse is recorded in variable $N_a$. Variable $N_a$ is mapped to the attribute nurse, i.e., $\nu(N_a) = \text{nurse}$. Within 30 minutes ($T'_a - T_a \leq 30\ min$ with $\nu(T_a) = \text{time}$), the responsible nurse visits the patient and the call signal system records a button press (CS1A). Finally, the nurse leaves the room and another button press is registered (CS0A) resetting the status. The *second pattern* $ap_b$ describes a similar sequence (i.e., transitions CS1B and CS0B), but represents a different high-level activity: The patient is attended outside of the normal routine. Transition CS4 has to be executed at most 10 minutes beforehand (i.e., $T'_b - T_b \leq 10\ min$). The low-level activity corresponding to CS4B is an *alarm* triggered by the patient. We assign $hl(t) = \text{Alarm}$ for each transition $t$ in the pattern. The *third pattern* describes a simple handover between nurses: Only the responsible nurse changes (NCC) without any consultation of the patient. We assign $hl(t) = \text{Handover}$ for each transition $t$ in the pattern. Transition NCC is an example of shared functionality. Both transitions NCC and NCA are labeled with the same activity name, i.e., $\lambda_a(\text{NCA}) = \lambda_c(\text{NCC}) = \text{NurseChanged}$.

Activity patterns represent the knowledge about how high-level activities are reflected by low-level events in the event log. Please note that we do not expect an activity pattern to be an *exact representation* of every possible way a high-level activity manifests itself in the event log. In fact, in Section 3.5 we show that our method is able to deal with *approximate matches*. Since obtaining suitable activity patterns is crucial for the GPD method, we elaborate here on how to obtain activity patterns. We categorize activity patterns based on the way they have been obtained into: manual patterns and discovered patterns. Table 2 provides a list of sources and examples for activity patterns.

### *3.2. Manual patterns*

Manual patterns are created based on domain knowledge about the high-level activities of the process at hand. We further subdivided manual patterns based on the source of the domain knowledge into patterns based on *expert knowledge*, *process questions*, and *standard models*.

Table 2: Sources for manual and discovered activity patterns.

| **Source** | **Category** | **Examples** |
|---|---|---|
| Expert knowledge | Manual | Patterns $ap_a, ap_b, ap_c$ (Figure 4) |
| Process questions | Manual | Different admission variants (Section 5) |
| Standard models | Manual | Transactional life-cycle model [1] (Figure 5), clinical protocols |
| Local behavior | Discovered | Local process models [7], sub sequences [8], episodes [9], instance graphs [10] |
| Decomposed behavior | Discovered | Region theory [11], clustering [12] |
| Data attributes | Discovered | Discovery per department (Section 5) |

*Expert knowledge.* Manual patterns based on *expert knowledge* encode assumptions on the system. Stakeholders of the process can provide initial assumptions on how high-level activities are manifested in the event log. Moreover, semantically related activities can be grouped together to form sub-processes. If the expected behavior of such a sub-process is known, it can be captured as an activity pattern. The sub-process captured by the activity pattern can be seen as a single activity on a higher level of abstraction. In Example 4 (Figure 4), we describe three activity patterns that are based on expert knowledge. The patterns encode the assumption that the low-level activities CS1 and CS0 both occur in the context of shift change and in the context of an alarm. This knowledge was obtained from a domain expert who is familiar with the system.

*Process questions.* Often, questions on the process can be used as a source for activity patterns. These patterns are not driven by knowledge about the questions, but knowledge about the required type of output. For example, in the evaluation (Section 5) we use an activity pattern that is based on the process question: "What are the trajectories of patients in the hospital based on their admission?". The activity pattern encodes the different variants of the admission.

*Standard models.* Some patterns appear in processes across all domains. Those patterns are based on standard models that are independent of the concrete domain, e.g., the transactional life-cycle model [1]. Discovering such patterns from event logs is challenging for state-of-the-art process discovery algorithms. For example, specialized algorithms exist for event logs with life-cycle information [13]. It is possible to encode the expected behavior as activity patterns and, thus, to avoid the usage of a specialized algorithm for certain patterns. For example, in Figure 5 we show how to adapt the transactional life-cycle model to encode the lifecycle transitions for the high-level activity X-Ray. An X-Ray is scheduled ($x_{scheduled}$), started ($x_{start}$), possibly suspended ($x_{suspended}$), resumed ($x_{resumed}$), and eventually completed ($x_{complete}$). The XES standard [14] for event logs defines an extension for the transactional life-cycle model.

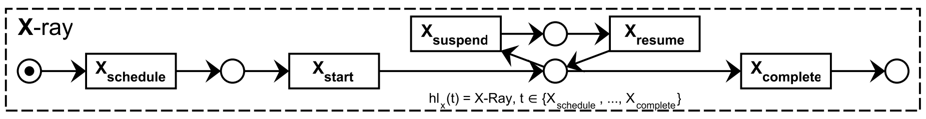


Figure 5: An activity pattern capturing the life-cycle of the high-level activity X-Ray.

### *3.3. Discovered patterns*

It is also possible to automatically discover patterns from the low-level event log. We distinguish between patterns that are discovered based on *local behavior*, based on *decomposed behavior*, and based on *data attributes*.

*Local behavior.* There are dedicated pattern mining techniques [7–10] that discover patterns of local behavior from event logs. Such patterns do not capture the behavior of the complete traces. The models describe subsets of the events and the same event can be part of several patterns. Such discovered local patterns can be directly used as input to the GPD method. It can be challenging to automatically assign good labels to discovered patterns. Methods for automatic labeling of process fragments [15] do exist.

*Decomposed behavior.* Work on decomposed process discovery [11, 12] could be leveraged to obtain activity patterns that represent parts of the observed behavior. Different from methods that discover patterns of local behavior, the event log is decomposed into several disjunct sub-logs using an automated technique. Then, a full process model is discovered for each of the logs using standard process discovery techniques. This process model can be used as activity pattern.

*Data attributes.* Next to automatic decomposition approaches, information can be exploited on the hierarchical structure that is stored in the data attributes of the event log. For example, later in the evaluation (Section 5), we use information on the department in which an event occurred. We split the event log into sub logs based on the department and discover three separate process models that are used as activity patterns.

### *3.4. Compose Activity Patterns to an Abstraction Model*

With a set of activity patterns for the process under analysis at hand, we compose their behavior into an integrated *abstraction model*. The composition of activity patterns may restrict the interaction that is possible between the high-level activities that are captured by the activity patterns. By restricting the interaction, we help to find a more precise mapping between low-level events and high-level activities.

**Definition 6 (Composition Function).** A composition function $f : AP^* \not\rightarrow AP$ combines the behavior activity patterns $ap_1, \ldots, ap_n$ into an (composite) activity pattern $cp \in AP$, i.e., $f(ap_1, \ldots, ap_n) = cp$. We denote with $F = AP^* \not\rightarrow AP$ the set of all composition functions.

We provide the semantics for five basic composition functions: *sequence*, *choice*, *parallel*, *interleaving* and *repetition*. Our abstraction method is not restricted to these functions. Further composition functions can be added.

We introduce some necessary notations for sequences and functions. Given a sequence $\sigma \in S^*$ and a subset $X \subseteq S$, $\mathit{proj}(\sigma, X)$ is the *projection* of $\sigma$ on $X$. For example, $\mathit{proj}(\langle w, o, r, d\rangle, \{o, r\}) = \langle o, r\rangle$. $\sigma_1 \cdot \sigma_2 \in S^*$ *concatenates* two sequences, e.g., $\langle w, o\rangle \cdot \langle r, d\rangle = \langle w, o, r, d\rangle$. Given two functions $f : X_1 \to Y$ and $g : X_2 \to Y$ with disjoint domains, i.e., $X_1 \cap X_2 = \varnothing$, $f \oplus g$ denotes the union of functions $f$ and $g$, i.e., $(f \oplus g)(x) = g(x)$ if $x \in X_1$ and $(f \oplus g)(x) = f(x)$ if $x \in X_2$. Given activity patterns $ap_i = (p_i, \lambda_i, \nu_i, hl_i, lt_i) \in AP$ with $i \in \mathbb{N}$, we introduce the following composition functions:

- **Sequence** composition $\odot \in F$:

$$\begin{aligned} ap_1 \odot ap_2 = (p, \lambda, \nu, hl, lt) \text{ with} & \\ p = \{\sigma \in S^* \mid \sigma_1 \in p_1 \wedge \sigma_2 \in p_2 \wedge \sigma = \sigma_1 \cdot \sigma_2\}, & \text{ and} \\ \lambda = \lambda_1 \oplus \lambda_2, \nu = \nu_1 \oplus \nu_2, hl = hl_1 \oplus hl_2, lt = lt_1 \oplus lt_2. & \end{aligned}$$

  The operation $\odot$ is associative. We write $\bigodot_{1 \leq i \leq n} ap_i = ap_1 \odot ap_2 \odot \ldots \odot ap_n$ to compose ordered collections of patterns in sequence. Moreover, we define $\bigodot_{1 \leq i \leq 0} ap_i = \{\langle\rangle\}$.

- **Choice** composition $\otimes \in F$:

$$\begin{aligned} ap_1 \otimes ap_2 = (p, \lambda, \nu, hl, lt) \text{ with} & \\ p = p_1 \cup p_2, \text{ and} & \\ \lambda = \lambda_1 \oplus \lambda_2, \nu = \nu_1 \oplus \nu_2, hl = hl_1 \oplus hl_2, lt = lt_1 \oplus lt_2. & \end{aligned}$$

  The operation $\otimes$ is commutative and associative. We write $\bigotimes_{1 \leq i \leq n} ap_i = ap_1 \otimes ap_2 \otimes \ldots \otimes ap_n$ to compose sets of patterns in choice.

- **Parallel** composition $\diamond \in F$:

$$\begin{aligned} ap_1 \diamond ap_2 = (p, \lambda, \nu, hl, lt) \text{ with} & \\ p = \{\sigma \in (S_1 \cup S_2)^* \mid \mathit{proj}(\sigma, S_1) \in p_1 \wedge \mathit{proj}(\sigma, S_2) \in p_2\} \text{ and} & \\ \lambda = \lambda_1 \oplus \lambda_2, \nu = \nu_1 \oplus \nu_2, hl = hl_1 \oplus hl_2, lt = lt_1 \oplus lt_2. & \end{aligned}$$

  The operation $\diamond$ is commutative and associative. We write $\Diamond_{1 \leq i \leq n} ap_i = ap_1 \diamond ap_2 \diamond \ldots \diamond ap_n$ to compose sets of patterns in parallel.

- **Interleaving** composition $\leftrightarrow \in F$ with $p(n)$ denoting the set of all permutations of the numbers {1, . . . ,n}:

$$\leftrightarrow (ap_1, \ldots, ap_n) = \bigotimes_{(i_1, \ldots, i_n) \in p(n)} \bigodot_{1 \leq k \leq n} ap_{i_k}.$$

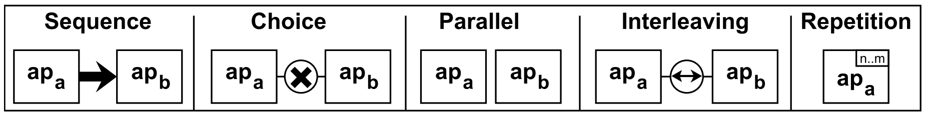


Figure 6: Overview of the graphical notation for the composition functions.

- **Repetition** composition $[n, m] \in F$ with $n \in \mathbb{N}_0, m \in \mathbb{N} \cup \{\infty\}$, and $n \leq m$:
$$ap_1^{[n,m]} = \bigotimes_{n \leq i \leq m} \bigodot_{1 \leq k \leq i} ap_1.$$

We build a composed abstraction model $cp = (p, \lambda, \nu, hl, lt) \in AP$ with a formula that composes all patterns of interest. The process model $p \in P$ corresponds to the overall behavior that we expect to observe for the execution of *all* high-level activities in a single process instance.

**Example 5.** Given the activity patterns $ap_a$, $ap_b$ and $ap_c$ shown in Figure 4, we can compose their behavior to $cp = (\leftrightarrow (ap_a^{[0,\infty]}, ap_b^{[0,\infty]}))^{[0,\infty]} \diamond ap_c^{[0,\infty]}$. We allow indefinite repetitions of all activity patterns using the repetition composition. We allow the absence of patterns using the repetition composition as the corresponding high-level activities might not have been executed in every process instance. We restrict *cp* to only contain the interleaving of patterns $ap_a$ and $ap_b$ as there is only one responsible nurse per patient. Therefore, the activities expressed by $ap_a$ and $ap_b$ can occur in any order but should not happen in parallel. We add $ap_c$ using the parallel composition as handovers can take place in parallel to $ap_a$ and $ap_b$. The result of this composition is the abstraction model *cp*. Model *cp* corresponds to all behavior that could be observed for executions of the three high-level activities. For example, process trace $\langle(\text{NCA}, w_1), (\text{CS1A}, w_2), (\text{NCC}, w_3), (\text{CS0A}, w_4)\rangle$ with $w_1(N_a) = \text{NurseA}$, $w_1(T_a) = 0\ min$, $dom(w_2) = \varnothing$, $w_3(N_c) = \text{NurseB}$, and $w_4(T_a) = 29\ min$, belongs to the set of process traces of the composed abstraction model. Whereas the process trace $\langle(\text{NCA}, w'_1), (\text{CS1A}, w'_2), (\text{CS4B}, w'_3), (\text{CS0A}, w'_4)\rangle$ with $w'_1(N_a)$ $= \text{NurseA}$, $w'_1(T_a) = 0\ min$, $dom(w'_2) = dom(w'_3) = \varnothing$, and $w'_4(T_a) = 29\ min$, is not part of the process behavior of *cp*.

We designed a graphical representation for each composition function, which can be used to design abstraction models in the implementation of our approach. It would also be possible to use BPMN, Petri nets, or Process trees to visualize the composition of patterns. However, abstraction models are not meant to be full specifications of processes. Mostly basic functions, such as the parallel, interleaving, and repetition composition are used. Moreover, process discovery based on the resulting high-level event log is still necessary. Therefore, we use this compact graphical representation.

Figure 6 shows the graphical notation for the proposed composition functions: sequence, choice, parallel, interleaving and repetition. The parallel composition of two patterns does not restrict the interaction between patterns.

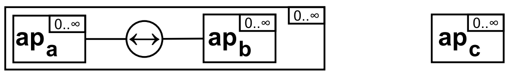


Figure 7: Abstraction model $cp$ created by composing the patterns $ap_a$, $ap_b$, and $ap_c$.

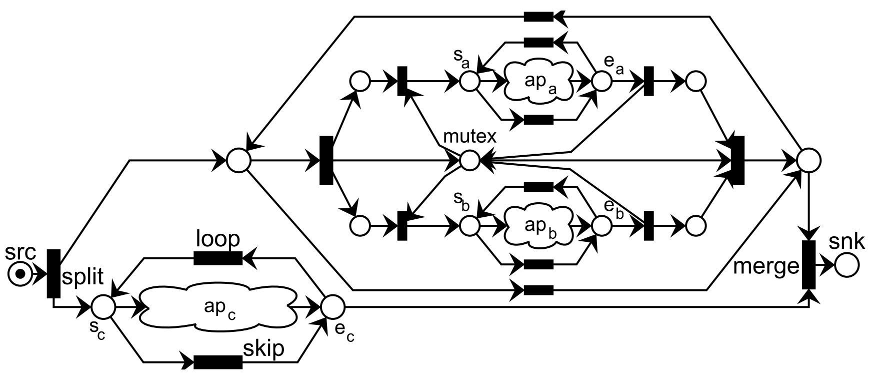


Figure 8: DPN created by our implementation for the abstraction model $cp$. The process models of the activity patterns $ap_a, ap_b, ap_c$ are depicted as clouds $p_a, p_b, p_c$ with source places $s_a, s_b, s_c$ and sink places $e_a, e_b, e_c$. Black transitions are invisible routing transitions, which are not recorded in event logs.

Therefore, unless otherwise specified, we assume that patterns are composed in parallel. The interleaving composition is depicted by connecting interleaved patterns to the interleaving operator $\leftrightarrow$. Patterns are composed in choice by connecting all patterns in choice to a choice operator $\otimes$. The sequence composition is depicted by a directed edge between two patterns. We attach the unary repetition composition directly to the patterns. If necessary, we draw a box around composed patterns to clarify the precedence of operations. Each of the proposed composition functions is implemented using the DPN notation as described in Appendix A. Here, we show it based on an example.

**Example 6.** For example, Figure 7 shows the graphical representation of the composition of activity patterns $ap_a$, $ap_b$, and $ap_c$ to the abstraction model $cp = (\leftrightarrow (ap_a^{[0,\infty]}, ap_b^{[0,\infty]}))^{[0,\infty]} \diamond ap_c^{[0,\infty]}$, which was introduced in Example 5. Patterns $ap_a$ and $ap_b$ are first interleaved and then composed in parallel with $ap_c$. It is straightforward to implemented the composition of activity patterns using the DPN notation. Figure 8 depicts the DPN implementation of the abstraction model $cp$. To simplify the composition, we assume that the DPNs of activity patterns have a single source place and a single sink place. The abstraction model starts with a single source place `src` and ends with a single sink place `snk`. We model the parallel composition of $ap_c^{[0,\infty]}$ with $\leftrightarrow (ap_a^{[0,\infty]}, ap_b^{[0,\infty]})^{[0,\infty]}$ by adding invisible transitions `split` and `merge`, which realize a parallel split and join. Invisible transitions cannot be observed; they are only added for routing purposes. We use place `mutex` to model the mutual exclusion constraint of the interleaving composition of patterns $ap_a^{[0,\infty]}$ and $ap_b^{[0,\infty]}$. Place `mutex`

Table 3: The top three rows show an excerpt of an alignment $\gamma_{\sigma,\bar{p}}$ between the running example log trace $\sigma \in \mathcal{E}$ and the labeled process model $\bar{p}$. Low-level events $e$ are aligned to process steps $(t, w)$ that relate to the low-level activity recorded by the event $(\#_{act}(e) = \lambda(t))$. The bottom five rows show an excerpt of a trace $\sigma_H \in \mathcal{E}_H$ from the high-level event log. Events $\hat{e}$ and the four attributes $\#^H$ are returned by the GPD method.

| $\mathbf{e}$ | $e_{12}$ | $e_{13}$ | $e_{14}$ | … | $e_{20}$ | $e_{21}$ | $\gg$ | … | $e_{30}$ | $e_{31}$ | $e_{32}$ | $e_{33}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\#_{\mathbf{act}}(\mathbf{e})$ | NC | CS1 | CS0 | … | CS4 | CS1 | | … | CS4 | CS1 | NC | CS0 |
| $\mathbf{m}, \mathbf{w}$ | NCA,$w_1$ | CS1A,$w_2$ | CS0A,$w_3$ | … | CS4B,$w_4$ | CS1B,$w_5$ | CS0B,$w_6$ | … | CS4B,$w_7$ | CS1B,$w_8$ | NCC,$w_9$ | CS0B,$w_{10}$ |
| $\#^{\mathbf{H}}_{\mathbf{act}}(\hat{\mathbf{e}})$ | Shift | | Shift | … | Alarm | | Alarm | … | Alarm | | Hando. | Alarm |
| $\#^{\mathbf{H}}_{\mathbf{lc}}(\hat{\mathbf{e}})$ | start | | comp. | … | start | | comp. | … | start | | comp. | comp. |
| $\#^{\mathbf{H}}_{\mathbf{ai}}(\hat{\mathbf{e}})$ | 3 | | 3 | … | 6 | | 6 | … | 11 | | 12 | 11 |
| $\#^{\mathbf{H}}_{\mathbf{time}}(\hat{\mathbf{e}})$ | 122 | | 124 | … | 185 | | 194 | … | 310 | | 312 | 315 |
| $\hat{\mathbf{e}}$ | $\hat{e}_5$ | | $\hat{e}_6$ | … | $\hat{e}_{11}$ | | $\hat{e}_{12}$ | … | $\hat{e}_{21}$ | | $\hat{e}_{22}$ | $\hat{e}_{23}$ |

guarantees that only either $ap_a$ or $ap_b$ can be executed at the same time, yielding the interleaving of $ap_a$ and $ap_b$. Each repetition composition is implemented by adding two invisible transitions `loop` and `skip`, which allow to repeat the pattern indefinitely or to skip its execution, respectively.

## *3.5. Align Event Log and Abstraction Model*

With an abstraction model at hand, we need to relate the behavior in the low-level event log to process traces defined by the abstraction model $cp = (p, \lambda, hl, lt)$. More specifically, we need to determine the mapping between low-level events in traces $\sigma \in \mathcal{E}$ of the event log and process steps in process traces of the labeled process model $\bar{p} = (p, \lambda, \nu)$ defined by the abstraction model. Concretely, we use the *alignment* technique for DPNs presented in [5] to establish alignments $\gamma_{\sigma,\bar{p}} \in \Gamma$ between log traces and process traces. The alignment guarantees that its sequence of model steps without $\gg$-steps is a process trace defined by the composed abstraction model. Please note that we can uniquely identify sub-sequences of the initial (uncomposed) activity patterns in the alignment since we required transitions to be unique among activity patterns.

**Example 7.** The top three rows of Table 3 show an excerpt of such an alignment $\gamma_{\sigma,\bar{p}}$ between the example log trace (Table 1) and a process trace from the example abstraction model (Figure 6). In fact, the third row in Table 3 is a process trace of the process model. Pattern $ap_a$ and pattern $ap_c$ are both executed once. Pattern $ap_b$ is executed twice and. The sub-sequence $\langle(\text{CS4B}, w_4), (\text{CS1B}, w_5), (\text{CS0B}, w_6)\rangle$ with variable assignments $w_4(T_b) = 185$ $min$, $w_5(T_b) = 194$ $min$, and $dom(w_6) = \varnothing$ contains only transitions that are part of pattern $ap_b$. Please note that step $(\text{CS0B}, w_6)$ was inserted as model move by the alignment, i.e., there is no corresponding event in the event log.

## *3.6. Abstract the Event Log using the Alignment*

We describe how to build the high-level event log $(E_H, \Sigma_H, \#^H, \mathcal{E}_H)$ using an alignment of the low-level event log with the abstraction model. In general, there might be scenarios where one event could be mapped to several activity instances. We simplify the discussion by assuming that events are only mapped to single activity instances. This is not a limitation, as described by Baier et al. [3]: Those events could be duplicated in a pre-processing step beforehand.

### *3.6.1. Abstraction Algorithm*

The bottom four rows of Table 3 show how we obtain the high-level event log from the information provided by the alignment. We align each trace $\sigma \in \mathcal{E}$ of the low-level event log with the abstraction model. Doing so, we obtain an alignment $\gamma_{\sigma,\bar{p}}$ as shown in the first three rows for each trace in the low-level log.

**Algorithm 1:** Abstraction of an event log based on an abstraction model.

```
Input: Low-level Event Log $L = (E, \Sigma, \#, \mathcal{E})$, Abstraction Model
       $cp = (p, \lambda, \nu, hl, lt) \in AP$
Result: High-level Event Log $L^H = (E_H, \Sigma_H, \#^H, \mathcal{E}_H)$
$E_H \leftarrow \{\}$, $\Sigma_H \leftarrow hl\,[\Sigma]$, $i \leftarrow 0$
for $h \in \Sigma_H$ do $ai(h) \leftarrow 0$
for $\sigma \in \mathcal{E}$ do
    $\gamma \leftarrow computeOptimalAlignment(\sigma, cp)$
    $\sigma_H \leftarrow \langle\rangle$
    for $(e, s) \in \gamma$ s.t. $s \neq \gg \wedge s = (t, w) \wedge t \in dom(lt)$ do
        $E_H \leftarrow E_H \cup \{\hat{e}\}$
        $\sigma_H \leftarrow \sigma_H \cdot \langle\hat{e}\rangle$
        $assignAttributes(\hat{e}, (e, (t, w)), \gamma, ai, hl, lt)$
        if $lt(t) = complete$ then $ai(hl(t)) \leftarrow ai(hl(t)) + 1$
        $i \leftarrow i + 1$
    end
    $\mathcal{E}_H \leftarrow \mathcal{E}_H \cup \{\sigma_H\}$
end
return $(E_H, \Sigma_H, \#^H, \mathcal{E}_H)$
```

We build the high-level event log based on the alignment and the mapping functions *hl* and *lt* of the abstraction model *cp* as specified by Algorithm 1. Function *hl* obtains the name of the high-level activity for a transition. Function *lt* is used to decide for which transition a new event needs to be created. New high-level events are added to $E_H$ for those alignment moves $(e, (t, w))$ for which the transition $t$ is mapped to a life-cycle transition, i.e., in our case either *start* or *complete*. Those transitions correspond to visible transitions in the life-cycle of the high-level activity.

Algorithm 2 describe how we assign each new high-level event a name based on the mapping $hl(t)$, a *unique activity instance identifier* (i.e., as defined the XES standard [14]) for each execution of an activity pattern, and the life-cycle transition obtained from the mapping $lt(t)$. Moreover, we copy the values of all variables $v \in dom(w)$ to the attributes of the new event. Hence, low-level attributes can be transformed to high-level attributes. In this manner, we create a high-level trace in $\mathcal{E}_H$ for each low-level trace in $\mathcal{E}$. Algorithm 1 provides us with the abstracted high-level log $(E_H, \Sigma_H, \#^H, \mathcal{E}_H)$.

**Example 8.** For example, events $\hat{e}_5$ and $\hat{e}_6$ in Table 3 are created based on the alignment of low-level events $e_{12}$ and $e_{14}$ to transitions NCA and CS0A. We

**Algorithm 2:** Procedure *assignAttributes*, which assigns the attributes of event $\hat{e}$ that was created for the execution of a high-level activity.

```
Input: Event $\hat{e}$, Alignment move $(e,(t,w))$, Alignment $\gamma$, Instance mapping $ai$,
       High-level activity mapping $hl$, Life-cycle mapping $lt$
$\#^H_{act}(\hat{e}) \leftarrow hl(t)$                          // assign high-level activity name
$\#^H_{ai}(\hat{e}) \leftarrow ai(hl(t))$                        // assign activity instance
$\#^H_{lc}(\hat{e}) \leftarrow lt(t)$                           // assign life-cycle transition
for $v \in dom(w)$ do $\#^H(\hat{e})(\nu(v)) \leftarrow w(v)$              // copy variables
if $e \in E$ then
    $\#^H_{time}(\hat{e}) \leftarrow \#_{time}(e)$                        // assign timestamp
else
    $\#^H_{time}(\hat{e}) \leftarrow obtainTime(\gamma,(e,(t,w)))$
end
```

assign event $\hat{e}_5$ the high-level activity name Shift, i.e., $\#^H_{act}(\hat{e}_5) =$ Shift. We assign the unique activity instance identifier 3 to both events, i.e., $\#^H_{ai}(\hat{e}_5) = \#^H_{ai}(\hat{e}_6) = 3$. Instance 3 of the high-level activity Shift was started by event $\hat{e}_5$ and completed by event $\hat{e}_6$. Then, we assign the life-cycle transition start to $\hat{e_5}$ (i.e., $\#^H_{lc}(\hat{e}_5) =$ start) and the life-cycle transition complete to event $\hat{e_6}$. Finally, we copy the values of the variables $N_A$ and $T_A$ as $\#^H_{nurse}(\hat{e_5}) =$ NurseA and $\#^H_{time}(\hat{e_5}) = 122$ min.

We ensure that every high-level event $\hat{e} \in E_H$ is assigned a *timestamp.* We consider two cases depending on the alignment move $(e,(t,w))$: (1) The process step was aligned to a low-level event $e$ and (2) the process step was mapped to $e = \gg$, i.e., a model move. In the first case, we assign the timestamp of the aligned low-level event to the high-level event. For example, $\#^H_{time}(\hat{e}_{11}) = \#_{time}(e_{20}) = 185\ min$. In the second case, there are multiple possible methods to determine the most likely timestamp for a model move (e.g., based on statistical methods [16]). Therefore, we abstract from the concrete implementation with the method *obtainTime*. For the case study, we used the timestamps of neighboring low-level events that are mapped to the same activity instance. For example, for the unmapped high-level event $\hat{e}_{12}$, we use the timestamp from the neighboring event $e_{21}$, i.e., $\#^H_{time}(\hat{e}_{12}) = \#_{time}(e_{21}) = 194\ min$.

### *3.6.2. Quality of the Abstraction*

The alignment enables the definition of two quality measures for the abstraction mapping. First, we use **global matching error** $\epsilon_{L,cp} \in [0,1]$ as a measure for how well the entire low-level event log $L$ matches the behavior imposed by the composed abstraction model *cp*. In this context, a fitness measure such the one defined in [5] for alignments of DPNs can be seen as global measure for the quality of the used abstraction model. A low fitness indicates that there are many events that cannot be correctly matched and, thus, the abstraction model does not capture the whole process correctly. The resulting abstracted event log would not allow a reliable analysis. In case of low fitness, the assumptions made in the identification and encoding of the activity patterns should be revised.

Second, we define a **local matching error** $\epsilon_{L,cp}(h) \in [0,1]$ on the level of

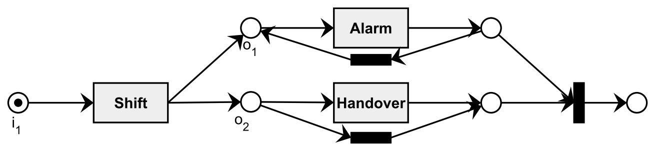


Figure 9: A discovered high-level model in DPN notation. We distinguish high-level activities from activities used in activity patterns by using a gray background.

each recognized high-level activity. Some process steps in the alignment are not matched to an event in the log, i.e., the event is missing. To obtain $\epsilon_{L,cp}(h)$, we determine the fraction of incorrect and model moves for process activities that are part of the activity pattern for the high-level activity $h$ over the total number of all alignment moves for $h$.

**Definition 7 (Local matching error).** Let $L = (E, \Sigma, \#, \mathcal{E})$ be a low-level event log. Let $cp = (p, \lambda, \nu, hl, lt) \in AP$ be an abstraction model with the labeled process model $\bar{p} = (p, \lambda, \nu)$. We define the matching error $\epsilon_{L,cp}(h) \in [0, 1]$ as the fraction of incorrect and model moves over the total number of all moves for process activities that are mapped to $h$ in the alignment $\gamma_{\sigma,\bar{p}}$:

$$\begin{aligned}\epsilon_{L,cp}(h) &= \frac{\sum_{\sigma \in \mathcal{E}} |proj(\gamma_{\sigma,\bar{p}}, \Gamma_h^{err})|}{\sum_{\sigma \in \mathcal{E}} |proj(\gamma_{\sigma,\bar{p}}, \Gamma_h)|}, \text{ where}\\ \Gamma_h &= \{(e,s) \in \Gamma \mid s \neq \gg \wedge\, s = (t,w) \wedge hl(t) = h\}, \text{ and}\\ \Gamma_h^{err} &= \{(e,s) \in \Gamma_h \mid (e,s) \text{ is a model move or}\\ &\qquad\qquad (e,s) \text{ is an incorrect synchronous move}\}.\end{aligned}$$

The matching error can be used to exclude individual unreliable activity pattern matches, which exceed a certain $\epsilon$-threshold.

**Example 9.** For example, in Table 3 one execution of transition CS0B in the activity pattern for the Alarm activity is mapped to $\gg$. The local matching error $\epsilon_{L,cp}(\text{Alarm})$ is $\frac{5}{6}$ for high-level activity Alarm based on the alignment shown in Table 3.

### *3.7. Discover a High-Level Process Model*

After creating the abstracted event log $L_H$, we discover a process model based on the abstracted high-level activities $\Sigma_H$. For the process discovery, any state-of-the-art process discovery technique can be employed. However, as the abstracted event log contains information on the life-cycle of activities (i.e., the *start* and the *complete* transition), we propose to use a process discovery technique that can harness this information, such as the Inductive Miner [13]. We could use the discovered process model directly for further analysis (e.g., performance analysis, deviation analysis, decision rule mining etc.).

**Example 10.** Figure 9 shows an example of a high-level model in DPN notation that can be discovered based on the abstracted event log. The discovered process starts with a change of shifts. Then, in parallel, patients raise one or more alarms and, sometimes, a handover takes place. In fact, in Table 3 the Alarm and the Handover activity are observed to be overlapping each other. An instance of the Handover instance (event $\hat{e}_{22}$) is interleaved between the start (event $\hat{e}_{21}$) and the complete (event $\hat{e}_{23}$) of an Alarm activity instance.

## *3.8. Expand the High-Level Activities and Validate Against the Event Log*

The abstracted event log $L_H$ hides details on the low-level events. High-level events in $L_H$ might have been introduced by the approximate abstraction mapping that was obtained through the alignment. As we allow log moves and model moves in the alignment, some parts of the abstracted event log might be based on our assumptions on the process rather than the actual data in the original, low-level event log. We can quantify this error by using the previously introduce measures *fitness* and *matching error*. However, even a small error during the abstraction can be multiplied by the discovery algorithm. In other words, we might have misguided the discovery and resulting process model does reflect our assumption rather than reality. This is clearly undesirable.

To evaluate the quality of the discovered high-level model, we generate a so-called *expanded process model*. We substitute each high-level activity with the DPN representation associated with the activity. In fact, the high-level activities in the discovered process model can be seen as composite activities and the activity patterns as sub-processes. This step depends on the concrete modeling notation. The expansion of high-level activities with sub-processes cannot be described using solely the language of the process model. When defining a process model through its language, it is not possible to distinguish whether two high-level activities $a$ and $b$ are executed in parallel (i.e., the execution of low-level activities in the sub processes of $a$ and $b$ can overlap), or their execution is only interleaved (i.e., we observed both $\langle a, b\rangle$ and $\langle b, a\rangle$ but the activities are not overlapping). Since the steps are similar in each modeling notation, we focus on the case in which DPNs are employed. For the concrete approach described here, we assume that the discovered model and the activity patterns are provided as DPN (e.g., as in Figure 9).

The mapping from high-level activities to activity patterns can be obtained automatically through function $hl$. Each pattern is mapped to exactly one high-level activity. We replace each transition with the entire corresponding activity pattern. Figure 10 illustrates the expansion of a high-level activity $a$ with the process model $p_a$ of the associated activity pattern $ap_a$. Again, we assume that activity patterns are provided as DPNs with a single source and a single sink place (i.e., workflow nets). With the help of two invisible routing transitions $s$ and $c$, we connect the source place $src$ and sink place $snk$ of the activity pattern to the input places $i_1, \ldots, i_n$ and output places $o_1, \ldots, o_n$ of the replaced high-level activity. The resulting expanded process model describes the behavior of the discovered model in terms of low-level events, i.e., each high-level activity is replaced with a sub process that captures its behavior on a lower abstraction level.

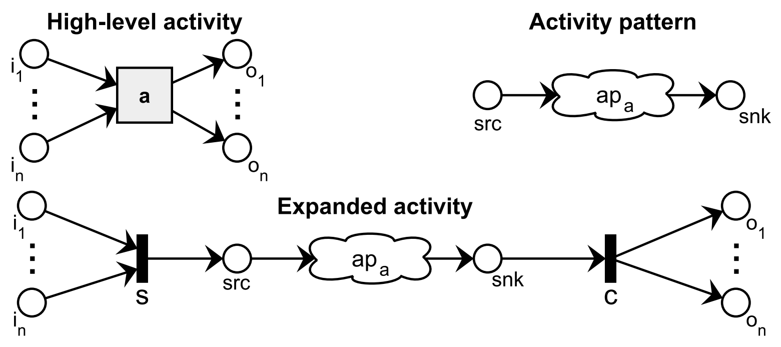


Figure 10: Expansion of a single high-level activity $a$ with input places $i_1, \ldots, i_n$ and output places $o_1, \ldots, o_n$ in the discovered model with the DPN modeling the activity pattern $ap_a$.

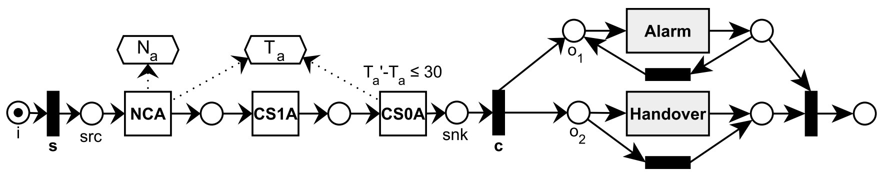


Figure 11: A partially expanded high-level model. The high-level activity Shift in the high-level model has been replaced by the DPN that models the activity pattern $ap_a$.

**Example 11.** Figure 11 shows a partially expanded high-level model. We replaced high-level activity Shift in the model shown in Figure 9 by the DPN that models the activity pattern $ap_a$. Routing transitions $s$ and $c$ have been added. Transition $s$ is connected to input place $i$ of high-level activity Shift and to the source place *src* of the activity pattern. Transition $c$ is connected to the only sink place *snk* of the activity pattern and to both output places $o_1$ and $o_2$ of the high-level activity Shift. The invisible routing transition $s$ is not strictly necessary. The activity pattern has a single source place and the high-level activity is connected to a single input place and, thus, we can simplify the expanded model by fusing places $i$ and *src* and removing transition $s$. We use standard reduction rules for this [17].

We can now validate the quality of the expanded model against the low-level input event log. We use existing conformance checking techniques (e.g., alignment-based techniques [5, 6]) that determine the quality (e.g., fitness) of a process model given an event log. We need to validate the model against the original input event log rather than against the abstracted event log. Otherwise, the validation would be biased by the domain knowledge that we introduced in the abstraction. However, when measuring the quality of the expanded model against the original event log, the result is independent from the domain knowledge assumed by using the activity patterns.

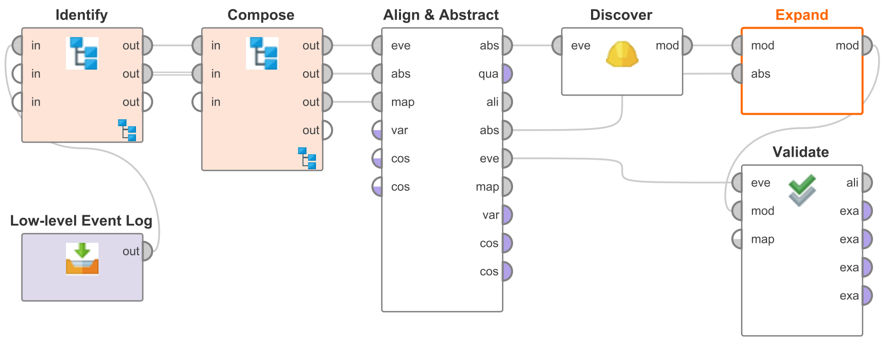


Figure 12: A RapidMiner workflow that implements the GPD method. The operators Identify and Compose are sub-processes, in which the activity patterns are imported or discovered and composed with an operator that implements the composition functions.

## 4. Implementation

Our method is implemented both in the open-source process mining framework ProM and as set of operators in the RapidMiner extension RapidProM [18].[2] Figure 12 shows a workflow in RapidProM that implements the entire GPD method.[3] First, the original event log is imported from a XES file. Then, we import manually created patterns and use discovery method (e.g., Inductive Miner) to obtain discovered patterns in the sub-process *Identify* (i.e., step 1 of the method). All patterns are composed in the *Compose* sub-process. The *Compose* operator implements step 2 of the method and requires a collection of process models and a composition formula as input. By default all patterns are composed in parallel. We used a sub process for both steps to ensure that the workflow fits in a figure. We apply the *Align & Abstract* operator, which implements steps 3 and 4 of the method. The required inputs are the event log, the abstraction model and a mapping between the event and transition labels. Afterwards, the standard Inductive Miner operator is used to discover a high-level process model (step 5). This process model is, then, expanded with the new *Expand* operator. Finally, we use the standard conformance checking capabilities of RapidProM to measure the fitness of the expanded model.

## 5. Evaluation

We evaluated the usefulness of our proposed GPD method by using two real-life event logs and its efficiency by using a series of synthetic event logs for perfor-

[2]The ProM package LogEnhancement can be downloaded under `http://promtools.org` and the source code of the RapidProM extension under `https://github.com/rapidprom/rapidprom-source/tree/fmannhardt`.

[3]The RapidProM workflow can be downloaded under: `https://svn.win.tue.nl/repos/prom/Packages/LogEnhancement/Trunk/data/gpd-workflow.rmp`

mance evaluation. The first event log was obtained from a digital whiteboard system used to support the daily work of nurses in the observation unit of a Norwegian hospital. We used the whiteboard to evaluate the abstraction part of the GPD method (i.e., until step 4 in Figure 3). We created an abstraction model with 18 manual activity patterns. Three of these patterns were already used as running examples before: $ap_a, ap_b$, and $ap_c$ in Figure 4. Based on the abstracted event log that was created by applying our method until step 4, we gathered insights into the usage of the digital whiteboard system. In this paper we do not further elaborate on the evaluation of the abstraction method on the whiteboard event log, as it was already reported in [19].

In Section 5.1, we evaluate the usefulness of the full GPD method by applying it on a new real-life event log that contains events on the management of sepsis patients in a different hospital from The Netherlands. In Section 5.2, we perform a controlled experiment to evaluate the performance of the proposed method. Finally, in Section 5.3 we acknowledge its limitations.

## *5.1. Application on the Sepsis Process*

The process entails the trajectory of patients with a suspicion for sepsis, a life-threatening condition typically caused by an infection [20], within the hospital. We analyzed this process as part of a project carried out together with the emergency department and the data analytics team of the hospital.

First, we provide more details on the event log that we used (Section 5.1.1) and report on the process questions that we identified together with the stakeholders from the hospital (Section 5.1.2). Then, we describe the identified activity patterns (Section 5.1.3) and show the results that could be obtained (Section 5.1.4). Finally, we compare our results with those that can be obtained by state-of-the-art methods (Section 5.1.5).

### *5.1.1. Event Log*

We extracted the event log from an Enterprise Resource Planning (ERP) system of the hospital. The ERP systems is used for most activities in the hospital, such as the electronic patient record, laboratory tests, and information on the movement of patients between hospital wards. There are about 1,000 cases and 15,000 events that were recorded for 16 different activities. Moreover, 39 data attributes are recorded, e.g., the department responsible for the activity, the results of blood tests and information from checklists filled in by nurses. The recorded events relate to activities of different categories. Some activities are executed in the emergency department (e.g., checklists, infusions), some activities are related to blood tests, some activities are about the admission and transfer within the hospital, and some activities are about the discharge of patients.

### *5.1.2. Process Questions*

Together with the doctor responsible for the emergency department and a data analyst of the hospital, we identified three process questions that are of interest for the hospital:

1. What are the trajectories of patients depending how they were initially admitted to the hospital? Is there any influence on the remaining process, e.g., does a certain category of patient return more often. Specifically, the hospital is interested in the following three categories: (1) patients that are first admitted to the *normal care ward*, (2) first to the *intensive care ward*, or (3) patients that are first admitted to the *normal care ward* and, only then, to the *intensive care ward*? Each of the categories is of interest to the hospital because the category of the first admission indicates the severity of the sepsis condition. In particular, the third category is of high interest as it indicates that the patient's condition has worsened after being admitted. The hospital is interested in minimizing the number of those patients and in visualizing the effects of this category of patients to the operation of the hospital, e.g., in terms of the outcome and the time that those patients remain in the hospital.

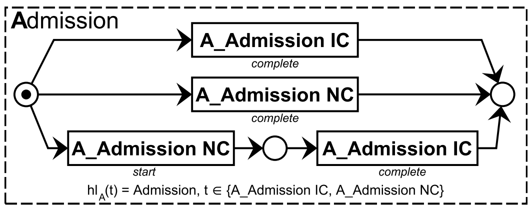
2. Are patients with a sepsis condition given antibiotics and liquid and if so, then, what is the delay from the initial triage until antibiotics are administered in the emergency ward. The rationale for this question is a medical guideline that recommends the administration of antibiotics within one hour after a sepsis condition has been identified.

3. Are there decision rules that can be discovered based on the attributes recorded in the event log? Discovering such rules may lead to insights on the conditions under which patients follow a certain trajectory in the process. It would be interesting for the hospital to know the most likely trajectory of a patient in order to prevent problems from arising early in the process.

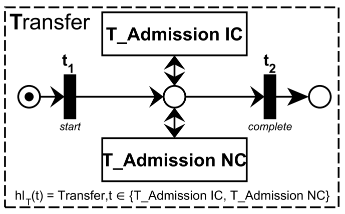
We apply the proposed GDP method and show that the obtained process model is suitable to answer the process questions. Moreover, the discovered process model is comprehensible by stakeholder and, thus, can be used for communication. Finally, we show that state-of-the-art process discovery methods applied directly on the low-level event log provided process models that were unsuitable to answer these questions.

#### *5.1.3. Activity Patterns*

We identified two manual and three discovered activity patterns. The manual patterns are shown in Figure 13(a) and Figure 13(b). Both patterns are based on the activities Admission NC and Admission IC, which relate to the admission of a patient to an intensive care or normal care ward, respectively. We designed the manual patterns together with the data analyst of the hospital based on the first process question that was articulated by the doctor. Activity pattern Admission (Figure 13(a)) encodes the three admission variants, which were also encoded in a similar manner in a flowchart-like process documentation provided by the emergency department. We deliberately duplicated the activities to encode the problematic third variant of admission that is of great interest to the doctor. A patient may further be transferred between the different wards of



(a) Admission pattern that models three different variants of an admission.



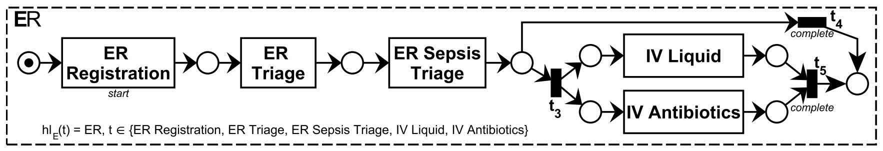
(b) Transfer pattern that models a series of transfers.

Figure 13: Two manual patterns that were created for the sepsis event log. The respective activity mapping is defined by removing the prefixes A_ or T_ from the transition name (e.g, $\lambda_A$(A_Admission IC) = Admission IC).

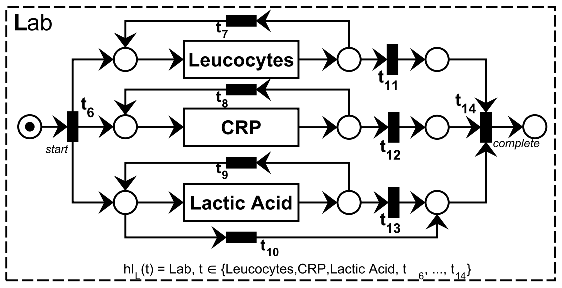
the hospital. Therefore, there may be subsequent events that record one of the low-level admission activities. Those transfers are not of interest to the emergency department. Therefore, we added activity pattern Transfer (Figure 13(b)), which encodes that any number of subsequent transfers may occur.

Moreover, we obtained three discovered patterns based on information on the organizational perspective that is part of the event log. We extracted three sub logs: Each log contained all activities performed by employees of a certain department. Then, we discovered a process model for each of the sub logs using the Inductive Miner (Figure 14). Activity pattern ER (Emergency Department) is shown in Figure 14(a). All activities in pattern ER are executed in the emergency department (departments A and C in the event log). Therefore, we denote this pattern as ER. First, the patient is registered and triage checklists are filled. Then, antibiotics and liquid infusions can be given. Figure 14(b) shows activity pattern *Lab* discovered for department B, clearly this is the laboratory department responsible for blood tests. Figure 14(c) is based on a sub log obtained for department E and contains five different activities that relate to different (anonymized) variants on how patients are discharged.

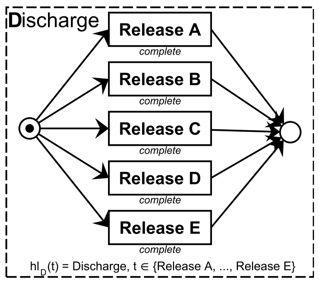
We composed six activity patterns[4] into an abstraction model (Figure 15). Here, we used the composition functions to add the constraint that a Transfer can only occur after an Admission has taken place. Clearly, patients need to be admitted before they can be transferred. Overall, we used only basic domain knowledge on the process and organizational information taken from the event log to build an abstraction model.



(a) ER pattern discovered for departments A and C.



(b) Lab pattern discovered for department B.



(c) Discharge pattern discovered for department E

Figure 14: Three discovered patterns that were obtained by splitting the log based on the department attribute and using the Inductive Miner on the resulting sub logs.

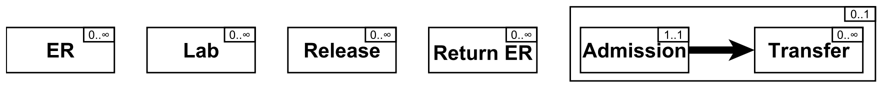


Figure 15: Abstraction model used for the case study. We added the restriction that the high-level activity Transfer can only occur after the high-level activity Admission.

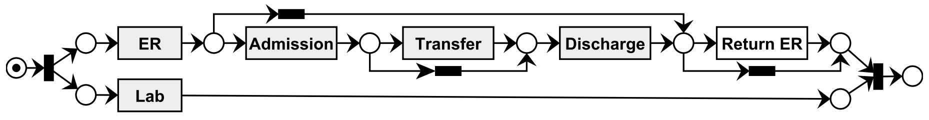
### *5.1.4. Results of the GPD Method*

We created an abstracted high-level event log with the abstraction model shown in Figure 15. The abstracted event log has about 8,300 events for the six high-level activities. The abstracted event log could be computed in less than 2 minutes using 2 GB of memory. The global matching error of the abstraction model was $\epsilon = 0.02$. Only for pattern ER a non-zero local matching error $\epsilon(\text{ER}) = 0.006$ was recorded, i.e., all other patterns match perfectly.

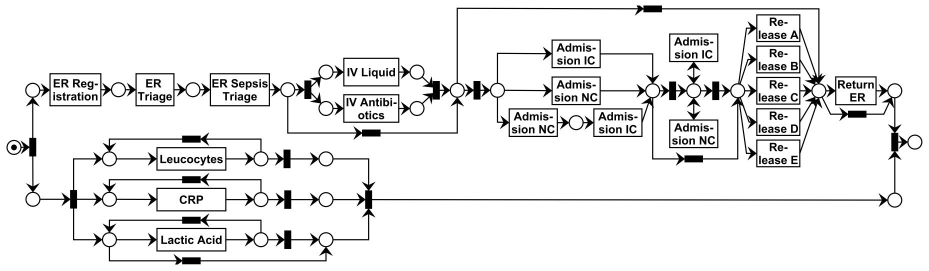
Next, we discovered the guided process model shown in Figure 16(a) on the abstracted event log using the Inductive Miner[5]. This process model describes

[4]The sixth activity pattern for the activity *Return ER* consists only of the activity *Return ER* itself, i.e., this activity can be left unchanged by our method.

[5]We used the Inductive Miner infrequent with a noise threshold of 0.2 (the default setting).



(a) High-level guided Petri net



(b) Expanded guided Petri net

Figure 16: High-level and expanded Petri net discovered using IM when applying the GPD method. Gray transitions are abstracted high-level activities.

the trajectory of a patient on a high level of abstraction. Results of blood tests are obtained during the whole process. First, patients are in the emergency room. Then, patients are either admitted to a hospital ward or they leave the hospital. Admitted patients are possibly transferred to another hospital ward and eventually discharged. Finally, patients may return to the emergency room at a later time. The process model matches the high-level description of the process by stakeholders from the hospital that we obtained beforehand.

We expanded the high-level activities of the guided process model with the corresponding activity patterns as described in Section 3.8. Figure 16(b) shows the resulting expanded process model. We validated the quality of the discovered process model by measuring the average fitness (0.97) of the expanded process model with regard to the original low-level event log. The process model fits most of the behavior seen in the event log. Thus, we can use the expanded process model to reliably answer the three initially posed process questions.

*Category of Admission.* The first question on how patients are initially admitted to the hospital wards can be answered using the expanded process model. We projected the low-level event log on the process model using the Multi-perspective Process Explorer (MPE) [21]. Figure 17 shows the output of the MPE. We could determine that 2.3% of the admitted patients are of the problematic category: They are first admitted to the normal care ward and, then, re-admitted to the intensive care ward. Around 86.2% of the patients are ad-

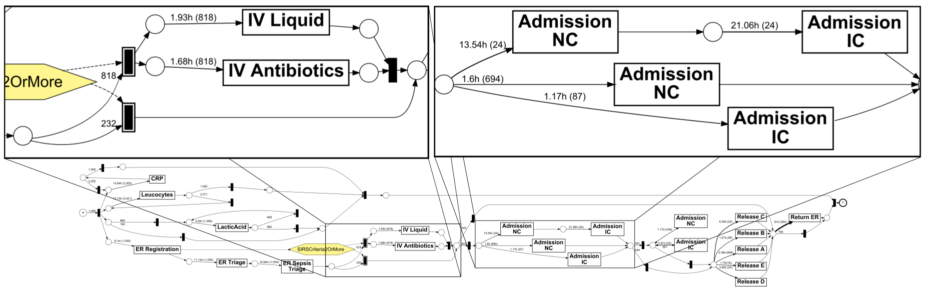


Figure 17: Performance information and a decision rule projected on the expanded model discovered for the sepsis event log.

mitted to the normal care ward and 10.9% of the patients are admitted to the intensive care ward. Moreover, we used the filtering capabilities of the MPE to visualize only the trajectories of the problematic patients. This revealed that 56.5% of those patients return to the emergency room within one year (i.e., activity Return ER). Among the other patients only 27.4% return. This indicates that the problematic category of patients should, indeed, be monitored more closely.

*Infusions.* We used the discovered process model to investigate whether antibiotics and liquid infusions are given to patients with a sepsis condition. There are several criteria that are checked to determine a sepsis. The event log contains the attribute *SIRSCriteria2OrMore*, which indicates whether two or more of these criteria are fulfilled. We used the MPE to retain only cases with *SIRSCriteria2OrMore*=true and projected those cases on the expanded process model. This revealed that 95.3% of those patients eventually get an antibiotics infusion. However, according to the event log 15% of those patients do not receive a infusion of liquid (i.e., the alignment contains a model move for low-level activity IV Liquid). Then, we projected the average time between activities in the entire event log on the process model. This revealed that it takes, on average, 1.68 hours until the antibiotics are administered (Figure 17). When reporting both findings to the hospital, we found that data about the infusions is entered manually into the ERP system. Therefore, it is unclear whether the average time represents the real waiting time. Moreover, missing liquid infusions are, most probably, simply not registered.

*Decision Rules.* We also applied our decision mining techniques [22] to discover decision rules for the decision points in the expanded process model. We discovered that it depends on the attribute *SIRSCriteria2OrMore* whether patients receive infusions. This can be expected as patients with more than two criteria for a sepsis should definitely receive infusions. We also discovered decision rules regarding the three different variants of admission. We found rules for the admission of patients to the normal care and intensive care. However, we did not find a good rule for the problematic category based on the attributes in the event log.

Overall, using the process model shown in Figure 16(b) we could provide meaningful answers to the process questions. Moreover, it was possible to use the discovered process model to communicate with stakeholders. By guiding the process discovery with activity patterns, we show that it is possible to discover a model that is comprehensible to domain experts.

### *5.1.5. Comparison to State-of-the-Art Methods*

To assess the usefulness of our GPD method, we compared the insights that can be obtained from the expanded model shown in Figure 16(b), which was obtained by applying the GPD method and Inductive Miner, with the models that were solely discovered by state-of-the-art methods on the same abstraction level. Thus, in this comparison we ignore that the GPD method additionally provides a high-level process model. We applied the following four discovery methods directly to the original low-level event log: the Heuristics Miner (HM) [23], the ILP Miner [24], the Inductive Miner (IM) [13], and the Evolutionary Tree Miner (ETM) [25].

*Heuristics Miner.* The process model discovered by the HM[6] is unsound, it contains unbounded behavior that prevents the process from completing. Process model that are unsound are not suited for a wide range of analysis tasks [1]. Therefore, we could not use the process model discovered by the HM.

*ILP Miner.* The process model discovered by the ILP Miner[7] is fitting the event log well (fitness 0.80), but it is very complex with several non-free choice constructs. This made it impossible to explain it to the stakeholders (i.e., doctor and data analyst) of the hospital. Thus, the model is unsuitable in our case.

*Evolutionary Tree Miner.* The process model discovered by the ETM[8] after running for 100 generations (fitness 0.84) was complex and was missing the infrequently occurring activities Admission IC, Release B, Release C, and Release E. Since those activities are an important part of the process this model could not be used.

*Inductive Miner.* Figure 18 shows the process model discovered by the IM[9]. Among the state-of-the-art methods, we consider this the best model for our purposes. Still, note how difficult it is to use this process model to directly answer any of the process questions. The model does have a comparable average fitness (0.99), but it fails to properly reflect the structure of the process. Semantically related activities are not grouped together since the IM does not take the

[6]We used the ProM 6.7 plug-in *Mine for a Heuristics Net using Heuristics Miner.*

[7]We used the ProM 6.7 plug-in *ILP-Based Process Discovery (Express).*

[8]We used the ProM 6.7 plug-in *Mine Pareto fron with ETMd.*

[9]We used the same parameter settings as when using the IM together with the GPD method.

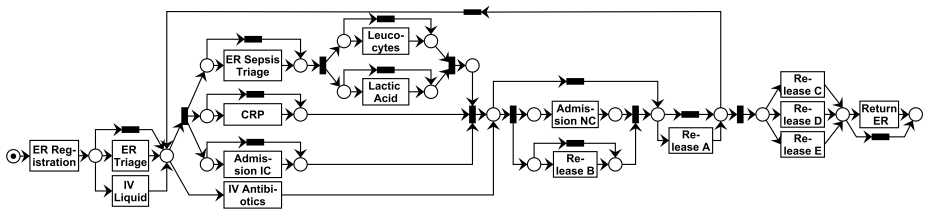


Figure 18: Unguided Petri net discovered using IM without applying the GPD method.

organizational information and the domain knowledge on the admissions into account. For example, antibiotics and liquid infusions are placed on different decision points and the blood tests are placed within the main process flow. Moreover, it is possible to repeat most of the process after the two discharge activities Release A and Release B occurred. We know from the stakeholders that administering antibiotics is not repeated in the context of the treatment in the emergency room. Based on the model in Figure 18 it is impossible to answer the first question on the problematic category of patients. Similarly, it is difficult to answer the second question on the antibiotics and liquid infusions as the process model does not contain a decision point for the infusions. The application of decision mining (i.e., the third question) requires suitable decision points to be present in the process model. The decisions modeled in Figure 18 are on a very low level of abstraction, i.e., on the level of skipping a single low-level activity. Therefore, we were not able to find the decision rules described in the previous paragraph. To conclude, the unguided process model is clearly not suited for our type of analysis.

## *5.2. Evaluation of the Efficiency of the Method*

The computation time is dominated by the alignment computation in step 3 of the method. Computing an optimal alignment has exponential worst-case complexity [5]. The abstraction itself in step 4 can be computed in time linear to the size of the input trace. Therefore, we evaluated the efficiency of our method by computing alignments for several models and traces. All experiments have been conducted on a standard laptop with 16 GB of memory. We tested the computation time for a set of eight randomly generated process models (25–263 transitions) and event logs (6–340 events per trace) that were previously used in [26]. We decomposed each model based on the method described in [27] to obtain between 3 and 25 activity patterns for each model. Then, we compared the performance of using the parallel and the interleaving composition of all activity patterns for increasing levels of noise (10%–30% of swapped events as described in [26]) that was injected into the event log.

Figure 19 shows the resulting average computation time per trace of our method when applied to each of the event logs. We limited the computation time to 100 seconds (100,000 ms) since we consider this to be a feasible computation time in practice. As expected, the computation time grew exponentially both with increasing length of the traces as well as with the increasing number of

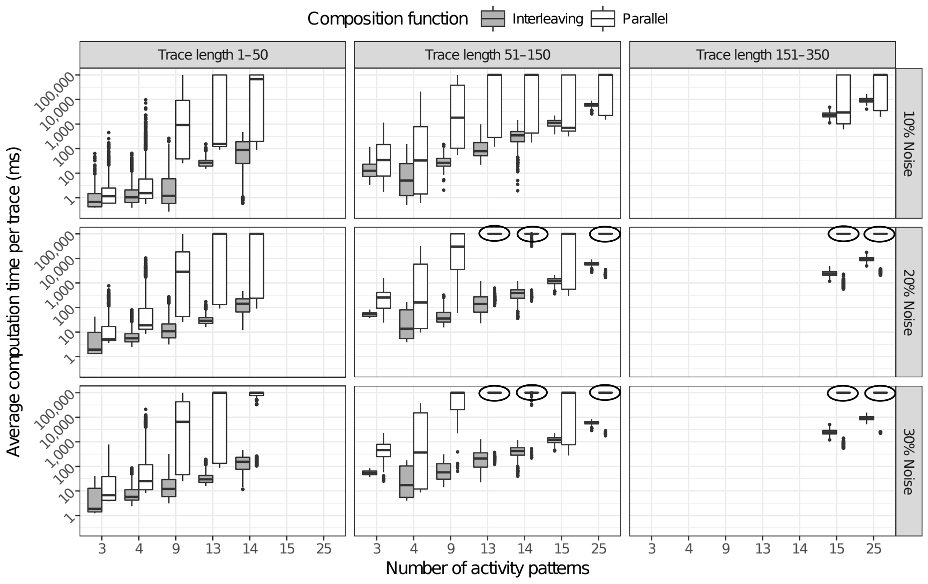


Figure 19: Average computation time per trace of the alignment used in the GPD method.

activity patterns. Moreover, using the parallel composition lead to a worse performance than using the interleaving composition, which is expected due to the large state-space that needs to be explored in case of parallel branches. In comparison, the increasing level of noise had little influence on the computation time.

Overall, the experiment showed that it is feasible to use the GPD method with up to 25 activity patterns and traces of up to length 350 on occasion of the interleaving composition. When composing all of the patterns in parallel, the computation finished within the maximum of 100 seconds in the cases when less than 13 activity patterns and an event log with less than 20% noise were tested. Conversely, the computation did not finish within 100 seconds for some traces when using the parallel composition. In situations with 13 or more activity patterns, more than 10% noise, or traces that are longer than 150 events our method would have required a longer computation time. Readers should however notice that a composition configuration with all patterns in parallel is the worst case that rarely happens. Usually, an abstraction model is built using a combination of the available composition patterns (cf. Figure 15).

## *5.3. Limitations*

We showed that our method can be used to guide the process discovery towards a more useful process model. Still, we acknowledge that there are some limitations to the GPD method. First, the performance of the method highly depends on the quality of the used activity patterns. We introduced the expansion step in the GPD method to limit the risks of using low-quality activity patterns, i.e., patterns that do not properly capture the real execution of the process. The expanded process model can be validated on the original event log. Hence, quality problems can be detected. Second, if activity patterns share low-level events, then, there may be multiple mappings from low-level events to activities. The cost-based alignment techniques that we use to determine the optimal mapping chooses an arbitrary pattern in case there are multiple mappings with the same cost. As we show in Section 5.2, our method is computationally expensive due to the complexity of the alignment problem [5]. It is infeasible to compute the exact solution for very long traces and larger sets of activity patterns when using parallel composition. However, real-life event logs with traces of up to 250 events could be abstracted without problems as demonstrated in our prior work [19].

## 6. Related Work

Several event abstraction methods have been proposed. Moreover, there is a large body of work on activity recognition [28] and event processing [29]. We focus on work tha t is related to the field of process mining (i.e., an explicit process representation is used). We categorize the related work in *unsupervised* and *supervised* event abstraction methods. First, we describe how this paper extends our own previous work.

*Previous work.* This paper *extends* and *revises* our previous work [19] on event abstraction along three directions. (1) We define two different types of activity patterns depending on how they are obtained: manual patterns and discovered patterns. (2) We extended the original method by adding three additional steps: discover, expand, and validate (cf. Figure 3). Thereby, it is possible to use the resulting process model for reliable analytics. Assumptions that were made during the abstraction can be validated. (3) We evaluated the revised method using a new, more extensive case study.

*Unsupervised methods.* There are several approaches for unsupervised event abstraction in the field of process mining. Unsupervised method generally try to determine this relation based on identifying sub-sequences or using machine learning methods. Among the first proposals for event abstraction in process mining was the Fuzzy Miner by Günther et al. [30]. Activities are grouped together based on the significance of the edges between activities. Bose et al. [8] proposed to use common patterns based on repeating sequences to abstract events. Later, Günter et al. [2] used agglomerative hierarchical clustering to build a hierarchy of event clusters that can be used for event abstraction. Cook et al. [31] proposed an unsupervised algorithm for activity discovery based on sensor data that is guided by the minimum description length principle. Folino et al. [32] turned event abstraction into a predictive clustering problem and did not assume the notion of an event label in the new approach. Unsupervised abstraction methods, clearly, do not take existing knowledge into account and fail to provide meaningful labels for discovered event clusters.

*Supervised methods.* Approaches for supervised event abstraction assume some knowledge on the relation between low-level events and activities. Methods based on Complex Event Processing (CEP) [29] and activity recognition [28] typically assume a stream of events over which queries are evaluated. When a query is matched a high-level activity is detected. Traditionally, CEP does not consider the notion of process instance (i.e., case) and in case of overlapping queries (e.g., shared functionalities) both high-level activities would be detected. Still, there is some work that uses CEP withing a business process context [33–36]. However, none of these works provides a complete process discovery method based on domain knowledge. There are also proposals for supervised event abstraction that are more closely related to the field of process mining. Tax et al. [37] assume the existence of a labeled training set of traces. Moreover, the approach is limited to processes without concurrent high-level activities. Machine learning methods are used to infer the correct mapping. Senderovich et al. [38] determine an optimal mapping between sensor data of a real-time locating system and activities based on finding an optimal mapping using integer linear programming. Ferreira et al. [39] assume a complete process model of the high-level activities. They use hierarchical Markov models together with an expectation maximization method to find the mapping between low-level events and the high-level events in the process model. Later work [40] proposed a different, greedy approach that can better deal with noise in the event log. Fazzinga et al. [41] proposed a probabilistic method to the same problem, finding a mapping between an existing high-level model and events. The method is limited to traces of length less than 30 events due to the computational complexity. Most related to our work are the methods developed by Baier et al. [3, 42, 43]. Again, the methods assume knowledge about a single high-level model for the overall process. The goal is to automatically discover the relation between events and activities. Therefore, these methods are mainly targeting the situation where the process is assumed to be well known. The proposed methods use clustering methods and heuristics when challenged with event logs from processes that feature *concurrent* high-level activities and *noise* (i.e., erroneous or missing events). A later proposal using constraint programming approach in [43] only considers the control-flow perspective, i.e., rules based on data are not supported. Clearly, none of the supervised methods guides process discovery method towards a better process model that can be validated on the original event log.

## 7. Conclusion

We presented a new method that uses *event abstraction* based on domain knowledge to guide process discovery towards better results. We use *multi-perspective activity patterns* that encode assumptions on how high-level activities manifest themselves in terms of recorded low-level events. An abstracted event log is created on the basis of an alignment between activity patterns and the low-level event log. We use the abstracted event log to discover a high-level process model. We expand the high-level activities of this process model with the activity patterns to validate the quality of the discovered process model based on the original event log. We evaluated the GPD method by applying it to a real-life event log of a hospital. The case study shows that the GPD method can be successfully applied in complex, real-life environments. We created an abstracted event log and a high-level process model from an event log that was recorded by a system, in which (1) multiple high-level activities share low-level events with the same label, (2) high-level activities occur concurrently, and (3) erroneous events (i.e., noise) are recorded. We validated that the process models obtained with the GPD method are, indeed, good representations of the event logs studied. The guided process model is comprehensible by stakeholders and suitable to answer process questions, whereas the unguided process model discovered by standard process discovery methods is unsuitable for these purposes. By adding only basic domain knowledge, we could guide the discovery process towards a much more useful process model. Future work is needed to address some limitations of our method. At this point, if there are multiple optimal alignments for a sequence of events, i.e., multiple different instantiations of activity patterns could explain the observed behavior, then, one of them is chosen arbitrarily. A prioritization of activity patterns used during the alignment computation could be introduced. Moreover, it would be possible to introduce a simple heuristic that minimizes the number of pattern instantiations by introducing a small cost for instantiating a pattern to the alignment technique, Finally, alignment techniques require a lot of resources for event logs with very long traces. Work on decomposing or approximating the alignment computation could help to alleviate this limitation.

## References

[1] W. M. P. van der Aalst, Process Mining - Data Science in Action, Second Edition, Springer, 2016. doi:10.1007/978-3-662-49851-4.

[2] C. W. Günther, A. Rozinat, W. M. P. van der Aalst, Activity mining by global trace segmentation, in: BPM 2009 Workshops, Vol. 43 of LNBIP, Springer, 2009, pp. 128–139. doi:10.1007/978-3-642-12186-9_13.

[3] T. Baier, J. Mendling, M. Weske, Bridging abstraction layers in process mining, Inf. Syst. 46 (2014) 123–139. doi:10.1016/j.is.2014.04.004.

[4] H. A. Reijers, J. Mendling, R. M. Dijkman, Human and automatic modularizations of process models to enhance their comprehension, Inf. Syst. 36 (5) (2011) 881–897. doi:10.1016/j.is.2011.03.003.

[5] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, Balanced multi-perspective checking of process conformance, Computing 98 (4) (2016) 407–437. doi:10.1007/s00607-015-0441-1.

[6] W. M. P. van der Aalst, A. Adriansyah, B. F. van Dongen, Replaying history on process models for conformance checking and performance analysis, Wiley Interdiscip Rev Data Min Knowl Discov 2 (2) (2012) 182–192. doi:10.1002/widm.1045.

[7] N. Tax, N. Sidorova, R. Haakma, W. M. P. van der Aalst, Event abstraction for process mining using supervised learning techniques, in: IntelliSys, IEEE, 2016, pp. 161–170, pre-print, `https://arxiv.org/abs/1606.07283`.

[8] R. P. J. C. Bose, W. M. P. van der Aalst, Abstractions in process mining: A taxonomy of patterns, in: BPM 2009, Vol. 5701 of LNCS, Springer, 2009, pp. 159–175. `doi:10.1007/978-3-642-03848-8_12`.

[9] M. Leemans, W. M. P. van der Aalst, Discovery of frequent episodes in event logs, in: SIMPDA 2014, Vol. 237 of LNBIP, Springer, 2015, pp. 1–31. `doi:10.1007/978-3-319-27243-6_1`.

[10] C. Diamantini, L. Genga, D. Potena, Behavioral process mining for unstructured processes, J. Intell. Inf. Syst. 47 (1) (2016) 5–32. `doi:10.1007/s10844-016-0394-7`.

[11] J. Carmona, Projection approaches to process mining using region-based techniques, Data Min. Knowl. Discov. 24 (1) (2012) 218–246. `doi:10.1007/s10618-011-0226-x`.

[12] B. F. A. Hompes, H. M. W. E. Verbeek, W. M. P. van der Aalst, Finding suitable activity clusters for decomposed process discovery, in: SIMPDA 2014, Vol. 237 of LNBIP, Springer, 2015, pp. 32–57. `doi:10.1007/978-3-319-27243-6_2`.

[13] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Using life cycle information in process discovery, in: BPM 2015 Workshops, Vol. 256 of LNBIP, Springer, 2016, pp. 204–217. `doi:10.1007/978-3-319-42887-1_17`.

[14] IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams, IEEE Std 1849-2016 (2016). `doi:10.1109/IEEESTD.2016.7740858`.

[15] H. Leopold, J. Mendling, H. A. Reijers, M. L. Rosa, Simplifying process model abstraction: Techniques for generating model names, Inf. Syst. 39 (2014) 134–151. `doi:10.1016/j.is.2013.06.007`.

[16] A. Rogge-Solti, R. S. Mans, W. M. P. van der Aalst, M. Weske, Repairing event logs using timed process models, in: OTM 2013 Workshops, Springer, 2013, pp. 705–708. `doi:10.1007/978-3-642-41033-8_89`.

[17] T. Murata, Petri nets: Properties, analysis and applications, Proc. IEEE 77 (4) (1989) 541–580. `doi:10.1109/5.24143`.

[18] A. Bolt, M. de Leoni, W. M. P. van der Aalst, Scientific workflows for process mining: building blocks, scenarios, and implementation, STTT 18 (6) (2016) 607–628. `doi:10.1007/s10009-015-0399-5`.

[19] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, P. J. Toussaint, From low-level events to activities - A pattern-based approach, in: BPM 2016, Vol. 9850 of LNCS, Springer, 2016, pp. 125–141. doi:10.1007/978-3-319-45348-4_8.

[20] F. Mannhardt, Sepsis Cases - Event Log. Eindhoven University of Technology. Dataset. (2016). doi:10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

[21] F. Mannhardt, M. de Leoni, H. A. Reijers, The multi-perspective process explorer, in: F. Daniel, S. Zugal (Eds.), BPM 2015 (Demos), Vol. 1418 of CEUR Workshop Proceedings, CEUR-WS.org, 2015, pp. 130–134.

[22] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, Decision mining revisited - discovering overlapping rules, in: CAiSE 2016, Vol. 9694 of LNCS, Springer, 2016, pp. 377–392. doi:10.1007/978-3-319-39696-5_23.

[23] A. J. M. M. Weijters, J. T. S. Ribeiro, Flexible heuristics miner (FHM), in: CIDM 2011, IEEE, 2011, pp. 310–317. doi:10.1109/cidm.2011.5949453.

[24] S. J. van Zelst, B. F. van Dongen, W. M. P. van der Aalst, Avoiding over-fitting in ILP-based process discovery, in: BPM 2015, Vol. 9253 of LNCS, Springer, 2015, pp. 163–171. doi:10.1007/978-3-319-23063-4_10.

[25] J. Buijs, Flexible evolutionary algorithms for mining structured process models, Ph.D. thesis (2014). doi:10.6100/IR780920.

[26] M. Leoni, A. Marrella, Aligning real process executions and prescriptive process models through automated planning, Expert Syst Appldoi:10.1016/j.eswa.2017.03.047.

[27] J. Munoz-Gama, J. Carmona, W. M. van der Aalst, Single-entry single-exit decomposed conformance checking, Inf. Syst. 46 (2014) 102–122. doi:10.1016/j.is.2014.04.003.

[28] Y. Liu, L. Nie, L. Liu, D. S. Rosenblum, From action to activity: Sensor-based activity recognition, Neurocomputing 181 (2016) 108–115. doi:10.1016/j.neucom.2015.08.096.

[29] G. Cugola, A. Margara, Processing flows of information: From data stream to complex event processing, ACM Comput. Surv. 44 (3) (2012) 15. doi:10.1145/2187671.2187677.

[30] C. W. Günther, W. M. P. van der Aalst, Fuzzy mining - adaptive process simplification based on multi-perspective metrics, in: BPM 2007, Vol. 4714 of LNCS, Springer, 2007, pp. 328–343. doi:10.1007/978-3-540-75183-0_24.

[31] D. J. Cook, N. C. Krishnan, P. Rashidi, Activity discovery and activity recognition: A new partnership, IEEE T. Cybernetics 43 (3) (2013) 820–828. doi:10.1109/TSMCB.2012.2216873.

[32] F. Folino, M. Guarascio, L. Pontieri, Mining multi-variant process models from low-level logs, in: BIS 2015, Vol. 208 of LNBIP, Springer, 2015, pp. 165–177. doi:10.1007/978-3-319-19027-3_14.

[33] S. Bülow, M. Backmann, N. Herzberg, T. Hille, A. Meyer, B. Ulm, T. Y. Wong, M. Weske, Monitoring of business processes with complex event processing, in: BPM 2013 Workshops, Vol. 171 of LNBIP, Springer, 2013, pp. 277–290. doi:10.1007/978-3-319-06257-0_22.

[34] C. A. L. Oliveira, N. C. Silva, C. L. Sabat, R. M. F. Lima, Reducing the gap between business and information systems through complex event processing, Computing and Informatics 32 (2) (2013) 225–250.

[35] M. Weidlich, H. Ziekow, A. Gal, J. Mendling, M. Weske, Optimizing event pattern matching using business process models, IEEE Trans. Knowl. Data Eng. 26 (11) (2014) 2759–2773. doi:10.1109/TKDE.2014.2302306.

[36] S. Hallé, S. Varvaressos, A formalization of complex event stream processing, in: EDOC 2014, IEEE Computer Society, 2014, pp. 2–11. doi:10.1109/EDOC.2014.12.

[37] N. Tax, N. Sidorova, R. Haakma, W. M. van der Aalst, Mining local process models, J. of Innovation in Digital Ecosystems, in press. doi:http://dx.doi.org/10.1016/j.jides.2016.11.001.

[38] A. Senderovich, A. Rogge-Solti, A. Gal, J. Mendling, A. Mandelbaum, The ROAD from sensor data to process instances via interaction mining, in: CAiSE 2016, Vol. 9694 of LNCS, Springer, 2016, pp. 257–273. doi:10.1007/978-3-319-39696-5_16.

[39] D. R. Ferreira, F. Szimanski, C. G. Ralha, Mining the low-level behaviour of agents in high-level business processes, IJBPIM 6 (2) (2013) 146–166. doi:10.1504/IJBPIM.2013.054678.

[40] D. R. Ferreira, F. Szimanski, C. G. Ralha, Improving process models by mining mappings of low-level events to high-level activities, J. Intell. Inf. Syst. 43 (2) (2014) 379–407. doi:10.1007/s10844-014-0327-2.

[41] B. Fazzinga, S. Flesca, F. Furfaro, E. Masciari, L. Pontieri, A probabilistic unified framework for event abstraction and process detection from log data, in: OTM Conferences, Vol. 9415 of LNCS, Springer, 2015, pp. 320–328. doi:10.1007/978-3-319-26148-5_20.

[42] T. Baier, A. Rogge-Solti, J. Mendling, M. Weske, Matching of events and activities: an approach based on behavioral constraint satisfaction, in: SAC 2015, ACM, 2015, pp. 1225–1230. doi:10.1145/2695664.2699491.

[43] T. Baier, Matching events and activities, Ph.D. thesis, Universität Potsdam (2015).

## A. Implementation of the Composition Functions as DPNs

As explained in Section 3.4, we describe here the implementation of all proposed composition functions as DPNs. Figure A.20 and the following 5 paragraphs describe the implementation of the composition of activity patterns as a DPN. A comprehensive introduction of DPNs is given in [5]. Given activity patterns $ap_a = (p_a, \lambda_a, \nu_a, hl_a, lt_a), ap_b = (p_b, \lambda_b, \nu_b, hl_b, lt_b) \in P$ that are implemented as DPNs with source places $s_a, s_b$ and sink places $e_a, e_b$, we describe how to compose $ap_a$ and $ap_b$ to a combined pattern for each of the introduced composition functions. We focus on the composition of their process models $p_a$ and $p_b$ since the remaining mapping functions are combined by taking their union.

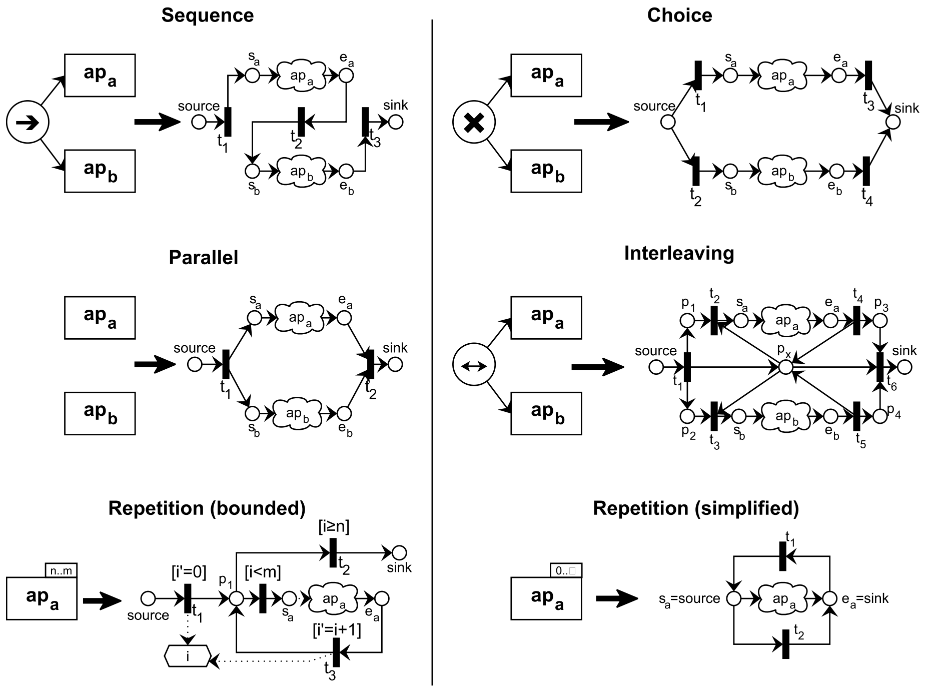


Figure A.20: Implementation of the composition functions using the DPN notation

*Sequence.* Pattern $ap_a$ and pattern $ap_b$ are composed in sequence by adding two places (`source`, `sink`) and three transitions $(t_1, t_2, t_3)$ as shown in Fig A.20. Places `source` and `sink` are the entry and exit points of the composed pattern and transitions $t_1, t_2, t_3$ connect the source places $s_a, s_b$ and sink places $e_a, e_b$ of both patterns in sequence.

*Choice.* Pattern $ap_a$ and pattern $ap_b$ are composed in choice by adding two places (`source`, `sink`) and four transitions ($t_1, t_2, t_3, t_4$) as shown in Fig A.20. Places `source` and `sink` are the entry and exit points of the composed pattern. The control-flow is split after place `source` such that either $t_1$ or $t_2$ has to be executed. Transitions $t_1, t_2$ are connected to the source places $s_a, s_b$ of the patterns. The sink places $e_a, e_b$ of the patterns are connected to transitions $t_3, t_4$. Finally, both transitions $t_3, t_4$ are connected to the exit place *sink*.

*Parallel.* Pattern $ap_a$ and pattern $ap_b$ are composed in parallel by adding two places (`source`, `sink`) and two transitions ($t_1, t_2$) as shown in Fig A.20. The control-flow is split using transition $t_1$ such that both patterns $p_a$ and $p_b$ have to be executed. Afterward, both places $e_a$ and $e_b$ are connected to transition $t_2$, which merges the control-flow.

*Interleaving.* Pattern $ap_a$ and pattern $ap_b$ are composed in interleaving by adding seven places (`source`, `sink`, $p_1, p_2, p_3, p_4$ and $p_x$) and six transitions ($t_1, t_2, t_3, t_4, t_5, t_6$) as shown in Fig A.20. Intuitively, the interleaving of $p_a$ and $p_b$ can be expressed as choice between any possible ordering of $p_a$ and $p_b$. The control-flow is split in parallel using $t_1$ enabling any possible re-ordering of patterns $p_a$ and $p_b$. Place $p_x$ is added restricting the behavior such that only either $p_a$ or $p_b$ can be executed at the same time. Finally, transition $t_6$ merges the control-flow from places $p_x, p_3$ and $p_4$.

*Repetition.* The repetition of a pattern $ap_a$ is modeled by adding three places (`source`, `sink`, $p_1$) and three transitions ($t_1, t_2, t_2$). We use a counter variable $i$ that keeps track of the repetitions and add guards to transitions $t_1, t_2$ and $t_3$ that constrain the maximum allowed and minimum required number of repetitions accordingly. Transition $t_3$ increases the counter $i$ on each iteration. Please note because we have a-priori knowledge of the number of repetitions such a construct can always be unfolded to a normal Petri net, e.g., by repeated use of the sequence and choice composition and duplicating the pattern. Moreover, in case the number of repetitions is unbounded, i.e., $m = \infty$ and $n = 0$ we can simplify the construction as shown on the right-hand side of Figure A.20.