Chapter 10
Process Intelligence

If you can't measure something, you can't understand it. If you can't understand it, you can't control it. If you can't control it, you can't improve it.
H. James Harrington (1929–)

It is a central idea of BPM that processes are explicitly defined, then executed, and that information about process execution is prepared and analyzed. In this way, this information provides a feedback loop on how the process might be redesigned. Data about the execution of processes can stem from BPMSs in which processes are specified, but also from systems that do not work with an explicit process model, for instance ERP systems or ticketing systems. Data from those systems have to be transformed to meet the requirements of intelligent process execution analysis. This field is typically referred to as process mining.

This chapter deals with intelligently using the data generated from the execution of the process. We refer to such data as event logs, covering what has been done when by whom in relation to which process instance. First, we investigate the structure of event logs, their relationship to process models, and their usefulness for process monitoring and controlling. Afterwards, we discuss three major objectives of intelligent process analysis, namely transparency, performance and conformance. We discuss automatic process discovery as a technical step to achieve transparency of how the process is executed in reality. Then, we study how the analysis of event logs can provide insights into process performance. Finally, we discuss how the conformance between event logs and a process model can be checked.

10.1 Process Execution and Event Logs

In the previous chapter, we studied how a process model can be specified in a way that a BPMS can support its execution. Both process participants and process owners are involved in the execution of business processes. However, their perspective is quite different. Process participants work on tasks, which produce execution data as a side product. We call this data *event logs*. Process owners are particularly interested in drawing conclusions from such event logs. In this section, we discuss which

M. Dumas et al., *Fundamentals of Business Process Management*,
DOI 10.1007/978-3-642-33143-5_10, © Springer-Verlag Berlin Heidelberg 2013

questions can be answered using event data and how event logs and process models relate to each other.

10.1.1 The Perspective of Participants on Process Execution

When a process is executed on a BPMS or another piece of software, there is a clear separation between coordination and execution of tasks. The system usually takes care of coordinating individual cases informing participants about which tasks they need to work on. Accordingly, participants often see only those tasks that they are directly responsible for, while the system hides the complexity of the overall process. Each participant typically has a personal *worklist* that shows the set of *work items* that still need to be worked on. If an explicit process model exists, each of these work items corresponds to a task in the process model. However, there might exist multiple work items corresponding to a single task if several cases are currently being worked on. For example, at a specific point in time, Chuck as a process participant might see that four work items are in his worklist, all relating to the "Confirm order" task of the order fulfillment process: one work item relates to an order from customer A, one from customer B, and two from customer C.

The structure of a work item is defined in the executable process model or directly implemented in the software. This means that participants see those data fields that have been declared as input for a task. For each work item they are working on, they are supposed to document at least the completion. In this way, the system can keep track of the state of the process at any point in time. Among others, it is easy to record at which point in time somebody has started working on a work item, which input data were available, what output data were created, and who was the participant working on it. For example, when Chuck has confirmed the order of customer B, he enters the result in the system, and the system can decide automatically if next the invoice should be emitted or the order confirmation should be escalated to someone above Chuck. Most BPMSs and also other information systems record such data on what has been done at which point in time. The file in which these data is stored is called a *log file*, and the data in it is called *event logs*. Each time another task is completed, a new entry is added to the log file. That is, once Chuck has entered his data, the system appends a line in the log file stating that Chuck has confirmed an order with a corresponding timestamp.

10.1.2 The Perspective of the Process Owner on Process Execution

Event logs have the potential to reveal important management insights into how a process works in reality. Therefore, the process owner is most interested in analyzing it in a systematic way. In essence, we distinguish three major application scenarios for using event logs: automatic process discovery, performance analysis and conformance checking, all related to questions the process owner might ask.

What is the actual process model? Automatic process discovery is concerned with the question of how a process actually works in reality. In Chap. 5, we mentioned that the event logs can be used as an input to evidence-based process discovery. Automatic process discovery utilizes event logs for the generation of a corresponding process model. In this way, event logs are valuable to find a process model where no model existed before, and to adjust an existing model according to how the process really works.

What is the performance of the process? In Chap. 7, we discussed that process analyses such as flow analysis suffer from the fact that the average cycle time for each task in the process model needs to be estimated. Also, often strong assumptions are required such that the behavior of the process is not influenced by the load. Using event logs, we are able to inspect the actual behavior of a process and compare it with insights from process analysis. Furthermore, historic information about process execution can be leveraged for making operational decisions.

To which extent are the rules of the process model followed? Conformance checking is a collection of techniques that compare a set of event logs with a set of constraints or an existing process model. There are situations when process models are defined, but they are not strictly enforced by a corresponding BPMS. In these situations, conformance checking can be utilized in order to determine how often the process is executed as expected and, if not, at which stages deviations can be found. Here, event logs help to understand either where the model needs to be corrected or where the behavior of the participants working in the process has to be adapted.

By providing answers to these three types of question, we can get insights into the process, which may help to reposition it in the *Devil's Quadrangle*. Specifically, the dimension of time increases in transparency by investigating event logs: the timestamps show when tasks are executed and how long they take. There is also a strong association with cost if the working time of the participants working in the process can be assigned to a particular process instance. Flexibility can be analyzed based on the different paths a process takes. The historic set of actually used paths and their variety gives an indication for this dimension. Finally, also quality issues might be identified from event logs, for instance when inspecting the number of reworks and iterations required for a specific task.

The process owner can use event logs as input to two different control mechanisms: on an aggregated level and on an instance level. The mechanisms are called *process controlling* and *process monitoring*, respectively.

Process Controlling deals with the analysis of historic process execution. The input for process controlling are event logs that relate to a particular period of time, for instance a quarter or a full year. Process controlling provides insights into whether the general objectives of a process have been met and whether the KPIs are in line. Typically, process controlling is an *offline* activity, which involves logs of completed process executions.

Process Monitoring is concerned with the quality of currently running process instances. The input for process monitoring are the event logs of individual process instances or cases. Process monitoring works with objectives and rules that are formulated for these individual cases, and triggers counteractions when these rules are violated, for instance when a customer request is not responded in time. Typically, process monitoring is a continuous *online* activity which involves events of currently running instances.

Both process monitoring and process controlling play an important role in aligning the process with its overall business objectives. In that respect, they are closely related to ideas of quality management and the Plan-do-check-act (PDCA) cycle. PDCA can be regarded as an inspiration for the concept of a business process management lifecycle as discussed in the first chapter of this book. Process monitoring and controlling (check) investigate the data from executing processes (do) such that redesign measures (act) can be taken to realign the execution with the objectives (plan). All these concepts have inspired the idea of a *process cockpit* as a software tool where data on the execution of processes is provided online using charts and appropriate visualization (see Fig. 9.4 for an example). Often, these tools are also called Business Activity Monitoring (BAM) tools or Process Performance Measurement (PPM) tools.

10.1.3 Structure of Event Logs

Process monitoring and controlling strongly rely on event data to be recorded during the execution of processes. *Event logs* contain a set of events. Accordingly, we can understand an event log as a list of event recordings. Figure 10.1 gives an illustration of what data is typically stored in event logs. We can see that a single event has a unique event ID. Furthermore, it refers to one individual case, it has a timestamp, and it shows which resources executed which task. These may be participants (e.g. Chuck and Susi) or software systems (SYS1, SYS2, DMS). For several analysis techniques that we will discuss in this chapter, it is a minimum requirement that the events in the log refer to (i) one case, (ii) one task, and (iii) a point in time. Having these three pieces of information available, we can for instance discover a process model from the logs. In practice, there are often additional pieces of information stored for each event like costs, system being used, or data on the handled business case. These can be used for clustering, correlating or finding causal relationships in the event logs.

The event log of Fig. 10.1 is captured as a list in a tabular format.[1] The problem with event logs is that each vendor and software system defines individual log formats. In order to leverage the adoption of event log analysis tools such as the open

[1] For simplicity, we only consider one supplier in this example.

Case ID	Event ID	Timestamp	Activity	Resource
1	Ch-4680555556-1	2012-07-30 11:14	Check stock availability	SYS1
1	Re-5972222222-1	2012-07-30 14:20	Retrieve product from warehouse	Rick
1	Co-6319444444-1	2012-07-30 15:10	Confirm order	Chuck
1	Ge-6402777778-1	2012-07-30 15:22	Get shipping address	SYS2
1	Em-6555555556-1	2012-07-30 15:44	Emit invoice	SYS2
1	Re-4180555556-1	2012-08-04 10:02	Receive payment	SYS2
1	Sh-4659722222-1	2012-08-05 11:11	Ship product	Susi
1	Ar-3833333333-1	2012-08-06 09:12	Archive order	DMS
2	Ch-4055555556-2	2012-08-01 09:44	Check stock availability	SYS1
2	Ch-4208333333-2	2012-08-01 10:06	Check materials availability	SYS1
2	Re-4666666667-2	2012-08-01 11:12	Request raw materials	Ringo
2	Ob-3263888889-2	2012-08-03 07:50	Obtain raw materials	Olaf
2	Ma-6131944444-2	2012-08-04 14:43	Manufacture product	SYS1
2	Co-6187615741-2	2012-08-04 14:51	Confirm order	Conny
2	Em-6388888889-2	2012-08-04 15:20	Emit invoice	SYS2
2	Ge-6439814815-2	2012-08-04 15:27	Get shipping address	SYS2
2	Sh-7277777778-2	2012-08-04 17:28	Ship product	Sara
2	Re-3611111111-2	2012-08-07 08:40	Receive payment	SYS2
2	Ar-3680555556-2	2012-08-07 08:50	Archive order	DMS
3	Ch-4208333333-3	2012-08-02 10:06	Check stock availability	SYS1
3	Ch-4243055556-3	2012-08-02 10:11	Check materials availability	SYS1
3	Ma-6694444444-3	2012-08-02 16:04	Manufacture product	SYS1
3	Co-6751157407-3	2012-08-02 16:12	Confirm order	Chuck
3	Em-6895833333-3	2012-08-02 16:33	Emit invoice	SYS2
3	Sh-7013888889-3	2012-08-02 16:50	Get shipping address	SYS2
3	Ge-7069444444-3	2012-08-02 16:58	Ship product	Emil
3	Re-4305555556-3	2012-08-06 10:20	Receive payment	SYS2
3	Ar-4340277778-3	2012-08-06 10:25	Archive order	DMS
4	Ch-3409722222-4	2012-08-04 08:11	Check stock availability	SYS1
4	Re-5000115741-4	2012-08-04 12:00	Retrieve product from warehouse	SYS1
4	Co-5041898148-4	2012-08-04 12:06	Confirm order	Hans
4	Ge-5223148148-4	2012-08-04 12:32	Get shipping address	SYS2
4	Em-4034837963-4	2012-08-08 09:41	Emit invoice	SYS2
4	Re-4180555556-4	2012-08-08 10:02	Receive payment	SYS2
4	Sh-5715277778-4	2012-08-08 13:43	Ship product	Susi
4	Ar-5888888889-4	2012-08-08 14:08	Archive order	DMS

Fig. 10.1 Example of an event log for the order fulfillment process

source tool ProM,[2] the *IEEE Task Force on Process Mining* promotes the usage of the *eXtensible Event Stream* (*XES*) format. Several tools work with XES event logs or offer features to convert events logs into this format. The metamodel of XES is illustrated in Fig. 10.2. Each XES file represents a log. It contains multiple traces, and each trace can contain multiple events. All of them can contain different attributes. An attribute has to be either a string, date, int, float, or boolean element as a key-value pair. Attributes have to refer to a global definition. There are two global elements in the XES file, one for defining trace attributes, the other for defining event attributes. Several classifiers can be defined in XES. A classifier maps one of more attributes of an event to a label that is used in the output of the analysis tool. In this way, for instance, events can be associated with activities.

[2]The software is available at http://www.promtools.org.

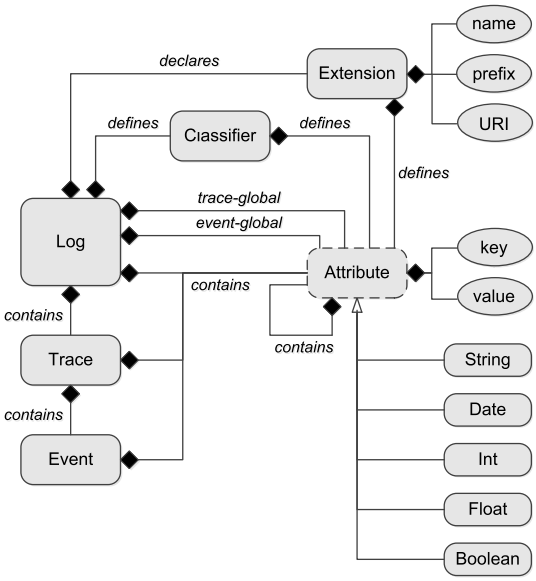

Fig. 10.2 Metamodel of the XES format

Fig. 10.3 Example of a file in the XES format

```
<log xes.version="1.0" xes.features="arbitrary-depth" xmlns="http://.../xes">
  <extension name="Concept" prefix="concept" uri="http://.../xes/concept.xesext"/>
  <extension name="Time" prefix="time" uri="http://.../xes/time.xesext"/>
  <global scope="trace">
    <string key="concept:name" value=""/>
  </global>
  <global scope="event">
    <string key="concept:name" value=""/>
    <date key="time:timestamp" value="1970-01-01T00:00:00.000+00:00"/>
    <string key="resource" value=""/>
  </global>
  <classifier name="Activity" keys="concept:name"/>
  <float key="log attribute" value="2335.23"/>
  <trace>
    <string key="concept:name" value="1"/>
    <event>
      <string key="concept:name" value="Check stock availability"/>
      <date key="time:timestamp" value="2012-07-30T11:14:00:000+01:00"/>
      <string key="resource" value="Chuck"/>
    </event>
    <event>
      <string key="concept:name" value="Retrieve product from warehouse"/>
      <date key="time:timestamp" value="2012-07-30T14:20:00:000+01:00"/>
      <string key="resource" value="Rick"/>
    </event>
  </trace>
</log>
```

Figure 10.3 shows how parts of the information of the event log example from Fig. 10.1 can be stored in an XES file. From the first "global" element (scope = "trace"), we see that each trace element is expected to have a "concept:name" attribute. For the trace defined below, this attribute has the value 1. Furthermore, there are three different attributes expected for an event ("global" element with scope = "event"): "concept:name", "time:timestamp", and "resource". In the trace defined below, we observe two events. The first one refers to "Check stock availability",

which was completed by SYS1 on the 30th of July 2012 at 11:14. The second event captures "Retrieve product from warehouse" conducted by Rick at 14:20.

10.1.4 Challenges of Extracting Event Logs

Event data that is available in some tabular format like that visualized in Fig. 10.1 can be readily converted to XES, and then analyzed using appropriate tools. In many cases though, the data which is relevant for event logs is not directly accessible in the required format, but has to be extracted from different sources and to be integrated. Therefore, we can identify five major challenges for log data extraction, potentially requiring considerable effort in a project. These challenges are:

1. **Correlation challenge:** This refers to the problem of identifying the case an event belongs to. Many information systems do not have an explicit notion of process defined. Therefore, we have to investigate which attribute of process-related entities might serve as a case identifier. Often, it is possible to utilize entity identifiers such as order number, invoice number, or shipment number.
2. **Timestamps challenge:** The challenge to work properly with timestamps stems from the fact that many information systems do not consider logging as a primary task. This means that logging is often delayed until the system has idle time or little load. Therefore, we might find sequential events with the same timestamp in the log. This problem is worsened when logs from different information systems potentially operating in different time zones have to be integrated. Partially, such problems can be resolved with domain knowledge, for example when events are known to always occur in a specific order.
3. **Snapshots challenge:** This point refers to the issue of having log data available for a certain period of time. For long running processes, we might not be able to observe all cases of the log with their full end-to-end duration in the considered period of time. It is a good idea to exclude such incomplete cases with missing head or tail. However, one should be aware that such a filtering might also introduce a bias, for instance that only brief cases are considered. Therefore, the time span reflected by the log should be significantly longer than the average duration of a case.
4. **Scoping challenge:** The scoping of the event spectrum is a challenge when the available information system does not directly produce event logs. Information systems such as ERP systems record an extensive amount of process-related events in numerous tables. Event logs have to be generated from the entries in these tables, which requires a detailed understanding of the data semantics. Such system expertise may not readily be available.
5. **Granularity challenge:** Typically, we are interested in conducting event log analysis on a conceptual level for which we have process models defined. In general, the granularity of event log recording is much finer such that each activity of a process model might map to a set of events. For example, an activity like "Retrieve product from warehouse" on the abstraction level of a process

model maps to a series of events like “Work item #1,211 assigned”, “Work item #1,211 started”, “Purchase order form opened”, “Product retrieved” and “Work item #1,211 completed”. Often, fine-granular events may show up repeatedly in the logs while on an abstract level only a single task is executed. Therefore, it is difficult to define a precise mapping between the two levels of abstraction.

Exercise 10.1 Consider the final assembly process of Airbus for their A380 series. The final assembly of this aircraft series is located at the production site in Toulouse, France. Large parts are brought by ship to Bordeaux and brought to Toulouse by waterway and road transport. What is the challenge when log data of the A380 production process has to be integrated?

10.2 Automatic Process Discovery

Automatic process discovery is a specific process mining technique. The goal of automatic process discovery is to construct a process model that captures the behavior of an event log in a representative way. The construction is meant to work automatically and generically, using an algorithm that should make minimal assumptions about properties of the log and the resulting process model. Being representative in this context loosely means that the constructed process model should be able to replay the cases of the event log and forbid behavior not found in the logs. In the following, we first discuss log data assumptions, then we present the α-algorithm as a basic discovery algorithm, and turn to the notion of representativeness in more detail.

10.2.1 Assumptions of the α-Algorithm

The *α-algorithm* is a basic algorithm for discovering process models from event logs. It is basic as it is less complex than other, more advanced algorithms. Beyond that, it makes certain assumptions about the provided event logs that we will later discuss as partially being problematic. These assumptions are the following:

- **Order of Events:** The events in the log are chronologically ordered. Such a chronological order can be defined based on timestamps.
- **Case Reference:** Each event refers to a single case.
- **Activity Reference:** Each event relates to a specific activity of the process.
- **Activity Completeness:** Each activity of the process is included in the log.
- **Behavioral Completeness:** The log is behaviorally complete in the sense that if an activity a can be directly followed by an activity b, then there is at least one case in the log where we observe ab.

Case ID	Event ID	Timestamp	Activity
1	Ch-468	2012-07-30 11:14	Check stock availability
1	Re-597	2012-07-30 14:20	Retrieve product from warehouse
1	Co-631	2012-07-30 15:10	Confirm order
1	Ge-640	2012-07-30 15:22	Get shipping address
1	Em-655	2012-07-30 15:44	Emit invoice
1	Re-418	2012-08-04 10:02	Receive payment
1	Sh-465	2012-08-05 11:11	Ship product
1	Ar-383	2012-08-06 09:12	Archive order
2	Ch-405	2012-08-01 09:44	Check stock availability
2	Ch-420	2012-08-01 10:06	Check materials availability
2	Re-466	2012-08-01 11:12	Request raw materials
2	Ob-326	2012-08-03 07:50	Obtain raw materials
2	Ma-613	2012-08-04 14:43	Manufacture product
2	Co-618	2012-08-04 14:51	Confirm order
2	Em-638	2012-08-04 15:20	Emit invoice
2	Ge-643	2012-08-04 15:27	Get shipping address
2	Sh-727	2012-08-04 17:28	Ship product
2	Re-361	2012-08-07 08:40	Receive payment
2	Ar-368	2012-08-07 08:50	Archive order

Letter	Actvities
a	Check stock availability
b	Retrieve product from warehouse
c	Check materials availability
d	Request raw materials
e	Obtain raw materials
f	Manufacture product
g	Confirm order
h	Get shipping address
i	Ship product
j	Emit invoice
k	Receive payment
l	Archive order

Workflow Log
a,b,g,h,j,k,i,l
a,c,d,e,f,g,j,h,i,k,l
...

Fig. 10.4 Definition of a workflow log

The first three assumptions refer to the information content of an event in the logs. These assumptions are rather general and non-restrictive. The activity completeness criterion refers to the fact that we can only include those activities in the generated process model that we observe in the logs. The behavioral completeness has the strongest implications. In practice, we can hardly assume that we find the complete set of behavioral options in a log. Advanced techniques try to make weaker assumptions on this point.

In line with these assumptions, we use a so-called *workflow log* as starting point for using the α-algorithm. Figure 10.4 shows how a workflow log can be constructed from an event log. In the following, we will work with letters as a reference to tasks. A workflow log is a collection of all unique execution sequences observed in the log. The α-algorithm does not distinguish how often a specific execution sequence was observed in a workflow log.

Exercise 10.2 Have a look at Fig. 10.1 and translate it into a workflow log following the same mapping rules as in Fig. 10.4.

10.2.2 The Order Relations of the α-Algorithm

The α-algorithm works in two phases in order to construct a process model. In the first phase, different order relations are extracted from the workflow log L. In the second phase, the process model is constructed in a stepwise fashion from these identified relations. The *order* relations refer to tasks which directly follow one another in the log. It provides the basis for the definition of three more specific relations that refer to *causality*, to potential *parallelism* and to *non-succession*. We refer to this set of relations as the α relations.

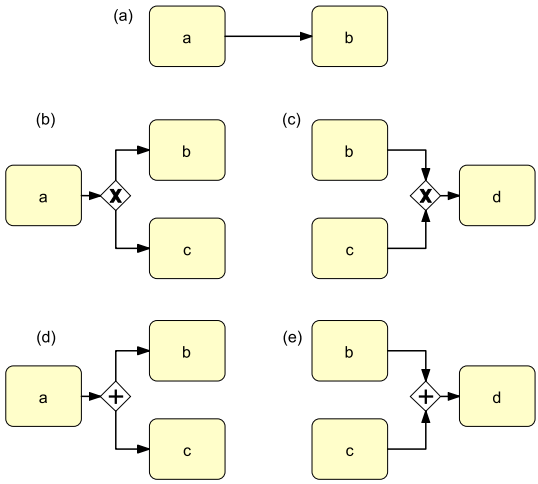

Fig. 10.5 Simple control flow patterns

- The basic order relation $a > b$ holds if we can observe in the workflow log L that an task a is directly followed by b.
- The causality relation $a \rightarrow b$ is derived from the basic relation. It holds if we observe in L that $a > b$ and that $b \not> a$.
- The relation of potential parallelism $a \parallel b$ holds if both $a > b$ and $b > a$ is observed in the workflow log L.
- The relation of no direct succession $a\#b$ holds if $a \not> b$ and $b \not> a$.

The reason why exactly these relations are used is shown in Fig. 10.5. There are five characteristic combinations of relations between the tasks in a workflow log that can be mapped to simple control flow patterns.

Pattern (a) depicts a sequence of tasks a and b. If we model them in this way, it should be guaranteed that in a workflow log we will find a followed by b, i.e. $a > b$, but never b followed by a, i.e. $b \not> a$. This means that the causality relation $a \rightarrow b$ should hold.

Pattern (b) also relates to a characteristic combination of relations. The workflow log should show that $a \rightarrow b$ and $a \rightarrow c$ hold, and that b and c would not be mutual successors, i.e. $b\#c$.

Pattern (c) also requires that b and c would not be mutual successors, i.e. $b\#c$, while both $b \rightarrow d$ and $c \rightarrow d$ have to hold.

Pattern (d) demands that $a \rightarrow b$ and $a \rightarrow c$ hold, and that b and c show potential parallelism, i.e. $b \parallel c$.

Pattern (e) refers to $b \rightarrow d$ and $c \rightarrow d$ while b and c show potential parallelism, i.e. $b \parallel c$.

The idea of the α-algorithm is to identify the relations between all pairs of tasks from the workflow log in order to reconstruct a process model based on Patterns (a) to (e). Therefore, before applying the α-algorithm, we first have to

Fig. 10.6 Footprint represented as a matrix of the workflow log $L = [\langle a, b, g, h, j, k, i, l\rangle, \langle a, c, d, e, f, g, j, h, i, k, l\rangle]$

	a	b	c	d	e	f	g	h	i	j	k	l
a	#	$\rightarrow$	$\rightarrow$	#	#	#	#	#	#	#	#	#
b	$\leftarrow$	#	#	#	#	#	$\rightarrow$	#	#	#	#	#
c	$\leftarrow$	#	#	$\rightarrow$	#	#	#	#	#	#	#	#
d	#	#	$\leftarrow$	#	$\rightarrow$	#	#	#	#	#	#	#
e	#	#	#	$\leftarrow$	#	$\rightarrow$	#	#	#	#	#	#
f	#	#	#	#	$\leftarrow$	#	$\rightarrow$	#	#	#	#	#
g	#	$\leftarrow$	#	#	#	$\leftarrow$	#	$\rightarrow$	#	$\rightarrow$	#	#
h	#	#	#	#	#	#	$\leftarrow$	#	$\rightarrow$	$\parallel$	#	#
i	#	#	#	#	#	#	#	$\leftarrow$	#	#	$\parallel$	$\rightarrow$
j	#	#	#	#	#	#	$\leftarrow$	$\parallel$	#	#	$\rightarrow$	#
k	#	#	#	#	#	#	#	#	$\parallel$	$\leftarrow$	#	$\rightarrow$
l	#	#	#	#	#	#	#	#	$\leftarrow$	#	$\leftarrow$	#

extract all basic order relations from the workflow log L. Consider the workflow log depicted in Fig. 10.4 containing the two cases $\langle a, b, g, h, j, k, i, l\rangle$ and $\langle a, c, d, e, f, g, j, h, i, k, l\rangle$. From this workflow log, we can derive the following relations.

- The basic order relations $>$ refer to each pair of tasks that directly follow one another. It can be directly read from the log:

$a > b$	$h > j$	$i > l$	$d > e$	$g > j$	$i > k$
$b > g$	$j > k$	$a > c$	$e > f$	$j > h$	$k > l$
$g > h$	$k > i$	$c > d$	$f > g$	$h > i$	

- The causal relations can be found when we check of each order relation whether it does not holds in the opposite direction. This holds for all pairs except (h, j) and (i, k), respectively. We get:

$a \rightarrow b$	$j \rightarrow k$	$a \rightarrow c$	$d \rightarrow e$	$f \rightarrow g$	$h \rightarrow i$
$b \rightarrow g$	$i \rightarrow l$	$c \rightarrow d$	$e \rightarrow f$	$g \rightarrow j$	$k \rightarrow l$
$g \rightarrow h$					

- The potential parallelism relation holds true for $h \parallel j$ as well as for $k \parallel i$ (and the corresponding symmetric cases).
- The remaining relation of no direction succession can be found for all pairs that do not belong to $\rightarrow$ and $\parallel$. It can be easily derived when we write down the relations in a matrix as shown in Fig. 10.6. This matrix is also referred to as the *footprint matrix* of the log.

Exercise 10.3 Have a look at the workflow log you constructed in Exercise 10.2. Define the relations $>, \rightarrow, \parallel, \#$ and the footprint matrix of this workflow log.

10.2.3 The α-Algorithm

The α-algorithm is a basic algorithm for automatic process discovery that takes an event log L and its α relations as a starting point. The essential idea of the algorithm is that tasks that directly follow one another in the log should be directly connected in the process model. Furthermore, if there is more than one task that can follow after another, we have to determine whether the set of succeeding tasks is partially exclusive or concurrent. An exception from the principle that tasks should be connected in the process model are those that are potentially parallel, i.e. those pairs included in $\|$. The details of the α-algorithm are defined according to the following eight steps.[3]

1. Identify the set of all tasks in the log as T_L.
2. Identify the set of all tasks that have been observed as the first task in some case as T_I.
3. Identify the set of all tasks that have been observed as the last task in some case as T_O.
4. Identify the set of all connections to be potentially represented in the process model as a set X_L. Add the following elements to X_L:
   a. Pattern (a): all pairs for which hold $a \rightarrow b$.
   b. Pattern (b): all triples for which hold $a \rightarrow (b\#c)$.
   c. Pattern (c): all triples for which hold $(b\#c) \rightarrow d$.

   Note that triples for which Pattern (d) $a \rightarrow (b \parallel c)$ or Pattern (e) $(b \parallel c) \rightarrow d$ hold are not included in X_L.
5. Construct the set Y_L as a subset of X_L by:
   a. Eliminating $a \rightarrow b$ and $a \rightarrow c$ if there exists some $a \rightarrow (b\#c)$.
   b. Eliminating $b \rightarrow c$ and $b \rightarrow d$ if there exists some $(b\#c) \rightarrow d$.
6. Connect start and end events in the following way:
   a. If there are multiple tasks in the set T_I of first tasks, then draw a start event leading to a split (XOR or AND depending on the relation between the tasks), which connects to every task in T_I. Otherwise, directly connect the start event with the only first task.
   b. For each task in the set T_O of last tasks, add an end event and draw an arc from the task to the end event.
7. Construct the flow arcs in the following way:
   a. Pattern (a): For each $a \rightarrow b$ in Y_L, draw an arc a to b.
   b. Pattern (b): For each $a \rightarrow (b\#c)$ in Y_L, draw an arc from a to an XOR-split, and from there to b and c.
   c. Pattern (c): For each $(b\#c) \rightarrow d$ in Y_L, draw an arc from b and c to an XOR-join, and from there to d.

[3] Note that the α-algorithm was originally defined for constructing Petri nets. The version shown here is a simplification based on the five simple control flow patterns of Fig. 10.5 in order to construct BPMN models.

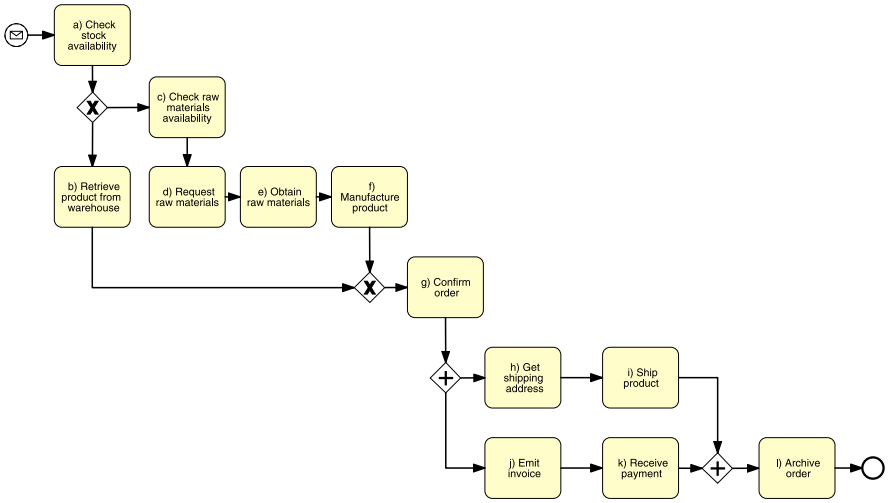

Fig. 10.7 Process model constructed by the α-algorithm from workflow log $L = [\langle a, b, g, h, j, k, i, l\rangle, \langle a, c, d, e, f, g, j, h, i, k, l\rangle]$

d. Pattern (d) and (e): If a task in the so constructed process model has multiple incoming or multiple outgoing arcs, bundle these arcs with an AND-split or AND-join, respectively.

8. Return the newly constructed process model.

Let us step through the α-algorithm with the workflow log $L = [\langle a, b, g, h, j, k, i, l\rangle, \langle a, c, d, e, f, g, j, h, i, k, l\rangle]$ as an example input. The Steps 1–3 identify $T_L = \{a, b, c, d, e, f, g, h, i, j, k, l\}$, $T_I = \{a\}$, and $T_O = \{l\}$. In Step 4a all causal relations are added to X_L including $a \rightarrow b$ and $a \rightarrow c$, etc. In Step 4b, we work row by row through the footprint matrix of Fig. 10.6 and check if there are cells sharing a $\rightarrow$ relation while relating to tasks that are pairwise in #. In the row a, we observe both $a \rightarrow b$ and $a \rightarrow c$. Also, $b\#c$ holds. Therefore, we add $a \rightarrow (b\#c)$ to X_L. We also consider row g and its relation to h and j. However, as $h \parallel j$ holds, we do not add them. In Step 4c, we progress column by column through the footprint matrix and see if there are cells sharing a $\rightarrow$ relation while relating to tasks that are mutually in #. In column g, we observe two $\rightarrow$ relations to b and f. Also, $b\#f$ holds. Accordingly, we add $(b\#f) \rightarrow g$ to X_L. We also check i and k that share the same relation to l. However, as $i \parallel k$ holds, we do not add them. There are no further complex combinations found in Step 4d.

In Step 5, we eliminate the basic elements in X_L that are covered by the complex patterns found in Steps 4b and 4c. Accordingly, we delete $a \rightarrow b$, $a \rightarrow c$, $b \rightarrow g$, and $f \rightarrow g$. In Step 6a we introduce a start event and connect it with a; in 6b, task l is connected with an end event. In Step 7, arcs and gateways are added for the elements of Y_L. Finally, in Step 8 the process model is returned. The resulting model is shown in Fig. 10.7.

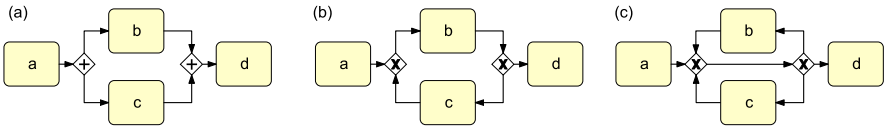

Fig. 10.8 Examples of two short loops, which are problematic for the α-algorithm

Exercise 10.4 Have a look at workflow log and the footprint you constructed in Exercises 10.2 and 10.3. Document progressing through the steps of the α-algorithm with this input and draw the resulting process model.

10.2.4 Robust Process Discovery

Clearly, the α-algorithm has its merits. It can reconstruct a process model from a behaviorally complete event log if that log has been generated from a structured process model. There are also limitations to be noted, though. The α-algorithm is not able to distinguish so-called *short loops* from true parallelism. As can be seen in Fig. 10.8, all three models can produce the workflow logs that yield $b \parallel c$ in the corresponding footprint. Several extensions to the α-algorithm have been proposed. The idea of the $\alpha+$-algorithm is to define the relation $\parallel$ in a stricter way such that $b \parallel c$ is only included if there is no sequence bcb in the logs. In this way, models (a) and (b) in Fig. 10.8 can be distinguished from each other in their generated logs. Furthermore, we can use preprocessing to extract direct repetition like aa or bb from the logs, note down the corresponding tasks, and continue with a log from which such repeated behavior is mapped to a single execution.

Further problems for the α-algorithm are *incompleteness* and *noise*. The notion of completeness assumed by the α-algorithm relates to the $>$ relation from which the other relations are derived. The number of required different cases increases with the factorial of the number of potentially concurrent tasks. Often, this number is low. But already for 10 concurrent tasks, we require $10! = 3,628,800$ cases. Therefore, it is desirable to use algorithms that can explicitly distinguish likely and unlikely behavior in order to generalize when logs are not complete. This direction also helps in addressing problems with noise. Event logs often include cases with a missing head, a missing tail, or a missing intermediate episode. Furthermore, there may be logging errors with events being swapped or recorded twice. Such unlikely behavior should not distort the process discovery result.

Several approaches have been defined for addressing problems of completeness and noise. In general, they try to balance four essential quality criteria, namely fitness, simplicity, precision, and generalization. *Fitness* refers to the degree of log behavior that a process model is able to replay. It can be defined based on the fraction of event patterns represented by the model or based on the fraction of cases that can be replayed in the model. *Simplicity* means that the resulting process model should be readily understandable. It can be measured using different complexity metrics for process models such as model size or degree of structuredness. *Precision* refers

to the degree of behavior that is allowed by the model, but not observed in the logs. We can easily create a process model that allows the execution of all tasks in any arbitrary order with potential repetition. However, we can hardly learn any specifics of the process from such a model. *Generalization* refers to the ability of a process model to abstract from the behavior that is documented in the logs. A discovery technique that is able to generalize helps to work with incomplete behavior.

Exercise 10.5 Draw a model in BPMN with which you can replay any execution sequence that includes tasks a, b, c, d, e. Furthermore, discuss the fitness, simplicity, precision, and generalization of such a model for the trace $\langle a, b, c, d, e \rangle$.

10.3 Performance Analysis

In Sect. 7.1 we introduced the four performance dimensions of time, cost, quality and flexibility. In turn, Chap. 8 demonstrated that these four dimensions form a Devil's Quadrangle when we try to improve a process. These performance measures are usually considered to be generally relevant for any kind of business. Beyond this general set, a company should also identify specific measures. Often, the measures are industry-specific, like profit per square-meter in gastronomy, return rate in online shopping or customer churn in marketing. Any specific measure that a company aims to define should be accurate, cost-effective and easy-to-understand. Here, we focus on the four general performance measures of time, cost, quality and flexibility. The question of this section is how we can spot that a process does not perform well according to one of these dimensions. Event logs provide us with very detailed data that is relevant to performance. We will describe techniques that help us to measure and to visualize potential performance problems that relate to time, cost, quality and flexibility.

10.3.1 Time Measurement

Time and its more specific measures *cycle time* and *waiting time* are important general performance measures. Event logs typically show timestamps such that they can be used for time analysis. Time analysis is concerned with the temporal occurrence and probabilities of different types of event. The event logs of a process generally relate each event to the point in time of its occurrence. Therefore, it is straightforward to plot events on the time axis. Furthermore, we can utilize classifiers to group events on a second axis. A classifier typically refers to one of the attributes of an event, like case ID or participant ID. There are two levels of detail for plotting events in a diagram: *dotted charts* using the timestamp to plot an event, and *timeline chart* showing the duration of a task and its waiting time.

The dotted chart is a simple, yet powerful visualization tool for event logs. Each event is plotted on a two-dimensional canvas with the first axis representing its

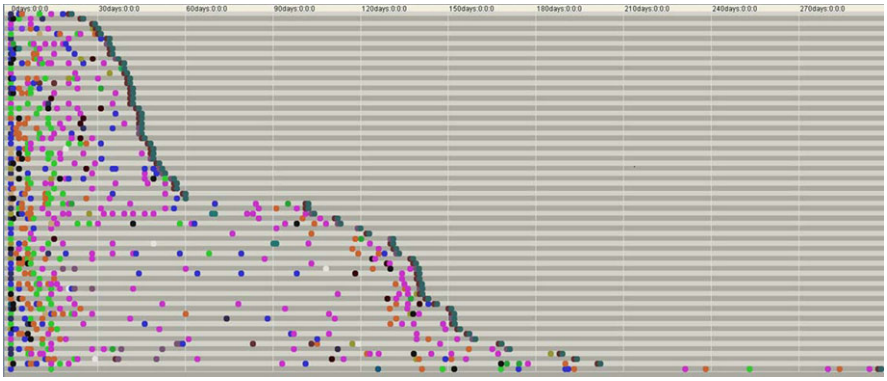

Fig. 10.9 Dotted chart of log data

occurrence in time and the second axis as its association with a classifier like a case ID. There are different options to organize the first axis. Time can be represented either *relative* such that the first event is counted as zero, or *absolute* such that later cases with a later start event are further right in comparison to cases that started earlier. The second axis can be sorted according to different criteria. For instance, cases can be shown according to their historical order or their overall cycle time.

Figure 10.9 shows the dotted chart of the logs of a healthcare process. The events are plotted according to their relative time and sorted according to their overall cycle time. It can be seen that there is a considerable variation in terms of cycle time. Furthermore, the chart suggests that there might be three distinct classes of cases: those that take no more than 60 days, those taking 60 to 210 days, and a small class of cases taking longer than 210 days. Such an explorative inspection can provide a good basis for a more detailed analysis of the factors influencing the cycle time.

Exercise 10.6 Draw a dotted chart of the event log in Fig. 10.1 showing relative cycle time and being sorted by cycle time.

The temporal analysis of event logs can be enhanced with further details if a corresponding process model is available and tasks can be related to a start and to an end event. The idea is to utilize the concept of token replay for identifying the point in time when a task gets activated. For tasks in a sequence, the activation time is the point in time when the previous task completed. For tasks after an AND-join, this is the point in time when all previous tasks have completed. For XOR-joins and splits it is the point when one of the previous tasks completes.

Using this information, we can plot a task not as a dot but as a bar in a timeline chart (see Fig. 10.10). A timeline chart shows a waiting time (from activation until starting) and a processing time (from starting until completion) for each task. The timelines of each task can be visualized in a similar way as a dot in the dotted chart. The timeline chart is more informative than the dotted chart since it shows the duration of the tasks. Furthermore, it is helpful to see the waiting times. Both

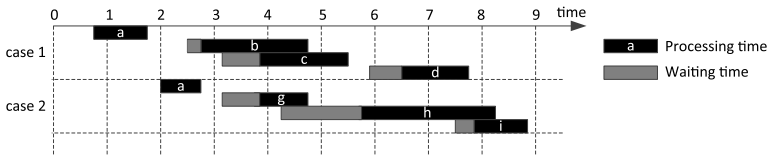

Fig. 10.10 Timeline chart of log data like PM 232

pieces of information are a valuable input for quantitative process analysis. When thousands of cases are available as a log, one can estimate the distribution of waiting time and processing time of each task. In this way, bottlenecks with long waiting times can be spotted and, similarly, tasks that are most promising to focus redesign efforts upon. Furthermore, this information can also be used for predicting execution times of running process instances, which is helpful for process monitoring.

Exercise 10.7 Calculate the waiting times required by a timeline chart for the event log in Fig. 10.1 using the process model of Fig. 10.7.

10.3.2 Cost Measurement

In a process context, cost measurement is mainly related to the problem of assigning indirect costs to cases. After all, direct costs like the purchasing costs of four wheels which are assembled on a car can be easily determined. Indirect labor or machine depreciation are more difficult. In accounting, the concept of *Activity-based Costing* (*ABC*) was developed to more accurately assign indirect costs to products and services, and to individual customers. The motivation of ABC is that human resources and machinery are often shared by different products and services, and they are used to serve different customers. For instance, the depot of BuildIT rents out expensive machinery such as bulldozers to different construction sites. On the one hand, that involves costs in terms of working hours of the persons working at the depot. On the other hand, machines like bulldozers lose in value over time and require maintenance. The idea of ABC is to use activities for distributing the indirect costs, e.g. associated with the depot.

Example 10.1

According to Fig. 1.6 in Chap. 1, the rental process of BuildIT contains five major activities. We observe the following durations in the event logs for the case of a bulldozer rental requested on 21st August:

- "Submit equipment rental request" is conducted by the site engineer. It takes the engineer 20 minutes to fill in the form on paper. The production of each paper form costs €1. The site engineer gets an annual salary of €60,000.
- The clerk receives the form and selects suitable equipment and checks the availability. Both activities together take 15 minutes. The clerk works at an annual rate of €40,000.
- The works engineer reviews the rental request (annual salary of €50,000). This review takes 10 minutes.

- The clerk is also responsible for sending a confirmation including a purchase Order for renting the equipment, which takes 30 minutes.

In order to work with these numbers we have to make certain assumptions. First, at BuildIT the actual working year contains 250 working days of 8 hours. Furthermore, all employees receive health insurance and pension contributions of 20 % on top of their salary. Finally, people are on average 10 days on a sick leave per year. Taking this into account, we can calculate the labor cost of each participant per minute as $\frac{salary \times 120\ \%}{(250-10)\times 8\times 60}$. This is for the site engineer €0.63 per minute, for the clerk €0.42 per minute, and for the works engineer €0.52 per minute. Altogether, this case created costs of 20 minutes × €0.63 per minute + (15 + 30) minutes × €0.42 per minute + 10 minutes × €0.52 per minute, which sums up to €36.70.
Now consider a case of a street construction process. We observe the following durations from the event logs for these two activities:

- "Prepare the foundation" is conducted by four worker. It took one week at a specific construction case. The used excavator costs €100,000. It is written off in five years and has annual maintenance costs of €5,000. A worker gets an annual salary of €35,000.
- "Tar the road" is conducted by six workers. It took two days in this case. The tarring machine costs €200,000, is also written off in five years and costs €10,000 annual maintenance.

For this case, we can also take the costs of the machinery into account. The labor cost per day is €175 for one worker. The excavator costs €20,000 + 5,000 per annum for write-off and maintenance, which is €104.17 per working day. For the preparation of the foundation, this amounts to 4 × 5 × €175 + 5 × €104.17 = €4,020.85. For the tarring of the road, the costs are 6 × 2 × €175 + 2 × €208.34 = €2,516.68.

Exercise 10.8 Consider that the paper form is printed by the site engineer in the rental process, that the printer costs €300 written off in three years, and that a pile of 500 pieces of paper costs €10. Why could it make sense to include these costs in the calculation, why not?

An inherent problem of ABC is the detail of data that is required for tracking the duration of activities like renting out equipment or approving rental requests. Event data stored in process-aware information systems can help to provide such data. Also, technologies like Radio-frequency identification (RFID), which helps to track physical objects based on little RFID chips attached to it, are promising to overcome the tracking problem of ABC. Some systems only keep track of activity completion. However, ABC also requires the start of activities to be kept. This can be achieved by tracking timestamps of the point in time when a resource starts working on a specific task. What is important to take into account here is the cost of achieving additional transparency. There is a trade-off, and once it becomes overly expensive to gain more transparency, it is good to not include those costs in the calculation.

10.3.3 Quality Measurement

The quality of a product created in a process is often not directly visible from execution logs. However, a good indication is to check whether there are repetitions in the

execution logs, because they typically occur when a task has not been completed successfully. Repetitions can be found in sequences of task. In Chap. 7, we saw that the loop of a rework pattern increases the cycle time of a task to $CT = \frac{T}{1-r}$ in comparison to T being the time to execute the task only once. The question is now how we can determine the repetition probability r from a series of event logs?

The first part of the answer to this question can be given by reformulating the equation such that it is solved for r. By multiplication with $1 - r$ we get $CT - r \times CT = T$. Subtraction of CT yields $-r \times CT = T - CT$, which can be divided by $-CT$ resulting in

$$r = 1 - \frac{T}{CT}.$$

Both CT and T can now be determined using the data of the event logs. Consider the five cases in which we observe the following execution times for task a:

1. 5 minutes, 10 minutes.
2. 10 minutes.
3. 20 minutes, 6 minutes, 10 minutes.
4. 5 minutes.
5. 10 minutes, 10 minutes.

The cycle time CT of a can now be calculated as the average execution time of a per case, while the average execution time T is the average execution time of a per instantiation. Both can be determined based on the sum of all executions of a, which is 86 minutes here. We have five cases, such that $CT = 86/5 = 17.2$. Altogether, a is executed nine times yielding $T = 86/9 = 9.56$. Hence, we get $r = 1 - \frac{86/9}{86/5} = 1 - \frac{5}{9} = 0.44$. Of course, this calculation is only an approximation of the real value for r. It builds on the assumption that the duration of a task always follows the same distribution, no matter if it is the first, the second or another iteration.

Exercise 10.9 Determine the probability of repetition r for the following execution times of task b:

1. 20 minutes, 10 minutes.
2. 30 minutes.
3. 30 minutes, 5 minutes.
4. 20 minutes.
5. 20 minutes, 5 minutes.
6. 25 minutes.

Also explain why the value is misleading for these logs.

In some information systems it might be easier to track repetition based on the assignment of tasks to resources. One example are *ticketing systems* that record which resource is working on a case. Also, the logs of these systems offer insights

into repetition. A typical process supported with ticketing systems is incident resolution. For example, an incident might be a call by a customer who complains that the online banking system does not work. Such an incident is recorded by a dedicated participant, e.g. a call center agent. Then, it is forwarded to a first-level support team who tries to solve the problem. In case the problem turns out to be too specific, it is forwarded to a second-level support team with specialized knowledge in the problem domain. In the best case, the problem is solved and the customer notified. In the undesirable case, the team identifies that the problem is within the competence area of another team. This has the consequence that the problem is rooted back to the first-level team. Similar to the repetition of tasks, we now see that there is a repeated assignment of the problem to the same team. According log information can be used to determine how likely it is that a problem is rooted back.

10.3.4 Flexibility Measurement

Flexibility refers to the degree of variation that a process permits. This flexibility can be discussed in relation to the event logs the process produces. For the company owning the process, this is important information in order to compare the desired level of flexibility with the actual flexibility. It might turn out that the process is more flexible than what is demanded from a business perspective. This is the case when flexibility can be equated with lack of standardization. Often, the performance of processes suffers when too many options are allowed. Consider again the process for renting equipment at BuildIT. The process requires an equipment rental request form to be sent by e-mail. Some engineers, however, prefer to call the depot directly instead of filling the form. Since these engineers are often highly distinguished, it is not easy for the clerk to turn down these calls. As a result, the clerk fills out the form while being on the phone. Not only does this procedure take more time, but also, due to noise at the construction site, it increases the likelihood of errors. In practice, this means that the rental process has two options to submit a request: by form (the standard procedure) and via the phone.

Partially, such flexibility as described above can be directly observed in the event logs. We have seen that the workflow log of a process plays an important role for automatic process discovery. It can also be used to assess the flexibility of the process. The workflow log summarizes the essential behavior of the process. As it defines each execution as a sequence of tasks, it abstracts from the temporal distance between them. In this way, the workflow log contains a set of traces that have a unique sequence. This means that if two executions contain the same sequence of tasks, then they only result in a single trace to be included in the workflow log. This abstraction of process executions in terms of a workflow log makes it a good starting point for discussing flexibility. Accordingly, the number of *distinct executions DE* can be defined based on a workflow log L as

$$DE = |L|.$$

Exercise 10.10 Consider the event logs of the order fulfillment process in Fig. 10.1. What is the number of distinct executions DE?

The question arises whether the number of distinct executions always gives a good indication for flexibility. At times, the number of distinct executions might give an overly high quantification of flexibility. This might be the case for processes with concurrency. Such processes can be highly structured, but having only a small set of concurrent tasks results in a rich set of potential execution sequences. Consider the process model constructed by the α-algorithm in Fig. 10.7. The tasks i and h are concurrent to j and k. Indeed, there are six options to execute them:

1. i, h, j, k
2. j, k, i, h
3. i, j, k, h
4. j, i, h, k
5. i, j, h, k and
6. j, i, k, h

While the order is not strict, all of them must be executed. Therefore, it might be a good idea to additionally consider whether a task is optional. If T refers to the number of tasks that appear in the workflow log, then the set T_{opt} contains those tasks that are optional. Optionality according to the log means that for a particular task there exists at least one trace in which it does not occur. For the logs of Fig. 10.1, we observe that the tasks b to f depend upon the availability of raw materials. We can quantify the degree of *optionality* OPT as

$$OPT = \frac{T_{\text{opt}}}{T}.$$

We can also approach the question of flexibility from the perspective of the automatically discovered process model. This has some advantages since the degree of optionality does not reveal how many decisions have to be taken. If we consider the process model constructed by the α-algorithm in Fig. 10.7, we see that one decision node is included (XOR-split). It distinguishes the situation whether the product is in stock or not. This observation can be quantified as the number of *discovered decision points* (DDP). For instance, the α-algorithm can be used to this end.

10.4 Conformance Checking

While performance analysis is concerned with measuring performance indicators, conformance checking is concerned with the question whether or not the execution of a process follows predefined rules or constraints. This question can be answered by inspecting the event logs. If a particular constraint does not hold, we speak of a *violation*. Conformance checking is concerned with identifying these violations and making statements about the extent of them altogether. This information provides important insights for the process owner. Apparently, when rules are not followed,

one should take corrective action. Violations might relate to one of the three process perspectives of control flow, data and resources in isolation or in combination. In the following, we describe how they can be specified.

10.4.1 Conformance of Control Flow

Conformance of control flow can be studied in two ways, either based on *explicit constraints* or based on a *normative process model*. Both can be related to each other, as many constraints can be automatically derived from a process model. First we will look into the approach assuming explicit constraints, after which we will discuss the use of a normative process model.

We focus on three types of control-flow related constraint: mandatoriness, exclusiveness, and ordering. All these three constraint types define how two activities are allowed to be related in a process. A company might want to define that certain activities are *mandatory* because they are required from a control perspective. Consider again the case of BuildIT and the equipment rental process. A works engineer is supposed to review the rental request. This activity serves as a control for securing that only appropriate equipment is rented. This measure might help to keep the rental costs in line. Such a control activity is a likely candidate for being a mandatory activity. On the level of the logs, mandatory activity violations can be found by searching for traces in which they are left out. *Exclusiveness* might be specified for activities that relate to a decision. If, for instance, a rental request is rejected, BuildIT wants to make sure that there is no way to overwrite this decision. On the level of the log, this means that it shall never be possible that a rejection of a request is followed by an approval. This exclusiveness can be checked by searching for traces in which both exclusive activities appear. The *order* of activities might be of specific importance to balance the performance of the process. In the equipment rental process, first the availability of the requested equipment is checked before the request is reviewed. This order constraint helps to relieve the workload of the works engineer who is supposed to review the request. Obviously, it is a waste of effort to review requests that cannot be met because the equipment is not available. Violations to order constraints can be found by searching for traces with the activities appearing in the wrong order.

Exercise 10.11 Consider the event logs of the order fulfillment process in Fig. 10.1. Which activities can be considered mandatory and exclusive to one another?

Conformance of control flow can also be checked by comparing the behavior observed in the logs with a normative process model. The idea here is to replay each trace in the workflow log and to record at each step whether an activity was allowed to be executed according to the rules of the model. Typically, this requires the replay of the logs in the process model. Based on the transition rules of BPMN, we can replay the case $\langle a, b, g, i, j, k, l \rangle$ on the model shown in Fig. 10.11. In the initial

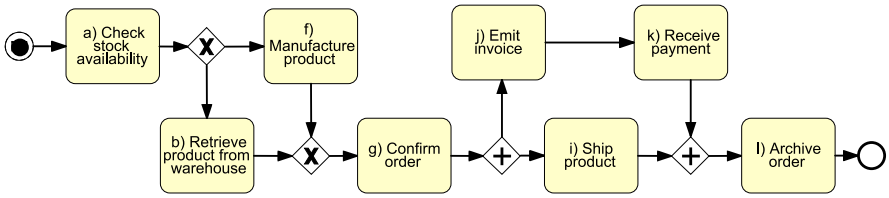

Fig. 10.11 BPMN model with token on start event for replaying the case $\langle a, b, g, i, j, k, l \rangle$

state, the process has a token on the start event. Once the case is started, this token moves to the output arc of the start event. This arc leads to activity a ("Check stock availability"), which means that the token enables this activity to be executed. The token moves to this activity while it is executed, and it is forwarded to the output arc once the activity is completed. Now, the XOR-split is activated, which means a decision has to be taken to either continue with b ("Retrieve product from warehouse") or with f ("Manufacture product"). For the considered case, we continue with b. In the same way, we can continue. After g ("Confirm order") has been completed, we arrive at an AND-split. An AND-split consumes a token from its input arc and creates one token on each of its output arcs. As a result, we have two tokens afterwards: one enabling i ("Ship product") and one enabling j ("Emit invoice"). In this state we can proceed either with i or j. These activities are concurrent. In order to replay the case, we first execute i and j afterwards. Once i and later k is completed, the AND-join is allowed to proceed. One token on each of its input arcs is required for that. Both these tokens are consumed and a single token is created on its output arc. As a result, l ("Archive order") can be executed.

Based on the concept of token replay, we can also assess the conformance of a trace to a process model. Consider the case $\langle a, b, i, j, k, l \rangle$ in which the confirmation of the order has been omitted. The idea is to compare at each step the number of tokens that are required for replaying an activity with tokens that are actually available. At each step, we might observe two situations of conformance and two of non-conformance. In case of conformance, we can count the following four facts for tokens:

- p: the number of tokens that are correctly produced
- c: the number of tokens that are correctly consumed
- m: the number of tokens that are missing for executing the next activity in the log, and
- r: the number of tokens remaining unconsumed after executing the final activity in the log

Figure 10.12 first shows the state before replaying b of the case $\langle a, b, i, j, k, l \rangle$. After replaying b, there is a token available to execute g, but it is not consumed. Instead, there is a missing token for firing the AND-split, which would activate i and j. The figure also shows the number of correctly produced and consumed tokens for each step until the completion. Using the four measures c, p, m and r, we can calculate the fitness measure as an indicator of conformance. It is defined

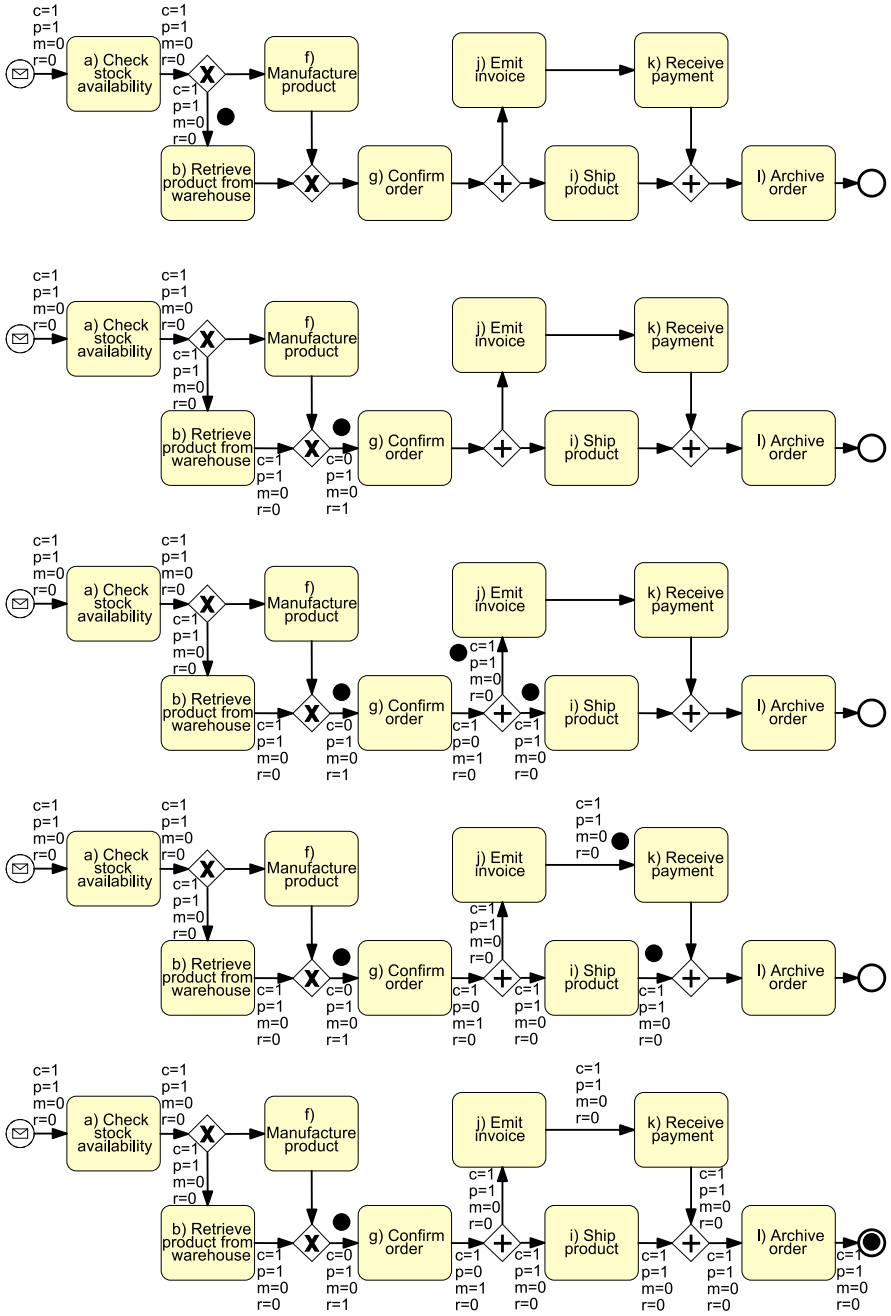

Fig. 10.12 Replaying the non-conforming case $\langle a, b, i, j, k, l \rangle$

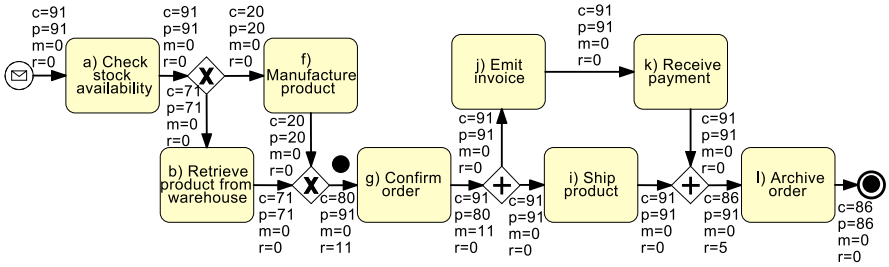

Fig. 10.13 Result of replaying cases in the process model

based on the fraction of missing tokens to correctly consumed tokens ($\frac{m}{c}$) and the fraction of remaining tokens to produced tokens ($\frac{r}{p}$) as

$$\mathit{fitness} = \frac{1}{2}\left(1 - \frac{m}{c}\right) + \frac{1}{2}\left(1 - \frac{r}{p}\right).$$

For our values $c = 12$, $p = 12$, $m = 1$ and $r = 1$, we get a fitness of $\frac{1}{2}(1 - \frac{1}{12}) + \frac{1}{2}(1 - \frac{1}{12}) = 0.8333$. When we consider a set of cases, not just a single case, we can easily calculate the fitness in the same way. The idea is to simply continue counting c, p, m and r by replaying the next case in the process model. Once we have replayed all cases, we get the resulting overall fitness of this set of cases.

The results of such a conformance analysis can be interpreted in two ways. First, we can use the overall fitness measure to get an idea of how accurately the process model matches the actually observed behavior as reflected by the set of cases. While the fitness as an overall measure is useful to this end, it does not help us to analyze the deviations in more detail. Therefore, and secondly, we can inspect at which arcs of the process model we have encountered missing or remaining tokens. Figure 10.13 shows the corresponding numbers from replaying several cases in the process model. It can be seen that apparently most deviations relate to activity g and some to activity l. This information can be utilized to interview process participants why g has been omitted for some cases. The goal of such an inquiry would be to find whether this omission is desirable. The question is whether the process participants have found a more efficient way to handle the confirmation, or whether an omission has to be considered bad practice which should be further discouraged. For the case of the archival (activity l), the omission is likely to be bad practice.

10.4.2 Conformance of Data and Resources

Beyond constraints on the control flow, there are often additional constraints on data and resources. Consider the situation that an expensive caterpillar is requested for a construction site of BuildIT. Many companies have extra rules for high expenses

or risky commitments. In the case of BuildIT, the rent of the caterpillar requires the additional signature of a manager. Similar cases can be found in banks where a loan of more than €1,000,000 might require the additional sign-off from a director. There might also be constraints that providing a loan to a black-listed applicant is simply not allowed at all. Such constraints can be checked by searching for cases in the log where a certain data field takes a forbidden value.

The example of the additionally required signature already points to a combination of data and resource constraints. If a certain amount is exceeded, then there is a dedicated resource who is required to approve. However, there are also constraints that purely refer to the resource perspective. Participants usually require *permissions* to execute certain activities. For instance, the person signing off the caterpillar rent has to be a manager, and it is not allowed that this person is a construction worker. Typically, permissions are bundled for specific *roles*. For example, this can be done by explicitly listing what a manager and a construction worker are allowed to do. Violations of permissions can be checked by searching for each activity conducted by a participant whether or not an appropriate role or permission existed. Specific control rules which require two different persons to approve a business transaction are called *separation of duties* constraints. These rules do not necessarily involve supervisors. For example, it might be OK with a particular bank if a loan of €100,000 is signed by two bank clerks, while a €1,000,000 loan requires the signature of a clerk and a director. Such separation of duties constraints can be checked by searching for cases in the log where the same participant or two participants with the same role have approved the same transaction.

Exercise 10.12 Consider the intake process after the medical file redesign, which is shown in Fig. 8.3. Which separation of duties constraints would you define for this process?

10.5 Recap

In this chapter we discussed the topic of process intelligence. The starting point for process intelligence is the availability of event logs and, in general, data on the execution of processes. These data provides the basis for process monitoring and process controlling. Such event logs have to refer to a specific case, a task and a point in time in order to facilitate process-related analysis. In many cases, it is a challenge to extract data in such a way that it can readily support process intelligence.

An important area of process intelligence is automatic process discovery. Corresponding techniques like the α-algorithm yield process models describing how a process runs in reality according to the log data. The α-algorithm is a good example for how automatic process discovery works. However, it has some limitations in terms of robustness, which are addressed by more recent process mining algorithms.

Event logs also support the assessment of the performance of a process. We discussed the four dimensions of the Devil's Quadrangle. The time dimension of a process can be visualized as a dotted chart and further inspected using a timeline chart. Then, we demonstrated that the calculation of costs for a process heavily depends upon the level of detail with which execution times of tasks are captured. We turned to quality and related it to the number of repetitions that are encountered in a log. Finally, we discussed ways of quantifying the flexibility of a process based on the number of distinct executions, optionality and discovered decision points.

The area of conformance checking can be related to various types of constraint. There are different types of constraint that can be checked for control flow including mandatoriness, exclusiveness and ordering of activities. A notion of fitness can be calculated based on replaying the logs in a normative process model. Finally, we discussed important types of constraint for the data and the resource perspectives. Often, these are intertwined. They typically relate to additional requirements for approval beyond a certain monetary threshold or to enforce separation of duties in order to control risks of a business transaction.

10.6 Solutions to Exercises

Solution 10.1 It might be the case that parts of the production process are administered using different information systems. Accordingly, the event logs have to be integrated. In terms of correlation, this means that case identifiers from different systems have to be matched. If the timestamps of the events are recorded in different time zones, they have to be harmonized. Shipment might not be arranged by Airbus. Therefore, it might be the case that different snapshots of transportation might not be accessible. Also, information systems might not be directly producing case-related event logs. The data would then have to be extracted from the databases of these systems. Finally, the events might be recorded at diverging levels of granularity, ranging from a detailed record of production steps to coarse-granular records of transportation stages.

Solution 10.2 The workflow log considers the order of events for each case. We use letters a to l for referring to the tasks. The workflow log L containing three elements as the first and the fourth execution sequence are the same. Therefore, we get $L = [\langle a, b, g, h, j, k, i, l\rangle, \langle a, c, d, e, f, g, j, h, i, k, l\rangle, \langle a, c, f, g, j, h, i, k, l\rangle]$.

Solution 10.3 The following basic relations can be observed:

$a > b$	$h > j$	$i > l$	$d > e$	$g > j$	$i > k$
$b > g$	$j > k$	$a > c$	$e > f$	$j > h$	$k > l$
$g > h$	$k > i$	$c > d$	$f > g$	$h > i$	$c > f$

The matrix shows the resulting relations.

	a	b	c	d	e	f	g	h	i	j	k	l
a	#	→	→	#	#	#	#	#	#	#	#	#
b	←	#	#	#	#	#	→	#	#	#	#	#
c	←	#	#	→	#	→	#	#	#	#	#	#
d	#	#	←	#	→	#	#	#	#	#	#	#
e	#	#	#	←	#	→	#	#	#	#	#	#
f	#	#	←	#	←	#	→	#	#	#	#	#
g	#	←	#	#	#	←	#	→	#	→	#	#
h	#	#	#	#	#	#	←	#	→	‖	#	#
i	#	#	#	#	#	#	#	←	#	#	‖	→
j	#	#	#	#	#	#	←	‖	#	#	→	#
k	#	#	#	#	#	#	#	#	‖	←	#	→
l	#	#	#	#	#	#	#	#	←	#	←	#

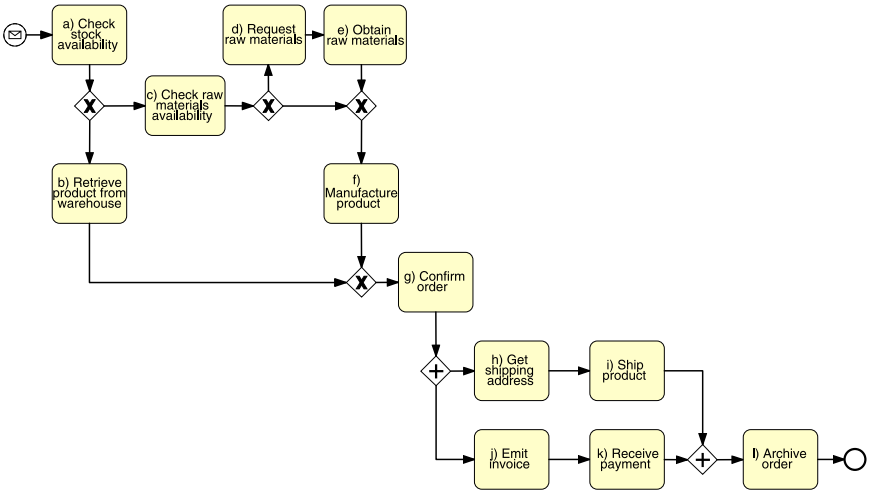

Fig. 10.14 Process model constructed by the α-algorithm

Solution 10.4 The α-algorithm stepwise yields the following sets:

1. $T_L = \{a, b, c, d, e, f, g, h, i, j, k, l\}$.
2. $T_I = \{a\}$.
3. $T_O = \{l\}$.
4. $X_L = Z_1 \cup Z_2$ with $Z_1 = \{a \to b, a \to c, b \to g, c \to d, c \to f, d \to e, e \to f, f \to g, g \to h, g \to j, h \to i, i \to l, j \to k, k \to l\}$ and $Z_2 = \{a \to (b\#c), c \to (d\#f), (c\#e) \to f, (b\#f) \to g\}$
5. $Y_L = Z_2 \cup \{d \to e, g \to h, g \to j, h \to i, j \to k, i \to l, k \to l\}$.
6. Add start event pointing to a and end event following after l.
7. Construct process model based on Y_L with XOR- and AND-gateways.
8. Return process model, see Fig. 10.14.

Solution 10.5 Include a, b, c, d, e in an XOR-block and put this XOR-block in an XOR-loop. This model shows a perfect fitness to the execution sequence as it is able to replay any occurrence of a to e at any stage. The model is not completely simple as it includes four gateways for characterizing the behavior of five activities. Also, the model is not very precise in this sense that it does not introduce specific constraints on the behavior: any occurrence of a to e is allowed at any stage. Generalization refers to the ability of the model to abstract. As the model does not constrain the behavior, there is hardly any general insight that we can learn from it.

Solution 10.6 First, we have to determine the cycle time. Case 1 takes roughly two hours less than seven days, case 2 one hour less than six days, case 3 takes four days and 20 minutes, and case 4 takes four days and six hours. Therefore, the relative order must be case 3, case 4, case 2 and case 1. Each event has to be visualized according to the time elapsed since the first event of the case.

Solution 10.7 The timeline diagram follows the same principle as the dotted chart. Additionally, it shows waiting times as soon as an activity becomes enabled in the process model.

Solution 10.8 In general, it is a good idea to include all expenses in the cost calculation as it provides transparency on what is spent where. In this case, however, it might not be a good idea since the cost per paper is relatively low with €0.02. It has to be kept in mind though that the decision whether to record certain costs should not be governed by the unit piece, but by the relative impact on cost calculation. A cost of paper of €0.02 is small as compared to overall process costs of several thousand Euros. However, it can be relatively high if the overall process produces €0.30 costs per instance and millions of instances are processed per day. Think of the situation of processing bank transactions using paper forms versus using online banking. The high amount of transactions per day might result in considerable costs per year.

Solution 10.9 The formula yields $r = 1 - \frac{T}{CT} = 1 - \frac{18.3}{27.5} = 0.333$. Apparently, this result is misleading for the figures because every second task required rework. However, the rework time was in general much smaller as the first try duration. This has the effect that T appears to be relatively small in comparison to CT, which results in a low value for r.

Solution 10.10 We have to consider four different cases. However, as the first and the fourth case show the same sequence of tasks, there are three distinct executions.

Solution 10.11 There are several activities that can be observed in all cases. These mandatory activities are a, g, h, i, j, k, l. The other activities b, c, d, e, f are not mandatory. Exclusiveness relationships hold between b and c, d, e, f pairwise.

Solution 10.12 The intake process demands meeting two intakers. The persons conducting "Meet with first intaker" and "Meet with second intaker" should be mutually

exclusive. Therefore, we can identify a separation of duty constraint that the participant conducting "Meet with first intaker" should be another one as the participant conducting "Meet with second intaker".

10.7 Further Exercises

Exercise 10.13 Download the process mining tool ProM, write down the workflow log L in XES, and run the α-algorithm.

Exercise 10.14 Apply the α-algorithm and document the different steps for the workflow log L containing four cases $L = [\langle a, b, c, d\rangle, \langle a, b, d, c\rangle, \langle a, b, d, c, e\rangle, \langle a, c, d, e\rangle]$.

Exercise 10.15 Apply the α-algorithm and document the different steps for the workflow log L containing four cases $L = [\langle a, b, c, d, e, f\rangle, \langle a, b, d, c, e\rangle, \langle a, b, d, c, e, f\rangle, \langle a, b, c, d\rangle]$.

Exercise 10.16 Consider there is an AND-split in a process model with two subsequent activities a and b. Which kind of pattern do these activities show on the timeline chart?

Exercise 10.17 Consider the workflow log, which you created for Exercise 10.2 from the cases shown in Fig. 10.1. Replay these logs in the process model of Fig. 10.13. Note down consumed, produced, missing and remaining tokens for each arc, and calculate the fitness measure. Assume that activities not shown in the process model do not change the distribution of tokens in the replay.

10.8 Further Reading

Process intelligence relates to various streams of current research. The workflow resource patterns give an extensive account of strategies that can be used for effectively assigning work items to participants at the time of execution. Process mining has been a very influential area of research in the recent years, offering various tools and techniques for process intelligence. An excellent overview of this research area is provided in the book by van der Aalst on process mining [94]. Among others, it summarizes work on leveraging shortcomings of the α-algorithm, namely the heuristic miner, the fuzzy miner, and the genetic miner. The idea of the *heuristic miner* is to generate so-called heuristic nets in which the arc probabilities are included. Using appropriate threshold values, the user can eliminate unlikely behavior. The *fuzzy miner* takes correlations between events and behavior into account. In this way, tasks can be clustered and inspected at different levels of abstraction. The *genetic miner* generates a population of potential models and stepwise manipulates

these models using change operations and quality metrics in order to improve the approximation of the log. This process is repeated until a model with a predefined quality level is found.

Various guiding principles and research challenges for process mining have been formulated by the IEEE Task Force on Process Mining in the Process mining manifesto [37]. There are dedicated tools that help with putting process mining into practice (see http://www.processmining.org). Process mining tools usually cover automatic process discovery and techniques for conformance checking.

A good summary of current research on process performance analysis is provided by the BPM Handbook [102, 103] and, most notably, its chapters on process performance management and on business process analytics [33, 111]. A good overview on process controlling and its relationship to process automation is the book by zur Muehlen [110]. More generally, the book by Harmon provides a good perspective on how to define process measures within a process governance framework [32]. A good book on foundations on performance from an operations management point of view is the book by Anupindi et al. [4]. Various metrics for event logs are discussed in the Ph.D. thesis of Günther [25].

Conformance checking is also nicely covered in the book by van der Aalst on process mining [94]. Research with a focus on generating constraints from process models for conformance checking is conducted by Weidlich et al. [104, 105]. Case studies on process mining and conformance checking are, among others, reported in [13, 22, 97]. Separation of duties is traditionally discussed as a topic within the research area of role-based access control (RBAC). A summary of essential RBAC concepts and their relationship to process-centered concepts is provided in [90].