Chapter 8
Conformance Checking

After covering control-flow discovery in depth in Part III, this chapter looks at the situation in which both a process model and an event log are given. The model may have been constructed by hand or may have been discovered. Moreover, the model may be normative or descriptive. Conformance checking relates events in the event log to activities in the process model and compares both. The goal is to find commonalities and discrepancies between the modeled behavior and the observed behavior. Conformance checking is relevant for business alignment and auditing. For example, the event log can be replayed on top of the process model to find undesirable deviations suggesting fraud or inefficiencies. Moreover, conformance checking techniques can also be used for measuring the performance of process discovery algorithms and to repair models that are not aligned well with reality.

8.1 Business Alignment and Auditing

In Sect. 2.4, we introduced the terms Play-In, Play-Out, and Replay. *Play-Out* is the classical use of process models; the model generates behavior. For instance, by playing the "token game" in a WF-net, example behaviors can be generated. Simulation and workflow engines use Play-Out to analyze and enact process models. *Play-In* is the opposite of Play-Out, i.e., example behavior is taken as input and the goal is to construct a model. The discovery techniques presented in Chaps. 6 and 7 can be used for Play-In. *Replay* uses both an event log and a process model as input, i.e., history is replayed using the model to analyze various phenomena. For example, in Chap. 9 we will show that replay can be used for analyzing bottlenecks and decision analysis. In Chap. 10, replay will be used to predict the behavior of running cases and to recommend suitable actions. In this chapter, we focus on *conformance checking* using replay.

Figure 8.1 illustrates the main idea of conformance checking. The behavior of a process model and the behavior recorded in an event log are compared to find commonalities and discrepancies. Such analysis results in *global conformance measures*

W. van der Aalst, *Process Mining*, DOI 10.1007/978-3-662-49851-4_8

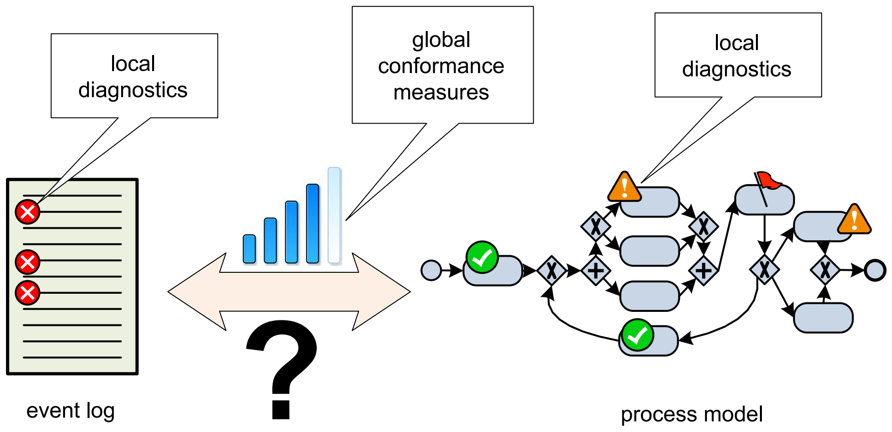


Fig. 8.1 Conformance checking: comparing observed behavior with modeled behavior. Global conformance measures quantify the overall conformance of the model and log. Local diagnostics are given by highlighting the nodes in the model where model and log disagree. Cases that do not fit are highlighted in the visualization of the log

(e.g., 85% of the cases in the event log can be replayed by the model) and *local diagnostics* (e.g., activity x was executed 15 times although this was not allowed according to the model). The interpretation of non-conformance depends on the purpose of the model. If the model is intended to be *descriptive*, then discrepancies between model and log indicate that the model needs to be improved to capture reality better. If the model is *normative*, then such discrepancies may be interpreted in two ways. Some of the discrepancies found may expose *undesirable deviations*, i.e., conformance checking signals the need for a better control of the process. Other discrepancies may reveal *desirable deviations*. For instance, workers may deviate to serve the customers better or to handle circumstances not foreseen by the process model. In fact, flexibility and non-conformance often correlate positively. For example, in some hospitals the phrase "breaking the glass" is used to refer to deviations that are recorded but that actually save lives. Nevertheless, even if most deviations are desired, it is important that stakeholders have insight into such discrepancies.

When checking conformance it is important to view deviations from two angles: (a) the model is "wrong" and does not reflect reality ("How to improve the model?"), and (b) cases deviate from the model and corrective actions are needed ("How to improve control to enforce a better conformance?"). Conformance checking techniques should support both viewpoints. Therefore, Fig. 8.1 shows deviations on both sides.

In Chap. 2, we related process mining to corporate governance, risk, compliance, and legislation such as the Sarbanes–Oxley Act (SOX) and the Basel II Accord. Corporate accounting scandals have triggered a series of new regulations. Although country-specific, there is a large degree of commonality between Sarbanes–Oxley (US), Basel II/III (EU), J-SOX (Japan), C-SOX (Canada), 8th EU Directive (EURO-SOX), BilMoG (Germany), MiFID (EU), Law 262/05 (Italy), Code Lippens

(Belgium), Code Tabaksblat (Netherlands), and others. These regulations require companies to identify the financial and operational risks inherent to their business processes, and establish the appropriate controls to address them. Although the focus of these regulations is on financial aspects, they illustrate the desire to make processes transparent and auditable. The ISO 9000 family of standards is another illustration of this trend. For instance, *ISO 9001:2008* requires organizations to model their operational processes. Currently, these standards do not force organizations to check conformance at the event level. For example, the real production process may be very different from the modeled production process. Nevertheless, the relation to conformance checking is evident. In this chapter, we take a more technological perspective and show concrete techniques for quantifying conformance and diagnosing non-conformance. However, before doing so, we briefly reflect on the relation between conformance checking, business alignment, and auditing.

The goal of *business alignment* is to make sure that the information systems and the real business processes are well aligned. People should be supported by the information system rather than work behind its back to get things done. Unfortunately, there is often a mismatch between the information system on the one hand and the actual processes and needs of workers and management on the other hand. There are various reasons for this. First of all, most organization use product software, i.e., generic software that was not developed for a specific organization. A typical example is the SAP system which is based on so-called "best practices", i.e., typical processes and scenarios are implemented. Although such systems are configurable, the particular needs of an organization may be different from what was envisioned by the product software developer. Second, processes may change faster than the information system, because of external influences. Finally, there may be different stakeholders in the organization having conflicting requirements, e.g., a manager may want to enforce a fixed working procedure whereas an experienced worker prefers to have more flexibility to serve customers better.

Process mining can assist in improving the alignment of information systems, business processes, and the organization. By analyzing the real processes and diagnosing discrepancies, new insights can be gathered showing how to improve the support by information systems.

The term *auditing* refers to the evaluation of organizations and their processes. Audits are performed to ascertain the validity and reliability of information about these organizations and associated processes. This is done *to check whether business processes are executed within certain boundaries set by managers, governments, and other stakeholders*. For instance, specific rules may be enforced by law or company policies and the auditor should check whether these rules are followed or not. Violations of these rules may indicate fraud, malpractice, risks, and inefficiencies. Traditionally, auditors can only provide *reasonable assurance* that business processes are executed within the given set of boundaries. They check the operating effectiveness of controls that are designed to ensure reliable processing. When these controls are not in place, or otherwise not functioning as expected, they typically *only check samples of factual data*, often in the "paper world".

However, today detailed information about processes is being recorded in the form of event logs, audit trails, transaction logs, databases, data warehouses, etc.

Therefore, it should no longer be acceptable to only check a small set of samples off-line. Instead, *all events in a business process can be evaluated and this can be done while the process is still running*. The availability of log data and advanced process mining techniques enables new forms of auditing [166]. Process mining in general, and conformance checking in particular, provide the means to do so.

8.2 Token Replay

In Sect. 6.4.3, we discussed four quality criteria: fitness, precision, generalization, and simplicity. These were illustrated using Fig. 6.24. In this figure one event log is given and four process models are shown. For each of these models, a subjective judgment is given with respect to the four quality criteria. As the models are rather extreme, the scores for the various quality criteria are evident. However, in a more realistic setting it is much more difficult to judge the quality of a model. This section shows how the notion of *fitness* can be quantified. Fitness measures "the proportion of behavior in the event log possible according to the model". Of the four quality criteria, fitness is most related to conformance.

To explain the various fitness notions, we use the event log L_{full} described in Table 8.1. This is the same event log as the one used in Fig. 6.24. There are 1391 cases in L_{full} distributed over 21 different traces. For example, there are 455 cases following trace $\sigma_1 = \langle a, c, d, e, h \rangle$, 191 cases following trace $\sigma_2 = \langle a, b, d, e, g \rangle$, etc.

Figure 8.2 shows four models related to event log L_{full}. WF-net N_1 is the process model discovered when applying the α-algorithm to L_{full}. WF-net N_2 is a sequential model that, compared to N_1, requires the examination (activity b or c) to take place before checking the ticket (activity d). Clearly, N_2 does not allow for all traces in Table 8.1. For example, $\sigma_3 = \langle a, d, c, e, h \rangle$ is not possible according to WF-net N_2. WF-net N_3 has no choices, e.g., the request is always rejected. Many traces in Table 8.1 cannot be replayed by this model, e.g., $\sigma_2 = \langle a, b, d, e, g \rangle$ is not possible according to WF-net N_3. WF-net N_4 is a variant of the "flower model": the only requirement is that traces need to start with a and end with g or h. Clearly, all traces in Table 8.1 can be replayed by N_4.

A naïve approach towards conformance checking would be to simply count the fraction of cases that can be "parsed completely" (i.e., the proportion of cases corresponding to firing sequences leading from [*start*] to [*end*]). Using this approach the fitness of N_1 is $\frac{1391}{1391} = 1$, i.e., all 1391 cases in L_{full} correspond to a firing sequence of N_1 ("can be replayed"). The fitness of N_2 is $\frac{948}{1391} = 0.6815$ because 948 cases can be replayed correctly whereas 443 cases do not correspond to a firing sequence of N_2. The fitness of N_3 is $\frac{632}{1391} = 0.4543$: only 632 cases have a trace corresponding to a firing sequence of N_2. The fitness of N_4 is $\frac{1391}{1391} = 1$ because the "flower model" is able to replay all traces in Table 8.1. This naïve fitness metric is less suitable for more realistic processes. Consider for instance a variant of WF-net N_1 in which places $p1$ and $p2$ are merged into a single place. Such a model will have a

Table 8.1 Event log L_{full}: a = *register request*, b = *examine thoroughly*, c = *examine casually*, d = *check ticket*, e = *decide*, f = *reinitiate request*, g = *pay compensation*, and h = *reject request*

Frequency	Reference	Trace
455	σ_1	$\langle a, c, d, e, h \rangle$
191	σ_2	$\langle a, b, d, e, g \rangle$
177	σ_3	$\langle a, d, c, e, h \rangle$
144	σ_4	$\langle a, b, d, e, h \rangle$
111	σ_5	$\langle a, c, d, e, g \rangle$
82	σ_6	$\langle a, d, c, e, g \rangle$
56	σ_7	$\langle a, d, b, e, h \rangle$
47	σ_8	$\langle a, c, d, e, f, d, b, e, h \rangle$
38	σ_9	$\langle a, d, b, e, g \rangle$
33	σ_{10}	$\langle a, c, d, e, f, b, d, e, h \rangle$
14	σ_{11}	$\langle a, c, d, e, f, b, d, e, g \rangle$
11	σ_{12}	$\langle a, c, d, e, f, d, b, e, g \rangle$
9	σ_{13}	$\langle a, d, c, e, f, c, d, e, h \rangle$
8	σ_{14}	$\langle a, d, c, e, f, d, b, e, h \rangle$
5	σ_{15}	$\langle a, d, c, e, f, b, d, e, g \rangle$
3	σ_{16}	$\langle a, c, d, e, f, b, d, e, f, d, b, e, g \rangle$
2	σ_{17}	$\langle a, d, c, e, f, d, b, e, g \rangle$
2	σ_{18}	$\langle a, d, c, e, f, b, d, e, f, b, d, e, g \rangle$
1	σ_{19}	$\langle a, d, c, e, f, d, b, e, f, b, d, e, h \rangle$
1	σ_{20}	$\langle a, d, b, e, f, b, d, e, f, d, b, e, g \rangle$
1	σ_{21}	$\langle a, d, c, e, f, d, b, e, f, c, d, e, f, d, b, e, g \rangle$

fitness of $\frac{0}{1391} = 0$, because none of the traces can be replayed. This fitness notion seems to be too strict as most of the model seems to be consistent with the event log. This is especially the case for larger process models. Consider, for example, a trace $\sigma = \langle a_1, a_2, \dots a_{100} \rangle$ in some log L. Now consider a model that cannot replay σ, but that can replay 99 of the 100 events in σ (i.e., the trace is "almost" fitting). Also consider another model that can only replay 10 of the 100 events in σ (i.e., the trace is not fitting at all). Using the naïve fitness metric, the trace would simply be classified as non-fitting for both models without acknowledging that σ was almost fitting in one model and in complete disagreement with the other model. Therefore, we use a fitness notion defined *at the level of events* rather than full traces.

In the naïve fitness computation just described, we stopped replaying a trace once we encounter a problem and mark it as non-fitting. Let us now just continue replaying the trace on the model but record all situations where a transition is forced to fire without being enabled, i.e., we count all missing tokens. Moreover, we record the tokens that remain at the end. To explain the idea, we first replay σ_1 on top of WF-net N_1. Note that σ_1 can be replayed completely. However, we use this example to introduce the notation. Figure 8.3 shows the various stages of replay. Four counters are shown at each stage: p (produced tokens), c (consumed tokens),

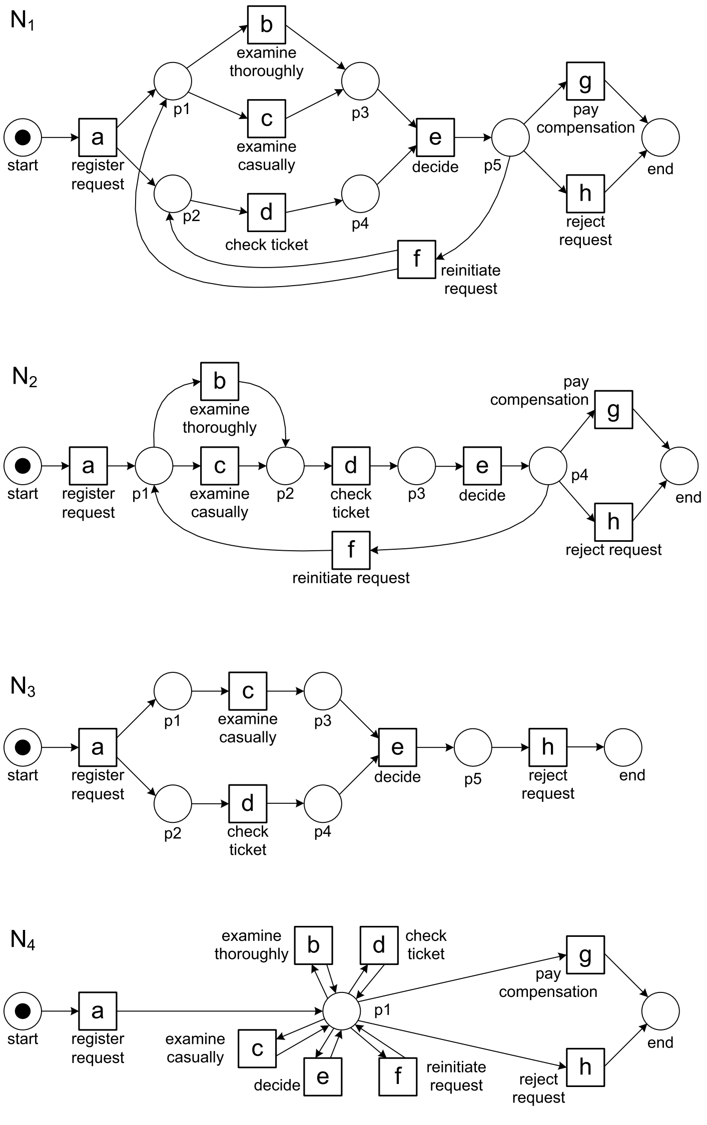


Fig. 8.2 Four WF-nets: N_1, N_2, N_3 and N_4

m (missing tokens), and r (remaining tokens). Let us first focus on p and c. Initially, $p = c = 0$ and all places are empty. Then the environment produces a token for place *start*. Therefore, the p counter is incremented: $p = 1$. Now we need to replay $\sigma_1 = \langle a, c, d, e, h \rangle$, i.e., we first fire transition a. This is possible. Since a consumes one token and produces two tokens, the c counter is incremented by 1 and the p counter is incremented by 2. Therefore, $p = 3$ and $c = 1$ after firing transition a. Then we replay the second event (c). Firing transition c results in $p = 4$

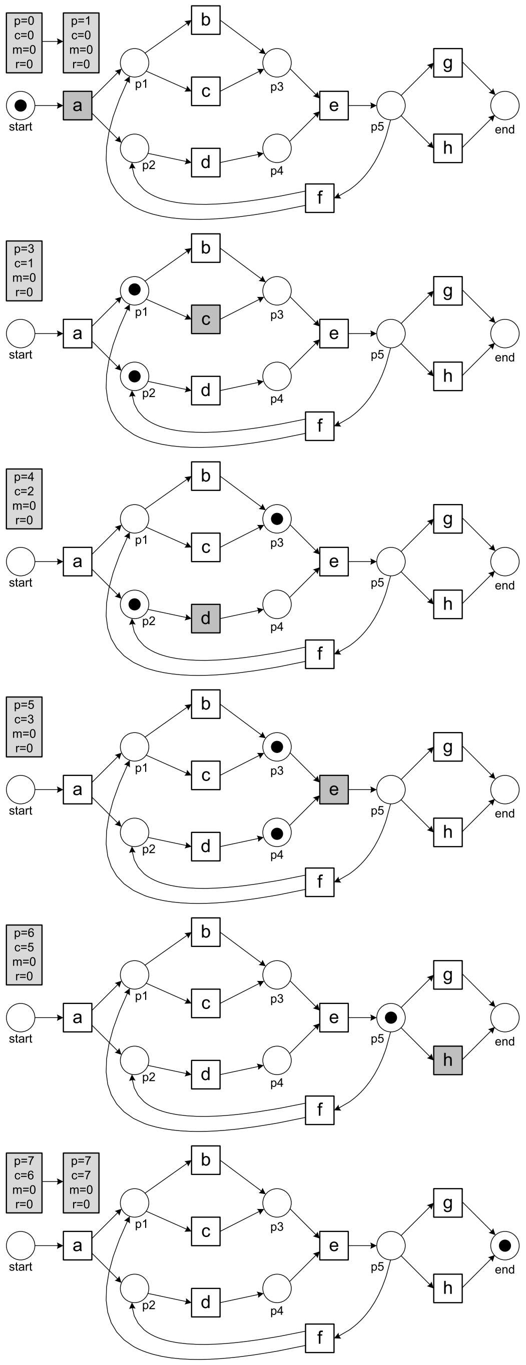


Fig. 8.3 Replaying $\sigma_1 = \langle a, c, d, e, h \rangle$ on top of WF-net N_1. There are four counters: p (produced tokens), c (consumed tokens), m (missing tokens), and r (remaining tokens)

and $c = 2$. After replaying the third event (i.e. d) $p = 5$ and $c = 3$. They we replay e. Since e consumes two tokens and produces one, the result is $p = 6$ and $c = 5$. Then we replay the last event (h). Firing h results in $p = 7$ and $c = 6$. At the end, the environment consumes a token from place *end*. Hence the final result is $p = c = 7$ and $m = r = 0$. Clearly, there are no problems when replaying the σ_1, i.e., there are no missing or remaining tokens ($m = r = 0$).

The fitness of a case with trace σ on WF-net N is defined as follows:

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

The first parts computes the fraction of missing tokens relative to the number of consumed tokens. $1 - \frac{m}{c} = 1$ if there are no missing tokens ($m = 0$) and $1 - \frac{m}{c} = 0$ if all tokens to be consumed were missing ($m = c$). Similarly, $1 - \frac{r}{p} = 1$ if there are no remaining tokens and $1 - \frac{r}{p} = 0$ if none of the produced tokens was actually consumed. We use an equal penalty for missing and remaining tokens. By definition: $0 \leq \mathit{fitness}(\sigma, N) \leq 1$. In our example, $\mathit{fitness}(\sigma_1, N_1) = \frac{1}{2}(1 - \frac{0}{7}) + \frac{1}{2}(1 - \frac{0}{7}) = 1$ because there are no missing or remaining tokens.

Let us now consider a trace that cannot be replayed properly. Fig. 8.4 shows the process of replaying $\sigma_3 = \langle a, d, c, e, h\rangle$ on WF-net N_2. Initially, $p = c = 0$ and all places are empty. Then the environment produces a token for place *start* and the p counter is updated: $p = 1$. The first event (a) can be replayed. After firing a, we have $p = 2$, $c = 1$, $m = 0$, and $r = 0$. Now we try to replay the second event. This is not possible, because transition d is not enabled. To fire d, we need to add a token to place $p2$ and record the missing token, i.e., the m counter is incremented. The p and c counter are updated as usual. Therefore, after firing d, we have $p = 3$, $c = 2$, $m = 1$, and $r = 0$. We also tag place $p2$ to remember that a token was missing. Then we replay the next three events (c, e, h). The corresponding transitions are enabled. Therefore, we only need to update p and c counters. After replaying the last event, we have $p = 6$, $c = 5$, $m = 1$, and $r = 0$. In the final state $[p2, \mathit{end}]$ the environment consumes the token from place *end*. A token remains in place $p2$. Therefore, place $p2$ is tagged and the r counter is incremented. Hence the final result is $p = c = 6$ and $m = r = 1$. Figure 8.4 shows diagnostic information that helps to understand the nature of non-conformance. There was a situation in which d occurred but could not happen according to the model (m-tag) and there was a situation in which d was supposed to happen but did not occur according to the log (r-tag). Moreover, we can compute the fitness of trace σ_3 on WF-net N_2 based on the values of p, c, m, and r:

$$\mathit{fitness}(\sigma_3, N_2) = \frac{1}{2}\left(1 - \frac{1}{6}\right) + \frac{1}{2}\left(1 - \frac{1}{6}\right) = 0.8333$$

As a third example, we replay $\sigma_2 = \langle a, b, d, e, g\rangle$ on top of WF-net N_3. Now the situation is slightly different because N_3 does not contain all activities appearing in the event log. In such a situation it seems reasonable to abstract from these events. Hence, we effectively replay $\sigma_2' = \langle a, d, e\rangle$. Figure 8.5 shows the process of replaying these three events. The first problem surfaces when replaying e. Since c did not

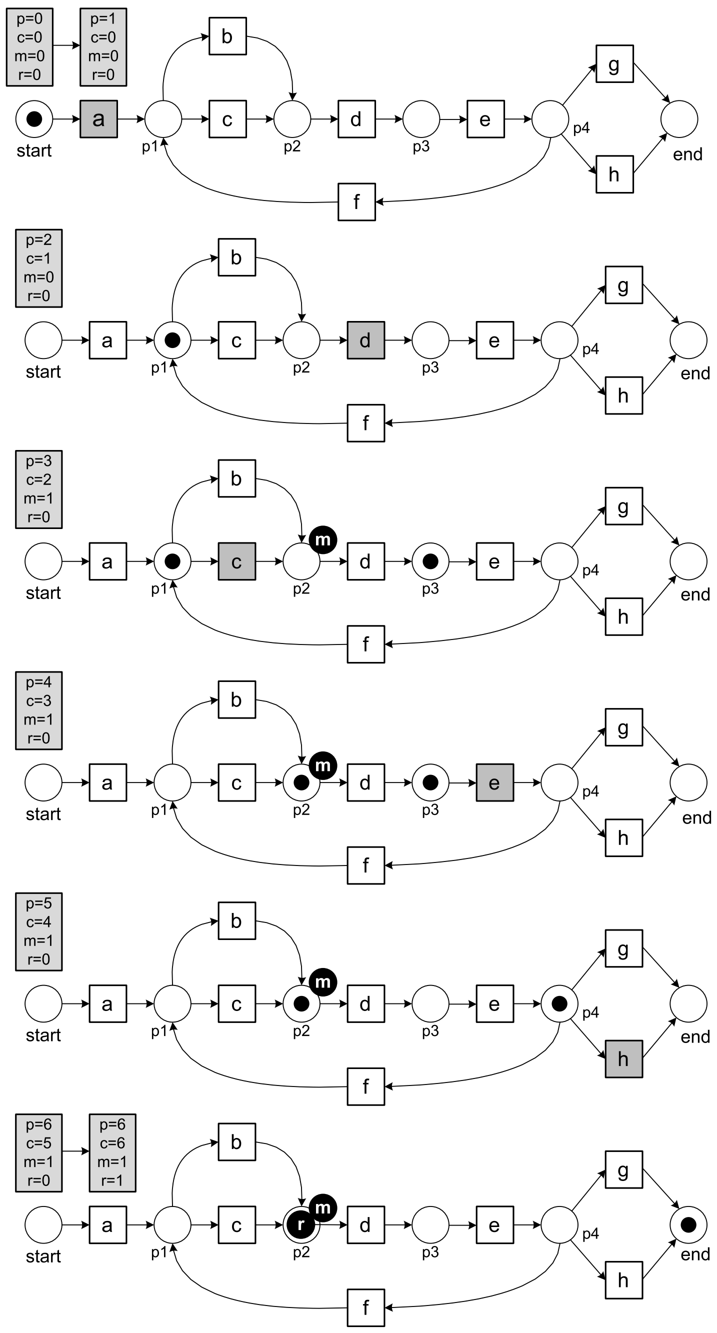


Fig. 8.4 Replaying $\sigma_3 = \langle a, d, c, e, h \rangle$ on top of WF-net N_2: one token is missing ($m = 1$) and one token is remaining ($r = 1$). The r-tag and m-tag highlight the place where σ_3 and the model diverge

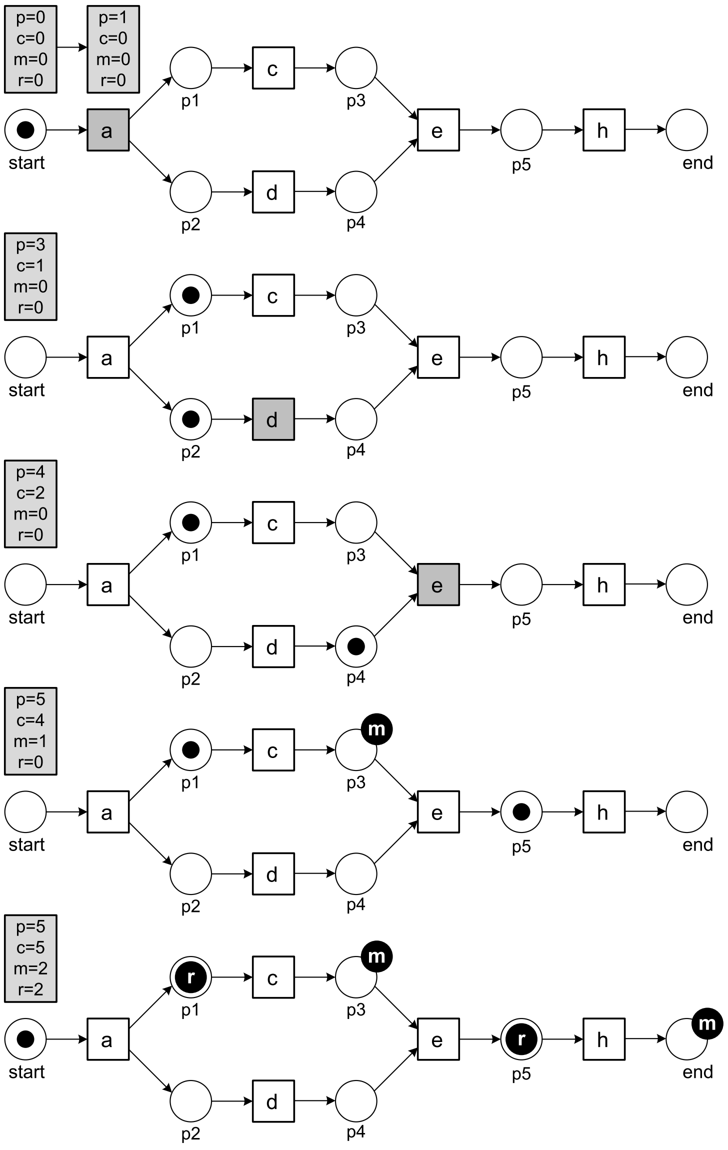


Fig. 8.5 To replay $\sigma_2 = \langle a, b, d, e, g \rangle$ on top of WF-net N_3, all events not corresponding to activities in the model are removed first. Replaying $\sigma'_2 = \langle a, d, e \rangle$ shows that two tokens are missing ($m = 2$) and two tokens are remaining ($r = 2$) thus resulting in a fitness of 0.6

fire, place $p3$ is still empty and e is not enabled. The missing token is recorded ($m = 1$) and place $p3$ gets an m-tag. After replaying σ'_2, the resulting marking is $[p1, p5]$. Now the environment needs to consume the token from place *end*. However, place *end* is not marked. Therefore, another missing token is recorded ($m = 2$)

and also place *end* gets an m-tag. Moreover, two tokens are remaining: one in place $p1$ and one in place $p5$. The places are tagged with an r-tag, and the two remaining tokens are recorded $r = 2$. This way we find a fitness of 0.6 for trace σ_2 and WF-net N_3 based on the values $p = 5$, $c = 5$, $m = 2$, and $r = 2$:

$$\mathit{fitness}(\sigma_2, N_3) = \frac{1}{2}\left(1 - \frac{2}{5}\right) + \frac{1}{2}\left(1 - \frac{2}{5}\right) = 0.6$$

Moreover, Fig. 8.5 clearly shows the cause of this poor conformance: c was supposed to happen according to the model but did not happen, e happened but was not possible according to the model, and h was supposed to happen but did not happen.

Figures 8.3, 8.4, 8.5 illustrate how to analyze the fitness of a single case. The same approach can be used to analyze the fitness of a log consisting of many cases. Simply take the sums of all produced, consumed, missing, and remaining tokens, and apply the same formula. Let $p_{N,\sigma}$ denote the number of produced tokens when replaying σ on N. $c_{N,\sigma}$, $m_{N,\sigma}$, $r_{N,\sigma}$ are defined in a similar fashion, e.g., $m_{N,\sigma}$ is the number of missing tokens when replaying σ on N. Now we can define the fitness of an event log L on WF-net N:

$$\mathit{fitness}(L, N) = \frac{1}{2}\left(1 - \frac{\sum_{\sigma\in L} L(\sigma) \times m_{N,\sigma}}{\sum_{\sigma\in L} L(\sigma) \times c_{N,\sigma}}\right) + \frac{1}{2}\left(1 - \frac{\sum_{\sigma\in L} L(\sigma) \times r_{N,\sigma}}{\sum_{\sigma\in L} L(\sigma) \times p_{N,\sigma}}\right)$$

Note that $\sum_{\sigma\in L} L(\sigma) \times m_{N,\sigma}$ is total number of missing tokens when replaying the entire event log, because $L(\sigma)$ is the frequency of trace σ and $m_{N,\sigma}$ is the number of missing tokens for a single instance of σ. The value of $\mathit{fitness}(L, N)$ is between 0 (very poor fitness; none of the produced tokens is consumed and all of the consumed tokens are missing) and 1 (perfect fitness; all cases can be replayed without any problems). Although $\mathit{fitness}(L, N)$ is a measure focusing on tokens in places, we will interpret it as a measure on events. The intuition of $\mathit{fitness}(L, N) = 0.9$ is that about 90% of the *events* can be replayed correctly.[1] This is only an informal characterization as fitness depends on missing and remaining tokens rather than events. For instance, a transition that is forced to fire during replay may have multiple empty input places. Note that if two subsequent events are swapped in a sequential process, this results in one missing and one remaining token. This seems reasonable, but also shows that the relation between the proportion of events that cannot be replayed correctly and the proportion of tokens that are missing or remaining is rather indirect.

By replaying the entire event log, we can now compute the fitness of event log L_{full} for the four models in Fig. 8.2:

$$\mathit{fitness}(L_{\mathit{full}}, N_1) = 1$$

[1] In the remainder of this book, we often use this intuitive characterization of fitness, although from a technical point of view this is incorrect as $\mathit{fitness}(L, N)$ is only an indication of the fraction of events that can be replayed correctly.

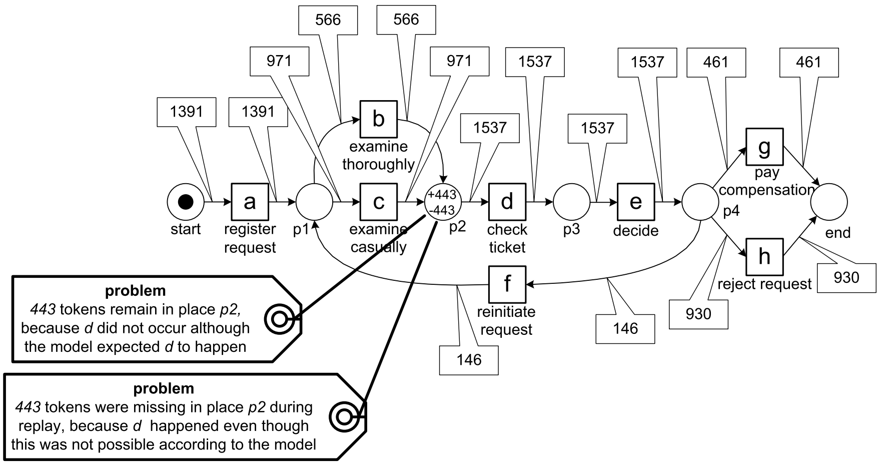

566
566
971
971
1537
1537
461
461
1391
1391
1537
1537
b
examine thoroughly
g
pay compensation
a
register request
c
examine casually
+443
-443
d
check ticket
e
decide
start
p1
p2
p3
p4
end
h
reject request
f
reinitiate request
930
930
146
146
problem
443 tokens remain in place p2, because d did not occur although the model expected d to happen
problem
443 tokens were missing in place p2 during replay, because d happened even though this was not possible according to the model

Fig. 8.6 Diagnostic information showing the deviations ($\mathit{fitness}(L_{\mathit{full}}, N_2) = 0.9504$)

$$\mathit{fitness}(L_{\mathit{full}}, N_2) = 0.9504$$
$$\mathit{fitness}(L_{\mathit{full}}, N_3) = 0.8797$$
$$\mathit{fitness}(L_{\mathit{full}}, N_4) = 1$$

This shows that, as expected, N_1 and N_4 can replay event log L_{full} without any problems (i.e., fitness 1). $\mathit{fitness}(L_{\mathit{full}}, N_2) = 0.9504$. Intuitively, this means that about 95% of the events in L_{full} can be replayed correctly on N_2. As indicated earlier, this can be viewed in two ways:

- Event log L_{full} has a fitness of 0.9504, i.e., about 5% of the events deviate; and
- Process model N_2 has a fitness of 0.9504, i.e., the model is unable to explain 5% of the observed behavior.

The first view is used when the model is considered to be normative and correct ("the event log, i.e. reality, does not conform to the model"). The second view is used when the model should be descriptive ("the process model does not conform to reality"). $\mathit{fitness}(L_{\mathit{full}}, N_3) = 0.8797$, i.e., about 88% of the events in L_{full} can be replayed on N_3. Hence, process model N_3 has the lowest fitness of the four models.

Typically, the event-based fitness is higher than the naïve case-based fitness. This is also the case here. WF-net N_2 can only replay 68% of the cases from start to end. However, about 95% of the individual events can be replayed.

Figure 8.6 shows some the diagnostics than can be generated based on replaying event log L_{full} on process model N_2. The numbers on arcs indicate the flow of produced and consumed tokens. These show how cases flowed through the model, e.g., 146 times a request was reinitiated, 930 requests were rejected and 461 requests resulted in a payment. The places tagged during replay (i.e., the m and r-tags in Figs. 8.3, 8.4, and 8.5) can be aggregated to diagnose conformance problems and reveal their severity. As Fig. 8.6 shows, 443 times activity d happened although

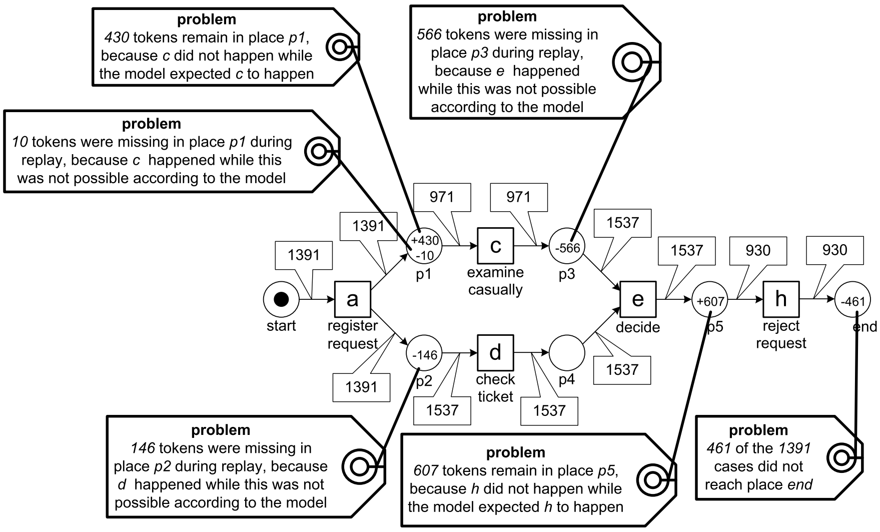

problem
430 tokens remain in place p1, because c did not happen while the model expected c to happen
problem
566 tokens were missing in place p3 during replay, because e happened while this was not possible according to the model
problem
10 tokens were missing in place p1 during replay, because c happened while this was not possible according to the model
971
971
1391
1537
1391
+430
-10
c
-566
1537
930
930
p1
examine casually
p3
start
a
e
+607
h
-461
register request
decide
p5
reject request
end
-146
d
p4
1537
1391
p2
check ticket
1537
1537
problem
146 tokens were missing in place p2 during replay, because d happened while this was not possible according to the model
problem
607 tokens remain in place p5, because h did not happen while the model expected h to happen
problem
461 of the 1391 cases did not reach place end

Fig. 8.7 Diagnostic information showing the deviations ($fitness(L_{full}, N_3) = 0.8797$)

it was not supposed to happen and 443 times activity *d* was supposed to happen but did not. The reason is that *d* was executed before *b* or *c*, which is not possible according to this sequential model.

Similarly, diagnostic information is shown for N_3 in Fig. 8.7. There the problems are more severe. For example, 566 times a decision was made (activity *e*) without being examined casually (activity *c*), and 461 cases did not reach the end because the request was not rejected.

As Fig. 8.8 shows, an event log can be split into two sublogs: *one event log containing only fitting cases and one event log containing only non-fitting cases*. Each of the event logs can be used for further analysis. For example, one could construct a process model for the event log containing only deviating cases. Also other data and process mining techniques can be used. For instance, it is interesting to know which people handled the deviating cases and whether these cases took longer or were more costly. In case fraud is suspected, one may create a social network based on the event log with deviating cases (see Sect. 9.3).

One could also use classification techniques to further investigate non-conformance. Recall that a decision tree can be learned from a table with one response variable and multiple predictor variables. Whether a case fits or not can be seen as the value of a response variable whereas characteristics of the case (e.g., case and event attributes) serve a predictor variables. The resulting decision tree attempts to explain conformance in terms of characteristics of the case. For example, one could find out that cases from gold customers handled by Pete tend to deviate. We will elaborate on this in Sect. 9.5.

See [14, 119, 121] for more information on token-based replay.

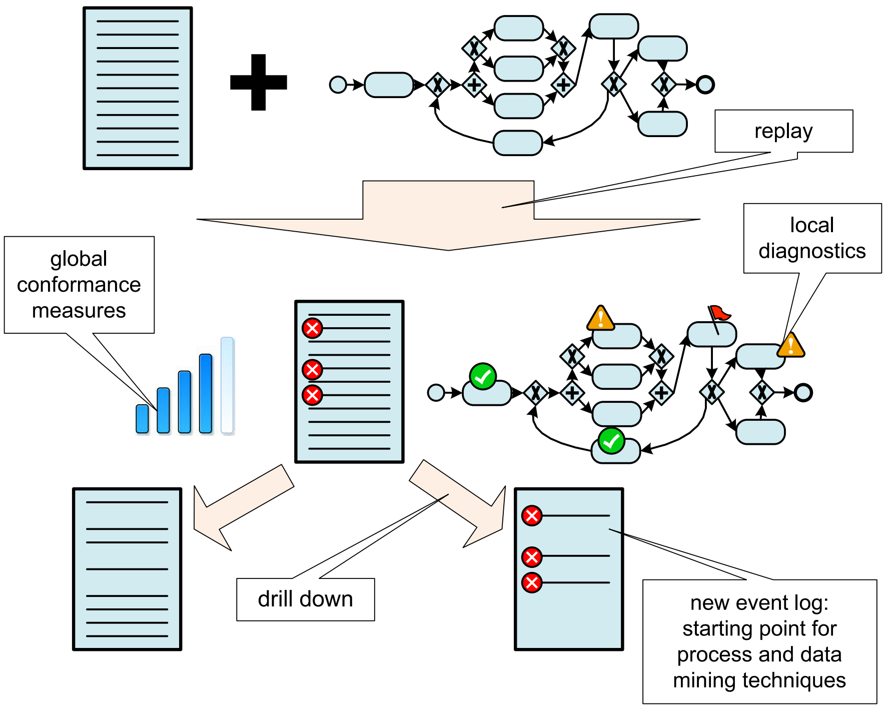

replay
global conformance measures
local diagnostics
drill down
new event log: starting point for process and data mining techniques

Fig. 8.8 Conformance checking provides global conformance measures like *fitness*(L, N) and local diagnostics (e.g., showing activities that were executed although not allowed according to the process model). Moreover, the event log is partitioned into fitting and non-fitting cases. Both sublogs can be used for further analysis, e.g., discovering a process model for the deviating cases

8.3 Alignments

Using token-based replay we can differentiate between fitting and non-fitting cases (see Fig. 8.8). Moreover, the approach is easy to understand and can be implemented efficiently. However, the approach also has some drawbacks. Intuitively, fitness values tend to be too high for extremely problematic event logs. If there are many deviations, the Petri net gets "flooded with tokens" and subsequently allows for any behavior. The approach is also Petri-net specific and can only be applied to other representations after conversion. Moreover, if a case does not fit, the approach does not create a corresponding path through the model. We would like to map observed behavior onto modeled behavior to provide better diagnostics and to relate also non-fitting cases to the model. For example, to compute the mean waiting time between two activities, we cannot leave out all activities that do not fit perfectly. If we would do so, the results could be biased. *Alignments* were introduced to overcome these limitations [169].

To explain the notion of alignments informally, consider trace $\sigma = \langle a, d, b, e, h\rangle$ and the four models in Fig. 8.2. It is easy to see that σ fits perfectly in N_1 and N_4, but not in N_2 and N_3. A so-called *optimal alignment* is a best match given a trace and a model. Given σ and N_1 there is precisely one optimal alignment,

$$\gamma_1 = \begin{array}{|c|c|c|c|c|} a & d & b & e & h \\ \hline a & d & b & e & h \end{array}$$

The top row corresponds to σ and the bottom row corresponds to a path from the initial marking to the final marking of N_1.

Given σ and N_2 there are multiple optimal alignments:

$$\gamma_{2a} = \begin{array}{|c|c|c|c|c|c|} a & \gg & d & b & e & h \\ \hline a & b & d & \gg & e & h \end{array} \qquad \gamma_{2b} = \begin{array}{|c|c|c|c|c|c|} a & \gg & d & b & e & h \\ \hline a & c & d & \gg & e & h \end{array} \qquad \gamma_{2c} = \begin{array}{|c|c|c|c|c|c|} a & d & b & \gg & e & h \\ \hline a & \gg & b & d & e & h \end{array}$$

The "$\gg$" symbols denote misalignments. In γ_{2a}, the model makes a "b move" before d may occur in both log (top row) and model (bottom row). Subsequently, the log makes a "b move", not possible anymore in the model. In γ_{2b}, the model makes a "c move" (rather than a "b move") before d. In γ_{2c}, the log first makes a "d move" not possible in the model, followed by b and a "d move" made by the model. All three alignments have two $\gg$'s ("no moves").

Given σ and N_3, there are also three optimal alignments:

$$\gamma_{3a} = \begin{array}{|c|c|c|c|c|c|} a & \gg & d & b & e & h \\ \hline a & c & d & \gg & e & h \end{array} \qquad \gamma_{3b} = \begin{array}{|c|c|c|c|c|c|} a & d & \gg & b & e & h \\ \hline a & d & c & \gg & e & h \end{array} \qquad \gamma_{3c} = \begin{array}{|c|c|c|c|c|c|} a & d & b & \gg & e & h \\ \hline a & d & \gg & c & e & h \end{array}$$

The model needs to make a "c move" and the log needs to make "b move" not possible in the model.

Given σ and N_4, there is just one optimal alignment,

$$\gamma_4 = \begin{array}{|c|c|c|c|c|} a & d & b & e & h \\ \hline a & d & b & e & h \end{array}$$

The alignment shows that σ perfectly fits N_4: there are no $\gg$'s signaling discrepancies between modeled and observed behavior.

The examples illustrate the usefulness of alignments. *Detailed diagnostics* can be given per case and these can be aggregated into *diagnostics at the process model level*. For example, we can indicate that a specific activity is often skipped or that some other activity occurs at times it is not supposed to happen. Moreover, observed behavior is related to modeled behavior in a precise manner.

Token-based conformance checking becomes more complicated when there are duplicate and silent activities, e.g., transitions with a τ label or two transitions with the same label. Alignments can be defined *for any process notation*, including Petri nets having duplicate and silent activities. To illustrate this, consider Fig. 8.9. The labeled Petri net N_5 is composed of 8 transitions and 7 places. Transition $t1$ has label a modeling the initial registration step, transition $t2$ has label b modeling an examination step, etc. There are two decision transitions ($t4$ and $t5$) having the same

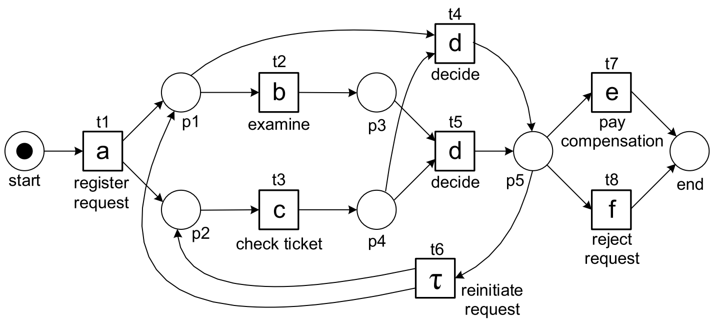


Fig. 8.9 A WF-net N_5 with duplicate and silent activities

label (d). There is one *silent* transition ($t6$). This transition models the reinitiation step. This step is invisible as is reflected by the τ label. Given $\sigma_1 = \langle a, c, d, e\rangle$ and N_5, there is precisely one optimal alignment,

$$\gamma_{5,1} = \begin{array}{|c|c|c|c|} a & c & d & e \\ \hline a & c & d & e \\ t1 & t3 & t4 & t7 \end{array}$$

The top row of the alignment corresponds to "moves in the log" and the bottom two rows correspond to "moves in the model". Moves in the model are now represented by both the transition and its label. This is needed because there could be multiple transitions with the same label, e.g., in N_5 both $t4$ and $t5$ have a d label. The d event in trace $\sigma_1 = \langle a, c, d, e\rangle$ represents a decision and is not connected to a specific transition. However, during replay is becomes clear that d must refer to $t4$.

If a move in the model cannot be mimicked by a move in the log, then a "$\gg$" ("no move") appears in the top row. Consider $\sigma_2 = \langle a, b, d, f\rangle$. There are two optimal alignments for σ_2 and N_5:

$$\gamma_{5,2a} = \begin{array}{|c|c|c|c|c|} a & b & \gg & d & f \\ \hline a & b & c & d & f \\ t1 & t2 & t3 & t5 & t8 \end{array} \qquad \gamma_{5,2b} = \begin{array}{|c|c|c|c|c|} a & \gg & b & d & f \\ \hline a & c & b & d & f \\ t1 & t3 & t2 & t5 & t8 \end{array}$$

If a move in the log cannot be mimicked by a move in the model, then a "$\gg$" ("no move") appears in the bottom row. Consider $\sigma_3 = \langle a, c, d, e, f\rangle$. There are two optimal alignments for σ_3 and N_5:

$$\gamma_{5,3a} = \begin{array}{|c|c|c|c|c|} a & c & d & e & f \\ \hline a & c & d & e & \gg \\ t1 & t3 & t4 & t7 & \end{array} \qquad \gamma_{5,3b} = \begin{array}{|c|c|c|c|c|} a & c & d & e & f \\ \hline a & c & d & \gg & f \\ t1 & t3 & t4 & & t8 \end{array}$$

Silent transition $t6$ leaves no trail in the event log. Given $\sigma_4 = \langle a, c, d, b, c, d, c, d, c, b, d, f \rangle$ and N_5, there is precisely one optimal alignment,

$$\gamma_{5,4} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}
a & c & d & \gg & b & c & d & \gg & c & d & \gg & c & b & d & f \\
\hline
a & c & d & \tau & b & c & d & \tau & c & d & \tau & c & b & d & f \\
t1 & t3 & t4 & t6 & t2 & t3 & t5 & t6 & t3 & t4 & t6 & t3 & t2 & t5 & t8
\end{array}$$

The alignment loops back three times. All $\gg$'s correspond to model moves of silent transition $t6$. These are considered harmless because these moves are invisible and cannot be observed in the log anyway. Hence, we consider $\sigma_4 = \langle a, c, d, b, c, d, c, d, c, b, d, f \rangle$ and N_5 to be perfectly fitting.

A *move* is a pair $(x, (y, t))$ where the first element refers to the log and the second element refers to the model. For example, $(a, (a, t1))$ means that both log and model make an "a move" and the move in the model is caused by the occurrence of transition $t1$. $(\gg, (c, t3))$ means that the occurrence of transition $t3$ with label c is not mimicked by a corresponding move of the log. $(f, \gg)$ means that the log makes an "f move" not followed by the model.

$(x, (y, t))$ is a *legal move* if one of the following four cases holds:

- $x = y$ and y is the visible label of transition t (*synchronous move*),
- $x = \gg$ and y is the visible label of transition t (*visible model move*),
- $x = \gg$, $y = \tau$ and transition t is silent (*invisible model move*), or
- $x \neq \gg$ and $(y, t) = \gg$ (*log move*).

Other moves such as $(\gg, \gg)$ and $(x, (y, t))$ with $x \neq y$ are illegal moves.

An *alignment* is a sequence of legal moves such that after removing all $\gg$ symbols, the top row corresponds to the trace in the log, and the bottom row corresponds to a firing sequence starting in some initial state of the process model and ending in some final state. Consider, for example, $\gamma_{5,2a}$. This is an alignment for σ_2 and N_5 because the top row $\langle a, b, \gg, d, f \rangle$ is indeed σ_2 after removing the $\gg$ and the bottom row $\langle t1, t2, t3, t5, t8 \rangle$ is indeed a firing sequence leading from [*start*] to [*end*].

Given a log trace and a process model, there may be many (if not infinitely many) alignments. For $\sigma_2 = \langle a, b, d, f \rangle$ and N_5, there are additional alignments like:

$$\gamma_{5,2c} = \begin{array}{|c|c|c|c|c|c|c|c|c|}
a & b & d & f & \gg & \gg & \gg & \gg & \gg \\
\hline
\gg & \gg & \gg & \gg & a & b & c & d & f \\
 & & & & t1 & t2 & t3 & t5 & t8
\end{array}
\qquad
\gamma_{5,2d} = \begin{array}{|c|c|c|c|c|c|c|}
a & b & d & f & \gg & \gg & \gg \\
\hline
a & \gg & \gg & \gg & c & d & e \\
t1 & & & & t3 & t4 & t7
\end{array}$$

Alignments $\gamma_{5,2a}$ and $\gamma_{5,2b}$ have just one $\gg$, alignment $\gamma_{5,2c}$ has nine $\gg$'s, and $\gamma_{5,2d}$ has six $\gg$'s. Clearly, $\gamma_{5,2a}$ (or $\gamma_{5,2b}$) describes the relation between σ_2 and N_5 better than the two longer alignments $\gamma_{5,2c}$ and $\gamma_{5,2d}$.

To select the most appropriate alignment, we associate *costs* to undesirable moves and select an alignment with the lowest total costs. Cost function δ assigns costs to legal moves. Moves where log and model agree have no costs, i.e., $\delta(x, (y, t)) = 0$ for *synchronous moves* (with $x = y$). Moves in model only have no costs if the transition is invisible, i.e., $\delta(\gg, (\tau, t)) = 0$ for *invisible model moves*.

$\delta(\gg, (y,t)) > 0$ is the cost when the model makes an "y move" without a corresponding move of the log (*visible model move*). $\delta(x, \gg) > 0$ is the cost for an "x move" in just the log (*log move*). These costs may depend on the nature of the activity, e.g., skipping a payment may be more severe than sending too many letters.

For simplicity we assume a fixed *standard cost function* which assigns cost 1 to all visible model moves and log moves ($\delta(\gg, (y,t)) = \delta(x, \gg) = 1$ with $y \neq \tau$). The cost of an alignment is simply the sum of the costs of all its moves. For example, $\delta(\gamma_{5,2a}) = \delta(\gamma_{5,2b}) = 1$, $\delta(\gamma_{5,2c}) = 9$, and $\delta(\gamma_{5,2d}) = 6$. $\delta(\gamma_{5,1}) = 0$ indicating that there are no misalignments. Also $\delta(\gamma_{5,4}) = 0$ because $\delta(\gg, (\tau, t6)) = 0$.

An alignment is *optimal* if there is no alternative alignment with lower costs. Obviously, $\gamma_{5,1}$ and $\gamma_{5,4}$ are optimal because the costs are 0 and cannot be lower. $\gamma_{5,2a}$ and $\gamma_{5,2b}$ are optimal alignments for trace σ_2 and model N_5. Both $\gamma_{5,3a}$ and $\gamma_{5,3b}$ are optimal alignments for σ_3 and N_5. These examples show that optimal alignments do not need to be unique. However, without loss of generality, we can assume a *deterministic* mapping that assigns any log trace σ to an optimal alignment $\lambda_{opt}^{N}(\sigma)$ in the context of a particular process model N. Such a mapping is sometimes referred to as an "Oracle": for any observed behavior a suitably chosen path through the model is returned.

It is possible to convert misalignment costs into a fitness value between 0 (poor fitness, i.e., maximal costs) and 1 (perfect fitness, zero costs). The worst-case scenario is that there are no synchronous moves and only "moves in model only" and "moves in log only". Note that we can always create an alignment where all events in trace σ are converted to log moves and a shortest path from an initial state to a final state of the model is added as a sequence of model moves. An example of a "worst-case alignment" for $\sigma_2 = \langle a, b, d, f\rangle$ and N_5 is

$$\gamma_{5,2w} = \begin{array}{|c|c|c|c|c|c|c|c|} a & b & d & f & \gg & \gg & \gg & \gg \\ \hline \gg & \gg & \gg & \gg & a & c & d & f \\ & & & & t1 & t3 & t4 & t8 \end{array}$$

This alignment has "moves in log only" for the observed events in $\sigma_2 = \langle a, b, d, f\rangle$ and "moves in model only" for firing sequence $\langle t1, t3, t4, t8\rangle$.

A worst-case alignment always yields a valid alignment and there cannot be optimal alignments with higher costs. Let us call this alignment $\lambda_{worst}^{N}(\sigma)$. Now the fitness of a trace σ can be defined as follows:

$$\mathit{fitness}(\sigma, N) = 1 - \frac{\delta(\lambda_{opt}^{N}(\sigma))}{\delta(\lambda_{worst}^{N}(\sigma))}$$

Assuming there is a path from some initial state to some final state, this always yields a value between 0 and 1. For σ_2 and model N_5, $\delta(\lambda_{opt}^{N_5}(\sigma_2)) = 1$, $\delta(\lambda_{worst}^{N_5}(\sigma_2)) = 8$, and $\mathit{fitness}(\sigma_2, N_5) = 1 - \frac{1}{8} = 0.875$.

The fitness notion can be extended to event logs in a straightforward manner:

$$\mathit{fitness}(L, N) = 1 - \frac{\sum_{\sigma \in L} L(\sigma) \times \delta(\lambda_{opt}^{N}(\sigma))}{\sum_{\sigma \in L} L(\sigma) \times \delta(\lambda_{worst}^{N}(\sigma))}$$

Note that $\sum_{\sigma \in L} L(\sigma) \times \delta(\lambda^{N}_{opt}(\sigma))$ is the sum of all costs when replaying the entire event log using optimal alignments. This is divided by the worst-case scenario to obtain a normalized overall fitness value.

To show alignment-based conformance checking in action, we revisit the event log L_{full} described in Table 8.1 and the four models in Fig. 8.2.

Let us first consider model N_1 in Fig. 8.2. For any trace in L_{full}, the optimal alignment has costs 0. There are 7539 synchronous moves for the 1391 cases. There are no separate log or model moves. Hence, $\mathit{fitness}(L_{full}, N_1) = 1$.

Next we consider model N_2 in Fig. 8.2. This model does not allow for concurrency and cannot handle traces where d occurs before b or c. There are 457 situations in event log L_{full} where d occurs before b or c. For some traces multiple optimal alignments are possible. This enables us to use a deterministic "Oracle" returning a particular optimal alignment. The choice of the Oracle does not influence the fitness computation. By definition these all yield the same fitness value. Consider, for example, $\sigma_8 = \langle a, c, d, e, f, d, b, e, h\rangle$ that occurred 47 times in L_{full}. Two examples of optimal alignments for this trace are (transition names are omitted):

$$\gamma_{8a} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} a & c & d & e & f & \gg & d & b & e & h \\ \hline a & c & d & e & f & b & d & \gg & e & h \end{array} \qquad \gamma_{8b} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|} a & c & d & e & f & d & b & \gg & e & h \\ \hline a & c & d & e & f & \gg & b & d & e & h \end{array}$$

For a particular collection of optimal alignments for L_{full} there are 7082 synchronous moves, 457 model moves, and 457 log moves. Hence, $\sum_{\sigma \in L_{full}} L_{full}(\sigma) \times \delta(\lambda^{N_2}_{opt}(\sigma)) = 457 + 457 = 914$. There are 7539 events in L_{full} and the shortest path from the initial marking to the final marking takes 5 model moves. Hence, $\sum_{\sigma \in L_{full}} L_{full}(\sigma) \times \delta(\lambda^{N_2}_{worst}(\sigma)) = 7539 + 1391 \times 5 = 14494$. This is the worst-case scenario. Therefore, $\mathit{fitness}(L_{full}, N_2) = 1 - \frac{914}{14494} = 0.936939$. Note that the fitness value is slightly lower than the fitness value using token based replay. This is caused by the cases where d occurs multiple times before b or c (within the same case). These are not sufficiently penalized using token based replay. A second or third misalignment in the same case is not detected due to a token remaining from the first misalignment.

Figure 8.10 shows the diagnostics based on a particular collection of optimal alignments. Compare the token replay diagnostics in Fig. 8.6 to the alignment-based diagnostics in this figure. Figure 8.6 suggests that d was executed 443 times before b or c. This is not the case as Fig. 8.10 clearly shows. b was executed 170 times after d. c was executed 287 times after d. d was executed $170 + 287 = 457$ times before b or c. The difference between 443 in Fig. 8.6 and the correct 457 in Fig. 8.10 is caused by tokens remaining in place $p2$ after the first iteration.

Next we consider model N_3 in Fig. 8.2. Several activities appearing in the event log do not appear in the model, thus causing unavoidable "moves in log only". Taking a particular collection of optimal alignments, we note that there are 6064 synchronous moves, 891 model moves, and 1475 log moves. Hence, $\sum_{\sigma \in L_{full}} L_{full}(\sigma) \times \delta(\lambda^{N_3}_{opt}(\sigma)) = 891 + 1475 = 2366$. The worst-case scenario still has costs 14494 since there are 7539 events in L_{full} and the shortest path from the

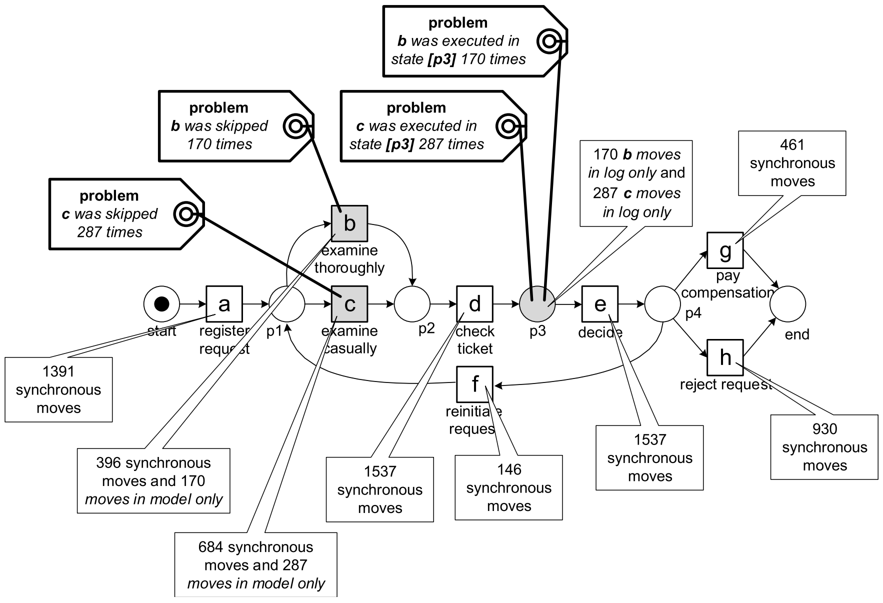


Fig. 8.10 Diagnostic information showing the deviations ($fitness(L_{full}, N_2) = 0.936939$)

initial marking to the final marking is still 5 steps. Therefore, $fitness(L_{full}, N_3) = 1 - \frac{2366}{14494} = 0.83676$.

Figure 8.11 shows the diagnostics based on this particular collection of optimal alignments. Activity c in the model was skipped 430 times in the event log and activity h was skipped 461 times. This explains the 891 "moves in model only". The 1475 log moves are scattered over the different states of the model. Figure 8.7 shows that c was executed 971 times. However, as Fig. 8.11 shows, activity c was executed 961 times at a time allowed by the model. The $971 - 961 = 10$ occurrences of c happened in the second or third iteration which are non-existent in N_3. Hence, the 971 in Fig. 8.7 is misleading. Unlike token-based replay, alignments map all traces in the log onto *actually existing* firing sequences from an initial state to a final state.

Finally, we align L_{full} with the "flower model" N_4 in Fig. 8.2. As expected, there are 7539 synchronous moves and no "moves in model only" and no "moves in log only". Hence, $fitness(L_{full}, N_4) = 1$.

Based on the above examples, we conclude that the following differences exist between token-based and alignment-based conformance checking:

- Alignments provide more *detailed* but *easy to understand diagnostics*. Skipped and inserted events are easier to interpret than missing and remaining tokens.
- Alignments provide more *accurate diagnostics*. Token-based replay may provide misleading diagnostics due to remaining tokens (earlier deviations mask later deviations). As a result fitness values are generally too low.

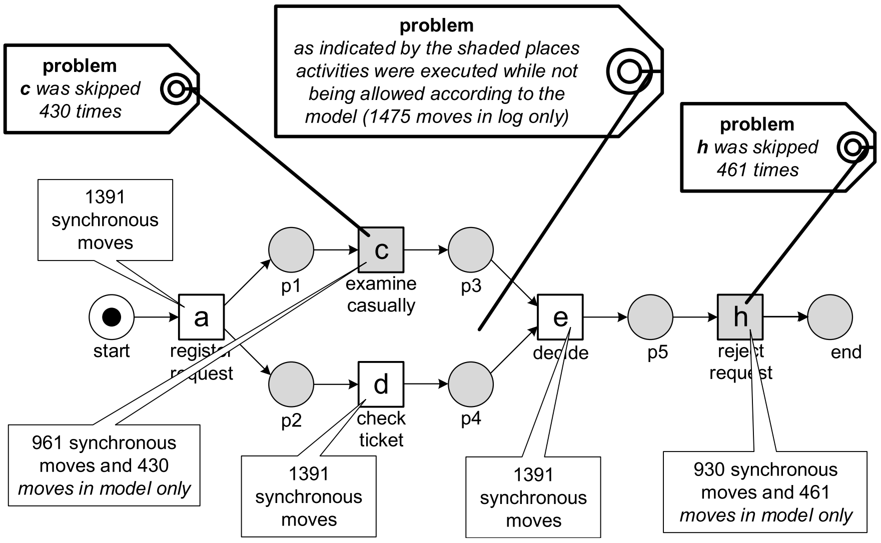

problem
c was skipped 430 times
problem
as indicated by the shaded places activities were executed while not being allowed according to the model (1475 moves in log only)
problem
h was skipped 461 times
1391 synchronous moves
start
a
register request
p1
p2
c
examine casually
d
check ticket
p3
p4
e
decide
p5
h
reject request
end
961 synchronous moves and 430 moves in model only
1391 synchronous moves
1391 synchronous moves
930 synchronous moves and 461 moves in model only

Fig. 8.11 Diagnostic information showing the deviations ($fitness(L_{full}, N_3) = 0.83676$)

- Alignments are *configurable* through the cost function. One can use multiple cost functions depending on the likelihood of a deviation and its severity [2].
- Alignments can be used to *map each case onto a feasible path in model*. This is important for projecting information (e.g., bottlenecks) on models. Moreover, the mapping ensures that non-fitting extra behavior is not causing misleading diagnostics. Token-based replay also relates observed and modeled behavior, but does not create the corresponding end-to-end execution sequences in the model.
- Alignments are *model independent*. Any process model with formal semantics and initial and final states can be used. Token-based replay assumes a Petri net, so conversions may be needed (e.g., from BPMN to Petri nets).
- Token-based replay provides *deterministic diagnostics* whereas multiple optimal alignments may exist for a trace. This can be addressed by deterministically picking one of possibly many optimal alignments. This does not influence the overall fitness value, but influences diagnostics based on alignments. Multiple optimal alignments can be returned for the same case, but this further complicates interpretation.

See [2–5, 169] for more precise alignment definitions and examples. As mentioned, the idea to align event logs and process models is not limited to Petri nets. Any process modeling notation with executable semantics can replay the event log in some way. See also the replay techniques used in [12, 66, 183, 184].

8.4 Comparing Footprints

In Sect. 6.2, we defined the notion of a *footprint*, i.e., a matrix showing causal dependencies. Such a matrix characterizes the event log. For instance, Table 8.2 shows the

Table 8.2 Footprint of L_{full} and N_1

	a	b	c	d	e	f	g	h
a	#	→	→	→	#	#	#	#
b	←	#	#	‖	→	←	#	#
c	←	#	#	‖	→	←	#	#
d	←	‖	‖	#	→	←	#	#
e	#	←	←	←	#	→	→	→
f	#	→	→	→	←	#	#	#
g	#	#	#	#	←	#	#	#
h	#	#	#	#	←	#	#	#

Table 8.3 Footprint of N_2 shown in Fig. 8.2

	a	b	c	d	e	f	g	h
a	#	→	→	#	#	#	#	#
b	←	#	#	→	#	←	#	#
c	←	#	#	→	#	←	#	#
d	#	←	←	#	→	#	#	#
e	#	#	#	←	#	→	→	→
f	#	→	→	#	←	#	#	#
g	#	#	#	#	←	#	#	#
h	#	#	#	#	←	#	#	#

Table 8.4 Differences between the footprints of L_{full} and N_2. The event log and the model "disagree" on 12 of the 64 cells of the footprint matrix

	a	b	c	d	e	f	g	h
a				→: #				
b				‖ :→	→: #			
c				‖ :→	→: #			
d	←: #	‖ :←	‖ :←			←: #		
e		←: #	←: #					
f				→: #				
g								
h								

footprint matrix of L_{full}. This matrix is derived from the "directly follows" relation $>_{L_{full}}$. Clearly, process models also have a footprint: simply generate a complete event log, i.e., Play-Out the model and record execution sequences. From the viewpoint of a footprint matrix, an event log is complete if and only if all activities that can follow one another do so at least once in the log. Applying this to N_1 in Fig. 8.2 results in the same footprint matrix (i.e., Table 8.2). This suggests that the event log and the model "conform".

Table 8.3 shows the footprint matrix generated for WF-net N_2, i.e., Play-Out N_2 to record a complete log and derived its footprint. Comparing both footprint matrices (Tables 8.2 and 8.3) reveals several differences as shown in Table 8.4. For example, the relation between a and d changed from → to #. When comparing event log L_{full}

with WF-net N_2 it can indeed be seen that in L_{full} activity a is directly followed by d whereas this is not possible in N_2. The relation between b and d changed from $\|$ to $\rightarrow$. This reflects that in WF-net N_2 both activities are no longer parallel. Besides providing detailed diagnostics, Table 8.4 can also be used to quantify conformance. For instance, 12 of the 64 cells differ. Hence, one could say that the conformance based on the footprints is $1 - \frac{12}{64} = 0.8125$.

Conformance analysis based on footprints is only meaningful if the log is complete with respect to the "directly follows" relation $>_L$. This can be verified using k-fold cross-validation (see Sect. 4.6.2).

Interestingly, both models and event logs have footprints. This allows for *log-to-model* comparisons as just described, i.e., it can be checked whether and model and log "agree" on the ordering of activities. However, the same approach can be used for *log-to-log* and *model-to-model* comparisons. Comparing the footprints of two process models (model-to-model comparison) allows for the quantification of their similarity. Comparing the footprints of two event logs (log-to-log comparison) can, for example, be used for detecting *concept drift*. The term concept drift refers to the situation in which the process is changing while being analyzed. For instance, in the beginning of the event log two activities may be concurrent whereas later in the log these activities become sequential. This can be discovered by splitting the log into smaller logs and analyzing the footprints of the smaller logs. A log-to-log comparison of a sequence of event logs may reveal concept drift. Such a "second order process mining" requires lots of data because all the smaller logs are assumed to be complete with respect to $>_L$.

A topic typically neglected in literature and tools is the *cross-validation of conformance*. The event log is just a sample of behavior. This sample may be too small to make a reliable statement about conformance. Moreover, there may be additional complications like concept drift. For example, the average conformance over 2011 is 0.80, however, in the beginning of the year it was 0.90, but during the last two months conformance has been below 0.60. Most techniques provide a single conformance metric without stating anything about the reliability of the measure or concept drift. For instance, suppose we have a large event log L_1 and a small event log L_2 such that $L_2 \subset L_1$ and $|L_2| = 0.01 \times |L_1|$, i.e., L_2 contains 1% of the cases in L_1. Suppose that $\mathit{fitness}(L_1, N) = 0.9$ and $\mathit{fitness}(L_2, N) = 0.6$. Clearly, the first value is much more reliable (as it is based on a log 100 times larger) but this is not expressed in the metric. If there is enough data to do cross-validation, the event log could be split randomly into k parts (see also Sect. 4.6.2). Then the fitness could be computed for all k parts. These k independent measures could then be used to create a confidence interval for the conformance of the underlying process, e.g., the fitness is, with 90% confidence, between 0.86 and 0.94. Some of the conformance measures have a tendency to go up or down when the event log is larger or smaller. Whereas token replay and alignments are insensitive to the size of the log, other measures like the ones based on the footprint matrix depend on the size and completeness of the event log. Consider, for example, an event log L split into two smaller logs L_1 and L_2. Assuming that the process is in steady state, the expected value for $\mathit{fitness}(L, N)$ is identical to the expected value for $\mathit{fitness}(L_1, N)$

and *fitness*(L_2, N). This does not hold for measures like the footprint matrix: relation $>_L$ can only grow if the log gets larger. Relative thresholds, as used for heuristic mining, may be used to reduce this effect.

The footprint is just one of many possible characterizations of event logs and models. In principle, any temporal property can be used. Instead of the "directly follows" relation also an "eventually follows" relation $\gg_L$ can be used. $a \gg_L b$ means that there is at least one case for which a was eventually followed by b. This can also be combined with some time window, e.g., a was followed by b within four steps or a was followed by b within four hours. It is also possible to take frequencies into account (see for example measures such as $|a >_L b|$ and $|a \Rightarrow_L b|$ defined in the context of heuristic mining) and use thresholds. Clearly, the characterizations used to compare logs and models should match the notion of conformance one is interested in.

The token-based replay technique described in Sect. 8.2, the alignments in Sect. 8.3, and the comparison of footprint matrices can be used to check the conformance of an event log and a *whole* process model. It is of course also possible to directly check specific *constraints*, sometimes referred to as "business rules". An example of a constraint is that activity a should always be followed by activity b. Another example is the so-called *4-eyes principle*: activities a and b should never be executed by the same person, e.g., to avoid fraud. In [158], it is shown how an LTL-based language can be used in the context of process mining. *Linear Temporal Logic* (LTL) is an example of a temporal logic that, in addition to classical logical operators, uses temporal operators such as: always ($\Box$), eventually ($\Diamond$), until ($\sqcup$), weak until (W), and next time ($\bigcirc$) [30]. For instance, one could formulate the rule $\Box(a \Rightarrow \Diamond(g \vee h))$, i.e., if a occurs, it should eventually be followed by g or h. Another example is the rule $\Diamond g \Leftrightarrow !(\Diamond h)$ stating that eventually either g should happen or h should happen, but not both. The directly follows relation $a >_L b$ used earlier can be expressed in LTL: "$\Diamond(a \wedge \bigcirc(b))$" (for at least one case). This illustrates that behavioral characterizations such as footprints can often be expressed in terms of LTL. LTL-based constraints as defined in [158] may also include explicit time and data. For example, it is possible to state that within two days after the occurrence of activity e, one of the activities g or h should have happened. Another example is that for gold customers the request should be handled in one week and for silver customers in two weeks.

Constraints may be used to split the log into two parts as described in Fig. 8.8. This way it is possible to further investigate cases that violate some business rule.

Declare: A constraint-based workflow language
In this book, we focus on mainstream process modeling languages like Petri nets, BPMN, EPCs, and YAWL. These languages are procedural and aim to describe end-to-end processes. In the context of conformance checking it is interesting to also consider declarative process modeling languages. *Declare* is such a language (in fact a family of languages) and a fully func-

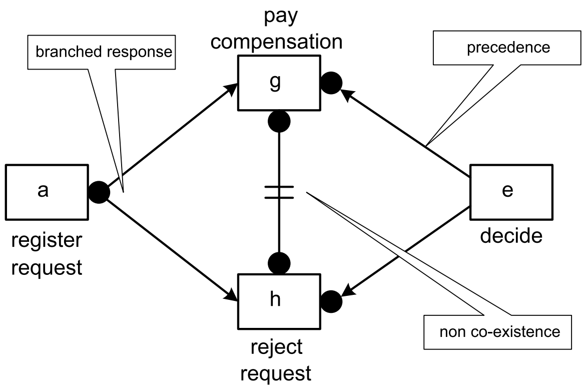


Fig. 8.12 *Declare* specification consisting of four constraints: two precedence constraints, one non-coexistence constraint, and one branched response constraint

tional WFM system [103, 162]. Declare uses a graphical notation and semantics based on LTL. Figure 8.12 shows a Declare specification consisting of four constraints. The construct connecting activities g and h is a so-called *non-coexistence constraint*. In terms of LTL this constraint means "$!((\Diamond g) \wedge (\Diamond h))$"; $\Diamond g$ and $\Diamond h$ cannot both be true, i.e., it cannot be the case that both g and h happen for the same case. There are two *precedence constraints*. The semantics of the precedence constraint connecting e to g can also be expressed in terms of LTL: "$(!g)\ W\ e$", i.e., g should not happen before e has happened. Since the weak until (W) is used in "$(!g)\ W\ e$", traces without any g and e events also satisfy the constraint. Similarly, h should not happen before e has happened: "$(!h)\ W\ e$". The constraint connecting a to g and h is a so-called branched constraint involving three activities. This *response constraint* states that every occurrence of a should eventually be followed by g or h: "$\Box(a \Rightarrow (\Diamond(g \vee h)))$". The latter constraint allows for $\langle a, a, a, g, h, a, a, h\rangle$ but not $\langle a, g, h, a\rangle$. Example traces that satisfy all four constraints are $\langle a, a, e, e, g\rangle$ and $\langle a, e, h, e\rangle$.

Procedural languages only allow for activities that are explicitly triggered through control-flow (token semantics). In a declarative language like Declare "*everything is possible unless explicitly forbidden*".

The Declare language is supported by a WFM system that is much more flexible than traditional procedural WFM/BPM systems [162]. Moreover, it is possible to learn Declare models by analyzing event logs [86, 103]. The graphical constraint language is also suitable for conformance checking. Given an event log, it is possible to check all the constraints. Consider, for instance, Fig. 8.12. Given an event log one can show for each constraint the proportion of cases that respects this constraint [103, 162]. In case of conformance checking, complex time-based constraints may be used (e.g., after every occurrence of activity a for a gold customer, activity g or h should happen within 24 hours).

8.5 Other Applications of Conformance Checking

Conformance checking can be used for improving the alignment of business processes, organizations, and information systems. As shown, replay techniques and footprint analysis help to identify differences between a process model and the real process as recorded in the event log. The differences identified may lead to changes of the model or process. For example, exposing deviations between the model and process may lead to better work instructions or changes in management. Conformance checking is also a useful tool for auditors that need to make sure that processes are executed within the boundaries set by various stakeholders.

In this section, we show that conformance checking can be used for other purposes such as repairing models and evaluating process discovery algorithms. Moreover, through conformance checking event logs get connected to process models and thus provide a basis for all kinds of analysis.

8.5.1 Repairing Models

When a process model and an event log "disagree" on the process, this should lead to adaptations of the model or the process itself. Let us assume that we want to use conformance checking to *repair the model*, i.e., to align it with reality. The diagnostics provided in Figs. 8.6 and 8.7 can be used to (semi-)automatically repair the model. For instance, paths that are never taken can be removed from the model. Note that Figs. 8.6 and 8.7 show the frequency of activities and their causal dependencies. This may lead to the removal of activities that are never (or seldom) executed or the removal of choices. Token replay does not help to remove concurrency that is never used (e.g. activities modeled in parallel but executed in sequence). However, this can be seen in the footprint matrix. After removing unused parts of the model, the m and r-tags pointing to missing and remaining tokens can be used to repair the model. An m-tag points out activities that happened in the process but that were not possible according to the model. An r-tag points out activities that did not happen but that were supposed to happen according to the model. Comparing the footprint matrices of the log and model will show similar problems. Such information can be used by a designer to repair the model. In principle, it is possible to do this automatically. For example, given a set of edit operations on the model one could look for the model that is "closest" to the original model but that has a fitness of, say, more than 0.9. It is fairly straightforward to develop a genetic algorithm that minimizes the edit distance while ensuring a minimal fitness level: edit operations to repair a model are closely related to the genetic operations (mutation and crossover) described in Sect. 7.3. See [52] for a concrete approach to repair process models using event data.

8.5.2 Evaluating Process Discovery Algorithms

The focus of this chapter has been on conformance checking and quantifying fitness, i.e., measuring the ability to replay observed behavior on a predefined process model. However, fitness is just one of four quality dimensions and conformance checking is also related to the evaluation of process discovery techniques. Therefore, we provide a few pointers to literature.

In Sect. 6.4, we discussed the challenges that process discovery algorithms are facing: incompleteness, noise, etc. Process discovery is a complex task and many algorithms have been proposed in literature. As discussed in [122], it is not easy to compare the different algorithms. Compared to classical data mining challenges, there seem to be much more dimensions both in terms of *representation* (see, for instance, the more than 40 control-flow patterns gathered in the context of the Workflow Patterns Initiative [155, 191]) and *quality criteria* (even for the fitness notion several definitions exist). In [43], several process discovery techniques are evaluated using real-life event logs and multiple criteria. The study does not include recent approaches like inductive mining, but shows that automated comprehensive evaluations are possible.

In Sect. 6.4.3, we described four quality dimensions: *fitness*, *simplicity*, *precision*, and *generalization*. Obviously, conformance checking is closely related to measuring the fitness of a discovered model. Whether the model used for conformance checking is made by hand or discovered using some process mining algorithm is irrelevant for the techniques presented in this chapter. Hence, conformance checking, as described in this chapter, can also be used to evaluate and compare process discovery algorithms. However, the "flower model" N_4 in Fig. 8.2 illustrates that fitness covers just one dimension. Simplicity, precision, and generalization also need to be considered when evaluating a discovered model. Leaving out one dimension may lead to degenerate models as shown in [26, 169].

Obviously, a process discovery algorithm should aim to generate the *simplest model* possible that is able to explain the observed behavior. See [101] for an overview of metrics used to quantify the complexity and understandability of a process model. The metrics consider aspects such as the size of the model (e.g., the number of nodes and/or arcs) and the "structuredness" or "homogeneity" of the model [101].

A model with a severe lack of *precision* is "underfitting" and will, on average, have too many enabled transitions during replay. See [5, 26, 105, 121, 169] for approaches to qualify precision.

Precision: Avoid underfitting
Precision can be quantified in different ways. Here, we sketch the approach used in [105, 169]. Let $E \subseteq \mathcal{E}$ be the set of events in some event log L. M is the corresponding process model having a set of states S. Let A be the set of activities and assume the default classifier $\underline{e} = \#_{activity}(e) \in A$ is the *activity* associated to event $e \in E$ (see Sect. 5.2).

Assume the log has been aligned and was "squeezed" into model M using alignments (see Sect. 8.3). Hence, without loss of generality we may assume that each event e fits into the model and that we are able to compute $\#_{state}(e) \in S$. This is the *state* just *before* the occurrence of event e. Using optimal alignments, events (log moves) are synchronized with model moves, yielding a deterministic mapping from events to model states.

Let $en_M(e) \subseteq A$ be the set of activities enabled in the model in $\#_{state}(e)$. Let $\#_{hist}(e) \in A^*$ be the history of e, i.e., the sequence of activities executed for the same case until e. $\#_{hist}(e)$ does not include the latest activity corresponding to e, but the sequence of activities leading to e. $en_L(e) = \{\#_{activity}(e') \mid e' \in E \wedge \#_{hist}(e') = \#_{hist}(e)\} \subseteq A$ is the set of activities that were executed by events having the same history. We assume that events with the same history are mapped onto the same state, i.e., $\#_{hist}(e') = \#_{hist}(e)$ implies $\#_{state}(e') = \#_{state}(e)$. This is the case for most process modeling notations (BPMN, Petri nets, UML activity diagrams, etc.).

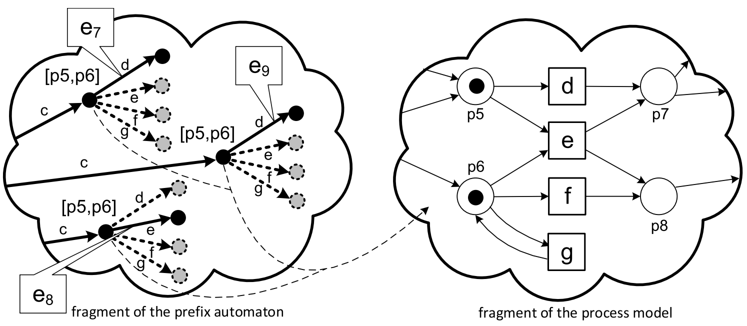
If precision is high, the model does not allow for much more behavior than observed. Hence, $|en_M(e)| \approx |en_L(e)|$. If precision is low, the model allows for much more behavior than observed. Hence, $|en_M(e)| \gg |en_L(e)|$. Precision can now be defined as follows:

$$precision(L, M) = \frac{1}{|E|} \sum_{e \in E} \frac{|en_L(e)|}{|en_M(e)|}$$

By definition, $en_L(e) \subseteq en_M(e)$ because the event log is perfectly fitting. Therefore, $0 < precision(L, M) \leq 1$ (assuming $E \neq \emptyset$). If all behavior allowed by the model is actually observed, then $precision(L, M) = 1$. If the model allows for much more behavior than observed, then $precision(L, M) \ll 1$. By taking the average over all events, we automatically take frequencies into account. If the model has an activity that is enabled on a frequent path but the activity is never executed, then this is more severe than an unused activity enabled along an infrequent path.

Figure 8.13 illustrates the precision computation. Three events (e_7, e_8, and e_9) share the same history and therefore also occur in the same state ($[p5, p5]$). In this state, four activities are possible (d, e, f, and g), but only two occur in the event log, d (events e_7 and e_9) and e (event e_8). There are no events having the same history corresponding to the occurrence of f or g. Therefore, $en_L(e_7) = en_L(e_8) = en_L(e_9) = \{d, e\}$ and $en_M(e_7) = en_M(e_8) = en_M(e_9) = \{d, e, f, g\}$. If the scope is limited to the three events in Fig. 8.13, then $precision(L, M) = \frac{1}{3}(\frac{2}{4} + \frac{2}{4} + \frac{2}{4}) = 0.5$. Of course, we would need to consider all events to compute the overall precision.

To illustrate the precision metric, consider the two perfectly fitting models in Fig. 8.2 (N_1 and N_4). For the other two models, alignment computations are needed to first "squeeze" the observed behavior into the model [5]. Using



e7
e9
e8
[p5,p6]
[p5,p6]
[p5,p6]
c
d
e
f
g
p5
p6
p7
p8
fragment of the prefix automaton
fragment of the process model

Fig. 8.13 Computing precision: $\#_{state}(e_7) = \#_{state}(e_8) = \#_{state}(e_9) = [p5, p6]$, $en_M(e_7) = en_M(e_8) = en_M(e_9) = \{d, e, f, g\}$, and $en_L(e_7) = en_L(e_8) = en_L(e_9) = \{d, e\}$

ProM, we find $precision(L, N_1) = 0.955$ and $precision(L, N_4) = 0.304$. This matches our intuition, N_1 is much more precise than the "flower model" N_4 that is indeed severely underfitting.

The precision computation just sketched can be applied to *any* type of process model for which optimal alignments can be computed (i.e., not just Petri nets). Using function $\#_{hist}$ to group events, we are implicitly creating a so-called "prefix automaton" (cf. Fig. 8.13). However, other abstractions (next to $\#_{hist}$) are possible as discussed in [2, 5, 169].

In general, a process model should not restrict behavior to just the examples seen in the log. A model that does not generalize is "overfitting": Future observations are likely to deviate from the model. It is difficult to reason about generalization because this refers to unseen examples.

When evaluating a process discovery *algorithm* and not a specific model, one can use *cross-validation* (e.g., k-fold cross-validation or leave-one-out cross-validation) (see Sect. 6.4.2.3). For cross-validation first a process model is learned for a selected part of the event log (e.g., 80% of the cases), called the *training log*. The remaining 20% of the cases forms the *test log*. Then the test log is replayed or aligned using the model learned for the training log. Such a test can be repeated k times when k-folds are used. If the average fitness for the different test logs is good, the discovery technique is able to generalize. If the average fitness is poor, then the discovery technique is clearly overfitting. In the latter case, the discovery technique produces models that are unable to explain future observations.

Cross-validation *cannot* be used to evaluate a *specific* process model (see Sect. 6.4.2.3). When the model is already given, there is no point in creating a test and training log. In such situations, we need to resort to simple frequency-based metrics such as the one presented in [169]. Every event can be seen as an observation of an activity in some state $s \in S$. Suppose that state s is visited n times and that w is the number of different activities observed in this state. Suppose that n

is very large (say 985 visits to the same state) and w is very small (say 3 unique activities observed in the state), then it is unlikely that a new event visiting this state will correspond to an activity not seen before in this state. However, if n and w are of the same order of magnitude, then it is more likely that a new event visiting state s will correspond to an activity not seen before in this state. An estimator can be derived under the Bayesian assumption that there is an unknown number of possible activities in state s and that the probability distribution over these activities follows a multinomial distribution. It estimates the probability that a new observation will reveal a path not seen before. The weight of each state is based on the number of visits. The computed *generalization value* in [169] is close to 0 if it is likely that new events will exhibit behavior not seen before. The computed generalization value is close to 1 if it is unlikely that the next event will reveal new behavior.

Another approach to compute precision and generalization is to *project* process model and event log on smaller sets of activities and compare their behaviors with respect to these activities only. This can be viewed as a generalization of comparing footprints using $k \geq 1$ dimensions and not limited to the "directly follows" relation. Language inclusion on the projected models and logs can be used to compute precision and recall metrics.

Also see the precision and recall metrics in [14] used to compare two models in the context of an event log.

The examples and pointers in this section show that many approaches are available to quantify the four quality dimensions introduced in Sect. 6.4.3. These can be used to objectively assess the quality of process discovery results.

8.5.3 Connecting Event Log and Process Model

While replaying the event log on the process model, events in the log are related to activities in the model, i.e., the event log is *connected* to the process model. This may seem insignificant at first sight. However, this is of crucial importance for the subsequent chapters. *By relating events to activities, information extracted from the log can be used to enrich the process.* For example, timestamps in the event log can be used to make statements about the duration of some modeled activity.

Figure 8.14 shows the class model presented in Sect. 5.4.1 (cf. Fig. 5.9) without case attributes. The process model level and the event level may *exist independent of one another*. People make process models without relating them to (raw) data in the information system and processes generate data while being unaware of the process models that may exist. Notable exceptions are WFM and BPM systems for which such a connection already exists. This is why process-aware systems are not just important for process automation and also serve as a powerful enabler for process analysis. However, for the majority of processes, there is no supporting WFM or BPM system. As a result, process models (if they exist) and event data are at best loosely coupled. Fortunately, both token replay and alignments *can be used to establish a tight coupling between the process model level and the event level.*

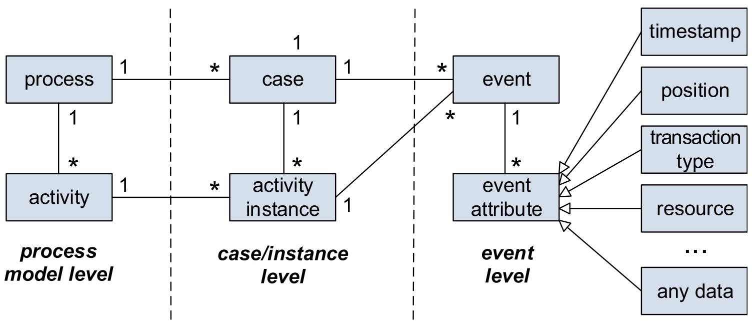


Fig. 8.14 Observed behavior (events) is related to modeled behavior (activities)

The case/instance level shown in Fig. 8.14 consists of *cases* and *activity instances* that connect *processes* and *activities* in the model to *events* in the event log. Within the same case (i.e., process instance) there may be multiple instances of the same activity. For instance, some check activity may be performed multiple times for the same customer request (cf. loops).

Typical event data exist in the form of collections of data records possibly distributed over multiple database tables. A record in such a table may correspond to one or more events while listing certain properties (attributes), e.g., date information, some amount, and credit rating. As discussed in Chap. 5, one of the main challenges is to locate these events and to correlate them. Each event needs to be related to a particular case. When replaying the event log on a model, each event that "fits into the model" is connected to an activity instance. Note that there can be multiple instances of the same activity for each case. Moreover, a single activity instance may correspond to multiple events. Consider a case c with a loop involving activity a. Two instances of a are executed for c, $a_{c,1}$ and $a_{c,2}$. For each of these two activity instances there may be multiple events. For example, the first activity is offered, started, and aborted (three events corresponding to $a_{c,1}$) whereas the second activity is assigned, started, suspended, resumed, and completed (five events corresponding to $a_{c,2}$). See also Sect. 5.2 where the transactional life-cycle is discussed in detail. When describing token replay and alignment computations we did not elaborate on the different *event types* (start, complete, abort, etc.). However, such transactional information can be taken into account when replaying the event log.

In Sect. 8.2, we showed that *token-based replay* can be used to relate observed behavior to modeled behavior. Section 8.3 introduced the notion of *alignments* as an even more direct way of relating observed behavior and modeled behavior. Both approaches can be used to detect and diagnose deviations as sketched in Fig. 8.15. Moreover, alignments can also be used to "squeeze" reality into the model for further analysis. Even if a case does not fit completely, we can find a corresponding path in the model. If we leave out non-fitting cases, the remaining set of cases is no longer representative for the whole. Therefore, it is important to "squeeze" reality into the model even when there are (minor) discrepancies. Subsequently, event data can be used to "breathe life" into otherwise static process models. As a result, all

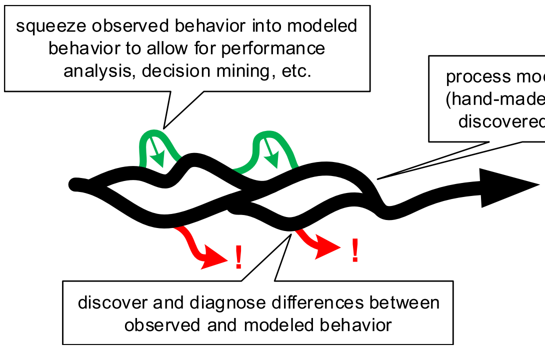

squeeze observed behavior into modeled behavior to allow for performance analysis, decision mining, etc.
process model (hand-made or discovered)
!
!
discover and diagnose differences between observed and modeled behavior

Fig. 8.15 The ability to replay event data on a process model helps to detect and diagnose deviations and to "squeeze" reality into the model for further analysis

kinds of information extracted from the event log can be projected onto the model, e.g., showing bottlenecks and highlighting frequent paths. The event attributes in Fig. 8.14 provide valuable information that can be aggregated and mapped onto activities and resources. For instance, timestamps can be used to visualize bottlenecks, waiting times, etc. Resource data attached to events can be used to learn working patterns and allocation rules. Cost information can be projected onto process models to see inefficiencies. The next chapter will elaborate on this.